Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Daniel Eduardo Guarecuco Aguiar

Multi-dimensional exploration of the Minimum Energy Point for RISC-V subsystem in 22 nm FDSOI

Master's thesis in Embedded Computing Systems
Supervisor: Snorre Aunet
Co-supervisor: Torbjørn Ness
June 2023

NTNU
Norwegian University of
Science and Technology

Daniel Eduardo Guarecuco Aguiar

Multi-dimensional exploration of the Minimum Energy Point for RISC-V subsystem in 22 nm FDSOI

Master's thesis in Embedded Computing Systems
Supervisor: Snorre Aunet
Co-supervisor: Torbjørn Ness
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

ABSTRACT

Battery operated devices are highly constraint in energy budget, there is a need for energy-efficient designs. Designing circuits to operate in the Minimum Energy Point (MEP) yields great energy savings, by at least 20%, by reducing supply voltage and operating frequency. The MEP is in the near or sub-threshold region, such that the circuit becomes very sensitive to PVT variations.

This thesis proposes a methodology to study and locate the MEP and its relationship with supply voltage, threshold voltage, channel length, body biasing, temperature and process variations, for a RISC-V Ibex CPU synthesized with standard cell libraries characterized for higher voltages with a commercially available 22 nm FDSOI technology. A critical path replica is converted into a ring oscillator for running SPICE simulations. The simulations are used to estimate the MEP of the CPU.

Results show that HVT and UHVT libraries are unfit for operating at the MEP due to a huge gap between slow and fast corners. It is shown that SLVT provides the best timing closure. The selection of threshold voltages becomes the most important parameter, adjustments can be made later by selecting a proper channel length and body-biasing voltages.

PREFACE

This thesis is the final part of my double-degree in the *European Master in Embedded Computing Systems* (EMECS or MSECS) at the Department of Electronic Systems, at the Norwegian University of Science and Technology in Trondheim, and the Department of Control and Computer Engineering at the *Politecnico Di Torino* in Italy.

First and foremost, I am immensely grateful to my academic supervisor, Snorre Aunet, to all the great people at Nordic Semiconductor: Torbjørn Ness, Anders Vinje, and Frode Pedersen, for their invaluable guidance, expertise, and continuous support throughout the entire research process. Their insightful feedback shown on every weekly meeting have played a pivotal role in shaping the direction of this thesis.

Likewise, I am indebted to the EMECS program for the amazing and enriching experience I had in these two beautiful countries. Both universities with a lot to offer in and outside the classrooms. A shout-out to all the faculty members, and fellow students who have been part of this journey.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **ABB** Adaptive Body-Bias
- **CMOS** Complementary Metal–Oxide–Semiconductor
- **DUT** Design Under Test
- **FBB** Forward Body-Bias
- **FDSOI** Fully Depleted Silicon-On-Insulator
- **FF** Fast-Fast
- **HVT** High Threshold Voltage
- **LVT** Low Threshold Voltage
- **MEP** Minimum Energy Point
- **NTNU** Norwegian University of Science and Technology
- **PVT** Process, Voltage and Temperature
- **RBB** Reverse Body-Bias
- **RO** Ring Oscillator
- **SOTB** Silicon on Thin-BOX
- **SLVT** Super-Low Threshold Voltage
- **SS** Slow-Slow
- **TT** Typical-Typical
- **UHVT** Ultra-High Threshold Voltage
- **VT** Threshold Voltage

CHAPTER

ONE

INTRODUCTION

In recent years, the field of embedded systems has experienced rapid development and has become an integral part of our daily lives. These are specialized computer systems designed to perform specific tasks in larger systems or devices [1]. They are found in many applications, including consumer electronics (such as smartphones, tablets, and smartwatches), automotive systems, medical devices, industrial control systems, and Internet of Things (IoT) devices. As the demand for these applications continues to grow, so does the need for efficient power utilization.

Most microprocessors ever produced go into the field of embedded systems [1]. It has been estimated that there are about 16 billion mobile devices (phones and tables) for the year 2023 [2],[3]. Other report forecasts 15 billion connected IoT devices for this year [4]. It is difficult to pinpoint the exact number of embedded systems currently available due to factors such as the variety of applications and industries they serve. In addition, the definition of an embedded system can vary depending scope of the classification.

One of the key challenges for embedded system designers is power consumption. These systems are typically powered by batteries or other limited power sources, as energy harvesting technologies. Their power requirements have a direct impact on their functionality, reliability, and lifespan. In addition, many embedded systems are deployed in remote or inaccessible locations, making it difficult or expensive to replace or frequently recharge their power supplies. Therefore, optimizing power consumption is extremely important to ensure sustainable systems.

To solve these power consumption challenges, various methods have been developed and implemented. These methods aim to reduce the energy consumption at various levels, including hardware design, software optimization, and circuit

level. From an architectural perspective, RISC processors are increasingly used in this domain due to their low power consumption and computing power. For low power requirements, RISC is most suited that CISC architectures [5]. A promising development in this field is the RISC-V CPU architecture. Which provides an open-source instruction set architecture (ISA) that is simple, flexible, modular and extensible [6].

Meanwhile, from a circuit-level perspective, voltage scaling is a known method to significantly reduce power and energy consumption, because of its square relationship with supply voltage [7]. There is an optimum supply voltage that yields the most energy-efficient operation, called the Minimum Energy Point (MEP), which is located in the sub-threshold region, and it is depend on the activity factor and threshold variations [8]. This region is ideal for applications where low energy is the primary focus instead of performance.

1.1 Objectives

The aim of the thesis is to further research voltage scaling techniques in the near and sub-threshold regions to find the Minimum Energy Point of a RISC-V CPU subsystem. Many parameters are taken into considerations, such as the impact of different threshold voltages, different transistors' channel length, as well as process and temperature variations. The design is synthesised in a commercially available 22 nm FDSOI technology with *Synopsys Design Compiler*. Analog simulations are run with *Cadence Virtuoso* in order to vary PVT conditions.

1.2 Organization

The report is structured into four main chapters: theory, methods, results and discussion. The theory chapter provides relevant basic theory that is important to the thesis. The methods describes the techniques and tools used to generate the data and results. The results extracted from the methods section are later described. Finally, the discussion chapter serves as a platform for analysis and comparison of the results with the theory.

CHAPTER

TWO

THEORY

This chapter gives an introduction to the theory supporting this study and some terms used in later chapters. First, it is important to understand how power is consumed in digital circuits and its relationship with energy. Later a common power management technique is introduced. Finally, a summary of what has been done in the field of Minimum Energy Point is given.

2.1 Power and energy

To fully understand the concept behind MEP it is important to have some basic background knowledge in how energy is utilized in digital CMOS circuits, so that low power optimizations can be targeted.

In an electric circuit, *power* P is the rate at which energy is transferred from the power source (e.g., battery) to an electrical element [9]. The power consumed at a given time is given by the product of the current through and the voltage across such element [7], see Equation 2.1.

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

The *energy* E consumed over a time interval is given by the accumulated power [7], see Equation 2.2.

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

In a digital circuit, if the average power is evaluated in a clock period, the previous expression can be simplified as Equation 2.3, or 2.4 if the frequency is used.

$$E = P_{avg} * T_{clk} \tag{2.3}$$

$$E = \frac{P_{avg}}{f_{clk}} \tag{2.4}$$

2.1.1 Sources of power dissipation

It is of particular interest to study the sources of power dissipation, which for CMOS technology can be divided mainly in two categories: the *dynamic* and *static* power.

The *dynamic power* refers to the power consumed by a transistor when switching states. Its dissipation is due mainly to the charging and discharging of load capacitances, this is also know as *switching power*. There is a second component to the dynamic power consumption, the *short-circuit* power, which is due to a direct short-circuit path between VDD and GND through the NMOS and PMOS transistors as they are both partially active. Since the short-circuit contribution is typically very small compared to the switching power, it can often be neglected. So that the dynamic power can be expressed as the contribution of the switching power, described by Equation 2.5 [7], [10].

$$P_{dynamic} \approx P_{switching} = \alpha C f_{clk} V_{DD}^2 \tag{2.5}$$

Where α refers to the activity factor, or the probability that a circuit node transitions from low to high. The term C represents the loading capacitance. In this study, the terms dynamic power and switching power might be use interchangeably. Figure 2.1.1 illustrates how the switching energy is consumed in an inverter.

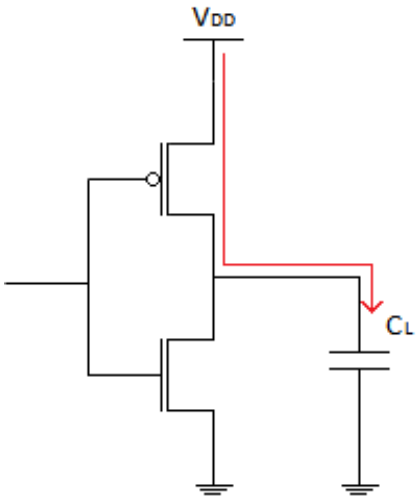


Figure 2.1.1: Dynamic power dissipated by an Inverter.

The *static power* is the power consumed by the circuit irrespective of the circuit state, even if it is not switching. This power loss is due to the non-ideal conditions of real transistors, which do not behave completely as ON/FF switches. The static power has multiple contributors, but the dominant source in modern CMOS technology is the *subthreshold leakage*. The static power is characterized by equation 2.6 and 2.7 [11]. This leakage is caused by a current flowing through a transistor that is meant to be OFF, but it is not due to reduced threshold voltages [12]. In this study, the terms static power and leakage power might be use interchangeably.

$$P_{static} \approx P_{subth} = I_{subth} V_{DD} \tag{2.6}$$

$$I_{subth} = \mu_N C_{ox} \frac{W_N}{L_N} V_t^2 e^{\frac{V_{GS}-V_{th}}{nV_t}} (1 - e^{-\frac{V_{DS}}{V_t}}) \tag{2.7}$$

Where μ_N is the electron carrier mobility, C_{ox} is the gate capacitance per unit area, W_N is the channel width, L_N is the channel length, V_t is the thermal voltage, V_{th} is the threshold voltage and n is the inverse slope of the subthreshold current [11]. Figure 2.1.2 shows how this type of energy is dissipated in an inverter.

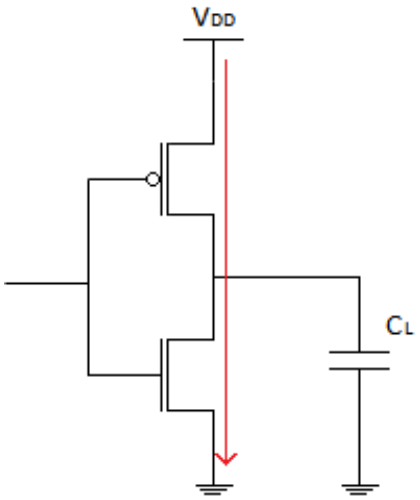


Figure 2.1.2: Static power dissipated by an Inverter.

Putting together the two main sources of power dissipation, Equations 2.8 and 2.9 are obtained.

$$P_{total} = P_{dynamic} + P_{static} \tag{2.8}$$

$$P_{total} = \alpha C f_{clk} V_{DD}^2 + I_{leakage} V_{DD} \tag{2.9}$$

In terms of energy:

$$E_{total} = E_{switching} + E_{leakage} \tag{2.10}$$

$$E_{total} = \alpha C V_{DD}^2 + \frac{I_{leakage} V_{DD}}{f} \tag{2.11}$$

2.2 Dynamic Voltage and Frequency Scaling (DVFS)

According to [13], there are two successful methods for reducing the dynamic power, which becomes evident from Equation 2.5: activity (α) reduction and supply voltage (VDD) reduction. Due to the quadratic relationship between power and supply voltage, reducing this term can lead to significant power savings. There are two approaches to achieving supply voltage reduction while maintaining performance: static and dynamic methods.

In the *static supply voltage reduction* method, multiple supply voltages are utilized withing the same circuit. The higher supply voltage is employed for performance-critical logic that requires higher speeds but consumes more power,

and low supply voltages are used for non-critical paths, with lower speeds but dissipating less power.

Alternatively, in *dynamic supply voltage scaling*, the logic chip is designed to operate at its maximum performance level when supplied with the highest voltage. However, when the performance demand is low, the chip can be operated at a lower voltage, resulting in reduced performance but substantial power reduction.

This second method is often called *Dynamic Voltage and Frequency Scaling* (DVFS), as the frequency often must scale down along with the supply voltage in order to guarantee correct operation in synchronous systems [10]. Evidence supports the effectiveness of this technique, for instance, the study carried by [10] on a 9-processor JPEG application, using 1.3 V and 0.8 V for voltage scaling, achieved a reduction of 48% energy consumption, while only diminishing performance by 8%, with an additional 12% area overhead for the DVFS circuit.

As voltage and frequency are both reduced, it can be seen from Equation 2.11 an opposing trend between the switching and leakage component of the energy. The switching energy term is always decreasing by a square relationship on VDD, while the leakage term is increasing due to slower operation times. At low supply voltages, leakage energy takes dominance. Their balance define the MEP location.

Implementing a DVFS systems in a general-purpose microprocessor involves three essential components: 1) An intelligent operating system capable of dynamically adjusting the processor speed. 2) A regulation loop that generates the minimum required voltage for the desired speed. 3) A microprocessor designed to function across a wide range of voltages. Software control is necessary as hardware alone may not be able to differentiate between computationally intensive tasks and non-speed-critical tasks. The supply voltage is regulated through a frequency-voltage feedback loop, controlled by dedicated ring oscillator, which is commonly used as a critical path replica. All chips within the system operate at the same clock frequency and supply voltage given by the ring oscillator [14].

2.3 Process, Voltage and Temperature variations

By looking at the leakage current in the sub-threshold regime (Equation 2.7), it can be seen that it is dependent on the threshold voltage, supply voltage and temperature (PVT). Meaning that changes in any of these parameters will reflect a variation in the drain current. According to [15], circuits are more sensitive to operating conditions in this region. Energy and performance are highly impacted due to process variations, operating temperatures and environment conditions.

The study also shows that an increased temperature improves speed, at the expense of increased energy dissipation. The effect is due to an reduced threshold voltage and an increased mobility at higher temperatures [8]. Temperature has

a much greater impact in the sub-threshold region than in the strong inversion region.

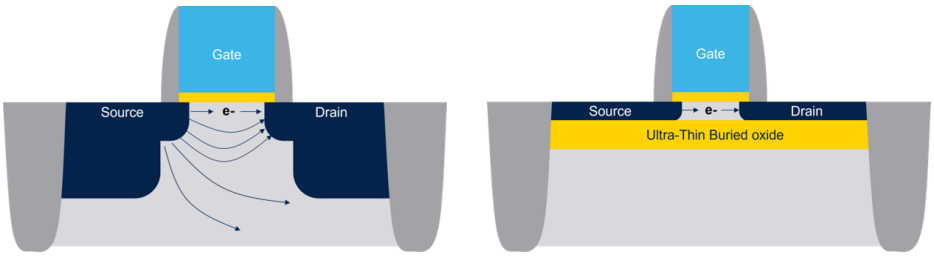
Moreover, due to manufacturing processes, transistors will show a random variation on the device length and width, that deviates the threshold voltage of each device [16]. It has been studied [17] that the standard deviation of the threshold voltage follows an inverse relationship with the square root of the transistor area. Meaning that variation spreads are higher as the device dimensions are reduced. From the same equation, in the sub-threshold regime, the threshold voltage has exponential effects in the drain current.

The stated above shows how difficult it might be to operate with low supply voltages in the sub-threshold region.

2.4 FDSOI

FDSOI is a planar process technology that provides an alternative solution to overcome some of the limitations of bulk CMOS technology at reduced silicon geometries, it stands for *Fully Depleted Silicon on Insulator*. The FDSOI process has two distinct features. First, an ultra-thin buried oxide layer is placed on the top of the base silicon. Second, a very thin silicon layer on top creates the transistor channel. As the transistor channel layer is very thin, no channel doping is required, which makes the transistor fully depleted [18].

FDSOI technology exhibits significantly improved electrostatic characteristics of transistors compared to conventional bulk technology. The incorporation of a buried oxide layer serves to decrease the parasitic capacitance between the source and the drain. Moreover, it effectively confines the electron flow from the source to the drain, resulting in a remarkable reduction in leakage currents that impair performance [18]. Figure 2.4.1 shows the cross-section of both technologies.



(a) Bulk transistor. **(b)** FDSOI transistor.

Figure 2.4.1: Bulk (a) and FSDOI (b) CMOS transistors cross-section. Illustration from STMicroelectronics [18].

In this technology, the transistors possess four effective terminals: source, drain, gate, and body, which refers to the volume beneath the conduction channel.

In FDSOI, by applying a voltage bias to the body terminal, the transistor's threshold voltage can be effectively manipulated , controlling on the fly the threshold voltage by software [19].

This boody-bias can be employed in two directions: *forward body-bias* (FBB) and *reverse body-bias* (RBB). In FBB, the PMOS body terminal is driven to a lower voltage than the source supply (VDD), while the NMOS body terminal is driven to a higher voltage than the source supply voltage (typically ground). This configuration lowers the device threshold voltage, thus incresing performance. Conversely, in RBB, the PMOS body terminal is driven to a higher voltage than VDD, while the NMOS body terminal is driven to a lower voltage than the supply (often a negative voltage). This increases the threshold voltage with reduced leakage [19].

In summary, the body-bias technique has different impacts on the MOS transistor's fundamental characteristics. Forward body-bias (FBB) is aimed at enhancing performance by increasing mobility and reducing the threshold voltage (VT). On the other hand, reverse body-bias (RBB) is suitable for mitigating leakage current by increasing VT and improving the sub-threshold slope [19].

2.5 Ring oscillator

A ring oscillator is a digital oscillator created by connecting an odd number (n) of inverters in a closed-loop cascade [20]. When each inverter has the same propagation delay (Ti), the oscillation frequency can be determined using Equation 2.12, as illustrated in Figure 2.5.1.

$$F_{CLK} = \frac{1}{n * T_i} \tag{2.12}$$

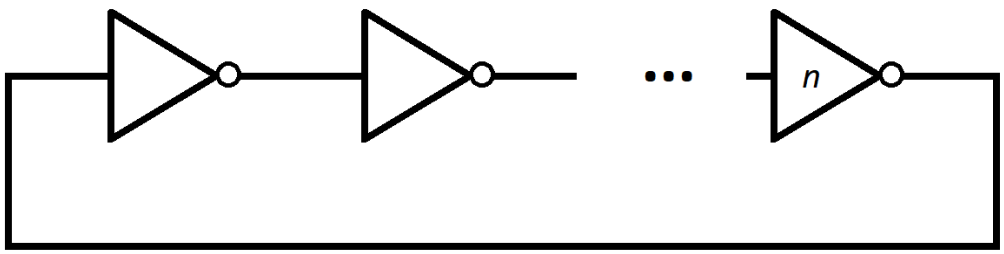


Figure 2.5.1: Schematic of a ring oscillator.

These oscillators are highly desirable due to their ease of design using CMOS

technology. They can achieve oscillations at low voltages, offer electrical tuning, and provide a wide range of tuning options [11]. Additionally, by adjusting the supply voltage, a Voltage-Controlled Oscillator (VCO) can be obtained, allowing an increase or decrease of the propagation delay in each gate. One common application is incorporating them into a dynamic voltage scaling controller to regulate the operating frequency based on the core's performance [7].

2.6 Previous work

Studying low-power and low-energy techniques is not new. Minimum Energy Point around the sub-threshold region has been researched before, both by mathematical models, simulations and physical implementations.

In the study carried by [15], an optimal pair of VDD-Vth is explored with simulations of 11-stage NAND ring oscillators with BSIM3 models (Berkeley Short channel Insulated gate field effect transistor Model). It shows the effects of activity factor and operating conditions, concluding that operating at the optimal point can lead to an order of magnitude in energy savings.

Likewise, [16] studied the MEP in a 11-stage NAND ring oscillator with variable activity factor. Later extending the simulations to CMOS SRAMs, showing that the leakage energy increases and the dynamic energy reduces in a quadratic manner as the supply voltage is reduced. Finally locating the MEP to be around 300 mV.

The MEP has been studied [21] for a 32-bit adder and 16-bit multiplier circuits in 65 nm Silicon on Thin-BOX (SOTB) and bulk devices. The circuits were implemented on chip using only 2-input gates. The results reveal that minimum energy of SOTB is up to 49% smaller than bulk devices and that the MEP is proportional to the temperature. The supply voltage for a minimum energy operation increases linearly with temperature.

Finite impulse response (FIR) filters have been simulated and tested on chip to examine the MEP in sub-threshold circuits, showing the dependence on technology, design, temperature, duty cycle and workload. Variation in these parameters require an adjustment of hundred of millivolts in the supply voltage, demonstrating the need for a MEP tracking mechanism. Moreover, the study confirms that CMOS standard cells libraries are fit for sub-threshold operations [8].

According to [22], some algorithms have been developed previously to estimate the MEP at run time, however they are time-consuming and not fit for real-time operations. They proposed a real-time tracking algorithm based in predefined MEP curves characterized at Slow-Slow (SS), Typical-Typical (TT), and Fast-Fast (FF) conditions generated for a given chip during the boot phase. It was tested with a 50-stage inverter chain designed with 55 nm deeply depleted channel

(DDC) process. It demonstrates that this model can quickly track the MEP even with wide PVT variations.

Finally, the *Minimum Energy Point Exploration in a CPU Subsystem* specialization project [23], carried in the fall semester 2022 at NTNU, also under Nordic Semiconductor supervision is the precursor of the current thesis. The study was limited to a single VT flavor (LVT) with a 24 nm channel length in a 22 nm FD-SOI process. The results were obtained at 25℃, with a typical transistor model and the body biasing recommended by the foundry.

This project provided a great opportunity for laying the foundations of the current work, as much of the initial setup and debugging of new tools happened during that time. Therefore, this specialization project enabled to extend the scope of this thesis and explore various conditions that yield better estimations.

CHAPTER

THREE

METHODS

The current thesis was conducted under the guidance of Nordic Semiconductor, located in Trondheim, Norway, during the spring semester 2023. It is the continuation of the previous study, *Minimum Energy Point Exploration in a CPU Subsystem*, carried in the fall semester 2022 [23].

This chapter presents the methodology and tools required to replicate the study, so that it can be expanded to different designs and technologies. Although there are many different tools, both licensed and freeware, the ones selected are those widely used both by the industry and by Nordic Semiconductor.

The flow used to extract the MEP of a RISC-V CPU is illustrated in Figure 3.0.1, every step is described in detail in the following sections. An overview of the methodology is as follows: The RTL design is simulated to extract the switching activity. The design goes under a synthesis process taking the selected standard cell library and the activity file as inputs. A gate-level netlist is obtained, from which a power estimation is generated and the critical path extracted. The critical path is used to build a ring oscillator that is simulated at the circuit-level sweeping through different PVT corners. From the analog simulation results and the original power report, the MEP for the selected conditions can be estimated. The worst-condition of the MEP is used to generated valid biasing points that can be feedback to the analog simulation to obtain better results through adaptive body-biasing techniques.

3.1 List of tools

- Mentor QuestaSim 10.7c_3.

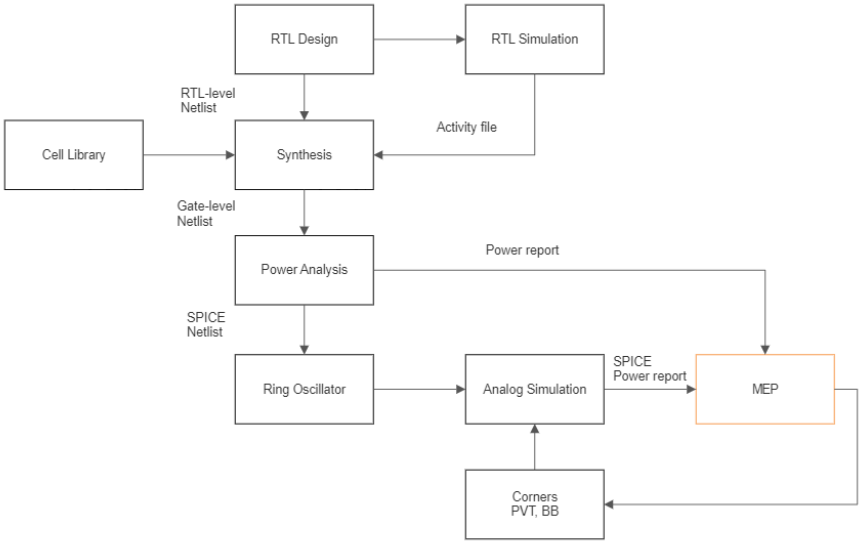


Figure 3.0.1: Minimum Energy Point exploration flow.

- Synopsys Verdi 2021.09-sp2-2
- Synopsys Design Compiler 2020.09.
- Synopsys PrimePower 2019.12-SP5.
- Cadence Virtuoso ICADVM20.1
- Python 3.6.8 and 3.11.1

3.2 The RTL design: Ibex RISC-V Core

The very first step when studying the Minimum Energy Point is to have a design that can benefit from this technique. In particular, processors can profit greatly from Dynamic Voltage and Frequency Scaling (DVFS) and from threshold voltage scaling [22].

The choice of the Ibex RISC-V Core for the MEP exploration was based on several factors. It is based on a reduced instruction set computer (RISC) architecture, which simplifies the processor design. These type of architecture is incredible popular. According to RISC-V International, there are more than 10 billion RISC-V cores in the market, and tens of thousands of engineers working on RISC-V initiatives globally. They have seen more than 26% membership growth year-over-year, with over 3,180 members across 70 countries [24].

The Ibex Core is an open-source project targeted for embedded applications and has been thoroughly tested. It contains all the necessary modules for a minimum working CPU subsystem and offers integration with various simulation and synthesis tools. Additionally, it provides a framework for building and simulating designs, along with extensive documentation [25].

According to the Ibex group [25], the Ibex Core is a 32-bit production quality CPU based on the RISC-V architecture. It is implemented in SystemVerilog and maintained by lowRISC. The developers state that this core is specifically designed for embedded control applications. It has undergone extensive verification and supports various RISC-V extensions such as Integer, Embedded, Integer Multiplication and Division, Compressed, and Bit Manipulation extensions.

The project repository provides a *Simple System*, which integrates the Ibex Core with a basic system for simulations. The Simple System includes the following components:

- Ibex Core.
- Single memory for instructions and data.
- Basic peripheral for writing ASCII output to a file and halting the simulation.
- Basic timer peripheral capable of generating interrupts.
- Software framework for program development.

The Ibex Core RTL source files are actively under review and development by the community in a public repository. Many changes have been pushed since the beginning of this thesis. The CPU design used in this study was released in October 14th, 2022.

3.2.1 Building the software

To test the capabilities of the Simple System, a *hello_test.c* file is also provided. This file contains instructions for printing strings to a log file based on a periodic timer. The source file was compiled using the developers' provided Make executable. The resulting binary was then dumped into a Verilog Memory file (vmem) following the memory organization of the Simple System. This allows the instructions to be fetched by the Ibex core.

The Simple System binary was built using FuseSoc, a package manager and set of build tools for HDL [26]. FuseSoc allows the passing of parameters to build the final RTL design. For this study, the following parameters were used: RV32MFast (fast multiply extension), RV32BNone (no bit manipulation extension), RegfileFF (register file based on flip-flops), and the path to the memory file (vmem) containing the software code. These parameters are the standard recommendations according to the Ibex documentation for a simple system.

3.2.2 RTL simulation

Once a design has been selected for the study, the next step in the MEP exploration process is to extract the switching activity through RTL simulations. According

to the Synopsys PrimePower user guide [27], three main components are required to obtain an accurate power estimate from a design:

- Gate-level netlist describing the design.
- Power characterization of cells in a standard library.
- RTL simulation traces.

The gate-level netlist is obtained after synthesis, and the power characterization is already available in the library. However, the RTL simulation traces need to be extracted at this stage to provide the switching activity of the design under a specific workload defined by the firmware running on the core.

To extract the RTL simulation traces, the Simple System design, with the software loaded into the instruction memory, was simulated using *Mentor QuestaSim 10.7c_3*. The simulation was executed using the `make run` instruction provided by the Ibex framework. The Makefile takes care of loading all source RTL files in the correct order using TCL scripts.

The primary focus at this stage is the extraction of the switching activity. As a result, all signals were stored in a Fast Signal Database (FSDB) during the simulation, allowing their utilization in the subsequent estimation phase. The clock period specified in the testbench was continuously fine-tuned within the exploration loop to achieve an optimal value for each corner. It is crucial for the clock speed used during synthesis to match the simulated clock in order to obtain the most accurate estimation. Failing to do so could potentially impact the toggle rates [28].

3.3 The synthesis

In accordance with a standard synthesis flow [29], the Ibex Simple System underwent synthesis using a 22 nm FDSOI library in *Synopsys Design Compiler 2020.09*. It was synthesized with different transistor flavors and different channel length: SLVT (20 to 36 nm), LVT (24 to 36 nm), HVT (20 to 28 nm) and UHVT (28 to 36 nm). The synthesis process was carried out through TCL scripting (refer to Appendix .6). The flow for synthesis is outlined below:

First, a 22 nm FDSOI 8-track standard cell library was selected for the synthesis process. The available libraries are listed below.

1. SLVT : in 20, 24, 28, 32, 36 nm characterized at 0.4V.
2. LVT: in 24, 28, 32, 36 nm characterized at 0.4V.
3. HVT: in 20, 24, 28 nm characterized at 0.8V.
4. UHVT: in 28, 32, 36 nm characterized at 0.8V.

Additionally, the RTL source files were loaded in the same hierarchical order as done during simulation. All files related to the testbench are excluded from the synthesis.

To ensure accurate power analysis, the switching activity needed to be reported in a Switching Activity Interchange Format (SAIF) file. This involved associating the design objects with their corresponding gate-level netlist elements. In other words, mapping RTL names to their gate-level counterparts. During synthesis, the switching activity was included in the FSDB file, and a name mapping database was created. The FSDB file was then converted into a SAIF file using the `fsdb2saif` command provided by *Synopsys Verdi 2021.09-SP2-2*.

A clock constraint was set, and its value was determined through a manual iterative loop. The loop continued until the critical path achieved a slack value of zero, or slightly negative. So that it is operating at its maximum speed. During this loop, new RTL simulations are obtained with the corresponding clock value (section 3.2.2), so that all clocks match.

The synthesized gate-level netlist, the Synopsys Design Constraint (SDC) file and the name mapping database were exported to be consumed in the following step.

3.4 Power estimation

An average power analysis was performed using a TCL script (refer to Appendix .7) in *PrimePower 2019.12-SP5* on the gate-level netlist generated by Synopsys Design Compiler in the previous step. To enhance accuracy, the analysis utilized the SDC file (constraints), SAIF file (containing switching activity), and an RTL-to-gate mapping file.

It is crucial for the power estimation to be reliable that close to 100% of the switching activity is properly annotated. This verification can be carried out using the `report switching activity -list not annotated` command to identify any unannotated switching activity.

A power report is generated for the Ibex Simple System. The report includes information about dynamic power (reported by Synopsys as *Net Switching* and *Cell Internal*), static power (*Cell Leakage*), and the total power consumption. The energy per cycle can be obtained by multiplying by the clock frequency used as a synthesis constraint. The report is only generated at typical corner characterized by the library (0.4 V/0.8 V, 25℃, Typical-Typical). There is no way of obtaining the MEP from this report. However, it will serve as a reference for extrapolating the power corners from a ring oscillator, for those corners not characterized in the library.

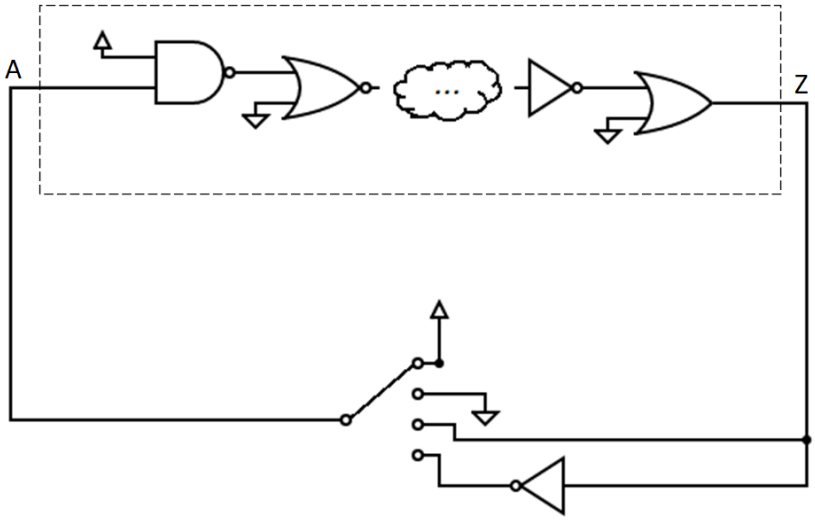


Figure 3.5.1: Diagram of a testbench for ring oscillator.

3.5 Building a ring oscillator

The power estimation conducted in the previous section 3.4 is limited to the characterized corners of the library. Specifically, the Typical-Typical (TT) process corner offers supply voltages of 0.4 V, 0.5 V, 0.6 V, and 0.8 V. Power estimation below 0.4 V cannot be performed using PrimePower, because the tool can only properly estimate the power consumption for the library corners, defined through liberty files. These files establish timing and power characteristics at a defined voltage for every cell in the library. These values are unknown for supply voltages not defined by the foundry. Instead, an analog simulation can be executed on the same transistor model with a lower supply voltage (or different VT flavor, temperature, or body biasing).

Due to the complexity and time-consuming task of running a SPICE simulation of the entire Ibex Core, only the critical path is simulated, as it is considered a representative sample. But also because this is the path that has the least slack and the one that requires the highest speeds. Using a replica of the critical path as a ring oscillator is a very common practice to regulate the speed and supply voltage of the main circuit [7], [22], [30].

The critical path is dumped into a SPICE netlist via the `write_spice_deck` command provided by Synopsys PrimePower. The netlist includes the cells in the path, resistors, ground capacitors, and coupling capacitors [27]. The circuit described by the netlist is converted into a ring oscillator by fixing the inputs that do not belong to the critical path either to VDD or VSS, such that the main input of the critical path can be observed at the output. Figure 3.5.1 illustrates this idea.

Using Python scripting (refer to Appendix .9), the exported SPICE netlist was modified to adhere to the Cadence Virtuoso simulator syntax. Additionally,

the initial and final sequential elements in the path were removed since only the combinational logic is of interest for building a ring oscillator. For each cell in the critical path there are individual voltage sources (VDD, VSS, VPW_P, VNW_N), they were removed, and respective pins were connected together and set as output ports of the critical path subcircuit. This reduces circuit complexity and allows values to be changed through a design variable in a testbench. A testbench circuit was created by instantiating the critical path subcircuit, a four-pole single-throw switch, all power supplies, and an ideal inverter (derived from a voltage-controlled voltage source).

After the critical path netlist has been parsed and cleaned it is converted into a SPICE subcircuit (SUBCKT), named *CRITICAL_PATH*, which makes the ring oscillator circuit. A proper testbench is created, instantiating the subcircuit, the voltages sources and the 4-input switch. The testbench netlist is described in Listing 3.1.

Listing 3.1: SPICE testbench

```
.include netlist_critical_path.spi

vvdd vdd 0 vdd
.global vss
vvss vss 0 0
.temp 25
vvnw_n vnw_n 0 vnw_n
vvpw_p vpw_p 0 vpw_p

xtop_01 vdd vss vnw_n vpw_p A Z CRITICAL_PATH

* Input A is connected to a sp4tswitch. Inputs are VSS, VDD, Z or !Z
* Switch is controlled via sw_pos variable

einverter nZ vss vdd Z 1
simulator lang=spectre
S0 (A vss vdd Z nZ) sp4tswitch paramTyp=string tranPosition_str=sw_pos model=switch
simulator lang=spice

.end
```

The four-pole single-throw switch is connected in the following manner, shown also in Figure 3.5.1:

- Input 1 is connected to VDD.
- Input 2 is connected to VSS.
- Input 3 is connected to Z (output of ring oscillator).
- Input 4 is connected to !Z (inverted output of ring oscillator).
- Output 1 is connected to A (input of ring oscillator).

It can be decided which switch configuration to use at simulation time by means of a design variable. Configurations 1 and 2 will not make the circuit oscillate due to their static behavior, therefore this will yield the static power information. Either configuration 3 or 4 generates an oscillatory response on the circuit. If the ring oscillator has an odd number of inverter-like gates, then configuration 3 makes the circuit oscillate. In the case when the number o inverter-like gates is odd, an additional inverter is needed to make the circuit oscillate, in this case, configuration 4 is needed.

The number of inverter-like gates in the netlist can be counted by studying the type of cells and their inputs-outputs paths, but it is very complex operations since there are near 1000 cells per library. Instead, it was decided that it is much easier to simulate for both possible conditions, and take the results of the oscillatory case.

3.5.1 Analog simulation of RO

The testbench and critical path netlists from the previous section, and the transistor models are imported into *Cadence Virtuoso ICADVM20.1* from command line by using Spice In. It creates a schematic of the circuit that can later be simulated. The import command is [31]:

```
spiceIn -netlistFile netlist_testbench.spi -outputLib $TESTBENCH
↪  -language SPICE -refLibList "cmos22fdsoi analogLib" -devMapFile
↪  devMapping.file
```

Where *netlist_ testbench.spi* is the netlist described in Listing 3.1, which is instantiating the critical path by means of the `.include` directive. Output files are located in the directory pointed by $TESTBENCH, and the devMapping file contains the device mapping information (such as resistor = res).

All SPICE simulations were performed using OCEAN scripts. The graphical user interface (GUI) was only used for the initial setup and understating how the tool works, everything was moved later to OCEAN scripting because of the amount of simulations and corners needed. This also allows greater automation during the exploration flow. See Appendix .8 for the script file.

The *Open Command Environment for Analysis* (OCEAN) is the scripting language used by Cadence Virtuoso Analog Design Environment to set up, simulate and analyze circuit data from command line without starting the GUI. Script files and simulations were design according to the recommended instructions [32], [33]. Initially, a basic simulation can be set up through the GUI and the configurations exported into an OCEAN file, that works as a baseline for designing more complex simulations.

The simulation time must be enough to accommodate a few cycles on the slowest corner. But because there is a huge difference in speed between the slowest and fastest corner, this time cannot be a fixed value, or it would be unfeasible to get results in a timely manner. For instance, ten cycles in the slowest corner could mean one million cycles in the fastest corner (five orders of magnitude greater) for a fixed stop time, creating huge amount of data with a big impact on computing resources. An optimization is therefore required.

The problem is that the frequency of oscillation is not know beforehand and

cannot be calculated until the simulation has finished and the simulation time cannot be defined until the oscillation frequency is known, in other words, a *chicken or egg* dilemma is faced. To circumvent this problem, a big enough fixed simulation time is used, and a counter written in Verilog-A (refer to Appendix .11) is instantiated so that the current analysis can be stopped dynamically once a given number of cycles have been obtained. The transient simulation will then continue with the following corner. The `finish_current_analysis(0)` command achieves the desire behavior.

Once the testbench, critical path, cell and transistor models are imported into Virtuoso, OCEAN scripts are used to simulate the circuit in all four possible switch cases: 1) connected to VDD, 2) connected to VSS, 3) connected to Z, 4) connected to !Z. All simulation parameters are described in the script, such as: supply voltage, temperature, transistor model, biasing voltages. The simulations can be started from the command line with the command:

```
virtuoso -nograph -restore script.ocn
```

The most important results are extracted using the script with the following directives:

- The frequency of oscillation is measured as the frequency of the voltage signal Z, after removing the first half of the simulation. This ensures that any transitory behavior is discarded.

```
frequency(clip(VT(\\"/Z\\") (lastVal(VT(\\"/Z\\")) / 2)
↪  lastVal(VT(\\"/Z\\"))))
```

- The power consumed by the circuit is equal to the average of the current signal through the VDD port, times the voltage value of VDD. First half of the simulation is also discarded ($P = I_{DD} * V_{DD}$).

```
average(clip((VAR(\\"vdd\\") * IT(\\"/xtop_01/VDD\\"))
↪  (lastVal(VT(\\"/Z\\")) / 2) lastVal(VT(\\"/Z\\"))))
```

- The duty cycle is given by the time it takes the voltage signal Z to cross half VDD on both edges.

```
dutyCycle(VT(\\"/Z\\") ?mode \\"user\\" ?threshold
↪  (VAR(\\"vdd\\") / 2) ?xName \\"time\\" ?outputType
↪  \\"average\\")
```

All data is dumped into CSV files automatically once the simulations are finished. Columns represent the simulated corner and rows contain the results of the previous expressions (frequency, power, duty cycle).

P	V	T(°C)	VNW_N(V)	VPW_P(V)
TT	0.20	25	0.8	-1.067
TT	0.21	25	0.8	-1.067
TT	0.22	25	0.8	-1.067
TT	0.23	25	0.8	-1.067
TT	0.24	25	0.8	-1.067
TT	0.25	25	0.8	-1.067
TT	0.26	25	0.8	-1.067
TT	0.27	25	0.8	-1.067
TT	0.28	25	0.8	-1.067
TT	0.29	25	0.8	-1.067
TT	0.30	25	0.8	-1.067
TT	0.31	25	0.8	-1.067
TT	0.32	25	0.8	-1.067
TT	0.33	25	0.8	-1.067
TT	0.34	25	0.8	-1.067
TT	0.35	25	0.8	-1.067
TT	0.36	25	0.8	-1.067
TT	0.38	25	0.8	-1.067
TT	0.40	25	0.8	-1.067
TT	0.50	25	0.542	-0.632
TT	0.60	25	0.346	-0.210
TT	0.80	25	-0.024	0.020

Table 3.5.1: Simulation corners for SLVT and LVT on typical conditions.

3.5.1.1 Simulation corners

The following tables show all the parameters that were set for the analog simulations. PVT refers to Process (SS, TT, FF), Voltage (supply voltage) and Temperature. VNW_N and VPW_P refers to the biasing voltages, used for forward body-biasing (FBB). Reverse body-bias (RBB) is not studied in this thesis.

Table 3.5.1 contains the corners used to extract the MEP with SLVT and LVT libraries at typical conditions and fixed body biasing. For voltages 0.4, 0.5, 0.6 and 0.8 V, the biasing voltages used are the ones recommended by the foundry. Under 0.4 V the libraries are undefined, and it was decided to replicate the same values used for 0.4 V. In the case of HVT and UHVT libraries, FBB is not supported, so their biasing voltages are always zero, this is illustrated in Table 3.5.2. This is the first approximation on studying the MEP: only typical conditions and fixed body-biasing, so that supply voltage, threshold voltage and channel length are studied. In this way 22 corners are studied per VT and channel length, yielding a total of 330 corners.

The study is extended to different PVT conditions (slow and cold, slow and hot, fast and cold, fast and hot) with fixed body-bias. The corners for SLVT and LVT are shown in Table 3.5.3. Since HVT and UHVT do not support FBB, the corners are the same as Table 3.5.3 but with zero bias voltages. Now the exploration is expanded to 110 corners per VT and channel length, giving a total of 1650 corners studied.

P	V	T(°C)	VNW_N(V)	VPW_P(V)
TT	0.20	25	0	0
TT	0.21	25	0	0
TT	0.22	25	0	0
TT	0.23	25	0	0
TT	0.24	25	0	0
TT	0.25	25	0	0
TT	0.26	25	0	0
TT	0.27	25	0	0
TT	0.28	25	0	0
TT	0.29	25	0	0
TT	0.30	25	0	0
TT	0.31	25	0	0
TT	0.32	25	0	0
TT	0.33	25	0	0
TT	0.34	25	0	0
TT	0.35	25	0	0
TT	0.36	25	0	0
TT	0.38	25	0	0
TT	0.40	25	0	0
TT	0.50	25	0	0
TT	0.60	25	0	0
TT	0.80	25	0	0

Table 3.5.2: Simulation corners for HVT and UHVT on typical conditions.

To further expand the study, different body-bias voltages are used, this is explained in section 3.7.

3.6 Estimating MEP from RO

Four CSV files are obtained from the simulation ran for the ring oscillator, described in the previous section, one for each switch case. They yield the following results: 1) static power on high input, 2) static power on low input, 3) total power and frequency, 4) total power and frequency (either 3. or 4. have valid data depending on the amount of inverters in the path).

The static power consumed by the ring oscillator is calculated as the average between the static power when input is high and when input is low. This is the best approach as it takes into consideration both equally possible cases. The dynamic power, due to switching state of transistors, can be computed as the subtraction between the total power and the average static power.

The energy per cycle is obtained by dividing the total power, dynamic power, and static power by the frequency of oscillation of the ring oscillator for each corner. By locating the lowest total energy consumption, the MEP is obtained for the RO. However, the focus of this study lies on finding the MEP of the original design, the Ibex core.

P	V	T(℃)	VNW_N(V)	VPW_P(V)	P	V	T(℃)	VNW_N(V)	VPW_P(V)
SS	0.20	-40	0.8	-1.067	SS	0.31	-40	0.8	-1.067
SS	0.20	125	0.8	-1.067	SS	0.31	125	0.8	-1.067
TT	0.20	25	0.8	-1.067	TT	0.31	25	0.8	-1.067
FF	0.20	-40	0.8	-1.067	FF	0.31	-40	0.8	-1.067
FF	0.20	125	0.8	-1.067	FF	0.31	125	0.8	-1.067
SS	0.21	-40	0.8	-1.067	SS	0.32	-40	0.8	-1.067
SS	0.21	125	0.8	-1.067	SS	0.32	125	0.8	-1.067
TT	0.21	25	0.8	-1.067	TT	0.32	25	0.8	-1.067
FF	0.21	-40	0.8	-1.067	FF	0.32	-40	0.8	-1.067
FF	0.21	125	0.8	-1.067	FF	0.32	125	0.8	-1.067
SS	0.22	-40	0.8	-1.067	SS	0.33	-40	0.8	-1.067
SS	0.22	125	0.8	-1.067	SS	0.33	125	0.8	-1.067
TT	0.22	25	0.8	-1.067	TT	0.33	25	0.8	-1.067
FF	0.22	-40	0.8	-1.067	FF	0.33	-40	0.8	-1.067
FF	0.22	125	0.8	-1.067	FF	0.33	125	0.8	-1.067
SS	0.23	-40	0.8	-1.067	SS	0.34	-40	0.8	-1.067
SS	0.23	125	0.8	-1.067	SS	0.34	125	0.8	-1.067
TT	0.23	25	0.8	-1.067	TT	0.34	25	0.8	-1.067
FF	0.23	-40	0.8	-1.067	FF	0.34	-40	0.8	-1.067
FF	0.23	125	0.8	-1.067	FF	0.34	125	0.8	-1.067
SS	0.24	-40	0.8	-1.067	SS	0.35	-40	0.8	-1.067
SS	0.24	125	0.8	-1.067	SS	0.35	125	0.8	-1.067
TT	0.24	25	0.8	-1.067	TT	0.35	25	0.8	-1.067
FF	0.24	-40	0.8	-1.067	FF	0.35	-40	0.8	-1.067
FF	0.24	125	0.8	-1.067	FF	0.35	125	0.8	-1.067
SS	0.25	-40	0.8	-1.067	SS	0.36	-40	0.8	-1.067
SS	0.25	125	0.8	-1.067	SS	0.36	125	0.8	-1.067
TT	0.25	25	0.8	-1.067	TT	0.36	25	0.8	-1.067
FF	0.25	-40	0.8	-1.067	FF	0.36	-40	0.8	-1.067
FF	0.25	125	0.8	-1.067	FF	0.36	125	0.8	-1.067
SS	0.26	-40	0.8	-1.067	SS	0.38	-40	0.8	-1.067
SS	0.26	125	0.8	-1.067	SS	0.38	125	0.8	-1.067
TT	0.26	25	0.8	-1.067	TT	0.38	25	0.8	-1.067
FF	0.26	-40	0.8	-1.067	FF	0.38	-40	0.8	-1.067
FF	0.26	125	0.8	-1.067	FF	0.38	125	0.8	-1.067
SS	0.27	-40	0.8	-1.067	SS	0.40	-40	0.8	-1.067
SS	0.27	125	0.8	-1.067	SS	0.40	125	0.8	-1.067
TT	0.27	25	0.8	-1.067	TT	0.40	25	0.8	-1.067
FF	0.27	-40	0.8	-1.067	FF	0.40	-40	0.8	-1.067
FF	0.27	125	0.8	-1.067	FF	0.40	125	0.8	-1.067
SS	0.28	-40	0.8	-1.067	SS	0.50	-40	0.542	-0.632
SS	0.28	125	0.8	-1.067	SS	0.50	125	0.542	-0.632
TT	0.28	25	0.8	-1.067	TT	0.50	25	0.542	-0.632
FF	0.28	-40	0.8	-1.067	FF	0.50	-40	0.542	-0.632
FF	0.28	125	0.8	-1.067	FF	0.50	125	0.542	-0.632
SS	0.29	-40	0.8	-1.067	SS	0.60	-40	0.346	-0.210
SS	0.29	125	0.8	-1.067	SS	0.60	125	0.346	-0.210
TT	0.29	25	0.8	-1.067	TT	0.60	25	0.346	-0.210
FF	0.29	-40	0.8	-1.067	FF	0.60	-40	0.346	-0.210
FF	0.29	125	0.8	-1.067	FF	0.60	125	0.346	-0.210
SS	0.30	-40	0.8	-1.067	SS	0.80	-40	-0.024	0.020
SS	0.30	125	0.8	-1.067	SS	0.80	125	-0.024	0.020
TT	0.30	25	0.8	-1.067	TT	0.80	25	-0.024	0.020
FF	0.30	-40	0.8	-1.067	FF	0.80	-40	-0.024	0.020
FF	0.30	125	0.8	-1.067	FF	0.80	125	-0.024	0.020

Table 3.5.3: Simulation corners for SLVT and LVT on SS, TT and FF conditions.

By utilizing the power report generated by PrimePower for the TT corner, the energy per cycle of the Ibex core can be calculated by multiplying it by the clock period. Subsequently, the ratio of leakage energy between the ibex core and the ring oscillator can be determined at the typical corner (TT, 25℃, 0.4 V for SLVT and LVT, 0.8 V for HVT and UHVT). Since the critical path (RO) is only a subcircuit of the entire design, this ratio is expressing how much bigger is the core compared to the critical path. Although the ratio is only calculated at a single corner point, it will be considered a fixed value since it is expected that the ibex core scales in the same magnitude as the ring oscillator. Therefore this ratio is used for extrapolating the leakage energy at all corners.

In that sense, the static energy of the core at a given supply voltage equals that of the critical path times the leakage ratio:

$$E_{static}(V_{DD}) = E_{staticCP}(V_{DD}) * Ratio$$

The switching energy at any supply voltage is computed mathematically considering it has a square relationship with VDD, using as a baseline the energy obtained from the PrimePower report. In this case a ratio between the ibex core and ring oscillator is not used, because the RO has a 100% toggle rate, which is not true for the original circuit.

For SLVT and LVT:

$$E_{dynamic}(V_{DD}) = \frac{E_{dynamic(0.4V)}}{(0.4V)^2} * V_{DD}^2$$

For HVT and UHVT:

$$E_{dynamic}(V_{DD}) = \frac{E_{dynamic(0.8V)}}{(0.8V)^2} * V_{DD}^2$$

Finally, the total energy represents the contribution of the previous two components.

$$E_{total}(V_{DD}) = E_{dynamic}(V_{DD}) + E_{static}(V_{DD})$$

The Minimum Energy Point is located at the global minimum of the E_{total} signal as the supply voltage scales down.

3.6.1 Excluding worst performing cells

Standard libraries contain hundreds of cells, somewhere between 400 and 1000 cells in the libraries used on this study. Typically, synthesis tools pick the best cells for a given design such that it meets the deadlines and constraints, and is still power-efficient. Nonetheless, this is only optimized at the corner where the

Combination	Timing Typical Case	Timing Worst Case	Ratio	Worst ratio
A->Z	1	3	3	3
B->Z	1	2	2	-
C->Z	1	2	2	-

Table 3.6.1: Timing ratio between typical and worst case for a NAND3 example.

library is characterized, for instance, typical case, 0.4 V, 25℃. It is possible that cells do not scale equally as the supply voltage is lowered and closest to the MEP. It means that there are good performing cell at typical corner, that behave poorly at low voltages. Being able to exclude these cells opens another degree of freedom in the MEP exploration domain.

In order to tell which cells can be excluded, a ranking of all available cells is needed. This is obtained by looking at the timing information on the library file in the typical-case, 0.4 V at 25℃ (0.8 V at 25℃ for HVT and UHVT), and the same information on the library file in the worst-case, 0.36 V at -40℃ (0.72 V at -40℃ for HVT and UHVT). The ratio between both measurements represents the scaling factor. The cells with the biggest scaling factor are those that scale poorly from the typical to worst-case condition.

A python script is used for such task (refer to Appendix .10). It goes through the entire library file and extracts the average between rise and fall time for all input-output combinations for every cell, on both typical and worst case conditions. Then, for every cell and every matching pair of input-output combinations, the ratio is calculated, but only the worst is taken.

For instance, table 3.6.1 illustrates this with a NAND3 example. The average timing from A to Z at typical conditions is 1, and for the worst condition is 3, so that the ratio between worst and typical is also 3. Therefore the A to Z combination has the worst ratio of the three possible cases, this value is taken to build a weighted ranking. For the ranking, the cells are evaluated against all other cells, and not against cells with similar logic function. For instance, a NAND3 is not compared against a NAND2 and NAND3 with different driving strength to choose the best, but compared globally with all other cells. Excluding only high fan-in cells was not studied here.

Once a ranking is obtained in a CSV format, a TCL script can be generated containing cell that are meant to be excluded from the synthesis. For this purpose, Synopsys provides the command `set_dont_use`. It means that the tools will not used any of the cells listed. The command can be used in the following way:

```
set sdc_version 2.1
set_dont_use [get_lib_cells */NAND3X010]
set_dont_use [get_lib_cells */NOR3X010]
```

To avoid running into synthesis errors due to lack of compatible library cells,

not able to find replacements for registers, all D-Flip-Flops are in a "do not touch" list. Meaning that these type of cells are always allowed to be used by the tool.

3.6.2 Scaling the leakage power

It results of interest to study how the MEP shifts as the leakage power of a given circuit changes but the switching power remains the same. For instance, if peripherals are added or removed, the leakage power will undoubtedly change as long as they are a powered-up, but the switching power will remain the same if they are not in used. Since this is a very real case, it becomes important to understand where the MEP lies and how to optimise for it.

The flow is exactly the same as described in previous sections. Since the switching activity remains unchanged, there is no need to rerun the entire process. However, the leakage power obtained from section 3.5.1 can be scaled by any factor. For this thesis, it was studied at 0.5 and 2 times the leakage.

3.7 Estimating MEP body-biasing corners

Different MEP can be obtained for different biasing-body pair values (VNW_N and VPW_P), as body-biasing techniques allow to dynamically change the transistor's VT, making them faster or slower, and with greater or less leakage. SLVT and LVT technologies allow only Forward Body-Biasing (FBB), in the following rage, VNW_N: 0 to 2 V, VPW_P: -2 to 0 V. That is, applying a positive voltage to the NMOS transistors, and a negative voltage on PMOS transistors. UHVT and HVT support only RBB, but it is not explored in this thesis.

There are infinite values in the range described before, and only a subset of them are considered valid. To limit the scope, the valid range of biasing points that can be applied to the circuit are those that give a near-symmetric response in rising and falling times of the basic gates NAND3, NOR3 and INV at the slow corner. The threshold for deciding whether the points are allowed or not is extracted from the liberty file.

This is an iterative process. First, the MEP is extracted by using the corners described in Table 3.5.3, where the biasing voltages are fixed. Later, the three gates are simulated in Virtuoso, such that the PVT conditions can be controlled, while at the same time sweeping through the entire range of VNW_N and VPW_P, every 100 mV. The PVT condition are those of the slow corner (SS, -40℃ at -10% VDD of the MEP). From the simulation, the rising and falling time are extracted.

Figure 3.7.1 shows the schematic of the testbench used to run these simulations (netlist available in Listing 3.2). Three INV gates are used as loads, connected to the output port of each DUT. The INV, NAND3 and NOR3 under test have their

VNW_N and VPW_P ports connected to a voltage source which is controlled through a design variable, such that all values can be swept. To get the worst-case scenario results, port A of NAND3 (which is the closest to VSS) is used as the switching input, similarly port A of NOR3 (which is the closes to VDD) is used for simulation. All other remaining inputs are fixed either to VDD or VSS respectively.

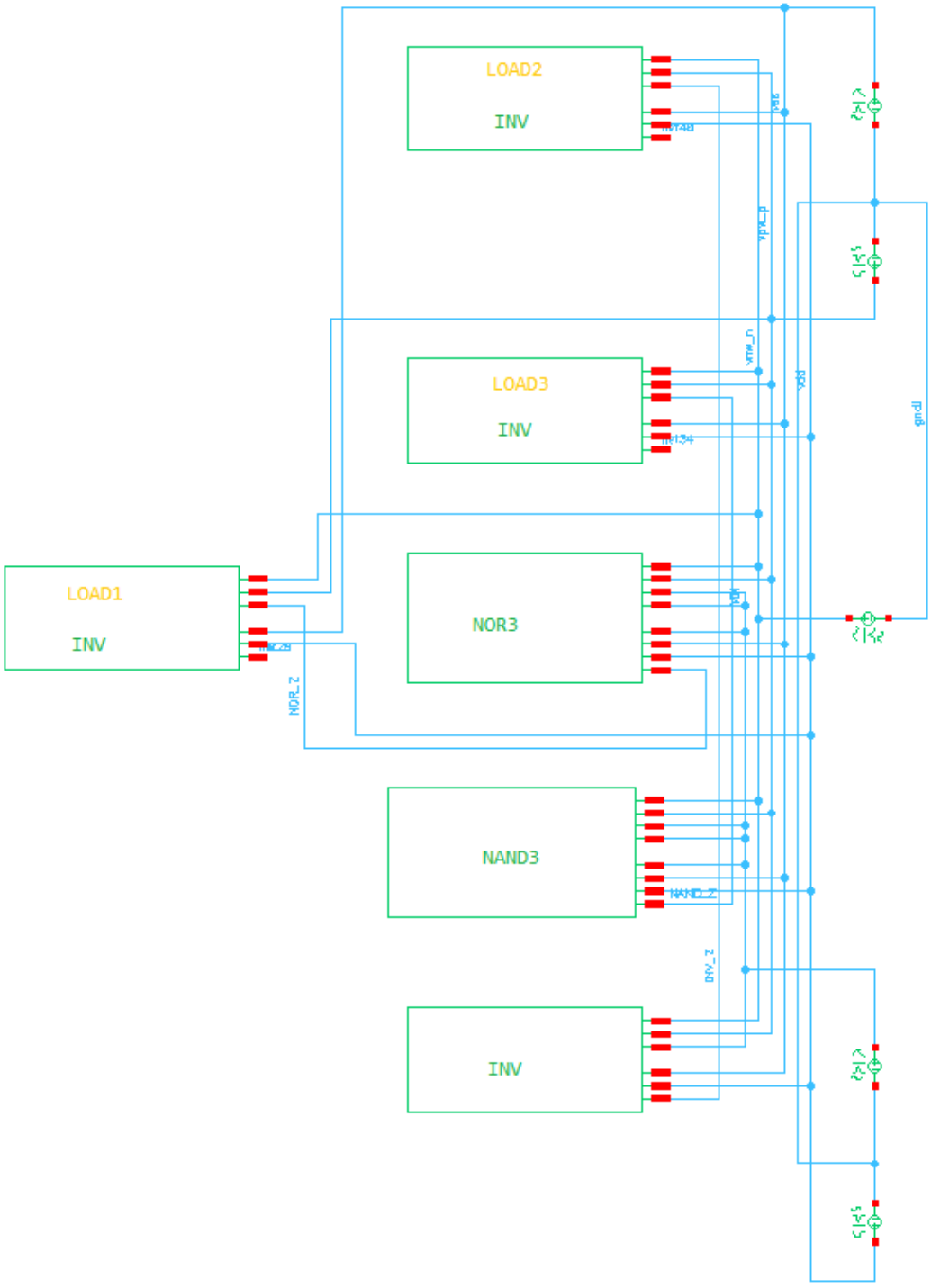


Figure 3.7.1: Testbench for simulating basic gates with body bias (netlist in Listing 3.2).

Listing 3.2: Testbench netlist for simulating basic gates

```
simulator lang=spectre
vvdd (vdd 0) vsource dc=vdd type=sine
vvss (vss 0) vsource dc=0 type=sine
vvpw_p (vpw_p 0) vsource dc=vpw_p type=sine
```

```
vvnw_n (vnw_n 0) vsource dc=vnw_n type=sine
vin (VIN 0) vsource type=pulse val0=0 val1=vdd period=period  \
        delay=50n rise=10n fall=10n width=period/2 - 10n

* pins: .Z .VDD .VNW_N .VPW_P .VSS .A .B .C
NAND (NAND_Z vdd vnw_n vpw_p vss VIN vdd vdd) NAND3X010
* pins: .Z .VDD .VNW_N .VPW_P .VSS .A
INV (INV_Z vdd vnw_n vpw_p vss VIN) INVX010
* pins: .Z .VDD .VNW_N .VPW_P .VSS .A .B .C
NOR (NOR_Z vdd vnw_n vpw_p vss VIN vss vss) NOR3X010

* Loads
LOAD1 (net28 vdd vnw_n vpw_p vss NOR_Z) INVX010
LOAD2 (net40 vdd vnw_n vpw_p vss INV_Z) INVX010
LOAD3 (net34 vdd vnw_n vpw_p vss NAND_Z) INVX010
```

CHAPTER

FOUR

RESULTS

This chapter presents the results on how the Minimum Energy Point is affected by different parameters. The role of supply voltage, threshold voltage, channel length, temperature and body biasing voltage is studied. The design under test is an Ibex Core CPU, it remains unchanged for all simulations. It follows the methodology presented in Chapter 3.

The study starts with a single degree of freedom, the supply voltage. This is shown on section 4.4. Progressively more parameters are added to the study. Later on section 4.5 the study is repeated for cold and hot, and slow and fast corners.

The study ends with the exploration of the MEP taking into account the effect of body biasing voltages, this is shown on section 4.6. It is limited only to two different VT flavors: SLVT and LVT, which showed the best performance.

4.1 RTL simulation

Since the RTL Ibex Core is already given, the first step is obtaining the switching activity. This is done through an RTL simulation with a proper testbench. All signals are saved in a FSDB file. Figure 4.1.1 shows a visual representation of these signals. This dump can initially help identify if the simulation was properly set, that signals are properly recorded and that the clock period matches the one set as constraint. But its main use is to serve as input parameter during synthesis.

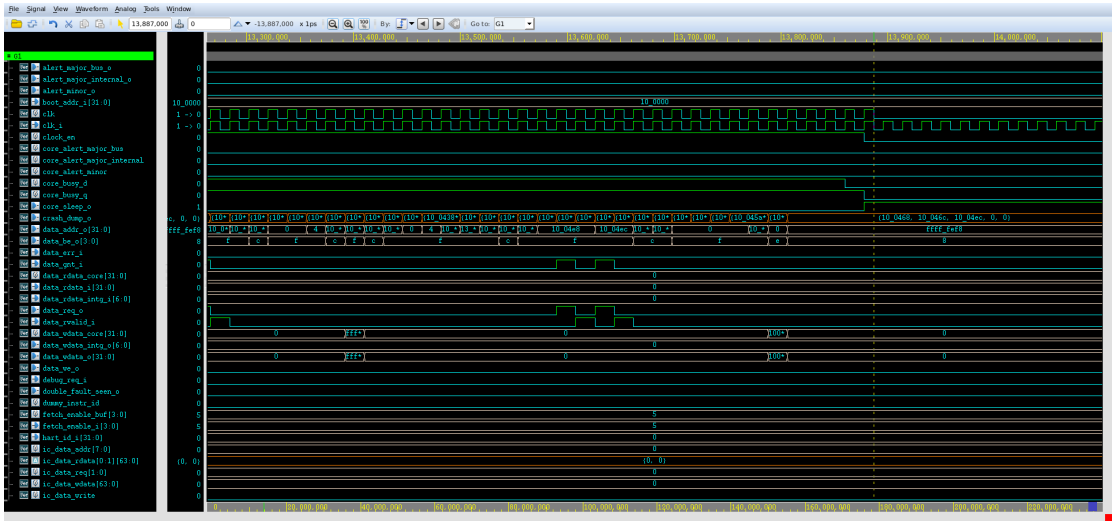


Figure 4.1.1: Simulation traces captured in FSDB file.

4.2 Synthesis and power estimation

The RTL design is synthesized for all of the 15 standard cell libraries available (see 3.3). It takes the RTL source files and the FSDB as inputs. For each library, the following files are obtained: a synthesized netlist described in verilog (ibex_top_map.v), the design constraints (ibex_top_sdc.sdc) and a name mapping file (ibex_top_name_mapping.tcl). Listings 4.1, 4.2 and 4.3 show a partial example for a select library.

Listing 4.1: Synthesized netlist for a SLVT technology in 36 nm

```
////////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : U-2022.12-SP2
// Date      : Tue Apr 18 10:12:36 2023
////////////////////////////////////////////////////////////////


module prim_generic_clock_gating_0_1 ( clk_i, en_i, test_en_i, clk_o );
  input clk_i, en_i, test_en_i;
  output clk_o;
  wire   N1, en_latch;

  SLVT36_LHNQX010 en_latch_reg ( .CN(clk_i), .D(N1), .Q(en_latch) );
  SLVT36_AND2X010 U2 ( .A(en_latch), .B(clk_i), .Z(clk_o) );
  SLVT36_OR2X005 U3 ( .A(en_i), .B(test_en_i), .Z(N1) );
endmodule


module prim_clock_gating ( clk_i, en_i, test_en_i, clk_o );
  input clk_i, en_i, test_en_i;
  output clk_o;
```

Listing 4.2: Synopsys Design Constraints for a SLVT technology in 36 nm

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

# Created by write_sdc on Tue Apr 18 10:12:37 2023

#####################################################################
set sdc_version 2.1

set_units -time ns -resistance MOhm -capacitance fF -voltage V -current uA
create_clock [get_ports clk_i]  -name CLK  -period 6  -waveform {0 3}
```

Listing 4.3: Name mapping file for a SLVT technology in 36 nm

```
# Primepower name mapping file
# created using Synopsys Power Compiler U-2022.12-SP2 */
# design ibex_top
#
```

```
set_rtl_to_gate_name -rtl {core_busy_q} -gate [get_cell {core_busy_q_reg}]
set_rtl_to_gate_name -rtl {gen_regfile_ff.register_file_i/g_rf_flops[7].rf_reg_q[4]} \
    -gate [get_cell {gen_regfile_ff.register_file_i/g_rf_flops[7].rf_reg_q_reg[4]}]
set_rtl_to_gate_name -rtl {gen_regfile_ff.register_file_i/g_rf_flops[3].rf_reg_q[4]} \
    -gate [get_cell {gen_regfile_ff.register_file_i/g_rf_flops[3].rf_reg_q_reg[4]}]
set_rtl_to_gate_name -rtl {gen_regfile_ff.register_file_i/g_rf_flops[7].rf_reg_q[19]} \
    -gate [get_cell {gen_regfile_ff.register_file_i/g_rf_flops[7].rf_reg_q_reg[19]}]
set_rtl_to_gate_name -rtl {gen_regfile_ff.register_file_i/g_rf_flops[28].rf_reg_q[4]} \
    -gate [get_cell {gen_regfile_ff.register_file_i/g_rf_flops[28].rf_reg_q_reg[4]}]
```

Following the methods described in 3.4, a power estimation is obtained for each synthesized netlist. It consumed the inputs described earlier and outputs a power estimation report, a timing report, a SPICE netlist of the critical path and a not annotated switching activity report (refer to Appendix .2 for the activity report).

Listing 4.4 contains the power report for a given library. In this report, *Net Switching Power* and *Cell Internal Power* are both components of the dynamic power. While *Cell Leakage Power* refers to the static power.

Listing 4.4: PrimePower average power report

```
****************************************
Report : Averaged Power
Design : ibex_top
Version: Q-2019.12-SP5
Date   : Sun Feb 26 20:09:14 2023
****************************************


  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network           3.264e-05 3.064e-05 2.028e-10 6.328e-05 (92.22%)  i
register                1.420e-07 4.554e-08 1.321e-06 1.508e-06 ( 2.20%)
combinational           6.420e-07 7.306e-07 2.456e-06 3.829e-06 ( 5.58%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                     0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 3.142e-05   (45.79%)
  Cell Internal Power  = 3.342e-05   (48.71%)
  Cell Leakage Power   = 3.777e-06   ( 5.50%)
                         ---------
Total Power            = 6.862e-05  (100.00%)
```

In order to validate the results, a not annotated switching activity report is generated, this is shown in Appendix .2, where it has been highlighted in a grey box that all activity is properly mapped from the activity file.

Table 4.2.1 shows the number of standard cells used in the synthesized netlist and the critical path. It also shows the number of unique cells in both cases. In the case of SLVT32 and LVT24, the critical path is dominated by the use of half-adders, causing a very low count on the number of unique cells.

	Total cells	Unique cells
SLVT20 Ibex Core	13489	189
SLVT20 Critical Path	77 (0.57%)	40 (21.2%)
SLVT24 Ibex Core	13300	165
SLVT24 Critical Path	58 (0.44%)	29 (17.6%)
SLVT28 Ibex Core	13268	189
SLVT28 Critical Path	71 (0.54%)	11 (5.9%)
SLVT32 Ibex Core	13099	165
SLVT32 Critical Path	65 (0.50%)	5 (3.0%)
SLVT32 Ibex Core	13528	199
SLVT32 Critical Path	58 (0.43%)	27 (13.6%)
LVT24 Ibex Core	12738	186
LVT24 Critical Path	64 (0.50%)	6 (3.2%)
LVT28 Ibex Core	12775	186
LVT28 Critical Path	62 (0.49%)	40 (21.5%)
LVT32 Ibex Core	12839	191
LVT32 Critical Path	62 (0.48%)	43 (22.5%)
LVT36 Ibex Core	12700	181
LVT36 Critical Path	54 (0.43%)	33 (18.2%)
HVT20 Ibex Core	16059	297
HVT20 Critical Path	54 (0.34%)	28 (9.4%)
HVT24 Ibex Core	16043	298
HVT24 Critical Path	58 (0.36%)	29 (9.7%)
HVT28 Ibex Core	16369	280
HVT28 Critical Path	58 (0.35%)	27 (9.6%)
UHVT28 Ibex Core	16004	193
UHVT28 Critical Path	59 (0.37%)	28 (14.6%)
UHVT32 Ibex Core	16803	193
UHVT32 Critical Path	67 (0.40%)	31 (16.1%)
UHVT36 Ibex Core	15693	185
UHVT36 Critical Path	62 (0.40%)	33 (17.8%)

Table 4.2.1: Cell comparison between Ibex Core and Critical Path.

4.3 The testbench

Both the testbench and critical path subcircuit are imported into Virtuoso along the standard cell libraries and transistor models. Figure 4.3.1 shows the imported testbech circuit. A *counter* block described in Verilog-A is also istantiate manually to interrupt the simulation after 10 cycles have been calculated per corner. Figure 4.3.2 shows an example of a critical path replica that will work as a ring oscillator, an example of the netlist is available in Appendix .1.

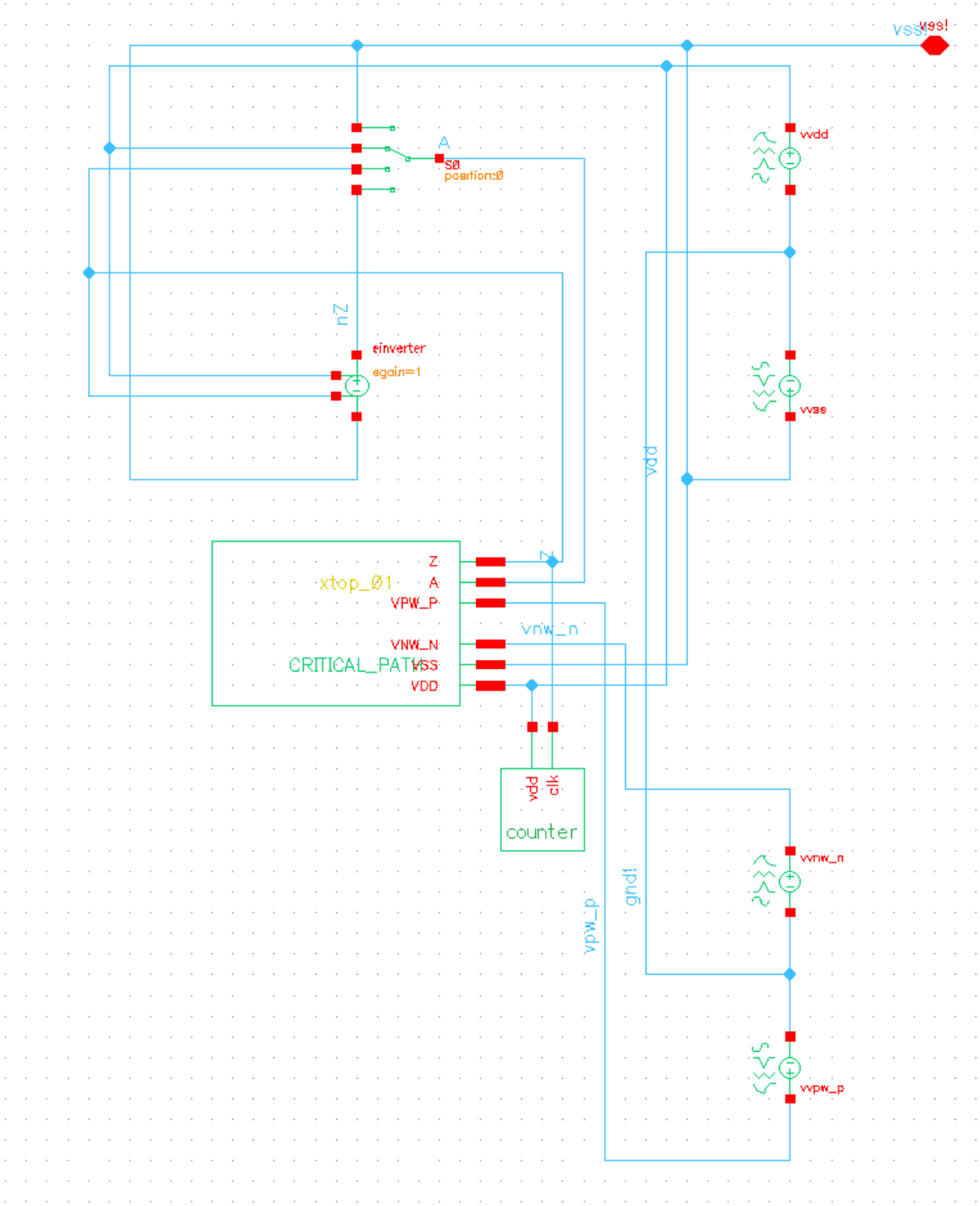


Figure 4.3.1: Testbench for ring oscillator.

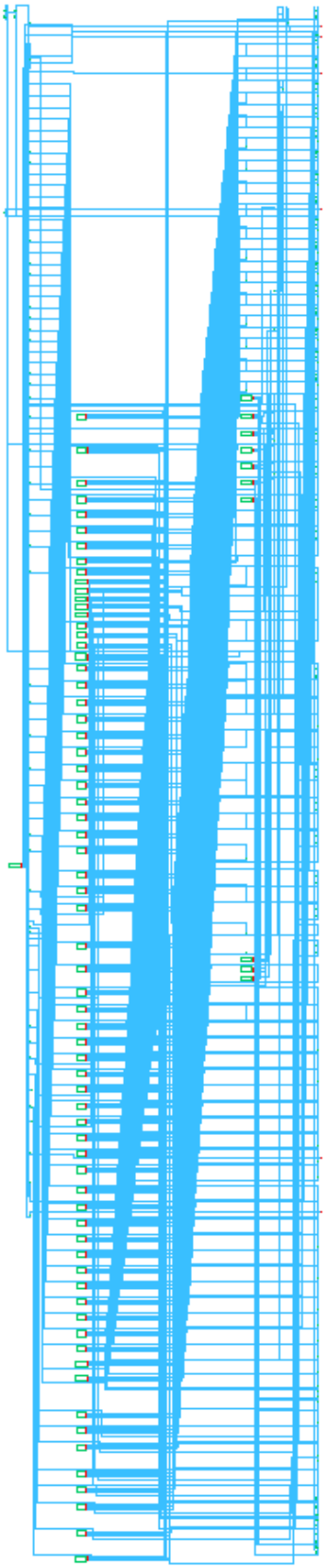


Figure 4.3.2: Critical path replica subcircuit, complete netlist available in Appendix .1.

To this point the Ibex Core has been simulated at RTL level and synthesized. A power estimation has been obtained and the critical path replica converted into a ring oscillator. A SPICE testbench has been built to run analog simulation on this replica. The sections that follows explore these simulations and their results.

4.4 MEP by sweeping supply voltage

This first section explores the relationship between Minimum Energy Point and voltage scaling. Starting the study with one degree of freedom allows to get the basic background on MEP, to establish a baseline for the results and to verify all scripts and flow are properly set. Further studies with more complex parameters stack upon these results.

The most important parameters used during these simulations are the following:

- A 22 nm FDSOI process.
- Room temperature (25℃).
- Fixed body biasing voltages, as suggested by the foundry.
- Multiple threshold voltages: SLVT, LVT, HVT, UHVT.
- Different channel length: 22, 24, 28, 32 and 36 nm.
- Typical-Typical (TT) condition: Both NMOS and PMOS operate in typical corner.
- See Table 3.5.1 and Table 3.5.2 for detailed PVT corners.

Initially, the Ibex Core is synthesized on typical conditions with a LVT library characterized at 0.4 V, with a 28 nm channel length. From the synthesized netlist, the critical path is extracted and converted into a ring oscillator. The ring oscillator is on typical conditions while lowering the supply voltage from 0.8 V down to 0.2 V. A finer-grain step of 10 mV is used in the range between 0.2 to 0.36 V, as the MEP is expected to be in the sub-threshold zone.

The leakage power and total power are extracted from the simulation. The leakage power is obtained when the ring oscillator is not oscillating (see Appendix .3.1 for example), so there is no change in state of the transistors. Whilst the total power measurement results from the oscillatory behavior (see Appendix .4.1 for example). The switching power is computed as the subtraction of the total power minus the leakage power. Finally the energy is calculated as the power divided by the frequency of oscillation.

As described ins section 3.6, the RO results are extrapolated to the entire Ibex Core by multiplying the leakage energy by a proper scaling factor. Figure 4.4.1 shows the main components of energy consumption of the entire Ibex Core. The blue curve represents the leakage energy, which increases as the supply voltage scales down. The orange curve represents the switching energy decreasing with a

square relationship on the supply voltage. Finally, the green line represents the total energy of the circuit, which is the results of the sum of the previous two. In dotted red lines, the Minimum Energy Point is marked at (0.31 V, 0.96 pJ). It shows how the MEP is located at the lowest point of the convex function described by the total energy function.

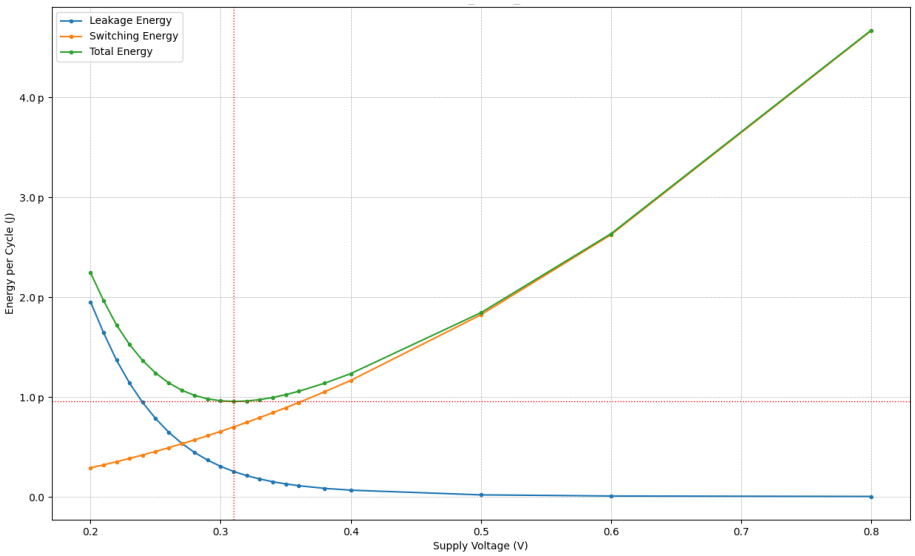


Figure 4.4.1: MEP for a 22 nm FDSOI process, TT, LVT, 28 nm channel length at 25℃.

Table 4.4.1 shows some valuable points for analysis, including supply voltage, frequency and energy consumed. The data show that operating at the MEP is saving 23% energy when compared to the 0.4 V corner, but with a 83% decrease in speed. If compared against the 0.8 V corner, it yields a saving of 79% in energy with a 98% reduced speed. If the 0.2 V corner is analyzed, it can be seen it consumes almost twice the energy that the 0.4 V corner, but 90 times slower.

VDD (V)	Total Energy (pJ)	Frequency (MHz)
0.20	2.25	0.32
0.28	1.02	2.33
0.31	(MEP) 0.96	4.81
0.34	1.00	9.45
0.40	1.24	28.77
0.80	4.67	284.50

Table 4.4.1: Energy and frequency comparison at different supply voltages for LVT28.

In order to study the effect of the threshold voltage on the MEP, the simulations are repeated under the same condition but with different VT. These results are

shown on Figure 4.4.2. In particular, the circuit was synthesized with SLVT, LVT, HVT and UHVT libraries, all at 28 nm.

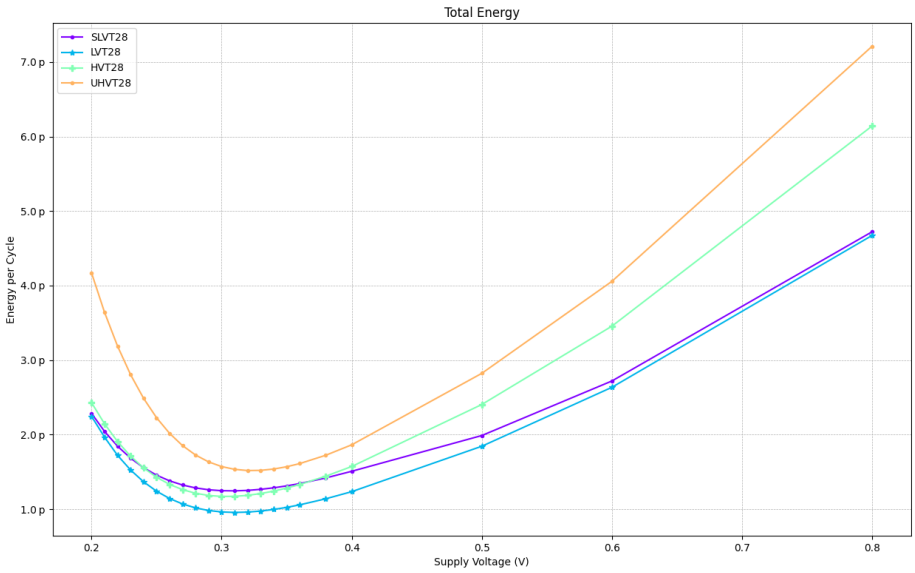


Figure 4.4.2: Total Energy for SLVT, LVT, HVT and UHVT with 28 nm channel length, TT at 25℃.

The study is expanded to different VT and different channel length, all other parameters are maintained. That is, the simulations are repeated for all possible VT and channel length combinations that are available. This is illustrated in Figure 4.4.3.

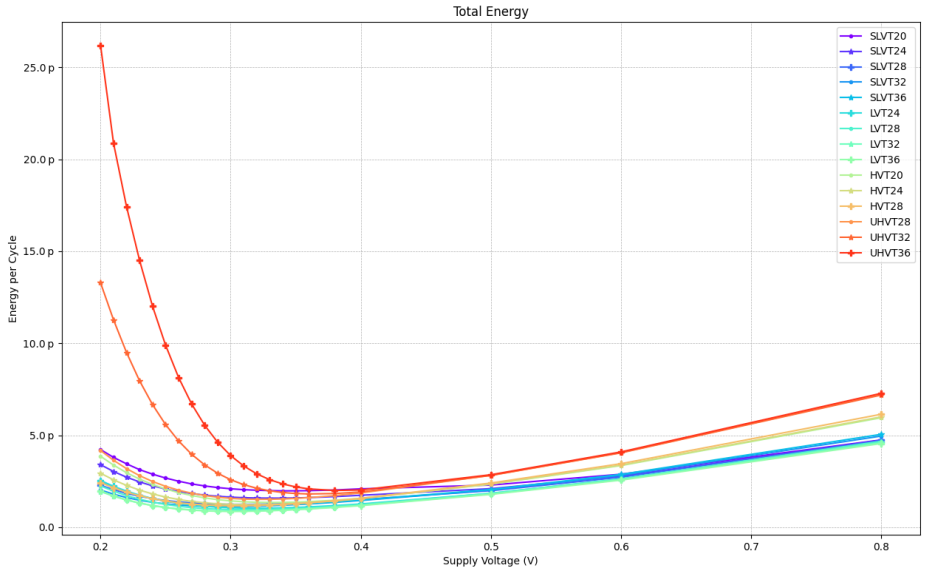


Figure 4.4.3: Total Energy for all VT and all channel length at typical conditions, 25℃.

To be able to analyze the results in a more intuitive way around the MEP, the plots are changed from Energy Vs VDD, to Frequency Vs Energy. This is how results are presented from now on. Three points are plotted per curve: The center point is located at the MEP, and the end points represent the frequency and energy pair located at ± 10% of the supply voltage of the MEP. So the plots can give an idea on how the frequency and energy shift as a 10% margin is given to the supply voltage.

In this sense, Figure 4.4.4 represents the same data plotted in Figure 4.4.3 but in a more useful manner. In this type of plot, top-left corner is the ideal case: high frequency and low energy consumption. Bottom-right corner is the opposite: low frequency and high energy consumption.

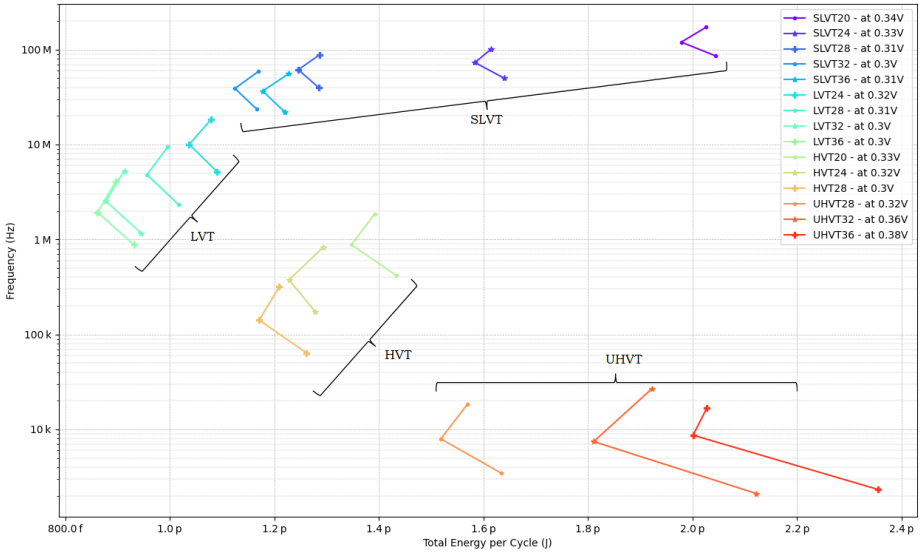


Figure 4.4.4: MEP for all VT and channel length, TT at 25℃.

In Figure 4.4.5 the frequency vs leakage energy can be observe. This is important for analysis as the leakage energy plays a key role in the location of the MEP.

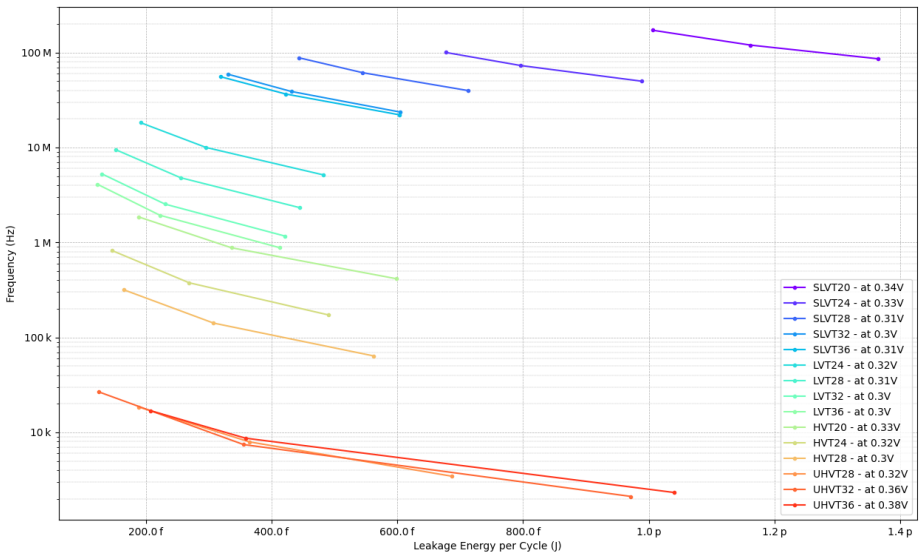


Figure 4.4.5: Leakage at MEP for all VT and channel length, TT at 25℃.

4.4.1 MEP excluding worst performing cells

Following section 3.6.1, a set of standard cells are excluded from the synthesis process to evaluate how the new circuit would perform. A list of 20% to 50% of the worst performing cells in the libraries are excluded. The results are show in 4.4.6.

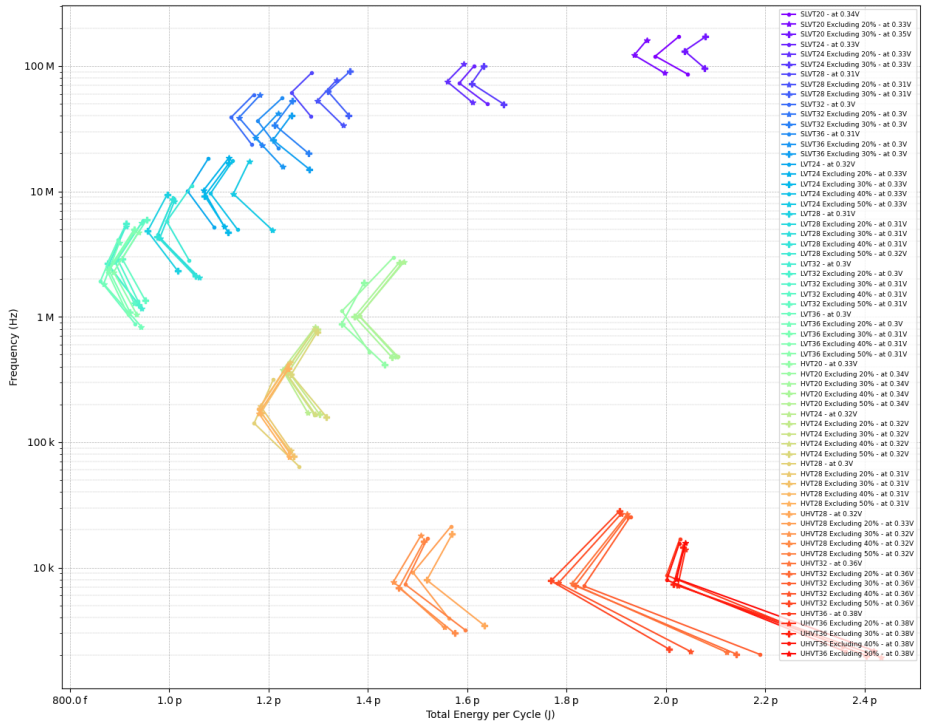


Figure 4.4.6: MEP for all VT and channel length excluding worst cells, TT at 25°C.

4.5 MEP by sweeping supply voltage and temperature

This section presents the results of the Minimum Energy Point exploration under the following PVT conditions:

- A 22 nm FDSOI process.
- Different process variations: SS, TT, FF.
- Cold, room and hot temperatures (-40, 25, 125°C).
- Fixed body biasing voltages, as suggested by the foundry.
- Multiple threshold voltages: SLVT, LVT, HVT, UHVT.

- Different channel length: 22, 24, 28, 32 and 36 nm.
- See Table 3.5.3 for detailed PVT corners.

The evaluation is started with a single VT flavor on 28 nm tested at -40, 25 and 125℃ for a TT process variation, so that the effect of temperature can be observed. Figure 4.5.1 shows the total energy as voltage is scaled down, at three different temperatures. The MEP is pointed with a red dot. Notice how it shifts into higher voltages and energy as the temperature increases.

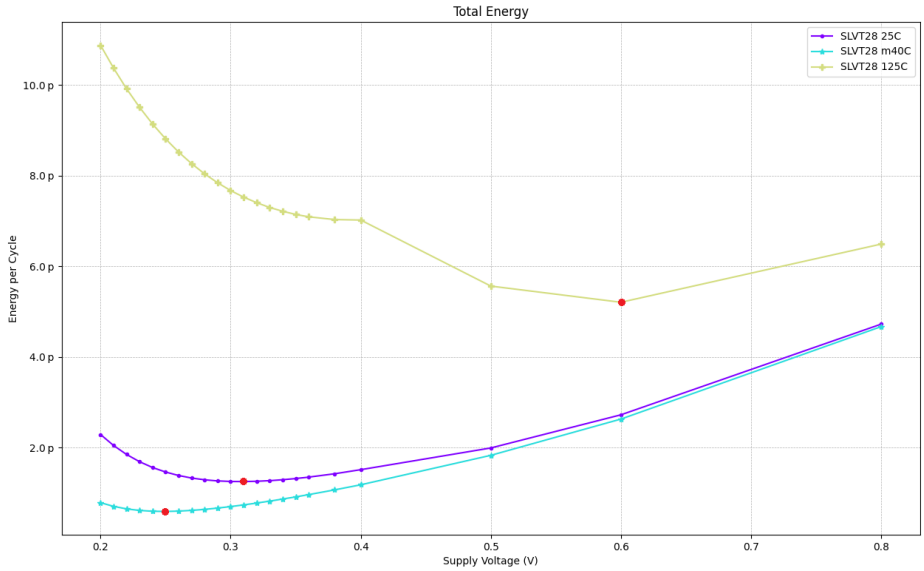


Figure 4.5.1: Total energy for SLVT28 at -40, 25 and 125℃. Note that the strange shape is due to different biasing voltages at 0.4, 0.5, 0.6 and 0.8 V as suggested by the foundry.

Figure 4.5.2 demonstrates the key effect of the temperature in the leakage energy. As the temperature rises, more leakage energy is consumed.

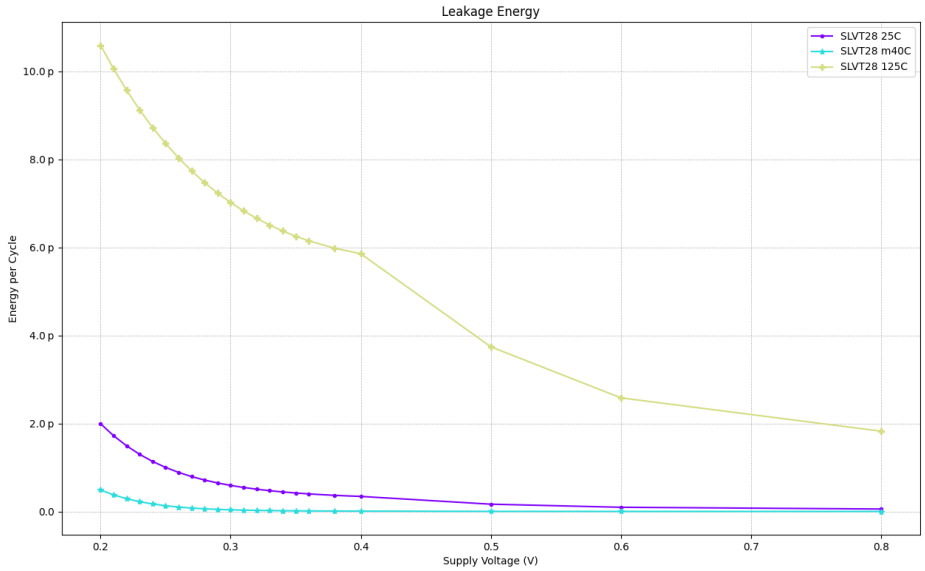


Figure 4.5.2: Leakage energy for SLVT28 at -40, 25 and 125℃.

The study is extended to slow and fast conditions. To clearly visualize the results, frequency against energy is plotted in semi-log or log-log graphs. Each series plotted represents a given technology forming a triad. The center point of each series is the MEP found at typical-typical, 25℃ condition. The other two points are those found at +10% of VDD at MEP, at the fast corner (FF process at -40 or 125℃, whatever is fastest), and -10% of VDD at MEP, at the slow corner (SS process at -40 or 125℃, whatever is slowest).

Figure 4.5.3 shows the results of all four VT flavors on a same channel length, so that comparison becomes easier. It is important to note that UHVT (orange line) is plotted only with two points since the RO does not oscillate at the slow corner. Table 4.5.1 illustrate this results in tabular form.

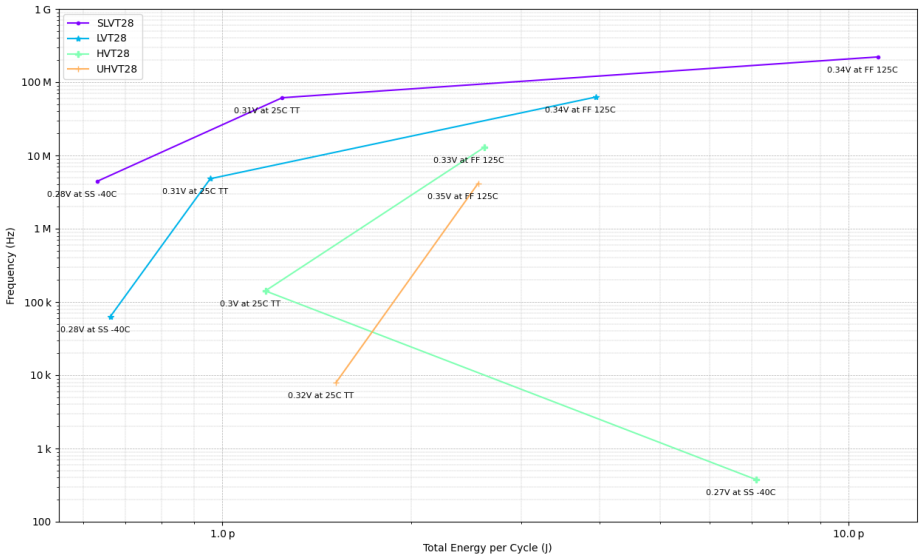


Figure 4.5.3: MEP for SLVT, LVT, HVT, UHVT 28 nm center at typical condition at 25℃, slow corner: -10% VDD, SS, -40℃; and fast corner: +10% VDD, FF 125℃. Note:UHVT (orange) does not oscillate at slow corner.

Library	Slow (-10% VDD)	Typical	Fast (+10% VDD)
SLVT28	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
HVT28	0.27 V at SS -40℃	0.3 V at TT 25℃	0.33 V at FF 125℃
UHVT28	-	0.32 V at TT 25℃	0.35 V at FF 125℃

Table 4.5.1: MEP conditions for all VT for 28 nm.

The results are expanded to SLVT and LVT technologies on all possible channel length. This is plotted in Figure 4.5.4 with the resulting conditions available in Table 4.5.2.

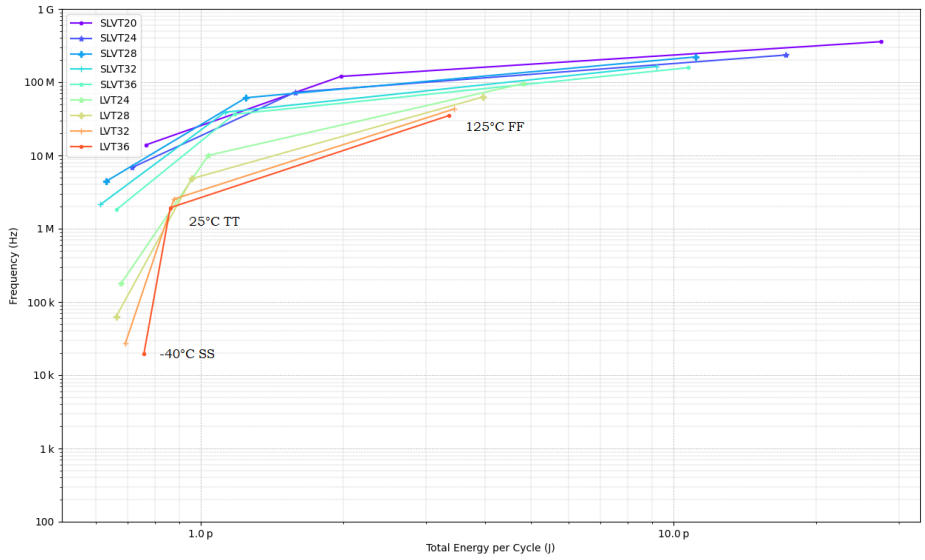


Figure 4.5.4: MEP for SLVT and LVT all channel length center at typical condition at 25℃, slow corner: -10% VDD, SS, -40℃; and fast corner: +10% VDD, FF 125℃.

Library	Slow (-10% VDD)	Typical	Fast (+10% VDD)
SLVT20	0.31 V at SS -40℃	0.34 V at TT 25℃	0.38V at FF 125℃
SLVT24	0.3 V at SS -40℃	0.33 V at TT 25℃	0.36V at FF 125℃
SLVT28	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34V at FF 125℃
SLVT32	0.27 V at SS -40℃	0.3 V at TT 25℃	0.33V at FF 125℃
SLVT36	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34V at FF 125℃
LVT24	0.29 V at SS -40℃	0.32 V at TT 25℃	0.35V at FF 125℃
LVT28	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34V at FF 125℃
LVT32	0.27 V at SS -40℃	0.3 V at TT 25℃	0.33V at FF 125℃
LVT36	0.27 V at SS -40℃	0.3 V at TT 25℃	0.33V at FF 125℃

Table 4.5.2: MEP conditions for SLVT and LVT for all channel length.

Finally, the simulations are extended to all VT flavors (SLVT, LVT, HVT, UHVT) on all possible channel lengths. This is illustrated in Figure 4.5.5 and the conditions shown on Table 4.5.3. It is worth noting that UHVT does not oscillate for the slow corner for any channel length, so that only two points are available in the figure.

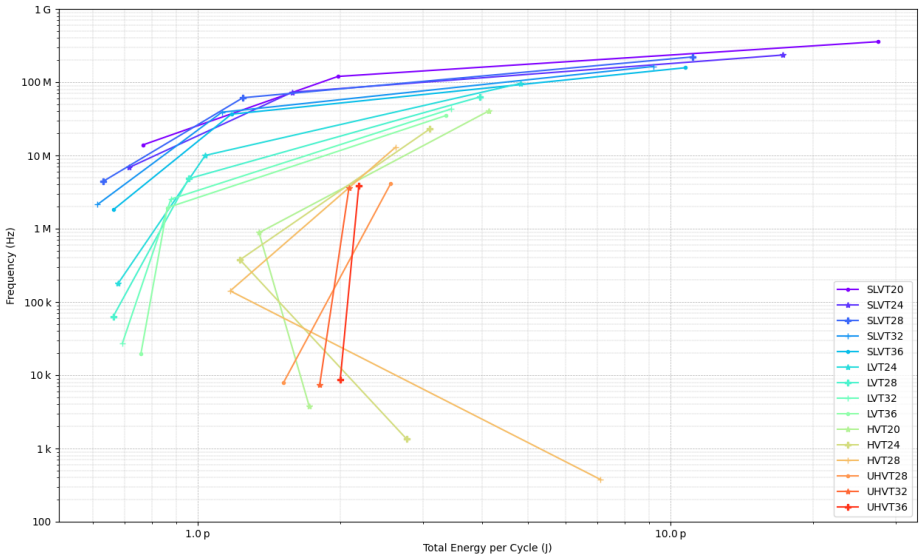


Figure 4.5.5: MEP for all VT and all channel length center at typical condition at 25℃, slow corner: -10% VDD, SS, -40℃; and fast corner: +10% VDD, FF 125℃.

Library	Slow (-10% VDD)	Typical	Fast (+10% VDD)
SLVT20	0.31 V at SS -40℃	0.34 V at TT 25℃	0.38 V at FF 125℃
SLVT24	0.30 V at SS -40℃	0.33 V at TT 25℃	0.36 V at FF 125℃
SLVT28	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
SLVT32	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
SLVT36	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT24	0.29 V at SS -40℃	0.32 V at TT 25℃	0.35 V at FF 125℃
LVT28	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT32	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
LVT36	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
HVT20	0.30 V at SS -40℃	0.33 V at TT 25℃	0.36 V at FF 125℃
HVT24	0.29 V at SS -40℃	0.32 V at TT 25℃	0.35 V at FF 125℃
HVT28	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
UHVT28	-	0.32 V at TT 25℃	0.35 V at FF 125℃
UHVT32	-	0.36 V at TT 25℃	0.40 V at FF 125℃
UHVT36	-	0.38 V at TT 25℃	0.42 V at FF 125℃

Table 4.5.3: PVT variations around MEP for all VT and all channel length with fixed body-bias.

4.5.1 Scaling leakage power

For SLVT and LVT technologies, the experiments are repeated considering the leakage power scales by x0.5 or x2. For all cases the simulations are run only at

TT, 25℃ conditions. Figure 4.5.6 shows the results in a manner that is easy to compare. The center point is the MEP found at the nominal leakage power, and the border points are those where the MEP is found if the leakage increases or decreases by a factor of two. That is, the MEP is always recomputed for every case.

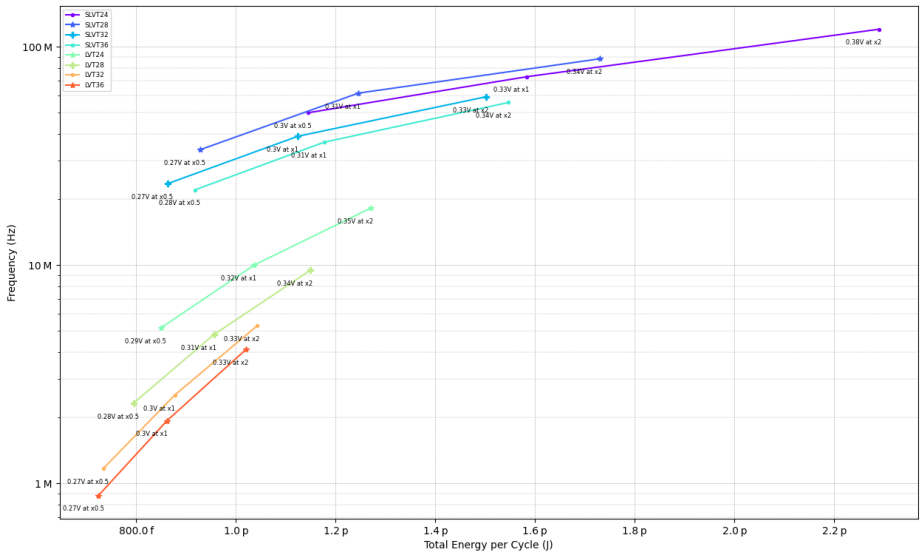


Figure 4.5.6: MEP for x0.5, x1 and x2 leakage.

4.6 MEP by sweeping supply voltage, temperature and body biasing

The results presented here are a continuation over those presented in the previous section 4.5. However, only technologies SLVT and LVT allow forward body biasing techniques. Moreover, only 28 nm is selected for these new simulations to reduce complexity and simulation times.

The main parameters for simulation are:

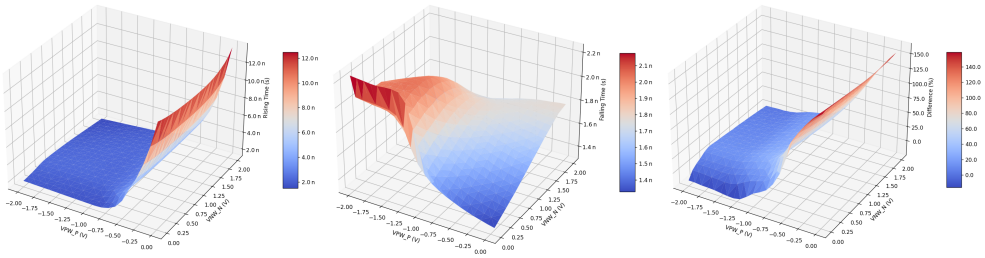
- A 22 nm FDSOI process.
- Cold, room and hot temperatures (-40, 25, 125℃).
- Forward body biasing.
- Two threshold voltage: SLVT, LVT.
- Single channel length: 28 nm.
- See Table 3.5.3 for detailed PVT corners. Columns VNW_N and VPW_P use a value extracted from the simulation of basic gates, presented also in this section.

In order to find the MEP when using forward body biasing techniques, a list of biasing points is needed. These points are extracted from the valid points of the simulation of three basic gates (INV, NOR and NAND).

The SPICE simulation of these gates are swept through the entire range of valid forward biasing voltages: VNW_N: 0 to 2 V, VPW_P: -2 to 0 V. The PVT condition used are those extracted from table 4.5.1 at the slowest point. So that 0.28 V at SS, -40°C is used for both SLVT and LVT simulations.

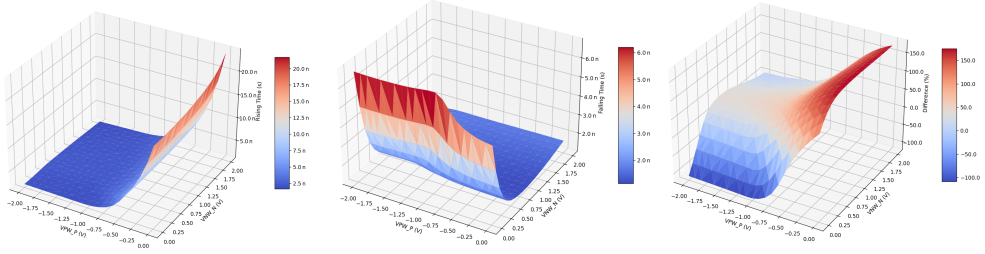
4.6.1 SLVT

Figures 4.6.1, 4.6.2, 4.6.3 show the result of sweeping through VNW_N and VPW_P at 0.28 V at SS, -40°C for an NOT, NAND and NOR gates of SLVT 28 nm technology. The results are plotted against rising time **(a)** , falling time **(b)** , and the average relative difference between rising and falling time **(c)** is computed .



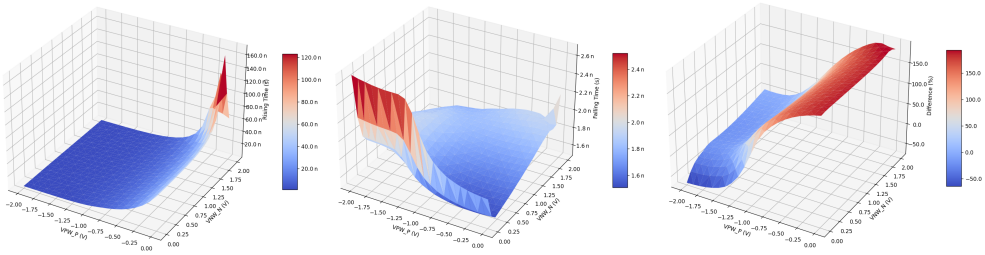
(a) Rising time. **(b)** Falling time. **(c)** Difference (%) between rising and falling time.

Figure 4.6.1: INVX010: Sweeping biasing voltages for SLVT, 28 nm, at the slow condition, -40°C, SS at 0.28 V.



(a) Rising time. **(b)** Falling time. **(c)** Difference (%) between rising and falling time.

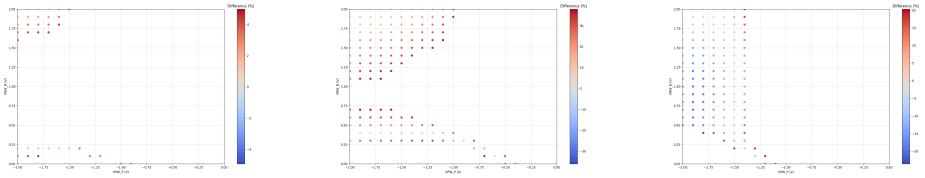
Figure 4.6.2: NAND3X010: Sweeping biasing voltages for SLVT, 28 nm, at the slow condition, -40°C, SS at 0.28 V. Inputs B and C are fixed to vdd.



(a) Rising time. **(b)** Falling time. **(c)** Difference (%) between rising and falling time.

Figure 4.6.3: NOR3X010: Sweeping biasing voltages for SLVT, 28 nm, at the slow condition, -40℃, SS at 0.28 V. Inputs B and C are fixed to gnd.

Figures 4.6.1, 4.6.2, 4.6.3 **(c)** show the relative difference between rising and falling time, so that a symmetric response is represented by 0%. A set of biasing points is allowed such that the difference is between ±threshold. The threshold is defined for each gate by looking at the liberty file, and computing the difference betwen rising and falling times decribed in the document.



(a) INV. Threshold: 5% **(b)** NAND. Threshold: 38% **(c)** NOR. Threshold: 24%

Figure 4.6.4: Skew between rising and falling time for SLVT28 basic gates.

Since the biasing points must be valid for these three gates, the results from Fig 4.6.4 are superposed in Figure 4.6.5 and their intersection is taken as valid. In other words, the biasing points where all three gate intersect are points that give a timing response that is under the desired threshold. The intersection is illustrated in Figure 4.6.6, where a line has been used to extend the biasing points in a symmetric manner, such that it is possible to test weak and strong biasing voltages. The points described by the line are printed in Table 4.6.1. They will be used later to obtain new MEP under those biasing pairs.

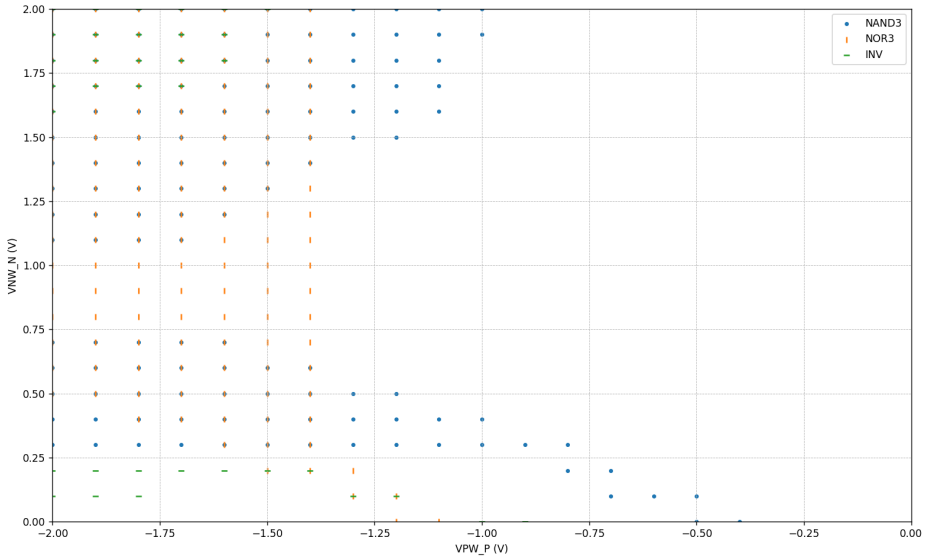


Figure 4.6.5: SLVT28 superposed skew results.

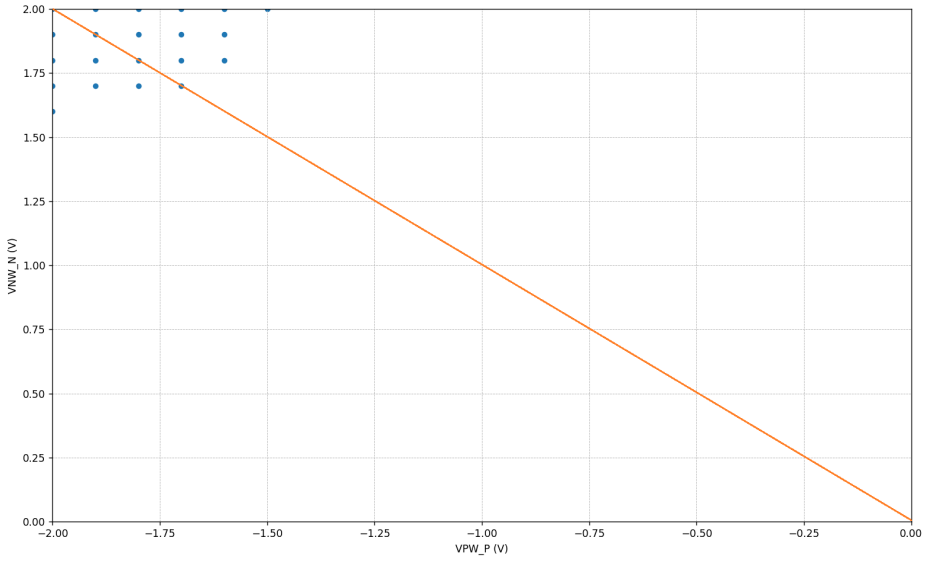


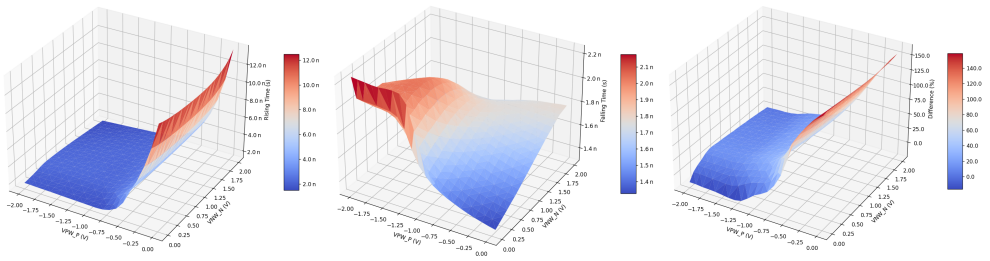
Figure 4.6.6: SLVT28 skew intersection.

VNW_N	VPW_P
2.0	-2.0
1.9	-1.9
1.8	-1.8
1.7	-1.7
1.6	-1.6
1.5	-1.5
1.4	-1.4
1.3	-1.3
1.2	-1.2
1.1	-1.1
1.0	-1.0
0.9	-0.9
0.8	-0.8
0.7	-0.7
0.6	-0.6
0.5	-0.5
0.4	-0.4
0.3	-0.3
0.2	-0.2
0.1	-0.1
0	0

Table 4.6.1: SLVT28 biasing points.

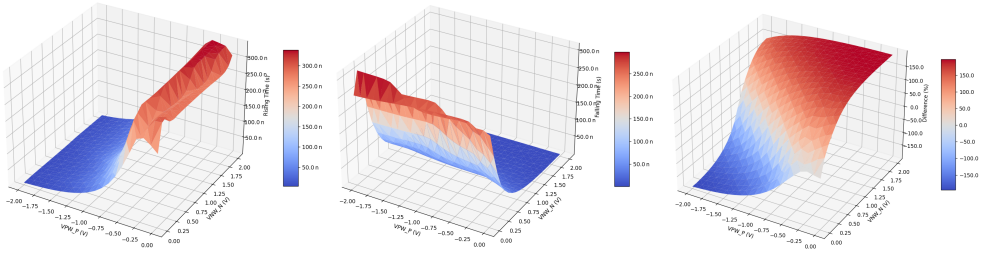
4.6.2 LVT

The exact same process is also repeated for LVT28. In this case, the results do not intersect, and so the line described by the INV is taken as valid, as it is symmetric between the NAND and NOR results.



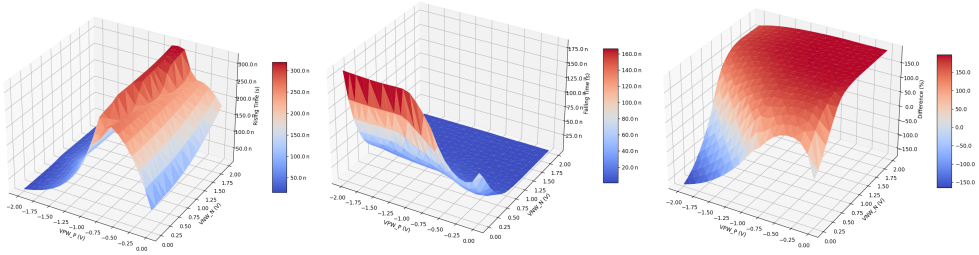
(a) Rising time. **(b)** Falling time. **(c)** Difference between rising and falling time.

Figure 4.6.7: INVX010: Sweeping biasing voltages for LVT, 28 nm, at the slow condition, -40℃, SS at 0.28 V.



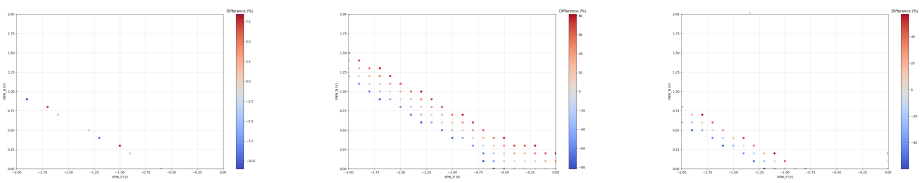
(a) Rising time. **(b)** Falling time. **(c)** Difference between rising and falling time.

Figure 4.6.8: NAND3X010: Sweeping biasing voltages for LVT, 28 nm, at the slow condition, -40°C, SS at 0.28 V. Inputs B and C are fixed to vdd.



(a) Rising time. **(b)** Falling time. **(c)** Difference between rising and falling time.

Figure 4.6.9: NOR3X010: Sweeping biasing voltages for LVT, 28 nm, at the slow condition, -40°C, SS at 0.28 V. Inputs B and C are fixed to gnd.



(a) INV. Threshold: 13% **(b)** NAND. Threshold: 83% **(c)** NOR. Threshold: 60%

Figure 4.6.10: LVT28 INV, NAND, NOR: Skew between rising and falling time under a given threshold.

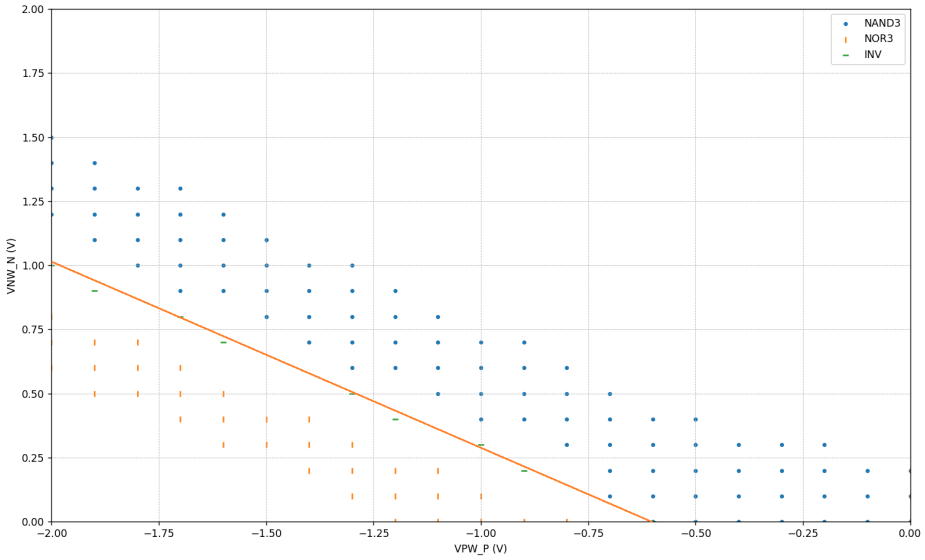


Figure 4.6.11: LVT28 superposed skew results.

VNW_N	VPW_P
1.0	-2.0
0.9	-1.9
0.8	-1.7
0.7	-1.6
0.6	-1.5
0.5	-1.3
0.4	-1.2
0.3	-1.0
0.2	-0.9
0.1	-0.7
0	-0.6

Table 4.6.2: LVT28 biasing points.

The entire exploration flow is repeated for SLVT28 and LVT28, using the PVT corners described in Table 3.5.3, with the biasing points described in Tables 4.6.1 and 4.6.2. These results are plotted in Figure 4.6.12, where the value in parenthesis are the VNW_N and VPW_P points described by the aforementioned tables. All three points for each series (Slow, Typical and Fast) are simulated under the same fixed biasing voltages. Each series uses a different biasing pair.

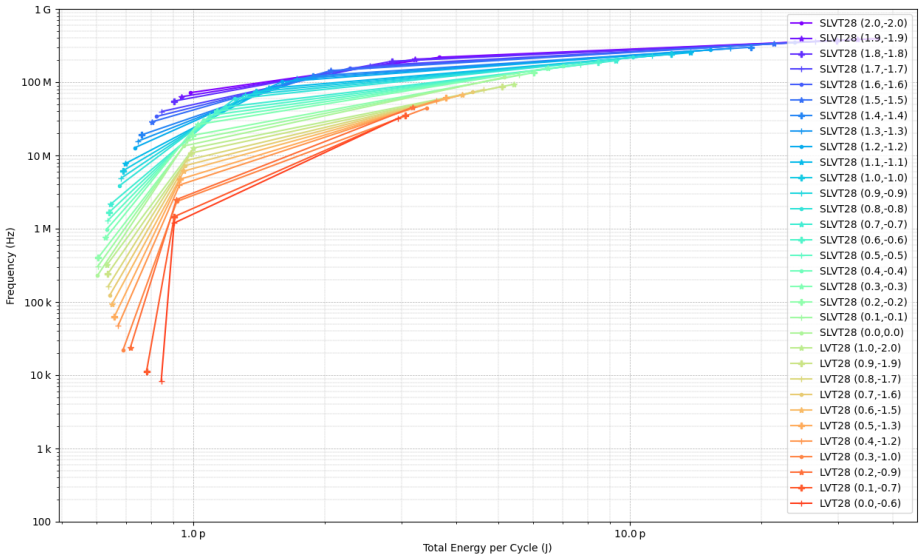


Figure 4.6.12: MEP for SLVT28 and LVT28 with different fixed body-bias. Biasing pair in parenthesis (VNW_N,VPW_P).

The resulting PVT conditions around the MEP for every single case are listed in Table 4.6.3.

Library	Slow (-10% VDD)	Typical	Fast (+10% VDD)
SLVT28 (2.0,-2.0)	0.32 V at SS -40℃	0.35 V at TT 25℃	0.38 V at FF 125℃
SLVT28 (1.9,-1.9)	0.32 V at SS -40℃	0.35 V at TT 25℃	0.38 V at FF 125℃
SLVT28 (1.8,-1.8)	0.32 V at SS -40℃	0.35 V at TT 25℃	0.38 V at FF 125℃
SLVT28 (1.7,-1.7)	0.31 V at SS -40℃	0.34 V at TT 25℃	0.38 V at FF 125℃
SLVT28 (1.6,-1.6)	0.31 V at SS -40℃	0.34 V at TT 25℃	0.38 V at FF 125℃
SLVT28 (1.5,-1.5)	0.31 V at SS -40℃	0.34 V at TT 25℃	0.38 V at FF 125℃
SLVT28 (1.4,-1.4)	0.30 V at SS -40℃	0.33 V at TT 25℃	0.36 V at FF 125℃
SLVT28 (1.3,-1.3)	0.30 V at SS -40℃	0.33 V at TT 25℃	0.36 V at FF 125℃
SLVT28 (1.2,-1.2)	0.30 V at SS -40℃	0.33 V at TT 25℃	0.36 V at FF 125℃
SLVT28 (1.1,-1.1)	0.29 V at SS -40℃	0.32 V at TT 25℃	0.35 V at FF 125℃
SLVT28 (1.0,-1.0)	0.29 V at SS -40℃	0.32 V at TT 25℃	0.35 V at FF 125℃
SLVT28 (0.9,-0.9)	0.29 V at SS -40℃	0.32 V at TT 25℃	0.35 V at FF 125℃
SLVT28 (0.8,-0.8)	0.29 V at SS -40℃	0.32 V at TT 25℃	0.35 V at FF 125℃
SLVT28 (0.7,-0.7)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
SLVT28 (0.6,-0.6)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
SLVT28 (0.5,-0.5)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
SLVT28 (0.4,-0.4)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
SLVT28 (0.3,-0.3)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
SLVT28 (0.2,-0.2)	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
SLVT28 (0.1,-0.1)	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
SLVT28 (0.0,0.0)	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
LVT28 (1.0,-2.0)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.9,-1.9)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.8,-1.7)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.7,-1.6)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.6,-1.5)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.5,-1.3)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.4,-1.2)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.3,-1.0)	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
LVT28 (0.2,-0.9)	0.28 V at SS -40℃	0.31 V at TT 25℃	0.34 V at FF 125℃
LVT28 (0.1,-0.7)	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃
LVT28 (0.0,-0.6)	0.27 V at SS -40℃	0.30 V at TT 25℃	0.33 V at FF 125℃

Table 4.6.3: PVT variations around MEP with FBB for SLVT and LVT28. Biasing pair in parenthesis (VNW_N,VPW_P)

4.6.3 Adaptive Body-Biasing

Figure 4.6.12 presented in previous section show a big spread between slow and fast corners, simulated at the same biasing voltages. To flatten the curve, to be able to close timing, different FBB can be used at slow, typical and fast corners. This is illustrated in Figure 4.6.13. The strongest biasing is applied to the slowest corner, while the weakest one is applied to the fast corner. The biasing for the typical corner is chosen to be somewhere in between. The results are compared against the green line, which represents the operating point characterized by the library

at 0.36 V (-40℃) 0.4 V (25℃) and 0.44 V (125℃) using the biasing suggested by the foundry.

The results in 4.6.13 cannot be directly compared with 4.6.12 for a same pair of body-bias voltages, because the frequencies and energy would not match. This is because the MEP is located at the typical case, and then the supply voltages of the slow and fast corners are computed. So that they are operating at different supply voltages. For instance, the slow corner of the SLVT28 curve with biasing (2,-2) in Figure 4.6.12, is located at 70 MHz and 0.32 V, because the MEP was located at 0.35 V. But when the new MEP is computed using biasing (0.8,-0.8) the MEP is shifted to 0.32 V, and so the slow corner (with bias 2,-2) is now located at 0.29 V running at 40 MHz. The drop in frequency is explained by the drop in supply voltage to accommodate for the new MEP.

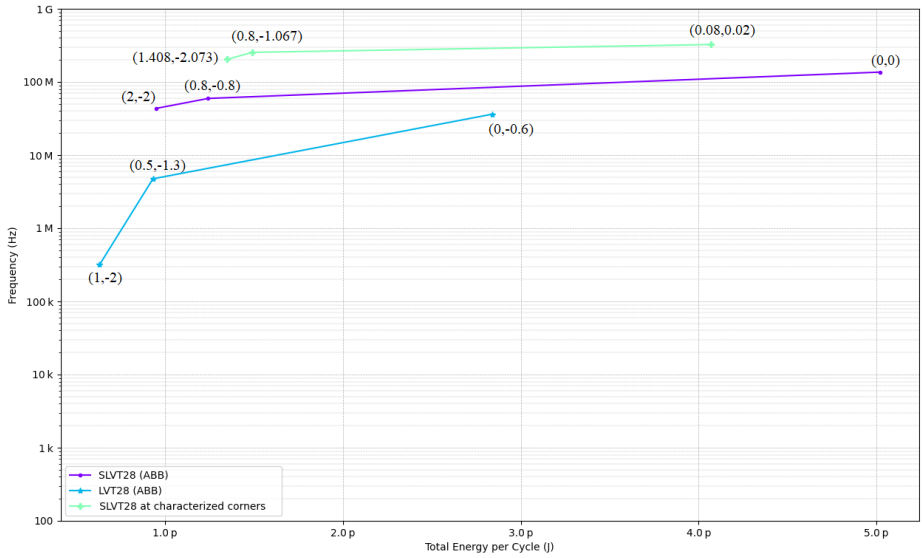


Figure 4.6.13: MEP for SLVT28 and LVT28 using Adaptive Body-Bias. Biasing pair in parenthesis (VNW_N,VPW_P).

Library	Slow (-10% VDD)	Typical	Fast (+10% VDD)
SLVT28 (ABB)	0.29V at SS -40℃(2,-2)	0.32V at TT 25℃ (1,-1)	0.35V at FF 125℃ (0,0)
SLVT28 (At lib corners)	0.36V at SS -40℃(1.408,-2.073)	0.40V at TT 25℃ (0.8,-1.067)	0.44V at FF 125℃ (0.08,0.02)
LVT28 (ABB)	0.28V at SS -40℃(1,-2)	0.31V at TT 25℃ (0.5,-1.3)	0.34V at FF 125℃ (0,-0.6)

Table 4.6.4: PVT variations around MEP with ABB for SLVT28 and LVT28.

CHAPTER

FIVE

DISCUSSION

The first thing to notice is that the critical path replica, used as a ring oscillator to estimate the MEP, is bound to impose limitations on the results. As it is shown in Table 4.2.1, the RO does not use more than 23% unique cells compared against the synthesized Ibex Core. It means that the majority of the standard cells are not evaluated through analog simulations. Moreover, in some particular cases only 3% of the cells are simulated, because the critical path is composed mainly by half-adders that are part of a counter module. In these cases, the analog simulation is basically evaluating how a chain of half-adders are affected by DFVS and PVT variations. Nonetheless, the critical path was considered a representative sample.

5.1 MEP and voltage scaling

The results provided in section 4.4 are very interesting, as they provide the basic foundations on the theory of Minimum Energy Point. It is demonstrated that operating at the MEP provides great opportunities for energy savings. In this particular RTL design, 23% reduced energy if compared against the typical corner where the LVT library is characterized (0.4 V), but even greater savings, more that 80%, if compared against that 0.8V corner (where the HVT library is characterized). However, it is shown that this does not come for free, the reduction in energy is paid by an even higher reduction in speed. It also becomes clear that operating under the MEP supply voltage is of no benefit at all. The leakage energy is so high, and the frequency of operation so slow, that it is worse than operating at high voltages. That is, for voltages below the MEP, voltage scaling is counterproductive.

Section 4.4, specifically Figure 4.4.4 contains some surprising findings. UHVT

technology is the worst performing of all. It might seem counter intuitive because UHVT transistor are designed to have low leakage currents. But they are also know to be slower. In fact, they are so slow near the MEP, that the reduced power dissipated is contrasted with longer times per operation, so that the energy consumption is higher. It can be seen that this technology is about 1000 times slower, and consuming almost as twice as the LVT technology. This figure also shows that HVT, although superior to UHVT, is still inferior both in speed and energy savings compare to SLVT and LVT. Meanwhile, both SLVT and LVT show good results, the first one can be use to target higher speeds, while the former to aim for greater energy savings.

It was initially thought that excluding from the synthesis flow the cells that scale poorly with voltage might bring benefits in regards the MEP, so that this could be an promising research area. However, section 4.4.1 shows that excluding cells from synthesis does not produce any significant saving in energy, or increase in speed. On the contrary, in most of the cases, limiting the options to the synthesis tool would yield worse results. There are some conditions where the design cannot even be synthesized due to lack of proper gates that implement a given functionality. The study can be repeated by limiting the fan-in to a maximum of two, as suggested by [21], [34], as they show better stability at ultra-low voltages.

5.2 MEP and temperature

In section 4.5, the effect of temperature was studied, in particular Figure 4.5.1 is showing that the MEP is forced to move to higher supply voltages with higher energy consumption as the temperature is increased. This is consistent with the results presented at [21], studying the MEP of an adder and multiplier with SOTB technology.

Also in section 4.5, the impact of temperature and process variation was explored. It can be deduced from Figure 4.5.4 and Table 4.5.2 that as the channel length of the transistor increases, it pushes the MEP down in the energy, frequency and voltage scales. But what is really important in this section is to study the spread between slow and fast conditions. As a circuit must be able to operate in a wide range of environments, these type of figures show how hard it is to comply with timing constraints around the MEP. LVT libraries show almost two orders of magnitude difference in speed, while SLVT shows a three orders of magnitude difference between the slow and fast corners. For instance, if the design was to be synthesized with a LVT36 technology and operated around the MEP, at the slow condition (-10% of VDD, SS process at -40℃), the clock would run at 20 kHz, but if it was operating under a fast condition (+10% of VDD, FF process at 125℃), the clock would run at 30 MHz. So that the designers must take into account this huge difference in the clock speed and design accordingly.

When the study is expanded to HVT and UHVT technologies, the situation

is even worse, this is shown in Figure 4.5.5. The critical path replica does not oscillate for any of the UHVT libraries at the slow condition, making this technology completely useless in this particular case. An Ibex Core synthesized with this technology and operating at slow condition of the MEP would not properly work. The circuit synthesized with HVT libraries does work, but with an even higher spread, about four orders of magnitude difference in speed. Based on these results, these two technologies were dropped from any further research.

It is interesting to think in what would happen if peripheral are connected to or disconnected from the original core, such that when the peripherals are in idle state, the switching energy remains the same, but the leakage energy changes. In such case, what would happen with the MEP. Figure 4.5.6 sheds some lights about it. The first observation is that as the leakage increases, the tendency is that the MEP is forced to move into the upper-right corner. That is, the total energy consumed increases, due to a higher leakage energy dissipation. The supply voltage that targets a MEP must also be raised, because the leakage energy takes dominance at supply voltages higher than before. With higher supply voltages, the frequency increases as well. In summary, adding components that do not contribute to the switching power, will push the MEP into higher energy, supply voltage and frequency scales. It is also important to notice in this figure that the relative position is kept between each MEP with different libraries and channel length. It means that once the best library is selected (either targeting speed or energy saving), it can be expected to remain the best library even if additional peripherals are added.

5.3 MEP and FBB

Section 4.6 goes into the exploration of the MEP with different body biasing. The first thing to highlight here is the selected threshold for deciding whether or not a biasing pair (VNW_N, VPW_P) is valid to be used in the MEP exploration flow. The threshold is extracted from the liberty file according to the timing matrix defined by the foundry at their recommended biasing points. But is is unknown what was the performance metric they used to select the biasing points they recommend, because it was verified they do not target neither symmetrical rising and falling time, nor 50% duty cycle. Nonetheless, the threshold will be considered as valid, as it is the same used by the commercial available libraries. Moreover, the valid biasing points were extracted from simulating only three basic gates (NAND, NOR, INV), so that there is already a limitation due to unrepresented cells.

With these new biasing voltages, the MEP is recomputed and show in Figure 4.6.12. It can be clearly seen that increasing a FBB will push the MEP into higher frequencies and energy consumption, both slow and fast corner also become faster with higher leakage. This effect is explain due to a lowered VT when applying a FBB technique.

A clear separation between LVT and SLVT is visible. Not even a circuit synthesized with LVT with the strongest bias can reach the speed of a circuit synthesized with SLVT with a weak bias (zero bias). Moreover, if the spread between slow and fast corners is analyzed, it can be seen that SLVT has a flatter response compared to LVT, meaning that it is easier to meet timing constraints with a SLVT technology. So that the selection of VT becomes a key parameter at design time.

In order to flatten the curve and close the gap between slow and fast conditions, Adaptive Body-Biasing (ABB) can be used. In such case a strong bias is applied to the slow corner to raise the speed, while a weak bias is applied to the fast corner to keep the frequency low. The typical point is selected to be somewhere in the middle, this is illustrated in Figure 4.6.13. There is a limitation imposed by the technology on how strong the biasing can be, the range is up to 2 V and -2 V. It can be seen that the purple curve (SLVT) has a much more flatten behavior than the blue one (LVT). For the SLVT case, the spread is less than one order of magnitude, whilst LVT shows almost a two order of magnitude difference. It can be concluded, that SLVT becomes the best performing library near the MEP, even when it consumes more energy, because it is easier to use to close timing constraints. Although ABB is introduced in this thesis, this work is not studying how to implement an ABB loop to adapt to PVT variations, but rather just exploring the effect of FBB and locating some valid biasing points.

CHAPTER

SIX

CONCLUSIONS

This thesis presented a methodology to obtain the Minimum Energy Point (MEP) of any RTL design by means of SPICE simulations, where the impact of supply voltage, threshold voltage, channel length, temperature, process variations, and body biasing was analyzed.

The study has shown the advantage of operating at the MEP, more than 20% of energy can be saved when compared to the typical point for the available LVT library at 0.4 V. However, it has also been stated that PVT variations have great impact around MEP voltages, causing difficulties for timing closure. It was deduced that SLVT libraries are the best suited for operation near the MEP, due to higher speeds and closer gap between slow and fast corners. Whilst HVT and UHVT are so slow and have such a huge timing gap that its usage is unfeasible. It was also concluded that the main parameter when targeting for MEP operation is selecting the right VT flavor, later a proper channel length selection can be used to target a more specific range of frequencies and energy. Through Adaptive Body-Biasing (ABB), it is possible to compensate at the slow and fast corners to get closer spreads in frequency.

These results add to the expanding field of low-energy designs by showing a basic guideline to locate the MEP when using commercial standard cell libraries characterized for higher voltages.

6.1 Future work

In order to test the results provided in this thesis, some further work is needed. A new library can be characterized around the MEP point, including worst, typical

and best conditions. With this new library, the Ibex Core can be fully synthesized again an cross-check the results against the power estimation and timing analysis extracted from PrimePower. The circuit can then be implemented in hardware and put to test under the same conditions.

The MEP exploration can also be extended to a Multi-VT circuit, that is, using different VT flavors within different cells in the same circuit, and studying how it affect the Minimum Energy Point.

Lastly, since this is a very iterative process, all the different scripts and tools should be fully automated and optimized to converge to the results in a timely manner.

REFERENCES

[1] Heath Steve. *Embedded Systems Design*. Vol. 2nd ed. Newnes, 2003. Chap. 1 - What is an embedded system? ISBN: 9780750655460. URL: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=196270&site=ehost-live&scope=site.

[2] Federica Laricchia. *Number of mobile devices worldwide 2020-2025*. Mar. 2023. URL: https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/.

[3] Inc The RADICATI Group. *Mobile Statistics Report, 2021-2025*. Jan. 2021. URL: https://www.radicati.com/wp/wp-content/uploads/2021/Mobile_Statistics_Report,_2021-2025_Executive_Summary.pdf.

[4] Lionel Sujay Vailshery. *IOT connected devices worldwide 2019-2030*. Nov. 2022. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[5] John Catsoulis. *Designing embedded hardware*. 2nd ed. O'Reilly Media, May 2005. Chap. 1 - An Introduction to Computer Architecture.

[6] RISC-V. *The RISC-V Instruction Set Manual*. Version 20191213. Dec. 2019. Chap. 28 - History and Acknowledgments.

[7] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and systems perspective*. 4th ed. Pearson, 2011. Chap. 5. ISBN: 9780321547743.

[8] B.H. Calhoun, A. Wang, and A. Chandrakasan. "Modeling and sizing for minimum energy operation in subthreshold circuits". In: *IEEE Journal of Solid-State Circuits* 40.9 (2005), pp. 1778–1786. DOI: 10.1109/JSSC.2005.852162.

[9] D. Halliday, R. Resnick, and J. Walker. *Fundamentals of Physics*. Fundamentals of Physics. Wiley, 2013. Chap. 26-5 Power, semiconductors, superconductors. ISBN: 9781118230718.

[10] Wayne H. Cheng and Bevan M. Baas. "Dynamic voltage and frequency scaling circuits with two supply voltages". In: *2008 IEEE International Symposium on Circuits and Systems*. 2008, pp. 1236–1239. DOI: 10.1109/ISCAS.2008.4541648.

[11] Antoni Ferré and Joan Figueras. "Leakage in CMOS Nanometric Technologies". In: *Low-Power CMOS Circuits*. Ed. by Christian Piguet. CRC Press, Oct. 2018. Chap. 3. DOI: 10.1201/9781315220710. URL: https://doi.org/10.1201/9781315220710.

[12] Synopsys. *Power Compiler User Guide*. Version D-2010.03-SP2. 2010. Chap. Power Modeling and Calculation.

[13] S. Borkar. "Low power design challenges for the decade". In: *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No.01EX455)*. 2001, pp. 293–296. DOI: 10.1109/ASPDAC.2001.913321.

[14] Kaushik Roy, Amit Agarwal, and Chris H. Kim. "Circuit Techniques for Leakage Reduction". In: *Low-Power CMOS Circuits*. Ed. by Christian Piguet. CRC Press, Oct. 2018. Chap. 13. DOI: 10.1201/9781315220710. URL: https://doi.org/10.1201/9781315220710.

[15] A. Wang, A.P. Chandrakasan, and S.V. Kosonocky. "Optimal supply and threshold scaling for subthreshold CMOS circuits". In: *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*. 2002, pp. 7–11. DOI: 10.1109/ISVLSI.2002.1016866.

[16] Even Låte. "Low Voltage Logic and Memories on Silicon". PhD thesis. Norwegian University of Science and Technology, 2021. Chap. 4.2. ISBN: 9788232669714. URL: https://hdl.handle.net/11250/2740488.

[17] M.J.M. Pelgrom, A.C.J. Duinmaijer, and A.P.G. Welbers. "Matching properties of MOS transistors". In: *IEEE Journal of Solid-State Circuits* 24.5 (1989), pp. 1433–1439. DOI: 10.1109/JSSC.1989.572629.

[18] STMicroelectronics. *FD-SOI: Fully Depleted Silicon On Insulator*. URL: https://www.st.com/content/st_com/en/about/innovation---technology/FD-SOI.html.

[19] Sylvain Clerc, Thierry Di Gilio, and Andreia Cathelin, eds. *The Fourth Terminal: Benefits of Body-Biasing Techniques for FDSOI Circuits and Systems*. Springer International Publishing, 2020. Chap. 2. DOI: 10.1007/978-3-030-39496-7. URL: https://doi.org/10.1007/978-3-030-39496-7.

[20] Vratislav Michal. "On the low-power design, stability improvement and frequency estimation of the CMOS ring oscillator". In: *Proceedings of 22nd International Conference Radioelektronika 2012*. 2012, pp. 1–4.

[21] Shohei Nakamura et al. "Measurement of the minimum energy point in Silicon on Thin-BOX(SOTB) and bulk MOSFET". In: *EUROSOI-ULIS 2015: 2015 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon*. 2015, pp. 193–196. DOI: 10.1109/ULIS.2015.7063746.

[22] Khyati Kiyawat et al. "Real-Time Minimum Energy Point Tracking Using a Predetermined Optimal Voltage Setting Strategy". In: *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2020, pp. 415–421. DOI: 10.1109/ISVLSI49217.2020.00082.

[23] Daniel Guarecuco Aguiar. *Minimum Energy Point Exploration in a CPU Subsystem*. Tech. rep. NTNU, 2022.

[24] *RISC-V Sees Significant Growth and Technical Progress in 2022 with Billions of RISC-V Cores in Market*. Dec. 2022. URL: https://riscv.org/announcements/2022/12/risc-v-sees-significant-growth-and-technical-progress-in-2022-with-billions-of-risc-v-cores-in-market/.

[25] lowRISC. *Ibex RISC-V Core*. Commit 75a93dbed0dbe340b637c82e2a8e3ba1b841ecc3. 2022. URL: https://github.com/lowRISC/ibex (visited on 10/14/2022).

[26] FuseSoc. *FuseSoc Documentation*. 2022. URL: https://fusesoc.readthedocs.io/en/stable/ (visited on 10/14/2022).

[27] Synopsys. *Prime Power User Guide*. Version Q-2019.12. 2019. Chap. Getting Started, and Annotating Switching Activity.

[28] Synopsys. *Understanding Your Power Profile from RTL to Gate-level Implementation*. URL: https://www.synopsys.com/designware-ip/technical-bulletin/understanding-power-profile.html.

[29] Synopsys. *Design Compiler User Guide*. Version R-2020.09. 2020. Chap. Working with Design Compiler: The synthesis flow.

[30] Cristinel Ababei and Chandana Tamma. "Distributed minimum energy point tracking for systems-on-chip". In: *IEEE International Conference on Electro/Information Technology*. 2014, pp. 246–251. DOI: 10.1109/EIT.2014.6871770.

[31] Cadence. *Design Data Translators Reference*. Version ICADVM20.1. 2022. Chap. 4 - Netlist Import Using Spice In.

[32] Cadence. *OCEAN Reference*. Version ICADVM20.1. 2021.

[33] Cadence. *OCEAN XL Reference*. Version ICADVM20.1. 2021.

[34] Somayeh Hossein Zadeh, Trond Ytterdal, and Snorre Aunet. "Comparison of Ultra Low Power Full Adder Cells in 22 nm FDSOI Technology". In: *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. 2018, pp. 1–5. DOI: 10.1109/NORCHIP.2018.8573516.

APPENDICES

A - REPORTS

.1 Critical path netlist

There is a different critical path for every synthesized circuit with different threshold voltage, channel length and number of cells excluded. That is, more than 30 netlists. Due to their size, they cannot be printed here, instead, an example is provided in a zip file with the submission of this thesis.

The file *critical_path_netlist.spi* contains the entire netlist, including capacitors and voltage sources for wire renaming. *timing_report_critical_path.txt* contains the timing report, only showing the input and output of each cell in the path.

.2 Not annotated switching activity report

This section present the not annotated switching activity report extracted from PrimePower. Gray box shows 100% of annotated activity from the activity file extracted from RTL simulations. See next page.

Listing 1: Not annotated switching activity report

```
****************************************
Report : Switching Activity
        -list_not_annotated
Design : ibex_top
Version: Q-2019.12-SP5
Date   : Sun Feb 26 20:09:11 2023
****************************************

 Switching Activity Overview Statistics for "ibex_top"
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
                  From Activity    From         From          From             From            From                                                                 Not
Object Type       File (%)         SSA (%)      SSA Force (%) SSA Force (%)    SCA (%)         Clock (%)       Default (%)     Propagated(%)   Implied(%)      Annotated(%)  Total
                                                Annotated     Implied
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Nets              2388(100.00%)    0(0.00%)     0(0.00%)     0(0.00%)         0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)      2388
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Nets Driven by
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Primary Input     379(100.00%)     0(0.00%)     0(0.00%)      0(0.00%)         0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)      379
Tri-State         0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Black Box         0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Sequential        2009(100.00%)    0(0.00%)     0(0.00%)      0(0.00%)         0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)      2009
Combinational     0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Memory            0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Clock Gate        0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Static Probability Overview Statistics for "ibex_top"
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
                  From Activity    From         From          From             From            From                                                                 Not
Object Type       File (%)         SSA (%)      SSA Force (%) SSA Force (%)    SCA (%)         Clock (%)       Default (%)     Propagated(%)   Implied(%)      Annotated(%)  Total
                                                Annotated     Implied
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Nets             2388(100.00%)    0(0.00%)     0(0.00%)      0(0.00%)         0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)      2388
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Nets Driven by
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Primary Input     379(100.00%)     0(0.00%)     0(0.00%)      0(0.00%)         0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)      379
Tri-State         0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Black Box         0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Sequential        2009(100.00%)    0(0.00%)     0(0.00%)      0(0.00%)         0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)        0(0.00%)      2009
Combinational     0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Memory            0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
Clock Gate        0(0%)            0(0%)        0(0%)         0(0%)            0(0%)           0(0%)           0(0%)           0(0%)           0(0%)           0(0%)         0
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


List of nonannotated nets :
1
```

.3 Data extracted from ring oscillator with open-loop: input to VDD

This sections presents an example on the data extracted from the ring oscillator simulation and saved in a CSV file when the switch is connected to VDD, such that there is no oscillation, notice the *eval err* in the Frequency row. The simulation is run at 25℃, TT model, with vdd sweeping from 0.2 V to 0.8 V for a LVT28 library. The Power row represents the static power consumption. Some parameters have been redacted to protect sensitive information. Likewise, some columns were removed so that the table can fit in one page.

	Parameter	tc.2_25	tc.21_25	tc.22_25	tc.23_25	tc.24_25	tc.25_25	tc.26_25	tc.27_25	tc.28_25	tc.29_25	tc.3_25
	process	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	temperature	25	25	25	25	25	25	25	25	25	25	25
	vdd	0.2	0.21	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.29	0.3
	vnw_n	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	vpw_p	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067
Test	Output	tc.2_25	tc.21_25	tc.22_25	tc.23_25	tc.24_25	tc.25_25	tc.26_25	tc.27_25	tc.28_25	tc.29_25	tc.3_25
LVT28	Frequency	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err
LVT28	DutyCycle	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err
LVT28	Power	12.0E-9	12.9E-9	13.8E-9	14.7E-9	15.7E-9	16.7E-9	17.8E-9	18.8E-9	20.0E-9	21.2E-9	22.4E-9
LVT28	Delay	1.7E-6	1.3E-6	1.0E-6	801.7E-9	624.0E-9	485.6E-9	377.9E-9	294.3E-9	229.6E-9	179.5E-9	140.8E-9
LVT28	SimTime	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3

	Parameter	tc.31_25	tc.32_25	tc.33_25	tc.34_25	tc.35_25	tc.36_25	tc.38_25	tc.4_25	tc.5_25	tc.6_25	tc.8_25
	process	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	temperature	25	25	25	25	25	25	25	25	25	25	25
	vdd	0.31	0.32	0.33	0.34	0.35	0.36	0.38	0.4	0.5	0.6	0.8
	vnw_n	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.542	0.346	-0.024
	vpw_p	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-0.632	-0.21	0.2
Test	Output	tc.31_25	tc.32_25	tc.33_25	tc.34_25	tc.35_25	tc.36_25	tc.38_25	tc.4_25	tc.5_25	tc.6_25	tc.8_25
LVT28	Frequency	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err
LVT28	DutyCycle	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err	eval err
LVT28	Power	23.7E-9	25.0E-9	26.3E-9	27.8E-9	29.2E-9	30.8E-9	34.0E-9	37.5E-9	29.1E-9	24.2E-9	28.2E-9
LVT28	Delay	110.9E-9	87.9E-9	70.1E-9	56.3E-9	45.6E-9	37.3E-9	25.6E-9	18.3E-9	7.2E-9	3.9E-9	1.8E-9
LVT28	SimTime	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3	400.0E-3

Table .3.1: Data from ring oscillator in open-loop: LVT28, TT, 25℃, input to VDD.

.4 Data extracted from ring oscillator with closed-loop

This sections presents an example on the data extracted from the ring oscillator simulation and saved in a CSV file when the switch is connected to !Z, such that there is an oscillatory behavior. The simulation is run at 25℃, TT model, with vdd sweeping from 0.2 V to 0.8 V for a LVT28 library. The Power row represents the total power consumption. Some parameters have been redacted to protect sensitive information. Likewise, some columns were removed so that the table can fit in one page.

	Parameter	tc.2_25	tc.21_25	tc.22_25	tc.23_25	tc.24_25	tc.25_25	tc.26_25	tc.27_25	tc.28_25	tc.29_25	tc.3_25
	process	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	temperature	25	25	25	25	25	25	25	25	25	25	25
	vdd	0.2	0.21	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.29	0.3
	vnw_n	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	vpw_p	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067
Test	Output	tc.2_25	tc.21_25	tc.22_25	tc.23_25	tc.24_25	tc.25_25	tc.26_25	tc.27_25	tc.28_25	tc.29_25	tc.3_25
LVT28	Frequency	315.2E+3	403.7E+3	517.8E+3	665.0E+3	854.7E+3	1.1E+6	1.4E+6	1.8E+6	2.3E+6	3.0E+6	3.8E+6
LVT28	DutyCycle	53.48	53.5	53.52	53.54	53.55	53.55	53.54	53.52	53.48	53.43	53.36
LVT28	Power	18.2E-9	21.8E-9	26.5E-9	32.7E-9	41.1E-9	52.4E-9	67.7E-9	88.3E-9	116.2E-9	153.6E-9	203.6E-9
LVT28	Delay	1.7E-6	1.3E-6	1.0E-6	801.7E-9	624.0E-9	485.6E-9	377.9E-9	294.3E-9	229.6E-9	179.5E-9	140.8E-9
LVT28	SimTime	31.7E-6	24.8E-6	19.3E-6	15.0E-6	11.7E-6	9.1E-6	7.1E-6	5.5E-6	4.3E-6	3.4E-6	2.6E-6

	Parameter	tc.31_25	tc.32_25	tc.33_25	tc.34_25	tc.35_25	tc.36_25	tc.38_25	tc.4_25	tc.5_25	tc.6_25	tc.8_25
	process	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT	TT
	temperature	25	25	25	25	25	25	25	25	25	25	25
	vdd	0.31	0.32	0.33	0.34	0.35	0.36	0.38	0.4	0.5	0.6	0.8
	vnw_n	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.542	0.346	-0.024
	vpw_p	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-1.067	-0.632	-0.21	0.2
Test	Output	tc.31_25	tc.32_25	tc.33_25	tc.34_25	tc.35_25	tc.36_25	tc.38_25	tc.4_25	tc.5_25	tc.6_25	tc.8_25
LVT28	Frequency	4.8E+6	6.1E+6	7.6E+6	9.5E+6	11.7E+6	14.2E+6	20.6E+6	28.8E+6	72.4E+6	133.4E+6	284.5E+6
LVT28	DutyCycle	53.27	53.17	53.06	52.94	52.81	52.68	52.42	52.17	51.55	51.15	50.4
LVT28	Power	269.8E-9	356.7E-9	469.4E-9	613.8E-9	796.4E-9	1.0E-6	1.6E-6	2.5E-6	10.0E-6	26.8E-6	104.3E-6
LVT28	Delay	110.9E-9	87.9E-9	70.1E-9	56.3E-9	45.6E-9	37.3E-9	25.6E-9	18.3E-9	7.2E-9	3.9E-9	1.8E-9
LVT28	SimTime	2.1E-6	1.6E-6	1.3E-6	1.1E-6	858.6E-9	703.0E-9	485.0E-9	347.7E-9	138.2E-9	75.0E-9	35.2E-9

Table .4.1: Data from ring oscillator in closed-loop: LVT28, TT, 25℃, input to !Z.

B - SCRIPTS

Most important scripts are presented here. Some directories and library names have been redacted to protect sensitive information.

Files are included in zip file together with this document.

.5 Main script

Most of the flow is automated and is run from here.

```
#!/bin/bash

# Author: Daniel Guarecuco Aguiar <deaguiar@stud.ntnu.no>
#   <daniel.aguiar@nordicsemi.no>
# This is the main script
# 1.Build Ibex Software
# 2.Run RTL simulation
# 3.Run Synthesis
# 4.Run Power estimation, export netlist
# 5.Clean netlist and create testbenches
# 6.Import netlist and run analog Simulation
# # ----------------------------------------------------------------------------

# Library name and Period in ps
#   SLVT20 3000
#   SLVT24 4000
#   SLVT28 4000
#   SLVT32 5000
#   SLVT36 6000
#   LVT24 15000
#   LVT28 18000
#   LVT32 24000
#   LVT36 28000
#   HVT20 1760
#   HVT24 2040
#   HVT28 2300
#   UHVT28 6000
#   UHVT32 7150
#   UHVT36 8900


arg_lib=$1 #Library name
arg_period=$2 #Period
exclude_cells=$3 # % to exclude (0,20,30,40,50)
arg1=$4 #Options: syn import analogSim
################################################################################
#                   EDIT THIS ACCORDINGLY
################################################################################
array_lib=(
    "${arg_lib} ${arg_period}"
)

project_path="/my_path/mep_thesis"
make_sw_path="${project_path}/design/ibex/examples/sw/simple_system/hello_test"
build_path="${project_path}/build/lowrisc_ibex_ibex_simple_system_0/sim-modelsim"

#Matches SLVT and LVT
```

```
regex1='(.*)_([a-zA-Z]+VT[0-9]+)_(.+)_0d([0-9]+)(V_m?[0-9]+C)(.+)\s+(\w+)'
#Matches HVT and UHVT lib
regex2='(.*)_([a-zA-Z]+[0-9]+[a-zA-Z]+)(_TT_0P)([0-9]+V).+(_[0-9]+C)(.+)\s+(\w+)'

#*******************************************************************************
# Setting environment
module load synopsys/verdi/2021.09-sp2-2

export PATH=$PATH\:$HOME/.local/bin
export PATH=$PATH\:$HOME/lowrisc-toolchain/bin
export ARCH=rv32imc
export MODEL_TECH=/path_tool/questasim/2022.3/questasim/bin


export exclude_cells=$exclude_cells
#125, 25, m40
temp=25
#*******************************************************************************
# Starts here

# Loop through libraries
for i in "${array_lib[@]}"
do
    #Evalute lib file name against regex
    if [[ "$i" =~ $regex1 ]]; then
        echo "Matches lib"
        PERIOD=$(( ${BASH_REMATCH[7]}/1000 ))
        echo "Period is ${PERIOD} ns"
    elif [[ "$i" =~ $regex2 ]]; then
        echo "Matches lib"
        PERIOD=$(( ${BASH_REMATCH[7]} ))
        echo "Period is ${PERIOD} ps"
    else
        echo "No Match - Library name invalid"
        exit
    fi
    #Get lib name
    LIB_NAME="${BASH_REMATCH[2]}_0d${BASH_REMATCH[4]}${BASH_REMATCH[5]}_${exclude_cells}"
    #Set output directory (e.g /my_path/mep_thesis/syn_out/LVT24_25C)
    OUT_DIR="${project_path}/analysis/syn_out/$LIB_NAME"

    #Set env variable used by tcl scripts
    export OUT_DIR=$OUT_DIR
    #Example LVT24
    TECH_CHAN="${BASH_REMATCH[2]}"
    export TECH_CHAN=$TECH_CHAN
    TESTBENCH="${TECH_CHAN}_Testbench"
    stringarray=($i)
    #Example lib_name.nldm.db
    export TECH_LIB=$stringarray
    export PERIOD=$PERIOD
    SIM_PERIOD=$(( ${BASH_REMATCH[7]} ))

    echo "*******************************************************************************"
    echo "RUNNING FLOW USING $i"
    echo "*******************************************************************************"

    current_dir=$PWD
    cd $project_path

    if [ "$arg1" = "syn" -o "$arg1" = "all"  ];
    then
        ## Make software
        echo "Making software"
        make -C $make_sw_path
        ## Generate VMEM from binary
        echo "Generating VMEM"
        hexdump -v -e '1/4 "%08x " "\n"' $make_sw_path/hello_test.bin > $make_sw_path/hello_test.vmem
        ## Build Simulation
        echo "Building Simulation"
        fusesoc --cores-root=. run --target=sim --tool=modelsim --setup --build lowrisc:ibex:ibex_simple_system --RV32E=0
        ↪  --RV32M=ibex_pkg::RV32MFast --SRAMInitFile="${make_sw_path}/hello_test.vmem" --Period=$SIM_PERIOD
        ## Run simulation
        echo "Running Simulation"
        cd $build_path
        make run
        echo "Simulation Log:"
        cat ibex_simple_system.log
        cd $current_dir;
        ## Create output directory
        mkdir -p $OUT_DIR
        ## Copy FSDB
        cp "${build_path}/ibex_simple_system.fsdb" ${OUT_DIR}

        ## Run sythesis
        echo "STARTING DC_SHELL..."
        dc_shell -f syn.tcl -o $OUT_DIR/compile.log

        ## Export histogram
        echo "EXPORTING CELL HISTOGRAM..."
        grep "LVT_\|HVT_" $OUT_DIR/ibex_top_map.v | awk '{print $1}' | sort | uniq -c | sort -nr | awk '{print $2 "," $1}' >
        ↪  $OUT_DIR/cell_histogram_ibex_$LIB_NAME.csv

        ## Run power stimation
        echo "STARTING PT_SHELL..."
        pt_shell -f power.tcl
    fi

    if [ "$arg1" = "import" ];
```

```
    then
        cd $project_path/analysis
        ## Build SPICE testbench
        echo "Building spice testbench"
        python3 generate_testbench.py $OUT_DIR

        ## Build OceanScript for simulation
        python3 generate_ocn.py $OUT_DIR $TECH_CHAN $temp

        ## Importing to Virtuoso
        cd $project_path/virtuoso
        echo "Importing netlist to Virtuoso"
        echo "***********************************Creating Testbench"
        spiceIn -netlistFile $OUT_DIR/netlist_testbench.spi -outputLib $TESTBENCH -language SPICE -refLibList "cmos22fdsoi
        ↪  analogLib" -devMapFile /my_path/mep_thesis/virtuoso/mapping
    fi

    if [ "$arg1" = "analogSim" ];
    then
        temp=25
        echo "Simulating for 25C"
        cd virtuoso
        echo "Simulating for input connected to gnd"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_1.ocn
        echo "Simulating for input connected to vdd"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_2.ocn
        echo "Simulating for input connected to Z"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_3.ocn
        # Run next simulation if previous one didnt oscillate
        if  grep "Frequency" ./results/${TECH_CHAN}_${temp}C_*_top_results_3.csv | grep -q "eval err" ;
        then
            echo "Simulating for input connected to !Z"
            virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_4.ocn
        fi

        temp=m40
        echo "Simulating for -40C"
        echo "Simulating for input connected to gnd"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_1.ocn
        echo "Simulating for input connected to vdd"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_2.ocn
        echo "Simulating for input connected to Z"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_3.ocn
        # Run next simulation if previous one didnt oscillate
        if  grep "Frequency" ./results/${TECH_CHAN}_${temp}C_*_top_results_3.csv | grep -q "eval err" ;
        then
            echo "Simulating for input connected to !Z"
            virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_4.ocn
        fi

        temp=125
        echo "Simulating for 125C"
        echo "Simulating for input connected to gnd"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_1.ocn
        echo "Simulating for input connected to vdd"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_2.ocn
        echo "Simulating for input connected to Z"
        virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_3.ocn
        # Run next simulation if previous one didnt oscillate
        if  grep "Frequency" ./results/${TECH_CHAN}_${temp}C_*_top_results_3.csv | grep -q "eval err" ;
        then
            echo "Simulating for input connected to !Z"
            virtuoso -nograph -restore $OUT_DIR/top_sim_${temp}C_Excluding_${exclude_cells}%_4.ocn
        fi
    fi

done


echo "All done"
echo

exit
```

.6 Synthesis script

Script used for synthesizing the RTL design using DesingCompiler

```
# Author: Daniel Guarecuco Aguiar <deaguiar@stud.ntnu.no>
#   <daniel.aguiar@nordicsemi.no>
# This script synthesizes RTL design, exports timing report
# Usage: Intented to be called from main.sh
#   dc_shell -f syn.tcl
# # ------------------------------------------------------------------------------

# Get library name from env varible defined in bash script
set TECH_LIB $env(TECH_LIB)
```

```
# Get relevant name to be used for saving all files
set OUT_DIR $env(OUT_DIR)
exec mkdir -p $OUT_DIR
# Get period
set PERIOD $env(PERIOD)
# Get exlude_cells flag
set exclude_cells $env(exclude_cells)
set TECH_CHAN $env(TECH_CHAN)

suppress_message LINT-52
suppress_message LINT-31
suppress_message LINT-29

# ############### Specify Libraries # ##########################
set LIB_PATH "/library_path"

set ADDITIONAL_SEARCH_PATH    [list {*}[subst {
  [glob $LIB_PATH/path_to_library]
  [glob ./]
  [glob $OUT_DIR]
}]]  ;

set search_path ""
append search_path " $ADDITIONAL_SEARCH_PATH"

set link_library "*"
lappend link_library $TECH_LIB

set target_library  $TECH_LIB

# ############### Read Design # ##############################
# Define workpath
define_design_lib -path ./work WORK

# Read verilog source files
source -echo -verbose ./design.tcl

set DESIGN_NAME "ibex_top"

# Initialize name mapping database
set hdlin_enable_upf_compatible_naming true
saif_map -start

analyze -library WORK -format sverilog -vcs ${RTL_SOURCE_LIST}
set top_entity ${DESIGN_NAME}

# Elaborate design
elaborate ${DESIGN_NAME}

# Convert FSDB to SAIF
sh fsdb2saif $OUT_DIR/ibex_simple_system.fsdb -o $OUT_DIR/ibex_top.saif  -flatten_genhier
# Map RTL simulation names to RTL design
saif_map -create_map -source_instance ibex_simple_system/u_top/u_ibex_top -input $OUT_DIR/ibex_top.saif

# ############### Set Design Constraints # ######################
create_clock clk_i -name "CLK" -period $PERIOD

if {$exclude_cells != 0} {
  source -echo -verbose $OUT_DIR/../${TECH_CHAN}_exclude_cells_${exclude_cells}.tcl
}

# ############### Optimize Design # ############################
# Compile design
compile -exact_map

# ############### Save Design # ################################
write_file -hierarchy -format verilog -output $OUT_DIR/ibex_top_map.v ibex_top
write_file -format svsim -output $OUT_DIR/ibex_top_wrapper.sv
write_file -hierarchy -format ddc -output $OUT_DIR/ibex_top_map.ddc
write_sdc $OUT_DIR/ibex_top_sdc.sdc
saif_map -write_map $OUT_DIR/ibex_top_name_mapping.tcl -type primepower

# Full. Include name of input pin in the cell
report_timing -input_pins > $OUT_DIR/report_timing_full.txt
# Input-to-Register
report_timing -from [all_inputs] -to [all_registers -data_pins] > $OUT_DIR/report_timing_in_reg.txt
# Register-to-Register
report_timing -from [all_registers -clock_pins] -to [all_registers -data_pins] > $OUT_DIR/report_timing_reg_reg.txt
# Register-to-Output
report_timing -from [all_registers -clock_pins] -to [all_outputs] > $OUT_DIR/report_timing_reg_out.txt
# Input-to-Output
report_timing -from [all_inputs] -to [all_outputs] > $OUT_DIR/report_timing_in_out.txt

quit
```

.7 Power estimation script

Script used for obtaining power estimation using PrimePower

```
# Author: Daniel Guarecuco Aguiar <deaguiar@stud.ntnu.no>
#   <daniel.aguiar@nordicsemi.no>
# This script reports power and timing from synthesized design
# Generates spice netlist of critical path
# Usage: Intented to be called from main.sh
#   pt_shell -f power.tcl
# # ----------------------------------------------------------------------------

# Get library name from env varible defined in bash script
set TECH_LIB $env(TECH_LIB);
# Get relevant name to be used for saving all files
set OUT_DIR $env(OUT_DIR)
set TECH_CHAN $env(TECH_CHAN)

suppress_message CMD-041

# ############### Specify Libraries # ########################
set LIB_PATH "/library_path"
set ADDITIONAL_SEARCH_PATH    [list {*}[subst {
  [glob $LIB_PATH/path_to_library]
  [glob ./]
}]]  ;
set search_path ""
append search_path " $ADDITIONAL_SEARCH_PATH"

set link_library "* "
lappend link_library  $TECH_LIB

set power_enable_analysis true
set case_analysis_propagate_through_icg true
set power_analysis_mode averaged

read_verilog "$OUT_DIR/ibex_top_map.v"
current_design ibex_top
link_design
read_sdc "$OUT_DIR/ibex_top_sdc.sdc"

set_power_analysis_options -through_mode

update_timing -full

check_power

source $OUT_DIR/ibex_top_name_mapping.tcl
read_saif -strip_path  "/ibex_simple_system/u_top/u_ibex_top" $OUT_DIR/ibex_top.saif

report_switching_activity -list_not_annotated -include_only {rtl}  > $OUT_DIR/not_annotated.txt

update_power
report_switching_activity -list_not_annotated -include_only {rtl}  > $OUT_DIR/not_annotated_post.txt

report_power > $OUT_DIR/report_power.txt
report_timing > $OUT_DIR/report_timing_pt.txt

#Finding CDL file path
set curr_lib_path [which $TECH_LIB]
regsub {liberty\/nldm} $curr_lib_path "cdl"  cdl_path
regsub {_TT\w+.db|_abblvt\w+.nldm.db} $cdl_path ".cdl"  cdl_path

# Writing SPICE netlist
write_spice_deck \
-output $OUT_DIR/netlist_critical_path.sp [get_timing_paths] \
-sub_circuit_file "$cdl_path"

quit
```

.8 Build Ocean script

This script is used to build Ocean scripts by means of print statements. Ocean scripts are used for analog simulations.

```
# Author: Daniel Guarecuco Aguiar <deaguiar@stud.ntnu.no>
#   <daniel.aguiar@nordicsemi.no>
# This script generates four oceanScript files to run analog simulations on virtuoso
# These four simulations are: Input to gnd, to vdd, to Z and to !Z
# All four top schematics must be first imported by SpiceIn
#
# Usage: Intented to be called from main.sh
#   python3 generate_ocn.py syn_out/LVT24_25C LVT24
# # ----------------------------------------------------------------------------

import sys
import time
import pandas as pd
import os

```

```
16  def write_ocn(sw_pos):
17      ocean_file = dir_path + '/' + "top_sim_"+ str(temp_str) + "C_Excluding_" + str(exclude_cells) + "%_"+ sw_pos+".ocn"
18      if (exclude_cells == "0"):
19          result_file = "./results/" + tech +'_'+ str(temp_str) + "C_" +vnw_name + vpw_name +"_"+ current_time +"_top_results_"+
            ↪  sw_pos+".csv"
20      else:
21          result_file = "./results/" + tech +'_'+ str(temp_str) + "C_Excluding_"+ str(exclude_cells) + "%_" +vnw_name + vpw_name
            ↪  +"_"+ current_time +"_top_results_"+ sw_pos+".csv"
22      maestro = "maestro_top_" + current_time
23
24      original_stdout = sys.stdout # Save a reference to the original standard output
25      with open(ocean_file, 'w') as f:
26          sys.stdout = f # Change the standard output to the file we created.
27          print(';Script generated by generate_ocn.py')
28          print(';====================Set to Maestro mode explorer ============================')
29          print('ocnSetXLMode("explorer")')
30          print('ocnxlProjectDir( "/my_path/work/daag/virtuoso" )')
31          print('ocnxlTargetCellView( "'+testbench+'" "top" "'+maestro+'" )')
32          print('ocnxlResultsLocation( "" )')
33          print('ocnxlHistoryPrefix("sw_pos_'+sw_pos+'")')
34          print('ocnxlSimResultsLocation( "" )')
35          print('ocnxlMaxJobFail( 20 )')
36          print('')
37          print(';====================== Tests setup ==========================================')
38          print('')
39          print(';---------- Test "'+testbench+'_top_1" ------------- ')
40          print('ocnxlBeginTest("'+testbench+'_top_1")')
41          print('simulator( \'spectre )')
42          print('design( "'+testbench+'" "top" "schematic")')
43          print('modelFile( ')
44          print(')')
45          print('analysis(\'tran ?stop "'+sim_stop+'\"  ?errpreset "conservative"  )')
46          print('desVar(    "vdd" 0.4  )')
47          print('desVar(    "vnw_n" 0.4  )')
48          print('desVar(    "vpw_p" 0  )')
49          print('desVar(    "sw_pos" '+sw_pos+')')
50          print('envOption(')
51          print('  \'analysisOrder  list("pz" "dcmatch" "stb" "tran" "envlp" "ac" "dc" "lf" "noise" "xf" "sp" "pss" "pac" "pstb"
            ↪  "pnoise" "pxf" "psp" "qpss" "qpac" "qpnoise" "qpxf" "qpsp" "hb" "hbac" "hbstb" "hbnoise" "hbxf" "sens" "acmatch") ')
52          print(')')
53          print('saveOption( ?infoOptions list(list("modelParameter" "models" "rawfile" "" "" "" t) list("element" "inst" "rawfile"
            ↪  "" "" "" t) list("outputParameter" "output" "rawfile" "" "" "" t) list("designParamVals" "parameters" "rawfile" "" ""
            ↪  "" t) list("primitives" "primitives" "rawfile" "" "" "" t) list("subckts" "subckts" "rawfile" "" "" "" t)
            ↪  list("asserts" "assert" "rawfile" "" "" "" nil) list("extremeinfo" "all" "logfile" "" "yes" "" nil) list("allcap"
            ↪  "allcap" "file" "" "" "" nil) list("<Click_To_Add>" "none" "rawfile" "" "" "" nil) ) )')
54          print('save( \'v "/A" "/Z" )')
55          print('save( \'i "/xtop_01/VDD" )')
56          print('temp( '+str(temp)+' ) ')
57          print('ocnxlOutputSignal( "/A" ?plot t ?save t)')
58          print('ocnxlOutputSignal( "/Z" ?plot t ?save t)')
59          print('ocnxlOutputExpr( "frequency(clip(VT(\\"/Z\\") (lastVal(VT(\\"/Z\\")) / 2) lastVal(VT(\\"/Z\\"))))" ?name
            ↪  "Frequency" ?plot t ?evalType \'point)')
60          print('ocnxlOutputExpr( "dutyCycle(VT(\\"/Z\\") ?mode \\"user\\" ?threshold (VAR(\\"vdd\\") / 2) ?xName \\"time\\"
            ↪  ?outputType \\"average\\")" ?name "DutyCycle" ?plot t ?evalType \'point)')
61          print('ocnxlOutputExpr( "average(clip((VAR(\\"vdd\\") * IT(\\"/xtop_01/VDD\\")) (lastVal(VT(\\"/Z\\")) / 2)
            ↪  lastVal(VT(\\"/Z\\")))))" ?name "Power" ?plot t ?evalType \'point)')
62          print('ocnxlOutputTerminal( "/xtop_01/VDD" ?save t)')
63          print('ocnxlOutputExpr( "delay(?wf1 VT(\\"/A\\") ?value1 (VAR(\\"vdd\\") / 2) ?edge1 \\"either\\" ?nth1 1 ?td1 0.0 ?tol1
            ↪  nil ?wf2 VT(\\"/Z\\") ?value2 (VAR(\\"vdd\\") / 2) ?edge2 \\"either\\" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple
            ↪  nil)" ?name "Delay" ?plot t ?evalType \'point)')
64          print('ocnxlOutputExpr( "lastVal(VT(\\"/Z\\"))" ?name "SimTime" ?plot t ?evalType \'point)')
65          print('ocnxlEndTest() ; "'+testbench+'_top_1"')
66          print('')
67          print(';====================== Sweeps setup ==========================================')
68          print('')
69          print(';====================== Model Group setup =====================================')
70          print('')
71          print(';====================== Corners setup =========================================')
72
73          for vdd in corners:
74              print('ocnxlCorner( "tc_'+vdd+'_'+str(temp_str)+'"')
75              print('   \'(')
76              print('      ("variable" "vdd" "'+vdd+'")')
77              print('      ("variable" "vnw_n" "'+str(corners_df[vdd].loc['vnw_n'])+'")')
78              print('      ("variable" "vpw_p" "'+str(corners_df[vdd].loc['vpw_p'])+'")')
79              print('      ("variable" "temperature" "'+str(temp)+'")')
80              print('      ("model" "HERE GOES TRANSISTOR MODEL" ?section "\\"tt\\" \\"ff\\" \\"ss\\"")')
81              print('      ("modelGroup" "")')
82              print('   )')
83              print(')')
84          print('')
85          print(';====================== Checks and Asserts setup ==============================')
86          print('ocnxlPutChecksAsserts(?netlist nil)')
87          print('')
88          print(';====================== Job setup =============================================')
89          print('ocnxlJobSetup( \'(')
90          print(') )')
91          print('')
92          print(';====================== Disabled items ========================================')
93          print(';ocnxlDisableCorner("tc_0.80_25") **This disables a corner')
94          print('ocnxlSetAllParametersDisabled(t)')
95          print('')
96          print(';====================== Run Mode Options ======================================')
97          print('')
98          print(';====================== Starting Point Info ===================================')
99          print('')
100         print(';====================== Run command ===========================================')
101         print('printf("*******************************************************************************************\\n")')
102         print('printf("Running simulation with switch position = '+sw_pos+'\\n")')
```

```
        print('printf("*****************************************************************************************\\n")')
        print('ocnxlRun( ?mode \'sweepsAndCorners ?nominalCornerEnabled nil ?allCornersEnabled t ?allSweepsEnabled t)')
        print('ocnxlOutputSummary(?exprSummary t ?specSummary t ?detailed t ?wave t)')
        print('ocnxlExportOutputView("'+result_file+'" "Detail")')
        print('ocnxlOpenResults()')
        print('')
        print(';==================== End XL Mode command ================================')
        print('ocnxlEndXLMode("explorer")')
        print('exit()')
        print('')

        sys.stdout = original_stdout # Reset the standard output to its original value

    print('Simulation file saved in '+ ocean_file)


dir_path = sys.argv[1]
#tech = "LVT24"
tech = sys.argv[2]
testbench = tech + "_Testbench"
temp_str = sys.argv[3]
temp = int(temp_str.replace('m', '-'))

exclude_cells = os.getenv('exclude_cells') # Percetage excluded

t = time.localtime()
current_time = time.strftime("%H%M%S", t)

sim_stop = '400m'

#corner_csv = dir_path +'/../'+tech+'_'+str(temp_str)+'C_corners.csv'
corner_csv = dir_path +'/../'+tech+'_'+'25C_corners.csv'
corners_df = pd.read_csv (corner_csv,index_col=0)
print(corners_df)
vnw_name = str(corners_df.iloc[0,0])
vpw_name = str(corners_df.iloc[1,0])
vpw_name = vpw_name.replace('-', 'm')
corners = corners_df.columns.values

print('Generating OCN scripts.')
write_ocn("1")
write_ocn("2")
write_ocn("3")
write_ocn("4")
print('Done...')
```

.9 Build testbench script

This script cleans the spice netlist generated by Synopsys and builds the testbench.

```
# Author: Daniel Guarecuco Aguiar <deaguiar@stud.ntnu.no>
#   <daniel.aguiar@nordicsemi.no>
# Created Date: 14/02/2023
# This script cleans the spice netlist generated by PrimeTime.
# Generates four testbenches, where input is connected to gnd, vdd, Z and !Z.
# Generates four spiceIn.params
#
# Usage: Intented to be called from main.sh
#   python3 generate_testbenches.py /my_path/master_thesis/analysis/syn_out/LVT24_25C LVT24_Testbench
# # ------------------------------------------------------------------------

import fileinput
import re
import shutil
import os
import sys

def copy_file(dest_path,spice_file_source,spice_file):
    if not os.path.exists(dest_path):
        os.mkdir(dest_path)
    shutil.copy(spice_file_source, spice_file)

# Append current line to previous one if it starts with +
def remove_break(file_path):
    previousLine = ""
    # Read spice file line by line
    for line in fileinput.input(file_path, inplace=True):
        if line.startswith("+"):
            previousLine = previousLine.replace("\n","")  + line.replace("+","")
        else:
            print('{}'.format(previousLine), end='')
            previousLine = line

# Return a list of cell names in the critical path, obtained from the timing report
def get_all_cell_names(file_path):
```

```
36      with open(file_path, 'r') as file:
37          # read all content of the file
38          content = file.read()
39          names = re.findall('\s+(.+)\/\w+ \(.+\)',content)
40          # remove duplicates (input + output)
41          res = []
42          res_prev = ""
43          for i in names:
44              if (i != res_prev):
45                  res_prev = i
46                  res.append(i)
47          return res
48
49  def remove_cell(file_path, cell_name):
50      # Read spice file line by line
51      for line in fileinput.input(file_path, inplace=True):
52          # Check if cell_name is in the line and remove it
53          if (line.find(cell_name) != -1):
54              newline = "* LINE REMOVED: " + line
55              print('{}'.format(newline), end='')
56          else:
57              # Do nothing, keep the same line
58              print('{}'.format(line), end='')
59
60  def get_cell_pins(file_path, cell_name):
61      with open(file_path, 'r') as file:
62          # read all content of the file
63          pins = []
64          content = file.read()
65          pins = re.findall('\s+(' + re.escape(cell_name) + '\/\w+) \(.+\)',content)
66      return pins
67
68  def append_to_file(file_path, string):
69      # Open a file with access mode 'a'
70      with open(file_path, "a") as file_object:
71          # Append string at the end of file
72          file_object.write(string)
73
74  def replace_slash(file_path):
75      # Read spice file line by line
76      for line in fileinput.input(file_path, inplace=True):
77          # Do nothing if it stats with a .
78          if line.startswith("."):
79               print('{}'.format(line), end='')
80          else:
81              line = line.replace('/', '_')
82              print('{}'.format(line), end='')
83
84  def replace_brackets(file_path):
85      # Read spice file line by line
86      for line in fileinput.input(file_path, inplace=True):
87          # Do nothing if it stats with a .
88          if line.startswith("."):
89               print('{}'.format(line), end='')
90          else:
91              line = line.replace('[', '_')
92              line = line.replace(']', '_')
93              print('{}'.format(line), end='')
94
95  def replace_parenthesis(file_path):
96      # Read spice file line by line
97      for line in fileinput.input(file_path, inplace=True):
98          # Do nothing if it stats with a .
99          if line.startswith("."):
100              print('{}'.format(line), end='')
101         else:
102             line = line.replace('(', '_')
103             line = line.replace(')', '_')
104             print('{}'.format(line), end='')
105
106 def remove_vsource(file_path, string):
107     # Read spice file line by line
108     for line in fileinput.input(file_path, inplace=True):
109         # Do nothing if it stats with a .
110         if line.startswith("v"):
111             if (line.find(string) != -1):
112                 line = "* LINE REMOVED: " + line
113         print('{}'.format(line), end='')
114
115 def replace_input(file_path, string):
116     # Read spice file line by line
117     for line in fileinput.input(file_path, inplace=True):
118         if line.startswith("x"):
119             # If _VDD
120             if (line.find(string) != -1):
121                 if (string == "_VDD"):
122                     list_line = re.search('(^x.+)(\s\S+_VDD)(.+)',line)
123                     line = list_line.group(1) + " vdd" + list_line.group(3) + "\n"
124                 elif (string == "_VNW_N"):
125                     list_line = re.search('(^x.+)(\s\S+_VNW_N)(.+)',line)
126                     line = list_line.group(1) + " vnw_n" + list_line.group(3) + "\n"
127                 elif (string == "_VPW_P"):
128                     list_line = re.search('(^x.+)(\s\S+_VPW_P)(.+)',line)
129                     line = list_line.group(1) + " vpw_p" + list_line.group(3) + "\n"
130                 elif (string == "_VNW_P"):
131                     list_line = re.search('(^x.+)(\s\S+_VNW_P)(.+)',line)
132                     line = list_line.group(1) + " vnw_n" + list_line.group(3) + "\n"
133                 elif (string == "_VPW_N"):
134                     list_line = re.search('(^x.+)(\s\S+_VPW_N)(.+)',line)
```

```
                    line = list_line.group(1) + " vpw_p" + list_line.group(3) + "\n"
        print('{}'.format(line), end='')

def unify_vdd(file_path):
    # Read spice file line by line
    for line in fileinput.input(file_path, inplace=True):
        if line.startswith("vvdd"):
            # Extract value of vdd
            vdd_value = re.search('^vvdd\s+vdd\s+\d\s+(\d+\.\d+)',line).group(1)
            line = "vvdd vdd 0 vdd\n"
        elif line.startswith("v"):
            if (re.match('^v\S+\s+\S+\s+0\s+' + vdd_value, line)):
                newline = re.search('(^v\S+\s+\S+\s)+0\s+' + vdd_value,line).group(1)
                line = newline + "vdd 0\n"
        print('{}'.format(line), end='')

def lower_case_vss(file_path):
    # Read spice file line by line
    for line in fileinput.input(file_path, inplace=True):
        line = line.replace('VSS', 'vss')
        print('{}'.format(line), end='')

# Remove stimulus, vvdd, vvss, .temp, .global
def remove_stim(file_path):
    # Read spice file line by line
    for line in fileinput.input(file_path, inplace=True):
        # Remove the following line
        if line.startswith(".include"):
            if (re.match('^~\.include\s+\S+stim\.sp\S', line)):
                line = "* LINE REMOVED: " + line
        print('{}'.format(line), end='')

def remove_corners(file_path):
    # Read spice file line by line
    ret_text = ""
    for line in fileinput.input(file_path, inplace=True):
        # Remove the following lines
        if line.startswith("vvdd vdd"):
            ret_text += line
            line = "* LINE REMOVED: " + line
        elif line.startswith(".global vss"):
            ret_text += line
            line = "* LINE REMOVED: " + line
        elif line.startswith("vvss vss"):
            ret_text += line
            line = "* LINE REMOVED: " + line
        elif line.startswith(".temp"):
            ret_text += line
            line = "* LINE REMOVED: " + line
        print('{}'.format(line), end='')
    return ret_text

def creat_subckt(file_path):
    # Read spice file line by line
    for line in fileinput.input(file_path, inplace=True):
        # Append subcircuit definition after .temp
        if line.startswith(".temp"):
            subckt="\n.SUBCKT CRITICAL_PATH VDD VSS VNW_N VPW_P A Z\n"
            line = line + subckt
        print('{}'.format(line), end='')

def connect_ext_pins(spice_file, outer_in, outer_out):
    conn_str = "\n* CONNECTION TO EXTERNAL PINS\n\
vext_vdd VDD vdd 0\n\
vext_vss VSS vss 0\n\
vext_vnw_n VNW_N vnw_n 0\n\
vext_vpw_p VPW_P vpw_p 0\n\
vext_in A "+ outer_in + " 0\n\
vext_out Z "+ outer_out + " 0\n\
.ENDS"
    # Append line to spice file
    append_to_file(spice_file, conn_str)


def write_to_file(file_path, content):
    f = open(file_path, "w")
    f.write(content)
    f.close()

def write_testbench(file_path, content):
    tb_path = file_path + '/netlist_testbench.spi'
    feedback_str = content + "\n* Input A is connected to a sp4tswitch. Inputs are VSS, VDD, Z or !Z\n\
* Switch is controlled via sw_pos variable\n\n\
einverter nZ vss vdd Z 1\n\
simulator lang=spectre\n\
S0 (A vss vdd Z nZ) sp4tswitch paramTyp=string tranPosition_str=sw_pos model=switch\n\
simulator lang=spice\n\n\
.end"
    write_to_file(tb_path, feedback_str)
    print('File ' +tb_path+ ' created.')


################################################################################
#dir_path = "/my_path/mep_thesis/analysis/syn_out/LVT24_25C"
# Path taken from argument, set at main.sh
dir_path = sys.argv[1]
spice_file_source =  dir_path + "/netlist_critical_path.sp"
report_file = dir_path + "/report_timing_full.txt"

```

```
dest_path= dir_path
spice_file_name = "netlist_critical_path.spi"
spice_file = dest_path + "/" + spice_file_name

################################################################################

# Copy netlist to spice directory
copy_file(dest_path,spice_file_source,spice_file)

# Gather data from timing report
# Get all cell names in critical path
list_cells = get_all_cell_names(report_file)
# Get name of input pin. Used for feedback
outer_in = get_cell_pins(report_file, list_cells[1])[0]
# Get name of output pin. Used for feedback
outer_out = get_cell_pins(report_file, list_cells[-2])[1]


# Processing netlist to be imported by Virtuoso
# Remove line breaks when line is too long. Easier to process
remove_break(spice_file)
# Remove first FF. Interested only in combinational circuits between two FF
remove_cell(spice_file, list_cells[0])
# Remove last FF
remove_cell(spice_file, list_cells[-1])
# Remove all individual vsource used for VDD, VNW_N, VPW_P
remove_vsource(spice_file, "_VDD")
remove_vsource(spice_file, "_VNW_N")
remove_vsource(spice_file, "_VPW_P")
remove_vsource(spice_file, "_VNW_P")
remove_vsource(spice_file, "_VPW_N")

replace_input(spice_file, "_VDD")
replace_input(spice_file, "_VNW_N")
replace_input(spice_file, "_VPW_P")
replace_input(spice_file, "_VNW_P")
replace_input(spice_file, "_VPW_N")
# Instead of having multiple vsource with same value as vdd, short-circuit them to vdd
unify_vdd(spice_file)
# Replace all VSS with vss
lower_case_vss(spice_file)

# Create subckt definition after .temp
creat_subckt(spice_file)

# Remove stimulus. Not needed
remove_stim(spice_file)
# Remove vvdd, vvss, .temp, .global. To be declared outside in testbench
corners=remove_corners(spice_file)

# Remove final .end. So we can append connecting signals
remove_cell(spice_file, ".end")
# Connect to external pins and add .ENDS
connect_ext_pins(spice_file, outer_in, outer_out)
# Replace slash / by _. To avoid issues with Virtuoso
replace_slash(spice_file)
# Replace [] by _. To avoid issues with Virtuoso
replace_brackets(spice_file)
# Replace () by _. To avoid issues with Virtuoso
replace_parenthesis(spice_file)


# Create testbench
content = ".include " + spice_file_name + "\n\n" + corners + "vvnw_n vnw_n 0 vnw_n\n"\
+ "vvpw_p vpw_p 0 vpw_p\n"\
+ "\nxtop_01 vdd vss vnw_n vpw_p A Z CRITICAL_PATH\n"
write_testbench(dest_path, content)

print('Done')
```

.10 Extract timing from liberty

This script is used to extract timing information from liberty files and build a ranking list of worst performing cells.

```
import re
import os
import csv
import gzip
import sys

#lib_file_tc = "/library_path/HVT36_tc.lib"
#lib_file_wc = "/library_path/HVT36_wc.lib"

'''
In a text such like this, return value in the middle
```

```
12
13    "0.0137688, 0.0150271, 0.0173965, 0.0219685, 0.0319864, 0.0548207, 0.106199, 0.197879",\
14    "0.0152693, 0.0165209, 0.0188896, 0.023457, 0.033496, 0.0563155, 0.107772, 0.199596",\
15    "0.0182839, 0.0195251, 0.0218886, 0.02647, 0.036507, 0.0593427, 0.11078, 0.202579",\
16    "0.0237705, 0.0250312, 0.0274031, 0.0319726, 0.042035, 0.0649499, 0.11637, 0.208143",\
17    "0.0311124, 0.032437, 0.0348371, 0.0394019, 0.049505, 0.0724021, 0.123879, 0.215561",\
18    "0.0412562, 0.042758, 0.0453693, 0.0499858, 0.0600137, 0.0829246, 0.134398, 0.226131",\
19    "0.0544402, 0.0562811, 0.0594323, 0.0644088, 0.0744447, 0.0972843, 0.148712, 0.240422",\
20    "0.0711327, 0.0734402, 0.0771574, 0.0832997, 0.0937454, 0.116406, 0.167798, 0.2595"
21
22    returns 0.049505
23    '''
24    def get_timing_value(string):
25        #split into a list when \ is encounter, remove any \n, space and "
26        value_list = string.replace('\n', '').replace(' ', '').replace('"', '').split(",\\")
27        # Build list of list
28        value_list_list = []
29        for row in value_list :
30            row_split = row.split(",")
31            value_list_list.append(row_split)
32
33        value = value_list_list[int(len(value_list_list)/2)][int(len(value_list_list[0])/2)]
34        return value
35
36    def get_all_timing_values(library):
37        regex_cell = r'\s*cell\s*\((\w+)\)'
38        #regex_timing = r'\s*timing\s*\(\)\s*\{[^{]*\s*(?:cell_fall|cell_rise)\s*\(\w+\)\s*\{(?:[^v]*)values\s*\(([^)]*)[^{]*\s*(?:ce
    ↪  ll_rise|cell_fall)\s*\(\w+\)\s*\{(?:[^v]*)values\s*\(([^)]*)'
39        regex_fall = r'\s*related_pin\s*:\s*\"(\w+)\"(?:.|\n)*?\s*cell_fall\s*\(\w+\)\s*\{(?:[^v]*)values\s*\(\s*\\*\s*([^)]*)'
40        regex_rise = r'\s*related_pin\s*:\s*\"(\w+)\"(?:.|\n)*?\s*cell_rise\s*\(\w+\)\s*\{(?:[^v]*)values\s*\(\s*\\*\s*([^)]*)'
41
42        # Extract all cells and their attributes from the library by trasversing the curly braces. Saved in data_from_lib
43        countBraces = 0
44        data = ""
45        data_from_lib = []
46        if library.endswith('.gz'):
47            with gzip.open(library,'rt') as f:
48                for line in f:
49                    if (countBraces == 0) :
50                        if (re.search(regex_cell, line) ):
51                            cell_name = re.search(regex_cell, line).group(1)
52                            countBraces = 1
53                    else:
54                        data += line
55                        if ('{' in line) :
56                            countBraces +=1
57                        elif ('}' in line) :
58                            countBraces -=1
59                            if (countBraces == 0) :
60                                data_from_lib.append([cell_name, data])
61                                data = ""
62        else:
63            with open(library, 'r') as f:
64                for line in f:
65                    if (countBraces == 0) :
66                        if (re.search(regex_cell, line) ):
67                            cell_name = re.search(regex_cell, line).group(1)
68                            countBraces = 1
69                    else:
70                        data += line
71                        if ('{' in line) :
72                            countBraces +=1
73                        elif ('}' in line) :
74                            countBraces -=1
75                            if (countBraces == 0) :
76                                data_from_lib.append([cell_name, data])
77                                data = ""
78
79
80        # Extract timing value for each cell
81        list_cell_timing = []
82        for cell in data_from_lib:
83            cell_name = cell[0]
84            cell_attributes = cell[1]
85            countBraces = 0
86            data = ""
87            timing_list = []
88            #Extract all timing attributers by trasversing the curly braces
89            for line in cell_attributes.splitlines():
90                    if (countBraces == 0) :
91                        if (re.search(r'\s*timing\s*\(\)', line) ):
92                            countBraces = 1
93                    else:
94                        data += line
95                        if ('{' in line) :
96                            countBraces +=1
97                        elif ('}' in line) :
98                            countBraces -=1
99                            if (countBraces == 0) :
100                               timing_list.append(data)
101                               data = ""
102
103           pin_results = []
104           for timing in timing_list:
105               result_fall = re.findall(regex_fall, timing)
106               result_rise = re.findall(regex_rise, timing)
107
108               #Go to next if no timing attributes
109               if(not result_fall and not result_rise):
```

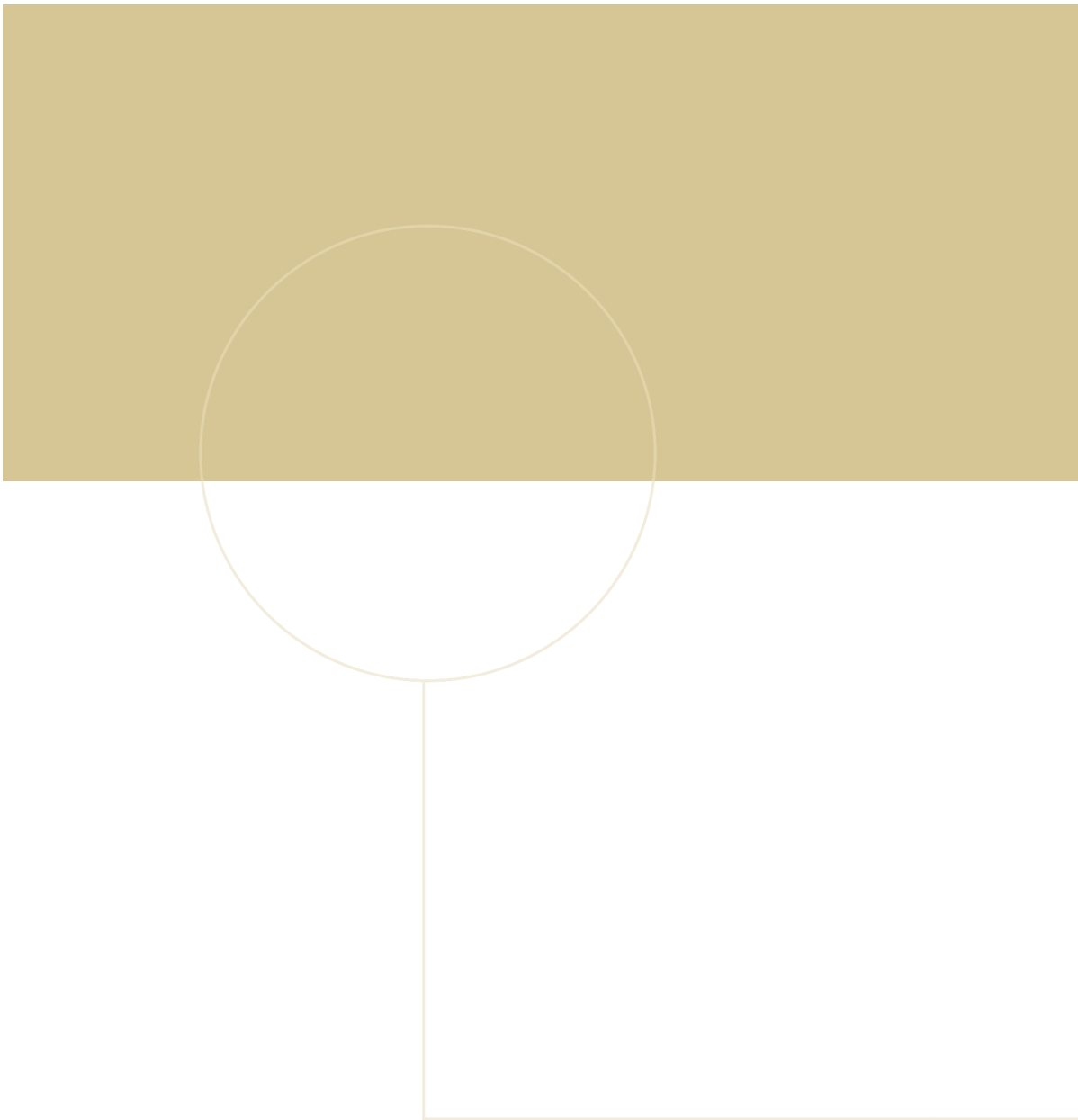
```
                    continue

                #If both exist, take the sum
                elif (result_fall and result_rise):
                    related_pin = result_fall[0][0]
                    related_pin_r = result_rise[0][0]
                    #Making sure they are the same pin. They should be
                    if (related_pin == related_pin_r):
                        timing_matrix_f = result_fall[0][1]
                        timing_matrix_r = result_rise[0][1]
                        cell_value_f = get_timing_value(timing_matrix_f)
                        cell_value_r = get_timing_value(timing_matrix_r)
                        cell_value = float(cell_value_f) + float(cell_value_r)
                        # rise-fall / (rise+fall)/2
                        timing_diff = 100*(float(cell_value_r) - float(cell_value_f))/((float(cell_value_r) + float(cell_value_f))/2)

                elif(not result_fall):
                    related_pin = result_rise[0][0]
                    timing_matrix_r = result_rise[0][1]
                    cell_value = float(get_timing_value(timing_matrix_r))

                elif(not result_rise):
                    related_pin = result_fall[0][0]
                    timing_matrix_f = result_fall[0][1]
                    cell_value = float(get_timing_value(timing_matrix_f))

                pin_results.append([related_pin,cell_value,timing_diff])
            list_cell_timing.append([cell_name,pin_results])

    return list_cell_timing


#############################################################################################
threshold = int(sys.argv[1])
lib_file_wc = sys.argv[3]

# Get timing for typical and worst case
print("Extracting data from typical case lib")
tc_value = get_all_timing_values(lib_file_tc)
print("Extracting data from worst case lib")
wc_value = get_all_timing_values(lib_file_wc)


#Matches LVT
regex1=r'_([a-zA-Z]+VT[0-9]+)_'
#Matches HVT
regex2= r'_([a-zA-Z]+[0-9]+[a-zA-Z]+)_TT_0P'

#Extract technology name from file
path, filename = os.path.split(lib_file_tc)
if (re.search(regex1, filename) ):
    TECH_CHAN = re.search(regex1, filename).group(1)
elif (re.search(regex2, filename) ):
    TECH_CHAN = re.search(regex2, filename).group(1)
else:
    TECH_CHAN = ""

cell_time_scaling = []
#Go through all cells in Typical case
for cell_tc in tc_value:
    cell_name = cell_tc[0]
    pin_list = cell_tc[1]
    worst_ratio = 0
    worst_diff = 0
    #Go through all cell in worst case
    for cell_wc in wc_value:
        #If they are the same cell
        if (cell_name == cell_wc[0]):
            for (pin,pin_wc) in zip(pin_list, cell_wc[1]):
                if (pin[0] == pin_wc[0]):
                    timing_ratio = pin_wc[1]/pin[1]
                    if (worst_ratio < timing_ratio ):
                        worst_ratio = timing_ratio

                    if (abs(worst_diff) < abs(pin_wc[2])):
                        worst_diff = pin_wc[2]

            cell_time_scaling.append([cell_name,worst_ratio,worst_diff])

# Order from worst to better
cell_time_scaling.sort(reverse=True,key=lambda x:x[1])

csv_filename = TECH_CHAN + "_time_scaling.csv"
# Save file
with open(csv_filename, "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerows(cell_time_scaling)


dont_touch_list = ['DF']

cell_time_scaling_new = []
#Build new list NOT containing any cell in the dont_touch_list
for cell in cell_time_scaling:
    found = False
    for i in dont_touch_list:
        if (i in cell[0]):
            found = True
            break
```

```
    if (found == False):
        cell_time_scaling_new.append(cell)

num_cell_to_exclude =  int((threshold/100)*len(cell_time_scaling))
#If the numbers of cells to be excluded is greater that the actual number of available cells:
if(num_cell_to_exclude > len(cell_time_scaling_new) ):
    num_cell_to_exclude = 0

tcl_filename = TECH_CHAN + "_exclude_cells_"+str(threshold)+".tcl"
original_stdout = sys.stdout # Save a reference to the original standard output
with open(tcl_filename, 'w') as f:
    sys.stdout = f # Change the standard output to the file we created.
    print('# ##################################################################')
    print('')
    print('# Script created by extract_timing_lib.py')
    print('# Excluding ' + str(threshold)+ '% of cells')
    print('')
    print('# ##################################################################')
    print('')
    print('set sdc_version 2.1')
    print('')
    for i in range(num_cell_to_exclude):
        cell = cell_time_scaling_new[i][0]
        print('set_dont_use [get_lib_cells */'+cell+']')
    sys.stdout = original_stdout # Reset the standard output to its original value


print("Finished\n")
```

.11 Verilog-A Counter

This is a Verilog-A counter used to stop simulation after 10 cycles.

```
// Verilogâ for counting number of cycles of signal and stop simulation
// Counter counts as signal cross 0
//Code partially taken from
↪  https://www.edaboard.com/threads/transient-simulation-termination-by-veriloga-model-in-cadence-virtuoso.399898/
`include "constants.vams"
`include "disciplines.vams"

module counter (clk,vdd);
    inout clk;
    electrical clk;
    inout vdd;
    electrical vdd;
    parameter real vtol = 0; // signal tolerance on the clk
    parameter real ttol = 0; // time tolerance on the clk

    integer cnt=0;

    analog begin
    @(cross(V(clk)-(V(vdd)/2),1,vtol,ttol))

    begin

    cnt=cnt+1;
    if( cnt == 10) begin
        $finish_current_analysis(0);    //command to stop the current simulation
    end
    end
    end
endmodule
```



NTNU

Norwegian University of
Science and Technology