International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:4, 2007

Compact Binary Tree Representation of Logic Function with Enhanced Throughput

Padmanabhan Balasubramanian, Cemal Ardil

***Abstract*—**An effective approach for realizing the binary tree structure, representing a combinational logic functionality with enhanced throughput, is discussed in this paper. The optimization in maximum operating frequency was achieved through delay minimization, which in turn was possible by means of reducing the depth of the binary network. The proposed synthesis methodology has been validated by experimentation with FPGA as the target technology. Though our proposal is technology independent, yet the heuristic enables better optimization in throughput even after technology mapping for such Boolean functionality; whose reduced CNF form is associated with a lesser literal cost than its reduced DNF form at the Boolean equation level. For cases otherwise, our method converges to similar results as that of [12]. The practical results obtained for a variety of case studies demonstrate an improvement in the maximum throughput rate for Spartan IIE (XC2S50E-7FT256) and Spartan 3 (XC3S50-4PQ144) FPGA logic families by 10.49% and 13.68% respectively. With respect to the LUTs and IOBUFs required for physical implementation of the requisite non-regenerative logic functionality, the proposed method enabled savings to the tune of 44.35% and 44.67% respectively, over the existing efficient method available in literature [12].

***Keywords*—**Binary logic tree, FPGA based design, Boolean function, Throughput rate, CNF, DNF.

I. Introduction

THE issue of performance enhancement has been a subject matter of much research [1] [2] [3] [4] [5]. Also the relevance of FPGAs based on LUTs in the last decade has fostered numerous efforts in finding effective methods to minimize and decompose functions. This paper deals with a novel technology-independent synthesis methodology to realize compact and throughput enhanced binary tree structures for combinational logic circuits, by way of reducing the logic depth. A number of techniques mentioned in [7] [8] [9] are technology-independent and aim at reducing the logic depth of the binary tree representing a Boolean network by restructuring. Directed acyclic directed graphs (DAG) are generally used to effectively represent single output combinational logic circuit functionality. A rooted DAG may be unfolded to a tree in such a way that no multiple-fanout nodes exist, except for the primary circuit inputs. Each internal node is labeled with a logical operator, AND and/or OR, although other operators are also used depending upon the functionality. In this work, we are primarily concerned with function representations employing just these two types of Boolean operations. A labeled edge (dot appearing on an edge) in a DAG or a binary tree would correspond to a logical inversion or negation operation. Let us have a reasonable and valid assumption that all DAGs are reduced and that isomorphism is not exhibited in the sub-DAGs.

Tree-height reduction was indeed proposed [6] in the scope of compiler optimization, for code generation in multiprocessor systems. Given the underlying inherent complexity of the problem, timing optimization is sought after, after the size of the Boolean network representing the circuit has been reduced. Even extraction of kernels, that can be shared, may lead to an increase in the depth of the network as an associated effect. This makes it clear that sharing logic is not always deemed to be a good approach, when considering the issue of timing optimization. A technique that performs logic decomposition during technology mapping has been proposed in [10] [11]. However, the accuracy of this approach is traded off for a higher computational cost. A recent activity [12] addresses the issue of delay improvement through functional decomposition. It actually builds on logic bi-decomposition of Boolean functions [13] [14] and also uses weak algebraic factorization operations. It implicitly relies upon OR disjunction for functional bi-decomposition. Then it combines this strategy with tree-height reduction of resulting Boolean expressions. Though it leads to enhancement in performance, vis-à-vis achieving logic depth reduction, the quasi-algebraic decomposition was normally performed on the minimized disjunctive normal form (DNF) [15], by iteratively applying a combination of associative, distributive and commutative (ACD) laws.

The remaining portion of this paper is organized as follows. In section 2, we introduce a novel terminology, namely the description set of a Boolean term and give its definition. Section 3 elucidates the proposed method by means of an illustrative example and compares it with the solution obtained using the ACD based algorithm [12] at both the technology-independent and technology-dependent phases. Section 4 depicts the simulation results obtained for several Boolean functions. A comparison of the methods in terms of the maximum operating frequency achievable for the designs is given in this section. The resource utilization summary is also listed in this section. Finally, we make the concluding remarks in the next section.

Padmanabhan Balasubramanian is with the School of Computer Science, The University of Manchester, Manchester, MAN M13 9PL UK (phone: +44-161-275 6294; e-mail: spbalan04@gmail.com, padmanab@cs.man.ac.uk).

Cemal Ardil is with the National Academy of Aviation, Baku, Azerbaijan (e-mail: cemalardil@gmail.com).

II. Description set of a Boolean term

A new terminology is proposed, namely the description set of a Boolean term (sum term or product term). The description set of a sum term [product term], shall be represented by the notation $D(S_i)$ [$D(P_i)$].

$D(S_i)$ specifies the set of all literals in their actual form, that the particular sum term S_i is dependent upon for its evaluation to a logic value of '0' and $D(P_i)$ indicates the set of all literals in their respective form, that a product term P_i depends upon for its evaluation to a logic value of '1'.

For e.g. let a sum-of-disjoint products (SoDP) function be, Z = AC'DE + B'FG', where there are two disjoint product terms; P_1 = AC'DE and P_2 = B'FG'. Hence $D(P_1)$ = {A, C',D,E} and $D(P_2)$ = {B',F,G'}.

The description set for a Boolean function would then be the union of the description sets of all its individual terms. For the above example, it is given by, $D(Z) = D(P_1) \cup D(P_2)$.

III. Illustration of Proposed heuristic

Let us take an arbitrary logic function, F to describe the effectiveness of our proposal. Let F(Q,R,S,T,U,V,W) be described by the following minimized expression,

$$F = TRU+TRV+ST'W+SU+SV+QT'W+QU+QV \quad (1)$$

Using the ACD based heuristic as described in [12], two logically equivalent and irredundant reduced expressions are obtained as follows,

$$F = (TR)\cdot(U+V)+(Q+S)\cdot(T'W+U+V) \quad (2)$$

$$F = (TR+Q+S)\cdot(U+V)+(T'W)\cdot(Q+S) \quad (3)$$

Both the above Boolean equations (2) and (3) have the same input literal cost. The reduced conjunctive normal form (CNF) equivalent for (1) is given by,

$$F = (T+S+Q)\cdot(R+S+Q)\cdot(T'+U+V)\cdot(W+U+V) \quad (4)$$

For (4), we could write $D(S_1)$ = {T,S,Q}, $D(S_2)$ = {R,S,Q}, $D(S_3)$ = {T',U,V} and $D(S_4)$ = {W,U,V}. We perform the union of the description set of a sum term with all other sum terms of (4) and we get the following: $D(S_1) \cup D\{S_2\}$= {S,Q}, $D(S_1) \cup D(S_3)$ = { }, $D(S_1) \cup D(S_4)$ = { }, $D(S_2) \cup D(S_3)$ = { }, $D(S_2) \cup D(S_4)$ = { } and $D(S_3) \cup D(S_4)$ = {U,V}. We now enumerate the cardinality of the above union and thereby obtain $|D(S_1) \cup D(S_2)| = 2$, $|D(S_1) \cup D(S_3)| = 0$, $|D(S_1) \cup D(S_4)| = 0$, $|D(S_2) \cup D(S_3)| = 0$, $|D(S_2) \cup D(S_4)| = 0$ and $|D(S_3) \cup D(S_4)| = 2$.

In general, for a function whose minimized two-level CNF expression contains 'k' product terms, the first product term, say, P_1 could be combined with (k-1) different product terms, the second product term, P_2 could be combined with (k-2) distinct product terms till the $(k-1)^{th}$ product term, which could be combined with just one another different product term at the end.

As a further generalization, it can be intuitively observed that if the total number of distinct product terms in the reduced two-level representation of a logic function is 'n'; whether 'n' is 'odd' or 'even'; the total number of set union operations required to be performed would be O[n(n-1)/2].

Now we make a decision with regard to grouping those terms, whose degree of literal matching is the highest, as determined by the cardinality of the union of the description set of all possible combinations of two unique Boolean terms. Therefore for (4), we find that S_1 and S_2 can be combined using the distributive law; similarly S_3 and S_4 are candidates to be combined using the same axiom. After applying the D rule for the appropriate terms of (4), which could be grouped, we get the following reduced expression,

$$F = (TR+S+Q)\cdot(T'W+U+V) \quad (5)$$

Comparing (2) [also (3)] and (5), we find that there is a savings of 20% in terms of literal count. After representing the tree structures for (2) and (5) in accordance with the DAG specification and with sharing of nodes permitted, we observe that there is a reduction in the number of operators and logic depth by 12.5% and 25%, for the latter in comparison with the former. Without node sharing, and for the worst case realization, the respective savings for the proposed method would be 22.22% and 40% respectively.

The binary tree representation with node sharing for (1), given by (2), is shown in fig. 1. The symbols ⊛ and ⊕ denote Boolean AND and Boolean OR operators respectively and these are referred to as atomic operators (AO) [16].

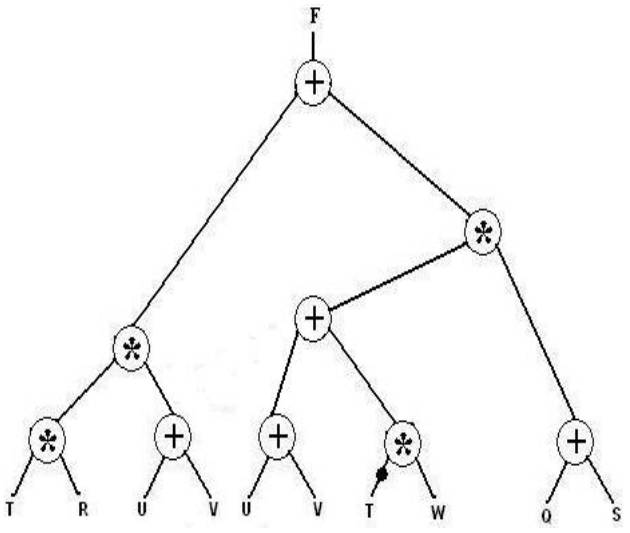

Fig. 1 Binary tree representation for (1) based on ACD heuristic

Theoretically speaking, the maximum operating frequency for fig. 1, given as a reciprocal of the longest path delay or critical path delay is given by,

$$f_{max} = \frac{1}{t_{AND} + 3t_{OR}} = \frac{1}{2t_{AND} + 2t_{OR}} \quad (6)$$

For representation without duplication of nodes and with no node sharing, the upper bound on the maximum frequency is,

$$f_{max} = \frac{1}{2t_{AND} + 3t_{OR}} \quad (7)$$

Fig. 1 is characterized by a maximum logic depth of 4 (specified by the number of nodes in the longest path from any of the primary inputs to a primary output for a MISO function) and maximum operating frequencies of 89.847 MHz and 101.626 MHz for technology mapping with Spartan IIE (XC2S50E-7FT256) and Spartan 3 (XC3S50-4PQ144) FPGA logic families as targets. The binary tree structure consumed 5 basic logic elements (LUTs of FPGA) and 16 input-output buffers for physical realization.

The binary logic tree representation corresponding to (5) is depicted by fig. 2.

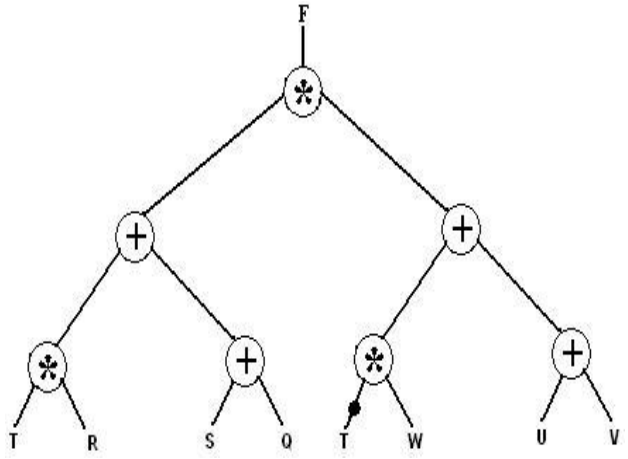

Fig. 2 Binary tree representation for (1) based on proposed method

As seen above, this tree representation requires less number of nodes than fig. 1 and the theoretical upper bound on the maximum throughput rate is given by the expression,

$$f_{max} = \frac{1}{t_{AND} + 2t_{OR}} = \frac{1}{2t_{AND} + t_{OR}} \quad (8)$$

The maximum logic depth of the tree structure is 3 and the highest operating frequency for Spartan IIE and Spartan 3 FPGA logic families is found to be 99.009 MHz and 118.203 MHz respectively. Also the structural representation required 3 basic logic elements and 9 input-output buffers for implementation with the above technology targets.

For this particular case study, we find that the throughput rate is increased by 10.19% and 16.31% for the FPGA target families in the above order. With respect to the basic logic elements and input-output buffers needed for technology mapping, corresponding savings of 40% and 43.75% has resulted for the proposed procedure over that of [12].

IV. Simulation Mechanism and Practical Results

Various combinational logic functions in canonical form of various types were considered to substantiate the theoretical claims by validating with experimental results. The functionalities in PLA format were first minimized using a commercial industry standard two-level logic minimizer, such as ESPRESSO [17] and they are listed in Table 5 (made available as an appendix).

The binary tree structures highlighting the BDAG representation for the combinational circuits were realized using the ACD rules based methodology described in [12]. VHDL coding was done for all the functions using structural modeling style with gate-level primitives strictly conforming to the binary DAG specification. The detailed design summary and timing reports were obtained after post place and route stage. The maximum operating frequency of the different designs was then determined as a reciprocal of the maximum combinational logic path delay.

The reduced conjunctive normal forms for the functions can be obtained by two methods; either by running a direct sum-of-products to product-of-sums subroutine or by considering the complementary phase of the function and a straightforward conversion to reduced product-of-sums expression could be done. Infact, a high level language implementation of the modified Quine-McCluskey's method for two-level logic minimization [18] can also be used in this regard. Next, the description set for the different product terms corresponding to each and every function was obtained as per the definition given in section 2. Set union operations were then performed on the different sets and the candidates suitable for grouping were found according to the method explained in section 3. Distributive axiom was applied, so that the function now tends to comprise reduced, compact and read-once functionality for the sub-functions, though not in the original function. Then the tree representation was created using the basic atomic operators and VHDL coding was done using a similar modeling style. The design summary and timing reports were obtained after the placement and routing phase.

The simulations were all performed with Xilinx project navigator suite targeting Spartan IIE and Spartan 3 FPGA boards. The Spartan FPGA logic families are ideally suited for gate-level designs [19].

Table 1 gives a description of the comparison between the two schemes in terms of the logical operators required and literal count. Table 2 shows the maximum throughput rate for the synthesized tree representations corresponding to the desired Boolean functionality, based on the two different methods. Table 3 gives the amount of basic logic elements utilized (BEL) for the different techniques and Table 4 gives an account of the input-output buffers (IOBUF) utilized for the two schemes.

TABLE I
LOGICAL OPERATORS AND LITERAL COST COMPARISON

Function ID	ACD_BDAG		P_BDAG	
	N_{AO}	N_{IL}	N_{AO}	N_{IL}
$Z1^{5}$	3	8	3	6
$Z2^{7}$	5	12	3	8
$Z3^{9}$	6	18	4	10
$Z4^{11}$	6	24	4	12
$Z5^{8}$	6	17	4	9
$Z6^{9}$	5	14	4	10
$Z7^{7}$	5	10	4	10
$Z8^{8}$	5	15	4	9
$Z9^{6}$	4	10	3	7
$Z10^{8}$	5	17	4	9
$Z11^{5}$	4	7	3	6
$Z12^{7}$	5	15	3	8
$Z13^{6}$	3	8	3	7
$Z14^{9}$	5	16	4	10
$Z15^{8}$	5	18	4	9
$Z16^{6}$	5	8	4	7
$Z17^{10}$	7	26	4	11
$Z18^{9}$	6	13	4	10
$Z19^{9}$	5	20	4	10
$Z20^{8}$	4	7	3	6
$Z21^{10}$	5	15	3	8
$Z22^{10}$	3	8	3	7
$Z23^{11}$	5	16	4	10
$Z24^{12}$	5	8	4	7
$Z25^{16}$	7	26	4	11
$Z26^{12}$	6	13	4	10
$Z27^{10}$	5	20	4	10
$Z28^{8}$	3	8	3	6
$Z29^{11}$	5	12	3	8
$Z30^{10}$	6	18	4	10
$Z31^{10}$	6	17	4	9
$Z32^{15}$	5	14	4	10
$Z33^{11}$	5	10	4	10
$Z34^{11}$	5	15	4	9
$Z35^{14}$	5	17	4	9
Total	175	500	129	308

ACD_BDAG – ACD rules based BDAG and P_BDAG – Proposed BDAG;
LFM^{n}: LF – Logic Function, M – Function ID, n – number of inputs

TABLE II
MAXIMUM OPERATING FREQUENCY (MHZ) FOR DIFFERENT FPGA TARGETS

Function ID	Spartan IIE (XC2S50E-7FT256)		Spartan 3 (XC3S50-4PQ144)	
	ACD_BDAG	P_BDAG	ACD_BDAG	P_BDAG
$Z1^{5}$	98.717	120.482	118.203	136.054
$Z2^{7}$	94.877	98.717	104.384	118.203
$Z3^{9}$	81.103	90.579	93.545	105.597
$Z4^{11}$	78.989	88.183	87.951	109.051
$Z5^{8}$	94.697	91.912	104.603	117.096
$Z6^{9}$	90.827	90.171	98.328	116.822
$Z7^{7}$	92.937	99.009	116.959	118.203
$Z8^{8}$	86.58	96.154	98.328	103.842
$Z9^{6}$	92.937	120.482	116.959	123.001
$Z10^{8}$	82.034	100.402	98.232	109.409
$Z11^{5}$	120.919	124.069	142.045	145.772
$Z12^{7}$	89.847	99.009	101.626	118.203
$Z13^{6}$	91.912	120.919	111.483	142.045
$Z14^{9}$	90.253	90.579	100.705	105.597
$Z15^{8}$	72.833	91.912	82.508	116.959
$Z16^{6}$	86.505	120.482	104.167	123.001
$Z17^{10}$	81.699	91.075	86.356	108.932
$Z18^{9}$	88.183	92.937	108.578	116.959
$Z19^{9}$	79.177	93.197	91.241	118.483
$Z20^{8}$	84.317	98.717	101.729	118.203
$Z21^{10}$	83.822	86.505	86.281	109.051
$Z22^{10}$	87.413	90.579	100.2	105.597
$Z23^{11}$	72.992	87.336	87.413	111.111
$Z24^{12}$	84.531	89.445	98.039	108.578
$Z25^{16}$	82.988	86.73	96.618	97.371
$Z26^{12}$	84.531	91.912	98.039	108.225
$Z27^{10}$	87.413	92.937	100.2	103.842
$Z28^{8}$	86.059	98.717	98.328	118.203
$Z29^{11}$	72.993	87.336	87.413	111.111
$Z30^{10}$	83.822	86.505	86.281	109.051
$Z31^{10}$	87.413	90.579	100.2	105.597
$Z32^{15}$	80.064	86.73	96.618	92.851
$Z33^{11}$	91.324	89.445	103.95	109.051
$Z34^{11}$	83.963	82.85	98.039	108.932
$Z35^{14}$	83.682	84.034	88.261	101.729
Total	3032.353	3350.627	3493.810	3971.732

TABLE III
BASIC LOGIC ELEMENTS (LUTS OF FPGA) FOR SPARTAN IIE AND SPARTAN 3

Function ID	Spartan IIE (XC2S50E-7FT256)		Spartan 3 (XC3S50-4PQ144)	
	ACD_BDAG	P_BDAG	ACD_BDAG	P_BDAG
$Z1^{5}$	3	2	3	2
$Z2^{7}$	6	3	6	3
$Z3^{9}$	7	5	7	5
$Z4^{11}$	10	4	10	4
$Z5^{8}$	6	3	6	3
$Z6^{9}$	5	4	5	4
$Z7^{7}$	3	3	3	3
$Z8^{8}$	5	6	5	6
$Z9^{6}$	3	2	3	2
$Z10^{8}$	7	3	7	3
$Z11^{5}$	2	2	2	2
$Z12^{7}$	5	3	5	3
$Z13^{6}$	3	2	3	2
$Z14^{9}$	6	5	6	5
$Z15^{8}$	7	3	7	3
$Z16^{6}$	3	2	3	2
$Z17^{10}$	10	4	10	4
$Z18^{9}$	4	3	4	3
$Z19^{9}$	7	3	7	3
$Z20^{8}$	5	3	5	3
$Z21^{10}$	8	4	8	4
$Z22^{10}$	7	5	7	5
$Z23^{11}$	7	4	7	4
$Z24^{12}$	11	4	11	4
$Z25^{16}$	13	6	13	6
$Z26^{12}$	11	5	11	5
$Z27^{10}$	7	5	7	5
$Z28^{8}$	5	3	5	3

$Z29^{11}$	7	4	7	4
$Z30^{10}$	8	4	8	4
$Z31^{10}$	7	5	7	5
$Z32^{15}$	13	6	13	6
$Z33^{11}$	8	4	8	4
$Z34^{11}$	11	4	11	4
$Z35^{14}$	9	5	9	5
Total	239	133	239	133

TABLE IV
INPUT-OUTPUT BUFFERS REQUIRED FOR THE TWO SCHEMES

Function ID	Spartan IIE (XC2S50E-7FT256)		Spartan 3 (XC3S50-4PQ144)	
	ACD_BDAG	P_BDAG	ACD_BDAG	P_BDAG
$Z1^{5}$	9	7	9	7
$Z2^{7}$	15	9	15	9
$Z3^{9}$	19	11	19	11
$Z4^{11}$	25	13	25	13
$Z5^{8}$	18	10	18	10
$Z6^{9}$	15	11	15	11
$Z7^{7}$	11	9	11	9
$Z8^{8}$	16	10	16	10
$Z9^{6}$	11	8	11	8
$Z10^{8}$	18	10	18	10
$Z11^{5}$	8	7	8	7
$Z12^{7}$	16	9	16	9
$Z13^{6}$	9	8	9	8
$Z14^{9}$	17	11	17	11
$Z15^{8}$	19	10	19	10
$Z16^{6}$	9	8	9	8
$Z17^{10}$	27	12	27	12
$Z18^{9}$	14	11	14	11
$Z19^{9}$	21	11	21	11
$Z20^{8}$	16	9	16	9
$Z21^{10}$	24	11	24	11
$Z22^{10}$	18	11	18	11
$Z23^{11}$	22	12	22	12
$Z24^{12}$	34	13	34	13
$Z25^{16}$	38	17	38	17
$Z26^{12}$	34	13	34	13
$Z27^{10}$	18	11	18	11
$Z28^{8}$	16	9	16	9
$Z29^{11}$	22	12	22	12
$Z30^{10}$	24	11	24	11
$Z31^{10}$	18	11	18	11
$Z32^{15}$	37	17	37	17
$Z33^{11}$	19	13	19	13
$Z34^{11}$	33	13	33	13
$Z35^{14}$	24	16	24	16
Total	694	384	694	384

V. CONCLUSION

This paper deals with a technology-independent synthesis methodology for combinational logic functionality that typically precedes the technology-mapping phase. An effective technique to address the important issue of throughput enhancement via, timing optimization, made possible by way of reducing the logic depth in a binary logic tree representation is discussed in this paper. A fair degree of correlation is observed between the depth of the Boolean network at the technology-independent stage represented by a tree and the practical critical delay parameter obtained experimentally; however, it turns out to be contrary in some cases after the technology-mapping phase. The approach seems to yield optimization in the throughput rate for a wide variety of problems, which tend to have compact conjunctive normal forms in comparison with disjunctive normal forms, with the degree of compactness measured in terms of literal count at the Boolean equation level.

The effectiveness of our contribution is evident from improved results of reachability along the computationally intensive path. Through extensive simulation studies, we infer that the proposed methodology is promising, as it enables higher operating frequency and less resource utilization (FPGA resources) in parallel for significant number of case studies. We have successfully addressed the issues of delay improvement and area reduction, highlighted in [12], by exploring the available design space and achieved enhancement in performance.

Before technology mapping, with respect to the reduced expressions governing the actual logic description, we find that in terms of the atomic operators and input literal count, the proposed procedure enabled savings of 26.29% and 38.4% respectively. From the experimental results obtained, we find that the average improvement in performance (measured in terms of maximum operating frequency) has been 10.49% and 13.68% for Spartan IIE and Spartan 3 FPGA logic family targets respectively. The corresponding average decrease in LUTs for the logic families stated in the above order has been the same and is around 44.35%. Based on the number of input-output buffers required for physical realization of the desired functionality, the proposed method effected mean savings to the tune of 44.67% for both the logic families.

For functions with DNF forms more compact than its CNF forms, the proposed heuristic returns the same results as that of [12], while for the contrary, the approach enables decent enhancement in throughput rate, whilst ensuring minimum resource utilization. The approach is pragmatic and results in tree representations for non-regenerative logic functions, which promise improved performance, evident from several problem cases considered in this work.

ACKNOWLEDGMENT

The authors would like to thank Mrs. Prathibha for her assistance in this work.

REFERENCES

[1] R. Ashenhurst, "The decomposition of switching functions," *Proc. of International Symposium. on Switching Theory*, 1959, pp. 74-116.
[2] J. Baer, and D. Bovet, "Compilation of arithmetic expressions for parallel computations," *Proc. of IFIP Congress*, 1968, pp. 340-346.
[3] J. Beatty, "An axiomatic approach to code optimization for expressions," *Journal of ACM*, vol. 19(4), 1972, pp. 613-640.

[4] R. Brayton, and C. McMullen, "The decomposition and factorization of Boolean expressions," *Proc. of International Symposium on Circuits and Systems*, 1982, pp. 49-54.
[5] J. Vasudevamurthy, and J. Rajski, "A method for concurrent decomposition and factorization of Boolean expressions," *Proc. of International Conf. on Computed-Aided Design*, 1990, pp. 510-513.
[6] D. Kuck, *The Structure of Computers and Computation*, Wiley, 1978.
[7] K. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic," *Proc. of IEEE/ACM International Conf. on Computer-Aided Design*, 1988, pp. 282-285.
[8] K. Chen, and S. Muroga, "Timing optimization for multi-level combinational circuits," *Proc. of ACM/IEEE Design Automation Conf.*, 1990, pp. 339–344.
[9] H.Touati, H.Savoj, and R.Brayton, "Delay optimization of combinational circuits by clustering and partial collapsing," *Proc. of IEEE/ACM International Conf. on Computer-Aided Design*, 1991, pp. 188-191.
[10] Eric Lehman, and Yosinori Watanabe, "Logic Decomposition during Technology Mapping," *Proc. of IEEE/ACM International Conf. on Computer-Aided Design*, 1995, pp. 264-271.
[11] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 16(8), August 1997, pp. 813-834.
[12] J. Cortadella, "Timing-Driven Logic Bi-Decomposition," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 22(6), June 2003, pp. 675–685.
[13] S. Yamashita, H. Sawada, and A. Nagoya, "New methods to find optimal nondisjoint bi-decompositions," *Proc. of ACM/IEEE Design Automation Conf.*, 1998, pp. 59-68.
[14] A. Mishchenko, B. Steinbach, and M. Perkowski, "An algorithm for bi-decomposition of logic functions," *Proc. of ACM/IEEE Design Automation Conf.*, 2001, pp. 282-285.
[15] Zvi Kohavi, *Switching and Finite Automata Theory*, McGraw Hill, 1999.
[16] Srinivas Devadas, Abhijit Ghosh, and Kurt Kuetzer, *Logic Synthesis* McGraw-Hill series on Computer Engineering, 1994.
[17] P.C. McGeer, J.V. Sanghavi, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "ESPRESSO-SIGNATURE: a new exact minimizer for logic functions," *IEEE Transactions on VLSI Systems*, vol. 1(4), December 1993, pp. 432-440.
[18] S.P. Tomaszewski, I.U. Celik, and G.E. Antoniou, "www based Boolean function minimization," *International Journal of Applied Mathematics and Computer Science*, vol. 13(4), 2003, pp. 577-583.
[19] Available: http://www.xilinx.com/support/mysupport.htm#Spartan-3

APPENDIX

TABLE V
LOGIC FUNCTION SPECIFICATION

Function ID	Minimized two-level logic obtained using ESPRESSO [17]
$Z1^{5}$	agb+agc+abf+fc
$Z2^{7}$	afe+afd+afg+aec+cd+cg+abe+bd+bg
$Z3^{9}$	dbfcg+dbfgh+dbfi+dceg+eh+ei+adcg+ah+ai
$Z4^{11}$	mnqst+mnqu+mnqv+mnqw+msto+uo+vo+wo+mstp+up+vp+wp+mstr+ur+vr+wr
$Z5^{8}$	ijn+ijo+ijp+kin+ko+kp+inl+ol+pl+min+mo+mp
$Z6^{9}$	pqrsuvw+pqrsx+puvwt+xt
$Z7^{7}$	mnpq+mnpr+mnqo+ro
$Z8^{8}$	qrwx+qru+qrv+qwxs+su+sv+qwxt+tu+tv
$Z9^{6}$	bgc+bgd+bge+abc+ad+ae
$Z10^{8}$	abe+abf+abg+abh+aec+fc+gc+hc+aed+fd+gd+hd
$Z11^{5}$	stw+s'vu+uw
$Z12^{7}$	tru+trv+t'ws+us+vs+t'wq+qu+qv (FOR ILLUSTRATION)
$Z13^{6}$	pmr+p'qn+nr+p'qo+or
$Z14^{9}$	abch+abci+a'fgh+dh+di+a'fge+eh+ei
$Z15^{8}$	pmx+pmy+qp'v+qx+qy+p'vw+wx+wy+p'vu+ux+uy
$Z16^{6}$	mnr+m'pqo+or
$Z17^{10}$	mnv+mnw+mnx+m'uq+vq+wq+xq+m'ur+vr+wr+xr+m'us+vs+ws+xs+m'ut+vt+wt+xt
$Z18^{9}$	abci+a'ghd+di+a'ghe+ei+a'ghf+fi
$Z19^{9}$	wxn+wxo+wxp+wxq+w'my+ny+oy+py+qy+w'mz+nz+oz+pz+qz
$Z20^{8}$	pqrsm+pqrsn+pqrso+pqrsp+p'xyzt+mt+nt+ot+pt+p'xyzu+mu+nu+ou+pu+ p'xyzv+ mv+nv+ov+pv+p'xyzw+mw+nw+ow+pw
$Z21^{10}$	defgl+defgk+d'onmh+lh+kh+d'onmi+li+ki
$Z22^{10}$	ijp+ijq+ijr+ijs+i'ok+pk+qk+rk+sk+i'ol+pl+ql+rl+sl+i'om+pm+qm+rm+sm+ i'on+pn+qn+rn+sn
$Z23^{11}$	cdefgn+cdefgo+c'jklmh+nh+oh+c'jklmi+ni+oi
$Z24^{12}$	a'be'f+a'bg+a'bh+ce'f+cg+ch+de'f+dg+dh
$Z25^{16}$	i'jkn'op+i'jkq+i'jkr+n'opl+ql+rl+n'opm+ qm+rm
$Z26^{12}$	p'qrv'wx+p'qry+p'qrz+v'wxs+sy+sz+v'wxt+ty+tz+uv'wx+uy+uz
$Z27^{10}$	a'bcdg'hij+a'bcdk+a'bcdl+g'hije+ke+le+g'hijf+kf+lf
$Z28^{8}$	r'sx'y+r'sz+r'sm+r'sn+r'so+x'yt+zt+mt+nt+ot+x'yu+zu+mu+nu+ou+x'yv+ zv+mv+nv+ov+x'yw+zw+mw+nw+ow
$Z29^{11}$	c'defgjklmn+c'defgo+c'defgp+j'klmnh+ho+hp+j'klmni+io+ip
$Z30^{10}$	p'qrsx'yzm+p'qrsn+p'qrso+p'qrsk+p'qrsl+x'yzmt+nt+ot+kt+lt+x'yzmu+nu+ ou+ku+lu+x'yzmv+nv+ov+kv+lv+x'yzmw+nw+ow+kw+lw
$Z31^{10}$	a'bcf'gh+a'bci+a'bcj+f'ghd+id+jd+f'ghe+ie+je
$Z32^{15}$	g'hk'l+g'hm+g'hn+k'li+mi+ni+k'lj+mj+nj
$Z33^{11}$	o'pqu'vw+o'pqx+o'pqy+u'vwr+rx+ry+u'vws+xs+ys+u'wvt+xt+yt
$Z34^{11}$	e'fj'k+e'fl+e'fm+e'fn+j'kg+lg+mg+ng+j'kh+lh+mh+nh+j'ki+li+mi+ni
$Z35^{14}$	q'rsv'wx+q'rsy+q'rsz+v'wxt+yt+zt+v'wxu+uy+uz

ZX^{n}; X – Function ID, n – Number of primary inputs