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

FELIPE RIBEIRO SCHNEIDER

Building Transistor-Level Networks Following the Lower Bound on the Number of Stacked Switches

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. André Inácio Reis
Orientador

Prof. Dr. Renato Perez Ribas
Co-orientador

Porto Alegre, Março de 2007.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Schneider, Felipe Ribeiro

On Building Transistor-Level Networks Following the Lower Bound on the Number of Stacked Switches / Felipe Ribeiro Schneider – Porto Alegre: Programa de Pós-Graduação em Computação, 2007.

110 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2006. Orientador: André Inácio Reis; Co-orientador: Renato Perez Ribas.

1.Logic Style. 2.Logic Synthesis. 3.Cell Libraries. I. Reis, André Inácio. II. Ribas, Renato Perez. III. On Building Transistor-Level Networks Following the Lower Bound on the Number of Stacked Switches.

AGRADECIMENTOS

Este trabalho é dedicado e em agradecimento aos meus pais, Genésio e Miraci Schneider, cuja motivação, suporte, carinho e doação em tempo integral permitiu-me concluir meus estudos com dedicação exclusiva, sem que eu tivesse que me preocupar por um único momento com a maioria das praticalidades da vida adulta enquanto estudante. Eu sempre soube que tive uma vida de privilégios proporcionada pela minha família e isso sempre me motivou a buscar nada menos que o melhor que eu tenho a oferecer como filho, amigo, estudante ou profissional.

Também dedico este trabalho aos meus mentores e amigos, Prof. André Reis e Prof. Renato Ribas, cuja dedicação se estende por vários anos, dentre e fora de classe, desde o momento que me interessei por pesquisa em microeletrônica até o princípio da minha carreira como Engenheiro.

Finalmente, gostaria de agradecer a todos os professores, colegas e amigos da UFRGS que de uma forma ou de outra contribuíram com humildade e genialidade ao meu crescimento pessoal e profissional, refletido aqui neste trabalho.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application-specific Integrated Circuit
BBL	Branch-based Logic
BDD	Binary Decision Diagram
CAD	Computer-aided Design
CMOS	Complementary MOS (Standard CMOS)
CPL	Complementary PTL
CSP	Complementary SP
DSM	Deep Submicron
FPGA	Field-programmable Gate-array
HDL	Hardware Design Language
IC	Integrated Circuit
ISOP	Irredundant SOP
LB	Lower bound (for the number of stacked switches)
MOS	Metal-oxide-semiconductor
NCSP	Non-complementary SP
NRE	Non-recurring Engineering
PDN	Pull-down Network
PLB	Programmable logic block
PTL	Pass-transistor Logic
PUN	Pull-up Network
RTL	Register transfer-level
SoC	Silicon-on-Chip
SOI	Silicon on Insulator
SOP	Sum-of-Products
SP	Series/Parallel
VLSI	Very Large Scale of Integration

LIST OF FIGURES

LIST OF TABLES

Building Transistor-Level Networks Following the Lower Bound on the Number of Stacked Switches

ABSTRACT

Both the propagation delay and the output slope in CMOS gates are strongly related to the number of stacked PMOS and NMOS devices in the pull-up and pull-down networks, respectively. The standard CMOS logic style is usually optimized targeting one logic plane, presenting then the complemented topology in the other one. As a consequence, the minimum number of stacked transistors is not necessarily achieved. In this work, a method to find the lower bound of stacked switches (transistors) in CMOS complex gates is presented. A novel CMOS logic style, derived from such method, is then proposed and compared to conventional CMOS style through a commercial cell characterizer. Electrical characterization of sets of 3- to 6-input functions was done in order to evaluate the new method. Significant gains in propagation delay were obtained without penalty in power dissipation or area.

Keywords: Logic style, Logic Synthesis, Cell Libraries.

Construindo Redes de Transistores De Acordo com o Número Mínimo de Chaves em Série

RESUMO

Em portas lógicas CMOS, tanto o atraso de propagação como a curva de saída estão fortemente ligados ao número de dispositivos PMOS e NMOS conectados em série nas redes de carga e descarga, respectivamente. O estilo lógico 'standard CMOS' é, em geral, otimizado para um dos planos, apresentando então o arranjo complementar no plano oposto. Consequentemente, o número mínimo de transistores em série não é necessariamente alcançado. Neste trabalho, apresenta-se um método para encontrar o menor número de chaves (transistores) em série necessários para se implementar portas lógicas complexas CMOS. Um novo estilo lógico CMOS, derivado de tal método, é então proposto e comparado ao estilo CMOS convencional através do uso de uma ferramenta de caracterização comercial. A caracterização elétrica de conjuntos de funções de 3 a 6 entradas foi realizada para avaliar o novo método, apresentando significativos ganhos em velocidade, sem perdas em dissipação de potência ou em área.

Palavras-Chave: Estilo lógico, Logic Synthesis, Bibliotecas de Células.

1 INTRODUCTION

1.1 Digital Integrated Circuit Design

An Integrated Circuit (IC) is an electronic system consisting of a number of miniaturized passive and active electronic devices (mainly transistors, resistors, capacitors and inductors) built on a monolithic semiconductor substrate. The large majority of the current ICs are implemented in the so called Metal-Oxide-Semiconductor (MOS) technology (WESTE, 1993; RABAEY, 2003), where in most fabrication processes the semiconductor is silicon.

IC design can be divided into two broad categories: analog and digital design. Analog IC design has specializations in power IC design and radio-frequency IC design. Analog design is used in the development of operational amplifiers, linear regulators, phase-locked loops, oscillators and active filters. Analog design is more concerned with the physics of the semiconductor devices such as gain, matching, power dissipation, and resistance. Fidelity of analog signal amplification and filtering is usually critical and as a result, analog ICs use larger area active devices than digital designs and are usually less dense in circuitry. In the other hand, digital IC design is used to produce components such as microprocessors, FPGAs (Field-Programmable Gate-Arrays), memories (RAM, ROM, and flash) and digital ASICs (Application-Specific Integrated Circuits). Digital design focuses on logical correctness, maximizing circuit density, and placing circuits so that clock and timing signals are routed efficiently. This work focuses on the digital VLSI (Very Large Scale of Integration) IC design flow (MEAD, 1980; WESTE, 1993; UYEMURA, 1999; CHEN, 2000; RABAEY, 2003).

Since the advent of the technology for constructing ICs, integration density and performance of these electronic systems have gone through an astounding revolution driven by the ability of integrating in a single system more and more transistors, the devices responsible by most of the complexity of digital ICs. Indeed, the increase in the number of transistors that can be integrated in a single die has grown exponentially in the last three decades, as predicted by the so called *Moore's Law* (INTEL, 2007; MOORE, 1965). Fig. 1.1 illustrates how this increase prediction has been proved correct so far. Although it has been frequently stated that such increase might cease in a few years due to physical limitations of IC manufacturing technologies, new design methodologies and fabrication process breakthroughs have proven that such cease can be postponed (MOORE, 2003).

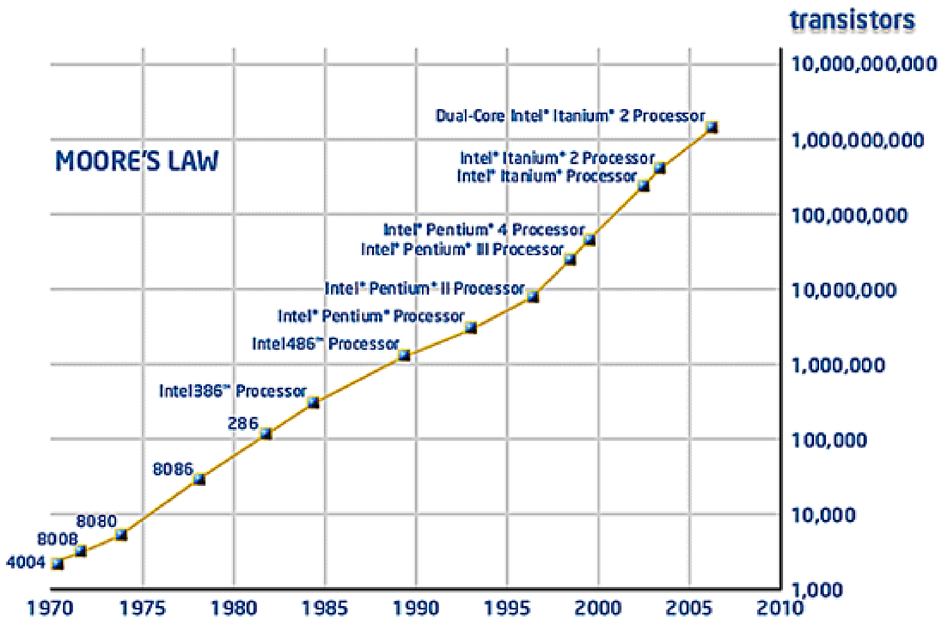


Figure 1.1: Moore's Law graph showing the exponential increase in the number of transistors along the last three decades for the microprocessors family from Intel (INTEL, 2007).

Designing ICs as complex as the ones available nowadays requires that engineers work with different levels of abstraction on a system design perspective. Fig. 2.1 shows a diagram illustrating these different levels of abstraction (GAJSKI, 1988). The highlighted abstraction levels, with special attention to *Logic Gates* in the *Structural Domain* and *Logic* in the *Behavioral Domain*, are the ones that concern the most to this work.

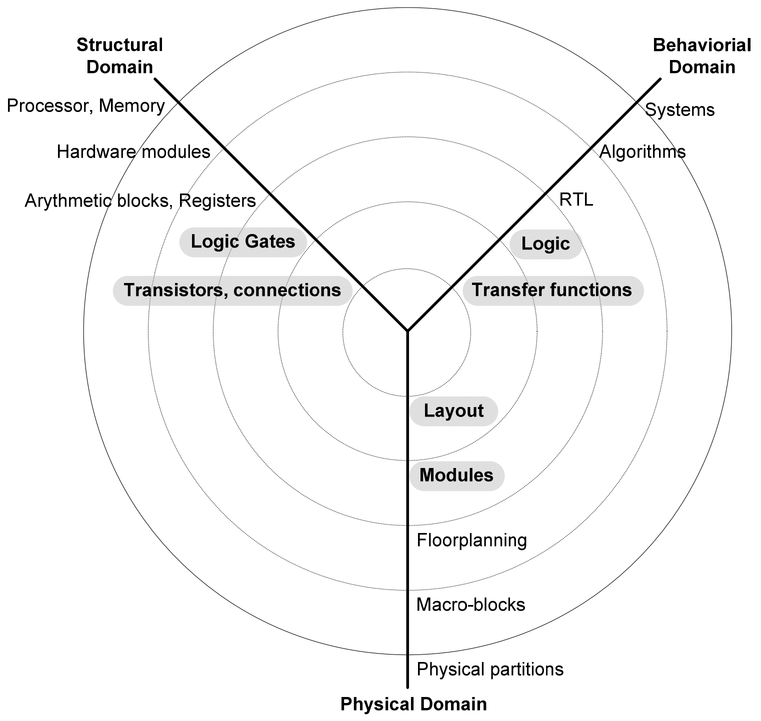


Figure 1.2: Abstraction levels on a system design perspective.

Among the technologies accessible today, two system design approaches play a major role in the scene of semiconductor-based integrated circuits: the FPGA-based design flow and the ASIC-based design flow. The FPGA design flow consists in configuring a set of programmable logic blocks (PLBs) built into a FPGA chip in order to achieve a desired application behavior. The ASIC design flow consists in assembling a set of basic electronic devices (i.e. transistors, resistors, capacitors, etc.) into structures called cells, which are the basic blocks in the construction of ASICs.

The choice between FPGAs and ASICs usually relies, among several factors, in the intended number of designs it is intended to be delivered to the market. The FPGA market has continuously growing as such reconfigurable chips becomes cheaper and holds more and more complex designs. However, when issues like power consumption, speed and area arise, nothing can beat the ASIC. The general rule states that for every FPGA-based design there is an ASIC that is more efficient considering area, energy and operation speed. On the other hand, ASICs are considered low-cost only for high volume designs (ZUCHOWSKI, 2002), due to the cost of dedicated masks.

This work focuses on the proposal of a new methodology for implementing, logically and structurally, the building blocks for ASICs. Therefore, next Section will focus on the most common ASIC design flows available at the present time.

1.2 ASIC Design Flow

As integrated circuits become more inexpensive and compact, many new types of products, such as digital cameras, digital camcorders and digital television (JURGEN 1997), are being introduced, based on digital systems. Consequently, logic design must be done under many different motivations. Since each case is different, one may have different design problems. Choosing an appropriate logic style, for example, is a very important issue when one wants to achieve certain performance requirements for a given IC design.

In this Section, let us consider two important cases of designing ICs, which leads to two contrasting logic approaches: quick design and high-performance design. Quick design of ICs is called *semi-custom design*. Recently, it has been also called *ASIC design* (CHINNERY, 2002), although the ASIC abbreviation is still used with a broader meaning. On the other hand, deliberate design for high-performance is called *full-custom design*, as this kind of design is fully customized to the high performance design space (area, speed, power and energy consumption).

In the next Subsections full-custom as well as different semi-custom approaches will be discussed.

1.2.1 Full-custom

In Full-custom design one does logic and physical synthesis in order to attain the highest performance or smallest size, making use of the most advanced technologies (CHEN, 2003). It is definitely the most technology dependent design approach existent: every transistor present in every cell inside every macro-block is tweaked in order to explore all the performance advantages that a given technology can deliver.

The benefits of full-custom design in general include reduced area (and therefore recurring component cost), performance improvements and also the ability to integrate (include) analog components and other pre-designed (and thus fully verified) components such as microprocessor cores that form a System-on-Chip (SoC).

The disadvantages of full-custom can include increased manufacturing and design time, increased non-recurring engineering (NRE) costs, more complexity in the Computer-Aided Design (CAD) system and a much higher skill requirement on the part of the design team.

However for digital only designs, cell-based semi-custom design together with modern CAD systems can offer considerable performance/cost benefits with much lower risk. Automated layout tools are quick and easy to use and also can offer the possibility to manually handcraft and optimize any performance limiting aspect of the design.

1.2.2 Gate Array

The Gate Array design is a manufacturing method in which the diffused layers, i.e. transistors and other active devices, are predefined. Wafers containing such devices are held in stock prior to metallization, in other words, unconnected (CHEN, 2000). The physical design process then defines the interconnections of the final device. For most ASIC manufacturers, this consists of two to as many as five metal layers, each one running perpendicular to the one below it. NRE costs are much lower as photo-lithographic masks are required only for the metal layers, and production cycles are much shorter as metallization is a comparatively quick process. It is also important to the designer that reduced propagation delays can be achieved in ASICs when compared to the available FPGAs solutions available in the marketplace.

Metal Programmable Gate Array Design is rarely implemented by circuit designers today. It has been replaced almost entirely by field programmable devices, such as FPGAs, which can be reprogrammed several times by the user and thus offer minimal tooling charges, marginally increased piece part cost and comparable performance. Nowadays, Gate Arrays are evolving into Structured ASICs (see Subsection 1.2.3) that typically consist of a large IP (Intellectual Property) core like a processor, DSP unit, peripheral components, standard interfaces, integrated SRAM memory, and a block of reconfigurable logic. This shift is largely because ASIC devices are capable of integrating such large blocks of system functionality and SoC designs requires far more than just logic blocks.

1.2.3 Structured ASIC

The basic premise of a Structured ASIC design (also referred to as Platform ASIC design) is that both manufacturing cycle time and design cycle time are reduced compared to cell-based ASICs due to pre-defined metal layers (thus reducing manufacturing time) and pre-characterization of what is on the silicon (thus reducing design cycle time). It is a relatively new design approach (PILLEGI, 2003; ZAHIRI, 2003).

In a Structured ASIC design, the logic mask-layers of a device are predefined by the ASIC vendor. Design differentiation and customization is achieved by creating custom metal layers that create custom connections between predefined lower-layer logic elements. Structured ASIC technology is seen as bridging the gap between field-programmable gate arrays and Standard-cell ASIC designs. Because only a small number of chip layers must be custom-produced, Structured ASIC designs have much smaller NRE than standard-cell or full-custom chips, which require that a full mask set be produced for every design. This is effectively the same definition as a Gate Array.

What makes a Structured ASIC different from a gate array is that in a gate array the predefined metal layers serve to make manufacturing turn-around time faster whereas in a Structured ASIC the predefined metallization is primarily to reduce cost of the mask sets and is also used to make the design cycle time significantly shorter as well. For example, in a cell-based or gate-array project the designers often must design power, clock, and test structures themselves: these are predefined in most Structured ASICs and therefore can save time and costs for the designer, when compared to gate-array. Another important aspect about Structured/Platform ASIC is that it allows IP that is common to certain applications or industry segments to be *built-in*, like one does in FPGAs, rather than *designed-in*, as it is done in cell-based design.

1.2.4 Standard-Cell

The idea behind cell-based design is to reduce the implementation effort by reusing a library of cells. The advantage of this approach is that the cells only need to be designed and verified once for a given technology, and they can be reused many times, thus amortizing the design cost. The disadvantage is that the constrained nature of the library (especially due to the limited number of cells) reduces the possibility of fine-tuning the design (RABAEY, 2003).

The Standard-cell approach standardizes the design entry-level at the logic gate (functional blocks). A library containing a wide selection of logic gates over a range of number of inputs and drive strengths is provided. Besides the basic logic functions such as inverter, AND/NAND, OR/NOR, XOR/XNOR and Flip-flops, a typical library also contains more complex functions such as AOI/OAI (AND/OR-OR/AND-INVERT), MUX, Full-adder, Comparator, Counter, Decoder and Encoder.

The layout of each cell in a specific library has a fixed height, while its width may vary, so the cell can be placed side-by-side, in such a way that their power rails and well regions properly connect to neighbor cells. Standard-cell design uses these functional blocks to achieve high gate density and good electrical performance. Standard-cell design fits between Gate Array and Full-Custom design in terms of both its NRE and recurring component cost.

The quality of a synthesized design based on standard-cells depends on three components: the synthesis tool, the place and route tools and the target cell library (SCOTT, 1994). Choosing the right cell library can have a significant impact on the characteristics of a designed circuit (VUJKOVIC, 2002; SECHEN, 2003).

The design flow, implemented with a level of skill common in the industry, almost always produce a final device that correctly implements the original design, unless flaws are later introduced by the physical fabrication process. It is as follows:

1. A team of design engineers starts with a non-formal understanding of the required functions for a new ASIC, usually derived from requirement analysis.
2. The design team constructs a description of an ASIC to achieve these goals using a HDL (Hardware Design Language). This process is analogous to

writing a computer program in a high-level language. This is usually called the RTL (Register transfer level) design.

3. A logic synthesis tool, in a process called technology mapping (SENTOVICH, 1992; REIS, 1995; JIANG, 2001; CORREIA, 2004), transforms the RTL design into a large collection of lower-level constructs called standard cells. These constructs are taken from a standard-cell library consisting of pre-characterized collections of gates. The standard cells are typically specific to the planned manufacturer of the ASIC. The resulting collection of standard cells, plus the needed electrical connections between them, is called a gate-level netlist. Standard-cells can be handcrafted or automatically generated.
4. The gate-level netlist is next processed by a placement tool which places the standard-cells onto a region representing the final ASIC. It attempts to find a placement of the standard-cells, subject to a variety of specified constraints.
5. The routing tool takes the physical placement of the standard-cells and uses the netlist to create the electrical connections between them. Since the search space is large, this process will produce a *sufficient* rather than *globally-optimal* solution. The output is a set of masks enabling a semiconductor fabrication to produce the physical ICs.
6. Close estimates of final delays, parasitic resistances and capacitances, and power consumptions can then be made. In the case of a digital circuit, this will then be further mapped into delay information that can be used to tune the design up.

In Fig. 1.3 an overview of the standard-cell design flow is presented.

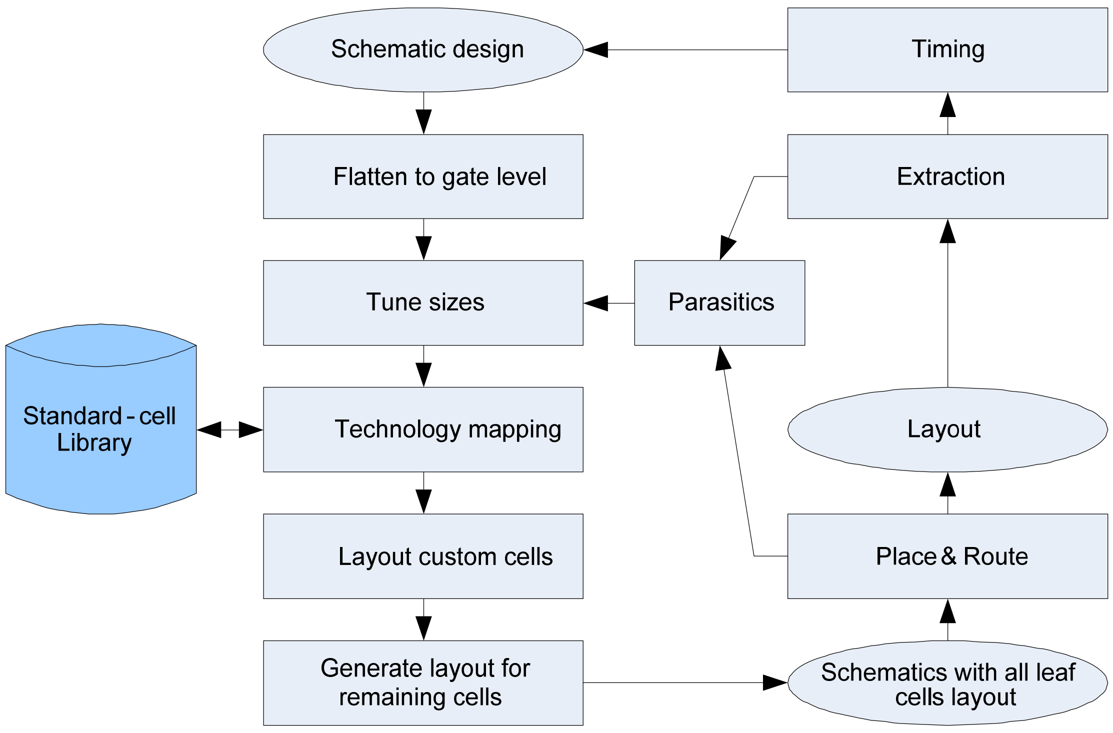


Figure 1.3: Design flow based on a standard-cell library (CHINNERY, 2002).

1.2.5 Library-free

One of the main restrictions of the standard-cell library-based designs is a limited number of cells and drive strengths available. In *library-free* (also called *library-less*) based design, as the name indicates, the library is said to be *virtual* (as it is not physically implemented before technology mapping) and may contain an unlimited number of cells (GAVRILOV, 1997; REIS, 1997; MORAES, 1999; CORREIA, 2004).

The main difference between standard-cell based technology mapping and library-free technology mapping concerns the libraries (pre-characterized and virtual, respectively) they must cope with. Library-free technology mapping implements the functions directly at the transistor-level, while guaranteeing that the final netlist of complex gates respect some topological constraints, e.g. number of transistors in series (REIS, 1998). The great number of available complex gates will improve the design space and lead to a minimization of the overall number of transistors, minimizing the design at the transistor-level.

1.3 A Novel Logic Style

ASIC design based on standard cells still remains the most applied approach, even if cell generators are used to create the library (LEFEBVRE, 1997; KEUTZER, 1999; CHINNERY, 2002; SECHEN, 2003). Although different CMOS logic styles, like PTL (BUCH, 1997; SCHOLL, 2000; JIANG, 2001; AVCI, 2003) and dynamic Domino gates (THORP, 2003), have been proposed as promising choices, the conventional Complementary Series/Parallel CMOS logic style (BERKELAAR, 1988; GAVRILOV, 1997), called here as CSP, continues to be the most widely adopted in cell libraries building.

Since the worst-case delay propagation is usually related to the longest path of stacked transistors, the construction of CMOS gates with minimum transistor stack length in the pull-up network (PUN) and pull-down network (PDN) is strongly recommended. As it will be demonstrated in this work, the number of stacked transistors has a direct impact in the *logical effort* (SUTHERLAND, 1999; KABBANI, 2005; WESTE, 2006; ROSA 2007) of the cells. However, this requirement cannot be guaranteed with CSP gates. This is explained by the fact that one plane is generated from the optimized logic equation, respecting then the minimum stack transistor number, but the other one is derived by making a complementary series/parallel topology, which will not guarantee minimum length transistor stacks if the gate has too many parallel branches.

A method to define the lower bound of stacked transistors in a logic plane presented in (SCHNEIDER, 2005) has some inconsistencies that were corrected in (SCHNEIDER, 2006a), with more practical results presented in (SCHNEIDER, 2006b) and further analyzed herein. The proposal of this work is both the *lower bound* theory as well as a novel Non-Complementary Series/Parallel CMOS logic style, named NCSP, derived from the *lower bound* method. Propagation delay reduction is expected for NCSP complex gates in comparison to standard CSP ones, with no penalty in power dissipation and area overhead. Electrical characterization using a commercial library characterization tool has been carried out over several logic cells, from 3- to 6- inputs, taking into account the TSMC 0.13um CMOS technology.

1.4 Organization of the Dissertation

This dissertation is organized as follows. In Section 2, it is presented a background of the most common static logic styles applied by the industry nowadays. Section 3 formulates the lower bound theory for the length of transistor stacks in logic cells and provides an enumeration of feasible cells using different logic styles considering such formulation. In Section 4, the implementation of a novel logic style so called NSCP is presented. Section 5 presents the electrical characterization of NCSP gates and a comparison with the most used static logic styles nowadays. Furthermore, Section 5 presents the impact in area, presenting some cell layouts. Finally, conclusions and an analysis of the impact of this work based on its results are pointed out on Section 6.

2 STATIC LOGIC STYLES

2.1 Introduction

The transistor is the most basic structure found in digital circuits, where it is mostly used as a switch. These switching devices are used to form the so called logic gates, which are the building blocks of digital integrated circuits. Each logic gate is designed to behave according to a desired logic function, where the output signals of a logic gate is a function of its input signals.

The circuit of a logic gate can be built using different configurations of transistors for a given logic function. These configurations are known as *logic styles* or *logic families*. There are numerous logic styles to implement a logic gate for a given logic function. Different styles are used to perform better for different design metrics like area, speed, energy and power consumption. Depending on the application, the emphasis will be on different metrics. For example, the switching speed of digital circuits is the primary metric in a high-performance processor, while in a battery operated circuit, it is energy consumption. Recently, power dissipation also has become an important concern and considerable emphasis is placed on understanding the mechanisms of power and approaches to dealing with power. One of these mechanisms currently highlighted by the industry and the academia is the leakage current, critical in the newest Ultra DSM (Deep Submicron) technologies (ROY, 2003). In addition to those metrics, robustness to noise and reliability are also very important considerations.

Logic styles are basically classified as being dynamic or static. *Dynamic* styles (THORP, 2003), rely on temporary storage of signal values on the capacitance of high-impedance circuit nodes. The implementation approach of dynamic circuits is simpler and faster but their design and operation are more prone to failure because of the increased sensitivity to noise. The most common dynamic logic styles are Domino and its variants: Dual Domino, Multiple-Output Domino, NORA Domino and Zipper Domino (WESTE, 2006). On the other hand, *Static* styles guarantee that, under fixed input vectors, each gate output is connected to either V_{DD} or GND via a low resistance path. Also, the outputs of the gate assume at all times the value of the Boolean function implemented by the circuit, meaning the circuit does not need to be pre-charged or pre-discharged. Some of the most common static logic styles are Static CMOS, Pseudo-NMOS, DCVSL and PTL (RABAEY, 2003).

Additionally, logic styles can be also classified as single- or dual-rail circuits. Single-rail circuits have only one output while dual-rail ones have two outputs, which very frequently are one for the direct polarity signal and one for the inverted polarity signal.

The most common logic styles used in the industry are the Complementary Series-Parallel CMOS (referred as CSP in this work) and the Pass-Transistor Logic (PTL), both static and single-rail. The logic style proposed in this work is also classified as static and single-rail. This allows its use with a design methodology flow very similar to the one applied currently to most integrated circuit designs, while it keeps the same robustness characteristic of CSP logic. In this sense the logic proposed here is superior to PTL, which may present drain inputs depending on how it is designed. Drain inputs are a design problem due to noise margin and also because the input capacitance seen at drain inputs is not constant.

In the next Sections, different logic styles available nowadays will be presented in order to allow a straightforward comparison with the logic style proposed and further explained on Chapter 4.

2.2 Complementary Series-Parallel CMOS

The Complementary Series-Parallel CMOS (CSP) is still nowadays the most used and well established logic style applied by the industry. The CSP style is basically an extension of the CMOS inverter to multiple inputs. The primary advantage of the CSP structure is robustness (i.e. low sensitivity to noise), good performance and low power consumption with almost no static power consumption for technologies with transistor channel length down to 130nm (WESTE, 2006).

However, newer Ultra DSM fabrication processes might considerably increase the leakage current responsible for static power consumption and in some point of time it ought to be comparable with dynamic power consumption. These increase has been dealt with more complex technologies, like SOI (Silicon-on-Insulator) and high-K dielectrics (WESTE, 2006), as well as with circuit-level modifications (ROY, 2003).

A static CMOS gate is a combination of two networks, called the *pull-up network* (PUN) and the *pull-down network* (PDN) (Fig. 2.1). The Fig. 2.1 shows a generic *N*-input logic gate where all the inputs are distributed to both the PUN and PDN. The function of the PUN is to provide a connection between the output and V_{DD} anytime the output of the logic gate is meant to be 1 (based on the inputs). Similarly, the function of the PDN is to connect the output to *GND* when the output of the logic gate is meant to be 0. The PUN and PDN networks are constructed in a mutually exclusive fashion such that *one and only one* of the networks is conducting in steady state. In this way, once the transients have settled, a path always exists between V_{DD} and the output *F*, realizing a high output (representing logic one), or, alternatively, between *GND* and *F* for a low output (representing logic zero). This is equivalent to stating that the output node is always a low-impedance node in steady state.

While constructing the PDN and PUN networks, the following observations should be kept in mind:

1. A transistor can be thought of as a switch controlled by its gate signal. An NMOS switch is ON when the controlling signal is high and is OFF when the controlling signal is low. A PMOS transistor acts as an inverse switch that is ON when the controlling signal is low and OFF when the controlling signal is high.

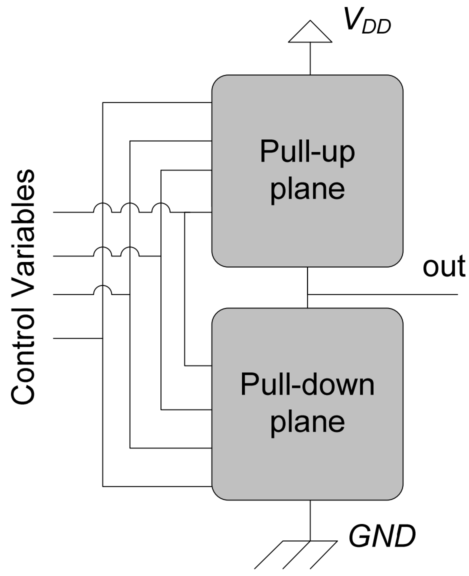
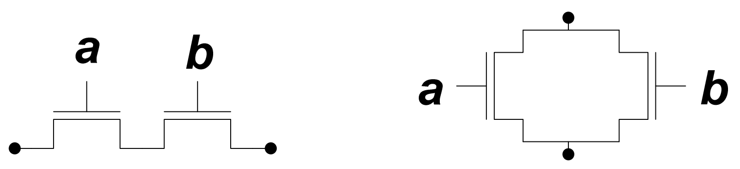


Figure 2.1: Static CMOS gate.

2. The PDN is constructed using NMOS devices, while PMOS transistors are used in the PUN. The primary reason for this choice is that NMOS transistors produce "strong zeros" and PMOS devices produce "strong ones" (RABAEY, 2003).

3. A set of construction rules can be derived to construct logic functions. NMOS devices connected in series corresponds to an NAND function (Fig. 2.2.a). With all the inputs high, the series combination conducts and the value at one end of the chain is transferred to the other end. Similarly, NMOS transistors connected in parallel represent an NOR function (Fig. 2.2.b). A conducting path exists between the output and input terminal if at least one of the inputs is high. Using similar arguments, construction rules for PMOS networks can be formulated. A series connection of PMOS conducts if both inputs are low, representing a NOR function ($\overline{a} \cdot \overline{b} = \overline{a + b}$), while PMOS transistors in parallel implement a NAND ($\overline{a} + \overline{b} = \overline{a \cdot b}$).

4. Using De Morgan's Theorems $\overline{a} \cdot \overline{b} = \overline{a + b}$ and $\overline{a} + \overline{b} = \overline{a \cdot b}$, it can be shown that the PUN and PDN networks of a complementary CMOS structure are dual networks. This means that a parallel connection of transistors in the pull-up network corresponds to a series connection of the corresponding devices in the pull-down network, and vice versa. Therefore, to construct a CMOS gate, one of the networks is implemented using combinations of series and parallel devices. The other network is obtained using duality principle by traversing the hierarchy, replacing series sub-nets with parallel sub-nets, and parallel sub-nets with series sub-nets. The complete CMOS gate is constructed by combining the PDN with the PUN.



(a) Series (b) Parallel

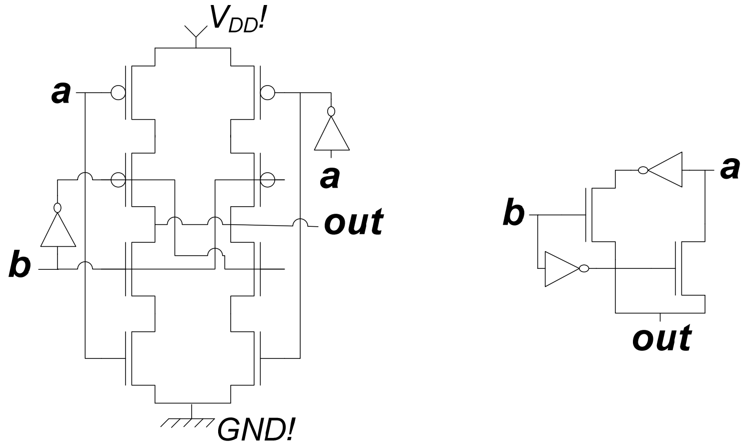
Figure 2.2: NMOS logic rules – series devices produces an AND, and parallel devices produces an OR

2.3 Pass-Transistor Logic

A popular and widely-used alternative to CSP is Pass-Transistor Logic (PTL) (BUCH, 1997; HSIAO, 2000; SCHOLL, 2000; LINDGREN, 2001; ZHOU, 2001; SHELAR, 2001; SHELAR, 2002; AVCI, 2003), which attempts to reduce the number of transistors required to implement logic by allowing the primary inputs to drive gate terminals as well as source/drain terminals. This characteristic contrasts with most logic families, which only allow primary inputs to drive the gate terminals of transistors.

The switches used in PTL circuits use either NMOS pass transistors or parallel pairs of NMOS and PMOS transistors called *transmission gates*. The most known PTL variation which uses transmission gates is called CPL (Complementary Pass-Transistor Logic) (YANO, 1990). Another distinct characteristic of PTL circuits is that it can present non-series/parallel network configurations.

One of the promises of PTL approach is that fewer transistors are required to implement some functions, especially XOR-based gates, which include MUXes. For example, the implementation of an XOR2 gate in Figure 2.3.a requires 6 transistors (including the inverter required to invert *b*), while a complementary CMOS implementation (Fig. 2.3.b) would require 12 transistors. The reduced number of devices has the additional advantage of lower capacitance. Many authors have claimed substantial area, speed and/or power improvements for pass-transistors compared to static CMOS. However, an independent evaluation finds that for most general-purpose logic, static CMOS is superior in speed power and area (ZIMMERMANN, 1997). Mixed approaches have also been proposed, as described in ().



(a) CSP (b) PTL

Figure 2.3: Different implementations of a 2-input XOR gate.

Another difference of PTL circuits it the way its logic is implemented. In logic styles like CSP, Boolean equations area translated into series/parallel arrangements. In PTL, however, the most common way of deriving circuits from logic is through the used of Binary Decision Diagrams (BBDs) (YANO, 1996). Fig. 2.4 presents the basic design flow of BDD-based PTL circuit synthesis.

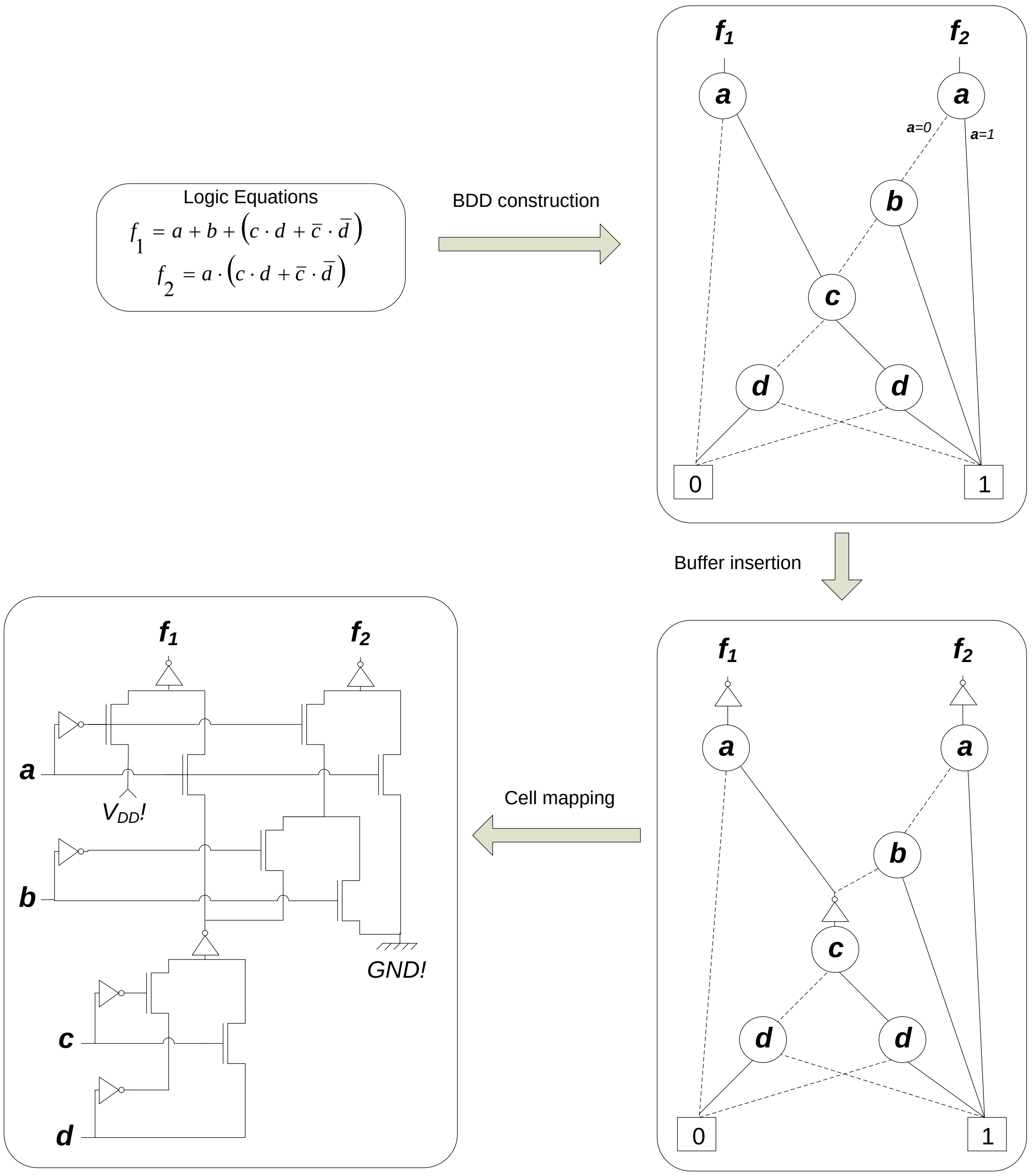

Logic Equations
$f_1 = a + b + (c \cdot d + \bar{c} \cdot \bar{d})$
$f_2 = a \cdot (c \cdot d + \bar{c} \cdot \bar{d})$
BDD construction
f_1
f_2
a=0
a=1
0
1
Buffer insertion
Cell mapping
$V_{DD}!$
GND!

Figure 2.4: Design flow of BDD-based PTL synthesis (YANO, 1996).

2.4 Branch-Based Logic

Branch-Based Logic (BBL) is a logic style that has been developed for low-power, low-voltage applications and for high-speed circuits (PIGUET, 1984; PIGUET, 1994; PIGUET, 1995; NÈVE, 2001). In this style, the transistor networks consist only of branches (i.e. a series of up to three transistors between power line and gate output).

The advantages of transistor branches are higher layout regularity (i.e. smaller diffusion capacitances) and simpler characterization (i.e. branch instead of gate modeling).

The construction of branch-based circuits is rather simple. It takes a flat (non-factorized) irredundant sum-of-products and translates each product into an and-stack (branch) in the circuit. It’s done for each PUN and PDN independently, using the on-set and the off-set logic expression respectively. Fig 2.5 shows an example of a BBL circuit.

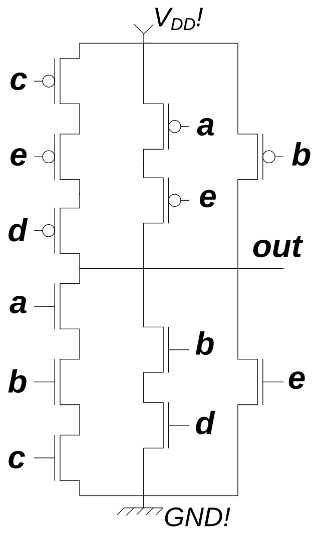


Figure 2.5: Branch-based circuit example for the function $out = a \cdot b \cdot c + b \cdot d + e$.

The low-power and high-speed capabilities of BBL circuits are due to its low capacitive nodes as there is no parallel connection among branches. This absence of interconnection among branches is also a positive characteristic from the layout point of view (PIGUET, 1984). However, one problem of BBL circuits is the use of too many transistors for some logic functions, as its logic expressions are flat (non-factorized) by nature.

2.5 Other Static Logic Styles

Several other logic styles have been proposed in order to explore different aspects of the area-speed-power design space. The choice depends on the design as well as on the tools available for the synthesis of the circuits.

An appealing approach for high-speed circuits is the BiCMOS family (ELRABAA, 1992) which uses bipolar transistors in order to achieve improved output drive capability. One of the biggest problems of this kind of logic is that the use of bipolar transistors is not very well supported for most of the fabrications processes and CAD tools available. Currently, BiCMOS is mostly used for bus drivers, I/O drivers and linear circuits like high-speed operational amplifiers (WESTE, 1993).

A couple of old yet very interesting works regarding to logic cells with minimal number of transistors are depicted in (GREA, 1958) and (NINOMIYA, 1965) which present tables with hand-crafted switch topologies for more complex logic cells.

Finally, one must cite the innovative approach developed by Zenasis Technologies (ZENASIS, 2007) so called Flex-cells (ROY, 2005). This approach was developed in order to cope with the gap between standard-cell based design and full-custom design, where it is estimated that automated design flows deliver circuits slower by at least a factor of 6 and consume a larger area at least by a factor of 10 (Chapter 10 of CHINNERY, 2002). Flex-cell is not a logic style, indeed, but it is cited here because it is an optimization approach done at transistor-level. In summary, the Flex-cell-based optimization consists in automating the process of creating tactical cells in a design, by grouping clusters of logic gates in a single gate, as shown in Fig. 2.6. With such approach it's expected to have significant improvements in area and speed without having to handcraft such tactical cells.

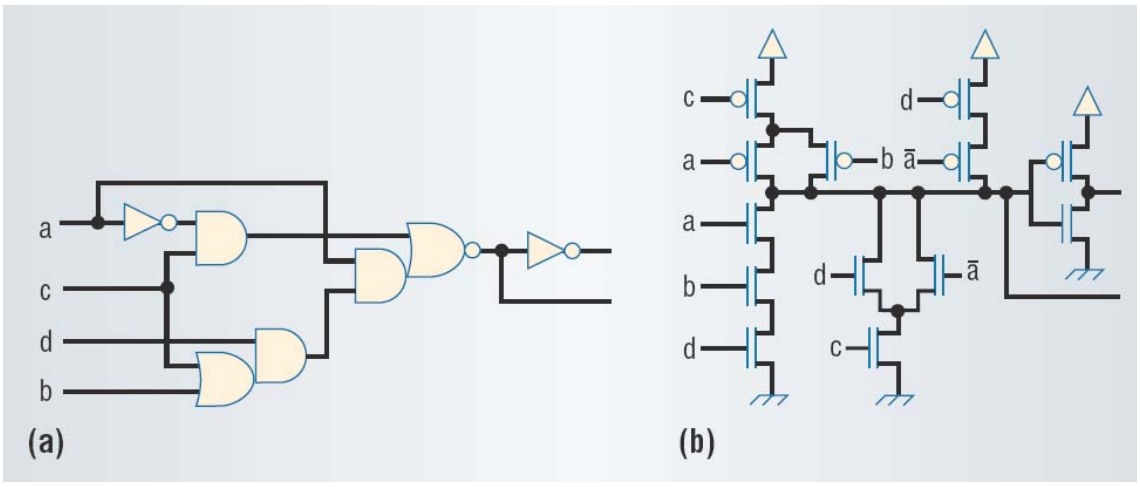


Figure 2.6: Figure 3. Flex-cell generation. Starting with (a) the original cluster of standard cells, the mapping process (b) creates a flex cell that replaces the cluster (ROY, 2005).

3 LOWER BOUND FOR STACKED SWITCHES

3.1 Introduction

The number of stacked switches (or switches in series) inside a cell is a limiting factor to the maximum speed it may attain in CMOS technologies. Regardless of the transistor topology used to implement a switching function, there is a strong correlation between the length of its longest transistor stack and its worst-case propagation delay. This correlation is verified because the switches along this path are likely to charge or discharge a path that corresponds to the worst-case delay scenario. This loss in delay is directly related with the logical effort (SUTHERLAND, 1999; KABBANI, 2005; WESTE, 2006; ROSA 2007) of the cell, as it will be demonstrated on Chapter 5. The approach in (SHELAR, 2001) made use of this correlation described above in its algorithm for performance-driven PTL synthesis. The method presented there exploits two separate effects. First, it aims to reduce the number of serially connected gates by applying functional decomposition. Second, it reduces the number of stacked transistors (switches) inside the gates by encoding decompositions with a one-hot code and deriving cell level PTL networks partially from a BDD (Binary Decision Diagram) and partially from a one-hot multiplexer. The results reported there show significant performance gains, proving the importance of the number of stacked transistors (switches) as a parameter to the quality of cell networks, especially when performance is the design goal.

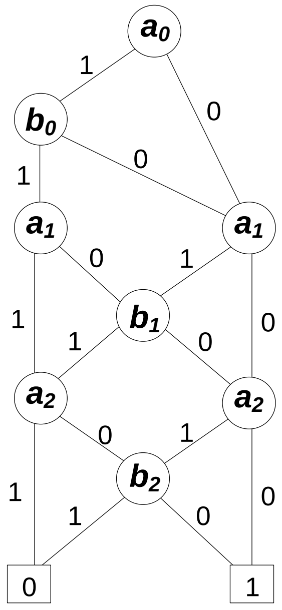
Synthesis techniques for PTL circuits have been closely related to BDD representation of logic functions, for reasons such as elimination of sneak paths and availability of efficient algorithms for the construction of BDDs (SHELAR, 2001). Indeed, the approaches in (BUCH, 1997; HSIAO, 2000; SCHOLL, 2000; LINDGREN, 2001; ZHOU, 2001; SHELAR, 2001; SHELAR, 2002; AVCI, 2003) are based on BDDs. Therefore, when discussing PTL networks in this work, it will be assumed in this work they are derived from BDDs. Despite the gains demonstrated by (SHELAR, 2001), this work will demonstrate that even the introductory example used there, the circuit *c3* (carry out for the first 3 bits of an adder), shown in Fig. 3.1, may be synthesized with a significantly smaller number of stacked transistors than originally presented. Fig. 3.1.a shows the BDD for *c3*. This BDD has a path with six arcs in series, and it would have six transistors in series if mapped as a single PTL gate. For this reason, PTL approaches will insert buffers to limit the number of transistors in series to three or four (SHELAR, 2001; ZHOU, 2001). Fig. 3.1.b shows a PTL gate for *c3*, with (inverting) buffers inserted. This way, the bottom part, below the first stage of buffers, has four transistors in series (counting the transistors inside the buffer that generates signal B2). On the other hand, the top part of the cell has three transistors in series (counting the transistors inside the buffers). Consequently, the PTL implementation in Fig. 3.1.b may then be viewed as two independent gates connected in cascade.

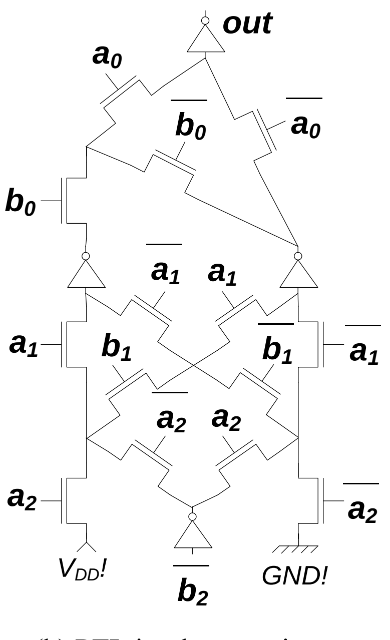
A Complementary Series/Parallel (CSP) CMOS (BERKELAAR, 1988; REIS, 1995; REIS, 1997; GAVRILOV, 1997) implementation of the circuit *c3* can be observed on Fig. 3.2.a, where there is a pull-down network with five transistors in series. This is not considered feasible from the electrical point of view as there is too much degradation of the signal in the discharge path due to increased resistance. Using CSP in this case would require the logic cell to be split into two or more stages.

However, one may notice that the *c3* function may be synthesized as a single cell where the pull-down network has at most four stacked transistors and the pull-up chain has at most three. This implementation is shown in Fig. 3.2.b, and it is 60% faster (according to SPICE simulations) than the fastest decomposed version presented in (SHELAR, 2001). Moreover, this is not a problem that is unique to the cell generation method in (SHELAR, 2001). Indeed, this seems to be true for several approaches based on PTL, as the are also examples of non-optimized pull-up and/or pull-down paths in other papers based on PTL logic (BUCH, 1997; ZHOU, 2001). Notice that although the circuit in Fig. 3.2.b is series/parallel CMOS, PUN and PDN are not topologically complementary. Therefore, this problem is not unique to the cell generation for PTL, since reducing (or controlling as a design parameter) the maximum length of transistor stacks in a circuit is important for most logic families. These observations lead us to formulate the following questions:

1) What is the minimum length for the PUN and PDN paths when designing a switch network for a given cell?

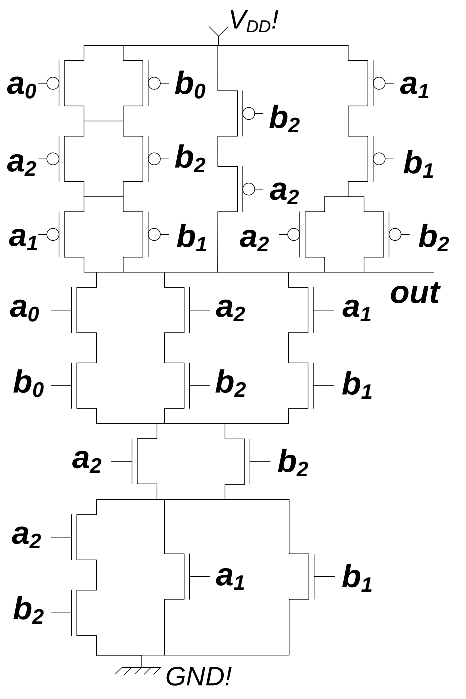
2) Is it possible to synthesize a network for the circuit in Fig. 3.2.b with shorter PUN and PDN path lengths?

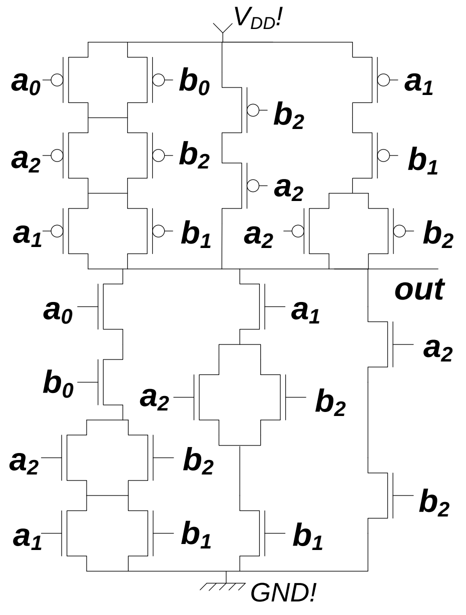




(a) BDD for function *c3* (b) PTL implementation

Figure 3.1: BDD and PTL implementation for function *c3*.





(a) Complementary (b) Non-complementary (designed by hand)

Figure 3.2: Series/parallel CMOS implementations of *c3*.

In the next Sections it will be addressed the question of deriving exact lower bounds for the number of switches in the longest PUN and PDN stacks inside a switch network for a logic cell.

3.2 Basic Concepts

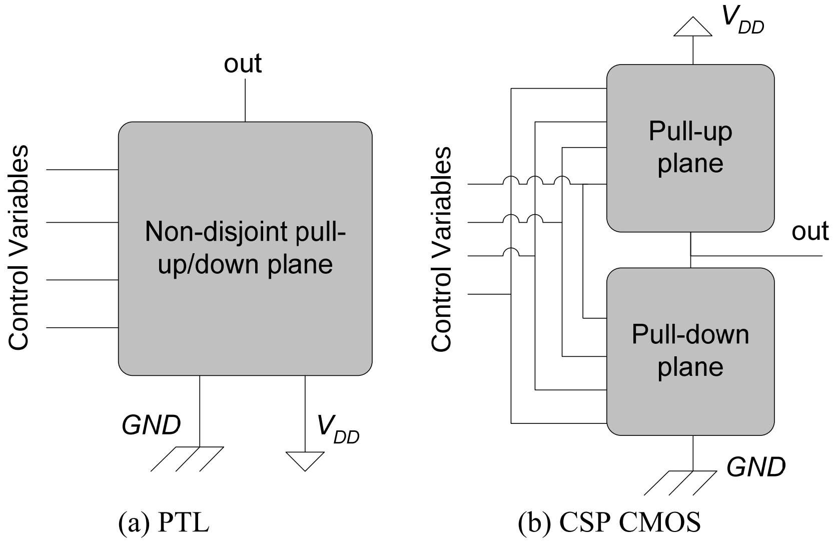
3.2.1 Boolean space and cube size

The Boolean set *B* is composed of the following elements *{0, 1}*. A Boolean variable may assume an arbitrary value in the set *B*. A *n*-dimensional Boolean space is defined through a set composed of *n* Boolean variables and is noted as $B^n = \{(a_0, a_1, \ldots, a_{n-1})\} | a_i \in \{0, 1\}\}$. The Boolean space B^n is composed of 2^n distinct points. A cube of B^n is a sub-space of B^n obtained through the assignment of specific values to a subset of variables in B^n. The assignment of values to *m* out of the *n* variables in B^n will denote a cube of size B^{n-m}. A cube of unitary size is said to be a minterm. If all variables have assigned values, a single point in the Boolean space is indicated. A Boolean function is a mapping $B^n \rightarrow B$, such that every point in B^n is mapped to one and only one value in *B*. The set of variables in B^n is the domain of the function. Boolean functions may be expressed through the Boolean algebra, composed of the following operations in the *B* set: *AND* (denoted by $\cdot$), *OR* (denoted by +) and *INVERSION* or *NOT* (denoted by a horizontal bar over the variable or function, like $\overline{a}$). Consider an ordering that states that *1>0*, over *B*. The *AND* of *n* Boolean variables is defined as the minimum value assumed by the input variables. The *OR* of *n* Boolean variables is defined as the maximum value assumed by the input variables. The *INVERSION* is a unary operator that returns the value in *B* that is different from the one assigned to the input. It is important to observe the operator precedence is *NOT* > *AND* > *OR*. A literal of an equation is an instance of a variable in the direct a_i or inverted $\overline{a_i}$ form. A specific cube of B^n may be expressed as a product of literals in the following manner: variables assigned to the value *1* appear as a direct literal a_i, while variables assigned to the value *0* appears as an inverted $\overline{a_i}$ literal. For instance the cube $a \cdot \bar{c} \cdot d$ represents the Boolean subspace where $a = 1$, $c = 0$ and $d = 1$. The subspace defined by this cube will have a size 2^{n-3}, with respect to a *n*-dimensional Boolean space B^n. A cube *C* is said to be an *on-set implicant cube* of a given Boolean function *f* if all the points in the subspace defined by *C* are mapped to *1* through the function *f*. An implicant cube is said to be an *on-set prime implicant* of function *f* if it is not contained in a distinct implicant cube of *f*. Similarly, a cube *C* is said to be an *off-set implicant cube* of a given Boolean function *f* if all the points in the subspace defined by *C* are mapped to *0* through the function *f*. An implicant cube is said to be an *off-set prime implicant* of function *f* if it is not contained in a distinct implicant cube of the off-set of function *f*.

3.2.2 Switches and logic cells

A switch controls the connection between two different points. The discussion in this paper will be restricted to two different kinds of switches, as described in the following. An *active-0* switch will connect two nodes if the control variable is equal to *0*; the switch will not connect these points when the control signal is equal to *1*. Similarly, an *active-1* switch will short-circuit two nodes if the control variable is equal to *1* and it will be an open circuit if the variable is *0*. PMOS transistors are active-*0* switches and NMOS transistors are active-*1* switches.

A logic cell that implements a given logic function is formed by a set of interconnected switches. These switches are controlled by the variables in the domain of the logic function. The main switch topologies used to design transistor networks for logic cells are Pass Transistor Logic (PTL) (BUCH, 1997; HSIAO, 2000; SCHOLL, 2000; LINDGREN, 2001; ZHOU, 2001; SHELAR, 2001; SHELAR, 2002; AVCI, 2003) and Complementary Series/Parallel (CSP) CMOS Logic (BERKELAAR, 1988; REIS, 1995; REIS, 1997; GAVRILOV, 1997). Fig. 3.3 illustrates these topologies. It is possible to notice that the PTL topology is composed of a single non-disjoint pull-up/down plane, while the CSP topology has two disjoint switch planes: one pull-up plane and one pull-down plane.



(a) PTL (b) CSP CMOS

Figure 3.3: PTL and CSP CMOS topologies.

Independently of the topology, the output of the cell is connected to V_{DD} or *GND* through a path composed of serially connected switches that are active (connected) under a given input assignment. A pull-up path connects the output of the cell to the V_{DD} (logic-*1*) reference, through a set of serially connected switches. A pull-down path connects the output of the cell to the *GND* (logic-*0*) reference, through a set of serially connected switches. A pull-up path is associated with an on-set implicant cube, while a pull-down path is associated with an off-set implicant cube. In this work it will be referred to a cell with longest pull-up chain *PU* and longest pull-down *PD* as being a *PU-PD* cell. For instance, a static CMOS 2-input NAND is a 1-2 cell.

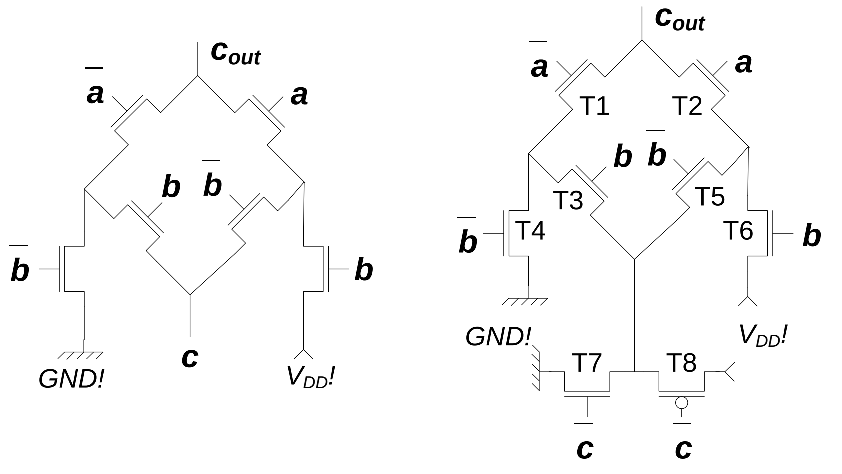
Example 1: Consider the PTL cells shown in Fig. 3.4, for the carry-out function of a full adder. The PTL cell in Fig. 3.4.a has the input *c* connected to transistor drains. However, inside an integrated circuit these signals will always be available through another cell that will generate them. In the best case, these signals will be generated through an inverter. Thus the following discussion will consider that drain signals are available through inverters. Fig. 3.4.b shows the PTL cell with the transistors corresponding to the inverters added to the transistor network. The path composed of the transistors T2-T5-T8 is a pull-up path and it is associated to the on-set implicant cube $a \cdot \overline{b} \cdot c$. The association between switches and literals follows the possibilities listed in Table 3.1: T2 corresponds to possibility *#3*, T5 corresponds to possibility *#4* and T8 corresponds to possibility *#2*. The other pull-up paths are T1-T3-T8 and T2-T6 corresponding to the on-set implicants $\overline{a} \cdot b \cdot c$ and $a \cdot b$. Similarly for the off-set, the path composed of the transistors T1-T3-T7 is a pull-down path and it is associated to the off-set implicant cube $\overline{a} \cdot b \cdot \overline{c}$. Again, the association between switches and literals follow the possibilities listed in Table 3.1: T1 corresponds to possibility *#4*, T3 corresponds to possibility *#3* and T7 corresponds to possibility *#4*. The other pull-down paths are T1-T4 and T2-T5-T7 corresponding to the on set implicants $\overline{a} \cdot \overline{b}$ and $a \cdot \overline{b} \cdot \overline{c}$. The on-set (off-set) is given by the sum of all on-set/off-set implicants corresponding to PUN/PDN paths, as given by equations (3.1)/(3.2) below. Notice that these equations are correct, but they are not prime covers. Prime covers for the on-set and off-set of this particular function would have cubes composed of at most two literals. The cell in Fig 3.4.b is a 3-3 cell, which is not the minimum transistor chain that may be achieved for this cell, as will be demonstrated later in this Chapter. Paths and associated cubes present in the switch network of Fig. 3.4.b are summarized in Table 3.2.

$$on-set = a \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot c + a \cdot b \tag{3.1}$$

$$off-set = \overline{a} \cdot b \cdot \overline{c} + \overline{a} \cdot \overline{b} + a \cdot \overline{b} \cdot \overline{c} \tag{3.2}$$

Table 3.1: Literals in cubes associated to paths.

Possibility number	Switch type	Literal in the switch	Literal in the cube
1	Active-*0*	ai	$\overline{ai}$
2	Active-*0*	$\overline{ai}$	ai
3	Active-*1*	ai	ai
4	Active-*1*	$\overline{ai}$	$\overline{ai}$



(a) drain inputs (b) strong signals through inverters

Figure 3.4: Two distinct PTL cells.

Table 3.2: Pull-up and pull-down paths for the PTL-based network shown in Fig. 3.4.b.

Type	Transistors	Cube
Pull-up	T1-T3-T8	$\overline{a}\cdot b\cdot c$
	T2-T5-T8	$a\cdot\overline{b}\cdot c$
	T2-T6	$a\cdot b$
Pull-down	T1-T4	$\overline{a}\cdot\overline{b}$
	T1-T3-T7	$\overline{a}\cdot b\cdot\overline{c}$
	T2-T5-T7	$a\cdot\overline{b}\cdot\overline{c}$

3.3 Exact Lower Bound for Stacked Switches

The lower bound for the number of stacked switches proposed hereby is based on the number of literals of the smallest cube in a prime and irredundant cover (set of prime implicants that covers a function and where each prime implicant on the set is not covered any other one in the same set). The problem with this is that if a function might have distinct prime and irredundant covers with a different number of literals in the smallest cube, then the lower bound would not be univocally defined. In the following, it is presented a proof to ensure that this condition will never happen as the size of the smallest cube in distinct prime irredundant covers of a logic function is univocally defined.

Definition 1: A cube with m literals will have cube size 2^{n-m} in Boolean space B^n. By definition, the smallest cube is the cube with larger number of literals.

Theorem 1: The number *m* of literals in the smallest cube does not change for distinct prime implicant covers of the same logic function.

Proof (by contradiction): Consider two distinct prime and irredundant covers *C1* and *C2* of the same function *f* such that the smallest cubes in *C1* and *C2* have different sizes. Suppose that cover *C1* has smallest cube(s) composed of *m* literals. Suppose also that the smallest(s) cube(s) in cover *C2* are composed of *m-i* literals, such that $0 < i < m$. To turn on any prime irredundant cube in *C1*, it is necessary to assign at most *m* variables to logic-*0* or logic-*1* as appropriate. The reason for this is because the smallest cube in *C1* is not redundant and has *m* literals. However, assigning at most *m-i* variables is sufficient to turn on any prime implicant cube in *C2*. Thus, the analysis of the complete set of all variable assignments containing *m-i* or fewer variables in B^n is sufficient to decide if the function represented by *C2* evaluates to logic-*1*. This same analysis is not sufficient, nonetheless, to decide if the function represented by *C1* evaluates to logic-*1*, due to the irredundant prime implicant(s) with *m* literals. Therefore, the functions given by covers *C1* and *C2* are not the same logic function, leading to a contradiction of our initial hypothesis that two distinct prime implicant covers of the same function could have smallest cubes with different sizes. QED.

Corollary: Theorem 1 is valid for two prime irredundant covers. Non-prime covers may have smaller non-prime cubes with more than *m* literals, where *m* is the size of the smallest cube in any prime irredundant cover. However, no cover may have only cubes greater (with a smaller number of literals) than the smallest cube in any prime irredundant cover.

Definition 2: A PUN path connects the output of the cell to the V_{DD} (logic-*1*) reference, through a set of serially connected switches.

Theorem 2: Given a function *f*, it is not possible to have a cell whose longest PUN path has fewer switches than *m*, being *m* the number of literals in the smallest cube of any prime irredundant cover *C* for the on-set of function *f*.

Proof (by contradiction): Recall that the smallest cube has the greater number of literals (Definition 1). By Theorem 1, all the prime irredundant covers of function *f* will have at least one cube with *m* literals, where *m* is the size of the smaller cube in any prime cover of *f*. Non-prime covers of *f* may have smaller cubes with more than *m* literals. However, covers where all the cubes have less than *m* literals are not possible, due to Theorem 1 and its Corollary. Consider a function *f* defined in B^n, such that a prime irredundant cover *C* of *f* has *m* literals. Suppose now that function *f* has a generic switch realization where the i^{th} pull-up path has size p_i, such that $p_i < m$. As described in detail through Example 1 of Subsection 3.2.2, each of these PUN paths would be associated with an on-set implicant cube with p_i literals. As a consequence, every PUN path with size $p_i < m$ will produce a cube with p_i literals where $p_i < m$. The cover obtained from the network this way will have only cubes with less than *m* literals, as $p_i < m$, for every path. According to Theorem 1, this contradicts the initial hypothesis that the smallest cube in the prime implicant cover *C* has *m* literals, as all the cubes in the realization would have a smaller number of literals $p_i < m$ and a greater size. QED.

Theorem 3: Given a function *f*, it is possible to produce a cell where the longest PUN path has *m* switches in series, being *m* is the number of literals in the smallest cube in a prime irredundant cover *C* for the on-set of function *f*.

Proof (by construction): It is possible to construct a pull-up plane for function *f* given a prime irredundant cover $C=\sum P_i$ of the function, where each cube $P_i=\prod l_i$ is a product of literals associated to the variables in the domain of the function. Every prime cube P_i contributes to the PUN with an independent path. The paths for each cube P_i are independent as they are parallel paths among each other. Each of the independent paths is composed of serially connected switches between the logic-*1* reference (V_{DD}) and the output of the cell. The path for a given cube in the cover contains one serially connected switch for each literal l_i in the cube, as described by Example 1 in Subsection 3.2.2. As the smallest cube has the greater number of literals, it will determine the size of the longest path. Thus this implementation will have by construction the longest PUN path with a size correspondent to the number m of literals in the smallest cube of C. Furthermore, as the path for each cube is independent, this solution has no sneak paths. QED.

Theorem 4: The exact lower bound in the number of stacked switches (or switches in series) in the longest PUN path of a logic function f is given by m, the number of literals in the smallest cube in any prime irredundant cover C for the on-set of function f.

Proof: Immediate Corollary of Theorems 2 and 3, and univocally defined as consequence of Theorem 1. QED.

Definition 3: A PDN path connects the output of the cell to the *GND* (logic-*0*) reference, through a set of serially connected switches.

Theorem 5: Given a function f, it is not possible to have a cell whose longest PDN path has fewer switches than m, the number of literals in the smallest cube of any prime irredundant cover C for the off-set of function f.

Proof: The proof is similar to that of Theorem 2. QED.

Theorem 6: Given a function f, it is possible to produce a cell where the longest PDN path has m switches in series, where m is the number of literals in the smallest cube in a prime irredundant cover C for the off-set of function f.

Proof: The proof is similar to that of Theorem 3. QED.

Theorem 7: The exact lower bound in the number of stacked switches in the longest PDN path of a logic function f is given by m, the number of literals in the smallest cube in any prime irredundant cover C for the off-set of function f.

Proof: Immediate corollary of Theorems 5 and 6, and univocally defined as consequence of Theorem 1. QED.

3.4 Evaluating the lower bound

At first, it may seem too time-intensive to calculate the lower bound for candidate functions, as it is necessary to calculate two prime ISOPs (Irredundant Sums-of-Products). However, as one evaluate only the PUN and PDN stacks inside cells, the evaluation process is not a critical step because it is easy to obtain prime ISOPs when the number of variables is small. For this purpose a BDD-based ISOP function (MINATO, 1996) may be used.

For the sake of simplicity, next Chapter will extract the ISOPs used to determine the lower bound for the number of transistors in PUN and PDN stacks using the well-established tabular method so called Quine-McCluskey (QUINE, 1955; MCCLUSKEY, 1956; MICHELI, 1994).

3.5 Consequences and Applications

This Section presents consequences and applications of the lower bounds introduced in previous Sections.

3.5.1 Lower bound impact analysis

For a better understanding of the impact of the lower bound theory for stacked switch paths length, some illustrative examples are presented.

Example 2: Consider a function *f* given by equation $f = a \cdot b + b \cdot c + a \cdot c \cdot d$. The minimum covers for the on-set and the off-set of this function are:

$$on-set = a \cdot b + b \cdot c + a \cdot c \cdot d \tag{3.3}$$

$$off-set = \overline{a} \cdot \overline{b} + \overline{a} \cdot \overline{c} + \overline{b} \cdot \overline{d} + \overline{b} \cdot \overline{c} \tag{3.4}$$

The smallest cube in the on-set is $a \cdot c \cdot d$, so the lower bound for the number of stacked transistors in the PUN is three. The cubes in the off-set are all the same size, and the lower bound for the number of stacked switches in the PDN is two. This way the cell corresponding to the function *f* is a 3-2 cell, when mapped with the constructive method proposed in the next Chapter. It could also be a 2-3 cell, if the function is inverted and an inverter is added to the cell output.

Example 3: Recall the function in Example 1, the carry-out of a full adder. Prime irredundant covers for the on-set and off-set are given by the following equations.

$$on-set = a \cdot b + a \cdot c + b \cdot c \tag{3.5}$$

$$off-set = \overline{a} \cdot \overline{b} + \overline{a} \cdot \overline{c} + \overline{b} \cdot \overline{c} \tag{3.6}$$

It is easy to see that the lower bounds for the carry-out in a full adder are two switches for both the pull-up and pull-down planes. This is consistent with the classic 2-2 cell for carry-out generation presented in the cover of the classic Weste-Eshraghian book (WESTE, 1993).

Example 4: What is the minimum number of transistor in series to implement the function *c3*? The minimum covers are given by the following equations.

$$\begin{aligned} on-set = {} & a2 \cdot b2 + a2 \cdot a1 \cdot b1 + b2 \cdot a1 \cdot b1 + a2 \cdot a1 \cdot a0 \cdot b0 + \\ & a2 \cdot b1 \cdot a0 \cdot b0 + b2 \cdot a1 \cdot a0 \cdot b0 + b2 \cdot b1 \cdot a0 \cdot b0 \end{aligned} \tag{3.7}$$

$$\begin{aligned} off-set = {} & a2 \cdot b2 + a2 \cdot a1 \cdot b1 + a2 \cdot a1 \cdot a0 + a2 \cdot a1 \cdot b0 + \\ & a2 \cdot b1 \cdot a0 + a2 \cdot b1 \cdot b0 + b2 \cdot a1 \cdot b1 + b2 \cdot a1 \cdot a0 + \\ & b2 \cdot a1 \cdot b0 + b2 \cdot b1 \cdot a0 + b2 \cdot b1 \cdot b0 \end{aligned} \tag{3.8}$$

Equation (3.7) has 4 literals in the smallest cube. Equation (3.8) has 3 literals in the smallest cube. This way, the circuit presented in Fig. 2 cannot be designed with shorter transistor chains, as it is clear that minimum transistor chain version of *c3* is either a 3-4 cell or a 4-3 cell, depending on polarity assignment. This is the main goal of the lower bound proposed here, to verify and ensure that performance is not being lost due to the misuse of cells with excessively long transistor stacks. The CSP implementation from equation (3.7) would be 4-7 cell, while the CSP implementation from equation (3.8) would be a 3-11 cell. The PTL cell would be a 6-6 cell.

3.5.2 Lower bound and general PTL styles

Notice that the lower bounds defined here apply to general PTL styles, due to the definition of PUN and PDN paths in this work. In the following it is treated a pitfall counterexample.

Example 5: Consider a 2-input AND function given by equation $a{\cdot}b$. The on-set and off-set prime irredundant covers are given by the following equations.

$$on-set = a \cdot b \tag{3.9}$$

$$off-set = \overline{a} + \overline{b} \tag{3.10}$$

From these equations, this cell should be implemented as a 1-2 cell. However, the network in Fig. 3.5 implements a 2-input AND with (apparently) only one transistor for PUN and PDN. The pitfall in this counterexample network is that it ignores the definition of PUN and PDN paths (Definitions 2 and 3). The path from drain input *b* to the output *out* is not a valid pull-up or pull-down path as node *b* is not a power supply (V_{DD} or *GND*). As node *b* may assume both logic values, at least one switch is necessary to connect/disconnect it to V_{DD} as well as to *GND*. This extra switch should be added to the length of the longest PUN and PDN paths. As a consequence, the lower bound is still valid, according to the given Definitions (2 and 3) of PUP and PDN paths, as the cell in Fig. 3.5 becomes a 2-2 cell, which is larger than the 1-2 lower bound for this function.

For completeness, it is presented the following Theorem for the general case where PTL with drain inputs are admitted, though as it will soon be obvious, this is not a practically useful result.

Theorem 8: In a general PTL topology, where drain inputs are admitted, every circuit may be synthesized with cells with at most one switch in series.

Proof (by construction): If a cell (set of switches S) has $n > 1$ switches in series, partition the set of switches S in two sets S1 and S2 such that S1 has only one switch in series and S2 has *n-1* switches in series. Repeat the procedure until every cell in the circuit has only one switch in series. QED.

Observation: Theorem 8 is correct under its assumptions, but it is not very useful from the electrical point of view, as it ignores the (lack of) strength of the signals in the drain inputs of PTL logic. The Theorems presented in Section 3.3, with PUN and PDN paths starting at strong power supplies are more practical for real circuits.

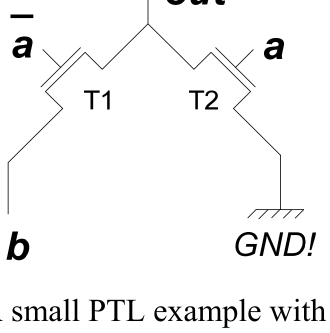


Figure 3.5: A small PTL example with drain inputs.

3.5.3 Lower bound and general non-SP logic styles

It is important to notice that the way the lower bound was derived is independent from circuit topology and may be applied to any transistor network. This includes the non-SP (non-series/parallel) ones presented as "minimal networks" in (GREA, 1958) and (NINOMIYA, 1965). For instance, the same analysis may be applied to bridge-based circuits, as illustrated through the following example.

Example 6: Consider the bridge-based circuit in Fig. 3.6. The PUN and PDN paths are listed in Table 3.3, as well as the associated cubes. Some cubes do not contribute to the cell functionality, so that they may be ignored in the on-set and off-set covers given by equations (3.11) and (3.12). Notice that equations (3.11) and (3.12) are logic equivalents to equations (3.4) and (3.3) respectively, from Example 2. Equations (3.3) and (3.4) represent the lower bound for this logic function, as they are prime irredundant cover. The bridge circuit does not respect the lower bound and this way some paths (T1-T4-T6) will contribute with non-prime cubes ($\overline{a} \cdot c \cdot \overline{b}$). This way, the cell in figure is topologically a 3-4 cell and logically a 3-3 cell. Both possibilities are worst than the optimal 2-3 cell presented in Fig. 4.1.b for this function.

$$on-set = \overline{a} \cdot \overline{c} + \overline{a} \cdot c \cdot \overline{b} + \overline{c} \cdot \overline{b} + \overline{d} \cdot \overline{b} \tag{3.11}$$

$$off-set = c \cdot b + a \cdot \overline{c} \cdot b + a \cdot c \cdot d \tag{3.12}$$

Table 3.3: Pull-up and pull-down paths for the bridge-based network shown in Fig. 3.6.

Type	Transistors	Cube
Pull-up	T1-T5	$\overline{a} \cdot \overline{c}$
	T1-T4-T6	$\overline{a} \cdot c \cdot \overline{b}$
	T2-T4-T5	$\overline{c} \cdot c \cdot \overline{c}$ (does not affect functionality)
	T3-T4-T5	$\overline{d} \cdot c \cdot \overline{c}$ (does not affect functionality)
	T2-T6	$\overline{c} \cdot \overline{b}$
	T3-T6	$\overline{d} \cdot \overline{b}$
Pull-down	T7-T10	$c \cdot b$
	T7-T9-T11-T12	$c \cdot \overline{c} \cdot c \cdot d$ (does not affect functionality)
	T8-T9-T10	$a \cdot \overline{c} \cdot b$
	T8-T11-T12	$a \cdot c \cdot d$

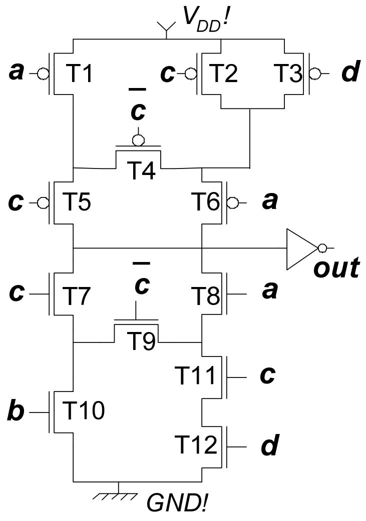


Figure 3.6: Bridge based 3-3 cell for function *f* from Example 2.

Notice that the analysis used to derive the lower bound is independent of circuit topology. Therefore the lower bound hold for every switch network independently of the chosen topology (SP, PTL, NCSP, bridge-based or non-planar topologies).

3.5.4 Applicability and compatibility with decomposition methods

The introduced lower bound has other applications, which include: 1) the evaluation of the quality of cells generated by different methodologies; and 2) the guidance for functional decomposition based technology mapping methods (BUCH, 1997; HSIAO, 2000; SHELAR, 2001; SCHOLL, 2001; LINDGREN, 2001; SHELAR, 2002; ROY, 2005), where the lower bound may be used to evaluate the complexity and the feasibility of different alternative sub-functions.

Another important conclusion is that, due to the fact that PTL logic exceeds the lower bound frequently, many PTL circuits and synthesis tools may benefit from the substitution of PTL style cells for NCSP cells that respect the introduced lower bound. In this sense, the lower bound and the NCSP topology are compatible with the decomposition methods mentioned. As PTL and CSP are widely used, the use of the NCSP topology proposed in this paper may imply a significant impact in the design of high performance integrated circuits.

3.5.5 Factorization and stacked transistors

The factorization (BRAYTON, 1982; BRAYTON, 1984; HACHTEL, 1996) does not affect the number of transistors in series. However, it affects the number of parallel branches in a network. As a consequence, factorization may reduce the number of transistors in series in the dual of a series/parallel network. Consider the example given by the following on-set and off-set equations:

$$\begin{aligned} on - set = {} & a2 \cdot b2 + a2 \cdot a1 \cdot b1 + b2 \cdot a1 \cdot b1 + a1 \cdot a0 \cdot b0 + \\ & + a2 \cdot b1 \cdot a0 \cdot b0 + b2 \cdot a1 \cdot a0 \cdot b0 + b2 \cdot b1 \cdot a0 \cdot b0 \end{aligned} \tag{3.13}$$

$$\begin{aligned} off - set = {} & \overline{a2} \cdot \overline{b2} + \overline{a2} \cdot \overline{a1} \cdot \overline{b1} + \overline{a2} \cdot \overline{a1} \cdot \overline{a0} + \overline{a2} \cdot \overline{a1} \cdot \overline{b0} + \\ & + \overline{a2} \cdot \overline{b1} \cdot \overline{a0} + \overline{a2} \cdot \overline{b1} \cdot \overline{b0} + \overline{b2} \cdot \overline{a1} \cdot \overline{b1} + \overline{b2} \cdot \overline{a1} \cdot \overline{a0} + \\ & + \overline{b2} \cdot \overline{a1} \cdot \overline{b0} + \overline{b2} \cdot \overline{b1} \cdot \overline{a0} + \overline{b2} \cdot \overline{b1} \cdot \overline{b0} \end{aligned} \tag{3.14}$$

Equation (3.13) has four literals in the smallest cube. Equation (3.14) has three literals in the smallest cube. This way, the transistor network for equations (13) and (14) is either a 3-4 cell or a 4-3 cell depending on polarity assignment. Without factorization, the series/parallel implementation from equation (3.13) would be 4-7 cell, while the series/parallel implementation from equation (3.14) would be a 3-11 cell. The equation (3.14) can be factorized into equation (3.15). Equation (3.15) may be used to implement a complementary series/parallel transistor network that respects the lower bound of Theorems 1 and 2:

$$off - set = (\overline{a2} + \overline{b2}) \cdot (\overline{a1} \cdot \overline{b1} + (\overline{a0} + \overline{b0}) \cdot (\overline{a1} + \overline{b1})) + (\overline{a2} \cdot \overline{b2}) \tag{3.15}$$

In the example above, the use of factorization allowed to achieve a series/parallel implementation that respects the lower bound. However, there are examples in which factorization may reduce the overall number of transistors, but it will not be sufficient to guarantee that the lower bounds of Theorems 2 and 5 are achieved with CSP implementations.

3.5.6 Unawareness of lower bound in the number of stacked transistors

Even if the lower bound in the number of stacked transistors presented by Theorems 2 and 5 seems rather obvious, this fact is largely ignored in the current literature, as it is illustrated by Table 3.4. Methods presented in (SHELAR, 2003), (BUSH, 1997) and (YANG, 2002) decompose functions whose PUN and PDN chains would allow them to be implemented as a single cell with timing and power advantages. Methods presented in (SCHOLL, 2000), (AVCI, 2003), (JIANG, 2001) and (POLI, 2003) present drawings of functions with non-optimal transistor stacks. The transistor count advantage obtained by (BUCH, 1997) is due to the use of NMOS only planes (conducting ones). The same is true for (SHELAR, 2005) and (SCHOLL, 2000), even if they lose in the transistor count. The low transistor count of (JIANG, 2001) is due to the use of drain inputs, which may lead to weak signals. The transistor count for functions *g* and *h* in (YANG, 2002) is combined because they have shared logic.

Table 3.4: Unawareness of the lower bound (#TR – number of transistors).

	Previous papers				Lower bound		
Reference	**Fig.**	**PUN**	**PDN**	**#TR**	**PUN**	**PDN**	**#TR**
(SHELAR, 2005)	#2	6	6	30	3	4	22
(BUCH, 1997)	#9.a	4	4	18	3	4	22
(SCHOLL, 2000)	#2	3	3	14	2	2	10
(AVCI, 2003)	#2	3	3	16	2	2	12
(JIANG, 2001)	#10	4	4	8	2	2	10
(POLI, 2003)	#6	4	4	8	3	3	22
(YANG, 2002)	#14.a(g)	5	5	32	4	4	32
(YANG, 2002)	#14.a(h)	5	5	16	3	3	16

3.6 Cell Enumeration

A table that is recurrently used to describe the number of logic functions with an exact maximum number of PMOS and NMOS transistors in series is shown in Table 3.5. This table is implicitly used for most methods aiming the use of large libraries, meaning that what is exploited is the universe of topologically complementary series/parallel gates presented in Table 3.5. The construction of Table 3.5 (DETJENS, 1987) has the underlying assumption of considering only negative-unate functions. The choice of negative-unate functions is justified by the fact that Complementary CMOS gates are negative gates by nature. The computation of the number of stacked transistors is made in a series/parallel association, due to the easy of computation.

Another common enumeration of feasible functions to exploit large libraries is a limitation in the number of inputs of the logic function. The huge number of functions is reduced by employing the equivalence under permutation of inputs – P-class equivalence – or equivalence under permutation of inputs and input/output negation – NPN-class equivalence – (SASAO, 1999). A cell in a library is capable of implementing all logic functions equivalent under input permutation. That means the same cell is able to implement functions $f = \overline{a \cdot b + c}$ and $f = \overline{a \cdot c + b}$, even if they are different functions.

Table 3.5: Number of functions with limited stacked transistors.

		Number of Stacked PMOS Transistors			
		1	**2**	**3**	**4**
Number of Stacked NMOS Transistors	**1**	1	1	1	1
	2	1	4	10	23
	3	1	10	58	285
	4	1	23	285	2798

Table 3.6: Number of functions, P- and NPN-classes until 4 inputs.

# of inputs	**1**	**2**	**3**	**4**
# of functions	4	16	256	65536
# of P-classes	4	12	80	3984
# of NPN-classes	2	4	14	222

3.6.1 Orthogonality of Enumerations

The enumerations presented in Tables 3.5 and 3.6 are orthogonal, in the sense that they are produced using different criteria. Fig. 3.7 illustrates this idea, by showing the number of common functions between the 3503 functions with 4 or less stacked transistors in Table 3.5 (3503 is the sum of all the numbers in Table 3.5) and the 3984 P-classes (SASAO, 1999) of Table 3.6; there are only 17 common functions. This happens because 3486 out of 3503 functions have more than four inputs. The same comparison is made for the 222 NPN-classes of Table 3.6. Again, the intersection between the two sets is 17 functions, under the supposition that negative unate functions will have the preference to represent their NPN-class. Notice that Table 3.5 refers only to negative-unate functions, while Table 3.6 has no restriction on polarity (positive- or negative-unate, as well as binate functions are allowed). To have a more complete library, both categories should be considered together. The merit of the evaluation method from Theorems 2 and 5 is that it is able to detect correctly the costs for the functions belonging to the union of all the sets represented in Fig 3.7.

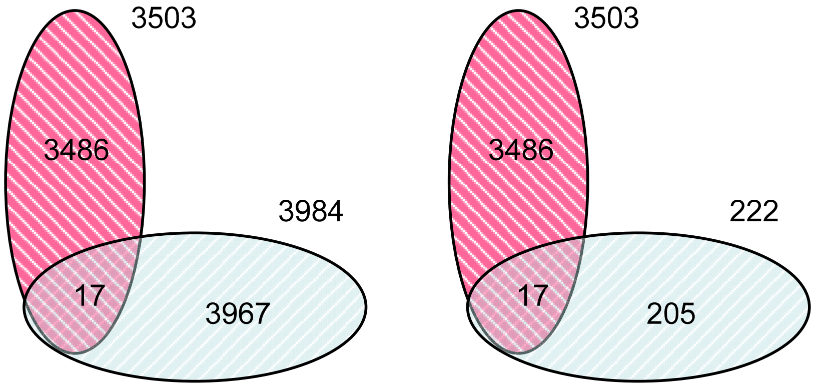


Figure 3.7: Relationship between enumerations.

3.6.2 Updating number of functions with limited stacked transistors

The seven functions listed in Table 3.5 for the case of a maximum of 2 NMOS and 2 PMOS transistors in series are the following:

$$f=\overline{a},\ f=\overline{a\cdot b},\ f=\overline{a+b},\ f=\overline{a\cdot b+c},\ f=\overline{(a+b)\cdot c},\ f=\overline{(a+b)\cdot(c+d)},\ f=\overline{a\cdot b+c\cdot d}.$$

It happens because the construction of Table 3.5 considers only CSP networks. The lower bound presented in Theorems 2 and 5 has smaller value for some functions, as it is the case of functions $f=\overline{a\cdot b+a\cdot c+b\cdot c}$ and $f=\overline{a\cdot b+a\cdot c+b\cdot d}$. This way, there are in fact 6 negative unate functions with exactly 2 transistors in series. The extra unate P-classes obtained by considering Theorems 2 and 5 for all the 6-input functions are presented in Table 3.7. This work limits the search of unate functions to 6-input functions due to the complexity of the method to find all P-equivalent classes. The functions in Table 3.7 are not listed in Table 3.5, once that this is a huge set of possible functions that has been ignored so far.

Table 3.7: Number of extra negative-unate P-classes with limited stacked transistors, considering Theorems 2 and 5.

		Number of Stacked PMOS Transistors			
		1	2	3	4
Number of Stacked NMOS Transistors	1	0	0	0	0
	2	0	+2	+13	+62
	3	0	+13	+498	+2897
	4	0	+62	+2897	+2222

3.6.3 Binate Function Modeling

As an effect of the method for computing stacked transistors described by Theorems 1 and 2, the computation of transistors in series can also be executed for functions that are binate, leading to a correct estimation of the number of serial transistors needed. The case of binate functions cannot always be reduced to the unate case were each polarity of a input is considered a different variable.

For instance, consider the network for a 3-input XOR presented in Fig. 3.8. If this network is considered as a network where a and $\overline{a}$ (b and $\overline{b}$, c and $\overline{c}$) are considered distinct variables, the output could connect to V_{DD} and *GND* at the same time. That means that Tables 4 and 6, when considering only unate functions, discard useful implementations of binate functions. This is the case of the 3-input XOR, implemented using CMOS from BDDs (POLI, 2003) of Fig. 3.8, which could be used to calculate the sum of three bits. It is not feasible as a single unate gate.

Table 3.8 presents the number of functions, with no polarity restrictions and limited number of transistors in series, considering Theorems 2 and 5. For Table 3.5 the search was limited to 4-input functions due to the complexity of the method.

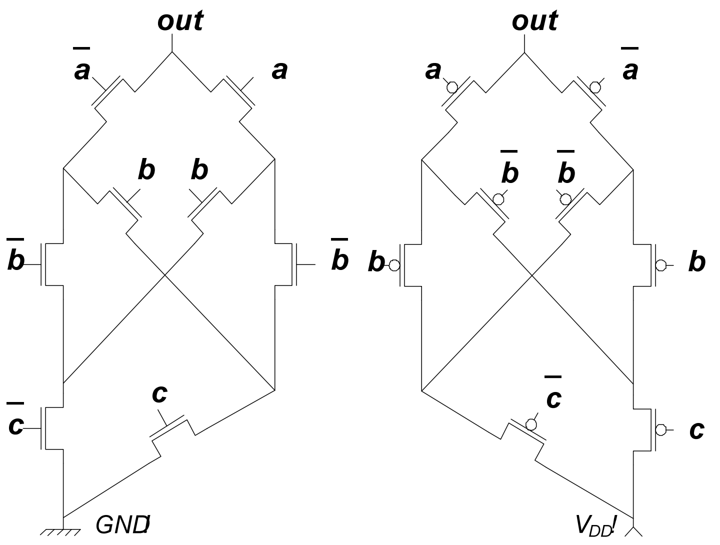


Figure 3.8: Transistor implementation for a 3-input XOR.

Table 3.8: Number of 4-input binate NPN-classes that would not be implemented with CSP logic.

		Number of Stacked PMOS Transistors			
		1	**2**	**3**	**4**
Number of Stacked NMOS Transistors	**1**	0	0	0	0
	2	0	0	0	0
	3	0	4	39	0
	4	0	2	47	31

4 NCSP LOGIC STYLE

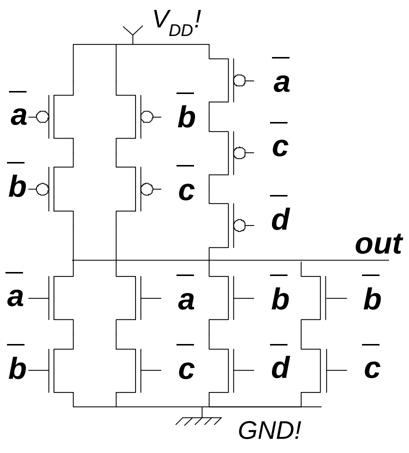
4.1 Introduction

In the previous Chapters it was demonstrated that the logic styles currently available and its associated constructive method do not take into consideration the lower bound for the number of transistors in series. In this Chapter it is presented in detail the constructive method for a novel logic style where both PUN and PDN respect the lower bound fully described on Chapter 3. As it is about to be demonstrated, respecting such constraint in the number of transistors in series requires sometimes PUN and PDN not to be topologically complementary, although it is still logically complementary. Henceforth, this work refers to the new logic style and its constructive method as Non-Complementary Series/Parallel (NCSP) logic style.

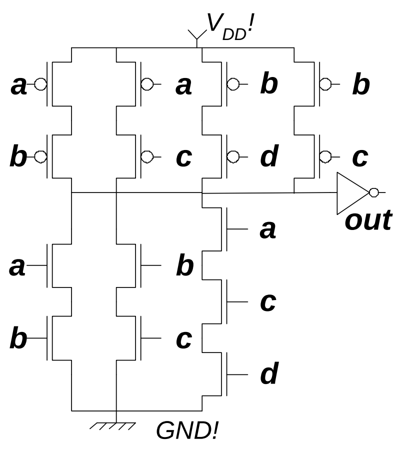
4.2 Constructive method

The constructive method for a NCSP cell was detailed in the proof of Theorems 3 and 6. As an example, consider the resulting cell for the function *f* in Example 2, presented on Chapter 3. The NCSP cell has the PUN derived from the on-set equation (hence the longest pull-up path has 3 switches) and the PDN derived from the off-set equation (thus the longest pull-down path has 2 switches), as it may be observed in Fig. 4.1.a. As the on-set and the off-set may be interchanged by inverting the function and adding an inverter at the output, it is always possible to use the smaller constraint in the pull-up network. This is desirable because PMOS transistors are more resistive than NMOS ones. The NCSP network considering the inverted version of function *f* is shown in Fig. 4.1.b.

One must notice that the networks could have the transistors count reduced by factorization (BRAYTON, 1982; BRAYTON, 1984; HACHTEL, 1996). Nevertheless, the number of stacked transistors would not change at all.



(a) 3-2 cell for direct *f*.

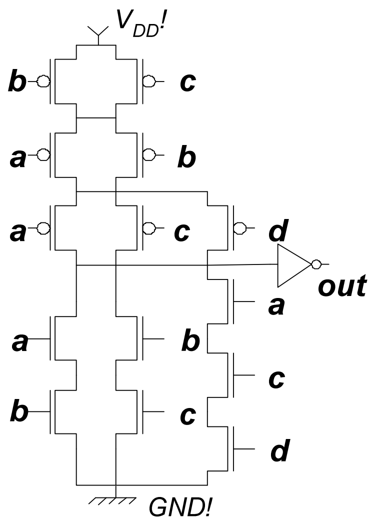


(b) 2-3 cell for inverted *f*.

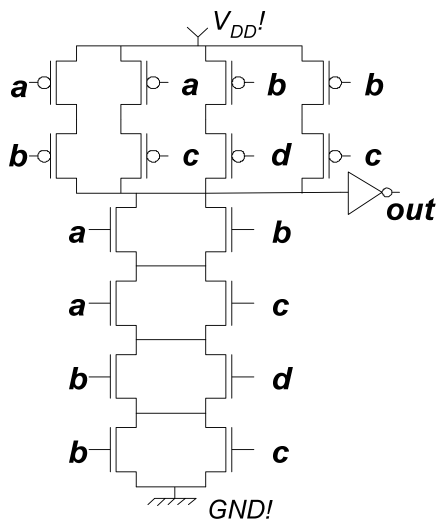
Figure 4.1: NCSP cells respecting the lower bounds for function *f* from Example 2.

4.3 Comparison with a CSP topology

A Complementary Series/Parallel (CSP) derived only from the on-set equation would result in a 3-3 cell, as shown in Fig. 4.2.a. Similarly, if a CSP cell were derived exclusively from the off-set equation, the result would be the 2-4 cell illustrated in Fig. 4.2.b. Thus, the use of the lower bound will produce a cell that has shorter pull-up and pull-down networks than CSP. Based on Example 2, it is possible to see that the NCSP topology propose hereby may be used to reduce the length of pull-up and pull-down longest chains when implementing cell level networks.



(a) 3-3 cell from on-set equation



(b) 2-4 cell from off-set equation

Figure 4.2: CSP cells not respecting the lower bounds for function *f* from Example 2.

4.4 Comparison with a PTL topology

The Pass Transistor Logic (PTL) realization of the cell from Example 2 would be a 4-4 cell, independently of the BDD variable order used to generate the PTL network (NAGAYAMA, 2004). One possible PTL network is shown in Fig. 4.3. This way, the use of the lower bound will produce a cell that has smaller pull-up and pull-down networks than PTL. Based on Example 2, it is possible to see that the NCSP proposed may be used to reduce the length of pull-up and pull-down chains when implementing cell level networks.

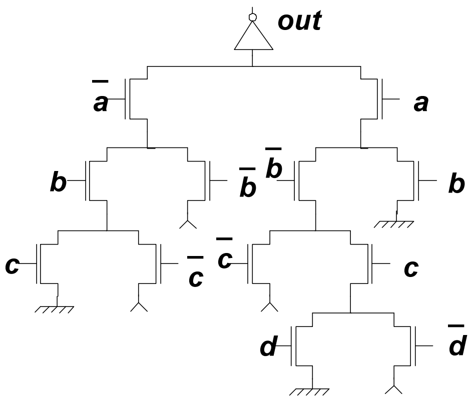


Figure 4.3: 4-4 PTL cell not respecting the lower bounds for function *f* from Example 2.

4.5 Algorithm for generating NCSP topologies

This Section depicts the NCSP logic style topology generation algorithm. The diagram in Fig. 4.4 represents the flow of the algorithm. In order to fully illustrate how the algorithm works, a pair of examples will be presented. In both examples, where NCSP networks are generated from different logic functions, the CSP counterparts will also be presented for comparison.

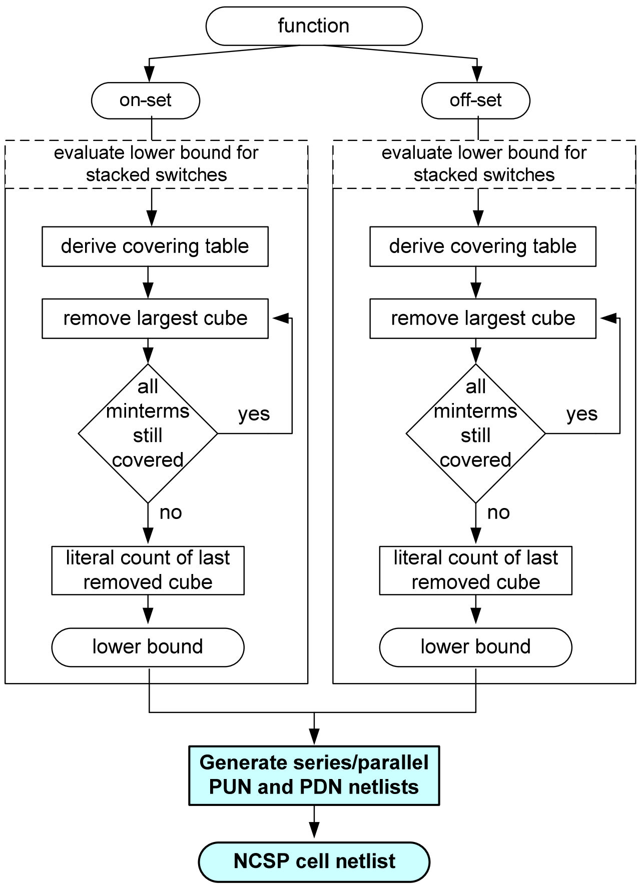

function
on-set
off-set
evaluate lower bound for stacked switches
derive covering table
remove largest cube
all minterms still covered
yes
no
literal count of last removed cube
lower bound
evaluate lower bound for stacked switches
derive covering table
remove largest cube
all minterms still covered
yes
no
literal count of last removed cube
lower bound
Generate series/parallel PUN and PDN netlists
NCSP cell netlist

Figure 4.4: NCSP logic style topology generation algorithm.

Example 7: Consider the example function shown in Fig. 4.5. An integer number will be used to represent the function, most of the time in hexadecimal radix. This number is built grouping in 4-bit sets from the output values of the truth-table. In the example presented in Fig. 4.5, the output compounds the integer $(0000.0001.1001.0111)_2 = (0197)_{16}$.

A	B	C	D	OUT	
0	0	0	0	**1**	
0	0	0	1	**1**	**7**
0	0	1	0	**1**	
0	0	1	1	**0**	
0	1	0	0	**1**	
0	1	0	1	**0**	**9**
0	1	1	0	**0**	
0	1	1	1	**1**	
1	0	0	0	**1**	
1	0	0	1	**0**	**1**
1	0	1	0	**0**	
1	0	1	1	**0**	
1	1	0	0	**0**	
1	1	0	1	**0**	**0**
1	1	1	0	**0**	
1	1	1	1	**0**	

Figure 4.5: Truth-table for the function $(0197)_{16}$.

In Fig. 4.6 it is shown a 5-4 bridge-based circuit (GREA, 1958; NINOMIYA, 1965) for the function in Fig. 4.5. Fig. 4.7 shows a 4-4 BDD-based PTL network corresponding to the logic function $(0197)_{16}$.

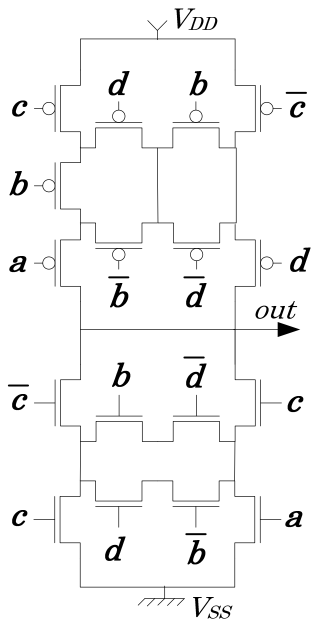


Figure 4.6: Bridge-based 5-4 cell for function $(0197)_{16}$.

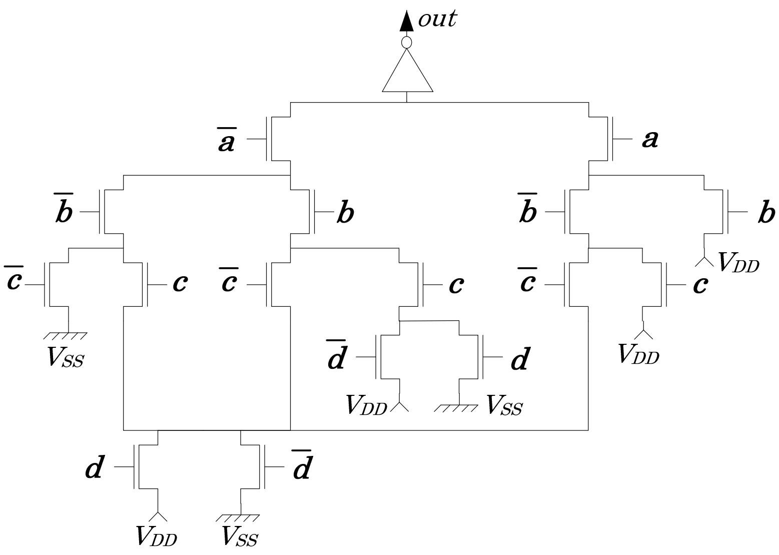


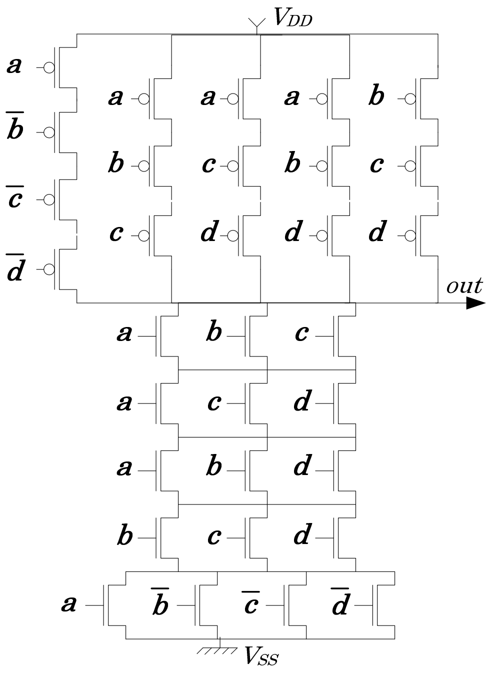
Figure 4.7: BDD-based 4-4 PTL for function $(0197)_{16}$.

Another way to implement a logic function is by extracting a representative logic equation and mapping it into a series/parallel switch network. The prime irredundant equations for the function $(0197)_{16}$ can be obtained by two-level minimization (BRAYTON, 1984). The logic equation (4.1) represents the on-set of the function $(0197)_{16}$ while the logic equation (4.2) represents the off-set for the same function.

$$f_{on-set} = \overline{a} \cdot \overline{b} \cdot \overline{c} + \overline{a} \cdot \overline{b} \cdot \overline{d} + \overline{a} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot c \cdot d + \overline{b} \cdot \overline{c} \cdot \overline{d}, \quad (4.1)$$

$$f_{off-set} = \overline{b} \cdot c \cdot d + b \cdot \overline{c} \cdot d + b \cdot c \cdot \overline{d} + a \cdot d + a \cdot c + a \cdot b. \quad (4.2)$$

Figures 4.8 and 4.9 show Complementary Series/Parallel (CSP) CMOS networks for the function $(0197)_{16}$. Fig. 4.8 presents the networks based on the on-set equation and Fig. 6 presents the networks based on the off-set one. For each CSP circuit in Figures 4.8 and 4.9 it is derived one plane from the equation and the other one as its topological complement.


(a) 5-4 cell from on-set equation (direct polarity).

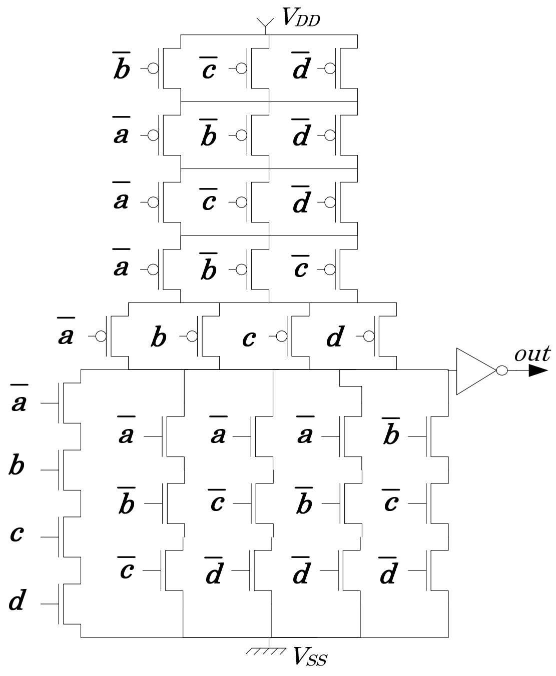
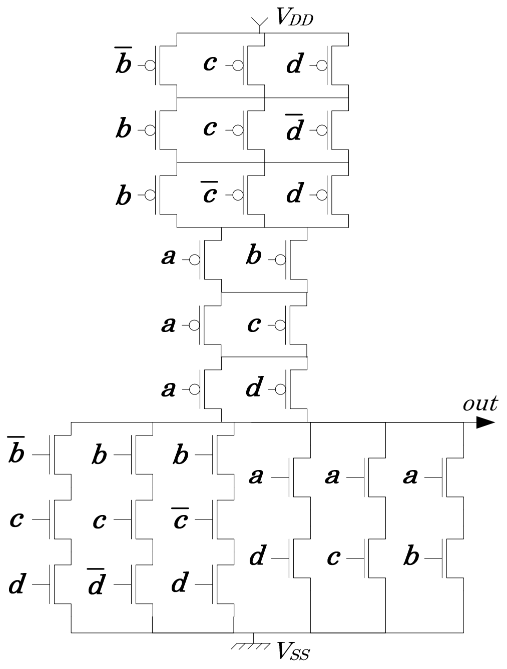

(b) 4-5 cell from on-set equation (inverted polarity).

Figure 4.8: CSP CMOS cells obtained from the on-set equations of function $(0197)_{16}$.


(a) 6-3 cell from off-set equation (direct polarity).

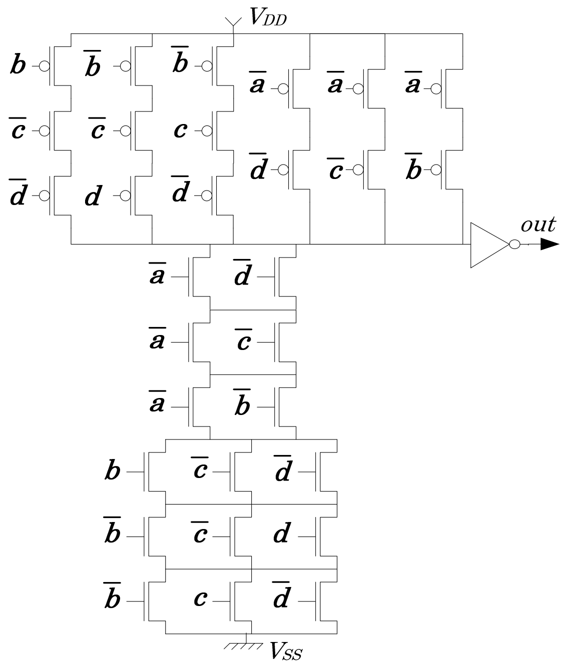

(b) 3-6 cell from off-set equation (inverted polarity).

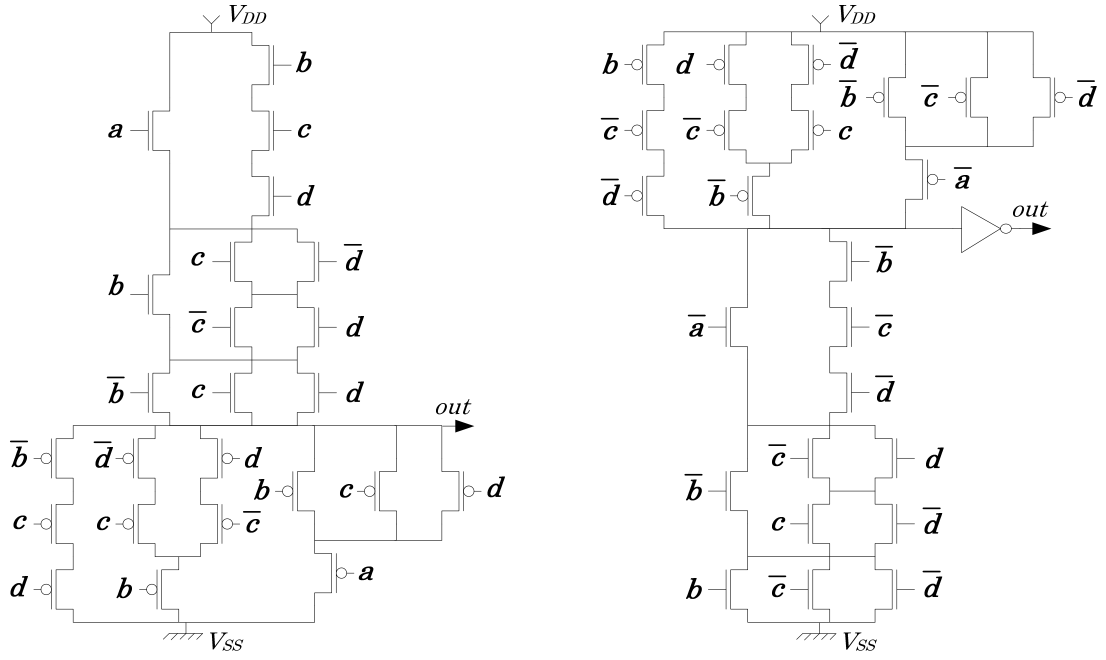
Figure 4.9: CSP CMOS cells obtained from the off-set equations of function $(0197)_{16}$.

In order to reduce the number of switches in the networks both on-set and off-set equations ((4.3) and (4.4) respectively) can be factorized (BRAYTON, 1982; BRAYTON, 1984; HACHTEL, 1996). Possible factorizations for the equations (4.1) and (4.2) are demonstrated as follows by (4.3) and (4.4) respectively.

$$f_{on-set} = \left(\bar{a} + \bar{b}\cdot\bar{c}\cdot\bar{d}\right)\cdot\left(\bar{b} + \left(\bar{c} + d\right)\cdot\left(c + \bar{d}\right)\right)\cdot\left(b + \bar{c} + \bar{d}\right), \tag{4.3}$$

$$f_{off-set} = \left(\bar{a} + \left(\bar{b} + \bar{c} + \bar{d}\right)\cdot\left(b + c\cdot d\right)\cdot\left(c + d\right)\right)\cdot\left(b + c + d\right). \tag{4.4}$$

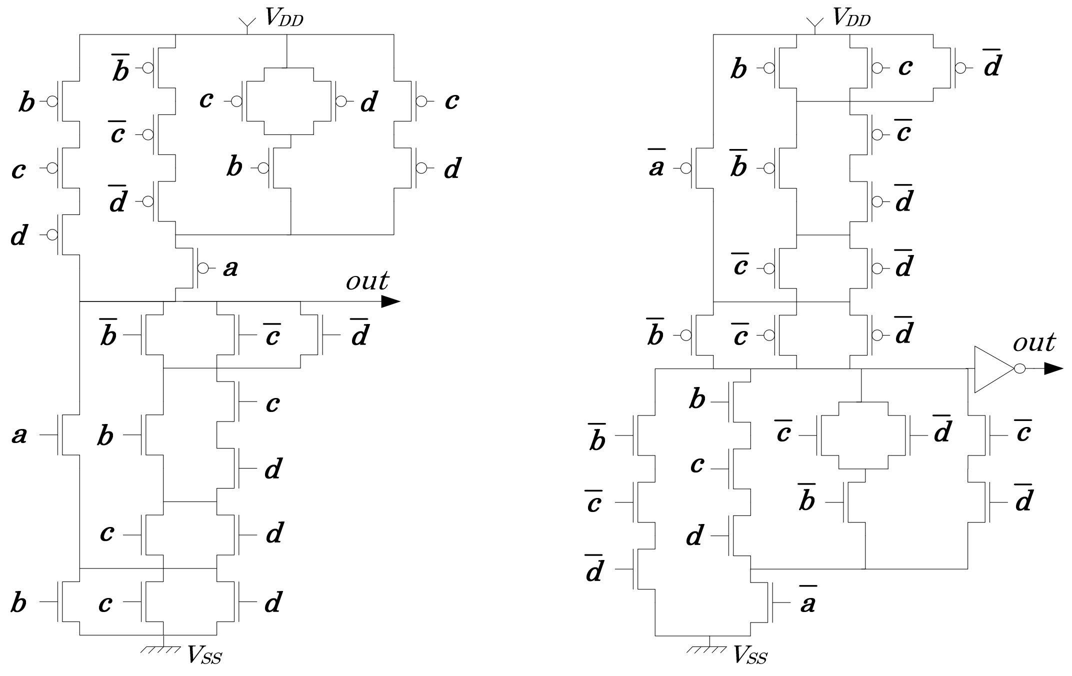
This factorization leads to the optimized networks presented in Figures 4.10 and 4.11.



(a) 3-6 cell from on-set equation (direct polarity).

(b) 6-3 cell from on-set equation (inverted polarity).

Figure 4.10: CSP CMOS cells (factorized) for function $(0197)_{16}$ – on-set equation.



(a) 5-4 cell from off-set equation (direct polarity).

(b) 4-5 cell from off-set equation (inverted polarity).

Figure 4.11: CSP CMOS cells (factorized) for function $(0197)_{16}$ – off-set equation.

In the methodology proposed hereby and illustrated in Fig. 4.1, one needs to generate PUN and PDN using a pair of on- and off-set equations which respect the lower bound for the number of transistors in series. The lower bound must be evaluated for both on- and off-set extracted from the function under evaluation. Hence, the evaluation of lower bounds requires the construction of the covering tables for both on- and off-set. Fig. 4.12 shows the covering table for the on-set of the function presented $(0197)_{16}$ while Fig. 4.13 shows the covering table for the off-set of the same function. The process of finding the prime implicants and building the covering table is well known as Quine-McCluskey minimization, and is further described in (QUINE, 1955; MCCLUSKEY, 1956). In the covering tables, the rows represent the prime implicants for the on-set/off-set logic under evaluation, while the columns represent the minterms/maxterms in the on-set/off-set equation.

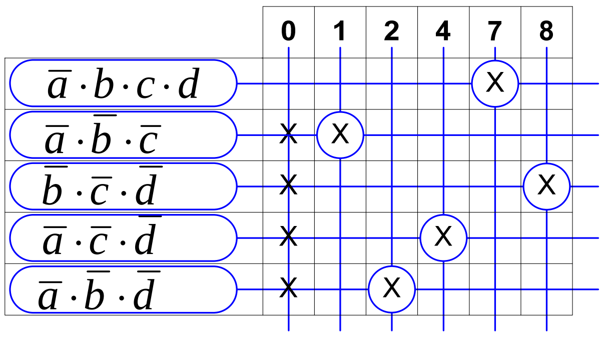
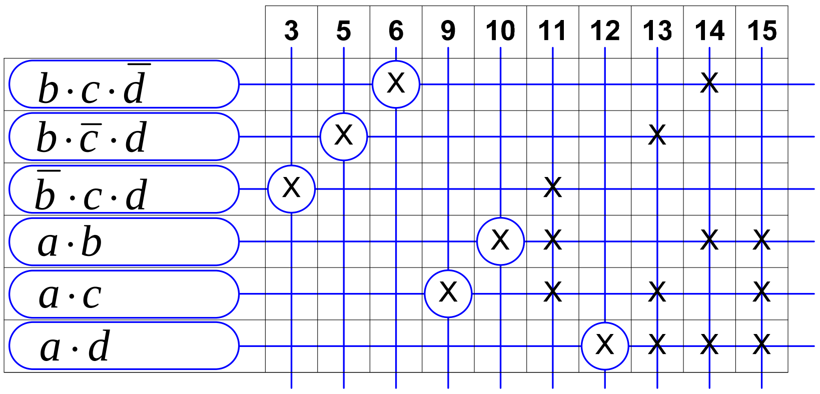

0
1
2
4
7
8
$\bar{a}\cdot b\cdot c\cdot d$
$\bar{a}\cdot\bar{b}\cdot\bar{c}$
$\bar{b}\cdot\bar{c}\cdot\bar{d}$
$\bar{a}\cdot\bar{c}\cdot\bar{d}$
$\bar{a}\cdot\bar{b}\cdot\bar{d}$
X

Figure 4.12: Covering table obtained from the on-set of function $(0197)_{16}$.



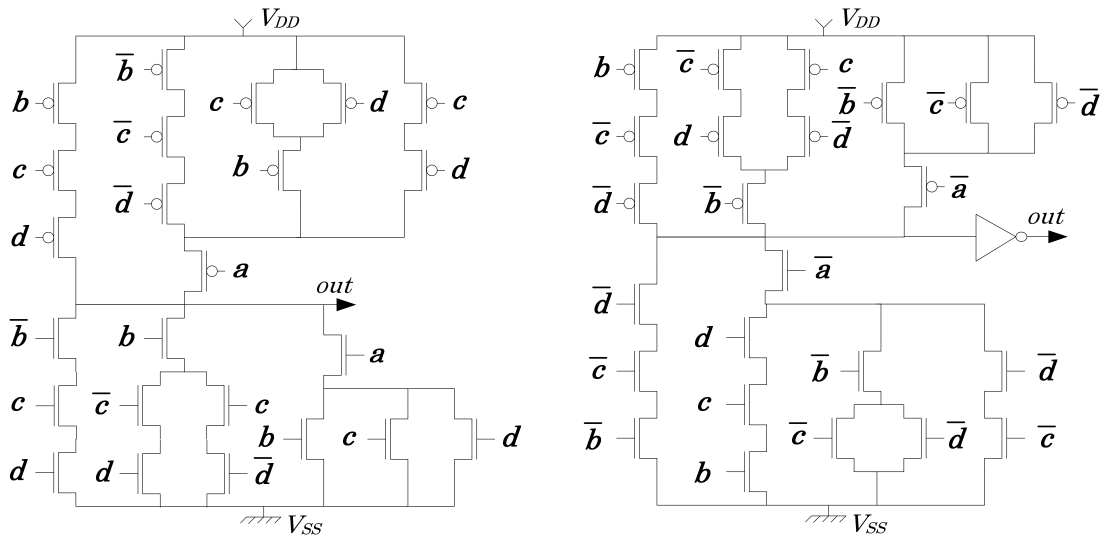
3
5
6
9
10
11
12
13
14
15
$b \cdot c \cdot \overline{d}$
$b \cdot \overline{c} \cdot d$
$\overline{b} \cdot c \cdot d$
$a \cdot b$
$a \cdot c$
$a \cdot d$

Figure 4.13: Covering table obtained from the off-set of function $(0197)_{16}$.

Looking at the covering table in Fig. 4.12 it is possible to notice that the cube with more literals ($\overline{a} \cdot b \cdot c \cdot d$) cannot be removed from the final cover as it is an essential prime implicant (i.e. it is the only prime implicant which covers the minterm 7). Hence, the lower bound for the number of stacked switches for the on-set is 4 (the number of literals in the cube). For covering table in Fig. 4.13, the lower bound evaluated is 3. Hence, the circuit with PUN and PDN respecting the lower bounds will be 3-4 or 4-3.

One must notice that none of the implementations previously presented in this example have minimum length switch stacks. Worse than that, some circuit implementations like the ones in Figures 4.8, 4.9, 4.10 and 4.11 are not even feasible due to the long transistor stacks.

Fig. 4.14 shows implementations of minimum length transistor stacks for the example function $(0197)_{16}$. The circuit on Fig. 4.14.a was generated based on the equation (4.4) for the PUN and on the equation (4.3) for the PDN. Using the same equations with inverted polarity for the inputs and output lead us to the circuit in Fig. 4.14.b. The circuit in Fig. 4.14.b might be more desirable than the one in Fig. 4.14.a, as it has a smaller path in the PUN, more resistive than the PDN for same-length paths.



(a) 4-3 cell (direct polarity). (b) 3-4 cell (inverted polarity).

Figure 4.14: NCSP CMOS cells for function $(0197)_{16}$.

The work in (PIGUET, 1984; PIGUET 1994; PIGUET 1995) also presents a technique of generating circuits using non-complementary pull-up/down networks. The circuits generated with such technique are so called *branch-based* (Section 2.4) due to the fact that the cubes of the on-set and off-set equations are directly translated into branches in the pull-up/down planes. However, this technique is not concerned with the length of transistor chains.

Example 8: Now it will be used another function to better exemplify the lower bound evaluation used to choose the correct networks for minimum transistor stacks. Fig. 4.15 shows the truth-table of a 5-input function whose representative hexadecimal integer is $(F1D12F33)_{16}$.

E = 0

A	B	C	D	OUT	
0	0	0	0	**1**	
0	0	0	1	**1**	**3**
0	0	1	0	**0**	
0	0	1	1	**0**	
0	1	0	0	**1**	
0	1	0	1	**1**	**3**
0	1	1	0	**0**	
0	1	1	1	**0**	
1	0	0	0	**1**	
1	0	0	1	**1**	**F**
1	0	1	0	**1**	
1	0	1	1	**1**	
1	1	0	0	**0**	
1	1	0	1	**1**	**2**
1	1	1	0	**0**	
1	1	1	1	**0**	

E = 1

A	B	C	D	OUT	
0	0	0	0	**1**	
0	0	0	1	**0**	**1**
0	0	1	0	**0**	
0	0	1	1	**0**	
0	1	0	0	**1**	
0	1	0	1	**0**	**D**
0	1	1	0	**1**	
0	1	1	1	**1**	
1	0	0	0	**1**	
1	0	0	1	**0**	**1**
1	0	1	0	**0**	
1	0	1	1	**0**	
1	1	0	0	**1**	
1	1	0	1	**1**	**F**
1	1	1	0	**1**	
1	1	1	1	**1**	

Figure 4.15: Truth-table for the function $(F1D12F33)_{16}$.

The prime irredundant equations for the function $(F1D12F33)_{16}$ obtained by two-level minimization are (4.5) for the on-set and (4.6) for the off-set.

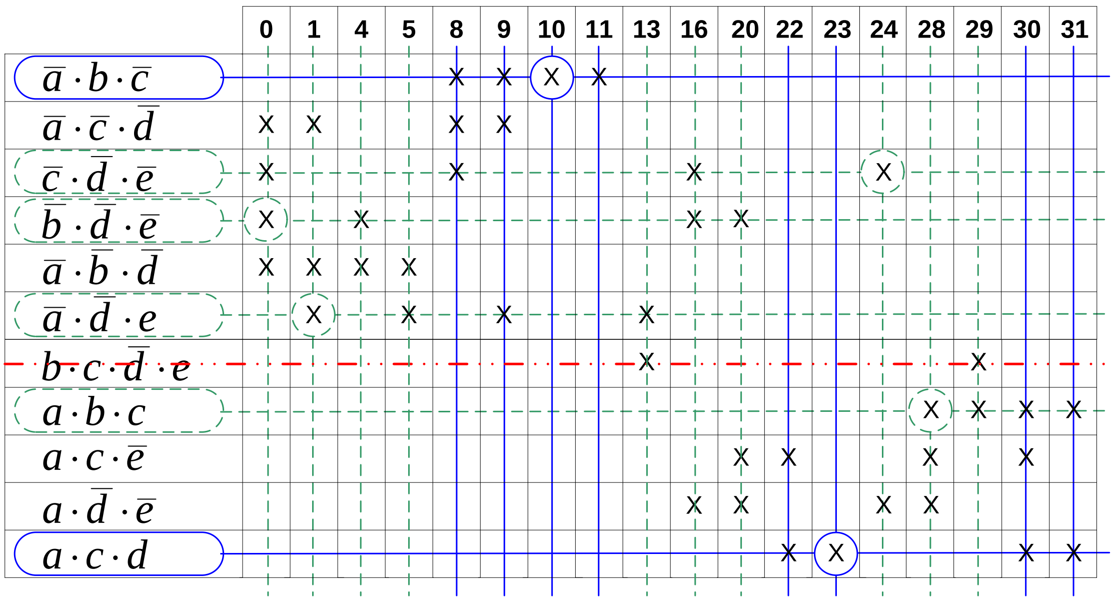
$$f_{on-set} = \overline{a} \cdot \overline{b} \cdot \overline{d} + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{d} \cdot \overline{e} + a \cdot c \cdot d + b \cdot c \cdot \overline{d} \cdot e \, , \tag{4.5}$$

$$f_{off-set} = \overline{a} \cdot b \cdot c \cdot \overline{e} + \overline{a} \cdot c \cdot d + a \cdot \overline{b} \cdot \overline{d} \cdot e + a \cdot \overline{c} \cdot e + a \cdot \overline{c} \cdot d + \overline{b} \cdot \overline{c} \cdot d \, . \tag{4.6}$$

These equations were generated aiming a smaller set of cubes. However, if one evaluates the lower bound just looking for the cube with more literals it will be wrongly assumed that the lower bounds for the function $(F1D12F33)_{16}$ should be 4-4. However, the covering table for the on-set of such function (Fig. 4.16) shows that the cube (prime implicant) with more literals on the equation (4.5) can be removed without losing the cover of all minterms. In equation (4.7) it is shown an equivalent representation for equation (4.5) where the cube with four literals can be replaced by a pair of cubes with only three literals.

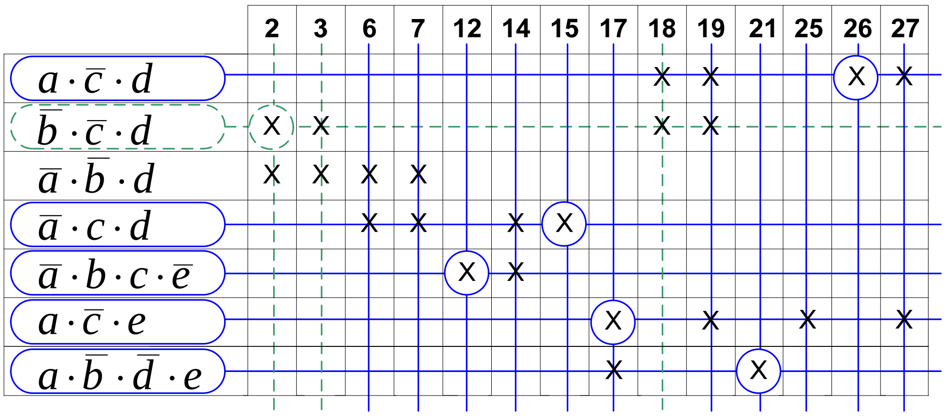
$$f_{on-set} = \overline{a} \cdot \overline{b} \cdot \overline{d} + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{d} \cdot \overline{e} + a \cdot c \cdot d + \overline{a} \cdot \overline{d} \cdot e + a \cdot b \cdot c \, . \tag{4.7}$$

The off-set covering table in Fig. 4.17 shows that the same reduction cannot be done to the off-set equation (4.6), once that the cubes with more literals are essential prime implicants. It means that they cannot be removed from the final solution and that the evaluation of lower bounds for the function $(F1D12F33)_{16}$ should deliver 3-4/4-3 circuits.



	0	1	4	5	8	9	10	11	13	16	20	22	23	24	28	29	30	31
$\overline{a}\cdot b\cdot \overline{c}$					X	X	(X)	X										
$\overline{a}\cdot \overline{c}\cdot \overline{d}$	X	X			X	X												
$\overline{c}\cdot \overline{d}\cdot \overline{e}$	X				X					X				(X)				
$\overline{b}\cdot \overline{d}\cdot \overline{e}$	(X)		X							X	X							
$\overline{a}\cdot \overline{b}\cdot \overline{d}$	X	X	X	X														
$\overline{a}\cdot \overline{d}\cdot e$		(X)		X		X			X									
$b\cdot c\cdot \overline{d}\cdot e$									X							X		
$a\cdot b\cdot c$															(X)	X	X	X
$a\cdot c\cdot \overline{e}$											X	X			X		X	
$a\cdot \overline{d}\cdot \overline{e}$										X	X			X	X			
$a\cdot c\cdot d$												X	(X)				X	X

Figure 4.16: Covering table obtained from the on-set of function $(F1D12F33)_{16}$.



	2	3	6	7	12	14	15	17	18	19	21	25	26	27
$a\cdot \overline{c}\cdot d$									X	X			(X)	X
$\overline{b}\cdot \overline{c}\cdot d$	(X)	X							X	X				
$\overline{a}\cdot \overline{b}\cdot d$	X	X	X	X										
$\overline{a}\cdot c\cdot d$			X	X		X	(X)							
$\overline{a}\cdot b\cdot c\cdot \overline{e}$					(X)	X								
$a\cdot \overline{c}\cdot e$								(X)		X		X		X
$a\cdot \overline{b}\cdot \overline{d}\cdot e$								X			(X)			

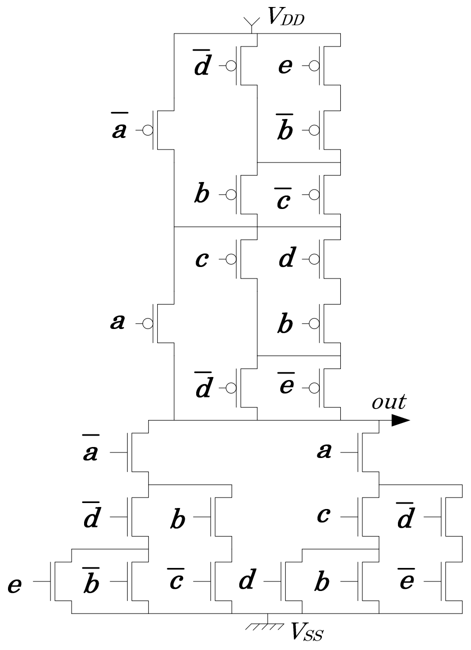
Figure 4.17: Covering table obtained from the off-set of function $(F1D12F33)_{16}$.

The equations (4.7) (on-set) and (4.6) (off-set) can be further factorized and possible results are provided by equations (4.8) and (4.9), respectively.

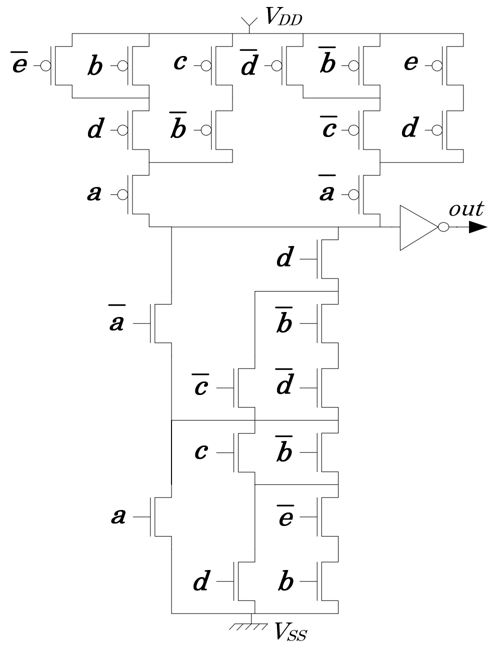
$$f_{on-set} = \overline{a}\cdot\left(\overline{d}\cdot\left(e+\overline{b}\right)+b\cdot\overline{c}\right)+a\cdot\left(c\cdot\left(d+b\right)+\overline{d}\cdot\overline{e}\right), \tag{4.8}$$

$$f_{off-set} = \overline{a}\cdot c\cdot\left(b\cdot\overline{e}+d\right)+\overline{b}\cdot\overline{c}\cdot d+a\cdot\left(\overline{b}\cdot\overline{d}\cdot e+\overline{c}\cdot\left(e+d\right)\right). \tag{4.9}$$

In Figures 4.18 and 4.19, circuits using the regular CSP CMOS approach are shown using the factorized equations in (4.8) and (4.9), respectively. The pull-up/down networks of these circuits were used in Fig. 4.20 to generate NCSP circuits with minimum length transistor chains.

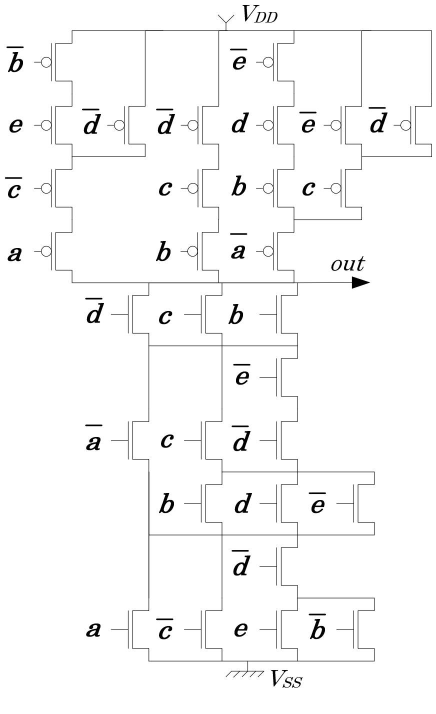


(a) 6-3 cell from on-set equation (direct polarity).

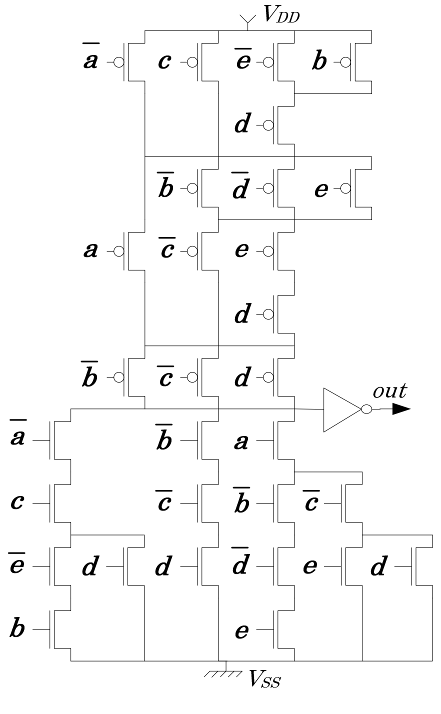


(b) 3-6 cell from on-set equation (inverted polarity).

Figure 4.18: CSP cells obtained from the on-set equation of function $(F1D12F33)_{16}$.

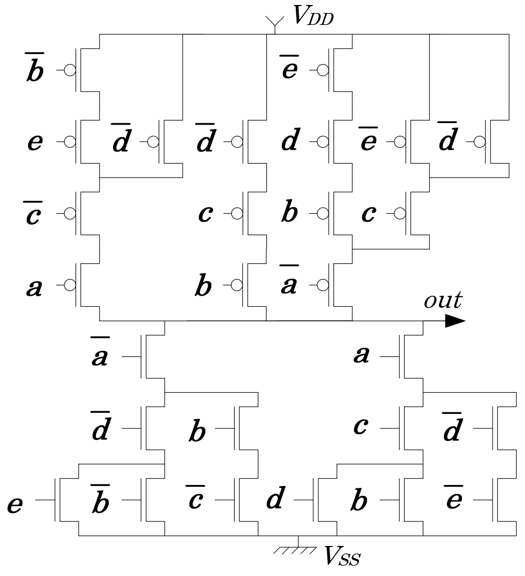


(a) 4-6 cell from off-set equation (direct polarity).



(b) 6-4 cell from off-set equation (inverted polarity).

Figure 4.19: CSP cells obtained from the off-set equation of function $(F1D12F33)_{16}$.


(a) 4-3 NCSP cell from equation (direct polarity).

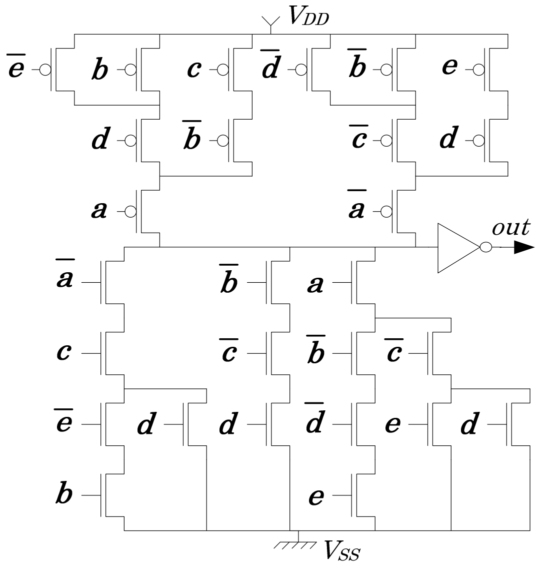

(b) 3-4 NCSP cell from equation (inverted polarity).

Figure 4.20: NCSP cells for function $(F1D12F33)_{16}$.

4.6 Other Lower-Bound-Based Style

The constructive method for generating transistor-level networks presented hereby is the first one which follows the lower bound for the number of stacked transistors for every desired logic gate. Another constructive method which also respects the mentioned lower bound is presented in (ROSA, 2007), where instead of translating factorized SOPs into series/parallel transistor configurations, it constructs disjoint transistor networks based on BDDs (POLI, 2003; ROSA, 2006). It has been reported for this method smaller transistor counts and smaller logical effort (see Table 5.5).

5 RESULTS

5.1 Introduction

The electrical evaluation of NSCP gates has been carried out using the commercial cell characterization tool Nangate Cell Characterizer™ (NANGATE, 2007), which provides a full characterization of a cell library described in Liberty format, with associated SPICE or GDS-II files describing each cell. The logic functions were described as SPICE netlists, and the transistors were straightforwardly sized for NCSP, CSP and PTL cells. The sizing strategy used for every cell in this Chapter is the same. In such strategy, transistors were equally sized and a PMOS/NMOS-ratio was also considered when sizing the PUN transistors. The typical process parameters from TSMC 0.13µm CMOS (TSMC, 2007) has been taken into account in the characterization, realized for 16 different conditions: four input slopes versus four output loads.

5.2 Electrical Characterization

5.2.1 'Black-Box' cell comparison

In the first analysis, the logic cells were considered as 'black-box' circuits, i.e. the logic functions were implemented with different topologies, implying in some cases that the addition or removal of input/output inverters were done. The set of cells used in this first comparison is the set of all 4-input cells, represented through the set of 3984 P-classes of 4-input functions (CORREIA, 2001), as depicted on Table 3.6.

5.2.1.1 CSP Generation

To generate a CSP implementation for a given function it is necessary to choose between the equations derived from the on-set and from the off-set of the function. The choice of the Boolean expression that closely respects the lower bound is slightly more complicated in some cases. For instance, being *PU_length* and *PD_length* the longest transistor stacks in the pull-up and pull-down networks respectively, consider the following two examples.

Example A: Lower Bound: PU_length=2, PD_length=3

off-set: PU_length=2, PD_length=6 [$out = a \cdot (c + b + d) + d \cdot (c + b) + b \cdot c$]

on-set: PU_length=3, PD_length=4 [$out = \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot (\overline{b} \cdot (\overline{d} + \overline{c}) + \overline{c} \cdot \overline{d})$]

Example B: Lower Bound: PU_length=3, PD_length=4

off-set: PU_length=3, PD_length=5 [$out = \overline{b} \cdot (c + d) \cdot (\overline{c} + \overline{d}) + b \cdot (\overline{c} \cdot \overline{d} + c \cdot d) + a$]

on-set: PU_length=4, PD_length=4 [$out = \overline{a} \cdot (\overline{b} \cdot (\overline{c} + d) \cdot (c + \overline{d}) + b \cdot (c \cdot \overline{d} + \overline{c} \cdot d))$]

To choose between on-set and off-set equations a criterion is necessary. Two different criteria used hereby are described below.

CSP_1 Criterion: The equation whose pull-up respects the lower bound is chosen. This is always possible as the polarity is changed to use the smaller stack as the pull-up. This choice corresponds to the off-set equation for both examples above.

CSP_2 Criterion: Choose the equation with smaller difference between both planes: *min{PD_length - PU_length}*. This choice corresponds to the on-set equation for both examples above.

For both examples above, the solutions chosen by CSP_1 criterion are not feasible, as they have more than four transistors in series. Out of the 3984 4-input P-classes, a set of 118 cases present different but feasible topologies when using either CSP_1 or CSP_2 criteria. The CSP_1 and CSP_2 versions for these cells have been characterized and compared with each other. Fig. 5.1 presents the result of this comparison. The X-axis shows the percentage differences in propagation delay, considering the CSP_1 criterion as the reference. The Y-axis presents the number of cases. It is possible to observe that there is not a general winner between CSP_1 and CSP_2 criteria. Similar behavior was also observed in terms of power dissipation and power-delay product. The characterization of each gate was done using a 2fF output load (reference inverter equivalent capacitance) and an input slope of 0.02ns.

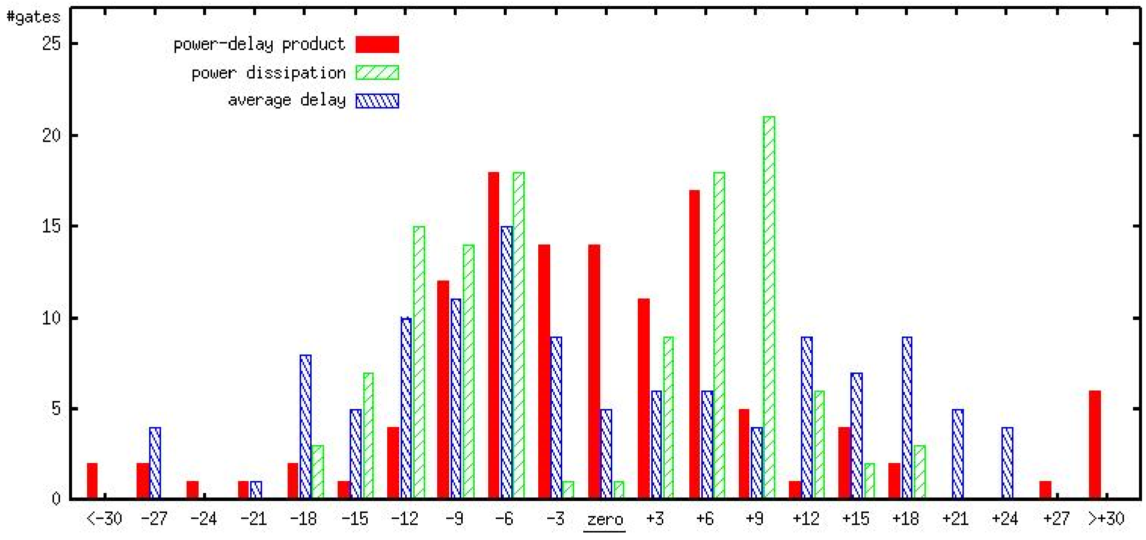
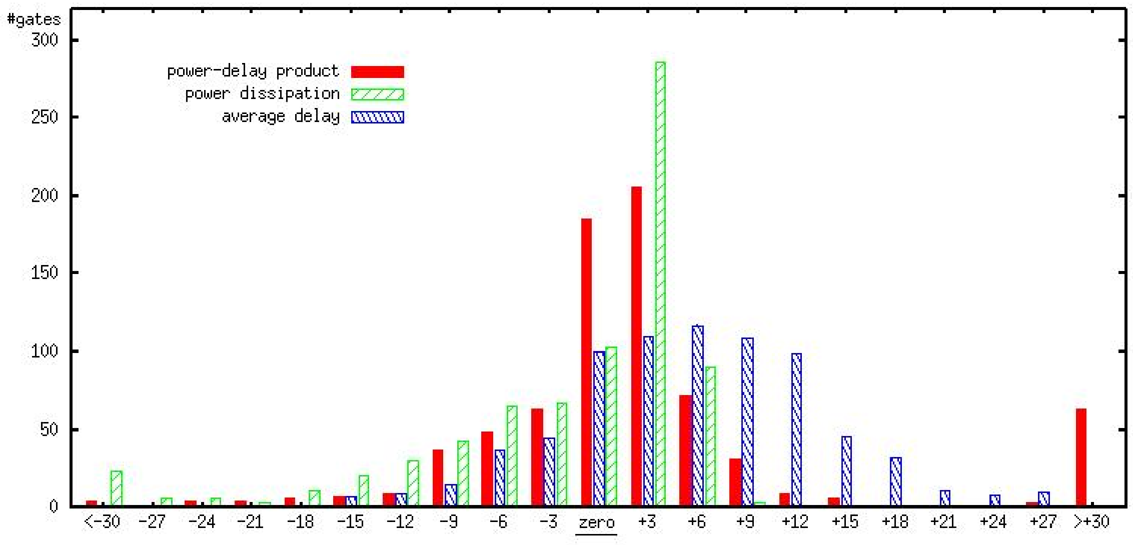


Figure 5.1: CSP_1 (reference) and CSP_2 propagation delay comparison, number of cells according to delay, power and PDP improvement (%).

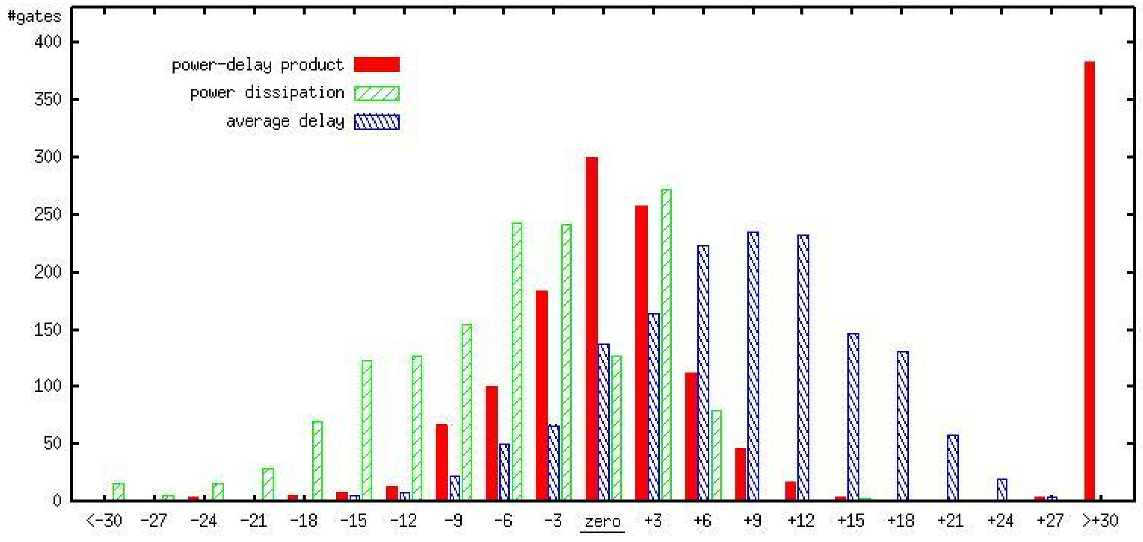
5.2.1.2 NCSP Evaluation

Gates using the NCSP style presented herein were then compared to both CSP_1 and CSP_2 approaches. Only feasible CSP cells (i.e. with four or less serial transistors) where the lower bound from Theorems 1 and 2 was not achieved were compared to NCSP. This corresponds to a set of 758 different topologies for CSP_1 and to a set of 1513 different cells for CSP_2. This difference happens because CSP_2 tends to return more feasible cells, as the PUP and PDN chain lengths difference is smaller.

In Fig. 5.2.a the sum of rise and fall propagation delay differences between NCSP and CSP_1 is observed, for different input slopes and output loads. It also shows the expected improvements for power dissipation and power-delay product. The X-axis is given in percentage, being the positive values related to the NCSP gains, i.e. the reduction in delay obtained using such new logic style. The comparison to CSP_2 is shown in Fig. 5.2.b.



(a)



(b)

Figure 5.2: NCSP compared to (a) CSP_1 and (b) CSP_2: number of cells according to delay, power and PDP improvement (%).

5.2.2 Unate Cell Comparison

The set of cells considered in such analysis includes only unate functions – each input has a single polarity. Moreover, in order to focus in the main pull-up and pull-down logic network building, this group of cells is only composed by gates without any inversion of the input and output signals. The number of negative unate functions initially identified is presented in Table 5.1 (*#NUF*). Another restriction to define the set of cells for NCSP characterization and comparison to CSP was again the number of stacked transistors in the PUN and PDN, limited to four transistors. Furthermore, the functions whose CSP gate respects the 'lower bound' metric were also subtracted from this group. As a result, the number of possible characterized and compared NCSP and CSP implementations was reduced, as shown in Table 5.1 (*#COMP*). In this table is also presented the average number of transistors per gate for both CSP (*#TR_CSP*) and NCSP (*#TR_NCSP*) styles.

Table 5.1: Number of negative unate functions**.**

INPUTS	#NUF	#COMP	#TR_CSP	#TR_NCSP
1	1	–	–	–
2	2	–	–	–
3	5	1	10	10
4	20	8	12.8	12.8
5	171	53	15.4	15.7
6	9007	280	18.2	19.1

The improvements in propagation delay of NSCP compared to CSP are observed in the graphs of Figs. 5.3 and 5.4. In Fig. 5.3 is shown the average delay gains (in percentage, X-axis) obtained with 6-input NCSP cell implementation compared to CSP, for four output loads (2fF, 5fF, 10fF and 20fF) and the input slope equal to 0.5ns. The right extremity of X-axis means gains equal to or greater than 30%. The best case attained around 45% of improvement. The Y-axis corresponds to the number of cells with respective savings. As expected, reducing the number of stacked transistors, the cell timing is also reduced. However, this assumption has not been verified for all cases. The performance degeneration in some few exceptions can be explained by the particular electrical arrangement, i.e. the transistor ordering and consequently the input position. The simplified sizing strategy used for these simulations is another issue that slightly reduces the accuracy of the results. However, it is expected for both CSP and NSCP cells used on this comparison an equal benefit from an improved sizing strategy as both are series/parallel circuits.

In Fig. 5.4 is presented the comparison of power dissipation, average delay, power-delay product and average input load for 6-input cells. It corresponds to the characterization with 0.02ns of input slope and 2fF of output load. It can be observed the increasing in input load for some cases probably due to the factorization method applied.

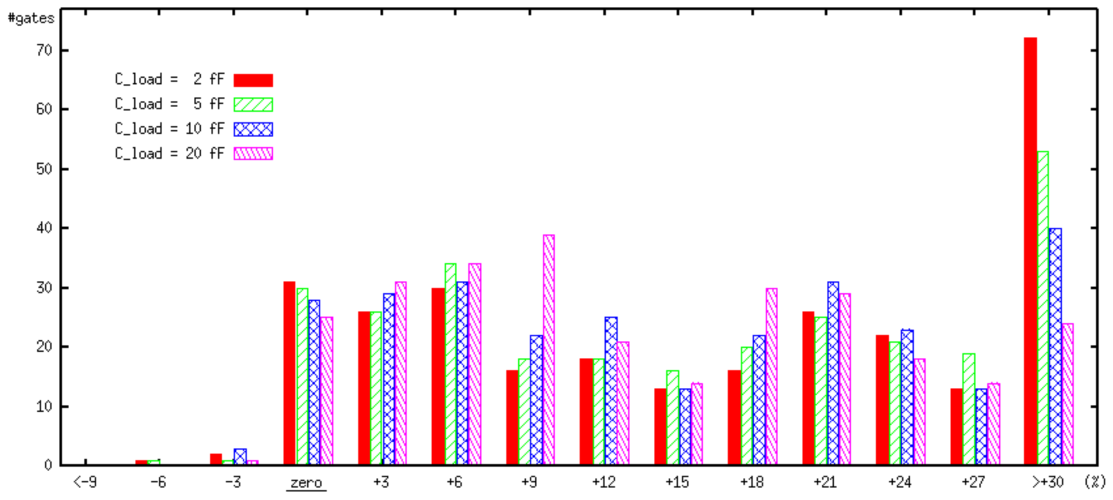


Figure 5.3: NCSP average delay improvement (%), X-axis, versus number of 6-input cells, Y-axis, for different output loads and input slope of 0.5ns. X-axis right extremity means ≥ +30%.

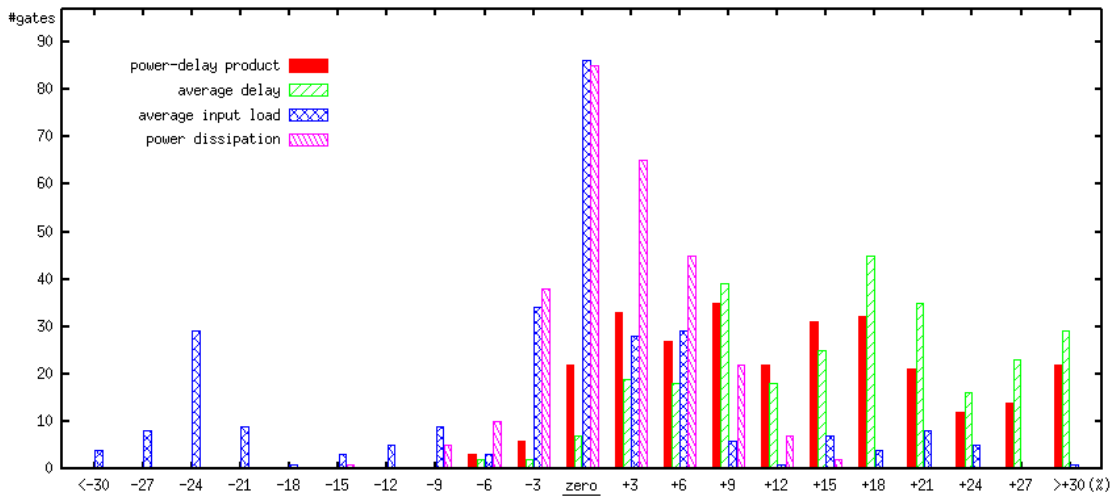


Figure 5.4: NCSP improvement (%), X-axis, versus number of 6-inputs cells, Y-axis: power-delay product, average delay, power dissipation and average input load. X-axis extremities mean ≥ ±30%.

5.2.3 Mapped Circuits Comparison

As it was presented in Table 3.7, when creating circuits which respect the lower bounds of stacked transistors – like the NCSP ones – a much broader range of functions can be implemented in a single gate. For any circuit, when it is not practical to implement it due to too many transistors in series, its logic is split in order to achieve multi-gate (multi-stage) implementations.

In this third step of the analysis, since this new CMOS logic style is able to build more complex functions in a single gate, seven critical functions were identified and implemented in NSCP and also mapped to multi CSP gate circuit using SIS Logic Synthesis tool (SENTOVICH, 1992), in order to exemplify this issue. The Boolean equations of such functions are the following:

$$\begin{aligned}F1=\;&\overline{x0}\cdot(\overline{x5}\cdot(\overline{x1}\cdot(\overline{x4}+\overline{x3}+\overline{x2})+(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4}))+\\&(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+(\overline{x2}\cdot\overline{x3}\cdot\overline{x4})))+\overline{x1}\cdot\overline{x2}\cdot\overline{x3}\cdot\overline{x4}+\\&\overline{x5}\cdot(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+(\overline{x2}\cdot\overline{x3}\cdot\overline{x4}))\end{aligned}\tag{5.1}$$

$$\begin{aligned}F2=\;&\overline{x0}\cdot(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+(\overline{x2}\cdot\overline{x3}\cdot\overline{x4}+\overline{x5}\cdot(\overline{x4}+\overline{x3}+\overline{x2})))+\\&\overline{x5}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+\overline{x1}\cdot(\overline{x2}\cdot\overline{x3}\cdot\overline{x4}+\overline{x5}\cdot(\overline{x4}+\overline{x3}+\overline{x2}))\end{aligned}\tag{5.2}$$

$$\begin{aligned}F3=\;&\overline{x0}\cdot(\overline{x3}\cdot(\overline{x1}\cdot(\overline{x5}+\overline{x4})+(\overline{x2}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5}))+(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x5}+\overline{x4})+\\&\overline{x4}\cdot\overline{x5})+(\overline{x2}\cdot\overline{x4}\cdot\overline{x5})))+\overline{x1}\cdot(\overline{x2}\cdot\overline{x4}\cdot\overline{x5})+\overline{x3}\cdot(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5})+\\&\overline{x2}\cdot\overline{x4}\cdot\overline{x5}))\end{aligned}\tag{5.3}$$

$$\begin{aligned}F4=\;&\overline{x0}\cdot(\overline{x4}\cdot((\overline{x1}\cdot(\overline{x3}+\overline{x2})+(\overline{x2}\cdot\overline{x3})+\overline{x5})+(\overline{x1}\cdot\overline{x2}\cdot\overline{x3}+\overline{x5}\cdot(\overline{x3}+\overline{x2}+\overline{x1})))+\\&\overline{x5}\cdot(\overline{x1}\cdot(\overline{x3}+\overline{x2})+\overline{x2}\cdot\overline{x3})+\overline{x4}\cdot(\overline{x1}\cdot\overline{x2}\cdot\overline{x3}+\overline{x5}\cdot(\overline{x3}+\overline{x2}+\overline{x1}))\end{aligned}\tag{5.4}$$

$$\begin{aligned}F5=\;&\overline{x3}\cdot\overline{x4}\cdot\overline{x5}+\overline{x0}\cdot(\overline{x2}\cdot((\overline{x3}\cdot\overline{x4}+\overline{x5})+\overline{x1})+(\overline{x1}\cdot(\overline{x3}\cdot\overline{x4}+\overline{x5})+\\&(\overline{x5}\cdot(\overline{x4}+\overline{x3}))))+\overline{x1}\cdot(\overline{x5}\cdot(\overline{x4}+\overline{x3}))+\overline{x2}\cdot(\overline{x1}\cdot(\overline{x3}\cdot\overline{x4}+\overline{x5})+(\overline{x5}\cdot(\overline{x4}+\overline{x3})))\end{aligned}\tag{5.5}$$

$$\begin{aligned}F6=\;&\overline{x1}\cdot\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4}\cdot(\overline{x2}+\overline{x1})+\overline{x0}\cdot(\overline{x5}\cdot(\overline{x4}+\overline{x3}+\overline{x2}+\overline{x1})+\\&((\overline{x2}+\overline{x1})\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4}+\overline{x1}\cdot\overline{x2}))+\overline{x5}\cdot((\overline{x2}+\overline{x1})\cdot(\overline{x4}+\overline{x3})+\\&\overline{x3}\cdot\overline{x4}+\overline{x1}\cdot\overline{x2})\end{aligned}\tag{5.6}$$

$$\begin{aligned}F7=\;&\overline{x3}\cdot\overline{x4}\cdot\overline{x5}+\overline{x0}\cdot((\overline{x2}\cdot(\overline{x5}+\overline{x4}+\overline{x3})+(\overline{x3}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5}))+\overline{x1})+\\&\overline{x2}\cdot(\overline{x3}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5})+\overline{x1}\cdot(\overline{x2}\cdot(\overline{x5}+\overline{x4}+\overline{x3})+(\overline{x3}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5}))\end{aligned}\tag{5.7}$$

For equations (5.1) to (5.7), three SIS libraries have been targeted: 33-4.genlib, 44-3.genlib and 44-6.genlib. It resulted in the circuit sizes presented in Table 5.2. The total number of transistors required in the function implementations was always less in the NSCP approach. The NCSP gate schematic of function *F1* (equation (5.1)) is depicted in Fig. 5.4. The multi-stage equivalent circuit using CSP gates is presented in Fig. 5.5. Table 5.3 shows the number of transistors in series when trying to implement CSP circuits for functions *F1* to *F7* in a single gate. The same table also presents the number of stacked transistors for the NCSP implementations – all respecting lower bound – used in the electrical evaluation.

The electrical performance comparison between NCSP and these three CSP circuit versions is shown in Table 5.4. The delay and power consumption improvements in these cases are even more significant, as it is expected when the number of gates is reduced in the circuit implementation and no important degradation is caused by a reasonable number of stacked transistors in the gates. Power-delay product attained up to 75% of gain for the function *F3* (equation (5.3)). Again, the accuracy of the results may be reduced due to the lack of a more careful transistor sizing in the NCSP gates.

Table 5.2: Size of CSP multi-gate mapped circuits:
#TR – number of transistors; *#CS* – number of cells**.**

	NCSP	**33-4.genlib**		**44-3.genlib**		**44-6.genlib**	
	#TR	**#CS**	**#TR**	**#CS**	**#TR**	**#CS**	**#TR**
F1	63	11	74	11	82	10	74
F2	65	13	82	9	78	7	70
F3	56	13	70	7	64	11	76
F4	58	13	80	9	74	9	74
F5	56	11	72	9	72	9	68
F6	65	15	92	6	74	6	74
F7	60	9	70	7	68	7	68

Table 5.3: Number of transistors in series for single/gate implementation of CSP and NCSP approaches.

	CSP_on[a]	CSP_off[b]	NCSP
	p-n (or n-p)	p-n (or n-p)	p-n (or n-p)
F1	14-4 (or 4-14)	17-3 (or 3-17)	3-4 or (4-3)
F2	15-4 (or 4-15)	16-3 (or 3-16)	3-4 (or 4-3)
F3	14-4 (or 4-14)	14-4 (or 4-14)	4-4
F4	15-4 (or 4-15)	15-4 (or 4-15)	4-4
F5	14-4 (or 4-14)	14-4 (or 4-14)	4-4
F6	16-3 (or 3-16)	15-4 (or 4-15)	3-4 (or 4-3)
F7	17-3 (or 3-17)	14-4 (or 4-14)	3-4 (or 4-3)

(a) CSP circuit generated from the factorized **on-set** equation.
(b) CSP circuit generated from the factorized **off-set** equation.

Table 5.4: NCSP single-gate improvement (%) compared to CSP multi-gate mapped circuits. [a]

	33-4.genlib		44-3.genlib		44-6.genlib	
	$\boldsymbol{T_d}$	***PDP***	$\boldsymbol{T_d}$	***PDP***	$\boldsymbol{T_d}$	***PDP***
F1	4	11	17	21	18	25
F2	-5	7	-16	-6	5	-8
F3	9	35	50	25	66	75
F4	42	38	29	29	29	26
F5	17	13	11	09	25	6
F6	27	50	25	11	25	11
F7	47	21	49	20	49	20

(a) Worst cases: $\boldsymbol{T_d}$ – delay propagation, ***PDP*** –power-delay product.

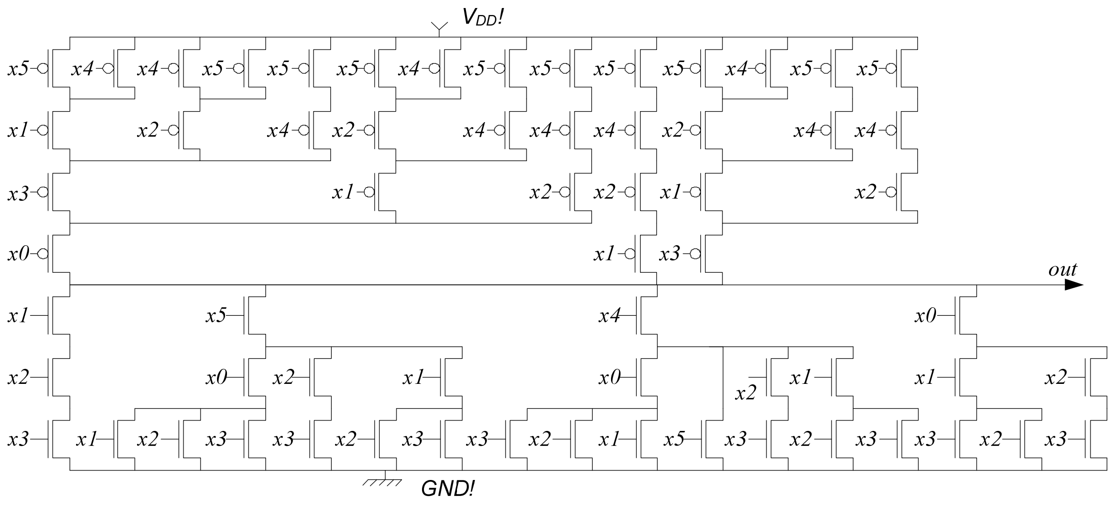
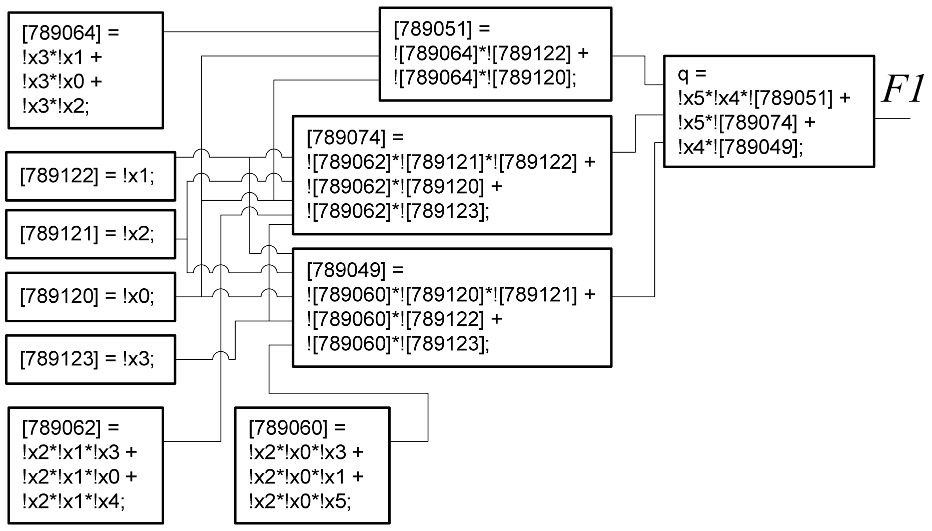
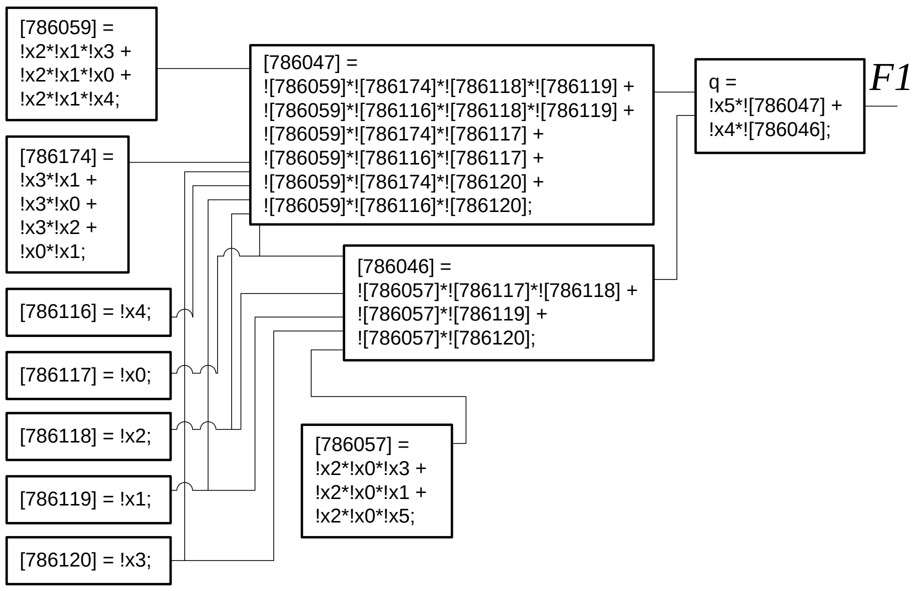


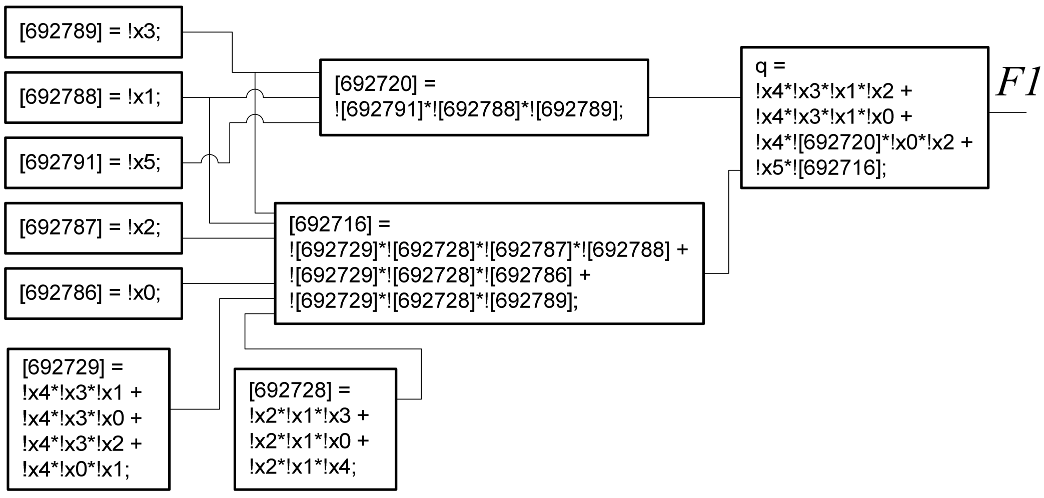
Figure 5.4: NCSP gate implementation of function ‘F1’.



(a)



(b)



(c)

Figure 5.5: CSP multi-stage implementation of function ‘F1’ targeting libraries (a) 33-4.genlib, (b) 44-3.genlib and (c) 44-6.genlib.

5.3 Logical Effort

The method of logical effort (SUTHERLAND, 1999) is founded on a simple model of the delay through a single logic gate. The model describes delays caused by the capacitive load that the logic gate drives and by the topology of the logic gate. Clearly, as the load increases, the delay increases, but delay also depends on the logic function of the gate. Inverters, the simplest logic gates, drive loads best and are often used as amplifiers to drive large capacitances. Logic gates that compute other functions require more transistors, some of which are connected in series, making them poorer than inverters at driving current. Thus a NAND gate has more delay than an inverter with similar transistor sizes that drives the same load. The method of logical effort quantifies these effects to simplify delay analysis for individual logic gates and multistage logic networks.

The logical effort of a gate is defined itself as the ratio of the input capacitance of the gate to the input capacitance of an inverter that can deliver the same output current. Equivalently, logical effort indicates how much worse a gate is at producing output current as compared to an inverter, given that each input of the gate may only present as much input capacitance as the inverter (WESTE, 2006).

Estimating the delay of a single stage in a circuit can be done calculating the so called *delay effort* of the gate. Additionally to the logical effort, the delay effort also depends on the output capacitance by input capacitance ratio, the so called *electrical effort*, and on a *parasitic delay*. The delay effort can be expressed by the following equation:

$$d = g.h + p. \tag{5.8}$$

In equation (5.8), g is the logical effort, h is the electrical effort, and p is the parasitic delay of the gate when it drives a zero load.

For multi-stage circuits, the delay is obtained through the evaluation of the so called *path effort delay*, which is expressed by the following equations:

$$D = N.F^{1/N} + P\ , \tag{5.8}$$

$$F = G.B.H\ , \tag{5.9}$$

$$G = \Pi g_i\ , \tag{5.10}$$

$$B = \Pi b_i\ , \tag{5.11}$$

$$b = (C_{on\text{-}path}+C_{off\text{-}path})/C_{on\text{-}path}\ , \tag{5.12}$$

$$H=C_{out}/C_{in}\ , \tag{5.13}$$

$$P = \Sigma p_i\ . \tag{5.14}$$

The variables in the equations (5.8) to (5.14), are as follows:

- D – path delay effort: the minimum possible delay for a N-stage path.
- N – number of stages in a path;
- P – path parasitic delay: the sum of the parasitic delay of each stage in a path;
- G – path logical effort: the products of the logical efforts of each stage in a path;
- b – branching effort: is the ratio of the total capacitance seen by a stage to the capacitance on a path;
- B – path branching effort: the product of the branching efforts along a path.
- H – path electrical effort: ratio of the output capacitance the path must drive by the input capacitance presented by the path.

Evaluating the delay effort of single-stage and multi-stage logic gates arises some important considerations when analyzing the gains presented on Table 5.4 in favor of NCSP logic gates. Single-stage gates tend to present higher logical effort when compared to the individual gates available into their multi-stage counterparts, as there is more capacitance associated to each individual input. However, two parameters go against multi-stage gates: 1) the logical efforts of each stage along a path are multiplied by each other and 2) the branching effort, i.e. each stage may drive capacitances presented in the on-path, as well the capacitances in the off-path.

Despite the limitations of logical effort, which is based on the linear delay model (WESTE, 2006), it still works remarkably well for many practical applications as it is a very fast, straightforward and reasonably accurate. It has been widely used in a variety of application domains (STOK, 1999; HU, 2003; KARANDIKAR, 2004) as well as in industry standard EDA synthesis tools (STOK, 1996; MAGMA, 2007; NANGATE, 2007).

Table 5.5 shows, among other metrics, the average logical effort obtained for all 3984 P-classes of 4-input functions implemented through six different transistor-level construction techniques. The logical effort was calculated using a PMOS/NMOS-ratio equal to 2, as it is seen in the examples in (SUTHERLAND, 1999). It can be seen that the average logical efforts for the transistor-level constructions based on the lower bound for the number stacked transistors – NCSP and CMOS LB (Lower Bound) from BDDs (ROSA, 2007) – are significantly smaller than the ones calculated for other logic styles.

Table 5.5: Comparison of six different methods for transistor-level network generation.

Metrics	CSP_1	CSP_2	*NCSP*	CMOS from BDDs[a]	Opt. CMOS from BDDs[b]	*CMOS LB from BDDs*[c]
Σ *transistors*	75530	75456	*75889*	75774	73438	*72307*
Σ *PU_length*	11954	13084	*11954*	15538	14227	*11954*
Σ *PD_length*	17009	15931	*14242*	15538	15321	*14242*
Logical Effort (average)	4.54	4.37	*3.83*	4.35	4.07	*3.68*
Functions not respecting LB	2312	2312	*0*	3148	2373	*0*
Unfeasible functions	1546	791	*0*	0	0	*0*

(a) (REIS, 1995) – (b) (POLI, 2003) – (c) (ROSA, 2007)

The logical effort for circuits based on the 'lower bound' is smaller because the base sizing used for individual logic gates in the logical effort approach is done based on the longest path between V_{DD}/GND and the output a transistor belongs to. It means that the smaller the stack lengths, the smaller the sizes of the transistors and, consequently, the smaller the input capacitance and the smaller the logical effort.

6 CONCLUSIONS

This work presented a method to derive the exact lower bound for the number of stacked switches needed to implement a logic function at the switch level. It was demonstrated that the most used transistor topologies – pass-transistor logic (PTL) and complementary series/parallel (CSP) – will not respect the presented lower bound for most 4+ inputs logic functions. This was demonstrated through the sum of the lengths of pull-up and pull-down longest transistor stacks for each cell in the critical path for circuits mapped with 3- to 6-input logic functions. The impact of this overhead in the number of stacked transistors in the delay of mapped circuits was evaluated through SPICE simulations, and compared to the delay of NCSP networks proposed here. The reduction of the length of pull-up and pull-down transistors stacks translates into better timing for the NCSP approach. It is believed that the lower bounds presented here will have a significant impact in the design of high performance integrated circuits using methods like the one in (SHELAR, 2001) and (ROY, 2005), through the guidance for the choice of better cell topologies. Note that the NCSP topology and the lower bounds proposed here are valid and useful independently of the decomposition used (BUCH, 1997; SCHOLL, 2000; HSIAO, 2000; LINDGREN, 2001; SHELAR, 2001; SHELAR, 2002) and could possibly increase the speed of many circuits by simply exchanging the switch-level cell networks in the critical paths with optimized versions.

The approach in (ROY, 2005) clearly states that the cell transistor network generation is an important point in their performance oriented design flow. Their paper shows examples that use complex cells with reduced pull-up and pull-down paths. However, they make no explicit mention to the PUN and PDN stack length of the cells as important parameters. This work is the first to explicitly and unequivocally address the importance of these parameters, to the best knowledge of the author.

The new CMOS logic style (NCSP) presented here is very promising. It was shown that the reduction of the number of stacked transistors can significantly improve networks. Moreover, it was demonstrated that respecting the lower bounds for the number of staked transistors while generating PUN and PDN make a much broader range of circuits feasible, as depicted on Table 3.7. Single-by-single gate comparison to standard CMOS topologies (CSP) has demonstrated the expected cell timing reduction. More complex NCSP gates, not feasible in a single CSP one, presents even more significant performance improvements that can be efficiently explored by logic synthesis tools. Improved networks, like the ones in (ROSA, 2007) which also take advantage of the lower bound theory may present even better results for some sets of logic functions.

The availability of cells respecting the lower bound presented here suits very well with library-free technology mapping which can explore a huge set of complex gates and generate them on-the-fly when they are needed. The work in (MARQUES, 2007) presents a DAG-based library-less technology mapping solution which considers the theory presented on Chapter 3 in order to improve the speed of circuits by reducing the number of stacked transistors in the critical path. It is important to highlight that the use of library-free technology mapping is seen as a very important solution for the gap between cell-based and full-custom designs. It is estimated that automated design flows using fixed libraries deliver circuits slower by at least a factor of 6 and consume a larger area at least by a factor of 10 (CHINNERY, 2002).

As presented by the graphs on Chapter 5, the NCSP cells are not the best choice for every logic function. Both delay and power consumption on logic gates are mainly due to both transistor channel resistance and node capacitance, and on NCSP cells one may reduce the resistive paths between V_{DD}/V_{SS} and the output but they may also present more capacitive nodes. A good example for this fact is the different implementations for the function represented by equations (4.5), which does not respect the ‘lower bound’, and (4.7), which respects the ‘lower bound’: the product $b \cdot c \cdot \overline{d} \cdot e$ (one and-stack of four transistors) in equation (4.5) is replaced by the pair of products $\overline{a} \cdot \overline{d} \cdot e + a \cdot b \cdot c$ (two and-stacks of three transistors) in equation (4.7). A good and straightforward way of reducing both node capacitance and path resistance is to combine the Branch-based logic with the lower bound theory. Finally, using a more appropriate transistor sizing strategy might provide even better performance results for NCSP cells.

It was noticed that in terms of area, NCSP cells present reductions for some logic functions as well as increased area for others, when compared with CSP. The difference is small though. When comparing the total number of transistors for all 4-input P-classes, as shown on Table 5.5, NCSP cells presents an increase of only 0.5%. Besides that, the values on Table 5.5 consider that all 3984 P-classes implemented in CSP are feasible, i.e. has less than four stacked transistors, which is not correct. When a CSP function is not feasible in a single stage, it needs to be split into more stages, and for each extra stage at least an extra pair of transistors needs to be accounted.

A strategic advantage of NCSP cells is that they can be perfectly mixed with CSP ones. Hence, a synthesis tool could easily choose among CSP and NCSP cells, taking the smaller ones for area improvement, and the faster ones, no matter the size, to be used in the critical path.

Finally, in terms of actual layout, no advanced study has been profoundly carried out in this work, mainly due to unavailability of tools for automatic generation of layouts using the NCSP logic style. Nevertheless, there are both favorable and unfavorable cases for NCSP. As NCSP cells may have topologically non-complementary (also known as auto-dual) PUN and PDN, those may also have a different number of transistors, which may require a more advanced transistor-level placement (transistor ordering) routine. For the same reason, the intra-cell routing might be also more difficult. However, in some cases one may obtain networks that are topologically symmetric in NCSP, which cannot be achieved with regular constructive method of CSP. In these cases, the ordering would have the same complexity as the one for CSP cells, but the layout would be more regular and easy to route.

REFERENCES

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

BERKELAAR, M.R.C.M.; JESS, J.A.G. Technology mapping for standard-cell generators. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1988, Santa Clara, USA. **Digest of Technical Papers.** New York: IEEE, 1988, p. 470 – 473.

BRAYTON, R.K. Factoring Logic Functions. **IBM Journal of Research and Development**, v. 31, n. 2, p. 187 – 198, Mar. 1987.

BRAYTON, R.K. et al. **Logic Minimization Algorithms for VLSI Synthesis.** Boston: Kluwer Academic Publishers, 1984.

BUCH, P. et al. Logic synthesis for large pass transistor circuits. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1997, San Jose, USA. **Proceedings…** New York: IEEE, 1997, p. 663 – 670.

CHEN, W-K. **The VLSI Handbook.** [S.l.]: CRC Press, 2000.

CHINNERY, D.; KEUTZER, K. **Closing the Gap Between ASIC & Custom:** Tools and Techniques for High-Performance ASIC Design. Boston: Kluwer Academic Publishers, 2002.

CORREIA, V.P., REIS, A.I. Classifying n-Input Boolean Functions. In: WORKSHOP IBERCHIP, 2001, Montevideo, Uruguay. **Memorias.** Montevideo: Universidad de la Republica, 2001.

CORREIA, V.P., REIS, A.I. Advanced Technology Mapping for Standard-cell Generators. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2004, Pernambuco, Brazil. **Proceedings...**, 2004.

DETJENS, E. et al. Technology mapping in MIS. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1987. **Proceedings…** p. 116-119.

ELRABAA, M.S.; ELMASRY, M.I.; Design and optimization of buffer chains and logic circuits in a BiCMOS environment. **IEEE Journal of Solid-State Circuits**, v. 27, n. 5, p. 792 –801, May 1992.

GAJSKI, D.D. **Silicon Compilation.** Reading: Addison-Wesley, 1988.

GAVRILOV, S. et al. Library-less synthesis for static CMOS combinational logic circuits. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 1997, San Jose, USA. **Proceedings…** New York: IEEE, 1997, p. 658 – 662.

GREA, R.A; HIGONNET, R.A. Table of four-relay contact networks. In: MOORE, E.F. **Logical Design of Electrical Circuits**, New York: McGraw-Hill, 1958.

HACHTEL, G.D.; SOMENZI, F. **Logic Synthesis and Verification Algorithms. Publisher**. [S.l.]: Kluwer Academic Publishers, 1996.

HSIAO, S-F.; YEH, J-S.; CHEN, D-Y. High-performance multiplexer-based logic synthesis using pass-transistor logic. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2000, Geneva, Switzerland. **Proceedings…** p. 325-328.

HU, B. et al. Gain-Based Technology Mapping for Discrete-Size Cell Libraries. In: DESIGN AUTOMATION CONFERENCE, DAC, 2003, Anaheim, CA, USA. **Proceedings…** p. 574 – 579.

INTEL CORPORATION. **Moore's Law, The Future – Technology & Research at Intel.** Available at: <http://www.intel.com/technology/mooreslaw/index.htm>, Visited on: March 2007.

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.

JURGEN, R.K. **Digital Consumer Electronics Handbook.** [S.l.]: Mc-Graw-Hill, 1997.

KABBANI, A.; AL-KHALILI, D.; AL-KHALILI, A.J. Delay analysis of CMOS gates using modified logical effort model. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, New York,** v. 24, n. 6, p. 937-947, June 2005.

KARANDIKAR, S.K.; SAPATNEKAR, S.S. Logical Effort Based Technology Mapping. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 2004, San Jose, CA, USA. **Proceedings…** New York: IEEE, 2004, p. 419 – 422.

KEUTZER, K. Cell libraries - build vs. buy; static vs. dynamic. In: DESIGN AUTOMATION CONFERENCE, DAC, 1999, New Orleans, LA, USA. **Proceedings…** p. 341 – 342.

LEFEBVRE, M.; MARPLE, D.; SECHEN, C. The future of custom cell generation in physical synthesis. In: DESIGN AUTOMATION CONFERENCE, DAC, 1997, Anaheim, CA, USA. **Proceedings…** p. 446 – 51.

LINDGREN, P.; KERTHU, M.; THORNTON, M.; DRESCHSLER, R. Low-power optimization technique for BDD mapped circuits. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 2001, Yokohama, Japan. **Proceedings…** p. 615 – 621.

MAGMA DESIGN AUTOMATION. **Gain Based Synthesis**: Speeding RTL to Silicon. Available at: < http://www.magma-da.com>. Visited on: February 2007.

MCCLUSKEY, E.J. Minimization of Boolean functions. **Bell Systems Technical Journal**, [S.l.], v. 5, n. 5, p. 1417 – 1444, 1956.

MARQUES, F.S.; ROSA, L.S.; RIBAS, R.P.; SAPATNEKAR, S.S.; REIS, A.I. DAG Based Library-Free Technology Mapping. In: ACM GREAT LAKES SYMPOSIUM ON VLSI, GLSVLSI, 2007, New York, NY, USA. **Proceedings…**

MEAD, A.C.; CONWAY, L.A. **Introduction to VLSI Systems.** Reading: Addison-Wesley, 1980.

MICHELI, G.D. **Synthesis and optimization of digital circuits.** New York: McGraw-Hill, 1994.

MINATO, S. **Binary Decision Diagrams and Applications for VLSI CAD.** Boston: Kluwer, 1996.

MOORE, G.E. Cramming more Components into Integrated Circuits, **Electronics**, [S.l.], v. 38, n. 8, Apr. 1965.

MOORE, G.E. No exponential is forever. In: IEEE INTERNATIONAL SOLID-STATE CIRCUIT CONFERENCE, 2003. **Digest of Technical Papers.** New York: IEEE, 2003, p. 20 – 23.

MORAES, F.; ROBERT, M.; AUVERGNE, D. A Virtual CMOS Library Approach for Fast Layout Synthesis. In: IFIP INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION: SYSTEMS ON A CHIP, 1999, **Proceedings…** p. 415 – 426.

NAGAYAMA, S.; SASAO, T. On the minimization of longest path length for decision diagrams. In: INTERNATIONAL WORKSHOP ON LOGIC AND SYNTHESIS, IWLS, 2004. **Proceedings…** p. 28 – 35.

NANGATE. **Nangate Library Characterizer.** Available at: <http://www.nangate.com>. Visited on: February 2007.

NÈVE, A.; FLANDRE, D. Branch-Based Logic for High Performance Carry-Select Adders in 0.25 μm Bulk and Silicon-On-Insulator CMOS Technologies. In: INTERNATIONAL WORKSHOP ON POWER AND TIMING MODELING, OPTIMIZATION AND SIMULATION, PATMOS, Yverdon-Les-Bains, Switzerland. **Proceedings…** Sep. 2001.

NINOMIYA, I. Table of minimal switching circuits for functions of four variables. In: **HARRISON, M.A., Introduction to Switching and Automata Theory**. [S.l.]: McGraw-Hill, 1966.

PIGUET, C. et al. A Metal-Oriented Layout Structure for CMOS Logic. **IEEE Journal of Solid-state Circuits,** New York, v. 19, n.3, p. 425 – 436, June 1984.

PIGUET, C. et al. Low-power low-voltage digital CMOS cell design. In: INTERNATIONAL WORKSHOP ON POWER AND TIMING MODELING, OPTIMIZATION AND SIMULATION, PATMOS, 1994, Oldenburg, Germany. **Proceedings…** p. 132–139.

PIGUET, C. et al. Low-power low-voltage standard cell libraries, In: Solid-State Circuits Conference, ESSCIRC, 1995, **Proceedings…**

PILLEGI. L. et al. Exploring regular fabrics to optimize the performance-cost trade-off. In: DESIGN AUTOMATION CONFERENCE, DAC, Anaheim, CA, USA. 2003. **Proceedings…**

POLI, R.E.B.; SCHNEIDER, F.R.; RIBAS, R.P.; REIS, A.I. Unified theory to build cell-level transistor networks from BDDs. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2003, São Paulo, SP, Brazil. **Proceedings…** p. 199 – 204.

QUINE, W. V. A way to simplify truth functions. **American Mathematical Monthly**, [S.l], v. 62, n. 9, p. 627 – 631, 1965.

RABAEY, J.M.; CHANDRAKASAN, A.; NIKOLIC, B. **Digital Integrated Circuits:** A Design Perspective**.** Upper Saddle River: Prentice-Hall, 2003.

REIS, A.; ROBERT, M.; AUVERGNE, D.; REIS, R. Associating CMOS transistors with BDD arcs for technology mapping. **IEE Electronics Letters**, [S.l.], v. 31, n. 14, p. 1118 – 20, July 1995.

REIS, A.; REIS, R.; AUVERGNE, D.; ROBERT, M. Library free technology mapping. In: IFIP INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, 1997, **Proceedings…**

REIS, A.; REIS, R.; ROBERT, M. Topological Parameters for Library Free Technology Mapping. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 1998. **Proceedings…**

ROSA, L.S.; MARQUES, F.; CARDOSO, T.M.G.; RIBAS, R.P.; SAPATNEKAR, S.S.; REIS, A.I. Fast Transistor Networks from BDDs. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2006. **Proceedings…**

ROSA, L.S, SCHNEIDER, F.R.; RIBAS, R.P., REIS, A.I. Analysis of Transistor Networks Generation. In WORKSHOP IBERCHIP, 2007. **Memorias.**

ROY, K.; MUKHOPADHYAY, S.; MAHMOODI-MEIMAND, H. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. **Proceeding of the IEEE**, New York, v. 91, n. 2, February 2003.

ROY, R.; BHATTACHARYA, D.; BOPPANA, V. Transistor-Level Optimization of Digital Designs with Flex Cells. **Computer**, 2005.

SASAO, T. **Switching Theory for Logic Synthesis.** Boston: Kluwer Academic Publishers. 1999.

SCHNEIDER, F.R.; RIBAS, R.P.; SAPATNEKAR, S.S.; REIS, A.I. Exact lower bound for the number of switches in series to implement a combinational logic cell. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, ICCD, 2005, San Jose, CA, USA. **Proceedings...** p. 357 – 362, Oct. 2005.

SCHNEIDER, F.R.; RIBAS, R.P.; REIS, A.I. Fast CMOS Logic Style Using Minimum Transistor Stack for Pull-up and Pull-down Networks. In: INTERNATIONAL WORKSHOP ON LOGIC AND SYNTHESIS, IWLS, 2006a. **Proceedings...**

SCHNEIDER, F.R.; RIBAS, R.P.; REIS, A.I. CMOS Logic Gates Based on the Minimum Theoretical Number of Transistor in Series. In: NORCHIP Conference, Linköping, Sweden. 2006b. **Proceedings...**

SCHOLL, C.; BECKER, B. On the generation of multiplexer circuits for pass transistor logic. In: DESIGN, AUTOMATION, AND TEST IN EUROPE, DATE, 2000. **Proceedings...** p. 372 – 378.

SCOTT, K.; KEUTZER, K. Improving Cell Libraries for Synthesis. In: CUSTOM INTEGRATED CIRCUITS CONFERENCE, CICC, 1994. **Proceedings...** p. 128-131.

SECHEN, C. Libraries: lifejacket or straitjacket. In: DESIGN AUTOMATION CONFERENCE, DAC, 2003, Anaheim, CA, USA. **Proceedings...** p. 642 – 643.

SENTOVICH, E.M. et al. **SIS**: A system for sequential circuit synthesis. Berkeley: Technical Report No. UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

SHELAR, R.S.; SAPATNEKAR, S.S. Recursive bipartitioning of BDDs for performance driven synthesis of pass transistor logic circuits. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 2001. **Proceedings...** New York: IEEE, p. 449 – 452.

SHELAR, R.S.; SAPATNEKAR, S.S. An efficient algorithm for low power pass transistor logic synthesis. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 2002. **Proceedings...** p. 87 – 92.

SHELAR, R.S.; SAPATNEKAR, S.S. BDD decomposition for delay oriented pass transistor logic synthesis. **IEEE Trans. on VLSI**, New York, v. 13, n. 8, p. 957 – 970, Aug. 2005.

STOK, L. et al. BooleDozer: Logic Synthesis for ASICs. **IBM Journal of Research and Development**, Armonk, v. 40, n. 4, p. 407-430, 1996.

STOK, L.; LYER, M.A.; SULLIVAN, A.J. Wavefront Technology Mapping. In: DESIGN, AUTOMATION, AND TEST IN EUROPE, DATE, 1999, **Proceedings...** p. 531–536.

SUTHERLAND, I.; SPROULL, B.; HARRIS, D. **Logical Effort:** Designing Fast CMOS Circuits**.** San Francisco, USA: Morgan Kaufmann Publishers, 1999.

THORP, T.J.; YEE, G.S.; SECHEN, C.M. Design and synthesis of dynamic circuits. **IEEE Transactions on VLSI**, New York, v. 11, n. 1, p. 141 – 149, Feb. 2003.

UYEMURA, J. P. **CMOS Logic Circuit Design.** Norwell: Kluwer Academic Publishers, 1999.

VUJKOVIC, M., SECHEN, C. Optimized power-delay curve generation for standard cell ICs. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 2002. **Proceedings…**

WESTE, N.H.E.; ESHRAGHIAN, K. **Principles of CMOS VLSI Design:** A Systems Perspective. 2nd ed. Reading: Addison-Wesley, 1993.

WESTE, N.H.E.; HARRIS, D. **CMOS VLSI Design:** A Circuits and Systems Perspective. 3rd ed. Boston: Pearson/Addison-Wesley, 2005.

YANG, C.; CIESIELSKI, M. BDS: a BDD-based logic optimization system. **IEEE Transactions on CAD**, New York, v. 21, n. 7, p. 866 – 876. July 2002.

YANO, K.; YAMANAKA, T.; NISHIDA, T.; SAITO, M.; SHIMOHIGASHI, K.; SHIMIZU, A. A 3.8-ns CMOS 16x16-b multiplexer using complementary pass-transistor logic, **IEEE Journal of Solid-State Circuits**, Piscataway, v. 25, p. 388 – 395, 1990.

YANO, K. et al. Top-down pass-transistor logic design. **IEEE Journal of Solid-State Circuits**, Piscataway, v. 31, p. 792 – 803, 1996.

ZAHIRI, B. Structured ASICs: opportunities and challenges. In: INTERNATIONAL CONFERENCE IN CIRCUITS DESIGN, ICCD, 2003. **Proceedings…** p. 404 – 409.

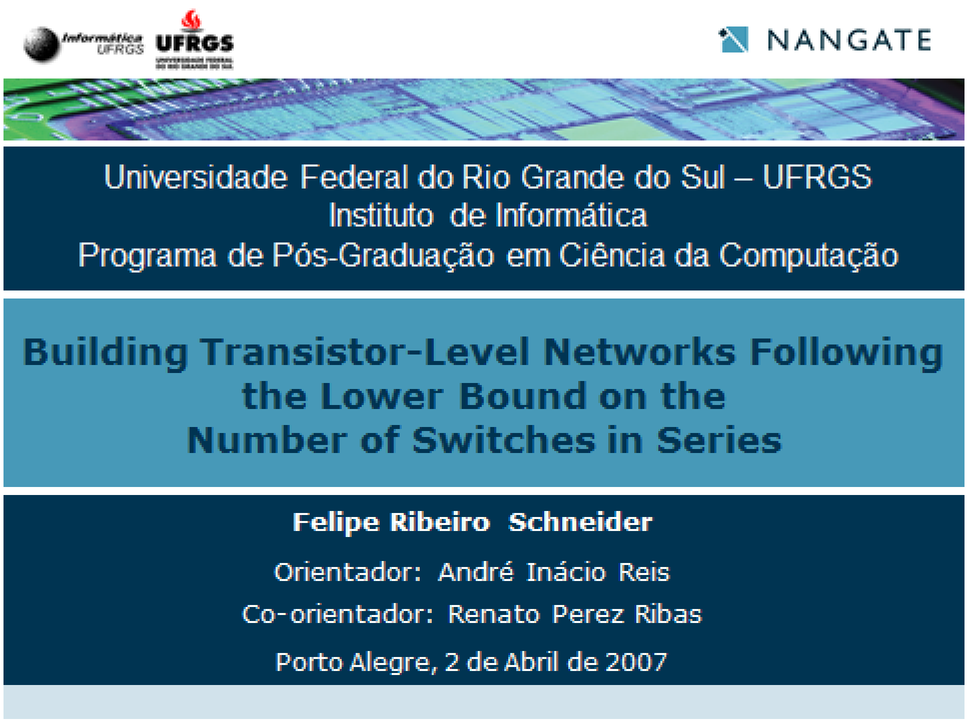
ZENASIS TECHNOLOGIES. Available at: <http://www.zenasis.com>. Visited on: February 2007.

ZHOU, H.; AZIZ, A. Buffer minimization in pass transistor logic. **IEEE Transactions on CAD**, New York, v. 20, n. 5, p. 693 – 697, May 2001.

ZIMMERMANN, R.; FICHTNER, W. Low-power logic styles: CMOS versus pass-transistor logic **IEEE Journal of Solid-State Circuits**, Piscataway, v. 32, no. 7, p. 1079 – 1090, July 1997.

ZUCHOWSKI, P.S.; REYNOLDS, C.B.; GRUPP, R.J.; DAVIS, S.G.; CREMEN, B.; TROXEL, B. A Hybrid ASIC and FPGA Design In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, ICCAD, 2002. **Proceedings…**

APPENDIX A: PRESENTATION SLIDES



Universidade Federal do Rio Grande do Sul – UFRGS
Instituto de Informática
Programa de Pós-Graduação em Ciência da Computação

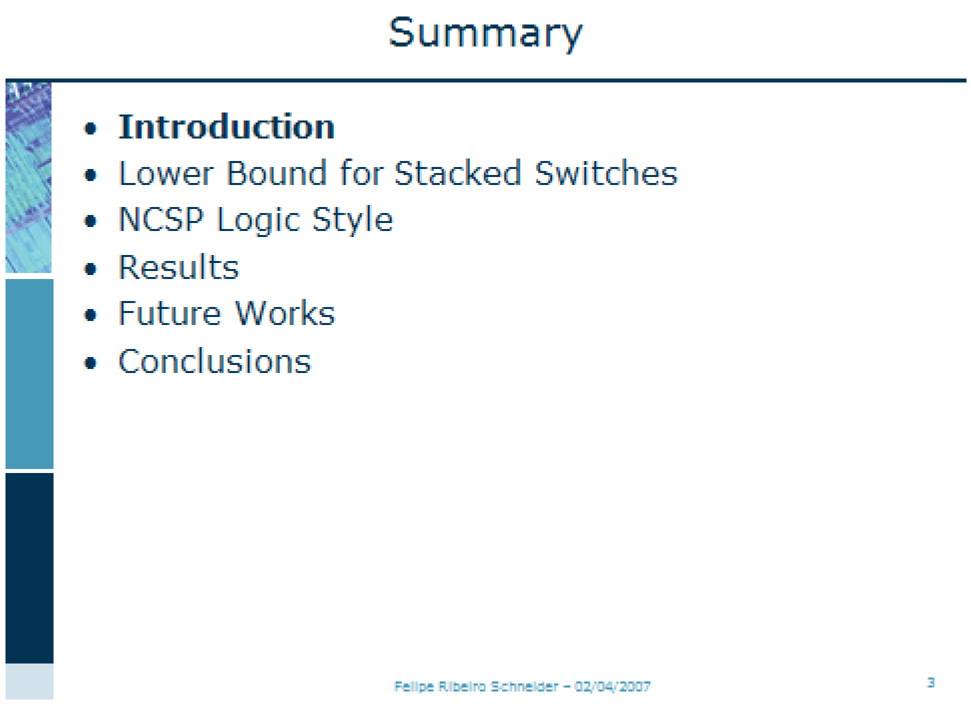
Building Transistor-Level Networks Following the Lower Bound on the Number of Switches in Series

Felipe Ribeiro Schneider

Orientador: André Inácio Reis

Co-orientador: Renato Perez Ribas

Porto Alegre, 2 de Abril de 2007

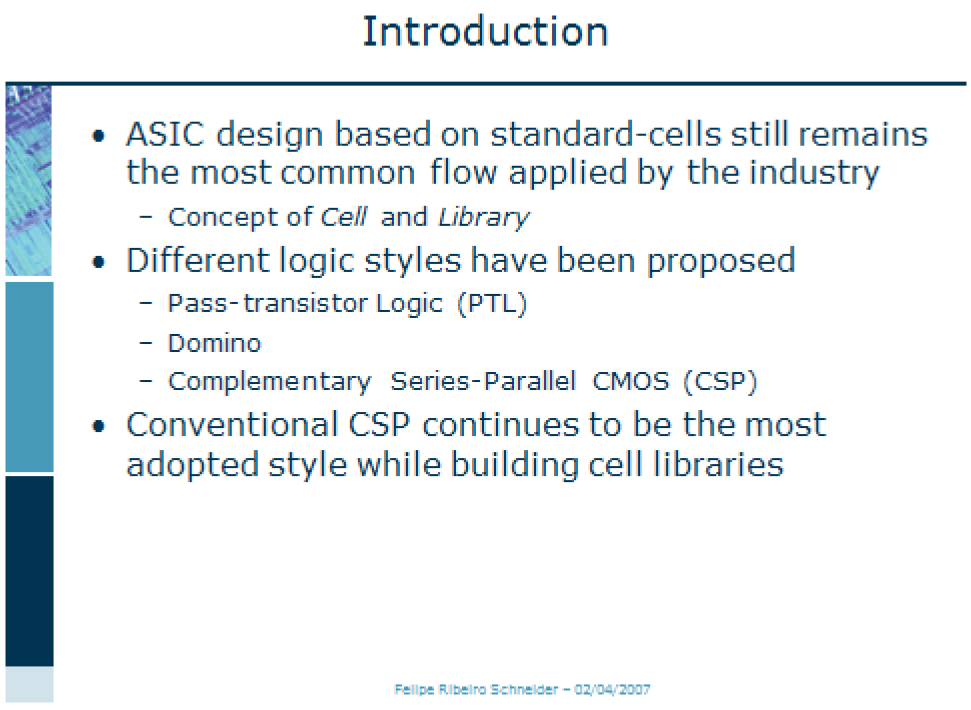


Summary

- **Introduction**
- Lower Bound for Stacked Switches
- NCSP Logic Style
- Results
- Future Works
- Conclusions

Felipe Ribeiro Schneider – 02/04/2007

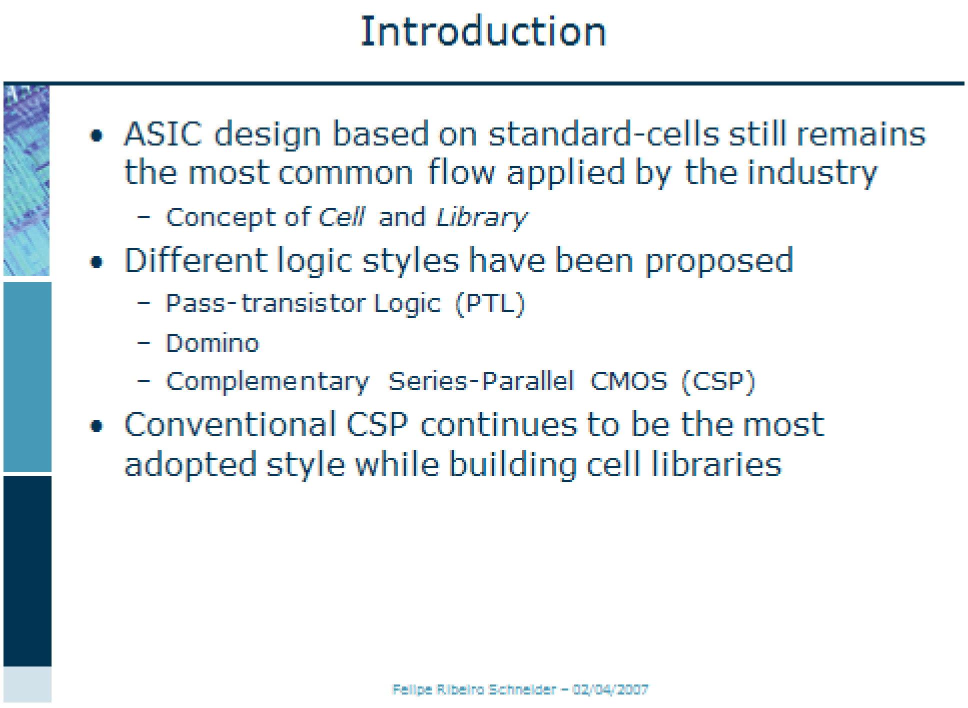
3



Introduction

- ASIC design based on standard-cells still remains the most common flow applied by the industry
  - Concept of *Cell* and *Library*
- Different logic styles have been proposed
  - Pass-transistor Logic (PTL)
  - Domino
  - Complementary Series-Parallel CMOS (CSP)
- Conventional CSP continues to be the most adopted style while building cell libraries

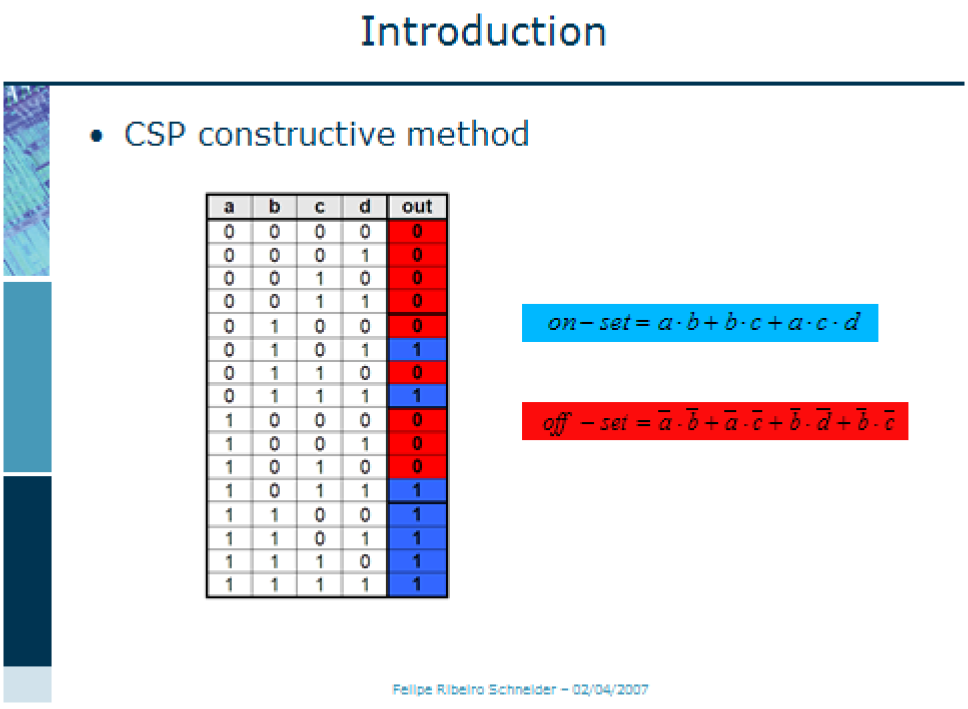
Felipe Ribeiro Schneider – 02/04/2007



Introduction

- ASIC design based on standard-cells still remains the most common flow applied by the industry
  - Concept of *Cell* and *Library*
- Different logic styles have been proposed
  - Pass-transistor Logic (PTL)
  - Domino
  - Complementary Series-Parallel CMOS (CSP)
- Conventional CSP continues to be the most adopted style while building cell libraries

Felipe Ribeiro Schneider – 02/04/2007



Introduction

- CSP constructive method

a	b	c	d	out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$on-set = a \cdot b + b \cdot c + a \cdot c \cdot d$$

$$off-set = \overline{a} \cdot \overline{b} + \overline{a} \cdot \overline{c} + \overline{b} \cdot \overline{d} + \overline{b} \cdot \overline{c}$$

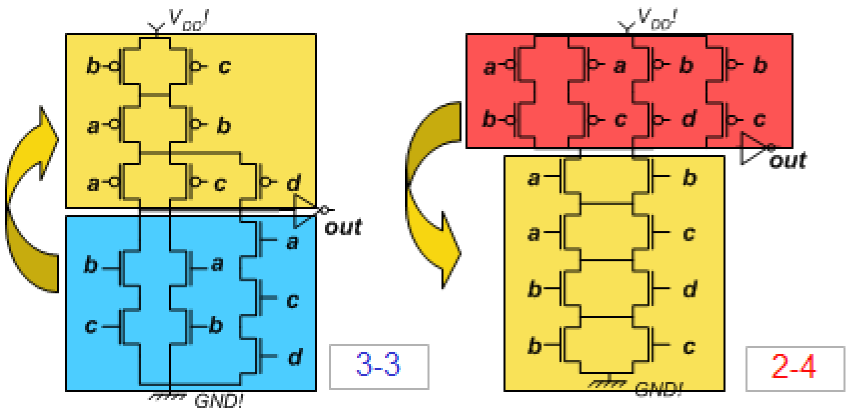
Felipe Ribeiro Schneider – 02/04/2007

Introduction

- CSP constructive method

$on-set = a \cdot b + b \cdot c + a \cdot c \cdot d$

$off-set = \bar{a} \cdot \bar{b} + \bar{a} \cdot \bar{c} + \bar{b} \cdot \bar{d} + \bar{b} \cdot \bar{c}$

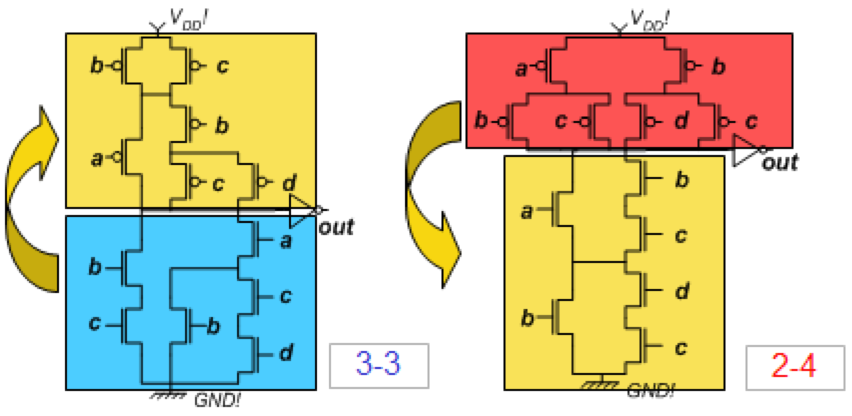


Introduction

- CSP constructive method

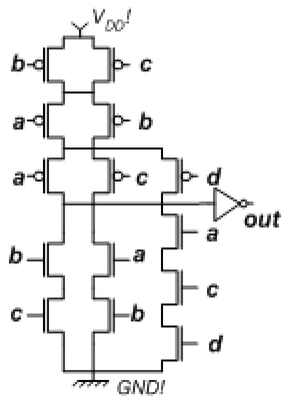
$on-set = a \cdot (b + c \cdot d) + c \cdot b$

$off-set = \bar{a} \cdot (\bar{b} + \bar{c}) + \bar{b} \cdot (\bar{d} + \bar{c})$



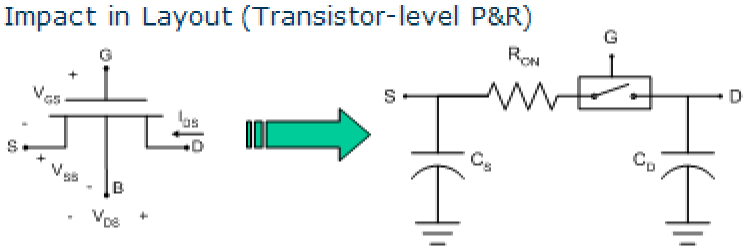
Introduction

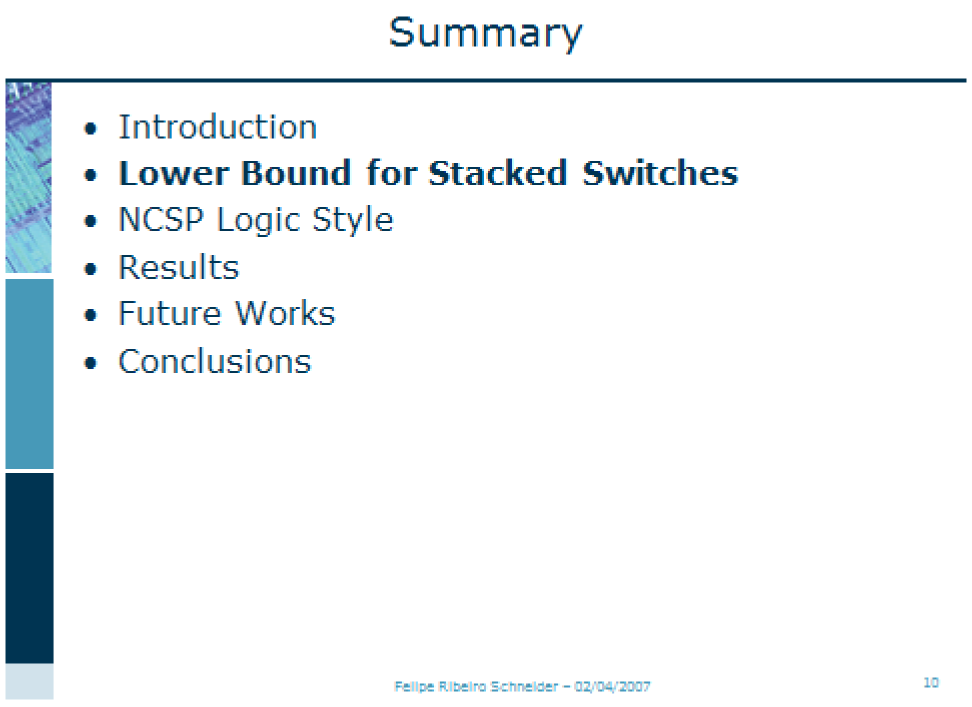
- Advantages of CSP
  - Low static power consumption
  - Robustness
    - Low sensitivity to noise
  - Consolidated cell design flow
  - Speed
    - While one plane is *on*, the other one is *off* most of the time
- Disadvantage of CSP
  - Transistor count
    - Higher input capacitance



Introduction

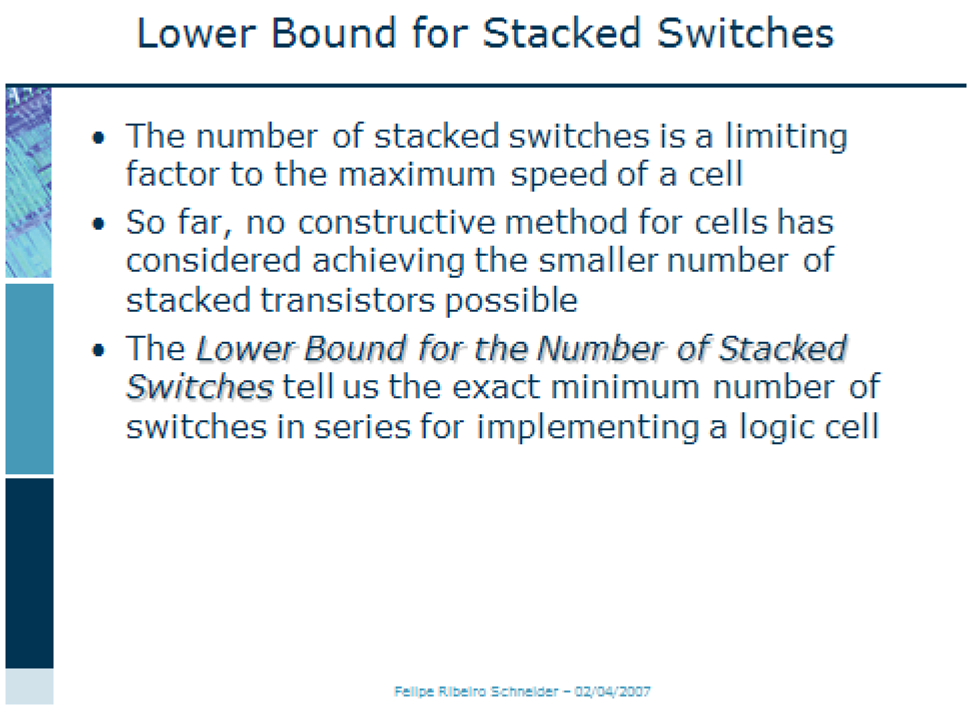
- Important metrics to be considered while designing cells
  - Transistor count (Area)
  - Resistance (Speed and Power)
  - Capacitance (Speed and Power)
  - Impact in Layout (Transistor-level P&R)





Summary

- Introduction
- **Lower Bound for Stacked Switches**
- NCSP Logic Style
- Results
- Future Works
- Conclusions



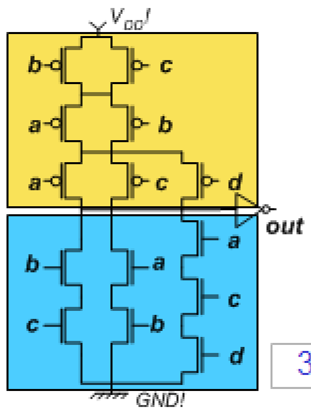
Lower Bound for Stacked Switches

- The number of stacked switches is a limiting factor to the maximum speed of a cell
- So far, no constructive method for cells has considered achieving the smaller number of stacked transistors possible
- The *Lower Bound for the Number of Stacked Switches* tell us the exact minimum number of switches in series for implementing a logic cell

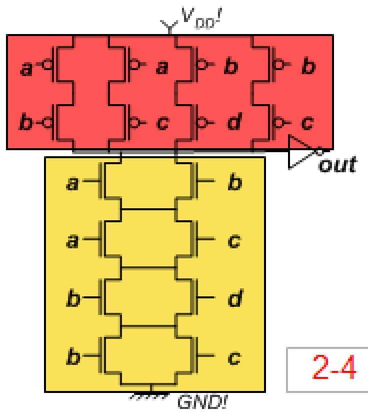
Lower Bound for Stacked Switches

$on-set = a \cdot b + b \cdot c + a \cdot c \cdot d$

$off-set = \bar{a} \cdot \bar{b} + \bar{a} \cdot \bar{c} + \bar{b} \cdot \bar{d} + \bar{b} \cdot \bar{c}$



3-3

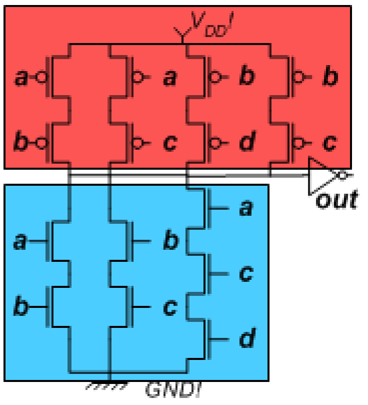


2-4

Lower Bound for Stacked Switches

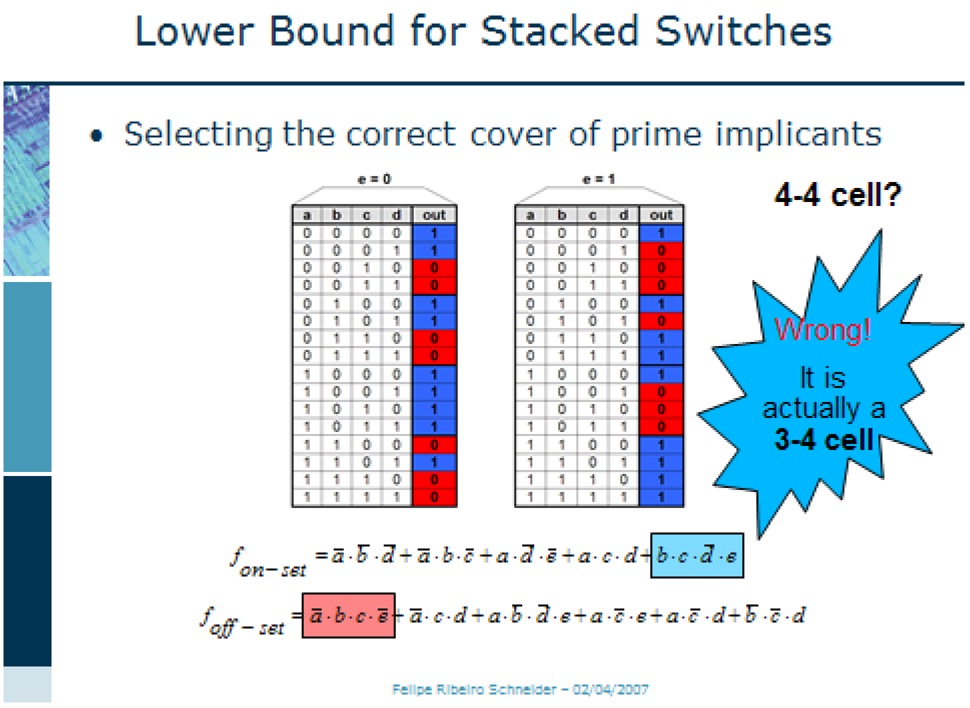
$on-set = a \cdot b + b \cdot c + \boxed{a \cdot c \cdot d}$

$off-set = \boxed{\bar{a} \cdot \bar{b}} + \bar{a} \cdot \bar{c} + \bar{b} \cdot \bar{d} + \bar{b} \cdot \bar{c}$



The product with more literals determines the stack length

2-3



Lower Bound for Stacked Switches

- Selecting the correct cover of prime implicants

e = 0

a	b	c	d	out
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

e = 1

a	b	c	d	out
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

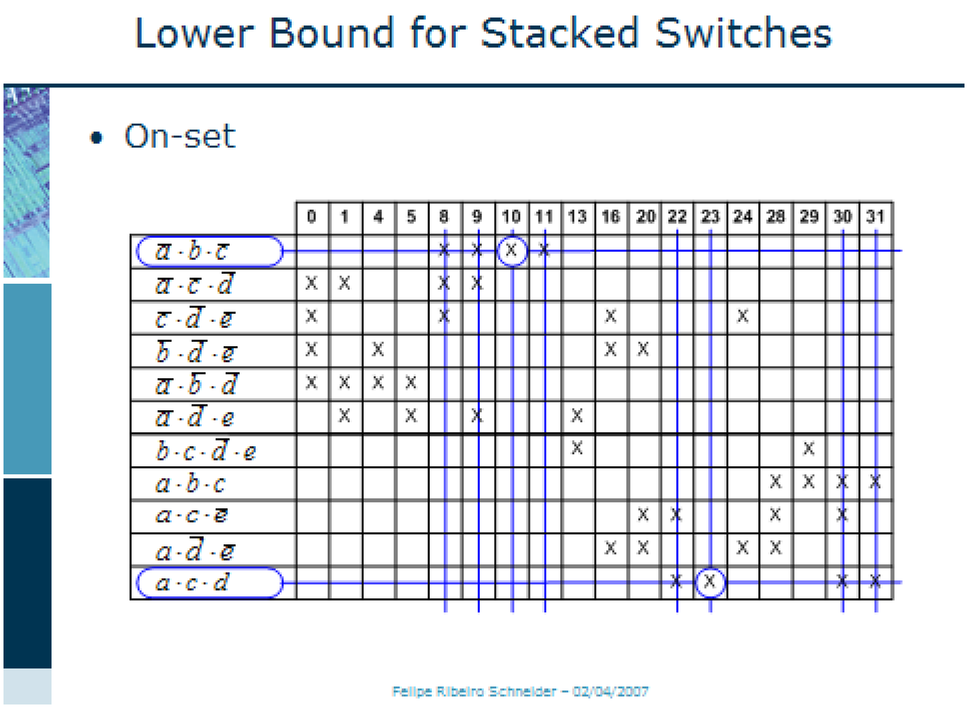
4-4 cell?

Wrong!

It is actually a **3-4 cell**

$$f_{on-set} = \bar{a}\cdot\bar{b}\cdot\bar{d} + \bar{a}\cdot b\cdot\bar{c} + a\cdot\bar{d}\cdot\bar{e} + a\cdot c\cdot d + \boxed{b\cdot c\cdot\bar{d}\cdot e}$$

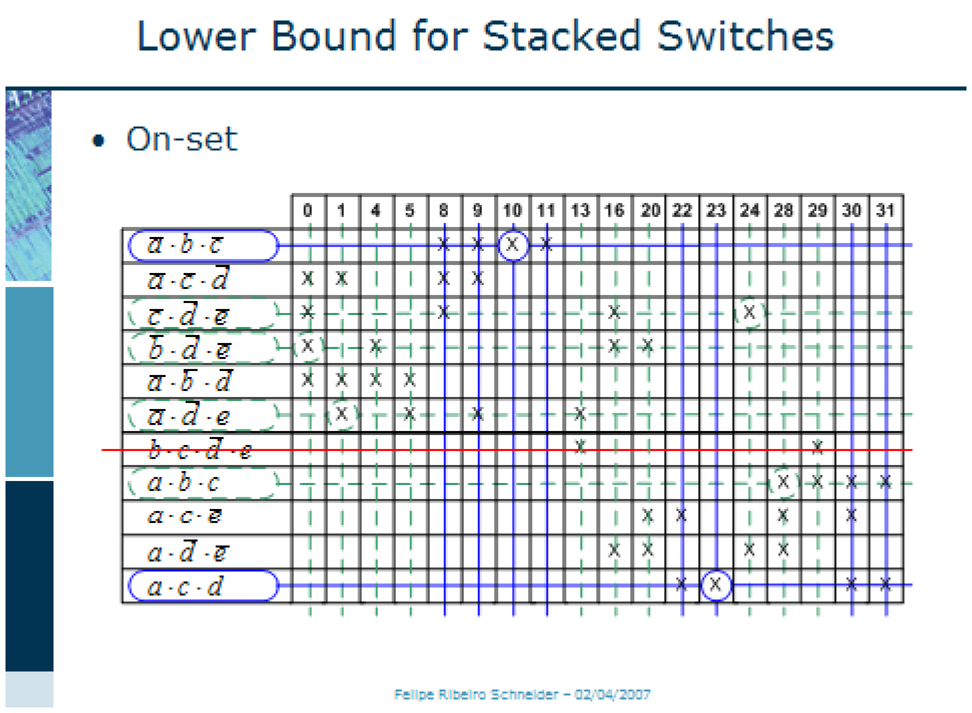
$$f_{off-set} = \boxed{\bar{a}\cdot b\cdot c\cdot\bar{e}} + \bar{a}\cdot c\cdot d + a\cdot\bar{b}\cdot\bar{d}\cdot e + a\cdot\bar{c}\cdot e + a\cdot\bar{c}\cdot d + \bar{b}\cdot\bar{c}\cdot d$$



Lower Bound for Stacked Switches

- On-set

	0	1	4	5	8	9	10	11	13	16	20	22	23	24	28	29	30	31
$\bar{a}\cdot b\cdot\bar{c}$					X	X	X	X										
$\bar{a}\cdot\bar{c}\cdot\bar{d}$	X	X			X	X												
$\bar{c}\cdot\bar{d}\cdot\bar{e}$	X				X					X				X				
$\bar{b}\cdot\bar{d}\cdot\bar{e}$	X		X							X	X							
$\bar{a}\cdot\bar{b}\cdot\bar{d}$	X	X	X	X														
$\bar{a}\cdot\bar{d}\cdot e$		X		X		X			X									
$b\cdot c\cdot\bar{d}\cdot e$									X							X		
$a\cdot b\cdot c$															X	X	X	X
$a\cdot c\cdot\bar{e}$											X	X			X		X	
$a\cdot\bar{d}\cdot\bar{e}$										X	X			X	X			
$a\cdot c\cdot d$												X	X				X	X

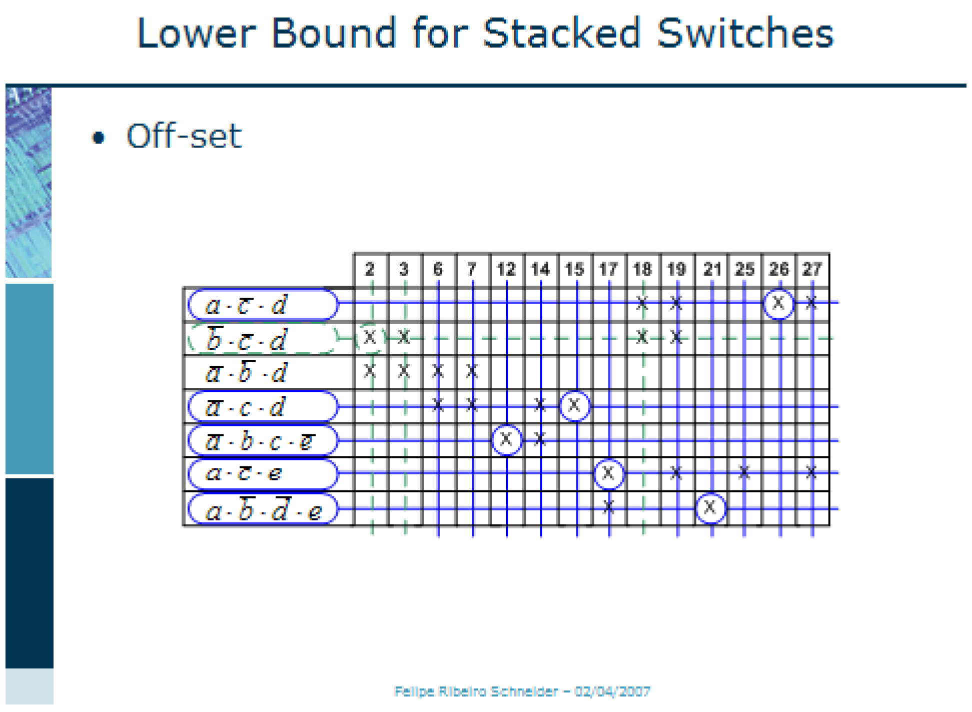


Lower Bound for Stacked Switches

- On-set

	0	1	4	5	8	9	10	11	13	16	20	22	23	24	28	29	30	31
$\bar a\cdot b\cdot\bar c$					X	X	X	X										
$\bar a\cdot\bar c\cdot\bar d$	X	X			X	X												
$\bar c\cdot\bar d\cdot\bar e$	X				X					X				X				
$\bar b\cdot\bar d\cdot\bar e$	X		X							X	X							
$\bar a\cdot\bar b\cdot\bar d$	X	X	X	X														
$\bar a\cdot\bar d\cdot e$		X		X		X			X									
~~$b\cdot c\cdot\bar d\cdot e$~~									X							X		
$a\cdot b\cdot c$															X	X	X	X
$a\cdot c\cdot\bar e$											X	X			X		X	
$a\cdot\bar d\cdot\bar e$										X	X			X	X			
$a\cdot c\cdot d$												X	X				X	X

Felipe Ribeiro Schneider – 02/04/2007



Lower Bound for Stacked Switches

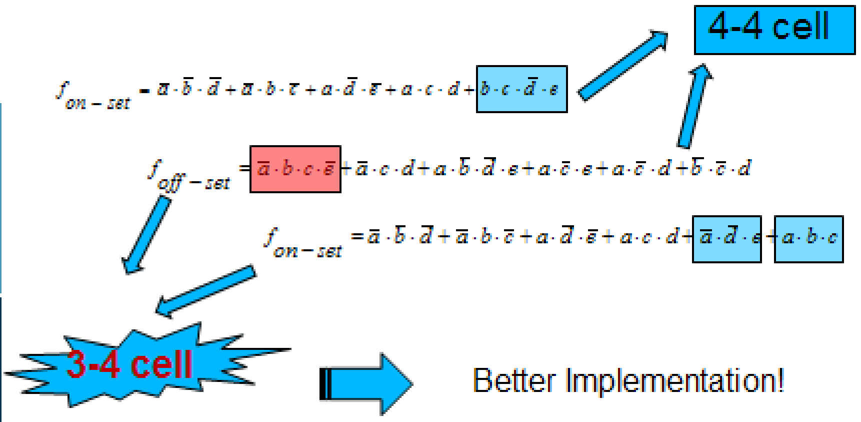
- Off-set

	2	3	6	7	12	14	15	17	18	19	21	25	26	27
$a\cdot\bar c\cdot d$									X	X			X	X
$\bar b\cdot\bar c\cdot d$	X	X							X	X				
$\bar a\cdot\bar b\cdot d$	X	X	X	X										
$\bar a\cdot c\cdot d$			X	X		X	X							
$\bar a\cdot b\cdot c\cdot\bar e$					X	X								
$a\cdot\bar c\cdot e$								X		X		X		X
$a\cdot\bar b\cdot\bar d\cdot e$								X			X			

Felipe Ribeiro Schneider – 02/04/2007

Lower Bound for Stacked Switches

- Selecting the correct cover of prime implicants



$$f_{on-set} = \bar{a}\cdot\bar{b}\cdot\bar{d}+\bar{a}\cdot b\cdot\bar{c}+a\cdot\bar{d}\cdot\bar{e}+a\cdot c\cdot d+\boxed{b\cdot c\cdot\bar{d}\cdot e}$$

$$f_{off-set} = \boxed{\bar{a}\cdot b\cdot c\cdot\bar{e}}+\bar{a}\cdot c\cdot d+a\cdot\bar{b}\cdot\bar{d}\cdot e+a\cdot\bar{c}\cdot e+a\cdot\bar{c}\cdot d+\bar{b}\cdot\bar{c}\cdot d$$

$$f_{on-set} = \bar{a}\cdot\bar{b}\cdot\bar{d}+\bar{a}\cdot b\cdot\bar{c}+a\cdot\bar{d}\cdot\bar{e}+a\cdot c\cdot d+\boxed{\bar{a}\cdot\bar{d}\cdot e}+\boxed{a\cdot b\cdot c}$$

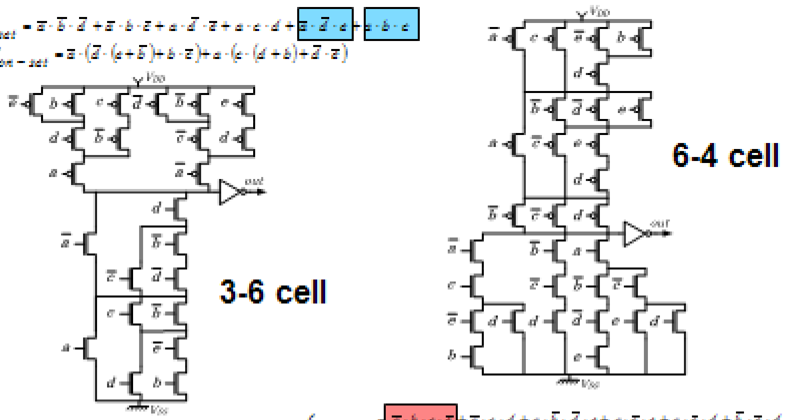
Better Implementation!

Lower Bound for Stacked Switches

- CSP implementation

$$f_{on-set} = \bar{a}\cdot\bar{b}\cdot\bar{d}+\bar{a}\cdot b\cdot\bar{c}+a\cdot\bar{d}\cdot\bar{e}+a\cdot c\cdot d+\boxed{\bar{a}\cdot\bar{d}\cdot e}+\boxed{a\cdot b\cdot c}$$

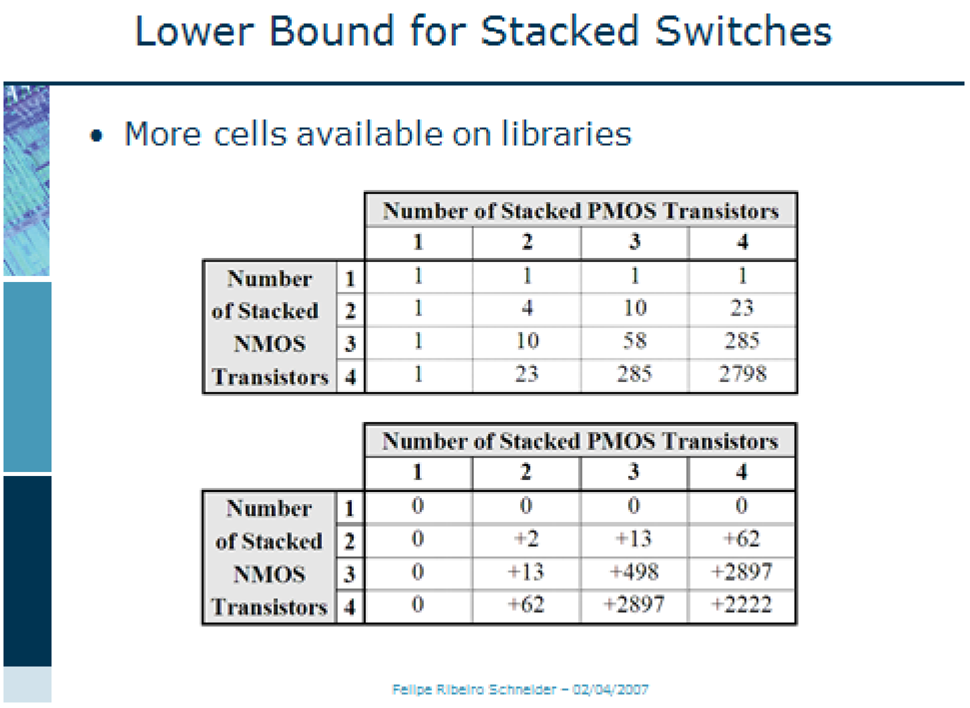
$$f_{on-set} = \bar{a}\cdot\left(\bar{d}\cdot\left(e+\bar{b}\right)+b\cdot\bar{c}\right)+a\cdot\left(c\cdot\left(d+b\right)+\bar{d}\cdot\bar{e}\right)$$



$$f_{off-set} = \boxed{\bar{a}\cdot b\cdot c\cdot\bar{e}}+\bar{a}\cdot c\cdot d+a\cdot\bar{b}\cdot\bar{d}\cdot e+a\cdot\bar{c}\cdot e+a\cdot\bar{c}\cdot d+\bar{b}\cdot\bar{c}\cdot d$$

$$f_{off-set} = \bar{a}\cdot c\cdot\left(b\cdot\bar{e}+d\right)+\bar{b}\cdot\bar{c}\cdot d+a\cdot\left(\bar{b}\cdot\bar{d}\cdot e+\bar{c}\cdot\left(e+d\right)\right)$$

There is a 3-4 cell for this function!



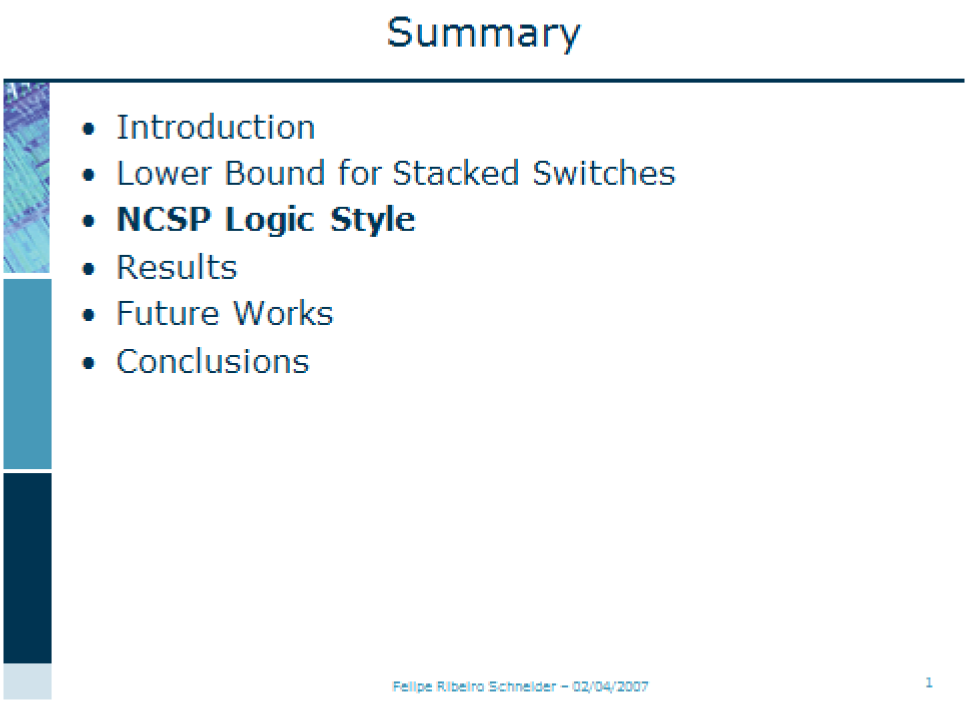
Lower Bound for Stacked Switches

- More cells available on libraries

		Number of Stacked PMOS Transistors			
		1	2	3	4
Number of Stacked NMOS Transistors	1	1	1	1	1
	2	1	4	10	23
	3	1	10	58	285
	4	1	23	285	2798

		Number of Stacked PMOS Transistors			
		1	2	3	4
Number of Stacked NMOS Transistors	1	0	0	0	0
	2	0	+2	+13	+62
	3	0	+13	+498	+2897
	4	0	+62	+2897	+2222

Felipe Ribeiro Schneider – 02/04/2007



Summary

- Introduction
- Lower Bound for Stacked Switches
- **NCSP Logic Style**
- Results
- Future Works
- Conclusions

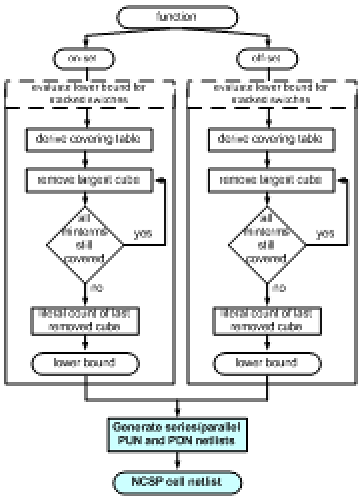
Felipe Ribeiro Schneider – 02/04/2007 1

NCSP logic style

- NCSP (Non-complementary Series-Parallel) has a constructive method for its pull-up and pull-down networks which follows the lower for the number of stacked switches
  - Pull-up and pull-down networks are generated independently
  - Unlike CSP, pull-up and pull-down networks may be topologically non-complementary
  - Like CSP, pull-up and pull-down networks are always logically complementary

NCSP logic style

- NCSP constructive method


function
on-set
off-set
evaluate lower bound for stacked switches
derive covering table
remove largest cube
all minterms still covered
yes
no
literal count of last removed cube
lower bound
evaluate lower bound for stacked switches
derive covering table
remove largest cube
all minterms still covered
yes
no
literal count of last removed cube
lower bound
Generate series/parallel PUN and PDN netlists
NCSP cell netlist

e = 0

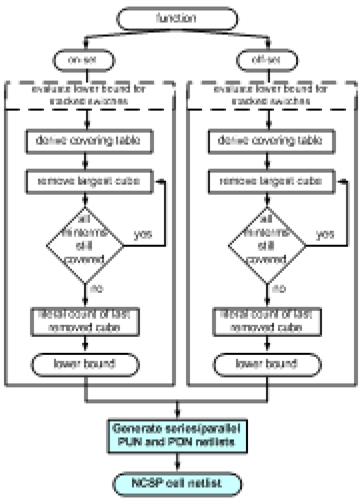
a	b	c	d	out
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

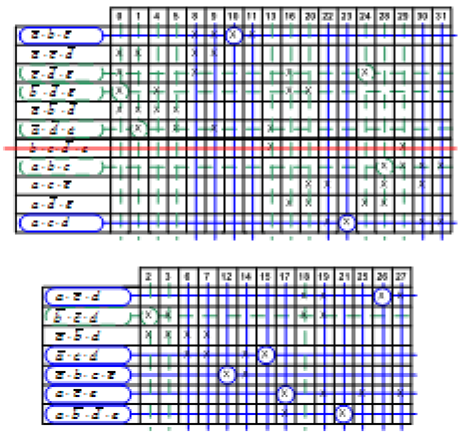
e = 1

a	b	c	d	out
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

NCSP logic style

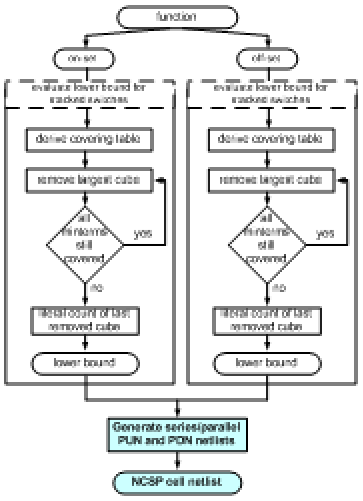
- NCSP constructive method


function
on-set
off-set
evaluate lower bound for stacked switches
derive covering table
remove largest cube
all minterms still covered
yes
no
literal count of last removed cube
lower bound
Generate series/parallel PUN and PDN netlists
NCSP cell netlist



NCSP logic style

- NCSP constructive method


function
on-set
off-set
evaluate lower bound for stacked switches
derive covering table
remove largest cube
all minterms still covered
yes
no
literal count of last removed cube
lower bound
Generate series/parallel PUN and PDN netlists
NCSP cell netlist

$$f_{on-set} = \bar{a}\cdot\bar{b}\cdot\bar{d} + \bar{a}\cdot b\cdot\bar{e} + a\cdot\bar{d}\cdot\bar{e} + a\cdot c\cdot d + \bar{a}\cdot\bar{d}\cdot e + a\cdot b\cdot c$$

$$f_{on-set} = \bar{a}\cdot\left(\bar{d}\cdot\left(e+\bar{b}\right)+b\cdot\bar{e}\right)+a\cdot\left(c\cdot\left(d+b\right)+\bar{d}\cdot\bar{e}\right)$$

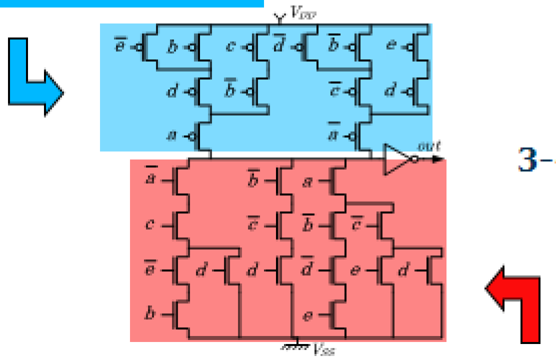
$$f_{off-set} = \bar{a}\cdot b\cdot c\cdot\bar{e} + \bar{a}\cdot c\cdot d + a\cdot\bar{b}\cdot\bar{d}\cdot e + a\cdot\bar{c}\cdot e + a\cdot\bar{c}\cdot d + \bar{b}\cdot\bar{c}\cdot d$$

$$f_{off-set} = \bar{a}\cdot c\cdot\left(b\cdot\bar{e}+d\right)+\bar{b}\cdot\bar{c}\cdot d + a\cdot\left(\bar{b}\cdot\bar{d}\cdot e+\bar{c}\cdot\left(e+d\right)\right)$$

NCSP Logic Style

- NCSP constructive method

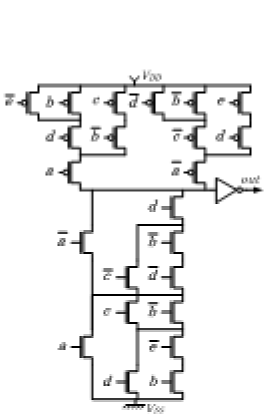
$f_{on-set} = \bar{a}\cdot(\bar{d}\cdot(c+\bar{b})+b\cdot\bar{c})+a\cdot(c\cdot(d+b)+\bar{d}\cdot\bar{e})$



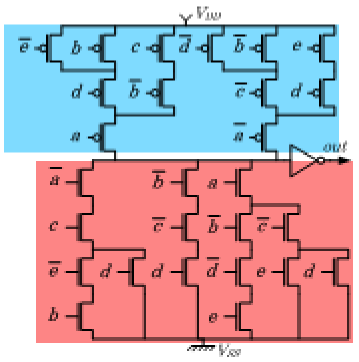
3-4 cell

$f_{off-set} = \bar{a}\cdot c\cdot(b\cdot\bar{e}+d)+\bar{b}\cdot\bar{c}\cdot d+a\cdot(\bar{b}\cdot\bar{d}\cdot e+\bar{c}\cdot(e+d))$

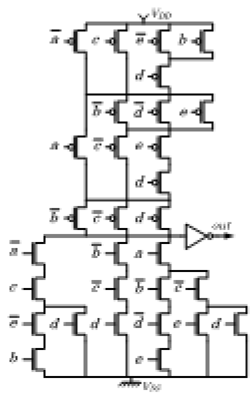
NCSP Logic Style



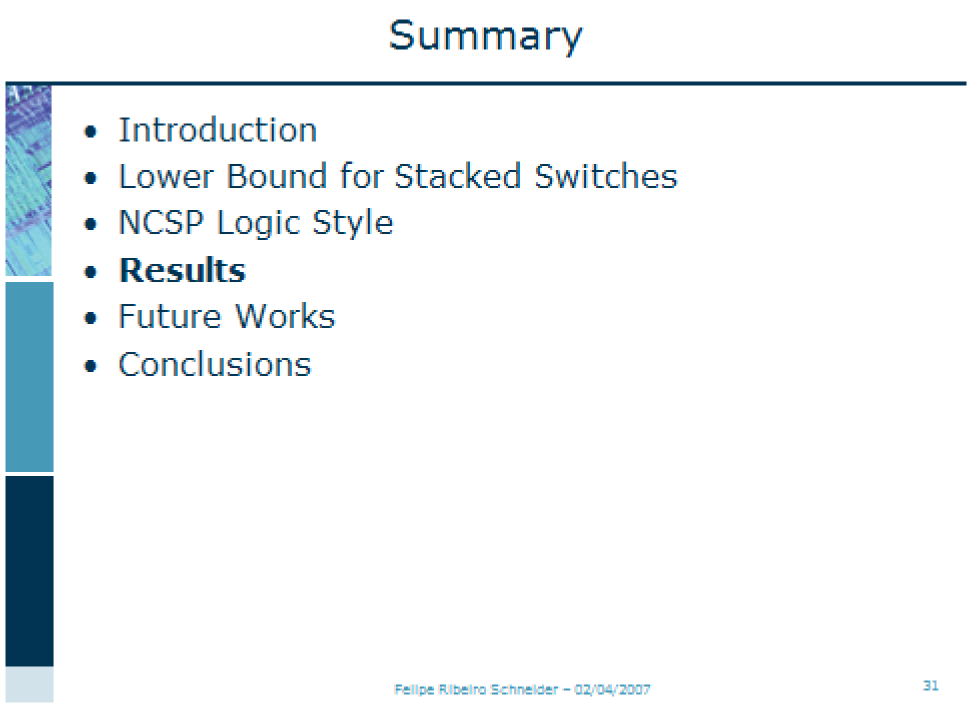
CSP 4-6



NCSP 3-4

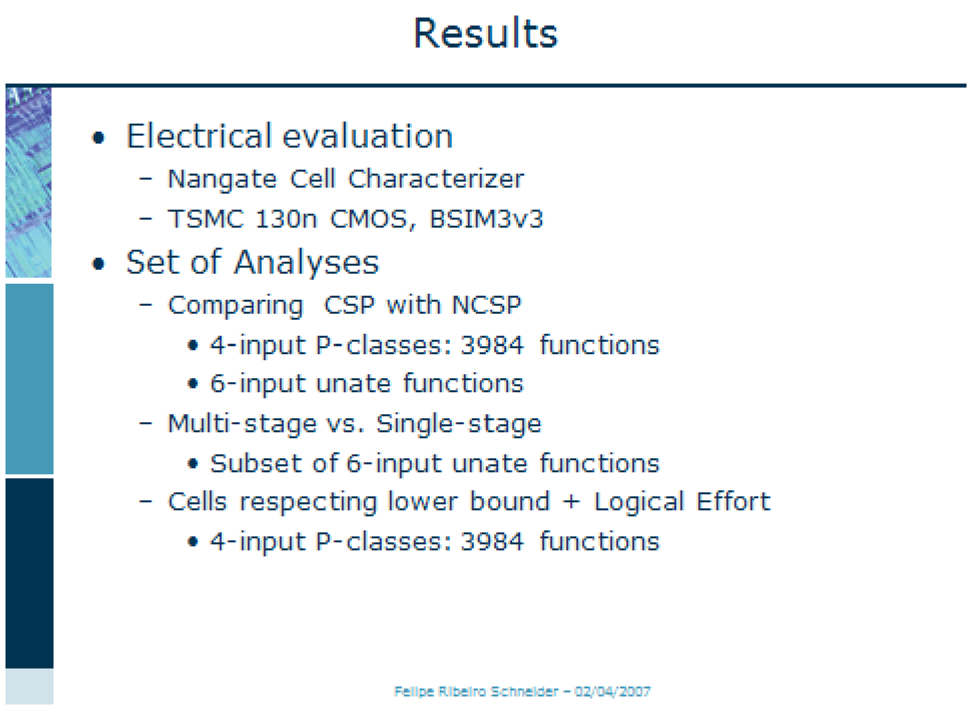


CSP 6-4



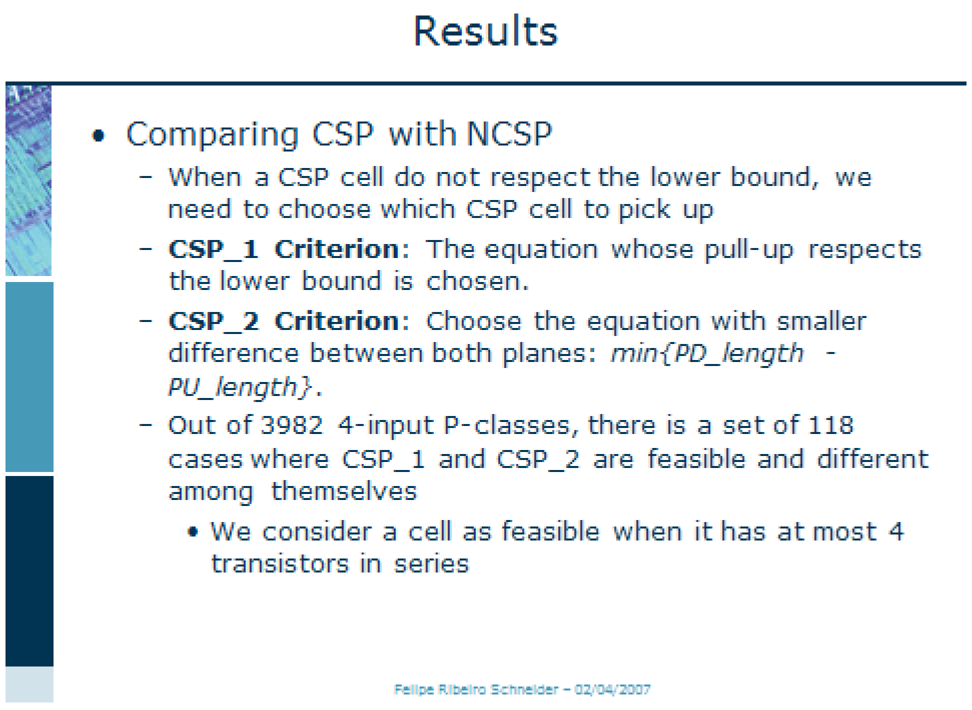
Summary

- Introduction
- Lower Bound for Stacked Switches
- NCSP Logic Style
- **Results**
- Future Works
- Conclusions



Results

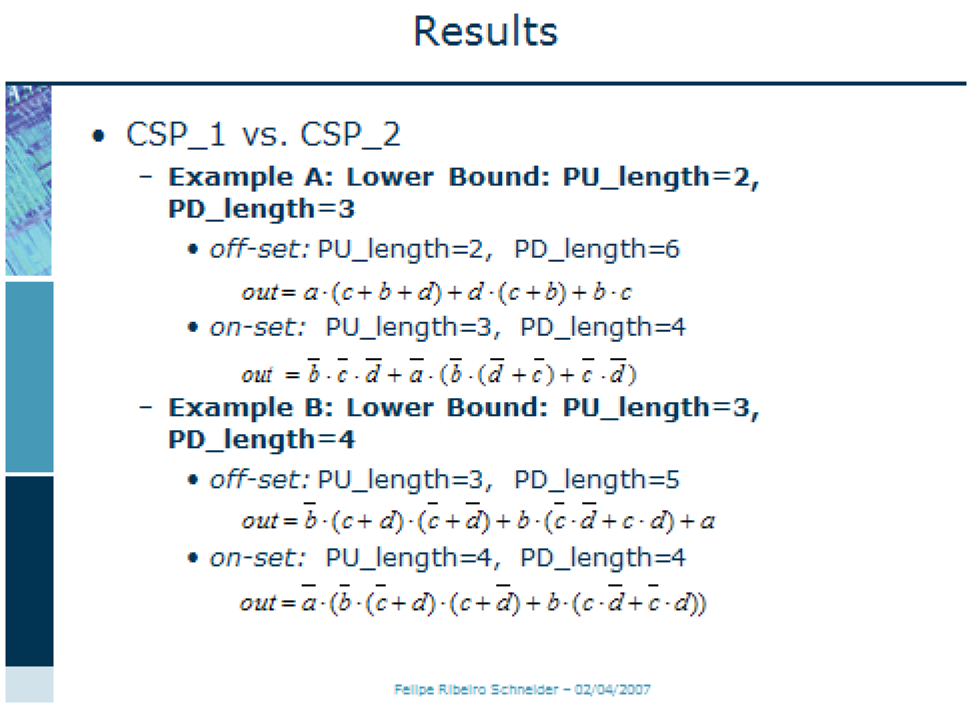
- Electrical evaluation
  - Nangate Cell Characterizer
  - TSMC 130n CMOS, BSIM3v3
- Set of Analyses
  - Comparing CSP with NCSP
    - 4-input P-classes: 3984 functions
    - 6-input unate functions
  - Multi-stage vs. Single-stage
    - Subset of 6-input unate functions
  - Cells respecting lower bound + Logical Effort
    - 4-input P-classes: 3984 functions



Results

- Comparing CSP with NCSP
  - When a CSP cell do not respect the lower bound, we need to choose which CSP cell to pick up
  - **CSP_1 Criterion**: The equation whose pull-up respects the lower bound is chosen.
  - **CSP_2 Criterion**: Choose the equation with smaller difference between both planes: *min{PD_length - PU_length}*.
  - Out of 3982 4-input P-classes, there is a set of 118 cases where CSP_1 and CSP_2 are feasible and different among themselves
    - We consider a cell as feasible when it has at most 4 transistors in series

Felipe Ribeiro Schneider – 02/04/2007



Results

- CSP_1 vs. CSP_2
  - **Example A: Lower Bound: PU_length=2, PD_length=3**
    - *off-set:* PU_length=2, PD_length=6

      $out = a \cdot (c + b + d) + d \cdot (c + b) + b \cdot c$
    - *on-set:* PU_length=3, PD_length=4

      $out = \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot (\overline{b} \cdot (\overline{d} + \overline{c}) + \overline{c} \cdot \overline{d})$
  - **Example B: Lower Bound: PU_length=3, PD_length=4**
    - *off-set:* PU_length=3, PD_length=5

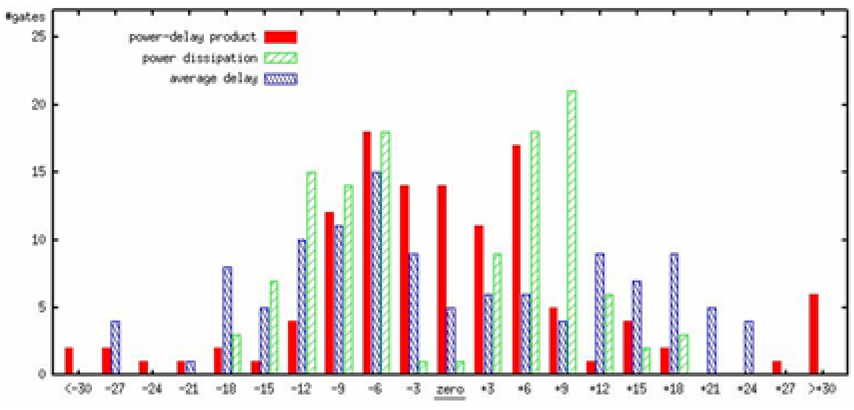
      $out = \overline{b} \cdot (c + d) \cdot (\overline{c} + \overline{d}) + b \cdot (\overline{c} \cdot \overline{d} + c \cdot d) + a$
    - *on-set:* PU_length=4, PD_length=4

      $out = \overline{a} \cdot (\overline{b} \cdot (\overline{c} + d) \cdot (c + \overline{d}) + b \cdot (c \cdot \overline{d} + \overline{c} \cdot d))$

Felipe Ribeiro Schneider – 02/04/2007

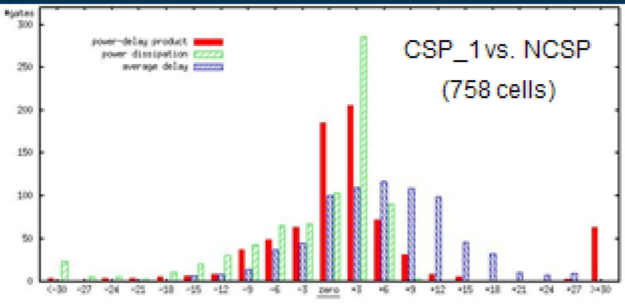
Results

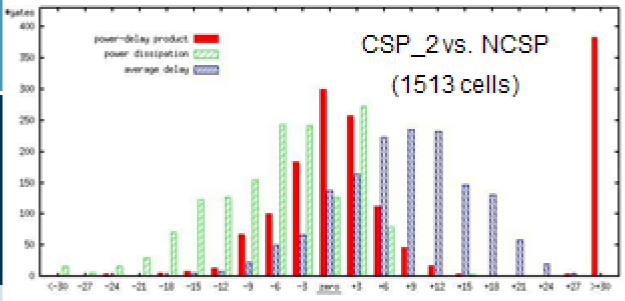
- CSP_1 vs. CSP_2
  - Input slope: 200 ps
  - Load capacitance: 2 fF
- X-axis represents percentage of improvement for CSP_1
- Y-axis represents number of logic gates

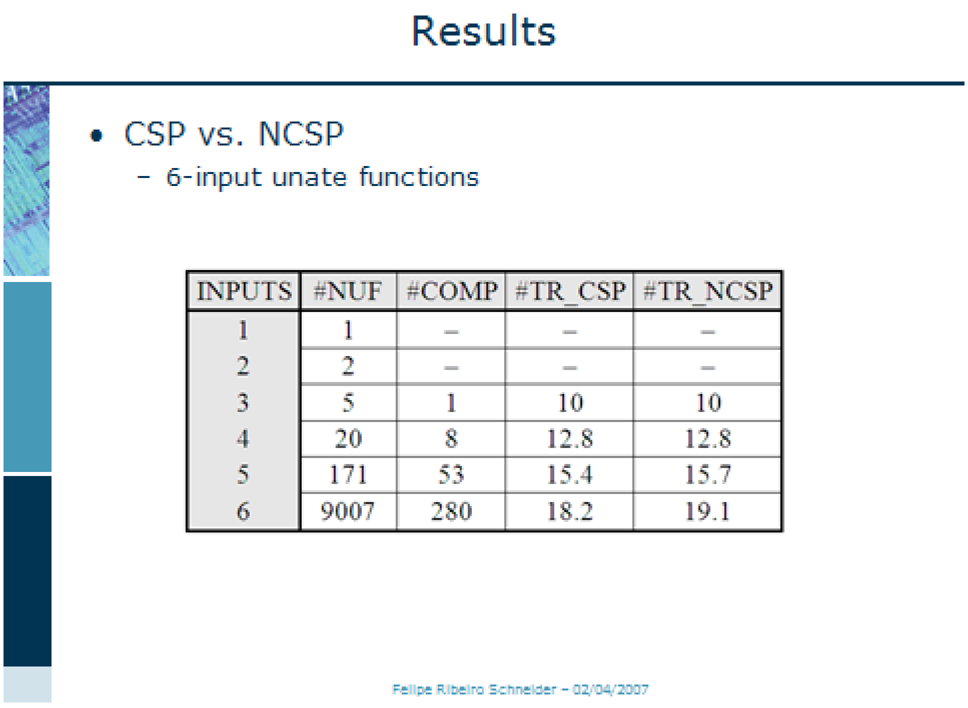


Results

- CSP vs. NCSP
  - Input slope: 200 ps
  - Load capacitance: 2 fF
  - Feasible cells





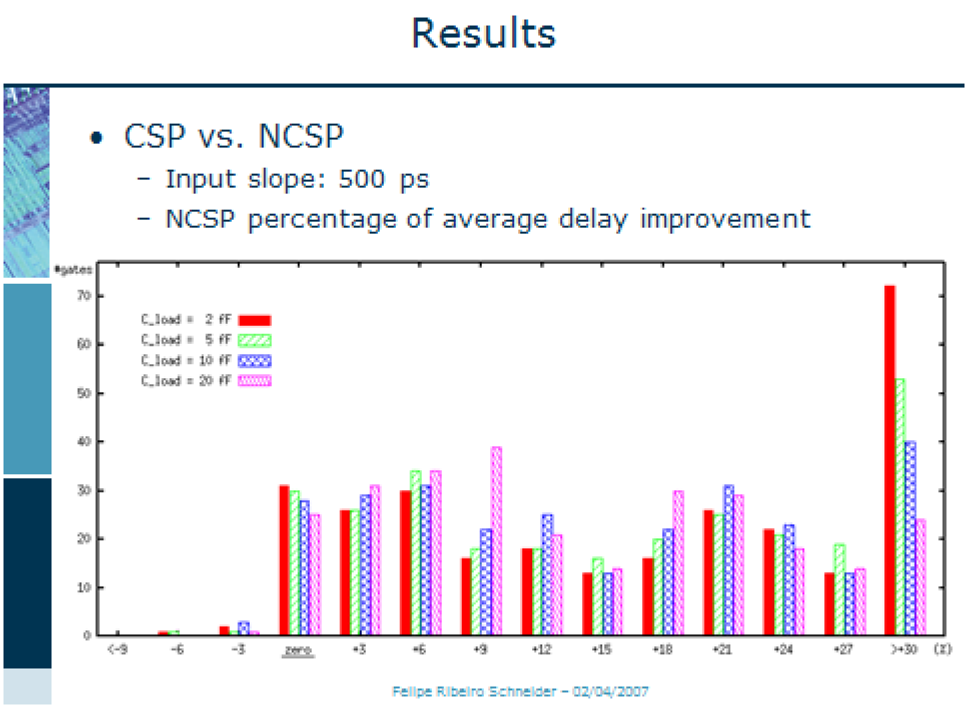


Results

- CSP vs. NCSP
  - 6-input unate functions

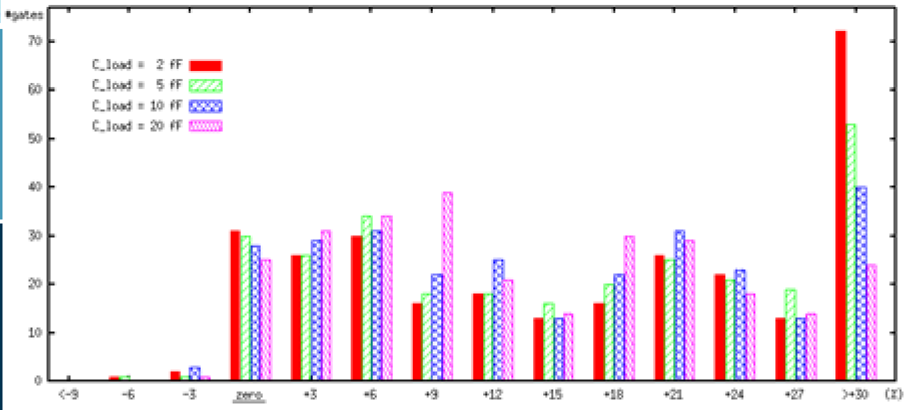
INPUTS	#NUF	#COMP	#TR_CSP	#TR_NCSP
1	1	–	–	–
2	2	–	–	–
3	5	1	10	10
4	20	8	12.8	12.8
5	171	53	15.4	15.7
6	9007	280	18.2	19.1

Felipe Ribeiro Schneider – 02/04/2007

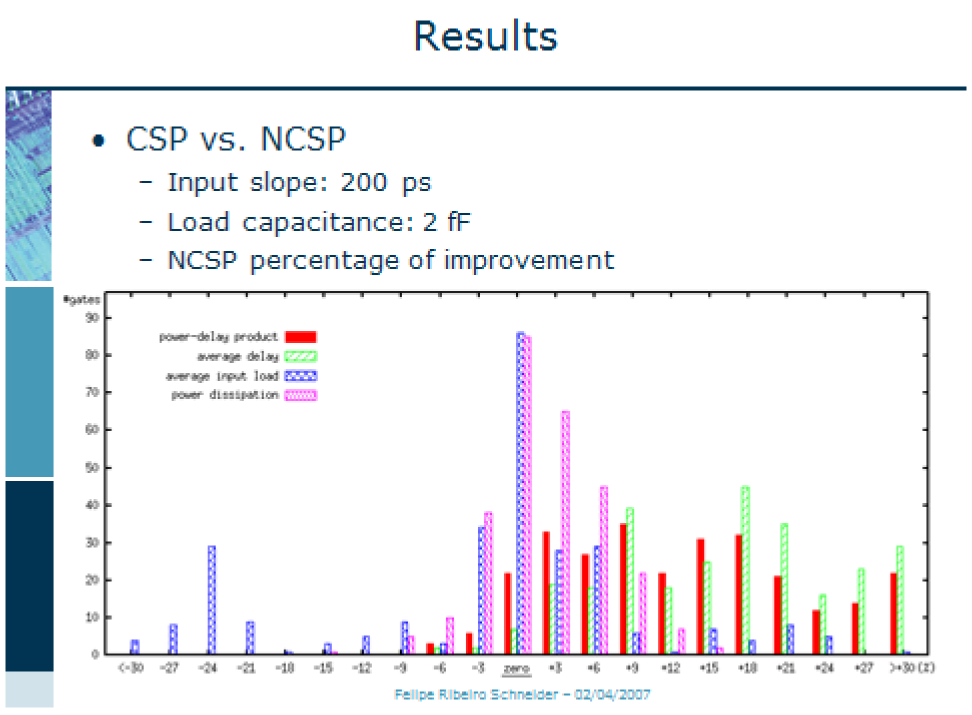


Results

- CSP vs. NCSP
  - Input slope: 500 ps
  - NCSP percentage of average delay improvement

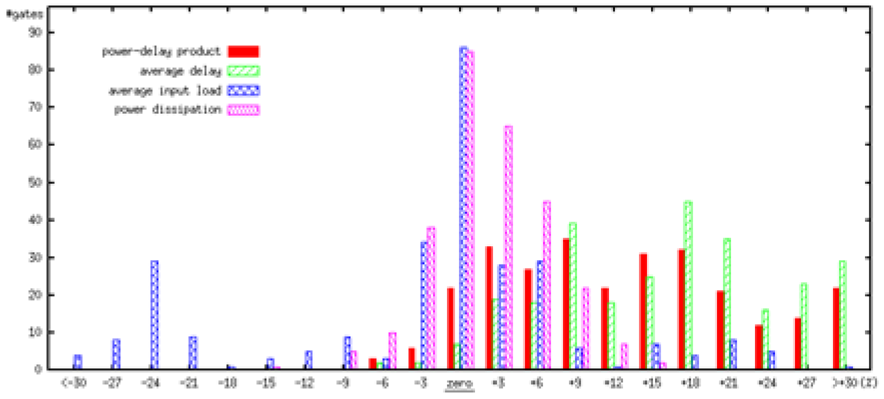


Felipe Ribeiro Schneider – 02/04/2007

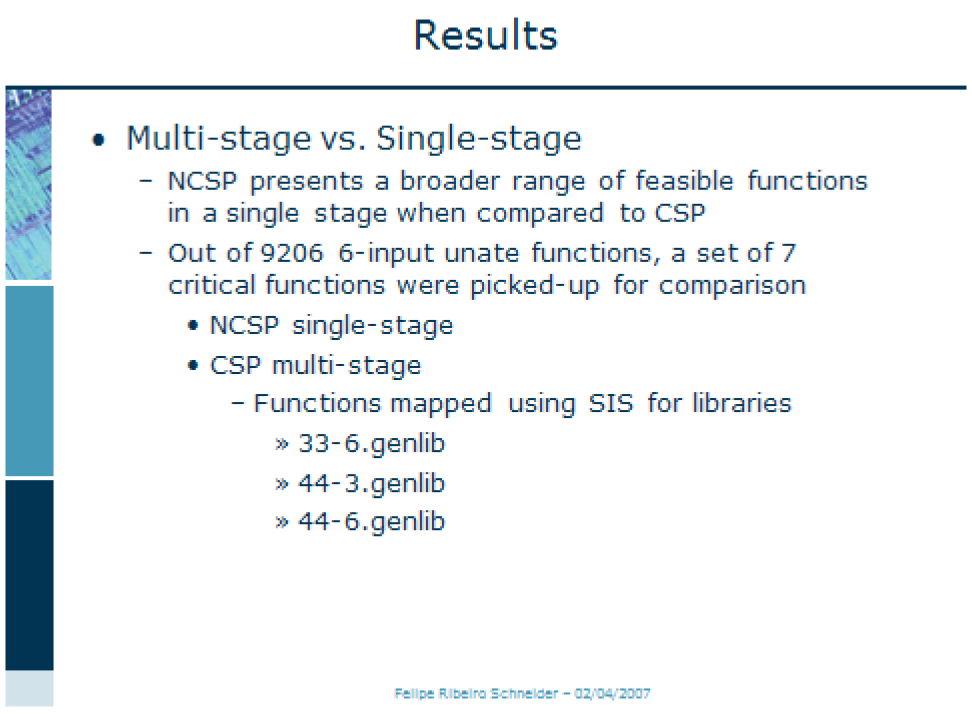


Results

- CSP vs. NCSP
  - Input slope: 200 ps
  - Load capacitance: 2 fF
  - NCSP percentage of improvement



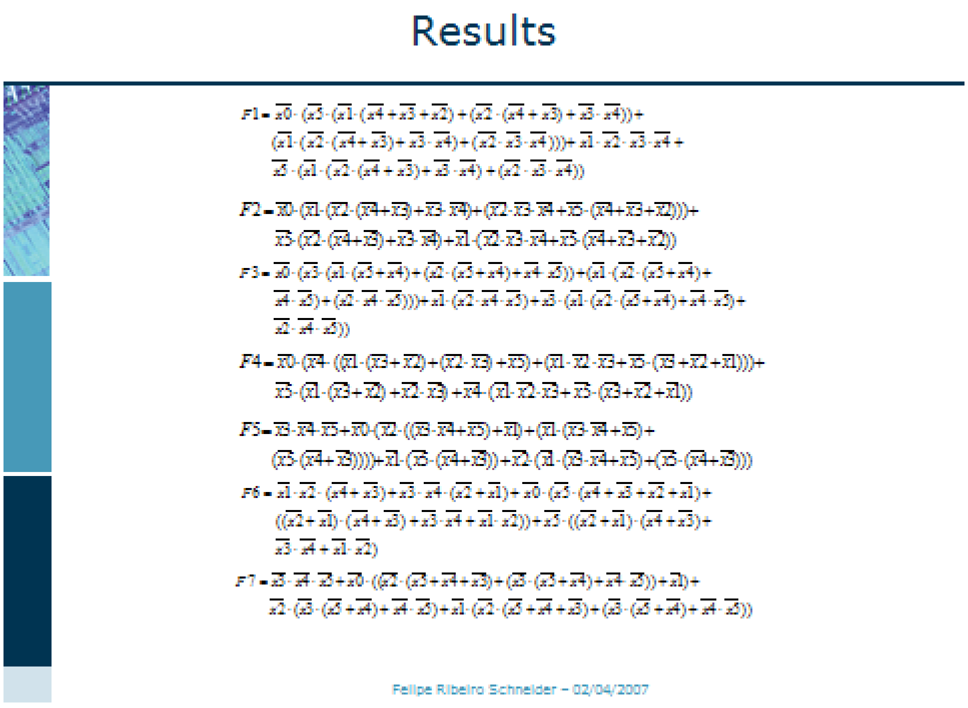
Felipe Ribeiro Schneider – 02/04/2007



Results

- Multi-stage vs. Single-stage
  - NCSP presents a broader range of feasible functions in a single stage when compared to CSP
  - Out of 9206 6-input unate functions, a set of 7 critical functions were picked-up for comparison
    - NCSP single-stage
    - CSP multi-stage
      - Functions mapped using SIS for libraries
        - » 33-6.genlib
        - » 44-3.genlib
        - » 44-6.genlib

Felipe Ribeiro Schneider – 02/04/2007



Results

$$\begin{aligned}
F1 = {} & \overline{x0}\cdot(\overline{x5}\cdot(\overline{x1}\cdot(\overline{x4}+\overline{x3}+\overline{x2})+(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4}))+\\
& (\overline{x1}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+(\overline{x2}\cdot\overline{x3}\cdot\overline{x4})))+\overline{x1}\cdot\overline{x2}\cdot\overline{x3}\cdot\overline{x4}+\\
& \overline{x5}\cdot(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+(\overline{x2}\cdot\overline{x3}\cdot\overline{x4}))
\end{aligned}$$

$$\begin{aligned}
F2 = {} & \overline{x0}\cdot(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+(\overline{x2}\cdot\overline{x3}\cdot\overline{x4}+\overline{x5}\cdot(\overline{x4}+\overline{x3}+\overline{x2})))+\\
& \overline{x5}\cdot(\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4})+\overline{x1}\cdot(\overline{x2}\cdot\overline{x3}\cdot\overline{x4}+\overline{x5}\cdot(\overline{x4}+\overline{x3}+\overline{x2}))
\end{aligned}$$

$$\begin{aligned}
F3 = {} & \overline{x0}\cdot(\overline{x3}\cdot(\overline{x1}\cdot(\overline{x5}+\overline{x4})+(\overline{x2}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5}))+(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x5}+\overline{x4})+\\
& \overline{x4}\cdot\overline{x5})+(\overline{x2}\cdot\overline{x4}\cdot\overline{x5})))+\overline{x1}\cdot(\overline{x2}\cdot\overline{x4}\cdot\overline{x5})+\overline{x3}\cdot(\overline{x1}\cdot(\overline{x2}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5})+\\
& \overline{x2}\cdot\overline{x4}\cdot\overline{x5}))
\end{aligned}$$

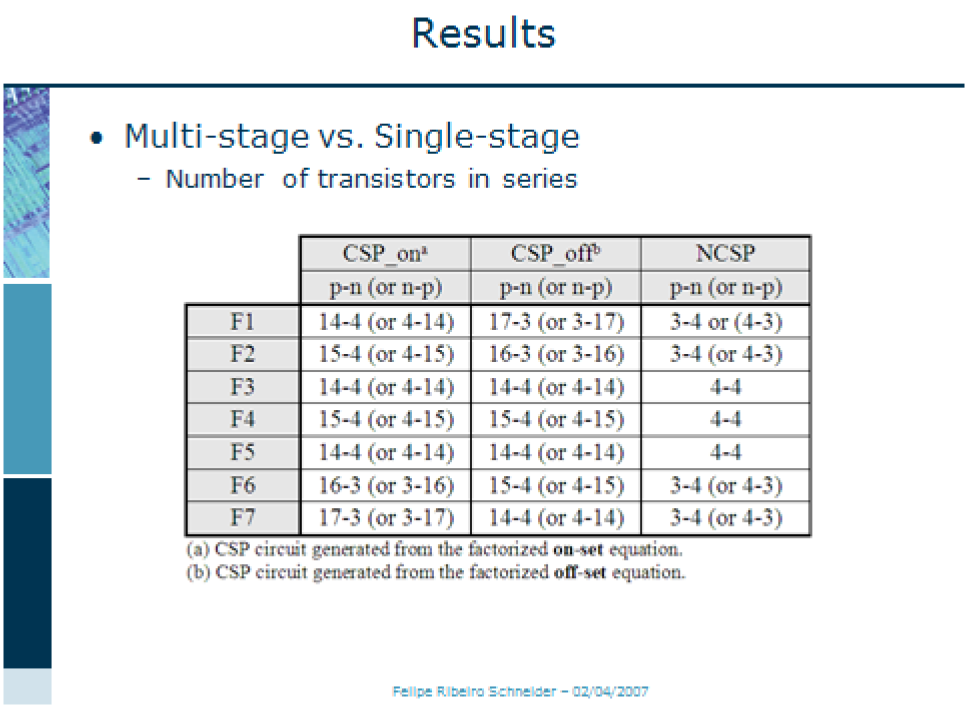
$$\begin{aligned}
F4 = {} & \overline{x0}\cdot(\overline{x4}\cdot((\overline{x1}\cdot(\overline{x3}+\overline{x2})+(\overline{x2}\cdot\overline{x3})+\overline{x5})+(\overline{x1}\cdot\overline{x2}\cdot\overline{x3}+\overline{x5}\cdot(\overline{x3}+\overline{x2}+\overline{x1})))+\\
& \overline{x5}\cdot(\overline{x1}\cdot(\overline{x3}+\overline{x2})+\overline{x2}\cdot\overline{x3})+\overline{x4}\cdot(\overline{x1}\cdot\overline{x2}\cdot\overline{x3}+\overline{x5}\cdot(\overline{x3}+\overline{x2}+\overline{x1}))
\end{aligned}$$

$$\begin{aligned}
F5 = {} & \overline{x3}\cdot\overline{x4}\cdot\overline{x5}+\overline{x0}\cdot(\overline{x2}\cdot((\overline{x3}\cdot\overline{x4}+\overline{x5})+\overline{x1})+(\overline{x1}\cdot(\overline{x3}\cdot\overline{x4}+\overline{x5})+\\
& (\overline{x5}\cdot(\overline{x4}+\overline{x3}))))+\overline{x1}\cdot(\overline{x5}\cdot(\overline{x4}+\overline{x3}))+\overline{x2}\cdot(\overline{x1}\cdot(\overline{x3}\cdot\overline{x4}+\overline{x5})+(\overline{x5}\cdot(\overline{x4}+\overline{x3})))
\end{aligned}$$

$$\begin{aligned}
F6 = {} & \overline{x1}\cdot\overline{x2}\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4}\cdot(\overline{x2}+\overline{x1})+\overline{x0}\cdot(\overline{x5}\cdot(\overline{x4}+\overline{x3}+\overline{x2}+\overline{x1})+\\
& ((\overline{x2}+\overline{x1})\cdot(\overline{x4}+\overline{x3})+\overline{x3}\cdot\overline{x4}+\overline{x1}\cdot\overline{x2}))+\overline{x5}\cdot((\overline{x2}+\overline{x1})\cdot(\overline{x4}+\overline{x3})+\\
& \overline{x3}\cdot\overline{x4}+\overline{x1}\cdot\overline{x2})
\end{aligned}$$

$$\begin{aligned}
F7 = {} & \overline{x5}\cdot\overline{x4}\cdot\overline{x3}+\overline{x0}\cdot((\overline{x2}\cdot(\overline{x5}+\overline{x4}+\overline{x3})+(\overline{x3}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5}))+\overline{x1})+\\
& \overline{x2}\cdot(\overline{x3}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5})+\overline{x1}\cdot(\overline{x2}\cdot(\overline{x5}+\overline{x4}+\overline{x3})+(\overline{x3}\cdot(\overline{x5}+\overline{x4})+\overline{x4}\cdot\overline{x5}))
\end{aligned}$$

Felipe Ribeiro Schneider – 02/04/2007



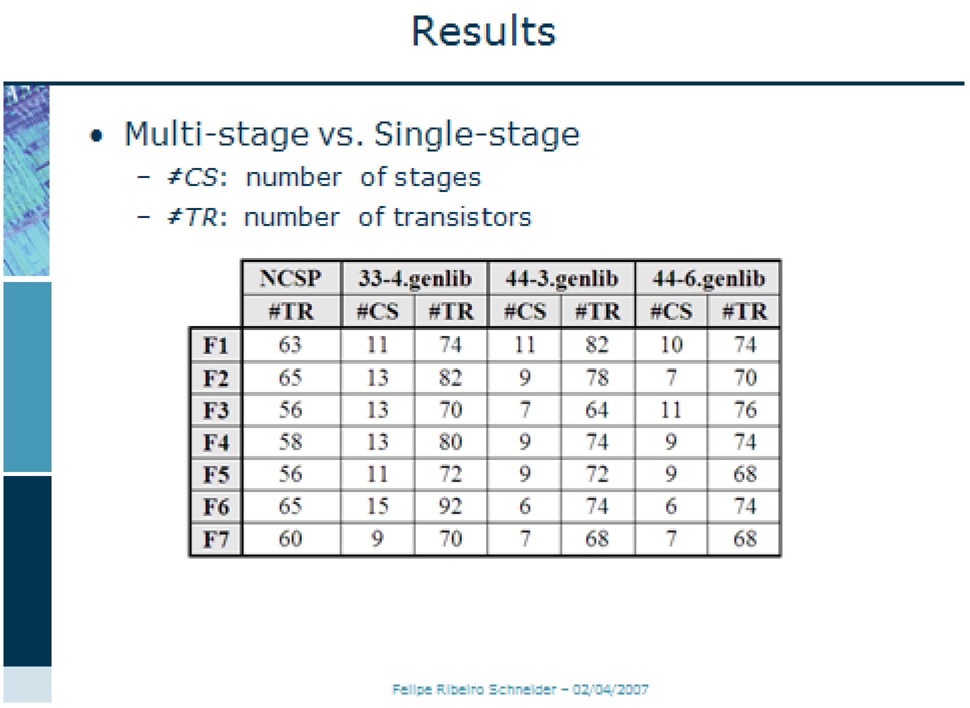
Results

- Multi-stage vs. Single-stage
  - Number of transistors in series

	CSP_on[a]	CSP_off[b]	NCSP
	p-n (or n-p)	p-n (or n-p)	p-n (or n-p)
F1	14-4 (or 4-14)	17-3 (or 3-17)	3-4 or (4-3)
F2	15-4 (or 4-15)	16-3 (or 3-16)	3-4 (or 4-3)
F3	14-4 (or 4-14)	14-4 (or 4-14)	4-4
F4	15-4 (or 4-15)	15-4 (or 4-15)	4-4
F5	14-4 (or 4-14)	14-4 (or 4-14)	4-4
F6	16-3 (or 3-16)	15-4 (or 4-15)	3-4 (or 4-3)
F7	17-3 (or 3-17)	14-4 (or 4-14)	3-4 (or 4-3)

(a) CSP circuit generated from the factorized **on-set** equation.
(b) CSP circuit generated from the factorized **off-set** equation.

Felipe Ribeiro Schneider – 02/04/2007

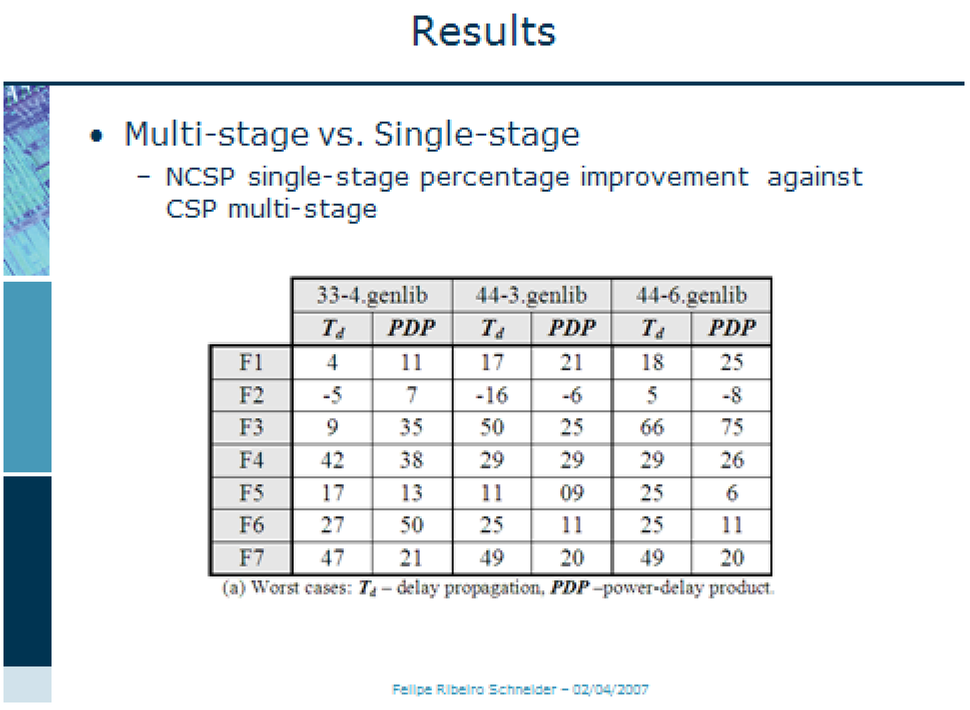


Results

- Multi-stage vs. Single-stage
  - ≠*CS*: number of stages
  - ≠*TR*: number of transistors

	NCSP	33-4.genlib		44-3.genlib		44-6.genlib	
	#TR	**#CS**	**#TR**	**#CS**	**#TR**	**#CS**	**#TR**
F1	63	11	74	11	82	10	74
F2	65	13	82	9	78	7	70
F3	56	13	70	7	64	11	76
F4	58	13	80	9	74	9	74
F5	56	11	72	9	72	9	68
F6	65	15	92	6	74	6	74
F7	60	9	70	7	68	7	68

Felipe Ribeiro Schneider – 02/04/2007



Results

- Multi-stage vs. Single-stage
  - NCSP single-stage percentage improvement against CSP multi-stage

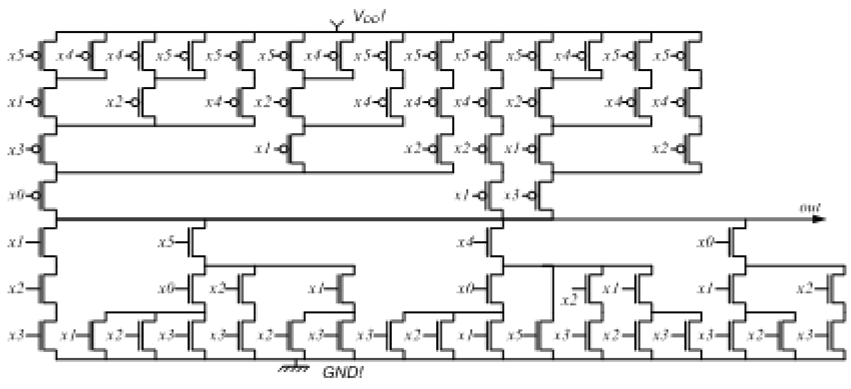
	33-4.genlib		44-3.genlib		44-6.genlib	
	T_d	***PDP***	T_d	***PDP***	T_d	***PDP***
F1	4	11	17	21	18	25
F2	-5	7	-16	-6	5	-8
F3	9	35	50	25	66	75
F4	42	38	29	29	29	26
F5	17	13	11	09	25	6
F6	27	50	25	11	25	11
F7	47	21	49	20	49	20

(a) Worst cases: T_d – delay propagation, ***PDP*** –power-delay product.

Felipe Ribeiro Schneider – 02/04/2007

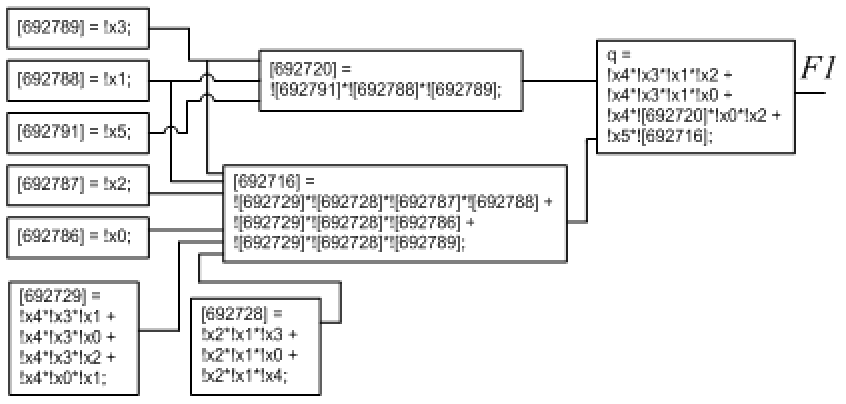
Results

- NCSP single-stage circuit for function F1

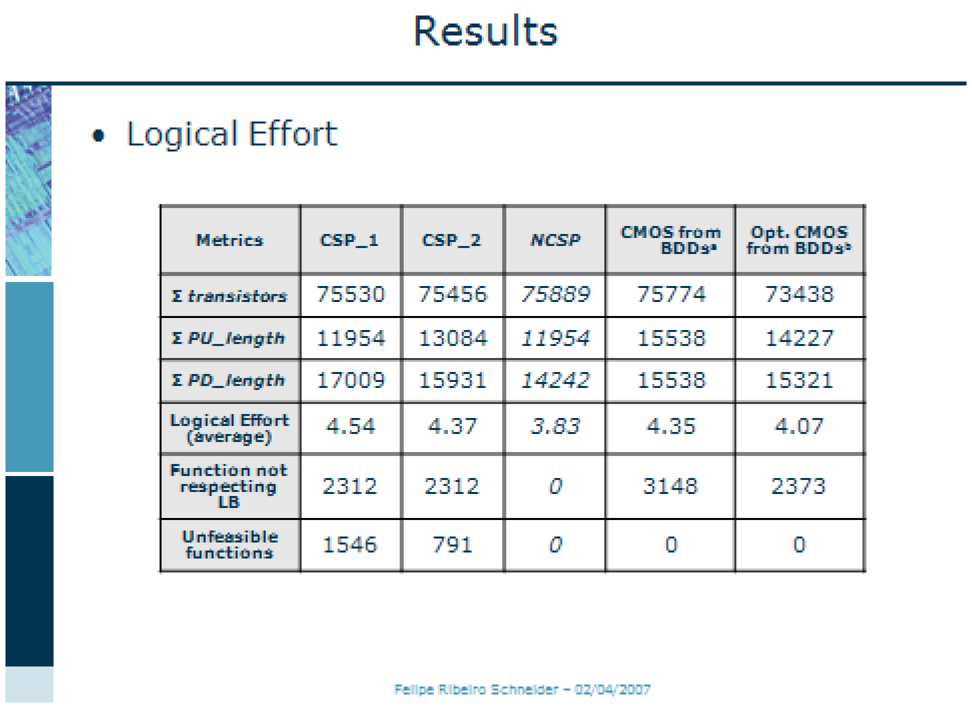


Results

- CSP multi-stage circuit for function F1
  - Mapped with 44-6.genlib library



```
[692789] = !x3;
[692788] = !x1;
[692791] = !x5;
[692787] = !x2;
[692786] = !x0;
[692729] =
!x4*!x3*!x1 +
!x4*!x3*!x0 +
!x4*!x3*!x2 +
!x4*!x0*!x1;
[692720] =
![692791]*![692788]*![692789];
[692716] =
![692729]*![692728]*![692787]*![692788] +
![692729]*![692728]*![692786] +
![692729]*![692728]*![692789];
[692728] =
!x2*!x1*!x3 +
!x2*!x1*!x0 +
!x2*!x1*!x4;
q =
!x4*!x3*!x1*!x2 +
!x4*!x3*!x1*!x0 +
!x4*![692720]*!x0*!x2 +
!x5*![692716];
```

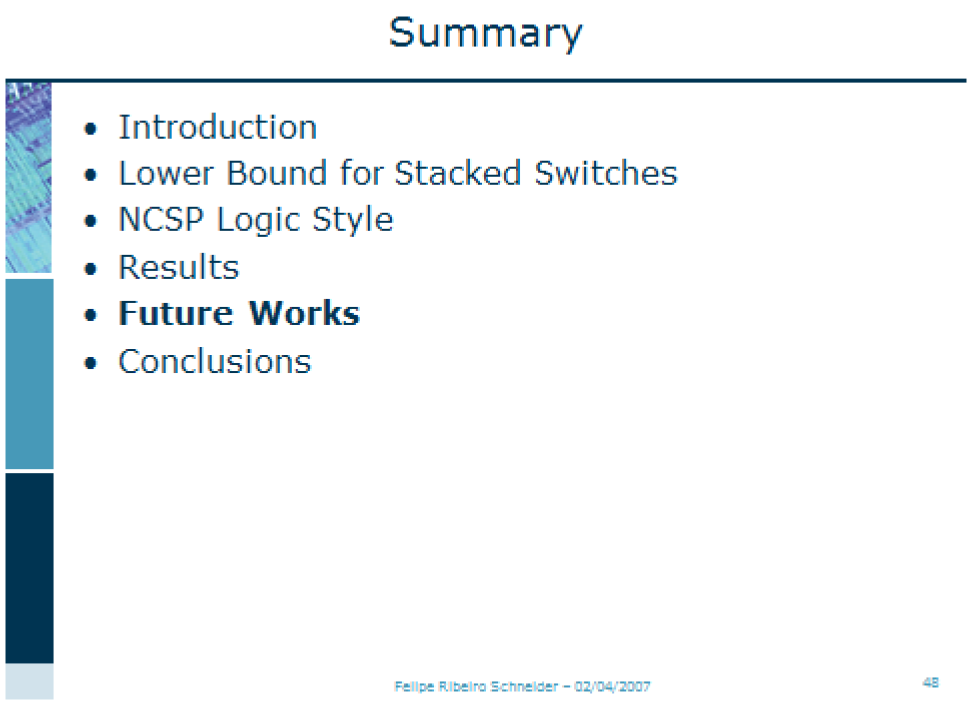


Results

- Logical Effort

Metrics	CSP_1	CSP_2	*NCSP*	CMOS from BDDs[a]	Opt. CMOS from BDDs[b]
Σ transistors	75530	75456	*75889*	75774	73438
Σ PU_length	11954	13084	*11954*	15538	14227
Σ PD_length	17009	15931	*14242*	15538	15321
Logical Effort (average)	4.54	4.37	*3.83*	4.35	4.07
Function not respecting LB	2312	2312	*0*	3148	2373
Unfeasible functions	1546	791	*0*	0	0

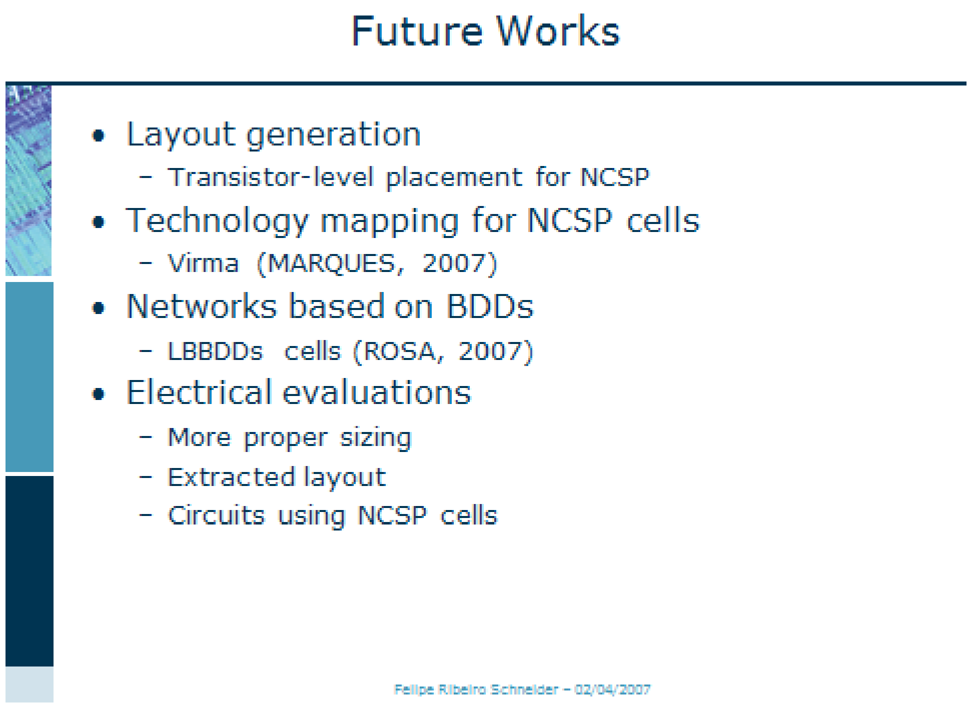
Felipe Ribeiro Schneider – 02/04/2007



Summary

- Introduction
- Lower Bound for Stacked Switches
- NCSP Logic Style
- Results
- **Future Works**
- Conclusions

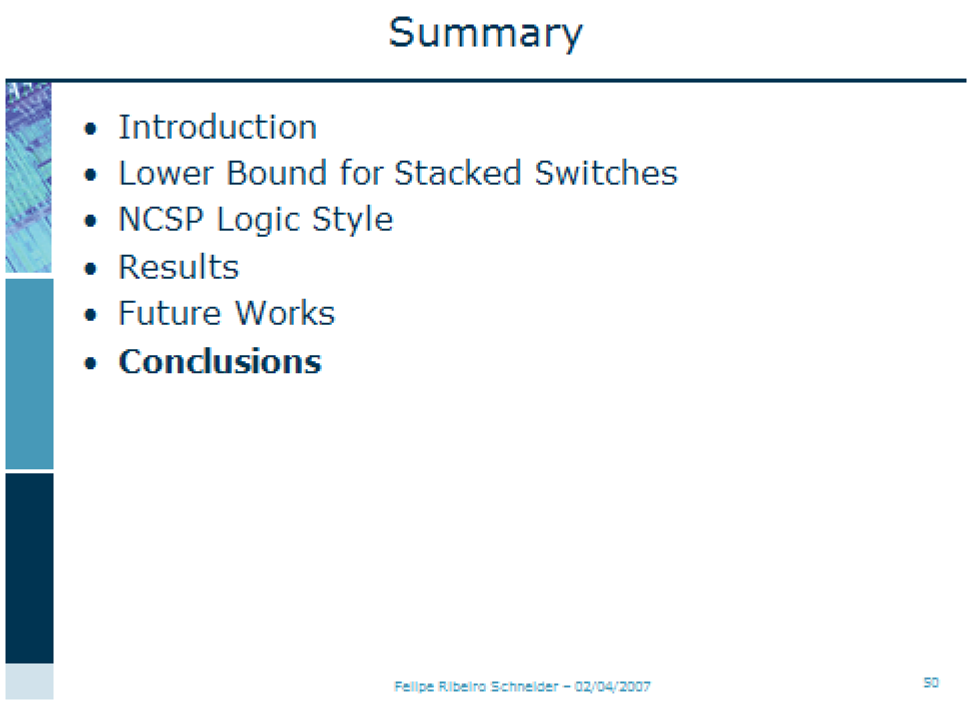
Felipe Ribeiro Schneider – 02/04/2007 48



Future Works

- Layout generation
  - Transistor-level placement for NCSP
- Technology mapping for NCSP cells
  - Virma (MARQUES, 2007)
- Networks based on BDDs
  - LBBDDs cells (ROSA, 2007)
- Electrical evaluations
  - More proper sizing
  - Extracted layout
  - Circuits using NCSP cells

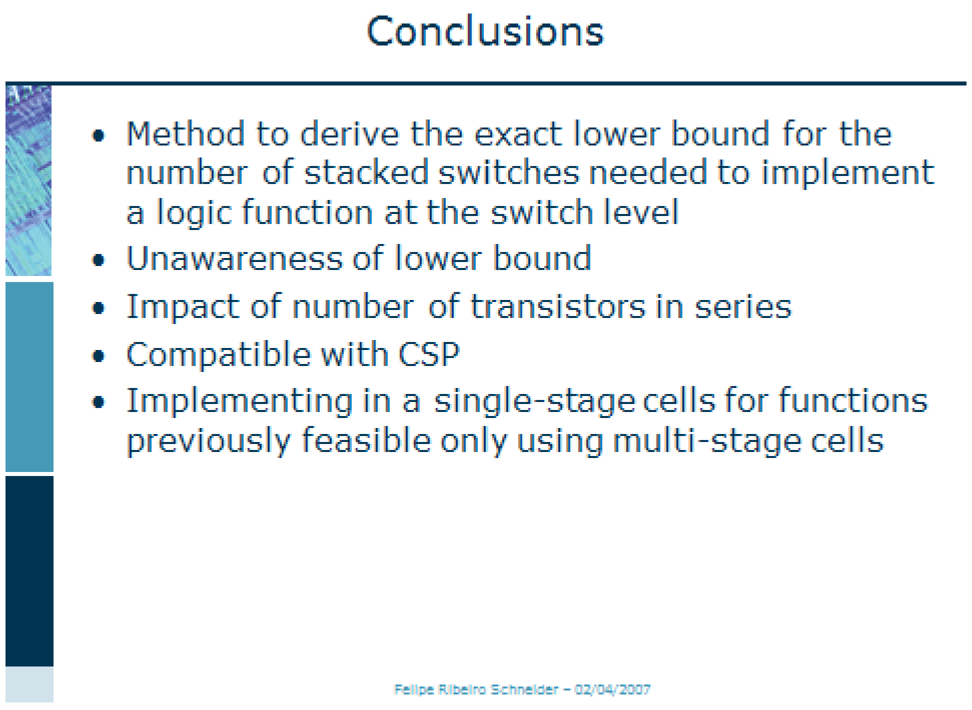
Felipe Ribeiro Schneider – 02/04/2007



Summary

- Introduction
- Lower Bound for Stacked Switches
- NCSP Logic Style
- Results
- Future Works
- **Conclusions**

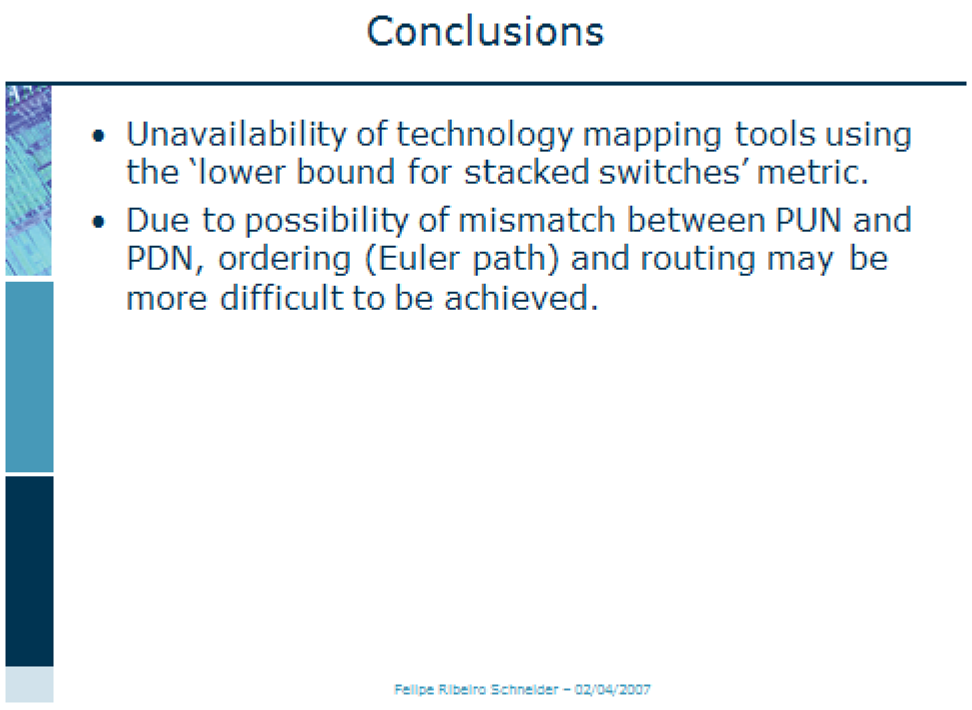
Felipe Ribeiro Schneider – 02/04/2007 50



Conclusions

- Method to derive the exact lower bound for the number of stacked switches needed to implement a logic function at the switch level
- Unawareness of lower bound
- Impact of number of transistors in series
- Compatible with CSP
- Implementing in a single-stage cells for functions previously feasible only using multi-stage cells

Felipe Ribeiro Schneider – 02/04/2007



Conclusions

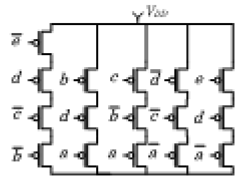
- Unavailability of technology mapping tools using the 'lower bound for stacked switches' metric.
- Due to possibility of mismatch between PUN and PDN, ordering (Euler path) and routing may be more difficult to be achieved.

Felipe Ribeiro Schneider – 02/04/2007

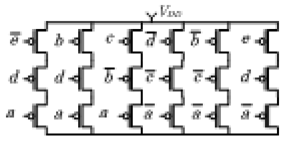
Conclusions

- NCSP with smaller resistance paths, but sometimes with higher capacitance

$$f_{on-set} = \bar{a}\cdot\bar{b}\cdot\bar{d}+\bar{a}\cdot b\cdot\bar{c}+a\cdot\bar{d}\cdot\bar{e}+a\cdot c\cdot d+\boxed{b\cdot c\cdot\bar{d}\cdot e}$$



$$f_{on-set} = \bar{a}\cdot\bar{b}\cdot\bar{d}+\bar{a}\cdot b\cdot\bar{c}+a\cdot\bar{d}\cdot\bar{e}+a\cdot c\cdot d+\boxed{\bar{a}\cdot\bar{d}\cdot e}+\boxed{a\cdot b\cdot c}$$



Work-related publications

- SCHNEIDER, F.R.; RIBAS, R.P.; SAPATNEKAR, S.S.; REIS, A.I. Exact lower bound for the number of switches in series to implement a combinational logic cell. **ICCD**, 2005, 02-05 Oct. 2005 Page(s):357 – 362.
- SCHNEIDER, F.R.; RIBAS, R.P.; REIS, A.I. Fast CMOS Logic Style Using Minimum Transistor Stack for Pull-up and Pull-down Networks. **IWLS**, 2006a.
- SCHNEIDER, F.R.; RIBAS, R.P.; REIS, A.I. CMOS Logic Gates Based on the Minimum Theoretical Number of Transistor in Series. **NORCHIP**, 2006b.
- ROSA, L.S, SCHNEIDER, F.R.; RIBAS, R.P., REIS, A.I. Analysis of Transistor Networks Generation. **IBERCHIP**, 2007.

APPENDIX B: CONSTRUINDO REDES DE TRANSISTORES DE ACORDO COM O NÚMERO MÍNIMO DE CHAVES EM SÉRIE

Resumo da Dissertação em Português

Introdução

Em portas lógicas CMOS, tanto o atraso de propagação como a curva de saída estão fortemente ligados ao número de dispositivos PMOS e NMOS conectados em série nas redes de carga e descarga, respectivamente. Essa correlação está diretamente relacionada com o Esforço Lógico da célula (SUTHERLAND, 1999; KABBANI, 2005; WESTE, 2006; ROSA 2007). O estilo lógico *standard CMOS* é, em geral, otimizado para um dos planos, apresentando então o arranjo complementar no plano oposto. Conseqüentemente, o número mínimo de transistores em série não é necessariamente alcançado.

Neste trabalho, apresenta-se um método para encontrar o menor número de chaves (transistores) em série necessários para se implementar portas lógicas complexas CMOS. Um novo estilo lógico CMOS denominado NCSP, derivado de tal método, é então proposto e comparado ao estilo CSP (*standard CMOS*). Finalmente, uma análise acerca da performance, área e potência são então apresentados.

Limite inferior para o número de chaves em série

O limite inferior para o número de chaves em série (aqui simplesmente referido como *lower bound*) proposto neste trabalho é baseado no número de literais do menor cubo em uma cobertura prima e não-redundante (conjunto de implicantes primos que cobrem uma função e onde cada implicante primo no conjunto não é coberto por nenhum outro no mesmo conjunto). O problema é que se uma função pode ter coberturas primas e não-redundantes com um número de literais diferentes no menor cubo, então o *lower bound* não seria univocamente definido. Este trabalho apresenta uma conjunto de Teoremas e Provas (aqui referido apenas como Teoria do *lower bound*) que garantem que o tamanho do menor cubo em coberturas primas e não-redundantes distintas de uma mesma função lógica é univocamente definido.

Para melhor entender o impacto da Teoria do *lower bound*, observe o seguinte exemplo.

Exemplo 1: Considere a função *f* dada pela equação $f = a \cdot b + b \cdot c + a \cdot c \cdot d$. As coberturas mínimas para o *on-set* e o *off-set* desta função são:

$$on-set = a\cdot b + b\cdot c + a\cdot c\cdot d$$

$$off\ -set = \overline{a}\cdot\overline{b} + \overline{a}\cdot\overline{c} + \overline{b}\cdot\overline{d} + \overline{b}\cdot\overline{c}$$

O menor cubo no *on-set* é $a\cdot c\cdot d$, então o *lower bound* para o número de transistors em série no plano de *pull-up* é três. Os cubos no *off-set* são todos do mesmo tamanho, e o *lower bound* para o número de transistors em série no plano de *pull-down* é dois. Dessa forma, a célula correspondente a função *f* é uma célula 3-2 (3 transistores em série em um plano e 2 no outro), quando mapeado com o método construtivo a ser detalhado a seguir. Uma célula 2-3 também poderia ser derivada para a função *f* com a inversão da função.

Uma tabela conhecida no meio acadêmico para descrever o número de funções lógicas com um determinado número de transistores em série é apresentado na Tabela B.1. A construção da Tabela B.1 (DETJENS, 1987) considera apenas funções *negative-unate*. A computação do número de transistores em série é feita considerando associações em série/paralelo.

Tabela B.1: Número de funções *negative-unate* com um determinado número de transistors em série (DETJENS, 1987).

		#Transistores PMOS em Série			
		1	**2**	**3**	**4**
#Transistores NMOS em Série	**1**	1	1	1	1
	2	1	4	10	23
	3	1	10	58	285
	4	1	23	285	2798

Na Tabela B.1 pode-se observar que há 7 funções com no máximo 2 transistores em série:

$f=\overline{a}$, $f=\overline{a\cdot b}$, $f=\overline{a+b}$, $f=\overline{a\cdot b+c}$, $f=\overline{(a+b)\cdot c}$, $f=\overline{(a+b)\cdot(c+d)}$, $f=\overline{a\cdot b+c\cdot d}$.

Este número está limitado a apenas 7 funções porque a Tabela 1 considera apenas redes CSP. A Teoria do *lower bound* descrita nesse trabalho nos permite revelar que na o *lower bound* para algumas funções é na realidade menor para algumas funções, como no caso das funções $f=\overline{a\cdot b+a\cdot c+b\cdot c}$ e $f=\overline{a\cdot b+a\cdot c+b\cdot d}$. Dessa forma, de fato há 5 funções *negative-unate* com exatamente 2 transistores em série em cada plano. O conjunto de funções *negative-unate* extras obtidas depois de considerar a Teoria do *lower bound*, para todas as funções com até 6 entradas, é apresentado na Tabela B.2.

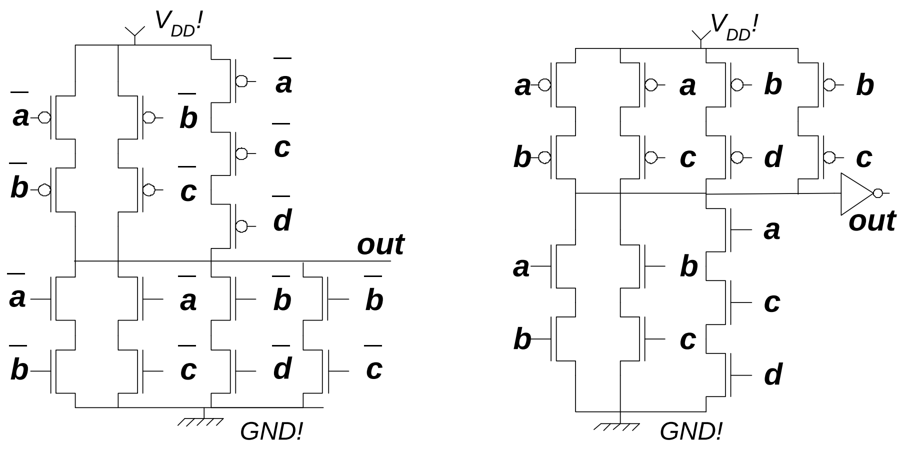
Tabela B.2: Número de funções *negative-unate* extras com um determinado número de transistors em série, considerando a Teoria do *lower-bound*.

		#Transistores PMOS em Série			
		1	**2**	**3**	**4**
#Transistores	**1**	0	0	0	0
NMOS	**2**	0	+2	+13	+62
em	**3**	0	+13	+498	+2897
Série	**4**	0	+62	+2897	+2222

Estilo Lógico NCSP

O estilo lógico CSP comumente utilizado para gerar portas lógicas CMOS assume não apenas que os planos de *pull-up* e *pull-down* são logicamente complementares – característica inerente de células CMOS – mas que eles são também topologicamente complementares – entradas em série em um plano se arranjam em paralelo no plano oposto. Entretanto, para se respeitar o *lower bound*, algumas células terão seus planos opostos não complementares topologicamente falando (ainda que logicamente eles continuam complementares). Assim sendo, neste trabalho um novo estilo lógico chamado NCSP, e que implementa células que sempre respeitam o *lower bound*, é então proposto e seu método de construção é detalhado.

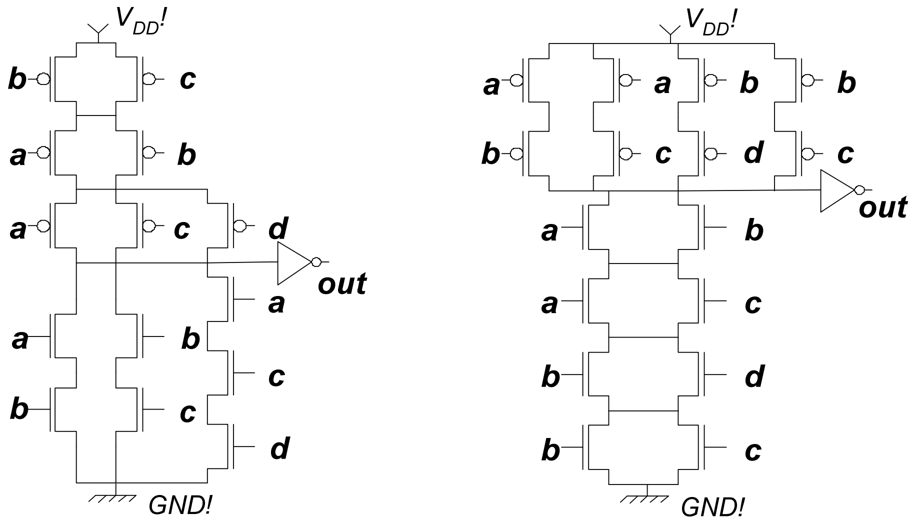
Como um exemplo, considere a célula resultante da função *f* apresentada no Exemplo 1. A célula NCSP tem tem seu plano de *pull-up* derivado da equação de *on-set* (i.e. o caminho mais longo nesse plano tem 3 transistores em série) e seu plano de *pull-down* derivado da equação de *off-set* (i.e. o caminho mais longo nesse plano tem 2 transistores em série), como pode ser observado na Figura B.1.a. Alternativamente, as equações de *on-set* e *off-set* podem ser trocadas quando a função for invertida e um inversor for adicionado à saída da células, resultando na célula da Figura B.1.b. Como transistores PMOS são mais resistivos que os NMOS, a implementação da Figura B.1.b é mais recomendada que a da Figura B.1.a.



(a) célula 3-2 para versão direta de *f*. (b) célula 2-3 para a versão invertida de *f*.

Figura B.1: Células NCSP respeitando o *lower-bound* para a função *f* apresentada no Exemplo 1.

Para a mesma função *f* mencionada acima, uma célula CSP derivada unicamente da equação de *on-set* resultaria em uma célula 3-3, como demonstrado na Figura B.2.a. De forma semelhante, se a célula CSP for derivada da equação de *off-set*, o resultado seria a Celula 2-4 ilustrada na Figura B.2.b. Este é um exemplo claro de como é possível, através de redes lógicas NCSP, atingir o *lower bound* que até então havia sido negligido na geração da maioria das células complexas.



(a) célula 3-3 cell a partir da equação de *on-set.* (b) célula 2-4 a partir da equação de *off-set.*

Figura B.2: Células CSP não respeitando o *lower bound* para a função *f* do Exemplo 1.

Resultados

Para gerar células CSP para uma determinada função é necessário escolher entre a função derivada do *on-set* ou do *off-set* da função. A escolha da expressão Booleana que melhor respeita o *lower bound* pode ser um pouco complicada em alguns casos. Por exemplo, sendo *PU_length* e *PD_length* o número de transistores no caminho com mais transistores no plano de *pull-up* e *pull-down*, respectivamente, considere os dois seguintes exemplos:

Exemplo A: *Lower Bound*: PU_length=2, PD_length=3

off-set: PU_length=**2**, PD_length=6 [$out = a \cdot (c + b + d) + d \cdot (c + b) + b \cdot c$]

on-set: PU_length=**3**, PD_length=4 [$out = \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot (\overline{b} \cdot (\overline{d} + \overline{c}) + \overline{c} \cdot \overline{d})$]

Exemplo B: *Lower Bound*: PU_length=3, PD_length=4

off-set: PU_length=**3**, PD_length=5 [$out = \overline{b} \cdot (c + d) \cdot (\overline{c} + \overline{d}) + b \cdot (\overline{c} \cdot \overline{d} + c \cdot d) + a$]

on-set: PU_length=**4**, PD_length=**4** [$out = \overline{a} \cdot (\overline{b} \cdot (\overline{c} + d) \cdot (c + \overline{d}) + b \cdot (c \cdot \overline{d} + \overline{c} \cdot d))$]

Para escolher entre as equações de *on-set* e *off-set* um critério é necessário. Dois critérios são então utilizados e aqui descritos:

CSP_1: Neste critério, a equação cujo plano de *pull-up* (mais crítico) respeita o *lower bound* é escolhida. Essa opção sempre é possível já que a polaridade da função pode ser invertida pra acomodar a equação que resultada num menor número de transistores em série. Para ambos Exemplos A e B, a equação escolhida é a de *off-set*.

CSP_2: Neste critério, escolhe-se a equação que resulte na menor diferença entre *PU_length* e *PD_length*. Para ambos Exemplos A e B, a equação escolhida é a de *on-set*.

Para ambos Exemplos A e B, a equação escolhida pelo critério CSP_1 não podem ser implementadas já que têm mais de 4 transistores em série. Das 3984 funções de quatro entradas existentes (*P-classes*), 118 casos apresentam topologias diferentes mas implementáveis para ambos critérios CSP_1 e CSP_2. As versões CSP_1 e CSP_2 dessas células foram caracterizadas e comparadas (Figura B.3).

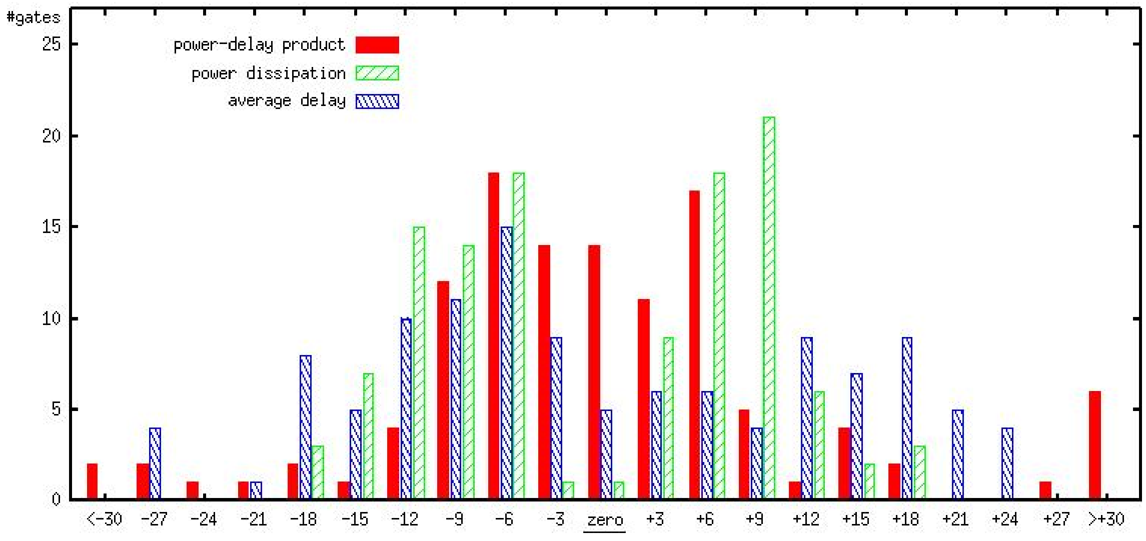
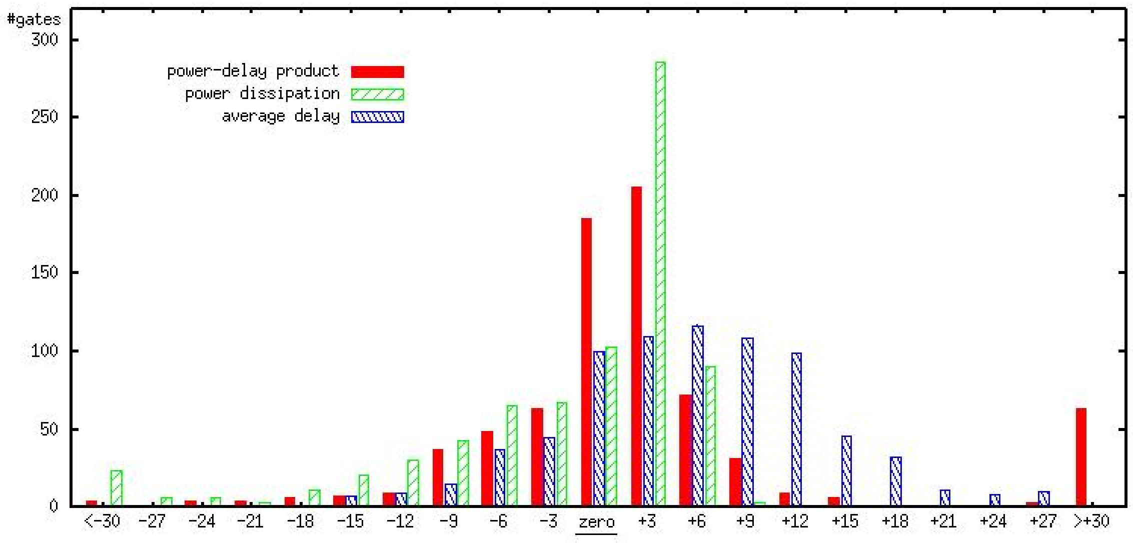
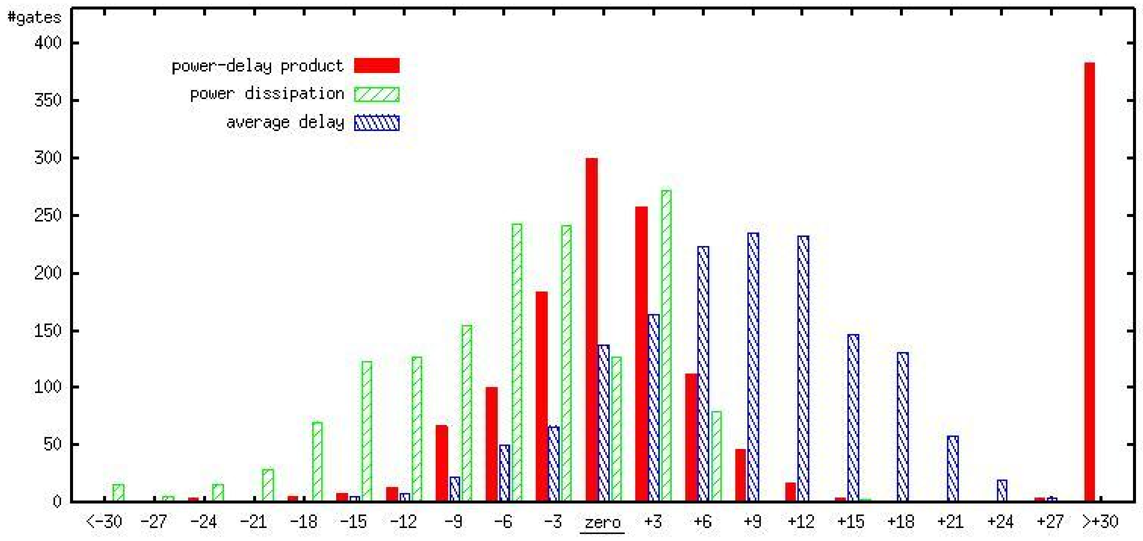


Figura B.3: Comparação do atraso de propagação entre CSP_1 (referência) e CSP_2, número de células de acordo melhoria (%) em atraso, potência e PDP.

Portas lógicas NCSP apresentadas neste trabalho foram então comparadas com os critérios CSP_1 e CSP_2 (Figura B.4). Apenas células eletricamente implementáveis (i.e. com no máximo 4 transistores em série) e que não respeitam o *lower bound* em ambos os planos foram utilizadas na comparação.



(a)



(b)

Figura B.4: NCSP comparado a (a) CSP_1 e (b) CSP_2: número de células de acordo com melhoria (%) em atraso, potência e PDP.

Na Figura B.5 é mostrado os ganhos em média de atraso obtidos com células de 6 entradas NCSP, comparadas com a implementação CSP, para quatro cargas de saída (2fF, 5fF, 10fF and 20fF) e curva de entrada de 0.5ns.

Na Figura B.6 é apresentada a comparação de potência, atraso, PDP e carga media de entrada para células de 6 entradas. A caracterização foi realizada usando 0.02ns para a curva de entrada e 2fF para carga de saída. Pode-se observar a carga de entrada aumentando para alguns casos, provavelmente devido ao metido de fatoração aplicado.

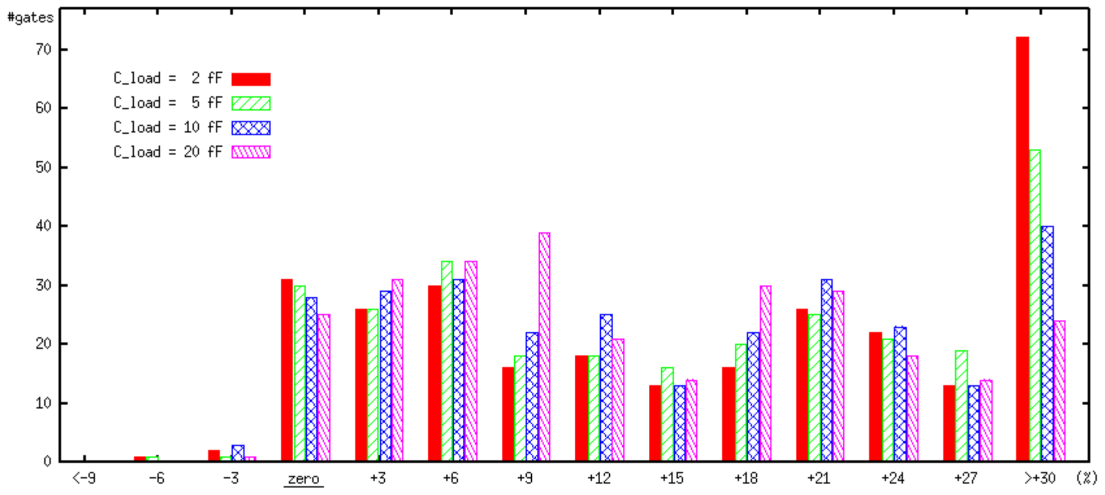


Figura B.5: média de melhoria (%) de NCSP, eixo-X, versus número de células de 6 entradas, eixo-Y, para diferentes cargas de saída e uma curva de entrada de 0.5ns.

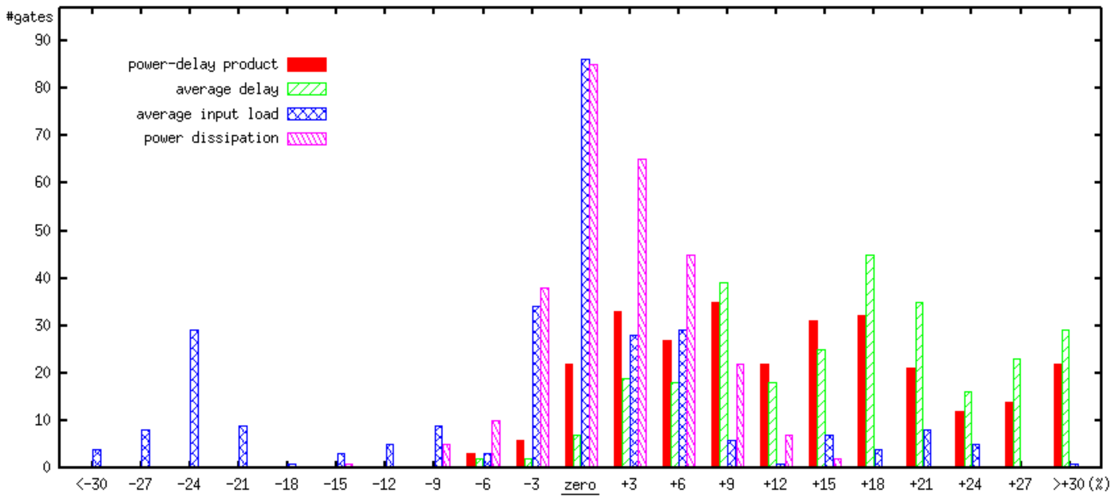


Figure B.6: melhoria (%) de NCSP, eixo-X, versus número de células de 6 entradas, eixo-Y, PDP, atraso médio, potência e carga de entrada média.

A tabela B.3 mostra, entre outras métricas, o esforço lógico médio obtido para todas as 3984 P-classes de funções de 4 entradas implementadas usando 6 diferentes técnicas para implementação das células em nível de transistores. O esforço lógico foi calculado usando a razão PMOS/NMOS igual a 2, tal como exemplificado em (SUTHERLAND, 1999). Pode-se observar que o esforço lógico médio para as construções que respeitam o *lower bound* – NCSP e CMOS LB from BDDs (ROSA, 2007) – são significativamente menores do que as calculadas com outros estilos lógicos.

Tabela B.3: Comparação de seis métodos diferentes para geração de redes de transistores.

Métricas	**CSP_1**	**CSP_2**	***NCSP***	**CMOS de BDDs**[a]	**CMOS de BDDs**[b]	***CMOS LB de BDDs***[c]
Σ *transistores*	75530	75456	*75889*	75774	73438	*72307*
Σ *PU_length*	11954	13084	*11954*	15538	14227	*11954*
Σ *PD_length*	17009	15931	*14242*	15538	15321	*14242*
Esforço Lógico (média)	4.54	4.37	*3.83*	4.35	4.07	*3.68*
Funções não respeitando *lower bound*	2312	2312	*0*	3148	2373	*0*
Funções não-implementáveis	1546	791	*0*	0	0	*0*

(a) (REIS, 1995) – (b) (POLI, 2003) – (c) (ROSA, 2007)

Conclusões

O novo estilo lógico CMOS desenvolvido neste trabalho, aqui referido como NCSP, é muito promissor. Essa tecnologia provou o quão significante podem ser as melhorias em timing – sem afetar área e potência de forma significativa – nas redes de transistores quando um número mínimo de transistores em série é utilizado. Além do mais, demonstrou-se que respeitando o *lower bound* para as redes de *pull-up* e *pull-down* permite que um maior conjunto de células lógicas sejam factíveis dentro de parâmetros elétricos aceitáveis.

Finalmente, uma das vantagens estratégicas das células NCSP é que elas podem ser utilizadas sem qualquer problema com outras células CSP disponíveis atualmente. Dessa forma, ferramentas de síntese podem escolher entre células CSP e NCSP baseando-se unicamente nas métricas que favorecem um ou outro estilo lógico.