Athira P V, Ramesh S R / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp. 324-330

An Approach towards Logic Synthesis by Functional Decomposition

Athira P V*, Ramesh S R**
*(PG Student, Department of Electronics and Communication Engineering, Amrita Vishwa Vidyapeetham University, Coimbatore)
** (Assistant Professor, Department of Electronics and Communication Engineering, Amrita Vishwa Vidyapeetham University, Coimbatore)

ABSTRACT
This paper surveys some of the basic principles behind logic synthesis. A few methods of logic synthesis are also discussed. Functional decomposition is an efficient technique for synthesis of logic circuits targeted on Look Up Table based FPGAs. It decomposes any circuit into a network of sub circuits. A method of functional decomposition for single output XOR based circuits is presented. It utilizes Gauss Jordan elimination, a method based on linearity, to decompose the circuits. The method was tested on a set of MCNC benchmark circuits in Blif format, and was successful in decomposing circuits efficiently. In case of XOR based circuits, the XOR relationship between the different sub circuits can be exposed by this method. A reduction in area was obtained due to this in case of large XOR based circuits and hence can be used for area driven logic synthesis.

Keywords – **Decomposition chart, Field Programmable Gate Arrays, functional Decomposition, Gauss Jordan elimination, logic synthesis.**

1. Introduction
Field Programmable Gate Arrays (FPGAs) are programmable logic devices capable of implementing any logic circuit. When compared to the Application Specific Integrated Circuits (ASIC), the FPGAs offer high flexibility and low cost but have high power consumption. Despite this drawback, they are heavily used in low and medium volume applications due to their generality. FPGAs are available in different architectures out of which the Look Up Table (LUT) based FPGAs are the most popular and widely used due to their flexibility. The basic element in an LUT based FPGA is a K-input LUT (K-LUT). A K input LUT is an SRAM cell which can implement any logic function of K variables or less. It consists of 2^K memory cells and a K input multiplexer. The typical values of K are 4, 5 or 6. Fig. 1 shows a 2 input LUT. It takes two inputs: *x1* and *x2* and returns the output bit F depending on the correct combination of inputs. A K input LUT is capable of implementing 2^n different functions where $n = 2^K$.

Many approaches have been adapted to the synthesis of logic circuits for LUT based FPGAs. Synthesis is a step prior to implementation and it helps in modifying the circuit netlist into an equivalent netlist that can be implemented on the available FPGA architecture while optimizing parameters such as area, power, delay etc. Synthesizing a logic circuit involves two steps [1]: logic optimization which is technology independent and the technology mapping. Logic optimization involves changing the circuit structure by methods such as network simplification and node decomposition whereas technology mapping involves mapping a gate level netlist into a netlist of standard cells available in the library. For LUT based FPGAs, the given netlist is mapped into a network of LUTs. This paper describes a logic optimization technique for synthesis. It is based on functional decomposition, one of the node decomposition techniques.

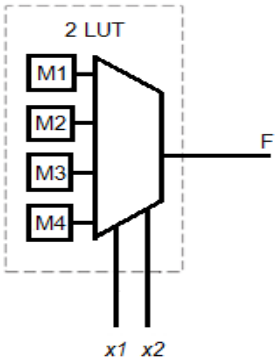


Fig. 1 A two input LUT.

Synthesis is usually addressed using different methods such as AND/OR based decomposition, XOR based decomposition, multiplexer based decomposition etc. Each of these types of decomposition has their own properties that make them effectively decompose circuits that consist of that type of logic. The XOR based circuits are widely used in arithmetic, error correction and communication based circuits. But the decomposition of these circuits is difficult and time consuming because there are seven different classes of XOR logic functions [2] each having its own properties. In this paper we present an XOR based functional decomposition technique that can decompose any XOR based function. The distinct advantage of this method is that it can be used in area and delay driven FPGA architectures.

In this paper, a comprehensive description of the synthesis process is given as follows: Section 2 contains the background required to understand the paper. Section 3 describes some of the existing methods of decomposition. Section 4 discusses the motivation towards the work. In section 5, the methodology used in the presented work is described. In section 6, the area and depth results obtained

from this work are presented. The obtained results are also compared with those from other tools like ABC and BDS-PGA 2.0. The paper is concluded and the topics of future work are discussed in Section 7.

2. Background

In this section, the basics of logic synthesis will be discussed. Some of the terminologies required to understand this paper will also be discussed.

2.1 Logic Synthesis

It is the process of transforming a gate level netlist of a multilevel circuit into a netlist of LUTs in which each LUT has at most K inputs. It involves two major steps: logic optimization and technology mapping.

2.1.1 Logic Optimization

Logic optimization transforms a network of logic gates into another set of logic gates in such a way that the final netlist is more optimized and suitable for mapping. The most important criterion while optimizing the gate level netlist is that every node should have a K feasible cone. A cone for any node (gate) is a combination of that node and some of its predecessors with the criterion that all the predecessors should have a path from it to that node. The main goals of logic optimization are to reduce the number of gates, reduce the logic depth, reduce the gate complexity or reduce the number of interconnections. But for an LUT based FPGA, reducing the gate count will not produce an optimized circuit but rather the objective is to reduce the number of literals that are present at the input. Some of the methods commonly used for logic optimization are

1. Combine a set of gates into a single gate
2. Duplicate a gate and redistribute its outputs
3. Decompose a gate into set of gates
4. Add a wire
5. Delete a wire
6. Delete unconnected gates

These methods can be broadly classified into two: network simplification and node decomposition.

In network simplification, the one or more nodes are simplified and the corresponding interconnections are also modified. The resulting network should be simpler with less number of gates and less dense interconnections. These results produce a reduction in the total area occupied by the circuitry. Simplification can be based on support reduction or by don't care simplification. Also there can be local simplifications or more efficient global simplifications with no range limit. BDDs and Karnaugh maps are tools that aid such simplification.

In node decomposition, the sub functions that represent a particular network/function are extracted. These new sub functions when combined together will form the original function. In effect the functionality should remain unchanged for the entire network. For LUT based synthesis, decomposition is an efficient method since the original circuit may not be mappable. If a particular node has more than K inputs then it cannot be mapped into a single K input LUT. So it has to be decomposed into two or more nodes so that each of them is K feasible. Thus decomposition can result in a mapping that is independent of the complexity of the function being implemented.

Node decomposition techniques can be divided into three depending on their optimization objectives: structural decompositions that are applied to simple gates or certain simple-gate networks, symbolic decompositions that are applied to complex gates based on symbolic operations on a given form of functional representation, and Boolean decompositions.

Boolean decomposition is a generalized technique that exploits the full functionality of the circuit. Here a Boolean equivalence exists between the original and the decomposed function. Some methods of Boolean decomposition are co factoring and functional decomposition. Co factoring involves decomposing the circuit in terms of Shannon's expansions, Davio expansions etc.

Functional decomposition [3] is an efficient method of expressing a function of n variables as a function of functions of fewer variables so that the functionality of the original network remains unchanged. For example, function *F(X)* can be decomposed into functions *G* and *H*, such that *F(X) = G(H(A), B)* where *X* is the set of inputs and *A* and *B* are proper subsets of *X*, such that $X = A \cup B, A \cap B = \emptyset$. Such a decomposition is called disjoint decomposition. If $A \cap B \neq \emptyset$, then it is called a non-disjoint decomposition. The whole of this paper deals with disjoint functional decomposition. Fig. 2 shows an example for functional decomposition where *F(X) = G(H(A), B)* and $A \cup B = X$, the set of inputs. The set of variables in *B = {S, T}* are called the free variables and the set of variables in *A = {P, Q, R}* are called the bound variables.

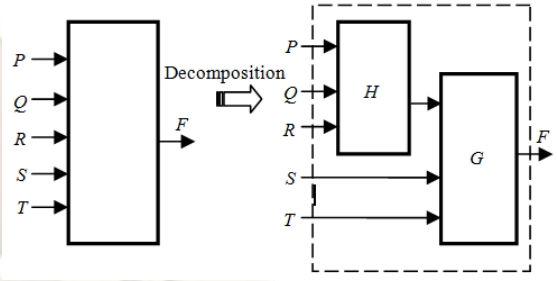
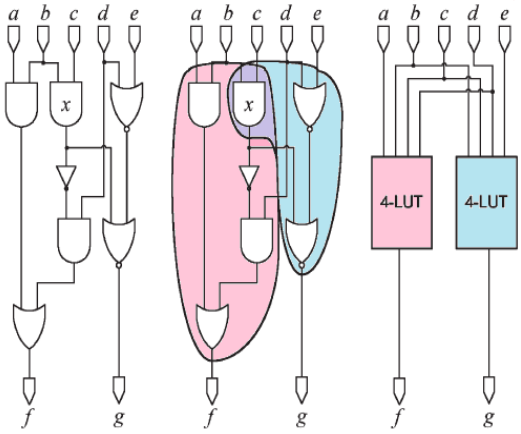


Fig. 2 An example for functional decomposition

2.1.2 Technology Mapping

Technology mapping is the process of mapping a gate level netlist into a netlist of standard cells available in the library. In case of Look Up Table based FPGAs the library contains Look Up Tables. Mapping is a technology dependent process and is usually preceeded by logic optimization. The technology mapping step can be considered as a cone selection problem or as a node covering problem. For an LUT based FPGA with K input LUTs available, the cones selected should be K feasible.



(a) (b) (c)

Fig. 3. Technology mapping as a covering problem
(a) Initial netlist. (b) Possible covering. (c) LUT mapping.

Fig. 3 illustrates the process of mapping a two output circuit into a set of 4 LUTs. Fig. 3(a) shows the initial gate-level netlist, Fig. 3 (b) shows a possible covering of the initial netlist using 4 LUTs and Fig. 3(c) shows the LUT netlist produced by the covering. In the mapping given, the gate labeled *x* is said to be duplicated since it is covered by both LUTs. Looking at technology mapping as a cone selection problem, the subcircuits circled in Fig. 3(b) are examples of cones. Technology mapping attempts to find the best set of cones that can be mapped to the current LUT architecture. "Best" is in terms of the optimizing goals such as area, speed, or power. Any cone with K-inputs or less can be implemented in a K LUT and is K-feasible. Therefore, to technology map circuits to K LUTs the circuit simply has to be decomposed into a set of K feasible cones followed by the direct assignment of a cone to an LUT.

2.2 Decomposition Chart

A	*B*	*C*	*F*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a)

AB \ *C*	0	1
00	0	0
01	1	0
10	1	0
11	0	1

(b)

Fig. 4. (a) Truth table of function *F*
(b) Decomposition chart of *F*

Decomposition chart is a form of representation of the truth table of any circuit. It is similar to the Karnaugh map except that the row and column indexes of the chart are in the binary order while that of the Karnaugh map are in the Gray code order. Fig. 4 shows the truth table of a three variable function *F(A,B,C)* and the corresponding decomposition chart. The set of variables used to index the columns of the decomposition chart form the bound set variables and the set of variables used to index the rows form the free set variables. In the example given in Fig. 4, {*C*} is the bound set and {*A*, *B*} is the free set.

3. Previous Works

There exists a wealth of research on XOR based decomposition and synthesis. One of the classical approaches is the Ashenhurst Curtis decomposition [4], [5] that makes use of the column multiplicity of the decomposition chart of any network to identify the number of unique columns in the chart. Column multiplicity is the number of distinct columns in the chart. Each unique column represents a compatible class and forms a min term in the decomposed circuit. Fewer the number of distinct columns, fewer is the number of min terms and lesser will be the area occupied by the circuit. Ref. [6] explains a method of encoding the compatible classes to improve the decomposability by extracting common sub expressions for multiple output functions. This is an efficient method of decomposition but does not address XOR based circuits.

Some of the XOR based decompositions were approached by positive Davio Expansions, negative Davio Expansions and Shannon Expansions. Based on these expansions the logic function can be represented in Reed Muller form, Kronecker form [7] and Exclusive OR Sum of Product form. But these methods decompose only one variable at a time, making the technique slow. Also the XOR relationship between non XOR functions cannot be exposed.

In BDD based methods, XOR decomposition is performed by finding x dominators in BDDs [8] and by BDD partitioning. An example for such an approach is given in [9]. This is similar to the concept of column multiplicity and is insufficient for XOR decomposition as it does not consider the XOR relation between the distinct columns. Also they do not consider logic simplifications with don't care sets.

Roth-Karp decomposition [10] requires λ minterms of compatible classes to be encoded with the same code. A modified version of the same is presented in [11] where an encoding algorithm is used to minimize the support of the decomposition functions.

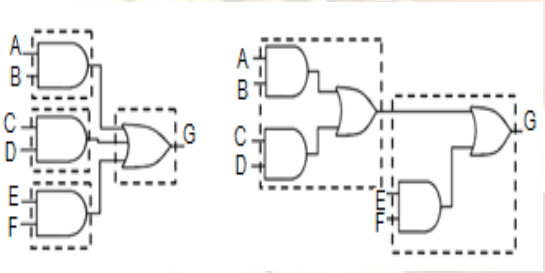
The approach used in this work retains the advantages of the above methods: it has the ability to decompose networks using Shannon's Expansions and Davio Expansions by using only one variable in the free set. Thus they turn out to be the special cases of the decomposition method used here. Also it is able to expose relationships between the columns of a decomposition chart without using the concept of column multiplicity and x dominators.

4. Motivation

This section describes how the functional decomposition will result in area optimization of the circuit. As far as an LUT based FPGA is considered, the area occupied by a circuit on the FPGA is given in terms of the number of LUTs to which the circuit is mapped. Consider an FPGA architecture that consists of K-LUTs. Each LUT consists of 2^K memory elements and is capable of implementing any logic function of K or fewer inputs. While implementing a network of n inputs on an FPGA, there can be three possibilities:

1. n < K: If the number of inputs is less than the value of K, then the circuit can be implemented on a single LUT. Not all memory locations of that LUT will be occupied.
2. n = K: If the number of inputs is equal to the value of K, then the circuit can be implemented on a single LUT by occupying all the memory bits of the LUT.
3. n > K: If the number of inputs is greater than the value of K, then the circuit cannot be implemented on a single LUT. If an n input LUT was available, then a single LUT was sufficient. In case of K-LUTs, a minimum of $2^n/2^K$ LUTs are required to map the circuit onto the FPGA. If the value of n is very large (say, n = 256) and K = 6, then the number of LUTs required = $2^{256}/2^6 = 2^{250}$. An FPGA has limited number of LUTs. So such a circuit with large number of inputs cannot be implemented on an FPGA.

From the above example it can be concluded that reduction in number of literals (input variables) [12] is required to minimize the area occupied by the circuit on an LUT based FPGA, rather than reducing the gate count. The following example shows how functional decomposition helps in reducing the literal count. Consider a network with 6 inputs and a single output, G. Fig. 5(a) shows the original network and Fig. 5(b) shows the network functionally decomposed [13], [14]. The corresponding reduction in number of LUTs can also be seen. If 4-LUT architecture is used, four 4-LUTs are required to implement the network without decomposition, and none of these LUTs are fully utilized whereas in the decomposed circuit, only two LUTs are required one of which is fully utilized. Thus using functional decomposition the LUT count is reduced.



(a) (b)

Fig. 5. (a) Without decomposition, four 4 LUTs
(b) With decomposition, two 4 LUTs

For XOR intensive circuits, area minimization is achieved in a different manner. Here the number of min terms (product terms) is reduced by using the AND/XOR implementation rather than going for the conventional AND/OR implementation. Consider a function, f which is dependent on four variables. The AND/OR implementation consists of 4 min terms and is given by

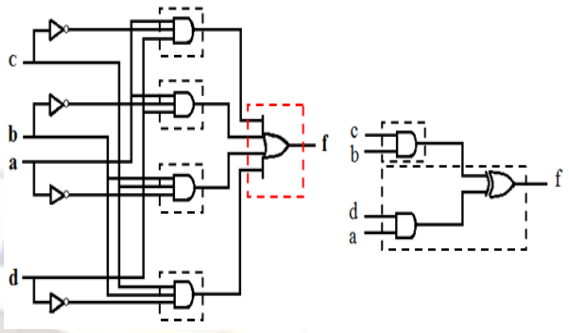
$$f = a\bar{c}d + a\bar{b}d + \bar{a}bc + bc\bar{d} \quad (1)$$

If 3-LUTs are used, then each min term will be mapped to one LUT. This requires four LUTs in total for all the min terms. Also the four input OR gate cannot be implemented in a single LUT and has to be decomposed. The decomposed circuit is shown in Fig. 6(a).

The corresponding XOR implementation is given by

$$f = bc \oplus ad \quad (2)$$

There are only two min terms with each min term having 2 inputs. One min term can be mapped into one LUT and the other min term along with the XOR gate can be mapped into the second LUT as shown in Fig. 6(b). The total number of LUTs required is 2 whereas it is 6 in the former case.



(a) (b)

Fig. 6. Function f implemented using
(a) AND/OR logic
(b) AND/XOR logic

The speed of operation is another important factor for any circuit. The interconnection length between the primary input and primary output determines the propagation delay and hence the speed. For an LUT based FPGA, speed is determined by the number of LUTs along the critical path and is termed as logic circuit depth. By reducing the logic depth, the speed increases.

5. Methodology

The method employed here is based on the property of linearity [15]. Any Boolean network can be decomposed by functional linearity which is described as

$$f(X) = \sum_i G_i(Y)H_i(X\text{-}Y) \quad (3)$$

where X is the set of input variables and $Y \subseteq X$. In equation (3), the function f is represented as a weighted sum of functions G_i, called the basis functions. The weighting factors are defined by functions H_i, called the selector functions. Here the summation represents an XOR operation. This definition exposes the XOR relationship between different logic functions.

The hardware required for implementation can be reduced by deriving the logic sub functions that can be reused in a logic expression. To find out such sub functions a technique in linear algebra is used. All the operations are done in a Galois Field, GF(2) where there are only two symbols 1 and 0. In this field, addition is represented using XOR operation and multiplication using AND operation.

Consider a k input logic function represented using a truth table of 2^m rows and 2^n columns (m + n = k). The truth table is equivalent to a two dimensional matrix of 2^m rows and 2^n columns. Viewing the matrix as a set of columns, a set of independent columns called basis have to be found out such that all other columns can be represented using these columns. For this the method of Gaussian elimination [16] is used. The Gaussian elimination converts the matrix into a

row echelon form by applying a series of elementary row transformations. To illustrate the method consider a 4 variable function *f* whose decomposition chart is given in Fig. 7. Initially the rows of the matrix are swapped such that the rows are ordered from top to bottom depending on the column index of the respective leading one entries. A leading one is the first occurrence of 1 in a row of the truth table. Thus if any two successive rows do not consist entirely of zeros, the leading 1 in the lower row occurs farther to the right than the leading 1 in the higher row. Next if a column contains a leading 1 then all the occurrences of 1 below that leading one is made a 0 by XORing the two columns.

ab \ *cd*	00	01	10	11
00	0	0	0	0
01	0	0	1	1
10	0	1	0	1
11	0	1	1	0

Fig. 7 Decomposition chart for the function *f* in the example

For the decomposition chart in Fig. 7 the set of operations to be performed are given in Fig. 8. Initially the rows are arranged depending on the column index of their leading one entries. Now row 1 has a leading one entry at position 2. All the other rows are checked for a 1 entry in the same position as that of the leading one. Since there is a 1 in the corresponding position in row 2 also, row 1 and row 2 are XORed and row 2 is replaced with this value. Since there are no one's in rows 3 and 4 below the leading one, we swap the rows again depending on the column index of leading 1 entries. The next occurrence of leading one is now checked and the process is continued. Finally rows 3 and 4 are entirely sets of zeros. This completes the Gaussian elimination and the matrix is reduced to row echelon form. Now the columns which contain leading one are selected. The corresponding columns in the original matrix correspond to the basis vectors. The basis vectors are
G_1 = [0 0 1 1] and G_2 = [0 1 0 1].
The corresponding basis functions are
$G_1 = a$, and $G_2 = b$.

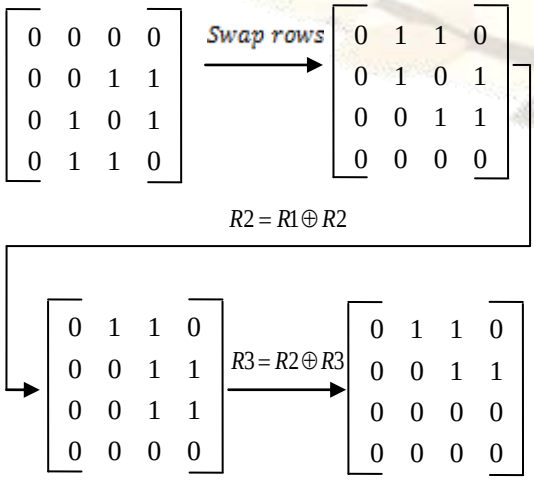


Fig. 8. Gaussian elimination applied to the example in Fig.7.

To obtain the selector, Gauss Jordan elimination is applied. This is an extension to the Gaussian elimination method and it converts the matrix to reduced row echelon form. In this form, the columns containing leading ones should have only one nonzero entry. For finding the reduced row echelon form, the row echelon form is considered and each column is checked for the presence of a leading one. If it is found then all the 1's above that leading 1 (if there exists any) is made 0 by XORing the corresponding rows and replacing the latter with the same. For the example in Fig. 7 the reduced row echelon form can be obtained by replacing row 1 by the XOR of row 1 and row 2 as shown in Fig. 9. The rows of the reduced row echelon matrix that contains leading ones correspond to the selector vectors. The selector vectors are
H_1 = [0 1 0 1] and H_2 = [0 0 1 1]
The corresponding selector functions are
$H_1 = d$ and $H_2 = c$
The circuit is thus decomposed as:

$$f = ad \oplus bc \quad (4)$$

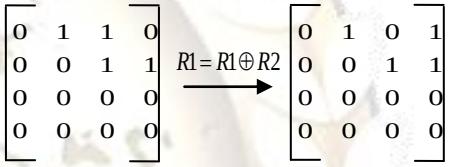


Fig. 9. Gauss Jordan elimination applied to the example in Fig. 7.

Mapping this circuit into a set of LUTs will require smaller number of LUTs when compared with the AND/OR implementation as explained in Section 4. Thus this method will efficiently decompose the given network and produce a netlist optimized in both area and speed.

6. Experimental Results

This section contains a discussion on the results obtained. The methodology used in the work, comparison of results with those obtained from other state of the art tools and discussion on individual circuits will be presented.

The efficiency of the technique was tested on a set of MCNC circuits in BLIF format [17]. The benchmark circuits were read and the truth table for each network was extracted from it. The basis and selector vectors were found out by implementing the algorithms for Gaussian elimination and Gauss Jordan elimination. The decomposed network corresponding to these vectors were obtained by running the *read_dsd* command in the mapping tool, ABC [18],[19]. The LUT mapping was done using the ABC's *if –K 4* command. The area results obtained by doing so are given in column 3 of Table 1. Area is given in terms of the number of LUTs consumed by each circuit. The delay results are given in column 3 of Table 2. Delay is given in terms of the logic depth on the critical path i.e., the number of LUTs on the longest path from input to output.

To find the results with ABC, the original circuit was strashed using ABC and then mapped into 4-LUTs. The area and delay results obtained from ABC are given in column 4

of Table 1 and Table 2 respectively. Column 2 represents the number of inputs for each circuit.

Table 1. Area results obtained from various tools

Name	Inputs	Linearity Approach	ABC	BDS-PGA2.0
9sym	9	27	157	58
9symml	9	13	78	19
t481	16	314	355	524
xor5	5	3	2	2
majority	5	4	3	3
parity	16	5	5	5
cm152a	11	10	10	10

To find the results with BDS-PGA 2.0 [20], the command *bds –options <circuit.blif>* is run on BDS-PGA 2.0. The decomposed network is mapped using ABC technology mapper. The options can be sharing, xhardcore or heuristic. Column 5 gives the results obtained from BDS-PGA 2.0.

Table 2. Delay results obtained from various tools

Name	Inputs	Linearity Approach	ABC	BDS-PGA2.0
9sym	9	4	5	6
9symml	9	4	6	6
t481	16	7	7	6
xor5	5	2	2	2
majority	5	2	2	3
parity	16	2	2	2
cm152a	11	3	3	3

6.1 Discussion of Individual Circuits

As can be seen in Table 1, area savings were not obtained for circuits like *xor5* and *majority*. *xor5* is a circuit with 5 inputs and a single output. The output is the XOR of all the 5 inputs. The decomposition chart for the circuit is given in Fig. 10.

The reduced row echelon form for the decomposition chart is given in Fig. 11. There are only two basis and selector pairs for this circuit. The synthesized circuit corresponding to this decomposition is as shown in Fig. 12. Each dashed box show the functionality realized by an LUT. The circuit requires 3 LUTs when mapped into a set of 4-LUTs. The logic depth is 2. When compared with ABC and BDS-PGA 2.0, FLDS produces an increase of 1 in the number of LUTs. This can be reduced to 2 by further optimizing the basis and selector pairs obtained. The actual advantage of the method can be known more evidently in case of large circuits like *t481*.

ab \ cd	000	001	010	011	100	101	110	111
00	0	1	1	0	1	0	0	1
01	1	0	0	1	0	1	1	0
10	1	0	0	1	0	1	1	0
11	0	1	1	0	1	0	0	1

Fig. 10 Decomposition chart for *xor5*

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 11 Reduced row echelon form for the decomposition chart of *xor5*

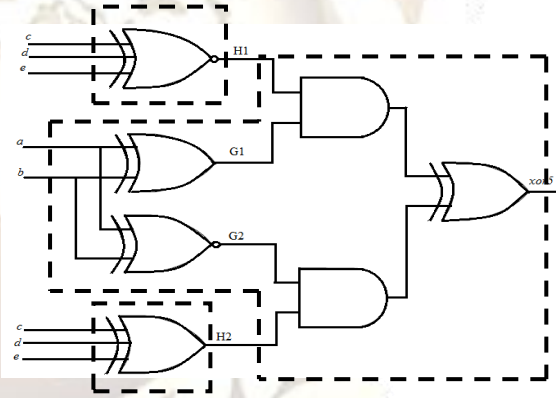


Fig. 12 Decomposed circuit for *xor5*

7. Conclusion

A survey of literature on decomposition and synthesis of logic circuits is presented. A logic synthesis approach for LUT based FPGAs based on functional decomposition has been discussed in this paper. The method is based on linear algebra and requires only the truth table of any network as its input. XOR relationship between the different logic sub functions can be exposed by this method and hence is efficient in decomposing XOR based circuits in terms of area and delay. Further the method can be extended to decompose multioutput networks by putting the truth tables of each output side by side.

References

[1] J. Cong, and Y. Ding, Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays, *ACM Trans. Des. Autom. Electron. Systems, 1 (2)*, 1996, 145-204.

[2] T. Sasao, *Switching Theory for Logic Synthesis* (Norwell, MA: Kluwer, 1999).

[3] C. Scholl, *Functional Decomposition with Applications to FPGA Synthesis*. (Boston: Kluwer, 2001).

[4] R. L. Ashenhurst, The decomposition of switching functions, *Proc. Int. Symp. Theory Switch*ing, 1957, 74-116.

[5] M. Perkowski, and S. Grygiel, *A survey of literature in Functional Decomposition*, A Final Report for Summer Faculty Research Program, Wright Laboratory, Washington, DC, 1994.

[6] J. -H Jiang, J. -Y. Jou, and J. -D. Huang, Unified functional decomposition via encoding for FPGA Technology Mapping, *IEEE Trans. VLSI Syst.*, 9 (2), 2001, 251-260.

[7] T. Sasao, and J.T. Butler, A design method for look-up table type FPGA by pseudo-Kronecker expansion, *Proc. 24th Int. Symp. Multi-Valued Logic*, 1994, 97-106.

[8] R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Comput., C-35 (8)*, 1986, 677–691.

[9] N. Vemuri, P. Kalla, and R,Tessier, BDD-based logic synthesis for LUT-based FPGAs, *ACM Trans. Des. Autom. Electron. Devices, 7 (4)*, 2002, 501–525.

[10] J. P. Roth and R. M. Karp, Minimization over Boolean graphs, *IBM J.*, 1962, 227–238.

[11] J.-D. Huang, J.-Y. Jou, and W.-Z. Shen, Compatible class encoding in Roth–Karp decomposition for two-output LUT architecture, *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1995, 359–363.

[12] H. Sawada, T. Suyama, and A. Nagoya, Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization, *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1995, 353-358.

[13] A. Sangiovanni-Vincentelli, A. El Gammal, and J. Rose, Synthesis Methods for Field Programmable Gate Arrays, *Proc. IEEE, 81 (7)*, 1996, 1057–1083.

[14] S.D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field- Programmable gate Arrays.* (Boston: Kluwer, 1992).

[15] T.S. Czajkowski, and S. D. Brown, Functionally linear decomposition and synthesis of logic circuits for FPGAs, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., 27 (12)*, 2008, 2236–2249.

[16] H. Anton, and C. Rorres, *Elementary Linear Algebra, Applications Version*, ninth ed., (Hoboken, NJ: John Wiley, 1994).

[17] S. Yang, *Logic synthesis and optimization benchmarks User Guide 3.0*, Microelectron. Center North Carolina, North Carolina, 1991.

[18] R. Brayton, and A. Miskchenko, ABC: An academic industrial-strength verification tool, *Proc. CAV 2010*, vol. 6174, 2010, 24-40.

[19] Berkeley Logic Synthesis Group, *ABC: A System for Sequential Synthesis and Verification*, Feb. 2010 [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc.

[20] BDS-pga version 2.0, *BDD-based Logic Synthesis System for LUT-based FPGAs* [Online]. Available: http://www.ecs.umass.edu/ece /tessier/rcg/bds-pga-2.0/