11-4-1997

Routing, Driven Placement for ATMEL 6000 Architecture FPGAs

Songhua Zhang
Portland State University

Recommended Citation

Zhang, Songhua, "Routing, Driven Placement for ATMEL 6000 Architecture FPGAs" (1997). *Dissertations and Theses.* Paper 6119.

https://doi.org/10.15760/etd.7979

THESIS APPROVAL

The abstract and thesis of Songhua Zhang for the Master of Science in Electrical and Computer Engineering were presented November 4, 1997, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS:

[redacted]

Malgorzata E. Chrzanowska-Jeske, Chair

[redacted]

Marek A. Perkowski

[redacted]

Jingke Li
Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL:

[redacted]

Rolf Schaumann, Chair
Department of Electrical and Computer Engineering

ABSTRACT

An abstract of the thesis of Songhua Zhang for the Master of Science in Electrical and Computer Engineering presented November 4, 1997.

Title: Routing – Driven Placement for ATMEL 6000 Architecture FPGAs

Based on the concept of Cell Binary Tree (CBT), a new technique for mapping combination circuits into ATMEL 6000 Architecture FPGAs is presented in this thesis. Cell Binary Tree (CBT) is a net-list representation of combinational circuits. For each node of CBT there is a distinguished variable associated with it, the node itself represents a certain logic function, which is selected according to target FPGA architecture. The proposed CBT placement algorithms preserve local connectivity and allow better mapping into ATMEL FPGA. Experiments reveal that the new mapping technique achieved reduction in a number buses used for routing comparing with previously proposed Modified Squashed Binary Tree (MSBT) approach and possibly reduction of area as well. In general, the new technique is realized through following four major steps:

1. Grouping and generating CBT: This is a step to read *blif format* file, which is the result of logic synthesis, into a CBT data structure through grouping algorithm, which is a process of gathering logic functions into nodes for mapping based on a targeted FPGA architecture. The main objective of creating CBT is to generate a minimum number of nodes (or cells) to be mapped.

2. CBT placement: Upon getting the minimum number of nodes in CBT to be mapped, the next step is to map those nodes into cells in FPGA. The significance of the placement method in this thesis is to lineup the cells with the same variable into the same row in the FPGA.

3. Bus Assignment: The process of assigning variables to local buses, which run in two possible directions; horizontal and vertical. ATMEL 6000 has two horizontal buses and two vertical buses for each cell. The assignment is based on the number of times a variable appears in a row or column.

4. Routing: The last stage of the process is the connecting cells which have the same input variable. One of the important steps in the routing process is to choose connection bridge cells with the minimum impact on the area.

ROUTING – DRIVEN PLACEMENT
FOR
ATMEL 6000 ARCHITECTURE FPGAS

by

SONGHUA ZHANG

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

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University

1998

ACKNOWLEDGEMENTS

I would like to thank Dr. Malgorzata Chrzanowska-Jeske, my advisor, for providing guidance in my research work and supporting my career growth. I thank her for the methodical introduction to my thesis work by seminers and the reading and conference group, which had help me to gein knowledge beyond the research area.

I would like to thank Dr. Marek A. Perkowski and Dr. Jingke Li for serving on my committee and for their numerous suggestions in the preparation of the thesis.

Also, my special thanks go to Ms. Shirley Clark and Ms. Laura Riddell for their support through these years.

Specially, I would like to thank Kiswanto Thayib, Dezheng Tang, and Sida Zhou, who are my best friends, for their patience in going through my thesis work and presentation, and working valuable suggestions and origination of my thesis.

TABLE OF CONTENTS

PAGE

Chapters

LIST OF FIGURES

FIGURE PAGE

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Very Large Scale Integration Technology (VLSI) has opened the door to the implementation of powerful digital circuits at low cost. It has become possible to build chips with millions of transistors, as exemplified by state-of-art microprocessors. Such chips are realized using the full-custom approach, where all parts of a VLSI circuit are carefully tailored to meet a set of specific requirements. Semi-custom approaches such as Standard Cells and Mask-Programmed Gate Array have provided an easier way of designing and manufacturing Application-Specific Integrated Circuits (ASICs). Each of these techniques, however, requires extensive manufacturing effort, taking several months from beginning to end. In the electronics industry it is vital to reach the market with new products in the shortest possible time. Furthermore, it is important that the financial risk incurred in the development of a new product be limited so those more new ideas can be prototyped. Field Programmable Gates Arrays (FPGAs) have emerged as a solution to these time-to-market and risk problems because they provide instant manufacturing and very low cost prototyping.

Programmable devices, which are becoming a popular technology for designers seeking fast and cost effective implementations of their circuits, have gone through a complete evolution from simple PLDs (Programmable Logic Devices) to FPGAs.

1.1 Programmable Devices

1.1.1 Programmable Logic Devices (PLDs)

A PLD typically consists of an array of AND gates connected to an array of OR gates. A logic circuit to be implemented in a PLD is thus represented in a sum of product form. There are two versions of the PLD:

- Programmable Array Logic (PAL): A PAL is comprised of a programmable AND-plane followed by a fixed OR-plane.
- Programmable Logic Array (PLA) is also comprised of an AND-plane followed by an OR-plan, but, in this case, they are more flexible than PALs, because the connections to both planes are programmable.

PALs and PLAs are available in both mask programmable and field programmable versions. With their simple two-level structure, the PLDs allow high speed-performance implementations of logic circuits. However, the simple structure also leads to their main drawback; they can only implement small logic circuits that can be represented with a modest number of product terms. This limitation confined the PLDs from emerging as a general device for digital designs.

1.1.2 Field Programmable Gate Array (FPGA)

A more evolved Logical Device, the Field Programmable Gate Array (FPGA), combines the programmability of PLD and the scalable interconnection structure of Mask-Programmed Gate Array (MPGA). This has resulted in programmable devices with much higher logic density. An FPGA consists of an array of uncommitted elements that can be interconnected using available routing structure. The interconnections between the elements are user-programmable.

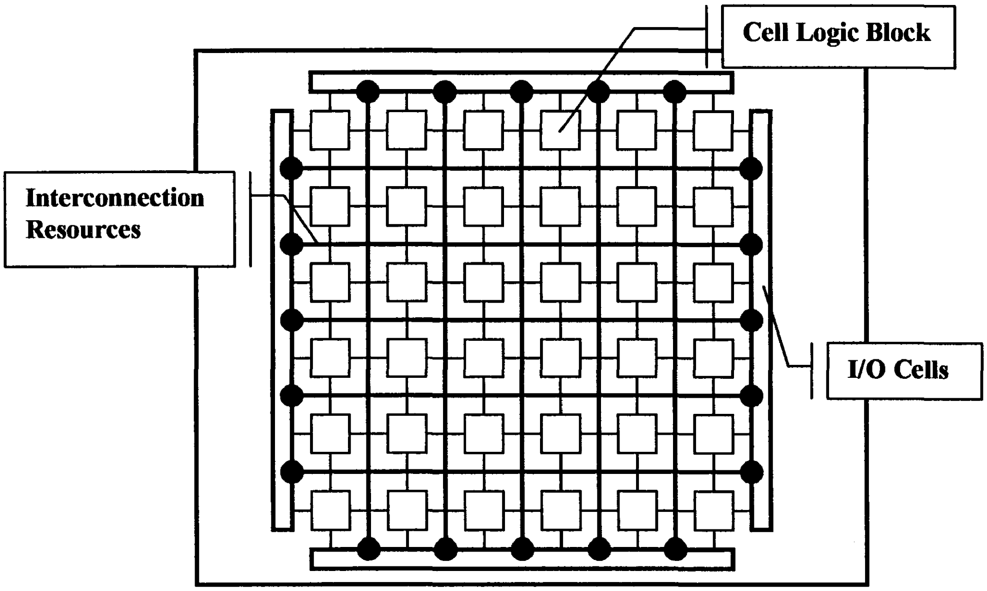


Figure 1.1: CA-Type FPGAs

A conceptual diagram of a typical FPGA is shown in Figure 1.1. An FPGA consists of a two-dimensional array of logic blocks that can be connected by general interconnection resources. The interconnections comprise of segments of wire, and the segments are of various lengths. Programmable switches serve to connect the logic blocks to the wire segments or one segment to other. Logic circuits are implemented in the FPGA by partitioning the logic into individual logic blocks and then interconnecting the blocks as required via switches.

1.2 Traditional Design Process with FPGA

A designer who wants to make good use of FPGAs must have access to an efficient CAD system. A good design tool can greatly improve the quality of the resulting circuit. The traditional steps involved in a typical CAD system for implementing a circuit using FPGA is given in Figure 1.2.

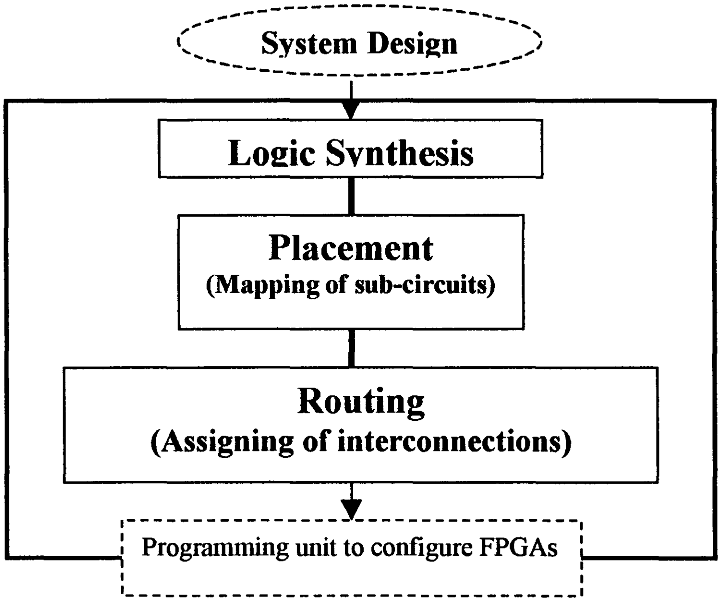

System Design
Logic Synthesis
Placement
(Mapping of sub-circuits)
Routing
(Assigning of interconnections)
Programming unit to configure FPGAs

Figure 1.2: CAD system for FPGAs

The CAD system for FPGAs consists of the following steps:

1. **Logic Synthesis:** It is an essential part of the entire design process. It starts from the functional description of the circuit or system, and ends with the description, typically referred to as a net-list, which depends on the particular target technology. In general, logic synthesis consists of three steps; translation from a Register-Transfer level representation written in a hardware description language like VHDL to a gate-level circuit; optimization of the gate-level circuit; and technology mapping.

2. **Placement**: In this step of the design cycle, the sub-circuits, which are formed in the technology mapping phase, are allocated to a physical location on the FPGA, i.e., the logic block on the FPGA is programmed to behave like the

sub-circuit that is mapped to it. This placement must be carried out in a manner that the routing can complete the interconnections. This is very critical, as the routing resources of the FPGA are limited. The placement algorithms [28,29,30,31,32,33,34,42,43] for general gate arrays are normally used for the placement in FPGAs, which will be discussed in next few sections.

3. **Routing**: In this phase, all the sub-circuits which have been programmed on the FPGA blocks are interconnected with the routing segments [32,35,36,37].

In this thesis, the steps of placement and routing will be the main focus. In the following section, the problems of placement and routing will be defined and followed by a section to overview existing placement and routing solutions for FPGA. Whether or not these existing algorithms can be used in our defined problem or ATMEL 6000 architecture (CA-Type) (which will be discussed in detail in next chapter) will also be analyzed.

1.3 Problem Description

In traditional methods, technology mapping, placement and routing are dealt with separately. Therefore, for Cellular-Architecture Type (CA-Type) FPGAs, a large number of logic cells is used for wiring connections, or is left unused [3,7]. This problem is mainly caused by not preserving the local connectivity during the logic optimization step. Frequently, local buses need to be used to complete even very short connections, which increases circuit delay. To avoid the excessive usage of the local buses a better solution is to use different logic implementations with logic cells used as wiring cells. But this will lead to wasting a large number of logic cells for wiring in the phase of technology mapping.

In general, the placement step is a process used to transfer net-list into physical layout in FPGA, which assigns the logic blocks in the FPGA to implement the modules in the net-list. The placement step is very crucial in the overall layout design because it must make provisions for optimal routing. An ill-placed layout results in poor quality routing. The routing step determines the routing channels (vertical and horizontal) and logic cells to be used in each interconnection path without specifying the actual geometric layout of tracks.

1.4 Overview of Existing Algorithms for FPGA Placement and Routing

Before going to the detailed overview of placement and routing methods, a brief description of types of FPGA architecture and their differences will be helpful for the review. ATMEL 6000 architecture, which is the target architecture of this thesis, is a symmetrical array of identical cells. Except for "repeaters" spaced every eight cells. Each cell can realize logic functions as AND, AND-XOR etc. The array is continuous and completely uninterrupted from one edge to the other. In addition to logic and storage functions, cells can also be used as wires. Buses, which include local and global buses, support fast, efficient communication over medium and long distances. Local buses deal with communications within eight-cell array and global buses are used for communication among eight-cell array. Both resources are very limited with maximum usage of four local buses for each cell. Therefore, routing is critical due to the fact that connections among cells are restricted and the number of possible connections is much smaller than other FPGAs, such as Xilinx's FPGA.

Another typical FPGA architecture is Xilinx's Configurable Logic Blocks (CLBs) based FPGA, which had a very different architecture from the previous "AND-OR" array PLD architectures. Xilinx's FPGA architecture has an interior matrix of CLBs and a surrounding ring of I/O interface Blocks (IOBs). Interconnect resources occupy the channels between the rows and columns of CLBs, and between the CLBs and IOBs. A CLB can be configured to function as one or two lookup tables. A configuration program stored in an on-chip memory control the functions, which include the CLBs, IOBs and their interconnection. The configuration program is loaded automatically from an external memory on power-up or on command, or is programmed by a microprocessor as a part of the system initialization.

Compared with fine grained FPGA such as ATMEL's FPGA, there are several important differences between them:

- Input variables: Xilinx has 5 input variables for each CLB. ATMEL's FPGA has 3 input variables (maximum) for each cell.
- Cell size: Xilinx can implement many large logic functions for CLB. ATMEL's FPGA only can implement certain small simple functions and any large or complex functions have to be implemented in multiple cells.
- Interconnection: ATMEL's FPGA has more limited resources on local interconnection due to large or complex logic function that has to be

realized by multiple cells, which results are more interconnection than the CLB which can realize relatively large or complex logic functions.

Given the above architecture reasons, it can be inferred that methods developed for Xilinx cannot be applied or will be very difficult to apply to ATMEL's FPGA because most of them do not consider the interconnection among the cells (which is a much simpler problem in Xilinx's FPGA). Such methods like TRADE [26], MIS-PGA [25] and chortle-crf [24] are focused on logic decomposition to mapping the logic into CLBs. In the following, some of the important algorithms for FPGA placement and routing will be discussed. The discussion will cover the graph [7] and tree-based approaches [3,12], the Complex Maitra Logic Array approach [22], the efficient logic synthesis (i.e considering the placement and routing at the logic synthesis stage) approach [1,3,6] and the macro-cell approach [12].

1.4.1 Graph/Tree Based Approach

Graph and tree based approaches are the most common ones to deal with placement and routing problems. There are many papers in this area. Several graph [7] and tree-based algorithms [3,12], which are closely related to FPGA placement and routing, will be reviewed in this section.

Directed Acyclic Graph (DAG)-Approach: A simulated evolution mapping method, which is based on the general Directed Acyclic Graph (DAG), was developed in [7]. The DAG is used as a multi-level representation of a Boolean function, where each node represents the logic, which can be realized in one logic block of the target architecture and an edge represents the connectivity between the cells. Taking into consideration the routing constraints for target FPGA architecture, a cost function of goodness was developed in [7]. The evolution process goes like this: It starts with an initial placement of the cells, then every cell in the placement is subjected to evolution phases. In each evolution phase, the goodness of each of the cells is determined and followed by a selection phase to identify which cells needs to be replaced.

As pointed out in [12], the alternate columns are taken to place the grouping cells in initial placement. Therefore, the room for evolution improvements is ensured. But this approach leads to a lot of unused cells, which are considered to be wasted, when the rectangular area of the mapped circuit is taken into account. Simulated Evolution is a non-deterministic algorithm for incremental design change. In this circumstance, the routing paths may change due to different placement and hence the timing of the entire circuit may change.

Modified Squashed Binary Tree (MSBT) Approach: The new mapping approach called Modified Squashed Binary Tree (MSBT) [3,12] has improved the area usage significantly compared to Squashed Binary Tree (SBT)[44], which leaves a lot of unused cells around the placed area. In MSBT method, the net-list is represented as a binary tree with decision variables associated with each node. Each node is also associated with logic functions of the target FPGA architecture. The author of [12] has demonstrated that this method can be used in ATMEL 6000 architecture. Based on this binary decision tree, a tree restructuring algorithm has been developed to directly map logic nodes into cells of FPGA. The most significant feature of MSBT is to preserve local connectivity during the placement process and map uncompleted binary tree to the FPGAs.

However, MSBT has two obvious weaknesses;

(1) The bigger number of the unused cells is included in the placed area when a rectangular area is concerned.

(2) Using more routing cells to connect the buses of same variable name from the rows and columns if the input pad is limited.

Sometimes it may not be possible to complete the routing in an ATMEL 6000 series. More details about MSBT approach will be described in Chapte5.

Binary Tree Mapping [43]: which presented an algorithm based on binary decision tree mapping. This method is an improvement over MSBT because of the following shortcomings of MSBT as pointed out by [43]:

- Start with the left longest path and map it to a column (initial column) of the two-dimensional array.
- Map the right longest path (root's right child) to the column (second column) right of the initial column.
- Start from the leave node of the longest path and recursively map the nodes on the left side of the initial column as close as possible to that column.
- Start from the leave node of the second column and recursively map the nodes on the right side of the second column as close as possible to the column.

To modify the above problem of the MSBT algorithm as follows by Chrzanowska-Jeske and Xu approach [43]:

- First map the critical longest path.
- Map from bottom to top, the advantages of which are that the nodes of higher level can occupy the unused cells left by the lower level.

In general, the modified method generated more compact mapping results. But the disadvantage of this method is that it does not consider variable allocations, which may result in the need of extra cells or buses to rout cells and may not be able to complete of the routing if the mapping size is limited.

Other FPGA Routings: Most of FPGA routing has primarily concentrated on producing feasible solutions that use the fewest routing resources, and most of the algorithms are developed for the Xilinx-like architecture with the switches, which are used programming the interconnections between cells. A key problem in the routing of FPGA is that successful routing of some connection may rely on the assignment of a specific wiring segment in the FPGA for that connection. If this essential segment is assigned to some other connection, then routing failure is guaranteed. A Coarse Graph Expansion (CGE) method [35] was developed for FPGA routing. This routing algorithm has the ability to resolve routing conflicts by considering the side-effects of options for cell connections, which can be realized by routing nets, based on demand and assigning critical nets a higher routing priority. The coarse graph expansion (CGE) router [35] decides specific wire segments implementing particular connections. In the first phase of CGE, an expanded graph is generated for each net by examining the routing switches and wire segments along the path described by the coarse graph. In the second phase, CGE places all paths from all of the expanded graphs into a single path list. The router then selects paths from the list based on a cost function. Each selected path defines the detailed route of its corresponding connection.

Other researchers also focus on minimizing path length, for example, the bounded-radius bounded-cost (BRBC) method of [38] achieves wire-length-radius tradeoffs in weighted graphs, but can not directly produce a shortest-paths tree with minimum wire-length. Rather, with the tradeoff parameter tuned completely toward path-length minimization, the methods of [38] produce the same shortest-paths tree as would *Dijkstra's* algorithm [40]. The recent A-Tree algorithm of [39] for rectilinear *Arborescence Steiner* trees depends heavily on the Manhattan norm.

However, none of these works directly on the CA-type of FPGAs such as ATMEL 6000 architecture. One of the main reasons is that the switch box model does not apply well on ATMEL 6000 architecture due to the fact that connection segments (local buses) in the channel are limited to about four, and most of the routings are through logic cells, hence it behaves differently.

1.4.2 Complex Maitra Logic Array (CMLA)

This algebraic approach presented in [6, 22], provides a well-defined theoretical background for the manipulation of Boolean function application to Cellular Architecture two-dimensional arrays. The synthesis model of [6] is composed of two planes: the complex (input) and the collecting (output) plane. It is similar to the conventional PLA architecture. Since each cell in the ATMEL and Xilinx architectures can realize an AND, OR, EXOR or their combinations, the outputs of the cell array constitute a special class of Boolean function called *Maitra terms* [23] which are named from *Maitra Cascade*. In ATMEL 6000 architecture, because of the local connectivity among adjacent cells, a *Maitra terms* can be realized using a column of logic cells and primary variables are assigned to local buses, one per each column of cells. To illustrate this, let's look at the following example [7]:

$f_0 = ac \oplus abd \oplus bcd$

$f_1 = b \oplus d$

$f_2 = c \oplus bd \oplus a$

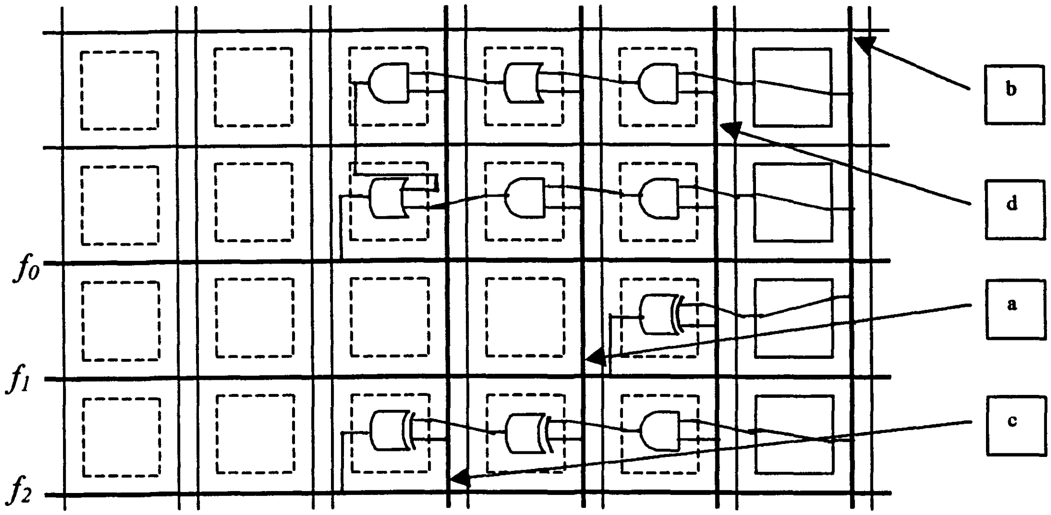


Figure 1.3: CMLA: Realization of a Function

After factorization phase, the results are:

$f_0 = (bd + a)c + bda$ *which is two complex terms:*

$f_1 = b \oplus d$ *which is one complex term:*

$f_2 = bd \oplus a \oplus c$ *which is one complex term.*

The realization of the above complex terms with setting the variable order as (b, d, a, c) is illustrated in Figure 1.3. One advantage of this approach is that no separate routing step is required to realize a complex term [7].

This method was developed for generic CA-type FPGA. In general, it assumes that two inputs and one output for every logic cell in CA-type FPGA. However, if we attempt a realization with ATMEL 6000 FPGA [2] as the target architecture, we see the following two major disadvantages of the chip architecture which ware pointed out by [12]:

- "Two EXOR gates have been realized into two logic blocks (see Figure 1.3). To avoid this case we have to add a switch cell between the two EXOR gates when they are mapping into ATMEL 6000 FPGA. It means that we have to move the entire column, associated with the expansion variable "c" to the next column".
- "In the ATMEL architecture, AND and EXOR gates can be combined and realized into one logic block. The CMLA method does not take advantage of this powerful feature. As a result of the additional cell used increases the delay of the design".

1.4.3 Efficient Logic Synthesis

Since the routing resources are very limited, efficient usage of these resources can significantly reduce the area occupied by the design. Consequently, it will increase the capacity of the chip and improve performance. The early 1993 publications [1,3,6] have revealed several logic synthesis approaches applicable to CA-Type FPGAs. These approaches (in one way or another) try to link logic synthesis steps with placement and routing steps. The spectral methods based on orthogonal expansion [20] and Universal XOR Forms [9] on classical cellular array [10] are examples. The spectral methods are more general and usually lead to better solution.

Decision diagram (DD) approaches are the most important development in recent history. The DD approach is based on the decomposition of Boolean functions using combinations of Shannon and two Davio expansions. There are many forms of decision diagrams. Function representation based on EXOR gates are very attractive for ATMEL 6000 architecture, because EXOR gates are available in logic with same cost of other gates. In addition, EXOR gate based representations give the more compact implementation for special functions. All the decision diagrams have the structure of a binary tree with decomposition variables associated with each node of the tree. The advantages and disadvantages of binary tree placement and routing have been discussed in section 1.4.1

1.4.4 Macro-Cell Approach

In the industry, the layout problem for CA-Type FPGA, such as ATMEL 6000 architecture is solved by the macro-cell approach. The macro cell represents certain logic functions, which is a technology dependent representation of a circuit design. The macro-cells are organized into libraries. The designed circuit is covered with a minimum number of relatively small macro cells, the placement and routing of these cells are done through automatic placement and routing techniques, such as simulated annealing [41], heuristic approaches and other methods published in papers. But this macro-cell approach has inherent disadvantages as pointed out by [12]:

- "This method does not provide any opportunity for the synthesis of the general purpose function where decompositions into submodules are not know".
- "The modules have irregular shapes and routing requires many cells to be used just for connections".

1.5 Objectives of the Thesis

In this thesis, we focus on layout synthesis for Cellular-Architecture Type (CA-Type) FPGA [3,4]. The distinguished feature of these devices is the local connectivity between logic blocks placed in a symmetrical array. Logic blocks are usually of small granularity and of the standard-cell type with a limited number of inputs and outputs. Local or global buses are used for distance connections. As we know, the approach of separating technology mapping, placement and routing, which is currently used for other FPGAs, has little value for CA-type devices primarily due to the local connectivity. Therefore, new comprehensive methods to the layout-synthesis problem need to be developed to efficiently utilize the potential of CA-Type FPGAs.

We present a new approach to the placement and routing problems in the ATMEL 6000 architecture FPGAs that takes local bus connections into consideration. Permuted Reed-Muller Tree (PRMT), which is obtained by applying Davio I Expansion [1,7] to a Boolean function, is used as input to our algorithm. Input data are represented in *blif format*. The new approach is to restructure a Permuted Reed-Muller Tree (RPRMT) (input blif file from logic synthesis) into a Cells Binary Tree (CBT) with minimized nodes, such that these nodes (cells) can be mapped to the ATMEL 6000 Series FPGA. The CBT will result in minimized area that can be easy to route. The method developed here is applicable to any general binary tree. In our approach, the ATMEL 6000 Series FPGA is the target architecture, but the approach can be adapted to other CA-Type FPGAs. Our algorithms are written in C and implemented on a SPARC 10 workstation.

1.6 Outline of the Thesis

The thesis is organized as follows. In Chapter 2 architecture and restrictions of the ATMEL 6000 FPGA are discussed. Chapter 3 provides an overview of logic synthesis methods and describes some logic synthesis methods to generate a binary tree; specifically Permuted Reed-Muller Tree (PRMT). Chapter 4 introduces the restructuring of a PRMT into a Cell Binary Tree form and how it relates to the architecture of the CA-Type FPGAs. The formulation of the problem and our approach (including grouping, placement, bus assignment and routing algorithms) will also be described. Chapter 5 compares our methods to the MSBT method with layout examples and results tables. Conclusions and future work from our research are given in Chapter 6. Finally, in Appendix, some examples of physical layouts are manually presented and the differences between the MSBT method and our algorithm are shown.

CHAPTER 2

ARCHITECTURE OF ATMEL 6000 FPGA

FPGA combines the high density and the versatility of gate arrays with the time–to-market advantages and the off-the-shelf availability of user programmable standard parts. There are several kinds of FPGAs in the market, the primary interest of this thesis is in ATMEL 6000 Series FPGAs. A single ATMEL FPGA chip is an array of small yet highly functional cells that have many routing resources and provide excellent flexibility and silicon utilization. In general, the following are the major advantages:

- *Lots of Powerful Cells*: ATMEL 6000 Series FPGAs have thousands of small, powerful cells that can be programmed into any of 35 cell state types. A single cell can be configured into 20 combinatorial states, including NOR, OR, AND, NAND and two-input multiplexer.
- *Fast, Flexible Bussing*: Two type of buses such as local and express buses support fast efficient communication over medium and long distances in the ATMEL 6000 Series array. Local buses are the link between the cells and the bussing network. Repeaters, located every eight cells in the array, transfer signals to maintain signal strength and enhance design performance. Because the since express buses are not connected directly to cells, they are fast.

- *Flexible Synchronous and Asynchronous Clocking*: Each cell column has an independent clock and reset or a global clock can provide low-skew distribution of external clock signals.
- *Simpler Placement and Routing*: Each cell is symmetrical in the array. Signals can enter and exit from any side, and cells can also be used as wires that can make placement and routing faster.
- *High-I/O Options*: The lower-density devices are available in high-I/O versions to accommodate small designs requiring lots of input and outputs.
- *Pipelined Designs*: The registers of any comparable FPGA, designs can be pipelined for super-fast performance.

Since technology mapping is architecture specified, in order to describe the proposed methodology for technology mapping and routing in this thesis, the general features of the architecture are reviewed and the main restrictions that lead to the presented algorithms are pointed out in this chapter.

2.1 The Symmetrical Array

At the heart of the ATMEL 6000 architecture is a symmetrical array of identical cells which is shown Figure 2.1. Except for "repeaters" spaced every eight cells; the array is continuous and completely uninterrupted from one edge to the other. In addition to logic and storage functions, cells can also be used as wires. Buses support fast, efficient communication over medium and long distances.

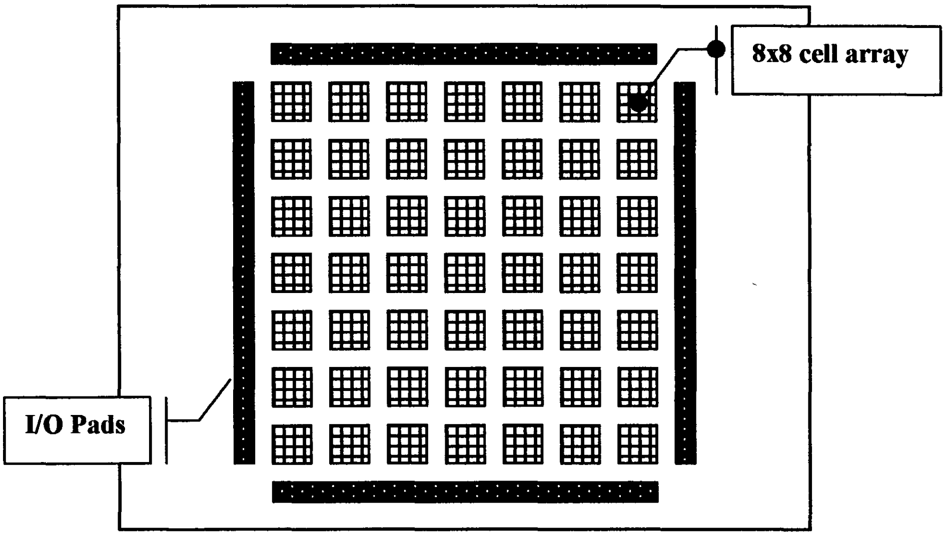


Figure 2.1: Symmetric array of the ATMEL 6000 chip

2.2 The Bussing Network

The ATMEL 6000 architecture has two kinds of buses as indicated in Figure 2.2. which are:

- **Local buses:** They are the link between the array of cells and the global bussing network. There are two vertical local buses for every column of cells, and two horizontal local buses for every row of cells. Every cell in the array has a read/write access to two vertical and two horizontal buses. See Figure 2.3 in the next page. In addition, each cell provides the ability to make a 90 degree turn between either of the two vertical buses and either of the two horizontal buses.
- **Express buses:** They are not directly connected to the cells, they are used for global connections via the local buses. Express busses are the fastest way to cover long, straight-line distances within the array. Each express

bus is paired with a local bus, so there are two express buses for each column and row of cells. Connective units called repeaters, spaced every eight cells, divide each bus, both local, and express, into segments spanning eight cells. Each "***Repeater***", see example in Figure 2.2, is associated with a local/express pair.

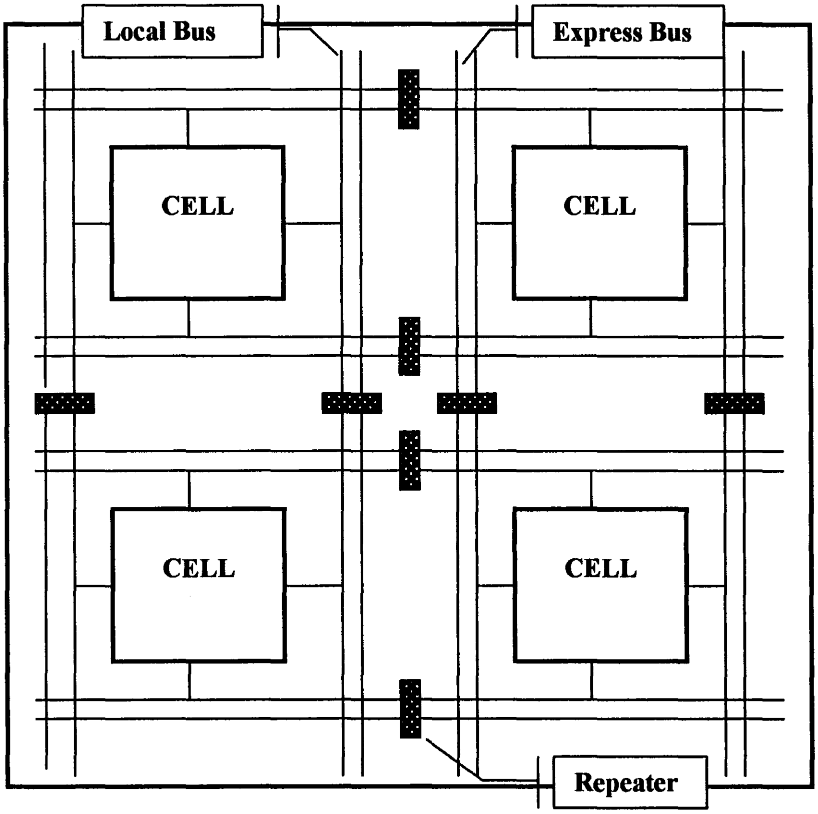


Figure 2.2: Bussing network

In addition to the four local bus connections (see example in Figure 2.3), a cell receives eight inputs and provides two outputs to its Top, Bottom, Left, and Right neighbors. These ten inputs and outputs are divided into two classes: "A" and "B". There are an "A" input and a "B" input for each neighbouring cell and a single "A" output and single "B" output driving all four neighbours. For outside connections, an "A" output is always connected to an "A" or a "B" output to go to "B" input.

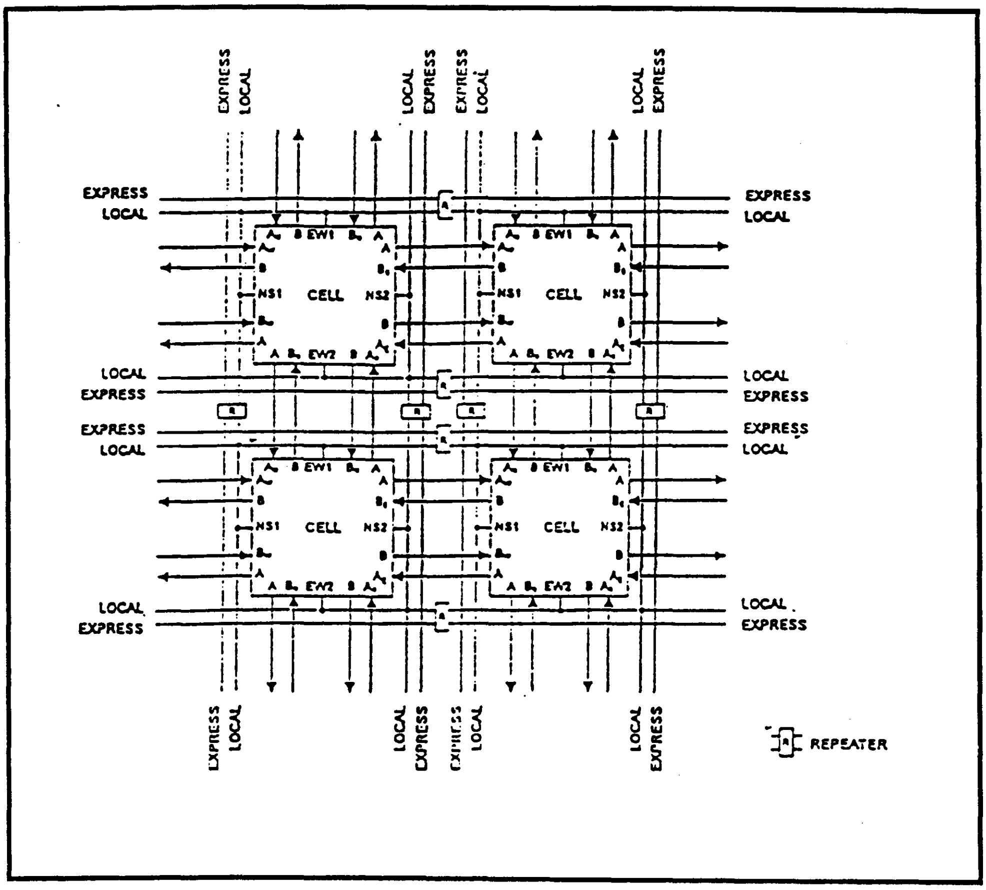


Figure 2.3: Cell-to-cell and cell-to-bus connections

Within the cell, the four inputs for "A" and the four inputs for "B" enter two separate and independently configurable multiplexers. The flexibility of cells is enhanced by allowing each multiplexer to select also the logic constant 1. The two multiplexers outputs enter the two upstream AND gates. The write access to the four local buses are controlled by the tri-state buffer. Hence, this single cell can be programmed to perform logic, wire and constant state.

2.3 The ATMEL 6000 Macorocell

ATMEL 6000 is macrocell based architecture; there are thirty five logical functions, which can be used for implementation of circuits. Out of these functions

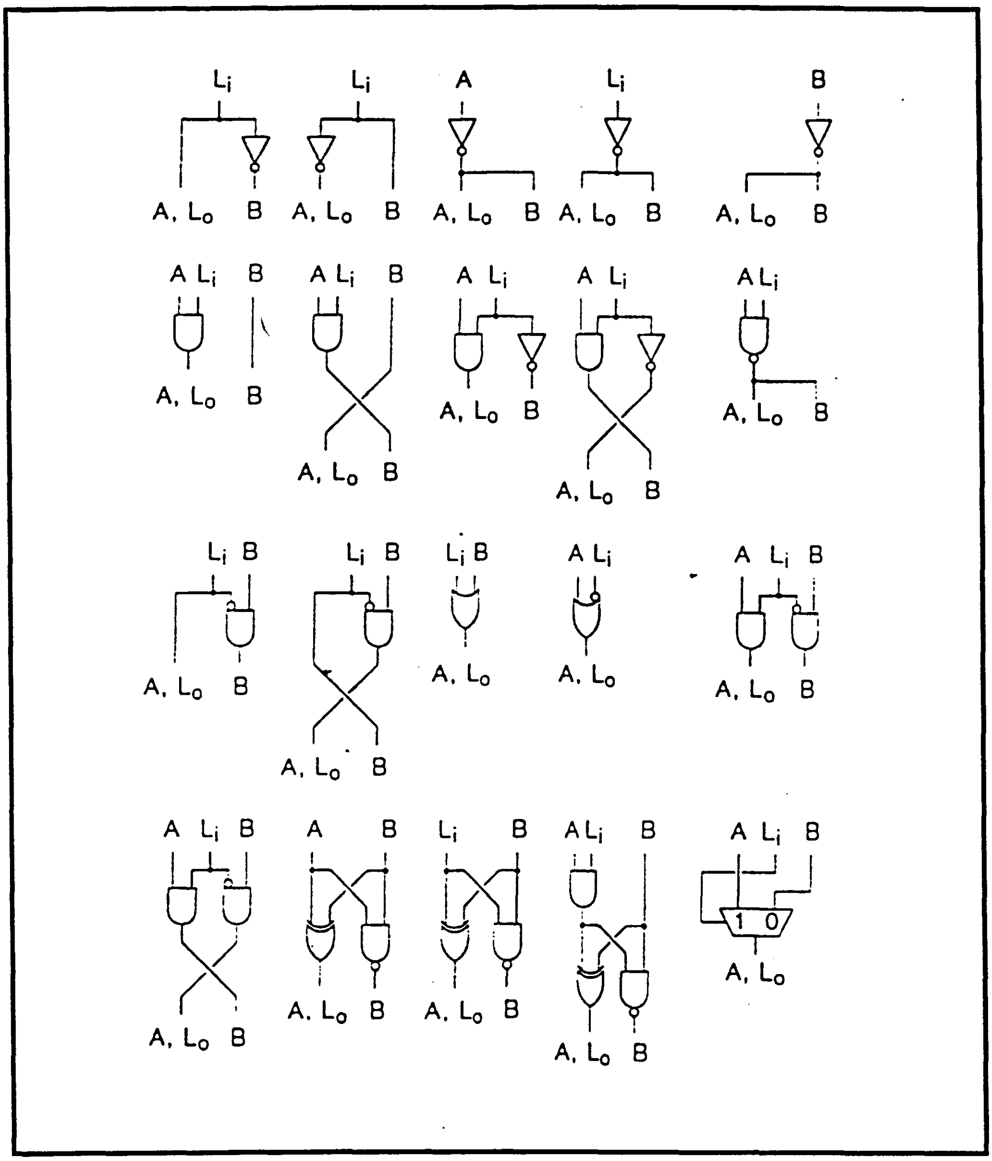


Figure 2.4: Combinatorial states of ATMEL macrocell

(or macro-cells), twenty of them are purely combinatorial cells which provides all primitive logic functions like NOR, NAND, AND, OR, 2-input multiplexer, and some combinations of the primitive gates as shown in Figure 2.4. AND/EXOR realization is of special interest to us. The logic synthesis method, Permuted-Read-Muller-Tree (PRMT) approach (which will be discussed in Chapter 3) decomposes a Boolean function to an AND/EXOR tree. The AND/EXOR function can be realized in a logic cell of ATMEL.

2.4 Architecture Restriction

Each cell has only one input from the local bus and at most two inputs from the neighbor cells and one output can be either to next adjacent cell or to local bus. All possible input configurations are shown in Figure 2.5 and can be described as follows bellow:

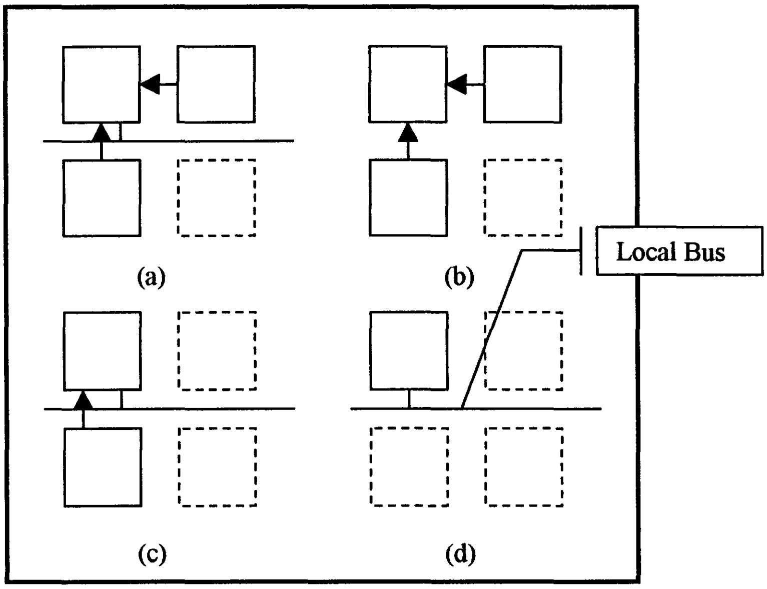


Figure2.5: Four input/output configurations to a cell

(a) Two inputs from the adjacent cells; from left and right or bottom and right or bottom and left. However, one input to a cell must be signed "A" and another input must be signed "B". Input is from local bus.

(b) Two inputs from the adjacent cells (from left and right/ bottom and right/bottom and left).

(c) One input is from adjacent cell (left or right or bottom) and another input is from local bus.

(d) Only one input from local bus.

CHAPTER 3

LOGIC SYNTHESIS

In order to understand the technology mapping method proposed in this thesis, logic synthesis methodology will be first overviewed in this chapter. An in-depth analysis of a binary tree approach to logic synthesis will be presented in order to explain the connections between the logic synthesis and the proposed technology mapping.

3.1 Logic Synthesis Overview

The ultimate goal of automatic synthesis of a digital system is to provide the transformation of the functional specification of the system down to the physical implementation. It involves three major design tasks:

- Functional/behavioral modeling and definition
- Logical design
- Physical implementation

Logic synthesis (as a way of logic design) is an essential part of this design process. It starts from the functional description of the circuit or system, and ends with the description, typically referred to as a net-list, which depends on the particular target technology. In general, logic synthesis consists of two steps; translation from a *Register-Transfer* level representation written in a hardware description language like VHDL to a *gate-level* circuit, and optimization of the *gate-level* circuit.

3.1.1 Two-Level Logic Optimization

One of the most celebrated problems in logic design of digital circuits is the minimization of Boolean functions, i.e., the reduction of the number of logic gates/devices needed to implement a given function. Since a Boolean function can be conveniently represented algebraically in its two-level sum-of-products form, the minimization of a two-level function is equivalent to the minimization of the number of product terms in the expression. This minimization problem has been extensively studied during the last three decades, and many important results have been obtained. There are a variety of two-level logic implementations. The most common one is the sum-of-products implementation as indicated above, where the first level of logic corresponds to AND gates and the second level to OR gates. NOR-NOR structures, NAND-NAND structures, AND-XOR structures, and OR-AND structures are also possible all have been investigated.

The minimization of product terms is important in VLSI circuits, where two-level Boolean function can be easily implemented as a Programmable Logic Array (PLA). PLA design is an attractive custom design methodology due to its simplicity, regularity and flexibility. The mapping of symbolic representation of two-level logic onto PLA structure is straightforward, and logic optimization is equivalent to the minimization of two-level sum-of-products expression. On the other hand, another reason is that two-level logic optimization has many applications in other areas of logic synthesis and computer-aided design. It is used for the design of PLA's with input and output decoders, PLA partitioning, state machine assignment, encoding of micro-programmed control units, and multi-level logic synthesis. These logic minimization techniques have become well understood, and a number of efficient computer programs have been developed. Topological optimization methods, such as PLA folding and partitioning, have been also developed and are being used in industry [4,6,12].

3.1.2 Multilevel Logic Optimization

Because multilevel logic can often result in a faster or smaller implementations of a function than two-level logic, synthesis of multilevel logic has received considerable attention over the past decade [14,15]. Efficient algebraic optimization methods were proposed [16] and successfully implemented in the MIS-II program [15]. The program BOLD [14] uses Boolean optimization methods that exploit external and internal "don't-care conditions. The program SOCRATES [13] uses a rule-based approach combined with an algorithmic approach for area and timing optimization. A comprehensive treatment of the state-of-the art in multilevel logic optimization can be found in program. It is of great interest to analyze the algebraic and Boolean transformations, used in the various multilevel logic optimization programs, from the standpoint of testability. Constraining these transformations can result in highly testable circuits. However, constraining logic optimization may adversely affect the area and speed of the resulting design.

3.1.3 EXOR Based Logic Synthesis

The basic logic synthesis methods based on EXOR gates, which are very attractive because they lead to the same functions being represented with a smaller number of gates (which means a smaller number of logic blocks needed to implement the circuit on the FPGA) and also because most of the FPGAs include EXOR gate in their logic blocks. For a long time, EXOR gate has been considered not useful for circuit implementation because its realization in silicon especially for large fan-in or fan-out was slow compared to other simple gates. However, with introduction of FPGAs, the delay of an EXOR gate became similar to the delay of other gates. For some types of FPGAs, like for example LUT-based Xilinx series, the delay of the logic block depends on the number of input variables and not on the functions realized by that block. In CA-Type FPGAs which are based on small granularity of their logic blocks and localized connections, the fan-in and fan-out of the EXOR gate are low [3,4,6,12]. Recently, EXOR gates have been more often used for the implementation of Boolean function due to easier testability. It has been already shown that AND/EXOR representation of linear and nearly linear functions costs less (in a number of gates) than the inclusive (AND/OR) representation.

3.2 Binary Tree Approach for Logic Optimization

A variety of Boolean function representations have been developed. Classical representations like sum-of-products, truth tables and Karnaugh maps are impractical because any function of n variables has a representation of size 2^n. Representations like the set of prime and irredundant cubes and Boolean network are generally used. However, such representations suffer from some critical drawbacks. First, certain common functions may require representations of exponential size. Second, simple operations like complementation may yield a function with exponential size. Finally, some of these representations do not have a canonical form, i.e.; a function may have different representations. This makes it difficult to check for equivalency and tautology.

Binary decision diagrams (BDDs) were first proposed by Lee [19]. This approach was further developed by Akers [17]. As such, BDDs are not canonical. Bryant introduced restrictions on the ordering of variables and proposed a reduction algorithm, which transformed BDDs to be reduced, ordered BDDs (ROBDDs) [18] that have a canonical form. Many logic optimization methods have been developed based on BDDs and their variations. In this thesis, we will focus on Functional Decision Diagrams (FDDs) representations of logic functions since the input data, i.e. the technology independent logic optimization results, are from a method called Permuted Reed-Muller Trees (PRMT) (i.e. a form of function decision diagrams (i.e. a form of FDD). FDDs was developed based on BDDs by substituting Shannon Expression with Davio Expansions. In the exit section, we will discuss Davio expansions, PRMT and a binary tree.

3.2.1 Davio Expansions

One of the most fundamental concepts for the decomposition of a logic function is the Shannon expansion. The Shannon expansion can always be applied to a logic function in contrast to other types of Boolean decompositions like the *Ashenhurt* [9] or the *Curtis* [21] decomposition, which can be applied only to certain classes of functions. By applying certain rules to the Shannon expansion we can generate the Davio I and Davio II expansions as shown below. The well-known Davio expansion [20] is given in Figure 3.1.

The decompositions represented by equations (2) and (3) are called Davio I and Davio II, respectively. The circuit realization of equation (1) is given by a multiplexer gate, while equations (2) and (3) describe and AND-EXOR gate structure, as shown in Figure 3.1. Since we have chosen to use ATMEL 6000 series FPGA as our target architecture, AND-EXOR combination, which can be realized in one logic cell of that architecture, is of special interest.

$$(1) \qquad f = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}$$

$$(2) \qquad f = f_{x_i} \oplus \overline{x_i} f_{x_i} \oplus f_{\overline{x_i}} = f_{x_i} \oplus \overline{x_i} g$$

$$(3) \qquad f = f_{\overline{x_i}} \oplus x_i f_{x_i} \oplus f_{\overline{x_i}} = f_{x_i} \oplus \overline{x_i} g$$

Figure 3.1: Shannon Davio I and Davio II expansions

Any combination of Shannon and Davio I and II expansions can be used to produce decision diagram [1]. A Binary Decision Diagram (BDD) is a Directed Acyclic Graph (DAG) with a single root node. The terminal (leaf) nodes represent the values 0 and 1, while non-terminal nodes represent Boolean functions. The function associated with the root node specifies the function represented by the entire BDD.

Each non-terminal node has an associated variable and two outgoing edges. The function represented by the non-terminal node is specified by its cofactors with respect to its associated expansion variable.

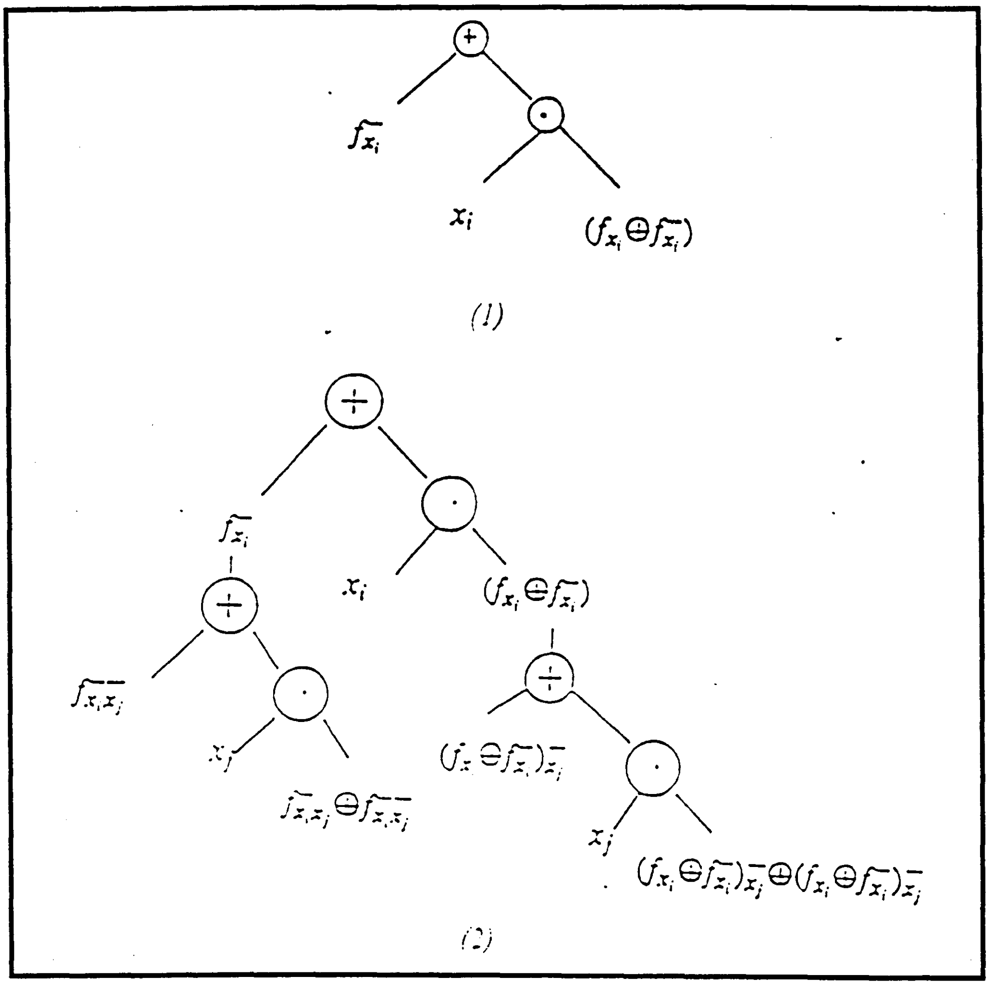
3.2.2 Permuted Reed-Muller Trees (PRMT)

A new decomposition method, which generates AND/EXOR tree representation, has been proposed by Perkowski [1]. The approach is called Permuted-Reed-Muller-Tree (PRMT). PRMT is the expansion tree in which all variables appear in positive polarity but in each sub-tree the decomposing variables could be in a different order. A given Boolean function can be decomposed using equation (2):

$$f = f_{\overline{xi}} \oplus x_i\, g_{xi} \text{ where } g_{xi} = f_{\overline{xi}} \oplus f_{xi}$$

The two sub-functions $f_{\overline{xi}}$ and g_{xi} are independent of the variable x_i. The AND-gate takes (x_i) and g_{xi} as its two inputs. The EXOR-gate takes ($f_{\overline{xi}}$) and the output of the AND–gate as its two inputs. Their relationship is shown in Figure 3.2:(1). For each sub-function $f_{\overline{xi}}$ and g_{xi}, choosing another variable, such as (x_j), the Davio Expansion is used to continue the decomposition. After the further decomposition, both functions $f_{\overline{xi}}$ and g_{xi} will be decomposed into another two sub-functions, which are connected by an AND-gate and also EXOR-gate to extent to next level in the PRMT, see example in Figure 3.2:(2).

The decomposition is repeated recursively until the functions are all trivial (0,1,x_j). After completing the decomposition, a tree representation of a circuit is created. The output of the tree representation of a circuit is the original function **f**. In the tree representation of the circuit, all input variables appear in positive polarity and are connected by AND-gates and EXOR-gates. The tree representation of the circuit is called the PRMT. The PRMT can be presented in the *Blif format* by the program REMIT [1], which is as input file for our algorithm.



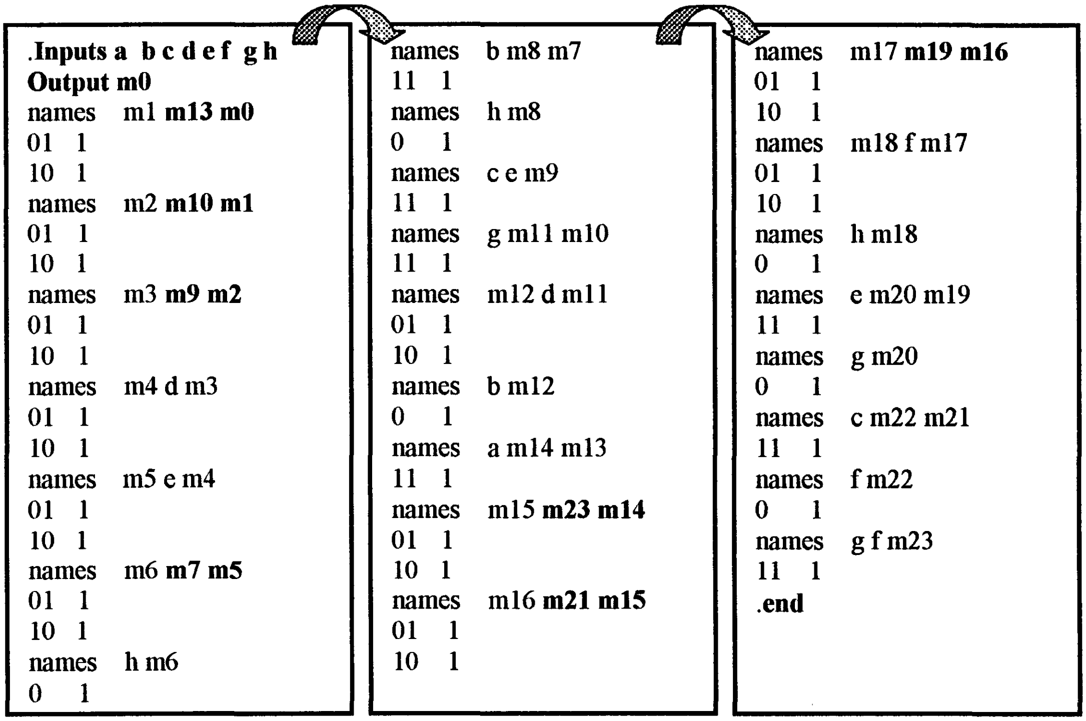
$f_{\bar{x}_i}$
x_i
$(f_{x_i} \oplus f_{\bar{x}_i})$
(1)
$f_{\bar{x}_i}$
x_i
$(f_{x_i} \oplus f_{\bar{x}_i})$
$f_{\bar{x}_i \bar{x}_j}$
x_j
$f_{\bar{x}_i x_j} \oplus f_{\bar{x}_i \bar{x}_j}$
$(f_{x_i} \oplus f_{\bar{x}_i})_{\bar{x}_j}$
x_j
$(f_{x_i} \oplus f_{\bar{x}_i})_{x_j} \oplus (f_{x_i} \oplus f_{\bar{x}_i})_{\bar{x}_j}$
(2)

Figure 3.2: Example of expansion tree

3.3 Input Data File

The objective of this thesis is to develop a technology mapping for the CA-Type FPGA. The logic optimization will not be the major concern. Instead, we will use the results of the PRMT. The output file from the PRMT is in the *Blif format* which is used as input data files to our approach. Our algorithm starts by reading *Blif format* of the PRMT, next a Cell Binary Tree (CBT) is generated by using our grouping algorithm (which will be discussed in detail in chapter 4).

In order to illustrate our approach, we'll look at an example that will show all the steps of our algorithm. Example "*Example.blif*" is given most possibilities of our algorithm and its *Blif format* is given bellow in Figure 3.3.



```
.Inputs a  b c d e f  g h
Output m0
names   m1 m13 m0
01  1
10  1
names   m2 m10 m1
01  1
10  1
names   m3 m9 m2
01  1
10  1
names   m4 d m3
01  1
10  1
names   m5 e m4
01  1
10  1
names   m6 m7 m5
01  1
10  1
names   h m6
0   1
names   b m8 m7
11  1
names   h m8
0   1
names   c e m9
11  1
names   g m11 m10
11  1
names   m12 d m11
01  1
10  1
names   b m12
0   1
names   a m14 m13
11  1
names   m15 m23 m14
01  1
10  1
names   m16 m21 m15
01  1
10  1
names   m17 m19 m16
01  1
10  1
names   m18 f m17
01  1
10  1
names   h m18
0   1
names   e m20 m19
11  1
names   g m20
0   1
names   c m22 m21
11  1
names   f m22
0   1
names   g f m23
11  1
.end
```

Figure 3.3: Blif format

Example "*Example.blif*" has eight variables. In Figure 3.3, where

- Input: list input variables
- Output: list function outputs (one output only: m0)

Names: lists input name and output name, and specify a function represented by the given name. (When 01 or 10 = 1 is an EXOR gate, 11 = 1 is an AND gate and 0 = 1 is a NOT gate), and nodes connections [Example: m0 (as a Root node) is an EXOR gate and it connects to m1 and m13].

CHAPTER 4

TECHNOLOGY-DEPENDENT BINARY TREE APPROACH

The result of logic optimization is an optimized gate-level (also called logic-level) net-list with combinational sub-circuits reintegrated with sequential memory elements. This net-list is composed of generic components such as NORs and NANDs. The next step is to efficiently map this net-list into a library of gates available from the semiconductor vendor. This step is called technology mapping. Simply translating a net-list of generic components into a cell library is not a challenging process. The real challenge lies in maximally utilizing the components in the library such that the resulting net-list realizes its area, speed, and testability goals. In this chapter, a new approach for technology mapping for ATMEL 6000 is proposed with the aim of reducing the area and routing resources. The detailed implementation of the proposed approach will be presented in the following.

4.1 Approach Overview

In this section, we propose to use the binary tree for our approach because the binary tree is used for most technology mapping into the different type of FPGAs (CA-Type). However, the results of technology mapping aren't completely ideal for CA-Type FPGAs because the important problem of routing of the same variable in the different levels is not solved efficiently. Following our proposed approach, four major steps to obtain the goal of reducing the area and the requirements of routing resources are presented in this section. The four steps which will be discussed later in this section are:

- Grouping and generating Cell Binary Tree (CBT)
- CBT placement.
- Bus assignment
- Variable routing in mapping area.

These steps are also illustrated in Figure 4.1. A brief introduction of these steps is presented in the following.

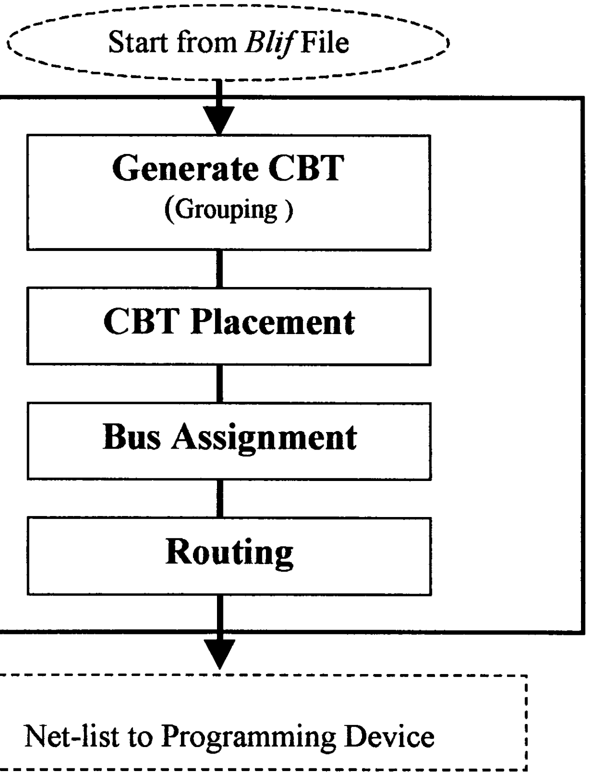


Figure 4.1: Approach overview

As shown in Figure 4.1, we start from reading *blif* file, which is the result of logic synthesis. After initializing the tree structure, a Cell Binary Tree(CBT) targeted for ATMEL 6000 is generated by using a grouping algorithm which will be discussed in detail in the following sections. In general, the grouping algorithm will identify the EXOR_AND, NOT_AND, AND, EXOR and NOT cells, and group them together into CBT form. The following limitations are applied to each cell: each cell has only one output, and the number of inputs for each cell is limited to three. These limitations are specified for ATMEL 6000 architecture.

A placement stage in this thesis is to map CBT onto the CA-Type cell array of the ATMEL 6000 Series FPGA. The significance of the placement method is to line-up the cells with the same variable into the same row or column of the mapping area when each cell is mapped. There are two different cases that need to be taken into consideration for the CBT placement:

- **Search for the same variable:** In this stage, when each cell is mapped into a cell block in the mapping array, the cell will horizontally and vertically search whether there is a same variable on the row or column. We will place the cell with the same variable name in the row or column with the corresponding variable name.
- **Leaf cell reposition:** This procedure moves leaf cells to match the same variable in the scatter mapping area after a CBT is completely mapped into the CA-Type cell array. A cell can be allowed to move to the previous

row on the left or right column or down to the next level, if the same variable can be found from the next level.

Considering these two situations will make variable routing much easier and generate better results (see Chapter 5 for detail). This is because the number of local buses with the same variable is reduced by our placement method.

The bus assignment stage is presented for the variables of cells to be assigned to horizontal and vertical local buses. The ATMEL 6000 Series FPGA has two vertical buses (top and bottom) and two horizontal buses (left and right) around each cell and a signal is connected from local buses to the cells. By our bus assignment method, the same variable needs to be connected by local buses by row line or column line. Bus assignment consists of the following two basic methods: (1) finding the largest number of the same variable from a row or a column. (2) Selecting the buses from row line to assign the same variable name with the largest number of cell, if horizontal buses are free (top or bottom), otherwise, select column line buses.

The routing stage is introduced to connect variables, which are not connected by bus assignment, with the same variable name in different place in the placement array. There are three basic ways to interconnect the same variable buses from the different comports of the mapping area:

- row to row
- column to column
- row and column

groupings are executed until all is done. A detailed illustration of this process is given in Figure 4.10.

The Cell Binary Tree (CBT) is constructed from input file (*Blif format*) and built by starting at root node (output of the logic function, i.e. m0), the connectivity of the left side of the node is traced and grouped as needed until all branches in the left side become exhausted. Then the connectivity of the right branches is explored and groupings are executed until all is done. A detailed illustration of this process is given in Figure 4.10. In Figure 4.10, some of the nodes can be grouped into EXOR_AND cells by the following order: m0/m13 to cell 0EA.a, ml/m10 to cell 1EA.f, m2/m9 to cell 2EA.c, m5/m7 to cell 5EA.g, m14/m23 to cell 14EA.d, m15/m21 to cell 15EA.g and m16/m19 to cell 16EA.e. Also, two NOT gates (m6 and m8) can be grouped into one cell block 6N.h, which has two outputs to 5EA.g. In addition, a "sw" needs to be added between two EXOR cells or EXOR and EXOR_AND. Also, an Extend Cell (ec) needs to be added for a variable of m2 and m14 nodes to extend to the next row.

4.2.2 Description of Placement

The objective of the placement algorithm is to map the CBT onto the CA-Type cell array (i.e. ATMEL 6000). The significance of the placement method in this thesis is to line-up the cells with the same variable into the same row in the mapping area. Two different cases need to be taken into consideration:

- **Search for the same variable name**: When each cell is mapped into a cell block in the array, the mapping algorithm will horizontally and vertically search for the row or column with the same variable name. It will place the cell with the same variable name in the row or column with corresponding variable name.

- **Leaf cell reposition after CBT is mapped**: It moves leaf cells to match the same variable in the scatter mapping area after a CBT is completely mapped into CA-Type cell array. It can only be allowed to move the previous row or to the left or to the right column or down to the next level, where the same variable can be matched.

By considering these two situations, Leaf cell reposition will make variable routings much easier. Due to that the fact that the number of local buses with the same variable names is reduced by our placement method.

4.2.2.1 Three Types of Tree Form in CBT

There are three types of tree forms in CBT. See example in Figure 4.11. The explanations for all types are given below:

- *Full Tree Form* (FTF): a cell in a branch has two inputs from adjacent cells in the CBT and one variable input is from a local bus.
- *Single Tree Form* (STF): a cell has only one input from an adjacent cell in the CBT and another input is a variable from a local bus.
- *Leaf Tree Form* (LTF): a cell is the last node in a branch of the CBT and has only one input variable from a local bus.

All three forms are very important for CBT placement because each form has the different rules of placement algorithm. More details are described in the next sections.

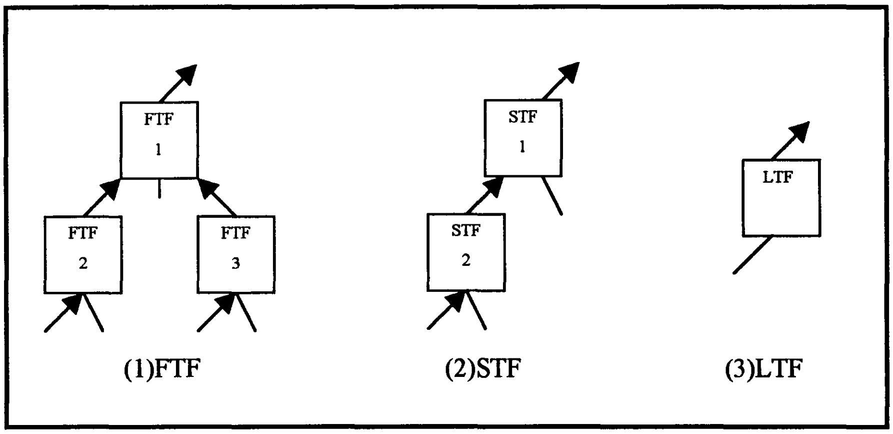


Figure 4.11: Three tree forms in the CBT

4.2.2.2 CBT Placement

4.2.2.2.1 Full-Tree Form (FTF) Placement

There are three ways to map FTF into a CA-Type of ATMEL 6000 FPGA. An example is presented in Figure 4.12.

These three cases are:

- Figure 4.12 (1) shows that if the FTF is the root of CBT, three cells of the FTF are horizontally mapped into the first row of the cell array.
- Figure 4.12 (2) shows the case when the FTF is the inside of the CBT. The left cell (L) of the FTF can be vertically placed next to the branch cell (B) of the FTF (left column). The right cell (R) of the FTF can be horizontally placed in the same column of the branch cell (B) of the FTF (next row).

- Figure 4.12 (3) also shows that the right cell (R) of the FTF can be vertically placed next to the branch cell (B) of the FTF (right column). The left cell (L) of the FTF can vertically be placed in the same column of the branch cell (B) of the FTF (next row).

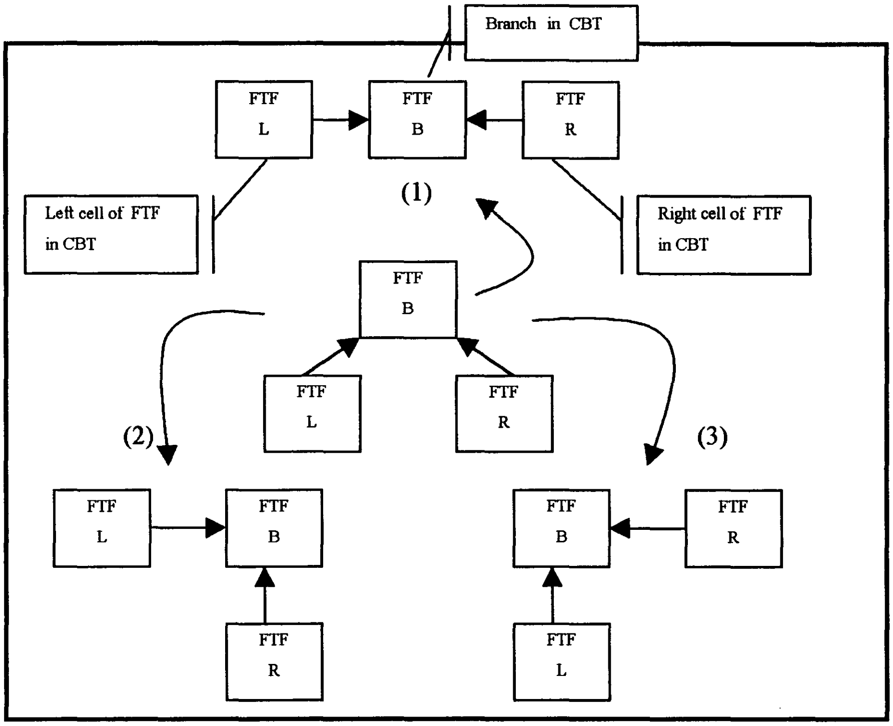


Figure 4.12: Mapping FTF cells in three ways

4.2.2.2.2 Single-Tree Form (STF)

The STF also has three ways to be mapped. See examples in Figure 4.13. Since the STF has one input from an adjacent cell and one from a local bus, in example Figure 4.13, the cell "1" is a previously mapped cell and the cell "2" is given three possible ways to be mapped. All three mapping methods depend on the row and column space and the variable names.

- Cell 2 can be horizontally placed next to the left column of cell 1 if the variable on the row of cell 1 is the same and a left side space is available.
- Cell 2 is vertically placed down to the next level of cell 1 if the space is available.
- The same as (a) in Figure 4.13, but its placement is horizontal and directed to the right column if the variable on the row of cell 1 is the same and a right side space is available.

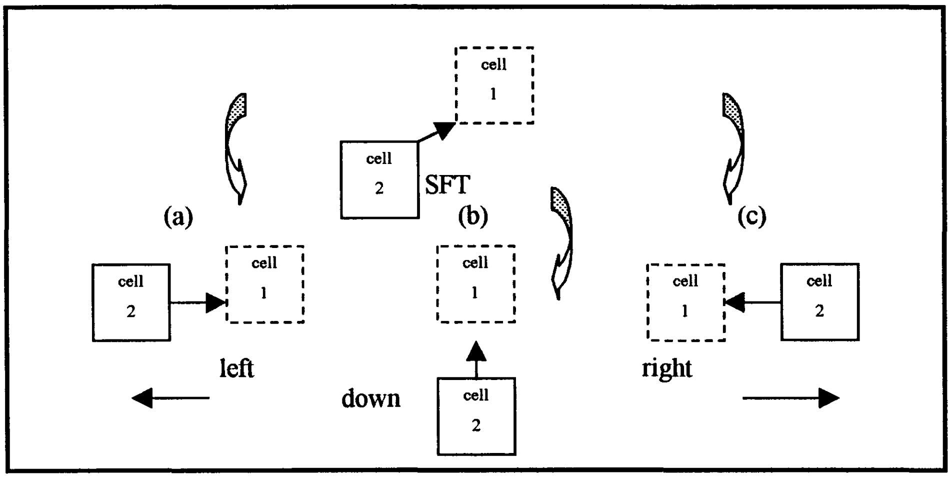


Figure 4.13: STF placement

LTF placement follows STF methods to map onto the cell array. If LTF has the same shape as the STF, the placement method is similer to the STF description. See STF example in Figure 4.13.

4.2.2.2.3 Placement Algorithm with the Same Variable Search

The basic rules of CBT placement algorithm are as follows:

- Start by mapping the root cell (output of function, i.e. Cell_0).

- Compare the numbers of cells on both sides of the branch cell to find the longest branch in the CBT.
- Map the CBT cells into ATMEL arrays
  - (a) If a cell has two inputs from the left and right cells, then mapping this cell follows the FTF placement.
  - (b) If a cell has the input from an adjacent cell or it is the last cell in the branch of the CBT, then perform the mapping following the STF placement.
- Stop mapping, when the mapping cell has *no where to go* in the mapping array. Then select a bigger size of array and start from the beginning.

The pseudo code for CBT placement is given by the programming function *mapping(node).* The *mapping(node)* starts by assigning the ROOT node (cell) of the CBT into CA-Type array (i.e. 8x8 array).

Assign the ROOT cell to the matrix based on the size of nodes on each side. The rules are as follows:

- If the number of nodes in the left side of the tree is two times greater (or equal) than in the right side of the tree, the ROOT cell is put on the first row in cells array (Example: root cell position is on [0][5] for the 8x8 array).
- if the numbers of nodes in the left tree are greater or equal to the right tree, but also less than two times right nodes, then the ROOT cell is put on the first row in cells array (Example: root cell position is in [0][4] for the 8x8 array).
- If the numbers of nodes in the right of tree are two times greater (or equal) than

the left of the tree, the ROOT cell is put first row in cells array (Example: root cell position is in [0][2] for the 8x8 array).

```
Mapping(node)
NODE_PTR node;
{
  NODE_PTR ptr;
  factor_flag = 0, exit_flag = 0;
  node ->flag = 1;
 if (compare_left_right(node) == TRUE){
    if (factor_flag == 1) /* left nodes > right nodes */
    {
         Root_node position:        matrix[0][5] if lefty >= 2*righty
                                    and matrix[0][4] if righty <= lefty, 2*righty;
                      do_left_first(node, prt)}
    else
    {
    if (factor_flag == -1) /* right nodes > left nodes */
       {
         Root_node position:        matrix[0][2] if righyt >= 2*lefty
                                    and matrix[0][3] if lefty <= righty <  2*lefty;
                      do_right_first(node, prt)}
    }
}
```

Pseudo-Code of Root Node Placement

- If the numbers of nodes in the let size tree are greater than and equal to the left of the tree, and also less than two times the number of left nodes. Then, the ROOT cell is put first row in cells array (Example: root cell position is in [0][3] for the 8x8 array).

After the ROOT cell is mapped, the children of the ROOT on the two sides of the ROOT cell are placed into the same row of the ROOT cell. The *mapping(node)* is called.

The routines in the function *mapping(node)* are as follows:

- *compare_left_right(node)* compares the number of nodes on the left and right. If the numbers of nodes are greater or equal to the right size nodes on the left then left size node maps first. Otherwise, right node maps first.

- *do_right_first(node, ptr) and do_left_first(node,ptr)* are the two major functions being used for mapping technology, they are both recursive functions in the program. The difference between *do_left_first(node,ptr)* and *do_right_first (node,ptr)* is when the order is called from *mapping(node,optr)* that if left nodes' number are greater than right, the *do_left_first(node,ptr)* is called, otherwise, *do_right_first(node, ptr) is called.*

```
void do_right_first(node, ptr)
NODE_PTR node, ptr;
{
  if (ptr -> left == NULL && ptr -> right == NULL); return;
  else
  {
    ptr = do_placement (node, ptr); /* do nodes mapping */
                 if (ptr == NULL)
                 {
                          if (ptr == node); return }
          if (compare_left_right(node) == TRUE)
          {
                 if(ptr -> flag == 0)
                 {
                 do_left_first(node, ptr)}
                 if(exit_flag == 1)
                 {
                 end mapping.  return;}}
          else
          {
                 if(ptr -> flag == 0)
                 {
                 do_right_first(node, ptr)
                 }
                 if(exit_flag == 1)
                 {
                 end mapping.  return;} } }
 return
}
```

Pseudo-Code of Right Node Placement

An example for *do_right_first(node, ptr)*, gives in pseudo-code, is on the last page. The routines in the function *do_right_first(node, ptr)*: *do_placement(node,ptr)* shows that if ptr (ptr -> node type) is a full-tree node, *do_full_placement(node,ptr)* is called. Otherwise, call function *do_single_placement(node,ptr)*. The *do_right_first (node,ptr)* and *do_left_first(node,ptr)* are the same in function *mapping(node)*

The pseudo-code for *do_placement(node, ptr)* is given below. The routines in the function *do_placement(node, ptr)* are:

```
NODE_PTR do_placement( node, ptr )
NODE_PTR node, ptr;
{
        if ( is_it_full_tree( ptr ) == TRUE )
        {
                ptr = do_full_placement( node, ptr );
                fflush( stdout );
        }
        else
        {
                ptr = do_single_placement( node, ptr );
                fflush( stdout );
        }                               /* Pointer has been updated. */
        return ( ptr );
}
```

Pseudo-Code of Cells Placement

- *do_full_placement(node, ptr)*: Get the variables of the left node, also get the variables of the right node. There are three possibilities to place the full tree nodes.

  (1) left-left, right-right

  (2) left-left, right-down

  (3) left-down, right-right

The way of putting the cell into the position of matrix [i][j], where ***i*** is row number from 0 to *Ni* and ***j*** is column number from 0 to *Nj* (*Ni* and *Nj* are matrix size from *8 to 56for ATMEL chip),* is based on the rule, which the same variable goes to the same row (horizontal line) or column (vertical line). After the node is placed, checking whether the node number on the left side of the parent node is bigger than the right side. Then return the left node pointer for future use. Otherwise, return the right node.

- *do_single_placement(node, ptr)*:Node is not a full tree. If both left and right are NULL, return NULL, otherwise, do placement based on the same variable rule.

  (1) left-left or left-down

  (2) right-right or right-down

  If left is not NULL, return the left of the parent node. If the right is not NULL, return right of the parent node.

4.2.2.2.4 Placement Example

In order to understand our placement algorithm, a complete example is given in Figure 4.14. In the example we realize the physical layout of the CBT into the 8x8 cell array layout of the ATMEL 6000 FPGA by following our placement algorithm. A step by step description is given in the following (all the node names are referred to Figure 4.10):

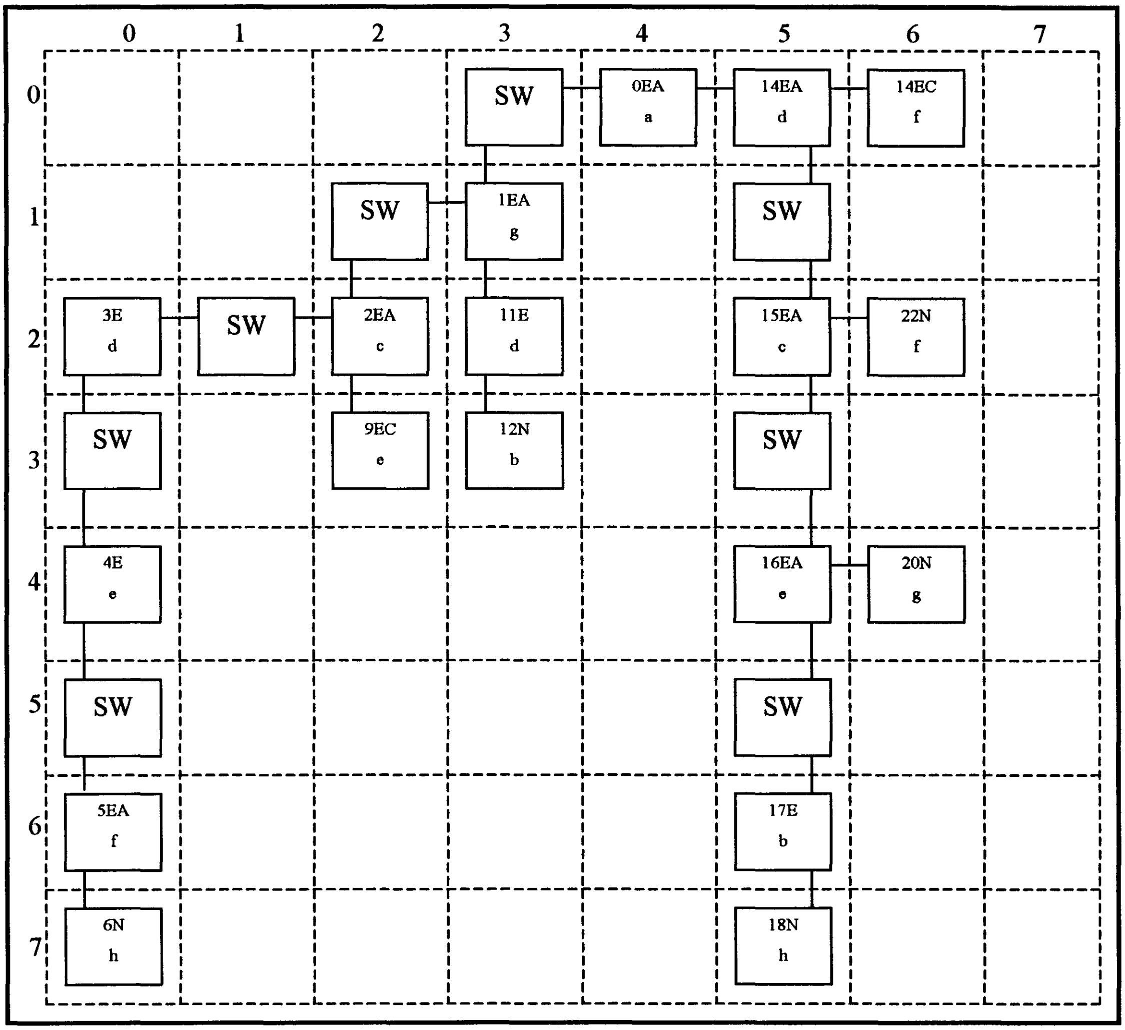


Figure 4.14: CBT placement

Step1: Root cell placement: Node “0EA.a” is placed onto matrix[0][4] (i.e. ATMEL 8x8 array), a “sw” is placed onto matrix[0][3] and node “14EA.d is assigned to matrix[0][5] because the numbers of cell nodes in the left tree are greater or equal the right of the tree, and also less than two times the right nodes.

Step2: Place the left CBT first:

1. Parent node "1EA.g" is placed into matrix [1][3] and its two children nodes "sw" (left to left) and "11E.d" (right to down) are placed into matrix [1][2] and matrix [2][3].
2. Parent node "2EA.c" is placed into matrix [2][2] and its two children nodes "sw" (left to left) and "9EC.e" are placed into matrix [2][1] and matrix [3][2].
3. "3E.d" is placed into matrix [2][0] because on the same line in row "2", it has the same variable "d".
4. Placed "sw", "4E.e", "sw", "5EA.b" and 6N.h" cells into the same column of "3E.d" because there are not any other parent nodes and variable compression.
5. Back to place cell "12N.b" into matrix [3][3] (it is the right node of parent nodes in the left side of CBT).

Step3: Do right after left is finished:

1. "sw" and "14EC.f" must be placed as left to down and right to right because this is only way they can be placed, so "sw" is placed into matrix [1][5] and "14EC.f" is into matrix [0][6] after "14EA.d" was placed to matrix [0][5] in Step 1.
2. "15EA.c" is a parent node and it is placed into matrix [2][5] because the same variable "c" is found in the same row "2". Two children nodes, "22N.f" is placed into matrix [2][6] because the same variable "f" is found in the column "6" and "sw" is placed into matrix [3][5].
3. "16EA.e" is also a parent cell and placed into matrix [4][5] because the same

variable "e" is found in the row "4". Two children nodes are placed as following right to right and left to down, so "20N.g" is placed into matrix [4][6] and d "sw" is placed into matrix [5][5].

4. "17E.b" and "18N.h" are SFT and they cannot find any comparison so they are placed into the same column "5". Here the placement is terminated after all right side of CBT is done.

4.2.2.3 Leaf Cell Reposition (LCR)

In this section, Leaf Cell Reposition (LCR) method, which is for inducing numbers of the same variable name in the dispersing mapping area, is presented. The method allows us to move a leaf cell to another row or column if the same variable can be matched. There are also three ways to reposition the leaf cell. See example in Figure 4.15.

From the example in Figure 4.15, we assume that cell "2" is a leaf cell in the CBT and it can be moved in one of the following three ways:

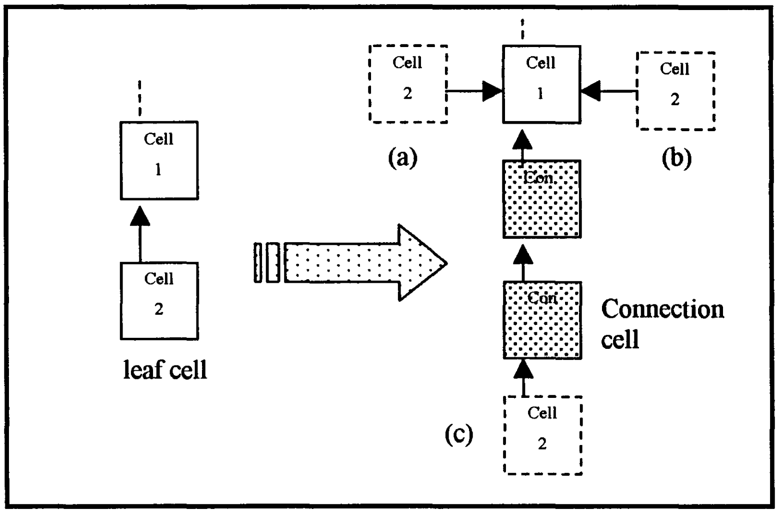


Figure 4.15: Three ways to reposition leaf cell

- Move up one row to the left column of cell "1", if that row has a matching variable name.
- Move up one row to the right column of cell "1", if that row has a matching variable name.
- Move down to next row along the same column if the row is matching variable name. In this move, the connection cells must be added between cell "1" and cell "2". The numbers of connection cells need to be added that depend how many steps cell "2" is to be moved down.

The example of the leaf cell reposition is given in Figure 4.16, which continues the example of the CBT placement from Figure 4.14 on the page 47.

After the placement is completed, the leaf cells can be moved if the same variable that can be matched in another row. In Figure 4.16, "2EC.e" can be moved down to matrix[4][2] from matrix[3][2] and "12N.b" can be moved down to matrix[6][3] from matrix[3][3]. After leaf cells are moved, Connection cells "Conn" need to be inserted.

The result of cell reposition reduced the number of the same variables on the rows. This process is important for bus assignment and routing because it can make easier progress for both algorithms. The basic rules for repositioning of the leaf cell are: (1) The leaf cells can be replaced in three ways such as left, right and down when the same variable names that can be found and space is available. (2) Only one level can be moved up to the left or to the right. (3) Leaf cells can be moved down to the next rows in the inside of the mapping area. (4) Leaf cell can not be removed if it has matched the same variable name on the column.

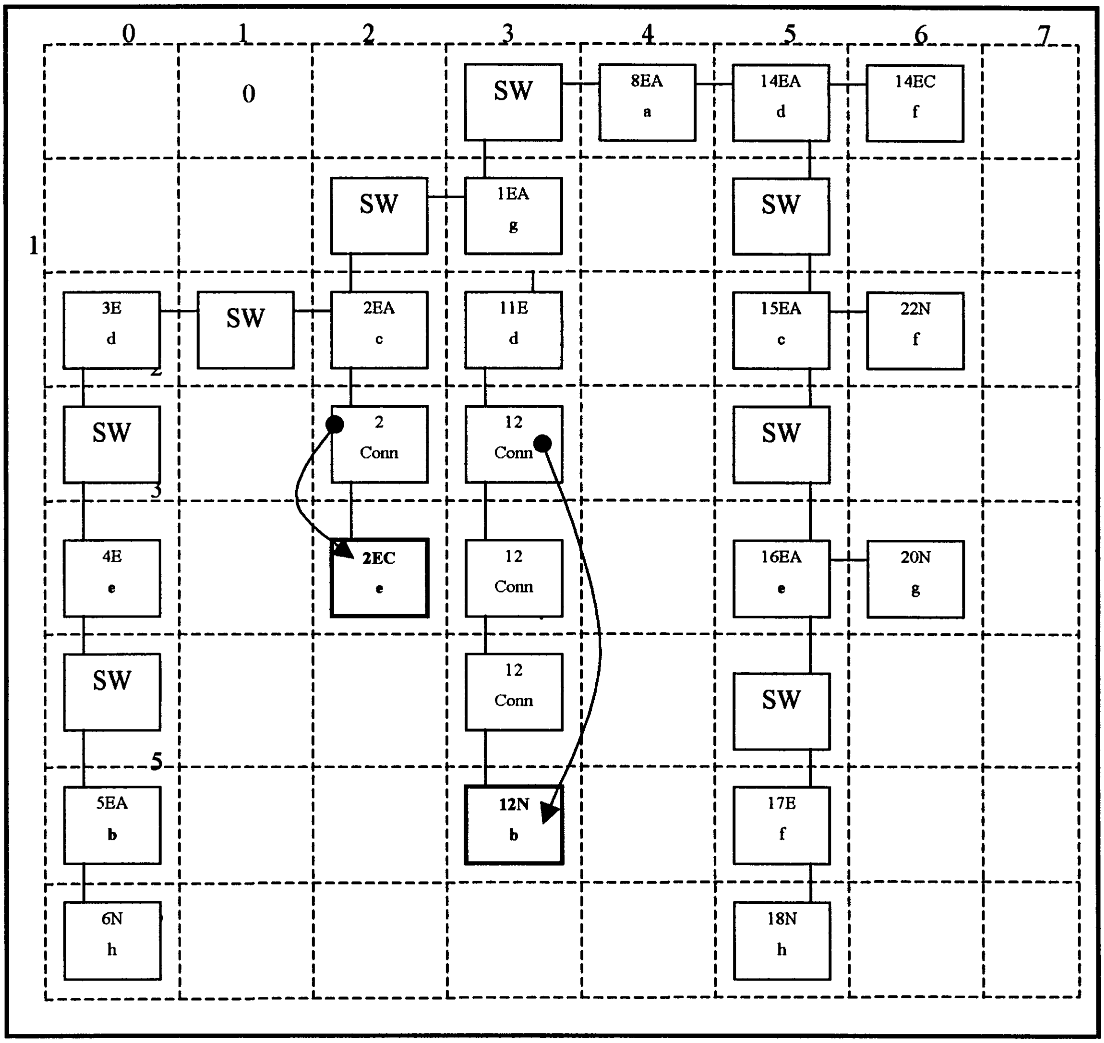


Figure 4.16: Leaf cell reposition result

4.2.3 Bus Assignment

The bus assignment algorithm is used for the connection between variables and local buses (horizontal and vertical) with assignment of variables. The ATMEL 6000 FPGA has two horizontal buses and two vertical buses. A signal can be connected from local buses to the cells, and each input variable needs to be connected in the mapping area. The assignment methods are discussed in the following section and followed by an example to further explain the methods.

4.2.3.1 Basic Rules of Bus Assignment

The basic rule of bus assignment is following the *inverted-pyramid* method, which starts by searching the same variable name with the largest number of cells from rows or columns. This is very important for our bus assignment methodology because we start to select a row that has the largest number of the same variable names, then to assign this row first. And then, find second largest number of the same variable names to assign and so on. This way can decrease the number of the routing buses. When the variable is found on a row, the top local bus should be selected to assign first if it is free, otherwise, it is assigned to the bottom local bus. On the other hand, if the variable is found on a column, the left local bus should be selected first if it is free, otherwise, it is assigned to the right local bus. If two different variables have the same number of cells, we select the first searched variable from the left on a row and assign it to the top local bus and the next to the bottom local bus if both are free. In summary, the bus assignment rules are as follows:

1. Start to search the variable, which has the largest number of cells with the same variable name from rows and columns in the mapping array.

2. When above indicated variable has been found from a row or a column, then assign that variable to the local bus to follow order: top and bottom or left and right.

3. If two different variables with the same number of cells have been found on the row or column, the rule for this case, which is searched first and assigned first.

4.2.3.2 Bus Assignment Example

Following our bus assignment rules, first find and rank the variables with the number of cells connected, and then assign variables to the local bus in the order of number of cells connected to the local bus. See the example in Figure 4.17 (which continues from Figure 4.16) in the next page. The following provides step by step explanation of how the bus assignment is carried out:

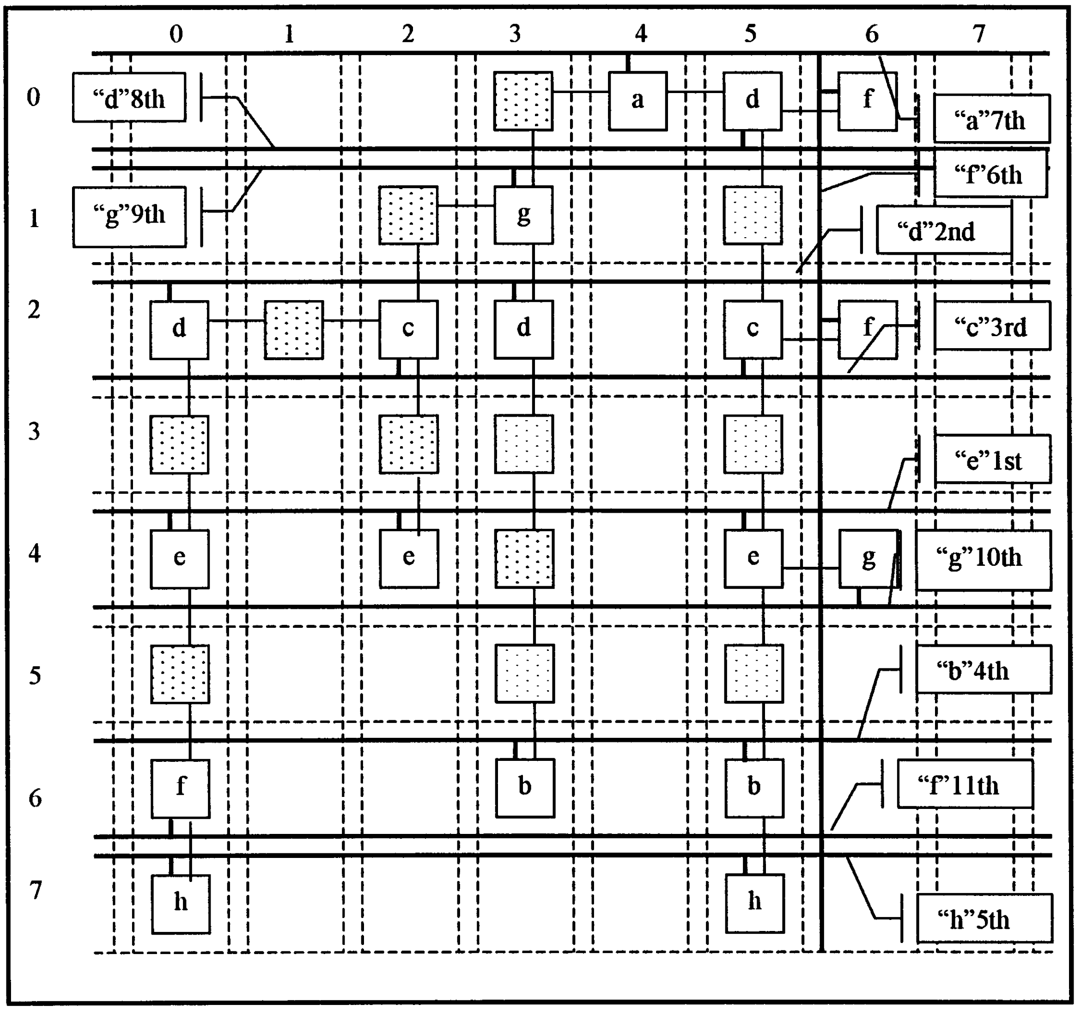


Figure 4.17: Bus assignment result

Step1: Variable "e" (3 cells) has the largest number of cells, found in "R4", and it is assigned to the top local bus of row "R4".

Step2 and 3: Variables "d" and "c" with the next largest number of cells (2 cells) are found in the same row "2". Variable "d" is assigned first to the top bus of row "2", because it is searched form the left, and then "c" can be assigned to the bottom bus of row "2".

Step4: Variable "b" (2 cells) is found in "6" and assigned to the top local bus of row "6".

Step5: Variable "h" (2 cells) is found in "7" and assigned to the top local bus of row "7".

Step6: Variable "f" (2 cells) is found in "6" and assigned to the left local bus of Column "6".

Step7: Variable "a" (1 cell) is found in "1" and assigned to the top local bus of row "1".

Step8: Variable "d" (1 cell) is found in "0" and assigned to the bottom local bus of row "0".

Step9: Variable "g" (1 cell) is found in "1" and assigned to the top local bus of row "1".

Step10: Variable "g" (1 cell) is found in "4" and assigned to the bottom local bus of row "4" because the top bus has been taken by variable "e" in ***Step1***.

Step11: Variable "f" (1 cell) is found in "4" and assigned to bottom local bus of row "4" because the top bus has been taken by variable "b" in ***Step4***.

The results of the bus assignment are illustrated in Figure 4.17. Three variables, “d” in row0 and row1, “f” in column6 and row6, and “g” in row1 and row4, are not completely connected. How to connect them will be discussed in the next section, and we will use our routing algorithm to solve this problem.

4.2.4 ROUTING

In this section a routing algorithm is presented to connect buses which have been assigned variables in different levels of mapping area. There are three major ways for variable connections between buses:

- row to row (vertical connection)
- column to column (horizontal connection)
- row and column (horizontal and vertical connection)

To accomplish the above connections, two methods are presented, Parallel Wire (PW) and Cross Wire (CW). Both methods use free cells in the mapping area to combine two buses, which have been assigned with a variable, and link them together using the shortest path if it is possible. Both PW and CW can easily find free cells in the mapping area and use them to connect buses. Both methods will be introduced in the following sections.

4.2.4.1 Parallel Wire

In this section, Parallel Wire (PW) means that there are buses with the same variable, which occupy more than one row or column. We can use free cells (as a routing cell “R”) to interconnect two local buses that have the same input variables. There are two ways of parallel routing:

1. Horizontally connect two column buses,

3. Vertically connect two row buses.

Connections among buses are realized by using routing cells (“R”) to interconnect them together from different mapping areas. See the example in Figure 4.18(1) for two-level parallel routing, we used a *step-wise* method to connect the same bus name together with routing cells. See the example from Figure 4.18(2) for three levels parallel routing.

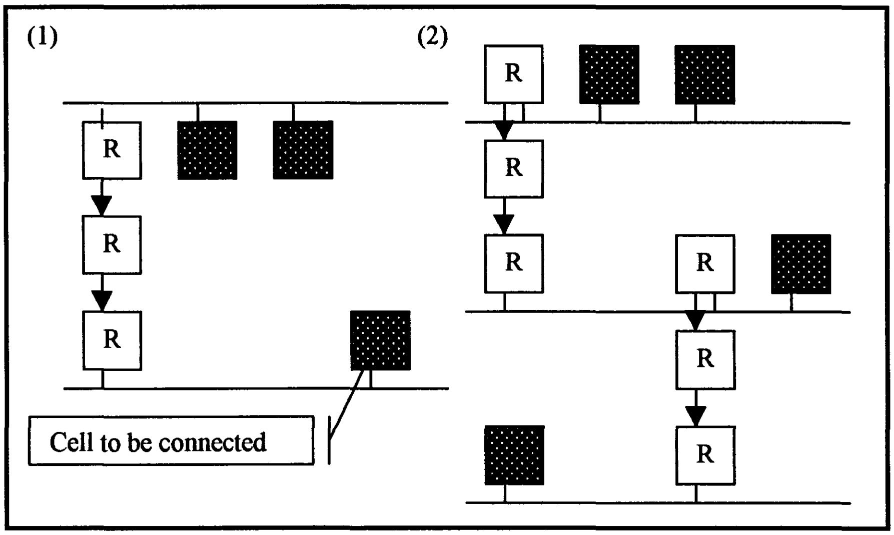


Figure 4.18: Parallel wire

There are three types of routing cells to be used for PW method, which are (in Figure 4.19):

(1) “$\boldsymbol{R_c}$” is as an interconnection cell and used between two routing cells (“$\boldsymbol{R_{in}}$” and “$\boldsymbol{R_{out}}$”) and its input and output can be assigned to “$\boldsymbol{B}$”.

(2) “$\boldsymbol{R_{in}}$” is a NOT cell type and used for a signal from a local bus. Its input comes from a local bus (Top, Bottom, left and right) and the output goes to the next cell “$\boldsymbol{R_c}$” and it is assigned to “$\boldsymbol{B}$”.

(3) "***R***" is a NOT cell type and used for a signal out from the cell to a local bus. Its input is from "$\boldsymbol{R_c}$" and assigned to "***B***". The output goes to the local bus.

All three cells connection is presented in Figure 4.19.

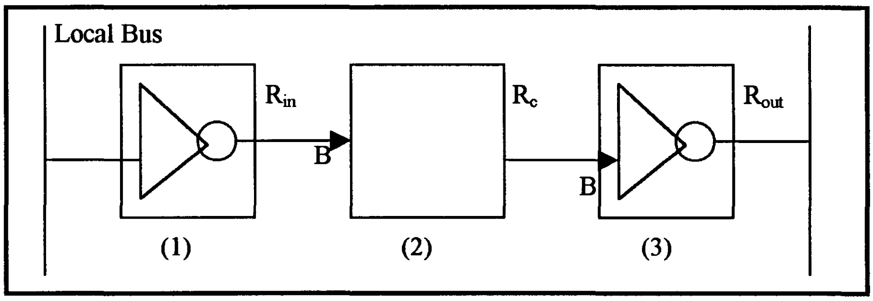


Figure 4.19: Routing cells connection

4.2.4.2 Cross Wire (CW)

The CW method is used to connect row and column buses when these buses have been assigned to the same variable. CW is realized by using a routing cell ("R") to join both horizontal and vertical buses. CW can be used when only a *cross block* is free, otherwise, both horizontal and vertical buses cannot be connected. See examples in Figure 4.20.

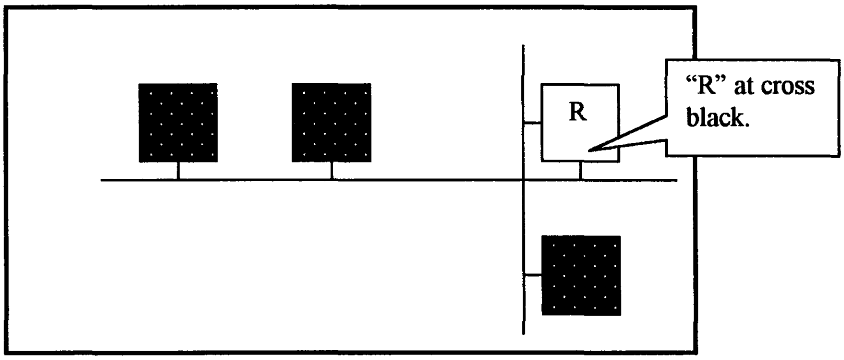


Figure 4.20: Cross Wire

There is one routing cell (R) that can be used for CW to connect two buses, both input and output signals of "R" cell connect to local buses.

4.2.4.3 The Basic Rules of the Routing Algorithm

The routing algorithm is presented in the following steps:

Step1: Find a variable which is incompletely routed after the bus assignment stage: start from a bus which has the largest number of cells connected to it, then find the next bus which has the same variable.

Step2: Routing type is chosen as follows:

If both buses are parallel then go to ***Step3*** to do PW.

If both buses are cross then go to ***Step4*** to do CW.

Step3: PW routing of a given variable: (1) find numbers of levels between the two buses (horizontal or vertical) which have the same variable. (2) Find free cells between the two buses to the left of the Root node first, and then search to the right if it is not found in the left. (3) if it found free cells from the inside of the mapping array in the left or right, it uses $\boldsymbol{R_{in}}$, $\boldsymbol{R_c}$ and $\boldsymbol{R_{out}}$ cells to connect the two buses. (4) If it cannot find free space from inside of the mapping array between the two buses then check outside of the mapping array for free space.

Step4: CW routing for a given variable: (1) if cross block is free, using "R" (both input and output are connected to a local bus) to connect both horizontal and vertical buses. (2) If cross block is not free, this variable cannot be connected.

The principle of our algorithm is to use space inside the mapping array to connect variable, if space is available. If we cannot find free space from inside, then we use outside cells of mapping array to make possible connections for buses with the same variable.

4.2.4.4 Routing Example

The routing example in Figure 4.21 (continue from Figure 4.17) demonstrates our routing algorithm. The explanation in the following is to show steps, in order, how to connect variable “d”, “f” and “g”:

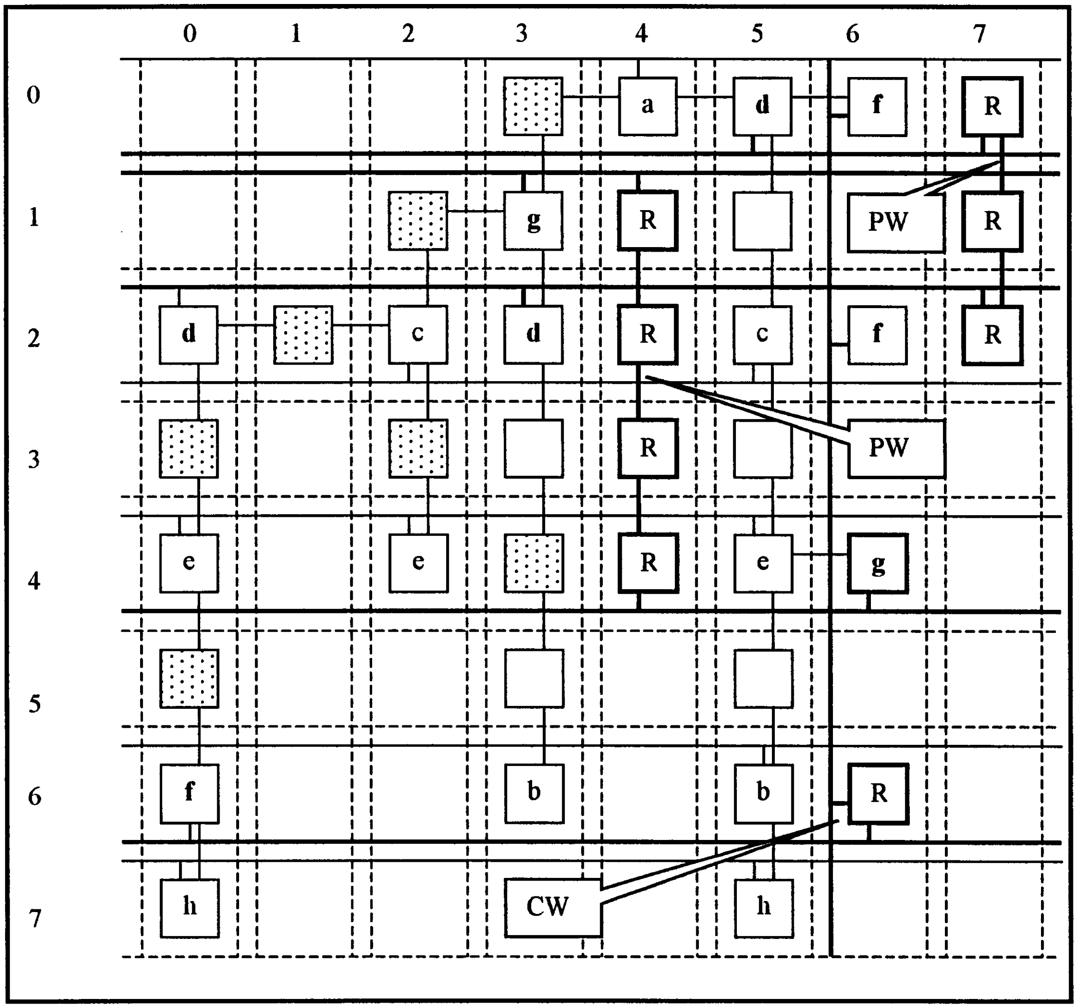


Figure 21: Routing example

Step1: Find that Variable "d" is the variable with the largest number of cells connected to it, which is located at the top bus of row (R2) and the bottom bus of row "R0". Both "d" buses are in rows so they can be connected by the PW method.

Step2: Find that Variable "f" is the variable with the largest number of cells connected to it, which is located at the left bus of column (C6) and the bottom bus of row "R6". Because two "d" buses are crossed, they can be connected by the CW method.

Step3: Find that variable "g" is at the end from the Top bus of row (R1) and from the bottom bus of row "R4". Both "g" buses are in rows so they can also be connected by the PW method.

CHAPTER 5
COMPARISON AND RESULTS EVALUATION

In this Chapter, the Modified Squashed Binary Tree (MSBT) is illustrated and compared with our algorithm. We choose MSBT to be a main comparison with the results due to of the following reasons:

1. Similarly to our approach, the MSBT is based on the Cellular-type architecture FPGAs and the layout results are targeted on the ATMEL architecture.
2. MSBT used the grouping algorithm presented in this thesis for its placement.
3. MSBT used the same logic cells from ATMEL 6000 cell states that we selected.

In order to compare, a brief introduction of the Modified Squashed Binary Tree (MSBT) will also be presented. All testing results are focused on the ATMEL 6000 architecture for both our technology placement algorithm and the MSBT algorithm, and the results are presented in the form of tables. Furthermore, we also compare our placement and routing results with the layout results created with ATMEL tools in [12].

5.1 Comparison

5.1.1 Comparison with Modified Squashed Binary Tree

In this section, we introduce a method, which is called the Modified Squashed Binary Tree (MSBT). The MSBT is formed by projecting nodes of the

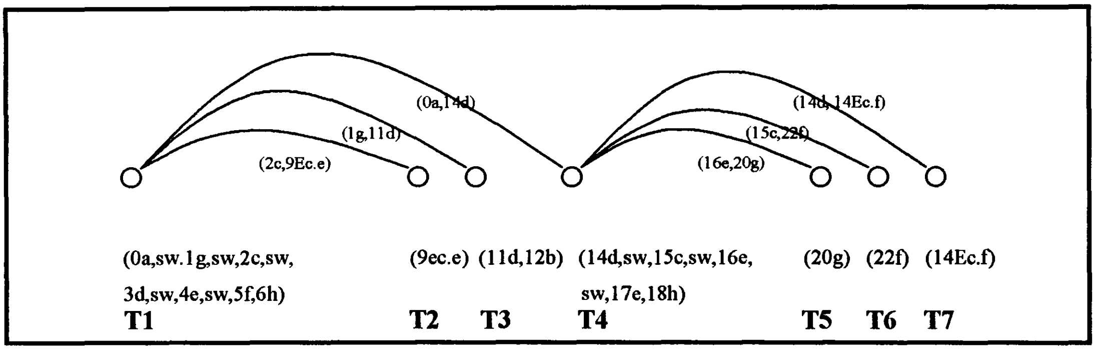


Figure 5.1: Modified Squashed Binary Tree

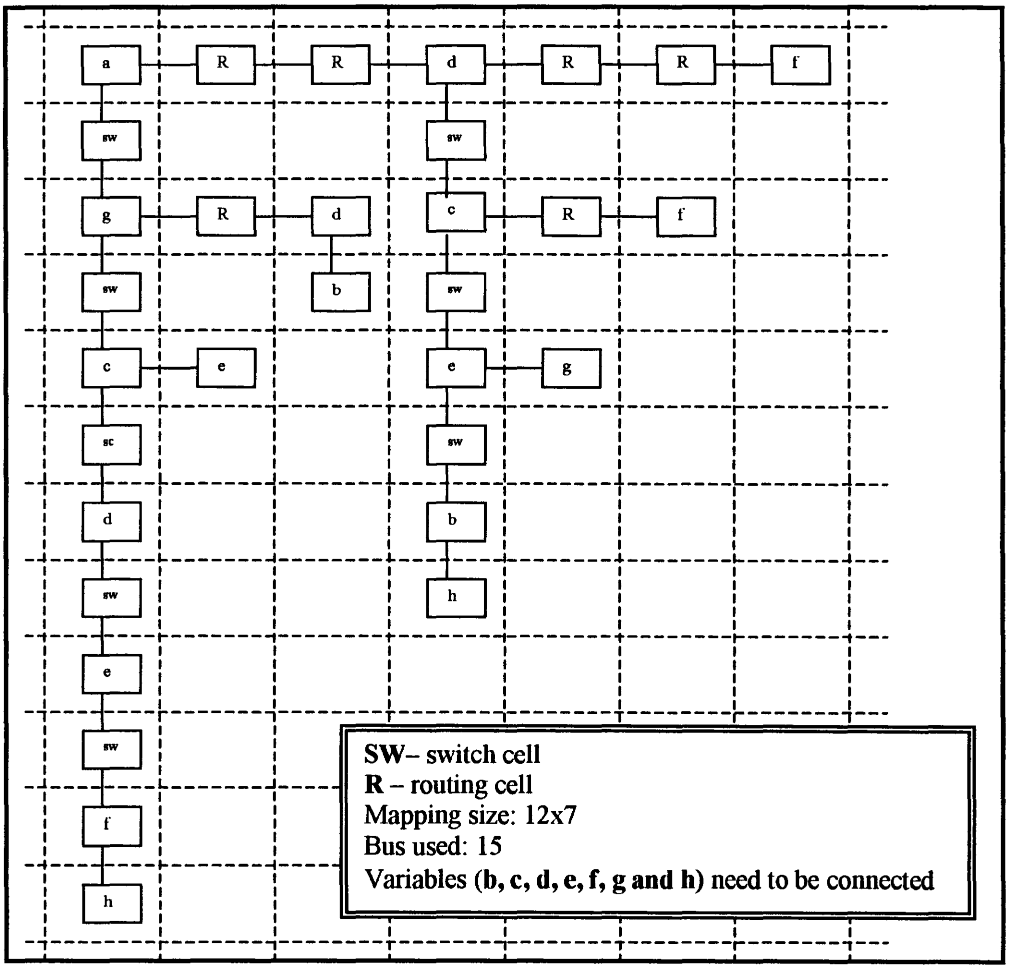


Figure 5.2: MSBT placement example

binary tree onto its leaves in the depth-first manner [8].

In general, the modified squashed binary tree $T_b(V_b,E_b)$ consists of the set of vertices V_b and the set of directed E_b. Where $V_b = [v_b|v_b$ represents the vertices of the tree $T_b(V,E)$, projected onto the same leaf] and $E_b = [e_b|e_b$ represents the directed edge from v_{bi} to v_{bj} if any of the vertices of the tree T(V,E) which were collapsed to vertex v_{bi} is connected to any of the vertices of the tree T(V,E) which are collapsed to vertex v_{bj}], here the directed edge represents the direction of input signal flow [8,7].

The example for MSBT is given in Figure 5.1. This example uses the same input as our example that is shown on the page 70 of this thesis.

The MSBT is mapped into the cellular array by following the order of placing nodes group (example as notes group of T_1 to T_7 in Figure 5.1) from left to right. Therefore, T_1 is mapped into the first column, and then T_2 is mapped into the second column and so on until the placement is finished. In addition, the routing cell (R) is inserted between the first column and the third column when they are connected. See the example in Figure 5.2, which shows the mapping of the MSBT from Figure 5.1 to the cellular array.

The result of the MSBT placement, which is in Figure 5.2, has two major problems:

1. MSBT uses many local buses for the assignment of the same variable name. The example of Figure 5.2 has totally spent 16 local buses for 8 variables bus assignment.

2. The size of the mapping matrix is 12x7, and it could be extended to very large size if the MSBT has more branches, because the MSBT needs to use many routing cells (R) to connect nodes between columns.

The above indicated problems can give difficulty to routing because large numbers of local buses and cells are needed by the bus connections with the same variable name. On the other hand, the result of our placement is presented in Figure 5.3, which is the same in Figure 4.16 from Chapter 4.

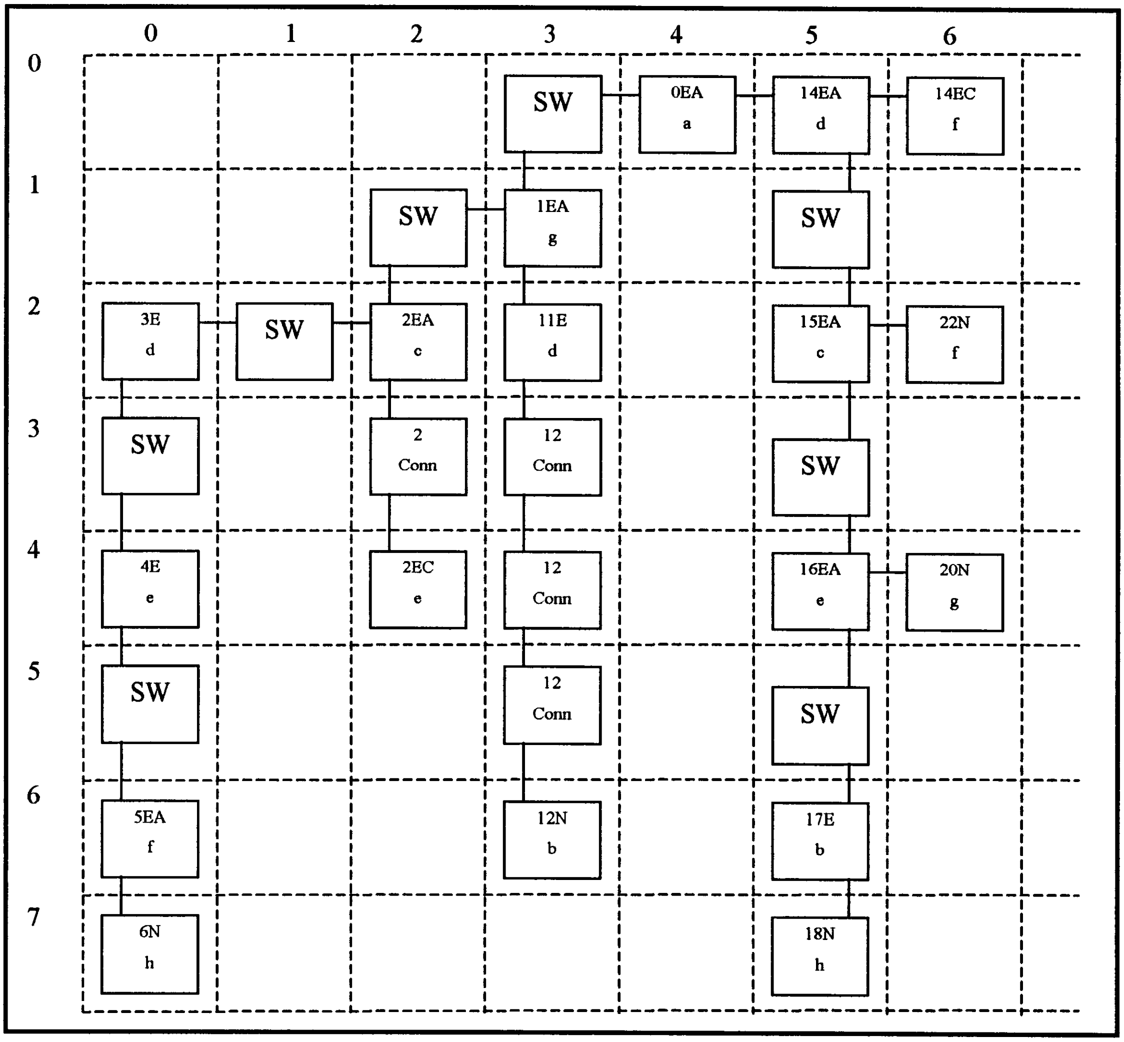


Figure 5.3: CBT placement example

Comparing with MSBT, two problems indicated in the above from MSBT have been greatly improved in our placement algorithm, which are: a total of 11 local buses are used for 8 input variables and the 8x7 array is spent.

Both results of placement are obtained from the same testing input file "*Example.blif*". The results of final layouts for MSBT and our approaches are shown in Figure 5.2 and Figure 5.3 respectively. The comparison is given in Table5.1.

COMPARISONS	ALGORITHMS	
	MSBT	Our
Number of Variable disconnected	7	3
Mapping size	12x7	8x7
Buses used	15	11

Table 5.1: Comparison with MSBT

The comparison column lists items to be compared and the algorithm column lists all results from both placements. See more comparisons with the MSBT results in the next section.

5.1.2 Comparison with ATMEL Place&Route Tools

In this section, the difference between the results of our approach and the results the ATMEL tools, which placement and routing methods are based on *Simulated Annealing* [41] as cost driven, for the placement and routing are illustrated. The input files are the same for both methods, which are from the MCNC benchmark. They are also represented by the PRMT, which was listed in the Table 5.2 (MCNC column). The results of the final layouts created by the ATMEL tools are from [12]. The comparison results between both methods and also including the results of the MSBT are presented in Table 5.2.

MCNC	MSBT				ATMEL TOOLS[12]				OUR ALGORITHM			
	LB	LC	Cn	As	LB	LC	Cn	As	LB	LC	Cn	As
5x10	19	15	6	27	46	17	23	90	8	15	3	25
misex54	19	18	13	35	58	24	27	180	7	23	1	30
f5	10	8	6	12	28	11	12	45	6	8	3	12

Table 5.2: Our algorithm Vs. ATMEL tools and MSBT

Where:

- **LB** column is the total number of buses is used.
- **LC** column is the total number of logic blocks that are used for implementing logic.
- **Cn** column is the total number of routing cells, which is used for interconnections during the mapping.
- **As** column represents the rectangular area, which is occupied by the core of the design because the ATMEL tools perform bus assignment in an inefficient way, only the core area with mapping design is compared [3].

In Table 5.2, comparing between OUR ALGORITHM and the ATMEL TOOLS is presented using the MCNC benchmarks and MSBT as well. As it can be seen the results that OUR ALGORITHM gives much more compact layouts than the ATMEL tools and also better compact layouts than MSBT. The number of local buses, logic blocks used for routing and the rectangular area are much smaller than ATMEL tools and also smaller than MSBT.

5.2 Results Evaluation

5.2.1 Results

In the above comparisons, an example has been completely given to compare the results of placement from the MSBT and the method presented in the chapter 4. In this section, more results of examples from both our algorithm and MSBT algorithms will be presented. The results from the comparison of both algorithms are given in Table 5.3. The definition of columns from Table 5.3 is as follows:

I column: The number of input variables.

IN column: The number of the input nodes from the blif format (as our input file).

Gp. column:

L: The number of the logic blocks in CBT after grouping.

C: The number of connection type cells, which are included

-Switch Cell -Extend Cell

MSBT and Our Algorithm columns:

R: The number of routing cells (MSBT) and connection cells (Ours) used.

T: The total number of cells used after the placement, which include L and C from Gp, and R.

As: The matrix size of the smallest rectangle enclosing the mapped circuit after placement is finished.

TB: The numbers of total local buses used during bus assignment after placement is done.

MCNC	I	IN	GP.		MSBT				OUR ALGORITHM			
			L	C	R	T	As	TB	R	T	As	TB
5x1	7	29	21	5	2	28	48(8x6)	12	3	29	28(4x7)	12
5x10	7	17	15	0	0	15	27(9x3)	11	3	18	25(5x5)	8
con11	7	18	15	7	1	23	36(9x4)	12	0	22	42(7x6)	10
con12	7	12	9	4	0	13	18(9x2)	10	0	13	25(5x5)	8
misex21	6	25	17	3	1	20	54(6x9)	13	16	36	64(8x8)	6
misex22	6	20	16	1	0	17	24(3x8)	11	1	18	24(4x6)	8
misex23	6	19	13	1	0	14	27(9x3)	10	9	23	32(8x4)	7
misex48	6	25	16	3	0	17	45(5x9)	12	6	25	64(8x8)	11
misex49	6	20	16	1	0	17	28(4x7)	10	0	17	30(5x6)	10
misex50	6	21	14	1	1	16	32(4x8)	12	8	24	24(6x4)	8
misex52	6	20	16	7	1	24	45(5x9)	11	8	31	49(7x7)	12
misex53	6	15	10	2	1	13	24(4x6)	10	1	13	24(6x4)	7
misex54	6	24	18	4	1	23	35(5x7)	11	1	23	21(7x3)	7
misex56	6	21	14	2	1	16	32(4x8)	11	8	24	24(6x4)	8
misex57	6	20	17	3	1	21	27(3x9)	12	0	20	36(4x9)	10
sao21	10	551	337	105	151	595	1792 (16x112)					
sao22	10	779	408	108	175	691	2025 (15x135)					
sao23	10	846	475	125	205	805	2422 (14x173)					
majority	5	20	15	5	0	20	32(4x8)	9	6	26	42(6x7)	6

Table 5.3: MSBT Vs. our algorithm

Overall, the above results from Table 5.3 has shown two major parts in which our approach can give better results than MSBT such as reducing the total number of buses and reducing the total number of eight-cell arrays (see Figure 2.1) of the AMTEL architecture. Also, the area of our placement could be smaller than MSBT if a CBT has more branches. The details of description for the result comparisons from both algorithms will be described in the next section.

5.2.2 Results Description

In this section, we describe and discuss the placement results of both our algorithm and MSBT algorithm in more details by using Table 5.3, which include the comparisons of number of buses used, number of eight-cell arrays used and the mapping size in the mapping areas of ATMEL architecture.

***Reducing bus numbers*:** The comparisons are with TB columns between both algorithms in Table 5.3, most of the examples from MSBT used more buses than ours. The "misex21" in Table 5.3 is a very good example to show the difference between both results. MSBT shows that 13 buses are used to connect the variables of the cells after placement with incomplete routing. The algorithm presented in this thesis used only 6 buses with complete routing when placement was done. The reason is that MSBT algorithm minimizes the size of placement in the mapping area. However, the results are not taken into the consideration about reducing the number of buses for the variables routing. In fact, MSBT can do largest areas' mapping in the CA-Type cell array. On the other hand, our placement not only considered to place CBT onto a smaller cells array, but it also consider the variable routing by lining up the same variable names using leaf cell reposition methods. Therefore, our placement results for variables routing after CBT mapping for small examples and better than MSBT.

***Reducing the total number of eight-cell arrays of the AMTEL*:** In this case, we have known the symmetrical array of ATMEL 6000 architecture from the previous chapter (see Figure 2.1 and 2.2), the express buses using "repeaters" connect every eight-cell array. We follow example "misex21" to compare the "As" column of both results from Table 5.3. The results show that MSBT can have smaller placing size than our algorithm (MSBT is 6x9 and ours is 8x8). Therefore, the MSBT vertically takes two eight-cell arrays of ATMEL 6000 architecture to complete placement, which means that the MSBT placement algorithm has to use "Repeater" to associate with a local/express buses to connect cells between two eight-cell arrays which can result in increasing delays for the routing. However, our placement algorithm just used a single eight-cell array, which means that our algorithm can place more cells into an eight-cell array than MSBT.

Size of mapping area: For this issue, the results of "As" columns from Table 5.3 show that MSBT gives better results for most files than ours, it also can run any size examples. Unlike MSBT, our mapping is limited to run the larger examples. However, in the some cases, if an input tree is reformed to take the advantage of our algorithm, For example, if we reformed a CBT to add some branches (especially, if branches only have a few nodes), our placement algorithm can get better results than MSBT. For this improvement, we give some physical layout printouts in the Appendix, which examples are manually generated, and they show the difference between both algorithms. See the results of comparison for both algorithms from Table 5.4. The input file column lists the manual examples, which are based on the blif format of PRMT and the definition of other columns are as same as Table 5.3 (see columns' definition on page 87). We compare the "As" column of both algorithms from Table 5.4 and find that MSBT used more mapping space than ours.

INPUT FILES	I	IN	GP.		MSBT				OUR ALGORITHM			
			L	C	R	T	As	TB	R	T	As	TB
testrf1.blif	6	20	16	0	4	20	36(6x6)	11	1	17	18(3x6)	9
testrf2.blif	7	18	13	7	4	24	48(8x6)	16	4	24	35(5x7)	10
testrf3.blif	6	15	11	4	4	19	42(7x6)	11	2	17	32(4x8)	8
testrf4.blif	6	18	13	4	6	23	35(5x7)	12	1	18	24(3x8)	9

Table 5.4: MSBT Vs. our algorithm with manual examples

5.2.3 Limitations

As introduced and explained earlier in this thesis, our approach is to restructure the net-list of the cell binary tree (CBT), and place it on the ATMEL 6000 FPGA. Therefore, our methodology is suitable only for the CA-Type cell array. The following are the major limitations of our placement and routing algorithms, which may limit its usefulness and applicability in some cases.

Our placement algorithm depends on input files: For this limitation, we give some examples in Table 5.3 such as sao21 (10 input variables with 551 nodes), sao22 (10 variables with 779 nodes) and aso23 (10 variables with 846 nodes). Unlike MSBT, our automatic placement cannot run those files because those files have many cells with only few input variables. However, if a larger file has the 70 cells with 13 input variables, it could be run through. Of course, this is a major problem with our mapping application because our placement used lining up method to match the same variable into the same row or column, so this way can reduce the number of the interconnected buses but it cannot run for a larger file that.

Routing is limited: In the CW method, the two local buses with the same variable name cannot be connected when the cell of Cross-Block is not freed (taken by another variable). This cell is named a *"Dead-Block"*, see example (a) in Figure 5.5. For this issue, two local buses with the same variable name can be connected with PW and CW combination. See examples in Figure 5.5.

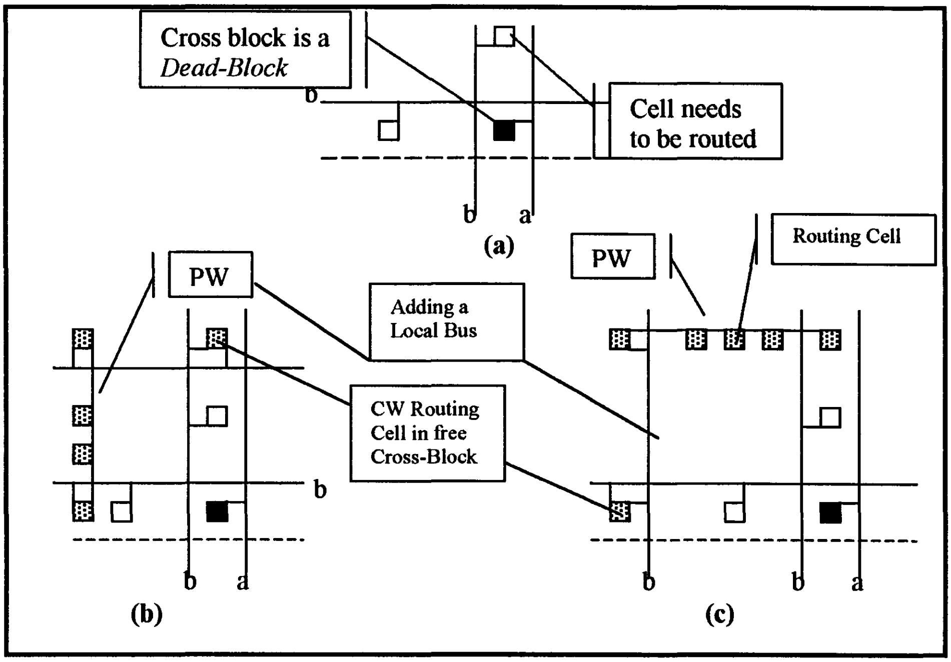


Figure 5.4: PW and CW combination routing

Figure 5.5 shows two possible ways for routing variables, which connect two buses from the different directions for the same variable, where;

(b) Shows that PW connects CW in horizontal way.

(c) Shows that PW connects CW in Vertical way.

In both ways, the free Cross-Block needs to be found in the short path. If the free Cross-Block is found in the row, the same variable name should be interconnected as (c) in Figure 5.5, otherwise, the same variable name should be interconnected as in (b). Also, a local bus needs to be added to interconnect both PW and CW for the variable routing.

These limitations can be overcome with the right solutions. For example, the architecture of CA-type FPGAs can be changed with multiple local buses in any given direction.

5.3 Summary

In general, the objectives for MSBT and our algorithm are different. MSBT algorithm focuses on placing the largest size of logic into the architecture of CA-type FPGA, which is ATMEL 6000 architecture. However, the MSBT doesn't consider the number of the local buses to be used for the routing. Our placement algorithm is also targeted on the ATMEL 6000 architecture but we started by focusing on the eight-cell array for the placement, and then extended to larger array. Therefore, some larger files limit our placement algorithm. For example, a larger file with the small number of the input variables cannot be successfully run by our placement. Another point is, unlike MSBT, our placement results take into consideration the routing by lining up the same variable name and leaf cell reposition methods. Therefore, in this case, our results can be better than MSBT in two major areas: smaller number of buses and smaller number of eight-cell arrays, but only for small examples.

CHAPTER 6
CONCLUSIONS

In this thesis, a new technology mapping system is proposed in following four major stages: restructuring the PMRT to the CBT for the mapping combinatorial circuits onto CA-Type FPGAs, the CBT placement, bus assignment and routing. We presented those algorithms, which are targeted at combinatorial logic and the ATMEL 6000 architecture FPGAs. The mapping used to line up method to reduce the number of local buses, which is a very important advantage for the local connectivity in this approach.

A new technique based on the concept of Cell Binary Tree (CBT) for mapping combination circuits into ATMEL 6000 Architecture FPGAs is presented in this thesis. Cell Binary Tree (CBT) is a net-list representation of combinational circuits. For each node of CBT there is a distinguished variable associated with it, the node itself represents a certain logic function, which is selected according to target FPGA architecture. As we pointed out, the significance of the placement method is to line up the same variable name into the same row or column in the mapping array. This way the interconnections among buses can be reduced. The algorithm further reduces the area by the process of leaf cell repositioning. Therefore, the main focus of our placement method is to reduce the number of local buses and the size of the mapping array; these in turn can make routing much easier. The bus assignment is a very important step to reduce the overall area. The bus assignment method is based on the number of times a variable appears in a row or column. After the CBT placement, there are two kinds of routing, PW and CW, considered and presented to connect buses with the same variable name in the interconnection path after placement is done.

For future works, we should consider and improve the problem of limitations in our mapping algorithm, which can allow running any larger without considering the number of input variables. The minimization of mapping area needs to be obtained, when the placement is improved. Also, the routing heuristic needs to be improved. Another further improvement could be achieved by global connectivity to interconnect eight-cell arrays with "express" buses on ATMEL architecture FPGAs.

In summary, my contribution to this thesis is that I investigated, proposed and implemented a heuristic placement and routing approach for ATMEL 6000 architecture FPGAs. The important parts of the methods consist of logic grouping, placement, bus assignment and routing with considerations. The results compared with MSBT and AMTEL Tools, we used the smaller number of the local buses for variable routing.

BIBLIOGRAPHY

[1] L. Wu, and M. A. Perkowski, "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays," In H. Gruenbacher and R. Hartenstein (eds.), *LNCS, No. 705, Springier Verlag, pp. 78-87*, 1993.

[2] ATMEL Corporation CMOS Integrated Circuit Data Book, 1993:94. 2125 O'Neil Drive, San Jose, CA, 95131.

[3] N. Ramineni, and M. Chrzanowska-Jeske, "A Routing-Driven Mapping For Cellular-Architecture FPGA's", *36th Midwest Symposium On Circuits & Systems pp. 308-311*, Aug 1993.

[4] M. Chrzanowska-Jeske, "Architecture and Synthesis Issues in FPGAs," *Proc. of Northcon, pp. 102-105*, October 1993.

[5] N. B. Bhat and D. D. Hill, "Routable Technology Mapping for FPGA's," *1992 ACM First International Workshop on FPGAs, pp. 143-148*, 1992.

[6] A. Sarabi, N. Song, M. Chrzanowska-Jeske, M. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-dimensional Logic Arrays", *Proc. of 31st DAC, pp. 321-326*, June 1994.

[7] A. K. Dasari, N. Song, M. Chrzanowska-Jeske, "Layout Driven Factorization and Fitting for Cellular Architecture FPGAs", *Proc. Of Northcon, pp. 106-111*, 1993.

[8] N. Woo, "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility", *28th ACM/IEEE Design Automation Conference, pp. 248-251*, 1991.

[9] M. A. Perkowski, A. Sarabi, and F. R. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg, Germany, pp. 27-32*, September 1993.

[10] R. C. Minnick, "Cutpoint Cellular Logic," *IEEE Trans. on Electron. Comput. EC-13, pp. 685-698*, 1964.

[11] E. A. Walkup, and S. Hauck, “Routing-Directed Placement for the TRIPTYCH FPGA”, *Department of Computer Science and Engineering, FR-35, University of Washington, pp 33-38*, 1992.

[12] N. Ramineni, *Thesis:* “Restructuring Approach to Mapping Problem in Cellular-Architecture FPGAs”, *Department of Electrical and Computer Engineering, Portland State University*, 1995

[13] K. Bartlett, W. Cohen, A. J. De Geus, and G. D. Hachtel. “Synthesis of Multilevel Logic under Timing Constraints”, *IEEE Transactions on computer-Aided Design of Integrated Circuits, CAD-5(4): pp. 582-595*, October 1986.

[14] D. Bostick, G. D. Hachtel, and R. Jacoby, “The Boulder Optimal Logic Design System”, *In Proceedings of the International Conference on Computer-Aided Design, pp. 62-65*, November 1987.

[15] R. Brayton and R. Rudell, “MIS: A Multiple-Level Logic Optimization System”, *IEEE Transactions on computer-Aided Design of Integrated Circuits, CAD-6(6): pp 1062-1081*, November 1987.

[16] R. K. Brayton and C. McMullen, “The Decomposition and Factorization of Boolean Expressions”, *In Proceedings of the International Symposium on Circuits and Systems, pp. 49-54, Rome*, May 1982.

[17] S.B. Akers. “Binary Decision Diagrams”, *IEEE Transactions on Computers, C-27(6):pp.509-516*, June 1978.

[18] R. Bryant. “Graph-Based Algorithms for Boolean Function Manipulation”, *IEEE Transactions on Computers, C-35(8):pp.677-691*, August 1986.

[19] C. Y. Lee. “Representation of Switching Circuits by Binary-Decision Programs”, *Bell Systems Technical Journal, 38(4):pp.985-999*, June 1959.

[20] U. Kebschull, E. Schubert, “Multilevel Logic Synthesis Based on Functional Decision Diagrams”, *Proc. IEEE European Design Automation Conference, pp.43-47*, 1992.

[21] I. Schaefer, M. A. Perkowski, H.Wu, “Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions”, *Proc. IFIP WG 105 Workshop on*

Applications of the Reed-Muller Expansion in Circuit Design, Hamburg, Germany, pp. 42-51, Sept. 1993.

[22] N. Song and M. A. Perkowski. "A New Design Methodology for Two-Dimensional Logic Arrays". *Proc. Of IWLS, Tahoe City, CA, pp.132-137,* May 1993.

[23] K. K. Maitra. "Cascaded Switching Networks of Two-Input Flexible Cells". *IRE Trans. Electron. Comput., pp. 136-143*, 1962.

[24] Robert Francis, Jonathan Rose, and Zvonko Vranesic, "Chortle-Crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. 28th ACM/IEEE Design Automation Conf.*, San Francisco, CA, *pp. 227-233,* June 1991.

[25] Rajeev Murgai, Narendra Shenoy and Robert K. Brayton, " Improved Logic Synthesis Algorithms for Table Look Up Architectures", *ICCAD 1991*, Santa Clara, CA, *pp. 564-567,* Nov. 1991.

[26] Wei Wan and Marek A Perkowski, "TRADE: A Lookup Table FPGA Mapper Based on a Generalized Boolean Decomposition", *ICCAD, Santa Clara, CA, pp. 362-367,* Nov. 1992.

[27] Robert Francis, Jonathan Rose, and Kevin Chung, "Chortle: A Technology Mapping for Lookup Table-Based Field Programmable Gate Array," *Proc. 27th ACM/IEEE Design Automation Conf.*, San Francisco, *pp. 613-619*, CA, June 1991.

[28] Ebeling, Carl; Mcmurchie, Larry; Hauck, Scott A; Burns, Steven, "Placement and Routing Tools for the Triptych FPGA", *IEEE Trans. On Very Large Scale Integration (VLSI) Systems v3 n4, pp. 473-482, Dec. 1995.*

[29] Hoffman, Stephen C, "Automatic Gate Allocation Placement and Routing", *Proc of a Symp on Computer-Aided Design of Digital Electron Circuits and Syst, Amsterdam, Neth, pp. 645-642, Nov 1978-1979*

[30] Opitz, Frank Voelker, "Aulis-Automatic Placement and Routing of Date Array" *Circuit Theory and Design, Proceedings of the 6th European Conference, pp. 331-337,* 1983.

[31] Kelly, John, "Placement and Routing Techniques for Gate Arrays", *New Electron v18 n5, p47-48,* March 5, 1985.

[32] Lauther. Ulrich, "Overview of Placement and Routing Techniques", *IEEE Electron Devices Soc, New York, NY, pp. 615-620,* 1987.

[33] Chrzanowska-Jeske, Malgorzata, "Regular Repesentation for Mapping to Fine-Grain, Locally-Connected FPGAs", *IEEE International Symposium on Circuits and Systems, pp. 2749-2752,* ISCAS, June 9-12, 1997.

[34] Swartz, William, "New Algorithms for the Placement and Routing of Macro-Cells*", IEEE International Conference on Computer-Aided Design, ICCAD-90, pp. 336-339,* Nov 11-15, 1990

[35] Brown, Stephen; Rose, Jonathan and Vranesic Zvonko, "A Detailed Router for Field-Programmable Gate Arrays", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 11, pp. 620-628*, May 1992.

[36] Wu, Yu-Liang and Marek-Sadowska, Malgorzata, "Routing for Array-Type FPGA's", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 5, pp. 506-516*, May 1997.

[37] Alexander, Michael and Robins Gabriel, "New Performance-Driven FPGA Routing Algorithms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 12, pp. 1505-1515,* December 1996.

[38] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Provably Good Performance-Driven Global Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 11, pp. 739-752,* 1992.

[39] J. Cong, K. S. Leung, and D. Zhou, "Performance-Driven Interconnection Design Based on Distributed RC Delay Model", *in Proc. ACM/IEEE Design Automation Conf., Dallas, TX, pp. 314-318,* June 1993.

[40] E. W. Dijkstra, A Note on Two Problems in Connection with Graphs, Numerische Mathematik, *Vol. 1, pp. 269-271,* 1959.

[41] S. Kirkpatrick, C. Gelatt and M. Vecchi, “Optimization by Simulated Annealing”, IBM Computer Science/Engineering Technology Watson Res. Center, Yorktown Heights, NY, Tech. Rep., 1982.

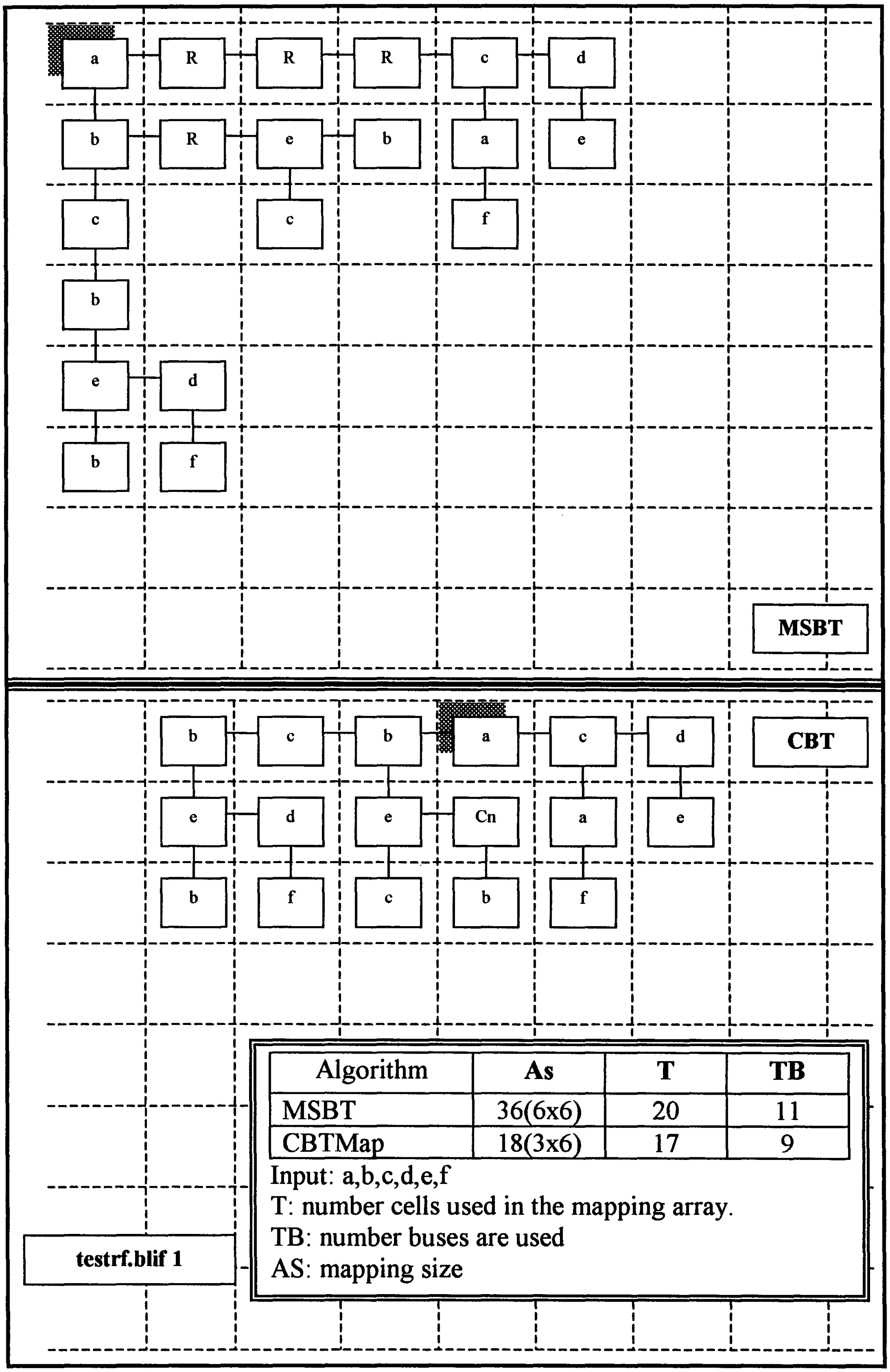
[42] T. Gao, P. M. Vaidya, and C. L. Liu, “A New Performance Driven Placement Algorithm”, *in Proc. ICCAD, pp. 44-47*, 1991.

[43] Chrzanowska-Jeske, Malgorzata and Xu, Yang, “Optimized Embedding of an Incomplete Binary Tree in a Two-Dimensional Array of Programmable Logic Blocks”, *39th IEEE Midwest Symposium on Circuits and Systems Proceedings, pp. 353-356,* 1996.

[44] F. T. Leighton, “Introduction to Parallel Algorithms and Architectures” *Morgon Kaufmann Publishers, Inc.*, San Mateo, CA, 1992.

APPENDIX

Comparison of our approach Vs MSBT



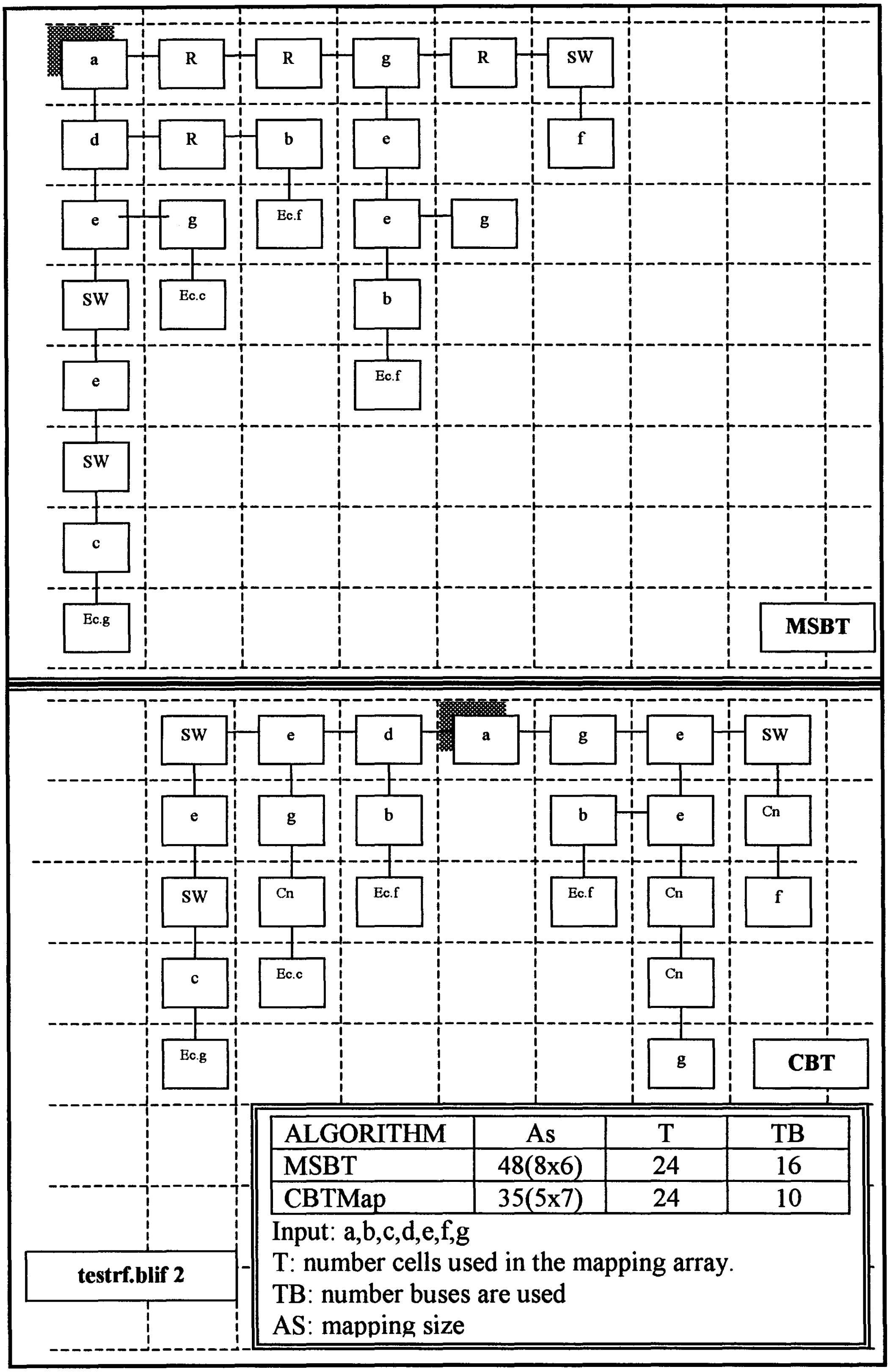
Algorithm	**As**	**T**	**TB**
MSBT	36(6x6)	20	11
CBTMap	18(3x6)	17	9

Input: a,b,c,d,e,f

T: number cells used in the mapping array.

TB: number buses are used

AS: mapping size



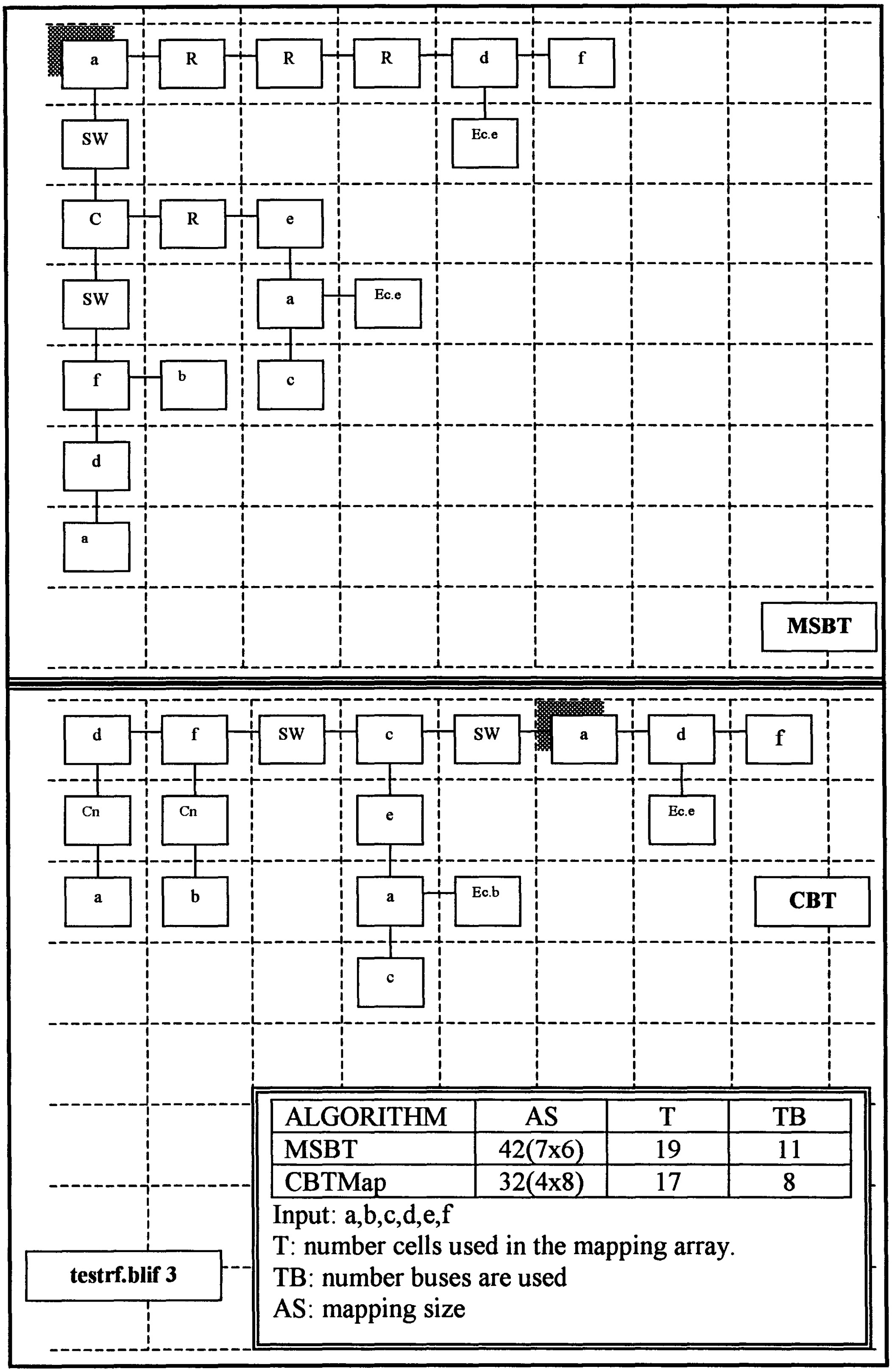
ALGORITHM	As	T	TB
MSBT	48(8x6)	24	16
CBTMap	35(5x7)	24	10

Input: a,b,c,d,e,f,g

T: number cells used in the mapping array.

TB: number buses are used

AS: mapping size



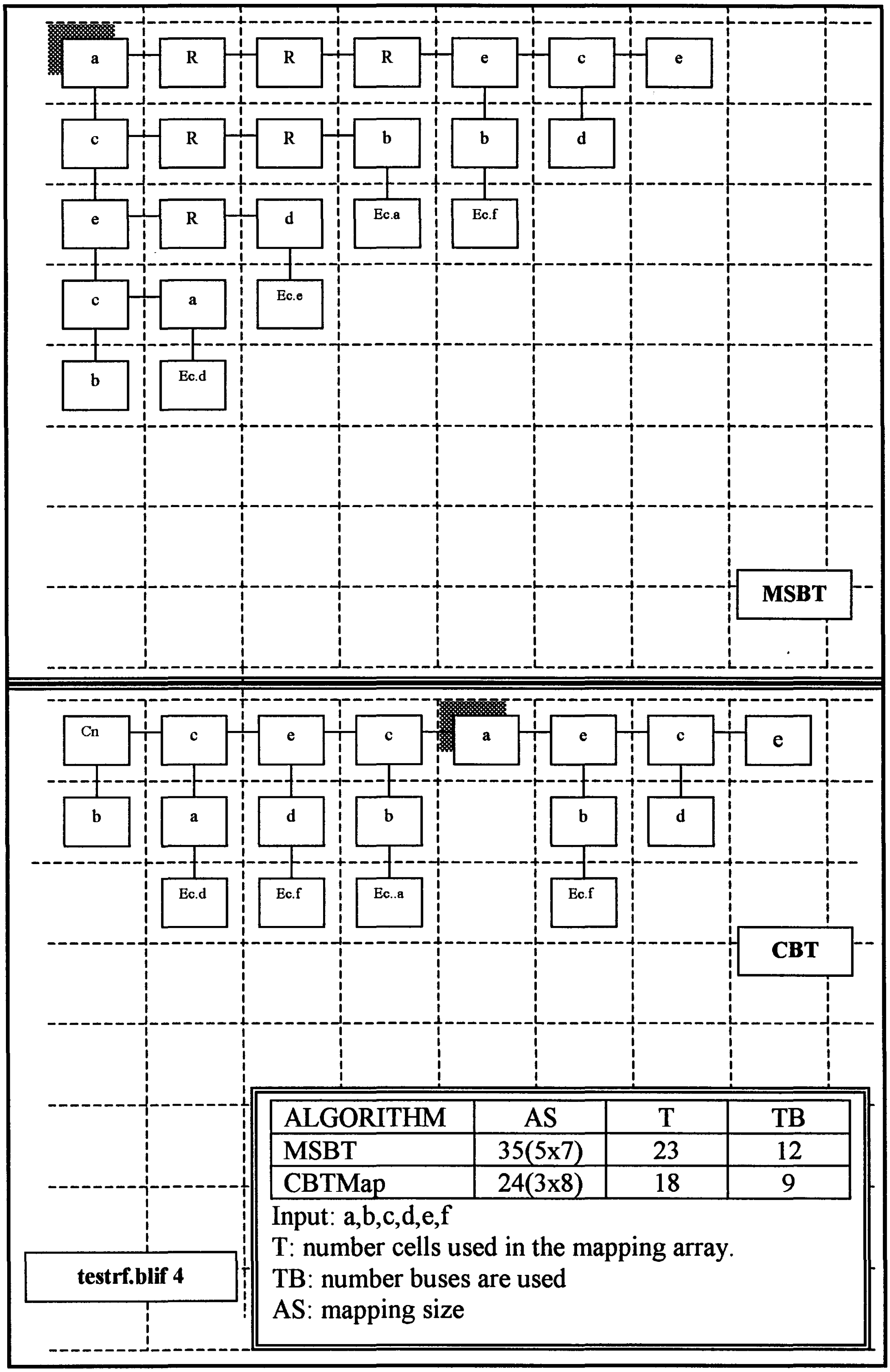
ALGORITHM	AS	T	TB
MSBT	42(7x6)	19	11
CBTMap	32(4x8)	17	8

Input: a,b,c,d,e,f

T: number cells used in the mapping array.

TB: number buses are used

AS: mapping size



ALGORITHM	AS	T	TB
MSBT	35(5x7)	23	12
CBTMap	24(3x8)	18	9

Input: a,b,c,d,e,f

T: number cells used in the mapping array.

TB: number buses are used

AS: mapping size