

**A Path Based Algorithm for Timing Driven  
Logic Replication in FPGA**

By Giancarlo Beraudo

B.S., Politecnico di Torino, Torino, 2001

THESIS

Submitted as partial fulfillment of the requirements

for the degree of Master of Science in Electrical and Computer Engineering

in the Graduate College of the

University of Illinois at Chicago, 2002

Chicago, Illinois

# TABLE OF CONTENTS

<b>CHAPTER 1.....</b>	<b>6</b>
<b>INTRODUCTION.....</b>	<b>6</b>
1.1    TECHNOLOGY TRENDS.....	6
1.2    OVERVIEW OF THE TIMING DRIVEN PLACEMENT PROBLEM.....	7
1.2.1    State of the Art.....	7
1.2.2    Our Approach.....	10
1.3    THE LOGIC REPLICATION PROBLEM .....	11
1.3.1    State of the Art.....	11
1.3.2    Our Approach.....	15
1.4    FIELD PROGRAMMABLE GATE ARRAY.....	18
<b>CHAPTER 2.....</b>	<b>22</b>
<b>METHODS .....</b>	<b>22</b>
2.1.    VERSATILE PLACE AND ROUTE.....	22
2.2.    BENCHMARK FPGA ARCHITECTURE.....	25
2.3.    STATIC TIMING ANALYSIS .....	28
2.4.    DELAY MODEL .....	31
<b>CHAPTER 3.....</b>	<b>35</b>
<b>A PATH BASED ALGORITHM .....</b>	<b>35</b>
3.1.    PRELIMINARY ANALYSES .....	35
3.2.    OVERVIEW OF THE ALGORITHM.....	39
3.3.    NETWORK ANALYSIS AND CELL SELECTION .....	41
3.4.    TIMING CONSTRAINTS GENERATION AND SLOT SELECTION .....	43
3.5.    FANOUT PARTITIONING .....	47
3.6.    LEGALIZER .....	48
3.7.    AN EXAMPLE .....	49
<b>CHAPTER 4.....</b>	<b>56</b>
<b>EXPERIMENTAL RESULTS .....</b>	<b>56</b>
4.1.    TIMING PERFORMANCE IMPROVEMENT .....	56
4.2.    WIRE LENGTH IMPACT .....	59
4.3.    ROUTABILITY IMPACT .....	60
<b>CHAPTER 5.....</b>	<b>62</b>
<b>CONCLUSIONS AND FUTURE WORK.....</b>	<b>62</b>
5.1.    CONCLUSIONS.....	62
5.2.    NON-MONOTONE CLUSTERS .....	63
5.3.    REPLICATION OF MEMORY ELEMENTS AND PADS.....	64
<b>CITED LITTERATURE.....</b>	<b>65</b>

## LIST OF TABLES

Table 1: Number of modules with criticality greater than 0.9.....	38
Table 2: Pre-routing and post-routing results .....	57
Table 3: Results with greater availability of routing resources .....	58
Table 4: HPWL degradation.....	59
Table 5: Degradation in minimum required number of tracks .....	60

## LIST OF FIGURES

Figure 1: Net based failure scenario .....	9
Figure 2: Example of logic replication during gate sizing .....	12
Figure 3: Situation in which replication can improve cut size .....	13
Figure 4: Situation in which replication can improve cut size .....	14
Figure 5: Situation in which replication can improve timing performance.....	15
Figure 6: Situation in which replication can improve timing performance.....	16
Figure 7: Situation in which replication can improve timing performance.....	16
Figure 8: Situation in which replication can improve timing performance.....	17
Figure 9: Monotone regions .....	18
Figure 10: FPGA structure.....	18
Figure 11: FPGA interconnections .....	20
Figure 12: Design flow of VPR.....	22
Figure 13: A 4x4 instance of the benchmark FPGA .....	25
Figure 14: A configurable logic block.....	26
Figure 15: Pin locations .....	26
Figure 16: Routing structure.....	27
Figure 17: From logic blocks to timing nodes.....	29
Figure 18: Circuit elements modeled as RC ladders .....	32
Figure 19: Manhattan distance and number of crossed switches.....	33
Figure 20: Delay as function of Manhattan distance in Xilinx Virtex FPGAs.....	34
Figure 21: Sink delays of Bigkey.....	36
Figure 22: Sink delays of Elliptic .....	36
Figure 23: Sink delays of Frisc .....	36
Figure 24: Positions of the nodes with criticality greater than 0.9 in Frisc.....	39
Figure 25: Pseudo code of the algorithm .....	40
Figure 26: A path and three monotone regions .....	43
Figure 27: Feasibility region .....	44
Figure 28: Four feasibility regions with a common intersection .....	46
Figure 29: Failure scenario for wire length driver fanout partitioning .....	47
Figure 30: The four sub paths .....	50
Figure 31: Feasibility regions with respect to the node E.....	51
Figure 32: Replication of cell C.....	52
Figure 33: Situation at the beginning of the second iteration .....	53
Figure 34: Feasibility regions with respect to node B.....	54
Figure 35: Final situation .....	54
Figure 36: A path with non-monotone clusters.....	63

## SUMMARY

In this thesis we study the possibility of using logic replication in order to improve timing performance in VLSI design. In particular, we restrict our analysis to FPGA architectures. We describe an algorithm for post-placement timing optimization that exploits the additional freedom degree of logic duplication.

Experiments on 20 large MCNC benchmark circuits demonstrate that that our technique improves performance of a circuit of an average 13.5% with respect to VPR, a well known timing driven placement tool for FPGA, by replicating a very small amount of cells (less than 0.4% of the cells). The wire length degradation is of an average 2.7%.

# CHAPTER 1

## Introduction

### 1.1 Technology Trends

VLSI design process is usually divided in several steps: logic synthesis, physical design and mask design. This thesis will focus on physical design. In particular physical design can further be divided into circuit partitioning, floorplanning, placement and routing. Each of these phases determines with an increasing degree of accuracy the position of modules and wires in the final layout. Partitioning is necessary if the circuit is too big and can't be placed on a single chip. Floorplanning is necessary in macrocell design and consists in figuring out alignment and orientation of modules so that some parameters are minimized. Placement is the task that finds the exact position of each module. Routing consists in assigning routing resources (buffers, tracks, switching boxes...) to the nets.

One of the most important trends in silicon technology has been the scaling down of device and line geometries. The minimum transistor width that can be built on silicon is becoming smaller and smaller, from 8  $\mu\text{m}$  in the late seventies to a value of less than 0.2  $\mu\text{m}$  at the end of the nineties. Moore's law [1] of 1965 gives a good picture of this ongoing reduction. It says that the minimum transistor feature size decreases by 0.7X every three years. Transistor scaling has been made possible by the improved lithographic capability to print shorter gate lengths and the ability to grow nearly perfect insulators with ever decreasing thickness. As explained in [2] when the transistor size is reduced by a factor  $S$  on the three dimensions, the

switching delay of the transistor is reduced of a factor  $S$ . At the same time the local interconnect delay doesn't vary and the global interconnect delay increases of a factor  $S^2$ . The consequence is that the total delay of a circuit will depend more and more heavily by the interconnection delay. As a result working on performance optimization of the interconnections is becoming one of the most important tasks of the modern approach to the physical layout problem.

The problem of reducing the impact of interconnections on overall performance of the circuit has been treated with several approaches: wire sizing, driver sizing, buffer insertion, placement and routing algorithm oriented to performance improvement. In the last years each single step of the layout design is focusing on timing optimization of the circuits. In particular, in order to significantly reduce the length of interconnections, the placement and routing steps are the most important ones. This thesis will focus on timing optimization at placement level exploiting the possibility of little modifications of the netlist.

## 1.2 Overview of the Timing Driven Placement Problem

### 1.2.1 State of the Art

Traditionally the objective of placement algorithms is to minimize the total layout area. The usual metric for this minimization is the total wire length. This operating scheme tries to put very close to each other the modules more strongly interconnected. Intuitively it seems obvious that decreasing net length increases performance. However, the speed of a circuit is bounded by the speed of the slowest path among memory elements or pads. Therefore there are some paths that are much more critical than the others and the nets belonging to those paths must be shortened more than the others in order to increase the performance of the circuit. Thus, timing driven placement must focus on an objective different from that of traditional

placement. Traditional placement has as objective the minimization of net length and as constraint the impossibility of overlapping modules. On the other hand, timing driven placement problem can be formalized in two ways: adopting as objective the minimization of wire length and as constraints the impossibility of overlapping modules (physical constraints) and an upper bound on the critical path delay (timing constraint); or adopting as objective the minimization of critical path delay and as constraint the impossibility of overlapping modules.

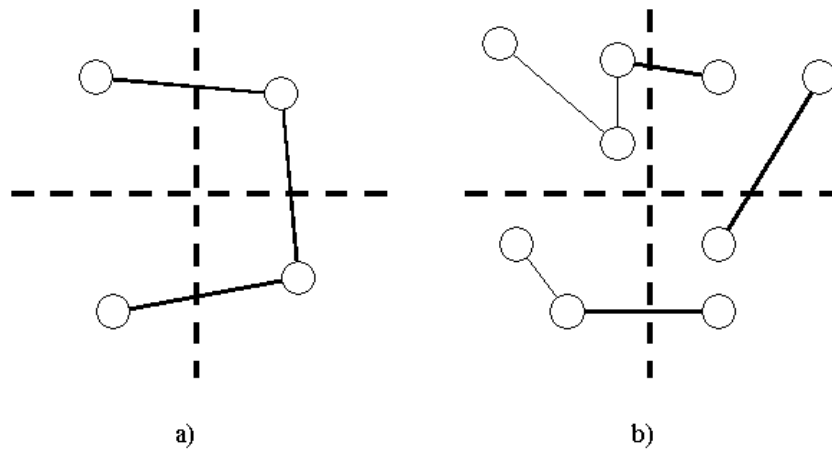
The problem of timing driven placement received first attention in the late seventies [3]. The first approaches to the problem consist in a first layout that doesn't consider timing requirements. Then timing verification by either simulation or static timing analysis is performed. If the layout doesn't meet the timing constraints, the design goes back to the physical layout phase that takes into account the nets that were limiting the performance and tries to shorten them [4] [5]. The problem with this approach was the huge overhead of design time. In fact, going back to the beginning of the layout phase after the timing verification is a very time consuming task. Another problem of this approach was that there was no guarantee of convergence of the iterative process.

Modern approaches to the problem can be roughly classified as Net Based approaches and Path Based approaches. The Net Based algorithms don't consider the timing constraints directly. They try to translate them into something else (mainly in net bounds or net weights). In contrast, the Path Based approaches consider directly the timing constraints generated by the critical paths of the circuit.

One of the first Net Based algorithms was to determine at the netlist level constraints to the length of each nets so that the layout meet timing constraints [6] [7]. The problem with this kind of approach was that it over constraints the placement and when it fails, there is no further indication in order to achieve a feasible placement in the future. This approach is too strict because by indicating for each net a length bound it forces the algorithm to search in very small portion of the solution space. In fact there is no reason why every net should respect the

constraints. Some net could violate them without compromising the total delay of the path; in fact another net of the path could balance the additional delay induced by the failing one.

Another very popular Net Based approach was to transform the timing constraints in net weights that capture the criticality of every net using a particular heuristic. The first algorithms of this category are [8] and [9]. In this case the problem is to figure out the criticality of a net before the placement is performed. Moreover these techniques can't recognize the critical paths. They will try to shorten the most critical nets without considering the paths to which they belong. For instance, the algorithm proposed in [9] uses an iterative partitioner that tries to minimize the weighted sum of the cut set. In Figure 1 there are two situations that are very different from the performance point of view, but that the tool can't distinguish.



**Figure 1: Net based failure scenario**

In the figure the bold nets are near critical nets. However in a) they belong to the same path, in b) they belong to different paths. The situation a) is much worse than the other one but these algorithms can't notice it.

The path based approach takes into account the paths directly. There are three main strategies that exploit this kind of algorithm: constructive strategies, simulated annealing strategies and analytical strategies. An algorithm for constructive timing driven placement is [10] that proposes an algorithm that places cells sequentially with a cost function that captures the timing behavior.

An algorithm for simulated annealing placement that reaches great timing performances is proposed in [11]. Simulated annealing starts from an initial situation and iteratively generates perturbations. It accepts a perturbation if it permits an improvement and with a certain probability also if it degrades the objective function. The probability that a degrading perturbation is accepted decreases iteration after iteration. Also the criteria of generating a perturbation becomes tighter and tighter. This method permits to move through local minimums during the initial iterations and to refine the placement during the final steps. This kind of algorithm is very computationally expensive but usually reaches good placements.

There are also very interesting algorithms that try to use both the Path Based and the Net Based approach. For instance, [13] uses a two phases iterative algorithm. In the first phase it performs a kind of compression on the critical path by replacing the intermediate nodes closer and closer to the sink without considering the constraints related to possible overlaps of modules (Path Based phase). During the second phase it modifies the placement in order to eliminate the overlaps using a Net Based algorithm.

### 1.2.2 Our Approach

The approach followed by this work will be an iterative algorithm that improves at each step the delay on the critical path using both placement and netlist modification without deteriorating the other near critical paths. During the timing optimization it's permitted one

overlap at a time. Immediately after, the overlap is solved with an incremental legalization procedure that creates the room for the overlapping cell without perturbing too much the placement. From this point of view the algorithm can be considered as a path based timing driven optimization algorithm with the additional freedom degree of being able to modify the netlist during the layout by performing logic replications.

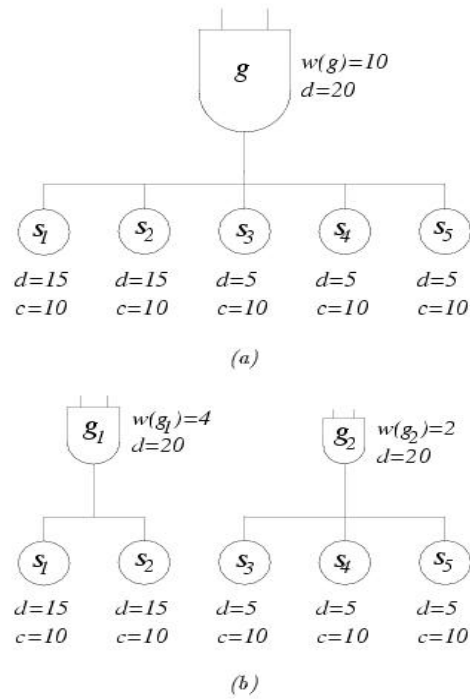
### 1.3 The Logic Replication Problem

#### 1.3.1 State of the Art

Logic replication, the duplication of logic elements in order to improve some performance parameter, is a technique that many recent works are trying to exploit. At the moment the main uses of logic replication are in timing optimization during logic synthesis and in min-cut partitioning.

A large fanout net may have a heavy capacitive load. The consequence is that in order to reduce the delay on that net it's necessary to have a large driving gate. Duplication of the driving gate reduces the fanout of the net and with appropriate fanout partitioning can potentially permit to use smaller drivers without compromising the delay or improve the speed of the circuit without any area increase [14]. In fact, among the pins of a net there are pins associated to critical paths that force the driver to be larger and pins that are non-critical and needn't a big driver. Without gate duplication you have to put a large driver in order to satisfy the timing constraints on the critical pins; however that driver is unnecessarily big for the not critical pins that make the capacitive load heavier but don't need to be so fast. If you decouple the critical pins from the not critical pins duplicating the driving gate you can speed up the pins that need to be speeded up without deal with the large capacitive load of the previous situation.

In Figure 2 you can see an example from [14] that explain this possibility. In the example  $w(g)$  is the width of the gate  $g$ ;  $d$  is the required delay of the corresponding gate;  $c$  is the input capacitance of the gate; the delay of the gate is approximated as the total capacitance of the output net divided by the width of the gate. In the situation (b), using gate duplication you can save 4 units area. Otherwise you can speed up the circuit of 12% without any increase in circuit area by sizing the gates  $g_1$  and  $g_2$  with width of 7.62 and 2.38. [14] proposes algorithms for exploiting this additional freedom degree in gate sizing.



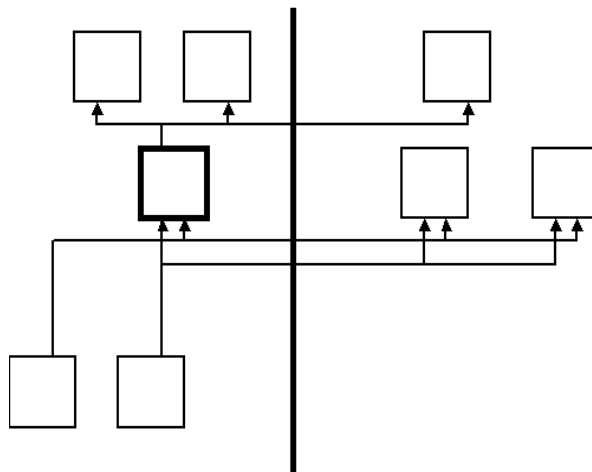
**Figure 2: Example of logic replication during gate sizing**

After this first approach to the problem the authors of [18] formalized the topic demonstrating that gate duplication is NP-complete. They demonstrated that both the fanout partitioning problem (given a gate that is about to be duplicated, splitting the fanout in order to

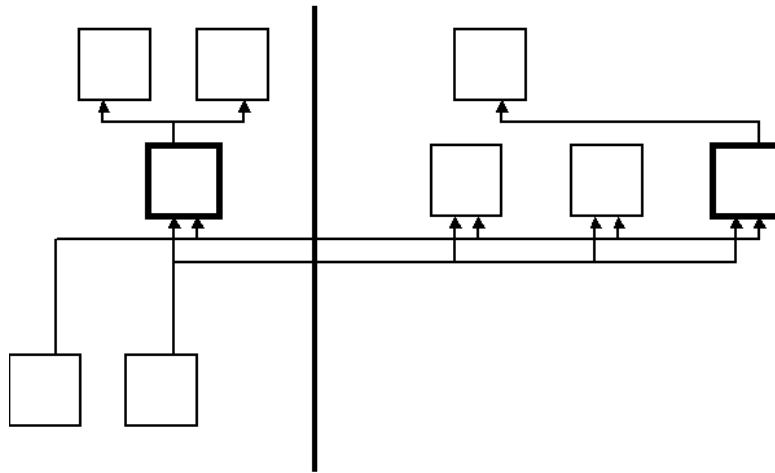
minimize the delay) and the global optimization by duplication problem (given a network, choosing which gate must be split in order to minimize the total delay) is NP-Complete.

Another widely discussed issue about logic replication is its application in min-cut partitioning. Here duplication is used in order to minimize communication between partitions. This kind of application is pretty easy to understand. In fact the goal of partitioning is to minimize the number of nets in the cut-set. Sometimes some modules are strongly connected to the two partitions. Replicating those modules can effectively reduce the number of nets in the cut-set.

As you can see in Figure 3 the bold cell is strongly connected to both the right and left partition. If you duplicate the cell, you can obtain a situation like that of Figure 4. In the new configuration the cut-size is 2 while before it was 3.



**Figure 3: Situation in which replication can improve cut size**



**Figure 4: Situation in which replication can improve cut size**

Of course in logic replication in order to maintain the same functional behavior of the network, is necessary to connect each input pin of the replicated block to the same input nets of the original cell. Hence, the gain is equal to the number of output nets that were in the cut-set minus the number of input nets that were not in the cut-set.

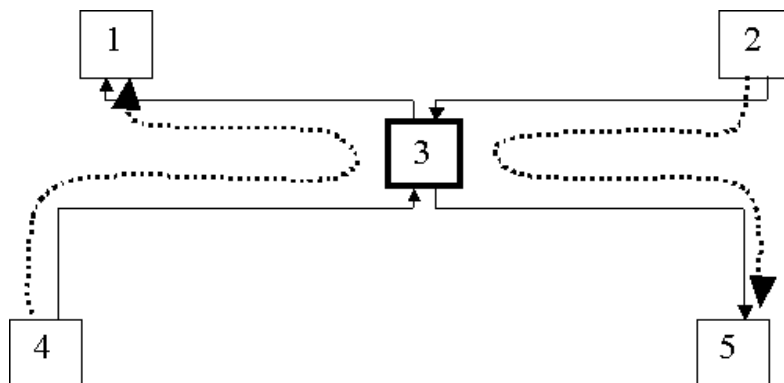
One of the first algorithms that use the additional freedom degree of logic replication is the one proposed by Kring and Newton [15]. This algorithm is an adaptation of the FM partitioner [16] that adds a third possible state for a cell that is being in both the partitions at the same time.

So far, the only research about timing implications of logic replication at the layout stage is probably [17]. The approach followed by the authors is a Net Based approach based on partitioning of a weighted network. For partitioning they use a modified version of the FM partitioner. After each partition they evaluate with a better approximation of the delay of each net and perform timing analysis. They assign to each net a weight taking into account its criticality. They try to reduce the weighted sum of the nets in the cut-set using replication. This

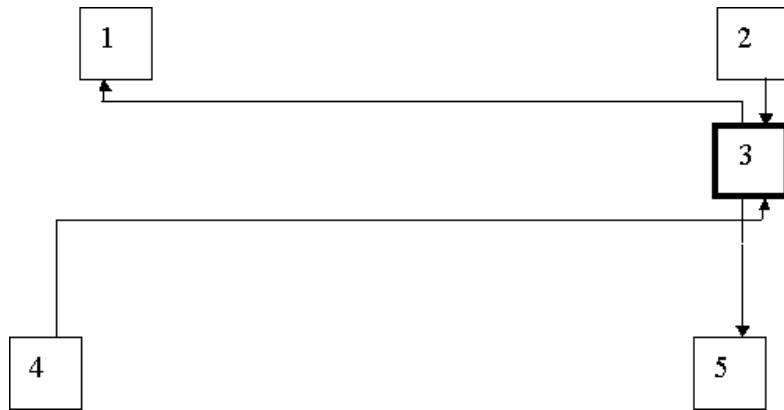
kind of approach as every Net Based approach isn't able to capture the path nature of the delay of circuits as explained in the example of Figure 1.

### 1.3.2 Our Approach

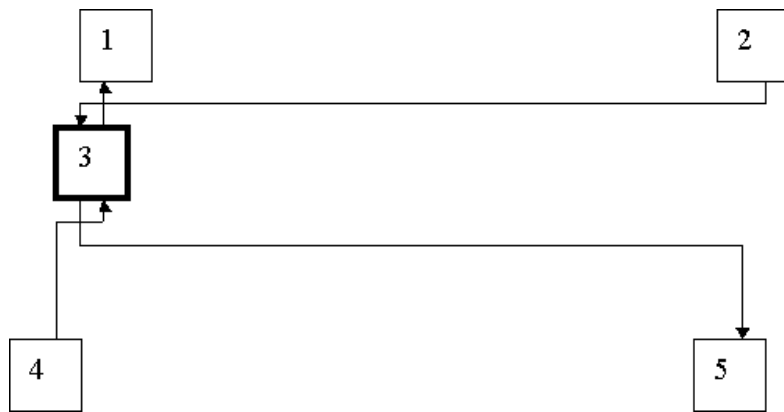
Our approach is to find the most critical cells on the critical paths and try to reduce their criticality by replication and relocation. The idea is born with the following situation. Look at the circuit of Figure 5; suppose that two critical paths flow through the cell 3. One is 4-3-1 and the other is 2-3-5. Suppose that it's impossible to relocate the cells 1,2,4 and 5 (maybe because they are pads or pins of macro-cells or because they are strongly connected to many other blocks in their region). In such a situation it's impossible to improve the performance of the circuit. In fact neither the first nor the second can be shortened without increasing the delay of the other. The optimization would oscillate among the situations of Figure 6 and Figure 7 that are worse than the previous one.



**Figure 5: Situation in which replication can improve timing performance**

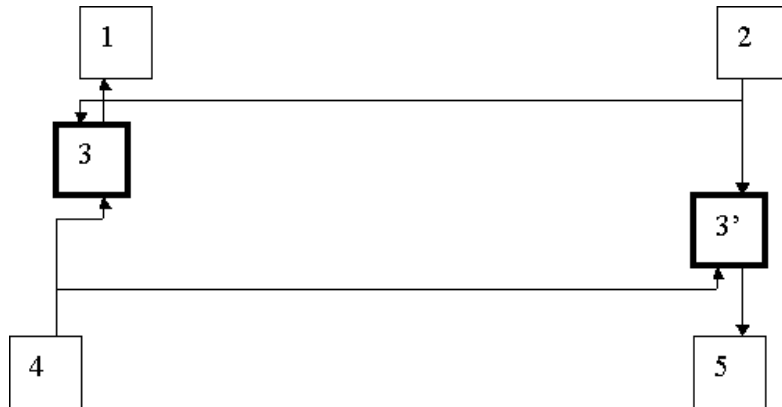


**Figure 6: Situation in which replication can improve timing performance**



**Figure 7: Situation in which replication can improve timing performance**

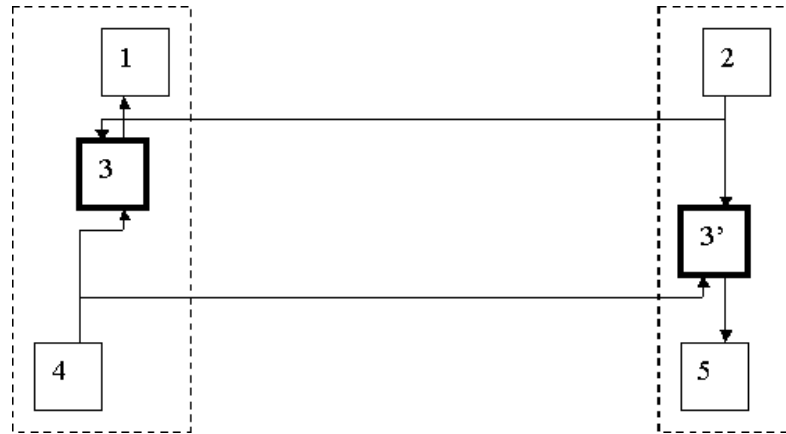
However replication of cell 3 can lead to a much better situation (Figure 8). In this case the delay on both the first and the second path can be largely improved without any increase in wire length. In particular in the initial situation there were two non-monotone paths flowing through the cell 3.



**Figure 8: Situation in which replication can improve timing performance**

A monotone path between two nodes is a path with a length that is equal to the Manhattan distance between the two nodes. Our initial assumption is that an easy way to reduce the delay on the critical path is to make the path more monotone. Monotonicity is a good parameter to guide the optimal location of the cells. In fact each couple of nodes (a sink and a source of a sub path) defines which is the area in which the intermediate nodes must be placed in order to minimize the delay on the sub path.

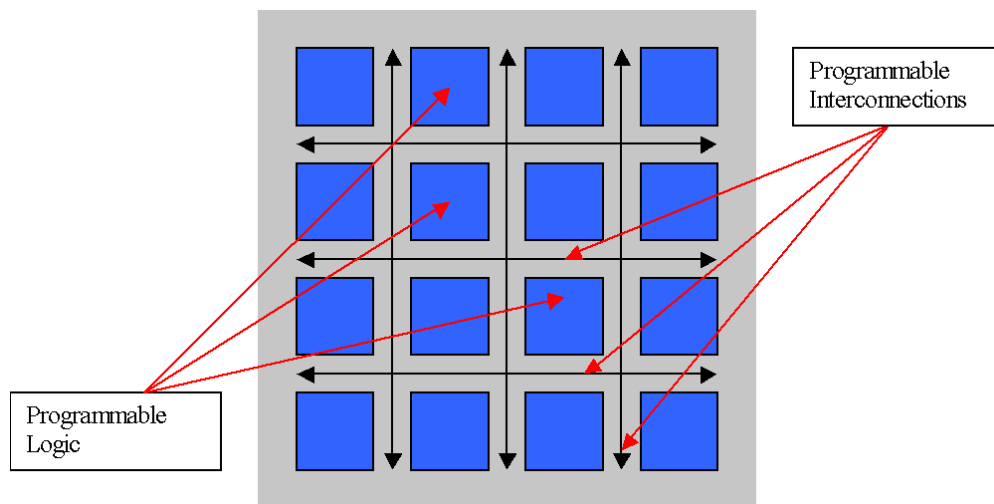
As you can see in Figure 9 the nodes 1 and 4 and the nodes 5 and 2 define two regions in which the intermediate nodes can be placed in order to minimize the delay. The farther the intermediate nodes are from those regions the more the delay increases. If a node is an intermediate node for two different critical sub paths and you want to minimize the delay on those sub paths by enforcing monotonicity, you have to place the node in the intersection of the monotone regions induced by the sub paths. If the regions have no intersection, the only solution is to replicate the intermediate node. And this is the main idea that will be developed in this work.



**Figure 9: Monotone regions**

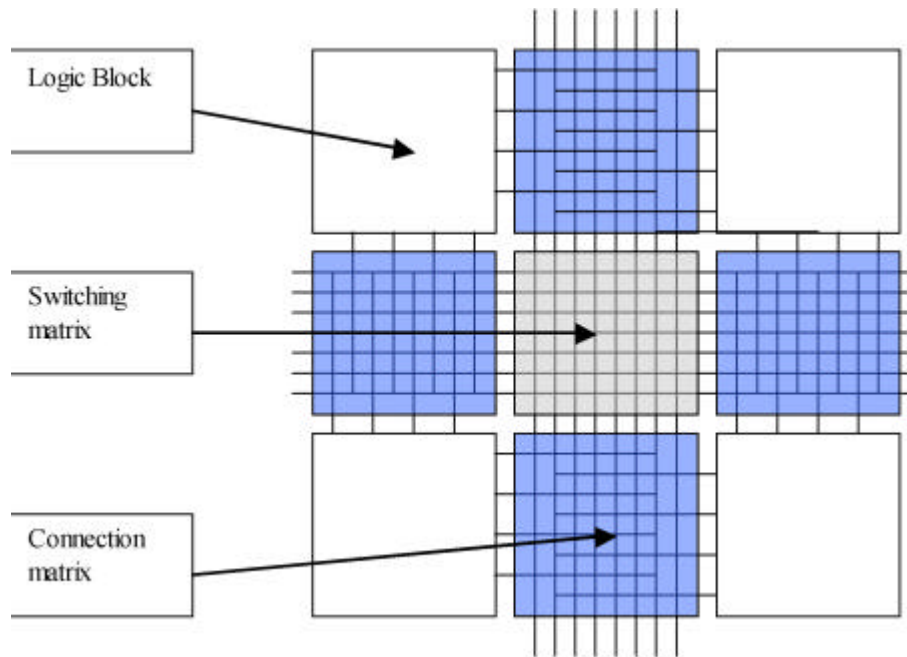
#### 1.4 Field Programmable Gate Array

Field Programmable Gate Array (FPGA) consists of a layout design style in which programmable logic blocks are embedded in programmable interconnection (see Figure 10).



**Figure 10: FPGA structure**

The success of FPGA in the last years is due to their extreme flexibility. In fact on FPGA you can implement any circuit without passing through the fabrication step. This fact permits to reduce the time to market and to significantly reduce the design cost. In fact unlike ASICs logic and interconnections are not committed but can be configured after fabrication in order to implement different functions and connectivity. The complexity of the logic blocks can vary from the simplest logic function with a few inputs and one output to the most complex arithmetic operation with multiple outputs. Usually inside each logic block there is at least one memory element that permits to implement on FPGA also sequential circuits. The FPGAs from the point of view of the silicon flexibility are between ASICs and microprocessors. In fact in ASICs the silicon is configured once and then it executes always the same operations. In microprocessors the silicon is configured every clock cycle and the same piece of silicon can potentially execute different operations at each clock cycle. In FPGA the user programs the silicon. Microprocessors pay their great flexibility with the overhead of reprogramming the silicon at each clock cycle. In FPGA that overhead is paid only when the silicon must actually be reprogrammed. The problem with FPGA is that due to the programmable nature of their silicon they are intrinsically slow. In fact programmable interconnections have values of capacitance and resistance absolutely higher than those of not programmable interconnections because of the presence of programmable switches instead of simple wires. The typical structure of the interconnections in an island style FPGA is in Figure 11.



**Figure 11: FPGA interconnections**

The connection matrixes allow each logic block to connect to the interconnection resources. The switching matrixes control the routing of the signals among the tracks. Usually switching and connection matrix are implemented with pass transistor. The pass transistor augments the capacitive load and the total resistance of the line and occupies much more room in the layout region; the consequence is a great increase of the interconnection delay and of the physical dimensions of the circuit. It was estimated that a circuit implemented on FPGA occupies 10 times the room occupied by the same circuit on ASICs and has a speed 3 times slower [19].

Moreover, the layout of FPGAs can't use buffer insertion and wire sizing techniques in order to reduce the impact of the interconnection delay due to their regular and fixed architecture. The consequence is that the speed of the circuits that are mapped on FPGAs is heavily bounded by interconnection delay. From here the necessity of developing timing driven layout technique specialized on FPGA.

But the necessity of timing driven layout for FPGA is not the only reason for developing our work on FPGA. In fact due to their regular structure FPGAs are the ideal benchmark for an algorithm. In fact delay models are much more reliable on them than on ASICs; the discreteness of the placement area allow the placement algorithms to be easier and faster. In addition, testing is not a problem on FPGA.

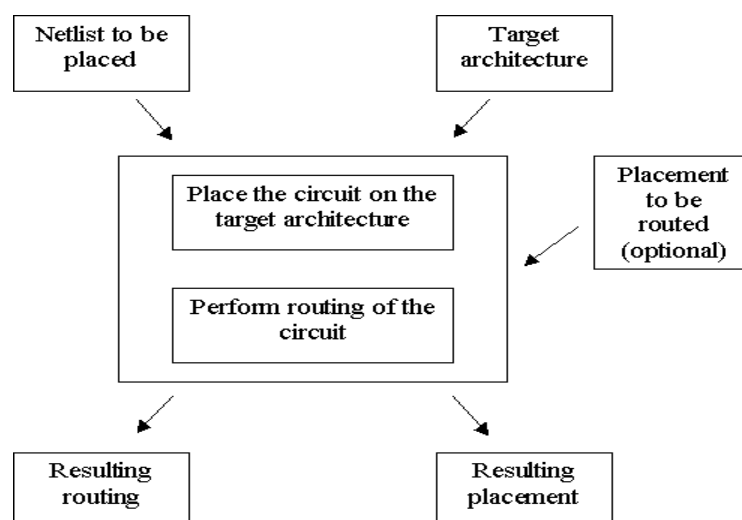
Moreover, due to the fixed number of routing tracks among the logic blocks, it's very common that some logic blocks remain empty in order to avoid that the FPGA becomes congested and, as a consequence, non-routable. This empty room inside the FPGA layout region can be used in order to replicate logic blocks.

## CHAPTER 2

### Methods

#### 2.1. Versatile Place and Route

Versatile Place and Route (VPR) is our starting point in developing our algorithm [21]. We chose VPR because it's one of the best available tools for placement and routing of FPGAs. Its good behavior in term of timing performance and wire length makes it very good for any comparison. Moreover it's a very good environment for testing because its after routing timing analyzer is very reliable. Our experiments will be always based on circuits placed by the VPR's placement tool and the final validation of the results will always be done by routing the modified netlist and placement using the VPR's router and timing analyzer.



**Figure 12: Design flow of VPR**

The design flow of VPR is that described in Figure 12. As you can see, it's a complete CAD environment in which, starting from netlist and architectural description of the FPGA, you get the placement and routing of your circuit. Its main advantage over the other tools is its great versatility. In fact it's not suited for a specific FPGA, but, due to the possibility of receiving as input the description of your target FPGA, it is appropriate for every kind of architecture [20].

The placement tool embedded in VPR is a simulated annealing placement tool. It can be configured as wire length driven and as timing driven. It uses a particular adaptive schedule for the simulated annealing algorithm. In fact usually in simulated annealing the probability that a perturbation that degrades the objective function is accepted is given by:

$$P = e^{-\Delta C / T}$$

$\Delta C$  is the degradation of the objective function caused by the perturbation and  $T$  (temperature) is a parameter indicating the current probability of accepting moves that degrade the quality of placement. The way in which  $T$  is updated is one of the factors that characterize different simulated annealing schedules. In VPR  $T$  is updated trying to maximize the time in which the algorithm performs the most productive perturbations.

The objective function varies if the placement is wire length driven or timing driven. If the placement is wire length driven, the objective function is slightly different from the usual objective function for pure wire length driven algorithms. In fact it tries to make the task of the router more effective by penalizing the congested placements. The objective function is:

$$\sum_{i=0}^{N_{nets}} q(i) \cdot [bb_x(i) / C_{av,x}(i) + bb_y(i) / C_{av,y}(i)]$$

$q(i)$  is a parameter that is function of the fanout of the net and determines how much the actual length of the net is far from the half perimeter of the bounding box containing all the pins of the net. Its value is obtained from [23].  $bb_x(i)$  is the horizontal span of the bounding box of the net; the same for  $bb_y$ .  $C_{av,x}(i)$  is the average number of tracks per channel in the x direction; the same for  $C_{av,y}(i)$ . This kind of Objective function is able to penalize the placements that try to use more routing resources than those that are available.

The timing driven placement algorithm has as objective function a tradeoff between the wire length driven objective function and a pure timing driven objective function. For each interconnection the algorithm calculates its timing criticality. The timing criticality is:

$$\text{Criticality}(i, j) = 1 - \text{Slack}(i, j) / \text{Critical\_Path\_Delay}$$

The slack of an interconnection is how much delay you can add to that interconnection without increasing the critical path delay. The timing cost of an interconnection is:

$$\text{Timing\_Cost}(i, j) = \text{Delay}(i, j) \text{Criticality}(i, j)$$

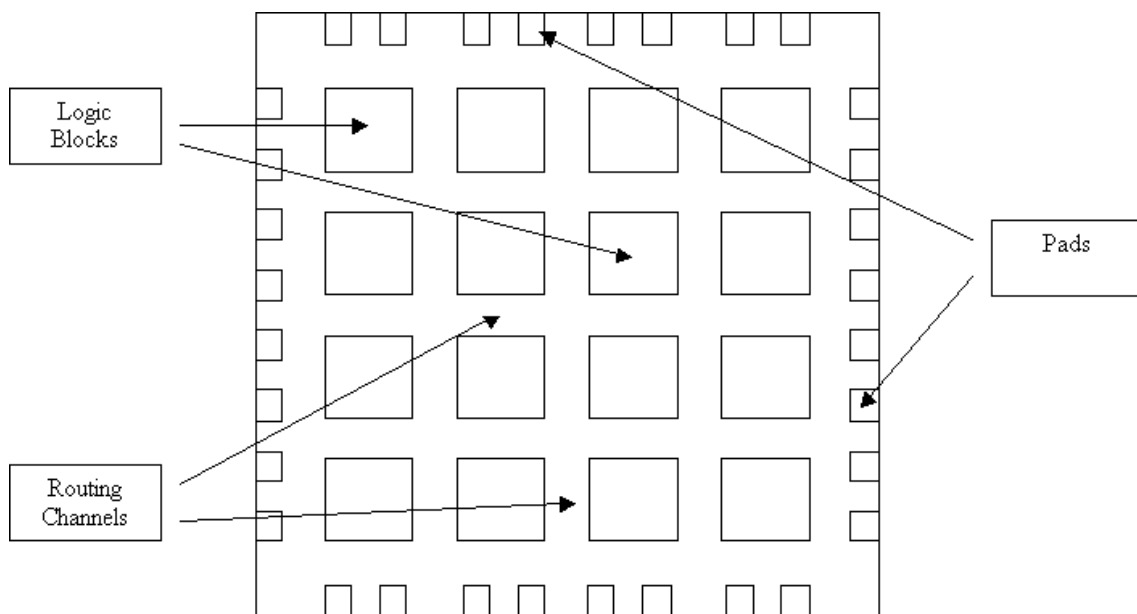
The objective function for the timing driven placement is the sum of the timing cost of all the pin-to-pin interconnections. The  $\Delta C$  is calculated as a tradeoff between the  $\Delta C$  of the pure timing driven objective function and the  $\Delta C$  of the wire length driven objective function. The tradeoff is balanced by a parameter  $\Lambda$  that can give more or less weight to the perturbations that affect wire length or timing performance.

One of the strong points of VPR is its timing analyzer. It assumes that all the slots of the FPGA from the point of view of the connectivity to the others are identical. In fact each cell due to the regularity of the architecture has the same amount of routing resources. The consequence is that the delay between two blocks of the FPGA is function of the only horizontal and vertical distances between the two blocks. Thus the timing analyzer builds a lookup table addressable with the distances  $\Delta x$  and  $\Delta y$  between the two blocks that holds the delays. The values of the delays are computed by performing the routing of the two pins net connecting the two blocks and evaluating the delay on the net using the Elmore delay model. The great advantage of this approach is the computational speed; the disadvantage is that it doesn't take into account the fanout of the actual net and the congestion of the FPGA in the region in which the net will be actually routed. In order to avoid that the timing analyzer slows the algorithm down too much, timing analysis is performed only after a number of modification of the placement. The delays of the interconnections are updated after every change (it's very fast because it requires just an access to the lookup table) but the analysis of the paths in order to extract the slacks of each

interconnection is performed only after several modifications (this would be very computationally expensive because timing analysis has complexity linear in the number of the nodes of the timing graph).

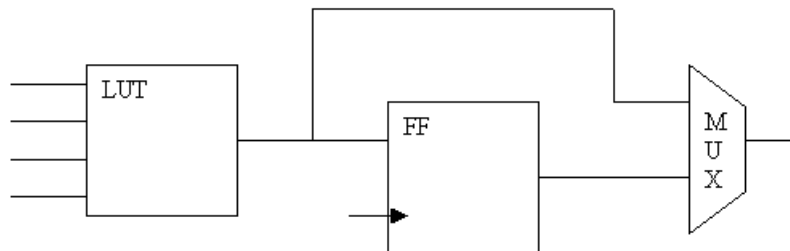
## 2.2. Benchmark FPGA Architecture

The FPGA architecture used for testing is an island style FPGA. It's a square array of logic blocks with a channel of routing tracks among every row and column of logic blocks. All the routing channels have the same number of tracks. At the perimeter of the array there are the pads; there are two pads at the end of each row and column of logic blocks. In Figure 13 there is a 4X4 instance of the architecture.



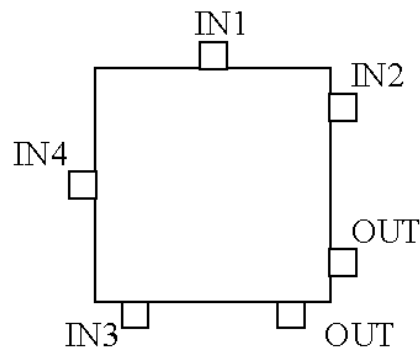
**Figure 13: A 4x4 instance of the benchmark FPGA**

The Configurable Logic Block (CLB) has a four input – one output lookup table (LUT) and a Flip Flop (FF) as shown in Figure 14



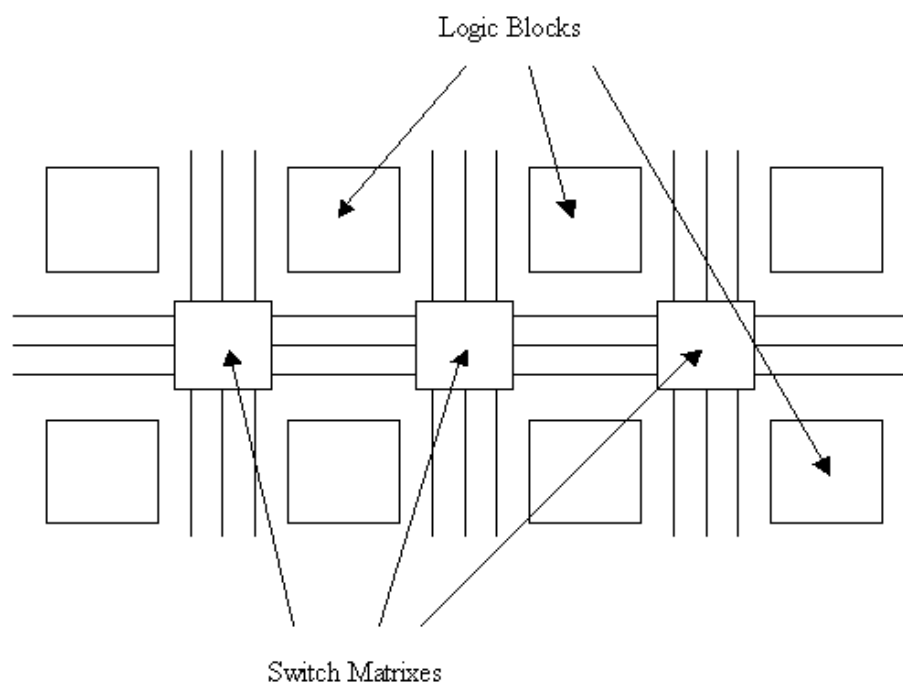
**Figure 14: A configurable logic block**

The location of the input and output pins are shown in Figure 15. As you can see there is no indication about the location of the clock pin. This is due to the fact that VPR doesn't consider the clock signal during routing because it assumes that dedicated routing resources will be used for propagating the clock (and this is the trend in commercial FPGA).



**Figure 15: Pin locations**

Each pin of the CLB can be connected to all the adjacent tracks. Each input and output pad can be connected to all the adjacent tracks. There is only one type of wire segment with length of one logic block. Each segment starts from a switch matrix and ends in the adjacent switch matrix (Figure 16).



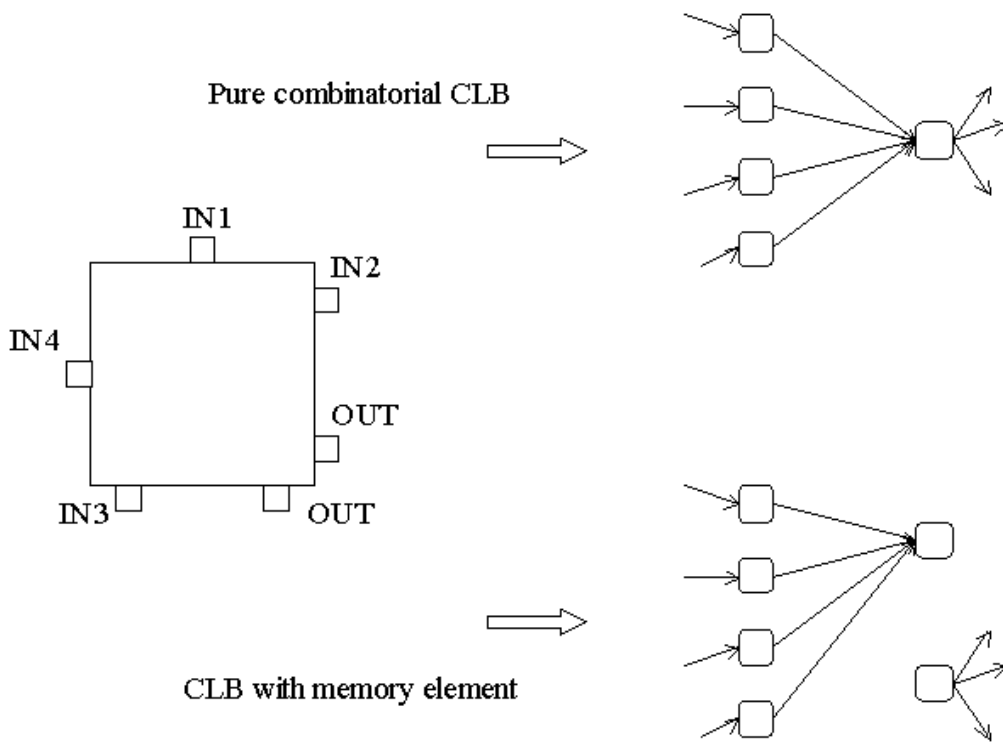
**Figure 16: Routing structure**

When a track enters in a switch matrix there are three switches that connect the track to other three tracks in the adjacent channels. This means that the signal on track number  $j$  of a channel can only be routed toward the track number  $j$  of another channel. Using the standard FPGA terminology it's an i-to-i switch matrix. All the switches of the FPGA are implemented using tri-state buffers in order to make the timing analysis easier.

### 2.3. Static Timing Analysis

Static timing analysis is the procedure that figures out the maximum speed of a digital circuit without simulating it. From this point of view timing analysis is just a tool for the final verification of a design. However, in timing driven layout it's fundamental to know during the development of the layout the current timing behavior of the circuit. Thus, the timing analyzer should produce as result not only the estimated maximum delay of the circuit, but also other information (for instance the time in which the input signals of a gate are all stable or the maximum delay from a gate to the memory elements or pads of the circuit or the timing criticality of a net or of a gate).

The basic data structure of a timing analyzer is the timing graph. There are many possibilities for the structure of the timing graph. One of the most common is to represent the logic elements of the circuit as the nodes of the graph and the pin-to-pin interconnections as the edges of the graph. In this case to each edge is assigned the delay of the corresponding interconnection and to each node is assigned the delay of the corresponding gate [13]. Another possibility is to assign to each net a node and the edges of the nets represents the interconnections of each net with the driver gate of the other nets [9]. In this case a delay is associated only to the nodes. In fact each net is assumed to have the same delay on all the pins and the delay of a gate is added to the delay of the driven net. The scheme that we will adopt is very similar to that proposed by the authors of VPR [21]. To each pin and to each pad and to each memory element is assigned a node. The edges represent the functional interconnections among the pins. Thus, a combinatorial logic block is represented as a sub-graph with a number of nodes equal to the number of active pins of the block; from each input pin starts an edge toward the output pin; from the output pin starts one edge toward all the pins of the driven net. A sequential logic block is represented in the same way, but the edges from the input pins are directed toward the node corresponding to the memory element. In Figure 17 you can see the possible schemes of the logic blocks in the timing graph.



**Figure 17: From logic blocks to timing nodes**

The only nodes without any incoming edge correspond to the output pins of the memory elements and to the input pads of the FPGA. The only nodes without any outgoing edge correspond to the memory elements and to the output pads. Due to the fact that one of the basic rules of design of digital circuits is to avoid combinatorial cycles, and because the timing graph decouples input and output of the memory elements, loops in the timing graph are impossible. Thus, the timing graph is a Directed Acyclic Graph (DAG). The sources of the DAG are the nodes without any incoming edge (input pads and memory elements) and the sinks of the DAG are the nodes without any outgoing edge (output pads and outputs of memory elements). To each edge of the graph is assigned a weight that is the delay between the pins corresponding to

source and destination of the edge. If the two pins belong to the same block it's a combinatorial delay. Otherwise it's an interconnection delay.

The arrival time of a block is the time in which all its input signals are stable. The required arrival time of a gate is the arrival time that is required for that gate in order to not increase the maximum delay of the circuit. The downstream delay of a gate is the maximum delay over all the paths that from that gate go to a memory element or to an output of the circuit. The slack of an interconnection or of a gate is the delay that you can add to that interconnection or gate without increasing the maximum delay of the circuit. On the timing graph these values are:

$$T_{arrival}(i) = \max_{j \in in\_edges(i)} \{delay(j, i) + T_{arrival}(j)\}$$

$$T_{downstream}(i) = \max_{j \in out\_edges(i)} \{delay(i, j) + T_{downstream}(j)\}$$

$$T_{required}(i) = MaximumDelay - T_{downstream}(i)$$

$$Slack(i, j) = T_{arrival}(i) - T_{required}(j) - delay(i, j)$$

Of course the arrival time of a source node of the timing graph is 0 and the same for the downstream delay of a sink node. In order to figure out the other values the timing analyzer visits the nodes of the graph twice with a visit in topological order. During the first visit the algorithm computes the arrival times that at the beginning are initialized to 0 starting from the sources of the graph. At the end of the first traversal, the biggest arrival time is the maximum delay of the circuit. The second visit computes the downstream times and the required arrival times of the nodes and the slacks of the edges. At the end of the second traversal the edges with the most critical ones and those with a slack equal to 0 form the critical paths. It's obvious that,

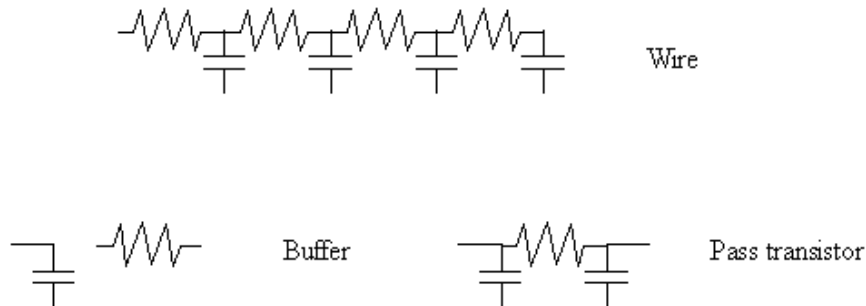
if the delays on the edges of the DAG are known, the complexity of the timing analyzer is linear.

## 2.4. Delay Model

In FPGA the delay of the logic blocks is exactly known at the netlist level. In fact the logic functions are implemented using lookup tables and the output of the logic block is decoupled from the load by buffers. The problem is the interconnection delay. In fact, the delay on a pin-to-pin interconnection is a function of several parameters: distance between the two pins, fanout of the net, presence, position and size of buffers, geometry of the net tree, capacitance and resistance of the wires, capacitive load of the source pin and others. In FPGA layout the problem is easier because of the regularity of the architecture. In FPGA position of buffers and sizing of pass transistors and wires and capacitive load of pins are known from the description of the target FPGA.

In order to calculate the value of the pin-to-pin delay, the most reliable way is the simulation of the circuit using SPICE (Simulation Program Integrated Circuits Especially), a circuit simulation program developed at the University of California, Berkley. This kind of simulation is very computationally expensive and requires a huge amount of CPU time for the simulation of very small circuits. With a modern digital circuit it's absolutely inapplicable. The most common delay model in recent works is the Elmore delay model [24]. The Elmore delay is a technique that evaluates the delay of the RC trees. Thus, in order to use it with VLSI interconnection you have to approximate the various elements of an interconnection as resistances and capacitances. Buffers are transformed in output resistances and input capacitances with a fixed delay; pass transistors are modeled with input and output capacitances

and resistances, wires as RC ladders, drivers as resistances and input pins as capacitances (Figure 18).



**Figure 18: Circuit elements modeled as RC ladders**

The Elmore delay between the node  $i$  and the node  $j$  of an RC tree is defined as ([25]):

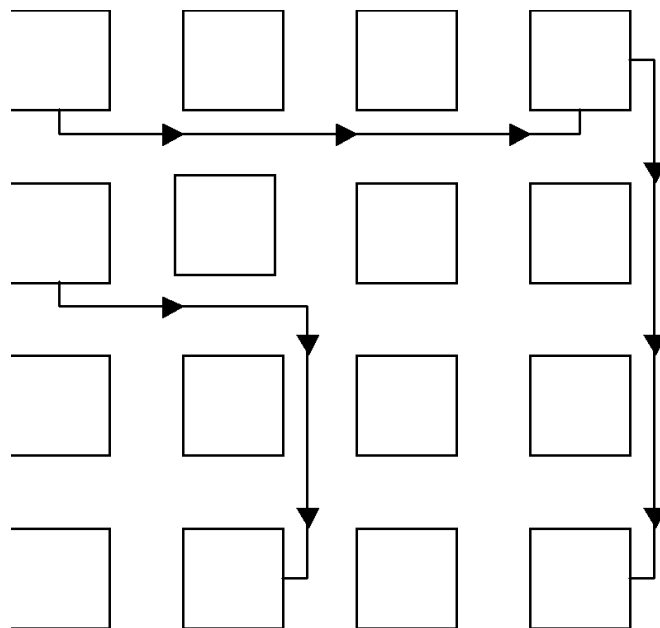
$$T(i, j) = \sum_{k \in \text{path}(i, j)} R(k)C(\text{Tree}(k))$$

Where  $\text{path}(i, j)$  is the set of resistors belonging to the path between  $i$  and  $j$ , and  $C(\text{Tree}(k))$  is the total capacitance of the tree rooted at  $k$ .

Using the Elmore delay you can calculate with good approximation the delays between buffer and buffer on an interconnection and, from these delays and the intrinsic delays of the buffers, the delay of a pin-to-pin interconnection. The problem is that you need to know the tree structure of the net, and the value of resistance and capacitance of each single segment. The values of resistance and capacitance are known from the description of the target architecture. The exact tree structure of a net is unknown until the end of the routing phase. However, routing the circuit after every placement modification in order to update the delays would be too expensive in terms of computation time. Thus our approximation is to consider each pin-to-pin interconnection individually. The delay of the interconnection is the delay of the monotone path spanning between the two pins. This assumption with a general FPGA architecture would be

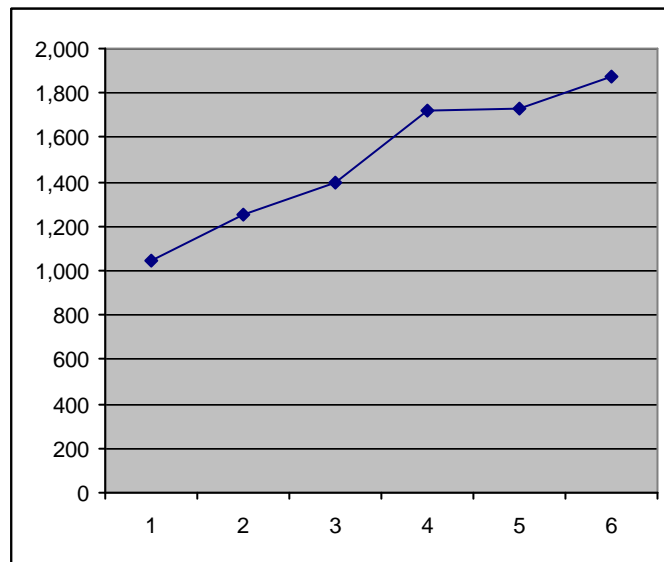
completely arbitrary. However, with our particular target architecture it's not very imprecise. In fact, due to the fact that all the switches are buffers, the delay on a pin is completely decoupled by the fanout of its net. In fact buffers isolate the capacitances at their outputs introducing just a constant capacitance. Moreover, if the fanout of the net is small and the FPGA is not very congested, the router is usually able to route a monotone path between two pins.

Using the Elmore delay it's easy to demonstrate that the delay on this kind of path is a linear function of the Manhattan distance between the two pins. In fact the buffered switches decouple the capacitances on the single length wire segments and linearizes the total delay. Thus the delay becomes a function of the number of switches on the path that as you can see from Figure 19 is a linear function of the Manhattan distance between source and destination (obviously with some inaccuracy due to the fact that the position of the pins is approximated as the center of the CLB). In Figure 19 there are 3 paths with the same Manhattan length and the same number of buffers.



**Figure 19: Manhattan distance and number of crossed switches**

Moreover this approximation seems to capture the trend of modern FPGA architecture to linearize the delay. In fact in Figure 20 you can see a graph with data about the delay in a Xilinx Virtex FPGA extracted by experiments with the timing analyzer provided by Xilinx. On the X-axis there is the Manhattan length in terms of logic blocks of the interconnection and on the Y-axis the delay in ns of the interconnection. As you can see the delay is in first approximation a linear function of the Manhattan distance.



**Figure 20: Delay as function of Manhattan distance in Xilinx Virtex FPGAs**

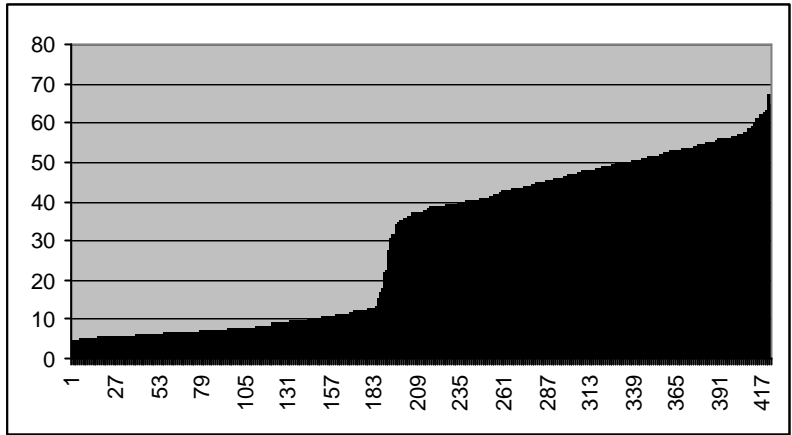
## CHAPTER 3

### A Path Based Algorithm

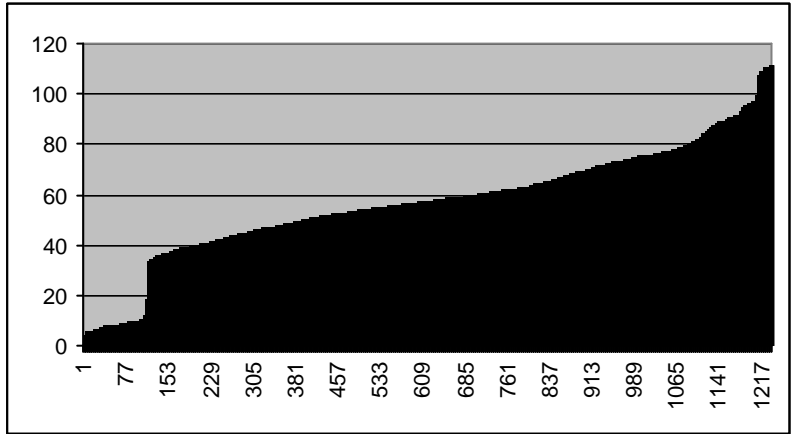
#### 3.1. Preliminary Analyses

The purpose of our algorithm is, starting from an existing placement of a circuit, exploiting the logic replication of a small amount of combinatorial CLBs in order to improve the clock period. The first necessary task for the design of such an algorithm is to analyze the characteristics of the most critical paths of some circuits in order to get indications about how to improve their performance.

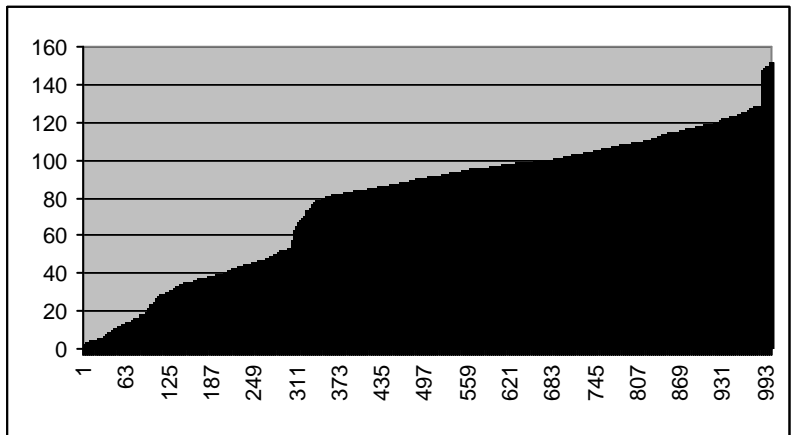
The first indications come from the distribution of the arrival times at outputs and memory elements of the FPGA. In Figure 21, Figure 22 and Figure 23 you can see them for the benchmark circuits Bigkey (Figure 21), Elliptic (Figure 22) and Frisc (Figure 23). On the X-axis of the graph there are the outputs and the memory elements of the circuits (sinks of the timing graph) ordered from the fastest to the slower; on the Y-axis the corresponding arrival times. As you can see, the sinks can roughly be grouped in three sets whose elements have similar arrival times: the first set, characterized by the best timing behavior, is composed by the most part of the output pads and some memory elements; the second set is composed by most of the memory elements and the remaining output pads; the last set contains the memory elements that actually slow down the circuit. The last set is the smallest one and is the most interesting one from the point of view of the timing optimization of the circuit.



**Figure 21: Sink delays of Bigkey**



**Figure 22: Sink delays of Elliptic**



**Figure 23: Sink delays of Frisc**

The consequence of this distribution is that optimization of a small number of critical sinks can greatly improve the performance of the circuit. If the distribution was continuous, after you have improved the performance of the worst sink, the performance would be bounded by the delay on the next critical sink that is just slightly better than that you have improved. With a segmented distribution of the delays, after you have improved the delays of the worst segment, you can observe a jump on the performance improvement due to the fact that the next critical sink belongs to the next set. Moreover, with a uniform distribution of delays, you couldn't improve the critical path by penalizing the other paths. In fact, in such a situation all the paths are near critical and penalizing one path leads to the creation of a new critical path.

Another interesting consideration comes from the analysis of the criticality of the modules of the FPGA. We define criticality of a node  $i$  as:

$$Criticality(i) = 1 - \frac{Slack(i)}{Critical\_Path\_Delay}$$

The maximum criticality is 1 and corresponds to the nodes of the critical path. If you consider the most critical nodes (those with criticality greater than 0.9) you can find that they constitute a very small portion of the circuit. In fact, you can see from Table 1 that the most critical modules are on average only the 5% of the modules.

Let the critical sub circuit be the set of modules with a criticality greater than some value [26][9]. It's easy to demonstrate that its translation in timing graph is a set of connected DAG. From each connected DAG you can extract a set of nodes that, if deleted from the DAG, makes the DAG unconnected (Proper Node Separator with the terminology of [9]). If you can reduce the delay on outgoing or incoming edges of the proper node separator you can improve the timing performance of all the connected DAG that you are considering. It's obvious that if the number of connected DAG is bigger, the optimization of the circuit is more difficult because involves the optimization of many independent proper node separators. On the contrary, if there is just one connected DAG, optimizing the performance is much easier. The analysis of the sub circuits constituted by the most critical modules shows that the most usual structure is

composed by many sources and many sinks with a small amount of possible intermediate sub paths. The consequence is that the number of connected DAGs that constitute the critical sub graph is usually small and it's usually possible to find proper node separators of just one node or at least with a very small number of nodes.

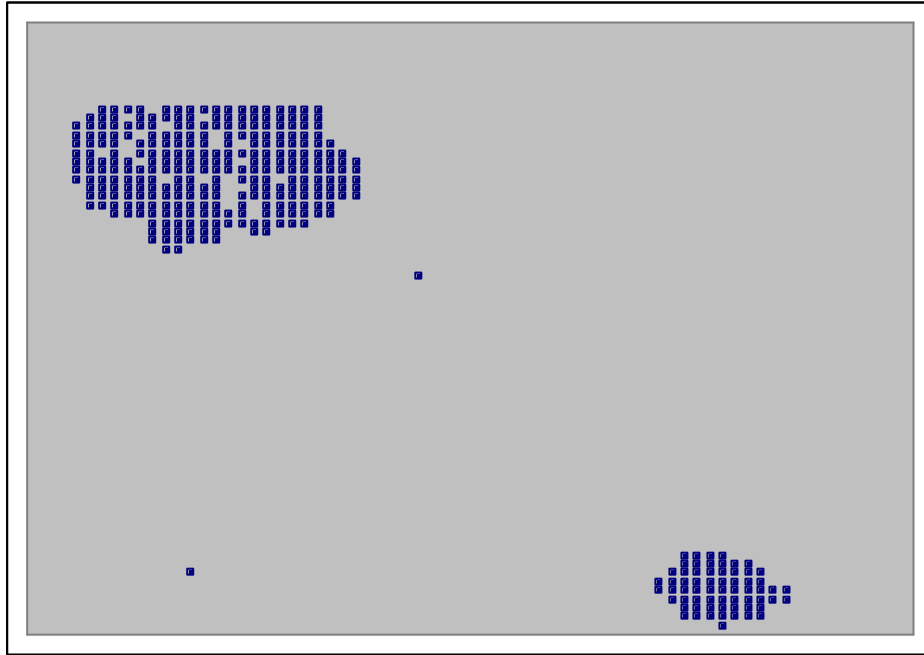
Circuit	Number of most critical nodes	Total number of modules	Percentage of most critical modules
frisc	342	3692	9.263272
spla	68	3752	1.812367
s298	186	1941	9.582689
elliptic	189	3849	4.910366
s38417	150	6541	2.293227
des	176	2092	8.413002
bigkey	13	2133	0.60947
Average			5.269199

**Table 1: Number of modules with criticality greater than 0.9**

Our path based timing optimizer iteratively chooses from the current critical DAG a node that is very likely to be a proper node separator and that can be effectively optimized enforcing monotonicity to all the sub paths flowing through it. Our algorithm optimizes its position using either relocation or replication.

Another interesting consideration comes from the observation of the locations of the nodes of the critical sub circuits. In fact they are usually clustered in a few regions. In Figure 24 there is the benchmark circuit Frisc placed on FPGA. The points represent the modules with a criticality greater than 0.9. They are very close to each other and mainly clustered in two highly congested zones. Of course optimization in that zones is very difficult because there are no empty slots where I can put a replicated cell. If you want to create room for a cell, you have to

relocate some critical cells with the possibility of deteriorating the timing performance of the circuit.



**Figure 24: Positions of the nodes with criticality greater than 0.9 in Frisc**

### 3.2. Overview of the Algorithm

The algorithm, starting from an initial placement, iteratively generates a perturbation to the placement trying to relocate a cell on the critical path in order to enforce monotonicity on the sub paths flowing through it. The algorithm chooses the new position for the cell taking into consideration only the timing constraints and not the physical constraints of the placement (impossibility of cell overlapping). If the timing behavior of the circuit is better by replicating the cell instead of just relocating it, the algorithm replicates the cell otherwise it

just move the cell. Thus at the end of this first step the placement could be illegal due to the overlapping of the replicated (or relocated) cell on another cell. In order to solve this overlapping the algorithm creates the room for the new cell with an iterative procedure that tries to induce very small modifications to the placement. In

Figure 25 there is the pseudo code of the algorithm. This pseudo code is very general and can be adapted in order to obtain many possible variations of the algorithm. The version that is implemented and tested is a kind of greedy algorithm. In fact, the current algorithm terminates when it fails getting an improvement for a certain number of iterations in row; the only accepted modifications are those that lead to a timing improvement of the circuit and the modification are evaluated step by step.

```

1  While (exit_criteria() )
2      Network analysis ()
3      Save current situation()
4      Timing constraints satisfied=FALSE
5      While ( (cell selection is possible)&&( Timing constraints satisfied==FALSE) )
6          Seleted cell=cell selection()
7          While (possible to generate timing constraints)
8              Target clock= timing constraints generation()
9              While (possible to select a slot)
10                 Selected slot=slot selection()
11                 If (timing constraint satisfied (Seleted cell,
12                     Selected slot, Target clock) )
13                     Timing constraints satisfied=TRUE
14             If (Timing constraints satisfied)
15                 New cell=cell generation(Seleted cell, Selected slot)
16                 Fanout partitioning (New cell, Seleted cell )
17                 If (placment is illegal)
18                     Legalize(Selected slot)
19                 Modification discarded=check placement
20                 If (Modification discarded)
21                     Restore last saved situation()

```

**Figure 25: Pseudo code of the algorithm**

Of course, due to the structure of the algorithm, it's easy to adapt it to a simulated annealing algorithm. In fact, the behavior of the iterative routine that tries to exploit the replication of a cell is not completely deterministic. Thus, you can substitute the greedy criteria of evaluation of the modifications with a probabilistic one where the probability that a modification that makes the placement worse is accepted is function of the induced degradation and of the current temperature.

$$P = e^{-\Delta C / T}$$

You can also, during the initial iterations, expand the dimension of the perturbations produced by the algorithm by checking the placement only after some iteration and then, step-by-step, you can shrink the dimension of the perturbation by checking more and more often. Of course you have to decide a schedule for the behavior of temperature and perturbation dimension.

This simulated annealing version would be probably more effective than the current greedy version, but the implementation and testing of a simulated annealing schedule is outside the purposes of this work that wants just to investigate the possibilities of timing driven logic replication.

### 3.3. Network Analysis and Cell Selection

This section is about the first part of the iterative loop, the part in which the cell that will be considered for relocation and timing optimization is selected. The purpose of this section is to find a cell whose relocation can induce a good gain on the circuit performance. The technique that will be applied by the algorithm will be to enforce monotonicity on the critical sub path; thus, the first idea is to figure out which are the nodes that determine non-monotone sub paths. A measure of how much the relocation of a node of the critical path can improve the

delay of the critical path is its deviation from the critical sub path. Where the deviation of a node  $i$  is:

$$deviation(i) = dist(prev(i), i) + dist(i, next(i)) - dist(prev(i), next(i))$$

Of course the distance is intended as Manhattan distance and next ( $i$ ) and prev ( $i$ ) are the next and the preceding nodes on the critical path. This is the measure of how much a node is outside of its monotone region. Due to the fact that in our FPGA and with our delay model, the delay is a linear function of the distance, the improvement that you can get on the critical path is a linear function of how much you can put the node closer to its monotone region.

In order to choose the node, the algorithm evaluates for each node of the critical path its deviation from the monotone region. Then a vector of random values varying from 0 to 1 multiplies the vector of the deviations of the nodes. Finally the vector is sorted by deviation from the biggest one to the smallest one. The selection of the cells will follow the order induced by the sorted vector.

In Figure 26 you can see an example of a path and three monotone regions. The vector of the deviations will be something like:

(2, 0, 4, 2, 6, 0)

The vector of the nodes:

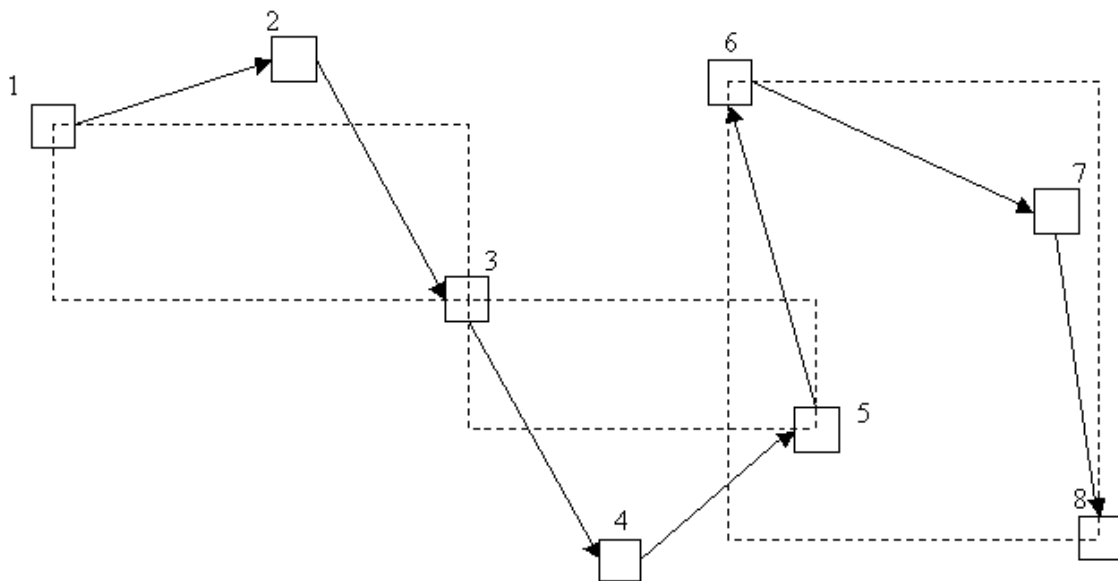
(2, 3, 4, 5, 6, 7)

The random vector:

(0.845228514, 0.326446948, 0.246800495, 0.038992786, 0.159481936, 0.483416546)

The resulting vector:

(1.690457028, 0, 0.987201982, 0.077985573, 0.956891614, 0)



**Figure 26: A path and three monotone regions**

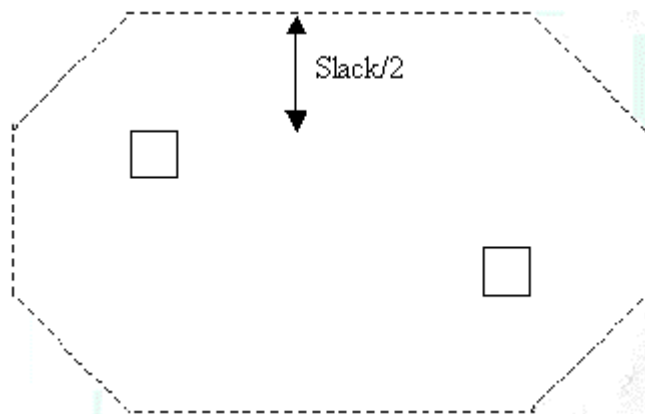
The purpose of the multiplication by the random vector is to give to the algorithm the possibility of improving the performance if it fails with the apparently best possible node (the one with the biggest deviation). The experimental results, as we will see, confirm that the randomized vector works better than the not randomized one.

### 3.4. Timing Constraints Generation and Slot Selection

The purpose of this section of the algorithm is to figure out which is the best position for the new cell in order to obtain the maximum performance gain on the critical path avoiding that other paths flowing through the cell become more critical than the current critical path. The adopted routine is: generate a target clock; if, by relocation or replication of the selected cell, you can generate a new placement able to respect that maximum delay on all the paths flowing through the selected, accept that configuration; otherwise increase the target clock. The

generated placements can violate the physical constraints because the legalizer will enforce them. If the procedure can't find a slot with a target clock better than the current clock, another cell is selected and the routine starts again.

If you consider a cell, a path flowing through it and a target clock, it's possible to define a region in which you can put the cell so that the delay on the path is less than the target clock. That's the feasibility region of that cell from the timing point of view. That region is the set of slots whose sum of the delay from the preceding and to the next node on the path is less than the current delay between preceding and next node plus the slack of the path with respect to that target clock. Due to the fact that delays are linear function of Manhattan distances, that region is the region inside the ellipsis (in Manhattan geometry) with as focuses preceding and next node. Of course if you select a target clock that induces a negative slack on the path, it's possible that the region becomes of 0 area. In Figure 27 there is an example of feasibility region. When you want to impose a target clock to a cell, you are forcing that cell to be put inside the intersection of the feasibility regions imposed by all the paths flowing through it.



**Figure 27: Feasibility region**

Let's consider a general cell with multiple fanin and multiple fanout. The current position of the cell is somewhere inside the intersection of the feasibility regions imposed by the

current clock. If you choose as target clock a value less than the current target clock, there is no reason why a region that is common to all the new feasibility regions exists. In fact each possible path flowing through the cell imposes a feasibility region. The number of possible feasibility regions is:

$$num\_feasibility\_region(i) = num\_node(fanin(i)) * num\_nodes(fanou(i))$$

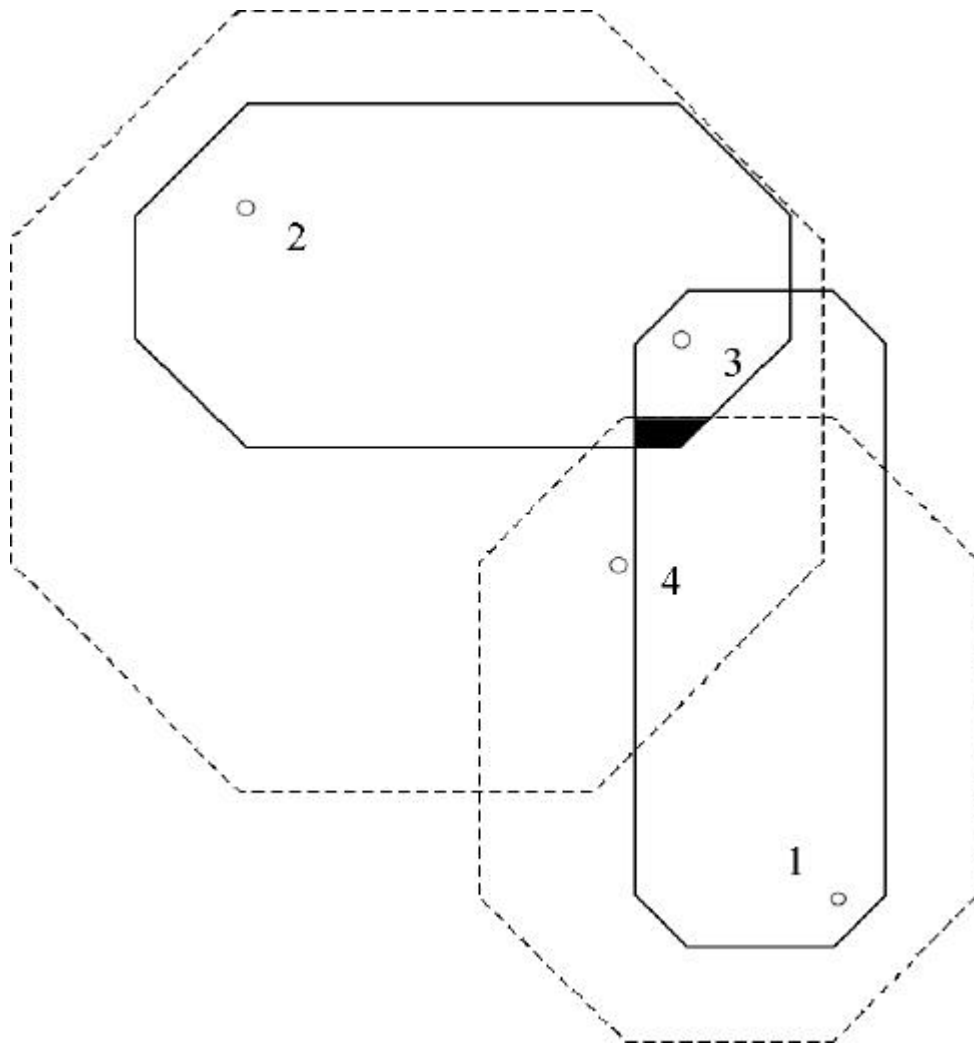
If you don't put any constraint to the feasibility regions you have no guarantee that these regions have a common intersection. In Figure 28 there is a possible situation. The cell that we are considering does not appear. The cell is a cell with two inputs (from nodes 1 and 2) and 2 outputs (to nodes 3 and 4). The feasibility regions induced by the two inputs with the node 4 are those inside the dotted lines. The others are inside the continuous lines. The cell is currently somewhere inside the black zone that is the intersection of all the feasibility regions. If you shrink just a little the feasibility regions by reducing the target clock, the common region disappears and there is no way to place the cell so that the target clock is respected.

Let's consider a cell with multiple fanin and fanout equal to one. The situation is similar to that described above; the difference is that, if you choose as target clock a value that is less than the current clock but at least equal to the delay on the critical path if the critical sub path flowing through that node was monotone, a region that is common to all the new feasibility regions exist. In the worst case in fact the next node on the critical path constitutes it.

Let's go back to the situation of multiple fanin and multiple fanout. If you can duplicate the selected node, you can place one node in the region induced by the node 4 and the other in the region induced by the node 3. In a general situation we can consider the decoupled regions induced by each output node of the selected cell.

That's exactly what our algorithm will try to exploit. When you begin to reduce the clock, the feasibility regions begin to shrink; at a certain point the region induced by the most critical output node becomes separated from the others. At this point, it's impossible to reach the target clock by just relocate the node. The only way to reduce the clock is to replicate the

cell and to put it in the feasibility region induced by the most critical target node. If you go on reducing the target clock, the number of unconnected regions grows. In order to improve the delay in such a situation the logic duplication is not enough and you need multiple replications. Our algorithm imposes a target clock and check if there is position where I can place a new cell in order to respect that clock for all the paths flowing through it in the direction of the most critical node. If there exists it places the new cell, otherwise it increases the target clock. The fanout-partitioning routine decides if the old cell is necessary or not.

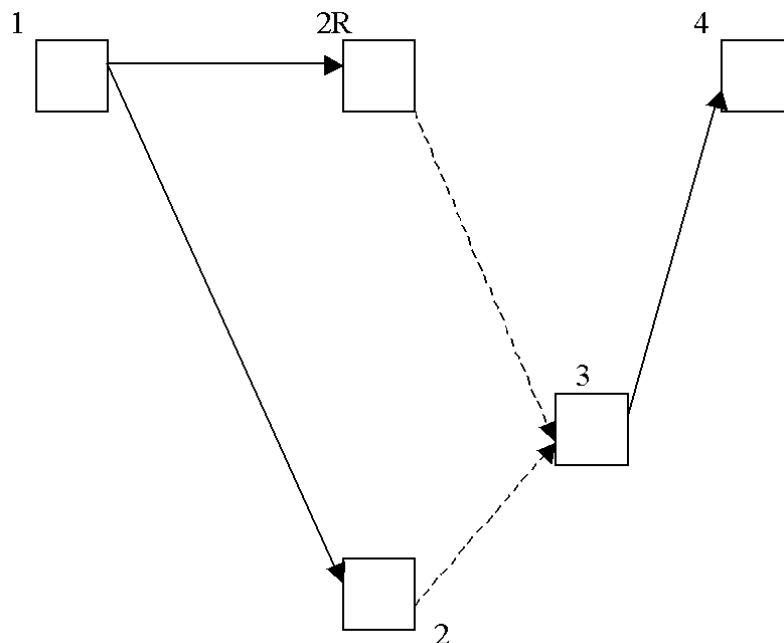


**Figure 28: Four feasibility regions with a common intersection**

### 3.5. Fanout Partitioning

Fanout partitioning is the operation of choosing for each node of the fanout of the nets that start from the cell that is considered for replication if it will be connected to the original block or to the replicated one.

A first simple approach could be to connect the blocks that are closer to the replicated cell to the replicated one and those that are closer to the old cell to the old one. This approach is the wire length driven approach. However this approach leads to a local wire length improvement but can lead to timing performance and global wire length degradation. In fact look at the example in Figure 29. The dotted edges represent the two possible pin-to-pin connections. If you connect the cell 3 to the cell 2 you are reducing the wire length on that pin-to-pin interconnection; however, the path 1-2-3-4 is longer than the path 1-2R-3-4. Thus you are inducing degradation both in timing performance and in global wire length.



**Figure 29: Failure scenario for wire length driver fanout partitioning**

Our algorithm takes into account such a situation. It checks which is the best timing configuration for each pin-to-pin interconnection and select the best one. If at the end the old cell has 0 fanout, it is deleted. There are also other possible approaches that are suitable in order to not deteriorate the timing behavior. For example, in order to reduce the number of duplicated cell, you can force the interconnections that can be put on the replicated cell without violating the timing constraints also if they make a path slower. However, this approach doesn't seem very effective especially on large fanout where it's impossible to force a 0 fanout on the old cell without violating the timing constraints. In that case in fact you are deteriorating the timing behavior of not critical paths without getting the elimination of the old cell. Another possible approach could be trying to maintain a small fanout for the net containing the most critical interconnections in order to make easier for the router to route the tree of the nets belonging to the most critical paths. This approach seems to be interesting but it's not tested in this work.

### 3.6. Legalizer

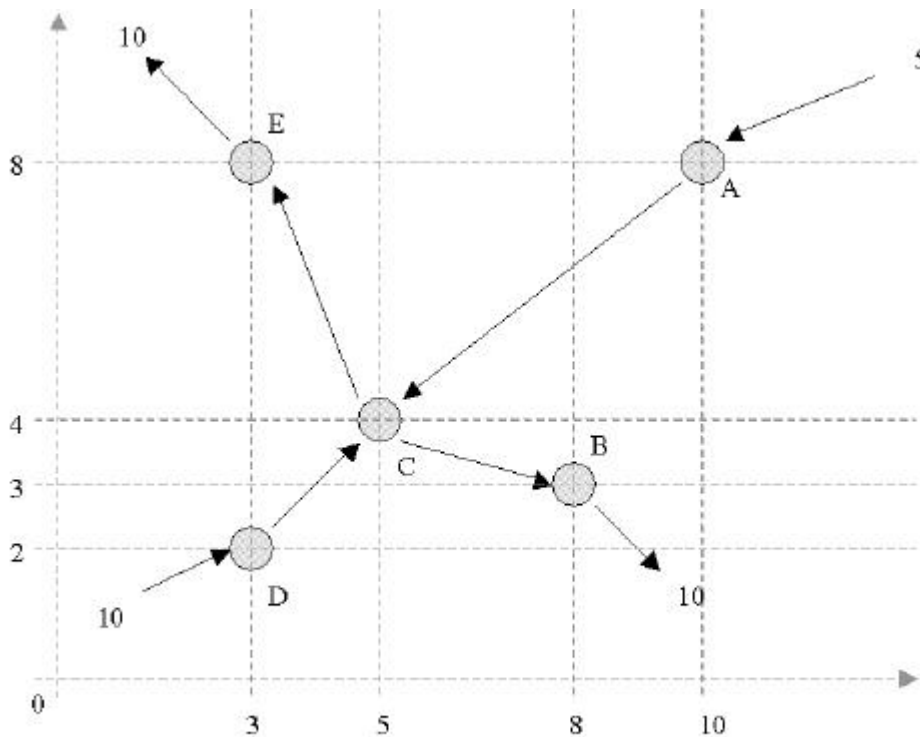
The purpose of the legalizer is, starting from an illegal placement (with overlap of modules), obtaining a placement without overlap. The perfect timing driven legalizer is able to solve the overlap of the modules without violating the timing constraints. Such a legalizer does not exist. In fact it's impossible to guarantee for each illegal placement a legal one because legalization could always induce deterioration of near critical paths. If you consider that the critical nodes are usually very close to each other (see Figure 24), the degradation of near critical paths seems to be a common situation during legalization. However due to the fact that a displacement of just one cell can't heavily affect the timing behavior of the circuit, a wire length driven legalizer that performs a minimum amount of swaps of adjacent cells is good for

legalization in a timing optimization context. A possible algorithm is: find the nearest empty cell to the overlapping module; compute for all the cells in the rectangle that has as vertices the overlapping module and the nearest empty slot the wire length gains of the two possible displacements in the direction of the empty slot; build a graph in which each cell of the rectangle is a node and there is an edge between two nodes if they are adjacent and the destination is in the direction of the empty slot (the weight of the edge is the gain calculated above); the graph is a DAG and the path with the maximum cost is the reconfiguration path with the maximum wire length gain.

### 3.7. An Example

Let's examine an example with a couple of run of the algorithm. In Figure 30 you can see an example of a possible situation in which a timing optimizer without the freedom degree of logic replication can't do anything. Let's see which is the behavior of our algorithm. The delays are the distances in terms of CLBs. Logic delay is not taken into consideration. Through the cell four paths flow: A-C-B, D-C-B, A-C-E and D-C-E. The cell A has an arrival time of 5; D has an arrival time of 10; B and E have down stream delays of 10; the delay among A and C is 9; the delay among D and C is 4; the delay among C and B is 4; the delay among C and E is 6. Thus the total delay of the 4 paths is for A-C-B 28, for D-C-B 28, for A-C-E 30 and for D-C-E 30. Let's suppose that 30 is the current maximum delay through the circuit. The cell C is very critical. In fact, if I want to reduce the delay on the two critical paths (A-C-E and D-C-E) I have to move the cell up and on the left. However, if I move it up of one position, I penalize of 2 the path A-C-E that from 28 gets a delay of 30 and becomes the new critical path. The same if I move the cell on the left of one position because of the path A-C-B that gets a delay of 30. from the point of view of the feasibility regions the cell C is at the intersection of the four feasibility

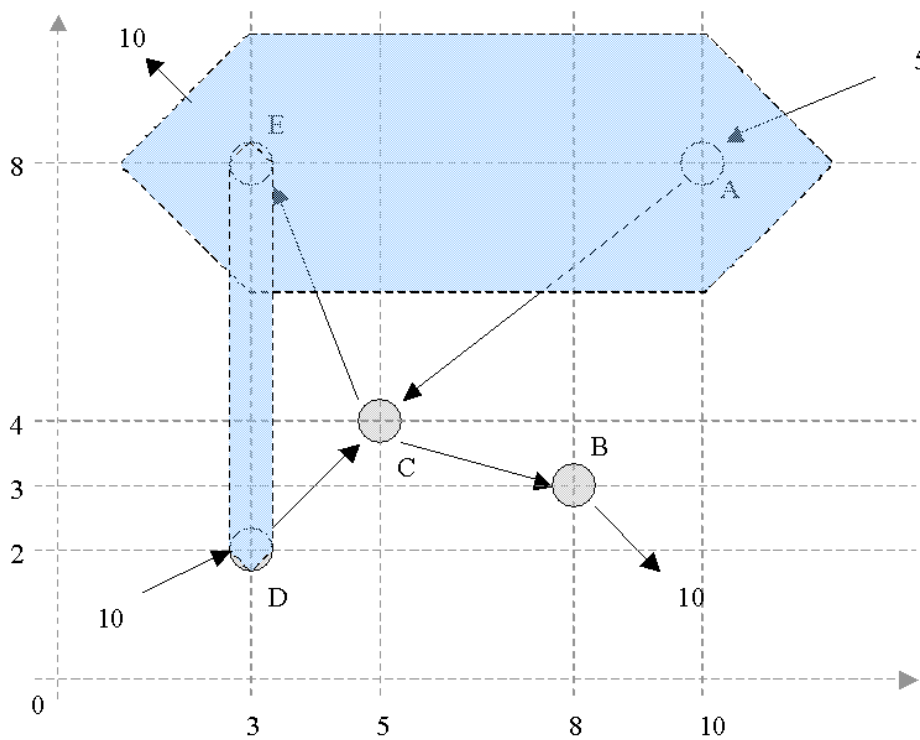
regions induced by the four paths. But if you try to reduce the clock, there are two separated feasibility regions. One induced by the paths A-C-E and D-C-E and one induced by the other two paths.



**Figure 30: The four sub paths**

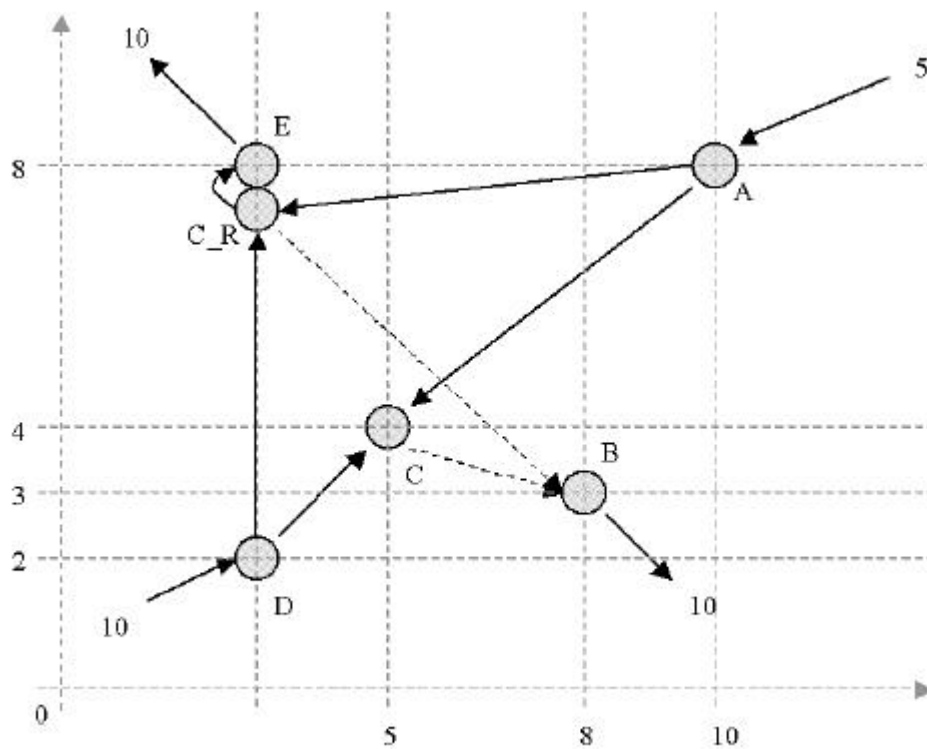
Our algorithm follows one of the two critical paths (let's suppose A-C-E) and considers the cell with the bigger deviation from his feasibility region. Let's suppose that we are using the deterministic version of the algorithm. in this case it chooses to optimize the cell C. First of all it generates the timing constraints and chooses as target clock the better target clock that can be reached on the critical path imposing monotonicity to the sub path flowing through C. in this case is the downstream delay of E (10) plus the arrival time at A (5) plus the distance between A and E (7). The total is 22. The algorithm looks for a position in which it's possible to place C in

order to get a maximum delay on all the paths flowing through C toward E of 22. It doesn't exist because the minimum possible delay on the path D-C-E is 26. Thus it begins to relax the timing constraints until the target clock becomes 26. With this target clock the feasibility regions become those in Figure 31. As you can see an intersection exists and are the two slots (3,6) and (3,7). The algorithm put the new cell in the slot (3,7) because it's closer to the ideal position imposed by the critical path that is in the center of the feasibility region imposed by A-C-E.



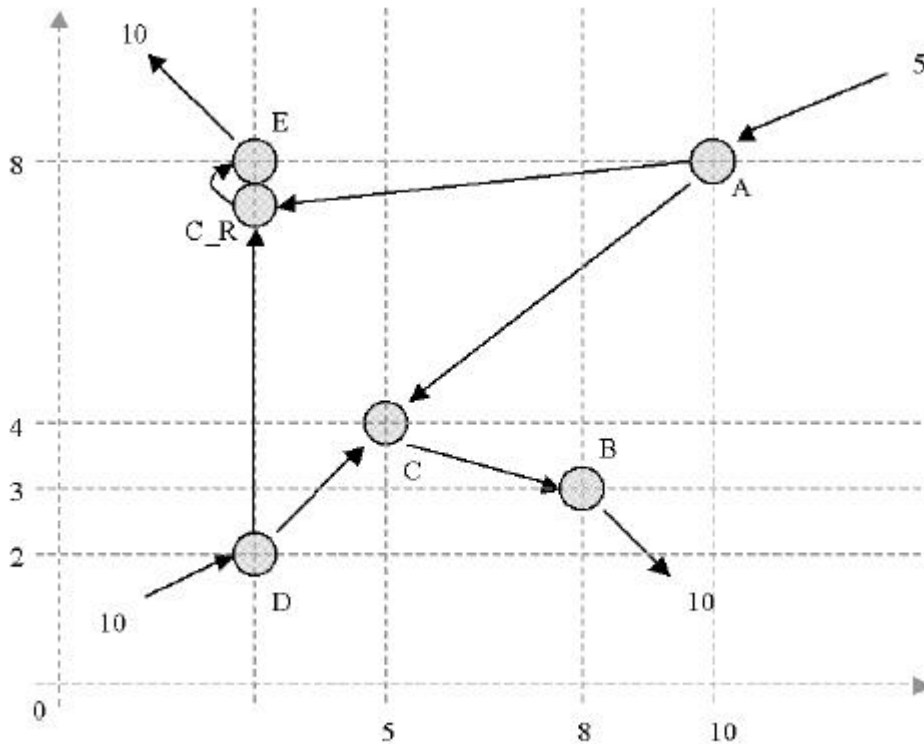
**Figure 31: Feasibility regions with respect to the node E**

After the replication of the cell C we have the situation of Figure 32. Now the algorithm has to decide if the edge C-B (the dotted one) must be connected to the replicated cell C\_R (and this would kill the cell C) or to the original cell C.



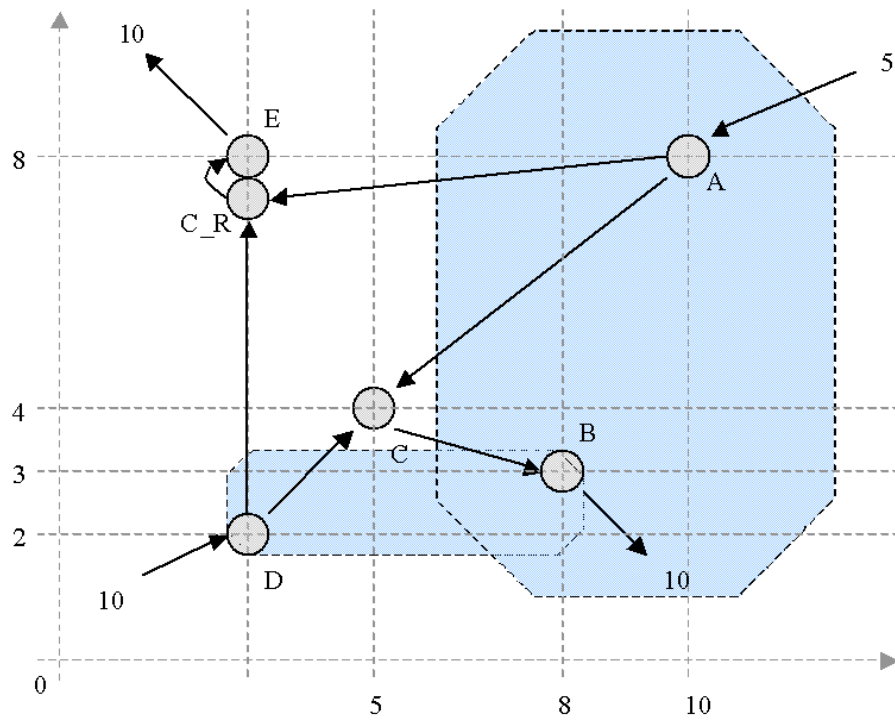
**Figure 32: Replication of cell C**

The choice of the algorithm is to connect the cell B to the original cell because the worst path flowing through has a delay of 28 while with the other connection it would have a delay of 34. After the legalization, the situation that the algorithm finds at the next iteration is in Figure 33.

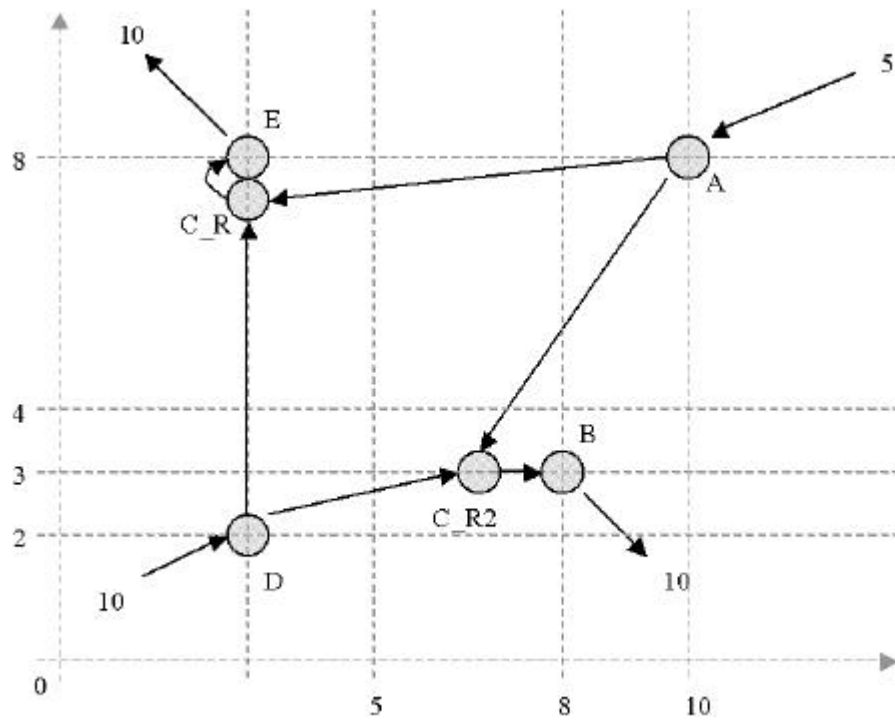


**Figure 33: Situation at the beginning of the second iteration**

There are again two critical paths: A-C-B and D-C-B. Both of them have a delay of 28. Let's suppose the algorithm chooses to consider D-C-B. It finds a deviation of 2 on the node C. It tries to find a new position for the cell C. It imposes as target clock the minimum possible for the selected critical path that is the arrival time at D (10) plus the distance between D and B (6) plus the downstream delay from B (10). The total is 26. Thus the algorithm searches a feasible cell for a target clock of 26. With such target clock the feasibility regions are in Figure 34. As you can see the intersection exist and is constituted by the slots (3,6), (3,7), (2,7) and (2,8). The algorithm chooses the slot (3,6) because it's the closest to the ideal position imposed by the critical path that it's considered (D-C-B). The algorithm places the cell C in (3,6) and finds the situation of Figure 35. Of course there is no need of fanout partitioning because the fanout of the cell is just one.



**Figure 34: Feasibility regions with respect to node B**



**Figure 35: Final situation**

The situation in Figure 35 is very hard to improve. In fact we have two critical path of length 26 that are already monotone (D-C\_R2-B and D-C\_R-E) and one that is not monotone and can be improved by moving to the left the cell C\_R2. However the algorithm doesn't move the cell in that direction because there is no possibility of actual timing improvement.

## CHAPTER 4

### Experimental Results

#### 4.1. Timing Performance Improvement

In this section we compare the speed of 20 MCNC benchmark circuits placed using VPR and with that of the same circuits that were optimized with replication of logic blocks after the placement performed by VPR. The circuits were placed on square FPGAs of minimum dimensions with respect to the number of pins and logic blocks. The circuits will be routed by VPR assuming a number of track that is 20% more than the minimum required number of tracks in order to make routable the original circuit. The timing performance will be evaluated using both our pre-routing timing analyzer and the post-routing timing analyzer embedded in VPR. Both the placement and the routing tool of VPR were executed in timing driven mode with default parameters. Thus the entire design flow is timing driven.

In Table 2 we compare the minimum clock period for the original placement and that after the algorithm has timing optimized the circuit. As you can see the average improvement is a 14.32% after the post-placement timing analysis and 12.92% after the post-routing timing analysis. The degradation of the post-routing gain with respect to the pre-placement gain is due to the fact that a timing driven router works better with a less optimized placement because in a very optimized placement there are a lot of near critical paths while in a less optimized there are just a few critical paths with a bigger delay. Thus the timing driven router with a small number of critical paths can allocate the routing resources to them without penalizing to much the near critical paths; on the contrary with a lot of critical paths it's very difficult for the routing to find

routing resources for all the near critical paths. This situation has a smaller impact if the number of routing resources is very high. In this case in fact the router can easily find routing resources for many paths without penalizing no one.

Circuit	Pre-Routing Timing Analysis			Post-Routing Timing Analysis		
	Clock period before replication (ns)	Clock period after replication (ns)	Gain (%)	Clock period before replication (ns)	Clock period after replication (ns)	Gain (%)
<i>alu4</i>	116.9	92.6	20.787	120.25	98.81	17.82952
<i>apex2</i>	128.88	104.73	18.73836	132.42	108.56	18.01843
<i>apex4</i>	109.7	98.3	10.39198	113.62	101.09	11.02799
<i>bigkey</i>	67.2	56.93	15.28274	69.01	64.82	6.071584
<i>clma</i>	229.4	178.11	22.35833	241.07	184.86	23.31688
<i>des</i>	81.99	76.71	6.43981	82.93	82.32	0.73556
<i>diffeq</i>	78.4	76.71	2.155612	80.87	76.08	5.923086
<i>dsip</i>	70.05	69.48	0.813704	74.93	74.97	-0.05338
<i>elliptic</i>	111.42	102.3	8.185245	112.08	104.31	6.932548
<i>ex1010</i>	205.6	169.7	17.46109	212.68	176.84	16.85161
<i>ex5p</i>	109.5	84.63	22.71233	111.51	89.01	20.17756
<i>frisc</i>	151.58	122.34	19.29014	154.64	125.46	18.86963
<i>misex3</i>	102.3	83.28	18.59238	104.65	88.52	15.41328
<i>pdcc</i>	244.38	185.66	24.02815	254.98	201.27	21.0644
<i>s298</i>	128.52	116.76	9.150327	130.58	123.76	5.222852
<i>s38417</i>	117.15	117.15	0	123.01	123.01	0
<i>s38584_1</i>	118.3	90.78	23.26289	122.92	96.07	21.84348
<i>seq</i>	114.63	92.25	19.52368	117.96	95.1	19.37945
<i>spla</i>	125.25	101.31	19.11377	149.26	117.73	21.12421
<i>tseng</i>	77.07	70.8	8.135461	76.34	69.77	8.606235
<i>average</i>			14.32115			12.91775

**Table 2: Pre-routing and post-routing results**

Another fact that could have a bad impact on the post-routing results is that the placer of VPR always tries to avoid the FPGA congestion. Our optimizer doesn't take into account congestion and, when it can't get a high gain, it degrades the post-routing performance of the

circuits. An example of this phenomenon is the circuit Dsip in which the pre-routing gain was of less of 1% and after the routing we have performance degradation.

We have also tried to route the circuit with in a situation with greater availability of routing resources and we have found a better situation. In Table 3 we compare the results for a routing of the circuits on FPGA with 20% more routing resources than required and for a routing of the circuits on FPGA with 50% more routing resources than required. As you can see our algorithm gains more with more routing resources.

Circuit	20% more tracks than minimum required			50% more tracks than minimum required		
	Clock period before replication (ns)	Clock period after replication (ns)	Gain (%)	Clock period before replication (ns)	Clock period after replication (ns)	Gain (%)
<i>alu4</i>	120.25	98.81	17.82952	120.43	96.2	20.11957
<i>apex2</i>	132.42	108.56	18.01843	132.42	108.57	18.01087
<i>apex4</i>	113.62	101.09	11.02799	113.62	101.11	11.01039
<i>bigkey</i>	69.01	64.82	6.071584	69.01	61.84	10.3898
<i>clma</i>	241.07	184.86	23.31688	239.84	184.07	23.253
<i>des</i>	82.93	82.32	0.73556	82.94	82.33	0.735471
<i>diffeq</i>	80.87	76.08	5.923086	79.67	76.08	4.506088
<i>dsip</i>	74.93	74.97	-0.05338	75.52	74.85	0.887182
<i>elliptic</i>	112.08	104.31	6.932548	112.08	104.32	6.923626
<i>ex1010</i>	212.68	176.84	16.85161	213.87	175.65	17.87067
<i>ex5p</i>	111.51	89.01	20.17756	111.51	88.81	20.35692
<i>frisc</i>	154.64	125.46	18.86963	154.64	125.46	18.86963
<i>misex3</i>	104.65	88.52	15.41328	104.42	86.74	16.93162
<i>pdcc</i>	254.98	201.27	21.0644	253.24	200.91	20.66419
<i>s298</i>	130.58	123.76	5.222852	131.77	119.59	9.243379
<i>s38417</i>	123.01	123.01	0	125.4	125.4	0
<i>s38584_1</i>	122.92	96.07	21.84348	122.92	96.82	21.23332
<i>seq</i>	117.96	95.1	19.37945	117.96	95.36	19.15904
<i>spla</i>	149.26	117.73	21.12421	128.49	106.38	17.20756
<i>tseng</i>	76.34	69.77	8.606235	76.34	70.36	7.833377
<i>average</i>			12.91775			13.26029

**Table 3: Results with greater availability of routing resources**

## 4.2. Wire Length Impact

Each time you perform replication it's possible that the wire length grows up. In fact when you replicate a cell you have to add to all the input nets of the original cell one more pin. The consequence is that, if the new pin is outside the current bounding box of the net, the half perimeter wire length (HPWL), that is one of the most reliable wire length measure at placement level, grows up. In Table 4 you can see the HPWL degradation of the benchmark circuits after the timing optimization of the algorithm.

Circuit	HPWL before replication (# of blocks)	HPWL after replication (# of blocks)	HPWL degradation (%)
<i>alu4</i>	8449	8708	-3.06545
<i>apex2</i>	13831	14119	-2.08228
<i>apex4</i>	10020	10836	-8.14371
<i>bigkey</i>	10349	10509	-1.54604
<i>clma</i>	57564	59346	-3.09568
<i>des</i>	14796	14840	-0.29738
<i>diffeq</i>	7048	7068	-0.28377
<i>dsip</i>	6241	6620	-6.07274
<i>elliptic</i>	24112	24402	-1.20272
<i>ex1010</i>	31959	34786	-8.84571
<i>ex5p</i>	9870	10432	-5.69402
<i>frisc</i>	33040	33298	-0.78087
<i>misex3</i>	9614	10127	-5.33597
<i>pdca</i>	44426	46702	-5.12313
<i>s298</i>	7759	8033	-3.53138
<i>s38417</i>	34681	34681	0
<i>s38584_1</i>	29017	29774	-2.60882
<i>seq</i>	12916	13650	-5.68287
<i>spla</i>	29778	31411	-5.48391
<i>tseng</i>	4549	4758	-4.59442
<i>average</i>			-3.67354

**Table 4: HPWL degradation**

### 4.3. Routability Impact

Our algorithm doesn't take into consideration routability constraints. Thus it's possible that the minimum number of tracks required to route a circuit increases after the optimization of our algorithm. The possible causes could be: the increase in the number of logic blocks in critical regions of the FPGA; the growth of the fanout of some net; the legalizer that degrades the uniformity of the placement.

Circuit	Min # of tracks before replication	Min # of tracks before replication	Routability degradation (%)
<i>alu4</i>	10	11	-10
<i>apex2</i>	11	12	-9.09091
<i>apex4</i>	11	12	-9.09091
<i>bigkey</i>	7	8	-14.2857
<i>clma</i>	12	13	-8.33333
<i>des</i>	8	8	0
<i>diffeq</i>	8	8	0
<i>dsip</i>	6	6	0
<i>elliptic</i>	12	12	0
<i>ex1010</i>	11	12	-9.09091
<i>ex5p</i>	14	15	-7.14286
<i>frisc</i>	14	14	0
<i>misex3</i>	11	11	0
<i>pdc</i>	17	18	-5.88235
<i>s298</i>	8	9	-12.5
<i>s38417</i>	8	8	0
<i>s38584_1</i>	9	8	11.11111
<i>seq</i>	12	12	0
<i>spla</i>	16	16	0
<i>tseng</i>	7	7	0
<i>average</i>			-3.71529

**Table 5: Degradation in minimum required number of tracks**

In Table 5 we compare the minimum number of tracks before and after our timing optimization. As you can see there is usually a small degradation in routability. Only in one case there is an improvement and in 9 cases there is degradation.

## CHAPTER 5

### Conclusions and Future Work

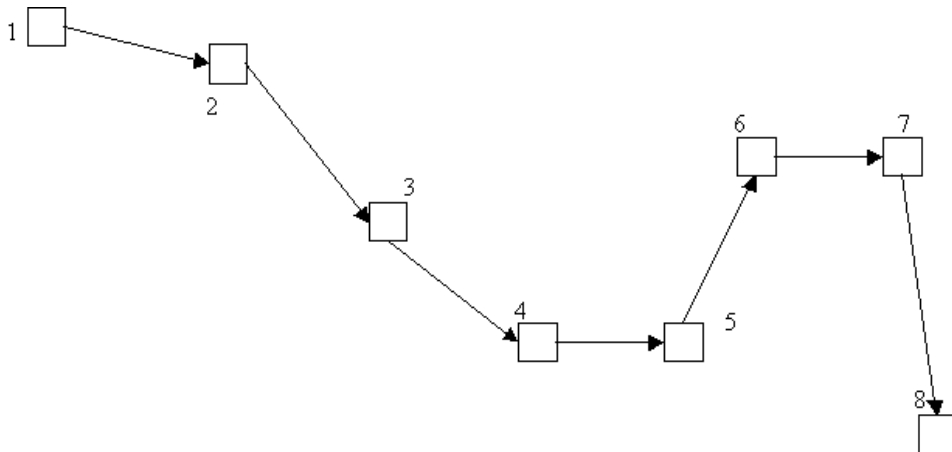
#### 5.1. Conclusions

This work examines the possibilities of timing driven logic replication at placement level and presents a path based algorithm for timing driven logic replication in FPGA. The algorithm tries to enforce monotonicity on the sub-paths that slow down the circuit with an iterative process that exploits the additional freedom degree of logic replication. The assumption of this work is that the algorithm will work on very timing optimized circuits. Thus, further timing optimization will always operate in regions with high density of critical cells. In such a situation the main focus of an optimizer must be to avoid penalizing too much the near critical sub-paths. And this is exactly the focus of our algorithm that always take into account the timing requirements on all the paths involved in a move. The experimental results confirm the effectiveness of our algorithm. In fact the comparison of our algorithm with a well-known timing driven placement tool that doesn't use logic replication reveals that our algorithm gains an average 13% on the clock period with a wire length degradation of just 3.7%.

There are two main problems that require further investigation: the problem of non-monotone clusters and the problem of replication of memory elements and pads.

## 5.2. Non-monotone Clusters

Our approach with respect to monotonicity is to optimize the sub paths of three elements. This approach doesn't consider the situations in which non-monotonicity of a path is not induced by local non-monotonicity of a sub path of 3 elements. Let's see an example. In Figure 36 there is a non-monotone path. In fact the total length of the path is more than the distance between source and target. However the length of the internal sub paths is always equal to the distance between source and target of the sub paths. The problem is with the nodes 4 and 5 and with 6 and 7. In fact from the local point of view of the sub paths of dimension three they are monotone; but if you look at the sub paths of dimension four they are not monotone. In fact the sub paths 3-4-5-6 and 5-6-7-8 are visibly non-monotone. At the moment our algorithm is not able to deal with this kind of non-monotonicity. The problem with these clusters of non-monotone nodes is that in order to eliminate the cluster you have to respect the constraints imposed by the feasibility regions induced by all the sub paths that constitutes the cluster.



**Figure 36: A path with non-monotone clusters**

Thus the placement of a cell is over-constrained by this huge amount of feasibility regions that it must respect. Another problem is that a single move made in order to increase monotonicity on a cluster can be completely ineffective by itself. In fact, in order to get an immediately measurable gain, you need a number of ineffective moves. In the example of Figure 36, in order to improve monotonicity on the sub path 5-6-7-8 you need to move down cell 6 without any immediate timing gain and then cell 7 obtaining a non-monotone sub path.

Simulated annealing can help in such a situation. In fact, if you permit also moves that don't lead to any gain, and you have a probability that also ineffective moves are accepted, the algorithm will be able to solve clusters. The problem with simulated annealing is that the number of replication that you can perform on an FPGA is not huge. Thus you need to develop algorithm able to get a good gain in a small number of iterations. A possible solution could be to run sometimes a kind of garbage collector that checks for each replicated cell if it is really necessary in order to improve performance or if it's possible to kill it and to free room for another more useful replication.

### 5.3. Replication of Memory Elements and Pads

Like for combinatorial blocks, also for sequential blocks and pads replication could be useful. In this case monotonicity of critical paths is not a good parameter in order to guide the placement of a replicated memory element or pad because they are at the beginning and at the end of the paths. However placing a memory element so that monotonicity of the pseudo paths induced by the concatenation of incoming and outgoing paths would improve performance. Moreover in such a situation the position inside the feasibility region would be an important issue. In fact putting a node closer to the more critical paths would decrease the delay of the paths.

## CITED LITTERATURE

1. G. E. Moore: Cramming More Components Onto Integrated Circuits. Electronics Magazine, 38: 114-117, 1965
2. H. B. Bakoglu: Circuits, Interconnections and Packaging for VLSI. Reading, Massachusetts, Addison-Wesley, 1990.
3. P. K. Wolff, A. E. Ruehli, B. J. Agule, J. D. Lesser and G. Goertzel: Power/Timing: Optimization and Layout Techniques for LSI Chips. Journal of Design Automation & Fault Tolerant Computing, 145-164, 1978.
4. A. E. Dunlop , V. D. Agrawal , D. N. Deutsch , M. F. Jukl , P. Kazak: Chip layout optimization using critical path weighting. Papers on Twenty-five years of electronic design automation, 278-281, 1988
5. M. Burstein, M. N. Youssef: Timing influenced layout design. ACM/IEEE conference on Design automation, 124-130, 1985.
6. P.S. Hauge, R. Nair, E J, Yoffa: Circuit Placement For Predictable Performance. ICCAD, 88-91, 1987.
7. R. Nair, C.L. Bennan, P.S. Hauge, E.L Yoffa: Generation of Performance Constraints for Layout, IEEE Trans. CAD, 8: 860-874, Aug. 1989.
8. S. Teig, R.L. Smith, and J. Seaton: Timing-Driven Layout of Cell-Based iCs. Design Automation Guide, 94-101, 1987.
9. M. Marek-Sadowska and S. P. Lin: Timing-Driven Placement. ICCAD, 94-97, 1989.
10. I. Lin , D. H. C. Du: Performance-Driven Constructive Placement, ACM/IEEE design automation conference, 103-106, 1990.
11. W. E. Donath , R. J. Norman , B. K. Agrawal , S. E. Bello , S. Y. Han , J. M. Kurtzberg, P. Lowy , R. I. McMillan: Timing Driven Placement Using Complete Path Delays. ACM/IEEE design automation conference, 84-89, 1990.

12. A. Srinivasan, K. Chaudhary, and E. S. Kuh: RITUAL: A Performance Driven Placement Algorithm for Small Cell IC's. ICCAD, 48–51, 1991.
13. A. Mathur, C. L. Liu: Compression-Relaxation: a New Approach to Performance Driven Placement for Regular Architectures. IEEE/ACM on Computer-aided design, 130-136, 1994.
14. J. Lillis, C.-K. Cheng, T.-T. Y. Lin: Algorithms for Optimal Introduction of Redundant Logic for Timing and Area Optimization. Proc. IEEE International Symposium on Circuits and Systems, 4: 452-455, 1996.
15. C. Kring and A. Newton: A cell-Replicating Approach to Minicut-Based Circuit Partitioning. IEEE International Conference on Computer-Aided Design, 2-5, 1991.
16. C. M. Fiduccia , R. M. Mattheyses: A Linear-Time Heuristic For Improving Network Partitions. DAC, 175-181, 1982.
17. I. Neumann , D. Stoffel , H. Hartje , W. Kunz: Cell Replication and Redundancy Elimination During Placement for Cycle Time Optimization, International conference on Computer-aided design, 25-30, 1999.
18. A.Srivastava, R. Kastner and M. Sarrafzadeh: On the Complexity of Gate Duplication. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20: 1170-1176, 2001.
19. S. Brown, R. Francis, J. Rose, Z. Vranesic: Field Programmable Gate Arrays. Norwell, Massachusetts, Kluwer Academic Publisher, 1992.
20. V. Betz and J. Rose: VPR: A New Packing, Placement and Routing Tool for FPGA Research. International Workshop on Field-Programmable Logic, 213-222, 1997.
21. V. Betz, J. Rose, and A. Marquardt: Architecture and CAD for Deep-Submicron FPGAs. Norwell, Massachusetts, Kluwer Academic Publishers, 1999
22. A. Marquardt, V. Betz and J. Rose: Timing-Driven Placement for FPGAs. International Symposium on FPGAs, 203-213, 2000.

23. C. Cheng: RISA: Accurate and Efficient Placement and Routability Modeling. ICCAD, 690-696, 1994.
24. W. C. Elmore: The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. J. Appl. Phys., 19: 55-63, 1948.
25. J. Robinstein, P. Penfield Jr., and M. A. Horowitz: Signal Delay in RC Tree Networks. IEEE Trans. Compuc -Aided Design, CAD-2: 202-211, 1983.
26. B.Chen and M.Marek-Sadowska: Timing Driven Placement of Pads and Latches. Application Specific Integrated Circuits Conference, 30-33, 1992.
27. H. Eisenmann , F. M. Johannes: Generic Global Placement and Floorplanning. DAC, 269-274, 1998.
28. S. L. Ou and M. Pedram: Timing-Driven Placement Based on Partitioning With Dynamic Cut-Net Control. DAC, 472-476, 2000.
29. A. B. Kahng, S. Mantik, I. L. Markov: Min-Max Placement for Large-Scale Timing Optimization. ISPD: 143-148, 2002.
30. B. Riess and G. Ettl: SPEED: Fast and Efficient Timing Driven Placement. IEEE International Symposium on Circuits and Systems, 377 - 380, 1995.
31. A. Srinivasan, A K. Chaudhary, E. S. Kuh: RITUAL: Performance Driven Placement Algorithm for Small Cell Ics. ICCAD, 48-51, 1991.
32. J. Hwang, A. El Gamal, Optimal Replication for Min-Cut Partitioning. IEEE/ACM international conference on Computer-aided design, 432-435, 1992.
33. L. T. Liu, M. T. Kuo, C. K. Cheng, and T. C. Hu: A Replication Cut for Two-Way Partitioning. IEEE Transactions on Computer-Aided Design, 14: 623-629, 1995.
34. Wai-Kei Mak , D. F. Wong: Minimum Replication Min-Cut Partitioning: IEEE/ACM international conference on Computer-aided design, 205-210, 1996.
35. M. Enos , S. Hauck , M. Sarrafzadeh, Replication for Logic Bipartitioning, IEEE/ACM international conference on Computer-aided design, 342-349, 1997.