Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

1992

A partitioning-based approach to the fitting problem in special architecture EPLDs

Steffan Goller

Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Goller, Steffan, "A partitioning-based approach to the fitting problem in special architecture EPLDs" (1992). *Dissertations and Theses*. Paper 4218.

<https://doi.org/10.15760/etd.6102>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

AN ABSTRACT OF THE THESIS OF Steffen Göller for the Master of Science in Electrical Engineering presented July 9, 1992.

Title: A Partitioning-Based Approach to the Fitting Problem in Special Architecture EPLDs

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:

Malgorzata E. Chrzanowska-Jeske, Chair

Marek A. Perkowski

Maria E. Balogh

In this thesis, we describe an architecture-driven fitting algorithm for an Application-Specific EPLD, the CY7C361, from Cypress Semiconductor. Traditional placement and routing tools for PLDs perform placement and routing separately. Several placement possibilities are created and the router tries to realize the connections between the physical locations of the cells on the chip. The Cypress CY7C361 has a very unique chip architecture with a highly limited connectivity between the physical cells. Therefore, it is necessary to consider the routability when the placement of cells is performed. The combination of the two stages is called *fitting*. The specific architecture-dependent constraints, imposed on the connectivity

of the CY7C361 chip were used to develop a hierarchical partitioning structure of the algorithm. This approach limits very effectively the solution space in the early stage of the search for a solution of the fitting problem. The partitioning approach for the fitting algorithm is not limited on the Cypress CY7C361. It can be applied to other architectures with similar connectivity restrictions, too.

A PARTITIONING-BASED APPROACH TO THE FITTING PROBLEM IN
SPECIAL ARCHITECTURE EPLDS

by
STEFFEN GÖLLER

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL ENGINEERING

Portland State University
1992

TO THE OFFICE OF GRADUATE STUDIES:

The members of the committee approve the thesis of Steffen Göller presented
July 9, 1992.

Malgorzata E. Chyzanowska-Jeske, Chair

Marek A. Perkowski

Maria E. Balogh

APPROVED:

Rolf Schaumann, Chair, Department of Electrical Engineering

Roy W. Koch, Vice Provost for Graduate Studies and Research

TABLE OF CONTENTS

	PAGE
LIST OF TABLES	v
LIST OF FIGURES	vi
GLOSSARY	viii
CHAPTER	
I INTRODUCTION	1
II PLACEMENT AND ROUTING TECHNIQUES FOR EPLDS/FPGAS	8
II.1 FPGAs from Xilinx and Actel	9
II.2 The Tryptich FPGA	11
II.3 The Cypress EPLD	13
II.4 The Concurrent Logic FPGA	14
III THE STATE MACHINE CY7C361 EPLD	16
III.1 The Architecture of the CY7C361	16
III.2 Development System for the CY7C361	22
IV ARCHITECTURE CONSTRAINTS FOR THE FITTING ALGO- RITHM	23
V DEFINITIONS AND PROBLEM FORMULATION	34
V.1 Representation of the Chip Resources	34
V.2 Representation of the Netlist	41
V.3 Formulation of the Fitting Problem	43
VI DESCRIPTION OF THE FITTING ALGORITHM	44

VI.1	P_3 Assignment	48
VI.2	First Partition	51
VI.3	P_{13} and P_{23} Assignment	59
VI.4	Second Partition	63
VI.5	Physical Placement	69
VI.5.1	Placement Matrix	
VI.5.2	Verification of Local Reset Output Restrictions	
VI.5.3	Final Physical Placement	
VII	RESULTS OF THE FITTER	81
VIII	HEURISTICS FOR THE FIRST PARTITION	89
IX	COMPLEXITY ANALYSIS	93
X	CONCLUSION	97
	REFERENCES	99

LIST OF TABLES

TABLE	PAGE
I Result of First and Second Partition	70
II Columns of the Placement Matrix and Partitions	72
III Industrial Examples	83
IV Examples where PABFIT is Better than Cyperss Fitter	84
V Additional Local Resets	87
VI Examples where PABFIT Uses Heuristics	92

LIST OF FIGURES

FIGURE		PAGE
1	Split Plane Architecture of the CY7C361.	18
2	Architecture of the CY7C361.	19
3	Array of Local Reset Groups of the CY7C361.	21
4	Interconnection Matrix of the CY7C361.	24
5	First-Level Partitioning of the Interconnection Matrix.	26
6	Second-Level Partitioning of the Interconnection Matrix.	28
7	First- and Second-Level Partitioning of the Interconnection Matrix.	29
8	Output Availability of State Macrocells to Local Resets.	32
9	First-Level Partitioning of G_P	38
10	Second-Level Partitioning of $G_{P1} \cup G_{P31}$ and $G_{P2} \cup G_{P32}$	40
11	Additional Normal Edge due to Input Restrictions of Local Resets.	42
12	Flowchart of the PABFIT Algorithm.	45
13	Chains in the P_3 Assignment.	50
14	Example of a First Partition.	54
15	Example of a First Partition with a chain_cut_1.	57
16	Chains in the P_{13} Assignment.	61
17	Example of a Second Partition.	65
18	Example of a Second Partition with chain_cut_2.	67
19	Placement Matrix.	71
20	Examples of Local Reset Output Violations.	74

21	State Cell Feasible Mapping.	77
22	Local Resets and the Symbolic Graph.	78
23	Local Reset Connectivity Matrix.	80
24	Degree Vector.	91

GLOSSARY

P_α	partition of physical macrocells
G_P	physical connectivity graph
V_P	vertex set of G_P
v_{Pi}	element of V_P
EN_P	normal edge set of G_P
en_{Pi}	element of EN_P
EC_P	chain-edge set of G_P
ec_{Pi}	element of EC_P
$G_{P\alpha}$	subgraph of G_P representing P_α
$V_{P\alpha}$	vertex set of $G_{P\alpha}$
$EN_{P\alpha}$	normal edge set of $G_{P\alpha}$
$EC_{P\alpha}$	chain-edge set of $G_{P\alpha}$
G_T	super graph with $G_{P\alpha}$ as vertices
V_T	vertex set of G_T
EN_T	normal edge set of G_T
EC_T	chain-edge set of G_T
G_S	symbolic graph representing the netlist
V_S	vertex set of G_S
v_{Si}	element of V_S
EN_S	normal edge set of G_S
en_{Si}	element of EN_S

EC_S	chain-edge set of G_S
ec_{Si}	element of EC_S
p_i	physical macrocell of the CY7C361
plr_i	physical local reset
LRG_i	physical local reset group
lr_i	symbolic local reset
CH_i	symbolic chain i

CHAPTER I

INTRODUCTION

At the beginning of circuit integration, the design of a circuit on a piece of silicon was done completely manually by the designer. The design started with an empty piece of silicon at the transistor-level. The designer had to place the transistors, interconnect them to gates or memory blocks, and realize the connections between the different blocks by hand. Therefore, design process was very time consuming, especially as the integration of the circuit was increasing. Within the last years, more and more CAD tools have been developed for the design automation of integrated circuits. Today, tools for all different design methodologies are available.

The approach to circuit realization can be classified in two distinct design methodologies [1], both covering a variety of different design styles. One methodology is the *Full Custom Design* [1] style that offers a maximum flexibility for applications of the technology in terms of the transistor sizing, the placement of the transistor on the silicon, and the interconnections. A development system for a Full Custom Design usually includes large libraries with predefined blocks (gates or modules), that can be included in the design. However, the designer still has the possibility of customizing the design at the transistor-level. After the design of the modules and the specification of the connections between them, the designer can use CAD tools that perform the placement and routing of the modules on the chip. Several tools are available for floorplanning [2] [3], placement [4], global routing, and detailed routing [5] [6] [7].

Contrary to the Full Custom Design, the *Semi Custom Design* [1] starts with a chip with regular pre-patterned silicon arrays that are customized at the interconnect-level in order to realize a given circuit. This design style offers less flexibility than a Full Custom Design, but the regularity of the predefined silicon arrays made it possible to develop CAD tools that automatize the complete design process. The circuit can be described in an abstract High-Level Description Language (HDL) [8], rather than on the transistor/gate-level. Thus, the design becomes independent from the available device and the manufacturing cycle and manufacturing cost of the circuit design is strongly reduced.

The Semi Custom Design style is very important for many *Application Specific Integrated Circuits* (ASICs) [1]. These are special purpose integrated circuits where a Full Custom Design is too time consuming and too expensive. The devices used for ASICs are *Gate Arrays*, *Standard Cells*, *Sea of Gates*, and *Programmable Logic Devices* (PLDs). They differ in the logic and routing resources available on the chip and the programmability of these resources.

Gate Arrays and Standard Cells are devices with pre-fabricated arrays of logic blocks, consisting of transistors, gates (AND, OR etc.), or memory cells. Gate Arrays consist of blocks that have all the same size (length and height). Standard Cells contain blocks where the height is predefined but the length is variable. The locations of the blocks on the chip are fixed. They are surrounded by horizontal and vertical *routing channels* that allow the realization of connections between the blocks. Each connection requires a *routing track*, that is routed directly on the silicon, where the routing track is located in the routing channels. The circuit design starts with empty routing channels and then routes sequentially all connections. Several placement and routing tools [9] are available that show good results for these

kind of devices. Devices where the routing channels are pre-fabricated are called *Mask Programmable Gate Arrays* (MPGAs). The available routing tracks between the blocks can be programmed by the designer. Programming means, connecting a routing track to a logic block on the silicon (hardware programmable).

Sea of Gates devices are quite different. The silicon is filled with a high number of transistors and there are no routing channels available. Only a certain fraction of the transistors can be used as circuit elements. The other transistors are for connections between cells or blocks. Still, there is a very high number of transistors that can be used in order to realize logic and memory elements. A CAD tool for the circuit design using Sea of Gates was presented in [3].

Gate Arrays, Standard Cells, and Sea of Gates have the disadvantage, that the circuit design is followed by the actual manufacturing of the chip. Therefore, they are not very well applicable for prototyping and for designs that require a short circuit design cycle. Devices that overcome this disadvantage are PLDs [10]. They are devices that are completely manufactured and where the connections and blocks on the chip are customized by programming so called *switches* (software programmable). These switches can realize certain connections or configure the logic blocks. PLD devices *Programmable Logic Arrays* (PLAs), *Programmable Array Logic* (PALs), *Erasable Programmable Logic Devices* (EPLDs), and *Field Programmable Gate Arrays* (FPGAs) [10]. They differ in the switch realizations and the flexibility of the logic and memory blocks and the connections between them.

PLAs and PALs [11] allow the user to realize Boolean functions depending on several input variables by programming a matrix-like interconnection array. The inputs of the chip feed an AND-plane which allows the realization of product terms of input variables. These product terms can be combined at the following OR-plane.

Therefore, PLAs/PALs can realize Boolean functions that are described by Sum of Product (SOP) terms. Typically, PLAs/PALs contain an array of memory elements (D-Flip-Flops) that are triggered by the Sum of Product terms. Programming the PLA/PAL means, creating connections on the AND/OR-plane of the chip. PALs allow the programming of the OR-plane only, while PLAs allow for the programming of the OR-plane and the AND-plane. The cells themselves, the AND, OR gates and the memory elements are predefined.

The restriction of non-configurable logic gates and memory blocks does not exist any longer for FPGAs or EPLDs. The programmable *macrocells* (logic and memory blocks) that are available on the chip, can be configured by the user according to the requirements of the design. Depending on the chip architecture the macrocells can be programmed as AND, OR gates, Multiplexors, D-flip-flops, or more complex types. The routing resources of the chip consist of routing channels surrounding all the macrocells. Each routing channel consists of several wire segments. The wire segments are used in order to realize connections between macrocells. That means, the basic structure of the routing resources is pre-defined. The user can program the connection between the wire segments and the macrocells, by choosing from all the possibilities offered by the chip architecture. FPGAs exhibit similar properties as MPGAs. Contrary to MPGAs, the circuit design of FPGAs is based on a completely manufactured chip. The wire segments are connected by configuring programmable switches. For most FPGAs, these switches are realized as pass-transistor-switches or diffusion-switches. EPLDs usually use EPROM-based switches.

As FPGAs and EPLDs become more and more popular for prototyping and low production volume applications of digital circuits, there is a high demand for

design automation systems for these kind of devices. Contrary to Gate Arrays, FPGAs/EPLDs usually have a more restricted connectivity between the macrocells. The routing resources are not as flexible as the routing channels of Gate Arrays. Gate Arrays allow the designer to create the routing tracks in the channels by himself. At FPGAs, the user is restricted to the available wire segments. Therefore, special synthesis and technology mapping methods have been developed [12] [13] [14].

In a design automation system for FPGAs/EPLDs, the designer can describe the circuit that he wants to implement in a High-Level Description Language. This description is transformed within the system to internal formats or expressions that describe the sequential and combinatorial logic of the circuit separately. State machine and logic minimization methods are used to optimize these expressions. The next step is a *technology mapping* [15] [16], where the circuit is divided into pieces of logic each of which can be realized by a single macrocell. In this stage, the flexibility of the macrocell is very important. The higher the flexibility, the more complex Boolean functions can be realized with one macrocell. The result of the technology mapping is a *netlist* that contains the modules of the circuit that can be mapped to a single macrocell of the chip and the connections between the modules. The final two stages in the design automation system are the *placement* and *routing*. The placement assigns each module of the netlist to a physical macrocell on the chip. In the routing stage the connections between the cells are realized using the available wire segments in the routing channels. This implies the configuration of the switches that realize the connections. Traditional approaches to the placement and routing for FPGAs/EPLDs are based on the same general tools and algorithms as for Gate Arrays. However, the routing resources the placement and routing problem is of high complexity (NP-complete problem).

The traditional approach for FPGAs and EPLDs separates placement and routing. The placement is performed first, followed by routing. Most placement tools perform the placement such that an estimated wire length of the routed connections resulting from the placement is minimized. However, the overall routability of the circuit is not considered within the placement stage. Because of the limited connectivity of FPGAs/EPLDs the router may not be able to realize all the connections from the netlist for a given placement. In such a case, the traditional approach has been to create a new placement followed by another routing effort until a routable placement is found.

A placement tool that does not take the routability into account, becomes very inefficient for EPLDs and FPGAs with specific architectures like the *Cypress CY7C361* EPLD where the connectivity is highly restricted. For the CY7C361, a routing cannot be performed, because the connections between the physical macrocells are pre-defined by the chip architecture. There are no configurable wire segments available. For each element of the netlist that is assigned to a single physical macrocell on the chip, the possible output and input connections to or from other physical macrocells are fixed. Therefore, the placement determines completely, if connections between elements of the netlist are realizable or not.

A new approach, that was proposed in [17], to solve the placement problem for special-architecture EPLDs, takes the connectivity restrictions of the chip architecture into account during the placement stage. This approach is denoted by the term *fitting*. A cell is placed to a physical location only if no connectivity constraints are violated. Placements that violate the architecture constraints are excluded in a very early stage of the fitting algorithm, and a final placement is performed only if no architecture constraints are violated. This approach reduces drastically the solution

space of the placement problem and makes it possible to explore the whole solution space for highly restricted chips architectures. As a result it is possible to show if a netlist can be mapped on the device or not. However, the fitting algorithm for the CY7C361 proposed in [17] considered barely the architecture constraints. The fitting algorithm presented in this thesis, reduces the solution space of the fitting problem much more effectively by taking all architecture constraints into account.

The general Fitting Approach is not limited to the Cypress CY7C361. The principle of using the connectivity restrictions in order to exclude unfeasible placements is applicable to different chip architectures. However, since the architecture constraints depend on the specific chip used for the design, the connectivity restrictions have to be stated differently for each chip. A fitting algorithm has to be based on the specific restrictions of the chip it is applied to.

In Chapter II several FPGAs/EPLDs with different architectures are illustrated. The placement and routing techniques that are currently used for these devices are described. Other architectures where a Fitting Approach is advantageous are also presented. Chapter III describes the architecture of the CY7C361 in detail. Then in Chapter IV the connectivity restrictions that are used in the fitting algorithm are shown. The specific architecture-dependent constraints are applied to develop a hierarchical partitioning structure of the fitting algorithm. The partitioning hierarchy is illustrated in Chapter V. The architecture-driven fitting algorithm itself is presented in Chapter VI. The program, called PABFIT (PARTition Based FITter) which implements the proposed algorithm has been tested on several industrial examples. Experimental results are listed in Chapter VII. Heuristics to speed up the algorithm are presented in Chapter VIII. Finally, Chapter IX investigates the complexity of the fitting algorithm.

CHAPTER II

PLACEMENT AND ROUTING TECHNIQUES FOR EPLDS/FPGAS

Chapter I describes a design automation system for FPGAs and EPLDs. In this thesis we concentrate on the last two stages of such a system: the placement and routing.

In general, FPGAs/EPLDs can be divided into two categories. To the first category belong the FPGAs/EPLDs that have a very flexible routing structure similar to the routing resources of Gate Arrays. Such devices are for example the Xilinx [18] and Actel [19] FPGAs. There, a separation of the placement and routing stage is well applicable. General placement and routing algorithm originally designed for Gate Arrays and modified for Xilinx and Actel FPGAs show good results [18] [19] [20] [16]. However, the complexity of the routing depends strongly on the result of the previous placement stage. In the worst case, the router may not be able to route all the connections of the netlist for a given placement (routing conflict).

The second category includes EPLD/FPGA chip architectures where the connectivity between the macrocells is much more restricted. For these devices a separation of the placement and routing stage is not effective. Placement and routing are highly dependent, and the placement determines already how the router has to realize certain connections of the netlist. Therefore, it is more likely that a placement obtained by the above tools leads to a routing conflict. The placement for these EPLDs/FPGAs has to take the connectivity restrictions of the chip architecture into

account. Therefore, the placement problem can be solved by the Fitting Approach as described in Chapter I.

Examples of the second category are the CLi6000 FPGA from Concurrent Logic [21] and the Cypress CY7C361 [22]. In both devices, the connectivity between the macrocells is highly restricted. The CLi6000 allows still a routing after the placement. However, the possibilities of the router are limited. For the CY7C361 the placement stage determines completely if a connection of the netlist is realizable or not. The chip architecture allows no routing after placement. Therefore, the placement problem of the CY7C361 can be solved more effectively by a Fitting Approach where a placement is performed only if the connections of the netlist are realizable.

In the following, placement and routing approaches for several FPGA/EPLD architectures are presented. The tools for the Xilinx FPGA, the Triptych FPGA and the Cypress CY7C361 EPLD are described. For the CY7C361, we start with a description of the fitter that is included in the Cypress Development System. Then we describe the new fitting algorithm that is introduced in this thesis. The fitter of the Cypress System is denoted by *Cypress Fitter*, and our new fitter is denoted *PABFIT* (PArTition Based FITter). At the end of this Chapter, we describe the architecture of the CLi6000, and how the placement problem could be solved for this device.

II.1 FPGAS FROM XILINX AND ACTEL

The Xilinx-chip [18] includes three basic user programmable elements: *I/O Blocks*, *Configurable Logic Blocks* (CLBs) and *Programmable Interconnections*. The CLBs are arranged in a matrix-like structure. Each CLB has a combinatorial logic

section (lookup table based) and a storage element. The cell can be configured as a combinatorial cell only, realizing any logic functions of few inputs (2000-series: 4 input variables). Each CLB is surrounded by horizontal and vertical routing-channels. A routing channel itself consists of several wire segments that represent the possible physical connections between the logic cells. At the intersections of the horizontal and vertical routing channels, a vertical wire segment can be connected to a horizontal wire segment by a user programmable switch matrix consisting of *switch blocks*. The wire segments themselves can be connected to CLBs by *connection blocks* that are placed at each side of the CLB (north, east, south, and west). By programming or configuring a switch or connection block, we mean creating connections between the wire segments incident to a switch or connection block. The architecture of the Actel [19] chip is similar. It differs only in the logic structure of the macrocell, the routing structure is very similar.

In [20] a routing algorithm for the Xilinx chip is proposed. Before the routing stage, the technology mapping is performed by a standard mapper for lookup table based FPGAs [16]. For the placement a *min-cut* placement algorithm [23] adopted from Standard Cell placement tools is used. The connection and switch blocks of the Xilinx are very flexible such that the number of possible configurations is very high. This gives the router several possibilities to choose from. Based on a given placement, the router tries to realize each connection of the netlist by choosing from a limited set of routing possibilities only. The set of all routing possibilities would be too high to determine for all connections that have to be realized. If a routing conflict occurs when the router tries to realize a single connection, a new set of routing possibilities is determined until the whole netlist is mapped to the chip.

In general, increased flexibility of the switch and connection block will in-

crease the number of different configurations that can be programmed. Therefore, the set of routing possibilities will become larger and the routability for a given netlist might be increased. The interaction between CAD tools that are used to configure FPGAs routing resources (connection and switch blocks) and the design of the routing architecture itself (flexibility of the connection and switch blocks) has been investigated in [24]. It was observed that the routing resources can be designed such that higher routability does not necessarily mean more complexity of the switch and connection blocks. The result of the study was that higher flexibility of the connection blocks is required for a good routability. However, a relatively low flexibility is sufficient for the switch blocks. That means, the availability of macrocell inputs and outputs at the wire segments is more important than the connectivity between the wire segments themselves.

Similarly to common tools for Gate Array, the router introduced in [20] does not consider the routability during the placement stage. As a result the placement of the macrocells might lead to routing conflicts that can not be solved by creating a new set of routing possibilities. In that case new placements have to be determined. In [24] it was shown, that these routing conflicts are more likely to occur when the flexibility of the connection block is low. This leads to the conclusion that it is necessary to combine placement and routing for chips with limited availability of macrocell inputs and outputs at wire segments. The Fitting Approach introduced in Chapter I has to replace the traditional placement followed by routing approach.

II.2 THE TRYPTICH FPGA

An effort to combine placement and routing was reported in [25]. There, a new FPGA chip called Triptych was introduced. The architecture of Triptych was

designed to make the placement and routing resources more applicable to common occurring logic functions. The authors of [25] claim that the macrocell architecture and the routing resources of the Triptych FPGA matches the form of many circuits. A placement that takes the structure into account can produce routable implementations. For example, if a netlist has a tree-structure, a placement starting from the root of the tree towards the leaves is very likely to lead to a routable placement.

In the following, we describe the architecture of the Triptych FPGA and the placement tool that was introduced in [25]. The chip consists of an array of macrocells. Each element in the array has short hard wired connections to its nearest neighbors in adjacent columns. This basic structure is augmented with segmented vertical routing channels between the columns. A routing-directed placement algorithm heavily influenced by the proposed chip architecture has been developed in [25]. The placement tool tries to create routable placements by maintaining and meeting, for each cell that is placed, a limited set of local routing criteria representing some of the overall routing constraints. A cell i of the netlist is placed to a physical macrocell only if all the criteria are met for the cell i and all the cells around cell i . The criteria are based on the input and output signals required for the cell i according to the netlist. They are stated as follows:

- All input signals of cell i coming from other cells that are already placed can be routed.
- The cell i can realize the output signal required according to the netlist as a function of the input signals.
- The output signal of cell i can be routed to all cells that require that signal as an input signal.

Nevertheless, this approach is only local, and placement and routing are not completed in one process.

The placement approach for the Triptych FPGA is in a certain way similar to the approach used in the Cypress Fitter [17].

II.3 THE CYPRESS EPLD

The Cypress Fitter [17] is a part of the *Cypress Development System* [26] for the CY7C361 chip [22]. It will be presented in Chapter III.2. The Cypress Fitter is based on the Fitting Approach where the connectivity restrictions of the chip architecture are taken into account during the placement. Therefore, the Cypress Fitter can be viewed as a routing-directed placement algorithm where the routing constraints are checked locally for each placed cell, and a backtracking mechanism is used if the routing is not feasible. A possible placement is determined by placing one macrocell at a time and checked subsequently for emerging routing conflicts. If a routing conflict occurs the algorithm generates another placement by shifting the last placed macrocell to the next open position until a feasible placement and routing is found. The constraint-backtracking is used as a local greedy search method. In the worst case the whole solution space of all possible placements has to be explored. The limited connectivity and the so called C-IN chain restrictions are taken into account during the routing directed placement phase. The 32 physical macrocells of the CY7C361 are ordered in a vertical array and each macrocell can have a *C-IN-chain* from the previous macrocell. That means two macrocells of the netlist that are neighbors in a C-IN-chain have to be placed to physically adjacent macrocells on the chip. When a macrocell that is included in a chain is mapped to a single physical location on the chip, the Cypress Fitter places also the other

macrocells of the same C-IN chain by taking the adjacency into account. If a placement of a symbolic macrocell to a physical location is not possible because of a non-realizable connection, or because the physical location is already occupied, the algorithm backtracks.

The results presented in [17] demonstrate the weaknesses of the Cypress Fitter. It could not find a *feasible* placement for a number of more complicated examples in acceptable time. By feasible placement we understand a placement of each symbolic macrocell of the netlist to a physical location on the chip such that all the connections of the netlist are realizable. The solution space for the Cypress Fitter is very large and it is not limited effectively enough by the architecture constraints.

The new fitting algorithm (PABFIT) for the CY7C361 introduced in this thesis takes advantage of the restrictions imposed by the chip architecture in a very effective way. Unfeasible placements are excluded in an early stage of the algorithm, such that the solution space of the fitting problem is reduced tremendously.

The concept of architecture-driven fitting algorithms for EPLDs/FPGAs will become more important as more new special architecture EPLDs/FPGAs come to market. The increasing demand for especially high speed FPGAs/EPLDs requires architectures with a limited connectivity between the macrocells, like the Cypress CY7C361. There is always a tradeoff between a high connectivity and the speed of the device.

II.4 THE CONCURRENT LOGIC FPGA

Another FPGA with limited connectivity is the CLi6000 from Concurrent Logic [21]. The chip consists of an array of macrocells. Each macrocell has a possible connection to all its neighbors (north, east, south, west) and is surrounded

by horizontal/vertical *local* and *global buses*. The chip is divided into blocks of 8x8 macrocells. Each macrocell in the 8x8 block can be connected to a local bus wire segment of this block. The global bus is not connected to the macrocells of the CLi6000. It is used in order to connect the local buses of a 8x8 block to the local bus of another 8x8 block. Therefore, the routing resources of the CLi6000 can be classified into two categories. First, the macrocell to macrocell connections and the local bus can be used in order to connect highly connected elements of the netlist (*clusters*). Second, the connections between these clusters are realized using the global buses. The hierarchy of the routing resources of the CLi6000 is similar to the hierarchy of the output availability of macrocells of the CY7C361. Chapter III shows the classification of the macrocells of the CY7C361 to *local*, *intermediate*, or *global* according to their output availability to inputs of other macrocells. This hierarchy is used in the PABFIT algorithm.

CHAPTER III

THE STATE MACHINE CY7C361 EPLD

A new Programmable Logic Device from Cypress Semiconductor, the *Cypress CY7C361 EPLD* [27], has been optimized for realization of very fast asynchronous (self-synchronized) sequential controllers by means of concurrency and partitioning of FSMs. Controllers are designed from multiple deterministic, non-deterministic, and parallel finite state machines, and mapped into this new device [8]. In the following Chapter III.1, we present the architecture of the CY7C361. Chapter III.2 describes the design automation system for this new device.

III.1 THE ARCHITECTURE OF THE CY7C361

The Cypress CY7C361 [27] represents a EPLD device with highly limited connectivity between the macrocells. There are only 32 macrocells available on the chip. They allow the realization of only three different configurations: (1) START, (2) TERMINATE, and (3) TOGGLE. Contrary to other architectures from Xilinx [18], Actel [19] or Concurrent Logic [21], the macrocells of the Cypress chip do not correspond to the direct realizations of Boolean function (AND, OR, etc.) and storage elements (D-flip-flops, latches, etc.). They are designed for the immediate realization of the basic elements of Tokenized State Machines. The methodology of Tokenized State Machine design developed for this chip is described in [8]. The idea for this new architecture was created by analyzing the utilization of the behavioral PLA structure.

Conventional two-level PLD architectures consist of an input plane and an array of state macrocells triggered by the outputs of the input plane. For PLAs/PALs the input plane consists of a programmable AND/OR array that allows the realization of Sum Of Product (SOP) terms as Boolean functions for the conditions to the state macrocells. Usually, there is only little feedback from the output of the macrocells back to the input plane.

The architecture of the CY7C361 is shown in Figure 1. The *state macrocell array* is located between an *input* and an *output array*. The input array allows the realization of very fast conditions to state macrocells as a function of the inputs of the chip and the outputs of the state macrocells. The output array allows the realization of the outputs of the chip as a function of the state macrocells output only. When comparing the *CY7C361* architecture to conventional two-level PLD architectures, there is much more feedback from the state output of the macrocell array back to the input array. The feedback-pathes of the CY7C361 are realized with very short wires that allow a shorter cycle time and result in a higher speed of the device.

More details about the chip architecture of the CY7C361 are shown in Figure 2. The input array consists of the Segmented Interconnection Array and an array of logic gates feeding, in a one-for-one fashion, the state macrocell array. Each logic gate in the array is the product of a product term and the complement of a product term. It is called *Condition Decoder (CDEC) gate* ($CDEC = P * \bar{Q}$, where P and Q are product terms). Compared to traditional PLA architectures where the excitation functions of the state macrocells are Sum of Product (SOP) terms, the excitation functions of the *CY7C361* macrocells are Sum of CDEC gate expressions. As it was found by the analyzing the structural behavior of many real life

Figure 1. Split plane architecture of the CY7C361.

state machines, this form of condition decoder is optimal for a very fast realization of conditions to state macrocells [8]. The CDEC gates require a new method to minimize Boolean functions to Sum of CDEC gates expressions. In [28] the problem of quasi-optimum minimization of multi-output incompletely specified Boolean functions using Sum of CDEC gates is formulated and solved.

The inputs to each CDEC gate come from all external inputs to the device and from a number of feedback connections from the macrocells outputs. As it can be seen in Figure 2, the feedback paths are realized as short wires being very close to the associated macrocell. The segmented interconnection array allows only certain feedback connections from the output of a macrocells i to the inputs of the CDEC-gates that trigger a macrocell j . The matrix representation of the interconnection array that shows exactly which macrocell i has an output connections to a macrocell j will be shown in Chapter IV. We will see that there are 8 macrocells each having

Figure 2. Architecture of the CY7C361.

output connections to all 32 macrocells, 8 macrocells with outputs to 16 macrocells, and 16 macrocells with outputs to only 8 macrocells. According to the output availability, the macrocells of the CY7C361 can be classified into the three different groups: (1) global, (2) intermediate, and (3) local.

The segmentation of the interconnection array results in a highly restricted connectivity between the macrocells. However, the segmentation is necessary in order to avoid long feedback paths that would decrease the speed of the device. There is always a tradeoff between speed and connectivity in PLDs. For the designers

of the CY7C361 the achievement of high speed was the major goal. Therefore, the CY7C361 is one of the fastest EPLDs and the maximal toggle rate is 120 MHz.

Additional to the feedback connections between macrocells, physical adjacent macrocells can be connected by *C_IN chains*. A C_IN chain is a very short hard-wired connection that allows also the realization of conditions to state macrocells.

After we have explained that each state macrocell of the CY7C361 can be triggered by CDEC gate expressions or C_IN chain signals, the behavior of the different types of the state macrocells can be described. As it was mentioned before, there are three different programmable configurations for each macrocell: (1) START, (2) TERMINATE, and (3) TOGGLE.

- (1) START: When the CDEC expression is “1” the output of the START cell goes to “1” exactly for one clock cycle;
- (2) TERMINATE: When the C_IN signal is “1”, the output of the TERMINATE cell goes to “1” in the next clock cycle, and it goes back to “0” when the CDEC expression becomes “1”.
- (3) TOGGLE: When the C_IN signal or the CDEC expression becomes “1”, the output of the TOGGLE cell will toggle for the next state.

Each macrocell of the type TOGGLE has a programmable input connection from a *local* and *global resets* signal. The global and local resets are used to set the TOGGLE cells to a defined state. There is only one global reset signal available on the chip. It can trigger all 32 macrocells. The number of available local reset signals on the chip is 8. One single local reset has output connections to at most four physically adjacent macrocells, that means, it can trigger only these four macrocells. Hence, the macrocell array of the CY7C361 is divided into 8 blocks of four adjacent

macrocells, so called *local reset groups* $LRG_1, \dots, LRG_8$. Each local reset group LRG_i is associated with exactly one local reset plr_i . All TOGGLE cells within LRG_i must be triggered by the same local reset. A local reset group consists of two local, one intermediate, and one global macrocells. Figure 3 shows the array of local reset groups of the CY7C361 and one single local reset group LRG_1 in detail. The arrow associated with plr_i means, that the local reset plr_i triggers the macrocells configured as TOGGLE that are included in the local reset group LRG_i .

Figure 3. Array of Local Reset Groups of the CY7C361.

The local, intermediate, and global macrocells can be defined in terms of their output availability to macrocells included in certain local reset groups. The macrocell i being in the LRG_k is called:

- local, if macrocell i has outputs only to macrocells that are also in LRG_k , in

LRG_{k+1} (if k is odd), or in LRG_{k-1} (if k is even).

- intermediate, if macrocell i has outputs only to macrocells that are in $LRG_1, \dots, LRG_4$ (if $k \leq 4$) or in $LRG_5, \dots, LRG_8$ (if $k \geq 5$).
- global, if macrocell i has outputs to macrocells in all local reset groups $LRG_1, \dots, LRG_8$.

III.2 DEVELOPMENT SYSTEM FOR THE CY7C361

A VHDL based optimizing compiler WARP1 [26] was developed for the *CY7C361* and other PLD devices from Cypress [22]. The user can describe, in a subset of VHDL, either single or multiple communicating parallel state machines, logic functions, and specialized blocks (such as counters). This high-level description is then compiled, optimized, placed and routed into a form suitable for the JEDEC assembler and graphically-based unit-delay functional simulator. An optional output file produced by the system contains a register-transfer level structural VHDL description consisting of primitives which include the START, TERMINATE, and TOGGLE macrocells. This netlist file may be used to verify the synthesis process, or can be modified by the user to tune the design, and then fed back into the system. It contains the macrocells of the chip and their configurations, as well as the connections between them. Each macrocell of the netlist has to be mapped to exactly one physical macrocell on the chip, such that all connections are realizable with the limited routing resources of the chip. The physical design is accomplished in the present system by a placement and routing tool (Cypress Fitter) based on general shift-placement approach [17]. The netlist is also the input for the new partition-based fitting algorithm (PABFIT) which is presented in this thesis.

CHAPTER IV

ARCHITECTURE CONSTRAINTS FOR THE FITTING ALGORITHM

Chapter II presented placement and routing techniques for EPLDs/FPGAs. It was concluded, that for special chip architectures like the Cypress CY7C361 placement and routing needs to be performed in one step. One possibility to solve the placement problem is a fitting approach. As reviewed in Chapter I the fitting problem consists of finding an assignment of symbolic macrocells of a netlist to the physical resources of the chip, subject to the architecture constraints. The netlist represents the circuit that has to be realized. In the case of the CY7C361 it contains all information about the connections among the cells and the C_IN chains. In addition, we need information about the local reset signals.

The physical resources of the CY7C361 chip, the 32 macrocells, the global and local reset signals, and the possible connections between them is represented by a matrix representation. The matrix is called Interconnection Matrix. The C_IN chains between physically adjacent macrocells are represented by the consecutive row and column numbers. The Interconnection Matrix of the CY7C361 is shown in Figure 4.

The rows and columns p_i of the Interconnection Matrix represent the physical macrocells on the chip, that means the locations where the elements of the netlist and the resets can be placed. A “1” in Figure 4 indicates that the macrocell p_i (row i) has an output connection to the macrocell p_j (column j). Elements of the netlist that are neighbors in C_IN chains have to be placed to adjacent macrocells p_i and

Figure 4. Interconnection Matrix of the CY7C361.

p_{i+1} . The first 32 rows and 32 columns $p_1, \dots, p_{32}$ are the state macrocells of the chip that can be configured as (1) START-, (2) TERMINATE-, or (3) TOGGLE.

The row and column p_{33} corresponds to the *global reset* of the chip and the rows and columns $plr_{34}, \dots, plr_{41}$ correspond to the *local resets*. The global and local resets can be used to trigger the macrocells that are configured as TOGGLE. Each local reset plr_i can trigger at most four macrocells that are included in the local reset group LRG_i associated with plr_i . The local reset group LRG_1 contains the macrocells $p_1, \dots, p_4$, LRG_2 contains $p_5, \dots, p_8$, etc.

Connectivity restrictions among the physical macrocells can be observed by analyzing the Interconnection Matrix in Figure 4, They can be used to limit the solution space of the fitting algorithm. The connectivity restrictions can be expressed by the following classification of the macrocells.

According to the availability of the output of a state macrocell $p_i \in \{p_1, \dots, p_{32}\}$ at the input of state macrocell $p_j \in \{p_1, \dots, p_{32}\}$ ($i \neq j$), macrocell p_i can be defined as one of the following types:

Definition IV. 1: The macrocell p_i is called *global* if its output is available at all other 32 macrocells.

Definition IV. 2: The macrocell p_i is called *intermediate* if its output is available at 16 other macrocells, belonging to one of the two 16 macrocell blocks.

Definition IV. 3: The macrocell p_i is called *local* if its output is available at 8 other macrocells, belonging to one of the four 8 macrocell blocks.

The CY7C361 contains 8 global, 8 intermediate and 16 local macrocells. As it can be seen in Figure 4, p_4 is a global, p_3 an intermediate, and p_1 a local macrocell. For the purpose of visualizing the partitioning properties of this architecture, the rows of the Interconnection Matrix are reordered accordingly to the above classifi-

cations of the macrocells.

The new representation of the state macrocell part of the Interconnection Matrix ($p_1, \dots, p_{32}$), given in Figure 5, shows the first-level partitioning properties of the Interconnection Matrix. The first-level partitioning separates the global macrocells from the intermediate and local macrocells and classifies the intermediate/local macrocells into two groups that have no possible connections among each other.

Figure 5. First-level partitioning of the Interconnection Matrix.

The global macrocells grouped in the partition P_3 consisting of the partitions

P_{31} and P_{32} , denoted by $P_3 = P_{31} \cup P_{32}$. The partitions P_1 and P_2 contain the intermediate and the local macrocells. The global macrocells in partition P_3 have outputs available at all other macrocells in P_3 and at all macrocells in P_1 or P_2 . The macrocell of partition P_1 have outputs available only at macrocells in the same partition P_1 or in P_{31} . Analogously, the macrocells in partition P_2 have outputs available only at macrocells in P_2 or in P_{32} .

The partition P_1 and P_2 in Figure 5 are partitioned further analogously to the first-level partitioning of the Interconnection Matrix. The second-level partitioning properties of the Interconnection Matrix is shown in Figure 6. In the second level, the intermediate macrocells in the partition P_1 or P_2 are separated from the local macrocells, and the local macrocells are classified into two groups (two groups in P_1 and two groups in P_2) that have no connections among each other. The partitioning of P_1 and P_2 is identical. Therefore, only the partitioning of P_1 is explained in the following.

The intermediate macrocells are grouped in the partition $P_{13} = P_{131} \cup P_{132}$. The partitions P_{11} and P_{12} contain the local macrocells. The intermediate macrocells in partition P_{13} have outputs available at all other macrocells in P_{13} as well as at all macrocells in P_{11} or P_{12} . The macrocell of partition P_{11} have available outputs only at macrocells in the partition P_{11} or in P_{131} . Analogously, the macrocells in partition P_{12} have available outputs only at macrocells in P_{12} or in P_{132} .

Figure 7 shows both, the first- and second-level partitioning of the Interconnection Array. As in Figure 5, the global macrocells in Figure 7 are grouped in the partition P_3 .

Partition P_3 consists of the partitions P_{311} , P_{312} , P_{321} , P_{322} denoted by:

Figure 6. Second-level partitioning of the Interconnection Matrix.

$$P_3 = P_{31} \cup P_{32} = P_{311} \cup P_{312} \cup P_{321} \cup P_{322} \quad (1)$$

The macrocells in partitions P_1 and P_2 illustrated in Figure 5 are grouped further to the partitions P_{11} , P_{12} , P_{21} , P_{22} , P_{13} , and P_{23} according to Figure 6. The intermediate macrocells are grouped in P_{13} and P_{23} :

$$P_{13} = P_{131} \cup P_{132} \quad P_{23} = P_{231} \cup P_{232} \quad (2)$$

The local macrocells are grouped in the partitions P_{11} , P_{12} , P_{21} , and P_{22} . We introduce the index α to be able to refer to a single partition P_α .

According to the Interconnection Matrix, the following properties can be given for the number of physical macrocells in a partition P_α (denoted by $|P_\alpha|$):

Figure 7. First- and second-level partitioning of the Interconnection Matrix.

$$|P_{131}| = |P_{132}| = |P_{231}| = |P_{232}| = |P_{311}| = |P_{312}| = |P_{321}| = |P_{322}| = 2 \quad (3)$$

$$|P_{11}| = |P_{12}| = |P_{21}| = |P_{22}| = 4 \quad (4)$$

The first- and second-level partitioning properties that describe the connectivity restrictions among the state macrocells have been illustrated in Figure 5 and 7. These architecture restrictions are called *state macrocell connectivity restrictions* and are defined in Definition 4.

Definition IV. 4: *State macrocell connectivity restrictions* are the following architecture restrictions imposed on the connectivity between the state macrocells of the CY7C361:

- C-IN chains are possible only between adjacent macrocells p_i and p_{i+1}
- The outputs of the macrocells in P_1 are available at macrocells which are in $P_1 \cup P_{31}$ only. The outputs of macrocells in P_2 are available at macrocells in $P_2 \cup P_{31}$ only (first-level partitioning properties).
- The outputs of the macrocells in P_{11} are available at macrocells which are in $P_{11} \cup P_{131} \cup P_{311}$ only. The outputs of macrocells in P_{12} are available at macrocells in $P_{12} \cup P_{132} \cup P_{312}$ only (second-level partitioning properties of P_1).
- The outputs of the macrocells in P_{21} are available at macrocells which are in $P_{21} \cup P_{231} \cup P_{321}$ only. The outputs of macrocells in P_{22} are available at macrocells in $P_{22} \cup P_{232} \cup P_{322}$ only (second-level partitioning properties of P_2).

In addition to the state macrocell connectivity restrictions there are global and local reset restrictions. These are restrictions imposed on the availability of the output of state macrocells $p_1, \dots, p_{32}$ to the inputs of the global (p_{33}) and local resets $plr_{34}, \dots, plr_{41}$. They include also the restrictions imposed on the availability of the output of local resets $plr_{34}, \dots, plr_{41}$ to the inputs of state macrocells $p_1, \dots, p_{32}$. The *global and local reset restrictions* are defined in Definition 5 and 6.

Definition IV. 5: The *global reset restriction* is the following architecture constraint of the CY7C361 chip:

- The global reset can only be triggered from macrocells p_i which are in $P_{11} \cup P_{13} \cup P_3$.

The restrictions on the output availability of state macrocells at the inputs of local resets are identical to the restrictions on their availability at the inputs of state macrocells. Figure 8 the first- and second-level partitioning of the state macrocells $p_1, \dots, p_{32}$ according to their output availability at local resets. The partitions in Figure 8 are the same as in Figure 7. Only state macrocells $p_i \in P_{31} \cup P_{32}$ can trigger all local resets. State macrocells $p_i \in P_1$ can trigger only local resets $plr_{34}, \dots, plr_{37}$, and state macrocells $p_j \in P_2$ can trigger only local resets $plr_{38}, \dots, plr_{41}$. The partitions P_1 and P_2 can be partitioned further analogously to the partitions of the state macrocells.

Definition IV. 6: The *local reset restrictions* are the following architecture constraints of the CY7C361 chip:

- A local reset plr_i can trigger 4 state macrocells. These four state macrocells are in the LRG_i associated with plr_i . Only TOGGLE cells can be triggered by the local reset and all TOGGLE cells within a local reset group LRG_i are triggered from the same local reset plr_i .
- State macrocells in P_3 have available outputs at all local resets $lrp_{34}, \dots, lrp_{41}$.
- State macrocells in P_{13} have available outputs at the local resets $lrp_{34}, \dots, lrp_{37}$. State macrocells in P_{23} have available outputs at the local resets $lrp_{38}, \dots, lrp_{41}$,
- State macrocells in P_{11} , P_{12} , P_{21} , and P_{22} have available outputs at the groups

Figure 8. Output availability of state macrocells to local resets.

of local resets $\{lrp_{34}, lrp_{35}\}$, $\{lrp_{36}, lrp_{37}\}$, $\{lrp_{38}, lrp_{39}\}$, and $\{lrp_{40}, lrp_{41}\}$, respectively.

The state macrocell connectivity restrictions and the global and local reset restrictions are used in PABFIT to exclude unfeasible placements. Unfeasible placements are assignments of state macrocells of the netlist to physical macrocells such that the connections between the physical macrocells are not realizable on the chip. Incorporating these restrictions in the partitioning stage of PABFIT reduces the solution space of the fitting problem very effectively. The restrictions are verified

again as the last step of the algorithm to check if the solution that was found by PABFIT is actually feasible.

CHAPTER V

DEFINITIONS AND PROBLEM FORMULATION

The fitting problem can be stated as a graph monomorphism problem [17]. The netlist that has to be mapped on the chip is represented as a *symbolic graph* G_S . The physical resources are represented as a *physical connectivity graph* G_P . The graph monomorphism problem consists of finding an embedding of G_S in G_P and is defined as follows [29].

Definition V. 1: Suppose $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are two graphs. If $\Pi : V_G \rightarrow V_H$ is an injection such that $\langle v_{Gi}, v_{Gj} \rangle \in E_G$ implies that $\langle \Pi(v_{Gi}), \Pi(v_{Gj}) \rangle \in E_H$ for any edge $\langle v_{Gi}, v_{Gj} \rangle$ of G , then Π is called monomorphism from G to H . If there exists a monomorphism from G to H , then we say G is embeddable in H .

The previous Chapter introduced the partitioning properties of the Interconnections Matrix of the Cypress CY7C361. The physical connectivity graph G_P that represents the Interconnection Matrix has exactly the same partitioning properties. They are stated in Chapter V.1. The basic fitting algorithm is based on the property that a netlist, represented by the symbolic graph G_S can be mapped to the chip resources only if G_S has the same partitioning properties as G_P .

V.1 REPRESENTATION OF THE CHIP RESOURCES

The state macrocell part $(p_1, \dots, p_{32})$ of the Interconnection Matrix of the CY7C361 chip in Figure 4 can be represented as a directed graph

$G_P = (V_P, EN_P, EC_P)$, denoted *physical connectivity graph*.

Definition V. 2: The *physical connectivity graph* $G_P = (V_P, EN_P, EC_P)$ consists of the ordered vertex set V_P , the set of directed normal edges EN_P , and the directed chain-edge set EC_P defined as follows:

- $V_P = \{v_{pi} \mid v_{pi} \text{ represents the state macrocell } p_i\}$
- $EN_P = \{en_{pi} = \langle v_{pk}, v_{pl} \rangle \mid en_{pi} \text{ represents the possible output connection from } v_{pk} \text{ to } v_{pl}\}$
- $EC_P = \{ec_{pi} = \langle v_{pj}, v_{pj+1} \rangle \mid ec_{pi} \text{ represents the possible C-IN-chain from } v_{pj} \text{ to } v_{pj+1}\}$

Definition V. 3: A directed chain-edge $ec_{pi} = \langle v_{pj}, v_{pj+1} \rangle$ is called an *output chain-edge* of the vertex v_{pj} and an *input chain-edge* of v_{pj+1} .

The physical graph G_P for the CY7C361 chip architecture consists of $|V_P| = 32$ vertices, $|EN_P| = 512$ normal-edges, and $|EC_P| = 31$ chain-edges. It describes completely how the 32 state macrocells are connected (C-IN-chains and output connections). The physical adjacency of the macrocell is represented by the label $i = 1, \dots, 32$ of the vertices $v_{pi} \in V_P$.

The physical global and local resets are not included in the physical connectivity graph. As it was shown in Figure 3 in Chapter III.1, a local reset lrp_i can trigger only TOGGLE cells that are included in the local reset group LRG_i associated to lrp_i . A local reset group consists of four physically adjacent macrocells. The output availability of local resets at the inputs of state macrocells has not the same partitioning properties as the output availability of the state macrocells to local resets or other state macrocells. However, the output availability of state macrocells

at the inputs of the local resets exhibit these partitioning properties (Figure 8). The latter are considered in the partitioning stage of the PABFIT algorithm by adding an additional edge to the graph representation of the netlist.

To illustrate the two-level partitioning properties of the physical connectivity graph G_P , we define the subgraphs G_{P_α} of G_P . A subgraph G_{P_α} represents the physical macrocells that are included in the partitions P_α according to Figure 7 and the possible connections among them.

Definition V. 4: The subgraph $G_\alpha = (V_{P_\alpha}, EN_{P_\alpha}, EC_{P_\alpha})$ of G_P consists of the vertex set V_{P_α} , the normal edge set EN_{P_α} , and the chain-edge set EC_{P_α} of G_{P_α} defined as follows:

- $V_{P_\alpha} = \{v_{pi} \in V_P \mid v_{pi} \text{ is in partition } P_\alpha\}$
- $EN_{P_\alpha} = \{\langle v_{pi}, v_{pj} \rangle \mid v_{pi}, v_{pj} \in P_\alpha \ \& \ \langle v_{pi}, v_{pj} \rangle \in EN_P\}$
- $EC_{P_\alpha} = \{\langle v_{pi}, v_{pi+1} \rangle \mid v_{pi}, v_{pi+1} \in P_\alpha \ \& \ \langle v_{pi}, v_{pi+1} \rangle \in EC_P\}$

According to this definition, the subgraph $G_{P_3} = G_{P_{311}} \cup G_{P_{312}} \cup G_{P_{321}} \cup G_{P_{322}}$ contains the global macrocells. The subgraphs $G_{P_{13}} = G_{P_{131}} \cup G_{P_{132}}$ and $G_{P_{23}} = G_{P_{231}} \cup G_{P_{232}}$ contain the intermediate macrocells. The local macrocells are included in the subgraphs $G_{P_{11}}$, $G_{P_{12}}$, $G_{P_{21}}$, and $G_{P_{22}}$. Each of the subgraphs $G_{P_{11}}$, $G_{P_{12}}$, $G_{P_{21}}$, and $G_{P_{22}}$ consists of four vertices $v_{pi} \in V_P$. Each of the subgraphs $G_{P_{131}}$, $G_{P_{132}}$, $G_{P_{231}}$, $G_{P_{232}}$, $G_{P_{311}}$, $G_{P_{312}}$, $G_{P_{321}}$, and $G_{P_{322}}$ consist of two vertices $v_{pi} \in V_P$ (compare Equations (1) through (4) in Chapter IV).

As described in Chapter IV the first-level partitioning separates the global macrocells from the intermediate/local macrocells. The second-level separates the intermeditate macrocells from the local ones. As a result, the Interconnection Matrix is partitioned according to Figure 7 into the partitions P_α . Now, we represent these

partitions by the subgraphs $G_{P\alpha}$. There are no normal edge connectivity restrictions among vertices v_{pi} within a single subgraphs. However, the connectivity among vertices v_{pi} of different subgraphs is limited according to the partitioning properties of the Interconnection Array. In the following the connectivity restrictions among vertices in different subgraphs $G_{P\alpha}$ are illustrated by representing each subgraph $G_{P\alpha}$ as a single vertex of a *super graph* $G_T = (V_T, EN_T, EC_T)$. The set of directed normal edges EN_T and the set of directed chain-edges EC_T represent possible normal or chain-edge connections between the state macrocells that are included in the subgraphs $G_{P\alpha}$. The subgraphs G_α themselves are represented by single vertices $v_{Ti} \in V_T$.

Definition V. 5: The *super graph* $G_T = (V_T, EN_T, EC_T)$ is defined as follows:

- $V_T = \{v_{Ti} \mid v_{Ti} \text{ represents the subgraphs } G_{P\alpha}\}$
- $EN_T = \{\langle v_{Ti}, v_{Tj} \rangle = \{\langle v_{pk}, v_{pl} \rangle \in EN_P \mid v_{pk} \in v_{Ti} \text{ and } v_{pl} \in v_{Tj}\}\}$
- $EC_T = \{\langle v_{Ti}, v_{Tj} \rangle \mid \langle v_{pk}, v_{pk+1} \rangle \in EC_P \text{ and } v_{pk} \in v_{Ti} \text{ and } v_{pk+1} \in v_{Tj}\}$

The first- and second-level partitioning of the physical connectivity graph G_P , illustrated with the supergraph G_T , is shown in Figure 9 and 10, respectively. The thin arrows correspond to the normal edges of EN_T and the bold lines to the chain-edges of EC_T . A thin edge covers several normal edges of the physical connectivity graph. A bold line represents only one single chain-edge of G_P . Comparing the Figures 9 and 10 for the first- and second-level partitioning hierarchy it can be seen that both partitioning levels are identical.

Figure 9 shows the first-level partitioning of the physical connectivity graph G_P into the two groups of two subgraphs $G_{P1} \cup G_{P31}$ and $G_{P2} \cup G_{P32}$, respectively. The cut that partitions the graph G_P is represented by axis 1. The first-level par-

Figure 9. First-level partitioning of G_P .

tition separates the global macrocells that are in G_{P31} and G_{P32} from the intermediate/local macrocells that are in G_{P1} and G_{P2} . There are no possible connections among vertices in G_{P1} and G_{P2} . The *first-level partitioning properties* are stated below. They follow directly from the state macrocell connectivity restrictions extracted in Chapter IV.

Definition V. 6: The *first-level partitioning properties* of the physical connectivity graph G_P can be stated as follows:

- There are no connections (neither normal nor C-IN-chains) between subgraphs G_{P1} and G_{P2} .
- There exist only output connections from subgraph G_{P31} to G_{P2} and not vice versa.
- There exist only output connections from subgraph G_{P32} to G_{P1} and not vice versa.

versa.

- Vertices $v_{pi} \in G_{P2}$ may only have *one* C-IN-chain connection from $v_{pi-1} \in G_{P31}$.
- The number of vertices $v_{pi} \in G_{P\alpha}$ is given by the number of elements in the corresponding partition P_α (Equations (1) through (4) in Chapter IV).

In other words: The cut represented by axis 1 in Figure 9 is allowed to cut only normal edges $en_{pi} \in G_P$ that are output edges incident to vertices $v_{pj} \in G_{P31} \cup G_{P32}$ and only *one* chain-edge $ec_{pk} \in G_P$ such that $ec_{pk} \in \{\langle v_{pl}, v_{pl+1} \rangle \mid v_{pl} \in G_{P31} \text{ and } v_{pl+1} \in G_{P2}\}$.

The result of the first-level partitioning of G_P are the two groups of subgraphs $\{G_{P1}, G_{P31}\}$ and $\{G_{P2}, G_{P32}\}$. They are both partitioned further analogously to the first-level partition. The second-level partition of the group of subgraphs $G_{P1} \cup G_{P31}$ is shown in Figure 10a. The results are two groups of two subgraphs $G_{P11} \cup G_{P131} \cup G_{P311}$ and $G_{P12} \cup G_{P132} \cup G_{P312}$, indicated by axis 2. The second-level partitioning of $\{G_{P2}, G_{P32}\}$ is shown in Figure 10b.

Because of the symmetry of the Interconnection Matrix the partitioning of $G_{P2} \cup G_{P32}$ is analogous to the partitioning of $G_{P1} \cup G_{P31}$. Therefore, only the partitioning of $G_{P1} \cup G_{P31}$ is explained in the following.

In the first-level partitioning, the global macrocells have been separated from the intermediate/local ones. The global macrocells are included in the subgraph G_{P31} . The second-level partition separates the intermediate macrocells that are in G_{P131} and G_{P132} from the local macrocells that are in G_{P11} and G_{P12} . Therefore, G_{P31} has to be partitioned further into G_{P311} and G_{P312} according to the availability of the outputs of the local macrocells at the inputs of the global ones. The local

Figure 10. Second-level partitioning of $G_{P1} \cup G_{P31}$ and $G_{P2} \cup G_{P32}$.

macrocells in G_{P11} have outputs only to macrocells in G_{P311} . The local macrocells in G_{P12} have outputs only to macrocells in G_{P312} . The *second-level partitioning properties* are stated below. They follow directly from the state macrocell connectivity restrictions found in Chapter IV.

Definition V. 7: The *second-level partitioning properties* of the group of subgraphs $G_{P1} \cup G_{P31}$ can be stated as follows:

- There are no connections (neither normal nor C-IN-chains) between subgraphs G_{P11} and G_{P12} .
- There exist only output connections from the group of subgraphs $G_{P131} \cup G_{P311}$ to G_{P12} and not vice versa.
- There exist only output connections from the group of subgraphs $G_{P132} \cup G_{P312}$ to G_{P11} and not vice versa.
- Vertices $v_{pi} \in G_{P12}$ may only have *one* C-IN-chain connection from $v_{pi-1} \in$

G_{P311} .

- The number of vertices $v_{pi} \in G_{P\alpha}$ is given by the number of elements in the corresponding partition P_α (Equations (1) to (4)).

In other words: The cut represented by axis 2 in Figure 10a is allowed to cut only normal edges $en_{pi} \in G_P$ that are output edges incident to vertices $v_{pj} \in G_{311} \cup G_{312} \cup G_{P131} \cup G_{P132}$ and only *one* chain-edge $ec_{pk} \in G_P$ with $ec_{pk} \in \{\langle v_{pl}, v_{pl+1} \rangle \mid v_{pl} \in G_{P311} \text{ and } v_{pl+1} \in G_{P12}\}$.

V.2 REPRESENTATION OF THE NETLIST

The netlist of macrocells is represented by the directed *symbolic graph* $G_S = (V_S, EN_S, EC_S)$. The set of vertices V_S , the set of directed normal edges EN_S , and the set of directed chain-edges EC_S are defined below:

Definition V. 8: The *symbolic graph* $G_S = (V_S, EN_S, EC_S)$ consists of the vertex set V_S , the set of directed normal edges EN_S , and the set of directed chain-edges EC_S defined as follows:

- $V_S = \{v_{Si} \mid v_{Si} \text{ represents the state macrocell of the netlist}\}$
- $EN_S = \{en_{Si} = \langle v_{Sk}, v_{Sl} \rangle \mid en_{Si} \text{ represents an output connection from } v_{Sl} \text{ to } v_{Sk}\}$
- $EC_S = \{ec_{Si} = \langle v_{Sj}, v_{Sj+1} \rangle \mid ec_{Si} \text{ represents a C_IN-chain from } v_{Sj} \text{ to } v_{Sj+1}\}$

In addition to the netlist there are connections among the local/global resets and state macrocells. The local and global resets are not included in the physical connectivity graph. Since the fitting problem is stated as a graph monomorphism

problem they cannot be included in the symbolic graph. However, the output availability of the state macrocells at the inputs of other state macrocells and at the inputs of local resets exhibits the same partitioning properties. Let us assume that a state macrocell v_{Sn} triggers a local reset lr_i , and lr_i triggers the macrocell v_{Sm} . Then the output of v_{Sn} has to be available at the input of v_{Sm} according to the Interconnection Matrix, Figure 4, in order to find a feasible placement.

Therefore, the input restrictions of the local resets can be considered in the partitioning stage of the fitting algorithm, by adding additional normal edges $\{en_{Sk}, \dots, en_{Sl}\}$ not included in the netlist to EN_S . If the state macrocell $v_{Sm} \in V_S$ triggers a local reset and the same local reset triggers another state macrocell $v_{Sn} \in V_S$ then an additional normal edge $en_{Sk} = \langle v_{Sm}, v_{Sn} \rangle$ is added to the set of normal edges EN_S (Figure 11). The local resets themselves and their in- and output connections are not included in the Symbolic Graph G_S . The additional edges are added in order to consider the input restrictions of the local resets during the partitioning phase. Therefore, they are considered only in the partitioning stage of the algorithm and removed later, when the Fitter starts the Physical Placement.

Figure 11. Additional normal edge due to input restrictions of local resets.

V.3 FORMULATION OF THE FITTING PROBLEM

The fitting problem for the Cypress CY7C361 can be formulated as follows:

Problem Formulation V. 1 Given a directed physical connectivity graph $G_P = (V_P, EN_P, EC_P)$ and a directed symbolic graph $G_S = (V_S, EN_S, EC_S)$ determine, if the symbolic graph G_S is monomorphic to the physical connectivity graph G_P , and if so, find the mapping (monomorphism Π) of the symbolic vertices $v_{Si} \in V_S$ to the physical vertices $v_{Pi} \in V_P$, such that no global and local reset restrictions are violated.

The graph monomorphism problem is of high complexity [29]. To reduce this complexity we take advantage of the partitioning properties of the physical connectivity graph G_P . A symbolic graph G_S must exhibit the same partitioning properties as the physical connectivity graph G_P . A two-level partitioning is performed on the symbolic graph G_S , subject to the first- and second-level partitioning properties of the physical connectivity graph G_P . That means the symbolic graph G_S is examined for subgraphs $G_{S\alpha}$ that can be assigned to the physical partition P_α (represented by G_α) without violating the first- and second-level partitioning properties. The final assignment of symbolic vertices to the physical locations (state macrocells and reset cells) is performed in the Physical Placement stage. In the last step, before the PABFIT displays the solution, it is verified, if no connectivity and reset restrictions are violated.

CHAPTER VI

DESCRIPTION OF THE FITTING ALGORITHM

Given the symbolic graph $G_S = (V_S, EN_S, EC_S)$, representing the netlist that has to be fitted to the CY7C361 chip, the two-level partitioning algorithm PABFIT is used to examine if G_S has the same partitioning properties as the physical connectivity graph G_P according to Figure 9 and 10. During the partitioning stage, G_S is decomposed to subgraphs that are assigned to partitions P_α . Since the assignments are checked for violations of the first- and second-level partitioning properties, unfeasible assignments are excluded in an early stage of the algorithm. All possible subgraphs and all possible assignments have to be determined until a feasible solution of the fitting problem is found. By exploring all possibilities we are able to claim that the algorithm proposed in this thesis is exact. It finds a solution if there is one, or it can verify that no feasible solution exists.

Figure 12 shows the flowchart of the PABFIT algorithm. It explains how unfeasible partitions are excluded and when the final placement stage is reached. The first level of the partitioning algorithm is called *First Partition*, and the second level *Second Partition*. The final physical placement is called *Physical Placement*.

According to Figure 12, PABFIT starts with the P_3 Assignment. The physical macrocells of partition P_3 are global according to their output availability at the inputs of other physical macrocells. Therefore, there are no output connectivity restrictions on the symbolic vertices $v_{S_i} \in V_S$ that are assigned to P_3 . The algorithm determines all possible P_3 assignments. Each P_3 Assignment represents one

Figure 12. Flowchart of the PABFIT algorithm.

possibility of choosing a subgraph G_{S3} from G_S . The P_3 Assignment is explained more detailed in Chapter VI.1.

For each possible P_3 Assignment, PABFIT determines a First Partition of the symbolic graph G_S if possible. That means PABFIT tries to find subgraphs G_{S1} and G_{S2} that can be assigned to the partitions P_1 and P_2 without violating the first-level partitioning properties. For a given P_3 Assignment, all possible First Partitions are determined, meaning, all possible subgraph G_{S1} and G_{S2} are determined. If PABFIT cannot find any First Partition for a given P_3 Assignment, the algorithm backtracks to the P_3 Assignment. The “condition” in the flowchart checks if all possible P_3 Assignments are explored or not. If there is a new P_3 Assignments that has not been explored yet, PABFIT determines this assignment and performs again the First Partition. The First Partition is explained in Chapter VI.2.

As a result of the each possible First Partition, the graph G_S is divided into two groups of two subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$ assigned to the corresponding partitions. The symbolic vertices that are assigned to P_3 become global macrocells, and the symbolic vertices that are assigned to P_1 or P_2 become intermediate and local macrocells. At the beginning of the PABFIT algorithm the local macrocells are separated from the intermediate/local macrocells. The next step is to separate the intermediate macrocells from the local ones. Therefore, a P_{13} and P_{23} Assignment is performed. In the P_{13} Assignment, all possibilities of assigning vertices $v_{Si} \in G_{S1}$ to partition P_{13} are determined. Analogously, all possibilities of assigning vertices $v_{Si} \in G_{S2}$ to partition P_{23} are determined in the P_{23} Assignment. Thus, all possibilities of choosing a subgraph G_{S13} or G_{S23} from G_{S1} or G_{S2} are determined based on the given First Partition. The P_{13} and P_{23} Assignment is explained at the beginning of Chapter VI.4.

For each possible P_{13} and P_{23} Assignment, PABFIT determines a Second Partition of the groups subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$, respectively. Because

of the symmetry of the Interconnection Matrix of the CY7C361, the Second Partition of $G_{S2} \cup G_{S32}$ is identical to the Second Partition of $G_{S1} \cup G_{S31}$. Therefore, only the partitioning of $G_{S1} \cup G_{S31}$ is explained.

In the Second Partition, PABFIT tries to find a partitioning of the subgraph $G_{S1} \cup G_{S31}$ to the group of subgraphs $G_{S11} \cup G_{S131} \cup G_{S311}$ and $G_{S12} \cup G_{S132} \cup G_{S312}$, such that no second-level partitioning properties are violated. Like in the First Partition, all possible Second Partitions of $G_{S1} \cup G_{S13}$ that do not violate the second-level partitioning properties are determined by PABFIT. If no possible Second Partition exists, PABFIT determines new P_{31} Assignments as long as there are new possible P_{31} Assignments to explore (“condition” in flowchart).

If PABFIT is able to find a Second Partition, the subgraph $G_{S1} \cup G_{S13}$ is divided into two groups of subgraphs $G_{S11} \cup G_{S131} \cup G_{S311}$ and $G_{S12} \cup G_{S132} \cup G_{S312}$. The subgraph $G_{S2} \cup G_{S32}$ is divided into $G_{S21} \cup G_{S231} \cup G_{S321}$ and $G_{S22} \cup G_{S232} \cup G_{S322}$. For each possible Second Partition, it is verified if the local reset output restrictions are not violated.

In the partitioning stages of the PABFIT, explained above, all the possibilities of assigning sets of connected vertices $v_{Si} \in V_S$ to the different partitions have been determined. The assignments do not violate any of the first- nor second-level partitioning properties. That means no feasible solution of the fitting problem has been excluded. However, in the First and Second Partition the mapping possibilities of a single symbolic vertex $v_{Si} \in V_S$ to a physical macrocell on the chip have been limited efficiently.

The last stage of the PABFIT algorithm is the *Physical Placement*, which maps each state vertex of the symbolic graph $G_S(V_S, E_S)$, grouped in appropriate partitions, to exactly one physical state macrocells on the chip. Once the state

macrocell mapping is completed, the algorithm performs the mapping of the global and local resets. The Physical Placement is described in Chapter VI.5.

If a feasible mapping of each symbolic vertex $v_{S_i} \in G_S$ to a single physical macrocell can be determined in the Physical Placement stage, PABFIT has found one feasible solution of the fitting problem. By feasible mapping we mean a mapping that allows the realization of all the connections between the state macrocells and reset signals of the netlist. Since all the connectivity restrictions and the local and global reset restrictions are verified during the First and Second Partition, in most cases PABFIT is able to find a feasible solution as soon as it enters the Physical Placement stage. However, if the symbolic graph contains vertices that are in C_IN chains with at least two other vertices, it can happen, that PABFIT enters the Physical Placement without finding a feasible solution. In that case a backtracking is performed and new Second Partitions are explored.

VI.1 P_3 ASSIGNMENT

As there are no output connectivity restrictions on vertices in partition P_3 , the first step of the PABFIT is to choose $m = k, \dots, 8$ symbolic vertices, from the set of vertices V_S to be assigned to partition P_3 . The chip contains 8 global and 24 intermediate/local macrocells. The number of vertices of the symbolic graph is $n = |V_S|$. The algorithm starts with assigning k vertices to the partition P_3 . The starting value k is given by Equation (5).

$$k = \begin{cases} n - 24 & n \geq 24 \\ 0 & n < 24 \end{cases} \quad (5)$$

These m vertices out of the symbolic graph G_S and their interconnections form the subgraph G_{S3} of G_S . The exact algorithm considers all possibilities to

choose m symbolic vertices out of the set of $n = |V_S|$ symbolic vertices. Therefore, the upper bound on the number of possibilities can be estimated by the following equation.

$$\text{number of possibilities} \leq \sum_{l=k}^8 \binom{n}{l} \quad (6)$$

In PABFIT, the order in which the vertices are being selected for P_3 is random and depends only on the ordering of the netlist. For a fixed value of m , the algorithm starts with assigning the first m vertices of the ordered symbolic graph V_S to P_3 . The next step is to keep the assignment of the first $m-1$ vertices, and to assign sequentially one of the vertices $v_{Sm+1}, \dots, v_n$ to P_3 . In the next step, the first $m-2$ vertices and the vertex v_{Sm+1} of the netlist are assigned to P_3 . Now, the vertices $v_{Sm+2}, \dots, v_{Sn}$ are assigned sequentially to P_3 . This procedure is repeated until all the possibilities and all values of $m = k, \dots, 8$ are explored. The order in which the vertices are being selected for P_3 can be changed, using the heuristics described in Chapter VIII.

The number of possibilities is drastically reduced if the symbolic graph contains C-IN-chains. In order to describe how C-IN chains are considered during the P_3 Assignment, the following definitions are necessary.

Definition VI. 1: A chain CH_i is the ordered set of vertices $CH_i = \{v_{Sj+m} \in V_S\}$ with $\langle v_{Sj+m}, v_{Sj+m+1} \rangle \in EC_S$ for all $m = 0, \dots, n$.

Definition VI. 2: The *chain_length* n is the number of elements in the ordered set of vertices CH_i .

Definition VI. 3: The *chain_number* $m+1$ of a vertex $v_{Sj+m} \in V_S$ is the position of

the vertex v_{Sj+m} in the ordered set CH_i .

In Figure 13, we show an example of a chain $CH_i = \{v_{S3}, \dots, v_{S8}\}$. The *chain_length* is 9 and the *chain_number* of vertex v_{S4} is 2. In Figure 13 it is assumed that the vertices v_{S4} and v_{S8} have been assigned to partition P_3 , indicated by the two arrows.

Figure 13. Chains in the P_3 Assignment.

It follows directly from the Interconnection Matrix (Figure 4) that it is necessary to assign some of the vertices $v_{Sj} \in V_S$ which are included in a chain CH_i with a *chain_length* greater than 3 to P_3 . Vertices that are neighbors in a chain have to be placed physically adjacent on the chip. Therefore, each fourth vertex from a chain has to be assigned to partition P_3 . Let us assume that the symbolic vertex $v_{Sj} \in CH_i$ is the first vertex that PABFIT chooses to be assigned to P_3 . Depending on the *chain_length* of CH_i and the *chain_number* of v_{Sj} in CH_i , the number of vertices of CH_i that have to be assigned to P_3 is given by the following equation:

$$\# \text{ of vertices} = (\text{chain_length}(CH_i) - \text{chain_number}(v_{Sj}) + 4) / 4 \quad (7)$$

In our example in Figure 13 it is assumed that v_{S4} is assigned to P_3 . Hence, the symbolic vertex v_{S8} has to be assigned to P_3 , too. The other vertices cannot be

assigned to P_3 .

As we can see from the previous example, the possible P_3 Assignments are reduced drastically, if many vertices of the symbolic graph are included in chains. Therefore, the solution space of the problem decreases strongly if the netlist contains many macrocell included in C-IN chains. This limitation is very effective, especially because it affects the first step of the algorithm. In the following the pseudocode of the P_3 Assignment is given.

```

vertex P3[8]; /* vertices assigned to P3 */
short p3; /* number of vertices assigned to P3 */
short k; /* starting value for p3 */
pos_P3()
{
    for(p3=k; p3≤8; p3++)
        det_all_poss(p3);
}
det_all_poss(p3)
short p3;
{
    while (new P3 Assignment possibilities exist) {
        determine the new possibility
        First_Partition(P3,p3);
    }
}

```

VI.2 FIRST PARTITION

In the P_3 Assignment, the symbolic vertices that are assigned to partition P_3 have been determined. These vertices form the the symbolic subgraph G_{S3} of G_S . In the First Partition, PABFIT examines the symbolic graph G_S for the group of subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$ subject to the first-level partitioning properties. The problem solved by the P_3 Assignment and the First Partition can be stated as

follows:

Problem Formulation VI. 1 Given the symbolic graph $G_S = (V_S, EN_S, EC_S)$, explore all possible sets of vertices $V_{S3} = \{v_{Si} \mid v_{Si} \in V_S\}$ ($|V_{S3}| \leq 8$) such that if the output edges incident to the vertices $v_{Si} \in V_{S3}$ are removed from the graph G_S , the resulting graph $G_R = (V_R, EN_R, EC_R)$ can be partitioned into two groups of subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$ subject to the first level partitioning properties.

The output edges of symbolic vertices assigned to P_3 can be removed because their outputs are globally available at all other physical macrocells.

In order to find the two groups of subgraphs that can be assigned to the partitions $P_1 \cup P_{31}$ or $P_2 \cup P_{32}$ the graph G_R is examined for *disconnected components* G_{Ci} .

Definition VI. 4: A *disconnected component* $G_{Ci} = (V_{Ci}, EN_{Ci}, EC_{Ci})$ of the graph $G_R = (V_R, EN_R, EC_R)$ is a subgraph of G_R with no normal nor chain-edges between vertices $v_{Sk} \in V_{Ci}$ and $v_{Sl} \notin V_{Ci}$ ($v_l \in V_R$).

Disconnected components can be assigned to the different partitions $P_1 \cup P_{31}$ or $P_2 \cup P_{32}$ because they have no normal nor chain-connections between each other. Hence, they are used to form the subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$.

In PABFIT a standard breadth-first search algorithm [1] is used in order to determine all disconnected components G_{Ci} resulting from the previously determined P_3 Assignment. The assignment of each disconnected component to the partitions $P_1 \cup P_{31}$ or $P_2 \cup P_{32}$ is done randomly. All the assignment possibilities are explored by the algorithm, such that no possible solution is excluded. At the end

of the First Partition it will be verified if the number of vertices assigned to a particular partition P_α does not exceed the maximum number given by Equations (1) through (4).

Figure 14 illustrates the procedure of finding a possible First Partition of the symbolic graph G_S . The graph G_S in Figure 14a consists of the set of vertices $\{v_{S1}, \dots, v_S\}$, the set of normal edges (represented by thin arrows), and the set of chain-edges (represented by bold lines). The direction of the chain-edges can be obtained from the vertex number of the vertices incident to the chain-edge. A chain-edge goes from a vertex v_{Si} to a vertex v_{Si+1} . Vertex v_{S4} has an input chain-edge from v_{S3} , and vertex v_{S5} has an input chain-edge from v_{S4} . It is assumed that vertex v_{S4} (the filled circle in Figure 14) has been assigned to partition P_3 during the P_3 Assignment stage of the algorithm.

As shown in Figure 14b the assignment of v_{S4} to P_3 and the removal of the output normal edges incident to v_{S4} disconnects the symbolic graph G_S . The result is the graph G_R that consists of three disconnected components G_{C1} , G_{C2} , and G_{C3} . There are no connections between any of these components. That means, they can be assigned to the partition $P_1 \cup P_{31}$ or $P_2 \cup P_{32}$. All possible assignments are determined by the algorithm. In our example we assume the assignment of the disconnected components G_{Ci} according to Figure 14b. The disconnected component G_{C1} is assigned to $P_1 \cup P_{31}$. G_{C2} , G_{C3} are both assigned to $P_{32} \cup P_2$. Another possibility would be to assign G_{C1} and G_{C2} to the partition $P_1 \cup P_{31}$ and G_{C3} to the partition $P_2 \cup P_{32}$.

When PABFIT determines the disconnected components G_{Ci} of the graph G_R it does not allow any chains-edges between different disconnected components. That means, vertices v_{Si} that belong to the same chain CH_k can not be assigned

Figure 14. Example of a First Partition.

to different partitions. As a result, they have to be included either in the group of subgraphs $G_{S1} \cup G_{S31}$ or in the group of subgraphs $G_{S2} \cup G_{S32}$. However, according to the first-level partitioning properties it is allowed that there exists *one* chain-edge between a vertex $v_{Si} \in G_{31}$ and a vertex $v_{Si+1} \in G_{S2}$. Since this chain-edge is cut by the First Partition (see axis 1 in Figure 9) the procedure of cutting this chain-edge is called *chain_cut_1*.

The concept of a *chain_cut_1* is introduced in order to reduce the number of sets of vertices that can be assigned to different partitions without excluding any possible First Partition. If we removed the output chain-edges of vertices in P_3 before, the disconnected components are determined, there would be much more set

assignment possibilities. When we do not allow a *chain_cut_1* when the disconnected components are determined and assigned to $P_1 \cup P_{31}$ or $P_2 \cup P_{32}$, we have to explore all these additional First Partitions later.

A *chain_cut_1* for the first-level partitioning of the symbolic graph G_S is defined as follows.

Definition VI. 5: A *chain_cut_1* of the chain CH_k at the *cut_vertex* $v_{S\ cut} \in V_S$ is a mapping of the symbolic vertex $v_{S\ cut}$ to the physical macrocell p_{16} . The vertices $v_{S\ cut-i}$ and $v_{S\ cut+j}$ are mapped to the single physical macrocells p_{16-i} and p_{16+j} . The values of i and j are given below:

- $i = 1, \dots, chain_number(v_{S\ cut}) - 1$
- $j = 1, \dots, chain_length(v_{S\ cut}) - chain_number(v_{S\ cut})$

These are the symbolic vertices that belong to the same chain CH_k as the vertex v_{Scut} . As a result, the set of vertices $\{v_{Sl} \mid v_{Sl} \in CH_k \mid l \leq cut\}$ is assigned to partition $P_1 \cup P_{31}$ and the set of vertices $\{v_{Sl} \mid v_{Sl} \in CH_k \mid l > cut\}$ is assigned to partition $P_2 \cup P_{32}$.

In the following it is explained how the additional First Partitions resulting from a *chain_cut_1* are determined. Based on an initial partition without a *chain_cut_1*, all possible *chain_cut_1s* are determined. A *chain_cut_1* can be performed only at vertices v_{Sj} that are mapped to the physical macrocell p_{16} . Therefore, only vertices $v_{Sj} \in G_{P_{31}}$ have to be considered to become the *cut_vertex*. Hence, each disconnected component G_{Ci} assigned to $\{P_1, P_{31}\}$ is examined for vertices v_{Sj} assigned to partition P_{31} which have an output chain-edge.

Once such a vertex is identified, it is verified if a *chain_cut_1* can actually be performed. The necessary condition is, that G_{Ci} is being disconnected by removing

the output chain-edge incident to the prospective cut_vertex. That means, the `chain_cut_1` results in disconnected components that are subgraphs of G_{C_i} . When it is verified that a `chain_cut_1` can be performed, v_{S_i} becomes the cut_vertex v_{Scut} according to Definition 5 and all the symbolic vertices that are in the same chain as v_{Scut} are mapped to physical macrocells. This condition has to be verified, because as a result of a `chain_cut_1` some vertices of G_{C_i} will become assigned to $P_1 \cup P_{31}$ and others to $P_2 \cup P_{32}$.

In order to show an example of a `chain_cut_1` we continue with our example from Figure 14 and perform a `chain_cut_1` at the disconnected component G_{C_1} . Figure 15a shows again the symbolic graph G_S . The graph G_R where the output normal edges of vertices v_{S_i} assigned to P_3 are removed is shown in Figure 15b.

In the example from Figure 15 the *chain_cut_1* can only be performed at the cut_vertex v_{S_4} , because v_{S_4} is the only one assigned to P_{31} which has an output chain-edge. When we remove this output chain-edge, G_{C_1} becomes disconnected into two disconnected components $G_{C_{11}}$ and $G_{C_{12}}$. $G_{C_{11}}$ contains the vertices $v_{S_1}, \dots, v_{S_4}$ and $G_{C_{12}}$ consists only of v_{S_5} . Therefore, based on the initial assignment of G_{C_1} to $P_1 \cup P_{31}$ and of G_{C_2}, G_{C_3} to $P_2 \cup P_{32}$, one additional First Partition can be obtained by the *chain_cut_1*. As a result, vertex v_{S_3} is mapped to physical location p_{15} , v_{S_4} to p_{16} , and v_{S_5} to p_{17} . The graph in Figure 15b consists now of 4 disconnected components. Contrary to the initial partition, the vertex v_{S_5} is now in $P_2 \cup P_{32}$ and no longer to $P_1 \cup P_{31}$. If the graph G_R had a normal edge from vertex v_5 to vertex v_2 , the above described `chain_cut_1` would not have been possible, because G_{C_1} would not have been disconnected by the `chain_cut_1`.

In the previous paragraphs we explained how PABFIT determines the disconnected components of G_S and how they are assigned to the partitions $P_1 \cup P_{31}$

Figure 15. Example of a First Partition with a chain_cut.1.

or $P_2 \cup P_{32}$ (including chain_cut.1). All these assignments do not violate the first-level partition properties, that means there are no connections between symbolic vertices assigned to P_1 and P_2 , respectively. An architecture restriction that has not been verified yet is the limited number of physical macrocells that are located in a partition P_α . These constraints are verified for each possible First Partition. The maximum number of vertices that can be assigned to the different partitions is given by Equations (1) through (4).

If the previously mentioned constraints are not violated, PABFIT verifies also the global and local reset output restrictions. Only vertices assigned to partitions

P_1 and P_3 have output connections to the global reset, hence only these vertices can trigger the global reset. The verification of the local reset output restrictions is explained very detailed for the second-level partitioning of the subgraphs resulting from the First Partition. Therefore, it is not explained here. The principle idea how to check them is the same for the First and Second Partition.

If all the architecture constraints are not violated by the First Partition of the symbolic graph G_S , PABFIT has found a *feasible_1* partition defined as follows.

Definition VI. 6: A first-level partitioning of the symbolic graph G_S into the two groups of subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$ is called a *feasible_1* partitioning if $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$ do not violate the first-level partitioning properties nor the input restrictions of the global and local reset. Additionally, the Equations (8) and 9 have to be true.

$$|V_{S1}| \leq |P_1| \quad |V_{S2}| \leq |P_2| \quad (8)$$

$$|V_{S31}| \leq |P_{31}| \quad |V_{S32}| \leq |P_{32}| \quad (9)$$

As a result of the First Partition the initial symbolic graph G_S is divided into two groups of subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$ assigned to the partitions $P_1 \cup P_{13}$, and $P_2 \cup P_{23}$, respectively. For each *feasible_1* partition, PABFIT performs the second-level partitioning which is described in Chapter VI.4. The pseudocode for the First Partition is shown below.

```
vertex P1[12]; /* vertices assigned to P1 */
vertex P2[12]; /* vertices assigned to P2 */
vertex P31[4]; /* vertices assigned to P31 */
vertex P32[4]; /* vertices assigned to P32 */
```

```

short p31, p32; /* number of vertices in  $P_{31}$ ,  $P_{32}$  */
array of vertex Comp[32]; /* disconnected components  $G_{Ci}$  */
short feasible1 = FALSE;
vertex cut_vertex1;
First_Partition(P3,p3)
vertex P3;
short p3;
{
    remove output edges incident to vertices in P3;
    breadth first search for disconnected components Comp;
    determine all possible assignment Comp[i] to P1,P2,P31,P32;
    for(all possible assignments) {
        feasible1 = FALSE;
        if (components are feasible according to Definition 6)
            feasible1 = TRUE;
        if (feasible1 == TRUE)
            Second_Partition(P31,p31, P32,p32);
        for (i=1; i ≤ p31; i++) { /* all vertices in  $P_{31}$  */
            cut_vertex = P31[i];
            determine additional component;
            feasible1 = FALSE;
            if (components are feasible according to Definition 6)
                feasible1 = TRUE;
            if (feasible1 == TRUE)
                Second_Partition(P31,p31, P32,p32);
        }
    }
}

```

VI.3 P_{13} AND P_{23} ASSIGNMENT

The result of the First Partition is the partitioning of the symbolic graph G_S into the two groups of subgraphs $G_{S1} \cup G_{S31}$ and $G_{S2} \cup G_{S32}$. The outputs of vertices $v_{Si} \in V_S$ that are assigned to partition P_3 are globally available at the inputs of all other physical macrocells. The outputs of symbolic vertices v_{Sj} assigned to the partition P_1 are not available at the inputs of vertices assigned to P_2 and vice

versa. The physical macrocells of the partition P_1 and P_2 are intermediate or local.

In the P_{13} Assignment, several vertices $v_{S_i} \in V_{S_1}$ are being assigned to P_{13} , that means to the partition of intermediate macrocells. The remaining vertices of V_{S_1} that are not in P_{13} will then be assigned either to the partition P_{11} or P_{12} (local macrocells) in the Second Partition stage of PABFIT. Because of the symmetry of the Interconnection Matrix, the P_{23} Assignment can be performed analogously to the P_{13} Assignment. The only difference is that the vertices v_{S_i} are being selected from the vertex set V_{S_2} and not from V_{S_1} . The remaining vertices of V_{S_2} will then be assigned to partition P_{21} or P_{22} . In the following we explain only the P_{13} Assignment.

The vertices that are being assigned to P_{13} form the subgraph $G_{S_{13}}$. Since these vertices are assigned to the partition of intermediate physical macrocell, their outputs are availability at all other physical macrocells of P_1 . That means, they are available to all symbolic vertices that will be assigned to P_{11} or P_{12} (local macrocells) in the Second Partition of PABFIT.

PABFIT has to determine all possible P_{13} Assignments such that no possible solution of the fitting algorithm is excluded. The procedure is very similar to the P_3 Assignment before the First Partition. The difference is that in the P_{13} Assignment, PABFIT can only choose from the symbolic vertices assigned to P_1 and not from all the vertices of G_S . However, the selection of vertices $v_{S_i} \in V_{S_1}$ in the P_{13} Assignment has several additional restrictions. These restrictions are due to the P_3 Assignment of vertices that are in C_IN chains. Let us assume that a vertex $v_{S_i} \in V_S$ that is included in a C_IN chain has been assigned to partition P_3 during the P_3 Assignment. A symbolic vertex $v_{S_{i-1}} \in V_S$ that is included in the same chain has to be placed physically adjacent on the chip. According to the Interconnection Matrix of Figure 4, a macrocell p_{k-1} is intermediate, if the macrocell p_k is global. That

means, the vertex $v_{S_{i-1}}$ has to be preassigned to P_{13} .

In the following, all conditions when a vertex v_{S_i} has to be preassigned to partition P_{13} (assuming that v_{S_i} is not assigned to P_3) are given. The number of all vertices that have to be assigned to P_{13} is p .

- CASE 1: v_{S_i} is the first element in a chain with 3 elements & $v_{S_{i+1}}$ nor $v_{S_{i+2}}$ is assigned to partition P_{13} nor P_{31}
- CASE 2: $v_{S_{i+1}} \in P_{31}$ & $\langle v_{S_i}, v_{S_{i+1}} \rangle \in EC_S$
- CASE 3: $v_{S_{i-3}} \in P_{31}$ & $\langle v_{S_{i-3}}, v_{S_{i-2}} \rangle, \langle v_{S_{i-2}}, v_{S_{i-1}} \rangle, \langle v_{S_{i-1}}, v_{S_i} \rangle \in EC_S$

An example of the P_{13} Assignment where several vertices $v_{S_i} \in V_{S_1}$ have to be preassigned to P_{31} is presented in Figure 16. We continue with our example from Figure 13 where the P_3 Assignment for chained vertices was explained. In Figure 16 it is assumed that the vertices v_{S_4} and v_{S_8} are assigned to P_{31} . The vertices v_{S_3} and v_{S_7} have to be preassigned to P_{13} because they have output chains to the vertices assigned to P_3 . The physical macrocell that are physically adjacent to the global macrocells and that have output chains to the global macrocells are the intermediate ones. The vertex $v_{S_{11}}$ has to be preassigned to P_{13} , too (condition CASE 3).

Figure 16. Chains in the P_{13} Assignment.

In addition to the p preassigned vertices, PABFIT has to determine all possible P_{13} Assignments. The number $\tilde{m}$ of vertices that can be randomly assigned to

P_{13} is given by $\tilde{m} = \tilde{k}, 1, \dots, 4 - p$ with k according to Equation (10). By assigning vertices randomly, we mean exploring assignment possibilities of vertices that are not preassigned to P_{13} .

$$\tilde{k} = \begin{cases} |V_{S1}| - 8 - p & |V_{S1}| \geq 12 \\ 0 & |V_{S1}| < 12 \end{cases} \quad (10)$$

All possible P_{13} Assignment are determined by PABFIT analogously to the P_3 Assignment that was performed before the First Partition. Based on a First Partition which assigns $|V_{S1}|$ vertices to the partition P_1 the upper bound on the number of possibilities for the P_{13} Assignment is given by the following equation.

$$\text{number of possibilities} \leq \sum_{l=\tilde{k}}^{4-p} \binom{|V_{S1}| - p}{l} \quad (11)$$

The pseudocode for the P_{13} Assignment is shown below.

```

vertex P13[4]; /* vertices assigned to P13 */
short p13; /* number of vertices assigned to P13 */
short p; /* number of preassigned vertices to P13 */
short add; /* additional vertices assigned to P13 */
pos_P13()
{
    determine preassigned vertices to P13
    for(add=0; add<4-p; add++)
        det_all_poss2(add);
}
det_all_poss2(add)
short add;
{
    while (new P13 Assignment possibilities exist) {
        determine the new possibility
        First_Partition(P13,p13);
    }
}

```

VI.4 SECOND PARTITION

As mentioned at the beginning of Chapter VI, the Second Partition performs a partitioning of the groups of subgraphs resulting from the First Partition. The group of subgraphs $G_{S1} \cup G_{S31}$ is partitioned further into the two groups of subgraphs $G_{S11} \cup G_{S131} \cup G_{S311}$ and $G_{S12} \cup G_{S132} \cup G_{S312}$. The group of subgraphs $G_{S2} \cup G_{S32}$ is partitioned into the two groups of subgraphs $G_{S21} \cup G_{S231} \cup G_{S321}$ and $G_{S22} \cup G_{S232} \cup G_{S322}$. The second-level partitioning properties have been stated in Chapter IV. Because of the symmetry of the chip it is sufficient to explain the partitioning of the group of subgraph $G_{S1} \cup G_{S31}$ only. The partitioning of $G_{S2} \cup G_{S32}$ is identical.

Once the vertices assigned to partition P_{13} are determined the remaining vertices can be assigned either to P_{11} or to P_{12} . The problem solved by the P_{31} Assignment and the Second Partition can be stated as follows:

Problem Formulation VI. 2 Given the group of symbolic graphs $G_{S1} \cup G_{S31}$ explore all possible sets of vertices $V_{S13} = \{v_{Si} \mid v_{Si} \in V_{S1}\}$ ($|V_{S13}| \leq 4$) such that if the output edges incident on the vertices $v_{Si} \in V_{S13}$ are removed from $G_{S1} \cup G_{S31}$, the resulting graph $\tilde{G}_R = (\tilde{V}_R, \tilde{E}N_R, \tilde{E}C_R)$ can be partitioned into two groups of subgraphs $G_{S11} \cup G_{S131} \cup G_{S311}$ and $G_{S12} \cup G_{S132} \cup G_{S312}$ subject to the second-level partitioning properties.

In the First Partition, the output edges of the vertices assigned to P_3 have been removed, because their outputs are globally available at all other physical macrocells. Now in the Second Partition, PABFIT removes the output edges of vertices that are assigned to P_{13} . Their outputs are available at all vertices that have been assigned to P_1 in the First Partition. In other words, there are no restrictions

on the output connections of vertices assigned to P_{13} within the partition P_1 .

Analogous to the First Partition, the graph $\tilde{G}_R$ is examined for disconnected components $\tilde{G}_{Ci}$. A disconnected component $\tilde{G}_{Ci} = (\tilde{V}_{Ci}, \tilde{E}N_{Ci}, \tilde{E}C_{Ci})$ is defined analogously to the disconnected component G_{Ci} in the First Partition. Each disconnected component can be assigned to a different partitions $P_{11} \cup P_{131} \cup P_{311}$ or $P_{12} \cup P_{132} \cup P_{312}$ because there are no normal nor chain connections between different disconnected components.

Like in the First Partition, the assignment of each component to the partition $P_{11} \cup P_{131} \cup P_{311}$ or $P_{12} \cup P_{132} \cup P_{312}$ is done randomly. All different assignment possibilities are generated and tested if the number of vertices assigned to a certain partition does not exceed the maximum number given by Equations (1) through 4.

Figure 17 gives an example of the Second Partition. We continue our example from Figure 14 where a First Partition of the symbolic graph G_S was shown. In the example of the Second Partition, we assume that no chain_cut_1 was performed in the First Partition. Figure 17a shows the disconnected component G_{C1} resulting from the First Partition. Since only one disconnected component has been assigned to $P_1 \cup P_{31}$ during the First Partition, the group of subgraphs $G_{S1} \cup G_{S31}$ (input for the Second Partition) is formed only by G_{C1} . In Figure 17 it is assumed, that the symbolic vertex v_{S4} has been assigned to P_3 in the P_3 Assignment stage of PABFIT. The vertices $v_{S3}, \dots, v_{S5}$ are included in one C-IN chain, that means, they have to be placed physically adjacent of the chip. Therefore, vertex v_{S3} has to be preassigned to P_{13} because v_{S3} has an output chain-edge to v_{S4} .

The graph $\tilde{G}_R$ where the output edges of the vertex v_{S3} are removed is shown in Figure 17b. As it can be seen, the assignment of v_{S3} to P_{13} results in two disconnected components $\tilde{G}_{C1}$ and $\tilde{G}_{C2}$ that can be randomly assigned to partition

Figure 17. Example of a Second Partition.

$P_{11} \cup P_{131} \cup P_{311}$ or $P_{12} \cup P_{132} \cup P_{312}$. For our example we assume the assignment that is shown in Figure 17b. $\tilde{G}_{C1}$ is assigned to $P_{12} \cup P_{132} \cup P_{312}$ and $\tilde{G}_{C2}$ is assigned to $P_{11} \cup P_{131} \cup P_{311}$. The disconnected component $\tilde{G}_{C1}$ forms the subgraph $G_{S12} \cup G_{S132} \cup G_{S312}$ and $\tilde{G}_{C2}$ forms the subgraph $G_{S11} \cup G_{S131} \cup G_{S311}$.

Analogously to the First Partition where the concept of a chain_cut_1 was introduced, we use the concept of a chain_cut_2 in the Second Partition. In the First Partition, a chain_cut_1 allowed a chain-edge between a vertex v_{Si} that is assigned to P_{31} and a vertex v_{Si+1} that was assigned to P_2 . Now, in the Second Partition, a chain_cut_2 allows a chain-edge between a vertex v_{Sj} that is assigned to P_{311} and a vertex v_{Sj+1} that was assigned to P_{12} .

Definition VI. 7: A *chain_cut_2* of the chain CH_k at the *cut_vertex* $v_{Scut} \in V_{S311}$ is a mapping of the symbolic vertex v_{Scut} to the physical macrocell p_8 . The vertices v_{Scut-i} and v_{Scut+j} are mapped to the single physical macrocells p_{8-i} and p_{8+j} . The values of i and j are given below:

- $i = 1, \dots, \text{chain_number}(v_{Scut}) - 1$
- $j = 1, \dots, \text{chain_length}(v_{Scut}) - \text{chain_number}(v_{Scut})$

That means, that the set of vertices $\{v_{Sl} \mid v_{Sl} \in CH_k \ \& \ l \leq \text{cut}\}$ is assigned to partition $P_{11} \cup P_{131} \cup P_{311}$ and that the set of vertices $\{v_{Sl} \mid v_{Sl} \in CH_k \ \& \ l > \text{cut}\}$ is assigned to partition $P_{12} \cup P_{132} \cup P_{312}$.

Similarly to the First Partition all possible *chain_cut_2* are determined based on a given initial second partition with no *chain_cut_2*. Now, we are allowed to choose only vertices $v_{Si} \in G_{S311}$ with output chains-edges as prospective *cut_vertices*.

An example of a *chain_cut_2* is given in Figure 18. The example is based on the example from Figure 17 where the disconnected component $\tilde{G}_{C1}$ was assigned to the partition $P_{11} \cup P_{131} \cup P_{311}$. A *chain_cut_2* is possible at the vertex $v_{Scut} = v_4$. This result in an additional disconnected component consisting of vertex v_{S5} and assigned to partition P_{12} . The vertex v_3 is mapped to p_7 , v_4 to p_8 , and v_5 to p_9 .

All assignments of the disconnected components $\tilde{G}_{Ci}$ (including *chain_cut_2*) to the partitions $P_{11} \cup P_{131} \cup P_{311}$ or $P_{12} \cup P_{132} \cup P_{312}$ do not violate the second-level partitioning properties. However, for each assignment the architecture constraints that limit the number of vertices in a partition P_α have to be checked. Equations (1) through (4) show the maximum number of vertices in each partition.

Analogously to the First Partition, where a *feasible_1* partitioning was defined, we define now a *feasible_2* partitioning of the subgraph $G_{S1} \cup G_{S31}$.

Figure 18. Example of a Second Partition with *chain_cut_2*.

Definition VI. 8: A second-level partitioning of the group of subgraphs $G_{S1} \cup G_{S31}$ into the two groups of subgraphs $G_{S11} \cup G_{S311} \cup G_{S131}$ and $G_{S12} \cup G_{S312} \cup G_{S132}$ is called a *feasible_2* partitioning if $G_{S11} \cup G_{S131} \cup G_{S311}$ and $G_{S12} \cup G_{S132} \cup G_{S312}$ do not violate the second-level partitioning properties and the input restrictions of the global and local resets. Additionally, the Equations (12) through (14) have to be true.

$$|V_{S11}| \leq |P_{11}| \quad |V_{S12}| \leq |P_{12}| \quad (12)$$

$$|V_{S131}| \leq |P_{131}| \quad |V_{S132}| \leq |P_{132}| \quad (13)$$

$$|V_{S311}| \leq |P_{311}| \quad |V_{S312}| \leq |P_{312}| \quad (14)$$

As a result of the Second Partition all the possibilities of assigning symbolic vertices to different partitions are determined. However, in each partition P_α , only some symbolic vertices are assigned to their final physical locations (*chain_cuts*). Therefore, the next step of PABFIT determines the physical placement of all the remaining vertices of the symbolic graph. The pseudocode for the *Second Partition* is shown below.

```

vertex P11[4]; /* vertices assigned to  $P_{11}$  */
vertex P12[4]; /* vertices assigned to  $P_{12}$  */
vertex P311[2]; /* vertices assigned to  $P_{311}$  */
vertex P312[2]; /* vertices assigned to  $P_{312}$  */
vertex P131[2]; /* vertices assigned to  $P_{131}$  */
vertex P132[2]; /* vertices assigned to  $P_{132}$  */
short p311, p312; /* number of vertices in  $P_{311}, P_{312}$  */
array of vertex Comp2[16]; /* disconnected components  $\tilde{G}_C$ ; */
short feasible2 = FALSE;
vertex cut_vertex2;

```

SecondPartition($P_{31}, p_{31}, P_{13}, p_{13}$)

```

{
  remove output edges incident to vertices in  $P_{31}$  and  $P_{13}$ ;
  breadth first search for disconnected components Comp;
  determine all possible assignment of disconnected components Comp2[i]
    to  $P_1, P_2, P_{31}, P_{32}$ ;
  for(all possible assignments) {
    feasible2 = FALSE;
    if (components are feasible according to Definition 8)
      feasible2 = TRUE;
    if (feasible2 == TRUE)
      Physical_Placement();
    for ( $i=1; i \leq p_{311}; i++$ ) { /* all vertices in  $P_{311}$  */
      cut_vertex =  $P_{311}[i]$ ;
      determine additional component;
      feasible2 = FALSE;
      if (components are feasible according to Definition 8)
        feasible2 = TRUE;
    }
  }
}

```



```

        if (feasible2 == TRUE)
            Physical_Placement();
    }
}

```

VI.5 PHYSICAL PLACEMENT

The First and Second Partition assigns each vertex $v_{Si} \in V_S$ to a single partition P_α . Now, the Physical Placement performs a one-to-one mapping of each vertex v_{Si} to a single physical macrocell p_j . Chapter VI.5.1 describes the so called placement matrix which represents the output of the First and Second Partition. Then, Chapter VI.5.2 describes how the local reset output restrictions are verified before the final physical placement is performed. Finally, Chapter VI.5.3 presents the placement algorithm.

VI.5.1 Placement Matrix

The output of the Second Partition is a so called *Placement Matrix*. It describes the possible assignments of symbolic vertices v_{Sk} to a subset of physical macrocells. The row k of the matrix represents the symbolic vertex v_{Sk} and the column l the physical macrocell p_l . If the element a_{kl} of the Placement Matrix is “1” it means that symbolic vertex v_{Sk} can be mapped to the physical macrocell p_l without violating any partitioning properties, nor global/local reset input restrictions. If the element a_{kl} is “0”, the mapping of v_{Sk} to p_l is not feasible. Since the partitioning stage of the algorithm assigns the symbolic vertices only to partitions P_α of macrocells and not to single macrocells, the rows and the columns of the Placement Matrix may contain several “1s”.

An example of a Placement Matrix is given in Figure 19. We continue the ex-

ample from Figure 14 and Figure 17 where the First and Second Partition have been performed on the symbolic graph G_S of Figure 14a. PABFIT always tries to find a Physical Placement for First and Second Partitions without a `chain_cut_1` nor a `chain_cut_2`, before it determines the additional First and Second Partitions. Therefore we continue our example for a First and Second Partition without `chain_cut_1` (shown in Figure 15) nor `chain_cut_2` (shown in Figure 18).

According to Figure 17b and Figure 17b, the symbolic vertices $v_{S1}, \dots, v_{S9}$ have been assigned to the partition P_α as it is shown in Table I. The first column of Table I contains the symbolic vertices and the second column the corresponding partitions P_α .

TABLE I
RESULT OF FIRST AND SECOND PARTITION

vertex	v_{S1}	v_{S2}	v_{S3}	v_{S4}	v_{S5}	v_{S6}	v_{S7}	v_{S8}	v_{S9}
partition	P_{12}	P_{12}	P_{131}	P_{311}	P_{11}	P_{21}	P_{21}	P_{21}	P_{22}

In Chapter VI.4, only the Second Partition of the $G_{P1} \cup G_{P31}$ has been explained, because the partitioning of $G_{P2} \cup G_{P32}$ can be performed analogously. Therefore, we have not explained the Second Partition of the disconnected components G_{C2} and G_{C3} in Figure 14b. In Table I it is assumed that the Second Partition of the subgraphs $G_{S2} \cup G_{S32}$ in Figure 14b results in an assignment of G_{C2} to P_{22} and of G_{C3} to P_{21} .

Finally, Figure 19 shows the Placement Matrix of our example. The relation among the columns of the Placement Matrix (physical locations of the macrocells) and the partitions P_α is shown in Table II. The row “partition” lists the different

partitions, and the row “column” shows the physical macrocells (columns of the Placement Matrix) included in the corresponding partition. The relation can be verified in the reordered Interconnection Matrix from Figure 7.

a_{kl}																																
	p p p p p p p p								1 1 1 1 1 1 1 1								1 1 1 2 2 2 2 2 2								2 2 2 2 2 3 3 3							
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
v_{S1}	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v_{S2}	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v_{S3}	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v_{S4}	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v_{S5}	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v_{S6}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
v_{S7}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
v_{S8}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
v_{S8}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	LRG ₁		LRG ₂		LRG ₃		LRG ₄		LRG ₅		LRG ₆		LRG ₇		LRG ₈																	
	axis2								axis1								axis2															

Figure 19. Placement Matrix.

If a vertex v_{Si} is assigned to P_α it can be assigned to all physical macrocells of P_α . Therefore, the number of assignment possibilities of a single vertex v_{Si} is given by the number of macrocells in P_α . Row “# placement possibilities” in Table II shows the number of placement possibilities of a vertex v_{Si} depending on the partition P_α it is assigned to. These numbers result from Equations(1) through(4).

The number of placement possibilities of a vertex v_{Si} is identical with the number of “1s” in the row k that represents v_{Sk} in the Placement Matrix. If vertex v_{Sk} is assigned to one of the partitions P_{11} , P_{12} , P_{21} , or P_{22} , the row k contains four

TABLE II
COLUMNS OF THE PLACEMENT MATRIX AND PARTITIONS

partition	P_{11}	P_{12}	P_{131}	P_{132}	P_{311}	P_{312}
column	1,2,5,6	9,10,13,14	3,7	11,15	4,8	12,16
placement possibilities	4	4	2	2	2	2

partition	P_{21}	P_{22}	P_{231}	P_{232}	P_{321}	P_{322}
column	17,18,21,22	25,26,29,30	19,23	27,31	20,24	28,32
# placement possibilities	4	4	2	2	2	2

“1s”. In case v_{sk} is assigned to P_{131} , P_{311} , P_{132} , P_{312} , P_{231} , P_{321} , P_{232} , or P_{322} , the row k contains only two “1s”. Based on these observations, the output restrictions of the local resets are checked next.

VI.5.2 Verification of Local Reset Output Restrictions

Before PABFIT executes the final physical placement, the output restrictions of the local resets are checked for each local reset group. According to the Placement Matrix from Figure 19 a local reset group is given by 4 adjacent columns starting with the first four columns representing the local reset group LRG 1, the columns 5, ..., 8 representing LRG 2 etc. All vertices v_{si} of type TOGGLE that are assigned to a single physical local reset group have to be triggered by the same local reset lr_j .

As mentioned, each vertex v_{si} is assigned to a single partition P_α . Each partition P_α covers either 4 columns of the placement matrix or 2 columns (# of placement possibilities in Table II). If the partition P_α offers 4 placement possibilities, then two of them are in one local reset group LRG_j and the other two are in

one adjacent local reset group LRG_{j+1} (if j is odd) or LRG_{j-1} (if j is even). If the partition P_α offers only two placement possibilities, then one of them is in one local reset group LRG_j and the other is in one adjacent local reset group LRG_{j+1} (if j is odd) or LRG_{j-1} (if j is even). Thus, after the Second Partition, each symbolic vertex v_{Si} is restricted to be assigned to at most two different local reset groups. It is not possible to find a feasible physical assignment if more than two symbolic vertices v_{Si} triggered from different local resets have been assigned to one partition P_α in the Second Partition (P_α has four placement possibilities).

In the following, all cases that lead to violations of local reset output restrictions are shown. Let us assume that n is the number of different local resets that trigger vertices assigned to one single partition P_α . Then, the following conditions result in violations of the local reset output restrictions.

- The number n is larger than 2.
- The partition P_α with 4 placement possibilities contains symbolic vertices v_{Si} that are triggered from $n = 2$ distinct symbolic local resets lr_k and lr_l . Each of the following conditions leads to a violation of the local reset output restriction:
 - CASE 1: There are four symbolic vertices assigned to P_α . Three of them are triggered from lr_k and one of them is triggered from lr_l .
 - CASE 2: 1 vertex that is assigned to a different partition $P_{\alpha 1}$ that has two placement possibilities and that covers the same local reset group as P_α is triggered by a different local reset lr_m ($lr_m \neq lr_k \neq lr_l$).

Figure 20 shows two examples of placement matrices where the local reset output restrictions are violated.

a)	a_{kl}	1 2 3 4 5 6 7 8	9 0 1 2 3 4 5 6	7 8 9 0 1 2 3 4	5 6 7 8 9 0 1 2
	v_{S1}	1 1 0 0 1 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	v_{S2}	1 1 0 0 1 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	v_{S3}	1 1 0 0 1 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	v_{S4}	1 1 0 0 1 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
		$\underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}}$	axis2	axis1	axis2
		LRG1 LRG2			
b)	a_{kl}	1 2 3 4 5 6 7 8	9 0 1 2 3 4 5 6	7 8 9 0 1 2 3 4	5 6 7 8 9 0 1 2
	v_{S1}	1 1 0 0 1 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	v_{S2}	1 1 0 0 1 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	v_{S3}	1 1 0 0 1 1 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
	v_{S4}	0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
		$\underbrace{\hspace{1cm}} \underbrace{\hspace{1cm}}$	axis2	axis1	axis2
		LRG1 LRG2			

Figure 20. Examples of local reset output violations.

In Figure 20a it is assumed that the vertices $v_{S1}, \dots, v_{S3}$ are triggered by the local reset lr_1 and the vertex v_{S4} is triggered by lr_2 (CASE 1). It is not possible to assign each vertex of the set $\{v_{S1}, \dots, v_{S4}\}$ to a physical macrocell of the set $\{p_1, p_2, p_5, p_6\}$ without violating the local reset output restrictions. Each possible placement leads to two vertices in one local reset group that should be triggered from two different local resets. For example v_{S1} could be placed to p_1 , v_{S2} to p_2 , and v_{S3} to p_5 . So far the mapping would be feasible, if both local reset groups LRG_1 and LRG_2 are associated with the local reset lr_1 . However, v_{S4} has not been placed yet. Since it is trigger from $lr_2 \neq lr_1$, v_4 is not allowed to be in LRG_1 nor in LRG_2 . As a result the mapping is not feasible.

In Figure 20b it is assumed that the vertices v_{S1}, v_{S2} are triggered from lr_1 , the vertex v_{S3} is triggered from lr_2 , and v_{S4} from lr_3 . The vertices $v_{S1}, \dots, v_{S3}$ are assigned to P_{11} and v_{S4} is assigned to P_{311} (CASE 2). The vertex v_{S1} could be assigned to p_1 , v_{S2} to p_2 , and v_{S3} to p_5 without violating the local reset output restrictions. But then, there is no more available distinct local reset group for the vertex v_{S4} left.

VI.5.3 Final Physical Placement

If the partitioning of the symbolic graph G_S does not result in a violation of the output restrictions of the local resets, PABFIT executes the final physical placement. The Physical Placement is divided into two steps as it is shown below.

- STEP 1: Given a Placement Matrix, determine a mapping of each symbolic vertex $v_{Si} \in V_S$ to one single physical macrocell p_j such that all normal and chain connections of the netlist are feasible. This mapping is denoted *state-cell feasible* mapping.
- STEP 2: Given a state-cell feasible mapping, determine a mapping of all local resets lr_k to one or more (splitting) physical local resets plr_l , such that no local reset restrictions are violated. This mapping is denoted *state/reset-cell feasible* mapping.

The algorithm explores all possible state-cell feasible mappings of a given Placement Matrix until a state/reset-cell feasible mapping is found. In this case the algorithm terminates and the solution of the fitting problem is shown. Otherwise, if no state/reset-cell feasible mapping is found, different Placement Matrices (different partitions) are used in order to examine different mapping possibilities.

STEP 1 of the Physical Placement stage is very similar to the shift-placement algorithm of the Cypress Fitter [17]. However, the mapping possibilities of a symbolic vertex v_{Si} to a single physical vertex p_j are highly restricted according to the limited number of “1”s in the Placement Matrix. Therefore, the solution space that has to be examined by PABFIT in the Physical Placement stage is much smaller than the solution space of the Cypress Fitter.

PABFIT starts with assigning the symbolic vertex v_{S1} to the first physical macrocell p_j of the Placement Matrix that has a “1” in the row associated with v_{S1} . Then, v_{S2} is assigned to p_k , the first physical macrocell that has a “1” in the row of v_{S2} if $p_j \neq p_k$, etc. C-IN chains are taken into account during the placement stage, that means a vertex v_{Si+1} that has an input chain from a vertex v_{Si} is always placed physically adjacent to the physical location of v_{Si} . The algorithm generates all possible placements. Each placement is checked for a state/reset-cell feasible mapping.

Figure 21 shows a state-cell feasible mapping for the Placement Matrix from Figure 19. The left side of the state cell part of the Interconnection Matrix (shown already in Figure 4) gives the one-to-one assignment of the symbolic vertices to the physical state macrocells. The normal connections among the macrocells are indicated by an “x” in the Interconnection Matrix. An “x” in the row k and the column l means the symbolic vertex v_{Si} that is mapped to the physical macrocell p_k has an output connection to the symbolic vertex v_{Sj} that is mapped to p_l . If all “x” match with “1” of the Interconnection Matrix, all symbolic connection of the netlist are realizable on the chip, and the mapping is state-cell feasible.

In the STEP 2 of the Physical Placement, the symbolic local resets lr_i are assigned to the physical local reset groups LRG_j . The assignment is done based

Figure 21. State cell feasible mapping.

on a given state-cell feasible mapping. Since the state macrocell mapping is fixed, the assignment of local resets lr_i to local reset groups is fixed, too. Let us assume that the local reset lr_i triggers a symbolic vertex v_{Sk} and that v_{Sk} is assigned to a physical macrocell included in the local reset group LRG_j . Then, lr_i has to be assigned to the physical local reset plr_j associated with the local reset group LRG_j .

The assignment of all symbolic local resets can require a so called *splitting* of the symbolic local resets. Splitting of the local reset lr_i means, that one or more

additional local resets $lr_{i2}, \dots, lr_{in}$ that are triggered from the same set of vertices $\{v_k, \dots, v_l\}$ that triggers lr_i , are added to the initial local reset (initial local reset is renamed lr_{i1}). Splitting of the local reset lr_i is necessary if lr_i triggers TOGGLE cells that are mapped to distinct local reset groups, as determined in STEP 1 of the physical placement.

The local reset assignment is explained on our example from Figure 21. There, the state cell feasible mapping of the netlist described by the symbolic graph G_S of Figure 14 is shown. The local resets were not included in G_S of Figure 14. In Figure 22 the local resets represented by small boxes are shown together with G_S . If a local reset lr_i triggers a symbolic vertex v_{sj} , then there is an arrow from lr_i to v_{sj} . Analogously, there is an arrow from v_{sk} to lr_l if the local reset lr_l is triggered from the symbolic vertex v_{sk} .

Figure 22. Local resets and the symbolic graph.

The additional edges that are added to the symbolic graph in order to consider the input restrictions of the local resets in the partitioning stage of PABFIT (see Chapter V.2) are not shown in Figure 22. There is only one additional edges $\langle v_{S4}, v_{S6} \rangle$. The local reset lr_1 triggers the vertices v_{S2} , v_{S3} , v_{S5} , and v_{S6} mapped to the physical macrocells p_{10} , p_3 , p_5 , and p_{17} . The local reset lr_2 triggers v_{S9} mapped to p_{25} . The local reset lr_1 is triggered only by vertex v_{S4} mapped to the state macrocell p_4 .

Figure 23 illustrates the local reset assignment using the so called *Local Reset Connectivity Matrix*. In the upper part of Figure 23 the state cell mapping from Figure 21 is shown again. The rows of the Local Reset Connectivity Matrix represent the physical local resets, and the columns represent the 32 physical state cells. An output connection from a local reset cell of the row k to a state cell of the column l is possible only if the element a_{kl} of the matrix is a “1” and is included in one of the 8 small boxes in Figure 23. These 8 boxes represent the 8 local reset groups (LRGs). The Local Reset Connectivity Matrix of Figure 23 is a part of the Interconnection Matrix of Figure 4.

According to the netlist, the local reset lr_1 triggers the vertex v_{S2} mapped on the physical macrocell p_{10} . The physical macrocell p_{10} belongs to the local reset group LRG_3 associated with the physical local reset plr_{36} . Therefore, PABFIT maps lr_1 to the physical location plr_{36} . Then, it is verified, if lr_1 can trigger all other symbolic vertices v_{S3} , v_{S5} , and v_{S6} as required from the netlist. If an output is realizable (falls within the local reset group associated with lr_1), then the corresponding column of the Local Reset Connectivity Matrix contains a “x”, otherwise a “*”. As it is shown in Figure 23a only the output connections from lr_1 to v_{S2} is realizable. Hence, lr_1 has to be split to $lr_{11}, \dots, lr_{14}$. The split local resets lr_{11}

Figure 23. Local Reset Connectivity Matrix.

has to be mapped to p_{34} , lr_{12} to p_{35} , lr_{13} to p_{36} , and lr_{14} to p_{38} . Then, all the output connections of the local resets are realizable. The splitting of lr_1 can be performed, because no TOGGLES triggered from a different local reset than lr_1 are assigned to the physical local reset groups associated with lr_{11} , lr_{12} , lr_{13} , nor lr_{14} . The assignment of the local reset lr_2 is performed analogously to the assignment of lr_1 .

CHAPTER VII

RESULTS OF THE FITTER

The PABFIT program that was presented in Chapter VI was tested on several industrial examples. These examples are real life examples and test files obtained from Cypress Semiconductor. The Cypress Development System [26] for the CY7C361 chip starts from a high-level VHDL description of the circuit and produces the netlist with the additional information about the local and global resets which are used as an input to PABFIT (see Chapter III.2). The results of PABFIT are compared with the results of the fitter that is presently included in the Cypress Development System. Their fitter is called Cypress Fitter.

Table III and IV show the examples on which PABFIT was tested. All examples which were solved by the Cypress Fitter (denoted by "+") were also solved by PABFIT and are presented in Table III. The CPU times of PABFIT and the Cypress Fitter are not comparable, as the Cypress Fitter is implemented on a PC and PABFIT on a SparcII. In a significant number of cases, the Cypress Fitter could *not* find a feasible solution within several hours. These examples are shown in Table IV and are indicated by a "NC" (non completed) in the column "Cyp Fit". The column contains a "-" if the result of the Cypress Fitter is not known.

A "feasible solution" of the fitting problem is a state/reset-cell feasible placement of each element of the netlist and the local and global resets. If there is no feasible solution it means that the netlist can not be mapped on the CY7C361 chip. The column "PABFIT feasible solution exists" on Table III and IV shows

the results of our approach. The answer “yes” means, that PABFIT was able to find a state/reset-cell feasible placement, “no” means, that PABFIT showed that the netlist of the given examples can not be mapped on the device. If there is an “NC” in that column, PABFIT could not show in a reasonable time if a feasible solution exists or not. In order to show the non-existence of a feasible solution, the fitter has to examine the whole solution space of the fitting problem. Because of the effective limitation of the solution space due to the partitioning approach, PABFIT can give the answer that there is no feasible solution for certain examples within minutes. The Cypress Fitter was not able to do this, because its solution space was not limited effectively enough by the architecture constraints. However, the proof of non-existence might be very time consuming, because all possible P_3 Assignments, First Partitions, P_{13}/P_{23} Assignments, and Second Partitions have to be explored. The examples, for which the non-existence of a feasible solution was proven by PABFIT are shown in Table IV. The CPU time for these examples varies from a few seconds up to several minutes, what is still acceptable.

As it can be seen in Table IV, PABFIT found solutions to the fitting problem for 13 of the 15 examples, which were not completed (NC) by the Cypress Fitter. In 9 cases feasible state/reset cell mappings were found and in 5 other cases it was shown that a feasible solution does not exist (“no” in column “PABFIT feasible solution exists”). The information, that a given netlist can not be mapped on the CY7C361 is also very important. If the reason for the failure of the mapping attempt can be detected, a feedback loop to the logic synthesis stage can be constructed, such that a new improved netlist can be generated and the fitting process can be pursued again, with a higher probability of success. Only one example out of the set of examples obtained from Cypress Semiconductor could not be solved by

PABFIT. For “dees2_bal” PABFIT could not find a solution even if the program was running for several hours. At the end of this Chapter we will describe the netlist of “dees2_bal” and explain why PABFIT has problems to solve this example.

TABLE III
INDUSTRIAL EXAMPLES

name	#vert	#edges	max conn	#C-IN	PABFIT feas sol exists	CPU time [s]	Cyp Fit
busa_bal	16/0/0	12	3	10	yes	0.2	+
cntr_ba	9/0/1	45	9	0	yes	0.2	-
counter1	13/0/0	12	2	1	yes	0.1	+
demo2	9/0/0	9	3	4	yes	0.1	+
dram1_bal	22/0/0	28	6	11	yes	5.7	+
epee11	24/0/0	35	8	13	yes	131.6	+
epeecon_ba	16/1/0	34	6	8	yes	12.4	+
examp33	14/0/2	47	9	0	yes	0.4	+
example8	10/1/0	64	9	0	yes	2.4	+
micro_ba	9/1/1	20	9	0	yes	0.1	-
mlt_fsm1	21/0/0	22	3	5	yes	0.1	+
reaword	15/1/1	23	4	5	yes	0.1	+
seqdetec	14/0/0	18	5	9	yes	0.9	+
stepper	16/0/0	19	6	8	yes	2.8	+
tsrbug_b	15/0/0	43	16	0	yes	5h	-
tsr2_bug	18/0/0	19	2	2	yes	0.1	+
vmerq3_b	22/0/0	20	9	10	yes	120.4	+
vmes1111	15/0/0	17	8	9	yes	0.2	+
word	19/1/1	34	6	8	yes	0.2	+

Table III and IV contain also information about the characteristics of the symbolic graph G_S . This information can be used to evaluate the complexity of the algorithm for a given example, and in order to develop certain heuristics that will speed up the search process. The number of vertices and the number of edges of the

TABLE IV
EXAMPLES WHERE PABFIT IS BETTER THAN CYPRESS FITTER

name	#vert	#edges	max conn	#C-IN	feasible solution exists	CPU time [s]	Cyp Fit
cadman_b	25/0/1	45	11	0	<i>yes</i>	150.1	NC
cadman_ba	25/0/1	45	11	0	<i>yes</i>	226.9	NC
cpu2_ba	31/1/0	33	4	9	<i>yes</i>	0.1	NC
data8	10/1/4	78	15	0	<i>no</i>	0.6	NC
dees2_ba1	30/1/2	100	13	15	<i>NC</i>		NC
epeel_res	27/0/0	53	13	13	<i>no</i>	4159.3	NC
examp11	13/1/2	70	9	8	<i>no</i>	0.1	NC
mbarn_ba	31/0/0	28	4	22	<i>no</i>	92.6	NC
new1	25/0/2	40	9	0	<i>yes</i>	0.9	NC
own361	31/1/4	51	9	25	<i>no</i>	18.0	NC
sqr_ex_ba	32/1/0	60	8	3	<i>yes</i>	6.7	NC
tgen_ba1	25/1/1	60	14	0	<i>yes</i>	4.6	NC
timgen	25/0/2	55	15	0	<i>yes</i>	2.7	NC
timgen2	25/0/1	62	15	0	<i>yes</i>	3.6	NC
warpt_ba	32/1/0	64	5	2	<i>yes</i>	22.0	NC

symbolic graph G_S are given in the column #vertices and #edges, respectively. The second column gives also the information about the number of global and local resets included in the netlist (# state macrocells / # global resets / # number local resets). From the ratio #edges/#vertices, information about the connectivity of G_S can be obtained. If this ratio is large, G_S is highly connected and the partitioning approach restricts the solution space of the fitting problem very effectively. Otherwise, the solution space is only weakly restricted by the connectivity, and if the netlist does not contain chains, the solution space of the algorithm is very large. However, it should be easy to find a solution of the fitting problem in this case. The solution space of the fitting problem is restricted most effectively by the chain connections. They

influence the partitioning of the symbolic Graph G_S very strongly, since chained vertices have to be mapped to adjacent macrocells on the chip. Therefore, a solution can be found very fast if there are many chained vertices. The number of vertices of G_S that have an input chain from other vertices is given in the column $\#C_INS$.

The Cypress Fitter is not able to find solutions for examples with a high number of vertices and a high connectivity between them. In such a case the solution space of the shift-placement algorithm becomes too large to obtain a feasible solution within a reasonable time. A complexity analysis and a comparison between the Cypress Fitter and PABFIT is given in Chapter IX. If the total number of vertices is large (20 and larger), and only a few vertices are included in chains, the Cypress Fitter is not able to find a feasible solution. The examples “cadman_b”, “cpu2_ba”, “seq_ex_ba”, and “own361” are not solved by the Cypress Fitter. For all these examples PABFIT could find feasible solutions in seconds.

Other hard cases for the Cypress Fitter are examples that include many TOGGLE cells that are triggered from local resets. The example “timgen” contains 2 local reset nodes that trigger TOGGLE cells. One local reset triggers 15 state macrocells, and the other triggers one state macrocell. Example “timgen2” is much easier, because it contains only 1 local reset node that triggers 16 state macrocells. In this example, the local reset can be split to all local reset groups, such that its output is available at all inputs of the 32 macrocells. The splitting can be performed, since there are no input connections to the local reset.

Other examples with many many local resets and TOGGLE cells triggered by them are “examp11” and “data8”. For these examples, the connections between the local resets and the TOGGLE cells are the reason why the netlist can not be mapped on the chip. The connectivity between the macrocells themselves is

very low. Because PABFIT checks the local reset restrictions before performing the Physical Placement, the non-existence of a feasible solution could be shown in a very short time. The Cypress Fitter has to determine all possible physical placements and check the reset restrictions for all of them.

The previously mentioned example that could not be solved by the Cypress Fitter nor by PABFIT is “dees2.ba1”. The netlist contains many state macrocells and two local reset cell lr_1 and lr_2 . The local reset lr_1 is triggered from several state macrocells, and both lr_1 and lr_2 trigger TOGGLE cells. Additionally, the connectivity between the state macrocells is high.

Based on the results gained from the examples shown in Table III and IV a prediction on the behavior of PABFIT applied on more general examples with more TOGGLES and higher connectivity between the macrocells can be made. The higher the connectivity, the more effectively the solution space will be limited, and especially the First Partition will exclude many unfeasible placements. The same is true for examples with many TOGGLES triggered from several local resets because the local reset restrictions are included in the partitioning stage. However, PABFIT will have to go through many P_3 Assignments until a feasible solution will be found. The developed heuristics described in Chapter VIII will become even more important for these cases.

In order to test PABFIT on examples that include several TOGGLE cells, three of the examples from TABLE IV have been modified. Additional local resets have been added to the netlist of “epeecon.ba”, “vmerq3_b”, and “cadman_b”, such that a state/reset cell feasible mapping exists. We can be sure that a feasible solution exists, because the changings have been done based on a solution that has been determined by PABFIT previously. The additional local resets trigger several

TOGGLE cells and are triggered by state macrocells.

The local reset restrictions are included in the partitioning stage of PABFIT. Therefore, they are used in order to exclude unfeasible placements during the partitioning stage. If local resets are added to the netlist, it means, that the placement possibilities of the symbolic vertices $v_{S_i} \in V_S$ become more limited. Hence, adding local resets speeds up the search for the feasible solution because more unfeasible placements are excluded in an early stage of the algorithm. TABLE V shows the CPU time of the examples where additional local resets have been added. The CPU time with the additional local resets are shown in the column “new CPU time”, the original CPU time from Table III and IV in column “old CPU time”. The new number of symbolic vertices, global resets and local resets is shown in the column “#vert” of TABLE V

TABLE V
ADDITIONAL LOCAL RESETS

name	#vert	new CPU	old CPU
		time	time
cadman_b	25/0/5	132.4	150.1
epeecon_ba	16/1/4	6.4	12.4
vmerq3_b	22/0/4	1.4	120.4

Comparing the “old” and “new” CPU times of TABLE V it can be observed, that the CPU time is reduced for all three examples. The highest reduction occurs at the example “vmerq3_b”. Here, many First Partitions are excluded by the additional local resets. Without these local resets, PABFIT found many feasible First Partitions that lead to unfeasible Second Partitions. All of these Second Partitions

had to be checked and the result was the high CPU time.

CHAPTER VIII

HEURISTICS FOR THE FIRST PARTITION

In Chapter VI.1 the assignment of vertices $v_{Si} \in V_S$ to the partition P_3 was described. The outputs of macrocells assigned to P_3 are available at the inputs of all other state macrocells and reset cells, because the physical macrocells of P_3 are global. All other state macrocells that are not assigned to P_3 can be assigned only to intermediate or local macrocells (partitions P_1 , P_2 and their subpartitions). Therefore, the P_3 Assignment determines directly if PABFIT can find a First Partition of the symbolic graph G_S or not. After the P_3 Assignment, the output edges of vertices $v_{Si} \in V_S$ that are assigned to P_3 are removed. A good P_3 Assignment disconnects the symbolic graph G_S into several components G_{Ci} that can be assigned to different partitions. By properly choosing the vertices assigned to P_3 , we will be able to find a solution without going through many P_3 Assignment possibilities. As a result, a good P_3 Assignment can speed up the search for a feasible solution and reduce the CPU time of PABFIT.

If no feasible solution exists for a given example, the P_3 Assignment does not influence the CPU time. The whole solution space of the fitting problem as determined by the algorithm, has to be explored and a different P_3 Assignment changes only the order of the search.

As it has been described in chapter VI, PABFIT is based on an exact algorithm. That means, all possible partitions are determined by PABFIT and checked for a state/reset-cell feasible mapping. No solution is excluded and the whole so-

lution space is examined. The heuristics that are presented in this Chapter do not influence the exactness of the algorithm. Heuristics are applied only to change the order in which vertices are assigned to P_3 . No possible P_3 Assignment is excluded, and PABFIT determines still all possible partitions if necessary. However, it is more likely, that a good P_3 Assignment, one that leads to a feasible solution, is found faster.

The non-heuristic P_3 Assignment as described in Chapter VI.1 was determined randomly. The order was given by the initial order of the netlist. Basically, there are two possibilities to influence the P_3 Assignment.

- The order in which a different numbers of vertices (number given by m) are assigned to P_3 can be changed. The non-heuristic PABFIT starts with $m = 0$ and increases m step by step. The heuristic PABFIT might start with $m = 7$, continue with $m = 8$, and than start with $m = 0$, until all possibilities are explored.
- The order in which m vertices are assigned to P_3 can be changed. For example, the non-heuristic PABFIT starts with assigning vertex v_1 , v_2 , and v_3 ($m = 3$) to P_3 . The heuristic PABFIT might start with v_2 , v_3 , and v_4 . It continues assigning step by step a new vertex.

The heuristic that has to be applied in order to speed up the search for a feasible solution depends highly on certain properties of the netlist. Testing PABFIT on several examples it was not possible to find a good heuristic that could be applied for all of them. For some examples a certain heuristic could speed up the search, but for others, the CPU time was even increased. Therefore, heuristics are applied only if the netlist has certain characteristics.

The netlist characteristics are based on the in-degree of the vertices of the symbolic graph G_S representing the netlist. The in-degree of the vertex $v_{Si} \in V_S$ is defined as the number of directed normal edges $en_k = \langle v_{Si}, v_{Sj} \rangle \in EN_S$ ($i \neq j$) for all vertices v_{Sj} .

Heuristics are applied in the P_3 Assignment stage if some vertices of the symbolic graph have a high in-degree compared to other vertices. Therefore, the first step is to determine the in-degree of all the vertices of the symbolic graph and store all the values in a so called *degree vector*. Then, the *degree vector* is ordered subject to an increasing degree and the difference Δ between adjacent elements of the degree vector is calculated. The maximum difference is denoted by Δ_{max} and the element of the degree vector where Δ_{max} occurs is denoted $i_{\Delta_{max}}$. Figure 24 shows an example of the degree vector, Δ_{max} , and $i_{\Delta_{max}}$.

Figure 24. Degree vector.

If $\Delta_{max} > 5$, the algorithm starts assigning $m = \Delta_{max}$ vertices to the partition P_3 and increases the value of m after all possibilities for $m = \Delta_{max}$ have been explored. The variable m is increased until it reaches $m = 8$, then PABFIT starts again at $m = 0$ and counts up to $m = \Delta_{max}$. If $\Delta_{max} > 8$, PABFIT starts with $m = 8$.

TABLE VI shows some of the industrial examples from TABLE III and IV shown in Chapter VII. Only for these examples, PABFIT applies the heuristics described in the previous paragraphs. The column “max in-degree” shows the maximum in-degree of all vertices of G_S and the column “ Δ_{max} ” the maximum degree difference between adjacent elements of the degree vector. The starting value of m is shown in column “ m_{Start} ”.

TABLE VI
EXAMPLES WHERE PABFIT USES HEURISTICS

name	max in- degree	Δ_{max}	m_{Start}	CPU time heuristic [s]	CPU time non-heuristic [s]
cadman_b	9	6	6	58.3	150.1
cadman_ba	9	6	6	136.6	226.9
tsrbug_b	14	13	8	0.1	5h

For all the examples where PABFIT applies heuristics the CPU time is decreased compared to the non-heuristic PABFIT. In the case of “tsrbug_b” the reduction is huge. The CPU time is decreased from several hours to less than one second.

CHAPTER IX

COMPLEXITY ANALYSIS

The previous Chapter presented the results of PABFIT. PABFIT could find feasible solutions for many examples where the Cypress Fitter was not able to. This is due to the reduced solution space of the new algorithm compared to the Cypress Fitter algorithm. At first the complexity of the Cypress Fitter is discussed, followed by an examination of the complexity of the algorithm of PABFIT.

In the following, an asymptotic complexity analysis of the fitting algorithm is presented. The fitting problem is an algorithmic problem $\Pi : I \rightarrow 2^S$ where I is the set of problem instances and S is the set of configurations. For the fitting algorithm of the Cypress CY7C361, the set I represents the n macrocells of the netlist and the set S all possible permutations of the n macrocells. It can be stated also as a decision problem $\tilde{\Pi} : I \rightarrow \{0, 1\}$ that gives the answer if the netlist can be mapped to the device or not. Like other placement and routing problems, the fitting problem for the CY7C361 [1] is an NP-complete problem. The upper bound on the run time of the algorithm can be estimated by $O(n^2)$ with.

The number of different mapping possibilities for the Cypress Fitter which is based on shift-placement is given by Equation (15). The number of state macrocells of the netlist that are in a C-IN chain is given by the variable a and the number of different chains is given by the variable b . The number of vertices of the netlist is $n = |V_S|$.

$$\# \text{ of possible state cell mappings} = \frac{32!}{(32 - [n - (a - b)])!} \quad (15)$$

The Cypress Fitter maps one state vertex at a time to a single physical macrocell and backtracks if a connectivity restriction is violated. The C-IN chains are taken into account, that means vertices included in one chain are placed physically adjacent.

If the netlist contains also local resets, a mapping of each local reset to a single local reset group has to be performed in order to find a state/reset-cell feasible mapping. The CY7C361 has 8 local reset nodes. The possibilities to map m local reset nodes to 8 local reset groups is given by Equation (16).

$$\# \text{ of possible reset cell mappings} = \frac{8!}{(8 - m)!} \quad (16)$$

Additionally, each local reset can be split to 2 or more local resets. If only one local reset is split, the number of possibilities is given by Equation (17)

$$\# \text{ reset cell mappings} = m * \sum_{n=m}^8 \frac{8!}{(8 - n)!} \quad (17)$$

In general, each local reset can be split. Hence, the number of possibilities is given by Equation (18)

$$\# \text{ reset cell mappings} = \left[\sum_{l=1}^m \binom{m}{l} \right] * \left[\sum_{n=m}^8 \frac{8!}{(8 - n)!} \right] \quad (18)$$

Equation (18) describes the worst case when all different mapping possibilities are explored by the algorithm.

The new fitting algorithm implemented in PABFIT and based on graph partitioning does not start with the physical placement. Before the placement possibil-

ities based on the Placement Matrix are determined, the First and Second Partition is performed. The algorithm starts with assigning vertices from the symbolic graph to the partition P_3 that may contain up to 8 vertices. The number of possibilities to assign $k, \dots, 8$ vertices from the symbolic graph consisting of n vertices to P_3 is given by Equation (6) in Chapter VI.1.

For each P_3 Assignment, the resulting disconnected components are determined and assigned either to partition $P_1 \cup P_{31}$ or $P_2 \cup P_{32}$. Let $nc1 \leq n$ be the number of disconnected components, then the number of possible assignments can be calculated according to Equation (19).

$$\# \text{ of possible assignments} = \sum_{l=0}^{nc1} \binom{nc1}{l} \quad (19)$$

For each assignment the PABFIT algorithm performs a Second Partition, if the assignment of the components in the First Partition does not violate the first-level partitioning properties. The Second Partition starts partitioning the partition $P_1 \cup P_{31}$ resulting from the First Partition. For a feasible partitioning of $P_1 \cup P_{31}$ the algorithm continues partitioning $P_2 \cup P_{32}$. Both partitionings start with the assignment of vertices to P_{13} and P_{23} , respectively. The number of assigning $k, \dots, 4$ vertices to P_{13} is given by Equation (11) in Chapter VI.3 with $p = 0$.

Like in the First Partition, the components resulting from the P_{13} assignment are assigned to $P_{311} \cup P_{131} \cup P_{11}$ or $P_{312} \cup P_{132} \cup P_{12}$, respectively. If the number of disconnected components is $nc2 \leq |V_{S1}| + |V_{S31}|$, then the number of possible assignments can be calculated by Equation (20).

$$\# \text{ of possible assignments} = \sum_{l=0}^{nc2} \binom{nc2}{l} \quad (20)$$

After the First and Second Partition, the partitioning based algorithm, starts with the Physical Placement, if no local reset output restrictions are violated. As explained in Chapter VI.5 each vertex v_{Si} can be mapped only to four or two physical locations, depending on partition P_α it is assigned to. Table II in Chapter VI.5.1 shows how many placement possibilities exist for a vertex, as a function of the partition it is assigned to.

The algorithm of the Physical Placement and the algorithm of the Cypress Fitter are basically identical. However, the mapping possibilities of PABFIT are drastically reduced by the partitioning stage. Each row of the Placement Matrix in Chapter VI.5 contains at most four “1s”, such that at most four mapping possibilities of each macrocell from the netlist have to be considered. The Cypress Fitter would be equal to PABFIT if all the elements of the placement were “1” (that case can not happen in PABFIT).

It is not possible to compare the overall complexity of the Cypress Fitter and PABFIT. The behavior of both algorithms depends strongly on the properties of the netlist. Therefore, it is impossible to estimate the general run times of the algorithms. However, the testing of PABFIT on several examples, showed that PABFIT was able to find solutions in a shorter time than the Cypress Fitter.

CHAPTER X

CONCLUSION

In this thesis we formulated the placement and routing problem for special architecture EPLDs as a fitting problem. The Fitting Approach considers the chip architecture constraints during the placement stage in order to exclude unfeasible placements. These architecture constraints are due to the limited connectivity between the macrocells on the chip. The Fitting Approach will play an important role in developing tools for Application-Specific EPLDs with special chip architectures.

An architecture-driven exact algorithm (PABFIT) for the fitting problem of the Cypress CY7C361 has been developed, using an effective partitioning approach. A computer program implementing the described algorithm has been written in C for a SPARC II workstation.

The results of PABFIT tested on several real-life and test examples show the superiority of our approach over existing ones. As presented in Chapter VII, PABFIT could find feasible solutions for many examples where the Cypress Fitter could not find any. Contrary to the Cypress Fitter, PABFIT is also able to show if a netlist can be mapped to the chip or not. For several examples where it could be shown by hand, that no feasible solution exists, PABFIT proofed the non-existence. This ability of PABFIT is due to the reduced solution space of the fitting problem by the partitioning approach. The solution space of the Cypress Fitter is much larger, such that it could never show the non-existence of a feasible solution.

The previously mentioned observations lead to the conclusion, that the par-

tioning approach shows much better results than the traditional shift-placement method of the Cypress Fitter. Both approaches are based on a Fitting Approach, however, PABFIT takes the architecture restrictions much better into account than the Cypress Fitter. Therefore, PABFIT can find a feasible solution in a much shorter time.

In conclusion, a Fitting Approach that takes all the architecture constraints into account is a very effective approach for the placement problem of special architecture EPLDs. The partitioning approach may not be directly applicable to other chip architectures. However, the basic idea of partitioning the physical available macrocell according to their output availability and assigning sets of symbolic elements from the netlist to the partitions, can be applied for other architectures with restricted connectivity.

REFERENCES

- [1] Lengauer, T. "Combinatorial Algorithms for Integrated Circuits Layout", *B.G. Teubner, Stuttgart*, 1990.
- [2] Peram, M., Marek-Sadowska, M., Kuh, E.S., "Floorplanning with Pin Assignment," *Proc. ISCAS'90*, pp 327, 1990.
- [3] Murofushi, M., Yamada, M., Mitsuhashi, T., "FOLM-Planner: A New Floorplanner with a Frame Overlapping Floorplan Model Suitable for Sea-Of-Gates Type Gate Arrays," *Proc. IEEE ECCAD-90*, pp. 98, 1990.
- [4] Sha, L., Dutton, R. "An Analytical Algorithm for Placement of Arbitrarily Sized Rectangular Blocks," *Proc. 22st Design Automation Conference*, pp 602-608, 1985.
- [5] Burstein, M., Pelavin, R., "Hierarchical Wire Routing," *IEEE Trans. on CAD, Special Issue on Automatic Wire Routing, Ed. E.S. Kuh CAD-2*, October 1983, pp. 223-234.
- [6] Leiserson, C.E., Pinter, R.Y., "Optimal Placement for River Routing," *Proc. 1981 CMU Conf. VLSI Systems and Computations*, pp 126-142, 1981.
- [7] Sangiovanni-Vincentelli, A., Santomauro, M., Reed, J., "A New Gridless Channel Router: Yet Another Channel Router the Second (YACR-11)," *Proc. IC-CAD*, 1984.
- [8] Anderson, R. E., Coppola, A., Perkowski, M. A., "VHDL Synthesis of Concurrent State Machines to a Programmable Logic Device", *VHDL International User's Forum*, Scottsdale, Arizona, May 3-6, 1992.
- [9] Sechen, C., Sangiovanni-Vincentelli, A., "The TimberWolf Placement and Routing Package", *IEEE Journal of Solid-State Circuits*, Vol. SC-20, 2 April, 1985.
- [10] Auer, A., *PLD Handbuch: Tabellen und Daten* Hüthig Buch Verlag, 1990.
- [11] Sicard, P., "Logic Synthesis on PALS," *Logic and Architecture Synthesis for Silicon Compilers*, G. Saucier and P.M. Mc Lellan, Elsevier Science Publishers B.V., pp. 119-125, 1989.
- [12] Brayton, R. K., Rudell, R., Sangiovanni-Vincentelli, A., Wang, A.R., "MIS: Multi-Level Interactive Logic Optimization System," *IEEE Trans. on CAD*, pp. 1062-1082, November, 1989.

- [13] Murgai, R., Nishizaki, Y., Shenoy, N., Brayton, R. K., and Sangiovanni-Vincentelli, A., "Logic Synthesis for Programmable Gate Arrays," *27th ACM/IEEE Design Automation Conference*, pp. 620-625, 1990.
- [14] Murgai, R., Shenoy, N., Brayton, R. K., and Sangiovanni-Vincentelli, A., "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD-91*, pp. 564-567, Santa Clara, CA, November 1991.
- [15] Keutzer, A., Dagon, "Technology Binding and Local Optimization by Dag Matching," *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 341-347, June, 1987.
- [16] R. J. Francis, J. Rose, Z. Vranesic, "Chortle-cdf: Fast Technology Mapping for Lookup-Table Based FPGAs," *Proc. 28th DAC*, June 1991, pp. 613-619.
- [17] Perkowski, M.A. and Chrzanowska-Jeske, M.E., Coppola, A., and Pierzchala, E., "An Exact Algorithm for the Technology Fitting Problem in the Application Specific State Machine Device," *ISCAS*, 1991.
- [18] XILINX, "The Programmable Gate Array Design Handbook", 1986.
- [19] Actel Inc., Databook, 1990.
- [20] S. Brown, J. Rose and Z. Vranesic, "A Detailed Router for FPGAs," *IEEE Journal of Solid State Circuits*, pp. 277-282 Vol 26 No 3 March 1991.
- [21] Concurrent Logic, Inc., "CLi6000 Series Field Programmable Gate Arrays," Preliminary Information, Dec 1. 1991.
- [22] Cypress, in *BIMOS CMOS Data Book*, 1991.
- [23] J. Rose, Z. Vranesic, W. M. Snelgrove, "ALTOR: An Automatic Standard Cell Layout Program," *Proc. Can. Conf. VLSI*, Nov. 1985, pp. 168-173.
- [24] B. Tseng, J. Rose and S. Brown, "Using Architectural and CAD Interactions to Improve FPGA Routing Architectures," *1992 ACM First International Workshop on FPGAs*, pp. 3-8, 1992.
- [25] E. A. Walkup, S. Hauck, G. Borriello and C. Ebeling, "Routing-directed Placement for the Triptych FPGA," *1992 ACM First International Workshop on FPGAs*, pp. 33-38, 1992.
- [26] Cypress Semiconductor, Warp1 PLD Compiler.
- [27] Anderson, R., "Programmable High-Speed State Machine with Sequencing Capabilities," *U.S. Patent*, no. 4,965,472, October 23, 1990.
- [28] Göller, S., Perkowski, M.A., Coppola A., "Minimization of Sum of "Condition Decoder" Expressions for New Programmable Logic Devices", *submitted to IEE*, July 1992.
- [29] Yap, H.P., "Some Topics in Graph Theory", *Cambridge University Press*, Cambridge 1986, p. 88.