Instituto Politécnico Nacional
Centro de Investigación en Computación

Laboratorio de Microtecnología y Sistemas Embebidos

T E S I S

Clock Gating Implementation in a RISC-V Processor

Que para obtener el grado de:

Maestría en Ciencias en Ingeniería de Cómputo

Presenta:

Alexis Silva Heredia

Directores de Tesis:
Dr. Ricardo Barrón Fernández
Dr. Victor Hugo Ponce Ponce

Abril 2023

SIP-13
REP 2017

INSTITUTO POLITÉCNICO NACIONAL
SECRETARIA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REGISTRO DE TEMA DE TESIS
Y DESIGNACIÓN DE DIRECTOR DE TESIS

Ciudad de México, a 04 de noviembre de 2022

El Colegio de Profesores de Posgrado del **Centro de Investigación en Computación** en su Sesión

Ordinaria No. 06 Celebrada el día 29 del mes junio de 2020 conoció la solicitud presentada por el (la) alumno (a):

Apellido Paterno:	SILVA	Apellido Materno:	HEREDIA	Nombre (s):	ALEXIS

Número de registro: B 1 9 0 4 2 1

del Programa Académico de Posgrado: **Maestría en Ciencias en Ingeniería de Cómputo**

Referente al registro de su tema de tesis; acordando lo siguiente:

1.- Se designa al aspirante el tema de tesis titulado:

"Clock Gating Implementation in a RISC-V Processor"

Objetivo general del trabajo de tesis:

Evaluar el consumo de potencia en el procesador Lagarto utilizando la técnica de "Clock-Gating".

2.- Se designa como Director de Tesis a los profesores:

Director: **Dr. Ricardo Barrón Fernández** 2° Director: **Dr. Victor Hugo Ponce Ponce**

No aplica: ☐

3.- El Trabajo de investigación base para el desarrollo de la tesis será elaborado por el alumno en:

Centro de Investigación en Computación

que cuenta con los recursos e infraestructura necesarios.

4.- El interesado deberá asistir a los seminarios desarrollados en el área de adscripción del trabajo desde la fecha en que se suscribe la presente, hasta la aprobación de la versión completa de la tesis por parte de la Comisión Revisora correspondiente.

Director(a) de Tesis

Dr. Ricardo Barrón Fernández

2º Director de Tesis

Dr. Victor Hugo Ponce Ponce

Aspirante

C. Alexis Silva Heredia

Presidente del Colegio

Dr. Francisco Hiram Calvo Castro

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
DIRECCIÓN
IPN-CIC

SIP-14
REP 2017

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México siendo las 12:00 horas del día 17 del mes de junio del 2022 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Posgrado del: **Centro de Investigación en Computación** para examinar la tesis titulada: ***"Clock Gating Implementation in a RISC-V Processor"*** del (la) alumno (a):

Apellido Paterno:	SILVA	Apellido Materno:	HEREDIA	Nombre (s):	ALEXIS

Número de registro: B 1 9 0 4 2 1

Aspirante del Programa Académico de Posgrado: **Maestría en Ciencias en Ingeniería de Cómputo**

Una vez que se realizó un análisis de similitud de texto, utilizando el software antiplagio, se encontró que el trabajo de tesis tiene **24**% de similitud. **Se adjunta reporte de software utilizado**.

Después que esta Comisión revisó exhaustivamente el contenido, estructura, intención y ubicación de los textos de la tesis identificados como coincidentes con otros documentos, concluyó que en el presente trabajo **SI** [ ] **NO** [x] **SE CONSTITUYE UN POSIBLE PLAGIO.**

JUSTIFICACIÓN DE LA CONCLUSIÓN: *(Por ejemplo, el % de similitud se localiza en metodologías adecuadamente referidas a fuente original)*
Reporta el efecto y las mejoras que resultan de incorporar la técnica de "Clock-Gating" en un procesador RISC-V, usando la herramienta Cadence. El trabajo desarrollado por el alumno no refleja un posible plagio.

****Es responsabilidad del alumno como autor de la tesis la verificación antiplagio, y del Director o Directores de tesis el análisis del % de similitud para establecer el riesgo o la existencia de un posible plagio.**

Finalmente, y posterior a la lectura, revisión individual, así como el análisis e intercambio de opiniones, los miembros de la Comisión manifestaron **APROBAR** [x] **NO APROBAR** [ ] la tesis, en virtud de los motivos siguientes:

COMISIÓN REVISORA DE TESIS

Dr. Ricardo Barrón Fernández
Director de tesis

Dr. Luis Alfonso Villa Vargas

M. en C. Osvaldo Espinosa Sosa

Dr. Víctor Hugo Ponce Ponce
2º Director de tesis

Dr. Herón Molina Lozano

Dr. Ramírez Salinas Marco Antonio

Dr. Francisco Hiram Calvo Castro
PRESIDENTE DEL COLEGIO DE PROFESORES

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
DIRECCIÓN
IPN-CIC

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA DE AUTORIZACIÓN DE USO DE OBRA PARA DIFUSIÓN

En la Ciudad de México el día 3 del mes de Abril del año 2023, el que suscribe Alexis Silva Heredia alumno del programa de Maestría en Ciencias en Ingeniería de Cómputo con número de registro B190421, adscrito al Centro de Investigación en Computación, manifiesta que es autor intelectual del presente trabajo de tesis bajo la dirección del Dr. Ricardo Barrón Fernández y del Dr. Victor Hugo Ponce Ponce y cede los derechos del trabajo intitulado "Clock Gating Implementation in a RISC-V Processor", al Instituto Politécnico Nacional, para su difusión con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expresado del autor y/o directores. Este puede ser obtenido escribiendo a las siguientes direcciónes de correo. alexis.silva1293@gmail.com / vponce@cic.ipn.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente de este.

Alexis Silva Heredia

Resumen

El mundo ha entrado en una era donde tenemos computadoras en todas partes, tenemos computadoras en nuestros teléfonos, relojes, lavadoras, carros y mas. Esta tendencia parece seguir incrementandose por lo que la necesidad de tener computadoras con mejor rendimiento es una busqueda constante. Uno de los temas principales a trabajar en computadoras, son las temperaturas de los SoC y los mayores tiempos de duracion de la batería cuando se usan estos dispositivos. Existen muchas técnicas que se han empleado para lograr que la potencia disipada en un microprocesador sea tan baja como sea posible.
En esta tesis evaluamos los beneficios y costos de usar una técnica llamada clock gating cuando es implementada en un procesador basado en RISC-V.

Abstract

The world has entered into an era where we have computers everywhere, we have computers on our phones, watches, washing machines, cars, and more. As this trend seems to keep increasing, the need of having computers with better performances is always wanted. One of the main subjects to work in computers are the SoC temperatures and longer battery lifes when using devices as Laptops. There are several techniques worked out to achive a microprocessor where the power dissipated is as low as possible.
This thesis evaluates the benefits and costs of using a technique called clock gating when it is implemented on a RISC-V based processor.

Acknowledgements

I would like to acknowledge the guidance from Professor Victor Ponce. His unconditionally support from the day i met him until the end of this work, is something i will appreciate for the rest of my life. I would also thank Profs. Ricardo Barrón Fernández and Marco Antonio Ramírez Salinas, they gave me their trust to work in a group of very talented people.
Professor Frances Moll Echeto was a mentor to me during the development of this thesis, he shared to me some of his knowledge in order to face the different problems that arose when working on this thesis. I want to express all of my gratitude to you.
I want to thank all students and members from the MICROSE team. This work is completed because of all your support and mentorship. I have learned a lot of things from the entire team. Oswaldo Franco Garcia, thank you for all your support during the Master degree, you are an incredible friend. Carlos Rojas Morales, thank you for all of your knowledge that in a very kindly way you shared to me. My family and friends who give me their support during my master degree. I will always have a piece from you on my heart.
I want to also thank the CONACYT, without the financial support from the mexican government institution, take the challenge to study the master degree would have been impossible to overcome. It was a responsibility but also an honour to receive the support from the CONACYT and now i will always work to set the prestige from the CONACYT and the CIC as high as possible.

Contents

List of Figures

Chapter 1

Introduction

As the world moves on into a new technological revolution where computers are the top of indispensable things to accomplish this revolution, the countries should no longer have a full technology dependency.

The Microtechnology and Embedded Systems group from the Centro de Investigación en Computación del Instituto Politécnico Nacional is concerned with this situation and since 2013 is involved in the development of a computer architecture called Lagarto with the objective of generate an open computer platform in order to facilitate the comprehension of essential concepts in computer architecture and operating systems for the academy and research fields. [10].

The Lagarto platform is also useful to generate several projects which are pretended to enhance the same Lagarto architecture and at the same time is generating knowledge inside the MICROSE group. However a System on Chip (SoC) is a very complex system where the processor can be the main section of it but not the only one. Lagarto is also a project that allow us to collaborate with different institutions around the world in order to face bigger challenges.

One of these collaborations is the *Designing RISC-V based Accelarators for next generation Computers (DRAC)* project. DRAC is an international collaboration involving multiple institutions, including, *the Barcelona Super Computing Center (BSC), the Centro Nacional de Microelectrónica (CNM), Centro de Investigación en Computación del Instituto Politécnico Nacional (CIC-IPN) and Universitat Politécnica de Catalunya (UPC).* The initial

phase of this project consisted on the design, verification, implementation, and fabrication of a RISC-V general-purpose microprocessor capable of booting Linux. The initial RISC-V microprocessor and SoC implementation are the basis for the designs that are right now under development in the *DRAC* and, for this reason it is denoted as *preDRAC* SoC [11].

1.1 Motivation

Although a common set of hardware technologies is used in computers ranging from smart home appliances to cell phones to the largest supercomputers, these different applications have distinct design requirements and employ the core hardware technologies in different ways. Generally, computers are used in three different classes of applications.

- **Personal Computers (PCs)**

  The personal computers emphasize delivery of good performance to single users at low cost and usually execute third-party software.

- **Servers**

  Usually servers and supercomputers have the highest performance and cost;

- **Embedded computers**

  Embedded applications often have unique application requirements that combine a minimum performance with stringent limitations on cost or power

[9]

Thus depending on the requirements of the design or the class of the computer, we can always make trade offs between performance, power, and cost. For example, some markets as the Personal Computers always look for the best performance but also for the most efficient power consumption chip with the lower-medium cost. If we analyse this situation we can figure out why these requirements. Someone using a Laptop wants the Laptop to be as fast

as possible because of all the programs that may be required to run at the same time. Also, it is very likely that the one using this laptop would want the laptop to run on the battery as much as possible. Lastly this same person would require that the laptop does not get too hot during its use.

The dominant technology for integrated circuits is called CMOS (complementary metal oxide semiconductor). For CMOS, the primary source of energy consumption is called dynamic energy, that is, energy that is consumed when transistors switch states from 0 to 1 and vice versa. The dynamic energy depends on the capacitive loading of each transistor and the voltage applied:

$$Energy = Capacitive\ Load * Voltage^2 \tag{1.1}$$

[9]

This thesis is concerned about the necessity to achieve a grater power performance on a processor architecture in order to evaluate its impact on power, timing and area using a technique to reduce dynamic power.

1.2 Background

Computer world is going through a big change as never before in history as we are using computers everywhere. Today we have computers in our smartphones, laptops, watches, cars and a lot of more devices.
Since the invention of the transistor and some time later the formulation of Moore's law (which stated that the number of transistors that could be stacked, while remaining profitable, on a microchip it would double every two years), transistors have reduced its size so much that today we have processors built with technologies up to 5 nano-meters. However, this increased number of transistors on a microchip has brought a diverse amount of problems and limitations. Currently the law of Moore has even gone unfulfilled due to these issues. A significant problem arose: eventually, more transistors were placed in the same area; consequently, high temperature issues began to emerge due to power dissipation. One of these methods was called "Clock Gating".

1.3 Justification

As stated earlier, a lot of devices as cars, washing machines, smartphones, watches, and more rely on computer chips. This trend only seems to augment because a new revolution based on Artificial Intelligence (AI) and the Internet of Things (IoT). Some experts even predict AI will have a more profound impact on humanity than fire, the electricity, and the internet. It is clear that we have to invest a lot of efforts on preparing new generations of students whom may embrace this new revolution and can take on the problems that will arise if we pretend to minimize the technology dependency.
On the other hand, although some versions of the Lagarto processor have already been manufactured as the *preDRAC*, there are so many areas which can be worked out to improve the Lagarto architecture. This thesis is concerned with the improvement of power performance. We have already stated that the trend of devices containing computer chips is only increasing more and more. Depending on the kind of these devices, some trade-offs between power, performance and price can be established.

1.4 Goals of the thesis

This research focuses on implementing a clock gating technique in the synthesis of the Lagarto architecture in order to evaluate its impact in area, timing and power consumption. The power consumption can be evaluated in 2 different ways:

- The dafault way evaluates the power consumption without specifying a realistic activity. In such a case the evaluation tool gives an over estimate of the activity and the power consumed.
- The other and more realistic way is about trying to generate patterns of activity from realistic simulations (throughout the development of this work we are going to explain how to generate these patterns).

Thus, the main objective is to evaluate if we can enhance the performance of the Lagarto processor architecture by using the clock gating technique.
This thesis also intends to generate helpful knowledge for the Microse Group, facilitating other experts interested in designing SoC devices to get involved in the designing digital synthesis flow with the aid of the state of the art Cadence EDA tool.

1.5 Organization

The rest of this thesis is structured as follows. The chapter 2 summarizes a technical background in order to understand/refresh all the concepts needed throughout the development of this thesis. Chapter 3 explains the flow used to simulate and synthesize an SoC using EDA tools from Cadence. Chapter 4 includes the clock gating insertion to the Lagarto Processor and evaluates its impact on the performance of the SoC. Chapter 5 summarizes this work by giving all the conclusions and ideas of all the future work.

Chapter 2

Theoretical Background

The Complementary Metal Oxide Semiconductor (CMOS) technology is used to develop microprocessors, digital logic circuits, and many other integrated mixed-signal complex circuits. It facilitates low-power dissipation and high-packing density with a very noise margin. Several process steps are employed to fabricate a modern CMOS chip by integrating the NMOS and PMOS transistors on the same chip substrate. Special semiconductor regions called wells or tubs are required to integrate these NMOS and PMOS transistors on the same chip. An N-well has to be created on a P-substrate and serves as the substrate for the PMOS transistors, and NMOS transistors are commonly placed directly on a P-substrate.
Most of the information in this chapter is based on [15]

2.1 CMOS logic

This section shows how some logic circuits are built using CMOS technology (because these circuits use both nMOS and pMOS transistors). Additionally the symbols of these gates/circuits are displayed.

CMOS Inverter

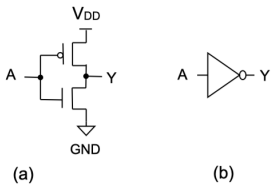

Figure 2.1: CMOS inverter: (a) schematic (b) symbol.

NAND gate

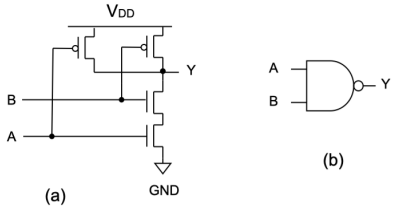

Figure 2.2: CMOS NAND gate: (a) schematic (b) symbol.

NOR gate

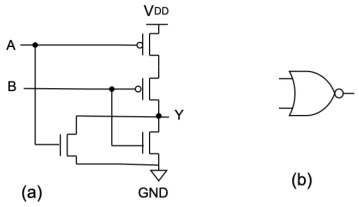

Figure 2.3: CMOS NOR gate: (a) schematic (b) symbol.

CMOS logic Gates

In general, static CMOS gates have an nMOS pull-down network to connect the output to 0 (GND) and a pMOS pull-up network to connect the output to 1 (V_{DD})

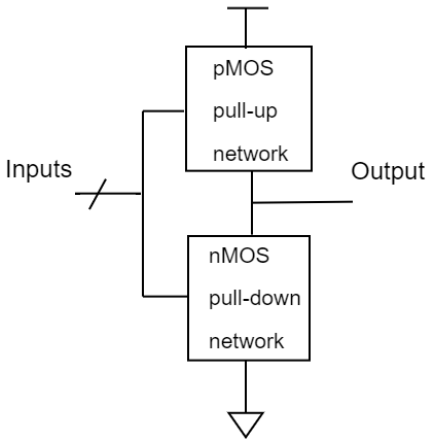

Figure 2.4: Logic gate using pull-up and pull-down networks.

Pass transistors and transmision gates

The strenght of a signal is measured by how closely it approximates an ideal voltage source. The power supplies, or rails, (V_{DD} and GND) are the sources of the strongest 1s and 0s. An nMOS transistor is almost perfect when passing a 0 (GND) and thus we say it passes a strongest 0. However, it is not as good when passing a 1 (V_{DD}). On the other side, the pMOS transistor is very efficent when passing a 1 but not when passing a 0. That is why most of the time we use pull-up networks using pMOS transistors and pull-down networks using nMOS transistors. [15] The transistor behaviours are summarized in the next figure.

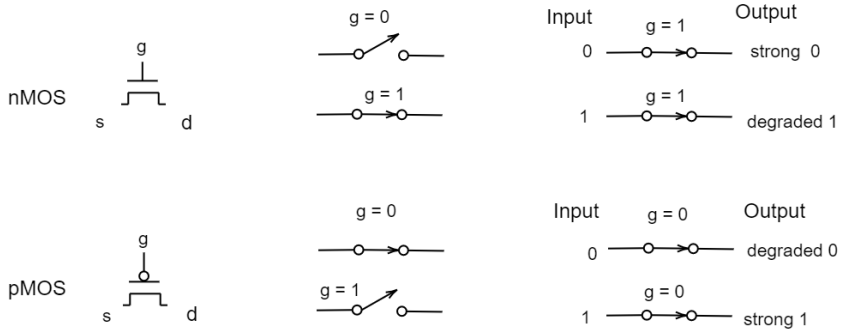

Figure 2.5: Logic gate using pull-up and pull-down networks.

By combining an nMOS and a pMOS transistor in parallel, we obtain a switch that turns on when a 1 is applied to the gate in which 0s and 1s are both passed in an acceptable fashion. We term this a transmission gate or pass gate. The next figure displays (a) the schematic symbol (b) behaviour and (c) the strength of the outputs.

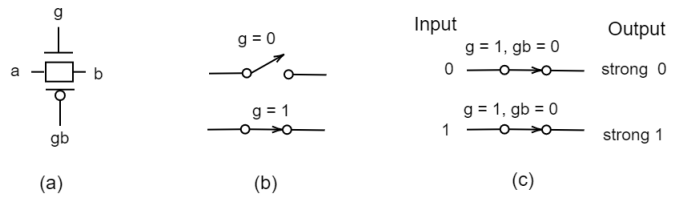

Figure 2.6: Transmission gate.

Multiplexers

Multiplexers are key components in CMOS memory elements and data manipulation structures. A multiplexer chooses the output from among several inputs based on a select signal. A 2-input, or 2:1 multiplexer, chooses input D0 when the select is 0 and input D1 when the select is 1.

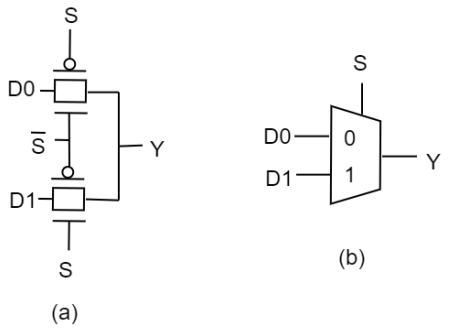

Figure 2.7: Multiplexer

Latches

A D-latch is a sequential circuit that receives a clock signal, a data input D, and it has an output Q. The latch is transparent when the clock signal is active, this means that the output "Q" follows the input "D". The D latch can be built from a 2 input multiplexer and 2 inverters. Thus, it can be built from 2 transmission gates and 2 inverters. The inverters play a restore function. A feedback path is established in order to hold the current state of Q indefinitely. The figure 2.8 shows (a) a D-latch built from a multiplexor and 2 inverters. (b) The same latch built from 2 transmission gates and 2 inverters, (c) the behaviour of the D latch when the clock signal is 0, and (d) when the clock signal is 1.

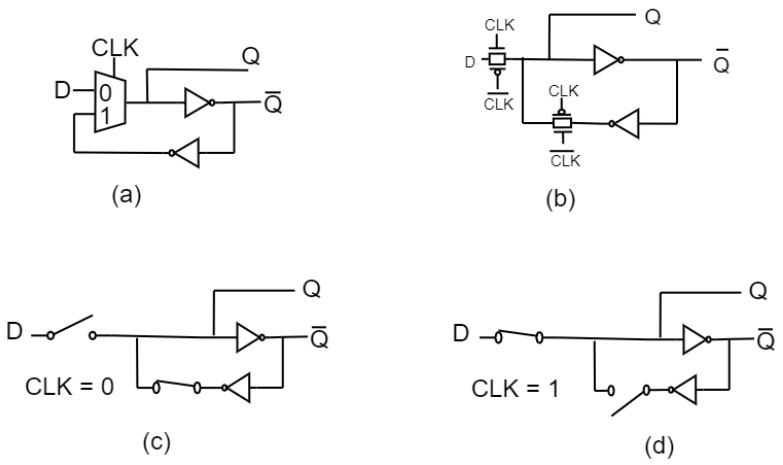

Figure 2.8: CMOS positive-level-sensitive D latch

Flip-Flops

A flip-flop is another sequential circuit which is built from 2 level-sensitive latches, one positive-sensitive and one negative-sensitive as shown in figure 2.9 The first latch stage is called the master and the second is called the slave. When the clock signal "clk" is low, the master latch is transparent and the output from this latch follows the input "D" and the slave latch retains its value. When the clk transitions from 0 to 1, the master latch becomes opaque and it retains the value from the input D (just before the transmission gates stop sampling the input). On the other hand, the slave latch becomes transparent, and the output from the flop follows the retained input from the master latch.

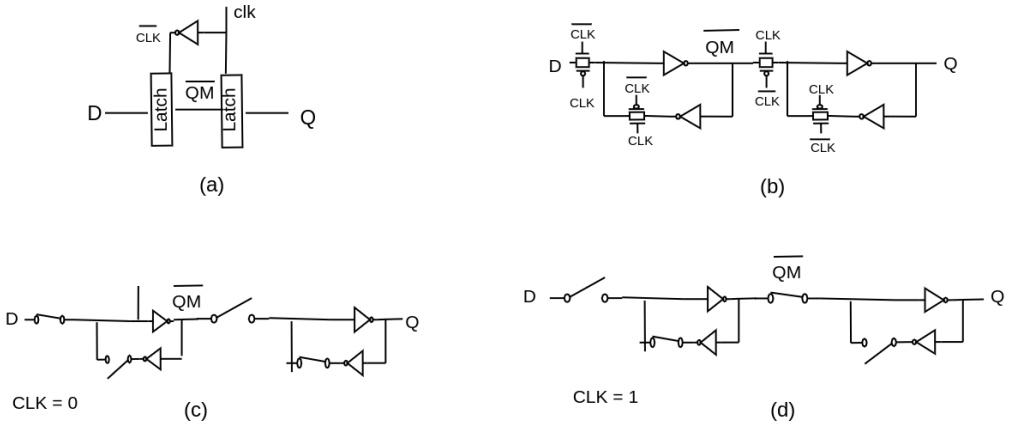

Figure 2.9: CMOS positive-level-sensitive flip-flop

2.1.1 Logic design

Digital circuits are build using CMOS logic. However, the abstraction level provided by these ones is very inefficient to build complex circuits as an SoC. We rather specify the logic of our designs using a Hardware Description Language (HDL) which provides a higher level of abstraction than schematics. The two most popular HDL's are Verilog and VHDL. System Verilog is another HDL and it can be seen as an extension of Verilog which allows the designers to improve some areas of design. The Lagarto project used in this thesis was constructed using System Verilog.
It has to be clear that no matter what HDL was preferred to specify the logic of the design because at a lower level of the hierarchy, the design is translated to logic gates, multiplexers and more.

2.1.2 Circuit design

Circuit design is the area concerned with arranging transistors to perform a particular logic function. The circuit (as a Processor) can be represented as a schematic, or in textual form as a netlist. Common transistor level netlist formats include Verilog and SPICE. Verilog netlists are used for functional verification, while SPICE netlists have more detail necessary for delay and power simulations.
Because a transistor gate is a good insulator, it can be modeled as a capacitor, C. When the transistor is ON, some current I flows between source and drain. Both the current and capacitance are proportional to the transistor width.
The delay of a logic gate is determined by the current that it can deliver and the capacitance that it is driving, as shown in the figure 2.10, for one inverter driving another inverter.

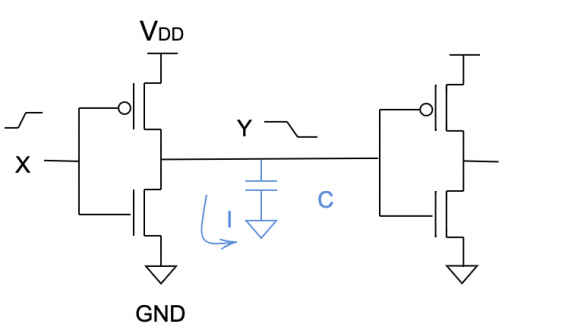

Figure 2.10: Inverter driving another inverter

2.2 Power

The instantaneous power is *P(t)* supplied by a circuit element is the product of the current through the element and the voltage across the element.

$$P(t) = I(t)V(t) \tag{2.1}$$

The energy supplied over some time interval T is the integral of the instantaneous power

$$E = \int_0^T P(t)dt \tag{2.2}$$

The average power over this interval is

$$P_{avg} = \frac{E}{T} = \frac{1}{T}\int_0^T P(t)dt \tag{2.3}$$

Power is expressed in units of Watts *(W)*. Energy in circuits is usually expressed in Joules *(J)* where *1 W = 1 J/s*.
Figure 2.11 shows a capacitor. When the capacitor is charged from 0 to V_C, it stores energy E_C.

$$E_C = \int_0^\infty I(t)V(t)dt = \int_0^\infty C\frac{dV}{dt}V(t)dt = C\int_0^{V_C} V(t)dt = \frac{1}{2}CV_C^2 \tag{2.4}$$

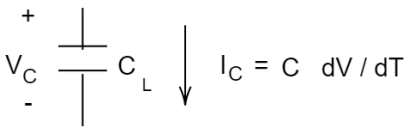

Figure 2.11: Current equation on a capacitor

The capacitor releases this energy when it discharges back to 0.

The figure 2.12 shows a CMOS inverter driving a load capacitance (C_L).
When the input switches from 0 to 1, the pMOS transistor turns OFF and the nMOS transistor turns on, discharging the capacitor (previously charged when the input was 0). The energy stored in the capacitor is dissipated in the nMOS transistor.
Whenever the input voltage transitions from logic low to logic high or from logic high to logic low, there is a small quantity of current that occurs when 2 transistors are partially fighting each other, called the *short-circuit current*.

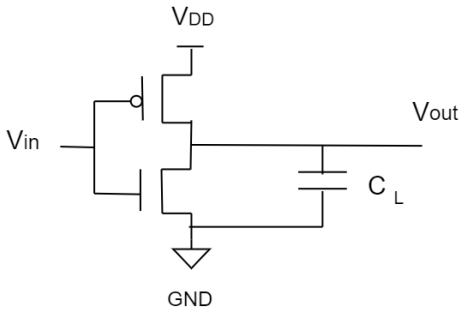

Figure 2.12: CMOS inverter driving a load

The power lost while a circuit is actively switching at a given frequency is refered to as *Dynamic Power*.
If a gate switches at a switching frequency f_{sw} during a time interval T, the load will be charged and discharged Tf_{sw} times. Then the average power dissipation is

$$P_{switching} = \frac{E_c}{T} = \frac{Tf_{sw}CV_{DD}^2}{T} \tag{2.5}$$

As most gates do not switch every clock cycle, it is often more convenient to express switching frequency f_{sw} as an activity factor α times the clock frequency f. Thus, the dynamic power dissipation can be rewritten as

$$P_{switching} = \alpha CV_{DD}^2 f \tag{2.6}$$

Dynamic power includes both:

- The charging and discharging load capacitances as gates switch
- The short-circuit power while both pMOS and nMOS stacks are partially ON.

2.3 Power dissipaation on integrated circuits

In the world of digital integrated circuits, power dissipation consists of

- Leakage power dissipation
- Net power dissipation
- Internal power dissipation

2.3.1 Leakage power dissipation

Advances in CMOS technology have resulted in a reduction of the transistor's threshold voltage, which prevents the transistor from turning off completely. As a result, static power is dissipated, which is caused by the leakage current between source and drain.
With today's technology shrinking down to 90 nm and below, the leakage power becomes more and more dominant

2.3.2 Net power dissipation

The net power (or capacitive load power) dissipation is the power consumed when charging or discharging the capacitive load which includes the net capacitance and the capacitance of the input pins.
Power dissipated by the internal capacitance is modeled as part of the internal cell power.

2.3.3 Internal Power Dissipation

The internal power dissipation is the power consumed by an instantaneous short-circuit connection between the voltage supply and the ground when

the gate transitions and the internal net power dissipated when charging or discharging internal capacitances.
The figures 2.13 and 2.14 display an example of a short-circuit connection and the current produced at the output of an inverter.

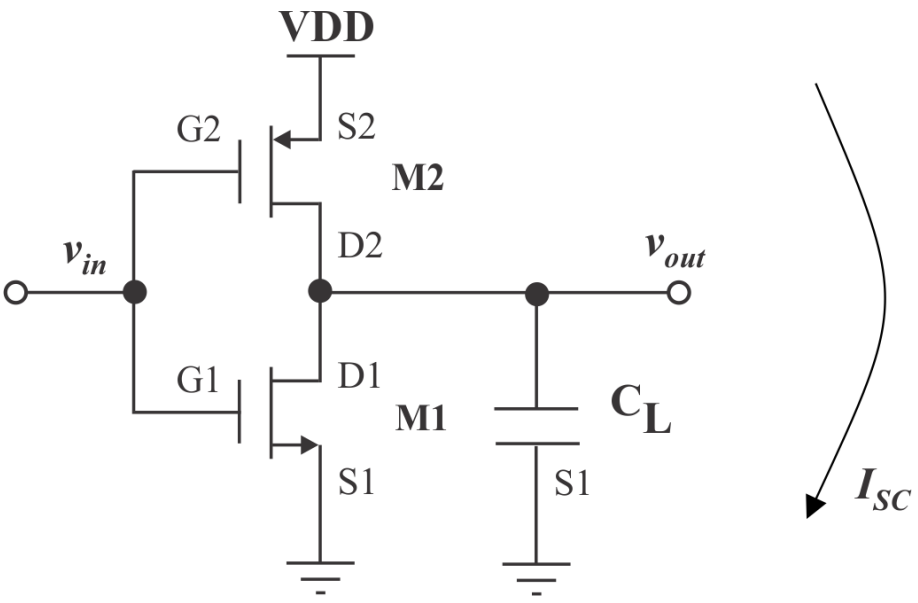

Figure 2.13: short-circuit between VDD and GND

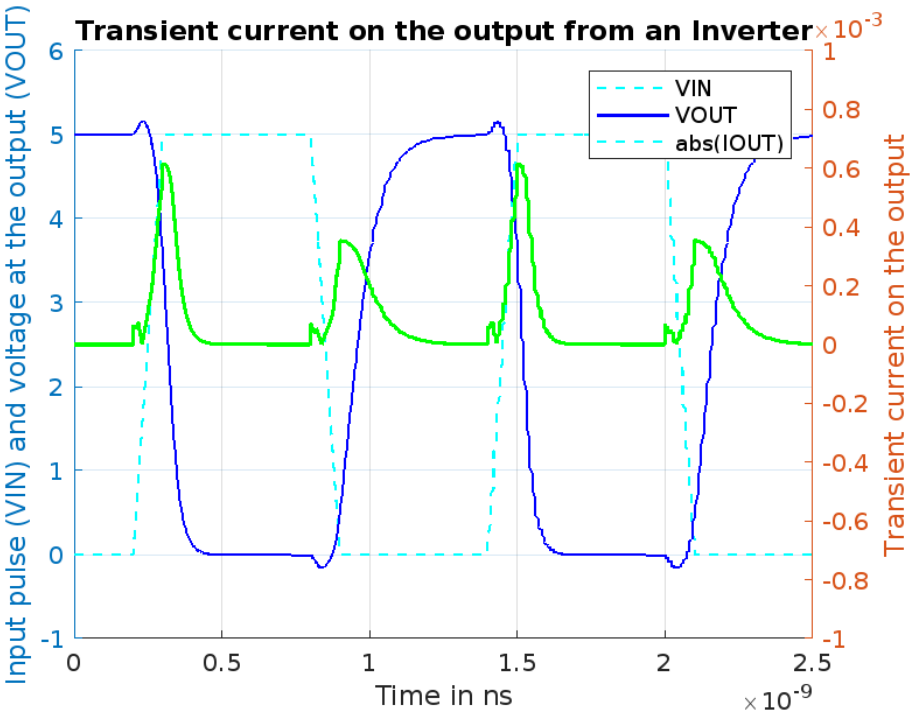

Figure 2.14: short-circuit current on an inverter (green line)

2.4 Clock gating

Dynamic Power is mostly conformed by the switching power given on equation 2.6. As it can be seen, it is possible to reduce the dynamic power by reducing the activity factor α. Activity factors are dependent on the particular application being executed. In further chapters we are going to cover how CAD tools help us to obtain accurate power data from realistic simulations. Complex designs as the Lagarto RISC-V processor has a lot of blocks (for example registers) that are not always required to supply a value. However, the clock signal (which is the only signal with an activity factor of 1) triggers all of these blocks independently if these blocks are required or not. Notice how the figures from the 2.8 and the 2.9 indicate that independently of the data being selected on the multiplexer, the clock signal will trigger the circuit and the capacitive load will be charged and discharged every clock cycle. In order to reduce the activity factor from a lot of registers, cells, or blocks from a design, we can use a gating circuit that will not allow the clock signal to

trigger a register or block whenever it is not needed during the execution of an application. Thus, gating a group of latches or flip-flops that are enabled by the same control signal reduces unnecessary clock toggles and in this way we can achieve a big dynamic power saving. The figure 2.15 shows the basic idea of clock gating.

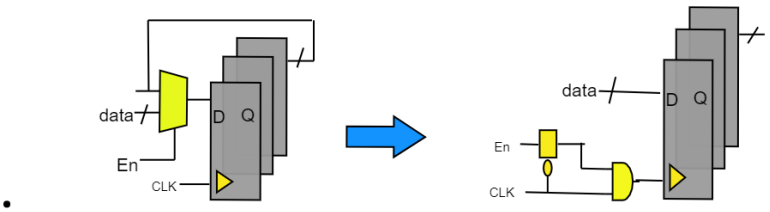

Figure 2.15: Clock gating illustration

Clock gating can be employed in any enabled register. Although the enable logic can be implement in different ways, most of the time the clock gating logic implementation uses a latch in order to ensure that the enable signal does not change while the clock is active. Thus, an inverter is also used on the clock signal to generate the control signal of the latch.

When a large block of logic is turned off, the clocks can gated early in the design on a different stage turning off not only the register but a portion of the clock network. This can save a large amount of power and area on the design. The figure 2.16 shows a design which has multiple instances of clock gating. However, these instances share the same enable signal. Thus, it can be extracted this common enable. This can improve the dynamic power savings because it reduces de switching activity and by using less clock gating instances, it also reduces the logic complexity.

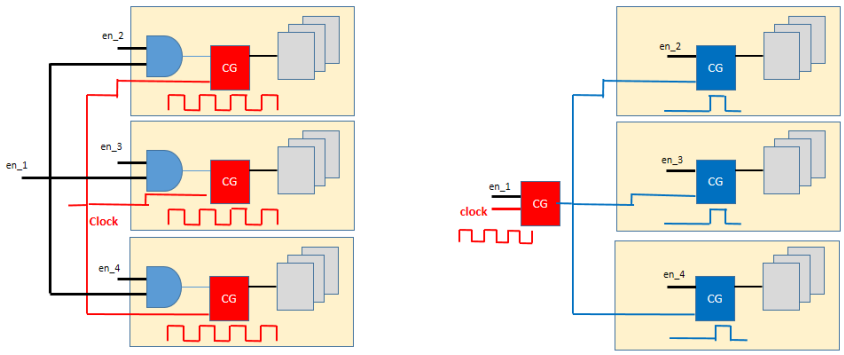

Figure 2.16: Clock gating share

One disadvantage when gating the clock of a design using clock gating standard cells from a technology library is that most of the times, we are not able to see how this standard cell is built. Sometimes it is important to see the standard cell at the transistor level. For example, in [12] the authors explain that the figure 2.17 shows how a conventional clock gating cell is implemented (latch based - positive edge triggered) and then they provide a new model which uses less transistors. However, it does not indicate neither if those transistors in the layout view are linked to a standard cell from technology library nor we can't affirm the clock gating cell we are going to use is like the one seen in figure 2.17. As mentioned earlier, in this work it is not visible how our standard cell are built. So this provides us a limitation in our work scope.

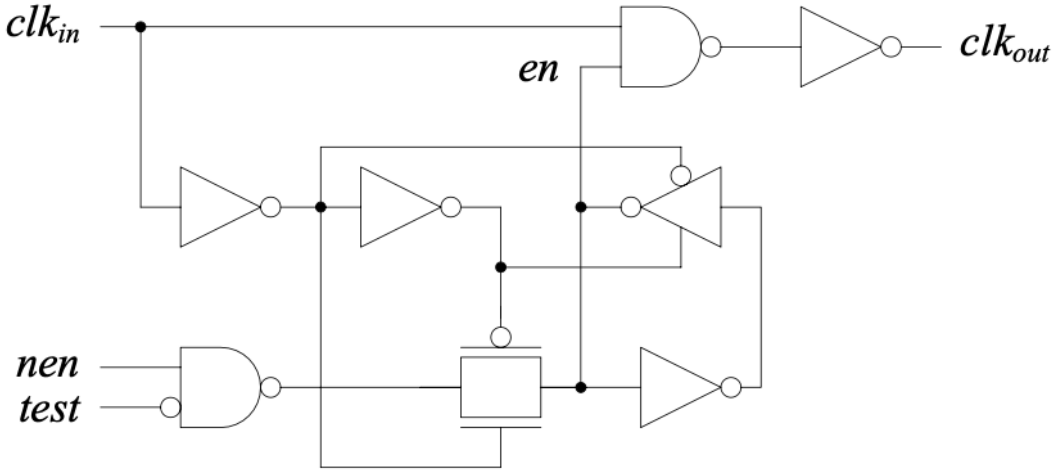

Figure 2.17: Conventional clock gating cell according to Martin Laurent and Animesh Datta

Chapter 3

Simulation and synthesis of an SoC

This section first introduces some basic concepts on computer architecture. Then, we will cover the simulation process for the DRAC project in this work. Finally, it will cover the synthesis of the DRAC project used in this thesis.

The design of complex systems as the ones designed in the world of computer architecture usually start by doing specifications according to the requirements of the design. These specifications are generally established using some Hardware Description Language. During the design process flow there are some steps in order to assure that the design described using the HDL is working correctly. Actually because of the the high costs of manufacturing chips, some areas of research are emerging on testing in order to avoid as much as possible failures on the manufactured design.

3.1 EDA tools

The Electronic Design Automation companies emerged because of the necessity on a rapid growing market of electronic designs and circuits. As the transistors shrinked more and more every two years according the Moore´s Law, and the electronic designs needed to be more complex, it was noticed

that some processes could not be tackle by human operators. These companies offer tools in order to make more efficient all the phases of a design process.

The EDA business has profoundly influenced the integrated circuit (IC) business and vice-versa, e.g., in the areas of design methodology, verification, cell-based and (semi)-custom design, libraries, and intellectual property (IP). In 1987, commercial logic synthesis is launched, allowing designers to automatically map netlists into different cell libraries. During the 90s, synthesizing netlists from RTL descriptions, which are then placed and routed, becomes the mainstream design methodology. Structural test becomes widely adopted by the end of the decade. RTL simulation becomes the main vehicle for verifying digital systems. Formal verification and static timing analysis complement simulation. By the end of the 1990s, Cadence and Synopsys have become the main commercial players. The EDA market for ICs has grown to approximately 3B dollars. [8].

The next picture illustrates some of the Cadence EDA tools

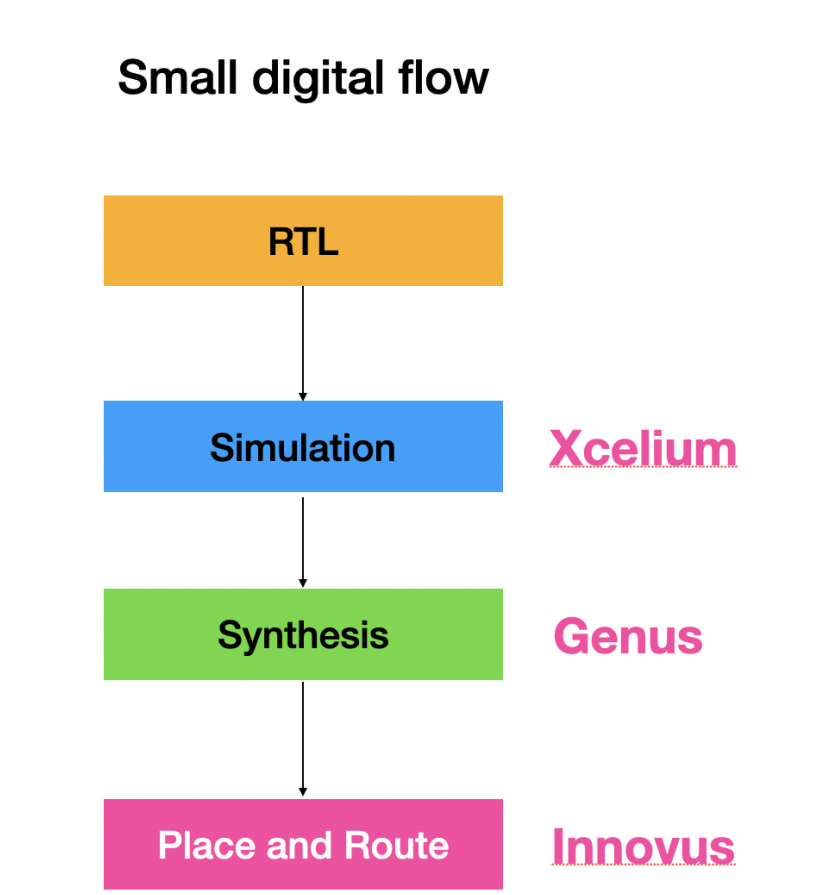

Figure 3.1: Some EDA tools related to an specific process

3.2 Simulation

Currently, the DRAC core has 4 different ways of testing/simulation.

- RISC-V ISA Tests
- Torture tests
- FPGA test

3.2.1 RISC-V ISA tests

The riscv-test repository is part of the RISC-V SW tools. This repository is a set of open-source assembly programs that test each instruction individually. The RISC-V cross-compiler compiles each test, and all regular assembly directives can be used. The test repository includes integer, floating-point, and vector extensions; it is possible to use different levels of execution privilege, virtual memory, and the timer interrupt. [1] During the development of this work, we are only using the RISC-V ISA tests.

3.2.2 RTL simulation

In order to check the correct functionality of the DRAC project, we made use of the Cadence Xcelium Logic Simulator. Further, we used multiple automated scripts and a directory structure in order to run the simulation as easy as possible. This directory structure is as follows inside the *simulation_folder*:

- Makefile

  We use a Makefile to automate the simulation process. Some of the most important parts this file does are the following:

  - Sets all the RTL files, packages, and other files to build the top instance
  - Sets all the files corresponding to testbenches
  - Defines all the xrun flags (e.g -64 bit, -access +rwc)
  - Target for only elaborate the design
  - Target for simulating the design with/without gui
  - Target for simulating a GLS

- scripts folder

  This folder has multiple tcl files which initializes the memory and also defines the RISC-V test to be simulated. These files are used by the targets in the Makefile.

- chip_top_tb.sv

  This is the testbench file. In this file the chip_top gets instantiated. It also defines the task **load_mem** which is used to load the ISA test into main memory.

Using Xcelium, we were able to run logic simulations and visualize signals waveforms to verify circuit performance. The next pictures illustrate something about this.

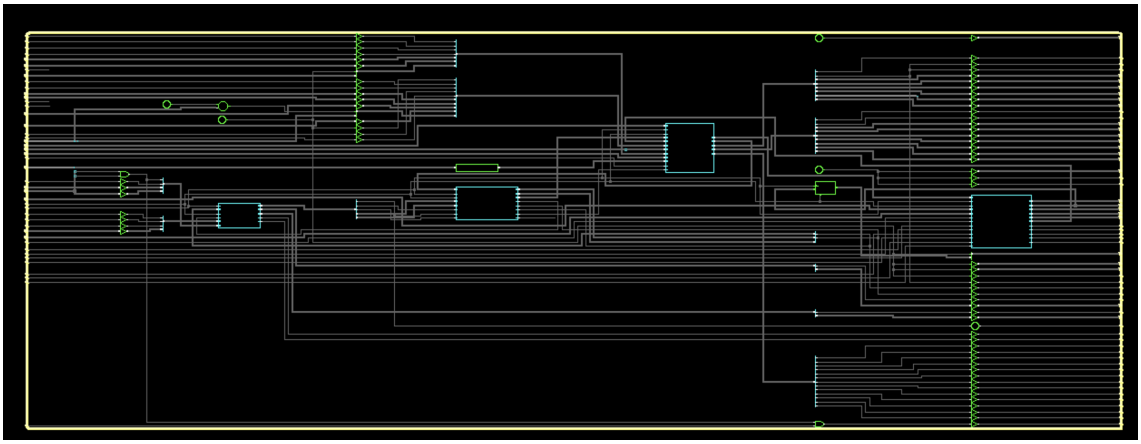

Figure 3.2: Lagarto core schematic

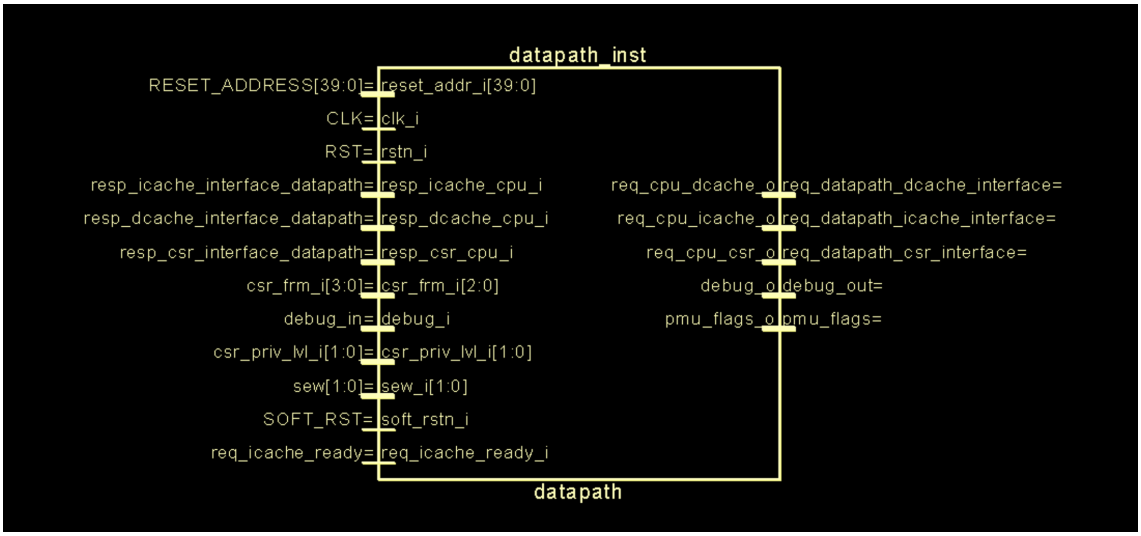

Figure 3.3: datapath schematic inside core hierarchy

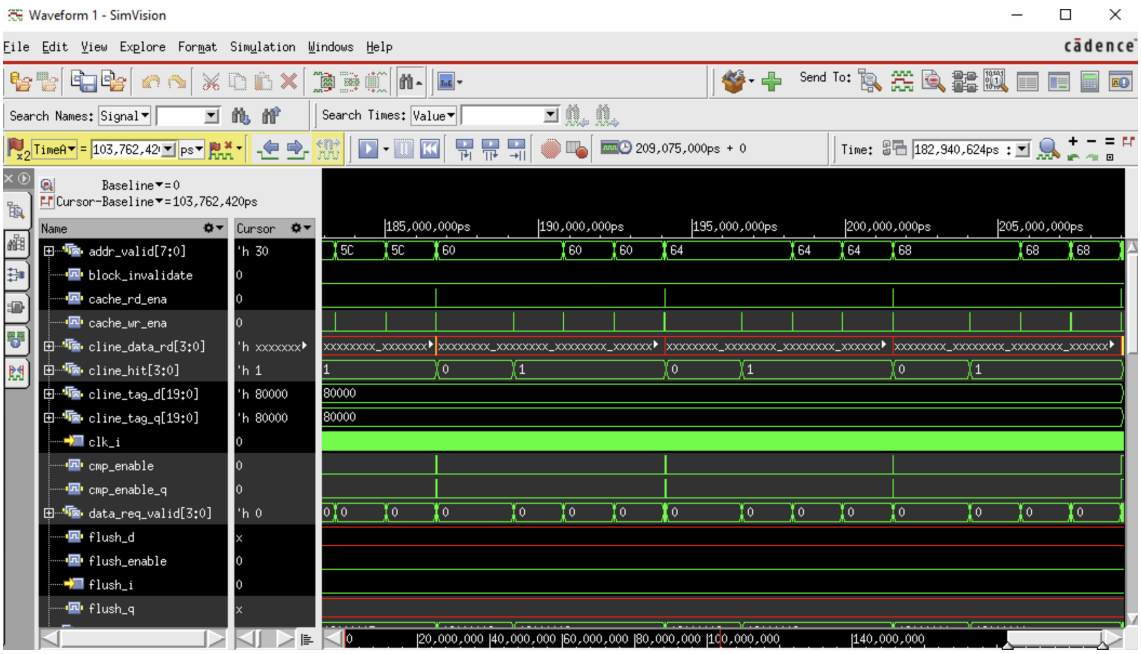

Figure 3.4: waveforms from the icache

With the aid of Xcelium, we could print the memory to verify the system response as well as the ISA execution successfully completion, see the next picture

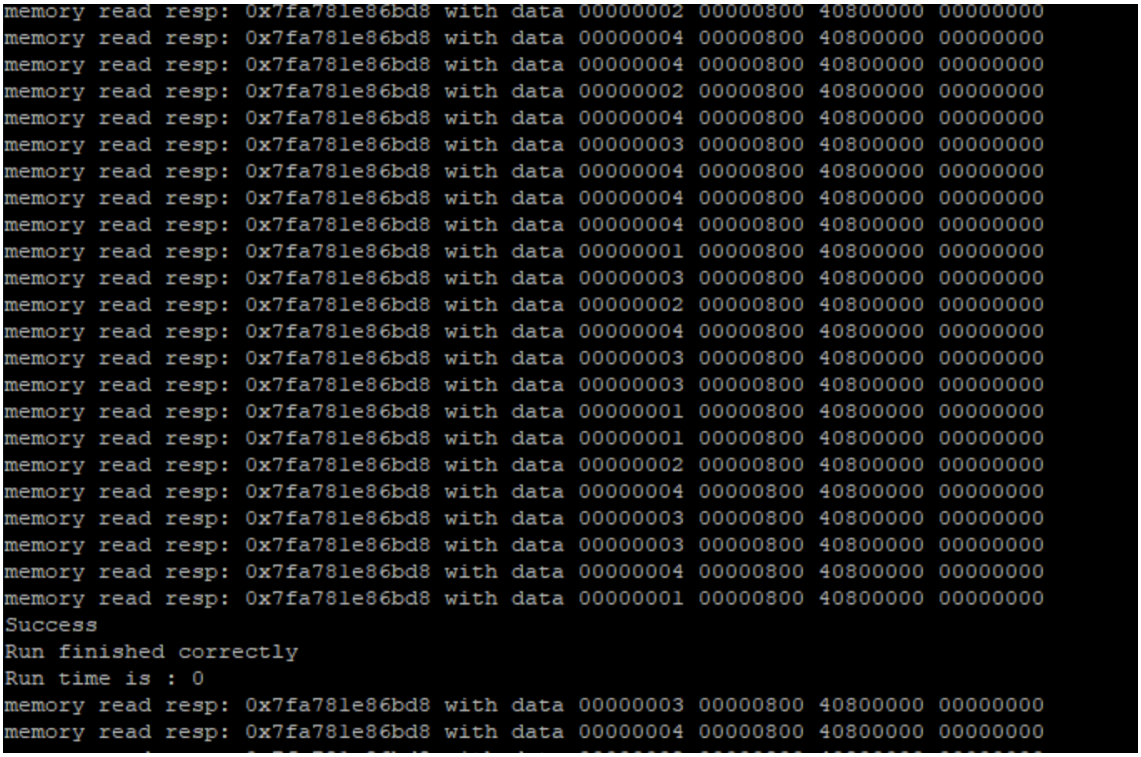

Figure 3.5: Main memory response

3.2.3 Generating a stimulus file

Stimulus, a key ingredient of power analysis, is the dump of signal activity data from simulation of the design. All simulators (and emulators) offer utilities that dump signal activity data into files for use by tools such as waveform viewers, debugging tools, and power analysis tools. [5]
In order to generate an stimulus file containing accurate switching activities the design both RTL or Gate Level Netlist, should be simulated using a testbench that

- Represents the real chip behaviour
- Generates the pin activities of all the pins in the design

Some of the stimulus formats we can generate using our Cadence EDA tools and the ones we used in this work are the next ones:

- VCD file: The value change dump (VCD) file contains information about value changes on an specific scope of the design. The file contains header information, variable definitions and value changes for all specified variables.
- TCF file: This is a text format that records average activity data (toggle count) over certain simulation duration
- SHM file: As the VCD format, it captures the changes of signals over time but it has the adventage that stores the values in binary format which makes the stimulus files smaller than the VCD format files. This format is more recommended to use when simulating big designs.

3.3 Synthesis

This section explains the full process of synthesis and also defines some basic concepts used along the development of this thesis.

3.3.1 What is synthesis?

Logic synthesis is basically the process of transforming hardware description language (HDL) code into a logic circuit based on a compiled technology specific library and user specified optimization constraints. [3]
The result of the sysnthesis is thus, a Gate Level Netlist.
The tool used for the process of synthesis in this thesis was Genus an EDA tool from Cadence.

3.3.2 Synthesis flow

The basic synthesis flow to follow on Genus can be seen in the flowchart in picture 3.6 which can be referred in [4].

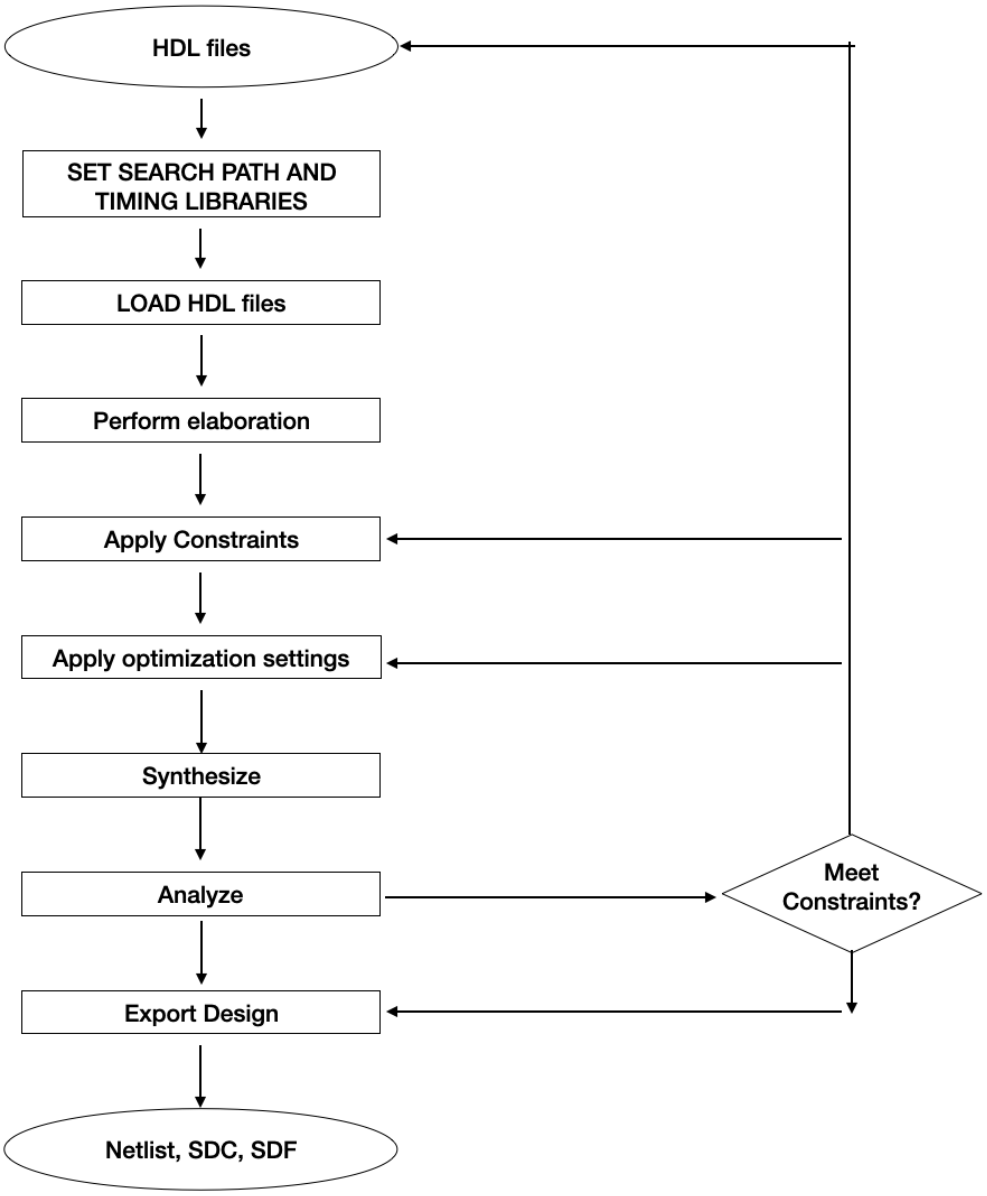

Figure 3.6: Synthesis Flow

3.3.3 Different stages of synthesis

The synthesis flow uses different stages to generate the best possible implementation of our design. The 3 different synthesis stages are:

- Generic synthesis
- Mapped Synthesis
- Optimized synthesis

Generic synthesis: This synthesis is independent and has no connection to the technology liberty files. Some of the tasks this synthesis execute are: logic pruning, resource sharing, multiplexer optimization, retiming.

Mapped synthesis: This stage of the synthesis is the one in which standard cells from the technology library are selected to implement the circuit.

Optmized synthesis: Some of the optimizations performed during this stage are: fix timing violations, incremental clock gating.

3.3.4 Directory structure for synthesis process

This sub section explains the files used as inputs for the synthesis of the DRAC project. It also gives details on generated files (outputs).

Inputs

- RTL files

  These are the files describing the design in a hardware description language. (Required file)

- Liberty files

  Files containing the standard cells, including the information related to power and timing specifications. The chip manufacturing companies provide these files (Required file).

- LEF files

  Files contain physical information and design rules of the standard cells you are using. This is not a required file to perform the synthesis but it can enhance the process to obtain more accurate results.

- Captable files

  The captable files contain the information about resistance and capacitance values which are used to model the interconnect parasitics of a design. This is not a mandatory file but it can enhance the process of synthesis.

- SDC file

  An ASCII text file (with the extension .sdc) that contains design constraints and timing assignments in the industry-standard Synopsys® Design Constraints format. The constraints in a Synopsys Design Constraints file are described using the tcl tool command language and follow tcl syntax rules. (Required file)

- TCL scripts

  The tcl scripts contain all the Genus commands. (Required file in order to make use of multiple genus commands at once)

3.3.5 Library information

The library TCBN65LPHPBWP is part of the technology process named CLN65LP from Taiwan Semiconductor Manufacturing Company (TSMC). The Computer Research Center (CIC-IPN) has signed an NDA within the Europractice Consortium to access the TSMC 65nm CMOS technological node. [14]
Some of the features of this library are:

- Number of tracks: 10 tracks for high speed;

- Threshold voltages: regular-Vt; high-Vt for low leakage; low-Vt for high speed.

- Timing modeling: NLDM for faster synthesis; ECSM for higher timing accuracy (In this work, we use NLDM).

In this synthesis process, only core cell libraries are used , regular-Vt, and three different corners: typical, best case, and worst case with NLDM models.

Corner	Conditions	Library name
typical	1.2 V and 25C	tcbn65lphpbwptc.lib
fast	1.32 V and 0C	tcbn65lphpbwpbc.lib
slow	1.08 V and 125C	tcbn65lphpbwpwc.lib

3.3.6 RTL files

All RTL files needed to build the Lagarto Core are found using an automatic process inside the Makefile. All the names and full location of all these files are generated and written into a file named src_file_sv.

3.3.7 Hard IP BLOCKS

The drac project originally only used hard IP blocks in the SRAM memories needed for the register file, cache memories, and associated tables. However, in the synthesis process of this work, we are only synthesizing a section of the full design ("datapath"). Thus, we are not requiring the Hard IP Blocks as we are not synthesizing the sram memories.

3.3.8 TCL scripts

To make the synthesis process as clear as possible, we use multiple TCL files. In these, we do things as elaborate, set the library files, execute the synthesis and more. In this way, we can automate the synthesis process and at the same time have a clearer stucture on how the files work. The list of tcl files used is the following:

- run_mmmc_test.tcl

This script is basically the one that starts the synthesis process. The main tasks this script does are the set up of some variables as the top design and the reading of the RTL files through a file named 'src_file_sv'. It also elaborates the design and writes a provitional hdl netlist.

- mmmc10t.tcl

  This file is the one that set up all the liberty files that will be used to execute the synthesis. To make it more complete, library sets are created to each library we use (3 sets). We also establish our timing and operating conditions, and RC corners. Finally, it also sets the sdc constraint file that will be used.

- run_mmmc.tcl

  This file sources run_mmmc_test.tcl and creates the directories where the generated files will be saved. Creates cost gruops, executes the synthesis and use different genus commands to write all the necessary outputs (synthesis outputs, reports, interfaces files to other tools)

Outputs

The files we want to generate after the synthesis is completed are listed below.

Gate Level Netlist: this Netlist is the representation of the HDL code into an specific technology library.

SDC constraint file: The file will constraint the netlist to be used as input for the PR process.

Reports: We generate reports for power consumption, timing, area for the design synthesis. As we will see later, the power estimate that Genus give us, is less exact to other approaches we can take.

SDF file: The SDF (Standard Delay Format) files contain timing delay information that will allow us to perform a Gate Level Simulation.

It is important to remember that in this case, we will be only making a synthesis of the datapath, which is a sub instance inside below the core hierarchy level.
Now, the results of the synthesis are going to be presented.
The Gate Level Netlist is the main output from the synthesis. As previously mentioned, the synthesis is the process of transforming HDL code from the RTL to a Netlist representation of the same RTL using standard cell from a technology library. For example, we can see in 3.7 an extract of the netlist produced by the synthesis tool when transforming the datapath design into a Gate Level Netlist.

```
 5 // Verification Directory fv/datapath
 6
 7 module fpuv_classifier_FpFormat0_NumOperands3(operands_i, is_boxed_i,
 8      info_o);
 9   input [95:0] operands_i;
10   input [2:0] is_boxed_i;
11   output [23:0] info_o;
12   wire [95:0] operands_i;
13   wire [2:0] is_boxed_i;
14   wire [23:0] info_o;
15   wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
16   wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
17   wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
18   wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
19   wire n_32, n_33, n_34, n_35, n_36, n_37, n_38, n_39;
20   wire n_40, n_41, n_42, n_43, n_44, n_45, n_46, n_47;
21   INR3D0HPBWP g1554__2398(.A1 (is_boxed_i[1]), .B1 (operands_i[54]),
22        .B2 (n_47), .ZN (info_o[10]));
23   INR3D0HPBWP g1555__5107(.A1 (is_boxed_i[0]), .B1 (operands_i[22]),
24        .B2 (n_46), .ZN (info_o[2]));
25   INR4D0HPBWP g1556__6260(.A1 (is_boxed_i[2]), .B1 (operands_i[86]),
26        .B2 (n_36), .B3 (n_44), .ZN (info_o[18]));
```

Figure 3.7: Extract from the datapath netlist

It should be note how for the module fpuv_classifier_FpFormat0_NumOperands3 the synthesis defines a set of wires n_0, n_1, n... n_47. It also instantiates an standard cell named INR3D0HPBWP using an instance named by the synthesis process g1554__2398. Then, the synthesis links the inputs outputs of these standard cells with the inputs,outputs, wires, regs, from the module. Although in this case the netlist from the datapath is a file containing 812848 lines, all the other lines apart from extract we just seen, are basically doing the same thing. These code is instantiating the datapath design and all the sub designs below this hierarchy, to standard cells from the technology library.

Yet, it is not possible to analyze all the reports that are generated during the synthesis process. We can see the more significant features of our datapath netlist.

Gates Report

The gates report displayed in 3.8 indicate that the datapath uses 326951 standard cells. This number is important and alongside with other characteristics, these are going to be compared against the same design using clock gating.

Type	Instances	Area	Area %
sequential	74030	697263.200	47.4
inverter	19868	58100.000	4.0
buffer	11261	38173.200	2.6
logic	221792	677140.800	46.0
physical_cells	0	0.000	0.0
total	326951	1470677.200	100.0

Figure 3.8: Gates report from our datapath netlist

Power Report

Instance: /datapath
Power Unit: W
PDB Frames: /stim#0/frame#0

Category	Leakage	Internal	Switching	Total	Row%
memory	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
register	4.10510e-05	1.58006e-03	7.03411e-05	1.69145e-03	3.08%
latch	1.36154e-07	8.06218e-07	6.41351e-08	1.00651e-06	0.00%
logic	5.60159e-05	2.78039e-04	5.28713e-02	5.32054e-02	96.92%
bbox	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
clock	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
pad	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
pm	0.00000e+00	0.00000e+00	0.00000e+00	0.00000e+00	0.00%
Subtotal	9.72030e-05	1.85891e-03	5.29417e-02	5.48978e-02	100.00%
Percentage	0.18%	3.39%	96.44%	100.00%	100.00%

Figure 3.9: Power report from our datapath netlist

The schematic view of our design displayed in the figure 3.10 will not reveal anything now. However, in the next section it is going to be show the instances from our clock gating insertion in the schematic view.

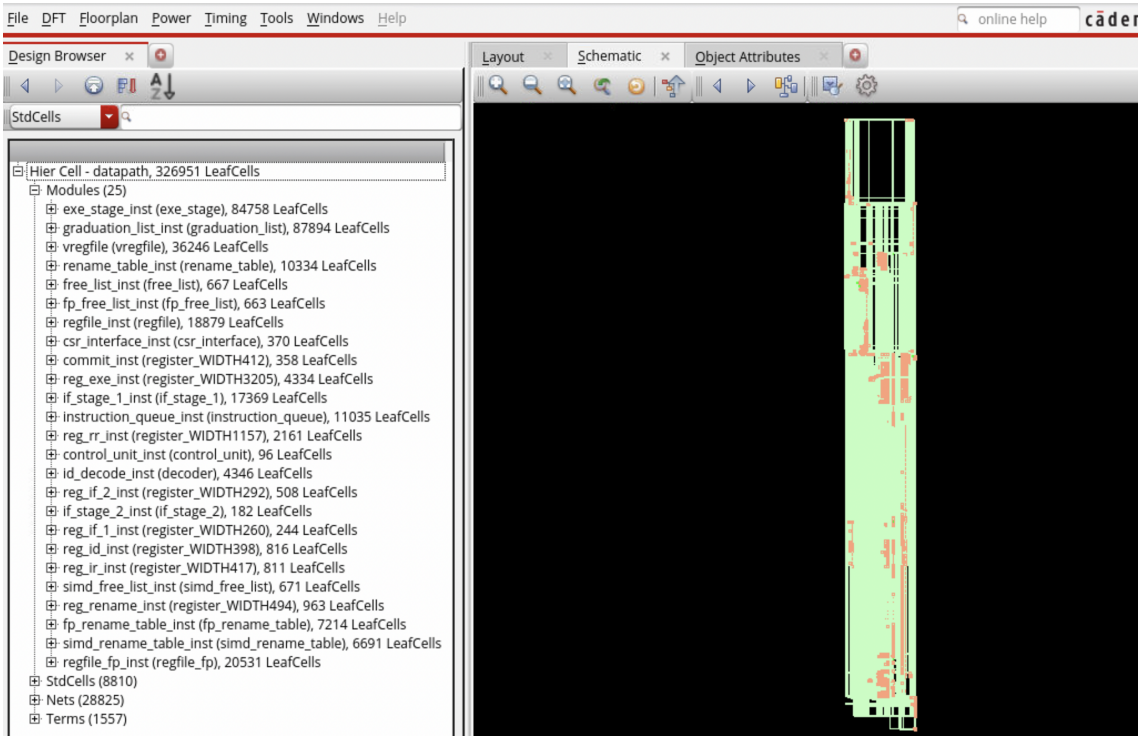

Figure 3.10: Schematic view from our datapath after the synthesis

3.4 Simulation of a Gate Level Netlist

Gate Level Simulations takes place after the synthesis process and it is very important because it allows us to make sure that the correct functionality of the design is still working after the translation from the RTL code to gate-level implementation. Although there are some other methods to ensure the correct functionality of the design as Static Timing Analysis (STA) and Logic Equivalence Checking (LEC), we will cover Gate Level Simulation because in this work we are trying to analyse the power efficiency and thus we will generate an stimulus file from the Gate Level Simulation which is a file that

will be used in another process to obtain more accurate power consumption evaluations.
Some of the reasons why Gate Level Simulations are important are the following:

- GLS can identify problems that Static Timing Analysis and Logic Equivalence Checking tools are not able to verify.
- Verifying the system initialization and reset sequence are correct.
- Validate the power-up operation on all power domains
- Validate Clock Gating insertion as this is inserted during the synthesis process
- Annotate switching factors for accurate power estimation.

There are 3 kinds of gate level simulations.

Zero Delay Simulation: This simulates the netlist without any timing information. It is mainly used to test the system initialization, power-up sequence, reset sequence, and more generally to check the proper functionality of the design.

Unit Delay Simulation: This simulation inserts a unique timing delay to all elements in circuit.

SDF Simulation: In this simulation, the delay information from the SDF files is annotated into the design simulation and all timing checks are validated. As mentioned earlier, our synthesis tool will generate this SDF file. However, the SDF files can also be generated from Place and Route tools (as the Gate Level Simulation may be performed post synthesis process in a more advanced process as the post place and route).

What we need to execute a Gate Level Simulation are the following input files:

- Gate Level Netlist
- Library cell information (in verilog format)
- Delay information (SDF file)

- Testbench to test the netlist

Besides the gate level netlist of the sub design datapath, it was needed to input all the other files that were not synthesized.

3.5 Power analysis flows

To obtain accurate power analysis results, it was needed to generate real switching activity data by simulating the design using tests close to the real functionality of the design. For exmaple, in our case we did simulate the design using official ISA tests and program benchmarks to generate accurate stimulus files.
As previously mentioned different format of stimulus files can be generated, for example: vcd, tcf, saif,shm and more using our simulator software. Furthermore these files can be generated from the RTL simulation or in a more advanced step during the Gate Level Simulation.
In order to perform the power analysis, Joules power calculator from Cadence was used. In Joules we can follow different flows to perform power analysis of our design:

- RTL power analysis flow seen in 3.11
- Netlist power analysis flow (with RTL stimulus) seen in 3.12
- Netlist power analysis flow (with Gate stimulus) seen in 3.13

[6]

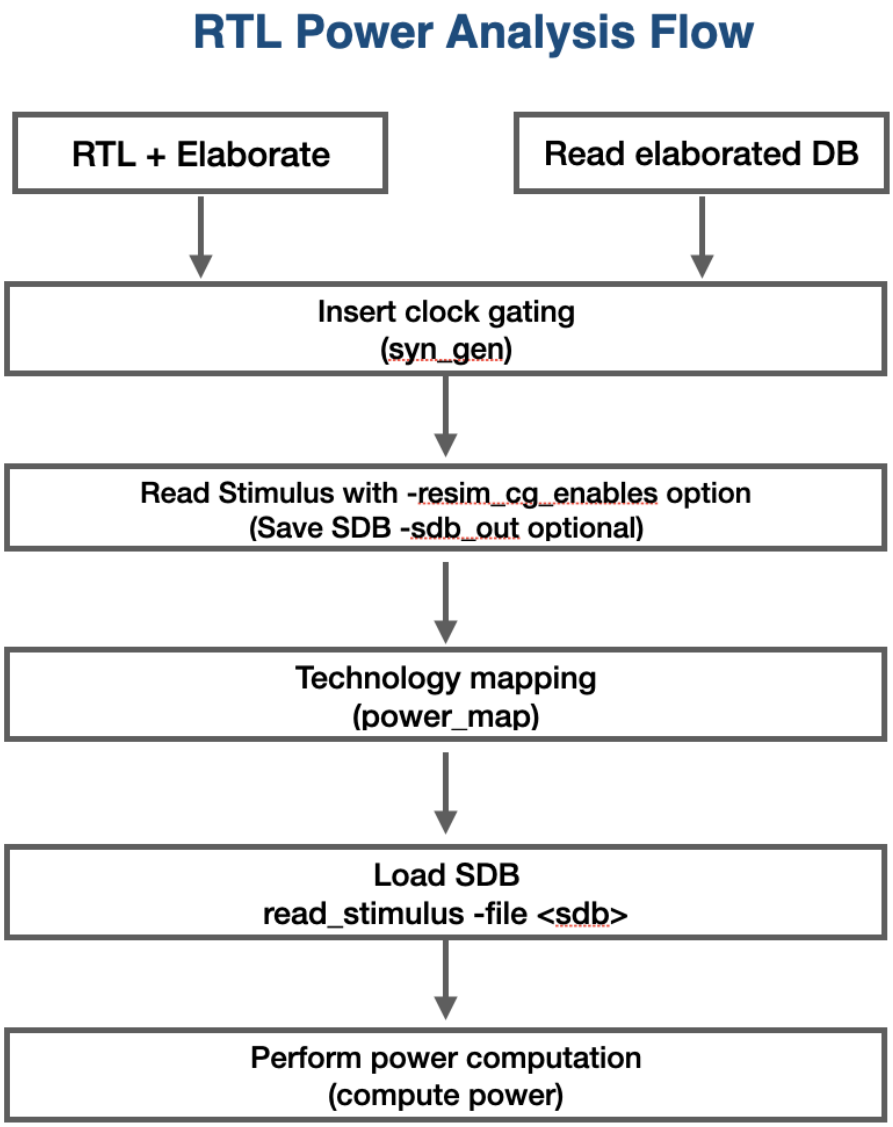

Figure 3.11: Flow to use when doing power analysis synthesizing an RTL design

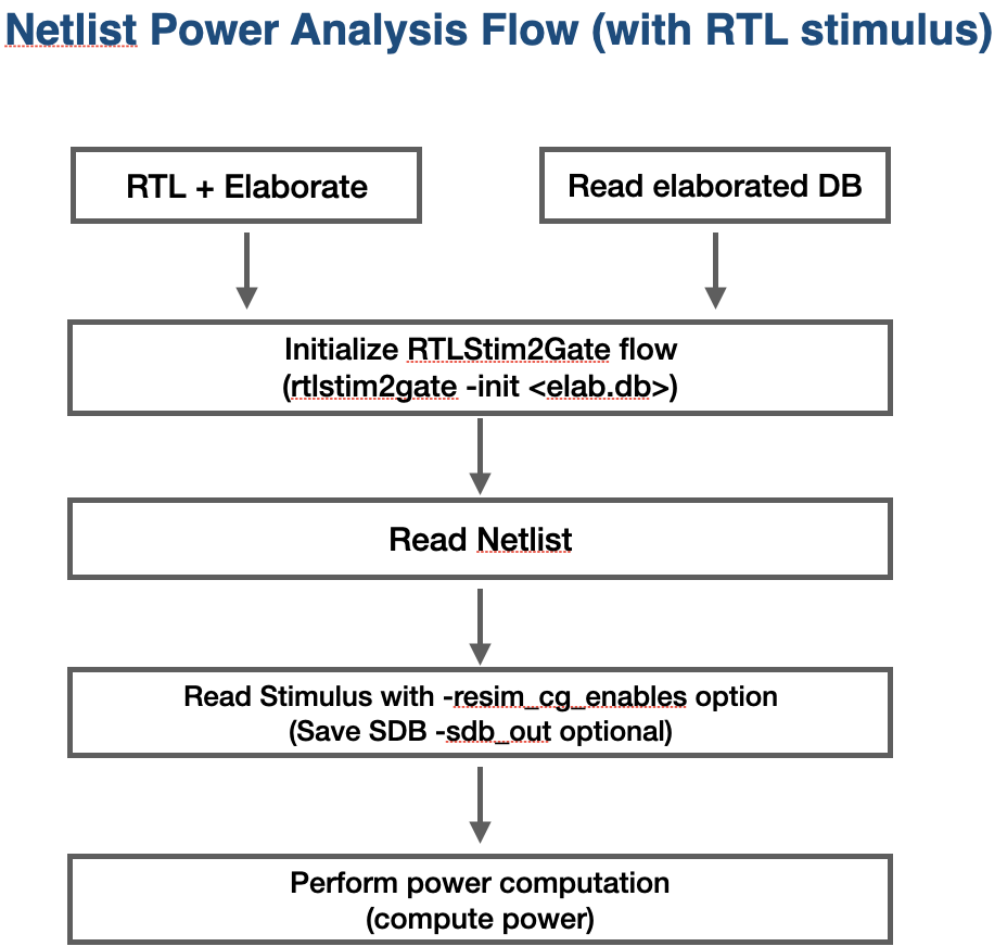

Figure 3.12: Flow to use when doing power analysis with a Netlist and an RTL stimulus

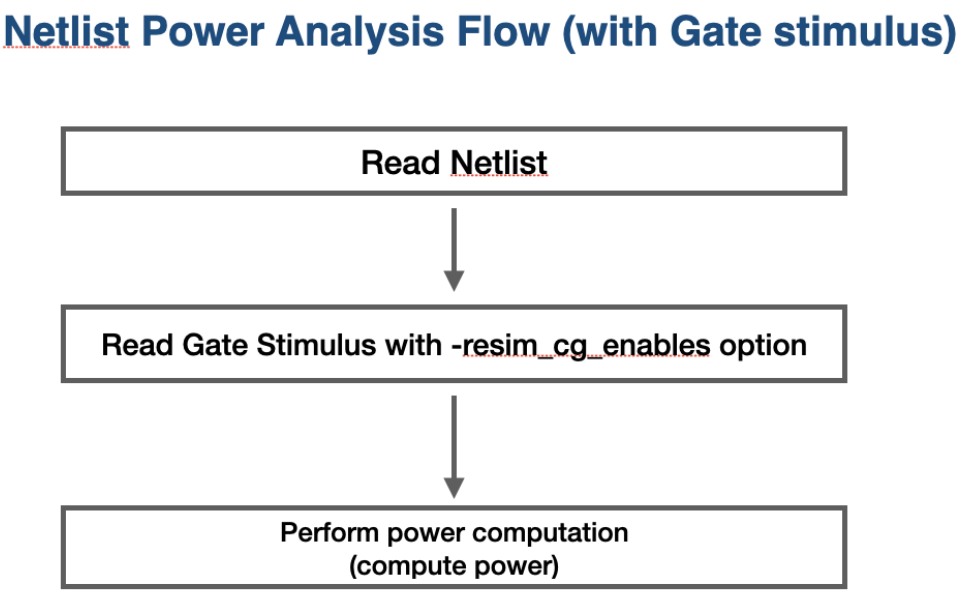

Figure 3.13: Flow to use when doing power analysis with a Netlist and an Gate stimulus

During the development of this work, it was tested the 3 previous flows but it is only going to be presented the netlist flow using gate stimulus as these are the ones that perform the most accurate power analysis. As previously mentioned, there were series of tasks that were automated using script and a makefile. The next files are some of the most important blocks in order to understand the flow of the GLS.

elab-GLS.log: $(sim_vsrcs) $(vsrcs) $(csrcs) $(TB_RTL)
xrun -f run-netlist-top.f -64bit | $tee $@; exit "$PIPESTATUS[0]"
junk += elab-GLS.log

- Xcelium elaboration of the design.
- Requires other targets that prepare all simulation files, verilog sources, c sources and the TB files needed to simulate the DRAC.

- Xcelium execution through xrun on the run-netlist-top.f

run-netlist-top.f:

- This file set the xrun flags
- Reads the library used to synthesize the design in verilog format.
- Reads the datapath netlist generated during the synthesis
- Reads the system verilog files that were not synthesized
- It sets the sdf file using the flag -sdf_cmd_file command_file_local_top
- Establishes the top module testbench

sim-GLS-RTL-gui: elab-GLS
    xrun -R -gui -input scripts/sim1.tcl -64bit

- Loads the ISA test into memory (executed in the testbench)
- Initializes the regfile_inst register.

chip_top_tb.sv

- Test bench file containing tb module (top)

Lastly, for the next demonstration it is going to be loaded into memory the rv64ui-p-and isa test.

Once the Makefile target is executed, we can validate that our clock gating cell is being instantiated under the datapath design hierarchy. See the picture 3.14. Besides the design browser, the schematic view provides the I/O we expect to see from the clock gating cell (picture 3.15).

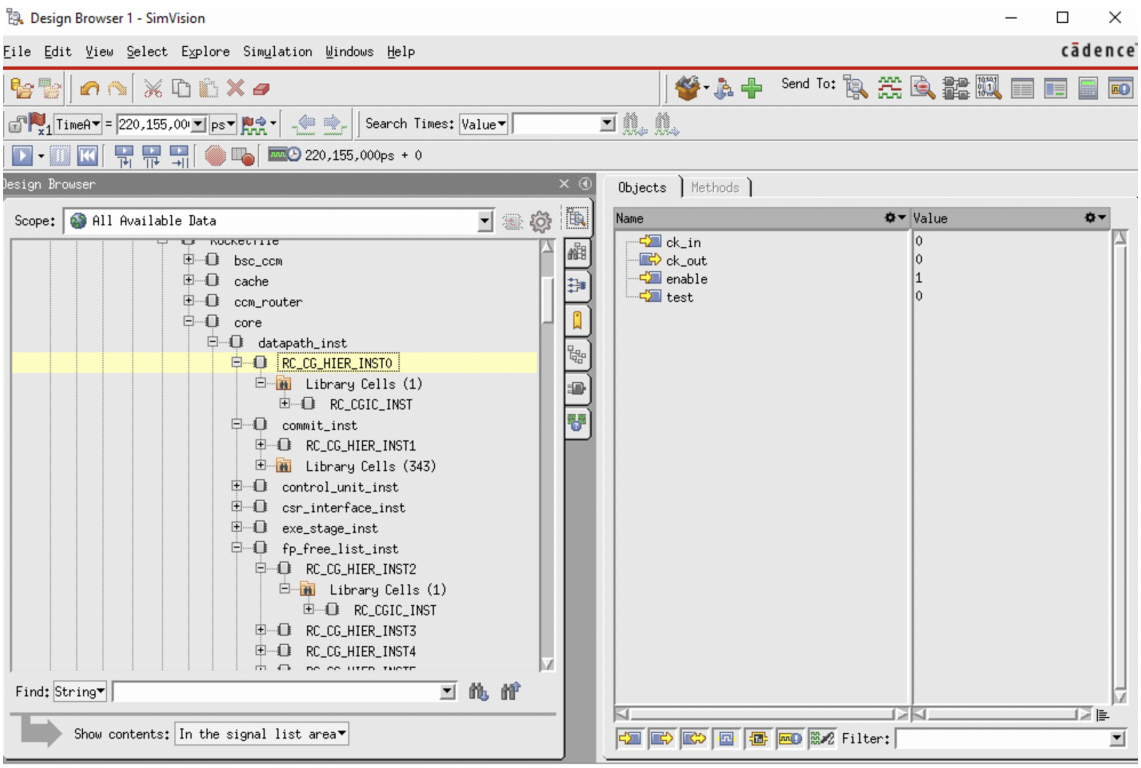

Figure 3.14: Clock gating under the datapath hierarchy

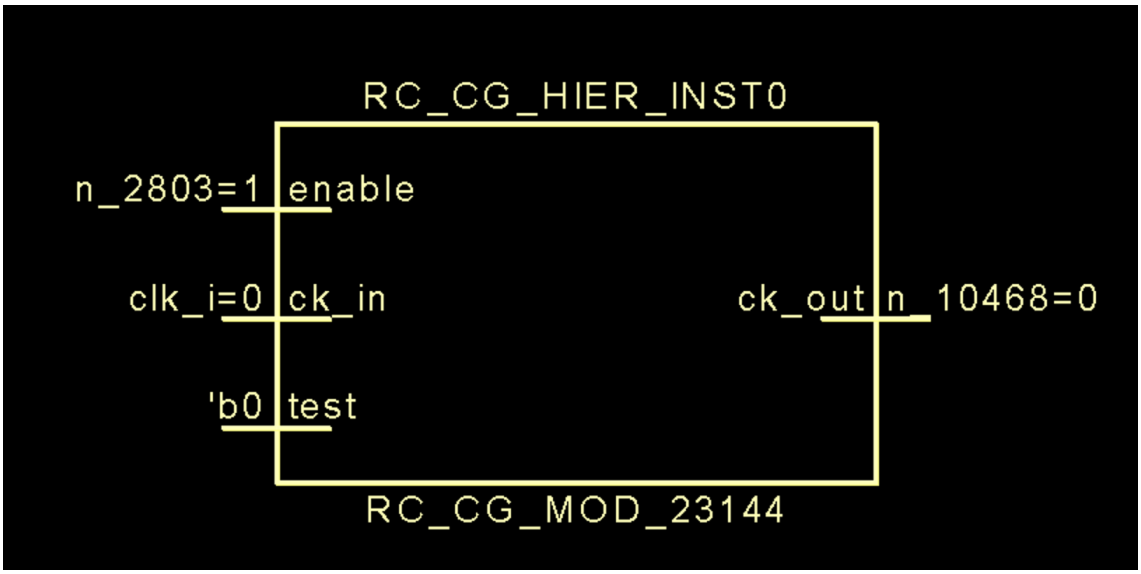

Figure 3.15: Clock gating instance schematic

Now that the GLS is being executed, some generic commands need to be executed in our Xcelium session in order to take some data of this simulation and store it in order to use it in the following processes. In our case, we are collecting this information because it is needed to make more realistic power analysis. The next is a list of commands that can be used to generate a TCF stimulus file.

- database: Lets you open a database. The kind of database you can open are SHM, VCD and EVCD.
- probe: for saving signal values to a database
- scope: for specifying design scope references
- output: name of the file to be generated
- depth: This option specifies how deep in the simulation hierarchy to probe signals
- memories: This flag specifies that multi-dimensional arrays are to be probed.
- internal: requests to include internal nets and signals in the probe.
- dumpwireasnet: requests to only dump ports in the pins section of the TCF file.
- overwrite: allows overwriting of an existing TCF file.

[7]

As can be seen in figure 3.16, it was created a TCF stimulus file. It can be seen how it is set the scope to tb.DUT.Top_asic_inst.Rocket.RocketTile.core.datapath_inst. The simulation is run in order to start dumping the signals. In this way, all of our signals (in the specified scope) will be dumped into the database.

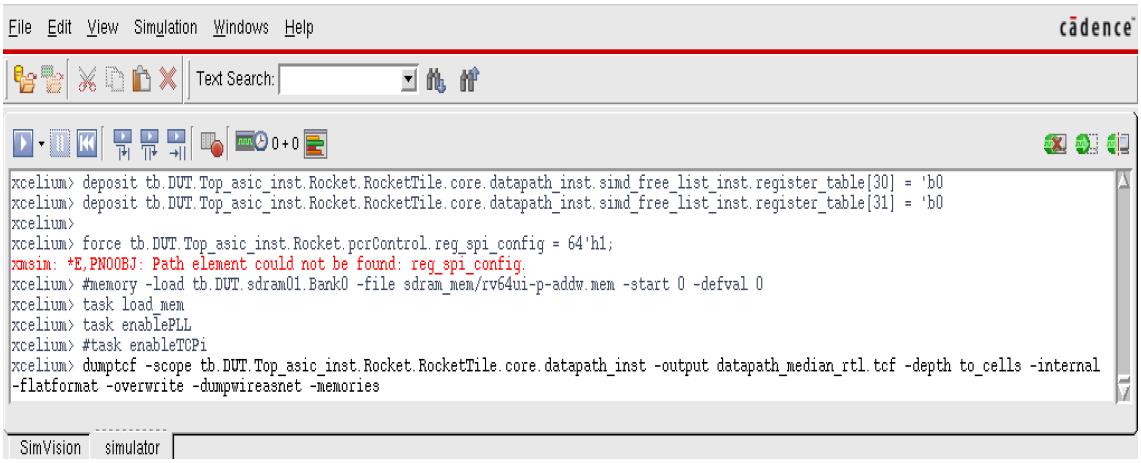

Figure 3.16: An example of TCF stimulus file creationg

Chapter 4

Clock Gating Insertion

As mentioned in chapter 2, clock gating is a powerful technique to reduce power dissipation. This chapter first will explain how clock gating is implemented on a design using Genus from Cadence. Then it will cover the results after implementing clock gating on the Lagarto architecture and analyze its different impacts on power, area, and timing.

4.1 Clock gating flow

The recommended flow to implement clock gating during the synthesis process is very similar to the one seen in 3.6. The new flow can be seen in figure 4.1.

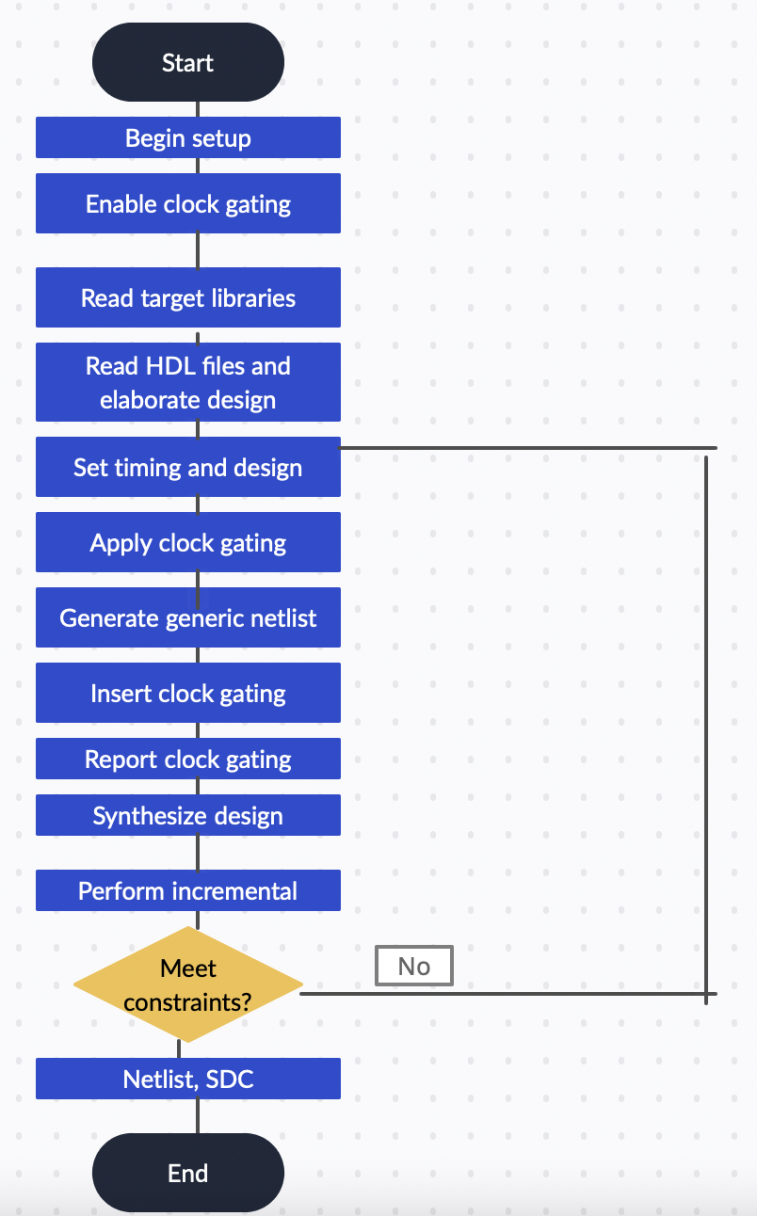

Figure 4.1: Recommended synthesis flow using clock gating

4.2 RTL coding styles

When we write our RTL code design to describe flip-flops or latches as the ones we see in chapter 2, we may or may not infer the synchronous set or reset signal on the flip-flop. In the case we do not infer the reset signal, the Genus CAD tool will not use the synchronous set or reset in the clock gating logic.
We will take the next code as an example

```
module flop(enable,in_1,in_2,clk,out_1);

input en,clk;
input [3:0] in_1, in_2;
output [3:0] out_1;
reg [3:0] out_1;

always @(posedge clk)
        if (enable)
                out_1 <= in_1 + in_2;

endmodule
```

In this example, Genus will not infer a synchronous reset if clock gating is inserted on this simple design. The next figure illustrates this statement.

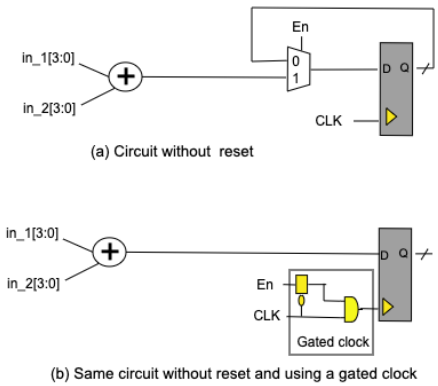

Figure 4.2: flop without reset: (a) before clock gating (b) after clock gating

If we code our flops to have a synchronous reset as shown in the next code

```
module flop(enable,reset,in_1,in_2,clk,out_1);

input en,reset,clk;
input [3:0] in_1, in_2;
output [3:0] out_1;
reg [3:0] out_1;

always @(posedge clk)
        if (reset)
                out_1 <= 0;
        else
                if (en)
                        out_1 <= in_1 + in_2;

endmodule
```

we will see that the Genus CAD tool will combine the reset signal with the synchronous enable signal to create a new signal as shown on the next figure. We can note that Genus will use this new singal as the new enable signal on the clock gating logic. Of course, this means that either the enable signal or the reset signal can trigger the clock gating logic along with clock pulse (clk).

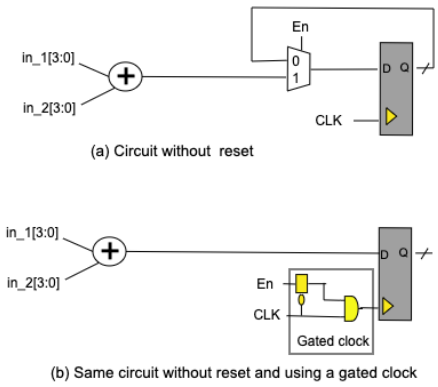

Figure 4.3: flop with synchronous reset: (a) before clock gating (b) after clock gating

The last case is when we infer an an asynchronous reset signal. In this case,

the Genus tool will not use the asynchronous reset signal in the clock gating logic inserted in the design.

```
module flop(enable,reset,in_1,in_2,clk,out_1);

input en,reset,clk;
input [3:0] in_1, in_2;
output [3:0] out_1;
reg [3:0] out_1;

always @(posedge clk or posedge reset)
        if (reset)
                out_1 <= 0;
        else
                if (en)
                        out_1 <= in_1 + in_2;

endmodule
```

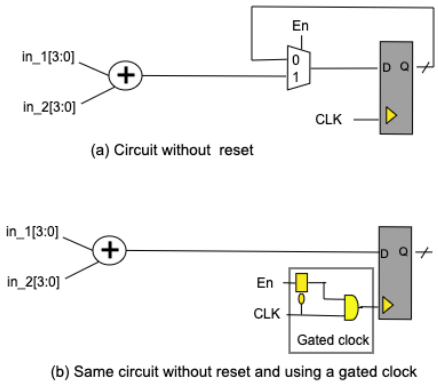

Figure 4.4: flop with asynchronous reset: (a) before clock gating (b) after clock gating

4.3 Selection of Clock Gating logic

We are able to choose in which way we can implement our clock gating logic in Genus. Note that we can implement different clock gating logic in different ways. The different ways to implement it are the ones we are going to cover on this section and are listed below:

- We may specify the library cell name of an integrated clock-gating cell
- We can define our own clock gating module
- we can use attributes to select an integrated clock-gating cell in the technology library.

The datapath RTL instance uses a positive edge-triggered logic. Thus, if we want to implement a clock gating logic, we need to implement a positive edge triggered logic. In case of a clock gating cell, we are going to instantiate a positive edge triggered latch as is the only type of cell we have available in our technology library.

Although it may seem tempting to insert our clock gating at the RTL level, the advantage of using automated synthesis over manual clock gating is obvious because automated clock gating can be easily applied with no modification in HDL coding and can directly be applied at gate-level. [2]

4.4 Synthesis using a latch posedge clock gating cell

4.4.1 Clock gating cell used

The results from inserting the clock gating logic in the datapath instance are going to be presented in this section but before we move into that analysis, it is going to be briefly presented the clock gating cell we selected to do this. The clock gating cell used is of the type: latch_posedge_precontrol.
This means it was a clock gating cell based on a latch which is transparent when the clock singal is active on its input. The precontrol comes from the fact that there is test control (for DFT designs) located before the latch.

Furthermore, as explained in [13] the FF based CG cell is neither power nor performance efficient. Hence, the Latch based and Gate based clock gating cells are the only available good options for a designer. In our case, we will not be using this input port as we will see when we analyse our schematic (it will cause no issues).

Here is an extract of the clock gating cell in our technology library.

```
/* ------------------------- *
 * Design : CKLNQD16HPBWP *
 * ------------------------- */
cell (CKLNQD16HPBWP) {
    /* gs02 */
    pg_pin(VDD) {
      voltage_name : VDD;
      pg_type : primary_power;
    }
    pg_pin(VSS) {
      voltage_name : VSS;
      pg_type : primary_ground;
    }
  cell_leakage_power : 2.012 ;
  leakage_power() {
  when : "!CP !E !TE" ;
   value : 2.054 ;
  }

  pin(CP)  {
    direction : input;
    clock_gate_clock_pin : true;
    capacitance : 0.002386;
    rise_capacitance : 0.002386;
    fall_capacitance : 0.002307;
    internal_power() {
      rise_power(passive_energy_template_8x1) {
        index_1 ("0.0048, 0.0080, 0.0152, 0.0288, 0.0568,
        0.1120, 0.2224, 0.4432");
        values ("0.0023, 0.0023, 0.0023, 0.0022, 0.0021,
        0.0021, 0.0021, 0.0024");
      }
```

```
      fall_power(passive_energy_template_8x1) {
        index_1 ("0.0048, 0.0080, 0.0152, 0.0288, 0.0568,
        0.1120, 0.2224, 0.4432");
        values ("0.0075, 0.0075, 0.0074, 0.0074, 0.0073,
        0.0073, 0.0073, 0.0076");
      }
    }
    timing() {
      related_pin : "CP";
      timing_type : min_pulse_width;
      fall_constraint(width_template_3x1) {
        index_1 ("0.0048, 0.0536, 0.4432");
        values ("0.0690, 0.0729, 0.5558");
      }
    }
  }

  clock_gating_integrated_cell : "latch_posedge_precontrol";
  cell_footprint : "cklnqd1" ;
```

In this extract we can see the following.

- The name of the cell: CKLNQD16HPWBP
- pg_pins : VDD and VSS. These are the pins that are going to power the cell. These pins are not used for the logic of the cell.
- Leakage power of the cell.
- Specifications of the pin(CP).
  Pin(CP) is an input pin. It is the clock pin from the clock gate as it is indicated by clock_gate_clock_pin. The capacitance from the pin, rise_capacitance and fall_capacitance. Some specification for power, and the timing for the CP pin. At the end of this extract it is indicated it is a clock gating cell of type latch activated during the positive edge trigger of the clock by the specification
  clock_gating_integrated_cell: latch_posedge_precontrol.

The following is the schematic view of the cell. As sated before, the pg_pins are not part of the schematic symbol. The next figure illustrates a part of the schematic where the clock gating cell can be seen. We can see all the

input pins: E, CP, TE. The name of the cell and lastly we can see the output pin Q.

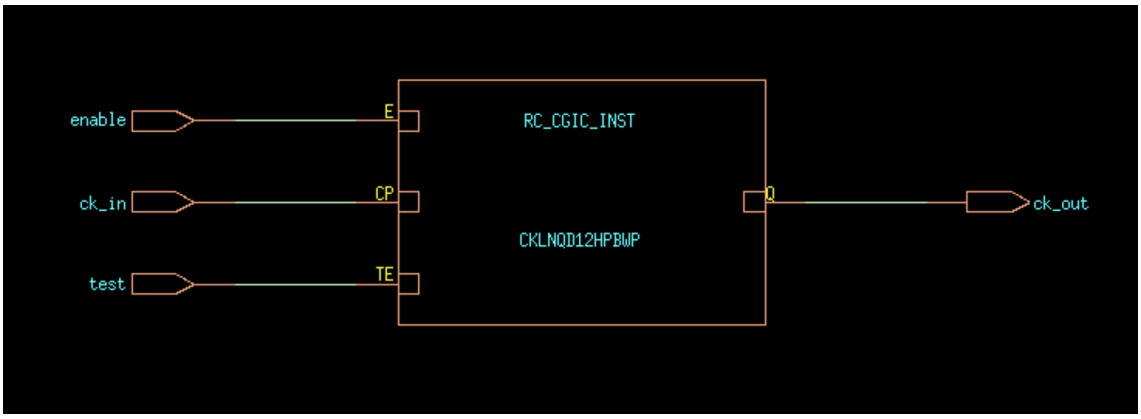

Figure 4.5: Schematic view of the clock gating cell

4.4.2 Datapath synthesis results

The clock gating cell CKLNQD16HPWBP was inserted during the datapath synthesis. There were 1326 instances from the cell CKLNQD16HPWBP inserted.

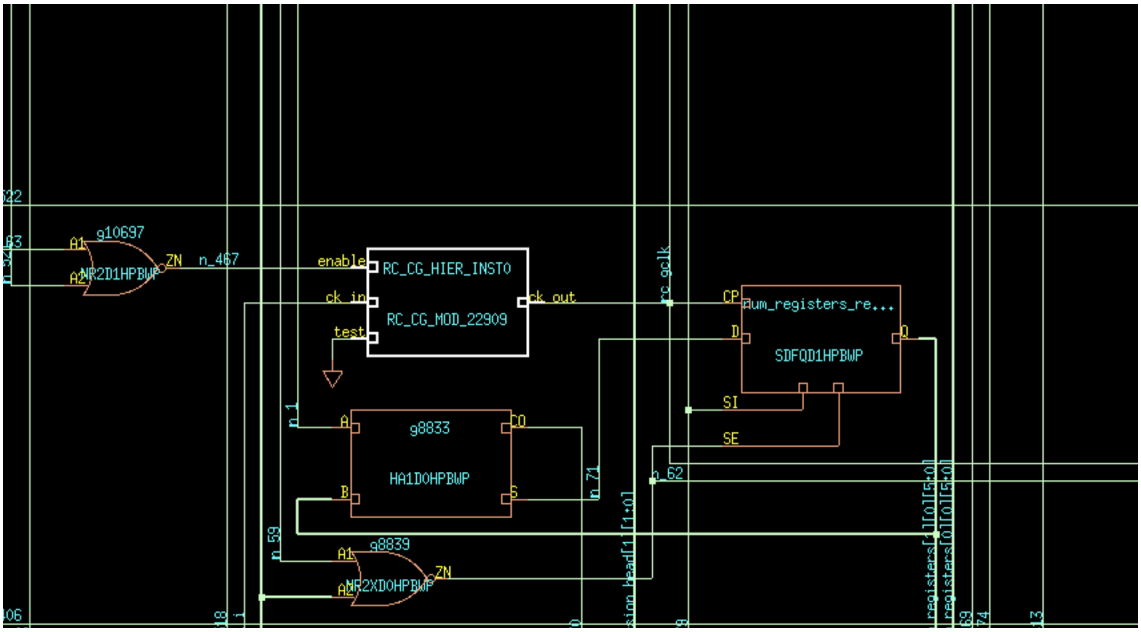

Figure 4.6: Schematic view from one clock gating instance

The netlist instantiates the cell in the next way. It creates a module and then instantiates the clock gating cell. Next, it uses an instance from this created module inside a block from the original netlist (if_stage_2 in this case).

```
module RC_CG_MOD_345_23699(enable, ck_in, ck_out, test);
  input enable, ck_in, test;
  output ck_out;
  wire enable, ck_in, test;
  wire ck_out;
  CKLNQD12HPBWP RC_CGIC_INST(.E (enable), .CP (ck_in),
  .TE (test), .Q
       (ck_out));
endmodule

module if_stage_2(clk_i, rstn_i, fetch_i,
          resp_icache_cpu_i, stall_i,
       flush_i, fetch_o, stall_o);
....
RC_CG_MOD_315 RC_CG_HIER_INST315(.enable (n_176),
       .ck_in (clk_i),
```

```
        .ck_out (rc_gclk), .test (1'b0));
```

The summary of our clock gating insertion is displayed in the picture 4.7. As we can see, there are 1326 clock gating instances. Furthermore, 93.73 percent of the flip-flops in the design were gated by the cg instances.

```
Summary
-------
--------------------------------------------------------------------------------
            Category            Number     %    Average Toggle Saving %
--------------------------------------------------------------------------------
 Total Clock Gating Instances     1326  100.00                            -
--------------------------------------------------------------------------------
 RC Clock Gating Instances        1326  100.00                        93.35
 Non-RC Clock Gating Instances       0    0.00                         0.00
--------------------------------------------------------------------------------
 RC Gated Flip-flops             68655   93.73                        94.87
 Non-RC Gated Flip-flops             0    0.00                         0.00
--------------------------------------------------------------------------------
 Total Gated Flip-flops          68655   93.73                            -
 Total Ungated Flip-flops         4591    6.27                            -
 Enable not found                 4371   95.21                            -
 Register bank width too small     220    4.79                            -
--------------------------------------------------------------------------------
 Total Flip-flops                73246  100.00                            -
--------------------------------------------------------------------------------
Multibit Flip-flop Summary
--------------------------
-------------------------------------------------------
Width     Number     Bits     RC Gated     Ungated
-------------------------------------------------------
1-bit     73246      73246    68655 (93.73%) 4591 (6.27%)
-------------------------------------------------------
```

Figure 4.7: Clock gating summary from the synthesis

The total area used by the clock gating instances is 18564 units. These area is equivalent to only 1.6% of the total area as can be seen in the figure 4.8

Type	Instances	Area	Area %
sequential	73793	609268.800	51.0
inverter	14043	41926.400	3.5
buffer	9444	37487.600	3.1
clock_gating_integrated_cell	1326	18564.000	1.6
logic	149333	487163.600	40.8
physical_cells	0	0.000	0.0
total	247939	1194410.400	100.0

Figure 4.8: Gates report

The figure 4.9 indicates an slack of 105870 ps.

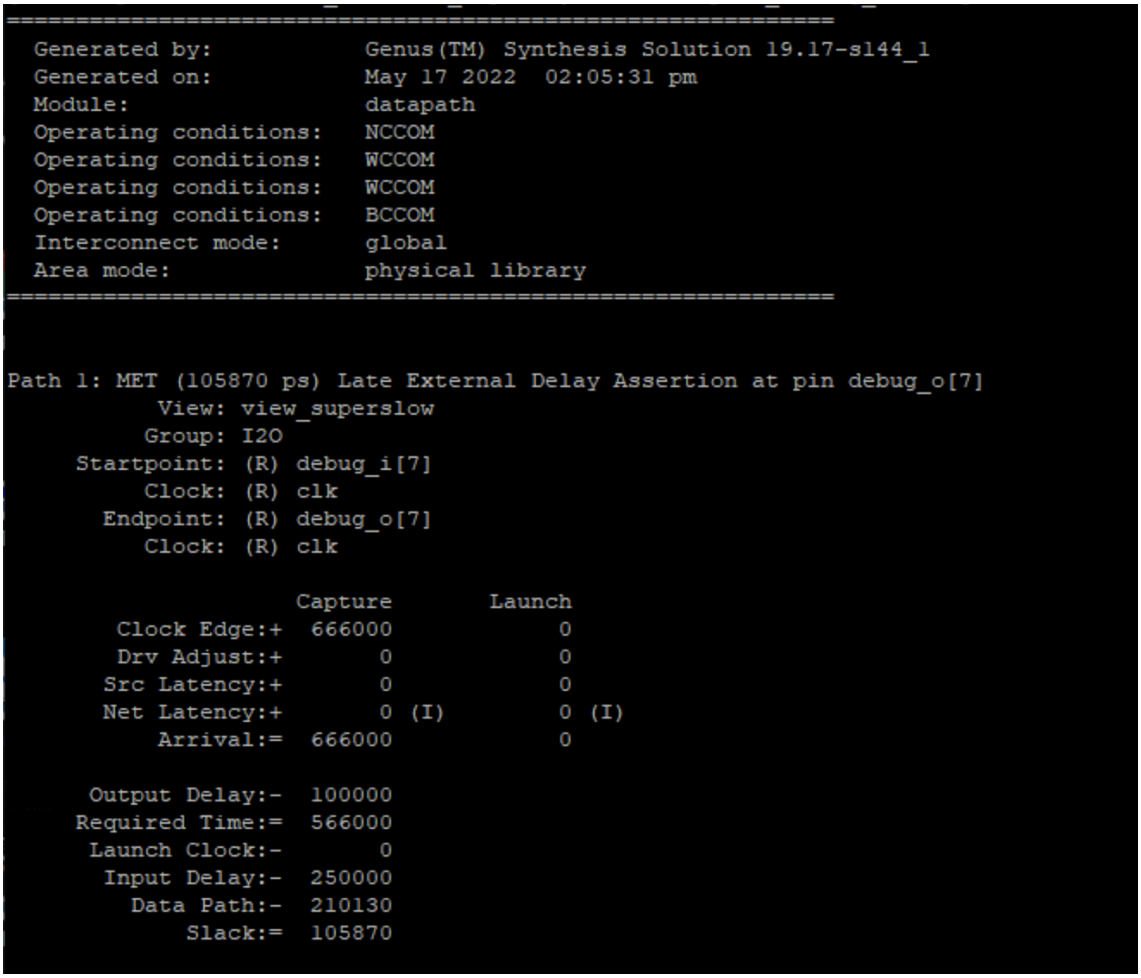

Figure 4.9: Timing report after the synthesis

Chapter 5

Results

First, the results are going to be presented by comparing the given numbers from the synthesis in Genus and then it is going to be presented the results from using a realistic load during the simulation in order to calculate the power consumption using Joules Power calculator.
The given numbers from Genus are the following in power, area, and timing.

Power consumption comparison

Power Unit: W	Datapath	Datapath with clock gating
Leakage	9.7203exp-05	6.97488exp-05
Internal	1.85exp-03	1.6063exp-03
Switching	5.29417exp-02	2.68706exp-02
Total	5.48978exp-02	2.85466exp-02

Figure 5.1: Power comparision on the datapath

As expected we can see that Genus did a power calculation with a lot of power saving on switching activity which is the one kind of power consumption that consumes more than any other. We can also note some savings on leakage and internal power consumption.

In picture 5.2 we can see all of the modules on the first level under datapath

hierarchy and its corresponding power consumption on the datapath without clock gating.

Power Unit: mW
PDB Frame : /stim#1/frame#0

	Cells	Pct_cells	Leakage	Internal	Switching	Total	Lvl	Instance
1	364466	100.00	1.23270e-1	9.55294e+1	1.24184e-1	9.57769e+1	0	/datapath
2	92093	25.27	3.47554e-2	2.57856e+1	1.93105e-2	2.58396e+1	1	/datapath/exe_stage_inst
3	86779	23.81	2.43140e-2	1.46260e+1	2.31536e-2	1.46734e+1	1	/datapath/graduation_list_inst
4	21094	5.79	8.59733e-3	1.19834e+1	8.17977e-3	1.20001e+1	1	/datapath/if_stage_1_inst
5	44547	12.22	1.44886e-2	1.09088e+1	6.59242e-3	1.09299e+1	1	/datapath/vregfile
6	12103	3.32	5.64360e-3	7.97309e+0	9.88271e-3	7.98862e+0	1	/datapath/instruction_queue_inst
7	24901	6.83	6.51487e-3	5.42723e+0	2.47591e-3	5.43622e+0	1	/datapath/regfile_fp_inst
8	23241	6.38	5.81043e-3	5.36764e+0	4.20323e-3	5.37765e+0	1	/datapath/regfile_inst
9	7935	2.18	5.36928e-3	2.31258e+0	7.08850e-4	2.31866e+0	1	/datapath/reg_exe_inst
10	7391	2.03	1.72522e-3	1.50059e+0	3.95768e-3	1.50628e+0	1	/datapath/simd_rename_table_inst
11	7650	2.10	1.76170e-3	1.49961e+0	4.56897e-3	1.50595e+0	1	/datapath/fp_rename_table_inst
12	11334	3.11	2.47287e-3	1.45881e+0	1.55313e-2	1.47682e+0	1	/datapath/rename_table_inst
13	2466	0.68	1.01048e-3	1.42198e+0	3.52464e-3	1.42652e+0	1	/datapath/reg_rr_inst
14	4658	1.28	1.28237e-3	1.37118e+0	2.05239e-3	1.37451e+0	1	/datapath/id_decode_inst
15	1076	0.30	3.91458e-4	6.35214e-1	1.12958e-3	6.36736e-1	1	/datapath/reg_rename_inst
16	972	0.27	3.54858e-4	5.33327e-1	2.10234e-3	5.35784e-1	1	/datapath/reg_ir_inst
17	912	0.25	3.20920e-4	5.32751e-1	5.22861e-4	5.33595e-1	1	/datapath/reg_id_inst
18	478	0.13	3.03892e-4	4.54912e-1	6.39820e-5	4.55280e-1	1	/datapath/commit_inst
19	739	0.20	2.84205e-4	3.28121e-1	1.31989e-3	3.29725e-1	1	/datapath/free_list_inst
20	729	0.20	2.40816e-4	3.27035e-1	2.56832e-4	3.27533e-1	1	/datapath/fp_free_list_inst
21	734	0.20	2.41717e-4	3.26911e-1	2.58675e-4	3.27411e-1	1	/datapath/simd_free_list_inst
22	597	0.16	2.23897e-4	2.76714e-1	6.86757e-4	2.77624e-1	1	/datapath/reg_if_2_inst
23	280	0.08	1.01166e-4	1.46723e-1	3.14550e-4	1.47139e-1	1	/datapath/reg_if_1_inst
24	165	0.05	4.08351e-5	4.56561e-2	8.77515e-5	4.57847e-2	1	/datapath/if_stage_2_inst
25	92	0.03	2.48942e-5	4.16898e-3	1.40148e-4	4.33402e-3	1	/datapath/control_unit_inst
26	452	0.12	3.84911e-4	1.92600e-4	1.27594e-4	7.05106e-4	1	/datapath/csr_interface_inst

Figure 5.2: Power consumption on the datapath design modules (first level)

The picture 5.3 displays the same modules but now with clock gating insertion during the synthesis. Note that we only have one clock gating instance in 5.3 because it is only displaying the modules on level 1 under the hierarchy of datapath (as it was seen earlier, the datapath module only uses one clock gating cell).

Power Unit: mW
PDB Frame : /stim#1/frame#0

	Cells	Pct_cells	Leakage	Internal	Switching	Total	Lvl	Instance
1	291094	100.00	9.09964e-2	1.69712e+1	1.79413e+0	1.88563e+1	0	/datapath
2	88203	30.30	2.81333e-2	4.77504e+0	5.10550e-1	5.31372e+0	1	/datapath/exe_stage_inst
3	3391	1.16	1.19270e-3	1.87412e+0	2.32129e-2	1.89853e+0	1	/datapath/reg_exe_inst
4	55063	18.92	1.57668e-2	1.35381e+0	2.89710e-1	1.65929e+0	1	/datapath/graduation_list_inst
5	2464	0.85	8.68091e-4	1.45387e+0	1.51352e-2	1.46987e+0	1	/datapath/reg_rr_inst
6	34237	11.76	9.31884e-3	6.60687e-1	1.67266e-1	8.37272e-1	1	/datapath/vregfile
7	3450	1.19	9.50863e-4	7.03493e-1	8.46676e-2	7.89112e-1	1	/datapath/id_decode_inst
8	20570	7.07	7.67364e-3	6.84414e-1	9.37622e-2	7.85850e-1	1	/datapath/if_stage_1_inst
9	1071	0.37	3.32998e-4	6.50134e-1	5.96987e-3	6.56437e-1	1	/datapath/reg_rename_inst
10	900	0.31	2.87538e-4	5.40359e-1	6.07874e-2	6.01434e-1	1	/datapath/reg_ir_inst
11	15800	5.43	3.99936e-3	4.68553e-1	8.94874e-2	5.62040e-1	1	/datapath/regfile_fp_inst
12	901	0.31	3.01043e-4	5.46422e-1	5.06870e-3	5.51792e-1	1	/datapath/reg_id_inst
13	425	0.15	2.00430e-4	4.62131e-1	5.08426e-2	5.13174e-1	1	/datapath/commit_inst
14	6471	2.22	1.60118e-3	4.29162e-1	4.30944e-2	4.73858e-1	1	/datapath/simd_rename_table_inst
15	6625	2.28	1.49175e-3	4.12491e-1	4.22245e-2	4.56207e-1	1	/datapath/fp_rename_table_inst
16	7364	2.53	1.64536e-3	4.03729e-1	4.48886e-2	4.50263e-1	1	/datapath/rename_table_inst
17	18482	6.35	5.04797e-3	2.94479e-1	9.03479e-2	3.89875e-1	1	/datapath/regfile_inst
18	11708	4.02	4.28587e-3	2.78069e-1	4.24480e-2	3.24802e-1	1	/datapath/instruction_queue_inst
19	343	0.12	1.40245e-4	1.13599e-1	1.40667e-2	1.27806e-1	1	/datapath/reg_if_2_inst
20	741	0.25	3.11056e-4	1.17989e-1	7.99082e-3	1.26291e-1	1	/datapath/simd_free_list_inst
21	743	0.26	3.05554e-4	1.17768e-1	7.95749e-3	1.26031e-1	1	/datapath/fp_free_list_inst
22	749	0.26	3.04274e-4	1.17705e-1	7.90934e-3	1.25919e-1	1	/datapath/free_list_inst
23	205	0.07	5.34657e-5	3.11902e-2	3.88385e-3	3.51275e-2	1	/datapath/if_stage_2_inst
24	712	0.24	6.10913e-4	1.88158e-2	4.13032e-3	2.35571e-2	1	/datapath/csr_interface_inst
25	323	0.11	1.97853e-4	1.42029e-2	2.40960e-3	1.68104e-2	1	/datapath/reg_if_1_inst
26	1	0.00	1.52746e-6	5.99459e-3	9.17562e-3	1.51717e-2	1	/datapath/RC_CG_HIER_INST0
27	88	0.03	2.95136e-5	5.06415e-3	4.49940e-4	5.54360e-3	1	/datapath/control_unit_inst

Figure 5.3: Power consumption on the datapath design modules (first level) with clock gating

Gates and area comparision

Datapath			
Gates type	**Instances**	**Area**	**Area %**
sequential	74,030	697,263.20	47.4
inverter	19,868	58,100.00	4.0
buffer	11,261	38,173.20	2.6
logic	221,792	677,140.80	46.0
physical_cells	0	0.00	0.0
total	326,951	1,470,677.20	100.0

Datapath with clock gating insertion			
Gates type	**Instances**	**Area**	**Area %**
sequential	73,793	609,235.20	51.1
inverter	13,291	37,643.20	3.2
buffer	9,933	38,415.20	3.2
clock_gating_integrated_cell	1,326	18,564.00	1.6
logic	149,344	486,582.00	40.9
physical_cells	0	0.00	0.0
total	247,687	1,190,439.60	100.0

Figure 5.4: Gate count and area comparision on the datapath

Although 1326 clock gating instances were inserted, at the end there was total of 247,687 instances used to build this datapath sub design. Also, there is a saving in area from 1,470,677.20 to 1,190,439.60 which completes a saving of 20 % approximately.
The reason why Genus uses less standard cells to build this design when inserting clock gating is because the design can save a lot of standard cells, the clock gating instances reduce all the standard cells used during the unnecessary feedback to registers. For example, in sub design vregfile, we can see how the clock gating instances gate the registers (flop standard cells) registers_reg[m][n], where m is a variable that ranges from 0 to 63, and n a variable from 0 to 123. As there is an instance of this standard cell for every bit, we have a total of 7936 instances (regarding to only these registers_reg[m][n]

instances) in this subdesign.
When there is no insertion of clock gating, we can see how the standard cells used during the synthesis to build the registers_reg[m][n] are the DFQD1HPBWP standard cell as can be seen in the figure 5.5. In this case we have registers_reg[1][1] and we can see how the output of this instance is connected to the net registers_[1][1] which is then feedback to the instance g630974 (standard cell A0222D1HPBWP).

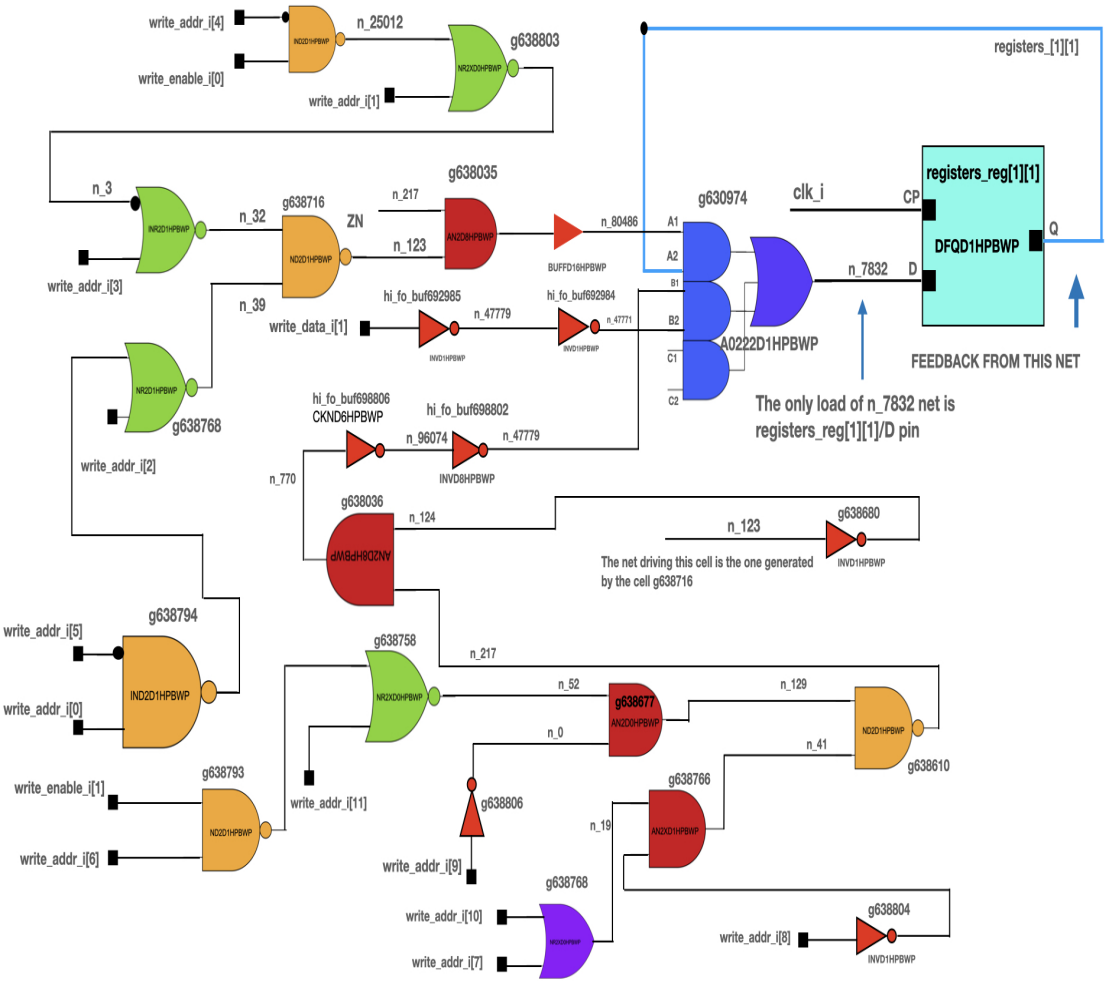

Figure 5.5: Schematic representation of instance registers_reg[1[1]

If we insert the clock gating using our standard cell CKLNQD16HPBWP, and compare the schematic around the registers[reg[1][1] instances until we reach the hports (write_addr_i[n], write_enable_i[n],write_data_i[n], clk_i) we will obtaing the schematic displayed in figure 5.6.

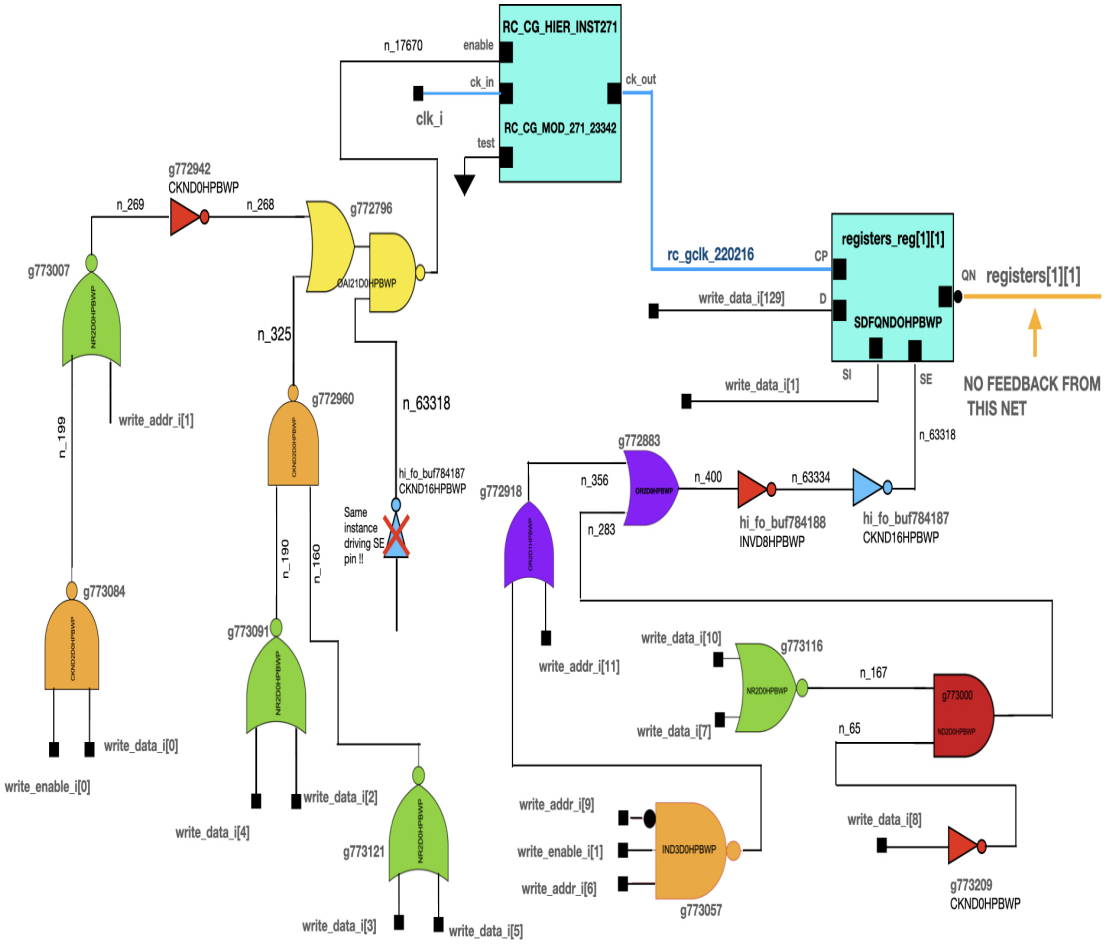

Figure 5.6: Schematic representation of instance registers_reg[1[1] (clock gating inserted)

As we can see in the implementation without clock gating in 5.5, there are more standard cells instantiated driving registers_reg[1][1] (even though we are not considering the standard cells instances driving pins C1 and C2) than there are driving the same instance registers[1][1] in the implemen-

tation inserting the clock gating. There are 23 standard cells driving the registers_reg[1][1] in 5.5 (only in the drawing as there are more standard cells but we are not counting the ones driving C1 and C2 pins) compared to the 16 standard cells driving the registers_reg[1][1] in the implementation inserting clock gating.

Besides the number of cells instantiated, we can see how in our synthesis results we have a smaller area in our design with clock gating insertion. This is not something that just results in because of using less standard cells as the cells also have an area. So, you can build a circuit with 10 cells having an area of 3 each one resulting in a total area of 30. And you can also build a circuit with 8 standard cells having an area of 4 resulting in a total area of 32. Taking the example of registers[1][1] we can note the following:

- Most of the standard cells that correspond to only a simple gate function (i.e and, nand, or, nor, inverters, buffers) have approximately the same area.
- The standards cells that correspond to complex logic functions, complex flops, and clock gating instances have bigger areas.

The results indicate that when no clock gating was inserted, the synthesis uses a less complex cell to build the registers[m][n], in this case it uses the standard cell DFQD1HPBWP, which is a d-flop. As previously said, the registers_reg[m][n] where m goes grom 0 to 63, and n from 0 to 127 gives a total of 8192 instances, one for each bit. The table indicatates that there are actually 8,192 instances of the DFQD1HPBWP only in the subdesign datapath/vregfile (corresponding to the instances registers_reg[m][n]). Further this cell has an area of 7.6 as can be seen in table 5.7.

datapath/vrgefile without clock gating (standard cells used to build registers_reg[m][n])			
Standard cell	**Area**	**Instances**	**Total Area**
DFQD1HPBWP	7.6	8,192.00	62,259.2

Figure 5.7: Data from stadard cells in registers_reg[m][n]

When the desing was synthesized using a clock gating cell there were 64 clock gating instances inserted in our subdesign datapath/vregfile/ as can be seen in figure 5.8. Besides, we got our registers_reg[m][n] instances built from 3 different standard cells:

- SDFQNDOHPBWP
- SDFQDOHPBWP
- DFXQD1HPBWP

datapath/vrgefile with clock gating (clock gating instances)			
Standard cell	**Area**	**Instances**	**Total Area**
CKLNQD12HPBWP cg cell	14	64.00	896.0

Figure 5.8: Data from clock gating instances (clock gating inserted)

but as we can see in the table shown in figure 5.9, almost all of these cells are instances from the SDFQNDOHPBWP standard cell which has an area of 9.2. It can be observed how the synthesis needed from bigger standard cells to instantiante the 8192 registers registers_reg[m][n] and an important quantity of total area just to instantiate these registers compared to the instantiation of these ones in the design without clock gating.

datapath/vrgefile with clock gating (standard cells used to build registers_reg[m][n])			
Standard cell	**Area**	**Instances**	**Total Area**
SDFQND0HPBWP	9.2	7,850.00	72,220.0
SDFQD0HPBWP	10	88.00	880.0
DFXQD1HPBWP	10	254.00	254.0
Total	NA	8,192.00	73,354.0

Figure 5.9: Data from stadard cells in registers_reg[m][n] (using clock gating)

Although it may seem that the design using clock gating during the synthesis should use a bigger area than the synthesis of the design without clock gating because of the bigger standard cells used to build the registers_reg[m][n] instances (SDFQNDOHPBWP, SDFQDOHPBWP, DFXQD1HPBWP) and the area needed by the 64 clock gating instances, there are 8201 instances from the standard cell AO222D1HPBWP (it is the cell driving the registers_reg[1][1] in figure 5.5 and it consists of 3 AND gates driving an OR gate) which has an area of 4.8 in the datapath/vregfile subdesign (one the most biggest standard cells in area), see table in figure 5.10. On the other hand, there is not a single instance of the A0222D1HPBWP standard cell when the clock gating was inserted (as can be seen in figure 5.6.

datapath/vrgefile without clock gating (standard cells used to drive the register_reg[m][n]			
Standard cell	**Area**	**Instances**	**Total Area**
A0222D1HPBWP	4.8	8,201.00	39,364.8

Figure 5.10: Standard cells driving the registers_reg[m][n] when there is no clock gating insertion

The results let us conclude that as in the work executed by [2], when the design is synthesized using clock gating the synthesis tool can find opportunities where it can save instances from standard cells as shown in figure 5.6. Furthermore, these savings can be from standard cells that consume a lot of area as the AO222D1HPBWP standard cell (compare fig 5.5 and fig 5.6.

5.1 Power calculation using realistic loads

In this section, it is presented the power calculation results obtained when using the flow netlist flow with gate stimulus (3.13).
As a first demonstration, it was generated a vcd stimulus file to evaluate the power consumption over time (only vcd and shm formats have this option). The picture 5.11 illustrates the power consumption over time in our datapath instance when executing the isa test rv64ui-p-add. Our power plots throughout the approximate 220 us that lasts our stimulus file let us inspect our results.

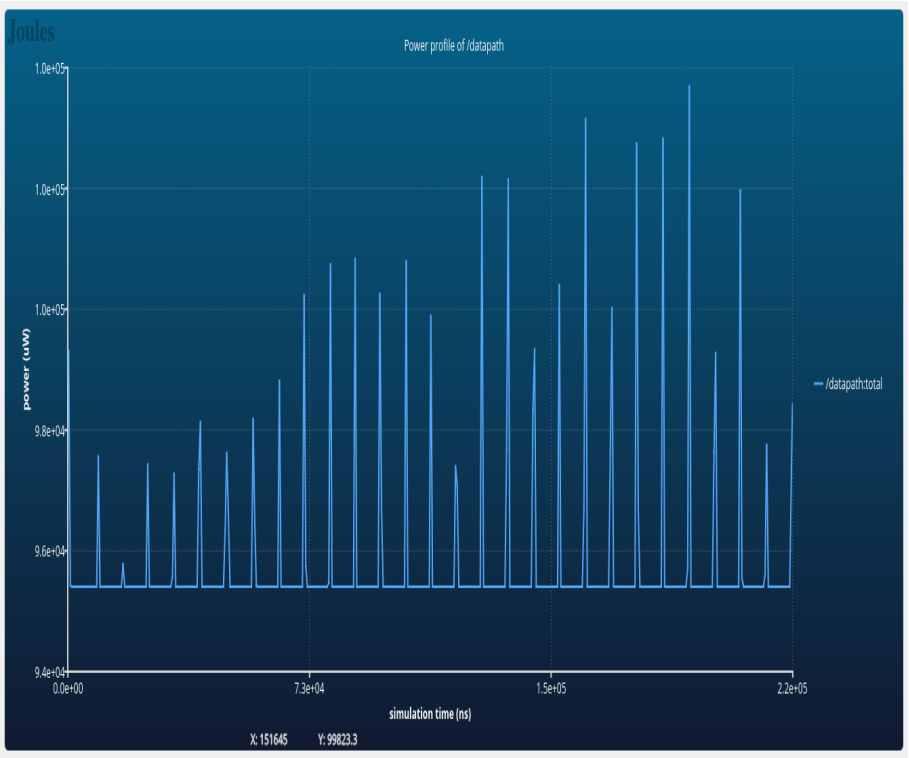

Figure 5.11: Power plot of our datapath inst without clock gating

Now, when we plot the power of the datapath inst that have clock gating inserted using the gate level stimulus from our datapath with clock gating inserted we can see how the total power is reduced over time in (fig 5.12). In this test we can see how most of the time the power consumption on

the design without clock gating is around 100mw and when the design has the clock gating inserted most of the time the power consumption is around 14,000 and 16,000 uW (14-16 mW).

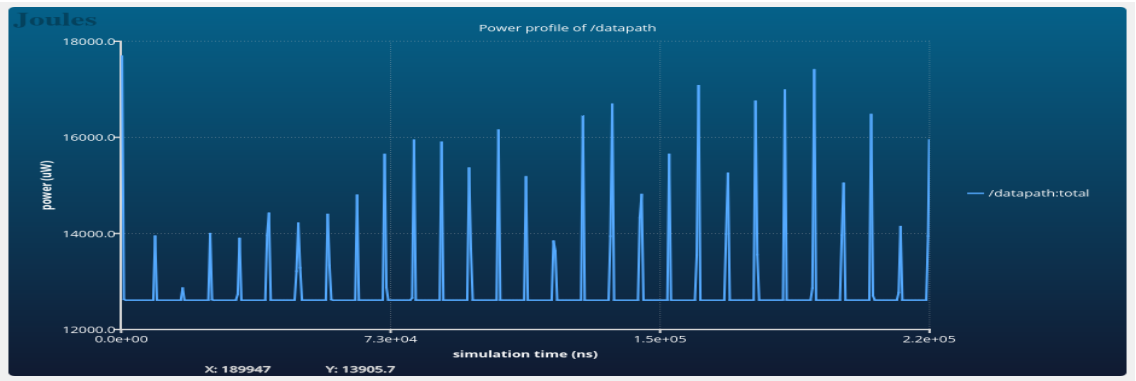

Figure 5.12: Power plot of our datapath with clock gating

The next table shows the ISA tests that were executed in order to evaluate our power performance. We can see how the clock gating insertion reduced the power for about 85%. For these tests, the stimulus files were generated in TCF format.

Power consumption comparison in the datapath sub design with/without clock gating			
Simulation case	**no clock gating (mW)**	**with clock gating (mW)**	**Power saving %**
add	95.75	12.82	86.00
beq	95.81	12.87	86.00
bne	95.78	12.84	86.00
lw	95.82	12.82	86.00
median	101.60	15.60	85.00
towers	101.68	15.69	85.00
histogram	100.78	15.08	85.00

Figure 5.13: Power comparision running some ISA tests

Here is a brief explanation of every test:

- add: Addition isa test
- beq: Branch if equal isa test
- bne: Branch if not equal isa test
- lw: Load word isa test (one of the most expensive isa test in terms of performance)
- median: This benchmark performs a 1D three element median filter
- towers: Towers of Hanoi is a classic puzzle problem. The game consists of three pegs and a set of discs. Each disc is a different size, and initially all of the discs are on the left most peg with the smallest disc on top and the largest disc on the bottom. The goal is to move all of the discs onto the right most peg. The catch is that you are only allowed to move one disc at a time and you can never place a larger disc on top of a smaller disc
- histogram: histogram risc-v benchmark.

Power analysis requieres high numbers in annotation reports. Our annotation report (which was generated using a gate stimulus) indicates a 100% of annotated ports, and the same for the sequential outputs. In the case of the annotation report from the design with clock gating, its report indicate also 100% of annotation (our 1326 instances) in the Total ICGC. See the following tables:

```
Stim Id                  : /stim#1
Stim file                : /home/asilva/respaldo_servidor_viejo/asilva/sargantana/pruebas_finales/gate_stimulus/nocg/datapath_nocg_median.tcf
Stim file format         : tcf
Top instance             : /tb/DUT/Top_asic_inst/Rocket/RocketTile/core/datapath_inst
Design Top               : /datapath
Num frames               : 1
Duration                 : 3.04852e+06ns
------------------------------------------------ Annotation Report -------------------------------------------------
Object Type         Asserted  User_Asserted  Default  Computed  Clock_Source  Unconnected   Total  Asserted%
                                                                              + Constant
--------------------------------------------------------------------------------------------------------------------
Primary Ports
  Inputs                753              0        0         0             0            0     753    100.00%
  Outputs               563              0        0         0             0          241     804    100.00%
  I/O                     0              0        0         0             0            0       0        N/A
Sequential Outputs
  Memory                  0              0        0         0             0            0       0        N/A
  Flop                76992              0        0         0             0            0   76992    100.00%
  Latch                 547              0        0         0             0            0     547    100.00%
  Arch ICGC               0              0        0         0             0            0       0        N/A
  Inferred ICGC           0              0        0         0             0            0       0        N/A
  Total ICGC              0              0        0         0             0            0       0        N/A
Drivers
  Driver nets        324410              0        0     49054             0            0  373464     86.86%
  RTL Driver nets    312796              0        0         0             0            0  312796    100.00%
DFT
  Input Ports             0              0        0         0             0            0       0        N/A
  Flop Outputs            0              0        0         0             0            0       0        N/A
  Memory Outputs          0              0        0         0             0            0       0        N/A
--------------------------------------------------------------------------------------------------------------------
```

Figure 5.14: Annotation report from the datapath without clock gating

```
Stim Id                  : /stim#1
Stim file                : /home/asilva/respaldo_servidor_viejo/asilva/sargantana/pruebas_finales/gate_stimulus/cg/datapath_cg_median.tcf
Stim file format         : tcf
Top instance             : /tb/DUT/Top_asic_inst/Rocket/RocketTile/core/datapath_inst
Design Top               : /datapath
Num frames               : 1
Duration                 : 3.04852e+06ns
------------------------------------------------ Annotation Report -------------------------------------------------
Object Type         Asserted  User_Asserted  Default  Computed  Clock_Source  Unconnected   Total  Asserted%
                                                                              + Constant
--------------------------------------------------------------------------------------------------------------------
Primary Ports
  Inputs                753              0        0         0             0            0     753    100.00%
  Outputs               563              0        0         0             0          241     804    100.00%
  I/O                     0              0        0         0             0            0       0        N/A
Sequential Outputs
  Memory                  0              0        0         0             0            0       0        N/A
  Flop                75455              0        0         0             0            0   75455    100.00%
  Latch                 547              0        0         0             0            0     547    100.00%
  Arch ICGC               0              0        0         0             0            0       0        N/A
  Inferred ICGC        1326              0        0         0             0            0    1326    100.00%
  Total ICGC           1326              0        0         0             0            0    1326    100.00%
Drivers
  Driver nets        260243              0        0     38634             0            0  298877     87.07%
  RTL Driver nets    254841              0        0         0             0            0  254841    100.00%
DFT
  Input Ports             0              0        0         0             0            0       0        N/A
  Flop Outputs            0              0        0         0             0            0       0        N/A
  Memory Outputs          0              0        0         0             0            0       0        N/A
--------------------------------------------------------------------------------------------------------------------
```

Figure 5.15: Annotation report from the datapath with clock gating

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The development of this work was a very rich experience in understanding the flow that needs to be followed to take your RTL design into a mapped Gate Level Netlist. One of the main challenges is that you need experience using the EDA tools because of all the technical issues that may arise during the work flow. Furthermore, analysing all of the aspects of a big design as the one we used in this thesis is a very challenging task and it should be addressed by a group of people because it becomes a very convoluted analysis. On the other hand, we were able to analyse the effects of adding clock gating to our design. The effect of adding clock gating did improve the area used for the design and the timing was still very close to the timing without clock gating. The power performance was improved about savings as high as these ones. However, it should be noted again that the analysis was only executed on the datapath block (and all of its sub hierarchies) and there are other blocks in the SoC that even with clock gating, could not be able to achieve savings as the datapath did , for example the memory SRAM blocks. Although the clock gating is a very powerful power saving technique that can be integrated through our synthesis tools since at least 15 years, this work has permitted to gain this experience in how to implement it and all of the effects that should be analysed when we implement it. Lastly, this work made us improve our knowledge to a group of people that will continue working in a lot of subjects related to these EDA tools.

6.2 Future work

The work flows used during this thesis were just the first part in order to produce the GDS files which are the ones that are sent to the Semiconductor Foundry companies so the CIC-IPN can no longer depend on other groups to generate the physical layout of the Gate-Level Netlist. At the same time, we need to enhance the knowledge on all the EDA tools and all theoretical background we need to have in order to have better understanding of all the effects of implementing our design into physical chips. Lastly, we need to create a new group of students and researchers to start to tackle the design of analog, digital, and mixed-signal designs.

Bibliography

[1] R.-v. foundation, "riscv-tests repository.". URL `https://github.com/riscv-software-src/riscv-tests/tree/master/isa/rv64ui`.

[2] R. Bhutada and Y. Manoli. Complex clock gating with integrated clock gating logic cell. In *2007 international conference on design & technology of integrated systems in nanoscale era*, pages 164–169. IEEE, 2007.

[3] Cadence. *Genus Low Power Guide*, .

[4] Cadence. *Genus User Guide*, .

[5] Cadence. *Joules User Guide*, .

[6] Cadence. *Power Analysis Flow from RTL Power through Signoff Power with Joules and Voltus*, .

[7] Cadence. *Xcelium Simulator Tcl Command Reference*, .

[8] D. MacMillen, R. Camposano, D. Hill, and T. W. Williams. An industrial view of electronic design automation. *IEEE transactions on computer-aided design of integrated circuits and systems*, 19(12):1428–1448, 2000.

[9] D. A. Patterson and J. L. Hennessy. *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[10] C. Ramírez, C. Hernández, C. R. Morales, G. M. García, L. A. Villa, and M. A. Ramírez. Lagarto i-una plataforma hardware/software de arquitectura de computadoras para la academia e investigación. *Res. Comput. Sci.*, 137:19–28, 2017.

[11] C. Rojas Morales et al. From fpga to asic: A risc-v processor experience. 2019.

[12] M. Saint-Laurent and A. Datta. A low-power clock gating cell optimized for low-voltage operation in a 45-nm technology. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 159–163. IEEE, 2010.

[13] D. K. Sharma. Effects of different clock gating techinques on design. *International Journal of Scientific & Engineering Research*, 3(5):1–4, 2012.

[14] TSMC. *TCBN65LPHPBWPCG TSMC N65LP Standard Cell Library Datasheet.*

[15] N. H. Weste and D. Harris. *CMOS VLSI design: a circuits and systems perspective.* Pearson Education India, 2015.

Chapter 7

Appendix

Synthesis scripts (setup_run.tcl, power_run.tcl, run.tcl) with no mmmc flow.

```
##############################################
#               MAIN SETUP (root attributes & setup
    variables)
##############################################
##############################################
## Preset global variables and attributes
##############################################

set DESIGN datapath
set GEN_EFF medium
set MAP_OPT_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T
    "]
set _OUTPUTS_PATH outputs
set _REPORTS_PATH reports_${DATE}
set _LOG_PATH logs_${DATE}
##set MODUS_WORKDIR <MODUS work directory>
set_db / .init_lib_search_path {. /usr/local/TSMC/
    stclib/10-track/tcbn65lphpbwp-set/
    tcbn65lphpbwp_200a_FE/TSMCHOME/digital/Front_End/
    timing_power_noise/NLDM/tcbn65lphpbwp_140a/}
set_db / .script_search_path {.}
set_db / .init_hdl_search_path {.}
set_db / .information_level 7

##generates <signal>_reg[<bit_width>] format
```

```
set_db / .hdl_array_naming_style %s\[%d\]


################################
## Library setup
################################


read_libs tcbn65lphpbwptc.lib
read_physical -lef /usr/local/TSMC/stclib/10-track/
   tcbn65lphpbwp-set/tcbn65lphpbwp_200a_FE/TSMCHOME/
   digital/Back_End/lef/tcbn65lphpbwp_140a/lef/
   tcbn65lphpbwp_9lmT2.lef

set_db / .lp_insert_clock_gating true

set_db / .leakage_power_effort medium

## The following line is to avoid errors on complex
   ports [][]
set_db hdl_flatten_complex_port true
```

Listing 7.1: setup_run.tcl

```
################################
#            LOW POWER setup (Leakage/Dynamic power/
   Clock Gating setup)                    #
################################
################################
## Leakage/Dynamic power/Clock Gating setup.
################################

set_db "design:$DESIGN" .lp_clock_gating_cell
   lib_cell:default_emulate_libset_max/tcbn65lphpbwptc/
   CKLNQD12HPBWP

set_db leakage_power_effort low

#the next line is to exclude negative edge triggered
```

```
   flops (if any)
set_db [get_db insts -if {.is_flop==true &&
   .negative_edge_clock==true}]
   .lp_clock_gating_exclude true

#next line is exclude flops without set/reset signal
set_db lp_clock_gating_infer_enable
   set_reset_flops_only
```

Listing 7.2: power_run.tcl

```
#### Template Script for RTL->Gate-Level Flow (
   generated from GENUS 19.17-s144_1)

if {[file exists /proc/cpuinfo]} {
  sh grep "model name" /proc/cpuinfo
  sh grep "cpu MHz"    /proc/cpuinfo
}

puts "Hostname : [info hostname]"

######################
## Load Design
######################

## Including setup file (root attributes & setup
   variables).
include setup_run.tcl

read_hdl -sv -f ./genus_build/src_file_sv
elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl'"
time_info Elaboration

write_db ${_OUTPUTS_PATH}/${DESIGN}_elab_cg.db

check_design -unresolved

######################
```

```
## Constraints Setup
#####################

read_sdc ./Constraints/LAGARTO_SOC.sdc

if {![file exists ${_LOG_PATH}]} {
  file mkdir ${_LOG_PATH}
  puts "Creating directory ${_LOG_PATH}"
}

if {![file exists ${_OUTPUTS_PATH}]} {
  file mkdir ${_OUTPUTS_PATH}
  puts "Creating directory ${_OUTPUTS_PATH}"
}

if {![file exists ${_REPORTS_PATH}]} {
  file mkdir ${_REPORTS_PATH}
  puts "Creating directory ${_REPORTS_PATH}"
}
check_timing_intent

## Including LOW POWER setup. (Leakage/Dynamic power/
   Clock Gating setup)
include power_run.tcl

###########################################
## Synthesizing to generic
###########################################

set_db / .syn_generic_effort $GEN_EFF
syn_generic
puts "Runtime & Memory after 'syn_generic'"
time_info GENERIC
write_snapshot -outdir $_REPORTS_PATH -tag generic
report_summary -directory $_REPORTS_PATH

##############################
## Synthesizing to gates
##############################

set_db / .syn_map_effort $MAP_OPT_EFF
```

```
syn_map
puts "Runtime & Memory after 'syn_map'"
time_info MAPPED
write_snapshot -outdir $_REPORTS_PATH -tag map
report_summary -directory $_REPORTS_PATH

write_do_lec -revised_design fv_map -logfile ${
   _LOG_PATH}/rtl2intermediate.lec.log > ${
   _OUTPUTS_PATH}/rtl2intermediate.lec.do

##########################################
## Optimize Netlist
##########################################

set_db / .syn_opt_effort $MAP_OPT_EFF
syn_opt
write_snapshot -outdir $_REPORTS_PATH -tag syn_opt
report_summary -directory $_REPORTS_PATH
puts "Runtime & Memory after 'syn_opt'"
time_info OPT

##########################################
## write backend file set (verilog, SDC, config, etc.)
##########################################

write_sdf >  ${_OUTPUTS_PATH}/${DESIGN}_cg_sdf.sdf
write_db -to_file ${_OUTPUTS_PATH}/${DESIGN}_cg_opt.db
write_hdl > ${_OUTPUTS_PATH}/${DESIGN}_cg_netlist.v
write_design -base_name  ${DESIGN}_cg
write_sdc > ${_OUTPUTS_PATH}/${DESIGN}_m.sdc

##########################################
## Reports
##########################################

report_clock_gating > $_REPORTS_PATH/${DESIGN}
   _clock_gating.rpt
report_clock_gating -gated > $_REPORTS_PATH/${DESIGN}
   _cg_gated.rpt
report_clock_gating -ungated > $_REPORTS_PATH/${DESIGN}
   _cg_ungated.rpt
```

```
report_clock_gating -detail > $_REPORTS_PATH/${DESIGN}
    _cg_detail.rpt

report_power  > $_REPORTS_PATH/${DESIGN}_power_cg.rpt
report_gates -power > $_REPORTS_PATH/${DESIGN}
    _gates_power_cg.rpt
report_timing > $_REPORTS_PATH/${DESIGN}
    _timing_report_cg.rpt
report_area > $_REPORTS_PATH/${DESIGN}
    _area_report_cg.rpt

#####################################
### write_do_lec
#####################################


write_do_lec -golden_design fv_map -revised_design ${
    _OUTPUTS_PATH}/${DESIGN}_m.v -logfile ${_LOG_PATH}/
    intermediate2final.lec.log > ${_OUTPUTS_PATH}/
    intermediate2final.lec.do
##Uncomment if the RTL is to be compared with the final
     netlist..
##write_do_lec -revised_design ${_OUTPUTS_PATH}/${
    DESIGN}_m.v -logfile ${_LOG_PATH}/rtl2final.lec.log
    > ${_OUTPUTS_PATH}/rtl2final.lec.do

puts "Final Runtime & Memory."
time_info FINAL
puts "================================"
puts "Synthesis Finished ........."
puts "================================"
```

Listing 7.3: run.tcl

Joules script to generate power calculations.

```
set isa add
set formato vcd
set DESIGN datapath
set GEN_EFF medium
set MAP_OPT_EFF high
set DATE [clock format [clock seconds] -format "%b%d-%T
   "]
set _OUTPUTS_PATH outputs_${DATE}_${isa}_cg_gate
set _REPORTS_PATH reports_${DATE}_${isa}_cg_gate
set _LOG_PATH logs_${DATE}_${isa}_cg_gate

set_db init_lib_search_path /usr/local/TSMC/stclib/10
   -track/tcbn65lphpbwp-set/tcbn65lphpbwp_200a_FE/
   TSMCHOME/digital/Front_End/timing_power_noise/NLDM/
   tcbn65lphpbwp_140a/

read_libs tcbn65lphpbwptc.lib

read_netlist /home/asilva/respaldo_servidor_viejo/
   asilva/sargantana/pruebas_finales/netlists/cg/
   datapath_netlist.v

# Read stimulus with -resim_cg_enables option
read_stimulus /home/asilva/respaldo_servidor_viejo/
   asilva/sargantana/pruebas_finales/gate_stimulus/cg/
   datapath_cg_${isa}.${formato} -dut_instance
   tb.DUT.Top_asic_inst.Rocket.RocketTile.core.datapath_inst
    -interval_size 500ns -resim_cg_enables  -sdb_out
   stim_${isa}_gate.sdb

# Re-read SDB and Compute Power
read_stimulus -file stim_${isa}_gate.sdb

estimate_data_buffers
gen_clock_tree
compute_power -mode time_based

write_db ${_OUTPUTS_PATH}/DB/pwr_db_cg_${isa}.jdb
```

```
report_power > ${_OUTPUTS_PATH}/${DESIGN}_power_cg_${
    isa}_rtl.rpt
report_power -by_hierarchy > ${_OUTPUTS_PATH}/${DESIGN}
    _power_cg_${isa}_hier.rpt

#Report annotation
report_sdb_annotation > ${_OUTPUTS_PATH}/${DESIGN}_cg_$
    {isa}_sdb_annotation.rpt

#Plot power profile
plot_power_profile
```

Listing 7.4: joules_cg.*tcl*