UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LEONARDO IRI NICOLA CRISTOFARI ROSENBACH

Domino Logic Library Design and Logic Synthesis

Trabalho de Diplomação.

Prof. Dr. André Inácio Reis
Orientador

Porto Alegre, 24 de junho de 2011.

ACKNOWLEDGEMENTS

I would like to thank André Inácio Reis for the valuable time spent with me and insight conversations (sometimes drinking a Carlsberg at Nangate's kitchen in Denmark!) beside believing on my ideas and trying to achieve something more realistic than what I initially dreamed of (which I would not get even close to accomplish on time!).

I have pretty sure that all the engineers at Nangate had a huge participation on my growth as an academic student and as a worker. Thanks for sharing the knowledge in Microelectronics over the last four years and for the things we learn how important they are on the industry.

I think I'll be eternally grateful to the professors at UFRGS whom were dedicated and taught most of the things that somehow made me choose to pursue the Microelectronics field and decide to write about this topic on this document.

I would like to thank my mother (Jane Nicola) whom never gave up on me on the most difficult times and my father (Luiz Rosenbach) whom always made me notice how important is to have a good education and helped me to keep focus on the job.

During the period that I was writing this document, I had help reviewing the document from Carla Lopes. Thanks a lot for that.

And finally but no least important, I´d like to say thanks to my family and friends for being very supportive.

SUMMARY

LIST OF ABBREVIATIONS **6**
LIST OF FIGURES **7**
LIST OF TABLES **8**
ABSTRACT **9**
RESUMO **10**
1 INTRODUCTION **11**
1.1 Motivation **11**
2 BASIC CONCEPTS **13**
2.1 Static Complementary CMOS Gate Design **13**
2.2 Dynamic CMOS Gate Design **13**
2.2.1 Charge Leakage 14
2.2.2 Charge Sharing 14
2.2.3 Capacitive Coupling 15
2.3 Different Logic Styles Characteristics **15**
2.4 Domino Logic Style **16**
2.5 Traditional Logic Synthesis **18**
3 DOMINO LOGIC LIBRARY DESIGN **19**
3.1 Sizing Domino Gates **19**
3.2 Keeper Transistors **19**
3.3 Additional Pre-charge Transistors **20**
3.4 Multi Output Domino Gates **21**
3.4.1 Fast Adder Gates 21
4 DOMINO LOGIC SYNTHESIS **22**
4.1 Binate to Unate Conversion **22**
4.1.1 Logic Optimization by Output Phase Assignment 22
4.1.2 Bubble Pushing 23
4.2 Technology Independent Optimization **23**
4.3 Technology Mapping **23**
4.3.1 Using Complex Static Gates 24
5 PROJECT SPECIFICATION **25**
5.1 Logic Synthesis Tool Structure **25**
5.2 Supported RTL Format **26**
5.3 Supported Spice Format **27**
6 IMPLEMENTATION AND ALGORITHMS **28**
6.1 Circuit Loader (Equation Parser) **28**
6.2 Technology Independent Transformations **28**
6.2.1 De Morgan 29
6.2.2 Inverter Propagation and Redundancy Clean 29

6.2.3 Removal of input inverters 29
6.2.4 Removal of duplicated inverters 30
6.2.5 Logic Sharing 30
6.3 Virtual Mapping 31
6.3.1 Grouping of gates 31
6.3.2 Virtual mapping cuts 31
6.4 Binate to Unate Conversion 32
6.4.1 Logic Duplication 32
6.4.2 Bubble Pushing 33
6.5 Circuit Exporter 33
6.6 Spice Generation 34
6.6.1 Stack Sizing Technique 34
6.7 UML Class Diagram 35
7 RESULTS 37
7.1 Mapping Analysis for Domino 37
7.2 Overall mapping comparisons 38
7.2.1 Comparing to Elis 38
7.2.2 Comparing to ABC 40
7.3 Circuit Verification 41
7.4 Timing Analysis using Spice Simulations 41
8 CONCLUSIONS 42
REFERENCES 43

LIST OF ABBREVIATIONS

UFRGS	Universidade Federal do Rio Grande do Sul
CMOS	Complementary Metal-Oxide Semiconductor
ASIC	Application-Specific Integrated Circuit
EDA	Electronic Design Automation
RTL	Register Transfer Level
PUP	Pull-Up Transistor Network
PDN	Pull-Down Transistor Network
PMOS	P-type Metal-Oxide Semiconductor
NMOS	N-type Metal-Oxide Semiconductor
CLA	Carry Look-Ahead Adder
DS	Dynamic-Static
IEEE	Institute of Electrical and Electronics Engineers
ACM	Association for Computing Machinery
OAI	Or-And-Inverter gate
UML	Unified Modeling Language
I/O	Input/output

LIST OF FIGURES

Figure 2.1 - Charge sharing at a NAND4 dynamic gate. 14
Figure 2.2 - NOR2 in static (left) and dynamic (right) CMOS styles. 15
Figure 2.3 – Cascade of Domino logic gates. 17
Figure 2.4 - Logic Synthesis flow. 18
Figure 3.1 - Weak keeper (left) and Feedback keeper transistor (right). 20
Figure 3.2 - Adding precharge transistors to the NAND4 Domino logic gate. 20
Figure 3.3 - Typical multi output Domino gate. 21
Figure 4.1 – A reconvergent fan-out net with a trapped inverter. 22
Figure 4.2 - DS Domino Logic gate. 24
Figure 5.1 - Domino logic synthesis flow overview. 25
Figure 5.2 – An example of the Equation format (EQN). 27
Figure 5.3 –Transistor level description of Nand2 gate in Spice format. 27
Figure 6.1 – Two steps to parse equation to N-ary Tree. 28
Figure 6.2 – Inverter propagation and logic redundancy clean. 29
Figure 6.3 – Input inverter removal on gate level making new inverter gates. 30
Figure 6.4 – Circuit containing double inverters. 30
Figure 6.5 – Demonstration of how duplicated logic is created and then shared. 31
Figure 6.6 : Virtual mapping cuts on to respect constraints. 32
Figure 6.7 – Logic duplication in the cone where two different phases are assigned. ... 32
Figure 6.8 – Bubble Pushing method propagating inverters to primary inputs. 33
Figure 6.9 – Building an OAI221 equation. 34
Figure 6.10 – Building an OAI32 transistor network. 34
Figure 6.11 – Stack sizing method applied on the N-ary Tree. 35
Figure 6.12 – Overview of the UML Implementation Model. 36

LIST OF TABLES

Table 2.1: Comparison of Static versus dynamic logic styles characteristics. 16
Table 2.2 : Domino logic style characteristics. 17
Table 7.1 : Average fan-in of mapped circuits 38
Table 7.2 : Comparison of Gates and Area between Elis and DS. 38
Table 7.3 : Comparison of Worst Depth, Fan-in and Fan-out between Elis and DS. 39
Table 7.4 : Comparison of Gates and Area between ABC and DS. 40
Table 7.5 : Comparison of Worst Depth, Fan-in and Fan-out between Elis and DS. 41

ABSTRACT

Domino is an interesting logic style for achieving high speeds and area efficient design due to reduced number of transistors, reduced fan-in capacitance and faster switching thresholds. However, domino logic brings new challenges to the logic library design as well as to the ASIC design flow. Logic synthesis is one of the steps which is highly impacted due to the non-inverting nature of the gates and may be a crucial for reaching the high performance ASIC design. This is the reason why this document addresses the issues of designing a domino logic library and performing domino logic synthesis. A different logic synthesis flow to resolve such problems is proposed.

Keywords: domino logic, logic synthesis, high performance ASIC design.

Design de Bibliotecas para Lógica Dominó e Síntese Lógica

RESUMO

Dominó é um estilo de implemetação de portas lógicas interessante para obter um design com altas velocidades e que é eficiente em área devido ao reduzido número de transistores e baixa capacitância de entrada. Porém este estilo traz novos desafios na criação da biblioteca de células lógicas assim como para o fluxo de projeto ASIC. A síntese lógica é um dos passos que é fortemente impactado devido à natureza não-inversora das portas lógicas e pode ser crucial para atingir o alto desempenho esperado do projeto. É por esta razão que este documento apresenta os problemas enfrentados para criar uma biblioteca para lógica dominó e realizar a síntese lógica voltada a dominó. Um fluxo especializado para síntese lógica dominó é proposto para resolver tais problemas.

Palavras-chave: lógica dominó, síntese lógica, projeto ASIC de alto desempenho.

1 INTRODUCTION

The invention of the transistor at Bell Telephone Laboratories in 1947 followed by the conception of the first integrated circuit gave birth to a new era in the digital electronics systems (RABAEY,2002), an era where all the components are integrated into a single semiconductor substrate. After that, the evolution of speed and density on the integrated circuits never stopped.

The growth in complexity of the integrated circuits leaded to a revolution on how digital circuits are designed. The ability to design and verify an integrated circuit was unfeasible due to the large number of transistors. As a consequence, newer classes of tools to assist on these tasks called EDA tools started to be developed.

The capabilities and the range of EDA tools have been increasing. The EDA tools for logic synthesis are really powerful today. However, the majority is limited to the usage of a fixed static cell library built for static complementary CMOS design and traditionally does not support dynamic logic styles.

Domino is an interesting dynamic logic style for high performance ASIC design due to inherent characteristics which are reduced number of transistors, reduced load per fan-in, fast switching thresholds, glitch free operation. However, some care should be taken when building the domino logic gates because charge sharing, charge leakage and capacitive coupling may be a problem.

The major issue of using domino logic, at the logic synthesis point of view, is the non-inverting nature of the gates. Conventional logic synthesis approach used to convert an RTL description into a gate level using complementary CMOS design is incompatible to domino logic because the network has to be inverter free before the technology mapping.

Algorithm to make an inverter free network may have the penalty of logic duplication which it has to make as minimum as possible or may judge that the penalty does not worth the logic duplication and decide to have a mixed domino/static design and the next steps will have to respect that decision.

1.1 Motivation

First of all, I always followed the Microelectronics field during my academic life and worked over three years in the development of EDA tools. The university introduced to me to the basic concepts on Microelectronics. Three years working in an EDA company made my knowledge on this area greatly increase. The fascination for logic optimization methods and standard cell library creation improvements lead me to

start thinking about the usage of different logic styles and how it would impact in the ASIC flow, to be more specific at the logic synthesis step.

Conversations with colleagues and teachers about related topics triggered the desire to understand why such promising area was not been that much explored. Then I found out that there are companies working with other logic styles than static complementary CMOS and a lot of researches on the field. Recently a company which has a set of EDA tools for re-implementation of circuits using dynamic logics claiming to be able to create high performance and low power designs was acquired by one of the big companies on the market of consumer electronics, computer software, and personal computers (VANCE, 2010).

2 BASIC CONCEPTS

This chapter describes the basic concepts necessary to understand this work. Initially, Section 2.1 presents static CMOS style, while Section 2.2 discusses dynamic logic. Section 2.2 also gives an overview of different design issues found in dynamic logic styles. Static and dynamic styles are then compared in Section 2.3. Afterwards, Section 2.4 provides a deep overview of the Domino dynamic style addressed in this work. This deeper description of domino logic is done to later on create a logic synthesis tool which takes into consideration the limitations and the characteristics of Domino logic. The final section is reserved to present usual (traditional) logic synthesis flow and the steps they perform during the synthesis process.

2.1 Static Complementary CMOS Gate Design

The most common and spread logic style is the static complementary CMOS. The word *static* refers to the fact that each output of the gate is connected to either the high voltage power (VDD) bar or to the low voltage power bar (GND) at a certain point in time. The word *complementary* refers to the fact that the logic inputs (of the gate) will select the specific power bar that will be connected to the gate output. This way, the connections between the output and GND or VDD are logically *complementary*.

The PMOS transistors which connect the outputs to the high voltage level are the components of the so called pull-up network (PUP) and the NMOS transistors of the gate are the components of the pull-down network (PDN) which are responsible of connecting the outputs of the gate to the low voltage level. The non-glitch free characteristic of the complementary static CMOS gates is a potential problem causing some gates to have non-steady transitions.

2.2 Dynamic CMOS Gate Design

An alternate logic style is the dynamic CMOS gates. This logic style incorporates the clock input in all gates. The operation of these gates is divided into two phases which explains a lot of how those gates are built as well. The phases are called pre-charge and evaluation. In the pre-charge phase, the gate outputs are charged to the high level voltage because the PMOS transistors are controlled by clock input which in this phase is low. In the evaluate phase, the outputs of the gate can conditionally change to a low voltage level. The logic of the gate is implemented only with NMOS transistors, those transistors dictate if the outputs will be connected to the low voltage level to be discharged or not.

Some characteristics of the dynamic CMOS gates which are a potential problem (despite that pure dynamic gates cannot be cascaded) are:

- Charge leakage and charge sharing;
- Capacitive coupling;

The next sections address those problems with more details, explaining why they happen.

2.2.1 Charge Leakage

The operation of the dynamic CMOS logic style gates are based on the pre-charge of the capacitor at the output of the gate. The capacitor charged has to remain on its state during all the evaluate phase if the PDN network is off. Due to the leakage currents in the circuit, that capacitor start discharging and if it reaches a voltage level lower than the allowed by the noise margins, the gate will have a malfunctioning behavior.

A series of precautions have to be taken to avoid this issue. The most effective approach is to add keeper transistors to the gate which are explained in the Section 3.2. Other precautions are required either to avoid having long periods on evaluation phase or to size correctly the pre-charge transistors.

2.2.2 Charge Sharing

Other issue in the dynamic logic style gates is charge sharing. Take as an example the Figure 2.1 and assume that initially all capacitances are discharged and that all inputs are set to zero during the pre-charge. Then, during the pre-charge phase, the output capacitance has to be charged again. If the inputs make transitions from zero to one, the charge is distributed with the internal capacitances. This causes the voltage of the output node to drop and may result in an inconsistent behavior of the logic gate. This issue just happens if the sum of internal capacitances sharing the charge is big enough to make the output capacitance reach voltage levels below the acceptable by noise margins. There are two solutions for this problem. The first is to increase the output capacitance and the other is to have additional pre-charge transistors charging the internal capacitances as is explained in the Section 3.3.

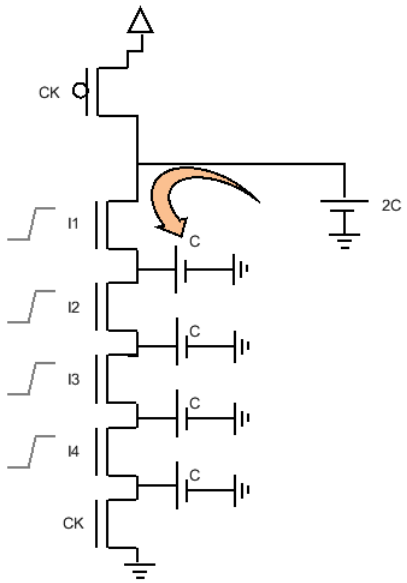

Figure 2.1 - Charge sharing at a NAND4 dynamic gate.

2.2.3 Capacitive Coupling

The capacitive coupling problem is most common to happen when the dynamic gate output is driving a multiple input gate and one of the side-inputs of the gate being driven makes a transition. As consequence, the parasitic capacitances of the transistor which is being driven suffers capacitive coupling from the side input that is transitioning. If the voltage at the output node is changed due to capacitive coupling, it may become a potential problem to the design. Attention must be paid when implementing the circuit to avoid such problems.

2.3 Different Logic Styles Characteristics

The Figure 2.2 shows how the function of NOR2 which is “F=!(A+B)” are implemented using different logic styles.

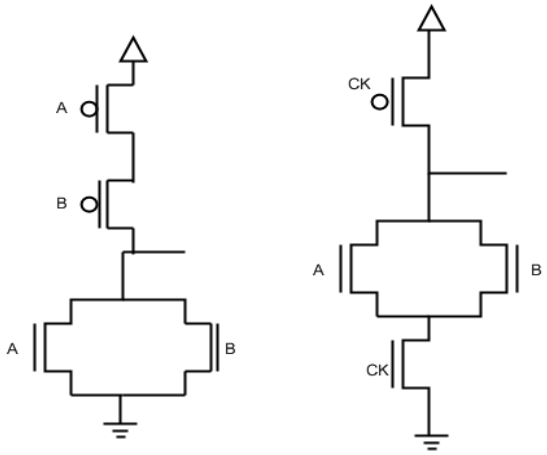

Figure 2.2 - NOR2 in static (left) and dynamic (right) CMOS styles.

The Table 2.1 presents a comparison of the static and dynamic logic styles regarding characteristics like: power consumption, noise sensitiveness, capacitance load, typical number of transistors, easiness to design and possible design issues.

Table 2.1: Comparison of Static versus dynamic logic styles characteristics.

Logic styles characteristics		
	Static	**Dynamic**
Power Consumption	Low with practically no static power consumption.	Low and no static power consumption (removed by the clock input).
Noise Sensitiveness	Low.	Reasonable.
Capacitance Load	1 PMOS + 1 NMOS input capacitance.	1 NMOS input capacitance with small size.
Number of Transistors (N Inputs)	2*N	N+2 or N+4
Design	Easy.	Complex.
Issues	Glitch.	Charge sharing, leakage and capacitive coupling.

2.4 Domino Logic Style

The dynamic gates (in general) have a problem because is not possible to freely cascade them, without taking the sequence of transitions into account. This limitation arises because dynamic gates require that only transitions in a single direction (either rise or fall, depending on the topology of the gate) are allowed in the inputs during the evaluation phase. Distinct transitions during the evaluation phase are not allowed because the next stage dynamic gate output may end up with an intermediate voltage value which do not represents high or low. This issue arises as dynamic gates are pre-charged and cannot be recharged once they have been discharged.

The Domino logic gates are obtained by attaching a dynamic gate to a static complementary CMOS gate which in most of times is the static inverter (LAW,1982). During the pre-charge phase the output node of the dynamic gate rises up to a high level and the static CMOS gate output falls to the low voltage level. Then in the evaluation phase the unique transition which can happen at the output of static gate is a single rise transition. This means that there are no problems cascading such logic gates to form a complete design.

Some characteristics of the domino logic style are presented on the Table 2.2.

Table 2.2 : Domino logic style characteristics.

Domino Logic style	
Power Consumption	No static power consumption (removed by the clock input) on the first stage. Dynamic power on the second stage.
Noise Sensitiveness	Reasonable.
Capacitance Load	1 NMOS input capacitance with small size.
Number of Transistors	N+4 (where N is the number of inputs)
Design	Complex.
Issues	Non inverting logic.
Pros	Glitch free, low load, fast switching thresholds and possibility to cascade.

The name domino comes from the behavior of a chain of these logic gates. First they are all switched to a low level (at the output of the inverter) during the pre-charge. On the evaluation phase, the outputs (of the inverters) can switch to a high level which may affect the next stage gate and so on which can be considered similar to a chain of dominoes falling.

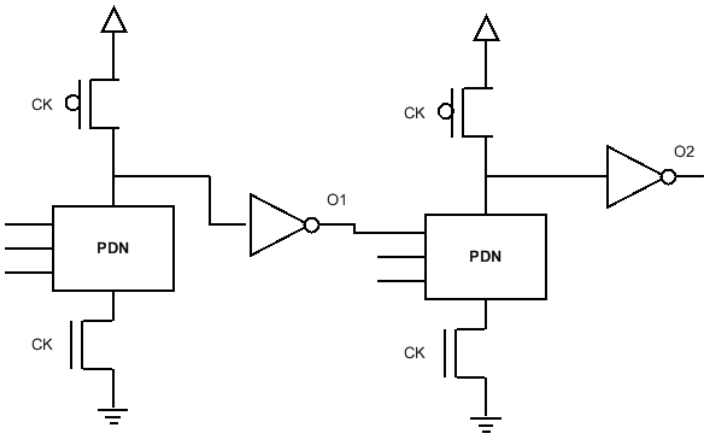

Figure 2.3 – Cascade of Domino logic gates.

2.5 Traditional Logic Synthesis

Logic Synthesis is the step in the ASIC design flow where the process of converting a RTL (Register Transfer Level) hardware description into an optimized gate level description is done.

This process contains three major sub-steps:

- **Translation** converts the RTL description to a non-optimized internal representation;
- **Technology Independent Optimization** also known as Logic Optimization removes redundant logic, and perform a series of technology independent logic optimizations;
- **Technology Mapping** is the final phase of the logic synthesis which maps the optimized logic representation into gates contained in the technology library. The gates chosen to map the logic representation may change according to the design constraints: area, timing or power.

The Figure 2.4 shows the steps of the Logic Synthesis flow and more detailed information.

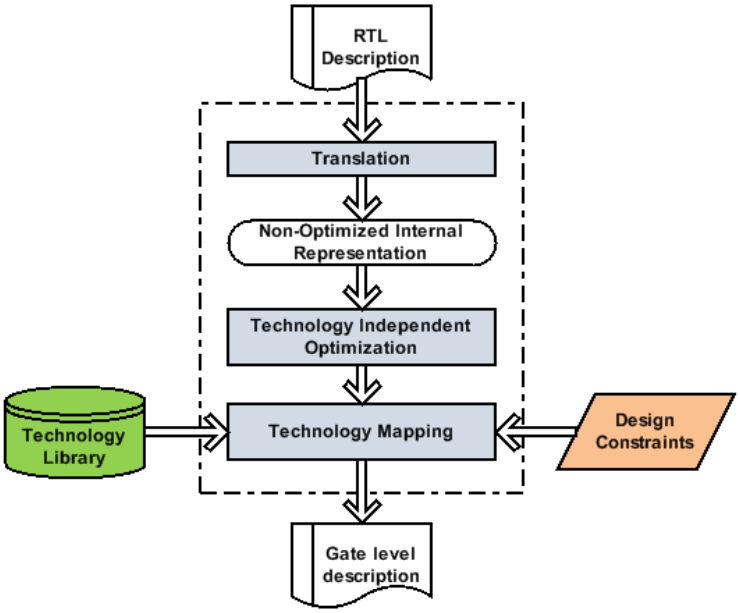

Figure 2.4 - Logic Synthesis flow.

3 DOMINO LOGIC LIBRARY DESIGN

This chapter is dedicated to describe how to create gates in Domino logic style avoiding problems like bad sizing, charge leakage, charge sharing and capacitive coupling. The last sections are reserved to present some good domino gates to have in the library.

3.1 Sizing Domino Gates

The goal of the sizing of Domino logic gates is to make the evaluate phase as quick as possible. The PMOS pre-charge transistor is typically sized as small as it can be and still is not a problem for pre-charge. The same is valid for the NMOS transistors of the static complementary stage. The NMOS transistors of the dynamic gate have to be sufficiently large to drive the input capacitance of the static complementary gate used as output stage. This is due to the fact that the NMOS transistors of the dynamic gate and the PMOS transistor of the output inverter are the transistors that affect the evaluation phase speed.

Transistor sizing of the gates is done by using a simple scheme partially derived from logical effort (SUTHERLAND, 1999). After defining the values to be applied for pre-charge and evaluation transistors, the transistors are sized according to the length of the stack they belong. This is done similarly as in a usual sizing for static complementary CMOS gate design.

3.2 Keeper Transistors

The dynamic gates have a charge leakage problem on the output node. The first approach to solve this problem is to add a “weak keeper” transistor which is a small PMOS transistor charging the output node by making a low resistive path between the dynamic gate output node and the power supply. This approach is not so good because it introduces another problem, static power consumption.

The best approach to solve this problem is to have keeper transistor controlled by the feedback of the output node of the static gate. Whenever the output of the static gate is low, the dynamic is charging because the PMOS keeper transistor is “on”. But as soon as the output of static gate changes to high, the keeper transistor is switched “off” avoiding static power consumption.

See the Figure 3.1 which explains both approaches to solve the charge leakage problem.

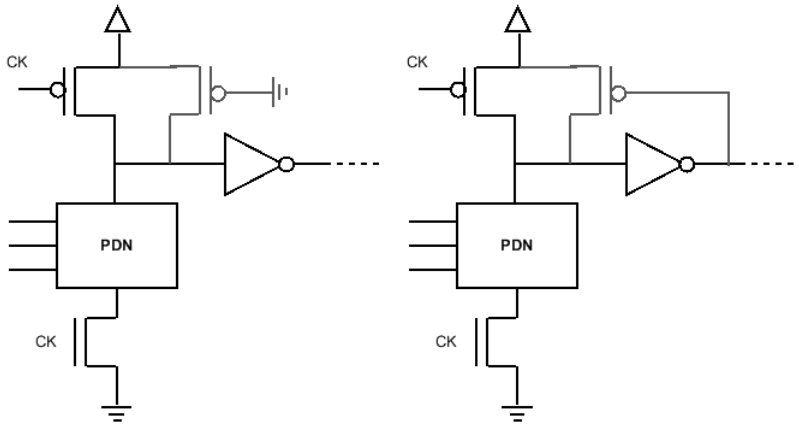

Figure 3.1 - Weak keeper (left) and Feedback keeper transistor (right).

3.3 Additional Pre-charge Transistors

The best practice to avoid charge sharing problem on the dynamic gate is to put additional pre-charge transistors to the gates containing long chain of transistors in series. These additional pre-charge transistors are connected directly to the PDN network internal nodes to make sure that those capacitances are charged during the pre-charge phase and the output capacitance will not be sharing its charge with such capacitances, see Figure 3.2.

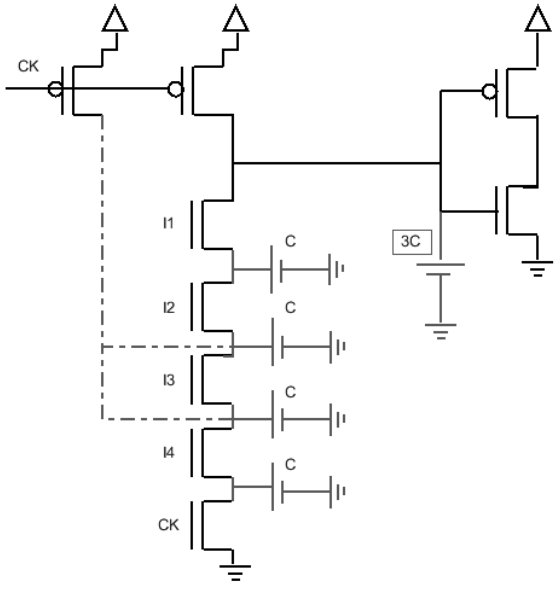

Figure 3.2 - Adding precharge transistors to the NAND4 Domino logic gate.

3.4 Multi Output Domino Gates

Larger complex gates are the key to achieve smaller area and delay during the technology mapping process when using Domino logic gates.

The Domino logic style supports multiple output gates. These gates explore the fact that the logic of some outputs may be a subset of the logic of other outputs and what could be implemented as a set of single output gates can be implemented as one multi output gate. This option results in more possibilities to use larger gates which for area saving and speed is great.

The implementation of multi output gates has the advantage of sharing the evaluation transistor but each dynamic gate output requires its own pre-charge transistor and static gate attached, see Figure 3.3. So, for each additional output, an additional pre-charge and an additional static gate are incorporated in the Domino gate.

Each pair of Domino gates merged to become a multi output gate saves at least two transistors and reduces the logic depth on that path.

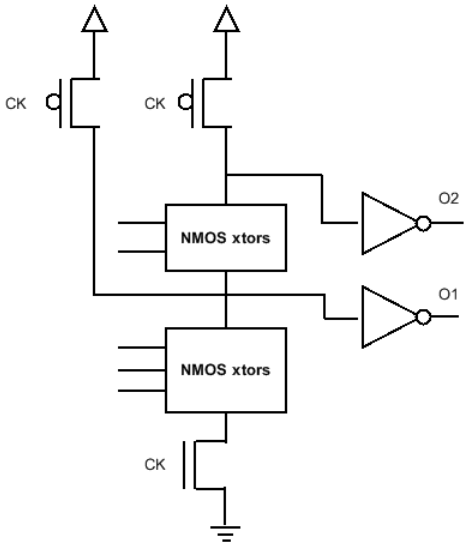

Figure 3.3 - Typical multi output Domino gate.

3.4.1 Fast Adder Gates

The usage of multi output Domino logic gates for building carry look ahead adders (CLA) makes the adder circuit to have a reduced number of stages, a smaller delay and area as well as proving that such type of gates can highly improve the performance of the final gate level description of the circuitry (BIZZAN,1997).

4 DOMINO LOGIC SYNTHESIS

The most complicated limitation of the domino logic style for performing logic synthesis is the incapacity of implementing intermediate inverting logic. This limitation imposes the addition of one step on the domino logic synthesis called removal of the intermediate inverters or also called binate to unate conversion. This step can be performed using different techniques as is described in Section 4.1. The other sections on this chapter are reserved to explain the additional steps of the domino logic synthesis.

4.1 Binate to Unate Conversion

The binate to unate conversion purpose is to make an inverter free logic. One way of achieving this is to push all the inverters to the primary inputs or primary outputs where it can be absorbed by registers and as consequence the logic becomes inverter free. Two methods to make binate to unate conversion are presented on the next sections.

4.1.1 Logic Optimization by Output Phase Assignment

The Logic Optimization by Output Phase Assignment (PURI, 1996) algorithm has the purpose of making the network inverter free (unate) by suffering the minimum logic duplication penalty possible. The algorithm presents some interesting procedures for identifying the duplicated logic region and the optimizable logic region.

The duplicated logic region is found at the fan-in node of reconvergent fan-out with trapped inverters. A reconvergent fan-out has a trapped inverter only if one of the paths has an odd number of inverters and the other path has an even number of inverters (see Figure 4.1). On these cases, the logic under this region must be duplicated. There is no solution for such cases other than duplicating the logic and afterwards pushing the inverters to the primary inputs.

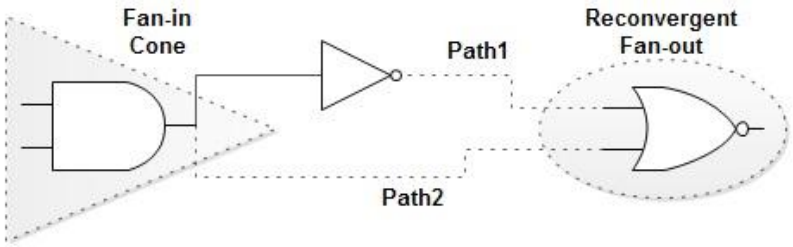

Figure 4.1 – A reconvergent fan-out net with a trapped inverter.

The important part of the network is the optimizable logic region which can minimize the logic duplication by an optimal output phase assignment. The modification of phase assign to an output can make inverters to “disappear” from the network, actually they are been pushed to the primary output. If the internal nets are imposing the same phase for that output then no logic duplication is required and inverters can be pushed to the output.

Sometimes, there are nets trying to impose different phase assignments to an output, these nets are called incompatible nets. The simple modification of the output phase assignment will just take the inverter from one net and put into the other. One of the nets will have an inverter and will require logic duplication so the best that can be done is to choose the net to be duplicated which will result in the minimum logic duplication. The choice is made by using a Boolean satisfiability framework.

4.1.2 Bubble Pushing

The Bubble Pushing (PRASAD, 1997) algorithm is a generalization of DeMorgan’s laws which transforms a binate network into an unate network. It traverses the network from outputs to inputs making all nodes unate, i.e. each time a binate node is found then a new node is created for implementing the complementary logic of the fan-in node making the previous node binate. The method is only able to push inverters to primary inputs.

This method is proposed in such way that the technology independent optimizations are applied after the binate to unate conversion which seems to be the opposite of what is proposed when using the technique described at Section 4.1.1.

4.2 Technology Independent Optimization

The logic optimizations which can be performed at this step will only depends on the network which is given. As the binate to unate conversion can be performed before or after the technology independent optimization, the network being handled can be a binate or an unate network.

If the network is binate, i.e. the binate to unate conversion was not applied then all the logic optimizations known for this step can be performed without restrictions, otherwise the network was already converted to unate then just unate optimization procedures can be performed and sometimes the last approach may result in better optimized domino circuit (PRASAD, 1997).

4.3 Technology Mapping

Technology mapping is the last step of the domino logic synthesis and is responsible for converting an optimized network into a gate level netlist. In a static complementary CMOS design, the optimized network is traditionally mapped to a fixed standard cell library (see Figure 2.4). As the Domino logic design allows the creation of huge, wide and complex gates it would be such a waste to have a fixed library for mapping the network. For this reason most of algorithms for domino logic technology mapping use the parameterized library approach (PRASAD, 1997).

"A parameterized library is defined as a collection of gates that satisfy the constraints on the width and height of the PDN and PUP implementations of the gate" (SAPATNEKAR,1998). An issue of using the parameterized library approach is that it requires a cell generator for domino logic gates. This cell generator has to be able to build a gate layout based on a description of it.

4.3.1 Using Complex Static Gates

An interesting approach in the technology mapping phase is the ability of mapping networks to domino gates using complex static gates instead of the traditional static inverter (SECHEN, 1998). During the mapping, the network has to be colored in such way that a dynamic gate is always followed by a static complex gate also called DS Domino gates (see Figure 4.2).

This result in many dynamic gates attached into a single static complex gate forming a multi-stage gate. Beside the fact that creating large gates is very good for domino logic synthesis, this approach is also good for reducing the clock loading.

This way, other possibilities available on the technology mapping for domino logic gates are the use of multi output gates and compound gates. Compound gates use different static logic than the usual output inverter to compose the outputs of several dynamic gates using a multiple input static gate instead of the inverter. However, the use of such gates in technology mapping is too complex for the scope of this work.

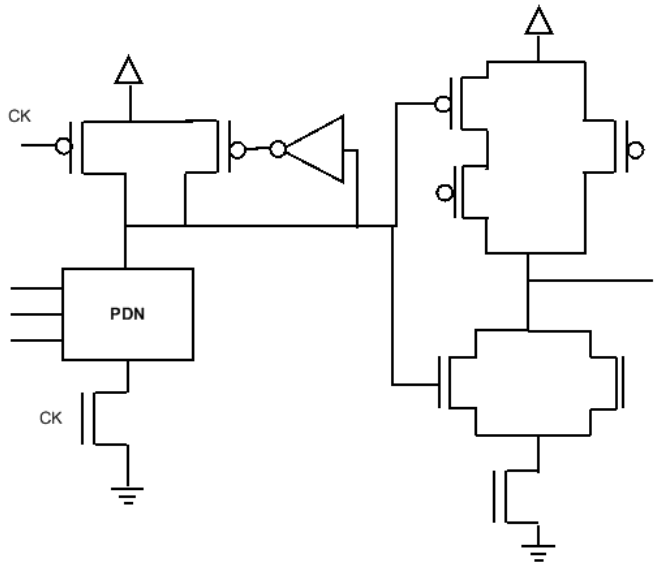

Figure 4.2 - DS Domino Logic gate.

5 PROJECT SPECIFICATION

The project specification involves the requirements to create a tool to perform static and domino logic synthesis. The application will be defined in terms of the input, the outcomes and the main features necessary to achieve the project purpose and goals.

5.1 Logic Synthesis Tool Structure

The tool to be implemented has the purpose to convert an RTL description into an optimized mixed domino/static gate level description, in other words, to perform logic synthesis in an environment where the support of Domino logic together with static complementary logic style is allowed.

The flow proposed for this tool is slightly different than the common logic synthesis flow and other domino logic synthesis flows (see Figure 5.1).

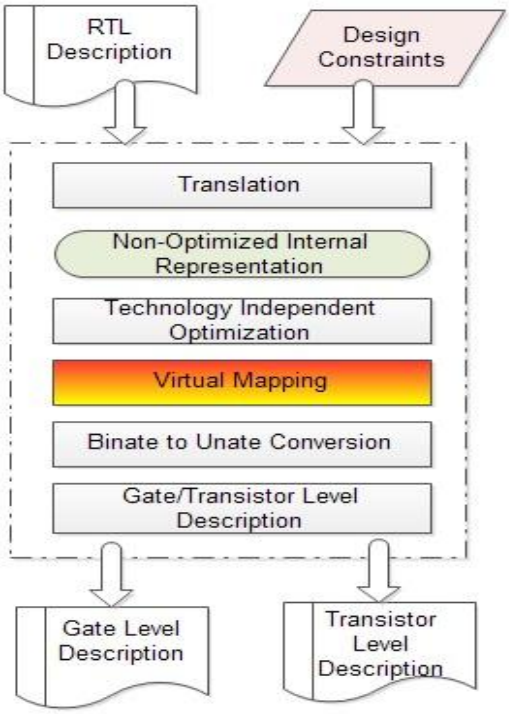

Figure 5.1 - Domino logic synthesis flow overview.

The translation of the RTL description to an internal representation requires the creation of a parser supporting an input format. The initial RTL circuit is described in this input format which is read and translated by the parser to an internal data structure.

This internal data structure has to be capable of representing such circuits efficiently for later steps manipulation.

The posterior steps are techniques presented on the Section 4 as follows:

- **Technology Independent Optimizations** are a small set of well-known algorithms to perform the logic optimizations, typically refactoring of equations, duplicated inverters removal, logic sharing with the goal of creating smaller equations which in the end means saving of transistors.
- **Virtual Mapping** maps the network to a gate level description. The virtual mapping algorithm does not require a pre-characterized library but has to be aware of the possibilities of creation of multi output domino, and simple domino gates when mapping for such type of logic style.
- **Binate to Unate Conversion** conditionally converts a technology independent optimized network into an inverter free network using the logic duplication and bubble (inverter) pushing algorithm.

The initial idea of this logic synthesis tool was to explore the possibility of implementing only part of the circuit with domino logic gates and the rest of it with static complementary gates because it may not be worth to perform a lot of logic duplication to have the implementation of some gates with domino logic. The conditional conversion isolates some parts of the network to stay binate and lately be mapped with static complementary gates.

The initial idea was far too complex to implement and we kept focus on mapping the circuits to either domino logic style or to static complementary style. We decide to invest more on the creation of a good mapping algorithm and optimization techniques which would enable the creation of a domino circuit with a lower area than static complementary circuit.

Additional interesting features are the capability to export back the internal structure to a RTL description again or to make a transistor level description. Such features would permit the integration to other tools or reuse of mapped circuits and the simulation of the circuit on a transistor level for a better timing analysis respectively.

5.2 Supported RTL Format

The RTL format supported by the application for both import and export is the Equation format (EQN). This format is one of the easiest formats because it has the ports description on the header of the file and the rest are simple logic expressions representation of the logic gates in the circuit.

Despite of being an easy to use format, this format is also supported by other logic synthesis tools as SIS (BRAYTON, 1992) and ABC from Berkeley Logic Synthesis and Verification Group.

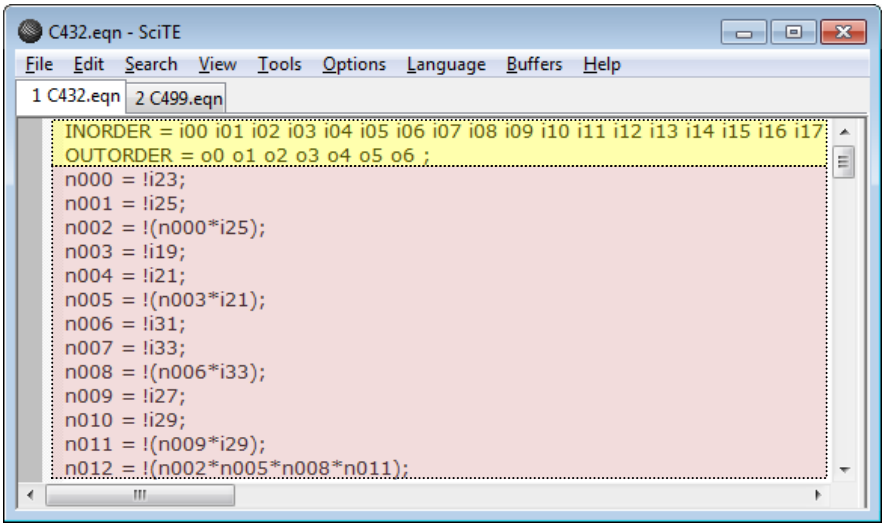

Figure 5.2 – An example of the Equation format (EQN).

5.3 Supported Spice Format

The application supports also the export to a transistor level description of the circuit for analysis and simulation purposes. The Spice format is supported by the tool because it is the most worldwide known and chosen by companies as well as there are several free Spice simulators available.

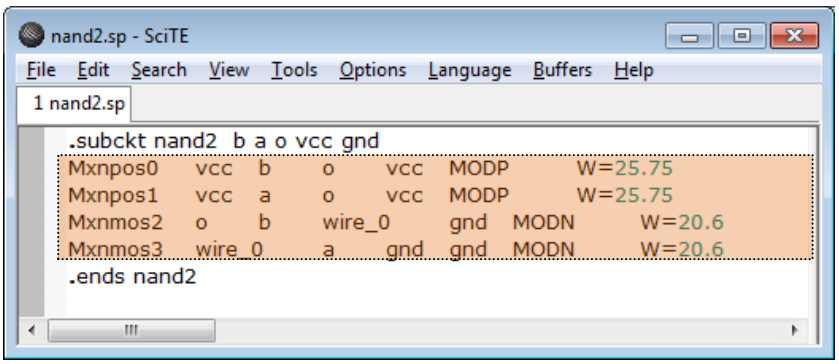

Figure 5.3 –Transistor level description of Nand2 gate in Spice format.

6 IMPLEMENTATION AND ALGORITHMS

This chapter has details about the implementation of the tool for Logic Synthesis and it describes important methods and features of the tool. The description starts from the loading of the circuit, which is a pre-processing step. Then, the methods belonging to each of the logic synthesis steps are explained. Finally, a description of the exporting options of the tool is provided.

The last section contains the UML implementation diagram (also known as class relationship diagram).

6.1 Circuit Loader (Equation Parser)

The purpose of this parser is to read (beside the list of I/O ports which is straightforward) the sequence of expressions defining the circuit and translate them into the N-ary Trees which are the internal data structure. In order to do that, the equation which initially is on infix format is transformed into a postfix format. Then, the equation on the postfix format is used to create the heaps (any two operands plus an operator build a heap). The later step is to merge adjacent nodes containing same logic. The outcome is an N-ary Tree representing the equation. The conversion of infix equation to postfix format and the technique to build the heaps are explained at (WEISS, 1995).

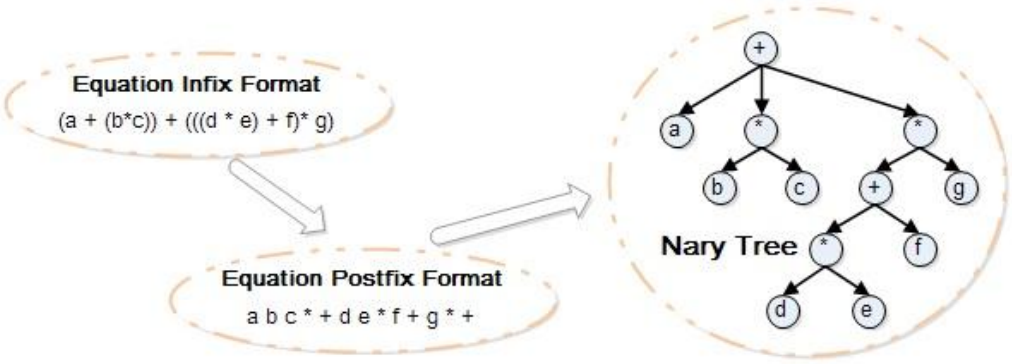

Figure 6.1 – Two steps to parse equation to N-ary Tree.

6.2 Technology Independent Transformations

These sections are dedicated to describe methods implemented to perform technology independent transformations trying to achieve better circuits. All the subsequent methods described are independent of each other. Despite of that, one may cooperate with other to be more efficient.

6.2.1 De Morgan

This method is applied on gate level and is more a transformation then an optimization itself but this transformation would make possible other optimizations. This method is also useful for making negated logic required for static complementary logic style with less transistors and non-negated logic which is mandatory for domino logic implementation.

6.2.2 Inverter Propagation and Redundancy Clean

The inverter propagation method is applied on gate level and its purpose is to push inverters from internal stages of the gate to the inputs. This method is required because it is too complicated to deal with multi-stage gates inside the synthesis tool (notice that inversions in the middle of the logic would create a multi-stage gate). Multi-stage gates contain inverters internally and this complicates functions like the calculation the number of series transistors of the gate. Additionally, it may also optimize the gate by reducing the number of inverters (see Figure 6.2).

The inverter propagation method may let the gate with a small inconsistency which is two levels containing the same logic and the merge of the nodes is required. This merge of nodes is herein called the logic redundancy clean.

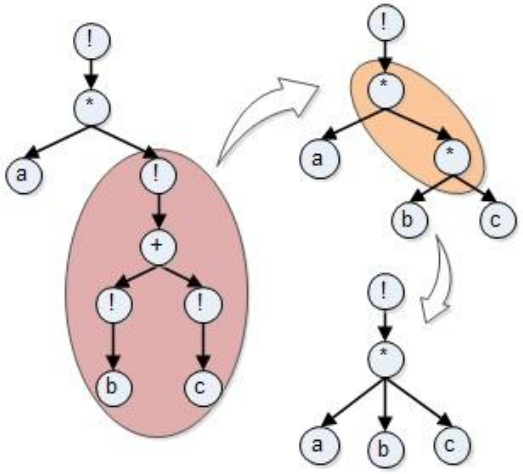

Figure 6.2 – Inverter propagation and logic redundancy clean.

6.2.3 Removal of input inverters

This method is made on circuit level and gate level. The purpose of this method is to remove input inverters from a gate and turn them into external inverter gates. This would allow, for example, the removal of duplicate inverters which before were not possible to remove due to the fact that one of them belongs internally to a gate.

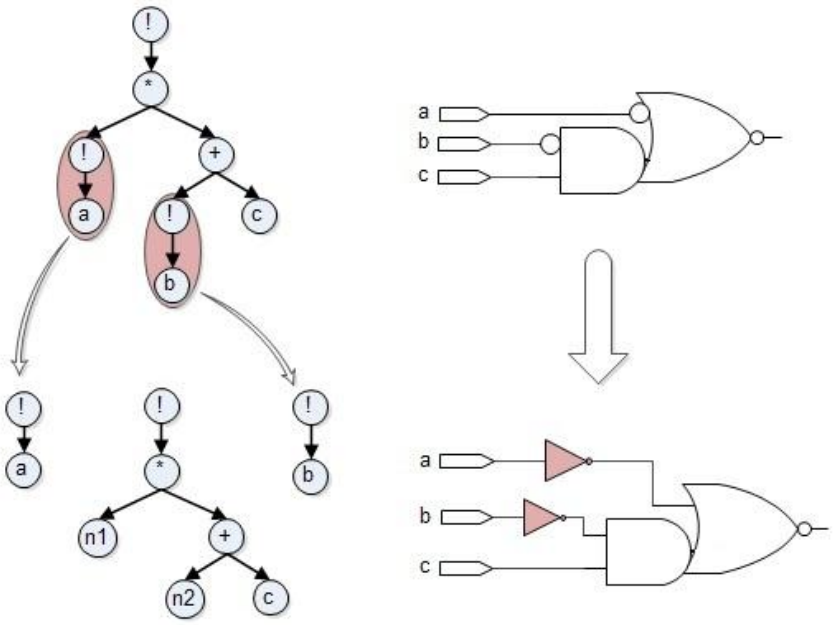

Figure 6.3 – Input inverter removal on gate level making new inverter gates.

6.2.4 Removal of duplicated inverters

This optimization is made on a circuit level and its purpose is to remove duplicated inverter gates from the circuit. Usually when double inverters are found, the best would be to eliminate both, but at some cases one of the inverters are used on other branches of the circuit making impossible to remove it. Another limitation of this method is that an output port cannot be directly assigned to an input port due to implementation issues. This implies that a double inverter ending on an output port and beginning on an input port will not be removed.

On the following circuit (see Figure 6.4), the red inverters are removed, the orange inverter is removed too and a special treatment is done to connect to a new net as the other inverter on the path can’t be removed.

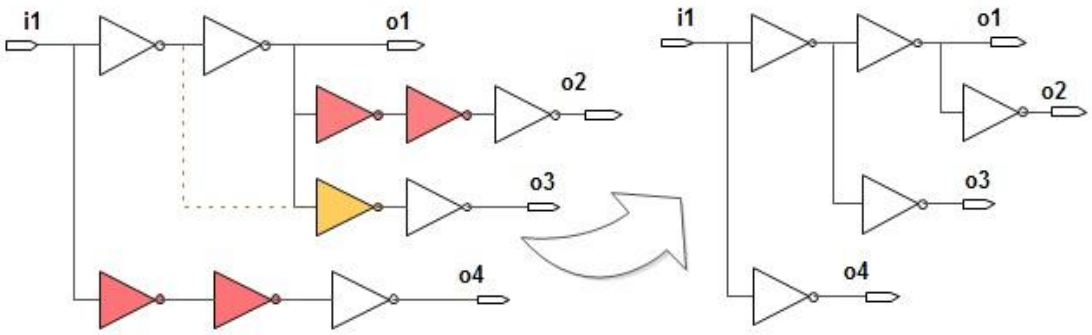

Figure 6.4 – Circuit containing double inverters.

6.2.5 Logic Sharing

Logic sharing is an important method for optimizing the circuit because it avoids to have duplicated logic when it is not needed. A static complementary circuit does not require any logic duplication but domino logic circuit sometimes does. On a domino logic circuit when two paths assign two different phases respectively to a logic cone, the

logic must be duplicated and only on this case this method won't perform any modification in the circuit. This method works on a circuit level, each gate has a unique equation. Before adding a new gate to the circuit, the method tries to match the new gate equation with all other equations of existing gates. If a match is found then the logic is shared. This method is very useful for reusing inverters.

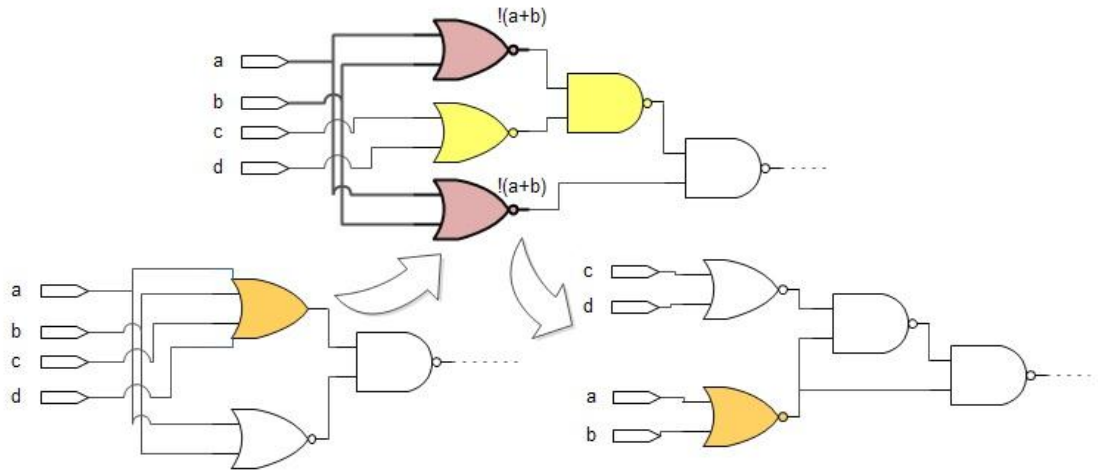

Figure 6.5 – Demonstration of how duplicated logic is created and then shared.

6.3 Virtual Mapping

The virtual mapping is based on merging (grouping) and splitting (virtual mapping cuts) gates. This procedure tries to make the circuit to have as few as possible gates which is (on most cases) the same as to have as large/complex as possible gates.

6.3.1 Grouping of gates

The grouping of cells is the first step of the virtual mapping. This method is applied on a circuit level and the purpose is to merge as much as possible gates to build large complex gates. The outcome of this step is a circuit containing the larger complex gates possible to have but so far not respecting the stack constraints.

This step is performed because the goal of the mapping is to achieve a circuit with larger complex gates which still respect the stack constraints. This method is a preparation step for making the best cuts afterwards.

6.3.2 Virtual mapping cuts

This method receives as input a circuit containing very large gates which do not respect the constraints for the mapping. Thus, the purpose of the method is to make the best cuts as possible, in other words, split the large gates into smaller ones that respect the constraints and optimize the circuit. This split can be done in many ways (core of gate splitting). This method has a core where the focus is to produce the fewer gates as possible as the main goal. The secondary goal is to make a split where the number of stages in sequence is the lowest (referred herein as logic depth). See Figure 6.6 containing a demonstration of this algorithm.

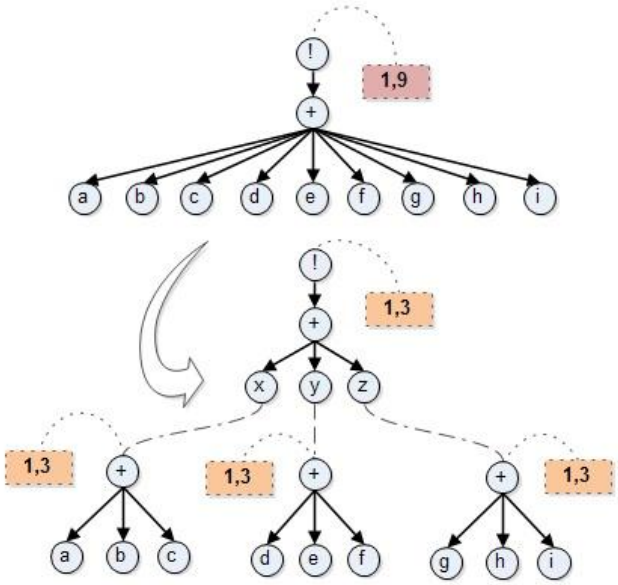

Figure 6.6 : Virtual mapping cuts on to respect constraints.

6.4 Binate to Unate Conversion

The binate to unate conversion is based on two methods. Section 6.4.1 describes the first method which is the logic duplication and section 6.4.2 describes the Bubble pushing algorithm which pushes the inverters to primary outputs.

6.4.1 Logic Duplication

The logic duplication is an essential method to make possible the circuit compatible with domino logic style. The initial task is to identify two paths trying to assign a different phase to the same logic cone which would not let the inverter propagation to be possible at the circuit level. In this case, any attempt to invert the phase on the logic cone would make mandatory to have an intermediate inverter in one of the subsequent paths (see Figure 6.7). Thus, this logic cone has to be duplicated. This is the first step for making the binate to unate conversion of the circuit.

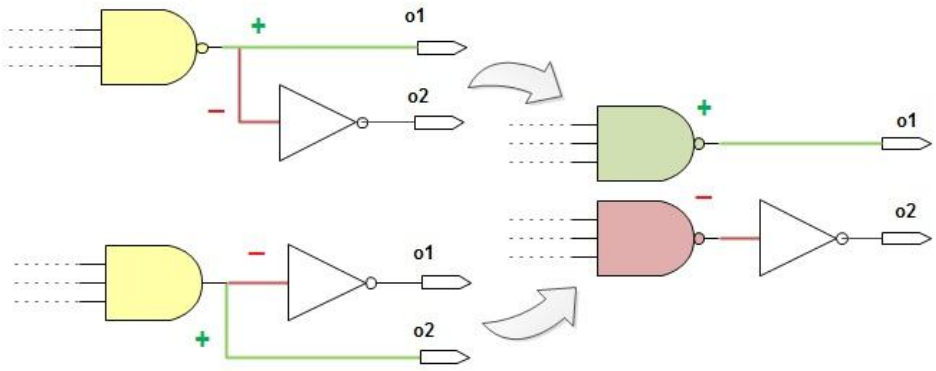

Figure 6.7 – Logic duplication in the cone where two different phases are assigned.

6.4.2 Bubble Pushing

This method is applied on a circuit level and gate level as well. The circuit received as input for this method has no restrictions to make the propagation of inverters to the inputs, because it was pre-processed in the previous step. In other words, it is assumed that the circuit has been pre-processed by prior step such that the circuit has no logic cone where two paths try to assign different phases. The goal of the bubble pushing method is to propagate intermediate inverters to inputs by merging them to the earlier stage cell, propagating the (possibly merged) inverter inside gate logics, extracting input inverters the merged gate can have and recursively repeating these steps for prior stage gates (see Figure 6.8). This is the final step for making the binate to unate conversion, as the outcome of this procedure is a circuit without any intermediate inverters.

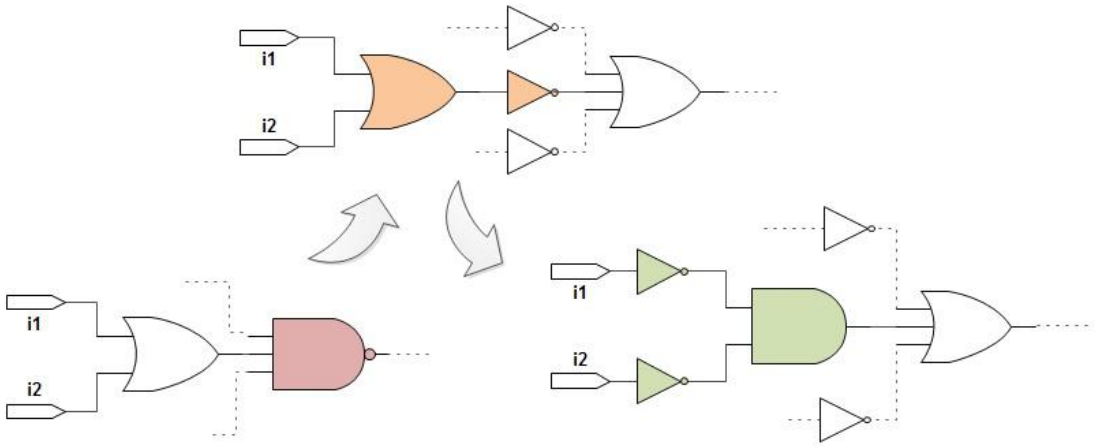

Figure 6.8 – Bubble Pushing method propagating inverters to primary inputs.

6.5 Circuit Exporter

The circuit exporter is very simple when compared to the circuit loader and it performs the opposite action. This component transforms the circuit into an Equation format file. The header of the file which has the input and output ports description are the first data collected from the nets. In the sequence, the equations are extracted from each gate which is represented by an N-ary Tree. The extraction of the equation is bottom-up as shown in Figure 6.9.

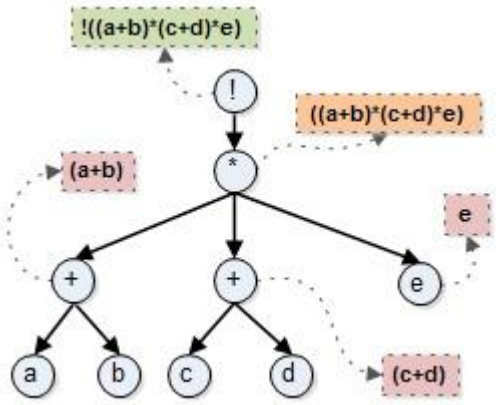

Figure 6.9 – Building an OAI221 equation.

6.6 Spice Generation

The goal of this component is to transform the circuit into a set of sub-circuits on Spice format. The sub-circuit information is extracted from each gate which is represented by an N-ary Tree and the concept is similar to building up equations, but here transistors networks are built (each literal becomes a transistor and the logic where it is inserted in determines how it connects with other transistors). The creation of the transistor network is bottom-up as demonstrated in Figure 6.10.

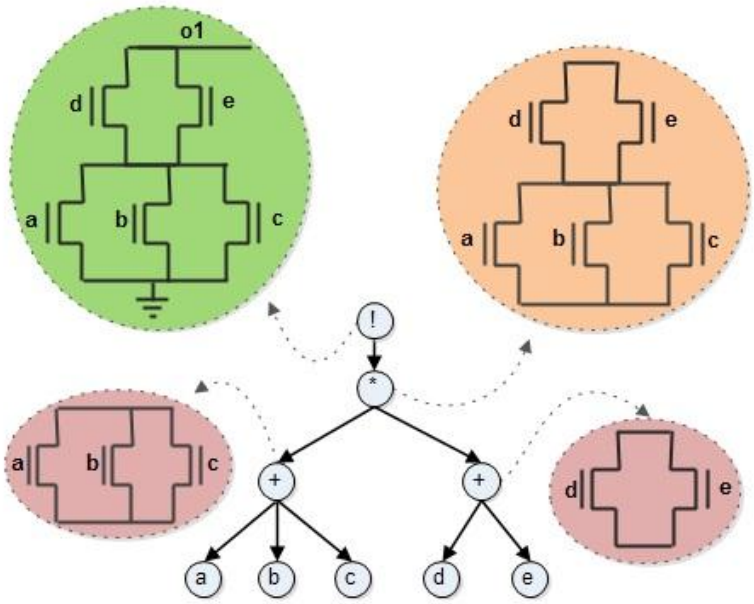

Figure 6.10 – Building an OAI32 transistor network.

6.6.1 Stack Sizing Technique

The sizing method used to determine transistor widths is based on the worst case stacks that each transistor belongs. The method generates base size multipliers for the transistor widths used for description in Spice format. This method will set a width multiplier to each transistor based on the stack size where it is inserted in. The method will take each sub-set of the transistor network, according to the worst case stack, and

size the transistors in the subset according the size of the stack. Afterwards, the other transistor sub-sets of the network are treated to apply an accumulated multiplier (see Figure 6.11).

Stack sizing techniques completely disregard the minimum sizing of the transistor as well as the PMOS to NMOS transistor ratio. These constants are defined by the technology used and not upon the size of the stack. More information about how determine these constants in (RABAEY, 2002).

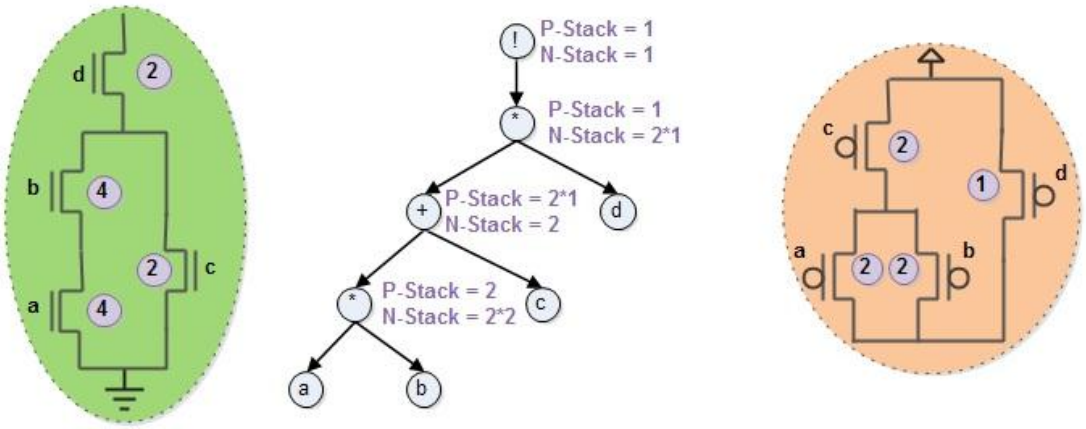

Figure 6.11 – Stack sizing method applied on the N-ary Tree.

6.7 UML Class Diagram

The Figure 6.12 presents a general overview of the logic synthesis tool implementation in UML. It was obtained by making a reverse engineering in the code using Star UML.

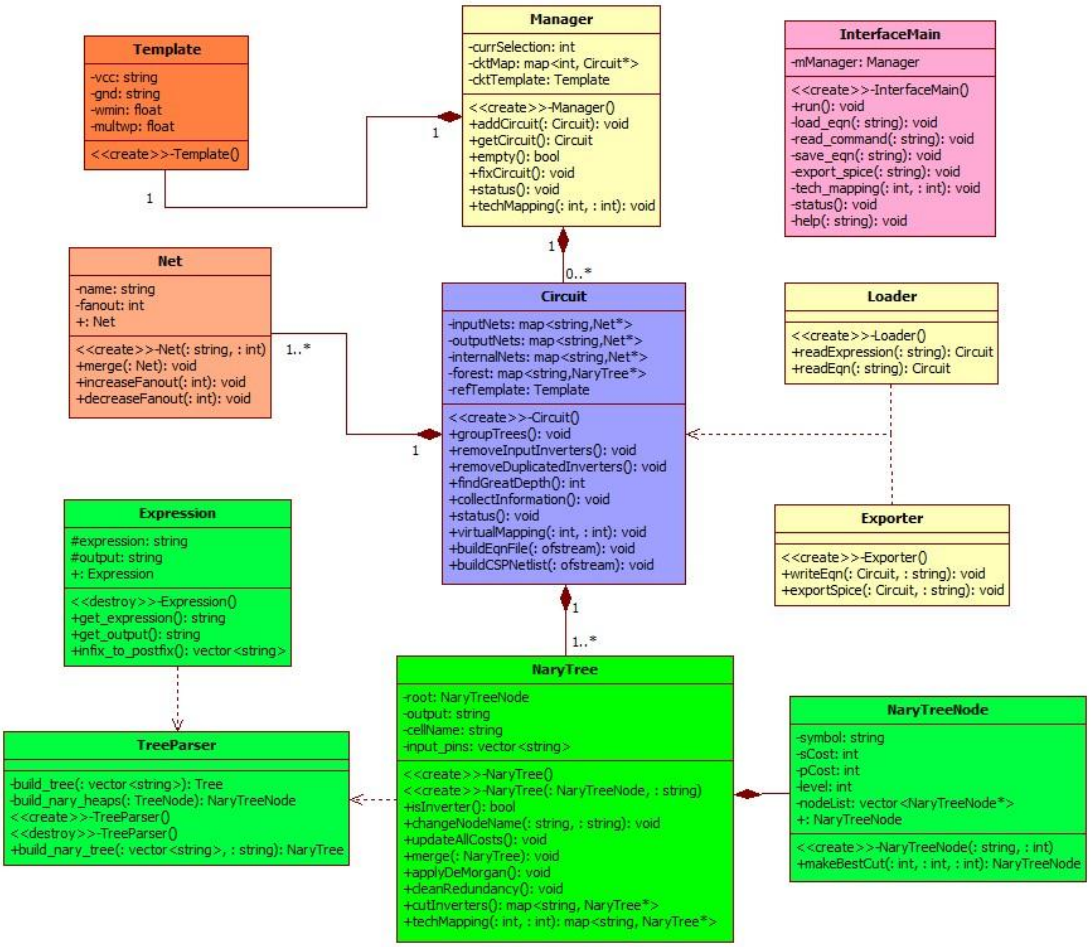

Figure 6.12 – Overview of the UML Implementation Model.

7 RESULTS

This section presents (Section 7.1) an analysis using the data collected from the synthesis tool to show the conditions where a domino circuit creation would give benefits in terms of area; and (Section 7.2) comparative results between the proposed tool and different tools/techniques for technology mapping. The results of the tool were checked for logical equivalence against the input circuits. Besides that, case studies with circuit simulators are being prepared to show how fast a domino circuit can be when compared to a static complementary circuit.

7.1 Mapping Analysis for Domino

On this section, an analysis of the virtual mapping is done trying to find the point where the constraints (maximum series stack) to be used will make possible to have an advantageous domino circuit for area.

We know that each gate in complementary static style has 2N transistors where N is the gate's number of inputs and in domino style the same gate has N+4 transistors. To make possible to have an advantageous domino circuit before the binate to unate conversion, we need an average fan-in which satisfy the following equations.

$$\bar{N} + 4 < 2 \times \bar{N}$$
$$\bar{N} > 4$$
$$\bar{N} : average\ fanin$$

After synthesizing and making virtual mappings for some benchmark circuits, it was verified that the minimum stack limitation of 3 transistors in series and maximum of 4 chains in parallel satisfy the conditions expressed by the equations. This way, by choosing 3 series and 4 parallel transistors, it is possible to have a benefit in area for domino circuits when compared to the same static complementary circuit before binate to unate conversion (see Table 7.1). Some circuits like C1355 would require more flexible constraints to have a gain on the conversion to a Domino circuit.

Table 7.1 : Average fan-in of mapped circuits

Average Fan-in					
	No map	**S=3, P=3**	**S=3, P=4**	**S=4, P=4**	**S=4, P=5**
C432	2.76	3.71	**4.26**	**4.53**	**5.00**
C880	2.18	3.70	**4.17**	**4.23**	**4.36**
C1355	2.09	2.88	2.89	2.95	3.01
C7552	2.25	3.97	**4.23**	**4.46**	**4.67**

The average fan-ins for the circuits shown in Table 7.1 disregard completely the inverter gates as those gates are removed after the binate to unate conversion.

7.2 Overall mapping comparisons

In this section, comparisons of the mapping with two other non-commercial tools (Elis and ABC) are presented. The metrics to be compared are: number of gates (G), number of inverters (I), and number of non-invert gates (NI), worst logic depth (WL), area estimation for static complementary circuit (ASC) and for domino circuit (ADC), average fan-out (AFO) and worst fan-out net (WFO). To be fair and make the comparisons using equivalent or similar commands to make a virtual mapping on other tools, we had to choose a constraint of a maximum of 4 transistors in series and 4 transistors in parallel.

The area estimation is based on the number of transistors each gate would typically have on the logic style which it is been built. This estimation considers only the non-inverter gates. The worst logic depth is considered as the number of gates on the largest path of the circuit.

7.2.1 Comparing to Elis

The equivalent command in Elis (CORREIA, 2004) to make a virtual mapping with such constraints is *vlib_tech_mapping 4 4.* The comparison done here is not completely fair because Elis was not able to either read or map the original ISCAS85 benchmark circuits. The workaround for that was to use a set of preprocessed circuits which came with the tool available on the tool installation folders. Those circuits contained fewer cells than the original ones. It's surprising that even not reading a pre-optimized circuit, the tool developed here presented better results for some cases.

The Table 7.2 presents the comparison of number of gates and area estimation on the mapped circuits.

Table 7.2 : Comparison of Gates and Area between Elis and DS.

Number of Gates and Area Estimation									
	ELIS				**DS**				
	G	**NI**	**I**	**ASC**	**ΔG**	**ΔNI**	**ΔI**	**ΔASC**	**ΔADC**
C432	162	70	92	564	0.90	**0.86**	0.92	**0.96**	**0.91**

C499	315	162	153	1152	1.03	1.00	1.07	1.00	1.06
C880	272	128	144	948	0.86	**0.84**	0.88	**0.96**	**0.93**
C1355	524	262	262	1568	1.02	1.02	1.02	1.00	1.18
C1908	478	244	234	1722	0.95	**0.96**	0.93	**0.99**	1.04
C2670	546	219	327	2190	1.12	**0.93**	1.24	**0.99**	**0.87**
C3540	697	394	303	3372	1.11	**0.95**	1.33	**0.97**	**0.93**
C5315	858	422	436	4998	1.13	1.00	1.26	1.00	**0.84**
C7552	1606	775	831	6812	1.06	**0.98**	1.13	1.00	**0.94**

This comparison shows some pros of the synthesis tool created here. The number of gates (excluding inverters) which represent the gates required really to represent the circuit logic had 5% less (gates) in the Domino Synthesis mapping. This is directly associated with how good the mapping can be. The area estimation also shows that the mapping was better. A circuit that has fewer gates does not necessary implies that it will have a lower number of transistors. If the gates selected are not the best, a circuit containing more gates may have fewer transistors. Another thing that we should clearly look is that depending upon the characteristics of the circuit will worth or not to convert into a domino circuit. The 9% average increase on the number of inverters shows that the synthesis can be even better that it actually is. An inverter minimization technique or, at least, a polarizing algorithm would reduce drastically the number of inverters. Unfortunately, none of those methods were developed in the scope of this work due to the lack of time.

The Table 7.3 presents the comparison of the worst logic depth, average fan-in and fan-out, and worst fan-out achieved after the mapping of the circuits.

Table 7.3 : Comparison of Worst Depth, Fan-in and Fan-out between Elis and DS.

Worst Logic Depth, Fan-in and Fan-out								
	ELIS				**DS**			
	WL	**AFI**	**AFO**	**WFO**	**ΔWL**	**ΔAFI**	**ΔAFO**	**ΔWFO**
C432	24	4.03	4.18	17	**0.75**	**1.13**	**1.20**	**1.24**
C499	19	3.56	1.53	8	**0.95**	1.00	**1.13**	**1.88**
C880	16	3.70	1.52	8	1.06	**1.14**	**1.23**	**1.63**
C1355	24	2.99	1.42	8	1.00	0.98	**1.15**	**1.88**
C1908	28	3.53	1.54	24	**0.96**	**1.03**	**1.28**	**1.54**
C2670	21	5.00	2.23	40	1.24	**1.07**	**1.14**	**1.65**
C3540	33	4.28	1.81	21	1.06	**1.03**	**1.19**	**1.19**
C5315	28	5.92	2.39	42	1.04	1.00	**1.25**	**1.55**
C7552	25	4.39	1.77	116	1.04	**1.02**	**1.87**	0.75

A higher average fan-in also is an indication of a better mapping as it means that you could merge more logic into fewer gates. A higher average fan-out and worst fan-out can be good and can be bad. It represents that the logic sharing algorithms were good. More gates could take advantage of an already implemented logic and reuse it. The problem is that there is a limitation for the maximum fan-out. This happens because a gate driving many other gates will be a problem; it will have to be a huge gate. Actually, on these cases the most common approach is to create inverter trees to reduce that fan-out.

7.2.2 Comparing to ABC

The ABC synthesis tool from Berkeley Logic Synthesis and Verification Group (version 51205) does not have an equivalent command to make a virtual mapping and due to that we had to read a circuit, optimize and map to a parameterized library *44-6.genlib* to compare the mappings. The circuits used were the same (ISCAS85-89 originals). The synthesis on ABC was done using the commands: *resyn, balance* and *map*.

The Table 7.4 presents the comparison of number of gates and area estimation on the mapped circuits.

Table 7.4 : Comparison of Gates and Area between ABC and DS.

Number of Gates and Area Estimation									
	ABC				**DS**				
	G	**NI**	**I**	**ASC**	**ΔG**	**ΔNI**	**ΔI**	**ΔASC**	**ΔADC**
5xp1*	59	52	7	396	1.17	**0.81**	3.86	1.66	1.25
C432	131	94	37	932	1.11	**0.64**	2.30	**0.58**	**0.55**
C499	483	442	41	3104	**0.67**	**0.37**	3.98	**0.37**	**0.39**
C880	291	234	57	1728	**0.80**	**0.46**	2.21	**0.52**	**0.51**
C1355	513	472	41	3120	1.04	**0.56**	6.51	**0.50**	**0.59**
C1908	447	405	42	2732	1.01	**0.58**	5.19	**0.62**	**0.66**
C3540	724	655	69	5666	1.07	**0.57**	5.87	**0.58**	**0.55**
C6288	2292	2235	57	18732	1.26	**0.64**	25.26	**0.41**	**0.51**

From this comparison we can see again that the mapping algorithm is really good by looking on the number of gates (ignoring inverters). Analyzing the number of inverters and the total number of gates we see how important an algorithm of inverter minimization is. Despite that, the area estimation was on average 33% less on Domino Synthesis. ABC seems to have very efficient algorithms for inverter minimization because the circuits mapped with this tool had 6 times less inverters.

The Table 7.5 presents the comparison of the worst logic depth, average fan-in and fan-out, and worst fan-out achieved after the mapping of the circuits.

Table 7.5 : Comparison of Worst Depth, Fan-in and Fan-out between Elis and DS.

Worst Logic Depth, Fan-in and Fan-out								
	ABC				**DS**			
	WL	**AFI**	**AFO**	**WFO**	**ΔWL**	**ΔAFI**	**ΔAFO**	**ΔWFO**
5xp1*	4	3.81	2.01	16	1.25	**2.05**	**2.23**	**3.19**
C432	11	4.96	2.85	13	1.64	0.91	0.62	**1.62**
C499	10	3.51	2.01	24	1.80	**1.01**	0.86	0.63
C880	8	3.69	1.77	21	2.13	**1.15**	**1.06**	0.62
C1355	10	3.31	1.91	16	2.40	0.89	0.85	0.94
C1908	13	3.37	1.91	19	2.08	**1.08**	**1.03**	**1.95**
C3540	17	4.33	2.74	28	2.06	**1.02**	0.79	0.89
C6288	38	4.19	2.31	59	3.26	0.63	0.65	0.27

The worst logic depth number on mapped circuit's shows how efficient is the command *balance* in ABC which made the circuits mapped on this tool to have on average half the depth of the circuits mapped with the Domino synthesis tool. Looking at the average fan-in and average fan-out numbers nothing can be concluded as for some circuits on tool had the higher values and for other circuits the other had. This means that the mapping and logic sharing algorithms on these tools are really powerful even being different.

7.3 Circuit Verification

All circuits mapped were compared to originals using the command *cec* on ABC which compare two circuits using a hybrid approach based on fraiging and SAT solving.

Some circuits present problem while reading the circuits due to the way it was exported and formatting issues. After fixing these problems, the circuits were read and fortunately all of them were considered equivalent to the originals.

7.4 Timing Analysis using Spice Simulations

This section was reserved to present results of timing when comparing a domino circuit to a static complementary circuit. This comparison would take the path with the worst depth (greater number of gate between a primary input and a primary output) and build a transistor level description of it on Spice format.

This transistor level description would make possible a simulation of those paths containing the gates implemented in the different logic styles. Thus, this comparison would only be possible on both circuits created by Domino synthesis because ABC does not export a transistor level description of the circuit or the worst path.

The comparisons results are still being obtained and will be on the final version of the document.

8 CONCLUSIONS

In this work, I have investigated the domino logic style. This style is very interesting because it has many characteristics which can help on the creation of high performance VLSI design. However, domino logic particularities require a different approach to make logic synthesis due to the non-inverting nature of these gates. A different logic style usage also brings new issues to the library design.

Due to the lack of a standard approach to generate domino circuits, I decided to make a logic synthesis tool that would deal with the required methods to create either a static complementary or a domino compatible circuit or mixed. Later, we had to exclude the creation of a mixed circuit due to the project complexity. A tool able to make synthesis for both logic styles has to be a little bit different than any other tool developed or proposed so far. This tool would require a step to make binate to unate conversion; this step has to be performed after the mapping, so the circuits to be compared had the same mapping. We chose to implement virtual mapping due to the fact that there is not any available free library designed for domino gates. This implies that we would also need a transistor network generator for both logic styles to later perform simulations.

Looking at the results presented, we can see that our technology independent optimization methods and the virtual mapping algorithms created circuits with an average of 1% less estimated area when compared to ELIS and 33% less when compared to ABC. The number of gates (excluding inverters) on the circuits created was (in most of the cases) lower and the average fan-in was higher. This represents how good the mapping of the proposed tool is. The higher average fan-out which happened most of times demonstrates how efficient the logic sharing algorithm is.

Several things were missing on this project and I will just enumerate a few. First, the initial idea of creating mixed circuits would be a great achievement, in my opinion, as it would bring together the better of the two worlds (domino and static). However, it was aborted due to the complexity to make it. The binate to unate conversion step uses the bubble pushing algorithm which is worse than the output phase assignment algorithm, which can propagate inverters to outputs giving a better circuit. And finally, looking at the results, we see that the circuits created had a huge number of inverters and a greater logic depth when compared to ABC. This means that we could have developed algorithms for inverter minimization and for balancing the circuit to reduce the depth.

REFERENCES

R.H. KRAMBECK, C.M. LEE, H.S. LAW. **High-Speed Compact Circuits with CMOS**. In IEEE Journal of Solid State Circuits, SC-17(3):614-619. June 1982.

J. Pretorius, A. Shubat, C. Salama, **Charge redistribution and noise margins in domino CMOS logic.** IEEE Transactions on Circuits and Systems, CAS-33, pp310-314. 1993.

R. PURI, A. BJORKSTEN, T. ROSSER. **Logic optimization by output phase assignment in dynamic logic synthesis.** In Int. Conference on Computer Aided Design, pages 2–8. 1996.

M. R. PRASAD, D. KIRKPATRICK, R. K. BRAYTON. **Domino Logic Synthesis and Technology Mapping.** In Int. Workshop on Logic Synthesis. 1997.

M. ZHAO, S. S. SAPATNEKAR. **Technology Mapping for Domino Logic**. In IEEE/ACM Proc. Of Design Automation Conference, pp 248-251. 1998.

T. THORP, G. YEE, C. SECHEN. **Domino Logic Synthesis Using Complex Static Gates**. In Proc. of Int'l Conference on Computer Aided Design, pp 242-247, 1998.

H. SONG, R.I. BAHAR. **Power, Delay, and Area Constrained Synthesis for Mixed Domino/Static Logic Optimization**.

Z. WANG, G. A. JULLIEN, W. C. MILLER, J. WANG, S. S. BIZZAN. **Fast adders using enhanced multiple-output domino logic**. IEEE J. Solid-State Circuits, vol. 32, pp. 206-213. Feb. 1997.

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

M. BERKELAAR, J. JESS. **Technology mapping for standard-cell generators**. In Int. Conf. Computer-Aided Design, p.470-473. Santa Clara, CA. November. 1988.

V. CORREIA, A. REIS. **Advanced technology mapping for standard-cell generators.** In: 17th SBCCI'2004, Proceedings… p. 254–259. Pernambuco, Brazil. 2004.

A. MISHCHENKO, S. CHATTERJEE, R. BRAYTON, X. WANG, T. KAM. **Technology mapping with Boolean matching, supergates and choices.** ERL Technical Report, EECS Dept., UC Berkeley, March 2005.

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

S. CHATTERJEE, A. MISHCHENKO, R. BRAYTON. **Factor cuts**. Proceedings…, p. 143-150, ICCAD. 2006.

T. WILLIAMS. **Dynamic Logic: Clocked and Asynchronous**. Tutorial notes at the Int. Solid State Circuits Conf. 1996.

J.M. RABAEY. **Digital Integrated Circuits**, 2nd Ed. 2002.

R. ROSSAIN. **High Performance ASIC Design: Using Synthesizable Domino Logic in an ASIC Flow**. 1st Ed. Cambridge University Press. 2008.

J. J. ROLAND, S. DEVADAS. **Logic Synthesis in a Nutshell.** Morgan Kaufmann. 2009.

G. D. HACHTEL, F. SOMENZI. **Logic Synthesis and Verification Algorithms.** 1996.

F. R. WAGNER, A. I. REIS, R. P. RIBAS. **Fundamentos de Circuitos Digitais.** Sagra Luzzatto. 2006.

S. DEVADAS, A. GHOSH, K. KEUTZER. **Logic synthesis**. McGraw-Hill, Inc. 1994.

S. HASSOUN, T. SASAO. **Logic Synthesis and Verification (The Springer International Series in Engineering and Computer Science).** Springer. 2001.

I. SUTHERLAND, B. SPROULL, D. HARRIS. **Logical Effort: Designing Fast CMOS Circuits**. Morgan Kaufmann. 1999.

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

M.A. WEISS. **Data Structures and Algorithm Analysis in C++**, 2nd Ed., Chapters 3.3.3, 4.2.2. 1995.

E. M. SENTOVICH, K. J. SINGH, L. LAVAGNO, C. MOON, R. MURGAI, A. SALDANHA, H. SAVOJ, P. R. STEPHAN, R. K. BRAYTON, A. L. SANGIOVANNI-VINCENTELLI. **SIS: a system for sequential circuit synthesis.** Memorandum no. M92/41, ERL, University of California, Berkeley, May 1992.

BERKELEY LOGIC SYNTHESIS AND VERIFICATION GROUP. **ABC: A System for Sequential Synthesis and Verification**, Release 61225. Link accessed in June 23, 2011. http://www.eecs.berkeley.edu/~alanmi/abc/

STANFORD ANALOG COMMUNICATIONS DESIGN LABORATORY. **Spice Quick Reference Sheet.** 2001. Link accessed in June 23, 2011. http://www.stanford.edu/class/ee133/handouts/general/spice_ref.pdf

A. VANCE, B. STONE. **Apple Buys Intrinsity, a Maker of Fast Chips**. New York Times, April 27, 2010. Link accessed in June 23, 2011. http://www.nytimes.com/2010/04/28/technology/28apple.html