Leakage Aware Digital Design Optimization for Minimal Total Power Consumption in Nanometer CMOS Technologies

by

Schuster Christian

A thesis submitted to the Faculty of Science of the University of Neuchâtel, in conformity with the requirements for the degree of Doctor of Science

Institute of Microtechnology
University of Neuchâtel
Switzerland

21 March 2007

This thesis was typeset with LaTeX 2_ε by the author

Université de Neuchâtel

FACULTE DES SCIENCES
Secrétariat-Décanat de la faculté
- Rue Emile-Argand 11
- CP 158
- CH-2009 Neuchâtel

IMPRIMATUR POUR LA THESE

Leakage aware digital design optimization for minimal total power consumption in nanometer CMOS technologies

Christian SCHUSTER

UNIVERSITE DE NEUCHATEL

FACULTE DES SCIENCES

La Faculté des sciences de l'Université de Neuchâtel,
sur le rapport des membres du jury

MM. P.-A. Farine (directeur de thèse),
C. Piguet (co-directeur de thèse, CSEM, Neuchâtel),
S. Tanner, J.-L. Nagel (CSEM, Neuchâtel)
et M. Belleville (CEA-LETI, Grenoble F)

autorise l'impression de la présente thèse.

Neuchâtel, le 27 mars 2007

Le doyen :
T. Ward

UNIVERSITE DE NEUCHATEL
FACULTE DES SCIENCES
Secrétariat-décanat de la faculté
Rue Emile-Argand 11 - CP 158
CH-2009 Neuchâtel

■ Téléphone : +41 32 718 21 00 ■ Fax : +41 32 718 21 03 ■ E-mail : secretariat.sciences@unine.ch ■ www.unine.ch/sciences

To Mario, Silvana
and Eliana

Abstract

Starting from deep submicron technologies ($< 0.13\mu m$), and even stronger in nanometer technologies, static power consumption, due to leaky "off" transistors, is becoming a non-negligible contributor to the total power dissipation. Under this condition, the total power optimization problem changes considerably. The high parallelization approach commonly used today to increase performances, will soon result in power inefficient designs. Indeed, the static power consumption of the large number of rarely used transistors will highly penalize the total power consumption.

The purpose of this thesis is to investigate the influence of static power on the design methodologies for low power. In particular, the effects of architectural as well as technology modifications are explored. The use of technology as an optimization parameter has become possible in recent technologies. In fact, they offer different threshold voltages, each one showing a different trade-off between speed and leakage current.

In this work, two different frameworks are considered. In the first one, both the supply voltage and the transistor threshold voltage are freely tunable parameters. This is the most general case and corresponds to the situation where the designer has the largest freedom. In the latter framework, we assume that the designer cannot change the supply voltage nor the transistor threshold voltage and they are hence considered constants. This case corresponds to the most common one, where the designer has a supply voltage and a technology type (and hence a threshold voltage) fixed by the application and by the devices the circuit has to interface. In both cases, lot of efforts have been put to the development of a handy way to rapidly estimate the total power consumption and consequently easily compare different architectural/technology variants at the early stages of development.

Examples, based on multipliers, are used extensively in the whole thesis and, at the end, the presented theory is applied to a real circuit implemented in a 90nm technology by ST Microelectronics. Measurements show a very large variability of the static power over 16 dies manufactured on the same wafer. For instance, the highest static power consumption at nominal condition (Vdd=1V, f=62.5MHz) over the lowest one corresponds to more than a factor of 2.5. Measured data also report multipliers able to work at 210mV for a frequency of 1MHz!

Keywords

Low power digital design, static power, leakage current, dynamic power, very low supply voltage, multiplier, architecture, 90nm, CMOS nanometer technology.

Mots clés

Circuit numérique à faible consommation, puissance statique, courant de fuite, puissance dynamique, tension d'alimentation très basse, multiplicateur, architecture, 90nm, technologie CMOS nanométrique.

Acknowledgements

During the last four years, an important number of people have contributed to my personal knowledge expansion and have helped me progressing in my thesis. I will try to acknowledge most of them, being difficult to extensively report everyone in a few lines. I apologize in advance for the missing ones.

I thank Prof P.-A. Farine who received me in his group and gave me the freedom and all the tools that I needed to successfully finish my work. I also thank my thesis co-director Prof. C. Piguet for his endless support and the large number of suggestions he gave me during our weekly meetings. He also provided a great effort in promoting my work outside the IMT walls. Moreover, I would like the acknowledge Dr. J.-L. Nagel for being my project leader for the first three years and for sharing his vast knowledge with me (besides sharing the office too). During the last year, my new project leader Dr. S. Tanner helped me to finalize my work and supported me in the integration and chip testing part of the project. Many thanks to Dr. M. Belleville too, who kindly accepted to be one of the jury experts.

I am also grateful to all my colleagues, who created a pleasant ambient in the group and helped me to master some not-so-easy-to-use tools in this work. Particularly, I would like to thank P. Stadelmann and D. Manetti for all kind of discussions, C. Robert for the help provided in the PCB design, R. Merz for the help in the use of GPIB based instruments with MATLAB, without forgetting J.-L. Nagel, P. Thoppay and M. Moridi for sharing the office with me.

Furthermore, I want to express my complete gratitude to my parents who permitted me to successfully end my studies and always motivated me to progress: "You can always fly higher, as long as you want it!".

Finally, I would like to acknowledge my wonderful wife for always being beside me, bearing with me, and accepting me as I am, with my merits and demerits as well as my various moods: Thank you very much!

This work has been supported by CSEM (Neuchâtel, Switzerland) and the Swiss National Science Foundation (SNSF, under grant 105619).

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Description	Unit
Vds	Transistor drain-to-source voltage	[V]
Vgs	Transistor gate-to-source voltage	[V]
Vbs	Transistor bulk-to-source voltage	[V]
Vdd	Power supply voltage	[V]
$Vth0$	Reference transistor threshold voltage	[V]
$Vth = Vth0 - \eta Vds - \gamma Vbs$	Effective threshold voltage	[V]
η	DIBL effect coefficient	
γ	Body bias effect coefficient	
n	Sub-threshold slope	
$Ut = k_b T/q$	Thermal potential	[V]
$k_b = 1.38\text{E-}23$	Boltzmann constant	[J/K]
T	Temperature	[K]
$q = 1.6\text{E-}19$	Elementary charge	[C]
I_{on}	Transistor on current	[A]
I_{off}	Transistor off current	[A]
I_0	Reference current	[A]
μ_0	Low field mobility	[cm^2/V/s]
μ_{eff}	Effective carriers mobility	[cm^2/V/s]
α	Alpha power law coefficient	
k_t	Delay proportional constant	
C_i	Capacitance of node i on the critical path	[F]
C	Average cell capacitance	[F]
LD	Logical depth	
a	Circuit activity	
N	Number of cells	
f	Circuit working frequency	[Hz]

Symbol	Description	Unit
χ	Intrinsic design delay relating Vdd to Vth	
$k_1 = Pdyn/Pstat$	Dynamic power over static power ratio	
t_{cout}	Full adder carry out delay	[s]
t_{sum}	Full adder sum delay	[s]
t_{dff}	Register delay	[s]
t_{dff_setup}	Register setup time	[s]
t_{FA}	Worst case full adder delay	[s]
t_{bk_adder}	Brent-kung adder delay	[s]

Chapter 1

Introduction

1.1 Motivations

Digital integrated circuits are found everywhere in modern life and many of them are embedded in mobile devices where limited power resource is available (e.g. mobile phones, watches, mobile computers, personal assistants, ...). To permit an usable battery runtime, such devices must be designed to consume the lowest possible power. Furthermore, low power is also very important for non-portable devices, too. Indeed, a reduced power consumption can highly decrease the packaging costs and highly increase the circuit reliability, which is tightly related to the circuit working temperature. For these reasons, low power design is now mandatory for all types of digital circuits.

	2006	2007	2008	2009	2010	2011	2012	2013	2014
Technology node [nm]	90	65	65	65	45	45	45	32	32
Printed gate length [nm]	48	42	38	34	30	27	24	21	19
Transistors Number [M]	193	386	386	386	773	773	773	1546	1546
Chip size [mm^2]	88	140	111	88	140	111	88	140	111
Voltage supply [V]	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6
Internal frequency [GHz]	6.7	9.2	10.9	12.3	15	17	20	22	28
Total power [W]	98	104	111	116	119	119	125	137	137

Table 1.1: The International Technology Roadmap for Semiconductors [1] (ITRS), update 2006 for low operating power, cost effective high volume MPU.

As shown in Table 1.1, the number of transistors per circuit will continue to increase as predicted by Moore's law [2], whereas the transistor sizes will continue to shrink. Despite a decreased supply voltage, the total power will continue to increase.

The reduction of the supply voltage is dictated by the need to maintain the electric field constant on the ever shrinking gate oxide. Unfortunately, to keep transistor speed

(proportional to the transistor "on" current) acceptable, the threshold voltage must be reduced too, which results in an exponential increase of the "off" transistor current, i.e. the current constantly flowing through the transistor even when it should be "non-conducting".

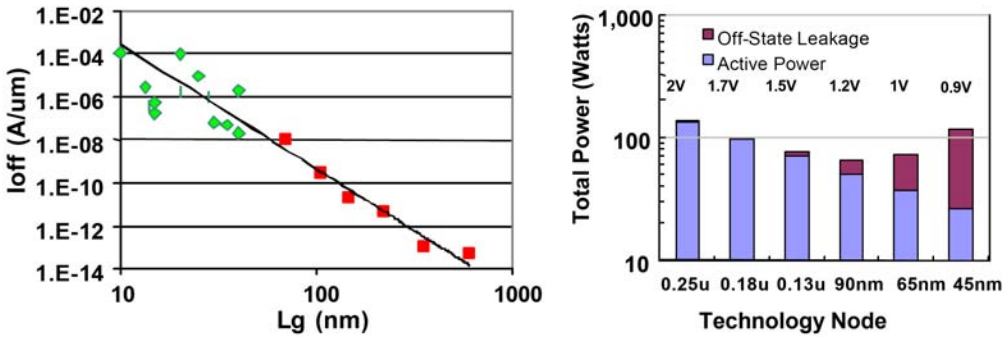

Figure 1.1: The left graph shows the transistor off-state current versus the gate length, squares indicate pre-production transistors and diamonds indicate research devices. The histogram on the right shows the total power as a function of technology node, for a fixed (30m) total transistor width. Source: Intel [3].

The left part of Fig. 1.1 shows the exponential increase of static power for real transistors of various sizes. By looking at the right part of Fig. 1.1, we can observe that this exponential increase of the static power can reach a point (starting on the 90nm node on the histogram) where it completely cancels the benefit of a reduced dynamic power (due to reduced capacitances and supply voltage).

Static consumption being now an important contributor to the total power, the design methodologies used in the past, based on dynamic power considerations only, are not effective any more and need to be reconsidered.

In the recent past years, static power was only relevant when the circuit was idle. This explains why many of static power reduction techniques are only applicable when blocks are unused. A typical example can be the Gated-Vdd approach [4] [5] [6], where a transistor is put between the real supply voltage and a virtual supply voltage, allowing to power off the unused blocks. However, Fig. 1.1 clearly shows that static power reduction should now be tackled in running mode, too.

Moreover, the large majority of the existing leakage reduction techniques apply at circuit and transistor level. Examples are:

- Multi Vth technology, with fast low Vth transistors on critical paths and slow high Vth transistors outside critical paths (MTCMOS) [7] [8] [9] [10]
- Electrical regulation of Vth (VTCMOS, SATS) [11] [12] [13]

- DTCMOS (Dynamic Vth) with transistor bodies connected to MOS gates [4] [14]

This thesis considers the reduction of the total power, i.e. dynamic plus static contributions, at a high level and during runtime. Basically, the low power consumption is searched through architectural and technology modifications in modern nanometer CMOS processes.

1.2 Thesis outline

In Chapter 2, the main sources of power consumption in CMOS technologies are reviewed, with an emphasis on the static ones. This permits to define the delay and power models in Chapter 3. These models are extensively used in the entire thesis and are hence considered as the foundation of this work. In Chapter 4, the 90nm CMOS technology from ST Microelectronics is described in details and the required model parameters are derived from SPICE-like simulations. Chapter 5 illustrates and describes the different multiplier architectures used in the various examples and case studies. In Chapter 6, the models for a total power consumption comparison in the case where the supply voltage and the threshold voltage are freely modifiable is derived. In particular, this chapter shows that, under such conditions, the total power consumption (for a given delay) presents a minimum. Its application to architectural modifications is reported in Chapter 7, followed by a similar analysis for technology modifications in Chapter 8. A different situation is considered in Chapter 9, where total power comparison models and charts are obtained for the case where the supply and threshold voltages are fixed. Finally, Chapter 10 reports the power consumptions of a circuit manufactured in a 90nm technology. This circuit is composed by 4 multipliers presenting different combinations of architecture and technology modifications. The thesis is closed by the conclusions in Chapter 11.

1.3 Contributions

The main contributions provided by this thesis are:

- Chapter 2-3: Collection and description of existing models for static power, dynamic power, total power and delay.
- Chapter 4: Complete characterization of the ST Microelectronics 90nm general purpose technology for all three available transistor types (LVT, SVT, HVT).

- Chapter 5: Detailed description and classification of thirteen multiplier architectures.
- Chapter 6: Development and analysis of closed-form equations for optimal total power, optimal supply voltage and optimal threshold voltage in a scenario where supply and threshold voltages are freely tunable.
- Chapter 7: Applications of the theory exposed in Chapter 6 to architecture modifications.
- Chapter 8: Applications of the theory exposed in Chapter 6 to technology modifications.
- Chapter 9: Development and application of easy-to-use equations and graphical tools for architectures comparison under fixed supply and threshold voltages condition.
- Chapter 10: Implementation, testing and analysis of a physical realisation of 4 multipliers representing different combinations of technology flavors and architectures.

Chapter 2

Sources of dissipation in CMOS transistors

Circuits designed before 1980 were mainly implemented in NMOS technology. Such devices presented the major inconvenient of a large current constantly flowing through the circuit even when no transitions occurred. To solve this issue, CMOS (Complementary Metal Oxide Semiconductor) technology was introduced. This seemed to be an ultimate solution for avoiding static power consumption. Thus, the only remaining sources of dissipation were the switched capacitance power (due to the charging/discharging of capacitance nodes) and the shortcut power (due to the current flowing from supply voltage (Vdd) to the ground (Vss) when switching), both only present during node transitions.

Unfortunately, the constant dimension reduction driven by Moore's law and the corresponding reduction of the supply voltage (needed to maintain the electric field on the transistor gates constant) yielded a huge increase of the static power consumption, taking it back to a non negligible source of consumption. The reasons why this occurred are mainly two. The former is the reduction of the threshold voltage imposed by the Vdd reduction in order to maintain the speed acceptable, and the latter is the new electrical effects originated by the reduction of the transistors geometrical dimensions, known under the name of short channel effects.

Starting from $0.13\mu m$ technology node (i.e. a technology with a minimal transistor size of $0.13\mu m$), the static power consumption cannot be neglected anymore and must be added to the dynamic power to correctly estimate the total power consumption.

In this chapter, the sources of dissipation in CMOS transistors are discussed in details, with a special focus on those contributing to the static consumption.

2.1 Dynamic consumption

Dynamic consumption is considered as the dissipation that occurs only when the circuit is active (i.e. internal circuit nodes are switching).

Two distinct contributions exist. The first is the so called switching energy and corresponds to the energy required to charge (and discharge) the node capacitances during transitions. The second is the energy dissipated during transitions due to the conductive path existing, for a short period of time, between the supply voltage and the ground. This effect is known as shortcut or short-circuit.

2.1.1 Switching energy

The energy consumed to charge (and then discharge) a capacitance C to a voltage V is given by[I] [7]:

$$\text{Capacitance switching energy} = CV^2 \tag{2.1}$$

This type of consumption can easily be reduced from a technology node to the other by reducing capacitance C and supply voltage V. Both reductions are effectively obtained in a new scaled technology; in fact, the supply voltage has to be reduced in order to avoid high electric fields on the transistor gates and the reduction of the transistor physical dimensions automatically results in reduced capacitances. This type of dissipation was the primary source of consumption in active mode for circuit implemented in technology larger than $0.13\mu m$ [15].

2.1.2 Shortcut energy

The second source of dynamic consumption arises from shortcut paths. Consider a CMOS inverter (Fig. 2.1) with the input node at zero. In this condition the NMOS transistor is off and the PMOS transistor is conducting. Now, if the input node potential increases from 0 to Vdd, the NMOS will start to conduct for $Vin > Vth\_nmos$ while the PMOS is still on, which result in a current flowing from Vdd to Vss. Then, when Vin acquires the potential $Vdd - Vth\_pmos$, the PMOS stops to conduct and the shortcut current vanishes too.

Clearly, this type of conduction only exists if the supply voltage Vdd is greater than the sum of the NMOS/PMOS sub-threshold voltages ($Vth\_nmos + Vth\_pmos$).

[I] This equation refers to the energy required to charge **and** discharge the capacitance, both processes contributing as $1/2CV^2$

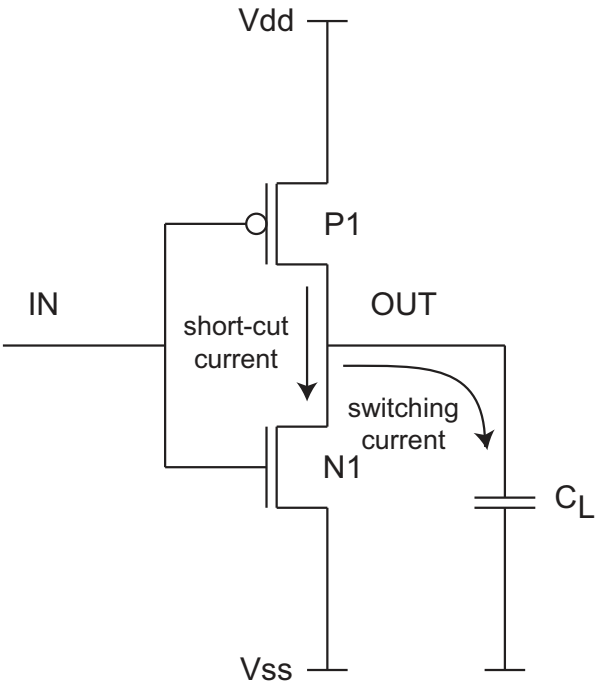

Figure 2.1: CMOS inverter

The energy dissipated during one transition can be expressed as [16]:

$$\text{Shortcut energy per transition} \propto (Vdd - Vth\_nmos - Vth\_pmos)^3 \cdot \tau \qquad (2.2)$$

With Vdd the supply voltage, Vth_nmos and Vth_pmos the threshold voltages for NMOS and PMOS, respectively and τ is the transition time, i.e. the period of time needed to sweep the input voltage from 0 to Vdd. More accurate models can be found in [17] [18] [19].

For well designed cells (i.e. with balanced rising and falling edges), the shortcut energy is in general much smaller than the switching energy. Moreover, for very low supply voltage designs, the value $Vdd - Vth\_nmos - Vth\_pmos$ can be very small. Additionally, the case where $Vdd < Vth\_nmos + Vth\_pmos$ will not present shortcut dissipation at all. For these reasons, in modern designs, shortcut power is often not considered or is simply included in the switching consumption by increasing the switching capacitance to an equivalent capacitance which incorporates the shortcut effect.

2.2 Static consumption

Contrary to the dynamic consumption, static power is defined as the consumption originated from currents constantly flowing from Vdd to ground. This means that even when the circuit is in idle mode (no transition occurs), power continues to be dissipated. For long channel transistors with high threshold voltage, this type of dissipation was completely negligible. Unfortunately, present and future technologies will suffer from high static power, which could even exceed the dynamic contribution in active mode. Hence, it is of uttermost importance to consider this type of dissipation in present and future design methodologies.

To understand the main sources of static dissipation, let us look at the structure of a transistor in CMOS technology. Fig. 2.2 shows 5 different leakage mechanisms that can be observed in a CMOS transistor (only the NMOS transistor is illustrated, as PMOS behaves exactly in the same way).

These mechanisms are:

(a) Sub-threshold current;

(b) Gate leakage current;

(c) Reverse-bias p-n junction current and band to band tunneling;

(d) Gate-Induced Drain Leakage (GIDL) current;

(e) Punchthrough current.

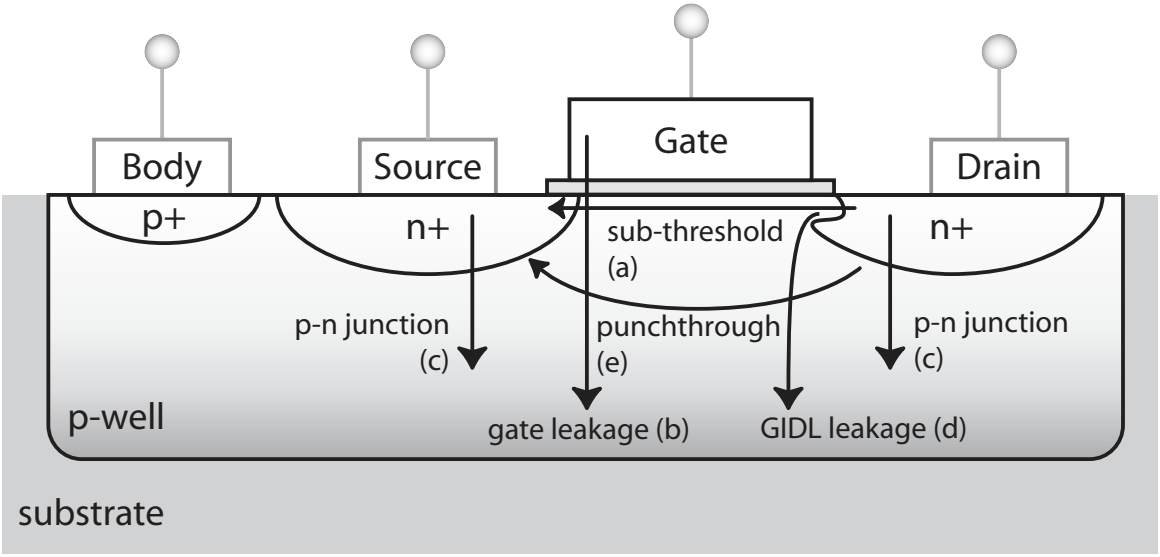

Figure 2.2: Sources of static power consumption in a NMOS transistor

2.2.1 Sub-threshold current

The most important leakage current is the sub-threshold one originated by the diffusion of minority carriers in a non conducting transistor ($V_{\text{gate}} - V_{\text{source}} < Vth$). Under this condition, the transistor is operating in weak inversion. The potential applied between drain and source creates a flow of the minority carriers on the surface of the channel. The equation describing this mechanism is [20] [21]:

$$I_{\text{sub-threshold}} = I_o \cdot e^{-\frac{Vth}{nUt}} \left(1 - e^{-\frac{Vds}{Ut}}\right) \approx I_o \cdot e^{-\frac{Vth}{nUt}} \tag{2.3}$$

With I_o the reference static current, Vth the threshold voltage, n the sub-threshold slope, $Ut(\equiv k_b T/q)$ the thermal potential and Vds the Drain-Source voltage.

Eq. (2.3) shows an exponential dependency of the sub-threshold current on the threshold voltage Vth. This is the reason why the low Vth characterizing recent technologies leads to large sub-threshold currents. Moreover, in typical digital designs, Vds is much larger than nUt, which leads to the approximation $1 - e^{-\frac{Vds}{Ut}} \approx 1$.

The value of Vth is not fixed for a given technology; in fact, it can be modulated through different effects like:

- **Drain Induced Barrier Lowering (DIBL) effect:** In short channel transistors, the potential on the drain contact modulates the threshold voltage by lowering the energy barrier at the surface of the channel. A schematic representation of this effect is illustrated in Fig. 2.3. For long channel transistors (L1), the potential in the channel is independent on the drain voltage ($Vd1$ and $Vd2$ show the same potential profile), whereas for short channels (L2), an increase of the drain voltage also reduces the barrier energy level in the channel, which can be modeled by a reduction of the threshold voltage. Ideally, the DIBL effect doesn't change the sub-threshold slope n. DIBL can be reduced by using high surface and channel doping and shallow source/drain junction depths.

- **Body effect:** The body effect appears when a potential difference is present between body (bulk) and source. This happens because bulk and source operate as a reverse biased p-n junction. By increasing the body potential in a NMOS or by decreasing it in a PMOS (forward biasing), the junction depletion reduces the channel potential and the sub-threshold leakage current increases. Similarly, a reduction of the body potential (lower than Vss for NMOS and higher than Vdd for PMOS, called reverse biasing) increases the channel potential, leading to a reduced sub-threshold leakage. It should be noted that for body-source potentials (Vbs) higher than 0.5 V the p-n junction starts to conduct as forward

biased diode, drawing very large current, which has to be avoided at all costs. Body effect is more pronounced for high bulk doping levels and decreases as substrate reverse bias increases. At $Vbs = 0$, the body effect sensitivity is equal to $(n - 1)$, with n the sub-threshold slope. The body effect can be modeled as a modification of the threshold voltage Vth.

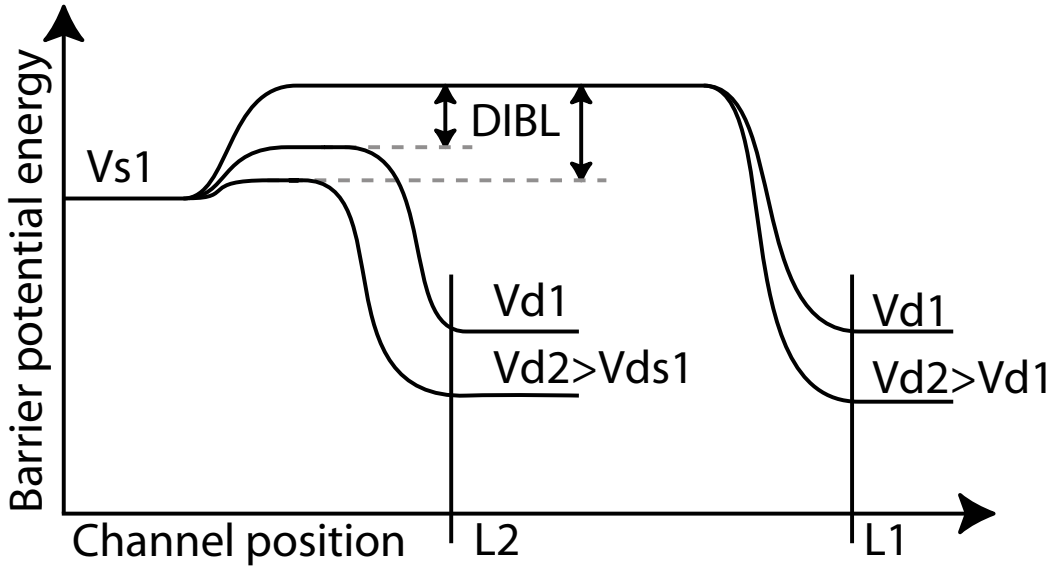

Figure 2.3: Effect of Drain Induced Barrier Lowering (DIBL) on short channel transistors

By considering the effects of DIBL and body bias, the threshold voltage can be expressed by [22] [23] [24]:

$$Vth = Vth0 - \eta Vds - \gamma Vbs \tag{2.4}$$

With $Vth0$ the reference threshold voltage for $Vds = Vbs = 0$, η (eta) the DIBL effect coefficient and γ (gamma, equal to n-1 for $Vbs = 0$) the linearized body effect coefficient.

By considering the described effects, the sub-threshold current can be expressed as:

$$I_{\text{sub-threshold}} = I_o \cdot e^{-\frac{Vth0 - \eta Vds - \gamma Vbs}{nUt}} \tag{2.5}$$

2.2.2 Gate leakage current

The transistor gate potential influences the charges in the channel by electrostatic effect: an accumulation of holes in the gate produces an accumulation of electrons at the surface of the channel, obtaining exactly the behavior of a capacitance with gate and channel as poles and the silicon oxide as dielectric. Ideally, no current should occur across the gate oxide, but practically some electrons are able to pass through the oxide, generating a gate current. The mechanisms behind this effect can be divided into two categories: oxide tunneling and hot carrier injection.

Oxide tunneling current

Tunneling through the gate oxide is primarily due to direct tunneling across very thin oxide layers (less than 3-4 nm). A model for this effect has been reported in [25] [26]:

$$I_{\text{gate}} = K_g \cdot W \left(\frac{V}{t_{ox}}\right)^2 e^{-\alpha_g t_{ox}/V} \tag{2.6}$$

With K_g and α_g[II] (alpha_gate) experimentally derived constants, W the width of the transistor, t_{ox} the gate oxide thickness and V the potential across the gate oxide. The previous equation clearly shows how the reduction of the oxide thickness exponentially increases the tunneling effect. An efficient way to reduce this source of leakage in future technologies is to use other insulators with a higher dielectric constant, resulting in a higher effective oxide thickness (i.e. the thickness of the silicon oxide that would show the same behavior as this high dielectric insulator). In this way, it should be possible to maintain the gate tunneling current to acceptable (i.e. negligible) levels. The main candidates to substitute the silicon oxide ($\kappa = 3.9$) are the hafnium oxide (HfO_2, $\kappa = 25$) and Hafnium silicate ($HfSiO_4$, $\kappa = 11$) [27].

Hot carrier injection

Due to the high electric field in the interface $Si - SiO_2$ (channel-oxide), electrons and holes can gain sufficient energy to enter into the gate oxide. Because the effective mass of the electrons, as well as their barrier height, is lower than the corresponding ones for holes, electrons injection is much more probable [28]. A reduction of the supply voltage will reduce the electric field on the gate, also reducing in this way the hot carrier injection.

[II] This α_g parameter has nothing to do with the α parameter used in the alpha power law model of the transistor on current, which is extensively used in this thesis

2.2.3 Reverse bias p-n junction leakage and band to band tunneling

In the normal transistor operation mode, the drain/source to well junctions are reverse biased. Under this condition, a small current exists due to the drift of carriers originated by the thermal electron-hole generation. Nevertheless, in advanced short channel MOS (where heavily doped and shallow junctions are used), such effects are masked by the dominating band-to-band tunneling.

Band to band tunneling happens on junctions with high electric field ($> 10^6 V/cm$) and is due to the direct tunneling of electrons from the band of valence of the p region to the band of conduction in the n region. Closed form equations describing this type of leakage exist [25] [29].

2.2.4 Gate-Induced Drain Leakage (GIDL)

In the overlapping zone between gate and drain, a high electric field can exist, leading to the generation of currents from drain to substrate. Consider a NMOS transistor; when a low gate potential is applied (Vg near zero volts or below), holes accumulate at the surface and create a region which is more heavily p doped than the substrate. If this happens while the drain is connected to a high potential (let say Vdd), the depletion layer near the drain becomes narrower. If this is important enough to invert the polarity of the n+ drain region under the gate, high field effects like band-to-band tunneling, avalanche multiplication and traps-assisted tunneling take place. As a consequence minority carriers are emitted in the drain region underneath the gate and pushed to the substrate due to the vertical electric field. All these effects are increased by a reduction of the gate oxide thickness.

This type of leakage is especially important for "relatively high" supply voltage circuits ($Vdd > 1.1$ V). Low power digital designs, with very low supply voltage (i.e. Vdd around 0.5V), are not heavily concerned by this type of leakage. More detail on GIDL effect can be found in [30] [28] [25].

The equivalent of the GIDL effect for a "high" source potential is called GISL (Gate-Induced Source Leakage). This effect is generally not considered because, in normal transistor operations, the source will show a low or zero potential compared to the bulk.

2.2.5 Punchthrough

With the physical dimensions reduction, the depletion layers of source and drain become nearer and nearer until they touch each other, originating punchthourgh currents. In submicron MOS transistors, implants at the substrate surface aiming Vth adjustment are used, forcing the punchthrough to occur deeper in the substrate. The size of the depletions directly depends on the Vds potential. Hence, low voltage design can prevent the generation of punchthrough currents [31] [25].

2.3 Summary

In deep sub-micron and nanometer technologies, the dynamic power consumption is no longer the only relevant source of power dissipation. In fact, present and future technologies will be characterized by large static power consumption coming from different leakage sources. In this chapter, the principal ones have been explained. However, it is important to observe that, depending on the transistor polarization, only a part of the described mechanisms occur. All realistic combinations of polarization are shown in Table 2.1 for a NMOS transistor.

Vg	Vd	Vs	Sub-threshold	Gate leakage	p-n junction	GID/SL	Punchtrough
0	0	0	NO	NO	NO	NO	NO
0	0	1	YES	NO	YES	GISL	YES
0	1	0	YES	NO	YES	GIDL	YES
0	1	1	NO	NO	YES	BOTH	NO
1	0	0	NO	YES	NO	NO	NO
1	1	1	NO	YES	YES	NO	NO

Table 2.1: Manifestation of specific leakage mechanism in a NMOS transistor depending on polarization

In a typical CMOS digital design, the NMOS transistor will have two modes of operation: Vg/Vd/Vs = 0/1/0 for the off transistor and Vg/Vd/Vs = 1/0/0 for a conducting transistor. When the transistor is on (conducting), the only mechanism that occurs is the gate leakage, whereas for an off transistor, sub-threshold, p-n junction, GIDL and punchthrough could be present. Nevertheless, the use of very low supply voltage (less than 1V) maintains the p-n junction and the punchthrough effects much lower compared to the sub-threshold one. Moreover, for gate potentials no lower than Vss for NMOS and not higher than Vdd for PMOS, the GIDL mechanism can also be neglected.

To summarize, the main sources of static power are the sub-threshold current for off transistors and gate leakage for conducting transistors.

		CLN90G	CL013GHP	CL013LVHP
	Transistor size [nm]	90	130	130
	Vdd [V]	1.0	1.2	1.0
INVD1	Sub-treshold current [nW]	5.14	0.56	6.93
	Gate leakage current [nW]	0.82	0.10	0.34
NAND2D1	Sub-treshold current [nW]	4.91	0.61	6.63
	Gate leakage current [nW]	1.40	0.15	0.56

Table 2.2: Gate and sub-threshold leakage current for three different TSMC technologies

Table 2.2 reports the sub-threshold and gate leakage power dissipation in 3 recent technologies. Values are reported for an inverter (INVD1) and a 2 input NAND gate (NAND2D1)[32]. We can observe that sub-threshold current remains the principal source of static power dissipation in deep sub-micron and nanometer technologies. The next generations could see an exponential increase in the gate leakage if silicon oxide is still used as insulator. Luckily, referring to [33], high dielectric constant oxide should be used starting from 2007. Intel also announced at the end of January 2007 [34] that high-k gate oxide will be used in their 45nm technology for the new generation of the Intel Core 2 Duo, Intel Core 2 Quad and Xeon families of multi-core processors.

Chapter 3

Delay and power models

3.1 Current models

As stated in Chapter 2, the main contribution to static power comes from sub-threshold currents flowing from drain to source in off transistors. In short channel transistors (L $< 1\mu m$), the voltage applied between drain an source also influences the channel conduction by a mechanism known as Drain Induced Barrier Lowering (DIBL).

$$I_{\text{off}} = I_o e^{-\frac{Vth0 - \eta Vdd - \gamma Vbs}{nUt}} = I_o e^{-\frac{Vth}{nUt}} \tag{3.1}$$

With I_o the reference static current, $Vth0$ the reference threshold voltage, Vth the modulated threshold voltage, η the DIBL effect coefficient, Vdd the supply voltage, γ the body effect coefficient, Vbs the body-source voltage, n the sub-threshold slope and Ut the thermal potential ($\equiv kT/q$).

The "on" current, i.e. the current flowing in a conducting transistor can be approximated by the following formula [35] [36] [37] [38]:

$$I_{\text{on}} = I_o \left(\frac{e}{\alpha n Ut}\right)^{\alpha} (V_{dd} - V_{th})^{\alpha} \tag{3.2}$$

With I_o the reference static current, e the euler number, α the alpha power law coefficient, n the sub-threshold slope, Ut the thermal potential, Vdd the supply voltage and $Vth(\equiv Vth0 - \eta V - \gamma V_{bs})$ the effective threshold voltage.

This model is an empirical fitting equation that accounts for the carriers mobility reduction. According to [39], the parameter α can be related to mobility by:

$$\alpha = 1 + \frac{\mu_{\text{eff}}}{\mu_0} \tag{3.3}$$

With μ_{eff} the effective carriers mobility and μ_0 the low field mobility. Being $0 < \mu_{\text{eff}} \leq \mu_0$, the parameter α will always be included in the range [1;2]; with $\alpha = 2$ for long channel transistors.

Based on these equations, it is now possible to define the dynamic and static power consumption as well as delay models.

3.2 Power models

As illustrated in the previous chapter, the total power can be divided into two categories: dynamic and static power.

3.2.1 Dynamic power

Dynamic power is due to the dissipation during the capacitances charge/discharge process. The well known equation describing it is:

$$P_{\text{dyn}} = \left(\sum_{i}^{N} a_i C_i \right) f \cdot V_{dd}^2 = aCNfV_{dd}^2 \tag{3.4}$$

With a_i the switching probability per clock period of the node i, C_i is the capacitance of node i plus the internal cell capacitance driven by node i, f is the circuit frequency, Vdd the supply voltage, N the number of cells, a the average activity per cell better understood as the average number of switching cells over the number of total cells during a clock cycle and C is the equivalent capacitance defined as $\left(\sum_i a_i C_i\right) / aN$. Using the proposed definition of activity, only the transitions from 0 to 1 are considered.

The expression of aCN using average parameters must be treated carefully. First, the average activity on the net is considered the same as the average activity in the cells, moreover the equivalent capacitance C is only equal to the average cell capacitance (net + internal cell) when all cells present the same activity, which is practically never the case. Therefore, C depends on activity distribution. For this reason, every time the parameters aCN are used together in equations, they must be considered as $\sum_i a_i C_i$, rather than average activity times average capacitance times number of cells.

A second contribution to dynamic power comes from the shortcut dissipation due to current flowing from Vdd to Vss during node transition. As seen in Chapter 2, this contribution is inexistent for supply voltage Vdd smaller than NMOS plus PMOS threshold voltages, and is very small for Vdd near $Vthn + Vthp$. Moreover, the quick

transition time, typically present in current technologies, further reduces the shortcut dissipation. Thus, this source of dynamic power can simply be accounted by lumping this effect into the cell capacitance, which will increase slightly.

3.2.2 Static power

This new source of dissipation coming from non-ideal transistor behavior is particularly important in deep submicron technologies and can become the main contributor even in running mode. Moreover, this type of consumption is always present as long as the circuit is supplied. Hence, even when the circuit does nothing (idle mode), static power continues to be dissipated. For simplicity of the model, only the main contributor (i.e. sub-threshold current) is considered. For a detailed discussion on the others existing sources of static power consumption, please refer to Chapter 2.

Static power model is given by:

$$P_{\text{stat}} = V_{dd} \cdot \sum_{i}^{N} I_{\text{off}}(i) = N \cdot V_{dd} \cdot I_o e^{-\frac{Vth}{nUt}} = N \cdot V_{dd} \cdot I_o e^{-\frac{Vth0 - \eta Vdd - \gamma V_{bs}}{nUt}} \tag{3.5}$$

With N the number of cells, Vdd the supply voltage, I_o the cell reference current, n the sub-threshold slope, Ut the thermal potential, Vth the modulated threshold voltage, $Vth0$ the reference threshold voltage, η the DIBL coefficient and γ the body bias coefficient.

It is important to note that I_o in Eq. (3.5) is the average reference off-current per cell. This factor is different from the single transistor reference off-current, because complex cells present a modified I_o due to stack effect, different transistor sizing, etc. According to [40], the ratio $k_{\text{design}} = (I_o\text{cell})/(I_o\text{transistor})/(\#\text{ of transistors})$ is about 1.4 for flip-flops, 2.0 for latches, 1.2 for 6T RAM cells and 11 for static logic. We carried out the same calculation for few cells with a driving force of 2 in the STM 90nm SVT technology and our results show a k_{design} spanning over a slightly narrower range; in fact, we obtain a $k_{\text{design}}$of 7.3 for a NAND gate, 6.5 for AND gate, 2.5 for a flip-flop and 3.7 for a full adder. Nevertheless, this shows that the static power consumption per cell can vary from cell to cell. For this reason, power comparison using Eq. (3.5) requires that both circuits present the same type of cells (i.e. static logic) or a similar distribution of different cell types. Otherwise, a compensation factor should be used depending on the type of cells used.

3.2.3 Total power

Total power is defined as the sum of dynamic plus static consumption. Referring to the previous sub-chapters, the total power model is given by:

$$\begin{aligned} P_{\text{tot}} &= P_{\text{dyn}} + P_{\text{stat}} \\ &= aCNfV_{dd}^2 + N \cdot V_{dd} \cdot I_o e^{-\frac{Vth}{nUt}} \\ &= N \cdot V_{dd} \left(aCfV_{dd} + I_o e^{-\frac{Vth}{nUt}} \right) \end{aligned} \tag{3.6}$$

With N the number of cells, Vdd the supply voltage, a the circuit activity, C the equivalent capacitance, f the frequency, I_o the average off-current per cell, Vth the modulated threshold voltage, n the sub-threshold slope and Ut the thermal potential.

3.3 Delay models

All power related discussions are worthless if the circuit delay (related to performance) is not considered. The model retained here is the very common one, that considers the delay of a cell as the time needed to charge the load capacitance by a driving current. So, to charge a capacitance C to the potential V the number of electric charges needed is $Q = CV$. Considering that these charges are coming at the speed of I_{on} [A=C/s], it is easy to find that:

$$t_{\text{gate}} = k_t \frac{CV}{I_{\text{on}}} \tag{3.7}$$

With k_t a constant accounting for the fact that the driving current is not constant during the capacitance charge (the values of this constant for the technology flavors used in this thesis are 15.1 for LVT, 24.7 for SVT and 30.1 for HVT. These values were obtained by multiplying the delay of a NAND2x2 cell with I_{on} and then by dividing it by the driven capacitance and by the supply voltage). I_{on} is the on transistor current and its formulation is given by Eq. (3.2).

In a digital design, the maximal achievable frequency is the inverse of the sum of delays on the critical path. In a mathematical form it appears as:

$$(f_{\text{max}})^{-1} = t_{\text{critical path}} = k_t \sum_i^{LD} \frac{C_i \cdot V_{dd}}{I_{\text{on}}} = k_t C \frac{LD \cdot V_{dd}}{I_{\text{on}}} \tag{3.8}$$

With C_i the load capacitance i on the critical path, LD the logical depth defined as the number of cells forming the critical path, C the average critical path capacitance defined as $\sum_i C_i/LD$.

Combining Eq. (3.8) with Eq. (3.2) yields:

$$f_{\max} = \frac{I_o \cdot e^{\alpha}}{k_t \cdot C \cdot LD \cdot (\alpha n U t)^{\alpha}} \frac{(V_{dd} - V_{th})^{\alpha}}{V_{dd}} \tag{3.9}$$

In the previous equation, it is interesting to observe that a high I_o corresponding to a high leaky technology also corresponds to a high maximal frequency, thus underlining the tight relation between high performance and static dissipation.

3.4 Summary

In this chapter, equations for the dynamic and static power consumption as well as the circuit delay (corresponding to the maximal frequency) have been obtained starting from simple and well known expressions of the on and off currents of a CMOS transistor. These equations are the foundation for the theory presented in this thesis. The use of very simplified equations, as well as the exclusion of secondary effects like gate leakage, are voluntary. This is necessary in order to be able to work with analytical expressions or simple closed form approximations, which makes it possible to understand the influence of each single parameter on the lowest achievable total power consumption.

Chapter 4

Technology characterization

The equations in the Chapter 3 depend on a certain number of technology parameters that must be characterized for a given technology before the equations can be exploited. To be sure that they really match the models used in this work, every parameter have been estimated by fitting SPICE simulations curves to our models with the program Graphical Analysis v3.2. The obtained values can vary compared to the original SPICE parameters, because used models are different. Actually, our models (explained in previous chapters) are much simpler than the BSIM3.3 ones, which are what the provided SPICE libraries use. In this thesis, the technology of ST Microelectronics with a minimal size of 90nm has been chosen as reference. The advantage of this technology is that it is available for 3 different transistor types, corresponding to 3 different threshold voltages.

4.1 Parameters extraction methodology

The technology parameters required in this work are:

- n : the sub-threshold slope;
- η : the DIBL effect coefficient;
- α : the alpha power law coefficient;
- $Vth0$: the reference transistor threshold;
- γ : the body effect coefficient.

Each one of these parameters will be discussed in details in the following sections.

4.1.1 The sub-threshold slope n

The sub-threshold slope n is extracted from the simulation of $I_{ds}(Vgs)$. The schematic used to measure the I_{ds} current is reported in Fig. 4.1.

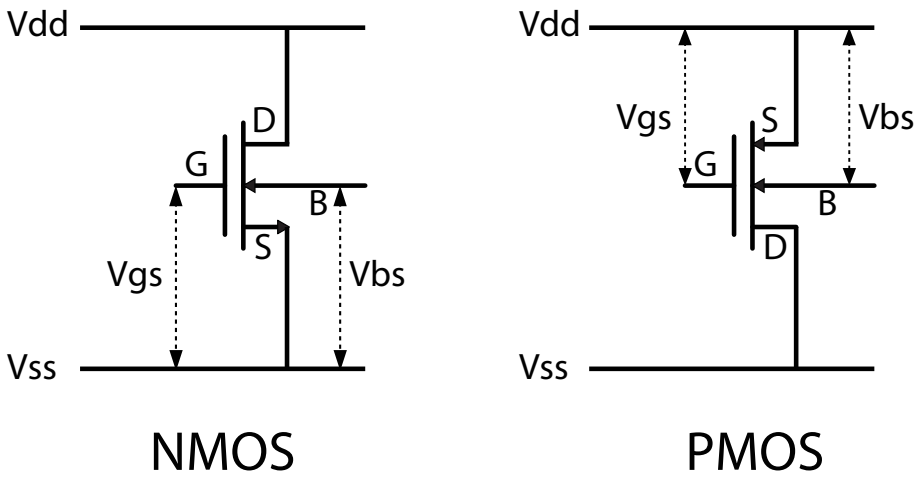

Figure 4.1: Schematic of the NMOS (left) and PMOS (right) transistors used for the extraction of the sub-threshold slope n . Transistor sizes are: $W_{nmos} = 0.51\mu m$, $W_{pmos} = 0.88\mu m$ and $L_{nmos} = L_{pmos} = 0.1\mu m$

The equation of the drain current in weak inversion is given by:

$$I_{ds}(Vgs) = I_o e^{\frac{Vgs-Vth}{nUt}} \tag{4.1}$$

Consequently, by considering the natural logarithm of the previous equation, the simulated curve should match the corresponding linear function:

$$ln\left(I_{ds}(Vgs)\right) = \frac{1}{nUt} \cdot Vgs + \left[ln(I_o) - \frac{V_{th}}{nUt}\right] \equiv m \cdot Vgs + b \tag{4.2}$$

Through a linear fitting, it is possible to extract the slope m of Eq. (4.2) to obtain $1/nUt$. Knowing the temperature used during the simulation, $Ut(\equiv k_b T/q)$ is also known (k_b = 1.38E-23, q = 1.6E-19).

As the values of n for the NMOS and the PMOS transistors can be different, the retained value will be their average.

The size of both NMOS and PMOS used in the SPICE simulations are the same than the corresponding ones in an inverter cell with a driving force of one.

4.1.2 The DIBL effect factor η

The extraction of the DIBL effect factor η is very similar to how n is obtained. The difference comes from the swept variable during simulation, which is now Vdd, while Vgs is set to $0V$, thus resulting in an off transistor. The corresponding equations are:

$$I_{off}(V_{dd}) = I_o e^{-\frac{V_{th0}-\eta V_{dd}}{nUt}} \tag{4.3}$$

$$ln\left(I_{off}(V_{dd})\right) = \frac{\eta}{nUt}\cdot V_{dd} + \left[ln(I_o) - \frac{V_{th0}}{nUt}\right] \equiv m \cdot V_{dd} + b \tag{4.4}$$

Once the slope η/nUt has been extracted, η is easily obtained, since $1/nUt$ was estimated in the previous section 4.1.1.

The static current I_{off} is measured as the supply current on a closed chain composed by an even number of inverters (10 in our case). In such a configuration, the circuit is in a stable condition and no node transitions occur. All inverters present a driving force of one.

4.1.3 The α factor and the reference threshold voltage $Vth0$

The parameter α (discussed in Chapter 3) and the reference threshold voltage $Vth0$ can both be estimated by fitting the delay equation (from Eq. (3.9)):

$$Delay(V_{dd}) \propto \frac{V_{dd}}{(V_{dd}-V_{th})^{\alpha}} = \frac{V_{dd}}{(V_{dd}(1+\eta)-V_{th0})^{\alpha}} \tag{4.5}$$

As η is a known parameter, a non-linear curve fitting on a circuit delay plotted in function of Vdd permits to determine the values of α and $Vth0$. Because both parameters are referred to the circuit delay (and this is the way the parameters will be used later), their values can be quite different from the single NMOS or PMOS ones defined by the manufacturer.

The delays are obtained by measuring the oscillating frequencies of a ring oscillator formed by 9 inverters with a driving force of one.

4.1.4 The body effect coefficient γ

The body effect coefficient γ models the first order influence of the body potential to the reference threshold voltage $Vth0$:

$$Vth(Vbs) = Vth0 - \gamma Vbs \tag{4.6}$$

The extracting methodology for this parameter is the same as for the DIBL effect coefficient η, but the measured parameter is $I_{off}(Vbs)$:

$$I_{off}(Vbs) = I_o e^{-\frac{V_{th0}-\eta V_{dd}-\gamma Vbs}{nUt}} \tag{4.7}$$

$$ln\left(I_{off}(Vbs)\right) = \frac{\gamma}{nUt}\cdot Vbs + \left[ln(I_o) - \frac{V_{th0}-\eta V_{dd}}{nUt}\right] \equiv m\cdot Vbs + b \tag{4.8}$$

A simple linear curve fitting on $ln(I_{off}(Vbs))$ is enough to determine $m = \gamma/nUT$. It is then easy to multiply the previous value by nUt to obtain γ.

Here too, the static current I_{off} is obtained by simulating a looped chain composed by an even number (10) of inverter with a driving force of one.

It is important to note that the body bias potential must be kept below 0.5V in the forward bias condition ($Vbs > 0$). Otherwise, the p-n junction between the body and the source will start to conduct as a forward-biased diode, creating an extremely large leakage current.

4.1.5 Remark on I_o

The parameter I_o representing the reference static current is also a technology related parameter, but its value cannot be extracted and used in a universal way as it is done for the other technology parameters. In fact, in this work, I_o is considered as the reference static power **per cell**. This means that the specific value is dependent on the cells used (as discussed in Chapter 3.2.2) and cannot simply be represented with an unique value. Except when stated differently, the average I_o of a circuit is estimated from cell nominal values of the static power in the following way:

$$I_o = \frac{\text{Total Nominal Static Power}}{Vdd\_nom\cdot N}\, e^{\frac{Vth}{nUt}} \tag{4.9}$$

With Vdd_nom the nominal supply voltage and N the number of cells.

4.2 STM 90nm technology

The STM 90nm is the most recent technology available at our laboratory and it presents the following main features:

- Designed for 1.0V ± 10% applications, with 1.8V/2.5V/3.3V IO's
- Shallow trench isolation, isolated P-Well (DNW) twin-tub, single poly CMOS process using a type <100> P-substrate

- 16Å gate oxide
- Cobalt silicide on junctions, polysilicon gates, lines, resistors on active and interconnect poly (N+ or P+)
- Dual Vth transistors
- IOs using 2.8nm or 5.0nm or 6.5nm gate oxide for 1.8V or 2.5V or 3.3V respectively
- 6 to 9 metal levels
- Damascene Copper for all metals
- Thick metal layer for power, clock, busses and major interconnect signal distribution, as well as for inductors in Analog/RF applications
- Tight pitch levels for routing on thin copper for lower metal layers
- Low K (< 3.0) inter-metal dielectric for thin metal layers

To extract the required parameters for each one of the 3 transistor flavors, the program ELDO version 6.1_1.1 from Mentor Graphics (SPICE-like simulator) has been used.

4.2.1 Low Vth Transistors (lvt)

The "Low Vth" transistor type is the fastest available flavor in the STM 90nm general purpose technology, and is used for applications where the speed is of primary importance. The disadvantage of this type of transistors is that, due to the low threshold voltage (Vth), the static power is very high.

The sub-threshold slope n

It is important to note that the linear fitting on Eq. (4.2) must be estimated on a region where the transistor is in the weak inversion mode (i.e. $Vdd < Vth$). Otherwise Eq. (4.2) is no longer valid and the alpha power law should be used instead to describe the transistor current. In our case the fitting range apply to $Vdd \in [0V; 0.2V]$. Moreover, the temperature was set to $27°C$, corresponding to an $Ut = 0.02588V$.

The following table summarizes the parameters extraction:

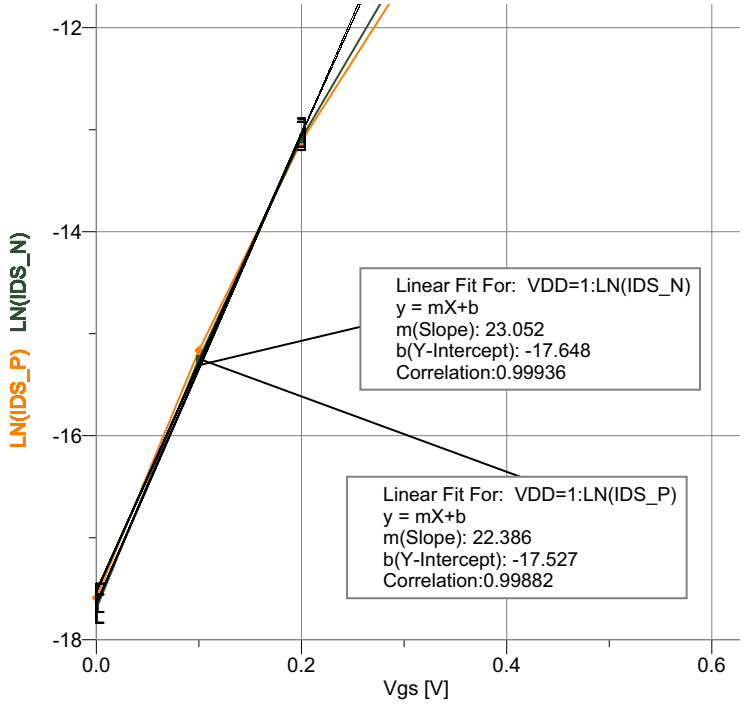

Figure 4.2: Linear fitting of $ln(I_{ds}(Vgs))$ for STM 90nm lvt

	$m = 1/nUt[V^{-1}]$	unified $1/nUT[V^{-1}]$	$Ut[V]$	n	unified n
NMOS	23.05	22.72	0.02588	1.68	1.70
PMOS	22.39			1.73	

Table 4.1: Results of the sub-threshold slope extraction for STM 90nm lvt

The DIBL effect factor η

The DIBL effect factor η is extracted from the curve $ln(I_{off}(Vdd))$. The static power is measured on a chain of 10 inverters, all with a driving force of one.

The results of the curve fitting in Fig. 4.3 are summarized in Table 4.2.

$m = \eta/nUt$	unified $1/nUt$ (from table 4.1)	η
1.98	22.72	0.087

Table 4.2: Results of the DIBL effect coefficient extraction for STM 90nm lvt

The α factor and the reference threshold voltage $Vth0$

The extraction of the α factor and of the reference threshold voltage $Vth0$ is done conjointly by fitting the non-linear equation (4.5) with a known value for η. The delays are obtained by measuring the oscillating frequency of a ring oscillator formed by 9 inverters with a driving force of one.

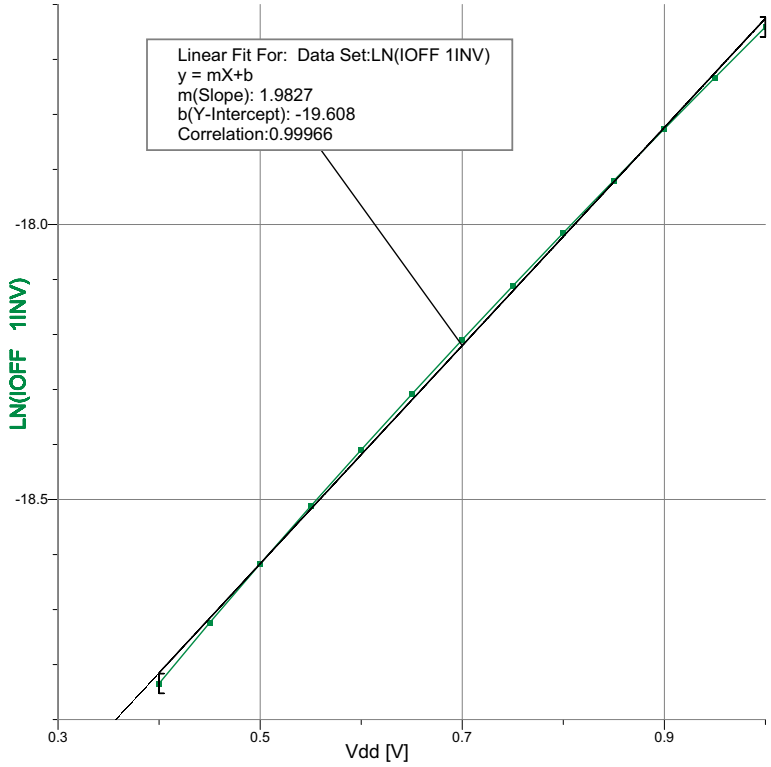

Figure 4.3: Linear fitting of $ln(I_{off}(Vdd))$ for 1 inverter (averaged over 10 inverters)

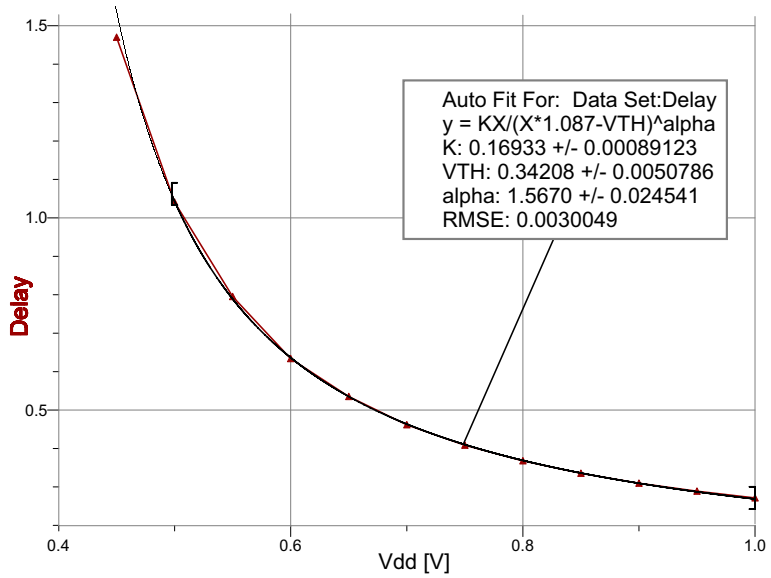

Figure 4.4: Fitting of delay vs. Vdd for STM 90nm lvt

Results, based on the fitting in figure 4.4, are presented in Table 4.3.

α	Vth0
1.56	0.342

Table 4.3: Results for the α factor and $Vth0$ for STM 90nm lvt

The body effect coefficient γ

The extraction of the body effect factor is achieved with a linear fitting on the curve $ln(I_{off}(Vbs))$. The static current is measured over 10 inverters connected in chain and the result has been divided by 10 to average the static current to one inverter.

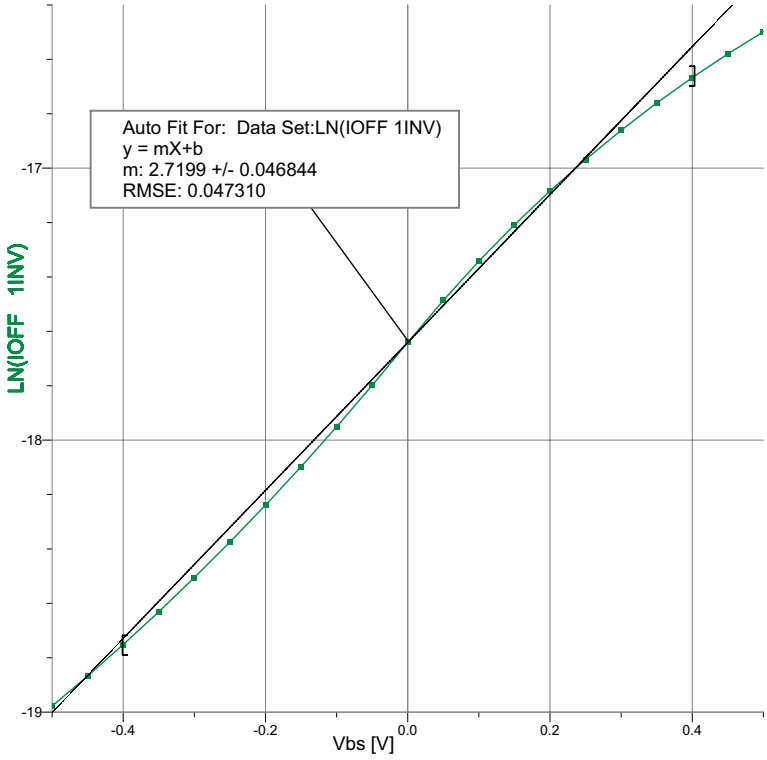

Figure 4.5: Linear fitting of $ln(I_{off}(Vbs))$ for 1 inverter (averaged over 10 inverters)

It should be noted that the γ is only a first order approximation of the body bias effect, because, as shown in Fig. 4.5, the curve is more like a square root function than a linear one.

Results are summarized by Table 4.4.

$m = \gamma/nUt$	unified 1/nUt (from table 4.1)	γ
2.72	22.72	0.12

Table 4.4: Results for the body effect coefficient for STM 90nm lvt

I_o for a 2 inputs NAND gate

Even if it is not possible to give here a unique I_o value for the technology, the value I_o for a 2 inputs NAND gate with a driving force of 2 is given as a reference.

I_o [μA]	30.9

Table 4.5: I_o for a NAND2x2 gate from the STM 90nm lvt technology

Summary of the lvt technology parameters

All the technology parameters for the lvt flavor are summarized by Table 4.6.

$Vth0$ [V]	α	n	η	γ	$I_o(NAND2x2)$ [μA]
0.342	1.56	1.70	0.087	0.12	30.9

Table 4.6: Technology parameters summary for the STM 90nm lvt

4.2.2 Standard Vth Transistors (svt)

The "Standard Vth" transistor type is an all-purpose flavor where delay and static power has been traded-off to match typical design requirements. The procedure used to characterize this technology variation is exactly the same as the one used for lvt. For the sake of simplicity, only the summary table is reported.

$Vth0$ [V]	α	$1/nUt[V^{-1}]$	n	η	γ	$I_o(NAND2x2)$ [μA]
0.353	1.65	26.30	1.47	0.060	0.14	26.0

Table 4.7: Technology parameters summary for the STM 90nm svt

4.2.3 High Vth Transistors (hvt)

The "High Vth" transistor type is a flavor especially optimized for extremely low static power consumption. Typical applications for this technology variation are circuit idle most of the time and/or where speed/performance are not of utmost importance. The procedure used to characterize this technology variation is exactly the same as the one used for lvt. For the sake of simplicity, only the summary table is reported.

$Vth0$ [V]	α	$1/nUt[V^{-1}]$	n	η	γ	$I_o(NAND2x2)$ [μA]
0.425	1.84	26.16	1.48	0.062	0.19	17.7

Table 4.8: Technology parameters summary for the STM 90nm hvt

4.3 Summary

In this chapter, the methodology used to extract the technology parameters has been presented. After an introduction of the general procedure, the parameters $Vth0, \alpha, n, \eta, \gamma$ and $I_o(NAND2x2)$ have been evaluated for all 3 transistor flavors available in the STM 90nm general purpose technology. In order to have an easy access to the extracted data, values are summarized in Table 4.9.

	$Vth0$ [V]	α	$1/nUt[V^{-1}]$	n	η	γ	$I_o(NAND2x2)$ [μA]
lvt	0.342	1.56	22.72	1.70	0.087	0.12	30.9
svt	0.353	1.65	26.30	1.47	0.060	0.14	26.0
hvt	0.425	1.84	26.16	1.48	0.062	0.19	17.7

Table 4.9: Technology parameters summary for the STM 90nm - Vdd = 1V

Chapter 5

Reference multiplier architectures

This chapter presents a set of 13 reference multipliers widely used in this thesis. The reason why we choose multipliers as reference comes from the fact that many possible implementations exist, each one with very different characteristics. The architectures proposed in this chapter can be divided into 3 families, each one containing more variations of the basic implementation.

The 3 families are:

1. **Ripple Carry Array (RCA):** This structure is based on a regular matrix of full adders; considered versions are:
   - basic;
   - 2 and 4 times parallel;
   - 2 and 4 times horizontal pipeline;
   - 2 and 4 times diagonal pipeline.
2. **Wallace:** This type of multiplier is based on a tree of full adder used as 3-to-2 compressors, considered versions are:
   - basic;
   - 2 and 4 times parallel.
3. **Sequential:** Here the multiplication is obtained by a sequential add and shift implementation, considered versions are:
   - basic;
   - sequential-wallace;
   - 2 times parallel.

5.1 Ripple Carry Array

The Ripple Carry Array multiplier (or RCA) is the most intuitive implementation for a multiplier. Its structure derives from the way we usually do multiplications by hand. That is, a sum of shifted partial products. A partial product (P_i) is the result of the multiplication of the multiplicand (A, first number to multiply) with one bit of the multiplier (B, second number to multiply). Practically, the multiplication by a bit is obtained by AND gates. The number of partial products will be equal to the size of B in bits. Mathematically, this can by written as (2^i represents the bits shift):

$$M = A * B = \sum_{i=0}^{size(B)-1} P_i \cdot 2^i = \sum_{i=0}^{size(B)-1} (A \textbf{ and } B_i) \cdot 2^i \tag{5.1}$$

In a physical implementation, the summation showed in Eq. (5.1) will be obtained by a series of full adder (FA), i.e. a 1 bit adder defined as:

$$\begin{aligned} S &= a \textbf{ xor } b \textbf{ xor } cin \\ Cout &= (a \textbf{ and } b) \textbf{ or } (a \textbf{ and } cin) \textbf{ or } (b \textbf{ and } cin) \end{aligned}$$

A graphical representation of a FA is provided in Fig. 5.1.

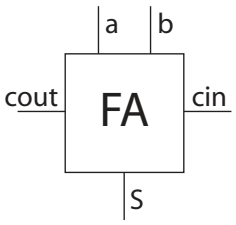

Figure 5.1: Full adder symbol

By implementing Eq. (5.1) directly, a multiplier known with the name of Ripple Carry Array multiplier (or RCA) can be constructed. Fig. 5.2 represents such implementation for N=8.

The first line of full adders in a RCA doesn't have to sum the partial products with the result of the precedent line because no precedent result exists. Hence, only partial products (AND gates) are generated by the synthesis tools. Moreover, the most right cell of each line has a fixed carry in of zero. Those cells can be simplified to an Half Adder (HA) i.e. an adder without carry in. The logical expressions of an HA are:

$$\begin{aligned} S &= a \textbf{ xor } b \\ Cout &= a \textbf{ and } b \end{aligned}$$

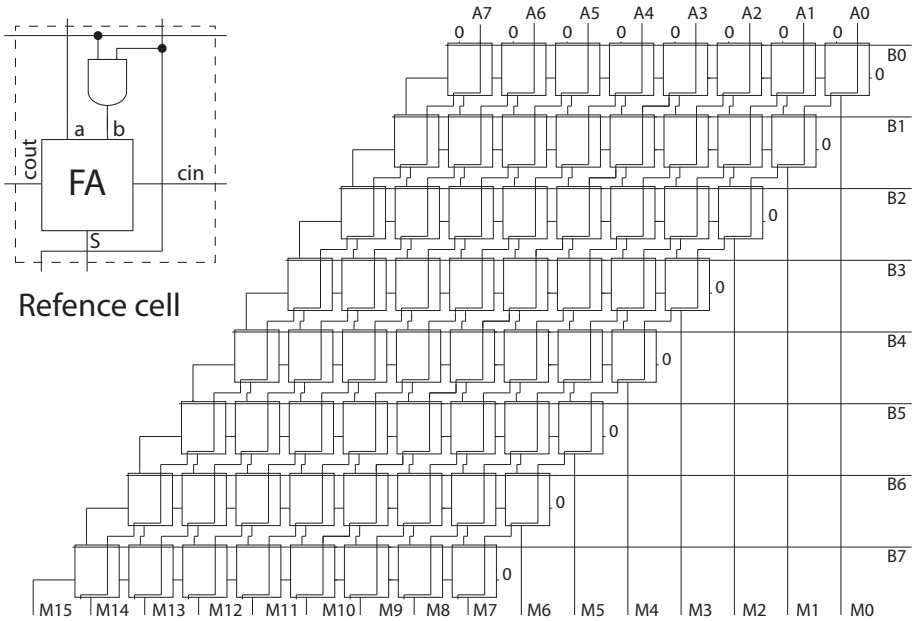

Figure 5.2: 8bit RCA multiplier

The FA has two characteristic delays. The first is the time that a signal needs to propagate from the inputs (a and b and cin) to the sum port (S). The second is the propagation delay for a signal going from the inputs (a and b and cin) to the carry out port ($Cout$). Fig. 5.3 shows one of the possible critical path that exists in such a multiplier.

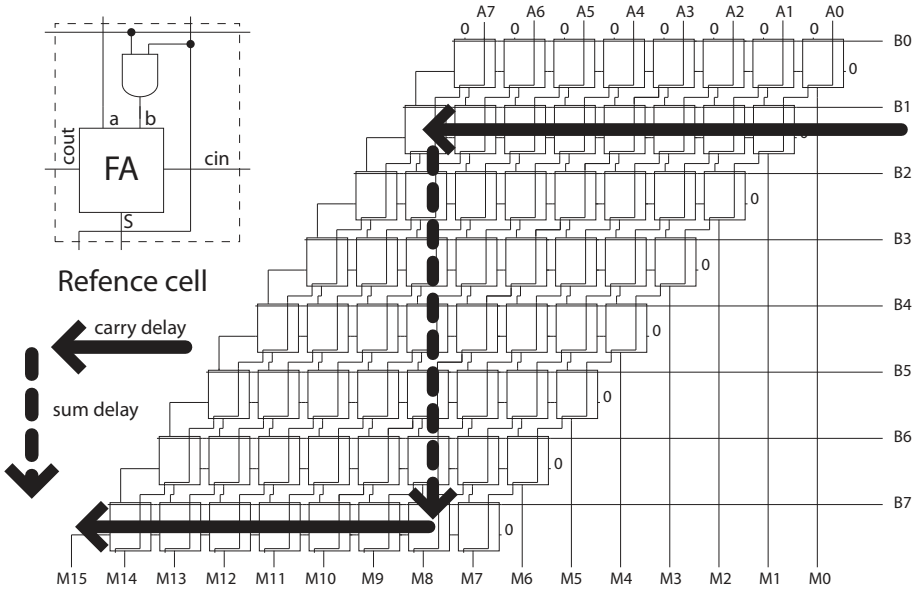

Figure 5.3: Critical path in a 8bit RCA multiplier

It is not surprising why, in Fig. 5.3, the critical path doesn't include the first line of full adders; indeed, it corresponds to simple AND gates for the generation of partial

products (because a and cin are zero), and they are executed in parallel with the partial products of the second line (corresponding to the bit B1).

The total delay for a RCA is given by :

$$t(\text{Basic RCA}) = (2 \cdot N - 2) \cdot t_{cout} + (N - 2) \cdot t_{sum} \tag{5.2}$$

With N the size of the multiplier, t_{cout} the carry out delay, t_{sum} the sum delay.

The structure presented in Fig. 5.2 and Fig. 5.3 are what we will call the "basic RCA" implementation. Others RCA implementations are explained hereafter.

5.1.1 RCA parallel variations

The first transformation of the RCA multiplier is the parallelization: the RCA multiplier is implemented twice (or more in general) and the data is multiplexed to a different multiplier at each clock period. The advantage of this architecture is that each multiplier has two (or as many as the number of instantiated blocks in general) clock periods to terminate the computation. So, the throughput is the same than for the non parallelized version, but the latency is bigger (corresponding to the number of blocks). Fig. 5.4 shows the structure of a 2 times parallelized multiplier.

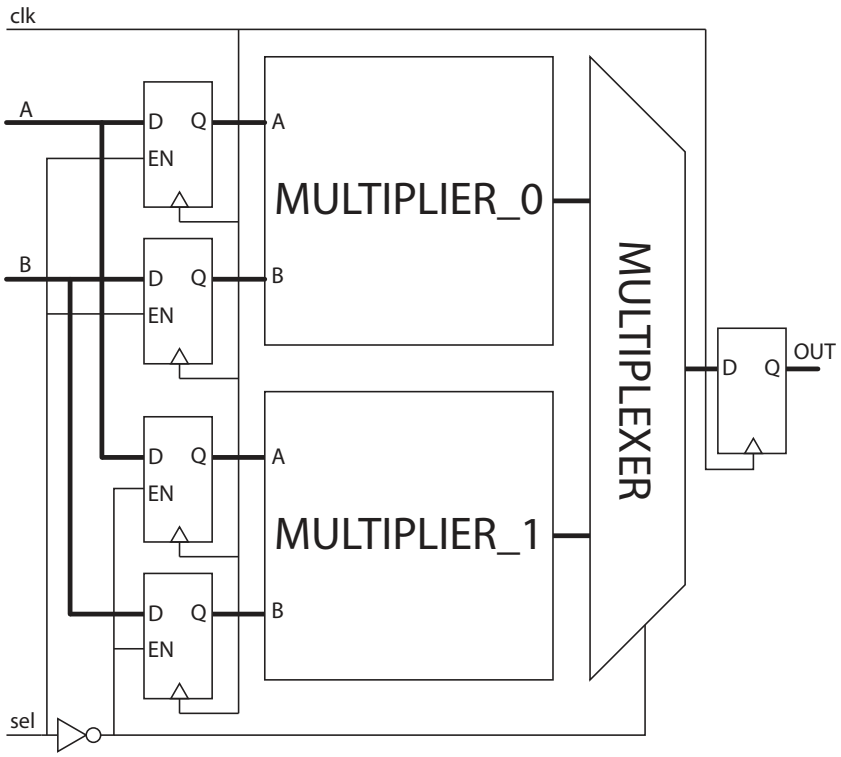

Figure 5.4: 2 times parallelized multiplier

The *sel* signal is used to select which multiplier will calculate the multiplication for the incoming data and it typically switches each clock cycles. The use of the input registers is required in order to latch the data at the input of the multipliers. In fact, each multiplier has now more than 1 clock cycle (corresponding to the degree of parallelization) to compute one multiplication, and the incoming data need to be stable over those clock cycles. Considering the throughput frequency as the reference clock, the effective logical depth, defined as the real logical depth divided by the number of clock cycles the signals have for propagating through it, is now reduced by the number of parallelizations.

The major drawback of the parallelization process is that the hardware is more than doubled (or N times for an N times parallel implementation). This also means that the static power is also more than doubled, while the dynamic power is only slightly increased due to the added registers and multiplexer.

5.1.2 RCA horizontal pipeline variations

The goal of pipelining is to reduce the critical path (logical depth) by inserting register banks in the design. This can be done in several ways with considerably different results. The more intuitive and easy manner to realize it is to "cut" the RCA horizontally in the middle of the structure. This can be imagined as two NxN/2 multipliers divided by a register bank as showed in Fig. 5.5.

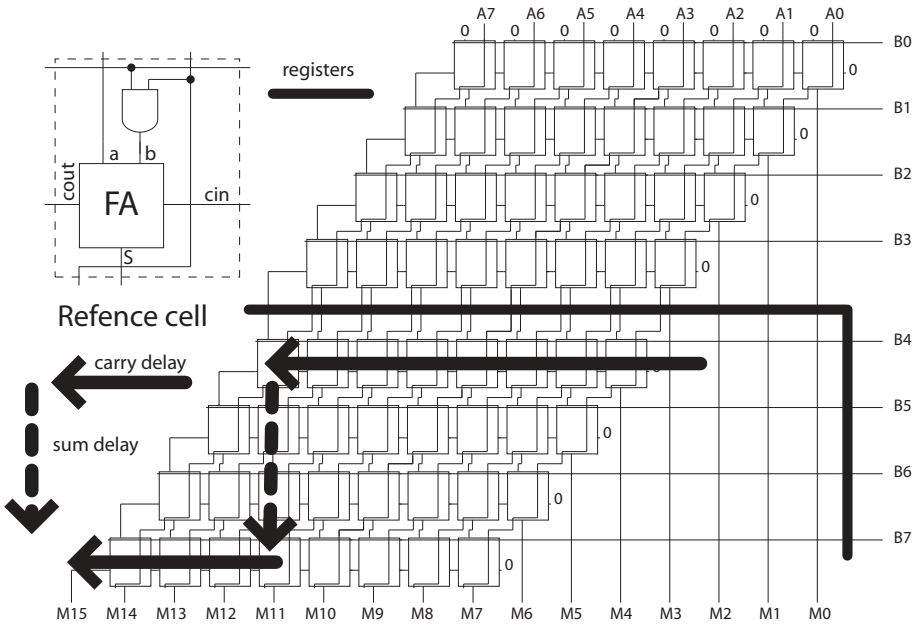

Figure 5.5: 2 stages horizontally pipelined 8 bit RCA

The number of registers needed to divide the multiplier in this way is easily ob-

tained from Fig. 5.5. Actually, all bits of A (N registers) plus all the result bits of the previous stage (N+N/2 registers) must be latched. Moreover, in order to maintain data synchronization, the most significant bits of B must be latched too (N/2 registers). Hence, the total overhead corresponds to 3N registers.

The critical path after such an architectural transformation is:

$$t(\text{Horizontal Pipeline}) = (3/2N - 1) \cdot t_{cout} + (1/2N - 1) \cdot t_{sum} + t_{dff} \tag{5.3}$$

With N the size of the multiplier, t_{cout} the carry out delay, t_{sum} the sum delay and t_{dff} the registers delay.

The "vertical delay" (corresponding to the t_{sum}) is effectively reduced by two, but the "horizontal delay" (related to t_{cout}) is just reduced by about 4/3. Additionally, the "clk to Q" delay of a register must be added. Hence, the global delay reduction compared to the non pipelined version is far from the expected (or hoped) value of 2.

A similar calculation can be done for a 4 stages pipeline, in this case the critical path delay will be of $(5/4N - 1)t_{cout} + (1/4N - 1)t_{sum} + t_{dff} + t_{dff\_setup}$.

It is important to remark that pipelining remains interesting only for a small number of stages (2 - 4); in fact, the quantity of needed registers rapidly grows for a large number of stages and the overhead is quickly non-negligible. In the case of a RCA multiplier with width N and S stages of pipeline, we have a register overhead of 3*N*(S-1). Just as an example, a 32 bit / 4 stages horizontal pipeline multiplier needs 288 extra flip-flops!

5.1.3 RCA diagonal pipeline variations

From a delay point of view, a better way to pipeline an RCA multiplier is to divide it in diagonal. This approach is less easy to code in a high level language compared to the horizontal split. In fact, the split parts cannot be considered anymore as multipliers of reduced size. An example on how to diagonally pipeline a RCA multiplier is illustrated in Fig. 5.6.

The critical path for a 2 stages diagonal pipeline is obtained by:

$$t(\text{Diagonal Pipeline}) = 3/4N \cdot t_{cout} + (3/4N - 1) \cdot t_{sum} + t_{dff} \tag{5.4}$$

With the diagonal pipeline implementation, the register overhead is slightly greater than the horizontal pipeline. In fact, for two stages pipeline, we can count N latches for the A bits, 3/4N latches for the B bits, 5/4N registers for the internal sum propagation

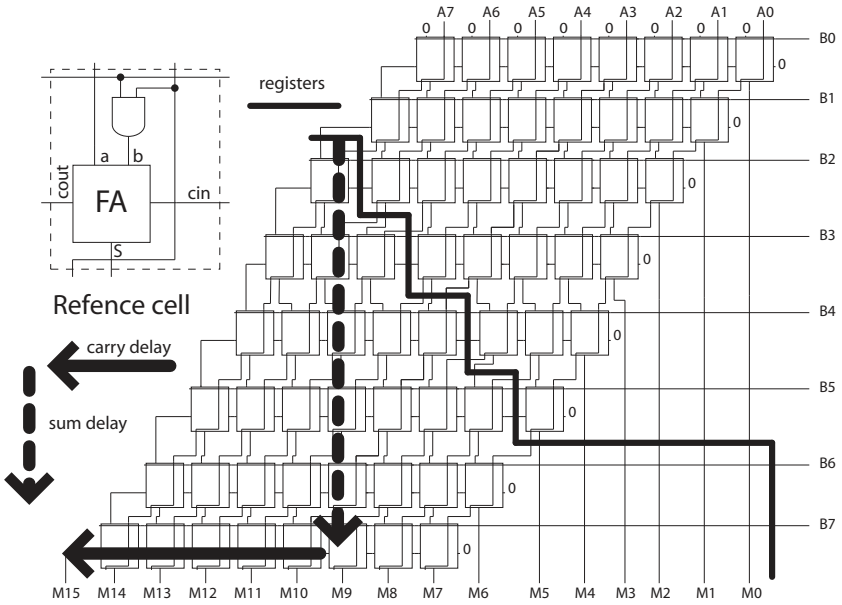

Figure 5.6: 2 stages diagonally pipelined 8bit RCA

and 1/2N registers for the carry propagation. All these contributions account for 3.5N registers. This value can be compared to the one for horizontal pipeline case where 3N registers were needed.

In a 4 stages diagonal pipeline version, the register overhead for each of the two new added banks is: 3/4N registers for the A bits, 3/8N registers for the B bits, 13/8N registers for internal sum propagation and 3/8N for the carry propagation. The total number of registers per stage is hence 25/8N. Summing all extra registers, the total overhead for a 4 stages diagonal pipeline is: 3.5N+2*(25/8N)=39/4N and the corresponding delay would be of $(3/4N-1)t_{sum}+t_{dff}+t_{dff\_setup}$. Just to compare with the horizontal pipeline version, a 32bit / 4 stages diagonal pipeline multiplier has 312 extra registers.

5.2 Wallace

The Wallace multiplier [41] [42] [43] is a very rapid and well balanced architecture. To achieve this efficiency, the partial products (i.e. $A \cdot B_i$, called P0-P7 in Fig. 5.8) are summed in parallel by using Carry Save Adders (CSA) [44]. A CSA (Fig. 5.7) is nothing else than a series of full adders disposed in a 3-2 compressor way. In a CSA, there exists no propagation delay between the full adders, consequently the total delay corresponds to the worst case delay of one FA. The main drawback of a CSA is that it doesn't return a unique sum but two vectors with a sum (S plus shifted C) equal to the sum of the three input vectors (x+y+z = S+2C).

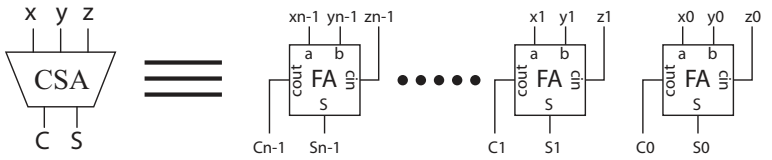

Figure 5.7: Internal implementation of a Carry Save Adder (CSA)

The structure of a Wallace multiplier is shown in Fig. 5.8 for a 8 bit version. The partial products P0-P7 are added 3 by 3 with CSAs until only two bit vectors remain (Sum and Carry). At this point, a fast final adder will sum them to obtain the result of the multiplication. The kind of final adder can vary from one implementation to the other. In the Wallace tree implementations presented in this thesis, a Brent-Kung [45] adder is used. The advantage of the Brent-Kung (bk) implementation is that it is very fast.

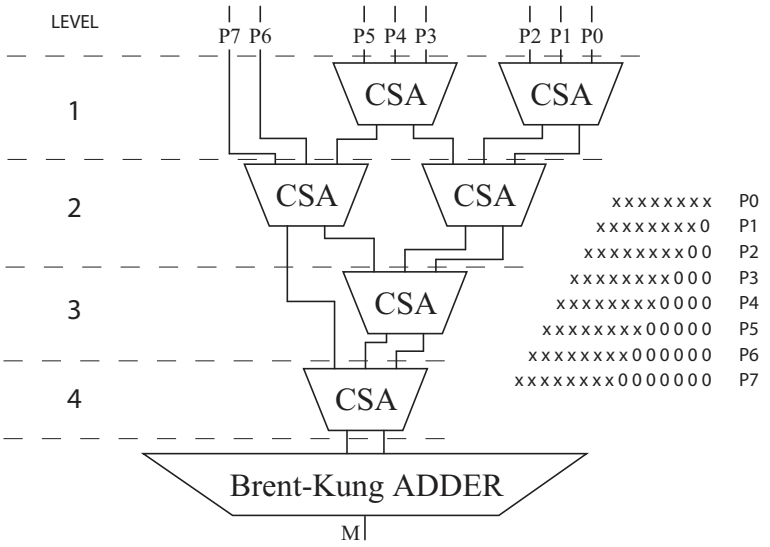

Figure 5.8: Wallace 8bit structure

The worst case delay for the multiplier tree (without the final adder) is equal to the number of levels times the worst case delay of a FA.

To calculate the total delay of the Wallace tree multiplier, the delay of the final adder (Brent-Kung type in this case) needs to be added.

$$t(\text{Basic Wallace}) \approx log_{1.5}(N) \cdot t_{FA} + t_{bk\ adder} \tag{5.5}$$

With N the bit width of the multiplier, t_{FA} the worst delay for a full adder and $t_{bk\ adder}$ the delay of the final bk adder, which is also dependent on the size of the multiplier.

Data width	Number of levels
8	4
16	6
32	8
64	10
128	12
N	$\approx log_{1.5}(N)$

Table 5.1: Number of CSA levels for some typical multiplier width

5.2.1 Wallace parallel versions

The parallelized versions of the Wallace multiplier are obtained exactly in the same way as for the RCA (Fig. 5.4). The description in Section 5.1.1 remains valid for the Wallace multiplier, too.

5.3 Sequential

The Sequential multiplier takes its name from the fact that this implementation uses several clock cycles to compute one multiplication by sequentially "adding and shifting" the previous partial result. The structure of such multiplier is illustrated in Fig. 5.9. The main advantage of this implementation is the compactness of the circuit. In fact, to calculate a 16 bit multiplication, only a 17 bit adder with some registers and a bit of control logic are required. On the other hand, the result will not be available until 16 clock cycles have taken place.

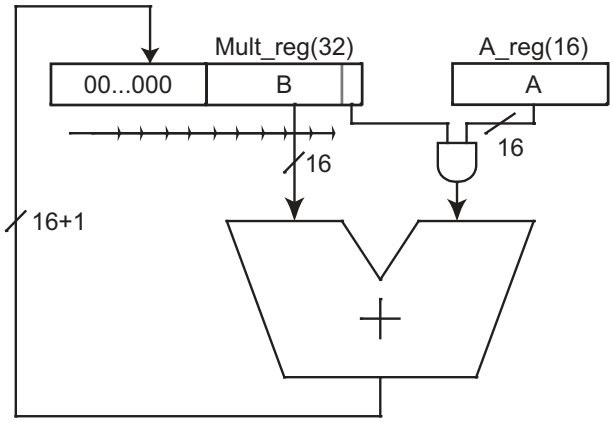

Figure 5.9: Sequential multiplier structure (16bit)

In the case of the present thesis, the adder used in the "add and shift" structure is a Brent-Kung type (bk), which is known for being a very rapid adder. Considering

the best case where the adder uses exactly all the clock cycle period, the total delay of a Sequential multiplier is given by:

$$t(\text{Sequential}) = N \cdot (t_{bk\ adder} + t_{AND} + t_{dff} + t_{dff\_setup}) \tag{5.6}$$

With N the multiplier bit width, $t_{bk\ adder}$ the Brent-Kung adder delay, t_{AND} the delay of the AND gate used to generate the partial products, t_{dff} the registers clock-to-Q delay and t_{dff_setup} the registers setup time.

In the case where the clock frequency is smaller than the maximal allowed one, the total delay will correspond to $N \cdot t_{clock}$.

5.3.1 Sequential-wallace

A special modification of the Sequential multiplier is what we call the Sequential-wallace (Fig. 5.10). The idea is to reduce the number of clock cycles required to compute one multiplication by adding partial multiplications rather than partial products. In the case of a 16 bit implementation (as reported in Fig. 5.10), a 4x16 bit Wallace multiplier is used to compute partial multiplications and then the results are summed sequentially. In this way we obtain a version between the Wallace (large area, small delay) and the Sequential (small area, large delay). Actually, for the proposed example, only 4 clock cycles are required per multiplication compared to the 16 cycles necessary for the basic Sequential implementation.

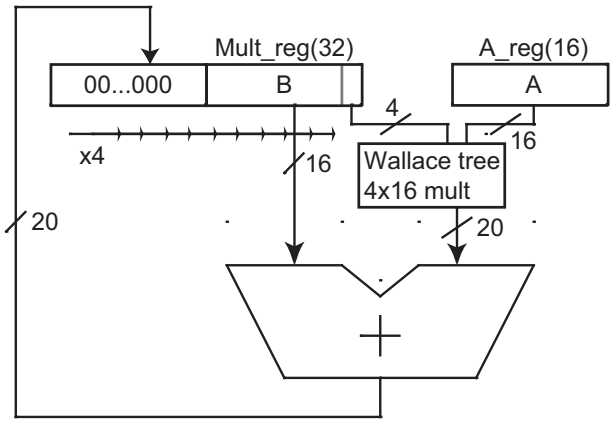

Figure 5.10: Sequential multiplier (16bit) with a 4x16 Wallace implementation

The delay of a Sequential-wallace multiplier is obtained by:

$$t(\text{Sequential-wallace}) = M \cdot (t_{bk\ adder} + t_{N/MxN\ Wallace} + t_{dff} + t_{dff\_setup}) \tag{5.7}$$

With $M(< N)$ the number of required cycles, N the bit width of the multiplier, $t_{bk\ adder}$ the Brent-Kung adder delay and $t_{N/MxN\ Wallace}$ the delay of the N/MxN Wallace multiplier.

5.3.2 Sequential parallel

The parallelized version of the Sequential multiplier is obtained exactly the same way as for the RCA and the Wallace (Fig. 5.4). The only difference is the *sel* pin that only switches once every N clock cycles, where N is the size of the multiplier.

5.4 Summary

In this chapter, 13 multiplier architectures have been discussed. These circuits are divided in 3 families (namely RCA, Wallace and Sequential) and they cover a large combination of delay, area and complexity. For this reason, they are well suited as reference circuits for the discussions presented further in this thesis. For commodity, the periods of the maximal throughput frequency as well as the cell count for each design are summarized in Table 5.2. The equations of the cell count for the Wallace implementations are obtained from [46] [47].

Name	Period of the maximal throughput frequency	Number of combinatorial cells	Regs[a]
RCA basic	$(2 \cdot N - 2) \cdot t_{cout} + (N-2) \cdot t_{sum}$	$(N-1)^2 \cdot \mathbf{FA} + N \cdot \mathbf{HA} + N^2 \cdot \mathbf{AND}$	$4N$
RCA parallel2	$(N-1) \cdot t_{cout} + (N/2-1) \cdot t_{sum} + t_{mux}$	$2(N-1)^2 \cdot \mathbf{FA} + 2N \cdot \mathbf{HA} + 2N^2 \cdot \mathbf{AND} + \mathbf{MUX2}$	$6N$
RCA parallel4	$(N/2-1/2) \cdot t_{cout} + (N/4-1/2) \cdot t_{sum} + t_{mux}$	$4(N-1)^2 \cdot \mathbf{FA} + 4N \cdot \mathbf{HA} + 4N^2 \cdot \mathbf{AND} + \mathbf{MUX4}$	$10N$
RCA horiz. pipeline 2	$(3/2N-1) \cdot t_{cout} + (1/2N-1) \cdot t_{sum} + t_{dff}$	$(N-1)^2 \cdot \mathbf{FA} + N \cdot \mathbf{HA} + N^2 \cdot \mathbf{AND}$	$7N$
RCA horiz. pipeline 4	$(5/4N-1)t_{cout} + (1/4N-1)t_{sum} + t_{dff} + t_{dff\_setup}$	$(N-1)^2 \cdot \mathbf{FA} + N \cdot \mathbf{HA} + N^2 \cdot \mathbf{AND}$	$13N$
RCA diag. pipeline 2	$3/4N \cdot t_{cout} + (3/4N-1) \cdot t_{sum} + t_{dff}$	$(N-1)^2 \cdot \mathbf{FA} + N \cdot \mathbf{HA} + N^2 \cdot \mathbf{AND}$	$7N$
RCA diag. pipeline 4	$(3/4N-1)t_{sum} + t_{dff} + t_{dff\_setup}$	$(N-1)^2 \cdot \mathbf{FA} + N \cdot \mathbf{HA} + N^2 \cdot \mathbf{AND}$	$13N$
Wallace basic	$\approx log_{1.5}(N) \cdot t_{FA} + t_{bk\ adder}$	$\approx (N^2-2N) \cdot \mathbf{FA} +$ "few" $\mathbf{HA} + N^2 \cdot \mathbf{AND}$	$4N$
Wallace parallel 2	$\approx 1/2(log_{1.5}(N) \cdot t_{FA} + t_{bk\ adder}) + t_{mux}$	$\approx 2(N^2-2N) \cdot \mathbf{FA} +$ "few" $\mathbf{HA} + 2N^2 \cdot \mathbf{AND} + \mathbf{MUX2}$	$6N$
Wallace parallel 4	$\approx 1/4(log_{1.5}(N) \cdot t_{FA} + t_{bk\ adder}) + t_{mux}$	$\approx 4(N^2-2N) \cdot \mathbf{FA} +$ "few" $\mathbf{HA} + 4N^2 \cdot \mathbf{AND} + \mathbf{MUX4}$	$10N$
Sequential basic	$N \cdot (t_{bk\ adder} + t_{AND} + t_{dff} + t_{dff\_setup})$	$N \cdot \mathbf{FA} + \mathbf{HA} + N \cdot \mathbf{AND}$	$5N$
Sequential-wallace	$M \cdot (t_{bk\ adder} + t_{N/MxN\ Wallace} + t_{dff} + t_{dff\_setup})$	$(N^2/M - N/M) \cdot \mathbf{FA} +$ "few" $\mathbf{HA} + N^2/M \cdot \mathbf{AND}$	$5N$
Sequential parallel 2	$N/2 \cdot (t_{bk\ adder} + t_{AND} + t_{dff} + t_{dff\_setup}) + t_{mux}$	$2N \cdot \mathbf{FA} + 2\mathbf{HA} + 2N \cdot \mathbf{AND} + \mathbf{MUX2}$	$8N$

Table 5.2: Summary of the multipliers delays and cell counts

[a] Including the input and output latching registers

Chapter 6

Total power comparison for free Vdd and free Vth

A very effective way to reduce the total power consumption in digital circuits is the reduction of the supply voltage Vdd. This approach is simple and easy to implement and it will simultaneously reduce dynamic power in a square way and static power linearly. Unfortunately, in this way, the performances or speed rapidly decrease. In order to avoid this, it is possible to re-establish the original performances by reducing the transistors threshold voltage Vth. The price for this is an exponential increase of the static power. For this reason, counterbalancing the reduction of the dynamic power with the increase of static power leads to a point in the (Vdd, Vth) space where, for a given delay, the total power presents a minimum. This chapter will discuss this minimum of the total power consumption and will derive an approximated formula for the total power at the optimal (Vdd, Vth) point.

6.1 Existence of a total power consumption optimum

To convince the reader of the existence of the minimum of the total power consumption, it is important to recall the power and delay equations reported in Chapter 3:

$$Ptot = Pdyn + Pstat = aCNfV_{dd}^2 + NV_{dd}I_0e^{-\frac{Vth}{nUt}} \tag{6.1}$$

$$f_{\max} = \frac{I_{\text{on}}}{k_t \cdot C \cdot LD \cdot V_{dd}} = \frac{I_0 \cdot e^{\alpha}}{k_t \cdot C \cdot LD \cdot (\alpha nUt)^{\alpha}} \frac{(V_{dd} - V_{th})^{\alpha}}{V_{dd}} \tag{6.2}$$

With a the activity factor, C the equivalent capacitance per cell, N the number of cells, f the working frequency, Vdd the supply voltage, I_0 the reference transistor current, Vth the transistor threshold current, n the sub-threshold slope, Ut the thermal potential, k_t the delay proportional constant, LD the logical depth and α the alpha power law coefficient.

If now we consider that the frequency $f_{\max}$ (called f from now on) is fixed and defined by the application, it is possible to rewrite Eq. (6.2) to obtain the formula tying Vdd and Vth together:

$$V_{th} = V_{dd} - \chi \cdot V_{dd}^{1/\alpha} \qquad \text{with: } \chi^{\alpha} = \frac{k_t \cdot C \cdot f \cdot LD}{I_0 \left(\frac{e}{\alpha n U t}\right)^{\alpha}} \tag{6.3}$$

The parameter χ in Eq. (6.3) is a very important one. This parameter ties together the supply voltage and the threshold voltage. Its value represents a kind of "global rapidness" accounting for both technology and architectural impacts. Actually, a large χ means a "slow" design, which can be due to a large logical depth or a slow technology or a combination of architectural and technology parameters. The presence of the working frequency in the equation of χ shows that the concept of slow or quick design is dependent on the desired working frequency. For instance, a design considered rapid for a working frequency of 1MHz, could be considered slow for a working frequency of 100MHz.

A graphical representation of Eq. (6.3) is given in Fig. 6.1. There, we can see that the reduction of the supply voltage requires a reduction of the threshold voltage too in order to maintain speed. Even if there exists an infinite number of couples (Vdd,Vth) showing the same performance, they don't present the same power consumption. In fact, while the reduction of the supply voltage Vdd reduces the dynamic power in a square way and reduces the static power linearly, the reduction of the threshold voltage Vth shows an exponential increase of the static power. Due to the exponential nature of this last dependency, the static power increase can rapidly cancel the benefit of the reduced supply voltage Vdd. Therefore, between all the combinations of (Vdd,Vth) guaranteeing the desired speed, only one couple will result in the lowest power consumption for a given architecture (Fig. 6.2). From now on, this working condition will be called optimal working point or ideal working point.

The location of this optimal working point and its associated total power consumption are tightly related to architectural and technology parameters. For instance, Fig. 6.2 illustrates the fact that reducing the activity factor allows a reduction of Ptot, whereas it tends to increase the optimal Vdd and Vth. As architectural modifications

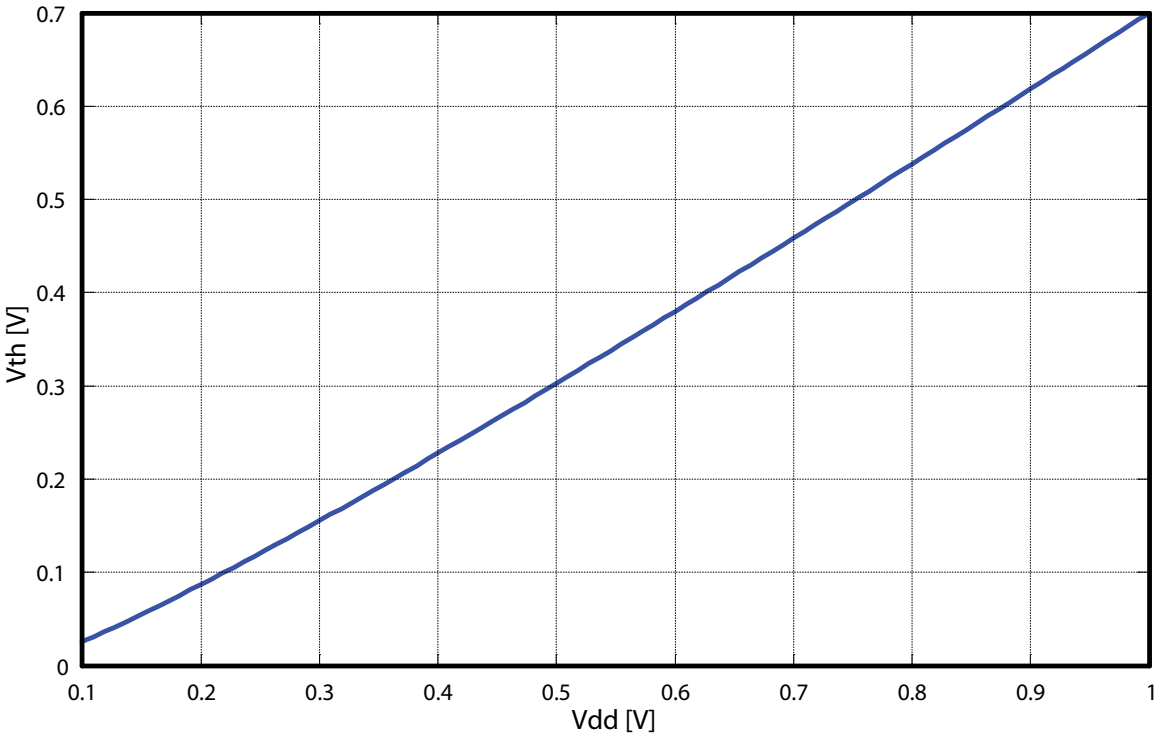

Figure 6.1: Relationship between Vdd and Vth for $\alpha = 1.65$ and $\chi = 0.3$

will change simultaneously several factors (not just the activity), it is necessary to develop a methodology to evaluate the influence of such transformations on the total power consumption (Ptot).

In related contributions ([48], [49], [50], [51], [52], [53], [54]), the authors preferred to seek for the minimum of the energy rather than the minimum of the total power as done in this work. From a mathematical point of view, looking for the minimum of the energy is slightly easier and the results are different from what we derive here. Indeed, they found that the minimum of total energy is most of the time located in the weak-inversion transistor region (optimal $Vdd <$ optimal Vth), which corresponds to very low performances logic.

6.2 Pdyn over Pstat ratio

Looking at the ratio Pdyn over Pstat at the optimal working point in Fig. 6.2, it is possible to observe that dynamic contribution still remain greater than the static one.

$$k1 = \left.\frac{Pdyn}{Pstat}\right|_{\text{optimum}} \tag{6.4}$$

This ratio (k1) is a measurement of the circuit usefulness. In fact, rarely used

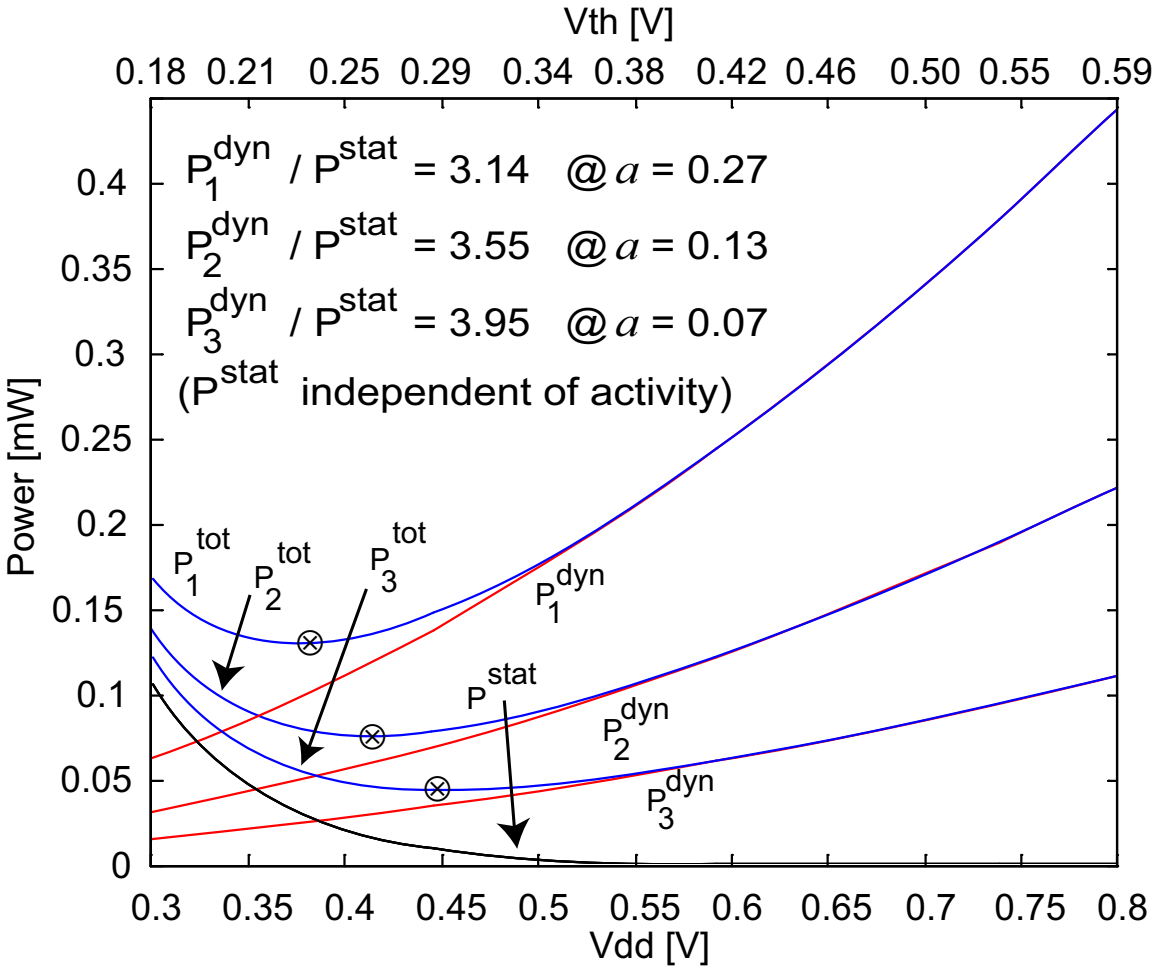

Figure 6.2: Total power consumption of a 16 bit Wallace multiplier in a STM 90nm technology (CMOS090-SVT, 100MHz) with freely modifiable Vdd and Vth. Three different circuit activities (a) are reported. The optimal working points are marked by a cross mark.

transistors will provide low k1 due to the high static consumption compared to the dynamic one. For this reason, it is better to have fewer transistors (less static power) working more actively (more dynamic power) than having lots of idle transistors that just increase the static power. In related works [55] [56], authors stated that this ratio should be equal to 1, whereas our experiences, based on many designs (like multipliers, FIR, shift registers, micro-processors, counters, ...) in deep sub-micron technologies ($0.18\mu m$, $0.13\mu m$ and 90nm), suggest that typical values of k1 are between 3 and 7.

6.2.1 k1 derivation

A precise calculation of k1 is possible and easy to obtain. In fact, k1 can be derived by searching the minimum of Ptot(Vdd) as:

$$\frac{\partial Ptot(V_{dd})}{\partial V_{dd}} = \frac{\partial Pdyn(V_{dd})}{\partial V_{dd}} + \frac{\partial Pstat(V_{dd})}{\partial V_{dd}} = 0 \tag{6.5}$$

The combination of Eq. (6.5) with Eq. (6.1) and Eq. (6.3) leads to:

$$k1 = \frac{(\alpha - 1)V_{dd}^{opt} + V_{th}^{opt}}{2nUt\alpha} - \frac{1}{2} \tag{6.6}$$

With α the alpha power law coefficient, n the sub-threshold slope and Ut the thermal potential.

Table 6.1 shows the equivalent of Eq. (6.6) in the case of the STM 90nm technology (used values are obtained from Chapter 4).

LVT	$k1 \approx 4.0V_{dd}^{opt} + 7.3V_{th}^{opt} - 0.5$
SVT	$k1 \approx 5.2V_{dd}^{opt} + 8.0V_{th}^{opt} - 0.5$
HVT	$k1 \approx 5.9V_{dd}^{opt} + 7.1V_{th}^{opt} - 0.5$

Table 6.1: Approximation of k1 for STM 90nm technology

From the equations in Table 6.1 is possible to see how the case k1=1 is very difficult to reach and it would correspond to extremely low optimal Vdd and Vth.

In Eq. (6.6), k1 was expressed in term of optimal Vdd and optimal Vth, but it can also be related to the on current (I_{on}) and the off current (I_{off}), or better to the ratio of these two. In fact, using Eq. (6.1) and Eq. (6.2):

$$Pdyn = k1 \cdot Pstat \tag{6.7}$$

$$a \cdot C \cdot N \cdot V_{dd}^2 \frac{I_{\text{on}}^{opt}}{k_t \cdot C \cdot LD \cdot V_{dd}} = k1 \cdot N \cdot V_{dd} \cdot I_{\text{off}}^{opt} \tag{6.8}$$

$$k1 = \frac{a}{LD}\frac{1}{k_t}\left.\frac{I_{\text{on}}}{I_{\text{off}}}\right|^{\text{opt}} \tag{6.9}$$

If now we remember that k_t is just a constant, k1 can easily be expressed by:

$$k1 \propto \frac{a}{LD}\left.\frac{I_{\text{on}}}{I_{\text{off}}}\right|^{\text{opt}} \tag{6.10}$$

It is important to note that in Eq. (6.10) the Ion/Ioff also depends on activity (a) and logical depth (LD).

Based on SIA International Technology Roadmap for Semiconductors 2004 [33], the expected ratios of I_{on} over I_{off} for present and future technologies are:

Year	2006	2009	2012	2015	2018
HP	23400	22714	17900	7000	4380
LOP	203333	154000	118571	90000	31667
LSTP	25.5E6	17.5E6	13.2E6	10.9E6	9.9E6

Table 6.2: SIA ITRS 2004 expected transistors $I_{\text{on}}/I_{\text{off}}$ for High Performance (HP), Low Operating Power (LOP) and Low Standby Power (LSTP) circuits.

Looking at Table 6.2, we see how the ratio I_{on} over I_{off} will decrease with time due to the large increase of the static power consumption. On the other hand, we have previously seen that the variable $k1$ doesn't change so much. Hence, we can conclude that an architecture with activity a and logical depth LD working at its optimal condition in a present technology will require a higher ratio a/LD in a future technology. This can be achieved by reducing the logical depth LD, but also by increasing the activity a, which correspond to having a better use of the implemented hardware. This reasoning, for instance, will tend to favor pipeline over parallelization. Indeed, the ratio a/LD is increased in a pipelined design due to the reduction of LD, whereas the same ratio will remain almost unchanged during parallelization (cf Table 6.5 and Table 6.6).

By using Eq. (6.22) (derived later in this chapter), it is possible to express k1 in a much simpler expression.

$$k1 = \left.\frac{Pdyn}{Pstat}\right|^{opt} = \frac{aCNfV_{dd}^2}{I_0 N V_{dd} e^{-Vth/nUT}} \cong \frac{aCfV_{dd}}{2nUtaCf/(1-\chi A)} = \frac{V_{dd}}{2nUt}(1-\chi A) \quad (6.11)$$

In a similar way, Eq. (6.11) can also be expressed by the optimal Vth by applying Eq. (6.15).

$$k1 = \frac{V_{th} + \chi B}{2nUt} \quad (6.12)$$

6.3 Optimal Vdd and Vth formulas

In this section the complete derivation of the optimal threshold voltage and supply voltage is presented. The difficulty of the derivation is to express Vth^{opt} without

the use of Vdd^{opt} and vice-versa, i.e. we need to decouple these two variables. To achieve this, we need to linearize the expression $Vdd^{1/\alpha}$ (with α the alpha power law coefficient, its value spanning from 1 to 2), which is the origin of the transcendental nature of Eq. (6.3).

Fig. 6.3 shows the expression $Vdd^{1/\alpha}$ and its linear approximation for Vdd from 0.3V to 1V for $\alpha = 1.65$.

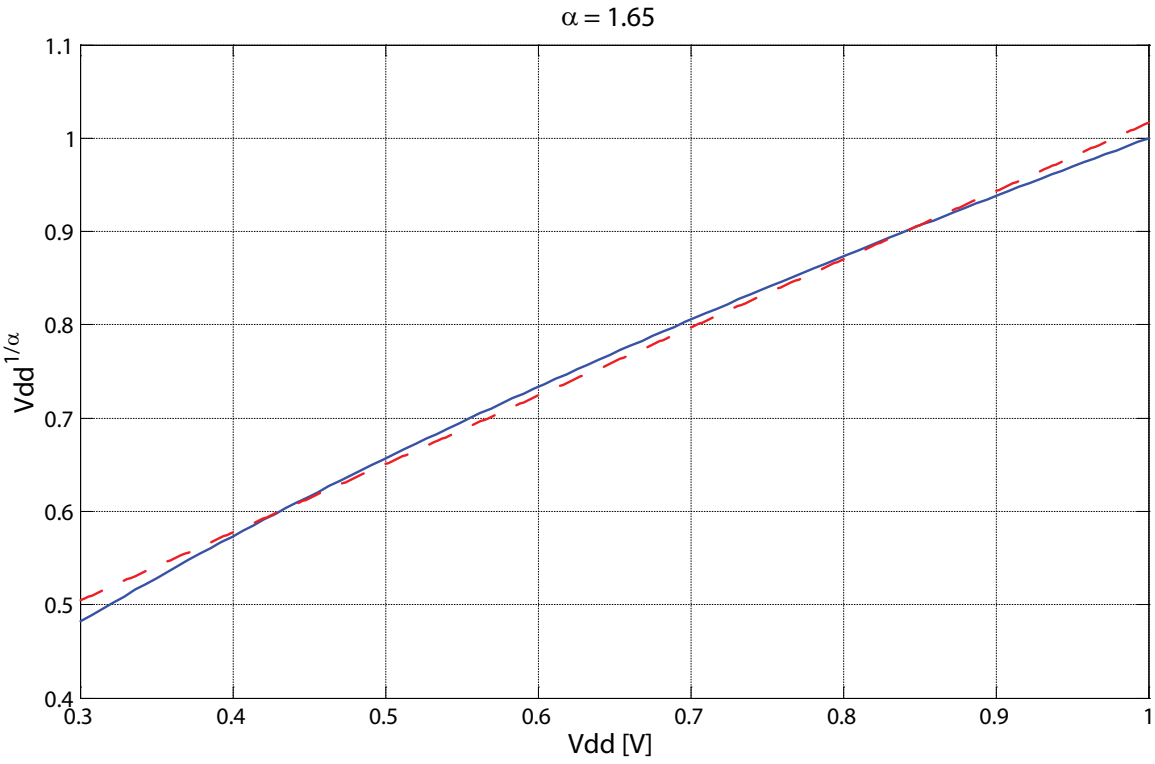

Figure 6.3: $Vdd^{1/\alpha}$ [solid line] and its linear approximation [dashed line]

From this figure, we see how well $Vdd^{1/\alpha}$ can be linearized over a relative large interval, leading to the follow approximation:

$$V_{dd}^{1/\alpha} \approx A(\alpha) \cdot V_{dd} + B(\alpha) \tag{6.13}$$

With A and B depending on α but also on the interval of Vdd where the approximation is done. A and B can be determined numerically (easy) and analytically (more complex, but feasible). For Vdd in the interval [0.3V;1V], the graph in Fig. 6.4 can be used to estimate A and B.

The lower graph in Fig. 6.4 shows the maximal error in percent obtained with the proposed linear approximation. For the range of Vdd restricted to the interval [0.3V;1V] the error always remain lower than 5%. It is important to note that newer technology will tend to have even smaller values of α which results in an even better

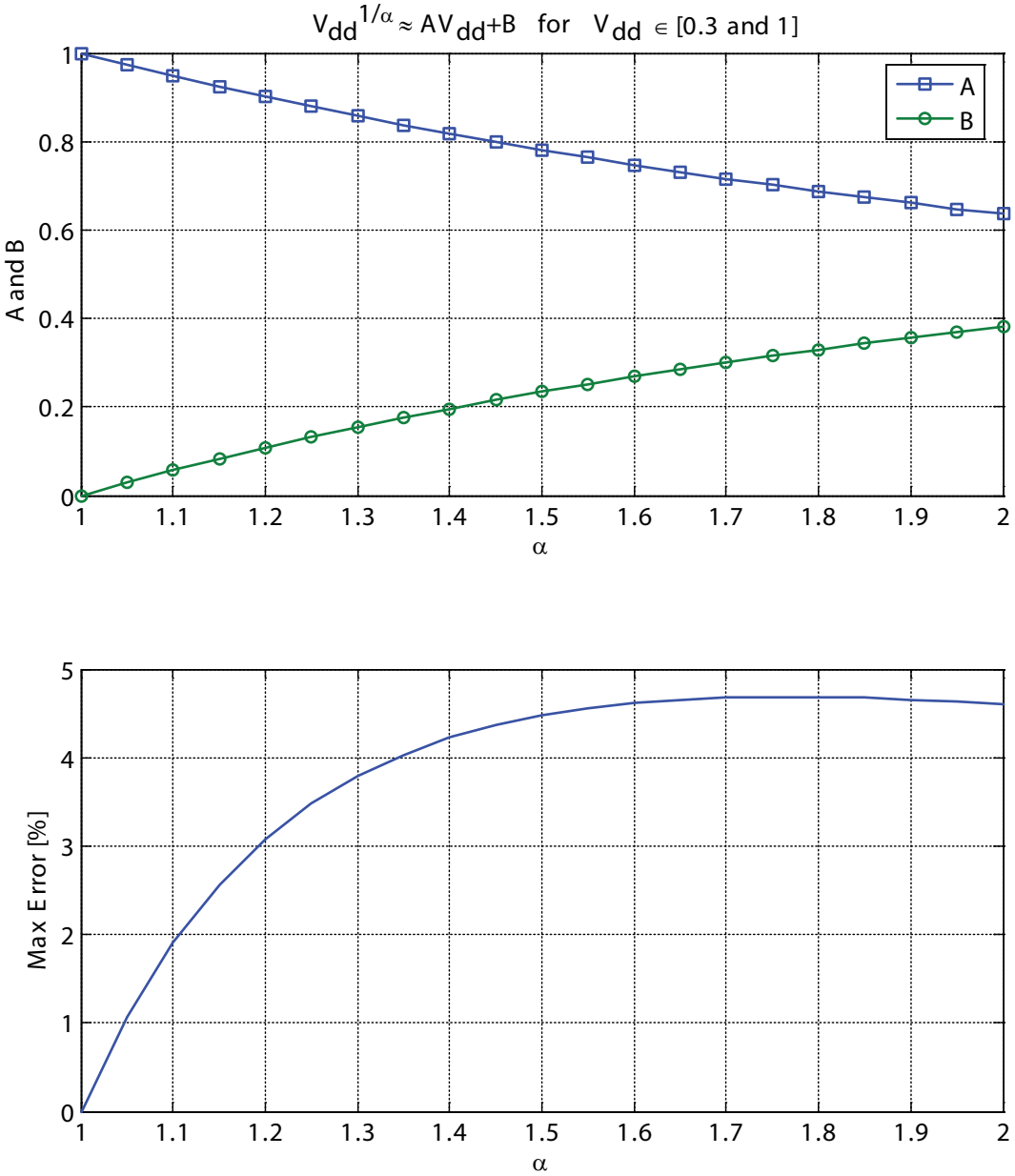

Figure 6.4: Linearization coefficients for Vdd in [0.3V;1V]

approximation of Eq. (6.13). Moreover, in the case where a better approximation is needed, the error can be further reduced by limiting the range of Vdd.

In Fig. 6.5, the parameters A and B are calculated for Vdd between 0.3V and 0.6V and they report a maximal error lower than 1.4%. The values of A and B for the α corresponding to the three different variations of the STM 90nm CMOS technology are reported in Table 6.3.

Using the approximation in Eq. (6.13) is now possible to rewrite Eq. (6.3) in a simpler way:

$$V_{th}^{opt}(V_{dd}^{opt}) \cong V_{dd}^{opt} - \chi(A \cdot V_{dd}^{opt} + B) = V_{dd}^{opt}(1 - \chi \cdot A) - \chi \cdot B \tag{6.14}$$

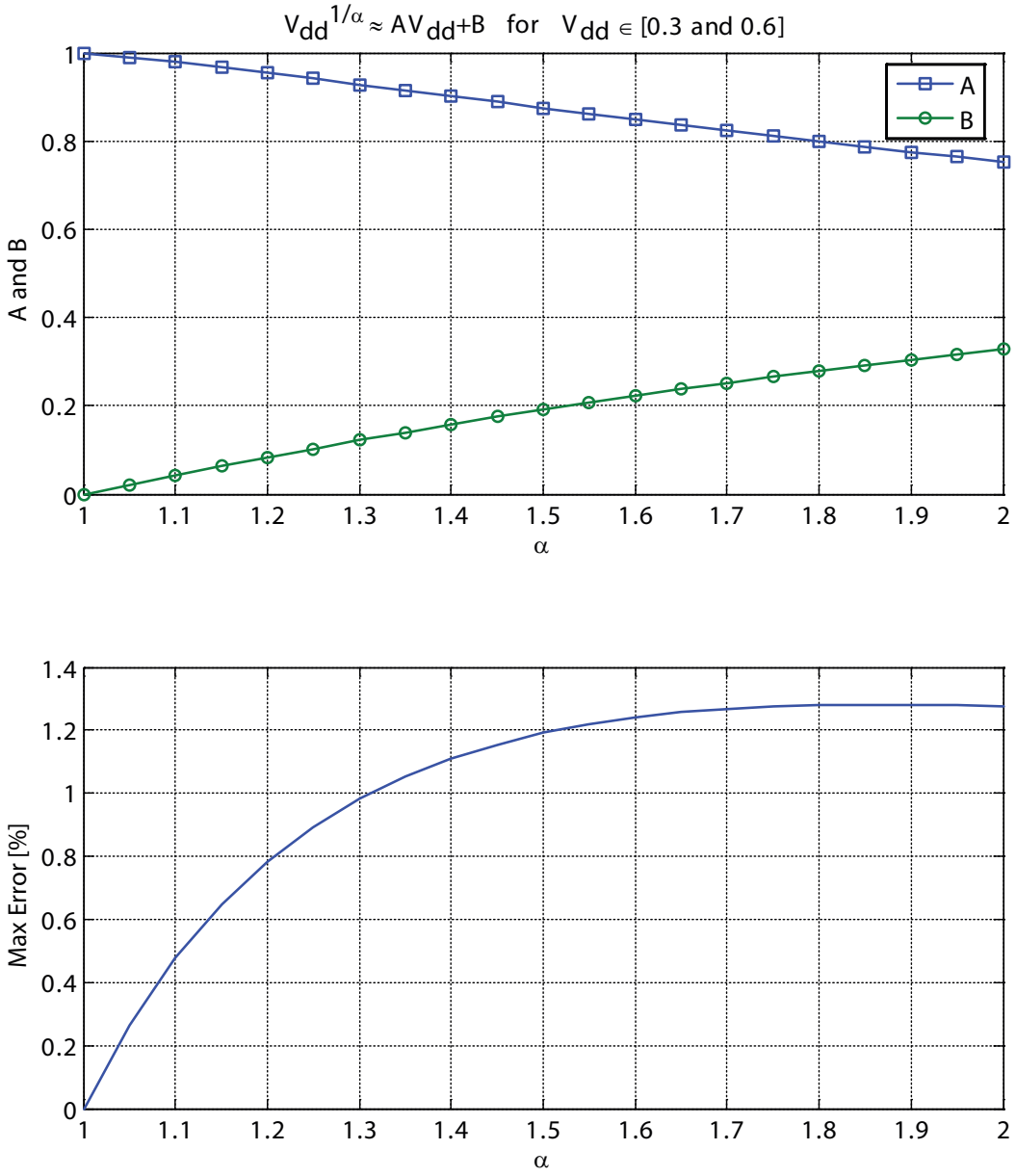

Figure 6.5: Linearization coefficients for Vdd in [0.3V;0.6V]

	$Vdd \in [0.3V; 1V]$			$Vdd \in [0.3V; 0.6V]$		
	LVT	SVT	HVT	LVT	SVT	HVT
α	1.56	1.65	1.84	1.56	1.65	1.84
$A(\alpha)$	0.760	0.731	0.676	0.859	0.835	0.788
$B(\alpha)$	0.260	0.286	0.342	0.210	0.238	0.290

Table 6.3: Values of A and B for the three types of STM090 transistors

Such an approximation is now invertible and permits to estimate the optimal Vdd:

$$V_{dd}^{opt}(V_{th}^{opt}) \cong \frac{V_{th}^{opt} + \chi \cdot B}{1 - \chi \cdot A} \underbrace{=}_{Vth = Vth0 - \eta Vdd} \frac{V_{th0}^{opt} + \chi \cdot B}{1 - \chi \cdot A + \eta} \tag{6.15}$$

Another useful expression is the first derivative of Vth with respect to Vdd. This expression will be used in the next sections, but for the sake of simplicity, it will be presented here. From Eq. (6.14) the partial derivative becomes:

$$\frac{\partial V_{th}^{opt}}{\partial V_{dd}^{opt}} \cong (1 - \chi \cdot A) \tag{6.16}$$

6.3.1 Optimal threshold voltage derivation

The expression of the optimal threshold voltage can be derived by searching for the Vth that would minimize the total power consumption. Hence:

$$\frac{\partial Ptot(V_{th})}{\partial V_{th}} = \frac{\partial Pdyn(V_{th})}{\partial V_{th}} + \frac{\partial Pstat(V_{th})}{\partial V_{th}} = 0 \tag{6.17}$$

Or, better:

$$\frac{\partial Pdyn(V_{th})}{\partial V_{th}} = -\frac{\partial Pstat(V_{th})}{\partial V_{th}} \tag{6.18}$$

It is now possible to substitute Eq. (6.1) in Eq. (6.18) to obtain:

$$2aCNfV_{dd}\frac{\partial V_{dd}}{\partial V_{th}} = -I_0 N e^{-Vth/nUt}\left(\frac{\partial V_{dd}}{\partial V_{th}} - \frac{V_{dd}}{nUt}\right) \tag{6.19}$$

$$e^{Vth/nUt} = \frac{I_0}{2nUtaCf}\left(\frac{\partial V_{th}}{\partial V_{dd}} - \frac{nUt}{V_{dd}}\right) \tag{6.20}$$

$$e^{Vth/nUt} \underbrace{\cong}_{Eq.\ (6.16)} \frac{I_0}{2nUtaCf}\left(1 - \chi A - \frac{nUt}{V_{dd}}\right) \tag{6.21}$$

At room temperature, nUt is about 0.04V (refer to Table 4.9 for the exact value in the case of STM090 technology). So, even if for instance the optimal supply voltage will be as low as 0.4V, the ratio nUt/V_{dd} will be as low as 0.1 or even lower for higher optimal Vdd. For this reason, we consider this term negligible compared to $1 - \chi A$. This is a mandatory approximation in order to be able to decouple Vth and Vdd.

The optimal Vth can finally be calculated:

$$e^{Vth/nUt} \cong \frac{I_0}{2nUtaCf}(1 - \chi A) \tag{6.22}$$

$$V_{th}^{opt} \cong nUt \ln\left(\frac{I_0}{2nUtaCf}(1 - \chi A)\right) \quad \text{with: } \chi^{\alpha} = \frac{k_t \cdot C \cdot f \cdot LD}{I_0\left(\frac{e}{\alpha nUt}\right)^{\alpha}} \tag{6.23}$$

Eq. (6.23) shows the influence of architectural parameters (like a, LD [included in χ], f) and technology parameters (like I_0, n, C, α, k_t) to the optimal threshold voltage Vth.

Consider a 16 bit Wallace multiplier with the following properties:

Technology	STM090 SVT
Nominal Dynamic Power	693.28 μW
Nominal Static Power	9.90 μW
Nominal Activity	0.267
Nominal Frequency	100 MHz
Nominal Max Delay	2.38 ns
Nominal Supply voltage	1 V
Nominal Threshold voltage	0.353 V

Table 6.4: Parameters of a 16 bit Wallace multiplier

Fig. 6.6 shows the optimal Vth vs. activity for the multiplier described in Table 6.4, while maintaining the other architectural parameters constant.

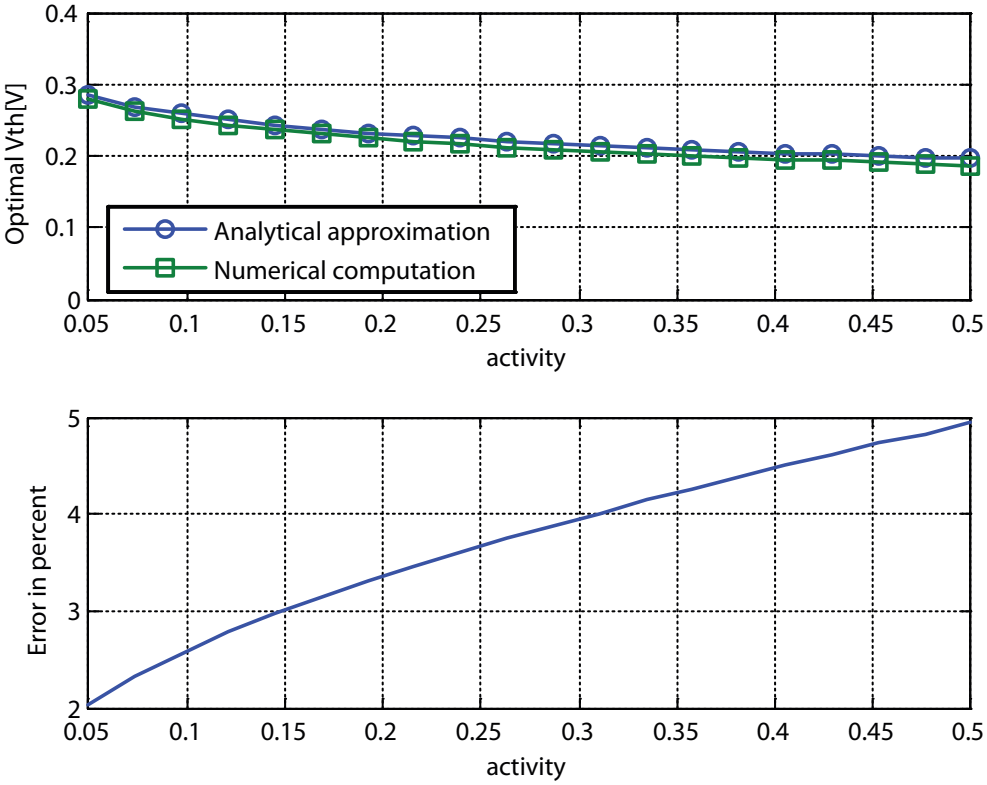

Figure 6.6: Optimal Vth vs. activity

The optimal Vth has been calculated in two separated ways. The former, called analytical approximation on the plot, is the direct use of Eq. (6.23) with $Vdd^{1/\alpha}$ lin-

earized over the interval [0.3V;1V], whereas the second, called numerical computation on the plot, is obtained with a high resolution numerical computation based on the non-approximated Eq. (6.1) and Eq. (6.3).

The first remark on Fig. 6.6 is that the error of the approximation remains lower than 5% for the proposed range of activities.

Another interesting point is the shape of the curve optimal Vth vs. a. In fact, we can observe how Vth^{opt} increases for low activity, while it decreases for high activities, as already noted on Fig. 6.2. Moreover, it is visible that for high activities, Vth^{opt} becomes almost constant or varies only very slightly.

A similar graph is found in Fig. 6.7, but this time with the frequency as a variable parameter, while the other parameters are kept constant.

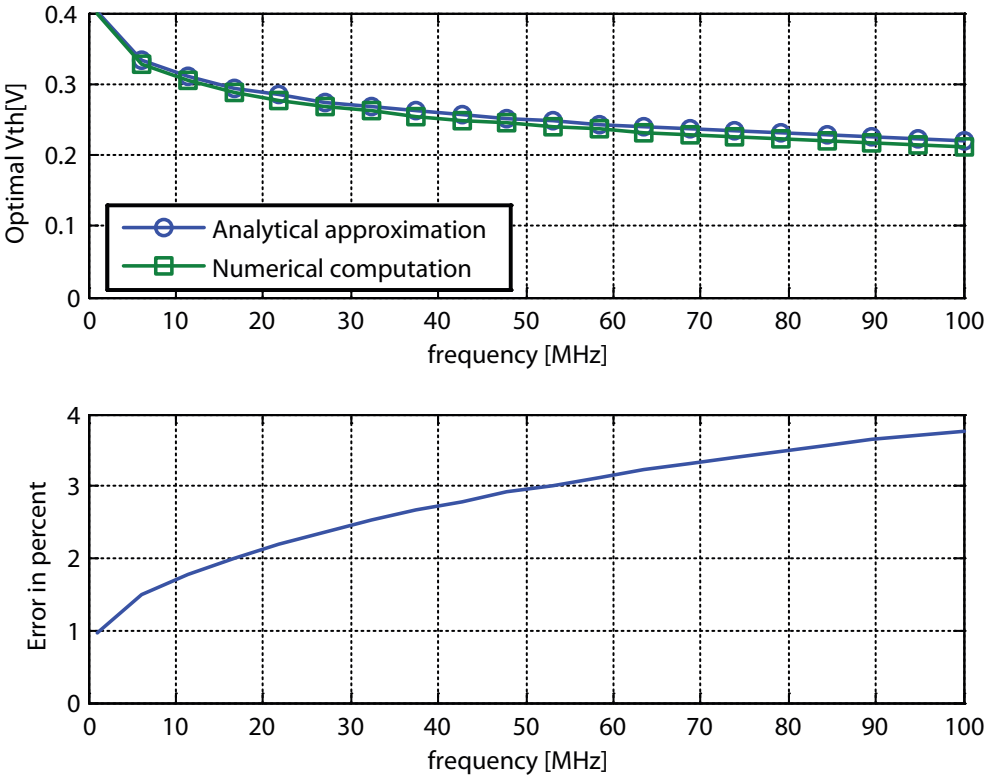

Figure 6.7: Optimal Vth vs. frequency

As expected, the increase of the working frequency results in a reduction of the optimal Vth. In fact, in order to achieve the higher frequency, Vth is reduced to obtain a larger $(Vdd - Vth)$.

The last optimal Vth graph is Fig. 6.8 and it shows the optimal Vth vs. the logical depth (LD).

It is important to note that the optimal Vth is almost insensitive to the logical depth. This can be quite surprising, but it is explained by the important change in

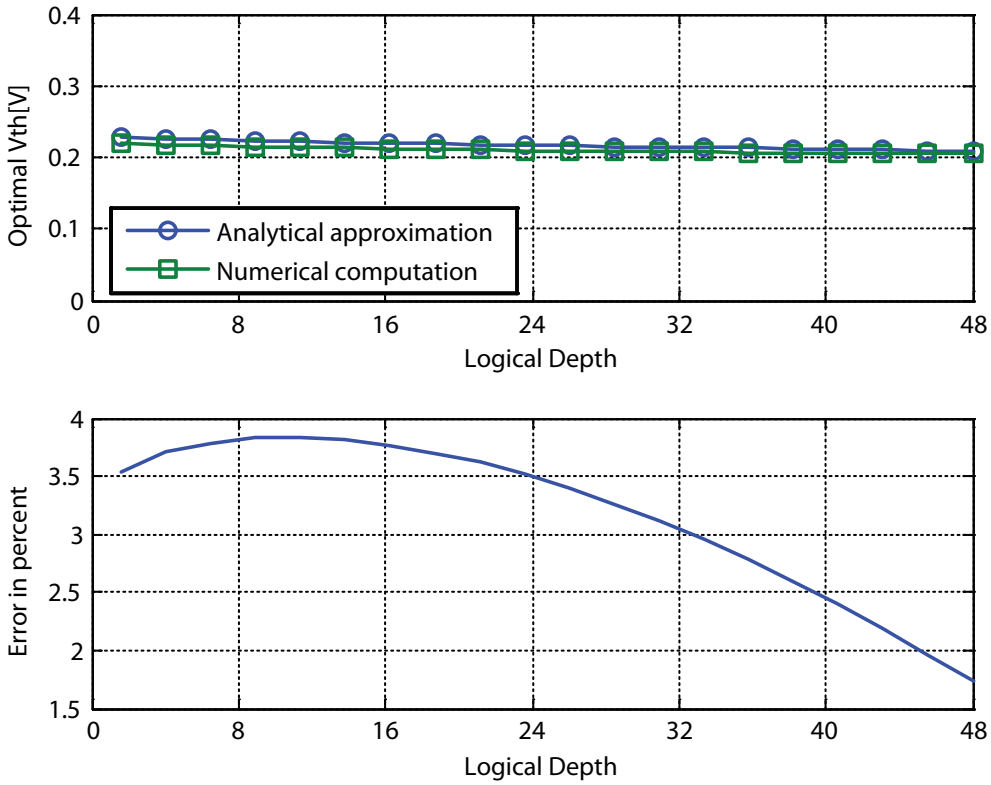

Figure 6.8: Optimal Vth vs. logical depth

the optimal Vdd (refer to the next section), which "absorbs" almost completely the changes in the logical depth.

In the case where the nominal technology values of Vdd, Vth, $Pdyn$ and $Pstat$ are known, Eq. (6.23) can be also written as:

$$V_{th}^{opt} \cong nUt \ln\left(\frac{Pstat^{nom}}{Pdyn^{nom}} \frac{V_{dd}^{nom} e^{(Vth0^{nom} - \eta Vdd^{nom})/nUt}}{2nUt}(1 - \chi A)\right) \tag{6.24}$$

6.3.2 Optimal supply voltage derivation

Once the optimal Vth has been calculated, the derivation of the optimal Vdd is very simple thanks to Eq. (6.15). In fact, by simply replacing the expression of Vth^{opt}, Eq. (6.25) and Eq. (6.26) can be obtained.

$$V_{dd}^{opt} \cong \frac{nUt \ln\left(\frac{I_0}{2nUtaCf}(1 - \chi A)\right) + \chi B}{1 - \chi A} \qquad \text{with: } \chi^{\alpha} = \frac{k_t \cdot C \cdot f \cdot LD}{I_0 \left(\frac{e}{\alpha nUt}\right)^{\alpha}} \tag{6.25}$$

$$V_{dd}^{opt} \cong \frac{nUt \ln\left(\frac{Pstat^{nom}}{Pdyn^{nom}} \frac{V_{dd}^{nom} e^{(Vth0^{nom} - \eta Vdd^{nom})/nUt}}{2nUt}(1 - \chi A)\right) + \chi B}{1 - \chi A} \tag{6.26}$$

To discuss the validity of this approximation, we can reconsider the circuit described in Table 6.4. Fig. 6.9 shows the optimal Vdd for different activities. The values of Vdd^{opt} are calculated in two ways. The analytical approximation is based on Eq. (6.25), whereas the numerical computation is based on the non-approximated equations (6.1) and (6.3).

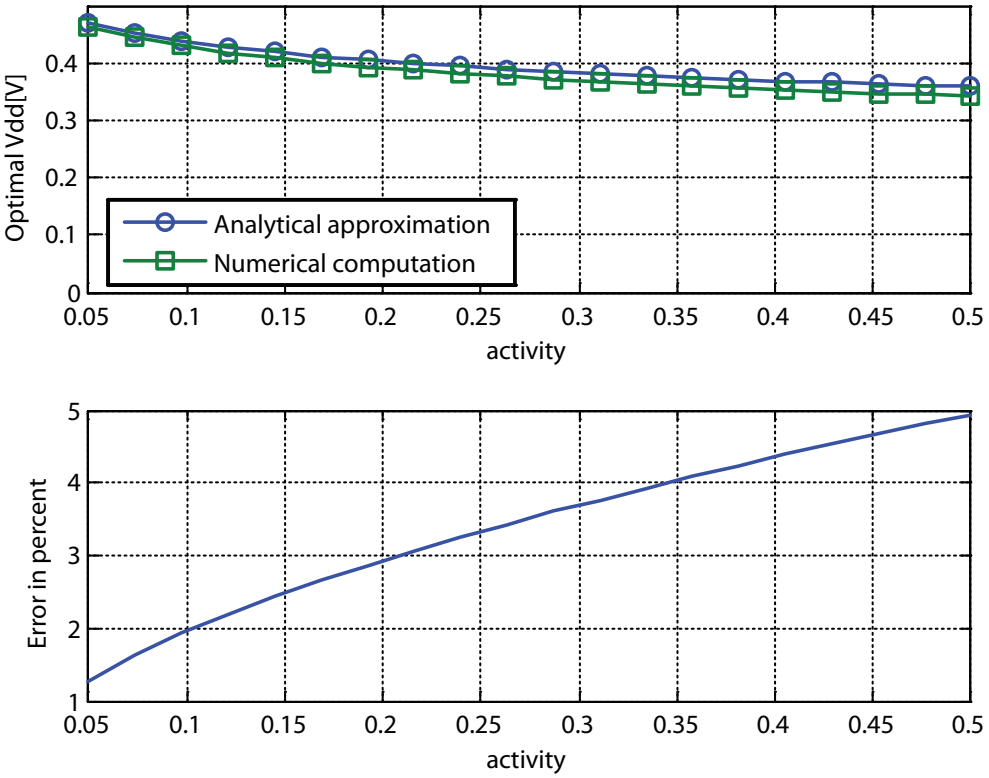

Figure 6.9: Optimal Vdd vs. activity

From Fig. 6.9, we can see that, for the chosen range of activity, the error remains smaller than 5%. Moreover, by looking at the shape of the Vdd^{opt} curve, we observe a trend very similar to the one for Vth. Actually, the increase of activity reduces both Vth^{opt} and Vdd^{opt} in a similar way. This can be explained by the fact that a change in activity doesn't modify the timing constraints, and hence the difference $Vdd - Vth$ (cf. Eq. (6.2)) remains almost unchanged.

A similar graph can be plotted for the frequency as the free variable. This situation is represented by Figure 6.10.

It is interesting to note the shape of the $Vdd^{opt}(f)$ curve. For the high frequencies the behavior corresponds to what we would expect, in fact the reduction of the working frequency allows a reduction of the optimal supply voltage (which correspond to an increase of the optimal threshold voltage), but for low frequencies the optimal Vdd starts to increase again. This behavior comes from the high increase of the optimal

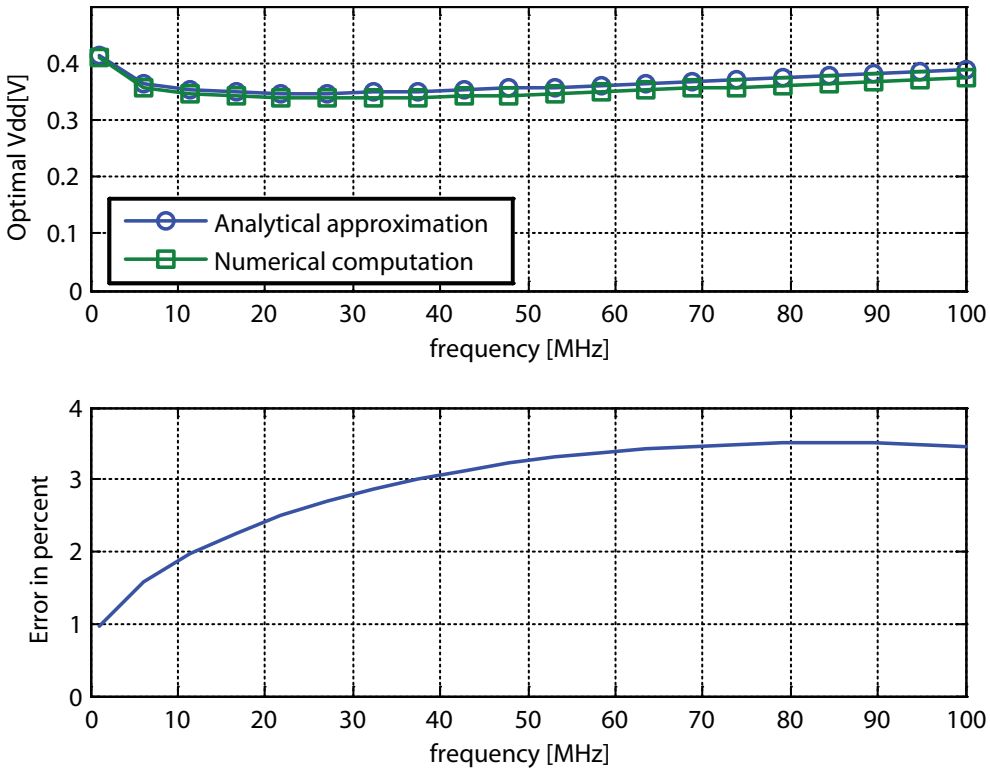

Figure 6.10: Optimal Vdd vs. frequency

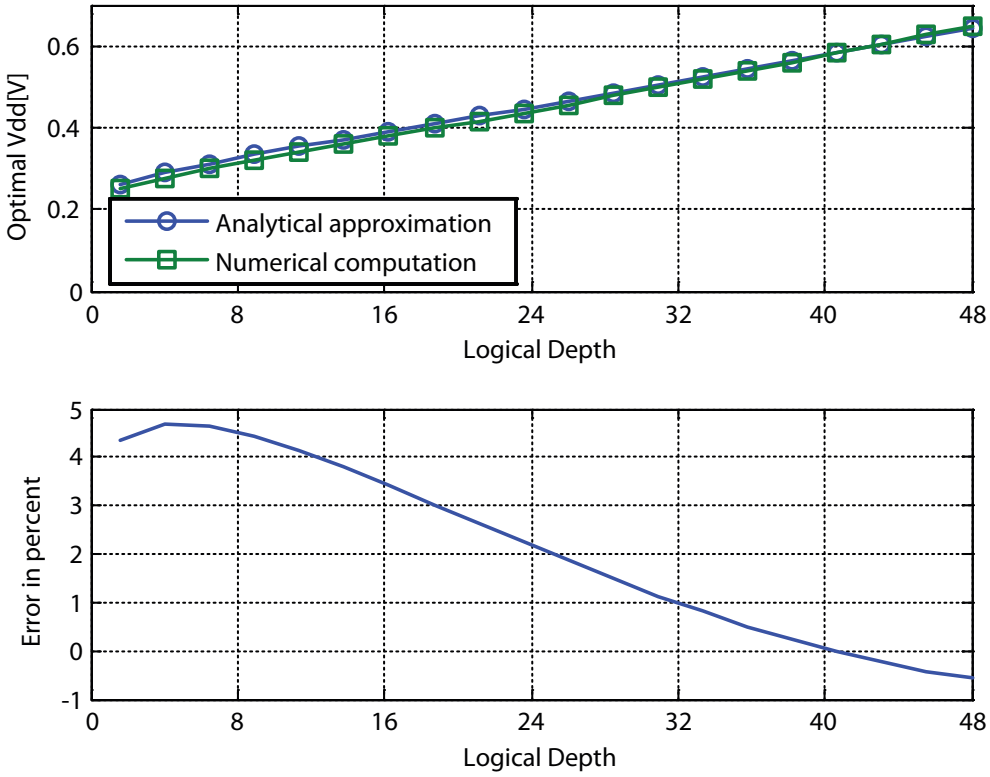

Figure 6.11: Optimal Vdd vs. logical depth

Vth in this zone. In fact, to avoid a weak inversion regime ($Vdd < Vth$), Vdd^{opt}

needs to increase in order to maintain the difference $Vdd - Vth$ positive.

The last graph of the optimal Vdd is reported in Fig. 6.11. There, Vdd^{opt} is plotted versus the logical depth. This curve shows an almost linear behavior. In fact, as stated before, the change in the timing requirements resulting from the change in the logical depth affects almost exclusively the optimal Vdd whereas the optimal Vth remains quite constant (cf. Fig. 6.8).

Finally, we can say that frequency mainly affects the optimal Vth, logical depth mainly affects the optimal Vdd, and activity affects both of them.

6.4 Optimal total power

From what has been developed in the previous pages, it is now possible to obtain some approximations of the optimal total power consumption. Unfortunately, due to the transcendental nature of the involved equations, no exact formula exists to determine the optimal $Ptot$. Nevertheless, with the help of a few basic assumptions, approximated equations can be found. In the next sections, two different approaches are proposed. The former develops a rough way to compare architectures that present similar values of k1 (≡ optimal Pdyn/Pstat), whereas the latter is a much more precise approximation for an absolute optimal total power estimation.

6.4.1 Optimal power comparison with $k1$ constant

For this first derivation, the assumption is done that k1 is constant or at least varies very few. This rough approach can be used as a quick way to compare the optimal total power consumption of two (or more) circuits having very similar characteristics in the sense of a similar k1 (≡ optimal Pdyn/Pstat).

The optimal total power can be expressed with k1 as:

$$Ptot^{opt} = Pdyn^{opt}\left(1 + \frac{1}{k1}\right) \tag{6.27}$$

From our experience, typical values of k1 span from 3 to 7 considering very different architectural blocks like multipliers, adders, counters, shift registers, FIR, microprocessors, etc. In the case of circuits with similar functions and working conditions, k1 can be considered constant, at least for a first rough approximation. Just as an example, ten different 16bit multipliers (7 RCA variations and 3 Wallace variations) implemented in a STM 90nm technology and with a working frequency of 33 MHz have a k1 included in the range between 4.22 and 4.69.

To fix the ideas, the error introduced by a $\Delta k1 \neq 0$ can be calculated:

$$\Delta Ptot = \frac{\partial}{\partial k1} Pdyn \left(1 + \frac{1}{k1}\right) \Delta k1 = -Pdyn \frac{\Delta k1}{k1^2} = -Ptot \frac{\Delta k1}{k1(k1+1)} \tag{6.28}$$

Or:

$$\frac{\Delta Ptot}{Ptot} = -\frac{\Delta k1/k1}{k1+1} \tag{6.29}$$

Practically, Eq. (6.29) means that the relative error ($\Delta k1/k1$) introduced by a non constant k1 has an effect divided by $k1+1$ on the optimal total power $Ptot$. Hence the worst case $\Delta Ptot/Ptot$ in our example of the ten 16 bit multipliers presents an error of about 2.1%.

Thanks to the constant k1 hypothesis, the optimal total power consumption comparison is now reduced to the comparison of the optimal dynamic power ($Pdyn$).

$$Ptot' \overset{?}{<} Ptot \tag{6.30}$$

$$Pdyn' \left(1 + \frac{1}{k1}\right) \overset{?}{<} Pdyn \left(1 + \frac{1}{k1}\right) \tag{6.31}$$

$$Pdyn' \overset{?}{<} Pdyn \tag{6.32}$$

$$a'C'N'f'V'^2_{dd} \overset{?}{<} aCNfV^2_{dd} \tag{6.33}$$

$$V'_{dd} \overset{?}{<} V_{dd}\sqrt{\frac{aCNf}{a'C'N'f'}} \tag{6.34}$$

The parameters with an apostrophe (') correspond to the new architecture which is compared to a reference design (no apostrophe).

Parallelization example

To better understand the usefulness of Eq. (6.34), let us apply it to the case of a circuit parallelization. Table 6.5 reports the typical architectural parameter variations in the case of a P times parallelization.

In a parallelization process, the number of cells is more than P times the original one due to the overhead introduced mainly by the multiplexer and the additional registers required to maintain a valid data on both blocks. We can define the Dynamic OverHead (DOH) as the relative increment of the dynamic power due to this overhead at nominal conditions (i.e. $Pdyn'_{nom} = (1 + DOH)Pdyn_{nom}$).

Symbol	Name	Effect of parallelization
a	activity	$\approx$ /P
N	number of cells	$\approx$ *P+ overhead
LD_{eff}	effective logical depth	/P
f	frequency	unchanged

Table 6.5: Effect of parallelization on architectural parameters

From Eq. (6.34) we now know that in order to reduce the optimal power consumption through parallelization, the following expression must be respected:

$$V'_{dd} \overset{!}{<} V_{dd}/\sqrt{1+DOH} \tag{6.35}$$

With V'_{dd} the optimal supply voltage after the parallelization and V_{dd} the optimal supply voltage before parallelization.

On the other hand, the optimal Vth, which depends mainly on activity, can be approximated as (from Eq. (6.23)):

$$Vth' \cong Vth + nUt \ln P \tag{6.36}$$

With Vth' the optimal threshold voltage after parallelization and Vth the optimal threshold voltage before parallelization.

Moreover, from Eq. (6.3) we can write:

$$\frac{V'_{dd} - V'_{th}}{V'^{1/\alpha}_{dd}} = \chi' = \chi/P^{1/\alpha} = \frac{V_{dd} - V_{th}}{P^{1/\alpha} V^{1/\alpha}_{dd}} \tag{6.37}$$

The combination of Eq. (6.35), Eq. (6.36) and Eq. (6.37) yields:

$$\chi \overset{!}{>} \left(\frac{P\sqrt{1+DOH}}{V_{dd}}\right)^{1/\alpha} \left(\frac{V_{dd}}{\sqrt{1+DOH}} - V_{th} - nUt \ln P\right) \tag{6.38}$$

All parameters in Eq. (6.38) refer to the design before parallelization. Hence, to know if a circuit can reach a lower optimal total power through parallelization it is sufficient to check that the previous inequality is respected.

In the same way, it is possible to determine the maximal value of DOH that still allow power savings when parallelization is performed.

Pipelining example

The same approach can be carried out in the case of a pipelining transformation. The effect of a typical pipelining transformation to the architectural parameters is shown in Table 6.6.

Symbol	Name	Effect of parallelization
a	activity	$\approx$ unchanged
N	number of cells	+ registers overhead
LD_{eff}	effective logical depth	$/p_f$
f	frequency	unchanged

Table 6.6: Effect of pipelining on architectural parameters

Ideally, the critical path would be divided by two (or by the number of pipelining stages in general) through a register bank insertion. Unfortunately, this ideal factor is practically never achieved because it is rare to be able to split the path exactly in the middle. For the sake of generality, the factor p_f (pipeline factor) is introduced. Its value represents the achieved ratio between the logical depth before and after the pipeline transformation.

Unlike the parallelization, the activity on a pipeline transformation remains almost unchanged, even if a small reduction could be observed due to less glitches. This will also mean that the optimal threshold voltage after the transformation is practically the same as before:

$$Vth' \approx Vth \tag{6.39}$$

With Vth and Vth' the optimal threshold voltage before and after the transformation respectively.

The overhead in a pipeline structure comes from the registers banks inserted in the data path to cut it in different segments. Like before, this overhead is considered as a dynamic power overhead and will be represented by the variable DOH (defined before). So, the condition on the optimal supply voltage remains the same as for the parallelization, i.e.:

$$V'_{dd} \overset{!}{<} V_{dd}/\sqrt{1+DOH} \tag{6.40}$$

Once more, a third condition can be obtained from Eq. (6.3):

$$\frac{V'_{dd}-V'_{th}}{V'^{1/\alpha}_{dd}} = \chi' = \chi/p_f^{1/\alpha} = \frac{V_{dd}-V_{th}}{p_f^{1/\alpha}V_{dd}^{1/\alpha}} \tag{6.41}$$

The combination of Eq. (6.39), Eq. (6.40) and Eq. (6.41) gives:

$$p_f \overset{!}{>} \frac{1}{\sqrt{1+DOH}} \left(\frac{V_{dd} - V_{th}}{V_{dd}/\sqrt{1+DOH} - V_{th}} \right)^{\alpha} \tag{6.42}$$

Or:

$$\chi \overset{!}{>} \left(\frac{p_f \sqrt{1+DOH}}{V_{dd}} \right)^{1/\alpha} (V_{dd}/\sqrt{1+DOH} - V_{th}) \tag{6.43}$$

Or even:

$$\chi \overset{!}{>} \left(\frac{p_f \sqrt{1+DOH}}{V_{dd}} \right)^{1/\alpha} \frac{V_{dd} - V_{dd}/\sqrt{1+DOH}}{(p_f \sqrt{1+DOH})^{1/\alpha} - 1} \tag{6.44}$$

If one of the conditions in Eq. (6.42) or Eq. (6.43) or Eq. (6.44) is respected, pipelining the design is worthwhile from a optimal total power point of view.

Considering both the results for parallelization and pipelining, we can say that these transformations are more effective for large logical depths or high frequencies. Moreover, new technologies will tend to reduce the value of χ, making pipelining and parallelization less interesting techniques.

If we want to compare parallelization against pipelining, we can use Eq. (6.38) and Eq. (6.43). The two equations are very similar. If we consider that $nUt \ln P$ is much smaller than $V_{dd}/\sqrt{1+DOH} - V_{th}$, which is in general the case, and we also assume that both transformations have the same DOH, we can compare parallelization against pipelining by simply comparing the parameter P against p_f. As we have seen before, p_f is always smaller than the ideal factor which would correspond to the number of stages. So, for the same degree of pipelining and parallelization, p_f will always be smaller than the factor P. For this reason we can conclude that the condition in Eq. (6.43) will be easier to fulfill compared to Eq. (6.38), making pipelining a preferred transformation against parallelization.

6.4.2 Absolute optimal total power

The previous section illustrates a rough approximation to quickly compare architectures with a similar $k1$. Even if this approach can be useful, we would sometimes prefer to be able to estimate the absolute value of the optimal total power, rather than by comparison with other architectures.

With Eq. (6.23) and Eq. (6.25), we are able to calculate the optimal total power, but it could be useful to be able to express the optimal total power directly from the

architectural and technology parameters. This would avoid the need to pre-calculate the optimal threshold and supply voltage and would permit to better understand the influence of the architectural and technology parameters on the optimal total power.

Let us start by including Eq. (6.23) in the total power equation:

$$\begin{aligned} Ptot &= aCNfV_{dd}^2 + NV_{dd}I_0 e^{-\frac{Vth}{nUt}} && (6.45)\\ &= aCNfV_{dd}^2 + 2V_{dd}\frac{nUtaCNf}{1-\chi A} && (6.46)\\ &= aCNf\left(V_{dd}^2 + 2V_{dd}\frac{nUt}{1-\chi A}\right) && (6.47) \end{aligned}$$

Eq. (6.47) shows a term in V_{dd}^2 and a term in $2V_{dd}$. This means that two of the three terms of the square development of $(a+b)^2 = a^2 + 2ab + b^2$ are present. Supposing that the missing term (b^2) is very small compared to the sum of the other two, then the development can be reversed.

$$\begin{aligned} Ptot &= aCNf\left(V_{dd}^2 + 2V_{dd}\frac{nUt}{1-\chi A}\right) && (6.48)\\ &\approx aCNf\left(V_{dd}^2 + 2V_{dd}\frac{nUt}{1-\chi A} + \left(\frac{nUt}{1-\chi A}\right)^2\right) && (6.49)\\ &= aCNf\left(V_{dd} + \frac{nUt}{1-\chi A}\right)^2 && (6.50) \end{aligned}$$

The approximation that has just been used is the same as the one used to obtain Eq. (6.23), namely that $nUt/V_{dd} \ll (1-\chi A)$. The validity of this approximation can be verified in the practical cases reported in the next chapters.

Finally, the expression of the optimal supply voltage (Eq. (6.25)) can be inserted in Eq. (6.50) to obtain the optimal total power formula.

$$Ptot^{opt} \cong \frac{aCNf}{(1-\chi A)^2}\left[nUt\left(\ln\left(\frac{I_0}{2nUtaCf}(1-\chi A)\right)+1\right)+\chi B\right]^2 \quad (6.51)$$

Eq. (6.51) is a fundamental equation, in fact it permits to analytically estimate an approximation of the optimal total power directly from architectural parameters like activity (a), number of cells (N), frequency (f), logical depth (LD, included in χ) and technology parameters like transistor reference current (I_0), sub-threshold slope (n), alpha power law coefficient (α, include in A and B), delay coefficient (k_t, included in

χ) and average capacitance C. The detailed discussion of the influence of these two families of parameters on the optimal total power consumption will be carried out in the next two chapters.

An alternative expression for the optimal total power can also be obtained by combining Eq. (6.50) with Eq. (6.11). The resulting formula illustrates the relationship between the optimal total power and $k1$:

$$Ptot^{opt} \cong aCNf \left(\frac{nUt}{1 - \chi A} \right)^2 (2k1 + 1)^2 \tag{6.52}$$

6.5 Summary

In this chapter, we have discussed the existence of a total power consumption optimum characterized by a trade-off between dynamic and static power contributions. We have also seen that typical values of k1 (optimal Pdyn over optimal Pstat ratio) are between 3 and 7.

After that, we have developed models for the optimal supply voltage and optimal threshold voltage, showing that frequency modifications mainly influence Vth, logical depth modifications mainly affect Vdd, whereas activity modifications have impacts on both of them. Then, a total power comparison based on the rough assumption of a quasi-constant k1 revealed that pipelining and parallelization are more effective for large logical depths and high frequencies and that new technologies (which will tend to have lower χ) will make these two transformations less interesting. Finally, we observed that the condition for achieving a power saving through pipelining is more easily fulfilled than the one for parallelization.

In the case where an absolute estimation of the optimal total power is required, the expression of an approximated closed-form equation has been given.

The most important equations provided in this chapter are summarized below to permit a quick access.

Starting from:

$$Ptot = Pdyn + Pstat = aCNfV_{dd}^2 + NV_{dd}I_0 e^{-\frac{Vth}{nUt}}$$

$$V_{th}^{opt} = V_{dd}^{opt} - \chi \cdot (V_{dd}^{opt})^{1/\alpha} \qquad \text{with: } \chi^\alpha = \frac{k_t \cdot C \cdot f \cdot LD}{I_0 \left(\frac{e}{\alpha n Ut} \right)^\alpha}$$

we obtained:

$$
\begin{aligned}
V_{th}^{opt} &\cong nUt \ln\left(\frac{I_0}{2nUtaCf}\left(1-\chi A\right)\right) \\
&\cong nUt \ln\left(\frac{Pstat^{nom}}{Pdyn^{nom}}\frac{V_{dd}^{nom} e^{(Vth0^{nom}-\eta Vdd^{nom})/nUt}}{2nUt}(1-\chi A)\right) \\
V_{dd}^{opt} &\cong \frac{nUt \ln\left(\frac{I_0}{2nUtaCf}\left(1-\chi A\right)\right)+\chi B}{1-\chi A} \\
&\cong \frac{nUt \ln\left(\frac{Pstat^{nom}}{Pdyn^{nom}}\frac{V_{dd}^{nom} e^{(Vth0^{nom}-\eta Vdd^{nom})/nUt}}{2nUt}(1-\chi A)\right)+\chi B}{1-\chi A} \\
Ptot^{opt} &\cong \frac{aCNf}{(1-\chi A)^2}\left[nUt\left(\ln\left(\frac{I_0}{2nUtaCf}(1-\chi A)\right)+1\right)+\chi B\right]^2
\end{aligned}
$$

Chapter 7

Architectural impact on total power

Many architectural parameters, e.g. activity a, number of cells N, logical depth LD (contained in χ), influence the optimal total power consumption (Eq. (6.51)). Knowing the effect of an architecture transformation (e.g. pipelining or parallelization) on such parameters allows to directly determine if a power saving can be obtained, just by using Eq. (6.51).

To discuss the impact of architectural modification on the optimal total power consumption, a set of thirteen 16 bit multipliers (described in details in Chapter 5) was designed in VHDL and synthesized using Synopsys Design Compiler (V2004.06). The library used for the synthesis was the 90nm CMOS090GPSVT from ST Microelectronics.

The data characterizing these thirteen multipliers at their nominal values (the ones provided by Synopsys DC) are reported in Table 7.1. Every multiplier works with a frequency able to generate one completed multiplication every 16ns. This means, for instance, that the 16 bit sequential architecture requires a local clock period of 1ns, whereas the 2 times parallelized implementation has 32ns of time per block.

The definitions of the parameters reported in Table 7.1 are:

- Cells: the number of design cells. One cell can be a very simple one (like an inverter) or a complex one (like a full adder);
- Nets: the number of inter-cells nets in the design;
- Area: the area of the design core; pads and routing spaces are not included;
- Activity: the average number of switching nets over the total number of nets per clock period. These values are obtained by an event-driven simulation under

									Nominal values				
	Cells	**Nets**	**Area** $[\mu m^2]$	**Activity**	**Delay** $[ns]$	**LD_eff**	χ	χ^{α}	**Vdd** $[V]$	**Vth0** $[V]$	**Pdyn** $[\mu W]$	**Pstat** $[\mu W]$	**Ptot** $[\mu W]$
RCA basic	640	897	6655.9	0.518	5.99	179	0.390	0.211	1.0	0.353	732.9	8.5	741.38
RCA parallel 2	1289	1771	13495.0	0.274	6.08	91	0.258	0.107	1.0	0.353	834.6	16.9	851.49
RCA parallel 4	2644	3574	26803.4	0.136	6.14	46	0.171	0.054	1.0	0.353	895.6	33.7	929.29
RCA horiz. pipeline 2	688	945	7329.8	0.413	4.13	123	0.311	0.146	1.0	0.353	678.8	9.4	688.18
RCA horiz. pipeline 4	816	1073	8745.7	0.338	2.95	88	0.254	0.104	1.0	0.353	735.1	11.3	746.31
RCA diag. pipeline 2	701	958	7427.5	0.432	3.67	110	0.290	0.129	1.0	0.353	722.1	9.5	731.60
RCA diag. pipeline 4	823	1082	8930.1	0.358	2.27	68	0.216	0.08	1.0	0.353	786.8	11.5	798.33
Wallace basic	789	1037	7119.0	0.347	2.45	73	0.227	0.086	1.0	0.353	542.5	9.3	**551.80**
Wallace parallel 2	1604	2068	14437.8	0.181	2.54	38	0.152	0.045	1.0	0.353	644.8	18.4	663.20
Wallace parallel 4	3252	4146	28627.6	0.091	2.61	19	0.102	0.023	1.0	0.353	717.5	36.7	754.15
Sequential basic	289	323	2565.1	2.888	0.86	411	0.645	0.485	1.0	0.353	2456.3	3.2	2459.49
Sequential-wallace	399	477	3590.3	1.080	3.09	369	0.605	0.436	1.0	0.353	1093.0	4.7	1097.71
Sequential parallel 2	594	628	4658.2	1.742	0.87	208	0.427	0.245	1.0	0.353	2230.8	5.7	2236.57

Table 7.1: Nominal values for thirteen 16 bit multipliers based on the STM 90nm technology and transistors of the SVT type at a throughput frequency of 62.5MHz

ModelSIM (from MentorGraphics). The results are based on the multiplication of uniformly distributed pseudo-random data during $2\mu s$; Standard library delays are used so that glitches can be accounted;

- Delay: the typical combinatorial delay from register output to register input on the critical path;
- LD_eff: the effective logical depth in equivalent NAND2 gates. The term "effective" is related to the fact that the length of the logical depth is considered against the throughput frequency or one-complete-multiplication frequency. In the case of a parallelization, for instance, LD_eff corresponds to half of the real LD because each block has two clock periods to compute one multiplication. Similarly, in the case of the sequential implementation, the LD_eff represents 16 times the real LD because to complete one multiplication, 16 1ns clock periods are required. The delay of the reference NAND2 gate has been estimated by building a 1000 NAND2 inverter chain. The inversion effect has been obtained by tying the two inputs together. The resulting delay per gate is 33.5ps for the SVT transistor type;
- χ and χ^{α}: these two parameters are obtained by using Eq. (6.3) from the nominal Vdd, Vth and delay. These parameters are reported there to be easily accessible during the following discussions;
- Nominal Vdd: the nominal technology supply voltage;
- Nominal Vth0: the nominal technology threshold voltage;
- Nominal Pdyn: the nominal dynamic power consumption as reported by Synopsys DC;
- Nominal Pstat: the nominal static power consumption as reported by Synopsys DC;
- Nominal Ptot: the nominal total power consumption obtained by summing the nominal Pdyn and the nominal Pstat.

With the data reported in Table 7.1, the optimal supply voltage Vdd and the optimal threshold voltage Vth can now be calculated. The values of Vdd and Vth in Table 7.2 are obtained in two different ways. In the first case, called numerical computation, a high resolution numerical search of the optimal supply and threshold voltage is used. This approach is very time consuming and requires the calculation of

a high number of total power consumption for a large amount of couple (Vdd, Vth) (100'000 in our case) using the non approximated equations described in Chapter 3. Moreover, such type of calculation doesn't permit to understand the real effect of each parameter on the final result. However, results calculated in this way are precise (up to the precision of models used) and for this reason they will be considered as a reference to be compared to the other approach which is based on Eq. (6.23) and Eq. (6.25) and is called analytical approximation. In this latter case, the optimal Vdd and the optimal Vth can easily be calculated from the values reported in Table 7.1.

The error between the reference data (numerical computation) and the analytical approximation is also reported in the same table. All the errors remains bounded to a few percent.

In Fig. 7.1 and Fig. 7.2, the same results are reported in a graphical manner, making it easier to read.

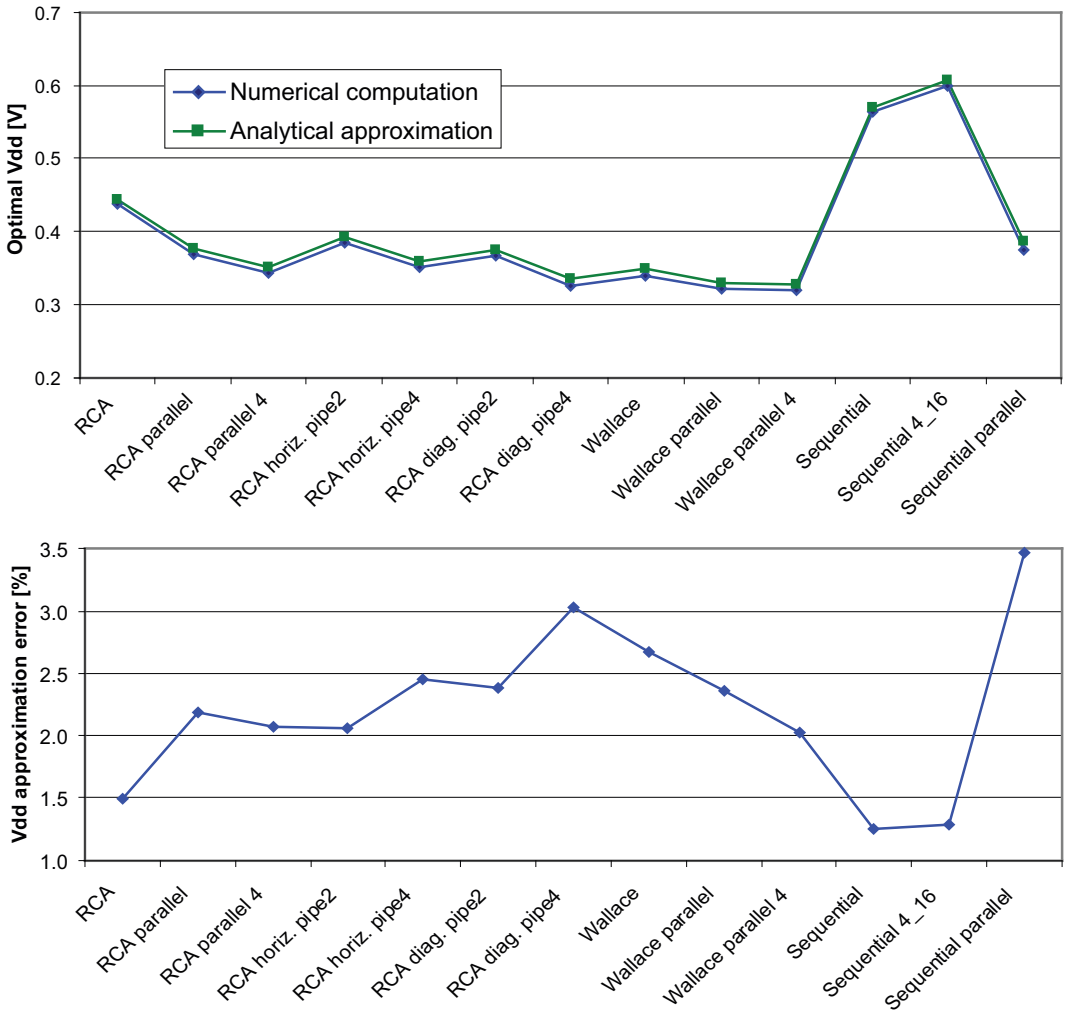

Figure 7.1: Optimal Vdd calculated with numerical computation (STM 90nm, 62.5MHz) using Eq. (6.25)

	Optimal values											
	Numerical computation		Analytical approximation		Approx. error		Numerical computation				Analytical approx.	Approx. error
	Vdd [V]	**Vth** [V]	**Vdd** [V]	**Vth** [V]	**Vdd** [%]	**Vth** [%]	**Pdyn** [μW]	**Pstat** [μW]	**Ptot** [μW]	**k1**	**Ptot** [μW]	**Ptot** [%]
RCA basic	0.437	0.201	0.444	0.206	1.5	2.8	140.08	41.62	181.70	3.4	183.16	0.8
RCA parallel 2	0.368	0.227	0.376	0.233	2.2	2.9	113.10	35.17	148.27	3.2	150.40	1.4
RCA parallel 4	0.344	0.254	0.351	0.260	2.1	2.4	105.83	32.01	137.84	3.3	140.05	1.6
RCA horiz. pipeline 2	0.385	0.211	0.393	0.217	2.1	2.8	100.53	31.72	132.25	3.2	133.99	1.3
RCA horiz. pipeline 4	0.351	0.216	0.360	0.223	2.5	3.3	90.47	29.76	120.23	3.0	122.27	1.7
RCA diag. pipeline 2	0.367	0.209	0.375	0.216	2.4	3.2	97.11	31.70	128.81	3.1	130.79	1.5
RCA diag. pipeline 4	0.325	0.216	0.335	0.223	3.0	3.4	83.15	28.67	111.82	2.9	114.37	2.3
Wallace basic	0.339	0.222	0.348	0.228	2.7	3.0	62.44	20.67	**83.11**	3.0	**84.64**	1.8
Wallace parallel 2	0.321	0.244	0.329	0.251	2.4	2.8	66.23	21.33	87.56	3.1	89.30	2.0
Wallace parallel 4	0.320	0.269	0.326	0.275	2.0	2.1	73.29	22.24	95.53	3.3	97.17	1.7
Sequential basic	0.563	0.107	0.570	0.109	1.3	2.0	777.27	237.90	1015.17	3.3	1045.75	3.0
Sequential-wallace	0.600	0.157	0.608	0.157	1.3	0.0	393.74	102.14	495.88	3.9	512.09	3.3
Sequential parallel 2	0.374	0.139	0.387	0.147	3.5	6.4	312.19	122.73	434.92	2.5	443.82	2.0

Table 7.2: Optimal Vdd, Vth and $Ptot$. These values are calculated once with a numerical computation and once using Eq. (6.25) for Vdd, Eq. (6.23) for Vth and Eq. (6.51) for $Ptot$. Relative errors are shown in the corresponding columns. Used A and B factors are for $Vdd \in [0.3V; 0.6]$

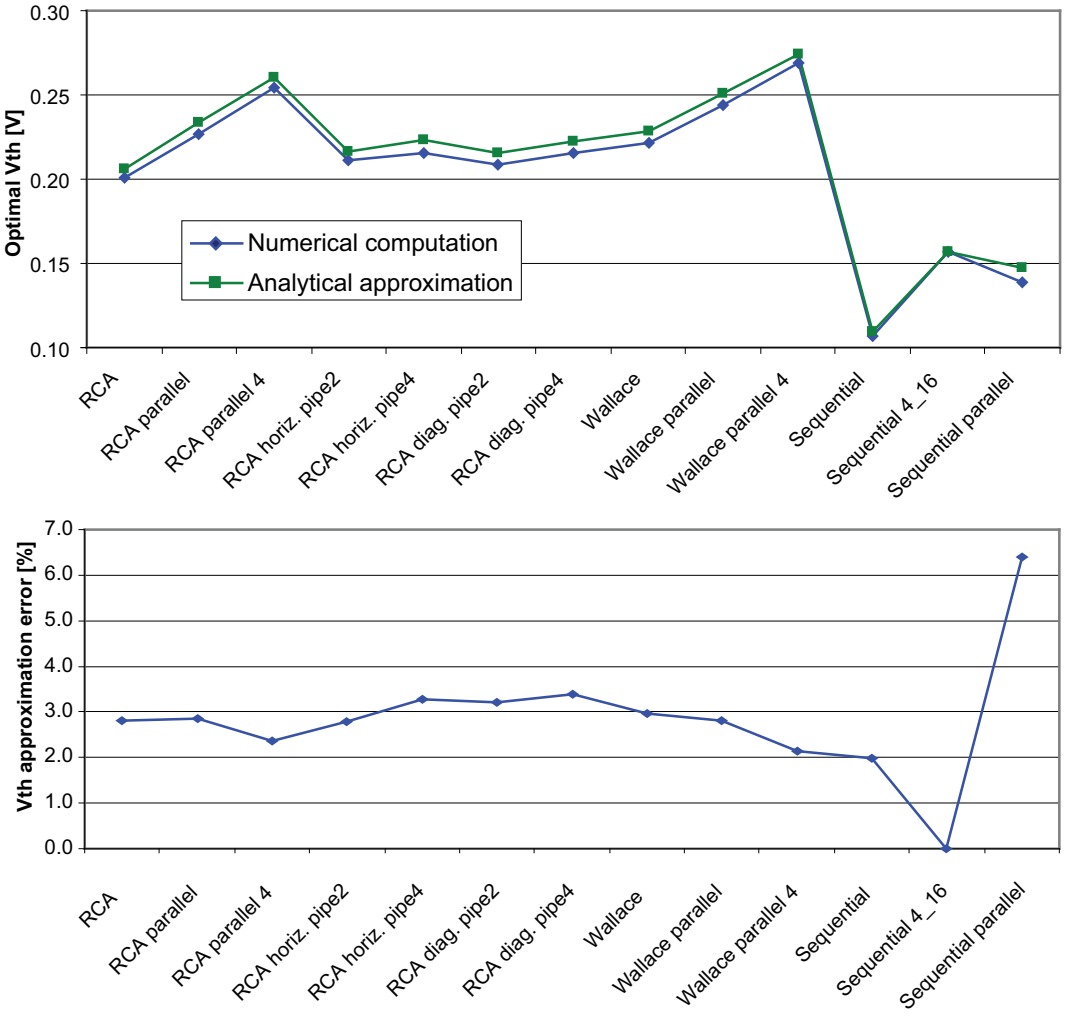

Figure 7.2: Optimal Vth calculated with numerical computation (STM 90nm, 62.5MHz) using Eq. (6.23)

What we can observe from the values of Vdd^{opt} and Vth^{opt} is, for instance, the effect of parallelization. In such a transformation, Vdd is reduced and Vth is increased. Both trends will favor a lower total power by reducing dynamic and static power at the same time. It is also interesting to note that the reduction of the supply voltage is less important for Wallace than for RCA. This can be easily explained by the lower χ factor of the Wallace implementation. In fact, being the Wallace already a quick architecture compared to the required frequency (62.5 MHz), the gain from the reduction of the effective logical depth (LD_eff) is only marginal, whereas it is much more consequent for the RCA multiplier.

It is also possible to observe that Vth is almost constant for the pipeline transformation as it was deduced in Chapter 6. Finally, the large delay involved in the sequential architectures (corresponding to a high χ) clearly shows a high Vdd and a low Vth, both negatively impacting the total power.

Nevertheless, optimal Vdd and Vth are not mandatory to compute the optimal total power consumption, thanks to Eq. (6.51). In fact, all required parameters can be obtained from Table 7.1 without needing intermediate steps. Once more, the results of our analytical approximation are compared to the numerical computation, where no approximations are applied. Results are reported in Table 7.2 with the corresponding errors. The same results are also provided in a graphical way in Fig. 7.3.

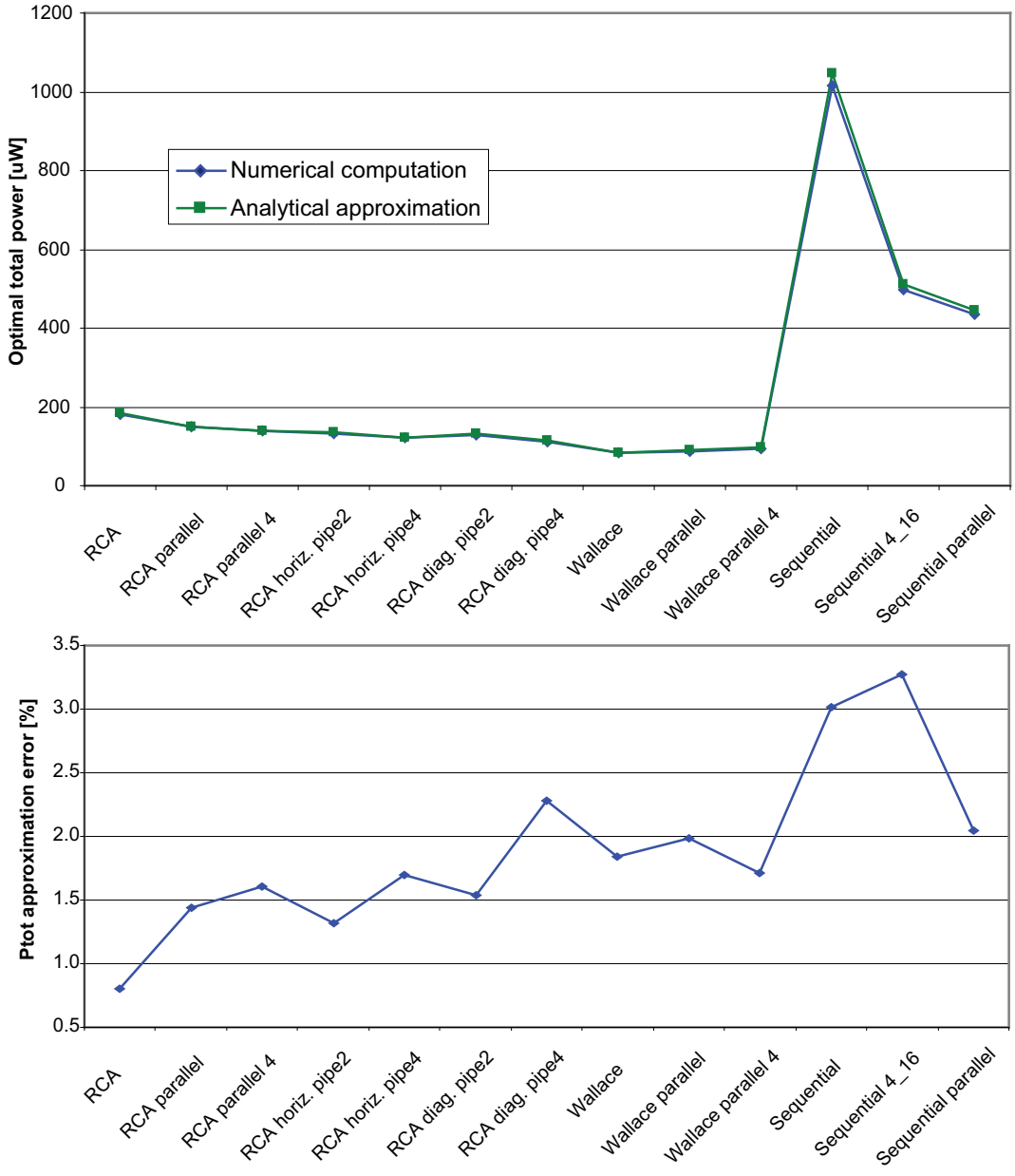

Figure 7.3: Optimal total power calculated with numerical computation (STM 90nm, 62.5MHz) using Eq. (6.51)

It is interesting to see that the errors for Eq. (6.51) over a set of so different implementations is always less than 3.5%. The second quite evident thing is the

huge optimal total consumption of the three sequential implementations compared to non-sequential ones. The explanation for this effect can be found by looking at the χ factor (Eq. (6.3)). This parameter, which establishes the relationship between the optimal Vdd and the optimal Vth, directly depends on the effective logical depth, which is very large for these three architectures. A large logical depth (i.e. a large χ) results in a high optimal Vdd (which increases the dynamic power in a square way and the static power linearly), and in a low optimal Vth (which increase the static power exponentially!). Moreover, sequential structures also present large activities. Because their activity is defined over a period of the throughput clock, it is not uncommon to observe activities higher than 1. Unfortunately, this high activity (a) is not counterbalanced by a small enough number of cells (N), which results in a much higher number of transitions ($a \cdot N$) compared to the others implementations. As stated in Eq. (6.51) a large number of transitions also penalize the optimal total power consumption.

The RCA architecture is based on a very regular structure that permits many variations to be implemented. Both parallelization and pipelining transformations shorten the effective logical depth (which correspond to a reduction of χ, although not proportionally). In this case, the benefit of the relaxed timing constraints permits to further reduce Vdd and increase Vth, reducing this way the optimal total power consumption.

The diagonal pipelined versions present a lower χ and a lower activity compared to the classical horizontal pipeline versions, and hence they feature a lower optimal total power consumption. Nevertheless, the gain in power between the two ways of pipelining is small, and the time spent by the designer to correctly implement a diagonal pipeline may not be worth the resulting gain in power.

Finally, the Wallace family presents the fastest circuits of our set. By applying a parallelization to the basic version, we observe that, similar to the RCA family, the logical depth is reduced and hence χ is also reduced. Once more, this results in a lower Vdd and higher Vth, which should be synonymous of power saving. However, if we look at the resulting optimal power we see that the Wallace basic version has a lower optimal total power compared to the two parallelized versions. The explanation comes from the fact that, the Wallace architecture being already a fast circuit (compared to the desired clock frequency), the reduction of χ obtained by parallelization is only marginal and its benefit is canceled by the increase of the static power due to the doubling in hardware and the overhead introduced to multiplex data. This is not the case for the RCA because its χ is higher. This example illustrates very well how the

same architectural transformation can yield completely different results. Fortunately, all these cases are well modeled by Eq. (6.51).

7.1 Summary

In this chapter we have shown how the architectural parameters like activity a, logical depth LD and frequency f can modify the optimal supply voltage Vdd, the optimal threshold voltage Vth and finally the optimal total power $Ptot$ of a design. In particular, we have pointed out how sequential circuits, characterized by very slow architectures (large LD), really present a huge power consumption compared to the other designs. Hence, unless a circuit working at extremely low frequency is needed, sequential implementations are not well suited for low power when working at the optimal point.

On the other hand, fast circuits (showing a short LD) like Wallace are not interesting for parallelization because the large increase of static consumption, caused by the hardware replication, easily cancels the poor benefit obtained from the reduced critical path.

For an architecture with an average logical depth like the RCA, we can observe that a moderate power gain can be obtained through parallelization, but even in this case, pipeline transformation reports better results with a much smaller area, which also correspond to lower production costs.

This leads us to the conclusion that, in designs where the static power consumption in not negligible, parallelization is rarely a good choice and most of the time pipelining should be preferred.

Chapter 8

Technology impact on total power

As explained in Chapter 6, the optimal total power not only depends on architectural parameters, but it also depends on technology parameters. In the past, it was in general not possible to change these parameters, because the designer had a given technology to use and was not able to modify them. This may change in the future. Until now, new technology nodes always presented better performances and a better power characteristics compared to the precedent ones, but nowadays, with the high increase in leakage current, performance gain can correspond to a power lost. For this reason, the technologies start now to exist under different "flavors", which are in general characterized by their Vth. For instance, the technology used in this thesis presents three different types of transistors, namely Low Vth (LVT), Standard Vth (SVT) and High Vth (HVT). Moreover, two of these three kinds can be implemented together on the same chip. Under such conditions, it is interesting to determine, between the proposed flavors, the best suited for a required work. Before that, we will consider the virtual case where the technology parameters could be freely modified in an independent way. This will permit us to understand the influence of each parameter to the optimal total power.

8.1 Technology as a free parameter

In general, technology parameters (I_0, n, α, k_t, C) are not independent and the variation of one of them results in a variation of others. Nevertheless, to understand the importance and the effective influence of a specific parameter, it is useful to observe how the total power is modified by single parameter variations. This is shown in Fig. 8.1 for a RCA 16 bit multiplier. The nominal case (no technology parameters variations) corresponds to the RCA basic structure reported in Table 7.1.

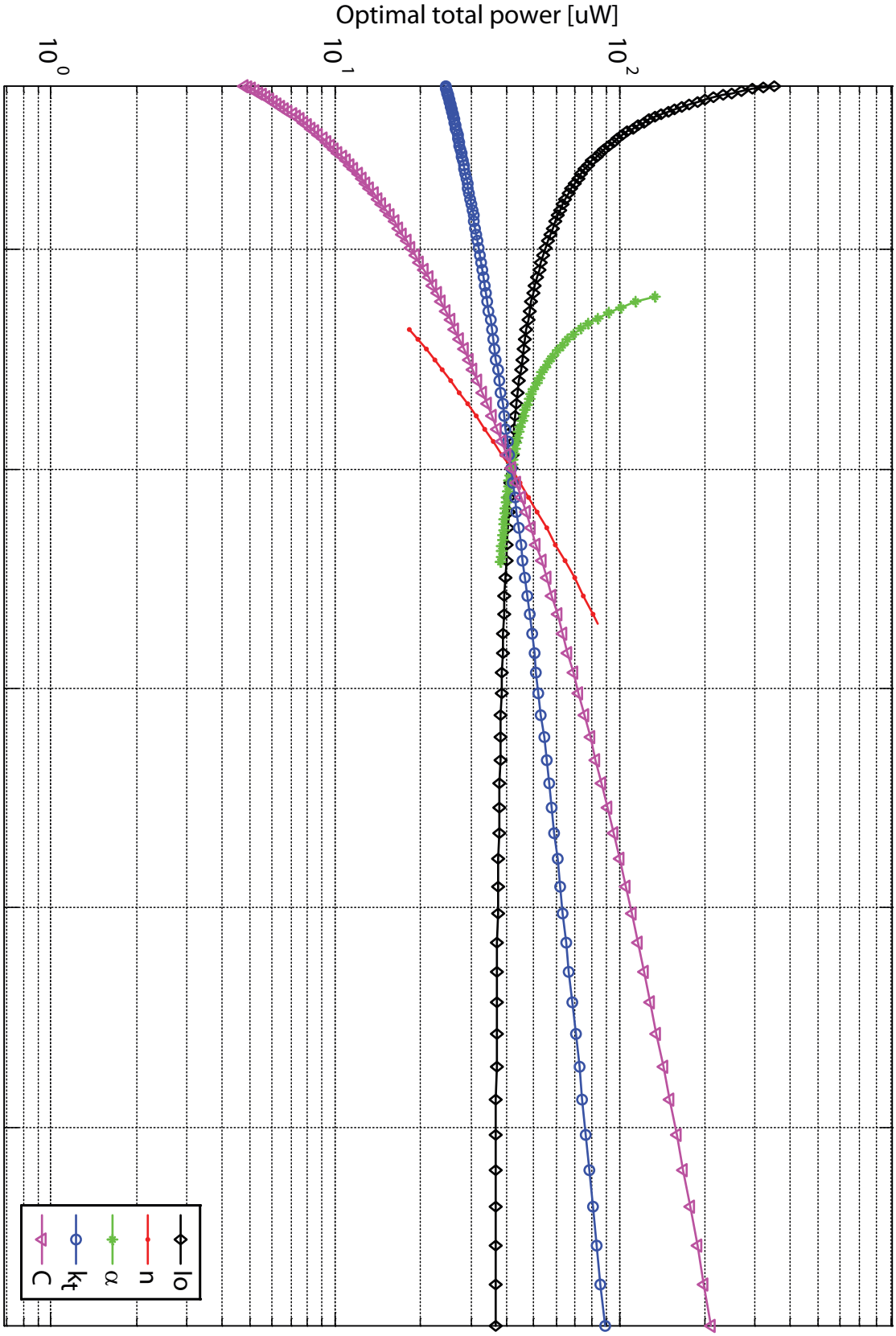

Figure 8.1: Technology parameters influence on a RCA 16 multiplier in a SVT STM 90nm technology

The abscissa represents the ratio of the new (modified) parameter over the original one, while the ordinate represents the optimal total power consumption.

The most sensitive parameter is α. This parameter comes from the alpha power law fitting formula and it represents the velocity saturation of electrons/holes. Typically, switching to a newer (finer) technology corresponds to a lower α. From Fig. 8.1, we can see how this is penalizing for the optimal total power. Actually, a low α will correspond to a reduced I_{on} current, which also means a slower technology. In practice, the speed reduction caused by α is largely counterbalanced by the reduced capacitances and k_t.

Moreover, it is interesting to observe that an increase of I_0, results in a very moderate power saving. The explanation comes from the fact that a bigger I_0 not only increases the static power, but also increases the on current by the same amount. Hence, it results that the speed related parameter χ is reduced, achieving a moderate gain. Conversely, the reduction of I_0 can highly penalize the total power. Once again, the delay increase easily explains this behavior.

The behavior of the capacitance C or delay parameter k_t is not really surprising. In fact, an increase of C means an augmentation of the delay (like for k_t) and so a worst optimal total power.

Finally, the curve of n shows a important increase of the optimal total power for an increase of the parameter and vice-versa. In fact, an increase in the factor n is equivalent to a reduction of Vth, i.e. an increase of the leakage current.

To summarize, the ideal technology would be characterized by a low C, k_t and n, whereas I_0 and α should be as high as possible. This may not be the trend in coming technologies, for instance in the case of α.

8.2 Application to technology selection

The 90nm technology from ST Microelectronics is available with 3 different transistor types (LVT; SVT; HVT). The optimal total power consumption for the 13 multipliers of Chapter 5 has been calculated for all existing flavors. Table 8.1 shows the results. By looking at the bold values, which represent the best technology choices for a given architecture, we can see that the best transistor type is not always the same. In particular, the HVT is the best for 6 cases, the SVT the best for 5 cases and the LVT is the best for 2 cases.

To better illustrate these results, they have been plotted in Fig. 8.2. Data corresponding to the sequential versions are omitted to permit a better reading of the

	Optimal Ptot [μW]		
Design Name	**LVT**	**SVT**	**HVT**
RCA basic	197.43	**181.70**	182.11
RCA parallel	179.39	**148.27**	152.53
RCA parallel 4	176.46	137.84	**135.16**
RCA horiz. Pipeline 2	151.93	132.25	**128.06**
RCA horiz. Pipeline 4	142.77	120.23	**113.34**
RCA diag. Pipeline 2	143.44	**128.81**	129.81
RCA diag. Pipeline 4	136.82	**111.82**	112.03
Wallace basic	**80.26**	83.11	96.95
Wallace parallel	104.17	87.56	**81.13**
Wallace parallel 4	121.57	95.53	**85.98**
Sequential basic	1547.98	1015.17	**1007.49**
Sequential-wallace	**358.37**	495.88	483.10
Sequential parallel 2	620.49	**434.92**	486.46

Table 8.1: Optimal total power consumption of thirteen 16 bit multipliers in all STM 90nm technology flavors. The bold values represent the best technology choice for the given architecture.

other cases.

Looking at the data for the three Wallace implementations, we can observe the effect of parallelization in different technology conditions. If we consider the HVT type (high Vth, hence low static power), we see that the parallelization of the basic implementation is interesting from a power point of view because doubling the hardware (so doubling the static power) is not so negative compared to reduction of the supply voltage and the increase of the threshold voltage coming from the relaxed timing constraints. Nevertheless, if the transformation is iterated one more time, leading to the Wallace parallel 4, the power figure is now starting to degrade, because Vdd and Vth are now only slightly modified, whereas the static power is doubled compared to Wallace parallel 2.

In the case of the SVT (standard Vth) the 2 times parallelization is already a bad transformation for low power, getting even worst in the 4 times parallelized version. This can be explained by a greater static power compared to the HVT, which penalize all types of parallelization for the Wallace structure.

Finally, the results for LVT (low Vth, hence high static power) clearly show an important increase of the optimal total power for each parallelized version. Once more, it is the doubling (or multiplying by 4) of the hardware that cannot be tolerated in a

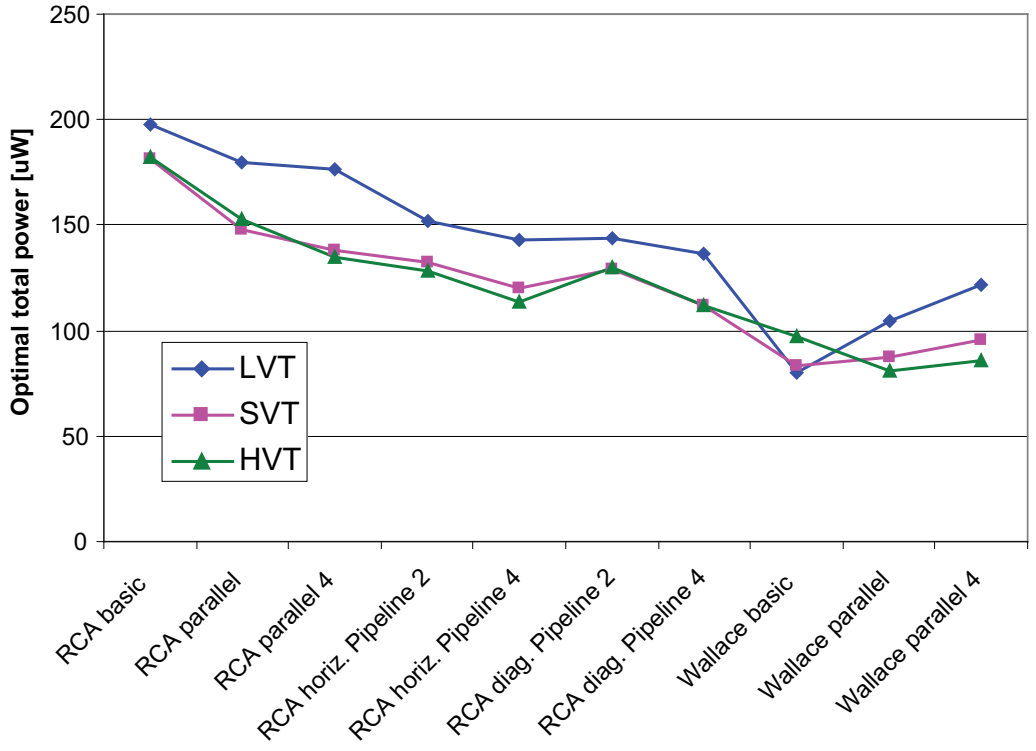

Figure 8.2: Optimal total power consumption of ten 16 bit multipliers in all STM 90nm technology flavors

flavor with so much leakage.

On the other hand, the parallelization of the RCA family remains interesting for all the three transistors types. This can be explained by the fact that the RCA multiplier has a longer logical depth and hence a higher χ compared to the Wallace. For this reason, the parallelization has a much important effect on the reduction of Vdd and the increase of Vth which can overcome the increase of hardware and hence of static power.

From Fig. 8.2, it is also possible to note that the pipeline transformations on the RCA multipliers present a better power consumption compared to the parallelized versions. This comes from the fact that pipelining can reduce the timing constraints without the need of doubling the static power due to hardware replication. It is hence possible to conclude that for technologies characterized by important leakage power, this situation being probably representative of all future technologies, pipelining needs to be preferred over parallelization. This also needs to be understood by the CAD programmers in order to include powerful automated pipelining tools that will replace the present massively parallelization-based algorithms.

Considering all the architectures and transistors types, the best choice for a frequency of 62.5MHz is the Wallace basic implemented with a LVT transistor flavor.

8.3 Discussion on the modifiability of Vth

All the theory developed in the last chapters considers Vth as a freely modifiable parameter. This is not the way people normally think about the threshold voltage, probably because the modification of Vth is not an easy task. In the precedent section, we discussed the possibility to select the best technology flavor from a set of given ones. This does not allow a continuous modification of the Vth, but still permits to modify it in a discrete way. An important drawback of such an approach is that the Vth cannot be dynamically modified to follow the various runtime needs. In this section, two other possible ways to interact with the threshold voltage are presented.

8.3.1 Body biasing

In Chapter 2, we discussed the body effect showing how a voltage between the body and the source of a transistor (Vbs) can modify the threshold voltage. The body biasing equation is replicated there:

$$Vth = Vth0 - \eta Vds - \gamma Vbs \tag{8.1}$$

With η the DIBL effect coefficient and γ the body bias coefficient.

This is clearly a simplification of the relationship between Vth and Vbs, but it is useful to understand the principle. In a more precise way, the body bias can be modeled by [57]:

$$Vth(Vbs) = Vth(Vbs = 0) - \frac{\sqrt{2q\epsilon_S N_A}}{C_0}\left(\sqrt{2\psi_B + Vbs} - \sqrt{2\psi_B}\right) \tag{8.2}$$

With q the elementary charge, ϵ_S the silicon permittivity, N_A the acceptor impurity density in the channel, C_0 the gate oxide capacitance per unit area, ψ_B the Fermi potential and Vbs the voltage between body and source.

From Eq. (8.2), we observe that the ability to modify Vth is more efficient for Vbs near zero, whereas it decreases in a typical square root way for larger values of Vbs. Moreover, the pre-factor $\sqrt{2q\epsilon_S N_A}/C_0$ tends to be smaller with newer technologies due to the reduction of the oxide thickness and hence the range where Vth can be modified will tend to be reduced on all new technology nodes.

Another important point is the sign of Vbs. In fact, the body can have a potential higher or lower than the source. When the body potential is higher than the source for the NMOS and lower than the source for the PMOS, the polarization is called forward

body biasing (FBB) and it corresponds to a reduction of the threshold voltage. The contrary, i.e. the body potential lower than the source for the NMOS and higher than the source for the PMOS, is called reverse body bias (RBB) and results in an increase of Vth.

If the RBB have no limit on the maximal Vbs other than the maximum reverse-bias junction potential, this is not the case for FBB. In FBB, if the potential goes over 0.5V the p-n junction between body and source will start to conduct, creating a very high current flow. For this reason, FBB always needs to be lower than 0.5V.

Just as an example, a FBB of 0.5V (the maximum applicable) on the 90nm STM SVT technology shows a Vth reduction of only 40 mV, whereas the same FBB correspond to a Vth variation of 60mV for the 130nm STM technology.

H. Ananthan & al. showed in [58] [59] that the FBB has the advantage to reduce the sensitivity of Vth to variations in gate length, oxide thickness and channel doping and it is hence preferable to RBB.

The principles of body bias has been successfully applied in circuits like the 150MHz discrete cosine transformation core processor of Kuroda et al. [60], the 200MHz processor of Mizuno et al. [13] and the 1Ghz router of Narendra et al. [61]

8.3.2 Transistor size modification

Another way to modify the threshold voltage of a transistor is by modifying its physical dimensions. The important dimensions of the transistor are the width (W) and the length (L) of its channel.

Fig. 8.3 (NMOS) and Fig. 8.4 (PMOS) show the plots of Vth versus W for the 130nm STM (HCMOS9GP_LL) technology. These graphs are part of the STM documentation and the details on how to generate them are not known. Nevertheless, these plots are very useful to understand the behavior of Vth under a transistor resizing.

From these graphs, we can remark that the influence of W to the Vth presents a huge asymmetry between the NMOS and the PMOS transistors. In fact, for instance, the maximal change of Vth due to a modification of W from 0.3 μm to 10 μm (which is a very large modification) corresponds to about 60 mV for the NMOS, whereas it is of only 6-7 mV for the PMOS. This means that any scaling of the device will create a completely unbalanced charging/discharging delays that will result in high shortcut currents, not mentioning the capacitances increase due to bigger channel area. Although the modification of channel width is probably not the best technique to modify the Vth, it is reported here for completeness.

The other modifiable size of the transistor is the channel length. In Fig. 8.5

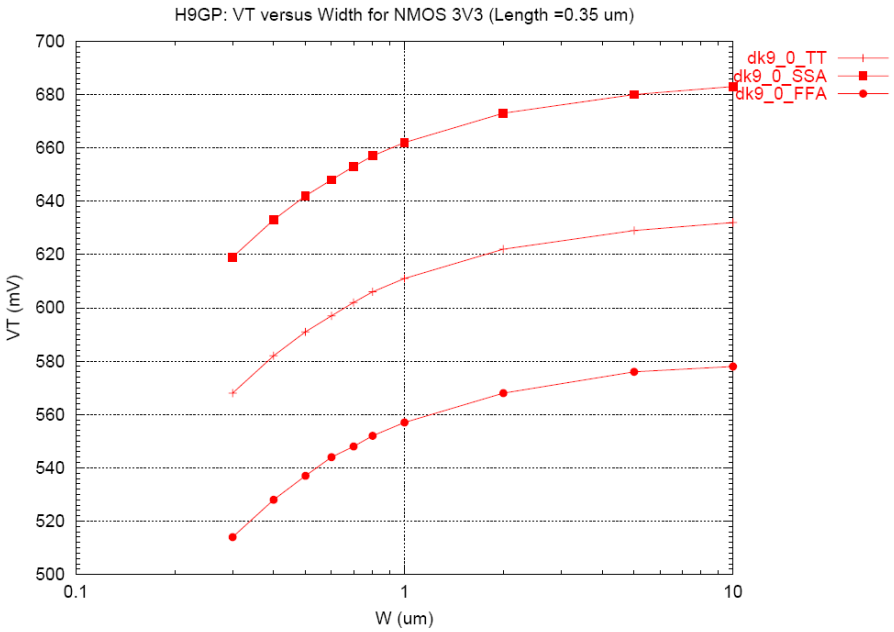

Figure 8.3: Vth vs. W for a NMOS transistor. Curves correspond to Slow-Slow(SSA), Typical-Typical(TT) and Fast-Fast(FFA) corners

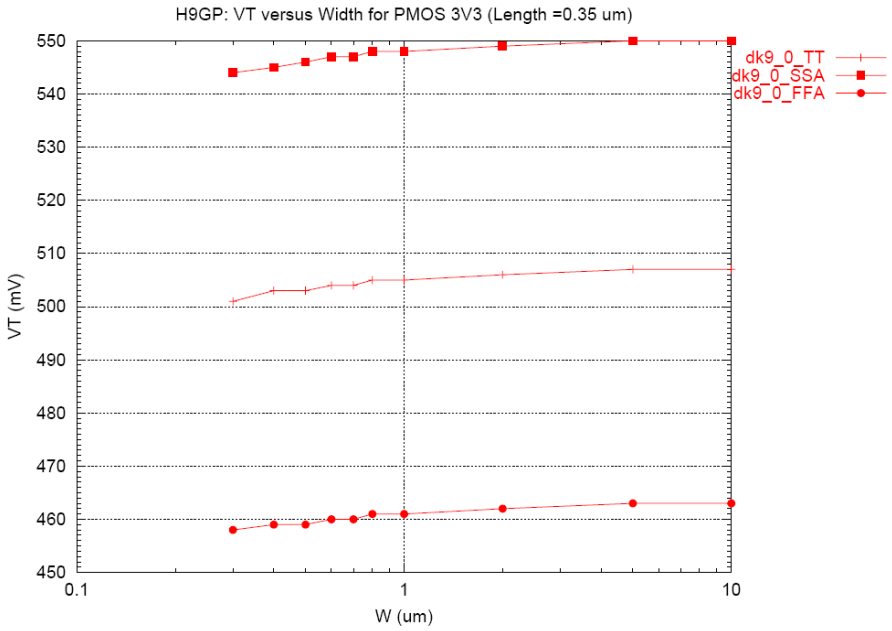

Figure 8.4: Vth vs. W for a PMOS transistor. Curves correspond to Slow-Slow(SSA), Typical-Typical(TT) and Fast-Fast(FFA) corners

(NMOS) and Fig. 8.6 (PMOS) the curves of Vth versus L are plotted for the HCMOS9GP_LL 130 nm STM technology. There, we can see that for small increases of the channel length, both NMOS and PMOS behave in a similar way with a relative steep slope. This is exactly the idea exploited by Gupta et al. [62]. What they

propose is to slightly increase (less than 10%) the transistors length L of devices that are not on the critical path, achieving a static power reduction of about 30% and delay penalty smaller than 10% in a 130nm technology.

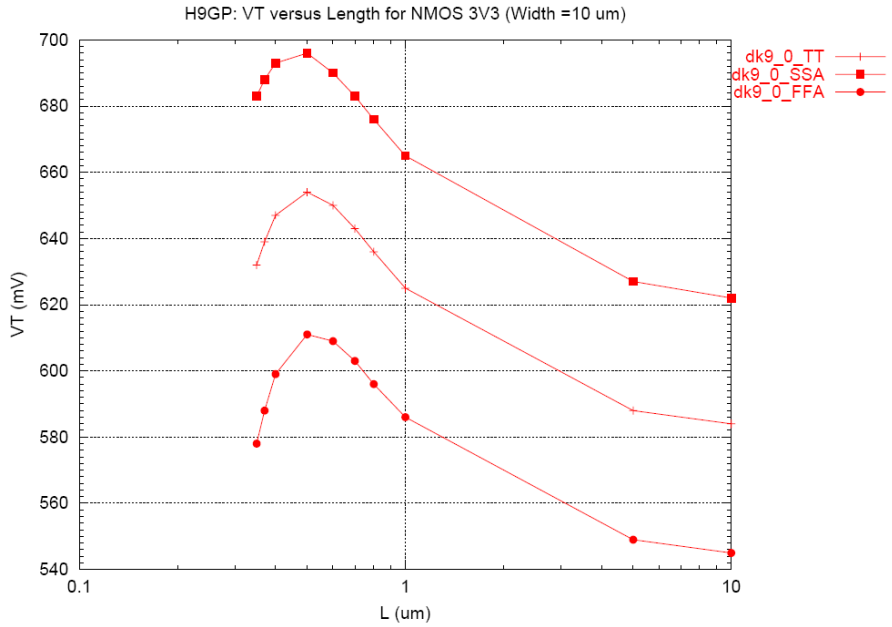

Figure 8.5: Vth vs. L for a NMOS transistor. Curves correspond to Slow-Slow(SSA), Typical-Typical(TT) and Fast-Fast(FFA) corners

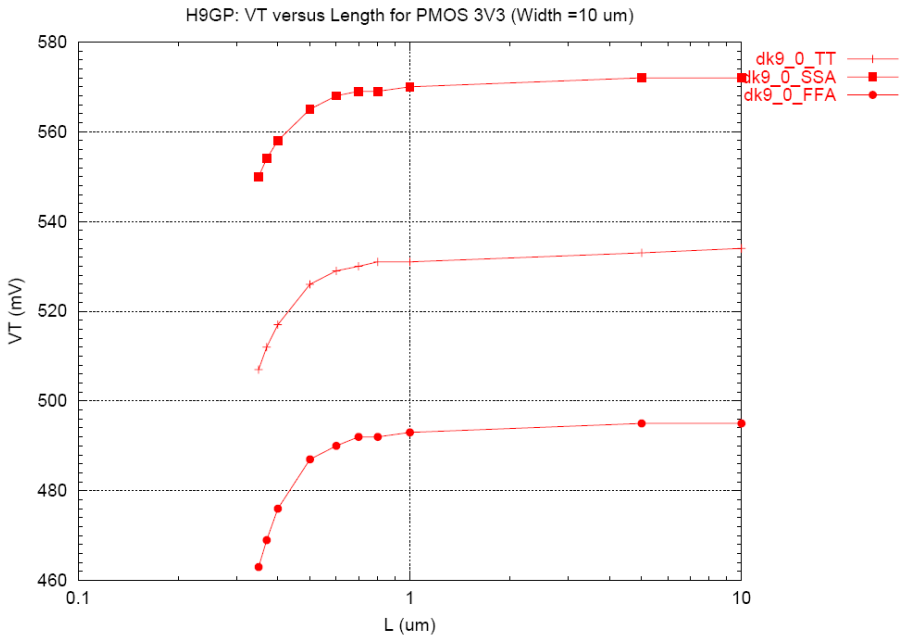

Figure 8.6: Vth vs. L for a PMOS transistor. Curves correspond to Slow-Slow(SSA), Typical-Typical(TT) and Fast-Fast(FFA) corners

It is also important to note that transistor size modifications influence more parameters than simply the threshold voltage Vth and the obtained Vth modifications are very moderate. For these reasons, technology flavor selection and body bias are preferable techniques to use for modifying the sub-threshold voltage Vth.

8.4 Summary

In this chapter we have discussed the influence of the principle technology parameters on the optimal total power. In particular, we have observed that an ideal technology would be characterized by low C, k_t and n, whereas I_0 and α should be as high as possible. Unfortunately, this will probably not be the trend of the future technologies.

Then we have analyzed thirteen different 16 bit multipliers synthesized in the three different technology flavors proposed by the STM 90nm technology. This illustrates very well how the technology can be used as a design parameter to achieve the lowest possible total power consumption. In the examples proposed, the best architecture/technology flavor is the Wallace basic in a LVT transistor type.

Finally, other two methods for modifying the sub-threshold voltage are proposed; namely body bias and transistor resizing. For both techniques, advantages and limitations have been discussed.

Chapter 9

Total power comparison for fixed Vdd and fixed Vth

This chapter presents a new methodology allowing to compare several architectures performing the same function and to select, among them, the one presenting the lowest total power consumption under fixed supply voltage (Vdd), threshold voltage (Vth) and frequency (f) constraints. This situation is much more common to designers than the one proposed in Chapter 6, because most of the time they cannot choose the technology to use. Moreover, this approach could be applied in parallel to the free Vdd/Vth one. Actually, the best Vth and Vdd could be chosen for the main block of the design and all the others will need to adapt. Thanks to the theory of this chapter secondary blocks can be optimized, too.

The lowest total power consumption, which is closely related to the architecture, results clearly from a trade-off between static and dynamic power. Static power reduction leads to the selection of architectures with a small number of cells and not with a small number of transitions, as it was the case when only dynamic power reduction was targeted. As an example, this methodology is applied to the selection of the lowest power consuming architecture among a set of thirteen 16 bit multipliers (described in Chapter 5). Moreover, by understanding the mechanism behind this selection, it is possible to propose and implement new architectures that will consume even less power as reported in Section 9.4.

9.1 Total power comparison

To be able to compare the consumption of two architectures under the same supply voltage Vdd, threshold voltage Vth and frequency f, we need a definition of the total

power. Once more, the used equation is the one described in Chapter 3.

$$Ptot = Pdyn + Pstat = aCNfV_{dd}^2 + NV_{dd}I_0e^{-\frac{Vth}{nUt}} \tag{9.1}$$

The equivalent capacity is roughly related to the average cell capacitance and could be obtained by dividing the dynamic power consumption by the number of transitions ($a \cdot N$), the squared supply voltage and the working frequency. Therefore C is not exactly the same for two circuits implementing the same function because it varies with their respective distribution of activity and capacitance products over the nodes. The same observation holds for the leakage current I_0, which represents an average static consumption per cell over the entire circuit, although some cells clearly involve more leakage than others. Considering that the methodology presented here is applied to the comparison of architectures performing the same task, we assume that the equivalent capacitance C and the average leakage current I_0 remain sufficiently similar across the set of architectures.

All the architectures in the implementation set share the same Vdd, Vth and f, but present different values for a (activity) and N (number of cells). Two architectures are characterized by $a1$ and $N1$, and $a2$ and $N2$ respectively, and their total power consumption can be compared as follows:

$$a_1N_1CfV_{dd}^2 + N_1V_{dd}I_0e^{-\frac{Vth}{nUt}} \overset{?}{<} a_2N_2CfV_{dd}^2 + N_2V_{dd}I_0e^{-\frac{Vth}{nUt}} \tag{9.2}$$

The inequality (9.2) is true if the first architecture consumes less power than the second one. This equation can be rewritten in the form:

$$(N_1 - N_2) \overset{?}{<} -(a_1N_1 - a_2N_2)\frac{CV_{dd}f}{I_0e^{-\frac{Vth}{nUt}}} \tag{9.3}$$

Then, by defining the difference between the number of cells as $\Delta N = (N_1 - N_2)$ and the difference between the number of transitions as $\Delta Tr = (a_1N_1 - a_2N_2)$, we can finally express this comparison as:

$$\Delta N \overset{?}{<} -\Delta Tr\frac{CV_{dd}f}{I_0e^{-\frac{Vth}{nUt}}} \tag{9.4}$$

$$\Delta N \overset{?}{<} -\Delta Tr \cdot R(V_{dd}, V_{th}, f) \tag{9.5}$$

The expression $R(V_{dd}, V_{th}, f)$ in Eq. (9.5) depends on Vdd, Vth, f and some technology parameters, which are imposed to the designer and are hence constant. Moreover, the value of R is always positive.

Eq. (9.4) shows that the comparison of the total power consumption between two architectures depends on the difference between the number of cells (ΔN) and on the difference between the number of transitions (ΔTr). This is quite different from the conventional approach where only the number of transitions is relevant as only dynamic power consumption is taken into account.

9.2 Comparison of two architectures

A logical function can be implemented in several ways, using different topologies, for instance by parallelizing, pipelining or performing algorithmic improvements. All these various structures can be categorized based on their characteristics: number of cells, logical depth, number of transitions and activity (Table 7.1 is an example of such a classification). Two architectures can lead to positive or negative ΔN and ΔTr values while the value of R (Eq. (9.5)) is always positive. If both designs present the same amount of cells and transitions (i.e. $\Delta N = 0$ and $\Delta Tr = 0$), the power consumption will clearly be the same. An architecture with more cells and more transitions will always consume more power, because inequality (9.5) becomes trivial, i.e. independent of R. Conversely, if one design has more cells but less transitions compared to the other (i.e. $\Delta N > 0$ and $\Delta Tr < 0$ or vice versa), the choice of the architecture consuming less power is more complex and depends on R. This means that the selection will depend on the working conditions too, i.e. on Vdd, Vth, f and the technology parameters. All possible cases are summarized in Table 9.1 .

	$\Delta Tr > 0$	$\Delta Tr = 0$	$\Delta Tr < 0$
$\Delta N > 0$	Circuit 2	Circuit 2	Depends on Eq. (9.5)
$\Delta N = 0$	Circuit 2	Same consumption	Circuit 1
$\Delta N < 0$	Depends on Eq. (9.5)	Circuit 1	Circuit 1

Table 9.1: Comparison table between two circuits having a difference of $\Delta N = (N_1 - N_2)$ cells and $\Delta Tr = (a_1 N_1 - a_2 N_2)$ transitions. The circuit indicated is the one presenting the lowest total power consumption

Plotting the lines of equal-consumption (i.e. $R(Vdd, Vth, f) = -\Delta N/\Delta Tr$) on the space ($Vdd$, Vth) allows a better understanding of the role of R in the architecture selection (Fig. 9.1). These equal-consumption lines delimit the points where two designs having the corresponding ratio $-\Delta N/\Delta Tr$ will present the same power consumption, despite the fact that the absolute value will vary with Vdd and Vth. For instance, if two architectures operating at Vdd=1 V and Vth=0.33 have $-\Delta N/\Delta Tr$

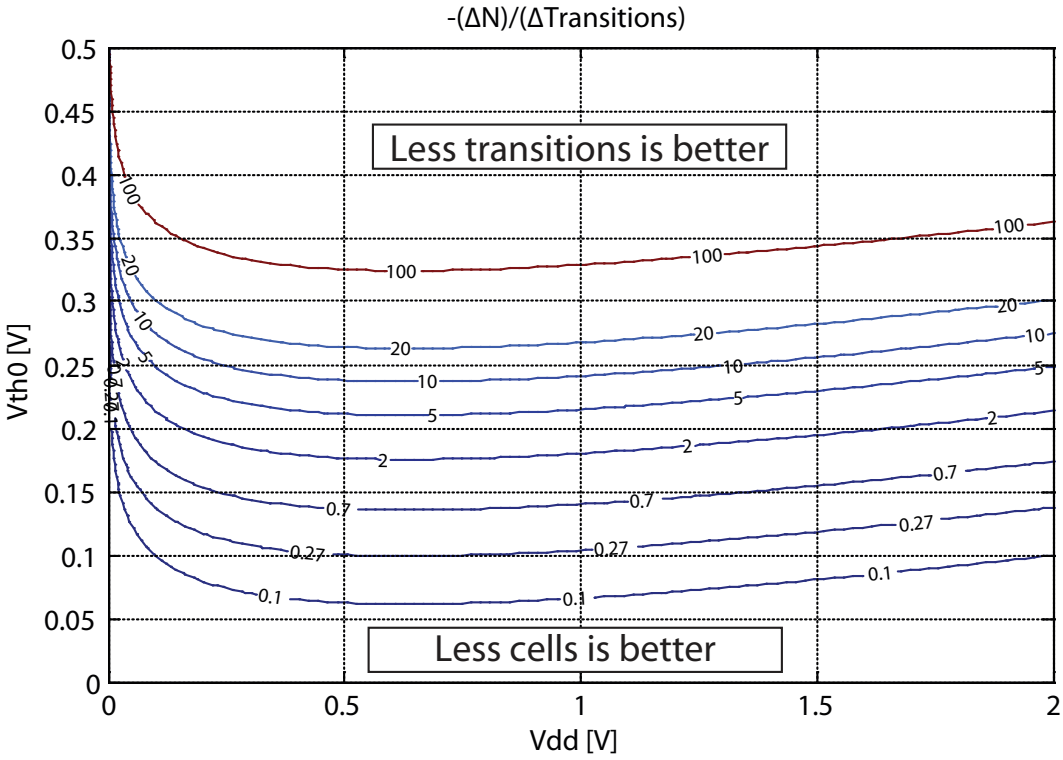

Figure 9.1: Lines of equal-consumption with f = 62.5MHz in a STM 90nm SVT technology. The Vdd and Vth constraints can be represented with a point on this plot. A pair of architectures to be compared corresponds to one $-\Delta N/\Delta Tr$ line in this space. If the working point is located above the $-\Delta N/\Delta Tr$ line, then the architecture with less transitions is better in term of power consumption, otherwise the design with less cells is preferred

= 100, they will present the same total power consumption. Otherwise, when the design constraints represented by Vdd and Vth correspond to a point that is above the equal-consumption line (which would be the case for Vdd=1V and Vth=0.4V in our example), the circuit with less transitions will dissipate less power. Conversely, if the working point is located below the equal-consumption line (which would be the case for Vdd=1V and Vth=0.2V), the design with less cells will consume less power. Actually, increasing Vth results in a large decrease in static power, which in turn leads to a consumption dominated by the dynamic contribution. The architecture with fewer transitions is then naturally preferred. It is important to remember that the plot of Fig. 9.1 depends on the technology used. Here, the STM 90nm SVT technology was chosen, which corresponds to an average C/I_0 of 1.36E-9 [s/V] and a working frequency of 62.5MHz.

9.3 Selection of the best architecture

The methodology illustrated in the precedent section to compare two architectures can be iterated over a large number of implementations of the same logical function. In this way, by repeating the comparisons on couples of structures, it is possible to eliminate the worst architectures and quickly converge to the best design for the specified Vdd, Vth and f constraints. It is important to note that the selected architecture is not always the same, but depends on the values of Vdd, Vth and f. This methodology can be used to easily select the better architecture under new constraints without re-synthesis. Generally speaking, the approach can be summarized as follows:

1. **Delay constraints:** Given Vdd, Vth, f, architectures that are too slow to meet the timing constraints are eliminated. A slow architecture can be parallelized or pipelined to meet the constraints, but this represents a new architecture to be added to the set of structures to compare.

2. **Compare a couple of architectures:** The comparison of two architectures is achieved using the parameter $-\Delta N/\Delta Tr$. If this value is negative the architecture with fewer cells and less transitions is chosen (circuit 1 or 2 in Table 9.1 when $-\Delta N/\Delta Tr$ is negative). On the other hand, when $-\Delta N/\Delta Tr > 0$, the choice depends on Eq. (9.5) and therefore on the position of the working point with respect to the line of equal consumption.

3. **Repeat step 2 for all remaining architectures:** It can be a good idea to start eliminating trivial cases ($-\Delta N/\Delta Tr < 0$) in order to reduce the number of non-trivial comparisons performed by using Fig. 9.1. Elimination of architectures will rapidly converge to a design presenting the overall lower total power consumption for the given working conditions (Vdd, Vth, f).

9.4 Designing new circuits

In addition to the above considerations, the same graphical tool can be used to define guidelines for the design of new architectures (i.e. not yet present in the set of available architectures) presenting an even smaller total power consumption. First, the $-\Delta N/\Delta Tr$ line that crosses the (Vdd, Vth) constraint point can be determined from Fig. 9.1. As a reminder, two architectures having this $-\Delta N/\Delta Tr$ share the same power consumption under these constraints, whereas the architecture with fewer cells should be favored when this $-\Delta N/\Delta Tr$ ratio is higher.

Starting from an existing design with N1 cells and Tr1 transitions, a new architecture with less cells ($N2 < N1$) can be searched for, which will usually present also more transitions (the trivial case where $N2 < N1$ and $Tr2 < Tr1$ would be in fact always better but rarely realizable). This new version with $N2 < N1$ cells and $Tr2 > Tr1$ transitions will consume less power, if and only if the ratio $-\Delta N/\Delta Tr$ is higher than the one extracted from the line crossing the (Vdd, Vth) constraints. Indeed, in this case this line will actually pass above the working point in Fig. 9.1 and the new design with fewer cells will consume less power. Conversely, an architecture presenting a reduced number of transitions (which in general will present more cells) can be searched for. In this case, the new structure should present a ratio $-\Delta N/\Delta Tr$ smaller than the one that can be read from the line crossing the (Vdd, Vth) constraints in Fig. 9.1.

As an example, an existing circuit with 10'000 cells and 100 transitions is working at Vdd=1V, Vth=0.24V and f=62.5MHz and a new architecture consuming less power is sought. Fig. 9.1 specifies that in order to consume less power a new architecture must have a $-\Delta N/\Delta Tr$ greater than 10 when reducing the number of cells, or smaller than 10 when reducing the number of transitions. Supposing that the designer can achieve a reduction of 1000 cells (N2 = 9000) by an architectural transformation, he should verify that the number of transitions of this new design is no more than 200 ($\Delta Tr < 100$), which is necessary in order to have $-\Delta N/\Delta Tr$ greater than 10.

When performing a parallelization, the number of cells is more than doubled (due to the multiplexer overhead) and the activity is reduced by slightly less than two. In general, this results in a small increase of the number of transitions and in a large increase in the number of cells. For this reason, parallelized versions will always present more power consumption than the original design at the same working conditions. However, when the original architecture does not meet the speed requirements, the parallelization can relax the timing constraints to achieve the required performances. This is the only case where a parallelized architecture may be useful when Vdd and Vth are fixed.

The same situation arises with pipelining where the overhead due to the extra registers often largely cancels the activity reduction achieved by suppressing glitches. At the same time, the number of cells increases due to the same overhead and, as a result, pipelining a circuit at the same working conditions is in general not interesting. Nevertheless, the pipelining technique can be used to reduce the logical depth and hence relax the timing constraints of circuits that do not meet the speed constraints at the required Vdd and Vth.

9.5 Case study: 16bit multipliers

To show how to apply the ideas of this chapter to a practical case, we will, one more time, refer to the thirteen 16 multiplier described in Chapter 5. The data of the architectural parameters for all the structures is available in Table 7.1.

Knowing that the key parameters for power discrimination are the number of cells (N) and the number of transitions (Tr), all architectures can be represented as points on a plot of N versus Tr (Fig. 9.2). The label on the arcs connecting points stands for the value of $-\Delta N/\Delta Tr$ for the corresponding couple of architectures. Fig. 9.2 allows a very easy detection of trivial cases characterized by $-\Delta N/\Delta Tr < 0$, as the slope of their arc is positive. Conversely, non-trivial cases present a negative slope. In Fig. 9.2, only non-trivial arcs are shown.

A. Example 1: Vdd = 1V, Vth = 0.4V, f = 62.5MHz

Applying the methodology described in section 9.3, we have:

1. **Delay constraints:** All design can work at these conditions.

2. **Compare a couple of architectures:** Architectures connected by a positive slope arc in Fig. 9.2, i.e. trivial cases such as RCA parallel 4 against Wallace parallel 2, are first considered. As RCA parallel 4 presents more cells and more transitions than Wallace parallel 2, it is eliminated.

3. **Repeat step 2 for all remaining architectures:**

   - By comparing other trivial cases, we can easily eliminate RCA horizontal pipeline 4, RCA diagonal pipeline 4, RCA parallel 2, Wallace parallel 2 and Wallace parallel 4 in favor of Wallace. Moreover the RCA diagonal pipeline 2 is eliminated in favor of RCA horizontal pipeline 2 and Sequential parallel in favor of the basic Sequential.

   - The remaining cases are then considered. Looking at RCA and Sequential in Fig. 9.2, it can be seen that the arc connecting the two structures is characterized by $-\Delta N/\Delta Tr = 0.7$. On Fig. 9.1, the equal-consumption line corresponding to this value splits the space in two regions with the label "less transition is better" on the upper part and "less cells is better" in the lower part, meaning that at Vdd=1V and Vth=0.14V the two designs will consume the same amount of power. However, in our example the working point corresponding to Vth=0.4V lies in the upper part of the plot where

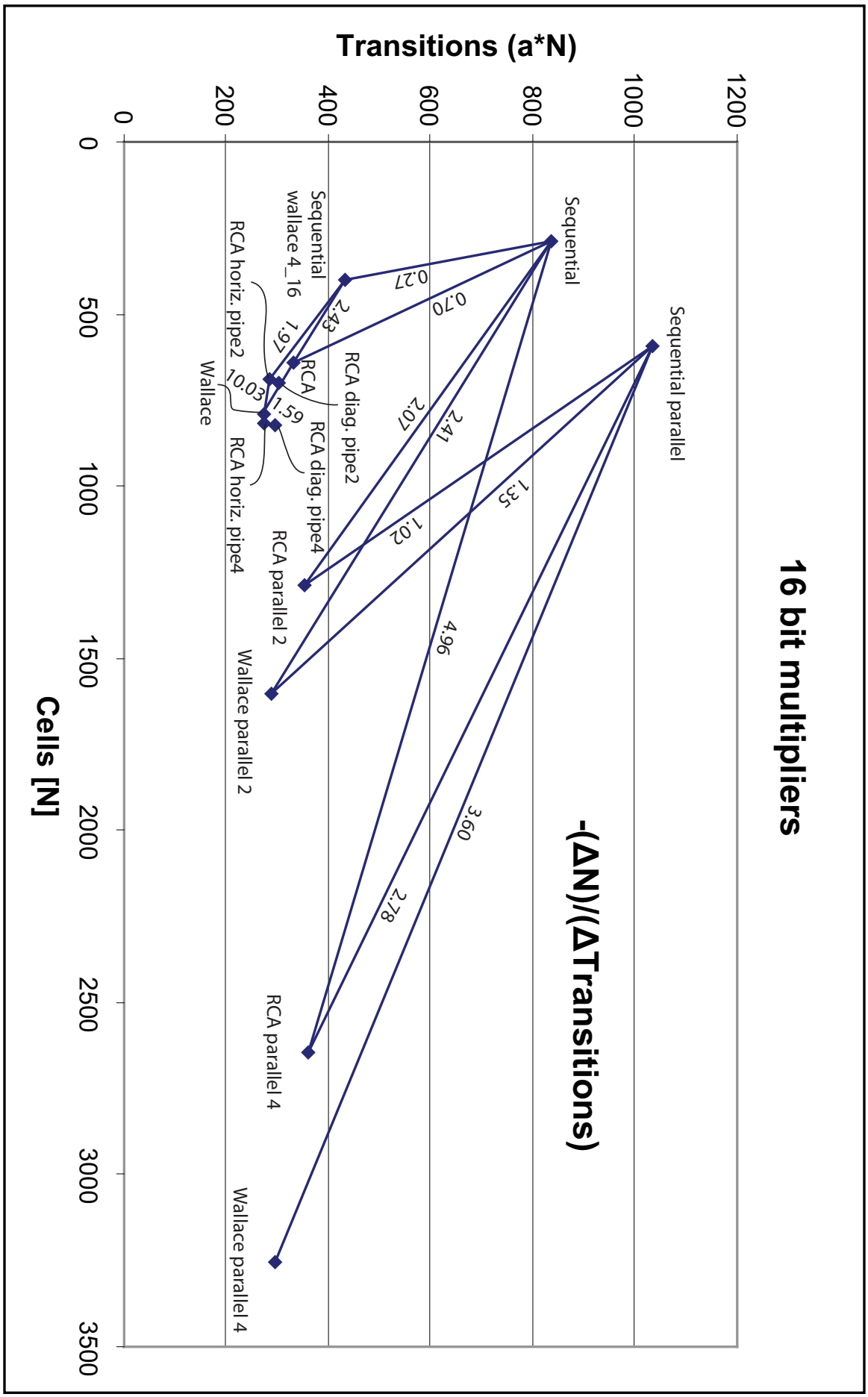

Figure 9.2: Thirteen 16 bit multipliers plotted on the cells vs. transitions space

the better structure is characterized by less transitions. Consequently, the RCA design is selected. The same reasoning can be applied to the Sequential-wallace 4_16 architectures which is eliminated in favor of the RCA. In fact, if the equal-consumption line is located in the lower part of Fig. 9.1, i.e. at low Vth, a working point above this line is dominated by dynamic consumption rather than static power. For this reason, designs with fewer transitions will present also less total power dissipation. The remaining architectures are RCA, RCA horizontal pipeline 2 and Wallace, but having all low values of $-\Delta N/\Delta Tr$ compared to Wallace (1.59 and 10.03 respectively) only the Wallace structure remains.

For Vdd=1V, Vth=0.4V and f=62.5MHz, the better architecture from a power point of view is the Wallace. In order to validate the methodology, the total power consumption of all designs was calculated for the given operating conditions and is shown in Table 9.2.

RCA	RCA par2	RCA par4	RCA horiz.pipe2	RCA horiz.pipe4
735.4	839.5	905.4	681.5	738.3
RCA diag.pipe2	RCA diag.pipe4	Wallace	Wallace par2	Wallace par4
724.9	790.1	**545.2**	650.2	728.1
Sequential	Sequential-wallace 4_16		Sequential parallel	
2457.2	1094.4		2232.5	

Table 9.2: Consumption of the thirteen multipliers in μW for Vdd=1V, Vth=0.4V and f=62.5MHz.

These values are first obtained at the nominal conditions (Vdd=1V, $Vth0$ = 0.353V) and then dynamic and static powers are separately recalculated based on Eq. (9.1) for the proposed working condition (i.e. Vdd=1V, Vth=0.4V).

B. Example 2: Vdd = 1V, Vth = 0.12V, f = 62.5MHz

As a second example, we choose a working condition with a very low threshold voltage (Vth=0.12V) and the same supply voltage and frequency as in the previous example.

1. **Delay constraints:** In this case too, all designs meet the timing constraints.

2. **Compare a couple of architectures:** As in the previous example, trivial cases are detected first. Hence, the RCA parallel 4 is eliminated in favor of the Wallace parallel 2.

3. **Repeat step 2 for all remaining architectures:**

   - By comparing other trivial cases, we can easily eliminate RCA horizontal pipeline 4, RCA diagonal pipeline 4, RCA parallel 2, Wallace parallel 2 and Wallace parallel 4 in favor of Wallace. Moreover, the RCA diagonal pipeline 2 is eliminated in favor of the RCA horizontal pipeline 2 and the Sequential parallel in favor of the basic Sequential.

   - The remaining architectures are: RCA, RCA horizontal pipeline 2, Sequential, Sequential-wallace 4_16 and Wallace. As before, the couple RCA and Sequential is characterized by $-\Delta N/\Delta Tr = 0.7$, which corresponds to an equal-consumption line on Fig. 9.1. For Vdd=1V these architectures will have the same power consumption if the threshold voltage is equal to 0.14V. As the imposed Vth is a little lower (0.12V), it is located in the region where less cells are preferred. Hence, the Sequential architecture will be selected. Similar is the comparison between the RCA horizontal pipeline 2 and the Wallace. With a $-\Delta N/\Delta Tr$ of 10.03, we know that the architecture with less cells is preferred (i.e. the RCA horizontal pipeline 2). For the same reason, the Sequential-wallace 4_16 will be preferred over the RCA horizontal pipeline 2. Finally, the comparison between the Sequential and the Sequential-wallace 4_16 is characterized by a $-\Delta N/\Delta Tr = 0.27$. From Fig. 9.1 we can see that the equal-consumption line passes under the working conditions couple (Vdd,Vth), meaning that the circuit with less transitions will present the best power figure. Hence, the only remaining architecture is the Sequential-wallace 4_16.

The results of the methodology indicate that the Sequential-wallace 4_16 is the circuit presenting the lowest total power consumption for Vdd=1V, Vth=0.12V and f=62.5MHz.

RCA	RCA par2	RCA par4	RCA horiz.pipe2	RCA horiz.pipe4
4618.5	8571.8	16342.5	4987.5	5898.4
RCA diag.pipe2	RCA diag.pipe4	Wallace	Wallace par2	Wallace par4
5070.3	6072.1	4788.5	9082.5	17537.0
Sequential	Sequential-wallace 4_16		Sequential parallel	
3939.4	**3245.1**		4857.9	

Table 9.3: Consumption of the thirteen multipliers in μW for Vdd=1V, Vth=0.12V and f=62.5MHz.

The actual power consumption in these conditions is shown (after calculation) in Table 9.3, confirming that the Sequential-wallace 4_16 presents actually the lowest total power consumption.

9.6 Summary

This chapter presented a new design methodology allowing the selection of the architecture presenting the lowest total power consumption within a set of equivalent designs working at the same (fixed) Vdd, Vth and f. This methodology considers dynamic power consumption (proportional to the number of transitions), as well as static power consumption (directly related to the number of cells). An example of application was reported for thirteen 16 bit multipliers, showing that, depending on the working condition (i.e. Vdd, Vth and f), the architecture with the lowest total power dissipation is not always the same. Moreover, this technique allows the determination of the architecture presenting the lowest total power consumption for conditions which are different from the one used during synthesis, without the need of re-synthesizing all the circuits.

Chapter 10

Physical implementation of four 32 bit multipliers

In the previous chapters, the models for the optimal total power consumption have been proposed. In order to validate the reported equations and to reinforce the drawn conclusions, a physical ASIC implementation has been done. The circuit was designed to demonstrate both architectural and technology influences to the optimal total power consumption in the case where the static power consumption also largely contribute to the total power. This has been achieved with a state-of-the-art 90nm technology from ST Microelectronics. The main advantage of this technology is the possibility to integrate, on the same die, 2 different kinds of transistor out of the 3 available. In this way, it is possible to "emulate" the effects of a technology change on the total power consumption with a single chip.

The implemented design is composed by two 32 bit multipliers (RCA basic and RCA parallel 4, these structures being described in details in Chapter 5) implemented once with the Standard Vth (SVT) transistors and once with the Low Vth (LVT) transistors, giving a total of 4 multipliers.

After a detailed description of the ASIC structure and functionality, this chapter will present the tools and resources used for the measurements. Then, measured data will be reported and commented. Finally, a discussion on technology parameter variations closes the chapter.

10.1 Circuit description

The test ASIC is mainly formed by 4 multipliers corresponding to all possible combinations of two technology flavors (SVT/LVT) with two architectures (RCA basic/RCA

parallel 4). The combinations are:

- **mult_0:** RCA basic with SVT transistor type;
- **mult_1:** RCA parallel 4 with SVT transistor type;
- **mult_2:** RCA basic with LVT transistor type;
- **mult_3:** RCA parallel 4 with LVT transistor type.

The choice of the RCA as the block to be implemented comes from the need to have an architecture "slow enough" (in fact, the RCA has a logical depth larger than the Wallace) to have the expected total power crosses (reported at the end of this chapter) at relatively low frequency (under 20MHz in this case). This permitted us to reduce the requirements for the testing tools. Fig. 10.1 illustrates the block diagram of the test circuit. All multipliers have a data size of 32 bit, which corresponds to 64 output bits. Each multiplier also has a separated power supply in order to be able to measure its power consumption without including the rest of the circuit. For the same reason, the clock signal was multiplexed to each block. In fact, in this way, only the clock tree corresponding to the desired multiplier is accounted during the power measurements. This clock multiplexing, as well as the multiplier register enables and the output demultiplexer are controlled by the external signal **sel**, which is the binary representation of the number corresponding to the multiplier under test.

To be able to verify the correct functioning of the multipliers over many multiplications, the results are added with the precedents and only the final sum is verified. Mathematically, the content of the shift register after n multiplications can be expressed by:

$$Sum = \left[\sum_{i=0}^{n} \text{multiplication(i)}\right] \ \textbf{mod}\ 2^{64} \tag{10.1}$$

This sum is stored in a 64 bit shift register which permits to serially output the result externally in order to be checked after the test.

10.1.1 Pseudo-random code generator

The circuit being designed to work at a maximal frequency of 62.5MHz (corresponding to 16ns of clock period) at nominal conditions (i.e. Vdd=1V), it was not possible to externally generate the input data for the multipliers due to the high throughput required. Hence, a pseudo-random data generator has been implemented internally. This generator is based on a linear feedback shift register [63] [64] and is constructed

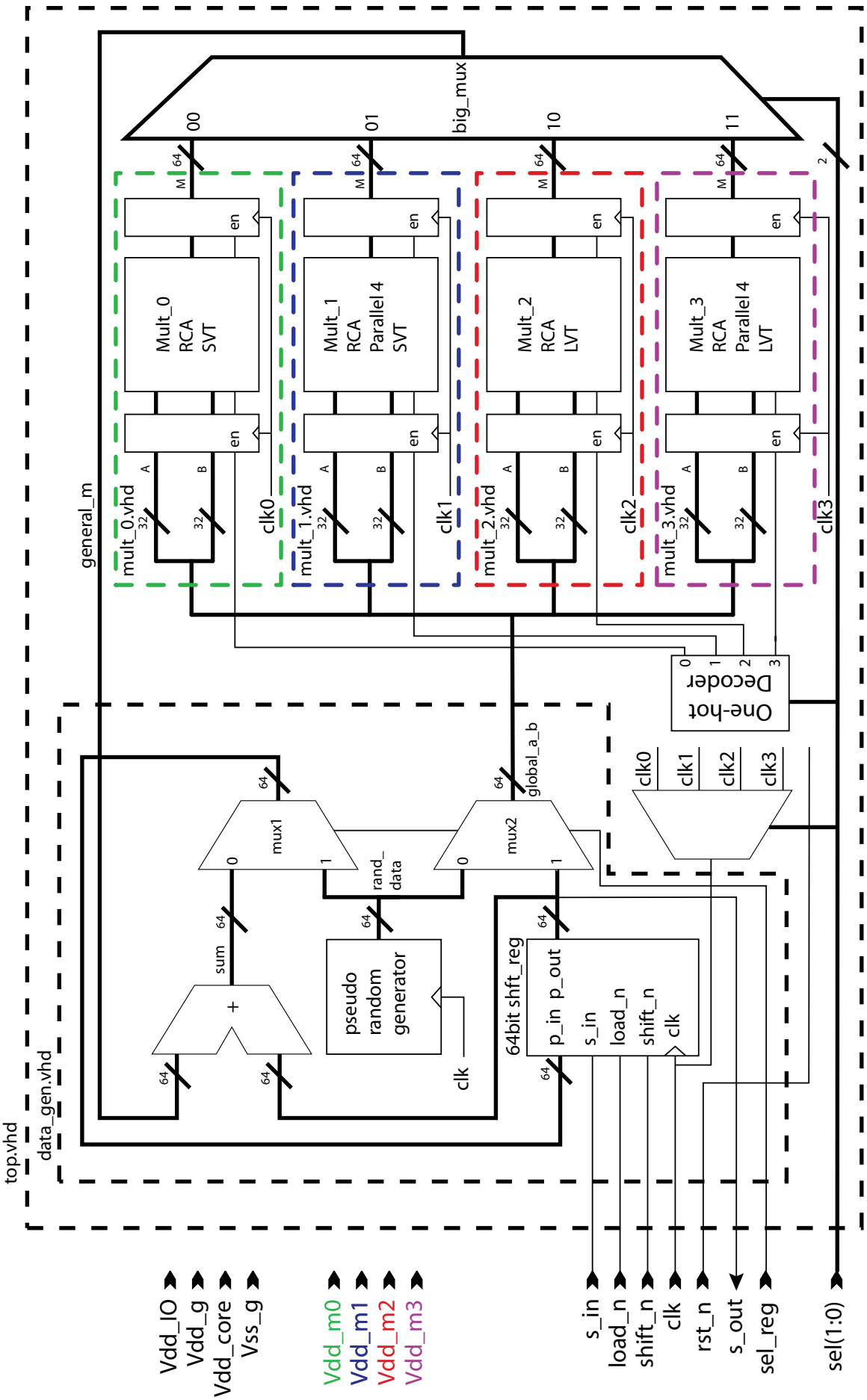

Figure 10.1: Block schematic of the test circuit

as a shift register with some bits logically "xnored" and seeded to the shift register input. The schematic of the data generator is depicted in Fig. 10.2.

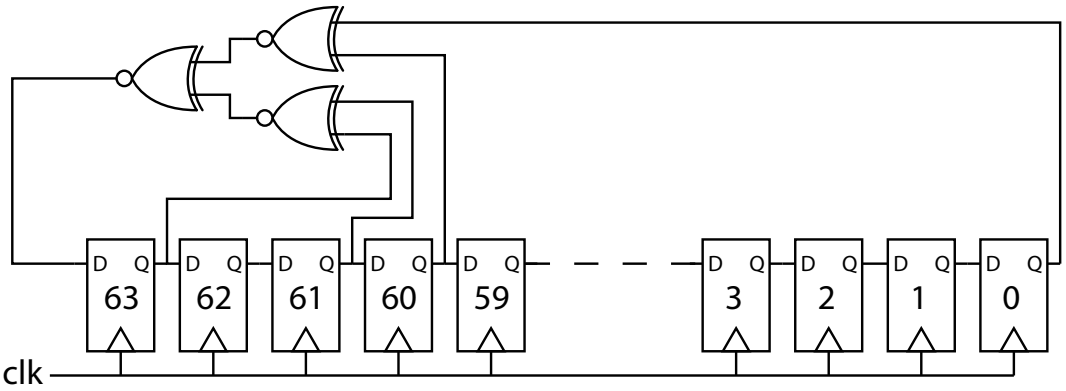

Figure 10.2: Schematic of the 64 bit linear feedback shift register

The data is 64 bit wide and provides the two 32 bit vectors used as the two inputs of the multiplier under test.

The particularity of a linear feedback shift register (lfsr) is that all possible codes are generated in a equally distributed way, without repetitions, until all codes have passed. The only code never generated , and also the one to be avoided, is the "all-ones" code, which is a stable code and always generates itself. Another advantage of this implementation is the fact that the generated sequence is always the same given the same starting code. In the case of our circuit, the shift register will be reset prior to every multiplication so that knowing the number of executed multiplications n permits us to pre-calculate the result of the cyclic adder expressed in Eq. (10.1) and in this way being able to verify that all the multiplications were executed correctly.

Fig. 10.3 shows the distribution of the generated numbers after 500 and after 10000 clock cycles. In the case of 500 generated numbers, it is possible to observe a slightly non uniform distribution due to the small number of generated data. If the amount of generated numbers increases, the distribution of probabilities becomes more uniform, as shown in Fig. 10.3. It is also interesting to note that, due to the shift nature of generated data, splitting the 64 bit code in two 32 bit vectors doesn't change the probability distribution, actually the new derivated vectors will present the same probability distribution as the original one. Moreover, the multiplication of two uniform distribution results in a distribution proportional to $\ln(1/x)$ as shown by the last two graphs of Fig. 10.3.

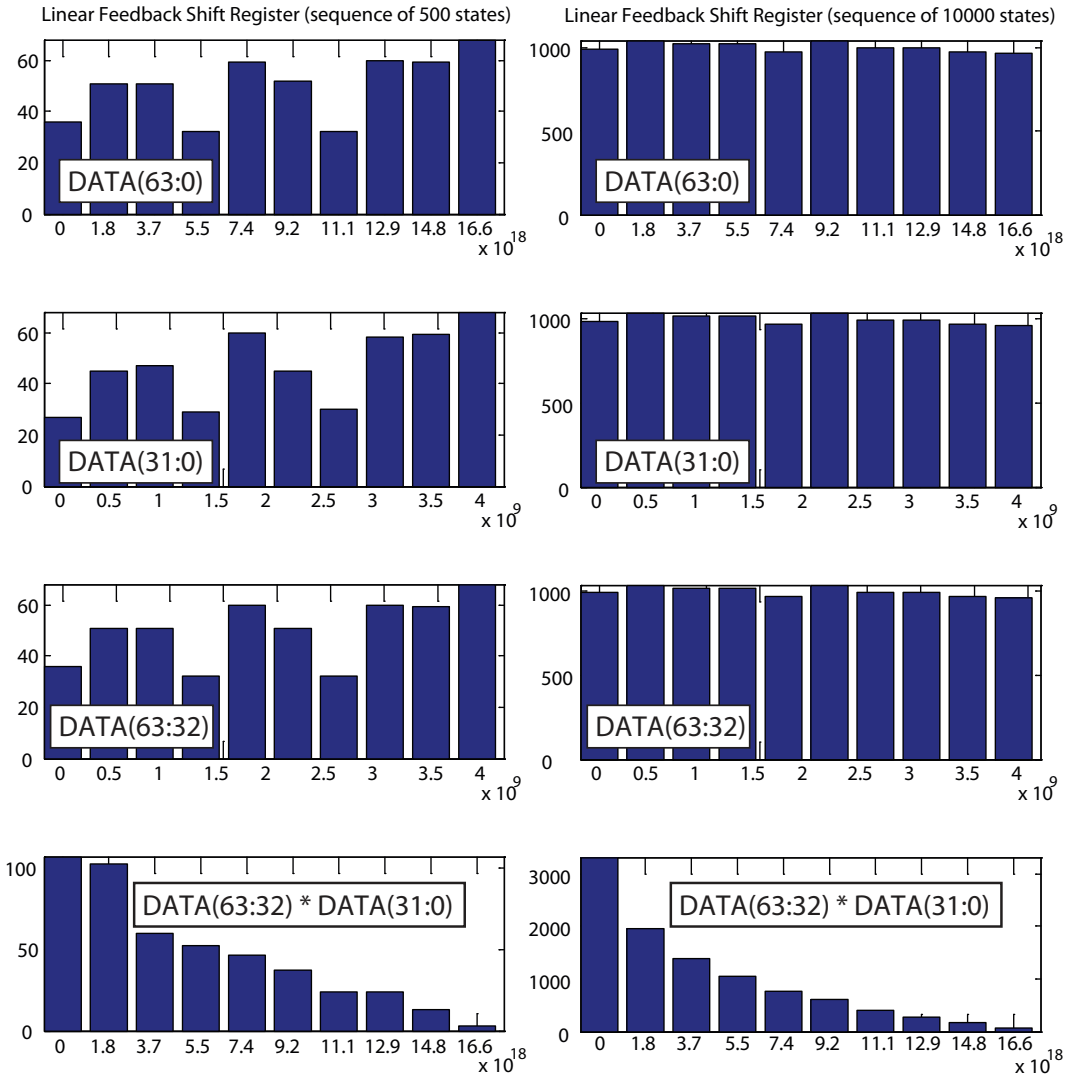

Figure 10.3: Probability distribution of the pseudo-random generated data for 500 and 10000 generated data

10.1.2 Ring oscillators

Besides the design described in the Fig. 10.1, two small ring oscillators have been added to the implemented circuit. One is implemented with inverters based on SVT transistors, whereas the other is implemented with inverters based on LVT transistors. Both ring oscillators were designed to have an oscillation frequency of 62.5MHz at nominal conditions, which corresponds to the expected working frequency of the multipliers under the same conditions. This means:

- **ring_lvt:** 533 inverters (IVLVTX1)
- **ring_svt:** 437 inverters (IVSVTX1)

10.2 Circuit design and implementation

The design has been written in the VHDL language and the source code can be found in Appendix A. The synthesis of this code has been done using Synopsys Design Compiler V2004.06-SP1 and the activity annotation for accurate power estimation has been obtained with ModelSIM from MentorGraphics version 5.6f. All the Synopsys scripts can be found in Appendix B.

The technology used for the synthesis is the 90nm from ST Microelectronics. This technology has been fully described in Chapter 4.

The results of the synthesis are stored in a verilog netlist ready to be used for the Place&Route (P&R) software. In our case, we used SoC Encounter version 4.10 from Cadence. The scripts used for P&R are reported in Appendix C.

Finally, the design passed the DRC (Design Rule Check) done using Calibre DRC from MentorGraphics. The final layout of the circuit is shown in Fig. 10.4.

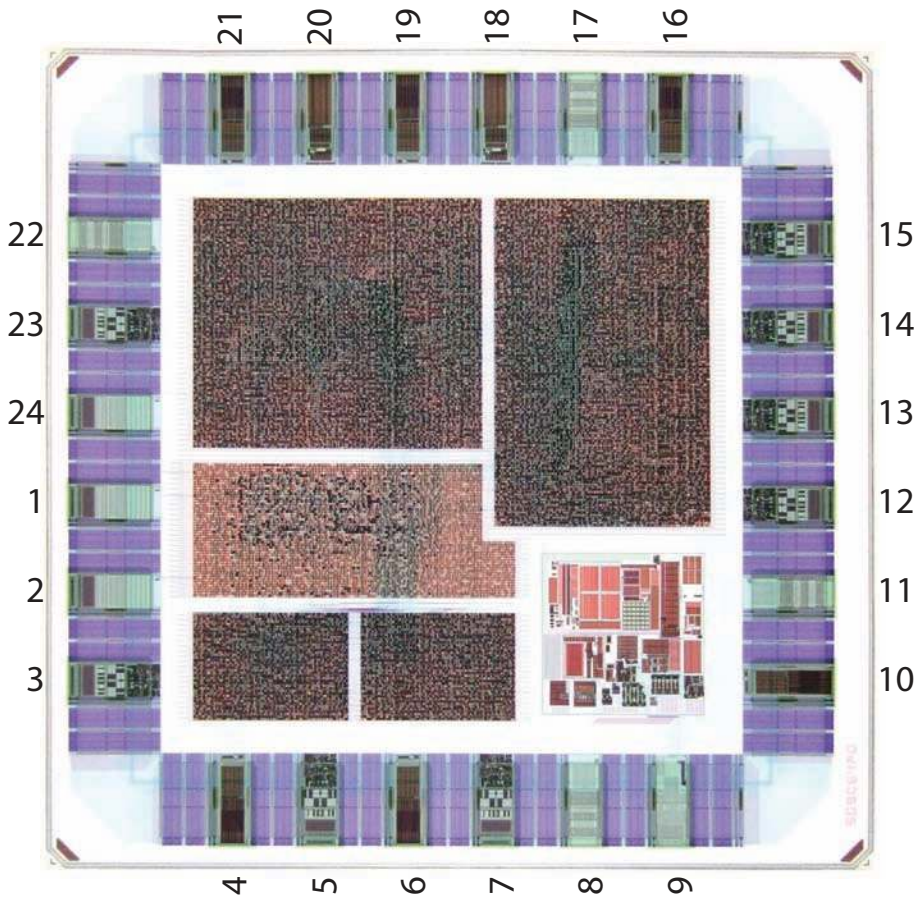

Figure 10.4: Final layout of the demonstrator circuit

In Fig. 10.4 we can recognize the two RCA parallel 4 multipliers in the upper part, the two RCA basic multipliers in the lower left part, whereas the control logic and the data generator are located in the middle left part. The square block located in

the bottom right angle is a compensation circuit required to stabilize the IO cells. A block view of the design in reported in Fig. 10.5.

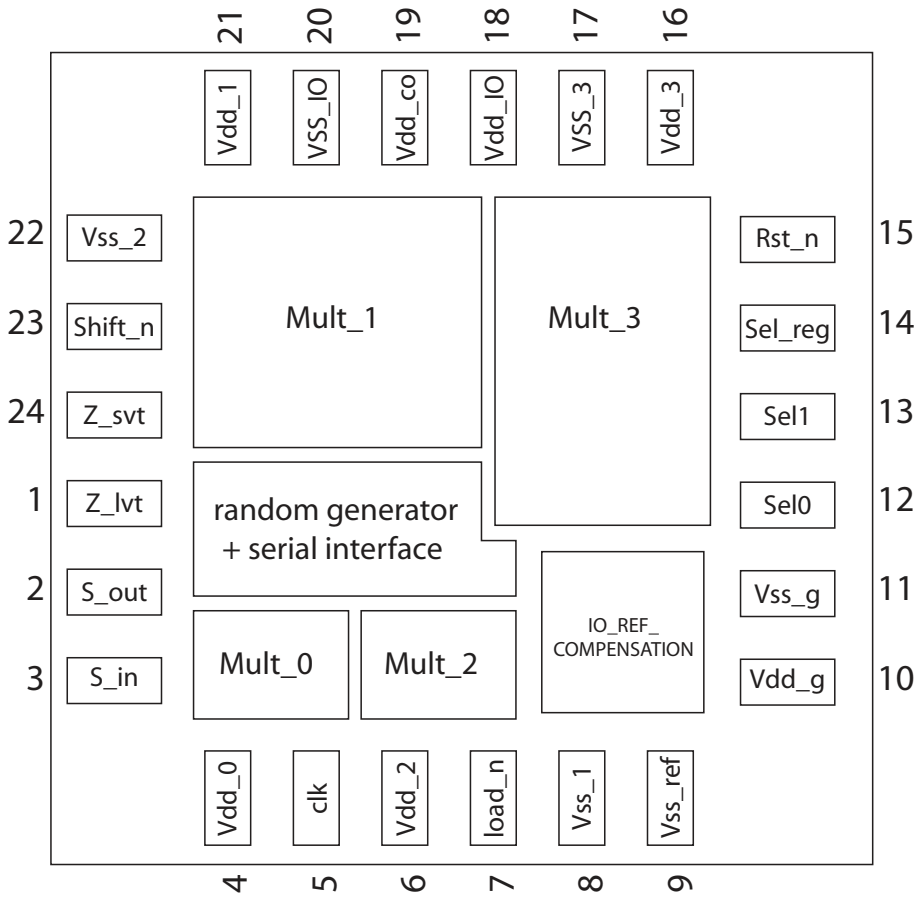

Figure 10.5: Block view of the demonstrator circuit

The pin names and their functions are:

1. **Z_lvt:** Output of the ring oscillator formed by 533 LVT type inverters;

2. **S_out:** Serial output of the shift registers. This output is used to read the content of the shift registers. From the read value the correct multiplier behavior can be verified;

3. **S_in:** Serial input of the shift registers. This pin can be used to enter a value to be multiplied or to verify the correct functioning of the shift registers;

4. **Vdd_0:** Supply voltage for the multiplier 0 (RCA basic with SVT transistors);

5. **Clk:** Clock of the system;

6. **Vdd_2:** Supply voltage for the multiplier 2 (RCA basic with LVT transistors);

7. **Load_n:** When low, data is loaded in parallel from the p_in input into the shift registers (see Fig. 10.1). This is the typical behavior during the sum and accumulation process;

8. **Vss_1:** System ground;

9. **Vss_ref:** System ground;

10. **Vdd_g:** Supply voltage for the IO_REF_COMPENSATION block (1.0V);

11. **Vss_g:** System ground;

12. **Sel0:** Bit zero of the sel signal. This signal select which multiplier is under test;

13. **Sel1:** Bit one of the sel signal. Sel coding is binary;

14. **Sel_reg:** Selector for routing data from the pseudo-random number generator and to/from the shift registers;

15. **Rst_n:** System asynchronous reset signal, active low;

16. **Vdd_3:** Supply voltage for the multiplier 3 (RCA parallel 4 with LVT transistors);

17. **Vss_3:** System ground;

18. **Vdd_IO:** I/O supply voltage (3.3V);

19. **Vdd_co:** Supply voltage for the pseudo-random generator and serial interface block;

20. **Vss_IO:** IO ground;

21. **Vdd_1:** Supply voltage for the multiplier 1 (RCA parallel 4 with SVT transistors);

22. **Vss_2:** System ground;

23. **shift_n:** When low (and load_n is high), data in the shift register shifts one bit on each clock rising edge;

24. **Z_svt:** Output of the ring oscillator formed by 437 SVT type inverters;

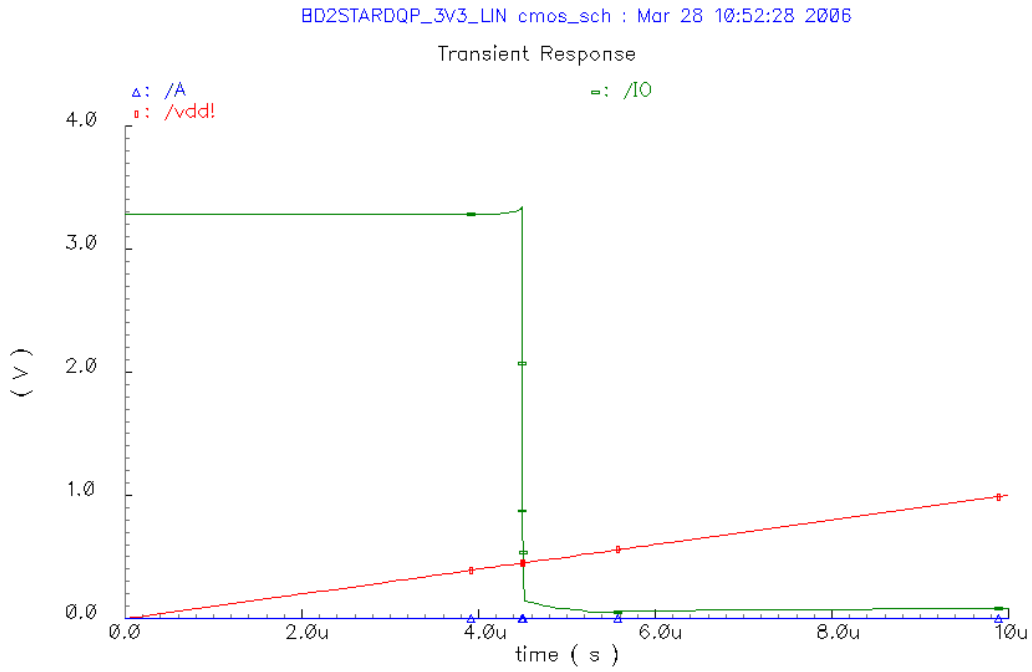

Figure 10.6: Output pad level converter for different core supply voltages. The linear ramp represents the core supply voltage, the line marked with triangles and constantly bound to zero is the logical level from the core and the line marked by wide rectangles is the corresponding IO output.

This circuit being destined to work at very low supply voltage (<0.5V), the level converter included in the standard output cells is not suited for granting a good level conversion under this condition as reported by Fig. 10.6.

Actually, in Fig. 10.6 we can observe that, for a core powered with a tension lower than about 0.45V, the output value jumps to 3.3V whereas 0V should be reported instead. For this reason the output ports (luckily only 3 ports of the design are outputs) have been assigned as analog pads and the level conversion has been left to an external circuit. This problem doesn't exist for the input ports, in fact, signals coming with a higher voltage than the core supply are never confused with the "0" logic level.

10.2.1 Nominal values

The nominal synthesis values, as well as the architectural parameters, for the four implemented multipliers are reported in Table 10.1.

The definitions of the parameters reported in Table 10.1 are:

- Cells: the number of design cells. Note however that cell can be a very simple

	Cells	Nets	Area	Activity	Delay	LD_eff	χ	χ^{α}	Nominal values				
			$[\mu m^2]$		[ns]				Vdd [V]	Vth0 [V]	Pdyn $[\mu W]$	Pstat $[\mu W]$	Ptot $[\mu W]$
Mult0	2633	3660	27250.1	0.686	13.28	396	0.6315	0.4684	1.0	0.353	3651	33	3684
Mult1	10122	14029	104333.4	0.194	13.65	102	0.2772	0.1204	1.0	0.353	4092	128	4220
Mult2	2568	3595	27105.2	0.696	10.73	392	0.5767	0.4237	1.0	0.342	4066	273	4339
Mult3	9732	13639	103502.6	0.197	11.16	102	0.2432	0.1102	1.0	0.342	4492	1030	5522

Table 10.1: Nominal values of the 4 implemented multipliers. Nominal frequency is 62.5MHz

one (like an inverter) or a complex one (like a full adder);

- Nets: the number of inter-cells nets in the design;
- Area: the area of the design core; pads and routing spaces are not included;
- Activity: the average number of switching nets over the total number of nets per clock period. These values are obtained by an event-driven simulation under ModelSIM (from MentorGraphics). The results are based on the multiplication of pseudo-random data over 500 multiplications; Standard library delays are used so that glitches can be accounted;
- Delay: the typical combinatorial delay from register output to register input on the critical path;
- LD_eff: the effective logical depth in equivalent NAND2 gates. The term "effective" is used to emphasize the fact that the length of the logical depth is considered against the throughput frequency or one-complete-multiplication frequency. In the case of a 4 times parallelization, for instance, LD_eff corresponds to a forth of the real LD because each block has four clock periods to compute one multiplication. The delay of the reference NAND2 gate has been estimated by putting 1000 NAND2 as in a chain of inverters. The inversion effect has been obtained by tying the two inputs together. The resulting delay per gate is 33.5ps for the SVT transistor type and 27.4ps for the LVT;
- χ and χ^{α}: these two parameters are obtained by using Eq. (6.3) from the nominal Vdd, Vth and delay;
- Nominal Vdd: the nominal technology supply voltage;
- Nominal Vth0: the nominal technology threshold voltage;
- Nominal Pdyn: the nominal dynamic power consumption as reported by Synopsys DC;
- Nominal Pstat: the nominal static power consumption as reported by Synopsys DC;
- Nominal Ptot: the nominal total power consumption obtained by summing the nominal Pdyn and the nominal Pstat.

10.3 Measurements setup

For measuring the power consumption of each multiplier at their limit of functionality (i.e. the lowest possible supply voltage guaranteeing correct results for a given frequency) the following things are required:

- **Generate the supply voltages:** The circuit requires many different supply voltages in order to work. The multiplier under test needs a separate supply voltage. Then, the core logic, containing the pseudo-random data generator and the cyclic adder, requires a supply voltage at the same potential in order to internally interface the multipliers without problems. Moreover, the IO controller IO_REF_COMPENSATION should always be maintained to 1.0V and finally the IO pads must be powered with 3.3V.

- **Generate the control signals:** The circuit requires a clock and a reset signal. Besides, other signals must be generated in order to select the multiplier under test and to read/write the shift registers for checking the correct functioning of the multiplier. All these signals are generated by an Altera FPGA based board.

- **Convert output pins to 3.3V logic level:** As reported previously, the circuit outputs (namely Z_lvt, Z_svt and S_out) are implemented as analog signals and they hence need to be converted to a 3.3V logical level in order to be interfaced by the FPGA. This is obtained by putting discrete comparator devices on the output signals.

- **Measure the consumed multiplier current independently:** Finally, once the circuit can run, we must be able to measure the consumed current of the specific multiplier under test. This is accomplished by multiplexing the multiplier power supply to the correct multiplier power pins through reed relays. The advantage of using reed relays is that, the contact being mechanical, virtually no extra consumption is added to the measure, which would not be the case if a CMOS multiplexer circuit would be used instead.

10.3.1 PCB design

Fig. 10.7 shows the schematic of the PCB (Printed Circuit Board, designed with Altium Designer 2004 SP3, formerly Protel) used to interface the demonstrator circuit. The three connectors J5, J7, J9 are the "bridges" between the PCB and the FPGA board. On the right of the schematic, we can see the 4 reed relays (K1-K4) used to

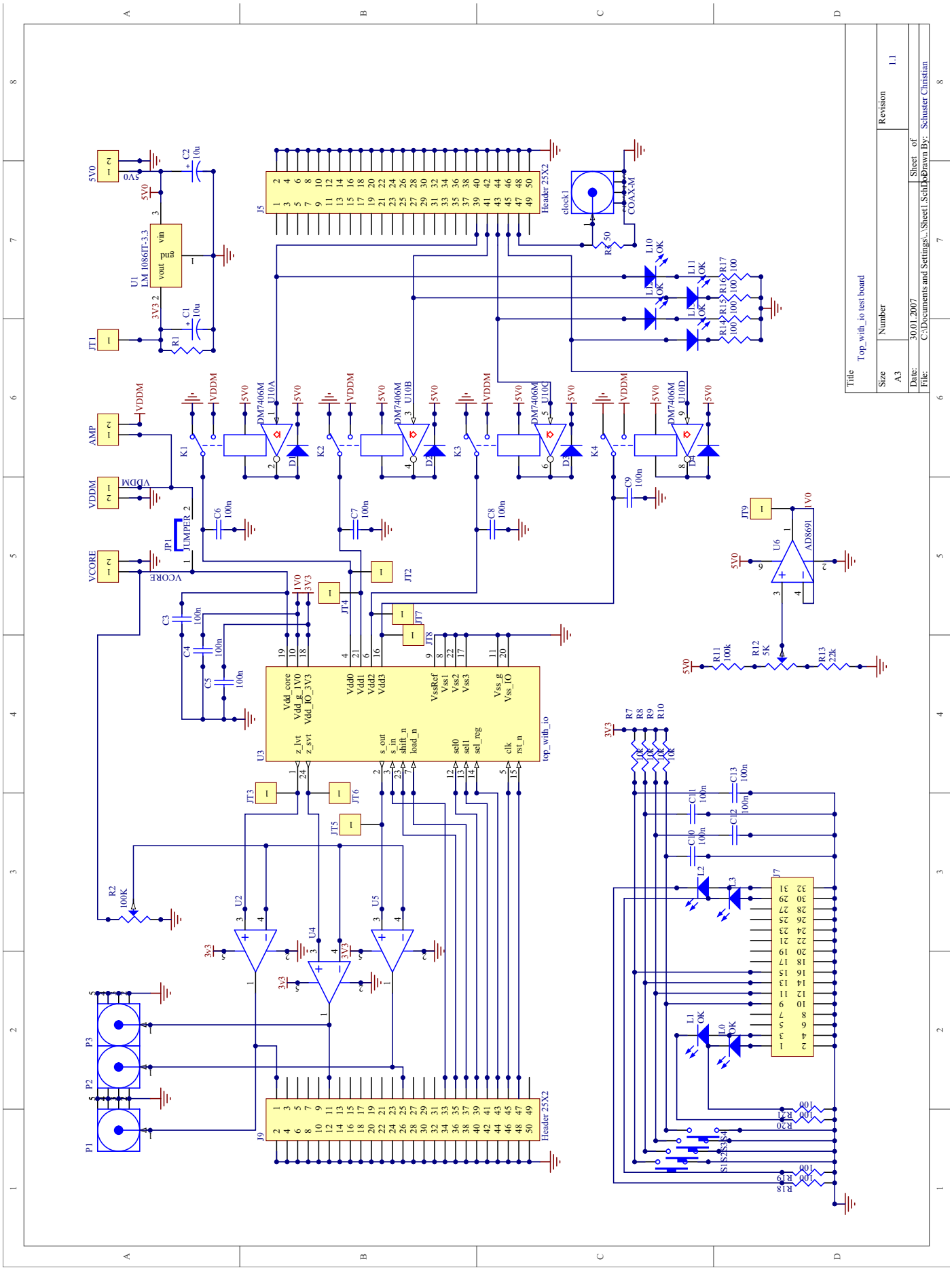

Figure 10.7: Schematic of the PCB used to test the demonstrator circuit

supply the multiplier under test and the corresponding LEDs which provide a visual feedback on which multiplier is currently selected. On the left bottom corner there is a small user interface with 4 buttons and 4 LEDs. Two of these LEDs (OK, KO) are used to show if the content of the shift register was the expected one, i.e. if the multiplier worked correctly or not. The other buttons/LEDs are there to expand the functionalities if needed. The comparators (U2, U4, U5), required to convert the output level of the three output pins Z_lvt, Z_svt and S_out to 3.3V, are visible on the left part of the schematic with extra connectors (P1-P3, on top) designed for debugging purpose. The reference voltage defining the separation between the logical level 0 and the logical level 1 has been obtained with a potentiometer from the VCORE pin. In this way, the reference voltage will always be proportional to the supply voltage used for the core. All the chip input signals are connected directly to the FPGA through J9. The 3.3V is generated from the 5V on the card with a voltage regulator shown in the top right edge. A stabilized 1.0V source was difficult to obtain from the 5V as no voltage regulator was found that can provide tensions so low. For this reason, this supply voltage has be generated using an operational amplifier used as a voltage follower. In this configuration, the tension set at the input through a resistor divider is replicated at the output (almost) independently from the drawn current. This block is shown in the bottom-centered part of Fig. 10.7. Finally, the multipliers power source is obtained externally by the connector VDDM and the current drawn is measured by applying a ammeter to the AMP connector. The tension for the core (which is all the design but the multipliers) can be obtained from VDDM with the jumper JP1 set or supplied separately by the VCORE connector.

10.3.2 FPGA based signal generation

The FPGA development card used in this work was a Nova Constellation 20 KE card [65], which is based on a Altera APEX EP20K600EFC672 FPGA. This card has 150 user programmable IOs working at 3.3V. It can be programmed through USB and JTAG interfaces. A serial programmer is also present, which permits automated FPGA reconfiguration on power-ups. Moreover, this card supports the SignalTrap II technology from Altera, allowing registers read back through JTAG during runtime. This feature is very practical for debugging. The card is powered by 5.0V and an internal 40MHz clock frequency is present. In our case, an external oscillator will be used in order to be able to measure the power consumption for different frequencies.

The FPGA code has been written in VHDL and compiled with Altera Quartus II v6.0 SP1. The source code is reported in Appendix D.

The FPGA pin assignments are reported in Table 10.2.

Name	**PIN**	**Name**	**PIN**
OK_led	PIN_E13	CHIP_rst_n	PIN_N19
KO_led	PIN_H15	CHIP_sel[0]	PIN_T22
Power_mult0	PIN_F12	CHIP_sel[1]	PIN_M17
Power_mult1	PIN_H13	CHIP_sel_reg	PIN_L20
Power_mult2	PIN_J16	CHIP_shift_n	PIN_T23
Power_mult3	PIN_K15	CHIP_sin	PIN_R23
Switch1	PIN_E16	CHIP_sout	PIN_M21
Switch2	PIN_G16	ext_clock	PIN_G15
Switch3	PIN_H16	mult_num[0]	PIN_E14
Switch4	PIN_E15	mult_num[1]	PIN_F15
CHIP_clock	PIN_N22	LED2	PIN_G18
CHIP_load_n	PIN_M18	LED3	PIN_F18
Z_svt	PIN_U21	Z_lvt	PIN_U22

Table 10.2: Pin assignments for the APEX EP20K600EFC672 FPGA

The FPGA code does:

- Select the desired multiplier;
- Reset internal registers;
- Execute 10'000'000 multiplications and accumulate the results on the 64 bit register;
- Read back the content of the accumulator register;
- Verify the read data with the expected value and output the decision on the pass/fail pins;
- At the end of this sequence, the chip clock is stopped to allow static power measurements.

A particularity of this code is the use of two clock frequencies for the circuit under test, depending on the executed task. In fact, while the chip clock runs at full speed (the same of the FPGA) during the execution of the 10'000'000 multiplications, a clock divided by 4 is used during the data read-back phase. This was required in order to execute tests with frequencies bigger than 35MHz (like the nominal circuit frequency

of 62.5MHz). The limiting factor was the propagation delay of the comparator used to convert the low voltage level of the s_out pin to the 3.3V level of the FPGA. In fact, if the frequency was too high, the read value was latched before it was ready.

10.3.3 MATLAB based measurements automation

To test the manufactured circuits, lots of current measurements were required at difference frequencies, supply voltages and this for every multiplier. Moreover, the measurement of the power consumption during runtime needed to be synchronized with the design under test. For these reasons, an automated way to set the parameters (frequency, supply voltage) and to check the results was required.

To perform an automated measurement the following devices have been used:

- **Agilent 33250A:** Frequency generator, this device can generate a square wave frequency up to 80MHz;
- **Keithley 213:** Power supply and control signal generator, this device is a Quad Voltage Source (QVS) and includes 8 digital inputs and 8 digital outputs.
- **Keithley Sourcemeter 2400:** Power supply and ammeter with a precision up to 10 pA.

All this devices support the GPIB (General Purpose Interface Bus) protocol. This protocol is a standard for controlling devices remotely. The described tools were connected with a cable to a computer provided with a National Instrument acquisition card and controlled by MATLAB. In order to be able to use the GPIB protocol, the Instrument Control Toolbox for MATLAB was required. The MATLAB source code used for the measurements is reported Appendix E.

To determine if a multiplier was able to work at the given frequency and supply voltage the test was performed 10 times in a row with the same frequency and supply voltage. If at least one of these 10 tries was successful, the multiplier was considered capable to work at this condition (even if not all the times).

The frequency range for most of the tests span from 1 to 20MHz, whereas the supply voltage accuracy chosen was of 10mV.

Finally, the core (i.e. all the design but the multipliers) was supplied with 100mV more than the multiplier under test, and this to avoid as much as possible to be limited by the working supply voltage of the data generator block.

10.4 Measurements

Two chips (No.2 and No.3) have been chosen (without any particular reason) for a complete power consumption analysis and discussion. First, the power measurements at nominal conditions (Vdd=1V and f=62.5MHz) and their comparison with values reported by Synopsys DC will be considered. Later, the detailed power measurements for each multiplier of both chips will be carried out for frequencies ranging from 1 to 20MHz. Finally, a discussion on the power and delay variability with data measured over 16 dies manufactured on the same wafer will be presented.

10.4.1 Nominal values

The nominal power consumptions and the critical path delay for chip No.2 and No.3 are reported in Table 10.3.

	Chip No.2				Chip No.3			
	Mult_0	Mult_1	Mult_2	Mult_3	Mult_0	Mult_1	Mult_2	Mult_3
Pstat [μW]	132	501	1152	4515	169	631	1571	5931
Pdyn [μW]	3080	3312	2957	3395	3103	3350	2978	3385
Ptot [μW]	3212	3813	4109	7910	3272	3981	4549	9316
fmax [MHz]	74.5	-	-	-	75.4	-	-	-
Delay [ns]	13.42	-	-	-	13.26	-	-	-

Table 10.3: Measured nominal (1V@62.5MHz) power consumption and maximal working frequency

These values can be compared with the ones provided by Synopsys and reported in Table 10.1. The most remarkable difference comes from the static power consumption. Indeed, real measurements of static power report values around 4-5 times bigger than the expected ones. This clearly point out a big problem related to the nanometer CMOS technologies: the parameters variability. As explained more in details on the coming subsections, this extreme increase of the static power should mainly be due to a threshold voltage much lower than the expected one, probably coming from a not so well mastered effective transistor dimensions and doping profiles. Nevertheless, the ratios of the static power between the 4 multipliers in the same chip remain almost correct, like the parallel 4 version which shows 4 times the static power of the basic version.

Regarding the dynamic power, the measurements are less astonishing, but still the results show a dynamic consumption lower than the expected one. The reasons

could be lower capacitances, due to variable transistor effective dimension, and/or an activity slightly different (as a reminder, activity of all nodes, including internal cell nodes, was estimated based on the activity on the nets connecting cells).

The delay of the critical path was measured by increasing the frequency at the nominal supply voltage of 1V until the multiplier stop working. The measurement was only possible for the Multiplier 0 (RCA basic SVT) because the frequency generator available at our laboratory only reach the 80MHz, and this was not enough for measuring the other three multipliers. The measured delay is very near to the expected one of 13.28ns.

10.4.2 Lowest working supply voltage

The expected lowest working supply voltage for a given frequency is reported in Fig. 10.8 and is based on Eq. (6.15).

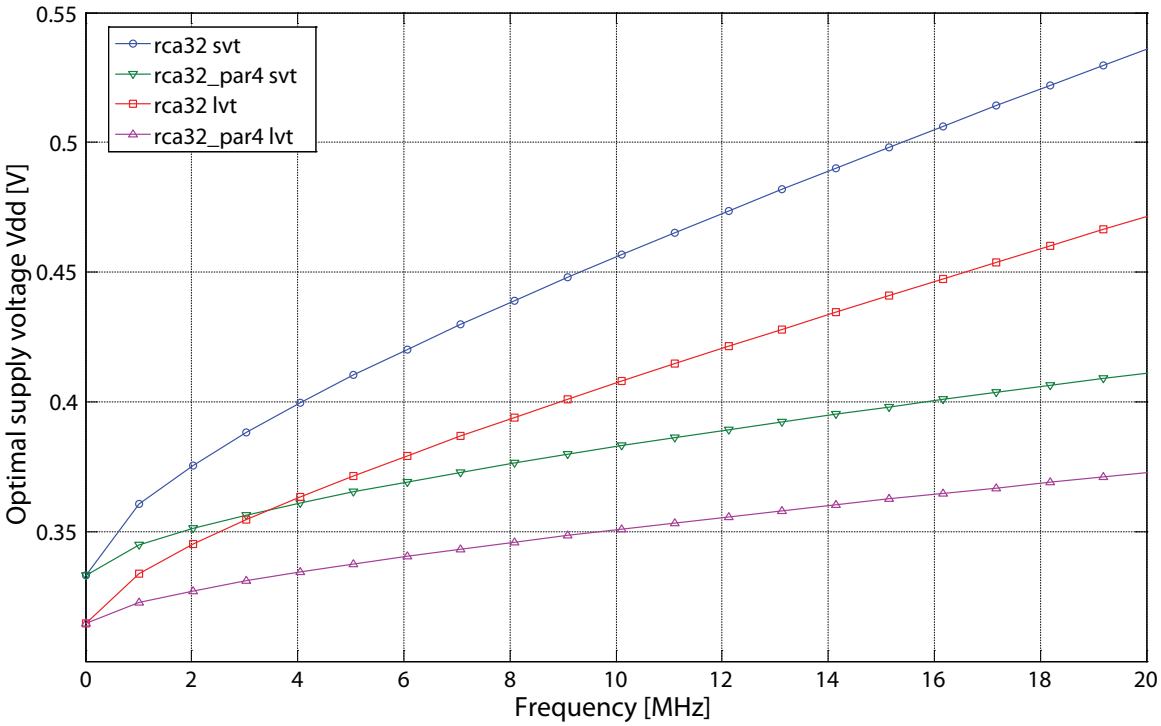

Figure 10.8: Expected optimal supply voltage

As we can observe, the supply voltages are reduced until they reach the corresponding threshold voltage at 0MHz. The non parallel versions have a slightly steeper slope compared to the parallelized versions. This is due to the larger LD_eff for the basic version, making it "harder" to reduce the supply voltage unless it reaches very low frequencies. Mathematically, the larger LD_eff is observed as a bigger χ.

A similar plot has been obtained, by measurement, for chip No.2 and No.3. Results are reported in Fig. 10.9 and Fig. 10.10 respectively.

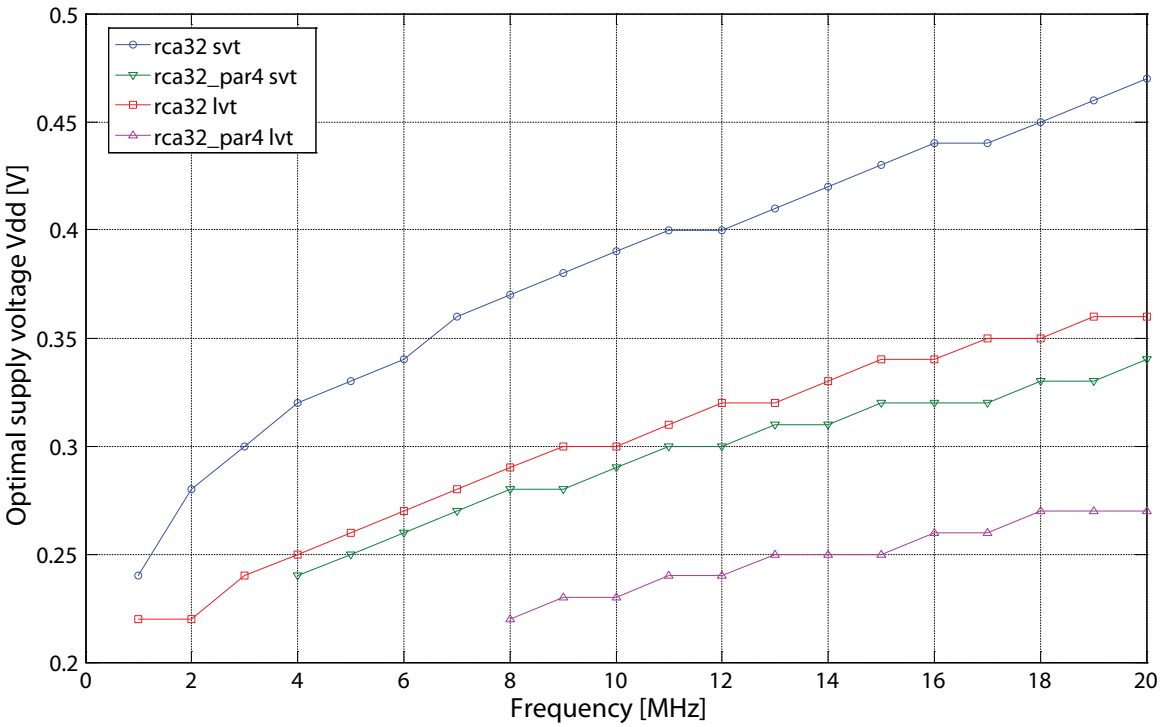

Figure 10.9: Measured optimal supply voltage for chip No.2

At a first look, Fig. 10.9 and Fig. 10.10 show the same shape of Fig. 10.8, but in reality they present lower values compared to the theoretical case. In particular, it is interesting to note the converging values for very low frequencies (the missing values are due to non working conditions resulting from a too low supply voltage). As seen before, this converging values correspond to the threshold voltage of the technology. From these plots, it is possible to imagine that the threshold voltages for the measured circuits should be around 0.2V or even lower. This is quite different from the one around 0.33V reported in Fig. 10.8 (Remember that $Vth = Vth0 - \eta Vdd$). With a lower Vth is now understandable why optimal Vdd are lower than the theoretical ones, while the shape of the plot is maintained.

This much lower threshold voltage can now also easily explain the large factor of 4-5 between the measured static power and the expected one at the nominal conditions. In fact, the static power depends exponentially on the threshold voltage, as reported in Eq. (3.5).

It is also worth to note that all multipliers of chip No.2 and No.3 worked at 250mV and that two multipliers (mult_1 and mult_2) of chip No.3 worked at a supply voltage as slow as 210mV with a frequency of 1MHz!

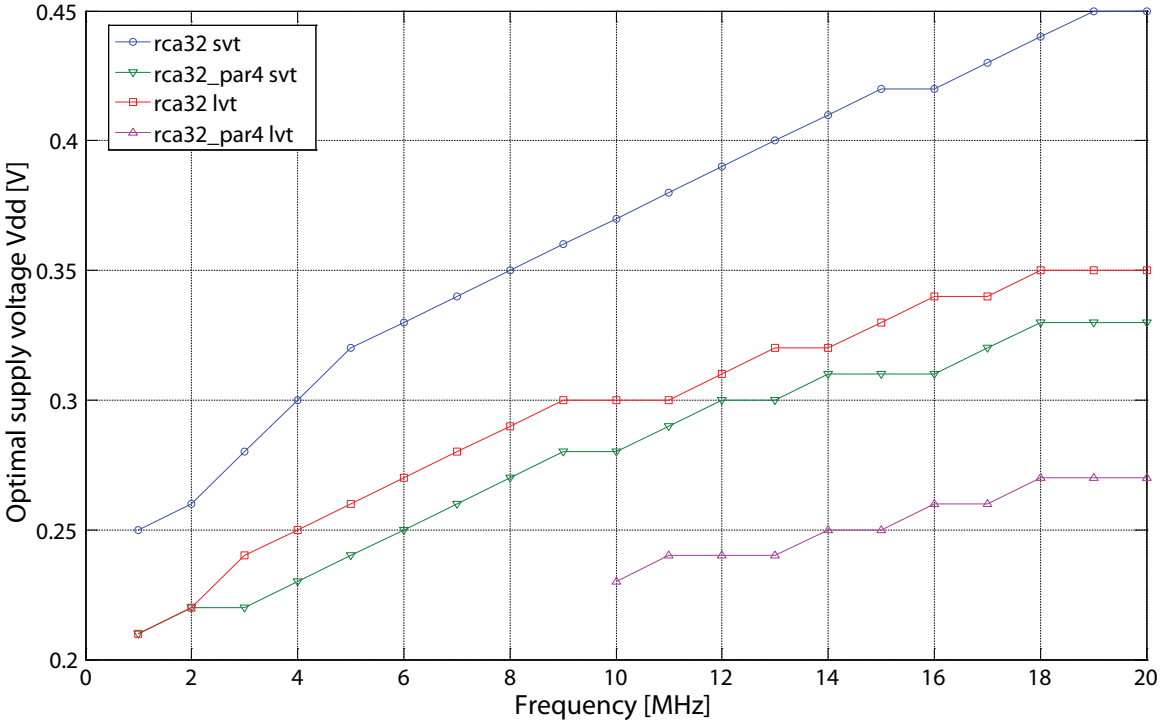

Figure 10.10: Measured optimal supply voltage for chip No.3

10.4.3 Optimal total power

The total power consumption can now be calculated for the lowest working supply voltage (optimal Vdd) thanks to Eq. (3.6). Fig. 10.11 illustrates it for the theoretical case.

The measured optimal total power for the chip No.2 and No.3 are reported in Fig. 10.12 and Fig. 10.13 respectively. The missing points correspond to values of the optimal Vdd too low to permit correct measurements.

As for the optimal supply voltage, we can observe that the shape of the plots measured is very similar to the theoretical one, but the corresponding optimal power is lower for the real circuits. This can, once more, be explained by the lower real threshold voltage, which permits a lower optimal supply voltage and hence a lower optimal total power.

The measured optimal supply voltages for mult_3 (RCA parallel 4 LVT) were very low and for this reason, reported optimal total power should be taken with care.

In both chips, the measurements for the multipliers corresponding to the SVT transistor type are very similar, whereas chip No.3 shows a slightly higher consumption for the LVT type compared to No.2. This can be explained by the higher power static consumption of chip No.3 as reported in Table 10.3, which manifests it mainly on

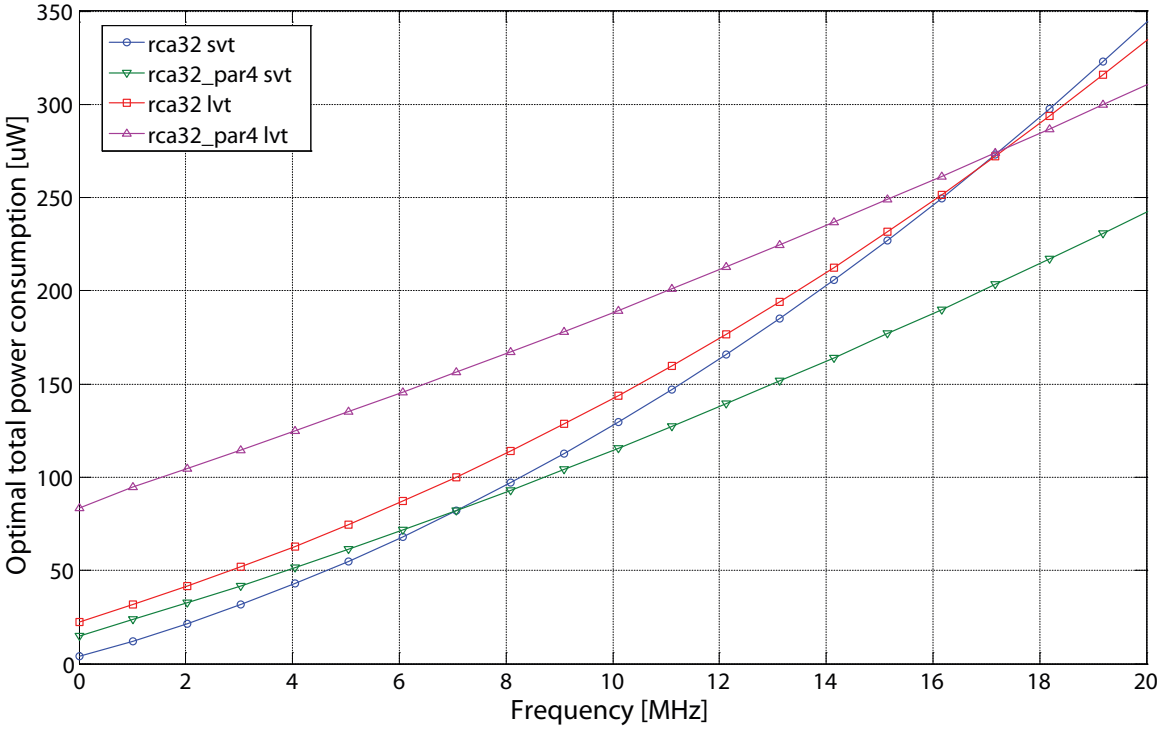

Figure 10.11: Expected optimal total power consumption

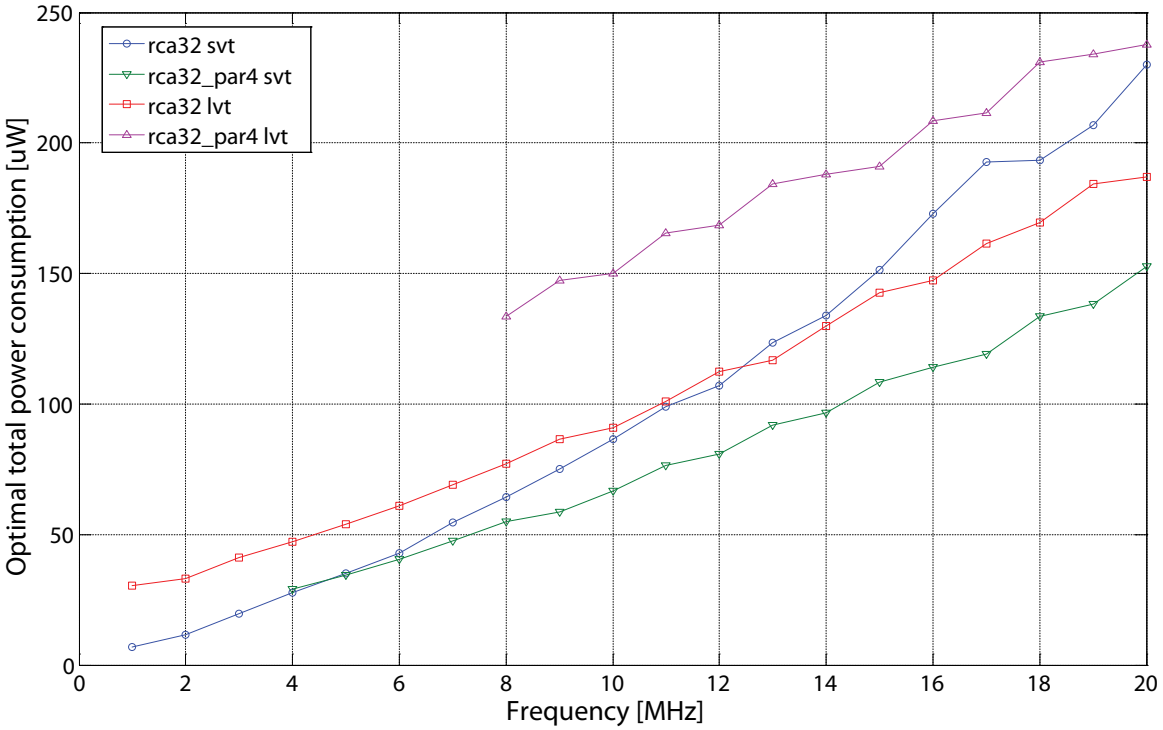

Figure 10.12: Measured optimal total power consumption for chip No.2

LVT multipliers where static power is predominant.

The large variations in the technology parameters, discussed further in the next

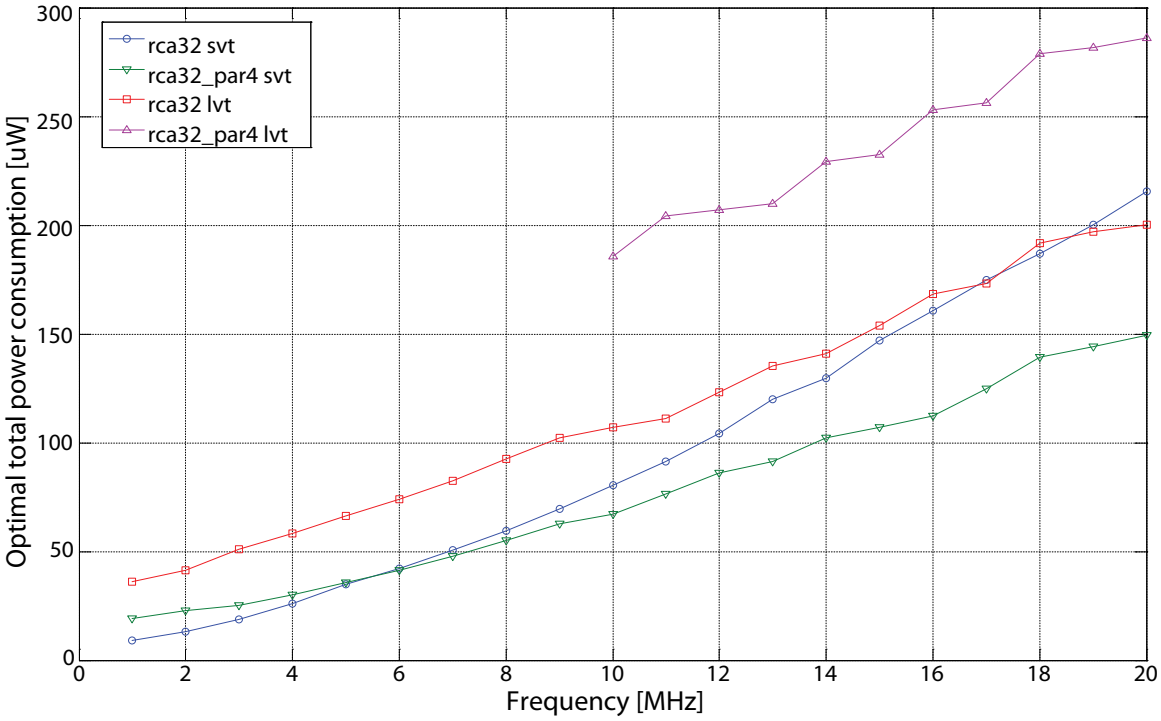

Figure 10.13: Measured optimal total power consumption for chip No.3

section, makes it very difficult to accurately predict the optimal total power over a so large range of frequencies. Nevertheless, the main shapes of the plots are maintained. In particular, let consider the cross points between the RCA basic SVT curves and both RCA parallel 4 SVT and RCA basic LVT. In the theoretical plot these crosses occur at 7MHz and 17MHz respectively.

If we look to the same crosses on the measured data, we observe them at 5MHz and 13MHz for chip No.2 and at 6MHz and 17MHz for chip No.3. These results are very similar to the expected ones, considering the high technology parameters variations observed.

Practically, we can say that if a design is destined to work at 2MHz, the RCA basic SVT is the best choice for low power, if it is designed for 10MHz RCA parallel 4 shows a better power profile and at 20MHz RCA basic SVT will consume more than the RCA basic LVT which will consume more than the RCA parallel 4 SVT.

10.4.4 Power and delay variability

In the preceding discussions, it was pointed out many times that technology parameters are quite variable from die to die even when they come from the same wafer, as it is the case for all the chips investigated in this thesis.

To explore a little deeper this aspect, the static power, dynamic power and critical

path delay (obtained from the maximal working frequency) of the multiplier 0 (RCA basic SVT) at nominal conditions (1V/62.5MHz) have been measured for 16 different dies.

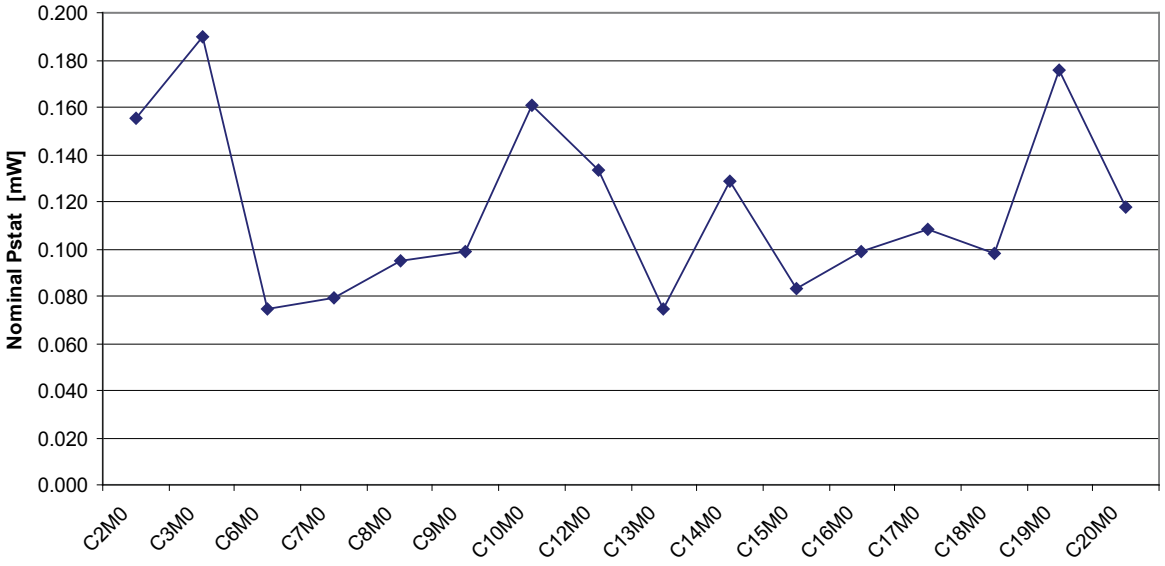

Figure 10.14: Nominal static power distribution for 16 chips

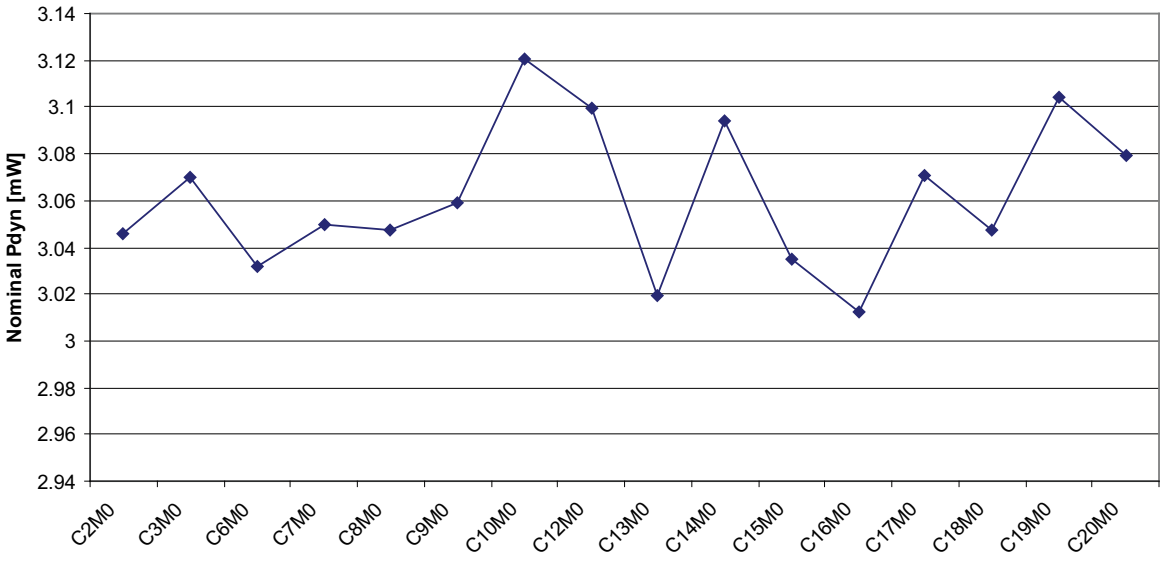

Figure 10.15: Nominal dynamic power distribution for 16 chips at 62.5MHz

The data corresponding to the nominal static power is reported in Fig. 10.14. Here, we can see that the static power spans from a minimum of 75 μW to a maximum of 190 μW, which correspond to a factor larger than 2.5! Moreover, the average value of 117 μW is more than 3.5 times larger the value estimated by Synopsys! This variability makes very problematic the power estimation for circuits dominated by static power.

The nominal dynamic power consumption presents a much lower variability between dies, as illustrated in Fig. 10.15. In fact, all measured values are included in a range from 3012 μW to 3121 μW, which correspond to a variation of $\pm 2\%$ around the average value of 3062 μW. This is "only" 17% lower compared to the value provided by Synopsys. Moreover, by comparing the static power distribution with the dynamic one, we can observe a small correlation between the two. Actually, most of the time a die with a higher static power consumption, also shows a relative high dynamic power. A possible answer to this can come from the shortcut current (explained in Chapter 2.1.2). In fact, a higher sub-threshold current (lower Vth or higher I_0 or both) also means a higher "on" current, which increases the shortcut dissipation. This could also explain why the variations of the dynamic power only account for a few percents.

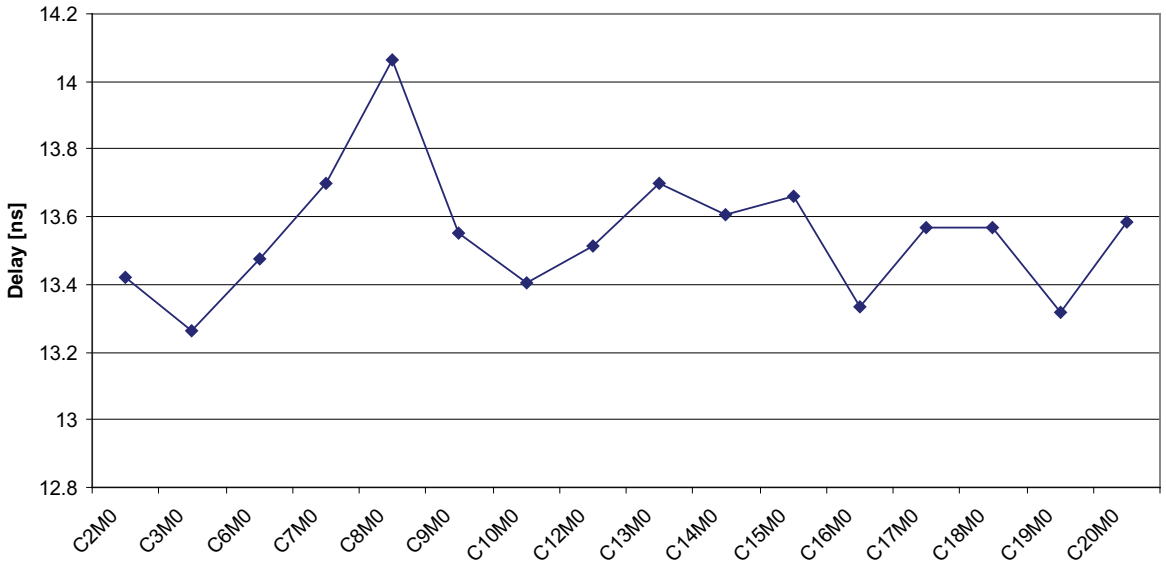

Figure 10.16: Delay distribution of the RCA SVT multiplier for 16 chips

Fig. 10.16 reports the measured critical path variability over 16 different dies. As for the dynamic power, the variation is quite limited and corresponds to $\pm 3\%$ around the average value of 13.55 ns. Moreover, this delay is only 2 % larger than the value reported by Synopsys. It is also worth noting that no correlation was observed between the power consumption and delay distribution.

10.5 Summary

This chapter discussed the demonstrator circuit used to investigate the influence of technology and architectural modifications to the optimal total power. The technology used was the 90nm from ST Microelectronics, which permitted us to implement

two different transistor types on the same chip. Moreover, two different 32 bit multipliers were implemented for each transistor type, yielding a total of 4 multipliers. The first part of the chapter was dedicated to the circuit design and conception, then the description of the measurements setup follows and, at the end, the measured data were exposed and commented. In particular, we observed an average static power 3.5 times higher than the typical values estimated by Synopsys, whereas the dynamic power was only 17% lower on average. The large difference between the simulation and real measurements can be explained with the threshold voltage, which, in reality, appeared to be much lower than the theoretical one. Besides these important differences observable at nominal conditions (Vdd=1V and f=62.5MHz), the total power for multipliers working at the lowest possible supply voltages was discussed. The measured values showed a shape very similar to the expected one, but with different absolute values. This can also be explained by the lower real Vth. At the end of the chapter, the variability of powers and delay were reported for the same multiplier in 16 different chips. The results showed a static power varying as much as a factor 2.5 between the lowest and highest value for multipliers coming from the same wafer! Without doubt, this large variability of static power will be a main issue in nanometer CMOS technologies, especially for designs where static power is a large contributor.

Chapter 11

Conclusions

With the introduction of nanometer CMOS technologies, new sources of power dissipation appeared. The continue shrinking of the transistor sizes, dictated by Moore's law, reached a point where new physical phenomena need to be faced. One of the most important problems related to these new phenomena is the huge increase of the static power consumption, which can become even bigger than the dynamic power for a running circuits. The static power consumption is the portion of the power dissipation that is constantly flowing from Vdd to Vss, even when the circuit is in idle state. For nowadays technologies, the principal contributor to static power comes from the sub-threshold current flowing through the transistors in off state. This type of current arises from the diffusion of the minority carriers in the transistor channel. The reason why this current is increasing so much in recent nanometer technologies is that it has an exponential dependency on the transistor threshold voltage, which is constantly reduced with new technologies to maintain the speed acceptable.

The goal of this thesis was to investigate the low power methodologies in technologies dominated by a large static power consumption. In particular, we were interested in the architectural as well as in the technology influence on the total power consumption.

The principal theoretical framework exposed in this thesis considers a scenario in which both the supply voltage and the threshold voltage can be freely modified. Under such assumption, the total power consumption clearly shows a minimum located at very low supply voltages (examples showed optimal Vdd lower than 0.4V even at frequency as 62.5MHz). The derivation of the ratio k1 (i.e. optimal dynamic power over optimal static power) showed that, this ratio being quite constant compared to the variation of I_{on}/I_{off} between technology nodes, nanometer technologies will require a growing ratio a/LD (activity over logical depth) to reach this optimum.

This, for instance, will make pipelining preferable over parallelization. After that, we have seen the influence of a, LD and f to the optimal Vdd and Vth, showing that frequency mainly influences the optimal Vth, logical depth mainly influences the optimal Vdd, while activity influences both of them. By comparing architectures under the rough approximation of a quasi-constant $k1$, we realized that pipelining and parallelization are more effective for low power when they show high logical depth and high frequency. We also observed that new technologies, characterized by a lower χ factor compared to older ones, will tend to penalize pipelining and parallelization, whereas the condition for a power saving by pipelining remains easier to fulfill compared to the parallelization one.

Going behind the quasi-constant k1 approach, analytical closed-form equations has been derived for the calculation of the optimal Vdd, optimal Vth and optimal total power directly from the architectural and technology parameters. Thanks to these equations, we observed that the optimal Vth is quite unchanged by pipelining, while the parallelization increases it by a precise amount, which only depends on the degree of parallelization. Moreover, sequential multipliers were clearly shown to be inadequate for low power at the optimal working condition due the large effective logical depth and the high number of transitions ($a \cdot N$).

From a low power point of view, the best characteristics for an ideal technology would be a capacitance C, delay constant k_t and sub-threshold slope n as low as possible, whereas the reference current I_0 and alpha power law coefficient α should be as high as possible.

After the technology influence discussion, a few possibilities for modifying the threshold voltage (like body bias, transistor resizing, technology choice) were also presented.

Under all the investigated architectural and technology modifications, the simple approximated analytical equations developed in Chapter 6 for the optimal Vdd, Vth and $Ptot$ showed very good results, reporting errors always lower than a few percent compared to numerical computation based on non-approximated equations.

In a second framework, the opposite case was considered, in which the threshold voltage as well as the supply voltage were assumed constant. This particular case was explored because it corresponds to the most typical case for industrial designers. In fact, they often have a fixed supply voltage and threshold voltage imposed by the technology and/or the devices the circuit has to interface. Under this condition, graphical tools for total power comparison of different architectures were presented. Examples of application of these tools to the same multipliers used in the precedent

framework were reported. In particular, we showed that, depending on the constraints used, the multiplier presenting the lowest total power is not always the same.

At the end of the thesis, a physical implementation of four different 32 bit multipliers was presented. These 4 multipliers represent all the possible combinations between two transistor types (SVT and LVT) and two architectures (RCA basic and RCA parallel 4). After an in-deep description of the circuit design flow and measurement setup, the nominal power consumptions as well as the optimal ones (those corresponding to the lowest working supply voltages) were compared to the theoretical values. The measured data showed, in average, a static power 3.5 times larger than expected. This was supposed to be due to real threshold voltages much lower than the simulated ones. Nevertheless, the shapes of the plots remained very similar to the expected ones. This means that, even if the absolute values were not well estimated by the models (due to the large technology parameters variability), the relation between them was respected. This was essential to be able to predict which multiplier presented the lowest total power for a given working frequency. It is also interesting to note that a few multipliers were able to work at 210mV of supply voltage at a frequency of 1MHz. Finally, the variability of powers and delay for 16 chips coming from the same wafer were reported. In particular, the variations on the static power at nominal condition (Vdd=1V, f=62.5MHz) were strongly fluctuating, accounting for a factor of more than 2.5 between the highest and the lowest measured values. On the other hand, the variations on the dynamic power and delays were within $\pm 3\%$.

From these observations, we can conclude that the major problem that the technologues will have to face in the future will be the difficulty to master the variations of the technology parameters. The price to pay for not achieving it would be lots of circuit instabilities and very low production yields, due to many dies unable to meet the specifications.

Bibliography

[1] SIA ITRS roadmap update 2006 - http://www.itrs.net/.

[2] http://en.wikipedia.org/wiki/moore's_law.

[3] Transistor elements for 30nm physical gate length and beyond. *Intel Technology Journal*, Vol. 06(No. 2):42–54, May 2002.

[4] H. Soeleman, K. Roy, and B. Paul. Robust ultra-low power sub-threshold DT-MOS logic. *International Symposium on Low Power Electronics and Design*, 2000.

[5] T. Enomoto, Y. Oka, H. Shikano, and T. Harada. A self controllable voltage level (SVL) circuit for low power high speed CMOS circuit. *European Solid-State Circuits Conference*, pages 411–414, 2002.

[6] S. Cserveny, J.-M. Masgonty, and C. Piguet. Stand-by power reduction for storage circuits. *PATMOS Conference*, September 2003.

[7] S.M. Kang and Y. Leblebici. *CMOS Digital Integrated Circuits: Analysis and Design, Third Edition*. McGraw-Hill, 2003.

[8] M. Anis and M. Elmasry. *Multi-Threshold CMOS Digital Circuits*. Kluwer Academic Publisher, 2003.

[9] K. Usami, N. Kawabe, M.Koizumi, K. Seta, and T. Furusawa. Automated selective Multi-Threshold design for ultra-low standby applications. *International Symposium on Low Power Electronics and Design*, 2002.

[10] J. Kao and A. Chandrakasan. MTCMOS sequential circuits. *European Solid-State Circuits Conference*, 2001.

[11] V.R. von Kaenel, M.D. Pardoen, E. Dijkstra, and E.A. Vittoz. Automatic adjustment of threshold & supply voltage for minimum power consumption CMOS

digital circuits. *IEEE Symposium on Low Power Electronics*, pages 78–79, October 1994.

[12] C. H. Kim and K. Roy. Dynamic Vt SRAM: A leakage tolerant cache memory for low voltage microprocessor. *International Symposium on Low Power Electronics and Design*, 2002.

[13] H. Mizuno, K. Ishibashi, T. Shimura, T. Hattori, S. Narita, K. Shiozawa, S. Ikeda, and K. Uchiyama. An 18uA standby current 1.8V, 200MHz microprocessor with self-substrate-biased data-retention mode. *IEEE International Solid-State Circuits Conference*, pages 280–281, 1999.

[14] F. Assaderaghi, D. Sinitsky, S. Parke, S. Bokor, P.K. Ko, and C.Hu. A dynamic threshold voltage MOSFET (DTMOS) for ultra-low voltage operation. *IEEE International Electron Devices Meeting Technical Digest*, pages 809–812, 1994.

[15] A. P. Chandrakasan and R.W. Brodersen. Low power CMOS digital design. *IEEE Journal of Solid-State Circuits*, Vol. 27(No. 4):473–484, April 1992.

[16] H.J.M Veendrick. Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *IEEE Journal of Solid-State Circuits*, Vol. 19(No. 4):468–473, August 1984.

[17] D. Auverne, P.Maurine, and N. Azémard. *Low Power Electronics Design, Chapter 6:Modeling for Designing in Deep Submicron Technologies*. CRC Press, 2005.

[18] K. Nose and T. Sakurai. Closed-form expression for short-circuit power of short channel CMOS gates and its scaling characteristics. *International Technical Conference on Circuits/Systems, Computers and Communications*, pages 1741–1744, 1998.

[19] S. Turgis, N. Azemard, and D. Auvergne. Short-circuit power dissipation calculation on CMOS inverters using the equivalent short-circuit capacitance concept. *PATMOS Conference*, 1995.

[20] T. Sakurai and A. R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *IEEE Journal of Solid-State Circuits*, Vol. 25(No. 2):584–594, April 1990.

[21] T. Sakurai and A.R. Newton. A simple MOSFET model for circuit analysis. *IEEE Transactions on Electron Devices*, Vol. 38(No. 4):887–894, April 1991.

[22] J.L. Rosselló and J. Segura. Accurate modelling of leakage currents in nanometre CMOS technologies. *Electronics Letters*, Vol. 41(No. 3):122–124, February 2005.

[23] Z. Chen, M. Johnson, L. Wei, and K. Roy. Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks. *International Symposium on Low Power Electronics and Design*, pages 239–244, 1998.

[24] B.J. Sheu, D.L.Scharfetter, P.-K. Ko, and M.-C. Jeng. BSIM: Berkley Short-Channel IGFET model for MOS transistors. *IEEE Journal of Solid-Sate Circuits*, Vol. 22(No. 4):558–566, August 1987.

[25] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meinand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE*, Vol. 91(No. 2):pp. 305–327, February 2003.

[26] N. S. Kim, T. Austin, D. Blaauw, T.Mudge, K. Flauter, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *IEEE Computer*, pages 68–75, December 2003.

[27] John Robertson. High dielectric constant gate oxides for metal oxide si transistors. *Reports on Progress in Physics*, Vol. 69:327–396, 2006.

[28] A. Keshavarzi, K. Roy, and C. F. Hawkins. Intrinsic leakage in low power deep submicron CMOS ICs. *IEEE International Test Conference*, pages 146–155, 1997.

[29] S. Mukhopadhyay and K. Roy. Modeling and estimation of total leakage current in nano-scaled CMOS devices considering the effect of parameter variation. *IEEE International Symposium on Circuits and Systems*, pages 172–175, 2003.

[30] A. Ferré and J. Figueras. *Low Power Electronics Design, Chapter 3: Leakage in CMOS nanometric technologies*. CRC Press, 2005.

[31] D. Helms, E. Schmidt, and W. Nebel. Tutorial: Leakage in CMOS circuits - an introduction. *PATMOS Conference*, pages 17–35, 2004.

[32] TSMC. *ANTCBN90G_110A - TSMC technology manual*.

[33] SIA ITRS roadmap 2004 - http://www.itrs.net/.

[34] http://www.intel.com/pressroom/archive/releases/20070128comp.htm.

[35] K. Nose and T. Sakurai. Optimization of Vdd and Vth for low-power and high-speed applications. *Asia South Pacific Design Automation Conference*, pages 469–474, January 2000.

[36] M. H. Fino. A simple submicron MOSFET model and its application to the analyitcal characterization of analog circuits. *European Conference on Circuit Theory and Design*, August 2005.

[37] K.A. Bowman, B.L. Austin, J.C. Eble, Xinghai Tang, and J.D. Meindl. A physical alpha-power law MOSFET model. *IEEE Journal of Solid-State Circuits*, Vol. 34(No. 10):1410–1414, October 1999.

[38] T. Sakurai. Alpha power-law MOS model. *IEEE Solid-State Circuits Society Newsletter*, Vol. 9(No. 4):4–5, October 2004.

[39] J.L Rosselló and J. Segura. Charge-based analytical model for evaluation of power consumption in submicron CMOS buffers. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and System*, Vol. 21(No. 4), April 2002.

[40] J.A. Butts and G.S. Sohi. A static power model for architects. *ACM International Symposium on Microarchitecture*, pages 191–201, December 2000.

[41] C.S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, Vol. 13:14–17, February 1964.

[42] P. C. H. Meier. *Analysis and Design of Low Power Digital Multipliers.* PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1999.

[43] H. Reza. A new multiplier using Wallace structure and carry select adder with pipelining. *IEEE International Symposium on Circuits and Systems*, 2002.

[44] R. Zimmermann. *Binary Adder Architectures for Cell-Based VLSI and their Synthesis.* PhD thesis, Swiss Federal Institute of Technology, Zurich, 1997.

[45] R. P. Brent and H. T. Kung. A regular layout for parallel adders. *IEEE Transactions on Computers*, Vol. 31(No. 3):260–264, 1982.

[46] W. J. Townsend, E. E. Swartzlander, Jr., and J. A. Abraham. A comparison of dadda and wallace multiplier delays. In F. T. Luk, editor, *Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, pages 552–560, December 2003.

[47] K.A.C. Bickerstaff, M. Schulte, and Jr. Swartzlander, E.E. Reduced area multipliers. *International Conference on Application-Specific Array Processors*, pages 478–489, October 1993.

[48] A. Wang and A. Chandrakasan. A 180-mV subthrehsold FFT processor using a minimum energy design methodology. *IEEE Journal of Solid-State Circuits*, Vol. 40(No. 1):310–319, January 2005.

[49] H. Q. Dao, B. R. Zeydel, and V. G. Oklobdzija. Architectural considerations for energy efficiency. *International Conference on Computer Design*, pages 13–16, 2005.

[50] B. Zhai et. al. A 2.6pJ/Inst subthreshold sensor processor for optimal energy efficiency. *VLSI Circuits Symposium*, 2006.

[51] B. H. Calhoun and A. Chandrakasan. Characterizing and modeling minimum energy operation for subthrehsold circuits. *International Symposium on Low Power Electronics and Design*, pages 90–95, 2004.

[52] S. Hanson, B. Zhai, D. Blaauw, D. Sylvester, A. Bryant, and X. Wang. Energy optimality and variability in subthreshold design. *International Symposium on Low Power Electronics and Design*, pages 363–365, October 2006.

[53] D. Markovic, V. Stojanovic, B. Nikolic, M. A. Horowitz, and R. W. Brodersen. Methods for true energy-performance optimization. *IEEE Journal of Solid-State Circuits*, Vol. 39(No. 8):1282–1293, August 2004.

[54] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. *Design Automation Conference*, pages 868–873, 2004.

[55] J. Burr and A. Peterson. Ultra low power CMOS technology. *NASA VLSI Design Symposium*, pages 4.2.1–4.2.13, 1991.

[56] C. Heer and J. Berthold. Designing low power circuits: an industrial point of view. *PATMOS Conference*, September 2001.

[57] S.M. Sze. *Semiconductor Devices - Physics and Technology.* John Wiley & Sons, 1985.

[58] H. Ananthan, C. H. Kim, and K. Roy. Larger-then-Vdd forward body bias in sub-0.5 V nanoscale CMOS. *IEEE International Symposium Low Power Electronics and Design*, pages 8–13, 2004.

[59] H. Ananthan. Evaluation of digital forward body bias for 70nm bulk CMOS. *Class Project, EE 695K*, Fall 2003.

[60] T. Kuroda et al. A 0.9V, 150MHz, 10mW, 4 mm2, 2D discrete cosine transform core processor with variable threshold voltage scheme. *IEEE Journal of Solid-State Circuits*, Vol. 31(No. 11):1770–1779, 1996.

[61] S. Narendra et al. 1.1V 1GHz communications router with on-chip body bias in 150nm CMOS. *IEEE International Solid-State Circuits Conference*, pages 270–271, 2002.

[62] P. Gupta, A.B. Kahng, P. Sharma, and D. Sylvester. Gate-length biasing for runtime-leakage control. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25:1475–1485, 2006.

[63] http://en.wikipedia.org/wiki/linear_feedback_shift_register.

[64] XilinX. *Xilinx LogiCORE: Linear Feedback Shift Register V3.0*, 28 March 2003.

[65] http://www.nova-eng.com/inside.asp?n=products&p=constellation.

List of Publications

Conferences

- C. Piguet, C. Schuster, J.-L. Nagel. "Static and dynamic power reduction by architecture selection". Proc. Int'l Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS'06, Montpellier, France, September 13-15, 2006.

- C. Piguet, C. Schuster, J.-L. Nagel. "Leakage reduction at architecture level". Proc. Int'l Conference on Integrated Circuit Design & Technology, ICICDT'06, Padova, Italy, May 24-26, 2006.

- C. Schuster, J.-L. Nagel, C. Piguet, P.-A. Farine. "Architectural and technology influence on the optimal total power consumption". Design, Automation and Test in Europe Conference, DATE06, Munich, Germany, March 06-10, 2006.

- C. Piguet, C. Schuster, J.-L. Nagel. "Réduction des consommations statique et dynamique par sélection des architectures". 5ème journées d'études Faible Tension Faible Consommation, FTFC05, Paris, May 18-19, 2005.

- C. Schuster, J.-L. Nagel, C. Piguet, P.-A. Farine. "Conception d'architectures à Vdd et Vth imposés avec consommation totale minimale". Journées Francophones sur l'Adéquation Algorithme Architecture, JFAAA'05, Dijon, France, January 18-21, 2005.

- C. Schuster, J.-L. Nagel, C. Piguet, P.-A. Farine. "Leakage reduction at the architectural level and its application to 16 bit multiplier architectures". Proc. Int'l Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS'04, Santorini Island, Greece, September 15-17, 2004.

- C. Piguet, C. Schuster, J.-L. Nagel. "Optimizing architecture activity and logic depth for static and dynamic power reduction". Proc. of the 2nd Northeast

Workshop on Circuits and Systems, NewCAS'04, Montréal, Canada, June 20-23, 2004

Journals

- C. Schuster, J.-L. Nagel, C. Piguet, P.-A. Farine. "An architecture design methodology for minimal total power consumption at fixed Vdd and Vth". Journal of Low Power Electronics, Vol.1(No.1):pp.3-10, April, 2005.

Appendix A

VHDL source code

A.1 top.vhd

```
1  ------------------------------------------------------------------------------
2  -- Title      : Circuit top (32bit)
3  -- Project    :
4  ------------------------------------------------------------------------------
5  -- File       : top.vhd
6  -- Author     :   <schuster@zebra>
7  -- Company    :
8  -- Created    : 2006-02-17
9  -- Last update: 2006-09-22
10 -- Platform   :
11 -- Standard   : VHDL'93
12 ------------------------------------------------------------------------------
13 -- Description: This is the top of the design.
14 --              It includes the follow blocks:
15 --              - data_gen
16 --              - 2 mult32
17 --              - 2 mult32 parallel 4
18 --              - one-hot decoder and mux
19 --              - 2 ring oscillators
20 ------------------------------------------------------------------------------
21 -- Copyright (c) 2006
22 ------------------------------------------------------------------------------
23 -- Revisions  :
24 -- Date        Version  Author  Description
25 -- 2006-02-17  1.0      schuster        Created
26 ------------------------------------------------------------------------------
27
28 library ieee;
29 use ieee.std_logic_1164.all;
30
31 entity top is
32
33   port (
34     clk     : in  std_logic;            -- clock
35     rst_n   : in  std_logic;            -- active low async reset
```

```
36 ------------------------------------------------------------------------------
37     s_in     : in  std_logic;            -- serial input
38     s_out    : out std_logic;            -- serial output
39     load_n   : in  std_logic;  -- when low registers are loaded in parallel
40     shift_n  : in  std_logic;            -- when low data is shifted
41 ------------------------------------------------------------------------------
42     -- select the source for data_out as well as for the data saved in registers
43     sel_reg  : in  std_logic;
44     sel      : in  std_logic_vector(1 downto 0);  -- select the working multiplier
45     Z_svt    : out std_logic;            -- out svt ring oscillator
46     Z_lvt    : out std_logic);           -- out lvt ring oscillator
47
48 end top;
49
50 architecture arch of top is
51
52   component data_gen
53     port (
54       clk         : in  std_logic;
55       rst_n       : in  std_logic;
56       data_in_v   : in  std_logic_vector(63 downto 0);
57       data_out_v  : out std_logic_vector(63 downto 0);
58       s_in        : in  std_logic;
59       s_out       : out std_logic;
60       load_n      : in  std_logic;
61       shift_n     : in  std_logic;
62       sel_reg     : in  std_logic);
63   end component;
64
65   component mult
66     port (
67       clk   : in  std_logic;
68       rst_n : in  std_logic;
69       en    : in  std_logic;
70       a_v   : in  std_logic_vector(31 downto 0);
71       b_v   : in  std_logic_vector(31 downto 0);
72       m_v   : out std_logic_vector(63 downto 0));
73   end component;
74
75   component mult_par4
76     port (
77       clk   : in  std_logic;
78       rst_n : in  std_logic;
79       en    : in  std_logic;
80       a_v   : in  std_logic_vector (31 downto 0);
81       b_v   : in  std_logic_vector (31 downto 0);
82       m_v   : out std_logic_vector (63 downto 0));
83   end component;
84
85   component ring_svt
86     generic (
87       length : integer);
88     port (
89       Z : out std_logic);
90   end component;
```

```
91
92    component ring_lvt
93      generic (
94        length : integer);
95      port (
96        Z : out std_logic);
97    end component;
98
99    -- demultiplexed mutlipliers output
100   signal general_m            : std_logic_vector(63 downto 0);
101   -- multipliers input data containing both A and B
102   signal general_a_b          : std_logic_vector(63 downto 0);
103   -- multipliers input data separeted as A and B
104   signal general_a, general_b : std_logic_vector(31 downto 0);
105
106
107   signal m0_v, m1_v, m2_v, m3_v : std_logic_vector(63 downto 0);  -- multipliers
108                                                                   -- results
109   signal en0, en1, en2, en3     : std_logic;  -- multipliers registers enable
110   signal clk0, clk1, clk2, clk3 : std_logic;  -- multipliers registers clock
111
112 begin  -- arch
113   ------------------------------------------------------------------------------
114   -- component mapping
115   ------------------------------------------------------------------------------
116   data_gen_1 : data_gen
117     port map (
118       clk        => clk,
119       rst_n      => rst_n,
120       data_in_v  => general_m,
121       data_out_v => general_a_b,
122       s_in       => s_in,
123       s_out      => s_out,
124       load_n     => load_n,
125       shift_n    => shift_n,
126       sel_reg    => sel_reg);
127
128   mult_0 : mult
129     port map (
130       clk   => clk0,
131       rst_n => rst_n,
132       en    => en0,
133       a_v   => general_a,
134       b_v   => general_b,
135       m_v   => m0_v);
136   mult_1 : mult_par4
137     port map (
138       clk   => clk1,
139       rst_n => rst_n,
140       en    => en1,
141       a_v   => general_a,
142       b_v   => general_b,
143       m_v   => m1_v);
144   mult_2 : mult
145     port map (
```

```
146       clk   => clk2,
147       rst_n => rst_n,
148       en    => en2,
149       a_v   => general_a,
150       b_v   => general_b,
151       m_v   => m2_v);
152   mult_3 : mult_par4
153     port map (
154       clk   => clk3,
155       rst_n => rst_n,
156       en    => en3,
157       a_v   => general_a,
158       b_v   => general_b,
159       m_v   => m3_v);
160
161   ring_svt_1: ring_svt
162     generic map (
163       length => 437)
164     port map (
165       Z => Z_svt);
166   ring_lvt_1: ring_lvt
167     generic map (
168       length => 533)
169     port map (
170       Z => Z_lvt);
171
172   ------------------------------------------------------------------------
173   --combinatorial part
174   ------------------------------------------------------------------------
175   general_a <= general_a_b(63 downto 32);
176   general_b <= general_a_b(31 downto 0);
177
178
179   -- one-hot decoder
180   en0  <= '1' when sel = "00" else '0';
181   en1  <= '1' when sel = "01" else '0';
182   en2  <= '1' when sel = "10" else '0';
183   en3  <= '1' when sel = "11" else '0';
184
185   -- clock demux
186   clk0 <= clk when sel = "00" else '0';
187   clk1 <= clk when sel = "01" else '0';
188   clk2 <= clk when sel = "10" else '0';
189   clk3 <= clk when sel = "11" else '0';
190
191   -- ouput mux
192   with sel select
193     general_m <=
194     m0_v when "00",
195     m1_v when "01",
196     m2_v when "10",
197     m3_v when "11",
198     m0_v when others;
199
200   ------------------------------------------------------------------------
```

```
201   --sequential part
202   ------------------------------------------------------------------------------
203
204 end arch;
```

A.2 data_gen.vhd

```
1   ------------------------------------------------------------------------------
2   -- Title      : Data generator (32bit)
3   -- Project    :
4   ------------------------------------------------------------------------------
5   -- File       : data_gen.vhd
6   -- Author     :   <schuster@zebra>
7   -- Company    :
8   -- Created    : 2006-02-15
9   -- Last update: 2006-09-22
10  -- Platform   :
11  -- Standard   : VHDL'93
12  ------------------------------------------------------------------------------
13  -- Description: This block contains the pseudo-random data generator,
14  --              as well as the cyclic adder and corresponding registers.
15  --              Both generated random and input data can be outputed in
16  --              serial by the shift_reg block.
17  ------------------------------------------------------------------------------
18  -- Copyright (c) 2006
19  ------------------------------------------------------------------------------
20  -- Revisions  :
21  -- Date        Version  Author  Description
22  -- 2006-02-15  1.0      schuster        Created
23  ------------------------------------------------------------------------------
24  library ieee;
25  use ieee.std_logic_1164.all;
26  use ieee.std_logic_unsigned.all;        -- used to add std_logic_vectors
27
28  -- designware implementation of the
29  -- shift register
30  library DWARE, DW03;
31  use DWARE.DWpackages.all;
32  use DW03.DW03_components.all;
33
34  entity data_gen is
35    port (
36      clk        : in  std_logic;         -- clock
37      rst_n      : in  std_logic;         -- active low async reset
38  ------------------------------------------------------------------------------
39      -- data coming from the multipliers (result)
40      data_in_v  : in  std_logic_vector(63 downto 0);
41      -- generated_data (extern or pseudo random)
42      data_out_v : out std_logic_vector(63 downto 0);
43  ------------------------------------------------------------------------------
44      s_in       : in  std_logic;         -- serial input
45      s_out      : out std_logic;         -- serial output
46      load_n     : in  std_logic;  -- when low registers are loaded in parallel
47      shift_n    : in  std_logic;         -- when low data is shifted
```

```
48 ------------------------------------------------------------------------------
49     -- select the source for data_out as well as for the data saved in registers
50     sel_reg     : in  std_logic
51     );
52
53 end data_gen;
54
55 architecture arch of data_gen is
56   -- DesignWare shift register
57   component DW03_shfreg
58     generic (
59       inst_length : integer);
60       port (
61       inst_clk      : in  std_logic;
62       inst_s_in     : in  std_logic;
63       inst_p_in     : in  std_logic_vector(inst_length-1 downto 0);
64       inst_shift_n  : in  std_logic;
65       inst_load_n   : in  std_logic;
66       p_out_inst    : out std_logic_vector(inst_length-1 downto 0));
67   end component;
68
69   -- local signals
70   signal sum_v        : std_logic_vector(63 downto 0);  -- result of the adder
71   signal p_out_v      : std_logic_vector(63 downto 0);  -- output of the register bank
72   signal p_in_v       : std_logic_vector(63 downto 0);  -- input of the register bank
73   signal rand_data_v  : std_logic_vector(63 downto 0);  -- output of the pseudo
74                                                         -- random generator
75   signal next_rand_data_v : std_logic_vector(63 downto 0);  -- next of rand_data
76
77   -- local constants
78   constant inst_length : natural := 64;  -- size of the shift register bank
79
80 begin  -- arch
81
82   -- recursive cyclic adder
83   adder :
84     sum_v <= data_in_v + p_out_v;
85
86   -- link the highest bit of p_out_v to s_out
87   s_out <= p_out_v(63);
88
89
90   ------------------------------------------------------------------------------
91   -- instance of the shift_register
92   -- based on a DW model
93   shift_register: DW03_shftreg
94       generic map (length => inst_length)
95       port map (clk     => clk, s_in => s_in, p_in => p_in_v,
96                 shift_n => shift_n, load_n => load_n,
97                 p_out   => p_out_v);
98
99
100  ------------------------------------------------------------------------------
101  -- purpose: instanciation of mux1 and mux2
102  -- type    : combinational
```

```
103   -- inputs : sel_reg, sum_v, rand_data_v, p_out_v
104   -- outputs: p_in_v, data_out_v
105   muxes : process (sel_reg, sum_v, rand_data_v, p_out_v)
106   begin  -- process
107     case sel_reg is
108       when '0' =>
109         p_in_v       <= sum_v;
110         data_out_v <= rand_data_v;
111       when '1' =>
112         p_in_v       <= rand_data_v;
113         data_out_v <= p_out_v;
114       when others => null;
115     end case;
116   end process;
117
118
119   -------------------------------------------------------------------------------
120   -- pseudo-random code generator
121   -- the next bit is based on the
122   -- taps 63, 61, 60, 0
123   -- the state to avoid is 1...1
124   pseudo_rand_logic:
125     next_rand_data_v <= ((rand_data_v(63) xnor rand_data_v(61)) xnor
126                          (rand_data_v(60) xnor rand_data_v(0))) &
127                           rand_data_v(63 downto 1);
128
129   -------------------------------------------------------------------------------
130   -- purpose: insert shift register bank used for the pseudo code generation
131   -- type   : sequential
132   -- inputs : clk, rst_n, next_rand_data_v
133   -- outputs: rand_data_v
134   pseudo_rand_regs: process (clk, rst_n)
135   begin  -- process
136     if rst_n = '0' then                 -- asynchronous reset (active low)
137       rand_data_v <= (others => '0');
138     elsif clk'event and clk = '1' then  -- rising clock edge
139       rand_data_v <= next_rand_data_v;
140     end if;
141   end process;
142
143
144 end arch;
```

A.3 mult.vhd

```
1  -------------------------------------------------------------------------------
2  -- Title      : Simple multiplier (32bit)
3  -- Project    :
4  -------------------------------------------------------------------------------
5  -- File       : mult.vhd
6  -- Author     :   <schuster@zebra>
7  -- Company    :
8  -- Created    : 2006-02-16
9  -- Last update: 2006-09-22
```

```
10 -- Platform    :
11 -- Standard    : VHDL'93
12 ----------------------------------------------------------------------
13 -- Description: simple multiplier block with registers
14 ----------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ----------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-02-16  1.0      schuster        Created
20 ----------------------------------------------------------------------
21 library ieee;
22 use ieee.std_logic_1164.all;
23 use ieee.std_logic_unsigned.all;
24
25
26 entity mult is
27   port (
28     clk    : in  std_logic;                -- clock
29     rst_n  : in  std_logic;                -- active low async reset
30     en     : in  std_logic;                -- registers enable
31 ----------------------------------------------------------------------
32     a_v    : in  std_logic_vector(31 downto 0);  -- input A
33     b_v    : in  std_logic_vector(31 downto 0);  -- input B
34     m_v    : out std_logic_vector(63 downto 0)   -- result
35     );
36 end mult;
37
38 architecture arch of mult is
39
40   -- generic RCA multiplier declaration
41   component RCA
42     generic (
43       A_width : integer;                -- size of A
44       B_width : integer);               -- size of B
45     port (
46       S          : out std_logic_vector (A_width+B_width-1 downto 0);
47       A          : in  std_logic_vector (A_width-1 downto 0);
48       B          : in  std_logic_vector (B_width-1 downto 0));
49   end component;
50
51   -- local signals
52   signal a_int_v : std_logic_vector(31 downto 0);
53   signal b_int_v : std_logic_vector(31 downto 0);
54   signal m_int_v : std_logic_vector(63 downto 0);
55
56 begin  -- arch
57
58   ----------------------------------------------------------------------
59   --combinatorial part
60   ----------------------------------------------------------------------
61
62   -- infer the 32 bit multiplier
63   mult_1: RCA
64     generic map (
```

```
65        A_width => 32,
66        B_width => 32)
67      port map (
68        S           => m_int_v,
69        A           => a_int_v,
70        B           => b_int_v);
71
72    ------------------------------------------------------------------------------
73    -- sequential part
74    ------------------------------------------------------------------------------
75
76    -- purpose: input and output registers
77    -- type    : sequential
78    -- inputs : clk, rst_n, a_v, b_v, m_int_v
79    -- outputs: a_int_v, b_int_v, m_v
80    mult1_regs : process (clk, rst_n)
81    begin  -- process mult1_regs
82      if rst_n = '0' then                 -- asynchronous reset (active low)
83        a_int_v <= (others => '0');
84        b_int_v <= (others => '0');
85        m_v     <= (others => '0');
86      elsif clk'event and clk = '1' then  -- rising clock edge
87        if en = '1' then
88          a_int_v <= a_v;
89          b_int_v <= b_v;
90          m_v     <= m_int_v;
91        end if;
92      end if;
93    end process mult1_regs;
94
95  end arch;
```

A.4 mult_par4.vhd

```
1   ------------------------------------------------------------------------------
2   -- Title      : 4 times parallel multiplier (32bit)
3   -- Project    :
4   ------------------------------------------------------------------------------
5   -- File       : mult_par4.vhd
6   -- Author     :   <schuster@zebra>
7   -- Company    :
8   -- Created    : 2006-02-16
9   -- Last update: 2006-09-22
10  -- Platform   :
11  -- Standard   : VHDL'93
12  ------------------------------------------------------------------------------
13  -- Description: 4 times parallel multiplier block with registers
14  ------------------------------------------------------------------------------
15  -- Copyright (c) 2006
16  ------------------------------------------------------------------------------
17  -- Revisions  :
18  -- Date        Version  Author  Description
19  -- 2006-02-16  1.0      schuster         Created
20  ------------------------------------------------------------------------------
```

```
21 library ieee;
22 use ieee.std_logic_1164.all;
23 use ieee.std_logic_unsigned.all;
24
25 entity mult_par4 is
26   port(
27     clk   : in  std_logic;       -- clock
28     rst_n : in  std_logic;       -- active low async
29     en    : in  std_logic;       -- registers enable
30     ------------------------------------------------------------------------
31     a_v   : in  std_logic_vector (31 downto 0); -- input A
32     b_v   : in  std_logic_vector(31 downto 0);  -- input B
33     m_v   : out std_logic_vector(63 downto 0)   -- result
34     );
35 end mult_par4;
36
37
38 architecture arch of mult_par4 is
39
40   -- generic RCA multiplier declaration
41   component RCA
42     generic (
43       A_width : integer;         -- size of A
44       B_width : integer);        -- size of B
45     port (
46       S          : out std_logic_vector (A_width+B_width-1 downto 0);
47       A          : in  std_logic_vector (A_width-1 downto 0);
48       B          : in  std_logic_vector (B_width-1 downto 0));
49     end component;
50
51   signal count, next_count              : std_logic_vector(1 downto 0);
52   signal A0, A1, A2, A3, B0, B1, B2, B3 : std_logic_vector(31 downto 0);
53   signal S, S0, S1, S2, S3              : std_logic_vector(63 downto 0);
54
55 begin
56
57 ------------------------------------------------------------------------
58 --combinatorial part
59 ------------------------------------------------------------------------
60
61 -- output multiplexer
62   with count select
63     S <= S0 when "00",
64     S1      when "01",
65     S2      when "11",
66     S3      when "10",
67     S3      when others;
68
69 --multiplexers counter incrementer 00 -> 01 -> 11 -> 10 ->
70   next_count <= "01" when count = "00" else
71                 "11" when count = "01" else
72                 "10" when count = "11" else
73                 "00";
74
75 --implementation of the four 32bit multipliers
```

```
76    mult_par4_0 : RCA
77      generic map (
78        A_width => 32,
79        B_width => 32)
80      port map (
81        S           => S0,
82        A           => A0,
83        B           => B0);
84    mult_par4_1 : RCA
85      generic map (
86        A_width => 32,
87        B_width => 32)
88      port map (
89        S           => S1,
90        A           => A1,
91        B           => B1);
92    mult_par4_2 : RCA
93      generic map (
94        A_width => 32,
95        B_width => 32)
96      port map (
97        S           => S2,
98        A           => A2,
99        B           => B2);
100   mult_par4_3 : RCA
101     generic map (
102       A_width => 32,
103       B_width => 32)
104     port map (
105       S           => S3,
106       A           => A3,
107       B           => B3);
108
109   ------------------------------------------------------------------------
110   --sequential part
111   ------------------------------------------------------------------------
112   process (clk, rst_n)
113   begin
114     if rst_n = '0' then
115       m_v   <= (others => '0');
116       count <= "00";
117       A0    <= (others => '0');
118       A1    <= (others => '0');
119       A2    <= (others => '0');
120       A3    <= (others => '0');
121       B0    <= (others => '0');
122       B1    <= (others => '0');
123       B2    <= (others => '0');
124       B3    <= (others => '0');
125     elsif clk = '1' and clk'event then
126       if en = '1' then
127         -- output registers
128         m_v                    <= S;
129         -- increment state machine counter
130         count                  <= next_count;
```

```
131         -- input reg and demultiplexer for A
132         if count = "00" then A0 <= a_v; end if;
133         if count = "01" then A1 <= a_v; end if;
134         if count = "11" then A2 <= a_v; end if;
135         if count = "10" then A3 <= a_v; end if;
136         -- input reg and demultiplexer for B
137         if count = "00" then B0 <= b_v; end if;
138         if count = "01" then B1 <= b_v; end if;
139         if count = "11" then B2 <= b_v; end if;
140         if count = "10" then B3 <= b_v; end if;
141       end if;
142     end if;
143   end process;
144
145 end arch;
```

A.5 RCA_generic_arch.vhd

```
1  ----------------------------------------------------------------------------
2  -- Title      : Genreric RCA Mulitplier
3  -- Project    :
4  ----------------------------------------------------------------------------
5  -- File       : RCA_generic_arch.vhd
6  -- Author     :   <mtschuster@WS-3439>
7  -- Company    :
8  -- Created    : 2006-04-27
9  -- Last update: 2006-09-22
10 -- Platform   :
11 -- Standard   : VHDL'93
12 ----------------------------------------------------------------------------
13 -- Description: Simple Ripple Carry Array multiplier implementation
14 ----------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 ----------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-04-27  1.0      mtschuster      Created
20 ----------------------------------------------------------------------------
21 library ieee;
22 use ieee.std_logic_1164.all;
23 use ieee.std_logic_unsigned.all;
24
25 entity RCA is
26   generic(
27     A_width : integer := 32;            --size of A
28     B_width : integer := 32);           --size of B
29   port(
30     S          : out std_logic_vector (A_width+B_width-1 downto 0);  --output
31     A          : in  std_logic_vector (A_width-1 downto 0);   --input A
32     B          : in  std_logic_vector (B_width-1 downto 0));  --input B
33 end RCA;
34
35 architecture arch of RCA is
36
```

```
37    type std_logic_array is                -- array of internal nodes
38      array (B_width-1 downto 1) of std_logic_vector (A_width-1 downto 0);
39    type enlarged_std_logic_array is       -- extended array of internal nodes
40      array (B_width-1 downto 0) of std_logic_vector (A_width downto 0);
41
42    --local signals
43    signal AandB      : std_logic_array;   --partial products
44    signal S_partial : enlarged_std_logic_array; --internal sums
45    signal Init_val   : std_logic_vector(A_width-1 downto 0); --first line values
46
47  begin
48
49    ------------------------------------------------------------------------------
50    --combinatorial part
51    ------------------------------------------------------------------------------
52
53    first_cell :                           --implemet first line
54    Init_val <= A when B(0) = '1' else (others => '0');
55    S_partial(0) <= '0'&Init_val;
56    S(0)         <= S_partial(0)(0);
57
58    int_cell :                             --implement internal lines
59    for i in 1 to B_width-1 generate
60      S(i)     <= S_partial(i)(0);
61      AandB(i) <= A when B(i) = '1' else (others => '0');
62      S_partial(i) <= ('0'&S_partial(i-1)(A_width downto 1))+('0'&AandB(i));
63    end generate;
64
65    last_cell :                            --copy the result to the output
66      S(A_width+B_width-1 downto B_width) <= S_partial(B_width-1)(A_width downto 1);
67
68    ------------------------------------------------------------------------------
69    --sequential part
70    ------------------------------------------------------------------------------
71  end arch;
```

A.6 ring_svt.vhd

```
1   ------------------------------------------------------------------------------
2   -- Title      : Ring oscillator SVT
3   -- Project    :
4   ------------------------------------------------------------------------------
5   -- File       : ring_svt.vhd
6   -- Author     :   <schuster@zebra>
7   -- Company    :
8   -- Created    : 2006-04-27
9   -- Last update: 2006-09-22
10  -- Platform   :
11  -- Standard   : VHDL'93
12  ------------------------------------------------------------------------------
13  -- Description: A simple ring oscillator with direct instatiation of the
14  --              STM 090 SVT technology inverters. Inverter type is IVSVTX1
15  ------------------------------------------------------------------------------
16  -- Copyright (c) 2006
```

```
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-04-27  1.0      schuster     Created
20 ------------------------------------------------------------------------------
21
22 library ieee;
23 use ieee.std_logic_1164.all;
24
25 entity ring_svt is
26   generic (
27     length : integer := 100);          -- default inverter chain length
28   port (
29     Z : out std_logic                  -- ring oscillator output
30 );
31 end ring_svt;
32
33 architecture arch of ring_svt is
34
35   --local signal
36   signal internal_nets : std_logic_vector(length-1 downto 0);
37
38   --declare the technology inverter
39   component IVSVTX1
40     port(
41       Z      :    out STD_LOGIC ;  --in
42       A      :    in  STD_LOGIC    --out
43       );
44   end component;
45
46 begin  -- arch
47
48   --connect each inverter with the follow
49   invs: for i in 0 to length-2 generate
50     IVSVTX1_gen: IVSVTX1
51       port map (
52         Z => internal_nets(i),
53         A => internal_nets(i+1));
54   end generate invs;
55
56   --connect last inverter with the first
57   IVSVTX1_last: IVSVTX1
58     port map (
59       Z => internal_nets(length-1),
60       A => internal_nets(0));
61
62   --output the first node
63   Z <= internal_nets(0);
64
65 end arch;
```

A.7 top_tb.vhd

```
1 ------------------------------------------------------------------------------
2 -- Title      : Testbench for design "top"
```

```
3  -- Project    :
4  -------------------------------------------------------------------------------
5  -- File       : top_tb.vhd
6  -- Author     :   <schuster@zebra>
7  -- Company    :
8  -- Created    : 2006-02-17
9  -- Last update: 2006-09-22
10 -- Platform   :
11 -- Standard   : VHDL'93
12 -------------------------------------------------------------------------------
13 -- Description: Testbench for design "top"
14 -------------------------------------------------------------------------------
15 -- Copyright (c) 2006
16 -------------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2006-02-17  1.0      schuster        Created
20 -------------------------------------------------------------------------------
21
22 library ieee;
23 use ieee.std_logic_1164.all;
24 use ieee.std_logic_textio.all;          -- write std_logic signal to line
25 use ieee.std_logic_unsigned.all;
26 library std;
27 use std.textio.all;             -- output data to std_output or text file
28
29 -------------------------------------------------------------------------------
30
31 entity top_tb is
32
33 end top_tb;
34
35 -------------------------------------------------------------------------------
36
37 architecture top_func_test of top_tb is
38
39   --component decalration
40   component top
41     port (
42       clk      : in  std_logic;          --clock
43       rst_n    : in  std_logic;          --active low async reset
44 -------------------------------------------------------------------------------
45       s_in     : in  std_logic;          --serial data in
46       s_out    : out std_logic;          --serial data out
47       load_n   : in  std_logic;          --registers parallel load when low
48       shift_n  : in  std_logic;          --shift data when low
49 -------------------------------------------------------------------------------
50       -- select the source for data_out as well as for data saved in registers
51       sel_reg  : in  std_logic;
52       sel      : in  std_logic_vector(1 downto 0); --select multiplier
53       Z_svt    : out std_logic;          --SVT ring oscillator
54       Z_lvt    : out std_logic);         --LVT ring oscillator
55   end component;
56
57   -- component ports
```

```
58    signal rst_n    : std_logic;            --active low async reset
59    signal s_in     : std_logic;            --serial data in
60    signal s_out    : std_logic;            --serial data out
61    signal load_n   : std_logic;            --registers parallel load when low
62    signal shift_n  : std_logic;            --shift data when low
63    signal sel_reg  : std_logic;            --registers select
64    signal sel      : std_logic_vector(1 downto 0); --select multiplier
65    signal Z_svt    : std_logic;            --SVT ring oscillator
66    signal Z_lvt    : std_logic;            --LVT ring oscillator
67    -- clock
68    signal clk : std_logic := '1';          --clock
69
70    -- constants
71    constant HALFCLOCKPERIOD : time := 8 ns;
72
73  begin  -- top_func_test
74
75    -- component instantiation
76    DUT :  top
77      port map (
78        clk      => clk,
79        rst_n    => rst_n,
80        s_in     => s_in,
81        s_out    => s_out,
82        load_n   => load_n,
83        shift_n  => shift_n,
84        sel_reg  => sel_reg,
85        sel      => sel,
86        Z_svt    => Z_svt,
87        Z_lvt    => Z_lvt);
88
89    -- clock generation
90    clk <= not clk after HALFCLOCKPERIOD;
91
92    --main testbench processor
93    main : process
94  ------------------------------------------------------------------------
95      variable match_v : boolean;           -- check_output_data
96      variable pass_v  : boolean;           -- detect if an error accured
97
98      variable global_pass_v : boolean := true; -- pass flag for all tests
99
100     variable d_l             : line; -- output line for debugging purpose
101
102     variable DEBUG_MODE : boolean := false; -- activate/desactivate verbose mode
103
104 ------------------------------------------------------------------------
105
106     -- procedures
107 ------------------------------------------------------------------------
108     --check if specified test pass
109 ------------------------------------------------------------------------
110     procedure check_errors(test_name : in string) is
111     begin
112       if pass_v then
```

```
113         report test_name & "____ALL PASS____";
114       else
115         report test_name & "_____FAILED_____";
116       end if;
117       global_pass_v := global_pass_v and pass_v;
118     end procedure check_errors;
119 ------------------------------------------------------------------------
120     --check if all tests pass
121 ------------------------------------------------------------------------
122     procedure global_check is
123     begin
124       if global_pass_v then
125         report  "____ALL TESTS PASS -> OK!_____";
126       else
127         report  "___ONE OR MORE TEST FAILED____";
128       end if;
129     end procedure global_check;
130 ------------------------------------------------------------------------
131     --write and read specific patterns to/from the shift register
132 ------------------------------------------------------------------------
133     procedure serial_read_write is
134       --four different pattern to test
135       constant SERIAL_DATA_IN0 : std_logic_vector(63 downto 0) := (others => '0');
136       constant SERIAL_DATA_IN1 : std_logic_vector(63 downto 0) := (others => '1');
137       constant SERIAL_DATA_IN2 : std_logic_vector(63 downto 0) :=
138         X"5555555555555555"; --"0101010101010...1010101010101010101";
139       constant SERIAL_DATA_IN3 : std_logic_vector(63 downto 0) :=
140         X"AAAAAAAAAAAAAAAA"; --"1010101010101...0101010101010101010";
141
142       variable serial_data_out : std_logic_vector(63 downto 0);  -- read data
143                                                                  -- from the registers
144     begin
145       -- init
146       rst_n   <= '0';
147       load_n  <= '1';
148       shift_n <= '1';
149       s_in    <= '0';
150       sel_reg <= '1';
151       sel     <= "00";
152
153       pass_v := true;
154       wait for 9*HALFCLOCKPERIOD;
155       wait until clk='0';
156
157       -- write first pattern
158       shift_n <= '0';
159       for i in 63 downto 0 loop
160         s_in <= SERIAL_DATA_IN0(i);
161         wait until clk = '0';
162       end loop;  -- i
163
164       --write second pattern and read first pattern
165       for i in 63 downto 0 loop
166         s_in <= SERIAL_DATA_IN1(i);
167         serial_data_out(i) := s_out;
```

```
168         wait until clk = '0';
169       end loop;  -- i
170
171       --check first pattern
172       match_v := serial_data_out = SERIAL_DATA_IN0;
173       pass_v  := pass_v and match_v;
174       assert serial_data_out = SERIAL_DATA_IN0
175         report "Error on SERIAL_DATA_IN0" severity error;
176       --ieee.std_logic_textio.write(d_l,serial_data_out);  writeline(output, d_l);
177       if DEBUG_MODE then
178         report "SERIAL_DATA_IN0 read!" severity note;
179       end if;
180
181       --write third pattern and read second pattern
182       for i in 63 downto 0 loop
183         s_in               <= SERIAL_DATA_IN2(i);
184         serial_data_out(i) := s_out;
185         wait until clk = '0';
186       end loop;  -- i
187
188       --check second pattern
189       match_v := serial_data_out = SERIAL_DATA_IN1;
190       pass_v  := pass_v and match_v;
191       assert serial_data_out = SERIAL_DATA_IN1
192         report "Error on SERIAL_DATA_IN1" severity error;
193       if DEBUG_MODE then
194         report "SERIAL_DATA_IN1 read!" severity note;
195       end if;
196
197       --write fourth pattern and read the third pattern
198       for i in 63 downto 0 loop
199         s_in               <= SERIAL_DATA_IN3(i);
200         serial_data_out(i) := s_out;
201         wait until clk = '0';
202       end loop;  -- i
203
204       --check the third pattern
205       match_v := serial_data_out = SERIAL_DATA_IN2;
206       pass_v  := pass_v and match_v;
207       assert serial_data_out = SERIAL_DATA_IN2
208         report "Error on SERIAL_DATA_IN2" severity error;
209       if DEBUG_MODE then
210         report "SERIAL_DATA_IN2 read!" severity note;
211       end if;
212
213       --read the fourth pattern
214       for i in 63 downto 0 loop
215         serial_data_out(i) := s_out;
216         wait until clk = '0';
217       end loop;  -- i
218
219       --check the fourth pattern
220       match_v := serial_data_out = SERIAL_DATA_IN3;
221       pass_v  := pass_v and match_v;
222       assert serial_data_out = SERIAL_DATA_IN3
```

```
223         report "Error on SERIAL_DATA_IN3" severity error;
224       if DEBUG_MODE then
225         report "SERIAL_DATA_IN3 read!" severity note;
226       end if;
227     end serial_read_write;
228 ------------------------------------------------------------------------------
229     -- check that the shift register can be reset from  the pseudo random generator
230 ------------------------------------------------------------------------------
231     procedure check_shift_reg_rst is
232       variable serial_data_out : std_logic_vector(63 downto 0);
233     begin  -- check_shift_reg_rst
234       -- init
235       rst_n   <= '0';
236       load_n  <= '1';
237       shift_n <= '1';
238       s_in    <= '1';
239       sel_reg <= '1';                   -- from pseudo random generator
240       sel     <= "00";
241 
242       pass_v := true;
243       wait for 9*HALFCLOCKPERIOD;
244       wait until clk='0';
245 
246       -- load parallel zeros to shift registers
247       load_n <= '0';
248       wait until clk = '1';
249       wait until clk = '0';
250 
251       -- output shift registers data serially
252       for i in 63 downto 0 loop
253         serial_data_out(i) := s_out;
254         wait until clk = '0';
255       end loop;  -- i
256 
257       -- check that data is zeroed
258       match_v := serial_data_out = X"0000000000000000";
259       pass_v  := pass_v and match_v;
260       assert match_v report "Error on shift register reset" severity error;
261       --ieee.std_logic_textio.write(d_l,serial_data_out);  writeline(output, d_l);
262 
263     end check_shift_reg_rst;
264 ------------------------------------------------------------------------------
265     -- purpose: read the first 100 and 200 pseudo random generated data
266 ------------------------------------------------------------------------------
267     procedure read_rand is
268       variable serial_data_out : std_logic_vector(63 downto 0);
269     begin  -- read_rand
270 
271       -- init
272       rst_n   <= '0';
273       load_n  <= '1';
274       shift_n <= '1';
275       s_in    <= '1';
276       sel_reg <= '1';                   -- from pseudo random generator
277       sel     <= "00";
```

```
278
279         pass_v := true;
280         wait for 9*HALFCLOCKPERIOD;
281         wait until clk='0';
282
283
284         sel_reg <= '1';                    -- pseudo random data to shift_reg
285         load_n <= '0';                     -- ready to load the zeroed vector
286         wait until clk = '0';
287
288         rst_n <= '1';                      -- clear the reset
289         wait until clk = '0';
290         wait for 200*HALFCLOCKPERIOD;
291
292         load_n <= '1';                     -- switch to serial mode to extract data
293         shift_n <= '0';
294         -- output shift registers data serially
295         for i in 63 downto 0 loop
296           serial_data_out(i) := s_out;
297           wait until clk = '0';
298         end loop;  -- i
299
300         -- check data
301         match_v := serial_data_out = X"2F8D072F8D0BD0BD";
302         pass_v  := pass_v and match_v;
303         assert match_v report "Error on read random data " severity error;
304         --ieee.std_logic_textio.write(d_l,serial_data_out);  writeline(output, d_l);
305
306         load_n <= '0';                     -- reselect parallel input to shift_regs
307
308         wait for 72*HALFCLOCKPERIOD;
309
310         load_n <= '1';                     -- switch to serial mode to extract data
311         shift_n <= '0';
312         -- output shift registers data serially
313         for i in 63 downto 0 loop
314           serial_data_out(i) := s_out;
315           wait until clk = '0';
316         end loop;  -- i
317
318         -- check data
319         match_v := serial_data_out = X"7DF14972F14972F1";
320         pass_v  := pass_v and match_v;
321         assert match_v report "Error on read random data " severity error;
322         --ieee.std_logic_textio.write(d_l,serial_data_out);  writeline(output, d_l);
323       end read_rand;
324
325 ------------------------------------------------------------------------------
326       -- purpose: reset, read external data, multiply, output result
327 ------------------------------------------------------------------------------
328       procedure mult_ext_data (
329         -- the number corresponding to the tested multiplier
330         constant mult_number : in integer) is
331
332         variable serial_data_out : std_logic_vector(63 downto 0);
```

```
333         -- input multiplier data
334         constant DATA_IN : std_logic_vector(63 downto 0) := X"AC0E5F8EAC0E5F8E";
335         -- expected result = DATA_IN+DATA_IN(63 downto 32)*DATA_IN(31 downto 0)
336         constant EXPECTED_DATA_OUT : std_logic_vector(63 downto 0) :=
337           DATA_IN+(DATA_IN(63 downto 32)*DATA_IN(31 downto 0));
338
339       begin  -- mult_ext_data
340
341       -- init
342       rst_n   <= '0';
343       load_n  <= '1';
344       shift_n <= '1';
345       s_in    <= '1';
346       sel_reg <= '1'; -- from regs to mults
347       case mult_number is
348         when 0 => sel <= "00";
349         when 1 => sel <= "01";
350         when 2 => sel <= "10";
351         when 3 => sel <= "11";
352         when others => sel <= "XX";
353       end case;
354
355       pass_v := true;
356       wait for 9*HALFCLOCKPERIOD;
357       wait until clk='0';
358
359       sel_reg <= '1';                   -- pseudo random data to shift_reg
360       load_n <= '0';                    -- ready to load the zeroed vector
361       wait until clk = '0';
362
363       rst_n <= '1';                     -- clear the reset
364       wait until clk = '0';
365
366       shift_n <= '0';                   -- enter serial data
367       load_n  <= '1';
368       for i in 63 downto 0 loop
369         s_in <= DATA_IN(i);
370         wait until clk = '0';
371       end loop;  -- i
372     shift_n <= '1';
373     load_n  <= '1';
374
375     wait until clk = '0';
376     sel_reg <= '0';
377     wait until clk = '0';
378
379     -- delay if parallel 4 implementation is used
380     if mult_number = 1 or mult_number = 3 then
381       wait for 6*HALFCLOCKPERIOD;
382     end if;
383
384     load_n  <= '0';
385     wait until clk = '0';
386
387
```

```
388       load_n  <= '1';  -- switch to serial mode to extract data
389       shift_n <= '0';
390       -- output shift registers data serially
391       for i in 63 downto 0 loop
392         serial_data_out(i) := s_out;
393         wait until clk = '0';
394       end loop;  -- i
395
396       -- check data
397       match_v := serial_data_out = EXPECTED_DATA_OUT;
398       pass_v  := pass_v and match_v;
399       assert match_v report "Error on multiply external data " severity error;
400       --ieee.std_logic_textio.write(d_l,serial_data_out);  writeline(output, d_l);
401
402         end mult_ext_data;
403   ------------------------------------------------------------------------------
404       -- purpose: multiply and add pseudo random generated data
405   ------------------------------------------------------------------------------
406       procedure random_mac (
407         --  the number corresponding to the tested multiplier
408         constant mult_number : in integer) is
409
410           variable serial_data_out : std_logic_vector(63 downto 0);
411
412       begin  -- random_mac
413
414         -- init
415         rst_n   <= '0';
416         load_n  <= '0';                    -- store incoming data
417         shift_n <= '1';
418         s_in    <= '0';
419         sel_reg <= '1';                    -- from rand to regs
420         case mult_number is
421           when 0 => sel <= "00";
422           when 1 => sel <= "01";
423           when 2 => sel <= "10";
424           when 3 => sel <= "11";
425           when others => sel <= "XX";
426         end case;
427
428         pass_v := true;
429         wait for 9*HALFCLOCKPERIOD;
430         wait until clk='0';
431
432         sel_reg <= '0';                    -- pseudo random data to mult
433         load_n <= '0';                     -- ready to load sum to regs
434         rst_n <= '1';                      -- clear the reset
435         wait until clk = '0';
436
437         wait for 4*HALFCLOCKPERIOD;
438         -- delay if parallel 4 implementation is used
439         if mult_number = 1 or mult_number = 3 then
440           wait for 6*HALFCLOCKPERIOD;
441         end if;
442         wait for 1000*HALFCLOCKPERIOD;
```

```
443
444
445       load_n  <= '1';  -- switch to serial mode to extract data
446       shift_n <= '0';
447       -- output shift registers data serially
448       for i in 63 downto 0 loop
449         serial_data_out(i) := s_out;
450         wait until clk = '0';
451       end loop;  -- i
452
453       -- check data
454       match_v := serial_data_out = X"14C9836842DEF744";
455       pass_v  := pass_v and match_v;
456       assert match_v report "Error on multiply random data " severity error;
457       --ieee.std_logic_textio.write(d_l,serial_data_out);  writeline(output, d_l);
458
459       end random_mac;
460 ------------------------------------------------------------------------------
461       --test sequence
462 ------------------------------------------------------------------------------
463     begin
464       check_shift_reg_rst;
465       check_errors("Shift Registers Reset: ");
466       serial_read_write;
467       check_errors("Serial Read/Write: ");
468       read_rand;
469       check_errors("Read Rand: ");
470       mult_ext_data(0);
471       check_errors("Multiply external data on mult0: ");
472       mult_ext_data(1);
473       check_errors("Multiply external data on mult1: ");
474       mult_ext_data(2);
475       check_errors("Multiply external data on mult2: ");
476       mult_ext_data(3);
477       check_errors("Multiply external data on mult3: ");
478       random_mac(0);
479       check_errors("Add random multiplied data for mult0: ");
480       random_mac(1);
481       check_errors("Add random multiplied data for mult1: ");
482       random_mac(2);
483       check_errors("Add random multiplied data for mult2: ");
484       random_mac(3);
485       check_errors("Add random multiplied data for mult3: ");
486
487       global_check;
488       wait;
489     end process;
490
491 end top_func_test;
492
493 ------------------------------------------------------------------------------
494
495 configuration top_tb_top_func_test_cfg of top_tb is
496   for top_func_test
497   end for;
```

```
498 end top_tb_top_func_test_cfg;
499
500 ------------------------------------------------------------------------------
```

Appendix B

Synopsys compilation scripts

B.1 compile_top.tcl

```
1  ########################
2  ## Global variables ##
3  ########################
4  set BIN ./bin/
5  set DB ./db/
6  set PAR ./par/src/
7  set GATE ./gate/
8  set WORK ./work/
9  set SRC ./vhdl/
10 set reports_path ./reports/
11
12 set design_name top_stm090
13
14 #ignore case to avoid problem on activity annotation
15 set find_ignore_case true
16 set suppress_errors "VHDL-2285 OPT-150 TIM-111 TIM-112"
17 #remove the limit for high fanout nets
18 set high_fanout_net_threshold 0
19
20 #define the working path
21 define_design_lib work -path $WORK
22
23 ########################
24 ## Bus name variables ##
25 ########################
26 set bus_naming_style %s(%d)
27
28 ##############################
29 ## Remove previous designs ##
30 ##############################
31 #remove_constraint -all
32 #remove_design -all
33
34 ########################
35 ## Read design ##
```

```
36 ########################
37 source read_vhdl.tcl
38
39 #link design with library (i.e. load required libraries)
40 link
41
42 #Uniquify the multilpiers blocks
43 set uniquify_naming_style %s_%d
44
45 #rename the 4 main multipliers
46 uniquify -cell {mult_0 mult_1 mult_2 mult_3} -base_name multd
47
48 #rename the remaining design
49 uniquify
50
51 ################################
52 ## Create clock MHz  62.5MHz ##
53 ################################
54
55 #main clock
56 create_clock clk -period 16
57
58 #generated clock
59 create_generated_clock -source clk -name dclk0 -divide_by 1 mult_0/clk
60 create_generated_clock -source clk -name dclk1 -divide_by 1 mult_1/clk
61 create_generated_clock -source clk -name dclk2 -divide_by 1 mult_2/clk
62 create_generated_clock -source clk -name dclk3 -divide_by 1 mult_3/clk
63
64 #allow maximum delay at input and output
65 set_input_delay 0  -clock clk [all_inputs]
66 set_output_delay 0 -clock clk [all_outputs]
67
68 #next lines are there to avoid TIM-111 warning
69 set_input_delay 0  -clock dclk0 mult_0/clk
70 set_input_delay 0  -clock dclk1 mult_1/clk
71 set_input_delay 0  -clock dclk2 mult_2/clk
72 set_input_delay 0  -clock dclk3 mult_3/clk
73
74 #Set_propagated_clock automaticaly set the correct
75 #set_clock_latency value for the generated clocks
76 set_propagated_clock [all_clocks]
77
78 #report_clock -skew
79
80 ########################
81 ## Set Loads          ##
82 ########################
83 set_drive [drive_of CORE90GPSVT_NomLeak.db:CORE90GPSVT/IVSVTX1/Z] [all_inputs]
84 #2.004893
85 set_load [load_of  CORE90GPSVT_NomLeak.db:CORE90GPSVT/IVSVTX1/A] [all_outputs]
86 #0.002090
87
88 #put an high load on the s_out because it
89 #will drive a analogic pad
90 set_load 10 s_out
```

```
91
92  ########################
93  ## Design Constraints##
94  ########################
95  set_max_area 0
96
97  ########################
98  ## Compile Design     ##
99  ########################
100
101 # Write unmapped top
102 current_design top
103 write -hierarchy -output ${DB}unmapped_top.db
104
105 # characterize the 4 multipliers
106 set target_library CORE90GPSVT_NomLeak.db
107 characterize -constraint {mult_0 mult_1 mult_2 mult_3}
108
109 # compile mult_0 and mult_1 with SVT
110 set target_library CORE90GPSVT_NomLeak.db
111 current_design multd_0
112 compile
113 current_design multd_1
114 compile
115
116 # compile mult_2 and mult_3 with LVT
117 set target_library CORE90GPLVT_NomLeak.db
118 current_design multd_2
119 compile
120 current_design multd_3
121 compile
122
123 #set dont touch to compile mutlipliers
124 current_design top
125 set_dont_touch {mult_0 mult_1 mult_2 mult_3 ring_svt_1 ring_lvt_1}
126
127 # the rest of the design will be compiled with
128 # the SVT technology
129 set target_library CORE90GPSVT_NomLeak.db
130 compile -map_effort high
131
132 #show which DW implementation has been selected
133 report_resources -hier  > ${reports_path}${design_name}.syn_rprh
134
135 #remove unconnected ports in DW designs
136 set_dont_touch {mult_0 mult_1 mult_2 mult_3} false
137 remove_unconnected_ports [get_cells -hier *]
138 remove_unconnected_ports -blast_buses [get_cells -hier *]
139 set_dont_touch {mult_0 mult_1 mult_2 mult_3} true
140
141 ##########################
142 ## Fix hold violations ##
143 ##########################
144 set_fix_hold [all_clocks]
145
```

```
146 #recompile top only
147 compile -inc
148
149 ######################
150 ##Write Mapped design##
151 ######################
152 change_names -rules vhdl -hierarchy
153 write -hierarchy -format vhdl -output ${GATE}top.vhd
154 write_sdf ${GATE}top.sdf
155
156 ######################
157 ## Annotate Activtiy ##
158 ######################
159 #sh cp msim/modelsim.ini ./modelsim.ini
160 sh cp ${SRC}top_tb.vhd ${GATE}top_tb.vhd
161 sh vsim -c -do ${BIN}power_sdf.do
162 read_saif -unit ns -scale 1 -instance top_tb/dut -input ${GATE}back.saif
163
164 ##############################################
165 ##   Save reports in the defined directory   ##
166 ##############################################
167 report_area                 > ${reports_path}${design_name}.syn_rpa
168 check_design                > ${reports_path}${design_name}.syn_rpd
169 report_timing               > ${reports_path}${design_name}.syn_rpt
170 report_hierarchy            > ${reports_path}${design_name}.syn_rph
171 report_resources            > ${reports_path}${design_name}.syn_rpr
172 report_cell                 > ${reports_path}${design_name}.syn_rpc
173 report_power -net -cell -flat -include_input_nets  > ${reports_path}${design_name}
        .syn_rpp
174 report_saif -flat           > ${reports_path}${design_name}.syn_rps
175 report_constraint           > ${reports_path}${design_name}.syn_rpn
176 report_reference -nosplit   > ${reports_path}${design_name}.syn_rpf
177 report_clock -skew          > ${reports_path}${design_name}.syn_rpk
178 report_power -include_input_nets -hier -hier_level 1 > ${reports_path}${design_name}
        .syn_rpph
179
180 ######################
181 ##    Save design    ##
182 ######################
183 change_names -rules verilog -hierarchy
184 set bus_naming_style %s\[%d\]
185 write -hierarchy -output ${DB}top_gate.db
186 write -hierarchy -format verilog -output ${PAR}top_gate.v
187 write_sdf ${PAR}top_gate.sdf
188 write_sdc ${PAR}top_gate.sdc
189
190 quit
```

B.2 read_vhdl.tcl

```
1 analyze -f vhdl vhdl/ring_svt.vhd
2 analyze -f vhdl vhdl/ring_lvt.vhd
3 analyze -f vhdl vhdl/RCA_generic_arch.vhd
4 analyze -f vhdl vhdl/mult.vhd
```

```
analyze -f vhdl vhdl/mult_par4.vhd
analyze -f vhdl vhdl/data_gen.vhd
analyze -f vhdl vhdl/top.vhd
elaborate top -update
```

B.3 power_sdf.do

```
##############################################################
# Script to computate the switching activity with ModelSim #
# Schuster Christian, June 2003, IMT Neuchatel             #
##############################################################
#execute this script with vsim -c -do power_sdf.do
#needed files are: top.sdf, top.vhd, top_tb.vhd

# Testbench path and file names
set work_dir /scratch/schuster/stm090_svt_lvt
set testbench top_tb
set bench_path $testbench/dut
set dir gate
set sdffname $dir/top.sdf

# Time and simulation settings
set time_scale ps
set backsaif_basetime 1E-12

set init_time 19592000
#19592ns
set evaluation_time 36704000
#36704ns

#compile design+testbench
vcom -93 $dir/top.vhd -work $work_dir
vcom -93 $dir/top_tb.vhd -work $work_dir

# Use the same path separator as Synopsys SAIF file
set PathSeparator /
set DatasetSeparator :

vsim +notimingchecks -sdftyp $bench_path=$sdffname -foreign "dpfli_init /synopsys/
    v2004.06/auxx/syn/power/dpfli/lib-sparcOS5/dpfli.so" -lib $work_dir -t
    $time_scale $testbench
#+notimingchecks is used to avoid unreal problems on sdf annotation and verilog model


#initialize ring oscillators
force top_tb/dut/ring_svt_1/z_port 0 0 -c 16
force top_tb/dut/ring_lvt_1/z_port 0 0 -c 16

# Select toggle region
set_toggle_region  $bench_path

# Init the circuit
run $init_time

```

```
46 # Start switching annotation
47 toggle_start
48
49 # Execute testbench
50 run $evaluation_time
51
52 # Stop switching annotation
53 toggle_stop
54
55 # Write back annotation SAIF
56 toggle_report $dir/back.saif $backsaif_basetime $bench_path
57
58 quit
```

Appendix C

SoC Encounter P&R scripts

C.1 main.tcl

```
1  ########################################
2  ## Main file for ENCOUNTER SOC4.1      ##
3  ## CSch, July 2006, version 1.1        ##
4  ########################################
5  # Required Files:
6  # Scripts:
7  # - top.conf
8  # - IO_Filler.tcl
9  # - create_global_net.tcl
10 # - pwr.tcl
11 # - do_power_domains
12 # - followPin.tcl
13 # - top.ctstch
14 # - output_nets.tcl
15 # - place_output_bufs.tcl
16 # - fix_drc_errors.tcl
17 # Data:
18 # - ioplace.io
19 # - LEF/IO90GPHVT_BASIC_50A_7M2T_PGC.lef (ALL external layers of COREVDD1V0 pin need
       the line "CLASS CORE ;" in order to be be routed by sroute, diff file present)
20 # - LEF/IO90GPHVT_3V3_50A_7M2T_PGC.lef (diff file present)
21 # Src:
22 # - top_io.v (from cat src/top_gate.v data/io_wrapper.v > src/top_io.v)
23 # - top_gate.sdc (change get_pins -> get_pins -hierarchy)
24
25 Puts "###################################"
26 Puts "###"
27 Puts "### Load Design                   "
28 Puts "###"
29 Puts "###################################"
30
31 ### uniquify the netlist (shell to execute before an encounter session)
32 ### uniquifyNetlist -top
33 #In my case netlist was already unique!
34
```

```
35 set CMOS090GP_DIR /designkit/cmos090_50a
36 set scpts scripts/
37 set data  data/
38
39
40 setRCFactor -cap 1.1
41
42 #set the size of the smallest displayed module -> display all
43 setPreference MinFPModuleSize 1
44
45 #load the design + io + corners
46 loadConfig ${scpts}top.conf
47
48 #load footprints used for timining driven analysis
49 loadfootprint -infile ${CMOS090GP_DIR}/SocEncounter_cmos090gp_2.2/cmos090gp_50a.cfp
50 setInvFootPrint IVSVTX1
51 setBufFootPrint BFSVTX1
52 #setDelayFootPrint DLY1SVTX2
53
54 Puts "############################"
55 Puts "###"
56 Puts "### Create Floorplan      "
57 Puts "###"
58 Puts "############################"
59
60 ### define floorplan
61 #floorPlan -r 1 0.7 40 40 40 40
62 #Fixed dimension allow io_corners to be aligned with the 0.56um grid
63 floorPlan -s 800.28 800.28 50.08 50.08 50.08 50.08
64
65 ### Add IO filler
66 source ${scpts}IO_Filler.tcl
67
68 Puts "################################"
69 Puts "###"
70 Puts "### Create PowerDomains and "
71 Puts "### Place Block(s)"
72 Puts "###"
73 Puts "################################"
74
75 #create the 5 separated power domains
76 source ${scpts}do_power_domains.tcl
77
78 #place ioref_comp instance needed for the 3V3 IOs
79 placeInstance ioref_comp 732.88 204.04 R180
80 addHaloToBlock 31.5 20 20 30 -allBlock
81
82 #connects all global nets
83 source ${scpts}create_global_net.tcl
84
85 Puts "################################"
86 Puts "###"
87 Puts "### Create power stripes "
88 Puts "###"
89 Puts "################################"
```

```
90
91  ### add power ring + stripes
92  source ${scpts}pwr.tcl
93
94  ### std-cell follow pin
95  source ${scpts}followPin.tcl
96
97  # save floor-plan
98  saveFPlan ./fplan.fp
99
100 # check floor-plan
101 verifyGeometry
102
103 saveDesign ./top.fp.enc
104 #source ./top.fp.enc
105
106 Puts "####################"
107 Puts "###"
108 Puts "### Place Design ..."
109 Puts "###"
110 Puts "####################"
111
112 #exec mkdir Timing
113 source ${scpts}/place_output_bufs.tcl
114
115 amoebaPlace -timingdriven \
116             -doCongOpt \
117             -highEffort \
118             -ignoreScan \
119             -ignoreSpare \
120             -QA \
121             -slack init_virtual.slk
122
123 saveDesign ./top.place.enc
124 #source ./top.place.enc
125
126 checkPlace
127
128 buildTimingGraph
129 timeDesign -preCTS -outDir ./Timing/PLACE.timing
130
131 Puts "####################"
132 Puts "###"
133 Puts "### Optimization..."
134 Puts "###"
135 Puts "####################"
136
137 setOptMode  -highEffort \
138             -fixFanoutLoad \
139             -maxDensity 0.8 \
140             -reclaimArea \
141             -setupTargetSlack 0.0 \
142             -holdTargetSlack 0.0
143
144 optDesign -preCTS -setup -drv
```

```
145
146 saveDesign ./top.IPO.enc
147 # source ./top.IPO.enc
148
149 timeDesign -preCTS -outDir ./Timing/IPO.timing
150
151 Puts "###############"
152 Puts "###"
153 Puts "### Run CTS..."
154 Puts "###"
155 Puts "###############"
156
157 #clock with different libraries (SVT/LVT) depending on power domains
158 setCTSMode -fence -MSMV
159 #specify clocktree file
160 specifyClockTree -clkfile ${scpts}top.ctstch
161 #create report directory
162 createSaveDir top_cts
163
164 #do clock tree synthesis
165 ckSynthesis -rguide top_cts/top_cts.guide -report top_cts/top_cts.ctsrpt
166 saveClockNets -output top_cts/top_cts.ctsntf
167 saveNetlist top_cts/top_cts.v
168 savePlace top_cts/top_cts.place
169
170 saveDesign ./top.POST_CTS.enc
171 #source ./top.POST_CTS.enc
172
173 setAnalysisMode -clockTree
174 buildTimingGraph
175 timeDesign -postCTS -outDir ./Timing/POST_CTS.timing
176
177 Puts "##############################"
178 Puts "###"
179 Puts "### Optimization post CTS..."
180 Puts "###"
181 Puts "##############################"
182
183 setOptMode  -highEffort \
184             -fixFanoutLoad \
185             -maxDensity 0.8 \
186             -reclaimArea \
187             -setupTargetSlack 0.0 \
188             -holdTargetSlack 0.0
189
190 optDesign -postCTS
191
192 saveDesign ./top.POST_CTS_IPO.enc
193 # source ./top.POST_CTS_IPO.enc
194
195 timeDesign -postCTS -outDir ./Timing/POST_CTS_IPO.timing
196
197 Puts "###################"
198 Puts "###"
199 Puts "### Nanoroute.... "
```

```
200 Puts "###"
201 Puts "###################"
202
203 # Filler Cell between std-cells
204 addFiller -cell FILLERCELL64 FILLERCELL32 FILLERCELL16 FILLERCELL8 FILLERCELL4
        FILLERCELL2 FILLERCELL1 -prefix FILLER -powerDomain PDCORE
205 addFiller -cell FILLERCELL64 FILLERCELL32 FILLERCELL16 FILLERCELL8 FILLERCELL4
        FILLERCELL2 FILLERCELL1 -prefix FILLER -powerDomain PD0
206 addFiller -cell FILLERCELL64 FILLERCELL32 FILLERCELL16 FILLERCELL8 FILLERCELL4
        FILLERCELL2 FILLERCELL1 -prefix FILLER -powerDomain PD1
207 addFiller -cell FILLERCELL64 FILLERCELL32 FILLERCELL16 FILLERCELL8 FILLERCELL4
        FILLERCELL2 FILLERCELL1 -prefix FILLER -powerDomain PD2
208 addFiller -cell FILLERCELL64 FILLERCELL32 FILLERCELL16 FILLERCELL8 FILLERCELL4
        FILLERCELL2 FILLERCELL1 -prefix FILLER -powerDomain PD3
209
210 # connect all new std-cell instances to vdd/gnd
211 source ${scpts}create_global_net.tcl
212
213
214 ########################
215 ## Route clocks first ##
216 ########################
217
218 setAttribute -net @clock -weight 5 -avoid_detour true -bottom_preferred_routing_layer
        4 -preferred_extra_space 1
219 selectNet -allDefClock
220 setNanoRouteMode -quiet routeWithTimingDriven false
221 setNanoRouteMode -quiet envNumberProcessor 1
222 setNanoRouteMode -quiet route_selected_net_only true
223
224 globalDetailRoute
225
226 saveDesign ./top.POST_CLK_ROUTE.enc
227 #source ./top.POST_CLK_ROUTE.enc
228
229 #allow wide routing for s_out Z_lvt Z_svt
230 convertNetToSNet  -nets {s_out Z_lvt Z_svt}
231 source ${scpts}/output_nets.tcl
232
233 ####################
234 ## Route All Nets ##
235 ####################
236
237 setNanoRouteMode -quiet routeFixPrewire true
238 setNanoRouteMode -quiet route_selected_net_only false
239 setNanoRouteMode -quiet routeWithTimingDriven false
240 setNanoRouteMode -quiet routeTdrEffort 1
241 setNanoRouteMode -quiet drouteFixAntenna true
242 setNanoRouteMode -quiet routeWithSiDriven true
243 setNanoRouteMode -quiet routeSiLengthLimit 200
244 setNanoRouteMode -quiet routeSiEffort normal
245
246 globalDetailRoute
247
248 #fix errors found with calibre DRC check
```

```
249 source ${scpts}/fix_drc_errors.tcl
250
251 saveDesign ./top.POST_ROUTE.enc
252 #source ./top.POST_ROUTE.enc
253
254 ##########################
255 ## Check for violations ##
256 ##########################
257
258 clearDrc
259 verifygeometry -allowDiffCellViols
260 verifyConnectivity -type regular -error 1000 -warning 50
261 verifyProcessAntenna
262
263 reportLeakagePower
264
265 Puts "#################################"
266 Puts "###"
267 Puts "### Create abstract views : verilog / LEF / DEF / GDS /SDF    ..."
268 Puts "###"
269 Puts "#################################"
270
271 #exec mkdir RESULTS
272
273 ########
274 ### verilog
275 ########
276 saveNetlist ./RESULTS/top.v
277
278 ########
279 ### lef
280 ########
281 lefOut ./RESULTS/top.lef -stripePin -PGpinLayers 6 7
282
283 #######
284 ### def
285 #######
286 defOut -floorplan -routing ./RESULTS/top.def
287
288 #######
289 ### gds
290 #######
291 streamOut ./RESULTS/top_with_io.gds \
292             -mapFile ${CMOS090GP_DIR}/SocEncounter_cmos090gp_2.2/gds2_cmos90.map \
293             -libName DesignLib \
294             -structureName top_with_io \
295             -stripes 1 \
296             -units 2000 \
297             -mode ALL
298
299 #######
300 ### sdf
301 #######
302 setExtractRCMode -detail
303 extractRC
```

```
304 delayCal -sdf ./RESULTS/top.sdf
```

C.2 top.conf

```
1  ################################################
2  #                                              #
3  # Input configuration file                     #
4  #                                              #
5  ################################################
6
7  #set designkit path
8  set CMOS090GP_DIR /designkit/cmos090_50a
9
10 global rda_Input
11
12 #set cwd ./work
13
14 set rda_Input(import_mode) {-treatUndefinedCellAsBbox 0 -verticalRow 0
15   -keepEmptyModule 1 }
16 set rda_Input(ui_netlist) "src/top_io.v"
17
18 set rda_Input(ui_netlisttype) {Verilog}
19 set rda_Input(ui_ilmlist) {}
20 set rda_Input(ui_settop) {1}
21 set rda_Input(ui_topcell) {top_with_io}
22 set rda_Input(ui_celllib) {}
23 set rda_Input(ui_iolib) {}
24 set rda_Input(ui_areaiolib) {}
25 set rda_Input(ui_blklib) {}
26 set rda_Input(ui_kboxlib) {}
27 set rda_Input(ui_gds_file) {}
28 set rda_Input(ui_timelib,min) "
29 ${CMOS090GP_DIR}/CORE90GPLVT_SNPS-AVT_2.1/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORE90GPLVT_Best.lib
30 ${CMOS090GP_DIR}/CORE90GPSVT_SNPS-AVT_2.1/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORE90GPSVT_Best.lib
31 ${CMOS090GP_DIR}/CORX90GPLVT_SNPS-AVT_4.2/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORX90GPLVT_Best.lib
32 ${CMOS090GP_DIR}/CORX90GPSVT_SNPS-AVT_4.2/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORX90GPSVT_Best.lib
33 ${CMOS090GP_DIR}/CLOCK90GPLVT_SNPS-AVT_2.1/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB
       /CLOCK90GPLVT_Best.lib
34 ${CMOS090GP_DIR}/CLOCK90GPSVT_SNPS-AVT_2.1/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB
       /CLOCK90GPSVT_Best.lib
35 ${CMOS090GP_DIR}/PR90M7_SNPS-AVT_3.0/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       PR90M7_Best.lib
36 ${CMOS090GP_DIR}/IO90GPHVT_3V3_50A_7M2T_SNPS-AVT_4.0/SIGNOFF/
       bc_1.10V_m40C_wc_0.90V_125C/PT_LIB/IO90GPHVT_3V3_50A_7M2T_Best.lib
37 ${CMOS090GP_DIR}/IO90GPHVT_BASIC_50A_7M2T_SNPS-AVT_4.0/SIGNOFF/
       bc_1.10V_m40C_wc_0.90V_125C/PT_LIB/IO90GPHVT_BASIC_50A_7M2T_Best.lib
38 ${CMOS090GP_DIR}/IO90GPHVT_REF_COMPENSATION_3V3_50A_SNPS-AVT_4.0/SIGNOFF/
       bc_1.10V_m40C_wc_0.90V_125C/PT_LIB/IO90GPHVT_REF_COMPENSATION_3V3_50A_Best.lib"
39
40 set rda_Input(ui_timelib,max) "
```

```
41 ${CMOS090GP_DIR}/CORE90GPLVT_SNPS-AVT_2.1/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORE90GPLVT_Worst.lib
42 ${CMOS090GP_DIR}/CORE90GPSVT_SNPS-AVT_2.1/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORE90GPSVT_Worst.lib
43 ${CMOS090GP_DIR}/CORX90GPLVT_SNPS-AVT_4.2/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORX90GPLVT_Worst.lib
44 ${CMOS090GP_DIR}/CORX90GPSVT_SNPS-AVT_4.2/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       CORX90GPSVT_Worst.lib
45 ${CMOS090GP_DIR}/CLOCK90GPHVT_SNPS-AVT_2.1.a/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/
       PT_LIB/CLOCK90GPHVT_Worst.lib
46 ${CMOS090GP_DIR}/CLOCK90GPLVT_SNPS-AVT_2.1/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB
       /CLOCK90GPLVT_Worst.lib
47 ${CMOS090GP_DIR}/PR90M7_SNPS-AVT_3.0/SIGNOFF/bc_1.10V_m40C_wc_0.90V_105C/PT_LIB/
       PR90M7_Worst.lib
48 ${CMOS090GP_DIR}/IO90GPHVT_3V3_50A_7M2T_SNPS-AVT_4.0/SIGNOFF/
       bc_1.10V_m40C_wc_0.90V_125C/PT_LIB/IO90GPHVT_3V3_50A_7M2T_Worst.lib
49 ${CMOS090GP_DIR}/IO90GPHVT_BASIC_50A_7M2T_SNPS-AVT_4.0/SIGNOFF/
       bc_1.10V_m40C_wc_0.90V_125C/PT_LIB/IO90GPHVT_BASIC_50A_7M2T_Worst.lib
50 ${CMOS090GP_DIR}/IO90GPHVT_REF_COMPENSATION_3V3_50A_SNPS-AVT_4.0/SIGNOFF/
       bc_1.10V_m40C_wc_0.90V_125C/PT_LIB/IO90GPHVT_REF_COMPENSATION_3V3_50A_Worst.lib"
51
52 set rda_Input(ui_timelib) {}
53 set rda_Input(ui_smodDef) {}
54 set rda_Input(ui_smodData) {}
55 set rda_Input(ui_dpath) {}
56 set rda_Input(ui_tech_file) {}
57 set rda_Input(ui_io_file) {data/ioplace.io}
58 set rda_Input(ui_timingcon_file) {src/top_gate.sdc }
59 set rda_Input(ui_latency_file) {}
60 set rda_Input(ui_scheduling_file) {}
61 set rda_Input(ui_buf_footprint) {}
62 set rda_Input(ui_delay_footprint) {}
63 set rda_Input(ui_inv_footprint) {}
64 set rda_Input(ui_leffile) "
65 ${CMOS090GP_DIR}/SocEncounter_cmos090gp_2.2/cmos090gp_soc.lef
66 ${CMOS090GP_DIR}/CORE90GPLVT_SNPS-AVT_2.1/SIGNOFF/common/LEF/CORE90GPLVT_ANT.lef
67 ${CMOS090GP_DIR}/CORE90GPSVT_SNPS-AVT_2.1/SIGNOFF/common/LEF/CORE90GPSVT_ANT.lef
68 ${CMOS090GP_DIR}/CORX90GPLVT_SNPS-AVT_4.2/SIGNOFF/common/LEF/CORX90GPLVT_ANT.lef
69 ${CMOS090GP_DIR}/CORX90GPSVT_SNPS-AVT_4.2/SIGNOFF/common/LEF/CORX90GPSVT_ANT.lef
70 ${CMOS090GP_DIR}/CLOCK90GPLVT_SNPS-AVT_2.1/SIGNOFF/common/LEF/CLOCK90GPLVT_ANT.lef
71 ${CMOS090GP_DIR}/CLOCK90GPSVT_SNPS-AVT_2.1/SIGNOFF/common/LEF/CLOCK90GPSVT_ANT.lef
72 ${CMOS090GP_DIR}/PR90M7_SNPS-AVT_3.0/SIGNOFF/common/LEF/PR90M7_ANT.lef
73 data/LEF/IO90GPHVT_3V3_50A_7M2T_PGC.lef
74 data/LEF/IO90GPHVT_BASIC_50A_7M2T_PGC.lef
75 ${CMOS090GP_DIR}/IO90GPHVT_REF_COMPENSATION_3V3_50A_SNPS-AVT_4.0/SIGNOFF/common/LEF/
       IO90GPHVT_REF_COMPENSATION_3V3_50A.lef"
76
77
78 set rda_Input(ui_core_cntl) {aspect}
79 set rda_Input(ui_aspect_ratio) {1.0}
80 set rda_Input(ui_core_util) {0.7}
81 set rda_Input(ui_core_height) {}
82 set rda_Input(ui_core_width) {}
83 set rda_Input(ui_core_to_left) {}
84 set rda_Input(ui_core_to_right) {}
```

```
85  set rda_Input(ui_core_to_top) {}
86  set rda_Input(ui_core_to_bottom) {}
87  set rda_Input(ui_max_io_height) {0}
88  set rda_Input(ui_row_height) {3.92}
89  set rda_Input(ui_isHorTrackHalfPitch) {0}
90  set rda_Input(ui_isVerTrackHalfPitch) {1}
91  set rda_Input(ui_ioOri) {R0}
92  set rda_Input(ui_isOrigCenter) {0}
93  set rda_Input(ui_exc_net) {}
94  set rda_Input(ui_delay_limit) {1000}
95  set rda_Input(ui_net_delay) {1000.0ps}
96  set rda_Input(ui_net_load) {0.5pf}
97  set rda_Input(ui_in_tran_delay) {120.0ps}
98  set rda_Input(ui_captbl_file) {}
99  set rda_Input(ui_defcap_scale) {1.0}
100 set rda_Input(ui_detcap_scale) {1.0}
101 set rda_Input(ui_xcap_scale) {1.0}
102 set rda_Input(ui_res_scale) {1.0}
103 set rda_Input(ui_shr_scale) {1.0}
104 set rda_Input(ui_time_unit) {none}
105 set rda_Input(ui_cap_unit) {}
106 set rda_Input(ui_oa_reflib) {}
107 set rda_Input(ui_oa_abstractname) {}
108 set rda_Input(ui_sigstormlib) {}
109 set rda_Input(ui_cdb_file) {}
110 set rda_Input(ui_echo_file) {}
111 set rda_Input(ui_xilm_file) {}
112 set rda_Input(ui_qxtech_file) {}
113 set rda_Input(ui_qxlib_file) {}
114 set rda_Input(ui_qxconf_file) {}
115 set rda_Input(ui_pwrnet) {vdd vdde vdd0 vdd1 vdd2 vdd3 vddcore}
116 set rda_Input(ui_gndnet) {gnd gnde \
117 CLKSLEEP TQ DIGA DIGB KOFF REFA REFB REFC REFD REFE REFF \
118 A13SRC A12SRC A11SRC A10SRC A9SRC A8SRC A7SRC A6SRC A5SRC A4SRC A3SRC A2SRC A1SRC
        A0SRC \
119 IO_CLKSLEEP IO_TQ IO_DIGA IO_DIGB IO_KOFF IO_REFA IO_REFB IO_REFC IO_REFD IO_REFE
        IO_REFF \
120 IO_A13SRC IO_A12SRC IO_A11SRC IO_A10SRC IO_A9SRC IO_A8SRC IO_A7SRC IO_A6SRC IO_A5SRC
        IO_A4SRC IO_A3SRC IO_A2SRC IO_A1SRC IO_A0SRC \
121 }
122 set rda_Input(flip_first) {1}
123 set rda_Input(double_back) {1}
124 set rda_Input(assign_buffer) {1}
125 set rda_Input(ui_gen_footprint) {0}
```

C.3 IO_Filler.tcl

```
1 #define user grid
2 setPreference ConstraintUserXGrid 0.56
3 setPreference ConstraintUserYGrid 0.56
4 snapFPlanIO -usergrid
5 redraw
6
7 #add IO filler from the bigger to the smaller
```

```
addIoFiller -cell IOFILLER64_LIN -prefix io_fillperi
addIoFiller -cell IOFILLER32_LIN -prefix io_fillperi
addIoFiller -cell IOFILLER16_LIN -prefix io_fillperi
addIoFiller -cell IOFILLER8_LIN -prefix io_fillperi
addIoFiller -cell IOFILLER4_LIN -prefix io_fillperi
addIoFiller -cell IOFILLER2_LIN -prefix io_fillperi
addIoFiller -cell IOFILLER1_LIN -prefix io_fillperi
redraw
```

C.4 do_power_domains.tcl

```
#create power domains
deletePowerDomain
createPowerDomain PD0 -timinglibs "CORE90GPSVT"
createPowerDomain PD1 -timinglibs "CORE90GPSVT"
createPowerDomain PD2 -timinglibs "CORE90GPLVT"
createPowerDomain PD3 -timinglibs "CORE90GPLVT"
createPowerDomain PDCORE -timinglibs "CORE90GPSVT"


#include instances
modifyPowerDomainMember PD0 -instance core/mult_0 -power (vdd0:vdd) -ground (gnd:gnd)
modifyPowerDomainMember PD0 -instance ioco_vddioco_0 -power (vdd0:VDDCORE1V0) -move

modifyPowerDomainMember PD1 -instance core/mult_1 -power (vdd1:vdd) -ground (gnd:gnd)
modifyPowerDomainMember PD1 -instance ioco_vddioco_1 -power (vdd1:VDDCORE1V0) -move

modifyPowerDomainMember PD2 -instance core/mult_2 -power (vdd2:vdd) -ground (gnd:gnd)
modifyPowerDomainMember PD2 -instance ioco_vddioco_2 -power (vdd2:VDDCORE1V0) -move

modifyPowerDomainMember PD3 -instance core/mult_3 -power (vdd3:vdd) -ground (gnd:gnd)
modifyPowerDomainMember PD3 -instance ioco_vddioco_3 -power (vdd3:VDDCORE1V0) -move

modifyPowerDomainMember PDCORE -instance ioco_vddioco_core -power (vddcore:VDDCORE1V0
    ) -move
modifyPowerDomainMember PDCORE -instance * -power (vddcore:vdd) -ground (gnd:gnd)

#resize it
modifyPowerDomainAttr PDCORE -box  194.04 381.76 691.76 587.76  -rsExts 10 10 40 10
    -minGaps 10 10 10 10
createPowerDomainCut               640.88 469.28 691.76 597.76
modifyPowerDomainAttr PD0 -box     194.04 194.04 433.76 361.76  -rsExts 10 10 10 10
    -minGaps 10 10 10 10
modifyPowerDomainAttr PD1 -box     194.04 606.56 640.88 994.32  -rsExts 10 10 10 10
    -minGaps 10 10 10 10
modifyPowerDomainAttr PD2 -box     452.26 194.04 691.76 361.76  -rsExts 10 10 10 10
    -minGaps 10 10 10 10
modifyPowerDomainAttr PD3 -box     659.48 490.44 994.32 994.32 -rsExts 10 10 10 10
    -minGaps 10 10 10 10
```

C.5 create_global_net.tcl

```
Puts "#################################"
```

```
2  Puts "###"
3  Puts "### Power declaration for std-cells and IO PADs"
4  Puts "###"
5  Puts "#################################"
6
7
8  ###
9  ### WARNING : All the global nets should be declared first in the ".conf" file
10 ###
11
12
13 ###
14 ### first, declare vdd/gnd pin's for all std-cells
15 ###
16
17 globalNetConnect vdd -type pgpin -pin {vdd } -inst * -module {}
18 globalNetConnect gnd -type pgpin -pin {gnd } -inst * -module {}
19
20 ### declare 0/1 vhdl/verilog constants to be on vdd/gnd supplys
21 globalNetConnect vdd -type tiehi -module {}
22 globalNetConnect gnd -type tielo -module {}
23
24 ###
25 ### IO pads
26 ### - All the instance names for the IO pads must have the "io_" prefix
27 ###
28
29
30 ### IO's & core supply
31 globalNetConnect vdd -type pgpin -pin {vdd } -inst io* -module {} -override
32 globalNetConnect gnd -type pgpin -pin {gnd } -inst io* -module {} -override
33
34 ### remaining IOs pins
35  globalNetConnect gnde -type pgpin -pin {gnde } -inst io* -module {} -override
36  globalNetConnect vdde -type pgpin -pin {vdde } -inst io* -module {} -override
37
38  globalNetConnect IO_CLKSLEEP -type pgpin -pin {CLKSLEEP } -inst io* -module {}
        -override
39  globalNetConnect IO_TQ -type pgpin -pin {TQ } -inst io* -module {}  -override
40  globalNetConnect IO_DIGA -type pgpin -pin {DIGA } -inst io* -module {}  -override
41
42  globalNetConnect IO_DIGB -type pgpin -pin {DIGB } -inst io* -module {}  -override
43  globalNetConnect IO_KOFF -type pgpin -pin {KOFF } -inst io* -module {}  -override
44
45  globalNetConnect IO_REFA -type pgpin -pin {REFA } -inst io* -module {}  -override
46  globalNetConnect IO_REFB -type pgpin -pin {REFB } -inst io* -module {}  -override
47  globalNetConnect IO_REFC -type pgpin -pin {REFC } -inst io* -module {}  -override
48  globalNetConnect IO_REFD -type pgpin -pin {REFD } -inst io* -module {}  -override
49  globalNetConnect IO_REFE -type pgpin -pin {REFE } -inst io* -module {}  -override
50  globalNetConnect IO_REFF -type pgpin -pin {REFF } -inst io* -module {}  -override
51
52  globalNetConnect IO_A0SRC -type pgpin -pin {A0SRC } -inst io* -module {}  -override
53  globalNetConnect IO_A1SRC -type pgpin -pin {A1SRC } -inst io* -module {}  -override
54  globalNetConnect IO_A2SRC -type pgpin -pin {A2SRC } -inst io* -module {}  -override
55  globalNetConnect IO_A3SRC -type pgpin -pin {A3SRC } -inst io* -module {}  -override
```

```
56  globalNetConnect IO_A4SRC -type pgpin -pin {A4SRC } -inst io* -module {}  -override
57  globalNetConnect IO_A5SRC -type pgpin -pin {A5SRC } -inst io* -module {}  -override
58  globalNetConnect IO_A6SRC -type pgpin -pin {A6SRC } -inst io* -module {} -override
59  globalNetConnect IO_A7SRC -type pgpin -pin {A7SRC } -inst io* -module {}  -override
60  globalNetConnect IO_A8SRC -type pgpin -pin {A8SRC } -inst io* -module {}  -override
61  globalNetConnect IO_A9SRC -type pgpin -pin {A9SRC } -inst io* -module {}  -override
62  globalNetConnect IO_A10SRC -type pgpin -pin {A10SRC } -inst io* -module {}
        -override
63  globalNetConnect IO_A11SRC -type pgpin -pin {A11SRC } -inst io* -module {}
        -override
64  globalNetConnect IO_A12SRC -type pgpin -pin {A12SRC } -inst io* -module {}
        -override
65  globalNetConnect IO_A13SRC -type pgpin -pin {A13SRC } -inst io* -module {} -override
66
67
68  ############ remaining 3V3 IOs pins ############
69
70  globalNetConnect vdde -type pgpin -pin {vdde3v3 } -inst io* -module {} -override
71
72  globalNetConnect IO_CLKSLEEP -type pgpin -pin {CLKSLEEP3V3 } -inst io* -module {}
        -override
73  globalNetConnect IO_TQ -type pgpin -pin {TQ3V3 } -inst io* -module {}  -override
74  globalNetConnect IO_DIGA -type pgpin -pin {CHIPSLEEP3V3 } -inst io* -module {}
        -override
75
76  globalNetConnect IO_REFA -type pgpin -pin {REFAPBIAS3V3 } -inst io* -module {}
        -override
77  globalNetConnect IO_REFB -type pgpin -pin {REFBAMPL3V3 } -inst io* -module {}
        -override
78  globalNetConnect IO_REFC -type pgpin -pin {REFCAMPH3V3 } -inst io* -module {}
        -override
79  globalNetConnect IO_REFD -type pgpin -pin {REFDNBIAS3V3 } -inst io* -module {}
        -override
80  globalNetConnect IO_REFE -type pgpin -pin {REFEIO3V3 } -inst io* -module {}
        -override
81
82  globalNetConnect IO_A0SRC -type pgpin -pin {A0SRC3V3 } -inst io* -module {}
        -override
83  globalNetConnect IO_A1SRC -type pgpin -pin {A1SRC3V3 } -inst io* -module {}
        -override
84  globalNetConnect IO_A2SRC -type pgpin -pin {A2SRC3V3 } -inst io* -module {}
        -override
85  globalNetConnect IO_A3SRC -type pgpin -pin {A3SRC3V3 } -inst io* -module {}
        -override
86  globalNetConnect IO_A4SRC -type pgpin -pin {A4SRC3V3 } -inst io* -module {}
        -override
87  globalNetConnect IO_A5SRC -type pgpin -pin {A5SRC3V3 } -inst io* -module {}
        -override
88  globalNetConnect IO_A6SRC -type pgpin -pin {A6SRC3V3 } -inst io* -module {} -override
89  globalNetConnect IO_A7SRC -type pgpin -pin {A7SRC3V3 } -inst io* -module {}
        -override
90  globalNetConnect IO_A8SRC -type pgpin -pin {A8SRC3V3 } -inst io* -module {}
        -override
91  globalNetConnect IO_A9SRC -type pgpin -pin {A9SRC3V3 } -inst io* -module {}
        -override
```

```
92  globalNetConnect IO_A10SRC -type pgpin -pin {A10SRC3V3 } -inst io* -module {}
        -override
93  globalNetConnect IO_A11SRC -type pgpin -pin {A11SRC3V3 } -inst io* -module {}
        -override
94  globalNetConnect IO_A12SRC -type pgpin -pin {A12SRC3V3 } -inst io* -module {}
        -override
95  globalNetConnect IO_A13SRC -type pgpin -pin {A13SRC3V3 } -inst io* -module {}
        -override
96
97  ###
98  globalNetConnect CLKSLEEP -type pgpin -pin {CLKSLEEP3V3 } -inst ioref* -module {}
        -override
99  globalNetConnect TQ -type pgpin -pin {TQ3V3 } -inst ioref* -module {}  -override
100 globalNetConnect DIGA -type pgpin -pin {CHIPSLEEP3V3 } -inst ioref* -module {}
        -override
101
102 globalNetConnect REFA -type pgpin -pin {REFAPBIAS3V3 } -inst ioref* -module {}
        -override
103 globalNetConnect REFB -type pgpin -pin {REFBAMPL3V3 } -inst ioref* -module {}
        -override
104 globalNetConnect REFC -type pgpin -pin {REFCAMPH3V3 } -inst ioref* -module {}
        -override
105 globalNetConnect REFD -type pgpin -pin {REFDNBIAS3V3 } -inst ioref* -module {}
        -override
106 globalNetConnect REFE -type pgpin -pin {REFEIO3V3 } -inst ioref* -module {}
        -override
107
108 globalNetConnect A0SRC -type pgpin -pin {A0SRC3V3 } -inst ioref* -module {}
        -override
109 globalNetConnect A1SRC -type pgpin -pin {A1SRC3V3 } -inst ioref* -module {}
        -override
110 globalNetConnect A2SRC -type pgpin -pin {A2SRC3V3 } -inst ioref* -module {}
        -override
111 globalNetConnect A3SRC -type pgpin -pin {A3SRC3V3 } -inst ioref* -module {}
        -override
112 globalNetConnect A4SRC -type pgpin -pin {A4SRC3V3 } -inst ioref* -module {}
        -override
113 globalNetConnect A5SRC -type pgpin -pin {A5SRC3V3 } -inst ioref* -module {}
        -override
114 globalNetConnect A6SRC -type pgpin -pin {A6SRC3V3 } -inst ioref* -module {} -override
115
116 ###
117 globalNetConnect CLKSLEEP -type pgpin -pin {CLKSLEEP3V3 } -inst ioco_vssio_ref_asrc
        -module {}  -override
118 globalNetConnect TQ -type pgpin -pin {TQ3V3 } -inst ioco_vssio_ref_asrc -module {}
        -override
119 globalNetConnect DIGA -type pgpin -pin {CHIPSLEEP3V3 } -inst ioco_vssio_ref_asrc
        -module {}  -override
120
121 globalNetConnect REFA -type pgpin -pin {REFAPBIAS3V3 } -inst ioco_vssio_ref_asrc
        -module {}  -override
122 globalNetConnect REFB -type pgpin -pin {REFBAMPL3V3 } -inst ioco_vssio_ref_asrc
        -module {}  -override
123 globalNetConnect REFC -type pgpin -pin {REFCAMPH3V3 } -inst ioco_vssio_ref_asrc
        -module {}  -override
```

```
124 globalNetConnect REFD -type pgpin -pin {REFDNBIAS3V3 } -inst ioco_vssio_ref_asrc
        -module {}  -override
125 globalNetConnect REFE -type pgpin -pin {REFEIO3V3 } -inst ioco_vssio_ref_asrc -module
        {}  -override
126
127 globalNetConnect A0SRC -type pgpin -pin {A0SRC3V3 } -inst ioco_vssio_ref_asrc -module
        {}  -override
128 globalNetConnect A1SRC -type pgpin -pin {A1SRC3V3 } -inst ioco_vssio_ref_asrc -module
        {}  -override
129 globalNetConnect A2SRC -type pgpin -pin {A2SRC3V3 } -inst ioco_vssio_ref_asrc -module
        {}  -override
130 globalNetConnect A3SRC -type pgpin -pin {A3SRC3V3 } -inst ioco_vssio_ref_asrc -module
        {}  -override
131 globalNetConnect A4SRC -type pgpin -pin {A4SRC3V3 } -inst ioco_vssio_ref_asrc -module
        {}  -override
132 globalNetConnect A5SRC -type pgpin -pin {A5SRC3V3 } -inst ioco_vssio_ref_asrc -module
        {}  -override
133 globalNetConnect A6SRC -type pgpin -pin {A6SRC3V3 } -inst ioco_vssio_ref_asrc -module
        {} -override
134
135 ### Mult IO power pad
136 globalNetConnect vddcore -type pgpin -pin {VDDCORE*} -inst ioco_vddioco_core  -module
        {} -override
137 globalNetConnect vdd0 -type pgpin -pin {VDDCORE*} -inst ioco_vddioco_0  -module {}
        -override
138 globalNetConnect vdd1 -type pgpin -pin {VDDCORE*} -inst ioco_vddioco_1  -module {}
        -override
139 globalNetConnect vdd2 -type pgpin -pin {VDDCORE*} -inst ioco_vddioco_2  -module {}
        -override
140 globalNetConnect vdd3 -type pgpin -pin {VDDCORE*} -inst ioco_vddioco_3  -module {}
        -override
141
142 ### connect cells to the correct io
143 globalNetConnect vddcore -type pgpin -pin {vdd} -inst *  -module core -override
144 globalNetConnect vdd0 -type pgpin -pin {vdd} -inst *  -module core/mult_0 -override
145 globalNetConnect vdd1 -type pgpin -pin {vdd} -inst *  -module core/mult_1 -override
146 globalNetConnect vdd2 -type pgpin -pin {vdd} -inst *  -module core/mult_2 -override
147 globalNetConnect vdd3 -type pgpin -pin {vdd} -inst *  -module core/mult_3 -override
148
149
150 ###
151 ### execute command
152 ###
153 applyGlobalNets
154
155
156 ###
157 ### check all design
158 ### (a specific check can also be performed in menu : FloorPlan->Global Net
        Connection-> check button)
159 ###
160 #checkdesign -all
```

C.6 pwr.tcl

```
#add rings (core + power_domains)

#extern ring
addRing \
    -spacing_bottom  3.0 \
    -spacing_top     3.0 \
    -spacing_right   3.0 \
    -spacing_left    3.0 \
    -width_bottom 10 \
    -width_top    10 \
    -width_right  10 \
    -width_left   10 \
    -layer_bottom M7 \
    -layer_top    M7 \
    -layer_right  M6 \
    -layer_left   M6 \
    -offset_bottom 0.45 \
    -offset_top    0.45 \
    -offset_right  0.45 \
    -offset_left   0.45 \
    -center 1 \
    -stacked_via_top_layer M7 \
    -stacked_via_bottom_layer M1 \
    -around core \
    -jog_distance 0.45 \
    -threshold 0.45 \
    -nets {gnd vddcore}

#PD0
deselectAll
selectGroup PD0
addRing \
    -type block_rings\
    -around power_domain \
    -spacing_bottom 1.5 \
    -spacing_top    1.5 \
    -spacing_right  1.5 \
    -spacing_left   1.5 \
    -width_bottom 8 \
    -width_top    8 \
    -width_right  8 \
    -width_left   8 \
    -layer_bottom M7 \
    -layer_top    M7 \
    -layer_right  M6 \
    -layer_left   M6 \
    -offset_bottom 0.45 \
    -offset_top    0.45 \
    -offset_right  0.45 \
    -offset_left   0.45 \
    -stacked_via_top_layer M7 \
    -stacked_via_bottom_layer M1 \
    -jog_distance 0.45 \
```

```
54      -threshold 0.45 \
55      -nets {vdd0}
56  deselectGroup PD0
57
58  #PD1
59  selectGroup PD1
60  addRing \
61      -type block_rings\
62      -around power_domain \
63      -spacing_bottom 1.5 \
64      -spacing_top    1.5 \
65      -spacing_right  1.5 \
66      -spacing_left   1.5 \
67      -width_bottom 8 \
68      -width_top    8 \
69      -width_right  8 \
70      -width_left   8 \
71      -layer_bottom M7 \
72      -layer_top    M7 \
73      -layer_right  M6 \
74      -layer_left   M6 \
75      -offset_bottom 0.45 \
76      -offset_top    0.45 \
77      -offset_right  0.45 \
78      -offset_left   0.45 \
79      -stacked_via_top_layer M7 \
80      -stacked_via_bottom_layer M1 \
81      -jog_distance 0.45 \
82      -threshold 0.45 \
83      -nets {vdd1}
84  deselectGroup PD1
85
86  #PD2
87  selectGroup PD2
88  addRing \
89      -type block_rings\
90      -around power_domain \
91      -spacing_bottom 1.5 \
92      -spacing_top    1.5 \
93      -spacing_right  1.5 \
94      -spacing_left   1.5 \
95      -width_bottom 8 \
96      -width_top    8 \
97      -width_right  8 \
98      -width_left   8 \
99      -layer_bottom M7 \
100     -layer_top    M7 \
101     -layer_right  M6 \
102     -layer_left   M6 \
103     -offset_bottom 0.45 \
104     -offset_top    0.45 \
105     -offset_right  0.45 \
106     -offset_left   0.45 \
107     -stacked_via_top_layer M7 \
108     -stacked_via_bottom_layer M1 \
```

```
109      -jog_distance 0.45 \
110      -threshold 0.45 \
111      -nets {vdd2}
112  deselectGroup PD2
113
114  #PD3
115  selectGroup PD3
116  addRing \
117      -type block_rings\
118      -around power_domain \
119      -spacing_bottom 1.5 \
120      -spacing_top    1.5 \
121      -spacing_right  1.5 \
122      -spacing_left   1.5 \
123      -width_bottom 8 \
124      -width_top    8 \
125      -width_right  8 \
126      -width_left   8 \
127      -layer_bottom M7 \
128      -layer_top    M7 \
129      -layer_right  M6 \
130      -layer_left   M6 \
131      -offset_bottom 0.45 \
132      -offset_top    0.45 \
133      -offset_right  0.45 \
134      -offset_left   0.45 \
135      -stacked_via_top_layer M7 \
136      -stacked_via_bottom_layer M1 \
137      -jog_distance 0.45 \
138      -threshold 0.45 \
139      -nets {vdd3}
140  deselectGroup PD3
141
142  #PDCORE
143  selectGroup PDCORE
144  addRing \
145      -type block_rings\
146      -around power_domain \
147      -spacing_bottom 1.5 \
148      -spacing_top    1.5 \
149      -spacing_right  1.5 \
150      -spacing_left   1.5 \
151      -width_bottom 8 \
152      -width_top    8 \
153      -width_right  8 \
154      -width_left   8 \
155      -layer_bottom M7 \
156      -layer_top    M7 \
157      -layer_right  M6 \
158      -layer_left   M6 \
159      -offset_bottom 0.45 \
160      -offset_top    0.45 \
161      -offset_right  0.45 \
162      -offset_left   0.45 \
163      -stacked_via_top_layer M7 \
```

```
164     -stacked_via_bottom_layer M1 \
165     -jog_distance 0.45 \
166     -threshold 0.45 \
167     -left 0 \
168     -tl 1\
169     -bl 1\
170     -nets {vddcore}
171 deselectGroup PDCORE
172
173 #IO_REF_COMPENSATION
174 addRing \
175     -type block_rings\
176     -around each_block \
177     -spacing_bottom 1.5 \
178     -spacing_top    1.5 \
179     -spacing_right  1.5 \
180     -spacing_left   1.5 \
181     -width_bottom 8 \
182     -width_top    8 \
183     -width_right  8 \
184     -width_left   8 \
185     -layer_bottom M7 \
186     -layer_top    M7 \
187     -layer_right  M6 \
188     -layer_left   M6 \
189     -offset_bottom 0.55 \
190     -offset_top    0.55 \
191     -offset_right  0.55 \
192     -offset_left   0.55 \
193     -stacked_via_top_layer M7 \
194     -stacked_via_bottom_layer M1 \
195     -jog_distance 0.45 \
196     -threshold 0.45 \
197     -nets {vdd vdde}
198
199 addRing \
200     -type block_rings\
201     -around each_block \
202     -spacing_bottom 1.5 \
203     -spacing_top    1.5 \
204     -spacing_right  1.5 \
205     -spacing_left   1.5 \
206     -width_bottom 8 \
207     -width_top    8 \
208     -width_right  8 \
209     -width_left   8 \
210     -layer_bottom M7 \
211     -layer_top    M7 \
212     -layer_right  M6 \
213     -layer_left   M6 \
214     -offset_bottom 20.5 \
215     -offset_top    20.5 \
216     -offset_right  20.5 \
217     -offset_left   20.5 \
218     -stacked_via_top_layer M7 \
```

```
219     -stacked_via_bottom_layer M1 \
220     -threshold 0.45 \
221     -bottom 0 \
222     -right 0 \
223     -lb 1 \
224     -tr 1 \
225     -nets {gnd}
```

C.7 followPin.tcl

```
1  Puts "#################################"
2  Puts "###"
3  Puts "### Create std-cell follow pin"
4  Puts "###"
5  Puts "#################################"
6
7  deselectAll
8  cutCoreRow
9
10 #avoid via6_7 on VDDCO_HDRV_MT_1V0_LIN edge
11 createRouteBlk -box 233.4700 1042.6900 276.7550 1047.9950 -layer 6
12 createRouteBlk -box 911.768 1042.69 955.176 1049.237 -layer 6
13 createRouteBlk -box 504.69 139.071 548.532 146.015 -layer 6
14 createRouteBlk -box 232.895 140.142 277.333 146.272 -layer 6
15
16 # Use editPowerVia to generate stripes-followpins
17 #-noBlockPins           firstAfterRowEnd
18 sroute -verbose -noPadRings    -noStripes \
19        -corePinMaxViaWidth 30 -corePinMaxViaHeight 70 \
20        -targetViaTopLayer 7 -crossoverViaTopLayer 7 \
21        -secondaryStopSCPin firstStripe \
22        -viaConnectToShape { stripe ring } \
23        -deleteExistingRoutes \
24        -padPinWidth 7\
25        -nets {gnd vdd vdd0 vdd1 vdd2 vdd3 vddcore vdde}
26
27 #avoid routing too close to the pad
28 createRouteBlk -box 923.796 143.219 927.711 156.959 -layer all
29
30 #Route IO_REF special nets
31 sroute -verbose -noPadRings -padPinToAlignedBlockPin \
32        -stopStripeSCPin lastPadRing -deleteExistingRoutes -nets {\
33      CLKSLEEP TQ DIGA DIGB KOFF REFA REFB REFC REFD REFE REFF \
34      A6SRC A5SRC A4SRC A3SRC A2SRC A1SRC A0SRC }
35 #      A13SRC A12SRC A11SRC A10SRC A9SRC A8SRC A7SRC
36
37 #Remove blockages
38 deleteAllRouteBlks
39
40 clearCutRow
41 deselectAll
```

C.8 place_output_bufs.tcl

```
placeInstance  Z_svt_buf 194.04 570.385 MY
placeInstance  Z_lvt_buf 194.04 535.085  R180
placeInstance  s_out_buf 194.04 409.714 R180
```

C.9 output_nets.tcl

```
#outputs nets with width of 1 um
setEdit -force_special 1
setEdit -width_horizontal 1
setEdit -width_vertical 1

#s_out
setEdit -nets s_out
setEdit -layer_horizontal M1
setEdit -layer_vertical M2
uiSetTool addWire
editAddRoute 194.847 411.480
editAddRoute 145.233 411.459
editAddRoute 144.821 413.911
editAddRoute 145.115 413.911
editCommitRoute 145.115 413.911
setEdit -layer_horizontal M2
editAddRoute 142.614 413.933
editAddRoute 145.259 413.439
editAddRoute 145.043 413.933
editCommitRoute 145.043 413.933
uiSetTool select

#Z_lvt
setEdit -nets Z_lvt
setEdit -layer_horizontal M1
setEdit -layer_vertical M2
uiSetTool addWire
editAddRoute 195.073 536.971
editAddRoute 145.874 536.559
editAddRoute 145.028 549.990
setEdit -layer_horizontal M2
editAddRoute 142.497 549.872
editCommitRoute 142.497 549.872
uiSetTool select

#Z_svt
setEdit -nets Z_svt
uiSetTool addWire
setEdit -layer_horizontal M1
setEdit -layer_vertical M2
editAddRoute 195.030 572.574
editAddRoute 146.930 572.574
editAddRoute 146.764 685.499
setEdit -layer_horizontal M2
editAddRoute 142.467 685.146
editCommitRoute 142.467 685.146
```

```
47 uiSetTool select
```

C.10 fix_drc_errors.tcl

```
1  ##################################################
2  #Fix the DRC errors discovered with calibre DRC
3  ##################################################
4
5  #lower net
6  #move the first part
7  selectWire 950.5100 203.6300 952.6100 203.7700 3 gnd
8  selectWire 952.4700 203.6300 954.2900 203.7700 3 gnd
9  selectWire 954.1500 203.6300 981.6100 203.7700 3 gnd
10 editMove y -1.399
11 deselectAll
12 #add via at the end
13 setEdit -layer_horizontal M3 -layer_vertical  M4 -nets gnd
14 setEdit -width_horizontal 0.14 -width_vertical 0.14
15 editAddRoute 981.604 202.292
16 editAddRoute 983.905 202.306
17 editCommitRoute 983.905 202.306
18 uiSetTool select
19 deselectAll
20
21 ##
22 selectWire 983.9900 203.6300 984.6900 203.7700 3 gnd
23 editDelete -objects Selected
24 deleteTiles -selected
25 deleteBumps -selected
26 selectWire 984.5500 203.6300 984.6900 244.8500 2 gnd
27 editStretch y -0.683 low
28 editAddRoute 984.610 203.019
29 editAddRoute 981.323 203.076
30 editCommitRoute 981.323 203.076
31 uiSetTool select
32 deselectAll
33
34 ##################################
35
36 #upper net
37 #delete existing wire
38 selectWire 946.4700 450.8700 947.7300 451.0100 5 gnd
39 selectWire 947.5900 450.8700 947.7300 451.5700 4 gnd
40 selectWire 947.5900 451.4300 981.6100 451.5700 3 gnd
41 editDelete -objects Selected
42 deleteTiles -selected
43 deleteBumps -selected
44 #create the new one
45 editAddRoute 942.920 448.604
46 editAddRoute 944.305 455.932
47 editAddRoute 984.069 455.849
48 editCommitRoute 984.069 455.849
49 setEdit -layer_vertical M3
50 #add extra via
```

```
editAddRoute 942.905 455.921
editAddRoute 942.913 455.503
editAddRoute 942.961 455.527
editCommitRoute 942.961 455.527
setEdit -layer_vertical M4
uiSetTool select
deselectAll

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

#move offending net M1.S.3.1
#uiSetTool moveWire
selectWire 385.6300 383.6700 387.7300 383.8100 3 clk_p
selectWire 383.6700 383.6700 385.7700 383.8100 3 clk_p
editMove y 1.246
deselectAll

selectWire 382.8400 383.4000 385.4200 383.5200 1 core/data_gen_1/clk_p__Fence_N0
editDelete -objects Selected
deleteTiles -selected
deleteBumps -selected
selectWire 380.5900 383.3900 382.9700 383.5300 3 core/data_gen_1/clk_p__Fence_N0
editStretch x 3.292 high
uiSetTool select
deselectAll

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

#move an offending net on M1
selectWire 911.2600 528.7200 911.6000 528.8400 1 core/mult_3/mult_par4_0/
    AandBx15xx17x
editMove y 0.558
uiSetTool select
deselectAll
```

C.11 top.ctstch

```

### CLK ###

# Sample Gated CTS Command
AutoCTSRootPin io_clk/ZI

NoGating     NO
Buffer   IVSVTX6 BFSVTX1 BFSVTX8 BFSVTX10 BFSVTX12 IVLVTX6 BFLVTX1 BFLVTX8 BFLVTX10
    BFLVTX12

MaxDelay     10ps
MinDelay     0ps
MaxSkew  100ps

End


```

```
17 ### Reset ###
18
19 # Sample Gated CTS Command
20 AutoCTSRootPin io_rst_n/ZI
21
22 NoGating      NO
23 Buffer   IVSVTX6 BFSVTX1 BFSVTX2 BFSVTX4 BFSVTX6 BFSVTX8 BFSVTX12 IVLVTX6 BFLVTX1
       BFLVTX2 BFLVTX4 BFLVTX6 BFLVTX8 BFLVTX12
24
25
26 MaxSkew 1ns
27
28 End
```

C.12 ioplace.io

```
1  ############################################################
2  #                                                          #
3  #  Silicon Perspective, A Cadence Company                  #
4  #  FirstEncounter IO Assignment                            #
5  #                                                          #
6  ############################################################
7
8  Version: 2
9
10 Pad: io_corner_4 SE  CORNER_LIN
11 Pad: io_corner_3 NE  CORNER_LIN
12 Pad: io_corner_2 NW  CORNER_LIN
13 Pad: io_corner_1 SW  CORNER_LIN
14
15 Pad: ioco_vddioco_1      N VDDCO_HDRV_MT_1V0_LIN
16 Pad: io_vssio_2          N VSSIO_3V3_LIN
17 Pad: ioco_vddioco_core   N VDDCO_HDRV_MT_1V0_LIN
18 Pad: io_vddio_2          N VDDIO_3V3_LIN
19 Pad: ioco_vssioco_3      N VSSIOCO_LIN
20 Pad: ioco_vddioco_3      N VDDCO_HDRV_MT_1V0_LIN
21
22 Pad: io_s_in             W
23 Pad: io_s_out            W
24 Pad: io_Z_lvt            W
25 Pad: io_Z_svt            W
26 Pad: io_shift_n          W
27 Pad: ioco_vssioco_2      W VSSIOCO_LIN
28
29 Pad: ioco_vddioco_0      S VDDCO_HDRV_MT_1V0_LIN
30 Pad: io_clk              S
31 Pad: ioco_vddioco_2      S VDDCO_HDRV_MT_1V0_LIN
32 Pad: io_load_n           S
33 Pad: ioco_vssioco_1      S VSSIOCO_LIN
34 Pad: ioco_vssio_ref_asrc S VSSIO_3V3_REF_ASRC_LIN
35
36 Pad: ioco_vddioco_g      E VDDIOCO_LIN
37 Pad: ioco_vssioco_g      E VSSIOCO_LIN
38 Pad: io_sel0             E
```

```
39 Pad: io_sel1              E
40 Pad: io_sel_reg           E
41 Pad: io_rst_n             E
```

Appendix D

FPGA source code

D.1 main_FPGA.vhd

```
1  ----------------------------------------------------------------------------
2  -- Title      : FPGA code for demostrator test board
3  -- Project    :
4  ----------------------------------------------------------------------------
5  -- File       : main_FPGA.vhd
6  -- Author     :   <mtschuster@WS-3439>
7  -- Company    :
8  -- Created    : 2007-02-03
9  -- Last update: 2007-02-03
10 -- Platform   :
11 -- Standard   : VHDL'93
12 ----------------------------------------------------------------------------
13 -- Description: This code generate the stimuli needed to:
14 --                  1) Select the desired multiplier;
15 --                  2) Reset internal registers;
16 --                  3) Execute 10'000'000 of Multiply and Accumulate on the 64
17 --                  bit register;
18 --                  4) Read back the content of the accumulator register with a
19 --                  frequency divided by 4;
20 --                  5) Verify the read data with the expected value and output
21 --                  the decision on the pass/fail pins;
22 --                  6) At the end of this sequence, chip clock is stopped to
23 --                  allow static power measurements.
24 ----------------------------------------------------------------------------
25 -- Copyright (c) 2007
26 ----------------------------------------------------------------------------
27 -- Revisions  :
28 -- Date        Version  Author  Description
29 -- 2007-02-03  1.0      mtschuster  Created
30 ----------------------------------------------------------------------------
31 library ieee;
32 use ieee.std_logic_1164.all;
33 use ieee.std_logic_unsigned.all;
34
35 entity main is
```

```
36    port(
37  --4 user switches (normally ON)
38       S1            : in  std_logic;
39       S2            : in  std_logic;
40       S3            : in  std_logic;
41       S4            : in  std_logic;
42  --relay outputs for the 4 multipliers
43  -- BEWARE: 0=on (VDDM); 1=off(GND)
44       P1            : out std_logic;  -- mult0: RCA32_SVT
45       P2            : out std_logic;  -- mult1: RCA32_PAR4_SVT
46       P3            : out std_logic;  -- mult2: RCA32_LVT
47       P4            : out std_logic;  -- mult3: RCA32_PAR4_LVT
48  -- Test result leds
49       OK_led        : out std_logic;       -- test passed
50       KO_led        : out std_logic;       -- test failed
51  -- Control FPGA pins
52       mult_num      : in  std_logic_vector(1 downto 0);  --multiplier selector
53  -- Serial interface pins
54       CHIP_sout     : in  std_logic;       -- serial interface output
55       CHIP_sin      : out std_logic;       -- serial interface input
56       CHIP_shift_n  : out std_logic;       -- enable bit shifting, active low
57       CHIP_load_n   : out std_logic;       -- enable parallel load, active low
58       CHIP_sel      : out std_logic_vector(1 downto 0); -- select the multiplier unde
             test
59       CHIP_sel_reg  : out std_logic;       -- route to/from the shift register
60       CHIP_clock    : out std_logic;       -- chip clock
61       CHIP_rst_n    : out std_logic;       -- chip asynchronous reset, active low
62       clock         : in  std_logic;       -- FPGA clock
63       );
64  end main;
65
66  architecture arch of main is
67    -- state machine states
68    type   FSM_states is (INIT, RUN, READBACK, VERIFY);
69    signal curr_state, next_state : FSM_states;
70    signal rst_n                  : std_logic;  -- global reset
71    signal count                  : integer range 0 to 16777215;   -- counter delaying
          the next state
72
73    signal clock_slow             : std_logic; -- clock divided by 4
74    signal clock_slow_enable      : std_logic; -- enable clock_slow, active high
75    signal clock_div_counter      : std_logic_vector(1 downto 0); -- clock divider
          counter
76
77    signal read                   : std_logic; -- readback trigger
78    signal data                   : std_logic_vector(63 downto 0); --readback data
79    signal fail                   : std_logic;  -- test passed
80    signal pass                   : std_logic;  -- test failed
81
82    signal mult_sel               : std_logic_vector(1 downto 0); --multiplier
          selection
83  begin
84  ------------------------------------------------------------------------------
85  -- COMBINATORIAL LOGIC
86  ------------------------------------------------------------------------------
```

```
87
88  -- reset: through switch 4
89    rst_n       <= S4;
90  --multiplier selection and chip clock multiplexing
91    mult_sel    <= mult_num;
92    CHIP_sel    <= mult_sel;
93    clock_slow  <= clock_div_counter(1);
94    CHIP_clock  <= clock_slow when clock_slow_enable = '1' else clock ;
95  --test result leds
96    OK_led      <= pass;
97    KO_led      <= fail;
98  --select multipliers power
99    P1          <= '0' when mult_sel = "00" else '1';
100   P2          <= '0' when mult_sel = "01" else '1';
101   P3          <= '0' when mult_sel = "10" else '1';
102   P4          <= '0' when mult_sel = "11" else '1';
103
104   -- Finite state machine definition
105   FSM : process(curr_state, count, mult_sel, data, S4)
106     -- number of clock of the init state
107     constant INIT_LENGTH         : integer                          := 4;
108     -- number of clocks for the running state
109     -- it corresponds to number of multiplications + 2
110     -- parallel multipliers require 3 extra clocks due to latency
111     constant RUNNING_LENGTH      : integer                          := 10000002;
112     constant RUNNING_LENGTH_PAR : integer                           := RUNNING_LENGTH +
            3;
113     -- number of clock to execute readback task based on full speed clock
114     constant READBACK_LENGTH     : integer                          := 254;
115
116     -- expected result after 10'000'000 multiplications and accumulations
117     constant EXPECTED_RESULT     : std_logic_vector(63 downto 0)    := X"0
            E4DD39EA61421FC";
118     -- on low supply voltages (<0.4V) one extra multiplication can occur
119     constant EXPECTED_RESULT_LV  : std_logic_vector(63 downto 0)    := X"1628
            d37ce47c248c";
120
121   begin
122 --- chip defaults values
123     CHIP_sin          <= '0';
124     CHIP_shift_n      <= '1';
125     CHIP_load_n       <= '1';
126     CHIP_sel_reg      <= '0';
127     clock_slow_enable <= '0';
128     CHIP_rst_n        <= '1';
129     read              <= '0';
130     pass              <= '0';
131     fail              <= '0';
132     -- state machine
133     case curr_state is
134       ------------------------------------------------------------------------
135       -- load zeros from random generator to serial interface registers
136       ------------------------------------------------------------------------
137       when INIT =>
138         CHIP_load_n  <= '0';              -- parallel load
```

```
139           CHIP_shift_n <= '1';                -- no shift
140           CHIP_sel_reg <= '1';                -- from rand to regs
141           CHIP_rst_n   <= '0';                -- mantain the random generator to zeros
142           -- after the init time is passed go to the next state
143           if count = INIT_LENGTH then
144             next_state <= RUN;
145           else
146             next_state <= INIT;
147           end if;
148         ------------------------------------------------------------------------
149         -- run the multiplications and accumulations
150         ------------------------------------------------------------------------
151         when RUN =>
152           CHIP_load_n  <= '0';                -- parallel load
153           CHIP_shift_n <= '1';                -- no shift
154           CHIP_sel_reg <= '0';                -- from rand to multiplier
155           CHIP_rst_n   <= '1';                -- activate the random generator
156           -- after the multication and accumulation, go to the next state
157           -- due to the parallel nature of mult1 and mult3, few extra clocks are
                required
158           if (count = RUNNING_LENGTH and mult_sel(0) = '0')
159             or (count = RUNNING_LENGTH_PAR and mult_sel(0) = '1') then
160             next_state <= READBACK;
161           else
162             next_state <= RUN;
163           end if;
164         ------------------------------------------------------------------------
165         -- read back values from the registers through the serial interface
166         ------------------------------------------------------------------------
167         when READBACK =>
168           CHIP_load_n  <= '1';              -- serial behaviour
169           CHIP_shift_n <= '0';              -- activate data shifting
170           read         <= '1';              -- activate readback
171           clock_slow_enable <= '1';         -- switch to slow clock
172           -- trigger data reading and once finished go to the final state
173           if count = READBACK_LENGTH then
174             next_state <= VERIFY;
175           else
176             next_state <= READBACK;
177           end if;
178         ------------------------------------------------------------------------
179         -- verify read data with the expected data and output the result to leds
180         ------------------------------------------------------------------------
181         when VERIFY =>
182           next_state        <= VERIFY;      -- looped state until a reset is fired
183           clock_slow_enable <= '1';         -- remain with slow clock
184           if (data = EXPECTED_RESULT) or (data = EXPECTED_RESULT_LV) then
185             pass <= '1';                    -- green LED on
186           else
187             fail <= '1';                    -- red LED on
188           end if;
189         ------------------------------------------------------------------------
190         -- if something strange happens, go to the init state
191         ------------------------------------------------------------------------
192         when others =>
```

```
193           next_state          <= INIT;
194       end case;
195     end process FSM;
196
197   ------------------------------------------------
198   -- SEQUENTIAL LOGIC
199   ------------------------------------------------
200   --asynchrounous reset registers
201     FSM_regs : process(clock, rst_n)
202     begin
203       if rst_n = '0' then
204         curr_state <= INIT;
205         data       <= (others => '0');
206       elsif (clock'event and clock = '1') then
207         curr_state <= next_state;
208         -- read data back when read = '1'
209         if read = '1' and clock_div_counter = "01" then
210           data <= data(62 downto 0)&CHIP_sout;
211         end if;
212       end if;
213     end process FSM_regs;
214
215   --synchronous reset registers
216   --at each new FSM state the counter is reset
217     counter : process(clock, rst_n)
218     begin
219       if clock'event and clock = '1' then
220         if rst_n = '0' or (curr_state /= next_state) then
221           count <= 0;
222         else
223           count <= count + 1;
224         end if;
225       end if;
226     end process counter;
227
228   --generate lower frequency clock
229     gen_CHIP_clk: process (clock, read)
230     begin
231       if clock'event and clock = '0' then
232         if read = '0' then
233           clock_div_counter <= "10";
234         else
235           clock_div_counter <= clock_div_counter + "01";
236         end if;
237       end if;
238     end process gen_CHIP_clk;
239   end arch;
```

Appendix E

MATLAB based automated test functions

E.1 test_mult.m

```
function data = test_mult(mult, freq, volt)
%mult in 0-3
%freq lower than 80MHz
%volt lower or equal to 1V

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Connect devices
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Open devices
k2400 = visa('ni', 'GPIB0::13::0::INSTR');
k213 = visa('ni', 'GPIB0::11::0::INSTR');
agilent = visa('ni', 'GPIB0::10::0::INSTR');
fopen(k2400);
fopen(k213);
fopen(agilent);

% Get information about devices
fprintf(k2400, '*IDN?');
current_sense = fscanf(k2400)
voltage_source = fscanf(k213)
fprintf(agilent, '*IDN?');
frequency_generator = fscanf(agilent)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize devices
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Reset devices
fprintf(k2400, '*RST');
fprintf(agilent, '*RST');
% Prepare the k2400 for current measurements
fprintf(k2400, ':SOUR:FUNC VOLT'); % set source to voltage
```

```
33 fprintf(k2400, ':SOUR:VOLT:MODE FIXED'); % set source to DC
34 fprintf(k2400, ':SOUR:VOLT 0'); % reset source to 0
35 fprintf(k2400, ':SENS:FUNC "CURR"'); % select current measurement
36 fprintf(k2400, ':CURR:NPLC 0.1'); % set integration time 1 = 1/50Hz, 0.1 = 1/500Hz
37 fprintf(k2400, ':CURR:PROT 0.02'); % set Compliant to 20mA
38 fprintf(k2400, ':CURR:RANG 0.01'); % set range to 10mA
39 fprintf(k2400, ':FORM:ELEM CURR'); % set current data format
40 fprintf(k2400, ':TRIG:COUNT 5'); % number of multi read
41 fprintf(k2400, ':ARM:SOUR PSTEST'); % enable trigger on positive edge of SOT
42 fprintf(k2400, ':SOUR:DEL 0.05'); % intra measure delay to 50ms
43 fprintf(k2400, ':OUTP ON'); % enable output
44
45
46 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
47 % Body of the code
48 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49
50 set_mult(mult, k213); % select multiplier
51
52 i = 0;
53 for f = freq % for each frequency do
54     set_freq(f, agilent); % set the frequency
55     pause(2);  % allow frequency to stabilize
56     i = i+1; j = 0;
57     for V = volt % for each supply voltage do
58         j = j+1;
59         % check that supply voltage never exceed 1V
60         vdd_core = V+0.1; % set core voltage 100mV higher than multiplier
61         if vdd_core > 1
62             vdd_core = 1;
63         end
64         if V > 1
65             V = 1;
66         end
67
68         set_voltage(vdd_core, k213);   % set the core supply voltage
69         fprintf(k2400, [':SOUR:VOLT ' num2str(V) ]); % set multiplier supply voltage
70         start_off(k213);         % reset the FPGA
71         fprintf(k2400, ':INIT');% arm the current sensing
72         start_on(k213);          % activate the FPGA and trigger the sensing
73         dyn = str2num(get_current(k2400)); % read the current values
74         pass = pass_test(k213); % check if test pass of fail
75         fprintf(k2400, ':ARM:SOUR IMM'); % take an immadiate measure for static
76         fprintf(k2400, ':INIT');  % arm the current sensing
77         stat = str2num(get_current(k2400)); % read the current values
78         data(i,j,:) = [f V max(dyn) min(stat) pass]; % store results in data
79         fprintf(k2400, ':ARM:SOUR PSTEST'); % enable trigger on positive edge of SOT
80     end
81 end
82
83 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
84 % Disconnect devices
85 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
86 % Disable outputs
87 fprintf(k2400, 'OUTP OFF');
```

```
88  fprintf(agilent, 'OUTP OFF');
89
90  % Close devices
91  fclose(k2400);
92  fclose(k213);
93  fclose(agilent);
94
95  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
96  %% Agilent 33250A frequency generator code
97  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
98  function set_freq(freq, dev) % set a square clock on agilent
99  fprintf(dev, ['APPL:SQU ' num2str(freq) ',3.3, 1.65']);
100
101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
102 %% Keithley 2400 current sensing code
103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104 function curr = get_current_one(dev ) % take a one-shot measure
105 fprintf(dev, ':MEAS:CURR?');
106 curr = fscanf(dev);
107
108 function curr = get_current(dev) % take a current measure
109 fprintf(dev, ':FETCH?')
110 curr = fscanf(dev);
111
112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
113 %% Keithley 213 quad voltage source code
114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
115 function set_voltage(v, dev) % calibrated voltage on k213
116 fprintf(dev, ['P1C0A0R1H0J128,143V' num2str(v) 'X']);
117
118 function start_on(dev) % CHIP_rst_n high
119 fprintf(dev, 'P4V3.3X');
120
121 function start_off(dev) % CHIP_rst_n low
122 fprintf(dev, 'P4V0X');
123
124 function set_mult(num, dev); % select the multiplier under test
125 if num == 0
126     str1 = 'P2V0X';
127     str2 = 'P3V0X';
128 elseif num == 1
129     str1 = 'P2V3.3X';
130     str2 = 'P3V0X';
131 elseif num == 2
132     str1 = 'P2V0X';
133     str2 = 'P3V3.3X';
134 else
135     str1 = 'P2V3.3X';
136     str2 = 'P3V3.3X';
137 end
138 fprintf(dev, str1);
139 fprintf(dev, str2);
140
141 function pass = pass_test(dev) %wait for test results and check if test passed
142 pass = 0;
```

```
143 fail = 0;
144 while (pass == 0  && fail == 0)
145     fprintf(dev, 'U5X');
146     din = str2num(fscanf(dev));
147     pass = bitget(din,1);
148     fail = bitget(din,3);
149 end
```