MemMap: Technology Mapping Algorithm for Area Reduction in FPGAs with Embedded Memory Arrays using Reconvergence Analysis

Manoj Kumar A Jayaram Bobba Kamakoti.V
Indian Institute of Technology Madras
email: kama@iitm.ernet.in

Abstract

Modern day Field Programmable Gate Arrays (FPGA) include in addition to Look-up Tables, reasonably big configurable Embedded Memory Blocks (EMB) to cater to the on-chip memory requirements of systems/applications mapped on them. While mapping applications on to such FPGAs, some of the EMBs may be left unused. This paper presents a methodology to utilize such unused EMBs as large look-up tables to map multi-output combinational sub-circuits of the application, which, otherwise would be mapped on to a number of small Look-Up Tables (LUT) available on the FPGA. This inturn leads to a huge reduction in the area of the FPGA, utilized for mapping an application. Experimental results show that our proposed methodology, when employed on popular benchmark circuits, can lead to additional 50% reduction in area utilized when compared with other methodologies reported in the literature.

1. Introduction

Field Programmable Gate Arrays (FPGAs) are programmable devices that can be configured for a wide variety of applications. They enable faster implementation and emulation of circuit designs on hardware. The flexibility provided by Field Programmable Gate Arrays (FPGAs) through the presence of reconfigurable elements has increased their popularity in comparison with the conventional ASIC designs.

Among the various possible architectures, lookup-table (LUT) based FPGA architectures have been the most popular ones. A LUT-based FPGA consists of an array of programmable logic blocks (PLBs) together with programmable interconnections. The core of a PLB is a k-input LUT (k-LUT) that can implement any combinational logic function with up to k inputs and a single output. The problem of mapping a synthesized boolean circuit into a LUT based network has been studied extensively. Different algorithms have been proposed based on various optimization criteria, namely, *performance* [1, 2], *area* [3, 4], and *routability* [5, 6]. Architectures involving multi-output LUTs and heterogeneous LUTs have also been studied and technology mapping algorithms for such architectures have also been proposed.

Modern day FPGA based technologies facilitate a complete System-On-Chip implementation on FPGAs. Such large systems require more memory than smaller circuits. Since provision of on-chip memory leads to higher clock frequencies and lower I/O pin requirement for such systems, FPGA vendors have included memory arrays or Embedded Memory Blocks (EMBs) in most of their architectures. Two implementations of on-chip memory have been considered: *fine-grained* and *coarse-grained*. In FPGAs employing fine-grained memory, such as the Xilinx 4000 series of FPGAs, each lookup-table can be configured as a small RAM, and these RAMs can be combined to implement larger memories [7]. The coarse-grained approach is used in Altera 10K devices [8], the Actel 3200DX and SPGA parts [9] and the Lattice ispLSI 6192 FPGAs [10]. In these devices, large arrays are embedded onto the FPGA. Devices in the Altera 10K family contain between 3 and 16 2K-bit arrays, the Actel 3200DX parts contain between 2 and 32 256-bit arrays, Actel SPGAs contain between 2 and 32 2K-bit arrays, and the Lattice ispLSI 6192 devices contain a single 4608 bit array.

The coarse-grained approach results in significantly denser memory implementations, since the per-bit overhead is much smaller [11]. However, the FPGA vendor has to partition the chip into memory and logic regions when the FPGA is designed. Since circuits have varying memory requirements this could lead to poor device utilizations for logic-intensive or memory-intensive circuits. To avoid this wastage of resources, techniques have been proposed to *utilize unused RAM as large multi-output lookup-tables to implement the combinational sub-circuits in the system*. We then have a heterogeneous architecture in which both LUTs and memory are used as *logic elements* in the implementation of a circuit. Multi-output

1530-1591/04 $20.00 (c) 2004 IEEE

memory arrays can implement large regions of the synthesized circuit with significant area savings. Similarly, if several combinational levels can be packed into a single memory array, significant speed improvements can also be obtained. Hence, we need to identify regions of the circuit that can be efficiently mapped into the memory arrays.

2. Previous Work

Murgai has studied the implementation of logic in memory arrays [12]. However it was used for the development of a circuit emulator system. Wilton has proposed algorithms for utilizing the unused memory arrays in FPGAs with on-chip memory [13]. The approach, SMAP, first involves mapping the entire circuit into LUTs and then attempting to pack maximum number of LUTs into the available memory heuristically. The idea of maximal network cuts is used in identifying the feasible regions. Cong and Xu have proposed another approach for mapping logic into memory [14]. In their approach, EMB_Pack, the circuit is first mapped into LUTs using the best available algorithms. They then identify regions of the LUT network which when mapped into the embedded memory arrays would not require duplication of existing LUT nodes. While in the case of SMAP, the objective of the algorithm is to achieve area minimization, EMB_Pack minimizes the area maintaing the circuit delay. In this paper, we propose a different approach for area minimization in which the reconvergent regions of the circuit are identified as potential candidates for mapping to the memory arrays.

3. Preliminaries and Problem Definition

The circuit is represented by a directed graph $G(V, E)$ where the vertices V represent nodes, and the edges E represent the interconnections between the nodes. We now present a few definitions pertaining to reconvergence in the circuit.

Given two nodes x and y, if there exists at least one directed path from x to y, then node x is said to *drive* node y and y is said to be *driven* by x. Since every gate has a unique output net, this concept could be used analogously for nets also.

A *stem node* $s \in V$ is one which drives more than one gate. A stem node s is called a *reconvergent fanout stem* if there exists more than one disjoint path from s to another node $t \in V$. In this case, t is called reconvergent node of s.

A *closing reconvergent node* of a reconvergent fanout stem s is a reconvergent node of s that does not drive any other reconvergent node of s.

Primary Stem Region of a reconvergent fanout stem s consists of all the nodes v and v's output edges such that v is located on a path from s to any of the closing reconvergent nodes of s. Two primary stem regions are said to $overlap$ if there exists at least one node common to both regions.

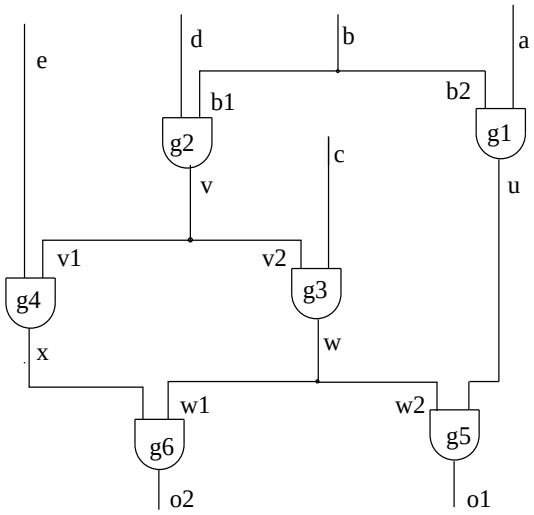


Figure 1. Gate_level circuit

In the circuit shown in Figure 1, b is a reconvergent fanout stem and $g5$ is its reconvergent node. The primary stem region of b includes the nodes $g1, g2, g3$ and $g5$ while the primary stem region of v includes the nodes $g3, g4$ and $g6$. These two regions overlap with one another.

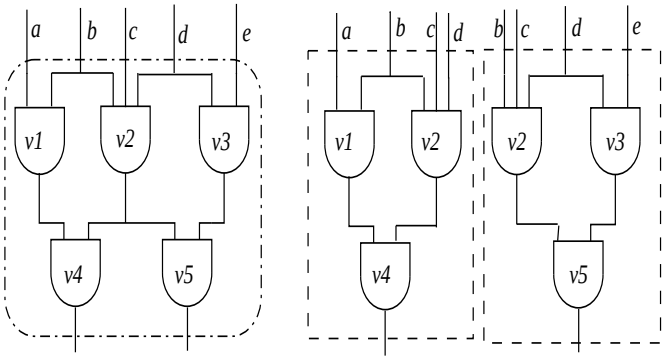
The technology mapping problem deals with the implementation of a synthesized Boolean circuit using logic cells from a pre-specified family. It is one of the tough and challenging problems in the field of automated design. In this paper, we address this problem for LUT based FPGA architectures consisting of memory elements. In this case two types of resources are available, viz., *the k-LUTs provided in PLBs* and *the reconfigurable arrays provided by embedded memory blocks*. Hence, the result of an algorithm mapping on such architectures is a circuit mapped to both memory arrays and small look-up-tables. The technology mapping problem for a LUT and embedded memory based hybrid FPGA architecture can be defined as follows:

An application would typically consist of memory elements (that have to be mapped on the available on-chip embedded memory) and combinational elements that could be mapped both on LUTs and on-chip embedded memory. Given such an application that requires t memory arrays and a FPGA architecture with a maximum of $N(\geq t)$ on-chip embedded memory arrays and k-LUTs, find a mapping such that the number of k-LUTs used is minimized. In other words, the utilization of the $n\ (= (N - t))$ available on-chip embedded memory for implementing the combinational elements of the circuit is maximized.

The value of k is a constant and without loss of generality we assume it to be 4.

/* Phase 1: Pre-processing */
Decompose given circuit into a 2-input network using DMIG.
/* Phase 2: Reconvergence Analysis */
θ = set of all non-intersecting overlapping reconvergence regions satisfying the input/output constraints.
/* Phase 3: Memory Mapping */
for each region in θ **do**
  while region is expandable to nearest memory configuration **do**
    Assign priorities to each of the gates neighbouring the region.
    Select the gate with highest priority that can be added to region without violating the input-output constraints.
  end while
  Assign priority to each of the regions
end for
Let there be n unused memory arrays in the FPGA after the mapping of all the memory elements of the application. Select the n best non-intersecting regions and map them into these memory arrays.
/* Phase 4: LUT mapping */
Map the rest of the circuit into LUTs using DAG-Map

Figure 2. MemMap Algorithm



(a) Mapping of a reconvergent region using MemMap

(b) Mapping solution by DagMap

Figure 3. Mapping of reconvergent regions

4. The Algorithm

We now give an overview of our approach. The algorithm, as illustrated in Figure 2, consists of four main stages, namely, *Pre-Processing*, *Reconvergence Analysis*, *Memory Mapping* and *LUT Mapping*. In the pre-processing stage, we convert the given circuit into an equivalent circuit to improve performance of the mapping algorithms. In the next stage, we analyze the circuit structurally for reconvergence and identify regions that could be mapped into memory. In the memory mapping stage, the prospective regions identified in the previous stage are further expanded for area optimization and the best ones are mapped into the memory arrays. In the final stage, the remaining circuit is mapped into LUTs using the existing LUT mapping algorithm DAG-Map [15].

4.1. Pre-Processing

In the first stage of the algorithm, we take the given circuit in the form of a Boolean network and convert it into an equivalent two-input network. A two-input network is one in which each gate has at most two inputs. This decomposition is done using the DMIG algorithm proposed by Cong et al [15]. It has been shown that this conversion leads to better mapping of the circuit into LUTs by minimizing the overall depth of the decomposed circuit.

4.2. Reconvergence Analysis

In this stage, the circuit obtained from the pre-processing stage is analyzed for reconvergence. We identify regions of circuits that consist of overlapping reconvergent regions using an approach similar to the one proposed by Dey, Brglez and Kedem [16].

We first find out the primary fanout stem regions of all the fanout stems in the circuit. Next we combine overlapping primary fanout stem regions to form larger reconvergence regions. However, if we are to map these regions into memory arrays, we have to make sure that the number of inputs and outputs of the regions are less than the number allowed by the memory arrays. Hence, while combining primary fanout stem regions, we impose a constraint on the number of inputs and outputs of the resultant region. Memory reconfigurability allows us to have different input-output configurations. For example, a 2048-bit memory array could be configured as a 2048×1(11 input, 1 output), 1024×2(10 input, 2 output), 512×4(9 input, 4 output) or 256×8(8 input, 8 output). So we combine the regions until they satisfy at least one of the above configurations. Note that the resulting set of regions is not unique as the order of combination of regions affects the final set of regions.

Intuitively, these regions can be attributed with a high node to external pin count ratio (sum of inputs and outputs). The internal reconvergence can be expected to significantly decrease the number of output pins for the region while the reconvergent fanout stems at the input of the region would lead to lesser number of input pins. By attempting to map these regions into the memory, we are trying to pack a large number of nodes with *limited external pin*

counts into the memory arrays. This has a great impact in reducing the number of LUTs required to map the rest of the circuit. The results that we obtained suggest that reconvergent regions are particularly suitable for mapping into the memory arrays.

In Figure 3, we give an example of a reconvergent subcircuit where the logic represented can be efficiently implemented using memory arrays. If the circuit were to be mapped using 4-LUTs, then the node $v2$ will have to be duplicated and the mapped solution would contain 2 LUTs. However, if we consider the concept of overlapping reconvergence regions, then the whole sub-circuit can be implemented in a single memory array with 5 inputs and 2 outputs. Thus by mapping reconvergence regions to memory, it is highly likely that we avoid node duplication and hence decrease the number of LUTs required.

4.3. Memory Mapping

In this stage, the overlapping reconvergent regions that can be mapped to the memory arrays are selected and expanded till they just satisfy the pin constraint imposed by the memory arrays. Expansion is a crucial step since there is a possibility of including more gates into the reconvergent regions. We have developed a heuristic that attempts to maximize the LUTs covered by the region while maintaining the input/output constraints specified by the memory arrays.

The heuristic employs a greedy approach in finding out the gate to be selected next from an array of *candidate gates*. The candidate gates of region R are essentially the neighbouring gates of R, i.e the gates that are not part of R and either drive a gate in R or are driven by a gate in R. We then expand the regions by progressively including the candidate gates. The candidate gates are selected on the basis of a weight function which is determined as explained below.

First the given circuit is mapped into LUTs using the DAG-Map algorithm. All the gates in the circuit are marked with the LUTs they belong to. Note that a gate can belong to more than one LUT because of duplication of gates in DAG-Map algorithm. If R be the reconvergence region being considered, and g be a candidate gate:
Let $\phi(g)$ be the set of LUTs containing g in the mapping of the circuit and let $\alpha(g)$ be the set of LUTs that contain g and at least one gate that belongs to R.

cov(g) = $|\phi(g)|$
pcov(g) = $|\alpha(g)|$
weight(g) = pcov(g)/cov(g).

The motivation for choosing this function is as follows:
When R is expanded, there will be a change in the number of LUTs completely covered by R as well as the number of those partially covered. A completely covered LUT with respect to R is one for which all the gates in it are part of R, while a partially covered LUT is one that is not a completely covered LUT but has at least one gate in common with R. In Figure 4, g is the candidate node being considered for expansion of region R. $L1$, $L2$ and $L3$ are LUTs obtained by mapping the circuit. While $L1$ is not covered by R, $L2$ is partially covered by R and $L3$ is totally covered by R.

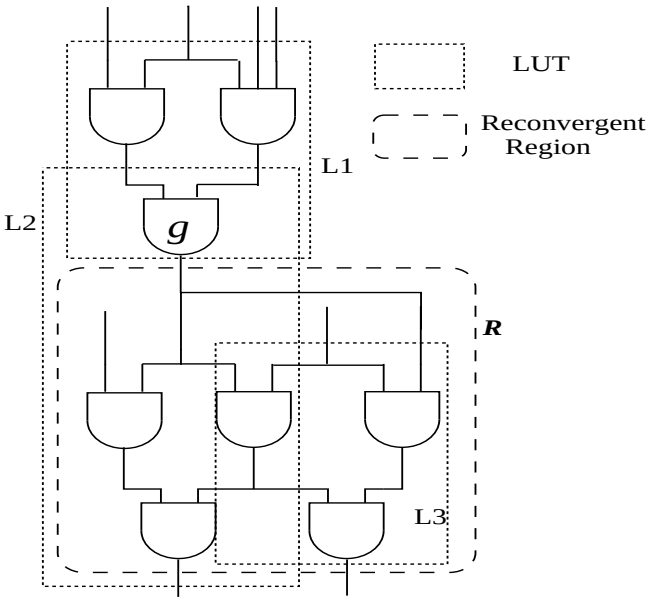


Figure 4. Expansion of Reconvergent Regions

While increasing the number of completely covered LUTs remains the final objective, increase in the number of partially covered LUTs is not desirable since the fate of partially covered LUTs remains unpredictable. Hence R should be expanded in a manner that increases the possibility of covering more LUTs without significantly increasing the number of partially covered LUTs. Since the number of LUTs containing g and not covered at all by R add to partially covered LUTs after g is added, we choose that g which has the minimum fraction of such LUTs, i.e the gate g with the maximum weight.

After expanding each of the regions, we next assign a priority to them. Here we consider the regions on the basis of m, the number of completely covered LUTs. The region R having maximum m is assigned the highest priority and so on. Note that the expansion stage could cause some of the regions to intersect with each other. So if we have n available memory arrays, then we consider the n best non-intersecting regions for mapping into the memory blocks.

Circuit	# 4-LUTs in original circuit	Number of 4-LUTs removed (CPU time(in secs)			
		N = 1	N = 4	N = 8	N = 16
pair	654	27(3)	59(4)	110(4)	187(6)
c7552	850	23(27)	95(27)	181(29)	295(30)
c5315	673	23(6)	49(6)	105(7)	154(8)
c6288	555	20(2)	52(2)	86(2)	134(3)
i10	1116	26(11)	54(12)	98(12)	146(13)
rot	296	21(< 1)	45(< 1)	79(< 1)	158(< 1)
dsip	1468	28(42)	62(42)	93(44)	136(45)
alu4	367	67(< 1)	127(< 1)	205(< 1)	-
alu2	201	36(< 1)	79(< 1)	118(< 1)	-
dalu	454	23(8)	55(8)	110(9)	145(9)
b14	1562	15(29)	28(30)	43(33)	82(37)
b15	1837	13(46)	33(47)	47(53)	106(60)
b20	3412	19(230)	44(241)	57(255)	123(271)
s13207	538	37(36)	54(36)	90(38)	176(40)

Table 1. Results for benchmark circuits using MemMap algorithm

4.4. LUT Mapping

This is the final phase of the algorithm in which the residual circuit left after mapping onto memory arrays is mapped into LUTs. The DAG-Map algorithm is used to implement this mapping. The LUT mapping algorithm could be oriented towards different optimization criterion like area, delay or routability.

4.5. Salient Features of Our Method

We now discuss some advantages offered by our algorithm:

1. It reduces reconvergence in the circuit to be mapped onto LUTs by mapping reconverging areas onto memory blocks. DAG-Map which is the algorithm used for mapping onto LUTs is more efficient when reconvergence in circuit is less. DAG-Map algorithm is optimal when the initial network is a tree or a set of trees [15].
2. All potential regions that can be mapped onto EMBs are obtained in a single phase. Depending on the number of available EMBs, we select the best possible regions. Hence, the same algorithm can be used for mapping to both single EMB and multiple EMBs, without adding extra complexity.
3. The second phase of the algorithm exploits the flexibility for reconfiguration provided by embedded memory blocks. This is done by expanding a given overlapping reconvergent region to meet constraints of the nearest configuration of the memory block.

5. Experimental Results

We now present the experimental results obtained by implementing the MemMap algorithm and testing it on a set of both combinational and sequential circuits taken from various benchmark suites. The algorithms were coded in C and were run on an Intel Xeon based dual processor 1.2 GHz server with 2 GB RAM.

The results presented here assume an FPGA with N 2048-bit arrays, each of which can be configured as $2048 \times 1, 1024 \times 2, 512 \times 4$, or 256×8. This is the array size and flexibility available with Altera FLEX 10K devices.

First, we ran MemMap on various benchmark circuits varying N, the number of memory blocks. The results are presented in Table 1. For the set of benchmarks that we have taken, for $N = 1$, an average of 28 LUTs were removed using the technique. For $N = 4, 8$ and 16, the number of LUTs removed on an average are 61, 103 and 154 respectively. The results clearly indicate the effectiveness of the identified regions in reducing the overall number of LUTs in the circuit. The CPU time taken for the whole mapping process is indicated. We observed that the time taken for memory mapping stage of MemMap is negligible. We also studied the impact of MemMap on delay of the circuit. Table 3 gives the delay results of the circuits mapped using DagMap and MemMap for $N = 8$ respectively. We do not observe a significant increase in delay in most cases.

We also compared our results with that of SMAP [13]. In Table 2, we give the number of LUTs removed by both the algorithms for different number of memory blocks. The number of LUTs initially present before memory mapping are also indicated. In case of SMAP, a blocking factor(BF)

Circuit	# 4-LUTs removed/Initial # of 4-LUTs					
	N=1			N=8		
	SMAP	Mem Map	%Diff	SMAP	Mem Map	%Diff
pair	13/641	27/654	110	81/641	110/654	28
c7552	15/679	23/850	55	94/679	181/850	95
c5315	12/596	23/673	94	76/596	105/673	42
c6288	19/527	20/555	5	93/527	86/555	-7
dsip	18/1370	28/1468	56	87/1370	93/1468	8
bigkey	18/1707	21/1750	16	88/1707	106/1750	22
i10	18/994	26/1116	45	94/994	98/1116	4
ralu32	20/3659	26/3726	30	118/3659	143/3726	24

Table 2. Comparison with SMAP algorithm

Circuit	Delay	
	DagMap	MemMap
pair	9	14
c7552	13	15
c5315	15	15
c6288	25	24
i10	15	15
rot	12	15
alu4	14	13
alu2	12	13
dalu	13	16
b14	19	21
b15	23	25
b20	21	22
s13207	12	13

Table 3. Delay comparison for circuits

of 1 was chosen as it gave better results for most of the circuits. BF is a parameter used by the algorithm in [13] to combine available memory arrays into larger arrays. If n is the number of available EMBs, then they are combined into n/BF larger arrays for mapping. MemMap when compared to SMAP, on an average reduced 51% more nodes for a single block of memory and 27% more nodes for 8 blocks of memory.

6. Conclusions

This paper presents a new algorithm for technology mapping onto heterogeneous architectures containing LUTs and embedded memory blocks. For the first time, the concept of reconvergence is used in the field of FPGA mapping and is shown to be effective. We have tested our algorithm on a set of large benchmark examples and achieved satisfactory results. The algorithm presented has focused on minimizing the area required to implement circuits.

References

[1] R.J.Francis, J.Rose, and Z.Vranesic, "Technology mapping for lookup table-based FPGAs", in *Digest Intl. Conf. on Computer-Aided Design*,pages 568-571, 1991.

[2] J.Cong and Y.Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," in *IEEE Trans. on Computer-Aided Design*, 13:1-11, 1994.

[3] R.J.Francis, J.Rose and Z.Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," in *Proc. ACM/IEEE Design Automation Conf.*, pages 227-233, 1991.

[4] R.Murgai, Y.Nishizaki, N.Shenoy, R.K.Brayton and A.Sangiovanni-Vincentelli, "Logic synthesis algorithms for table look up programmable gate arrays," In *Proc. ACM/IEEE Design Automation Conf.* pages 620-625, 1990.

[5] M.Schlag, J.Kong and P.K.Chan, "Routability-driven technology mapping for lookup table-based FPGAs," in *IEEE Trans. on Computer-Aided Design*, 13:13-26, 1994.

[6] N.Bhat and D.Hill, "Routable technology mapping for FPGAs," in *ACM/SIGDA Workshop on FPGAs*, pages 143-148, 1992.

[7] Xilinx, Inc., *XC4000 Series (E/L/EX/XL) Field Programmable Gate Arrays v1.04*, September 1996.

[8] Altera Corporation, *Databook*, June 1996.

[9] Actel Corporation, *Datasheet: 3200DX Field-Programmable Gate Arrays*, 1995.

[10] Lattice Semiconductor Corporation, *Datasheet: ispLSI and pLSI 6192 High Density Programmable Logic with Dedicated Memory and Register/Counter Modules*, July 1996.

[11] T.Ngai, J.Rose, and S.J.E.Wilton, "An SRAM-Programmable fi eld-confi gurable memory," in *Proceedings of the IEEE 1995 Custom Integrated Circuits Conference*, pp.499-502, May 1995.

[12] R.Murgai, F.Hirose and M.Fujita, "Logic Synthesis for a single large look-up table," in *Proc. Int. Workshop on Logic Synthesis*, May 1995.

[13] S.J.E.Wilton, "Heterogeneous Technology Mapping for area reduction in FPGA's with embedded memory arrays," in *IEEE Transactions on Computer-Aided Design on Integrated Circuits and Systems*, Vol.19, No.1, Jan 2000.

[14] J.Cong and S.Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb 1998, pp.171-178

[15] K.C.Chen, J.Cong, Y.Ding, A.B.Kahng and P.Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," in *IEEE Design and Test of Computers*, pp.-20, Sep.1992.

[16] S.Dey, F.Brglez and G.Kedem, "Corolla-based circuit partitioning and resynthesis," in *ACM/IEEE 27th Design Automation Conference*, pp.607-612, June 1990.