UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FELIPE DE SOUZA MARQUES

Technology Mapping for Virtual Libraries Based on Cells with Minimal Transistor Stacks

Thesis presented in partial fulfillment of the requirements for the degree of Doctor of Computer Science

Prof. Dr. André Inácio Reis
Advisor

Porto Alegre, March 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Marques, Felipe de Souza

Technology Mapping for Virtual Libraries Based on Cells with Minimal Transistor Stacks / Felipe de Souza Marques – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2008. Advisor: André Inácio Reis.

1. Mapeamento Tecnológico 2 Síntese Lógica 3 Bibliotecas de Células 4 Bibliotecas Células Virtuais 5 Geradores de Células. I. Reis, André Inácio. II. Título.

ACKNOWLEDGMENTS

Primeiramente, gostaria de agradecer as pessoas que estiveram diretamente envolvidas no desenvolvimento do meu trabalho de tese. Meu orientador, prof. André Reis, com quem trabalhei desde o mestrado e sempre trouxe grandes contribuições para a realização deste trabalho. Ao prof. Renato Ribas, que apesar de não ser oficialmente co-orientator deste trabalho, atuou como tal e esteve sempre disposto a ajudar no que foi preciso. Quero estender este agradecimento ao colega Leomar Soares da Rosa Junior que também teve uma importante contribuição e que sempre esteve disposto a conversar, seja sobre trabalho, seja sobre questões pessoais. Eu gostaria de agradecer do mesmo modo a todos os colegas com quem eu trabalhei no laboratório da Nangate, em especial ao Pedro e ao Erasmo, pelos esforços para a realização de experimentos.

Também gostaria de agradecer a minha família, meus pais, Sandra e Rogério Marques, e meus irmãos Rodrigo, Diego e Letícia, pois o apoio e carinho deles foram sempre muito importantes em todas as minhas conquistas. Não poderia deixar de agradecer uma pessoa que é fundamental na minha vida, minha esposa, Milena Rota Sena Marques, por todo o companheirismo, compreensão e amor que me deram forças para vencer mais um desafio, além da paciência para agüentar todos os dias de mau humor. Agradeço aos amigos, pelas cervejas, churrascos e conversas, que certamente ajudaram a espairecer nos momentos mais estressantes do doutorado.

É importante também agradecer ao CNPq pela bolsa de doutorado, que, através do Programa Nacional de Microeletrônica, me possibilitou dedicação exclusiva a minha formação acadêmica. A CAPES, agradeço a bolsa de estágio no exterior, que me permitiu colaborar com um excelente laboratório de pesquisa na Universidade de Minnesota, nos Estados Unidos, e ter uma experiência muito relevante para minha formação. Em especial, agradeço ao prof. Sachin Sapatnekar, por ter me dado à oportunidade de trabalhar em seu grupo de pesquisa (Thank you!).

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ABC	A System for Sequential Synthesis and Verification
AIG	AND-Inverter Graph
AND	AND Boolean operator
AOI	AND-OR-Inverter Boolean operator
BDD	Binary Decision Diagram
LC	Logic Cones
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CSP	Complementary Series/Parallel
DAG	Directed Acyclic Graph
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
GND	Ground drain
HDL	Hardware Description Language
IC	Integrated Circuit
ISCAS	International Symposium on Circuits and Systems
LLWF	Library-less Wavefront
LUT	Lookup Table
NAND	Inverted AND Boolean operator
NCSP	Non-Complementary Series/Parallel
NMOS	N-type Metal Oxide Semiconductor
NOR	Inverted OR Boolean operator
NOT	Inversion Boolean operator
NP	Algorithm complexity class
OR	OR Boolean operator
OTR	Odd-level Transistor Replacement
PC	Personal Computer

PD	Maximum number of stacked transistors in the pull-down plane
PI	Primary Input
PMOS	P-type Metal Oxide Semiconductor
PO	Primary Output
POS	Product of Sums
PTL	Pass Transistor Logic
PU	Maximum number of stacked transistors in the pull-up plane
ROBDD	Reduced Ordered Binary Decision Diagram
RTL	Register Transfer Language
SIS	Sequential Interactive System
SOP	Sum of Products
SPD	Sum of PDs in a path
SPICE	General-Purpose Circuit Simulation Program
SPU	Sumo f PUs in a path
STA	Static Timing Analysis
TABA	Tool for Library Free Technology Mapping
TSBDD	Terminal-Suppressed Binary Decision Diagram
VDD	Power supply voltage
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VIRMA	Virtual technology mapping tool
VLSI	Very Large Scale Integration
XNOR	Inverted XOR Boolean operator
XOR	XOR Boolean operator

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Currently, microelectronic technologies enable high degrees of semiconductor integration. However, this integration makes the design, verification, and test challenges more difficult. The circuit design is often the first area under assault by the effects of aggressive scaling in deep-submicron technologies. Therefore, designers have adopted strict methodologies to deal with the challenge of developing high quality designs on a reasonable time. Electronic Design Automation tools play an important role, automating some of the design phases and helping the designer to find a good solution faster. One of the hardest challenges of an integrated circuit design is to meet the timing requirements. It depends on several steps of the synthesis flow. In standard cell based flows, it is directly related to the technology mapping algorithm and the cells available in the library. The performance of a cell is directly related to the transistor sizing and the cell topology. It determines the timing, power and area characteristics of a cell. Technology mapping has a major impact on the structure of the circuit, and on its delay and area characteristics. The quality of the mapped circuit depends on the richness of the cell library. This thesis proposes two different approaches for library-free technology mapping aiming delay reduction in combinational circuits. Both algorithms rely on a cell topology able to implement Boolean functions using minimal transistors stacks. They reduce the overall number of serial transistors through the longest path, considering that each transistor network of a cell has to obey to a maximum admitted chain. The mapping algorithms are integrated to a cell generator that creates cells with minimal transistor stacks. This cell generator is also in charge of performing the transistor sizing. Significant gains can be obtained in delay due to both aspects combined into the proposed mapping tool.

Keywords: technology mapping, logic synthesis, virtual cell library, standard cell library, automatic cell generator.

Mapeamento tecnológico para Bibliotecas Virtuais Baseado em Células com Cadeias Mínimas de Transistores em Série

RESUMO

Atualmente, as tecnologias disponíveis para a fabricação de dispositivos eletrônicos permitem um alto grau de integração de semicondutores. Entretanto, esta integração torna o projeto, a verificação e o teste de circuitos integrados mais difíceis. Normalmente, o projeto de circuitos integrados é consideravelmente afetado com a diminuição do tamanho dos dispositivos eletrônicos em tecnologias sub-micrônicas. Conseqüentemente, os projetistas adotam metodologias rígidas para produzir circuitos de alta qualidade em tempo razoável. Ferramentas de auxílio ao projeto de circuitos eletrônicos são utilizadas para automatizar algumas das etapas do projeto, ajudando o projetista a encontrar boas soluções rapidamente. Uma das tarefas mais difíceis no projeto de circuitos integrados é fazer com que o circuito respeite as restrições de atraso. Isto depende de várias etapas do processo de síntese. Em metodologias baseadas em bibliotecas de células, isto está diretamente relacionado ao algoritmo para mapeamento tecnológico e as células disponíveis na biblioteca. O atraso de cada célula depende do tamanho dos transistores e da topologia da rede de transistores. Isso determina as características de atraso, potência e área de uma célula. O mapeamento tecnológico define as principais características estruturais do circuito, principalmente em área, potência e atraso. A qualidade do circuito mapeado depende das células disponíveis na biblioteca de células. Este trabalho propõe um novo método para mapeamento com bibliotecas virtuais para redução de atraso em circuitos combinacionais. Ambos os algoritmos baseiam-se em uma topologia de células capaz de implementar funções Booleanas com cadeias mínimas de transistores em série. Os algoritmos reduzem o número de transistores em série do caminho mais longo do circuito, considerando que cada célula é implementada por uma rede de transistores que obedecem um número máximo de transistores em série. O número de transistores em série é calculado de forma Booleana, garantindo que este seja o número mínimo necessário para implementar a função lógica da célula. Os algoritmos estão integrados a um gerador de células que utiliza tal topologia e realiza o dimensionamento dos transistores. Ganhos significativos podem ser obtidos combinando estas duas técnicas em uma ferramenta para mapeamento tecnológico.

Palavras-Chave: mapeamento tecnológico, síntese lógica, bibliotecas de células, bibliotecas de células virtuais, geradores de células.

1 INTRODUCTION

According to the ‘Moore’s Law’ (MOORE, 1965), since the invention of the integrated circuit (IC) in 1958, the number of transistors that can be placed “inexpensively” on an integrated circuit has increased exponentially due to advances on technology scaling. Currently, billions of electronic components can be integrated on a single chip. The processing speed and memory capacity of digital electronic devices almost increases in the same proportion of the number of transistors.

Although deep-submicron microelectronic technologies enable greater degrees of semiconductor integration, such integration makes the design, verification, and test challenges more difficult. The circuit design is often the first area under assault by the effects of aggressive scaling in deep-submicron technologies. Effects like leakage power, noise and electro-migration were considered irrelevant in early technologies. Currently, the analysis of these effects is crucial for a successful design. Therefore, designers have adopted strict methodologies to deal with the challenge of developing high quality designs on a reasonable time. Electronic Design Automation (EDA) tools play an important role, automating some of the design phases and helping the designer to find a good solution faster.

The methodology adopted by most of the EDA flows is based on standard cell libraries. In a typical standard cell based flow the synthesis starts from a high-level description using Hardware Description Languages (HDL), such as VHDL (VHDL ORG, 2008) and Verilog (VERILOG DOT COM, 2008), at Register Transfer Level (RTL). The second step is the logic synthesis that performs several logic manipulation procedures over the high level description resulting in a netlist composed by a set of cells of the standard cell library. The last step is the physical synthesis that places and routes the cells of a netlist on a floorplan. The main advantage of this methodology is that each cell in the library is fully characterized through many simulations, resulting in a set of accurate information about the behavior of the cell. Thereby, the designer can, with the aid of an EDA tool, predict with a very good precision the characteristics of the final circuit.

Even though the available EDA tools perform a good job on finding good solutions for a given design, there are still some open issues in the automation flow. Furthermore, every time that the manufacturing technology process advances to the next generation, new problems come up. It demands a constant update in the available tools or even completely new tools. Hence, there is a high cost associated to the technology process shifting. It requires investments on tools and on manufacturing process. Alternatively, the designer can explore other optimization strategies in order to increase performance and to reduce area and power without changing the technology.

One of the big challenges in high-performance circuits design is the timing closure of the combinational logic (or random logic). Usually, combinational logic is not regular enough to be implemented in an intuitive design flow. Furthermore, it can be changed until the last steps of the design cycle. Logic synthesis has been shown to be an effective tool for designing logic circuits, especially for semicustom designs using a standard cell methodology. The computer-aided synthesis of a logic circuit involves two major steps: the optimization of a technology-independent logic representation, using Boolean and/or algebraic techniques, and technology mapping. Logic optimizations are used to modify the structure of a logic description, such that the final structure has a lower cost than the original one. These optimizations are performed before the technology mapping.

Technology mapping is the step of logic synthesis that chooses the cells that will be used to implement a design in a given technology. This step of logic synthesis has a major impact on the structure of the circuit and, consequently, on delay and area characteristics. Most existing techniques are based on static pre-characterized libraries (standard cell methodology), where a set of cells is defined and characterized for a given technology. First methods for technology mapping (KEUTZER, 1987-a) (DETJENS, 1987) (ABOUZEID, 1992) (MAILHOT, 1993) (LIEN, 1992) used trees as the initial description of the circuit to be mapped. More recent methods (LEHMAN, 1997) (KUKIMOTO, 1998) (STOK, 1999) (MISHCHENKO, 2005) are based on Directed Acyclic Graph (DAG) representations that allow duplicating logic to some extent to increase speed. Another important contribution to technology mapping was Boolean matching (MAILHOT, 1993), where the matching of a portion of the circuit and a cell from the library is done by comparing the Boolean function of the candidates, instead of the structure. Structural comparison would not be able to find all matches.

In the early days of technology mapping, it was considered that the use of a cell generator would enable the use of larger virtual (built on demand) cell libraries. Berkelaar (1988) has presented a pioneer work aiming cell generators, which maps decomposed logic expressions onto complex gates. Reis (1997) presented another approach which uses a Binary Decision Diagram (BDD) representation for the circuit network, and performs BDD decomposition using constraints in the number of serial transistors. Each decomposed BDD is mapped onto a static CMOS complex gate. The work in (CORREIA, 2004) dynamically explores many embedded AND/OR decompositions by using n-ary trees for the circuit network representation. Each sub-tree that is limited by the number of serial transistors is also mapped onto static CMOS cells. In (JIANG, 2001) two techniques for technology mapping are presented. The first method maps circuits to a virtual cell library of complex static CMOS gates. The second technique uses a mixed logic of static CMOS and PTL gates, considering the relation between PTL and BDDs.

Unfortunately, the use of such approaches was not widely verified in a commercial level, even if other references suggest that the increased number of cells in a library could lead to significant improvements in the quality of the final design (KEUTZER, 1987-b) (SCOTT, 1994) (SECHEN, 1996) (GAVRILOV, 1997). A recent approach presented in (ROY, 2005) suggests that the addition of some custom cells to a library can improve the speed of the final circuit.

1.1 Motivation and thesis contributions

According to the previously statements, one of the hardest challenges of an IC design is to meet the timing requirements. It depends on several steps of the synthesis flow. In standard cell based flows, it is directly related to the technology mapping algorithm and the cells available in the library. The performance of a cell is directly related to the transistor sizing and the cell topology. The transistor sizing also affects the power consumption and, off-course, the cell area.

Recently, some methods for generating efficient cell networks were proposed (KANECKO, 1997) (POLI, 2003) (TANAKA, 2004) (SCHNEIDER, 2005), including a method to compute the minimum number of transistors in series needed to implement an arbitrary Boolean function that was proposed by Schneider (2005). The reduction of the number of series switches leads to timing efficient networks.

The topology presented in (SCHNEIDER, 2005) was only verified at the cell level, lacking of an efficient methodology to evaluate the use of these cell topologies in larger circuits. Motivated by this, this thesis presents two different methods for "VIRtual library technology MApping" (VIRMA) based on DAGs. Both mapping methods combine the method for Boolean computation of the number of series transistors introduced by Schneider (2005) with state of the art technology mapping algorithms inspired by the approaches presented by Stok (1999) and Mishchenko (2005). Significant gains can be obtained in delay due to both aspects combined into the proposed mapping tool.

Since this cell topology was not well explored yet, both algorithms implemented in the VIRMA tool are library-free. It chooses the transistor configuration for the cells that will have to be created through a cell generation tool in a subsequent step. Currently, the VIRMA mapping flow is integrated to a cell generator that implements the techniques proposed in (Rosa, 2008). This cell generator is also in charge of performing the transistor sizing.

1.2 Thesis organization

The remaining of this thesis is organized as follows: chapter 2 review some general concepts required for a better comprehension of the proposed methods. Specific concepts of technology mapping as well as some previous approaches are described in the chapter 3. The new approaches for library-free technology mapping are presented in the chapter 4. All results obtained through the implemented prototypes are shown in the chapter 5. Finally, conclusions and future works are presented in the chapter 6.

2 TERMINOLOGY AND BASIC CONCEPTS

Electronic design automation is divided into many sub-areas. We are particularly interested in logic synthesis and physical synthesis (related to cell generators) concepts. This chapter introduces some of these concepts and definitions that will be used in the following chapters. For next chapters, we assume the knowledge of all definitions described here.

2.1 Boolean logic and logic expressions

The **Boolean domain** (*B*) is defined as a two element set, say, $B = \{0, 1\}$, whose elements are interpreted as logical values, typically $0 = false$ and $1 = true$. A Boolean function describes how to determine a Boolean value output based on some logical calculation from Boolean inputs. A **Boolean function** is a function of the form $f: B^n \rightarrow B$, where $B = \{0, 1\}$ is a Boolean domain and where n is a non-negative integer. In the case where $n = 0$, the "function" is simply a constant element of *B*. More generally, a Boolean-valued function is a function of the type $f: X \rightarrow B$, where *X* is an arbitrary set and where *B* is a Boolean domain. If $X = [n] = \{1, 2, 3, \ldots, n\}$, then *f* is a **binary sequence** of length *n*. Therefore, there are 2^{2^n} such functions.

The **support** of a Boolean function is the set of variables that may change the output value of a function. For instance, consider the function $f(a,b,c)=ab+a'c$. Its support is the set $\{a, b, c\}$. An **input vector** is an element defined in Boolean domain and indicates the value of each variable that defines the Boolean space. An input vector $v \in B^n$ belongs to the ***ON-set*** of *f* if and only if $f(v) = 1$. Otherwise, if $f(v) = 0$, then *v* belongs to the ***OFF-set*** of *f*. While the vectors $v_1 = \{1,1,0\}$ and $v_2= \{0,1,1\}$ belong to the ON-set of the function $f(a,b,c)=ab+a'c$, the vector $v_3 = \{1,0,0\}$ goes into the OFF-set of $f(a,b,c)$.

A **logic expression** (or equation) is a Boolean function representation. Each function is unique for any application $f: B^n \rightarrow B$ in the whole Boolean space. However, a Boolean function has infinite representations. All Boolean functions can be expressed in a canonical form through **sum of products** (**SOP**) or **product of sums** (**POS**). A SOP is said canonical when all variables appear in all products. Every instance of a Boolean variable is called **literal** according to Wagner (2006). A product of literals is formally called **cube**. For instance, $\{a, b, c\}$ is a cube interpreted like $a.b.c$ or just like abc. A **minterm** is a cube that contains all variables of the function support.

Figure 2.1 shows a truth table and the minterms of the Boolean function *f*. Equation 2.1 represents *f* through a SOP in a canonical form. This equation has 32 literals and it is not a minimal SOP. Karnaugh maps (KARNAUGH, 1953) and the Quine-McCluskey method that comes from (QUINE, 1955) and (MCCLUSKEY, 1956) are the main exhaustive optimization techniques for two-level minimization. Although they are not practical algorithms for large circuits, they are easy to use and simple to understand. The Espresso algorithm (McGeer, 1993) is a heuristic method for two-level minimization that is computationally less expensive and presents good results. An example of two-level minimization can be seen in Figure 2.2. It shows the Karnaugh map for the Boolean function *f*. The minimal cover for the ON-set is composed by four cubes. It can be represented through the Equation 2.2. Equation 2.3 shows the minimal cover for the OFF-set of function *f*.

$$f = \overline{a}.\overline{b}.c.d + \overline{a}.b.\overline{c}.\overline{d} + \overline{a}.b.\overline{c}.d + \overline{a}.b.c.d + a.\overline{b}.\overline{c}.d + a.b.\overline{c}.d + a.b.c.\overline{d} + a.b.c.d \qquad (2.1)$$

$$ON - set(f) = \overline{a}.c.d + \overline{a}.b.\overline{c} + a.\overline{c}.d + a.b.c \qquad (2.2)$$

$$OFF - set(f) = \overline{a}.c.d + \overline{a}.b.\overline{c} + a.\overline{c}.d + a.b.c \qquad (2.3)$$

a	b	c	d	*f*		
0	0	0	0	0		
0	0	0	1	0		
0	0	1	0	0		
0	0	1	1	1	$\rightarrow$	$\overline{a}.\overline{b}.c.d$
0	1	0	0	1	$\rightarrow$	$\overline{a}.b.\overline{c}.\overline{d}$
0	1	0	1	1	$\rightarrow$	$\overline{a}.b.\overline{c}.d$
0	1	1	0	0		
0	1	1	1	1	$\rightarrow$	$\overline{a}.b.c.d$
1	0	0	0	0		
1	0	0	1	1	$\rightarrow$	$a.\overline{b}.\overline{c}.d$
1	0	1	0	0		
1	0	1	1	0		
1	1	0	0	0		
1	1	0	1	1	$\rightarrow$	$a.b.\overline{c}.d$
1	1	1	0	1	$\rightarrow$	$a.b.c.\overline{d}$
1	1	1	1	1	$\rightarrow$	$a.b.c.d$

Figure 2.1: Truth table for the function *f* and the respective minterms.

cd \ ab	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	1	1	1	0
10	0	0	1	0

Figure 2.2: Karnaugh map for the Boolean function *f*.

Both equations can also be represented as **factored forms**. According to Brayton (1987), a factored form can be defined as a representation of a logic function that is either a single literal or a sum or product of factored forms. It is very similar to a parenthesized algebraic expression. This parenthesized representation seems to be the most appropriate representation for use in multilevel logic synthesis. As an example, consider the representations in Figure 2.3. The parenthesized expression can be seen as a **logical operator tree**. Any representation with more than two levels is called a multilevel representation. In this example, the logical operator tree has depth four (or four levels).

There are several methods for obtaining different factored forms for a given logic function. These methods range from purely algebraic ones, which are quite fast, to so-called Boolean ones, which are slower but are capable of giving better results. Since obtaining an optimal (shortest length) factorization for an arbitrary Boolean function is an NP-hard problem (MINTZ, 2005), all practical algorithms for factoring are heuristic and provide a correct, logically equivalent formula, but not necessarily a minimal length solution. Brayton (1987) has presented one of the most known heuristics for algebraic factorization. More recently, another heuristic was introduced by Mintz (2005). Usually, it gets significantly better factorizations than former algebraic factoring and are quite competitive with Boolean factoring but with lower computation costs.

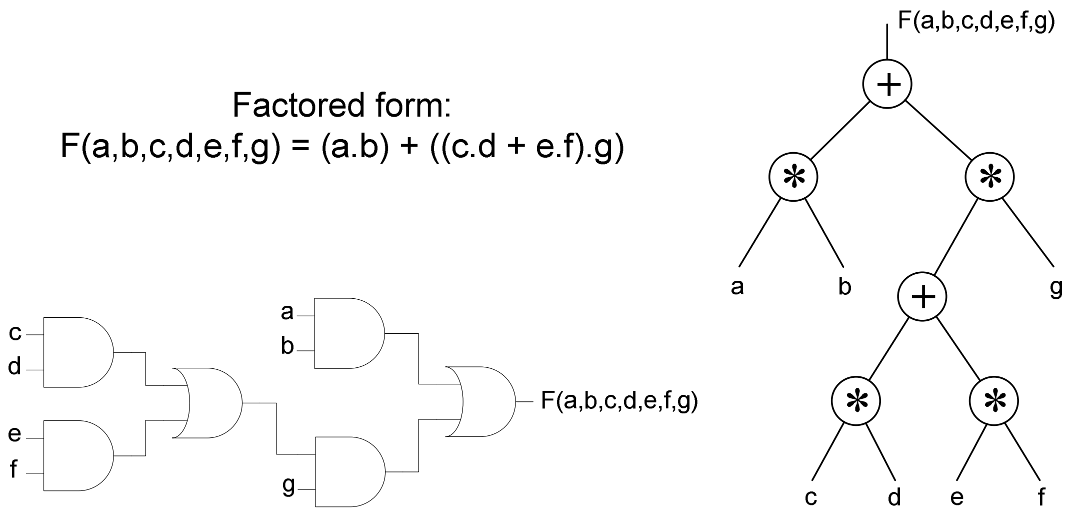

Factored form:
F(a,b,c,d,e,f,g) = (a.b) + ((c.d + e.f).g)

Figure 2.3: Multilevel representations.

2.2 Logic styles of transistor networks

A **logic gate** or **logic cell** can be defined as a network of transistors (or other electronic/electromagnetic components). The association of these components is used to implement arbitrary Boolean functions. Usually, logic gates are created following a given topology. The most common logic styles are the Pass-Transistor Logic (PTL) and the Complementary Series/Parallel (CSP) CMOS (also known as static CMOS). Independently of the topology, the output of the cell is either connected to *VDD* or *GND* through one or more paths of interconnected transistors. A path from *VDD* to the cell output is called **pull-up path**, while a path from *GND* to the cell output is called **pull-down path**.

There are several techniques for an automated transistor network generation. Buch (1997), Hsiao (2000), Shelar (2001), Shelar (2002) and Avci (2003) have proposed methods for PTL network generation. Most of them are based on Binary Decision Diagrams[1] (BDDs). Every node of a BDD is a decision point which matches to a two-input multiplexer (refer to Figure 2.4). This element implements the Shannon expansion that is expressed by Equation 2.4. In our BDD representation, the dashed arcs represent the decision for the negative cofactor(*f(a=0)* in the example of Figure 2.4), while the other arcs correspond to the positive cofactor (*f(a=1)* in the example). The composition of the cofactors and the decision variable (*a* in this case) are able to express the function *f* through the Shannon expansion. Due to this, a PTL cell can be easily derived from a BDD using pass transistors to build multiplexers. This will be demonstrated in Example 1.

$$f = a.f(a=1) + \overline{a}.f(a=0) \qquad (2.4)$$

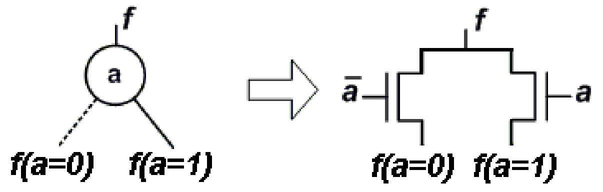


Figure 2.4: Correspondence among BDD nodes and multiplexers.

Example 1: Consider the function *f(a,b,c,d)* such that the ON-set and the OFF-set are represented by Equations 2.5 and 2.6, respectively.

$$ON - set(f) = \overline{a}.\overline{b} + \overline{a}.\overline{c} + \overline{b}.\overline{c} + \overline{b}.\overline{d} \qquad (2.5)$$

$$OFF - set(f) = a.b + b.c + a.c.d \qquad (2.6)$$

[1] Binary Decision Diagrams were introduced by Lee (1959). The basic idea from which this data structure was created is the Shannon decomposition (SHANNON, 1938). These two concepts resulted in efficient data structure proposed by Bryant (1986). Few years later, Brace (1990) extended it to a strongly canonical form that is so-called *Reduced Ordered Binary Decision Diagram* (ROBDD). In popular usage, the term BDD almost always refers to ROBDDs.

Figure 2.5 shows a BDD that represents the Boolean function *f(a,b,c,d)* and a possible PTL implementation. Every BDD node was replaced by a pair of transistors implementing a multiplexer that is controlled by the node variable. The resulting transistor network is composed by NMOS transistors only. Moreover, it has an inverter in the cell output that works like a signal amplifier. This way, the assignments for the source nodes (the terminal nodes *0* and *1* of the BDD) are also inverted for a correct implementation. Consider the input vector *v(a,b,c,d) = {0, 1, 0, 0}*. In the Figure 2.6, this vector activates a path in the BDD that leads to the *terminal node 1*. In the PTL transistor network, a pull-down path is sensitized. Therefore, it gives *VDD* in the inverter output which corresponds to the logical value 1. ■

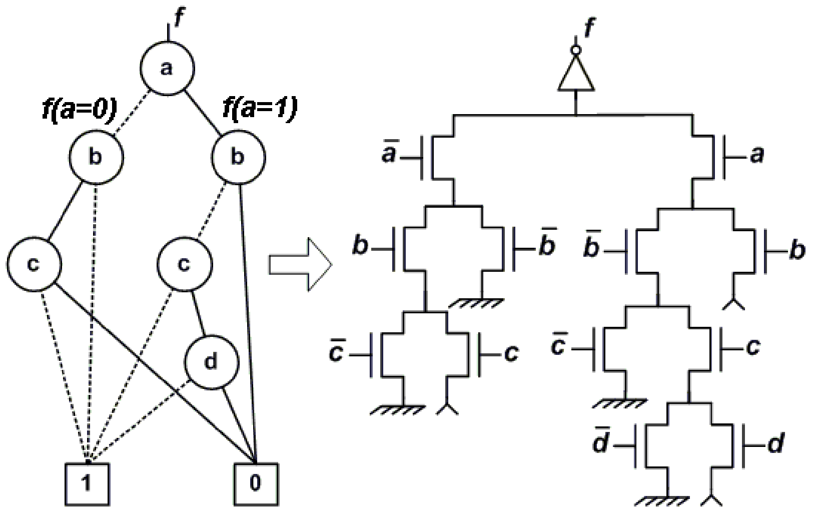


Figure 2.5: PTL transistor network derived from a BDD.

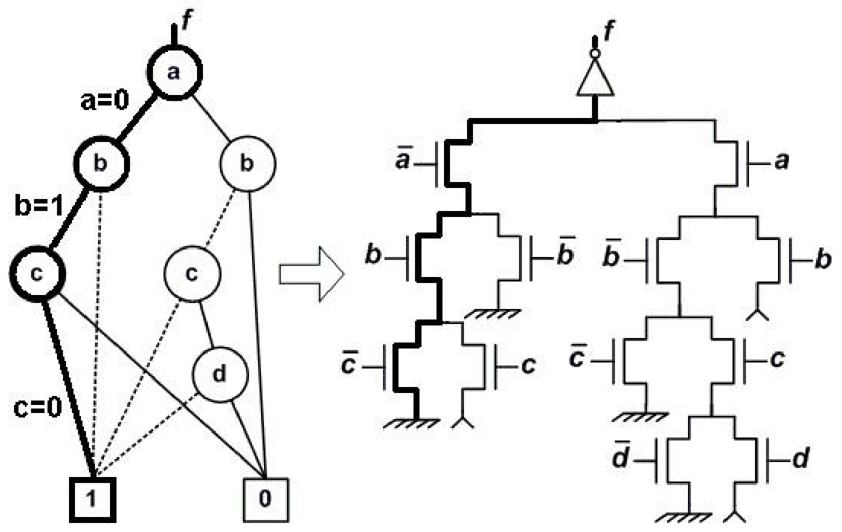
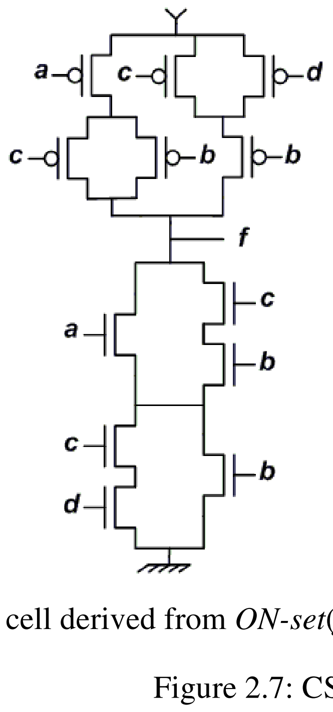


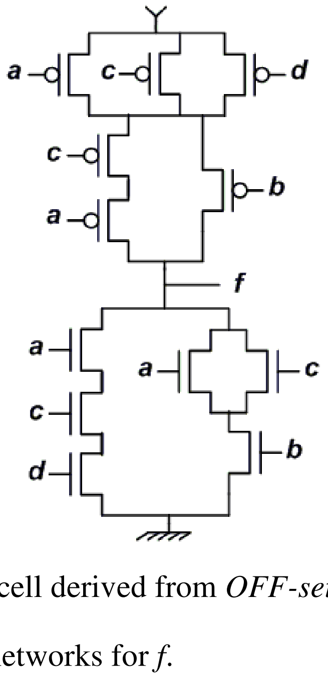
Figure 2.6: Path sensitization in a PTL network.

Generally, the number of transistors in PTL cells is linearly proportional to the number of BDD nodes. There are techniques to reduce this number in some special cases. These techniques are discussed in (ROSA, 2006). In PTL cells, the same intermediate node can be both in a pull-up path and in a pull-down path. When the transistor network has these shared nodes, it is said a **non-disjoint** network. Another important point is the length of the longest path. The maximum number of stacked transistors corresponds to the number of arcs in the longest path in the BDD.

Complementary Series/Parallel cells are implemented using two disjoint transistor planes. The **pull-up plane** (**pull-down plane**) corresponds to the set of interconnected transistors between the cell output and *VDD* (*GND*). While the pull-up plane is only composed by PMOS transistors, the pull-down plane is composed by NMOS transistors. When the pull-up plane is derived from the ON-set equation of a Boolean function, the topology of the pull-down plane is the series/parallel complement of the pull-up plane. In a similar way, CSP cells can be derived from the OFF-set equation. In this case, the pull-down is derived directly from the equation and the pull-up is the series/parallel complement of the pull-down. The topological complementarity assures the logical complementarity of the transistor network. Figure 2.7 shows CSP cells derived from the ON-set and OFF-set equations (Equations 2.5 and 2.6, respectively). In the cell of Figure 2.7a, each pull-up path matches to a cube of the Equation 2.5. The pull-down paths depend on the series/parallel associations of the pull-up plane. Notice that the longest pull-up path has two series transistors, while the longest pull-down path has four series transistors. The longest pull-up and pull-down paths of the cell derived from the Equation 2.6 have three series transistors. This illustrates that the length of the longest path can vary for different implementations of the same logic function.



a) cell derived from *ON-set*(*f*)



b) cell derived from *OFF-set*(*f*)

Figure 2.7: CSP CMOS transistor networks for *f*.

2.3 Minimal length for transistor stacks in standard cell libraries

For approaches that are intended for cell generators, the number of serial transistors inside a cell is an important parameter. The methods presented by Berkelaar (1988), Gavrilov (1997), Reis (1997), Jiang (2001) and Correia (2004) use a maximum allowed number of transistor in series as a parameter to restrict the size of feasible cells. All these approaches are limited to the use of serial/parallel implementations, and the computation of the number of serially connected transistors is done by serial/parallel association. Schneider (2005) proposes a design methodology to implement complex gates with the exact lower bound for the number serial transistors. More details of this approach can be also found in (SCHNEIDER 2006). It is based on the observation that the number of literals on the smallest cube[2] of the ON-set (OFF-set) prime irredundant cover determines the maximum number of serial transistors of the pull-up (pull-down) plane. Due to their topologies, PTL and CSP CMOS cells do not respect the lower bound in several practical cases. A different topology can be used to implement complex gates with the exact lower bound for the number serial transistors. Such topology is called Non-Complementary Series/Parallel (NCSP), where the minimum cover for the ON-set is used to derive the pull-up plane, and the minimum cover for the OFF-set is used to derive the pull-down plane. These cells do not have topological complementary plans. However, they are logically complementary.

Consider the CSP CMOS cells in Figure 2.7. The NCSP cell which implements the Boolean function *f* is shown in Figure 2.8. Just like the CSP CMOS cell, the longest pull-down path has three serial transistors. However, the smallest cube of the minimum cover for the ON-set has two literals. Thus, two serial transistors is the length of the longest pull-up path.

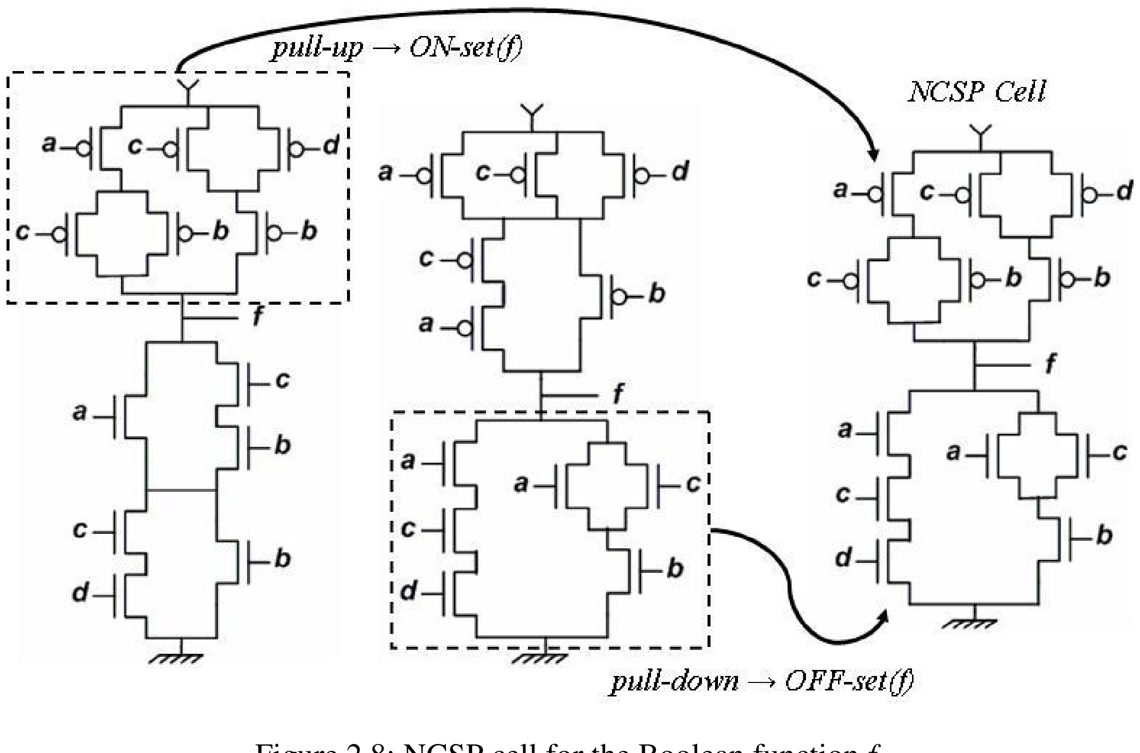
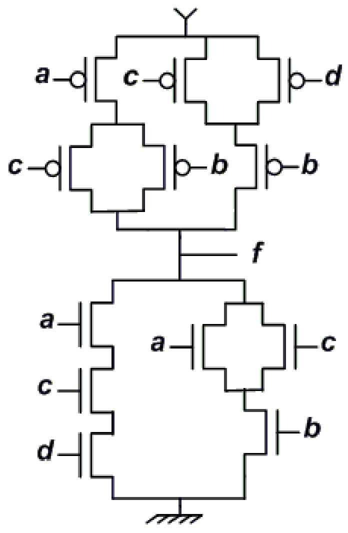


Figure 2.8: NCSP cell for the Boolean function *f*.

[2] The smallest cube is the one with more literals.

Accordingly to Weste (1994), the usual CSP CMOS has one important characteristic: low static power consumption. Significant power is only drawn when the transistors in the CMOS device are switching between on and off states. Although the pull-up and pull-down networks are not complementary in NCSP, they are logically complementary. Hence, there will not be any viable path connecting *VDD* and *GND* for any input vector. Therefore, the same principles of the CSP CMOS logic are applied to the NCSP logic. Figure 2.9 shows this property through an example. The truth table of the function *f* was associated to two columns named *VDD* and *GND*. They represent the occurrence of the voltages corresponding to *VDD* and *GND* in the cell output.

a	b	c	D	*f*	*VDD*	*GND*
0	0	0	0	1	yes	no
0	0	0	1	1	yes	no
0	0	1	0	1	yes	no
0	0	1	1	1	yes	no
0	1	0	0	1	yes	no
0	1	0	1	1	yes	no
0	1	1	0	0	no	yes
0	1	1	1	0	no	yes
1	0	0	0	1	yes	no
1	0	0	1	1	yes	no
1	0	1	0	1	yes	no
1	0	1	1	0	no	yes
1	1	0	0	0	no	yes
1	1	0	1	0	no	yes
1	1	1	0	0	no	yes
1	1	1	1	0	no	yes



NCSP cell

Figure 2.9: Simulation table of the NCSP cell.

2.4 Digital circuits representation

Digital circuits can be represented in many ways. Graphs[3] are widely used for this purpose. A Directed Acyclic Graph (DAG) is one of the most popular representation for logical circuits. On these structures, each logic gate is represented by a vertex and its connections by edges.

A gate *g1* is **fanin** of *g2* if the output of *g1* is connected to a *g2* input. This way, *g2* is part of the **fanout** of *g1*. The **fanout degree** of a logic gate is determined by the number of logic gates connected to its output. For instance, if the logic gate *g1* is connected to inputs of the gates *g2* and *g3* then its fanout degree (or just fanout) is two.

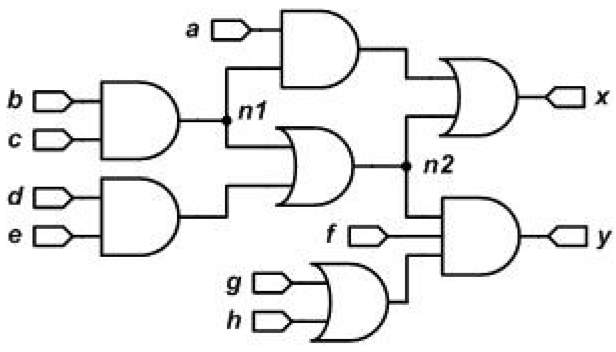
[3] In mathematics and computer science graphs are used to model pairwise relations between objects from a certain collection. A graph refers to a collection of vertexes or 'nodes' and a collection of edges that connect pairs of vertexes. Each vertex has at least an incoming edge and an outgoing edge.

When there is no gate with fanout greater than one, the circuit is called fanout-free. Fanout-free regions are also known as **logic cones**.

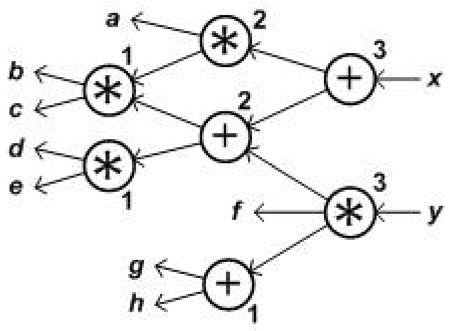
A path in a graph is an alternate sequence of vertexes such that from each of its vertexes there is an edge to the next vertex in the sequence. Formally, it is a set of vertex and connections $\{c_0, g_0, c_1, ..., c_n, g_n, c_{n+1}\}$, where a connection c_i, $1 \leq i \leq n$, connects the output of g_{i-1} to an input of g_i. The connections c_0 and c_{n+1} corresponds to a **primary input** (PI) and **primary output** (PO), respectively. The **depth** of a graph (circuit) is the maximum number of vertexes (gates) in any path of the graph. Consequently, each vertex has a depth value that is given by the distance from the vertexes. Each logic gate *g* has a delay *d(g)* as well as each connection has an associated delay *d(c)*. The delay of path is defined as $d(P)=\sum_{i=0}^{n} d(g_i)+\sum_{i=0}^{n+1} d(c_i)$.

Consider the combinational circuit of Figure 2.10.a. This circuit can be represented by the DAG of Figure 2.10.b. The graph is directed from the primary outputs to the primary inputs. The logic depths, which are numerically represented, can be calculated through a depth-first search algorithm. In this example, the circuit has depth three. The networks *n1* and *n2* are connected to gates with fanout greater than one. In the DAG, these gates are represented by the vertexes that have more than one arriving edge. The graphical representation of Figure 2.10.b is a more formal way to see the usual graphical representation of Figure 2.10.a. Thus, we will refer DAGs just by using the representation of Figure 2.10.a.

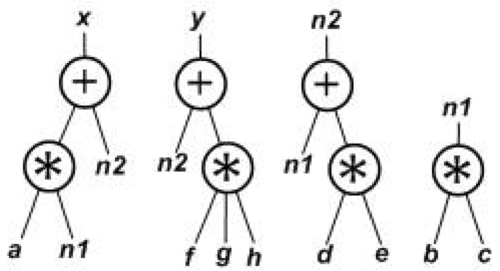
Combinational circuits can also be represented by trees. The trees are a specialized kind of graph where all vertexes have fanout one (they are fanout-free representations). The Figure 2.10.c shows a forest of logical operator trees. Each tree corresponds to a logic cone of the circuit.



a) combinational circuit



b) representation with DAGs



c) representation with trees

Figure 2.10: Combinational circuit representations

3 TECHNOLOGY MAPPING

Technology mapping is a foundation of the logic synthesis process. It is used to define the set of elements from a library that will implement a circuit in a given technology. Typically, the objective function aims the optimal use of all gates in the library to produce a circuit with critical-path delay less than a target value and minimum area. It may sound to be an interpretation of the general logic optimization problem. However, the role of technology mapping is to finish the synthesis of the circuit by performing the final gate selection from a particular library. It assumes that the technology-independent circuit has already undergone through a significant Boolean/structural optimization. In general, these algorithms do not change the structure of the circuit radically, for instance, either by finding common expressions between two or more parts of the circuit or reducing the logic depths of the critical paths. They are simplified because they are constrained by the structure of the equations produced by the technology-independent optimizations. This structural dependence has been studied by Chatterjee (2005), and it is also known as *structural bias*.

Most existing techniques for technology mapping are based on precharacterized libraries, and can be classified into four categories: rule-based mapping (GREGORY, 1986), graph matching (KEUTZER, 1987-a), direct mapping (LEGA, 1988) and functional matching (MAILHOT, 1993). Ideally, technology mapping algorithms should be able to satisfy several goals and to handle different libraries. It is a quite hard task since the cell libraries normally have a different set of cells that implements a limited set of logic functions. A library of fixed size restricts the choices for covering a given circuit. Other approaches for technology mapping propose techniques based on cell generators. Instead of having a static library, they assume that arbitrary cells can be generated on the fly through a cell generator, increasing the matching search space.

Besides specifying the technology mapping problem, this chapter shows specific concepts and techniques for technology mapping, such as data structures, technology libraries and some of the existing methods.

3.1 Cell libraries

A cell library can be defined as a finite set of logic gates that implements different Boolean functions with different *drive strengths* and topologies. Traditionally, the technology mapping methods rely on static precharacterized libraries aiming delay, area and power optimizations. Each cell in the library is fully characterized through many simulations using complex numerical methods. The result of this process is a set of accurate information about the behavior of the cell, concerning timing and power consumption, and its physical area. According to Sechen (2003), the characterization cost of a library is expensive. Hence, commercial libraries are typically composed of few hundred combinational cells and sequential elements like latches and flip-flops for which highly optimized layouts have been optimized for a particular technology. The logic designers are then restricted to using these cells in their circuit designs. Figure 3.1 shows the usual circuit design flow considering technology mapping methodologies based on libraries with a fixed size.

A very simple library is illustrated in Figure 3.2. The cell library names are shown together with their area costs, their function and their DAG representations in terms of two-input AND/OR gates and inverters. The DAG representations correspond to graph patterns used by the matching algorithm. An equivalent library description is given in the Figure 3.3. It corresponds to a subset of the *lib2.genlib* that is distributed with the SIS tool (SENTOVICH, 1992).

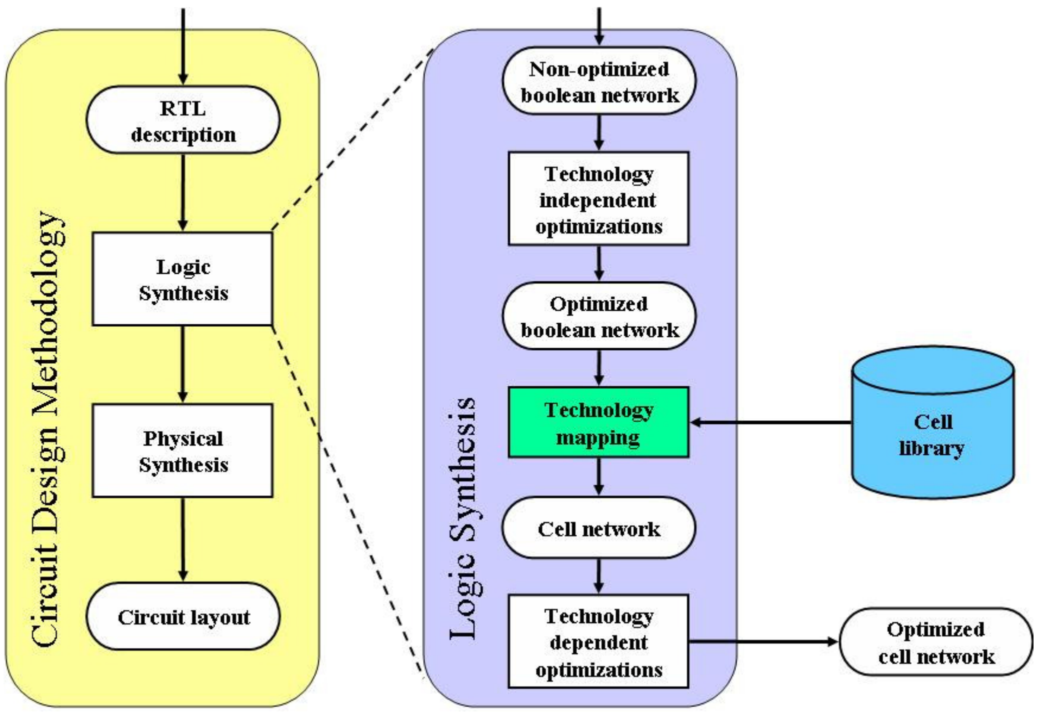

Circuit Design Methodology
RTL description
Logic Synthesis
Physical Synthesis
Circuit layout
Logic Synthesis
Non-optimized boolean network
Technology independent optimizations
Optimized boolean network
Technology mapping
Cell library
Cell network
Technology dependent optimizations
Optimized cell network

Figure 3.1: Digital circuits design methodology.

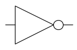
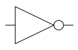
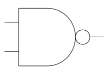
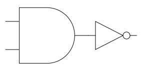
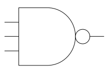
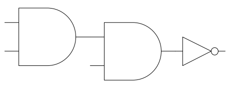
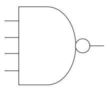
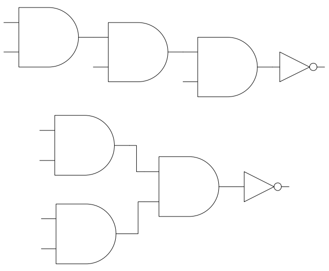
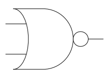
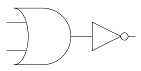
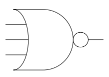
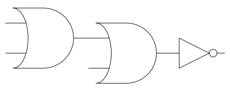
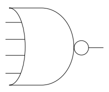
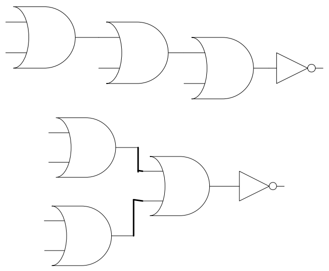
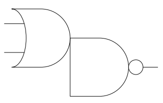
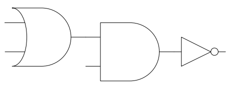
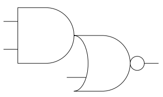
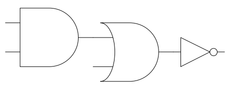
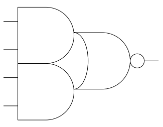
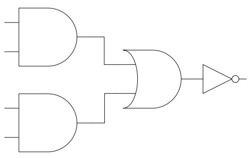
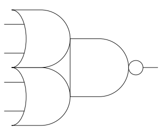
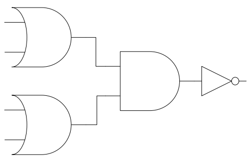
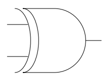
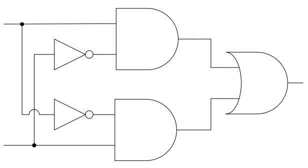
Cell	Cost	Symbol	Primitive graph pattern
INV	2		
NAND2	4		
NAND3	6		
NAND4	8		
NOR2	4		
NOR3	6		
NOR4	8		
AOI21	6		
OAI21	6		
AOI22	8		
OAI22	8		
XOR2	10		

Figure 3.2: Static library example.

```
GATE inv1   928.00 O=!a;
PIN a INV 0.0514 999.0 0.4200 4.7100 0.4200 3.6000
GATE xor  2320.00 O=(!a*b)+(a*!b);
PIN a UNKNOWN 0.1442 999.0 1.7700 5.2300 0.9600 4.6400
PIN b UNKNOWN 0.1381 999.0 1.9400 4.6500 1.1400 5.2200
GATE nand2  1392.00 O=!(a*b);
PIN a INV 0.0777 999.0 0.6400 4.0900 0.4000 2.5700
PIN b INV 0.0716 999.0 0.4600 4.1000 0.3700 2.5700
GATE nand3  1856.00 O=!(a*b*c);
PIN a INV 0.1000 999.0 0.8900 3.6000 0.5100 2.4900
PIN b INV 0.0828 999.0 0.7100 4.1100 0.4200 2.5000
PIN c INV 0.0777 999.0 0.5600 4.3900 0.3500 2.4900
GATE nand4  2320.00 O=!(a*b*c*d);
PIN a INV 0.1030 999.0 1.2700 3.6200 0.6700 2.3900
PIN b INV 0.0980 999.0 1.0900 3.6100 0.6100 2.3900
PIN c INV 0.0980 999.0 0.8200 3.6200 0.5500 2.4000
PIN d INV 0.1050 999.0 0.5800 3.6200 0.3800 2.3900
GATE nor2  1392.00 O=!(a+b);
PIN a INV 0.0736 999.0 0.3300 3.6400 0.4500 3.6400
PIN b INV 0.0968 999.0 0.5000 3.6400 0.7000 3.6600
GATE nor3  1856.00 O=!(a+b+c);
PIN a INV 0.0856 999.0 0.8400 5.0400 1.3000 3.4500
PIN b INV 0.0806 999.0 0.7800 5.0300 1.1400 3.4300
PIN c INV 0.0826 999.0 0.5200 5.0300 0.8400 3.4400
GATE nor4  2320.00 O=!(a+b+c+d);
PIN a INV 0.0887 999.0 0.4100 5.9100 1.1600 3.2000
PIN b INV 0.0867 999.0 0.8500 5.9100 1.5300 3.1800
PIN c INV 0.0867 999.0 1.1100 5.9200 1.7500 3.1900
PIN d INV 0.0887 999.0 1.2700 5.9100 1.9400 3.2000
GATE aoi21  1856.00 O=!((a*b)+c);
PIN a INV 0.1029 999.0 0.7500 3.5200 0.6700 2.5300
PIN b INV 0.0908 999.0 0.6700 3.6400 0.6200 2.5200
PIN c INV 0.1110 999.0 0.5800 3.6400 0.2100 1.2800
GATE aoi22  2320.00 O=!((a*b)+(c*d));
PIN a INV 0.1019 999.0 0.9200 3.4600 0.9400 2.7900
PIN b INV 0.0908 999.0 0.8400 3.6400 0.8500 2.7900
PIN c INV 0.0958 999.0 0.6100 3.6400 0.4900 2.9300
PIN d INV 0.0988 999.0 0.7000 3.6400 0.5400 2.9300
GATE oai21  1856.00 O=!((a+b)*c);
PIN a INV 0.1019 999.0 0.6900 3.9400 0.5300 2.4700
PIN b INV 0.0979 999.0 0.8700 3.9300 0.6300 2.4700
PIN c INV 0.0998 999.0 0.3700 2.0500 0.5700 2.5100
GATE oai22  2320.00 O=!((a+b)*(c+d));
PIN a INV 0.1009 999.0 1.1000 4.0600 0.9000 2.5000
PIN b INV 0.1029 999.0 0.9900 4.0600 0.6800 2.3600
PIN c INV 0.0958 999.0 0.6900 3.6600 0.7400 2.5300
PIN d INV 0.1039 999.0 0.6100 3.6600 0.5600 2.0600
```

Figure 3.3: A subset of the *lib2.genlib*.

The quality of the mapped circuits is very dependent on the richness of the library in terms of the number of implemented logic functions, drive strengths and topologies. Libraries that implement a large number of Boolean functions lead to better results when compared to sparsely populated libraries. Keutzer (1987-b) analyzed the impact of the library size. He demonstrated that a better area optimization can be achieved using large libraries. As Jiang (2001) has observed, the most recent device technologies encourage the usage of complex gates in deep-submicron circuits. It leads to better circuit performance. Nevertheless, it complicates the problem of the traditional library-based technology mapping. In order to increase the use of complex gates in the design, the number of implemented Boolean functions has to be increased. The side effect is that this number grows exponentially. Thus, the number of gates in any library of a reasonable size can only capture a small fraction of the total number of possibilities. It makes the traditional technology mapping too restrictive.

There are approaches for technology mapping based on virtual/dynamic cell libraries (it is also know by library-less technology mapping). These methods assume that each cell in the library is generated on-the-fly by a module generator. Figure 3.4 illustrates the logic synthesis flow of these approaches. The mapping algorithm defines the set of cells used in the circuit implementation. This set is the input for a cell generator which provides the cell layouts that are further used in physical synthesis.

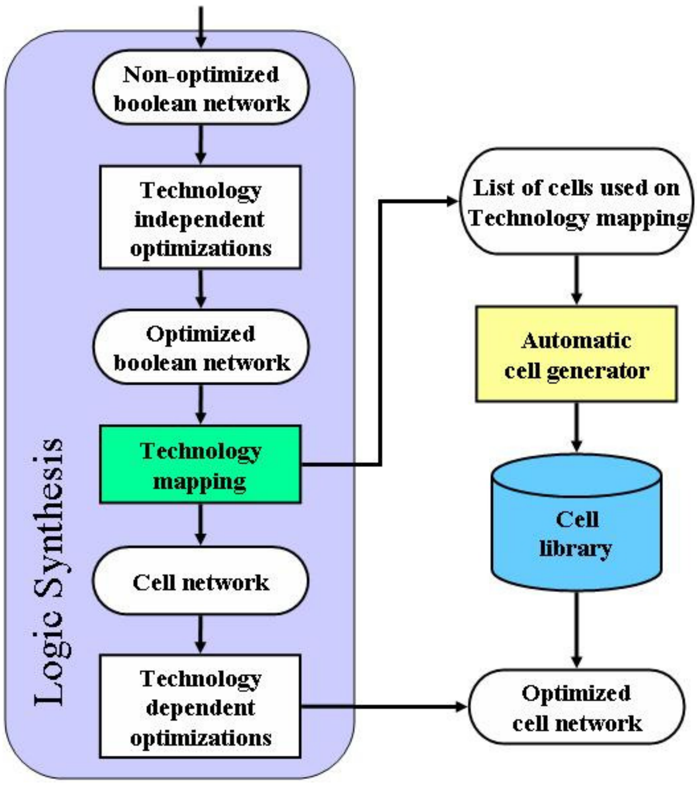

Logic Synthesis
Non-optimized boolean network
Technology independent optimizations
Optimized boolean network
Technology mapping
Cell network
Technology dependent optimizations
List of cells used on Technology mapping
Automatic cell generator
Cell library
Optimized cell network

Figure 3.4: Logic synthesis flow for virtual library technology mapping.

The virtual cell libraries also have a finite number of cells. This number is limited by a set of constraints that represent characteristics of the virtual cells. These constraints can impose topological restrictions such as the maximum number of inputs. For example, consider a library restricted to 2-input cells. It results in the library in Figure 3.5.a. Usually, virtual library-based technology mapping uses the maximum number of series transistor to restrict the pull-up and pull-down planes of each cell. An example of library limited by two transistors in series in both planes is demonstrated in Figure 3.5.b. In this case, the library has only seven cells. Table 3.1 shows the size of the virtual libraries in terms of number of cells. It uses the maximum number of series transistors as constraint to limit the library. These values were calculated considering the CMOS CSP topology. Notice that after the limit of three series transistors the number of cells is much bigger than a normal static library. For instance, the library (4,4) implements 3,503 distinguished Boolean functions (up to four serial transistors in both planes). This number grows exponentially when increasing the number of serial transistors.

The matching and covering methods used by the traditional technology mappers cannot be applied to virtual libraries since the number of patterns is far too large. A structural/Boolean matching method, which compares the Boolean function of the cells in the library with subfunctions of the circuit representation, operates by choosing a covering that is the best solution over all possible matchings. Therefore, if the number of patterns/cells is large then more comparisons would be necessary, increasing the execution time. Virtual cell libraries do not have patterns for cell representation. There are methods, for example, to calculate the number of serial transistors of a sub-graph of the circuit representation. It is enough to know if the calculated result fits in a set of constraints. Hence, there is no need of a pattern matching algorithm.

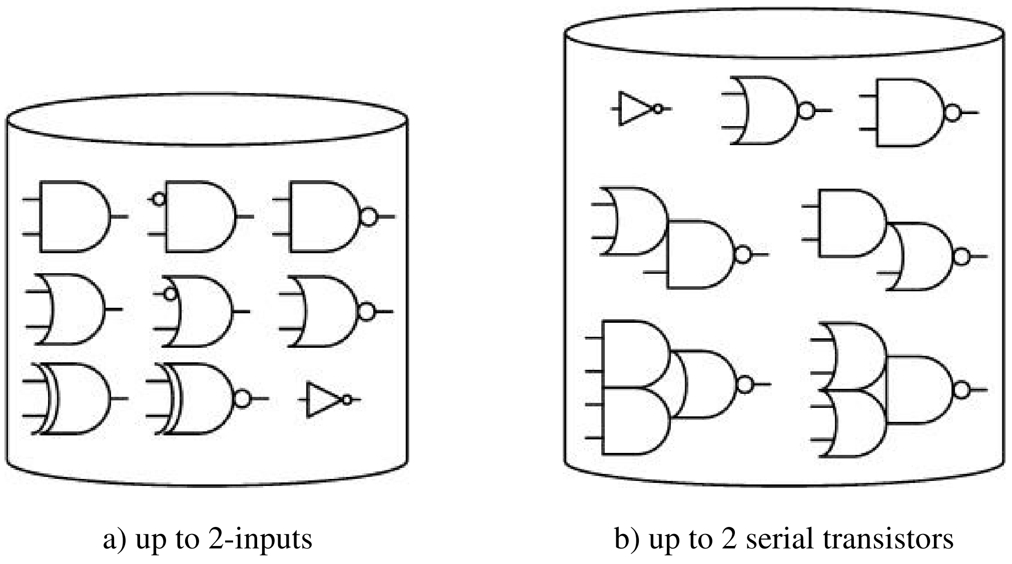

a) up to 2-inputs b) up to 2 serial transistors

Figure 3.5: Virtual library examples.

Table 3.1: The size of CMOS CSP cell libraries induced by the number of serial transistors.

PU / PD	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	7	18	42	90	186
3	3	18	87	396	1,677	6,877
4	4	42	396	3,503	28,435	222,913
5	5	90	1,677	28,435	425,803	6,084,393
6	6	186	6,877	222,913	6,084,393	154,793,519

Source: (SECHEN, 1996).

Clearly, both the quality of the mapped circuit and the CPU time are related to the size of the library. The main barrier of the virtual library approach is the dependency on a good layout generator and the lack of accurate information about the cell behavior. Due to this, the static precharacterized libraries are still popular in the industry. Currently, there are layout synthesis tools able to automatically generate layout for arbitrary complex gates more accurately and efficiently. The big problem is to find good models to estimate area, timing and power on demand. In this sense, some estimative models have been proposed such as the leakage power estimation method introduced by Butzen (2008).

3.2 Subject graphs

All technology mapping methods are performed over a circuit representation that is called subject graph. Digital circuits can be represented in many ways through different kinds of data structures. Each algorithm uses a well defined structure according to the criteria that will be explored during the technology mapping process. The evolution of the technology mapping algorithms is strongly related to their different subject graph types.

The first technology mapping methods used logical operator trees to represent parts of a circuit. Several import methods found in the literature, including more recent approaches, use this kind of subject graph (KEUTZER, 1987-a) (DETJENS, 1987) (ABOUZEID, 1992) (MAILHOT, 1993) (LIEN, 1992) (LEHMAN, 1997) (REIS, 1999) (ZHAO, 2001) (CORREIA, 2004). The operator trees are very simple data structures, and they are usually used to represent logic cones of a circuit. Therefore, algorithms that use these structures can only find optimal solutions for each part of the represented circuit.

The Binary Decision Diagrams are also used as subject graphs in some technology mapping approaches. There are different types of BDDs, and each one of them has a particular property. The TSBDDs were used in (REIS, 1998) to map circuits with CMOS CSP gates. Some methods, such as the methods presented by Yamashita (1997) and Jiang (2001), use ROBDDs to map circuits to PTL cells. A recent method proposed by Rosa (2006) describes techniques to generate gates in different topologies, including NCSP gates.

Other methods, such as (KUKIMOTO, 1998) (STOK, 1999) (MISHCHENKO, 2005), are based on DAGs, or similar data structures, that provide a full representation of the circuit. This kind of representation does not impose limits to the boundaries of the logic cones. This way, optimal solutions can have matches that cross nets with multiple fanout.

As discussed above, each technology mapping method uses a well defined subject graph. Generally, it determines the main characteristics of the algorithm and its application. The memory consumption to store the circuit and its matches will depend on the subject graph type. Therefore, the choice of the subject graph can determine the success of a technology mapping algorithm.

3.3 Conventional technology mapping

As mentioned in the beginning of this chapter, the technology mapping chooses a set of logic gates to implement a circuit in a given technology. This set of cells is defined aiming the minimization of an objective function, and it is not an easy task. The majority of the algorithms reduce the technology mapping problem by handling smaller parts of the circuit. Therefore, the result can be (locally) minimal, but generally it does not correspond to global minimal costs.

The most conventional approach of technology mapping can be described as a three step procedure: technology decomposition, matching phase and covering phase. The most common methods have trees as subject description. This way, an additional step is required right before the technology decomposition. The initial graph must be partitioned into trees. Next section describes the technology mapping using tree-based approaches.

3.3.1 Graph partitioning

In order to apply the tree-based technology mapping, the graph must be converted in a forest of trees. It can be done by breaking the graph at each multiple-fanout point. Each node with fanout greater than one becomes the root of a tree, and fanout of this node becomes a leaf-node of another tree. Thus, this technique does not duplicate nodes in the graph. An example of partitioning is demonstrated in Figure 3.6.

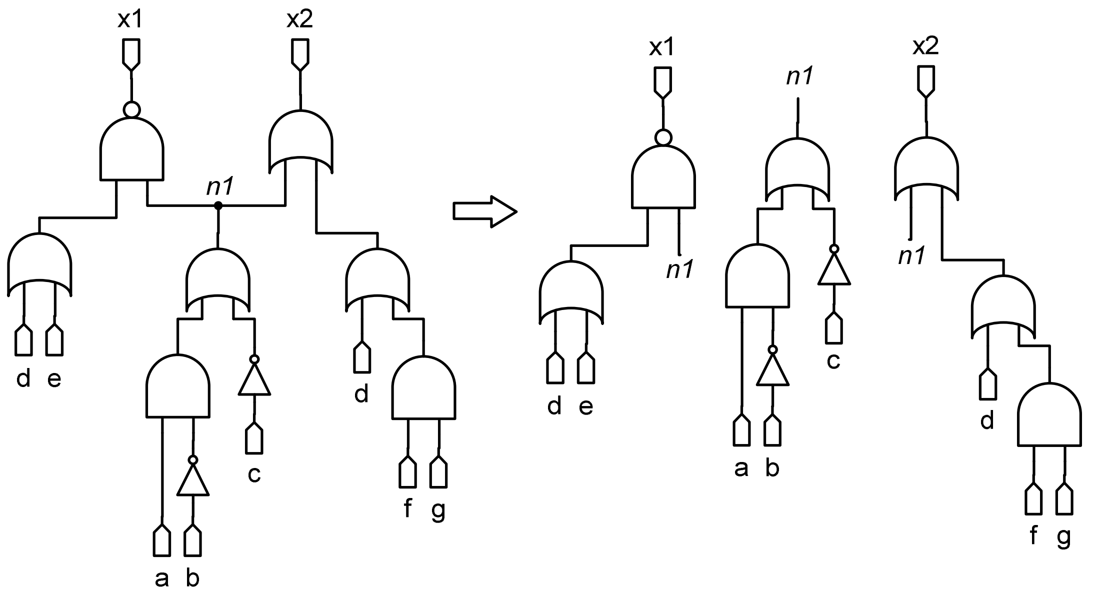


Figure 3.6: Graph partitioning.

3.3.2 Technology decomposition

The technology decomposition phase translates the graph representation into a subject graph decomposed in simple logic primitives. Any complex gate representation is replaced by a set of primitives normally composed of AND, OR and NOT operators or only AND and NOT operators. The main purpose of this step is to facilitate the matching algorithm task and to increase the graph granularity. The graph is decomposed using the same primitives in which the patterns of the cell in the library are represented. Therefore, the matcher can compare the circuit sub-graphs to the pattern graphs. Figure 3.7 shows an example of technology decomposition in the tree rooted by the node *n1*. Three different decompositions are shown. In the first one, the subject graph is expressed by AND, OR and NOT operators. All OR nodes can be replaced by a set of AND and NOT nodes in order to reduced the number of distinguished nodes. This is demonstrated in the second subject graph/tree (from left to right). As a last step, another trick can be used to increase the granularity of the subject graph. A pair of inverter nodes can be added at each input that is not connected to NOT nodes.

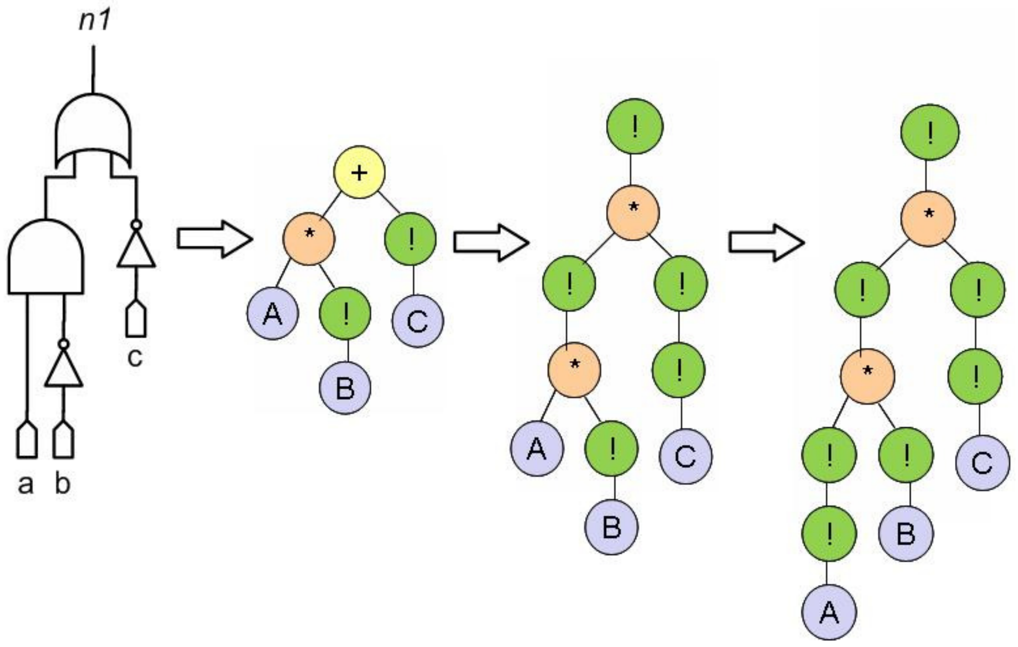


Figure 3.7: Technology decomposition.

3.3.3 Matching phase

This step consists in establishing the initial set of candidate matches attempting to match each node of the graph against each pattern of the library. If there are *p* patterns in the library and *n* nodes in the subject graph, then this naive approach has complexity $O(n \cdot p)$. Structural matchers look for patterns and sub-graphs that are structurally isomorphic. Some approaches reduce the tree matching problem to the string matching problem. It is possible to find all of the strings that match a given string in time proportional to the length of the longest string in the pattern set. Boolean matchers identify patterns independently of the graph structure. There are several approaches for functional matchers that usually give better results than the structural matchers. However, these methods are expensive in terms of CPU time. Most of them have a limited search space containing Boolean functions with support up to 10 variables.

The result of the matching phase is illustrated through Figure 3.8. The subject graph of Figure 3.8 was matched against the library of Figure 3.2. Only the best matches for each node are bound in the graph. All cumulative costs are also shown in Figure 3.8, considering all possible matches.

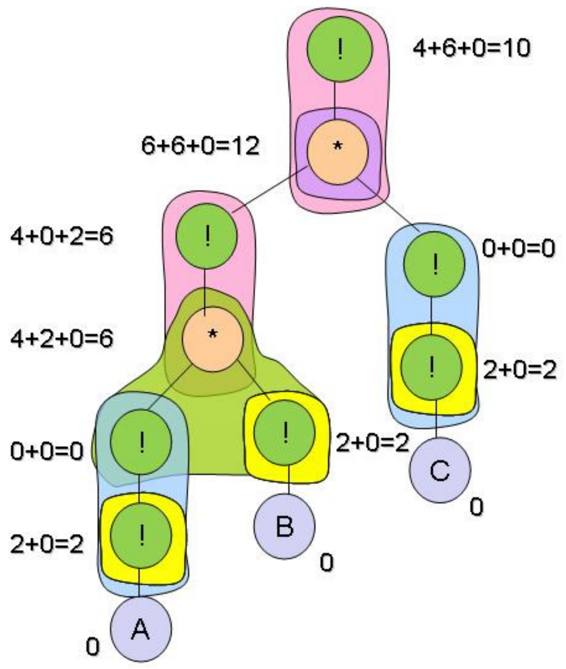


Figure 3.8: Matching generation.

3.3.4 Covering phase

The last step of technology mapping is the covering phase. This procedure finds the optimal set of cells for final circuit implementation. Dynamic programming is a general technique for many algorithms which can be applied to the covering problem.

Consider the problem of finding a minimum area cover for a subject tree *T*. A scalar cost is assigned to each tree pattern, and the cost for a cover is the sum of costs for each pattern in the cover. The minimum-area cover for a tree *T* can be derived from the minimum-area covers for every node below the root of *T*. This is the principle of optimality for tree covering and is used as follows to find an optimal cover for *T*. For every match at the root of the tree the cost of an optimal cover containing that match equals the sum of the cost of the corresponding gate and the sum of the costs of the optimal covers for the nodes which are inputs to the match. For instance, consider the possible covers shown in Figure 3.9. The tree in the right side of the figure represents the minimal cover.

Note that each node in the tree is visited only once through a depth-first search algorithm. It is not necessary to re-compute the optimal cover for each input of each match. Hence, the complexity of this algorithm is proportional to the number of nodes in the subject tree times the maximum number of matches at any node in the subject tree. As result the covering algorithm has linear complexity in the size of the subject tree, and the memory requirements are also linear in the size of the subject tree. The optimal cover of each matched tree is presented in Figure 3.10. Its total cumulative cost is 10.

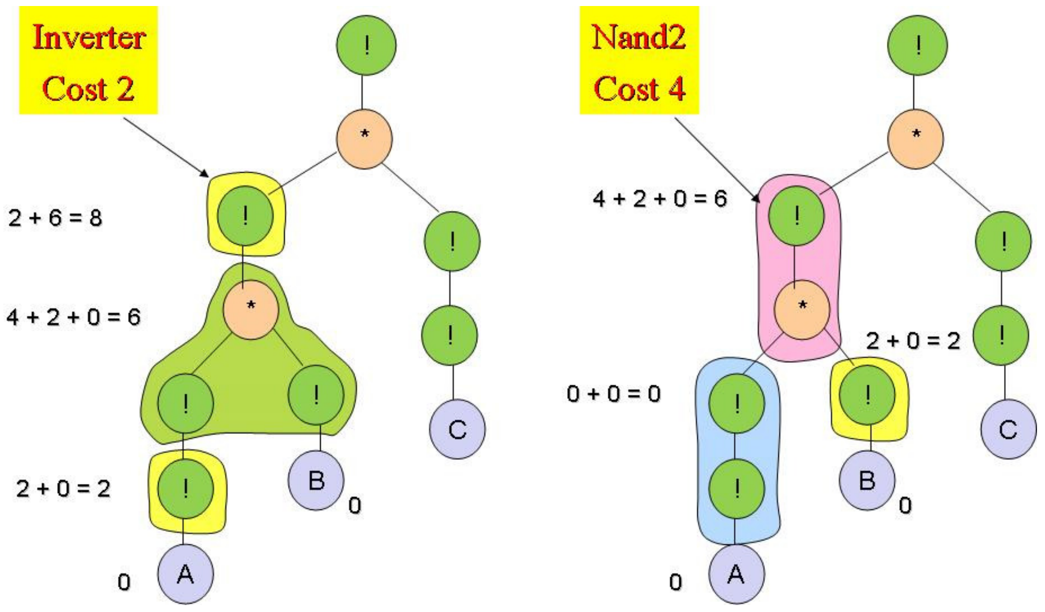


Figure 3.9: Calculating cost on the subject graph.

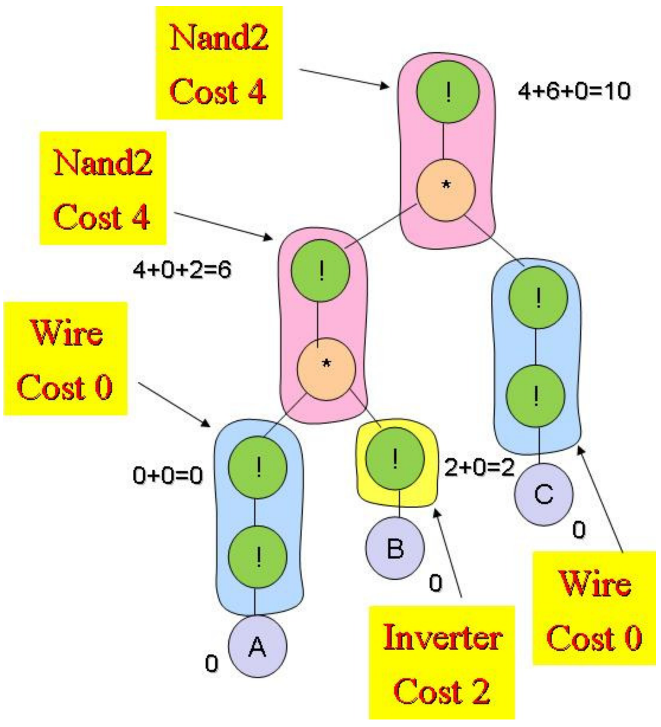


Figure 3.10: Tree covering result.

3.4 Technology mapping for standard cell libraries

This section reviews some of the main methods for technology mapping based on static precharacterized cell libraries, analyzing how they handle the technology mapping problem, the subject graphs and the objective functions. First, it starts with the first technology mapping method that was presented by Keutzer (1987-a). After, the methods of Kukimoto (1998), Stok (1999) and Mishchenko (2005) are reviewed.

3.4.1 Keutzer (1987-a) – DAGON

The first technology mapping algorithm was presented by Keutzer (1987-a). He has found a relationship between technology mapping and programming language compiler techniques. More specifically, matching graph patterns of a technology independent circuit representation against a library of patterns, such as standard cell libraries, is similar to matching graph patterns of intermediate representations of a computer program against the patterns of an instruction set of a given machine.

The result is an algorithm for technology mapping, called DAGON, which is able to minimize area, timing or a function of both. The initial description of a circuit is represented by a DAG. This graph is partitioned into a forest of trees that represents all logic cones of the circuit. Each sub-tree of the forest is matched against the library of patterns that are equivalent to technology cells. The tree matching is made by an external tool called *twig* (TJIANG, 1986). This tool was generally used to construct code generators for programming language compilers. This tool needs to be fed with list of patterns that are matched against circuit trees. The pattern list is composed by small trees in the canonical NAND/NOT form. The developers of the patterns are responsible for providing the cost of each one of them. Therefore, given a tree to be matched, the *twig* tool uses these costs to evaluate cost of candidate matches. Once the matches and their costs are bound in the circuit tree, the covering algorithm can select the patterns/cells that will cover the tree/circuit with minimal cost. The whole DAGON mapping flow is summarized by the Figure 3.11.

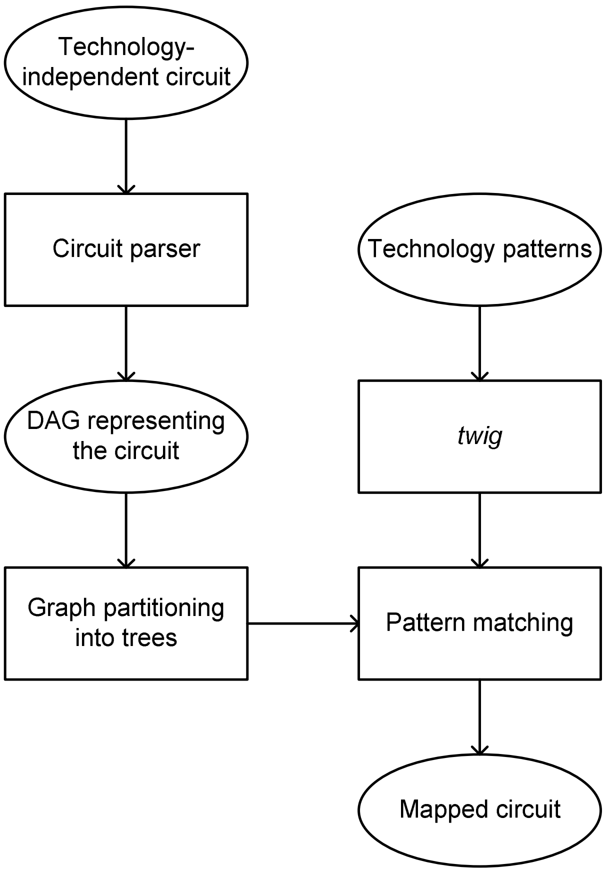
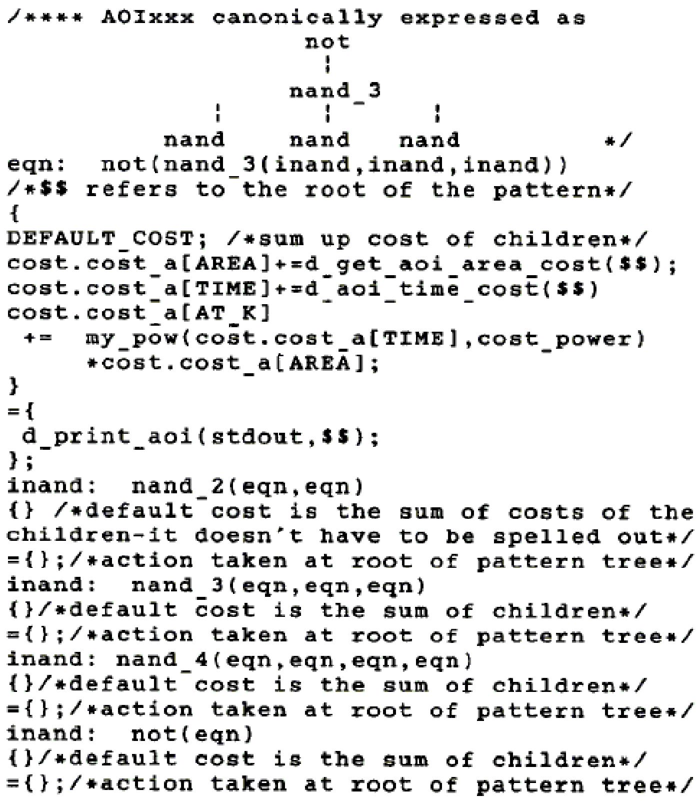

Technology-independent circuit
Circuit parser
Technology patterns
DAG representing the circuit
twig
Graph partitioning into trees
Pattern matching
Mapped circuit

Figure 3.11: DAGON algorithm flow (KEUTZER, 1987-a).

Keutzer gives an example of a set of five patterns for AND-OR-INVERTER (AOI) gates. These patterns can be seen in Figure 3.12. They actually describe sixty-four unique pattern instances from an AOI444 to AOI211. Many of these patterns are symmetric, thus an AOI114 is equivalent to an AOI411. A graphical representation of the set of trees described by these patterns is given in Figure 3.13. Comparing the current cell library descriptions (refer to Figure 3.3) against the library of patterns, we can see that it is a very rudimentary description. It can only describe a single cost value for each pattern while the most recent descriptions are able to describe different costs, even for each cell pin.

Since it uses the tree covering approach, optimal solutions can be found in linear time. The main disadvantage of this method is the size of the library, which is very limited. Moreover, it does not achieve good results regarding timing optimizations since it is a tree based mapping.



```
/**** AOIxxx canonically expressed as
                     not
                      !
                    nand_3
               !      !      !
             nand    nand   nand         */
eqn:  not(nand_3(inand,inand,inand))
/*$$ refers to the root of the pattern*/
{
DEFAULT_COST; /*sum up cost of children*/
cost.cost_a[AREA]+=d_get_aoi_area_cost($$);
cost.cost_a[TIME]+=d_aoi_time_cost($$)
cost.cost_a[AT_K]
 +=  my_pow(cost.cost_a[TIME],cost_power)
     *cost.cost_a[AREA];
}
={
 d_print_aoi(stdout,$$);
};
inand:  nand_2(eqn,eqn)
{} /*default cost is the sum of costs of the
children-it doesn't have to be spelled out*/
={};/*action taken at root of pattern tree*/
inand:  nand_3(eqn,eqn,eqn)
{}/*default cost is the sum of children*/
={};/*action taken at root of pattern tree*/
inand: nand_4(eqn,eqn,eqn,eqn)
{}/*default cost is the sum of children*/
={};/*action taken at root of pattern tree*/
inand:  not(eqn)
{}/*default cost is the sum of children*/
={};/*action taken at root of pattern tree*/
```

Figure 3.12: An AOIxxx pattern (KEUTZER, 1987-a).

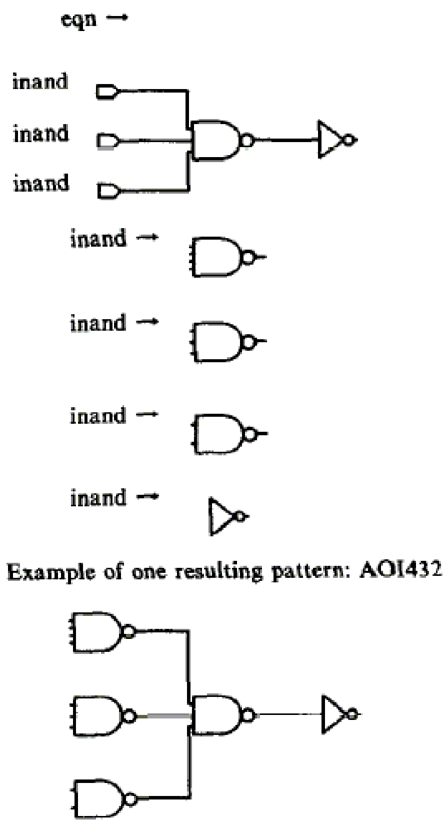


Figure 3.13:Geometric interpretation of Figure 3.12 (KEUTZER, 1987-a).

3.4.2 Kukimoto (1998) – DAG based technology mapping

A pioneer work for standard cell technology mapping, which effectively uses DAGs as subject descriptions, was the algorithm introduced by Kukimoto (1998). As Keutzer (1989) has shown, if a subject graph is a DAG, graph covering for minimal area mapping is a NP-complete problem. When using trees as subject graphs, some methods can guarantee a minimal cover in linear time. It leads to the usual approach of technology mapping algorithms that decompose a subject DAG into trees, solve the technology mapping problem optimally for each tree and glue the results together.

Based on these assumptions, Rudell (1989) worked on a minimal-delay technology mapping. If loading effects are completely ignored, then the minimal-delay mapping problem for subject trees can be solved optimally by dynamic programming in linear time. In the early 90's, the appearance of Field-programmable Gate Arrays (FPGAs) brought a new technology mapping problem. LUT-based FPGAs can implement any function of *k* inputs in a single LUT, where *k* is a fixed constant depending on the FPGA technology. The traditional library-based technology mapping cannot be applied in FPGAs because it would be necessary to create 2^{2^n} patterns that correspond to the number of Boolean functions of *n* variables. Cong (1994) introduced a technology mapping algorithm able to find timing-optimal solutions in LUT-based FPGAs in polynomial time. Unlike the conventional tree mapping, it maps a circuit directly over a DAG.

The dynamic programming approach proposed in (CONG 1994) is not specifically for FPGA. Kukimoto (1998) observed that it could be easily extended to a library-based mapping method. It was the first method to show that the minimum-delay technology mapping problem for DAGs can be solved optimally in polynomial time. This method guarantees the minimal cover under a load-independent delay model. It only considers a fixed delay between each input and the output of a gate, discarding any information about load effects. The method assumes that the mapped circuit can be continuously sized to match the delay of the matches and the actual loads. It also can take into account buffer insertion methods to reduce the load of high fanout nets. As most of the DAG-based methods, Kukimoto's method focuses on delay optimization without any area consideration. Therefore, at each intermediate node the fastest mapping is simply created no matter how critical the node is. It leads to a certain amount a of logic duplication increasing the area of the resulting circuit.

3.4.3 Stok (1999) – Wavefront technology mapping

The wavefront technology mapping algorithm leads to a very simple and efficient implementation that elegantly decouples pattern matching and covering but circumvents that patterns have to be stored for the entire network simultaneously. This approach coupled with dynamic decomposition enables trade-off of many more alternatives than in conventional mapping algorithms. The wavefront algorithm maps optimally for minimal delay on DAGs when a gain based delay model is used. It is optimal with respect to the arrival times on each path in the network.

The *wavefront* was defined as a subgraph of the DAG, such that every path from input to output goes through the subgraph. The subcircuit isolated by the wavefront is bounded by the head and the tail of the wavefront. The head of the wavefront is the boundary closer to the primary outputs (POs) and the tail of the wavefront is the boundary closer to the primary inputs (PIs) of the circuit. If, in Figure 3.14, the vertical line with label 1 is the head and the line with label 0 is the tail, the subgraphs containing the inverters form the wavefront. In addition to decoupling the match generation and covering problems, the wavefront algorithm allows the match generation to work only on a subcircuit, thereby minimizing the number of matches stored at any time. Also, matches are allowed to be generated and maintained dynamically, as opposed to generating all the matches for the entire circuit a-priori.

The algorithm is illustrated through Figure 3.14 (the same example found in Stok (1999)). It supposes that the target library, which will be used to map the circuit, is composed by the cells: NOR, AOI, XNOR and inverter. Furthermore, the wavefront algorithm presumes that the circuit is levelized from inputs to outputs. Initially, both the head and the tail of the wavefront start at level 0 (at the PIs level). The head of the wavefront advances one step and the steps of match generation and implementation are performed for all nets on this level. Matches are generated using Boolean and Structural matchers. All pattern matchers have one thing in common; given (a) net(s) in the circuit, they generate a set of technology cell matches for the subcircuit in the wavefront driving the(se) net(s). All technology/library cells and their embedding in the network are shown in dotted lines.

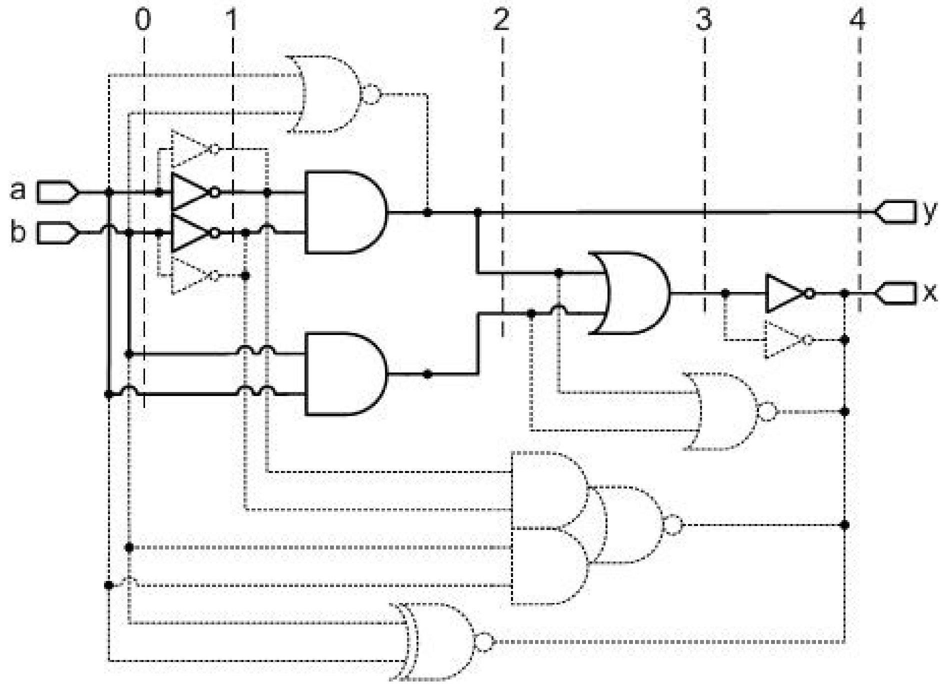
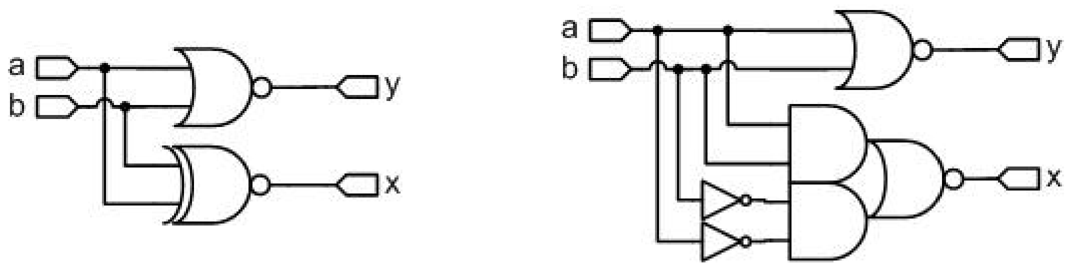


Figure 3.14: Wavefront algorithm (STOK, 1999).

Unlike conventional technology mapping, matches are not limited to fanout-free regions; i.e. the match generation search process performs its search across fanout. However, the subcircuit for the match generation process is isolated by the wavefront; i.e. the match generation search process starts at a net(s) on the head of the wavefront and stops as soon as it encounters nets on the tail of the wavefront or a net on a level below the tail of the wavefront. As matches are generated for a node, they are implemented in the underlying netlist as drivers of a multi-source net. The multiple drivers on a net include the technology-independent cell and all the technology cell matches found for that node. For instance, the net connected to the PO *x* has four technology cells plus the technology-independent cell as drivers. The head of the wavefront keeps advancing one step at a time until matches have been generated for all nets on the highest level of the circuit.

The tail does not start moving until the head has moved for a number of steps equal to the width of the wavefront. Covering for selecting the best match occurs for all nets on the tail of the wavefront. After choosing the best match of a net, a cleanup operation is recursively performed in order to remove all matches deemed sub-optimal and the technology-independent cells. Therefore, the matches for the technology cells are just kept in the sub-graph of the wavefront. The tail of the wavefront keeps advancing one step at a time until all nets on the highest level of the circuit have been covered.

Figure 3.15 demonstrates the effect of using different wave widths. Both the quality of the mapped circuit and CPU time are affected by the width of the wavefront. While the circuit in Figure 3.15.a is the timing-optimal cover, the circuit in Figure 3.15.b is not. When using a wave width of three levels, the XNOR match is not allowed. It reduces the number of generated matches/patterns reducing the CPU time.



a) wave width = 4　　　　　b) wave width = 3

Figure 3.15: Wavefront resulting circuits (STOK, 1999).

3.4.4 Mishchenko (2005) – Technology Mapping with Boolean Matching, Supergates and Choices

Mischenko has recently proposed an approach for library-based technology mapping (MISHCHENKO, 2005) that relies and enhances upon several known techniques, integrated and fine tuned to work in a new way. The previous work on DAG mapping is extended, by proposing new methods for enumerating mapping choices and performing Boolean matching, which guarantees delay-optimum phase assignment at the gate boundaries. These techniques are used to reduce the mapping dependence on the initial subject graph (CHATTERJEE, 2005) (MISCHENKO, 2006). Mishchenko's method is incorporated in the ABC tool (ABC, 2008).

The technology mapping approach proposed by Mishchenko (2005) is devoted to minimize the delay of the longest path in the mapped netlist under a load-independent delay model. As shown by Kukimoto (1998), this problem can be optimally solved using dynamic programming for DAG-covering. The key difference of Mishchenko's method with the conventional approaches based on DAG-covering lies in the matching step, which uses Boolean instead of structural matching. The subject graph is represented as an And-Inverter Graph (AIG). An AIG is a DAG whose nodes represent either AND gates or primary inputs. Its edges represent wires. Inverters are represented by bubbles on the edges. An example of AIG can be seen in Figure 3.16.

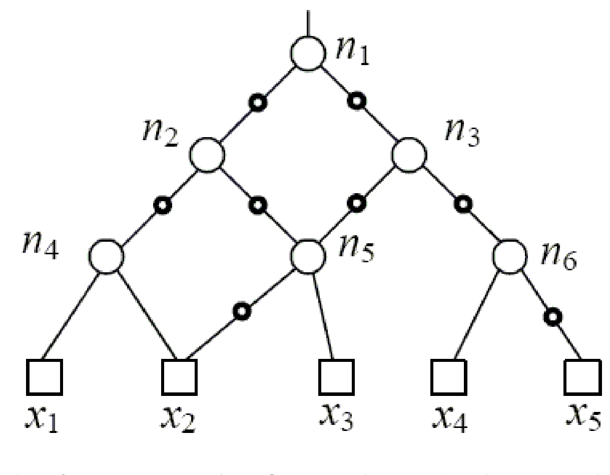
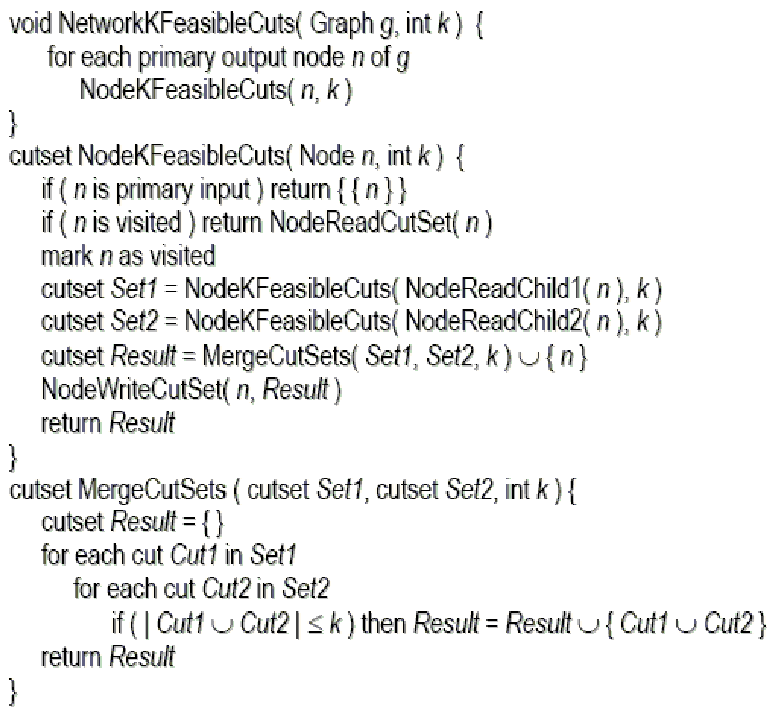


Figure 3.16: An example of an AIG (MISHCHENKO, 2005).

The mapping is divided into 5 steps. First, for every node, it computes all of its *k-feasible cuts*. A feasible cut of a node *N* in the AIG is a set of nodes *{xi}* in the transitive fan-in cone of *N* such that an arbitrary assignment of values to *xi* completely determines the value of *N*. A feasible cut is redundant if the value of a node in the cut is completely determined by an assignment of values to the other nodes in the cut. A k-feasible cut is a feasible cut of size at most *k* that is not redundant. The cut *{N}* composed of node *N* alone is always a *k*-feasible cut of node *N* (for any *k*) and is called the trivial cut. Each *k*-cut of node represents a Boolean function of *k* variables. In the AIG of Figure 3.16, *{n2}*, *{n4, n5}*, *{n4, x2, x3}*, *{n5, x1, x2}*, *{x1, x2, x3}* are all the 3-feasible cuts of *n2*. The algorithm for computing all *k*-feasible cuts for all nodes in the AIG is performed in one pass over the nodes as shown in Figure 3.17.

Second, for every cut, it assigns a formal variable to each node in the cut and computes the function of the corresponding node in terms of these variables. (The function is computed as a bit-vector representing the truth-table.) Third, these functions are looked up in a hash table and matched with gates from the library. Fourth, in topological order, starting with arrival times of the primary inputs, the best arrival time for each node is computed by choosing the library gate with minimum delay. Finally, in reverse topological order, the best covering is chosen. (The last two steps are exactly the same as in traditional mapping.) There is an additional post-processing step, which can recover area on the non-critical paths.

The results presented by Mishchenko (2005) demonstrate that the Boolean matching technique with optimal phase assignment is a better alternative to the graph matching since it produces superior results with shorter run-time. Supergates and choices fit nicely into this framework and greatly improve the quality of mapping by mitigating structural bias. Furthermore, the intermediate networks seen during technology independent synthesis are a useful source of choices for the final mapping. However, these techniques have limitations. The exhaustive cut computation is only practical for cuts up to 6 variables. There is a heuristic method presented in (CHATTERJEE, 2006) that allows non-exhaustive computation of *k*-cuts up to 12 variables.



```
void NetworkKFeasibleCuts( Graph g, int k ) {
    for each primary output node n of g
        NodeKFeasibleCuts( n, k )
}
cutset NodeKFeasibleCuts( Node n, int k ) {
    if ( n is primary input ) return { { n } }
    if ( n is visited ) return NodeReadCutSet( n )
    mark n as visited
    cutset Set1 = NodeKFeasibleCuts( NodeReadChild1( n ), k )
    cutset Set2 = NodeKFeasibleCuts( NodeReadChild2( n ), k )
    cutset Result = MergeCutSets( Set1, Set2, k ) ∪ { n }
    NodeWriteCutSet( n, Result )
    return Result
}
cutset MergeCutSets ( cutset Set1, cutset Set2, int k ) {
    cutset Result = { }
    for each cut Cut1 in Set1
        for each cut Cut2 in Set2
            if ( | Cut1 ∪ Cut2 | ≤ k ) then Result = Result ∪ { Cut1 ∪ Cut2 }
    return Result
}
```

Figure 3.17: Computation of all *k*-feasible cuts (MISHCHENKO, 2005).

3.5 Technology mapping for virtual libraries

There are some technology mapping methods based on virtual libraries. Instead of having a library with a limited amount of cells, these approaches can handle a large number of cells in libraries limited by some constraints. This section review some of these techniques focusing on the objective function, the matching algorithm and the constraints used to restrict the library.

3.5.1 Berkelaar (1988) – The first technology mapping algorithm for cell generators

The algorithm presented by Berkelaar (1988) was a pioneering work aiming at virtual libraries. It relies on the completeness of a generated library in an automatic way. As other traditional technology mapping approaches, it works over DAG-partitioned representation. This method also applies the partitioning procedure over an initial DAG to divide the representation in logic cones. The algorithm takes as input complex Boolean expressions that represent the logic cones of the circuit. All Boolean expressions are denoted by a prefix notation, and they are assumed to be non-redundant and factorized.

The expressions are represented by a specialized kind of graph as demonstrated in Figure 3.18. Each edge symbolizes a literal of the expression. A set of three different nodes connected through two serial edges is equivalent to an AND operation between two literals. This is the case of the edges *j* and *k* of the graph. The association of two different nodes connected through two parallel edges is equivalent to an OR operation. Either the pair of edges *f* and *g* or *h* and *i* represent a parallel association. The size of the sub-expressions is limited to reduce the number of recursive evaluations.

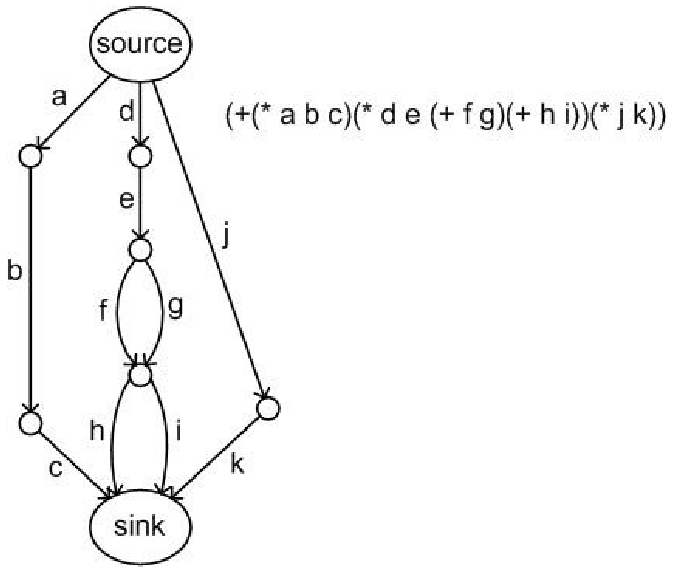


Figure 3.18: Graph expression representation (BERKELAAR, 1988).

The method performs a top-down traversal (from source to sink) over the graph evaluating the expressions from source to sink (giving priority for the operators with less precedence). Each sub-graph starting from any intermediate node to the sink corresponds to a sub-expression. Each sub-expression should match with some cell in the virtual library. When the optimal match is found, the nodes in the bottom of the sub-graph become intermediate variables in the sub-expression. Then this process is recursively repeated until the sub-expression becomes single variables.

The result of the mapping process is a multi-level gate network with minimal logic depth and the lowest possible number of gates. The main problem of this approach is that the algorithm cuts the top of the three first and the bottom intermediate nodes are not mapped yet. Therefore, the algorithm does not know the depth of the uncovered sub-functions. This depth is estimated through a heuristic instead of solving it with a dynamic programming strategy, which could result in non-optimal solutions when compared to dynamic programming. Even if the algorithm works on a graph, it treats single output parts of the circuit, which is equivalent to tree mapping.

3.5.2 Reis (1998) – TABA

A different solution for virtual library technology mapping was proposed by Reis (1998). The method TABA, as it was called, uses Terminal-Suppressed Binary Decision Diagrams (TSBDDs), which are a specialized kind of BDD, as subject graph. These structures have a special property. Each edge of the TSBDD corresponds to a transistor as depicted in Figure 3.19.

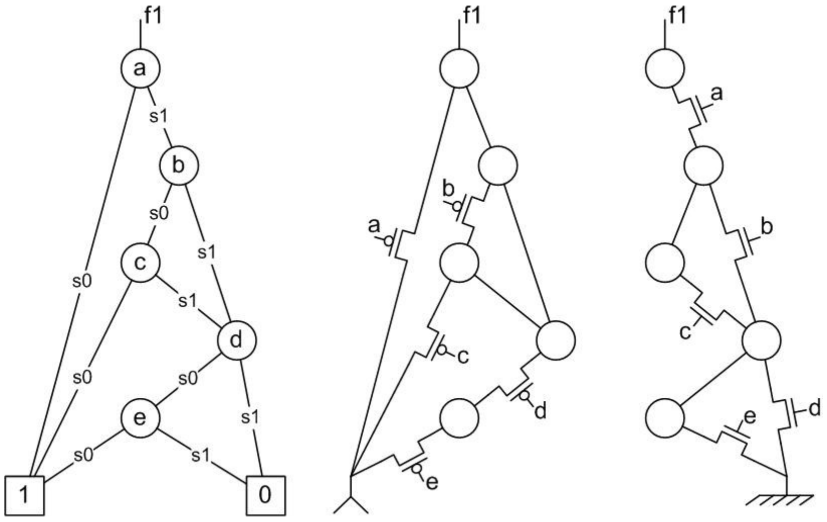


Figure 3.19: Relation among TSBDD edges and transistor networks.

This method also applies the partitioning procedure over an initial DAG to divide the representation in logic cones. Each logic cone is represented by a TSBDD. The algorithm evaluates part of the diagram verifying if they respect the constraint of the virtual library. It uses the number of series transistors for the pull-up and pull-down networks to restrict the size of the virtual library. Each sub-TSBDD that respects the constraints is considered a viable match. Therefore, this sub-set of nodes is replaced by an intermediate node in the original TSBDD. The sub-set of nodes becomes a new TSBDD that is referenced by the intermediate nodes. It is done in a hierarchical data structure that results in a multi-level circuit representation where each instance of a TSBDD corresponds to a technology cell.

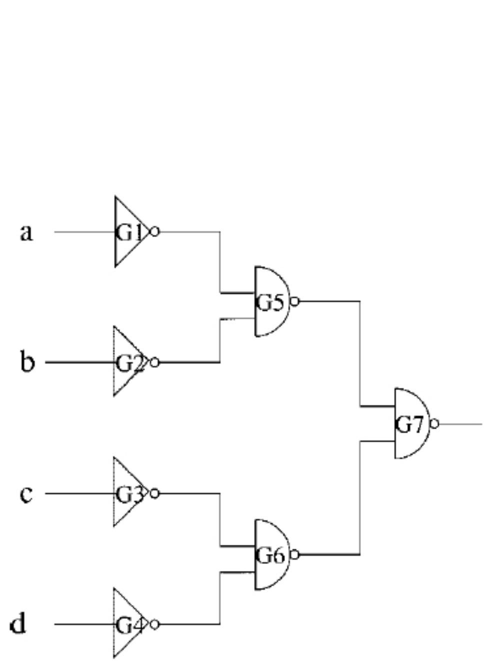
Later, the author observed that there is no advantage in having TSBDDs as subject graphs. By the way, Berkelaar's method does not benefit from using graphs as well; all the operations in Berkelaar's algorithm could be easily implemented on trees. The properties of TSBDDs are kept in tree representations. However, it was one of the first methods to claim the re-ordering of the subject graph aiming better results, mainly, in terms of logic depth.

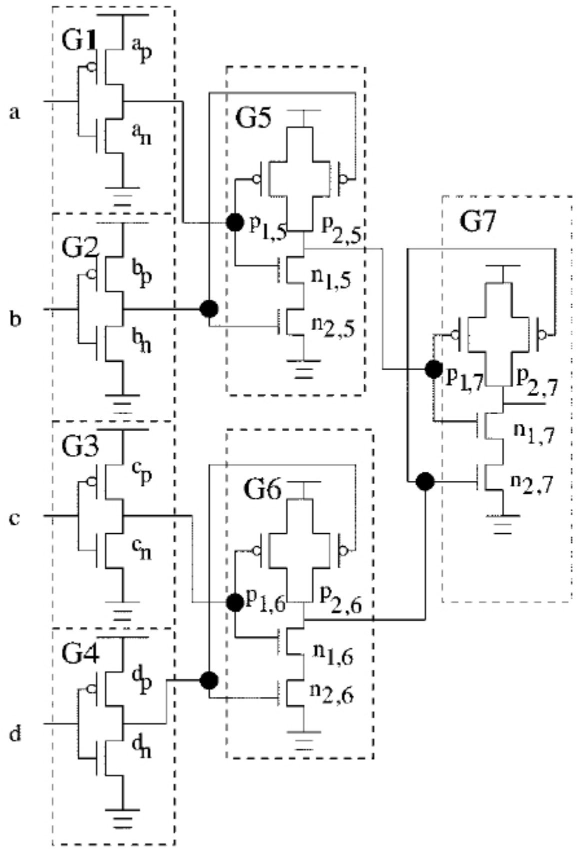
3.5.3 Jiang (2001) – OTR: PTL / CMOS technology mapping

Jiang (2001) introduces two independent algorithms that can be used to map circuits with complex gates using either PTL or CMOS cells. One of the algorithms is called Odd-level Transistor Replacement (OTR). It maps circuits using static CMOS complex gates through a topological gate collapsing method. It also uses the number of series transistors to limit the size of the virtual library. The other algorithm performs a Boolean mapping using BDDs to derive PTL cells. As discussed in the first chapter of this work, PTL cells can be easily derived from BDDs. In the remainder of this section, the OTR method will be reviewed.

The OTR method is fast and simple and avoids the intractable sub-problems in technology mapping, such as matching and covering, by constructing complex gates topologically. The basic idea of the OTR method is to use the pull-down (pull-up) transistor structure from the gates at the previous level gates to replace the pull-up (pull-down) transistors of the gates at the next level. To illustrate this, consider the circuit in Figure 3.20.a consisting of gates *G1* through *G7*. This structure has a total of 20 transistors, and a transistor-level version is shown in Figure 3.20.b. During the procedure of transforming the circuit into a complex gate, we will need to generate intermediate gates as shown in Figure 3.21.a for temporary use. Those intermediate gates will be transformed into a normal static CMOS gate at the end of transformation.

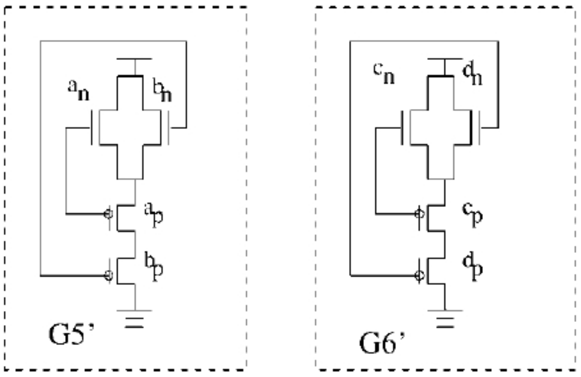
As shown in Figure 3.20.b, we will refer to the pull-down and pull-up transistor in *G1* (*G2*) as a_n and a_{pe} (b_{an} and b_{op}), respectively. The pull-down (pull-up) transistors in *G1* and *G2* are used to replace the fanout pull-up (pull-down) transistors of these gates in *G5* to obtain the gate *G5'*, resulting an intermediate static CMOS gate shown in Figure 3.21.a. For example, the pull-down blocks of *G1* and *G2* fan out to the pull-up transistors *p1;5* and *p2;5* in *G5*, respectively, and hence a_n and b_n are inserted in their place to create *G5'*. Similarly, the transistors in *G3* and *G4* are inserted into *G6* to obtain another intermediate static CMOS gate, *G6'*. These intermediate gates are treated as intermediate synthesis stages and we will eliminate them in the next step by performing the same operation, replacing the pull-down (pull-up) block of *G7* consisting of transistors *p1;7* and *p2;7* (*n1;7* and *n2;7*) by the pull-up (pull-down) blocks of the intermediate gates *G5'* (*G6'*). Therefore, noting that for *G5'*, the pull-up block consists of a_n and b_n and the pull-down block of a_p and b_p, that *G6'* has a pull-up block comprising transistors c_n and d_n and a pull-down block comprising c_p and d_p, the following operations are performed to obtain the final collapsed gate: (1) use a_n and b_n from *G5'* to replace *n1;7* of *G7* (2) use c_n and d_n from *G6'* to replace *n2;7* of *G7* (3) use a_p and b_p from *G5'* to replace *p1;7* of *G7* (4) use c_p and d_p from *G6'* to replace *p2;7* of *G7*. The detailed illustration of the final collapsed gate is shown in Figure 3.21.b. Note that the final implementation has only 8 transistors, a transistor count reduction of 60%.

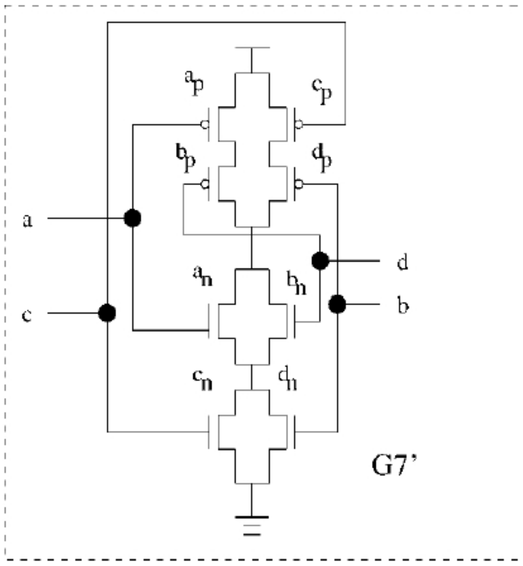




a) Combinational circuit b) Electrical diagram

Figure 3.20: Circuit example (JIANG, 2001).





a) Intermediate cells b) Final circuit

Figure 3.21: Logic cells generated by OTR algorithm (JIANG, 2001).

From the principle illustrated in this example, it is easy to see that if we collapse an even number of levels of gates, we will be left with an intermediate static CMOS gate, whereas if we collapse an odd number of levels, we will return to the formal CMOS complex gate structure, and therefore this technique is called the *odd-level transistor replacement* (OTR) method. The number of levels used in the replacement process depends on the constraints for the maximum number of serial transistors. If the length of the longest transistor chain in a given cell is smaller than the constraint, then the cell can be used as a replacement product in the forward levels.

3.5.4 Correia (2004) – ELIS - Technology mapping for symmetric and asymmetric virtual libraries

The approach presented in Correia (2004) also performs technology mapping over trees. It uses n-ary trees, where nodes can have multiple outgoing edges (more than two child nodes), as subject graph. These trees allow the use of a distinct covering strategy that exploits several decompositions in the same subject description, taking into account criteria that optimize the logic depth of the nodes before gathering them into complex gates.

After the DAG partitioning procedure, each tree is transformed in a n-ary tree, with multiple input AND/OR operators and leaf nodes representing the input literals. There are two basic rules for this transformation. First, the DeMorgan's theorem is applied in the root of a tree to propagate all inverters to the leaf boundary of the tree. This is demonstrated in Figure 3.22. Afterward, every adjacent node with the same operator label is grouped into a single node as illustrated in Figure 3.23.

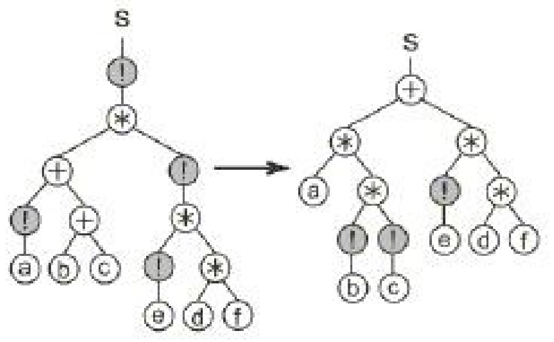


Figure 3.22: Appling the DeMorgan's theorem on n-ary trees (CORREIA, 2004).

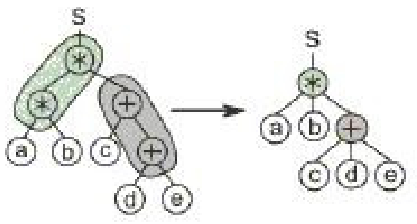


Figure 3.23: Grouping equivalent nodes (CORREIA, 2004).

The n-ary subject trees embed all possible decompositions achieved by the application of associative transformations. As stated before, the initial decomposition of the subject graph has a significant impact in the technology mapping due to the structural bias. The impact of the initial structure can be reduced when different decompositions are explored. Thus, better results can be achieved. Figure 3.24 shows different decompositions for the tree represented by Figure 3.23.

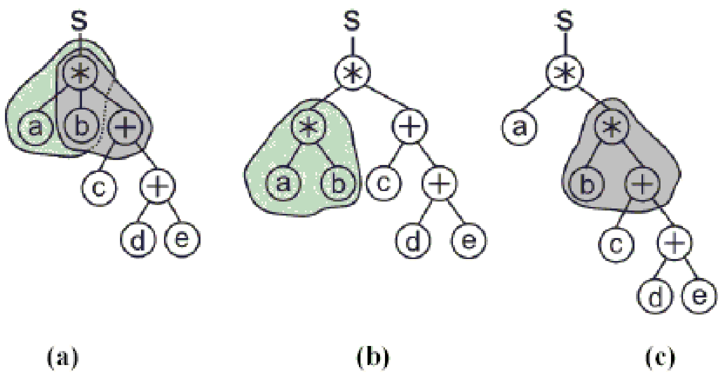


Figure 3.24: Embedded decompositions (b,c) in a n-ary tree (a).

The trees are traversed through a depth-first search procedure, and each node has its depth and (s,p) costs calculated. The series (parallel) cost of an AND (OR) node is given by the sum of all the s (p) costs of its children. The parallel (series) cost of AND (OR) node is given by the highest cost among all de p (s) costs at its children. Leaf-nodes have (s,p) costs equals to (1,1). At each node where the series and parallel costs comply with the specified constraints, a match is established. The covering procedure is bottom-up, following a dynamic programming strategy. As a result, each node, whose children have already being covered, has a precise information about (s, p) costs. The costs for a complete tree are shown in Figure 3.25.a. If a node is found having the (s,p) costs in the maximum value allowed, it is marked as an ideal cut. An example of an ideal cut made at a node for a (2,2) restriction is shown in Figure 3.25.b. The sub-tree rooted at this node is then cut and directly associated to a CMOS complex gate. The cut node at the original tree is replaced by a new leaf node, with associated cost (1,1) and logic depth given by the value of the root of the recently cut tree plus 1. The search then restarts at this node. If a node is found with any of its costs exceeding the limits, it is further sliced. Its set of sons is scanned in terms of costs and depths. Several clusters grouping the nodes with lower depths are considered at this time. A new node with the same operator is added as its son, and this new node receives a set of sons that maximizes the (s,p) costs and minimizes the depth of the new node. The new node is then cut and the cluster directly associated to a CMOS complex gate. The set of nodes removed from the original tree is substituted by a single new leaf node, with associated costs (1,1) and logic depth given by the worst value of this set plus 1. The search restarts then at the node that had some sons removed. If it still has any exceeding cost, the last step is repeated. After each iteration, the costs of the remaining nodes that root subtrees recently modified are recalculated.

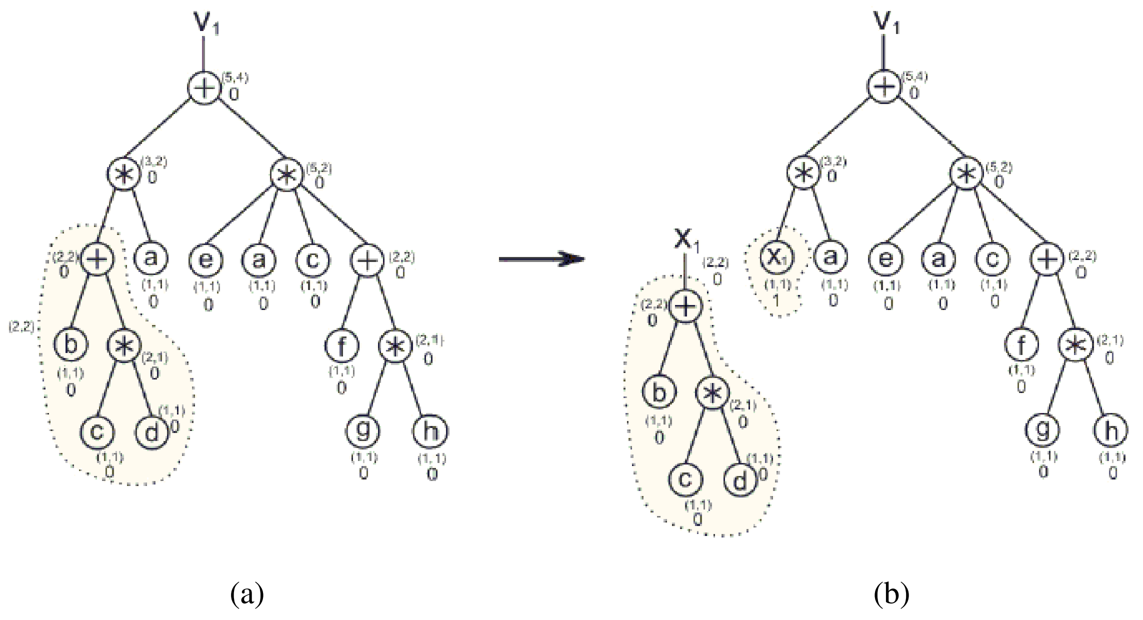


Figure 3.25: Cost calculation and the first cut.

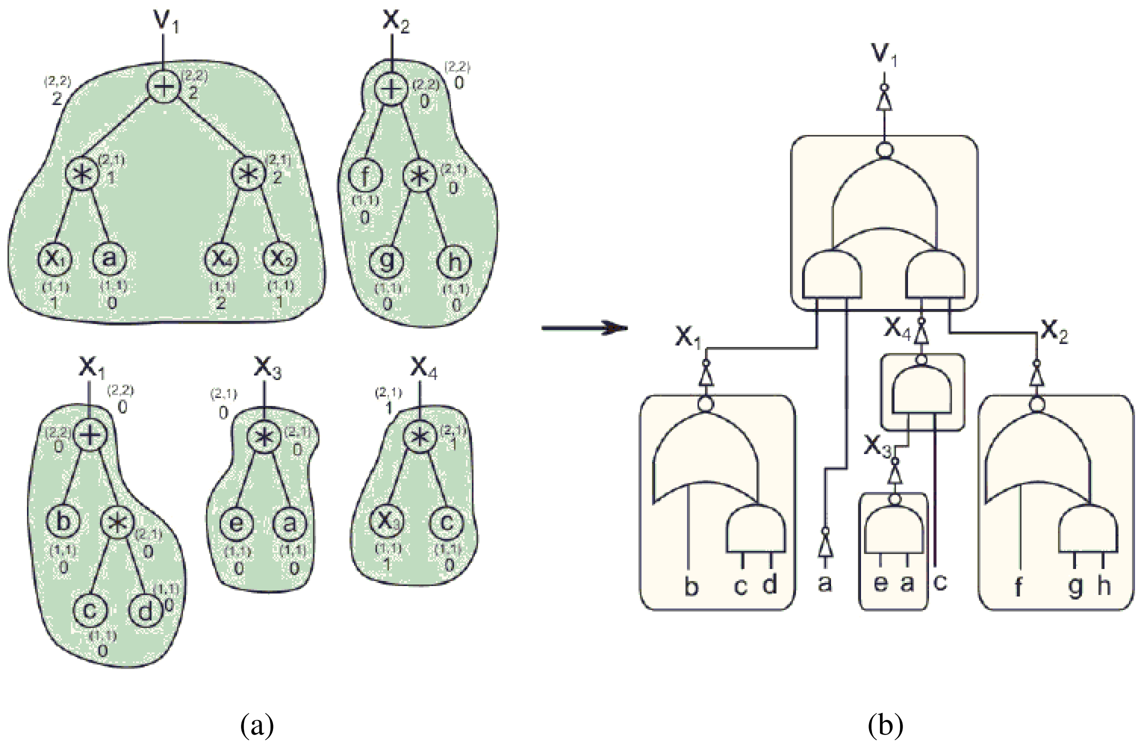


Figure 3.26: Final cover (before inverter minimization).

The main advantage of this method is that it considers dynamically several decompositions of the subject description at low or no cost and covers them optimally in a dynamic programming manner. It finds the optimal cover in linear time. A final cut and its associated implementation is shown in Figure 3.26. After this, a dedicated process will proceed the gate polarization step, where inverters are used where necessary, prioritizing the critical paths. By exploring dynamically several decompositions embedded in the same tree, this method can achieve better results than simply relying on the initial structure provided. This improvement is accomplished at a lower computational cost than barely trying exhausting all the possibilities over numerous instances of the same description. More detailed results were present by Correia (2004). In agreement with Keutzer (1987-b), these results show that rich libraries improve the quality of the mapped circuit. A comparison against SIS technology mapping shows significant improvements in terms of delay. The final area achieved by the Correia's method is slightly bigger than the circuits mapped by SIS.

3.6 Technology mapping methods overview

All technology mapping methods reviewed in the previous sections are summarized in Table 3.2. The second column shows the subject graph used by each method. In column three, we can see three different matching types: structural, topological constraint and structural/Boolean. Structural matching is strongly dependent on the structure of the subject graph. Some methods use a mixed solution of Boolean and structural matching. Although the Boolean matching is slower than structural matching, it usually gives better results. In some cases, it can be used to avoid structural bias. The other matching type is 'topological constraint'. It may be dependent on the structure of a circuit. However, it relies on how the constraints are calculated. The covering strategies of each method are enumerated in the fourth column. Except by the Berkelaar's method, the other methods use a bottom-up covering strategy. The top-down strategy leads to non-optimal results. Finally, column five categorizes the methods according to its dependence on a static library.

Table 3.2: Differences among previous technology mapping methods.

Method	**Subject Graph**	**Matching Type**	**Covering Type**	**Static Library**
Keutzer (1987-a)	Binary tree represented through a string	Structural	Bottom-up	Yes
Berkelaar (1988)	Boolean expression represented by a graph	Topological constraint	Top-down	No
Kukimoto (1998)	DAG	Structural	Bottom-up	Yes
Reis (1998)	TSBDD	Topological constraint	Bottom-up	No
Stok (1999)	DAG	Structural / Boolean	Bottom-up	Yes
Jiang (2001)	Circuit electrical diagram	Topological constraint	Bottom-up	No
Correia (2004)	N-ary tree	Topological constraint	Bottom-up	No
Mishchenko (2005)	AIG	Structural / Boolean	Bottom-up	Yes

4 TECHNOLOGY MAPPING USING CMOS GATES WITH MINIMUM TRANSISTOR STACKS

This chapter introduces a new approach for library-free technology mapping using CMOS gates with minimum transistor stacks (the lower bound cells presented in (SCHNEIDER, 2005)). The available technology mapping methods/tools are analyzed in order to verify how these techniques can handle lower bound cells. Our method for library-free technology mapping, which is called VIRMA ("VIRtual technology MApping tool"), is described in the following. It includes a description of the object function, subject graph and all supporting functions.

4.1 Previous technology mapping techniques and CMOS gates with minimum transistor stacks

The previous chapter presented a quick overview of several technology mapping techniques that are based on different subject graphs and on dynamic or static libraries. In this section, some crucial points concerning lower bound cells are discussed. Not all available technology mapping methods can take advantage of these complex gates, either due to their subject graphs or library limitations. These aspects are discussed in the following.

4.1.1 Cell instances

The exploration of lower bound benefits in the circuit design is strongly attached to the technology mapping procedure, as stated by Rosa (2007).

Lemma 1: every function that can be expressed as a prime irredundant sum of products (ISOP) without repeated literals will have a complementary series-parallel (CSP) implementation with minimum length transistor chains.

Example 1: The cell shown in the left part of Figure 4.1 will have a regular series-parallel implementation which is $PU=2$ and $PD=3$ (2,3). This corresponds to the lower bounds for this cell, implementing a function without repeated literals.

Corollary 1: It is a necessary condition for a cell not to respect the lower bound that it has more than one transistor in the same plane controlled by the same variable (in positive or negative polarity).

Example 2: The cell in the right part of Figure 4.1 will have a regular series-parallel implementation which is (2,3). However, the lower bounds for this function (with repeated literals) are (2,2), and a non-complementary cell is necessary to achieve the lower bounds. This corresponds to a self-dual implementation of a carry-out circuit that is used in practice.

Corollary 2: Respecting the lower bound does not depend on the cell structure itself, but also on the way the cell is instantiated in a circuit. This way, a cell library where all the cells respect the lower bound is not sufficient to have a final circuit where all the cell instances respect the lower bounds, as some instances may not respect the lower bounds if different input pins are externally (to the cell) connected to logically equivalent signals.

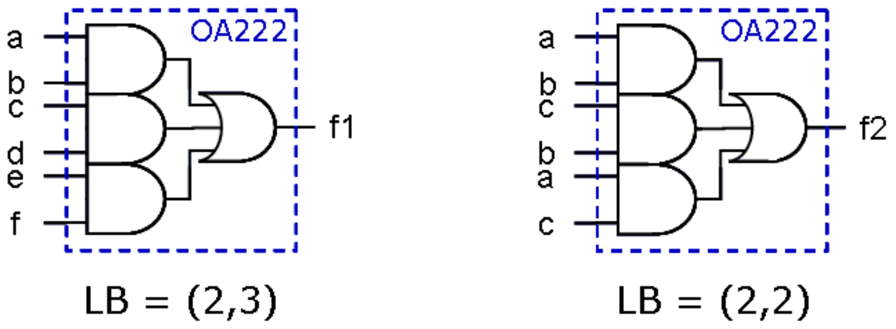


Figure 4.1: Cell instances and the lower bounds.

Example 3: Consider the cell given by the function $c1 = x1 \cdot x2 + x3 \cdot x4 + x5 \cdot x6$. This cell would be available in many commercial libraries under the name OA222. This cell has a regular series-parallel implementation which is (2,3). Now consider two different instances of the same cell as shown in the Figure 4.1. An instance $f1 = a \cdot b + c \cdot d + e \cdot f$ will respect the lower bounds, as the structure of the cell is (2,3) and the lower bounds for this function are also (2,3). An instance $f2 = a \cdot b + b \cdot c + a \cdot c$ will not respect the lower bounds, as the structure of the cell is (2,3) and the lower bounds for the implemented function are (2,2).

Corollary 3: It is not sufficient to look at the instance of a cell to determine if it respects the lower bounds, as logic equivalency among different signals may happen through syntax that is expressed externally to the cell instance, like buffer and inverter connections.

Example 4: Consider the cell given by the function $c1 = x1 \cdot x2 + x3 \cdot x4 + x5 \cdot x6$ and an instance $f1 = a \cdot b + c \cdot d + e \cdot f$. The instance will potentially respect the lower bounds, as the structure of the cell is (2,3) and the lower bounds for this function are also (2,3). However, if some buffers instantiated in the circuit say that $a = e$, $b = d$ and $c = f$, then the function is a (2,2) function. In order to discover this, it is necessary to determine which signals are equivalent through chains of inverters and buffers.

As a general conclusion about cell libraries and the respect to the lower bounds, it is possible to notice the following:

1. All the cells in the *sp.genlib* libraries distributed with SIS (SENTOVICH, 1992) have structures that respect the lower bounds. This is a consequence of these cells not presenting repeated literals. However, when instantiated in a circuit, these cells may have logically equivalent signals connected to

different cell inputs and as a consequence some instances may not respect the lower bounds.

2. Most cells inserted in commercial libraries have a structure that respects the lower bounds. However this does not guarantee instances respecting the lower bounds.
3. If a cell with a structure that does not respect the lower bounds is inserted in a library, none of its instances will respect the lower bounds. However, this is a situation that rarely happens because commercial libraries are usually composed by cells that can be expressed by an equation without repeated literals (except by adders, XORs and memory cells).
4. Any statistics about the number of cells that respect the lower bounds should be divided into two different questions. First question is to look at cells in the library, and see if the cell structures respect the lower bounds. This question involves only the quality of cell networks in the library. Second question is to look at cell instances in a circuit. This question evaluates how the library, the circuit, and the technology mapping fit together. In order to be valid, this second analysis should also be aware of logic equivalency expressed through buffers and inverters.

4.1.2 Tree based and DAG based technology mapping techniques

Another very import point for mapping circuits using lower bound cells are the subject graphs. Some of these representations impose some structural barriers for the mapping methods (structural bias). Most techniques are based on trees and DAGs. Trees are a fanout-free representation. It means that repeated literals can only appear in sub-trees that reach the leaf-nodes (refer Example 3 of the previous section). Moreover, the depth of the trees is generally not too large for many circuits. Therefore, it reduces the number of possibilities for implementing Boolean functions with the lower bounds for serial transistors.

On DAG representations there is path re-convergence. Thus, sub-graphs starting on any node of the DAG can represent functions with repeated literals. It makes this kind of data structure more appropriated for handling lower bound cells.

4.1.3 The computation of series transistors

All existing library-free mapping methods such as (BERKELAAR, 1988) (GAVRILOV, 1997) (REIS, 1997) (CORREIA, 2004) (JIANG, 2005) computes the number of series transistors by series/parallel associations. This is a structural non-Boolean way of computing the number of transistors in series, and it is monotonically increasing.

The series (parallel) association of transistors computes the number of series (parallel) elements of an association as the sum (maximum) of the elements in series (parallel) in the association. Following the notation (series, parallel) or just (s, p), the computation rules are resumed below:

- AND association: $(s, p)\ cost = (sum(s_1,...,s_n), max(p_1,...,p_n))$.
- OR association: $(s, p)\ cost = (max(s_1,...,s_n), sum(p_1,...,p_n))$.
- Inversions: $(s, p)\ cost = (p, s)\ cost$.

The series/parallel association of two logic functions will result in a function with a higher (or equal) number of transistors in series than the functions associated. This is demonstrated through the DAG of the Figure 4.2. The association of the sub-functions *f3* and *f4* results in the sub-function *f6*. The (s,p) cost of *f6* is higher than *f3* and *f4*. The costs are always increasing until the node that represents *f7*.

Schneider (2005) presented a method for Boolean computation of the number of series transistors. The main advantages of this Boolean computation are:

1) The values are minimal.
2) The values depend solely on the candidate function and are univocally defined. They do not depend on sub-functions being associated (it is a non-structural method).

The first affirmation was demonstrated in the second chapter of this thesis using examples of NCSP cells. These cells are constructed from the minimal SOP of the ON-set and OFF-set of a Boolean function. The minimal SOP is achieved through a Boolean method which is based on a modification of the Quine-McCluskey methods. The second affirmation is quite interesting since we can conclude that the Boolean computation is not monotonically increasing. This is also illustrated in the Figure 4.2. While the structural computation gives a (s,p) cost of (4,2) for the sub-function *f6*, the Boolean method gives (3,2), even considering that *f4* has alone a higher cost. This is more apparent for the function *f7* which has cost (2,2).

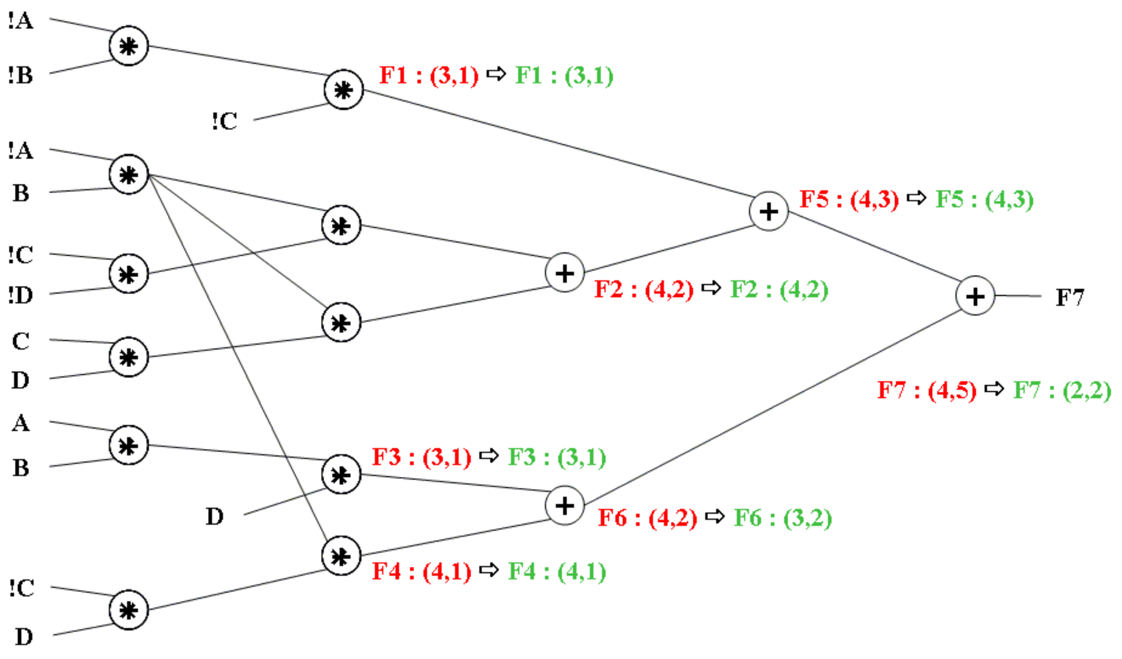


Figure 4.2: Computation of series transistors.

Based on the explanation above, we stress that our technology mapping algorithm depends on a Boolean method for computing the number of series transistors. The structural method is quite simple and its complexity is linear to the number of nodes in the sub-graph. In case of the Boolean method, the complexity is very dependent of the method that searches for the minimal SOPs. It might be a bottleneck for the mapping algorithm since it is called for each match created during the matching phase. Therefore, it must perform a very quick computation.

4.1.4 Contextualizing the problem

The evolution of VLSI circuits is directly related to their manufacturing processes. Accordingly to the Moore's Law, each new generation of the lithography process brings more integration, storage capacity, performance and energy efficiency for VLSI circuits. However, the design of these circuits involving new technologies is very expensive. Many companies would prefer to invest in good EDA tools that explore specific optimizations during the circuit design process. There is a whole search space that is usually not explored in industrial tools. Instead of changing the used technology to obtain more performance, for instance, the designer could count on efficient cells in the library (either virtual or static). This leads to the approach introduced in (SCHNEIDER, 2005), which shows that transistor networks can be constructed using minimal transistor chains through NCSP gates.

As stated in the previous section, there are many occurrences of cells that do not respect the lower bounds on implementing some Boolean functions. None of the previous techniques explores the use of these cells. Recent works proposed automatic techniques to construct transistor networks that respect the lower bounds with respect to the length of series transistor chains (ROSA, 2006). In order to verify how these cells can contribute to implement circuits with better performance, a new technology mapping method is needed.

4.2 VIRMA technology mapping tool

The VIRMA technology mapping tool puts together two concepts: library free technology mapping and lower bound cells. The first prototype to evaluate our method started to be developed early in 2005. It was strongly based on the wavefront technology mapping presented by Stok (1999). Later, a new Boolean technology mapping algorithm was introduced in (MISHCHENKO, 2005). Since this algorithm has a Boolean nature and has presented good results with static libraries, a new version of VIRMA was developed on its principles to explore the potential of virtual libraries. Both methods are described in the remaining of this section, including the objective function and all algorithms/functions needed by the VIRMA algorithms.

4.2.1 Defining the object function

Usually, a precharacterized cell library has accurate information for each designed cell. It includes delay, area, power consumption, etc. The availability of delay information can help timing analysis algorithms, providing a very accurate timing analysis for mapped circuits. Since our technology mapping tool is not based on a pre-characterized library, our cost function uses topological metrics to estimate delay costs. Basically, the number of series transistors of the longest pull-up/pull-down path is used as metric. Even though this is not accurate, the minimization of this number leads to circuits with lower delay.

In our topological model, each cell has its own cost for the pull-up plane and pull-down plane. There are also costs attached to each cell output. They represent the cost for the longest path from a primary input to the output net of the cell. The pull-up cost is calculated by the sum of series transistors in the pull-up plane (SPU) of each cell along the path. The pull-down cost (SPD) is computed similarly using the pull-down plane costs.

The circuit in Figure 4.3 illustrates how the costs are calculated. It is composed by the gates G1 to G5. The set of logic cells that implements the circuit is shown in Figure 4.4. A pair of numbers represents the PU/PD costs of a logic cell. For instance, the gates G1, G2, and G3 have cost 1,2, where PU is 1 and PD is 2. Similarly, a pair of numbers represents the SPU and SPD costs for each cell output. There are three driver cells connected to the multi-source net of the primary output. The cell with the lowest SPU is considered the fastest implementation. When there is more than one cell with the same value for SPU, the cell with the lowest SPD is deemed the fastest implementation. Therefore, the gate G5 is the fastest implementation for the net of the primary output.

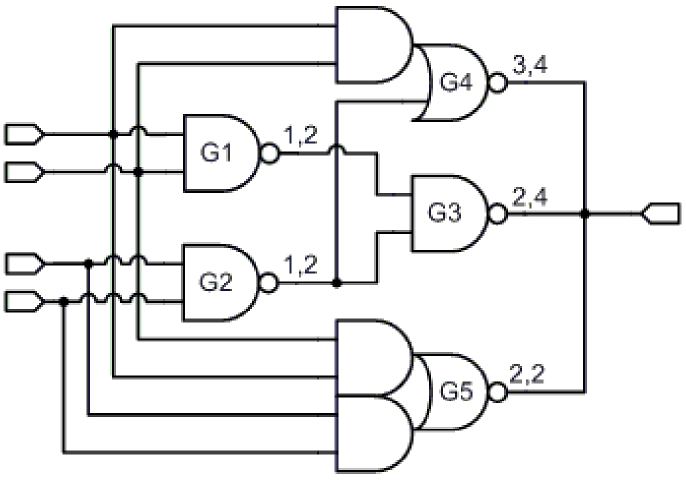


Figure 4.3: Cost function modeling.

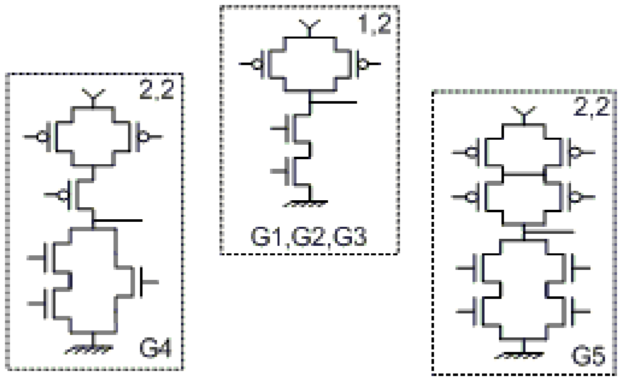
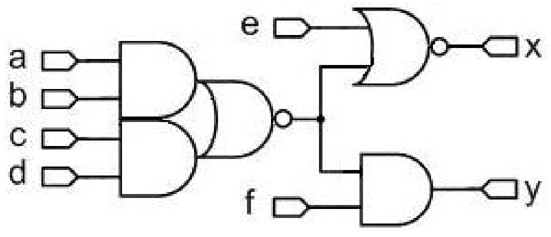


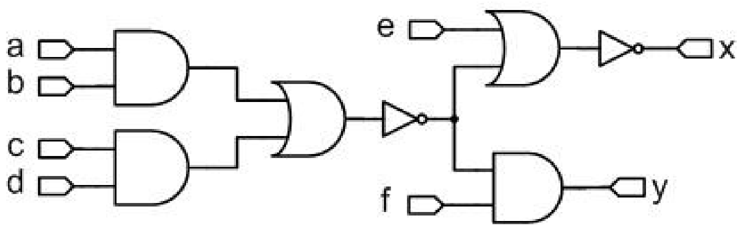
Figure 4.4: A set of logic cells.

4.2.2 Pre-processing procedures

As most of the technology mapping methods, VIRMA needs some pre-processing procedures before starting the matching and covering phases. First, the original circuit representation must be converted to a subject graph. Our algorithm assumes that the initial circuit is decomposed in 2-Input AND/OR gates, in order to increase the granularity of the circuit, to provide more freedom for the matching algorithm in generating arbitrary complex gates. It is represented by a DAG containing only nodes with two outgoing edges. The nodes can be primary inputs or outputs, inverters, and OR and AND operators. This procedure is demonstrated through Figure 4.5.



a) original circuit with complex gates



b) decomposed circuit

Figure 4.5: Creating the subject graph.

Second, all inverter nodes in the subject graph are removed. This technique is used in order to reduce the total number of pattern matches, given that every possible match containing an inverter is discarded. As demonstrated in the Figure 4.6, inversion flags are placed in all incoming edges (circuit nets) connected to inverter nodes. Therefore, the signal phase assignments are preserved.

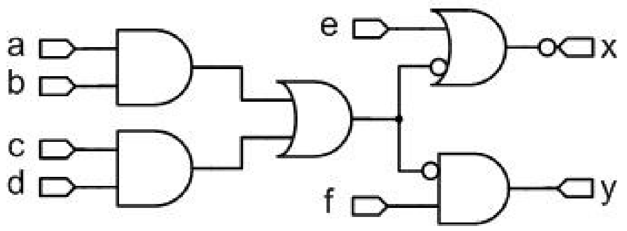


Figure 4.6: Inverter removal and phase assignment.

The last step is the circuit depth calculation. This is done through a depth-first search algorithm. The depths are annotated in the respective nodes. Inverters are not counted in the depth calculation since they are not represented by DAG nodes at this moment (refer to Figure 4.7). The depths are needed for both matching and covering algorithms. Each iteration of these algorithms works on a set of nodes of a given level/depth.

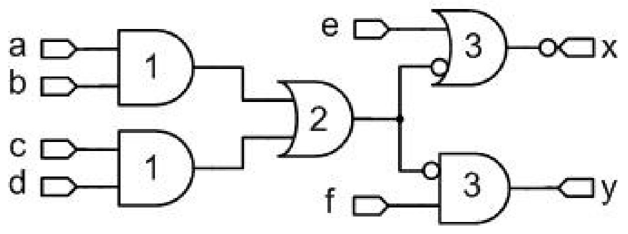


Figure 4.7: Levelizing the subject graph.

4.2.3 Post-processing procedures

Since all inverters are removed before the technology mapping algorithm, a post-processing procedure may be required to ensure the correct phase assignment of all circuit nets. This can be done through a very simple procedure that verifies the polarity of each net of the circuit. The algorithm checks all inversions assignments in a net either for the inputs of the net driven cells or for the output of the driver cell. Figure 4.8 illustrates this procedure. The net *n* of Figure 4.8.a is connected to the cells *X*, *Y*, *G1*, *G2* and *G3* (primary inputs and outputs are also represented by nodes of a DAG). As the output of the gate *G1*, the inputs of the cells *X* and *G2* have inversion assignments. Therefore, the connection of these cells is in the correct phase. Nevertheless, the cells *Y* and *G3* do not have inversion assignments. Thus, an inverter is needed to correct the signal phase. This process results in the circuit of Figure 4.8.b.

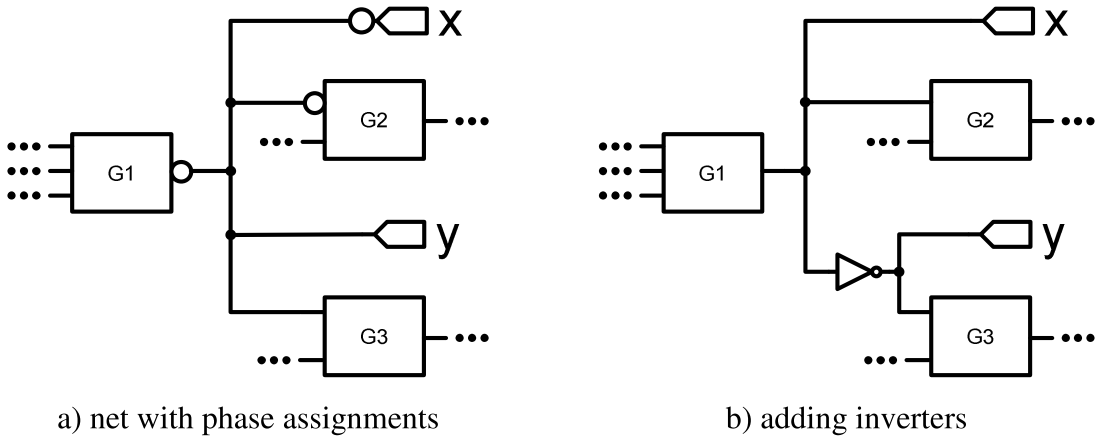


Figure 4.8: Adjusting polarities of the circuit nets.

4.2.4 VIRMA wavefront technology mapping

VIRMA wavefront (VIRMA-WF) technology mapping was derived from the wavefront algorithm presented in (STOK, 1999). The wavefront algorithm leads to a simple implementation and maps optimally for minimal delay on DAGs using a static cell library. Since our approach is based on virtual cell libraries, some small modifications were needed. However, the original characteristics such as the management of the pattern matches and the objective function model for multi-source nets were preserved.

The VIRMA-WF method is outlined in Figure 4.9. As the original wavefront (STOK, 1999), the wavefront is defined as a subgraph of a DAG, such that every path from input to output goes through the subgraph. The subcircuit isolated by the wavefront is bounded by the head and the tail of the wavefront. The matching generation window is given by the *wave_width*. For instance, consider the window shown in Figure 4.10 that has width 3. The head starts at level 0 (primary input nets). It advances level-by-level and the match generation, which is outlined in Figure 4.11. This is done for all nodes on the head level. After the matching generation step is finished for a given node *n*, the covering algorithm (Figure 4.12) is immediately invoked, in order to choose the best match for *n*. These steps will be repeated until the head reaches the highest level in the circuit. Finally, considering the inversion flags in the circuit representation, inverters are inserted when it is necessary.

Algorithm 4.1: Main Algorithm of the VIRMA-WF method.

Input: subject graph (DAG decomposed into AND/OR/NOT primitives), maximum number of series transistors in the PU plane, maximum number of series transistors in the PD plane, *wavefront* window width.

Output: mapped circuit netlist.

```
procedure wavefront_mapping(dag_cir, pu_max, pd_max, wave_width) {
    remove_all_inverters(dag_cir);
    levelize_circuit(dag_cir);
    highest_level = highest_level_of_the_circuit(dag_cir);
    head_level = 1;
    while (head_level ≤ highest_level) {
        head_nets = list_of_all_nets_on_head_level(dag_cir, head_level);
        foreach net, n in head_nets {
            /*Generate all matches considering a set of constraints*/
            generate_matches(dag_cir, n, max_pu, max_pd, head_level, wave_width);
            /*Select the best match for the net n*/
            covering_algorithm(dag_cir, n);
        }
        increment head_level;
    }
    add_inverters(dag_cir);
}
```

Figure 4.9: Main algorithm of VIRMA-WF.

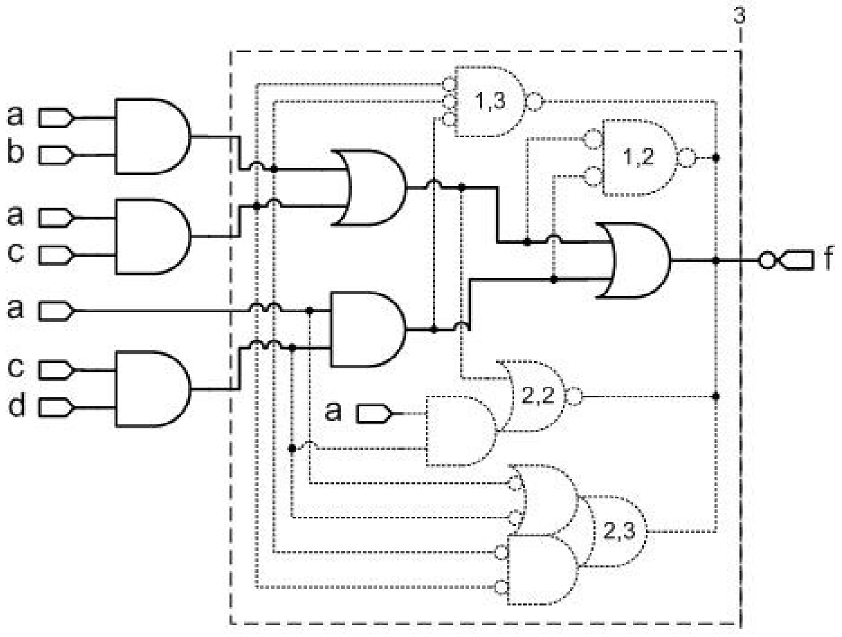


Figure 4.10: Matching generation window.

Algorithm 4.2: Matching generation.

Input: subject graph (DAG decomposed into AND/OR/NOT primitives), reference net, maximum number of series transistors in the PU plane, maximum number of series transistors in the PD plane, the head level, *wavefront* window width.

Output: matches sub-graphs are inserted directly into the DAG representation.

```
1:   procedure generate_matches(dag_cir, n, pu_max, pd_max, head_level, wave_width) {
2:       - In the DAG, generate all the pattern matches for the net n, such that the search for pattern
         matches is performed in the interval [head_level – wave_width : head_level]; At this point,
         other constraints can be used to limit the match generation;
3:       foreach pattern match, pat in the list_of_pattern_matches {
4:           /* Compute lower bound for pull-up and pull-down planes*/
5:           cost_pu = compute_lower_bound_pu(pat);
6:           cost_pd = compute_lower_bound_pd(pat);
7:           if (cost_pu <= max_pu && cost_pd <= max_pd) {
8:               - Store pat as a logic_cell;
9:               - Make logic_cell, a driver of n in the DAG representation (dag_cir);
10:              - Connect all inputs of logic_cell to their correspondent nets;
11:          }
12:      }
13:  }
```

Figure 4.11: Matching algorithm.

Algorithm 4.3: Covering algorithm.

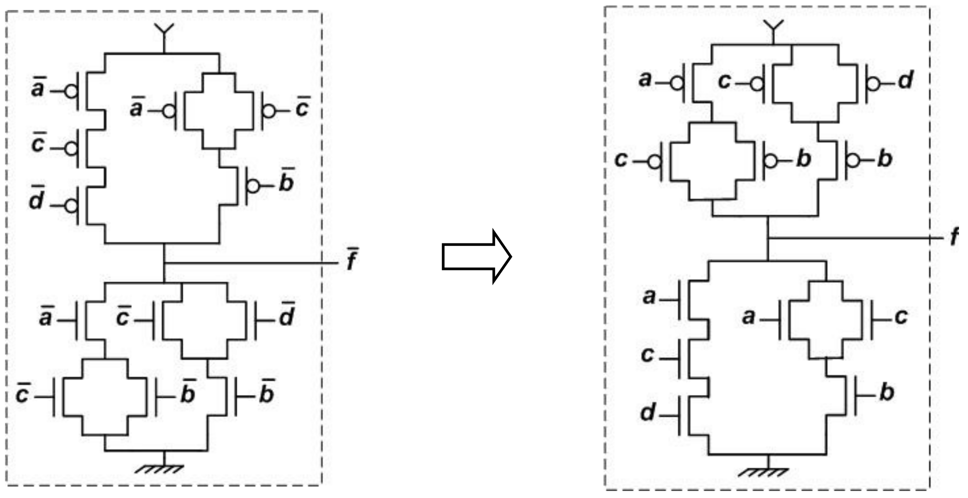
Input: subject graph (DAG decomposed into AND/OR/NOT primitives), reference net.

Output: mapped DAG.

```
1:  procedure covering_algorithm(dag_cir, r) {
2:      - Compute the sum of pull-up for all cells driving net n;
3:      - Compute the sum of pull-down for all cells driving net n;
4:      - Select the cell with the lowest sum of pull-up and pulldown;
5:      - Disconnect all driver cells on n, except the selected cell, and perform a cleanup
6:      operation on their exclusive inputs;
7:  }
```

Figure 4.12: Covering algorithm.

The main difference of the VIRMA-WF method to the original wavefront technology mapping algorithm is found in the matching phase. While the library based algorithm uses structural patterns for library matching, the matching algorithm of the VIRMA-WF is purely Boolean. During the matching generation procedure, the lower bound for the number of series transistors is calculated from BDDs that represent Boolean functions of each match. All matches are enforced to obey the inequality *pull-up_cost* <= *pull-down_cost*. When this condition is false, the inverse function is considered (refer to Figure 4.13). Therefore, the inequality will be respected. Two constraints are used to validate the patterns: *max_pu* and *max_pd*. Both limit the maximum number of series transistors for the pull-up (max_pu) and pull-down (max_pd) chains for each match of a given circuit net. Besides these constraints, another set of restrictions can be used to avoid an excessive number of matches. For instance, the number of variables and/or the number of literals can also limit pattern matches. The matchings are not limited to fanout-free regions; i.e. the match generation search process performs its search across fanout, when the nets are in the wavefront. However, it can be easily limited to fanout-free regions testing the fanout of each net in the search space. This technique can be used in order to save area.



a) *pu_cost > pd_cost*   b) *pu_cost < pd_cost*

Figure 4.13: Function inversion during the mapping process.

The library-free wavefront algorithm is illustrated by Figure 4.15. Assume the following constraint values: *max_pu = 2*, *max_pd = 3* and *wave_width = 3*. The circuit to be mapped is shown in Figure 4.14. The labeled vertical dashed lines represent the head of the wavefront. All matches and their embedding in the network are shown in dashed lines. When the head reaches the level 1, the 2-Input NAND gates are added as drivers of their respective nets in the circuit, creating multi-source nets. After the covering algorithm selects the best match for the last net on level 1, the head is moved to level 2. As the *wave_width* is equal to the highest level of the circuit, on level 2, matches are generated until the primary inputs. The covering algorithm selects the best match for a net, and also disconnects all driver cells, except the selected match, performing a cleanup operation on their exclusive inputs. Figure 4.16.a shows the mapped circuit under these constraints. If the *wave_width* is reduced to 2, the circuit in Figure 4.16.b is obtained. The same circuit is obtained, assuming the following constraints: *max_pu = 1*, *max_pd = 2* and *wave_width = 2*.

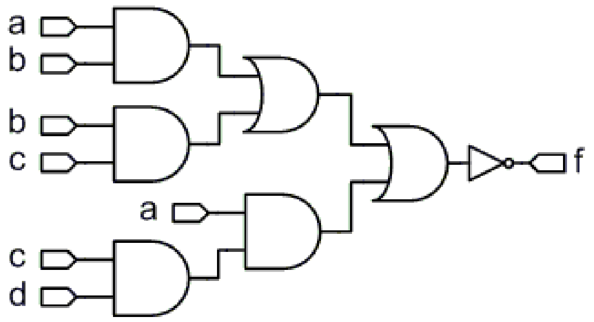


Figure 4.14: Decomposed circuit.

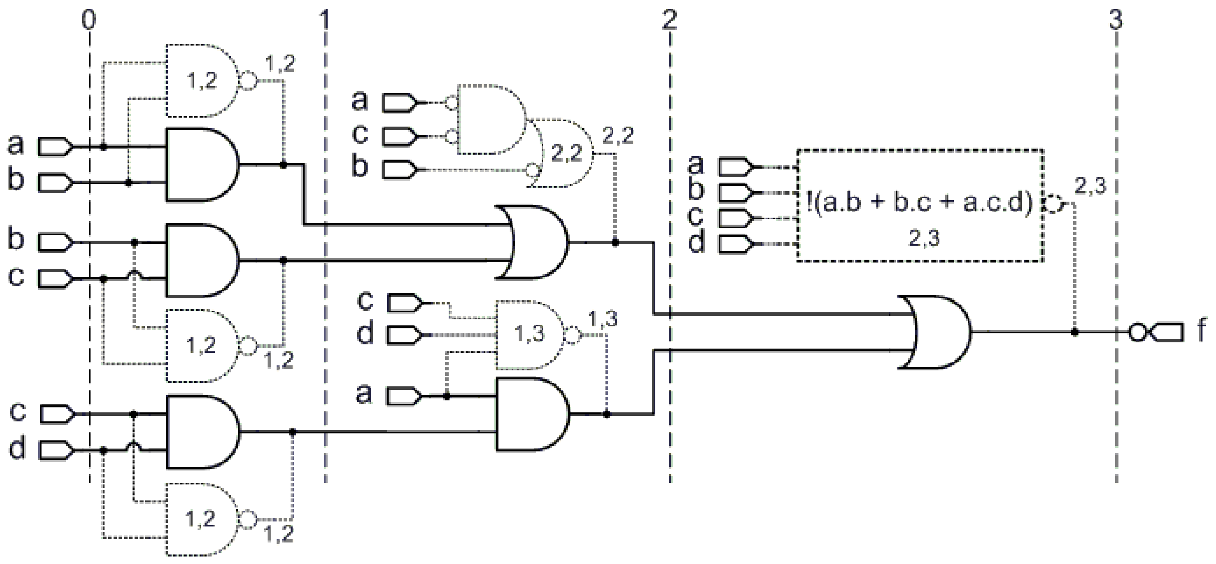
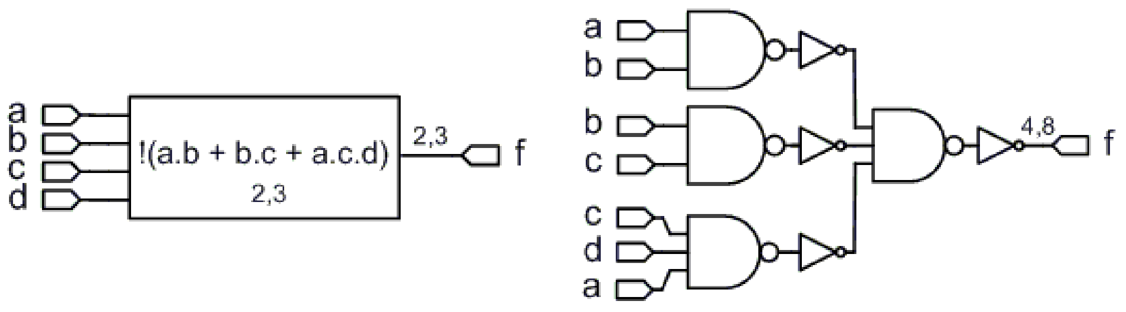


Figure 4.15: Best matches generated by VIRMA-WF.



a) wave width = 3 b) wave width = 2

Figure 4.16: Circuit mapped with VIRMA-WF.

4.2.5 VIRMA and *k*-cuts

Mishchenko (2005) has recently presented an approach for technology mapping based on DAGs was that relies on a simple exhaustive procedure able to enumerate all *k*-feasible cuts where *k* is the number of variables in a sub-function of the circuit. Each *k*-cut is matched against the library using a Boolean matcher.

This algorithm can also be extended to perform technology mapping using virtual libraries. VIRMA-*K*-Cuts algorithm, which uses k-cuts, is introduced in this section. It uses the same subject graph as the VIRMA-WF method. Before starting the technology mapping algorithm the subject graph is also pre-processed. After the mapping procedure, it also performs the correction of the net phase assignments. The difference of Mishchenko's method and VIRMA-*K*-cuts is in the Boolean matcher. Instead of matching every *k*-cut against a library of patterns, the lower bounds for the number of series transistors are calculated through a BDD-based algorithm. If the lower bounds for pull-up and pull-down networks respect the constraints of the virtual library, then it is considered a valid match. As an example, consider the *k*-cuts of Figure 4.17. The whole circuit is covered by a *k*-cut of 4 variables. This cut is a Boolean equivalent to the 4-input XOR gates presented in Figure 4.18. The implementation of this gate in the regular CMOS CSP is prohibitive since it has eight serial transistors in the longest pull-down path. However, it can be implemented with a NCSP cell that does not have more than four series transistors in any of the network planes.

In library-based approaches that use structural matchers, each sub-graph rooted by a given node is matched against each pattern of the library. Although the number of possible matches is directly related to the library size, typically, it binds few patterns in the root node. It results in less CPU time to compute the accumulated match costs. Even in library-based approaches that use Boolean matchers, the number of attached matches in a node is not so high. Accordingly to Mishchenko (2005), when computing all 5-feasible cuts, the number of matches attached to each node varies from 20 to 30 matches in many practical cases.

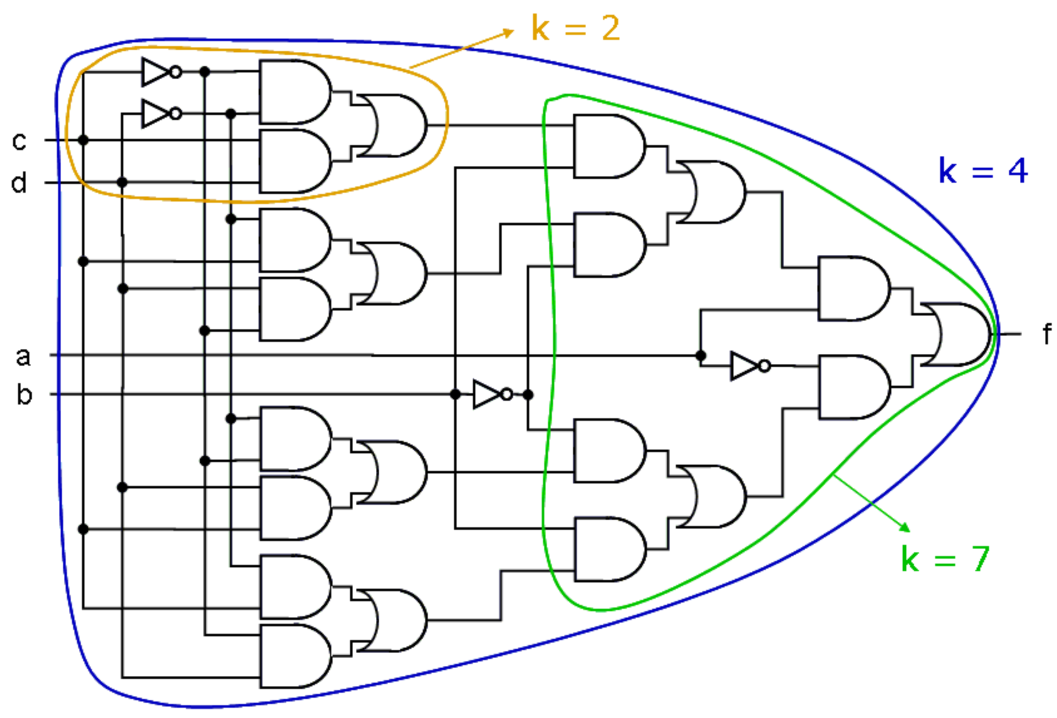
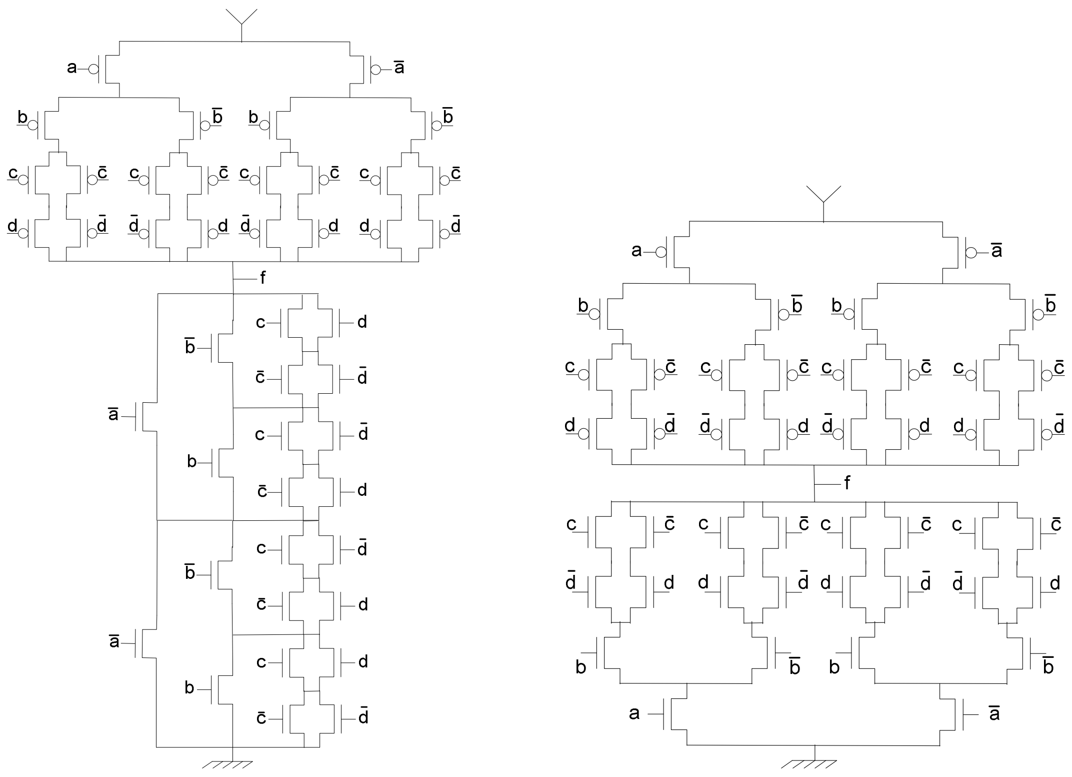


Figure 4.17: *K*-cut examples.



a) CMOS CSP b) NCSP

Figure 4.18: 4-input XOR gates.

Unlike the traditional library based approaches, the VIRMA-WF has to compute all structural combinations of adjacent nodes inside the wavefront window. Since the lower bound cost is not monotonically increasing, it has to compute all combinations. This number grows exponentially by increasing the wave width, which is demonstrated through Figure 4.19. In order to find the 4-input XOR match, a wavefront with width 6 must be used. This way, the algorithm will compute 1294 combinations of candidate matches. The VIRMA-*K*-Cuts method would find less than 30 candidate matches by computing all 6-feaseble cuts. This is one of the main advantages of the *k*-cuts method.

The computation of all feasible *k*-cuts has limitations. The exhaustive enumeration of k-cuts is limited to 6 variables in most of the practical cases. Chatterjee (2006) proposed a non exhaustive heuristic method to compute *k*-cuts up to 12 variables. This heuristic was not yet incorporated in the VIRMA-*K*-cuts approach.

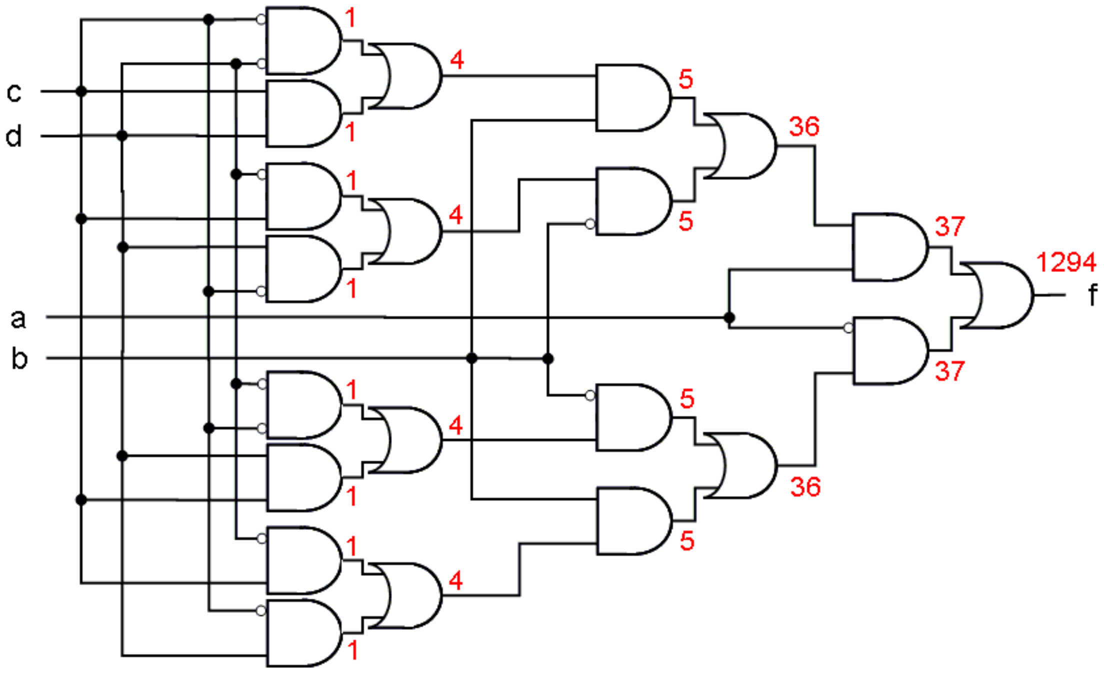


Figure 4.19: Computing the number of possible structural combinations.

4.3 Final considerations

Both VIRMA-WF and VIRMA-*K*-Cuts methods can be extended to support other features. Since it is not restricted to fanout-free regions, the area can grow fast during the technology mapping. In order to find a good trade-off between area and delay, the algorithm can be limited to map fanout-free graphs in regions that are not critical in delay. Therefore, the DAG mapping approach would be used only in critical regions.

In (STOK, 1999), another extension based on a two-pass algorithm was proposed to optimize other cost functions. Instead of keeping only the best match for each network, one or more other matches are kept in the first pass. In a second pass, from the primary output to the primary inputs, the slower (smaller) patterns can be chosen in the off-critical regions to minimize area.

In addition, this strategy can be used to perform a mixed technology mapping using different topologies for logic designs. The works presented in (YAMASHITA, 1997) and (JIANG, 2001) show that mixed designs using PTL and static CMOS cells can have better area/delay/power than circuits realized with only one logic style. The transistor network generator proposed in (ROSA, 2006) is able to generate PTL cells, and other different topologies as well. Thus, the two-pass approach can be used to realize mixed technology mapping using PTL and NCSP cells. Following the idea of keeping more than one match for each net in the circuit, extra matches can be generated considering different topologies. Hence, matches implemented by PTL cells will be available for the second pass algorithm.

5 EXPERIMENTS

This chapter presents different sets of results to validate the concepts proposed in this thesis. The first set of results contemplates an analysis done over the VIRMA-WF method against SIS (SENTOVICH, 1992). These results were presented in (MARQUES, 2007). A more complete analysis of the VIRMA methods is done by comparisons against the academic *state-of-art* tool, ABC (ABC, 2008) that incorporates the method presented by Mishchenko (2005), as well as by comparing VIRMA against two industrial technology mapping tools.

5.1 Comparisons between SIS and VIRMA-WF

In this section, results for a sub-set of seven ISCAS'85 benchmark circuits are presented, comparing the technology mapping performed by SIS (SENTOVICH, 1992) and VIRMA-WF (MARQUES, 2007). The algorithm prototype was developed in Java. All results were generated on a PC workstation running Windows XP using an AMD Athlon 64/3200+ processor.

For this experiment, the circuits were first decomposed into inverters and 2-Input NAND/NOR gates using SIS. Next, we performed technology mapping, using SIS and our method, for all benchmark circuits. Finally, using our cell generator, NCSP and CSP CMOS transistor networks are derived for the mapped circuits produced by our method and by SIS, respectively. Results are shown in seven different tables. In Tables 4.1 to 4.7, the first columns show the name of the circuit. The labels of the following columns describe the cell libraries used during the technology mapping. The columns *33-4* and *lib2* show results for the cell libraries *33-4.genlib* and *lib2.genlib*, respectively, mapped by SIS targeting minimum delay. Since the cell libraries *33-4.genlib* and *lib2.genlib* have cells with costs equal or lesser than 3 for the pull-up and pull-down planes, and few cells where PU or PD cost can be 4 (such as cells found in the *lib2.genlib*), all circuits were mapped by our method using *wave_width = 4* and the constraints *3,3* and *3,4* to limit PU and PD costs. The label T (e.g. (3,3)-T and (3,4)-T) indicates that the mapping was limited to trees (fanout free regions). The label D (e.g. (3,3)-D and (3,4)-D) indicates that the mapping was allowed to duplicate logic, resulting on DAG mapping.

Table 4.1 shows the accumulated sum of series transistors on the pull-up and pull-down planes of each cell on the longest path of the circuit. The longest path was found following the same criteria of the covering algorithm that is based on SPU and SPD costs. It is noticeable that VIRMA-WF reduces the accumulated transistor chains along the longest path. In order to prove that the reduction of transistor in the pull-up and pull-down planes can reduce the circuit delay, we used SPICE simulation to estimate delay. This way, the ten most critical paths (considering SPU and SPD costs) were extracted from each circuit and were individually simulated in order to find the worst path delay. The transistors used on the SPICE description have fixed size using a technology model 130 nanometers. Table 4.2 presents a delay comparison between SIS technology mapping and VIRMA-WF technology mapping. The second column (*33-4 (ns)*) shows delay values expressed in nanoseconds for circuits mapped by SIS. The columns 3-7 show normalized values corresponding to the delay values of the second column. VIRMA-WF method provides better results than SIS results, with average delay reductions of about 27% and 33% considering virtual cell libraries restricted by the constraints *3,3* and *3,4*, respectively. The technology mapping limited to trees (all column labels tagged with -T) performed by our method also shows improvements of 13%-15% in average. These gains demonstrate the effectiveness of having Boolean matching combined with a virtual library that is able to use lower bound cells.

Table 4.1: Pull-up and pull-down sums in the longest path.

	SIS				**VIRMA-WF**							
circuit	**33-4**		**lib2**		**(3,3)-T**		**(3,3)-D**		**(3,4)-T**		**(3,4)-D**	
	SPU	**SPD**	**SPU**	**SPD**	**SPU**	**SPD**	**SPU**	**SPD**	**SPU**	**SPD**	**SPU**	**SPD**
c1355	31	52	31	55	34	45	25	30	31	46	19	27
c1908	41	58	40	56	39	52	30	38	37	50	27	40
c3540	60	78	54	69	42	67	39	60	39	69	33	52
c432	39	55	34	52	33	49	31	43	25	47	24	41
c499	29	35	24	35	27	32	25	27	24	33	18	26
c6288	124	243	131	247	153	213	95	113	153	213	88	125
c880	32	41	21	39	26	35	21	32	20	36	16	38

Table 4.2: Delay comparisons between SIS and VIRMA-WF.

	SIS		**VIRMA-WF**			
circuit	**33-4 (ns)**	**lib2**	**(3,3)-T**	**(3,3)-D**	**(3,4)-T**	**(3,4)-D**
c1355	2,3205	1,07	0,94	0,61	0,93	0,52
c1908	2,5334	1,08	0,87	0,65	0,87	0,71
c3540	3,3702	1,04	0,94	0,90	0,91	0,65
c432	2,7021	0,88	0,82	0,76	0,79	0,70
c499	1,8734	0,91	0,82	0,71	0,80	0,64
c880	2,2671	0,91	0,83	0,77	0,82	0,80
average		**0,98**	**0,87**	**0,73**	**0,85**	**0,67**

Area comparison, considering the number of transistors of each circuit, can be seen in Table 4.3. The absolute values are shown in the column *33-4*. All other values are relative to those values. Due to logic duplications during technology mapping, which is inherent to DAG mapping, our method may increase the area. The area penalty for using our technology mapping algorithm is 18% and 31% in average for the virtual cell libraries *3,3* and *3,4*, respectively. There are cases where the average area increase is negligible. It happens for the technology mapping limited by fanout. Although for these cases the delay gains were not maximized, a good area/delay trade-off is still achieved.

Table 4.4 shows the execution times for SIS and VIRMA-WF, given in seconds. VIRMA-WF is clearly more time consuming than SIS. As the obtained execution times show, they are not proportional to the size of the circuit. For instance, considering the virtual cell library (3,4), the circuit c499 uses more time than the circuit c3540. However, c499 is smaller than c3540. This is mainly due to the complexity of the lower bound calculus for each match, and also to the number of generated matches during the technology mapping process.

Table 4.3: Area comparisons between SIS and VIRMA-WF.

	SIS		**VIRMA-WF**			
circuit	**33-4**	**lib2**	**(3,3)-T**	**(3,3)-D**	**(3,4)-T**	**(3,4)-D**
c1355	2140	1,03	0,92	0,87	0,89	1,15
c1908	2390	1,01	1,02	1,19	0,94	1,11
c3540	4410	1,09	0,99	1,32	1,05	1,38
c432	790	1,13	1,13	1,37	1,08	1,39
c499	1484	1,00	1,04	1,00	1,00	1,26
c880	1256	1,00	1,08	1,33	1,03	1,54
average		**1,05**	**1,03**	**1,18**	**1,00**	**1,31**

Table 4.4: SIS and VIRMA-WF runtime.

	SIS		**VIRMA-WF**			
circuit	**33-4**	**lib2**	**(3,3)-T**	**(3,3)-D**	**(3,4)-T**	**(3,4)-D**
c1355	0,4	0,3	13,0	494,1	27,8	542,4
c1908	0,6	0,5	15,3	404,7	19,1	766,3
c3540	2,9	2,6	41,3	469,2	44,6	777,7
c432	0,2	0,1	5,5	25,6	6,3	36,0
c499	0,3	0,3	12,7	992,8	27,3	1641,3
c880	0,3	0,2	1,7	93,0	2,1	281,2
total (s)	**4,7**	**4,0**	**89,5**	**2479,3**	**127,1**	**4044,9**

Table 4.5 shows results for the benchmark circuit c6288. For delay and area results, the column *'33-4'* presents, respectively, absolute values in nanoseconds and number of transistors. The following columns show the correspondent relative values. The CPU time is expressed in seconds for all libraries. Delay gains are very significant for the libraries *(3,3)* and *(3,4)*, when DAG mapping is applied. However, the area penalty is high. The c6288 is a multiplier composed by regular logic blocks, and it has several regions that are not fanout-free. Therefore, best matches that cross fanout, will probably be best matches for other regions, resulting in many duplications. This area penalty can be reduced by allowing duplication of logic only for timing critical regions.

Table 4.5: C6288 circuit results.

	33-4 (ns)	lib2	(3,3)-T	(3,3)	(3,4)-T	(3,4)
delay	10.20	1.02	1.00	0.57	1.00	0.63
area	9444	1.01	1.00	1.90	1.00	2.07
CPU time (s)	9.0	8.6	7.6	1028.4	8.0	1304.0

The prototype implemented to obtain the experimental results is devoted to prove our concepts. The results show considerable delay gains. Nevertheless, area results show that better area/delay trade-offs have to be found. The area increase can be better controlled allowing duplication only in critical regions of the circuit.

Another experiment uses a very naive approach to identify critical regions in a circuit. It is based on path slack computation using the logic level of each gate. Consider the sub-circuit of Figure 5.1. The values in the inputs correspond to the logic level of previous gates. It shows the logic level of each gate propagating the values from the inputs to the outputs. Figure 5.2 illustrates the reverse level computation. It assumes that the outputs are at level zero. In this case, the values are propagated from the outputs to inputs. This way, each gate has two associated values: the forward level and the backward level. The slack of each circuit net can be computed using the formula expressed by Equation 5.1 that takes the level of the net driver gate and the reverse level of a driven gate. Figure 5.3 shows the slack computation of each net of the circuit. Assuming that the critical paths are the ones with *slack = 1*. In this scenario, there are only two critical paths in the circuit that are highlighted in the figure. The remaining of the logic is not timing critical.

$$slack = max\_circuit\_level - (driver\_forward\_level + driven\_backward\_level) \qquad (5.1)$$

Using this approach to identify critical regions, different mappings were made, considering different slacks to determine the criticality. The results are shown in Tables 4.6 and 4.7. The columns 5 to 8 that are tagged with '-T-D' show the obtained results for the mixed mapping strategy. Assuming that the critical slack is one, only nets with slack one are considered critical. Thus, the DAG mapping strategy is applied, allowing logic duplication. Non-critical regions are mapped using the tree mapping methodology (which does not allow logic duplications over multiple fanout points). The higher is the critical slack, the larger is the amount of logic duplications in the circuit.

Table 4.6 demonstrates the area effect by using this strategy considering different slack values to determine criticality of paths. When the area results are compared to the estimated delay results of Table 4.7, it clearly seems to be a good strategy to circumvent the area increasing problem. It can also decrease the CPU time, since the number of matches will be reduced. Another possibility to reduce CPU time is to store pre-computed lower bounds in a hash table, to avoid repeated computations.

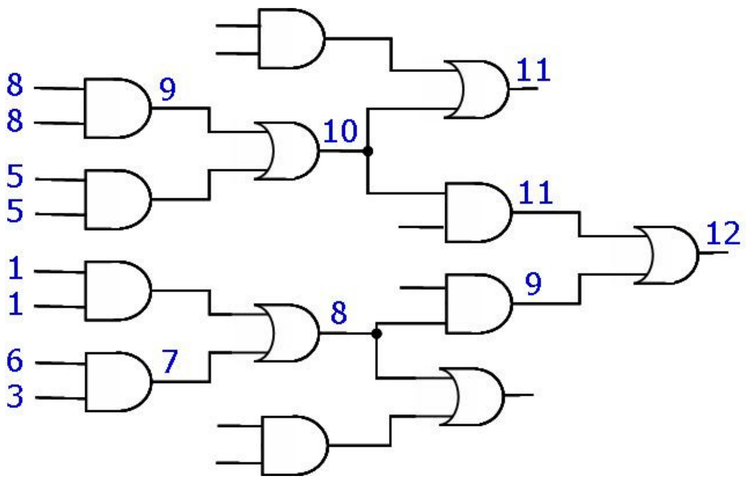


Figure 5.1: Computing forward node levels (from inputs to outputs).

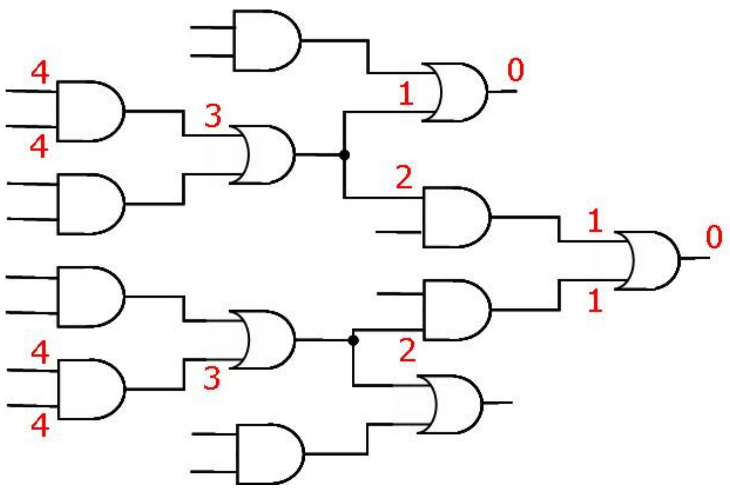


Figure 5.2: Computing backward node levels (from outputs to inputs).

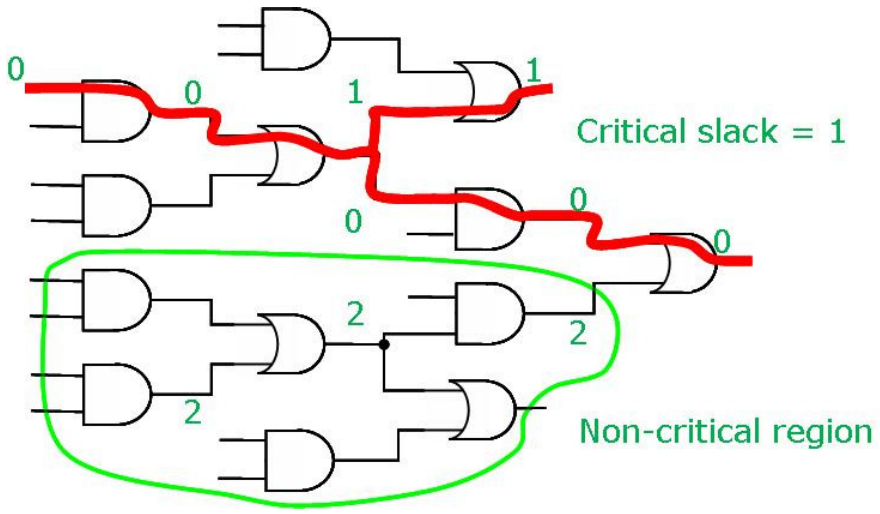


Figure 5.3: Identifying timing critical regions.

Table 4.6: Area comparison – Area saving heuristic.

	SIS		VIRMA-WF					
circuit	**33-4**	**lib2**	**(3,4)-T**	**(3,4)-T-D slack 1**	**(3,4)-T-D slack 2**	**(3,4)-T-D slack 3**	**(3,4)-T-D slack 4**	**(3,4)-D**
c1355	2140	1,03	0,89	0,87	1,12	1,12	1,12	1,15
c1908	2390	1,01	0,94	0,96	1,10	1,07	1,07	1,11
c3540	4410	1,09	1,05	1,05	1,06	1,14	1,14	1,38
c432	790	1,13	1,08	1,20	1,41	1,41	1,41	1,39
c499	1484	1,00	1,00	1,22	1,22	1,22	1,22	1,26
c6288	9444	1,01	1,00	1,04	1,15	1,33	1,33	2,07
c880	1256	1,00	1,03	1,10	1,15	1,17	1,17	1,54
average		**1,04**	**1,00**	**1,06**	**1,17**	**1,21**	**1,21**	**1,42**

Table 4.7: Delay comparison – Area saving heuristic.

	SIS		VIRMA-WF					
circuit	**33-4 (ns)**	**lib2**	**(3,4)-T**	**(3,4)-T-D slack 1**	**(3,4)-T-D slack 2**	**(3,4)-T-D slack 3**	**(3,4)-T-D slack 4**	**(3,4)-D**
c1355	2,32	1,07	0,93	0,57	0,57	0,57	0,57	0,52
c1908	2,53	1,08	0,87	0,75	0,73	0,71	0,71	0,71
c3540	3,37	1,04	0,91	0,85	0,85	0,75	0,75	0,65
c432	2,70	0,88	0,79	0,79	0,70	0,70	0,70	0,70
c499	1,87	0,91	0,80	0,64	0,64	0,64	0,64	0,64
c6288	10,20	1,02	1,01	0,90	0,88	0,86	0,86	0,63
c880	2,27	0,91	0,82	0,77	0,77	0,77	0,77	0,80
average		**0,99**	**0,88**	**0,76**	**0,74**	**0,72**	**0,72**	**0,66**

5.2 Comparisons between ABC and VIRMA

This section shows comparisons of results obtained by ABC tool (ABC, 2008) that implements the algorithm proposed by Mishchenko (2005) and from both versions of the VIRMA methods: VIRMA-WF and VIRMA-*K*-Cuts. All circuit netlists used for comparisons came from the same starting point as the previous experiment; i.e., all circuits were decomposed in 2-inputs AND/OR gates and balanced to reduce the logic depth. After this, they were mapped using ABC and VIRMA for libraries containing cells up to 4 serial transistors in both pull-up and pull-down planes.

Table 4.8 shows the accumulated sum of series transistors on the pull-up and pull-down planes of each cell on the longest path of the circuit. Both versions of VIRMA methods achieved similar results. The sum series transistors is generally much smaller in VIRMA methods than ABC method. Table 4.9 presents a delay comparison among ABC and VIRMA technology mapping methods. The second column shows delay values expressed in nanoseconds for circuits mapped by ABC. The columns 3-4 show normalized values corresponding to the delay values of the second column. VIRMA-*K*-Cuts method provides better results than VIRMA-WF method. However, in most of the cases, the ABC presented slightly better results than both. VIRMA gains (up to 29%) in the c6288 benchmark (that is a multiplier) are very reasonable. The VIRMA methods handle better with XOR gates. XORs up to 4 inputs can be implemented using lower bound cells with cost (4,4), while ABC does not see these cells available in the *44-6.genlib*.

Table 4.8: Pull-up and pull-down sums in the longest path.

	ABC		**VIRMA-WF**		**VIRMA-*K*-Cuts**	
	44-6		**(4,4)**		**(4,4)**	
circuit	**SPU**	**SPD**	**SPU**	**SPD**	**SPU**	**SPD**
c1355	21	28	16	25	16	25
c1908	30	41	24	41	24	43
c3540	42	49	32	49	33	52
c432	25	38	19	34	22	36
c499	21	26	16	25	16	25
c6288	110	127	83	93	81	94

Table 4.9: Delay comparisons between ABC and VIRMA.

	ABC	**VIRMA-WF**	**VIRMA-*K*-Cuts**
circuit	**44-6 (ns)**	**(4,4)**	**(4,4)**
c1355	1,54	1,05	1,00
c1908	2,28	1,03	1,00
c3540	2,84	1,15	1,06
c432	1,86	1,26	1,06
c499	1,51	1,08	1,03
c6288	7,79	0,74	0,71
average		**1,05**	**0,98**

Table 4.10 demonstrates area comparisons in terms of number of transistors. In the average case, VIRMA-WF method is 19% better than ABC, while VIRMA-*K*-Cuts method is 21% better than ABC. ABC mapping results in a considerable amount of logic duplication. This is the result of the dual-rail assignments that are made to achieve better delay results. There are area recovery heuristics implemented in ABC environment. In order to make an appropriated comparison, one iteration of the area recovery heuristic was performed. Area recovering heuristics can also be applied in VIRMA methods as suggested in the fourth chapter (algorithm extensions).

Table 4.11 shows the number of instances of lower bound cells in circuits mapped by VIRMA methods. The matching algorithm finds much more sub-functions with repeated literals in DAG representations. Hence, the lower bound cells occur more frequently when a DAG is used as subject graph.

Table 4.10: Area comparisons between ABC and VIRMA.

	ABC	**VIRMA-WF**	**VIRMA-*K*-Cuts**
circuit	**44-6**	**(4,4)**	**(4,4)**
c1355	3660	0,75	0,77
c1908	2894	0,74	0,72
c3540	5824	0,94	0,80
c432	1108	1,07	0,87
c499	3368	0,83	0,84
c6288	17256	0,53	0,73
average		**0,81**	**0,79**

Table 4.11: Number of instances of lower bound cells.

	VIRMA-WF								**VIRMA-*K*-Cuts**	
	(3,3)-T		**(3,3)-D**		**(4,4)-T**		**(4,4)-D**		**(4,4)-D**	
circuit	**#cells**	**#LB cells**	**#cells**	**#LB cells**	**#cells**	**#LB cells**	**#cells**	**#LB cells**	**#cells**	**#LB cells**
c1355	280	0	349	16	276	0	420	8	376	0
c1908	358	2	343	25	339	2	344	22	314	39
c3540	937	39	1069	64	919	11	958	61	951	47
c432	203	4	206	6	183	4	210	16	205	1
c499	275	0	272	16	276	0	420	8	345	0
c6288	3618	0	1268	414	3618	0	1206	415	1921	593

5.3 Comparisons between commercial tools and VIRMA

This section shows comparisons of results obtained by VIRMA, ABC and two commercial tools. We have created a simple framework that integrates four different methodologies for technology mapping into the synthesis flow of a commercial tool.

The framework is demonstrated in Figure 5.4. It is an iterative flow that performs gate sizing and buffer insertion to meet a timing constraint. One of the inputs of the framework is a mapped design described in the Verilog format (VERILOG DOT COM, 2008) that only instantiates the smallest version of each referred cell (usually, cells named with the suffix 'X1'). The other input file is a pre-characterized library described in the Liberty format (SYNOPSYS LIBERTY LIBRARY FORMAT, 2008). Each of the iterations performs gate sizing and buffer insertion preserving the previous mapping. The sizing algorithm is allowed to change the drive strength of a cell instance, but not the cell function, preserving the cell boundaries. After the sizing procedure, a Static Timing Analysis (STA) is performed in order to measure the circuit delay. Thus, the currently delay value, decreased in 5%, will be the timing constraint for the next iteration. In the end of each iteration, the sized design is exported. Therefore, timing, area and power analysis can be performed over each design version (it is also done using the analysis tools of the commercial synthesis tool). This process is repeated for a given number of iterations.

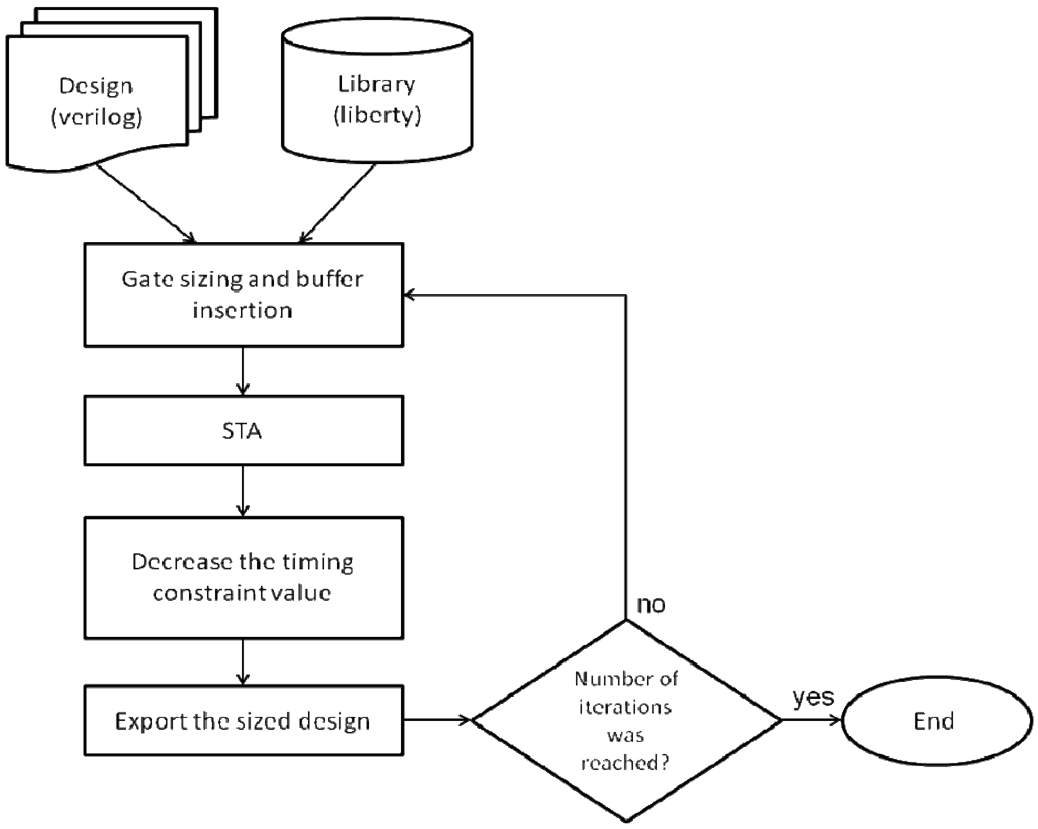

Design (verilog)
Library (liberty)
Gate sizing and buffer insertion
STA
Decrease the timing constraint value
Export the sized design
Number of iterations was reached?
no
yes
End

Figure 5.4: Buffering and sizing framework.

The whole experiment depends on a cell library. We have created and characterized a library of 664 cells composed by:

- All cells of the genlib 44-6 up to 6 inputs in two versions: positive and negative unate and in three different drive strengths.
- PTL XOR2, XOR3, XNOR2 and XNOR3 cells in three drive strengths each.
- NCSP cells that were identified by VIRMA in two drive strengths each.

The library was characterized using a commercial tool for library characterization. All SPICE netlists of each logic cell were automatically generated using the techniques presented by Rosa (2008). It includes the transistors sizing that were calculated through the logical effort method proposed by Sutherland (1999) using a fixed gain constant calibrated for a 130nm technology process. The characterization tool does not provide the area of the cell. All area values were estimated using our own area estimator, taking into account the cell topology, the transistor sizes and the cell height (in our experiment, 9 rows). It was necessary five full days to characterize the whole library using two dual-core processors. The area estimation has taken around two hours.

We have run the sizing framework for the circuits analyzed in the previous experiments. Results are summarized by Figures 5.5 to 5.8. Figure 5.5 shows a delay comparison for the circuit c1908. It reflects the common observed behavior for most of the analyzed circuits. After some iterations, the VIRMA tool can achieve a slight better delay than the other tools. Area results are shown in Figure 5.6 for the same benchmark circuit. As expected, commercial tools give a smaller area given that they produce worst delay. The ABC method mapping comes in third place with 30% of area increase. Breaking the expectations, the VIRMA mapping gives the worst results in terms of area. Figure 5.7 shows a delay comparison for circuit c6288. In this case, VIRMA begins with the best delay. However, in the second iteration, the ABC method starts to give better results, keeping the same proportion to the other methods. The area results that are shown in Figure 5.8 reflect the effect observed for the circuit c1908. In all cases, the power measurements were linearly proportional to the area values.

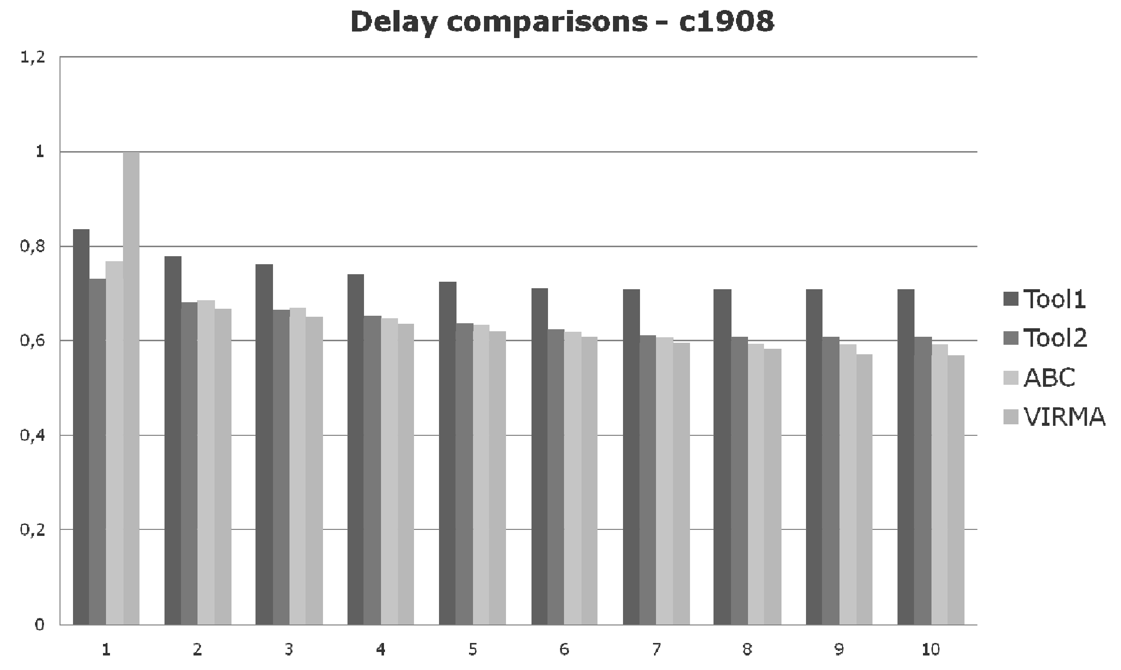


Figure 5.5: Delay comparisons using the benchmark circuit c1908.

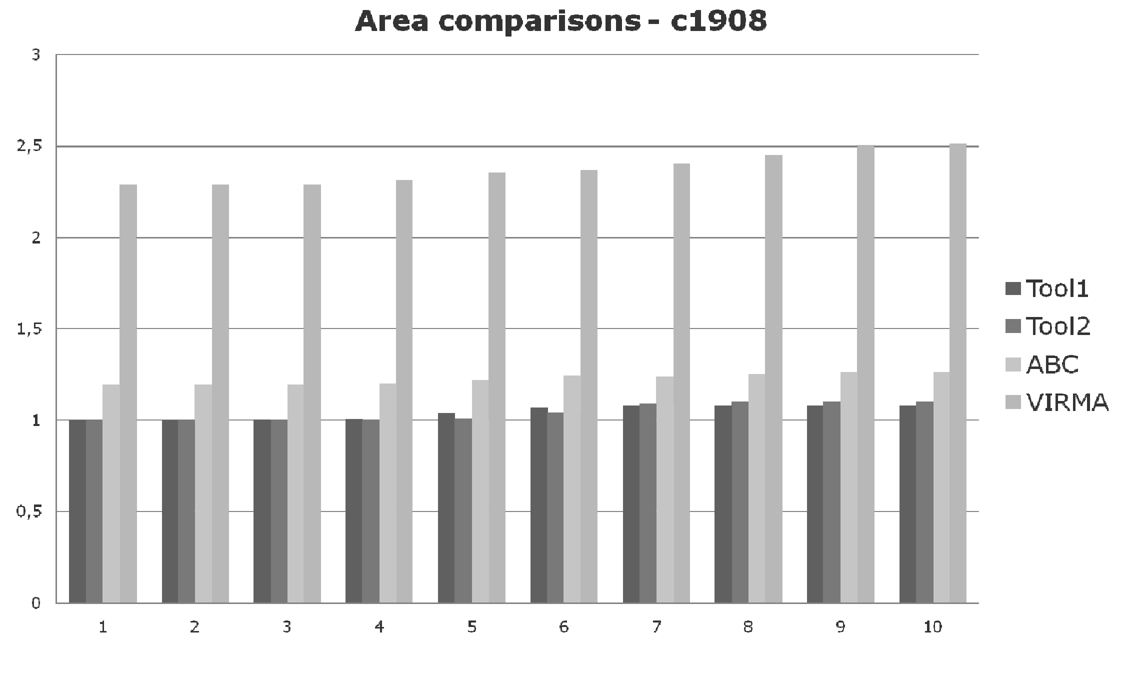


Figure 5.6: Area comparisons using the benchmark circuit c1908.

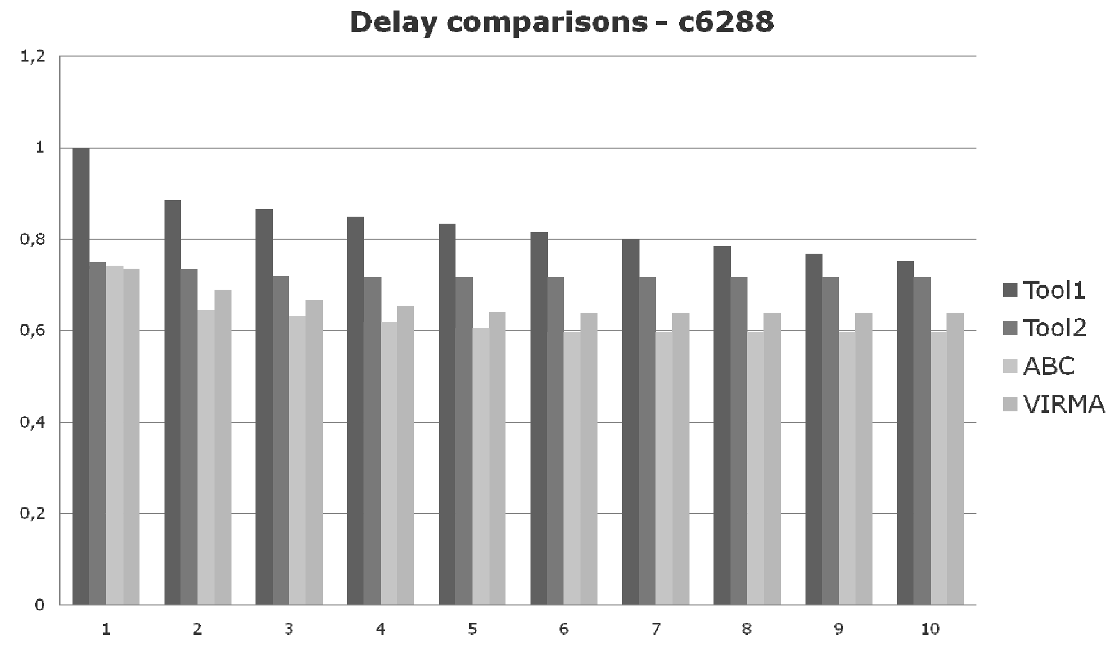


Figure 5.7: Delay comparisons using the benchmark circuit c6288.

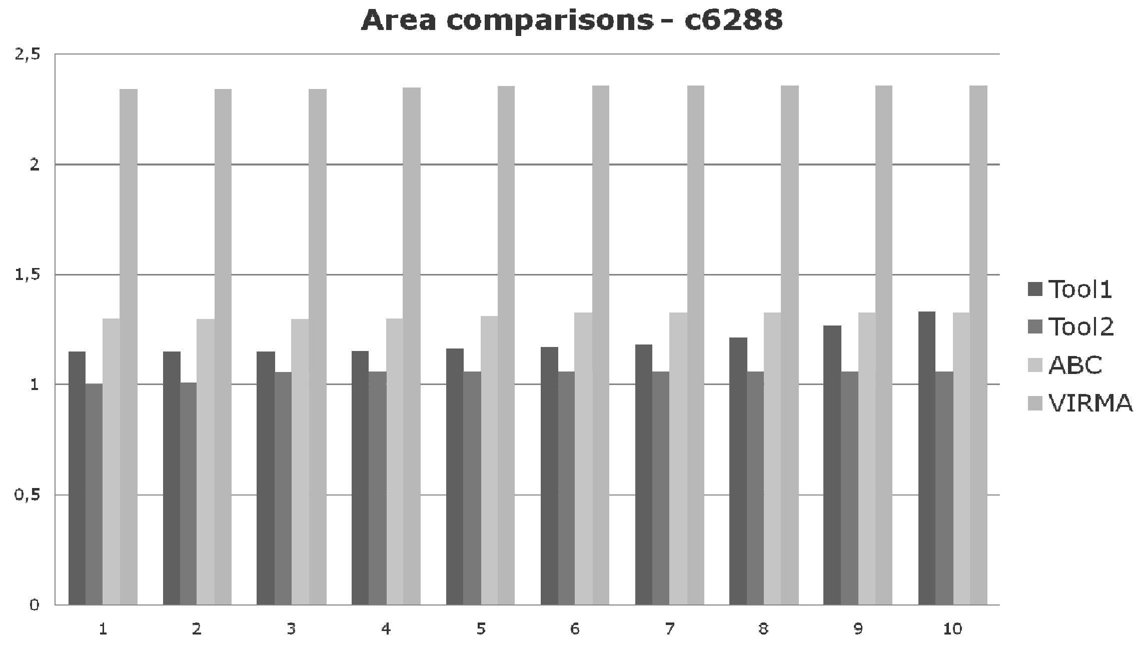


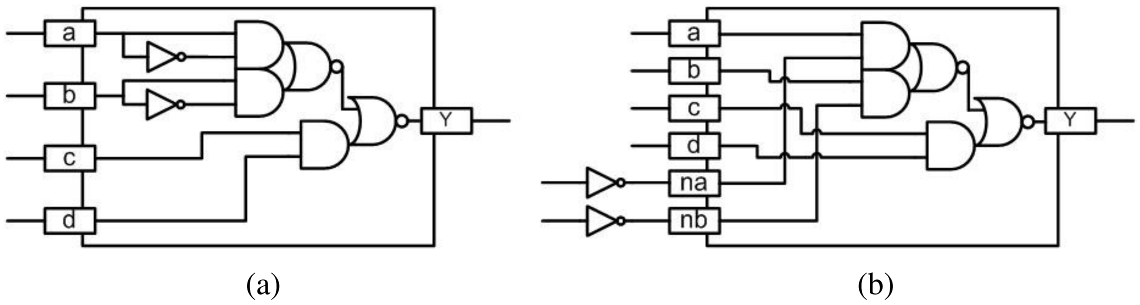
Figure 5.8: Area comparisons using the benchmark circuit c6288.

In order to understand why VIRMA gives bad results for area, the NCSP cells have to be analyzed. Table 4.12 shows the area report of the circuit c6288. The number of inverters of the circuit mapped using VIRMA method is more than five times bigger that in the circuit mapped with the ABC method. Another import point is that the area occupied by NCSP cells corresponds to 44.1% of the total area.

Table 4.12: Close look at the c6288 area report.

	VIRMA			ABC		
Type	**Instances**	**Area (μ²)**	**Area(%)**	**Instances**	**Area (μ²)**	**Area(%)**
Inverter	870	6776.980	18.6	156	1214.600	5.9
Buffer	4	37.638	0.1	8	75.276	0.4
Logic	1148	29656.124	81.3	1236	19212.981	93.7
Total	**2022**	**36470.742**	**100.0**	**1400**	**20502.858**	**100.0**
NCSP Cells	*451*	*16087.257*	*44.1*	*0*	*0*	*0*

Most of the NCSP cells of the library are similar to the cell of Figure 5.9.a. These cells can be expressed through equations containing repeated literals and inversions at the inputs. In some cases, it would be interesting to have cells like the one in Figure 5.9.b. This way, the cell instances would guarantee that the input *na* and *nb* are the complements of the inputs *a* and *b*, respectively, keeping the logic equivalence. However, the tool used to perform the characterization does not allow a setup of the input stimuli for characterization simulations. It automatically generates the input vectors considering the cell inputs. Therefore, it assumes that the function implemented by the cell of Figure 5.9.b depends on six variables and a vector *v(a,b,c,d,na,nb) = {0,1,1,1,0,1}* could be used in the characterization process. It leads to the characterization of a non-equivalent cell resulting, for instance, in different timing and power characteristics. Due to this limitation, all NCSP cells that have input inversions have inverters inside the cell.



(a) (b)

Figure 5.9: Characterization problem of NCSP cells.

The consequence of having the inverters inside the cell is demonstrated in Figure 5.10. The circuit of Figure 5.10.a has two times more inverters than the circuit of Figure 5.10.b. The delay of the circuit can also be affected. Consider the circuit of Figure 5.11. In this case, there is one extra inverter in the longest path of the circuit. This effect was observed in the benchmark circuit c6288. This circuit has a regular structure and has several instances of the same cell. Each one of the instances needs extra input inverters that results in area and delay increase.

Another problem, which was observed in the library cells, is the transistor sizing. All transistors have to fit into the specified cell height. When it is not possible, the transistor sizing algorithm performs a transistor folding procedure. In average, it at least duplicates the area of occupied by a transistor. Some of the generated cells were just a little bit higher to fit on the maximum height. In this case, the transistor folding is applied resulting in area increase. Alternatively, it could make transistors larger enough to fit on the cell height and have a small delay penalty. It requires a fine tuning in the transistors sizing. Since each cell topology has its peculiarities, this is a hard task, which is out of the scope of this work.

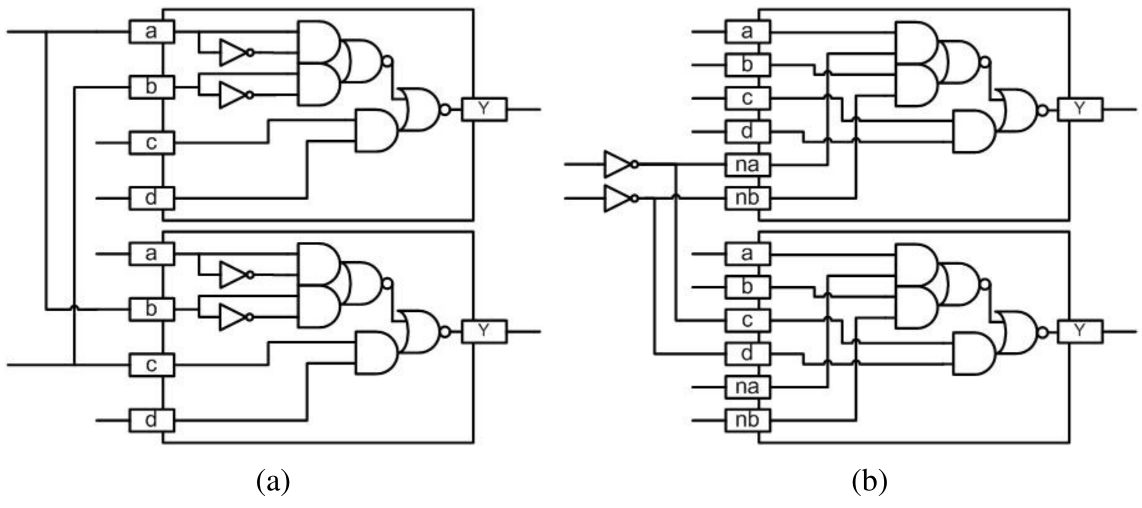


Figure 5.10: Area effects of having inverters inside the cell.

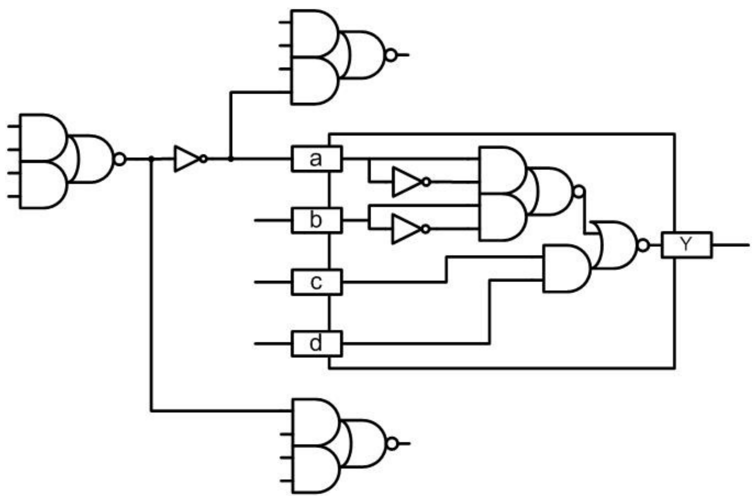


Figure 5.11: Delay effects of having inverters inside the cell.

Although the results obtained from the last experiment are not good enough for area and power, they show that VIRMA can be competitive in terms of delay. The next chapter presents some conclusions about it, as well as guidelines for future works.

6 CONCLUSIONS AND FUTURE WORKS

This thesis presented two DAG-based approaches for technology mapping using virtual libraries. VIRMA methods are pioneering approaches regarding the use of cells with minimal length transistor chains (SCHNEIDER, 2005), aiming delay minimization in combinational circuits.

The first method, which is called VIRMA-WF, was derived from the wavefront algorithm presented in (STOK, 1999). A comparison between the traditional technology mapping performed by SIS tool and VIRMA-WF method was presented in (MARQUES, 2007). It shows delay reductions up to 43%. For some circuits, better delay means a high area penalty. Although the VIRMA-WF algorithm uses DAGs as subject graphs, the method was easily adapted to handle trees. It was demonstrated that the mapped circuits are 14% faster in average and have a negligible area increase.

Based on the work presented by Mishchenko (2005), a new version of VIRMA was proposed later. This approach relies on a simple algorithm that is able to enumerate all *k*-feasible cuts of a graph. Each *k*-cut is matched against the library and can be used as part of an optimal cover. Both VIRMA methods were compared to ABC logic synthesis tool (ABC, 2008). Even limited to *k*-cuts up to 6 variables, VIRMA-*K*-Cuts method shows better results than VIRMA-WF method. ABC mapping gives slight better results in terms of delay. Early results revealed that the circuits mapped with VIRMA tool have a smaller transistor count.

The results obtained from naive implementations show the potential of these techniques. However, there are several open issues that should be better explored to improve VIRMA methods. The implicit logic duplication that occurs in DAG based approaches can be circumvented by allowing duplications only in the critical regions of the circuit. It was demonstrated through some experiments that a good area/delay trade-off can be achieved when this technique is used. Both VIRMA methods search for delay optimal implementations. The algorithms can be extended to a two-pass algorithm that can consider other non-delay metrics to map circuits.

One of the main problems in virtual library technology mapping approaches is the lack of accurate information of the cell characteristics. Methods to estimate area, delay and power consumption could be used to solve this problem. Butzen (2008) has presented a fast algorithm to estimate leakage-current power consumption. It could be a good approach for power estimation. The other alternative is to create hash tables with the characterization data. Therefore, a regular Boolean matcher would be needed to create the keys (for instance, binary strings) for each match. This hash tables could also store the lower bound values in terms of series transistors for each cell match. It would reduce the mapping time since these values would not be calculated on the fly.

In parallel with this work, there was another work exploring different cell topologies and making comparisons concerning timing, power and area. The results, which were presented in (ROSA, 2008), show that the NCSP topology is generally a good alternative to implement logic functions. However, it is not the best choice for every logic function. Such results were validated at the cell level. The main objective of this thesis is to evaluate the effectiveness of the use of NCSP in a design. In order to validate the experiments, a framework for mapping, sizing and analysis was created over a commercial synthesis tool. Four mapping method of different tools (including VIRMA, ABC and two commercial tools) were integrated in this framework. The results show that VIRMA can be competitive to other tools in terms of delay. Nevertheless, area and power results are not good enough.

The analysis performed in this work depends on several variables concerning almost all steps of the logic synthesis flow. Assumptions made in intermediate steps have major impact in the subsequent steps. One of the main problems found during the analysis is related to the automated transistor sizing strategy used to generate an experimental cell library. It requires a fine tuning in the algorithm to find a good trade-off for area, power and timing. A library design project is very complex and it is hard to be fully automated. There are many issues to be explored in this field. A good strategy for gate sizing was proposed by Schlinker (2008). This iterative approach can also be extended to perform design optimizations taking into account NCSP cells. Since it relies in a more robust methodology for design analysis, it would lead to better results. All techniques presented in this work are just the starting point for constructing a more powerful logic synthesis tool.

REFERENCES

ABC (from Berkeley Logic Synthesis and Verification Group). **ABC**: A System for Sequential Synthesis and. Available at: <http://www.eecs.berkeley.edu/~alanmi/abc/abc.html>. Visited on: in Feb. 2008.

ABOUZEID, P. et al. Logic synthesis for automatic layout. In: THE EUROPEAN EVENT IN ASIC DESIGN, 1992. **Proceedings…** [S.l.: s.n.], 1992. p. 146-151.

AVCI, M.; YILDIRIM, T. General design method for complementary pass transistor logic circuits. **Electronics Letters**, [S.l.], v. 39, n. 1, p.46–48, Jan. 2003.

BERKELAAR, M.; JESS, J. Technology mapping for standard-cell generators. In: INT. CONF. COMPUTER-AIDED DESIGN, 1988, Santa Clara, CA. **Proceedings…** [S.l.: s.n.], 1988. p. 470-473.

BRACE, K.; RUDELL, R.; BRYANT, R. Efficient Implementation of a BDD Package. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, DAC, 1990. **Proceedings...** [S.l.: s.n.], 1990. p. 40-45.

BRAYTON, R. Factoring logic functions. **IBM Journal of Research and Development**, Riverton, NJ, USA, v. 31, n. 2, p. 187-198, Mar. 1987.

BRAYTON, R. et al. MIS: A multiple-level logic optimization system. **IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems**, [S.l.], v. 6, n. 6, p. 1062-1081, Nov. 1986.

BRYANT, R. Graph-Based Algorithms for Boolean Function Manipulation. **IEEE Transactions on Computers**, New York, v. C-35, n. 8, p. 677-691, 1986.

BUCH, P. et al. Logic synthesis for large pass transistor circuits. In: ICCAD, 1997. **Proceedings…** New York: IEEE, 1997. p. 663-670.

BUTZEN, P. **Leakage Current Modeling in Sub-micrometer CMOS Complex Gates**. 2008. Master Degree Thesis. Engenharia de Computação. Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

CHATTERJEE, S.; MISHCHENKO, A.; BRAYTON, R. Factor cuts. In: ICCAD, 2006. **Proceedings…** New York: ACM, 2006. p. 143-150.

CHATTERJEE, S. et al. Reducing Structural Bias in Technology Mapping. ICCAD, 2005. **Proceedings…** [S.l.: s.n.], 2005. p. 519–526.

CONG, J.; DING, Y. FlowMap: An optimal technology mapping algorithm for delay optimization in look-up table based FPGA designs. **IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems**, [S.l.], v. 13, n. 1, p. 1-12, Jan. 1994.

CORREIA, V.; REIS, A. Advanced technology mapping for standard-cell generators. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, SBCCI, 17., 2004, Porto de Galinhas. **Proceedings...** Los Alamitos: IEEE Computer Society, 2004. p. 254–259.

DETJENS, E. et al. Technology mapping in MIS. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1987. **Proceedings...** [S.l.: s.n.], 1987. p. 116-119.

DEVADAS, S.; GHOSH, A.; KEUTZER, K. **Logic synthesis**. New York: McGraw-Hill, 1994.

GRAVRILOV, S. et al. Library-less synthesis for static CMOS combinational logic circuits. In: INT. CONF. COMPUTER-AIDED DESIGN, 1997. **Proceedings...** [S.l.: s.n.], 1997, p. 658 – 662.

GREGORY, D. et al. SOCRATES: a system for automatically synthesizing and optimizing combinational logic. In: ACM/IEEE DESIGN AUTOMATION CONFERENCE, 23., 1986. **Proceedings...**[S.l.: s.n.], 1986. p. 79-85.

HSIAO, S.; YEH, J.; CHEN, D. High performance multiplexer-based logic synthesis using pass-transistor logic. In: ISCAS, 2000, Geneva. **Proceedings...** Piscataway: IEEE, 2000. v. 2, p. 325-328.

JIANG, Y.; SAPATNEKAR, S.; BAMJI, C. Technology mapping for high-performance static CMOS and pass transistor logic designs. **IEEE Transactions on VLSI**, New York, v. 9, n. 5, p. 577–589, Oct. 2001.

KANECKO, M.; TIAN, J. Concurrent cell generation and mapping for CMOS logic circuits. In: ASPDAC, 1997. **Proceedings...** New York: ACM SIGDA, 1997. p. 247–252.

KARNAUGH, M. The Map Method for the Synthesis of Combinational Circuits. **AIEE Transactions**, [S.l.], p. 593-599, Nov. 1953.

KEUTZER, K. Dagon: Technology binding and local optimization by DAG matching. In: DESIGN AUTOMATION CONFERENCE, DAC, 24., 1987. **Proceedings...** [S.l.: s.n.], 1987. p. 341-347.

KEUTZER, K.; KOLWICZ, K.; LEGA, M. Impact of library size on the quality of automated synthesis. INT. CONF. COMPUTER-AIDED DESIGN, 1987. **Proceedings...** [S.l.: s.n.], 1987. p. 120-123.

KEUTZER, K.; RICHARDS, D. Computational complexity of logic synthesis and optimization. In: INTERNATIONAL WORKSHOP ON LOGIC SYNTHESIS, 1989. **Proceedings...** [S.l.: s.n.], 1989. p. 1-15.

KUKIMOTO, Y.; BRAYTON, R.; SAWKAR, P. Delay-optimal technology mapping by DAG covering, In: IEEE/ACM DESIGN AUTOMATION CONFERENCE, 1998. **Proceedings…** [S.l.: s.n.], 1998. p. 348-351.

LEE, C. Representation of switching circuits by Binary-Decision Programs. **Bell System Technical Journal**, New York, v. 38, n. 7, p. 985-999, July 1959.

LEGA, M. Mapping properties of multi-level logic synthesis operations. IEEE INT. CONF. COMPUTER DESIGN, 1988. **Proceedings…** [S.l.: s.n.], 1988. p. 257-260.

LEHMAN, E. et al. Logic decomposition during technology. **IEEE Transactions on Computer-Aided Design**, New York, v.16, n. 8, p. 813-834, Aug. 1997.

LIEN, C.; LEFEBVRE, M. A constructive matching algorithm for cell generator based technology mapping. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 1992. **Proceedings…** [S.l.: s.n.], 1992. v. 6, p. 2965-2968.

MAILHOT, F.; DEMICHELI, G. Algorithms for technology mapping based on binary decision diagrams and on Boolean operations. **IEEE Transactions on CAD for IC and Systems**, [S.l.], v. 12, n. 5, p. 599-620, May 1993.

MARQUES, F.; ROSA, L.; RIBAS, R.; SAPATNEKAR, S.; REIS, A. DAG Based Library-Free Technology Mapping. In: GLSVLSI, 2007. **Proceedings…** [S.l.: s.n.], 2007.

MISHCHENKO, A.; CHATTERJEE, S.; BRAYTON, R. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In: DAC, 2006. **Proceedings…** [S.l.: s.n.], 2006. p. 532-536.

MISHCHENKO, A. et al. **Technology mapping with Boolean matching, supergates and choices**. Berkeley: EECS Dept., UC Berkeley, 2005. Technical Report.

MCCLUSKEY, E. J. Minimization of Boolean Functions. **Bell Systems Technical Journal**, [S.l.], v. 35, p. 1417-1444, June 1956.

MCGEER, P. et al. Espresso-Signature: A new exact minimizer for logic functions. In: DESIGN AUTOMATION CONFERENCE, 1993. **Proceedings…** [S.l.: s.n.], 1993.

MINTZ, A.; GOKUMBIC, M. Factoring Boolean functions using graph partitioning. **Discrete Applied Mathematics**, [S.l.], v. 149, n. 1-3, p. 131-153, Aug. 2005.

MOORE, G. Cramming more components onto integrated circuits. **Electronics Magazine**,[S.l.], v. 38, n. 8, Apr. 1965.

POLI, R.; SCHNEIDER, F.; RIBAS, R.; REIS, A. Unified theory to build cell-level transistor networks from BDDs. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, SBCCI, 16., 2004, São Paulo. **Proceedings…** Los Alamitos: IEEE Computer Society, 2003. p. 199 – 204.

QUINE, W. V. A Way To Simplify Truth Functions. **American Mathematical Monthly**, [S.l.], v. 62, p. 627-631, 1955.

REIS, A. **Assignation Technologique sur Bibliotheques Virtuelles de Portes Complexes CMOS**. 1998. 123 f. These (Doctorate m Electronique, Optronique et Systemes) – Université de Montpellier, Grenoble.

REIS, A.; REIS, R. Covering strategies for library free technology mapping. In: ACM INTERNATIONAL WORKSHOP LOGIC SYNTHESIS, 1999, Lake Tahoe. **Workshop notes…** [S.l.: s.n.], 1999.

REIS, A.; REIS, R.; AUVERGNE, D.; ROBERT, M. Library free technology mapping. In: INT. CONF. ON VERY LARGE SCALE INTEGRATION, 1997, Gramado, RS, Brazil. **Proceedings…** [S.l.: s.n.], 1997.

ROSA, L.; MARQUES, F.; CARDOSO, T.; RIBAS, R.; REIS, A. BBDs and transistor networks with minimum pull-up/pull-down chains. In: INTERNATIONAL WORKSHOP ON LOGIC SYNTHESIS, 2006. **Proceedings…** [S.l.: s.n.], 2006.

ROSA, L.; MARQUES, F.; SCHNEIDER, F.; RIBAS, R.; REIS, A. A Comparative Study of CMOS Gates with Minimum Transistor Stacks. In: SBCCI – Chip in Rio, 2007. **Proceedings…** [S.l.: s.n.], 2007.

ROSA, L.; REIS, A. **Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles**. 2008. PhD Degree Thesis. Engenharia de Computação. Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

ROY, R.; BHATTACHARYA, D.; BOPPANA, V. Transistor-level optimization of digital designs with flex cells. **Computer**, [S.l.], v.38, n.2, p.53-61, Feb. 2005.

RUDELL, R. **Logic Synthesis for VLSI design**. Berkeley: University of California, 1989. (TR UCB/ERL M89/49).

SCHNEIDER, F.; RIBAS, R.; SAPATNEKAR, S.; REIS, A. Exact lower bound for the number of switches in series to implement a combinational logic cell. In: ICCD, 2005. **Proceedings…** [S.l.: s.n.], 2005. p. 357-362.

SCHNEIDER, R. **Exact lower bound for the number of switches in series to implement a combinational logic cell**. 2006. Master Degree Thesis. Engenharia de Computação. Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

SCOTT, K.; KEUTZER, K. Improving cell libraries for synthesis, In: CICC, 1994 **Proceedings…** [S.l.: s.n.], 1994. p. 128-131.

SCHLINKER, G.S. **Gate Sizing Algorithms For Standard Cell Designs**. 2008. Graduate Degree Thesis. Engenharia de Computação.Universidade Federal do Rio Grande do Sul, Porto Alegre, Brasil.

SECHEN, C. et al. LIBRARIES: Lifejacket or straitjacket. In: IEEE DESIGN AUTOMATION CONFERENCE, 2003. **Proceedings…** [S.l.: s.n.], 2003. p. 642-643.

SECHEN, C.; GUAN, B. Large standard cell libraries and their impact on layout area and circuit performance. In: ICCD, 1996. **Proceedings…** [S.l.: s.n.], 1996. p. 378-383.

SENTOVICH, E. et al. **SIS**: A system for sequential circuit synthesis. Berkeley: EECS Department, University of California, 1992. (TR UCB/ERL M92/41).

SHANNON, C. A symbolic analysis of relay and switching circuits. **Transactions American Institute of Electrical Engineers**, [S.l.], v. 57, p. 713-723, 1938.

SHELAR, R.; SAPATNEKAR, S. An efficient algorithm for low power pass transistor logic synthesis. In: ASPDAC, 2002. **Proceedings…** [S.l.: s.n.], 2002. p. 87-92.

SHELAR, R.; SAPATNEKAR, S. Recursive bipartitioning of BDDs for performance driven synthesis of pass transistor logic circuits. In: ICCAD, 2001. **Proceedings…** [S.l.: s.n.], 2001. p. 449-452.

STOK, L.; IYER, M.; SULLIVAN, A. Wavefront technology mapping. In: DESIGN, AUTOMATION AND TEST IN EUROPE, 1999, Germany. **Proceedings…** [S.l.: s.n.], 1999. p. 531-536.

SUTHERLAND, I.; SPROULL, B.; HARRIS, D. **Logical Effort**: Designing Fast CMOS Circuits. [S.l.]: Morgan Kaufmann, 1999.

SYNOPSYS LIBERTY LIBRARY FORMAT. **Liberty CSS**. Available at: <http://www.synopsys.com/products/libertyccs/libertyccs.html>. Visited on: Feb. 2008.

TANAKA, K.; KAMBAYASHI, Y. Transduction method for design of logic cell structure. In: ASPDAC, 2004. **Proceedings…** [S.l.: s.n.], 2004. p. 600–603.

TJIANG, S. **Twig reference manual**. [S.l.: s.n.], 1986.

VERILOG DOT COM. **Verilog Resources**. Available at: <http://www.verilog.com/>. Visited on: Feb 2008.

VHDL ORG. **VASG: VHDL Analysis and Standardization Group**. Available at: <http://www.vhdl.org/vasg/>. Visited on: Feb 2008.

WAGNER, F.; RIBAS, R.; REIS, A. **Fundamento de Circuitos Digitais**. Porto Alegre: Sagra Luzzatto, 2006.

WESTE, N.; ESHRAGHIAN, K. **Principles of CMOS VLSI design**: a systems perspective. 2nd ed. [S.l.]: Addison-Wesley, 1994.

YAMASHITA, S. et al. Pass-transistor/CMOS collaborated logic: the best of both worlds. In: SYMPOSIUM ON VLSI CIRCUITS, 1997. **Proceedings…** [S.l.: s.n.], 1997. p. 31–32.

ZHAO, M.; SAPATNEKAR, S. A new structural pattern matching algorithm for technology mapping. In: IEEE/ACM DESIGN AUTOMATION CONFERENCE, 2001, Las Vegas, NV. **Proceedings…** New York, NY: ACM Press, 2001. p. 371-376.

APPENDIX A MAPEAMENTO TECNOLÓGICO PARA BIBLIOTECAS VIRTUAIS ULITILIZANDO CÉLULAS COM UM NÚMERO MÍNIMO DE TRANSISTORES EM SÉRIE

De acordo com a “Lei de Moore” (MOORE, 1965), desde a invenção dos circuitos integrados (CIs), em 1958, o número de transistores que podem ser colocados em um único *chip* aumenta exponencialmente a cada evolução tecnológica que reduz a escala dos transistores. Atualmente, bilhões de componentes eletrônicos podem ser integrados em um único *chip*. O poder de processamento e a capacidade de memória dos dispositivos eletrônicos digitais aumentam nas com o número de transistores.

Embora tecnologias sub-micrônicas permitam um alto grau de integração de semicondutores, esta integração torna o projeto, a verificação e o teste de circuitos integrados mais difíceis. Normalmente, o projeto de circuitos integrados é consideravelmente afetado com a diminuição do tamanho dos dispositivos eletrônicos em tecnologias sub-micrônicas. Efeitos como corrente de fuga, ruído e eletro-migração eram considerados irrelevantes em tecnologias mais antigas. Atualmente, a análise destes efeitos é fundamental para o sucesso de um projeto. Conseqüentemente, os projetistas adotam metodologias rígidas para produzir circuitos de alta qualidade em tempo razoável. Ferramentas para automação do projeto de circuitos eletrônicos são utilizadas para automatizar algumas das etapas do projeto, ajudando o projetista a encontrar boas soluções rapidamente.

A metodologia adotada na maioria dos fluxos para automação são baseadas em bibliotecas de células. Neste caso, o fluxo de síntese inicia com uma descrição em alto-nível do projeto, utilizando linguagens de descrição de *hardware*, tais como VHDL (VHDL ORG, 2008) e Verilog (VERILOG DOT COM, 2008). O segundo passo é a síntese lógica, que realiza varias manipulações lógicas sobre a descrição de alto-nível, transformando-a em uma descrição a nível de portas lógicas, que instanciam células de uma biblioteca. O último passo é a síntese física que posiciona e conecta as células em um *floorplan*. A principal vantagem desta metodologia é que cada célula de uma biblioteca é caracterizada através de várias simulações, resultando em um conjuntos de informações bastante precisas sobre o comportamento das células. Desta forma, o projetista pode, com o auxílio de ferramentas de automação, prever as características do circuito final com uma boa precisão.

Embora as ferramentas para automação desempenhem um bom trabalho na busca por boas soluções para um dado projeto, existem diversos pontos que podem ser melhorados no fluxo de automação. Além disso, novos problemas surgem cada vez que um processo de fabricação evolui para a próxima geração. Isto demanda uma constante atualização nas ferramentas disponíveis ou ferramentas completamente novas. Logo, existe um alto custo associado à migração para novas tecnologias. Isto exige investimentos tanto em ferramentas quanto em processos de fabricação. Alternativamente, o projetista pode explorar outras estratégias para otimizações a fim de aumentar o desempenho e reduzir área e potência de circuito sem mudar a tecnologia.

Um dos maiores desafios no projeto de circuitos integrados de alto-desempenho é atingir as metas de atraso na lógica combinacional de controle. Normalmente, a lógica de controle não é regular o suficiente para ser implementada em um fluxo de projeto intuitivo. Além disso, este tipo de lógica sofre modificações até as últimas etapas do ciclo de projeto. Consequentemente, a síntese lógica é necessária para viabilizar a implementação dos circuitos, garantindo uma implementação rápida e correta de sub-circuitos irregulares. A maioria das ferramentas de síntese lógica consiste de três etapas bem definidas. O mapeamento tecnológico normalmente segue uma etapa de otimizações em uma descrição do circuito, sendo esta independente de tecnologia. A etapa posterior ao mapeamento tecnológico é uma etapa para otimizações específicas, considerando as restrições de atraso do circuito.

O mapeamento tecnológico é uma etapa da síntese lógica que escolhe quais células serão utilizadas para implementar um circuito em uma determinada tecnologia. Esta fase da síntese lógica tem o maior impacto na estrutura do circuito, definindo suas principais características de área e atraso. A maioria das técnicas são baseadas em bibliotecas estáticas de células (*standard cell*), onde um conjunto de células é definido e caracterizado para uma determinada tecnologia. Os primeiros métodos para mapeamento tecnológico (KEUTZER, 1987-a) (DETJENS, 1987) (ABOUZEID, 1992) (MAILHOT, 1993) (LIEN, 1992) usavam árvores para representar o circuito a ser mapeado. Algoritmos mais recentes (LEHMAN, 1997) (KUKIMOTO, 1998) (STOK, 1999) (MISHCHENKO, 2005) são baseados em grafos acíclicos direcionados (DAGs), permitindo a duplicação de lógica para aumentar o desempenho do circuito. Outra contribuição importante para o mapeamento tecnológico foi o matching Booleano (MAILHOT, 1993). O *matching* de uma porção do circuito e uma célula da biblioteca é feito por uma comparação Booleana entre funções candidatas. Uma comparação estrutural pode não ser suficiente para encontrar todos matchings possíveis.

Além das abordagens que realizam mapeamento tecnológico com bibliotecas estáticas, outras abordagens consideram o uso de geradores de células, o que possibilitaria o uso de uma grande biblioteca virtual (construída sob demanda). Um trabalho pioneiro visando geradores de células foi apresentado por Berkelaar (1988). Neste método, expressões lógicas decompostas são mapeadas em portas complexas. Um outro método, proposto por Reis (1997), utiliza Diagramas de Decisão Binária (BDDs) para representar um circuito e aplica decomposições nestas estruturas, considerado uma restrição para o número máximo de transistores em série admitido para implementar uma dada função Booleana. Cada BDD decomposto é mapeado em uma porta complexa CMOS estática. O trabalho apresentado em (CORREIA, 2004) explora dinamicamente decomposições AND/OR usando árvores de grau livre que representam o circuito a ser mapeado. Cada sub-árvore é limitada pelo número de transistores em série necessários para implementar uma célula CMOS estática. Em (JIANG, 2005) duas técnicas para mapeamento tecnológico são apresentadas. O primeiro método mapeia circuitos para uma biblioteca virtual de células CMOS estáticas. A segunda técnica usa uma lógica mista de células CMOS e PTL, considerando a relação direta de uma célula PTL e um BDD.

Infelizmente, o uso destas abordagens não foi suficientemente verificado pela indústria, mesmo considerando que outras referências sugerem que um número maior de células lógicas pode melhorar a qualidade do circuito mapeado (KEUTZER, 1987-b) (SCOTT, 1994) (SECHEN, 1996) (GAVRILOV, 1997). Uma abordagem recente (ROY, 2005) propõe a adição de células específicas à biblioteca de células para aumentar o desempenho dos circuitos mapeados.

Recentemente, alguns métodos para geração eficiente de células lógicas foram propostos (KANECKO, 1997) (POLI, 2003) (TANAKA, 2004) (SCHNEIDER, 2005), incluindo um método para calcular o número mínimo de transistores em série necessários para implementar uma função Booleana arbitrária, apresentado por Schneider (2005). Além do dimensionamento dos transistores, a topologia da rede de transistores também interfere no atraso das células. Logo, a redução do número de elementos em série tende a reduzir o atraso das células.

A topologia proposta por Schneider (2005) foi apresentada no nível de células lógicas. Logo, é necessária uma metodologia para avaliar o uso desta topologia em circuitos maiores. Motivado por este fator, esta tese apresenta dois métodos para mapeamento tecnológico baseados em grafos acíclicos direcionados (DAGs), que deram origem a uma ferramenta para mapeamento tecnológico, chamada VIRMA (VIRtual library technology MApping). Os algoritmos reduzem o número de transistores em série do caminho mais longo do circuito, considerando que cada célula é implementada por uma rede de transistores que obedecem um número máximo de transistores em série. O número de transistores em série é calculado de forma booleana, garantindo que este seja o número mínimo necessário para implementar a função lógica da instância da célula. Os algoritmos estão integrados a um gerador de células que utiliza as técnicas apresentadas em (ROSA, 2008) para a geração de redes de transistores e também realiza o dimensionamento dos transistores.

O primeiro método, que é chamado de VIRMA-WF, foi derivado do algoritmo *wavefront* apresentado por Stok (1999). Uma comparação entre o tradicional algoritmo para mapeamento tecnológico realizado pela ferramenta SIS e o algoritmo VIRMA-WF foi apresentado em (MARQUES, 2007). Ela mostra reduções de atraso que chegam a 43%. Em alguns circuitos, menor atraso significa maior área. Embora o método VIRMA-WF seja baseado em DAGs ele pode facilmente ser adaptado para realizar o mapeamento sobre árvores. Resultados comparativos mostram que o mapeamento utilizando árvores pode aumentar o desempenho dos circuitos em 14% com um aumento insignificante em área.

Baseado no método apresentado por Mishchenko (2005), uma segunda versão do método VIRMA foi proposta. A nova abordagem utiliza o algoritmo de Mishchenko para enumeração de *k-cuts* sobre um grafo. Cada *k-cut* equivalente a uma célula da biblioteca, poderá ser parte de uma cobertura ótima. Os métodos da ferramenta VIRMA foram comparados com o método de Mishchenko (2005), que está disponível na ferramenta ABC (ABC, 2008). Mesmo limitado a *k-cuts* de até 6 variáveis, o algoritmo VIRMA-*K-Cuts* mostra resultados melhores que o algoritmo VIRMA-WF. O mapeamento da ferramenta ABC consegue obter resultados um pouco melhores em termos de atraso. Os primeiros resultados relativos à área mostraram que a ferramenta VIRMA obtém circuitos com uma menor contagem de transistores.

Os resultados obtidos com as primeiras versões dos algoritmos mostram o potencial destas técnicas. No entanto, existem vários problemas que devem ser explorados visando melhorias nos métodos da ferramenta VIRMA. As duplicações implícitas de lógica, que ocorrem em técnicas de mapeamento baseado em DAGs, podem ser evitadas se o algoritmo limitá-las as regiões críticas do circuito. Este efeito foi analisado em alguns experimentos, demonstrando que uma boa relação de compromisso entre área e atraso pode ser encontrada quando esta técnica é utilizada. Os métodos da VIRMA buscam somente a otimização do atraso dos circuitos. Os algoritmos podem ser estendidos para duas passagens, de forma que a segunda passada considere outras métricas que não somente o atraso do circuito.

Um dos principais problemas do mapeamento tecnológico utilizando bibliotecas virtuais é a falta de informações precisas sobre o comportamento das células. Métodos para estimativa de área, potência e atraso poderiam ser utilizados para resolver este problema. Butzen (2008) apresentou um método que realiza uma rápida estimativa do consumo causado por corrente de fuga. Assim como este método para estimativa de potência, outros métodos podem ser utilizados para medir outros critérios. Uma outra alternativa é a criação de *hash tables* contendo os dados provenientes de um processo de caracterização. Desta forma, seria necessário utilizar um método *matching* Booleano para identificar equivalências entre porções do circuito e as entradas desta *hash table*. Estas tabelas também poderiam armazenar os valores correspondentes ao número mínimo de transistores em série de uma célula. Isto poderia reduzir o tempo de mapeamento, visto que este valor não seria calculado durante a execução do mapeamento.

Paralelamente a este trabalho, outro estudo estava sendo desenvolvido, analisando diferentes topologias de células e realizando comparações de atraso, potência e área entre um conjunto de células. Os resultados, que foram apresentados em (ROSA, 2008), mostram que, em geral, a topologia NSCP é uma boa alternativa para implementar funções Booleanas. Todos os experimentos foram realizados no nível de células lógicas. O principal objetivo do trabalho é avaliar o uso efetivo de células NCSP em circuitos combinacionais. Para validar os experimentos inicias, um fluxo para mapeamento tecnológico, dimensionamento e análise de circuitos foi criado sobre uma ferramenta comercial. Quatro métodos para mapeamento (VIRMA, ABC e duas ferramentas comerciais) foram integrados neste fluxo. Os resultados mostram que a ferramenta VIRMA pode ser competitiva em termos de atraso. Porém, os resultados de área e potência não são ainda bons o suficiente.

A análise que foi realizada neste trabalho depende de várias variáveis de cada uma das etapas do fluxo de síntese. Algumas decisões tomadas em passos intermediários podem ter um impacto muito grande nos passos subseqüentes. Um dos principais problemas encontrados durante a análise foi a estratégia de dimensionamento de transistores do gerador de células, que foi utilizado durante criação da biblioteca experimental. Ela precisa de ajustes mais precisos para encontrar um bom compromisso entre área, potência e atraso. O projeto de uma biblioteca de células é bastante complexo e difícil de ser automatizado. Uma boa estratégia para o dimensionamento de portas lógicas foi proposta por Schlinker (2008). Este algoritmo iterativo pode ser estendido para realizar otimizações em um circuito considerando células NCSP. Como o algoritmo é baseado em uma metodologia mais robusta para a análise do circuito, ele tende a encontrar melhores resultados. Todas as técnicas apresentadas neste trabalho são somente o ponto inicial para a construção de uma ferramenta de síntese lógica mais completa.

APPENDIX B ACADEMIC LIBRARY DESCRIPTIONS USED IN THE EXPERIMENTS

This appendix shows the full description of two academic libraries that are distributed with the SIS technology mapping tool (SENTOVICH, 1992). The libraries are described in the *genlib* format. This format is briefly illustrated below.

1. Genlib library format

A cell is specified in the following format:

GATE <cell_name> <cell_area> <cell_logic_function>

PIN <pin_name> <phase> <input_load> <max_load> <rise_block_delay> <rise_fanout_delay> <fall_block_delay> <fall_fanout_delay>

<cell_name> is the name of the cell in the cell library.

<cell_area> defines the relative area cost of the cell. It is a floating point number, and may be in any unit system convenient for the user.

<cell_logic_function> is an equation written in conventional algebraic notation using the operators "+" for OR, "*" or nothing (space) for AND, "!" or "" (post-fixed) for NOT, and parentheses for grouping. The names of the literals in the equation define the input pin names for the cell; the name on the left hand side of the equation defines the output of the cell. The equation terminates with a semicolon.

<pin_name> must be the name of a pin in the <cell_logic_function>, or it * to specify identical timing information for all pins.

<phase> is INV, NONINV, or UNKNOWN corresponding to whether the logic function in negative unite, positive unate, or binate in this variable respectively. This is required for the separate rise-fall delay model.

<input_load> gives the input load of this pin. It is a floating point value, in arbitrary units convenient for the user.

<max_load> specifies a loading constraint for the cell. It is a floating point value specifying the maximum load allowed on the output.

<rise_block_delay> and **<rise_fanout_delay>** are the rise-time parameters for the timing model. They are floating point values, typically in the units nanoseconds, and nanoseconds/unit_load respectively.

<fall_block_delay> and **<fall_fanout_delay>** are the fall-time parameters for the timing model. They are floating point values, typically in the units nanoseconds, and nanoseconds/unit_load respectively.

2. Library: *lib2.genlib*

```
GATE inv1   928.00 O=!a;
PIN a INV 0.0514 999.0 0.4200 4.7100 0.4200 3.6000
GATE xor  2320.00 O=(!a*b)+(a*!b);
PIN a UNKNOWN 0.1442 999.0 1.7700 5.2300 0.9600 4.6400
PIN b UNKNOWN 0.1381 999.0 1.9400 4.6500 1.1400 5.2200
GATE xnor  2320.00 O=(!a*!b)+(a*b);
PIN a UNKNOWN 0.1502 999.0 1.1100 4.8600 1.0700 3.3900
PIN b UNKNOWN 0.1352 999.0 1.5500 4.8700 1.0700 3.3900
GATE nand2  1392.00 O=!(a*b);
PIN a INV 0.0777 999.0 0.6400 4.0900 0.4000 2.5700
PIN b INV 0.0716 999.0 0.4600 4.1000 0.3700 2.5700
GATE nand3  1856.00 O=!(a*b*c);
PIN a INV 0.1000 999.0 0.8900 3.6000 0.5100 2.4900
PIN b INV 0.0828 999.0 0.7100 4.1100 0.4200 2.5000
PIN c INV 0.0777 999.0 0.5600 4.3900 0.3500 2.4900
GATE nand4  2320.00 O=!(a*b*c*d);
PIN a INV 0.1030 999.0 1.2700 3.6200 0.6700 2.3900
PIN b INV 0.0980 999.0 1.0900 3.6100 0.6100 2.3900
PIN c INV 0.0980 999.0 0.8200 3.6200 0.5500 2.4000
PIN d INV 0.1050 999.0 0.5800 3.6200 0.3800 2.3900
GATE nor2  1392.00 O=!(a+b);
PIN a INV 0.0736 999.0 0.3300 3.6400 0.4500 3.6400
PIN b INV 0.0968 999.0 0.5000 3.6400 0.7000 3.6600
GATE nor3  1856.00 O=!(a+b+c);
PIN a INV 0.0856 999.0 0.8400 5.0400 1.3000 3.4500
PIN b INV 0.0806 999.0 0.7800 5.0300 1.1400 3.4300
PIN c INV 0.0826 999.0 0.5200 5.0300 0.8400 3.4400
GATE nor4  2320.00 O=!(a+b+c+d);
PIN a INV 0.0887 999.0 0.4100 5.9100 1.1600 3.2000
PIN b INV 0.0867 999.0 0.8500 5.9100 1.5300 3.1800
PIN c INV 0.0867 999.0 1.1100 5.9200 1.7500 3.1900
PIN d INV 0.0887 999.0 1.2700 5.9100 1.9400 3.2000
GATE aoi21  1856.00 O=!((a*b)+c);
PIN a INV 0.1029 999.0 0.7500 3.5200 0.6700 2.5300
PIN b INV 0.0908 999.0 0.6700 3.6400 0.6200 2.5200
PIN c INV 0.1110 999.0 0.5800 3.6400 0.2100 1.2800
```

```
GATE aoi31  2320.00 O=!((a*b*c)+d);
PIN a INV 0.1009 999.0 0.9100 4.0400 0.8100 2.8600
PIN b INV 0.1049 999.0 1.0500 3.9300 0.8700 2.8700
PIN c INV 0.1059 999.0 1.1500 3.9400 0.9400 2.8600
PIN d INV 0.0979 999.0 0.8900 4.0600 0.2500 1.2800
GATE aoi22  2320.00 O=!((a*b)+(c*d));
PIN a INV 0.1019 999.0 0.9200 3.4600 0.9400 2.7900
PIN b INV 0.0908 999.0 0.8400 3.6400 0.8500 2.7900
PIN c INV 0.0958 999.0 0.6100 3.6400 0.4900 2.9300
PIN d INV 0.0988 999.0 0.7000 3.6400 0.5400 2.9300
GATE aoi32  2784.00 O=!((a*b*c)+(d*e));
PIN a INV 0.1029 999.0 1.0600 3.8100 0.9600 2.9100
PIN b INV 0.1009 999.0 1.2000 3.8100 1.0300 2.9000
PIN c INV 0.1060 999.0 1.2900 3.6900 1.0600 2.9100
PIN d INV 0.0979 999.0 0.9100 3.8100 0.4300 2.1200
PIN e INV 0.1049 999.0 0.7800 3.5900 0.4300 2.1200
GATE aoi33  3248.00 O=!((a*b*c)+(d*e*f));
PIN a INV 0.1029 999.0 1.3300 3.9100 1.3000 2.9100
PIN b INV 0.1029 999.0 1.4600 3.8400 1.4100 2.9100
PIN c INV 0.1120 999.0 1.4700 3.6500 1.4100 2.9100
PIN d INV 0.1029 999.0 1.1100 3.5900 0.7600 2.9000
PIN e INV 0.0949 999.0 1.0400 3.9100 0.6800 2.9100
PIN f INV 0.1039 999.0 0.8400 3.5800 0.6400 2.9000
GATE aoi211  2320.00 O=!((a*b)+c+d);
PIN a INV 0.1039 999.0 1.1200 4.8100 1.0300 2.3800
PIN b INV 0.1090 999.0 1.2900 4.8100 1.0300 2.3800
PIN c INV 0.1080 999.0 1.0400 4.8300 0.5200 1.4000
PIN d INV 0.1008 999.0 0.6800 4.8300 0.5100 1.7900
GATE aoi221  2784.00 O=!((a*b)+(c*d)+e);
PIN a INV 0.1089 999.0 1.4800 4.4300 1.3300 2.7800
PIN b INV 0.0948 999.0 1.4200 4.5600 1.4000 2.7500
PIN c INV 0.1029 999.0 0.7600 4.4700 0.7900 2.8900
PIN d INV 0.1049 999.0 0.7300 4.5800 0.7800 2.9100
PIN e INV 0.1110 999.0 1.3900 4.5600 0.7000 1.5100
GATE aoi222  3712.00 O=!((a*b)+(c*d)+(e*f));
PIN a INV 0.1019 999.0 1.7700 4.5800 1.5600 2.9500
PIN b INV 0.0958 999.0 1.7300 4.6900 1.6000 2.9300
PIN c INV 0.1039 999.0 1.3400 4.6800 1.2100 2.9200
PIN d INV 0.1039 999.0 1.5000 4.6900 1.2200 2.9200
PIN e INV 0.0958 999.0 0.9200 4.6700 0.8100 2.9200
PIN f INV 0.1039 999.0 0.7700 4.4700 0.7600 2.9200
GATE oai21  1856.00 O=!((a+b)*c);
```

```
PIN a INV 0.1019 999.0 0.6900 3.9400 0.5300 2.4700
PIN b INV 0.0979 999.0 0.8700 3.9300 0.6300 2.4700
PIN c INV 0.0998 999.0 0.3700 2.0500 0.5700 2.5100
GATE oai31  2320.00 O=!((a+b+c)*d);
PIN a INV 0.1089 999.0 1.2700 4.7100 1.0300 2.4300
PIN b INV 0.1049 999.0 1.1100 4.7100 1.0400 2.5700
PIN c INV 0.1090 999.0 0.8500 4.7100 0.6900 2.3800
PIN d INV 0.1059 999.0 0.3800 1.8600 0.8100 2.7300
GATE oai22  2320.00 O=!((a+b)*(c+d));
PIN a INV 0.1009 999.0 1.1000 4.0600 0.9000 2.5000
PIN b INV 0.1029 999.0 0.9900 4.0600 0.6800 2.3600
PIN c INV 0.0958 999.0 0.6900 3.6600 0.7400 2.5300
PIN d INV 0.1039 999.0 0.6100 3.6600 0.5600 2.0600
GATE oai32  2784.00 O=!((a+b+c)*(d+e));
PIN a INV 0.1130 999.0 1.3900 4.4600 1.0400 2.4600
PIN b INV 0.1069 999.0 1.2500 4.4600 1.0900 2.6300
PIN c INV 0.1140 999.0 0.9900 4.4600 0.7400 2.4200
PIN d INV 0.1059 999.0 0.5800 3.2000 0.7900 2.7100
PIN e INV 0.1130 999.0 0.6800 3.2100 0.8300 2.3400
GATE oai33  3248.00 O=!((a+b+c)*(d+e+f));
PIN a INV 0.1170 999.0 1.5800 4.3000 1.4800 2.4700
PIN b INV 0.1089 999.0 1.5000 4.3100 1.4200 2.6300
PIN c INV 0.1079 999.0 1.2400 4.3100 1.1700 2.6500
PIN d INV 0.1170 999.0 0.8000 4.3000 0.8200 2.2700
PIN e INV 0.1089 999.0 0.0000 4.3000 1.1700 2.6400
PIN f INV 0.1109 999.0 1.1300 4.3100 1.3500 2.6500
GATE oai211  2320.00 O=!((a+b)*c*d);
PIN a INV 0.1070 999.0 1.1200 4.1700 0.5900 2.3100
PIN b INV 0.1131 999.0 1.3000 4.1600 0.7900 2.3600
PIN c INV 0.1050 999.0 0.5100 2.1300 0.6900 2.4000
PIN d INV 0.1050 999.0 0.5000 2.4600 0.5200 2.4100
GATE oai221  2784.00 O=!((a+b)*(c+d)*e);
PIN a INV 0.1039 999.0 1.5800 4.1700 1.1100 2.4700
PIN b INV 0.1050 999.0 1.4800 4.1700 0.8600 2.3600
PIN c INV 0.1080 999.0 0.9400 4.0300 0.8100 2.5000
PIN d INV 0.1060 999.0 0.7600 4.0300 0.6400 2.5000
PIN e INV 0.1019 999.0 0.7800 2.2800 0.9000 2.5400
GATE oai222  3248.00 O=!((a+b)*(c+d)*(e+f));
PIN a INV 0.1161 999.0 1.7700 3.7500 1.2100 2.4700
PIN b INV 0.1110 999.0 1.6200 3.7500 1.1300 2.4800
PIN c INV 0.1009 999.0 1.1700 3.5800 1.0700 2.4800
PIN d INV 0.1191 999.0 1.3500 3.5800 1.1000 2.3500
```

```
    PIN e INV 0.1060 999.0 0.9900 3.5900 0.9300 2.4900
    PIN f INV 0.1140 999.0 0.8200 3.5800 0.7900 2.4800
    GATE zero O=CONST0;
    GATE one O=CONST1;
```

3. Library: 33-4.genlib

```
    GATE zero O=CONST0;
    GATE one O=CONST1;
    GATE buf 1 O = a;
    PIN a INV 1 999 1.0 0.2 1.0 0.2
    GATE "!a" 2 O=!a;
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab)'" 3 O=!(a*b);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+c))'" 4 O=!(a*(b+c));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+cd))'" 5 O=!(a*(b+c*d));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+c(d+e)))'" 6 O=!(a*(b+c*(d+e)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+(c+d)(e+f)))'" 7 O=!(a*(b+(c+d)*(e+f)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(bc+de))'" 6 O=!(a*(b*c+d*e));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(bc+d(e+f)))'" 7 O=!(a*(b*c+d*(e+f)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(bc+(d+e)(f+g)))'" 8 O=!(a*(b*c+(d+e)*(f+g)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+c+d))'" 5 O=!(a*(b+c+d));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+c+de))'" 6 O=!(a*(b+c+d*e));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+cd+ef))'" 7 O=!(a*(b+c*d+e*f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(bc+de+fg))'" 8 O=!(a*(b*c+d*e+f*g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+d))'" 5 O=!((a+b)*(c+d));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+de))'" 6 O=!((a+b)*(c+d*e));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+d(e+f)))'" 7 O=!((a+b)*(c+d*(e+f)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+(d+e)(f+g)))'" 8 O=!((a+b)*(c+(d+e)*(f+g)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(cd+ef))'" 7 O=!((a+b)*(c*d+e*f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(cd+e(f+g)))'" 8 O=!((a+b)*(c*d+e*(f+g)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(cd+(e+f)(g+h)))'" 9 O=!((a+b)*(c*d+(e+f)*(g+h)));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+d+e))'" 6 O=!((a+b)*(c+d+e));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+d+ef))'" 7 O=!((a+b)*(c+d+e*f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+de+fg))'" 8 O=!((a+b)*(c+d*e+f*g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(cd+ef+gh))'" 9 O=!((a+b)*(c*d+e*f+g*h));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+bc)(d+e+f))'" 7 O=!((a+b*c)*(d+e+f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b(c+d))(e+f+g))'" 8 O=!((a+b*(c+d))*(e+f+g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+(b+c)(d+e))(f+g+h))'" 9 O=!((a+(b+c)*(d+e))*(f+g+h));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((ab+cd)(e+f+g))'" 8 O=!((a*b+c*d)*(e+f+g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((ab+c(d+e))(f+g+h))'" 9 O=!((a*b+c*(d+e))*(f+g+h));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((ab+(c+d)(e+f))(g+h+i))'" 10 O=!((a*b+(c+d)*(e+f))*(g+h+i));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b+c)(d+e+f))'" 7 O=!((a+b+c)*(d+e+f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b+c)(d+e+fg))'" 8 O=!((a+b+c)*(d+e+f*g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b+c)(d+ef+gh))'" 9 O=!((a+b+c)*(d+e*f+g*h));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b+c)(de+fg+hi))'" 10 O=!((a+b+c)*(d*e+f*g+h*i));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(abc)'" 4 O=!(a*b*c);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab(c+d))'" 5 O=!(a*b*(c+d));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab(c+d+e))'" 6 O=!(a*b*(c+d+e));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
```

```
GATE "(a(b+c)(d+e))'" 6 O=!(a*(b+c)*(d+e));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a(b+c)(d+e+f))'" 7 O=!(a*(b+c)*(d+e+f));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a(b+c+d)(e+f+g))'" 8 O=!(a*(b+c+d)*(e+f+g));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "((a+b)(c+d)(e+f))'" 7 O=!((a+b)*(c+d)*(e+f));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "((a+b)(c+d)(e+f+g))'" 8 O=!((a+b)*(c+d)*(e+f+g));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "((a+b)(c+d+e)(f+g+h))'" 9 O=!((a+b)*(c+d+e)*(f+g+h));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "((a+b+c)(d+e+f)(g+h+i))'" 10 O=!((a+b+c)*(d+e+f)*(g+h+i));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+b)'" 3 O=!(a+b);
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+bc)'" 4 O=!(a+b*c);
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+b(c+d))'" 5 O=!(a+b*(c+d));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+b(c+de))'" 6 O=!(a+b*(c+d*e));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+b(cd+ef))'" 7 O=!(a+b*(c*d+e*f));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+(b+c)(d+e))'" 6 O=!(a+(b+c)*(d+e));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+(b+c)(d+ef))'" 7 O=!(a+(b+c)*(d+e*f));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+(b+c)(de+fg))'" 8 O=!(a+(b+c)*(d*e+f*g));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+bcd)'" 5 O=!(a+b*c*d);
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+bc(d+e))'" 6 O=!(a+b*c*(d+e));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+b(c+d)(e+f))'" 7 O=!(a+b*(c+d)*(e+f));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(a+(b+c)(d+e)(f+g))'" 8 O=!(a+(b+c)*(d+e)*(f+g));
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(ab+cd)'" 5 O=!(a*b+c*d);
PIN * INV 1 999 1.0 0.2 1.0 0.2
GATE "(ab+c(d+e))'" 6 O=!(a*b+c*(d+e));
PIN * INV 1 999 1.0 0.2 1.0 0.2
```

```
    GATE "(ab+c(d+ef))'" 7 O=!(a*b+c*(d+e*f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+c(de+fg))'" 8 O=!(a*b+c*(d*e+f*g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+(c+d)(e+f))'" 7 O=!(a*b+(c+d)*(e+f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+(c+d)(e+fg))'" 8 O=!(a*b+(c+d)*(e+f*g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+(c+d)(ef+gh))'" 9 O=!(a*b+(c+d)*(e*f+g*h));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+cde)'" 6 O=!(a*b+c*d*e);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+cd(e+f))'" 7 O=!(a*b+c*d*(e+f));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+c(d+e)(f+g))'" 8 O=!(a*b+c*(d+e)*(f+g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+(c+d)(e+f)(g+h))'" 9 O=!(a*b+(c+d)*(e+f)*(g+h));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+c)+def)'" 7 O=!(a*(b+c)+d*e*f);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(b+cd)+efg)'" 8 O=!(a*(b+c*d)+e*f*g);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a(bc+de)+fgh)'" 9 O=!(a*(b*c+d*e)+f*g*h);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+d)+efg)'" 8 O=!((a+b)*(c+d)+e*f*g);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(c+de)+fgh)'" 9 O=!((a+b)*(c+d*e)+f*g*h);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "((a+b)(cd+ef)+ghi)'" 10 O=!((a+b)*(c*d+e*f)+g*h*i);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(abc+def)'" 7 O=!(a*b*c+d*e*f);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(abc+de(f+g))'" 8 O=!(a*b*c+d*e*(f+g));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(abc+d(e+f)(g+h))'" 9 O=!(a*b*c+d*(e+f)*(g+h));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(abc+(d+e)(f+g)(h+i))'" 10 O=!(a*b*c+(d+e)*(f+g)*(h+i));
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a+b+c)'" 4 O=!(a+b+c);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a+b+cd)'" 5 O=!(a+b+c*d);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a+b+cde)'" 6 O=!(a+b+c*d*e);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a+bc+de)'" 6 O=!(a+b*c+d*e);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a+bc+def)'" 7 O=!(a+b*c+d*e*f);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(a+bcd+efg)'" 8 O=!(a+b*c*d+e*f*g);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+cd+ef)'" 7 O=!(a*b+c*d+e*f);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+cd+efg)'" 8 O=!(a*b+c*d+e*f*g);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(ab+cde+fgh)'" 9 O=!(a*b+c*d*e+f*g*h);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
    GATE "(abc+def+ghi)'" 10 O=!(a*b*c+d*e*f+g*h*i);
    PIN * INV 1 999 1.0 0.2 1.0 0.2
```

APPENDIX C VIRMA USER GUIDE

How to run VIRMA tool (and Java Virtual Machine setup)

java -Xmx512m -Xms256m -jar virma.jar <parameters> [options]

If you run it without any parameter or option, a brief help will be printed out. You do not need to specify the parameters and options in a pre-defined order. It is recommended to use 512Mb for the main memory and 256Mb for the stack memory.

Parameter description

- Input netlist: VIRMA can read EQN and BLIF files.
- Output netlist: the output netlist is a "mapped" EQN file. In this file, each equation represents a cell and it is associated to a logic style (CSP or NCSP). The logic style is used for the cell generator.
- Constraints and the Virtual Library: the virtual library is defined by two constraints: pu and pd. They determine the maximum number of transistors in the pull-up (pu) plane and in the pull-down (pd) plane of a cell. The parameters 'max_tree_k_cut' and 'max_dag_k_cut' define the maximum number of variables for matches generated from trees and DAGs. The implemented algorithm for finding the k-cuts is very limited. Therefore, practical values for these constraints are 10 and 6, respectively. Otherwise, it could take too long time for mapping a circuit.
- Options: since VIRMA maps a circuit for a virtual library, a list of cells can be provided for a cell generator. This list will contain all cells used in the mapped circuit. In order to generate it, you just need to specify the file name through the option '-output_library'. If you use the option '-lc' (short for library costs), the CSP and NSCP costs will be written in the output library file. The options '–CSP' and '–NSCP' are used to specify the logic style used for mapping. By default, the NSCP is taken. If you choose CSP, only the standard series/parallel CMOS cells will be used.