POWER AND DELAY OPTIMIZED GRAPH REPRESENTATION FOR COMBINATIONAL LOGIC CIRCUITS

Padmanabhan Balasubramanian, and Karthik Anantha

Abstract— Structural representation and technology mapping of a Boolean function is an important problem in the design of non- regenerative digital logic circuits (also called combinational logic circuits). Library aware function manipulation offers a solution to this problem. Compact multi-level representation of binary networks, based on simple circuit structures, such as AND-Inverter Graphs (AIG) [1] [5], NAND Graphs, OR-Inverter Graphs (OIG), AND-OR Graphs (AOG), AND-OR-Inverter Graphs (AOIG), AND-XOR- Inverter Graphs, Reduced Boolean Circuits [8] does exist in literature. In this work, we discuss a novel and efficient graph realization for combinational logic circuits, represented using a NAND-NOR-Inverter Graph (NNIG), which is composed of only two-input NAND (NAND2), NOR (NOR2) and inverter (INV) cells. The networks are constructed on the basis of irredundant disjunctive and conjunctive normal forms, after factoring, comprising terms with minimum support. Construction of a NNIG for a non-regenerative function in normal form would be straightforward, whereas for the complementary phase, it would be developed by considering a virtual instance of the function. However, the choice of best NNIG for a given function would be based upon literal count, cell count and DAG node count of the implementation at the technology independent stage. In case of a tie, the final decision would be made after extracting the physical design parameters. We have considered AIG representation for reduced disjunctive normal form and the best of OIG/AOG/AOIG for the minimized conjunctive normal forms. This is necessitated due to the nature of certain functions, such as Achilles' heel functions. NNIGs are found to exhibit 3.97% lesser node count compared to AIGs an OIG/AOG/AOIGs; consume 23.74% and 10.79% lesser library cells than AIGs and OIG/AOG/AOIGs for the various samples considered. We compare the power efficiency and delay improvement achieved by optimal NNIGs over minimal AIGs and OIG/AOG/AOIGs for various case studies. In comparison with functionally equivalent, irredundant and compact AIGs, NNIGs report mean savings in power and delay of 43.71% and 25.85% respectively, after technology mapping with a 0.35 micron TSMC CMOS process. For a comparison with OIG/AOG/AOIGs, NNIGs demonstrate average savings in power and delay by 47.51% and 24.83%. With respect to device count needed for implementation with static CMOS logic style, NNIGs utilize 37.85% and 33.95% lesser transistors than their AIG and OIG/AOG/AOIG counterparts.

Keywords—AND-Inverter Graph, OR-Inverter Graph, Directed Acyclic Graph, Low power design, Delay optimization.

I. INTRODUCTION

In many commercial logic synthesis systems, approaches to technology-specific multilevel logic synthesis, global logic structure synthesis followed by local optimization and technology mapping are found. A Boolean network is often the target of global logic structure synthesis, serving as an intermediate description between input specification in HDL or PLA-like format and the final network in a specific target technology. Multilevel minimization algorithms have been developed to further reduce the cost of an initially synthesized Boolean network. However, the designs they generate after initial synthesis by an algebraic approach are far inferior to manual designs in some cases, because changes during multilevel minimization are usually constrained to a local node, although global information is used to propose or validate such changes. Hence initial structure synthesis is important in finding better designs because improvement by multilevel minimization seems to be bounded by the initial structure. During

multilevel network synthesis, each node of a Boolean network is represented by minimal ON- and/or OFF-covers and in some cases together with factored expressions derived from those minimal sum-of-products (SOP), with cost measured in terms of the number of cubes, literals or supporting variables. The reason for using minimal disjunctive normal forms is not that they always lend themselves to the best factored forms or provide the best sub-functions, but rather that we cannot easily ascertain other forms, which are guaranteed to be generally better than minimal SOP forms.

There are cases where non-minimal expressions may be preferred, but there are too many non-minimal expressions to explore all of them, and it is difficult, if not impossible, to predict which non-minimal form will be desirable at the next step of synthesis. Hence, it is prudent to generally use minimal disjunctive normal forms. This paper does not propose a new algorithm for multilevel logic synthesis, but rather modifies the proposed functional decomposition into gates [1], to incorporate cells (library cells) which understood in terms of gates, still preserves the advantages of the functional decomposition. In this context, it becomes clear that this paper focuses on a technology driven multilevel logic synthesis based on functional decomposition, which utilizes minimum disjunctive and conjunctive normal forms. We understand that since the initial synthesis is performed with close relation tothe actual technology target, the opportunities created by modern microelectronic technology can be more effectively exploited because of a direct relevance of the synthesis procedure to technology implementation, in terms of the actual cells used for final implementation. This would carry significance for semi-custom, full-custom or standard cell-based digital IC design approaches, where constraints are imposed on the number of serial and parallel transistor realizing gates and on the interconnections between the gates. The remainder of this paper is organized as follows. Section 2 provides concise background information relevant to the paper. Section 3 deals with the properties and practical advantages of AIGs. Section 4 deals with the construction and description of some of the inherent advantages of NNIGs. This section also contains an illustrative example, to demonstrate the optimization in design metrics achieved by a NNIG in comparison with its functionally equivalent AIG and OIG. Simulation results for the other problem cases, including the special functions considered, are mentioned in section 5. We finally make the concluding remarks in section 6.

II. PRELIMINARIES

A. Definition 1

A single output Boolean function, F(xn-1 , xn-2,....,x0) is amapping, f: {0,1} n → {0,1,d}, where ‘d’ denotes a ‘don’t care’condition. If the DC condition does not exist, then it is a completely specified function (CSF). Each of the 2 n nodes inthe Boolean space corresponds to a minterm. If a minterm is mapped to output 0 (1 or d), then it is called an OFF-set (ON-set or DC-set) minterm.

B. Definition 2

A binary logic network is a directed acyclic graph (DAG) with nodes representing Boolean functions. The sources of the graph are referred to as the primary inputs of the network; the sinks are the primary outputs. The output of a node may be an input to other nodes called its fan-outs. The inputs of a node are called its fan-ins.

III. AND-INVERTER G RAPHS

AND-Inverter Graph (AIG) is a directed acyclic graph that represents a structural implementation of the combinational logic functionality of any random Boolean network. As the name implies, it is composed of only two-input AND gates and inverters. An AIG consists of two-input nodes, representing logical conjunction, and edges optionally containing markers indicating logical negation. The size (area) of the DAG representing an AIG is the number of its nodes and its depth (delay) is the number of nodes present in the longest path from any of its primary inputs to any of its primary outputs. This representation of a logic function is rarely structurally efficient for large circuits, but is an efficient representation for manipulation of Boolean functions. Although concise multi-level representation of binary networks, based on simple circuit structures, such as OIG (obtained from minimal product-of-sums form or POS), NAND graphs, AND-XOR-INV graphs do exist, conversion from the network of logic gates to AIGs is fast, easy and scalable [1]. It only requires that every gate be expressed in terms of AND gates and inverters. This conversion does not lead to unpredictable increase in memory use and runtime. This makes the AIG an efficient representation in comparison with either the binary decision diagram (BDD) or the sum-of- products (SOP) form. The BDD and DNF may also be viewed as circuits, but they involve formal constraints that deprive them of scalability. For example, reduced DNFs obtained using standard commercial minimizers, such as ESPRESSO correspond to circuits with at most two levels, while BDDs are canonical, that is, they require that input literals be evaluated in the same order on all paths. The abstract data graph (here binary network) is usually represented by a data structure. Circuits composed of simple gates, including AIGs, are an old research topic. The interest in AIGs started in the early 1960's [7] and continued in the 1970's when various local transformations have been developed. These transformations were implemented in several logic synthesis and verification systems, such as [4], which reduce circuits to improve area and delay during synthesis, or to speed up formal equivalence checking. Several important techniques were discovered early at IBM [9], such as combining and reusing multiple-input logic expressions and sub-expressions, now known asstructural hashing or strashing [5]. A hash table is used to remove redundant components during network construction by structure sharing. Recently, there has been a renewed interest in AIGs as a functional representation for a variety of tasks in synthesis and verification. This is because representations popular in the 1990's (such as BDDs) have reached their limits of scalability in many of their applications. Another important development was the emergence of much more efficient Boolean satisfiability (SAT) solvers. When coupled with AIGs as the circuit

representation, they lead to remarkable speedups in solving a wide variety of Boolean problems.

AIGs found successful use in diverse electronic design automation (EDA) applications. A well-tuned combination of AIGs and Boolean satisfiability made an impact on formal verification, including both model checking and equivalence checking. In [8], efficient circuit compression techniques were developed using AIGs. There is a growing understanding that logic and physical synthesis problems can be solved using AIGs. Another recent activity [10] shows that the simple and uniform structure of AIGs allow rewriting, simulation, mapping, placement and verification to share the same data structure.

In addition to combinational logic, AIGs have also been applied to sequential logic and sequential transformations. Specifically, the method of structural hashing was extended to work for AIGs with memory elements (such as D-flip flops with an initial state, which, in general, can be unknown) resulting in a data structure that is specifically tailored for applications related to retiming [3]. A modern logic synthesis system, ABC [11], developed by EECS Dept. of the University of California, Berkeley, completely based on AIGs, features an AIG package, several AIG-based synthesis and equivalence-checking techniques, as well as an experimental implementation of sequential synthesis. Although optimizations can be implemented using networks composed of arbitrary gates, use of AIGs makes them fast and scalable.

IV. NAND-NOR-INVERTER GRAPHS

Similar to AIGs, we find that a combinational logic circuit can be conveniently represented using NAND-NOR-Inverter Graphs (NNIG). A NNIG also corresponds to a DAG representation for a non-regenerative logic function. The gate analogy pertaining to functional decomposition is now replaced by cell analogy, whilst preserving the definition and attribute, corresponding to a function. In this case, the network would only comprise of two-input NAND, NOR and single input inverter cells. NNIGs are also non-canonical structures: although a Boolean function can have many functionally equivalent NNIG representations corresponding to different expressions at the two-level logic, typically two structures would be compact representations; one obtained from a factored minimum SOP (MSOP) of the function and other based on factored minimum POS (MPOS) of the function. The internal nodes of a NNIG may have equivalent functionality, if the standard expressions from which they are created are not factored. This may increase the number of NNIG nodes and makes reasoning on the graph structure inefficient and time consuming. NNIGs also have the inherent capability to incorporate functional reduction similar to AIGs [5], such that no functionally identical nodes exist in a network, which is required for many applications cited in [5]. This is possible by simple algebraic factoring. Such weak factorization operations would ensure that no functional redundancy exists. However such factoring operations would yield best results provided the literals used for decomposition are the biggest divisors and result in reduced sub-function remainders. Functionally reduced NNIGs, obtained by strashing, hence become semi-canonical and would be much more robust than BDDs and can be constructed for a wid range of practical circuits in reduced polynomial time. Simulation of NNIGs can be performed efficiently because

they comprise a regular structure. NNIGs can also be easily and effectively combined with satisfiability-solvers [6] for combinational equivalence checking. NNIG for a logic function can be constructed starting from either an input description in PLA format or directly from the circuit netlist level. For example, given a canonical logic function description in PLA format, the reduced disjunctive normal form can be obtained using industry standard two-level logic minimizers, such as ESPRESSO [13]. Then the reduced disjunctive normal form can be factorized to comprise only the primitive two-input gates, which directly correspond to library cells. Sharing common logic at this stage, i.e. before proceeding with the construction of a NNIG would always result in best cost optimization as mentioned above and ensure that all the functional redundancies are eliminated. This technique would be significant for custom and semi-custom designs. The factored form can then be converted into a NNIG, by recursive application of De-Morgan's laws, based on several depth-first traversals of the graph structure. This technology-independent procedure is clearly library aware and hence could directly be correlated to the technology-mapping phase. To construct the virtual NNIG (VNNIG) for a given function, the starting point would be to consider a virtual instance of the function, which is opposite to the normal function phase. The minimized disjunctive normal form corresponding to this function phase is then used as the starting point for the development of a VNNIG for a given function. Of course, for this case, the VNNIG would have an additional inverter at the terminal output node to compensate for the reversal of the function phase. Hence, it becomes sufficiently clear from the above discussion that if the compact AIG representation of a Boolean function would be created based on the real instance of the function then the OIG for that function would be basedon the virtual instance and vice-versa. This approach is deemed necessary as it facilitates a reduction in the number of operators (also called DAG nodes) required for creating a logic structure graphically representing an OIG.

A. Illustrative Example

Consider the ON-set of a 5-variable function given by,

$$Z(A,B,C,D,E) = \{6,10,12,13,14,15,22,26,28,29,30,31\} \quad (1)$$

The corresponding optimized structures for AIG and OIG structures derived from minimum and irredundant SOP and POS forms of the function are represented by Figures 1 and 2, respectively, in terms of the primitive gates as shown below. For this example, an AIG was created from the virtual instance of the function as it was found to be compact; the OIG from the virtual instance of the function comprising maxterms. The inverter at the output compensates for the complementation of the function polarity. NNIG structure shown in Fig. 3 was created from the MPOS of the virtual function instance and NNIG shown in Fig. 4 was created from the MSOP of the real function instance, after algebraic factorization operations on the corresponding minimum expressions. We will associate a two-valued parameter for each graph structure (n, c), where 'n' and 'c' stand for the number of DAG nodes and cells constituting the graphs respectively. Henceforth, the symbols , , and shall stand for AND, NAND, OR and NOR operators respectively. The flipped edges correspond to NOT.

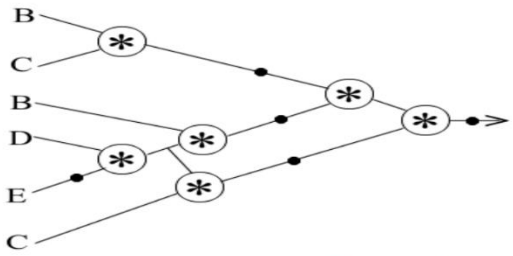

Fig. 1 Typical AIG structure based on real instance of the function

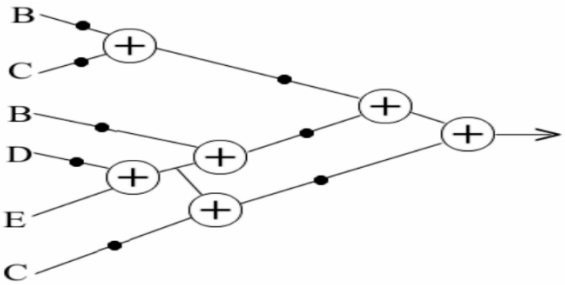

Fig. 2 Typical OIG structure based on the virtual function instance

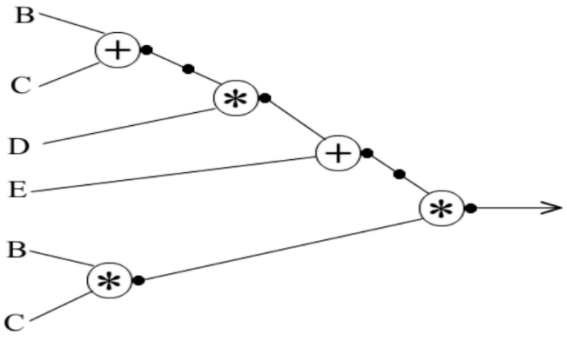

Fig. 3 NNIG-POS derived from virtual MPOS of the function

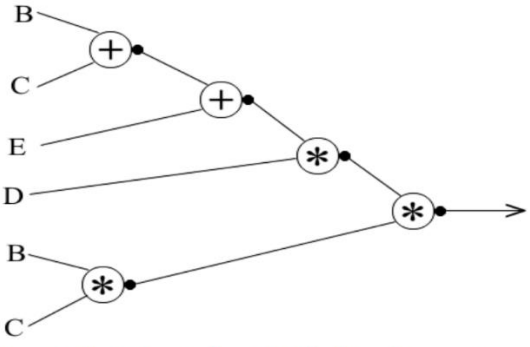

Fig. 4 NNIG-SOP created from MSOP of the real function instance

The structures are hence described by AIG(6,11), OIG(6,12) (inputs B and C have flipped edges twice), NNIG- POS(5,7) and NNIG-SOP(5,5). The cell count is the sum of the number of graph nodes and flipped edges in the BDAG. Here, for this example, the NNIG-POS is identified as the VNNIG as it is obtained by considering the virtual instance of the function and the bubble (flipped edge in case of a DAG) associated with the AND gate

connected to the output compensates for the inversion of the function polarity. In other words, NNIG-POS is constructed by a virtual POS of the function and not the POS corresponding to the real function phase, since the number of literals for MPOS for the real function instance would be 10, instead of 8 for the virtual instance. Also the node count and cell count for the real instance would 9 and 16 respectively, whereas for the virtual instance they would be 6 and 12 respectively. Indeed, the POS expression can be factored by applying distributive axiom. Forthe SOP: the literal cost for the real and virtual function phases are 8 and 10. The node count and cell count for the real phase would be 6 and 11; for the complementary phase it would be 9 and 16 respectively. NNIG-SOP is found to be the best structure for this example, decided on the basis of node count, cell count and literal cost at the graph level. For validation purpose, the different logic structures were implemented using static CMOS logic style, based on a 0.35 micron TSMC CMOS process technology. The device count for AIG, OIG, NNIG-SOP and NNIG-POS structures are 46, 48, 24 and 20 transistors respectively. Minimum size nMOS and pMOS transistors were used for simulation purpose and no tapering of the gates had been done. An input pattern which exhibits spatio-temporal correlations has been used to determine the power consumption and delay parameter of the circuits implemented. The different logic structures are governed by the design metrics, as follows: AIG(2.5969nW, 1.13ns), OIG(3.5465nW, 1.002ns), NNIG-POS(1.1833nW, 0.715ns) and NNIG-SOP(0.8572nW, 0.689ns).

V. SIMULATION MECHANISM AND RESULTS

Various non-regenerative Boolean functions have been considered for simulation studies to determine and validate the efficiency of our proposal and are listed in Table 1. Figures 5 and 6 describe the power dissipation and critical delay parameter obtained for the different graph structures, simulated under identical conditions, based on an industry standard BSIM3 device model, with a supply voltage of 3.3V and an input frequency of 100MHz.

TABLE I
LOGIC FUNCTION SPECIFICATION

Logic Function ID	ON-set specification of the Boolean function
LF1	{1,3,4,5,6,7,9,12,13,15}
LF2	{0,1,3,4,5,7,12,13,15}
LF3	{0,2,4,5,6,7,8,9,12,13}
LF4	{0,1,2,3,4,5,6,7,8,9,10,11,12,14,24,25,26,27,30}
LF5	{0,1,2,3,4,5,7,8,9,10,11,12,13,15,25,26,27,29}
LF6	{6,14,22,23,24,25,26,27,28,29,30,31}
LF7	{0,1,2,3,4,5,7,8,16,17,18,19,20,24,25,26,27,28,29,31}
LF8	{0,1,2,3,4,5,7,8,9,11,12,13,15,24,25,27,28,29,31}
LF9	{0,4,6,14,16,18,20,22,23,30}
LF10	{4,5,6,7,10,11,14,15,23,31}
LF11	{5,7,13,15,17,19,25,27,28,29,30,31}
LF12	{0,1,2,3,4,6,8,9,10,11,12,14,16,18,21,22,23,24,26}
LF13	{0,1,3,4,5,7,8,9,11,16,17,19,20,21,23,24,25,27}
LF14	{6,10,12,13,14,15,22,26,28,29,30,31}
LF15	{0,2,4,6,8,12,16,20,24,28,32,33,34,35}
LF16	{0122222,2201222,2222012,2222220}
LF17	{102222222,220122222,222210222,222222012,222222220}
LF18	{11222222,22002222,22221122,22222200}
LF19	{00122222,22210222,22222011}
LF20	{00122222,22210222,22222011}
LF21	(abc+def+ghi+jkl)
LF22	(abc+def+ghi+jkl+mno)
LF23	(abc+def+ghi+jkl+mno+pqr)
LF24	(abc+def+ghi+jkl+mno+pqr+stu)
LF25	(abc+def+ghi+jkl+mno+pqr+stu+vwx)
LF26	(abc+def+ghi+jkl+mno+pqr+stu+vwx+yza[1])
LF27	(a'b'c'+d'e'f'+g'h'i'+j'k'l')
LF28	(a'b'c'+d'e'f'+g'h'i'+j'k'l'+m'n'o')
LF29	(a'b'c'+d'e'f'+g'h'i'+j'k'l'+m'n'o'+p'q'r')
LF30	(a'b'c'+d'e'f'+g'h'i'+j'k'l'+m'n'o'+p'q'r'+s't'u')
LF31	(a'bc+d'ef+g'hi+j'kl)
LF32	(a'bc+d'ef+g'hi+j'kl+m'no)
LF33	(a'bc+d'ef+g'hi+j'kl+m'no+p'qr)
LF34	(a'bc+d'ef+g'hi+j'kl+m'no+p'qr+s'tu)
LF35	(a'bc+d'ef+g'hi+j'kl+m'no+p'qr+s'tu+v'wx)

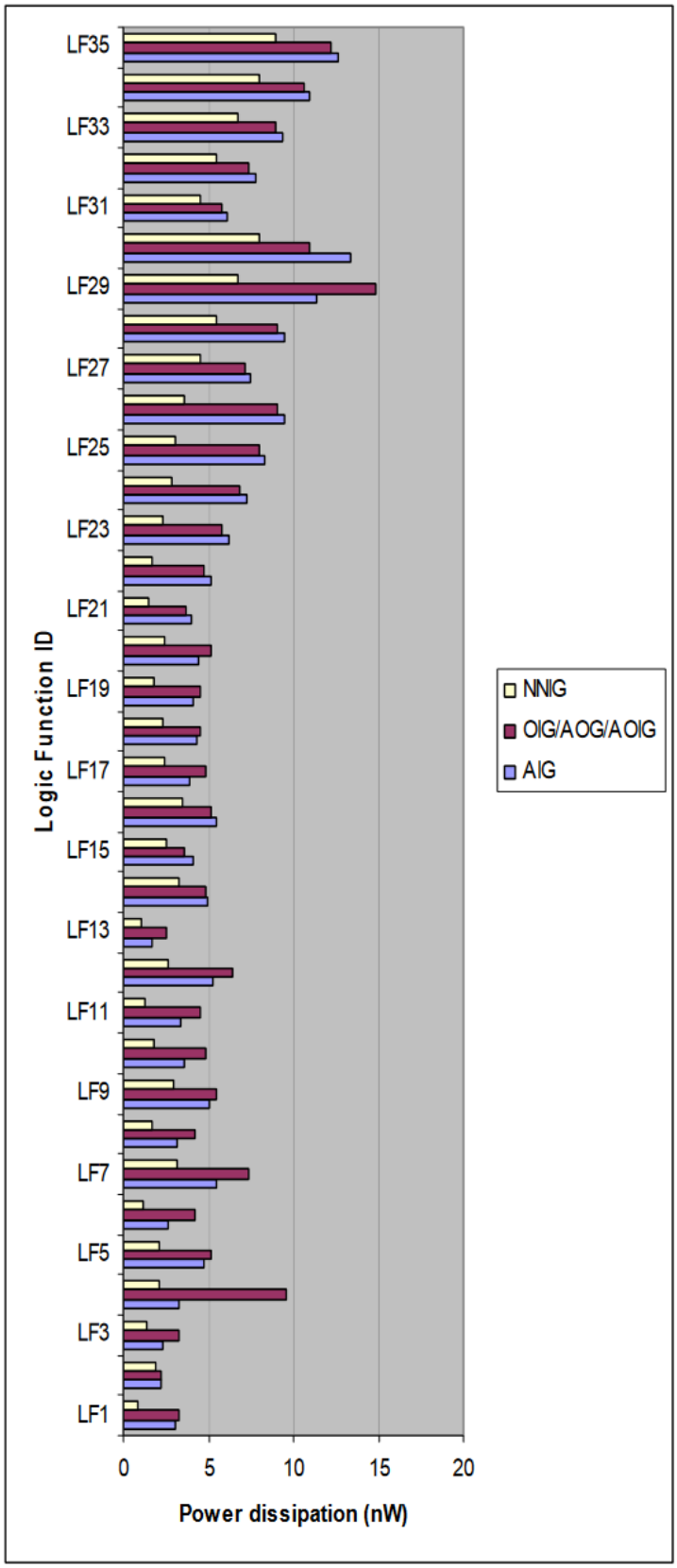

Fig. 5 Power consumption of various logic structures

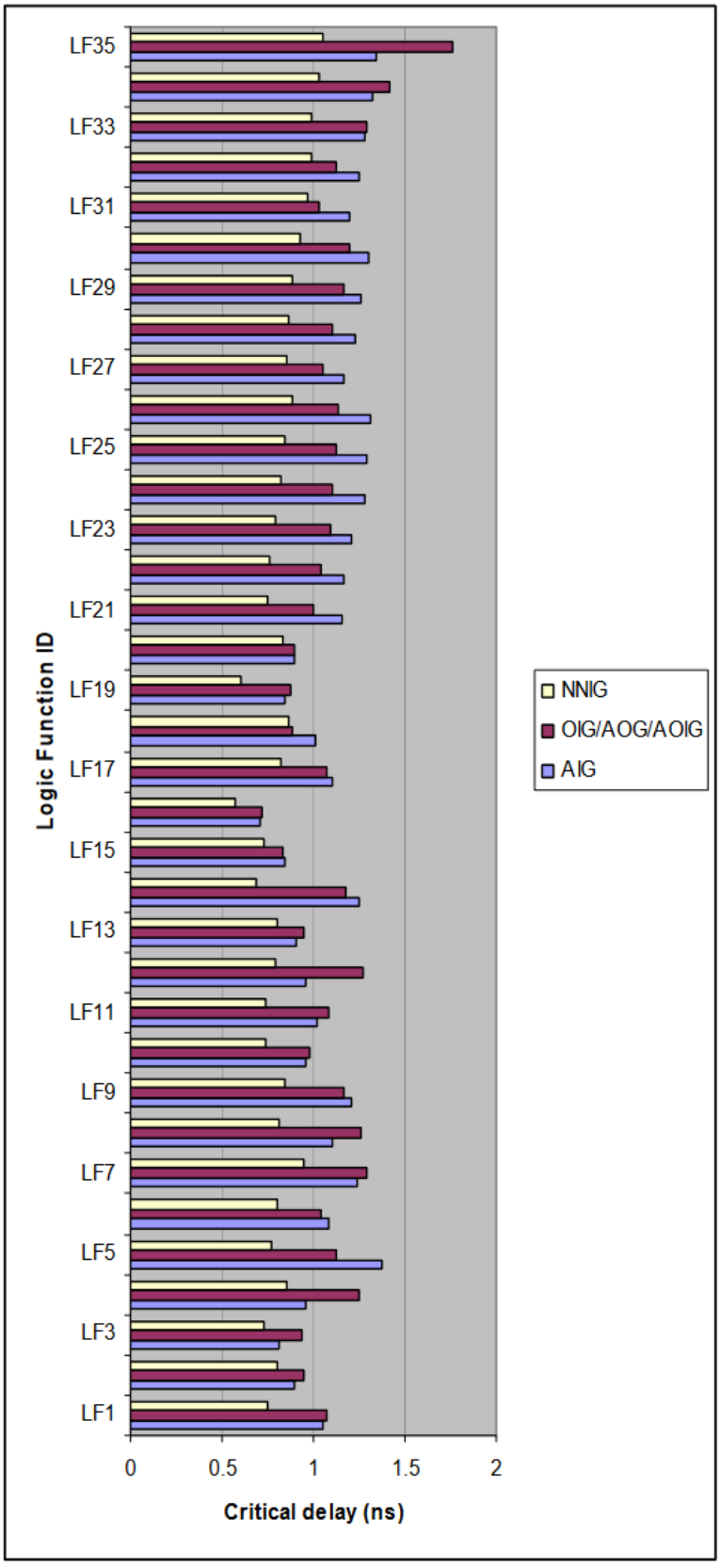

Fig. 6 Delay comparison of different graph structures

In the above table, logic functions LF1-LF15 include both arbitrary cases as well as some lower order benchmarks upto 6 inputs. Logic functions LF21-LF35 represent special functions. Logic functions 16-20 follow a ternary representation and include functions with upto 9 inputs. AIG, OIG and NNIGs are created for these. A product term of the above functions (LF16-LF20) is represented in the ternary notation of '0', '1' and '2', where '0' and '1' denote a variable appearing in the complementary and true form, while '2' indicates the absence of a variable.

Problem cases identified as LF21-LF26 represents unate and positive Achilles' heel functions, whose input file is specified by (i,c), where 'i' stands for the number of input variables and 'c' stands for the number of input cubes. For these functions, 'c' is 3 and 'i' varies from 12 to 27; AIG, AOG and NNIGs are used for circuit realizations. Logic functions LF27-LF30 represent unate and negative Achilles' heel functions (since all input variables appear in complementary form), whose input file has the same connotation as above. However, the number of input variables considered varied from 12 to 21. Functions LF31-LF35 denotes unate Achilles' heel functions with horn terms [14]. The input file bears the same specification and the number of inputs considered was from 12 until 24. For problems LF27- LF35; AIG, AOIG and NNIGs were constructed. This is because OIGs tend to be very expensive in terms of delay for such functions, even though they are created from the virtual instance of the respective functions. Hence the other structures have been considered. Construction of OIG from the real function phase for Achilles' heel functions will certainly be a disadvantage, as they would have much greater number of input cubes. For e.g. a 30 variable function would have 59049 cubes in its OFF-set. With respect to all the problem cases listed in Table 1, it is evident that functions exhibiting special properties have been considered for simulation purposes. The experimental results obtained report mean savings in power for NNIG over AIG and OIG/AOG/AOIG structures by 43.71% and 47.51% respectively. In terms of delay, the corresponding average improvement in delay for NNIG logic structures over functionally equivalent AIG and OIG/AOG/AOIG was found to be 25.85% and 24.83%. The simulation results were obtained using input patterns, which exhibit spatio-temporal correlation. The above savings in the critical design parameters have been calculated by comparing the compact and best multi-level NNIG structure for a given Boolean function with those corresponding to optimal and functionally equivalent AIGs and OIG/AOG/AOIGs respectively.

VI. CONCLUSIONS

This paper discusses a systematic procedure for the construction of power and delay optimized binary logic structures viz. NAND-NOR-Inverter Graphs for combinational logic circuits. AIGs and OIGs usually rely on the MSOP and MPOS of a function respectively for their construction, and exhibit canonicity due to various representations possible, directly resulting from different but logically equivalent expressions; the construction of NNIGs would be straightforward. This has been made possible by defining decision metrics at the technology-independent phase itself, which is not a regular feature in the creation of AIGs

and OIG/AOG/AOIGs. We essentially obtain four reduced disjunctive and conjunctive normal forms of a logic function, resulting from considering both the output function polarities. Weak factorization operations are then performed to reduce the literal count. We also apply distributive laws of Boolean algebra to this end. Then we arrive at a decision based on the initial literal count of the factored forms corresponding to either of the function phases. We then create a graph structure using 2-input logic gate primitives, in tune with the DAG specification of an arbitrary Boolean network. We now recursively employ De-Morgan's theorems [12] in order to translate the gate-level specification of the function in terms of cell analogy. The estimation of node count and cell count at this stage helps us to finalize two structures, one corresponding to the factored MSOP and the other corresponding to the factored MPOS format of the function. Indeed, the cell count is a new decision parameter introduced in this paper and would be the sum of the number of DAG nodes and flipped edges in the DAG. This approach helps to reduce the time required to arrive at an optimal data structure as the canonicity is greatly reduced and the optimal solutions can be obtained in less polynomial time from fewer choices (ideally two). The results obtained so far, indicate that NNIG logic data structures are potential candidates for enabling low power, delay and even area optimized compact multi-level combinatorial logic designs and they promise good scalability. It remains to be seen as to how the NNIGs would make a profound practical impact in reducing the time required for solving Boolean satisfiability, functional verification and formal equivalence checking problems and this provides scope for further research in this direction.

REFERENCES

[1] A. Mishchenko, and R.K. Brayton, "Scalable logic synthesis using a simple circuit structure," Proc. of International Workshop on Logic Synthesis, 2006, pp. 15-22.

[2] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-Aware AIG rewriting A fresh look at combinational logic synthesis," 43 rd ACM/IEEE Design Automation Conference, 2006, pp. 532-535.

[3] A. Mishchenko, S. Chatterjee, R. Brayton, and P. Pan, "Integrating logic synthesis, technology mapping, and retiming," ERL Technical Report, EECS Dept., UC Berkeley, April 2006.

[4] A. Mishchenko, and R.K. Brayton, "Scalable logic synthesis using a simple circuit structure," Proc. of International Workshop on Logic Synthesis, 2006, pp. 15-22.

[5] A. Mishchenko, S. Chatterjee, R. Jiang, and R.K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," ERL Technical Report, UCB, March 2005.

[6] N. Een, and N. Sorensson, "An extensible SAT-solver," 6th International Conference on Theory and Applications of Satisfiability Testing, 2003, pp. 502-518.

[7] L. Hellerman, "A catalog of 3-variable OR-Inverter and AND-Inverter logical circuits," IEEE Transactions on Electr. Comput. vol. 12, pp. 198- 223, 1963.

[8] P. Bjesse, and A. Boralv, "DAG-aware circuit compression for formal verification," International Conference on Computer Aided Design, pp. 42-49, 2004.

[9] G.L. Smith et al., "Boolean comparison of hardware and flowcharts," IBM Jour. of Research and Development, vol. 26(1), 1982, pp.106-116.

[10] A. Mishchenko, et al., “Using simulation and satisfiability to compute flexibilities in Boolean networks,” IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 25(5), May 2006, pp. 743-755.

[11] A. Mishchenko, Available: www.eecs.berkeley.edu/~alanmi/abc

[12] M. Morris Mano, Digital Design. New Jersey: Prentice Hall, 2002.

[13] P.C. McGeer, J.V. Sanghavi, R.K. Brayton, and A.L. Sangiovanni- Vincentelli, “ESPRESSO-SIGNATURE: a new exact minizer for logicfunctions,” IEEE Trans. on VLSI Systems, vol. 1(4), pp. 432-440, December 1993.

[14] Peter L. Hammer, and Alexander Kogan, “Horn functions and their DNFs,” Information Processing Letters, vol. 44(1), pp.23-29, November 1992.