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

SIMONE BAVARESCO

On-Silicon Testbench for Validation of Soft Logic Cell Libraries

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

Prof. Dr. Renato Perez Ribas
Advisor

Prof. Dr. André Inácio Reis
Co-advisor

Porto Alegre, February 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bavaresco, Simone

On-Silicon Testbenh for Validation of Soft Logic Cell Libraries / Simone Bavaresco – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

15 f.:il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2008. Advisor: Renato Perez Ribas; Co-advisor: André Inácio Reis.

1.Integrated circuit 2.ASIC 3.Digital design 4.Standard cell 5.Library-free technology mapping 6.Soft library 7.Test circuit. I. Ribas, Renato Perez. II. Reis, André Inácio. III. Title.

ACKNOWLEDGEMENTS

This dissertation is dedicated to the memory of my mother-in-law, Gioconda Madalena Loëblein. In the beginning I was a stranger and even though she helped me and gave me all comfort. She encouraged me in this study and over the last five years she supported me with her love and caring. To get on with her during these years gave me a new appreciation for the meaning and importance of family and friends. She faced her death bravely. Her example and will power kept me working when I wanted to give up.

I would like to express my deepest gratitude to my advisors, Renato Perez Ribas and André Inácio Reis for their excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research. Especially I would like to thank Renato for urged me by providing critical assistance when all seemed dark and guided me when the project seemed endless.

I would like to thank Nangate for financial support and thank all the staff, in particular, my colleagues from Nangate Lab Research, Caio Alegretti, Mateus Gomes, Leomar da Rosa, Carlos Afonso da Silva, Digeorgia Silva, Paulo Butzen, Felipe Marques and Tiago Cardoso. They were always willing to help and give their best suggestions.

There are many other people who contributed to this dissertation in many ways. My sincere gratitude goes to all my friends, especially José Francisco Szücs, Laura Ghia, Cleber Ughini and Sílvia Osório, who have contributed to my emotional and spiritual well-being over this time. I would like to thank for their love, inspiration and support.

I would also like to thank my parents, my brother, my sisters and brothers-in-law. They were always supporting me and encouraging me with their best wishes.

Finally, I would like to thank my husband, João Daniel Togni, for his support, encouragement, love and caring. He was always there cheering me up and stood by me through the good times and bad. His knowledge, valuable ideas and useful suggestions were also extremely helpful.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
BIST	Built-In Self-Test
CAD	Computer-Aided Design
CMOS	Complementary metal–oxide–semiconductor
CUT	Circuit Under Test
DFT	Design For Testability
EDA	Electronic design automation
FPGAs	Field Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
IP	Intellectual Property
IPO	In Place Optimization
ISCAS	International Symposium on Circuits and Systems
ITC	International Test Conference
IWLS	International Workshop on Logic and Synthesis
MCNC	Microelectronics Center of North Carolina
MOS	Metal-Oxide-Semiconductor
MPGA	Metal Programmable Gate Array
N-equivalent	equivalent under input Negation
NP-equivalent	equivalent under input Negation and input Permutation
NRE	Non-Recurring Engineering
NBTI	Negative-Bias Temperature Instability
OBIST	Oscillation Built-In Self Test
P-equivalent	equivalent under input Permutation
RTL	Register Transfer Level
SA	Stuck-At
SoC	System-on-Chip
SPICE	Simulation Program with Integrated Circuit Emphasis

STA	Static Timing Analysis
UFRGS	Universidade Federal do Rio Grande do Sul
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuits
VLSI	Very Large-Scale Integration

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Cell-based design is the most applied approach in the ASIC market today. This design approach implies re-using pre-customized cell libraries to build more complex digital systems. Therefore the ASIC design efficiency turns to be bounded by the library in use. The use of automatically generated CMOS logic gates in standard cell IC design flow represents an attractive perspective for ASIC design quality improvement. These soft IPs (logic cells generated by software) are the key elements for the novelty library-free technology mapping, already proposed in literature and now being adopted by the industry. Library-free technology mapping approach, based on the on-the-fly creation of cells, by software, can provide flexibility to IC designers providing an optimized fit in a particular application. However, such approach represents an IC design flow based on logic cells created on-the-fly by software which have not been previously validated in silicon yet, until the target ASIC is prototyped.

In this work, a specific test circuit (testbench) is proposed to validate the full functionality of a set of logic cells, as well as to verify timing and power consumption behaviors, which can be correlated with design timing and power estimations in order to validate the cell data provided by electrical characterization. The proposed architecture for the test circuit is composed by combinational blocks that ensure full logic verification of every library cell. The basic architecture of the test circuit is slightly modified to allow different operating modes which provide distinct data evaluation using SPICE electrical simulations. Since this test circuit brings little silicon overhead to the final design, it can be implemented together with the target ASIC acting as a 'library certification circuit'.

Keywords: Integrated circuit, ASIC, digital design, standard cell, library-free technology mapping, soft library, test circuit

Circuito de Teste em Silício para Validação de Bibliotecas de Células Lógicas Geradas por Software

RESUMO

Projeto baseado em células-padrão é a abordagem mais aplicada no mercado de ASIC atualmente. Essa abordagem de projeto consiste no reuso de bibliotecas de células pré-customizadas para gerar sistemas digitais mais complexos. Portanto a eficiência de um projeto ASIC está relacionado com a biblioteca em uso. A utilização de portas lógicas CMOS geradas automaticamente no fluxo de projeto de circuito integrado baseado em células-padrão representa uma perspectiva atraente para melhorar a qualidade de projeto ASIC. Essas células geradas por software são os elementos-chave dessa nova abordagem de mapeamento tecnológico livre de biblioteca, já proposto na literatura e agora adotado pela indústria. O mapeamento tecnológico livre de biblioteca, baseado na criação de células sob demanda, por software, gera flexibilidade aos projetistas de circuitos integrados, fornecendo ajuste otimizado em aplicações específicas. Contudo, tal abordagem representa um fluxo de projeto de circuito integrado baseado em células lógicas criadas sob demanda por software, as quais não são previamente validadas em silício até que o ASIC alvo seja prototipado.

Neste trabalho, um circuito de teste específico é proposto para validar a funcionalidade completa de um conjunto de células lógicas, bem como verificar comportamentos de atraso e consumo, os quais podem ser correlacionados com as estimativas de atraso e consumo do projeto, a fim de validar os dados das células gerados pela caracterização elétrica. A arquitetura proposta para o circuito de teste é composta por blocos combinacionais que garantem a completa verificação lógica de cada célula da biblioteca. A estrutura básica do circuito de teste é ligeiramente modificada para permitir diferentes modos de operação que permitem avaliação de diferentes dados utilizando simulações elétricas SPICE. Visto que o circuito de teste gera pequeno acréscimo de silício ao projeto final, ele pode ser implementado junto com o ASIC alvo, atuando como um ‘circuito de certificação de biblioteca’.

Palavras-Chave: Circuito integrado, ASIC, projeto digital, célula-padrão, mapeamento tecnológico livre de biblioteca, biblioteca de células geradas por software, circuito de teste.

1 INTRODUCTION

Microelectronics became the key technology of many industry branches like information technology, telecommunication, medical equipment and consumer electronics. The ability of microelectronics to process, transport and store data digitally made many new applications possible. The continuously increasing level of integration of electronic devices on a single substrate has led to the fabrication of increasingly complex systems. An *Integrated Circuit* (IC) is an electronic system consisting of a number of miniaturized electronic devices, such as transistors, resistors, capacitors and inductors, built on a monolithic semiconductor substrate. The large majority of the current ICs are implemented in the Metal-Oxide-Semiconductor (MOS) technology (WESTE, 2005) (RABAEY, 2003).

The integrated circuit technology has progressed tremendously. The increase in the number of transistors that can be integrated in a single die has grown exponentially in the last decades, as predicted by the so called *Moore's law* (INTEL, 2007) (MOORE, 1965). The level of integration of chips has been classified as small-scale (up to 10 gates –half a dozen transistors per gate), medium-scale (up to 1,000 gates), large-scale (up to 10,000) and *very large-scale integration* (VLSI). The term VLSI is used to describe most integrated circuits from the 1980s onward.

At present, many electronic systems require integrated and dedicated components that are specialized to perform a task or a limited set of tasks. These are called *Application Specific Integrated Circuits*, or ASICs.

Electronic design automation (EDA) is the category of tools for designing and producing electronic systems. This is sometimes referred to as CAD (*computer-aided design*). The modern ASIC design flow has evolved and increased in complexity just as the devices that are being designed have dramatically increased in complexity. This design flow is now heavily dependent on EDA tools and many of the tasks that were once carried out manually are now automated by EDA tools with little or no manual intervention.

Cell-based design is definitely the most applied approach in the ASIC market today. This design approach implies re-using pre-customized cell libraries to build more complex digital systems. A typical standard cell design environment includes timing and power analysis, as well as automatic generation of circuit layout. In standard cell libraries, three groups of logic cells co-exist: (1) inverters/buffers; (2) combinational and (3) sequential gates. Especially due to the large number of different logic functions and driving strength options needed in typical designs, the largest of the three aforementioned groups turns to be the combinational cells. The creation of handcrafted standard cell libraries needs skilled designers and requires long development times, even when simply dealing with technology migration for the same set of cells. Each cell is carefully defined and characterized for different input slopes, output loads, and design corners. This practice usually limits the number of cells in libraries due to the high engineering costs involved.

More recently, the library-free technology mapping has been addressed by EDA platforms (GAVRILOV, 1997). It automatically generates on-the-fly the cells required (or identified) by the mapping task. This new technology is based on virtual libraries, whose original cells are not previously designed and physically verified. It means that such virtual libraries are, in fact, tested together with the target ASIC.

Since in a standard IC design flow, the technology mapping is based on pre-designed and pre-characterized cells, the ASIC design efficiency turns to be bounded by the library in use. The more cells and drive strength options are available, the larger is the variety of logic functions and circuit performances the designer can implement. Only a library-free technology mapping approach, capable of creating on-the-fly cells that perfectly fit a particular application, can provide such flexibility to IC designers (DEDOOD, 2003) (GAVRILOV, 1997) (KAGARIS, 2007) (MARQUES, 2007) (ROY, 2005) (SCHNEIDER, 2005). The enrichment of a library can be done by adding only new drive strengths (DEDOOD, 2003) or through the addition of new functions with standard series/parallel implementation (GAVRILOV, 1997) or with special transistor topologies (KAGARIS, 2007) (MARQUES, 2007) (SCHNEIDER, 2005) and adding new cells for in-place optimization combining new topology and IPO (in place optimization) sizing (ROY, 2005).

A cell library is usually an ensemble of hundreds of each individual cell. If it is going to function as a whole to provide building blocks for larger designs, not only should each individual cell be correctly designed, but also the synthesized designs based on these library cells should be absent from errors. However, designing a cell library requires carrying out many complex tasks and involving design efforts from a number of engineers. Errors are easily made in this situation. In order to uncover the leftover errors of a cell library, benchmark circuits are usually designed to fulfill this mission. Thus, the quality of a cell library can be substantially improved (LIN, 1999).

Without accurate library validation, ASIC customers cannot be assured that their design will perform to specifications using simulation results alone. The need to correlate simulation precisely to silicon demands the manufacturing, testing and characterization of test circuits specifically designed for that purpose.

In the past, designing of a cell library was usually thought as a discipline in industrial community where the advancement in cell library design is largely made. Since a viable cell library will strengthen a company’s competitiveness, the know-how with regard to designing a high quality cell library is not often seen in open literature (AGASTEIN, 1990) (MARTINEZ, 1995) (SCOTT, 1994). However, with ever increasing availability of semiconductor foundry and commercial CAD tools to the university community and advancement in cell-based synthesis technology, more activities in cell library design within university community have been carried out (LIN, 1999) (CHONG, 1992).

There are certainly many interpretations for “high quality cell libraries”. A cell library regarded as high quality by one company may not be considered as viable by another company. However, high quality cell libraries possess many common characteristics. Here are listed some of them (LIN, 1999):

(1) the functionality of each individual cell should be correct in the models for logic synthesis and simulation,
(2) the timing performance figures of each individual cell claimed in the data sheet or models should be accurate enough,
(3) the layout of each cell should be free of design rule violations,
(4) the cells should be best utilized by a synthesis tool, and
(5) the cells should be able to optimize placement and route of a large design.

The testbench (test circuit) developed in this work enforces (1) and (2) which are essential to a high quality cell library.

The computing evolution, including parallel and distributed processing, allows nowadays the automatic generation and electrical characterization of a huge number of cells in a few minutes, although some design aspects like cell layout compaction are still challenging programmers. However, even if complete back-annotation and design verification are already feasible in a library-free approach, designers are sometimes still skeptical and reluctant to develop products based on software-generated cells that are not previously validated on silicon. This skepticism results from simple things like the lack of knowledge about the reliability of the circuit with respect to degradation due to electromigration (CHRISTIANSEN, 2006) and negative-bias temperature instability (NBTI) (KUMAR, 2007) effects.

The validation and physical characterization of the set of cells, included in a library, are usually done through specific structures and benchmark circuits. Such test structures are composed by ring oscillators, delay chains, counters, and others (LONG, 1984) (BHUSHAN, 2006). They are generally designed in full custom style, and must be carefully built for a specific process.

Benchmark circuits, such as ISCAS’89, MCNC’90, ITC’99 and IWLS’05 benchmarks, on the other hand, correspond to different applications and architectures in order to represent commercial circuits and system blocks, such as purely combinational circuits, finite state machines, arithmetic blocks, and so on. The use of benchmarks for validation and physical characterization of standard cell libraries may lead to two situations: (a) not all cells from the library are used in the benchmark circuit design, and (b) the cells used in this circuit are not stimulated by all possible input combinations, not guaranteeing the complete functionality of these cells. For instance, in Table 1.1, the use of different cells from a library, which contains 64 combinational cells, in the ISCAS benchmark circuit synthesis is observed. Note that, only the ‘tv80_core’ benchmark used the entire set of available cells.

Table 1.1: Using a 64-cell library to synthesize benchmark circuits.

*ISCAS Benchmarks**	*# cells in the circuit*	*# used cells from library*
c7552	1,311	41
i2c_master_top	679	42
iu	9,203	43
mc_top	6,245	58
tv80_core	5,594	64
wb_conmax_top	28,089	43

*Benchmarks source: http://www.fm.vslib.cz/~kes/asic/iscas, last access on Sep.20th, 2007.

Within this context, a naive manner to achieve library verification consists in designing a circuit that connects all cells in such a way that the primary inputs are shared and individual output signals are multiplexed to reduce the number of circuit primary outputs. This strategy can obviously achieve the desired verification of cell logic behavior but, in addition to multiplexing the cell outputs, it requires buffering to compensate the high capacitance on input nodes, making it difficult to obtain timing and power dissipation data.

An efficient approach to generate a testbench for testing a set of new cells, possibly created on-the-fly, should cover the following aspects:

(1) to ensure complete functionality test for the instantiated cells;
(2) to ensure coverage (instantiation) of all the cells to be tested;
(3) to allow the verification of the accuracy of the models used in the design process;
(4) to provide means to perform long and medium term reliability tests (needed for electromigration, sufficiency of contacts, NBTI degradation, etc) without additional equipment;
(5) to have a feasible number of cell instances compared to the set of cells to be tested.

The goal of this work is to propose a methodology to automatically generate testbench circuits for on-silicon soft-library validation meeting the aspects above.

Some works related to this propose can be found in the literature (LIN, 1999) (AGASTEIN, 1990) (MARTINEZ, 1995) (SCOTT, 1994) (CHONG, 1992). The most similar related work, presented by Rung-Bin Lin (LIN, 1999), creates benchmarks to improve the quality of a standard cell library, but such benchmarks are not viable for virtual libraries. It focuses on fixed libraries, not regarding the benchmark size and the automatic generation. It also uses specific structures (ring oscillators, delay chains, counters, and others) to compound the benchmark. Therefore, to the best of our knowledge, previous approaches fail to cope efficiently with this goal.

In this work, a straightforward and efficient testbench methodology is proposed aiming the validation of an entire set of soft-cells in terms of logic and electrical behavior. The presented solution merges well-established design and test concepts to cope with the five aspects mentioned above. A specific combinational block is built to guarantee the logic coverage (aspect 1) of a sub-set of the cells to be validated, and to provide at the output the same bit vector received at its inputs, allowing thus to cascade long chains with these blocks. The use of several blocks allows to instantiate all the cells (aspect 2). The circuit architecture is then composed of such combinational chain in a ring configuration, synchronized by a register barrier. Both synchronous and asynchronous operating modes provide different features for the proposed goals. The ring configuration allows verifying the accuracy of the models, by comparing with the predicted circuit behavior (aspect 3). The oscillation BIST technique is also included in the circuit operation for a wide range of different paths, and allows medium and long term tests (aspect 4). In case of an eventual error, the circuit diagnostic is facilitated through an arrangement of multiplexers. Finally, as the ring oscillator is composed of a variety of cells, the number of instances is not very expensive compared to the initial set of cells (aspect 5), as shown in the overhead section.

From a business model point-of-view, the methodology presented herein is useful for the soft-library vendor and to the ASIC designer client. For the vendor, it is quite important to dispose of a physical testbench in order to guarantee the correctness of its EDA environment, as well as to verify the quality of the generated cells in terms of performance and reliability, including design-for-manufacturability issues. This is essential for the continuous improvement of the library generation CAD tool. For the ASIC designer, a circuit that validates all distinct cells created on-the-fly to be used in a specific circuit provides means to exclude those errors on silicon due to the cell generators. If this test circuit is fabricated in the same die of the ASIC, it can act as a kind of ‘certification circuit’ for the soft-library, in different design corners and operating conditions. In this case, the overhead in terms of area and I/O pins is a compromise in fabricating together the test circuit and the ASIC, and the low cost approach presented here is very attractive.

The main concepts related to the work are presented in Chapter 2. In Chapter 3, the basic combinational block is presented. Chapter 4 presents the circuit architecture and its operation modes. Finally, the conclusions are discussed in Chapter 5.

2 BACKGROUND & MOTIVATION

In this chapter, the required concepts to the comprehension of this document, such as standard cell design, library-free technology mapping, testing, testability and design-for-testability are briefly explained.

2.1 ASIC Flow

Designing ICs as complex as the ones available nowadays requires engineers working with different levels of abstraction on a system design perspective. Figure 2.1 shows a diagram illustrating these different levels of abstraction known as Y-chart (GAJSKI, 1988). The Y-diagram can be used to illustrate each domain and the transformations between domains at varying levels of design abstraction. The radial lines on the Y-chart represent three distinct design domains: behavioral, structural, and physical. That is a classical tentative, among others, to demonstrate and classify such different working levels.

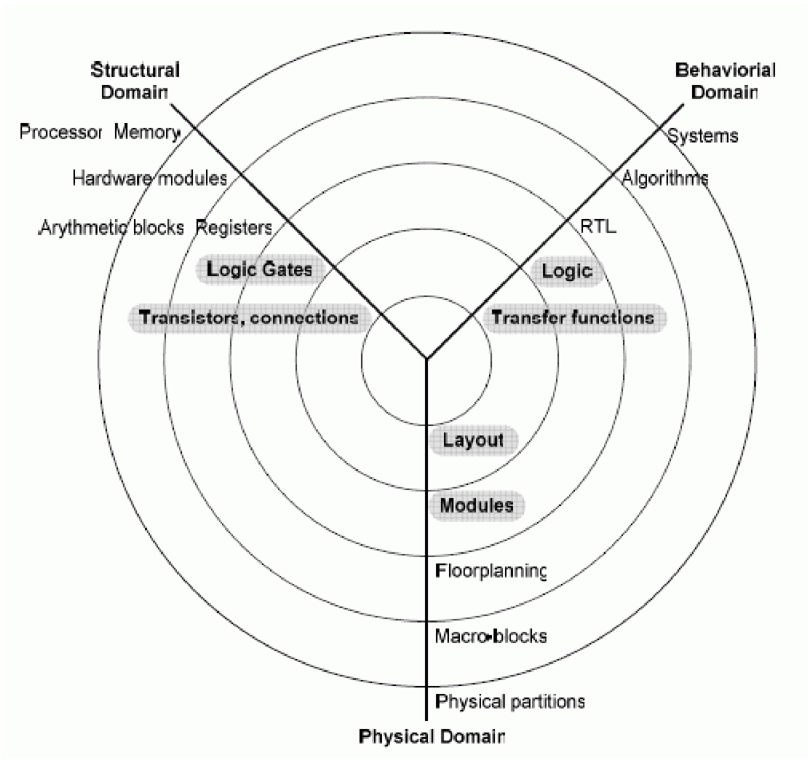


Figure 2.1: Levels of design abstraction (GAJSKI, 1988).

At the *behavioral domain*, the operation of the system is captured without having to specify the physical implementation. The design starts with a specification. From the specification a behavioral description of the design is then generated. The behavioral synthesis transforms a RTL behavioral description in a hardware description language (HDL), such as VHDL or Verilog. The design is then simulated and tested by applying testbenches to verify the correct behavior as defined by the specification.

The next step is to synthesize the behavioral description. This involves converting the RTL to generic gates and registers, then optimizing the logic to improve speed and area. The behavioral code is synthesized into a structural code using a generic gate library.

The step known as *logic synthesis* transforms a design from the behavioral to the *structural domain*. In the logic synthesis the task called *technology mapping* or *library mapping* takes a generic HDL gate-level description (boolean network) and translates it to a netlist (cell network) that specifies particular gates in the target library. Thus, the mapping task transforms the generic gate-level description into a gate-level description in the target ASIC technology.

Layout generation is the last step in the procedure of turning a design into a manufacturable database. It transforms a design from the structural to the physical domain. This step is called physical synthesis when the structural netlist is manipulated as physical layout is generated (WESTE, 2005). Figure 2.2 shows a diagram illustrating the ASIC design flow.

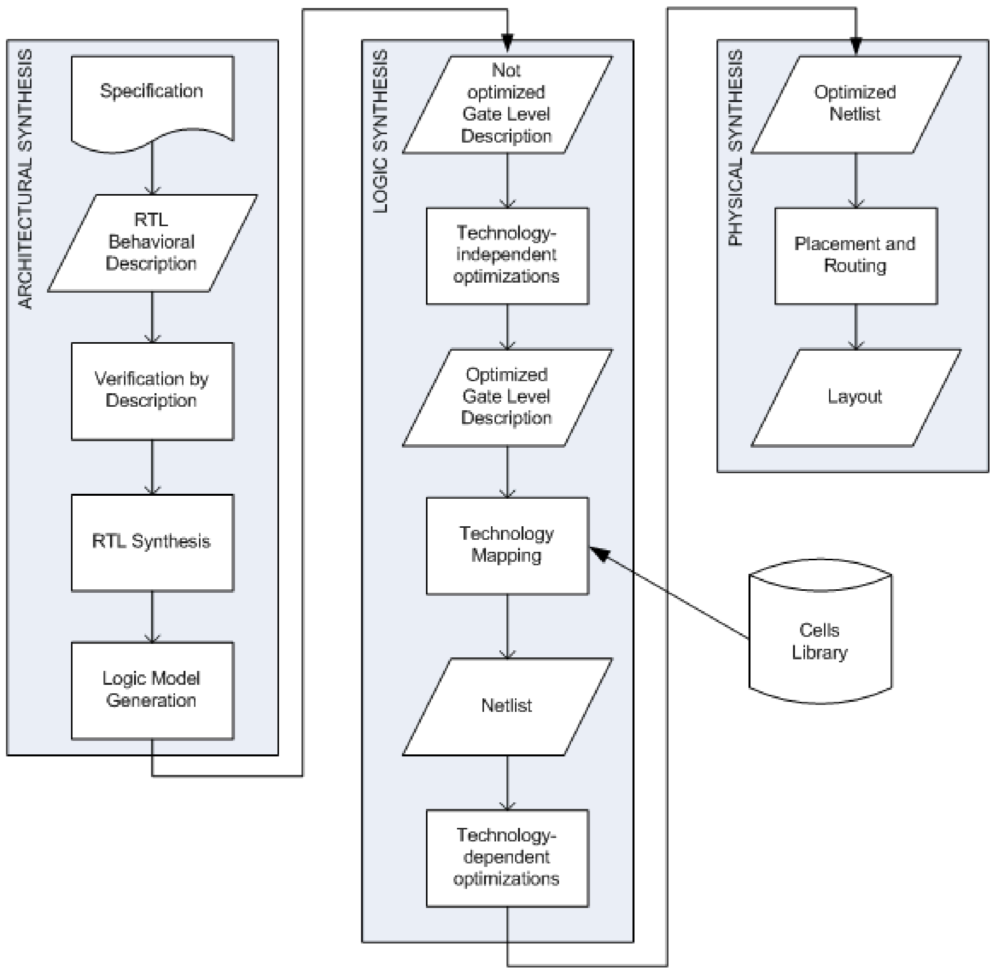

ARCHITECTURAL SYNTHESIS
Specification
RTL Behavioral Description
Verification by Description
RTL Synthesis
Logic Model Generation
LOGIC SYNTHESIS
Not optimized Gate Level Description
Technology-independent optimizations
Optimized Gate Level Description
Technology Mapping
Netlist
Technology-dependent optimizations
PHYSICAL SYNTHESIS
Optimized Netlist
Placement and Routing
Layout
Cells Library

Figure 2.2: ASIC design flow diagram (WESTE, 2005) (MARQUES, 2007a).

2.2 Design Styles

The economic viability of a microelectronics design depends upon a number of conflicting factors, such as production volume, cost, and circuit performance required to be competitive. These economic considerations have stimulated the development of a number of distinct implementation approaches that range from high-performance handcrafted design to fully programmable, medium-to-low performance designs (RABAEY, 2003).

Under these circumstances, different implementation approaches, often called design styles or even methodologies, have been used for microelectronic circuits. They are usually classified as full-custom (custom) and semicustom design styles (MICHELI, 1994). Figure 2.3 provides an overview of the design styles.

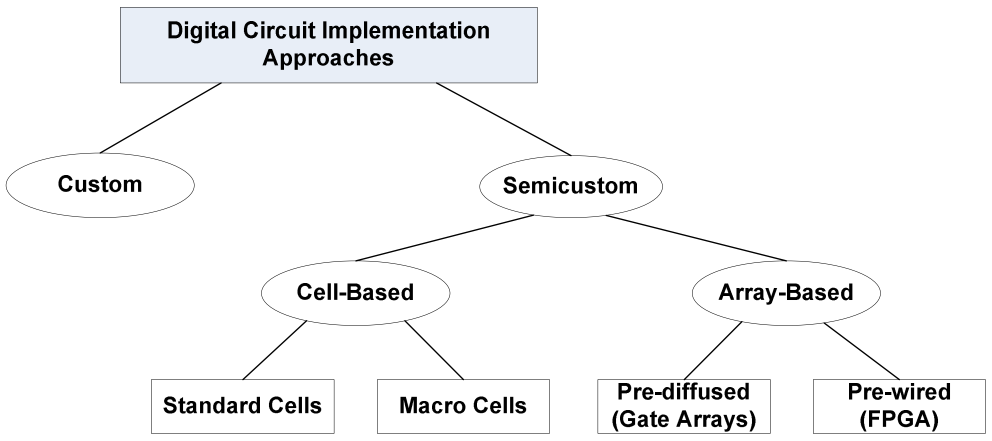


Figure 2.3: Overview of IC design styles (MICHELI, 1994) (RABAEY, 2003).

Full-custom design was popular in the early years of microelectronics. Today, the design complexity has confined custom design techniques to specific portions of a limited number of projects, such as processors, chip-sets, and arithmetic units (MICHELI, 1994).

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, 2000).

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 this design style can include increased manufacturing and design time, increased non-recurring engineering (NRE) costs and a much higher skill requirement on the part of the design team.

However, for digital designs only, 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.

Semicustom designs can be partitioned in two major classes: *cell-based design* and *array-based design*. These classes further subdivide into subclasses, as shown in Figure 2.3. Cell-based design leverages the use of library cells, that can be designed once and stored, or the use of cell generators that synthesize macro-cell layouts from their functional specifications. Array-based design exploits the use of a matrix of configurable elements to implement the logic. Array-based circuits can be classified as pre-diffused and pre-wired, also called *mask programmable* and *field programmable* gate arrays, respectively (MPGAs and FPGAs) (MICHELI, 1994).

This work is included in the context of the s*tandard-cell design* which is a *cell-based design style*. 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 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-INVERTER), MUX, full-adder, comparator, counter, decoder, encoder, and so on.

The layout of each cell in a specific library has a fixed height, while its width may vary. Thus, 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. 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).

Library-free Technology Mapping

Technology mapping is the choice of the elements from a technology (typically cells from a library) that will be effectively used to implement a given circuit. Technology mapping then transforms the network of functions into a gate level netlist by trying to find an optimal network covering of library gates, with respect to some cost function (CORREIA, 2004).

The library gates used by a technology mapper can be specified using functionality (also known as library-based mapping) or parameters such as the number of inputs and series/parallel devices (also known as library-free mapping).

Figure 2.4 shows the digital circuit design flow regarding the library-based methodology for the technology mapping. In this methodology the quality of the mapped circuit has a directly relation with the library richness. Static libraries usually are not so big, because the cells characterization cost is too elevated (SECHEN, 2003). A commercial library contains no much more than one hundred cells that implement different boolean functions (MARQUES, 2007). The great advantage of this approach is the pre-characterization of the cells and the set of information associated to each cell of the library, which is considered as design cost in the map matching. Usually, this information is about area, delay, input capacitances and power consumption.

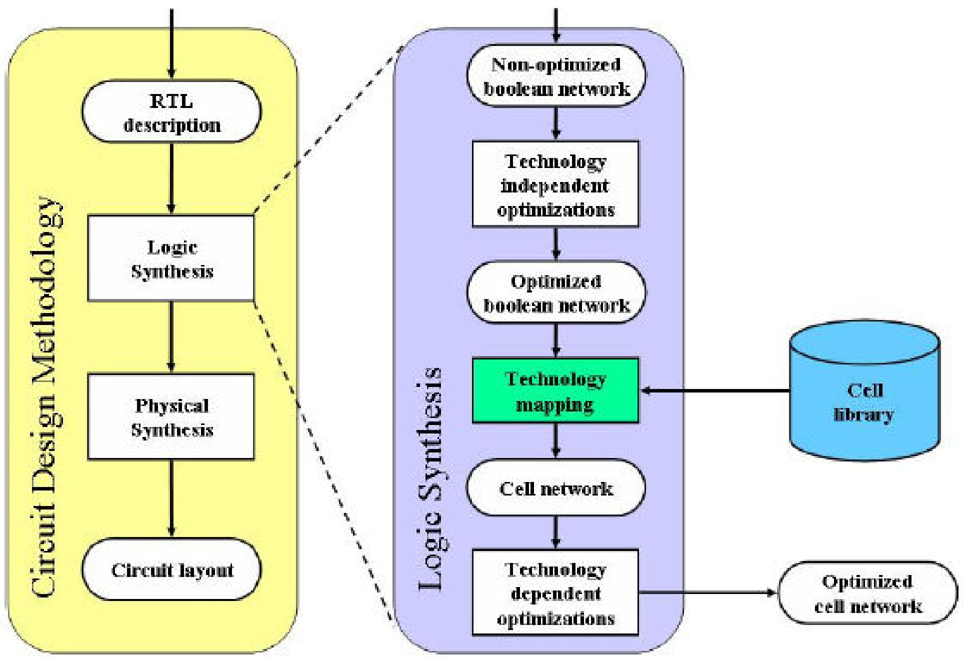

Circuit Design Methodology
RTL description
Logic Synthesis
Physical Synthesis
Circuit layout
Logic Synthesis
Non-optimized boolean network
Technology independent optimizations
Optimized boolean network
Technology mapping
Cell network
Technology dependent optimizations
Cell library
Optimized cell network

Figure 2.4: ASIC design flow with library-based technology mapping (MARQUES, 2007).

Figure 2.5, in turn, shows the digital circuit design flow regarding the methodology library-free for the technology mapping. In this approach, the mapping defines a set of cells that will be used to implement the circuit. This list of cells is then used by an automatic cell generator in order to obtain the implementation of cells to use in the circuit layout.

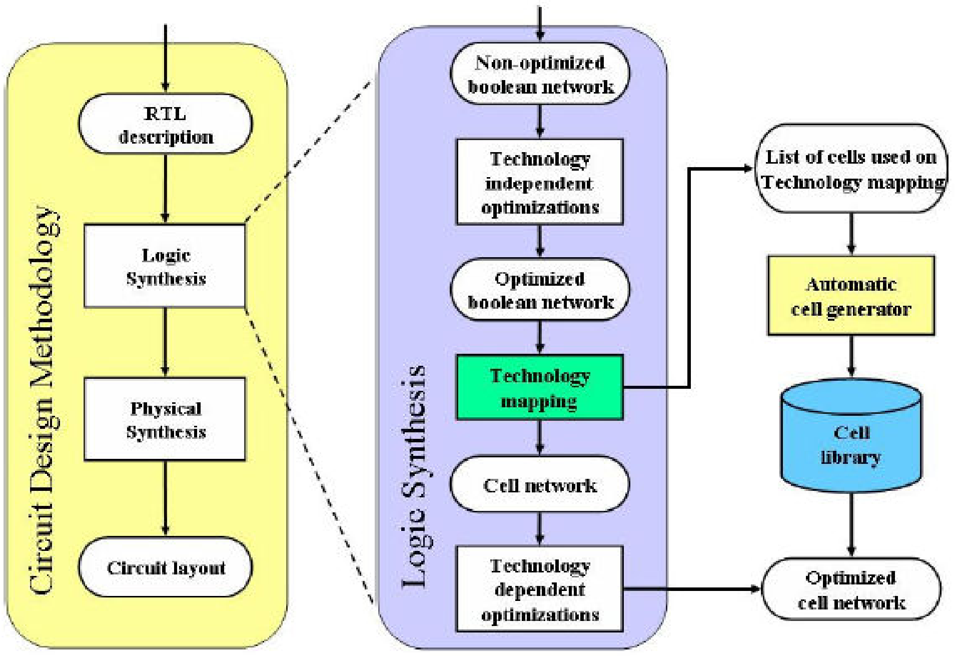

Circuit Design Methodology
RTL description
Logic Synthesis
Physical Synthesis
Circuit layout
Logic Synthesis
Non-optimized boolean network
Technology independent optimizations
Optimized boolean network
Technology mapping
Cell network
Technology dependent optimizations
List of cells used on Technology mapping
Automatic cell generator
Cell library
Optimized cell network

Figure 2.5: ASIC design flow with library-free technology mapping (MARQUES, 2007).

Thus, the concept of library-free design is based on using a virtual library available through a layout generator instead of using a set of pre-designed cells, like in the library-based design. The set of available cells is given by a user-defined constraint in the number of series transistors. This constraint in the number of series transistors is done for electrical reasons. As the cells are generated on-the-fly, a virtual library contains a great number of poorly characterized cells when compared to a pre-designed standard cell library (REIS, 1998) (REIS, 1999). Therefore, the fact of the cells not being previously designed and physically verified is the main drawback of the library-free technologic mapping and it means that such virtual libraries are, in fact, tested together with the target ASIC. In this work, a circuit for on-silicon verification and validation of digital cell libraries is proposed. This solution uses test concepts that will be presented next.

2.3 Testing

2.3.1 Defect, fault, failure and error

A *defect* in an electronic system is the unintended difference between the implemented hardware and its intended design. A *fault* is a representation of a "defect" at the abstracted function level, reflecting a physical condition that causes a circuit to fail to perform in a required manner. A *failure* is a deviation in the performance of a circuit or system from its specified behavior and represents an irreversible state of a component such that it must be repaired in order to provide the intended design function. A circuit *error* is a wrong output signal produced by a defective circuit. A circuit defect may lead to a fault, a fault can cause a circuit error, and a circuit error can result in a system failure (BUSHNELL, 2000) (WANG, 2006).

2.3.2 Test Generation, Functional and Structural Testing

In order to test a circuit, a set of input patterns is applied to the circuit under test (CUT), and its responses are compared to the known good responses of a fault-free circuit. Each input pattern is called a *test vector*. The goal of *test generation* is to find an efficient set of test vectors that detects all faults considered for that circuit (WANG, 2006).

Traditionally, manufacturing test has been done using *functional testing*. Functional test patterns verify that the model or logic behaves as it was intended. Every entry in the truth table for the combinational logic circuit is tested to determine whether it produces the correct response. Therefore, functional testing is measured by the logic committing the correct action to the applied stimuli. Full functional correctness is the standard expectation and this should be verified at behavioral (RTL) or at gate level of the design with a simulation process (GIZOPOULOS, 2006).

A more practical approach is to select specific test patterns based on circuit structural information and a set of *fault models*. This approach is called *structural testing*. Structural testing is used to verify the topology of the manufactured chip (GIZOPOULOS, 2006). Such testing relies on fault models, which assume that the physical defect will represent itself in a certain way. Structural testing saves time and improves test efficiency, as the total number of test patterns is decreased because the test vectors target specific faults that would result from defects in the manufactured circuit.

Structural testing cannot guarantee detection of all possible manufacturing defects, as the test vectors are generated based on specific fault models. However, the use of fault models provides a quantitative measure of the fault detection capabilities of a given set of test vectors for a targeted fault model. This measure is called *fault coverage* and is defined as:

fault coverage = (*number of detected faults*) / (*total number of faults*)

It may be impossible to obtain a fault coverage of 100% because of the existence of undetectable faults (WANG, 2006). An undetectable fault means there is no test to distinguish the fault-free circuit from a faulty circuit containing that fault. As a result, the fault coverage can be modified and expressed as the *fault detection efficiency*, also referred to as the *effective fault coverage*, which is defined as:

fault detection efficiency = (*number of detected faults*) / (*total number of faults* - *number of undetectable faults*).

2.3.3 Fault models

Manufacturing faults can be of a wide variety and manifest themselves as short-circuits between signals, short-circuits to the supply rails, and floating nodes. Because of the diversity of defects, it is difficult to generate tests for real defects. In order to evaluate the effectiveness of a test approach, generating and evaluating a set of test vectors, and qualify a good or bad circuit, these faults must be related to the circuit model, i.e. derive a *fault model*.

Generally, a good fault model should satisfy two criteria: (1) it should accurately reflect the behavior of defects, and (2) it should be computationally efficient in terms of fault simulation and test pattern generation (WANG, 2006). Many fault models have been proposed, but no single fault model accurately reflects the behavior of all possible defects that can occur. As a result, a combination of different fault models is often used.

2.3.3.1 Stuck-At Faults

Stuck-at fault is the most common fault model used in fault simulation. Functional testing uses the *single stuck-at model* because of its effectiveness in finding many common defect types. The stuck-at fault models the behavior that occurs if the terminals of a gate are stuck at either a high (stuck-at-1, sa1) or low (stuck-at-0, sa0) voltage.

2.3.3.2 Delay Faults

Fault-free operation of a logic circuit requires not only performing the logic function correctly but also propagating the correct logic signals along paths within a specified time limit. A *delay fault* causes excessive delay along a path such that the total propagation delay falls outside the specified limit.

Timing-related failures may be caused by isolated gate delays or process-related timing problems that accumulate along logic paths and prevent the circuit from functioning at-speed. The timing-related malfunction is characterized by defining the concept of delay faults related to circuit critical paths. Conventional techniques for delay test require two distinct primary input vectors that provoke a transition signal at the fault site and propagate the faulty delay effect to a primary output (ARABI, 1998).

Timing related defects have been modeled as different delay fault models. In *gate-delay fault* and *transition fault* models, a delay fault occurs when the time interval taken for a transition from the gate input to its output exceeds its specified range. The other model is *path-delay fault*, which considers the cumulative propagation delay along a signal path through the CUT, i.e. the sum of all gate delays along the path. The combinational path begins at a primary output or another clocked flip-flop, contains a connected chain of gates, and ends at a primary input or a clocked flip-flop. The specified time range can be the duration of the clock period (or phase) or the vector period. The propagation delay is the time that a signal event (transition) takes to traverse the path (BUSHNELL, 2000).

In a combinational circuit the path that has the longest propagation time from a primary input to a primary output, called *critical path*, determines the operating speed of the circuit. Also in a sequential circuit, the system is free of timing failures if every combinational path between two memory elements propagates its signal in less time than the interval of the operating system clock. In other words, the input signal of every memory element in the system should have a stable signal before the arrival of the active clock edge, obviously not forgetting the setup time of such memory point (registers). To be reliable, at least all critical paths in the system should be tested (ARABI, 1998).

2.4 Delay test verification

Digital circuits are tested by verifying the state malfunction in logic based on a standard fault model, the “stuck-at 0 or 1” fault. This fault model successfully describes most of state malfunction in logic. However, as the structure of logic circuits has become increasingly complex, system-timing failures occur more frequently.

The timing-related malfunction is characterized by defining the concept of delay faults related to circuit critical paths. Conventional techniques for delay testing require two distinct primary input vectors that provoke a transition signal at the fault site and propagate the faulty delay effect to a primary output.

Timing related defects have been broadly modeled as gate delay faults or as path delay faults. The gate delay fault model assumes that the incorrect timing behavior of the circuit is due to excess delays in one or more components in the path. Test vector generators based on gate delay fault model deal with one fault at a time and try to find a test which sensitizes some path trough the fault location such that the transition at the output is affected by the target fault. The path delay fault model considers the propagation delay through one or more paths exceeding the timing constraint. Therefore, this model makes no assumption about the individual component delays. To be reliable, at least all critical paths in the system should be tested.

2.4.1 Characterization

Accurate determination of logic gate propagation delay and its correlation with processing variables and layout design rules is needed to optimize the high-speed performance of actual logic circuits (LONG, 1984). However, accurate wafer-level high speed measurement of timing and delay is limited by the test interface.

Synchronous circuits have been employed for measurement of propagation delay. The maximum clock frequency at which the circuit presents the proper data signature can be directly measured. An average propagation delay per gate can be inferred from the maximum clock frequency through transient simulation or timing analysis. However, synchronous circuits are often much more difficult to evaluate with accuracy because of the high-speed test interface problems, such as reflections and ground noise and because the observed maximum frequency of operation is critically dependent on clock waveform properties such as symmetry, amplitude, and offset. Also, it can be shown that the maximum clock frequency for a given flip-flop type and implementation (NAND, NOR, etc.) will be achieved only at a specific clock symmetry or nonsymmetry.

Therefore, to stay within these limitations, self-test or built-in test circuitry must also be included on all large-scale high-speed ICs to reduce the total number of off-chip interfaces needed to verify functionality at speed.

The design of a test structure for propagation delay measurement must facilitate accuracy and simplicity of measurement and provide correlation between the test structure and actual logic circuits. The "classic" or default asynchronous test structure has generally been the ring oscillator, often configured with minimum fan-in, fan-out and interconnect capacitive loading. Measurement of this circuit is very simple, requiring only power supply, ground, and output connections.

2.4.2 Delay fault testing

The maximum allowable clock rate is determined by the propagation delays of the combinational logic block between latches. To observe delay defects, it is necessary to create and propagate transitions in the circuit running *at-speed* (at its specified operating frequency).

Creating transitions requires application of a vector pair, $V = <v_1, v_2>$, at the inputs of the combinational part of the circuit. The first vector initializes the relevant internal signals to desired initial logic values, while the second vector causes the desired transitions and sensitizes the transition from the target fault site to an output. The test application scheme for combinational circuits is shown in Figure 2.6.

In normal operation, only one clock, the system clock, is used to control the input and output latches and its period is T_c. In this illustration, the input and output latches are controlled by two different clocks in the test mode: the input and output clocks, respectively. The period of these clocks, T_s, is assumed to be larger than T_c. The input and output clocks are skewed by an amount equal to T_c.

The first vector, v_1, is applied to primary inputs at time t_0. The second vector, v_2, is applied at time t_1. Time $T_s = t_1 - t_0$ is assumed to be sufficient for all signals in the circuit to stabilize under the first vector. After the second vector is applied, the circuit is allowed to settle down only until time t_2, where $t_2 - t_1 = T_c$. At time t_2, the primary output values are observed and compared to a prestored response of a fault-free circuit to determine if there is a defect.

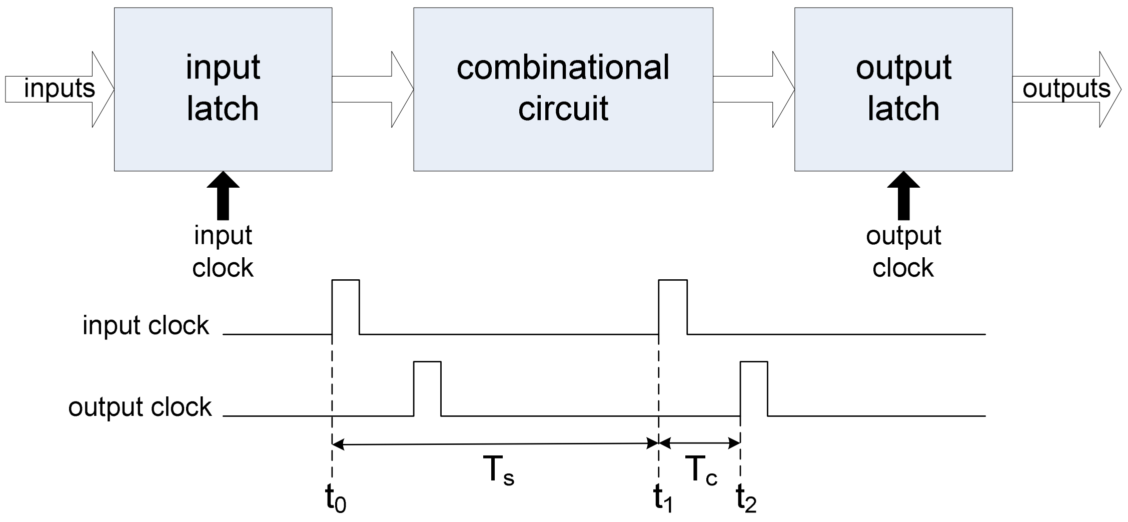


Figure 2.6: Delay fault testing.

Transition fault model, gate fault model and path delay fault modes are the classical fault models that have been used to represent delay defects.

2.5 Testability and Design-for-Testability

Testability is a relative measure of the effort or cost of testing a logic circuit. When considering the testability of designs, two properties are important: controllability and observability. (1) *Controllability* is the ability to establish a specific signal value at each node in a circuit by setting values on the circuit inputs. (2) *Observability* is the ability to determine the signal value at any node in a circuit by controlling the circuit inputs and observing its outputs (ABRAMOVICI, 1990).

The correctness of a combinational circuit can be validated by exhaustively applying all possible *input patterns* and observing the responses. A more feasible testing approach is based on the following premises. Firstly, an exhaustive enumeration of all possible input patterns contains a substantial amount of *redundancy*, that is, a single fault in the circuit is covered by a number of input patterns. Detection of that fault requires only one of those patterns, while the other ones are superfluous. Secondly, a substantial reduction in the number of patterns can be obtained by relaxing the condition that all faults must be detected. For instance, detecting the last single percentage of possible faults might require a huge number of extra patterns, and the cost of detecting them might be larger than the eventual replacement cost. Typical test procedures only attempt 95-99% fault coverage.

By eliminating redundancy and providing reduced fault coverage, it is possible to test most combinational logic blocks with a limited set of input vectors. However, this does not solve the sequential problem. To test a given fault in a state machine, it is not sufficient to apply the correct input excitation since the engine must be brought to the desired state first. This requires a sequence of inputs to be applied. Propagating the circuit response to one of the output pins might require another sequence of vectors.

One way to address the problem is to turn the sequential network into a combinational one by breaking the feedback loop in the course of the test. This is one of the key concepts behind the *scan-test* methodology (ABRAMOVICI, 1990). Another approach is to let the circuit test itself. Such a test, called *self-test*, does not require external vectors and can proceed at a higher speed (RABAEY, 2003).

Design-for-testability – DFT represents design techniques that are required in order to improve the testability of the design for achieving quality and reducing the test cost of the digital circuit, while at the same time simplifying the test, debug and diagnose tasks. By considering testing from the early phases of the design process, it is possible to simplify the whole validation process. *Scan design* and *built-in self-test (BIST)* are examples of design techniques to improve testability. Later these concepts will be better explained in the context of this work.

2.6 How to test a library?

The main question of the work presented herein is "How to test a digital cell library". In particular, when such cells are automatically generated by software, they are directly applied to the target ASIC, and this set of cells tends to be composed by a huge number of different combinational logic gates.

Instead of verifying the correctness of a specific circuit, the objective of this work is to propose a circuit, whose main goal is the verification and validation of each cell presented in a standard cell library. It represents, in fact, a new issue in the test domain where usually test methods are applied to test the functionality of application circuits and systems. In this particular case, the proposed circuit is not useful for any specific application or electronic product, but to test a set of logic cells generated by software to their use in a particular ASIC design.

Before discussing the circuit architecture, the structure of the combinational blocks are presented. The combinational blocks represent the basic components of the proposed test circuit.

3 COMBINATIONAL BLOCK

This chapter presents the description of the combinational blocks which are the basic structures of the proposed circuit.

In terms of logic cell functionality, three main groups may be identified: (1) inverters and buffers; (2) sequential cells; (3) combinational cells. Group (1) is easily verified since such a kind of cell presents only one input signal. Group (2), in turn, has generally a small and limited number of different latches and flip-flops, facilitating its verification. Moreover, in this group the timing performance is usually more important than the logic functionality that is somewhat trivial in the pass and storage modes. In the case of group (3), the number of cells is generally more expressive than the other ones. Moreover, the number of input nodes in these cells makes the functional test a more complex task due to the 2^n different input combinations, being n the number of input nodes.

The most naïve strategy to test the group (3) consists in placing all cells connected in parallel to the same input bus, where all input combinations are applied simultaneously. However, in doing so, input buffering must be considered due to the high node capacitances (great number of inputs connected to the same node). Furthermore, multiplexers should be used at the output signals to reduce the number of output pads. In this approach, timing and power consumption characteristics are not easily verified since the signals propagate only through single cells in testing.

The proposed architecture consists in building combinational blocks that receive an input bus, where all signal combinations are provided, and produce a sequence of output vectors also presenting all possible signal combinations, to be then applied to the next combinational block. The combinational block is illustrated in Figure 3.1.

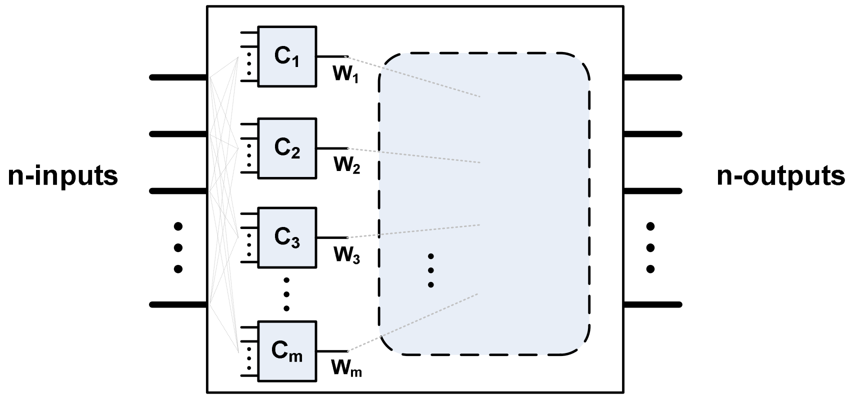


Figure 3.1: Combinational block illustration.

In order to ensure full logic verification of every library cell, the combinational blocks used in our verification circuit are built according to the following principles:

- a cell from the library is declared verified if it is instantiated such that inputs of the cell consume primary inputs of a combinational block. Polarity is indifferent and the cell can consume the input signals through an inverter or directly;
- every library cell is instantiated at the primary inputs of at least one combinational block, to ensure it is verified;
- block creation continues while there are cells from the library that are not yet declared as verified cells;
- every block instantiates a non-empty subset of non-verified cells at the block inputs, so that more cells receive the verified label. Indeed, the creation of a block is done in a way to maximize the number of cells that passes from the unverified to verified status.

From a block instantiation point of view, the combinational blocks obey to the following principles:

- all possible input combinations are applied to every block;
- every block reproduces the input signals at its outputs, providing thus all possible signal combinations for the next block.

Each combinational block is built using two cascaded stages. The minimum number of signals at the input bus is determined by the greatest number of inputs of a single cell used in the first stage.

The first stage is built with instances of cells placed in one-logic level exercised by all possible input combinations. The number of cells used in the first stage is the sufficient number to create 2^n logic combinations at the output nodes of the first stage.

The output signals of verified cells are used as inputs to the second stage, which regenerates the primary inputs. The second stage is composed of synthesized functions with additional cells instances in a multi-logic level.

3.1 First stage

To generate the first stage, the whole set of cells in the library is ordered according to a given criterion; for example alphabetic order, number of inputs, number of 0s and 1s at the output of the logic function, random order, or any other. Once the cells are ordered, non verified cells are picked one-by-one to create a verified instance.

Assuming an n-input block, the goal is to choose a number m of cells in the first stage sufficiently large to produce at least 2^n different m-bit output vectors at the first stage. The 2^n intermediate different combinations (m-bit vectors) are required to reconstruct the original 2^n input vectors (combinations) with a one-to-one correspondence. To achieve this goal, the instance of a new cell considers all possible direct and inverted connections between the block primary inputs and the inputs of each newly selected cell are tried.

For a library where only single-output cells are available, the smallest circuit possible for the first stage has n cells, while the worst case is a first stage containing 2^n-1 cells. Verified selected cells may or may not re-appear in the second stage or in other combinational blocks.

An example for generation of a 3-bit output vector with the smallest circuit possible for the first stage is shown in Table 3.1. In this case, all cells have the same number of logic values ‘0’ and ‘1’.

Table 3.1: Minimum number of cells at the first stage.

A	**B**	**C**	**XOR3_X1 (A,B,C)**	**XOR2_X1 (B,C)**	**XOR2_X2 (A,C)**
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	0	0

This maximum limit of necessary cells is explained by the worst case of a block constitution. If a block must generate 3-bit output vector and has only cells with one logic value ‘1’ available to use, it will be needed at most 7 cells to generate 8 distinct vectors at the output of the first stage as shown in Table 3.2.

Table 3.2: Maximum number of cells at the first stage.

A	**B**	**C**	**NOR3_X1 (A,B,C)**	**AND3_X2 (B,A,!C)**	**NOR3_X2 (!C,A,B)**	**AND3_X1 (C,B,!A)**	**NOR3_X4 (A,C,!B)**	**OR3_X4 (!C,!B,!A)**	**NAND3_X1 (C,!B,A)**
0	0	0	1	0	0	0	0	1	1
0	0	1	0	0	1	0	0	1	1
0	1	0	0	0	0	0	1	1	1
0	1	1	0	0	0	1	0	1	1
1	0	0	0	0	0	0	0	1	1
1	0	1	0	0	0	0	0	1	0
1	1	0	0	1	0	0	0	1	1
1	1	1	0	0	0	0	0	0	1

3.1.1 Implementation

In general, it is possible to briefly describe the test circuit generation flow as follows. Firstly, a parsing of a library is done for the identification of the existing cells. Thus, these cells are sorted in a list and the construction of the first part of the block begins with the search of cells in the list. In the search task, the cells are tested with some or all inputs combinations in order to target a certain number of output vectors combinations.

The first part of a block is complete when the desired number of distinct output vectors combinations is achieved and then the construction of another one starts. This process continues until all cells of the library have been used at the first stage of one of the blocks.

After the first stage is complete in every block, the functions that will recreate the input vector will be generated. These functions represent the second part of each block.

In this work, the generation of combinational blocks was automated by a specific CAD tool, with parameters to configure the cell list sorting criterion, the number of input/output nodes of the block and the enabling/disabling of the NP equivalences for the cells in the first stage of the blocks. Next, the implementation of this CAD tool will be detailed.

(a) Parse

The generation of the combinational blocks starts with the parsing of a Liberty library file or an Eqn file, which is loaded into the tool data structure.

(b) Sort

The initial stage consists in sorting the available cells. This choice will influence on the order in which they will be used later. Cells can be sorted through five methods: (i) alphabetical ordering; (ii) input number; (iii) quantity of minterms; (iv) random ordering; (v) difference between the quantity of zeros and ones of the cell function.

(c) Search

Having the set of cells sorted as desired, the construction of the first stage for a block begins. The order in which the cells are picked from the available list can be set in three different manners: sequentially, alternately (first-last) and randomly.

(d) Combinations testing

The first picked cell is put in the block with its original inputs. So, each picked cell, except the first one of the block, is tested with all the combinations of the block and cell inputs – if no stopping criterion is used regarding the number of combinations; or it is tested with some of the block and cell inputs – if using a stopping criterion.

The goal is to achieve the maximum number of different combinations with the addition of each cell. A cell is added to the block, only if its addition creates a new output combination and the combination of cell inputs that generates more different output combination will be selected. Although, if the cell input combination in test generates the maximum number of possible combinations for the given number of cells, the combinations testing stop and the actual combination is used as the cell input combination.

The *block inputs combinations* are defined as the arrange of the block inputs that can be used as cell inputs.

Defined the block input arrange which will be used as cell input, the permutations and/or inversions of this arrange are defined as the *cell inputs combinations*.

There are three sets of combinations that can be tested as cell inputs, given a block input combination:

- N: combinations considering negations.
- P: combinations considering permutations.
- NP: combinations considering negations and permutations.

Given the set of combinations that will be tested as inputs for each cell, the software can follow the standard sequence of the combinations list to test or it is possible to select a combination from the list randomly in order to test. Also for the block inputs combinations, the test can follow the standard sequence of combinations or the combination to test can be selected randomly.

(e) Number of test combinations

Being *b* the number of block inputs and *c* the number of cell inputs, Table 3.3 shows how to calculate the number of combinations for each combination set.

Table 3.3: Number of combinations of the combination sets.

combinations	*N*	*P*	*NP*
block	$\frac{b!}{(b-c)!\,c!}$	$\frac{b!}{(b-c)!\,c!}$	$\frac{b!}{(b-c)!\,c!}$
cell	2^c	$c!$	$c!\,2^c$
block and cell	$\frac{b!}{(b-c)!\,c!} \times 2^c$	$\frac{b!}{(b-c)!}$	$\frac{b!}{(b-c)!} \times 2^c$
block and cell (b=c)	$\frac{b!}{(b-c)!\,c!} \times 2^c = 2^c$	$\frac{b!}{(b-c)!} = c!$	$\frac{b!}{(b-c)!} \times 2^c = c! \times 2^c$

One of the biggest problems in the blocks construction arises at this point. There may have too many inputs combinations to testing for each cell, and the exhaustive search for the best combination can become impracticable for a library with cells that have more than 7 inputs.

(f) Stopping criteria for combinations testing

To solve such problem, some stopping criteria have been implemented, besides exhaustive search (which would be up to the user to try). The first one is the number *x* of tests performed for cell inputs combinations and the number *y* of tests performed for block inputs combinations. The *x* cell inputs combinations are tested with all the *y* blocks inputs combinations. The cell is put into the block with its best input order after *xy* tests.

Another method is stopping when the combinations number is increased by one. If there are currently *z* different combinations, when *z+1* is reached tests will stop.

These methods are important because they guarantee the completion of the blocks building task in a short time.

Whenever the desired number of distinct output vectors combinations, i.e. the 2^n different signal combinations, is achieved, a block is complete and the construction of another one starts. Until all cells in library have been used at the first stage of one of the blocks, the generation continues.

(g) Cell usage repetition

The algorithm tries to use cells that have not been used. However, a cell is added to the block, only if its addition creates a new output combination. So, if it does not occur and all the cells not used have already been tested with the possible inputs combinations, the algorithm tries to use a cell already used. Consequently, the last blocks in the structure may have to use in the first stage one or more cells that have already been used before, because they may be needed to complete the number of distinct output vectors required.

(h) Second stage functions

After the first stage is complete in every block, it is necessary to generate the equations which will generate the n-output vector. If we have an n-input block with m cells, there will be 2^n expected output vectors from the first stage, and 2^m-2^n vectors that are not supposed to happen; the first ones are used to recreate the desired signals, while the invalid output vectors are assigned to zero logic value for synthesis purpose.

(i) Output

The output file is the description of the first stage of each combinational block in mapped Verilog and the description of the functions of the second stage of each combinational block in non-mapped Verilog. The functions of the second stage of the blocks are mapped through script with RTLCompiler Cadence tool and ABC Berkeley tool. The Verilog description of all blocks are simulated with ModelSim Mentor tool and converted to SPICE format also through script. The testbench flow is illustrated in Figure 3.2.

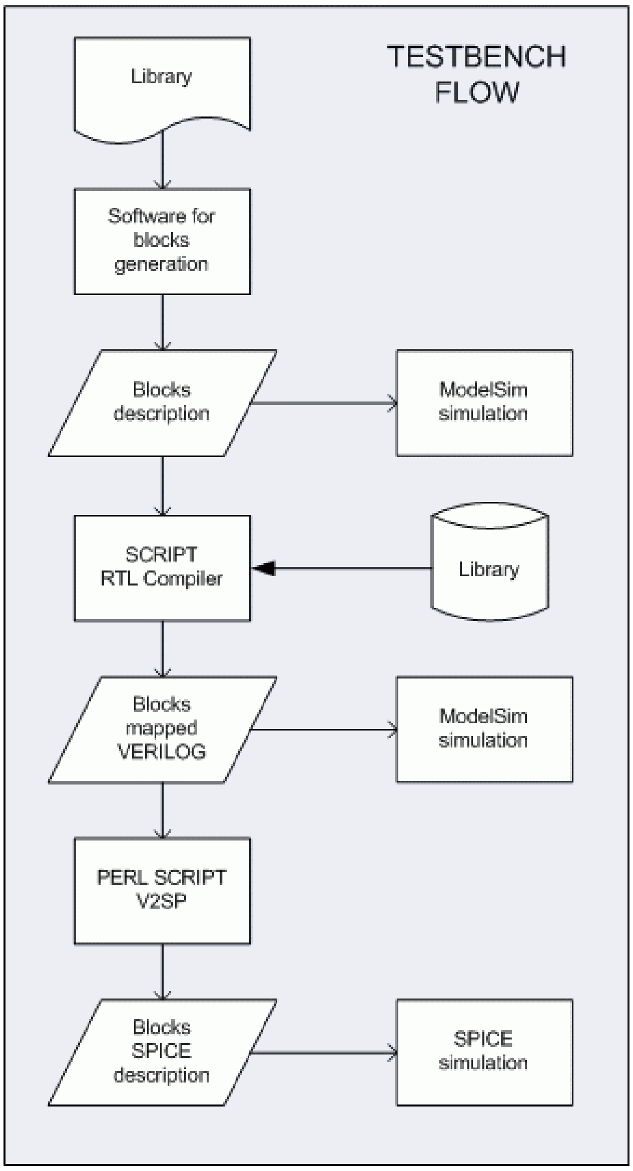


Figure 3.2: Testbench flow.

3.1.2 Sample block

Supposing a block with 3-bit input vector in which it is desired to generate a 3-bit output vector. For that, 2^3 distinct signal combinations must be identified at the outputs of the first stage i.e. it will be necessary at least 3 cells in the first stage of this block.

The cells were searched alternately (first-last). Starting the process, the first cell was picked with its normal inputs (without trying permutations and inversions). When the second cell is picked it is tested with all block inputs combinations and cell inputs combinations, i.e. in this block generation, no stopping criterion for combinations testing was used. Both block and cell inputs combinations were selected randomly. The combination of inputs which generated more distinct output signal combinations will be fixed as the input of that cell. The cells of a sample block are shown in Table 3.4 with their selected block and cell inputs combinations.

Table 3.4: Cells with their selected block and cell inputs combinations.

cell	*cell inputs*	*block input combination*	*cell input combination*	*function*
XNOR2_X2	A_1A_2	IN_1 IN_2	A_1A_2	IN_1 IN_2
NAND2_X1	A_1A_2	IN_1 IN_3	A_2A_1	IN_3 IN_1
XNOR2_X1	A_1A_2	IN_1 IN_3	$!A_1!A_2$	not_IN_1 not_ IN_3
NAND2_X2	A_1A_2	IN_1 IN_3	$A_1!A_2$	IN_1 not_IN_3

Figure 3.3 shows the sample block with the following Verilog description for its first stage:

```
module Block1_block(input[3:1] IN, output[4:1] W);
  wire not_IN[1];
  wire not_IN[3];
  INV_X1 I1 ( .A(IN[1]), .Q(not_IN[1]) );
  INV_X2 I3 ( .A(IN[3]), .Q(not_IN[3]) );
  XNOR2_X2 C1 ( .A1(IN[1]), .A2(IN[2]), .Q(W[1]) );
  NAND2_X1 C2 ( .A1(IN[3]), .A2(IN[1]), .Q(W[2]) );
  XNOR2_X1 C3 ( .A1(not_IN[1]), .A2(not_IN[3]), .Q(W[3]) );
  NAND2_X2 C4 ( .A1(IN[1]), .A2(not_IN[3]), .Q(W[4]) );
Endmodule
```

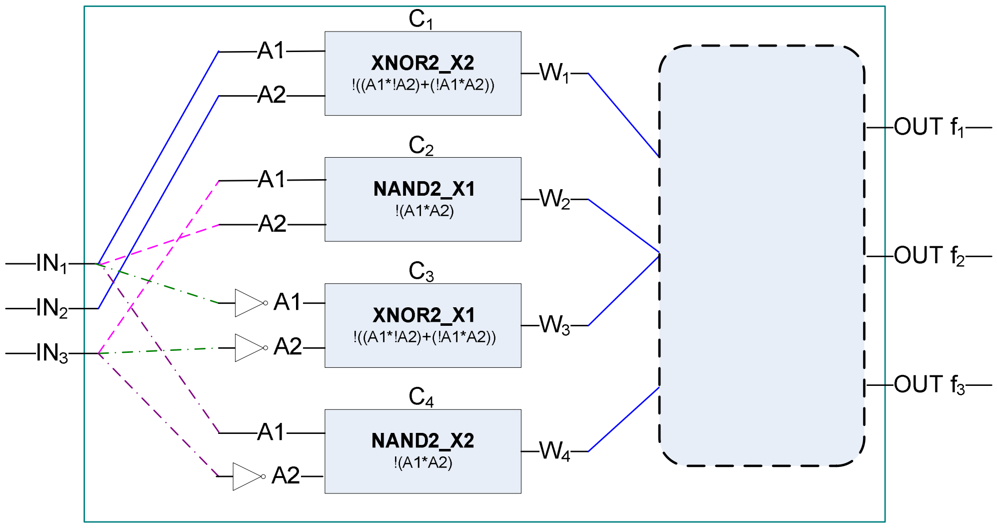


Figure 3.3: Sample block.

The gray vectors in Table 3.5 are repeated combinations. In the generation of this block the tree first cells were selected and fixed, but it was not possible generate the 2^3 output signal combinations. As shown in Table 3.5, 6 distinct combinations were generated. So, the fourth cell was added. Considering this cell C_4 with `A1(IN[2]), A2(not_IN[3])`as inputs, only 7 distinct 4-bit vectors values are identified. Trying a new configuration, C_4 with `A1(IN[1]),A2(not_IN[3])`as inputs, 8 distinct 4-bit vectors values are produced and, at this point, the generation of the first stage of this block is completed. Note that, although C_1 and C_3 (similar to C_2 and C_4) have the same functionality, they are, in fact, different cells due to their distinct drive strengths (indicated by X1 and X2 in Figure 3.3). Some approaches for enrichment of a library work exclusively by adding new drive strengths (DEDOOD, 2003).

Table 3.5: Sample block with 3 cells; **C_4** with inputs `A1(IN[2]),A2(not_IN[3]))` and **C_4** with inputs `A1(IN[1]),A2(not_IN[3]))`.

Inputs			3 cells			C_4 A1(IN[2]),A2(not_IN[3])				C_4 A1(IN[1]),A2(not_IN[3])			
IN_1	IN_2	IN_3	W_1	W_2	W_3	W_1	W_2	W_3	W_4	W_1	W_2	W_3	W_4
0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	0	1	1	0	1	1	1	0	1
0	1	0	0	1	1	0	1	1	0	0	1	1	1
0	1	1	0	1	0	0	1	0	1	0	1	0	1
1	0	0	0	1	0	0	1	0	1	0	1	0	0
1	0	1	0	0	1	0	0	1	1	0	0	1	1
1	1	0	1	1	0	1	1	0	0	1	1	0	0
1	1	1	1	0	1	1	0	1	1	1	0	1	1

3.2 Second stage

The second stage re-creates the block inputs at the output of the whole block from the 2^n distinct intermediate m-bit vectors of the first stage. This synthesis process can be performed using standard synthesis tools. For the example in Figure 3.3 and Table 3.5, it is necessary to generate 3 combinational functions which combined reproduce input vectors applied to the first stage. Although the first stage of this block has 4 output bits, it produces only 2^3 distinct 4-bit vectors. This way, 2^4-2^3 vectors are not supposed to happen as depicted in the white lines of Table 3.6 and 3.7. The valid output vectors of the first stage (gray lines in Table 3.6 and 3.7) are used to recreate the input signals of the first stage and invalid outputs are considered *don't cares*, as shown in Table 3.6. Table 3.7 shows the same information of the Table 3.6, but sorted according with the cells output vectors.

Table 3.6: Block inputs, intermediate vectors and outputs.

IN_1	IN_2	IN_3	W_1	W_2	W_3	W_4	f_1	f_2	f_3
0	0	0	1	1	1	1	0	0	0
0	0	1	1	1	0	1	0	0	1
0	1	0	0	1	1	1	0	1	0
0	1	1	0	1	0	1	0	1	1
1	0	0	0	1	0	0	1	0	0
1	0	1	0	0	1	1	1	0	1
1	1	0	1	1	0	0	1	1	0
1	1	1	1	0	1	1	1	1	1
No vector			0	0	0	0	X	X	X
No vector			0	0	0	1	X	X	X
No vector			0	0	1	0	X	X	X
No vector			0	1	1	0	X	X	X
No vector			1	0	0	0	X	X	X
No vector			1	0	0	1	X	X	X
No vector			1	0	1	0	X	X	X
No vector			1	1	1	0	X	X	X

Table 3.7: Second stage functions.

IN_1	IN_2	IN_3	W_1	W_2	W_3	W_4	f_1	f_2	f_3
No vector			0	0	0	0	X	X	X
No vector			0	0	0	1	X	X	X
No vector			0	0	1	0	X	X	X
1	0	1	0	0	1	1	1	0	1
1	0	0	0	1	0	0	1	0	0
0	1	1	0	1	0	1	0	1	1
No vector			0	1	1	0	X	X	X
0	1	0	0	1	1	1	0	1	0
No vector			1	0	0	0	X	X	X
No vector			1	0	0	1	X	X	X
No vector			1	0	1	0	X	X	X
1	1	1	1	0	1	1	1	1	1
1	1	0	1	1	0	0	1	1	0
0	0	1	1	1	0	1	0	0	1
No vector			1	1	1	0	X	X	X
0	0	0	1	1	1	1	0	0	0

Therefore, considering C_1, C_2, C_3, C_4 as `input[4:1]` and the output of each block (`OUT`) representing a second stage function, it is possible to have the following Verilog description.

```
module Block1_F1(input[4:1] IN, output OUT);
   assign OUT = IN[2]&!IN[3]&!IN[4]|!IN[2]&IN[3]&IN[4];
endmodule
```

```
module Block1_F2(input[4:1] IN, output OUT);
   assign OUT =
IN[2]&(!IN[1]&IN[4]|IN[1]&!IN[4]&!IN[3])|IN[1]&!IN[2]&IN[4]&IN[3];
endmodule

module Block1_F3(input[4:1] IN, output OUT);
   assign OUT = IN[4]&(IN[2]&!IN[3]|!IN[2]&IN[3]);
endmodule
```

Since the invalid output vectors of the first stage are assigned to zero logic value for synthesis purpose, these invalid vectors may produce eventual redundancies in the synthesized second stage. These redundancies may prevent faults affecting the cells of the first stage from being observable at the output of the second stage. A straightforward way to overcome this problem consists in synthesizing the second stage including an additional signal ‘check’ (see Figure 3.4) that identifies at the stage output, all valid output vectors of the first stage. An additional input signal called ‘en_check’ (see Figure 3.4) is included in the combinational block to ensure that the logic generating the signal ‘check’ is able to detect an invalid vector. This signal emulates the occurrence of an invalid vector and exercises the detection capability of the ‘check’ signal. The truth table of the synthesized ‘check’ logic is given in Table 3.8, and is a function of the first stage outputs and the external signal ‘en_check’.

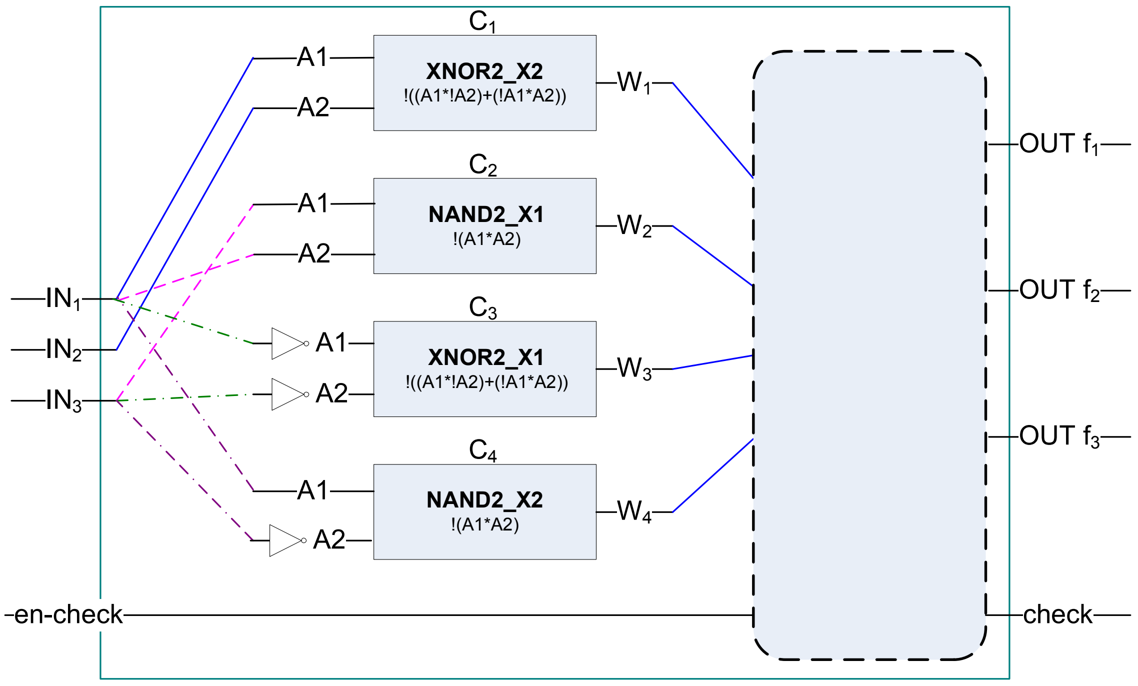


Figure 3.4: Combinational block with ‘check’ signal.

The combinational block, designed as described above, ensures that all single stuck-at faults in the cells of the first stage are detected at the output of the second stage. Additional redundancies may still exist and prevent that some stuck-at faults affecting the second stage are detected at the combinational block output. However, since the full fault coverage in the first stage is just enough to prove the correctness of the set of cells in the library under verification, no other action is taken here to eliminate the remaining redundancies in the second stage.

Table 3.8: Truth table used to synthesize the second stage of a combinational block.

W_1	W_2	W_3	W_4	f_1	f_2	f_3	en-check
0	0	0	0	X	X	X	0
0	0	0	1	X	X	X	0
0	0	1	0	X	X	X	0
0	0	1	1	1	0	1	1
0	1	0	0	1	0	0	1
0	1	0	1	0	1	1	1
0	1	1	0	X	X	X	0
0	1	1	1	0	1	0	1
1	0	0	0	X	X	X	0
1	0	0	1	X	X	X	0
1	0	1	0	X	X	X	0
1	0	1	1	1	1	1	1
1	1	0	0	1	1	0	1
1	1	0	1	0	0	1	1
1	1	1	0	X	X	X	0
1	1	1	1	0	0	0	1

Using a library with 64 combinational cells, eight combinational blocks were automatically generated to build a verification circuit that embedded all library cells in the first stage of blocks. Fault coverage analysis was performed in each block with and without the 'check' signal using Flextest Mentor tool. Results show that redundant faults presented in the first stage of some blocks were eliminated with the addition of the 'check' signal as described in the following reports.

Block2 – first stage

```
CMD> report faults /bBlock2/C1 /bBlock2/C2 /bBlock2/C3 /bBlock2/C4
/bBlock2/C5 /bBlock2/C6 /bBlock2/C7 /bBlock2/C8 /bBlock2/C9 -class
REDUNDANT
  1     RE    /bBlock2/C6/and2/IN0
  1     RE    /bBlock2/C6/or1/OUT
  1     RE    /bBlock2/C6/or1/IN0
  1     RE    /bBlock2/C6/or1/IN1
```

Block2 – first stage (with checker)

```
CMD> report faults /bBlock2/C1 /bBlock2/C2 /bBlock2/C3 /bBlock2/C5
/bBlock2/C4 /bBlock2/C6 /bBlock2/C7 /bBlock2/C8 /bBlock2/C9 -class
REDUNDANT
//  WARNING: No faults reported
```

Block5 – first stage

```
CMD> report faults /bBlock5/C1 /bBlock5/C2 /bBlock5/C3 /bBlock5/C4
/bBlock5/C5 /bBlock5/C6 /bBlock5/C7 /bBlock5/C8 /bBlock5/C9
/bBlock5/C10 -class REDUNDANT
  0     RE    /bBlock5/C2/or3/IN1
```

Block5 – first stage (with checker)

```
CMD> report faults /bBlock5/C1 /bBlock5/C2 /bBlock5/C3 /bBlock5/C4
/bBlock5/C5 /bBlock5/C6 /bBlock5/C7 /bBlock5/C8 /bBlock5/C9
/bBlock5/C10 -class REDUNDANT
//  WARNING: No faults reported
```

Block8 – first stage

```
CMD> report faults /bBlock8/C1 /bBlock8/C2 /bBlock8/C3 /bBlock8/C4
/bBlock8/C5 /bBlock8/C6 /bBlock8/C7 /bBlock8/C8 /bBlock8/C9
/bBlock8/C10 /bBlock8/C11 -class REDUNDANT
  0     RE    /bBlock8/C5/or3/IN0
```

Block8 – first stage (with checker)

```
CMD> report faults /bBlock8/C1 /bBlock8/C2 /bBlock8/C3 /bBlock8/C4
/bBlock8/C5 /bBlock8/C6 /bBlock8/C7 /bBlock8/C8 /bBlock8/C9
/bBlock8/C10 /bBlock8/C11 -class REDUNDANT
//  WARNING: No faults reported
```

3.3 Combinational blocks stage analysis

3.3.1 First stage analysis

Blocks may be constructed using N, P or NP combination sets on the cell input combinations (AGOSTA, 2007). Each of these configurations generates different amount of blocks with distinct sizes.

A library with 64 cells up to 4 inputs was used to test the software with (a) NP, (b) N and (c) P combination sets. The blocks were constructed without restriction, i.e. testing all the block and cell combinations. The tool tries to use each library cell once (new cells), but sometimes it is necessary to use a cell already used (reused cells) in order to complete the last block(s). If a chosen cell does not produce at least one different output signal combination, this cell is discarded (discarded cells) and another is chosen to test.

(a) Cell combinations: NP

Table 3.9 shows the number of NP combinations used to test a cell when the block presents 4 inputs and cells have 2, 3 or 4 inputs.

Table 3.9: Number of combinations tested using NP configuration.

block inputs	cell inputs	Block combs	Cell combs	Total combs
4	2	6	8	48
4	3	4	48	192
4	4	1	384	384

Table 3.10 reports the structure of the first stage of each combinational block generated using NP combinations. It is possible to observe the total cells, new cells, reused cells, discarded cells and build time of each block first stage.

Table 3.10: NP configuration report for the set of 64 cells.

NP					
	total cells	new cells	reused cells	discarded cells	build time (ms)
block1	5	5	0	0	121
block2	9	9	0	0	300
block3	8	8	0	0	150
block4	8	8	0	0	130
block5	10	10	0	0	261
block6	8	8	0	0	120
block7	8	8	0	0	160
block8	11	8	3	0	130
total	67	64	3	0	1372

(b) Cell combinations: N

Table 3.11 shows the number of N combinations used to test a cell when the block presents 4 inputs and cells have 2, 3 or 4 inputs.

Table 3.11: Number of combinations tested using N configuration.

block inputs	cell inputs	Block combs	Cell combs	Total combs
4	2	6	4	24
4	3	4	8	32
4	4	1	16	16

Table 3.12 reports the structure of the first stage of each combinational block generated using N combinations. It is possible to observe the total cells, new cells, reused cells, discarded cells and build time of each block first stage.

Table 3.12: N configuration report for the set of 64 cells.

N					
	total cells	new cells	reused cells	discarded cells	build time (ms)
block1	5	5	0	0	100
block2	8	8	0	0	50
block3	9	9	0	0	40
block4	8	8	0	0	30
block5	10	10	0	0	40
block6	8	8	0	0	30
block7	9	9	0	0	20
block8	10	7	3	0	31
total	67	64	3	0	341

(c) Cell combinations: P

Table 3.13 shows the number of P combinations used to test a cell when the block presents 4 inputs and cells have 2, 3 or 4 inputs.

Table 3.13: Number of combinations tested using P configuration.

block inputs	cell inputs	Block combs	Cell combs	Total combs
4	2	6	2	12
4	3	4	6	24
4	4	1	24	24

Table 3.14 reports the structure of the first stage of each combinational block generated using P combinations. It is possible to observe the total cells, new cells, reused cells, discarded cells and build time of each block first stage.

Table 3.14: P configuration report for the set of 64 cells.

	P				
	total cells	new cells	reused cells	discarded cells	build time (ms)
block1	7	7	0	0	60
block2	11	11	0	4	70
block3	10	10	0	7	80
block4	9	9	0	4	30
block5	12	12	0	4	80
block6	10	7	3	8	20
block7	11	4	7	10	40
block8	11	2	9	5	40
block9	12	1	11	3	30
block10	11	1	10	5	30
total	104	64	40	50	480

Figure 3.5 shows the total number of cells used in the first stage of the combinational blocks using NP, N and P set of combinations.

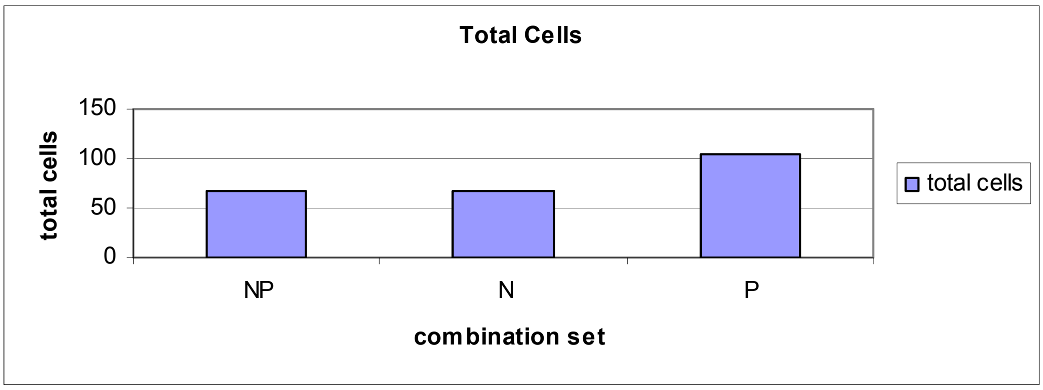


Figure 3.5: Total cells / combination set.

Figure 3.6 shows the number of reused cells in the first stage of the combinational blocks using NP, N and P classes.

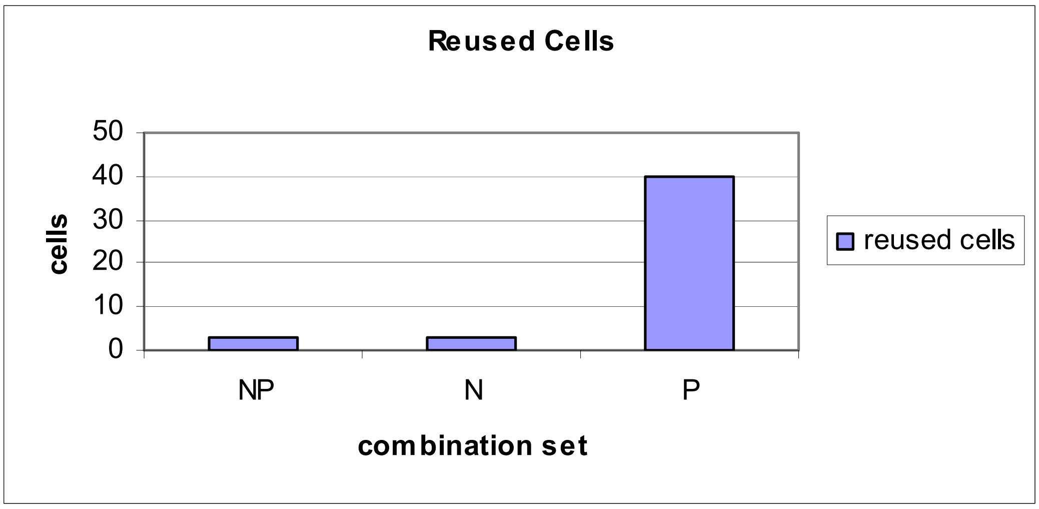


Figure 3.6: Reused cells / combination set.

Figure 3.7 shows the distribution of cells in the first stages of the combinational blocks using NP, N and P set of combinations.

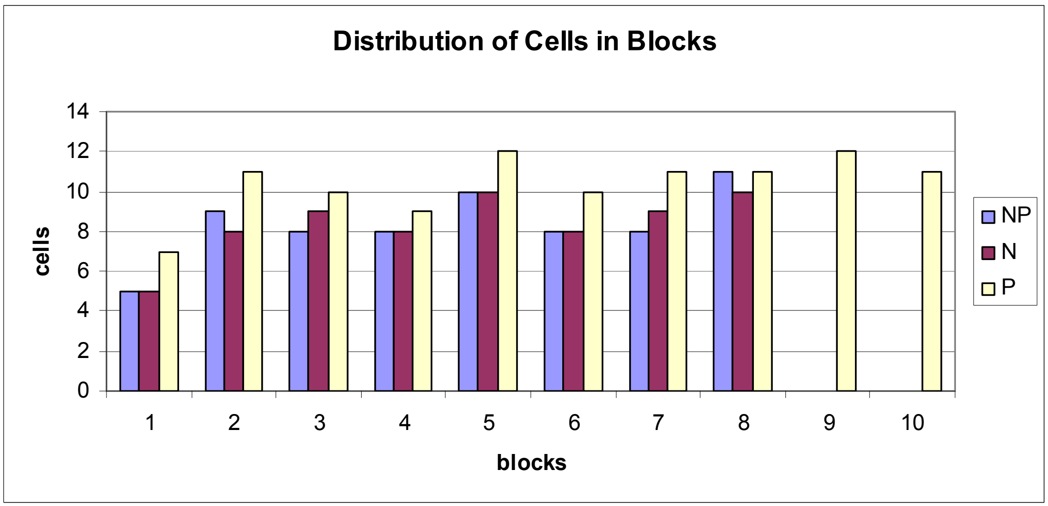


Figure 3.7: Distribution of cells in blocks.

Figure 3.8 shows the build time of the first stages of the combinational blocks using NP, N and P set of combinations.

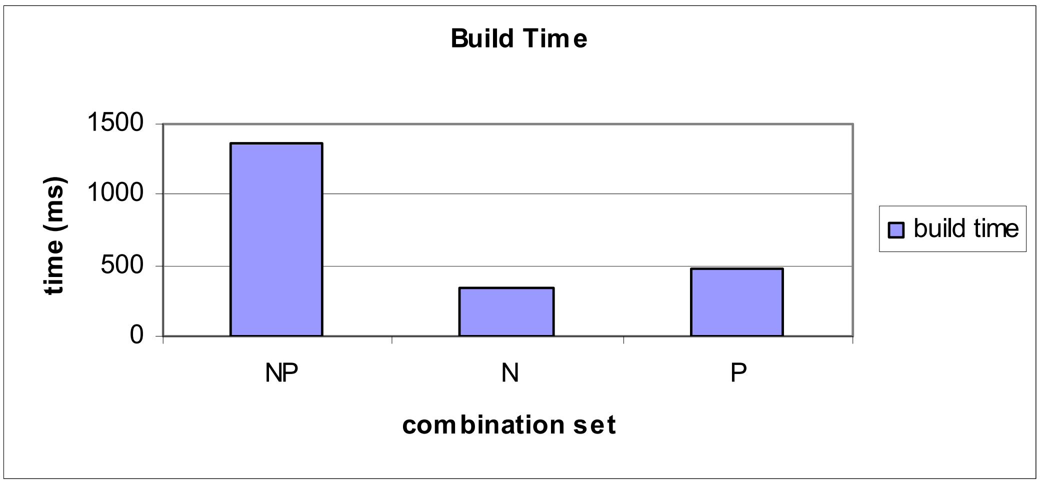


Figure 3.8: Build time / combination set.

Tests show the following results:

- NP set has the biggest number of combinations to test, following by the N set and P set (Tables 3.9, 3.11 and 3.13).

- NP and N set generate similar structures of blocks, while P set uses more cells (Figure 3.5), generates more and bigger blocks (Figure 3.7) and reuses more cells (Figure 3.6) and also discards more tested cells (Table 3.14).

-NP set presents the greatest blocks total build time (Figure 3.8).

Other test was performed using the library 44-6.genlib with 208 cells up to 7 (Figure 3.9), 92 cells up to 6 (Figure 3.10) and 38 cells up to 5 (Figure 3.11) inputs, for all three sets of combinations. Figures 3.9, 3.10 and 3.11 show the number of blocks and the amount of cells used in the blocks. Results show that NP equivalence usually generates more blocks with similar sizes and fewer cells per block than N and P equivalencies.

In Figure 3.12 it is possible to observe that the total cells used in the first stage of the blocks is less sensitive to the combination set used.

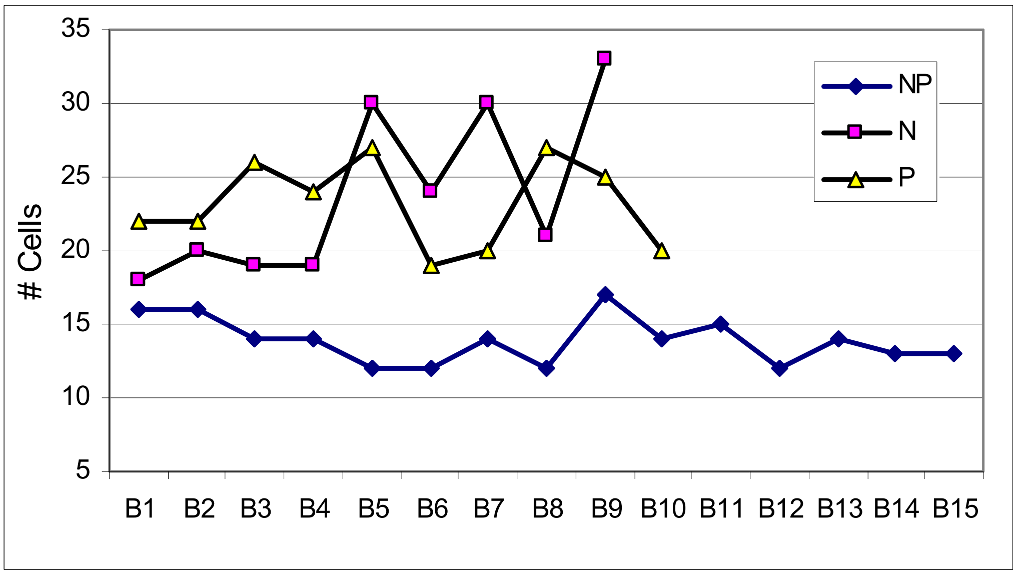


Figure 3.9: Number of cells per block (first stage) for the combination set using the library 44-6.genlib with 208 cells up to 7 inputs.

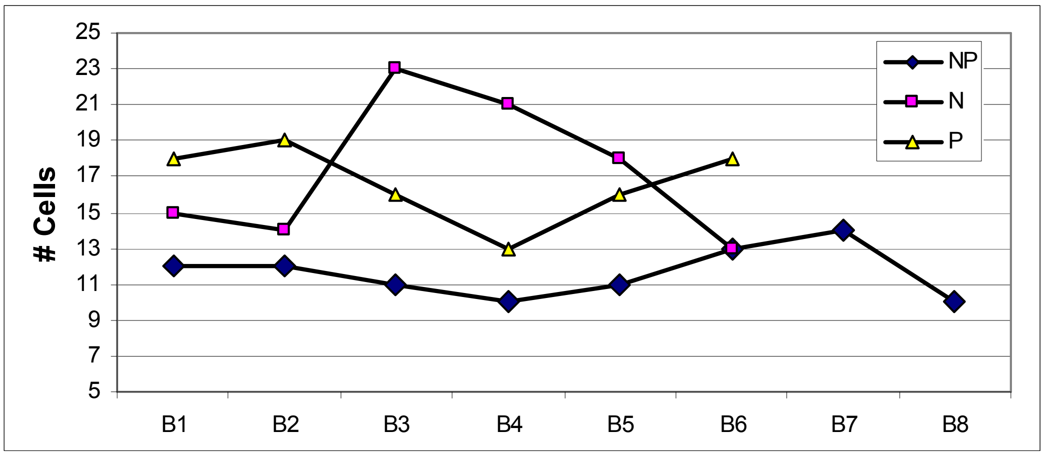


Figure 3.10: Number of cells per block (first stage) for the combination set using the library 44-6.genlib with 92 cells up to 6 inputs.

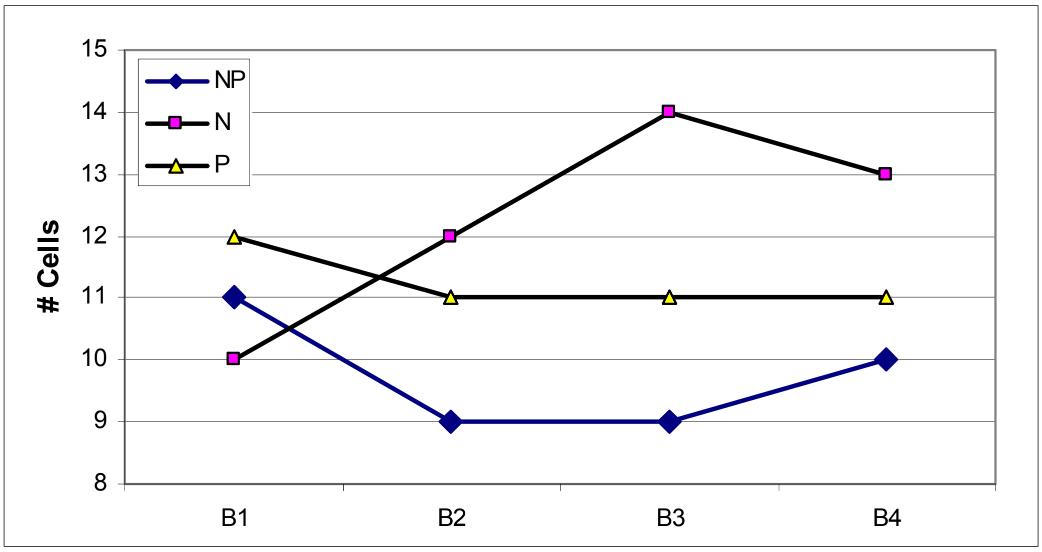


Figure 3.11: Number of cells per block (first stage) for the combination set using the library 44-6.genlib with 38 cells up to 5 inputs.

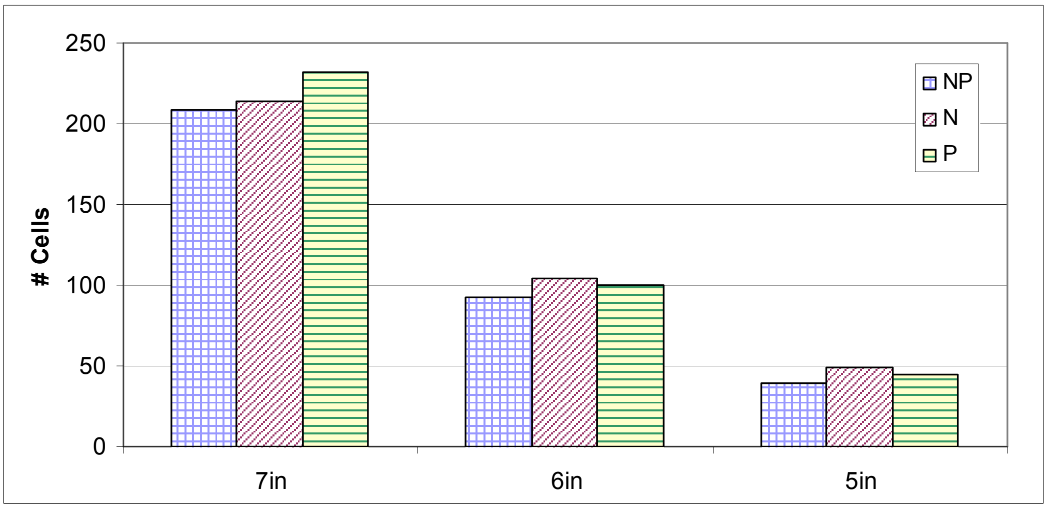


Figure 3.12: Total cells in the first stage considering the combination set for libraries with different maximum number of cell inputs.

3.3.2 Second stage analysis

The maximum number of cell inputs of a given library has a significant impact in the amount of cells required by the proposed approach. The relation between the cells used in the second stage, and the cells used on the first stage is used as a measure of area overhead. Tests have shown that an increase of one cell input duplicates the area overhead. Figure 3.13 reports the area overhead measurements using libraries with different cell input size.

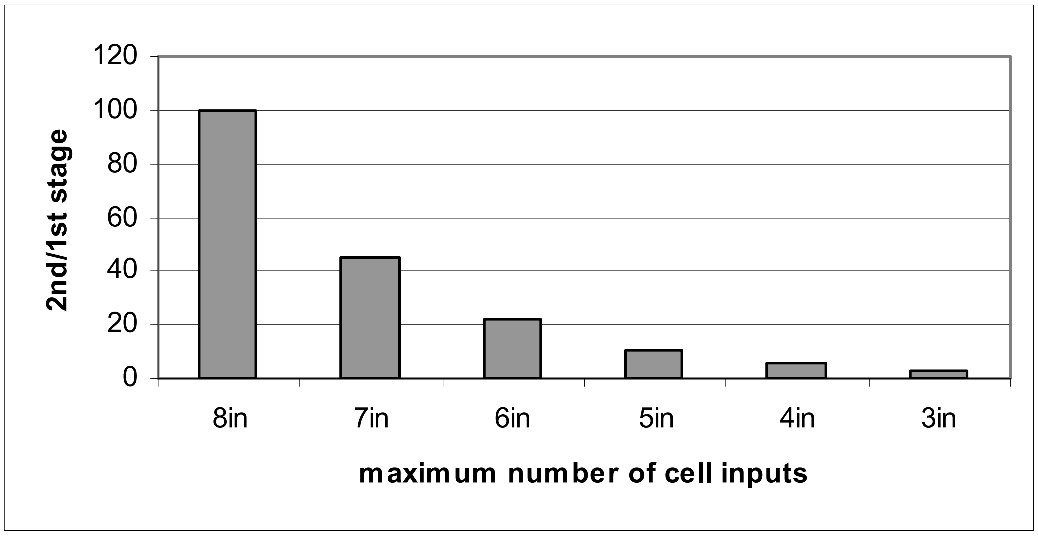


Figure 3.13: Area overhead measurements: 2nd / 1st stage, versus number of cell inputs.

Another test was performed using the library 44-6.genlib with 16 cells up to 4 (Figure 3.14), and 38 cells up to 5 (Figure 3.15) inputs for NP combination set. In this test the intention was to show the total number of cells in the first and second stage when the number of block inputs is increased. In Figures 3.14 and 3.15, it is possible to verify that in each input block increase, there is a significant increase in the number of cells used in the second stage.

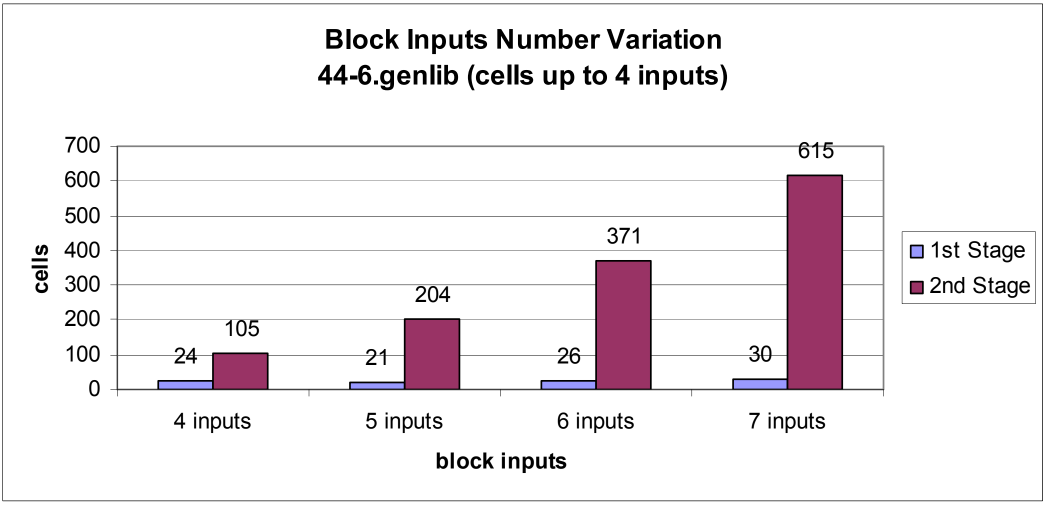


Figure 3.14: Number of cells used in the 1st and 2nd stages for different number of block inputs using library 44-6.genlib (16 cells up to 4 inputs).

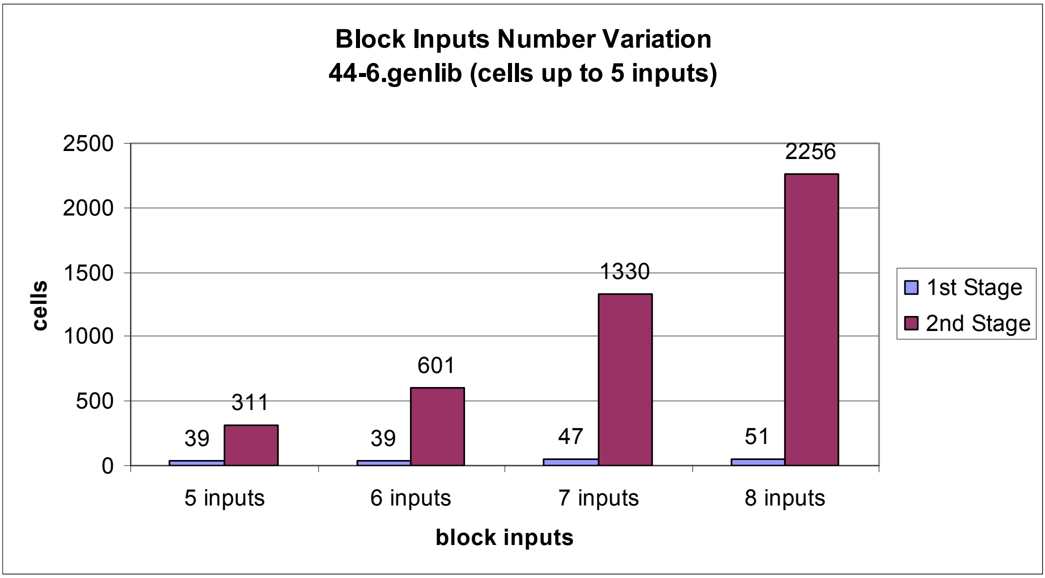


Figure 3.15: Number of cells used in the 1st and 2nd stages for different number of block inputs using library 44-6.genlib (38 cells up to 5 inputs).

Other test was performed using the library 44-6.genlib with 92 cells up to 6 (Figure 3.16), and 208 cells up to 7 (Figure 3.17) inputs, for all three set of combinations. Figures 3.16 and 3.17 show the number of blocks and the amount of cells used in the blocks. Results show that NP equivalence usually generates more blocks with similar sizes and fewer cells per block than N and P equivalencies. Also NP equivalence uses less total cells in second stage than N and P equivalencies. These same results have been obtained in the first stage analysis.

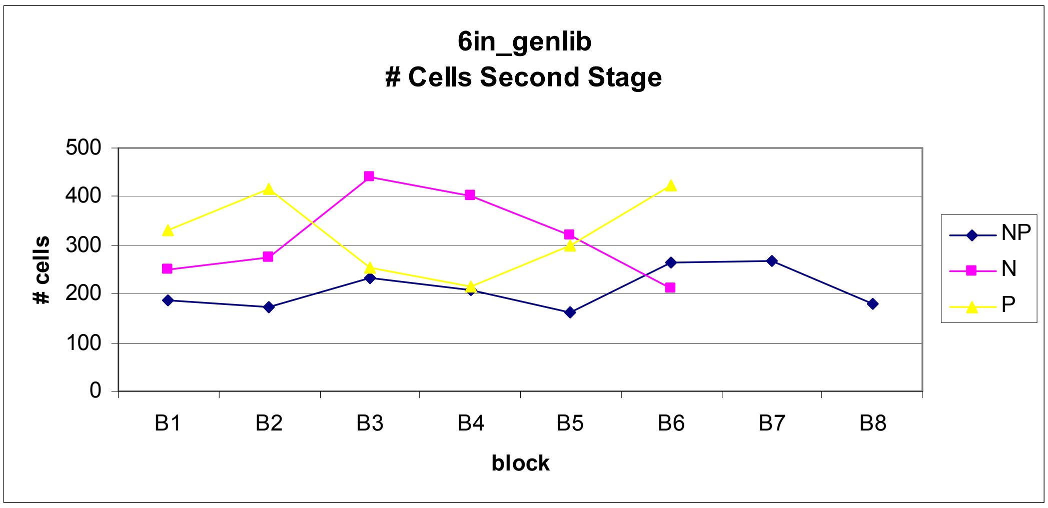


Figure 3.16: Number of cells per block (second stage) for the combination set using the library 44-6.genlib with 92 cells up to 6 inputs.

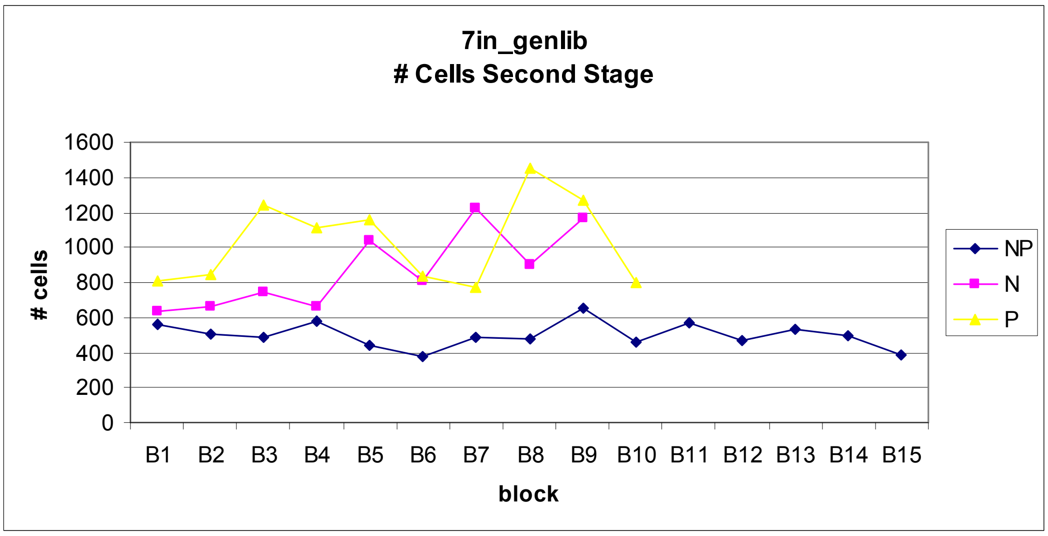


Figure 3.17: Number of cells per block (second stage) for the combination set using the library 44-6.genlib with 208 cells up to 7 inputs.

3.4 Last considerations

Some specifications were used in the construction of the test circuit. For instance, the number of inputs is the same of the block outputs. This specification allows the blocks and also the testbench circuit to present the property of the *reversible logic* (SHENDE, 2002), so they can be called as a reversible circuit. Reversible circuits (gates) have the same number of inputs and outputs and have one-to-one mappings between input and output vectors; thus the vector of the input states can be uniquely reconstructed from the vector of the output states (PERKOWSKI, 2001).

Besides, the sequence of the block output signals should be the same as the block input sequence and consequently, the sequence of the output signals in the last cascaded block should be the same as the input sequence in the first block.

In the second stage of an n-input block with m cells, the 2^m-2^n vectors that are not supposed to happen were assigned to zero logic value for synthesis purpose

Since the combinational blocks, which are the basic structures of the proposed test circuit were described, in the next chapter the test circuit architecture and its operation modes will be presented and discussed.

4 CIRCUIT ARCHITECTURE AND OPERATION MODES

In this chapter the architecture of the test circuit and its operation modes are described. The architecture was described in Verilog and verified with ModelSim Mentor tool and HSPICE Synopsys tool.

As mentioned previously, the test circuit must verify and validate a cell library. In order to do that, the test circuit must also guarantee its correctness, testing itself, i.e., the circuit needs to be auto testable. It also means that the generation of test vectors must be simplified. In order to obtain an auto testable circuit, the faults have to be verified and identified. These faults may be either functional or temporal faults.

4.1 Circuit Architecture

As described in previous chapter, the combinational blocks guarantee the functional validation of the entire set of logic cells since each cell has been placed at least once in the first stage of such blocks, and the 2^n input combinations are applied at n inputs of each block. Once the block outputs reproduce the same input signal, they can be arranged in different ways creating the test circuit architecture. Long paths can be built, for instance, by cascading the combinational blocks in chain configuration using the output signals from one circuit as inputs of the subsequent block. Doing so, the primary chain input values should be observed at the chain output nodes in the case of fault free behavior. Figure 4.1 shows the combinational blocks in chain and its simulation can be observed in Figure 4.2.

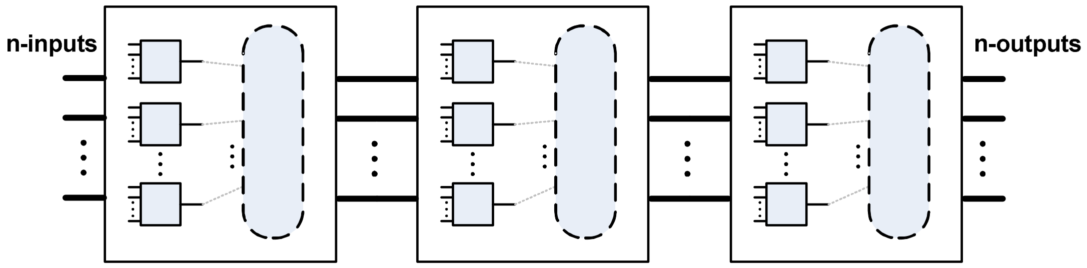
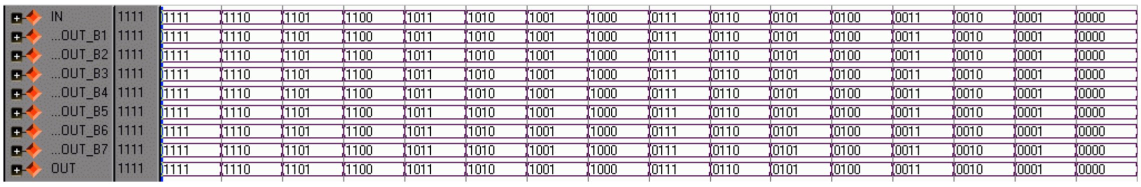


Figure 4.1: Combination blocks in chain.



```
IN        1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
...OUT_B1 1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
...OUT_B2 1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
...OUT_B3 1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
...OUT_B4 1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
...OUT_B5 1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
...OUT_B6 1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
...OUT_B7 1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
OUT       1111  1111 1110 1101 1100 1011 1010 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000
```

Figure 4.2: Simulation of combination blocks in chain.

The ‘check’ signal can also be cascaded with the block signals as shown in Figure 4.3. Once the ‘check’ is able in the beginning of the chain, invalid vectors could be detected.

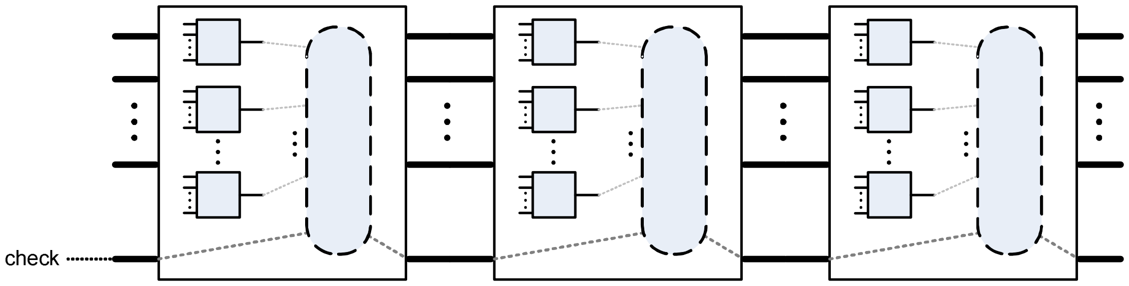


Figure 4.3: Combination blocks in chain with ‘check signal’.

In order to reduce the use of input pins, it means provide a sequence of test vector with minimum external intervention, the signals at the end of the chain are reconnected to primary inputs in a finite state machine (ring) configuration. In this configuration a register barrier, composed by D-type flip-flops, is added to the feedback path to avoid racing. The configuration is represented in Figure 4.4.

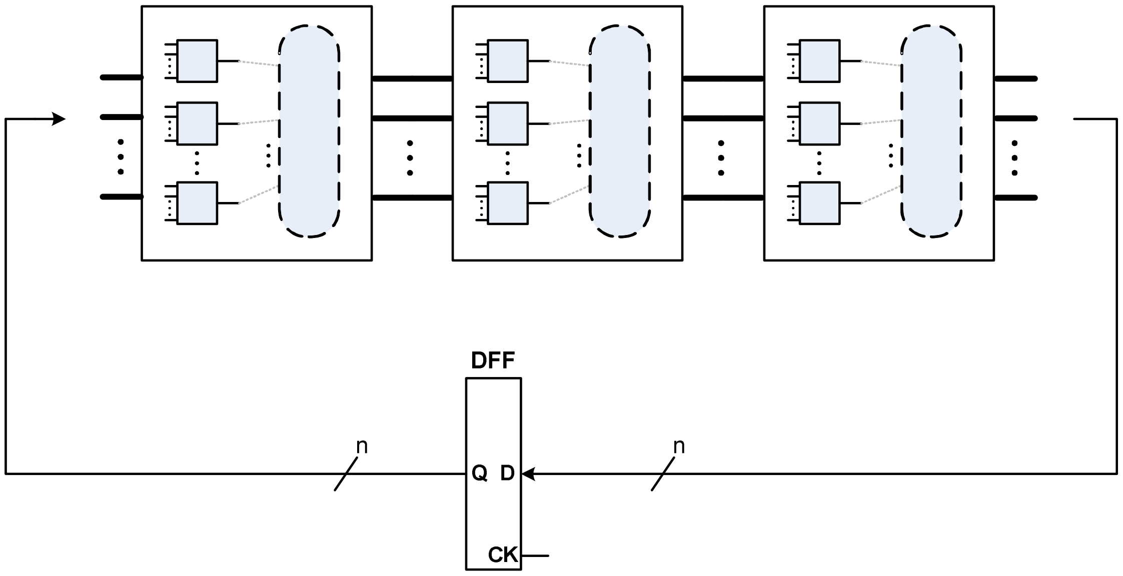


Figure 4.4: Combination blocks with flip-flop barrier.

However, since the feedback signals should be modified to provide the chain input variation, an adder is used to increment the binary vector and make the finite state machine works as a counter as shown in Figure 4.5. It can perform sums by ‘K’ allowing thus other than just a counting 1-by-1 operation, providing different vector transitions that are important to check some charging and discharging conditions in internal nodes intra- and inter-cells.

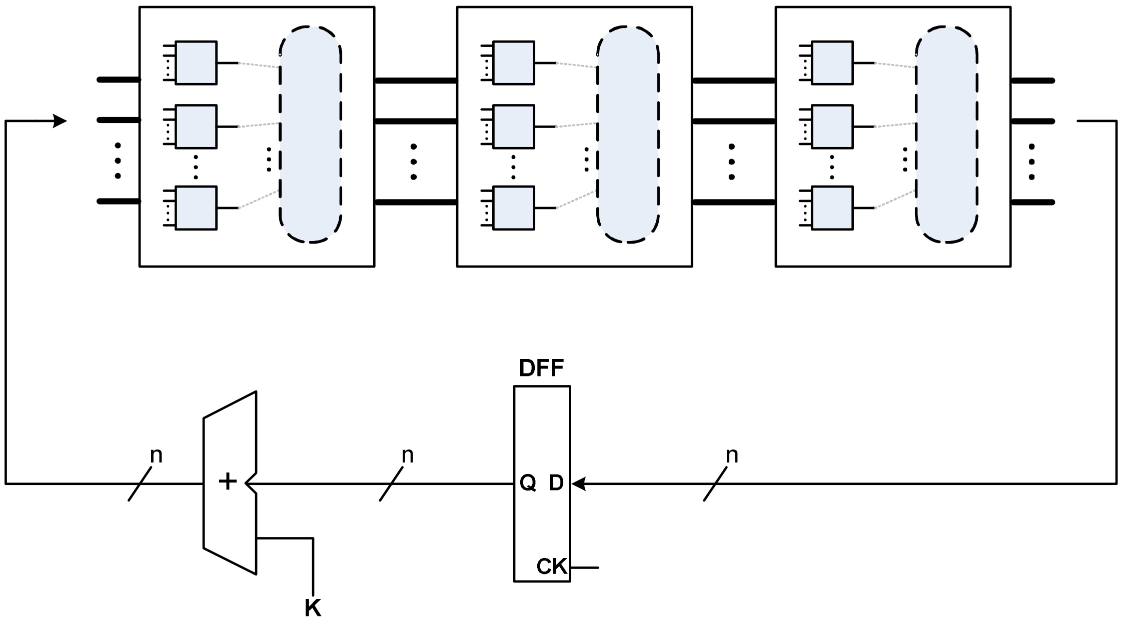


Figure 4.5: Combination blocks in finite state machine configuration.

4.2 Operation Modes

The basic architecture of the testbench circuit is slightly modified to allow different operating modes which provide distinct data evaluation. Notice that the aim of this circuit is to validate the full functionality of the entire set of cells, as well as evaluate the accuracy of the electrical characterization values of each cell (timing and power dissipation data) through the correlation of the static timing analysis (STA) and power analysis with some experimental measures. The following circuit operation modes and their benefits are discussed in this section.

- synchronous mode;
- asynchronous mode;
- oscillation BIST mode;
- diagnosis mode.

4.2.1 Synchronous Mode

In the synchronous mode, the barrier register is controlled by an external clock signal. The adder is used to increment the vector in the ring, acting as a synchronous counter +K. The correct behavior of the counter demonstrates the correct functionality of the combinational blocks and, consequently, of the whole set of cells under testing.

The maximum operation frequency for the circuit, which indicates the worst case path delay, is obtained by decreasing the clock period until the ring counter gives an erroneous result. Such critical path delay is probably different by changing the increment value K. Figure 4.5 shows the synchronous operation mode and its simulation is presented in Figure 4.6.

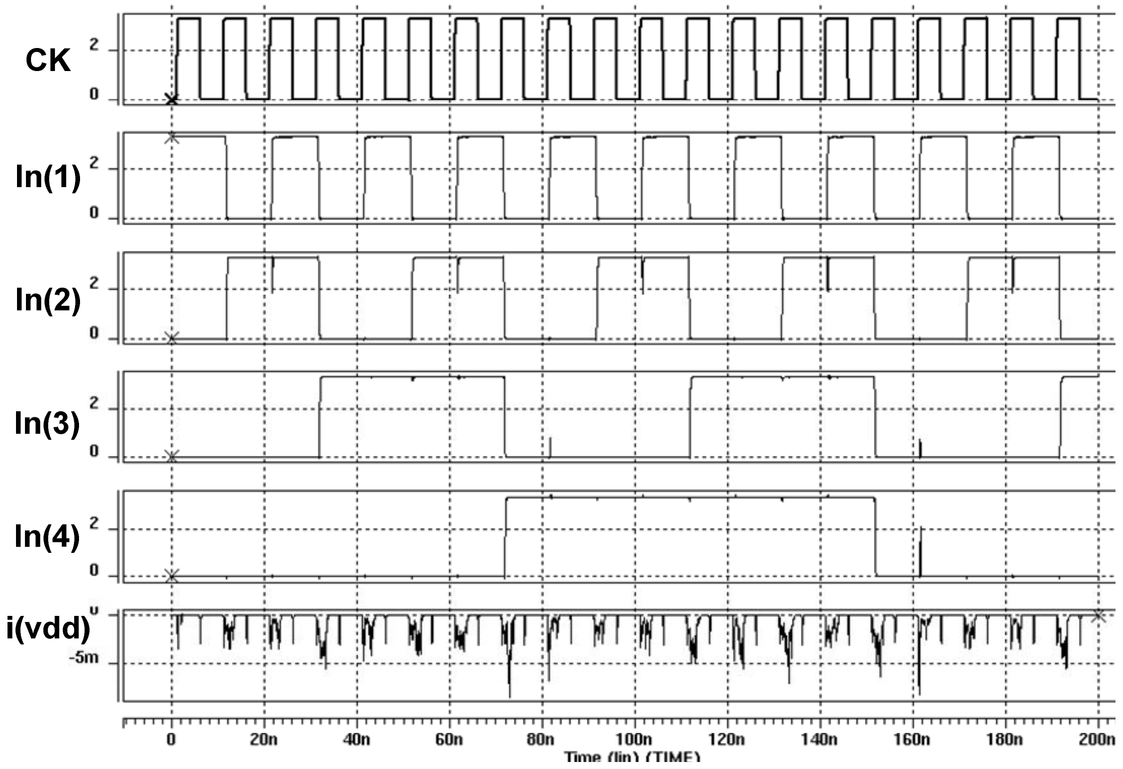


Figure 4.6: Simulation of synchronous mode.

The main contribution of the synchronous mode is the evaluation of the power dissipation, including its dynamic and static components. The external control of the clock signal imposes the frequency operation for switching, and the dynamic power consumption can be related to that. The static power, on the other hand, could be measured at low frequencies as illustrated in Figure 4.7 and simulated in Figure 4.8. At each new input state in the chain the static consumption should be obtained since such power dissipation component depends strongly on the steady state. The extracted information should then be correlated to design power analysis data, obtained considering the electrical simulation values from the cell characterization process.

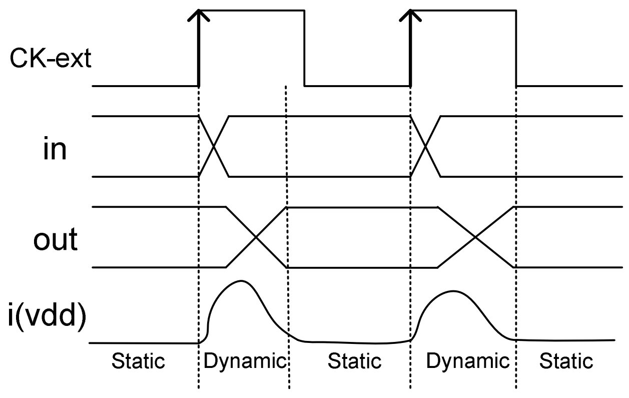


Figure 4.7: Power consumption in synchronous mode.

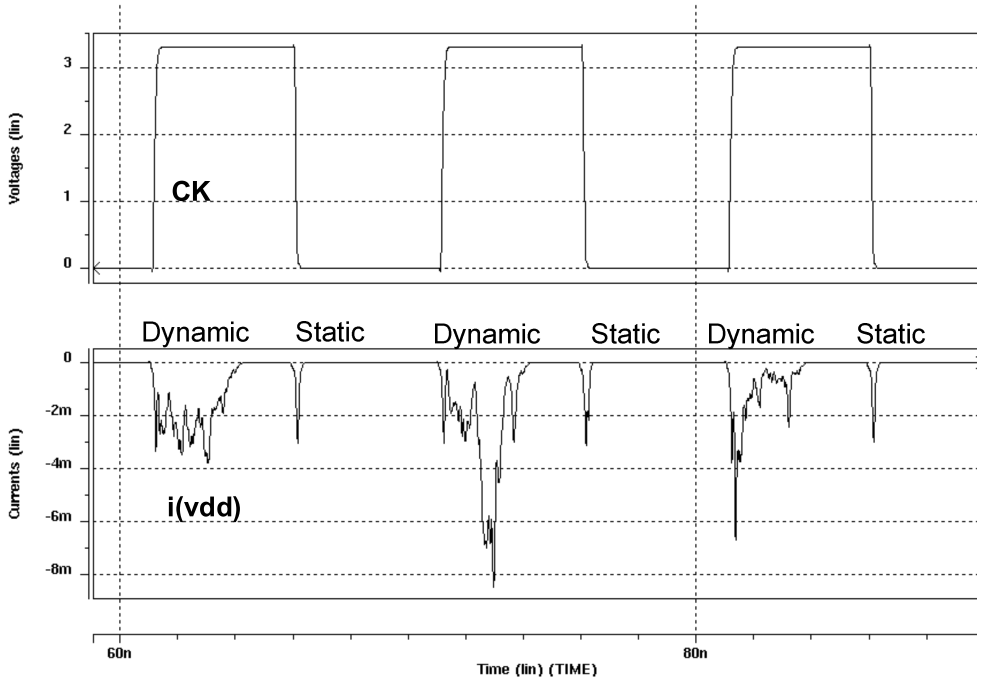


Figure 4.8: Power consumption in the synchronous mode simulation.

4.2.2 Asynchronous Mode

In this architecture the concept of self-timed ring (WILLIAMS, 1992) is applied. If an application uses a pipeline to solve an iterative problem requiring a sequence of internal computation operations, then the pipeline can take input from its own output. A looped self-timed pipeline forms a self-timed ring. If a given problem is fully specified by its initial operands, then the ring's performance is not limited by a need for additional external data inputs during its iterations. Since the ring is composed of self-timed pipeline stages, which communicate locally, its performance is also independent of an external clock.

Thus, in the asynchronous mode, or self-timed ring, the clock signal in the barrier register is provided by the comparator that checks whether or not the input vector In(n..1) has already propagated to the end of the chain Out(n..1). When the same values applied to the circuit inputs get to the end of the chain, the comparator switches from '0' to '1', clocking the register. The new data is retrieved in the register and passed to the adder. The adder increments the register output and applies the new vector to the chain. At this moment, since In(n..1) is no longer equals to Out(n..1), the comparator output is back to '0' and remains at this state until the new vector propagates through the whole chain of combinational blocks. Figure 4.9 shows the asynchronous operation mode. In Figure 4.10 it is possible to observe when the Ctrl_CK changes from '0' to '1', i.e. from the synchronous to the asynchronous mode.

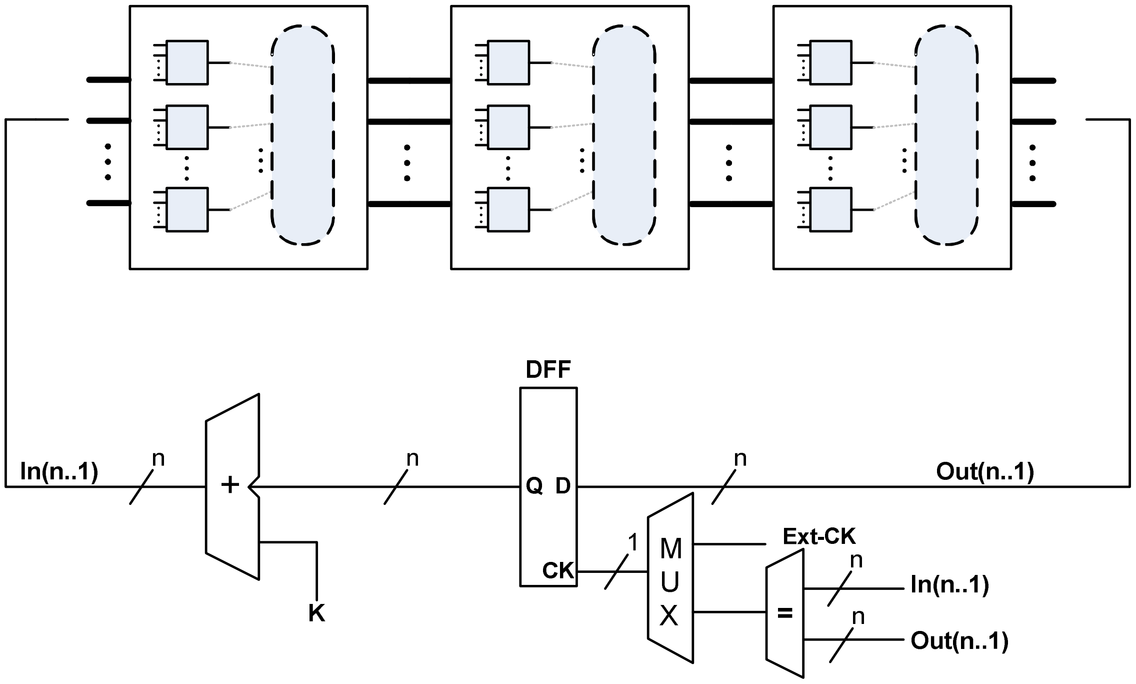


Figure 4.9: Asynchronous mode.

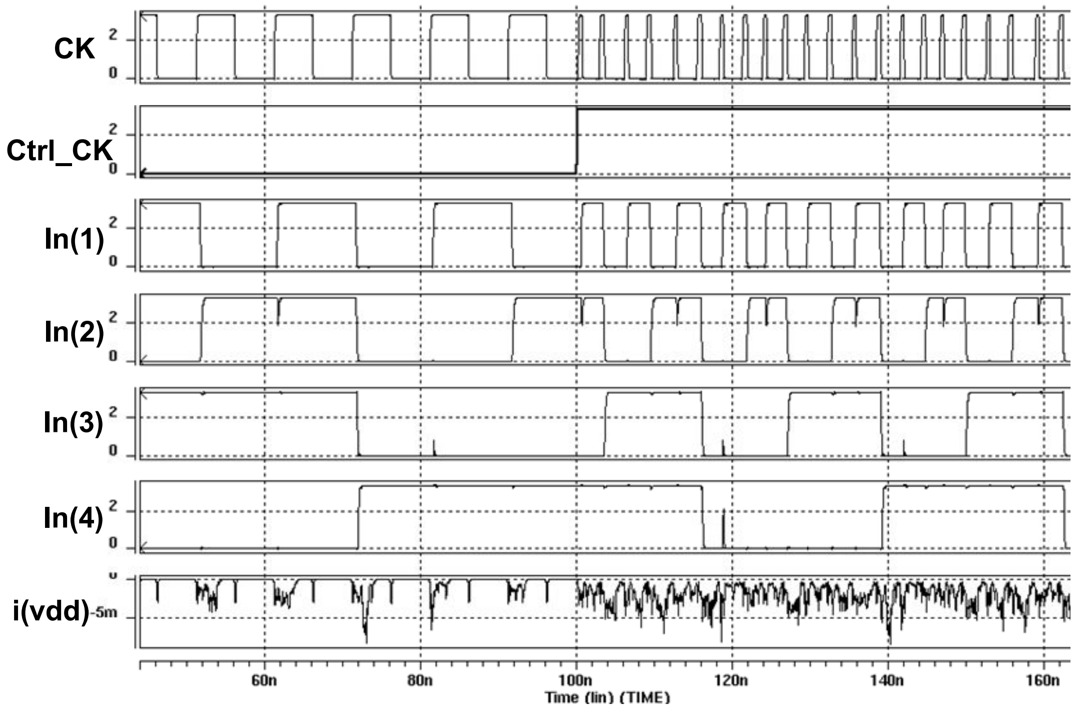


Figure 4.10: Simulation of synchronous and asynchronous mode.

At this operation mode, just an input signal transition is enough to start the self-timed counting, and the right operation keeps the circuit running. It means, for instance, the starting of the self-timed operation is obtained by modifying an external adder input bit ‘Ki’. In case an error occurs, the data at the end of the chain will not be equal to the data at the circuit inputs, the comparator will not switch to ‘1’ and the self-timed execution will stop.

The ‘check’ signal from the last block can also be considered in the comparator evaluation. If an error is detected through the chain, ‘check’ is set to ‘0’, propagates and disables the comparator, freezing the verification circuit in the erroneous state. The self-checking property of the asynchronous mode makes it appropriate for functional cell verification. Clock signal monitoring is used to check the right logic operation. Figure 4.11 illustrates the simulation of a fault occurrence during the asynchronous mode, demonstrating that the self-timed counting stops when the right signal vector does not propagate through the combinational block chain.

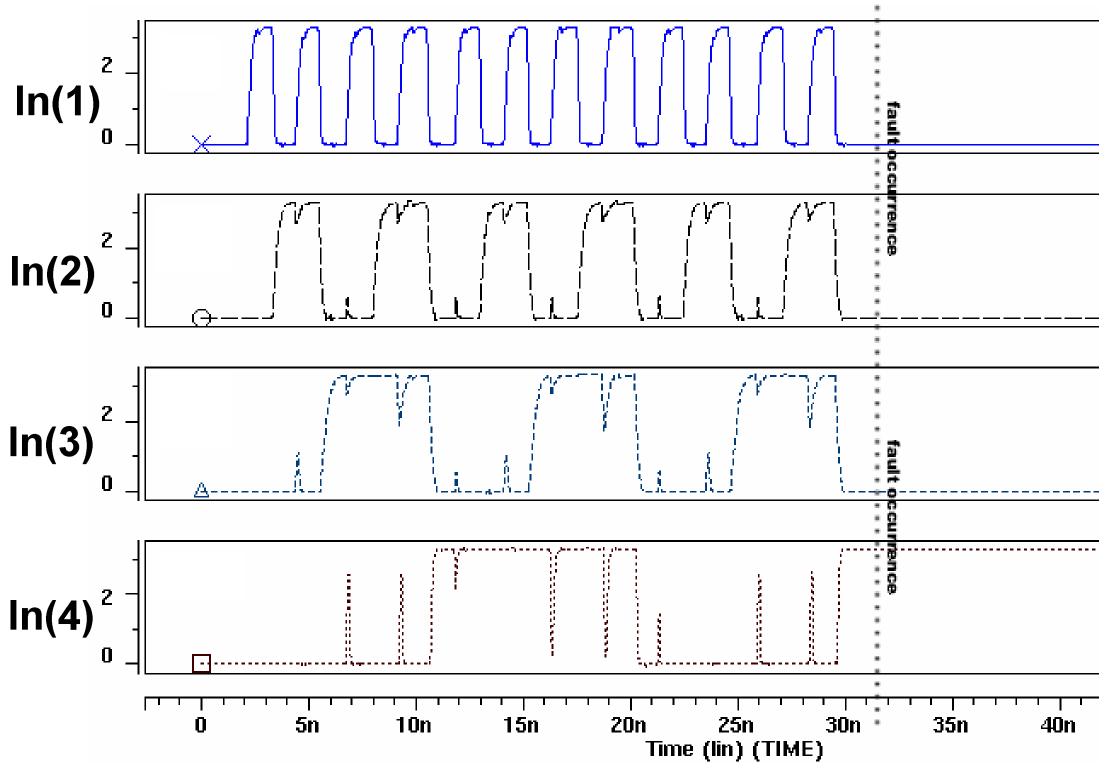


Figure 4.11: Electrical simulation of fault occurrence during the self-timed counting (asynchronous mode).

Timing information can be extracted from complete counting +K, which means the average logic path delay since in a self-timed circuit the speed is as fast as possible according to the delay of each transition, or the time to finish a calculation. This mode is not appropriated to analyze power consumption since it is quite impossible to control the counter state transition since it is not synchronized by a clock signal.

Furthermore, since the calculation is continuous in time, i.e. once a calculation is finished the new one starts immediately, it is not possible to observe the static dissipation. On the other hand, the dynamic power consumption can be related to the complete counter computation but not related to a given frequency like in the synchronous mode as illustrated in Figure 4.12 and simulated in Figure 4.13.

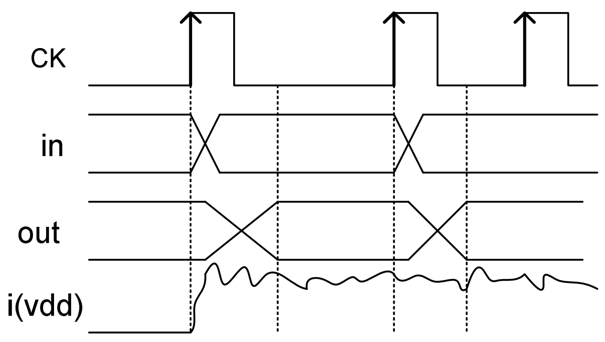


Figure 4.12: Power consumption in asynchronous mode.

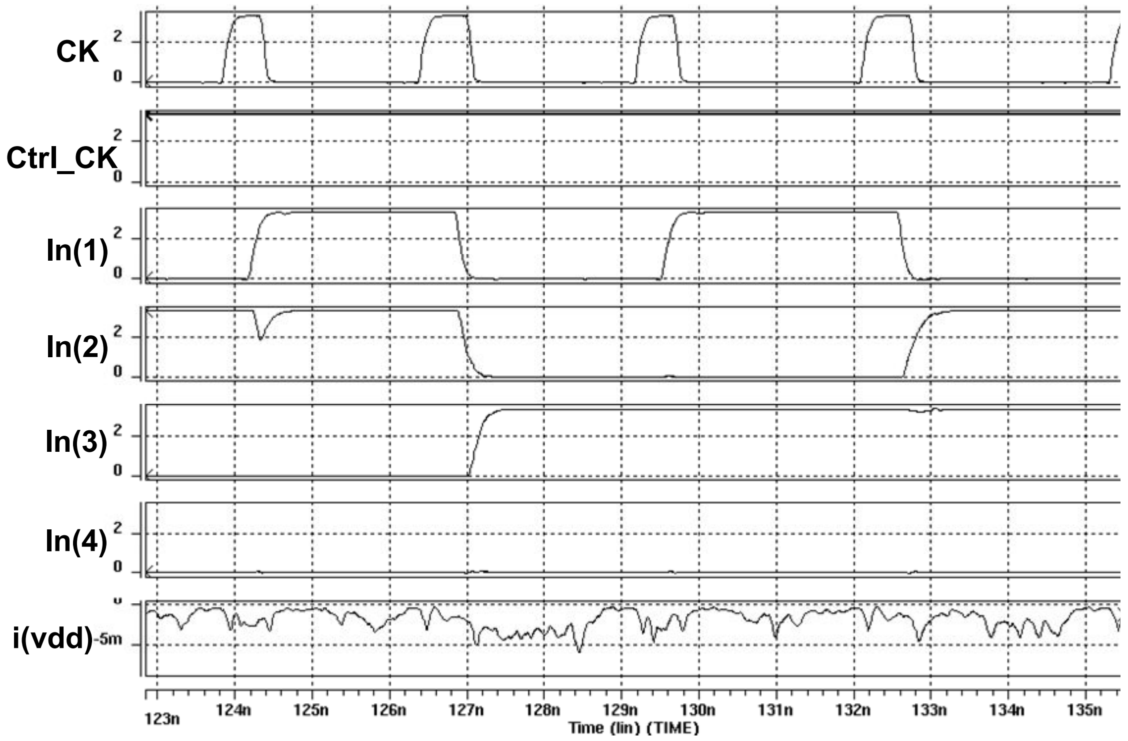


Figure 4.13: Power consumption in asynchronous mode simulation.

Notice that a set of circuits working continuously in this mode can be used to verify library reliability and robustness against degradations coming from continuous utilization caused by effects like electromigration (CHRISTIANSEN, 2006), NBTI (KUMAR, 2007), or other causes. NBTI is a degradation of PMOS transistors that depends on the amount of time the output of the cell is connected to Vdd. These circuits after being continuously stressed for a period of timing can be used, even if the circuit does not fail, to test performance degradation caused by NBTI.

4.2.3 Digital OBIST Mode

In high-complexity mixed-signal circuits, there are severe restrictions on accessibility to the various parts of the CUT and also a serious limitation on the number of test pins. Several design-for-testability techniques have been developed to increase observability and controllability, and to minimize the number of test pins. The alternative approach is the BIST (Built-in Self-Test) in which control, test stimulus generator and measurement circuitry are placed on the same chip and should be able to present a binary pass or fail result.

In (ARABI, 1997) a vectorless digital-output BIST technique for mixed-signal integrated circuits based on the oscillation-test methodology (TONER, 1993) (ARABI, 1995) (ARABI, 1996) is proposed. The oscillation-test method for analog and mixed-signal circuits consists of first partitioning the complex analog circuit into functional building blocks such as amplifier, op-amp, comparator, filter, and data converter and then converting each building block to an oscillating circuit. Faults in the initial circuit related to components (or specifications) that are involved in the oscillator structure manifest themselves as a deviation of the oscillation frequency. Therefore, the loss or the deviation of the oscillation frequency from its nominal value can be employed to test the initial circuit. This test method has proven to be very efficient in terms of fault coverage and area overhead (ARABI,1998) (ARABI, 1996) (ARABI, 1996a).

In (ARABI, 1998) (ARABI, 1998a), a method and associated circuitry for testing delay and stuck-at faults in digital integrated circuits are presented. The method is an extension of oscillation-test method already proposed for analog and mixed-signal circuits to digital systems. However, there is a significant difference between the application of the oscillation-test to analog and digital circuits due to the different nature of these circuits.

In order to introduce this test concept for digital circuits, consider a well-known digital ring oscillator in which oscillations occur when there is an odd number of inverters in the ring. The oscillation frequency is determined by the sum of 0 to 1 and 1 to 0 propagation delays in the loop. Therefore, the propagation delay through the chain of inverters constructing the oscillator can be obtained by measuring the oscillation frequency. Any fault that affects the propagation delay varies the oscillation frequency. Besides, any stuck-at 0 or 1 fault in the circuit will stop the oscillation frequency. If the circuit under test was a chain of inverters, it will be possible to test the circuit for all stuck-at and path delay faults by simply connecting its output to its input and observing the oscillation frequency.

Considering this special case we introduce the test methodology called Digital oscillation-test which consists of first sensitizing a path in the digital circuit under test and then incorporating it into a ring oscillator to test for delay and stuck-at faults. As the oscillation frequency is determined by the propagation delay through the path, it can be used to measure the path propagation delay. Any faults that can stop the oscillations, such as stuck-at faults in the loop, can be detected by observing the oscillation frequency. This procedure should be exercised for all, or at least critical paths in the circuit. If the target path is inverting, we establish a non-inverting feedback from its output to its input to convert it to an oscillator. For a non-inverting path, an inverting feedback should be established by connecting its output to its input via an inverter. In other words, we should make sure that there is an odd number of inverters in the loop to guarantee oscillations. To sensitize a path in the circuit, off-path inputs of all gates directly involved in the path should be set to non-controlling values by properly setting the primary inputs.

In synchronous or asynchronous mode, the same binary value of an i-index input is expected to re-appear at the corresponding i-index output of the verification circuit. This property is ensured by construction of the combinational blocks. If, while closing the feedback loop, i-output is directly connected to i-input, i-path is kept in steady state. But, if i-output is complemented before connecting to i-input (oscillating bit), a positive feedback will result and i-path will oscillate. According to this principle, in oscillation mode the feedback loop is closed such that at least one of the primary inputs of the chain receives the complement of its previous value, as depicted in Figure 4.14. The MUX used to perform such operation is illustrated in Figure 4.15. The bit inversion necessary to ensure the oscillation condition is implemented through the inverter placed at the input of the multiplexing system in Figure 4.16. If a single input-output pair is considered, just the associated path will oscillate, and all others are kept in steady state.

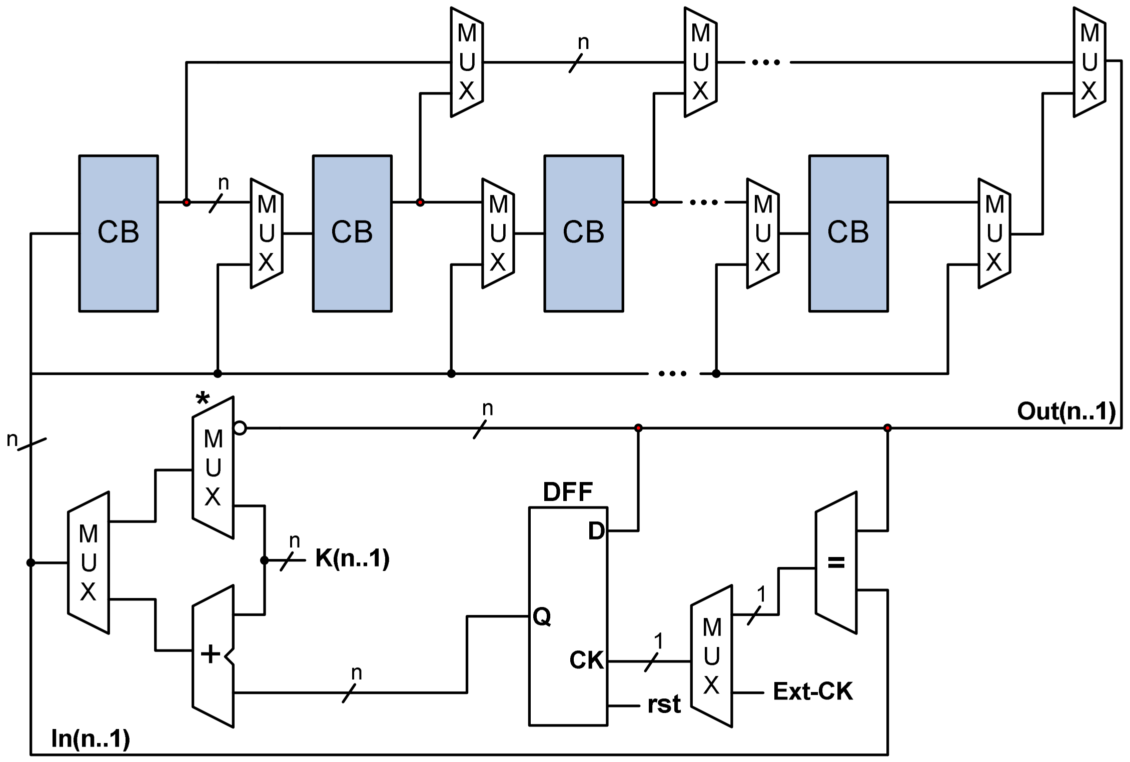


Figure 4.14: Digital OBIST mode.

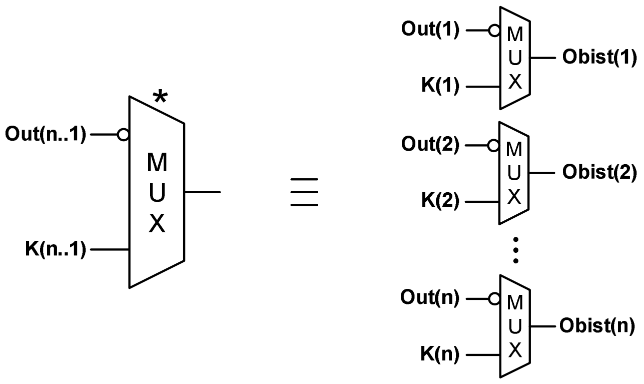


Figure 4.15: Multiplexer for oscillations BIST mode.

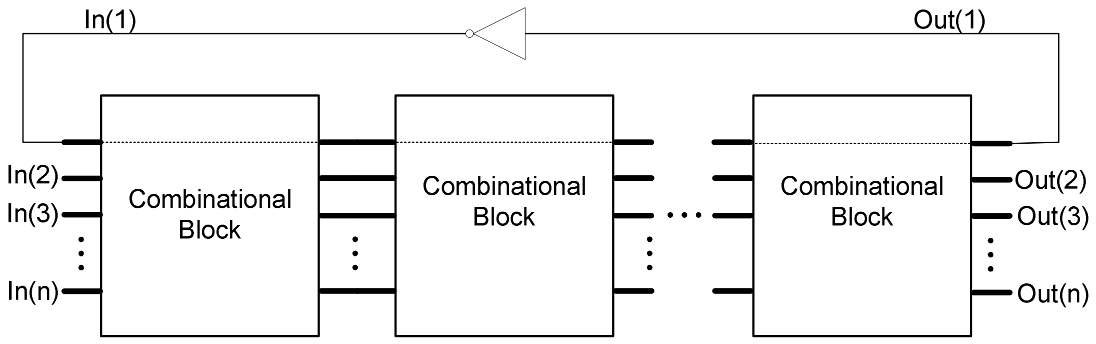


Figure 4.16: Oscillation BIST path.

The main contribution of the oscillation BIST mode to the verification process is that it makes it possible to measure the signal delay propagation through different logical paths involving different library cells. These delay results are then compared to those obtained from a previous static timing analysis, which took into account the cell timing data got from simulations carried out for electrical library characterization. The simulation of the OBIST mode is presented in Figure 4.17.

Individual cell data obtained from the library characterization only make sense if timing and power analysis, performed during the IC design flow, provide good correlation to the actual behavior of fabricated circuits. This correlation may be investigated for different voltages and temperatures (LASBOUYGUES, 2007). Dedicated structures, like ring oscillators and cell chains, can be applied to perform this correlation (HANRIAT, 2001). In this case, however, the number of test structures to guarantee the validation of the entire set of cells becomes unacceptable. The three operation modes described previously are such that timing and power of fabricated circuits can be easily measured and checked against the simulated behavior, also for power supply variation.

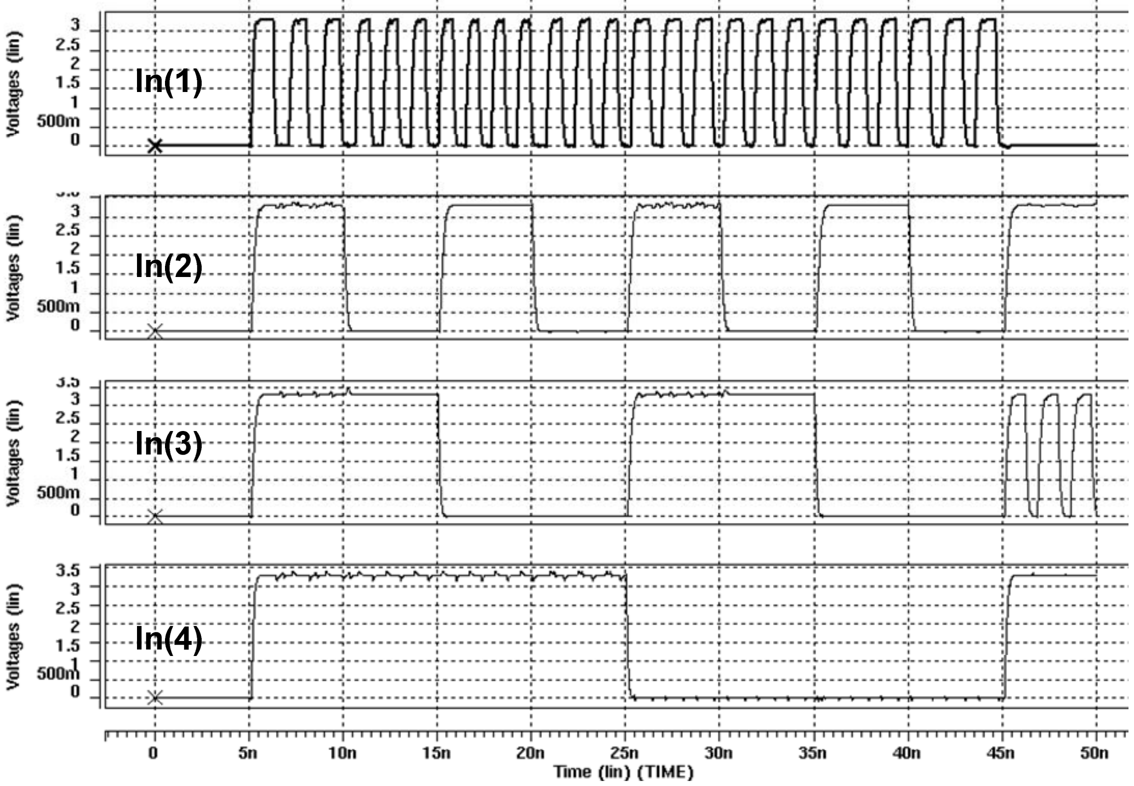


Figure 4.17: Simulation of digital OBIST mode.

Table 4.1 shows the period obtained from a bit oscillation, according to the different state of the fixed inputs and consequently different logic paths.

Table 4.1: Data of the OBIST mode simulation.

Interval (ns)	**($In_{(1)}$,$In_{(2)}$,$In_{(3)}$,$In_{(4)}$)**	**Period (ns)**
5-10	($In_{(1)}$,1,1,1)	1.76
10-15	($In_{(1)}$,0,1,1)	1.47
15-20	($In_{(1)}$,1,0,1)	1.35
20-25	($In_{(1)}$,0,0,1)	1.49
25-30	($In_{(1)}$,1,1,0)	1.65
30-35	($In_{(1)}$,0,1,0)	1.59
35-40	($In_{(1)}$,1,0,0)	1.58
40-45	($In_{(1)}$,0,0,0)	1.75
45-50	(0,1, $In_{(3)}$,1)	1.74

4.2.4 Diagnosis Mode

Additional multiplexers can be included in the circuit architecture, at no significant penalty in area, to select part of the combinational chain. Multiplexers at the input of each combinational block can select the signal from the previous block or directly from the beginning of the chain, removing the influence of the previous blocks in the ring loop. Similarly, multiplexers at the output of the blocks can send the data in the middle of the chain directly to the end of that. Making so, the chain can be easily reduced until a single block or even no one, allowing in this case the verification of the counter and the register barrier without the influence of the combinational chain. This mode, illustrated in Figure 4.18, provides the diagnosis of a faulty block, as well as allows the circuit operation at different modes, described above, running through part of the entire chain.

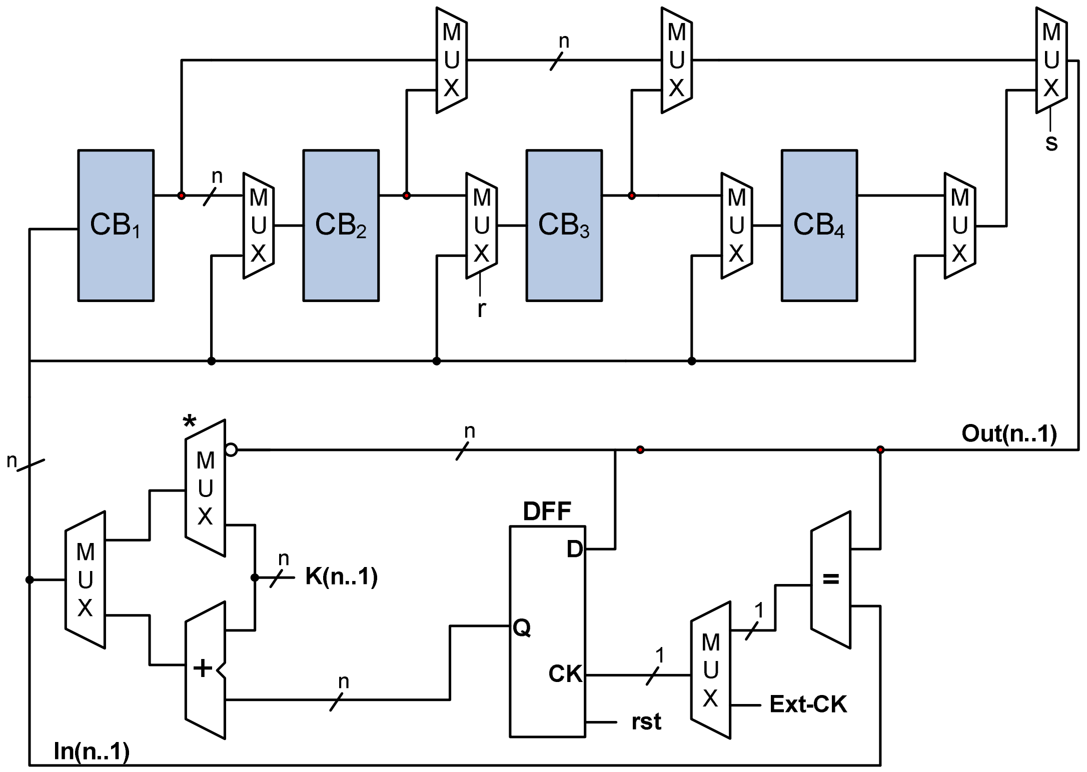


Figure 4.18: Diagnosis mode.

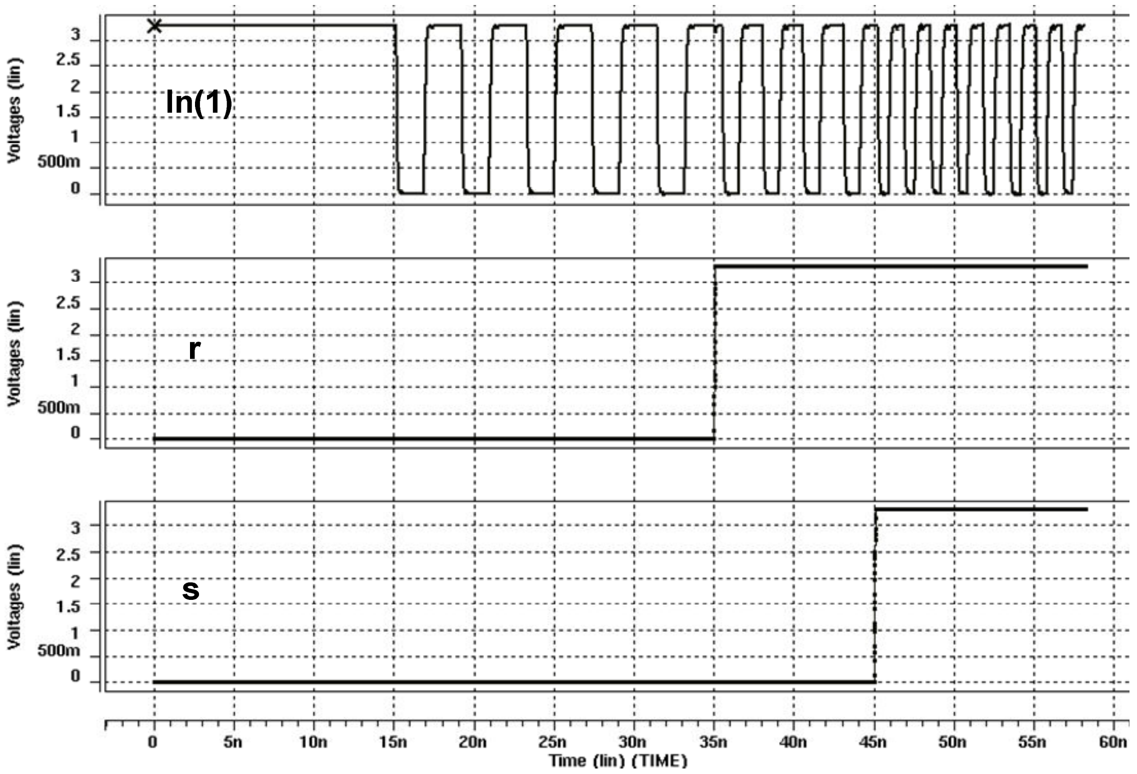


Figure 4.19: Diagnosis mode simulation.

Considering the architecture of the Figure 4.18, Figure 4.19 shows the simulation of the diagnosis mode with ‘In(1)’ as the oscillation bit. When ‘r’ MUX control selects the signal from the beginning of the chain, it removes the two first blocks and the oscillation bit presents a smaller period since the chain have been reduced. The period is even smaller when now the ‘s’ MUX control selects the signal of the CB_3 and the architecture presents only one block. The data of the diagnosis mode simulation can be observed in Table 4.2.

Table 4.2: Data of the diagnosis mode simulation.

Interval (ns)	Active blocks	Period (ns)
15-35	complete chain	4.07
35-45	half-chain	2.52
45-60	one-fourth chain	1.64

The multiplexer used to provide the oscillation BIST mode can also be applied to interrupt the ring configuration, with no additional circuit. This open chain mode allows the stimuli of the combinational blocks chain directly by external signals. It is useful for functional verification as well as for power dissipation data extraction.

4.3 Overhead Analysis

The testbench circuit proposed here can be integrated in the same die with an ASIC, resulting in an area overhead as illustrated in Figure 4.20. Two experiments have been done to evaluate this overhead.

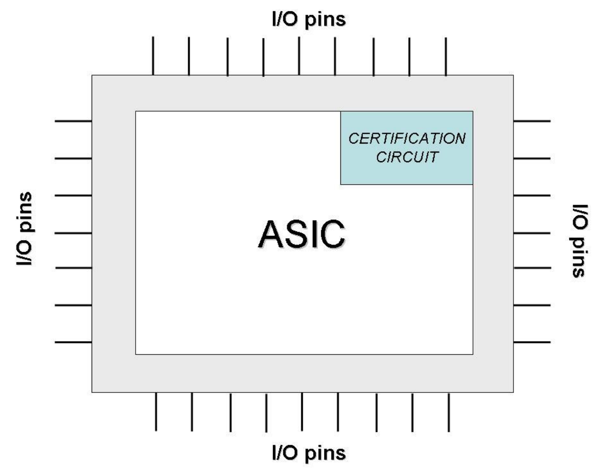


Figure 4.20: Certification circuit for the target ASIC.

In the first experiment, the 64 cells library previously used in this work was targeted. It is composed of cells with at most 4 inputs. The number of input/output nodes for the generation of combinational blocks was kept at four to ensure the instantiation of all library cells. Eight combinational blocks were automatically generated to build a verification circuit that embedded all library cells in the first stage of blocks. In terms of circuit complexity, the whole circuit used around 500 cells of the library to implement the chain of combinational blocks and the additional circuitry to implement the synchronous, asynchronous and oscillation modes of operation. The circuit ensures the verification of all the 64 cells in the library, and it uses an average of 7.03 circuit instances per cell from the library. Note that, in Table I the 'tv80_core' benchmark, that used all 64 cells of the library for its implementation, uses 5,597 cells in total. If the verification circuit generated here was the test vehicle to verify the library used to implement the 'tv80_core' benchmark, an area overhead of 8% logic cells would result. However, if 'wb_conmax_top' was chosen to validate the library, a circuit with 28,089 instances would be produced and yet not all the cells from the library would be instantiated.

The second experiment was produced to test a library containing 208 cells, mainly CMOS complex gates, with up to 7 inputs. This example resulted in a verification circuit composed of around 8,000 instances, in order to verify the complete set of 208 cells. The number of instances per cell library is around 16, which is still acceptable. This is a very small circuit to test such a huge library, considering that a cell based ASIC can easily have more than 100K cell instances.

Therefore, the test circuit brings little silicon overhead to the final design, allowing the fabrication embedded with the target ASIC, as illustrated in Figure 4.20.

The overhead in terms of I/O pins is described in the following, considering n-bit combinational blocks. The value 'n' represents the maximum number of inputs in a single cell to be validated. In this case, the final validation circuit presents:

- 'n' input signals – K(n..1);
- 'n' output signals – Out(n..1);
- one external clock signal;
- one D-type flip-flop reset signal;
- some multiplexer control signals, being that the number can be reduced by using a decoder circuit since some multiplexers are dependent on each others.

Depending on the complexity of the target ASIC, such a pin overhead may prove unaffordable for certification at the chip level. In this case, considering its little silicon overhead, the testbech could be though as a certification vehicle at the wafer-level and be used to validate sets of neighbouring ASICs.

From this analysis, one can conclude that the testbench methodology represents a low cost solution for validating all soft-cells included in an ASIC, and may act as a library certification circuit at chip or wafer-level. In terms of EDA vendor interest, the proposed methodology represents an efficient way to validate soft-libraries by using very compact circuits.

5 CONCLUSIONS

In this work, a test circuit was proposed to validate the full functionality of a set of logic cells, as well as to verify timing and power consumption behaviors which can be correlated with design timing and power estimations in order to validate the cell data provided by electrical characterization.

The methodology to generate such circuit is based in combinational blocks that ensure full logic verification of a set of cells. The proposed architecture consists in building combinational blocks that use all cells from the target library and receive an input bus, where all signal combinations are provided, producing a sequence of output vectors also presenting all possible signal combinations, to be then applied to the next combinational block. Each combinational block is built using two cascaded stages. The first stage is built with instances of cells placed in one-logic level exercised by all possible input combinations. The output signals of verified cells are used as inputs to the second stage, which regenerates the primary inputs. The second stage is composed of synthesized functions with additional cells instances in a multi-logic level.

A specific CAD tool was developed in Java in order to automate the generation of combinational blocks. The software presents parameters to configure the cell list sorting criterion, the number of input/output nodes of the block and the enabling/disabling of the NP equivalences for the cells in the first stage of the blocks. Tests were performed with different tool configurations in order to analyze the blocks characteristics using such configurations.

The tool output files are the descriptions of the first stage of each combinational block in mapped Verilog and the description of the functions of the second stage of each combinational block in non-mapped Verilog. The functions of the second stage of the blocks were mapped through script with RTLCompiler Cadence tool and ABC Berkeley tool. The Verilog description of the blocks was simulated with ModelSim Mentor tool and converted to SPICE format also through script. The basic architecture of the test circuit was slightly modified to allow different operating modes which provided distinct data evaluation using SPICE electric simulations. Also a fault coverage analysis of the blocks was performed through DFT Mentor Graphics tools.

The proposed circuit represents an efficient circuit for on-silicon verification and validation of digital cell libraries. This solution merges well established design and test concepts. The circuit functionality is verified through synchronous/asynchronous ring architecture and by applying the oscillation BIST (Built-In Self-Test) technique for different circuit paths. Also, fault diagnosis can be analyzed with the addition of multiplexers as shown in the diagnosis mode. The test circuit is easily run with few external interventions and the output verification turns out to be quite simple. Besides functional cell evaluation, timing and power consumption can also be extracted for comparisons to high-level estimations performed in the IC design flow, validating the cell electrical characterization data.

Additionally, since this verification circuit brings little silicon overhead to the final design, it can be implemented together with the target ASIC acting as a sort of 'library certification circuit'. Although software-generated cells are the main target of the proposed approach, due to the need of verifying logic gates not taped out before their use in ASIC design, pre-customized standard cell libraries can also benefit from this approach.

In future works, different configurations for the construction of the combinational blocks should be evaluated such as searching, sorting, inputs and outputs tested combinations and so on. Furthermore, different configurations in the architecture should be considered. For instance, the inputs and outputs of the combinational blocks could not be necessarily equal. Also some blocks could be parallelized taking into account the number of block cells inputs.

Diagnosis techniques should be considered in future works for cell error identification, by applying scan test techniques, for example. Another related study is the noise analysis in order to diagnose hazards (spikes) in block signals.

REFERENCES

ABRAMOVICI, M.; BREUER, M.; FRIEDMAN, A. **Digital Systems Testing and Testable Design**. New York, USA: IEEE, 1990.

AGASTEIN, W.; MCFAUL, K.; THEMINS, P. Validating an ASIC Standard Cell Library. In: ANNUAL IEEE ASIC SEMINAR AND EXHIBIT, 3.,1990., Rochester, USA. **Proceedings…** New York, USA: IEEE, 1990. p. 12(6.1)-12(6.5).

AGOSTA, G. et al. A unified approach to canonical form-based Boolean matching. In: DESIGN AUTOMATION CONFERENCE, DAC, 44., 2007, San Diego, USA. **Proceedings**... New York: ACM, 2007. p. 841-846.

ARABI, K. et al. Digital Oscillation-Test Method for Delay and Stuck-at Fault Testing of Digital Circuits. In: INTERNATIONAL TEST CONFERENCE, ITC, 28., 1998, Washington, USA. **Proceedings…** New York: IEEE, 1998. p. 91-100.

ARABI, K. et al. Dynamic digital integrated circuit testing using oscillation-test method. **Electronics Letters**, [S.l.], v. 34, n. 8, Apr. 1998.

ARABI, K.; KAMINSKA, B. Oscillation Built-In Self Test (OBIST) Scheme for Functional and Structural Testing of Analog and Mixed-Signal Integrated Circuits. In: INTERNATIONAL TEST CONFERENCE, ITC, 27., 1997, Washington, USA. **Proceedings…** New York: IEEE, 1997.

ARABI, K.; KAMINSKA, B.; SUNTER, S. Design for testability of integrated operational amplifiers using oscillation-test strategy. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, ICCD, 1996, Austin, USA. **Proceedings**… New York: IEEE, 1996.

ARABI, K.; KAMINSKA, B. Oscillation-test strategy for analog and mixed-signal integrated circuits. In: IEEE VLSI TEST SYMPOSIUM, VTS, 14., 1996, Princeton, USA. **Proceedings…** New York: IEEE, 1996.

ARABI, K.; B. KAMINSKA. **Oscillation-Based Test Strategy (OBTS) for Analog and Mixed-Signal Circuits**. [S.l.: s.n.], Oct. 1995.

BHUSHAN, M. et al. Ring Oscillator Based Technique for Measuring Variability Statistics. In: IEEE INTERNATIONAL CONFERENCE ON MICROELECTRONIC TEST STRUCTURES, ICMTS, 20., 2006, Austin Texas. **Proceedings…** New York: IEEE, 2006.

BUSHNELL, M. L.; AGRAWAL, V. D. **Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits**. Boston: Kluwer Academic, 2000.

CHEN, W.-K. **The VLSI Handbook**. Boca Raton, USA: CRC Press, 2000.

CHONG, J.W.W.; FORBES, R.G. Design of basic CMOS cell library. **IEE Proceedings G Circuits, Devices and Systems**, [S.l.], v. 139, n. 2, p. 256-260, Apr. 1992.

CHRISTIANSEN, C.J. et al. Via-depletion electromigration in copper interconnects. **IEEE Transactions on Device and Materials Reliability**, Piscataway, USA, v. 6, n. 2, p. 163-168, June 2006.

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

DEDOOD, P.; LEE, B.; ALBERS, D. **Optimization of circuit designs using a continuous spectrum of library cells**. [S.l.: s.n.], Sept. 2006. (US Patent 7107551, May 2003)

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

GAVRILOV, S. et al. Library-less Synthesis for Static CMOS Combinational Logic Circuits. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, ICCAD, 1997, San Jose. **Digest of Technical Papers**. New York, USA: IEEE, 1997. p. 658-662.

GIZOPOULOS, D. **Advances in Electronic Testing**: Challenges and Methodologies. Dordrecht: Springer, 2006.

HANRIAT, S.; SCHOELLKOPF, J.P. **Circuit for validating simulation models**. [S.l.: s.n.], June 2001. (US Patent 6253352, July 1998).

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

KAGARIS, D.; HANIOTAKIS, T. A Methodology for Transistor-Efficient Supergate Design. **IEEE Transactions on VLSI Systems**, New York, USA, v.15, n.4, p. 488-492, Apr. 2007.

KUMAR, S.V.; KIM, C.H.; SAPATNEKAR, S.S. NBTI-Aware Synthesis of Digital Circuits. In: DESIGN AUTOMATION CONFERENCE, DAC, 44., 2007, San Diego, USA. **Proceedings**... New York: ACM, 2007. p. 370-375.

LASBOUYGUES, B. et al. Temperature- and Voltage-Aware Timing Analysis. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, USA, v. 26, n. 4, p. 801-815, Apr. 2007.

LIN, C.J.; REDDY, S.M. On Delay Fault Testing in Logic Circuits. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, New York, USA, v. 6, n. 5, Sept. 1987.

LIN, R.B.; CHOU, S.H.; TSAI, C.M. Benchmark Circuits Improve the Quality of a Standard Cell Library. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 4., 1999, Wanchai, Hong Kong. **Proceedings…** New York: IEEE, 1999. p. 173-176.

LONG, S. Test Structures for Propagation Delay Measurements on High-Speed Integrated Circuits. **IEEE Transactions on Electron Devices**, New York, USA, v. 31, n. 8, Aug. 1984.

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, 17., 2007, Lago Maggiore, Italy. **Proceedings…**[S.l.]: ACM, 2007.

MARQUES, F. S. **Mapeamento Tecnológico para Bibliotecas Virtuais Baseado em DAGs**. Proposta de Tese. 2007a. (Doutorado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil.

MARTINEZ, A.A.; BHANDIA, A.S.; LIE, H.H.W. Benchmark Standards for ASIC Technology Evaluation. **Hewlett-Packard Journal**, [S.l.], v. 46, n. 4, Aug. 1995.

MICHELI, G. **Synthesis and Optimization of Digital Circuits**. New York: McGraw-Hill, 1994.

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

PERKOWSKI, M. et al. Regular realization of symmetric functions using reversible logic. In: EUROMICRO SYMPOSIUM ON DIGITAL SYSTEMS, EURO-DSD, 4., 2001, Warsaw, Poland. **Proceedings…**New York: IEEE, 2001. p. 245-252.

RABAEY, J.M.; CHANDRAKASAN. A.; NIKOLIC, B. **Digital Integrated Circuits**: A Design Perspective. 2nd ed. Upper Saddle River: Prentice Hall, 2005.

REIS, A.; REIS, R. Covering strategies for library free technology mapping. In: INTERNATIONAL WORKSHOP LOGIC SYNTHESIS, 1999, Granlibakken, USA. **Workshop Handouts**. [S.l.:s.n.], 1999. p.50-53.

REIS, A.; REIS, R.; ROBERT, M. Topological Parameters for Library Free Technology Mapping. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, SBCCI, 11., 1998, Rio de Janeiro, Brasil. **Proceedings…** New York: IEEE, 1998.

ROY, R.; BHATTACHARYA, D.; BOPPANA, V. Transistor-level optimization of digital designs with flex cells. **IEEE Transactions on Computers**, New York, v. 38, n. 2, p. 53-61, Feb. 2005.

SCHNEIDER, F.R.; RIBAS, R.P.; SAPATNEKAR, S.S. Exact lower bound for the number of switches in series to implement a combinational logic cell. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, ICCD, 23., 2005, San Jose, USA. **Proceedings…** New York: IEEE, 2005. p. 357-362.

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

SECHEN, C. Libraries: lifejacket or straitjacket. In: DESIGN AUTOMATION CONFERENCE, DAC, 40., 2003, Anaheim, USA. **Proceedings**... New York: ACM, 2003. p. 642-643.

SHENDE, V.V. et al. Reversible logic circuit synthesis. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN, ICCAD, 2002, San Jose, USA. **Proceedings…** New York: ACM, 2002.

TONER, M.F.; ROBERTS, G.W. A BIST scheme for an SNR test of a sigma-delta ADC. In: INTERNATIONAL TEST CONFERENCE, 1993, Baltimore, USA. **Proceedings...** New York: IEEE, 1993. p. 805-814.

WANG, L.; WU, C.; WEN, X. **VLSI Test Principles and Architectures**: Design for Testability. Amsterdam: Elsevier, 2006.

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

WILLIAMS, T.E. Analyzing and improving the latency and throughput performance of self-timed pipelines and rings. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1992, San Diego, USA. **Proceedings...** New York: IEEE, 1992.

APPENDIX A CIRCUITO DE TESTE EM SILÍCIO PARA VALIDAÇÃO DE BIBLIOTECAS DE CÉLULAS LÓGICAS GERADAS POR SOFTWARE

Projeto baseado em células é definitivamente a abordagem mais aplicada no mercado de ASIC atualmente. Essa abordagem de projeto implica o reuso de células de biblioteca para construir circuitos digitais mais complexos. Como a etapa do mapeamento tecnológico no fluxo de projeto padrão é baseado nos dados pré-caracterizados de células pré-projetadas, o espaço do projeto de ASIC e a eficiência do mesmo tende a ser limitado pela biblioteca em uso. Quanto mais opções de células e tamanhos disponíveis, maiores são as possibilidades de melhorar o projeto do circuito. O enriquecimento de uma biblioteca pode ser feito apenas adicionando novos tamanhos, ou através da adição de novas funções com implementação série/paralelo padrão, ou mesmo com topologias de transistor especiais. Devido a essa adição de flexibilidade no espaço de projeto, há um aumento no interesse comercial por abordagens que consideram geração de células sob demanda. Essas abordagens são também referenciadas como ‘livre de biblioteca’ ou ‘células geradas por software’. A principal desvantagem da técnica de mapeamento tecnológico livre de biblioteca é o uso de um conjunto de células não validadas em silício. Esse fato torna clientes conservadores relutantes em adotar essa técnica de projeto.

A maneira mais intuitiva de projetar um circuito de teste para o conjunto de células consiste em instanciar todas as células compartilhando as entradas primárias para controlabilidade do teste, enquanto os sinais individuais de saída são multiplexados a fim de reduzir o número de pinos de E/S do circuito, mantendo-se a observabilidade do teste. Essa estratégia fornece a validação funcional completa de cada porta lógica. Contudo, informações de tempo são difíceis de ser obtidas através de um circuito com apenas um nível lógico de profundidade. Adicionalmente, os multiplexadores na saída e eventual buferização para compensar as altas capacitâncias nos nodos de entrada, bem como a estrutura de encapsulamento, tornam essa metodologia impraticável para medir características temporais de uma porta lógica. Esse é um dos motivos para se utilizar osciladores em anel e estruturas oscilantes de teste na caracterização de atraso e teste, respectivamente.

Entretanto, o uso extensivo de osciladores em anel pode resultar em um grande número de instâncias de cada célula a serem validadas, o que seria impraticável para bibliotecas com um grande número de células. Outra desvantagem dessa abordagem de validação de bibliotecas é que o teste de confiabilidade (suficiência de contatos, eletromigração, etc), baseado na contínua aplicação de vetores de entrada e no monitoramento das saídas, requer um equipamento de teste automático para aplicar os vetores e checar os resultados, aumentando o custo do teste. Outra possibilidade para validar uma biblioteca seria projetar um circuito de aplicação cuja funcionalidade pudesse ser facilmente verificada. Contudo, é importante perceber que nem todas as células disponíveis em uma biblioteca são necessariamente usadas pela ferramenta de mapeamento tecnológico para um dado circuito.

Esse trabalho propõe uma metodologia de validação para testar em silício um conjunto de células lógicas geradas automaticamente. Essa metodologia de validação garante a completa funcionalidade das células e fornece dados de atraso e consumo que são úteis para validar e ajustar os modelos de dados das células derivados de caracterização elétrica. Um método para gerar automaticamente o circuito de validação específico para um dado conjunto de células é também proposto.

A solução apresentada combina um projeto estável com conceitos de teste. Um bloco combinacional específico é construído para garantir a cobertura da lógica do conjunto de células a ser validado e fornecer na saída o mesmo vetor aplicado na entrada, permitindo assim, cascatear longas cadeias com esses blocos. O uso de vários blocos permite instanciar todas as células. A arquitetura do circuito é então composta por tal cadeia combinacional em uma configuração de anel, sincronizada por um barreira de registradores. Diferentes modos de operação são usados para extrair dados do circuito.

As descrições detalhadas dos blocos combinacionais, da arquitetura e modos de operação do circuito de teste proposto são apresentadas nas próximas seções.

1. Blocos combinacionais

Os blocos combinacionais são construídos de forma que garantam o completo e correto comportamento lógico de todas as células. A fim de atingir esta meta, cada bloco combinacional é composto por dois sub-blocos ou estágios. O primeiro estágio é constituído pelas células em um único nível de profundidade, todas excitadas pelas entradas primárias do bloco, como ilustrado na Figura 1. Essa células são conectadas às entradas do bloco para garantir o exercício lógico. Os sinais de saída das células do primeiro estágio são então utilizados como entradas para o segundo estágio, o qual recalcula o mesmo vetor aplicado como entrada primária no primeiro estágio. Como resultado, as entradas primárias o primeiro estágio são iguais às saídas primárias do segundo estágio. A interface interna dos dois estágios pode ser vista como um código intermediário com o qual as entradas primárias são traduzidas e então recuperadas.

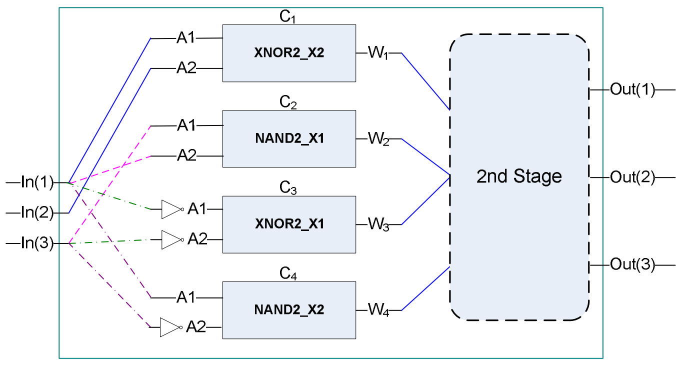


Figura 1 – Exemplo de bloco combinacional.

Antes de detalhar cada estágio dos blocos combinacionais, os seguintes princípios e requerimentos são apresentados:

1) todos os blocos apresentam o mesmo número de nodos de entrada e saída, o qual deve ser igual ou maior do que o maior número de entradas que um célula da biblioteca possui;

2) vetores de entrada e saída possuem o mesmo valor lógico;

3) cada célula deve ser instanciada pelo menos uma vez no primeiro estágio de um bloco combinacional;

4) o número total de blocos combinacionais depende do tamanho do conjunto de células e também da quantidade de células necessárias para compor o primeiro estágio de cada bloco;

5) o segundo estágio dos blocos combinacionais é sintetizado somente com as células da biblioteca a ser validada.

Para gerar o primeiro estágio de um bloco, todo o conjunto de células é inicialmente ordenado de acordo com um dos seguintes critérios: ordem alfabética, número de entradas da célula, quantidade de 0s e 1s fornecidos pela função lógica ou ordem randômica. Tendo as células ordenadas, elas são selecionadas uma a uma para criar o circuito do primeiro estágio. Considerando ‘n’ o número de sinais de entrada e de saída da interface do bloco, a meta é minimizar o tamanho do código intermediário na interface dos dois sub-blocos, sendo que o tamanho do código intermediário corresponde ao número de células usadas no primeiro estágio, se apenas células com uma única saída forem utilizadas. O processo de minimização deve escolher um número de células ‘m’ que produz pelo menos 2^n diferentes valores na saída de ‘m’ bits do primeiro estágio. Os 2^n valores de código intermediário (distintos valores de m bits) da saída do primeiro estágio representam o mínimo necessário pra reconstruir as 2^n combinações de entrada nos ‘n’ sinais de saída do segundo estágio do bloco, com uma correspondência de um para um.

A razão para minimizar o tamanho do código intermediário (representado pelo número de células do primeiro estágio) é devida ao fato de que a complexidade do segundo estágio do bloco é proporcional ao tamanho (número de bits) do código intermediário. Portando, reduzir o número de bits no código intermediário, reduz o acréscimo de área do segundo estágio. Uma forma de realizar esta minimização é a seguinte. A primeira célula selecionada fornece um cógido de comprimento de 1 bit, onde os códigos ‘0’ e ‘1’ são possíveis. A seguir, as células adicionam um bit cada uma e um certo número de vetores distintos. O critério para selecionar novas células é escolhê-las de forma a maximizar o número de diferentes vetores intermediários a cada nova instanciação (que adiciona um bit ao código intermediário). Para aumentar o número de novos vetores no código intermediário, os sinais de entrada de cada célula podem ser permutados (P) e/ou negados (N), e a melhor opção é então selecionada. Células que são incluídas no primeiro estágio de um bloco não serão consideradas na geração dos blocos subseqüentes. Células que não aumentam o número de vetores intermediários diferentes não são instanciadas e permanecem para serem utilizadas na geração do bloco subseqüente. A geração do bloco combinational termina quando todas as células forem utilizadas. Células já utilizadas podem ser reusadas durante a geração do último bloco, já que há poucas células não utilizadas ainda disponíveis. Suponha-se uma biblioteca onde células de apenas uma saída estão disponíveis e onde o circuito de validação é composto pode blocos de n bits (n entradas e n saídas). O número mínimo de células no primeiro estágio é igual a n, uma vez que o código intermediário não pode ter um tamanho menor do que o código de entrada/saída do bloco. O pior caso é composto de no máximo 2^n-1 células, já que a primeira célula fornece dois vetores distintos e as células subseqüentes devem introduzir pelo menos um novo vetor.

O segundo estágio recria os vetores de entrada na saída do bloco a partir dos códigos intermediários representados por W_i. Os sinais de saída do primeiro estágio são usados como entradas do segundo estágio, como mostra a Figura 1 e a Tabela I. Como o tamanho do código intermediário pode ser maior do que o tamanho dos códigos de entrada e saída, algumas combinações W_i nunca ocorrerão. Dessa forma, 'don't cares' são usados para otimizar a síntese do segundo estágio.

Nesse trabalho, a geração do primeiro estágio de cada bloco foi automatizada pelo uso de uma ferramenta de CAD específica, desenvolvida na plataforma Java. O segundo estágio, por sua vez, foi sintetizado com ferramentas padrão de mapeamento tecnológico.

Tabela I – Tabela-verdade utilizada para sintetizar o segundo estágio do bloco combinacional ilustrado na Figura 1.

In(3)	*In(2)*	*In(1)*	W_4	W_3	W_2	W_1	*Out(3)*	*Out(2)*	*Out(1)*
Sem vetor			0	0	0	0	X	X	X
Sem vetor			0	0	0	1	X	X	X
1	0	0	0	0	1	0	1	0	0
1	1	0	0	0	1	1	1	1	0
Sem vetor			0	1	0	0	X	X	X
Sem vetor			0	1	0	1	X	X	X
Sem vetor			0	1	1	0	X	X	X
Sem vetor			0	1	1	1	X	X	X
Sem vetor			1	0	0	0	X	X	X
Sem vetor			1	0	0	1	X	X	X
0	1	1	1	0	1	0	0	1	1
0	0	1	1	0	1	1	0	0	1
1	0	1	1	1	0	0	1	0	1
1	1	1	1	1	0	1	1	1	1
0	1	0	1	1	1	0	0	1	0
0	0	0	1	1	1	1	0	0	0

Utilizando as ferramentas de análise de cobertura de falhas da Mentor Graphics, verificou-se que todas as falhas simples nas células do primeio estágio são observáveis na saída do bloco, através de blocos projetados como descrito acima. Algumas falhas no segundo estágio podem não ser observadas na saída do bloco. Entretanto, já que a completa cobertura de falhas no primeiro estágio é suficiente para provar que o conjunto de células da biblioteca sob verificação estão corretas, não há necessidade de garantir esta propriedade para o segundo estágio.

2. Arquitetura e Modos de Operação

A arquitetura global do circuito é apresentada na Figura 2. A fim de fornecer uma seqüência de vetores de teste com a mínima intervenção externa, os sinais do fim da cadeia são reconectados às entradas primárias. Uma barreira de registradores, composta por flip-flops do tipo D, é adicionada. Um somador é utilizado para incrementar o vetor binário e fazer o circuito atuar como um contador, fornecendo variação na entrada da cadeia. O somador pode realizar somas de K, permitindo outras operações e não só a contagem de 1 em 1. Assim, é possível obter diferentes transições de vetores que são importantes para checar as condições de carga e descarga nos nodos das intra- e inter-células.

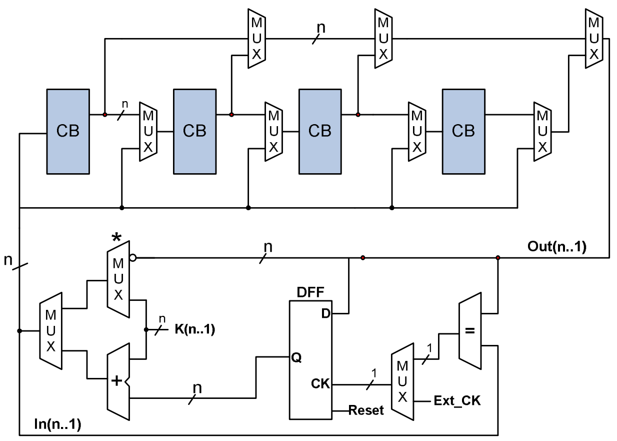


Figura 2 – Diagrama de blocos da arquitetura do circuito proposto.

A arquitetura básica do circuito é ligeiramente modificada, adicionando-se um comparador e multiplexadores para permitir diferentes modos de operação que proporcionam formas distintas de avaliação de dados. Os modos de operação síncrono, assíncrono, BIST oscilante e de diagnóstico são apresentados a seguir.

Modo Síncrono

No modo de operação síncrono, a barreira de registradores é controlada por um sinal de clock externo, o sinal ‘Ext_CK’ indicado na Figura 2. O somador é usado para incrementar o vetor no anel, atuando como um contador ‘+K’ síncrono. O comportamento correto da seqüência de contagem demonstra a correta funcionalidade dos blocos combinacionais e, conseqüentemente, de todo conjunto de células que está sendo testado. A máxima freqüência de operação do circuito, que indica o atraso no caminho crítico, pode ser obtido aumentando-se a freqüência do clock até que o contador em anel apresente um resultado errôneo. Outro benefício significante do modo síncrono é a avaliação da dissipação de energia, incluindo suas componentes, dinâmica e estática. O controle externo do sinal de clock impõe a freqüência de operação e o consumo dinâmico pode ser relacionado a isto. O consumo estático, por sua vez, pode ser medido em baixas freqüências ou mesmo usando um clock externo controlado manualmente. A cada novo estado de entrada na cadeia, o consumo estático pode ser obtido já que tal componente de dissipação de energia depende fortemente do estado estável do circuito.

Modo Assíncrono

No modo assíncrono ou anel auto-temporizado, o sinal de clock dos flip-flops é fornecido pelo comparador que verifica se o vetor de entrada In(n..1) já foi ou não propagado ao final da cadeia Out(n..1) (Figura 2). Quando o mesmo vetor aplicado na entrada do circuito chega ao final da cadeia, o comparador muda de '0' para '1', chaveando o registrador. O novo dado é armazenado no registrador e passado para o somador. O somador incrementa a saída do registrador e aplica o novo vetor na cadeia. Neste momento, como In(n..1) não é mais igual a Out(n..1), a saída do comparador volta para '0' e permanece neste estado até o novo vetor propagar-se através da cadeia completa de blocos combinacionais.

Nesse modo de operação, uma única transição de sinal externa é suficiente para iniciar o contador auto-temporizado e a operação correta mantém o circuito em funcionamento. Por exemplo, o começo da operação do circuito pode ser atingida modificando-se o bit 'Ki' da entrada externa do somador. Se uma célula está com defeito, o dado obtido no final da cadeia não será igual ao dado da entrada da cadeia. Dessa forma, o comparador não mudará para '1' e execução auto-temporizada terminará. A propriedade de auto-verificação do modo assíncrono torna-lhe bastante apropriado para realizar a verificação funcional da célula com a menor intervenção possível. A correta operação lógica pode ser checada pelo monitoramento do sinal do clock interno ou de um bit de dado no caminho realimentado.

Informações temporais podem ser extraídas dos ciclos de contagem '+K' completos. Essa medida representa o atraso médio do caminho lógico, já que em um circuito auto-temporizado a velocidade é tão alta quanto possível de acordo com o atraso de cada transição ou o tempo para terminar uma computação. Essas medidas podem ser usadas para validar os modelos se a simulação é feita no mesmo modo de operação.

Note-se que um conjunto de circuitos funcionando continuamente nesse modo pode ser usado para checar a confiabilidade e a robustez da biblioteca contra degradações. A contínua operação sem erros prova a confibilidade das células contra aspectos como insuficiência de contatos, eletromigração, NBTI (*negative bias temperature instability*) e outras causas. NBTI é a degradação dos transistores PMOS que depende de quanto tempo a saída da célula fica conectada a Vdd. Percebe-se que os circuitos não necessitam equipamentos caros para serem monitorados, uma vez que defeitos são auto-monitorados. Mesmo que os circuitos não falhem depois de serem continuamente estressados por um longo período de tempo, eles podem ser usados para medir a degradação causada pela NBTI.

Modo BIST oscilante

Nos modos síncrono e assíncrono, o mesmo valor binário de uma entrada i-indexada é esperado reaparecer na correspondente saída i-indexada do circuito de teste. Essa propriedade é garantida pela construção dos blocos combinacionais. Se a i-ésima saída é diretamente conectada à i-ésima entrada, o i-ésimo caminho é mantido fixo em um estado durante a realimentação. Entretanto, quando a i-ésima saída é invertida antes de conectá-la à i-ésima entrada, a polaridade lógica negativa ocorre e o i-ésimo caminho oscila. De acordo com esse princípio, no modo de oscilação a realimentação é conectada de tal forma que pelo menos uma das entradas primárias da cadeia receba o seu valor prévio negado, como ilustrado na Figura 3.

A inversão do bit é necessária para garantir que a condição de oscilação que é implementada através de um inversor inserido na entrada do multiplexador, indicado por um ‘*’ na Figura 2. Esse multiplexador é detalhado na Figura 4. Percebe-se que somente um bit do vetor Out(n..1) é selecionado de cada vez para fornecer somente a oscilação de um bit. Os outros bits são fixados aos valores fornecidos na entrada In(n..1). Isso é, se um para entrada-saída é considerado, somente o caminho associado oscila enquanto os outros são mantidos no mesmo estado.

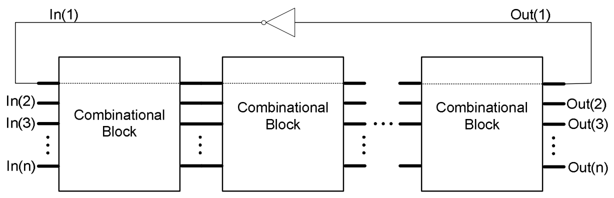


Figura 3 – Caminho do BIST em oscilação.

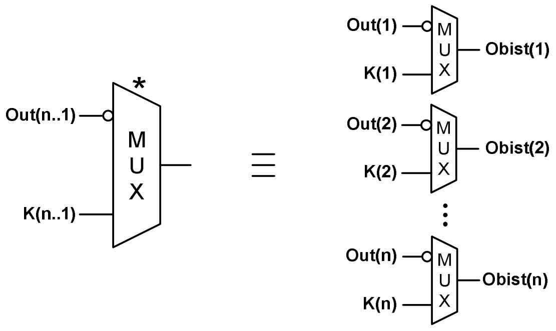


Figura 4 – Multiplexador utilizado no modo BIST em oscilação.

Percebe-se que o caminho interno percorrido pelo sinal em oscilação através da cadeia de blocos combinacionais dependerá dos valores binários estáticos aos quais as entradas do primeiro bloco são fixadas. Essa característica permite configurar uma grande quantidade de diferentes caminhos que oscilam. A principal contribuição do modo de oscilação para o processo de validação é que ele permite medir a propagação do sinal de atraso através de diferentes caminhos lógicos que envolvem instâncias de células distintas. Esses resultados de atraso são então comparados aos obtidos de uma prévia análise estática de tempo, que leva em conta a caracterização dos dados de tempo da célula (possivelmente obtidos através de simulações elétricas). Dessa forma é possível correlacionar os modelos utilizados pelo projeto com o comportamento real em silício.

Modo de Diagnóstico

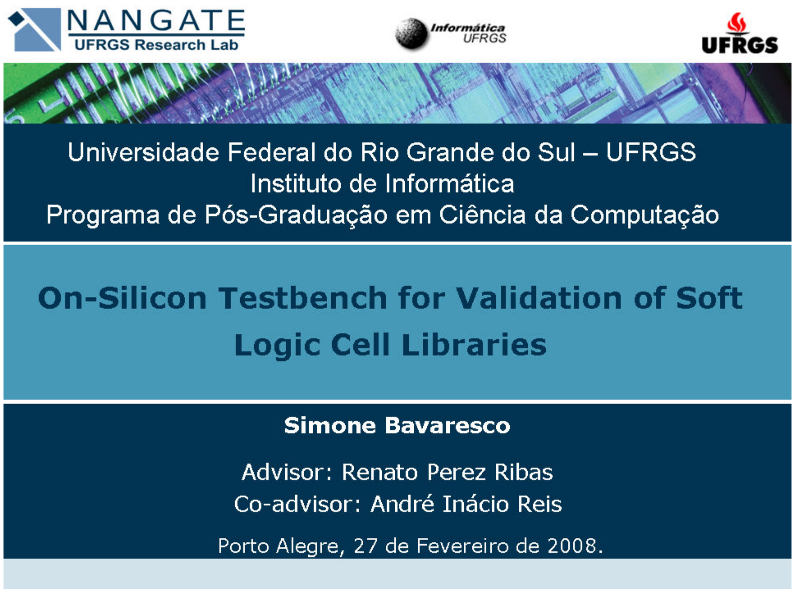
A fim de fazer o diagnóstico de defeitos, multiplexadores podem ser adicionados na arquitetura do circuito, sem penalidade significante em área, para selecionar partes da cadeia combinacional. Multiplexadores nas entradas de cada bloco combinacional podem selecionar o sinal do bloco anterior ou diretamente do começo da cadeia, removendo a influência dos blocos anteriores na realimentação do anel. Da mesma forma, multiplexadores na saída dos blocos podem mandar o dado do meio da cadeia diretamente para o fim da mesma. Dessa forma, a cadeia pode ser facilmente reduzida a um simples bloco ou mesmo a nenhum bloco, permitindo nesse caso a verificação do contador da barreira de registradores sem influência da cadeia combinacional.

Esse modo fornece o diagnóstico de um bloco com falhas e também permite os diferentes modos de operação do circuito descritos acima funcionarem em partes da cadeia completa. O diagnóstico também pode ser realizado na configuração de cadeia aberta, já que o multiplexador usado para fornecer o modo BIST oscilante também pode ser aplicado para interromper a configuração em anel sem circuito adicional. O modo de cadeia aberta permite o controle externo dos sinais de estímulo que são aplicados na cadeia de blocos combinacionais. Esse modo é bastante útil para a identificação de uma célula defeituosa.

4. Conclusões

Nesse trabalho foi proposto uma metodologia para validar em silício um conjunto de células geradas automaticamente. Essa validação testa completamente a funcionalidade das células e fornece meios de comparar os modelos de dados das células derivados da caracterização elétrica com o desempenho em silício. Um método para automaticamente gerar os circuitos de validação a partir de um conjunto de células inicial também foi proposto. O circuito de teste proposto apresenta pequeno acréscimo de área e diferentes modos de operação.

APPENDIX B WORK PRESENTATION



NANGATE UFRGS Research Lab

Informática UFRGS

UFRGS

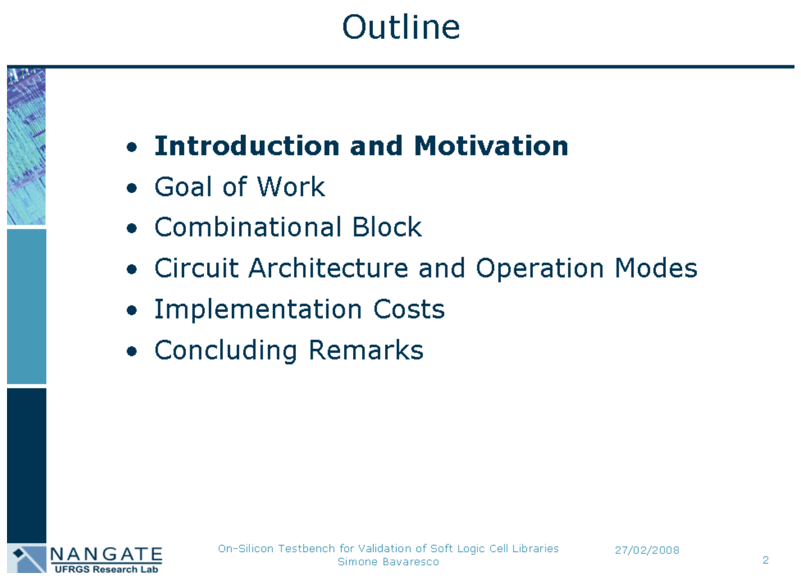
Universidade Federal do Rio Grande do Sul – UFRGS
Instituto de Informática
Programa de Pós-Graduação em Ciência da Computação

On-Silicon Testbench for Validation of Soft Logic Cell Libraries

Simone Bavaresco

Advisor: Renato Perez Ribas
Co-advisor: André Inácio Reis

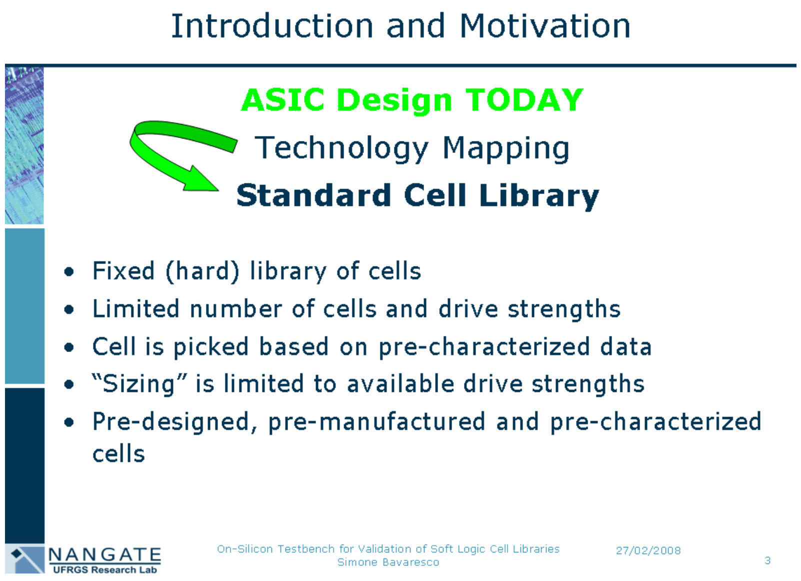
Porto Alegre, 27 de Fevereiro de 2008.



Outline

- **Introduction and Motivation**
- Goal of Work
- Combinational Block
- Circuit Architecture and Operation Modes
- Implementation Costs
- Concluding Remarks

NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries, Simone Bavaresco — 27/02/2008 — 2



Introduction and Motivation

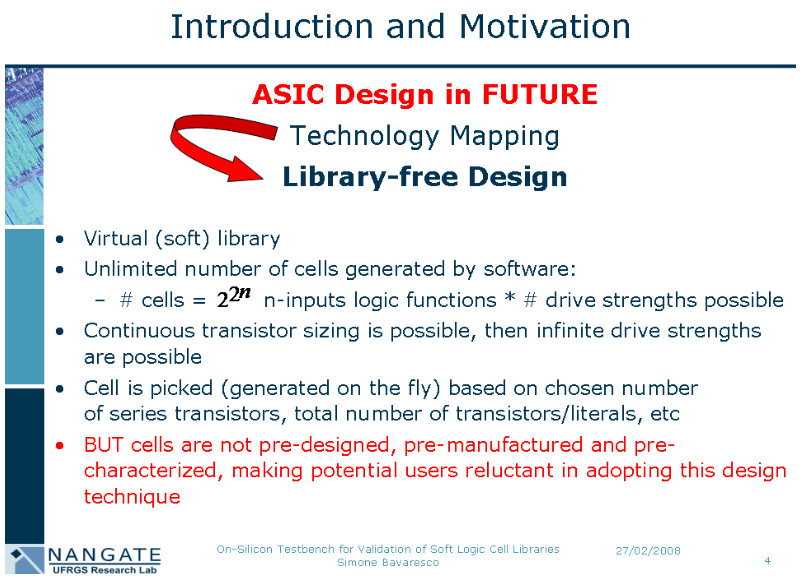
ASIC Design TODAY

Technology Mapping

Standard Cell Library

- Fixed (hard) library of cells
- Limited number of cells and drive strengths
- Cell is picked based on pre-characterized data
- "Sizing" is limited to available drive strengths
- Pre-designed, pre-manufactured and pre-characterized cells

NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries, Simone Bavaresco — 27/02/2008 — 3



Introduction and Motivation

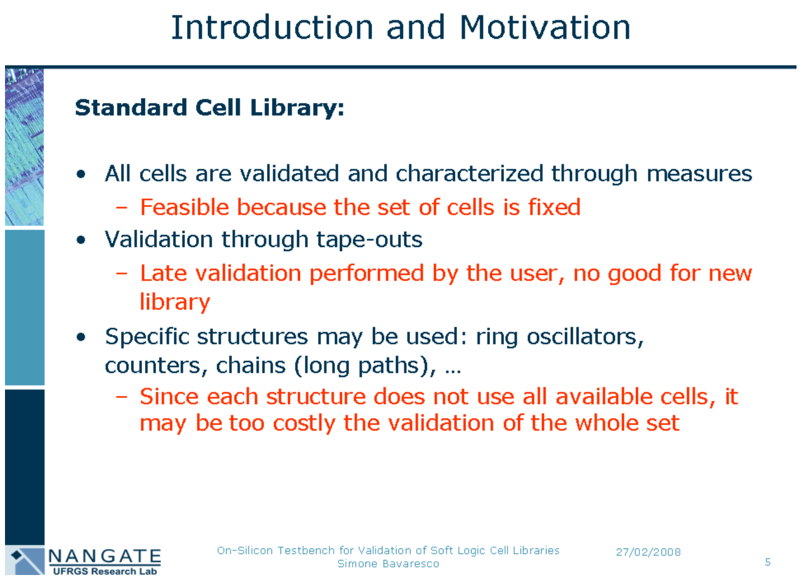
ASIC Design in FUTURE

Technology Mapping

Library-free Design

- Virtual (soft) library
- Unlimited number of cells generated by software:
  - # cells = 2^{2^n} n-inputs logic functions * # drive strengths possible
- Continuous transistor sizing is possible, then infinite drive strengths are possible
- Cell is picked (generated on the fly) based on chosen number of series transistors, total number of transistors/literals, etc
- BUT cells are not pre-designed, pre-manufactured and pre-characterized, making potential users reluctant in adopting this design technique

NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries, Simone Bavaresco — 27/02/2008 — 4

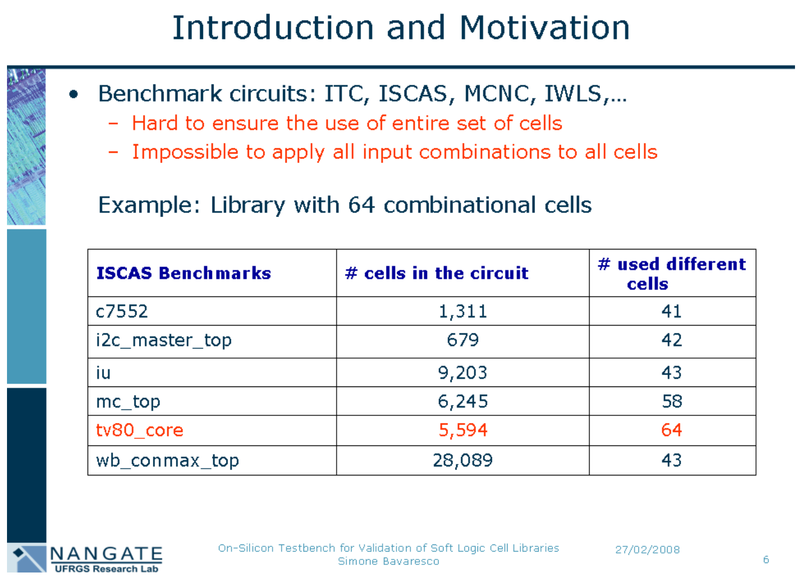


Introduction and Motivation

Standard Cell Library:

- All cells are validated and characterized through measures
  - Feasible because the set of cells is fixed
- Validation through tape-outs
  - Late validation performed by the user, no good for new library
- Specific structures may be used: ring oscillators, counters, chains (long paths), …
  - Since each structure does not use all available cells, it may be too costly the validation of the whole set

NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries, Simone Bavaresco — 27/02/2008 — 5



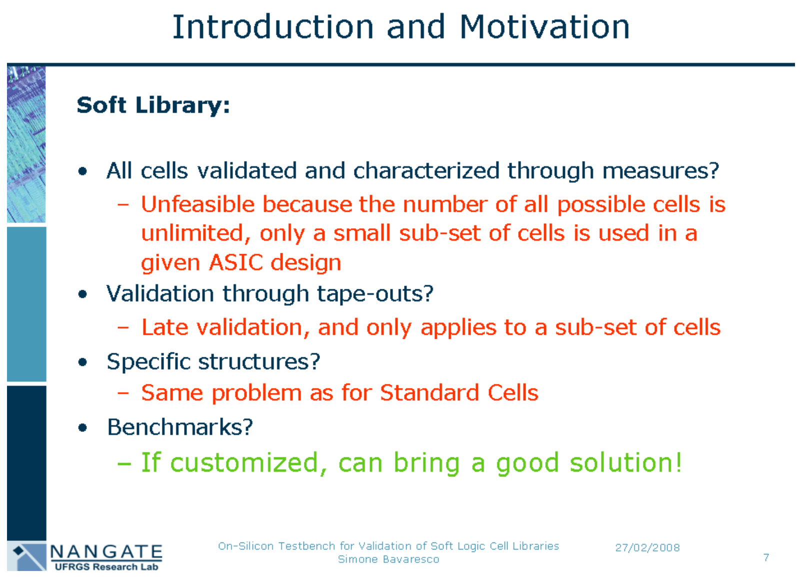
Introduction and Motivation

- Benchmark circuits: ITC, ISCAS, MCNC, IWLS,…
  - Hard to ensure the use of entire set of cells
  - Impossible to apply all input combinations to all cells

Example: Library with 64 combinational cells

ISCAS Benchmarks	**# cells in the circuit**	**# used different cells**
c7552	1,311	41
i2c_master_top	679	42
iu	9,203	43
mc_top	6,245	58
tv80_core	5,594	64
wb_conmax_top	28,089	43

NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries, Simone Bavaresco — 27/02/2008 — 6

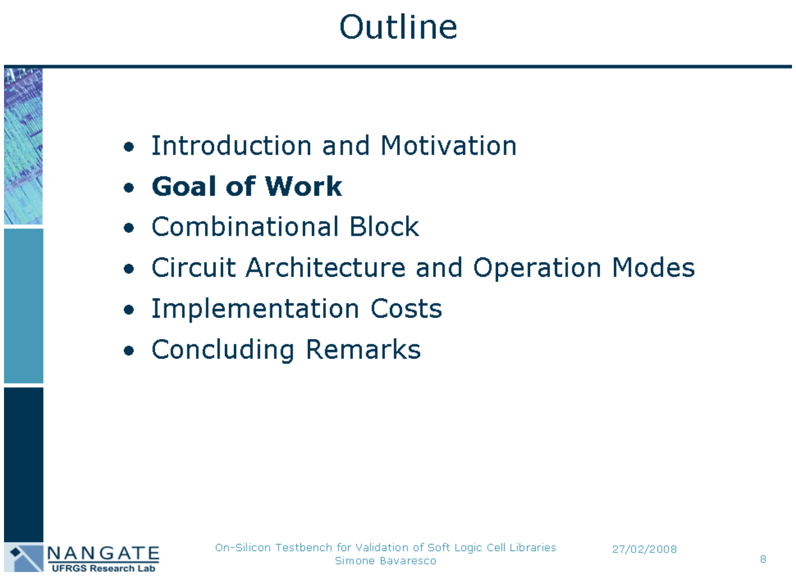


Introduction and Motivation

Soft Library:

- All cells validated and characterized through measures?
  - Unfeasible because the number of all possible cells is unlimited, only a small sub-set of cells is used in a given ASIC design
- Validation through tape-outs?
  - Late validation, and only applies to a sub-set of cells
- Specific structures?
  - Same problem as for Standard Cells
- Benchmarks?
  - If customized, can bring a good solution!

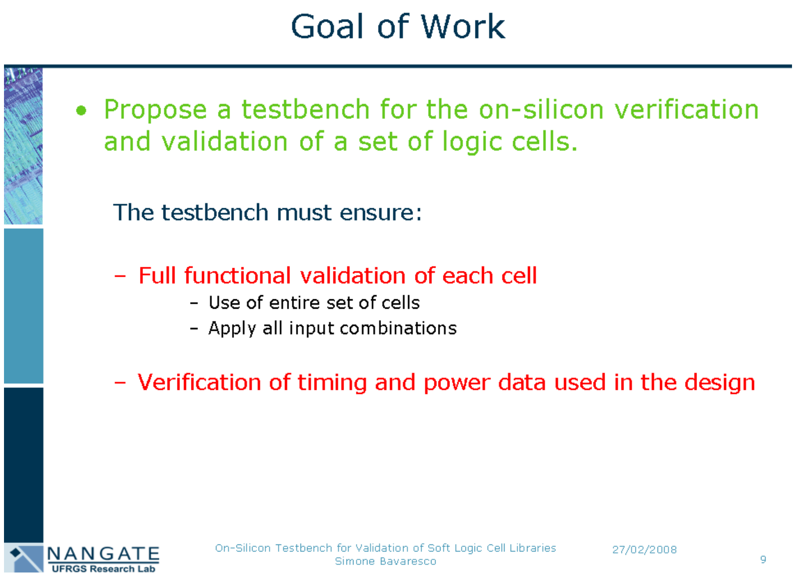
NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries, Simone Bavaresco — 27/02/2008 — 7



Outline

- Introduction and Motivation
- **Goal of Work**
- Combinational Block
- Circuit Architecture and Operation Modes
- Implementation Costs
- Concluding Remarks

NANGATE UFRGS Research Lab On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco 27/02/2008 8



Goal of Work

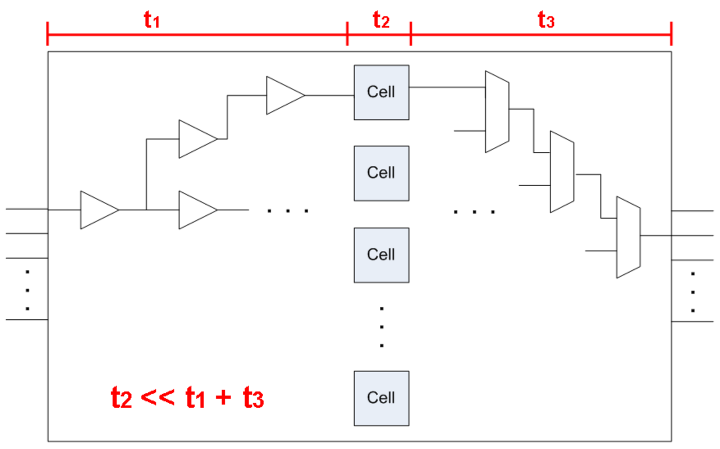
- Propose a testbench for the on-silicon verification and validation of a set of logic cells.

The testbench must ensure:

- Full functional validation of each cell
  - Use of entire set of cells
  - Apply all input combinations
- Verification of timing and power data used in the design

NANGATE UFRGS Research Lab On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco 27/02/2008 9

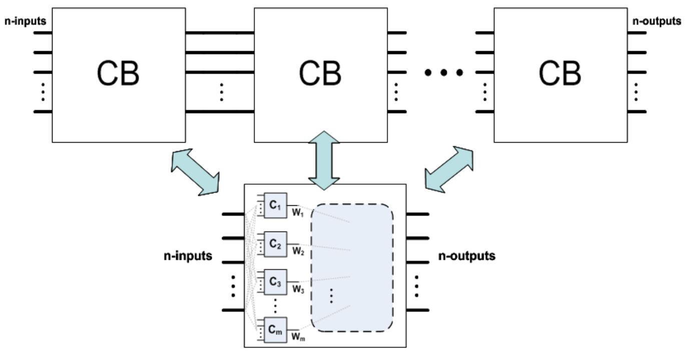
Trivial Solution?



NANGATE UFRGS Research Lab On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco 27/02/2008 10

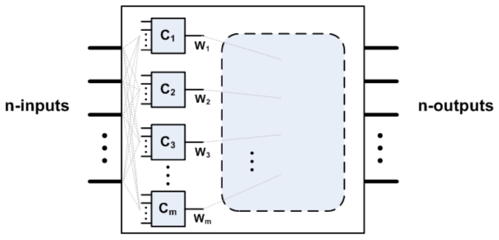
Proposed Testbench

- Cascade of combinational blocks reproducing the inputs at the outputs of the circuit



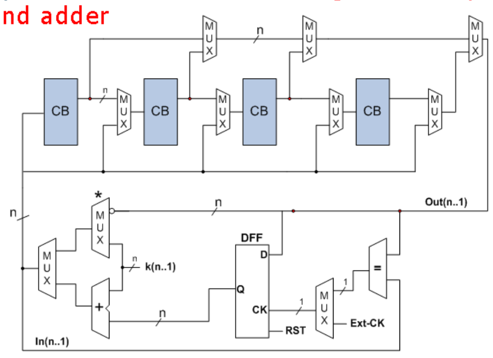
Combinational Block

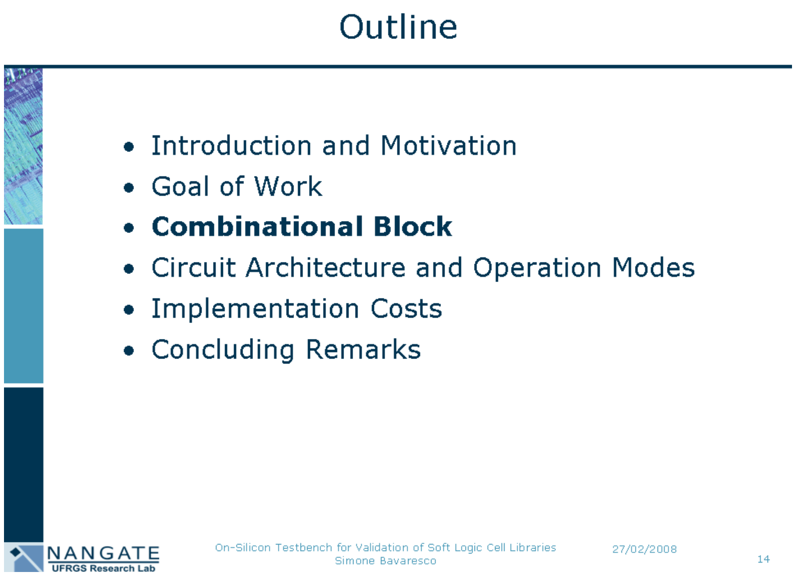
- Two stages block
- First: reduced number of cells picked from the library and placed in parallel
- Second: automatically synthesized function to recreate the original block input at its output



Additional Circuitry

- Needed for the implementation of different operation modes: muxes, register, comparator and adder

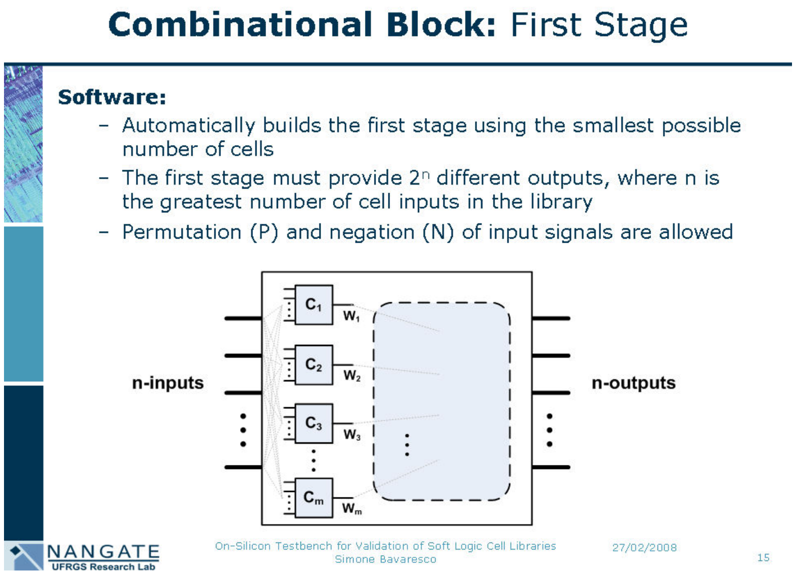




Outline

- Introduction and Motivation
- Goal of Work
- **Combinational Block**
- Circuit Architecture and Operation Modes
- Implementation Costs
- Concluding Remarks

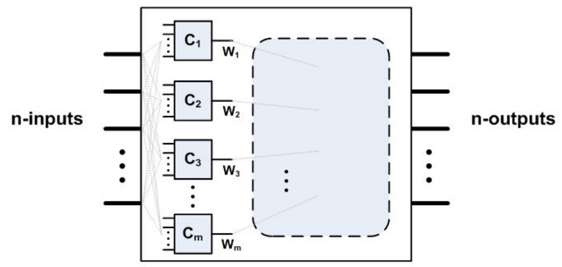
NANGATE UFRGS Research Lab
On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco
27/02/2008
14



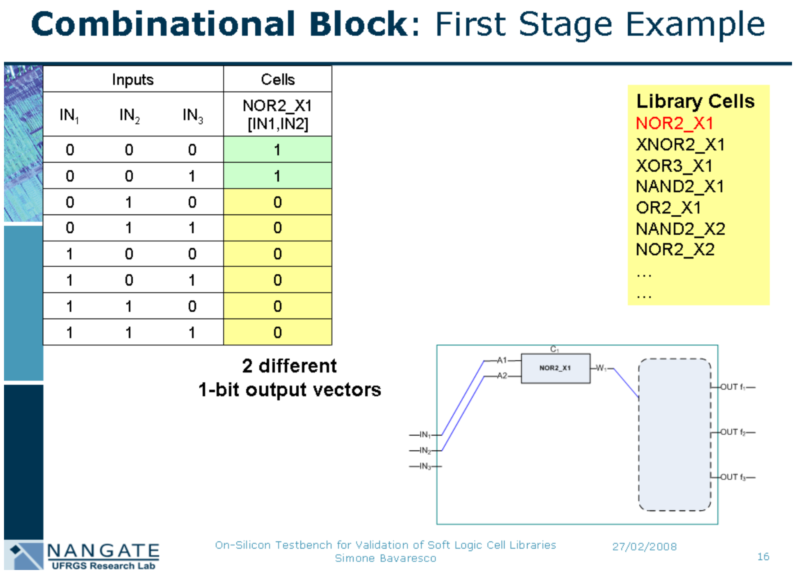
Combinational Block: First Stage

Software:

- Automatically builds the first stage using the smallest possible number of cells
- The first stage must provide 2^n different outputs, where n is the greatest number of cell inputs in the library
- Permutation (P) and negation (N) of input signals are allowed



NANGATE UFRGS Research Lab
On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco
27/02/2008
15

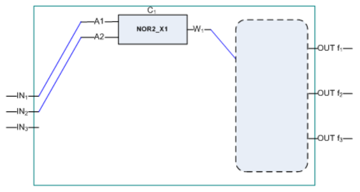


Combinational Block: First Stage Example

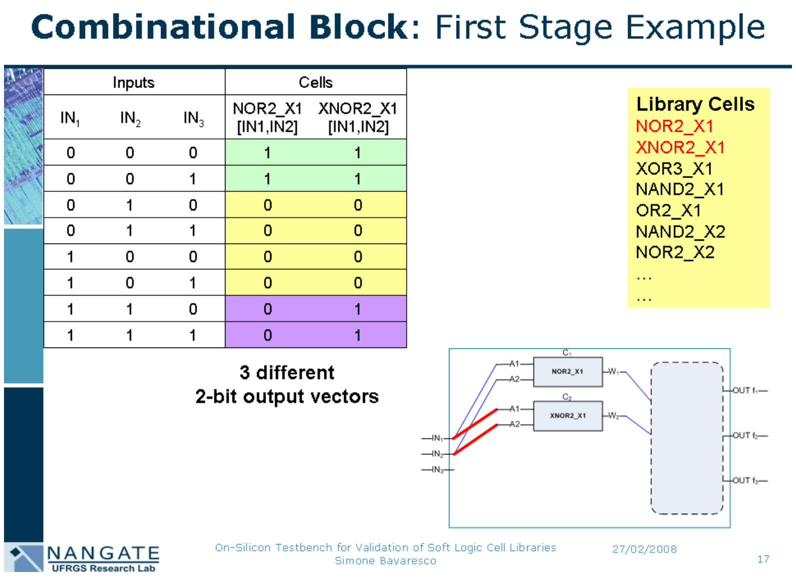
Inputs			Cells
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

2 different 1-bit output vectors

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…



NANGATE UFRGS Research Lab
On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco
27/02/2008
16



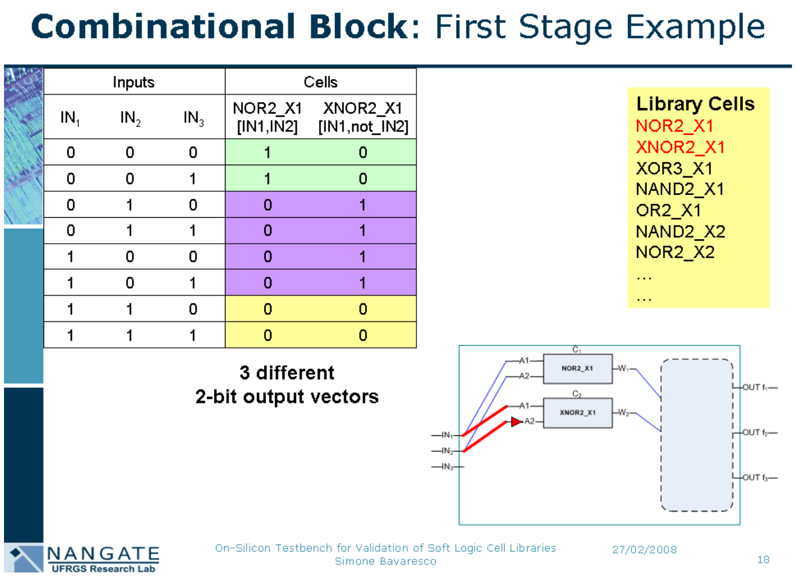
Combinational Block: First Stage Example

Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [IN1,IN2]
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	0	1

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

3 different
2-bit output vectors

C1
A1
A2
NOR2_X1
W1
C2
XNOR2_X1
W2
IN1
IN2
IN3
OUT f1
OUT f2
OUT f3



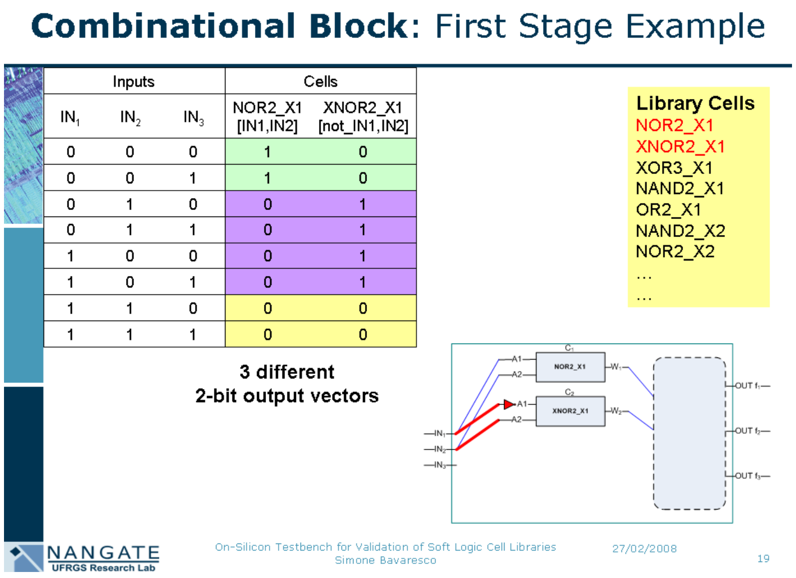
Combinational Block: First Stage Example

Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [IN1,not_IN2]
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

3 different
2-bit output vectors

C1
A1
A2
NOR2_X1
W1
C2
XNOR2_X1
W2
IN1
IN2
IN3
OUT f1
OUT f2
OUT f3



Combinational Block: First Stage Example

Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [not_IN1,IN2]
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

3 different
2-bit output vectors

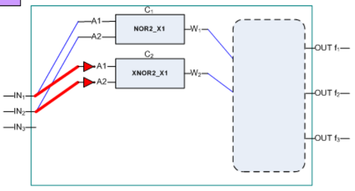
C1
A1
A2
NOR2_X1
W1
C2
XNOR2_X1
W2
IN1
IN2
IN3
OUT f1
OUT f2
OUT f3

Combinational Block: First Stage Example

Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [not_IN1,not_IN2]
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	0	1

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

3 different
2-bit output vectors

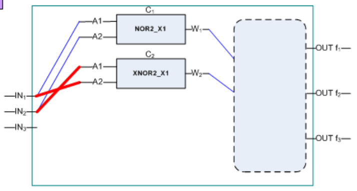


Combinational Block: First Stage Example

Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [IN2,IN1]
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	0	1

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

3 different
2-bit output vectors

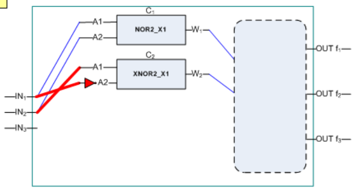


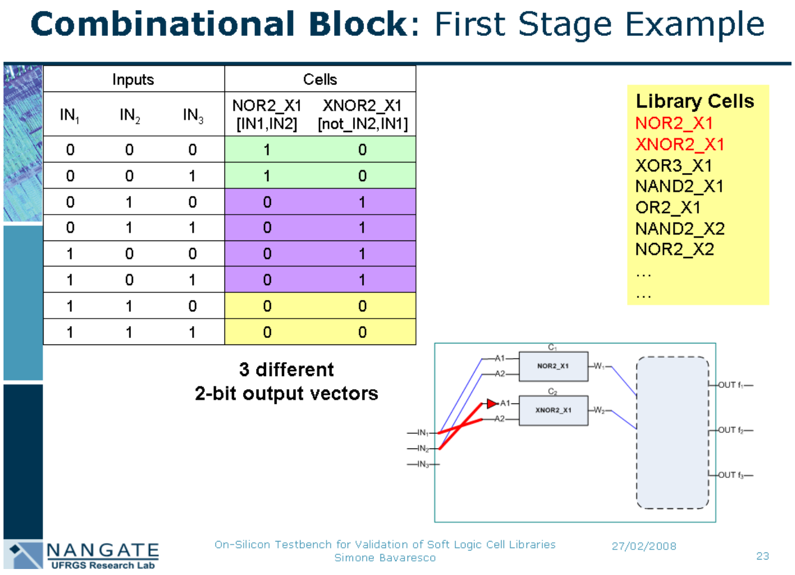
Combinational Block: First Stage Example

Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [IN2,not_IN1]
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

3 different
2-bit output vectors



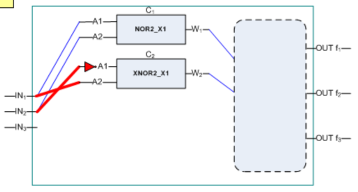


Combinational Block: First Stage Example

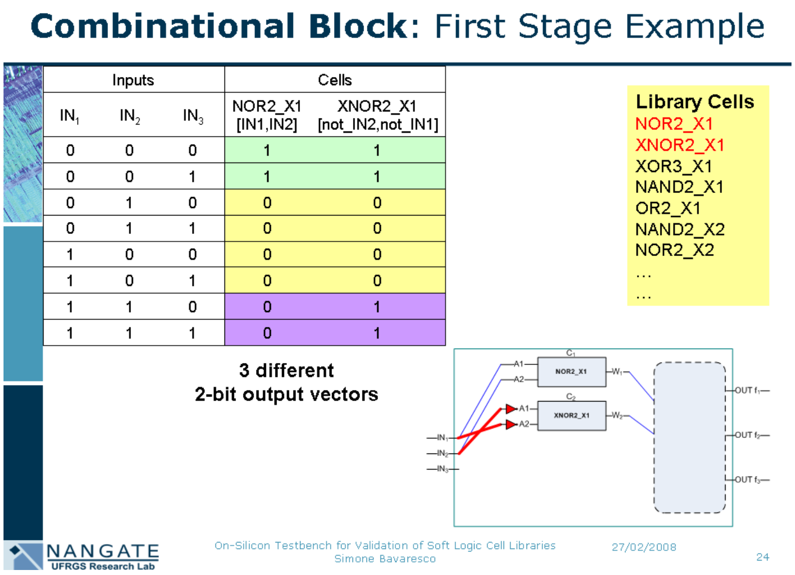
Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [not_IN2,IN1]
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

3 different
2-bit output vectors

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…



NANGATE UFRGS Research Lab

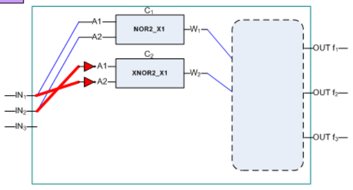


Combinational Block: First Stage Example

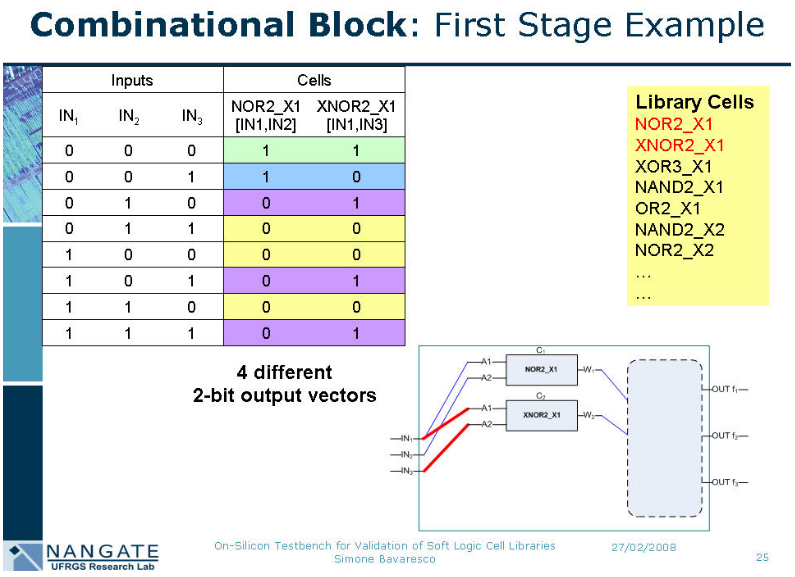
Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [not_IN2,not_IN1]
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	0	1

3 different
2-bit output vectors

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…



NANGATE UFRGS Research Lab

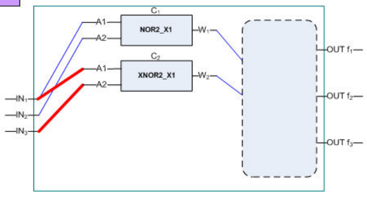


Combinational Block: First Stage Example

Inputs			Cells	
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [IN1,IN3]
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	0	1

4 different
2-bit output vectors

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…



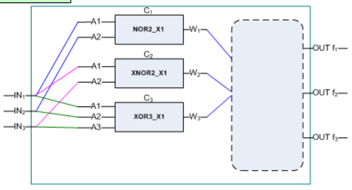
NANGATE UFRGS Research Lab

Combinational Block: First Stage Example

Inputs			Cells		
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [IN1,IN3]	XOR3_X1 [IN1,IN2,IN3]
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

6 different
3-bit output vectors



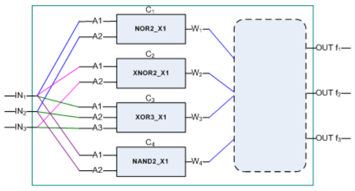
NANGATE UFRGS Research Lab
On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco
27/02/2008
26

Combinational Block: First Stage Example

Inputs			Cells			
IN_1	IN_2	IN_3	NOR2_X1 [IN1,IN2]	XNOR2_X1 [IN1,IN3]	XOR3_X1 [IN1,IN2,IN3]	NAND2_X1 [IN1,IN2]
0	0	0	1	1	0	1
0	0	1	1	0	1	1
0	1	0	0	1	1	1
0	1	1	0	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	0	0	0	0
1	1	1	0	1	1	0

Library Cells
NOR2_X1
XNOR2_X1
XOR3_X1
NAND2_X1
OR2_X1
NAND2_X2
NOR2_X2
…
…

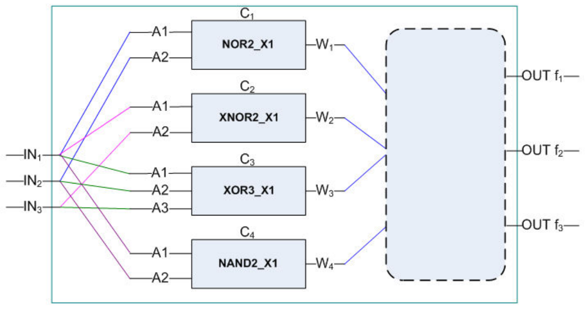
8 different
4-bit output vectors



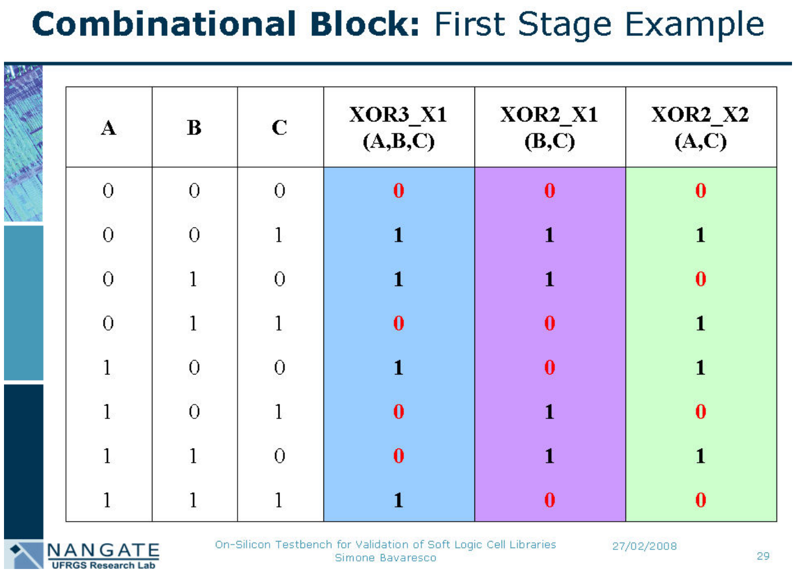
NANGATE UFRGS Research Lab
On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco
27/02/2008
27

Combinational Block: First Stage Example

- Single-output cells
  - smallest circuit → n cells
  - worst case → 2^n-1 cells.



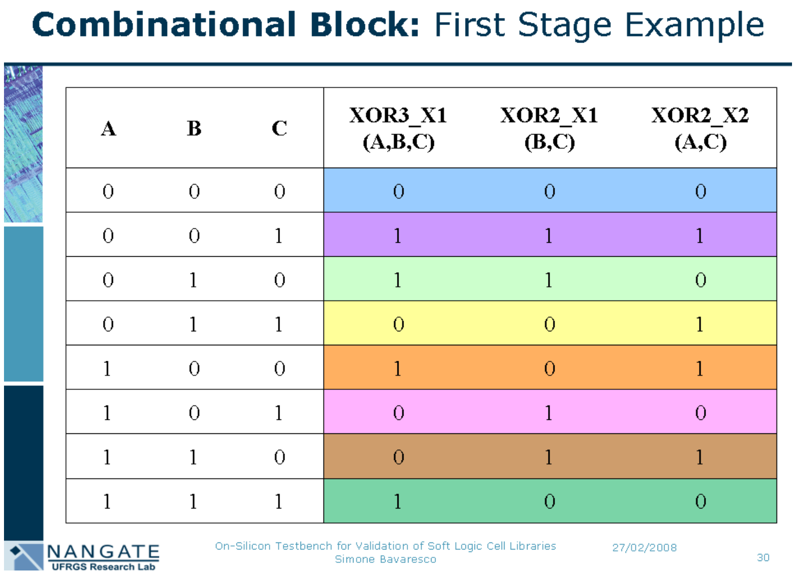
NANGATE UFRGS Research Lab
On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco
27/02/2008
28



Combinational Block: First Stage Example

A	B	C	XOR3_X1 (A,B,C)	XOR2_X1 (B,C)	XOR2_X2 (A,C)
0	0	0	**0**	**0**	**0**
0	0	1	**1**	**1**	**1**
0	1	0	**1**	**1**	**0**
0	1	1	**0**	**0**	**1**
1	0	0	**1**	**0**	**1**
1	0	1	**0**	**1**	**0**
1	1	0	**0**	**1**	**1**
1	1	1	**1**	**0**	**0**

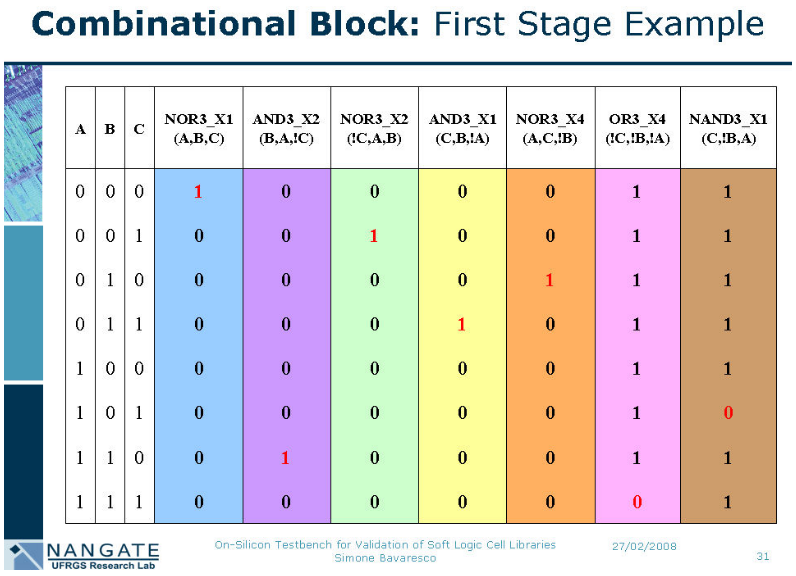
NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco — 27/02/2008 — 29



Combinational Block: First Stage Example

A	B	C	XOR3_X1 (A,B,C)	XOR2_X1 (B,C)	XOR2_X2 (A,C)
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	0	0

NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco — 27/02/2008 — 30



Combinational Block: First Stage Example

A	B	C	NOR3_X1 (A,B,C)	AND3_X2 (B,A,!C)	NOR3_X2 (!C,A,B)	AND3_X1 (C,B,!A)	NOR3_X4 (A,C,!B)	OR3_X4 (!C,!B,!A)	NAND3_X1 (C,!B,A)
0	0	0	**1**	**0**	**0**	**0**	**0**	**1**	**1**
0	0	1	**0**	**0**	**1**	**0**	**0**	**1**	**1**
0	1	0	**0**	**0**	**0**	**0**	**1**	**1**	**1**
0	1	1	**0**	**0**	**0**	**1**	**0**	**1**	**1**
1	0	0	**0**	**0**	**0**	**0**	**0**	**1**	**1**
1	0	1	**0**	**0**	**0**	**0**	**0**	**1**	**0**
1	1	0	**0**	**1**	**0**	**0**	**0**	**1**	**1**
1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**1**

NANGATE UFRGS Research Lab — On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco — 27/02/2008 — 31

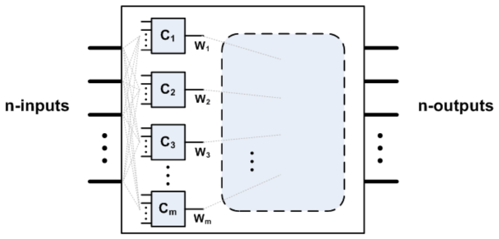
Combinational Block: First Stage Example

A	B	C	NOR3_X1 (A,B,C)	AND3_X2 (B,A,!C)	NOR3_X2 (!C,A,B)	AND3_X1 (C,B,!A)	NOR3_X4 (A,C,!B)	OR3_X4 (!C,!B,!A)	NAND3_X1 (C,!B,A)
0	0	0	1	0	0	0	0	1	1
0	0	1	0	0	1	0	0	1	1
0	1	0	0	0	0	0	1	1	1
0	1	1	0	0	0	1	0	1	1
1	0	0	0	0	0	0	0	1	1
1	0	1	0	0	0	0	0	1	0
1	1	0	0	1	0	0	0	1	1
1	1	1	0	0	0	0	0	0	1

Combinational Block: Second Stage

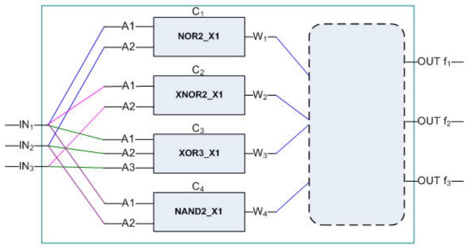
2nd Stage Design:

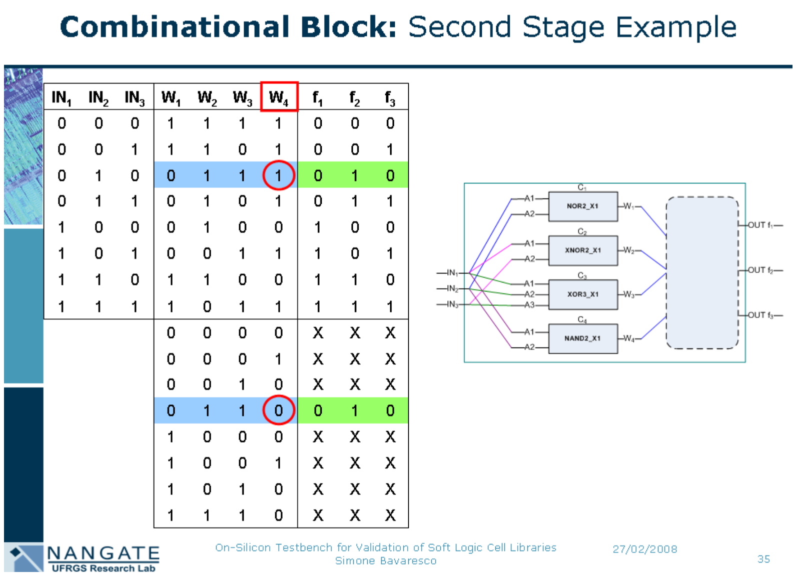
- Inputs → 1st stage m output signals
- Recreates the n block inputs at the block outputs from the 2^n m-bit vectors of the 1st stage
- Synthesized using standard synthesis tool



Combinational Block: Second Stage Example

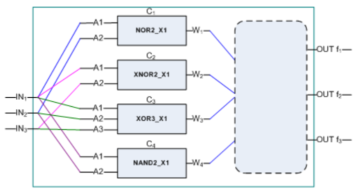
- Inputs → 4 output signals from 1st stage (m=4)
- Recreate at the block output, the 3 block inputs (n=3) from the 2^n (=8) 4-bit vectors from the 1st stage
- 2^3 distinct 4-bit vectors are available, but 2^4 are possible
- 2^4-2^3 vectors will not happen in fault-free condition, but may happen when block is faulty…





Combinational Block: Second Stage Example

IN$_1$	IN$_2$	IN$_3$	W$_1$	W$_2$	W$_3$	$\boxed{W_4}$	f$_1$	f$_2$	f$_3$
0	0	0	1	1	1	1	0	0	0
0	0	1	1	1	0	1	0	0	1
0	1	0	0	1	1	1	0	1	0
0	1	1	0	1	0	1	0	1	1
1	0	0	0	1	0	0	1	0	0
1	0	1	0	0	1	1	1	0	1
1	1	0	1	1	0	0	1	1	0
1	1	1	1	0	1	1	1	1	1
			0	0	0	0	X	X	X
			0	0	0	1	X	X	X
			0	0	1	0	X	X	X
			0	1	1	0	0	1	0
			1	0	0	0	X	X	X
			1	0	0	1	X	X	X
			1	0	1	0	X	X	X
			1	1	1	0	X	X	X

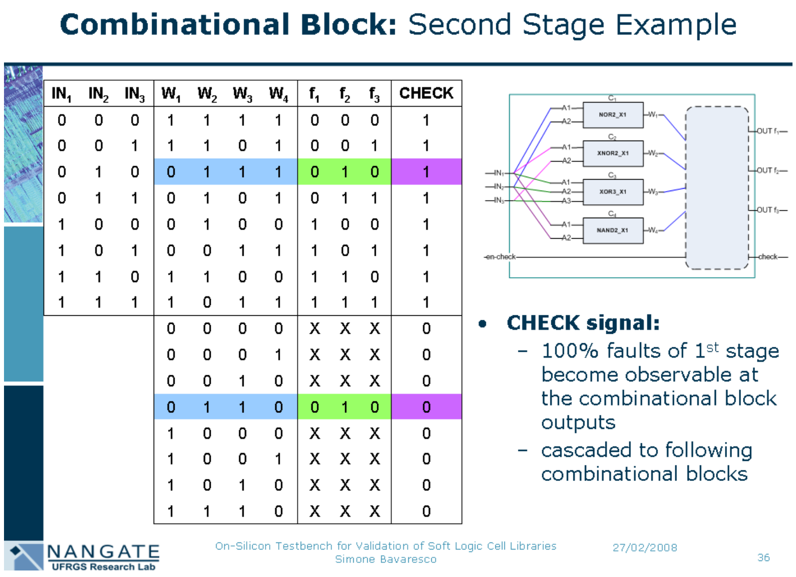


NANGATE
UFRGS Research Lab

On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco

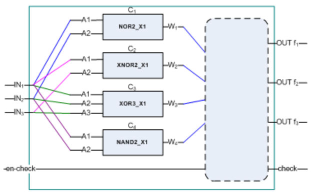
27/02/2008

35



Combinational Block: Second Stage Example

IN$_1$	IN$_2$	IN$_3$	W$_1$	W$_2$	W$_3$	W$_4$	f$_1$	f$_2$	f$_3$	CHECK
0	0	0	1	1	1	1	0	0	0	1
0	0	1	1	1	0	1	0	0	1	1
0	1	0	0	1	1	1	0	1	0	1
0	1	1	0	1	0	1	0	1	1	1
1	0	0	0	1	0	0	1	0	0	1
1	0	1	0	0	1	1	1	0	1	1
1	1	0	1	1	0	0	1	1	0	1
1	1	1	1	0	1	1	1	1	1	1
			0	0	0	0	X	X	X	0
			0	0	0	1	X	X	X	0
			0	0	1	0	X	X	X	0
			0	1	1	0	0	1	0	0
			1	0	0	0	X	X	X	0
			1	0	0	1	X	X	X	0
			1	0	1	0	X	X	X	0
			1	1	1	0	X	X	X	0



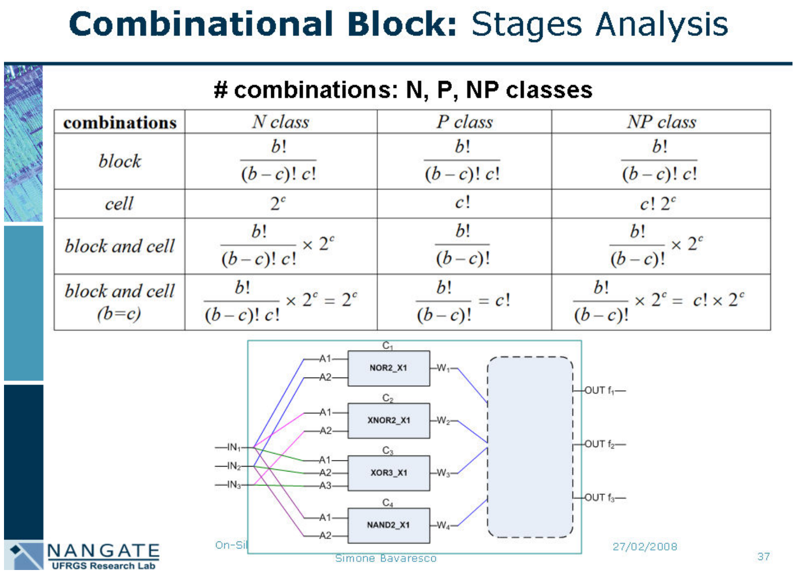
- **CHECK signal:**
  - 100% faults of 1st stage become observable at the combinational block outputs
  - cascaded to following combinational blocks

NANGATE
UFRGS Research Lab

On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco

27/02/2008

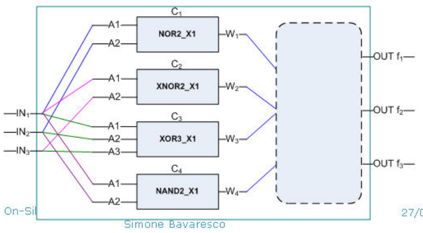
36



Combinational Block: Stages Analysis

# combinations: N, P, NP classes

combinations	*N class*	*P class*	*NP class*
block	$\frac{b!}{(b-c)!\,c!}$	$\frac{b!}{(b-c)!\,c!}$	$\frac{b!}{(b-c)!\,c!}$
cell	2^c	$c!$	$c!\,2^c$
block and cell	$\frac{b!}{(b-c)!\,c!} \times 2^c$	$\frac{b!}{(b-c)!}$	$\frac{b!}{(b-c)!} \times 2^c$
block and cell (b=c)	$\frac{b!}{(b-c)!\,c!} \times 2^c = 2^c$	$\frac{b!}{(b-c)!} = c!$	$\frac{b!}{(b-c)!} \times 2^c = c! \times 2^c$



NANGATE
UFRGS Research Lab

On-Si

Simone Bavaresco

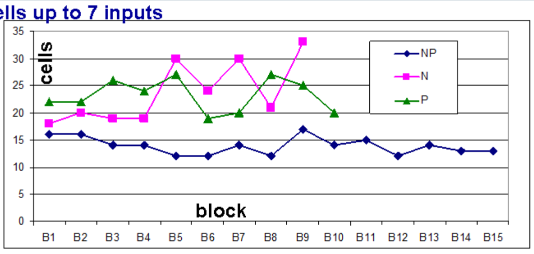
27/02/2008

37

Combinational Block: Stages Analysis

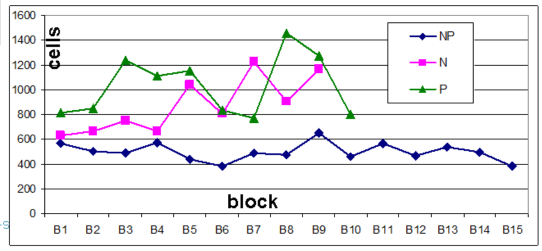
Target library: 208 cells up to 7 inputs

First Stage

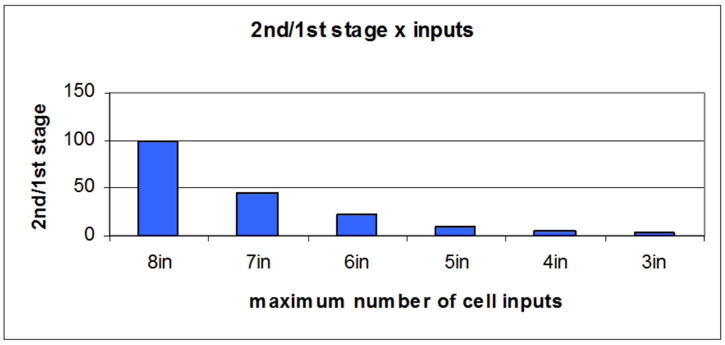


	1st (cells)	2nd (cells)	Build time (s)
NP	208	7472	25087
N	214	7860	15
P	232	10306	432

Second Stage



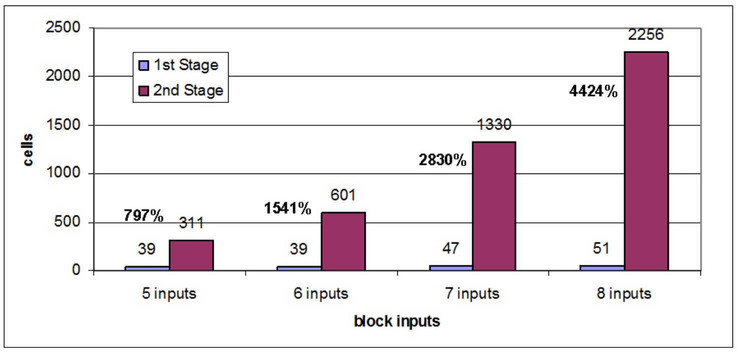
Combinational Block: Stages Analysis



Combinational Block: Stages Analysis

cells 1st e 2nd stages → # block inputs

Library: Cells up to 5 inputs

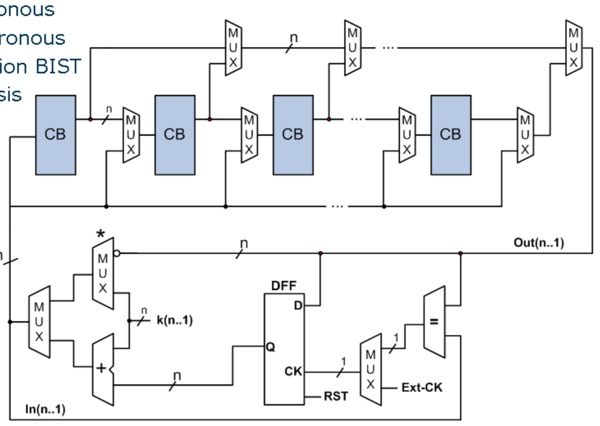


Outline

- Introduction and Motivation
- Goal of Work
- Basic Combinational Block
- **Circuit Architecture and Operation Modes**
- Implementation Costs
- Concluding Remarks

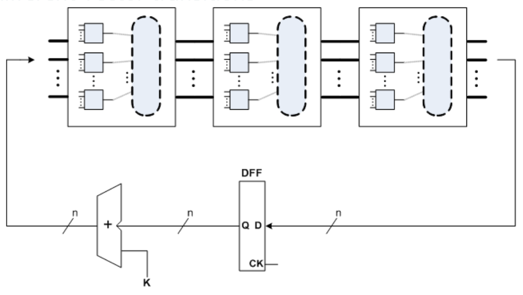
Operation Modes

- **Operation modes:**
  - Synchronous
  - Asynchronous
  - Oscillation BIST
  - Diagnosis

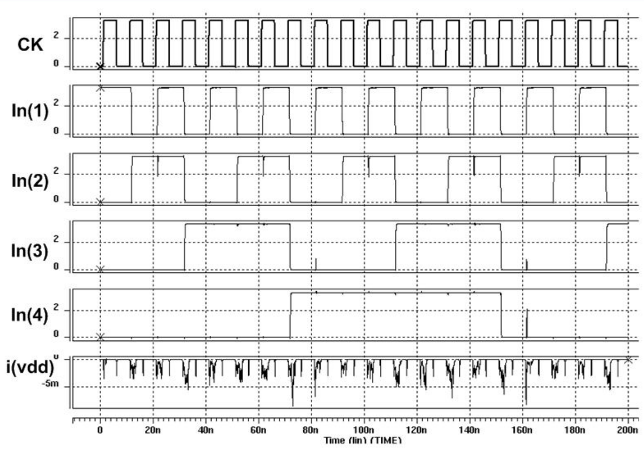


Synchronous Mode

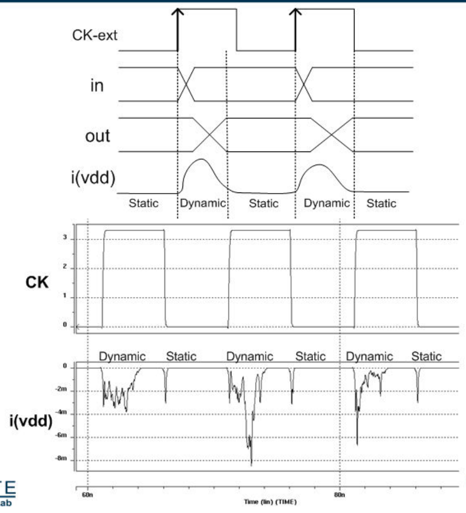
- **Synchronous mode**
  - External clock needed
  - Maximum operation frequency: worst case path delay
  - Static and dynamic consumption are evaluated
  - Perform sums by 'K': consumption can be evaluated for different vector transitions



Synchronous Mode

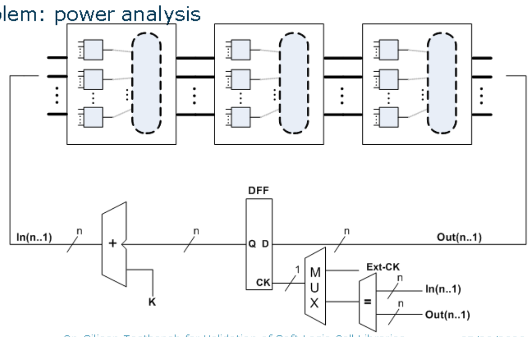


Synchronous Mode

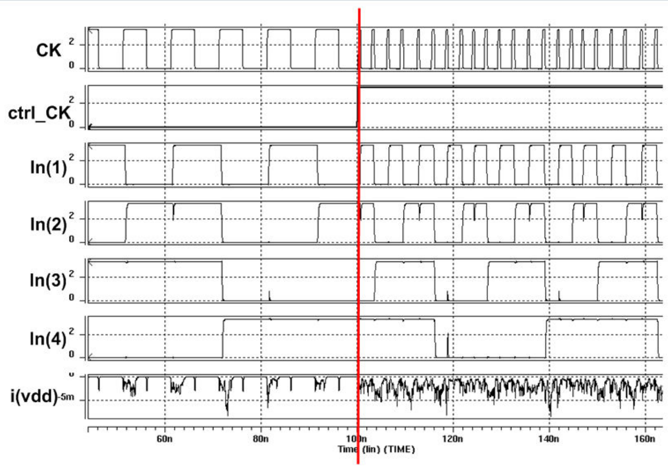


Asynchronous Mode

- **Asynchronous mode**
  - Self-timed counter, no external stimulus needed
  - Functional cell verification (Self-checking: logic error → circuit stops)
  - Average timing analysis
  - Problem: power analysis



Asynchronous Mode



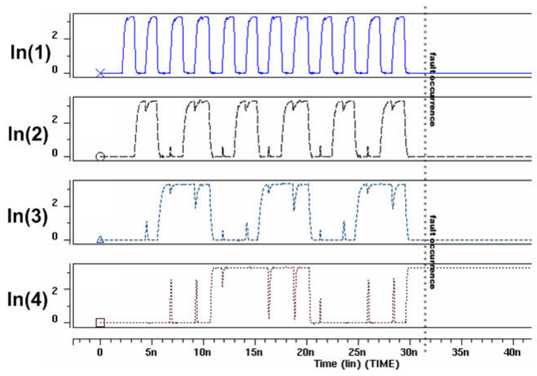
NANGATE UFRGS Research Lab

On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco

27/02/2008

47

Asynchronous Mode



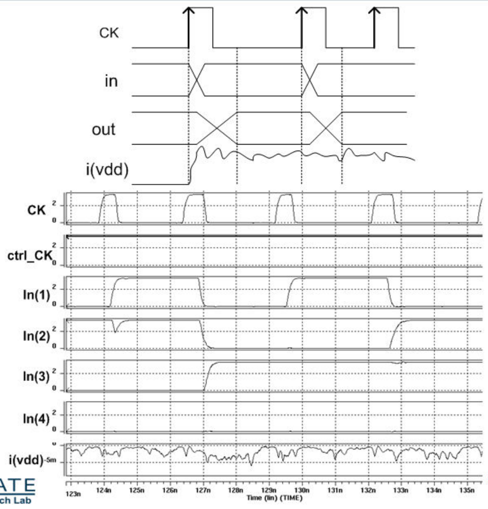
NANGATE UFRGS Research Lab

On-Silicon Testbench for Validation of Soft Logic Cell Libraries
Simone Bavaresco

27/02/2008

48

Asynchronous Mode



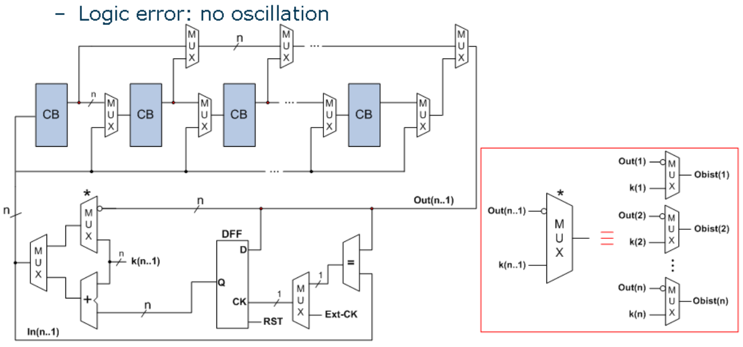
NANGATE UFRGS Research Lab

27/02/2008

49

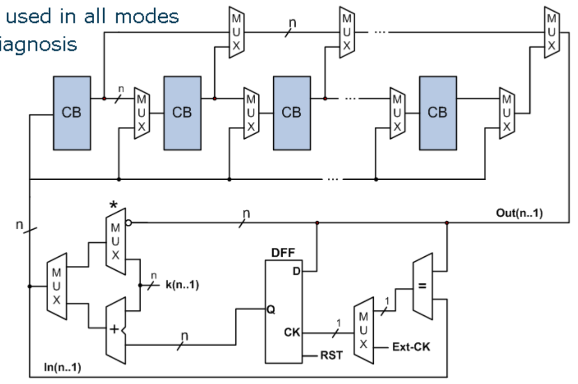
OBIST Mode

- **Oscillation BIST mode**
  - No external stimulus needed
  - Delay propagation through different logical paths
  - Delay verification → compared to static timing analysis
  - Logic error: no oscillation

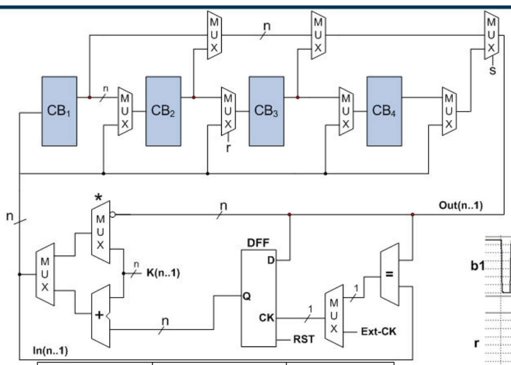


Diagnosis Mode

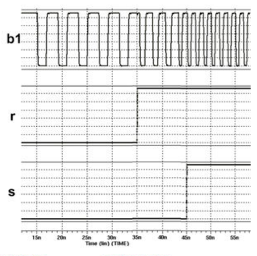
- **Diagnosis mode**
  - Isolate combinational blocks
  - Delay verification: parts of the circuit
  - Can be used in all modes
  - Fault diagnosis



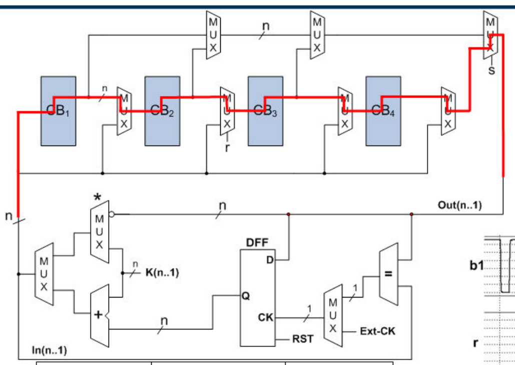
Diagnosis Mode

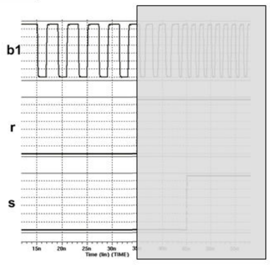


Interval (ns)	Active blocks	Period (ns)
15-35	complete chain	4.07
35-45	blocks 3 and 4	2.52
45-60	block 3	1.64



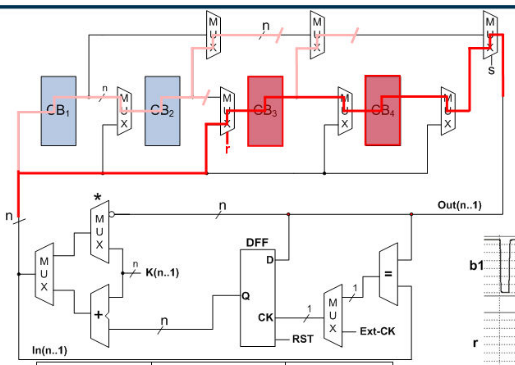
Diagnosis Mode

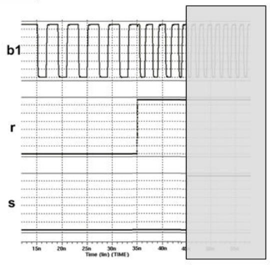




Interval (ns)	Active blocks	Period (ns)
15-35	complete chain	4.07
35-45	blocks 3 and 4	2.52
45-60	block 3	1.64

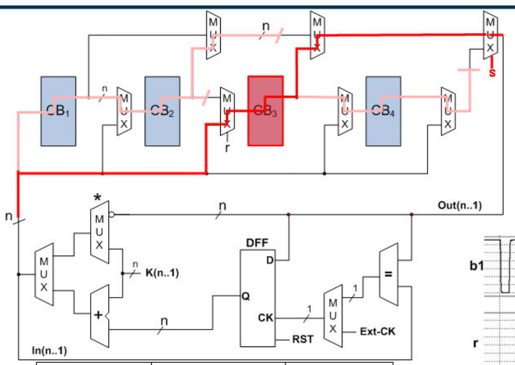
Diagnosis Mode

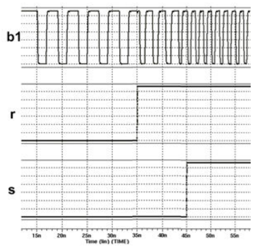




Interval (ns)	Active blocks	Period (ns)
15-35	complete chain	4.07
35-45	blocks 3 and 4	2.52
45-60	block 3	1.64

Diagnosis Mode





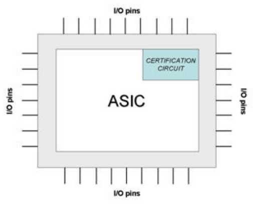
Interval (ns)	Active blocks	Period (ns)
15-35	complete chain	4.07
35-45	blocks 3 and 4	2.52
45-60	block 3	1.64

Outline

Use of the Testbench

Business model point-of-view

- Soft-library vendor
  - Physical testbench
    - Guarantee the correctness of its EDA environment
    - Verify the quality of the generated cells (performance and reliability / DFM issues)
    - Improvement of the library generation CAD tool
- ASIC designer / vendor
  - Exclude errors or faults on silicon due to the cell generators
  - Fabricated in the same die of the ASIC (certification circuit)
  - Overhead



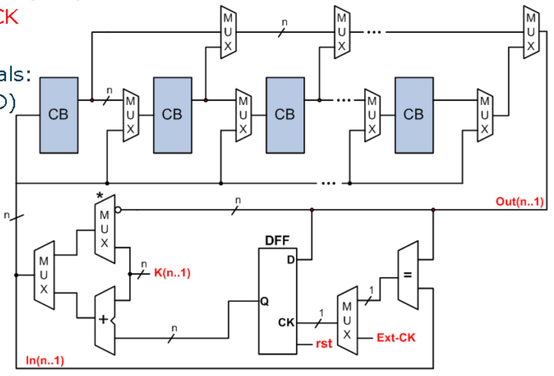
Testbench: Area Overhead

ISCAS Benchmarks	# cells in the circuit	# used different cells
c7552	1,311	41
i2c_master_top	679	42
iu	9,203	43
mc_top	6,245	58
tv80_core	5,594	64
wb_conmax_top	28,089	43

- **Using the proposed testbench:**
  - For a 64 cells library:
    - 450 cells testbench generated
    - Combinational blocks and additional circuitry included
    - 8% overhead, could serve for circuit-level certification
  - For a 208 cells library:
    - 8,000 cells testbench generated

Testbench: Pin Overhead

- **I/O pins**
  - n inputs: K(n..1)
  - n outputs: Out(n..1)
  - Clock: Ext-CK
  - Reset: rst
  - control signals: MUX(DECOD)



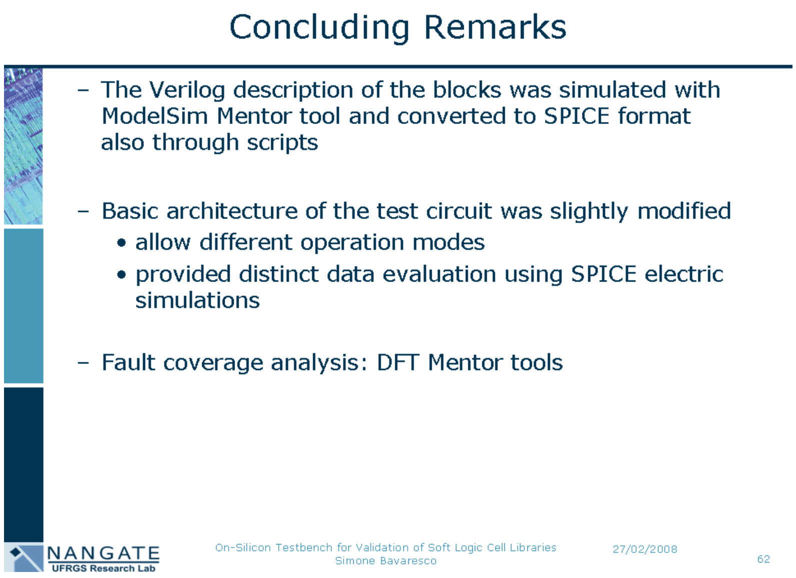
Outline

- Motivation and Goal of Work
- The Testbench
- Combinational Block Design
- Operation Modes
- Implementation Costs
- **Concluding Remarks**

Concluding Remarks

Summary:

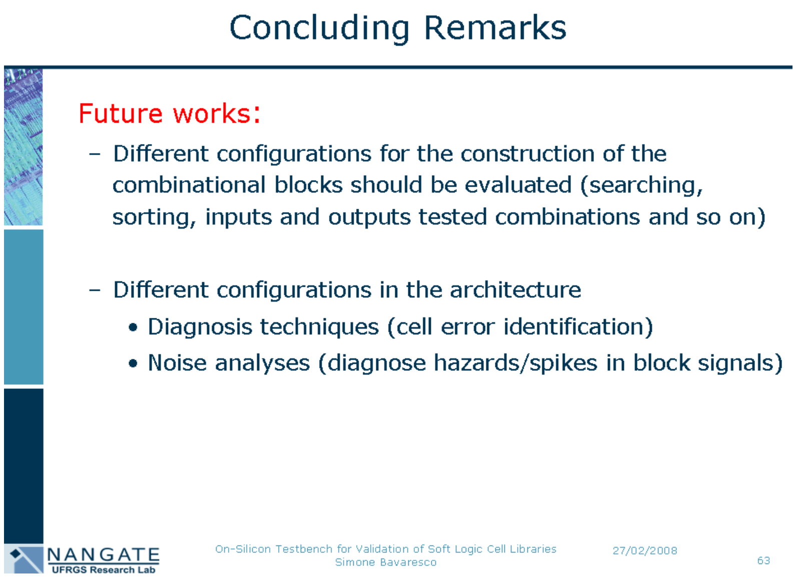
- An on-silicon testbench for the verification and validation of software-generated libraries was proposed
  - CAD tool was developed in Java in order to automate the generation of combinational blocks
  - Tool output:
    - descriptions of the first stage of each combinational block in mapped Verilog
    - description of the functions of the second stage of each combinational block in non-mapped Verilog
  - Functions of the second stage of the blocks
    - mapped through scripts with RTLCompiler Cadence tool and ABC Berkeley tool



Concluding Remarks

- The Verilog description of the blocks was simulated with ModelSim Mentor tool and converted to SPICE format also through scripts
- Basic architecture of the test circuit was slightly modified
  - allow different operation modes
  - provided distinct data evaluation using SPICE electric simulations
- Fault coverage analysis: DFT Mentor tools

NANGATE UFRGS Research Lab On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco 27/02/2008 62

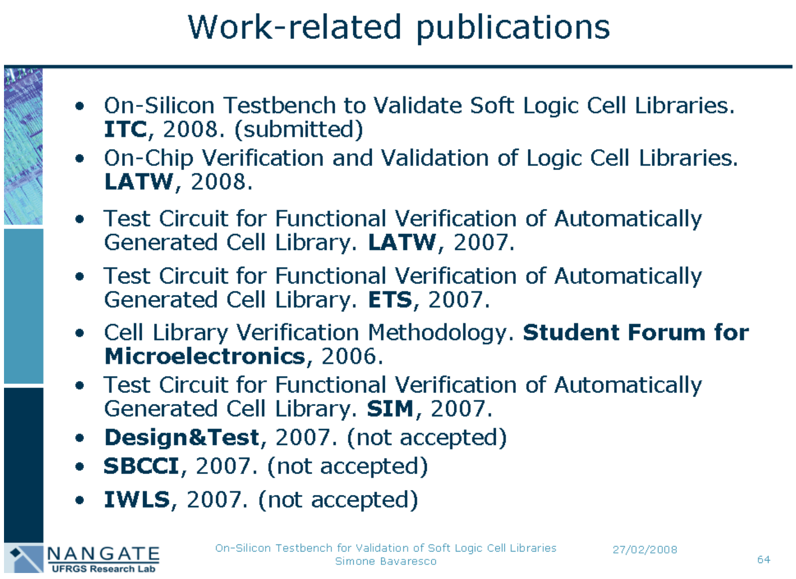


Concluding Remarks

Future works:

- Different configurations for the construction of the combinational blocks should be evaluated (searching, sorting, inputs and outputs tested combinations and so on)
- Different configurations in the architecture
  - Diagnosis techniques (cell error identification)
  - Noise analyses (diagnose hazards/spikes in block signals)

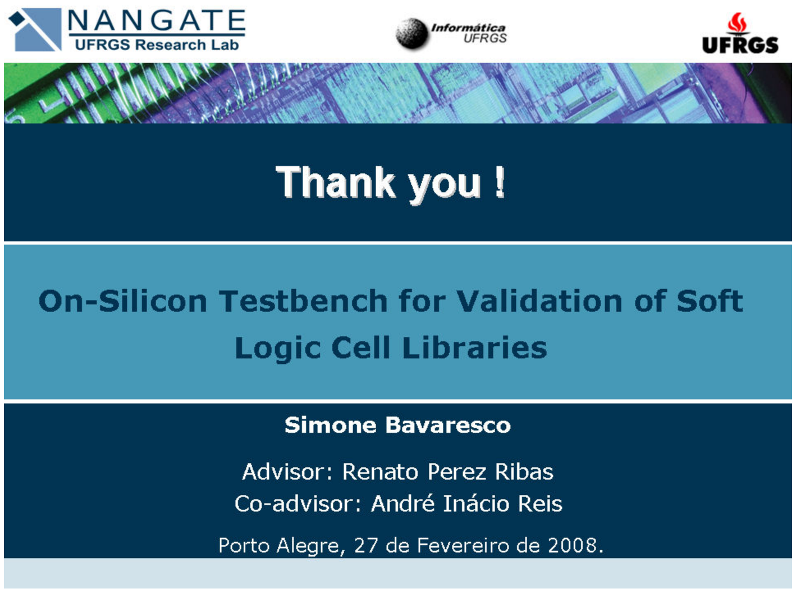
NANGATE UFRGS Research Lab On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco 27/02/2008 63



Work-related publications

- On-Silicon Testbench to Validate Soft Logic Cell Libraries. **ITC**, 2008. (submitted)
- On-Chip Verification and Validation of Logic Cell Libraries. **LATW**, 2008.
- Test Circuit for Functional Verification of Automatically Generated Cell Library. **LATW**, 2007.
- Test Circuit for Functional Verification of Automatically Generated Cell Library. **ETS**, 2007.
- Cell Library Verification Methodology. **Student Forum for Microelectronics**, 2006.
- Test Circuit for Functional Verification of Automatically Generated Cell Library. **SIM**, 2007.
- **Design&Test**, 2007. (not accepted)
- **SBCCI**, 2007. (not accepted)
- **IWLS**, 2007. (not accepted)

NANGATE UFRGS Research Lab On-Silicon Testbench for Validation of Soft Logic Cell Libraries Simone Bavaresco 27/02/2008 64



NANGATE
UFRGS Research Lab

Informática UFRGS

UFRGS

Thank you !

On-Silicon Testbench for Validation of Soft Logic Cell Libraries

Simone Bavaresco

Advisor: Renato Perez Ribas
Co-advisor: André Inácio Reis

Porto Alegre, 27 de Fevereiro de 2008.

APPENDIX C PAPER SUBMITED TO ITC'08

On-Silicon Testbench to Validate Soft Logic Cell Libraries

R. P. Ribas[1], S. Bavaresco[1], M. Lubaszewski[1], A. I. Reis[2]

[1]Instituto de Informática – UFRGS, Av. Bento Gonçalves, 9500, CEP 91501-970, Porto Alegre, Brazil
[2]Nangate Inc., Menlo Park, CA, 94025, USA
rpribas@inf.ufrgs.br, simoneb@inf.ufrgs.br, luba@ece.ufrgs.br, are@nangate.com

Abstract

This work proposes a validation methodology to silicon-prove a set of logic cells generated by software. It also presents an approach for the automatic design of testbenches to validate the cells in the set.

1. Introduction

Cell-based design is definitely the most applied approach in the ASIC market today. This design approach implies in re-using library cells to build more complex digital circuits. A typical standard cell design environment includes timing and power analysis, as well as automatic assembling of circuit layout through place and route tools. In standard cell libraries, three groups of cells co-exist: (1) inverters/buffers; (2) combinational cells and (3) sequential ones. Mainly due to the large number of different logic functions and driving strength options needed in typical designs, the largest of the three aforementioned groups is the set of combinational logic gates. The handcraft creation of standard cell libraries demands skilled designers and long development times, even when simply dealing with technology migration for the same set of cells. Each cell must be carefully designed and characterized for different input slopes, output loads and design corners. In practice, the high engineering costs of these tasks imposes a limitation on the number of available combinational cells in libraries.

As the technology mapping step in standard IC design flow is based on pre-characterized data of pre-designed cells, the ASIC design space and efficiency turns to be bounded by the library in use. The more cells and drive strength options are available, the larger are the possibilities to improve the circuit design. The enrichment of a library can be done by adding only new drive strengths [1], or through the addition of new functions with standard series/parallel implementation [2], or even with special transistor topologies [3-5]. New topologies can also be considered for in-place optimization (IPO), including in-context cell sizing [6]. The use of extended libraries leads to an optimized fit for particular applications. Due to this added flexibility to the design space, there is an increasing commercial interest for approaches that consider on-the-fly generation of cells, like [1, 6-9]. These approaches are sometimes referred as library-free or as soft-cell or liquid library based.

The main drawback of library-free technology mapping technique is the use of such soft IPs or non-silicon-validated set of cells in the ASIC design. This fact makes conservative customers reluctant in adopting this design technique.

This work proposes a validation methodology to silicon-prove the set of automatically designed

logic cells. This validation methodology covers the full functionality of the cells and provides timing and power consumption data useful to validate and fine tune cell data models derived from electrical characterization. Notice that, the circuit speed and consumption estimation tasks, during the design flow, are performed based on cell data from electrical characterization, available in a Liberty^{TM} file, for instance. Indeed, the on-silicon testbench for soft libraries, proposed herein, allows checking whether data used in the performance estimation during the circuit design flow produces good prediction of silicon behavior. A method to automatically generate the validation circuit specific for a given initial set of cells is also proposed.
The remainder of this paper is organized as follows. In Section 2, the motivations for this work are further discussed. The way the library cells are used to build the basic combinational blocks of the testbench is described in Section 3. In Section 4, the circuit architecture is then presented and its operating modes are shown pointing out their goals. The overhead analysis is given in Section 5, and conclusions are outlined in Section 6.

2. Problem Statement and Goal of Work

ASIC designs are usually bounded by the standard cell library in use. For this reason, there are a number of commercial and academic efforts that take advantage of on-the-fly creation of cells. The method proposed by DeDood *et al.* [1] creates new drive strengths for existing cells (from a starting cell library), in order to save power and reduce delay in a target ASIC. A method that creates dedicated complex cells to reduce delay in applications needing high-speed ASICs is introduced in [6, 8]. Jones *et al.* [9] proposed a method to optimize a design by changing transistor sizes in the available cells and then redesigns the original library to accommodate these new cells. For advanced technology nodes, research initiatives by leading semiconductor companies are considering the importance of the target library as part of the design space [10]. The use of more complex gates can reduce the overall number of transistors and provide layouts that are less dense in transistors/mm^2 but denser in terms of logic/mm^2. This idea has been pointed as part of a regular layout solution for process variability [11]. The use of such complex gates can also bring advantage to the regular ASIC world. Indeed, several methods to generate efficient transistor networks [3, 5, 7, 11] and to perform technology mapping targeting complex gates [2, 4, 6, 8] have been recently proposed.

Conservative customers feel uncomfortable with the idea of using a ‘non silicon-proven’ library. The silicon proof should address two main issues. First, it has to guarantee that the models used during the synthesis flow represent adequately the final silicon performance. Second, it has to prove that the cells are reliable under the expected working conditions. This aspect guarantees that the cells present a sufficient number of contacts, do not have latch-up problems, and so on. Notice that this proof has to be done properly by library providers every time a new technology node is available.

The most straightforward way to design a test circuit for a set of cells consists in instantiating all cells using shared primary inputs (for test controllability), while individual output signals could be multiplexed in order to reduce the number of circuit I/O pins, while maintaining test observability. This strategy can obviously provide full functionality validation of each logic gate. However, cell timing information is somewhat difficult to obtain through a circuit with only one-level of logic depth. Additionally, the multiplexers at the output and eventual buffering to compensate the high capacitance on input nodes, as well as the packaging structure, make this approach impractical to measure the timing characteristics of a single gate. That is one of the reason for using ring oscillators and oscillation test structures in delay characterization and testing, respectively [12, 13].

However, the extensive use of ring oscillators would result into a large number of instances of each cell to be validated, which is impractical for libraries with a large number of cells. Another drawback of this library validation approach is that the test for reliability (sufficiency of contacts, electromigration, etc), based on the continuous application of input vectors and the monitoring of the outputs, requires an automatic test equipment (ATE) to apply the vectors and check the results, increasing the test cost. Therefore, an approach that can make these tests viable without additional ATE is strongly desirable. One possibility for this would be to design an application circuit whose functionality could be easily verified. However, it is important to notice that not all the cells available in a library are necessarily used by the technology mapping tool for a given circuit. This fact is shown in Table I, where it is possible to see that out of a set of 64 available gates not all of them were instantiated after a given circuit was mapped, with exception of circuit 'tv80_core', which used all the cells. For these reasons, more efficient testbenches for testing a set of cells may be proposed.

Table I – Mapped circuits showing the number of effectively used cells out of a set of 64 different available combinational logic gates.

ISCAS Benchmarks*	Number of cells instances	Number of distinct cells
c7552	1,311	41
i2c_master_top	679	42
Iu	9,203	43
mc_top	6,245	58
tv80_core	5,594	64
wb_conmax_top	28,089	43

* Benchmarks source: http://www.fm.vslib.cz/~kes/asic/iscas, last access on Sep.20th, 2007.

An efficient approach to generate a testbench for testing a set of new cells, possibly created on-the-fly, should cover the following aspects:

1) to ensure complete functionality test for the instantiated cells;
2) to ensure coverage (instantiation) of all the cells to be tested;
3) to allow the verification of the accuracy of the models used in the design process;
4) to provide means to perform long and medium term reliability tests (needed for electromigration, sufficiency of contacts, NBTI degradation, etc) without additional equipment;
5) to have a feasible number of cell instances compared to the set of cells to be tested.

The goal of this work is to propose a methodology to automatically generate testbench circuits for on-silicon soft-library validation meeting the aspects above. To the best of our knowledge, previous approaches fail to cope efficiently with this goal.

In this work, a straightforward and efficient testbench methodology is proposed aiming the validation of an entire set of soft-cells in terms of logic and electrical behavior. The presented solution merges well-established design and test concepts to cope with the five aspects mentioned above. A specific combinational block is built to guarantee the logic coverage (aspect 1) of a sub-set of the cells to be validated, and to provide at the output the same bit vector received at its inputs, allowing thus to cascade long chains with these blocks. The use of several blocks allows to instantiate all the cells (aspect 2). The circuit architecture is then composed of such combinational chain in a ring configuration, synchronized by a register barrier. Both synchronous and asynchronous operating modes provide different features for the proposed goals. The ring configuration allows verifying the accuracy of the models, by comparing with the predicted circuit behavior (aspect 3). The oscillation BIST technique is also included in the circuit operation for a wide range of different paths [13], and allows medium and long term tests (aspect 4). In case of an eventual error, the circuit diagnostic is facilitated through an arrangement of multiplexers. Finally, as the ring oscillator is composed of a variety of cells, the number of instances is not very expensive compared to the initial set of cells (aspect 5), as shown in the overhead section.

From a business model point-of-view, the methodology presented herein is useful for the soft-library vendor and to the ASIC designer client. For the vendor, it is quite important to dispose of a physical testbench in order to guarantee the correctness of its EDA environment, as well as to verify the quality of the generated cells in terms of performance and reliability, including design-for-manufacturability issues [14]. This is essential for the continuous improvement of the library generation CAD tool. For the ASIC designer, a circuit that validates all distinct cells created on-the-fly to be used in a specific circuit provides means to exclude that errors on silicon due to the cell generators. If this test circuit is fabricated in the same die of the ASIC, it can act as a kind of 'certification circuit' for the soft-library, in different design corners and operating conditions. In this case, the overhead in terms of area and I/O pins is a compromise in fabricating together the test circuit and the ASIC, and the low cost approach presented here is very attractive.

The detailed descriptions of the combinational blocks, overall architecture and operation modes of the testbench are given in next sections.

3. Testbench: the Combinational Blocks

The combinational blocks are built in a way to guarantee the complete and correct logic behavior of all cells included in the soft-library under test. To attain this, each combinational block is composed by two sub-blocks or stages. The first one is built by cells in a single logic depth level, all of them excited by the (shared) primary inputs. This is similar to the straightforward arrangement to allow test controllability and observability, discussed in the previous section, but now including only a small sub-set of cells. These cells, in the first stage, are connected to the block inputs for full logic exercising. The output signals of the cells in the first stage are then used as inputs to the second stage, which recomputes the same bit vector applied at the first sub-block primary inputs. As a result, the primary inputs of the first sub-block are equal to the primary outputs of the second sub-block. The internal interface of the two sub-blocks can be viewed as an intermediate code for which primary inputs are translated and then recovered.

Before detailing each stage of the combinational blocks, the following principles and requirements might be pointed out:

1) all blocks present the same number of input and output nodes, which must be equal or higher than the biggest number of inputs in a single cell;
2) input and output vectors have equal steady state logic values.
3) every cell has to be instantiated at least once in the first stage of a combinational block;
4) the total number of combinational blocks depend on the size of the cells set and also the quantity of cells necessary to compose the first stage of each block;
5) the second stage of the combinational blocks is synthesized taking into account only the cells present in the soft-library to be validated.

To generate the first stage of a block, the whole set of cells is initially ordered according to one of the following criteria: alphabetic order; number of cell inputs; quantity (or rate) of 0s and 1s provided by the logic function; or random order. Once the cells are ordered, they are taken one-by-one to create the circuit in this first stage. Assuming ‘n’ the number of signals at both the input and output block interface, the goal is to minimize the length of the internal intermediate code in the interface of the two sub-blocks. Notice that the length of the intermediate code corresponds to the number of cells used in the first stage, if single output cells are assumed. The minimization procedure has to choose a number of cells (‘m’) that produces at least 2^n different values at the m-bit output of the first stage. The 2^n different intermediate code values (distinct m-bit values) at the output of the first stage represent the minimum required to reconstruct the original 2^n input combinations at the n-output signals at the second stage of the block, with a one-to-one correspondence.

The reason to minimize the length of the intermediate code (represented by the number of cells in the first stage) is that the complexity of the second stage of the block is proportional to the size (number of bits) of the intermediate code. Thus, reducing the number of bits in the intermediate code reduces the area overhead of the second stage. One way to perform this minimization is the following. The first selected cell will give a 1-bit length code where the codes ‘0’ and ‘1’are possible. Next, cells will add one bit each and a given number of distinct vectors. The criterion to select new cells it to choose them to maximize the number of different intermediate vectors at each new instantiation (that adds one bit to the intermediate code). To increase the number of new vectors in the intermediate code, the input signals of each cell can be permutated (P) and/or negated (N), and the best option is then selected. Cells that are included in a firsts stage sub-block will not be considered in the generation of subsequent ones. Cells that do not increase the number of different intermediate vectors are not instantiated and remain to be used by a subsequent first stage sub-block generation. The combinational block generation stops when there is no unused cell remaining. Already used cells can be reused during the generation of the last block, as few options of unused cells are available. Consider a soft-library where only single-output cells are available and where the validation circuit is composed of n-bit blocks (n-inputs and n-outputs). The minimum number of cells at the first stage is equal to ‘n’, as the intermediate code cannot have a smaller length than the input/output codes for the block. The worst case is composed by maximum 2^n-1 cells, as the first cell gives two distinct vectors and subsequent cells are required to introduce at least one new vector.

To illustrate the construction of the first stage, consider a 3-input combinational block, as shown in Fig. 1. The intermediate code composed of ‘Wi’ bits requires 2^3 distinct vectors, necessary to rebuild the 3-bits input vectors at the 3-bits block outputs. This requirement was not attained with the three first selected cells, as illustrated in Table II (second column). Thus, at least one additional cell $\boldsymbol{C_4}$ must be added to the first stage. The third and the fourth columns in Table II show two alternative $\boldsymbol{C_4}$ input connections where only one provides the 2^3 necessary distinct vectors. At this point, the first stage of this combinational block is concluded.

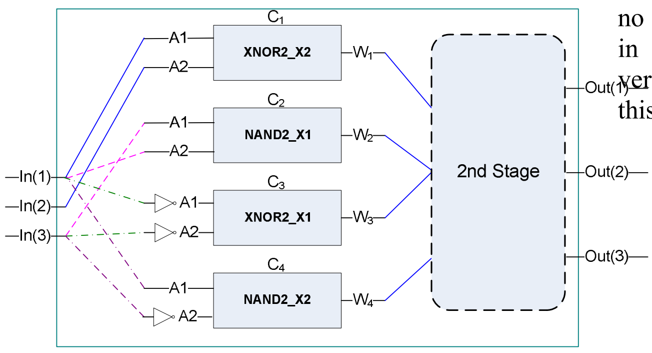


Figure 1 - 3-bits combinational block: example.

The second stage re-creates the input vectors at the output of the block, from the intermediate codes represented at ‘Wi’. The output signals of the first stage are used as inputs to the second stage, as shown in Table III. Since the length of the intermediate code can be larger than the length of I/O codes, some ‘Wi’ combinations will never occur. This way, *don’t cares* are used to optimize the synthesis of the second stage.

In this work, the generation of the first stage of each block was automated by using a specific CAD tool, developed in Java platform. The second stage, in turn, is synthesized with standard technology mapping engines.

The construction of the combinational blocks has been exercised for different set of cells in order to evaluate the complexity (size) of the circuits generated with this methodology. Fig. 2 shows the block building considering a set of 208 cells, with up to 7 inputs, obtained from the ‘genlib_44-6’ library [15]. Only logically distinct functions are included in this set. The possibility of applying permutations (P) and negations (N), or both (NP), at the inputs during the cell instantiation was also evaluated.

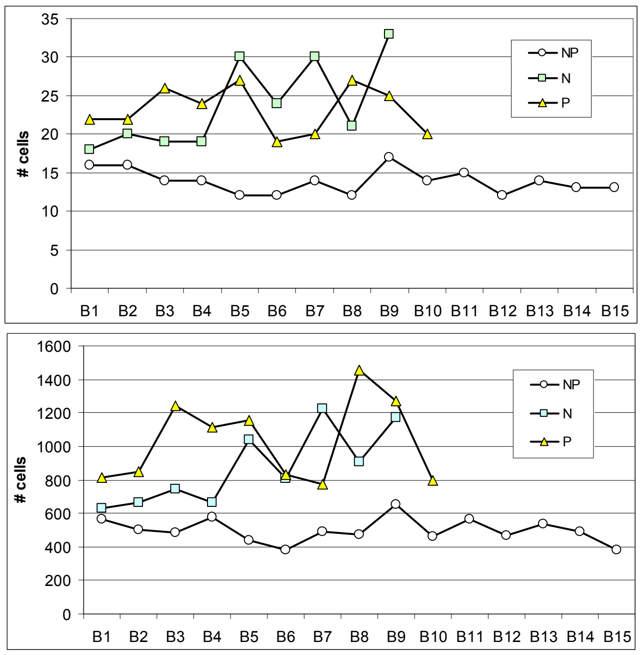
In Fig. 2a, the X-axis represents the number of blocks generated and the Y-axis shows the number of cells in the first stage of each block, which is equivalent to the length of the intermediate code. As it can be observed, the possibility to consider the permutation and negation (NP) of the cell inputs resulted in smaller circuits at the first stage. Consequently, more blocks are necessary to instantiate all cells. The total number of instances with P-variants was 10,538 (232 for the first stages + 10,306 for second stages), while it decreased to 8,074 (214 + 7,860) with N-variants and to 7,680 (208 + 7472) with NP-variants. The conclusion is that the extra flexibility given by the use of NP-variants allows obtaining a more efficient intermediate code, which minimizes the overall number of instances. Additionally, no cells were repeated to conclude the last block in the NP-version; while in the N- and P-versions, 6 and 24 cells were re-used to finish this task, respectively.

Table II – Construction of the first stage of the combinational block in Fig. 1.

Inputs			3 cells			C4 A1(IN(1)),A2(not_IN(2))				C4 A1(IN0),A2(not_IN(2))			
IN(3)	IN(2)	IN(1)	W_3	W_2	W_1	W_4	W_3	W_2	W_1	W_4	W_3	W_2	W_1
0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	1	0	1	1	1	0	1	1
0	1	0	1	1	0	0	1	1	0	1	1	1	0
0	1	1	0	1	0	1	0	1	0	1	0	1	0
1	0	0	0	1	0	1	0	1	0	0	0	1	0
1	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	1	1	0	1	1	1	0	1	1	1	0	1

Table III - Truth table used to synthesize the second stage of the combinational block, illustrated in Fig.1 and Table II.

In(3)	***In(2)***	***In(1)***	W_4	W_3	W_2	W_1	***Out(3)***	***Out(2)***	***Out(1)***
No vector			0	0	0	0	X	X	X
No vector			0	0	0	1	X	X	X
1	0	0	0	0	1	0	1	0	0
1	1	0	0	0	1	1	1	1	0
No vector			0	1	0	0	X	X	X
No vector			0	1	0	1	X	X	X
No vector			0	1	1	0	X	X	X
No vector			0	1	1	1	X	X	X
No vector			1	0	0	0	X	X	X
No vector			1	0	0	1	X	X	X
0	1	1	1	0	1	0	0	1	1
0	0	1	1	0	1	1	0	0	1
1	0	1	1	1	0	0	1	0	1
1	1	1	1	1	0	1	1	1	1
0	1	0	1	1	1	0	0	1	0
0	0	0	1	1	1	1	0	0	0



(a) (b)

Figure 2 - Number of cells per block according to the permutation (P) and negation (N), or both (NP) procedures in the logic gate inputs, considering a set of 208 cells, with up to 7 inputs: (a) first and (b) second stage complexity.

The computation time for block generation was 25,087 sec for the NP variants, 15 sec for the N variants and 432 sec for the P variants, by using a 1.8GHz dual-core processor, with 4Gb SRAM and 1Mb cache.

The number of cells in the second stage can be observed in Fig. 2b. It was mapped using the ABC tool [15]. This stage can be considered as the overhead to re-create the block input vectors from the intermediate code. Notice that increasing the quantity of cells at the first stage (length of the intermediate code) increases significantly the size of the second stage, generating an area overhead. This is the motivation to reduce the number of cells while generating the first stage. Also, the number of input/output pins in the block should be the minimum required, which is the largest number of inputs in a single cell in the set. This requirement is necessary to provide test controllability for the cell.

4. Testbench: Overall Architecture and Operation Modes

The combinational blocks, described in the previous section, guarantee the functional validation of the entire set of logic cells if each cell has been instantiated at least once in the first stage of a block, and the 2^n input combinations are applied at 'n' inputs of each block. Once the block outputs reproduce the same input signals, they can then be arranged in different ways to create the circuit architecture. Long paths can be built, for instance, by cascading the combinational blocks in chain configuration. This way, the primary input values should be observed at the output of chain in the case of fault free behavior.

The global circuit architecture is presented in Fig. 3. To provide a sequence of test vectors with minimum external intervention, the signals at the end of the chain are reconnected to the primary inputs. A register barrier, composed by D-type flip-flops, is added to the feedback path to avoid racing. An adder is available to increment the binary vector and make the circuit act as a counter to modify the feedback signals and provide the chain input variation. The adder can perform sums by 'K' allowing other than just a counting 1-by-1 operation, allowing thus different vector transitions that are important to check charging and discharging conditions at internal nodes intra- and inter-cells.

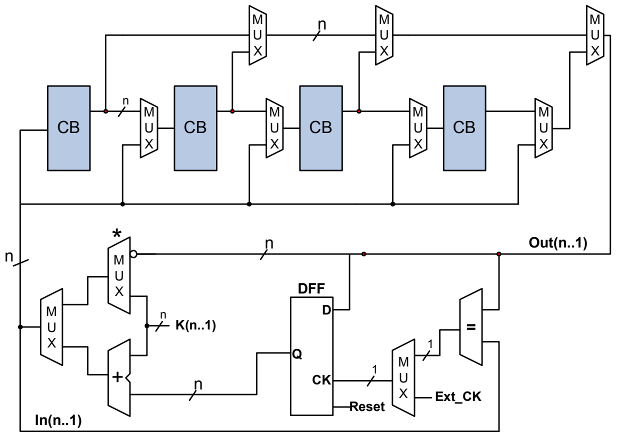


Figure 3 – Block diagram of the proposed circuit architecture.

The basic architecture is slightly modified by adding a comparator and multiplexers to allow different operating modes which provide distinct forms of data evaluation. Notice that, the aim of this circuit is to validate the full functionality of the entire set of cells, as well as evaluate the accuracy of the electrical characterization values of the cells (timing and power dissipation data) through the correlation of the static timing analysis (STA) and power analysis with experimental measures.

The circuit operating modes and their individual contributions to meet the five requirements described in Section 2 are discussed in detail in the next sub-sections. They are:

- synchronous mode;
- asynchronous mode;
- oscillation BIST mode;
- diagnosis mode.

4.1 Synchronous Mode

In the synchronous operating mode, the register barrier is controlled by an external clock signal, i.e. the 'Ext_CK' signal indicated in Fig. 3. The adder is used to increment the vector in the ring, acting as a synchronous '+K' counter. The right behavior of the counting sequence demonstrates the correct functionality of the combinational blocks and, consequently, the whole set of cells under test.

The maximum operation frequency of the circuit, which indicates the worst case path delay, can be obtained by increasing the clock frequency until the ring counter gives an erroneous result. Such critical path delay will be probably different by changing the increment value 'K'.

Another significant benefit of the synchronous mode is the evaluation of the power dissipation, including its dynamic and static components. The external control of the clock signal imposes the frequency operation for switching, and the dynamic power consumption can be related to that. The static power, on the other hand, can be measured at low frequencies or even by using an external clock manually controlled. At each new input state in the chain the static consumption can be obtained since such power dissipation component depends strongly on the circuit steady state.

Fig. 4 shows the electrical simulation of the circuit for both synchronous (before 100ns) and asynchronous (after 100ns) modes. According to Fig. 3, the 'CK' waveform is the internal clock signal; the 'ctrl_CK' signal is the multiplexer control to switch between an external clock signal and the comparator output (asynchronous clock); the 'In(1)' to 'In(4)' signals represent the 4-bit data in the ring counter; and the 'I(vdd)' waveform shows the power supply current. Notice that in the synchronous mode the supply current allows the identification of dynamic and static dissipation components, while in the asynchronous mode the circuit is always in dynamic operation.

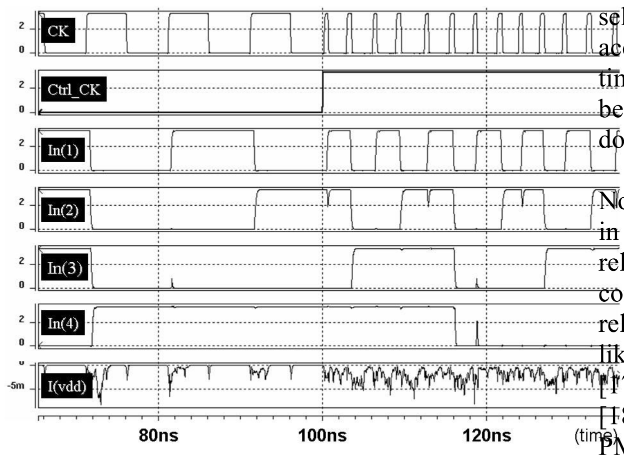


Figure 4 – Circuit behavior at synchronous and asynchronous modes.

4.2 Asynchronous Mode

In the asynchronous mode, or self-timed ring configuration, the clock signal of the flip-flops is provided by the comparator that checks whether or not the input vector In(n..1) has already propagated to the end of the chain Out(n..1) (Fig. 3). When the same vector applied to the circuit inputs get to the end of the chain, the comparator switches from '0' to '1', clocking the register. The new data is stored in the register and passed to the adder. The adder increments the register output and applies the new vector to the chain. At this moment, since In(n..1) no longer equals Out(n..1), the comparator output is back to '0' and remains at this state until the new vector propagates through the whole chain of combinational blocks. This behavior can be observed at the right side of Fig. 4.

In this operation mode, just an external signal transition is enough to start the self-timed counting, and the right operation keeps the circuit running. For instance, the starting of the self-timed operation can be achieved by modifying an external adder input bit 'Ki'. If a cell is defective, the data at the end of the chain will not be equal to the data at the circuit inputs. This way, the comparator will not switch to '1' and the self-timed execution will stop, as verified in Fig. 5.

The self-checking property of the asynchronous mode makes it quite appropriate for functional cell verification with least external intervention. The correct logic operation can be checked by just monitoring the internal clock signal or only one data bit in the loop path.

Timing information can be extracted from complete '+K' counting cycles. This measure represents the average logic path delay since in a self-timed circuit the speed is as fast as possible according to the delay of each transition, or the time to finish a computation. These measures can be used to validate the models if a simulation is done in the same operating mode.

Notice that a set of circuits working continuously in this mode can be used to check the library reliability and robustness against degradations. A continuous operation without errors proves the reliability of the cells against a number of issues like insufficiency of contacts, electromigration [17], negative bias temperature instability (NBTI) [18, 19], or other causes. NBTI is a degradation of PMOS transistors that depends on the amount of time the output of the cell is connected to Vdd; see [18, 19] for details. Notice that the circuits will not need expensive equipments to be monitored, once defects are self monitored, as shown in Fig. 5. Even if the circuits do not fail after being continuously stressed for a long period of time, they can be used to measure performance degradation caused by NBTI.

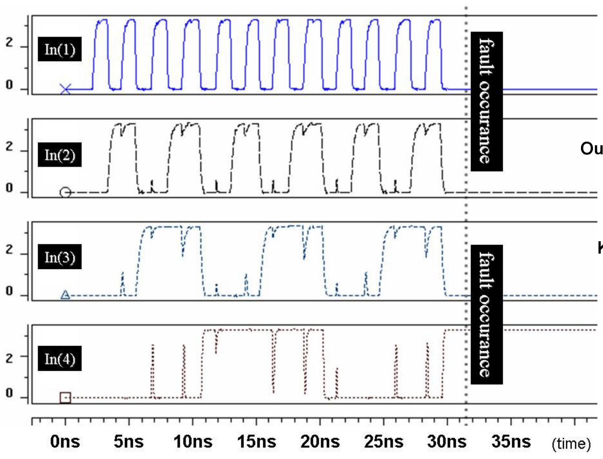


Figure 5 – Electrical simulation of fault occurrence during the self-timed counting (asynchronous mode).

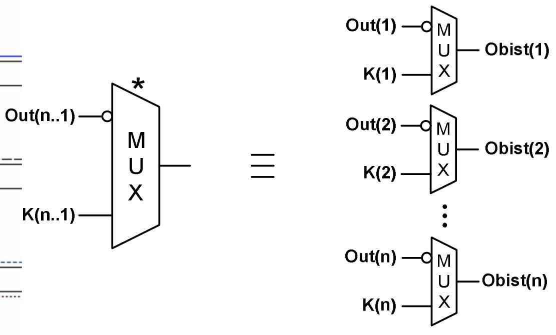


Figure 7 – Multiplexer for oscillations BIST mode.

4.3 Oscillation BIST

In synchronous or asynchronous mode, the same binary value of an i-index input is expected to re-appear at the corresponding i-index output of the testbench circuit. This property is ensured by construction of the combinational blocks. If the i_{th}-output is directly connected to the i_{th}-input, the i_{th}-path is kept in steady state, while closing the feedback loop. However, when the i_{th}-output is inverted before connecting to i_{th}-input, a negative polarity logic feedback occurs and the i_{th}-path oscillates. According to this principle, in oscillation mode the feedback loop is closed such that at least one of the primary inputs of the chain receives the negation of its previous value, as illustrated in Fig. 6.

The bit inversion necessary to ensure the oscillation condition is implemented through the inverter placed at the input of the multiplexer in Fig. 3, indicated by an ‘*’. This multiplexer is depicted in Fig. 7. Note that only one bit from the vector Out(n..1) is selected at a time to provide only one bit oscillation. The other ones are then fixed by the value provided in the input In(n..1). That is, if a single input-output pair is considered, just the associated path will oscillate, and all others are kept in steady state.

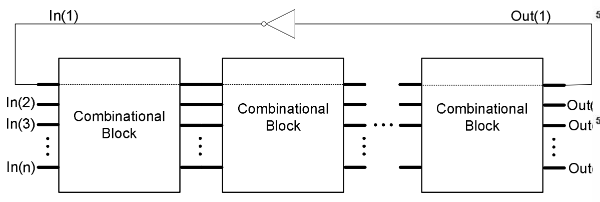


Figure 6 – Oscillation BIST path.

Notice that, the internal path followed by the oscillating signal through the chain of combinational blocks will depend on the static binary values to which the steady-state inputs of the first block are set. This feature allows configuring a wide range of different oscillating paths. Fig. 8 shows the electrical simulation of the circuit in oscillation BIST mode. While the bit ‘In(1)’ is kept oscillating, the others are modified at lower frequency to cover all the eight possible combinations. At the end of this exercise, the oscillation is switched to the bit ‘In(3)’. Table IV gives the signal period of the oscillating bit for all different states illustrated in Fig. 8.

The main contribution of the oscillation BIST mode to the validation process is that it makes possible to measure the signal delay propagation through different logic paths involving different cell instances. These delay results are then compared to those obtained from a previous static timing analysis, which took into account cell timing data from characterization (possibly done by electrical simulations). This way it is possible to correlate the models used for design with real silicon behavior.

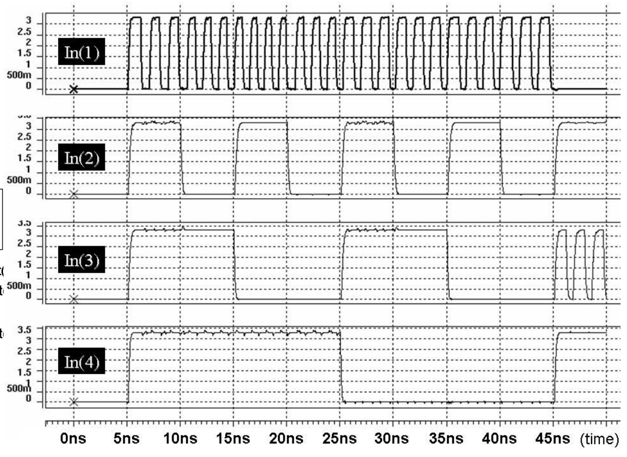


Figure 8 – Electrical simulation of oscillation BIST.

Table IV - Data of the oscillation BIST mode simulation, illustrated in Fig. 8.

Interval (ns)	(b_1,b_2,b_3,b_4)	Period (ns)
5-10	$(b_1,1,1,1)$	1.76
10-15	$(b_1,0,1,1)$	1.47
15-20	$(b_1,1,0,1)$	1.35
20-25	$(b_1,0,0,1)$	1.49
25-30	$(b_1,1,1,0)$	1.65
30-35	$(b_1,0,1,0)$	1.59
35-40	$(b_1,1,0,0)$	1.58
40-45	$(b_1,0,0,0)$	1.75
45-50	$(0,1,b_3,1)$	1.74

4.4 Diagnosis Mode

To perform defect diagnosis, additional multiplexers can be included in the circuit architecture, at no significant penalty in area, to select part of the combinational chain. Multiplexers at the inputs of each combinational block can select the signal from the previous block or directly from the beginning of the chain, removing the influence of the previous blocks in the ring loop. Similarly, multiplexers at the output of the blocks can send the data from the middle of the chain directly to its end. This way, the chain can be easily reduced to a single block or even none, allowing in this case the verification of the counter and the register barrier without the influence of the combinational chain.

This mode provides the diagnosis of a faulty block, and it also allows the different circuit operation modes, described above, to run through part of the entire chain. To exemplify this mode, the oscillation BIST was exercised for three conditions: (1) the complete chain; (2) half of the chain; and (3) one-fourth of the chain. Table V shows the data for these oscillation conditions.

The diagnosis can also be performed in open ring configuration, as the multiplexer used to provide the oscillation BIST mode (see Fig. 7) can also be applied to interrupt the ring configuration without additional circuitry. The open chain mode allows the external control of the signal stimuli that are sent to the combinational blocks chain. It is quite useful for the identification of a faulty cell.

Table V – Electrical simulation data of the diagnosis mode, considering one data bit in oscillation mode.

Interval (ns)	Chain configuration	Period (ns)
15-35	complete chain	4.07
35-45	half-chain	2.52
45-60	one-fourth chain	1.64

5. Overhead Analysis

The testbench circuit proposed here can be integrated in the same die with an ASIC, resulting in an area overhead as illustrated in Fig. 9. Two experiments have been done to evaluate this overhead.

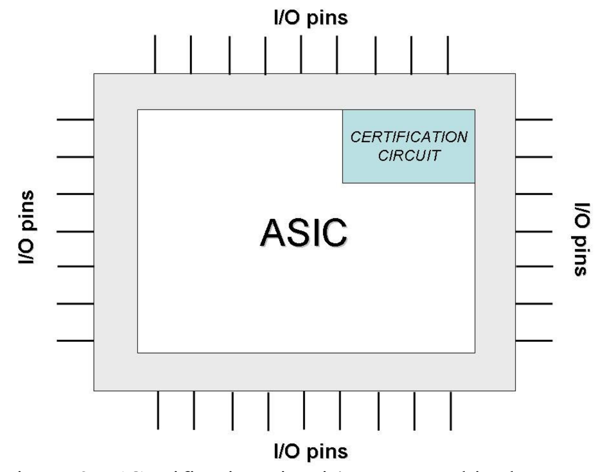


Figure 9 – 'Certification circuit' prototyped in the same die of the target ASIC.

In the first experiment, the set of 64 cells previously mentioned in Table I was considered. This set is composed of cells with up to 4 inputs. The number of input/output bits for the generation of combinational blocks was kept at four bits to ensure the controllability of the instantiated cells. Eight combinational blocks were automatically generated to build a circuit having all library cells instantiated in the first stage of blocks. The complete circuit used around 500 instances. The testbench circuit ensures the verification of all 64 cells in the library, with an average of 7.03 instances per cell from the library. Notice that, the 'tv80_core' circuit in Table I, which used all 64 cells of the library on its implementation, has 5,597 cell instances. Suppose that the verification circuit generated here was the test vehicle to verify the library used to implement the 'tv80_core' circuit. In this case, an overhead of 8% in terms of cell instances would be produced. However, if 'wb_conmax_top' was chosen to validate the library, a circuit with 28,089 instances would be produced and yet not all the cells from the library would be instantiated.

The second experiment was the design of a testbench circuit to test a library containing 208 cells, mainly CMOS complex gates, with up to 7 inputs. This example resulted in a testbench composed of around 8,000 instances, to validate the complete set of cells. The number of instances per cell library is around 16, which is still acceptable. This can be considered a very small circuit to test such a huge group of cells once an ASIC requiring such amount of cells can easily have more than 100k instances.

Therefore, the test circuit brings little silicon overhead to the final design, allowing the fabrication embedded with the target ASIC, as illustrated in Fig. 9.

The overhead in terms of I/O pins is described in the following, considering n-bit combinational blocks. The value 'n' represents the maximum number of inputs in a single cell to be validated. In this case, the final validation circuit presents:

- 'n' input signals – K(n..1);
- 'n' output signals – Out(n..1);
- one external clock signal;
- one D-type flip-flop reset signal;
- some multiplexer control signals, being that the number can be reduced by using a decoder circuit since some multiplexers are dependent on each others.

Depending on the complexity of the target ASIC, such a pin overhead may prove unaffordable for certification at the chip level. In this case, considering its little silicon overhead, the testbech could be though as a certification vehicle at the wafer-level and be used to validate sets of neighbouring ASICs.

From this analysis, one can conclude that the testbench methodology represents a low cost solution for validating all soft-cells included in an ASIC, and may act as a library certification circuit at chip or wafer-level. In terms of EDA vendor interest, the proposed methodology represents an efficient way to validate soft-libraries by using very compact circuits.

6. Conclusions

The use of automatically generated CMOS logic gates in standard cell IC design flow represents an attractive perspective for ASIC design quality improvement. Automatic cell generators can be considered as soft IPs and represent the key elements for the library-free technology mapping approach, already proposed in literature and now being adopted by the industry. This methodology leads to an IC design flow based on logic cells which are created on-the-fly by software, which have not been previously validated on silicon yet, until the target ASIC is prototyped. This fact makes conservative customers reluctant in adopting this design technique. This work proposed a validation methodology to silicon prove the set of automatically designed logic cells. The validation covers the full functionality of the cells and provides means to compare cell data models derived from electrical characterization against silicon performance. This methodology allows verifying the reliability of the cell models to be used in the performance estimation during the circuit design. A method to automatically generate the validation circuits from an initial set of cells was also proposed. The method has low area overhead and the several operation modes. Medium and long term standalone runs allow verifying the circuits for degradation effects like NBTI and electromigration. Self-testing modes allow performing medium and long term reliability certification without depending on ATE, which allows to reduce costs and to increase the number of samples under test.

References

[1] P. de Dood, B. Lee and D. Albers. "Optimization of circuit designs using a continuous spectrum of library cells". US Patent 7107551. 2003.

[2] S. Gavrilov, A. Glebov, S. Pullela, S.C. Moore, A. Dharchoudhury, R. Panda, G. Vijayan and D. T. Blaauw, "Library-less synthesis for static CMOS combinational logic circuits", ICCAD 1997, pp.658-662.

[3] D. Kagaris and T. Haniotakis. "Methodology for transistor-efficient supergate design". IEEE Trans. VLSI, Apr. 2007, vol. 15, no. 4, pp. 488-492.

[4] F. S. Marques, L. S. Rosa Jr, R. P. Ribas, S. S. Sapatnekar and A. I. Reis. "DAG based library-free technology mapping". GLSVLSI 2007, pp. 293-298.

[5] F. R. Schneider, R. P. Ribas, S. S. Sapatnekar and A. I. Reis, "Exact lower bound for the number of switches in series to implement a combinational logic cell". ICCD 2005, pp.357-362.

[6] R. Roy, D. Bhattacharya and V. Boppana. "Transistor-level optimization of digital designs with flex cells". IEEE Computer, Feb. 2005, vol. 38, no. 2, pp. 53- 61.

[7] A. I. Reis, F. R. Schneider and R. P. Ribas. "Methods of deriving switch networks". US Patent Application #20070214439.

[8] D.Bhattacharya, V.Boppana, R.Roy and J.Roy. "Method for automated design of integrated circuits with targeted quality objectives

using dynamically generated building blocks". US Patent 7,225,423. May 29, 2007.

[9] L.G.Jones, D.T.Blaauw, R.L.Maziasz and M. Guruswamy. "Method and apparatus for designing an integrated circuit". US Patent 5,666,288. Sep. 9, 1997.

[10] EEtimes. "NXP adopts Nangate's tool for 32-nm research". By: A.-F.Pele, EE Times Europe, 02/19/2008, available on: http://www.eetimes.eu/france/206800280?cid=RSSfeed_eetimesEU_france.

[11] H. Yoshida, M. Ikeda, and K. Asada. "A structural approach for transistor circuit synthesis". IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E89-A, 12 (Dec. 2006), 3529-3537.

[12] S. Long, "Test structures for propagation delay measurements on high-speed integrated circuits", IEEE Trans. Electron Devices, vol. ED-31, no. 8, Aug. 1984.

[13] E. Arabi , H.n Ihs I, C. Dufaza and B. Kaminska, "Digital oscillation-test method for delay and stuck-at fault testing of digital circuits", ITC 1998, pp. 91-100.

[14] J. Kibarian, "Overcoming the process variability crisis via proactive DFM". Keynote talk at ICCAD 2007.

[15] Berkeley Logic Synthesis and Verification Group. "ABC: A system for sequential synthesis and verification". Dec. 2005 Release. Berkeley. http://www-cad.eecs.berkeley.edu/-alanmi/abc

[16] Mentor Graphics Design-for-Test (DFT) Tools. http://www.mentor.com/products/dft/

[17] C. J. Christiansen, B. Li, J. Gill, R. Filippi and M. Angyal. "Via-depletion electromigration in copper interconnects". IEEE Trans. Device and Materials Reliability, vol. 6, no.2, pp. 163–168.

[18] S. V. Kumar, C. H. Kim, S. S. Sapatnekar, "NBTI-aware synthesis of digital circuits". DAC 2007. pp. 370-375.

[19] D.K.Schroder. "Negative bias temperature instability: What do we understand?" Microelectronics Reliability, Volume 47, Issue 6, June 2007, Pages 841-852.