Novel Primitives for Logic Synthesis

Winston Haaswijk

LSI, I&C, EPFL

***Abstract*—In this report we examine the use of novel logic primitives in the context of Logic Synthesis. We look at both practical and theoretical arguments for doing so. On the practical side, the use of the majority primitive has recently been shown to offer significant improvements over the state of the art in synthesis and optimization. Moreover, emerging nanotechnologies often natively support unconventional primitives. On the theoretical side, we will see how certain primitives can be shown to be less expressive than others. Based on these arguments we propose a research project that explores novel logic primitives and applies them to Logic Synthesis. The short-term goal of this project is to apply novel primitives to technology mapping.**

***Index Terms*—thesis proposal, candidacy exam write-up, EDIC, EPFL, Logic Synthesis, majority logic, MIG**

I. Introduction

IT is well known that in recent years the progress of Moore's law has been facing challenges. The continuous shrinking of transistors is pushing the limits of modern CMOS technology. Both performance and energy consumption are of increasing concern. The end of downward power density scaling has led to what some call the *Post-Dennard* era [1]. Several solutions have been proposed to reverse this trend. One approach has been to emphasize parallelism in both hardware and software architectures. Another has been in the area of emerging nanotechnologies [2], [3]. Yet other approaches have focused on the flexible integration of configurable hardware (such as FPGAs) with software components [4].

The research area of *Electronic Design Automation* (EDA) is noticeably affected by the recent changes in digital technology, as it has to adapt to new technologies when they are discovered. Within EDA, it is the goal of *Logic Synthesis* to develop efficient methods for the representation and optimization of Boolean functions. Note that we use the term Boolean function in this write-up to refer to general Boolean functions. Such functions are mappings between Boolean spaces of the form $f : \mathbb{B}^m \to \mathbb{B}^n$. They are also referred to as *multiple output Boolean functions*. Multiple output Boolean functions are used as an abstract model for digital circuits.

The inputs for a Logic Synthesis system are specifications of Boolean functions. The task of such a system is to provide representations for the functions that can be efficiently optimized, such that a near-optimal (physical) circuits can be synthesized in an automated way. Optimality of a circuit can be viewed in terms of its size, delay, and power consumption. This optimality has always been of prime concern to logic synthesis, and any method that can improve it is of interest to the EDA community. However, traditional methods are not always optimal in the context of new problems that have arisen in recent years. Therefore, the continued success of Logic Synthesis in the design and fabrication of computer hardware depends on our ability to improve and adapt its methods to changing circumstances.

The goal of this report is to give a motivation for, and examples of, the use of novel Boolean primitives for Logic Synthesis. The advantages of using new logic primitives are twofold. Firstly, these primitives can be used to improve traditional Logic Synthesis flows. Secondly, several emerging nanotechnologies, such as *Resistive RAM* (RRAM), *Nanoelectromechanical Relays* (NEM Relays), and *Carbon Nanotubes*, natively support non-traditional logic primitives [5], [6]. By supporting these logic primitives we may improve upon the state of the art and continue to use automated Logic Synthesis for the creation of efficient circuits in the future.

We consider three papers to support the argument for novel logic primitives. The first paper, "Majority-Inverter Graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization", introduces the concept of majority logic as a basis for Boolean function representation and optimization. We examine the advantages that the majority primitive offers over traditional logic representations. In the second paper, "Parity, Circuits, and the Polynomial-Time Hierarchy" we examine logic primitives in the context of *Circuit Complexity*. We

Proposal submitted to committee: July 27th, 2015; Candidacy exam date: September 29th, 2015; Candidacy exam committee: Viktor Kuncak, Giovanni De Micheli, Pierre-Emmanuel, Paolo Ienne.

This research plan has been approved:

Date: _____

Doctoral candidate: _____
(name and signature)

Thesis director: _____
(name and signature)

Thesis co-director: _____
(if applicable) (name and signature)

Doct. prog. director: _____
(B. Falsafi) (signature)

will see that traditional AND/OR/INV logic primitives cannot compactly represent certain functions (including the parity and majority functions). This is another argument for the use of more expressive primitives. Finally, in the last paper we turn to a practical application of Logic Synthesis: *technology mapping*. In "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", we examine an algorithm for the depth-optimal mapping of bounded Boolean circuits to FPGAs.

II. Majority-Inverter Graphs

The *Majority-Inverter Graph* (MIG) is a data structure for the representation and optimization of Boolean logic. The use of this data structure for the purpose of Logic Synthesis was first proposed in [7]. The novelty of the MIG representation stems from the fact that it it corresponds to a Boolean algebra based on *majority logic*. Traditionally, Logic Synthesis has made use of perhaps more familiar types of logic based on the AND, OR, INV and MUX operations. For example, early data structures were based on *Sum of Product* (SOP) forms (disjunctions of conjunctions). SOP-like representations are commonly referred to as *two-level* representations. *Binary Decision Diagrams* (BDDs) are another example of a traditional representation. BDDs are a canonical representation based on the MUX operator. While their canonicity is desirable, they have the disadvantage that, for certain functions, their size grows exponentially with the number of inputs [8].

The state of the art in Logic Synthesis is based on *multilevel* logic representations [9]. Such representations are commonly known as *Logic Networks* or *Boolean Circuits*. A Logic Network is a *Directed Acyclic Graph* (DAG), in which nodes correspond to logic operators. Edges connect the nodes to their inputs. Typically, the operators used in Boolean networks are the traditional AND, OR, and INV operators. The size of a logic network is its number of nodes. In a logic network, the *depth* of a node is the length of the longest path from a *primary input* to that node. The depth of a primary input is zero. The depth of a logic network is the greatest depth of any node in the network. An example of a type logic network that has been used for Logic Synthesis is the *And-Or-Inverter Graph* (AOIG). A network is called *homogeneous* if all nodes have the same number of incoming edges k (indegree), and if they all represent the same operator. An example of such a network is the *And-Inverter Graph* (AIG), in which every node is an AND operator with $k = 2$. Note that we allow edges to have a complementation attribute, which enables us to represent the inversion operator.

An MIG is a homogeneous logic network with $k = 3$, in which every node represents the M_3 majority operator. In general the n-input majority function is equal to the value given by more than half of its inputs. For example, $M_3(0, 0, 1) = 0$ and $M_3(1, 1, 0) = 1$. MIGs can be shown to contain both AOIGs and AIGs, by biasing certain inputs of the majority nodes [7]. (Note that if we set one of the inputs to 0, the resulting M_3 operation is an AND of the remaining operands.) Hence, any Boolean function can be represented by an MIG. Fig. 1 shows an example of an MIG for the function $f = abcd + (ab + cd)$.

A. A Majority Based Boolean Algebra

To use MIGs for the purposes of Logic Synthesis, we require a method for manipulating them. In principle, it is possible to use traditional AND/OR based techniques to do so. However, those do not enable the full expressivity of the majority operator. For this reason, [7] proposes the use of a Boolean algebra that natively supports the use of majority primitives. The axioms for this algebra can be seen in Fig. 2. Note that we omit the $_3$ subscript, since we may assume that all majority operators in an MIG have 3 inputs. The Boolean algebra is characterized by the 5-tuple $(\mathbb{B}, M, ', 0, 1)$, where ' is the complementation operator.

$$
\boldsymbol{\Omega}\begin{cases}
\textbf{Commutativity} - \boldsymbol{\Omega.C} \\
M(x,y,z) = M(y,x,z) = M(z,y,x) \\
\textbf{Majority} - \boldsymbol{\Omega.M} \\
\begin{cases} \text{if}(x = y)\text{: } M(x,y,z) = x = y \\ \text{if}(x = y')\text{: } M(x,y,z) = z \end{cases} \\
\textbf{Associativity} - \boldsymbol{\Omega.A} \\
M(x,u,M(y,u,z)) = M(z,u,M(y,u,x)) \\
\textbf{Distributivity} - \boldsymbol{\Omega.D} \\
M(x,y,M(u,v,z)) = M(M(x,y,u),M(x,y,v),z) \\
\textbf{Inverter Propagation} - \boldsymbol{\Omega.I} \\
M'(x,y,z) = M(x',y',z')
\end{cases}
$$

Fig. 2. A Boolean algebra based on majority primitives.

In [7], it is shown that the axioms in Fig. 2 induce a Boolean algebra. The proof is based on previous work on median algebras and lattice theory. Any median algebra with elements 0 and 1 that satisfies $M(0, x, 1) = x$ is a distributive lattice [10]. Note that this follows directly from the first axiom. Furthermore, it is straightforward to show that under the axioms we have a *complemented* distributive lattice. Every complemented distributive lattice is a Boolean algebra [11]. Additionally, [7] proves soundness and completeness of the axioms.

There is a one-to-one correspondence between the majority-based Boolean algebra and the representation structure of MIGs. This enables us to manipulate MIGs through the operations defined in the algebra. Furthermore, the soundness of the algebra guarantees that when we manipulate MIGs with the rules given by the axioms, we always create logically equivalent MIGs. Completeness guarantees that *any* valid MIG optimization can in principle be achieved with the axioms.

B. MIG Logic Optimization

We have introduced MIGs as a representation of Boolean functions, as well as a sound and complete algebra for manipulating them. In logic optimization, the goal is to transform MIGs in order to improve some figure of merit. Typically we are interested in decreasing the size, depth, or switching activity of logic networks. Intuitively speaking, we can view decreasing the size of an MIG to decreasing the area of a physical chip. Similarly, decreasing the MIG depth corresponds to improving the delay of a chip. We estimate power consumption via the switching activity of the MIG, which is the probability that nodes will switch between values 0 and 1.

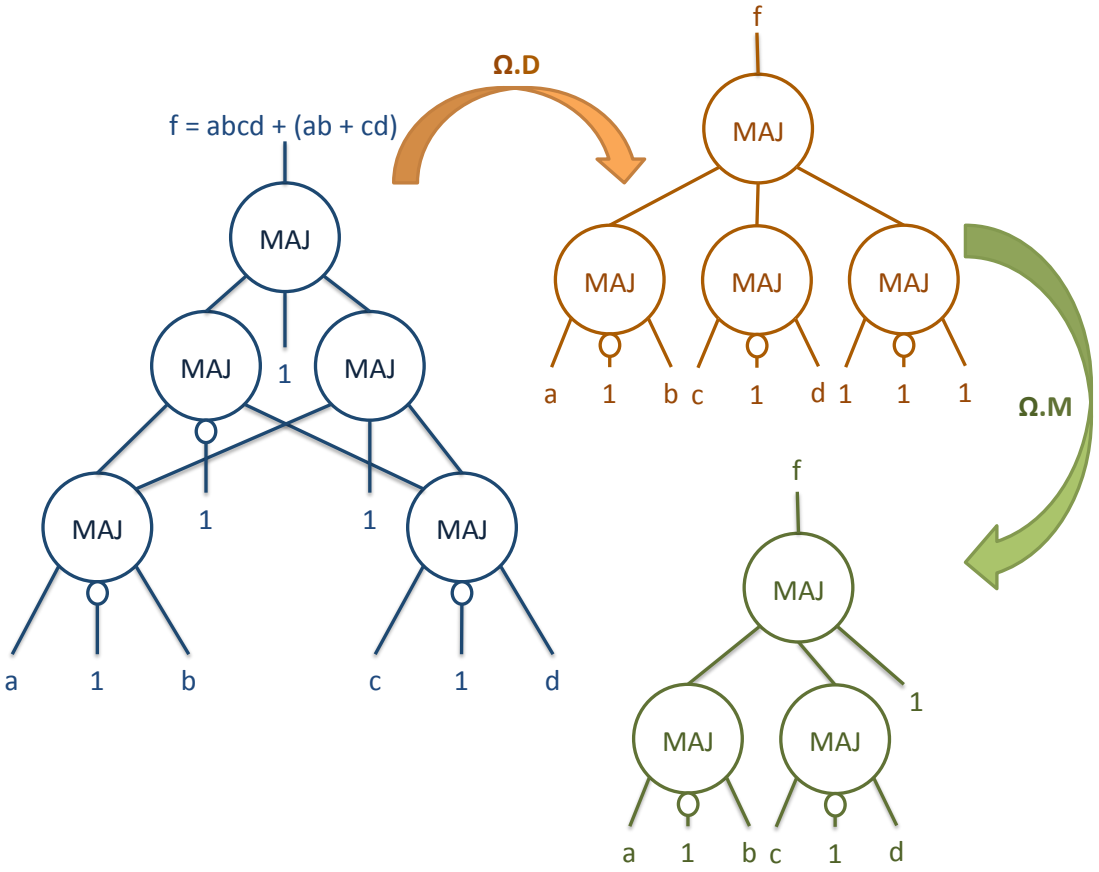

Fig. 1. A MIG for the function $f = abcd + (ab + cd)$. Note that complemented edges are indicated by a "bubble". To optimize the size of the MIG we can apply the distributive $\Omega.D$ rule from right to left in order to eliminate a node. The majority rule $\Omega.M$ can then be applied on the resulting network to eliminate yet another node.

In order to improve the speed of optimization [7] introduces a number of transformations that are derived from the axioms. We can view these compound transformations as sequences of the primitive transformations. The validity of these transformations is guaranteed by the soundness of the axioms.

Optimizing the size of an MIG can be viewed as eliminating nodes from the MIG while preserving its functionality. This is done by finding patterns in the graph that correspond to the transformation rules. Suppose that one of the patterns matching axiom $\Omega.M$ is found. By evaluating this rule from left to right we can eliminate a node from the graph. For example, the node $M(x, x, y)$ can be replaced by x. This creates an equivalent MIG that is smaller than the original one. In a similar way we can evaluate the distributive rule $\Omega.D$ from right to left in order to eliminate a node. See Fig. 1 for an example of this.

Given an MIG, we can reduce its size by applying the rules described above. When no more rules apply, we have either found the optimal size, or we have reached a local minimum. In order to avoid local minima, reshape operations can be applied to the MIG. Reshaping the MIG may lead to new elimination opportunities. The size optimization algorithm works as follows. For a given number of iterations (also called *effort*) it applies the elimination rules, followed by reshaping operations to avoid local minima. The depth and switching activity optimization algorithms are defined analogously.

The quality of a Logic Synthesis method can be measured in terms of how well it optimizes the size, depth, and switching activity of logic networks. In order to evaluate the quality of MIG based optimization and synthesis, [7] presents two experiments:

1) The first experiment compares three methods of logic optimization for a number of large benchmarks from the MCNC benchmark suite. It contrasts MIG optimization with BDD and AIG methods. The average depth of MIGs is 18.6% smaller than AIGs and 23.7% smaller than BDDs. The MIGs are 0.9% bigger than the AIGs on average, but still 2.1% smaller than the BDDs. The average activity of MIGs is 0.3% more than in AIGs, but 3.1% less than in BDDs.
2) The second experiment compares the results for a *optimization-mapping* flow. In other words, in this experiment the benchmark circuits are first optimized and then technology mapping is applied, covering the logic networks with cells from a standard cell library. Physical design is not taken into account. Rather, the area, delay, and power consumption of the synthesis-optimization flow is estimated after technology mapping. Note that these are estimations of physical phenomena, and therefore different from the size, depth, and switching activity measured in the previous experiment. The MIG flow is compared with an AIG flow by the state of the art academic tool ABC and with an unnamed proprietary

synthesis tool. The results show that the MIG flow generates circuits with a 14% decrease in area, a 22% decrease in delay, and a 11% decrease in power.

C. Contributions

The first experiment indicates that MIGs are more effective than BDDs for optimization purposes. Compared to AIGs, the main advantage is in the depth of the optimized networks. Area and activity results are comparable between MIGs and AIGs. Thus, the main advantage in optimization is in the depth of logic networks. Recall that this roughly corresponds to delay.

In the second experiment, MIGs perform better than AIGs and the commercial tool on all metrics. Note, however, that the experiment assumes that the standard cell library contains majority and minority gates. This is not an unreasonable assumption for standard cell libraries that are, for example, based on emerging nanotechnologies. If the library does not contain these gates, applying MIG based synthesis may be less advantageous. However, synthesis based on less expressive primitives (such as AIGs), cannot take advantage of more powerful standard cell libraries in any case.

In summary, this paper introduced the concepts of MIGs and a corresponding Boolean algebra. Its primary contribution is to show that using alternative logic primitives can lead to significant improvements over the state of the art in Logic Synthesis. The expressivity of majority logic allows us to synthesize circuits that are smaller in area, have less delay, and consume less power. This is a clear argument for the application and exploration of alternative logic primitives.

III. Limitations Of Traditional Primitives

The second paper looks into the limitations of fixed depth circuits. This is related to an area known as *Circuit Complexity* (also known as *Boolean Function Complexity*) [12]. Circuit complexity is a research area concerned with finding (strong) upper- and lower bounds on the size or depth of logic networks. In other words, given a Boolean function, circuit complexity is concerned with bounds on the size or depth of the logic networks that implement it. In general, this is a hard problem, and no strong techniques for solving questions of this nature have been discovered [12].

In [13], a superpolynomial lower bound is proved for circuits that implement the parity function. More specifically, it is shown that constant-depth circuits consisting of INV gates and AND/OR gates with unbounded indegree require a more than polynomial number of gates to compute the Boolean parity function. Additionally, this result for constant-depth circuits is extended to the majority, multiplication, and transitive closure functions. A connection to the polynomial-time hierarchy is also made, although that is of less immediate practical concern to the area of Logic Synthesis.

The restriction to constant-depth circuits may seem to weaken the result. Note, however, that in practice many types of circuits can be modeled by constant-depth logic networks. *Programmable logic arrays* (PLAs) are an example of such a class of integrated circuit. Moreover, physical limitations bound the depths of circuits. Finally, there are even limitations for polynomial-size variable-depth circuits. The result in [13] proves that certain functions cannot be efficiently implemented in such circuits. In doing so, it shows the limitations of the traditional INV/AND/OR logic primitives and it makes the notion that certain Boolean functions have greater "expressiveness" than others more tangible.

A. Lower Bound For The Parity Function

In this report we will not repeat the entire proof of the parity lower bound, since the interested reader can simply refer to [13]. However, we will give a high-level overview of the proof in order to understand the techniques that are used.

The proof begins with a list of definitions about the notion of *Boolean circuits*. These definitions are essentially equivalent to the logic networks described in the previous section. One notable difference is that the circuits described in [13] consist of alternating levels of AND/OR gates. However, any AND/OR/INV network can be converted into such an alternating circuit of the same depth with only a linear size increase.

An important technique used in the proof is the concept of a *restriction*. Intuitively, one may think of a restriction as a function that fixes certain inputs of a logic network while leaving others untouched. More formally, a restriction is a function

$$\rho : \overline{X_n} \to \{0, 1, *\}$$

where $\overline{X_n}$ corresponds to the network's input literals and the $*$ symbol is used to denote variables that remain unassigned. We write the restriction of a circuit A under a restriction ρ as A^ρ. Every Boolean circuit naturally corresponds to a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$. Given such a function we write the restriction of f under ρ as f^ρ.

It is clear that applying a restriction to a circuit changes the function that it computes. However, the proof uses an interesting property of parity functions. Let f be a function that computes a parity function. Let ρ be an arbitrary restriction. Note that f^ρ is still a parity function. We are now ready to state the main theorem and its proof.

Theorem. *Parity cannot be computed by constant-depth, polynomial-size circuits.*

Proof: by contradiction. Let d be the smallest depth for which parity can be computed with constant-depth circuits. The proof uses three steps to derive a contradiction. With the above assumption we are able to obtain polynomial size parity circuits of depth $d-1$, which contradicts the assumption that d is the smallest depth for which this is possible. In the following, we use n to denote the number of inputs to a circuit.

1) In this step we may assume that we have parity circuits of depth d and polynomial size. It is shown that there is a non-zero chance of picking a random restriction ρ that allows us to obtain a parity circuit for which all gates on the first level have constant size. More specifically, the probability of *failing* to obtaining such a circuit is $o(1)$. Thus, for large enough n the probability of success is greater than zero. Hence, the existence of the required

restriction is demonstrated. The restriction ρ is chosen such that it that randomly assigns 0, 1, or $*$ to inputs based on a given probability distribution.

2) The second step is similar to the first. Here it is shown that given a circuit with constant-size gates on the first level, we can (again through a suitable restriction ρ) obtain a polynomial size circuit of depth d for which the gates on the second level also have constant size.
3) The third step shows how, given the circuits obtained in the first two steps, using the distributive and DeMorgan's laws we can merge the first two levels. By doing so we obtain a parity circuit of depth $d-1$. Note that this circuit is a constant factor larger after merging, due to the restrictions chosen in the first two steps. Hence, we have obtained a polynomial-size depth $d-1$ parity circuit, which contradicts our initial assumption.

■

The authors further strengthen their result by showing the following corollaries:

Corollary. *Parity circuits of depth d must be of size $\Omega(n^{log^{(k)}n})$, where $k = 3(d-2)$.*

Corollary. *Polynomial-size parity circuits must have depth $\Omega(log^* n)$.*

where $log^{(k)}$ indicates the k-th log iterate and $log^* n$ is recursively defined as the inverse: $g(0) = 1$ and $g(n) = 2^{g(n-1)}$.

Thus, the authors establish a superpolynomial lower bound not only on the size of constant-depth parity circuits, but also on the depth of polynomial-size *variable-depth* parity circuits.

B. Extensions To Other Functions

The result that certain Boolean functions can not be efficiently computed with AND/OR/INV primitives naturally leads to the question: are there other such functions? In order to answer this question, the authors propose a reduction technique that is analogous to the one used to prove NP-hardness of decision problems.

Recall that the notion of NP-hardness of a decision problem is that every problem in NP can be reduced to it in polynomial time. The original proof of the NP-hardness of SAT is rather tedious. However, using that result, in order to prove the NP-hardness of other problems we only need to show that SAT is reducible to them in polynomial time. Analogously, we have seen that parity cannot be computed by constant-depth polynomial size circuits. In order to prove the same property for other Boolean functions, the authors introduce the notion of a *constant-depth, polynomial-size* reduction.

Definition. *A function f is constant-depth polynomial-size reducible to a function g ($f \leq_{cp} g$) if f can be realized with constant-depth, polynomial-size circuits using AND/OR/INV gates, and gates computing the function g.*

Theorem. *Note that $f \leq_{cp} g$ implies that g cannot be realized with constant-depth polynomial-size circuits.*

Proof: by contradiction. Suppose that g could be realized with constant-depth polynomial-size circuits. Then, if $f \leq_{cp} g$, we could use these circuits to construct a constant-depth polynomial-size circuit for f. This contradicts our initial assumption for f. ■

Using this reduction technique, the authors show that the majority, multiplication, and transitive closure functions all cannot be realized with constant-depth polynomial-size circuits.

C. Contributions

This paper shows that there exist Boolean functions that cannot be efficiently represented using tradition logic primitives. More specifically, it shows that there are superpolynomial lower bounds on both the depth and size for such circuits. In fact, these lower bounds were later improved by Yao and Hastad [14], [15]. It turns out that there is actually an *exponential* lower bound on the size of constant-depth parity circuits. However, the results of this paper remain important because it formalizes the claim that certain primitives are less powerful than others. Hence it is another argument for the exploration of alternative primitives in Logic Synthesis: more powerful primitives allow us to represent functions more compactly. Notably, this is the case for the majority primitive that is the basis for MIGs.

IV. Depth-Optimal Technology Mapping for FPGAs

Technology mapping can be viewed as the problem of covering a logic network with a collection of primitives. It is also referred to as *cell-library binding* [8]. The collection of primitives used to cover the network is known as a *standard cell library*. The library represents the primitives that are available in a particular technology. For example, some technologies may offer a device that acts as a XOR operator, while others offer one that acts as a M_3 operator [3], [6]. Typically, technology mapping is done after technology independent optimizations have been applied, although there are systems that apply both technology dependent and independent optimizations simultaneously [16].

The third paper in our examination represents a breakthrough in technology mapping for FPGAs. It introduces a polynomial time algorithm for the *depth-optimal* technology mapping of k-bounded logic networks with k-bounded Lookup Tables (LUTs). A k-bounded logic network is one where every node in the graph has an indegree bounded by k. A k-LUT is a lookup table with k-inputs; a logic block that can represent any Boolean function on k variables. In other words, the *FlowMap* algorithm presented in this paper shows how k-LUT primitives can be used to cover a logic network in such a way that the depth of the resulting k-LUT network is optimally minimized. And it does so in polynomial time. The only caveat is that the indegree of nodes in the network *is* restricted by some constant k. This is typically not a problem, since any (finitely) bounded logic network can be transformed into a k-bounded one.

Before the invention of the FlowMap algorithm, depth-optimal technology mapping for FPGAs had been attempted with the use of heuristics, but it had never been solved optimally for arbitrary networks. Moreover, some algorithms

had other primary objectives. For example, the MIS-pga and Chortle algorithms emphasized minimizing the number of LUTs [17], [18]. The FlowMap algorithm has depth optimization as its main goal and area recovery is a secondary concern.

A. The FlowMap Algorithm

FlowMap works in two phases called the *labeling* phase and the *mapping* phase.

1) The labeling phase computes for every node the depth of the k-LUT that implements it in an optimal mapping solution. This depth is expressed by a label that is attached to each node. A side-effect of the label computation is that every node is related to a minimum-height k-feasible cut rooted at that node. One of the main contributions of the paper is in fact an $O(km)$ algorithm for finding a minimum height k-feasible cut (where m denotes the number of edges).
2) In the mapping phase, k-LUTs are generated from the optimal labeling found in the first phase. This is done with a straightforward recursive descent from the outputs to the inputs.

In the remainder of this section we will examine the first phase of the algorithm, as this is where the optimal solution is computed and thus where the primary contribution of the paper is. In the interest of space we do not discuss the proofs of various aspects of the algorithm in detail. The interested reader is referred to [19].

FlowMap is based on a connection between k-feasible cuts and k-LUTs. In order to understand this connection we will introduce some concepts and notation. In a graph $G = (V, E)$ with a source s and a sink t, a cut $(X, \overline{X})$ is a partition of V such that $s \in X$ and $t \in \overline{X}$. The node cut size, denoted $n(X, \overline{X})$, is the number of nodes in X that are adjacent to some node in $\overline{X}$. More formally:

$$n(X, \overline{X}) = |\{x : (x, y) \in E, x \in X \wedge y \in \overline{X}\}|$$

A cut is called k-feasible if $n(X, \overline{X}) \leq k$. The labeling phase computes minimum-height cuts for every node by computing their labels. We use $l(t)$ to denote the label of a node t. The labels of the primary inputs are 0. The height $h(X, \overline{X})$ of a cut is defined to be the maximum label in X:

$$h(X, \overline{X}) = max\{l(x) : x \in X\}$$

Let N_t be the subgraph of G that consists of all predecessors of t. Let LUT(t) denote the LUT that computes t in an optimal-depth solution. Then, LUT(t) induces a cut $(X, \overline{X})$ where $\overline{X}$ is the set of nodes in LUT(t) and X is the rest of the nodes in N_t. If p is the maximum label of a node in X, then the level of LUT(t) is $p + 1$ in the optimal mapping solution. Note that in order to minimize the level of LUT(t), we need to find the minimum-height k-feasible cut $(X, \overline{X})$ in N_t. Thus, the minimum label of t is given by the equation

$$l(t) = min\{h(X, \overline{X})\} + 1$$

where we quantify only over k-feasible cuts.

In the labeling phase of FlowMap, the above equation is used to compute the minimum label for each node. This is done by finding minimum-height k-feasible cuts efficiently. The method used by FlowMap transforms subgraphs N_t in such a way that minimum-height cuts can be efficiently found. By performing collapsing and node-splitting transformations, the problem of finding a k-feasible cut is transformed into the problem of checking whether the maximum flow of a subgraph is no greater than k. This problem can be solved in polynomial time by the augmented path algorithm. Thus, the labeling phase works by traversing the graph in topological order from inputs to outputs and applying transformations and the augmented path algorithm to each node it encounters.

The secondary objective of FlowMap is to minimize the number of LUTs used after an optimal-depth solution has been found. This is implicitly done by maximizing the volume of cuts found in the labeling phase. After the first two phases of the algorithm, a *post-processing* step attempts to further recover area by merging k-LUTs together. This is achieved by using depth-preserving techniques based on *predecessor packing* and *gate decomposition* [20].

B. Contributions

FlowMap was a breakthrough because it was the first polynomial-time algorithm to perform depth-optimal technology mapping. Compared to the state of the art of the day, FlowMap reduced the depth of LUT networks by up to 7% and the size by up to 50%. It pioneered a technique for efficiently finding minimum-height cuts that has been the basis for LUT technology mapping ever since.

Over the years, several improvements have been made to the FlowMap algorithm. These improvements have focused on improving area recovery and the runtime of LUT mapping algorithms [21], [22]. Other approaches have focused on combining technology-independent optimization techniques with technology mapping in order to improve both depth and area results [16], [23].

V. Research Proposal

In this write-up we have looked at three papers related to logic primitives, Logic Synthesis, and optimization. We have looked at practical aspects of logic primitives in the first paper, where MIGs based on the novel majority primitive are shown to outperform traditional methods. MIGs are not the only practical example of novel primitives being applied successfully. *Biconditional BDDs* are a recently developed canonical extension of BDDs. They have been shown to perform better than BDDs, especially for XOR-rich circuits [24]. They have also been shown to have advantages in emerging technologies such as NEM Relays [25].

On the theoretical side, we have seen in the second paper that certain functions (such as parity and majority) cannot be expressed efficiently with traditional primitives. This gives a mathematical basis for our intuition of the advantages of novel primitives.

Hence, based on both practical and theoretical considerations, the research proposal is the following:

It is the goal of this research project to explore novel logic primitives and to use them in the development of efficient representation and optimization techniques for Logic Synthesis.

We may view this research project on the theoretical side as the exploration of novel primitives and the advantages they offer. New theoretical results may be provide logic primitives that are more expressive or better suited to optimization. They may also offer new insight into existing primitives, including majority.

More immediately, there is room to extend the use of novel primitives in a practical manner. For example, there currently exists no native technology mapping package for MIGs. This means that technology mapping cannot take full advantage of the compact MIG representation. Preliminary work suggests that there is room to improve upon the state of the art here. By applying the FlowMap algorithm directly to MIGs, the depth of LUT networks may be decreased by up to 9% as compared to AIGs [26]. Applying more modern technology mapping approaches to MIGs should allow us to gain even more ground over the state of the art.

REFERENCES

[1] C. Märtin, "Multicore Processors: Challenges, Opportunities, Emerging Trends Christian," in *Embedded World Conference*, 2014, pp. 25–27.

[2] O. Y. Loh and H. D. Espinosa, "Nanoelectromechanical contact switches," *Nature Nanotechnology*, vol. 7, no. 5, pp. 283–295, 2012.

[3] M. De Marchi, D. Sacchetto, J. Zhang, S. Frache, P.-E. Gaillardon, Y. Leblebici, and G. De Micheli, "TopDown Fabrication of Gate-All-Around Vertically Stacked Silicon Nanowire FETs With Controllable Polarity," *IEEE Transactions on Nanotechnology*, vol. 13, no. 6, pp. 1029–1038, 2014.

[4] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Prashanth, G. Jan, G. Michael, H. Scott, H. Stephen, A. Hormati, J.-y. K. Sitaram, L. James, and L. Eric, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services - Microsoft Research," in *41st Annual International Symposium on Computer Architecture (ISCA)*, 2014.

[5] A. Prakash, D. Jana, and S. Maikap, "TaOx-based resistive switching memories: prospective and challenges." *Nanoscale research letters*, vol. 8, no. 1, p. 418, Jan. 2013.

[6] P.-E. Gaillardon, L. Amarù, A. Siemon, E. Linn, A. Chattopadhyay, and G. D. Micheli, "Computing Secrets on a Resistive Memory Array," *Design Automation Conference*, 2015.

[7] L. Amarú, P.-E. Gaillardon, and G. D. Micheli, "Majority-Inverter Graph : A Novel Data-Structure and Algorithms for Efficient Logic Optimization," *Design Automation Conference*, 2014.

[8] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[9] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.

[10] G. Birkhoff and S. A. Kiss, "A ternary operation in distributive lattices," *Bulletin of the American Mathematical Society*, vol. 53, no. 8, pp. 749–753, 1947.

[11] T. Sasao, *Switching Theory For Logic Synthesis*. Springer, 1999. [Online]. Available: http://doi.wiley.com/10.1002/0470841915

[12] S. Jukna, *Boolean Function Complexity*. Springer, 2008.

[13] M. Furst, J. B. Saxe, and M. Sipser, "Parity, circuits, and the Polynomial-Time Hierarchy," *Mathematical Systems Theory*, vol. 17, no. 1, pp. 13–27, 1984.

[14] A. C.-C. Yao, "Separating the Polynomial-time Hierarchy by Oracles," in *Proc. 26th Annual Symposium on Foundations of Computer Science*. IEEE Press, 1985, pp. 1–10.

[15] J. Hastad, "Almost Optimal Lower Bounds for Small Depth Circuits," *STOC '86 Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pp. 6–20, 1986.

[16] G. Chen and J. Cong, "Simultaneous Logic Decomposition with Technology Mapping in FPGA Designs," in *ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, 2001, pp. 48–55. [Online]. Available: http://portal.acm.org/citation.cfm?doid=360276.360298

[17] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," in *27 ACM/IEEE Design Automation Conference*, 1990, pp. 620–625.

[18] R. J. Francis, J. Rose, and K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays," *27th ACM/IEEE Design Automation Conference*, pp. 3–9, 1990.

[19] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[20] K.-C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization," in *IEEE Design & Test of Computers*, 1992, pp. 7–20.

[21] J. Cong, C. Wu, and Y. Ding, "Cut Ranking and Pruning: Anabling A General And Efficient FPGA Mapping Solution," in *FPGA '99*, 1999, pp. 29–35. [Online]. Available: http://dl.acm.org/citation.cfm?id=296425

[22] D. Chen and J. Cong, "DAOmap: a depth-optimal area optimization mapping algorithm for FPGA designs," in *International Conference on Computer Aided Design*. IEEE, 2004, pp. 752–759.

[23] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Improvements to technology mapping for LUT-based FPGAs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, 2007, pp. 240–253.

[24] L. Amarú, P.-E. Gaillardon, and G. D. Micheli, "Biconditional Binary Decision Diagrams : A Novel Canonical Logic Representation Form," *IEEE Journal on Emerging and Selected Topics in Circuits And Systems (JETCAS)*, vol. 4, no. 4, pp. 487–500, 2014.

[25] W. Haaswijk, L. Amaru, P.-E. Gaillardon, and G. De Micheli, "NEM Relay Design with Biconditional Binary Decision Diagrams," in *NANOARCH*, 2015.

[26] W. Haaswijk, L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Private Communications."