Effect of Constant One and Zero, Shared and Non-decomposed Nodes on Runtime and Graph Size of the Shannon Factor Graph (SFG)

Basireddy Karunakar Reddy*, Srinivas Sabbavarapu†, Amit Acharyya‡
Department of Electrical Engineering
Indian Institute of Technology Hyderabad, Telangana, India.
*ee12m1006@iith.ac.in, †ee12p1010@iith.ac.in, ‡amit_acharyya@iith.ac.in

***Abstract*—In this paper, we propose two different algorithms for Shannon Factor Graph (SFG) construction, which can be used for cut-less mapping, to improve the runtime, graph size and required memory size. The first SFG construction algorithm does not consider the nature of the nodes (constant one or zero, non-decomposed and shared nodes) while building the SFG, whereas the second SFG construction algorithm finds out the nature of the nodes on-the-fly. We observed that the constant one and zero, shared and non-decomposed nodes can be used at the time of SFG construction to minimize the runtime and graph size significantly and to make the graph semi-canonical. The theoretical analysis and experiments performed on the standard benchmark circuits show that, by finding the constant one and zero, shared and non-decomposed nodes on-the-fly reduces the graph size by a factor of 126 and the runtime by a factor of 5.5.**

***Index Terms*—Logic synthesis, Cut-enumeration, Cut-based technology mapping, Cut-less technology mapping, Shannon decomposition theorem.**

I. Introduction

The logic synthesis is one of the important processes in the digital IC design automation flow, which decides the quality of the final physical layout [1]. Logic synthesis converts the input RTL/behavioral description into a network of standard cells (gates), called the gate-level netlist. Traditional logic synthesis process includes translation, optimization and technology mapping [2]–[4]. Technology mapping is the critical and final step in the logic synthesis process. The objective of the technology mapping in standard-cell logic synthesis is to express a given Boolean function as a network of gates chosen from a given standard-cell library to optimize some objective function such as total area or delay [1]. To simplify the mapping problem, first the Boolean function is represented as a good initial multi-level network of simple gates called the subject graph [1] through technology independent mapping. Finally the subject graph is transformed into multi-level network of library gates through the technology dependent mapping.

The technology dependent mapping is usually carried out using the cut-based Boolean matching [5]–[8]. Cut-based technology mapping techniques require the computation of K-feasible cuts and pruning, truth-tables of enumerated cuts, finding an appropriate gate for each node in the network, for every cut, using Boolean matching, and choosing the best cover based on delay and area. The cut-based technology mapping has unpleasant property of growing exponentially with the cut-size and the graph size. If there are n nodes in the graph and K is the cut size then the possible number of cuts will be $O(n^K)$. Therefore the number of cuts to be enumerated will increase exponentially with the cut size and the number of node of the graph, which increases the runtime and required memory to store the cuts drastically.

Here we propose two different algorithms for Shannon Factor Graph (SFG) construction, which can be used for cut-less mapping and analyze the effect of nature of nodes (non-decomposed, constant one and zero, and shared nodes) on the graph size and building time. The nodes of the SFG represent the Cofactors or cube cofactors value and the level, which will be used as node ID for finding the appropriate cell in the pre-computed library, and edges represent the connecting wires among the nodes. Unlike Binary Decision Diagrams (BDD) [9]–[11] and And-Inverter graph (AIG) [1], [12], the structure of the SFG helps in eliminating the cut-enumeration and pruning, computation of truth-tables and Negation-permutation-Negation (NPN) class representatives [13], [14] for each cut. Thereby it improves the runtime and reduces the required memory drastically. We found that, computation of constant zero and one, non-decomposed, and shared nodes (explained in next section) is critical in minimizing the runtime and graph size for complex circuits. By considering the nature of nodes while constructing the graph makes the SFG semi-canonical, because the nodes at each level of the SFG will have uniquely represented nodes. The proposed SFG construction algorithm (algorithm 2) computes the constant one and zero, non-decomposed and shared nodes on-the-fly, thereby it improves the overall size of the graph, which in turn reduces the final area and building time.

The remainder of the paper is organized as follows. Section II explains the basic terminology used in the paper. Section III discusses the construction of SFG and computation of non-decomposed nodes, constant one or zero nodes and shared nodes. Section IV presents the experimental results and finally section V concludes the paper.

II. Preliminaries

This section explains the basic terminology used in this paper.

978-1-4799-6965-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ISED.2014.35

A. *Shanon Decomposition Theorem*

A Shannon Decomposition is a method to represent any Boolean function as the sum of two sub-functions of the original function. A cofactor is a sub element of a Shannon Decomposition generated by setting the value of a given variable to either "0" or "1". A cofactor, which is generated for a function F by setting a variable x$_i$ to 0 is called the negative cofactor of the function F with respect to x$_i$, otherwise it is called positive cofactor (setting to "1"). A cube-cofactor is obtained by setting more than one variable to "0" and/or "1", i.e a cube-factor is a cofactor from a cofactor. Equation (1) shows the mathematical representation of Shannon decomposition theorem.

$$F(x_0,x_1,...,x_i) = x_0 * F(1,x_1,...,x_i) + x_0\prime * F(0,x_1,..,x_i) \tag{1}$$

Where * and $\prime$ represent the AND and NOT functions respectively.

For an example, the negative cofactor of the function *F(*x_0,x_1,x_2,x_3*)*=x_0*(x_1+x_2)+x_3 with respect to x_0 is x_3, whereas the cube-cofactor with respect to $x_0(=\ 0)$ and $x_3(=1)$ is 1.

B. *Non-decomposed , Constant one, Constant zero and Shared nodes*

Non-decomposed, constant one and zero, and shared nodes represent the nature of SFG nodes. A *non-decomposed node* is the node which has similar node(s) in the SFG, which have the same cofactor or cube cofactor value and level. Non-decomposed nodes improve the logic sharing, minimize the graph size and final area. If the building time of a non-decomposed node is t_b and the number of non-decomposed nodes in the SFG is *L*, then the building time and graph size will be reduced by a factor *(L-1)*t_b and *L-1* respectively. A node which receives two constant zeros (ones) is called *constant zero (one) node*, i.e the constant zero(one) node will have all zeros(ones) in its corresponding truth-table. If there are no constant zeros or ones as input to the node, then the node represents function of a typical Shannon cofactor (two AND gates, one inverter and one OR gate), otherwise the node represents an AND gate or AND gate followed by an OR gate. If the two primary outputs have the nodes, which have the same functionality and level, then those nodes are called *shared nodes*. Shared nodes minimize the size of the SFG.

III. Construction of Proposed SFG

This section explains the construction of the SFG with and without considering the non-decomposed, constant one and zero, and shared nodes and cut-less mapping technique in brief.

A. *SFG Construction without Considering the Non-decomposed, Constant one and zero, and Shared nodes*

Algorithm 1 shows the pseudo code for the SFG construction without considering the non-decomposed, constant one and zero, and shared nodes. The input to the SFG construction algorithm is truth-table and input size of the largest cells (MaxCellSize) available in the library. Using the equation (1) the Shannon cofactors and cube cofactors of the given truth-tables will be computed by successively dividing the truth-table into two equal halves. Initially, the given truth-table will be into divided two equal halves, the first half of the truth-table represents the negative cofactor and second half represents the positive cofactor. Again these two equal halves will be considered for the next phase of decomposition. This process continues till the cofactors/cube cofactors of all variables are calculated. Decimal values of the computed Shannon cofactors/ cube cofactors and their levels (number of primary input variables that are in the fan-in cone of a node) will be used as the node IDs. Fig. 1 (a) shows the basic structure of the SFG, without considering the nature of nodes, for an arbitrary 6-input Boolean function *F*

SFG construction algorithm takes the advantage of the size of the library cells during graph construction to improve the graph size and runtime. the The SFG is constructed to a level that the bottom most nodes of the SFG will have a level of MaxCellSize. This is because of the fact that the nodes which receive inputs from MaxCellSize number of primary input variables can be mapped with the library cells whose size is not less than MaxCellSize. For an instance, if *F* is a 10-input Boolean function and MaxCellSize is 4, then the bottom most nodes of SFG of the Boolean function *F* will have a level of 4, i.e only 6-variables are considered for the Shannon decomposition. Since bottom most nodes have a level 4, they can be mapped directly with the 4-input library cells. In this way, by considering the size of the library cells, the SFG construction algorithm reduces the graph size and improves the graph building time. The SFG constructed using the algorithm 1 is not canonical, because it has nodes which are redundant. In order to make the SFG semi-canonical, there should not be any constant one and zero, shared and non-decomposed nodes. The proposed algorithm 2 takes this into consideration and makes the SFG semi-canonical.

B. *SFG Construction-Considering the Non-decomposed, Constant one and zero, and Shared nodes*

Algorithm 2 shows the pseudo code for the SFG construction and computing the non-decomposed , constant one and zero, and shared nodes on-the-fly. It considers the size of the library cells and nature of nodes to minimize the graph building time and size. Fig. 1 (b) shows the basic structure of the SFG, with considering the nature of nodes, for an arbitrary 6-input Boolean function *F*. At every level, the value of Shannon cofactors or cube cofactors are checked to find the non-decomposed, constant one and zero nodes before proceeding to the next phase of decomposition. The cofactor or cube cofactor values of the nodes having the same level will be compared, then the nodes which are having the same cofactor or cube cofactor value in their truth-table (output decimal value) are classified as non-decomposed nodes and only one out of all non-decomposed nodes will be considered for the next phase of Shannon decomposition. If there are *m*

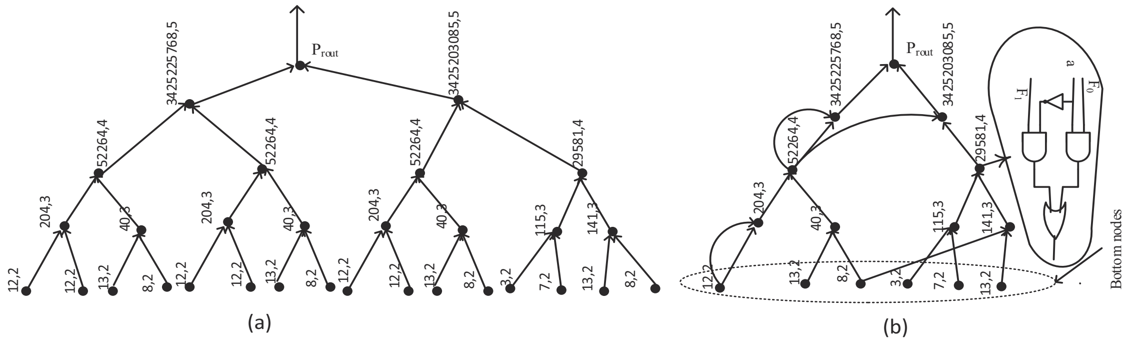

Fig. 1. Basic structure of the Shannon Factor Graph (SFG), assuming that size of the largest cells in the pre-computed library is 2 (MaxCellSize) (a) without considering the nature of nodes and b) considering the nature of nodes of a 6-input Boolean function *F*. The variable 'a' represents the hidden variable which will be used as the selection line of the multiplexer and F0 and F1 represent the negative and positive cofactors respectively

Algorithm 1 SFG Construction without considering the non-decomposed , constant one and zero, and shared nodes

```
int Truth2ShannonFactorGraph(truthtable, MaxCellSize)
{
int j,f, numvar, truthlength, tempTruth;
tempTruth=truthtable;
truthlength=length of the tempTruth;
numvar=log2(truthlength);
compute the decimal value of tempTruth save in f;
for $j$=1 to numvar-MaxCellSize
{
compute the Shannon cofactors and cube cofactors by successively dividing the tempTruth;
compute the decimal values of the cofactors and cube cofactors and save in f;
}
return f;
}
```

number of non-decomposed nodes, then only one out of *m* will be considered for the decomposition and the remaining *m-1* will be implemented from the decomposed node. So, there is no need to spend time to decompose all *m* non-decomposed nodes,which is runtime overhead. As the number of non-decomposed nodes increases, the graph building time and size decrease.

The constant one and zero nodes are found by identifying the nodes whose cofactors have all zeros or ones in their truth-table or corresponding decimal value. Once the constant one or zero nodes are found, they are no more considered for the Shannon decomposition and will be used to simplify their parent nodes. Even if the constant one or zero nodes are decomposed, the resulting nodes (children) will also be constant one or zero nodes. So considering the constant one or zero nodes for decomposition is redundant, runtime overhead

Algorithm 2 SFG construction and computation of non-decomposed, constant one and zero nodes on-the-fly

```
graphconstruct(int truth_table, int LibraryCell_Size)
{
int num_previousCubecofactor, number_of_variables, temp_table, i, j, k=1;
num_previousCubecofactor←1
number_of_variables=log$_2$(length(truth_table));
temp_truthtable=truth_table;
ShannonCofactor(1,1)=truth_table;
for $i = 2$ to num_of_variables-LibraryCell_Size+1
{
    if all Cube_cofactors are constant one/zeros
      Break;
      for $j = 1$ to num_previousCubecofactor
{
ShannonCofactor(i,k)=temp_truthtable(1:end/2);
k=k+1;
ShannonCofactor(i,k)=temp_truthtable((end/2+1):end);
k=k+1;
          }
num_previousCubecofactor ←number of cofactors in the Shannon Cofactor for $i$
Compare all the Cubecofactors and remove non-decomposed nodes and Constant one or zero Nodes
} }
```

and increases the graph size. The shared nodes among the primary outputs are determined by comparing the nodes of one primary output with the other primary output. If there are any shared nodes, only one of them will be considered for the Shannon decomposition and remaining nodes will be implemented from the decomposed node. From the Fig. 1 it is clear that, the size of the SFG can be minimized significantly even for small functions by identifying the nature of nodes on-the-fly (from 31 to 15) and experimental results show that, the reduction in SFG size is more prominent for bigger functions.

C. Cut-less Mapping in Brief

The mapping of the SFG to the library cells starts from the bottom most nodes. The decimal value and level of the bottom nodes of the SFG will be compared with the output decimal value and size of the library cells to find the appropriate match for each node. Once the bottom nodes are mapped with the library cells, then the level of the graph will be reordered to get the actual level of each node. Now, the nodes above the bottom most nodes are selected based on the multiplexers size that are presented in the library and . Every node of the SFG will have hidden variable which can be used as selection line of the multiplexer. Therefore, once the bottom most nodes are mapped with the library cells, the remaining nodes are mapped with the appropriate library cells (multiplexers) in a bottom-up fashion. Mapping continues till all nodes in the each primary output are covered.

IV. Experimental Results

We extensively verified the proposed algorithms for SFG construction with the standard benchmark circuits [16], [17]. The proposed algorithms are implemented in MATLAB running on a Xeon processor (3.4GHZ, 4GB RAM) operating in Linux-based environment. The benchmark circuits taken from [17] (circuits 10-14 in Table I), which are PLA format, converted into truth-tables manually and using the Simple-Solver [18]. Benchmark circuits taken from [16] (circuits 1-9 in Table I), which are in verilog format, converted into Boolean equations using the ABC tool [19], then truth-tables are harvested from the Boolean equations.

Table I shows the variation of graph size and runtime (graph building time) of the SFG with and without considering the non-decomposed, constant one and zero, and shared nodes. Column 1 represents the standard benchmark circuit name. Column 2 shows the number of primary inputs and outputs of the benchmark circuit. Column 4 and 5 represent the time taken to build the SFG and number of nodes of the SFG (graph size) without considering the nature of the nodes (non-decomposed, constant one and zero, and shared) respectively. Column 6 and 7 gives the graph size and time taken to build the SFG, considering the nature of nodes.

we considered 3-input library cells, so the SFG is decomposed to a level where the bottom most nodes will have a level of 3. The size of the SFG graph increases drastically, when the non-decomposed, constant one (zero) and shared nodes are not considered in constructing the SFG. This is because, these nodes will also be considered for the Shannon decomposition, which augments the graph building time and size of the SFG. Since the non-decomposed nodes will have a representative node, which will be considered for the decomposition, all these nodes can be implemented from the decomposed node assuming that there is no fan-out limitation on a node. Thus, by considering only one representative for *m* nodes can improve the runtime and graph size significantly.

Assume that there are *S* sets of non-decomposed nodes, each set has *n* nodes (level and decimal values of the cofactors are same) and t_d is the time required to decompose each node. Now each set can be implemented (*n-1* nodes) from a single node, which is considered for the Shannon decomposition. So the total time taken to decompose the nodes will be $m{*}t_b$, which saves $O(m{*}n)$ $((n-1){*}m{*}t_b)$ time and reduces the graph size of similar amount. Same is applicable for shared nodes case also. Non-decomposed nodes represent the nodes within a primary output, whereas shared nodes represent among the primary outputs.

If any constant one or zero nodes are found during the SFG construction at any level, then those nodes will not be considered for the decomposition to minimize the graph building time and size of the SFG. The Shannon decomposition of constant one or zero nodes (parent) results in constant one or zero (children), which are redundant to consider for the further decomposition. Constant one and zero nodes reduce the size and graph building time of the SFG drastically compared to the non-decomposed and shared nodes. Column 8 sows the runtime ration with and without considering the nature of nodes. At an average non-decomposed, constant one and zero, and shared nodes minimize the runtime by a factor 5.5 (for few circuits it is around 100). But the interesting observation is for few circuits, the runtime ratio is 1, this is due to the presence of the constant one or zero and non-decomposed nodes near the bottom most nodes which increases the runtime. Similarly for few circuits the variation in the graph size (circuits 8, 11-14) is minimum, this is because of, the input size of the benchmark circuits is almost equal to the size of the library cells. At an average, considering the constant one and zero, shared, non-decomposed nodes during graph construction reduces the size of the SFG by a factor of 126 (for few circuits it is around 700). So, by finding the nature of nodes on-the-fly makes the SFG semi-canonical, which in turn minimizes the graph size and runtime.

V. Conclusion

The proposed Shannon Factor graph's structure helps in eliminating the cut-enumeration, which is computationally expensive task. Therefore, SFG facilitates the cut-less mapping to overcome the runtime and memory bottlenecks for the today's highly complex designs. The nature of the nodes of the SFG makes it semi-canonical and are the potential candidates for minimizing the runtime and graph size. The constant one and zero, shared and non-decomposed nodes will not only minimize the runtime and memory but also the final circuit area also. The proposed SFG construction algorithm finds the constant one and zero, non-decomposed and shared nodes on-the-fly,thereby improving the runtime and graph size significantly. The SFG can also be applied to the methodology proposed in [7], [8]. Analysis of effect of non-decomposed, constant one and zero, and shared nodes on the final area and delay forms the part of our future work.

TABLE I
COMPARISON OF RUNTIME AND GRAPH SIZES OF THE PROPOSED SFG WITH AND WITHOUT CONSIDERING THE NON-DECOMPOSED, CONSTANT ONE AND ZERO, AND SHARED NODES

S. No	Circuit name	Inputs/Outputs	Proposed SFG without NCS*		Proposed SFG with NCS*		Speed up (R1/R2)	(S1/S2)
			Runtime(R1)	Graph size(S1)	Runtime(R2)	Graph size(S2)		
1	cm138	6/9	0.0128	135	0.005	27	2.6	5
2	cmb	16/4	0.576	65532	0.0381	85	15.2	770
3	cm163a	16/5	0.717	81915	0.116	131	6.2	625
4	cm162a	14/5	0.17	20475	0.0769	96	106	193
5	cm152a	11/1	0.007	511	0.006	88	1.2	5.8
6	alu2	10/6	0.011	1536	0.01	271	1.1	5.7
7	cm151a	12/2	0.02	2046	0.02	118	1	17
8	ex4	6/9	0.019	135	0.0083	101	2.2	1.4
9	ex1	9/19	0.64	2413	0.11	376	5.7	6.4
10	max46	9/1	0.002	127	0.003	72	0.67	1.8
11	7bit-even parity	7/1	5.76e−4	31	5.12e−4	5	1.12	6.2
12	mux4	6/1	3.68e−4	15	3.1e−4	9	1.2	1.67
13	majority	5/1	2.41e−4	7	2e−4	6	1.2	1.2
14	4gt13	4/1	2.4e−4	3	2.4e−4	3	1	1
Total			**2.2**	**174881**	**0.4**	**1388**		

*NCS=Non-decomposed, Constant one and zero, and shared nodes

ACKNOWLEDGMENT

This work is partly supported by Department of Electronics and Information Technology (DeitY), India under the "IOT for Smarter Healthcare" under Grant No: 13(7)/2012-CCBT, Dated 25th Feb, 2013. We thank Dr. Alan Mischenko, UC Berkeley for helping in harvesting the Boolean equations from the benchmark circuits.

REFERENCES

[1] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping" , IEEE Trans. CAD2006, Vol. 25(12), pp. 2894-2903.

[2] J. Jie-Hong Roland, Srinivas Devadas, "Logic Synthesis in a Nutshell", November 2009.

[3] Aniel D. Gajski, Robert H. Kuhn Guest editors ,"introduction to New VLSI tools," University of Illinois, Gould Research Center, IEEE computer'1983, vol.10, pp. 11-14.

[4] Robert A. Walker and Donald E. Thomas, "A Model of Design Representation and Synthesis," IEEE DAC1985, pp.453-459.

[5] J. Cong, C. Wu and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," Proc. FPGA '1999.

[6] U. Hinsberger and R. Kolla, "Boolean matching for large libraries," Proc. DAC'1998, pp. 206-211.

[7] B. Karunakar Reddy, Srinivas Sabbavarapu, Amit Acharyya, "A new VLSI IC design automation methodology with reduced NRE costs and time to market using the NPN class representation and functional symmetry", IEEE International Symposium on Circuits and Systems (ISCAS), Australia, 1-5 June,2014 (in press).

[8] B.Karunakar Reddy, S. Sabbavarapu, K. Gupta, R. Prabhat, A. Acharyya, R. A. Shafik and J.Mathew, "A Novel and Unified Digital IC Design and Automation Methodology with Reduced NRE Cost and Time-to-Market", IEEE International Symposium on Electronic System Design, Singapore, 12-13 December, 2013, pp. 36-40.

[9] C. Y. Lee, "Representation of switching circuits by binary decision programs", *Bell System Techn. J.*, vol. 38, no. 4, June 1959, pp. 985-999.

[10] S. B. Akers, "Functional testing with binary decision diagrams," in Eighth Annual Conf. Fault-Tolerant Computing, 1978, pp. 75-82.

[11] R. Bryant, "Graph-based algorithms for boolean function manipulation, IEEE Trans. Computers, vol. 35, Aug. 1986, pp. 677691.

[12] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification, IEEE Trans. CAD, Vol. 21(12), 2002, pp. 1377-1394.

[13] V. P. Correia, A. Reis, "classifying n-Input Boolean functions, in Proc. IWS'2001.

[14] D. Chai,A. Kuehlman,"Building a better Boolean matcher and symmetry detector, proc.of DATE2006,vol.1, pp.1-6.

[15] Z. Huang, L. Wang, Y. Nasikovskiy, A. Mishchenko, "Fast Boolean Matching for small Practical Functions, proc.of IWLS2013.

[16] www.eecs.berkeley.edu/ alanmi/benchmarks/

[17] http://www.revlib.org/

[18] home.roadrunner.com/ ssolver/

[19] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. http://wwwcad.eecs.berkeley.edu/ alanmi/abc.