An Exploration of Circuit Similarity for Discovering and Predicting Reusable Hardware

Kevin Zeng

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Peter M. Athanas, Chair
Michael S. Hsiao
Cameron Patterson
Harold E. Trease
Chao Wang

March 24, 2016
Blacksburg, Virginia

Keywords: Productivity, Reuse, Hardware, FPGA, Similarity Matching, Birthmarking,

An Exploration of Circuit Similarity for Discovering and Predicting Reusable Hardware

Kevin Zeng

(ABSTRACT)

A modular reuse-based design methodology has been one of the most important factors in improving hardware design productivity. Traditionally, reuse involves manually searching through repositories for existing components. This search can be tedious and often unfruitful. In order to enhance 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. To achieve this goal, methods for assessing the similarity of two designs are necessary. Different techniques for comparing the similarity of circuits are explored utilizing concepts from different domains. A new similarity measure was developed using birthmarks that allows for fast and efficient comparison of large and complex designs. Applications where circuit similarity matching can be utilized are examined such as IP theft detection and reverse engineering. Productivity experiments show that automatically suggesting reusable designs to the user could potentially increase productivity by more than 34% on average.

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-1266245.

An Exploration of Circuit Similarity for Discovering and Predicting Reusable Hardware

Kevin Zeng

(GENERAL AUDIENCE ABSTRACT)

The reuse of existing designs has been a common method to increase the time-to-market when designing circuits. With the number of existing designs increasing, there needs to be a way to efficiently find and retrieve circuits of interest. Traditionally, retrieving designs involves searching through different databases which can often be difficult and time consuming. Here, a design assistant tool is proposed where the circuit under design is monitored. Snapshots of the circuit are continuously taken and compared against a library of existing designs such that the most similar designs are suggested to the user for reuse. In order to suggest circuits, the tool has to be intelligent enough to accurately and efficiently compare how similar two designs are. Different approaches to calculating similarity is investigated in order to understand the different tradeoffs. Experiments conducted show that having a design assistant can greatly improve the productivity of the designer.

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-1266245.

Dedication

This work and its entirety is dedicated to my grandmother.

Acknowledgments

This dissertation was possible due to the support of many who have impacted me, motivated me, and consoled me during this journey.

To my professors who have guided me through not only my doctorate degree, but my undergraduate and master's as well. To Dr. Abbott who gave me guidance early on and instilled the potential of a Ph.D. I'm thankful for the classes with my committee members Dr. Wang, Dr. Hsiao, and Dr. Patterson that have taught me so much. I thank you Harold for your friendship and guidance you have given me. This would all have not been possible if it wasn't for my advisor, Peter Athanas. I thank you for not only suggesting a radically difficult problem, but seeing the potential and teaching me what research is all about even when the finish line seemed near impossible.

To my fellow classmates and colleagues Krzyztof, Ritesh, David, Shaver, Tony, Andrew and Rama for helping me during the start of my journey. Many thanks to Shenghou, Ryan, Jehandad, Muhammed, Jacob, and Ali who have been there for me during the final stretch. Thanks Ali for the countless weekends in the lab, with the occasional 2 hour bike ride in the middle.

I want to thank Brian, Andy, Kenney, George, Dan, Zihan, and everyone that has been there for me. Thank you Jenny for believing in me till the end. Thank you Jimmy for taking the time to not only be my personal Uber, but for giving me a place to stay during the final days before my defense. Lastly, no words can describe the amount of support and encouragement by my family, my dad and my mom. I hope that this work becomes an inspiration to my two younger sisters, Linda and Tina, as they have continued to become my source of inspiration and motivation when all seemed lost. Thank you all.

Contents

1 Introduction **1**

1.1 The Hardware Productivity Gap Problem 1

1.2 Motivation 2

1.2.1 Reuse-based Design Productivity 2

1.3 Problems with Design Reuse 3

1.4 Proposed Approach 4

1.4.1 New Reuse-Based Approach 4

1.4.2 Circuit similarity matching 6

1.5 Extended Applications 7

1.6 Contributions 8

1.7 Dissertation Organization 9

2 Background and Related Work **11**

2.1 Reuse-based Design Productivity 11

2.1.1 Hardware Design Productivity 11

2.1.2 Designing Reusable Components 12

2.1.3 Integration of IP 13

2.1.4 IP Libraries and Retrieval 14

2.1.5 Reuse Summary 18

2.2 Circuit Matching 18

2.2.1 *Exact* Circuit Matching 19

2.2.2 *Similarity* Circuit Matching 21
2.2.3 Circuit Matching Summary 21
2.3 Leveraging Research From Other Domains 22
2.3.1 Molecular similarity matching 22
2.3.2 Machine Learning 22
2.3.3 Software similarity matching 23
2.4 Summary 23

3 Graph-based Approaches **25**
3.1 System Overview 26
3.2 Isomorphism 26
3.2.1 Subgraph Isomorphism 26
3.3 Decomposition Subgraph Isomorphism 28
3.3.1 Similarity Metric 30
3.4 Maximum Common Subgraph 30
3.4.1 Similarity Metric 31
3.5 Implementation 31
3.5.1 Design Entry 32
3.5.2 Displaying Results 33
3.6 Results and Analysis 34
3.6.1 Benchmark 34
3.6.2 Accuracy 34
3.6.3 Performance 37
3.6.4 Scalability 41
3.6.5 Limitations 42
3.7 Summary 42

4 Molecular Similarity Approach **43**
4.1 Fingerprinting 43

4.2 System Overview 44
4.2.1 And Inverter Graphs 45
4.2.2 Bitslice Database and Fingerprint Definition 46
4.2.3 Cut Enumeration 47
4.2.4 Boolean Matching 47
4.2.5 Bitslice Aggregation 48
4.3 Similarity 49
4.4 Implementation 49
4.4.1 Converting design to AIG 49
4.4.2 Design Entry 49
4.5 Result and Analysis 51
4.5.1 Benchmark 51
4.5.2 Accuracy 52
4.5.3 Performance 54
4.5.4 Limitations 55
4.6 Summary 56

5 Hardware Birthmarking Approach **57**
5.1 Problems with Netlists 57
5.2 Software Birthmarks 58
5.3 Hardware Birthmark 58
5.3.1 System Overview 59
5.4 Datapath Hardware Birthmark 59
5.4.1 Functional Component 60
5.4.2 Structural Component 61
5.4.3 Constant Component 61
5.5 Computing Similarity Between Birthmarks 63
5.5.1 Functional Similarity 63
5.5.2 Structural and Constant Similarity 63

5.5.3 Similarity Score . . . 64
5.6 Implementation . . . 64
5.6.1 Front End Interface . . . 65
5.6.2 Extracting Birthmarks . . . 65
5.6.3 Similarity Calculation . . . 66
5.6.4 Comparing Designs . . . 66
5.7 Result and Analysis . . . 66
5.7.1 Performance . . . 67
5.7.2 Identifying Similar Circuits . . . 69
5.7.3 Limitations . . . 72
5.8 Summary . . . 72

6 Q-gram Birthmarking Approach **74**
6.1 Q-grams . . . 75
6.1.1 Graph Similarity Search Utilizing Q-grams . . . 75
6.2 Path-Based q-grams . . . 76
6.2.1 Path-Based q-gram Schemes . . . 76
6.3 Determining q . . . 77
6.4 Extension to *(q to 1)*-gram . . . 79
6.5 Predicting Reusable Subcomponents . . . 79
6.6 Calculating Similarity . . . 80
6.7 Implementation . . . 81
6.7.1 Q-gram Extraction . . . 81
6.8 Result and Analysis . . . 81
6.8.1 Finding Q . . . 82
6.8.2 Performance . . . 85
6.8.3 Birthmark Extraction . . . 85
6.8.4 Identifying similar circuits . . . 86
6.8.5 Circuit to database search . . . 86

6.9 Limitations . . . 89
6.10 Summary . . . 90

7 Case Studies **92**
7.1 Functional Block Prediction . . . 92
7.1.1 Prediction Model . . . 92
7.1.2 Evaluation of Model Using Cross Validation . . . 93
7.1.3 Experiment . . . 95
7.2 Soft-IP Theft and Reverse Engineering . . . 95
7.2.1 Claiming Theft . . . 96
7.2.2 Detection at the Physical Layer . . . 96
7.2.3 Credibility . . . 97
7.2.4 Reverse Engineering . . . 97
7.3 Productivity . . . 98
7.3.1 Design Environment . . . 98
7.3.2 Design Problem . . . 99
7.3.3 Results . . . 101
7.3.4 User Feedback and Future Work . . . 102
7.3.5 Discussion . . . 103

8 Conclusion **104**
8.1 Contributions . . . 105
8.2 Future Works . . . 106

A Appendix A: Top 10 results for 5 different circuits with varying q **107**

B Appendix B: Top 10 results for 5 different circuits with varying q using Frequency-based scheme **112**

Bibliography **117**

List of Figures

1.1 Comparison between the proposed flow and the traditional flow for reusing designs. The entire search process is completely automated allowing the designer to remain in the design scope. 5

1.2 Three structurally different implementations of a 2-input XOR gate. 6

1.3 Different case models of the proposed birthmarking approach for detecting theft. 7

2.1 Query interface used by [1]. 15

3.1 High level overview of graph based approach. 25

3.2 Decomposition subgraph matching example with a 1-bit counter. 29

3.3 Two circuits where (c) is a common sub-circuit of (a) and (b). 30

3.4 Implemented system overview. 32

3.5 Implemented reuse flow into an existing design environment, utilizing Xilinx ISE as the front end. 33

3.6 Execution time of FSI and VF2 for varying pattern circuit sizes. 38

3.7 Execution time MCS with varying input and pattern circuit sizes. 38

3.8 Execution time of Decomposer for different database sizes. 39

3.9 Execution time for varying database sizes. 40

3.10 Execution time of Matchers for varying input circuit sizes for a database of 13 Circuits. 41

4.1 High-level overview of molecular similarity approach. 44

4.2 Conversion of a 2-input XOR gate to an AIG. 45

4.3 Removing feedback loops from sequential circuits. 46

4.4 4-cut enumeration on Node A. There are no other possible 4-cut on Node A. 47
4.5 Truth table extraction and storage for a given cut. 47
4.6 Aggregating three 2-input AND gate to make a 4-input AND gate. 48
4.7 High-level overview of the implemented system. 50
4.8 Implemented system into an existing design environment, leveraging LabVIEW front-end with a simple interface on the side to display the results. . . 51
4.9 Similarity measurements comparing a b14 design against several other IWLS benchmarks. 52
4.10 Similarity measurements comparing a b15 design against several other IWLS benchmarks. 52
4.11 Heat map showing the similarity results of three sets of designs. 53
4.12 Execution time in extracting the fingerprint from a netlist. 54

5.1 System overview utilizing a 3 component birthmarking scheme, capturing functional, structural, and constant data. 59
5.2 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. 60
5.3 The figure shows how the fingerprinting process works. Each position in the vector indicates the number of occurrence of a specific subcomponent. 61
5.4 Extraction of the constant component. The figure also shows the basic layout of the binning approach. 62
5.5 Breakdown of occurrences of constants in 151 Verilog files. 62
5.6 High-level system implementation of the reuse flow using birthmarks. 64
5.7 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 DFG. 67
5.8 Performance of comparing each circuit in the database to the database. Performance is dependent on the varying sequence length of the functional component. 69
5.9 Heatmap extracted by autocorrelating the circuits in the database. The dark regions indicates dissimilarity and the bright regions indicates similarity. . . 71

6.1 System overview leveraging q-grams as the functional component of the birthmarking approach. 74

6.2 Path-based 3-gram model of basic 8-bit counter. (Q to 1)-gram is shown because it includes all the q-grams that are less than and equal to 3. 76

6.3 Heatmap of showing the difference in results when using different values of q. 77

6.4 High-level system implementation of the reuse flow using the q-gram birthmarking approach. 82

6.5 Run time of q-gram token extraction with varying q. 83

6.6 Mean average precision varying q. Path-based and frequency-based schemes were explored. 84

6.7 Precision varying q of three reference circuits. Frequency-based schemes was used. 84

6.8 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 DFG. ($q = 6$) 85

6.9 Performance of comparing each circuit in the database to the database for the datapath and q-gram birthmarks. Performance of the inverted indexing approach for q-gram birthmarks is compared as well. 88

6.10 Dataflow of FSMs using different state encodings. 90

7.1 The precision and applicability for different values of q obtained from cross validation of the circuit database. 93

7.2 The reference design of a Sobel operator. 94

7.3 Heatmap showing the auto-correlation between all 151 circuits. The green, yellow, and red signifies three levels of significance with green indicating that there is most likely theft and red indicating that there is little in common between the two. Blue depicts containment between two designs. 97

7.4 Design interface for case study. 98

7.5 The impact of the proposed reuse model on the overall design time of the user. 100

7.6 The impact of the proposed reuse model on the ideal design time of the user. 102

List of Tables

1.1 Cost of a typical reference design for FPGA [2] 3

3.1 Accuracy measurement of the three different matchers. The query passed in was a 1-bit counter using a Kogge-stone adder. 34

3.2 Comparison of results between FSI, VF2, and DSI. 36

3.3 Accuracy measurement of the three structurally different XOR gates using a MCS approach. 37

3.4 Accuracy measurement of the three structurally different XOR gates using DSI approach. 37

3.5 Accuracy measurement of the three structurally different XOR gates using a FSI approach. 37

3.6 Comparison of decomposition matcher using two different decompositions trees. 40

4.1 Brief Description of the ITC99 benchmarks used. 53

5.1 Ranking results for various modules of different types. 70

6.1 Detailed description of the different birthmark schemes. 77

6.2 Birthmarking statistics after processing 110 different circuits. 78

6.3 Ranking results using select datapaths as functional component. 87

6.4 Ranking results using q-gram birthmarks as functional component. 87

7.1 Functional block prediction at different time steps. 94

7.2 Additional Reuse statistics. 100

A.1 $q = 2$ Path-Based Scheme. 108

A.2 $q = 3$ Path-Based Scheme. 108
A.3 $q = 4$ Path-Based Scheme. 109
A.4 $q = 5$ Path-Based Scheme. 109
A.5 $q = 6$ Path-Based Scheme. 110
A.6 $q = 7$ Path-Based Scheme. 110
A.7 $q = 8$ Path-Based Scheme. 111

B.1 $q = 2$ Frequency-Based Scheme. 113
B.2 $q = 3$ Frequency-Based Scheme. 113
B.3 $q = 4$ Frequency-Based Scheme. 114
B.4 $q = 5$ Frequency-Based Scheme. 114
B.5 $q = 6$ Frequency-Based Scheme. 115
B.6 $q = 7$ Frequency-Based Scheme. 115
B.7 $q = 8$ Frequency-Based Scheme. 116

Chapter 1

Introduction

The constant push for more transistors on a single chip has allowed designers to create larger hardware designs that are more complex than ever before. This push has advanced the limits of physical devices and has given designers the capabilities to create circuits of greater complexity. As a result, hardware platforms and accelerators such as field programmable gate arrays (FPGAs) are becoming more and more appealing for many applications requiring extensive computational resources.

FPGAs are integrated circuits with reprogrammable logic and interconnect. This makes them highly flexible and an ideal hardware accelerator due to its natural parallelism and ability to quickly process massive amounts of data. For example, Microsoft's Catapult system utilizes 1,632 servers with high-end Altera Stratix V FPGAs for accelerating data center workloads such as Bing searches [3] and convolutional neural networks [4]. Though graphics processing units (GPUs) are widely used as accelerators in numerous applications, the performance per watt metric of FPGAs makes them appealing for creating power efficient systems. This is especially important for data centers where power-hungry servers are processing massive amounts of data. With the rising interest of utilizing FPGAs for high performance computing (HPC) applications, Intel plans to integrate reconfigurable logic onto the same die as their server class Xeon CPUs [5]. Many of these efforts and advancements further expose FPGAs as a platform of interest for accelerating computation and opens new doors for potential applications and research.

1.1 The Hardware Productivity Gap Problem

Similar to CPUs, FPGAs have followed the trend depicted by Moore's Law, doubling in logic capacity on-chip every two years [6]. As the number of available hardware resources increase on a FPGA, larger and more complex designs can now be realized. On the other hand, a significant limitation is the use and design of applications for not only FPGAs, but hardware

design in general. Hardware design is far more complicated than software engineering [7]. Experience and knowledge of the underlying hardware is essential to not only design a circuit that is fast and efficient, but also to be able to debug and verify functional correctness. Lack of an open-source community also limits the FPGA platforms to specialized areas. In addition, the compilation times can appear seemingly endless resulting in a few turns per day [8]. Turns is defined as the total number of design iterations performed, which typically is limited to about one to three turns a day, depending on the size of the design [9]. This cycle start from design, to synthesis, placement, routing, programming the FPGA, and finally debug and verification. This problem is true for not only FPGA design, but ASIC and hardware design in general.

As hardware complexity grows, design tools need to be able to keep pace and assist the user in designing a system that leverages the full potential of the device. Yet, current tools are unable to provide the necessary capabilities for the designer to maintain pace at which silicon density is progressing. As a result, complex designs consume a substantial amount of engineering hours before completion. The disparity between the silicon density and the capabilities of design tools is commonly known as the productivity gap. In order to address the expanding gap between the two, many hardware designers have turned to a modular design methodology that leverages the concepts of reuse.

1.2 Motivation

With a modular design methodology, hardware design becomes similar to system level design. Engineers create new circuits using pre-designed modules. The shift to a modular design methodology poses several new challenges in terms of furthering the productivity of the designer.

1.2.1 Reuse-based Design Productivity

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, it is unnecessary to reinvent the components again. Time spent during the redesign can be allocated to more important tasks of the design phase. Reuse in domain-specific environments seems to be more successful than in the general case since the design variants are small and manageable within the mind of an experienced designer. In the general case, where the number of design variants are in the thousands or more, even the most experienced designer may not have a good handle on the ever-changing constituents of the pool of available intellectual property (IP). The term IP here refers to hardware designs that are packaged and readily available for designers to reuse.

Table 1.1: Cost of a typical reference design for FPGA [2]

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

Intellectual Properties (IP)

IP cores come in a wide variety of formats and can be classified into one of three categories: hard, firm, or soft. Hard-IP represents a design at the physical layer that has been fully placed and routed to a fixed technology. This makes hard-IP less adaptable for reuse because many of the constraints are set for a target technology and would require a deep understanding of the design in order to induce changes. Firm-IPs are reusable designs that are synthesized to a particular cell library. Lastly, soft-IPs are the most versatile, usually given in a high description language (HDL). This allows the user to easily modify, reuse, and adjust the IP to satisfy the required specifications for different technologies and platforms. Each form of IP have their advantages and disadvantages, though, in most cases, soft-IPs are generally preferred over the three due to the flexibility to change and adapt the IP into existing designs.

The very reason software libraries exist and have been overly successful is that they provide the user with a set of commonly used functions that can be imported and used immediately. The overall productivity of the designer is increased while design time is decreased. It is assumed here that the libraries of available IP are valid and fully tested. Due to the underlying assumption of functional correctness for reusable designs, most of the simulation and verification for the IP is eliminated. All that is left is the integration of the component into the overall design. Table 1.1 shows the overall cost breakdown of a typical reference design for military applications [2]. The simulation, verification, and implementation phase of a design takes up the largest portion of the design and this is where 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.

1.3 Problems with Design Reuse

In spite of the benefits that reuse offers, it 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 reusable components hinder its full potential. Vendors have come a long way in terms of increasing productivity by providing powerful tools for synthesis, debug and simulation, and placement and routing [2, 10]. 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, enabling software designers to program and use FPGAs. Nonetheless, these tools have done little to incorporate design reuse into the design methodology. Many tools allow the import of third-party IP cores through which they can be searched, but that is the extent of reuse seen in most vendor tools.

Though the contemporary design environments simplify reuse to an extent, 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 fit the requirements of the design as well as the necessary documentation behind the IP. Common reusable hardware repositories themselves are not as prevalent either, making the search for relevant IPs difficult. Furthermore, as the number of IPs increase, the amount of data the user needs to analyze and process becomes unmanageable. For example, at the time of this dissertation, a search on GitHub for projects containing Verilog and VHDL HDL exceeds 10,000, and will continually increase as new designs emerge. This does not include other code management hosts such as BitBucket, SourceForge, etc. Many of these repositories can often be incomplete and disorganized. A complete and private library within an organization can also have certain limitations as users need to familiarize themselves with what the database contains.

1.4 Proposed Approach

As FPGAs gain a wider audience resulting in a growing hardware community, more and more designs will be become available. With an extensive collection of reusable designs scattered across different repositories, there requires a need to efficiently and quickly find relevant designs. If there exists a tool where the candidate IPs are automatically suggested to the user throughout the design phase, there is less effort of reuse required from designers. By extension, if the above assertion is true, the *design* process is transformed into a *discovery* process.

1.4.1 New Reuse-Based Approach

Figure 1.1 compares the proposed reuse model against the traditional reuse paradigm. As the user makes changes to a reference circuit, the proposed tool will continuously monitor the reference during the entry process. A vast number of comparisons are performed between the emerging design and an archive of existing designs. 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 the 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 reuse. Automatic integration of the

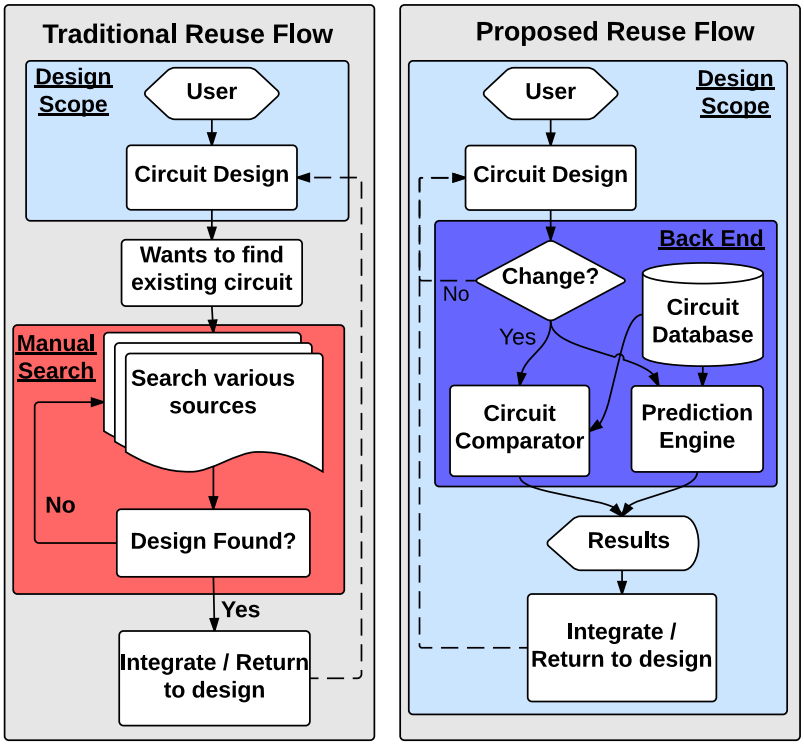


Figure 1.1: Comparison between the proposed flow and the traditional flow for reusing designs. The entire search process is completely automated allowing the designer to remain in the design scope.

existing design to the current can be a potential feature as well.

Traditional reuse flow requires the user to manually search for required designs which can be very time consuming and often leads to many dead-ends. Searching requires the user to know what they want; however, at times, even the designer may not necessarily know what they want or search for the right circuit. For example, when building a network on chip (NOC), routers, switches, allocators are used and are typical components that the user may search for. Yet, in that mindset, the possible existence of a connected and functional NOC does not occur to the designer.

The proposed flow takes away the manual search that has to be performed and suggests reusable designs in order to increase design awareness of the user. 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, design reuse becomes more appealing, requiring little to no effort from the user.

Consider the following example:

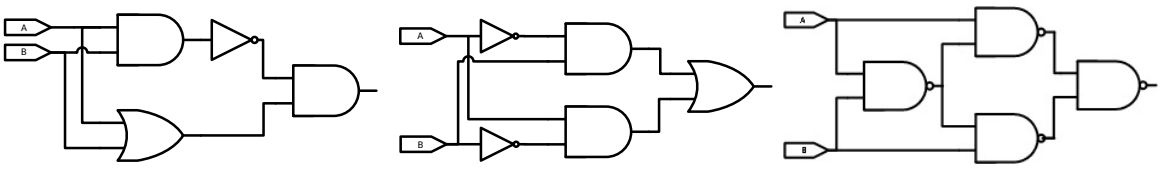
> *Linda is designing a simple Sobel edge detector. She searches for a reusable design in a database. No results are returned and so she assumes that that no such design exists. Linda now has to implement the design herself. Tina is a design expert and knows where a lot of existing designs are located. As Linda starts to design her circuit, Tina comes over to watch the design entry. Tina says to Linda, "I see what you have designed so far, are you trying to build a Sobel filter? There's this design in the database that you can take and reuse if you would like." Linda takes the design that Tina has and with some modification, finishes her project.*

Similar to having a personal assistant, the tool continuously monitors the design with no intervention from the user. Without the suggestion of Tina, Linda would have had to finish, simulate, and verify the circuit. By having Tina around, Linda's productivity was increased. Therefore, by having designs suggested to the user automatically, design reuse and the productivity of the user is enhanced.

1.4.2 Circuit similarity matching

In order to suggest potential reusable components, comparisons between two circuit designs need to be performed. Yet, the challenge is not determining if two designs are an exact match, but to derive a metric that determines how similar two designs are. Much of the research for comparing circuits is to find exact matches, typically by using graph-based approaches using information from the circuit netlist [11, 12]; however, these methods usually only consider the structural aspects of the circuits and neglect the functional aspects. Structurally different circuits can be can exhibit the same functionality depending on the primitive components used and the relationships between them. For example, Figure 1.2 shows three different implementations of a simple 2-input XOR gate using primitive logic. At the same time, functional matching has been focused on reverse engineering and verification and looks for exact matches as well.

The idea of similarity is extremely fuzzy. The concept of what is considered similar may vary



(a) (b) (c)

Figure 1.2: Three structurally different implementations of a 2-input XOR gate.

between different people. Not one approach is necessarily better than another and typically involves tradeoffs between accuracy and performance. In order to understand the limitations and define an approach for efficiently and accurately assessing the similarity of circuits, various concepts are explored. Before realizing the proposed method, ideas from includes graph theory, fingerprinting, and birthmarking are examined. A new methodology for comparing the similarity of two circuits using a q-gram *birthmarking* approach is proposed. A *birthmark* is defined by the inherent characteristics of a circuit and is constructed such that the structural and functional information of the circuit is captured. Structural information includes the components used and the overall layout of the circuit and functional information pertains to the dataflow of the circuit. Further details about the different approaches are elaborated on in chapters 3, 4, 5, and 6.

1.5 Extended Applications

Having methods to compare the similarity of circuit can be applied to a number of different applications.

1. **Soft IP Theft:** The flexibility of soft-IPs allow for them to be easily reused and integrated; however, as the number of reusable designs increase, soft-IPs have become

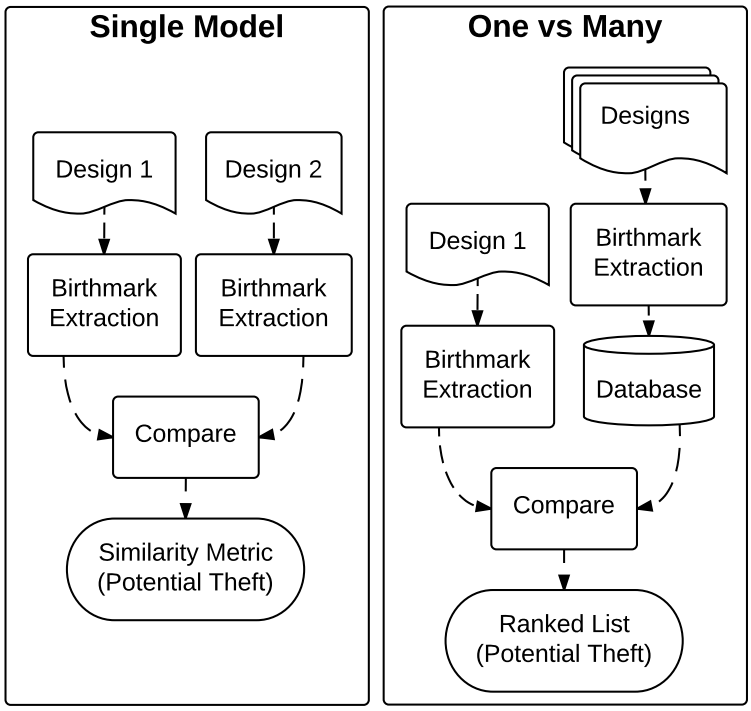


Figure 1.3: Different case models of the proposed birthmarking approach for detecting theft.

more susceptible to theft. With the ability to determine the similarity of circuits, there are several different use cases for determining theft, as outlined in Figure 1.3. One use case is: given two designs, determine whether one of the designs is a potential copy of the other. The second use case is: given a design of interest and a list of possible circuits that are suspected to be a copy, determine the circuits that most closely resembles the design of interest in a ranked order.

2. **Reverse Engineering:** With the ability to detect containment, insight to a given design can be achieved. Given a database of interested or commonly used modules, circuit similarity methods can be used to see if designs in the library can be found within the query.

3. **Design Exploration:** Design exploration can be performed by finding circuits of similar nature. Different designs exhibiting similar functionality are desired for satisfying required constraints. Consider the following example:

   Ruth has a completed design. However, there is a timing failure in one of her modules. She needs a similar design that can perform at a higher frequency. By using the module as a query, similar designs are suggested to Ruth with a design that has been documented to meet the necessary timing requirements.

1.6 Contributions

Determining suitable existing hardware for reuse requires a system that is able to compare a design against a multitude of patterns in order to search for a similar match. The system would require a library or database to store and keep track of the existing circuits. The work presented in this dissertation is an exploration of different approaches to circuit similarity matching. The focus and direction will be directed towards designing a system that is able to suggest reusable designs to the user automatically. Additionally, the impact regarding proposed reuse methodology to design productivity is investigated. Several extended applications are examined as well.

The key contributions that this dissertation provides are outlined below:

1. A new reuse-based design methodology integrating reuse into the design environment by automatically suggesting reusable designs is introduced to provide a better reuse experience for improving the productivity of the designer.

2. In order to compare two designs, circuits need to be described by some representation such that the similarity can be assessed efficiently. Various representations were explored which include graphs, molecular fingerprinting, software birthmarking, and q-grams detailed further in chapters 3, 4, 5, and 6 respectively.

3. With the different circuit representations, there needs to be methods to evaluate the *similarity* of the two designs. Different similarity metrics were explored for each representation in order to better understand how to achieve the results desired.

4. This work brings into fruition the proposed reuse-based design methodology that utilizes a circuit comparator with q-gram birthmarks for design suggestion. In addition, the effect this tool has on the productivity of hardware designers is analyzed in a case study.

1.7 Dissertation Organization

The remainder of this dissertation is organized as follows.

- **Chapter 2** : Design reuse is made up of different elements that affects one another such as designing a component to be reusable, integrating it, and finding it. Chapter 2 discusses the issues and problems associated with design reuse as well as related works that have addressed these problems. In addition, this chapter provides an overview of existing research in circuit matching, including the proposed birthmarking technique.

- **Chapter 3** : Graph are an ideal representation for complex structures such as circuits. Chapter 3 starts the exploration of circuit similarity matching with investigating graph-based approaches. Implementation details, results, and analysis of using graph-based approaches for comparing circuits are presented here.

- **Chapter 4** : Chapter 4 approaches the circuit similarity problem using concepts from molecular similarity. Circuits and molecules share many similarities and the methods from one domain can be leveraged for another. Capturing functional properties of circuits using Boolean matching is also explored. The algorithms, implementation, and results are presented here as well.

- **Chapter 5** : A birthmarking approach leveraging ideas from software similarity matching is explored in chapter 5, building upon the previous two approaches. The contributed approach is elaborated and highlighted in this chapter along with implementation and result details.

- **Chapter 6** : Birthmarks can be represented in many different ways. Utilizing the same concept presented in chapter 5. Chapter 6 explores a different approach for suggesting reusable designs utilizing q-grams. This chapter is concluded with the implementation details and results.

- **Chapter 7** : In order to study how suggesting reusable designs affect the designer's productivity, a case study was performed. Chapter 7 outlines the case study that

was conducted and results obtained. Furthermore, by having the ability to assess the similarity of circuits, new doors for different applications are opened. This chapter explores different opportunities and problems that circuit similarity matching can help contribute.

- **Chapter 8** : Lastly, the conclusion of this dissertation and future research is discussed in chapter 8.

Chapter 2

Background and Related Work

In order to understand the different aspects of hardware reuse, this chapter surveys related works and discusses different aspect of hardware design reuse and the problems involved. This chapter first examines the impact reuse has on design productivity. The methodology presented in this dissertation relies upon the ability to assess how closely a query design matches existing circuits in a reuse library. This depends upon circuit matching techniques, which are then examined. There are many other domains in which inexact ranked matching is performed. Those are also examined in this chapter.

2.1 Reuse-based Design Productivity

Reuse, along with raising the abstraction level, has been one of the more notable factors in providing a significant increase in design productivity [9, 13, 14, 15]. Not only is reuse applicable in hardware design, but in software and engineering in general [16, 17]. Every time a piece of software is written, there are many standard libraries that are included to do the most basic operations. Printing a simple "Hello World" requires a multitude of function calls that reuse existing code to handle the write out to the standard output [9].

Many of these same approaches can be very applicable for hardware design. For example, system on chip (SoC) designs have switched to a modular design approach designing systems by integrating and selecting the components and peripherals required.

2.1.1 Hardware Design Productivity

Hardware design productivity has many facets. One aspect is design. In other words, how to describe the circuit such that complex designs can be realized quickly. Circuit design entry started with schematic drawings where each component is placed and wired by hand.

Synthesis tools allowed users to describe the design with behavioral code using HDL. HLS that use software-like languages such as OpenCL opens doors for even programmers to use FPGAs [18]. Another aspect is the enhancement of algorithms and new approaches to reducing the overall compilation time typically associated with synthesis, placement, and routing such that the circuit is not only optimal but meets the constraints placed by the user [19]. Simulation tools and the ability to find and isolate bugs and verify functionality is important as well. One of the more time consuming sections of the design phase is in debug and verification, commonly taking around 70% of the entire design time [15].

Design, compilation, and verification are all standard procedures that comes with hardware design. It is a very cyclic process. Design reuse helps reduce the number of times the designers enter this cycle. Improvements in design and verification times are realized because components do not have to be redesign. These IP cores have been verified and proven to work before in the past [20]. Additionally, since the designer is not redesigning and verifying specific components of the circuit, the overall turns per day is reduced as well.

Though reuse seems simple at first, many engineers are still reluctant to reuse. On the other hand, the hardware market is very fierce, and there are stringent time-to-market windows where large and complex hardware designs need to be ready [21]. In many cases, IP reuse allows designers to meet these tight deadlines. Before committing to any reuse strategy, several typical questions need to be addressed:

1. Do the components and/or designs required exist?
2. Where would the existing designs be located?
3. Should reuse be an option or would it be more efficient to design from scratch?
4. If the circuit is designed from scratch, should it be generalized such that it can be adjusted easily?

Many of these questions form the basis that determines which portions of the design can be reused and which portions need to be designed. Design reuse has many components that needs to be considered consisting of designing reusable components, integration, IP libraries, and retrieval. The following related works focuses on reuse related to hardware design.

2.1.2 Designing Reusable Components

Designing a component to be reusable requires more resources and time. The design has to be flexible and reusable for a wide range of applications. Reusable components usually fall within two different classes: Parameterizable IP (PIP) and Static IP (SIP). PIPs are flexible with variable parameters that can be adapted and adjusted to fit specific requirements. SIPs are completed designs that can be reused as is. To reuse a SIP, a good understanding of

the underlying HDL is needed in order to manually adjust the design to the specification of the user if needed. Results from [1] show that even though PIPs require more engineering resources to design than SIPs, reusing PIPs in hardware designs increases the time-to-market by about two to four times more than SIPs, making it a good investment in the long run.

2.1.3 Integration of IP

Another part of design reuse deals with integrating an IP into an existing design. Many existing circuits come from different sources and are built having various requirements and constraints. This makes integration of existing components difficult, especially in SoC designs. Interface mismatch, different parameters, specification, and functionality all need to be adjusted resulting in increased effort for integration. For example, wrappers need to be included in order to connect existing IPs with different interfaces [22]. The difficulty of integrating existing components can be attributed to the lack of standards that are in place [23]. As a result, hardware modules are designed without *plug and play* capabilities. In order words, an existing design cannot be placed and connected easily. If the time associated with the reuse of a component exceed 30% of the design time, the designer will most likely choose not to reuse [20].

In order to better facilitate IP reuse, there have been many groups pushing for standardization. Many of these efforts resulted in standards being adopted. IP-XACT is a standard for encapsulating reusable IP cores for SoC designs by laying the foundation for describing the naming conventions used, interfaces, core parameters, models, etc. As a result, the transfer and use of IP becomes much easier [24]. In order to improve productivity of reuse of building reconfigurable computing systems, Rollins et al. [25] extended IP-XACT such that the core can be used in a reconfigurable systems design environment. Xilinx's latest design environment, Vivado, leverages IP-XACT when packaging their IP [26]. Open Core Protocol (OCP) is an open standard that provides an interface for communicating between two entities [27]. The OpenFPGA CoreLib Working Group focuses on facilitating the interoperability of FPGA IP core across various design tools [28].

Bus Standards

Standardization of bus protocols allow for different independent components to communicate with each other [29]. By having a common bus interface, components can be designed to be *plug and play* which simplifies many integration problems. The Wishbone interconnect, managed by OpenCores, is an open sourced bus architecture [30]. Often seen in OpenCores designs, this allows building designs using OpenCores IP much easier. The ARM Advanced Microcontroller Bus Architecture (AMBA) is an open standard bus protocol that was designed to allow various peripherals to efficiently communicate with the ARM processor [31]. More recently, AMBA 3 and 4, Advanced Extensible Interface (AXI) interface, is being used

for building high performance SoC designs [32]. Another bus protocol is IBM's CoreConnect which has three different buses depending on the requirements peripheral's bandwidth [33].

Virtual Socket Interface Alliance

One of the earlier attempts to standardize interconnect was the Virtual Socket Interface Alliance (VSIA) [34]. The alliance consisted of a large number of noteworthy companies pursuing open standards to ease the integration and reuse of design components. With design components originating from different sources consisting of various interconnect methodologies, integration and reuse becomes a difficult problem. Various standards that VSIA have produced includes System-Level Interface Behavioral Documentation (SLIF) and On-Chip Bus Virtual Component (OCB VCI) [35]. SLIF focuses on defining a common standard for defining an interface of any IP. Essentially, it is an abstraction of what any interface would need for making transactions. OCB abstracts the bus interface and was one of the first attempts at interoperability standards for virtual components (VC). VCs and IPs are of the same meaning. Unfortunately, VSIA has since been closed; however, much of its work have been transferred to IEEE [36].

By having certain aspects of hardware design standardize, it creates a common ground for design reuse. Designs can be integrated and reused much easier rather than having to create an interface such that two components can interact with one another. Many industry leaders are adopting many of these standards which helps enhance reuse within the hardware community. With standards in place, more of the reuse process can be automated. Modules can be connected together automatically. A single description that is used such as IP-XACT allows different tools to read in third-part IPs. Furthermore, there are less custom cases and exceptions to handle during processing. For example, many vendors have specific options and flags that are incorporated into designs such that their tools can infer additional information and requirements on the circuit. These makes porting designs to different platforms and leveraging different tools difficult, which makes many of these designs concentrated in specialized areas [7].

With the rising of standards, the foundation for sharing, distributing, and reusing IP becomes simplified. The number of IPs can be expected to continually rise as reuse becomes an increasingly vital component to design productivity.

2.1.4 IP Libraries and Retrieval

Having hundreds of thousands and more of reusable IPs available, there requires a need for systems that can aid the designer in finding IPs of interest. Searching for designs can become time consuming and can often be unfruitful, accounting for about 20% of the designer's time [37, 38]. With a small database, a dedicated engineer can effectively manage an IP library. Yet, as this number continues to rise, it becomes difficult to not only process what designs

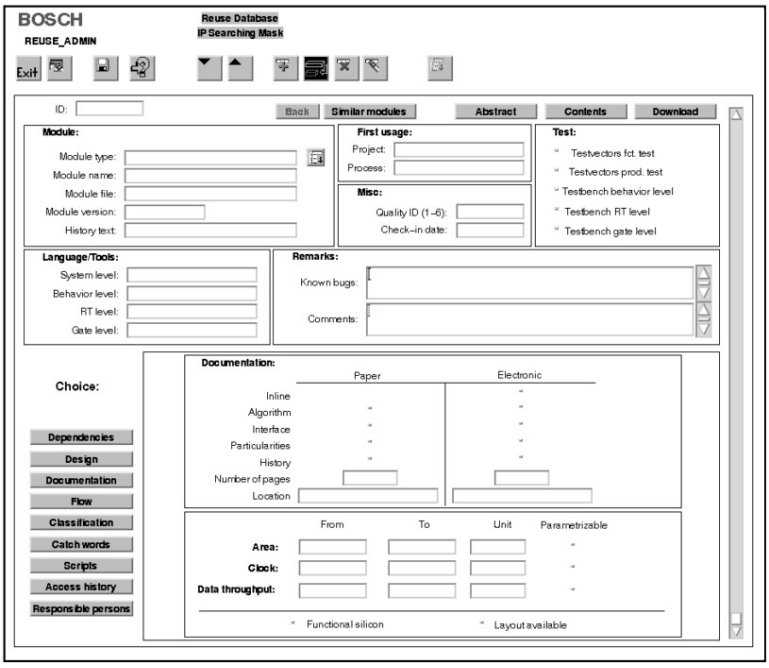


Figure 2.1: Query interface used by [1].

are in the database, but also the internal content. Even with a small database, a design may contain a number of modules. Many existing reuse-systems employ methods such as meta-data, case-based reasoning, design knowledge, graphs, etc. to aid the designer in finding the most pertinent designs quickly.

Keyword-based Retrieval

One of the most basic forms for retrieval are keyword-based retrieval schemes. In other words, there are specific keywords, metadata, or attributes which can be used to retrieve existing designs. These designs are typically contained in a large repository and may be partitioned into different application domains such as computer vision or signal processing to help reduce the search space.

Reutter et al. designed an efficient reuse system by leveraging keywords, properties, taxonomy, dependency, and similarity information [1]. Several keywords are chosen to describe the nature of the circuit. Various properties such as area and power are included as well. In order to limit the search space, a taxonomy for classifying IP was used. The information is entered into query model by the user as seen in Figure 2.1 and the system goes and searches for designs that match the user's requirements. Users have the ability to upload new designs into the database for other designers to reuse; however, to assure that the database of IP is well maintained, an administrator *marks* the designs that are of *high quality*.

OpenCores is an open database of reusable hardware modules. Many of these designs use various standards that allow them to be easily integrated. For example, the Wishbone interface is common to many OpenCores designs.

Design & Reuse is an Internet portal that provides various services and resources for design reuse. Their library contains thousands of IPs that can be retrieved in a similar manner to [1]. Taxonomy, keyword, and various properties such as device platform are used to search for the design of interest. However, most of the IP they provide are high-quality, verified designs and are usually not available for free. In other words, it is a marketplace for IP. The work of Schaaf et al. explored using IP Characterization Language (IPCHL) as an intermediate format for representing IP. IPCHL leverages case-based retrieval techniques in order to create an interface between a virtual marketplace for purchasing and moving IPs around repositories.

SAFIPS

Software Agents for IP Selection (SAFIPS) [39, 40] focuses on finding similar IPs from multiple vendors on the Internet using software agents to communicate with IP suppliers in order to retrieve similar designs for reuse. Questions from the software agent are presented to the user and the answers are configured into fuzzy logic rules and membership functions. The rules and functions are then sent to various software agents in order to determine potential candidates. Users have the option to submit test benches to the slave agent to validate functionality of the IP.

Xilinx Vivado

Vivado [26] focuses on IP and system design in order to accelerate design productivity. In their IP Integrator (IPI), Vivado provides a large catalog of highly parameterizable IPs that the design can reuse. The IPs come from various sources including IPs from Xilinx, third-party, and intra-company. The user can also add their own IPs and create custom repositories of designs. The search for IP is done by keywords associated with the name of the IP.

RODEO

RODEO [41] is a feature-based reuse system that focuses on using attribute-value pairs to define a specification. The search is done using Tversky-Contrast model: Let A be the set of features which belongs to x and B be the set of features which belongs to y, where x and y are the two designs of interest. The similarity of x and y, S is defined as $S = \theta \cdot (A \cap B) - \alpha \cdot (A - B) - \beta \cdot (B - A)$, where θ, α, and β are positive real numbers. Furthermore feature classes are taken into account such that similar functions will have a

higher similarity. For example, adds and subtracts have a higher similarity than add and multiply. The user performs the search by supplying RODEO with a specification that consists of features desired. The design object with the highest similarity value is the most likely candidate circuit.

READEE (Case-Based Retrieval)

One of the first attempts at creating an intelligent IP catalog for electronic design reuse was called Reuse Assistant for Designs in Electronic Engineering (READEE) [42, 43]. READEE leverages case-based retrieval (CBR) techniques to help guide the designer in retrieving the desired design. In CBR, there exists a case-base, C_b, that consists of solutions to previous problems, $C_b(P, S)$. The intuition is that for a given problem, p_{new}, the solution to p_{new} is similar to the solution of p_i, where $sim(p_{new}, p_i) == max(sim(p_{new}, p_i), 0 \leq i \leq |P|)$. In other words, similar problems have similar solutions.

READEE allows the user to specify various parameters in order to search for designs that met the user's requirement. An IP taxonomy is constructed where each class of IP consists of different attributes that characterize the desired function. The reuse flow first has the user select a specific taxonomy class. From the chosen class, attributes specific to the class are presented such that the user can define the necessary parameters for search. After the values to the attributes are given, READEE starts the retrieval process for IPs that are similar to the desired circuit.

Graph Matching Retrieval

A common representation for circuit designs is to describe it using graphs where in most cases, the vertices represent a primitive or functional component and the edges represent the interconnections between the components. Given a database of circuits represented as graphs and a circuit fragment, Whitam et al. [11] retrieve information about the designs using graph matching algorithms. Whitam extended the work of Ohlrich et al. [12] who used subgraph isomorphism to detect if a subcircuit, $S1$ is contained in another circuit $S2$, $S1 \subset S2$.

Software Reuse Systems

A similar reuse model can be seen in software, more commonly known as repository-driven assistant tools that help aid the design using the data contained in a repository of code. Some designs are more proactive in others in the fact that the tool actively suggests components to the user. Codebroker [44] was one of the first works that looked into having a reuse system autonomously delivers reusable components to the user; however, this automaticity still requires the user to explicitly query the system using comments and signatures. Code

Conjurer [45] pulls code out of thin air using a test-driven search. This approach is effective for designers that follow an agile development method where the testbenches are typically written before the actual code. Code Conjurer selects candidate components base on the success of the testbenches.

2.1.5 Reuse Summary

The culmination of these endeavors have helped designers leverage design reuse in a more efficient manner. Yet, these approaches still require the time and attention of a designer. Many of these approaches require the user to explicitly leave the design environment and manually search for designs to reuse. To search for the design, the user will have to describe the desired circuit in a format specific to the reuse system, whether it's using keywords, properties, or a circuit fragment. This component formulation may not be representative of the desired designed and may in fact differ from the way the IP is described in the repository [46]. Interfaces such as Figure 2.1 can be daunting with the amount of data needed to query a design. Though using keywords allows users to search for designs very fast and efficiently, it only looks for designs the user is semi-aware of. Just because the result of a search returns nothing, does not mean that the component does not exist. The design may, in fact, not exist, but at the same time, it may just be that the user is unaware of the existence of the design.

Aside from the retrieval, repositories need to be preprocessed as well. Most of the IP libraries of the reuse system have to be manually built. Metadata and keywords need to be assigned to designs for keyword-based queries. A defined taxonomy allows for hierarchical classifications of IP. Though many of these can be automated to an extent, there is a high cost of obtaining annotated data and lack of guidance on how to understand and fix mistakes. On the other hand, [11] finds existing designs by using subgraph isomorphism between a query circuit and a database of circuits. This approach is ideal since no manual preprocessing of the IP library is needed; however, the method of comparison is too restricting and fails to produce a match when two circuits are functionally equivalent but structurally different.

2.2 Circuit Matching

In order to implement the proposed approach of suggesting reusable components to the designer, a method of comparing circuits is required. 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 alike 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. This section explores related work in regards to *exact* circuit matching as well as *similarity*

circuit matching.

2.2.1 *Exact* Circuit Matching

Most applications regarding circuit comparison require the search for an exact match both structurally or functionally. Electronic design automation tools leverage these comparison for various stages of design. Layout vs Schematic (LVS) checking looks at the circuit's layout and schematic to validate that the two representations are an exact match. Technology mapping of primitive components requires an exact subcircuit match such that the subcircuit is replaced with a library primitive. The problem of finding subcircuits in a design is known as subcircuit identification.

Subcircuit Identification

Subcircuit identification (SI) is a problem where given two circuits C_1 and C_2, $|C_1| < |C_2|$, determine if $C_1 \subseteq C_2$. In short, SI identifies if C_1 exists in C_2. Subgemini [12] is a tool that uses subgraph isomorphism techniques to find subcircuits within larger circuits. Whitam et al. found similar designs in a circuit repository using a subgraph isomorphism technique as well [11]. Their approach is similar to SubGemini when performing the comparison; however, instead of *one-to-one* comparisons, they optimized the search to perform comparisons of *one to many*. DECIDE [47] implemented a SI algorithm where it identifies potential subcircuits by recursively checking to see if the neighbors of a candidate pair are identical. Ou et al. [48] uses a bit-parallel filtering algorithm to filter out impossible matches using predefined rules for transistor level designs. Zhang et al. [49] utilized fuzzy logic for comparing netlists with fuzzy attributed graphs. Due to the increase of circuit sizes to giga-scale, Hung et al. [50, 51] parallelizes the SI problem and incorporates general purpose GPUs (GPGPUs) for acceleration. These works focused primarily on the structural properties of a circuit and represented the designs using graphs in order to leverage the vast field of graph theory; however, as noted before, functionally equivalent designs can be structurally different.

Functional Matching

Module and functional extraction of a circuit have been important in terms of trying understand a given hardware design with only the netlist or other low level representations available. Rule-based functional matching was extensively studied by Takashima et. al [52]. Their approach was to use rule-based detection only when the structure of two circuits failed isomorphism. By doing so, Takashima ensured that his program not only checked for structural isomorphism, but also functional isomorphism as well. Eckmann et. al [53] used a rule-based system called OTTER (Organized Techniques for Theorem Proving and Effective Research) to assign functional meaning to circuits when given a detailed circuit description.

OTTER takes facts, or features about the circuit and uses rules to canonicalize them in order to compare if the function of the circuit is similar. BLEX [54] is another tool that extracts functional blocks from a network description similar to SPICE using graph algorithms. The search is performed by finding pairs of vertices with similar signatures. Doom et al. [55] identifies high-level components with a semantic matching algorithm using binary decision diagrams (BDDs). The work of White et al. closely resembles reverse engineering by trying to provide a general overview of the circuit with a module-level description [56, 57].

With the increase of methods for determining functional properties of a design, many of these methods have laid the groundwork for reverse engineering a hardware design from a given netlist. Hansen et al. [58] describes various methods and approaches to reverse engineering in order to discover the high-level structure of a given circuit. Subramanyan et al [59] 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 predefined bitslices which represents commonly used hardware. Yet, in order to be computationally feasible, the inputs are of the k-cuts are limited to six. Li et al [60] tackled the same problem but leverage formal verification techniques. They have extended their work to identify word-level structures of a circuit design [61]. Many of these applications and methods targets the design at a low level. Because the amount of information is abundant, extracting *meaningful* information can be difficult as well as compute expensive due to the amount of data that needs to be processed.

Combinational Equivalence Checking

Functional comparison of the entire circuit has typically been focused on using combinational equivalence checking (CEC) to determine if two circuits exhibit the same behavior. CEC is a specific form of the formal equivalence checking focused on the verification of digital circuits. The idea is to compare and see if two digital circuits are functionally equivalent [62]. Several techniques are commonly used in CEC such as BDDs, And-Inverter Graphs (AIGs), or satisfiability solvers (SAT).

Satisfiability (SAT) solvers are common tools for performing equivalence checking focused on hardware and software verification [63], but can be extended and used for CEC. This is done by representing both circuits in conjunctive normal form (CNF) and combining the two into a single circuit using a *miter* where the corresponding outputs are connected to an XOR gate. When a mismatch in the output occurs, then the resulting output will be a 1, indicating that the outputs do not match [64]. Still, these approaches have limitations such as input variable ordering and input output correspondence between two independent designs. This becomes difficult to leverage in the proposed reuse model as the snapshot of the reference circuit that is taken is not yet complete. The outputs of the design may not yet be defined and determining where the current *output* corresponds to the existing design becomes infeasible. These approaches also focus on finding an exact functional match since

the goal of these approaches is correctness and verification.

2.2.2 *Similarity* Circuit Matching

On the other hand, there has been little work done in determining the *similarity* between two circuits. Shi et al. [65] 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 [66] 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.

2.2.3 Circuit Matching Summary

Many of these endeavors do not give a good sense of how similar two circuits are. The focus of comparing two circuits have been focused on exact matches. Existing methods such as SAT or graph algorithms are NP-Complete and require extensive computing power. The proposed system needs to perform many comparisons quickly and efficiently for circuits of various sizes and complexities. Furthermore, the type of comparison desired is *one-to-many* instead of *one-to-one* and so the method needs to be able to scale with the size of the database.

In order to select the best approach to enhancing the productivity of the user, there are several requirements that have to be met.

1. **Flexibility:** The approach has to be flexible when comparing. The interest is not to determine whether two designs are the same, but how similar two designs are. Similar to a Google search where the webpage most similar, most pertinent to the user is displayed on top, or at least within the first page of the results.

2. **Performance:** In order to suggest reusable designs to the user, the comparison has to be fast. This is because if the time from when the snapshot is taken to when the results are given is too long, the design may have changed significantly such that the results suggested is no longer pertinent.

3. **Accuracy:** The results have to be meaningful. If the designs suggested do not resemble the circuit being designed, then the productivity of the user is not being enhanced.

4. **Naming should be ignored**: The match should be independent of naming information used. Even though naming information may contain insight to the overall function

and purpose of specific blocks, not all designs will use the same naming convention. Some may not even use it at all.

5. **Easy integration**: The entire automated process should be able to be leveraged and independent of any tool. This way, the approach is not limited to any one framework, but extensible to new and existing design tools.

2.3 Leveraging Research From Other Domains

Similarity search is a vast field with many different approaches that are optimized and targeted towards a specific domain. Nonetheless, many of these approaches can potentially be retargeted and applied to different fields. This section explores different domains where the concept of similarity and how it is calculated are investigated.

2.3.1 Molecular similarity matching

In terms of drug discovery, molecular similarity matching is a very important field [67]. There exists millions of known molecules stored in corporate-sized databases and the problem becomes how to find the molecules similar to a given query. One common approach is to use a feature-based similarity method with fingerprints [68]. A fingerprint is a bit vector where each index represents the presence or absence of a predefined substructure. Tanimoto's coefficient is then used to assess the similarity between the two fingerprints. Aside from fingerprinting, Rascal [69] uses graphs to represent molecules and finds the maximum common subgraph between the two. The larger the subgraph, the more similar the molecule.

2.3.2 Machine Learning

Concepts in machine learning and computer vision contain interesting applications. Given two images, how does one tell how similar the two images are? The same methods can be applied to hardware such as training a computer to be able to recognize and classify designs. Feature detection and object recognition in images can be applied to finding and recognizing specific functions in a circuit. Though many of these applications can be synonymous to circuit, the description and representation between a circuit and an image is very different.

The problem with many existing machine learning and computer vision methods is that they work on vectorial and structured data [70, 71]. The parallel nature of circuits, similar to graphs, are typically described in a non-vectorial fashion with an ever-changing structure. Even though the data in a graph can be placed in a vector, there is no common ground from one to another. The inputs of a circuit may appear at the beginning of one vector and at the end of another. Using a feature-vector to describe a circuit can be difficult in the

fact that the vector has to be descriptive enough for capturing the circuit data. Finding the correct set of features that can differentiate between designs is onerous as well. With more features and more data to represent the query, more processing is needed, but may result in overfitting. Yet, not enough features to represent the circuit will likely yield unsatisfactory results in terms of accuracy even though the results are computed quickly.

In addition, in order to train a learning algorithm, a large dataset is needed. A simple convolutional neural network used to classify images into ten categories is trained on 60,000 images in the CIFAR-10 dataset [72]. Each class has around 6,000 images and results in around 95% accuracy during classification while the CIFAR-100 dataset has 600 images per class with 100 classes and results in around 75% accuracy [73]. To produce similar results in terms of classifying circuits, a large dataset of designs is required to train such a system. Furthermore, the interest is which circuit is similar to the query rather than which class does the circuit fall under.

2.3.3 Software similarity matching

Even though hardware and software are fundamentally separated spaces, they share a multitude of similarities. There has been extensive research in software similarity matching for applications consisting of detecting malware, plagiarism, theft, code clones, etc in software programs. Many of the methods used for these applications can be extended for hardware similarity matching.

A common approach for software similarity matching is to use birthmarks. The idea of birthmarking is to take the characteristics that are inherent to a piece of software and compare these characteristics in order to assess the similarity of two programs. Tamada et al. [74] first introduced the idea of a birthmark in order to detect theft of Java programs, using information on constants, sequence of method calls, inheritance, and used classes. The similarity matching depends on the type of birthmarking scheme used. Many different types of birthmarking schemes have been explored such as sequence of bytes in executable [75], component dependence graphs [76], opcodes [77], class diagram [78], etc. In terms of determining similarity, techniques from using n-grams, bag of words, sequence alignment, graph theory, and others are used. The problem now is how to utilize the idea of birthmarking and develop a birthmark representation for hardware such that the requirements enumerated in 2.2.3 are met.

2.4 Summary

This chapter explored a wide range of existing work in regards to hardware productivity, IP reuse, and circuit matching; however, many of these methods are limited because the time complexity is exponential as the problem size grows. The methods used in searching and

comparing existing designs need to be scalable. If there exists a proper community structure of easing contributions and automating access, the number of existing designs can rapidly rise from tens to hundreds of thousands and more. Existing reuse systems require the user to explicitly search for circuits and can consume a significant portion of design time. Lastly, much of the work done has been focused on finding exact matches rather than analyzing the similarity of two designs.

The next chapters explore various representations and the methods used to compare the similarity of circuit designs. Comparing the similarity of circuits is a difficult task. 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, understand the tradeoffs, and select the best methods for matching.

The exploration starts with investigating methods in graph theory, followed by molecular similarity matching and Boolean matching, and finally birthmarks. The goal again is to be able to create a refined representation in order to quickly assess the similarity of two designs.

Chapter 3

Graph-based Approaches

Graphs are unstructured data that is free form and commonly used to model complex data. This representation is well suited to model circuits, where the vertices represent the logic components and the edges represent the wires connecting the components together. Graphs that represent circuits are typically simple directed graphs. Cycles are common in digital design which raises the complexity. There are many existing algorithms for extracting structural data from graphical representations in order to compare how similar two graphs are. The published work related to this section is in [79].

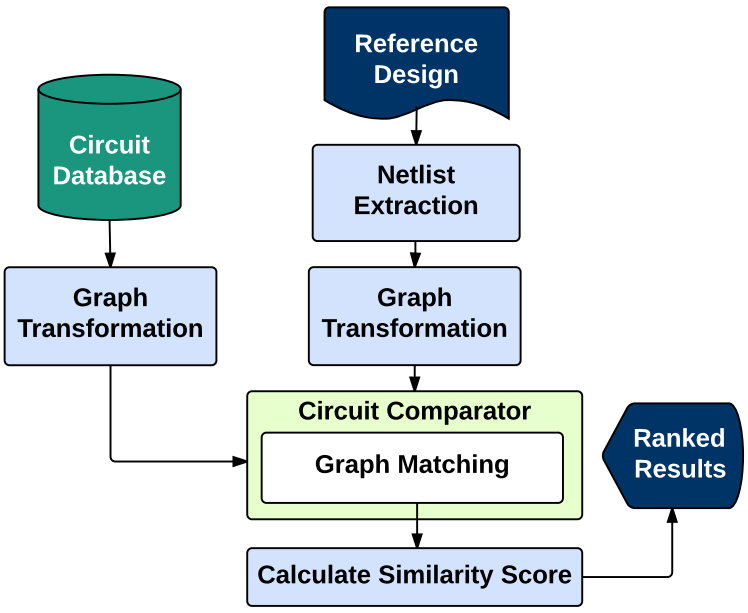


Figure 3.1: High level overview of graph based approach.

3.1 System Overview

The similarity flow for comparing circuits is depicted in Figure 3.1. A reference design is converted to a netlist which can be cast as a graph directly. Netlist was chosen as the input representation because it represented a common ground in hardware design. Whether the design entry was in Verilog, VHDL, or some other entry format, the circuit will eventually be synthesized down to a netlist. Even though the netlist is technology and platform dependent, by using a common synthesis tool such as Yosys [80], at least for generic HDL, the problem can be alleviated.

Once the netlist representation has been extracted, the graph methods can be applied in order to compare the reference netlist to the designs in the database.

3.2 Isomorphism

When two graphs, $C_1(V_1, E_1, l_1)$ and $C_2(V_2, E_2, l_2)$, are structurally identical, a graph isomorphism is said to exist, where C is the graph that describes the circuit, V is the vertex set, E is the edge set, and l is the labeling function for the vertex set. The labeling function does not include any information on the naming of the node. Rather, the label of the vertex refers to the primitive component used in the netlist. Isomorphism is detected if there exists a bijective function that maps $C_1(V_1)$ to the $C_2(V_2)$. Unfortunately, graph isomorphism only finds exact matches thus making it too restrictive. The odds that two independently design circuits will be isomorphic to each other is highly unlikely unless they are exact copies of one another.

3.2.1 Subgraph Isomorphism

By extending graph isomorphism to subgraph isomorphism, flexibility increases as now only a portion of the graph has to match. Formally, given two graphs $C_1(V_1, E_1, l_1)$ and $C_2(V_2, E_2, l_2)$ with $|V_2| < |V_1|$, a subgraph isomorphism is said to exist between C_1 and C_2 if V_2 is a subset of V_1, where $V_2 \subset V_1$ and $E_2 \subset E_1$.

Generic subgraph isomorphism algorithms such as VF2 [81] or Ullmann's [82] are both widely used and accepted. There exists many variations of subgraph isomorphism tailored to specific applications by adding certain heuristics and/or optimizations such as [12, 83, 84, 47]. VF2 and Ullmann are both guaranteed to return exact matches if one exists and does not give insight to similarity. A fuzzy subgraph isomorphism is explored such that it will not be as restricting as VF2 or Ullmann. The algorithm focuses more on the logic component and its neighbors rather than the edges that connect them and will return structures that are similar.

Fuzzy Subgraph Isomorphism

The implementation of the fuzzy subgraph isomorphism (FSI) takes from [12, 81, 47]. First a candidate vector is determined between G_1 and G_2. The candidate vector is a list of possible matches between the vertices in G_1 and G_2, or a candidate pair. By finding the candidate vector, only the most likely matches are explored. The candidate pairs are determined by the primitive type of the vertex. The more unique the component in the circuit, the more likely that a match can be found with the least amount of searching. Therefore, the candidate vector contains the components that appear the least in the input circuit.

To find if there is a potential match, both G_1 and G_2 need to be traversed. Traversal is done recursively so that if a mismatch is found, the matcher can backtrack to a valid state and check the next possible node for a match. At each level of traversal, the type of the logic component is compared. If the component of the nodes in question in G_1 and G_2 are identical, then the match counter is incremented, the node is marked, and then next node is compared. The matcher continuously scans the nodes until the match counter reaches the number of nodes in the smaller graph. This indicates that all the nodes in the smaller graph was successfully mapped to the nodes on the other graph and a subgraph exists. If all possible paths have been scanned and the number of nodes in the smaller graph does not match the match counter, no subgraph isomorphism is found.

Similarity Metric

The similarity of the circuit can be determined using FSI algorithm by calculating Equation 3.1; however, the problem is that FSI is not guaranteed to find a match. The similarity will depend on the candidate pairs that are determined initially. If the candidate pairs are poorly chosen with no matches found, then it will assume that there is no similarity between the input and pattern. On the other hand, if the candidate pairs are a good match, then even if the circuits do not match, a reasonable similarity metric will be returned. The similarity metric is calculated by the ratio of the number of nodes in the best possible match between the input and pattern and the size of the pattern graph.

$$\frac{sizeof(bestPossibleMatch)}{sizeof(patternCircuit)} \tag{3.1}$$

Performance Evaluation

The time complexity for isomorphism is NP-complete and would be infeasible for comparing complex circuits. Furthermore, when comparing one-to-many, the graphs need to be compared one by one and therefore, it is linear to the database size; however, there are many algorithms and heuristics in place to help accelerate the isomorphism problem. One such

approach is the decomposition subgraph isomorphism

3.3 Decomposition Subgraph Isomorphism

The idea of decomposition subgraph isomorphism is where each existing graph of a database is recursively decomposed into smaller graphs until only one vertex remains [85]. By decomposing the graphs, the subgraphs are represented once in the entire database, leading to a more compact representation of the database. The decomposition hierarchy of the graph database is stored in a tree structure. After every existing graph has been decomposed, the decomposition database can be used to determine if a subgraph exists in the input graph. The decomposition of the existing graphs in the database is a one-time procedure that is done offline. Moreover, additional graphs can be added without having to re-decompose the entire database. Subgraph detection is then done by trying to combine and build the circuit from the bottom up. Any patterns with a mapping to the input are a subgraph of the input graph.

Algorithm 1 Decomposing database of existing designs

```
1: function DECOMPOSE(G, D )
2:     if num_vertex(G) == 1 then
3:         return
4:     end if
5:     S_max = largestSubgraph(D);
6:     if S_max == G then
7:         return
8:     end if
9:     if S_max == ∅ then
10:        S_max = partition(G);
11:        decompose(S_max);
12:    end if
13:    decompose(G-S_max);
14:    D.add_node(G);
15:    D.add_edge(G, S_max);
16:    D.add_edge(G, G-S_max);
17: end function
```

The algorithm description for building the decomposition database is shown in Algorithm 1. Each circuit gets passed into the decomposer along the decomposition database. If the graph has only one node, then it cannot be partitioned further. S_{max} represents the largest subgraph that was decomposed already in D. If S_{max} is the same as the G, then there is no need to decompose further as this subgraph already exists in the tree. If no S_{max} exists, then G is partitioned into roughly two equal parts and each half is passed again into decompose

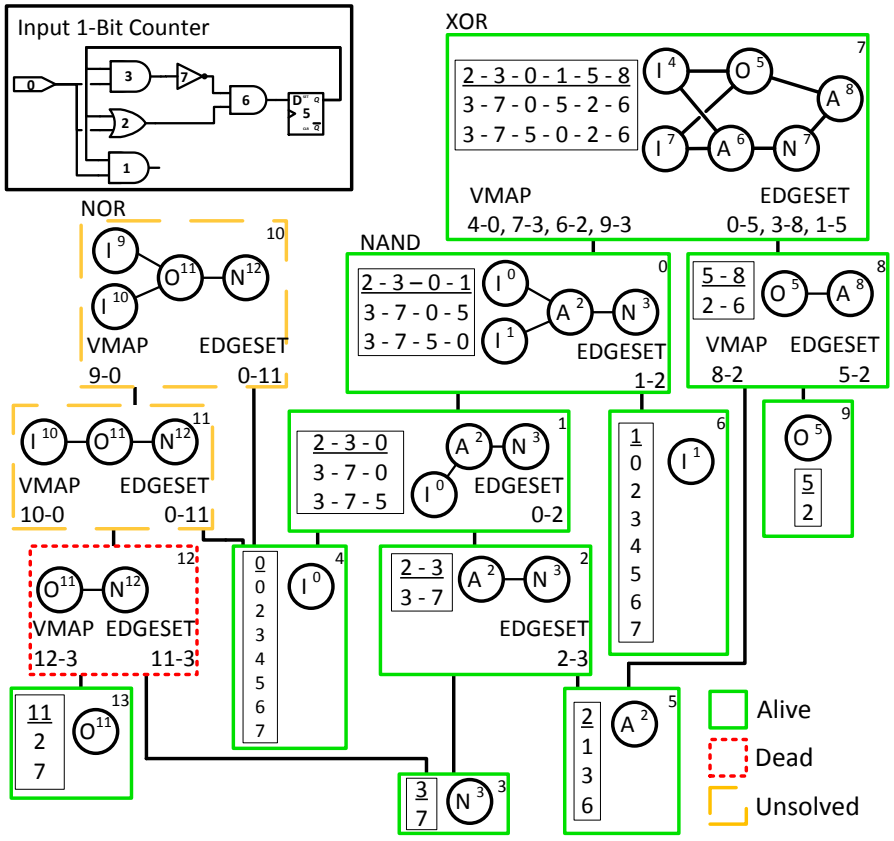


Figure 3.2: Decomposition subgraph matching example with a 1-bit counter.

recursively. The end result is that D is built where the nodes represents subgraphs of all the graphs that were decomposed and the edges represents the edge set that was removed during the partitioning.

Figure 3.2 shows an example of using the decomposition tree to find subgraph isomorphisms. The query circuit is a 1-bit counter. The database contains a NOR gate, XOR gate, and a NAND gate. There are three states the decomposition can reside in: unsolved, dead, or alive. Each decomposition node is initially labeled unsolved. The search first focuses on the nodes in the tree of a single vertex. For each of the decompositions with a single vertex, the vertices with the same component type in the input graph are added to a list of possible matches. An unsolved node in the tree is a possible match if its two children nodes are both marked alive. When no more unsolved decomposition nodes can be combined, all the decompositions that are marked alive are subgraphs of the input graph.

3.3.1 Similarity Metric

A similarity metric can be derived from the result of the matching. From each full pattern graph in the decomposition tree, the distance from the current position to the first node that is alive can be used as the distance metric. Furthermore, the number of matching nodes as well as the number of edges missing are used to further refine the similarity metric. Equation 3.2 below shows the similarity metric used.

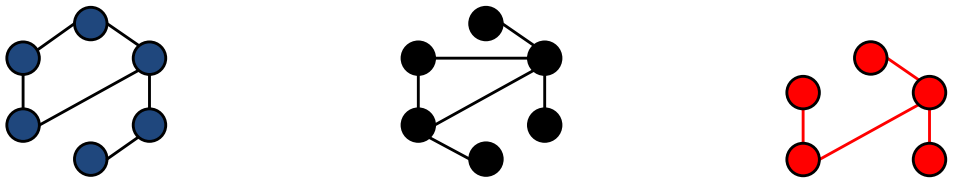
$$\frac{2 \times matchingNodes}{2 \times subgraphSize + \frac{missingEdges}{4}} \tag{3.2}$$

A heavier weight is placed on the matching nodes that were matched rather than the missing edges. This is because the nodes that were matched exist already somewhere in the circuit. Too high of a weight, and the missing edges will have little effect on the score. However, just because the nodes of the circuit match the input, the interconnects might not be the same. Therefore, the missing edges variable was added to the metric to differentiate the interconnections.

The similarity metric is not the same for every decomposition and depending on the decomposition of the pattern graphs, the similarity metric might be entirely different. One decomposition may provide a better metric than the other. However, the metric does not differentiate much when tested with two different decompositions. Therefore, the metric can be used to provide decent information as to how closely related the two circuits may be.

3.4 Maximum Common Subgraph

Maximum common subgraph (MCS) is a type of subgraph isomorphism in which the largest subgraph that is common to two given input graphs is determined [69, 86]. MCS focuses on edges that are compatible between G_1 and G_2. In other words, if two edges have the same source and destination type, then it is said that the two edges are compatible. These



(a) G1 (b) G2 (c) Largest common subgraph

Figure 3.3: Two circuits where (c) is a common sub-circuit of (a) and (b).

compatibilities are used to form the compatibility graph. By finding the modular product of G_1 and G_2, the relationship between the nodes of the compatibility graph can be determined.

The modular product of two graphs is used to determine the relationship of the nodes in the compatibility graph. Two vertices in the compatibility graph, cv_1 and cv_2, are adjacent if edge e_1 of cv_1 is incident on the same vertex as e_1 of cv_2 and e_2 of cv_1 is incident on the same vertex as e_2 of cv_2, or both e_1 and e_2 of cv_1 are not incident to e_1 and e_2 of cv_2, respectively. After the modular product of G_1 and G_2 have been determined, the largest cliques found are the largest subgraphs that are common to both G_1 and G_2.

The MCS approach compared to subgraph isomorphism is more flexible. Subgraph isomorphism indicates whether a graph is contained in a larger one and does not necessarily indicate similarity. MCS on the other can shows the largest subgraph that is common between both and can be used to indicate the portion of the circuit design that the two circuits share in common. The more the two share in common, the more similar they are.

3.4.1 Similarity Metric

The largest common circuit of the input and pattern circuit can be used to determine a similarity metric. The similarity is calculated by finding the ratio between the size of the largest common circuit and the size of the input circuit. The same ratio is calculated again but with the size of the pattern circuit. The larger ratio of the two is chosen as the similarity between the input and the pattern circuit. The equation is shown below in Equation 3.3.

$$max\left(\frac{sizeof(MCS)}{sizeof(inputCircuit)}, \frac{sizeof(MCS)}{sizeof(patternCircuit)}\right) \tag{3.3}$$

3.5 Implementation

For the graph-based approach, three different graph comparison techniques were explored: subgraph isomorphism, decomposition subgraph isomorphism, and maximum common subgraph. These methods provide the core of the circuit comparator. The VF2 algorithm [87] is used to perform subgraph isomorphism with the implementation readily provided by the Boost libraries [88]. To provide a more flexible alternative to subgraph isomorphism FSI was implemented, along with MCS and DSI using C++. The high-level overview of the implemented system can be seen in Figure 3.4

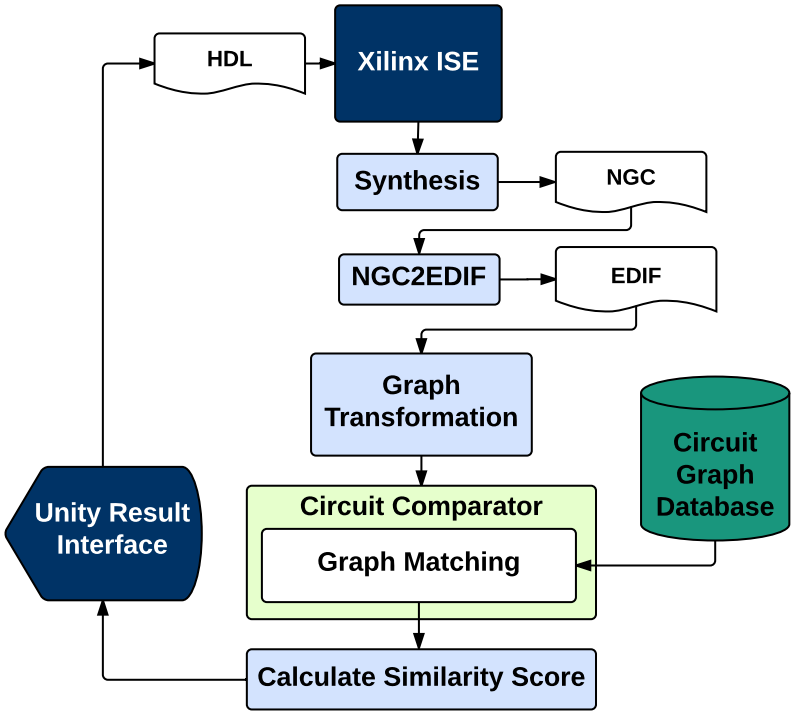


Figure 3.4: Implemented system overview.

3.5.1 Design Entry

The typical design entry for FPGA design is using design tools from vendors such as ISE or Vivado from Xilinx or Quartus from Altera. These design environments are full of features and tools that help aid the designer through every step of the design phase. This approach is flexible in that any design environment can be chosen.

Xilinx ISE is the front end chosen where the user enters the design using Verilog or VHDL. Since vendor tools typically are not open sourced, the system was loosely built as a proof of concept. As a result, in order for the netlist input to be obtained, the user has to manually synthesize the design. Once the design is synthesized, ISE produces an intermediate NGC file which is Xilinx's proprietary netlist format and contains both the logical design data and constraints. The NGC netlist is then converted to and Electronic Design Interchange Format (EDIF) with *ngc2edif*, where EDIF serves as the input representation into the reuse flow.

EDIF

EDIF is commonly used to describe a circuit design at the netlist level. The EDIF content contains the logic cells used, ports of each cell, modules, and nets that connects them to-

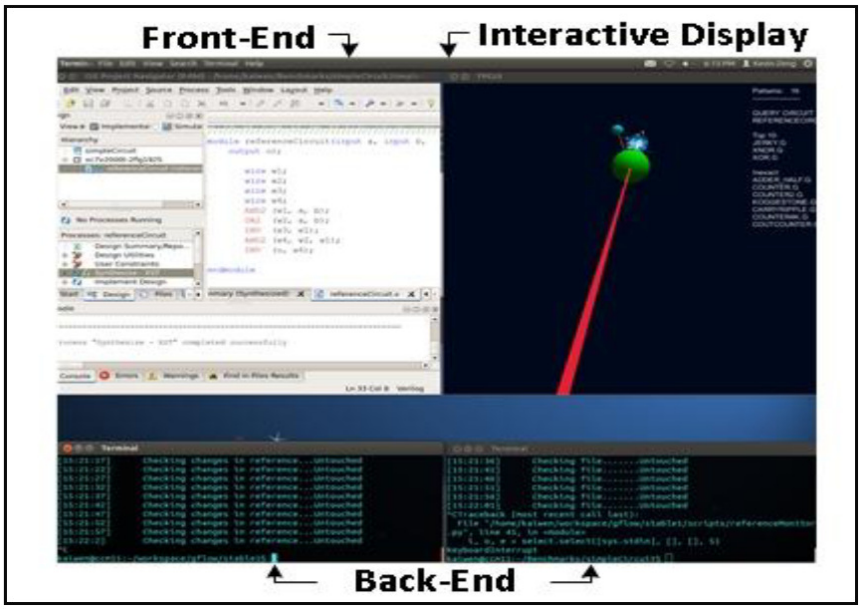


Figure 3.5: Implemented reuse flow into an existing design environment, utilizing Xilinx ISE as the front end.

gether. The Tools for Open-source Reconfigurable Computing (TORC) framework [89] gives the user access to all the design elements described in the EDIF. TORC also contains many methods that allows the user to quickly manipulate and extract components of interest. The conversion to EDIF from NGC is supported by the fact that NGC is proprietary to Xilinx. The EDIF format allows the back-end flow to be extensible to various other tools.

3.5.2 Displaying Results

The tool consistently monitors for changes in the *ngc* file, indicating that the design has been synthesized. *ngc2edif* is then used to obtain an EDIF file. Once the file is generated, the EDIF is read in and the circuit matcher searches for similar designs. After the matchers complete the search, the results are passed to the output to be displayed. The matches returned are ranked based on how similar the pattern circuit and the input are with the most similar circuit listed out front. In order to provide a better sense of similarity, a graphical representation of the results was explored.

The game engine, Unity [90] was used to develop the interactive interface. Game engines contain libraries and tools that make game development simple such as renderers for 2D and 3D graphic. In this case, the tools were used to create a visualization of the similarity between an input circuit and a multitude of patterns. By having such an interactive and visual representation of the results allows the user to gauge the similarity quickly. Access to documentation and data is simplified as well. Figure 3.5 is a snapshot of the implemented

system.

3.6 Results and Analysis

This section reveals results of the implemented system on comparing the similarity of circuits. The implemented graph-based approaches were analyzed and tested against standard benchmark designs.

3.6.1 Benchmark

A custom benchmark was first constructed and used to initially test the accuracy and functionality of the implemented matcher. The initial benchmark contains about 10 circuits ranging from 8 to 50 components. The circuits designed consist of logic gates, arithmetic operators, and counters. To test scalability on a larger scale, different benchmarks were used. The ISCAS models of the International Workshop for Logic Synthesis (IWLS) 2005 benchmarks were used as a larger dataset [91]. Around 30 circuits ranging from about 20 nodes to over several thousand nodes were used to test the scalability and accuracy of the different matchers.

3.6.2 Accuracy

The custom benchmark was used with a 1-bit counter circuit as the input circuit. Table 3.1 displays the similarity metric returned between a 1-bit counter and each of patterns from the three different matchers.

Table 3.1: Accuracy measurement of the three different matchers. The query passed in was a 1-bit counter using a Kogge-stone adder.

Circuit	VF2	FSI (%)	MCS (%)	DSI (%)
XOR	100	100	100	100
AOI	0	37.5	0	95
NAND2	100	100	100	100
Half adder	0	100	80	100
1-Bit Counter	100	100	100	100
2-Bit Counter	100	100	100	96
4-Bit Carry ripple Adder	0	87.5	66	89
4-Bit Kogge-stone Adder	0	87.5	66	82
4-bit Kogge-stone Counter	100	100	100	81

Even though both VF2 and FSI are both considered subgraph isomorphism algorithms, the results returned differ. The FSI algorithm returned more results due to the inexact methods and heuristics used. However, false positives may appear and matches aren't guaranteed to be found whereas if a match does exist, VF2 will find it. On the other hand, the FSI algorithm does provide more insight to the compatibility between the input and pattern circuit by calculating a similarity metric.

Since all the patterns are decomposed into a database for DSI, only the patterns in the database are used to find a subcircuit. This makes it difficult to see if the reference design is a subgraph of an existing design in the database. This is reflected when trying to match a 1-bit counter to a 4-bit counter compared to the other methods that were able to return a full match. At the same time, false positives may occur as well. For example, the matcher returned that the similarity between the AOI gate and the input is 95 % even though the structure of the AOI gate is fairly different from a counter.

ISCAS Benchmark

In order to assess the quality of the results for larger circuits, the designs in the ISCAS benchmark was used. Circuit s5378 in the ISCAS benchmark was modified by having one of its gates removed. The circuit was relabeled as s5378mod. By placing the actual s5378 circuit into the database, the matchers are expected to be able to find a match with the modified circuit to the original. IWLS-40 was used as the database. From the results shown in Table 3.2, all three matcher were able to successfully identify an exact match when comparing the input with itself as the pattern. Only DSI and FSI matchers gave additional information on the similarity between the input and pattern circuit. Many of the comparisons for FSI returned 0 as well due to the fact that the matching starts with a candidate pair, and if the candidate pairs are poorly chosen, then the matcher won't be able to find relevant results.

Functional Testing

As noted before, using graph-based approaches to compare the netlist of the circuit compares the structural layout of the design. Yet, there are some inherent functionality that can be captured based on the layout of a design and the arrangement of the components. Three different designs of XOR gates shown in Figure 1.2 are compared using the approaches above. The results of the match can be seen below in Table 3.5, 3.3, and 3.4. VF2 subgraph compares exact subgraph isomorphism and would, in this case, return no possible matches.

Table 3.2: Comparison of results between FSI, VF2, and DSI.

Circuit	FSI (%)	VF2	DSI (%)
s13207	0	-	79
s5378	100	1	91
s5378mod	100	1	100
s15850	0	-	76
s1494	0	-	83
s1488	2.78	-	84
s1423	0	-	80
s1238	0	-	85
s1196	12.02	-	84
s838.1	0	-	83
s832	0	-	86
s820	6.46	-	86
s713	0	-	90
s641	0	-	89
s526n	1.28	-	85
s510	0	-	87
s526	0	-	86
s444	0	-	85
s420.1	0.96	-	85
s400	0	-	84
s386	1.70	-	84
s382	0	-	79
s349	1.49	-	84
s344	0	-	84
s298	0	-	86
s208.1	0	-	81
s27	40.0	-	93
b01	0	-	86
b02	4.16	-	87
b03	0	-	84
b04	0	-	84
b05	0	-	83
b06	0	-	88
b07	0	-	84
b08	0	-	84
b09	0	-	84
b10	0	-	82
b11	0	-	81
b12	0	-	81
b13	14.4	-	81

Table 3.3: Accuracy measurement of the three structurally different XOR gates using a MCS approach.

Circuit (REF)	XOR (1.2a) (%)	XOR (1.2b) (%)	XOR (1.2c) (%)
XOR (1.2a)	100	50	75
XOR (1.2b)	40.0	100	80
XOR (1.2c)	37.5	50	100

Table 3.4: Accuracy measurement of the three structurally different XOR gates using DSI approach.

Circuit (REF)	XOR (1.2a) (%)	XOR (1.2b) (%)	XOR (1.2c) (%)
XOR (1.2a)	100	94.1	72.7
XOR (1.2b)	95.5	100	70.5
XOR (1.2c)	90.9	94.1	100

Table 3.5: Accuracy measurement of the three structurally different XOR gates using a FSI approach.

Circuit (REF)	XOR (1.2a) (%)	XOR (1.2b) (%)	XOR (1.2c) (%)
XOR (1.2a)	100	50	75
XOR (1.2b)	50	100	40
XOR (1.2c)	75	40	100

3.6.3 Performance

Having near real-time performance is critical for suggesting reusable designs for the user. This is because the design is continuously changing. The performance and scalability of the graph methods are explored in this section.

FSI and VF2

The FSI was tested alongside with the VF2 algorithm. The input circuit used was s5378 which is a decently large circuit of over a thousand nodes in the ISCAS benchmark with the IWLS-40 database. Performance of both the FSI and VF2 algorithms are not necessarily dependent on the size of the patterns or input circuit, to a certain degree. From the results shown in Figure 3.6, there is no apparent relationship between the execution time and the size of the pattern. This is due to the heuristics used in choosing candidate pairs as the initial starting point. If different candidate pairs do not produce a result, then the algorithm assumes that there is no match to be found and immediately exits.

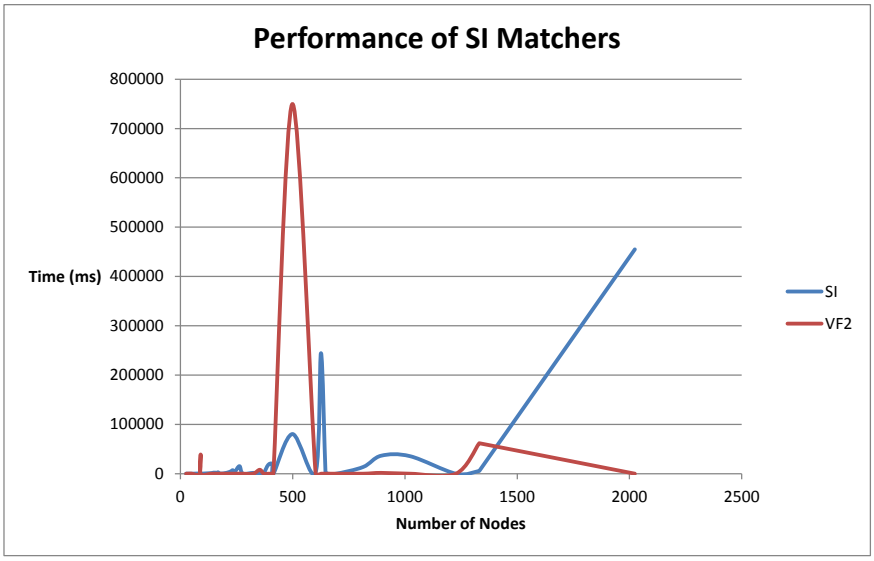


Figure 3.6: Execution time of FSI and VF2 for varying pattern circuit sizes.

Maximum Common Subgraph

The results the MCS matcher returned can be seen in Figure 3.7. For small circuits of less than 50 nodes, MCS performed well, but as the number of components increase to more than 50, the execution time increases significantly compared to the other matchers. This is not only true for the patterns, but for the input circuit as well. If the input circuit is fairly

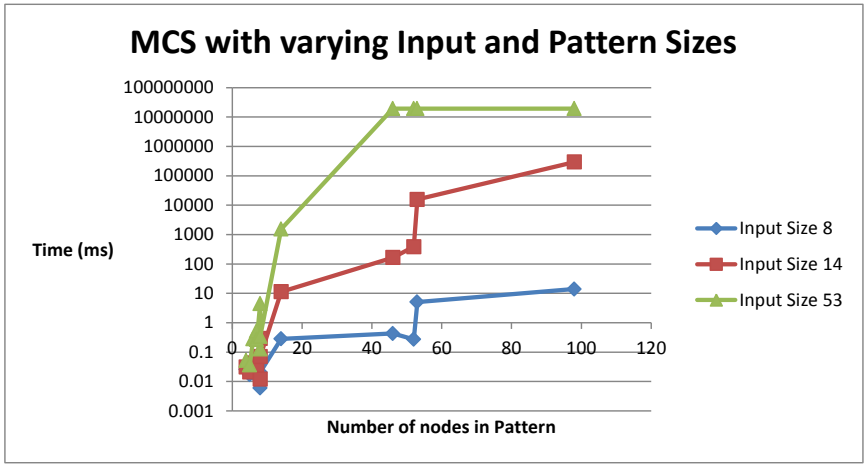


Figure 3.7: Execution time MCS with varying input and pattern circuit sizes.

small, and the pattern circuit is a couple hundred, MCS still runs reasonably well. However, if both the input and the pattern are large circuits, run time will increase significantly as seen in Figure 3.7

The input circuit of size 53 was terminated due to the execution time being several magnitudes higher than the previous input. One possible cause of this drastic increase is the clique detection. As the number of nodes increases, the size of the compatibility graph increases as well. The largest clique detection is also known as an NP-Complete problem and can be seen from the results. Furthermore the modular product of the two compatibility graphs will produce an extremely dense graph.

Decomposition

The decomposition method has two main parts: the decomposition of pattern graphs and the subgraph matching. The decomposition takes a significant amount of time with an increasing database size. This is because the algorithm does a subgraph match with every single decomposition. Moreover, the larger the circuit, the more time it will take to decompose due to the amount of partitioning required to break the entire circuit down to a single vertex. To test the decomposer, the database IWLS-5, IWLS-10, IWLS-20, and IWLS-40 were decomposed a total of five times each. Because decomposition is not unique, the databases were ordered from smallest circuit to largest and largest circuit to smallest in order to test how well DSI matches using different decompositions. Figure 3.8 shows the overall execution timing for decomposing the databases. Subsequent patterns added onto the circuit will take longer to decompose due to the search for a possible sub-circuit in the entire

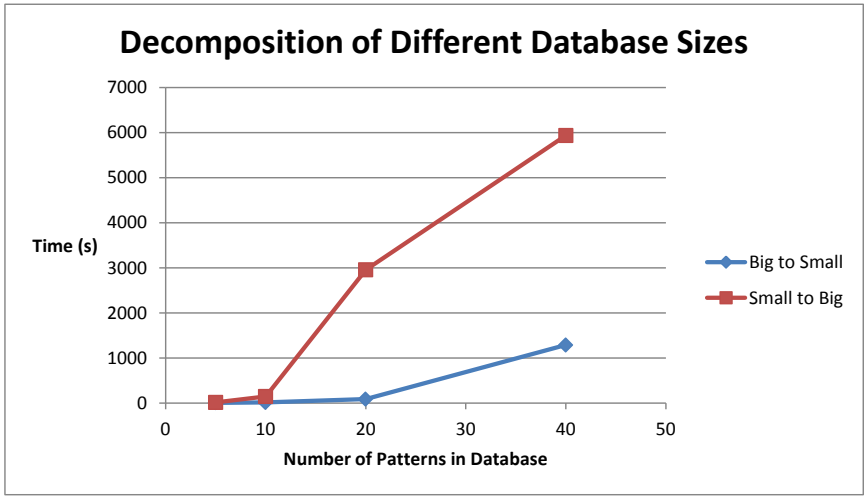


Figure 3.8: Execution time of Decomposer for different database sizes.

Table 3.6: Comparison of decomposition matcher using two different decompositions trees.

Database Size	Large to Small (sec)	Small to Large (sec)	Percent Difference (%)
5	28.826	28.956	0.44
10	36.450	35.919	1.46
20	58.836	59.145	0.523
40	179.126	179.210	0.046

decomposition tree. On the other hand, there was a noticeable difference in execution time depending on the order of the patterns that were passed into the decomposer. It can be seen in Figure 3.8 that decomposing the larger circuits first yields a lower overall decomposition time.

This is probably because with the first circuit, a majority of the operation done is partitioning. If the larger circuit is inserted at the end, there will be a significant amount of FSI tests running. Despite the two different decompositions, it can be seen that the overall DSI time is unaffected in Table 3.6. Therefore, to make the decomposition as efficient as possible, the larger circuits are decomposed first followed by consecutively smaller ones. This only applies to the initial decomposition rather than incremental decomposition when an entire database is set to be decomposed.

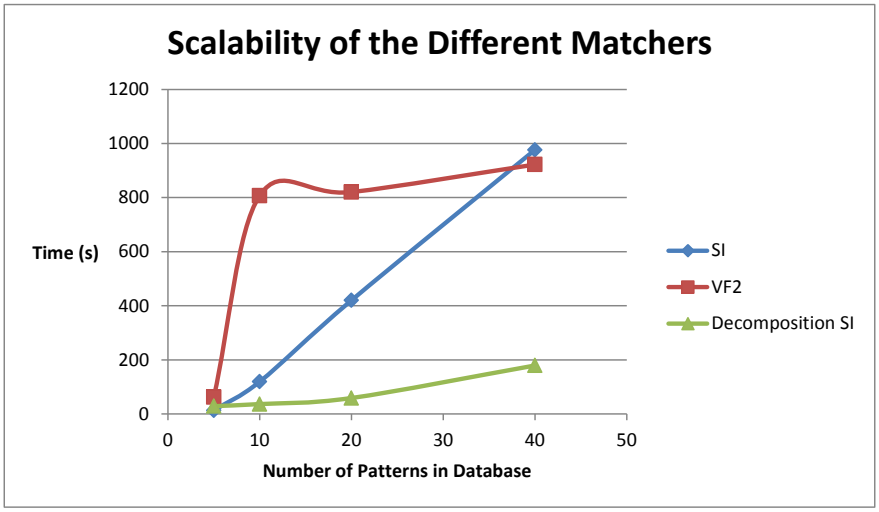


Figure 3.9: Execution time for varying database sizes.

3.6.4 Scalability

Two different aspects are observed here: The scalability based on the increase of reference circuit size and increase in database size. In order to test how well the program scales as the number of patterns in the database grows, each matcher was tested with the IWLS-5, IWLS-10, IWLS-20, and IWLS-40 database.

From the results shown in Figure 3.9, decomposition scales extremely well compared to the subgraph isomorphism matchers. Again the primary reason is that similar sub-circuits are represented once wherever possible, and the tree like structure of the decomposition allows an efficient search for similarities.

The FSI matcher scales linearly as expected because the search is performed linearly to the number of circuits in the database. For VF2, there was one circuit in the database of ten that took exceptionally long to determine if a sub-circuit exists. As explained early, this is due to the topology of the circuit and how there are many similar components between the input and circuit. Nonetheless, the overall execution time for VF2 is still several times larger than DSI. To further analyze how well the algorithm will scale for databases larger than 40 circuits, a polynomial regression was calculated from the data points of the decomposition SI results. The polynomial regression is shown below in Equation 3.4.

$$0.1132x^2 - 0.8403x + 31.408 \tag{3.4}$$

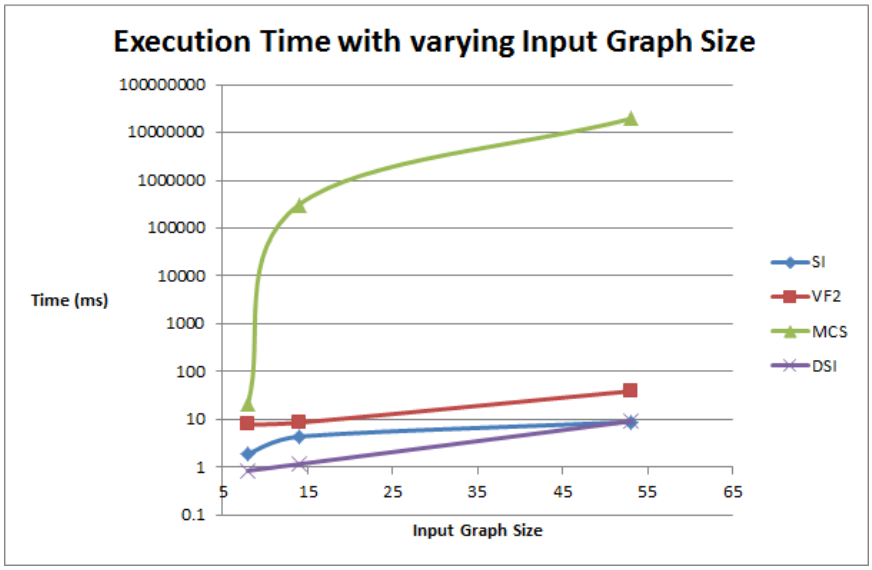


Figure 3.10: Execution time of Matchers for varying input circuit sizes for a database of 13 Circuits.

If the database grows to about a size of 1000, then the overall execution time of the decomposition SI matcher would be expected to take around 31.2 hours to try and find a match. However, this is also dependent on the size of the input graph.

Run time comparison between the three different matchers can be seen in Figure 3.10 using the simple dataset with varying input graph sizes. MCS run time increases significantly after doubling in size whereas the other matchers performed decently well.

3.6.5 Limitations

Very quickly, the comparison became expensive for very simple circuits and a small number of circuits in the database. MCS quickly becomes infeasible for larger benchmarks of simple designs. DSI, though much more efficient, consumes a lot of memory since each node in the tree represents a subgraph of the original graph such that it can be compared quickly and would eventually be infeasible for larger and more complicated circuits. Even for fairly small circuits, the run time is nowhere near the expectations of real-time systems.

In order to be more applicable, the comparator needs to be able to handle circuits of a much larger size without consuming the computation resources shown with the graph-based approaches mentioned above. In addition, the methods needs to be flexible enough to identify functionally similar designs. Only the DSI returned favorable results; even with two different implementations of adders. Though when looking at circuits that are composed of more complex functions, the results become more ambiguous as seen in Table 3.2. Even though the other circuits are functionally different, the similarity score does not differentiate much. This is most likely attributed to the commonality of the structures used in the layout of the circuit. There is only a small amount of commonly used primitives which does not provide much variation when building the circuit. Also, the DSI tree is not always going to produce the same results each time it is built. Therefore the quality of the may vary depending on the constructed tree.

3.7 Summary

Graphs have the ability to capture the raw nature of unstructured data which makes them an ideal candidate for representing circuits. This chapter explored graph-based approaches for comparing two designs. A system was implemented and integrated with Xilinx ISE as an initial proof of concept. Results are promising, though the designs used are quite simple with the inability to handle larger and complex designs without long run-times. The next chapter discusses alternatives for comparing designs incorporating ideas from Boolean matching and molecular similarity.

Chapter 4

Molecular Similarity Approach

There are an abundance of molecules and proteins that are built from many different combinations of elements and atoms. Due to that abundance, chemists and drug specialists require specific methods to find relationships between molecules in order to better understand the different compounds used. This is so new and better drugs can be developed with lower failure rates. Relationships between molecules are based on a principle that structurally similar compounds will exhibit similar properties, commonly known as molecular similarity [92].

This same principle can be applied to digital circuits. The depiction of a molecule and a circuit is very similar where the atoms in the molecules represent primitive circuit components and the bonds represent wires. Thus, leveraging molecular similarity techniques for circuit similarity is a viable option. One common method for determining molecular similarity is with the idea of fingerprints [93].

4.1 Fingerprinting

A typical fingerprint is a bit-vector where each bit represents the presence or absence of a specific structural pattern in a molecule. This is also commonly known as a feature-based representation. The intuition is that the more patterns two molecules share in common, the more similar the molecules are to each other. Incorporating the same ideas, the representation of the circuit can be represented in a vectorize form. Thus, comparisons can be done more efficiently and many similarity methods for vector data can be utilized. The idea of fingerprinting is extended and applied for comparing circuits.

The first step is to construct the basis for the fingerprint: what each index in the vector represents and how many features should be extracted. In molecular similarity, each index correlates to a specific graph that represents a common structure among different molecules

and can range anywhere from 166-bits using the MACCS key to 2048 using the Daylight fingerprint or more [94]. For digital circuits, finding common structures is difficult as there are not many that are *recurrent* at the netlist level. There countless arrangement of gates; however, this all depends on steps performed during synthesis. Based on constraints placed, designs may synthesize down to the netlists using different arrangements. A pattern database can be leverages where multiple structural descriptions of a specific function is described. As a result, to see if a specific function exists, the occurrence of the patterns are checked to see if there exists a match.

Though feasible, the number of patterns can be extensive utilizing different combinations of primitive components. In order to remedy this, functional matching was explored where each index in the vector represents a specific function that is present within a design. The intuition is that the more functions that two circuits share in common, the more similar the circuits are to each other.

4.2 System Overview

In order to construct the fingerprint, the capability to extract high-level functions from a netlist is required. Many of the techniques used for functional extraction incorporates methods from [59, 95]. The high-level overview of the system is shown in Figure 4.1

The process to extract the high-level components from a given netlist incorporates four main

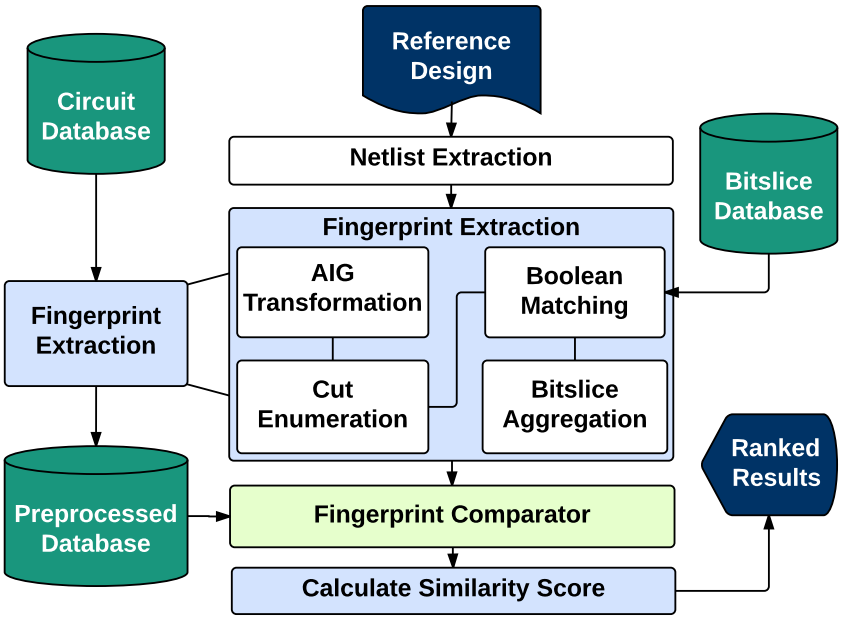


Figure 4.1: High-level overview of molecular similarity approach.

steps: converting the netlist to an AIG representation such that the function can be easily retrieved, cutting the netlist into smaller blocks for faster processing, matching of the small blocks against a predefined database of bitslices, and finally aggregating the cuts to identify the high-level function.

4.2.1 And Inverter Graphs

The representation of the circuit needs to be in a format where the functionality can be extracted easily. Various methods of representing circuit functions include conjunctive and disjunctive normal form (CNF/DNF), BDDs, AIGs, etc. Comparatively, AIGs are more scalable compared to other methods because the construction of the AIG is proportional to the size of the circuit [96]. Furthermore, the graph like structure of AIGs allow large Boolean functions to be manipulated and extracted effectively. Thus AIGs were chosen as the representation this approach.

An AIG is a directed acyclic graph (DAG) that represents the Boolean function of the circuit using only AND gates and inverters. There are two types of node in an AIG: primary inputs and 2-input AND gates. Inverters are represented by a property on the edge, commonly represented as a dot in the graphical representation. An example conversion from a 2-input XOR gate to an AIG is shown in Figure 4.2.

Conversion from the netlist to an AIG graph is simple since most of the basic primitives

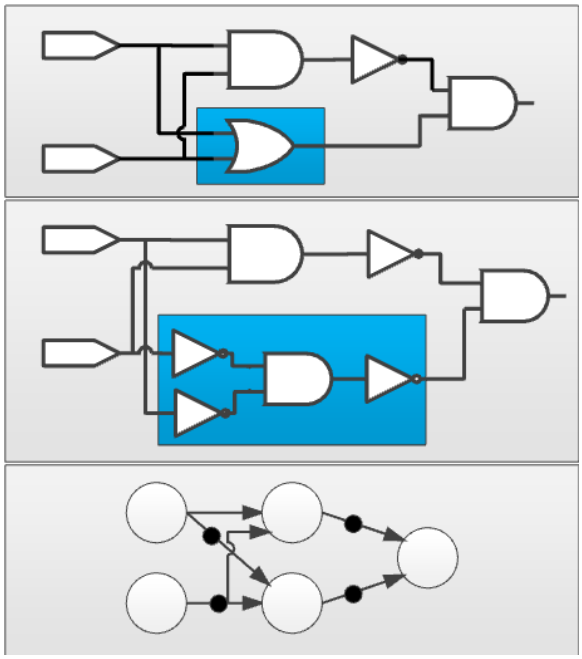


Figure 4.2: Conversion of a 2-input XOR gate to an AIG.

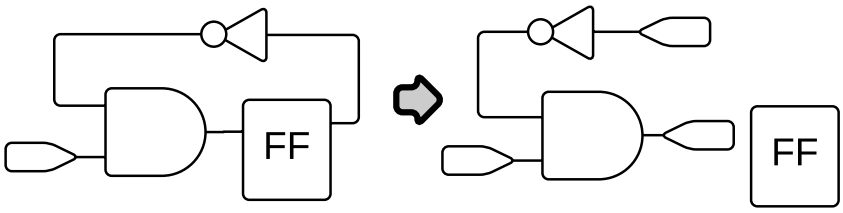


Figure 4.3: Removing feedback loops from sequential circuits.

can be easily mapped into an AND inverter representation. Known gates are substituted by their AIG counterpart. Furthermore, the function of look-up tables (LUTs) can be converted into a Boolean formula and replaced with an equivalent AIG. Note that AIGs only capture combinational logic. For sequential designs where feedback loops are common, the input and output of flip flops are assigned as primary output and inputs respectively as seen in Figure 4.3. This way, the sequential elements and cycles are removed. Though combinational loops can occur, they are not common in reconfigurable systems nor are they desired because the data of the output node will essentially be in an unstable state. In fact, most design tools will at least warn the user if a combinational loop is detected, if not throwing an error.

4.2.2 Bitslice Database and Fingerprint Definition

In order to construct the fingerprint, the meaning of each index needs to be predetermined. Common functions among digital circuits are chosen. These functions include primitive-type operations such as adders, counters, memory, multiplexers, control logic, comparators, etc. Not only are the functional components captured, but structural features are included as well by observing number of input outputs (I/Os), average fanout and fanin, size of circuit, and components used. In addition, the idea of a bit-vector is extended upon to include frequency of components. This is because many of the operations are common in circuits and another dimension is needed to capture more distinctive features. Total, the fingerprint is made up of 252 bits. 30 bits are assigned to counters, 2-1 mux, 3-1 mux, 4-1 mux, adders, XOR, AND, and OR gates. The 30 bits represent the size of a specific structure found. For example, a 4-bit adder vs a 32-bit adder. In addition, structural properties of the circuit were added. These properties include primary inputs and outputs, flip flops, number of LUT2, LUT3, LUT4, LUT5, and LUT6, average fanin and fanout, and total number of nodes and edges.

The functions that are being extracted are represented beforehand as a bitslice. A bitslice is defined as a Boolean function with one output and a small set of inputs that can be combined to construct larger functions [59]. The goal is to then identify the bitslices in the circuit by utilizing the AIG. This is done using cut enumeration.

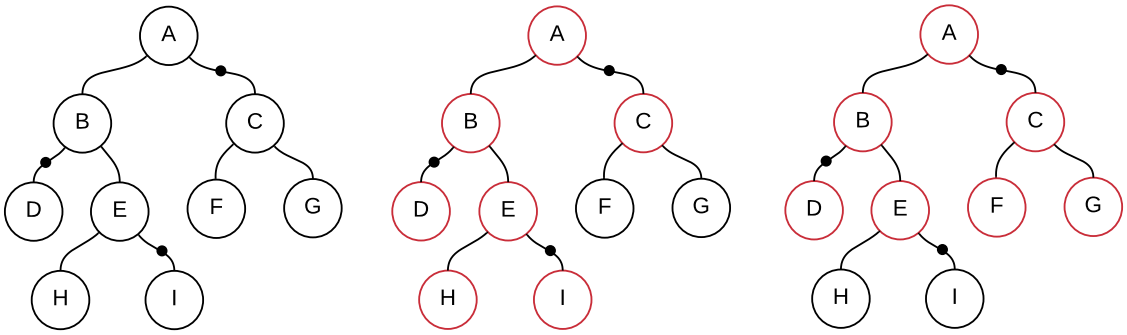


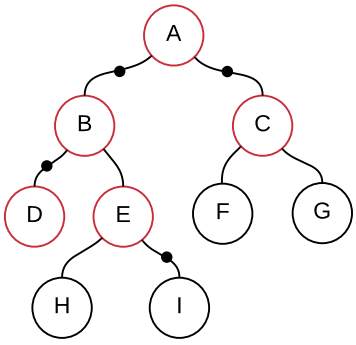
Figure 4.4: 4-cut enumeration on Node A. There are no other possible 4-cut on Node A.

4.2.3 Cut Enumeration

In order to extract the functions from the AIG with the bitslices, the circuit needs to be partitioned into cuts using k-cut enumeration. A cut is the immediate fanin cone of a specific node with at most k inputs. These cuts are enumerated for each node n in the AIG. A feasible cut is a cut that is no larger than k. The value of k is limited to 6-feasible cuts as the computation required for larger cuts increases exponentially [59]. Because of this limitation, the bitslices in the library all have inputs no more than k. Figure 4.4 shows an example where a 4-cut enumeration is performed on Node A. All possible enumeration is performed on every Node n in the AIG.

4.2.4 Boolean Matching

Once enumeration over the AIG is completed, a set of cuts for each node is obtained. These cuts can then be compared and aggregated with the functions in the bitslice library. The function of the cut is calculated by directly deriving the truth-table based on the AIG of the



DEC	A
000	1
001	0
010	0
011	0
100	1
101	0
110	1
111	0

Figure 4.5: Truth table extraction and storage for a given cut.

cut. With $k = 6$, the truth table can be calculated and stored efficiently in a 64-bit integer.

Figure 4.5 shows the truth table extracted for the indicated 3-cut on node A. Since the cut is of size 3, the truth table is replicated to fill 64 bits since the other inputs can be considered as don't cares. Thus, the 64-bit value that represents the function of the indicated cut is *0x5151515151515151*.

With the function of a cut determined, two functions are said to be equal if they belong to the same NPN-equivalence class. This means that regardless of negating the inputs, permuting the inputs, or negating the outputs of a cut, the overall functionality it represents is equivalent. The permutation of the inputs guarantees that input ordering will not be affected. For example, the truth table for the variable ordering abc for $a + bc$ will be different if the variable ordering cba is used. Yet, the two exhibit the same function. The negation of input and output assures the negation of specific inputs and outputs will not affect the result of the match.

4.2.5 Bitslice Aggregation

Once the existence of a cut with the same function as a bitslice is found, the cut is marked such that the it can be aggregated with other known bitslices. For example, in Figure 4.6, $CUT1$, $CUT2$ and $CUT3$ are all found to match an AND2 bitslice. Aggregation looks at the interconnections of the bitslices and aggregates them to be represented by the larger AND4 function. Essentially by determining the low-level primitives and how they are related, higher-level functionality can be deduced. This is done similarly with adders and multiplexers as well. The carry bit out of the adder goes into another adder bitslice signifying that there is a relationship between the two. Multiplexers can be aggregated if two share the same select signal.

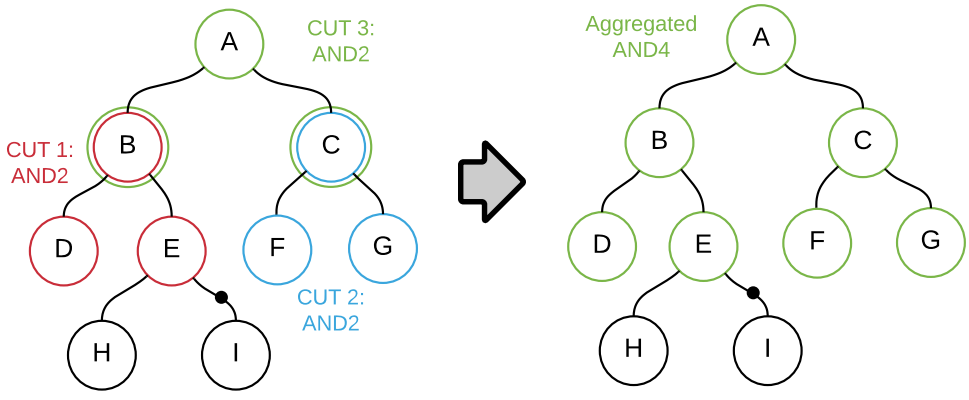


Figure 4.6: Aggregating three 2-input AND gate to make a 4-input AND gate.

4.3 Similarity

Data obtained from the functional extraction process is populated in the fingerprint. In terms of comparing the similarity of fingerprints, Tanimoto's similarity, defined in Equation 4.1, is commonly used.

$$\frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4.1}$$

Since the fingerprint 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 calculate the similarity between the two fingerprints. Therefore the structural and constant similarity metric is a number that depicts the Euclidean distance between the two different components.

4.4 Implementation

Much of the front end design entry concepts were reused from the graph based approach. Since functionality is extracted using AIGs, the netlist contains basic primitives that makes it simple to convert to AIG. The methods mentioned above were implemented in C++. The high-level overview of the implemented system can be seen in Figure 4.7.

4.4.1 Converting design to AIG

In an FPGA, most of the hardware resources on the device is composed of logic blocks consisting of look up tables (LUT). When a design in behavioral HDL is synthesized onto the FPGA, the logic is mapped into LUTs. LUTs contain the truth table of a specified function. In EDIF, the LUT truth tables are specified as a property of the cell. From the truth table, the minterms can be extracted to create a Boolean formula in DNF. This creates an unoptimized DNF and results in a large AIG. Espresso [97] is a logic minimization tool that is used to optimize the logic required to implement the specific function, resulting in a smaller AIG for the specific LUT. Once the AIG for the LUT is extracted, the LUT in the graph representation is replaced with its AIG representation. A mapping for every other primitive is used to replace the indicated cell with its AIG counterpart such as MUX and other gates.

4.4.2 Design Entry

In addition to the ISE design entry used in the previous chapter, the reuse flow was integrated with National Instrument's (NI) LabVIEW [98]. LabVIEW is a graphical design environment

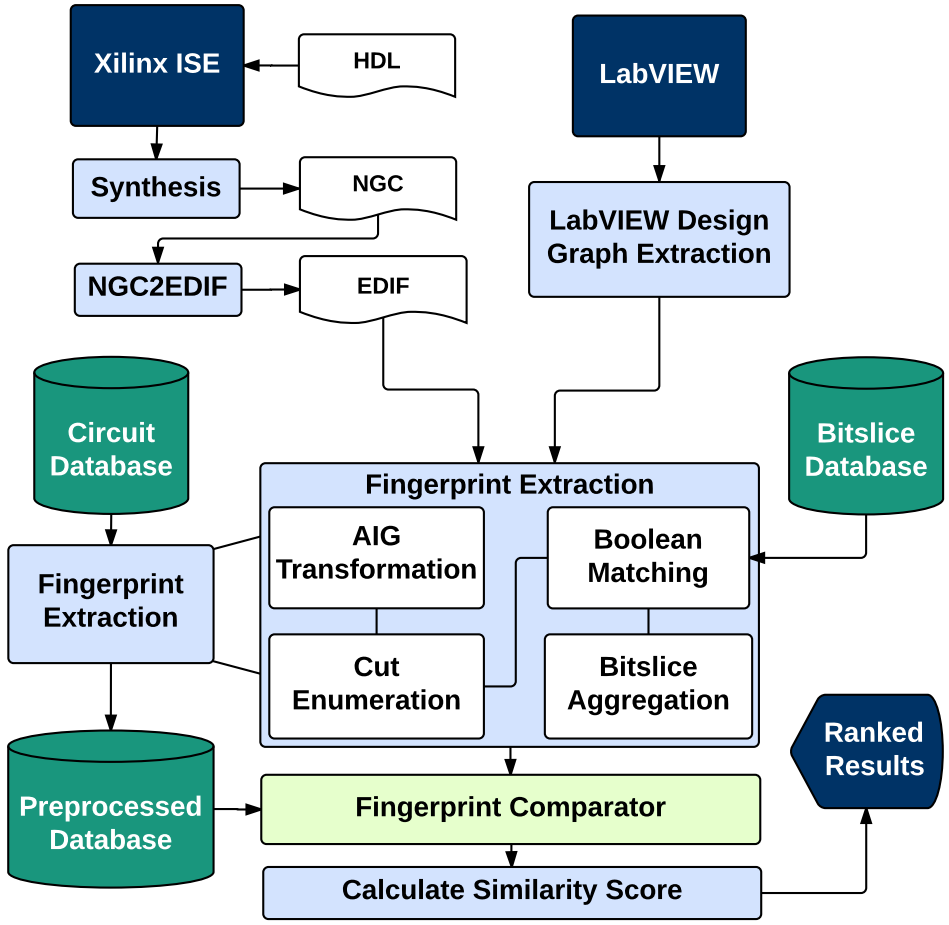


Figure 4.7: High-level overview of the implemented system.

where users drag, drop, and connect components together to create application designs. LabVIEW FPGA is an environment specifically for designing hardware applications. When the user enters the circuit in the design space, a snapshot of the design needs to be taken. This manual process is again due to the inability to manipulate certain features in the vendor's tool.

In a collaboration with NI to improve design productivity, access to the internal graph data structure was given. Once the snapshot is taken, the intermediate LabVIEW graph data structure is parsed and converted to an AIG. Once the complete AIG representation has been extracted the cuts are enumerated, compared with the bitslice library, and aggregated. The fingerprint is then populated with the result of the match along with the structural properties of the circuit. The results of the match are then listed on a separate open interface in a ranked order. By selecting the design, the layout of an existing design is shown, allowing the user to reuse the component.

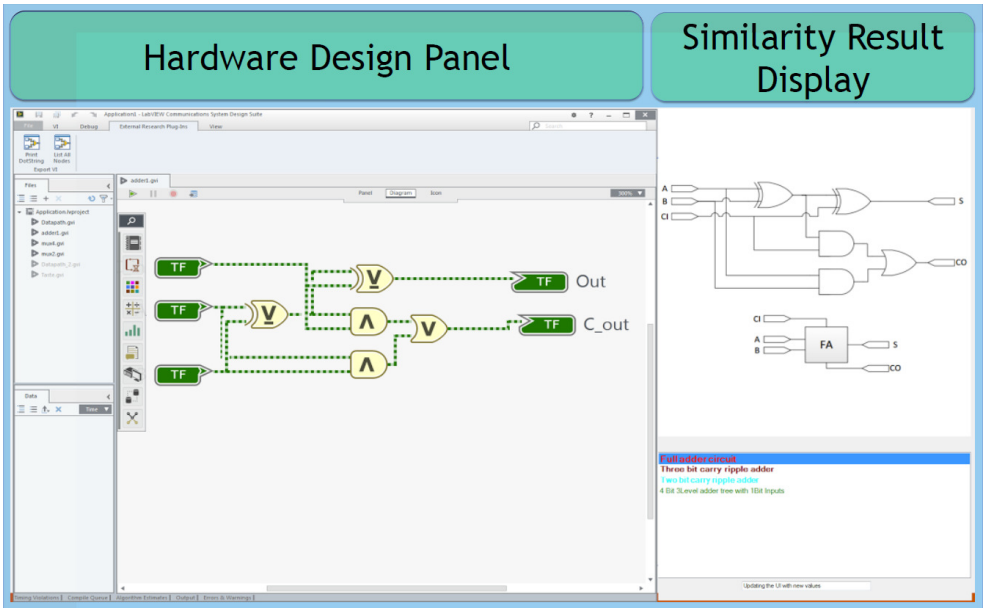


Figure 4.8: Implemented system into an existing design environment, leveraging LabVIEW front-end with a simple interface on the side to display the results.

4.5 Result and Analysis

The key problems with graph-based approaches is the lack of methods for functional comparisons as well as the expensive requirements of computational resources. As a result, the molecular similarity matching approach with Boolean matching was explored. The Boolean matching aspect allowed for the matching to focus on finding functional blocks. Furthermore, the vectorization of data using a feature vector allows for extensive similarity methods to be leveraged which results requires much less computational resources.

4.5.1 Benchmark

To evaluate the fingerprinting approach, the IWLS benchmarks are used again. A bitslice library consisting of basic functions such as logic gates, adders, comparators, multiplexers, decoders, encoders, etc were added. Furthermore, structural properties such as fanout, fanin, component/gate count, levels, number of PI that fanout to POs, etc. Though this time, the larger circuits can be compared much more efficiently. Previously, circuits b14 and up were omitted from the benchmarks used in the graph-based approaches due to the large complexity of the designs.

4.5.2 Accuracy

The accuracy of the fingerprinting method is first evaluated with two circuits in the ITC'99 benchmark, circuit b14 and b15. Figure 4.9 and 4.10 show the results when each of the reference is compared against the other circuits in the ITC'99 benchmark. The description of each circuit can be seen in Table 4.1. From the table, it can be seen that b14 is similar

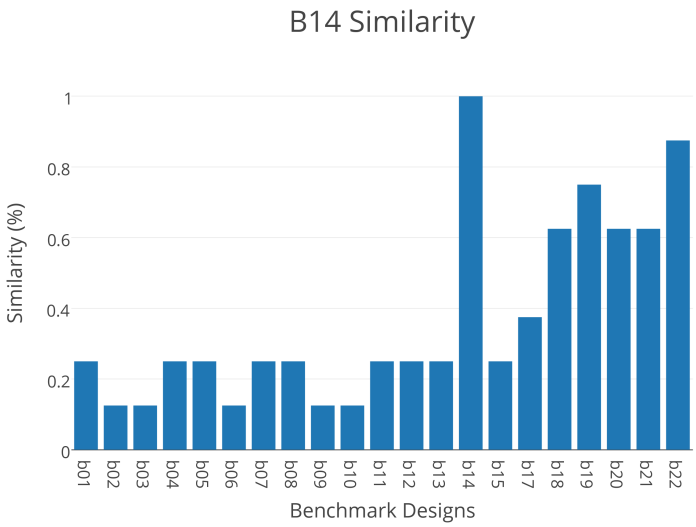


Figure 4.9: Similarity measurements comparing a b14 design against several other IWLS benchmarks.

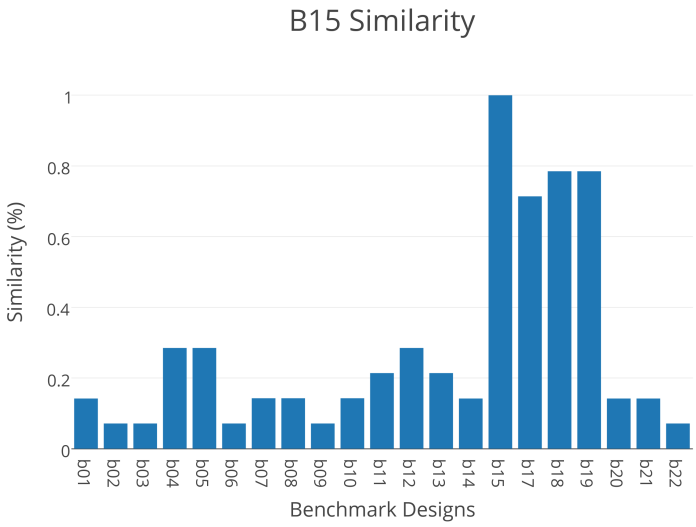


Figure 4.10: Similarity measurements comparing a b15 design against several other IWLS benchmarks.

Table 4.1: Brief Description of the ITC99 benchmarks used.

Name	Description
b12	1-player game (guess a sequence)
b13	Interface to meteo sensors
b14	Viper processor
b15	80386 processor
b16	Hard to initialize circuit
b17	Three copies of b15
b18	Two copies of b14 and two copies of b17
b19	Two copies of b14 and two copies of b17
b20	A copy of b14 and a modified version of b14
b21	Two copies of b14
b22	A copy of b14 and two modified versions of b14

to b18, b19, b20, b21, and b22 because there are multiple instances and modified versions of one in the other. The results are reflected in Figure 4.9. The same is can be seen for b15 when a high similarity score is returned for those that contain copies of b15 .

One problem with the DSI method in trying to assess the similarity is that a majority of similarity returned averaged around 85, making it difficult to differentiate between similar and different designs. From the results seen above, only designs that are similar, returned a larger similarity score. The disparity between similar and dissimilar is much larger and can be differentiated easier.

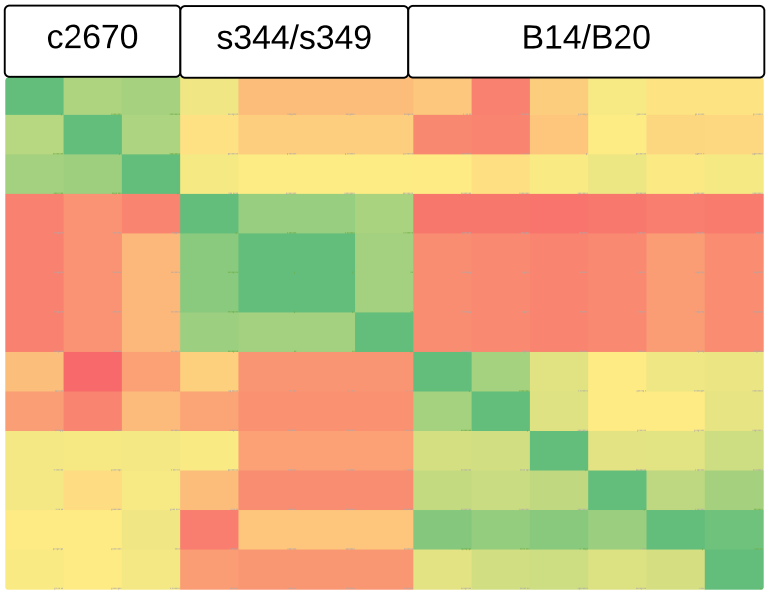


Figure 4.11: Heat map showing the similarity results of three sets of designs.

Figure 4.11 shows a heatmap of three sets of designs, where each set have had their netlists modified. The c2670 contains the original EDIF netlist and both optimized and unoptimized versions of EDIF after running the Verilog HDL through ISE. The s444 contains the same as c2670 except it is also compared with s449 which are implementations of an add-shift multiply. The last six represents b14 and b20 with the same variations as c2670. From the heatmap, a clear distinction can be seen between the three different sets of designs.

4.5.3 Performance

Performance of the fingerprint extraction from the circuit is shown in Figure 4.12. Designs with less than 500 AIG nodes can typically be processed within ten seconds, though larger designs still take a significant amount of time to process. Most of the time is spent during the calculations for the cut function. Determining NPN equivalence is difficult due to variable input ordering. There are heuristics that can be applied to try and reduce the search space such as input symmetry and permutation independent Boolean matching. Though computationally intensive, the results in terms of having greater disparity between similar and dissimilar can be seen using fingerprinting in the heatmap. Furthermore, functionality of the design was able to be captured as well.

In terms of searching for similar designs, each fingerprint will have to be compared with the others to see how similar they are. Since each fingerprint is a fixed vector that can be represented in Euclidean space, the problem becomes k-nearest neighbors where the top k results that are closest to the reference are returned.

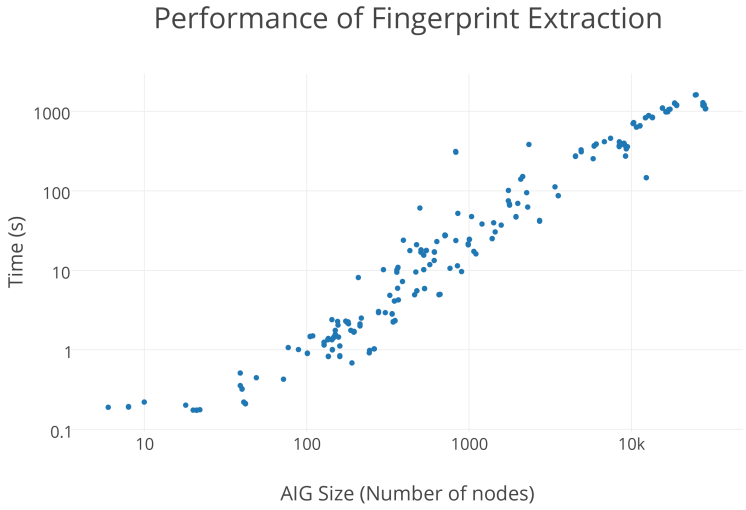


Figure 4.12: Execution time in extracting the fingerprint from a netlist.

4.5.4 Limitations

Even though extracting bitslices from a circuit allows for a more flexible way of extracting functional information from a design, the sheer size of the netlist again becomes a limiting factor. For each node, there are a number of cuts, typically around 20 or so for $k = 6$. One of the largest circuits in the ITC'99 benchmark contains over ten thousand primitive nodes where the extraction takes well over a minute. Scaling to more complex and practical circuits will constitute logic that is much larger making it difficult for use in real-time applications.

There are limitations to the components and blocks that the user is able to use. If the user inserts a block that cannot be converted to an AIG, then the functional extraction cannot be performed. One of the main difficulty is selecting the necessary features that will represent the design and which feature will give better results.

Side Inputs

Not all functionality can be easily extracted from the netlist. Many synthesis tools perform optimizations that combines common nodes to reduce area and increase performance. As a result, a cut of the node of interest may be influenced by an additional side input. There is no feasible cut such that the function at the node matches any of the inputs. QBF solvers can be used to find the values of the side inputs such that the cut and the bitslice are equivalent.

K-Cut Limitation

The size of the cut is limited to six, meaning the bitslice library contains of functions with no more than six inputs. In order to find larger functionality, bitslice aggregation is needed. Furthermore, the truth table approach works well due to the fact that the entire table can fit in a 64-bit unsigned integer. Therefore, comparing functions is just comparing the truth table; however as the inputs grow, the number of bits needed to represent the truth table doubles with each input.

NPN Equivalence

In order to make sure that the functionality is captured, the cut is tested to see if it is NPN Equivalent to the bitslice. However, computing NPN equivalence is expensive since for each cut, the truth table needs to be calculated by varying the input order and negating the inputs and outputs. Otherwise, if one of the inputs is negated, the truth table will change, even though the core logic remains the same.

4.6 Summary

This chapter explored the field of molecular similarity matching and Boolean matching in order to leverage ideas and concepts to compare the similarity of circuits. This method allows the functional components to be captured as well as certain structural properties.

Implementation details were given, outlining the implemented system for creating a fingerprint of a circuit. Functional extraction using Boolean matching techniques are expensive primarily because of the calculation of cut functions within the same NPN class which prevents it from scaling to larger designs. The next chapter suggests an alternative for scaling to more complex designs while improving upon the accuracy of the results.

Chapter 5

Hardware Birthmarking Approach

Many of the previous approaches suffered from scalability even with heuristics in place. A fairly simple design can result in tens of thousands of primitive components in the netlist. The extraction process needs to be simplified such that the necessary data can be obtained efficiently. Therefore, a new direction was needed that stepped away from what the previous approaches had in common, the netlist.

5.1 Problems with Netlists

The netlist level contains an extremely large amount of data, even more so for complex designs. To make sense of the data within the netlist requires a large amount of processing. For example, it was found to be very difficult to extract even an adder of arbitrary size from any netlist. Additionally, as the data-width of the adder increases, the structure of the circuit gets more complex even though the overall functionality remains the same. On the other hand, many of these low-level structures exist in very large and complex designs. Finding such low-level structures in a circuit does not significantly differentiate between one circuit or the other, especially when the design is large. Many designs contain adders, if not all. Furthermore, the design entry point is typically at a much higher level of abstraction, either in HDL or HLS languages. Synthesizing the design down to the netlist and then trying to go back up ends up being redundant and unnecessary when the existence of an adder is already known. Additionally, multiple levels of processing is needed: from HDL to NGC to EDIF to AIG. Because of the complexity and the sheer size of data in the netlist, the abstraction layer was moved up.

With the raise of abstraction, a new representation was needed in order to describe the circuit. Graphs were too computationally expensive to use once the circuit becomes large. The fingerprinting idea can still be leveraged, though it needs to be expressive and efficient enough to capture more useful features of large circuit design. Many of these same problems

for similarity comparison exist when trying to compare the similarity of software programs. Though fundamentally different, there are resemblances between software and hardware that can be exploited.

5.2 Software Birthmarks

For comparing similarity of software, birthmarks are the typical approach used when representing a piece of software [99]; however, the term birthmark is used very loosely. The goal of birthmarks is to determine a representation that captures the inherent characteristics that is innate to a particular program. Birthmarks for software can refer to the executable binary, control flow diagrams or dataflow diagrams, sequence of instructions, system calls, etc. The type of representation used depends on the available resources and the specific application for which the similarity of two programs is needed. Depending on which representation is used, an appropriate similarity metric is chosen. Then, by comparing the birthmarks directly, the similarity of the design can be assessed. This same concept is applied for hardware designs. The published work related to this section is in [100].

Birthmarks, in general, are formally defined as follows:

Definition 5.2.1. *Let x and y be two designs. b is a birthmark of x and y, $b(x)$ and $b(y)$, if the following properties hold true.*

1. *if $b(x)$ is a birthmark of x, $b(x)$ is obtained from only x itself.*
2. *if y is a copy of x, then $b(x) == b(y)$.*

The first condition of Definition 5.2.1 states that the birthmark is derived from only the design itself and no additional information is needed or included specifically with the design for the birthmark to work. This is ideal since no intervention from the user is required. The second condition guarantees that if two designs are the same, then the birthmark is the same; however, if two birthmarks are the same, this does not necessarily mean that the two designs are the same. It signifies that it is highly likely that the two designs are copies.

5.3 Hardware Birthmark

This same concept is extended for hardware designs. A *hardware birthmark* is defined here as a representation that characterizes the overall structure and functionality inherent to a specific circuit. The similarity of the circuit can be assessed by directly comparing the birthmark themselves.

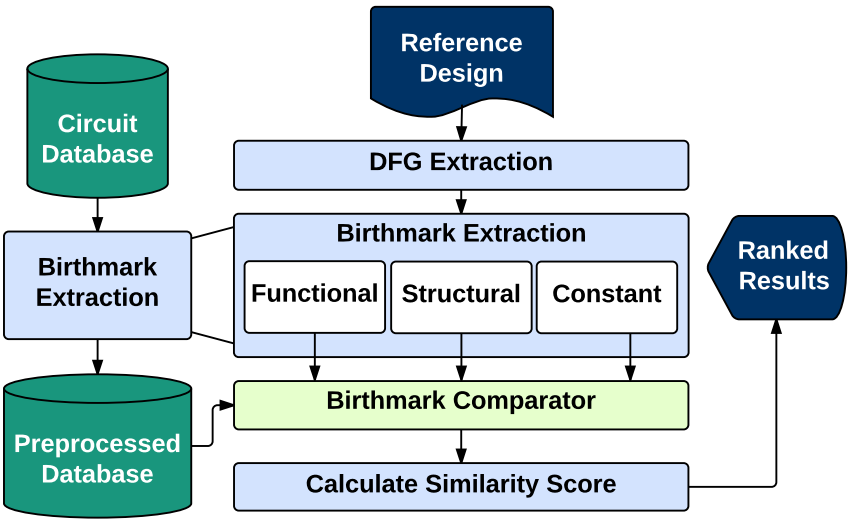


Figure 5.1: System overview utilizing a 3 component birthmarking scheme, capturing functional, structural, and constant data.

5.3.1 System Overview

The overall system utilizing the birthmarking approach is seen in Figure 5.1. Starting from a reference design, the dataflow graph (DFG) of the circuit is extracted. A textual design entry, typically written in a HDL, such as Verilog or VHDL, is used but can be extended to various design entry platforms such as graphical entry tools. 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 DFG on the other hand contains a compact and descriptive representation of the overall design. Once the DFG is extracted, the different components of the birthmark, defined in the following section, can be constructed. The birthmark is passed to a central server and compares it against the birthmarks of existing designs in order 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. Given that this process complements an interactive design session, the dynamics of this has to be highly responsive.

5.4 Datapath Hardware Birthmark

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.

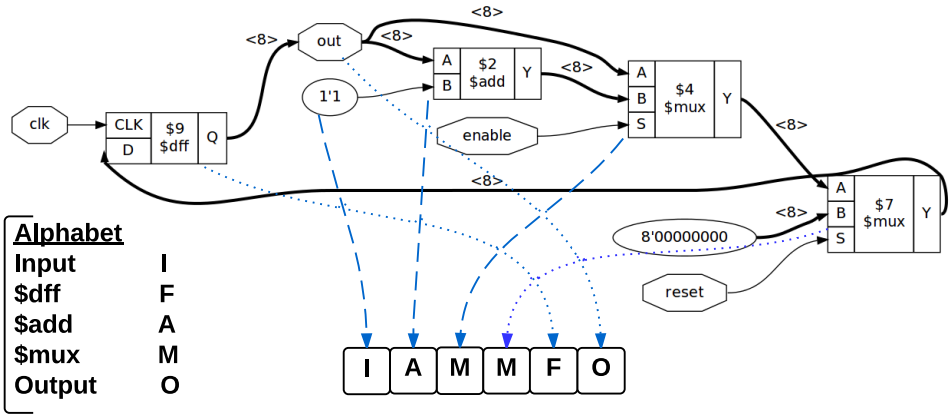


Figure 5.2: Datapath is extracted from the dataflow from an input port to an output port. Each operation is assigned a letter in an alphabet to for a sequence.

5.4.1 Functional Component

The first component of the birthmark extracts identifying features from the DFG D of a circuit C by looking at specific datapaths. A dataflow is a directed graph $D(V, E)$ of C, with a set of inputs $I \subset V$ and outputs $O \subset V$. The dataflow of an 8-bit up counter can be seen in Figure 5.2. 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 P is a path, $\{v_1...v_m\}$, in D where $v_1 \in I$ is the source node and $v_m \in O$ is the sink node. If no input or output exists in the design, nodes with no inputs or outputs are treated as source and sink respectively. Each $v_i \in P$ is assigned a specific letter in a given alphabet that corresponds to the type of operation. For example, in Figure 5.2, 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 (P_{max}), minimum length path (P_{min}), and the longest path with the highest entropy, or greatest number of unique components, (P_{alpha}). The longest captures the most information in the design in a single sequence. Yet, this sequence may not be informative about the functionality of the design. Therefore, the longest datapath with the highest entropy is calculated as well. The shortest path was added as another data point to compare. The functional component of the birthmark is formally defined as follows:

Definition 5.4.1. The functional component F of the birthmark B is

$$B_F(C) = \{P_{max}, P_{min}, P_{alpha}\}$$

where P_{max} is the sequence of the longest datapath, P_{min} is the sequence of the shortest

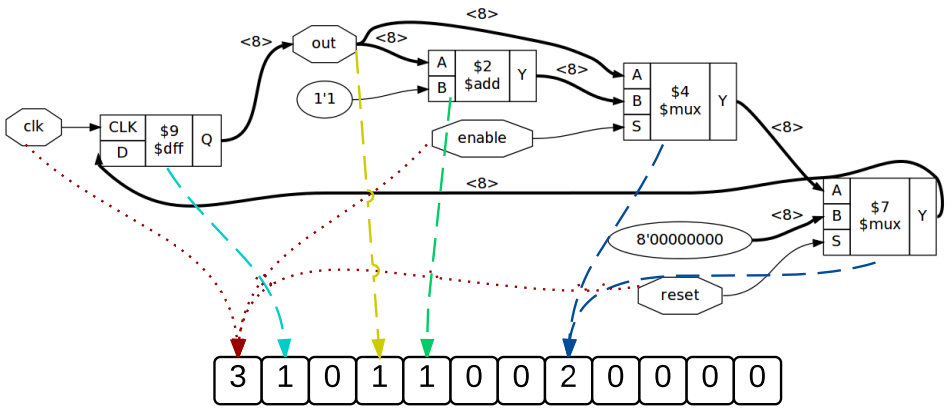


Figure 5.3: The figure shows how the fingerprinting process works. Each position in the vector indicates the number of occurrence of a specific subcomponent.

datapath, and P_{alpha} is the longest sequence with the greatest number of unique components in a datapath.

5.4.2 Structural Component

The second component corresponds to the structural aspect of the circuit and reuses concepts explored in molecular similarity matching and fingerprinting. The first step is to define the fingerprint for a circuit C. A fingerprint FP is defined here as a vector where each index i of FP represents a predefined subcomponent, $\{sc_1...sc_m\}$ where $sc_i \in FP$, $i = 1, 2, ..., m$. In order to capture the number of occurrences, a generic vector is used where $F(C)_i$ is the 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 as well as structural features of the circuit such as average fanin or fanout of components. Figure 5.3 shows the fingerprinting process of an 8-bit counter.

Definition 5.4.2. The structural component S of B is

$$B_S(C) = FP(C) = \{\#sc_1, \#sc_2, ..., \#sc_m\}.$$

5.4.3 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

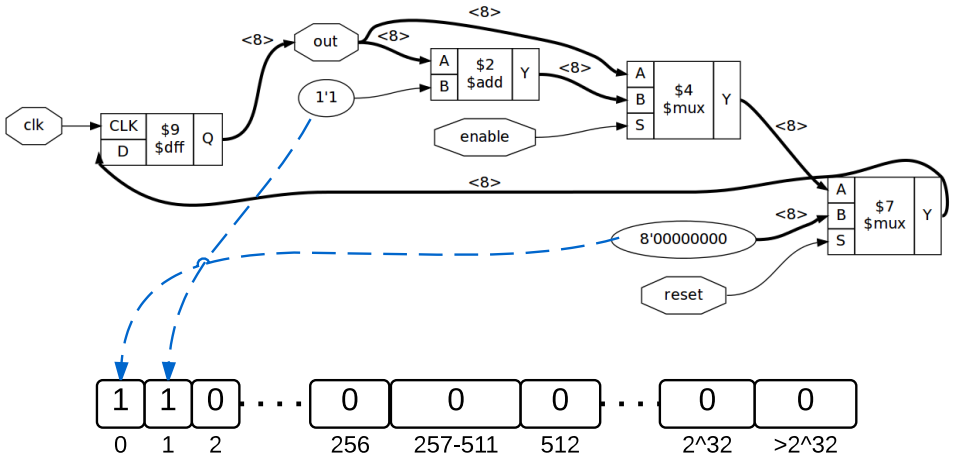


Figure 5.4: Extraction of the constant component. The figure also shows the basic layout of the binning approach.

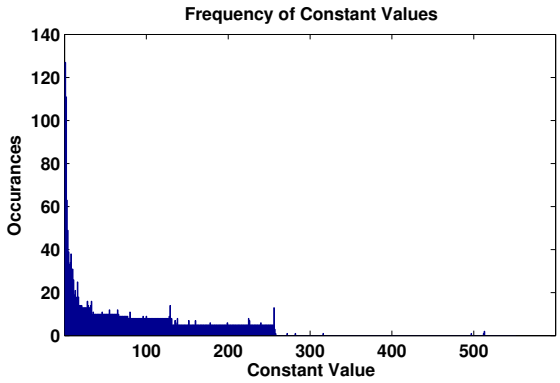


Figure 5.5: Breakdown of occurrences of constants in 151 Verilog files.

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 5.5 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 5.5 suggest 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 well as high-impedance inputs. Figure 5.4 shows the extraction of the constants as well as the layout of the bins used.

Definition 5.4.3. The constant component $Cnst$ of B is

$$B_{Cnst}(C) = \{b_1, ...b_n\}, k_i \in b_i$$

where b_i is the ith bin and k_i is a constant found in C.

5.5 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.

5.5.1 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 dynamic programming sequence alignment technique such as Smith-Waterman [101]. Sequence alignment methods are commonly used in bioinformatics as a way to find common regions or mutations in biological sequences. 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 uncommon have a higher score when matched such as lookup 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 also 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. Even with heuristics in place, the complexity can become an issue for real-time suggestion. Computing sequence alignment using 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 such that errors the user made during the design phase or subtle differences based on individual design choices can be accounted for. The score that the Smith-Waterman algorithm returns when comparing the two sequences is used as the functional similarity metric between two circuits.

5.5.2 Structural and Constant Similarity

The structural and constant components of the birthmark are of the same representation where they enumerate the different structural features and characteristics of a circuit. Both representations uses a fixed-vector size. Since the structural portion of the birthmark leverages concepts from fingerprinting in molecular similarity matching, the same Euclidean distance metric is used to calculate the similarity score.

5.5.3 Similarity Score

The scores of the different components are combined into a single score such that they can be ranked easily; however, the scale of the scores is different. The Euclidean metric that the structural and constant components use is based on the distance between two fingerprints where the greater 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 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 normalized score of the different components together. Many of the features and components can be weighted such that the more important component has a larger influence.

5.6 Implementation

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 a database, and a list of similar circuits in ranked order is produced. The implemented system overview can be seen in Figure 5.6.

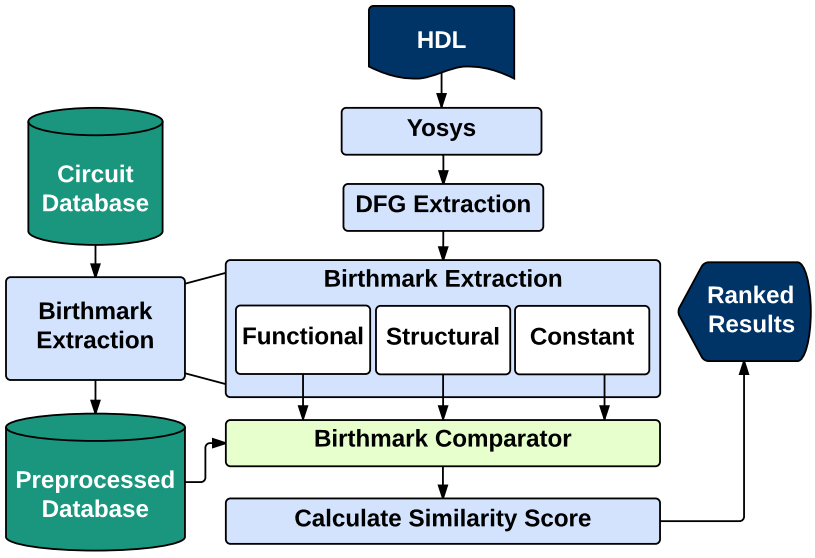


Figure 5.6: High-level system implementation of the reuse flow using birthmarks.

5.6.1 Front End Interface

For this implementation, the front-end tool was disregarded due to the inability to access and manipulate design data efficiently. For example, for Xilinx ISE, it was difficult to create an integrated design environment that is truly automated. The user still has be able to synthesize the design in order for the netlist to be extracted. Furthermore, at this point, the interest is no longer in the netlist. Therefore, getting intermediate data in terms of dataflow requires alternative methods.

As for LabVIEW, the internal representation was made available. However, to effectively test our approach, a large number of LabVIEW designs were needed. Even though traditional designs leveraging common primitive functions can be used alongside with Verilog designs, such as add and logic operators, LabVIEW also contains a number of constructs and functional blocks that are typically used to build systems and designs. For example, they have graphical loop structures to parallelize different areas of the design and case structures to handle conditional operations, which makes it difficult to compare against existing designs from other tools.

Having mentioned certain limitations, this is only due to the fact that internal access is limited as well as the data required. For such a system to work well, the database of existing designs needs to be plentiful and diverse. Verilog designs are plentiful and if given access to the tool, these methods can easily be integrated. As such, open-source tools are utilized to design a front end interface that demonstrated the proposed reuse methodology. The interface will be described further in chapter 7.

5.6.2 Extracting Birthmarks

The flow uses an open-source synthesis tool called Yosys [80] to extract the DFG from given an HDL design. For now, Yosys only supports Verilog designs. Though VHDL to Verilog converters can be used, the translation is not always successful. Even then, manual inspection of the translated product is highly encouraged. Yosys exports the dataflow graph as a DOT file which is then processed using a Python graph library called NetworkX such that the datapaths of interest can be extracted.

Functional Path Extraction

There are three main paths that need to be extracted: longest, shortest, and the longest path with the highest entropy. Entropy measures the uniqueness and predictability of information, in this case, the datapath sequence.

Each extract is done using a depth-first search (DFS) from source to sink. From the starting node, traverse the DFG until all possible sinks have been visited. However, DFS has a chance

of missing the longest path due to visited nodes. Therefore, during traversal, the path from a visited node to a sink node is recorded such that if the sum of the current path length and the length from the visited node is longer than the longest path, then the longest path is replaced. This method is replicated for shortest path and highest entropy.

One key thing to note is that the dataflow graph that Yosys produces contains some nodes that are used during intermediate processing. Therefore, these nodes are omitted and do not increase the datapath length. Once the functional paths are extracted, the similarity of the datapaths can be compared.

5.6.3 Similarity Calculation

The nodes in the DFG represents primitive operations such as multiplexers, adders, multipliers, RAM, etc. These node types form the basis of what each index represents in the structural component of the birthmark and what makes up the alphabet in the sequences extracted.

Each of these operations are assigned a letter in the alphabet. The functional paths extracted have their nodes replaced by an alphabet representing the operation. This creates a sequence that represents the datapath. Once the sequence is acquired, the corresponding sequences are aligned between two circuits. The Seqan library is used for pairwise sequence alignment of the functional birthmark.

5.6.4 Comparing Designs

A database of designs are preprocessed and stored in a XML database consisting of each circuit and their birthmark. This is all done offline ahead of time. New entries and designs can then be added incrementally.

When comparing designs, the snapshot is taken from the HDL. The process to extract the birthmark is executed and once the birthmark is extracted, it is compared against each design in the database one at a time. The fact that sequence alignment is used makes it difficult to try and reduce the search space. The outcome of the similarity is unknown until the sequence is compared. As such, taxonomy information could be leveraged where databases used are separated in smaller one that target a specific domain.

5.7 Result and Analysis

To evaluate the performance of the birthmarking approach, a database of circuits is constructed containing real-world circuit designs in order to show the feasibility of this ap-

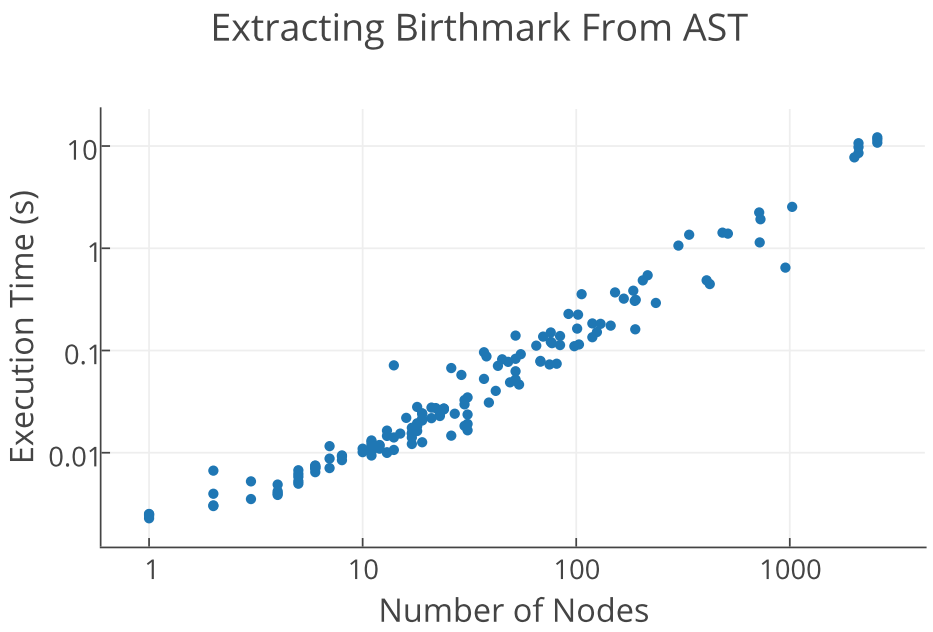


Figure 5.7: Performance of extracting a birthmark from a Verilog file. Execution time includes synthesis from Yosys and the proposed tools for extracting the birthmark from the DFG.

proach. These circuits are comprised from different sources including OpenCores, GitHub, [102], [103], [104], [105], and ranges from communications interfaces, such as UART and SPI, to filters and FFTs, totaling 151 different circuits for initial testing purposes.

5.7.1 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 are no longer meaningful to the designer. Real-time feedback whenever the design changes is desired. The time users typically will wait ranges anywhere from two seconds to no more than 30 seconds but depends heavily on the application[106].

In the case of reuse, the idea is that the user is in the design phase and will not necessarily be waiting for a response. The tool is not to be intrusive and allow the designer to remain focused at the task at hand. Yet, in certain cases, the user may be interested to see suggestions provided by the tool. Previous approaches suffered when performance was considered. Many graph-based approaches were NP-complete. Furthermore, determining NPN equivalence and scaling to larger functions in the bitslice library proved to be a significant drawback.

Birthmark Extraction

Figure 5.7 shows the total execution time to extract a birthmark from a Verilog design. The Yosys tool handles all the DFG extractions from the Verilog designs. Afterwards, the birthmark is extracted from the DFG 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 a reasonable amount of time to take for extraction. Though, the numbers in the graph does not consider the extraction of the DFG from HDL using Yosys. Larger and more complex circuits can be addressed by looking for matches of the submodules in the lower levels of the hierarchy.

Yosys is currently used to extract the DFG and it does so by synthesizing the Verilog design down depending on specific options set. For example, multidimensional registers such as n-bit shift registers, $n > 1$, are represented by a $mem node by default instead of cascading n-bit registers. The `memory` command expands this and maps the $mem nodes into basic cells. Likewise, if another tool is used to extract the DFG, similar operations needs to be performed such that the representation can remain compatible. The node type in the DFG extracted by Yosys are very basic operations such as shift, multiply, add, logic operators etc, and should be common with most synthesis tools. When Vivado performs synthesis, the report shows the amount of adders, multiplexers, etc, that it was able to infer.

Like most synthesis tools, it is necessary to prevent the tools from optimizing important logic away. For a design that is not complete, performing any sort of global optimization could potentially optimize the entire design away. For example, if the user has not connected the design to any output port, the $opt command sees that this does not contribute any functional purpose to the design and removes the entire design. For the proposed application, the incomplete design is still of interest and is needed in order to suggest reusable components.

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 5.8. 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

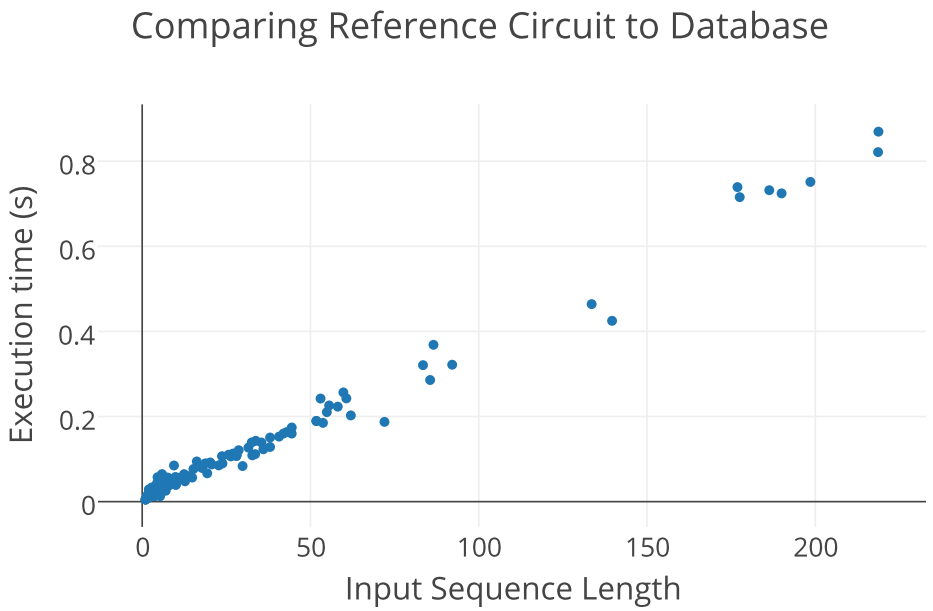


Figure 5.8: Performance of comparing each circuit in the database to the database. Performance is dependent on the varying sequence length of the functional component.

phase.

5.7.2 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 5.1 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 5.1 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 resemble 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 approach is robust against similarly designed modules using

Table 5.1: 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 [103]
82.22	rtfSimpleUartRx*	86.69	cf_fft_256_8*	80.34	firfilter	90.02	ncsu_regfile [108]
81.44	tiny_spi*	48.64	cf_fft_512_18*	78.86	qmult*	88.08	ram_sp_ar_sw [103]
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 [109]	28.51	IIR6sos_18bit_fp	71.29	qmult*	79.58	altera_ram_dual
74.97	uart_rx_only2 [110]	28.21	dft_4_4_strm_dt [111]	71.28	mult_piped_8x8_2sC [104]	79.27	ncsu_fifo [108]
74.79	uart_simple [103]	27.33	qdiv*	71.28	mult_para_rc_8x8_2sC [104]	77.07	altera_single_port_ram

The IIR and CIC filters are examples from [105]. Circuits with the * beside them are from the OpenCores database.
The circuits with the altera heading are from [102]. The rest are circuits that had been manually designed.

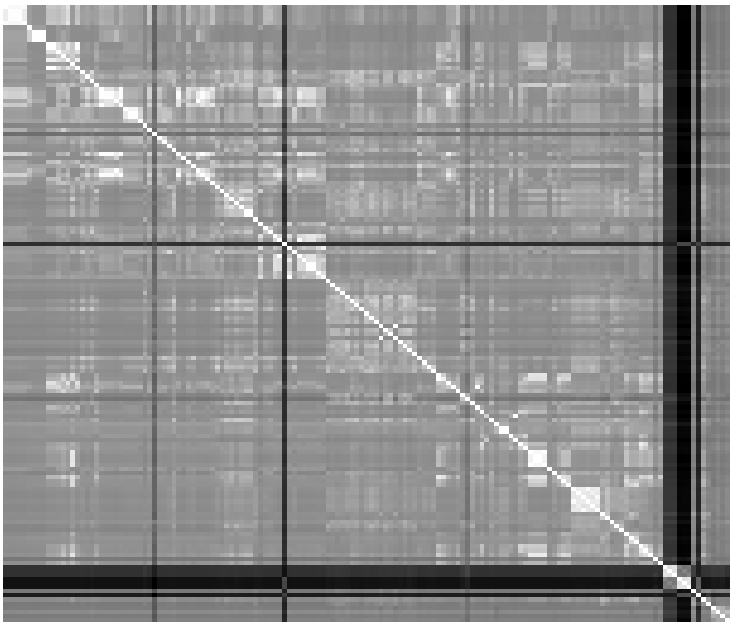


Figure 5.9: Heatmap extracted by autocorrelating the circuits in the database. The dark regions indicates dissimilarity and the bright regions indicates similarity.

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 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. To quantify and evaluate the performance of the ranking system, a mean average precision (MAP) was used to assess how well the birthmarking model finds relevant circuits. MAP is a typical measure used to evaluate the performance of information retrieval applications [107]. The MAP of the results provided in Table 5.1 with a precision of 10 is 0.697.

In Figure 5.9, a heatmap was extracted by autocorrelating the database of existing designs. The darker the regions are the less similar two circuits are and the brighter the region, the more similar the circuits are. Since the circuits are grouped 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.

5.7.3 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 interclass 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 [112]. 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 DFG extracted from a HDL design depends on the design choices the designer makes. If the Verilog module is designed using a behavioral syntax, the DFG 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, the nature of both designs are more likely to exist so that when the designer compares the reference to the circuits in the database, there will likely be a close match.

5.8 Summary

This chapter introduced hardware birthmarks as a way to expand and handle larger and more complex designs. Furthermore, the netlist representation was abandoned due to the amount of data that needs to be processed.

Details were given outlining the implemented system for extracting and comparing birthmarks using datapaths. Results show real world circuits used with the ability to process many of them quickly as well as producing very promising results during matching. The next chapter will propose an new birthmarking scheme leveraging the concept of n-grams in

order to scale to larger databases and designs as well as introducing further improvements and possibility in extracting information for reuse.

Chapter 6

Q-gram Birthmarking Approach

One limitation of the birthmarking approach specified in the previous section is that the functional component does not capture all the features of the dataflow. By looking at three different datapaths, much of the circuit may not be covered, especially if the circuit has large fanouts. To remedy this, a path-based q-gram approach is used as the new birthmark representation. The system overview using the q-gram birthmark is depicted in Figure 6.1. Much of the flow is the same as the previous approach with the exception of the birthmarking scheme used and the methods for comparing the birthmarks. The published work related to this section is in [113].

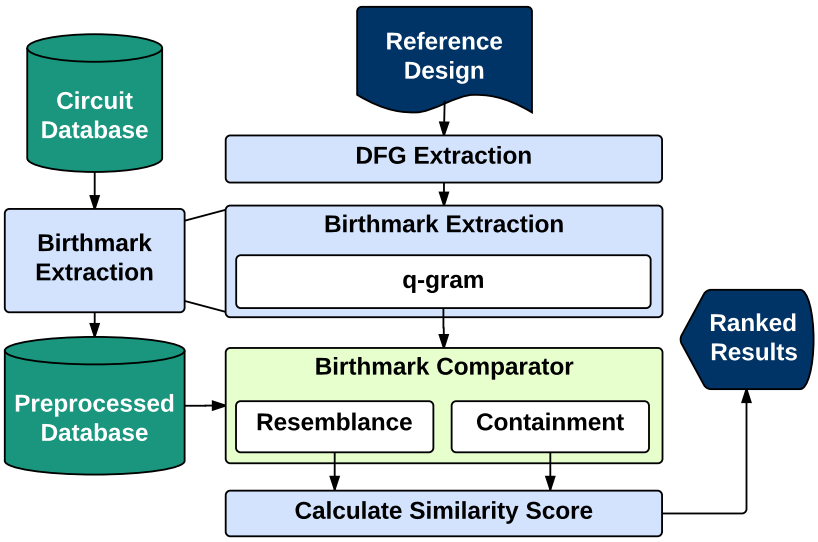


Figure 6.1: System overview leveraging q-grams as the functional component of the birthmarking approach.

6.1 Q-grams

A q-gram (n-gram) approach is commonly used in natural language processing (NLP) when trying to assess the similarity of two documents. The text of each document is decomposed into tokens of q length using a sliding window approach. This results in a model that represents the overall document. Similarity between two documents can be determined based on the number of shared tokens found between the two models. For example, Moss [114] is a common tool used by professors for detecting plagiarism of coding assignments submitted by students and leverages q-grams to determine the similarity of programs; however, Moss decomposes the source code into q-grams. This includes naming conventions used and relies much on the layout and format of the source code in hand. Though this is useful if the student copies and pastes the code directly; however, making semantic preserving transformations can easily undermine the usefulness of Moss.

6.1.1 Graph Similarity Search Utilizing Q-grams

Instead of deriving the q-gram model for a given design based on the source, the DFG is again extracted from the HDL. This DFG is used when processing the q-grams. The idea of q-grams works well with sequential data such as text in a document. For non-sequential data such as graphs, q-gram can be extended by using a path-based q-gram approach [115]. Utilizing q-grams allows for a more efficient approach for approximating similarity of graphs compared to the methods described in the graph-based approach in chapter 3.

Graph Edit Distance

The q-gram approach has recently been an attractive form of graph similarity searching based on graph edit distance. Graph edit distance (GED) is the number of edits that need to be made in order to transform one graph to another. Several different definitions for q-grams exists such as path-based [115], branch-based [116], tree-based [117], and star-based [118]. Each approach is compared typically by how tight their lower bound is. The lower bound of these approaches represents the number of q-grams affected when an edit operation happens.

Out of the approaches mentioned the branch-based method produces the tightest bound; however, the metric GED calculates is not necessarily the interest. Though knowing how many edits are needed to convert one graph to another is a good indicator of how similar two designs are, to tell if one is contained or will become another design, GED will not be able to perform. This is because a small design that is contained in a larger design will have a large GED because of the missing logic that surrounds the component. Furthermore, a single branch does not contain interesting information since a branch represents a vertex and the labels of the edges incident to it. Note that edges in the DFG typically have no labels.

One could use the width of the bus, but then the matching will depend on the datawidth. An adder with a datawidth of eight will no longer be the same as an adder with a datawidth of 16, even though they are fundamentally equivalent.

Star and tree-based approaches have a higher lower bound compared to path-based approaches [115]. Furthermore, path-based q-grams perform better in terms of similarity searching and pruning the search space [116]. As a result, the path-based approach was chosen.

6.2 Path-Based q-grams

Instead of a sliding window of size q across a sequence, paths of length q are selected from the dataflow. Path-based q-grams are formally defined as follows:

Definition 6.2.1. Let $D(V, E)$ be the DFG that represents the circuit. A path-based q-gram birthmark of D is a collection of simple paths, or tokens, starting from $v \in V$ of length q, for all nodes in V.

An example is shown in Figure 6.2 where a simple 8-bit counter is broken down into q-grams.

6.2.1 Path-Based q-gram Schemes

The accuracy or tightness of the match can also be decided based on the representation of the q-gram. Three different schemes for path-based q-grams were explored: list-based, set-based, and frequency-based. In a list-based scheme, the ordering of each element inside the q-gram

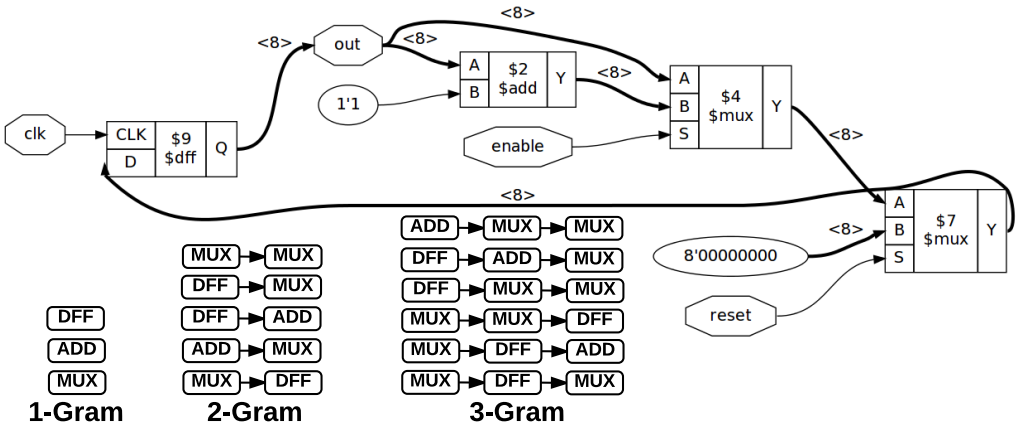


Figure 6.2: Path-based 3-gram model of basic 8-bit counter. (Q to 1)-gram is shown because it includes all the q-grams that are less than and equal to 3.

Table 6.1: Detailed description of the different birthmark schemes.

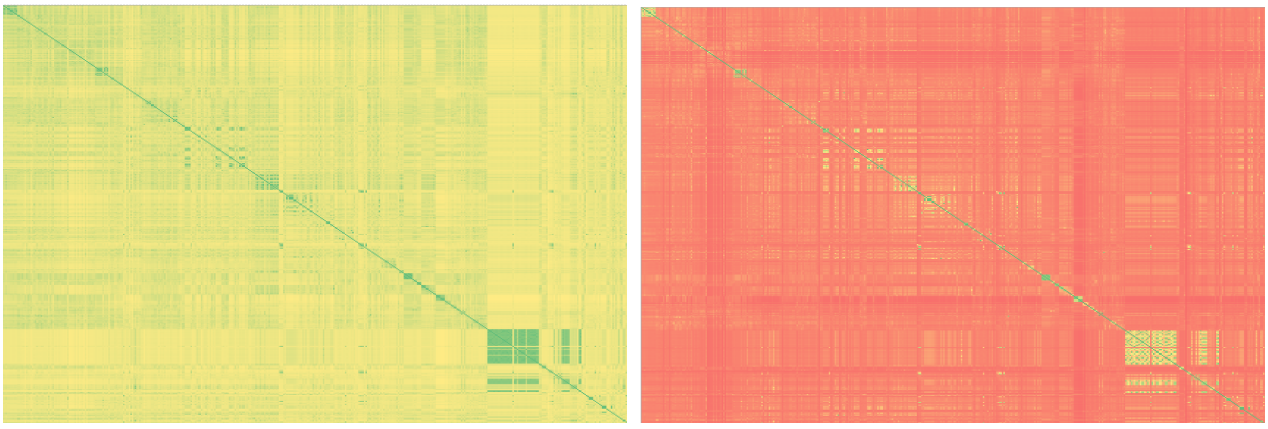
Scheme	q-gram	Description
List	AXAM	q-gram extracted from circuit
Frequency	AAMX	Ordering of elements is insignificant (sorted)
Set	AMX	Both ordering and frequency of element is ignored. Only one `A` is taken into account

is maintained. Figure 6.2 shows a strict list-based scheme where the order of the items in the q-gram is preserved. The set-based scheme is an extension of the list-based scheme where the frequency and the ordering of the elements are ignored. The frequency-based scheme maintains the frequency information but ignores the order of the elements. Examples of each scheme are detailed in Table 6.1.

Depending on the goal of the application, a different path-based q-gram scheme might be chosen. For example, the list-based scheme is useful for plagiarism and theft detection because it maintains ordering and frequency which can reduce the number of false-positives. On the other hand, if one was searching for a similar circuit in general, a looser search criteria might be desired. For the remainder of this dissertation, the frequency-based q-gram approach is used.

6.3 Determining q

The choice of q has two effects: the first is the performance of the search and the second is the accuracy of the result. This is because now for every node, there is a larger search space



(a) $q = 2$ (b) $q = 8$

Figure 6.3: Heatmap of showing the difference in results when using different values of q.

Table 6.2: Birthmarking statistics after processing 110 different circuits.

q	Unique q-grams	Average Unique per Circuit	Total q-grams
3	1405	4	243737
4	5189	20	738833
5	15098	69	1664255
6	37513	200	3435489
7	83056	496	7600598

that needs to be explored. By setting q too large, the runtime to calculate and extract the q-gram model will be time consuming.

In addition to the performance of the match, the quality of the results can be greatly affected as well. If the q is too small, then the quality of the results is quite poor. This is because most designs will contain these small paths of length one to two, such as adders going into flip-flops for logic implementing counters. On the other hand, by having a large q will restrict the search significantly such that no matches are found because no designs contains these long specific q-gram tokens.

Looking at a heat map of an autocorrelation in Figure 6.3 shows that with a low q, there will many results that can be potential matches. Even though there are still some notable sections that can be differentiated, those circuits may contain q-grams that use specific operations that are not common to the rest. For example, in Figure 6.3a, the designs in the green blob towards the lower right corner contains sequential XOR gates, typically common in cyclic redundancy check (CRC). On the other hand, if q is too big, there will be many designs that won't contain the same exact q-gram.

Table 6.2 shows the number of q-grams found after processing 110 different circuits. A small percentage of the total number of q-grams found are unique; however, many hardware designs have repeating structures in order to leverage the parallel nature of circuits. For example, cryptography modules consist of highly repetitive logic and for q=7, one q-gram from a DES birthmark contributed to over 10% of the total q-grams found. Over 70% of the circuits contain at least one q-gram that is unique to itself. The average unique column in Table 6.2 shows the average number of unique q-grams per circuit.

Note that 70% of the circuits having unique q-grams are also attributed to the selection of circuits and not necessarily a limitation of the birthmark. For example, two identical implementations of a FFT filter with different parameters will produce a nearly identical birthmark; however, many of the q-grams in the birthmark will be unique to only the two. The database also contains small circuits as well as circuits that are contained in others, which decreases the amount of circuits that contain unique q-grams. The average number of unique q-grams discovered for each circuit increases as q increases.

With the accuracy of the results and the performance taken into account, q is chosen to be

6. The data in terms of accuracy and performance can be seen later in the results section of this chapter (6.8).

6.4 Extension to *(q to 1)*-gram

Here, the notion of a q-gram is extended further to include the q-grams less than q. In other words, the q-gram model now contains tokens of length n, where $1 < n \leq q$. This model was chosen because for one, paths of length q may not exist in certain circuits. For example, the circuit in Figure 6.2 does not have a 5-gram since the longest path from any node in the circuit is 4. By taking the smaller q-gram models into account, this problem can be alleviated. Though this extension increases the lower bound because now there are more tokens that can be affected by an edit, it increases the potential for a likely candidate to not be discarded. For example, if there exists a path $AXMSX$ in one design, and in another design, there exists a path $AXFSX$, a 2-gram model will be able to capture some similarity since AX and SX will match; however, anything greater than two produce no results. Having $q = 2$ is not ideal since most of the tokens produced will be common among all designs.

6.5 Predicting Reusable Subcomponents

Predicting reusable subcomponents is an idea similar to auto complete. The q-gram model can be extended such that future elements in a token can be predicted by utilizing Markov models. The intuition behind using a Markov model is that future states can be predicted with certain probability based on what the current states are. This is typically done by recording the frequency of each token observed and using a maximum likelihood estimator (MLE) to predict the next most likely function. More formally:

Definition 6.5.1. The MLE is defined to be

$$P(w_n|w_{n-q+1}^{n-1}) = \frac{C(w_{n-q+1}^{n-1}w_n)}{C(w_{n-q+1}^{n-1})} \tag{6.1}$$

where w is the element that represents an operation in the DFG, $C(x)$ is the frequency of occurrence of token x, and n is the index of the element of interest. In other words, the MLE is the probability that the next element, given the current token, is w_n .

The idea here is, based on the components placed and connected, what is the next most likely component to be placed afterwards. The idea here is less applicable in terms of HDL; however, for graphical design environments such as LabVIEW and Vivado IPI, this approach becomes more appealing.

Consider the following example:

> *Chris is designing his circuit in Vivado IPI. He has a simple design going on and the next component he places and connects is an AXI memory interconnect. Seeing the components he's placed down, the next likely component to be placed after is suggested.*

6.6 Calculating Similarity

The similarity of the path-based q-gram approach is computed by finding how many of the tokens two circuits share in common. This is done using Jaccard's index. Jaccard's index is commonly used to compare the similarity of two sample sets [119]. Greater weight is placed on tokens that have a higher q, $1 \leq len(token) \leq q$. The modified Jaccard's index is described in Algorithm 2. The intuition is that the more tokens the circuits share, the more similar the two circuits are.

Algorithm 2 Calculating resemblance between two birthmarks

```
1: function BIRTHMARK_RESEMBLANCE(b(x), b(y))
2:     intersection = 0, union = 0
3:     for each q-gram g ∈ b(x) do
4:         union = union + len(g)
5:         if g ∈ b(y) then
6:             intersection = intersection + len(g)
7:         end if
8:     end for
9:     for each q-gram g ∈ b(y) do
10:        union = union + len(g)
11:    end for
12:    sim = intersection/(union − intersection)
13: end function
```

Algorithm 2 finds the resemblance of two circuits. This is helpful when the circuit in design is fairly close to the target design. If potential designs are to be suggested, a more flexible measure is needed in order to predict what the design might become. Therefore, a containment measure is also used. Containment indicates designs where the features of the current designs are contained in it. The procedure for containment is shown in Algorithm 3.

Algorithm 3 Calculating containment between two birthmarks

```
1: function BIRTHMARK_CONTAINMENT(b(x), b(y))
2:     intersection = 0, refSize = 0
3:     for each q-gram g ∈ b(x) do
4:         refSize = refSize + len(g)
5:         if g ∈ b(y) then
6:             intersection = intersection + len(g)
7:         end if
8:     end for
9:     sim = intersection/refSize
10: end function
```

6.7 Implementation

A circuit similarity comparator tool was developed using the methods described above in C++ and Python leveraging the same framework created in the previous chapter. A Verilog design is taken as input, comparisons are made with the reference against a database, and a list of similar circuits in ranked order is produced. The only thing different is the extraction, comparator, and the calculation of the similarity score. The implemented system overview can be seen in Figure 6.4.

6.7.1 Q-gram Extraction

Extraction of the q-grams starts with iterating through each node. At each node, a DFS is performed such that a path of q length is reached. Due to the extension used, at each step of the traversal, the token is recorded. Again, intermediate nodes with no functional purpose that is generated by Yosys are ignored. Each token retrieved is stored along with the frequency of the token seen. This creates a q-gram birthmark that represents the design. Once the birthmark is extracted, it can be directly compared with the others.

6.8 Result and Analysis

Two different approaches for the functional component were proposed. The datapath functional component tries to capture the most interesting or unique feature of the circuit but fails to capture the overall functionality of the design. On the other hand, the q-gram approach captures the majority of the design, which results in a larger search space during comparison. Performance and quality of the results needs to be balanced. Performance tests were executed on a machine with an Intel i7 and 16 GB of RAM.

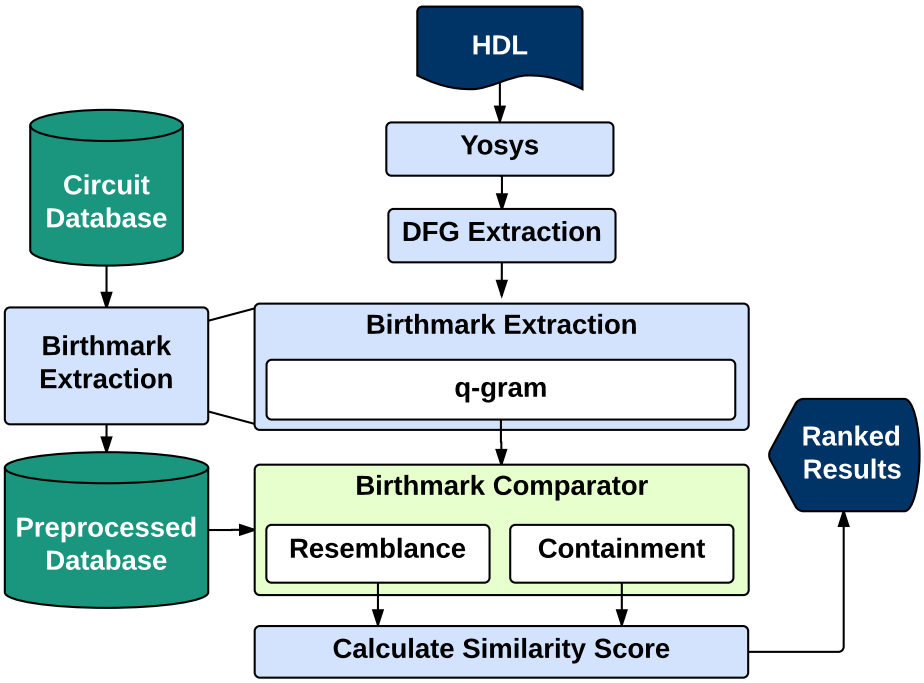


Figure 6.4: High-level system implementation of the reuse flow using the q-gram birthmarking approach.

The database used in the chapter 5 was extend further leveraging more designs from the same sources, totaling 655 different designs.

6.8.1 Finding Q

To capture more features of the design, q is increased; however, as mentioned before in Section 6.3, if q is too big, then the results can be overfitted and fail to return any match. In addition, execution time of the extraction will be greater as well.

Effect of q on Performance

Figure 6.5 analysis the execution time in extracting the birthmarks from circuits with varying levels of q. Note that the time measure for extraction does not take into account the time Yosys takes to generate the DFG. Measurement starts when the DFG is obtained to when the complete birthmark is extracted.

As seen in Figure 6.3a, even though $q = 2$ provided the quickest time, the results are not ideal in terms of finding similar circuits. As mentioned before, a reasonable time frame is

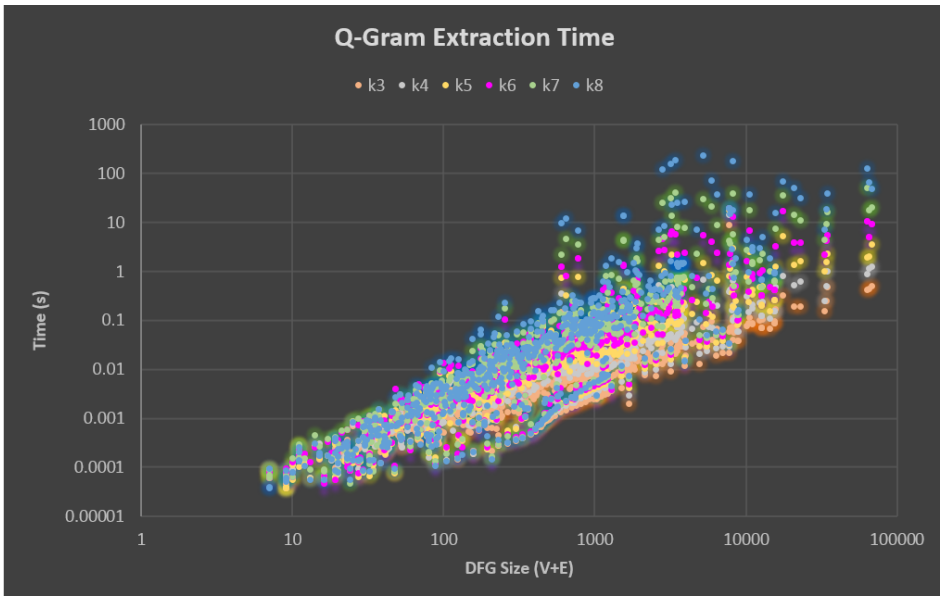


Figure 6.5: Run time of q-gram token extraction with varying q.

around ten seconds. This being said, q being set to 5 or 6 should allow plenty of time for the extraction to take place for a typical design and maintain the accuracy that is desired as seen from the results 6.8.5.

Mean Average Precision

With an upper bound set on q based on performance, the accuracy and precision of the birthmark is investigated. Figure 6.6 shows the mean average precision (MAP) of the top ten results suggested from the matcher using two different q-gram schemes while varying q. The MAP was calculated from the tables in Appendix A and B.

From the results, when q is 2, the MAP is the highest. Though MAP can be heavily affected by the queries chosen. For example, the FFT and the Sobel filter hardly changed. Yet as q increased, for the UART designs, precision increased as well, peaking at $q = 4$. On the other hand the precision of the dual port RAM and multiplier decreased as q increased as seen in Figure 6.7. This is most likely because of the depth circuit. Multipliers and RAM are not very deep and therefore shorter q will be able to capture those designs better, whereas a longer q captures the UART designs better in this case. This being said, $q = 5$ will be used for the rest of the experiments using a frequency-based q-gram scheme, unless otherwise stated.

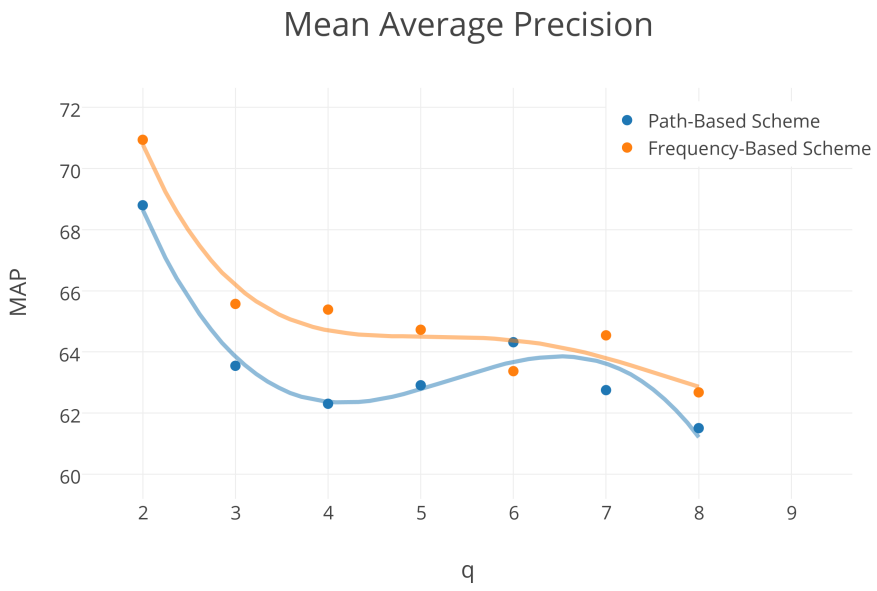


Figure 6.6: Mean average precision varying q. Path-based and frequency-based schemes were explored.

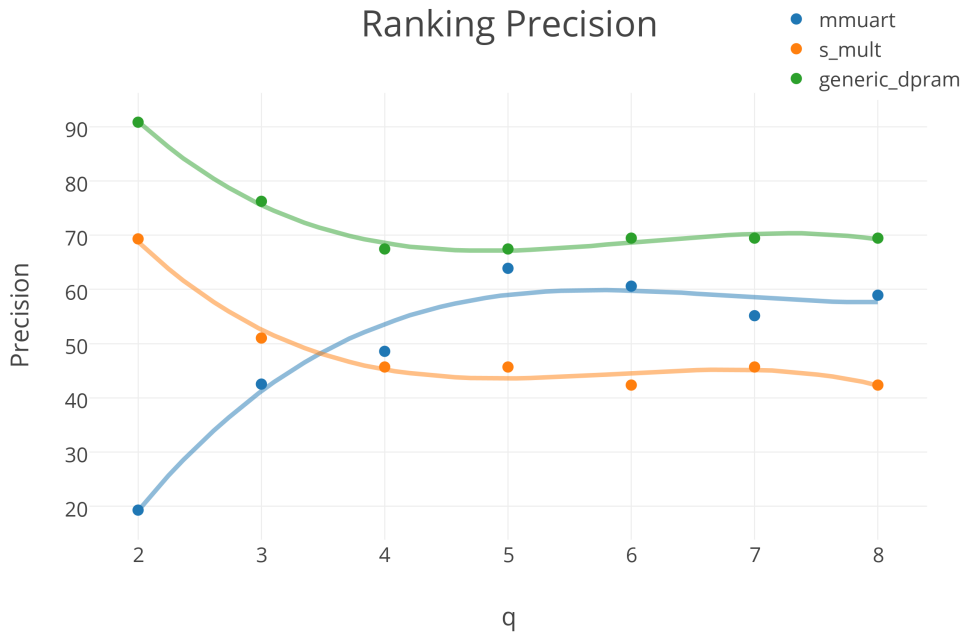


Figure 6.7: Precision varying q of three reference circuits. Frequency-based schemes was used.

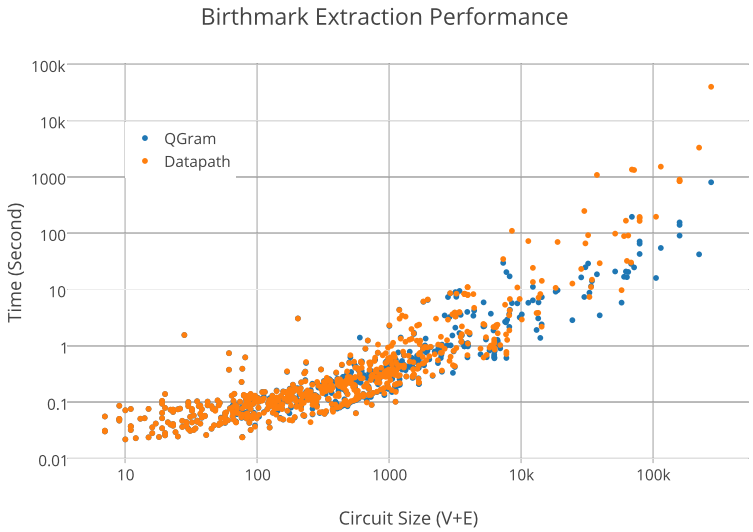


Figure 6.8: Performance of extracting a birthmark from a Verilog file. Execution time includes synthesis from Yosys and the proposed tools for extracting the birthmark from the DFG. ($q = 6$)

6.8.2 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 are no longer meaningful to the designer. Real-time feedback whenever the design changes is desired.

6.8.3 Birthmark Extraction

Though the search is very fast, the extraction of the birthmark currently has certain limitations. Like most applications, the more features that the birthmark captures, the more expensive the computation. Features such as being able to identify designs despite semantic preserving transformations and other functionally equivalent substitutes requires additional processing.

Q-gram vs Datapath Birthmark

Figure 6.8 shows the total execution time to extract a birthmark from a Verilog design for both the datapath and the q-gram birthmarks. The q-gram performs better overall. The Yosys tool handles all the DFG extractions from the circuit. Afterwards, the birthmark is extracted from the DFG that Yosys generates. For the most part, most of the designs take

about ten seconds to completely extract the birthmark. Based on the time it takes for a user to input and make changes on the design, this is a reasonable amount of time to take for extraction. Yet, execution time gets very expensive for very large circuits. Larger and more complex circuits can be addressed by looking for matches at the modular level, performing the matching at a higher level of abstraction.

6.8.4 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 6.3 presents the top ten results that the ranking system returned for several reference designs using the datapath functional birthmark. Table 6.4 presents the top-ten results for the q-gram functional birthmark.

Overall, the ranking of similar existing designs to the reference proved to be effective. Anomalies in the results can be seen and are to be expected since this is a similarity measurement. Compared with the previous approach [100], more disparity can be seen in the results due to the drastic increase of designs that the tool is searching through. Nonetheless, the results are still quite promising. With the `mmuart` reference design, the q-gram birthmark was able to pick out several designs that are similar whereas the datapath birthmark failed to identify any other `uart` designs.

The mean average precision of the datapath birthmark compared to the q-gram approach is 0.569 compared to 0.647 which indicates that the q-gram approach is able to suggest more relevant designs. Furthermore, by increasing the database size from 151 to 655, the MAP of the datapath birthmark dropped from 0.697 to 0.569. Even though the calculation included a fifth design in the MAP, the Sobel design had a precision of 0.663 which pulled the overall MAP of the datapath up.

6.8.5 Circuit to database search

Not only does the extraction of the birthmark have to be fast and efficient, finding similar birthmark has to scale efficiently as well. As the database grows, more and more designs needs to be searched. Previous approach with the datapath birthmark is linear in terms of the database size since each birthmark in the database is compared with the reference.

Results for comparing the datapath and q-gram birthmark of the reference circuit against a preprocessed database of 655 existing designs can be seen in Figure 6.9. For the datapath birthmark, the execution time is heavily dependent on the size of the sequences. On the other hand the q-gram functional component performs much better than the datapath functional component. For a linear search across the database, the datapath birthmark is typically

Table 6.3: Ranking results using select datapaths as functional component.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	arp_cache*	cf_fft_256_16*	s_altmult_add [102]	dpram_sclk [102]	prewitt [120]
2	mmuart_tx*	cf_fft_256_8*	ycbcr_to_rgb ⋄	ram_dual [102]	sobelv3[120]
3	comb_heir_fir [121]	cf_fft_512_18*	lm32_mult ⋄	ram_infer [102]	sobel_3 ⋄
4	aes128la*	cf_fft_512_16*	gfx_triangle ⋄	or1200_dpram*	sobelv2[120]
5	PIC168F4-T4 [122]	cf_fft_512_8*	synfir [121]	dram_2port ⋄	edgefilter ⋄
6	i2cSlaveTop*	Lattice_Micro ⋄	firfilter ⋄	generic_reg ⋄	edgedetect ⋄
7	PIC168F4-T3 [122]	fir[121]	firfilter4 [123]	pipeline_fft_64*	edgeDet_sobel ⋄
8	synfir[121]	usbhost*	u_mult [102]	ram_sp_sr_sw[103]	IIR2_parallel [105]
9	PIC168F4-T2 [122]	risc16f84*	u_altmult_acum [102]	single_port_ram [102]	edgedetection ⋄
10	arp64 ⋄	usbslave*	comb_heir_fir [121]	usb1_core*	IIR4_parallel [105]
MAP	19.28968254	82.28174603	49.0515873	67.99206349	66.34920635

Table 6.4: Ranking results using q-gram birthmarks as functional component.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver*	cf_fft_256_16*	altera_sig_altmult_add[102]	altera_true_dpram_sclk	prewitt
2	rc5*	cf_fft_256_8*	reset_filter	lm32_ram	sobel3
3	rtfSimpleUartTx	cf_fft_512_18*	reset_filter	ram_sp_sr_sw	sobel2
4	memtest	cf_fft_512_16*	clip_reg	or1200_dpram*	edgefilter
5	memcard	cf_fft_512_8*	altera_unsigned_mult	softusb_dpram	edgedetect
6	rtfSimpleUart*	eth_parser*	synchronizer	behave1p_mem*	sobel_3
7	buf_3to2	uart	qmult2*	altera_shift_1x64	sobel
8	buf_2to3	ip_checksum_ttl	gmii_phy_if	setting_reg	sobel_accelerator
9	comb_heir_fir	uart_transceiver*	add2_reg	generic_mem_small	add2_and_clip
10	uart_simple*	rtfSimpleUart*	clkgen	generic_mem_medium*	qmult
MAP	60.6031746	82.28174603	45.69047619	67.46428571	67.59920635

Circuits with the * beside them are from the OpenCores database. Circuits with the ⋄ were obtained from GitHub. Others are from [102], [109], [121], [120].

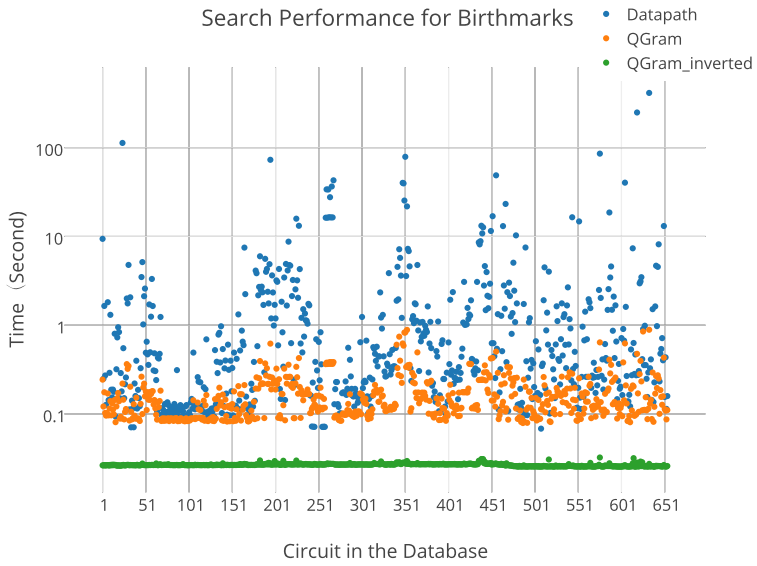


Figure 6.9: Performance of comparing each circuit in the database to the database for the datapath and q-gram birthmarks. Performance of the inverted indexing approach for q-gram birthmarks is compared as well.

$O(nmD)$ where n and m are the average length of the two datapaths and D is the database size. The q-gram model on the other hand is logarithmic, or even $O(t * D)$ if a hash table is used, where t is the number of tokens in the reference birthmark. To further improve search performance of the q-gram approach, an inverted index method was used.

Inverted Index

The inverted index approach is an additional data structure that keeps track of all the tokens that have been seen along with a list of designs that they were seen in. Then, instead of finding the set of tokens the reference has with each of the circuits, the inverted index can be used to find the circuits that contains the most tokens in common with the reference.

When processing the database, each token that is processed is put into a dictionary where the key is the token and the value is a list of designs in the database the token was found in. After the database has been processed, by searching for a token, the circuits that the token is found in can be determined. During comparison, instead of comparing the tokens of the reference against the tokens of each circuit in the database, the inverted index is used to determine which circuits contains that same token. A counter of similar tokens with each circuit is maintained. Then, by using Algorithm 2 and 3, the similarity can be determined quickly in one pass of the tokens in the reference birthmark.

Previous approach [100] for q-gram birthmarks performed the search linearly with a time

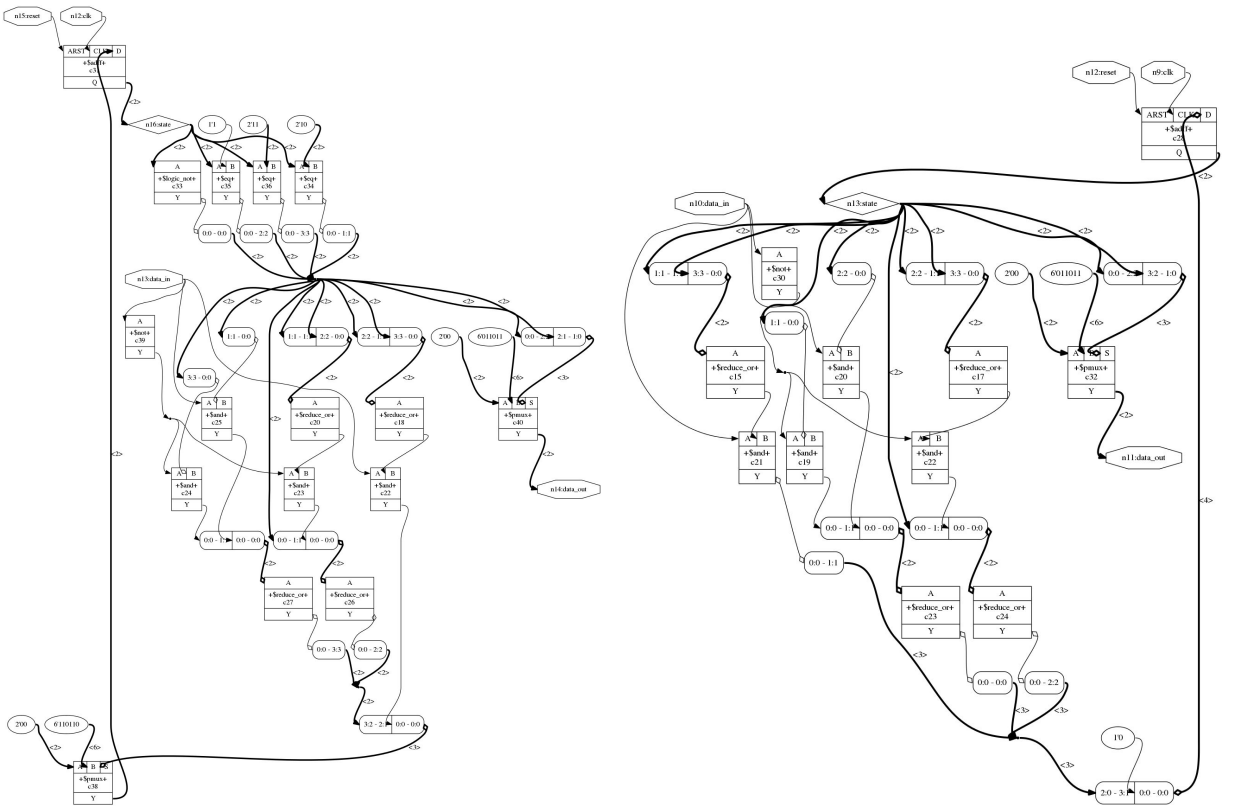
complexity of $O(rLOG(e)D)$, where r is the number of tokens in the q-gram model of the reference, e is the number of tokens in a q-gram model of the existing database design, and D is the number of designs in the database. Essentially, for each of the tokens of the reference design, check for the token in each of the circuit in the database.

The time complexity of the search with the inverted index is $O(rLOG(T)L)$, where T is the number of unique tokens in the database and L is the number of designs that the token appears in. Worse case for the inverted index should actually be worse than the linear search. This is because $L \leq N$ and $T > m$; however L should never be close to N. Otherwise, there will be no unique tokens that can be used to differentiate between designs. Out of the 655 circuits that were preprocessed, the average case for L is around 10.55, over a magnitude smaller than N. Even as N increases L will not be affected significantly and so L does not increase linearly to N. Furthermore since T and m are within a LOG operation, even if T is a couple magnitude larger, it will not affect the search significantly. The LOG operation is based on a binary search through the set of tokens. This can very much be close to constant time if hashing is used. Much of this can be seen in Figure 6.8.

6.9 Limitations

Even with the proposed birthmark approach, there are still limitations and further improvements could be made. It is important to note that the DFG extracted from a HDL design depends on the design choices the designer makes. If the Verilog module is designed using a behavioral syntax, the dataflow 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 datapaths 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, the nature of both designs are more likely to exist so that when the designer compares the reference to the circuits in the database, there will likely be a close match. Furthermore, two designs that are similar could still have very dissimilar datapaths depending on implementation. For example, different encodings for finite state machines (FSMs) will result in different dataflow graphs as seen in figures 6.10a and 6.10b for binary and one-hot encoding respectively.

Another limitation is the fact that the design needs to be synthesized first. Since the design needs to be synthesized, the Verilog design during the design phase needs to be syntactically correct. The Verilog code itself could be analyzed by using some sort of NLP; however this then becomes influenced by factors including naming, ordering of statements and syntax used. On the other hand, using a graphical design interface such as LabVIEW could better enhance the productivity of designers. In LabVIEW, designers drag components in and connect the components using wires. Once a component is placed onto the design plane,



(a) (b)

Figure 6.10: Dataflow of FSMs using different state encodings.

it indicates a portion of the design already and the comparison and suggestions can start right away. Since the graphical representation of the design represents the overall dataflow of the circuit, this graph structure can be used directly as input to the back end and no synthesis is needed. Though as mentioned before, there are constructs that are only present in LabVIEW.

6.10 Summary

This chapter expanded upon the datapath birthmark and leveraged q-grams to try and capture more of the functionality of the circuit. In addition, performance improvements in terms of extraction and search can be improved upon. Implementation details were given outlining the implemented system at hand. Results show that the approach can scale to very large databases with favorable run-times. With a back-end circuit similarity matcher able to compare real-world circuits, the productivity aspect can now be explored. The next chapter presents a case study on what effect does having reusable designs suggested

automatically during the design phase have on the productivity of the user. Furthermore, different applications where circuit similarity can be applied is further investigated.

Chapter 7

Case Studies

This chapter explore various applications where circuit similarity matching can be used such as functional block prediction, IP theft detection, and reverse engineering. Furthermore, to merit the claim that automatically suggesting reusable designs improves the productivity of the designer, a case study of the implemented tool is conducted in order to attempt to quantify productivity improvements in terms of time spent on design with and without the tool.

7.1 Functional Block Prediction

A case study was designed to investigate the use of extending the q-gram model in attempting to predict the next most likely design or module given the current state of the circuit. In other words, based on the IPs and modules being used, what is the most likely module to be next in the datapath. This then becomes a system-level design. The raise in abstraction level is a common method to increasing overall productivity of a designer.

Instead of suggesting designs that are similar to the current design, predict and suggest components that the designer might need in order to complete the circuit. The idea is autocomplete for circuit design. A more ideal use case would be to use this approach in a graphical design environment such as LabVIEW where the components are essentially drag, drop, and connect.

7.1.1 Prediction Model

Designs containing commonly used IPs are limited. In order to predict future components, a large number of existing designs that commonly use a specific library of components are needed in order to train the q-gram model. This is one reason why applying machine learn-

ing methods is difficult and is currently avoided. Therefore, the q-gram model is built not based on IPs and modules used, but the basic functional blocks of the design, such as adders, shifters, logic operators, etc in order to show the potential feasibility of this approach. The reference circuit in design is a Sobel operator, commonly used in image processing applications to detect edges. The q-gram model was trained on all the circuits in the database.

7.1.2 Evaluation of Model Using Cross Validation

Q-gram models that predict subsequent operations are typically evaluated using two measures that are commonly used in assessing information retrieval applications: precision and applicability.

$$precision = \frac{P_c}{P_n} \qquad applicability = \frac{P_n}{P_s} \tag{7.1}$$

Precision indicates the number of correct predictions over all predictions. Applicability is the number of predictions made out of the total number of attempts made to search for the prefix [124]. Typically, recall is used instead of applicability; however, for a prediction system, recall measures the ratio of relevant items retrieved over all possible relevant items. The concept of relevance is obscure because the interest is not to determine if a circuit is relevant or not, but rather how relevant.

A k-fold cross validation is performed on the q-gram model with $k = 10$, where one-tenth of the data set is used as test and the other is used as training data. The purpose is to

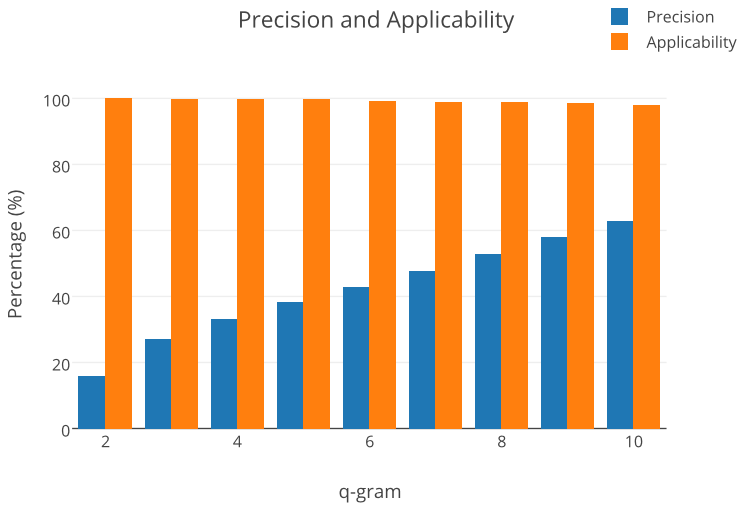


Figure 7.1: The precision and applicability for different values of q obtained from cross validation of the circuit database.

Table 7.1: Functional block prediction at different time steps.

Timestep	Q-Gram	Prediction 1	Prediction 2
1	AS	A (Adder)	M (Mux)
2	ASAA	M (Mux)	L (Logic)
3	ASAA	E (Equality)	X (Multiplier)
4	SAANLAMA	M (Mux)	L (Logic)

see how accurate the approach is able to predict the next item. The result of the cross validation is seen in Figure 7.1. Due to the extension to include tokens smaller than q, the applicability hardly changes as q increases. This is desired because for almost every search query, a prediction can be generated.

The precision of the model is fairly low and on the average case, gets around 50% of the predictions correct, yet at the same time, there are only about 20 different operations that could be chosen. The diversity in token patterns is not as large as a text corpus. Perhaps by

```
module sobel_t7(p0, p1, p2, p3, p5, p6, p7, p8, out);
  input  [7:0] p0,p1,p2,p3,p5,p6,p7,p8; // 8 bit pixels inputs
  output [7:0] out;                     // 8 bit output pixel
  wire signed [10:0] gx,gy, gx1,gx2,gx3, gy1,gy2,gy3;
  wire signed [10:0] abs_gx,abs_gy;
  wire [10:0] sum;

  assign gx1=(p2-p0);
  assign gx2=((p5-p3)<<1);              //Time Step 1
  assign gx3=(p8-p6);
  assign gy1=(p0-p6);
  assign gy2=((p1-p7)<<1);
  assign gy3=(p2-p8);

  assign gx=(gx1+gx2+gx3);              //Time Step 2
  assign gy=(gy1+gy2+gy3);
  assign abs_gx = (gx[10]? ~gx+1 : gx); //Time Step 3
  assign abs_gy = (gy[10]? ~gy+1 : gy);

  assign sum = (abs_gx+abs_gy);         //Time Step 4
  assign out = (|sum[10:8])?8'hff : sum[7:0];
endmodule
```

Figure 7.2: The reference design of a Sobel operator.

having more data to train the model as well as a dataset that is more diverse would allow for higher accuracy. On the other hand, by training the model towards a specific domain, the predictions that are suggested might be more accurate as well.

7.1.3 Experiment

In Figure 7.2, the Sobel code is partitioned into five time steps in order to mimic different stages of the circuit under design. The top two results of the functional block prediction can be seen in Table 7.1. At $T1$, a simple two element path consisting of a subtraction and a shift can be seen. With AS (add shift (line 9)) as the current q-gram, the tool predicted the next node as an add operation, reflected by the addition on line 17.

It is important to note that the circuits used to train the q-gram can heavily influence the prediction model. Models can be trained and used depending on the domain of the application. For example, for a filter design, one would want to use a model that best captures design patterns and trends associated with filters or DSP designs.

7.2 Soft-IP Theft and Reverse Engineering

The flexibility of soft-IPs allow for them to be easily reused and integrated; however, as the number of reusable designs increase, soft-IPs have become more susceptible to theft. Circuits can be used in ways that could be considered illegal or outside the terms of intended use. By looking at the underlying characteristics of a circuit, the similarity of two circuit can be determined and used to indicate the likelihood of theft. Not only can the proposed birthmarking approaches recognize circuits, it can also detect containment, indicating whether a design is contained or used as part of a larger design.

Detecting Theft

The similarity metric needs to determine if the possibility of theft has occurred. The answer most designers are looking for is one of two choices: whether the circuit is obtained and used outside the intended rights or not. The value the similarity metric returns is a score s, $0 \leq s \leq 1$. Therefore, a threshold ρ is used such that if $s > \rho$, then it is highly probable that theft has occurred. A second threshold θ is used such that $\theta \leq s \leq \rho$ indicates that the user may want to investigate the designs further. Anything less than θ is essentially marked as insignificant. Based on the initial experiments performed, it is found that $\rho = 0.7$, $\theta = 0.55$ gave the best results.

Class projects from a digital design course were obtained from the instructor. The projects were independent assignments and the results of the comparison should indicate that no

birthmarks are equivalent, assuming no plagiarism or copying is involved. The students were asked to design a simple hardware module to interact with the VGA controller in order to produce images onto a monitor.

In total, 27 students submitted a project. Comparing the birthmarks of all 27 designs with one another, none of the assignments returned a significant similarity score. The average top score from the match is 0.332, with the max at 0.441 and min at 0.163. Some of the scores are still fairly high; however, this is attributed to the fact that the designs all have the same functionality. Not only does the birthmark capture the functionality of the design, but also specific design patterns unique to that designer.

7.2.1 Claiming Theft

It is important to note that even though a birthmark returns a high similarity score, it is just a metric that indicates how similar two designs are. The intuition is that if two designs are highly similar, then it is highly likely that they are copies of one another. The definition of a birthmark does state that if two programs are copies, then the birthmark is the same; however, if two birthmarks are the same, it does not necessarily indicate that the two designs are necessarily copies. Like other plagiarism detection programs like Moss, the metric returned is only an indicator of how similar two designs appear and is no way an indicator that theft has indeed taken place.

7.2.2 Detection at the Physical Layer

One aspect where birthmarks are irrelevant is at the physical layer. With a physical device, unless the soft-IP is included, birthmarking methods cannot specify if the design in the device is stolen or not. Watermarking on the other hand has the potential to validate the origin of the design at the physical layer. A stringent condition for watermarks is that the watermark should be able to survive through the different layers of design: HDL, the netlist, and at the physical layer.

Birthmarks can be used alongside watermarks to further protect the circuit. By watermarking the design, additional functionality is introduced to the circuit such that when the birthmark of the circuit is derived, fragments of the watermarking or watermark extraction circuit is captured as well. Yet, by introducing birthmarks in the presence of a watermarked design, the birthmarks may reveal information about watermark.

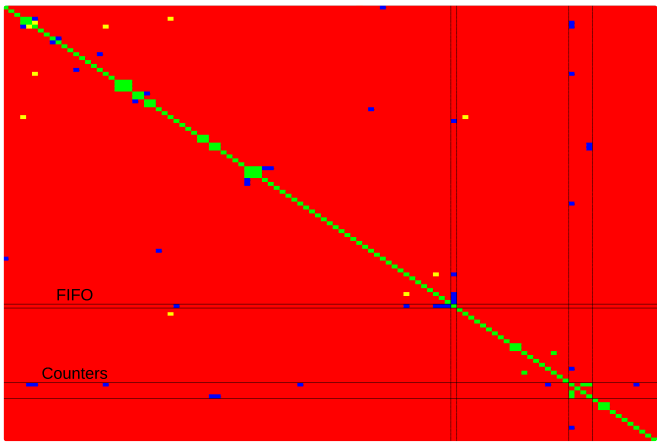


Figure 7.3: Heatmap showing the auto-correlation between all 151 circuits. The green, yellow, and red signifies three levels of significance with green indicating that there is most likely theft and red indicating that there is little in common between the two. Blue depicts containment between two designs.

7.2.3 Credibility

In another experiment, to evaluate the credibility of the birthmark, an autocorrelation was performed on a database of 110 designs. The heat map of the auto-correlation is seen in Figure 7.3. Most of the designs returned an insignificant score. Out of the 110 circuits that were analyzed, 16 circuits were found to have significant similarity. Only five circuits warranted further investigation and 21 circuits were found to be contained or have containing circuits. A majority of the containment matches found were due to smaller and simple designs such as counters and FIFOs, which many larger designs are going to contain. Looking into the positive matches, the results were in fact accurate indicating related designs.

7.2.4 Reverse Engineering

Understanding a design can be done by leveraging the containment search that the birthmarking methods provide. Processing a subset of designs from the database revealed interesting information about some of the designs and relationship certain circuits had with each other. Two OpenCores design, the Zet SoC platform and the ORGFXSoC, were found to contain a similar divider implementation. Additionally, a simple programmable interrupt controller from OpenCores was detected in a project on GitHub. The design had been heavily modified from the original; however, the original copyright statement was intact.

7.3 Productivity

With a back-end that is able to compare and determine how similar two hardware designs are, reusable designs and components can now be suggested to the user. A case study is performed to determine what effects the new reuse model has on the productivity of the designer as well if the suggestions that appear is relevant and helpful during the design phase. Participants of different design backgrounds were chosen ranging from students who had just finished an introductory digital design course to graduate students who have done digital design for many years. This case study was approved by the Virginia Tech Institution Review Board (IRB) 15-1192 for conducting human subject activities outlined by the specified IRB-approved protocol.

7.3.1 Design Environment

The design environment that the participants used is shown in Figure 7.4. The environment, designed using QT, contains a simple text editor where the Verilog code is entered. In the same design environment, a list of reusable designs are suggested to the user in a ranked order off to the right. There are two lists: one that shows resemblance and the other shows containment. The status window contains information about errors and the status of the

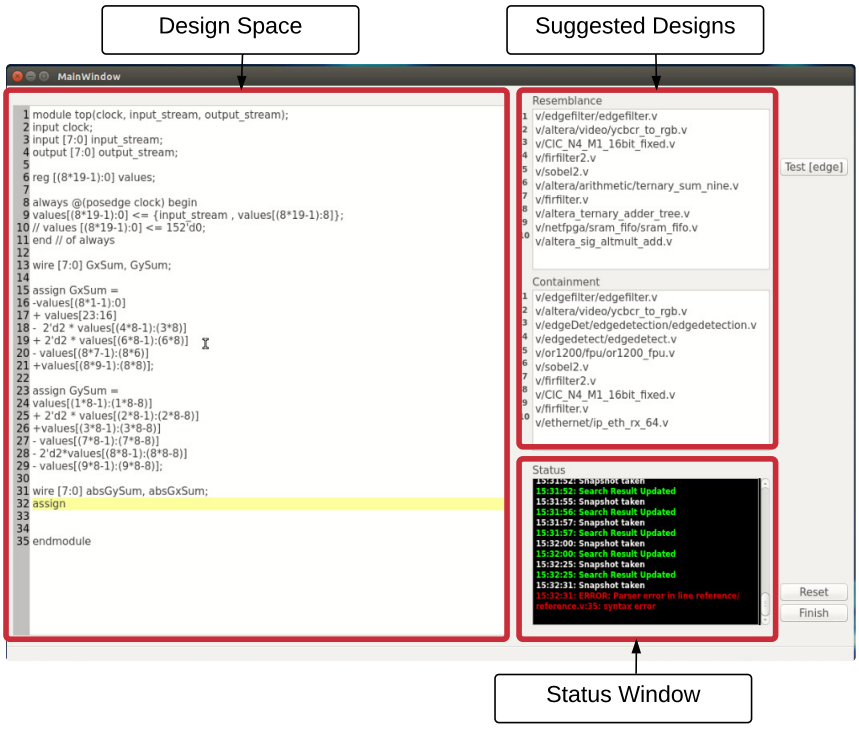


Figure 7.4: Design interface for case study.

design. A testbench was designed for the users to check to see if their designs are functionally correct.

As the user designs the circuit, snapshots of the reference design are taken. These snapshots are taken after the user stops modifying the design for 1.5 seconds. The snapshot is sent to the back-end where Yosys determines if there are any errors. If there are errors, the status window updates notifying the user of the error location. Currently, a synthesizable HDL design is needed in order for Yosys to extract the DFG. If there are no errors, the birthmark is extracted and used to look for similar designs. These designs then automatically populate in the window, notifying the designer of similar hardware that can be used. The entire process is automated and requires no intervention. Afterwards, the user can then double click the designs to look at the HDL and documentations in order to decide if the design suggested can be reused.

Quantifying Productivity

Productivity is a hard measure to quantify. Design time is a typical measure that is used [1, 9, 125]. Yet, it is difficult to capture productivity based solely on design time and can be affected by many factors. Different designers with varying experience, training, knowledge, and skills will all affect design time. Source line of code (SLOC) have been commonly used as well but based on the language and syntax, any lines of code can be combined into one [126]. Furthermore code reuse and object oriented programming affects the SLOC metric [127]. Designers may spend hours on end and only produce several lines of code. Function points are also another common measure that is a weighted sum of input, output, files, and other queries to the program in order to estimate the amount of *function* the program is to perform [128, 126]. Nelson et al. provides a model for measuring hardware design productivity taking into account reuse, turns per day, lines of code, and hardware complexity [9]. Yet, hardware complexity by counting gates and LUTs varies and depends on synthesis tools. Furthermore, modular design reduces the lines of code similar to object oriented designs in hardware.

For this case study, the metric of focus will be design time. In order to keep the experiment as controlled as possible, two designs were chosen for the participants to design. The circuits were constructed such that they are of close complexity and should require about the same amount of time to complete. With one design, the user will design it without using the tool, and the other with the tool. Productivity is measured by how much faster the participant was able to design the circuit with the tool used. Since the design time is relative to only the user with and without the tool, designer experience and skill can be controlled to an extent.

7.3.2 Design Problem

The first circuit each participant was asked to design was an edge detector. Given an 8x8 gray-scale image, return the resulting image after a Sobel image filter has been applied. This

Table 7.2: Additional Reuse statistics.

Subjects	Fin (Tool)	Reuse	Fin (No Tool)
P1	Y	Y	N
P2	Y	N	N
P3	N	N	N
P4	N	N	N
P5	Y	Y	N
P6	Y	Y	Y
P7	Y	N	N

design focused on the combinational logic of the Sobel operator. Image data is passed into the circuit one pixel at a time each clock cycle. The circuit performs the Sobel operation and sends the resulting pixel out. The subject was not required to determine whether the output pixel was valid or not. The second circuit is an image sampling design. Given an 8x8 gray-scale image, return an image that is 7x7. This design focused on timing accurate design and control logic. Pixel data is streamed in similar to the first design, except the user is responsible for determining when the data is valid. The sampling was performed by returning the maximum value of the pixels in a 2x2 window with a stride of 1.

Participants were not given direct access to the database and no search feature was implemented for the database. The current reuse model leverages repositories of circuits that are retrieved based on keywords and tags. Participants were allowed to search for reusable

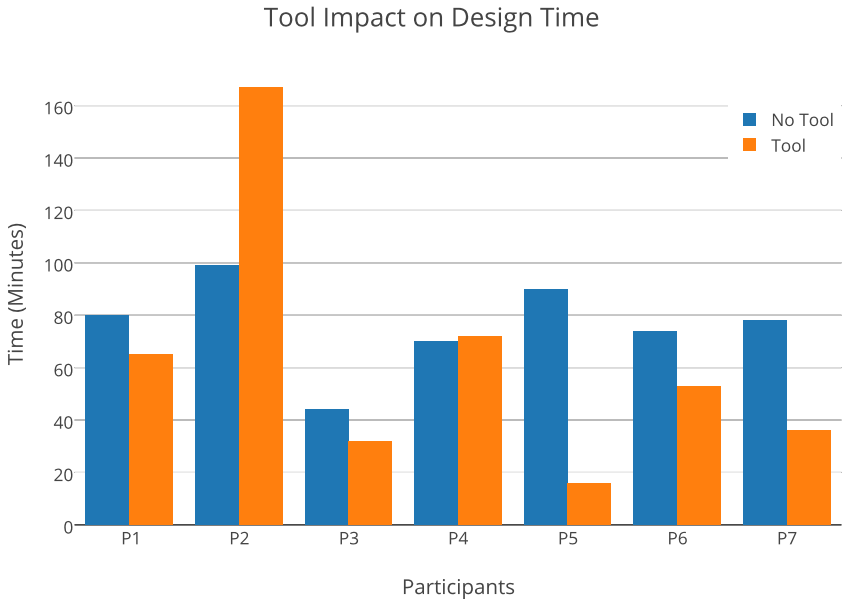


Figure 7.5: The impact of the proposed reuse model on the overall design time of the user.

components and designs using the web. All the circuits in the database are all available publicly online and therefore the participants should be able to find every single design that is in the preprocessed database. The time consuming part is understanding where and what to search for.

7.3.3 Results

It is important to note, that not all participants were able to get the design working, and the time indicated in the subsequent figures only captures the time they spent on the design during the study. Since the study required human participants, the participants were free to withdraw from the study at any time.

Figure 7.5 shows the overall design time with and without the tool for each of the participants. *P5* showed a drastic increase in performance between the two designs. This was because *P5* started with a simpler component of the design and the resulting design was found after a couple lines of HDL. *P2* on the other hand showed the complete opposite. However, it is important to note that the participant did not finish the design with no tool, but decided to stay and finish the second design. Also, the design that *P2* designed differed from what was in the database and therefore did not necessarily aid the user in the end, similarly with *P4*. Table 7.2 shows the designs completed by each of the participants. Out of those that designed with the tool, five of them finished the designed. Three out of the four found a complete design that met the specifications using the tool. *P7* completed the design with the tool by reusing the suggested sobel filter, but implemented the buffering of the incoming pixels. Only one of the user was able to complete the design not using the tool itself. Completion rate of participants that used the tool is much higher than not having the tool available.

Figure 7.6 shows the overall design time with and without the tool as well as the ideal finish time. The ideal finish time is when the tool first suggests the design that meets the specifications of the user. The actual design time and real design time differ due to the fact that just because the design is suggested, it does not necessarily mean that the user will notice that the circuit of interest is in the list. Furthermore, feedback from the users indicate that this model of design reuse was new and so the first thing was not to necessarily check the list for the circuit of interest when the list populated. Also, when the participants are focused on the design at hand, the updates in the list typically goes unnoticed. Additional user interface elements might help alleviate this problem as well as further improve productivity. Ideal time does not include additional time the user might need to spend to inspect the suggested design as well as integration of the design into the reference.

Based on the ideal finish time, for the most part, a speed up in design time could be seen. However, it can also be seen that there are several participants where the tool was unable to predict the design for them. The design first showed up for *P2* early on; due to debugging and various changes, the design of *P2* changed towards the end such that the suggestion no longer appeared anymore. The design of *P3* and *P4* differed significantly from the design

in the database as well. *P5* on the other hand forgot to make sure all the errors of the design were handled throughout the design process and therefore the tool was not able to generate the DFG for the design. However, with a strong hardware community consisting of members that contribute and take designs to and from the database, the most common methods that describes a specific function should be captured by the database. For example, if the design of *P3* was entered into the database, the circuit *P3* designed could have helped *P4* since the design of *P3* and *P4* were very similar. For the most part, the ideal finish time shows a significant improvement over the design time. On average, at least a 34% increase in productivity was seen based on ideal time. It is also important to note that the numbers for the designs that weren't finish shows a lower bound.

7.3.4 User Feedback and Future Work

Most of the feedback from the user involved additional features to the design environment. Better forms of suggestion were requested in order to allow the user to quickly identify the designs of interest. Additional analysis features were requested as well, such as identifying which part of the existing design is similar to the reference design, similar to a `diff` operation. In addition to a displaying the design, the similarity score could be displayed allowing the user to quickly see the importance of the design. If the top design was 95% similar vs 45% similar, the user would be more inclined to look further into the design. A threshold value could be added as well. If the similarity of a design to the reference exceeds a threshold, inform the user with a notification.

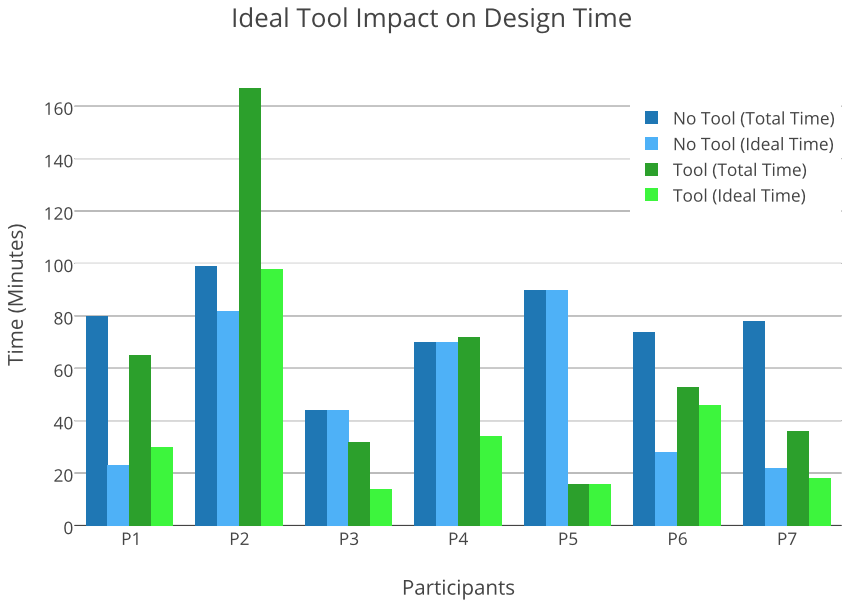


Figure 7.6: The impact of the proposed reuse model on the ideal design time of the user.

Many of the participants suggested that the suggestion of existing designs would be beneficial to their productivity and would use such a feature if it exists in a vendor's tool. Even though some of the participants did not interface with the tool in depth, the designs were simple and straightforward enough that they thought they could finish on their own. Because of this, participants did not go out of their way to search for existing designs online. Nonetheless, the debugging stage proved more time consuming for most. Had they chose to reuse and interface more with the suggestions the tool provided, productivity could have been increased.

7.3.5 Discussion

One noticeable disadvantage of this approach is the fact that there has to be enough of the reference design implemented such that the tool can suggest a circuit that the designer can leverage. This disadvantage favors a well design IP library such that the designer can just search and reuse a design immediately rather than having to design a circuit completely from scratch. However, such libraries need to be preprocessed with metadata and tags. Furthermore, the search is only as good as the keywords used. For example, in the case study, participants were asked what keywords they would have used to search for the image sampling design. Example keywords include reduce image, find max, scale image size, etc. However, those familiar with convolutional neural networks know that this is a typical operation called max pool [129]. Also, keywords may have more than one meaning associated with it. For example, keywords such as edge detector can be used for the design where participants have to detect the edges of the image; however, it also refers to hardware that detects edges of a particular signal. Also, the users may be unaware of what may exist. For the edge detector circuit, many of the participants looked for a Sobel operator circuit. However, none thought to look for a circuit that performed the entirety of the specification. Rather, the participants were unaware of the existing designs that they could have leveraged nor how to describe such a design in keywords that would give them the design they want.

What is interesting about this approach is that it does not disregard a well-designed IP library. By having a library database where the user can search for reusable designs, designers can pass more information into the tool quicker. For example, many of the participants in the case study implemented a shift register as a buffer for the pixel data. Instead of writing the shift register from scratch, the user could have pulled in an existing design of a shift register from a library. Once pulled in, the tool would have found designs that utilizes shift registers. Therefore, reusable components suggested can further complement a well-designed IP library.

Chapter 8

Conclusion

FPGAs are growing faster than ever before and are pursued as power-efficient accelerators for applications that require substantial computational resources. Yet, hardware productivity of the design tools has lingered far behind the growth of physical devices.

Circuit designers can reduce design time significantly by not having to redesign circuits that already exist. The designers can focus more on the application rather than the verification and debugging of existing hardware. As such, modular design methodology has become the norm as the number of reusable IPs increase. Standards are becoming widely accepted as industry leaders start adopting and promoting them.

With the rise in the number of IPs, there requires a need to be able to efficiently retrieve and analyze the existing pool of reusable content. Existing reuse systems are simple and require much attention from the designer. If there exists a tool where designs can be automatically suggested to the user throughout the design phase, productivity could be enhanced. Though, to do so, there requires a need to evaluate the similarity of two hardware designs.

Various representations and methods were presented for describing and comparing digital circuits. The interest when comparing is not to find an exact match, but to determine how similar two designs are. Four main concepts were explored: graph-based methods, molecular similarity, birthmarks consisting of datapaths, and birthmarks consisting of q-grams.

With graph-based methods, the netlist representation captures the entirety of the circuit at the cost of long run-times. These methods focused more on the structural properties of the design. As such, molecular similarity was explored using fingerprinting. The idea here is that two molecules are similar if they share many of the same subcomponents. In order to capture functional properties of the circuit, Boolean matching was used. Though better than using graphs to compare the netlists, scaling to real-world examples was out of reach. The netlist representation contains a lot of information that can be difficult to extract. Therefore, the DFG of the HDL was used. Complementary, a new birthmarking scheme, commonly seen with software similarity matching, is proposed that specifically targets hard-

ware designs. Results were much more promising as larger designs could now be compared. A different birthmarking scheme was explored in order to scale based on increasing database size. Concepts from q-grams and document similarity matching were used to show that the representation allows for a scalable approach for comparing the similarity of circuit designs accurately.

The circuit similarity explored in this dissertation is geared towards finding similar circuits in order to suggest reusable designs to the user. Even though the q-gram approach proved to be better overall, it doesn't not necessarily mean it is the best approach out of the four. The datapath birthmark allows for more flexibility during the match due to the sequence alignment technique used in order to find mismatches and errors in specific datapaths whereas the q-gram matches the entire token and does not necessarily handle insertions and deletions effectively. Furthermore, there are many ways to describe a circuit and to design a comparator that can detect any implementation of a FFT filter as a standalone or embedded in a larger design is extremely difficult. As such, even though the matching done is inexact, there are still many dependencies in design choices, components used, and architecture of the circuit that can affect the match. On another note, many IPs are protected and not necessarily publicly available or free. Yet, there are many IPs that are within a company that can be reused. A database could be created using the IPs internal to that one company. Also, databases can be created and customized for sharing with potential collaborators or customers with licensing agreements allowing for varying use cases for deployment in industry.

With the backbone in place, the entire system was implemented in order to assess the effects of automating the suggestion of reusable designs had on a hardware designer. Results show that this approach has the potential to increase productivity of the designer by 34%. User feedback was overall promising suggesting various features and research opportunities for future works.

8.1 Contributions

The contributions this dissertation provides are reiterated below:

1. A new reuse-based design methodology for hardware that integrates reuse into the design environment. By automatically suggesting reusable designs, the environment provides a better reuse experience, requiring no user effort.

2. Graphs, molecular fingerprinting, and software birthmarking were explored. The q-gram approach was found to be an effective representation that can not only capture the structural and functional attributes of a circuit design, but to be able to efficiently compare the similarity as well.

3. This work brings this concept into fruition with an implemented reuse environment where designs are suggested automatically in order to analyze the effect this tool has

on the productivity of hardware designers. Studies show that productivity of hardware designers could potentially be increased by 34%.

8.2 Future Works

Future work could explore different birthmarking schemes for digital circuits that 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. Also, various optimizations and selection of parameters and weights can be explored in order to further improve performance and scalability of the methods.

On the other hand, the front-end user interface can be explored. In other words, how best to notify the user of possible designs as well as make integration and analysis of existing designs easier for the user. For example, indicating regions of similarity between the reference and the existing design. With a centralized or even a local database of existing designs, the proposed reuse model 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.

The intuition is the designer is unaware of emerging circuits and design patterns that can be applied and the goal of this work is to empower designers with the capabilities to efficiently analyze large libraries of reusable components.

Appendix A

Appendix A: Top 10 results for 5 different circuits with varying q

Below are tables that depict the top ten results for five different circuits with varying q. The q-gram scheme used was a path-based scheme.

Table A.1: $q = 2$ Path-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	or1200_dpram	prewitt
2	memcard	cf_fft_256_8	lm32_multiplier	lm32_ram	sobel3
3	basic_fifo	cf_fft_512_18	altera_unsig_altmult_accum	softusb_dpram	sobel_3
4	or1200_sb	cf_fft_512_16	reset_filter	cam_lut_sm	edgedetect
5	tiny_spi	cf_fft_512_8	reset_filter	generic_mem_small	sobel2
6	rtfSimpleUart	uart	altera_unsigned_mult	generic_mem_medium	sobel
7	eth_axis_tx	unencoded_cam_lut_sm	clip_reg	altera_true_dpram_sclk	edgefilter
8	simple_spi_top	simple_spi_top	c_rand	behave2p_mem	edgedetection
9	arp_eth_tx	eth_parser	synfir	setting_reg	sobel_accelerator
10	uart_rx_only3	channel_ram	ycbcr_to_rgb	ready_skid	add2_and_clip
MAP	44.20238095	82.28174603	45.03571429	71.1031746	64.92063492

Table A.2: $q = 3$ Path-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	or1200_sb	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	basic_fifo	cf_fft_512_18	reset_filter	softusb_dpram	sobel2
4	rc5	cf_fft_512_16	clip_reg	or1200_dpram	edgedetect
5	memcard	cf_fft_512_8	altera_unsigned_mult	ram_sp_sr_sw	edgefilter
6	memtest	eth_parser	synchronizer	altera_shift_1x64	sobel_3
7	rtfSimpleUart	uart	lm32_multiplier	setting_reg	sobel
8	rtfSimpleUartTx	channel_ram	qmult2	generic_mem_small	edgedetection
9	rtfSimpleUartRx	op_lut_hdr_parser	gmii_phy_if	generic_mem_medium	sobel_accelerator
10	small_fifo_v3	fallthrough_small_fifo_v2	add2_reg	behave2p_mem	qmult
MAP	51.49206349	82.28174603	42.53571429	71.1031746	66.34920635

Table A.3: $q = 4$ Path-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	memtest	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	memcard	cf_fft_512_16	clip_reg	or1200_dpram	edgefilter
5	rtfSimpleUartTx	cf_fft_512_8	altera_unsigned_mult	softusb_dpram	edgedetect
6	or1200_sb	eth_parser	synchronizer	altera_shift_1x64	sobel_3
7	rtfSimpleUart	uart	qmult2	setting_reg	sobel
8	sd_spi	uart_transceiver	gmii_phy_if	generic_mem_small	sobel_accelerator
9	buf_3to2	ip_checksum_ttl	add2_reg	generic_mem_medium	edgedetection
10	buf_2to3	op_lut_hdr_parser	clkgen	behave2p_mem	qmult
MAP	56.82539683	82.28174603	45.03571429	71.1031746	66.34920635

Table A.4: $q = 5$ Path-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	rtfSimpleUartTx	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	memtest	cf_fft_512_16	clip_reg	or1200_dpram	edgefilter
5	memcard	cf_fft_512_8	altera_unsigned_mult	softusb_dpram	edgedetect
6	rtfSimpleUart	eth_parser	synchronizer	behave1p_mem	sobel_3
7	buf_3to2	uart	qmult2	altera_shift_1x64	sobel
8	buf_2to3	ip_checksum_ttl	gmii_phy_if	setting_reg	sobel_accelerator
9	comb_heir_fir	uart_transceiver	add2_reg	generic_mem_small	add2_and_clip
10	uart_simple	rtfSimpleUart	clkgen	generic_mem_medium	qmult
MAP	51.03571429	82.28174603 200 42.53571429	71.1031746	67.59920635	

Table A.5: $q = 6$ Path-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	rtfSimpleUartTx	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	memcard	cf_fft_512_16	altera_unsigned_mult	or1200_dpram	edgefilter
5	rtfSimpleUart	cf_fft_512_8	clip_reg	softusb_dpram	edgedetect
6	comb_heir_fir	uart	synchronizer	behave1p_mem	sobel_3
7	uart_simple	eth_parser	qmult2	setting_reg	sobel
8	buf_3to2	rtfSimpleUart	gmii_phy_if	altera_shift_1x64	add2_and_clip
9	buf_2to3	ip_checksum_ttl	add2_reg	generic_mem_small	sobel_accelerator
10	memtest	uart_transceiver	clkgen	generic_mem_medium	qmult
MAP	39.5515873	82.28174603	45.8968254	80.29761905	69.71031746

Table A.6: $q = 7$ Path-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	memcard	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	rtfSimpleUart	cf_fft_512_16	clip_reg	or1200_dpram	edgefilter
5	comb_heir_fir	cf_fft_512_8	altera_unsigned_mult	softusb_dpram	edgedetect
6	uart_simple	uart	synchronizer	behave1p_mem	sobel_3
7	rtfSimpleUartTx	rtfSimpleUart	qmult2	setting_reg	sobel
8	buf_2to3	timer	gmii_phy_if	altera_shift_1x64	add2_and_clip
9	buf_3to2	eth_parser	add2_reg	generic_mem_small	sobel_accelerator
10	memtest	uart_transceiver	clkgen	generic_mem_medium	qmult
MAP	39.5515873	82.28174603	45.8968254	80.29761905	69.71031746

Table A.7: $q = 8$ Path-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	memcard	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	comb_heir_fir	cf_fft_512_16	altera_unsigned_mult	or1200_dpram	edgefilter
5	rtfSimpleUart	cf_fft_512_8	clip_reg	softusb_dpram	sobel_3
6	uart_simple	uart	synchronizer	behave1p_mem	edgedetect
7	buf_2to3	rtfSimpleUart	qmult2	setting_reg	add2_and_clip
8	buf_3to2	timer	gmii_phy_if	altera_shift_1x64	sobel
9	serial	uart_transceiver	add2_reg	generic_mem_small	sobel_accelerator
10	arp_cache	rtfSimpleUartRx	clkgen	generic_mem_medium	qmult
MAP	36.74603175	82.28174603	69.32539683	85.93253968	69.71031746

Appendix B

Appendix B: Top 10 results for 5 different circuits with varying q using Frequency-based scheme

Below are tables that depict the top ten results for five different circuits with varying q. The q-gram scheme used was a Frequency-based scheme.

Table B.1: $q = 2$ Frequency-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	or1200_dpram	prewitt
2	memcard	cf_fft_256_8	lm32_multiplier	lm32_ram	sobel3
3	basic_fifo	cf_fft_512_18	altera_unsig_altmult_accum	softusb_dpram	sobel_3
4	or1200_sb	cf_fft_512_16	reset_filter	cam_lut_sm	edgedetect
5	tiny_spi	cf_fft_512_8	reset_filter	generic_mem_small	sobel2
6	rtfSimpleUart	uart	altera_unsigned_mult	generic_mem_medium	sobel
7	eth_axis_tx	unencoded_cam_lut_sm	clip_reg	altera_true_dpram_sclk	edgefilter
8	simple_spi_top	simple_spi_top	c_rand	behave2p_mem	edgedetection
9	arp_eth_tx	eth_parser	synfir	setting_reg	sobel_accelerator
10	uart_rx_only3	channel_ram	ycbcr_to_rgb	ready_skid	add2_and_clip
MAP	55.0515873	82.28174603	42.35714286	69.46428571	64.23809524

Table B.2: $q = 3$ Frequency-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	or1200_sb	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	basic_fifo	cf_fft_512_18	reset_filter	softusb_dpram	sobel2
4	rc5	cf_fft_512_16	clip_reg	or1200_dpram	edgedetect
5	memcard	cf_fft_512_8	altera_unsigned_mult	ram_sp_sr_sw	edgefilter
6	memtest	eth_parser	synchronizer	altera_shift_1x64	sobel_3
7	rtfSimpleUart	uart	lm32_multiplier	setting_reg	sobel
8	rtfSimpleUartTx	channel_ram	qmult2	generic_mem_small	edgedetection
9	rtfSimpleUartRx	op_lut_hdr_parser	gmii_phy_if	generic_mem_medium	sobel_accelerator
10	small_fifo_v3	fallthrough_small_fifo_v2	add2_reg	behave2p_mem	qmult
MAP	58.93650794	82.28174603	45.69047619	69.46428571	66.34920635

Table B.3: $q = 4$ Frequency-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	memtest	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	memcard	cf_fft_512_16	clip_reg	or1200_dpram	edgefilter
5	rtfSimpleUartTx	cf_fft_512_8	altera_unsigned_mult	softusb_dpram	edgedetect
6	or1200_sb	eth_parser	synchronizer	altera_shift_1x64	sobel_3
7	rtfSimpleUart	uart	qmult2	setting_reg	sobel
8	sd_spi	uart_transceiver	gmii_phy_if	generic_mem_small	sobel_accelerator
9	buf_3to2	ip_checksum_ttl	add2_reg	generic_mem_medium	edgedetection
10	buf_2to3	op_lut_hdr_parser	clkgen	behave2p_mem	qmult
MAP	55.15873016	82.28174603	42.35714286	69.46428571	67.59920635

Table B.4: $q = 5$ Frequency-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	rtfSimpleUartTx	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	memtest	cf_fft_512_16	clip_reg	or1200_dpram	edgefilter
5	memcard	cf_fft_512_8	altera_unsigned_mult	softusb_dpram	edgedetect
6	rtfSimpleUart	eth_parser	synchronizer	behave1p_mem	sobel_3
7	buf_3to2	uart	qmult2	altera_shift_1x64	sobel
8	buf_2to3	ip_checksum_ttl	gmii_phy_if	setting_reg	sobel_accelerator
9	comb_heir_fir	uart_transceiver	add2_reg	generic_mem_small	add2_and_clip
10	uart_simple	rtfSimpleUart	clkgen	generic_mem_medium	qmult
MAP	60.6031746	82.28174603	45.69047619	67.46428571	67.59920635

Table B.5: $q = 6$ Frequency-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	rtfSimpleUartTx	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	memcard	cf_fft_512_16	altera_unsigned_mult	or1200_dpram	edgefilter
5	rtfSimpleUart	cf_fft_512_8	clip_reg	softusb_dpram	edgedetect
6	comb_heir_fir	uart	synchronizer	behave1p_mem	sobel_3
7	uart_simple	eth_parser	qmult2	setting_reg	sobel
8	buf_3to2	rtfSimpleUart	gmii_phy_if	altera_shift_1x64	add2_and_clip
9	buf_2to3	ip_checksum_ttl	add2_reg	generic_mem_small	sobel_accelerator
10	memtest	uart_transceiver	clkgen	generic_mem_medium	qmult
MAP	63.8968254	82.28174603	45.69047619	67.46428571	67.59920635

Table B.6: $q = 7$ Frequency-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	memcard	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	rtfSimpleUart	cf_fft_512_16	clip_reg	or1200_dpram	edgefilter
5	comb_heir_fir	cf_fft_512_8	altera_unsigned_mult	softusb_dpram	edgedetect
6	uart_simple	uart	synchronizer	behave1p_mem	sobel_3
7	rtfSimpleUartTx	rtfSimpleUart	qmult2	setting_reg	sobel
8	buf_2to3	timer	gmii_phy_if	altera_shift_1x64	add2_and_clip
9	buf_3to2	eth_parser	add2_reg	generic_mem_small	sobel_accelerator
10	memtest	uart_transceiver	clkgen	generic_mem_medium	qmult
MAP	48.57936508	82.28174603	51.03571429	76.25396825	69.71031746

Table B.7: $q = 8$ Frequency-Based Scheme.

Rank	mmuart*	cf_fft_256_18*	s_mult [102]	generic_dpram*	sobel [120]
1	uart_transceiver	cf_fft_256_16	altera_sig_altmult_add	altera_true_dpram_sclk	prewitt
2	rc5	cf_fft_256_8	reset_filter	lm32_ram	sobel3
3	memcard	cf_fft_512_18	reset_filter	ram_sp_sr_sw	sobel2
4	comb_heir_fir	cf_fft_512_16	altera_unsigned_mult	or1200_dpram	edgefilter
5	rtfSimpleUart	cf_fft_512_8	clip_reg	softusb_dpram	sobel_3
6	uart_simple	uart	synchronizer	behave1p_mem	edgedetect
7	buf_2to3	rtfSimpleUart	qmult2	setting_reg	add2_and_clip
8	buf_3to2	timer	gmii_phy_if	altera_shift_1x64	sobel
9	serial	uart_transceiver	add2_reg	generic_mem_small	sobel_accelerator
10	arp_cache	rtfSimpleUartRx	clkgen	generic_mem_medium	qmult
MAP	42.53571429	82.28174603	69.32539683	90.84920635	69.71031746

Bibliography

[1] A. Reutter and W. Rosenstiel, "An efficient reuse system for digital circuit design," in *Proceedings of the conference on Design, automation and test in Europe*, p. 9, ACM, 1999.

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

[3] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pp. 13–24, IEEE, 2014.

[4] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, 2015.

[5] P. K. Gupta, "Intel xeon+fpga platform intel xeon+fpga platform for the data center," 2015.

[6] S. M. Trimberger, "Three ages of fpgas: A retrospective on the first thirty years of fpga technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, 2015.

[7] X. Xue, A. Cheryauka, and D. Tubbs, "Acceleration of fluoro-ct reconstruction for a mobile c-arm on gpu and fpga hardware: a simulation study," in *Medical Imaging*, pp. 61424L–61424L, International Society for Optics and Photonics, 2006.

[8] A. Love and P. Athanas, "Rapid modular assembly of xilinx fpga designs," in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013.

[9] B. Nelson, M. Wirthlin, B. Hutchings, P. Athanas, and S. Bohner, "Design Productivity for Configurable Computing ," 2008.

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

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

[12] 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.

[13] F. Rincón, F. Moya, J. Barba, and J. C. López, "Model reuse through hardware design patterns," *Proceedings -Design, Automation and Test in Europe, DATE '05*, vol. I, pp. 324–329, 2005.

[14] W. Savage, J. Chilton, and R. Camposano, "IP reuse in the system on a chip era," in *Proceedings of the 13th International Symposium on System Synthesis*, ISSS '00, (Washington, DC, USA), pp. 2–7, IEEE Computer Society, 2000.

[15] H. D. Foster, "Why the design productivity gap never happened," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 581–584, IEEE Press, 2013.

[16] R. Bergmann, M. M. Richter, S. Schmitt, A. Stahl, and I. Vollrath, "Utility-oriented matching: A new research direction for case-based reasoning," in *Professionelles Wissensmanagement: Erfahrungen und Visionen. Proceedings of the 1st Conference on Professional Knowledge Management. Shaker*, 2001.

[17] J. S. Meehan, A. H. Duffy, and R. I. Whitfield, "Supportingdesign for re-use'with modular design," *Concurrent Engineering*, vol. 15, no. 2, pp. 141–155, 2007.

[18] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design & Test of Computers*, no. 4, pp. 18–25, 2009.

[19] A. Chandrasekharan, S. Rajagopalan, G. Subbarayan, T. Frangieh, Y. Iskander, S. Craven, and C. Patterson, "Accelerating fpga development through the automatic parallel application of standard implementation tools," in *Field-Programmable Technology (FPT), 2010 International Conference on*, pp. 53–60, Dec 2010.

[20] E. Girczyc and S. Carlson, "Increasing design quality and engineering productivity through design reuse," in *Proceedings of the 30th international Design Automation Conference*, pp. 48–53, ACM, 1993.

[21] S. Sarkar and S. Shinde, "Effective IP reuse for high quality soc design," in *SOC Conference, 2005. Proceedings. IEEE International*, pp. 217–224, IEEE, 2005.

[22] R. Damasevicius, G. Majauskas, and V. Stuikys, "Application of design patterns for hardware design," in *Design Automation Conference, 2003. Proceedings*, pp. 48–53, June 2003.

[23] K. Lo, "Design for reuse," in *Systems on a Chip (Ref. No. 1998/439), IEE Colloquium on*, pp. 11/1–11/6, Sep 1998.

[24] V. Berman, "Standards: The p1685 IP-XACT IP metadata standard," *Design Test of Computers, IEEE*, vol. 23, pp. 316–317, April 2006.

[25] N. Rollins, A. Arnesen, and M. Wirthlin, "An XML schema for representing reusable IP cores for reconfigurable computing," in *Aerospace and Electronics Conference, 2008. NAECON 2008. IEEE National*, pp. 190–197, IEEE, 2008.

[26] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.

[27] W.-D. Weber, "Enabling reuse via an IP core-centric communications protocol: Open core protocol (tm)," *Proceedings of the IP*, pp. 20–22, 2000.

[28] M. Wirthlin, D. Poznanovic, P. Sundararajan, A. Coppola, D. Pellerin, W. Najjar, R. Bruce, M. Babst, O. Pritchard, P. Palazzari, *et al.*, "Openfpga corelib core library interoperability effort," *Parallel Computing*, vol. 34, no. 4, pp. 231–244, 2008.

[29] M. Mitić and M. Stojčev, "A survey of three system-on-chip buses: Amba, coreconnect and wishbone," *Proc. of ICEST*, 2006.

[30] S. Opencores, "Wishbone system-on-chip (SoC) interconnection architecture for portable IP cores," tech. rep., Technical report, Opencores, 2002.

[31] D. Flynn, "Amba: enabling reusable on-chip designs," *Micro, IEEE*, vol. 17, pp. 20–27, Jul 1997.

[32] F. ming Xiao, D. sheng Li, G.-M. Du, Y. kun Song, D. li Zhang, and M.-L. Gao, "Design of axi bus based mpsoc on fpga," in *Anti-counterfeiting, Security, and Identification in Communication, 2009. ASID 2009. 3rd International Conference on*, pp. 560–564, Aug 2009.

[33] I. Microelectronics, "Coreconnect bus architecture," 1999.

[34] M. Birnbaum and H. Sachs, "How vsia answers the soc dilemma," *Computer*, vol. 32, pp. 42–50, Jun 1999.

[35] C. Lennard, P. Schaumont, G. de Jong, A. Haverinen, and P. Hardee, "Standards for system-level design: practical reality or solution in search of a question?," in *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, pp. 576–583, 2000.

[36] V. Berman, "Standards update from IP 07," *Design Test of Computers, IEEE*, vol. 25, pp. 192–193, March 2008.

[37] D. Baxter, J. Gao, K. Case, J. Harding, B. Young, S. Cochrane, and S. Dani, "An engineering design knowledge reuse methodology using process modelling," *Research in engineering design*, vol. 18, no. 1, pp. 37–48, 2007.

[38] A. Lowe, C. McMahon, and S. Culley, "Information access, storage and use by engineering designers, part 1," *The Journal of the Institution of Engineering Designers*, vol. 30, no. 2, pp. 30–32, 2004.

[39] J. Liu, E. Shragowitz, and W.-t. Tsai, "Combining hierarchical filtering, fuzzy logic, and simulation with software agents for ip (intellectual property) selection in electronic design," *International Journal on Artificial Intelligence Tools*, vol. 10, no. 03, pp. 303–323, 2001.

[40] J. Liu and E. B. Shragowitz, "Application of fuzzy logic in intelligent software agents for IP selection," in *SPIE's International Symposium on Optical Science, Engineering, and Instrumentation*, pp. 168–175, International Society for Optics and Photonics, 1999.

[41] J. Altmeyer, S. Ohnsorge, and B. Schürmann, "Reuse of design objects in cad frameworks," in *Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design*, pp. 754–761, IEEE Computer Society Press, 1994.

[42] P. Oehler, I. Vollrath, P. Conradi, R. Bergmann, and T. Wahlmann, "READee–decision support for IP selection using a knowledge-based approach," in *IP98 Europe Proceedings, Miller Freeman*, 1998.

[43] I. Vollrath and P. Oehler, "Intelligent retrieval of electronic designs in online catalogs," *Center for Learning Systems and Applications (LSA)*, 1998.

[44] Y. Ye and G. Fischer, "Supporting reuse by delivering task-relevant and personalized information," in *Proceedings of the 24th international conference on Software engineering*, pp. 513–523, ACM, 2002.

[45] O. Hummel, W. Janjic, and C. Atkinson, "Code conjurer: Pulling reusable software out of thin air," *IEEE Software*, vol. 25, pp. 45–52, Sept 2008.

[46] A. C. Martins, V. C. Garcia, E. S. de Almeida, and S. R. d. L. Meira, "Suggesting software components for reuse in search engines using discovered knowledge techniques," in *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, pp. 412–419, Aug 2009.

[47] W.-H. Chang, S.-D. Tzeng, and C.-Y. Lee, "A novel subcircuit extraction algorithm by recursive identification scheme," in *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, vol. 5, pp. 491–494, IEEE, 2001.

[48] C. S. Ou, C. L. Lu, and R. Lee, "Solving the subcircuit extraction problem by using bit-parallel filtering algorithm," in *30th Workshop on Combinatorial Mathematics and Computation Theory*, pp. 116–123, 2013.

[49] N. Zhang, D. C. Wunsch, *et al.*, "Speeding up vlsi layout verification using fuzzy attributed graphs approach," *Fuzzy Systems, IEEE Transactions on*, vol. 14, no. 6, pp. 728–737, 2006.

[50] C. L. Hung, H. H. Wang, C. T. Fu, and C. S. Ou, "Parallel subcircuit extraction algorithm on gpgpus," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on*, pp. 1248–1252, IEEE, 2014.

[51] C.-L. Hung, C.-Y. Lin, C.-S. Ou, Y.-H. Tseng, P.-Y. Hung, S.-P. Li, and C.-T. Fu, "Efficient bit-parallel subcircuit extraction using cuda," *Concurrency and Computation: Practice and Experience*, 2015.

[52] M. Takashima, A. Ikeuchi, S. Kojima, T. Tanaka, T. Saitou, and J.-i. Sakata, "A circuit comparison system with rule-based functional isomorphism checking," in *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pp. 512–516, IEEE Computer Society Press, 1988.

[53] S. Eckmann and G. Chisholm, "Assigning functional meaning to digital circuits," tech. rep., Argonne National Lab., IL (United States), 1997.

[54] F. Luellau, T. Hoepken, and E. Barke, "A technology independent block extraction algorithm," in *Design Automation, 1984. 21st Conference on*, pp. 610–615, IEEE, 1984.

[55] T. Doom, J. White, A. Wojcik, and G. Chisholm, "Identifying high-level components in combinational circuits," in *VLSI, 1998. Proceedings of the 8th Great Lakes Symposium on*, pp. 313–318, IEEE, 1998.

[56] J. L. White, A. S. Wojcik, M.-J. Chung, and T. E. Doom, "Candidate subcircuits for functional module identification in logic circuits," in *Proceedings of the 10th Great Lakes Symposium on VLSI*, GLSVLSI '00, (New York, NY, USA), pp. 34–38, ACM, 2000.

[57] J. White, M. Chung, A. Wojcik, and T. Doom, "Efficient algorithms for subcircuit enumeration and classification for the module identification problem," in *Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on*, pp. 519–522, 2001.

[58] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, no. 3, pp. 72–80, 1999.

[59] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. 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.

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

[61] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. 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*, pp. 67–74, IEEE, 2013.

[62] E. Goldberg, M. Prasad, and R. Brayton, "Using sat for combinational equivalence checking," in *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pp. 114–121, 2001.

[63] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," *Handbook of knowledge representation*, vol. 3, pp. 89–134, 2008.

[64] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, pp. 836–843, Nov 2006.

[65] 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.

[66] K.-H. Chang, D. Papa, I. Markov, and V. Bertacco, "Invers: An incremental verification system with circuit similarity metrics and error visualization," pp. 487–494, March 2007.

[67] R. P. Sheridan and S. K. Kearsley, "Why do we need so many chemical similarity search methods?," *Drug discovery today*, vol. 7, no. 17, pp. 903–911, 2002.

[68] J. W. Raymond and P. Willett, "Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2d chemical structure databases," *Journal of computer-aided molecular design*, vol. 16, no. 1, pp. 59–71, 2002.

[69] J. W. Raymond, E. J. Gardiner, and P. Willett, "Rascal: Calculation of graph similarity using maximum common edge subgraphs," *The Computer Journal*, vol. 45, no. 6, pp. 631–644, 2002.

[70] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola, "Learning graph matching," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 6, pp. 1048–1058, 2009.

[71] N. S. Ketkar, L. B. Holder, and D. J. Cook, "Empirical comparison of graph classification algorithms," in *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*, pp. 259–266, IEEE, 2009.

[72] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[73] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[74] H. Tamada, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Design and evaluation of birthmarks for detecting theft of java programs.," in *IASTED Conf. on Software Engineering*, pp. 569–574, 2004.

[75] 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.

[76] X. Zhou, X. Sun, G. Sun, and Y. Yang, "A combined static and dynamic software birthmark based on component dependence graph," in *Intelligent Information Hiding and Multimedia Signal Processing, 2008. IIHMSP'08 International Conference on*, pp. 1416–1421, IEEE, 2008.

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

[78] D. Qiu, H. Li, and J. Sun, "Measuring software similarity based on structure and property of class diagram," in *Advanced Computational Intelligence (ICACI), 2013 Sixth International Conference on*, pp. 75–80, IEEE, 2013.

[79] 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.

[80] C. Wolf, "Yosys open synthesis suite." http://www.clifford.at/yosys/, 2013.

[81] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 10, pp. 1367–1372, 2004.

[82] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, pp. 31–42, Jan. 1976.

[83] Y. Tian, R. Mceachin, C. Santos, J. Patel, *et al.*, "Saga: a subgraph matching tool for biological graphs," *Bioinformatics*, vol. 23, no. 2, pp. 232–239, 2007.

[84] S. Zhang, S. Li, and J. Yang, "Gaddi: distance index based subgraph matching in biological networks," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pp. 192–203, ACM, 2009.

[85] B. Messmer and H. Bunke, "Efficient subgraph isomorphism detection: a decomposition approach," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 12, pp. 307–323, Mar 2000.

[86] P. J. Durand, R. Pasari, J. W. Baker, and C.-c. Tsai, "An efficient algorithm for similarity analysis of molecules," *Internet Journal of Chemistry*, vol. 2, no. 17, pp. 1–16, 1999.

[87] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 10, pp. 1367–1372, 2004.

[88] Boost, "http://www.boost.org,"

[89] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc: towards an open-source tool flow," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 41–44, ACM, 2011.

[90] Unity3D, "http://unity3d.com/,"

[91] C. Albrecht, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.

[92] A. Bender and R. C. Glen, "Molecular similarity: a key technique in molecular informatics," *Organic & biomolecular chemistry*, vol. 2, no. 22, pp. 3204–3218, 2004.

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

[94] J. Hert, P. Willett, D. J. Wilton, P. Acklin, K. Azzaoui, E. Jacoby, and A. Schuffenhauer, "Comparison of fingerprint-based methods for virtual screening using multiple bioactive reference structures," *Journal of chemical information and computer sciences*, vol. 44, no. 3, pp. 1177–1185, 2004.

[95] A. Mishchenko, S. Chatterjee, R. Brayton, X. Wang, and T. Kam, "Technology mapping with boolean matching, supergates and choices," 2005.

[96] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "Fraigs: A unifying representation for logic synthesis and verification," tech. rep., ERL Technical Report, 2005.

[97] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*, vol. 2. Springer Science & Business Media, 1984.

[98] J. Travis and J. Kring, *LabVIEW for Everyone: Graphical Programming Made Easy and Fun (National Instruments Virtual Instrumentation Series)*. Prentice Hall PTR, 2006.

[99] S. Cesare and Y. Xiang, *Software similarity and classification*. Springer Science & Business Media, 2012.

[100] K. Zeng and P. Athanas, "Discovering reusable hardware using birthmarking techniques," in *Information Reuse and Integration (IRI), 2015 IEEE International Conference on*, pp. 106–113, Aug 2015.

[101] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[102] "Altera examples," 2015.

[103] "Asic world," 2015.

[104] "Doulous," 2013.

[105] "Dsp examples," 2015.

[106] F. Dexter, A. Macario, R. D. Traub, M. Hopwood, and D. A. Lubarsky, "An operating room scheduling strategy to maximize the use of operating room block time: computer simulation of patient scheduling and survey of patients' preferences for surgical waiting time," *Anesthesia & Analgesia*, vol. 89, no. 1, pp. 7–20, 1999.

[107] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "Letor: Benchmark dataset for research on learning to rank for information retrieval," in *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pp. 3–10, 2007.

[108] "Ncsu," 2015.

[109] "Patchell IP archive," 2015.

[110] "Referencevoltage," 2015.

[111] "Spiral," 2015.

[112] G. Qian, S. Sural, Y. Gu, and S. Pramanik, "Similarity between euclidean and cosine angle distance for nearest neighbor queries," in *Proceedings of the 2004 ACM symposium on Applied computing*, pp. 1232–1237, ACM, 2004.

[113] K. Zeng and P. Athanas, "A q-gram birthmarking approach to predicting reusable hardware," in *Design Automation and Test in Europe (DATE), 2016 IEEE International Conference on*, March 2016.

[114] A. Aiken *et al.*, "Moss: A system for detecting software plagiarism," *University of California–Berkeley. See www. cs. berkeley. edu/aiken/moss. html*, vol. 9, 2005.

[115] X. Zhao, C. Xiao, X. Lin, and W. Wang, "Efficient graph similarity joins with edit distance constraints," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pp. 834–845, IEEE, 2012.

[116] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao, "Graph similarity search with edit distance constraint in large graph databases," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pp. 1595–1600, ACM, 2013.

[117] G. Wang, B. Wang, X. Yang, and G. Yu, "Efficiently indexing large sparse graphs for similarity search," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, pp. 440–451, March 2012.

[118] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: on approximating graph edit distance," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.

[119] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences 1997. Proceedings*, pp. 21–29, IEEE, 1997.

[120] "Kitiyo," 2015.

[121] "Scu-rtl benchmark," 2015.

[122] "Trust hub." https://www.trust-hub.org/resources/benchmarks, 2015.

[123] "Rfwireless." http://www.rfwireless-world.com/source-code/VERILOG/Low-pass-FIR-filter-verilog-code.html, 2015.

[124] Z. Su, Q. Yang, Y. Lu, and H. Zhang, "Whatnext: A prediction system for web requests using n-gram sequence models," in *Web Information Systems Engineering, 2000. Proceedings of the First International Conference on*, vol. 1, pp. 214–221, IEEE, 2000.

[125] D. Anselmo and H. Ledgard, "Measuring productivity in the software industry," *Communications of the ACM*, vol. 46, no. 11, pp. 121–125, 2003.

[126] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE Transactions on Software Engineering*, vol. SE-9, pp. 639–648, Nov 1983.

[127] V. R. Basili, L. C. Briand, and W. L. Melo, "How reuse influences productivity in object-oriented systems," *Commun. ACM*, vol. 39, pp. 104–116, Oct. 1996.

[128] A. J. Albrecht, "Measuring application development productivity," in *Proceedings of the joint SHARE/GUIDE/IBM application development symposium*, vol. 10, pp. 83–92, 1979.

[129] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.