INCORPORATING LEAKAGE CURRENT CONSIDERATIONS IN LOGIC SYNTHESIS

Michael Kristensen

LYNGBY 2004
EKSAMENSPROJEKT
NR. 63

IMM

Trykt af IMM, DTU

Preface

This master's thesis was conducted at the *Computer Science and Engineering* division at *Informatics and Mathematical Modelling* at the *Technical University of Denmark*, during the period February to August 2004.

Peter Østergaard Nielsen from Vitesse Semiconductor Corporation had the original idea for the thesis and Flemming Stassen was advisor. The official project description is attached in appendix A.

I would like to thank Jacob Gregers Hansen and Martin Hans for inspiring discussions during the work on our theses.

Michael Kristensen, Copenhagen

Abstract

Dual–threshold methods for reducing subthreshold leakage power while maintaining performance have matured and entered several commercial power reduction tools. While these methods traditionally are applied incrementally, without changing the overall structure of the circuit, this work has looked at the possibilities of performing leakage power reduction during technology mapping, with the use of a dual–threshold cell library.

KEYWORDS: Leakage power, threshold voltage, dual-Vt, logic synthesis

Resumé

Dual–threshold–metoder til reducering af effektbidraget fra *subthreshold leakage power*, uden at påvirke hastigheden af designet, har vundet indpas i flere kommercielle effektreduceringsværktøjer. Mens sådanne metoder ofte bliver anvendt inkrementelt, uden at ændre den overordnede struktur af kredsløbet, undersøges i dette arbejde mulighederne for at reducere *subthreshold leakage power* ved *technology mapping*, med brug af et *dual–threshold* celle–bibliotek.

STIKORD: *Leakage power*, *threshold voltage*, *dual-Vt*, logisk syntese

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Digital design with the use of static CMOS gates, was in the early days constrained by an upper limit on the area usage. Today, with several hundred millions transistors on a single chip, area is no longer an issue. The speed of transistors have been improved with each technology generation, and today clock frequencies in the GHz domain are not uncommon. The high clock frequencies combined with the huge amounts of transistors, have introduced a limiting factor in design of digital circuits: *Power dissipation*. While power dissipation can be thought of, as being a problem only related to battery powered devices, it is today raising a barrier that hinders increasing the functionality of large high performance designs.

Dynamic power has traditionally been the dominating source of the total power dissipation, but as the shrinking of transistors passed the 180 nm technology, the leakage power contribution to the total power dissipation, became significant. From figure 1.1, it is seen that in near future, the leakage power will constitute a significant part of the total power dissipation.

For small CMOS technologies, the assumption that no current flows trough the transistor when it is off, is invalidated by the leakage currents. It has also been assumed, that no current flows through the *gate* of a transistor, but as the transistors are shrunken, a significant current begins to flow through the gate. Thus the digital design methodologies needs to be revised to cope for the increasing leakage power.

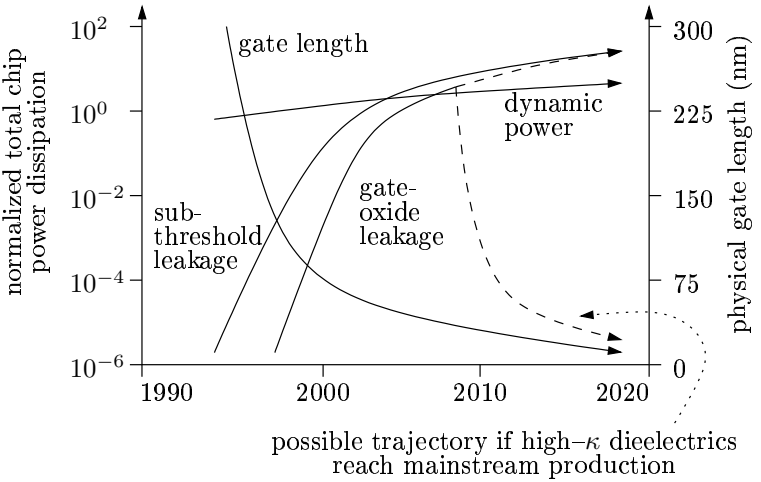

Figure 1.1: Total chip dynamic and leakage power dissipation [17]

1.1 Levels of solutions

The design of digital circuits are done at different levels of abstraction, see figure 1.2. In the hierarchy, functionality are propagated upwards, thus the technology level defines the transistors, the circuit level creates useful structures of the transistors and the synthesis level bridges the requirements from the architectural level with the capabilities of the circuit level.

While the source of the increasing leakage currents is arising from the technology level, this doesn't imply that leakage currents *only* can be reduced at the technology level.

This thesis was performed independently, but in a collaborate work with two other projects. Jacob Gregers Hansen explores logic families and evaluates how they perform in comparison to static CMOS, when leakage currents no longer are negligible [11]. Martin Hans is re-evaluating how architecture driven voltage scaling affects the leakage power and develops an architectural model for estimating power [10].

This thesis considers how logic synthesis can be used for reducing leakage power and the capabilities of state of the art tools for reducing leakage power are studied.

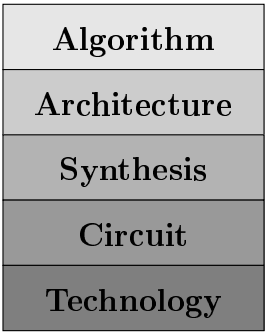

Figure 1.2: Levels of abstraction in digital design [2]

1.2 Scope of the thesis

Several leakage reduction techniques either assumes that the circuit is idle a significant amount of time, and thus shuts down part of the circuit, or reduces leakage with an increase in delay a consequence. Such techniques are not applicable to high performance circuits with high activity, thus a technique that reduces leakage while preserving performance is required.

The reduction of leakage power during synthesis, will be focused on the *subthreshold leakage*. As the threshold voltage has great impact on both the speed and leakage power of a transistor, see table 1.1, the overall aim is to balance the use of high V_t and low V_t transistors, such that leakage is minimized while delay requirements are satisfied. Such a scheme is depicted in figure 1.3.

threshold voltage	speed	subthreshold leakage
low V_t	high speed	high leakage
high V_t	low speed	low leakage

Table 1.1: Speed and leakage tradeoff

When several threshold voltages are used to reduce the leakage power of a design, there is a problem of deciding which transistors are to be assigned low V_t and high V_t. Most methods developed from the middle of the nineties and up to now, are intended to be applied post–layout, thus they consider the netlist of the circuit as being structurally fixed.

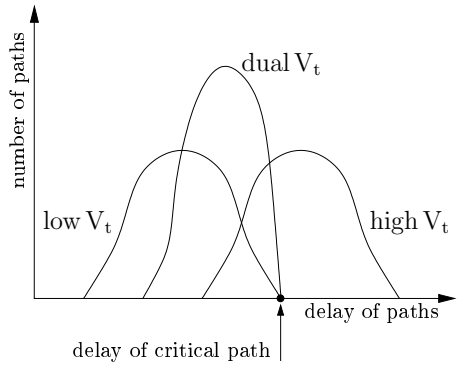

Figure 1.3: Distribution of slack on paths before and after leakage power reduction

In this work, the problem of assigning the threshold voltages is moved to the logic synthesis level, thus giving more freedom in the sense that the structure of the circuit can be adapted to maximize the use of high V_t transistors. The effort will be on the technology mapping process, thus the technology independent logic optimizations, will not be considered. The method will be compared and evaluated against the post–layout approaches and a commercial synthesis tool, Synopsys Design Compiler, will be used for experimentally testing the ideas.

1.3 Overview of the thesis

The structure of the thesis is organized in three parts: The theory of modeling power consumption, the application of multiple threshold voltages in logic synthesis and finally leakage power reduction methods are tested with the use of a commercial synthesis tool.

Chapter 2 introduces the theory of modeling power consumption.
Chapter 3 elaborates on the problem of non-negligible leakage power.
Chapter 4 analyzes the effects of using multiple threshold voltages and large logic gates for reducing leakage power.
Chapter 5 considers the process of synthesizing a design with focus on technology mapping. Furthermore the capabilities of commercial

tools and existing methods are reviewed.

Chapter 6 uses Synopsys Design Compiler for reducing leakage power of two example designs and the results are evaluated against the expectations from chapter 5.

Chapter 7 discusses the achieved results and concludes the thesis.

Appendix A contains the official project description.

Appendix B has details about the CMOS process used for Spice simulations.

Appendix C elaborates on how power consumption is computed with HSpice.

Appendix D lists the HSpice netlists used.

Appendix E is related to how Synopsys is used and behaves.

Appendix F contains the example designs and synthesis scripts.

Appendix G list the directory structure of the digital appendices.

Chapter 2

Theory of modeling power consumption

The total power consumption of a circuit consisting of static CMOS gates, is the sum of the dynamic power and the static power as shown in equation 2.1.

$$P_{total} = P_{dynamic} + P_{short\ circuit} + P_{leakage} \tag{2.1}$$

The *dynamic* component is due to charging and discharging of capacitances, the *short circuit* component is caused by a direct path from supply to ground during a transition and the *leakage* component is due to transistors not being able to fully "turn off".

2.1 Dynamic power

For a switching frequency f and a probability α of a $0 \rightarrow 1$ transition, the average dynamic power can be expressed as in [26]:

$$P_{dynamic} = \alpha\ f\ V_{dd}^2\ C_L$$

C_L is the load capacitance and consists of

- parasitic capacitances of MOS transistors
- capacitance C_i of the interconnect. The interconnect capacitance depends on fanout, length of wires and the size of the driven transistors. These can be joined to a capacitance C_L as in figure 2.1

From the expression it is clear that the dynamic power is proportional with f and C_L, and depends quadratically of the supply voltage V_{dd}.

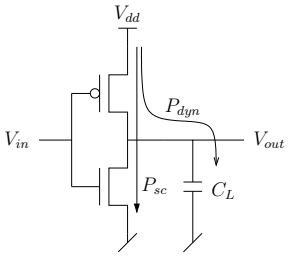

Figure 2.1: Dynamic and short circuit power

2.2 Short circuit power

The *short circuit* power is a dynamic component but instead of charging capacitances, it models a direct path from supply to ground. Because the slope of the input signal of a logic gate is finite, both the pull up and pull down tree of a CMOS gate will conduct for a short period of time.

The short circuit power is in [26] defined as

$$P_{sc} = \frac{\beta}{12}(V_{dd} - 2V_t)^3 \frac{\tau}{T} \tag{2.2}$$

β is the gain factor of the transistor, V_t is the threshold voltage, τ is the transition time of the input and T is the period of the input. The short circuit power is seen to be proportional to the slew of the input signal, and the absolute value of the short circuit power decreases as the threshold voltage is increased. This is caused by the saturation current decreases as the threshold voltage is increased [26]:

$$I_{ds} = \beta \frac{(V_{gs} - V_t)^2}{2}$$

2.3 Leakage power

As the sizes of transistors has been scaled, the supply voltage needs to be down–scaled in order to reduce the dynamic power consumption. A side effect of the reduced supply voltage is an increase in delay unless the threshold voltage is reduced too. This down–scaling has been done for many years, but at the 180 nm process technology the subthreshold leakage current became significant.

Besides the threshold voltage, the oxide thickness has also been down–scaled resulting in an increase in the field across the gate oxide. This has lead to a substantial current through the gate. In figure 2.2 are shown the leakage current mechanisms.

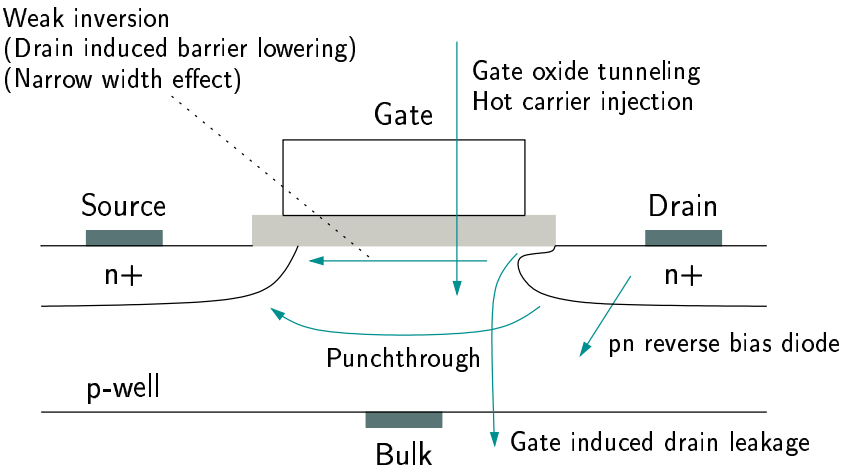

Figure 2.2: Leakage currents of a transistor

Weak inversion, also known as subthreshold, is the drain current that occurs when the gate voltage of a n-channel transistor is less than the threshold voltage $V_{gs} < V_t$.

Two short channel effects are tightly coupled to the subthreshold leakage current in the sense that they modify the threshold voltage: Drain induced barrier lowering (DIBL) reduces the threshold voltage when the drain voltage is increased or when the length of the channel is decreased. The narrow width effect modifies the threshold voltage at gate widths $<$

500nm. Whether the narrow width effect increases or decreases the threshold voltage depends on the technology [24].

The BSIM3v3 [38] model is commonly used for modeling the subthreshold leakage current:

$$I_{sub} = I_0\left(1 - e^{-\frac{V_{ds}}{\nu_t}}\right)e^{\frac{V_{gs}-V_{th}-V_{off}}{n\nu_t}}$$

V_{th} is the zero-bias threshold voltage, $\nu_t = \frac{K_B T}{q}$ is the thermal voltage, V_{off} is the offset voltage and n is the subthreshold swing coefficient.

$$I_0 = \mu_0 \frac{W}{L}\nu_t^2\sqrt{\frac{q\varepsilon_{si}N_{ch}}{2\Phi_s}}$$

here μ_0 is the carrier mobility, W and L are the width and length of the channel, N_{ch} is the channel doping and Φ_s is the surface potential.

As seen from the expression and figure B.3, the subthreshold leakage current increases exponentially with the reduction of the threshold voltage. The supply voltage and temperature also has an exponential impact on the leakage current (figure B.4 and B.5 respectively). The exponential dependency of the temperature is due to a reduction in the threshold voltage [24] and an increase in the thermal voltage ν_t. Finally the leakage current depends linearly on the physical dimension of the transistor (figure B.6).

To reduce the impact of the short channel effects, the gate oxide thickness is scaled in proportion to the channel length [28]. As the oxide thickness is decreased, the electric field at the gate increases resulting in a substantial leakage current *through* the gate. Fowler-Nordheim tunneling happens at high field strengths at the gate, however the effect is negligible at the field strengths used during normal operating conditions [16]. The direct tunneling [28] can be modeled as

$$I_{gate} = WLA_g\left(\frac{V_{ox}}{T_{ox}}\right)^2 e^{-\frac{B_g\left[1-\left(1-\frac{V_{ox}}{\Phi_{ox}}\right)^{\frac{3}{2}}\right]}{V_{ox}}T_{ox}}$$

here A_g and B_g are process parameters, T_{ox} is the thickness of the oxide, V_{ox} is the potential across the oxide and Φ_{ox} is the barrier height of tunneling electron.

The gate leakage is seen to increase exponentially with decreasing oxide thickness. Since the electric field across the gate oxide does not depend on temperature, the gate leakage is temperature independent [24]. Besides the gate oxide tunneling, the hot carrier injection also affects the current going through the gate. It occurs when the channel length is reduced and V_{dd} not is scaled appropriately.

The gate induced drain leakage (GIDL) arises because of a high electric field in the gate-drain overlap region. When there is a large negative bias at the gate (ex. $V_{gate} = 0V$ and $V_{drain} = V_{dd}$), carriers are injected into the substrate and drain resulting in a leakage current. As V_{dd} is lowered, the GIDL contribution to I_{off} decreases, thus for very low voltage processes the GIDL becomes insignificant compared to weak inversion [16].

Punchthrough happens when the source and drain depletion regions merges in the channel due to an increase in V_{ds}. When this happens, the gate is not in control of the channel.

A high electric field across the reverse biased pn junction, causes a significant current. If both n and p regions are heavily doped, band to band tunneling (BTBT) current is the dominating leakage of the pn junction leakage [28].

The BTBT leakage, subthreshold leakage and gate oxide leakage are the most significant leakage currents [28]. In this work the subthreshold leakage is the focus, and therefore the leakage power is simplified to

$$P_{leakage} = I_{sub} \cdot V_{dd}$$

Chapter 3

The leakage problem

In the past years, Moore's law has indirectly helped in reducing the dynamic power consumption, since the shrinking of transistors requires the supply voltage to be reduced. The dynamic power dissipation is a quadratic function of the supply voltage V_{dd}, thus a small reduction in V_{dd} will very effectively help reducing the dynamic power.

$$P_{dynamic} \propto V_{dd}^2$$

However, as the supply voltage has been further reduced, the threshold voltage V_t, will need to be lowered in order not to slow down the transistor. The impact of the threshold voltage on the delay of the transistor, can be modeled by the first order delay model, where $1 \leq \alpha \leq 2$:

$$\text{delay} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}}$$

At the 180 nm process technology, the lowering of supply and threshold voltage resulted in the subthreshold leakage power to become non-negligible, thus a nmos transistor with $V_{gs} = 0V$ could no longer be thought of as being "off". The subthreshold leakage power has an exponential dependency on the threshold voltage, this translates into a very significant power figure for small threshold voltages:

$$P_{leakage} \propto e^{-V_t}$$

When the transistors are shrinked, the gate oxide thickness has to be scaled appropriately for the transistor to operate correctly. Thus for small process technologies, the gate oxide will become very thin and result in a significant leakage current *through* the gate. The gate oxide leakage can be reduced by using a high-κ dielectric for the gate insulator, but such an insulator is not used in production yet.

However Intel has in [27] announced, that a high-κ material has been found and that the material is estimated to be used in the 45 nm process, in the year 2007. At this point, the gate dielectric is expected to change from SiO_2 to the high-κ material, and the gate electrode made of metal instead of the traditional polysilicon. The change will reduce the *gate* leakage by more than a factor of 100.

As the subthreshold leakage power is unaffected by the application of a high-κ material, the focus in this work will be on reducing the subthreshold leakage. Thus in the following, *leakage power* will refer to the subthreshold leakage power component.

3.1 Design trends

In a process where the leakage power is insignificant, *low power* design is a matter of reducing the charging and discharging of capacitances. Methods such as clock gating, power gating or architectural decisions like parallelism instead of serialism are commonly applied. However these methods will not necessarily reduce the total power dissipation when leakage power is considered.

Digital designs can roughly be grouped into the following categories

1. the design where speed is not an issue, low power is of primary concern and prolonged standby periods can be expected. Such a design will typically be a battery powered wearable product such as a hearing aid.
2. the design that occasionally require high speed but also has a demand for a low power mode: A microprocessor in a laptop.
3. the high speed design where external cooling are necessary and the cost of power supply and cooling are significant. An example of this is a high performance microprocessor used for doing intensive computations.

The three types of designs will need different approaches to reduce the total power dissipation, when a process with high leakage power is used. By defining the activity of the design, α, it is possible to decide whether methods for reducing the *active* power dissipation should be applied, or if the *idle* power dissipation is of more concern.

$$P_{total} = \alpha \cdot P_{active} + (1 - \alpha) \cdot P_{idle}$$

The design of type 1, will have a low activity factor whereas the type 3 design will have a high activity factor.

Entering or leaving an *idle* mode are not necessarily free in terms of power. In figure 3.1 is sketched how the power dissipation might change upon entry and exit of the idle mode. Depending on the idle mode scheme used,

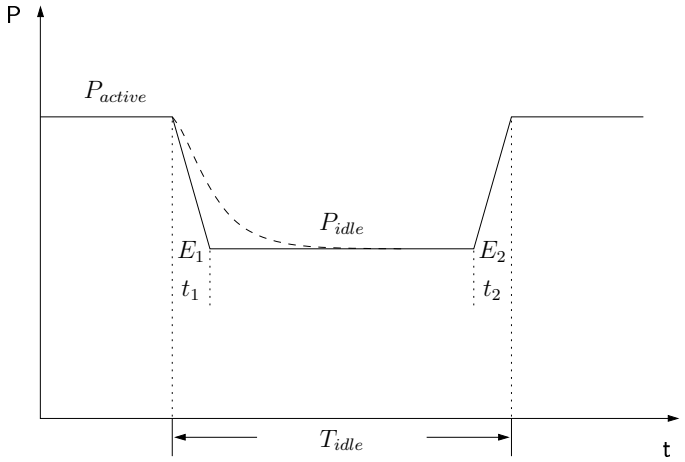

Figure 3.1: Entry and exit of idle mode

energy will be dissipated during the transition time t_1 and t_2, where the idle mode is entered and left, respectively. A method, discussed in chapter 5.2.8, applies a carefully chosen stimuli to a circuit, such that the idle power is reduced. The application and removal of such a stimuli will introduce switching activity in the circuit, and thus dissipate the energies E_1 and E_2.

Having a circuit with the option to enter an idle mode, the condition for

reducing power by entering the idle mode is

$$T_{idle} > \frac{E_1 + E_2 - (t_1 + t_2) \cdot P_{idle}}{P_{active} - P_{idle}}$$

This expression does not consider the temperature, which affects the sub-threshold leakage power significantly, thus for small values of T_{idle}, the P_{idle} is too optimistic. Considering the type 2 design, the microprocessor of a laptop would transition from a high active operating temperature to a lower idle mode temperature. The change in P_{idle} of such a transition is expected to change like the dashed line in figure 3.1.

3.1.1 Active leakage power

While entering an idle mode, can be an efficient way to reduce the average power dissipation, it obviously isn't an option for the high speed type 3 design. Thus the different types of leakage reduction schemes are typically targeted towards either

Idle mode The circuit is put into a low power mode, and is inoperable while in this mode. Such a mode can for instance be achieved by introducing a virtual V_{dd} and *gnd* controlled by two *sleep*–transistors as in MTCMOS [25].

Active mode The circuit is required to operate at full speed all the time. The leakage power has to be reduced without decreasing the speed of the circuit.

The reduction of the active leakage power can efficiently be done by utilizing that *only some* paths in a circuit, will be time critical. Thus logic gates that not are part of the critical path will have excess slack, that can be traded for a reduction in leakage power.

Two methods for reducing leakage, one by modifying the threshold voltage, the other by using large logic gates, are discussed in chapter 4.

Chapter 4

Methods for reducing leakage currents

Reduction of leakage power will be done with the use of two methods which both involves the modification of the threshold voltage V_t. One method directly increases the threshold voltage, thereby also introducing an increased delay of the transistor. The other method uses a second order effect, where the threshold voltage indirectly will be increased of some of the transistors in a stack.

To evaluate how these methods impacts the characteristics of transistors, a 70 nm Berkeley Predictive Technology Model [39, 7], for use with HSpiceare used. The parameters for the models, and the *model cards* for the process, are listed in appendix B.1. The MOSIS [36] design rules for deep submicron technology are used, such that transistor with reasonable dimensions can be simulated. The design rules of interest are shown in appendix B.2.

4.1 Threshold voltage

The threshold voltage is from equation 2.3 seen to relate exponentially with the gate–source voltage of the transistor. Thus, from figure 4.1, the source–current at $V_{gs} = 0$V will decrease significantly as the threshold voltage is increased. This implies that from a leakage power perspective, a high

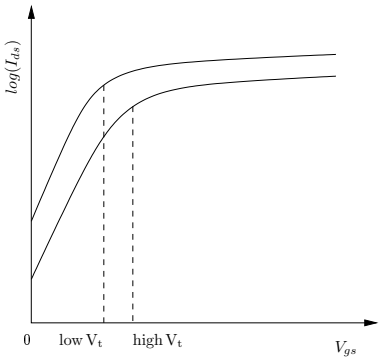

Figure 4.1: Source–current with different threshold voltages

threshold voltage is desirable. However, at $V_{gs} = V_{dd}$ the maximum current through the transistor will be reduced as the threshold voltage is increased. Given a capacitance C_L, that has to be charged and discharged at a regular rate, the transistor with the lowest threshold threshold voltage, will be able to move the charge faster than a transistor with same dimensions, but higher threshold voltage. This will be shown in the following example.

4.1.1 Impact on the static power dissipation

A 70 nm technology inverter shown in figure 4.2, with the design rules from appendix B.2 is simulated with HSpice. The spice deck is listed in appendix D.2.1, and the method used for computing power consumption with HSpice, is described in appendix C. The results of the simulation will show how power and delay interacts at different threshold voltages for this particular inverter.

The inverter that is simulated, is driven by a low V_t inverter, I_1, to get a realistic transition on signal a. The load C_L on signal b corresponds to the load of four minimum sized inverters, and $25\mu m$ of interconnect without any neighboring wires. The calculation of capacitances for gate and interconnect are performed in appendix B.3 and B.4 respectively. Both transistors $M1$ and $M2$ have equal dimensions, and I_1 is sized such that signal a has equal rise and fall times.

Table 4.1 shows the leakage power and table 4.2 shows the delays of the

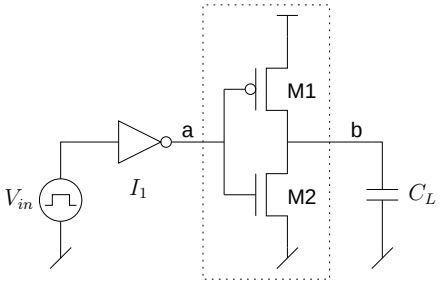

Figure 4.2: Inverter test setup

minimum sized inverter operating at a temperature of 70° C. As seen from the *average* columns, the behavior changes as expected: A low threshold voltage equals small delays and a higher threshold voltage introduces more delay. The use of mixed threshold voltages allows some characteristics in–between the single threshold voltage implementations.

$M1$	$M2$	$P_{leakage,\ a=gnd}$	$P_{leakage,\ a=Vdd}$	$P_{leakage,avg}$
Low V_t	Low V_t	30.8 nW	17.4 nW	24.1 nW
Low V_t	High V_t	508.8 pW	17.4 nW	8.9 nW
High V_t	Low V_t	30.8 nW	219.0 pW	15.5 nW
High V_t	High V_t	512.8 pW	219.0 pW	365.9 pW

Table 4.1: Leakage power for minimum-sized inverter (70 C)

$M1$	$M2$	$\frac{t_r}{[\text{ps}]}$	$\frac{t_f}{[\text{ps}]}$	$\frac{t_{pd,lh}}{[\text{ps}]}$	$\frac{t_{pd,hl}}{[\text{ps}]}$	$\frac{t_{pd,avg}}{[\text{ps}]}$
Low-V_t	Low-V_t	111.4	45.0	55.0	26.7	40.8
Low-V_t	High-V_t	111.3	63.0	54.9	39.3	47.1
High-V_t	Low-V_t	171.5	45.0	88.7	26.5	57.6
High-V_t	High-V_t	171.3	63.0	88.7	39.1	63.9

Table 4.2: Delays of minimum-sized inverter (70 C)

4.1.2 Impact on the dynamic power dissipation

While the previous analysis showed that the leakage power decreases as the threshold voltage is increased, the following analysis will look at what

happens to the dynamic power component, as the threshold voltage is modified.

A transistor with high threshold voltage, is expected to have slower transitions when driving a load C_L, than the equivalent transistor, but with lower threshold voltage. This will affect the short circuit power dissipation, given by equation 2.2, of the transistors driven by the fanout. However, the transistor with high threshold voltage, will itself have lower short circuit power and thus the total dynamic power dissipation may be unaffected.

A test setup, as shown in figure 4.3, is used with the same specifications as from figure 4.2. The spice deck is listed in appendix D.2.2. As seen from table 4.3, the total leakage power of D_1 and D_2 decreases as expected, as the number of high V_t relative to low V_t transistors are increased.

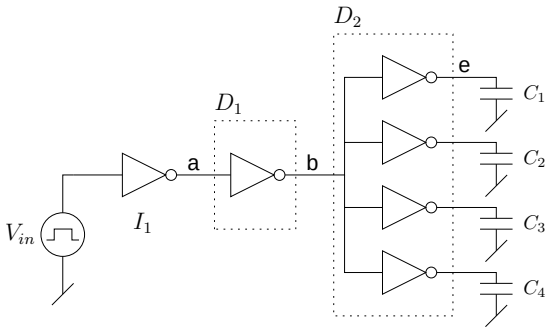

Figure 4.3: Inverter with a fanout of 4

$D1$	$D2$	$\frac{P_{leakage,\ a=gnd}}{[\mathrm{nW}]}$	$\frac{P_{leakage,\ a=Vdd}}{[\mathrm{nW}]}$	$\frac{P_{leakage,avg}}{[\mathrm{nW}]}$
Low V_t	Low V_t	100.73	140.40	120.6
Low V_t	High V_t	69.95	123.10	96.5
High V_t	Low V_t	31.71	19.42	25.6
High V_t	High V_t	1.43	2.24	1.8

Table 4.3: Leakage power for D_1 and D_2 (70 C)

The dynamic power of the circuit is consisting of the charging of capacitances and the short circuit power. Since the structure of the circuit is unaffected when the threshold voltage is modified, all capacitances, except

the gate capacitances, can be assumed to be constant. The gate capacitance is affected as a result of a change in T_{ox}, see appendix B.3.

Changes in switching energy, will be a result of changes in gate capacitances and changes in short circuit power, when the interconnect is assumed constant. As the short circuit power from equation 2.2 increases, when the slew is increased, it is expected that changing inverter D_1 in figure 4.3, from low V_t to high V_t, may increase the total switching energy. By making the inverter D_1 high V_t, it will itself impose a reduction in short circuit power, as the high V_t transistors has lower saturation current.

Table 4.4 lists the switching energy for a rising and falling transition of signal a, as well as the sum of these. From the table, it is seen that the switching energy for this particular circuit, increases by 14% when the threshold voltage of inverter D_1 is increased with inverters D_2 being low V_t. Thus, if the signal a toggles at a sufficiently high frequency, then the total power dissipation of D_1 and D_2 may increase above the both–low V_t circuit, even though the leakage power has been reduced. When all transistors are high V_t, then the total switching energy is less than for the all–low V_t circuit. Therefore it is likely, that the dynamic power of a circuit will decrease slightly, as the fraction of transistors assigned high V_t, are increased.

$D1$	$D2$	$\frac{a\uparrow}{[fJ]}$	$\frac{a\downarrow}{[fJ]}$	$\frac{a\uparrow\downarrow}{[fJ]}$	$\%\frac{a\uparrow\downarrow}{[fJ]}$
Low V_t	Low V_t	6.66	5.94	12.6	100.0%
Low V_t	High V_t	5.02	5.48	10.5	83.3%
High V_t	Low V_t	8.15	6.28	14.43	114.5%
High V_t	High V_t	5.06	5.48	10.54	83.6%

Table 4.4: Switching energy

As the power dissipated in charging and discharging interconnect capacitances, in general are significantly larger than the short circuit power, the effects of the threshold voltage will be relatively small compared to the total dynamic power dissipation. This will be shown in chapter 6.4.

4.2 Stacking effect

The *stacking effect* is the reduction in subthreshold leakage current when multiple series–connected transistors are "off". Consider the stack of nmos

transistors shown in figure 4.4. All transistors are assumed to be off, $V_g =$

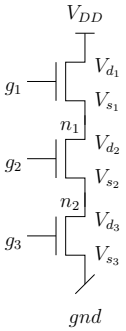

Figure 4.4: Transistor stack of 3 NMOS transistors

$0V$, and there will flow the same subthreshold leakage current through all transistors. Equivalating each transistor with a resistor, there will be a voltage drop across the resistors because of the current. Due to the voltage drop, the voltages at the nodes of the stack will be $V_{dd} \geq V_{n_1} \geq V_{n_2} \geq gnd$.

As the voltage at node n_2 is slightly positive with respect to gnd, the gate–source voltage of transistor 2 will be slightly negative $V_{g_2 s_2} < 0$. This negative gate–source voltage will, as seen from equation 2.3, reduce the subthreshold leakage current. Furthermore as the source–bulk voltage is increased, the threshold voltage will be increased due to the *body effect* [26], and indirectly reduce the leakage current. Finally the leakage current will be further reduced, as the drain–source voltage is decreased because of the raised voltage of node s_2.

Figure 4.5 shows a three input nand-gate with minimum sized 70 nm transistors and the histogram in figure 4.6 shows the leakage power of each state; the dotted line is the average leakage power of all states. The difference in leakage power for the different states are very significant. When all nmos transistors are conducting, the voltage at the output z of the nand gate will be $0V$, thus the source–drain voltage of all pmos transistors will equal V_{dd}, which is worst case seen from a leakage power point of view.

From the histogram it is clear, that it is advantageous to have long transistor stacks. A logic function implemented in a large logic gate are likely to have less average leakage power, than the same function implemented by several fewer–input gates. Thus the synthesis of a design, can benefit from

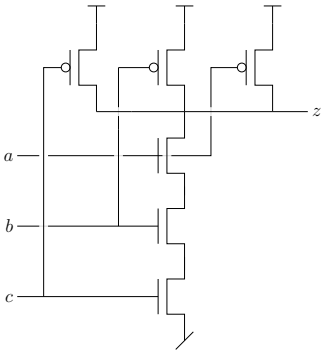

Figure 4.5: 3-input NAND gate

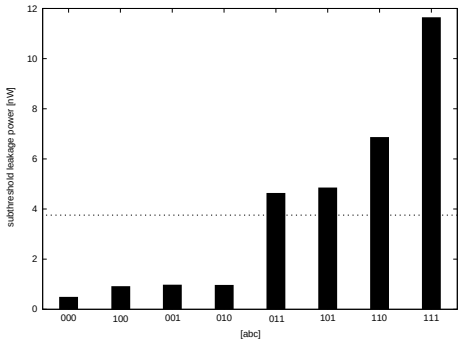

Figure 4.6: State dependent leakage power of a 70 nm 3-input NAND gate (25 C)

having a variety of large logic gates in the cell library, if low leakage power is of primary concern. However, as the delay of a logic gate increases as the number of transistors in a stack are increased, the practical use of very large logic gates might be limited. This will be analyzed further in chapter 6.

Chapter 5

Logic synthesis

Logic synthesis transforms a description of a digital circuit into a network of gates, a *netlist*. Typically designs are specified in a hardware modeling language such as VHDL or Verilog, but other methods such as signal transition graphs or plain boolean expressions are widely used among certain research fields.

When a hardware modeling language is used for describing a digital circuit, the level of detail used in the model directly affects the development time of the model and the correlation between the model and the final synthesized design. Traditionally the level of detail are divided into:

Behavioral The effort is put into specifying what an algorithm does, without many considerations about the underlying technology.

Register transfer level When the algorithm is specified, the location of registers (eg. memory elements) are determined, and the operations in-between the registers are described with arithmetic expression or flow control (`if-else`) structures.

Gate level The circuit is specified with the use of gates provided by a technology library. A design specified at this level is not easily portable to another technology.

As the size and complexity of designs are increasing, and there is a demand for a short development cycle, the use of automated synthesis is crucial. A modern design flow is shown in figure 5.1. The *RTL synthesis* is tightly coupled to architectural decisions, thus if a `z=a+b` expression is found in

the RTL code, the RTL synthesis will, guided by the constraints, select an appropriate implementation style (ripple carry adder, carry lookahead etc.) for the addition. The netlist provided by the RTL synthesis is then during the *logic synthesis* optimized in terms of delay, power, testability or any constraint specified.

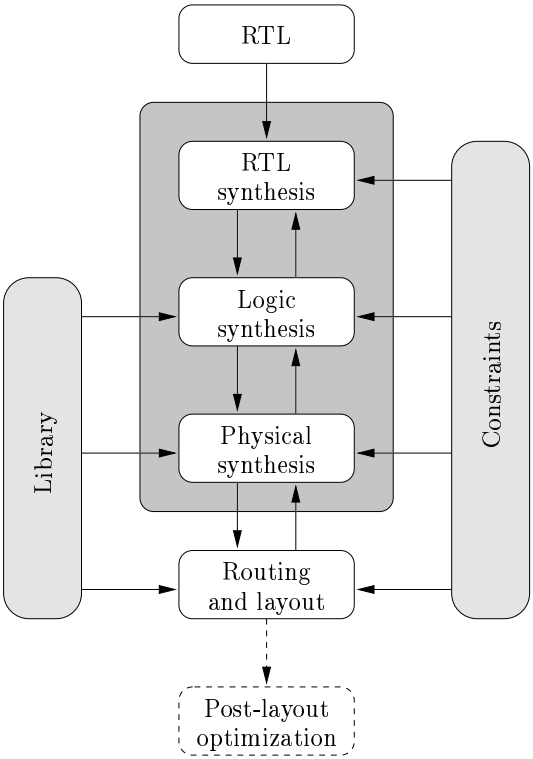

Figure 5.1: Design flow

A recent addition to the design flow is physical synthesis. As the device sizes shrinks, the early estimation of capacitances of the final design are imprecise and thus timing closure can be hard to reach. By performing the placement of gates while optimizing the logic, the length of wires are known and thus a better estimate of the capacitances are available. This estimate can be used to properly size gates or change the structure of the circuit, ex. insert buffers on wires that are longer than initially expected. Finally

the design is routed and it is verified that the final layout complies with the constraints. If a step in the synthesis fails to meet its requirements, the parent step must be revisited. By the introduction of the physical synthesis, the connection between logic netlist and layout has been made stronger, thus there is a potential for decreasing the number of iterations needed to complete a design.

The *post-layout optimization* is not part of the common design flow, but for subthreshold leakage power reductions, this step should not be neglected. In contrast to the preceding synthesis steps where the confidence of the capacitance estimates are questionable, the knowledge about capacitances are optimal at post-layout. By using algorithms that can benefit from the detailed information about capacitances, the leakage power can be reduced significantly.

Overview of logic synthesis

By providing a list of requirements that the optimized design shall fulfill, it is the job of the synthesis tool to apply suitable optimization algorithms and modify the circuit, such that the requirements are satisfied. As logic synthesis is a complex optimization problem with limited time to solve, there are two things the designer can do, given a synthesis tool:

- Write the specification of the circuit in a way that will allow the synthesis tool to fully exploit the optimizations algorithms.
- Guide the optimization algorithms in the synthesis tool by providing reasonable constraints.

Typically the optimization of a design is a NP-hard problem, and since it is infeasible to search the entire solution-space, heuristics are often applied. A problem with heuristics is that it is impossible to guarantee that the *optimal* solution has been found, instead the quality of the solution, the *cost*, can be measured against the requirement, in figure 5.2 is shown an example. Solution S_1 in the figure, is a local minima but it satisfies the requirement. If the algorithm tries small changes in either direction, the cost metric will be increased and the incorrect conclusion could be drawn that this is the best solution. In order to find the optimal solution S_2, then worst case the entire solution-space has to be searched, clearly not an option with limited development time. The best way to find a good solution, is by specifying a maximum cost, and then hope that the heuristics will find a solution that satisfies the cost.

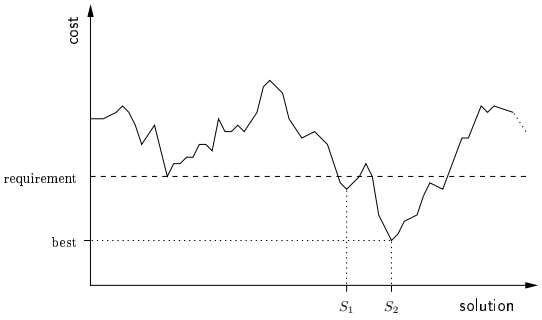

Figure 5.2: Synthesis heuristic

The netlist provided by the RTL synthesis is the input to the logic synthesis. Since synthesizing logic can be a complex and very time consuming process, it is not unusual to first optimize the netlist without in depth knowledge of the target technology. After the optimization, the optimized circuit is then mapped to the cells of the target technology. These two steps are known as technology independent optimizations and technology mapping, respectively. This separation of the two optimization phases can cause a sub-optimal cell mapping of the circuit, but for the purpose of analyzing how leakage power can be implemented in technology mapping, this is of minor concern.

5.1 Technology independent optimizations

The netlist provided by the RTL synthesis are likely not to be in a form, that are optimal for mapping directly onto logic gates by the technology mapping algorithm. Thus by performing logic optimizations on the circuit, followed by a decomposition into a tree of two-input NAND-gates and inverters (NAND2/INV), the circuit (tree) can be mapped to logic gates. The NAND2/INV tree is completely technology independent and does not contain any information about delay, it is just a functional representation of the circuit.

Two-level sum-of-products logic optimization methods or karnaugh maps, are not suitable for large digital designs. The reason for this is that it is unlikely that a large design consists solely of two-level logic. If two-level

logic optimization methods are to be used, then the multiple-level logic has to be reduced into a two-level representation, optimized and then refactored back into a multiple-level representation. This will introduce lots of unnecessary computations. Instead, decomposition by the use of multi-level algebraic methods such as cube and kernel factoring, are commonly used since they perform well on big circuits [23].

In the following it is assumed that an optimized logic network has been computed, and is to be mapped for low leakage power.

5.2 Technology mapping

Technology mapping is the process of transforming an unbound optimized logic network into a netlist of gates from a technology library. The technology library specifies the characteristics of gates (power, delay) and characteristics of the interconnect (capacitance per unit of length). By application of a variety of algorithms, an unbound network is iteratively mapped with different combinations of cells, until a solution is found that satisfies the requirements.

Assuming two cell libraries are available, one low V_t and the other high V_t, the goal of the synthesis for low leakage power is to find a solution with minimum leakage power. It is assumed that the all–low V_t design *just* is able to satisfy the delay constraint, and that the equivalent circuit using high V_t cells, not is able to satisfy the timing constraint. Thus the optimization problem is to tradeoff speed for leakage power, by choosing a high V_t cell instead of a low V_t cell, on paths that are not delay critical.

The commonly used methods to reduce the dynamic power of a circuit are either to *hide* high switching activity nodes inside big cells, or reduce the switched interconnect capacitance [37]. Neither of these methods reduce the leakage power significantly, but by using the dual V_t cell library, and restructure the circuit during the technology mapping process, more cells can be assigned high V_t thereby leading to a reduction in leakage power.

Technology mapping methods

The methods used to map a logic network to a set of cells from a library can be categorized as being either rule based or heuristic algorithms. With a

rule based method, the logic network is stepwise bound to cells. This is done by executing a *large* set of rules in a predetermined order, and applying each rule to the circuit, if it matched. Figure 5.3 shows some examples of rules [23]. While rule a and b are simple boolean rules, rule c is more complex since is can change the circuit to speed up the critical path. Rules

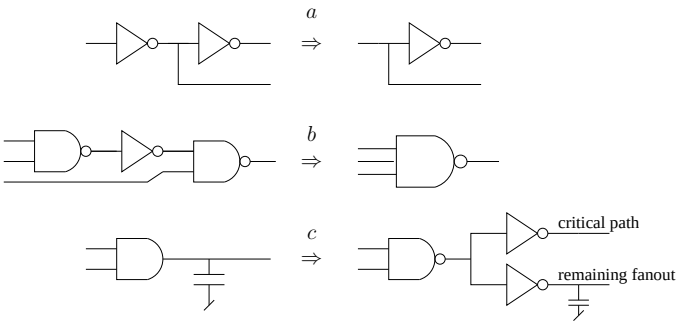

Figure 5.3: Examples of rules

have the benefit that they can be used with *any* type of cell, even multi–output cells. The drawbacks of rule based systems are an unpredictable runtime, and there is a risk that the rules will get "stuck" doing intensive local optimizations instead of focusing on the overall circuit [23]. In order to get a good result with a rule based system, many rules needs to be specified and maintained as the parameters of the cell library are changed. Considering a dual V_t cell library, the number of permutations of threshold voltages used in the rules, will increase very fast. Thus a rule based system is not ideal for doing dual V_t leakage power reduction.

An algorithmic method on the other hand, can relatively easy be extended to take the threshold voltage of cells into consideration during technology mapping. If the optimized logic network and the logic function of cells from the cell library, are represented as directed acyclic graphs (DAG), then by *matching* the graphs, the logic network can be bound to cells.

Figure 5.4 shows the flow of technology mapping. The wireload model and gate delay model are used to give a figure of the speed of the circuit. The cell library represents the cells that can be matched to the DAG of the circuit and the cost functions are used to evaluate the quality of the matching.

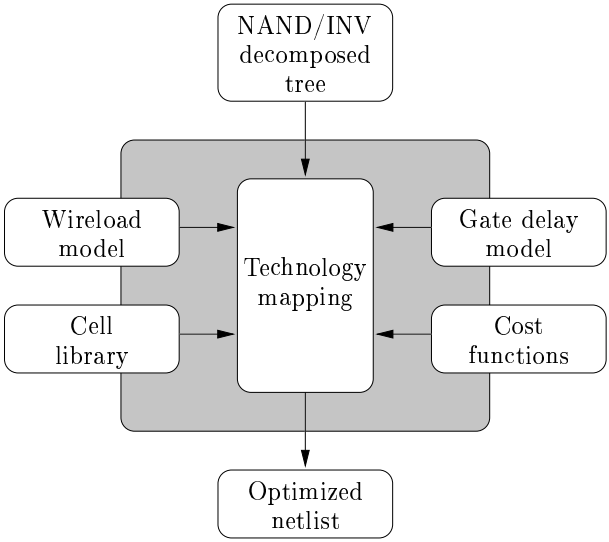

Figure 5.4: Technology mapping flow

5.2.1 Technology library

For a huge digital design, with a critical time to market factor, full custom design is rarely feasible. A commonly used alternative is to use a pre-characterized technology library containing models of functional elements and interconnect. In between these two approaches are automatic on-the-fly generation of cells, where the synthesis tool can request the creation of a cell with unique characteristics. However as with full custom design, both the description of logic characteristics and the layout, has to be performed, thus there is a significant increase in synthesis time compared to when using pre-characterized cells.

In the following pre-characterized cells will be considered, thereby allowing the synthesis tool to try several combinations of cells in short time, while performing technology mapping.

Cell model

The functional elements, or *cells*, are pre-characterized by the vendor of the technology library. This relieves the designer of the cumbersome task of specifying functional behavior of the cells in form of Spice netlists, performing physical layout and characterization of the cells for use by the synthesis tool.

In order for the synthesis tool to use a cell, some capabilities must be available in the cell library:

Function Assuming that all cells have a single output, the logic function is expressed as $z = f(inputs)$.
Capacitance The load capacitance of each input and the output z.
Power Dynamic power for each input as a function of input slew and output load. Leakage power for all combinations of the inputs.
Delay Transition delay for each input as a function of the load.
Size The area of the cell.

The listed capabilities are the minimum required. For a commercial library format such as Liberty [15] by Synopsys, the above list are supplemented by a set of design rules (ex. a limitation on the maximum fanout of a cell) and physical layout data. The characterization of a cell library, for use by Synopsys, can with benefit be characterized with the use of GSpice as described in [9].

Wireload model

A wireload model is used by the synthesis tool to estimate characteristics (capacitance, resistance, area) of a wire with the absence of physical layout data. The wireload models are usually supplied by the vendor of the cell library, and are created based upon a set of designs that the vendor thinks are representative, thus the models are not guaranteed to be suitable for *all* designs.

The generation of a wireload model are typically based on statistical data of a set of representative designs which have been synthesized, placed and routed. After routing, all capacitances for a given fanout are extracted and collected in a histogram, figure 5.5 shows an example. Using the histogram, the capacitance for say 90% of all nets with a fanout of f_o, can be determined. This model is very coarse grained and the usefulness is

questionable, however it can be easily improved by including the size of the design into the statistical collection of data. To improve the model, it is assumed that the average length of nets with a fanout f_o, is smaller for a little design, than the average length of nets with the same fanout f_o for a big design. By using this assumption, histograms can be created based upon the size (area or gate count) of the design, and the appropriate wireload model can be selected.

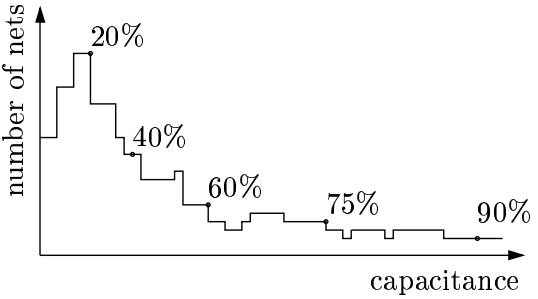

Figure 5.5: Example histogram of a wireload model

Another aspect is to choose the wireload model based upon the type of logic in the current design. Models could be created for regular arithmetic structures, random logic or interconnect used to connect blocks at the top level of the design, however the accuracy of wireload models are facing major problems in deep submicron designs, so the benefit of such detailed models will be insignificant. These problems includes effects such as crosstalk and degrades the accuracy of wireload models.

The accuracy of a wireload model has great impact on the final design. The ideal wireload model is able to predict the timing of the post-layout design, however such a model is hard to create. If the wireload models are too optimistic, such that the design is synthesized with less wire capacitances than present at post-layout, there is a big risk that timing closure will be impossible to reach. On the other hand, a pessimistic wireload model will force the synthesis tool to make the design faster than needed, possibly with increased area and power as a consequence.

5.2.2 Gate delay model

To enable calculation of delays in a circuit, the circuit is assumed to be represented as a directed acyclic graph $G(V, E)$, where the vertices V corresponds to logic gates and the edges E maps to paths (interconnect). By traversing all paths from inputs to outputs of the circuit, and accumulate the delays of all gates along paths, the delay and slack of any path are found.

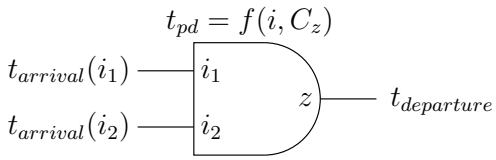

Figure 5.6: Gate delay model

The propagation delay of a logic gate at node n in the graph, is abbreviated $t_{pd}(n)$. As shown in figure 5.6, the propagation delay is a function of the input slew and the load of the gate. This function is pre-characterized in the cell library.

In a graph, the delay of the path j, where j is defined from a primary input to node n is

$$t_{pd,path}(j) = \sum_{nodes \in j} t_{pd}(nodes)$$

In figure 5.7, the highlighted path to node n_6 is $j = \{pi_3, k_1, k_2\}$.

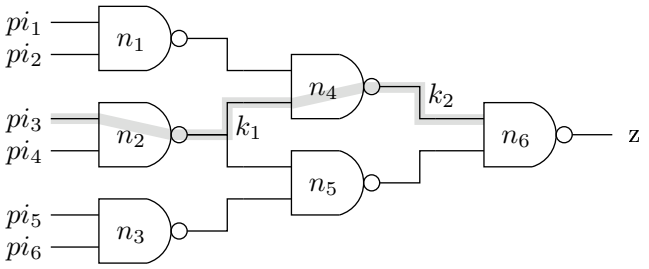

Figure 5.7: Delay along a path

The arrival time of a signal at input i of node n, is the maximum propagation delay of paths ending at input i, thus

$$t_{arrival}(n,i) = \max_{j \in paths\ ending\ at\ i} \left\{ t_{pd,path}(j) \right\}$$

In a similar way, the latest arrival time of all inputs of node n, is defined as

$$t_{max}(n) = \max_{i \in fanin(n)} \left\{ t_{arrival}(n,i) \right\}$$

Finally the departure time of node n is

$$t_{departure}(n) = t_{max}(n) + t_{pd}(n)$$

and the critical path of the circuit can be described as

$$t_{critical} = \max_{i \in primary\ outputs} \left\{ t_{departure}(i) \right\}$$

After the delay has been forward propagated through the circuit, the slack can be backwards calculated. For a delay requirement T, the slack of a primary output is

$$slack, po(n) = T - t_{departure}(n)$$

for logic gates that are not a primary output, the slack is recursively defined as

$$slack(n) = \min_{k \in fanout(n)} \left\{ T - t_{departure}(n) - slack(k) \right\}$$

The delay of interconnect net l, is assumed to be included in the delay function of the logic gate that drives net l. Thus for a net with a fanout, all logic gates that are connected to the net will have equal arrival time.

5.2.3 Cost function

In order to determine the quality of a logic network mapped to a set of cells from a cell library, a *cost metric* is needed. The cost metric is the result of a cost function which typically is weighing several optimizations goals such as power or area.

When a graph is mapped towards the cells of a library, the cost function evaluates the mapping and grades the mapping with a value, *the cost*. The

best solution has the lowest cost. For simplicity a fanout–free graph is considered, thus a very simple, but fully valid cost function for calculating the cost of mapping logic gate g to node n is

$$Cost_{area}(g,n) = area(g) + \sum_{i \in inputs(g,n)} Cost_{area}(g_i)$$

Computing the cost of power dissipation is more complex, mostly because of the dynamic power dissipation. Given a mapped circuit as shown in figure 5.8, the cost in terms of power is to be calculated. As leakage power

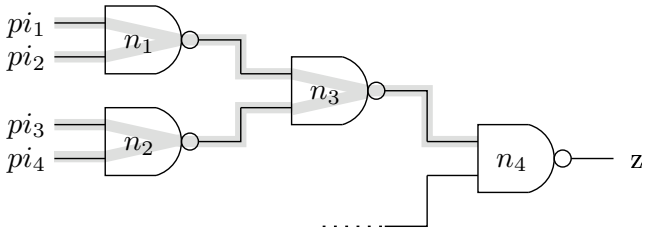

Figure 5.8: Cost for node n_3

is very state dependent, and dynamic power depends on the activity of the circuit, it is beneficial if the switching activity of the inputs are specified on beforehand. Naturally this cannot be known for any type of logic, but if there is a pattern in the data, ex. an instruction of a microprocessor that has much higher probability of being executed than other instructions, then this pattern should be applied. Most important is, that it is the pattern of typical data that the circuit will be exposed to, that are used as guide during the calculation of cost.

Starting at a node n of the mapped graph, the cost function will propagate backwards and visit the nets and logic gates of all paths passing node n. By definition, the cost of the fanin cone ending at node n_3 in figure 5.8, also includes the net driven by n_3.

Based upon the cost function given in [21]:

$$power(g,n) = power(g) + \sum_{i \in inputs(g,n)} min\_power(i)$$

where the cost of mapping a gate g to node n is computed, the cost function considering the total power consumption is developed. The total power of logic gate g is calculated as in 2.1.

It should be noted that the gate capacitance of logic gates driven by g, are included into the total load capacitance of g, thus

$$C_L(g,n) = C_{interconnect}(g,n) + \sum_{i \in fanout(g,n)} C_{gate}(i)$$

The dynamic power can then be calculated as the power required to charge $C_L(g,n)$ plus the diffusion capacitance of g. The short circuit power is dependent on the load $C_L(g,n)$, and the leakage power can either be computed with the specified switching activity, or just a simple average of the leakage power of all states.

$$\begin{aligned} Cost_{power}(g,n) \quad = \quad & \alpha \cdot \big(P_{dynamic}(g) + P_{short\ circuit}(g)\big) + \\ & (1-\alpha) \cdot P_{leakage}(g) + \\ & \sum_{i \in inputs(g,n)} \Big(Cost_{power}(g_i, n_i)\Big) \end{aligned} \tag{5.1}$$

The activity parameter α, where $0 \leq \alpha \leq 1$, specifies whether the effort should be put into mapping the circuit for low idle power, or focus on reducing the active power. The benefit of such a parameter is that a regular design can be synthesized for low active power dissipation whereas a battery powered application with prolonged idle periods can be synthesized with less leakage power at the cost of an increase in active power.

Both cost functions are only valid for a fanout free circuit. Since circuits typically *do* have fanouts, both models needs to be extended in order to be useful. By dividing the recursive part of the cost function, by the number of fanouts of the net, the cost function can effectively handle fanouts.

$$Cost_{area}(g,n) = area(g) + \sum_{i \in inputs(g,n)} \frac{Cost_{area}(g_i)}{fanout(i)}$$

A similar modification can be done to $Cost_{power}(g,n)$.

In case there are restrictions on both area usage and power dissipation, the two cost functions can be combined. The constant β in the expression, is a scaling factor that specifies which of the cost functions are most important.

$$Cost_{total}(g,n) = (1-\beta) \cdot Cost_{area}(g,n) + \beta \cdot Cost_{power}(g,n)$$

5.2.4 Mapping a graph

The mapping of a graph to cells from a library, is a problem of first identifying how the graph can be mapped. When the possible mappings are identified, they are one by one evaluated if they satisfy the delay and power requirements. This evaluation is repeated until a mapping is found that satisfy the requirements.

As the input to the technology mapper is a graph of NAND2/INV gates, the logic function of the cells from the library, needs to be decomposed into a similar NAND2/INV structure. Figure 5.9 shows the decompositions of some simple logic gates.

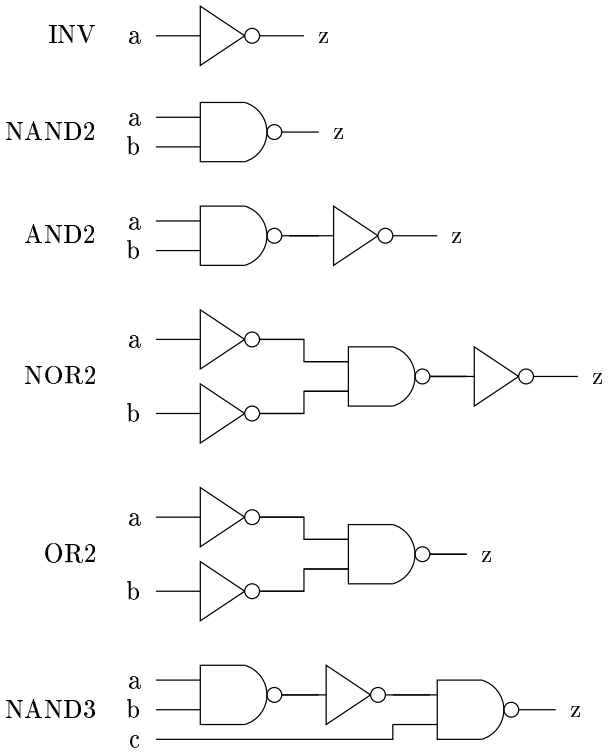

Figure 5.9: Pattern graphs of sample cell library

As long as the logic gate is symmetric and has a small number of inputs,

there only exists one decomposition of the function. However as the number of inputs grows, there exists several valid decompositions of the function, this is a weakness of the graph matching method. The four–input NAND gate, as shown in figure 5.10, is an example of this.

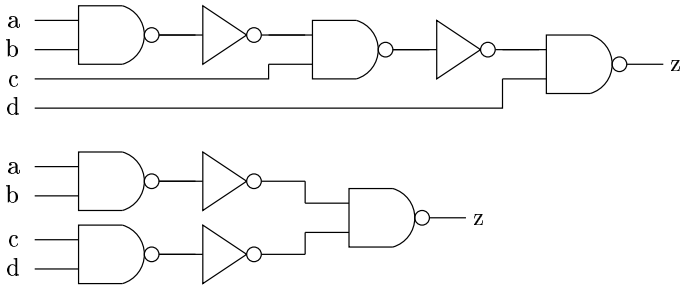

Figure 5.10: Pattern graphs of NAND4

Table 5.1 lists the area, average leakage power of all states and average propagation delay for all inputs, for the logic gates shown in figure 5.9 and 5.10, for a 180 nm process [33, 34].

		Low leakage		High speed	
Name	$\frac{Area}{[\mu m^2]}$	$\frac{P_{leakage}}{[\mathrm{pW}]}$	$\frac{t_{pd}}{[ps]}$	$\frac{P_{leakage}}{[\mathrm{pW}]}$	$\frac{t_{pd}}{[ps]}$
INV	8.19	13.96	68	268.74	55
NAND2	12.29	23.27	77	341.43	70
AND2	20.48	31.42	150	532.14	124
NOR2	12.29	23.66	80	505.39	64
OR2	16.38	34.40	155	770.36	127
NAND3	16.38	32.29	91	445.17	73
NAND4	24.58	41.61	106	555.35	85
NAND5	49.15	83.04	223	1314.28	184

Table 5.1: Characteristics of decomposed logic gates

To map a subject graph such as the one shown in figure 5.11, to a set of cells, the *matching* of cells needs to be performed first. Matching is the task of comparing the topology of patterns from the subject graph with the decomposed cells of the library. As the logic function is implicitly given by the topology, the matching process does not need to do explicit verification of the function. In the example graph, a two input NAND-gate and an

inverter will match four places and an AND-gate matches three places, other matching are also possible.

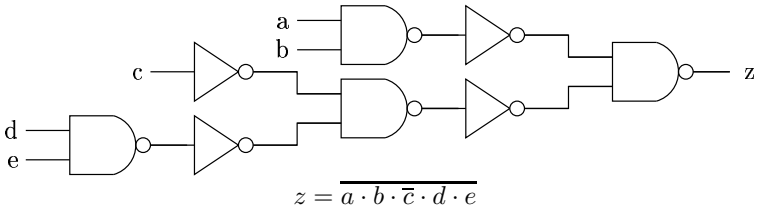

Figure 5.11: Example subject graph

Following the matching, is the *covering*. After the matching has identified which cells *can* be used, the task of covering is to select a set of the matchings, such that all nodes of the NAND2/INV graph are covered. The cover is then analyzed to verify that it satisfies $t_{critical} \leq T$ and has an acceptable cost. Figure 5.12 shows examples of some of the possible covers of the subject graph from figure 5.11. The corresponding costs are listed in table 5.2. Notice that the minimum leakage power is obtained at the least area usage.

Mapping	$\frac{Area}{[\mu m^2]}$	$\frac{P_{leakage}}{[\mathrm{pW}]}$
1	81.92	148.92
2	53.24	103.92
3	49.14	92.5
4	40.96	78.83
5	53.25	86.99
6	53.25	86.99

Table 5.2: The costs of coverings from figure 5.12

To allow for any graph to be efficiently matched, a pair of inverters can be inserted on all edges of the graph. An example is shown in figure 5.13. Afterwards when doing the covering of the graph, the remaining inverter pairs can be removed without the risk of modifying the logic function of the circuit.

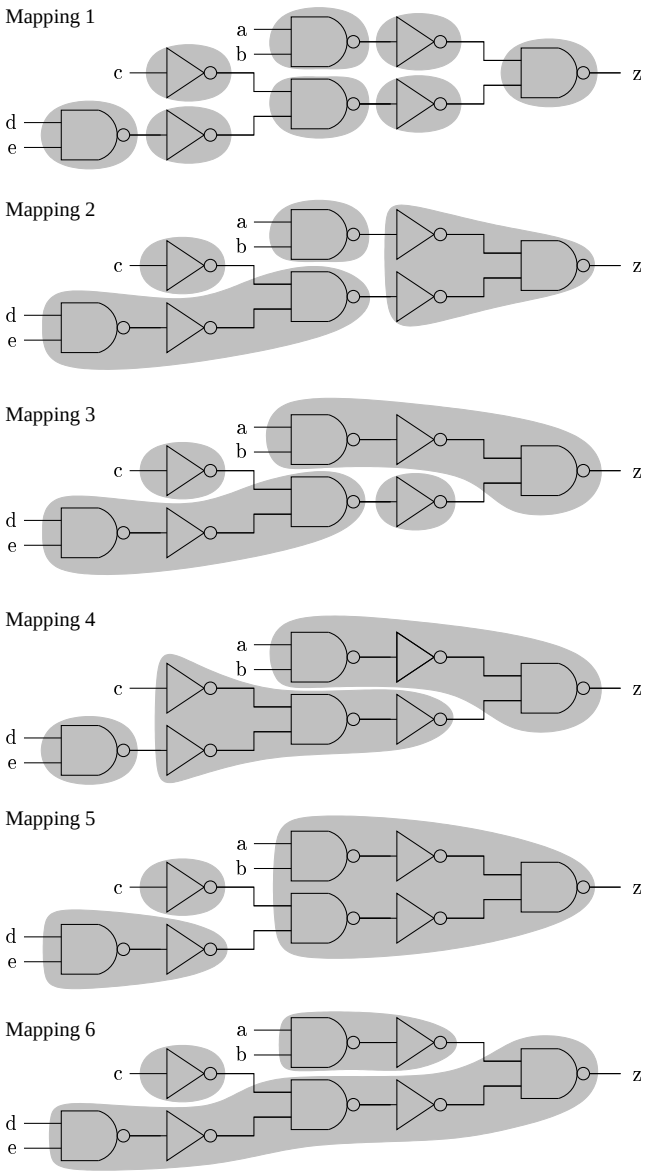

Figure 5.12: Example coverings of the subject graph from figure 5.11

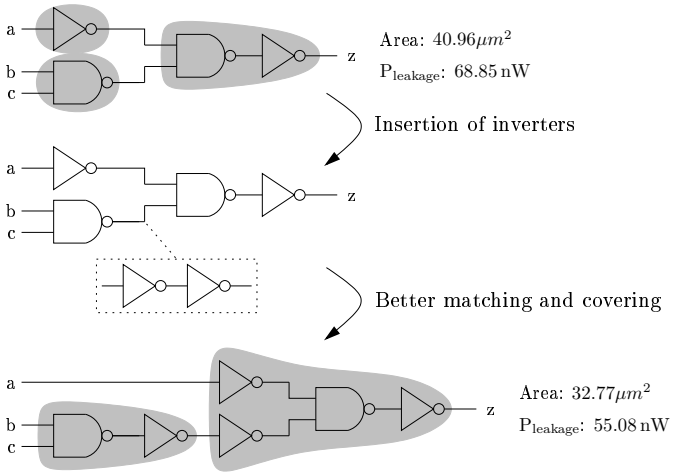

Figure 5.13: Insertion of inverters to improve matching

Mapping a graph with fanouts

If the decomposed circuit isn't a simple tree, but instead is a directed acyclic graph with fanouts, then the covering is a NP hard problem [23]. For small circuits, the solution space to the covering problem is limited, thus the general matching and covering methods still applies, an example is shown in figure 5.14.

However, as the size of the circuit increases, the fact that the covering problem is NP hard, requires a different approach for the covering of the graph. A reasonable way to solve such a problem is to partition the DAG into a forest of trees, see figure 5.15, and find an optimal solution for each tree by perform matching and covering. When a solution is found for all trees, the coverings of the trees are joined to give a total solution for the entire graph.

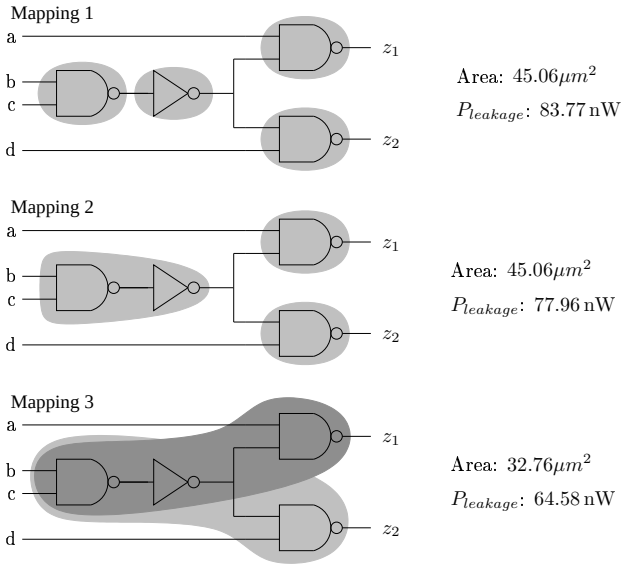

Figure 5.14: Example NAND2/INV graph with a fanout

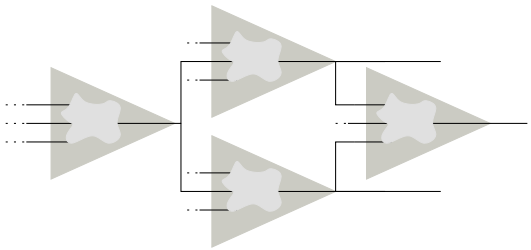

Figure 5.15: General DAG with fanouts

5.2.5 Reducing leakage power by increasing area

Table 5.2 shows that leakage power tends to increase as the area is increased. While this is the case for circuits using a single threshold voltage and no delay constraint, the behavior changes when a dual V_t cell library and a delay constraint are introduced. Three mappings of a two-input OR gate are considered, see figure 5.16.

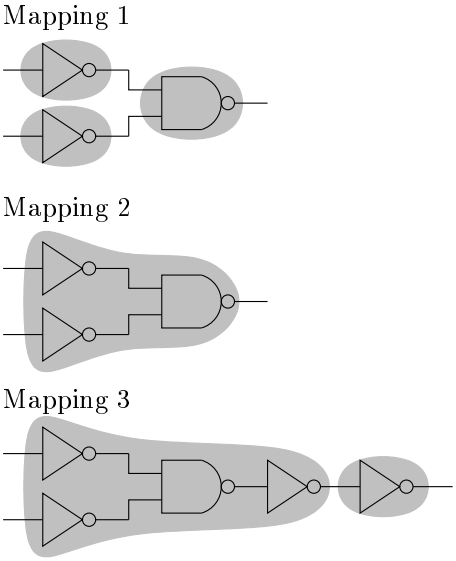

Figure 5.16: Mappings of a OR2 function

In table 5.3 are listed the leakage power for different mappings of the OR function. Note that an inverter pair has been inserted to allow mapping 3. For this simple function, the minimum leakage power for both threshold voltages are found where the area is least, however an interesting question arises: Does minimum area *always* implies the minimum leakage power? The answer is *no*, as will be shown by example in the following.

A circuit consisting of an OR function is synthesized with a delay constraint of 140 ps and only a low V_t cell library, furthermore the synthesis tool is not aware of leakage power. Since there is sufficient time, mapping 2 from figure 5.16 is chosen, because it results in least area and has less $C_{interconnect}$

Mapping	$\frac{Area}{[\mu m^2]}$	Low leakage $\frac{P_{leakage}}{[\mathrm{pW}]}$	Low leakage $\frac{t_{pd}}{[ps]}$	High speed $\frac{P_{leakage}}{[\mathrm{pW}]}$	High speed $\frac{t_{pd}}{[ps]}$
1	28.67	51.19	145	878.91	125
2	16.38	34.40	155	770.36	127
3	20.48	37.62	148	774.13	119

Table 5.3: Area and leakage power of mappings in figure 5.16

than the other mappings. Finally the post-layout leakage reduction tool determines a slack of 13 ps at the OR function, but this slack is not sufficient to substitute the gate by the high V_t equivalent, thus the leakage power of the OR function remains at 770.36 pW.

If leakage power is considered during synthesis, then this leakage power figure can be improved. In table 5.4 are calculated the leakage power and delay of different permutations of high speed and low leakage cells. Clearly mapping 1 and 3, which not are the least area mappings, results in less leakage power when the all–low leakage mappings cannot satisfy the delay constraint. In this particular example the leakage power, by choosing mapping 3 and making the NOR gate *low leakage*, is reduced to 40% of the leakage power obtained with the minimum area mapping.

Mapping		$\frac{P_{leakage}}{[\mathrm{pW}]}$	$\frac{t_{pd}}{[ps]}$
1: $\mathrm{INV_{hs}}$,	$\mathrm{NAND_{ll}}$	560.75	132
1: $\mathrm{INV_{ll}}$,	$\mathrm{NAND_{hs}}$	369.35	138
3: $\mathrm{NOR_{hs}}$,	$\mathrm{INV_{ll}}$	519.35	132
3: $\mathrm{NOR_{ll}}$,	$\mathrm{INV_{hs}}$	292.40	135

Table 5.4: Leakage power for different permutations of threshold voltages

Even though the leakage power is decreased, the total power dissipation might not decrease when using mapping 1 or 3. Both mappings introduce extra capacitance in form of interconnect, and if the gates are excited in a way, that results in excessive switching of the added interconnect, then the total power could be increased. Whether this happens, depends on the circuit and the stimuli, and needs careful analysis by the use of the cost function.

Mapping a function for delay, has great resemblance with mapping a function for minimum leakage power, however where the delay mapping oper-

ates globally on the entire circuit, the leakage power mapping also optimizes locally. A function on a path, that is not part of the critical path of the circuit, will not be subject to delay reduction by the traditional delay mapping. On contrary the leakage power aware mapping might select a mapping, that will result in less delay, given a low V_t cell library, for then afterwards using slack to assign high V_t to *some* cells of the function.

From the example can be generalized and concluded, that using large gates exclusively, will prevent a fine grained dual V_t assignment in the region where the large gate just has insufficient slack to be assigned high V_t.

5.2.6 Resizing

In section 2.3 was shown that the leakage power changes exponentially with the threshold voltage and proportional with the physical dimension. From a manufacturing point of view, it is highly impractical and costly to have continuous threshold voltages, but the size of the transistors on the other hand, can be scaled continuously to affect the time a load C can be charged or discharged. By the availability of a discrete number of sizes of each logic gate in the cell library, the leakage power can be reduced by choosing an appropriate size *and* threshold voltage during mapping.

Figure 5.17 shows an inverter I with a fanout of f. The problem is to decide the size and threshold voltage of the inverter such that the delay requirements are satisfied, and the total power dissipation is reduced.

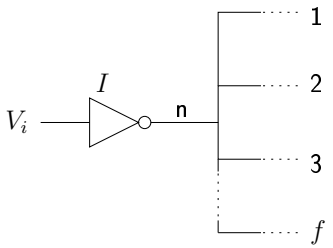

Figure 5.17: Inverter with a fanout of f

Assuming a delay requirement d, there are four options:

1. Inverter assigned low V_t, reduce width to reduce the leakage power

2. Inverter assigned low V_t, increase width to reduce the delay
3. Inverter assigned high V_t, reduce width to reduce the leakage power
4. Inverter assigned high V_t, increase width to reduce the delay

By changing the size and threshold voltage of the inverter, all terms of the total power dissipation of the circuit are affected. Not only will the leakage power of the inverter itself change, but the power dissipation of the surroundings will change too. If the size of the inverter is decreased or the threshold voltage is increased, the rise/fall times of the net n driven by the inverter will increase. Using expression 2.2 for the short circuit power, the logic gates driven by net n will show an increase in $P_{short\ circuit}$, as a result of the rise/fall times.

Instead of reducing the size of the driving inverter, the size could be increased thereby reducing (within limits) the delay of the inverter, but also increasing the leakage power. As the size of the inverter is increased, the diffusion capacitance and gate capacitances will be increased, thus the dynamic power from charging capacitances will increase. In figure 5.18 is shown the inverter from figure 5.17 but in a larger context. The increased gate capacitance of I_4 will increase the load on net n_1, thus the rise/fall times of n_1 might be increased leading to increased $P_{short\ circuit}$ of I_2, I_3 and I_4. Furthermore when the load is changed on net n_1, the timing needs be recalculated for all logic gates driven by the net to ensure that none of the logic gates "surprisingly" has become part of the critical path.

			Low leakage		High speed	
Name	Drive strength	$\frac{Area}{[\mu m^2]}$	$\frac{P_{leakage}}{[\mathrm{pW}]}$	$\frac{t_{pd}}{[ps]}$	$\frac{P_{leakage}}{[\mathrm{pW}]}$	$\frac{t_{pd}}{[ps]}$
INV	1×	8.19	13.96	68	268.74	55
INV	2×	12.29	20.81	60	417.96	48
INV	3×	16.38	33.06	56	663.96	45
INV	4×	16.38	40.37	51	833.89	42
INV	8×	20.48	79.16	49	1654.80	40

Table 5.5: Leakage power and delay of an 180 nm [33, 34] inverter with different drive strengths

From table 5.5, a high V_t inverter with 8× drive strength has almost same delay as a 2× drive strength low V_t inverter, but the leakage power is only a fifth. Even though increasing the size of logic gates can decrease the dynamic power consumption [4, 18], the dynamic power are likely not to

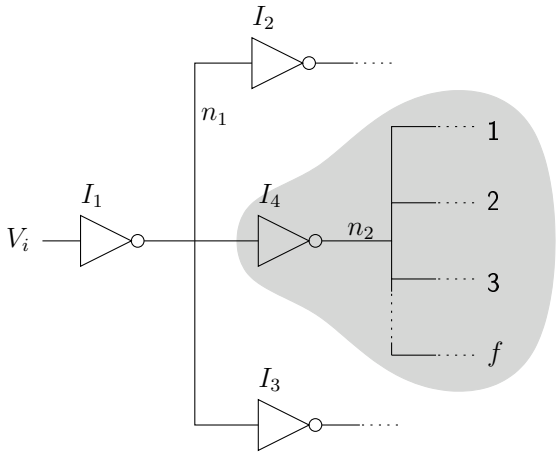

Figure 5.18: Inverter in larger context

decrease if high V_t transistors are sized to match the speed of low V_t transistors. Such an analysis is performed in [20], where the dynamic power component was found to be very dominating compared to the leakage power, when high V_t transistors were sized to have same delay as low V_t transistors.

Whether it is worth choosing a logic gate with an extremely high drive strength, such that it can be assigned high V_t without a delay penalty, is highly dependent on the switching activity on the input of the gate. The activity, α, of the circuit can be specified in the cost function 5.1, thus if the circuit often is idle, then the high threshold voltage combined with increased drive strength can be beneficial.

5.2.7 Pin reassignment

The delays from all inputs of a logic gate to the output, are not necessarily the same. Consider the NAND gate from chapter 4.2, where the topmost nmos transistor in the pull-down tree will have less diffusion capacitance to discharge than the other nmos transistors. This can effectively be used to assign the latest arriving signal to the topmost (bottommost) transistor of the pulldown (pullup) tree, and thereby decrease the departure time of

the logic gate. The pin permutation of a three-input nand gate can be formulated as:

$$\overline{A \cdot B \cdot C} \Leftrightarrow \{A, B, C\}$$

Meaning that there are no requirements to the order of the inputs.

For the logic gate shown in figure 5.19, the pin permutation is

$$\overline{A \cdot B + C + D} \Leftrightarrow \{A, B\}, \{C, D\}$$

here the two pairs can be interchanged according to the delay requirement.

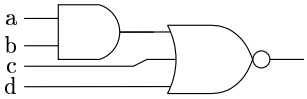

Figure 5.19: Pin permutation of a two–input AND into three–input NOR

Reassignment of pins is commonly used to reduce the delay of a path, but it can also help in reducing the leakage power. If the probabilities of the inputs to the logic gate being in a certain state are known in advance, this can be used to *slightly* reduce the leakage power. If the mapped circuit is simulated with set of test vectors, then the signal with highest probability of being 0 should be assigned an input, given by the pin permutation, which results in least leakage.

The benefit of this leakage power reduction method will be near zero if stochastic data are applied to the circuit. However, if the circuit spends a long time being idle, then the pin reassignment can be used in collaboration with an externally forced stimuli to the circuit, such that an low power mode can be entered.

5.2.8 Idle mode vector

When a circuit is idle, then clock gating is commonly used to reduce the dynamic power dissipation. Clock gating has the benefit that it easily can applied to to a circuit, in an automated synthesis flow, with minimum user interaction. In the following it will be evaluated if the techniques used in clock gating, can be used for leakage reduction too.

By applying a carefully chosen input pattern to a logic gate, the stacking effect can be utilized to reduce the leakage power. There are three problems in doing this:

- Apply an input pattern
- Find the input pattern
- When to apply the pattern

Application of input pattern

To apply a pattern to a circuit D, elements like shown in figure 5.20 are inserted before the primary inputs of a combinational circuit D. By asserting the *idle* signal, the precomputed value of v is propagated to the primary input of D. There are two problems concerning the addition of extra logic when the leakage power are to be reduced. First the added logic will itself

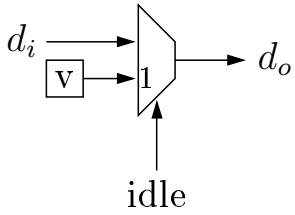

Figure 5.20: Input vector application logic

introduce excess leakage power, second it will increase the delay and thus might prevent logic gates from circuit D to be assigned high V_t. A variant of the logic needed to apply the input pattern is shown in figure 5.21 [1]. By moving the logic into the scan chain, the delay is completely hidden. Furthermore the logic marked in figure 5.21 can be assigned high V_t, since it is outside the critical path, thus the leakage power and dynamic power overhead is not critical.

Finding input pattern

Finding a least leakage input pattern is simple for a small gate such as the NAND3 gate from chapter 4.2. By assuming that all gates in the cell library are pre-characterized for state dependent leakage power, it is just a matter of searching 2^i records in the cell library, where i is the number of inputs to the logic gate and usually has a small value. However the situation is different when the entire circuit is considered.

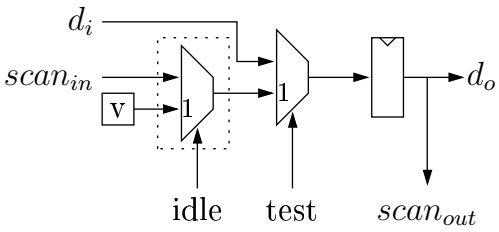

Figure 5.21: Input vector application logic combined with scan chain [1]

To search for an input pattern, a pattern p is applied to the primary inputs of the circuit and propagated throughout the rest of the circuit. Finally the leakage power of all logic gates are summarized and stored: $\{p,\ leakage\ power\}$. Clearly such a search for a least leakage vector will require a significant amount of time as the number of inputs of the circuit are increased.

A different approach is to *generate* a pattern that will result in least leakage power. For a single-level circuit the minimum leakage pattern can be found in linear time, but as the depth of the circuit is increased, the time required will rise exponentially.

When to apply input pattern

Entering the idle mode, requires that the values of the input pattern is applied and propagated throughout the circuit. This process will dissipate dynamic power. Similarly when reentering active mode, there might be a dynamic power penalty. Thus as illustrated in figure 3.1, there is a minimum idle period that has to be satisfied, in order for the input pattern to reduce the total power dissipation.

The decision of when to apply the pattern, are preferable done by a device that can predict the minimum duration of the idle period. This could be a centralized power management control unit or as part of an instruction in a microprocessor. If there no knowledge available of the minimum idle period, then an approach as illustrated in figure 5.22 can be used. By using the clock gating signal of the succeeding flipflop, the input pattern logic can automatically be enabled.

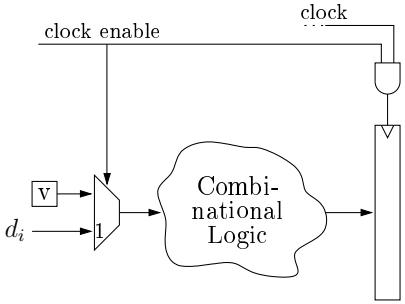

Figure 5.22: Input vector application combined with clock gating

Where clock gating benefits from disabling the clock–signal for just a single clock period, the control unit asserting the clock–enable signal might worst case toggle this signal every other clock period. Because the clock–enable signal lacks the knowledge of duration, the input pattern logic will be heavily exercised and will very likely result in an increase in total power dissipation. Thus it can be concluded that the use of such a setup *most unlikely* will be beneficial, as the minimum duration of the clock enable signal not can be guaranteed to be above a lower limit.

5.3 Incremental dual V_t leakage power optimization

The leakage power of a circuit that is mapped to a set of low V_t cells, possibly the circuit has even been layout, can most likely be reduced by applying an incremental leakage power optimization method. Such a method will visit each cell of the circuit, and substitute the cell with a high V_t functional equivalent cell, if there is sufficient slack.

Using the delay model from section 5.2.2, the extra delay introduced at node n when substituting the low V_t for a high V_t cell is

$$\Delta t_{pd} = t_{pd}(n)_{\text{high } V_t} - \ t_{pd}(n)_{\text{low } V_t}$$

The requirement that needs to be satisfied in order to change the cell is

$$\Delta t_{pd} \leq slack(n)$$

and the condition for sunstituting the cell is, that the power dissipation is reduced.

$$\Delta P_{leakage} = P_{leakage}(n)_{\text{low } V_t} - P_{leakage}(n)_{\text{high } V_t}$$

If only leakage power is considered, then it is sufficient to assume, that the leakage power will decrease as cells are substituted to high V_t.

The general approach of reducing the leakage power incrementally, can be simplified to

1. Calculate slack of all cells in the circuit
2. Select a cell that has not been visited before, and calculate $t = slack(n) - \Delta t_{pd}$
3. If t is positive then assign high V_t to the cell, mark it as being visited, and repeat step 1–3 until all cells have been visited

Depending on the structure of the circuit, the total leakage power reduction will be affected by the order of which the cells are visited, this is illustrated in figure 5.23. All nodes of the graph are initially assigned low V_t and are assumed to be of the same type, thus they have the same delay. It is assumed that the critical path of the graph can only allow for one level to be assigned high V_t, in order to satisfy the delay constraint. If the node at level s_3 is visited first and assigned high V_t, then only *one* high V_t node exists in the graph. If the graph instead was searched from the leaves, and the nodes at level s_1 were assigned high V_t, then the leakage reduction would be improved. If the situation was opposite, that all nodes initially were assigned high V_t, then the aim is to assign low V_t to as few cells as possible. In a real circuit there exists both fanins and fanouts, thus there are no general solution to the problem.

There has been developed numerous algorithms for the selection of which cell to substitute first. The results obtained with the algorithms are quite varying, this is mostly caused by the algorithms being applied to different types of designs, but also parameters such as operating temperature and the technology generation used, affects the leakage power savings. When reporting the total power savings, the activity of the designs are rarely specified, thus the results should only be considered as a guideline.

In [40] and [41] are used a very simplistic method that first identifies cells being part of the critical path. All cells outside the critical path, are then tested if they can be substituted with a high V_t cell without violating the

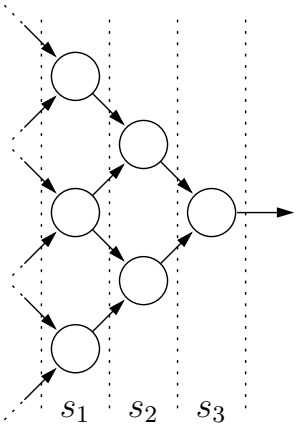

Figure 5.23: Order of V_t assignment

delay constraint. The leakage power is in [40] reduced by around 50% and the total active power dissipation is reduced by 13% for the example design used. The corresponding figures for [41] are 80% leakage power reduction and up to 20% total active power reduction.

The leakage power in [30] is reduced by 84% during active mode and an average of 20% total energy savings, without increase the overall delay of the circuit. By increasing the delay by $4.5\% - 9.5\%$, a 94% leakage power saving and 23% total energy saving are gained.

In [42] is developed a method that assigns a threshold voltage to the individual transistors of a logic gate. Two schemes are proposed:

- There is only one threshold voltage in the entire pullup or pulldown tree.
- There can be mixed threshold voltages in the pullup or pulldown tree, as long as series connected transistors have the same threshold voltage. This restriction is a process limitation. Since the channels of the transistors are very close, it can be hard to explicit change the channel doping over a very short distance.

The benefit from the mixed V_t method is up to 20% more leakage savings than a gate level method, for a collection of ISCAS benchmark circuits [42].

In logic gates with many inputs, the mixed V_t method will result in many

permutations of threshold voltages, thus the number of cells will be huge. Huge cell libraries might enable the selection of a cell that matches perfectly during synthesis, but as the solution-space grows with increasing number of cells, the optimization algorithm are likely to need more time to find an acceptable solution. Clearly this method is better applied in an incremental fashion, optionally post-layout.

The incremental threshold assignment methods all have the benefit, that they can be applied very late in the design flow. This is advantageous as the detailed knowledge of interconnect capacitance post-layout, can be used for optimal timing analysis and thus enable as many cells as possible to be assigned high V_t. Doing the threshold assignment before information about routing capacitance is available, are likely to result in either

- Increased routing time, since the slack of all paths in the circuit are reduced, thus the routing tool will have more time–critical paths to consider, or
- The resulting leakage power saving not being optimal, because the estimate of routing capacitance was too conservative and thus too few cells were assigned high V_t

5.4 A combined optimization method

In order to reduce the leakage power, the circuit must be synthesized in a way that allows many cells to be assigned high V_t. If large logic gates are used, it will be difficult to do a fine grained assignment of threshold voltages.

A way to enable the use of many high V_t cells, is to initially synthesize the circuit with only a high V_t cell library. The optimization algorithms will then be forced to restructure the circuit such that it, at the cost of area, will satisfy the timing constraint. As the cells of a high V_t library generally has more delay than the similar low V_t library, it is expected, that the high V_t synthesis not will be able to satisfy the same delay requirement, as when using the faster low V_t library. At some point, it will be necessary to supply a set of fast low V_t cells, such that the circuit initially synthesized with high V_t cells, will be able to satisfy the delay requirement.

Such an optimization method will heavily affect both leakage and dynamic power. The question is, if an increase in area and thus an increase in

the number of cells, will result in less leakage power than if just a few low V_t cells were used. Another question is, how much the dynamic power will increase, due to the increase in switched capacitance, because of the large number of cells and interconnect. If the dynamic power increases sufficiently, then the benefit from the leakage power reduction will be cancelled and will result in an increase of total power dissipation.

Using the cost function that considers leakage power, a DAG can be mapped with both high V_t and low V_t cells in one pass. It is expected that such a mapping will balance the dynamic power and leakage power better than the optimization method initially starting with only a high V_t library. As the cell library contains cells with different drive strengths, the covering of a DAG will automatically consider the sizing problem as part of the cost calculation of a cover.

5.4.1 Requirements for an implementation

Based upon the previous sections, two critical demands for a tool capable of performing dual V_t leakage power reductions, can be identified:

- Static timing analysis (STA)
- Power analysis

The STA can, depending on the level of detail, be using the delay model from chapter 5.2.2. However for a really accurate timing analysis, submicron issues such as IR-drop and crosstalk should be considered. When only leakage power is analyzed, there is a need for propagating the stimuli throughout the circuit only *once*, under the assumption that glitches are ignored. As the threshold voltage of a cell is changed, only the local ΔP for the already propagated stimuli, needs to be calculated. This will allow the algorithm to operate at a reasonable speed.

For the purpose of evaluating how a dual V_t algorithm could be implemented, the feasibility of using existing tools has been analyzed:

A simple script that interfaces with the command line interface of Synopsys Design Compiler and assigns a threshold voltage to a cell, has been written, see appendix E.4. Basic operations, such as changing a cell from *high speed* to *low leakage* took long time to execute, even on a small circuit. The reason for this is mainly that the user interface isn't meant to do intense modifications to the circuit and that the PERL–wrapper, used

for interfacing with the command line, itself imposes a delay. The static timing analyzer in Design Compiler also reanalyzes the entire design, even though just a single gate has been modified, this will increase the runtime of a algorithm even more. Calculation of power faces the same problem. What is needed, is a way to reuse part of the existing analysis to gain a speedup, however this is not possible.

Synopsys PRIMETIME (a static timing analyzer) is in [19] interfaced by a set of custom scripts in order to assign threshold voltages to cells. This method has shown an improvement up to 70% better than leakage power obtained with Physical Compiler, this is mainly because information about actual capacitances are available and therefore the exact delay of paths are known. In case only the threshold voltage is to be modified, then the post-layout route with PRIMETIME sounds reasonable.

SIS [31] is a synthesis tool developed by U.C. Berkeley. It has a usable delay model incorporated, but neither SIS or the cell library format GENLIB supports power. In [13] has been developed an extension to SIS to enable support for the calculation of only dynamic power, this simplification is reasonable considering the age of the modification. However as SIS does not support multi-level synthesis and the netlist format BLIF, isn't directly compatible with major netlist formats, it has been decided, that it not is worth the effort, to extend the work done in [13] to support leakage power.

The synthesis tool Synopsys Design Compiler, has been chosen for the evaluation of a similar method as the one explained in section 5.4. Design Compiler has been installed and configured as explained in appendix E.3.1.

5.5 Commercial solutions

In the nineties, the increasing leakage power began to attract the attention of several research groups. As the multi V_t methods have been developed and shown to produce good results, the vendors of commercial tools has adopted some of the techniques. The most popular technique, which is supported by most tools, is the incremental assignment of threshold voltage.

SEQUENCE has in [8] announced that their tool PHYSICALSTUDIO, has achieved a 30% reduction in leakage power of a GeForce^{TM} 5900 Ultra GPU (graphics processing unit) designed by NVIDIA. The device consists of 130

million transistors, runs at 500 MHz and is implemented in a 130 nm process. The leakage power reduction was performed a multi V_t cell library and sizing of cells. BLAST POWER [22] by MAGMA supports leakage power minimization by using a multi V_t cell library, and the same is the case for the CADENCE ENCOUNTER platform [6].

5.5.1 ISIS Power

In [29] is presented a leakage power reduction tool by IN2FAB [14], that exclusively can optimize designs post-layout. Given *any* cell based design, without requirements to the synthesis tools used in the development of the design, the cells that are not part of the critical path are substituted with a high V_t equivalent thus reducing the leakage power. The power of this tool, is that from the layout information, the exact capacitance of interconnect can be extracted, instead of using estimates as done during logic synthesis. Thus the static timing analysis, used to decide which cells can be substituted, will be very precise.

Consider a design team that has used the design products of vendor V_1 for years, and have developed scripts and customized the tools to suit their requirements. Next revision of the design will be in the next technology generation to increase the speed, but unfortunately also leakage power. As the synthesis scripts are tailored to the products from vendor V_1, and it is known, that the design has been synthesized correctly with these products, the design team will probably hesitate to instantly change to the products of vendor V_2, which happens to have integrated leakage power reduction methods in the design flow. In such a situation the ISIS Power tool are very useful, however on long term, it can be expected that the tools of vendor V_1 will likely get leakage power optimization integrated in the design flow.

The ideas behind the ISIS Power tool were presented in the autumn 2003 [29], however in the summer 2004, the tool was still in pre-release and no results or *success stories* were publiced. It is questionable if this tool ever will gain a market share in the area of leakage power reduction. Major ASIC design suites, that can be used for the entire development process, has already gotten leakage power reduction capabilities implemented.

5.5.2 Synopsys

Synopsys has presented a multi V_t functionality in [44]. The leakage reduction method operates at the logic synthesis level when using Design Compiler, or during physical synthesis when Physical Compiler is used. At both levels, there are the freedom that the structure of the circuit can be modified, such that more cells can be assigned high V_t. In both tools, a maximum leakage power is specified as a constraint, in the same way, as the maximum delay of a circuit is specified. This has the benefit, that existing synthesis methodologies of a design, only require small modifications to consider leakage power during optimization of the design.

Physical Compiler connects the knowledge of the placement of cells with synthesis of a logic function. The knowledge of the placement of cells, allows the length of interconnect between any two cells to be predicted with good accuracy. Even if the exact position of a single cell isn't known, but instead the position of a cluster of cells are known, the prediction of the overall length of interconnect will be significantly better, than using a statistical wireload model as in section 5.2.1. By knowing the length of the interconnect, and having a model of capacitance per unit of length, the interconnect capacitance can be estimated. This knowledge can be used to potentially get better leakage power savings, than what a similar optimization method is capable of during logic synthesis. However while this estimate of interconnect capacitance is better than a wireload model, it cannot predict the effects of crosstalk. Thus a worst case capacitance per length model will be used, to make sure that the timing requirements can be satisfied after physical synthesis has completed.

The combination of placement and synthesis is driven by a demand to reduce the number of iterations needed to obtain timing closure. An example of this, is wires that turns out to be longer than expected, automatically can be buffered during physical synthesis to reduce the RC delay. By doing the leakage power reduction during physical synthesis instead of during logic synthesis, there is less chance of having a design, where many paths after routing exceeds the delay constraint because of an optimistic wireload model.

The 180 nm cell libraries [33, 34] have with success been converted into a PLIB format as required by Physical Compiler, however a tool compatible with Physical Compiler for creating the floorplan and performing the routing was not available. As long as the final routing is not performed, the

results of the leakage power optimizations done by Design Compiler and Physical Compiler cannot be compared. Thus only logic synthesis with statistical wireload models and no layout will be performed in chapter 6.

Chapter 6

Leakage power reduction with Synopsys

Leakage reduction based on multiple threshold voltages are supported by Synopsys Design Compiler and Synopsys Physical Compiler. This requires the availability of a cell library featuring several instances of logically identical cells, but with different threshold voltages. Furthermore it is beneficial if the physical layout of each cell, especially position and layer of input, output and supply pins are identical for all threshold voltages of the cell. If the physical layout of the cells are identical, then it is possible to substitute low V_t cells with high V_t cells *post-layout.*

When a design is synthesized, a set of parameters are specified which defines the cost function used during the optimization. As part of the multi V_t functionality of Design Compiler, a maximum leakage power constraint can be specified as the cost. This works in a similar way like when a maximum delay of a circuit is specified. Assuming that a design subject to leakage reduction is constrained by an upper limit of the delay, the optimization problem can be defined as *leakage reduction under a delay constraint.*

With two example designs, specified in VHDL, a method like discussed in chapter 5.4, will be used to reduce the leakage power.

6.1 Characteristics of the cell library

A 180 nm cell library [33, 34] available at the department has been chosen for the analysis of leakage power optimizations. This cell library is not ideal for performing *total* power optimization with focus on leakage power, since the leakage power is approximately three orders of magnitude less than the dynamic power. However, there are two reasons why the 180 nm cell library is chosen

Library size Design Compiler is capable of reducing the leakage power by the use of several methods. One of the methods operates at the technology mapping level and exploits the stacking effect by selecting large multi–input gates in favor of smaller few–input gates.
In order for Design Compiler to perform technology mapping for low leakage power, a sufficient[1] number of gates are needed. For the purpose of only evaluating if large cells are used, a cell library in a process with comparable leakage power and dynamic power, is not necessary.

Wireload model In deep submicron technologies, the delays of interconnect are dominating the intrinsic delay of logic gates. Thus besides the characterization of gate delays in a cell library, it is essential to have a good model of interconnect capacitances.
Creating a wireload model with a reasonable accuracy for a deep submicron process, is not a simple task. Today, big deep submicron designs are synthesized with a placement aware synthesis tool such as Synopsys Physical Compiler. Afterwards routing of wires are performed and the capacitance per wire length, for different lengths of wires are calculated as explained in chapter 5.2.1.
The 180 nm cell library already contains a calibrated wireload model used for estimating the capacitance (and hence also delay) of interconnect during logic synthesis.

For the listed reasons, the 180 nm cell library is chosen. Furthermore the *best case* process operating conditions are used (1.6V, $-40°$C), because these, as the only ones, are characterized in terms of dynamic and leakage power.

The cell library consists of 777 combinations of logic functions and drive strengths, the distribution of cells with respect to number of inputs is shown

[1] approximately 120 different gates are used for the processor design, when synthesized with the 180 nm cell library

in figure 6.1. Each gate is available in a *high speed* and *low leakage* version, thus a total of 1554 logic gates with different characteristics are available.

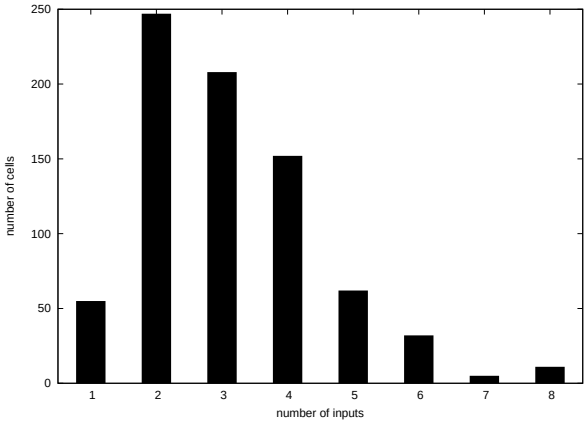

Figure 6.1: Histogram of cells sizes in the cell library

6.2 Optimization methods

Because of the large difference in dynamic and leakage power in the 180 nm technology, the two sources of power consumption are not directly comparable. Minimizing total power consumption will result in application of the traditional methods for reducing dynamic power consumption, since the leakage power is insignificant compared to dynamic power. Instead the leakage power is specified to be the primary optimization target and no optimizations are done towards dynamic power. While this not necessarily will result in a design with low total power consumption, it will show the leakage reduction potential for the design and the impact on dynamic power.

Two cell libraries, one containing fast low V_t cells with high leakage power and the other with slower high V_t cells but less leakage power, allows one to trade off speed for a reduction in leakage power. A design using only low V_t cells will result in great speed but have high leakage power, on the contrary the same design using only high V_t cells, will result in less leakage

power but also a reduction in speed. The goal of the leakage reduction optimization, is to balance the use of low V_t and high V_t cells, such that the timing constraint is satisfied while the leakage power is reduced compared to the all low V_t design.

The methods for reducing leakage power with Design Compiler can be divided into two groups, based on how they are applied:

- A netlist is synthesized with leakage power as optimization goal
- A netlist of an existing circuit is re-synthesized with the aim of reducing leakage power, at the same time the structure of the circuit is *preserved*

The first method is illustrated in figure 6.2. It resembles a common synthesis flow with a netlist (here register transfer level) as input as well as a cell library and optimization constraints. The output is a netlist of the design optimized for leakage power.

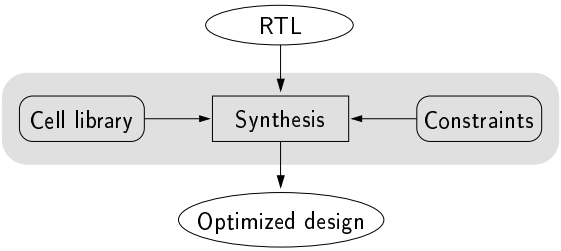

Figure 6.2: One-stage leakage power reduction flow

In the second method, the starting point of the synthesis is different. As shown in figure 6.3, the design is synthesized twice. This is done in order to simulate a situation, where an already completed (synthesis, placement and routing) design, is subject to leakage power reduction, this design will be referred to as the *base design*. If routing has been performed on the design, the capacitance of interconnect can be extracted and used during the leakage power optimization. This will allow the synthesis tool to achieve better results in terms of speed and power, than by using estimates of the interconnect capacitance.

The actual placement and routing, will not be performed in the following. This is because place and route only will provide better estimates of interconnect capacitance, and not will affect the *way* the leakage power reduc-

tion algorithms behave. Thus the results of the leakage power reductions, are likely to be biased in either direction, depending on the correlation between the wireload model and the actual capacitances after routing.

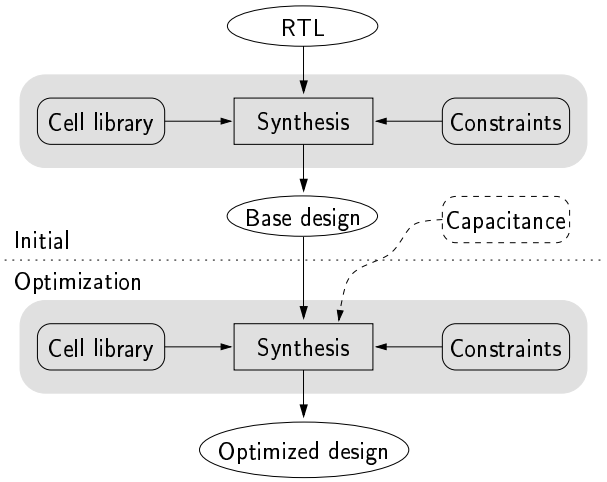

Figure 6.3: Two-stage leakage power reduction flow

In the following, six methods for reducing leakage power with Design Compiler will be presented and in the subsequent chapters the methods will be applied to the reference designs and evaluated. All the proposed methods performs *leakage reduction under a delay constraint*. Doing *delay reduction under a leakage constraint* would be useful for power critical applications, however there are limitations of what can be modified in the cost function used by Design Compiler. As explained in appendix E.1.3, delay always has higher priority than power and area, thus the latter optimization scheme is not easily applicable.

6.2.1 Minimizing area

A simple method for reducing leakage power is to specify an upper limit on the area of the design. Intuitively the leakage power is expected to decrease when the area decreases since the total transistor width is *roughly* proportional with area [5]. By constraining the area, multi–input cells might be

used in favor of fewer–input gates, if allowed by the timing requirements and if the structure of the design are suitable for multi–input gates.

Method 1 *Synthesize RTL code while minimizing area and satisfying timing constraint T.*

However, using area for controlling how the synthesis tool reduces leakage power is impractical and might not result in a lowest total power design. An example of this is a gate with a large fanout. If the gate that drives the net is small, the transition time of the gate increases and results in increased short circuit current in the gates of the fanout. Furthermore *area* cannot be used as means for controlling which cells are chosen from a dual V_t cell library, since the high V_t and low V_t cells of a dual V_t cell library, have the exact same size.

6.2.2 Incremental substitution

Incremental substitution of cells retains the structure of the circuit, hence a cell can only be replaced by another cell, if the logic function is unaffected. By assuming that some paths in a circuit, have less delay than the critical path, the balanced use of both high V_t and low V_t cells, can reduce the leakage power without changing the critical path. By synthesizing the RTL code from figure 6.3 with different timing constraints, the necessity of a circuit structure (the base design) suitable for incremental substitution of cells, will be shown.

There are two approaches for doing the incremental substitution, these are defined as method 2 and 3.

Method 2 *Synthesize RTL code with low V_t cells for timing constraint T, then in the base design, substitute low V_t cells with high V_t cells without violating the timing constraint T.*

Method 3 *Synthesize RTL code with high V_t cells for timing constraint T, then in the base design, substitute high V_t cells with low V_t cells such that the timing constraint T is satisfied.*

The two methods differ in the way that method 2 initially uses the traditional synthesis flow with low V_t cells, whereas method 3 is expected to have increased area usage because of the slower cells.

The quality of the incremental optimization is dependent on the structure of the base design. Therefore it is desirable that the base design is synthesized in a way that enables many cells to be assigned high V_t. By using a stricter than needed timing constraint, the structure of the base design can be modified. This is described by method 4 and 5

Method 4 *Synthesize RTL code with low V_t cells for timing constraint T_1, then in the base design, substitute low V_t cells with high V_t cells, while satisfying the relaxed timing constraint T_2.*

Method 5 *Synthesize RTL code with high V_t cells for timing constraint T_1, then in the base design, substitute high V_t cells with low V_t cells, such that the relaxed timing constraint T_2 is satisfied.*

All the incremental methods suffer from an unfortunate behavior of Design Compiler. Where it is expected that the structure of the design is fully preserved, Design Compiler decides to insert inverters and thereby changes the structure of the design. A side effect of the inverters is an increase in dynamic power because of the extra interconnect. The problem with the inverters, is described in detail in appendix E.1.1.

6.2.3 Dual V_t synthesis

If there are no restrictions regarding the preservation of the circuit structure, the synthesis can be performed with both the low V_t and high V_t cell library in one pass, as described in chapter 5.4. This method will be referred to a method 6. By avoiding the intermediate base design, as illustrated in figure 6.2, and having both cell libraries available during logic synthesis, this method is expected to result in optimal leakage power savings.

Method 6 *Synthesize RTL code with both low V_t and high V_t cells for timing constraint T.*

6.3 Synthesis of adder

For evaluating the leakage reduction capabilities of Design Compiler on a simple but commonly used arithmetic circuit, a N-bit ripple carry adder is examined. The VHDL code is listed in appendix F.2.1 and the synthesis scripts in appendix F.2.2.

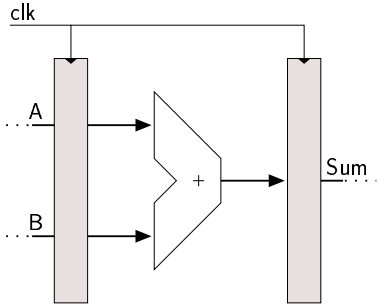

Figure 6.4: Adder encapsulated by registers

The ripple carry adder is constructed of $N = 64$ full adders each performing the logic function:

$$\begin{aligned} carry_i &= (a_i \cdot b_i) + (a_i \cdot carry_{i-1}) + (b_i \cdot carry_{i-1}) \\ sum_i &= a_i \oplus b_i \oplus carry_{i-1} \end{aligned}$$

The maximum delay through the adder will be determined by the delay of the propagating carry and is therefore a function heavily depending on N. The sum expressions will be outside the critical path (expect for the N'th element) and are thus subject to leakage reduction techniques even when the adder is operating at maximum speed.

6.3.1 Optimization results

Method 1 is applied to the adder design and the resulting leakage power and area are shown in figure 6.5. The minimum delay of the design using low V_t cells is $t_{\min} = 1.2$ns whereas the high V_t design has a minimum delay of 1.4ns. In general it is also seen that in order to satisfy the timing constraint, the high V_t has higher area usage than the low V_t design.

The difference in leakage power for the two implementations are for all values of t_{clk} very significant, notice the scale is logarithmic. For large values of t_{clk}, the area of the two implementations converges towards a common minimum size ($17600\mu m^2$), and at this point, the leakage power of the high V_t implementation (40 nW) is only 5% of the corresponding low V_t implementation (800 nW).

As the timing constraint t_{clk} is tightened, the area is as expected, seen to increase. The time at which $\frac{\Delta P}{\Delta t_{\text{clk}}} = -1$, is determined to be $t_{\text{critical}} = 3.0ns$, and if a timing constraint stricter than t_{critical} is specified, both area and leakage power increases significantly.

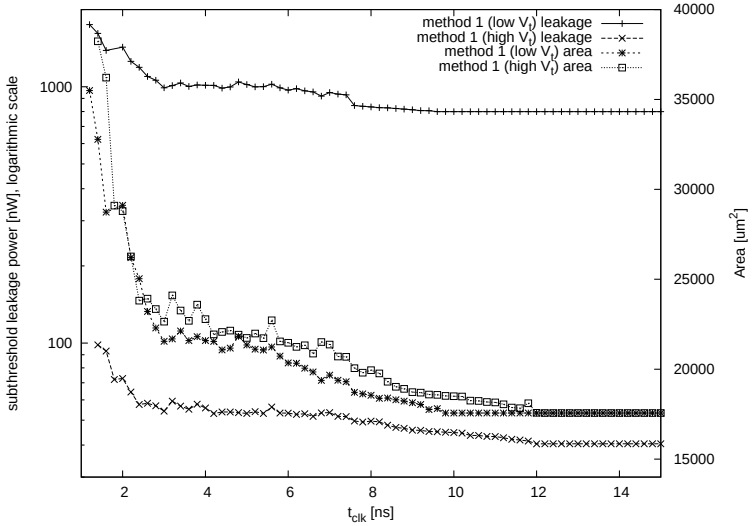

Figure 6.5: Leakage power with area constraint, logarithmic scale

The leakage power after application of methods 2, 3 and 6 is shown in figure 6.6. Method 3 – the high V_t to dual V_t approach, results in lowest leakage power, but from figure 6.7, it generally has slightly higher area usage. On contrary the leakage power obtained with method 2 and 6 is very odd. For $t_{\text{min}} \leq t_{\text{clk}} \leq t_{\text{critical}}$ the leakage power is decreasing, but above t_{critical}, the synthesis tool reaches a *worse* solution in terms of leakage power. This behavior is not what was expected, and is due to the way the synthesis tool selects gates from the cell library, as the following analysis will show.

In figure 6.8 is plotted the percentage of high V_t cells relative to the total number of cells in the design. It is evident that there is a correlation between the leakage power (figure 6.6) and the percentage of high V_t cells. If the leakage power is expressed as

$$P_{leakage} = Cells \cdot (k \cdot P_{\text{high V}_t} + (1-k) \cdot P_{\text{low V}_t}) \qquad \text{for } k \in [0..1]$$

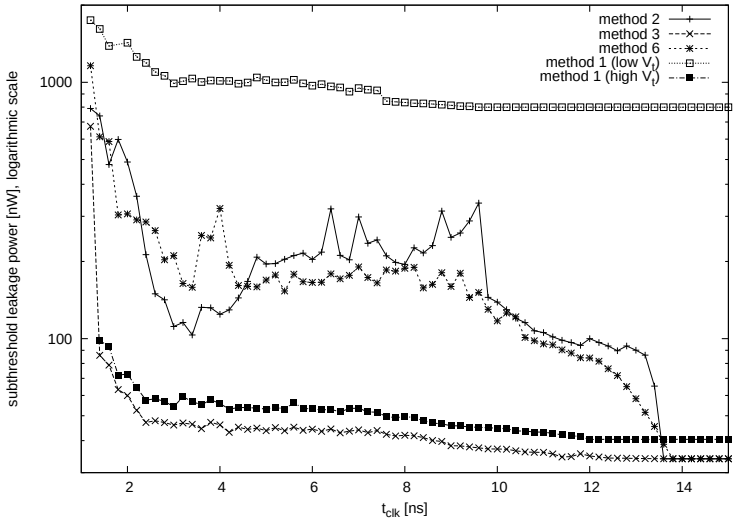

Figure 6.6: Leakage power with variable timing constraint (logarithmic scale)

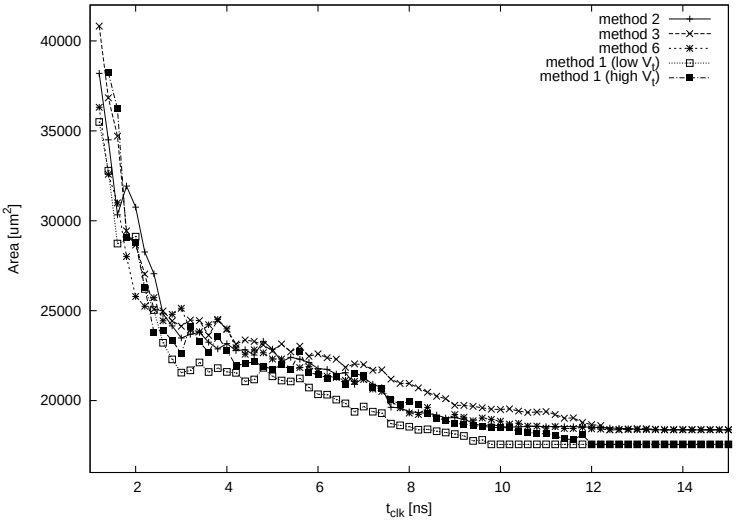

Figure 6.7: Resulting area with variable timing constraint

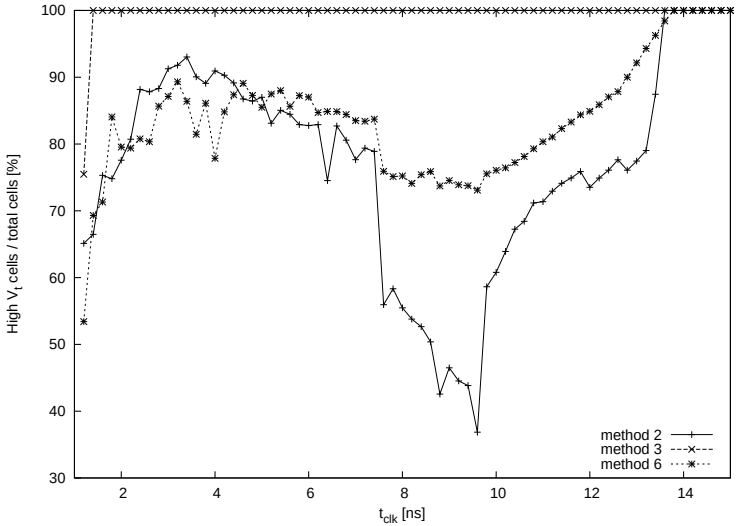

Figure 6.8: Percent high V_t cells with variable timing constraint

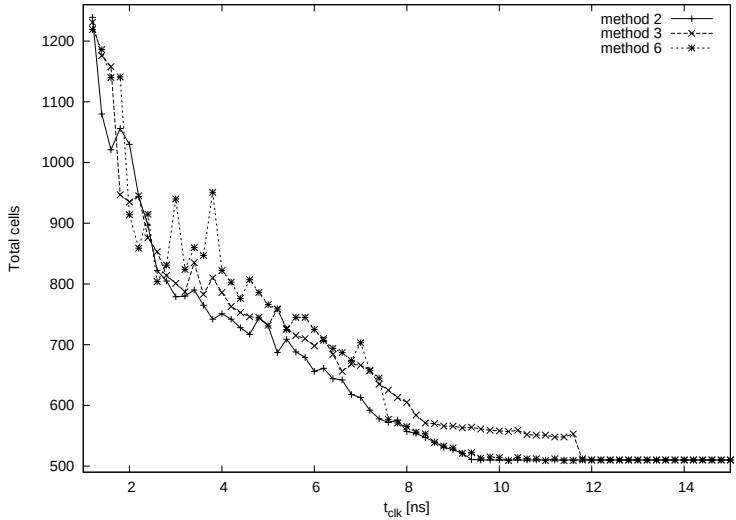

Figure 6.9: Number of cells with variable timing constraint

Where *Cells* is the total number of cells in the design, and k is the fraction of high V_t cells, then the leakage power can only increase, if either the number of cells increases or the fraction of low V_t cells increases. Comparison of the total number of cells in figure 6.9 with the leakage power, indicates that as the number of cells decreases, the leakage power does *not necessarily* decrease. In this particular case, the fraction of high V_t cells decreases as the total number of cells is decreasing.

The distribution of cells, based on the number of inputs for each cell, are plotted in figures 6.10, 6.11 and 6.12. By comparing the cell distributions with the fraction of high V_t cells in figure 6.8, it is clear, that selection of cells with respect to the number of inputs, has great impact on the fraction of high V_t cells in the design. It is especially interesting that the well performing method 3, have postponed the use of *big* 5-input gates until $t_{clk} = 11.8$ns. Thus it can be concluded that for this design, for certain timing constraints, the use of big cells reduces the opportunity for assigning cells high V_t instead of low V_t. Method 6 was expected to produce the best leakage power savings, but for an unknown reason, the synthesis tool makes some bad decisions when choosing cells, resulting in the same behavior as method 2.

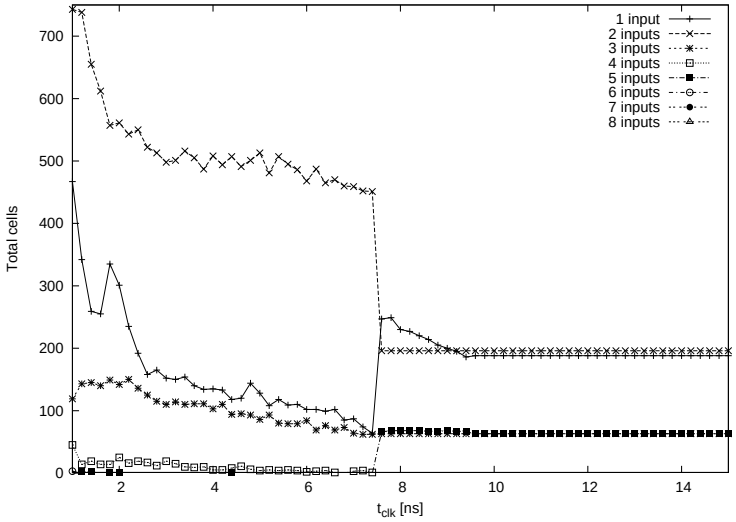

Figure 6.10: Distribution of cells with method 2

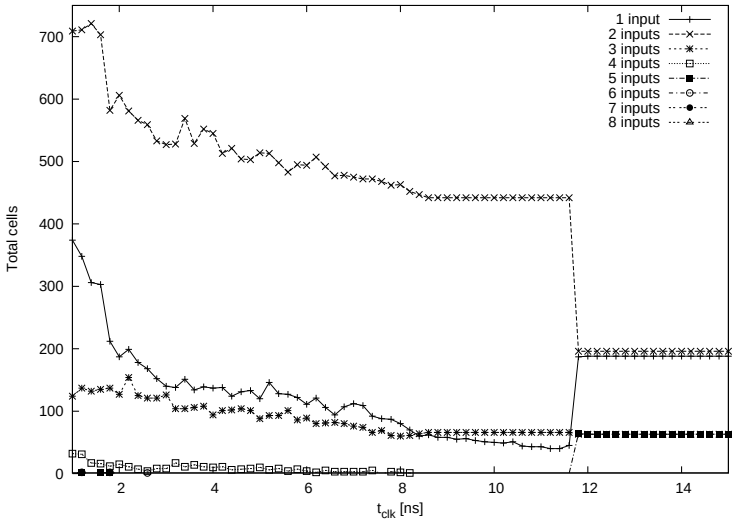

Figure 6.11: Distribution of cells with method 3

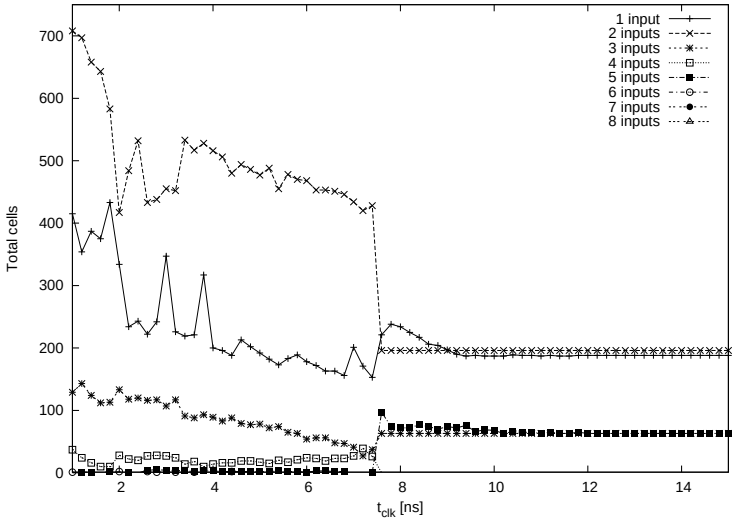

Figure 6.12: Distribution of cells with method 6

The dynamic power is shown in figure 6.13. Method 2, 3 and 6 results in approximately the same dynamic power, however for small values of t_{clk}, method 3 has highest dynamic power of the three methods, this is due to the high area usage. As the timing constraint is slackened, the dynamic power of the area constrained method 1 is the lowest. This is as expected, since the switched capacitance from the interconnect is reduced compared to the non-area constrained methods. Furthermore the high $\mathrm{V_t}$ area constrained method, have less dynamic power than the corresponding low $\mathrm{V_t}$ method, this is caused by the high $\mathrm{V_t}$ cells dissipates less short circuit power.

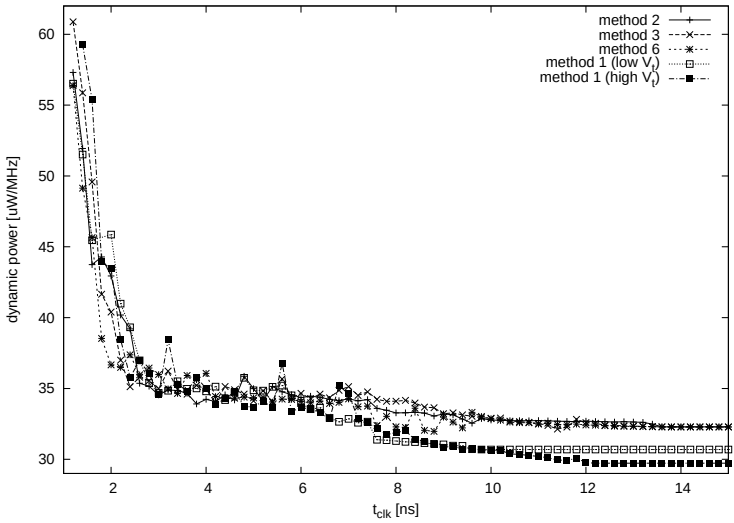

Figure 6.13: Dynamic power with variable timing constraint

By using method 4 and carefully selecting a good starting point from the area plot in figure 6.5, the leakage power reduction can be improved significantly over method 2, as seen in figure 6.14. The initial structure of the circuit designed for $t_{\mathrm{clk}} = \mathrm{T_1}$ is preserved, but by lowering the operating speed, more paths will get sufficient slack to enable $\mathrm{V_t}$ substitution. The corresponding method 5, shown in figure 6.15, has a similar behavior. However, as the leakage power reduction obtained with method 3 is very good already, there is no benefit from using method 5 for this design.

The impact on the dynamic power are shown in figures 6.16 and 6.17. It is

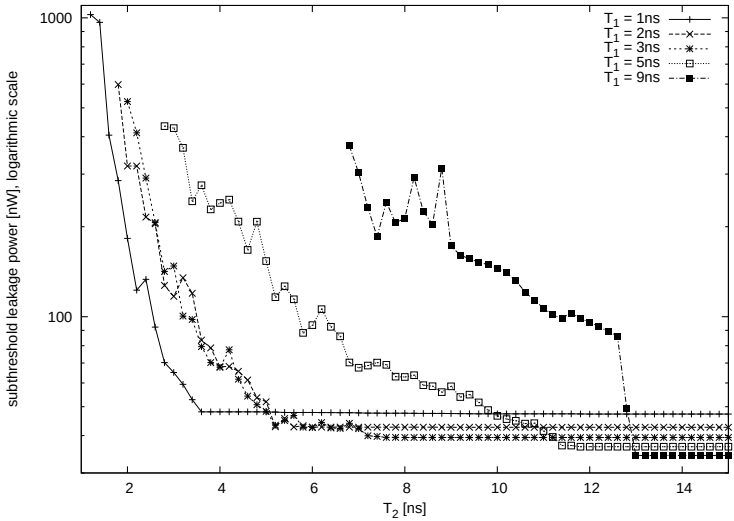

Figure 6.14: Leakage power with method 4, logarithmic scale

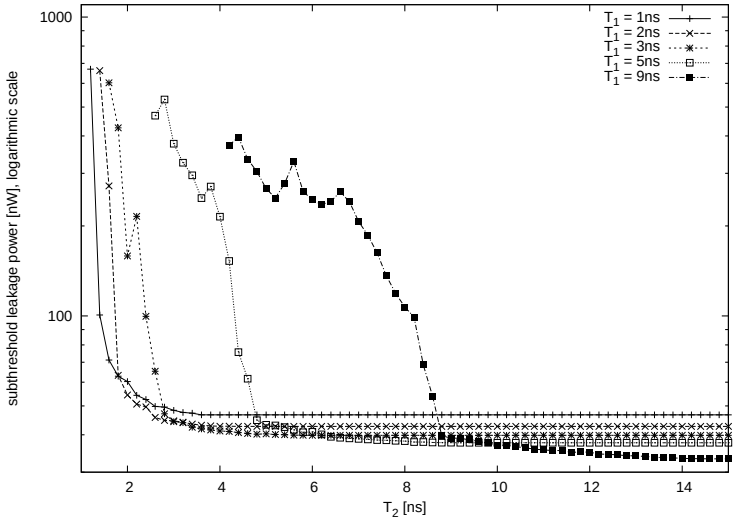

Figure 6.15: Leakage power with method 5, logarithmic scale

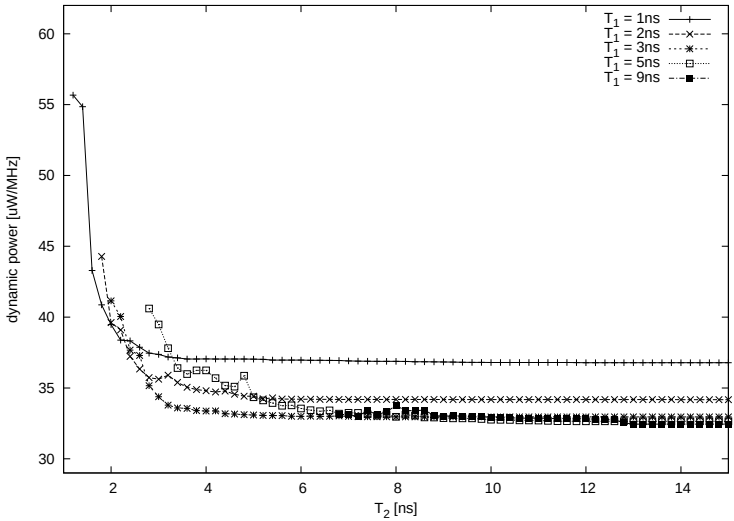

Figure 6.16: Dynamic power with method 4

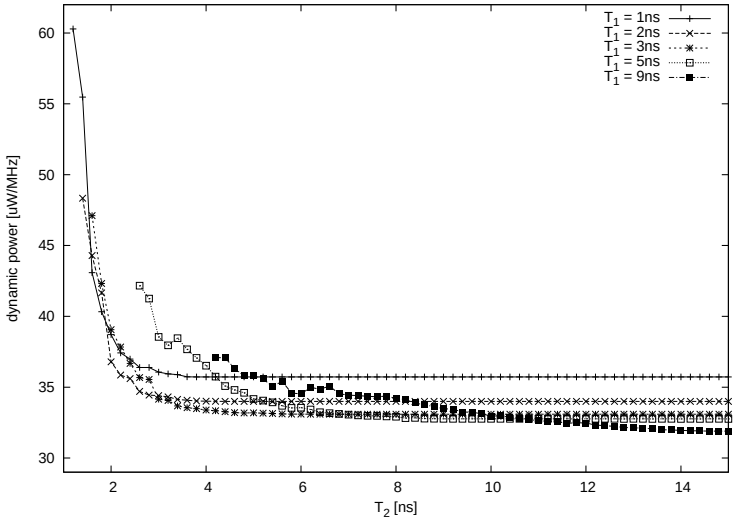

Figure 6.17: Dynamic power with method 5

clearly seen that the magnitude of the dynamic power depends on T_1. A design synthesized with a tight timing constraint, consists of many cells and are likely to have higher switching activity and interconnect capacitance. Therefore the methods 4 and 5 tends to have higher dynamic power, than if the base design was synthesized for the final timing constraint as in method 2 and 3.

Method	$P_{dyn}[\mu\mathrm{W/\,MHz}]$	$P_{leakage}[\mathrm{nW}]$
Method 1 (low $\mathrm{V_t}$)	56.5	1747
Method 1 (high $\mathrm{V_t}$)	–	–
Method 2	57.3	790 (45%)
Method 3	60.9	673 (39%)
Method 4 (1ns)	55.7	1025 (59%)
Method 5 (1ns)	60.3	669 (38%)
Method 6	56.4	1161 (66%)

Table 6.1: Synthesis of adder for maximum speed

Table 6.1 lists a summary of the improvements in leakage power when applying the methods at the highest operating speed of the adder. The high $\mathrm{V_t}$ to dual $\mathrm{V_t}$ methods 3 and 5 clearly increases the dynamic power, but in a technology where the leakage power is more significant, the reduction in leakage power may very well be more significant than the increase in dynamic power.

The most important observations from the synthesis and leakage power reduction of the adder are

- The leakage power of the adder design, using a dual $\mathrm{V_t}$ library, is always lower than for the all–low $\mathrm{V_t}$ design.
- The minimum area with high $\mathrm{V_t}$ cells, does not imply lowest leakage power.
- The initial structure of a circuit has great impact on the fraction of cells that can be substituted, given a timing constraint. Large cells prevents a fine grained gate-level threshold assignment, and can increase the leakage power compared to a design implemented with a high number of smaller gates.
- The dynamic power does not decrease significantly, but can increase depending on the leakage power optimization algorithm used.
- Performing the initial synthesis with a high $\mathrm{V_t}$ cell library and then incrementally substitute cells in the critical path to low $\mathrm{V_t}$, overall

results in the lowest leakage power. However this method also results in highest dynamic power.
- Synthesizing in one–pass with a dual V_t cell library (method 6) gives a disappointing result.

6.4 Synthesis of microprocessor

The ripple carry adder does not represent a typical digital design since it only performs a single arithmetic computation and doesn't have any parallelism. To evaluate what can be done with respect to leakage power on a more general design, a small microprocessor is synthesized. The MINI-MIPS processor with specifications listed in [32], is a 32-bit RISC 5-stage pipelined microprocessor, with an instruction–set being a subset of the instruction–set of the MIPS1 processor by MIPS TECHNOLOGIES. The implementation consists of three major parts:

Data path 32-bit ALU capable of performing eight operations, a barrel shifter and logic for computing the address of the next instruction.

Control logic Instruction decoding are performed by a central control unit. The same control unit is responsible for maintaining correct data flow throughout the pipeline.

Storage 32 flip-flop based registers and associated multiplexing network for storing and accessing operands and results in the register file. The instruction– and datamemory are not considered in the synthesis and power analysis.

Due to the nature of a pipelined design, it is the pipeline stage with the longest delay that limits the maximum speed of the entire design. In this processor, the ALU and instruction address computation logic are the primary contributors, to the delay of the critical path. The logic in the remaining pipeline stages are not time critical, and are expected to allow for cell substitution.

Where the logic synthesis of the adder was performed with an *ungrouped* hierarchy to allow for better optimizations, this was not possible with the processor. The processor is instead synthesized without *ungrouping*, thereby leading to a structure with slightly higher delay. In appendix E.2 is elaborated on the suspicious results generated Design Compiler during the synthesis with *ungrouping*.

6.4.1 Optimization results

Synthesis of the microprocessor overall shows the same trends as for the adder, however due to the size and structure of the design, there are a greater potential for reducing leakage power. Figure 6.18 shows the leakage power and area after using method 1. The minimum delay of the design using low V_t cells is $t_{min} = 4.6$ns, whereas the high V_t design has a minimum delay of 5.6ns. Thus for a high speed design, the all–high V_t implementation cannot satisfy the timing constraint.

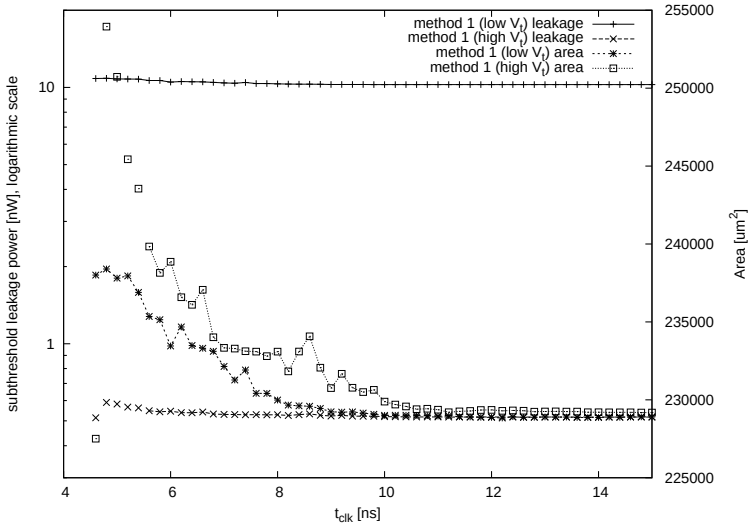

Figure 6.18: Leakage power with area constraint, logarithmic scale

The leakage power after application of the dual V_t methods 2, 3 and 6 are shown in figure 6.19. In contrast to the adder design, the leakage power of all the methods are absolutely decreasing for various timing constraints. This is caused by the structure of the design. Even though some big cells cannot be changed to high V_t, because they are part of the critical path, there are many other cells, which can be changed to high V_t, and thereby result in a total decrease of leakage power.

The one–pass method 6 never reaches an all–high V_t implementation, as seen from figure 6.20, This is because buffers of a large fanout net, for an

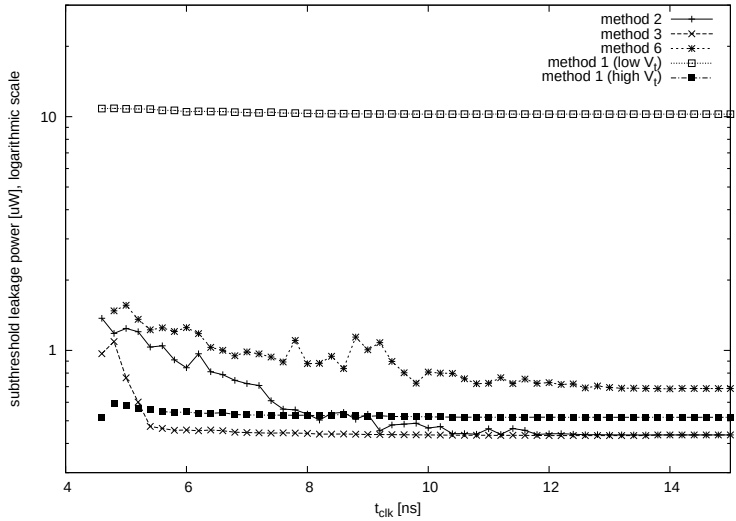

Figure 6.19: Leakage power with variable timing constraint (logarithmic scale)

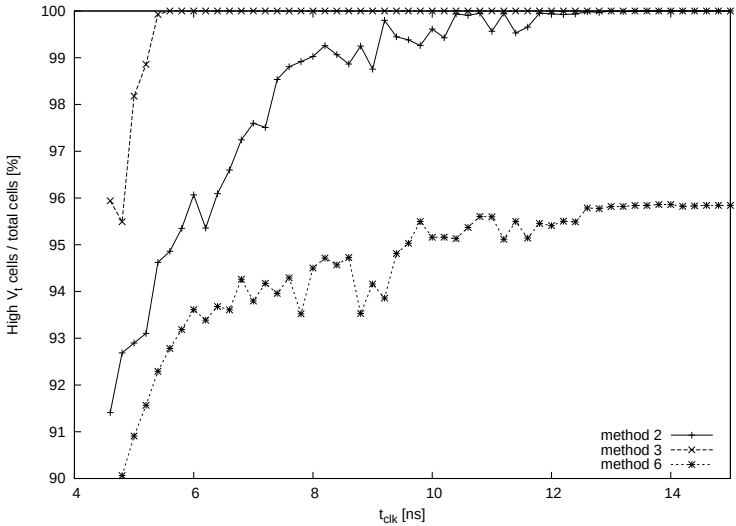

Figure 6.20: Percent high V_t cells with variable timing constraint

unknown reason, are kept low V_t. At a non-critical timing constraint, there should be no problem in replacing the low V_t buffers with high V_t buffers. Even if there is a design rule constraint, preventing a single high V_t buffer to drive a fanout of a certain size, the buffers could easily be duplicated, as illustrated in figure 6.21.

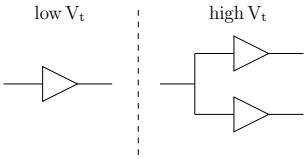

Figure 6.21: Low V_t and high V_t buffer

The area of the design is plotted in figure 6.22. Apparently the dual V_t method 6 has significantly higher area usage than *any* of the other methods. From

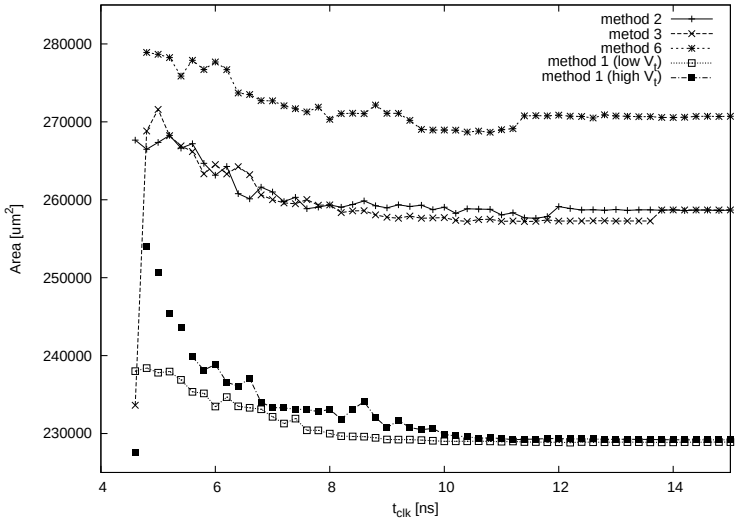

Figure 6.22: Resulting area with variable timing constraint

the number of cells in the design, figure 6.23, and the distribution of cells shown in figures F.1–F.5, it is clear that the dual V_t method introduces huge numbers of single–input cells compared to the other methods. These

cells are not contributing to the minimization of the cost metric: *leakage power*, instead they increase both power and delay. In appendix E.1.2 are elaborated on this problem.

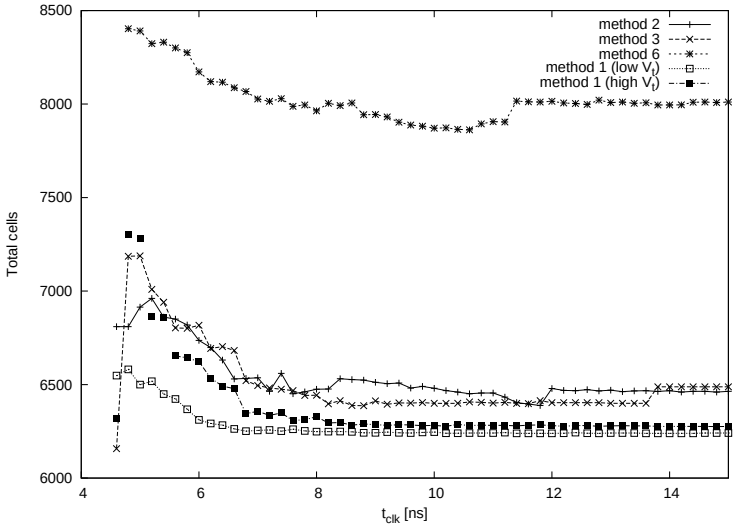

Figure 6.23: Number of cells with variable timing constraint

For $t_{\text{clk}} = 4.6$ns the area of the high V_t to dual V_t method makes an unrealistic jump downwards. This is tightly connected to a similar jump of the area of the *base design*. At values of $t_{\text{clk}} < 5.6$ns, the high V_t base design is in reality heavily over–constrained, thus the synthesis tool is requested to create a solution which is unrealistic. It is acceptable that the area increases rapidly when the synthesis tool tries to satisfy an unrealistic timing constraint, but the sudden jump downwards resulting in an area less than the area of the *area*-constrained design, cannot be right. The usefulness of the high V_t to dual V_t method 3 at $t_{\text{clk}} = 4.6$ns is therefore questionable.

Figure 6.24 and 6.25 shows the leakage power of method 4 and 5 respectively. The result of both methods shows that the initial timing constraint, T_1, doesn't affect the leakage power significantly as long as $T_2 > T_1$. The resulting leakage power of method 4 shown in figure 6.24, shows that the design initially synthesized with $T_1 = 8ns$, incrementally can be speeded up to satisfy a $4.6ns$ timing constraint without a significant increase in

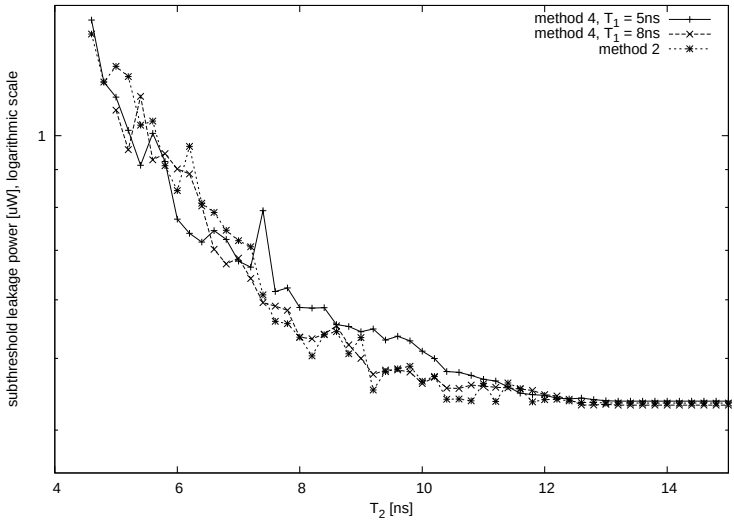

Figure 6.24: Leakage power with method 4, logarithmic scale

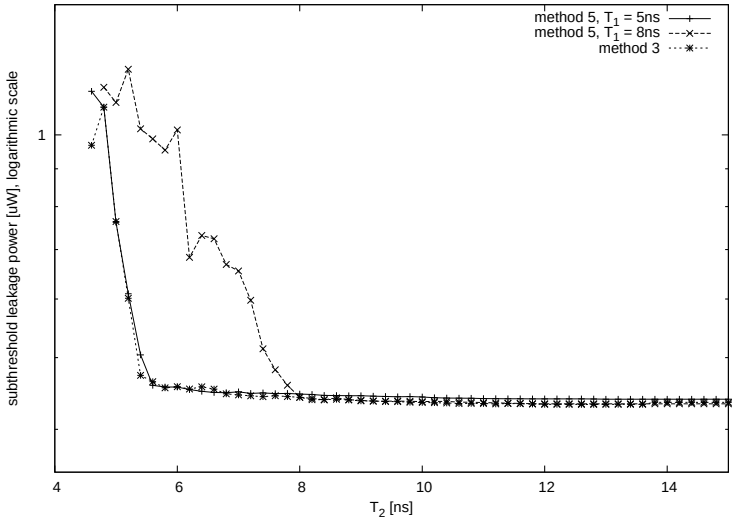

Figure 6.25: Leakage power with method 5, logarithmic scale

leakage power. This can be done by choosing an cell with an appropriate drive strength while also assigning a threshold voltage. However, such a speedup is rarely possible and should be considered as a coincidence.

The dynamic power of the area constrained methods shown in relation to the dual V_t leakage power reduction methods, are shown in figure 6.26. The dual V_t methods all have less dynamic power dissipation than the area constrained methods, however considering the unit, the difference is less than 1%.

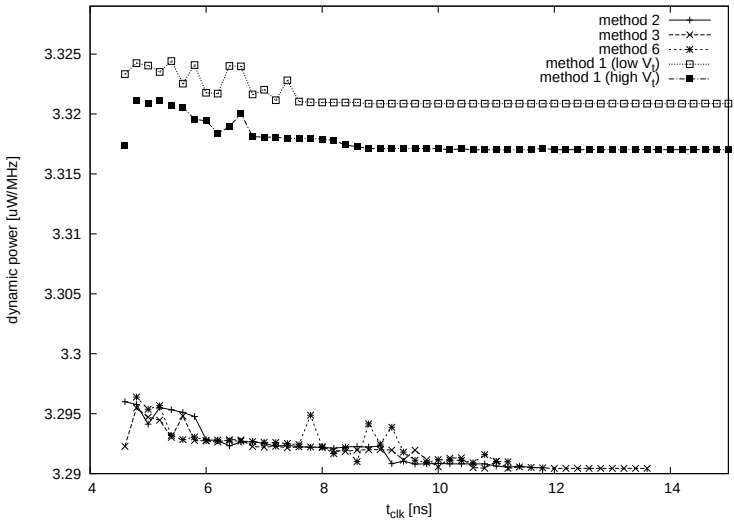

Figure 6.26: Dynamic power of the area constrained methods

When Synopsys computes dynamic power dissipation, the sources of dynamic power are:

$$P_{dynamic} = P_{dynamic,\ internal} + P_{dynamic,\ interconnect}$$

For the microprocessor design, the dynamic power dissipated internally in the cells (charging of parasitic capacitances and short circuit power) are by inspection of log files found to be in the range 3..5%, and it is this *insignificant* power contribution that are affected by the leakage power reduction methods.

In figure 6.27 are shown the dynamic power[2] obtained with method 2, 3 and 6. The dynamic power of the dual V_t method is a bit unpredictable, but in general all three methods follows the same pattern and only differs slightly. The dynamic power obtained with method 4 and 5 are plotted in figure 6.28 and 6.29. Even though the numbers of inverters and thereby also interconnect changes (see figure E.3 and E.4), the dynamic power behave approximately like the other methods. The increase in dynamic power for $T_1 = 8ns$, are due to the cells having increased drive strength, and thus consumes more energy when switching.

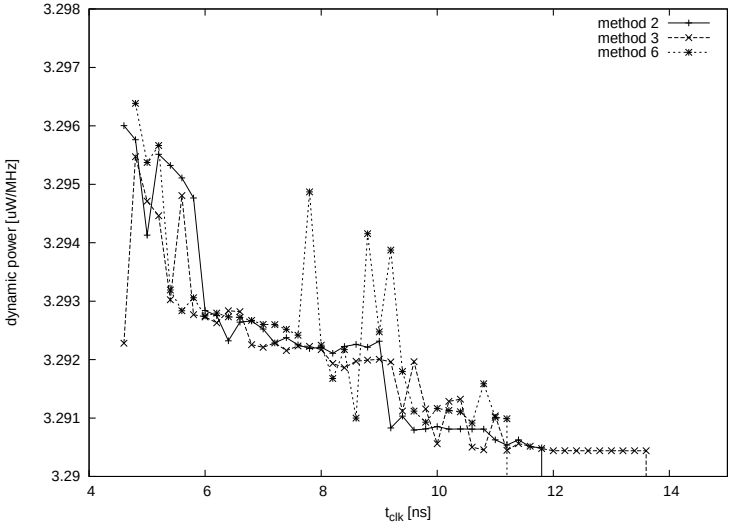

Figure 6.27: Dynamic power with variable timing constraint

[2]The calculation of dynamic power was not successful for all values of t_{clk}, this is shown in figure F.6

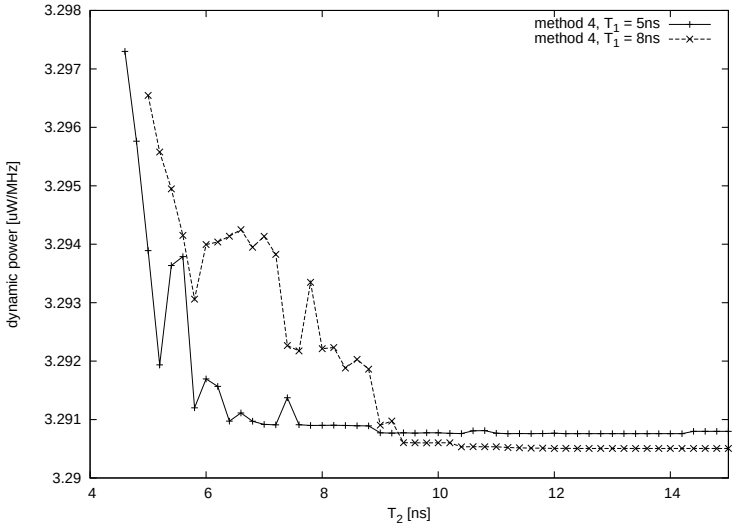

Figure 6.28: Dynamic power with method 4

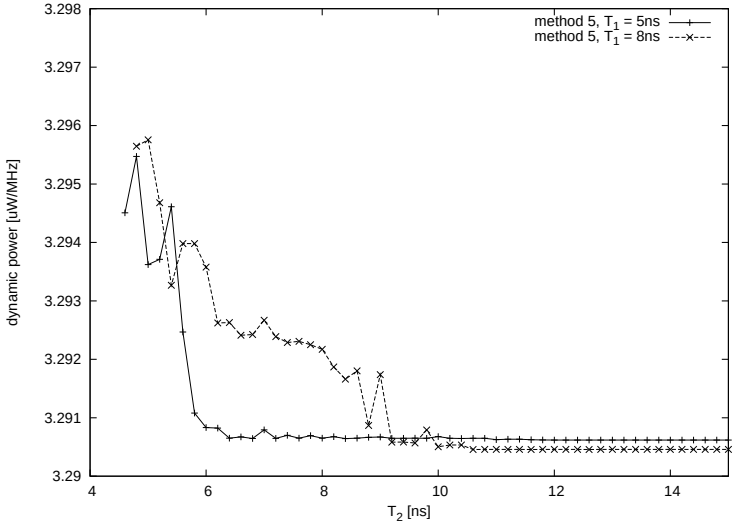

Figure 6.29: Dynamic power with method 5

Table 6.2 lists a summary of the improvements in leakage power when using the methods at the highest speed of the processor. The validity of method 3 is doubtful, and method 6 was unable to satisfy the $t_{\min} = 4.6$ns timing constraint.

Method	P_{dyn}[mW/MHz]	$P_{leakage}$[μW]
Method 1 (low V_t)	3.323	10.83
Method 1 (high V_t)	–	–
Method 2	3.296	1.372 (13%)
Method 3	*3.292*	*0.968 (9%)*
Method 4 (5ns)	3.297	1.433 (13%)
Method 5 (5ns)	3.294	1.146 (11%)
Method 6	–	–

Table 6.2: Synthesis of processor for maximum speed

As expected, any method that mixes the usage of low V_t and high V_t cells reduces the overall leakage power. The dynamic power is decreasing *slightly* when both low V_t and high V_t cells are used, but as the power dissipated in the cells are insignificant compared to the power dissipated in charging the interconnect capacitance, this reduction in dynamic power can be neglected.

The most important observations from the synthesis and leakage power reduction of the microprocessor are

- The leakage power of the dual V_t microprocessor design is significantly less than the all-low V_t design.
- The minimum area with high V_t cells does not imply lowest leakage power.
- In contrast to the adder, the microprocessor, which have several paths that are non-critical, are subject to huge leakage power savings without lowering the clock frequency.
- The dynamic power decreases slightly because of less power dissipated in form of short circuit power.
- Performing the initial synthesis with a high V_t cell library and then incrementally substitute cells in the critical path to low V_t, overall results in the least leakage power.
- Synthesizing in one-pass with a dual V_t cell library (method 6) results in the largest area and higher leakage power than the other leakage reduction methods. Furthermore the method was unable to satisfy the same timing constraint as the all–low V_t implementation.

6.5 Evaluation of results

The synthesis of both designs has shown, that a dual V_t cell library in conjunction with leakage power reducing methods, are very efficient. As there are no restrictions to the physical layout during logic synthesis, several options are available for reducing leakage power. In the two example designs, the leakage power were reduced by a combination of

- Selection of drive strength
- Assignment of threshold voltage
- Restructuring of the logic

The threshold voltage assignment and the drive strength selection are "simple" optimizations that structurally doesn't modify the netlist, thus they can be applied at post-layout too.

6.5.1 Drive strength

While the threshold voltage is controlled during fabrication of the device, the drive strength, eg. the size of the transistors in a logic gate, are more complicated to adjust post-layout. In [12] is developed a method for decreasing the size of transistors post-layout, without requiring re-placement and routing of a cell based design. While the sizing–method was developed with the purpose of reducing the dynamic power of a design, it is directly applicable for reducing leakage power. Thus the incremental methods used during logic synthesis, and the methods than can be applied post-layout, shares the same optimization–space for leakage power reductions, with the assumption that the drive strength of cells either is unchanged or decreased:

$$\text{Optimization–space} :: \langle \text{threshold voltage} \times \text{transistor sizing} \rangle$$

There are two exceptions to this, one is the change in the number of single–input cells as described in appendix E.1.1, the other is that the drive strengths of cells are increased, as a design is requested to operate at a higher speed than initially intended (method 2 and 3).

The decision of what drive strength to use for a cell, is done based on the cost function (appendix E.1.3) of Design Compiler. For the incremental method 3, where the design initially is synthesized with high V_t cells, the drive strength of time critical cells ($cells_{tc}$), will be at maximum. When

low V_t cells are introduced, such that the timing constraint can be satisfied, the drive strength of the $cells_{tc}$, will at most be *decreased*. Thus this method is comparable to what can be done post–layout.

The incremental method 2, where the leakage power of a low V_t design is reduced by introducing high V_t cells, are likely to be utilizing that the delay of a low V_t cell, in some cases can be matched by choosing a high V_t cell with higher drive strength, as discussed in chapter 5.2.6. As this approach increases the area of the cells to reduce leakage power, it will perform better than when applied post–layout, where increase in area not is possible. It should be noted, that starting with a design synthesized with a low V_t cell library, and then post–layout incrementally substitute cells, is the prevalent approach for leakage power reduction in the literature.

6.5.2 Area constrained leakage power reduction

Common to the results obtained with both the adder and microprocessor design, are the indication that minimum area doesn't imply minimum leakage power. There are three explanations for this

- The leakage power is state dependent. Thus if the least area implementation results in a circuit that often will be in a high leakage state, then the average leakage power may be higher, than for a design synthesized with another method.
- The physical dimensions of the cells in the cell library, all have same height but variable width. The widths are increased by a discrete number and are thus not continuous. This results in some cells having same area and logic function, but different drive strengths. When *area* is used as criteria for choosing cells, it is problematic when several cells have same area and logic function, but differs in terms of leakage power and delay.
- The multi–input cells, that efficiently implements a complex function in little area, are not available in as many drive strengths as the more commonly used logic gates such as NAND and NOR. To maximize the use of the multi–input cells, the cells with least drive strength, will use relatively larger transistors than the faster few–input cells, such that the delay of the multi–input cells are reduced. By using wider transistors the leakage power is increased, thus the leakage power for a big multi–input cell may not be minimum compared to the same function implemented with smaller cells.

6.5.3 Threshold voltage

Reducing leakage power with Design Compiler by synthesizing a design with both a low V_t and high V_t cell library (method 6), gives a disappointing result for both designs. It was expected that the presence of a dual V_t cell library would enable the synthesis algorithms to obtain the best result of all the proposed methods. As the logic optimization, technology mapping and assignment of threshold voltages all are performed in one tightly connected process without any irreversible steps, it is very strange that the method didn't perform better. From the synthesis of the adder, the dual V_t method is seen to chose a set of cells, that cannot be assigned high V_t. A possible explanation of this, is the large number of cells used during logic synthesis. Since the optimization is performed with a heuristic, the large number of cells *might* cause the optimization algorithm, to get stuck in a local minima.

The synthesis of the adder showed a very important tradeoff regarding gate-level assignment of threshold voltage:

> *To get maximum leakage power reduction either chose many few–input cells that can be assigned high V_t, or fewer but bigger cells, that are slower and thus cannot be assigned high V_t.*

The ratio of high V_t cells to low V_t cells were seen to be very dependent on the structure of the design, the timing constraint and the synthesis method used for reducing leakage power. Common to both designs, were a very significant decrease in leakage power, when the design initially was synthesized with a high V_t cell library and afterwards incrementally speeded up, by substituting some cells with the low V_t equivalent. This implies that by using small faster cells, the threshold voltage assignment becomes fine–grained, and thus enables better leakage power savings, for only a small increase in delay. However, the use of small cells increases the interconnect, and thus leads to a increase in dynamic power. Whether the total power dissipation for a design will decrease by initially synthesize with a high V_t cell library, highly depends on the activity of application.

Optimal threshold voltage

Assuming that each of the cells of the cell library are available in two versions, a high V_t and a low V_t, then how should these threshold voltages be selected, to give the best leakage power savings?

The value of the low threshold voltage is dictated primary by a delay requirement of the circuit. Thus if a pre-characterized cell library is bought from a vender, then the designer of the cell library has, by her opinion of what is most important for *all* designs, balanced parameters such as delay and power consumption.

The value of the high threshold voltage is chosen based on the power consumption of the cells. If the threshold voltage is high, then the power savings will be best, but the cells have increased delay. In figure 6.30 are sketched how the balance of delay and power affects the final leakage power reduction. For a very strict timing constraint, with many paths near timing critical, it will be beneficial to have a threshold voltage close to low V_t.

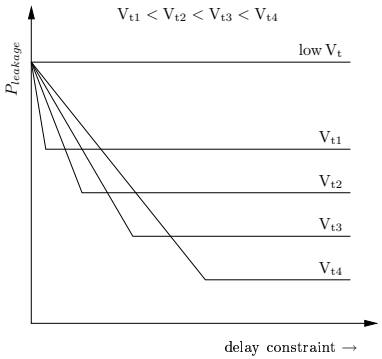

Figure 6.30: High threshold voltage tradeoff

The problem with the high threshold voltage, is that the vendor of the high V_t cell library, has chosen a threshold voltage that is believed to give good leakage power savings, for what the vendor thinks is an average design. Whether or not the actual design shares the same structure as the average design of the vendor, is question that are unlikely to be answered.

The selection of the best high threshold voltage for a given design, is not a simple task, since the V_t highly affects the leakage power reduction algorithms. As the behavior of the algorithms not easily can be predicted on beforehand, the only way to find the optimal value of high V_t, is to experimentally optimize the design with a large set of different high V_t libraries and find the best value of the high threshold voltage. Such an approach is very impractical and is very unlikely to be performed for a big design. An

alternative is to have an infinite numbers of threshold voltages, but this is not feasible because of:

Runtime of algorithm that assigns threshold voltage has to explore the tradeoff of what threshold value to chose, instead of the simple binary low V_t or high V_t selection.

Production costs would grow significantly due to the "infinite" number of extra processing steps needed.

Process variations. The actual threshold voltage follows a distribution with spread defined by the process, thus it is more reasonable to have a discrete number of threshold voltages that with high confidence can be guaranteed to be within an upper/lower bound.

With these limitations in mind, the best alternative is that the vendor instead of just a low V_t and high V_t cell library, also supplies a very high V_t and possibly a medium V_t cell library as well. The leakage reduction algorithms are then expected to result in even higher leakage power savings, with only a small increase in runtime.

Chapter 7

Discussion and Conclusion

7.1 Discussion

This thesis has addressed the problem of increasing leakage power by analyzing the possibilities of using a dual V_t cell library in the logic synthesis. Especially the consequences of using a dual V_t cell library has been analyzed, and the theory behind the threshold voltage assignment has been considered.

The design space of using several instances of logic gates with different threshold voltages, was seen to allow for a general reduction in leakage power. By raising the threshold voltage of a logic gate, the leakage power decreases at the cost of an increase in delay. However, as a side effect of the increased transition times of the logic gate, the short circuit power, P_{sc}, of other logic gates *may* increase.

While the P_{sc} increases the total dynamic power dissipation, it is of little significance compared to the power contribution from the charging and discharging of interconnect capacitances. As the number of high V_t cells in the design is increased, the total P_{sc} will be absolutely decreasing. Thus while a *local* increase in P_{sc} is possible, it will tend to be *hidden* as both the number of cells in the design and the fraction of high V_t cells increases.

Even though large logic gates are beneficial from a leakage power point of view, their use in a cell based design flow can be limited, when the delay

of the design is of importance. The cell library used in the simulations, is focused on providing the commonly used 2–4 input cells with a variety of characteristics. Thus the larger cells are only available with characteristics that maximizes their use when delay is of importance. This in general is a reasonably decision, since the maintenance and development of a big cell library is a very time consuming task. To enable the use of the large cells to reduce leakage power, a cell based design is not optimal – it is an enormous task to pre-characterize all possible cells with N inputs. An alternative approach is to *merge* smaller cells into larger ones during placement, as done by the tool ZenTime [3].

The incorporation of multiple threshold voltages into the technology mapping phase, and the development of a delay–cost model that considers the activity of the design, was done in chapter 5. By a simple example consisting of an OR–function, it was shown that by re-mapping the graph of the function, the leakage power can be reduced at the cost of a slight increase in area and delay. Such a restructuring is only possible when there are no restrictions on the preservation of the layout. Furthermore by letting the cost–function specify the drive strength of a cell, the cell can either be *speeded up* to allow for a high V_t assignment, or *slowed down* such that the leakage can be further reduced.

The application of an *idle mode pattern* during logic synthesis was discussed, however for large designs, the feasibility of finding a minimum idle pattern is questionable. Furthermore as the clock–enable signal lacks the knowledge of duration, the method can not be guaranteed to reduce total power dissipation.

A number of syntheses were performed with Synopsys Design Compiler. The results shows that *any* of the synthesis methods that uses a dual V_t cell library, reduces the leakage power. The absolute reduction in leakage power was greatest when the design initially was synthesized with high V_t cells, at the expense of an increase in area and dynamic power. Most importantly it was shown, that the initial selection of cells done during synthesis, not necessarily are suitable for an incremental leakage power reduction.

Finally the leakage reduction capabilities of the latest design tools were evaluated, and the benefit of placement–aware synthesis was discussed in relation to dual V_t methods.

7.2 Future work

Leakage power reduction performed early in the design flow has been shown to provide promising results. However the impact on the design phases following logic synthesis, has not been analyzed. A natural continuation of this project is to look at the subsequent phases of the design flow.

A topic of utmost concern, which due to technical problems not was possible to do in this project, is to compare the leakage power reduction obtained at logic synthesis with what can be done post–layout.

If the delay of many paths in a design are increased by using slow high V_t cells instead of the faster low V_t cells, thus the fraction of time critical paths are increased, then the problems of obtaining timing closure after placement and routing, may become worsened. Today it is not uncommon that several place and route iterations are needed to satisfy the timing requirement, but if the number of problematic paths are increased, then even more iterations might be needed. A different perspective of having many time critical paths is, that the place and route tool is encouraged to increase the effort, such that it is possible to use the slow, but less leaking high V_t cells.

Besides placing cells to enable the use of as many as possible high V_t cells, the temperature variations on the die could be considered. As the sub-threshold leakage power is very temperature dependent, there is a tradeoff in how the high-activity circuitry are distributed on the die. Should high activity circuitry be centralized, such that it will have a relatively high leakage power, while the low activity circuitry can be at a lower temperature. Or is it better to distribute the high activity circuitry evenly, such that an even temperature can be assumed, and all cells thus will have only slightly increased leakage power.

As the dual V_t methods uses slack to enable a cell to be changed to high V_t, *retiming* may enable more cells to be modified. While retiming normally is used for reducing delay of the critical path by re-positioning registers, it may be able to re-distribute the delays of non-critical paths such that more high V_t cells can be used.

The clock tree is a huge source of power dissipation in synchronous designs. The use of high V_t and low V_t cells in the clock tree can be used for not only reducing leakage power, but also give rise to a small reduction in dynamic power. However, the clock–signal at synchronous elements must

all arrive within a small time window, to guarantee the proper operation of the circuit, thus careful analysis is needed.

The primary focus in the simulations performed with Design Compiler, was to reduce leakage power and observe the impact on dynamic power. In a real design flow the individual components of the total power dissipation are of less importance, thus a cell library with a large variety of cells and comparable leakage and dynamic power is needed, such that the simulations can be performed aimed towards minimizing *total* power dissipation.

7.3 Conclusion

Reduction of leakage power with the use of a dual V_t cell library has been shown to be a very effective power reduction method. By substituting low V_t cells outside the critical path with functionally equivalent high V_t cells, the leakage power can be reduced significantly, *without making impact on the performance of the design.*

The leakage reduction potential of a design, is very dependent on the initial structure of the design. If cells are to be substituted on a design with a fixed structure, for example at post–layout, the initial selection and organization of cells, are a limiting factor for how much the leakage power can be reduced.

It was expected that the use of a dual V_t cell library would allow the logic synthesis algorithms to gain optimal leakage power savings, however by using Synopsys Design Compiler, this method produced the least leakage power savings.

The prevalent approach of leakage reduction by initially synthesizing a design with low V_t cells, and afterwards at post–layout introduce high V_t cells, will not give minimum leakage power, but the dynamic power are likely to be non-increasing. To obtain the best leakage power reduction, the design should initially be synthesized with small few–input cells, thus allowing a fine–grained assignment of threshold voltages, but at the cost of increased area and an increase in dynamic power dissipation.

Dynamic power is still the dominating source of total power dissipation, thus leakage power alone is not a sufficient power metric, and the best power reduction approach for a design, has to be determined by its activity. Even though the post–layout leakage reduction methods might not give

optimal leakage power savings, they have the benefit of being aware of actual capacitances of the final design. With the increasing problems of early estimation of interconnect capacitances of the final design, the post–layout methods will be able to push the design to its limits, and possibly outperform similar leakage power reduction methods applied during logic synthesis.

Appendix A

Project description

NR.:	63
Title:	Incorporating leakage current considerations in logic synthesis
Student:	Michael Kristensen
Period:	17.02.2004 – 13.08.2004

Project description

Objectives

The objective of this MSc thesis work is to investigate optimal design under the presence of static gate leakages, and to device how design rules and trade-offs are altered.

Description

A main concern during the design of System-on Chips (SOCs) is the power budget, especially battery supplied systems are considered. In general, dynamic and static contributions constitute total power dissipation. Dynamic power is primarily consumed by the information processing in the charging and discharging of internal capacitances. As such, dynamic power consumption is proportional to these capacitances, the switching frequency and the supply voltage. Static power consumption, on the other hand, is caused by leakage currents while the circuit is idle, i.e. not performing computations.

One key attraction of CMOS is negligible static power consumption. However, with decreasing device sizes this property is no longer satisfied due to subtreshold conduction. The reason for this is that for smaller devices, supply voltages are reduced. For speed, this in turn forces a reduction in threshold voltages. As a consequence, transistors are no longer turned off satisfactorily, i.e. drain currents contributes significantly to power losses in the transistor non-conductive state. For a 0.13ţm process, the static losses may constitute almost 50% of the total power consumption.

The issue has been addressed by offering libraries of gates and cells in both low V_t and high V_t versions. This offers the option of fast, low V_t cells with high static power losses where timing is critical, and a slower, high V_t design for other parts. Traditional synthesis tools do not offer the means to optimize for multi V_t libraries to reduce static power consumption. The solution, using such known synthesis tools, consists of synthesizing a design using a low V_t library, under the constraint that timing and performance requirements are met. Then, in a post-synthesis phase, the back-annotated circuit is analyzed with respect to power consumption and the circuit modified, replacing low V_t by high V_t library cells whereever possible. The update process does not involve any re-synthesis steps.

This thesis work addresses how to incorporate static power consumption as part of the synthesis process, utilizing the detailed output of a power simulation in the synthesis. A method for multi V_t synthesis is derived, and the capability of the method is compared to the capabilities of the Synopsys design tool.

The thesis work will be performed in parallel with two other MSc thesis works in a collaborative but independent effort. One work focusses on the design of libraries offering various speed to power alternatives, while the other work concentrates on the architectural aspects of multi V_t libraries.

Supervisor: Flemming Stassen

Appendix B

CMOS process

B.1 70nm PTM process

The *high speed* parameters in table B.1 are the by UC Berkeley Device Group suggested values for the predictive technology model [39]. The threshold voltages of the *low leakage* process have been arbitrary chosen, since their use only are for simple examples involving a high and low threshold voltage. The model cards are based upon the work done in [11].

	L_{eff}	T_{ox}	V_t^{sat}	R_{dsw}
NMOS (high speed)	0.038 μm	16 Å	0.15V	150 $\frac{ohm}{sq}$
NMOS (low leakage)	0.038 μm	16 Å	0.30V	150 $\frac{ohm}{sq}$
PMOS (high speed)	0.04 μm	17 Å	-0.16V	280 $\frac{ohm}{sq}$
PMOS (low leakage)	0.04 μm	17 Å	-0.30V	280 $\frac{ohm}{sq}$

Table B.1: 70 nm proces parameters

B.1.1 NMOS and PMOS model card (high speed)

```
*
* Predictive Technology Model Beta Version
* 0.07um NMOS SPICE Parameters (normal one)
*
```

```
* Vtn =  0.15
* Vtp = -0.15



.SUBCKT nmosths  Drain     Gate      Source  Bulk
+       M=1       W=10e-6  AD='(6e-07)*w' PD='2*(6e-07)+w' NRD=0
+                 L=10e-6  AS='(6e-07)*w' PS='2*(6e-07)+w' NRS=0
.param  WOTn='9.35338e-08-(4.19715e-15/l)-(1.50197e-14/w)'

M1       Drain Gate Source Bulk    BPTM70HSN_TYP
+        W=W L=L AS=AS AD=AD PS=PS PD=PD M=M NRD=NRD NRS=NRS

.model BPTM70HSN_TYP NMOS

+Level = 49

+Lint = 1.6e-08 Tox = 1.6e-09
+Vth0 = 0.1902 Rdsw = 150

+lmin=7.0e-8 lmax=7.0e-8 wmin=0.7e-7 wmax=1.0e-4
+Tref=27.0    version =3.1  Xj= 2.9999999E-08 Nch= 1.2000000E+18
+lln= 1.0000000  lwn= 1.0000000   wln= 0.00  wwn= 0.00
+ll= 0.00     lw= 0.00 lwl= 0.00 wint= 0.00 wl= 0.00
+ww= 0.00 wwl= 0.00 Mobmod=1 binunit= 2 xl= 0.00 xw= 0.00
+Lmlt= 1 Wmlt= 1 binflag=  0 Dwg= 0.00 Dwb= 0.00

+ACM= 0 ldif=0.00 hdif=0.00 rsh= 6 rd= 0 rs= 0 rsc= 0 rdc= 0

+K1= 0.3700000 K2= 1.0000000E-02 K3= 0.00
+Dvt0= 1.3000000 Dvt1= 0.5000000 Dvt2= 2.9999999E-02 Dvt0w= 0.00
+Dvt1w= 0.00 Dvt2w= 0.00 Nlx= 7.0000000E-08 W0= 0.00
+K3b= 0.00 Ngate= 5.0000000E+20

+Vsat= 1.1500000E+05 Ua= 5.0000000E-10 Ub= 1.0000000E-18 Uc↩
     = -2.9999999E-11
+Prwb= 0.00 Prwg= 0.00 Wr= 1.0000000 U0= 2.5000000E-02 A0↩
     = 1.5000000
+Keta= 4.0000000E-02 A1= 0.00 A2= 1.0000000 Ags= -1.0000000E-02
+B0= 0.00 B1= 0.00

+Voff= -0.1500000 NFactor= 1.5000000 Cit= 0.00 Cdsc= 0.00 Cdscb↩
     = 0.00
+Cdscd= 1.0000000E-14 Eta0= 0.2000000 Etab= 0.00 Dsub= 1.0000000

+Pclm= 0.2500000 Pdiblc1= 1.2000000E-02 Pdiblc2= 7.5000000E-03
+Pdiblcb= -1.3500000E-02 Drout= 1.5000000 Pscbe1= 8.6600000E+08
+Pscbe2= 1.0000000E-20 Pvag= -0.2800000 Delta= 1.0000000E-02
+Alpha0= 0.00 Beta0= 30.0000000

+kt1= -0.3700000 kt2= -4.0000000E-02 At= 5.5000000E+04
+Ute= -1.4800000 Ua1= 9.5829000E-10 Ub1= -3.3473000E-19
+Uc1= 0.00 Kt1l= 4.0000000E-09 Prt= 0.00

+Cj= 0.0015 Mj= 0.72 Pb= 1.25 Cjsw= 2E-10 Mjsw= 0.37
+Php= 0.77 Cjgate= 2E-14 Cta= 0 Ctp= 0 Pta= 0 Ptp= 0
+JS=1.50E-08 JSW=2.50E-13 N=1.0 Xti=3.0
```

```
+Cgdo=4.094E-10 Cgso=4.094E-10 Cgbo=0.0E+00 Capmod= 2
+NQSMOD= 0 Elm= 5 Xpart= 1 cgsl= 1E-10 cgdl= 1E-10
+ckappa= 0.08 cf= 1.266e-10 clc= 1.0000000E-07 cle= 0.6000000
+Dlc= 1.6E-08 Dwc= 0

.ENDS



.SUBCKT pmosths   Drain     Gate      Source   Bulk
+        M=1      W=10e-6   AD='(6e-07)*w' PD='2*(6e-07)+w' NRD=0
+                  L=10e-6   AS='(6e-07)*w' PS='2*(6e-07)+w' NRS=0

M1       Drain Gate Source Bulk    BPTM70HSP_TYP
+        W=W L=L AS=AS AD=AD PS=PS PD=PD M=M NRD=NRD NRS=NRS

.model BPTM70HSP_TYP PMOS

+Level = 49

+Lint = 1.5e-08 Tox = 1.7e-09
+Vth0 = -0.203 Rdsw = 280

+lmin=7.0e-8 lmax=7.0e-8 wmin=0.7e-7 wmax=1.0e-4
+Tref=27.0 version =3.1 Xj= 2.9999999E-08 Nch= 1.2000000E+18
+lln= 1.0000000 lwn= 0.00 wln= 0.00 wwn= 1.0000000
+ll= 0.00 lw= 0.00 lwl= 0.00 wint= 0.00 wl= 0.00 ww= 0.00
+wwl= 0.00 Mobmod= 1 binunit= 2 xl= 0.00 xw= 0.00
+Lmlt= 1 Wmlt= 1 binflag= 0 Dwg= 0.00 Dwb= 0.00

+ACM= 0 ldif=0.00 hdif=0.00 rsh= 7 rd= 0 rs= 0 rsc= 0 rdc= 0

+K1= 0.3800000 K2= 1.0000000E-02 K3= 0.00 Dvt0= 2.2000000
+Dvt1= 0.6500000 Dvt2= 5.0000000E-02 Dvt0w= 0.00 Dvt1w= 0.00
+Dvt2w= 0.00 Nlx= 8.0000000E-08 W0= 0.00 K3b= 0.00 Ngate↩
     = 5.0000000E+20

+Vsat= 8.5000000E+04 Ua= 1.8000000E-09 Ub= 3.0000000E-18
+Uc= -2.9999999E-11 Prwb= 0.00 Prwg= 0.00 Wr= 1.0000000
+U0= 1.4500000E-02 A0= 1.2000000 Keta= 4.0000000E-02
+A1= 0.00 A2= 0.9900000 Ags= -0.1000000 B0= 0.00 B1= 0.00

+Voff= -0.1500000 NFactor= 1.2000000 Cit= 0.00 Cdsc= 0.00
+Cdscb= 0.00 Cdscd= 0.00 Eta0= 0.2700000 Etab= 0.00 Dsub↩
     = 0.9500000

+Pclm= 0.5500000 Pdiblc1= 1.2000000E-02 Pdiblc2= 7.5000000E-03
+Pdiblcb= -1.3500000E-02 Drout= 0.9000000 Pscbe1= 8.6600000E+08
+Pscbe2= 1.0000000E-20 Pvag= -0.2800000 Delta= 1.0100000E-02
+Alpha0= 0.00 Beta0= 30.0000000

+kt1= -0.3400000 kt2= -5.2700000E-02 At= 0.00 Ute= -1.2300000
+Ua1= -8.6300000E-10 Ub1= 2.0000001E-18 Uc1= 0.00
+Kt1l= 4.0000000E-09 Prt= 0.00

+Cj= 0.0015 Mj= 0.72 Pb= 1.25 Cjsw= 2E-10 Mjsw= 0.37
+Php= 0.77 Cjgate= 2E-14 Cta= 0 Ctp= 0 Pta= 0 Ptp= 0
+JS=1.50E-08 JSW=2.50E-13 N=1.0 Xti=3.0
```

```
+Cgdo=3.853E-10 Cgso=3.853E-10 Cgbo=0.0E+00 Capmod= 2
+NQSMOD= 0 Elm= 5 Xpart= 1 cgsl= 0.6422E-10 cgdl= 0.6422E-10
+ckappa= 0.08 cf= 1.266e-10 clc= 1.0000000E-07 cle= 0.6000000
+Dlc= 1.5E-08 Dwc= 0

.ENDS

.ENDL
```

B.1.2 NMOS and PMOS model card (low leakage)

```
*
* Predictive Technology Model Beta Version
* 0.07um NMOS SPICE Parameters (normal one)
*

* Vtn =  0.30V
* Vtp = -0.30V



.LIB     BPTM70LL_LIB

.SUBCKT  nmostll   Drain     Gate      Source    Bulk
+        M=1       W=10e-6   AD='(6e-07)*w' PD='2*(6e-07)+w' NRD=0
+                  L=10e-6   AS='(6e-07)*w' PS='2*(6e-07)+w' NRS=0
.param   WOTn='9.35338e-08-(4.19715e-15/l)-(1.50197e-14/w)'

M1       Drain Gate Source Bulk    BPTM70LLN_TYP
+        W=W L=L AS=AS AD=AD PS=PS PD=PD M=M NRD=NRD NRS=NRS

.model BPTM70LLN_TYP NMOS

+Level = 49

+Lint = 1.6e-08 Tox = 1.6e-09
+Vth0 = 0.3402 Rdsw = 150

+lmin=7.0e-8 lmax=7.0e-8 wmin=0.7e-7 wmax=1.0e-4
+Tref=27.0    version =3.1  Xj= 2.9999999E-08 Nch= 1.2000000E+18
+lln= 1.0000000  lwn= 1.0000000   wln= 0.00  wwn= 0.00
+ll= 0.00     lw= 0.00 lwl= 0.00 wint= 0.00 wl= 0.00
+ww= 0.00 wwl= 0.00 Mobmod=1 binunit= 2 xl= 0.00 xw= 0.00
+Lmlt= 1 Wmlt= 1 binflag=  0 Dwg= 0.00 Dwb= 0.00

+ACM= 0 ldif=0.00 hdif=0.00 rsh= 6 rd= 0 rs= 0 rsc= 0 rdc= 0

+K1= 0.3700000 K2= 1.0000000E-02 K3= 0.00
+Dvt0= 1.3000000 Dvt1= 0.5000000 Dvt2= 2.9999999E-02 Dvt0w= 0.00
+Dvt1w= 0.00 Dvt2w= 0.00 Nlx= 7.0000000E-08 W0= 0.00
+K3b= 0.00 Ngate= 5.0000000E+20

+Vsat= 1.1500000E+05 Ua= 5.0000000E-10 Ub= 1.0000000E-18 Uc↩
     = -2.9999999E-11
+Prwb= 0.00 Prwg= 0.00 Wr= 1.0000000 U0= 2.5000000E-02 A0↩
     = 1.5000000
+Keta= 4.0000000E-02 A1= 0.00 A2= 1.0000000 Ags= -1.0000000E-02
```

```
+B0= 0.00 B1= 0.00

+Voff= -0.1500000 NFactor= 1.5000000 Cit= 0.00 Cdsc= 0.00 Cdscb↩
      = 0.00
+Cdscd= 1.0000000E-14 Eta0= 0.2000000 Etab= 0.00 Dsub= 1.0000000

+Pclm= 0.2500000 Pdiblc1= 1.2000000E-02 Pdiblc2= 7.5000000E-03
+Pdiblcb= -1.3500000E-02 Drout= 1.5000000 Pscbe1= 8.6600000E+08
+Pscbe2= 1.0000000E-20 Pvag= -0.2800000 Delta= 1.0000000E-02
+Alpha0= 0.00 Beta0= 30.0000000

+kt1= -0.3700000 kt2= -4.0000000E-02 At= 5.5000000E+04
+Ute= -1.4800000 Ua1= 9.5829000E-10 Ub1= -3.3473000E-19
+Uc1= 0.00 Kt1l= 4.0000000E-09 Prt= 0.00

+Cj= 0.0015 Mj= 0.72 Pb= 1.25 Cjsw= 2E-10 Mjsw= 0.37
+Php= 0.77 Cjgate= 2E-14 Cta= 0 Ctp= 0 Pta= 0 Ptp= 0
+JS=1.50E-08 JSW=2.50E-13 N=1.0 Xti=3.0
+Cgdo=4.094E-10 Cgso=4.094E-10 Cgbo=0.0E+00 Capmod= 2
+NQSMOD= 0 Elm= 5 Xpart= 1 cgsl= 1E-10 cgdl= 1E-10
+ckappa= 0.08 cf= 1.266e-10 clc= 1.0000000E-07 cle= 0.6000000
+Dlc= 1.6E-08 Dwc= 0



.ENDS


.SUBCKT pmostll   Drain     Gate      Source    Bulk
+        M=1       W=10e-6  AD='(6e-07)*w' PD='2*(6e-07)+w' NRD=0
+                  L=10e-6  AS='(6e-07)*w' PS='2*(6e-07)+w' NRS=0

M1        Drain Gate Source Bulk    BPTM70LLP_TYP
+         W=W L=L AS=AS AD=AD PS=PS PD=PD M=M NRD=NRD NRS=NRS

.model BPTM70LLP_TYP PMOS

+Level = 49

+Lint = 1.5e-08 Tox = 1.7e-09
+Vth0 = -0.353 Rdsw = 280

+lmin=7.0e-8 lmax=7.0e-8 wmin=0.7e-7 wmax=1.0e-4
+Tref=27.0 version =3.1 Xj= 2.9999999E-08 Nch= 1.2000000E+18
+lln= 1.0000000 lwn= 0.00 wln= 0.00 wwn= 1.0000000
+ll= 0.00 lw= 0.00 lwl= 0.00 wint= 0.00 wl= 0.00 ww= 0.00
+wwl= 0.00 Mobmod= 1 binunit= 2 xl= 0.00 xw= 0.00
+Lmlt= 1 Wmlt= 1 binflag= 0 Dwg= 0.00 Dwb= 0.00

+ACM= 0 ldif=0.00 hdif=0.00 rsh= 7 rd= 0 rs= 0 rsc= 0 rdc= 0

+K1= 0.3800000 K2= 1.0000000E-02 K3= 0.00 Dvt0= 2.2000000
+Dvt1= 0.6500000 Dvt2= 5.0000000E-02 Dvt0w= 0.00 Dvt1w= 0.00
+Dvt2w= 0.00 Nlx= 8.0000000E-08 W0= 0.00 K3b= 0.00 Ngate↩
      = 5.0000000E+20

+Vsat= 8.5000000E+04 Ua= 1.8000000E-09 Ub= 3.0000000E-18
```

```
+Uc= -2.9999999E-11 Prwb= 0.00 Prwg= 0.00 Wr= 1.0000000
+U0= 1.4500000E-02 A0= 1.2000000 Keta= 4.0000000E-02
+A1= 0.00 A2= 0.9900000 Ags= -0.1000000 B0= 0.00 B1= 0.00

+Voff= -0.1500000 NFactor= 1.2000000 Cit= 0.00 Cdsc= 0.00
+Cdscb= 0.00 Cdscd= 0.00 Eta0= 0.2700000 Etab= 0.00 Dsub↩
     = 0.9500000

+Pclm= 0.5500000 Pdiblc1= 1.2000000E-02 Pdiblc2= 7.5000000E-03
+Pdiblcb= -1.3500000E-02 Drout= 0.9000000 Pscbe1= 8.6600000E+08
+Pscbe2= 1.0000000E-20 Pvag= -0.2800000 Delta= 1.0100000E-02
+Alpha0= 0.00 Beta0= 30.0000000

+kt1= -0.3400000 kt2= -5.2700000E-02 At= 0.00 Ute= -1.2300000
+Ua1= -8.6300000E-10 Ub1= 2.0000001E-18 Uc1= 0.00
+Kt1l= 4.0000000E-09 Prt= 0.00

+Cj= 0.0015 Mj= 0.72 Pb= 1.25 Cjsw= 2E-10 Mjsw= 0.37
+Php= 0.77 Cjgate= 2E-14 Cta= 0 Ctp= 0 Pta= 0 Ptp= 0
+JS=1.50E-08 JSW=2.50E-13 N=1.0 Xti=3.0
+Cgdo=3.853E-10 Cgso=3.853E-10 Cgbo=0.0E+00 Capmod= 2
+NQSMOD= 0 Elm= 5 Xpart= 1 cgsl= 0.6422E-10 cgdl= 0.6422E-10
+ckappa= 0.08 cf= 1.266e-10 clc= 1.0000000E-07 cle= 0.6000000
+Dlc= 1.5E-08 Dwc= 0

.ENDS

.ENDL
```

B.2 Design rules

The MOSIS Scalable CMOS (SCMOS) (Revision 8.0) design rules [36] for a *deep* submicron technology, are used for estimating the dimensions of a minimum sized transistor.

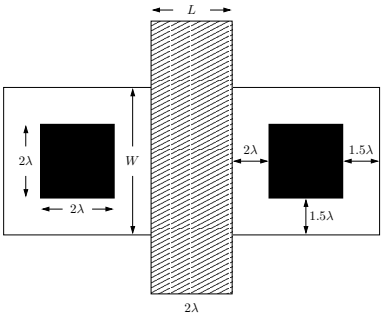

Figure B.1: Design rules for a transistor

For use in HSpice simulations, the minimum length and width of the transistor shown in figure B.1, are calculated to be

$$\begin{aligned} L_{min} &= 2\lambda = 70\text{nm} \\ W_{min} &= 5\lambda = 175\text{nm} \end{aligned}$$

B.3 Gate capacitance

The gate capacitance is calculated as in [26]:

$$C_{gate} = \frac{\epsilon_{SiO_2} \cdot \epsilon_0 \cdot A}{t_{ox}} = \frac{3.9 \cdot 8.854 \cdot 10^{-12} \cdot W \cdot L}{t_{ox}}$$

Using the parameters from table B.1 for the 70 nm process, the input capacitance of a minimum sized inverter is

$$\begin{aligned} C_{gate,inverter} &= C_{gate,pmos} + C_{gate,nmos} \\ &= 0.2488fF + 0.2644fF = 0.5132fF \end{aligned}$$

B.4 Interconnect capacitance

The interconnect capacitance per length unit is calculated with the model from [43] and the parameters from the Predictive Technology Model [39]. These parameters are listed in table B.2 and shown in figure B.2.

wire type	$width, w$	$thickness, t$	$space, s$	$height, h$
Local	0,10	0,20	0,10	0,20
Intermediate	0,14	0,35	0,14	0,20

Table B.2: PTM interconnect parameters for a 70 nm technology

The *interconnect capacitance calculator* at the PTM website [39] is using the model from [43] incorrectly. The correct expressions, using the model from [43], of capacitance per length are

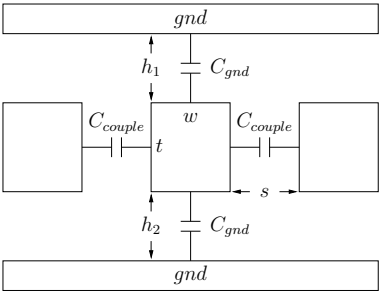

Figure B.2: Model of interconnect

$$C_{gnd} = \epsilon_0 k \left(\frac{w}{h} + 2.04 \left(\frac{t}{t+4.5311h} \right)^{0.071} \left(\frac{s}{s+0.5355h} \right)^{1.773} \right)$$

$$C_{couple} = \epsilon_0 k \left(1.4116 \frac{t}{s} e^{-\frac{4s}{s+8.014h}} + 2.3704 \left(\frac{w}{w+0.3078s} \right)^{0.25724} \cdot \left(\frac{h}{h+8.961s} \right)^{0.7571} \cdot e^{-\frac{2s}{s+6h}} \right)$$

and

$$C_{tot} = 2 \cdot C_{couple} + 2 \cdot C_{gnd}$$

Table B.3 and B.4 lists the interconnect capacitance. The dielectric constant for copper wire is $k = 2.2$ and it is assumed that the distances from wire to lower gnd layer, h_2 and wire to upper gnd layer, h_1 are equal, h.

wire type	$C_{gnd}\left[\frac{fF}{mm}\right]$	$C_{couple}\left[\frac{fF}{mm}\right]$	$C_{tot}\left[\frac{fF}{mm}\right]$
Local	44.933	≈ 0	89.866
Intermediate	49.926	≈ 0	99.852

Table B.3: Single wire capacitance, $s \gg w$

wire type	$C_{gnd}\left[\frac{fF}{mm}\right]$	$C_{couple}\left[\frac{fF}{mm}\right]$	$C_{tot}\left[\frac{fF}{mm}\right]$
Local	19.420	53.672	146.183
Intermediate	26.889	57.636	169.050

Table B.4: Multiple conductor capacitance

B.5 Subthreshold current

The following plots are of a 70 nm NMOS transistor using the design rules from appendix B.2, with operating conditions as listed in table B.5, unless specified otherwise in the caption of the plot. The corresponding HSpice netlists, are listed in appendix D.1.

supply voltage	$V_{dd} = 0.9V$
threshold voltage	$V_t = 0.16V$
operating temperature	$T = 25°\mathrm{C}$
transistor width	175nm
gate length	70nm

Table B.5: Parameters for the simulation of a NMOS transistor

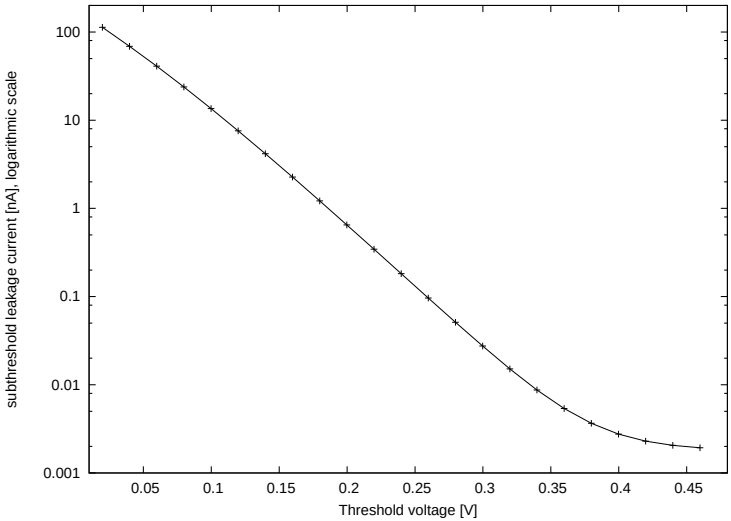

Figure B.3: Subthreshold current vs. threshold voltage

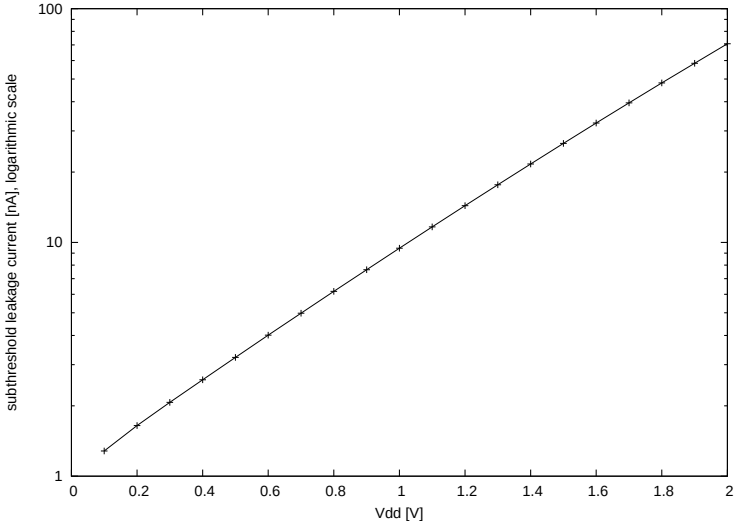

Figure B.4: Subthreshold current vs. supply voltage

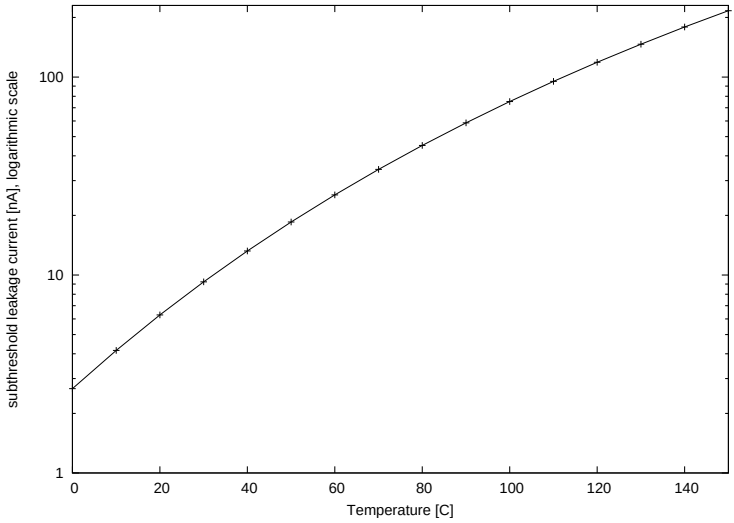

Figure B.5: Subthreshold current vs. temperature

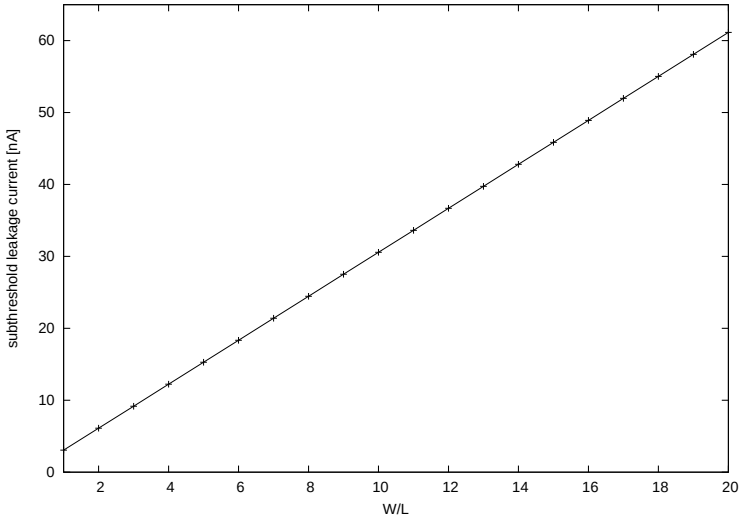

Figure B.6: Subthreshold current vs. transistor width

Appendix C

Computing power consumption with HSpice

HSpice version V-2004.03 and BSIM3v3 [38] model is used for the transistor level simulations. The BSIM3v3 model does not consider gate leakage, thus only the subthreshold leakage is simulated.

Computing the power consumption of the inverter shown in figure C.1 is done by measuring the current I_{vdd} at steady state and during a transition.

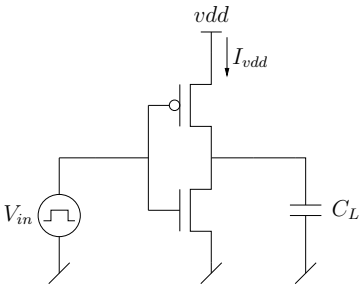

Figure C.1: Definition of I_{vdd} when measuring power of a sub circuit

When applying a falling transition at V_{in}, the current I_{vdd} will change (simplified) like shown in figure C.2.

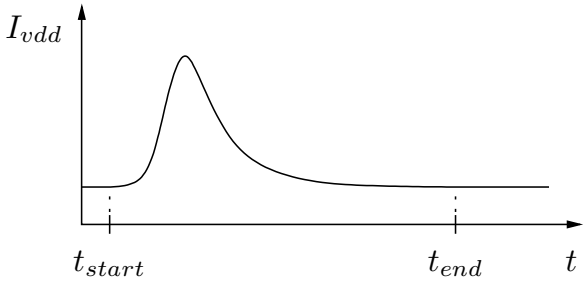

Figure C.2: I_{vdd} during a falling transition at V_{in}

The time at which the transition begins, is defined as t_{start} and the time where the current has decayed and settled at a constant value is defined as t_{end}.

The total energy used from the power supply during the transition is

$$E_{tot} = V_{dd} \int_{t_{start}}^{t_{end}} I_{vdd}(t)\ dt$$

At t_{end} the leakage power is the only non-zero term of equation 2.1, thus

$$P_{leakage} = V_{dd} \cdot I_{vdd}(t_{end})$$

Combining the two equations, the energy per switch (both *dynamic* and *short circuit*) is calculated as

$$E_{switch} = E_{tot} - (t_{end} - t_{start}) \cdot P_{leakage}$$

Finally the total power consumption with a switching frequency of f is

$$P_{total} = f \cdot E_{switch} + P_{leakage}$$

Appendix D

HSpice netlists of example circuits

D.1 Netlists related to chapter 2.3

D.1.1 Sweep of threshold voltage

```
.title NMOS1

* Michael Kristensen, 2004, mik@caffrey.dk
* NOTE: This spice deck requires a custom modelcard to allow ↩
    modification of Vt

* Parameters
.global vdd gnd
.param vdd=0.9V
.param vt0=0.02

.param lambda='70nm/2'
.param Lmin='2*lambda'
.param Wmin='5*lambda'
.param TpWidth=Wmin TnWidth=Wmin TLength=Lmin

* Cn: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.6e↩
    -9
* Cp: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.7e↩
    -9
* Cgate = Cn + Cp

.param Cgate='TnWidth*TLength*8.85418782e-12*3.9/1.6e-9 + TpWidth↩
    *TLength*8.85418782e-12*3.9/1.7e-9'
```

```
*.param Cgate=0.5145fF

.param S=2
.options POST=2 AUTOSTOP
.option METHOD=GEAR

* specify operating temperature
.temp 25

Xtn0 vdd a gnd gnd nmosths W='TnWidth' L='TLength'
+        AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+        PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
+        Vt='vt0'


* Voltage sources
Vvdd vdd gnd dc=vdd
Va a gnd 0

* Stimulus
.op

*.LIB 'bptm/70nm_ll' BPTM70LL_LIB
*.LIB 'bptm/70nm_hs' BPTM70HS_LIB
.LIB 'bptm/70nm_multi' BPTM70_LIB


* Analysis
*.probe V(Ain) I(vdd) V(out) V(before) V(after)

*.tran 50fs 2ns

.alter
.param vt0='0.04'
.alter
.param vt0='0.06'
.alter
.param vt0='0.08'
.alter
.param vt0='0.1'
.alter
.param vt0='0.12'
.alter
.param vt0='0.14'
.alter
.param vt0='0.16'
.alter
.param vt0='0.18'
.alter
.param vt0='0.2'
.alter
.param vt0='0.22'
.alter
.param vt0='0.24'
.alter
.param vt0='0.26'
.alter
.param vt0='0.28'
```

```
.alter
.param vt0='0.3'
.alter
.param vt0='0.32'
.alter
.param vt0='0.34'
.alter
.param vt0='0.36'
.alter
.param vt0='0.38'
.alter
.param vt0='0.4'
.alter
.param vt0='0.42'
.alter
.param vt0='0.44'
.alter
.param vt0='0.46'
.alter
.param vt0='0.48'
.alter
.param vt0='0.50'

.END
```

Modified NMOS modelcard

The modelcard has been modified to allow adjustments of V_t.

```
*
* Predictive Technology Model Beta Version
* 0.07um NMOS SPICE Parameters (normal one)
*

* Vt must be specified

.LIB     BPTM70_LIB

.SUBCKT nmosths   Drain     Gate      Source    Bulk
+       M=1       W=10e-6   AD='(6e-07)*w' PD='2*(6e-07)+w' NRD=0
+                 L=10e-6   AS='(6e-07)*w' PS='2*(6e-07)+w' NRS=0
.param  WOTn='9.35338e-08-(4.19715e-15/l)-(1.50197e-14/w)'

M1       Drain Gate Source Bulk    BPTM70HSN_TYP
+        W=W L=L AS=AS AD=AD PS=PS PD=PD M=M NRD=NRD NRS=NRS

.model BPTM70HSN_TYP NMOS

+Level = 49

+Lint = 1.6e-08 Tox = 1.6e-09
* user specified Vth0
+Vth0 = Vt
+Rdsw = 150

+lmin=7.0e-8 lmax=7.0e-8 wmin=0.7e-7 wmax=1.0e-4
+Tref=27.0    version =3.1   Xj= 2.9999999E-08 Nch= 1.2000000E+18
```

```
+lln = 1.0000000   lwn = 1.0000000   wln = 0.00   wwn = 0.00
+ll = 0.00   lw = 0.00 lwl = 0.00 wint = 0.00 wl = 0.00
+ww = 0.00 wwl = 0.00 Mobmod=1 binunit = 2 xl = 0.00 xw = 0.00
+Lmlt = 1 Wmlt = 1 binflag =  0 Dwg = 0.00 Dwb = 0.00

+ACM = 0 ldif =0.00 hdif =0.00 rsh = 6 rd = 0 rs = 0 rsc = 0 rdc = 0

+K1 = 0.3700000 K2 = 1.0000000E-02 K3 = 0.00
+Dvt0 = 1.3000000 Dvt1 = 0.5000000 Dvt2 = 2.9999999E-02 Dvt0w = 0.00
+Dvt1w = 0.00 Dvt2w = 0.00 Nlx = 7.0000000E-08 W0 = 0.00
+K3b = 0.00 Ngate = 5.0000000E+20

+Vsat = 1.1500000E+05 Ua = 5.0000000E-10 Ub = 1.0000000E-18 Uc↩
     = -2.9999999E-11
+Prwb = 0.00 Prwg = 0.00 Wr = 1.0000000 U0 = 2.5000000E-02 A0↩
     = 1.5000000
+Keta = 4.0000000E-02 A1 = 0.00 A2 = 1.0000000 Ags = -1.0000000E-02
+B0 = 0.00 B1 = 0.00

+Voff = -0.1500000 NFactor = 1.5000000 Cit = 0.00 Cdsc = 0.00 Cdscb↩
     = 0.00
+Cdscd = 1.0000000E-14 Eta0 = 0.2000000 Etab = 0.00 Dsub = 1.0000000

+Pclm = 0.2500000 Pdiblc1 = 1.2000000E-02 Pdiblc2 = 7.5000000E-03
+Pdiblcb = -1.3500000E-02 Drout = 1.5000000 Pscbe1 = 8.6600000E+08
+Pscbe2 = 1.0000000E-20 Pvag = -0.2800000 Delta = 1.0000000E-02
+Alpha0 = 0.00 Beta0 = 30.0000000

+kt1 = -0.3700000 kt2 = -4.0000000E-02 At = 5.5000000E+04
+Ute = -1.4800000 Ua1 = 9.5829000E-10 Ub1 = -3.3473000E-19
+Uc1 = 0.00 Kt1l = 4.0000000E-09 Prt = 0.00

+Cj = 0.0015 Mj = 0.72 Pb = 1.25 Cjsw = 2E-10 Mjsw = 0.37
+Php = 0.77 Cjgate = 2E-14 Cta = 0 Ctp = 0 Pta = 0 Ptp = 0
+JS=1.50E-08 JSW=2.50E-13 N=1.0 Xti=3.0
+Cgdo=4.094E-10 Cgso=4.094E-10 Cgbo=0.0E+00 Capmod = 2
+NQSMOD = 0 Elm = 5 Xpart = 1 cgsl = 1E-10 cgdl = 1E-10
+ckappa = 0.08 cf = 1.266e-10 clc = 1.0000000E-07 cle = 0.6000000
+Dlc = 1.6E-08 Dwc = 0

.ENDS

.ENDL
```

D.1.2 Sweep of supply voltage

```
.title NMOS1

* Michael Kristensen, 2004, mik@caffrey.dk

* Parameters
.global vdd gnd
.param vdd=0V

.param lambda='70nm/2'
```

```
.param Lmin='2*lambda'
.param Wmin='5*lambda'
.param TpWidth=Wmin TnWidth=Wmin TLength=Lmin

* Cn: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.6e↩
    -9
* Cp: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.7e↩
    -9
* Cgate = Cn + Cp

.param Cgate='TnWidth*TLength*8.85418782e-12*3.9/1.6e-9 + TpWidth↩
    *TLength*8.85418782e-12*3.9/1.7e-9'
*.param Cgate=0.5145fF

.param S=2
.options POST=2 AUTOSTOP
.option METHOD=GEAR

* specify operating temperature
.temp 25


Xtn0 vdd a gnd gnd nmosths W='TnWidth' L='TLength'
+        AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+        PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'


* Voltage sources
Vvdd vdd gnd dc=vdd
Va a gnd 0

* Stimulus
.op

*.LIB 'bptm/70nm_ll' BPTM70LL_LIB
.LIB 'bptm/70nm_hs' BPTM70HS_LIB


* Analysis
*.probe V(Ain) I(vdd) V(out) V(before) V(after)

*.tran 50fs 2ns

.alter
.param vdd= 0.1
.alter
.param vdd= 0.2
.alter
.param vdd= 0.3
.alter
.param vdd= 0.4
.alter
.param vdd= 0.5
.alter
.param vdd= 0.6
.alter
.param vdd= 0.7
.alter
```

```
.param vdd= 0.8
.alter
.param vdd= 0.9
.alter
.param vdd= 1.0
.alter
.param vdd= 1.1
.alter
.param vdd= 1.2
.alter
.param vdd= 1.3
.alter
.param vdd= 1.4
.alter
.param vdd= 1.5
.alter
.param vdd= 1.6
.alter
.param vdd= 1.7
.alter
.param vdd= 1.8
.alter
.param vdd= 1.9
.alter
.param vdd= 2.0

.END
```

D.1.3 Sweep of transistor width

```
.title NMOS1

* Michael Kristensen, 2004, mik@caffrey.dk

* Parameters
.global vdd gnd
.param vdd=0.9V

.param lambda='70nm/2'
.param Lmin='2*lambda'
.param Wmin='Lmin'
*.param Wmin='5*lambda'
.param TpWidth=Wmin TnWidth=Wmin TLength=Lmin

* Cn: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.6e↩
    -9
* Cp: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.7e↩
    -9
* Cgate = Cn + Cp

.param Cgate='TnWidth*TLength*8.85418782e-12*3.9/1.6e-9 + TpWidth↩
    *TLength*8.85418782e-12*3.9/1.7e-9'
*.param Cgate=0.5145fF

.param S=2
.options POST=2 AUTOSTOP
.option METHOD=GEAR
```

```
* specify operating temperature
.temp 25

Xtn0 vdd a gnd gnd nmosths W='TnWidth' L='TLength'
+         AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+         PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'


* Voltage sources
Vvdd vdd gnd dc=vdd
Va a gnd 0

* Stimulus
.op

*.LIB 'bptm/70nm_ll' BPTM70LL_LIB
.LIB 'bptm/70nm_hs' BPTM70HS_LIB


* Analysis
*.probe V(Ain) I(vdd) V(out) V(before) V(after)

*.tran 50fs 2ns

.alter
.param TnWidth='Lmin*2'
.alter
.param TnWidth='Lmin*3'
.alter
.param TnWidth='Lmin*4'
.alter
.param TnWidth='Lmin*5'
.alter
.param TnWidth='Lmin*6'
.alter
.param TnWidth='Lmin*7'
.alter
.param TnWidth='Lmin*8'
.alter
.param TnWidth='Lmin*9'
.alter
.param TnWidth='Lmin*10'
.alter
.param TnWidth='Lmin*11'
.alter
.param TnWidth='Lmin*12'
.alter
.param TnWidth='Lmin*13'
.alter
.param TnWidth='Lmin*14'
.alter
.param TnWidth='Lmin*15'
.alter
.param TnWidth='Lmin*16'
.alter
.param TnWidth='Lmin*17'
.alter
```

```
.param TnWidth='Lmin*18'
.alter
.param TnWidth='Lmin*19'
.alter
.param TnWidth='Lmin*20'

.END
```

D.1.4 Sweep of temperature

```
.title NMOS1

* Michael Kristensen, 2004, mik@caffrey.dk

* Parameters
.global vdd gnd
.param vdd=0.9V

.param lambda='70nm/2'
.param Lmin='2*lambda'
.param Wmin='5*lambda'
.param TpWidth=Wmin TnWidth=Wmin TLength=Lmin

* Cn: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.6e↩
    -9
* Cp: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.7e↩
    -9
* Cgate = Cn + Cp

.param Cgate='TnWidth*TLength*8.85418782e-12*3.9/1.6e-9 + TpWidth↩
    *TLength*8.85418782e-12*3.9/1.7e-9'
*.param Cgate=0.5145fF

.param S=2
.options POST=2 AUTOSTOP
.option METHOD=GEAR

* specify operating temperature
.temp 0


Xtn0 vdd a gnd gnd nmosths W='TnWidth' L='TLength'
+        AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+        PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'


* Voltage sources
Vvdd vdd gnd dc=vdd
Va a gnd 0

* Stimulus
.op

*.LIB 'bptm/70nm_ll' BPTM70LL_LIB
.LIB 'bptm/70nm_hs' BPTM70HS_LIB
```

```
* Analysis
*.probe V(Ain) I(vdd) V(out) V(before) V(after)

*.tran 50fs 2ns

.alter
.temp 10
.alter
.temp 20
.alter
.temp 30
.alter
.temp 40
.alter
.temp 50
.alter
.temp 60
.alter
.temp 70
.alter
.temp 80
.alter
.temp 90
.alter
.temp 100
.alter
.temp 110
.alter
.temp 120
.alter
.temp 130
.alter
.temp 140
.alter
.temp 150

.END
```

D.2 Netlists related to chapter 4

D.2.1 Static power

```
.title P:HS, N:HS
* three minimum-sized inverters
* measurement on inverter with different combinations of ↩
    threshold voltages

* Michael Kristensen, 2004, mik@caffrey.dk

* Parameters
.global vdd gnd
.param vdd=0.9V

.param lambda='70nm/2'
.param Lmin='2*lambda'
```

```
.param Wmin='5*lambda'
.param TpWidth=Wmin TnWidth=Wmin TLength=Lmin

* Cn: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.6e↩
    -9
* Cp: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.7e↩
    -9
* Cgate = Cn + Cp

.param Cgate='TnWidth*TLength*8.85418782e-12*3.9/1.6e-9 + TpWidth↩
    *TLength*8.85418782e-12*3.9/1.7e-9'
*.param Cgate=0.5145fF

.param S=2
.options POST=2 AUTOSTOP
.option METHOD=GEAR

* specify operating temperature
*.temp 25
.temp 70

.param invPmult=1

* The inverter that is being modified/tested
.subckt inv in out vdd
Xtp out in vdd vdd pmosths W='invPmult*TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

* input gate
.subckt invHS in out vdd
Xtp0 out in vdd vdd pmosths W='2.7*TpWidth' L='TLength'
*Xtp0 out in vdd vdd pmosths W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn0 out in gnd gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends


Xinv0 Ain before vdd invHS
Xinv1 before after vddi inv

Cout1 after gnd '4*Cgate + 2.25fF'


* Voltage sources
Vvdd vdd gnd dc=vdd
Vvddi vddi gnd dc=vdd

* Stimulus
* (low high delay risetime falltime duration period)
```

```
*VinA Ain gnd PULSE (0V vdd 0ps 10ps 10ps 985ps 2us)
VinA Ain gnd PULSE (0V vdd 0ps 15ps 15ps 985ps 2us)

.LIB 'bptm/70nm_ll' BPTM70LL_LIB
.LIB 'bptm/70nm_hs' BPTM70HS_LIB


* Analysis
.probe V(Ain) I(vdd) V(out) V(before) V(after)

.tran 50fs 2ns

* measure the rise/fall/tpd time
.measure tran tr trig v(after) val='0.1*vdd' rise=1 targ v(after)↩
      val='0.9*vdd' rise=1
.measure tran tf trig v(after) val='0.9*vdd' fall=1 targ v(after)↩
      val='0.1*vdd' fall=1
.measure tran tpdlh trig v(before) val='0.5*vdd' fall=1 targ v(↩
     after) val='0.5*vdd' rise=1
.measure tran tpphl trig v(before) val='0.5*vdd' rise=1 targ v(↩
     after) val='0.5*vdd' fall=1

* measure the leakage current when signal has settled
.measure tran cur_1 INTEG I(Vvddi) from=985ps to=986ps
.measure tran cur_2 INTEG I(Vvddi) from=1990ps to=1991ps

* measure the leakage POWER
.measure pleak1 PARAM='cur_1 * vdd / 1ps'
.measure pleak2 PARAM='cur_2 * vdd / 1ps'
.measure pleak_avg PARAM='(pleak1 + pleak2) / 2'

.measure tran tr1 trig v(before) val='0.1*vdd' rise=1 targ v(↩
     before) val='0.9*vdd' rise=1
.measure tran tf1 trig v(before) val='0.9*vdd' fall=1 targ v(↩
     before) val='0.1*vdd' fall=1


* then try other combinations of Vt...

.alter

.title P:HS, N:LL
.subckt inv in out vdd
Xtp out in vdd vdd pmosths W='invPmult*TpWidth' L='TLength'
+        AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+        PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmostll W='TnWidth' L='TLength'
+        AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+        PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

.alter

.title P:LL, N:HS
.subckt inv in out vdd
Xtp out in vdd vdd pmostll W='invPmult*TpWidth' L='TLength'
+        AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+        PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
```

```
Xtn out in gnd gnd nmosths W='TnWidth' L='TLength'
+        AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+        PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

.alter

.title P:LL, N:LL
.subckt inv in out vdd
Xtp out in vdd vdd pmostll W='invPmult*TpWidth' L='TLength'
+        AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+        PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmostll W='TnWidth' L='TLength'
+        AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+        PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends


.END
```

D.2.2 Dynamic power

```
* measure impact on switching power when the driving
* inverter is (Low-Vt / High-Vt)

* Michael Kristensen, 2004, mik@caffrey.dk

* the title of the first test:
.title A:HS, B:HS

* Parameters
.global vdd gnd
.param vdd=0.9V

.param lambda='70nm/2'
.param Lmin='2*lambda'
.param Wmin='5*lambda'
.param TpWidth=Wmin TnWidth=Wmin TLength=Lmin

* Cn: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.6e↩
    -9
* Cp: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.7e↩
    -9
* Cgate = Cn + Cp

.param Cgate='TnWidth*TLength*8.85418782e-12*3.9/1.6e-9 + TpWidth↩
    *TLength*8.85418782e-12*3.9/1.7e-9'
*.param Cgate=0.5145fF

.param S=2
.options POST=2 * AUTOSTOP
.option METHOD=GEAR

* specify operating temperature
*.temp 25
.temp 70
```

```
.param invPmult=1
.param invscale=1

* The inverter that is being modified
.subckt inv in out vdd
Xtp out in vdd vdd pmosths W='invscale*invPmult*TpWidth' L='↩
    TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmosths W='invscale*TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

* The inverters that are being measured
.subckt invTested in out vdd
Xtp out in vdd vdd pmosths W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

* input gate
.subckt invHS in out vdd
Xtp0 out in vdd vdd pmosths W='2.7*TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn0 out in gnd gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

Xinv0 Ain before vdd invHS
Xinv1 before t1 vdd1 inv

Xinv2a t1 out1 vdd2 invTested
Xinv2b t1 out2 vdd2 invTested
Xinv2c t1 out3 vdd2 invTested
Xinv2d t1 out4 vdd2 invTested

Cout1 out1 gnd '2*Cgate'
Cout2 out2 gnd '2*Cgate'
Cout3 out3 gnd '2*Cgate'
Cout4 out4 gnd '2*Cgate'

Cout5 t1 gnd '2.55fF'

* Voltage sources
Vvdd vdd gnd dc=vdd
Vvdd1 vdd1 gnd dc=vdd
Vvdd2 vdd2 gnd dc=vdd

* Stimulus
* (low high delay risetime falltime duration period)

VinA Ain gnd PULSE (0V vdd 0ps 15ps 15ps 985ps 2us)
```

```
.LIB 'bptm/70nm_ll' BPTM70LL_LIB
.LIB 'bptm/70nm_hs' BPTM70HS_LIB


* Analysis
.probe V(Ain) I(vdd) V(out) V(before) V(after) I(Vvdd2) I(out1)

.tran 25fs 2ns

* measure the rise/fall/tpd time

.measure tran tr_b trig v(t1) val='0.1*vdd' rise=1 targ v(t1) val↩
    ='0.9*vdd' rise=1
.measure tran tf_b trig v(t1) val='0.9*vdd' fall=1 targ v(t1) val↩
    ='0.1*vdd' fall=1

.measure tran tr_c trig v(out1) val='0.1*vdd' rise=1 targ v(out1)↩
     val='0.9*vdd' rise=1
.measure tran tf_c trig v(out1) val='0.9*vdd' fall=1 targ v(out1)↩
     val='0.1*vdd' fall=1

* measure the ENERGY consumption per transition

* first transistion
.measure tran cur_a_1 INTEG I(Vvdd1) from=0ps to=1000ps
.measure tran cur_a_1leak INTEG I(Vvdd1) from=985ps to=986ps
.measure tran cur_b_1 INTEG I(Vvdd2) from=0ps to=1000ps
.measure tran cur_b_1leak INTEG I(Vvdd2) from=985ps to=986ps

.measure tran e_a_1 PARAM='vdd * (cur_a_1 - 1000 * cur_a_1leak)'
.measure tran e_b_1 PARAM='vdd * (cur_b_1 - 1000 * cur_b_1leak)'

* total energy for the first transition
.measure tran e_1 PARAM='e_a_1 + e_b_1'

* second transition
.measure tran cur_a_2 INTEG I(Vvdd1) from=1000ps to=2000ps
.measure tran cur_a_2leak INTEG I(Vvdd1) from=1990ps to=1991ps

.measure tran cur_b_2 INTEG I(Vvdd2) from=1000ps to=2000ps
.measure tran cur_b_2leak INTEG I(Vvdd2) from=1990ps to=1991ps

.measure tran e_a_2 PARAM='vdd * (cur_a_2 - 1000 * cur_a_2leak)'
.measure tran e_b_2 PARAM='vdd * (cur_b_2 - 1000 * cur_b_2leak)'

* total energy for the second transition
.measure tran e_2 PARAM='e_a_2 + e_b_2'

* average energies

* both transistions
.measure tran e_avg PARAM='(e_1 + e_2) / 2'

* transistor a
.measure tran e_1_avg PARAM='(e_a_1 + e_a_2) / 2'

* transistor b's
```

```
.measure tran e_2_avg PARAM='(e_b_1 + e_b_2) / 2'


* measure the leakage POWER
.measure pleak1 PARAM='(cur_a_1leak + cur_b_1leak) * vdd / 1ps'
.measure pleak2 PARAM='(cur_a_2leak + cur_b_2leak) * vdd / 1ps'
.measure pleak_avg PARAM='(pleak1 + pleak2) / 2'

.measure f PARAM='pleak_avg / e_avg'
.measure tot PARAM='pleak_avg + 500e6 * e_avg'

.alter
.title A:LL, B:HS

.subckt inv in out vdd
Xtp out in vdd vdd pmostll W='invscale*invPmult*TpWidth' L='↩
    TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmostll W='invscale*TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

.subckt invTested in out vdd
Xtp out in vdd vdd pmosths W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends


.alter
.title A:HS, B:LL

.subckt inv in out vdd
Xtp out in vdd vdd pmosths W='invscale*invPmult*TpWidth' L='↩
    TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmosths W='invscale*TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

.subckt invTested in out vdd
Xtp out in vdd vdd pmostll W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmostll W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

.alter
```

```
.title A:LL, B:LL

.subckt inv in out vdd
Xtp out in vdd vdd pmostll W='invscale*invPmult*TpWidth' L='↩
    TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmostll W='invscale*TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

.subckt invTested in out vdd
Xtp out in vdd vdd pmostll W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'
Xtn out in gnd gnd nmostll W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'
.ends

.END
```

D.2.3 NAND3–gate

```
.title NAND3

* Michael Kristensen, 2004, mik@caffrey.dk

* Parameters
.global vdd gnd
.param vdd=0.9V

.param lambda='70nm/2'
.param Lmin='2*lambda'
.param Wmin='5*lambda'
.param TpWidth=Wmin TnWidth=Wmin TLength=Lmin

* Cn: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.6e↩
    -9
* Cp: A * e_0 * e_si / t_ox = W * L * 8.85418782e-12 * 3.9 / 1.7e↩
    -9
* Cgate = Cn + Cp

.param Cgate='TnWidth*TLength*8.85418782e-12*3.9/1.6e-9 + TpWidth↩
    *TLength*8.85418782e-12*3.9/1.7e-9'
*.param Cgate=0.5145fF

.param S=2
.options POST=2 AUTOSTOP
.option METHOD=GEAR

* specify operating temperature
.temp 25
*.temp 70
```

```
Xtp0 out a vdd vdd pmosths W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'

Xtp1 out b vdd vdd pmosths W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'

Xtp2 out c vdd vdd pmosths W='TpWidth' L='TLength'
+       AS='TpWidth*5.5*lambda' AD='TpWidth*5.5*lambda'
+       PS='TpWidth+10*lambda' PD='TpWidth+10*lambda'

Xtn0 out a 5 gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'

Xtn1 5 b 6 gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'

Xtn2 6 c gnd gnd nmosths W='TnWidth' L='TLength'
+       AS='TnWidth*5.5*lambda' AD='TnWidth*5.5*lambda'
+       PS='TnWidth+10*lambda' PD='TnWidth+10*lambda'


* Voltage sources
Vdd vdd gnd dc=vdd
Va a gnd 0
Vb b gnd 0
Vc c gnd 0

* Stimulus
.op

.LIB 'bptm/70nm_ll' BPTM70LL_LIB
.LIB 'bptm/70nm_hs' BPTM70HS_LIB


* Analysis
*.probe V(Ain) I(vdd) V(out) V(before) V(after)

*.tran 50fs 2ns

.alter
Vdd vdd 0 dc=vdd
Va a 0 vdd
Vb b 0 0
Vc c 0 0

.alter
Vdd vdd 0 dc=vdd
Va a 0 0
Vb b 0 vdd
Vc c 0 0

.alter
Vdd vdd 0 dc=vdd
Va a 0 vdd
```

```
Vb b 0 vdd
Vc c 0 0

.alter
Vvdd vdd 0 dc=vdd
Va a 0 0
Vb b 0 0
Vc c 0 vdd

.alter
Vvdd vdd 0 dc=vdd
Va a 0 vdd
Vb b 0 0
Vc c 0 vdd

.alter
Vvdd vdd 0 dc=vdd
Va a 0 0
Vb b 0 vdd
Vc c 0 vdd

.alter
Vvdd vdd 0 dc=vdd
Va a 0 vdd
Vb b 0 vdd
Vc c 0 vdd
.END
```

Appendix E

Using Synopsys for reducing leakage power

E.1 Synopsys optimization

E.1.1 Incremental compilation

Cell by cell substitution is done by executing the command `compile -incremental_mapping` on an already synthesized design. This will replace gate g by a functionally identical gate g' but with other characteristics in terms of speed and power.

It is expected that the number of gates in the circuit remains unchanged after the compilation, however this is not the case. Figure E.1 and E.2 show the change in the number of single–input gates (inverters and buffers) in the adder design, when it is compiled incrementally. The processor design shows same behavior in figure E.3 and E.4. In both designs, the number of gates with more than one input remained constant, after application of method 4 and 5.

Excerpt from the Design Compiler manual [35] regarding incremental compilation: "Portions of a design that are already mapped are exempt from logic level optimization, and the resulting design should be the same (if no improvements can be made) or better in terms of its design constraints."

As the timing constraint has higher priority than power, an possible explanation of the single–input cells could be, that they help in satisfying the timing constraint. Highly loaded nets could be subject to buffering, if the cell that drives the net, is changed from low V_t to high V_t. However, if the single–input cells are inserted, such that the timing constraint can be fulfilled, it is expected that the number of cells will be high for strict timing constraints and decrease as the timing constraint is relaxed. From the figures E.1–E.4 is seen, that the number of single–input cells changes in a very irregular and unpredictable behavior, thus it is unlikely that their use are for just delay reductions.

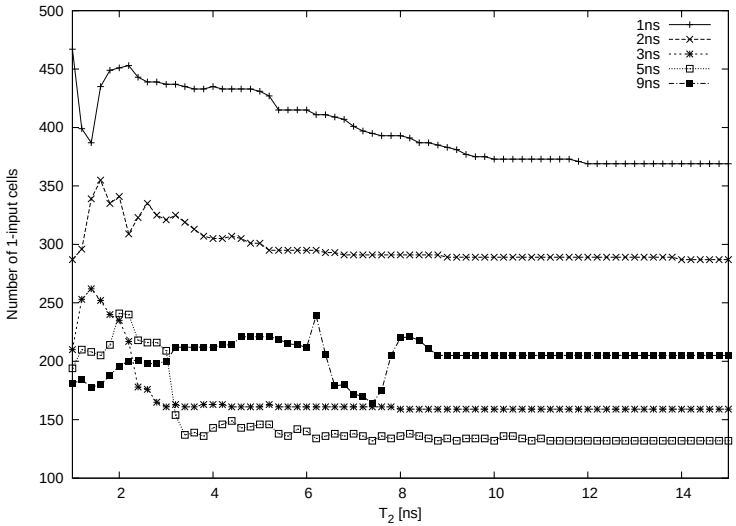

Figure E.1: Number of 1-input cells with method 4 for the adder design

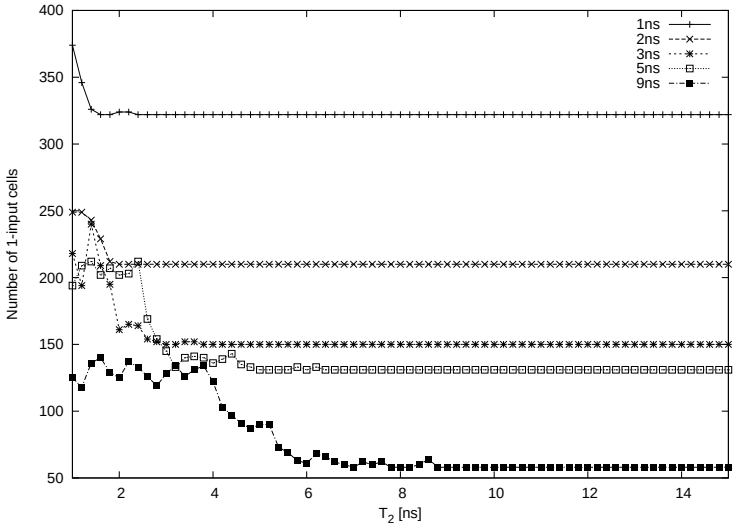

Figure E.2: Number of 1-input cells with method 5 for the adder design

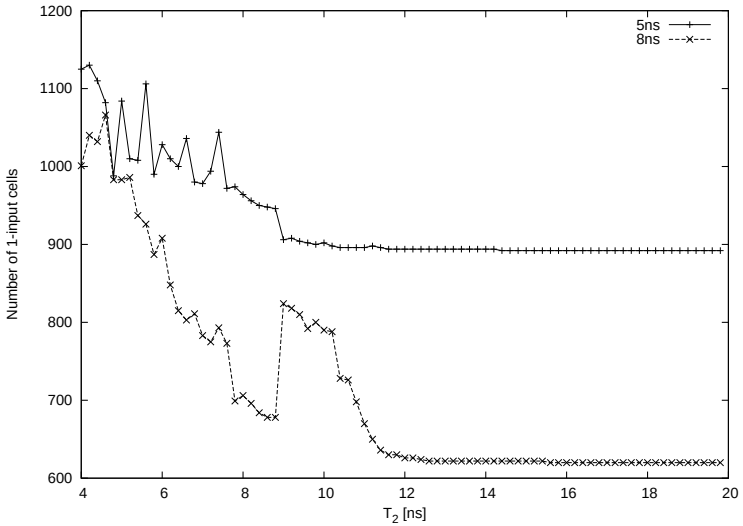

Figure E.3: Change in the number of inverters with method 4 for the processor design

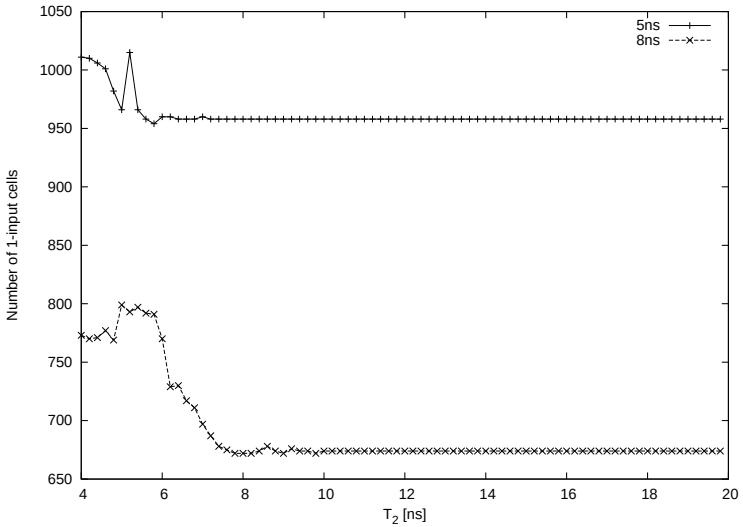

Figure E.4: Change in the number of inverters with method 5 for the processor design

E.1.2 Inverter reduction

Manual inspection of the synthesized circuits generated by Synopsys Design Compiler, has shown some peculiar logic constructions. Especially the reduction of unnecessary inverters acts strange and results in an increase in both delay and power consumption.

In figure E.5 are shown three examples of the unnecessary inverters.

The series connected inverters are commonly used for buffering long or heavy loaded nets, but for a small compact logic circuit such as a full adder, there shouldn't be a need for buffering.

The NAND–gate with an inverting input is unfortunate both in terms of leakage power, dynamic power and delay. The same can be said about the AND–gate followed by an inverter. These constructions mainly appeared in the dual V_t implementation of the processor design. The reason for the origin of these constructions was not found.

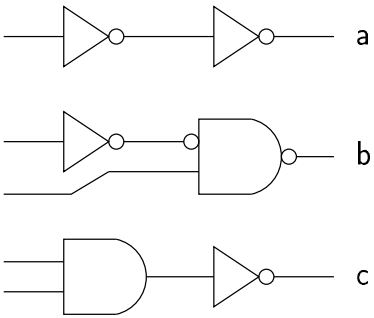

Figure E.5: Examples of suboptimal inverter constructions

E.1.3 Cost priority

When Design Compiler synthesizes a design, a constraints priority vector is used for guiding the optimization algorithms. Two categories of constraints are used:

The design rule constraints specifies some requirements that must be fulfilled, in order to guarantee that the circuit operates correctly at

the logic level. The technology library defines the design rule constraints and the constraints priority vector assigns weights to each constraint.

The optimization constraints are the goals defined by the designer. As for the design rule constraints, the weight of each constraint is specified in the constraints priority vector.

The *default* priority vector [35] is listed in table E.1. The priorities listed in parentheses in the table, cannot be reordered, thus delay will always be considered before power. If there is a conflict between two or more constraints, ex. delay versus power, then the priority of the constraints are used to decide which constraint to satisfy.

Priority	Constraint	Constraint type
(1)	Minimum capacitance	design rule
2	Maximum transition time	design rule
3	Maximum fanout	design rule
4	Maximum capacitance	design rule
5	Cell degradation	design rule
6	Maximum delay	optimization
7	Minimum delay	optimization
(8)	Power { Dynamic / Leakage }	optimization
(9)	Area	optimization

Table E.1: Design Compiler constraints priority (decreasing order)

E.2 Ungrouping

To give most flexibility in the synthesis, the adder design have been compiled with the option `ungroup_all`. When ungrouping, the hierarchy in the design is collapsed which allows for better optimizations. Ungrouping was not successful for the processor design, the result of the synthesis is shown in figure E.6. The figure clearly shows two things that cannot be correct:

- The area suddenly drops when t_{clk} is lowered. This *could* be because Synopsys changes the wireload model to a model with shorter (less

capacitance) wires and thereby allows the use of smaller gates (eg. less drive strength), however the wireload model is not changed.

- For $5.2\text{ns} \leq t_{\text{clk}} \leq 6.2\text{ns}$ the area of the high V_t implementation is significantly less than the area of the low V_t design. This is contradicts with the usual behavior, that more high V_t cells than low V_t cells are needed to satisfy a strict timing constraint.

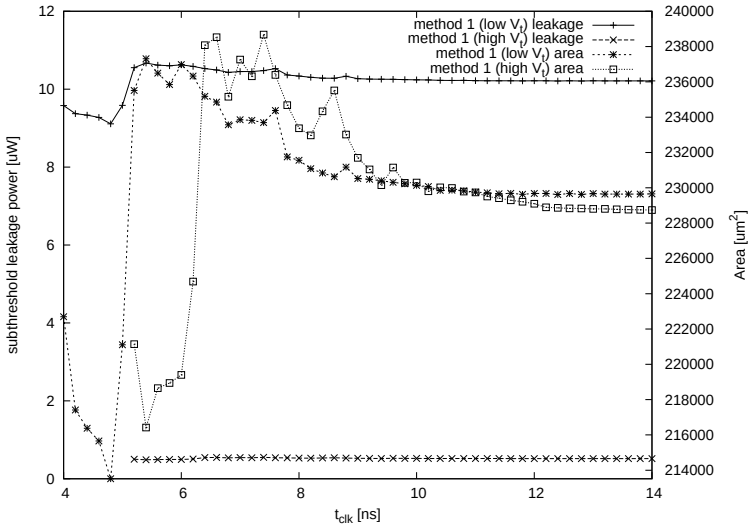

Figure E.6: Leakage power of processor with *ungrouping*

E.3 Synthesis method

In order to synthesize a design for *several* timing constraints, the synthesis process has been automated. Each synthesis script is embedded in a Perl script which iterates over the timing constraints and collects output from the synthesis tool. These scripts are listed in appendix F.1.3 and F.2.2.

Synopsys has been configured with two cell libraries, such that the selection of the low V_t or high V_t cell library easily can be done from the synthesis scripts. In the scripts the libraries are abbreviated `CORELIB_HS`

and CORELIB_LL respectively. The configuration files enabling the use of two cell libraries are listed in appendix E.3.1.

E.3.1 Synopsys configuration files

Synopsys V-2003.12-SP1-4 dated 28th of May 2004, has been installed on the server sunfire at the department, in /home322/s973396/synopsys.

.synopsys_dc.setup

```
/* Unix env variables  */
/*----------------------*/

/* Define path directories for file locations */
source_path = "./src/"
script_path = "./scr/"
log_path = "./log/"
db_path = "./db/"
netlist_path = "./netlist/"
cache_read = "~/cache/
cache_write = "~/cache/

search_path = search_path + {./src}
designer="Michael Kristensen"
company="IMM, DTU"


SYNOPSYS         = get_unix_variable("SYNOPSYS")

/*********************************************************/

/* OPCOND = nom_1.8 */
SYNOPSYS_VERSION = "2003.12"


UNICAD_DIR = "/ult/DK_HCMOS8_3.3"

CORELIB_OPCOND = "bc_1.60V_m40C"
CORELIB_VERSION = "3.1.a"

/* high speed */
CORELIB_HS_DIR = UNICAD_DIR + /CORELIB8DHS_HCMOS8D_ + ↩
     CORELIB_VERSION + /SYNOPSYS_DP/ + CORELIB_OPCOND
CORELIB_HS_DB = CORELIB8DHS.db
CORELIB_HS_SDB_DIR = UNICAD_DIR + /CORELIB8DHS_HCMOS8D_ + ↩
     CORELIB_VERSION + /SYNOPSYS_DP/
CORELIB_HS_SDB = CORELIB8DHS.sdb
CORELIB_HS = CORELIB_HS_DIR + "/" + CORELIB_HS_DB

/* low leakage */
CORELIB_LL_DIR = UNICAD_DIR + /CORELIB8DLL_HCMOS8D_ + ↩
     CORELIB_VERSION + /SYNOPSYS_DP/ + CORELIB_OPCOND
```

```
CORELIB_LL_DB = CORELIB8DLL.db
CORELIB_LL_SDB_DIR = UNICAD_DIR + /CORELIB8DLL_HCMOS8D_ + ↩
    CORELIB_VERSION + /SYNOPSYS_DP/
CORELIB_LL_SDB = CORELIB8DLL.sdb
CORELIB_LL = CORELIB_LL_DIR + "/" + CORELIB_LL_DB

/* IOlib */
IOLIB_VERSION = "4.0"
IOLIB_OPCOND = "bc_1.60V_m40C"
IOLIB_DIR = UNICAD_DIR + /IOLIB_80_HCMOS8D_ + IOLIB_VERSION + /↩
    SYNOPSYS_DC/ + IOLIB_OPCOND
IOLIB_DB = IOLIB_80.db
IOLIB_SDB_DIR = UNICAD_DIR + /IOLIB_80_HCMOS8D_ + IOLIB_VERSION↩
    + /SYNOPSYS_DC/
IOLIB_SDB = IOLIB_80.sdb
IOLIB = IOLIB_DIR + "/" + IOLIB_DB

search_library = { . , CORELIB_HS_DIR, CORELIB_LL_DIR, IOLIB_DIR↩
    }
search_library = search_library + { \
  /home322/s973396/synopsys/syn/doc/syn/interfaces/cadence/↩
      libraries/, \
  /home322/s973396/synopsys/syn/libraries/syn }


synthetic_library = { \
  /home322/s973396/synopsys/syn/libraries/syn/dw01.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw02.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw03.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw04.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw05.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw06.sldb, \
  standard.sldb}

symbol_library = { CORELIB_HS_SDB_DIR + "/" + CORELIB_HS_SDB ,\
                   CORELIB_LL_SDB_DIR + "/" + CORELIB_LL_SDB ,\
                     basic.sdb, ripper.sdb, class.sdb, ↩
                         IOLIB_SDB_DIR + "/" + IOLIB_SDB}

search_path = {  . , CORELIB_HS_DIR, CORELIB_LL_DIR, IOLIB_DIR }
search_path = search_path + { \
  /home322/s973396/synopsys/syn/libraries/syn, \
  /home322/s973396/synopsys/syn/doc/syn/interfaces/cadence/↩
      libraries }


vhdlout_use_packages = {IEEE.std_logic_1164, CORELIB.all, ↩
    CORELIB8DHS.all, CORELIB8DLL.all}
vhdlout_dont_write_types = "TRUE"


link_library = "*" + target_library + { \
  /home322/s973396/synopsys/syn/libraries/syn/dw01.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw02.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw03.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw04.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw05.sldb, \
  /home322/s973396/synopsys/syn/libraries/syn/dw06.sldb, \
```

```
  dw01.sldb, dw02.sldb, dw03.sldb, dw04.sldb, dw05.sldb, dw06.↩
      sldb }


/* to avoid escaped net names like: \net1234[3], and get wire ↩
    declarations */
hdlout_internal_busses = true
bus_naming_style = %s[%d]
bus_inference_style = bus_naming_style
```

.synopsys_vss.setup

```
TIMEBASE = ps
WORK > DEFAULT
DEFAULT : WORK
CORELIB8DHS : /ult/DK_HCMOS8_3.3/CORELIB8DHS_HCMOS8D_3.1.a/↩
    SYNOPSYS_VSS/lib_VITAL
```

E.4 DC_PERL script for dual V_t substitution

```
# dc_perl script that interfaces with Synopsys and
# changes all low-Vt cells to high-Vt cells.

# The runtime of the script is relatively long,
# thus it cannot be recommended to extend this script with the ↩
    purpose
# of making it useful.

# replace adder.db and "adder" with name of db-file and design ↩
    name

use strict;
use Synopsys;

$Synopsys::Verbose = 1;

print "Starting Synopsys...\n";
my $shell = Synopsys->new('dc_shell') or die "Init of dc_shell ↩
    failed, exiting.\n";

$shell->variable('target_library', 'CORELIB_LL', 'CORELIB_HS', '↩
    IOLIB');
$shell->variable('link_library', '"*"', 'CORELIB_LL', 'CORELIB_HS↩
    ', 'IOLIB');

# read a flat Synopsys db-file
$shell->read("-f db adder.db");

$shell->current_design("adder");
my $cellstr = $shell->output_string();
print $cellstr;

$shell->link;
my $cellstr = $shell->output_string();
```

```
print $cellstr;

# export it to vhdl
$shell->write("-f vhdl -output before.vhd");

my $cell_count=0;

# find and substitute cells
$shell->find("cell, U*, -hierarchy");
my $cellstr = $shell->output_string();
$cellstr =~ s/[{}" ]//g;
chomp($cellstr);
my @cells = split /,/, $cellstr;

foreach my $gate (@cells) {
  $shell->get_attribute("$gate ref_name");
  my $o = $shell->output_lastline();
  chomp($o);
  if ($o =~ /{"(\w+)"}/) {
    my $old = $1;
    my $new = $old;
    $new =~ s/HS/LL/;

# this is where the cell is substituted
    $shell->change_link("$gate CORELIB8DLL/$new");
    print "$gate: $old -> $new\n";
    $cell_count++;
  } else {
    print "Oops, unexpected answer: '$o'\n";
  }
}

print "$cell_count cells changed to high-Vt\n";

# export vhdl netlist
$shell->write("-f vhdl -output after.vhd");

$shell->exit;
```

Appendix F

Reference designs

F.1 Microprocessor

F.1.1 Synthesis results

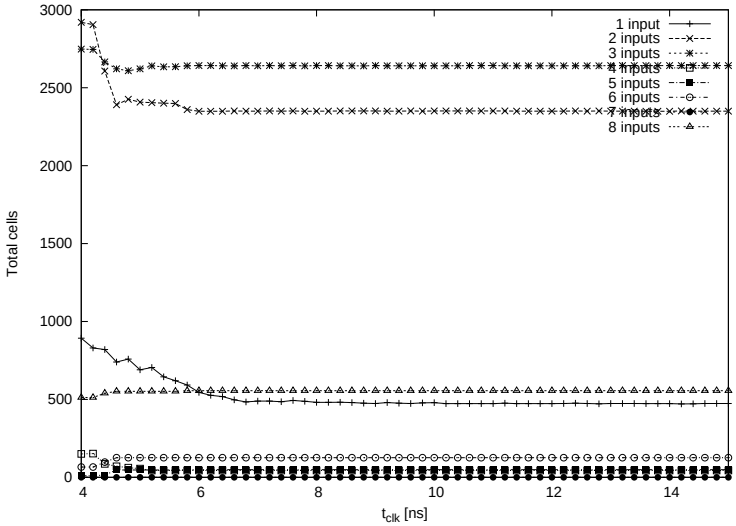

Figure F.1: Distribution of cells with method 1 (low V_t)

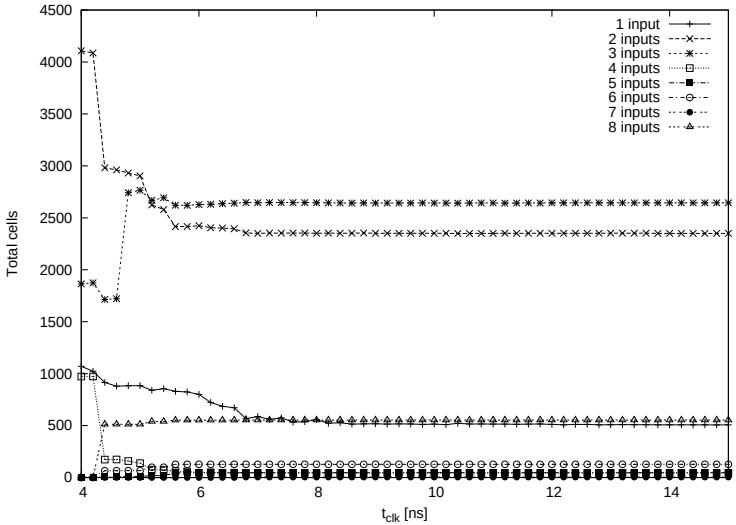

Figure F.2: Distribution of cells with method 1 (high V_t)

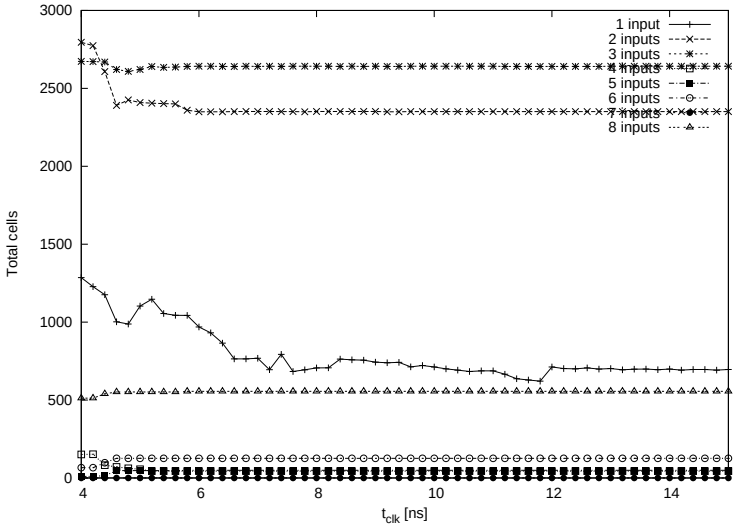

Figure F.3: Distribution of cells with method 2

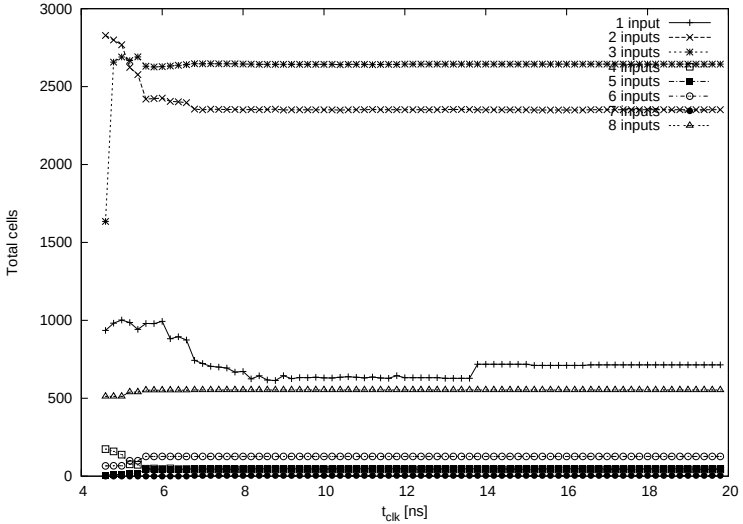

Figure F.4: Distribution of cells with method 3

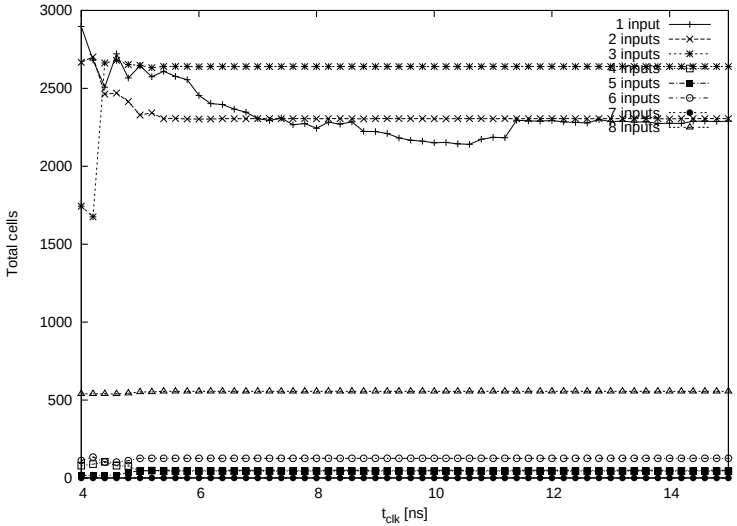

Figure F.5: Distribution of cells with method 6

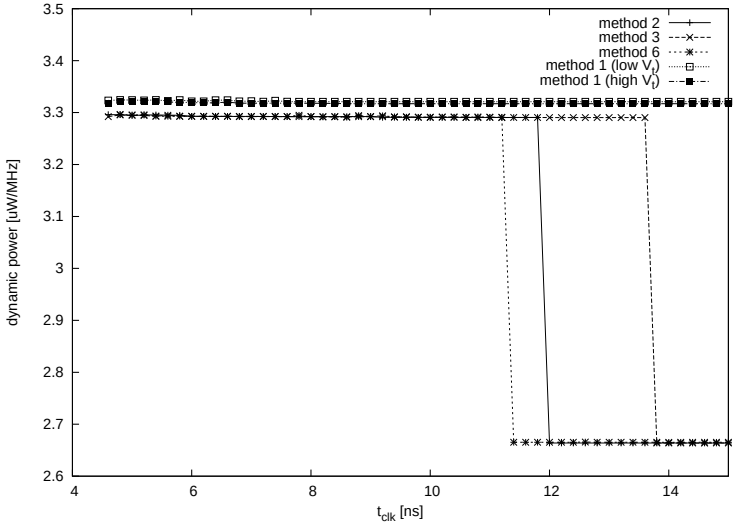

Figure F.6: Suspicious dynamic power calculation of processor

F.1.2 VHDL source

The VHDL source for the microprocessor is available on the CD, see appendix G.

F.1.3 Synthesis scripts

The synthesis methods refers to the descriptions from chapter 6.2. Location of files and programs might need to be adjusted depending on the system used. The runtime of each script is about 24 hours, when using the server at the department.

Method 1 – Low V_t

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=4.0;
$increment=0.2;
$runs=81;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_HS, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl" + search_path
analyze -format vhdl -lib WORK {types.vhd defs.vhd alu.vhd ↩
    shifter.vhd regfile.vhd cpu2.vhd minimips.vhd }
elaborate minimips -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify

set_max_area 0

compile -map_effort medium
```

```
report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output mips${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 1 – High V_t

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=4.0;
$increment=0.2;
$runs=81;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_LL, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl" + search_path
analyze -format vhdl -lib WORK {types.vhd defs.vhd alu.vhd ↩
    shifter.vhd regfile.vhd cpu2.vhd minimips.vhd }
elaborate minimips -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify

set_max_area 0

compile -map_effort medium

report_power > power${iteration}.rpt
```

```
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output mips${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 2

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=4.0;
$increment=0.2;
$runs=81;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_HS, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl" + search_path
analyze -format vhdl -lib WORK {types.vhd defs.vhd alu.vhd ↩
    shifter.vhd regfile.vhd cpu2.vhd minimips.vhd }
elaborate minimips -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify

compile -map_effort medium

report_power > power_hs${iteration}.rpt
report_area > area_hs${iteration}.rpt
report_timing > timing_hs${iteration}.rpt
report_reference -hier > reference_hs${iteration}.rpt
```

```
write -f db -hier -output mips_hs${iteration}.db


target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output mips${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 3

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=4.0;
$increment=0.2;
$runs=81;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_LL, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl" + search_path
analyze -format vhdl -lib WORK {types.vhd defs.vhd alu.vhd ↩
    shifter.vhd regfile.vhd cpu2.vhd minimips.vhd }
elaborate minimips -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
```

```
link
uniquify

compile -map_effort medium

report_power > power_hs${iteration}.rpt
report_area > area_hs${iteration}.rpt
report_timing > timing_hs${iteration}.rpt
report_reference -hier > reference_hs${iteration}.rpt

write -f db -hier -output mips_hs${iteration}.db


target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output mips${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 4 – 4 ns

The scripts for 5 and 8 ns are very similar to the script listed below, these scripts are availably digitally.

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=4.0;
$increment=0.2;
$runs=81;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
```

```
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_HS, IOLIB }
link_library = link_library + target_library

read -f db /home322/s973396/mm/mips/hs/mips_hs1.db
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }

target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 5 – 4 ns

The scripts for 5 and 8 ns are very similar to the script listed below, these scripts are availably digitally.

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=4.0;
$increment=0.2;
$runs=81;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
```

```
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_LL, IOLIB }
link_library = link_library + target_library

read -f db /home322/s973396/mm/mips/ll/mips_hs1.db
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }

target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 6

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=4.0;
$increment=0.2;
$runs=81;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";
```

```
    print OF <<END;
target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl" + search_path
analyze -format vhdl -lib WORK {types.vhd defs.vhd alu.vhd ↩
    shifter.vhd regfile.vhd cpu2.vhd minimips.vhd }
elaborate minimips -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify

set_max_leakage_power 0 mW
compile -map_effort medium

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt
write -f db -hier -output mips${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

F.2 Adder

F.2.1 VHDL source

```
--  Title      :  Fulladder
--  Developers :  Michael Kristensen
--  Revision   :  1.0    01-04-04

library ieee;
use ieee.std_logic_1164.all;
entity adder_test is
generic (
  size : integer := 64);
  port (
    clk, reset : in  std_logic;
    dina, dinb : in  std_logic_vector(size-1 downto 0);
    dout       : out std_logic_vector(size-1 downto 0);
    cout       : out std_logic);
end adder_test;

architecture rtl of adder_test is
  component adder
    generic (
```

```
      size : integer);
    port (
      a, b : in  std_logic_vector(size-1 downto 0);
      sum  : out std_logic_vector(size-1 downto 0);
      cout : out std_logic);
  end component;

  signal a_i, b_i : std_logic_vector(size-1 downto 0);
  signal sum_i    : std_logic_vector(size-1 downto 0);
  signal cout_i   : std_logic;

begin
  adder_inst: adder
    generic map (
        size => size)
    port map (
        a    => a_i,
        b    => b_i,
        sum  => sum_i,
        cout => cout_i);

  process (clk, reset)
  begin
    if reset = '0' then
      dout <= (others => '0');
      cout <= '0';
      a_i <= (others => '0');
      b_i <= (others => '0');
    elsif clk'event and clk = '1' then
      a_i <= dina;
      b_i <= dinb;
      dout <= sum_i;
      cout <= cout_i;
    end if;
  end process;

end rtl;


library ieee;
use ieee.std_logic_1164.all;

entity fa is
  port (a, b, cin : in  std_logic;
        sum, cout : out std_logic);
end fa;

architecture rtl of fa is
  signal c : std_logic;
begin

  cout <= (a and b) or (a and cin) or (b and cin);
  sum  <= (a xor b) xor cin;

end rtl;


library ieee;
```

```
use ieee.std_logic_1164.all;
entity adder is
  generic (
    size : integer := 64);
  port (
    a, b : in  std_logic_vector(size-1 downto 0);
    sum  : out std_logic_vector(size-1 downto 0);
    cout : out std_logic);
end adder;

architecture rtl of adder is
  component fa
    port (
      a, b, cin : in  std_logic;
      sum, cout : out std_logic);
  end component;

  signal carry : std_logic_vector(size-1 downto 0);
  signal zero  : std_logic;

begin

  zero <= '0';

  instantiate_adders : for i in size-2 downto 1 generate
    fa_inst          : fa
      port map (
        a    => a(i),
        b    => b(i),
        cin  => carry(i-1),
        sum  => sum(i),
        cout => carry(i));

  end generate instantiate_adders;

  fa_inst_first : fa
    port map (
      a    => a(0),
      b    => b(0),
      cin  => zero,
      sum  => sum(0),
      cout => carry(0));

  fa_inst_last : fa
    port map (
      a    => a(size-1),
      b    => b(size-1),
      cin  => carry(size-2),
      sum  => sum(size-1),
      cout => cout);

end rtl;
```

F.2.2 Synthesis scripts

The synthesis methods refers to the descriptions from chapter 6.2. Location of files and programs might need to be adjusted depending on the system used.

Method 1 – Low V_t

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=1.0;
$increment=0.2;
$runs=101;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_HS, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl_adder" + search_path
analyze -format vhdl -lib WORK {adder.vhd}
elaborate adder_test -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify

set_max_area 0

compile -map_effort medium -ungroup_all

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END
```

```
    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration}
        } 2>time_${iteration};`
}
```

Method 1 – High V_t

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=1.0;
$increment=0.2;
$runs=101;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_LL, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl_adder" + search_path
analyze -format vhdl -lib WORK {adder.vhd}
elaborate adder_test -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half
    " } { "clk" }
link
uniquify

set_max_area 0

compile -map_effort medium -ungroup_all

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;
```

```
    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 2

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=1.0;
$increment=0.2;
$runs=101;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_HS, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl_adder" + search_path
analyze -format vhdl -lib WORK {adder.vhd}
elaborate adder_test -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify

compile -map_effort medium -ungroup_all

report_power > power_hs${iteration}.rpt
report_area > area_hs${iteration}.rpt
report_timing > timing_hs${iteration}.rpt
report_reference -hier > reference_hs${iteration}.rpt

write -f db -hier -output adder_hs${iteration}.db


target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
```

```
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 3

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=1.0;
$increment=0.2;
$runs=101;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_LL, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl_adder" + search_path
analyze -format vhdl -lib WORK {adder.vhd}
elaborate adder_test -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify

compile -map_effort medium -ungroup_all

report_power > power_ll${iteration}.rpt
report_area > area_ll${iteration}.rpt
report_timing > timing_ll${iteration}.rpt
report_reference -hier > reference_ll${iteration}.rpt

write -f db -hier -output adder_ll${iteration}.db
```

```
target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 4 – 1 ns

The scripts for 2,3,5 and 9 ns are very similar to the script listed below, these scripts are availably digitally.

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=1.0;
$increment=0.2;
$runs=101;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_HS, IOLIB }
link_library = link_library + target_library

read -f db /home322/s973396/mm/vhdl_adder/ug_hs/adder_hs1.db
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }

target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
```

```
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration}
        } 2>time_${iteration};`
}
```

Method 5 – 1 ns

The scripts for 2,3,5 and 9 ns are very similar to the script listed below, these scripts are availably digitally.

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=1.0;
$increment=0.2;
$runs=101;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_LL, IOLIB }
link_library = link_library + target_library

read -f db /home322/s973396/mm/vhdl_adder/ug_ll/adder_ll1.db
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half
    " } { "clk" }

target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
```

```
link_library = link_library + target_library

set_max_leakage_power 0 mW
compile -map_effort medium -incremental_mapping

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Method 6

```
#!/home322/s973591/perl/bin/perl

`mkdir WORK`;

$iteration=0;

$start_val=1.0;
$increment=0.2;
$runs=101;

while ($iteration < $runs) {

    $clk = $start_val + $iteration * $increment;
    $clk_half = $clk/2;

    $iteration++;
    print "Now running clk=$clk\n";

    open (OF, ">run_me") or die "can't open file $fname\n";

    print OF <<END;
target_library = { CORELIB_LL, CORELIB_HS, IOLIB }
link_library = link_library + target_library

search_path = "/home322/s973396/mm/vhdl_adder" + search_path
analyze -format vhdl -lib WORK {adder.vhd}
elaborate adder_test -arch rtl
create_clock -name "clk" -period $clk -waveform { "0" "$clk_half↩
    " } { "clk" }
link
uniquify
```

```
set_max_leakage_power 0 mW
compile -map_effort medium -ungroup_all

report_power > power${iteration}.rpt
report_area > area${iteration}.rpt
report_timing > timing${iteration}.rpt
report_reference -hier > reference${iteration}.rpt

write -f db -hier -output adder${iteration}.db

exit
END

    close OF;

    `/usr/bin/time dc_shell -f run_me > dc_shell_log_${iteration↩
        } 2>time_${iteration};`
}
```

Appendix G

Digital appendices

The directory structure of the digital appendices

`/spice/` Spice netlists

`/spice/bptm/` BPTM modelcards

`/synopsys/configuration/` Configuration files for Synopsys

`/synopsys/dc_perl/` DC_PERL script

`/synopsys/scripts/` Scripts used for collecting results from the synteses

`/synopsys/mips/vhdl/` VHDL source files

`/synopsys/mips/scripts/` Synthesis scripts

`/synopsys/adder/vhdl/` VHDL source files

`/synopsys/adder/scripts/` Synthesis scripts

To run a synthesis script, copy the script to an empty directory and start the synthesis by executing `perl scriptname` from a Synopsys enabled shell. The location of files and programs specified in the scripts, may need to be adjusted according to the system used.

A file containing everything is available on the disc supplied with this thesis. From `http://caffrey.dk/leaksynth/leaksynth.tgz` can be downloaded a file containing everything *except* the VHDL code for the microprocessor. The md5sum of the file is:
`2ba6d40e635536d395df1a310487f5ba leaksynth.tgz`

Bibliography

[1] A. Abdollahi, F. Fallah, and M. Pedram. Leakage current reduction in sequential circuits by modifying the scan chains. In *Quality Electronic Design, 2003. Proceedings. Fourth International Symposium on*, pages 49– 54, March 2003.

[2] Robert W. Brodersen Anantha P. Chanandrakasan. *Low power digital CMOS design*. Kluwer Academic Publishers, 1995.

[3] Debashis Bhattacharya. Zentime^{TM}: A paradigm shift in automating next generation high performance cell-based designs. `http://www.zenasis.com/html/Zentime_flexcells_final-new.pdf`, 2004.

[4] Manjit Borah, Robert Michael Owens, and Mary Jane Irwin. Transistor sizing for minimizing power consumption of cmos circuits under delay constraint. In *Proceedings of the 1995 international symposium on Low power design*, pages 167–172. ACM Press, 1995.

[5] J. Adam Butts and Gurindar S. Sohi. A static power model for architects. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 191–201. ACM Press, 2000.

[6] Inc. Cadence Design Systems. Empowering design for quality of silicon, cadence encounter low-power design flow. `http://www.cadence.com/whitepapers/lowpower_wp.pdf`, 2004.

[7] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu. New paradigm of predictive mosfet and interconnect modeling for early circuit design. In *Proc. of IEEE CICC*, pages 201–204, June 2000.

[8] Sequence Design. Physicalstudio employs leakage power reduction. `http://www.sequencedesign.com/images/success_stories/nvidia_ss.pdf`, 2004.

[9] Duane Galbi and Anthony Jarvis. Understanding your cell library: Cell characterization for the masses. SNUG, 1999.

[10] Martin Hans. Architectural aspects of design for low static power consumption. Master's thesis, Informatics and Mathematical Modeling, Technical University of Denmark, 2004.

[11] Jacob Gregers Hansen. Design of CMOS cell libraries for minimal leakage currents. Master's thesis, Informatics and Mathematical Modeling, Technical University of Denmark, 2004.

[12] Masanori Hashimoto and Hidetoshi Onodera. Post-layout transistor sizing for power reduction in cell-based design. In *Proceedings of the 2001 conference on Asia South Pacific design automation*, pages 359–365. ACM Press, 2001.

[13] Sasan Iman and Massoud Pedram. Pose: power optimization and synthesis environment. In *Proceedings of the 33rd annual conference on Design automation*, pages 21–26. ACM Press, 1996.

[14] IN2FAB. Isis-power solves sub 0.13um leakage problem. `http://www.in2fab.com/isis_power.htm`.

[15] Synopsys Inc. Synopsys products – liberty. `http://www.synopsys.com/partners/tapin/lib_info.html`.

[16] Ali Keshavarzi, Kaushik Roy, and Charles F. Hawkins. Intrinsic leakage in low-power deep submicron cmos ics. In *Proceedings of the IEEE International Test Conference*, pages 146–155. IEEE Computer Society, 1997.

[17] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Moore's law meets static power. In *IEEE Computer Special Issue on Power- and Temperature-Aware Computing*, pages 68 – 75, December 2003.

[18] Uming Ko and P.T. Balsara. Short-circuit power driven gate sizing technique for reducing power dissipation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, pages 450 –455, 1995.

[19] Krzysztof A. Kozminski. Optimization of leakage power with primetime. SNUG, 2004.

[20] P. Larsson-Edefors, H. Eriksson, D. Eckerbert, and A. Alvandpour. Low-power design of delay-constrained circuits using dual-vt process technology. In *PATMOS*, 2001.

[21] Bill Lin and Hudo De Man. Low-power driven technology mapping under timing constraints. In *Proceedings of 1993 IEEE International Conference on Computer Design ICCD'93*, pages 421 –427, 1993.

[22] Inc. Magma Design Automation. Magma announces blast power for power optimization and management. `http://www.magma-da.com/c/@ZRETQAW5XzSUg/Pages/PRBlastPower.html`, 2004.

[23] Giovanni De Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill, Inc., 1994.

[24] Saibal Mukhopadhyay, Arijit Raychowdhury, and Kaushik Roy. Accurate estimation of total leakage current in scaled cmos logic circuits based on compact current modeling. In *Proceedings of the 40th conference on Design automation*, pages 169–174. ACM Press, 2003.

[25] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-v power supply high-speed digital circuit technology with multithreshold-voltage cmos. *Solid-State Circuits, IEEE Journal of*, 30:847–854, 1995.

[26] Kamram Eshraghian Neil N. E. Weste. *Principles of CMOS VLSI design – A systems perspective.* Addison-Wesley, second edition, 1993.

[27] Intel Research. Intel's high-k/metal gate announcement. `ftp://download.intel.com/research/silicon/HighK-MetalGate-PressFoils-final.pdf`, November 2003.

[28] Kaushik Roy, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. In *Proceedings of the IEEE*, volume 91, February 2003.

[29] Keith Sabine. A methodology for minimizing leakage current. `http://www.eedesign.com/showArticle.jhtml?articleID=16502300`.

[30] Debasis Samanta and Ajit Pal. Optimal dual -vt assignment for low-voltage energy-constrained cmos circuits. In *Proceedings of the 2002 conference on Asia South Pacific design automation/VLSI Design*, page 193. IEEE Computer Society, 2002.

[31] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sis: A system for sequential circuit synthesis. Technical report, U.C. Berkeley, May 1992.

[32] Jens Sparsø. Digital design and computer organization, mini-mips design project, 2002.

[33] STMicroelectronics. *CORELIB8DHS HCMOS8D 3.1 User's Manual*, september 2001.

[34] STMicroelectronics. *CORELIB8DLL HCMOS8D 3.1 User's Manual*, september 2001.

[35] Synopsys Inc. *Design Compiler Reference Manual: Constraints and Timing*, v-2003.12 edition, December 2003.

[36] The MOSIS Service. MOSIS Scalable CMOS layout rules. `http://www.mosis.org/Technical/Layermaps/lm-scmos_scn6m.html`.

[37] Vivek Tiwari, Pranav Ashar, and Sharad Malik. Technology mapping for low power in logic synthesis. *Integration, the VLSI Journal*, pages 243–268, 1996.

[38] UC Berkeley Device Group. Berkeley Short-channel IGFET Model. `http://www-device.eecs.berkeley.edu/~bsim3`.

[39] UC Berkeley Device Group. Predictive Technology Model. `http://www-device.eecs.berkeley.edu/~ptm`.

[40] Liqiong Wei, Zhanping Chen, Mark Johnson, Kaushik Roy, and Vivek De. Design and optimization of low voltage high performance dual threshold cmos circuits. In *Proceedings of the 35th annual conference on Design automation*, pages 489–494, 1998.

[41] Liqiong Wei, Zhanping Chen, Kaushik Roy, Mark C. Johnson, Yibin Ye, and Vivek K. De. Design and optimization of dual-threshold circuits for low-voltage low-power applications. *IEEE Trans. Very Large Scale Integr. Syst.*, 7(1):16–24, 1999.

[42] Liqiong Wei, Zhanping Chen, Kaushik Roy, Yibin Ye, and Vivek De. Mixed-vth (mvt) cmos circuit design methodology for low power applications. In *Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 430–435. ACM Press, 1999.

[43] Shyh-Chyi Wong, Gwo-Yann Lee, and Dye-Jyun Ma. Modeling of interconnect capacitance, delay, and crosstalk in vlsi. 13(1), 2000.

[44] Xiaodong Zhang. High performance, low leakage design using power compiler and multi-vt libraries. SNUG, 2003.