Using Genetic Algorithm in Binate Covering for LUT-based FPGA Technology Mapping

Nan Zhuang and Peter Y.K. Cheung[1]

Abstract

A Genetic Algorithm(GA) is introduced to the binate covering problem in LUT-based FPGA technology mapping. We formulated the problem of finding the area-optimal technology mapping as a problem of searching the quasi-optimal column selection ordering of the binate covering matrix using GA. A large set of LGSynth91 benchmarks are tested and compared to the *xl_cover* routine in SIS from UC Berkeley, and our package *GA_cover* is shown to be superior.

Indexing terms: FPGAs, Technology Mapping, Genetic Algorithm, Binate Covering

1 Introduction

Field-Programmable Gate Arrays(FPGAs) are increasingly popular in implementing Application Specific Integrated Circuits(ASICs) because designers can implement their ASIC quickly and cheaply. Many FPGA architectures have been proposed and the Look-Up-Table(LUT)-based architecture is most popular. Many tools for LUT-based FPGA technology mapping have been proposed[1-8]. Among them, SIS is a comprehensive synthesis tool developed by UC Berkeley[1]. FPGA technology mapping in SIS(called $mis-PGA$[2] consists of two main steps: decomposition and reduction. In the reduction step, the routines *xl_cover* and *xl_partition* are employed. However, it is well known that exact binate covering as handled by *xl_cover* is NP-Complete[2].

[1]The authors are with the Dept. of Electrical and Electronic Engineering, Imperial College London, UK, SW7 2AZ

Genetic Algorithm[9], which simulates the natural process of evolution, has been proposed as an efficient paradigm for solving hard optimization problems. It has been successfully employed for solving many problems in areas such as logic synthesis, VLSI placement, state assignment, BDD ordering, etc. [10-15]. In this paper, GA is employed in the binate covering problem of FPGA technology mapping.

2 Binary Covering Formulation

After decomposition, an m-feasible network is obtained, which can be implemented straightaway on the FPGAs by mapping each node to an CLB. This strategy, however, yields sub-optimal results.

For example, consider the following optimised network η, with one primary output f, eleven primary inputs, and four internal nodes:

$$\begin{aligned} f &= x + y + z \\ x &= degh \\ y &= abc \\ z &= ijkl \end{aligned}$$

Let m be 5. Now map η onto target LUT architecture. We need 4 CLBs to implement the network η. However, if y is collapsed into f, we found following result:

$$\begin{aligned} f &= x + abc + z \\ x &= degh \\ z &= ijkl \end{aligned}$$

which is 5-feasible and only 3 CLBs are needed. Covering routine is an important way to reduce the number of feasible nodes.

Definition 1. Covering operation can be stated as: given an m-feasible Boolean network η, iteratively collapse nodes into their fanout such that the resulting network η' is m-feasible and the number of nodes in η' is minimum.

To formulate the covering problem following Definitions should be introduced:

Definition 2. Given a graph $G = (V, E)$, and $U \in V$, the **induced subgraph** of G on U is (U, E_1), where $E_1 = \{(u_1, u_2) \mid u_1, u_2 \in U \ and \ (u_1, u_2) \in E\}$. In other words, (U, E_1) is G restricted to U.

Definition 3. A **supernode** corresponding to a node n of the network η is an induced directed subgraph ζ of η, with a single root n such that ζ does not contain any primary input node of η. A node may have more than one supernode. For example, the node n in Boolean network η shown in Fig. 1 has the following supernode associated with it: $\{n\}$, $\{n, j\}$, $\{n, k\}$, $\{n, j, k\}$, $\{n, j, i\}$, and $\{n, j, k, i\}$. Note that $\{n, i, k\}$ is not a supernode because it has two roots, n and i.

Definition 4 A supernode ζ is m-**feasible** if its support cardinality is at most m, i.e., $|\sigma(\zeta)| \leq m$. For example, the node n in Fig. 1 has following 5-feasible supernodes:

supernode	*support*
$\{n\}$	$\{j, k\}$
$\{n, j\}$	$\{i, d, k\}$
$\{n, k\}$	$\{d, e, g, j\}$
$\{n, j, k\}$	$\{i, d, e, g\}$
$\{n, j, i\}$	$\{a, b, c, d, k\}$

Note that supernode $\{n, j, k, i\}$ is not 5-feasible because its support has 6 menders, a, b, c, d, e, and g.

[16] proposed an algorithm to generate all supernodes corresponding to the every node of the network η by repeatedly applying the Maxflow algorithm[17]. If there are p nodes in the network(including primary inputs and primary outputs), the maximum number of m-feasible supernodes for one node is $C(p, m)$(which is the notation for the number of all possible ways of choosing m objects out of p). So, the total number of m-feasible supernodes is bounded by

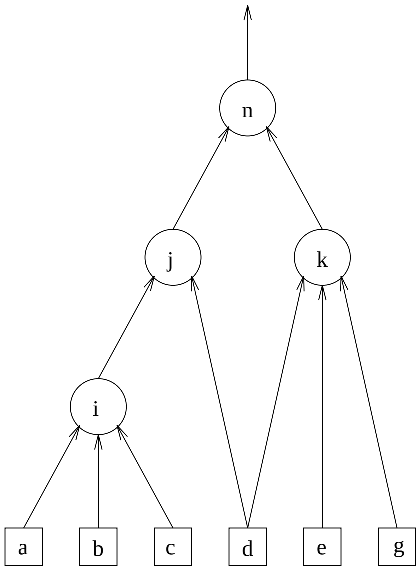

Figure 1: A Boolean network η.

$p \times C(p, m)$. Since m is a constant, this algorithm is quite fast even on large networks.

Having generated the set of all feasible supernodes, the **binate covering matrix M** is constructed as follows: there is a column for each supernode ζ_i, and the rows are basically for the nodes of the network(there are additional rows to take care of the binate covering constraints). $M(i, j) = 1$ if the node corresponding to row j is contained in the supernode ζ_i. Fig. 2 shows an example of a binate covering matrix. Since every supernode node can be implemented by one CLB, the task of the covering is to find a minimum subset Γ of supernodes that satisfied the following constraints via binate covering matrix:

- **output constraints**: for each primary output PO, at least one supernode with PO as the root must be chosen.

- **implication constraints**: if a supernode ζ is in Γ, each input of ζ should be either a primary input or an output of some supernode in Γ.

An exact algorithm can only be employed in small circuits (normally having less than 30 nodes)[2]. When a heuristic algorithm is employed for large circuits, the column containing the maximum number of 1's is greedy selected at each step, and the rows covered by this column are deleted (see Fig. 2). However, if two or more columns contain the same number of 1's, the first one or the last one is arbitrarily selected. For example, in Fig. 2, after selecting column 2, column 3, 5, 7 and 8 contain the same number of 1's. The covering routine in SIS will make an arbitrary selection (selects column 3 or column 8). For large circuits, there may be hundreds of columns contained the same number of 1's. Such arbitrary selection may not provide a good solution.

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

Figure 2: Binate covering matrix.

3 Genetic Algorithm for Binate Covering

3.1 Problem representation

To apply the Genetic Algorithm to the binate covering problem, we first generate N initial chromosomes, $S_1, S_2, ..., S_N$, each containing a set of random numbers, which range from 1 to M. Each chromosome S_i is applied to the binate covering column selection procedure using the following algorithm: (a) For the j^{th} selection step, find all k columns containing the maximum number of 1's (where k can be 1 or higher). (b) Relabel these columns as $C_0, C_1, ..., C_{k-1}$, and select the column C_v to provide covering, where $v = S_i[M \ mod \ j] \ mod \ k$. (c) Delete column C_v and all rows covered by the 1's in column C_v from the matrix M. (d) Repeat step (a) to (c) until all rows are covered. For example, if the ith chromosome, $S_i = (53312)$(here, m is 5), is used to control the column selection of the binate covering matrix shown in Fig. 2, the covering procedure would be as follows: (1) at the first selection step, we find a column(column 2) containing the maximum number of 1's(five 1's). We relabel column as C_0. $v = S_i[0] \ mod \ k = 5 \ mod \ 1 = 0$. So, C_0, i.e., column 2 is selected, and column 2 and all rows covered by the 1's in column 2 are deleted from the matrix. The updated matrix is as shown as Fig. 3(a). (2) from Fig. 3(a), we find columns 2, 4, 6 and 7(which are column 3, 5, 7, 8 in Fig. 2) are containing the maximum number of 1's. So, $k = 4$, and $v = S_i[1] \ mod \ k = 3 \ mod \ 4 = 3$. Column 7 is selected, and the new matrix is as shown as Fig. 3(b). Similar, from Fig. 3(b) matrix, column 4 is selected, and from Fig. 3(c) matrix column 3 is selected. The finally cover set is column $(2, 8, 7, 5, 11)$.

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

(a)

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

(b)

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

(c)

Figure 3: Column selection scheme.

3.2 Evaluation of the fitness

The fitnesses of all N solutions can be calculated from the size of the resultant network. However, if the N is 20, and the total evolution generations are 100, the covering routine would be running for 2000 times. It means that GA may spend 2000 times CPU time compared with heuristic algorithm in SIS! So, a quick evaluation strategy is necessary. Here, we proposed an algorithm which can predict the covering gain without change the Boolean network. The key procedure of this scheme is as follows: (a) get a subset Γ of supernodes via binate covering matrix according to the solution S_{best} obtained by GA; (b) Put supports of all supernodes in set Γ into a support set Ω; (c) check every internal node n, if n does not exist in the support set Ω, which means that node n has been covered by a supernode, and one CLB is saved($gain$++). For example, for LGSynth91 benchmark *z4ml* (a 7 inputs, and 4 outputs function), after technology independent optimisation and decomposition, we got 5-feasible network as follows:

$$\begin{aligned}
f_1 &= x_2x_5 + \{13\} \\
f_2 &= \{10\}f_1' + \{14\} + \{15\} \\
f_3 &= \{8\}\{9\} + \{8\}'\{9\}' \\
f_4 &= x_7\{12\} + x_7'\{12\}' \\
\{8\} &= x_1f_4' + x_4x_7 \\
\{9\} &= x_3x_6 + x_3'x_6' \\
\{10\} &= x_2 + x_5 \\
\{12\} &= x_1x_4 + x_1'x_4' \\
\{13\} &= x_3\{10\}f_3' + x_6\{10\}\{8\} \\
\{14\} &= x_2x_5f_3'\{8\} + f_1'f_3'\{8\} \\
\{15\} &= x_2x_3x_5x_6 + x_3x_6f_1'
\end{aligned}$$

11 CLBs are needed to implement above network. The Boolean network of above circuit is as shown as Fig. 4. There are total 46 5-feasible supernodes in the network. A *good* subset Γ of the supernodes, containing five supernodes, $\{f_1, \{13\}, \{10\}, f_3, \{9\}\}$, $\{f_2, \{10\}, \{14\}, \{15\}, \{13\}, f_3, \{9\}\}$, $\{f_3, \{9\}\}$, $\{f_4, \{12\}\}$, and $\{\{8\}\}$, which cover all nodes in the network, can be obtained using GA covering scheme. We found that the support set $\Omega = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, \{8\}\}$ by checking

above 5 selected supernodes. Internal nodes $\{9\},\{10\},\{12\},\{13\},\{14\}$ and $\{15\}$ are not exist in the support set Ω, which means that they have been covered by some other supernodes. So, the gain of the covering is 6, i.e., 6 CLBs are saved. The main advantage of above algorithm is that it is unnecessary to change the network will predict how many CLBs can be save, which will significantly speed up the evaluation function.

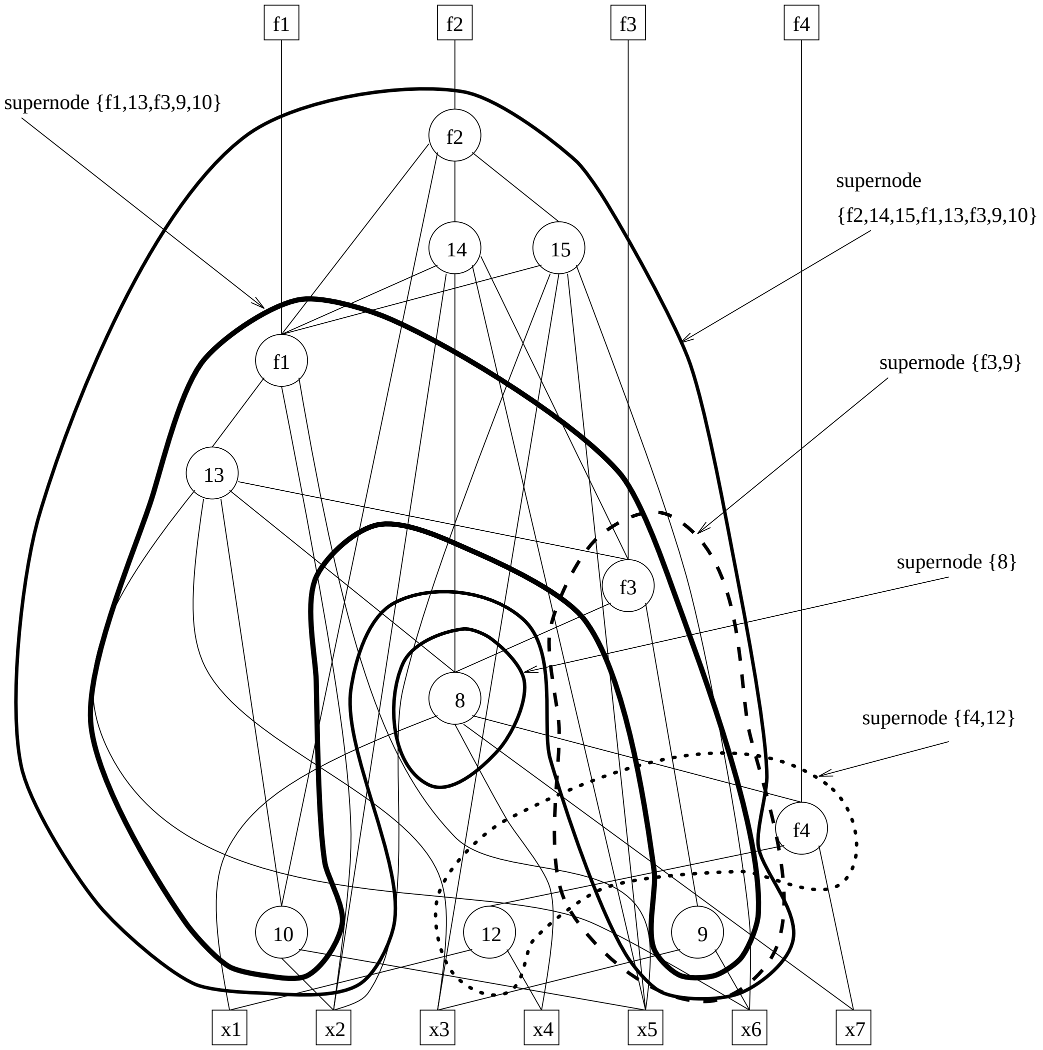

Figure 4: Covering a network η.

3.3 Other GA operators

The Roulette Wheel Parent Selection algorithm described in [13] is then applied. Although a gene here is expressed as a decimal as [13-15], the offspring chromosomes are always valid no matter what kind crossover scheme is employed. For an extremely example, when a chromosome is $(1, 1, ..., 1)$, the column selection algorithm will always select the first column when some columns contain the same number of 1's, which is what the heuristic algorithm in SIS does. So, the traditional one-point cut crossover scheme described in [13,14] is employed to produce the next generation of N solutions. We also tested two-point cut and multi-point cut crossover schemes. We found that one-point cut scheme is with the best result for this application. Mutation scheme here is the same as that in [14], i.e., randomly exchange two genes in a chromosome. To keep the CPU time reasonable for large circuits synthesis a low mutation rate of 0.5% or less is used to ensure fast convergence. Empirically a near optimal solution can be reached within 40 to 80 generations. The disadvantage of the lower mutation rate is that solutions are mature too fast. It may be trapped in a local optimisation. We can run the package for several times, and chose the best result, to avoid this problem.

4 Testing Results

The algorithm described above has been integrated into the SIS environment as the *GA_cover* routine which replaces the original *xl_cover* routine. To analyse the algorithm three sets of benchmarks from LGSynth91 are tested. Table 1 shows the testing results of benchmarks with more than 200 CLBs, Table 2 shows the results of benchmarks with CLBs between 30 and 200, and Table 3 shows the results of benchmarks with less than 30 CLBs. All results are compared with the *xl_cover* algorithm in SIS, and for single output LUTs that can implement any function of 5 inputs. The second column in the Tables is the number of CLBs after technology independent logic optimisation, decomposition, and partitioning using script: "$so\ -x\ script$; xl_ao; $xl\_partition\ -mt$. The third and seventh columns in the Tables show the number of CLBs after optimised by the original SIS algorithm, and our GA package with scripts $xl\_cover - e\ \ 30 - u\ \ 1000$", and GA_cover, respectively. The fourth and eighth columns show the gain obtained by *xl_cover* and *GA_cover*, respectively. CPU times are in seconds on a SUN SPARC 10 workstation.

Table 1 Test results for LGSynth91 benchmarks with CBL number larger than 200

Benchmarks	before cover	xl_cover	gain	CPU	GA_cover	gain	CPU
C5315	384	361	23	390.6	357	27	1427.9
C6288	482	482	0	338.3	475	7	1819.7
C7552	440	430	10	476.3	425	15	1669.7
alu4	215	212	3	425.5	206	9	737.5
dalu	360	359	1	320.4	340	20	1251.6
frg2	392	385	7	424.6	382	10	1235.3
i10	869	862	7	2969.8	859	10	5437.4
pair	388	379	9	388.6	376	12	1407.8
total	3530	3470	60	5734.1	3420	110	14986.9

Table 2 Results for LGSynth91 benchmarks with CLB number between 30 and 200

Benchmarks	before cover	xl_cover	gain	CPU	GA_cover	gain	CPU
C1908	70	70	0	28.3	70	0	137.5
C1908	100	98	2	33.7	98	2	175.7
apex7	62	61	1	14.6	61	1	91.3
f51m	30	27	3	13.3	26	4	84.2
i6	112	71	41	56.3	67	45	387.9
i7	140	107	33	98.5	103	37	584.7
my_adder	32	31	1	6.8	27	5	82.0
x4	116	114	2	46.4	114	2	125.9
total	662	579	83	297.9	566	96	1543.59

Table 3 Test results for LGSynth91 benchmarks with CLB number less than 30

Benchmarks	before cover	xl_cover	gain	CPU	GA_cover	gain	CPU
cc	19	16	3	3.0	17	2	12.8
cm162a	12	11	1	4.7	11	1	10.9
sct	17	17	0	5.4	17	0	20.7
tcon	8	8	0	0.8	8	0	3.3
x2	14	14	0	1.7	14	0	6.7
z4ml	8	5	3	1.6	5	3	17.7
total	78	71	7	17.2	72	6	72.1

Table 4 Test results for large LGSynth91 benchmarks

Benchmarks	before cover	xl_cover	GA_cover
C5315	472	396	378
C6288	933	855	791
C7552	600	481	471
alu4	220	218	209
dalu	386	356	350
des	1055	986	970
frg2	402	392	389
i10	926	880	862

For large circuits (more than 200 CLBs) our algorithm, *GA_cover*, got 83.8% more total gain compared with *xl_cover* in SIS with about 3 times total CPU time. The total gain improvement is reduced to 15.6%, and the CPU times consumption is about 5 times compared with *xl_cover* for middle size circuits shown in Table 2. For small circuits with the CLB number less than 30, *xl_cover* obtains better results in both CLB counter and CPU time consumption. From above results we found that: (a) *GA_cover* is suitable for large circuits; (b) Because a fast evaluation algorithm is employed in our GA, CPU time consumption is just 3 to 5 times compared with heuristic algorithm in SIS, but not thousands of times. (c) The absolute number of CLBs is not reduced significantly after execute both GA_cover and xl_cover because after technology independent logic optimisation, cube-packing decomposition, and iterative partitioning, most circuits are nearly optimised, and the room left for covering has been small.

In order to demonstrate the effectiveness and ability of our algorithm, as [8], suppose the partition routine is not used in the synthesis algorithm, i.e., the synthesis scripts are: $so - x\ script;\ xl\_ao,\ GA\_\ cover$, and $so - x\ script;\ xl\_ao;\ xl\_cover\ - e\ 30\ - u\ 1000$, respectively. The testing results for all LGSynth91 benchmarks with more than 200 CLBs are shown in Table 4. From Table 4, we can see that our algorithm produces much better results compared with SIS.

5 Conclusion

In this paper, GA is introduced to solve NP-complete problem, binate covering, in the FPGA technology mapping. To keep the CPU time reasonable, which decides that GA is acceptable or not, we employed two techniques to speed up the algorithm: (a) A fast evaluation algorithms are employed because in GAs almost 99% CPU time is consumed by evaluation function to calculate the fitness. In our algorithm fitness was predicted without change the network(by checking every internal node to see if it is existed in the support set Ω). (b) Combine GA and heuristic algorithm, called hybrid GA, to accelerate the convergence of the solution. We mixed the heuristic ideal into GA at the representation stage implicitly. From Section 2 we found that if a column contains more number of 1's than any other columns, $k = 1$, i.e., this column will be definitely selected. Actually, it is the heuristic selection employed in SIS. So, even for huge benchmarks, like $C6288$, des, $i10$, etc., our GA package can find the quasi-optimal results within 5,500 seconds on a single CPU workstation. Moreover, some novel GA operators, like dynamic parameters, partial mapping crossover, decimal gene representation, etc., are also proposed to improve the synthesis quality. These algorithms have been integrated in SIS environment, which provide a well interface to external physical design tools. The testing results show that GA produces better results for a reasonable increase in CPU times when compared with other heuristic algorithms. GA probably provides a way for solving NP-complete problems in the logic synthesis.

Acknowledgement: This project is supported by Mentor Graphics Corparation, USA.

References

[1] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, and R. K. Brayton, 'SIS: A system for sequential circuit synthesis', Technical report UCB/ERL M92/41, University of California, Berkeley, May 1992.

[2] R. Murgai, N. Nishizaki, N. Shenoy, R. K. Brayton, and Sangiovanni-Vincentelli, 'Logic synthesis for Programmable Gate Arrays', Proc. 27th ACM/IEEE Design Automation Conf., 1990, pp.620-625.

[3] R. Murgai, N. Shenoy, R. K. Brayton, and Sangiovanni-Vincentelli, 'Improved logic synthesis algorithm for Table Look Up architectures', Proc. IEEE Int Conf. Computer Aided Design, 1991, pp.564-567.

[4] R. J. Francis, J. Rose, K. and Chung, 'Chortle: A technology mapping program for Lookup Table-based Field Programmable Gate Arrays', Proc. 27th ACM/IEEE Design Automation Conf., 1990, pp.613-619.

[5] R. J. Francis, J. Rose, and Z. Vranesic, 'Chortle-crf: Fast technology mapping for Lookup Table-based FPGAs', Proc. 28th ACM/IEEE Design Automation Conf., 1991, pp.227-233.

[6] T. T. Hwang, R. M. Owens, M. J. Irwin, and K. H. Wang, 'Logic synthesis for Field-Programmable Gate Arrays', *IEEE Trans. on CAD*, 1994, **13**, (10), pp.1280-1286.

[7] J. Cong, and Y. Ding, 'FlowMap: An optimal technology mapping algorithm for delay optimization in Lookup Table based FPGA design', *IEEE Trans. on CAD*, 1994, **13**, (1), pp.1-12.

[8] W. Z. Shen, J. D. Huang, and S. M. Chao, 'Lambda set selection Roth-Karp decomposition for LUT-based FPGA technology mapping', Proc. 32nd ACM/IEEE Design Automation Conf., 1995, pp.65-69.

[9] D. E. Goldberg, 'Genetic algorithm in search, optimization, and machine learning', (Addison-Wesley Publishing Company. Inc., USA, 1989)

[10] B. Becker, and R. Drechsler, 'OFDD based on minimisation of fixed polarity Reed-Muller expressions using hybrid genetic algorithm', Proc. IEEE Int Conf. Computer Aided Design, 1994, pp.106-110.

[11] K. Shahookar, and P. Majumder, 'A genetic aproach to standard cell placement using meta-genetic parameter optimization', *IEEE Trans. CAD*, 1990, **9**, (5), pp.500–512.

[12] A.E.A. Almaini, J. F. Miller, P. Thomson, and S. Billina, 'State assignment of finite state machines using a genetic algorithm', *IEE Proc. E., Comput. and Digital Tech.*, 1995, **142**, (4), pp.279-286.

[13] A.E.A. Almaini, and N. Zhuang, 'Using genetic algorithm for the variable ordering of Reed-Muller binary decision diagrams', *Microelectronics Journal*, 1995, **26**, (5), pp.471-480.

[14] A.E.A. Almaini, N. Zhuang, and F. Bourset, 'Minimisation of multioutput Reed-Muller binary decision giagrams using hybrid genetic algorithm', *Electronics Letters*, 1995, **31**, (20), pp.1722-1723.

[15] N. Zhuang, M. Benten, and P. Cheung, "Improved variable ordering of BDDs with novel genetic algorithm", *Proc. of IEEE Int. Symposium on Circuits and Systems*, pp. III-414–417, May, 1996.

[16] R. Murgai, *Logic Synthesis for Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1995.

[17] E. L. Lawler, 'Combinational optimization: Networks and Matroids', (New York: Holt, Rinehart and Winston, 1976)