Majority-Inverter Graph for FPGA Synthesis

Luca Amarú[1], Ana Petkovska[2], Pierre-Emmanuel Gaillardon[1],
David Novo Bruna[2], Paolo Ienne[2], Giovanni De Micheli[1]
Integrated Systems Laboratory (LSI), EPFL, Switzerland[1]
Processor Architecture Laboratory (LAP), EPFL, Switzerland[2]

***Abstract*— In this paper, we present an FPGA synthesis flow based on *Majority-Inverter Graph* (MIG). An MIG is a directed acyclic graph consisting of three-input majority nodes and regular/complemented edges. MIG manipulation is supported by a consistent algebraic framework leading to strong synthesis properties. We propose MIG optimization techniques targeting high-speed FPGA implementations. For this purpose, we reduce the depth of logic circuits via MIG algebraic transformations enabling denser LUT mapping on FPGAs. Experimental results show that our MIG-based design flow reduces by 21%, on average, the delay of the arithmetic circuits synthesized on a state-of-art 28nm commercial FPGA device, as compared to a commercial design flow.**

I. Introduction

Majority-Inverter Graphs (MIGs) are a recently introduced data structure for logic synthesis [1]. Thanks to the expressive power of the majority operator, MIGs are provably superior to any other AND/OR/INV graph in terms of representation compactness. From a logic synthesis perspective, the competitive advantage of MIGs stands in the efficiency and ease of their automated optimization. Indeed, it is possible to explore the entire MIG representation space by using only five primitive transformation rules [1]. MIG-based synthesis has already shown promising results for semi-custom design [1]. However, no application to reconfigurable logic design has been investigated yet.

Field-Programmable Gate Arrays (FPGAs) offer flexibility and cost-effectiveness not achievable by semi-custom circuits. However, their performance, power consumption, and area utilization are worse compared to their semi-custom counterparts [2]. This is especially true for arithmetic intensive applications and circuits [3]. For this reason, FPGA synthesis tools are subject to strong optimization requirements to keep the logic implementation overhead as small as possible. In this scenario, extending the capabilities of logic synthesis techniques for FPGAs is of paramount importance.

In this paper, we propose a new synthesis methodology for FPGA based on MIGs. Even though multiple design metrics can be minimized using MIGs, we focus here on the circuit delay to unlock very high-speed FPGA implementations. We develop MIG optimization techniques that aggressively reduce the logic depth and enable denser LUT mapping on FPGAs. Experimental results show that our MIG-based design flow reduces by 21%, on average, the delay of arithmetic circuits synthesized on a state-of-art 28nm commercial FPGA, as compared to a commercial design flow.

The remainder of this paper is organized as follows. Section II provides a background on FPGA design and on MIGs. Section III presents our MIG-based synthesis methodology targeting FPGAs. Section IV shows experimental results over arithmetic benchmarks on a state-of-art commercial FPGA. Section V discusses the outcomes and limitations of this work in light of the experimental results. Section VI concludes the paper.

II. Background and Motivation

This section presents relevant background on FPGA design and on Majority-Inverter Graphs.

A. FPGA Design

Field Programmable Gate Arrays (FPGAs) are reconfigurable circuits consisting of a regular array of logic elements. Most logic elements exploit *Look-Up Tables* LUTs to implement generic logic functions, a K-input one-output LUT being able to implement any Boolean function of up to K variables. Specific circuits are also intertwined in order to implement specific hardware primitives such as adders, multipliers, memory blocks, etc.

The FPGA design process consists of high-level synthesis, logic synthesis and physical implementation. When designing a logic circuit, the high-level synthesis converts a programming language description (or alike) of a logic system into dedicated hardware and general logic. The general logic portions of the target circuit are processed by logic synthesis that consists of logic optimization and technology mapping. The logic optimization is technology independent and its objective is to reduce the complexity of an abstract logic circuit by minimizing metrics such as the size of the logic network, its depth or its net count. Then, technology mapping maps the logic circuit to the generic logic primitives available in the target FPGA, i.e., the LUTs. Finally, the whole synthesized circuit goes though physical implementation that consists of packing, and placement and routing. As most FPGAs nowadays use clustered-based logic blocks, efficient packing techniques group logic blocks into clusters. After packing, placement and routing assigns the clusters to physical FPGA resources to implement the target circuit.

In this paper, we focus on the logic synthesis step of the FPGA design flow.

B. Majority-Inverter Graph

A *Majority-Inverter-Graph* (MIG) is a data structure for Boolean function representation and optimization. An MIG is defined as a logic network that consists of 3-input majority nodes and regular/complemented edges [1].

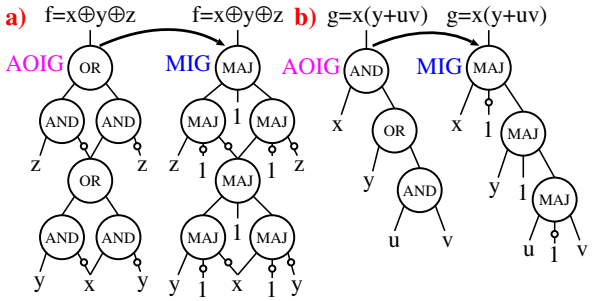

Fig. 1: Examples of MIG representations (right) for (a) $f = x \oplus y \oplus z$ and (b) $g = x(y + uv)$ derived by transposing their optimal AOIG representations (left). Complement attributes are represented by bubbles on the edges.

MIGs can efficiently represent Boolean functions thanks to the expressive power of the majority operator. Indeed, a majority operator can be configured to behave as a traditional conjunction (AND) or disjunction (OR) operator. In the case of 3-input majority operator, fixing one input to 0 realizes an AND while fixing one input to 1 realizes an OR. As a consequence of the AND/OR inclusion by MAJ, traditional AND/OR/INV graphs (AOIGs) are a special case of MIGs and MIGs can be easily derived from AOIGs. Two examples of MIG representations derived from their optimal AOIGs are depicted by Fig. 1. AND/OR operators are replaced node-wise by MAJ-3 operators with a constant input. Intuitively, MIGs are at least as compact as AOIGs. However, even smaller MIG representation arise when fully exploiting the majority functionality, i.e., with non-constant inputs [1]. Later, we will show two smaller and lower depth MIGs for the two examples in Fig. 1.

We are interested in compact MIG representations because they translate in smaller and faster physical implementations. In order to manipulate MIGs and reach advantageous MIG representations, a dedicated Boolean algebra was introduced in the original paper [1]. The axiomatic system for the MIG Boolean algebra, referred to as Ω, is defined by the five following primitive transformation rules.

$$\Omega \begin{cases} \textbf{Commutativity} - \boldsymbol{\Omega.C} \\ M(x,y,z) = M(y,x,z) = M(z,y,x) \\ \textbf{Majority} - \boldsymbol{\Omega.M} \\ \begin{cases} \text{if}(x = y)\text{: } M(x,y,z) = x = y \\ \text{if}(x = y')\text{: } M(x,y,z) = z \end{cases} \\ \textbf{Associativity} - \boldsymbol{\Omega.A} \\ M(x,u,M(y,u,z)) = M(z,u,M(y,u,x)) \\ \textbf{Distributivity} - \boldsymbol{\Omega.D} \\ M(x,y,M(u,v,z)) = M(M(x,y,u),M(x,y,v),z) \\ \textbf{Inverter Propagation} - \boldsymbol{\Omega.I} \\ M'(x,y,z) = M(x',y',z') \end{cases} \quad (1)$$

Some of these axioms are inspired from median algebra [4], [5] and others from the properties of the median operator in a distributive lattice [6]. A strong property of MIGs and their algebraic framework is about reachability. It has been proved that, by using a sequence of transformations drawn from Ω, it is possible to traverse the entire MIG representation space [1]. In other words, given any two equivalent MIG representations, it is possible to transform one into the other by just using axioms in Ω. This results is of paramount interest to logic synthesis because it guarantees that the best MIG, for a given target metric, can always be reached. Unfortunately, deriving a sequence of Ω transformations is an intractable problem. As for traditional logic optimization, heuristic techniques provide here fast solutions with reasonable quality [7]. An efficient depth optimization heuristic for MIGs consists of local Ω rules iterated over the critical path [1]. Here, the rational driving the local transformations is to push up variables with largest depth.

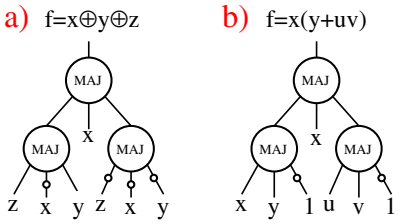

Fig. 2: Optimized MIGs for (a) $f = x \oplus y \oplus z$ and (b) $g = x(y + uv)$.

As previously anticipated, by using the MIG algebraic framework it is possible to obtain better MIGs for the examples in Fig. 1. Fig. 2 shows the new MIG structures, which are optimized in both depth (number of levels) and size (number of nodes). These MIGs can be reached using a sequence of Ω axioms starting from their unoptimized structures. Details on the optimization procedure for the example in Fig. 2(b) will be given in Section III. We refer the reader to paper [1] for an in-depth discussion on MIG optimization recipes.

III. MIG-based FPGA Synthesis

In the FPGA synthesis scenario, MIGs and their algebraic framework enable efficient logic optimization. As the performance overhead with respect to ASICs is one of the major hurdles of FPGA implementations, we develop high-speed MIG-based synthesis algorithms to keep the FPGA/ASIC performance gap just at its essential. Our contribution is mainly at the logic optimization level but we describe here also the integration of our technique in a complete FPGA design flow.

A. Note on Logic Depth vs. Delay Reduction

From a logic representation standpoint, reducing the delay of a physical circuit naturally corresponds to reducing the maximum number of logic levels (depth) in its abstract representation. This model has been proven reasonably accurate in early technology nodes, but, with the advent of nanometer-scale technology nodes, the impact of interconnects is changing such trend. This is especially true when a reduction of the depth implies an increase in area and thus more interconnect requirements. In the context of FPGA architectures, the interconnect comes at a higher cost than in semi-custom integrated circuits and therefore the trade-off between depth reduction vs. area increase has to be carefully considered. Nevertheless, when the depth reduction is much larger than its size overhead the final physical delay results are still coherent with the logic-level predictions. We refer to this scenario as aggressive-depth optimization. In this paper, we target aggressive depth-optimization of MIGs for high-speed FPGA implementations in nanometer-scale technologies.

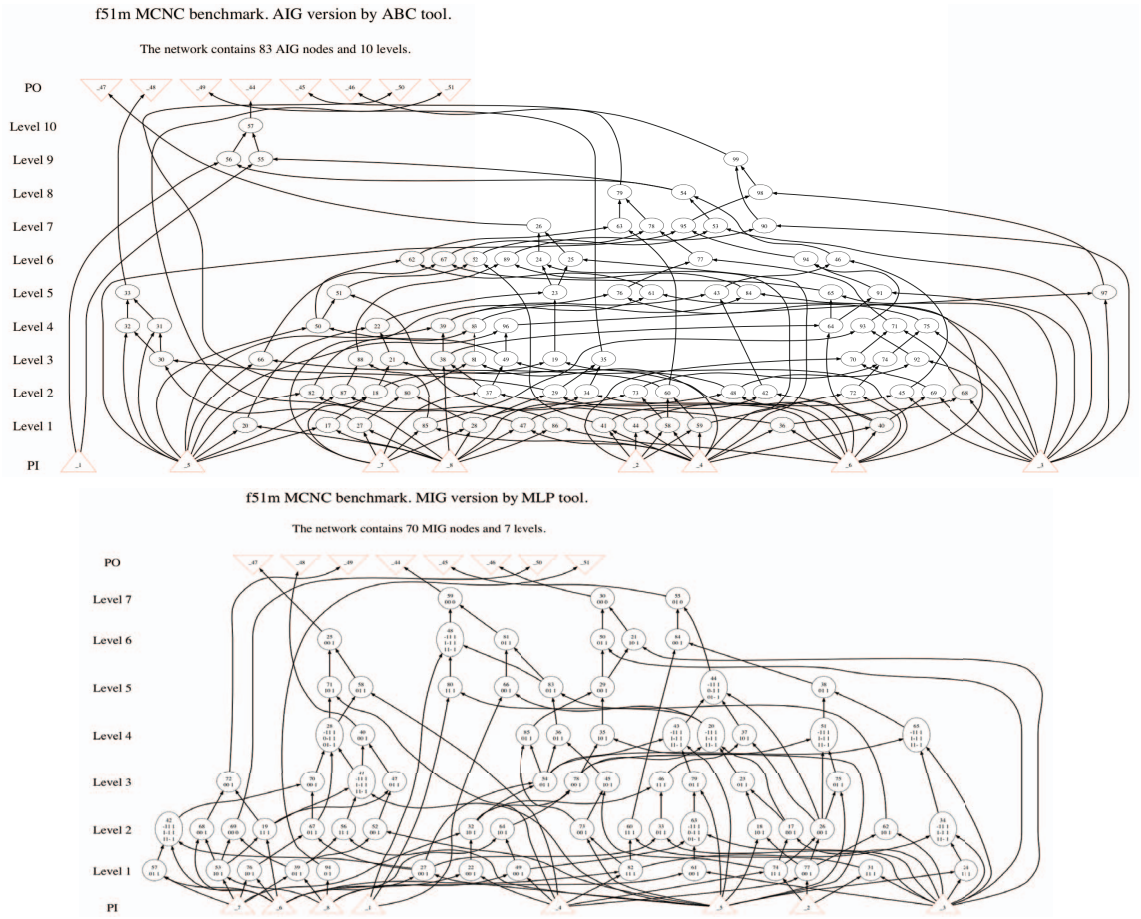

Fig. 3: f51m MCNC benchmark optimized by ABC (And-Inverter Graph) and by MIG depth optimization.

B. Powerful Depth-Reduction of MIGs

Our optimization goal is to reduce the depth, i.e., the maximum number of levels, in an MIG. The depth of an MIG is defined as the largest depth among all the majority nodes present in such a graph.

1) Generalities: The depth of a majority node $b = M(a_1, a_2, a_3)$ is defined as $D(b) = max_{i=1,2,3} D(a_i) + 1$, where a_1, a_2, and a_3 are the inputs of b. As boundary conditions, the depths of primary inputs and of 0/1 constants are usually set to 0 even though different arrival times can be specified.

An intuitive approach to reduce the depth of an MIG is to apply local transformations on the nodes with largest depth. Strategically, one can either focus only on the nodes on the critical path or apply depth optimization on all the nodes. On the one hand, when focusing on the critical path, we get depth reduction with the minimum size overhead. On the other hand, when considering all the nodes, we get even better depth reduction (because more optimization opportunities arise) with larger size overhead. In the latter case, size recovery techniques help reducing the overhead to (or close to) the minimum value. When targeting aggressive depth-optimization, a global depth reduction interlaced with tight size recovery phases usually produces the best results. In all cases, the key feature determining the MIG depth optimization quality is the efficiency of local depth reduction of a majority node.

In the following, we analyze the possible scenarios for depth optimization of a majority node $b = M(a_1, a_2, a_3)$. We restrict ourselves to the cases where the inputs a_1, a_2 and a_3 are unbalanced in depth. This is because we know that our local MIG algebraic transformations can always reduce the depth in such cases. More complex MIG Boolean techniques can reduce the depth also for the other cases, but their study is out of the scope of this paper. For the unbalanced depth case, we assume, without loss of generality, that $D(a_1) < D(a_2) < D(a_3)$. In this case, the depth of node b can be written as $D(b) = D(a_3) + 1$. So, if we can reduce the depth of a_3 at least of one unit, we would reduce the depth of b at least by one unit. To see how this depth reduction can be made possible, let us expand the expression of a_3 inside b as $b = M(a_1, a_2, a_3) = M(a_1, a_2, M(a_{3;1}, a_{3;2}, a_{3;3}))$. If a complemented edge appears in a_3, it is propagated down to $a_{3;1}, a_{3;2}, a_{3v3}$ via axiom $\Omega.I$. Again, we assume that a_3 inputs are unbalanced in depth, following the relation $D(a_{3;1}) < D(a_{3;2}) < D(a_{3;3})$. Note that, in some cases, the inequalities do not need to be strict, we will give an example about this case later on. We distinguish three cases to reduce the depth of $b = M(a_1, a_2, M(a_{3;1}, a_{3;2}, a_{3;3}))$.

a) *Two inputs at the same level are identical (up to complementation):* This happens, for example, when $a_1 = a_2$ or $a_1 = a_2'$. In this case, the majority axiom $\Omega.M$ applies and eliminates the top node. For example, if $a_1 = a_2'$, then b can be simplified as $b = M(a_{3;1}, a_{3;2}, a_{3;3})$, which corresponds to

depth reduction of one level. Similar reasoning holds when a pair of inputs are identical (up to complementation) in $M(a_{3;1}, a_{3;2}, a_{3;3})$.

b) *Two inputs between the levels are identical (up to complementation)* We assume, without loss of generality, that $a_1 = a_{3;1}$ or $a_1 = a'_{3;1}$. All the other cases can be obtained by commutativity $\Omega.C$. We first consider $a_1 = a_{3;1}$. By using the associativity axiom $\Omega.A$, it is possible to rewrite $b = M(a_1, a_2, M(a_{3;1}, a_{3;2}, a_{3;3}))$ as $b = M(a_1, a_{3;3}, M(a_{3;1}, a_{3;2}, a_2))$. By pushing the critical element $a_{3;3}$ one level up we reduce the depth of b by one unit. We now consider $a_1 = a'_{3;1}$. Using a combination of axioms in Ω (see [1] for more details), it is possible to rewrite $b = M(a_1, a_2, M(a_{3;1}, a_{3;2}, a_{3;3}))$ as $b = M(a_2, a_{3;3}, M(a_1, a_2, a_{3;2}))$. Also here, the critical element $a_{3;3}$ is pushed one level up and thus we reduce the depth of b by one unit.

c) *All inputs are independent* This is the most general case for the depth minimization of an MIG. In $b = M(a_1, a_2, M(a_{3;1}, a_{3;2}, a_{3;3}))$, with our unbalanced depth assumptions, the critical depth element is $a_{3;3}$. The distributivity axiom $\Omega.D$ is key in this case. Indeed, by applying $\Omega.D$ (from left to right), it is possible to rewrite b as $b = M(a_{3;3}, M(a_1, a_2, a_{3;1}), M(a_1, a_2, a_{3;2}))$ where $a_{3;3}$ is pushed one level up and thus we reduce the depth of b by one unit. While, in the other cases, the depth reduction required no size overhead, here one extra node is locally required. Note that such extra node might already exist in a global MIG.

C. Optimization Examples

In this section, we give two depth optimization examples for an MIG. The first one is a small hand-made example, to showcase the application of MIG depth reduction rules. On the other hand, the second one shows graphically the MIG depth optimization effect on a medium-sized benchmark.

1) Small-sized Example: We present a hand-made optimization over a relatively small Boolean function, by focusing on the MIG-depth optimization procedure used to obtain the MIG in Fig. 2(b) from the MIG in Fig. 1(b). The original function is $f = x(y + uv)$ whose initial MIG representations has been translated from its optimal AOIG (Fig. 1(b)). In this example, all the nodes belong to the critical path assuming that inputs/constants have 0 arrival time. The MIG can be also expressed in formula as $f = M(x, 0, M(y, 1, M(u, 0, v)))$. One can immediately notice that between the top and middle levels two inputs are identical, up to complementation. These are constant 0 and constant 1, respectively. Using the previously introduced notations, one can assign the variables in this formula as $f = M(a_1, a_2, M(a_{3;1}, a_{3;2}, a_{3;3}))$ where $a_1 = a'_{3;1} = 0$, $a_2 = x$, $a_{3;2} = y$, and $a_{3;3} = M(u, 0, v)$. Applying the depth reduction transformation described above (see III-B1), we obtain $f = M(a_2, a_{3;3}, M(a_1, a_2, a_{3;2})) = M(x, M(u, 0, v), M(0, x, y))$. Such result corresponds to the optimized MIG in Fig. 2(b) that has one level of depth less and the same number of nodes as compared to the original MIG in Fig. 1(b).

2) Medium-sized Example: To show the practical effect of the MIG depth reduction on a larger circuit, we consider the MCNC benchmark *"f51m"*, which has 8 inputs and 8 outputs. After optimization with the ABC synthesis tool [8], [9], where the commands *collapse* and *resyn2rs* are iterated until the structure cannot be further improved, this benchmark counts 83 AND nodes and 10 levels. An initial MIG can be derived from such AND-INV Graph preserving an identical size and depth. Then, we employ the MIG depth reduction techniques previously described. We combine them in an aggressive depth optimization strategy interlaced with size recovery phases. As a result, the depth-optimized MIG counts just 7 levels of depth and 70 MAJ nodes. AIG and MIG are depicted by Fig. 3.

To give an early idea about the impact of these results in FPGA synthesis, we perform LUT6-mapping on *"too_large"* benchmark via ABC. By mapping the MIG-optimized version, we get 3 levels of logic, while, by mapping the original version, we get 4 levels of logic. Section IV presents experiments on larger benchmarks, where even better advantages exist.

IV. System-Level Performance Evaluation

Up to this point, we explained how MIGs and their algebraic framework can reduce the delay of FPGA designs from a theoretical standpoint. In this section, we show how MIGs can be used to complement commercial FPGA flows and improve the mapping of logic circuits on state-of-the-art commercial FPGAs. First, we introduce the considered tool flow and the evaluation methodology. Then, we give design results over several arithmetic benchmarks for both our novel methodology and for traditional FPGA design flow.

A. The MIG-FPGA Design Flow

In the following, we consider a state-of-art commercial FPGA design flow that comprises high-level synthesis, logic optimization, technology mapping and physical implementation tools in a holistic environment. We refer to this flow as *traditional (commercial) FPGA design flow*.

On the other hand, for the *MIG-FPGA design flow*, we execute an MIG optimization as a pre-synthesis technique by applying MIG depth reduction techniques on combinational portions of the input circuit. The output of the MIG optimization is then fed to the standard flow to complete the FPGA design process. Note that the output of the flow consists of full FPGA programming bitstreams, demonstrating the immediate industrial potential of the approach.

Note that, for both flows, formal equivalence checking is performed between the original design and the final FPGA implementation represented as a post placement and routing netlist, in order to guarantee the legality of the results. Fig. 4 depicts, side by side, the traditional FPGA design flow and the MIG-FPGA design flow.

B. Methodology

We consider a commercial 28nm FPGA device with its associated design suite[1]. The primitive logic element here has the equivalent expressive power of a 6-input LUT and can be used in many different modes, e.g., memory, adders, multiplexers, etc. The tool suite is used in its standard settings, with a delay

[1]Note that the vendor name, device part and tool suite cannot be disclosed due to our license agreement.

TABLE I: MIG-based FPGA Design *vs.* Traditional FPGA Design

Traditional FPGA Design		LUT#	Logic Blocks#	Glob. Int.#	Loc. Int.#	Delay (ns)	Cell (ns)	Int. (ns)
Benchmark	I/O							
add32	64/33	93	50	119	46	32.56	9.87	22.69
csa464	256/66	472	253	460	275	80.95	18.61	62.33
sqrt32	32/16	797	519	1175	371	90.45	30.58	59.87
diffeq1	355/289	5015	3195	7064	2440	97.86	22.43	75.42
compr64to6	64/6	247	161	263	132	49.53	13.56	35.96
hamming	200/7	637	431	722	265	58.21	10.90	47.31
div16	32/32	432	292	587	211	74.05	25.61	48.43
revx	20/25	2309	1391	2713	1147	69.20	21.44	47.75
mul32	64/64	1893	1053	2311	865	23.84	7.64	16.20
Total	1088/538	11895	7345	15414	5752	576.65	160.67	415.98
MIG-based FPGA Design		LUT#	Logic Blocks#	Glob. Int.#	Loc. Int.#	Delay (ns)	Cell (ns)	Int. (ns)
Benchmark	I/O							
add32	64/33	311	184	361	185	13.87	6.42	7.45
csa464	256/66	759	432	937	419	27.28	6.48	20.80
sqrt32	32/16	291	194	396	142	73.67	26.40	47.27
diffeq1	355/289	4947	3140	6891	2401	88.99	22.32	66.67
compr64to6	64/6	328	208	348	177	42.55	15.31	27.20
hamming	200/7	570	387	723	233	45.57	12.57	33.00
div16	32/32	442	300	627	208	67.19	22.19	45.00
revx	20/25	2064	1287	2615	1034	70.92	20.54	50.38
mul32	64/64	1850	1048	2522	868	27.95	7.43	20.52
Total	1088/538	11562	7180	15420	5667	457.99	139.68	318.31
MIG/Trad. ratio	1/1	0.97	0.97	1.01	0.98	**0.79**	0.86	0.76

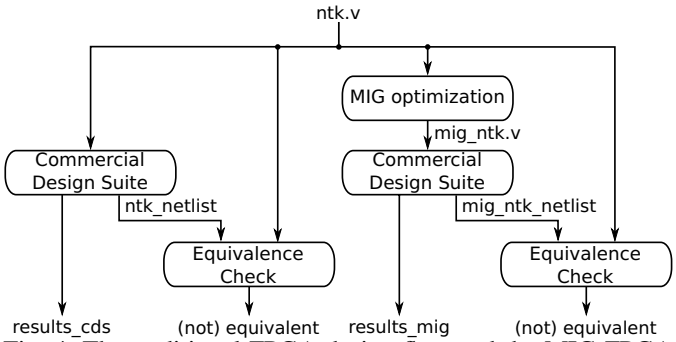

Fig. 4: The traditional FPGA design flow and the MIG-FPGA design flow.

oriented optimization. The benchmarks considered are combinational arithmetic circuits whose implementation on FPGA is challenging. For example, we consider square-root modules, bit-compressors, multi-operand adders, differential equation solvers and others. Sequential benchmarks are currently not handled by our software tool. Our reference flow is the native traditional design flow of the considered commercial FPGA tool suite.

As presented in Section IV-A, we use the MIG depth reduction techniques on top of the commercial tool, as a preoptimization step, which produces an alternative but equivalent Verilog description to the flow. These techniques are implemented within a *Majority Logic Package* (MLP) written in C language. The depth minimization strategy is *global* to enforce as much level reduction as possible, interlaced with size recovery techniques. The output of the MLP tool is a Verilog description of the optimized circuit in terms of atomic majority operators.

All formal verification experiments, at both logic and FPGA implementation levels, produced a positive (correct) result.

C. Experimental Results

Table I shows the FPGA design results, while Fig. 5 depicts the same results graphically. All numbers refer to the final design implementation, i.e., post place & route on the FPGA fabric. In total, the MIG-based flow reduces the delay by 21%. Regarding the utilization of FPGA elements, the total number of LUTs and interconnects is practically the same with only few percentage points of difference, on average. The delay breakdown accounted to logic blocks and interconnects is depicted in Fig. 5-right. One can note that the depth reduction does not reduce significantly the delay coming from the logic blocks. Logic block delay depends on the technology mapping results. In the MIG context, technology mapping does not group many logic levels per LUTs, as compared to the standard implementations. LUTs absorb nodes in *width* rather than in *depth* from the logic network. However, it is worth pointing out that the interconnect delay is reduced in most of the considered benchmarks. MIG optimization increases the number of logic nodes and, therefore, impacts the number of LUTs and interconnections. Nevertheless, this increase of likely to be *local*, i.e., having global and local interconnections with a smaller length and reduced number of hops, that reduces the overall constraint on the routing elements. This validates the advantage of MIG depth reduction in FPGA design for arithmetic benchmarks. The delay reduction is positive for all benchmarks except two: *revx* and *mul32*. For these two benchmarks, MIG optimization reduces the cell delay but increase the interconnect delay. The benchmark with the largest delay reduction is *csa464*. It implements an adder with 4 operand of 64-bit width each. In its original description, this benchmark has 139 levels of logic (in terms of cascaded AND2 gates, INV apart). Thanks to MIG depth reduction, the same benchmark has just 19 levels of logic (in terms of cascaded MAJ3 gates, INV apart). The final delay reduction

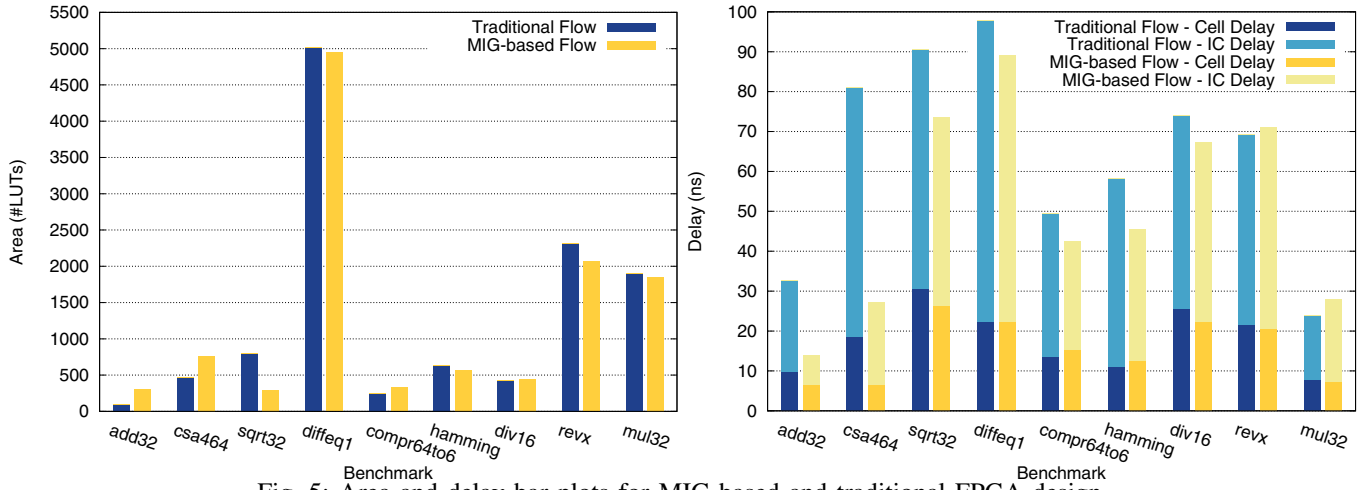

Fig. 5: Area and delay bar plots for MIG-based and traditional FPGA design.

is 3× as compared to the traditional FPGA design flow, but the LUT count overhead is 60%.

V. Discussion

The proposed approach demonstrates very promising performance improvements for FPGA design of arithmetic circuits. As compared to semi-custom circuits, FPGAs have reduced flexibility for implementing arithmetic circuits. Indeed, the high-level synthesis engine is usually employed to map arithmetic functions on the numerous specific hardware supports. Unfortunately, the designer looses the flexibility of the implementation. For instance, adders are typically implemented as ripple-carry adders as they are supported by the commercial FPGA devices. This would preclude the designer to efficiently design a carry-lookahead adder circuit with potentially higher performance. In addition, when the high-level synthesis fails to identify arithmetic primitives, all the design efforts are transferred to the logic synthesis engine that is likely to loose many optimization opportunities. The proposed technique provides an approach to increase the performances of the logic synthesis engine to handle such situations.

The MIG optimization is currently used as a front-end of a commercially available FPGA design suite. The flow is capable to generate programming bitstream for state-of-the-art FPGA devices. While the approach is directly employable in an industrial context (real circuits can be implemented using the technique with significant improvement), we can expect an even higher level of performances if MIGs were integrated natively into the logic optimization and technology mapping engines. Indeed, the current implementation is likely to miss many optimization advantages due to the use of non-unified data representation between the different tools.

Finally, we can expect more refined techniques exploiting MIGs. At the tool level, MIGs give the possibility to directly exploit the FPGA arithmetic macros during the logic optimization instead of only relying on high-level synthesis. In particular, it is possible to envisage the automatic realization of arithmetic circuits that tightly exploit both LUTs and specific carry-path primitives. At the circuit level, MIGs can be supported in a more disruptive way. Following the approaches proposed in [10], [11], LUTs can be replaced by dedicated hardware supports that natively implements the majority operation networks.

VI. Conclusions

With this paper, we present an FPGA synthesis flow based on *Majority-Inverter Graph* (MIG). An MIG is a directed acyclic graph consisting of three-input majority nodes and regular/complemented edges. MIG manipulation is supported by a consistent algebraic framework leading to strong synthesis properties. We proposed MIG optimization techniques targeting high-speed FPGA implementations. We reduce the depth of logic circuits via MIG algebraic transformations enabling denser LUT mapping on FPGAs. The experimental results show that our MIG-based design flow reduces by 21%, on average, the delay of arithmetic circuits synthesized on a state-of-art 28nm commercial FPGA device, as compared to a commercial design flow.

Acknowledgements

This research was supported by ERC-2009-AdG-246810.

References

[1] L. Amarú, P.-E. Gaillardon, G. De Micheli, *Majority-Inverter Graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization*, Proceedings of the 50th Design Automation Conference, 2014, pp. 1-6.

[2] I. Kuon, J. Rose. *Measuring the gap between FPGAs and ASICs*. Proceedings of the 14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, Calif., Feb. 2006, pp. 21-30.

[3] H. Parandeh-Afshar *et al.*, *Improving FPGA performance for carry-save arithmetic*, IEEE Trans. on VLSI Syst., vol. 18, no. 4, pp. 578-90, 2010.

[4] J. R. Isbell, *Median algebra*, Trans. Amer. Math. Soc., 319-362, 1980.

[5] D. Knuth, *The Art of Computer Programming*, Volume 4A, Part 1, New Jersey: Addison-Wesley, 2011.

[6] G. Birkhoff, *A ternary operation in distributive lattices*, Bull. of the Amer. Math. Soc., 53 (1): 749752, 1947.

[7] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.

[8] R. Brayton, A. Mishchenko, *ABC: An Academic Industrial-Strength Verification Tool*, in Proceedings of the International Conference on Computer Aided Verification, ser. Lecture Notes in Computer Science, vol. 6174. Springer, Jul. 2010, pp. 2440.

[9] ABC: A System for Sequential Synthesis and Verifica- tion, Berkeley Logic Synthesis and Verification Group, Berkeley, Calif., Sep. 2014.

[10] P.-E. Gaillardon, L. Amarú, G. De Micheli *A New Basic Logic Structure for Data-Path Computation*, in Proceedings of the 22nd ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2014.

[11] G. Zgheib, *et al.*, *Revisiting And-Inverter Cones*, in Proceedings of the 22nd ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2014, pp. 45-54.