POLITECNICO DI TORINO

Corso di Laurea Magistrale in Computer Engineering

Tesi di Laurea Magistrale

Development and evaluation of Synthesis and Optimization strategies for digital integrated circuits

Realizzata in collaborazione di:
STMICROELECTRONICS srl

Relatori

Dr. Prof. Enrico Macii
Firma ....

Dr. Prof. Andrea Calimera
Firma ....

Supervisore Aziendale

Dr. Michelangelo Grosso
Firma ....

Candidato

Dario Licastro
Firma ....

November 2022

Abstract

Synthesis and optimization strategies for digital integrated circuits are topics of great importance. Interest in this area has increased in recent years because of its utility in the various fields characterized by the current technological revolution. Computer-aided design (CAD) techniques have provided the methodology for efficient and successful design of large-scale high-performance circuits for a wide range of applications, from automotive to biomedical signal processing, and so on. The exponential increase in design complexity has necessitated the development of automated techniques to achieve adequate results in a shorter time. For this reason, smarter strategies need to be developed to reduce human interaction in the design process. Human interaction is time-consuming and error-prone. Strategies must overcome several challenges, from area timing, energy consumption, and testability. Testability is important to reduce testing time, which is the most expensive part of the design process. This thesis focuses on the development and evaluation of several synthesis and optimization strategies for digital integrated circuits, comparing the effects of different choices in the flow on the main design metrics, i.e., power, area and timing. The goal is to develop a flow capable of minimizing metrics with the least complexity and time. In addition, the developed strategies were verified and evaluated, showing how the key parameters affect the outcome and how the flow can be adjusted for better results. The strategies were applied in a mixed-signal ASIC design to evaluate the result. The project started with a basic synthesis flow that is stable and scalable, and starting from this flow, an exploration of possible further strategies is presented. The main areas in which these flow variants are developed are clock gating, the introduction of different cell libraries and the different sequences of optimizations in the flow. Clock gating was explored with the introduction of cloning techniques, or the variation of relevant parameters such as maximum fanout, minimum bandwidth and maximum number of stages. Various types of cell libraries, low leakage and low scale were used to investigate designs with less prohibitive power supply models or designs with fewer timing issues. Strategies have been developed for managing the

synthesis flow, varying parameters during synthesis and thus giving the EDA tool different starting points during synthesis. The main metrics taken into account are area, power consumption and timing. Each strategy developed is consequential to the information gained from the previous strategies. The aim is to create stable alternative strategies by analysing the results obtained and the various metrics.

Contents

1 Introduction **10**
1.1 Motivation . . . 10
1.2 Purpose and Objectives . . . 11
1.3 Thesis Structure . . . 11

2 Background **13**
2.1 Microelectronic Digital Design Flow . . . 13
2.2 Metrics . . . 19
2.2.1 Pareto curve . . . 19
2.3 Abstract models . . . 20
2.4 Architectural Level Synthesis . . . 22
2.4.1 Scheduling and Binding . . . 22
2.4.2 Sharing . . . 24
2.5 Logic Level Synthesis . . . 25
2.6 Verification Techniques . . . 26
2.6.1 Simulation . . . 26
2.6.2 Formal verification . . . 28
2.7 Power Modeling . . . 28
2.7.1 Switching Activity . . . 31
2.7.2 Clock Gating . . . 31
2.8 Timing Analysis . . . 34
2.8.1 Critical Paths . . . 35
2.9 Practical example of a RTL synthesis flow . . . 36
2.9.1 Loading Design and Technology Data . . . 37
2.9.2 Constraints application . . . 39
2.9.3 Synthesize the Design . . . 41
2.9.4 Write out results and Design Data . . . 42
2.9.5 Two-pass synthesis flow . . . 43

3 Approach **44**
3.1 Synthesis Purpose . . . 44
3.1.1 Custom synthesis parameters . . . 45
3.2 Hierarchical Design . . . 50

3.3 Simulation of Digital Systems . . . 53
3.3.1 VCD/FSDB file . . . 53
3.4 Validation and Evaluation of the Synthesis and Optimization strategies 54
3.4.1 Evaluation metrics . . . 54
3.4.2 Formal verification . . . 55

4 Implementation 57
4.1 The used software tools . . . 57
4.1.1 Cadence Xcelium . . . 57
4.1.2 Synopsis PrimePower . . . 58
4.1.3 Synopsis Design Compiler NXT . . . 58
4.1.4 Synopsis Formality . . . 59
4.2 The used libraries . . . 60
4.2.1 Standard Threshold Voltage cell library . . . 60
4.2.2 Low Threshold Voltage cell library . . . 60
4.3 Proposed Synthesis and Optimization strategies . . . 61

5 Results 72
5.1 Clock Gating . . . 73
5.1.1 Clock Gating disable . . . 73
5.1.2 Clock Gating without Cloning . . . 75
5.1.3 Clock Gating with high Fan-out . . . 76
5.1.4 Clock Gating with low Fan-out . . . 78
5.1.5 Clock Gating with more Stages . . . 79
5.1.6 Clock Gating with different Bitwidth . . . 82
5.1.7 Clock Gating without Sharing . . . 84
5.2 Libraries . . . 86
5.2.1 Low leakage cells . . . 86
5.2.2 Low leakage cells with low fan-out clock gating . . . 88
5.3 Custom Synthesis Flow . . . 90
5.3.1 Changing Clock Gating Strategies after first optimization . . . 91
5.3.2 Changing Libraries after first optimization . . . 92

6 Conclusions 95
6.1 Clock gating . . . 95
6.2 Cell libraries . . . 96
6.3 Custom Synthesis Flow . . . 97
6.4 Future works . . . 97

List of Figures

2.1 MOSFET from STMicroelectronics. 14
2.2 Microelectronic digital design Flow. 15
2.3 In black, Pareto point for a Delay-Area trade-off. In red, not optimal point. 20
2.4 Gajski–Kuhn chart showing the many perspectives on VLSI hardware design. 21
2.5 ASAP scheduling[16]. 23
2.6 ALAP scheduling. 24
2.7 A Testbench entity 27
2.8 Switching Activity, representation of 4 switch on 8 clocks. 29
2.9 I flow on transition. 30
2.10 Peak current during output transition. 30
2.11 Clock Gating example with ENABLE signal. 31
2.12 Clock Gating example with Control Unit that produce the ENABLE signal [1]. The Control Unit is composed by a counter, an input ”inc” and a logic memory containing the desired value. 32
2.13 AND gate-based clock gating[1]. 32
2.14 Glitch AND gate-based clock gating[1]. 32
2.15 Integrated Gated Clock (ICG) cells[1]. 33
2.16 Multi-level propagation of Enable clock gating signal[1]. 33
2.17 Figure a : Clock Gating without cloning, Figure b : Clock Gating with cloning and fanout distribution[3] 34
2.18 Critical path composed by nodes A,D,G,O is a false path. 35
2.19 Synthesis from a concept to an object. 36
2.20 RTL synthesis flow. 37
2.21 Example of timing path. 39
2.22 Timing constraints on input and output port. 40
2.23 Default synthesis step. 41
2.24 Two-pass synthesis flow. 43

3.1 Example of an ASIC RTL schematic with datapath and control unit[6]. 45
3.2 Phases of basic design flow and application location of new strategies. 47
3.3 Multistage Clock Gating. 49
3.4 Example of Behavioral Hierarchy of a CPU. 51

3.5 Example of Structural Hierarchy of a CPU. 52
3.6 Example of Physical Hierarchy of an Integrated Circuit. 52
3.7 xcelium ouput waveform for a 3 bits RCA. 53
3.8 Logic equivalence check (LEC) process [15] 56

4.1 A sketch about the top digital module of the design. 62
4.2 The clock gating structure is applied to the basic flow. 63

List of Tables

1.1 Integration density of Integrated Circuits (ICs). 10

5.1 Setup violation for no clock-gating flow. 73
5.2 Hold violation for no clock-gating flow. 73
5.3 Clock gating report for no clock-gating flow. 74
5.4 Power analysis for no clock-gating flow. 74
5.5 Area for no clock-gating flow. 74
5.6 Setup violation for no clock-gating cloning flow. 75
5.7 Hold violation for no clock-gating cloning flow. 75
5.8 Clock gating report for no clock-gating cloning flow. 75
5.9 Power analysis for no clock-gating cloning flow. 76
5.10 Area for no clock-gating cloning flow. 76
5.11 Setup violation for high Fan-out strategies. 76
5.12 Hold violation for high Fan-out strategies. 77
5.13 Clock gating report for high Fan-out strategies. 77
5.14 Power analysis for high Fan-out strategies. 77
5.15 Area for high Fan-out strategies. 77
5.16 Setup violation for low Fan-out strategies. 78
5.17 Hold violation for low Fan-out strategies. 78
5.18 Clock gating report for low Fan-out strategies. 79
5.19 Power analysis for low Fan-out strategies. 79
5.20 Area for low Fan-out strategies. 79
5.21 Setup violation for more stages strategies. 80
5.22 Hold violation for more stages strategies. 80
5.23 Clock gating report for more stages strategies. 81
5.24 Power analysis for more stages strategies. 81
5.25 Area for more stages strategies. 82
5.26 Setup violation for different Bitwidth exploration. 82
5.27 Hold violation for different Bitwidth exploration. 83
5.28 Clock gating report for different Bitwidth exploration. 83
5.29 Power analysis for different Bitwidth exploration. 84
5.30 Area for different Bitwidth exploration. 84
5.31 Setup violation for clock gating without sharing strategies. 85
5.32 Hold violation for clock gating without sharing strategies. 85

5.33 Clock gating report for clock gating without sharing strategies. 85
5.34 Area for clock gating without sharing strategies. 85
5.35 Setup violation for synthesis with LVT cells. 87
5.36 Hold violation for synthesis with LVT cells. 87
5.37 Clock gating report for synthesis with LVT cells. 87
5.38 Power analysis for synthesis with LVT cells. 87
5.39 Number of cell for each cell library for synthesis with LVT cells. . . . 88
5.40 Area for synthesis with LVT cells. 88
5.41 Setup violation for synthesis with LVT cells and low fonout clock gating structure. 88
5.42 Hold violation for synthesis with LVT cells and low fonout clock gating structure. 89
5.43 Clock gating report for synthesis with LVT cells and low fonout clock gating structure. 89
5.44 Power analysis for synthesis with LVT cells and low fonout clock gating structure. 89
5.45 Number of cell for each cell library for synthesis with LVT cells. . . . 90
5.46 Area for synthesis with LVT cells and low fonout clock gating structure. 90
5.47 Setup violation for synthesis with different clock gating structure after first compile. 91
5.48 Hold violation for synthesis with different clock gating structure after first compile. 91
5.49 Clock gating report for synthesis with different clock gating structure after first compile. 91
5.50 Power analysis for synthesis with different clock gating structure after first compile. 92
5.51 Area for synthesis with different clock gating structure after first compile. 92
5.52 Setup violation for synthesis with introduction of LVD cells after first compile. 92
5.53 Hold violation for synthesis with introduction of LVD cells after first compile. 93
5.54 Clock gating report for synthesis with introduction of LVD cells after first compile. 93
5.55 Power analysis for synthesis with introduction of LVD cells after first compile. 93
5.56 Area for synthesis with introduction of LVD cells after first compile. . 94

1 Introduction

In recent years, digital microelectronics has been the driving force behind the development of embedded systems. The ever-increasing level of integration of electronic devices has led to an increase in the complexity of systems[4]. The first generation of chips was called Small-Scale Integration (SSI) (Table 1.1) and enabled the development of logic gates on a single chip through the use of diodes, transistors, resistors, and capacitors. This was followed a few years later by the development of Very Large-Scale Integration (VLSI) (Table 1.1) chips that could contain millions of logic gates in a single chip. The increasing complexity of systems has led to an exponential increase in development time. This resulted in the importance of electronic design automation (EDA) tools, which made it possible to automate and simplify development processes. Synthesis and optimization are two processes in the development of embedded systems. Synthesis deals with compiling an abstract model into a logical model that contains information about the circuit, such as area, time, and power consumption. Optimization is the process of refining the logical model into a more efficient one.

Density of integration	**Gates per IC**
Small-Scale Integration (SSI)	< 10
Medium-Scale Integration (MSI)	10 to 100
Large-Scale Integration (LSI)	100 to 10,000
Very Large-Scale Integration (VLSI)	10,000 to 100,000
Ultra Large-Scale Integration (ULSI)	100,000 to 1,000,000
Giga-Scale Integration (GSI)	$> 1,000,000$

Table 1.1: Integration density of Integrated Circuits (ICs).

1.1 Motivation

The introduction of new technologies in the semiconductor market and the increasing complexity of chips lead to the need to research and analyse the best development strategies. The process of synthesis and optimization of integrated systems is a complex and highly customisable process.

Different synthesis strategies and the use of innovative techniques lead to variations in the quality of the final result. The strategy used must be able to minimise metrics such as area and power consumption while respecting the limits of metrics such as timing. The theoretical approach suggests several synthesis strategies that can be applied in EDA. These techniques need to be integrated into strategies that are most suitable for specific technologies and different design structures.

An analysis of the different strategies will allow the exploration and development of specific strategies for the synthesis of specific integrated circuits.

1.2 Purpose and Objectives

The first objective of this work is to develop a synthesis and optimization flow with good quality results, analyzing how the different strategies affect the metrics. Therefore, the first part of the project was spent analyzing the EDA tools used in RTL synthesis and studying the different possible customizations. Sequentially, a given basic synthesis flow is evaluated and used as a starting point. The evaluation and verification phases introduce some complexity that needs to be addressed. A key part is the development of a correct evaluation and verification methodology. The second objective of this work is to classify how different strategies affect the synthesis of the digital portion of a mixed-signal integrated circuit. To this end, a number of synthesis flows have been developed and evaluated. The two objectives are interrelated and should be developed in parallel. The classification of strategies helps the designer to achieve good quality of results.

Problem formulation

- What are the available synthesis and optimisation strategies?
- How do the metrics vary as the synthesis strategies change?
- What is the correct methodology for evaluating metrics?
- What are the methods for verifying the resulting netlist?
- Which combination of strategies produces the overall best results considering the design metrics?

1.3 Thesis Structure

The thesis will be structured in 6 chapters as follows:

- Chapter 2 presents an introduction to the concepts of Microelectronic design flow, the general use case of this study, and an overview of the state-of-the-art of synthesis and optimization strategies. It introduces basic knowledge about how to simulate an RTL netlist.

- Chapter 3 will focus on the technical description of RTL to logic synthesis flow, with an overview of the methodologies to handle with hierarchical design. Chapter 3 describes the approach proposed to evaluate and verify the netlist.
- Chapter 4 will describe the used software tools for the different parts of the implementation and the used libraries for the synthesis. Chapter 4 describe the proposed synthesis and optimization strategies implemented functionalities.
- Chapter 5 will describe the mixed-signal design under development and report the different results of the various Synthesis and Optimization strategies on the selected case study.
- Chapter 6 present an evaluation of the different strategies considering the main metrics and the execution time and draws some conclusions.

2 Background

This chapter introduces the necessary theoretical information to fully understand the main solution proposed in this work. The description of the microelectronic design flow is intended to situate this work within the overall development process. The different synhtesis strategies used in this thesis were evaluated using different metrics. This is to compare the different results and focus on adaptation. The measurement and classification methods are described in the metrics section. The synthesis thread starts with the definition of the abstract models and the synthesis connection process to reach each model. The state of art of power modeling and timing analysis is briefly discussed with the main concept of verification. Simulation and formal verification are both important to correctly analyzing the design structure. The reader may fully comprehend the objectives and decisions made to achieve the thesis goals using these principles.

2.1 Microelectronic Digital Design Flow

Over the last decades, the electronics industry has grown at a frenetic pace, owing to fast advancements in integration technologies and large-scale system design. The number of transistors per chip has increased enormously and, with it, the integration scale. The first MOSFET was invented in 1959 (an example by STMicroelectronics is shown in figure 2.1). Very large-scale integration (VLSI) was introduced in 1978 and is the paradigm changer in the Microelectronic ecosystem.

Microelectronic design flow (figure 2.2) is now a well-established procedure in silicon chip design. To ensure a successful design, is important to follow proven methodologies.

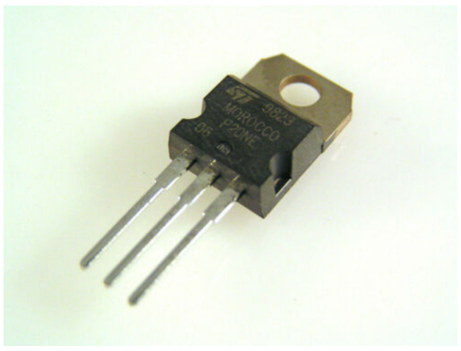

Figure 2.1: MOSFET from STMicroelectronics.

Microelectronic digital design incorporates two primary stages:

- Front-End Design includes the specification of digital design using a hardware description language, as Verilog, System Verilog, and VHDL. Additionally, this step includes design verification through simulation and other types of verification. It also includes architectural level synthesis. (2.4).
- Back-End Design concerns floorplanning, place and route, and verification of the physical design.

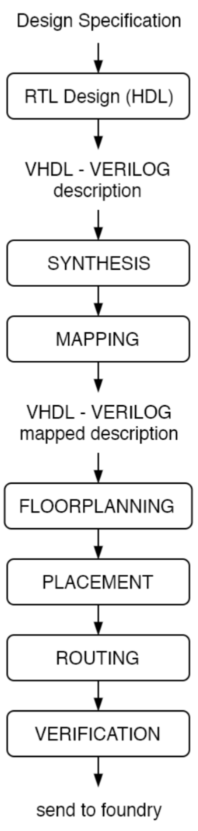

Figure 2.2: Microelectronic digital design Flow.

Design specification

Requirement definition is the starting point of each design flow. This step is not formal. Consequently, a step to formalize the design is needed. The formal language used in microelectronics is called Hardware Description Language (HDL). HDLs, such as VHDL and Verilog, are powerful computer languages for describing electronic circuit topology and function.

RTL design

In the logic design stage (HDL design in figure 2.2), a model is drawn using HDL. The function of the model is to be verified using a test bench to ensure the correctness of the design with respect to requirements.

The following lines provide an example of the RTL description of a generic multiplexer in VHDL:

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;

entity MUX21_GENERIC is
  Generic (NBIT: integer:= numBit;
           DELAY_MUX: Time:= tp_mux);
  Port (        A:      In      std_logic_vector(NBIT-1 downto 0);
                B:      In      std_logic_vector(NBIT-1 downto 0);
                SEL:    In      std_logic;
                Y:      Out     std_logic_vector(NBIT-1 downto 0));
end MUX21_GENERIC;

architecture BEHAVIORAL of MUX21_GENERIC is --normal multiplexer
  begin

    Y <= A when SEL='1'  else B;

end BEHAVIORAL;
```

Synthesis

In the synthesis stage, the model is converted into a gate level netlist. A synthesis tool performs this process, which takes as input:

- Hierarchy;
- Input and output ports of entities;
- Architectural structures: Generic maps and Port maps, Boolean functions, Process and Sensitivity lists.

It generates a generic gate-level architecture using Boolean laws and complex algorithms. At this stage, dependency on the real library and constraints are not taken into account.

Design for testability

Design for Testability is a strategy for incorporating testability characteristics into the design of a hardware product. The new capabilities simplify the development and application of manufacturing tests to the designed hardware.
Tests are performed at many stages of the hardware production process and may also be utilised for hardware maintenance in the customer's environment. Test programmes are often used to drive the tests, which are then executed on automated test equipment (ATE) or, in the case of system maintenance, within the constructed

system itself. Tests may be able to log diagnostic information regarding the type of test failures in addition to detecting and alerting the presence of faults. The diagnostic data can be utilised to pinpoint the source of the failure.

Mapping

The mapping's goal is to provide a netlist with the desired behaviour using only cells from the standard cell library. A standard cell library is a clustering of low-level electronic logic functions like AND, OR, INVERT, flip-flops, latches, and buffers. The key feature of these library cells is that they have a fixed height, allowing them to be planned in rows and thus simplifying the process of the automated digital layout. Cells are typically optimized full-custom layouts that reduce delays and area. A standard-cell library contains a library database and a timing abstract. The views that make up a library database include a variety of logical and simulation aspects, as well as views like layout, schematic, symbol, and abstract. The timing abstract gives each cell information on timing, power, and noise along with functional definitions (liberty file). The following extra elements could also be included in a standard-cell library:

- Transistor models of the cells (Spice).
- DRC rule decks.

The synthesis tool explores the power, speed, and size space to provide different implementations that meet the provided constraints. Each implementation can differ significantly from the others, each of which minimizes a metric or a combination of metrics. During mapping, static timing analysis and power analysis are performed for the purpose of analyzing and optimizing the critical path and the requested power. Both time analysis and power modeling theoretical elements are introduced in section 2.7 and 2.8. A verification step is used to to verify that the synthesis result is logically equivalent to the original RTL description.

Example of AREA report:

```
Number of ports:                           16
Number of nets:                            94
Number of cells:                           82
Number of combinational cells:             82
Number of sequential cells:                 0
Number of macros:                           0
Number of buf/inv:                         12
Number of references:                      11

Combinational area:          109.325999
Noncombinational area:         0.000000
Net Interconnect area:        undefined   (Wire load has zero net area)

Total cell area:             109.325999
Total area:                   undefined
```

Example of TIMING report with constraints (slack positive):

```
Startpoint: B[0] (input port)
Endpoint: Co (output port)
Path Type: max

Des/Clust/Port       Wire Load Model        Library
------------------------------------------------------
RCA

  Point                                                  Incr       Path
  ------------------------------------------------------------------------
  input external delay                                   0.00       0.00 f
  B[0] (in)                                              0.00       0.00 f
  add_1_root_add_58_2/B[0] (RCA_DW01_add_0)              0.00       0.00 f
  add_1_root_add_58_2/U14/ZN (OAI21_X1)                  0.05       0.05 r
  add_1_root_add_58_2/U1/ZN (INV_X1)                     0.02       0.07 f
  add_1_root_add_58_2/U13/ZN (AOI21_X1)                  0.06       0.13 r
  add_1_root_add_58_2/U3/ZN (INV_X1)                     0.02       0.16 f
  add_1_root_add_58_2/U12/ZN (OAI21_X1)                  0.04       0.20 r
  add_1_root_add_58_2/U11/ZN (OAI21_X1)                  0.04       0.24 f
  add_1_root_add_58_2/U10/ZN (OAI21_X1)                  0.05       0.29 r
  add_1_root_add_58_2/U2/ZN (INV_X1)                     0.02       0.31 f
  add_1_root_add_58_2/U9/ZN (AOI21_X1)                   0.06       0.37 r
  add_1_root_add_58_2/U5/ZN (INV_X1)                     0.02       0.40 f
  add_1_root_add_58_2/U8/ZN (OAI21_X1)                   0.04       0.44 r
  add_1_root_add_58_2/U7/ZN (OAI21_X1)                   0.03       0.47 f
  add_1_root_add_58_2/SUM[4] (RCA_DW01_add_0)            0.00       0.47 f
  Co (out)                                               0.00       0.47 f
  data required time                                                0.50
  data arrival time                                                 0.47
  ------------------------------------------------------------------------
  slack (MET)                                                       0.03
```

Floorplanning

A floorplan of an integrated circuit is a schematic depiction of the location of its principal functional blocks and pins in electrical design automation.
Floorplans are developed in the current electrical design process during the floorplanning design stage, which is an early stage in the hierarchical approach to integrated circuit design.

Placement

The tool places the various blocks and I/O pads across the chip area, following the design constraints. Physical components are placed and analog blocks or external IP cores are integrated. Placement tools have many inputs, such as a post-synthesis netlist, a layout view (LEF) for geometry and pin position, a timing description (.lib), and a power dissipation description. They place cells in a row of equal height, connecting them to power and ground lines.

Routing

Routing is the process of connecting each element of the design together, starting with the clock tree synthesis (CTS) and then designing logical interconnections. Place and route step produces a GDSII (GDS2) file. GDSII is a binary file format that hierarchically represents planar geometric forms, text labels, and other layout-related data. It is the file that the foundry uses to create silicon.

Verification

A detailed view is available after the physical design. Signoff checks are three-step physical design verifications. They are used to check if the layout works as it is designed to do.

The practice of comparing the geometry/layout to the schematic/netlist is known as layout versus schematic (LVS). Design rule checks (DRC) is the process of ensuring that the geometry in the GDS file complies with the guidelines provided by the foundry. Logical equivalence checks (LEC) is the process of checking the equivalence between RTL and post-synthesis and post-layout design.

Other custom verification could be applied at this step, such as a temperature check. A temperature check is used to better evaluate the distribution of heat the heat on the chip.

2.2 Metrics

Synthesis and optimization techniques have the main objective of creating a lower abstraction level representation of the design that is logically equivalent to the starting RTL design and with high Quality Of Result (QOR).

The main metrics are based on the composition of the circuit and are used to measure the synthesis results. Delay, area, and power consumption are the central metrics for evaluating a circuit. Delay is the time it takes for an impulse to travel from an entire path. The area is the dimension of a chip. Power consumption is divided into two types: dynamic and static; It measures the amount of power necessary for the system. New metrics, such as routability, have become central with the advancement of technology and increased density. Testability is the propensity of a system to be adequately tested. It became central with the increase in the number of transistors in a chips.

2.2.1 Pareto curve

Optimisation of one parameter most often leads to the deterioration of another parameter. This is because optimisation aims to decrease that parameter as much as possible at the expense of the others. As an example, in the project case, the optimization of the design with respect to timing violations can introduce a power consumption downgrade. Optimization techniques aim to find the respective minimum for parameter tuples respecting all constraints, that is called optimum.

A point is a Pareto Optimal point if there is no other point with at least one better parameter and all others are inferior or equal. A Pareto point definition is used to define a minimal point with respect to two or more metrics. In this work, the definition is used to identify great area/power/timing trade-offs. Vilfredo Pareto introduced the idea of the Pareto Curve in 1964[12]. The curve formed by the set of Pareto Optimal points is called the Pareto curve (figure 2.3).

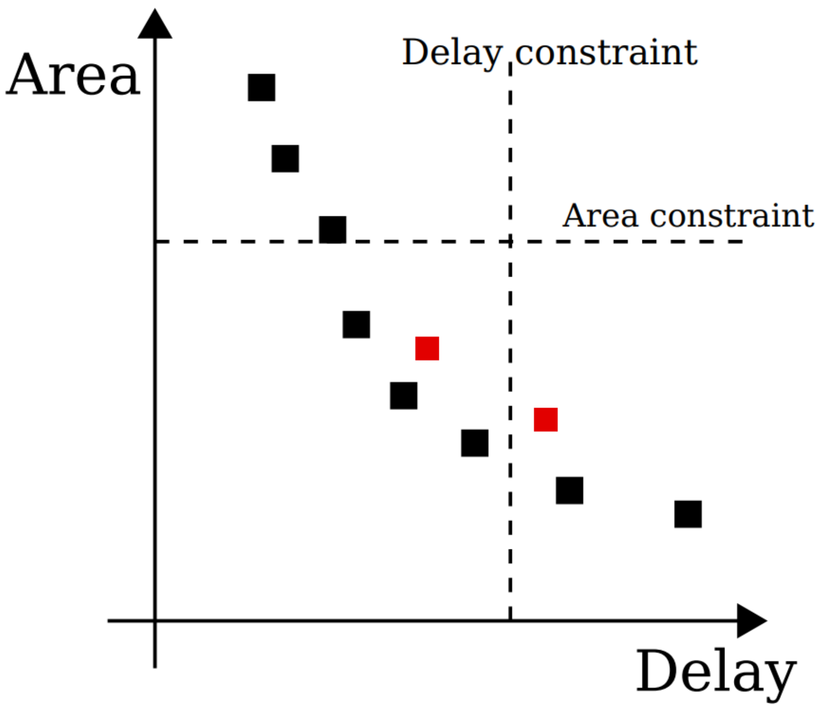

Figure 2.3: In black, Pareto point for a Delay-Area trade-off. In red, not optimal point.

2.3 Abstract models

The objective of synthesis is to take an abstract model of a higher level and produce an abstract model of a lower level. The lower level abstract model contains more information and more levels of optimization. For RTL synthesis, the tool takes a register-transfer level model and produces a logic level model.
Based on the integration capabilities gained from 1958 to 1965, Gordon Moore (Fairchild Semiconductor and later co-founder of Intel Corporation) observed that integration complexity was increasing and would have grown at an exponential rate[11].

The driving force of this evolution is “abstraction.” The first microprocessors were developed directly from the layout level with knowledge of the entire module, from the instruction set down to the transistor layout. The design flow is now based on models, each with some characteristics and parameters.

A model is a representation of a circuit showing some information relative to the use-case of the model. It enables correct communication about the design between humans and EDA tools.

Abstraction layers and views are used to categorize models. Views are the main language to describe the design; they could be behavioural, structural, or physical. Abstraction layers are formal, they are based on graphs and discrete mathematics to avoid ambiguity; Abstract models describe problems, and create an appropriate algorithm with the properties to formalize the problem.

Abstract models could be classified as:

- Architectural-level models, which describe the model as Data-flow and sequencing graphs. It is used to abstract behavioral models.
- Logic-level models, which are used for efficiently representing logic networks and state diagrams. Logic networks are described using Binary Decision Diagram (BDD). BDD is an acyclic graph that can model a Boolean function.
- Circuit-level models, used to work with the circuit based information of the design.

The abstract models and views are well organised in a specific graph called "Y-chart". The "Y-chart" (figure 2.4) was constituted by Gajski-Kuhn in 1983. It is used to determine the different stages of synthesis and optimization.

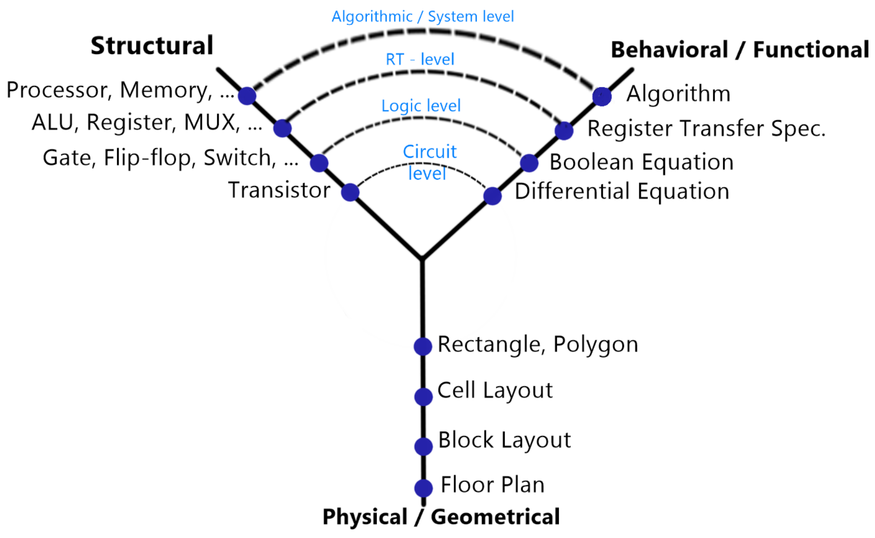

Figure 2.4: Gajski–Kuhn chart showing the many perspectives on VLSI hardware design.

2.4 Architectural Level Synthesis

Architectural Level Synthesis (ALS) is a series of techniques for transforming an abstract model of circuit behaviour to the datapath. A datapath is a collection of functional units that work together to process data. The hardware behavioural model is described using Hardware Description Language (HDL) and is used as an entry point for this technique. ALS techniques are evaluated based on some metrics, such as Area, Delay or Throughput. The main steps of a ALS are:

- Definition of the sequencing graph based on the formal information contained in the HDL;
- Optimization of the desired metrics;
- Constraint driven synthesis and optimization.

In each technique there are 3 fixed-steps:

- In Scheduling the resources are fixed in a given start time. Scheduling determines latency;
- In Binding the required resources are associated with the possible one;
- Sharing is the possibility of a resourced to be used in more than one operation. Sharing and Binding determines Area.

Performance, timing or area constraint are also considered in this technique.
In scheduling, the starting times of each operation are determined, considering the constraints. A Dataflow Graph (DFG) describing all the dependencies is the starting point of the Architectural Level Synthesis. A DFG is a direct graph which is described by a list of nodes and edges and it represents a data dependency between operations.

2.4.1 Scheduling and Binding

Binding is the process of assigning a given component to a given operation. In the case of several available components, this is referred to as an optimisation problem. The solution to the problem must aim to minimise metrics and takes place in parallel with sharing and sheduling.
Scheduling algorithms are classified in:

- Unconstrained scheduling;
- Scheduling with timing constraints;

- Scheduling with resource constraints;
- Minimum resources scheduling under latency constraints.

Unconstrained scheduling is used to calculate the minimum latency. The As soon as possible (ASAP) algorithm calculates the lowest starting time for a task. ASAP schedules the node with no data dependencies at the first possible time (figure 2.5). It starts from the node father down to the leaves (figure 2.5).

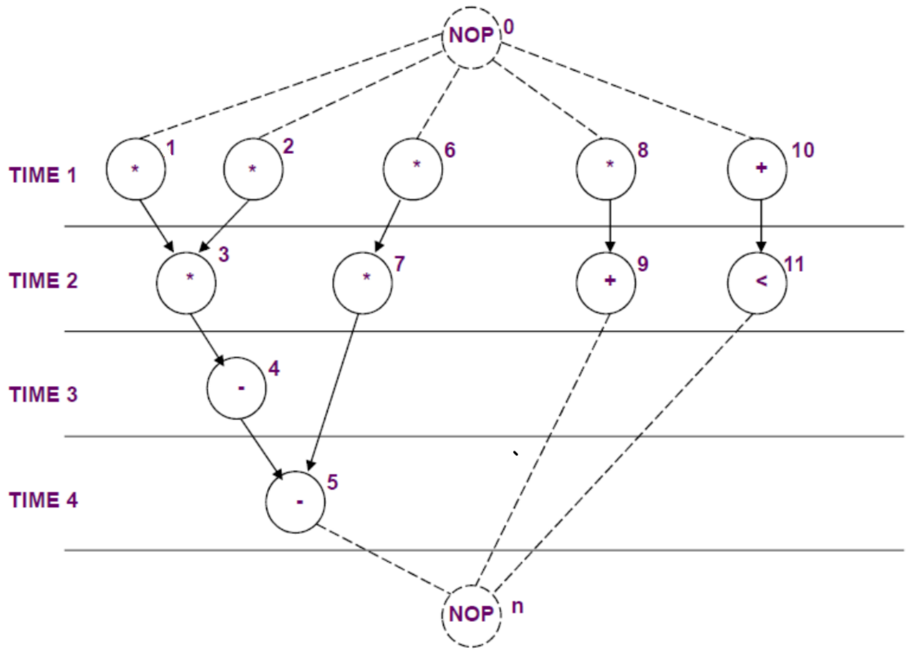

Figure 2.5: ASAP scheduling[16].

Scheduling with timing constraints introduces the concept of an external constraint. An external constraint is a limit imposed by an external necessity, such as throughput, area, or delay. A timing constraint is the maximum acceptable latency. The As late as possible (ALAP) algorithm assigns to each operation the maximum starting time possible(figure 2.6). It starts from the sink up to the node(figure 2.6).

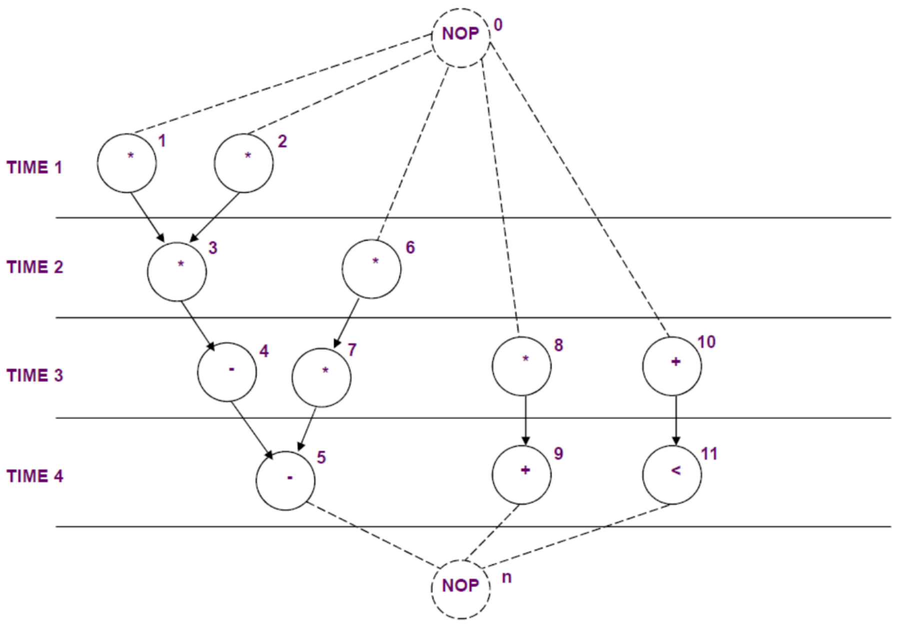

Figure 2.6: ALAP scheduling.

Scheduling with resource constraints is a type of algorithm with an exponential increase in complexity. Complexity increases due to the number of operations to be scheduled. The problem of scheduling with resource constraints can be split into two parts: scheduling with a minimum latency and area constraint and scheduling with a minimum area and latency constraint.Complexity can be decreased by using approximate methods that are faster but not exact.

Integer Linear Programming (ILP) is an exact algorithm that is used to minimize a cost function. The cost function could be related to area or delay and is an equation. The number of iterations required to solve this algorithm under a latency constraint is equal to *lambda* (latency) multiplied by the number of nodes.

List scheduling is an example of an approximate algorithm. It schedules first the node with no data dependencies and with the most resources available and the lowest slack. Slack is defined by the difference between the minimum possible starting time and the maximum.

2.4.2 Sharing

Resource sharing describes the problem of assigning a hardware resource to different tasks. Two tasks can be executed by the same resource if they need the same hardware and they are not concurrent.

The left-edge algorithm was proposed by Hashimoto and Stevens in 1971[7] and can be used for the sharing problem. The algorithm works with an ordered list of intervals and places the earlier operation first. It produces a representation of the interval and possible concurrent operations.

2.5 Logic Level Synthesis

Logic level synthesis is the process of converting an abstract specification described at Register Transfer Level (RTL) into a design implementation in terms of logic gates.

Two-level logic synthesis concerns the sum of the product expression of a Boolean function. The main purpose of two-level logic synthesis is to minimize the cover of a Boolean function. A cover of a Boolean function is a set of implicants that covers all its minterms. A product of one or some inputs is an implicant for a Boolean function if it implies a true value for the Boolean function. A minterm is a product of inputs in which each input appears once.

Multi-level logic synthesis concerns all types of basic logic gates. It is the state of the art of the current commercial synthesis tool. In two-level synthesis, behavioral and structural models have the same topology. In multi-level synthesis, we start from a Boolean function (model of behaviour) and arrive to a logic network (model of structure). Multi level's purpose is to find an equivalent representation of a given logic function that minimizes the key metrics (area, delay, power, etc.). There are no exact methods, on the other hand there are some heuristic algorithms.

Algorithms are based on five operators:

- Elimination;
- Decomposition;
- Substitution;
- Extraction;
- Simplification.

Elimination is the operation of removing a node from the network. Consequently, we have to update all the data dependencies with the node value. The main reason for the elimination is to reduce the number of levels.

Decomposition is the operation of splitting a node into two or more nodes. It increases the number of levels but decreases the cardinality of the literal set of the node.

Substitution is used to simplify a node using a local function. The division of the node for the local function returns the new node if the division has no remainder. This operation reduces the number of literals and avoids the recomputing of the same node.

Extraction finds a common sub-expression between two or more expressions. Extract it as a new function and introduce a new block. decomposition of a node and substitution in other nodes.

Simplification is used to simplify the local function of a node using a minimizer.

2.6 Verification Techniques

Modern chips are highly complex and can contain millions of transistors. Checking the correctness of the design flow output is a key point for a high Quality Of Result (QOR) design. The growing size and complexity of new chips produces the necessity for faster and more exact verification techniques. Verification can consume up to 70% of a design's time. Verification is used to check that a design meets requirements and specifications.

"Verification. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation."[9]

"Validation. The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification."[9]

2.6.1 Simulation

Simulation is a dynamic verification technique that verifies a model's behaviour in time. Simulation concerns the evaluation of the design output for some input stimuli. A simulator is a tool that can calculate signal values over time. It generates waveforms to check the behaviour of the systems. A waveform is a curve that represents the wave's shape at a specific time. Simulation is also used to evaluate the behaviour of signals over time and derive certain parameters such as switching activity. Switching activity is the measuring of changes in signal levels.

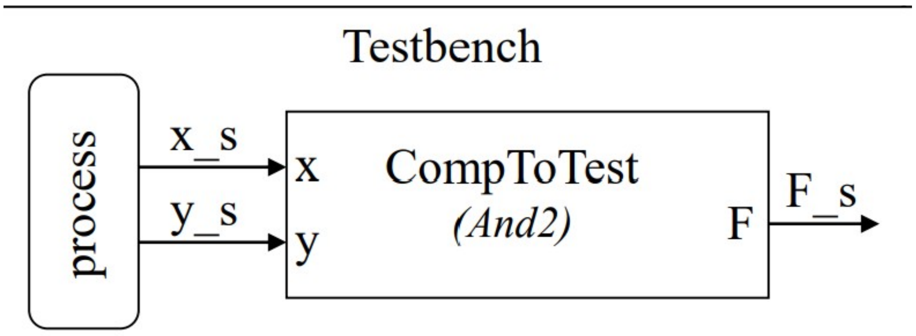

Figure 2.7: A Testbench entity

Simulation is performed by simulating a special entity called Testbench (figure 2.7). The testbench is an entity written in an HDL language. It implements the components under test and checks the components with some input stimuli.
The number of input stimuli needed to check all possible input combinations grows exponentially with the number of input 2^{input}. In large designs such as VLSI, the needed test time exceeds the possibility. The concept of coverage is introduced as the number of input stimuli on the number of possible inputs.

RTL description of a Testbench for a "numbit" MUX:

```
entity TBMUX21_GENERIC is
end TBMUX21_GENERIC;

architecture TEST of TBMUX21_GENERIC is

        constant NBIT: integer := 16;
        signal  A1:     std_logic_vector(NBIT-1 downto 0);
        signal  B1:     std_logic_vector(NBIT-1 downto 0);
        signal  S1:     std_logic;
        signal  output1:        std_logic_vector(NBIT-1 downto 0);
        signal  output2:        std_logic_vector(NBIT-1 downto 0);

        component MUX21_GENERIC
        Generic (NBIT: integer:= numBit;
                 DELAY_MUX: Time:= tp_mux);
        Port (  A:      In      std_logic_vector(NBIT-1 downto 0) ;
                B:      In      std_logic_vector(NBIT-1 downto 0);
                SEL:    In      std_logic;
                Y:      Out     std_logic_vector(NBIT-1 downto 0));
        end component;
```

```
begin

        U1 : MUX21_GENERIC
        Generic Map (NBIT, 3 ns)
        Port Map ( A1, B1, S1, output1);

        U2 : MUX21_GENERIC
        Generic Map (NBIT)
        Port Map ( A1, B1, S1, output2);



                A1 <= "0000000100000001";
                B1 <= "1000000000000001";
                S1 <= '0', '1' after 5 ns;


end TEST;
```

2.6.2 Formal verification

Formal verification is the process of applying formal mathematical techniques to demonstrate or refute the correctness of the intended algorithms supporting a system concerning a certain formal specification or property. The design is a set of interacting systems, each with a set of possible configurations, or states. A Finite State Machine (FSM) is a composition of states and interactions among them. An FSM can be represented by a state transition graph. Model-checking consists of exploring a state transition graph and checking the mathematical relations between each node. Equivalence checking is used to determine if another implementation of a design behaves the same as the verified one. It is used to check that a design function doesn't change after synthesis and optimization. In a microelectronic flow, formal verification is used to check equivalence between each stage.

2.7 Power Modeling

Digital integrated circuit design has become more complex. Timing analysis and power modeling are becoming part of the design flow. Power modeling has been talked about for years by the ASIC industry. PM should have become a game changer in the developing flow, but it encountered various problems due to the error-prone development of the integrated cicruits. There are no standard models and this leads to the possibility of estimation errors. The structure of a low-power synthesis has developed over the years, leading to an improvement in the standard for power modeling at the gate level.

Power consumption has two basic contributors:

$$P = P_{gates} + P_{interconnects}$$

The dissipation relative to the gate can be described as:

$$P_{gates} = P_{dynamic} + P_{short-circuit} + P_{leak}$$

$P_{dynamic}$ is due to the charging and discharging of the capacitance.

$$P_{dynamic} = \alpha * C_{load} * V_{dd}^2 * f$$

C_{load} is the total capacitance seen from the output gate, V_{dd} is the supply voltage. $f = 1/T$ is the working frequency, where T is the period.
In each period, the capacitance is charged and discharged. α is the switching activity (figure : 2.8); each gate is not switched in each clock cycle, consequently the capacitance is not discharged. Switching activity is the probability that a gate has to be switched.

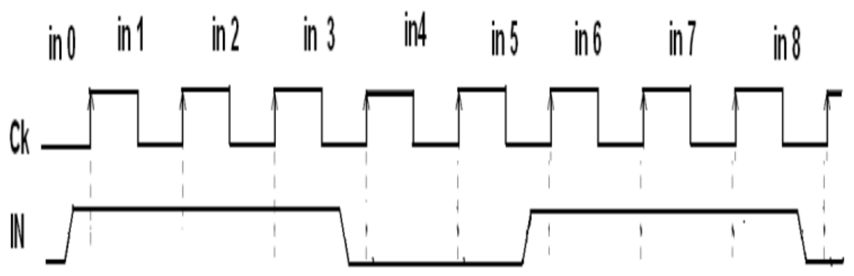

Figure 2.8: Switching Activity, representation of 4 switch on 8 clocks.

Low-to-high and high-to-low transitions are not instantaneous. Consequently, for an amount of time, both P and N networks are ON (figure 2.9) and a short-circuit current is present, $P_{short-circuit}$.

$$P_{short-circuit} = V_{dd} * (I_{PEAK} * t_{switch})/2 * \alpha * f$$

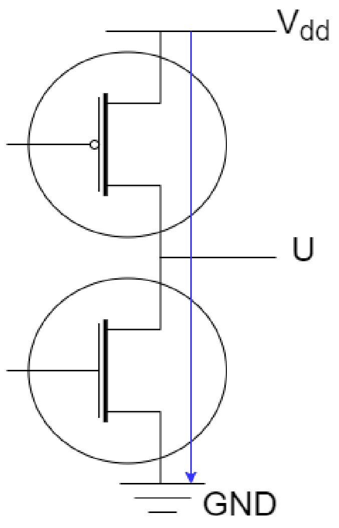

Figure 2.9: I flow on transition.

I_{peak} and t_{switch} depend on the transistor size, the total load, the input t_{rise} and t_{fall}, and the voltage threshold.

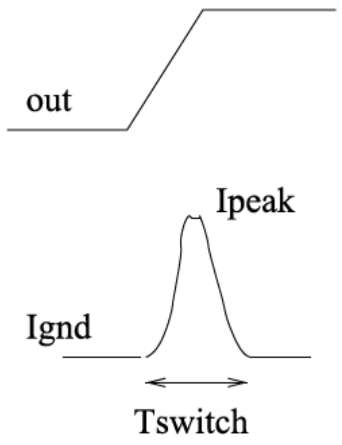

Figure 2.10: Peak current during output transition.

$P_{leakage}$ depends on the leakage current. Leakage current is due to the minority charge carriers flowing in the transistor. It depends on transistor size and temperature.

$$P_{interconnects} = V_{dd}^2 * \alpha * f * C_{int}$$

C_{int} is the total interconnection capacitance. Interconnections are also resistive; consequently, a part of power is dissipated in heat due to the Joule effect.

2.7.1 Switching Activity

Switching activity is a key measure for evaluating power consumption. It is affected by probability and switching density.
A probability of switching activity is the possibility that a signal has the value ”1”. Thus, if the signal is always ”0”, the probability is 0. The number of switches in a clock cycle is calculated as toggling density.
A digital simulation is important to measure the switching activity. Therefore, it is influenced by the behavior of the testbench. The simulation information is written to a file in VCD or FSDB format.
In power analysis, the power dissipation of a circuit is calculated. The required information, such as leakage power, capacitance and wire load model, is embedded in the cell library. This documentation is used along with switching activity data to calculate the design's power model and circuit physical information that take into account the parasitic effects of interconnecting signals. Switching activity is also an important consideration in optimization. It can be reduced by preventing flip-flop switching and memory gating. Clock gating, memory gating and power gating are techniques to reduce switching activity.

2.7.2 Clock Gating

Clock gating is a technique that disables the clock on a portion of the design when the input to the register does not change. This is done using an enable signal (Figure : 2.11). Switching activity is reduced by disabling the clock.

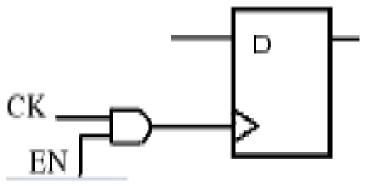

Figure 2.11: Clock Gating example with ENABLE signal.

The enable signal is controlled by a control unit that knows when the input changes (Figure: 2.12).

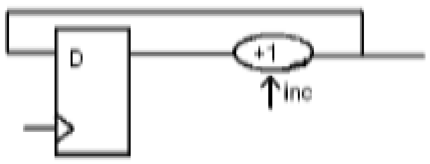

Figure 2.12: Clock Gating example with Control Unit that produce the ENABLE signal [1]. The Control Unit is composed by a counter, an input ”inc” and a logic memory containing the desired value.

There are two approaches to using clock gating. The first is done by the RTL designer, and it is introduced as functionality in the RTL. The second is applied by a synthesis and optimization tool by identifying the flip-flops that share the control logic. The simplest way to implement a clock gate is to use a ENABLE signal with a AND gate (Figure: 2.13). The clock is activated when the signal ENABLE is high.

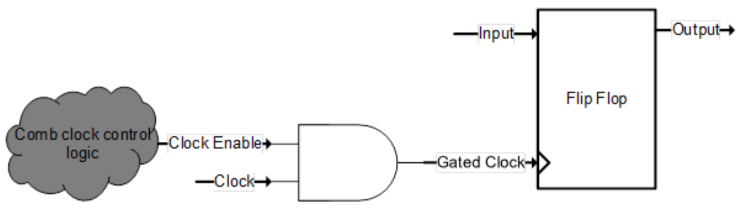

Figure 2.13: AND gate-based clock gating[1].

This solution is prone to glitches due to the asynchronous signal, the delay caused by the gate AND, and the combinatorial logic behind the signal ENABLE (Figure: 2.14).

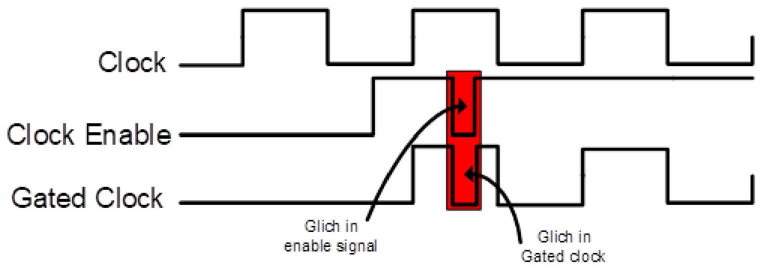

Figure 2.14: Glitch AND gate-based clock gating[1].

Synthesis tools use custom cells for clock gating called Integrated Gated Clock (ICG) (Figure: 2.15). IGC cells consist of a AND gate, an XOR gate to check the equivalence between the D value and the Q value, and a sensitive low-level latch to synchronize the enable signal.

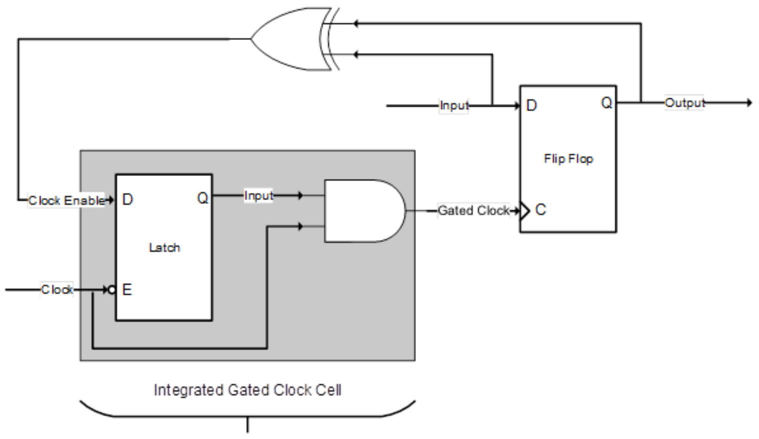

Figure 2.15: Integrated Gated Clock (ICG) cells[1].

Multi-level clock gating can be achieved by transferring the enable signal from a higher hierarchical level to a lower level (Figure: 2.16).

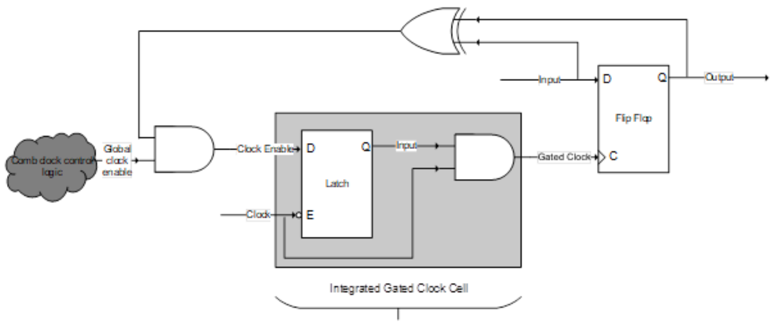

Figure 2.16: Multi-level propagation of Enable clock gating signal[1].

There are 3 options that typically characterize the clock gating techniques to be used in the synthesis tools:

- Fanout is the maximum number of FFs that can be gated;
- Bitwidth is the minimum number of bits of a register that can be gated by an ENABLE signal;
- Stage is the number of levels of multi-level clock gating.

Clock gate cloning

Clock gate cloning duplicates an ICG cell while maintaining the same enabling logic. It is applied in two cases:

- The fanout of the IGC cell is greater than the maximum;
- The output of the ICG cell is spread over a large area.

The second case is important because a large area entails longer interconnections (Figure: 2.17), which is a key timing constraint problem. A clustering algorithm is used to divide the FFs among the different ICG cells. The clustering algorithm considers the fanout and wire lengths. The number of ICG cells is referred to as sets.

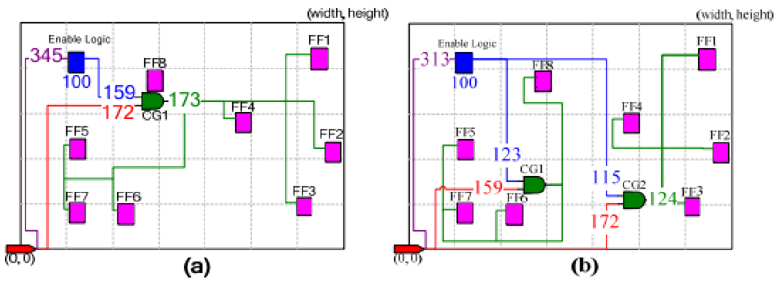

Figure 2.17: Figure a : Clock Gating without cloning, Figure b : Clock Gating with cloning and fanout distribution[3]

2.8 Timing Analysis

Static timing analysis (STA) is used to evaluate the timing performance of a design and to identify any possible violations. Dynamic simulation could be used to evaluate the timing performance, but, due to its complexity, it cannot be exhaustive. Therefore, STA is faster and more comprehensive. STA is faster because it does not require simulation time, and it is comprehensive because it examines the route rather than the pure logical circumstance from the testbench.

Some metrics are defined for each node in a network:

- The time taken by the node during the propagation process;
- Data arrival time: the time when an input signal is ready for processing;
- Data required time, i.e. the time constraint for data arrival time;
- Slack is the amount of time lag between data requests and data arrival.

A STA tool calculates the timing perspective of the design after dividing the design into a series of paths. The information about the timing behaviour of the cell is contained in the cell library or can be calculated from an SDF file that contains back-annotated delay information. The analysis is done with respect to different corners, each corner is defined as a different situation (worst, best, leakage, scan). Hold and setup time are calculated for each timing analysis. Setup violations are solved in previous stage, hold violations are solved in physical stages.

Timing constraints for synthesis are created by the designer and are a collection of constraints applied to a given set of paths or nets that determine the intended performance of a design. Period, frequency, net skew, maximum delay between end points, or maximum net delay are all possible constraints.

STA checks the design constraints after reading the cell information and the design data. The constraints result from the behaviour of the clock tree and the registers.

2.8.1 Critical Paths

The critical path is the longest sequence of components in the circuit, and it is the upper bound of the clock speed.

In STA, the critical path is the one that has slack of 0, i.e., the one that has the maximum latency for the given constraints.

Constraint could contain execptions, as false path. A false path is a path where the input signal could not be propagated to the output(Figure: 2.18). Synthesis tools reduce the number of false paths by logical optimization. STA has to take into consideration the presence of false paths for timing analysis.

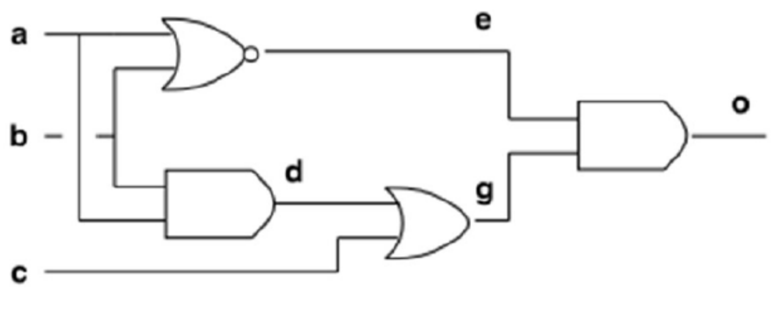

Figure 2.18: Critical path composed by nodes A,D,G,O is a false path.

2.9 Practical example of a RTL synthesis flow

Synthesis is the transformation of a concept into something useful and capable of satisfying the concept.

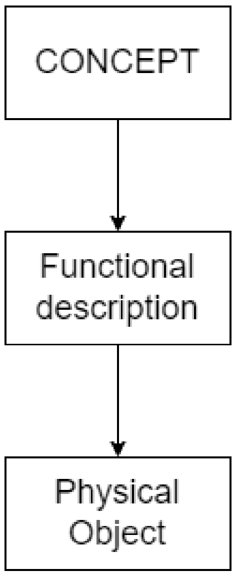

Figure 2.19: Synthesis from a concept to an object.

RTL synthesis is a process that generates a gate-level netlist from a block-level RTL design with acceptable post-placement timing and congestion.
The RTL synthesis flow includes the following steps:

- Reading hierarchical block-level RTL designs;
- Loading libraries, technology data and floorplan constraints;
- Applying and checking timing constraints;
- Synthesizing RTL to gate using a top-down or a bottom-up approach;
- Generating timing reports
- Analyzing and improving layout congestion
- Generating output file for physical design tools

Synthesis concerns three parts: Translation, Logic Optimization, and Gate Mapping. Translation from an RTL source to a Generic Boolean Gate (unmapped ddc format). Logic optimization works with a boolean equation using the logic level optimization technique. The last part concerned the mapping of the gate. Transform an unmapped ddc format to a mapped one. Constraints are important before optimization to prevent useless implementation. Synthesis ends with the saving of the results.

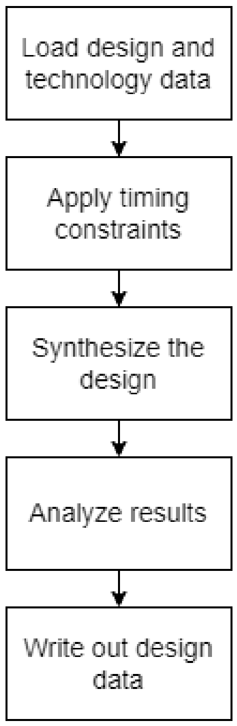

Figure 2.20: RTL synthesis flow.

2.9.1 Loading Design and Technology Data

The first step of RTL synthesis is to load the necessary information. The information loaded in this step is:

- Design description in an HDL language.
- Logical libraries (db file).
- Physical or layout libraries.
- Routing layer definition file (tf file).
- Floorplan data (TCL physical constraint or DEF file).

The last four files are specially used in topographical synthesis. Topographical synthesis performs placement-driven design mapping and optimization to achieve high QOR and facilitate the back-end flow.

Loading Logical Libraries

The name, cell area, timing behavior, functionality, design rule constraint, and electrical characteristics for each pin are all embedded in a .lib file.
Example of a OR description in .lib format:

```
cell ( or2_4x ) {
    area : 3.00 ;
    pin ( Y ) {
        direction : 2 ;
        timing ( ) {
            related_pin : "A" ;
            timing_sense : positive ;
            rise_propagation ( ) {
                value ("0.22, 0.28, ...")
            }
        }
        function : "(A | B)";
        max_capacitance : 2.90 ;
        min_capacitance : 0.02 ;
    }
    pin ( A ) {
        direction : 1;
        capacitance : 0.12;
        ...
    }
}
```

Logical libraries are used during compilation to create a netlist with technology specifications.

Loading RTL Designs

Hierarchical design is read starting from the bottom entity HDL file and going up to the top entity HDL file. The reading phase could be split into ”analyze” and ”elaborate” phases. In Analyze, the HDL description syntax is checked. Elaborate is used to connect specific designs and to establish the top entity.

Loading Physical Data

Some physical data is needed for topographical synthesis. Reference libraries of standard cells, IP or macro cells, and I/O pad cells contain information about size and pin locations.

Technology files contain definitions of routing layers, vias, and layout rules. It also contains information about parasitic net models. The technology files have the typical ".tf" extension. Each file is unique to each technology and describes metal layer technology parameters such as:

- Name, number, physical and electrical characteristic for each layer;
- Dielectric constant for technology;
- Units and precision for the value.

Floorplan information contains constraints on the placement of the cell that are needed in topographical synthesis.

The ASIC Vendor provides reference libraries and technology files. Floorplan information is from previous analysis of the netlist and specification.

2.9.2 Constraints application

Physical constraints are used in topographic synthesis. This constraint regards the floorplan and could be about the core area, ports, macros, and placement. The floorplanning constructs and constraints are contained in Design Exchange Format (DEF). Some options could not be embedded in an DEF file, so an optional tcl file is used.

From timing perspective, timing constraints are utilized to ensure that a design is functional and that it will perform as intended once it is manufactured.

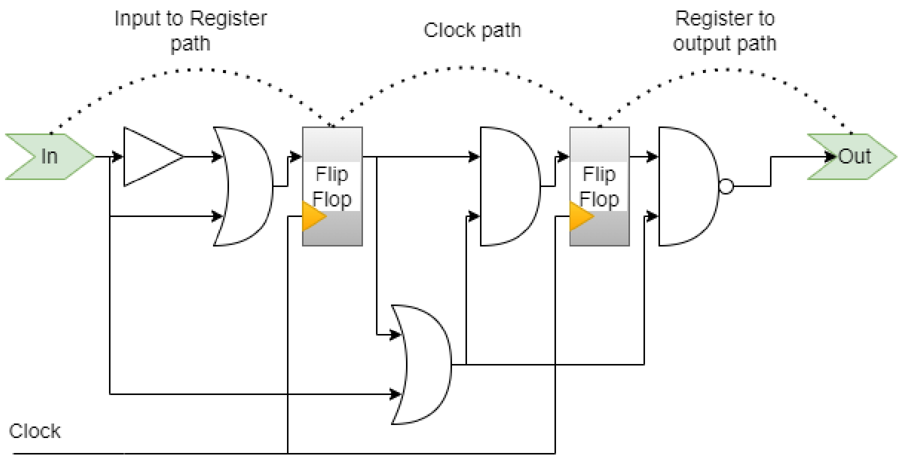

Figure 2.21: Example of timing path.

In figure 2.21, three path are underlined:

- Input to register path;

- The clock path, is called clock path because the timing length is equal to one clock cycle;
- Register to output path;
- InOut path, sum of the previous one.

There are three types of timing constraints that can be identified: Clocking Requirements, Boundary Settings, and Timing Exceptions.

Clocking Requirements

Clocks should be defined with the following parameters:

- Clock period, period contain the delay value of the clock path (figure 2.21);
- Clock source: port, net, pin or virtual;
- Duty Cycle, Skew and Uncertainty;
- Clock tree propagation latency;
- Rise and fall time.

Boundaries Settings and Timing Exceptions

Input and output delays are constraints applied to the I/O ports. They are the arrival and required times that should be considered by a block.

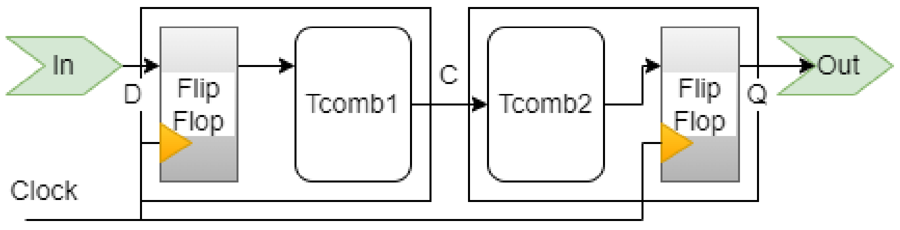

Figure 2.22: Timing constraints on input and output port.

In figure 2.22, an input delay example in C could be the clock timing to Q plus the Tcomb1 delay. The output delay on port D should consider the set-up time of the Flip Flop plus the Tcomb2 delay.

Tcomb2 with Tcomb1 is a combinational path. A minimum or maximum delay could be constrained.

2.9.3 Synthesize the Design

Synthesis is the central part of the flow, a brief introduction is presented in section 2.1. Good synthesis can only occur through two excellent previous steps. For example, an HDL code that leaves no optimisation possible at the architectural level results in a worse synthesis. Missing constraints could result in incorrect synthesis or worse QOR.

The synthesis approach must cover the entirety of the design in order to give the EDA tools the possibility of possible optimisation.In this flow, a topographical synthesis is considered, which performs an initial placement to improve the synthesis possibilities and to have greater reliability in the results.

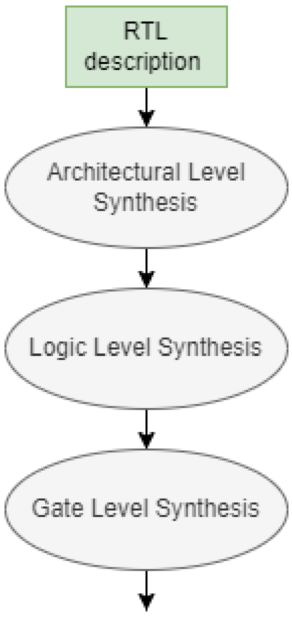

Figure 2.23: Default synthesis step.

In figure 2.23 the synthesis step is explained. The EDA tool uses three-level optimization to produce better QOR.

Each of these three-step synthesises produces some changes in the design. The produced design could not be accomplished with the timing constraint. For this reason, logic level and gate-level optimization techniques could be reused. For example, the duplication technique increases area but could solve the longest path timing problem.

Resource sharing is a technique of architectural level synthesis. This technique could be used only in well-coded RTL. An example of a component in which resource sharing is prevented:

```
Add1 <= A + B;
Add2 <= C + D;

if (sel == 1'b1)
    sum <= Add1;
else
    sum <= Add2;
end if;
```

Other technique of architectural level synthesis is:

- Boundary Optimization, it consist in optimization in the first/last gate of the module;
- Register Replication, it is used to deal with critical path timing;
- Adaptive retiming, involves moving the register across the combinational circuit to reduce the timing violations. Adaptive retiming could move, duplicate and merging register, but the "End-to-End" behaviour of the circuit is unchanged.

2.9.4 Write out results and Design Data

Congestion analysis using the heat map allows for identifying floorplan-based congestion and netlist topologies-based congestion. Netlist topologies-based congestion could be reduced with Physical optimization. Floorplan-based congestion could be optimised by modifying the floorplan.

Report Quality Of Result (QOR), Area, Timing and Power are used to analyze the synthesis results.

A timing report is used to determine all the constraint violations in the design. That could be related to low-quality RTL, wrong constraints, or errors in synthesis. Consequently, designers have to analyze the violating timing path.

Data output needed for Physical Design:

- Gate-level netlist;
- SDC constraints file;
- Floorplan DEF and Tcl files;
- Scan chain definitions;
- Propagate backward SAIF, power profile;
- Unified Power Format (UPF) command script with power intent.

SCAN-DEF file contains the scan chain information.

2.9.5 Two-pass synthesis flow

To perform a topographical synthesis, a floorplan is needed; to get a floorplan, a netlist is needed; and to get a netlist, a floorplan is needed. This problem is a Catch-22 problem. A catch-22 is a paradoxical circumstance from which a person is unable to escape due to competing rules or restrictions[8].

A solution to this problem is called "Two-pass synthesis." The synthesis tool would then work with a standard floorplan. The first synthesis loop produces a floorplan to be used in a second complete synthesis.

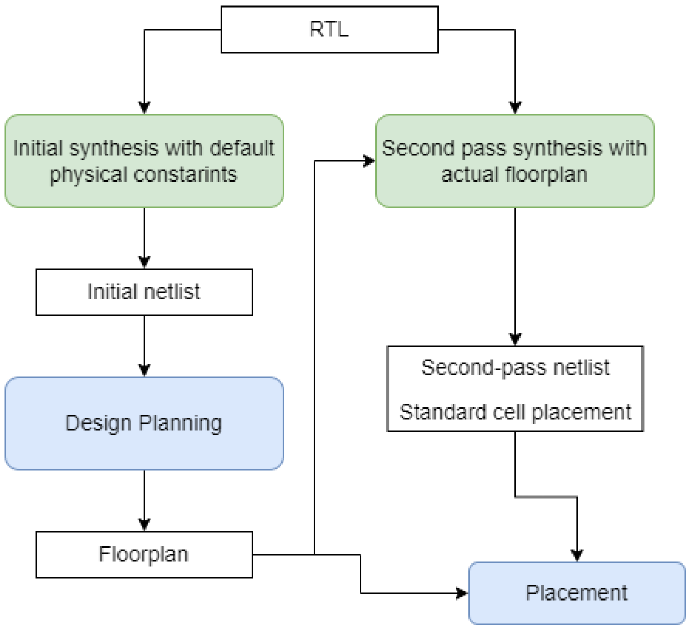
RTL
Initial synthesis with default physical constarints
Second pass synthesis with actual floorplan
Initial netlist
Second-pass netlist
Standard cell placement
Design Planning
Floorplan
Placement

Figure 2.24: Two-pass synthesis flow.

In figure 2.24, on the left the first pass synthesis, on the right the second pass synthesis. The first pass starts with a constrained synthesis using a default floorplan. Also, a user-defined custom floorplan could be used. The initial netlist is the output and it is sent to a design planning tool that generates the floorplan. The second pass starts using the produced floorplan. The netlist produced with the floorplan is used for placement. This technique should be used with challenging QOR requirements. Re-synthesizing with a more accurate starting point produces the best result. But if QOR is not critical, the second pass could be skipped in order to reduce the synthesis duration.

3 Approach

3.1 Synthesis Purpose

The objective of the thesis is the formation of a stable synthesis and optimization flow that leads to consistent and valuable results.

The consistency comes from the ability of the flow's ability to be replicated in different designs with minimal changes. Value is determined by the best results of the flow that minimize metrics and adhere to constraints. The results must be certain and verified.

The design to be analyzed is a complex mixed-signal component with memory and a core unit. The modules that make up the design are important to identify better options in the flow. A large memory could have an important improvement by using an important clock gating structure. This optimization is due to the mapping-structure of the memories that increases the probability of more registers switching at the same time. A core logic needs cells with low delay, and a peripheral controller needs cells with high strength. Most of the time, pads and memories are external modules, therefore they are not part of the design. In the specific case of the analyzed design, memories are both internal and external, as pads.

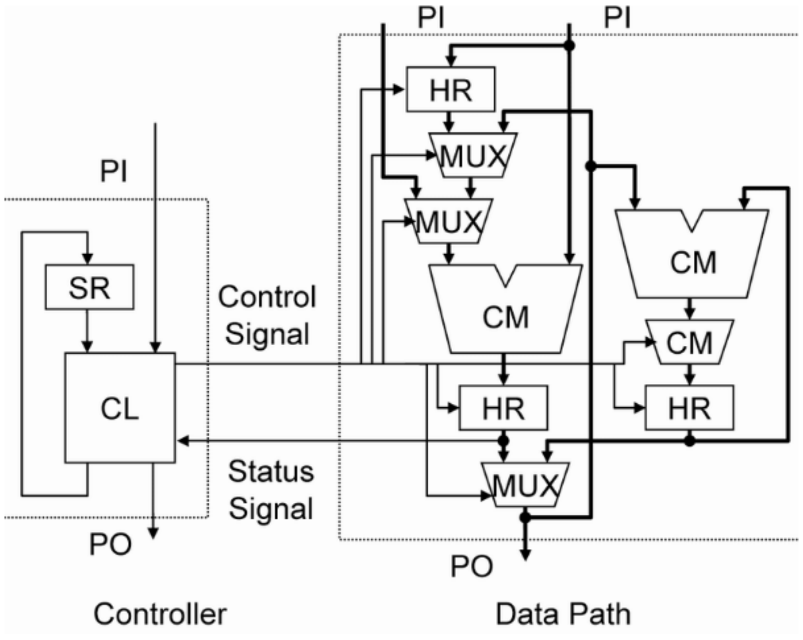

Figure 3.1: Example of an ASIC RTL schematic with datapath and control unit[6].

The main metrics to be met and optimised are area, delay and power consumption. These metrics are influenced by each other and other metrics such as congestion, fanout and strength of the gate, temperature, placement, etc.

The synthesis tool is heuristic and not exact. The synthesis time is also a metric, as usually better results can be obtained with a longer synthesis time. The synthesis time is constrained by the time to market of the product and the development time. Time to market is the time needed before the product can be sold. It is important to meet industry requirements. Development time is a non-recurrent cost, it is a one-time cost.

The objective of this work is to understand the effect of different synthesis settings and parameters (such as use of specific library subsets, clock gating, etc.) in the final results and to propose and compare different flows. The final netlist should be evaluated and verified.

3.1.1 Custom synthesis parameters

The approach to synthesis flow is both theoretical and experimental. The tool's algorithm for synthesis and optimization is heuristic rather than exact. Experimentation is the key to developing strategies with higher QoR. The basic flow was presented using only theoretical applications (Section 2.9). The basic flow serves as a starting point for feature development and investigation. It consists of three macros: Elaboration, First Compilation, and Second Compilation.

The elaboration phase is the setup part of the flow. It provides the EDA tool with the main information about the design. HDL files and cell libraries were read and analyzed. Both are linked together and the resulting design is checked. The cell libraries are used in this step, so all decisions about the use of the cell libraries should be made in this step. In the strategies presented below, the cell libraries are limited to a portion of the design or can be disbaled in an early stage of the flow to be used in the next phases.
Clock-gating techniques are introduced in the initial compilation phases. The first compilation phase is characterized by the elaboration of the timing constraint and the application of parameters such as clock gating, register cloning or pin buffering. After the first compilation, area optimization must be performed. Clock gating techniques are our main goal in this phase. This work focuses on adapting clock gating techniques to the given design and analyzing the results of different clock gating applications.
The second compilation introduces new concepts that will be used in the development of this work: the idea of incremental compilation, the clock-gating cloning technique, and the possibility of enabling cell libraries in future compilation phases. After compilation, an area optimization phase is introduced. This is a standard sequential command of each compilation phase. One of the strategies explored in this work is the use of additional cell libraries in subsequent phases. This is possible thanks to incremental compilation. An already synthesized netlist is used as a starting point for the introduction of new cell libraries. This application can also be applied to some modules to fix errors in their timing path.
The steps presented previously are those that characterize the basic flow. More steps have been analyzed in the different strategies presented in this work. The introduction of new stages opens the door for new analyses on new strategies related to different areas: cell libraries applied in the next stage, and different clock gating approaches in different compilation stages.
The concept underlying this evaluation is that each netlist can be used as a starting point for new compilation steps. Consequently, one synthesis strategy could be mixed with another at different stages to manually solve the resulting problems.
The strategies applied to the basic synthesis flow fall into two categories: Flow Structure and Parameter Variation. Parameter variation strategies can be divided into three categories: Clock control parameters, cell libraries, and placement parameters.

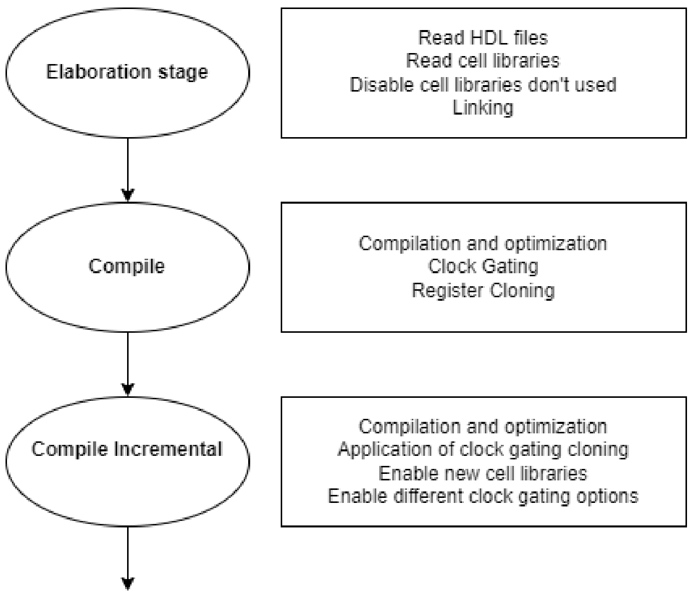
Elaboration stage
Read HDL files
Read cell libraries
Disable cell libraries don't used
Linking
Compile
Compilation and optimization
Clock Gating
Register Cloning
Compile Incremental
Compilation and optimization
Application of clock gating cloning
Enable new cell libraries
Enable different clock gating options

Figure 3.2: Phases of basic design flow and application location of new strategies.

Clock Gating approach

Clock gating is used to reduce the dynamic power consumption of the system. It introduces logic into the clock tree, as well as delays that could be reported as timing violations. The logic cell used for clock gating is called integrated clock gating (ICG). In this project, many aspects of clock gating are analyzed. The basic procedure is characterized by clock gating with cloning. Cloning is a technique to duplicate a clock cell to minimize the wire length. The wire length leads to capacitance and hence delays.
The first experiment in this work is related to the deactivation of cloning. This is done to understand if cloning is used to prevent timing violations and to know how many cells were cloned. The number of cloned cells can be calculated by simply taking the difference between the number of clock-gated cells before and after cloning. More cells means higher leakage power and more area. Cloning is done by introducing boundary as a module to the clock tree; it is introduced during the second compilation (the first incremental one). Removing the boundary is sufficient to disable cloning.
The contribution of clock gating could be calculated with an analysis of the base flow when clock gating is disabled. Clock gating introduces delays and more area, but reduces power consumption. The main impact of clock gating is the clock tree. The difference in the number of buffers in the clock tree gives an idea of the change in the timing model due to clock gating. A large number of buffers could be used to balance the clock tree between designs. If the number of buffers decreases significantly, it

is likely that the number of clock gating cells is not evenly distributed across the designs, and planned clock gating cloning could solve this problem without using a large number of buffers. Clock gating is applied in the second phase of the flow (initial compilation), but also in some RTL descriptions. The strategy is to disable clock gating insertion at the first compilation and allow the one introduced by the designer. Three main parameters characterize the clock gating technique used in the synthesis:

- Fanout of individual ICG cells;
- The highest possible bandwidth for which a clock gating cell is designed;
- The number of possible stages of clock gating.

The number of stages on the clock gating style is the number of maximum ICG cells that could be set in cascade. Each higher ICG cell drives the lowest ICG cells and other registers with similar switching activity. Lower ICG cells are used to drive a subgroup of the register with a more similar switching behavior. This parameter was used when the design was first compiled. A synthesis with higher fanout clock gating is used to reduce the number of cells and area. High fanout could be useless unless the design is expected to contain many flip-flops that switch together. A script was developed that analyzes the schematics to check when a higher fanout is useless. The script easily calculates the fanout of each ICG. Furthermore, a lower fanout could have a negative impact on the area. The lower fanout leads to the use of more ICG cells. However, these cells could be more easily distributed across the clock tree. Sometimes clock-gating cells are not instantiated due to timing violations. Lower fanout is used to reduce the timing impact. A higher fanout increases the delay in the path, but decreases the area and leakage. Lower fanout increases leakage power, but could decrease dynamic power as more flip-flops are gated. The approach to these parameters is to find a tradeoff between the number of flip-flops gated, power, and area.
The bandwidth is the minimum number of bits for which an ICG cell is instantiated. Decreasing the minimum bandwidth increases the number of flip-flops passed and the dynamic power, which in turn increases the leakage power and area. As more ICGs are instantiated, the decrease in dynamic power is asyntotic, but the increase in leakage power and area is linear. The approach to this parameter is to find the value at which the decrease in dynamic power justifies the increase in leakage power and area. Digital circuits using the power of two information systems can provide a guide to the best value.
Clock gating could have a hierarchical structure. The number of levels in this structure is called "stages". Each level has an ICG cell for the computational logic and a bypass for the gating of the higher level. The approach to these parameters is

to increase the number of stages and look for the maximum number of stages before a violation of the schedule occurs.

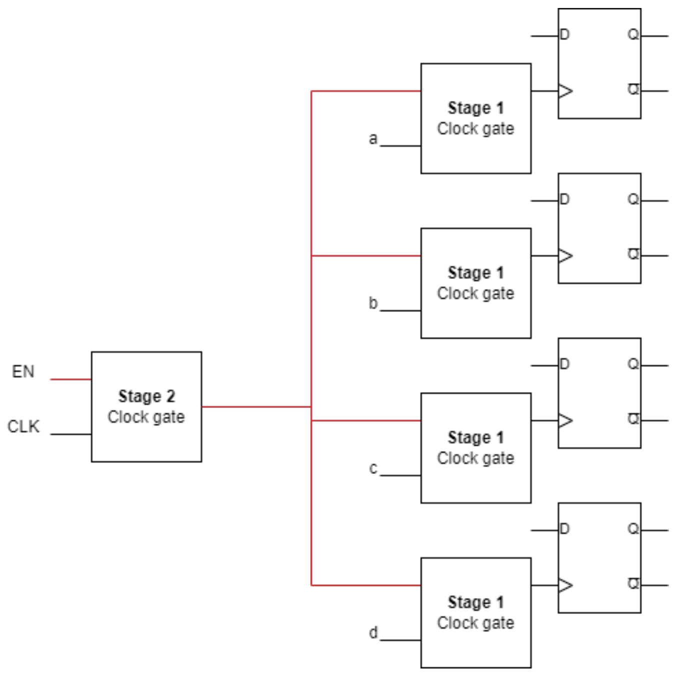

Figure 3.3: Multistage Clock Gating.

Cell library approach

Cell libraries contain sets of cells with similar physical characteristics. The main characteristics of a cell are: Vdd, timing model, internal capacitance, and dimension. Vdd is a designer choice, lower Vdd increase leakage power and decrease the transition delay. Dimension parameters are :

- L, the length of the gate. It is a technological choice, therefore it cannot be changed by the designer;
- W, the width of the transistor. It is a design choice, and influenced the strength S=L/W;
- X, the length of the drain/source;
- t_{ox}, the height of the oxide.

This thesis work addresses different types of cell libraries: low leakage and low scale. In both categories, there is one library with higher power and one with lower power. The ability to use different cell libraries opens up the possibility of different types of implementations. Synthesis strategies using only low-leakage cells lead to low-power results, but low-leakage cells have a higher delay than low-scale cells. A low-scale cell could be used to solve the timing violation. This application is used for specific modules or for the top module. The EDA tool for synthesis is able to optimize the cell for the entire design and use it only when needed. Moreover, the strategies used show that this work is not accurate and could lead to poor optimization. Adding the low-scale libraries after the initial compilation results in these cells only being used when needed for timing.

Other strategies approach

Some modules in the base flow are fixed and are set as untouchable in the range optimization. One strategy is to allow systems to change the location of these modules in advanced incremental compilation phases.
Any of the above strategies can be used after some basic compilation phases. This allows the synthesis tool to work with a design that has already been synthesized, but with a different starting point than the normal use of a different compilation. This strategy is used to check if the higher compiled stage gives the system a better quality result and if changing the starting point of the synthesis within the synthesis process leads to different results.

3.2 Hierarchical Design

Software designers have learnt to deal with more complicated programs during the last decade. A structured programming style has been created to deal with this complexity.

The designer could select from about two forms to relate to design structure:

- Top-down, the design specification is decomposed into less complex specification modules;
- Bottom-up, the design is built from a combination of lower blocks into larger blocks.

These two forms could be used to decompose three types of hierarchy: behavioural hierarchy, structural hierarchy, and physical hierarchy.

The key target of the hierarchical design methods is to reduce the design time. The use of the SCALD system for the design of the Standard-1 processor is an example of how hierarchy could decrease design time. The SCALD system automatically expands the wirelist at the physical level with a macroexpander, and a physical design system produces the wirewrapped design. The design time decreased by about 10%[10].

The second way a hierarchical design could decrease design time is through verification. Specification on the entire system and down to all the subsystems creates an easy way to formal verification. More comprehensive specifications also reduce possible misunderstandings and time loss.

The hierarchical design could also be a problem for formal verification. A large number of hierarchical behavior descriptions could exist and be very different from each other.

The hierarchical design could also preclude some structural optimization because of the existence of boundaries. To consider every optimization possibility, it is sometimes necessary to perform a macro-expansion.

Behavioral hierarchy

Figure 3.4 shows a possible hierarchical decomposition of a processor unit. The processor unit should fetch or execute an instruction. The fetching instruction step is followed by the calculation of the new address and new operands.

The hierarchical behavioural design process provides an iterative refinement of the requirements, from a global aspect and down to specific behaviour.

This information could be used to simulate and verify the design.

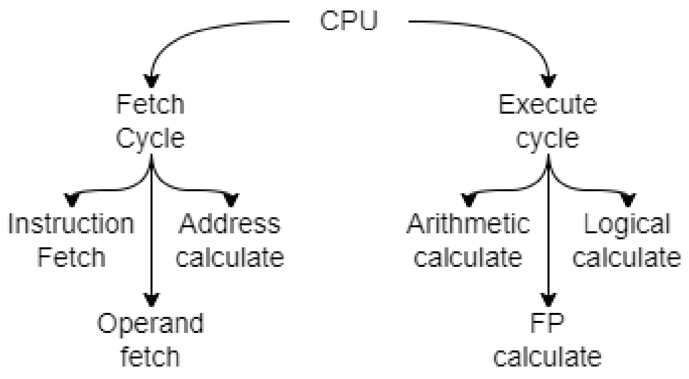

Figure 3.4: Example of Behavioral Hierarchy of a CPU.

Structural hierarchy

In the example in figure 3.5, a generic CPU is composed by an ALU and a Register file. An ALU is composed by two blocks, an adder and a Multiplier.

Structural hierarchy could be used with behavioral one, each module is also described by its behaviour. Each module can be simulated and verify by its behaviour, that is called dataflow verification. The structural modularization of the design is an advantage of the hierarchical structural design. Another significant advantage is the possibility to expand the models, which gives a user-controlled possibility for optimization.

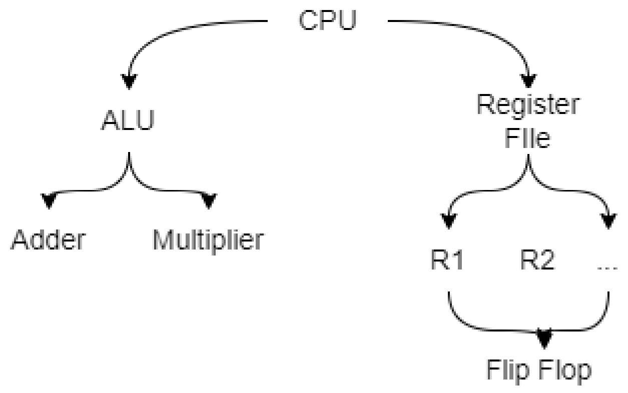

Figure 3.5: Example of Structural Hierarchy of a CPU.

Physical hierarchy

At a physical level, a designer has a set of standard circuits to deal with for implementing a design. In an integrated circuit (figure 3.6), the designer would have a hierarchy of supercells, macrocells, cells, and transistors.

The normal approach of a designer is to lay out a design from a global perspective and then successively refine it down to the transistor level. Partitioning the design into subtasks makes the mapping problem easier.

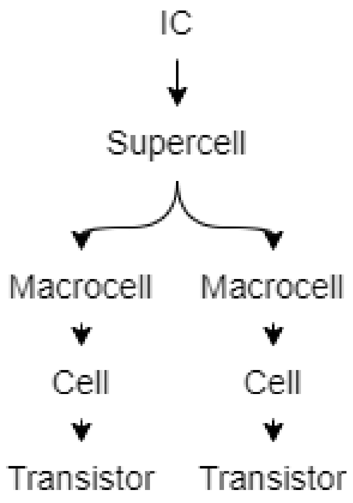

Figure 3.6: Example of Physical Hierarchy of an Integrated Circuit.

3.3 Simulation of Digital Systems

The simulation phase has two objectives: Validation of the generated netlist and measurement of the activity information.

A testbench is applied to the top module and test vectors are applied to the design. This technique is used to avoid mismatches in the netlist due to an incorrect synthesis process.

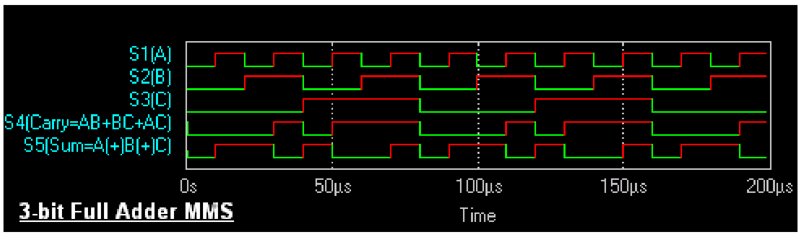

Figure 3.7: xcelium ouput waveform for a 3 bits RCA.

Measurement data, such as switching activity, is important to model the power behavior of the design. This measurement is performed in the synthesized netlist to achieve a higher correlation between the gate simulation and the gate in the netlist.

Using data from the simulation of netlists created with different flows may contain switching activity information that has a lower correlation than the synthesized netlist. Low correlation results in low quality power models.

The approach to the flow is to simulate each netlist to verify correct behavior and obtain data with high correlation.

Simulation is used to check a small set of input stimuli. Each simulation produces different switching operations depending on the given inputs. For this reason, different simulations are run with different conditions. The simulations may differ in terms of input and frequency. The power model created by this analysis should include this information about frequency and input.

This is important to be able to compare the results from the same frequency and input stimuli.

3.3.1 VCD/FSDB file

The information about the simulation is contained in a file in VCD or FSDB format. This file is used by the power modeling tool for the switching operations.

FSDB is a native Synopsys file optimized for use with a Synopsys tool. It has a smaller dimension and can contain additional information such as the strength of each signal.

3.4 Validation and Evaluation of the Synthesis and Optimization strategies

Synthesis and optimization processes change the structure of the design. This change could be reflected in an appreciation of the metrics, and the evaluation of the main metrics is the way to evaluate the main difference between the different synthesis strategies. The purpose of the evaluation leads to the choice of the most accurate methods that provide the most reliable references. Studying the development of different synthesis strategies with inaccurate evaluation methods leads to results that cannot be used.
This problem arose in the early stages of the development of the thesis. A poor evaluation method with low annotation rates of the cells derived from the simulation in the design resulted in non-compliant values. The same problems can occur in the evaluation of all other metrics, so it is important to choose standard evaluation methods with high efficiency.
Changes made by EDA tools during optimization can lead to changes in the formal logic of the circuit. For this reason, formal verification is performed. During formal verification, we check the equality of the reference logic. However, we can also use the results of formal verification to detect errors in the optimization phase, such as unused cells or floating gates. The logical equivalence checking procedure is error-prone due to the treatment of names. The first check that is performed concerns the names of the supporting cells, which must be promptly set in the verification flow.

3.4.1 Evaluation metrics

Metrics are evaluated at various points in the synthesis process. There are different metrics that are evaluated with different constraints. Timing is evaluated during synthesis based on the constraints that are inserted to describe the behaviour of the clock tree and the various registers. Variations in timing are important because it is the only constrained metric. Constraints are defined by the designer based on the system specification and optimised by the tool.
The area is calculated as the sum of the dimensions of the cells used. A floorplan file is included in the two-pass synthesis to support the evaluation of this metric.
The power analysis is performed by a dedicated tool that receives as input netlist, libraries, synthesis results and switching activity from the simulation (FSDB file). Different parameters are defined and a distinction is made between static and dynamic.
Other metrics such as congestion, fanouts in the clock tree, number of buffers, number of cells are considered as they provide information about the structure of the design. Area and power consumption are also examined hierarchically so that the design can be analysed using a top-down method.

All metrics provide information about the structure of the design and how changes in flow affect the design.
The flow parameters are then changed and the resulting metrics are analysed. The analyses are examined and justified so that intelligent changes can be made. A change in clock gating must be reflected in the number of clock cells, number of buffers, and power consumption.
Specific reports on the gating cells and that are noticeable. For an in-depth study of clock gating, specific reports are produced on the gating cells and the clock tree. For an area-specific investigation, specific analyses are created about the size of the different modules.
A change in the library also brings a change in the metrics. Thus, automatic scripts are developed to analyse all cells of the design and split them by library, type and module. This allows analyses to be performed directly on the subset and differences between the different approaches to be identified. Timing is a blocking problem that must be solved by applying specific methods. To apply exact methods, various reports are generated analysing the violations and the paths on which they occur.
RTL synthesis consists of various compilation and optimization processes. At the end of each process, all the above reports are generated for confirmation.
A final report is produced, called QoR, which provides an overview of the entire resulting design.

3.4.2 Formal verification

Formal verification is performed to check the logical equivalence between the RTL netlist and the post-synthesis netlist. The approach to verification consists of three phases. Two phases are used to map the two netlists. A third phase is used for the logical comparison.

In the first phase, name similarity is checked. This is done first because checking name equality is faster and can reduce the number of terminations that need to be solved. The synthesis tool changes the name of the node. The name changes are stored in a vsdc file. Sometimes some nodes do not have an equivalent name, so a renaming rule is calculated to prevent unmapped points.

In the second phase, the logical cone is used to map the node. This takes a lot of time because the tool has to analyze the whole design.

The third phase is the comparison and is only performed when all nodes are mapped.

Clock-gating options such as a higher number of stages can create large logical cones. This leads to aborts during the comparison. The abort is solved by introducing hierarchical modules in the cone.

The process of logical equivalence checking is shown in the figure 3.8. The various files for the netlists to be compared, the reference netlist, the revised netlist, and the reference libraries are read in. Then the design is processed and the cells between the two netlists are mapped. If a cell is not mapped, the process stops because it cannot be analysed. The cell is examined and the reason why it was not mapped (different name, missing supporting cells, etc.) is determined. The next step is to compare the two designs, the names are compared and then the logical cones. If there is an error, the process starts again from the mapping. However, before mapping, the two netlists are debugged to find the source of the error. The LEC ends when there are no more feilures are present.

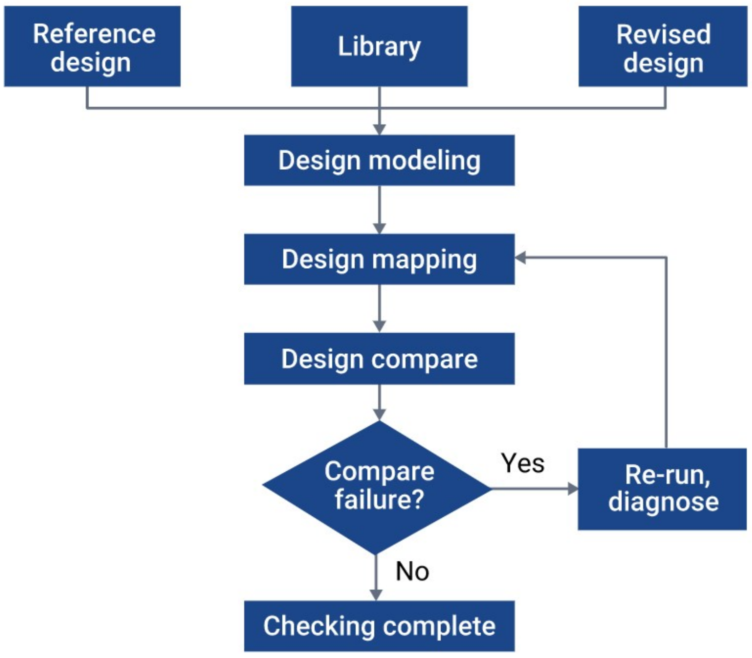

Figure 3.8: Logic equivalence check (LEC) process [15]

4 Implementation

The chapter illustrates all the implementations developed during this work.
It begins with an introduction of the EDA tools used during development and a brief reminder of the cell libraries used, with a focus on the technical features.
Design under synthesis is introduced as the basis on which all synthesis strategies are applied. The composition of the design influences the efficiency of the various techniques.
Lastly, an in-depth description of the algorithms for developing the synthesis strategies.

4.1 The used software tools

The technique presented in Chapter 3 involves a collection of Electronic Design Automation (EDA) tools.
Cadence Xcelium is used to simulate the design. Thanks to the result of the Xcelium simulation, Synopsys PrimePower is used for power estimation.
Topographical synthesis is backed by the Synopsis Design Compiler NXT. Synopsys Formality is used for post-synthesis and intra-synthesis formal verification.

4.1.1 Cadence Xcelium

"Cadence's Xcelium Logic Simulation provides best-in-class core engine performance for SystemVerilog, VHDL, mixed-signal, low power, and x-propagation. It supports both single-core and multi-core simulation, incremental and parallel build, and save/restart with dynamic test reload."[2]

During the development of synthesis strategies, xCelium was used for two reasons:

- To simulate the post-synthesis netlist;
- To generate a complete fsdb file with all the information about the switching activity of the netlist under a specific test.

The simulation environment and the tests were developed by a team of front-end designers, and after a short review they were taken for granted. The only

modification made to the test is the elimination of the power supply pins, since a before-layout simulation was carried out.
At the end of the simulation, an fsdb file is dumped thanks to Verdi3 of Synopsys. This file will be used by PrimeTime and PrimePower for power modeling.

4.1.2 Synopsis PrimePower

"PrimePower RTL power estimation leverages the Predictive Engine from Synopsys' RTL Architect™ product to provide RTL designers with fast, scalable, and accurate power estimation for early analysis of RTL blocks, subsystems, and full-SoCs."[14]

This tool, together with its predecessor xCelium, was used in the evaluation process of metrics and power modelling.
This tool takes the netlist, cell libraries, constraints, parasitic and switching activity as input and produces a power analysis with leakage and dynamic power.
The analysis was carried out in 0 delay mode. Then each delay was reset. As mentioned, the FSDB file containing the switching activity is received by the synthesis with xCelium. The file is then read thanks to the command:

```
read_fsdb -rtl -zero_delay -strip_path "tb/uut_top" $fsdb_file
```

The netlist post-synthesis is generated by the design compiler. It is read and linked with the command:

```
read_verilog $verilog_file
current_design $top_module
link
```

The parasitic from post-synthesis are read has:

```
read_parasitics -format SPEF $parasitic_file
```

The power module are computed after the elaboration of all input by the command:

```
update_power
```

Power model gives the estimation of the total power, cell leakage, net switching power and switching clock network power.

4.1.3 Synopsis Design Compiler NXT

"Design Compiler® NXT is the latest innovation in the Design Compiler family of RTL Synthesis products, extending the market-leading synthesis position of Design Compiler Graphical. Design Compiler NXT technology innovations include fast, highly efficient optimization engines, cloud-ready distributed synthesis, a new, highly accurate approach to RC estimation and capabilities required for the process nodes 5nm and below."[13]

Design Compiler is the main EDA tool used in this project. It is used for RTL to gate-level synthesis. For design and clock tree optimisation. For area optimisation using icc2 (an area optimisation tool). And to evaluate the main metrics.
The type of synthesis performed is called 'topological'. This is because in addition to optimising the netlist, it takes a floorplan file (DEF) as input and uses it during synthesis.
The type of synthesis is two-pass, so an initial synthesis process is performed to generate a correct floorplan from the given one. A second synthesis process is performed from the produced floorplan.
The evaluation of the post-synthesis netlist can be done by design compilers using the numerous reports.

```
check_timing
```

It is a command used for checking timing violations.

```
report_qor
```

Creates a report containing the QoR. It also reports all violations in timing that produce a negative slack. With a summary of area, timing and execution time.

```
report_clock_gating -structure
```

It creates a clock gating report containing the number of cells used, the number of gated registers and the total number of registers. The -structure option is used to create a report on all clock gating cells from the clock trees.
Power analysis reports can be obtained from the synthesis process.
In this thesis work, all power analyses will be performed by an external tool, Prime Power. This choice will be justified by the reliability data in the conclusions.

4.1.4 Synopsis Formality

"Formality® is an equivalence-checking (EC) solution that uses formal, static techniques to determine if two versions of a design are functionally equivalent."[5]

Formality is an EDA tool used for the logical equivalence check.
Synthesis techniques are studied in this project. The synthesis techniques start from an RTL netlist and arrive at a post-synthesis netlist. Formality is used to check that these two netlists are logically equivalent.
The formality script is a standard script for all designs. The synthesis process can however lead to a gate mapping problem. The problem is due to a name change

during synthesis.
The first process to be performed before performing a LEC is to check the name match. This is because the name is the first and simplest formality mapping method. Actually, design compiler produces a svf file with all the name variations. These variations, however, are not sufficient for the logic equivalence check process.
Another problem to be solved encountered during LEC is abort points. These are points that are too complex to be compared. The problem is solved by introducing dummy modules in the middle of the hierarchy, these introduce new support points.

4.2 The used libraries

The syntheses implemented during this thesis work tend to be developed using two classes of cell libraries. Each class of cell libraries contains several libraries with different strengths, different tracks, and different versions of the same implementation. The two classes are identified by the different threshold voltage and are : Standard Threshold Voltage (SVT) and Low Threshold Voltage (LVT).

4.2.1 Standard Threshold Voltage cell library

The standard threshold voltage class consists of six libraries:

- The two main libraries are two versions (optimised and non-optimised) of a low-strength library with a medium track.
- A library with higher strength than the previous ones.
- A library with a very high track.
- One library of latches for synchronisation and one of boundary cells for isolation.

SVT cells are characterised by lower speed and lower leakage. Them are used a lot in modules with higher slack, such as the peripheral core. They are the most presentable cells of the design and also contain the cells used in clock gating.
The synthesis that is performed is topographic, thus introducing physical elements, such as track. The two cells with different tracks are introduced to have different options for routability.

4.2.2 Low Threshold Voltage cell library

LVT cells are introduced during this thesis work to compensate for the use of clock gating structures with more stages. They are in fact cells with less toggling delay.

Consequently, they have greater leakage.
They are characterised by low delay but high leakage current.
They are used extensively in logic cores, where they solve stringent timing constraints.

4.3 Proposed Synthesis and Optimization strategies

The design on which the synthesis strategies will be developed is a proprietary ST Microelectronics design. It is a mixed-signal integrated circuit, so it consists of an analog top unit and a digital top unit.
The analog unit is not part of this work and is not inherent to the application of RTL to logic level synthesis strategies.
The digital unit (Figure 4.1) under-synthesis has about 7000 kilogates and runs with a clock frequencies up to 288 MHz. It needs to be implemented in a 40 nm CMOS technology, which is introduced in section 4.2. It is composed of a microprocessor core, a peripheral core able to drive the communication for the analog part, a memory, and an arithmetic logic subsystem. The memory is divided into 8 DRAM cells and some ROMs. Each type of memory has its own controller. The arithmetic logic subsystem contains a cache subsystem and uses the main clock from the top module at the maximum frequency.
The design was illustrated to give a physical reference to the modules that will be discussed in future sections and to which the theoretical strategies developed will be applied.

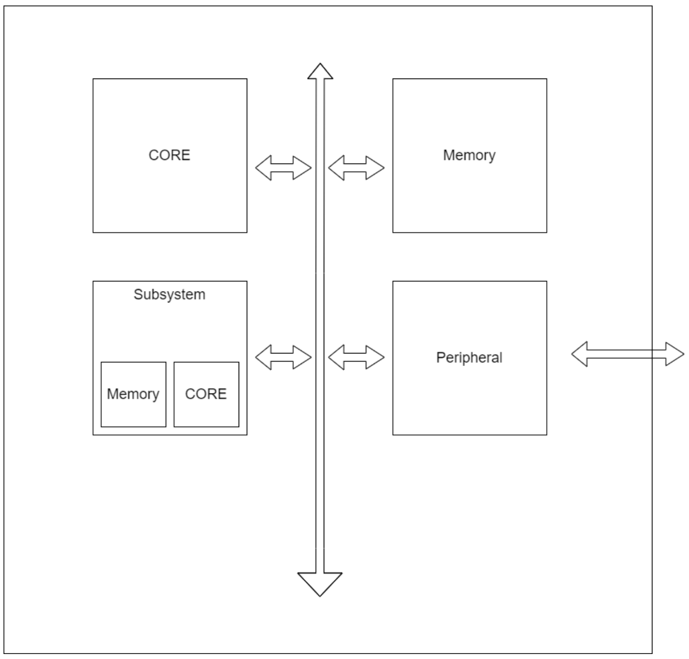

Figure 4.1: A sketch about the top digital module of the design.

The Implementation section starts with a description of the basic flow taken as "STANDARD" from an ST Microelectronics EDA Team. The basic flow is a two-level synthesis flow with a particular iteration composed by a compile and an incremental compile. It has a clock gating strategy (Figure 4.2) structured as 2 stages of elements, with a maximum fanout of 52 nets for the clock gating elements and a minimum bitwidth of the registers of 8 bits.
Basic synthesis is implemented in a 40 nm cmos technology, using an high track Standard Threshold Voltage cell libraries (Section 4.2). Once the basic flow has been described, the various implementations will be described.
Each implementation will be described in relation with respect to the basic synthesis flow with an explanation of the meaning of each change, following the approach described in chapter 3.1.

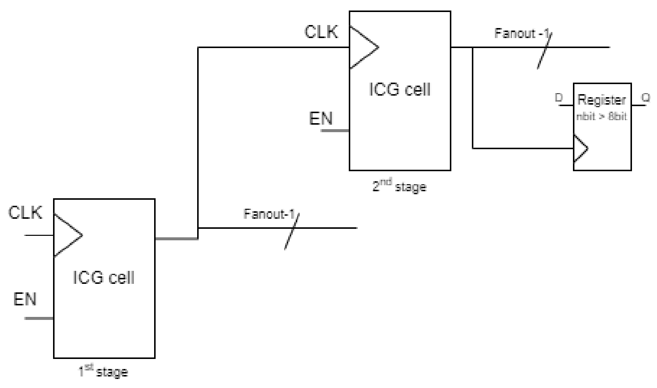

Figure 4.2: The clock gating structure is applied to the basic flow.

Basic flow

The basic synthesis flow starts by reading the source information. Source information is composed of the various RTL files described in an HDL language, the library cell with behavioural and physical parameters, and a plain floorplan. This information is then analysed for possible syntax errors.
The design is then checked for consistency and to see if it is synthesised. After checking the RTL, there is the link stage. In the link stage, the cells of the cell libraries are scheduled in the RTL code. In this stage is produced the first netlist (post-elaboration netlist).
This is followed by two compile, the second of which is incremental.
The first compile has to cope with functional constraints and introduces the chosen clock gating rules.
The second compile, the first incremental, adds the clock gating cloning technique, a constraint on the position of the core, and a limitation on the optimization of the gate between the boundary of the modules.

The first command introduced is used to define the cell libraries to be used in the synthesis. As a result, the RTL code of the project is read and analysed, and the top module is defined.
A command is also introduced to provide formal verification with hierarchies via the SVF declaration file.

```
define_design_lib LS -path work/LS
analyze -library common_LS_lib  -format vhdl $LS_lib_path
elaborate top -library top
set_verification_top
```

As introduced earlier, the RTL code and libraries are then synthesised into a first netlist by the link command, which ends the processing phase.

```
current_design $DESIGN_NAME
link
```

In the preparation phase for the first compile, various guide parameters are set for the EDA tool.
These three commands are used to indicate an optimisation mode according to the "power consumption" metric.
Low power placement instructs the Design Compiler to reduce the length of wires with a high switching activity, thus already using the behaviour produced during the RTL simulation to predict which nets will have a higher switching activity.
Dynamic optimisation is a wrapper to enable all forms of optimisation on the power metric. The different techniques can be disabled specifically by other environment variables.

```
set_app_var compile_enable_total_power_optimization true
set_app_var power_low_power_placement true
set_dynamic_optimization true
```

As defined above, the design must be checked for consistency.

```
check_design -summary
```

These commands are used to define the clock gating technique. Once the clock gating technique is synthesised, the clock gating structure is added to that introduced in RTL.
The cells used in clock gating are then defined, clock gating enabled or not, and then the structural parameters. The structural parameters are:

- The maximum fanout of newly instantiated clock gating cells (does not affect those instantiated in RTL).
- The maximum number of stages on a single path.
- The minimum number of bits a register must contain in order to be gated.

This last parameter is fundamental because it can happen that the design compiler does not reduce the number of bits of a register to allow it to be gated, but this leads to consistency errors in formal verification.

```
set ENABLE_CLOCK_GATING                                 "TRUE"
; # TRUE|FALSE
set ENABLE_SELF_GATING                                  "FALSE"
; # TRUE|FALSE
set CLOCK_GATING_CELLS_PATTERNS                         "*/*CGCELL*" ;
# Pattern to detect clock gating cells available
# (it works with get_lib_cell command)
```

```
set CLOCK_GATING_PREFERRED_CELL                      ""
; # Preferred ICG cell <Lib/CellName>
set CLOCK_GATING_MAX_FANOUT                          "52"
; # integer
set CLOCK_GATING_NUM_STAGES                          "2"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                        "8"
; # integer
```

Compile Ultra is used for high-effort mapping and implementation. It also forms the first clock gating structure.

```
compile_ultra -gate_clock -spg
```

High-effort area optimisation has been introduced in the latest versions of design compile NXT. It is used to perform monotonous gate-to-gate optimisation on mapped designs, thus improving area without degrading timing capability.

```
set_app_var optimize_area_ignore_path_group_weights true
optimize_netlist -area
```

Enabling boundary optimization gives the design compiler the possibility to duplicate clock gating elements among different modules. As will be introduced, the boundary optimization is enabled only for clock gating elements.

```
set_boundary_optimization [get_cells -hier -filter /
"is_hierarchical==true"] true
```

Incremental compile prevents the design from being mapped again. Both incremental compile and compile could be iterated more than once during the synthesis, taking into consideration the differences. The No-autoungrup option is used to allow clock gating cloning settings to remain enabled.

```
compile_ultra -incr -no_autoungroup -no_seq_output_inversion -spg
-scan
```

Area optimization is performed after each compile or compile incremental.

```
set_app_var optimize_area_ignore_path_group_weights true
optimize_netlist -area
```

This is the point at which the synthesis flow is iterated in the strategies we will see later. The iteration is used to settle with additional operations the violations obtained.
At the end of the flow, all reports useful for debugging and studying the design are produced. The three main metrics are measured, i.e., area, timing, and power, which will be used to classify the synthesis.

Clock Gating disable

The synthesis without clock gating produces a post-synthesis design that does not contain the clock gating logic along the clock tree. The clock tree is analysed to evaluate the distribution of the buffer. Buffers are used within the clock tree to balance the distribution of the clock to the registers,and they are composed of an inverter logic gate that introduces a capacitor and, consequently, a delay.
Data about the distribution of the clock tree is used to compute the possible influence of clock gating parameters, such as number of stages and fanout. Multi-stage clock gating techniques introduce more logic, and therefore more timing delay. Buffers can be changed to clock gating logic and multi-stages could be used in places in the tree where more buffers are used.
In Design Compiler, clock gating management is simple. It is not embedded in the compile command but can be obtained with the option -gate_clock of the compile command.
In the script developed and described for this work, clock gating could be disabled with a variable in the setup file. This variable is going to be used as an enable for an if statement.

```
set ENABLE_CLOCK_GATING                                    "FALSE"
; # TRUE|FALSE
```

As introduced before, the variable disable clock gating with an if-statement.

```
if {$ENABLE_CLOCK_GATING==TRUE} {
   set compile_cmd "$compile_cmd -gate_clock"
}
```

Clock Gating without Cloning

Cloning is an experimental technique. It uses the theory of interconnections. Wire could be modelled as a capacitor and resistor, consequently introducing a delta positive variable in delay. The gating logic could be duplicated to reduce the wire length (Section 2.7.2).
Cloning is disabled by prevent the design compiler from doing boundary optimization with the clock-gating elements.

```
compile_ultra -incr -no_seq_output_inversion -spg  -scan
```

For this strategy, the boundary optimization instruction is commented.

```
# set_boundary_optimization [get_cells -hier -filter /
"is_hierarchical==true"] true
```

Clock Gating with high/low Fan-out

Disabling clock-gating and cloning are two strategic synthesis methods for achieving the fundamental result of the main metrics, namely timing power and area.
Higher and lower fanout on clock gating give information on the impact of different clock gating structures on the design.
To calculate the best value for fanout, two aspects must be taken into account:

- RTL design structure and behavior provide information about potential registers that will toggle at the same time.
- Analysis of the post-synthesis result of synthesis done before give the distribution of the clock gating elements.

Clock gating parameters are applied by setting three variables.
The first two syntheses presented were carried out following principles; Memories, registers, and logic are implemented following a logic formed by a number of bits that is a power of 2. Values of 32 and 64 are therefore chosen because they are likely to indicate the size of the memories or logic pipelines.
Clock gating parameters application:

```
set CLOCK_GATING_MAX_FANOUT                              "64"
; # integer
set CLOCK_GATING_NUM_STAGES                              "2"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                            "8"
; # integer

set CLOCK_GATING_MAX_FANOUT                              "32"
; # integer
set CLOCK_GATING_NUM_STAGES                              "2"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                            "8"
; # integer
```

Other syntheses are carried out trying to understand the evolution of the design with the use of more clock gating elements in the same path, which drive fewer registers and therefore have less strength and less delay. This is reflected in the types of cells instantiated but leads to high timing violations.
The results of this synthesis, however, give an idea of some modules that can withstand this type of clock-gating structure, which decreases dynamic power by a lot.
From these assumptions, the flow structure of the present synthesis is studied.
Clock gating parameters application:

```
set CLOCK_GATING_MAX_FANOUT                               "22"
; # integer
set CLOCK_GATING_NUM_STAGES                               "3"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                             "4"
; # integer
```

Clock Gating with different Bitwidth

The second parameter is bitwidth. It is the minimum number of bits that can be used to instantiate a clock gating cell.

A first implementation is used to try if some registers are not gated because they contain less than 8 bits.

```
set CLOCK_GATING_MAX_FANOUT                               "32"
; # integer
set CLOCK_GATING_NUM_STAGES                               "1"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                             "4"
; # integer
```

The two following implementations of clock gating are intensive. These implementations are used to compute the higher percentage of registers gated.

```
set CLOCK_GATING_MAX_FANOUT                               "16"
; # integer
set CLOCK_GATING_NUM_STAGES                               "5"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                             "4"
; # integer
```

```
set CLOCK_GATING_MAX_FANOUT                               "16"
; # integer
set CLOCK_GATING_NUM_STAGES                               "5"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                             "8"
; # integer
```

Clock Gating with high Stages

In a large group of registers, there may be times when everyone does not toggle at the same time and times when only some subgroups switch while others do not. The establishment of multiple stages makes it possible to prevent register toggling

in the case of both groups and subgroups. This allows us to decrease power activity at higher levels.
As a result, the logic in the various paths grows, as do the delays. More delays reduce slack and lead to timing violations.
In the specific case study, it can be seen that a clock gating level is pre-established in RTL, consisting of over 200 elements and driving 91% of the registers. Thus, establishing a single-stage clock gating structure allows the structure established during synthesis to work with only 9% of the registers.
At the same time, the slack values are almost zero, and the establishment of multiple stages leads to variations in some modules.
A few attempts at multi-stage techniques have been made, but the timing violations are too high to take the direction of multi-stage among all the designs.
Consequently, it was decided to iterate on the design by varying the clock gating in only those modules that have a positive slack, allowing clock gating elements to be inserted in the path.

The multi-stage syntheses presented are of two types: very high stage number and medium stage number. Two syntheses are set up to see the variation of the number of cells per type and the variation of slacks in the different critical paths.
It is also important to note how the curve of gated registers varies as the number of clock gating elements increases, and when increasing them gives a considerable improvement.

```
set CLOCK_GATING_MAX_FANOUT                         "16"
; # integer
set CLOCK_GATING_NUM_STAGES                         "5"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                       "8"
; # integer
```

Three-stage clock gating synthesis is done to find a module where the systems could afford more logic in the path without timing violations.

```
set CLOCK_GATING_MAX_FANOUT                         "32"
; # integer
set CLOCK_GATING_NUM_STAGES                         "3"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                       "8"
; # integer

set_clock_gating_style \
    -minimum_bitwidth    ${CLOCK_GATING_MIN_BITWIDTH} \
    -positive_edge_logic "integrated" \
    -negative_edge_logic "integrated" \
    -control_point       before \
    -control_signal      scan_enable \
    -max_fanout          ${CLOCK_GATING_MAX_FANOUT} \
    -num_stages          ${CLOCK_GATING_NUM_STAGES}
```

Clock Gating without Sharing

No sharing is an option of the clock gating structure that causes an ICG cell to be set up for each bank of registers.

```
set_clock_gating_style -no_sharing
```

Application of low threshold voltage cells

The various libraries were introduced in Section 2. In this section, we introduce the method of application in EDA tools as Design Compiler.
The libraries introduced are of the low threshold voltage and standard threshold voltage types. They are inserted at the setup phase (Elaboration) with the standard threshold voltage at the beginning of the synthesis.

```
set REFERENCE_LIBRARIES {. CMOS040_CORE_LL}
```

A further synthesis using a different clock gating structure was tried. This test is done as a consequence of a quiet increase in leakage to reduce total power.
In this synthesis, LVT cells, which perform better than SVT cells and have more leakage, and a low-fanout clock gating structure are used.

```
set CLOCK_GATING_MAX_FANOUT                        "32"
; # integer
set CLOCK_GATING_NUM_STAGES                        "2"
; # integer
set CLOCK_GATING_MIN_BITWIDTH                      "4"
; # integer
```

Changing Libraries after first optimization

Using different library cells in advanced synthesis iterations allows design compilers to remap the design from a different structure and, if no optimisation is possible, return to the previous structure. This type of flow can be achieved by adding all libraries during setup with the following command:

```
set REFERENCE_LIBRARIES {. CMOS040_CORE_LL}
```

To then disable them by setting up a subset to be used in certain modules or in the entire design. In our case, we will apply it to the entire design with the option "all".

```
set_target_library_subset "CMOS040_CORE4_LS CMOS040_CORE0_LS" -all
```

All libraries are entered into the setup and then a subset of libraries used during the first stages of synthesis is created. This subset can then be deleted in subsequent iterations, and so introduced the low threshold voltage cell libraries.

```
remove_target_library_subset
```

5 Results

This chapter presents the results obtained by evaluating the different strategies proposed in chapter 4. The presentation of these results is divided into four categories:

- Clock gating (5.1), a category containing results of clock gating strategies.
- Cell libraries (5.2), category containing the results for the different cell library strategies.
- Synthesis flow (5.3), a category containing the results of the synthesis flow iteration.

The results are presented using tables in which each row contains a different strategy and each column contains the main metrics computed.
Depending on the details to be highlighted, the results are presented in different tables:

- Tables for setup and hold timing violations present possible problems of synthesis and optimization strategy. They are composed of the number of violations, the worst negative slack (WNS), and the total negative slack (TNS).
- A clock gating table is used to describe the clock gating structure. It is composed of the number of clock gating elements, the number and percentage of registers gated, and the number of buffers within the circuit. It is shown only in clock gating strategies.
- The power consumption table is divided into cell leakage, switching power, and total power.
- The area table contains the total area and the number of cells.
- The number of cells for each cell library is presented in a specific table. Different cells have different physical parameters. This table is used to find the use of each cell library.

Several attempts were made and are presented together for each strategy. In each iteration of the synthesis flow, specific techniques are applied, such as the one presented in section: 5.2 and 5.1
An analysis of the results obtained concludes the chapter.

5.1 Clock Gating

This section presents the results of different types of clock gating applications. The clock gating theoretical utility and development are introduced in section: 2.7.2. The implementations of the clock gating strategies are described in section: 4.3. The results are presented in different metric groups: timing, power, area, and clock-gating cell.
There are two timing tables: the first timing table contains the setup violations with information about the worst negative slack (WNS) and the total negative slack (TNS). The second contains the hold violations and has the same structure as the first.
The power table shows the results of the power model of the design. It contains the switching power, the cell leakage and the total power.
The last two tables contain: the total amount of area and the number of cells. The number of clock gating cells and the number of gated registers.

5.1.1 Clock Gating disable

Timing violations (Table 5.1) remained absent as the absence of clock gating reduces the number of logic elements before registers.

Strategy	**Number of Setup violations**	**WNS**	**TNS**
Basic	**0**	**0**	**0**
Clock gating disabled	**0**	**0**	**0**

Table 5.1: Setup violation for no clock-gating flow.

Strategy	**Number of Hold violations**	**WNS**	**TNS**
Basic	**263**	**-0.912**	**-15.333**
Clock gating disabled	**249**	**-0.179**	**-7.014**

Table 5.2: Hold violation for no clock-gating flow.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	**7165**	**81359 (94.42%)**	**82970**
Clock gating disabled	**169**	**78845 (91.61%)**	**96289**

Table 5.3: Clock gating report for no clock-gating flow.

The number of logic elements for clock gating decreases drastically in a synthesis without clock gating (Table 5.3), only the clock-gating elements instantiated in RTL remain. Such elements are instantiated to drive a lot of registers, have a high fanout and higher strength. In this synthesis, the registers driven by at least one clock gating level are already in a large percentage (Table 5.3). This leads us to deduce that instantiating one stage clock gating during synthesis will increase the percentage of gated registers but will in no way decrease dynamic power by considerable value, as the unnecessary toggling of registers that is preserved is most of the time the same as synthesis without clock gating.
Otherwise, instantiating clock gating with multiple stages during synthesis will give the possibility to drive registers even in smaller groups than those driven by RTL elements alone (subgroups).

Strategy	Total Power	Cell Leakage	Switching Power
Basic	**0.0892**	**0.0392**	**0.0169**
Clock gating disabled	**0.1188**	**0.0384**	**0.0208**

Table 5.4: Power analysis for no clock-gating flow.

Strategy	Total Area	Number of cells
Basic	**6787044**	**566863**
Clock gating disabled	**6776108**	**578894**

Table 5.5: Area for no clock-gating flow.

In the synthesis without clock gating, there is a decrease in area (Table 5.5) due to the decrease in clock gating elements but an increase in the number of buffer cells

(Table 5.3). To balance the clock propagation delays, buffer cells are inserted by the design compiler in place of the ICG clock gating cells. These cells have a lower delay and area than the ICG cells and are inserted in large numbers. These new cells, together with the absence of clock gating, increase the dynamic power value (Table 5.4).

5.1.2 Clock Gating without Cloning

Cloning of the gating elements has to be done to reduce the length of the wires. Wires introduce capacitance and, consequently, delays. For this reason, the number of timing violations is greater than with cloning (Table 5.6).

Strategy	**Number of Setup violations**	**WNS**	**TNS**
Basic	**0**	**0**	**0**
Clock-gating cloning disabled	**56**	**-0.001**	**-0.008**

Table 5.6: Setup violation for no clock-gating cloning flow.

Strategy	**Number of Hold violations**	**WNS**	**TNS**
Basic	**263**	**-0.912**	**-15.333**
Clock-gating cloning disabled	**285**	**-1.023**	**-15.986**

Table 5.7: Hold violation for no clock-gating cloning flow.

Strategy	**Clock Gating Element**	**Gated Register**	**Buffer Cells**
Basic	**7165**	**81359 (94.42%)**	**82970**
Clock-gating cloning disabled	**7206**	**81359 (94.42%)**	**82875**

Table 5.8: Clock gating report for no clock-gating cloning flow.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	**0.0892**	**0.0392**	**0.0169**
Clock-gating cloning disabled	**0.0902**	**0.0394**	**0.0167**

Table 5.9: Power analysis for no clock-gating cloning flow.

Strategy	Total Area	Number of cells
Basic	**6787044**	**566863**
Clock-gating cloning disabled	**6778677**	**566959**

Table 5.10: Area for no clock-gating cloning flow.

The number of gated registers remains the same with or without cloning , but cloning increases the number of buffer cells (Table 5.8), because compiler design have to balance longer wires. With cloning, the number of clock gating elements decreases, because the possibility of cloning clock gating cells out of the module allows them fanout to be merged with others.
The number of gated registers is the same. In fact, the dynamic power is almost identical (Table 5.9).

5.1.3 Clock Gating with high Fan-out

A synthesis with a higher maximum fanout does not lead to a strong variation in the number of clock gating elements in the single path, but it does increase their delay and strength, as they have to drive more output nets. This difference leads to minor timing violations (Table 5.11).

Strategy	Number of Setup violations	WNS	TNS
Basic	**0**	**0**	**0**
Clock gating Fanout=64 Bitwidth=8 Stages=2	**143**	**-0.001**	**-0.049**

Table 5.11: Setup violation for high Fan-out strategies.

Strategy	Number of Hold violations	WNS	TNS
Basic	**263**	**-0.912**	**-15.333**
Clock gating Fanout=64 Bitwidth=8 Stages=2	**269**	**-0.989**	**-15.717**

Table 5.12: Hold violation for high Fan-out strategies.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	**7165**	**81359 (94.42%)**	**82970**
Clock gating Fanout=64 Bitwidth=8 Stages=2	**7068**	**81359 (94.42%)**	**82716**

Table 5.13: Clock gating report for high Fan-out strategies.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	**0.0892**	**0.0392**	**0.0169**
Clock gating Fanout=64 Bitwidth=8 Stages=2	**0.0891**	**0.0394**	**0.0169**

Table 5.14: Power analysis for high Fan-out strategies.

Strategy	Total Area	Number of cells
Basic	**6780205**	**566505**
Clock gating Fanout=64 Bitwidth=8 Stages=2	**6776108**	**578894**

Table 5.15: Area for high Fan-out strategies.

The number of gated registers is identical to the base synthesis, but with fewer clock gating elements (Table 5.13), because a single element can drive more registers than the base synthesis elements. This results in a decrease in number of cells and area. The dynamic power is correspondingly equal, the leakage decreases somewhat due to the smaller number of cells (Table 5.14).

5.1.4 Clock Gating with low Fan-out

A synthesis with a lower maximum fanout does not lead to a strong variation in the number of clock gating elements in the single path but decreases their delay and strength. Furthermore, more elements are added in a new path where no clock gating element was present before, introducing some minor timing violations (Table 5.16).
More elements bring the synthesis to gate more registers (Table 5.18), and the dynamic power decrease. On the other hand, the leakage increase because of the higher number of cells (Table 5.19).

Strategy	**Number of Setup violations**	**WNS**	**TNS**
Basic	**0**	**0**	**0**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**50**	**-0.001**	**-0.006**
Clock gating Fanout=22 Bitwidth=4 Stages=3	**505**	**-11.485**	**-47.338**

Table 5.16: Setup violation for low Fan-out strategies.

Strategy	**Number of Hold violations**	**WNS**	**TNS**
Basic	**263**	**-0.912**	**-15.333**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**273**	**-0.897**	**-15.534**
Clock gating Fanout=22 Bitwidth=4 Stages=3	**65**	**-0.888**	**-8.03**

Table 5.17: Hold violation for low Fan-out strategies.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	**7165**	**81359 (94.42%)**	**82970**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**7913**	**81361 (94.42%)**	**82353**
Clock gating Fanout=22 Bitwidth=4 Stages=3	**8520**	**83224 (96.59%)**	**77333**

Table 5.18: Clock gating report for low Fan-out strategies.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	**0.0892**	**0.0392**	**0.0169**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**0.0974**	**0.0393**	**0.0169**
Clock gating Fanout=22 Bitwidth=4 Stages=3	**0.0674**	**0.0381**	**0.0073**

Table 5.19: Power analysis for low Fan-out strategies.

Strategy	Total Area	Number of cells
Basic	**6787044**	**566863**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**6787657**	**566324**
Clock gating Fanout=22 Bitwidth=4 Stages=3	**6816442**	**527421**

Table 5.20: Area for low Fan-out strategies.

5.1.5 Clock Gating with more Stages

Syntheses with more maximum stages lead to a structure with more gating elements in a single path. More gating elements in a single path means more delays and more timing violations (Table 5.21).

Strategy	**Number of Setup violations**	**WNS**	**TNS**
Basic	**0**	**0**	**0**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**50**	**-0.001**	**-0.006**
Clock gating Fanout=32 Bitwidth=8 Stages=3	**43**	**-0.006**	**-0.01**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**280**	**-0.025**	**-0.448**

Table 5.21: Setup violation for more stages strategies.

Strategy	**Number of Hold violations**	**WNS**	**TNS**
Basic	**263**	**-0.912**	**-15.333**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**273**	**-0.897**	**-15.534**
Clock gating Fanout=32 Bitwidth=8 Stages=3	**276**	**-1.11**	**-14.558**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**261**	**-1.243**	**-13.006**

Table 5.22: Hold violation for more stages strategies.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	7165	81359 (94.42%)	82970
Clock gating Fanout=32 Bitwidth=8 Stages=2	7913	81361 (94.42%)	82353
Clock gating Fanout=32 Bitwidth=8 Stages=3	9007	80868 (93.86%)	80239
Clock gating Fanout=16 Bitwidth=8 Stages=5	10506	80253 (93.15%)	79776

Table 5.23: Clock gating report for more stages strategies.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	0.0892	0.0392	0.0169
Clock gating Fanout=32 Bitwidth=8 Stages=2	0.0974	0.0393	0.0169
Clock gating Fanout=32 Bitwidth=8 Stages=3	0.0803	0.0392	0.0148
Clock gating Fanout=16 Bitwidth=8 Stages=5	0.0786	0.0389	0.0140

Table 5.24: Power analysis for more stages strategies.

Strategy	Total Area	Number of cells
Basic	**6787044**	**566863**
Clock gating Fanout=32 Bitwidth=8 Stages=2	**6787657**	**566324**
Clock gating Fanout=32 Bitwidth=8 Stages=3	**6782361**	**560497**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**6771683**	**559616**

Table 5.25: Area for more stages strategies.

More clock gating stages contain more clock gating elements, preventing register toggling at different times. Consequently, area increase and dynamic power strongly decrease (Table 5.24).

5.1.6 Clock Gating with different Bitwidth

The clock gating structure is characterised by three main parameters which describe the width and depth of the gating element tree. One of these is the bitwidth which is described in section 2.7.2.

Strategy	Number of Setup violations	WNS	TNS
Basic	**0**	**0**	**0**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**280**	**-0.025**	**-0.448**
Clock gating Fanout=16 Bitwidth=4 Stages=5	**600**	**-11.424**	**-47.114**
Clock gating Fanout=32 Bitwidth=4 Stages=1	**422**	**-0.115**	**-1.125**

Table 5.26: Setup violation for different Bitwidth exploration.

Strategy	Number of Hold violations	WNS	TNS
Basic	**263**	**-0.912**	**-15.333**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**261**	**-1.243**	**-13.006**
Clock gating Fanout=16 Bitwidth=4 Stages=5	**63**	**-0.958**	**-10.802**
Clock gating Fanout=32 Bitwidth=4 Stages=1	**254**	**-0.979**	**-18.004**

Table 5.27: Hold violation for different Bitwidth exploration.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	**7165**	**81359 (94.42%)**	**82970**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**10506**	**80253 (93.15%)**	**79776**
Clock gating Fanout=16 Bitwidth=4 Stages=5	**9874**	**83115 (96.47%)**	**76573**
Clock gating Fanout=32 Bitwidth=4 Stages=1	**4840**	**84323 (97.94%)**	**96747**

Table 5.28: Clock gating report for different Bitwidth exploration.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	**0.0892**	**0.0392**	**0.0169**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**0.0786**	**0.0389**	**0.0140**
Clock gating Fanout=16 Bitwidth=4 Stages=5	**0.0746**	**0.0377**	**8.21E-03**
Clock gating Fanout=32 Bitwidth=4 Stages=1	**0.1114**	**0.0392**	**0.0253**

Table 5.29: Power analysis for different Bitwidth exploration.

Strategy	Total Area	Number of cells
Basic	**6787044**	**566863**
Clock gating Fanout=16 Bitwidth=8 Stages=5	**6782361**	**579528**
Clock gating Fanout=16 Bitwidth=4 Stages=5	**6825521**	**527177**
Clock gating Fanout=32 Bitwidth=4 Stages=1	**6785498**	**579528**

Table 5.30: Area for different Bitwidth exploration.

Decreasing the bitwidth, i.e. the minimum size of the registers that are gated, increases the gating elements and area. This greatly increases the number of gated registers and decreases dynamic power.
Inserting logic elements in paths with very small registers, however, introduces large delays that are difficult for the EDA tool to resolve. So there will be big setup violations.

5.1.7 Clock Gating without Sharing

Sharing option in the clock gating structure prevents the use of a clock gating cell for more than one register. However, this has no effect for clock gating cells instantiated in RTL. It also renders the use of cloning useless.

Strategy	Number of Setup violations	WNS	TNS
Basic	**0**	**0**	**0**
Clock gating style no_sharing	**78**	**-0.001**	**-0.017**

Table 5.31: Setup violation for clock gating without sharing strategies.

Strategy	Number of Hold violations	WNS	TNS
Basic	**263**	**-0.912**	**-15.333**
Clock gating style no_sharing	**78**	**-0.794**	**-16.381**

Table 5.32: Hold violation for clock gating without sharing strategies.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	**7165**	81359 (94.42%)	**82970**
Clock gating style no_sharing	**5993**	81219 (94.36%)	**80583**

Table 5.33: Clock gating report for clock gating without sharing strategies.

Strategy	Total Area	Number of cells
Basic	**6787044**	**566863**
Clock gating style no_sharing	**6777837**	**566326**

Table 5.34: Area for clock gating without sharing strategies.

The result is that the gated cells are higher than in the synthesis without clock gating, and the two stage levels have become unnecessary. The area decreases with only one stage of clock gating and cells with less strength are used because the fanout is 1, no cell can drive more than one register.

5.2 Libraries

This section presents the results obtained by introducing additional types of cell libraries. The structure of the cell libaries was introduced in Section 3.1.1, while the various implementations were introduced in Section 4.3. The results are presented in different metric groups: timing, power, area, number of cell used for cell library and clock-gating cell.
There are two timing tables: the first timing table contains the setup violations with information about the worst negative slack (WNS) and the total negative slack (TNS). The second contains the hold violations and has the same structure as the first.
The power table shows the results of the power model of the design. It contains the switching power, the cell leakage and the total power.
The number of cells used for the various libraries is presented in a table. Three libraries are shown: Low threshold voltage libraries, standard threshold voltage libraries with low strength and standard threshold voltage libraries with high strength. Low threshold voltage cells are faster and introduce more leakage, high strength cells must be driven accordingly.
The last two tables contain: the total amount of area and the number of cells. The number of clock gating cells and the number of gated registers.

5.2.1 Low leakage cells

The introduction of LVT cells gives the tool the ability to resolve timing violations without the use of buffers. It also allows more registers to be gated as the delay is shorter and clock gating elements can be inserted in more paths. Leakage power, however, increases dramatically due to the use of low threshold voltage cells and consequently high leakage.

To cope with this drastic increase in power consumption, a higher clock gating structure with a lower fanout can be used. Lower fanout means the use of cells with lower strength and therefore less leakage.

Strategy	**Number of Setup violations**	**WNS**	**TNS**
Basic	**0**	**0**	**0**
Low leakage cells	**217**	**-0.1**	**-0.335**

Table 5.35: Setup violation for synthesis with LVT cells.

Strategy	**Number of Hold violations**	**WNS**	**TNS**
Basic	**263**	**-0.912**	**-15.333**
Low leakage cells	**217**	**-1.14**	**-39.346**

Table 5.36: Hold violation for synthesis with LVT cells.

Strategy	**Clock Gating Element**	**Gated Register**	**Buffer Cells**
Basic	**7165**	**81359 (94.42%)**	**82970**
Low leakage cells	**7212**	**81545 (94.66%)**	**78709**

Table 5.37: Clock gating report for synthesis with LVT cells.

Strategy	**Total Power**	**Cell Leakage**	**Switching Power**
Basic	**0.0892**	**0.0392**	**0.0169**
Low leakage cells	**0.1017**	**0.0496**	**0.0173**

Table 5.38: Power analysis for synthesis with LVT cells.

Strategy	**Number of SVD cells low strength**	**Number of SVD cells high strength**	**Number of LVT cells**
Basic	**115147**	**335346**	**0**
Low leakage cells	**19679**	**480291**	**28454**

Table 5.39: Number of cell for each cell library for synthesis with LVT cells.

Strategy	**Total Area**	**Number of cells**
Basic	**6787044**	**566863**
Low leakage cells	**6725040**	**552495**

Table 5.40: Area for synthesis with LVT cells.

5.2.2 Low leakage cells with low fan-out clock gating

By using less fanout, we therefore solve the problems of leackage by slightly increasing the area and timing violations.

Strategy	**Number of Setup violations**	**WNS**	**TNS**
Basic	**0**	**0**	**0**
Low leakage cells with low fan-out	**41**	**-0.02**	**-0.028**

Table 5.41: Setup violation for synthesis with LVT cells and low fonout clock gating structure.

Strategy	Number of Hold violations	WNS	TNS
Basic	**263**	**-0.912**	**-15.333**
Low leakage cells with low fan-out	**411**	**-0.966**	**-23.194**

Table 5.42: Hold violation for synthesis with LVT cells and low fonout clock gating structure.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	**7165**	**81359 (94.42%)**	**82970**
Low leakage cells with low fan-out	**10390**	**83529 (96.95%)**	**77346**

Table 5.43: Clock gating report for synthesis with LVT cells and low fonout clock gating structure.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	**0.0892**	**0.0392**	**0.0169**
Low leakage cells with low fan-out	**0.0786**	**0.0389**	**0.0140**

Table 5.44: Power analysis for synthesis with LVT cells and low fonout clock gating structure.

Strategy	**Number of SVD cells low strength**	**Number of SVD cells high strength**	**Number of LVT cells**
Basic	**115147**	**335346**	**0**
Low leakage cells	**19679**	**480291**	**25730**

Table 5.45: Number of cell for each cell library for synthesis with LVT cells.

Strategy	**Total Area**	**Number of cells**
Basic	**6787044**	**566863**
Low leakage cells with low fan-out	**6714460**	**529330**

Table 5.46: Area for synthesis with LVT cells and low fonout clock gating structure.

5.3 Custom Synthesis Flow

This section presents the results of using several different iterations in the synthesis flow. The theoretical basis of the synthesis flow is presented in section 2.9, while the tool used is presented in section 4.1.3. The implementation of the various flows is presented in section 4.3. The introduction of more iterations lengthens the computational time of synthesis and produces different results, which will be described as follows. The results are presented in different metric groups: timing, power, area, and clock-gating cell.
There are two timing tables: the first timing table contains the setup violations with information about the worst negative slack (WNS) and the total negative slack (TNS). The second contains the hold violations and has the same structure as the first.
The power table shows the results of the power model of the design. It contains the switching power, the cell leakage and the total power.
The number of cells used for the various libraries is presented in a table. Three libraries are shown: Low threshold voltage libraries, standard threshold voltage libraries with low strength and standard threshold voltage libraries with high strength.
The last two tables contain: the total amount of area and the number of cells. The number of clock gating cells and the number of gated registers.

5.3.1 Changing Clock Gating Strategies after first optimization

Changing the clock gating structure during the first incremental allows us to increase the number of gated registers and consequently decrease the switching power. Changing the structure on an already synthesised design means that there is no major deterioration, so we have less area because fewer buffers are instantiated.

Strategy	**Number of Setup violations**	**WNS**	**TNS**
Basic	**0**	**0**	**0**
Different clock gating after first compile	**173**	**-0.069**	**-0.268**

Table 5.47: Setup violation for synthesis with different clock gating structure after first compile.

Strategy	**Number of Hold violations**	**WNS**	**TNS**
Basic	**263**	**-0.912**	**-15.333**
Different clock gating after first compile	**256**	**-0.993**	**-18.23**

Table 5.48: Hold violation for synthesis with different clock gating structure after first compile.

Strategy	**Clock Gating Element**	**Gated Register**	**Buffer Cells**
Basic	**7165**	**81359 (94.42%)**	**82970**
Different clock gating after first compile	**8861**	**81223 (95.09%)**	**81746**

Table 5.49: Clock gating report for synthesis with different clock gating structure after first compile.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	0.0892	0.0392	0.0169
Different clock gating after first compile	0.0822	0.0379	0.0123

Table 5.50: Power analysis for synthesis with different clock gating structure after first compile.

Strategy	Total Area	Number of cells
Basic	6787044	566863
Different clock gating after first compile	6761238	563865

Table 5.51: Area for synthesis with different clock gating structure after first compile.

5.3.2 Changing Libraries after first optimization

Starting from a post-synthesis design, using cell libraries containing cells with less delay gives the tool the tools to increase slack and use this increased force in timing to reduce area. By introducing them in an incremental compile, they are only used in modules where they are needed without increasing the slack too much, thereby increasing QoR.

Strategy	Number of Setup violations	WNS	TNS
Basic	0	0	0
Different cell libraries after first compile	109	-0.056	-0.113

Table 5.52: Setup violation for synthesis with introduction of LVD cells after first compile.

Strategy	Number of Hold violations	WNS	TNS
Basic	**263**	**-0.912**	**-15.333**
Different cell libraries after first compile	**433**	**-1.174**	**-32.74**

Table 5.53: Hold violation for synthesis with introduction of LVD cells after first compile.

Strategy	Clock Gating Element	Gated Register	Buffer Cells
Basic	**7165**	**81359 (94.42%)**	**82970**
Different cell libraries after first compile	**7182**	**81301 (94.36%)**	**79944**

Table 5.54: Clock gating report for synthesis with introduction of LVD cells after first compile.

Strategy	Total Power	Cell Leakage	Switching Power
Basic	**0.0892**	**0.0392**	**0.0169**
Different cell libraries after first compile	**0.0932**	**0.0413**	**0.0157**

Table 5.55: Power analysis for synthesis with introduction of LVD cells after first compile.

Strategy	Total Area	Number of cells
Basic	**6787044**	**566863**
Different cell libraries after first compile	**6733454**	**568527**

Table 5.56: Area for synthesis with introduction of LVD cells after first compile.

6 Conclusions

The aim of the thesis work is to propose and explore different possible approaches to synthesis. The thesis flow is distributed among some iterative stages with the unique objective of using previous results and analysis for future synthesis. Each synthesis has a different impact on the QoR of the produced netlist. In particular, chapter 3 illustrates the approach to the thesis flow and the various possible implementations, while chapter 4 explains the proposed flows applied to specific case studies. The specific flow is based on an EDA tool called Design Compiler NXT, but can be adapted to any other compiler or case.
Chapter 5 shows the obtained results divided by category. Each category consists of a presentation of the results of the various implementations accompanied by a specific analysis of the results. The last chapter is the conclusion of the thesis work, and contains a summary of the obtained results.

6.1 Clock gating

There are numerous results from the point of view of clock gating applications, many of them obtained in conjunction with the use of additional techniques.
The clock gating structures explored are of various types and produce a variety of results. The exploration of all these solutions allows us to understand and create new synthesis methodologies to better structure them according to the design under test.
Resolving violations resulting from inefficient synthesis methodology is a key objective in the synthesis and optimisation process, the exploration gives the methodology to resolve violations, which may be timing, over area and over power consumption. Specifically, next lines shows how variation in clock gating structure impacts these violations and see how the right structure can help resolve them.
The clock gating structure consists of two parts, one instantiated in RTL by the front-end designer, and one that is inserted during synthesis. The various strategies implemented aim to experiment with all possible structures, structures which are characterised by three parameters: fanout, stage and bitwidth.
Variation in the maximum number of fanout varies the clock gating structure, a larger fanout means more path in the clock gating structure, a smaller fanout means less path.

The low fanout structure is composed by:

- Low-strength driving cell. Each cell of the clock gating structure drives fewer registers and needs less driving strength, correspondingly less area and less leakage.
- Fewer gated registers. More registers without gating because of the smaller clock gating structure, greater switching activity.

The high fanout structure is composed of:

- Greater cell strength. High fanout needs high-strength cells, that have higher leakage and delay.
- A tree with greater breadth. More fanout produce a larger structure that can drive more registers, which produces a lower switching activity.

The basic structure consists of a stage introduced in RTL. A structure with more stages makes it possible to solve various power problems. On the other hand, more stages mean more clock gating elements in the single path, resulting in a larger area, more complex cells and greater timing violations. It can therefore be used in paths with positive slack with high switching activity.
Lowering the bitwidth value makes it possible to expand the set of registers that can be gated. However, gating such small registers saves switching power, but at the cost of additional logic, which introduces further power consumption. The savings in power consumption will be very low given the small size, but we will introduce further logic and thus further delay.
From the studies presented above, it can be seen that a correct clock gating structure allows us to produce a netlist with higher QoR. However, this have to be adapted to the structure of the design, these analyses give us the adaptation methodology to solve the various problems in the three main metrics: timing, area and power consumption.

6.2 Cell libraries

The netlist produced by the various synthesis may contain timing violations, such as setup. These violations can be resolved by changes in the path, increases of the die area, or the use of different cell types. Adding different types of cells gives the EDA tool more options for resolving these violations.
In this thesis work, three classes of library cells were introduced:

- Standard threshold voltage with high strength.
- Standard threshold voltage with low strength.

- Low threshold voltage, which are the library cells with lower delay and higher leakage power.

The basic implementation contains no LVT cells. The introduction of LVT cells makes it possible to increase the slack and consequently reduce the area, while increasing leakage. For this reason, the following syntheses are carried out with a higher clock gating structure in order to reduce power consumption and limit the use of high-performance cells.

6.3 Custom Synthesis Flow

The synthesis flow is an iterative flow, which can be executed in its parts several times. Changing the synthesis conditions at each iteration allows the EDA tool to find new, better-performing solutions or, if it cannot find any, to return to the previous solution. This assumption gives rise to syntheses with multiple incremental compiles that produce better results based on the number of iterations.
Changing the clock gating structure at the first incremental compile gives the tool the possibility of optimising the clock gating structure from one already formed, so as to optimise that structure in other modules, keeping the same one if it performs well or introducing a new one if it performs better.
Introducing cell libraries containing cells with lower delays in incremental compiles following the first one, gives the EDA tool the possibility of optimising paths with lower slack and resolving possible timing violations.
These results initiate synthesis flows with more iterations for the production of post-synthesis netlist with higher QoR.

6.4 Future works

The dualism between the experimental, heuristic results and theory is the result of exploration that gives the user the capability of a conscious and efficient synthesis flow development. Many techniques are introduced in this work, but many others could be explored in the future works, that may be better tailored to specific synthesis tools, circuits, and technologies. The introduction of cell libraries containing higher-performance cells in specific modules in which small slacks are observed is an aspect to be studied for an adaptive synthesis process. Likewise, the introduction of specific clock gating structures for the various modules simplifies the optimisation work of the EDA tool. Incremental optimization steps are especially promising, but the number of useful iterations needs to be evaluated also considering the available processing time in today's competitive time-to-market schedules.

Bibliography

[1] AnySilicon. The ultimate guide to clock gating, Jul 2022.

[2] Cadence. Xcelium logic simulation user guide, 2022.

[3] Wun-Han Chen, Hsin-Hung Chang, Jui-Hung Hung, and Tsai-Ming Hsieh. Clock tree construction using gated clock cloning. In *2012 4th Asia Symposium on Quality Electronic Design (ASQED)*, pages 54–58, 2012.

[4] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* Electrical and Computer Engineering Series. McGraw-Hill, 1994.

[5] Synopsys Formality. Formality user guide, 2022.

[6] Hideo Fujiwara, Hiroyuki Iwata, Tomokazu Yoneda, and Chia-Yee Ooi. A non-scan design-for-testability for register-transfer level circuits to guarantee linear-depth time expansion models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27:pp. 1535–1544, 09 2008.

[7] Akihiro Hashimoto and James G. Stevens. Wire routing by optimizing channel assignment within large apertures. In *DAC*, 1971.

[8] J. Heller. *Catch-22: A Novel.* S & S classic edition. Simon & Schuster, 1999.

[9] IEEE. Ieee draft guide: Adoption of the project management institute (pmi) standard: A guide to the project management body of knowledge (pmbok guide)-2008 (4th edition). *IEEE P1490/D1, May 2011*, pages 1–505, 2011.

[10] Thomas M. McWilliams and Lawrence C. Widdoes. Scald: Structured computer-aided logic design. In *Proceedings of the 15th Design Automation Conference*, DAC '78, page 271–277. IEEE Press, 1978.

[11] Gordon E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*, 11(3):33–35, 2006.

[12] Vilfredo Pareto, G.-H. (Georges-Henri) Bousquet, and Giovanni Busino. *Cours d'economie politique / Vilfredo Pareto.* Droz, Geneve, nouvelle edition / par g.h. bousquet et g. busino. edition, 1964.

[13] Synopsys. Design compiler nxt user guide, 2022.

[14] Synopsys. Primepower user guide, 2022.

[15] Synopsys. What is equivalence checking? – how does it work?, 2022.

[16] Yiyin Wang and Rene Van leuken. *System Design Methodologies - High Level Synthesis And A VHDL Implementation of a Practical Scheme for UWB Communication*. 07 2008.