Review of Logic Synthesis Methods

Jazmine Boloor

Electrical and Computer Engineering Dept.

University of Nevada

Las Vegas, USA

***Abstract* — The goal of this research is to review logic synthesis methods and techniques that are available for different uses/mediums to provide a background on some of the available methods. The following sections will introduce and review ESPRESSO, PALMINI, and Quine-McCluskey methods of logic minimization, as well as examine different takes on reversible and scalable logic synthesis methods. The presented methods will be described, examined, and compared/contrasted among themselves to convey optimal usage for each. This paper will also analyze the future of logic synthesis and introduce upcoming potential ideas that are currently being researched and tested.**

Index Terms — Logic Synthesis, PALMINI, ESPRESSO, Quine-McCluskey, Scalable Logic Minimization

I. INTRODUCTION

The age of technology has opened the doors to an abundance of electronic/computer designs. It is intuitive to assume that the application of each of these designs may need to be described in different terms/languages. Logic synthesis is the process of automatic production of logic components, in particular digital circuits, thus synthesizing a design in terms of logic gates and components [1]. The functionality of transforming code from HDL into a netlist (a schematic, text file, ext.) that can present the hardware through logic gates that are connected by wires is a powerful tool that can allow for better optimization and performance in many systems. It is the means by which a register transfer level (RTL) of a design can be transformed into a netlist/gate specification [2]. Thus, efficient methods to do so have been relevant for quite some time.

The concept of logic synthesis has been studied for decades, and pioneering methods of minimization have paved the way for new and improved ones to come to light. The following sections of this paper will introduce and examine different synthesis methods and ideas, as well as compare/contrast them to each other.

II. ESPRESSO LOGIC MINIMIZATION

A heuristic logic synthesis tool that has been studied with several different derivatives is called ESPRESSO logic minimization. The ESPRESSO algorithm provides a simple method to optimize minimization through the use of binary encoding [3]. Furthermore, this method of synthesis provides an efficient support system for a large number of inputs and outputs, allowing for compatibility with a wide range of electronic and computer designs [2].

The ideology behind the ESPRESSO logic minimization method surrounds data coding, specifically binary encoding that is proceeded by data decoding after an algorithm of minimization is performed. The main steps in this method are expansion, irredundant covering, and reduction [2]. Expansion turns any implicits into prime implicits by building onto its original form. The irredundant covering phase then analyzes the prime implicits to remove any that are redundant - thus, “covering” those implicits that can be minimized and sorting them out [3]. Moreover, the actual sorting is done in the reduction phase. In this phase, the implicits are given weights, with the heaviest weight being given to the larger ones. These larger weights are then the first to be sorted, as they are the most likely to be overlapping another term [3]. This loop is iterated until the initial input is minimized to its smallest form, and it has a fairly high accuracy rate [3].

It is important to note that there have been several different derivatives of the ESPRESSO logic minimizers. This fast, two level minimization has many different versions. The family includes notable releases including the ESPRESSO-II, ESPRESSO Signature, ESPRESSO-Exact, and ESPRESSO-MV algorithms that have helped shape this heuristic approach to minimization [2]. As assumed with the title of it, this is an approximate method to minimization. Though this algorithm allows for a large number of variables, it comes at the cost of its speed and accuracy [3]. For smaller inputs, the method can usually accurately reach a minimal solution fairly quickly, but the con associated with having a larger input number, (100+), is a larger delay time and therefore a larger runtime [3]. Though this delay is introduced, it is not unreasonable for its use in many scenarios that do not prioritize time. For example, the algorithm was run with a Sum-of-Products input that has 3172 terms (12741 literals), and the result was accurately displayed after sixteen CPU seconds (in 1984) [4].

III. PALMINI LOGIC MINIMIZATION

Like the ESPRESSO method discussed in the previous section of this paper, the PALMINI logic minimization technique is an approximate method for synthesis. It should be noted that the PALMINI minimizer was introduced in the 1980s - much of its study was conducted decades ago [5]. Other methods of logic synthesis have since come about that prove to be more efficient in a variety of cases. Nonetheless, it paved the way for other techniques and shows importance in the history of logic synthesis. Thus, the study of the methodology and its implementation is presented and analyzed in this section.

PALMINI relies on the concept of minimal implicits. Moreover, the methodology of this algorithm surrounds the creation of the “graph of incompatibility of implicants” [6]. The incompatibility was tested through the analysis of the implicits - those with two or more differing minterms were considered to be incompatible [6]. These incompatible minterms were then used to solve the graph coloring problem, where the nodes of the graph correspond to minterms.

As mentioned previously, PALMINI optimizes graph coloring to perform logic synthesis, and it can do so in an exact way and in an approximate way [5]. An example an approximate methods graph of incompatibility of implicants is illustrated in the graph of incompatibility of implicants in Figure 1, where three colors are chosen. The given function is:

y = {x1, x2, x3, x4}={0000, 0011, 0100, 1111}

and thus the matching of the nodes creates the graph shown [5].

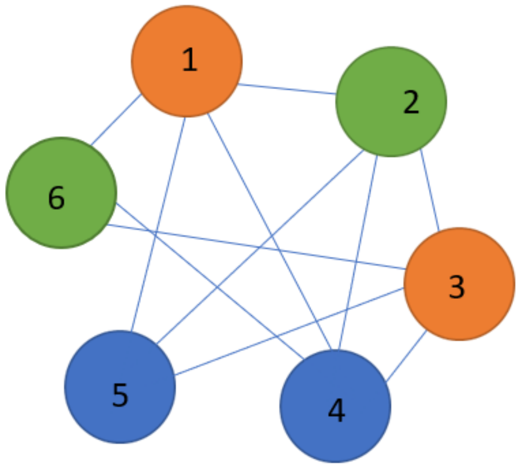


Figure 1 - The Graph of Incompatibility of Implicants for the Above Problem

Through color matching the minterms, the output will correspond to:

y = {x1, x2, x3, x4}={0-00, -011, 11-1}

Thus, the original set has been minimized using the PALMINI technique. Note that the dashes in the output function represent don’t cares. Also note that the lines on the graph correspond to different nodes in the logic system.

The nodes on the graph can represent different parts of the network, such as the minterms, disjoint product implicants, minimum implicants or minimal product implicants [5]. This can then be denoted as SOP or POS format [5].

IV. SCALABLE LOGIC SYNTHESIS

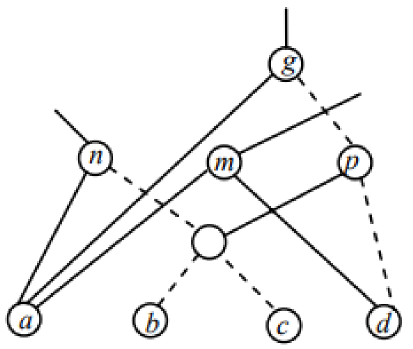
Scalable logic synthesis methods allow for a set of logic data to be represented in a much larger network. Thus, allowing for the "scalability" of the data to a different size, which is especially useful in current times when logic-based devices are constantly changing and expanding. The following paragraphs will analyze peephole optimization and Boolean resubstitution as methods of scalability.

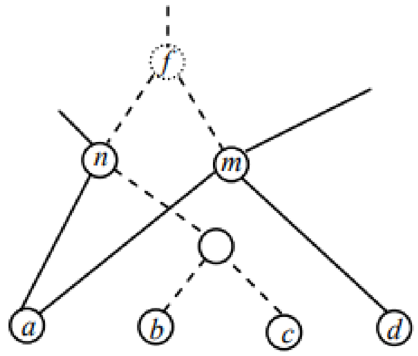
Peephole optimization presents a method of scalability that surrounds looping through sectioned parts of a logic network and examining patterns that they may have [7]. This can be done using a sliding window technique, which loops through different parts of the netlist and analyzes the categories dependendent on its movement. These components can then be replaced as necessary to help with optimization [8]. This is an exhaustive technique - thus it will continue running as long as it still assumes it has optimization left to do [7].

Boolean methods of logic synthesis introduce many methods of optimization that have been designed to be scalable. Boolean methods for logic synthesis, in particular, are closely examined because of their ability to provide very accurate results (at the price of the system's monetary cost and complexity) [9]. Furthermore, Boolean methods of minimization can be costly because of their reliance on full networks (including don't-cares) - scalability can minimize this cost because of its nature to work with smaller pieces of networks rather than the entire network [9].

Several methods of scalable Boolean methods have been proposed, all based on the general concept listed above. One example is the resubstitution - the theory behind this method lies in its transformative nature. This method surrounds finding the Boolean difference between two nodes so as to find commonalities that can lead to covered terms [10]. Boolean resubstitution allows for a new representation of a component in a logic network that is based on existing components in the network. These methods are distinguished by their number of operators, denoted by k, which are tools that aid in the remodeling of the logic network [7]. Thus, minimizing the network in a way that is scalable to larger systems.

Another AIG-based implementation uses a resubstition algorithm to express the same network with a smaller amount of nodes [11]. The algorithm is written so as to analyze the original system to a proposed better (smaller) system. If the system is deemed better in terms of synthesis, it is the new version of the system that the algorithm will compare to. Figures 2a and 2b show this occurring. The first figure of the two shows eight nodes. The second of the two shows the system after it run through an AIG resubstiution algorithm.





Figures 2a and 2b - Showing AIG resubstitution

It is clear that the new system has one less node than the first - this occurred by covering the g and p node with the f node [11]. The system is clearly better minimized because it has less nodes.

Another method of scalable Boolean simplification consists of an And-Inverter Graph (AIG) optimization. This can make use of a waterfall based model, as seen in Testas article [10]. Using this model within this optimization tool guarantees that once a successful move is shown, no other moves will be tried. In this model, there is a tradeoff between runtime and accuracy. Moreover, as the system will run faster within this model, it may not always pick the optimal quality of results. Nonetheless, this particular study listed the waterfall model as a “good tradeoff” between the speed and quality of results for the application in question [10]. The methodology behind this gradient-based AIG system is that it studies actions such as rewriting, refactoring, and resubstituting and places a weight on each action. Low weights (which are all but rewriting) are looped through until the gain of the network is greater than one. When the gain exceeds one, the AIG engine enters the method. There is a “cost budget” associated with the gradient that decides the amount of actions that can be performed (say, *k i*terations) [10]. Thus, the gradient-based AIG method will loop through optimization steps until the gain gradient is greater than one.

V. REVERSIBLE LOGIC SYNTHESIS

Reversible logic synthesis revolves around gates that have the same number of inputs and outputs. With this type of logic synthesis, the 1:1 ratio of features creates the possibility of unused pins - these are referred to as garbage [12]. It is integral to the design of a reversible logic system that each of the gates used within it is also reversible, so as to keep the 1:1 (or k:k) ratio [12]. There are significant reasons for the optimization of reversible logic, the most clear of which relating to energy dissipation. In fact, there is research suggesting that in order for energy dissipation to not occur, the circuit must be built from reversible gates [12]. The main goal in the optimization of reversible logic synthesis is to minimize the number of these garbage signals.

Much of the remodeling of circuits in reversible logic applications surrounds making a k:n ratio into a k:k ratio. This can be done through a variety of methods. An approach of reversible logic synthesis that focuses on not only reducing the number of garbage signals, but also the total delay of the system, was explored by Portland State University researchers [12]. The methodology of the method presented surrounds taking into consideration arbitrary circuits. These circuits may or may not have the one-to-one ratio that is necessary for reversible circuits, so it is to be analyzed and remodeled (if necessary) to a reversible layout [12].

Binary reversible gates present a method of representing networks in a reversible manner. Since there is only one classic gate - the NOT gate - that is traditionally known as a one-to-one gate, methods for representing the AND, OR, XOR, NAND, ect. gates is integral creating basic logic networks.. Three well known reversible gates are the Feynman gate (also called the Controlled-Not (CNOT) gate), the Fredkin gate, and the Toffoli gate (also called the 3:3 Feynman gate or Controlled-Controlled Not (CCNOT) gate) [12]. It should be noted that the NOT gate is integral to the design of networks - all of the classic gates can be represented through an placement of these reversible gates and NOT gates. Again, this is due in part to the classic design of the NOT gate having a 1:1 design, as it is already a reversible gate.

Starting with discussion on the Feynman gate, this gate is a k:k logic tool that allows for fanning out, as the P input takes the form of A and the Q output takes the XOR of its two inputs.. To be more specific, if inputs A and B are 0 and 1, respectively, outputs P and Q are 1 and 1; if inputs A and B are 1 and 1, respectively, outputs P and Q are 1 and 0 [12]. The truth table and a general block diagram for this 2:2 implementation of the 2:2 Feynman gate is shown in Table 1 and in Figure 3 below.

AB	PQ
00	00
01	01
10	11
11	10

Table 1 : Truth Table for the 2:2 Feynman Gate

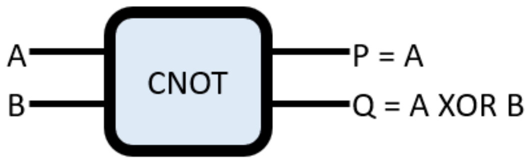
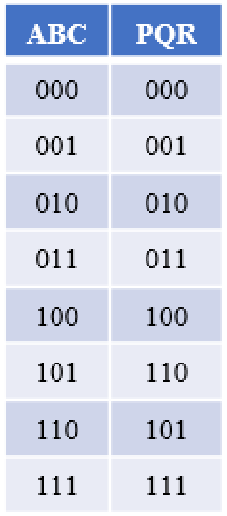


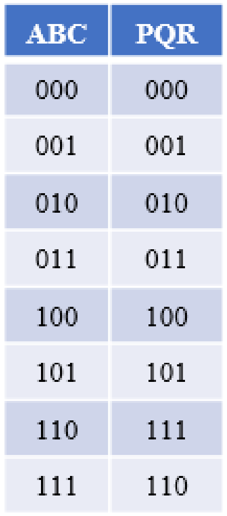
Figure 3: Block Diagram of the Feynman 2:2 (CNOT) Gate

The Fredkin and Toffoli gates are standard 3:3 reversible gates. The logic behind the Fredkin gate is simple - it just involves two multiplexers. The [12] research describes this gate as a “permutation gate” - it permutes the input data through the multiplexers. The use of a control input aids the system in making logic decisions, and this input is also propogated though to an output of the gate [12]. The simplicity of the design of the gate allows for a wider range of its usage. Similarly, the Toffoli gate allows for two inputs to propagate through the gate (thus, Fredkin is a one-through gate and Toffoli is a two-through device). This allows for the change in only one of the three inputs, which can be used in designs in conjunction with the Fredkin (and NOT gates) to create a wide range of networks, as they can perform most logic functions in some capacity.

The truth table of the 3:3 Fredkin and Toffoli gates are shown in tables 2a and 2b, respectively. Note that there are inputs that are identical to the outputs, as these are the ones that “permutate” through the logic gate [12]. Also note the similarities in the first five binary number inputs - they produce the same outputs in both of the gates given the same inputs (which again, could be used to its advantage from the design perspective).



ABC	PQR
000	000
001	001
010	010
011	011
100	100
101	110
110	101
111	111



ABC	PQR
000	000
001	001
010	010
011	011
100	100
101	101
110	111
111	110

Tables 2a and 2b: Truth tables for the Fredkin (a) and Toffoli (b) Gates

Thus, the two gates differ only by their 101,110, and 111 inputs. As shown by the table, the input/output relationships for the Fredkin gate can be characterized by the following three output equations [13]:

$$P = A$$

$$Q = A'B + AC$$

$$R = AB + A'C$$

Similarly, the input/output relationships for the Toffoli gate can be characterized by the following three output equations [14] :

$$P = A'$$

$$Q = B'$$

$$R = C \text{ XOR } B$$

Again, notice the passing of one input to output in the Fredkin vs the passing of two inputs to output in the Toffoli.

The use of these reversible logic gates is powerful in the creation of remodeled circuits that lack the dissipation of power. One important feature that makes these gates useful is that they are their own inverses (similar to the NOT gate), thus allowing what is known as Forward and Backward Synthesis modes [12]. This allows for the cascade of two of the same gate to produce an identity function, which can be useful in many logic processes. This is a process that is specific to reversible logic gates (as opposed to standard logic), making them all the more powerful in the right instances [12].

VI. SPECTRAL TECHNIQUES

Spectral methods allow for a logic synthesis technique that allow for minimization in a different space of representation. The authors of [2] relate this to the Fourier transform - a way of working with things in a different space. The spectral coefficients can be used for this. The examination of the autocorrelation coefficients in Boolean functions can be used in the adaptation of a minimization technique [16]. To do this, the network will be analyzed to decide whether or not decomposition is possible with this technique. Then, the spectral techniques are used to search for related subfunctions [16].

An issue that was present in past decades surrounded finding the spectra of larger logic functions - methods were proposed that could compute these spectra so that these methods could be used [17]. Since Boolean functions can notoriously have very large equations that describe them, a transformation that minimizes its complexity is necessary [16]. Many transform domains exist and have been proposed - Walsh and Reed-Muller are the most popular. Both techniques make use of matrix multiplication to find the spectral data [17]. The Walsh transformation, in particular, was proposed for a variety of different spectral techniques. This method takes the Walsh function, which is defined by the following equation:

$$W = -\sum wi * xi$$

where i is the bit position and x and w are the function vectors [17]. Once this is computed, it can be used to find the Walsh transform of a given function f(x), where [17]:

$$f(w) = \sum [W * f(x)]$$

Once the information regarding the spectra of the Boolean function is obtained, it is examined for patterns. These patterns can surround the similarity, difference, or logic-based analysis of the terms. Additionally, don't care values can be transformed into a domain that can compare and minimize them.

VII. QUINE-MCCLUSKEY EXTENSIONS

Quine-McCluskey minimization is an exhaustive minimization technique that is well known by most undergraduate electronics and computer engineering students. However, the classic method behind this technique does not suit digital circuits very well - even simple ones such as adders and parity checkers, due to the XOR gate [15]. Extensions to the way Quine-McClusky is typically implemented, such as the inclusion of the XOR gate and the application of reversible gates, can allow for the method to be better suited to a wide range of applications.

The inclusion of the XOR gate can allow for the Quine-McCluskey method of minimization to be implemented into a wider range of applications. As described by Turton [15], a proposed technique for this includes similar notation to that of what is already used, allowing for simplicity in usage. The idea behind this proposal surrounds the comparison of two minterms in the minimization process. For the case of four term minterms: if the comparison between exactly two terms of the minterm are identical, and the two terms that are different are compliments of each other, then the XOR is present in the function [15]. To illustrate this more clearly, the following two minterms can be examined:

Minterm 1 = A'BCD'

Minterm 2 = A'BC'D

Notice that the first two terms in both of the minterms are identical. This satisfies the first of the two criteria for there to be an XOR present. The second two terms are different, but only by compliments of each other. Thus, both of the criteria are met and an XOR can represent these two minterms.

Another extension on the Quine-McCluskey algorithm surrounds reversible logic (see section of paper on reversible logic synthesis). In this application, the Quine-McCluskey algorithm is executed, and then the common terms in the prime implicants are found. Then, the number of terms that can be covered is found, and the algorithm repeats until none can be found. At this point, the XOR operation is performed on the expression, ending the algorithm. Not only does this method allow for its implementation in reversible logic networks, but the study also found that the speed and accuracy of the method were improved in comparison to similar methods [1].

VIII. FUTURE OF LOGIC SYNTHESIS

The future of logic synthesis surrounds the same idea that it did decades ago: finding the optimal solution for logic networks. Recent times have introduced many new ideas for logic synthesis. The rapid advancement and seemingly ever-growing nature of the semiconductor industry, the advancement of system-on-chip integrated circuits

(SoC IC) has seen exponential growth [18]. The chips on these systems are generally designed to be very small - so as to not take up much area on the device it is attached to (say, a cell phone). This leads to the observation that the optimal logic performance is absolutely necessary. The authors of [18] have created a method called Progressive Automated Logic Synthesis (PALS) that is specifically designed for these very small chips [18]. In general, much of the future of logic synthesis will revolve around methods to enhance the semiconductor industry that deals with small real estates.

Another new method for logic synthesis, presented in January, 2022, surrounds the idea of approximating logic synthesis using Boolean Matrix Factorization [19]. This done through an algorithm that approximates the input circuit. The proposed method took into account scalability, versatility, and optimal runtime. It does not, however, account for reversibility, so it cannot be used in reversible logic circuits. All in all, this is a method that is designed for hardware metrics that have an unstrict requirement on a full range of accuracy [19]. This offers another option for designs that can list accuracy as a design tradeoff.

IX. CONCLUSION

This paper has reviewed and analyzed several different logic synthesis techniques. ESPRESSO, PALMINI, and Quine Mccluskey methods were explored, as well as methods relating to reversible and spectral logic techniques. The general/practical uses of the techniques were discussed, and the optimal applications were references.

REFERENCES

[1] S. Zhao, C. Wang and J. Sun, "Improvement and implementation of quine-McCluskey algorithm for synthesis of reversible logic circuits," 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2017, pp. 640-642.

[2] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, “Multilevel logic synthesis,” Proceedings of the IEEE, 01-Jan-1990.

[3] Ashmouni, Elhoussini & Ramadan, Rabie & Rashed, Ali. (2014). Espresso for Rule Mining. Procedia Computer Science. 32. 596-603.

[4] R. Rutenbar, “((lec 6) 2-level minimization: Basics - ece:course page,” 2-Level Minimization, 2001

[5] Nguyen, L. B., Perkowdki, M. A., & Goldstein, N. B. (1987, October 1). *Palmini-Fast Boolean minimizer for personal computers: Proceedings of the 24th ACM/IEEE Design Automation Conference.*

[6] N. Gorji and S. I. Poon, “Logic synthesis as an efficient means of minimal model ,” 2018.

[7] H. Riener et al., "Scalable Generic Logic Synthesis: One Approach to Rule Them All," 2019 56th ACM/IEEE Design Automation Conference (DAC), 2019, pp. 1-6.

[8] T. Chelcea and S. M. Nowick, "Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems," Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324), 2002, pp. 405-410.

[9] E. Testa et al., "Extending Boolean Methods for Scalable Logic Synthesis," in IEEE Access, vol. 8, pp.

[10] E. Testa et al., "Scalable Boolean Methods in a Modern Synthesis Flow," 2019 Design, Automation & Test in Europe Conference & Exhibition, 2019, pp. 1643-1648.

[11] Mishchenko , A., & Brayton, R. (n.d.). *Scalable logic synthesis using a Simple Circuit Structure*. EECS at UC Berkeley.

[12] Perkowski, Marek, Lech Jozwiak, Pawel Kerntopf, Alan Mishchenko, Anas Al-Rabadi, Svetlana N. Yanushkevich, Vlad P. Shmerko, and Malgorzata Chrzanowska-Jeske. "A General Decomposition for Reversible Logic." (August 2001).

[13] Dadjouyan, Ali & Sayedsalehi, Samira & Faghih Mirzaee, Reza & Jafarali Jassbi, Somayyeh. (2020). Design and evaluation of clocked nanomagnetic logic conservative Fredkin gate. Journal of Computational Electronics.

[14] Liu, Wenjie & Xu, Yinsong & Liu, Wen & Wang, Haibin & Lei, Zhibin. (2019). Quantum Searchable Encryption for Cloud Data Based on Full-Blind Quantum Computation. IEEE Access.

[15] B. C. H. Turton, "Extending Quine-McCluskey for Exclusive-Or logic synthesis," in IEEE Transactions on Education, vol. 39, no. 1, pp. 81-85, Feb. 1996.

[16] Porwik, Piotr & Stankovic, Radomir. (2006). Dedicated spectral method of Boolean function decomposition. Int. J. Appl. Math. Comput. Sci. 16. 271-278.

[17] Tsai, Y. T., D'Luna, L. J., & K., L. P. P. (1992). Efficient Spectral Techniques for Logic Synthesis. Institute of Electrical and Electronics Engineers.

[18] R. Li, T. Li, K. Liu, H. Man, X. Zou and C. Wang, "A Progressive Automatic Logic Synthesis Method for Timing-Limited SoC Design," 2021 6th International Conference on Integrated Circuits and Microsystems (ICICM), 2021, pp. 227-231.

[19] J. Ma, S. Hashemi and S. Reda, "Approximate Logic Synthesis Using Boolean Matrix Factorization," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 1, pp.