Power and Thermal Management of System-on-Chip

Liu, Wei

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

Power and Thermal Management of System-on-Chip

Wei Liu

Kongens Lyngby 2011
IMM-PHD-2011-250

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

With greater integration of VLSI circuits, power consumption and power density have increased dramatically resulting in high chip temperatures and presenting a heat removal challenge. To effectively limit the high temperature inside a chip, thermal specific approaches, besides low power techniques, are necessary at the chip design level.

In this work, we investigate the power and thermal management of System-on-Chips (SoCs). Thermal analysis is performed in a SPICE simulation approach based on the electrical-thermal analogy. We investigate the impact of interconnects on heat distribution in the substrate and present a way to consider temperature dependent signal delay in global wires at early design stages.

With the aim of reducing high local power density in hotspots, we propose two placement techniques to spread hot cells over a larger area. The proposed methods are compared in terms of temperature reduction, timing and area overhead to the general method, which enlarges the circuit area uniformly.

A case study analyzes the design of Floating Point Units (FPU) from an energy and a thermal perspective. For the division operation, we compare different implementations and illustrate the impact of power efficient dividers on the energy consumption and thermal distribution within the FPU and the on-chip cache. We also characterize the temperature dependent static dissipation to evaluate the reduction in leakage obtained from the decrease in temperature.

Resumé

Ved øget integration af VLSI kredsløb øges effektforbrug og effekttæthed dramatisk med højere temperaturer på chippen til følge. Dette forhold gør fjernelse af energi en design udfordring. På chip design niveau er termisk specifikke metoder nødvendige for effektivt at begrænse temperaturen på chippen.

I dette arbejde undersøges metoder for styring af effekt og temperatur for System-on-Chips (SoCs). Termisk analyse baseres på SPICE simuleringer af elektriske ækvivalentkredsløb for det termiske system. Indflydelsen af ledningsforbindelser for temperaturfordelingen i substratet undersøges, og en metode for estimering af temperaturafhængige signalforsinkelser på et tidligt stadie af design præsenteres.

Med målsætningen om at reducere punkter med høj lokal effekttæthed (hotspots) foreslås to placeringsteknikker for fordeling af hotspots over større areal. De foreslåede metoder sammenlignes med den generelle metode, hvor kredsløbsarealet øges uniformt, med hensyn til opnået temperaturreduktion samt omkostninger i form af øget forsinkelse og areal.

I et case study analyseres design af enheder til beregninger med flydende tal (Floating Point Units, FPU) med hensyn til energi og temperatur. For divisionsoperationen sammenlignes forskellige implementationer, og betydningen af energi-effektive divisionskredsløb illustreres ved energiforbrug og temperaturfordeling inden for FPU og i on-chip cache-lagre. Endelig karakteriseres det temperatur-afhængige statiske effektforbrug for at beregne reduktion i lækstrømme som følge af den lavere temperatur.

Preface

This thesis was prepared at the Department of Informatics and Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the degree of Doctor of Philosophy in engineering.

The thesis deals with power and thermal management of System-on-Chips. The main focus is on investigating thermal behaviors within VLSI circuits, reducing peak temperature in hotspots and optimizing energy and leakage consumption.

The thesis is self-contained and relies on the work done in a number of research papers written during the period 2007–2010.

Lyngby, December 2010

Wei Liu

Acknowledgements

First and foremost, I would like to thank my supervisor, Alberto Nannarelli, for his persistent encouragement and guidance over the years. I am deeply grateful for his confidence in me and for having the opportunity to work with him. I would also like to thank my co-supervisor, Jan Madsen, for providing great insight in state of the art of research and the way to conduct successful PhD study.

Part of the work was carried out during my external stay at the Electronic Design Automation group at Politecnico di Torino. I would like to express thanks to Professor Enrico Macii and Professor Massimo Poncino for hosting me at Polito and also to Andrea Calimera who worked closely with me on the project. Our discussions gave me great insight in the area of design automation and inspired many ideas that were later put down in this work.

I would also like to thank all the colleagues in the Embedded Systems Engineering section at DTU Informatics, for creating a great atmosphere to exchange ideas and opinions.

Finally, the greatest thanks go to my parents and my girlfriend for their enduring encouragement, support and love.

Curriculum Vitae

Wei Liu

2005	Bachelor in Engineering, Zhejiang University (China)
2006-07	Student Assistant, Center for Biological Sequence Analysis, Technical University of Denmark
2007	MSc in Engineering, Technical University of Denmark
2009	Visiting Student, Electronic Design Automation Group, Politecnico di Torino (Italy)
2011	PhD in Engineering, Technical University of Denmark Dissertation: Power and Thermal Management of System-on-Chip Supervisor: Alberto Nannarelli and Jan Madsen

PUBLICATIONS

W.Liu and A.Nannarelli, "Power Dissipation in Division", *Proc. of 42nd Asilomar Conference on Signals, Systems and Computers*, pp. 1790-1794, Oct. 2008

W.Liu and A.Nannarelli, "Net Balanced Floorplanning Based on Elastic Energy Model", *Proc. of 26th Norchip Conference*, pp. 258-263, Nov. 2008

W.Liu, A.Calimera, A.Nannarelli, E.Macii and M.Poncino, "On-Chip Thermal Modeling Based on SPICE Simulation", *Proc. of 19th International Workshop on Power And Timing Modeling, Optimization and Simulation*, pp. 66-75, Sept. 2009

W.Liu, A.Nannarelli, A.Calimera, E.Macii and M.Poncino, "Post Placement Temperature Reduction Techniques", *Proc. of 2010 Design, Automation Test in Europe Conference Exhibition*, pp. 634-637, Mar. 2010

W.Liu and A.Nannarelli, "Power Dissipation Challenges in Multicore Floating-Point Units", *Proc. of 21st IEEE International Conference on Application-specific Systems Architectures and Processors*, pp. 257-264, Jul. 2010

W.Liu and A.Nannarelli, "Temperature Aware Power Optimization for Multicore Floating-Point Units", *Proc. of 44th Asilomar Conference on Signals, Systems and Computers*, pp. 1134-1138, Nov. 2010

Contents

Chapter 1

Introduction

Over the past few decades, the semiconductor industry has seen a revolutionary increase in computing performance, which is largely achieved through doubling the number of transistors on a chip almost every two years. This trend of technology scaling was already predicted by Intel's co-founder Gordon Moore as early as 1965, and often referred to as Moore's Law. In 2010, the industry has passed the two billion transistor milestone with the release of several high end server processors.

The dramatic increase in the degree of integration has allowed the design of more and more powerful systems, ranging from large mainframe servers constituting backbones of the Internet to portable handheld devices, which enable people to be connected at anytime from anywhere. The advancement in manufacturing technology and design methodology makes it possible to put an entire System on a Chip (SoC), integrating all components of an electronic system into a single Integrated Circuit (IC).

The increasing performance of Very Large Scale Integration (VLSI) circuits is accompanied by the increasing power and power density, which exhibit themselves in the form of heat and present a cooling challenge. The cost of cooling solutions is a nonlinear function of power and to a large extent limits the maximum amount of power that can be dissipated in a chip. This is also referred to as the *Power Wall* of commodity processors, which makes power and thermal

management both a technical and an economic challenge.

Heat generated at transistor junctions flows towards the heat sink and the ambient environment mainly through the substrate and the chip package. Analogous to electrical resistance in current flow, thermal resistance can be defined as well, which is dependent on the dimension and the material used in the chip. To contain the peak junction temperature, a circuit has to operate within its thermal design power (TDP) so that heat generation does not outpace heat dissipation in the cooling system.

High temperature has a negative impact on many aspects of a Complementary Metal Oxide Semiconductor (CMOS) circuit, such as transistor performance, power consumption and system reliability. Thermal management is not only the task of package designers but also of the chip designers. To effectively limit the high temperature in a chip equipped with a cost-effective cooling system, thermal specific approaches, besides low power techniques, are necessary at the chip design level.

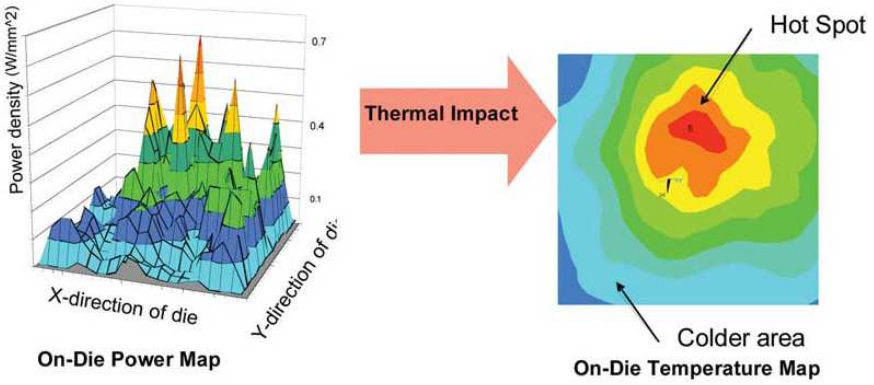


Figure 1.1: A typical power map and the corresponding thermal map (from [1]).

Figure 1.1 illustrates the correlation between the power profile and the corresponding thermal map, which schematically shows the temperature rise at different locations in the chip. The non-uniformity of power consumption can cause a much higher local power density (typically referred to as hotspots). In microprocessors, regions on the die with a temperature higher than 85 °C are usually called hotspots. Temperatures in the hotspots rise much faster than the full chip heating and can in the worst case cause severe damage to the chip.

In this work, we investigate the power and thermal management of SoCs. To

study the thermal distribution within a circuit, we implement a thermal simulator that numerically analyzes the power-temperature relationship. The thermal simulation provides valuable insights and forms the basis for characterization and optimization of thermal behaviors. We also studied the thermal impact on interconnect in terms of delay since the electrical resistance in metal is temperature dependent. The results give the perspective that wire planning at early design stages should take thermal impact into account to optimize for delay and reliability.

With the purpose of reducing power density in hotspots, we propose two placement techniques that spread cells in the hotspots over a larger area. Increasing the area occupied by the hotspot directly reduces its power density. We compare the proposed methods in terms of temperature reduction, timing and area overhead to the general method, which enlarges the circuit area uniformly.

In addition, as a case study we analyze the design of Floating Point Units (FPU) from both an energy and thermal perspective. We compare different implementations for the division operation and illustrate the impact of power efficient dividers on thermal profiles within the FPU and on-chip cache. The decrease in average temperature can reduce the power consumption in cache and any other leakage dominate circuit blocks. We provide a quantitative evaluation of the leakage and the total power reduction obtained from the decrease in temperature.

The rest of the chapters in this work is organized as follows. Chapter 2 describes power consumption and heat transfer within a CMOS VLSI circuit. Chapter 3 introduces floating-point (FP) number representations and describes algorithms and implementations for FP operations. Chapter 4 presents a thermal simulation method, which we use to analyze steady-state thermal profile within a circuit. In Chapter 4, we also describe a method to model temperature dependent wire delay during the early design stages. Chapter 5 presents two temperature reduction techniques that we propose to reduce the power density in hotspots. Chapter 6 discusses power consumption in FPUs and compares energy savings and peak temperature reductions from alternative implementations of FP division. Chapter 7 highlights some of the perspectives that can extend this work and finally Chapter 8 draws the conclusions.

Chapter 2

Power Dissipation and Heat Transfer in CMOS VLSI circuits

Power consumption has become a primary design constraint as CMOS manufacturing technology scales down to deep sub-micron geometries.

Traditionally, static CMOS circuits are very power efficient as they nearly dissipate zero power while idle. This is a compelling advantage over bipolar processes and allows much more CMOS transistors to be integrated onto a single die. Since the 1980s CMOS processes were widely adopted and are nowadays used for nearly all digital logic applications.

According to the *constant field scaling* theory [2], both the dynamic power and the area of a transistor scales to $1/S^2$ (S is the process scaling factor) when moving to a new process node. As a result, for the same die area designers can put S^2 times more transistors in a new process while maintaining the same power density.

In practice, however, power density has skyrocketed because clock frequencies have increased much faster and supply voltage remains higher than classical scaling would predict. The increasing power density results in higher junction

temperature, which affects many aspects of a semiconductor device. In high end processors, the performance is becoming increasingly limited by the maximum amount of power that can be dissipated by the cooling system without exceeding the maximum junction temperature. Increasing the volume of heat sinks or using more advanced cooling systems can permit more power dissipation. However, the cost of cooling systems is a nonlinear function of total chip power dissipation which makes using more powerful cooling system a non sustainable solution. In commodity microprocessors, a shift from high frequency single core designs to moderate frequency multi-core designs has already taken place because of the excessive cooling cost.

The emergence of new technologies can offer a one time solution to the heating problem, such as the displacement of bipolar technology by CMOS technology. However, at the moment no clear successor to the CMOS technology has emerged yet. Power consumption and power density will remain as the key design bottleneck as the degree of integration increases with each new process node. Consequently, it has become ever more important to build power efficient and thermal aware solutions at all design levels.

Many research work has been carried out in the past decades to design power efficient circuit but in recent years thermal issues quickly arose to become one of the potential show-stoppers to future CMOS scaling. In this chapter, we will first review the mechanisms of power dissipation in CMOS circuits and describe the methods to model and estimate on-chip temperature distribution. Then we will discuss about thermal trends with technology scaling and how temperature affect various properties of CMOS circuits. Finally, we briefly summarize the related work in the area of power and thermal management.

2.1 Power Dissipation in CMOS circuits

Power dissipation in CMOS circuits comes from two components [2]:

- Dynamic dissipation due to charging and discharging of load capacitance and to the short-circuit current.

- Static dissipation due to leakage current and other current drawn continuously from the power supply.

2.1.1 Dynamic Dissipation

The dominant component in dynamic power dissipation is due to the charging of the load capacitance C_L. In standard cell designs, C_L for a certain cell is the total gate capacitance on the driven cells C_g and the total capacitance contributed from the interconnecting wires to the driven cells C_w. In nanometer technologies, for long wires (longer than several tens of micrometer), C_w can dominate C_g and contribute the most to C_L in a standard cell.

$$C_L = C_g + C_w \tag{2.1}$$

The average dynamic power dissipation can be expressed as the average instantaneous energy dissipation:

$$P_{dynamic} = \frac{1}{T}\int_0^T i_{DD}(t)V_{DD}dt = \frac{V_{DD}}{T}\int_0^T i_{DD}(t)dt \tag{2.2}$$

where $i_{DD}(t)$ is the transient current drawn from the power supply and V_{DD} is the supply voltage. The integral is the total amount of charges delivered during the time interval T. During each transition, the load capacitance is charged to V_{DD} or discharged to ground and the amount of charges delivered Q is therefore equal to $C_L V_{DD}$. Assuming a nodal activity of α on the load and a clock frequency of f, the total number of transitions amounts to $\alpha T f$. Therefore, Eq. (2.2) simplifies to:

$$P_{dynamic} = \frac{V_{DD}}{T}\alpha T f C_L V_{DD} = \alpha C_L {V_{DD}}^2 f \tag{2.3}$$

αC_L is also called effective switching capacitance as it is the part of total capacitance that contributes power consumption.

Short circuit power dissipation occurs when both pull-up and pull-down network in a CMOS gate are partially ON during the input signal transition (as illustrated in Figure 2.1). This results in a current pulse directly from V_{DD} to GND in a short period of time. The short circuit power depends on the transition time, the peak current and the supply voltage. However, the derivation of an exact formula for the short circuit power is not easy and by making simplifying assumptions closed-form expressions have been proposed in [3, 4, 5]. To reduce short circuit power dissipation, it is desirable to make the edge slope of transition as sharp as possible, e.g. by increasing the size of the driving gate.

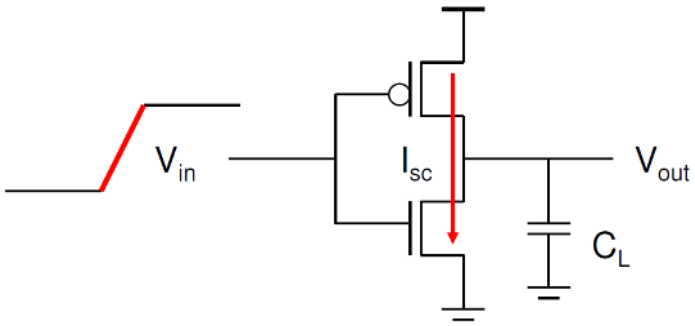


Figure 2.1: Short circuit power dissipation in an inverter.

2.1.2 Static Dissipation

Static dissipation is caused by secondary effects (as illustrated in Figure 2.2) such as sub-threshold conduction, gate oxide tunneling, reversed biased diode leakage, etc. which lead to small amounts of current flowing through an OFF transistor. In 130 nm processes and beyond, the static power (a.k.a leakage power) has rapidly become a primary design issue which increases exponentially as process moves to finer technologies. This is primarily due to the scaling of threshold voltage V_{th}[1] resulting in orders of magnitudes increase in transistor's leakage current.

According to the International Technology Roadmap for Semiconductors (ITRS) [6], various low power techniques such as dynamic V_{th}, multiple V_{th} transistors, power domains/voltage islands, body bias, etc. will mitigate leakage until 2012. As the use of high-κ dielectric[2] brings gate leakage under control, sub-threshold leakage is going to dominate and limit performance. Figure 2.3 shows the ITRS prediction of leakage in 2007 for the next decade.

The sub-threshold leakage current density, i.e. the current per unit transistor area, can be expressed by [7]:

$$J_{sub} = \frac{W}{L_{eff}}\mu\sqrt{\frac{q\epsilon_{si}N_{cheff}}{2\phi_s}}v_T^2 \exp\left(\frac{V_{gs}-V_{th}}{\eta v_T}\right)\left[1-\exp\left(\frac{-V_{ds}}{v_T}\right)\right] \qquad (2.4)$$

[1] Threshold voltage is the voltage at which there are sufficient carriers to make a conducting path between source and drain of a transistor.

[2] The silicon dioxide gate dielectric is replaced by a material with a higher dielectric constant κ (such as hafnium-based materials), which allows further device miniaturization.

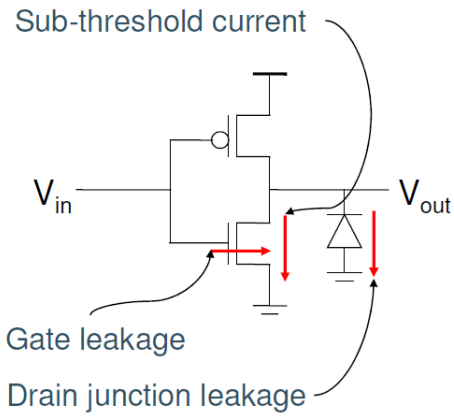


Figure 2.2: Static power dissipation in an inverter.

where W is the width of the transistor, L_{eff} is the effective channel length, N_{cheff} is the effective channel doping, ϕ_s is the surface potential, η is the sub-threshold swing, and v_T is the thermal voltage. v_T is given by kT/q where k is the Boltzmann's constant, q is the electrical charge, and T is the junction temperature.

From Eq. (2.4), we can see that sub-threshold leakage current is an exponential function of the junction temperature T. At high operating temperatures, static dissipation can contribute a significant amount to the total power dissipation in CMOS circuits. In cache and other circuit components, where signal switching only occurs to a small portion of transistors at a time, static power can dominate over dynamic power. Reports indicate that 40% or even a higher percentage of the total power consumption in 90 nm process technology is due to leakage [8] and this percentage is expected to increase with technology scaling.

2.2 Design for Low Power

Design for low power has been one of the main research subjects in VLSI design for the past decades. Many proposed low power techniques are quite effective and over the years major library providers and Electronic Design Automation (EDA) tool vendors have integrated these techniques into their products. Standards for power intent specification such as Unified Power Format [9] (UPF) and

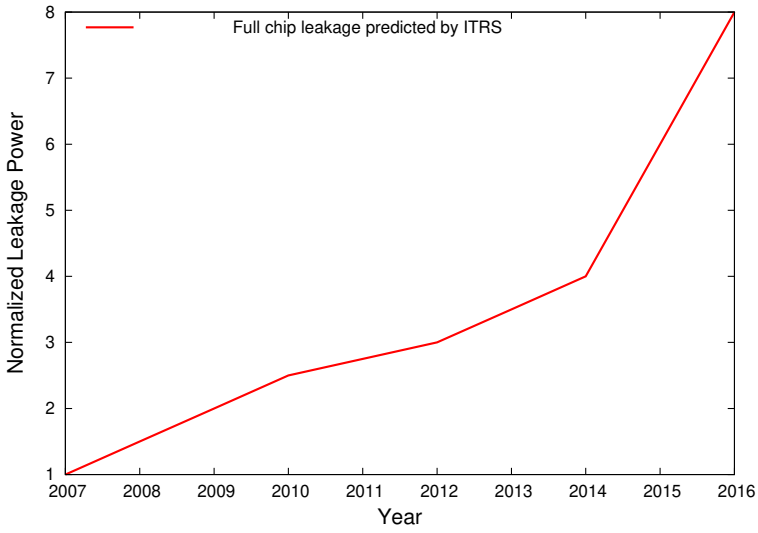


Figure 2.3: Full-chip leakage normalized to leakage power in 2007.

Common Power Format [10] (CPF) allow the specification of implementation-relevant power information in power aware design flows. Some of these techniques are general in nature and can be applied to most situations. Others require circuit designers to make decisions based on careful analysis of the various interacting factors. Low power techniques can be divided into those that reduce dynamic power and those that reduce static power.

2.2.1 Dynamic Power Reduction

The dynamic dissipation is governed by Eq. (2.3) and is reduced by decreasing the effective switching capacitance, the power supply or the operating frequency.

Clock gating [11, 12] is an effective method in reducing switching capacitance, which disables any changes to input registers of the idle part of a circuit. Clock gating can be applied at different granularities. Fine grained clock gating controls a small number of registers or even a single register and the area overhead due to extra control hardware can be high. On the other hand, coarse grained clock gating is applied to a large number of registers, probably also to portions of the clock tree since the clock tree itself consumes a significant amount of power due to the large activity factor. The disadvantage of coarse grained clock

gating is that it can only be applied when all the gated registers are idle, which can be infrequent in general.

Synthesis tools can also perform many power reduction techniques during logic synthesis such as operand isolation, logic restructuring and resizing, pin swapping, etc. to minimize the effective switching capacitance [13].

Intuitively, the most effective way of optimizing dynamic power would be to decrease the supply voltage V_{DD} as the reduction can be quadratic. A lower supply voltage can be used for cells not in the critical path without compromising a circuit's performance [14]. At the chip level, different blocks can have different speed requirements (e.g. peripheral interfacing circuitry can operate at a slower speed than the data processing circuitry) allowing effective power reduction through multiple supply voltages with no performance overhead. However, multivoltage design does introduce complexities especially for layout. Additional pins are required to connect all the supply voltages to the chip. The power grid has to distribute each of the power supplies separately to the appropriate blocks. The interfacing between different power domains has to ensure the correct signal levels by using level shifters.

Another technique that involves lowering supply voltage is dynamic voltage scaling (DVS) [15], which varies the speed of a circuit dynamically according to the timing requirement. A similar technique is dynamic frequency scaling (DFS) [16], which adjusts the clock frequency in order to save power. DVS and DFS are combined in some designs into dynamic voltage and frequency scaling (DVFS) [17].DVS and DFS can both be implemented either in hardware through circuit delay monitoring and matching or in software through operating system scheduling.

2.2.2 Static Power Reduction

The static dissipation, unlike dynamic dissipation, is very much dependent on the manufacturing process (e.g. channel length, doping profile, etc.) and the operating environment (e.g. temperature, supply voltage, etc.). As can be seen from Eq. (2.4), at the circuit design level, the static dissipation can be reduced by increasing the threshold voltage V_{th} or decreasing the gate to source voltage V_{gs} (decreasing the drain to source voltage V_{ds} is impractical since static CMOS swings from rail to rail).

Transistor threshold voltage V_{th} is dependent on several parameters such as body doping profile, gate oxide thickness, body voltage, temperature, etc. Standard cell libraries usually offer more than one cells to implement each logic function,

each using a different transistor threshold voltage. Cells with a low threshold voltage (LVT cells) have a faster transition time but leak more sub-threshold current. Cells with a high threshold voltage (HVT cells) have less leakage but also a slower speed. Thus, leakage optimization can proceed in such a way that LVT cells are used sparingly in timing critical paths and HVT cells are used elsewhere to save standby power [18].

Alternatively, the threshold voltage can be controlled at runtime through applying a variable biasing voltage to the body from a control circuitry [19]. Reverse body bias (RBB) can be applied to increase the threshold voltage and thus reduce leakage when the circuit is in standby mode. During active mode, forward body bias (FBB) can be applied to speed up the circuit to match timing requirement. In this way, the circuit can be seen as implemented in transistors with adaptable or variable threshold voltages. In fact, the technique is also known as variable threshold CMOS (VTCMOS). The advantage of adaptive body biasing is that dynamicly tuning the threshold voltage allows leakage reduction also at runtime. However, the technique requires a triple well process to achieve different body bias levels at different regions of the chip, which is quite expensive.

Another technique to reduce idle time leakage is power gating, also known as multiple threshold CMOS (MTCMOS), which cuts off power supply entirely to the idle part of a circuit [20]. Power gating uses low V_{th} transistors for computation and high V_{th} transistor as switches to disconnect the power supply during idle mode. Power gating is suitable for runtime behaviors with quite long idle periods (e.g. millions of clock cycles) and is very effective in reducing leakage power since the path from power supply to the ground is "cut-off".

Figure 2.4 illustrates a power gating design, where the controller controls the sequences when switching between power ON and power OFF mode. The power gated block is disconnected from supply rails by the power switching fabric. The isolation block insures any signal connected to the always-on block does not stay at intermediate signal levels when the power gated block is in OFF mode.

The technique also imposes several requirements to the design of standard cell libraries and physical implementation tools. For example, isolation cells and always-on cells have to be provided by the library in order to ensure signal integrity on the interface with non power gated domains. Any useful state information in memory elements (e.g. registers) has to be preserved during power down and correctly restored upon power on, which is usually accomplished by using state retention registers. State retention register is embedded with an always on low leakage shadow register that keeps the content of the main register during power gating. At the physical implementation stage, cells in the same power gating domain can be clustered in the layout as much as possible in order to minimize the number of switch cells, which are usually large area. Power

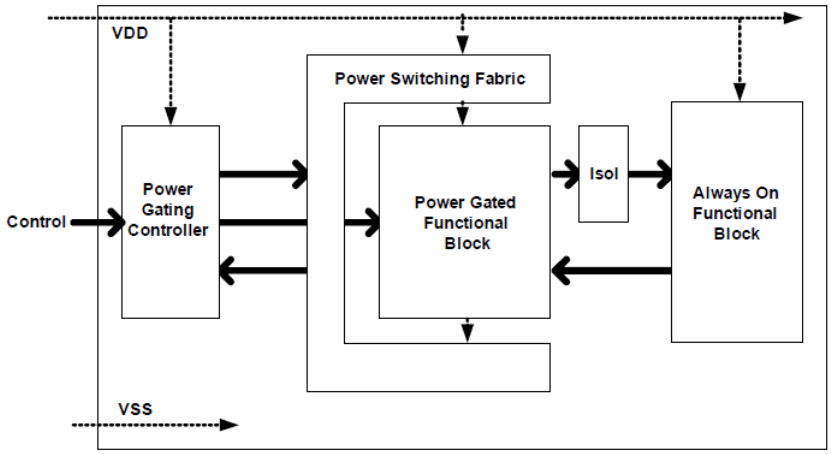


Figure 2.4: Block diagram of a SoC with Power Gating (from [21]).

gating also introduces virtual supply rails and complicates the design of power networks.

2.3 Heat Transfer and Distribution

The dissipated power in a VLSI circuit is manifested in the form of heat. In the context of a MOSFET, when a signal transition occurs, the applied voltage leads to a lateral electric field, which accelerates the charge carriers to move from source to drain. The charge carriers that gained kinetic energy release part of the energy whenever colliding with other carriers and atoms causing vibrations of these particles and consequently a rise in the temperature. In this way, electrical energy due to power consumption in CMOS circuit is transformed to heat.

Two sources of heat generation exist in a CMOS circuit: cells and interconnects. Power consumption in cells is composed of dynamic and static dissipation as described in the previous section. Power consumption in interconnects is caused by the flow of a small amount of current that charge and discharge the parasitic capacitance in the interconnect during signal transition.

The small amount of current flow can result in significant temperature rise in

metal wires, which is widely known as self-heating, due to the low-κ dielectric materials used in modern processes. However, self-heating is less related to substrate temperature distribution than reliability issues in metal wires such as electromigration. Due to the much smaller electrical resistance in interconnects, the major source of heat generation is the power dissipation in cells on the device layer.

For a flip-chip design[3], most of the heat generated from the transistor junctions is dissipated to the ambient environment through the heat sink attached to the back side of the substrate, which constitutes the primary heat transfer path. A small amount of heat is also conducted through interconnect layers and pads to the packaging and printed circuit boards (PCB). Figure 2.5 shows a typical flip chip design with a cross-sectional view of the PCB, packaging and a heat sink. The heat sink is attached to the backside of the substrate through thermal interface materials (TIM) and the wire leads are connected to the PCB through C4 and CBGA Joint. The system is modeled as a network of thermal resistors which shows the major heat dissipation paths. The heat is mainly transferred through conduction within the system and through radiation and convection to the ambient.

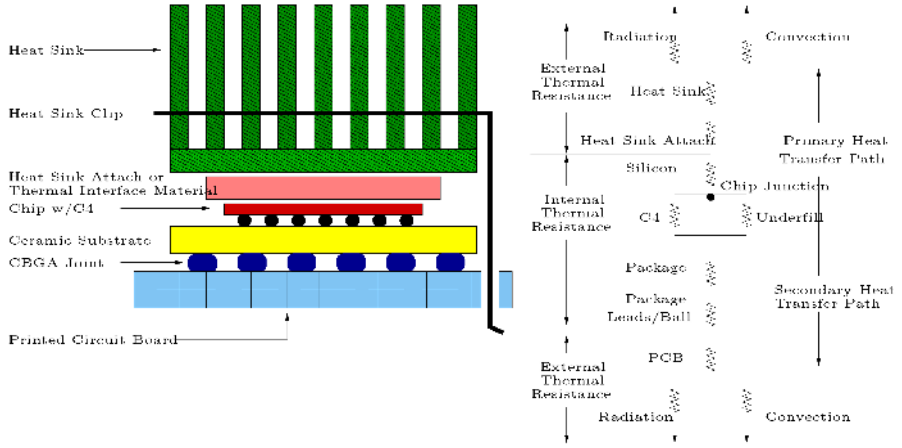


Figure 2.5: Heat dissipation paths of a chip (from [22]).

To describe the relationship between power consumption and junction temper-

[3]Flip-chip, also called Controlled Collapse Chip Connection (C4), is a method where the surface of the chip is covered with an array of pads on the top of metal. The chip is flipped upside down and in nearly direct contact with the package, eliminating the inductance associated with the bond wires [2].

ature, a first order expression can be expressed as:

$$T_j = T_a + P_{chip} \times R_{ja} \tag{2.5}$$

where T_a is the temperature of ambient environment (e.g. temperature of air in side the chassis of a desktop computer), P_{chip} is the total power consumption inside the chip, R_{ja} is the junction to ambient thermal resistance and T_j is the derived junction temperature. As illustrated in Figure 2.5, R_{ja} can be modeled as the series resistance from junction to ambient in different parts of the chip. Analogous to electrical resistance, thermal resistance can be defined as:

$$R = \frac{L}{k \times A} \tag{2.6}$$

where k is the material's thermal conductivity, L is the length and A is the cross-sectional area of the conducting path.

Substituting Eq. (2.6) into Eq. (2.5), we get:

$$T_j = T_a + P_{chip} \times \frac{L}{k \times A} \tag{2.7}$$

To a circuit designer, L and k are process and package dependent parameters and T_a is determined by the operating environment. Consequently, T_j is closely related to the ratio between P_{chip} and A which defines the power density of the chip.

Due to the large variation of power density in different regions on the die, significant temperature difference between different regions (typically referred to as thermal gradients) usually exists on the substrate layer. The one dimensional first order expression in Eq. (2.5) is incapable to capture the temperature difference within the chip.

In general, the heat diffusion equation is used to describe the rate of heat conduction in a chip:

$$q = -k_t \nabla T \tag{2.8}$$

which states that heat flux, q (in W/m^2), is proportional to the negative gradient of temperature, T (in K), with the constant of proportionality corresponding to the thermal conductivity of the material, k_t (in $W/(mK)$). k_t is in general temperature dependent, however the variation is not significant. For on chip thermal analysis, k_t can be treated as constant for each type of material.

The divergence of q is the difference between the power generated and the time rate of change of heat as described in Eq. (2.9),

$$\nabla \cdot q = -k_t \nabla \cdot \nabla T = -k_t \nabla^2 T = g(\mathbf{r}, t) - \rho c_p \frac{\partial T(\mathbf{r}, t)}{\partial t} \tag{2.9}$$

where $\mathbf{r}$ is the spatial coordinate of the point at which the temperature is being determined, g is the power density of the heat source (in W/m^3), c_p is the heat capacity of the material (in $J/(kgK)$) and ρ is the density of the material (in kg/m^3). This heat diffusion equation(Eq. (2.9)) can be rewritten as,

$$\rho c_p \frac{\partial T(\mathbf{r},t)}{\partial t} = k_t \nabla^2 T + g(\mathbf{r},t) \tag{2.10}$$

Eq. (2.9) is subject to the boundary condition,

$$-k_t \frac{\partial T(\mathbf{r},t)}{\partial n} = h_c (T_{\mathbf{r},t} - T_a) \tag{2.11}$$

which states that the heat generated inside the chip equals the heat dissipated to the ambient.

The time constant of on-chip heat conduction is in milliseconds, which is much larger than the clock periods in nanoseconds. This means transient currents with short time constants do not have significant impact on temperature distribution. If the power profile of a circuit does not change within an extended period of time, steady-state analysis can capture the thermal behavior of the circuit accurately [22]. Steady state thermal analysis determines temperature distribution when power density distribution does not change over time, which is sufficient for applications with stable power profiles or periodically changing power profiles that cycle quickly [23]. Thus to obtain steady state temperature profile, single average power for different location of the circuit can be used.

Eq. (2.10) can be solved numerically using one of several solution frameworks for partial differential equations. The Finite Difference Method (FDM) and the Finite Element Method (FEM) both discretize the chip and form a system of linear equations relating the temperature distribution to the power distribution. Many research work based their optimization on thermal simulators using the FDM or the FEM methods [24, 25, 26, 27]. The work in [28] proposed a compact thermal model for architectural thermal analysis, which based on the electrical-thermal analogy creates a coarse grained RC network of functional blocks and the package. The resulting matrix equations are then solved using LU decomposition to obtain a steady-state thermal analysis.

For full chip thermal analysis, in order to achieve high resolution the number of linear equations resulted from the FDM discretization can be huge. The multigrid algorithm was successfully used in [23, 29] to solve the problem and shown to be more efficient than other methods such as Gauss-Seidel iteration, conjugate gradient method, etc. The multigrid method follows a hierarchical approach and is based on the observation that an iterative solver is usually

more effective in removing high frequency solution errors in a FDM mesh than low frequency errors. Therefore, a hierarchy of mesh grids with multilevel granularities are constructed. The method starts with iterating over the finest grid and once the convergence deteriorates due to low frequency errors it changes to the coarser grid where the low frequency errors are removed through coarse grid correction. Once the solution to the coarser grid is obtained, it is mapped back to the finer grid through interpolation. The overall runtime is observed to be linear with the number of nodes in the finest mesh grid.

An alternative to the FDM and FEM methods is a boundary element method using Green functions [30, 31]. In this method, only layers of heat sources or layers of thermally interest are analyzed, thus resulting in a smaller problem size than FDM and FEM. However, the method is based on the assumption that chip materials are layer-wise uniform and consequently is more suitable for early stage analysis where efficiency is more in favor than accuracy. An improved method in [32] reduces the complexity in Green function based methods by recognizing the bottleneck corresponds to a convolution operation.

2.4 Technology Scaling and Thermal Issues

Eq. (2.5) shows that junction temperature depends on power consumption and thermal resistance. With manufacturing process scaled to finer geometries, power consumption in a single transistor decreases. Thermal resistance for a single transistor, on the other hand, increases due to the reduction in the transistor's geometrical size [33]. Transistor's temperature in a new process is thus dependent on the relative rate of changes of the two parameters.

To estimate full chip temperature increase in a new process, one also has to take into account the increase in transistor density. In [22], the authors using industrial technology data and ITRS prediction for future technologies showed that the normalized temperature increase of a chip is significantly elevated when CMOS technology scaled from 350 nm to 90 nm. The estimated junction temperature of a 90 nm process CMOS chip is about 4.5 times higher than that of a 350 nm process CMOS chip.

The rapid increasing junction temperature can affect several aspects of circuit design as many CMOS circuit parameters are temperature dependent. The carriers mobility of a transistor decreases with increasing temperature which lowers the drive current and leads to increased delays. On the contrary, transistor's threshold voltage decreases with temperature which improves transistor switching time. The performance of a transistor is therefore dependent on which of the

two factors dominate. The unit resistance of a wire segment increases with temperature, which makes the delay on wires especially global wires very sensitive to high temperature [34].

The unevenly distributed heat caused by large spatial variations in power consumption at different locations can make performance analysis difficult. Thermally induced device mismatch is a major concern in high speed and high precision IC design such as clock distribution networks, Arithmetic Logic Units (ALU), data converters, amplifiers, etc. Containing temperature and thermal gradient is also critical to the design of mixed signal and analog ICs as they are more sensitive to temperature.

Sub-threshold leakage, as described in Section 2.1 has an exponential dependence on temperature [7]. It has been shown in [35] that for every 30 °C increase in temperature, the amount of leakage more than doubles. The induced leakage in turn increases total power consumption and causes further temperature rise. If the cooling system is inadequate to remove the generated heat fast enough, the positive feedback loop between temperature and leakage will eventually cause thermal runaway and burn down the chip.

Temperature is a vital factor in microelectronics system's reliability [36]. Higher junction temperature reduces mean time to failure (MTTF) for the devices, which has a direct impact on the overall system reliability. It is reported in [37] that a small increase in operating temperature ($10 - 15\,^{\circ}\mathrm{C}$) can decrease the lifespan of devices by 2 times. Many physical effects that cause reliability degradation are thermally activated processes [38, 39, 40]. Negative Biased Temperature Instability (NBTI) and Hot Carrier Injection (HCI) effects, which are strongly dependent on temperature, degrade the performance of transistors in an irreversible manner over time. These effects reduce a circuit's lifetime and cause timing violations eventually. Other failure mechanisms such as electromigration, stress migration and dielectric breakdown are accelerated by high temperature and temperature gradients and cause permanent device failures [41]. According to ITRS [6], the junction temperature of a semiconductor device must be kept at $85\,^{\circ}\mathrm{C}$ or lower to ensure long term reliability.

All these factors dictate that the excessive heat generated from the circuit must be dissipated to the ambient at a reasonable speed and the circuit should be operated within a specified temperature range.

2.5 Thermal Management Techniques

Although the elevated temperature is caused by the increasing power consumption in a chip, effective power management methods may not be as useful for thermal management. The key differences between power reduction techniques and temperature reduction techniques can be classified into several aspects.

First, power dissipation is in general spatially nonuniform across a chip and thus temperature in local hotspots can increase much faster than full chip heating. If the high power density in hotspots is effectively reduced, a chip can dissipate the same amount or even more total power at a lower peak temperature. Power management performs a *minsum* optimization, which monitors full chip power consumption and attempts to lower the total energy consumed over an entire application run. In contrast, thermal management is a *minmax* optimization problem, where peak temperature at specific localized hotspots and full chip thermal gradient are of main concern.

Second, heat distribution within a circuit evolves over timescales of hundreds of microseconds while power dissipation changes every clock cycle, which is in nanoseconds. Consequently, low power techniques that reduce power dissipation at a very short timing granularity do not have effect in reducing temperature.

Although a vast literature on techniques to reduce power does exist, as we have seen in Section 2.2, not all low-power design solutions are effective for reducing temperature. For this reason, recent research has focused on specific solutions for *thermal management* ([42, 43, 44, 45, 46]), in which temperature and not power is the actual metric.

To perform thermal management within a circuit, techniques explicitly targeting the spatio-temporal thermal behavior are needed. Thermal management can be divided into design time and run time techniques based on the timing of activation.

2.5.1 Design Time Solutions

Design time thermal management involves the spatial arrangement of circuit structures in such a way that the maximum local power density is reduced. These structures are functional blocks at the chip level and individual gates at the block level. Placing high power consumption structures away from each other can reduce the thermal coupling and flatten out the die's thermal profile.

In [47], an *isothermal logic partitioning technique* was proposed. This method iteratively optimizes the thermal profile of a placed netlist by building isothermal logical clusters and then partitioning the hottest clusters into two parts. In this way, hotspots are split apart and can be placed close to cool cells. For standard cell based synthetic benchmarks, the algorithm achieved on average 5.54% and 9.9% in peak temperature reduction with a timing overhead of 5% and 10% respectively.

Many thermal aware floorplanning and placement algorithms were also proposed, typically using peak temperature as one component of the cost function when evaluating the quality of a candidate solution. In [48], a system level leakage aware floorplanner (LEAF) was proposed. The method also models the positive feedback loop between temperature and leakage in a simulated annealing based floorplanning algorithm. For each type of transistor in the library, a temperature leakage correlation table is generated from SPICE simulation. Within the annealing, an initial estimate of the leakage is used to determine the temperature, which in turn updates the leakage. The feedback loop between the leakage and the temperature continues until steady-state temperature is reached. The new floorplan is evaluated in terms of total leakage power and other metrics such as area and total wirelength.

In recent years, temperature variation induced clock skew in the clock distribution network has become prominent. In [49, 50], the authors described design time clock tree synthesis algorithms to take counter measures against nonuniform substrate thermal profile. While in [51], the optimal insertion of tunable delay buffers into clock trees, to adjust at run time the delay of clock distribution paths that are more susceptible to temperature variations, is discussed. Thermal aware global routing algorithms for improving reliability are also proposed in [52, 53].

The advantage of design time techniques is that they are applied during the physical implementation stage and the incurred performance overhead can usually be optimized. On the other hand, the solution is static and does not adapt to changes in thermal behaviors such as the shifting of locations of hotspots at run time.

2.5.2 Run Time Solutions

Run time thermal management is also called dynamic thermal management (DTM) as they are performed dynamically when applications are running. DTM monitors chip temperature through thermal sensors and triggers response mechanisms. Reactive DTM activates responsive mechanisms once the peak temper-

ature or thermal gradient exceeds predefined thresholds, while the timing of activation in proactive DTM is based on predictions of future temperatures. Response mechanisms include dynamic power reduction techniques like DVFS, power gating and clock gating as described in Section 2.2, and architectural adaptation methods like clock throttling, limiting the issue width in multiple issue processors, task migration in multicore systems, etc.

In [54, 55, 56], the authors studied the reductions in hotspots and spatial-temporal thermal gradients using static and dynamic task scheduling methods. The DTM schemes considered in [55] are thread migration and voltage scaling and the proposed scheduler is compared against load balancing schedulers, which are typically found in multi-core operating systems. To avoid significant performance degradation a probabilistic policy (*Adaptive-Random*) determines between load balancing and DTM schemes. The results showed that the proposed methods can effectively reduce hotspots with a performance overhead between 2.4% and 15.0% for different approaches. In [56], the authors proposed a proactive thermal management approach, which can estimate future temperature based on a moving window of temperature history. The robustness and accuracy of the method are achieved through adapting the model parameters according to the dynamic workload and temperature measurements at runtime. In [57], the authors proposed a thermal balancing policy specifically designed for multiprocessor stream computing applications. The experiment results showed that the proposed policy achieved thermal balance between cores at a performance cost less than general DTM schemes such as DVFS, Stop&Go, etc.

The advantage of run time thermal management is that it can adapt to dynamic thermal behaviors and is suitable for applications where the workload on blocks varies overtime as in the case of a microprocessor. The disadvantage is that these techniques are usually complex to implement and the performance overhead due to the triggering of DTM can be significant. For example, DVS requires to stall between 10 - 50 μs for the resynchronization of the clock's phase-locked loop [58]. Furthermore, aggressive DTM might introduce additional thermal stress-relax cycles that cause mechanical stress and impact system reliability and thus requires careful analysis.

Chapter 3

Floating Point Units

Floating-point units (FPU) could be a good case study for power and thermal aware design. FPUs are found in a wide variety of processors ranging from server and desktop microprocessors, graphic processing units (GPU), to digital signal processors (DSPs), mobile Internet devices and embedded systems. In NVIDIA's latest CUDA architecture Fermi, each streaming multiprocessor contains 32 FPUs totaling 512 of them in a single GPU. Floating-point operations are much more complex than their integer and fixed-point counterparts. Consequently, FPUs usually occupy a significant amount of silicon area and can consume a large fraction of power and energy in a chip. For scientific and graphics intensive applications, the high power consumption in FPU can make it the hotspot on the die.

In this chapter, we give a brief overview of the floating-point representation and the arithmetic operations on floating-point numbers, namely add/subtract, multiply, fused multiply-add and divide. For each operation, we illustrate the basic steps necessary to perform the operation in hardware and an example implementation. In particular, the division operation is implemented using several alternative algorithms. In Chapter 6, we compare the different implementations of division in terms of power and energy consumption and discuss FPU design from an energy and a thermal perspective.

3.1 Floating-Point Representation

A floating-point representation is used to represent real numbers in a finite number of bits. Since the set of real numbers is infinite, it is only possible to exactly represent a subset of real numbers in the floating-point representation. The rest of the real numbers either fall outside the range of representation (overflow or underflow) or are approximated by other floating-point numbers (roundoff). The most used representation is sign-and-magnitude, in which case a floating-point number x is represented by a triple (S_x, M_x, E_x):

$$x = (-1)^{S_x} \times M_x \times b^{E_x} \tag{3.1}$$

where $S_x \in \{0, 1\}$ is the sign, M_x denotes the magnitude of the significand, b is a constant called the base and E_x is the exponent.

A floating-point representation system involves many parameters and historically many floating-point processors were designed using a variety of representation systems. To avoid incompatibilities between different systems, the IEEE Floating-point Standard 754 was developed, which is followed by most floating-point processors today. The latest version of the standard (IEEE 754-2008 [59]) defines the arithmetic formats of binary and decimal floating-point numbers as well as arithmetic and other operations that perform on these numbers. We briefly summarize the main components of the IEEE Standard 754 for binary numbers in this section.

3.1.1 Formats

The magnitude of the significand M_x is represented in radix 2 normalized form with one integer bit:

$$1.F$$

where F is called the fraction and the leading 1 is called the hidden bit. The exponent E_x is base 2 and in biased representation with

$$B = 2^{e-1} - 1$$

where e is the number of bits of the exponent field. Denormalized numbers ($E_x = 0 \wedge F \neq 0$) do not have a leadinig 1 in the hidden bit. Consequently, the value of a normal floating-point number represented in the IEEE format can be obtained as:

$$x = (-1)^{S_x} \times 1.F_x \times b^{E_x - B} \tag{3.2}$$

The three components are packed into one word with the order of the fields in S, E and F. The system defines three basic binary floating-point formats:

- binary16 (Half): S(1), E(5), F(10).
- binary32 (Single): S(1), E(8), F(23).
- binary64 (Double): S(1), E(11), F(52).

3.1.2 Rounding

The standard defines five rounding algorithms:

- Round to nearest: Round to nearest, ties to even (default); Round to nearest, ties away from zero.
- Directed: Round toward 0 (truncated); Round toward $+\infty$; Round toward $-\infty$.

3.1.3 Operations

Required operations include:

- Numerical: add, subtract, multiply, divide, remainder, square root, fused multiply-add, etc.
- Conversions: floating to integer, binary to decimal (integer), binary to decimal (floating), etc.
- Miscellaneous: change formats, test and set condition flags, etc.

3.1.4 Exceptions

The standard defines five exceptions, each of which sets a corresponding status flag when raised and by default the computation continues.

- overflow (result is too large to be represented).
- underflow (result is too small to be represented).
- division by zero.

- inexact result (result is not an exact floating-point number).
- invalid operation (when a Not-A-Number result is produced).

In the following sections, we describe the algorithm and implementation for floating-point operations. In specific, the operations described are: add/subtract, multiply, fused multiply-add and divide. Of all these operations, division is more complex and we will present several algorithms and implementations for the division operation.

For each operation, we first present a high level description of the steps to be performed in generic form. We assume the operands and results are represented in the triple (S, M, E) as described in the previous section. To simplify the description of algorithms, let $M^* = (-1)^S M$ represent the signed significand. A hardware implementation of the operation is then given to illustrate the execution of different algorithms.

3.2 Floating-Point Addition

The addition/subtraction is described in the following expression:

$$z = x \pm y$$

The high level description of this operation is composed of the following steps:

1. Add/subtract significands and set exponent.

$$M_z^* = \begin{cases} (M_x^* \pm (M_y^* \times b^{(E_y - E_x)})) \times b^{E_x} & if\ E_x \geq E_y \\ ((M_x^* \times b^{(E_x - E_y)}) \pm M_y^*) \times b^{E_y} & if\ E_x < E_y \end{cases}$$

$$E_z = max(E_x, E_y)$$

2. Normalize significand and update exponent.
3. Round, normalize and adjust exponent.
4. Set flags for special cases.

A single path implementation of the floating-point add operation is shown in Figure 3.1 from [60], where a more detailed description of the unit is given. To

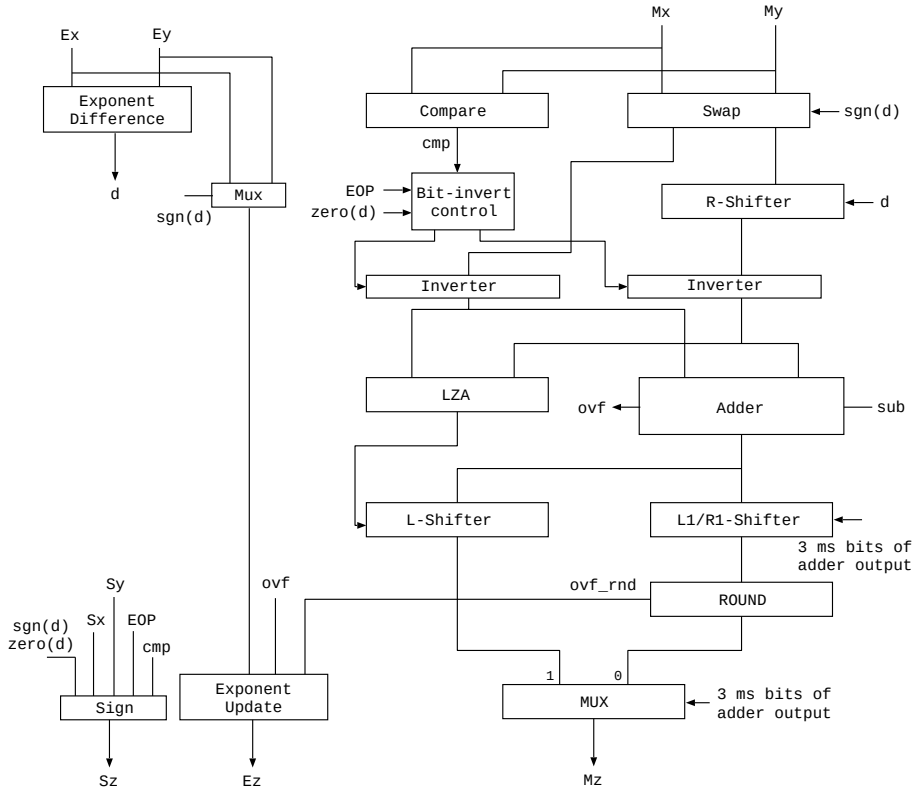


Figure 3.1: Single path floating-point addition.

avoid having two alignment shifters, the operands are swapped according to the sign of the exponent difference. A two's complement adder performs the sign-and-magnitude addition in step 1. When the effective operation is subtraction (determined by the operation and the signs of the operands), the smaller operand is complemented by bit-inversion plus carry-in to the adder. This is to avoid complementing the output of the adder when the result is negative. The leading zero anticipation (LZA) unit determines the position of the leading one in the result in parallel with the addition.

In the normalization step, two cases can occur. In the first case, the effective operation is subtraction and the output of the adder has many leading zeros, which requires a massive left shift of the result and no roundup is necessary since the exponents difference is less than 2 and no initial massive right shift was performed. In the second case, the output of the adder contains only one leading zero or has an overflow due to addition. In this case, a shifting of only one position to the left or to the right is required and subsequently a roundup is necessary. The two cases can be designed into separate paths in order to reduce the latency in both paths [61].

3.3 Floating-Point Multiplication

The multiplication of two floating-point numbers x and y is defined as:

$$z = x \times y$$

The high level description of this operation is composed of the following steps:

1. Multiply significands and add exponents.

$$M_z^* = M_x^* \times M_y^*$$

$$E_z = E_x + E_y + B$$

2. Normalize M_z^* and update exponent.
3. Round.
4. Determine exception flags and special values.

The basic implementation of floating-point multiplication is shown in Figure 3.2. For the sake of simplicity, we only show the data paths for the significands in

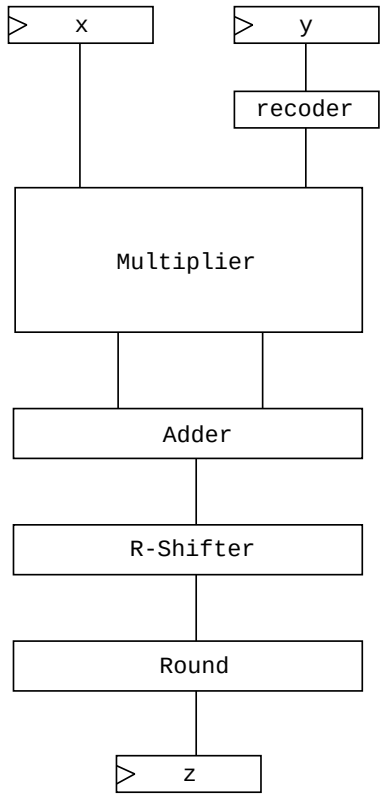


Figure 3.2: Implementation of floating-point multiplication (significands only).

block diagrams. Parallel multiplication (combinational) is a three steps computation [60]. We indicate with

$$z = x \times y$$

the product z ($n + m$ bits) of a n-bit operand x and a m-bit operand y.

1. First, m partial products

   $$z_i = 2^i x \cdot y_i \qquad i = 0, \ldots, m-1$$

   are generated. Because $y_i = \{0, 1\}$, this step can be realized with a $n \times m$ array of AND-2 gates[1]

2. Then, the m partial products are reduced to 2 by an adder tree

   $$\sum_{i=0}^{m-1} 2^i x \cdot y_i = z_s + z_c \ .$$

3. Finally, the carry-save product z_s, z_c is assimilated by a carry-propagate adder (CPA).

   $$z = z_s + z_c \ .$$

The delay in the adder tree and its area depend on the number of addends to be reduced ($m : 2$). By radix-4 recoding the multiplier y, often referred as Booth's recoding, the number of partial products is halved $\frac{m}{2}$. As, a consequence the multiplier's adder tree is smaller and faster. However, in terms of delay, the reduction in the adder tree is offset by a slower partial product generation, due to the recoding [60]. On the other hand, the reduction in area is significant, and the power dissipation is reduced as well due to both the reduced capacitance (area) and the nodes' activity because sequences of 1's are recoded into sequences of 0's resulting in less transitions.

The significand of the product might have an overflow in which case it is necessary to shift the result one position to the right and increment the exponent. Finally, rounding is performed according to the specified mode.

3.4 Floating-Point Fused Multiply-Add

The fused multiply-add (FMA) operation is a three operand operation defined by the following expression:

$$z = a + b \times c$$

[1] Shifting (2^i) is done by hard-wiring the AND-2 array's output bits.

The high level description of this operation is composed of the following steps:

1. Multiply significands M_b^* and M_c^*, add exponents E_b and E_c, and determine the amount of alignment shift of a.
2. Add the product of $M_b^* \times M_c^*$ and the aligned M_a^*.
3. Normalize the adder output and update the result exponent.
4. Round.
5. Determine exception flags and special values.

The multiply-add operation is fundamental in many scientific and engineering applications. Many commercial processors include a FMA unit in the floating-point unit to perform double precision floating point fused multiply-add operation as a single instruction. The main advantages of the fused implementation over the separate implementation of multiplication and addition are:

- The high occurrence of expressions of that type in scientific computation, and the consequent reduction in overhead to adjust the operands from the IEEE format to the machine internal representation (de-normalization, etc.).
- Improvement in precision, as the result of multiplication is added in full precision and the rounding is performed on $a + b \times c$.

The drawback is that if a large percentage of multiply and add cannot be fused, the overhead in delay and power is large especially for addition.

The architecture of an FMA unit for binary64 (double precision) significands, shown in Figure 3.3, is derived from the basic scheme in [60] and [62]. Registers A, B and C contain the input operands and register Z contains the final result. To prevent shifting both a and the product of b and c, a is initially positioned two bits to the left of the most significant bit (MSB) of $b \times c$ so that only a right shift is needed to align a and the product. The zero bits inserted in the two least-significant (LS) positions are used as the guard and round bits when the result significand is a. The amount of shift depends on the difference between the exponents of a and $b \times c$. Moreover, a is conditionally inverted when the effective operation is subtraction.

A Booth encoded tree multiplier computes the product of b and c and the result is output in carry-save format to be added with the shifted a. Since the product

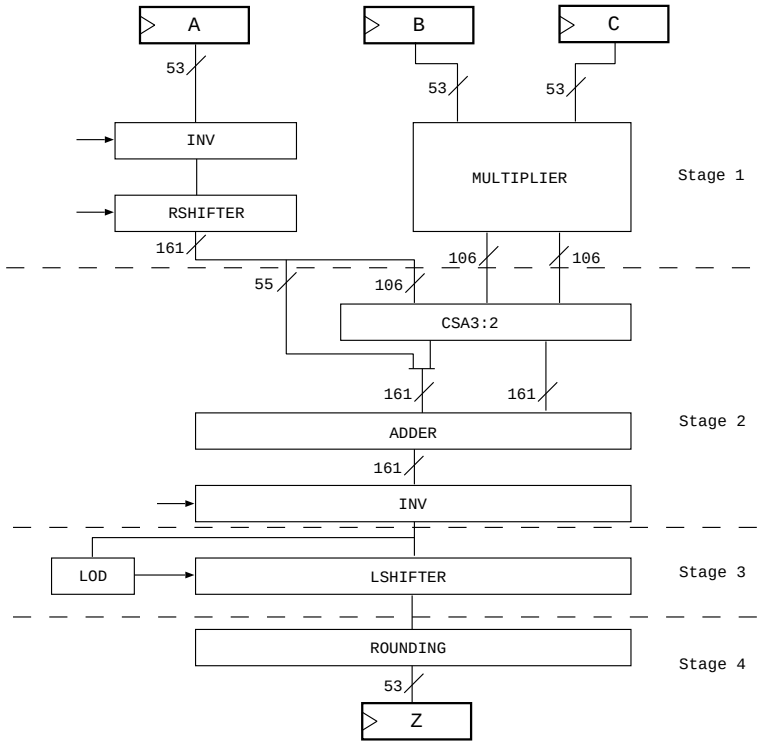


Figure 3.3: Scheme of an FMA unit (significands only).

has 106 bits, only the 106 LSBs of the shifted a are needed in the carry-save adder (CSA). The 55 MSBs of the shifted a are concatenated with the sum of the CSA to form input to the adder. Since the carry in the output of the CSA has 106 bits, only one of the input to the adder has 161 bits.

Consequently, the leftmost 55 bits portion of the adder is implemented as an incrementer with the carry-out of the lower part as the increment input. The adder also performs end-around-carry adjustment for effective subtraction. As the result might be negative, an array of inverters is required at the output of the adder.

Once the result of the addition is obtained, the amount of normalization shift is determined by the leading one detector (LOD). No right shift for normalization is required due to the initial position of a.

To increase throughput, the FMA unit is implemented in a four-stage pipeline. The position of the pipeline registers is indicated with dashed horizontal lines in Figure 3.3.

The FMA unit can be used to perform floating point addition by making $b = 1$ (or $c = 1$) and multiplication by making $a = 0$.

3.5 Floating-Point Division

The division operation is defined by the following expressions:

$$x = q \cdot d + rem$$

and

$$|rem| < |d| \cdot ulp \quad \text{and} \quad sign(rem) = sign(x)$$

where the *dividend* x and the *divisor* d are the operands and the results are the *quotient* q and the *remainder* rem.

The high-level description of the floating-point division algorithm is composed of the following steps:

1. Divide significands and subtract exponents.

$$M_q^* = M_x^* \,/\, M_d^*$$

$$E_q = E_x - E_d - B$$

2. Normalize M_q^* and update exponent accordingly.
3. Round.
4. Determine exception flags and special values.

Division is implemented in hardware in all general purpose CPUs and in most processors used in embedded systems. Several classes of algorithms exist to implement the division operation in hardware, the most used being the digit recurrence method, the multiplicative method and various approximation methods.

In the following we briefly review these algorithms and implementations. Due to the differences in the algorithms, a comparison among their implementation in terms of performance and precision is sometimes hard to make. In Chapter 6, we will use power dissipation and energy consumption as metrics to compare among these different classes of algorithms.

3.5.1 Division by Digit Recurrence

The digit-recurrence algorithm [63] is a direct method to compute the quotient of the division

$$q = \frac{x}{d} + rem$$

The radix-r digit-recurrence division algorithm is implemented by the residual recurrence

$$w[j+1] = rw[j] - q_{j+1}d \qquad j = 0, 1, \ldots, n$$

with the initial value $w[0] = x$. The quotient-digit q_{j+1}, normally in signed-digit format to simplify the selection function, provides $\log_2 r$ bits of the quotient at each iteration. The quotient-digit selection is

$$q_{j+1} = SEL(d_\delta, y) \qquad q_{j+1} \in [-a, a]$$

where d_δ is d truncated after the δ-th fractional bit and the estimated residual, $y = rw[j]_t$, is truncated after t fractional bits. Both δ and t depend on the radix and the redundancy (a). The residual $w[j]$ is normally kept in carry-save format to have a shorter cycle time.

The divider is completed by a on-the-fly convert-and-round unit [63] which converts the quotient digits q_{j+1} from the signed-digit to the conventional representation, and performs the rounding based on the sign of the remainder computed by a sign-zero detect (SZD) block. The conversion is done as the digits are produced and does not require a carry-propagate adder.

The digit-recurrence algorithm is quite a good choice for the hardware implementation because it provides a good compromise between latency, area and power and rounding is simple (the remainder is computed at each iteration). A radix-4 division scheme is implemented in Intel Pentium CPUs [64], in ARM processors [65] and in IBM FPUs [66].

Radix-4 Division Algorithm

We now briefly summarize the algorithm for radix-4 with the quotient digit selected by comparison [65]. The radix-4 recurrence is

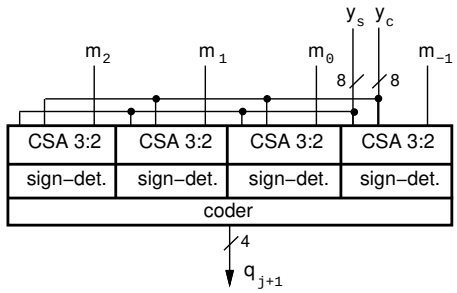
$$w[j+1] = 4w[j] - q_{j+1}d \qquad j = 0, 1, \ldots, n$$

with $q_{j+1} = \{-2, -1, 0, 1, 2\}$.

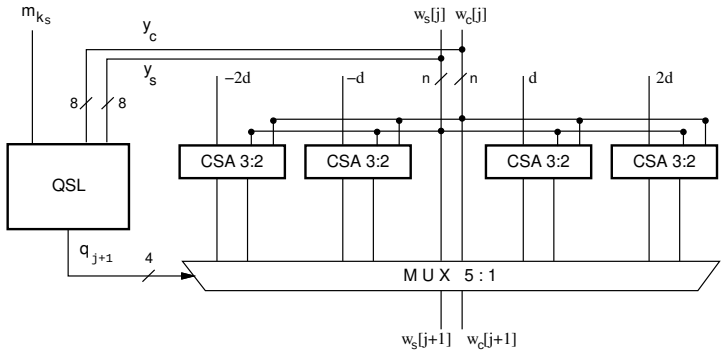
The quotient-digit q_{j+1} is determined by performing a comparison of the truncated residual $y = \widehat{4w[j]}$ (carry-save) with the four values (m_k) representing the boundaries to select the digit for the given d. That is,

$$\begin{array}{rcccl} & y & \geq & m_2 & \rightarrow \quad q_{j+1} = 2 \\ m_1 & \leq & y & < & m_2 \quad \rightarrow \quad q_{j+1} = 1 \\ m_0 & \leq & y & < & m_1 \quad \rightarrow \quad q_{j+1} = 0 \\ m_{-1} & \leq & y & < & m_0 \quad \rightarrow \quad q_{j+1} = -1 \\ & & y & < & m_{-1} \quad \rightarrow \quad q_{j+1} = -2 \end{array}$$

This selection can be implemented with a unit (QSL) similar to that depicted in Figure 3.4.a where four 8-bit comparators (sign-det.) are used to detect in which range y lies. The coder then encodes q_{j+1} in 1-out-4 code which is suitable to drive multiplexers.



a)



b)

Figure 3.4: **a)** Selection by comparison (QSL). **b)** Single radix-4 division stage.

In parallel, all partial remainders $w^k[j+1]$ are computed speculatively (Figure 3.4.b), and then one of them is selected once q_{j+1} is determined.

The critical path of the unit in Figure 3.4 is

$$t_{REG} + t_{CSA}^{QSL} + t_{8b-CPA}^{QSL} + t_{buffer} + t_{MUX}$$

Intel Penryn Division Unit

The division unit implemented in the Intel Core2 (Penryn) family is sketched in Figure 3.5 [64]. It implements IEEE binary32/binary64 compliant division, plus extended precision (64 bits significand) and integer division. The unit consists of three main parts: the pre-processing stage necessary to normalize integer operands to ensure convergence; the recurrence stage; and the post-processing stage where the rounding is performed.

The recurrence is composed of two cascaded radix-4 stages synchronized by a two-phase clock to form a radix-16 stage (4 bits of quotient computed) over a whole clock cycle. Each radix-4 stage is realized with a scheme similar to that of [65] shown in Figure 3.4.

This scheme was selected by Intel because of the reduced logical depth. However, the speculation on the whole w-word (54 bits for [65], 68 bit for the Core2 format) is quite expensive in terms of area and power dissipation.

According to [64], a maximum of 6+15=21 cycles are required to perform a division on binary64 (double-precision) operands.

Radix-16 by Overlapping Two Radix-4 Stages

An alternative to the Penryn solution, is to have a radix-16 divider obtained by overlapping (and not cascading) two radix-4 stages. In this scheme, the speculation is applied to the narrower y-path as explained next. Examples of radix-16 dividers by radix-4 overlapping are reported in [63] and [67].

The radix-16 retimed recurrence, illustrated in Figure 3.6.a, is

$$\begin{aligned} v[j] &= 16w[j-1] - q_{Hj}(4d) \\ w[j] &= v[j] - q_{Lj}d \end{aligned}$$

with $q_{Hj} \in \{-2,-1,0,1,2\}$, $q_{Lj} \in \{-2,-1,0,1,2\}$, and $w[0] = x$ (eventually shifted to ensure convergence). In Figure 3.6.a, the position of the registers is indicated with a dashed horizontal line. The recurrence is retimed (the selection function is accessed at the end of the cycle) to increase the time slack in the

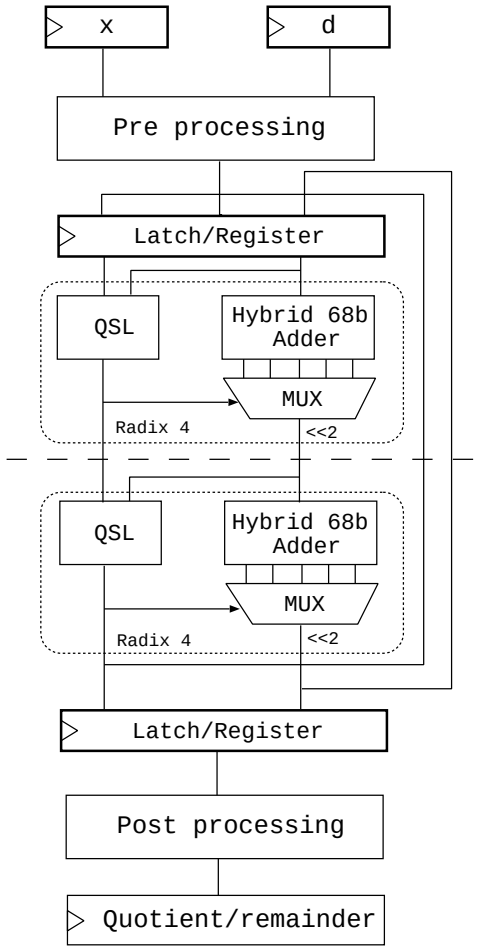


Figure 3.5: Architecture of Penryn divider (significands only).

bits of the wide w-path (at right) so that these cells can be redesigned for low power [67].

The block QSL in Figure 3.6.b is the same as that of Figure 3.4.a. In this case, while q_H is computed, all five possible outcomes of q_L are computed speculatively. Therefore the computation of q_L is overlapped to that of q_H, and q_L is obtained with a small additional delay.

The total number of iteration to compute a binary64 division, including initialization and rounding, is 18.

3.5.2 Division by Multiplication

The quotient q of the division can also be computed by multiplication of the reciprocal of d and the dividend x

$$q = \frac{1}{d} \cdot x$$

This is implemented by the approximation of the reciprocal $R = 1/d$, followed by the multiplication $q = R \cdot x$.

By determining $R[0]$ as the first approximation of $1/d$, R can be approximated in m steps by the Newton-Raphson (NR) approximation [60]

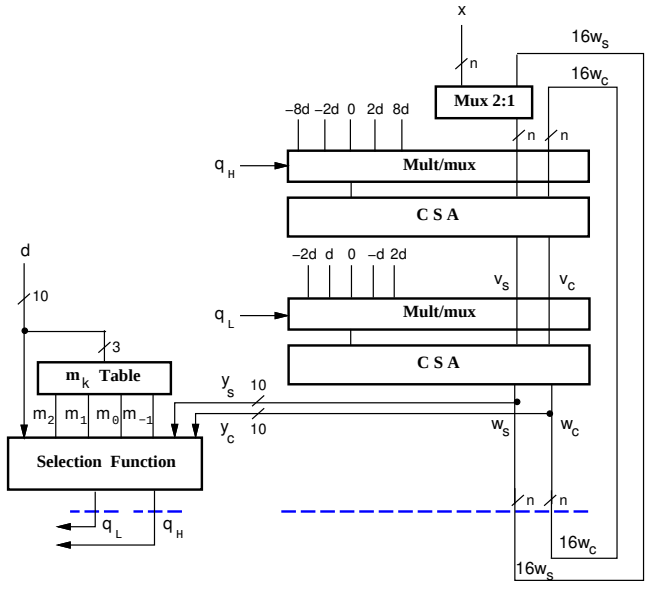
$$R[j+1] = R[j](2 - R[j]d) \qquad j = 0, 1, \ldots, m$$

Each iteration requires two multiplications and one subtraction. The convergence is quadratic and the number of iterations m needed depends on the initial approximation $R[0]$, which is usually implemented by a look-up table.

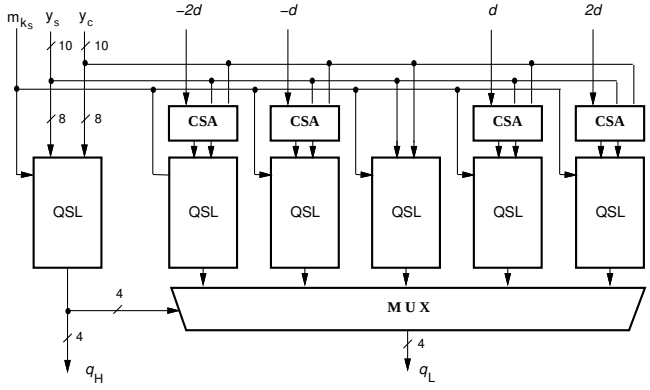
Once $R[m]$ has been computed, the quotient is obtained by an additional multiplication $Q = R[m] \cdot x$. To have rounding compliant with IEEE standard, extra iterations are required to compute the remainder and perform the rounding according to the specified mode [60]:

- $rem = Qd - x$
- $q = ROUND(Q, rem, mode)$.

The NR algorithm for binary64 division ($m = 2$) with an initial approximation of 8 bits is summarized below.



a)



b)

Figure 3.6: **a)** Recurrence radix-16. **b)** Overlapped selection function.

```
R[0] = LUT(d);
FOR i := 0 TO 2 LOOP
    W = 2 - d * R[i];
    R[i+1] = R[i] * W;
END LOOP;
Q = x * R[3];
rem = x - d * Q;
q = ROUND(Q,rem,mode);
```

Although division by iterative multiplication is expensive in power, it has been chosen to implement division in AMD processors [68], NVIDIA GPUs [69], and in Intel Itanium CPUs utilizing the already existent FMA unit.

To implement the NR algorithm using the existing FMA instruction, the look-up table for the initial approximation has to be performed in software. Subsequently, the NR iterations can be executed directly in the FMA unit in Figure 3.3. An extra clock cycle is required to forward the result from the output register to the input register between each FMA instruction. Thus, excluding the initial approximation a total of $8 \times 5 + 1 = 41$ cycles is required to implement division in software.

As can be seen, the latency of software implementation is quite long. In the following, we illustrate how to implement the NR algorithm in hardware based on the FMA unit shown in Figure 3.3. In order to achieve the initial approximation and implement the NR algorithm, the FMA unit in Figure 3.3 needs to be augmented with a look-up table and several multiplexers and registers to bypass intermediate results. The implementation of the multiplicative method based on a FMA unit is shown in Figure 3.7.

A look-up table, providing an 8-bit initial approximation is generated using the midpoint reciprocal method [70], of which the entries are the reciprocals of midpoints of the input intervals. The dividend x is stored in register B and divisor d in register C.

The first cycle is to obtain the initial approximation $R[0]$. After that, the operations performed in the 4-stage pipelined unit of Figure 3.7 are the following (Stage 1 is abbreviated S1, etc.):

S1 The initial approximation $R[0]$ is multiplied by d using the tree multiplier.

S2 The product is subtracted from 2 to obtain $2 - R[0]d$. This is achieved by setting register A to the value of 2 in the previous stage. The result is stored in register W ($W[1] \leftarrow (2 - R[0]d)$).

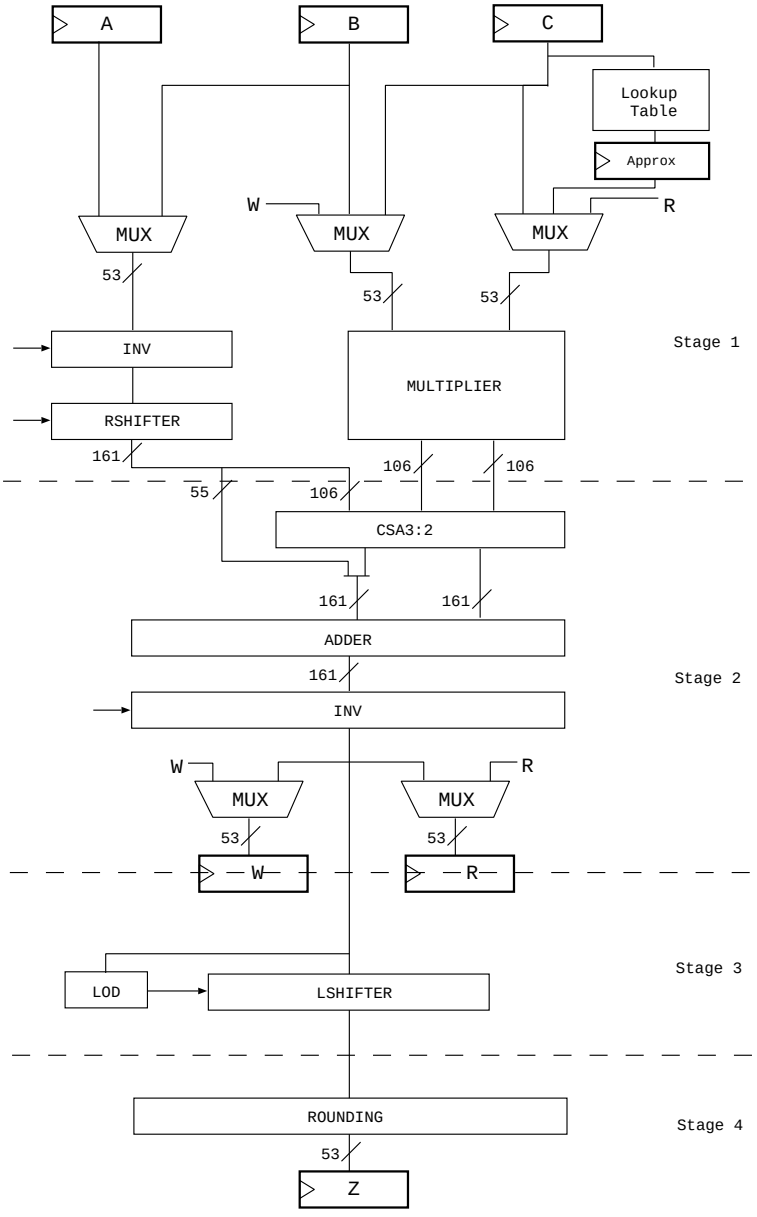


Figure 3.7: Scheme of the modified FMA unit to support division.

S1 $W[1]$ is multiplied by $R[0]$.

S2 The new approximation $R[1] \leftarrow W[1]R[0]$ is stored in register R. The new approximated reciprocal has a precision of 16 bits.

The above 4 steps have to be repeated two more times to have $R[3]$ with the precision necessary for binary64 division.

Once the correct approximation of $1/d$ has been computed, another two iterations in the multiplier are required to compute:

1. the non-rounded quotient: $Q = x \cdot R[3]$;
2. the remainder: $rem = Q \cdot d - x$ necessary for IEEE compliant rounding.

Finally, Q is rounded according to the remainder and the specified rounding mode

$$q = ROUND(Q, rem, mode) \ .$$

Summarizing, the number of clock cycles required for the implementation of the division algorithm with the unit of Figure 3.7 is 18 as detailed in Table 3.1. The intermediate results are stored in denormalized format and consequently the normalization and rounding stages can be bypassed between iterations.

	cycles
initial approx. $R[0]$	1
three NR iterations	$2 \times 6 = 12$
non-rounded quotient $Q = x \cdot R[3]$	2
remainder $rem = Q \cdot d - x$	2
rounding	1
Total cycles	18

Table 3.1: Cycles for binary64 division in FMA unit.

3.5.3 Division by Piecewise Interpolation

Alternatively, the reciprocal $1/d$ can also be obtained by polynomial approximation. This approximation is normally applied for operations in limited precision, such as single precision. For larger precisions, the coefficients look-up table is too large for practical implementation.

Once the reciprocal $1/d$ is obtained, the quotient can be obtained by multiplication of $1/d$ and x. The method proposed in [71] to generate the coefficients results in a smaller table than that of [72]. The function proposed in [71] to compute $1/d$ is approximated as follows:

$$f(d) \approx \begin{cases} K_y + K_m(d - d^*) + K_{p1}(d - d^*)^2 \\ \qquad\qquad\qquad\qquad \text{for } d < d^* \\ K_y + K_m(d - d^*) + K_{p2}(d - d^*)^2 \\ \qquad\qquad\qquad\qquad \text{for } d > d^* \end{cases}$$

where d^* is the mid-point in each interval. A look-up table is used to retrieve optimized coefficients and the polynomial is evaluated by a high speed datapath.

The first 6 fractional bits in d is used to index the look-up table to retrieve K_y and K_m with precision of 17, and 27 bits, respectively. K_p with a precision of 12 bits has twice the entries than K_y and K_m and thus requires 7 bits in d. In total, the coefficients correspond to a table size of $64 \cdot (2 \cdot 12 + 17 + 27) = 4288$ bits. The error to approximate $1/d$ is smaller than 2^{-24}. The approximation tables with the values of K_y, K_m and K_p are reported in [73, 74].

Figure 3.8 shows the implementation of the approximation unit (in blue color) followed by a multiplier. A squarer is used to compute $(d - d^*)^2$, which is then multiplied with K_p to obtain $K_p(d - d^*)^2$. In parallel to this, another multiplier computes $K_m(d - d^*)$. Once the individual terms are ready, they are summed up to form the approximation of reciprocal r. An additional multiplication of x and r produces the quotient.

Implementing binary64 (double precision) division by piecewise interpolation would require a too big coefficients table, therefore in Chapter 6 the polynomial approximation unit is not considered as an alternative. Instead, a comparison of power dissipation for binary32 (single precision) input is discussed in [73].

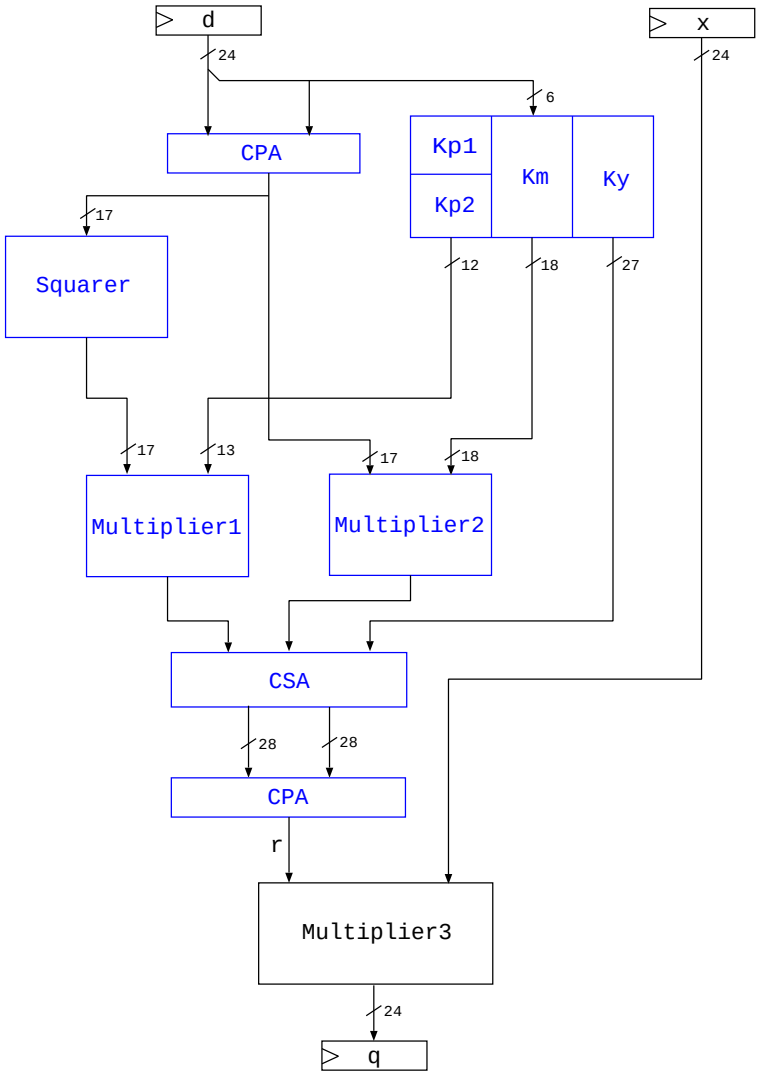


Figure 3.8: Implementation of division by polynomial approximation (significands only).

Chapter 4

Thermal Modeling

Modeling of thermal behavior within a CMOS circuit provides the basis for thermal management and optimization. In Section 2.3, we show that heat distribution can be described in Fourier's law of diffusion. In our work, steady state thermal analysis is performed using the Finite Difference Method (FDM) to approximate Fourier's equation. The resulting linear system from FDM discretization is converted to an RC equivalent circuit and solved by circuit analysis tool SPICE.

Functional cells and interconnects are the sources of heat generation in a CMOS circuit. Although the major heat source is cells, the interconnects could still play a role in shaping thermal profile on the silicon substrate by serving as low resistance conducting paths due to the much larger thermal coefficients in metals. We provide a preliminary investigation on the impact of interconnect using both analytical and experimental methods.

Electrical resistivity in metal is in general temperature dependent and interconnects of the same length but different temperature may have large difference in signal propagation delay. This is more pronounced in global wires which are routed across many functional blocks on the substrate with large temperature variations. We describe a way of estimating temperature dependent wire delay during the floorplanning stage.

4.1 A SPICE Simulation Based Thermal Modeling Method

4.1.1 Steady State Thermal Analysis

For steady state analysis, all derivatives with respect to time in Eq. (2.10) become zero and thermal analysis corresponds to solving the Poisson's equation [75],

$$\nabla^2 T(\mathbf{r}) = -\frac{g(\mathbf{r})}{k_t} \tag{4.1}$$

where $\mathbf{r}$ is the spatial coordinate of the point at which temperature is being determined and g is the power density function of the heat source (in W/m^3).

The Finite Difference Method (FDM) is used to approximate Eq. (4.1) as a difference equation through space discretization. The FDM method can easily account for non uniformity in thermal conductivities among different materials and obtain a highly accurate temperature distribution at very small geometries (standard cell level).

In the FDM method, a chip is discretized into 3-D cuboids with $\triangle x$, $\triangle y$ and $\triangle z$ denote the lengths of cuboids along the x, y and z axis. Let $T_{i,j,k}$ denote the steady state temperature at point ($i\triangle x$,$j\triangle y$ and $k\triangle z$), where i, j and k are the offsets in each dimension. Along the x direction we can write,

$$\frac{\partial^2 T(\mathbf{r})}{\partial^2 x} \approx \frac{\frac{T_{i-1,j,k}-T_{i,j,k}}{\triangle x} - \frac{T_{i,j,k}-T_{i+1,j,k}}{\triangle x}}{\triangle x} \tag{4.2}$$

Let $R_{i-1,j,k} = \triangle x / k A_x \triangle x$ denote thermal resistance in each cuboid along the x direction and $A_x = \triangle y \triangle z$ denote the cross sectional area. Eq. (4.2) can be rewritten as,

$$\frac{\partial^2 T(\mathbf{r})}{\partial^2 x} \approx \left[\frac{T_{i-1,j,k} - T_{i,j,k}}{R_{i-1,j,k}} - \frac{T_{i,j,k} - T_{i+1,j,k}}{R_{i,j,k}}\right] \cdot \frac{1}{k A_x \triangle x} \tag{4.3}$$

Similar equations can be written in the y and z directions.

Thus, the Poisson's equation (Eq. (4.1)) can be approximated using the following linear equation,

$$\begin{gathered}\left[\frac{T_{i-1,j,k}-T_{i,j,k}}{R_{i-1,j,k}} + \frac{T_{i+1,j,k}-T_{i,j,k}}{R_{i,j,k}}\right] + \\ \left[\frac{T_{i,j-1,k}-T_{i,j,k}}{R_{i,j-1,k}} + \frac{T_{i,j+1,k}-T_{i,j,k}}{R_{i,j,k}}\right] + \\ \left[\frac{T_{i,j,k-1}-T_{i,j,k}}{R_{i,j,k-1}} + \frac{T_{i,j,k+1}-T_{i,j,k}}{R_{i,j,k}}\right] = -G_{i,j,k}\end{gathered} \tag{4.4}$$

where $G_{i,j,k} = g_{i,j,k} \triangle V$ is the total power generated within each cuboid. Eq. (4.4) is equivalent to Kirchhoff's Current Law describing nodal voltage in circuit analysis [76]. By modeling $T_{i,j,k}$ as nodal voltage, boundary conditions as voltage sources and power consumption as current sources, we can obtain an RC equivalent circuit for the heat conducting network within the chip.

The RC equivalent circuit is a netlist of resistors, current sources and voltage sources for steady state analysis. For transient analysis, the netlist would also include capacitors modeling time varying power densities. In our work, we focus on steady-state analysis. The RC equivalent circuit can be solved using circuit analysis techniques to obtain all nodal voltages (thus temperatures).

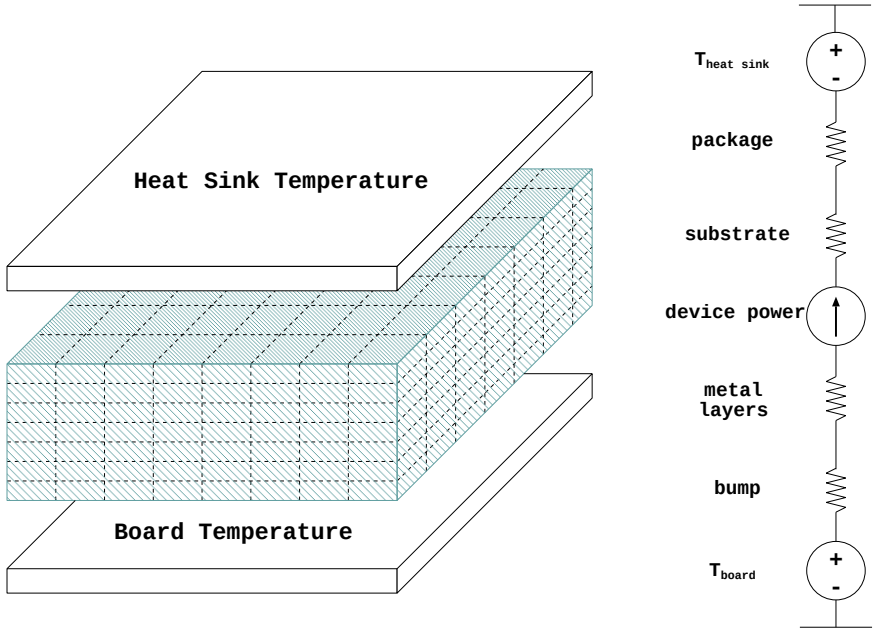


Figure 4.1: 3-D discretization of a chip.

Figure 4.1 illustrates the discretization of a chip. The PCB board and surface of the heat sink are assumed to be isothermal and form the boundary conditions for heat conduction within the chip. Two major heat conduction paths are modeled from the device layer to the ambient environment. The primary conduction path is through substrate layers and package layer to heat sink. The secondary conduction path is through metal layers and bump layer to PCB board. The sidewalls of the chip are assumed to be adiabatic where no heat exchange occurs.

The chip is meshed into a grid of thermal cells, of which the size can be tuned according to the granularity of analysis. For macro-scale thermal analysis, quantum mechanical effects at small length scales can be ignored and thus the size of thermal cells is much larger than standard cells. In addition, due to the low pass filter effect hotspot is usually formed by a cluster of high activity gates instead of a single gate, which means on the device layer a thermal cell may cover several standard cells.

A thermal cell is modeled as an RC equivalent circuit illustrated in Figure 4.2, composed of thermal resistance in x, y and z directions and a current source. We assume active gates to be the only sources of heat generation and self heating in interconnects do not have impact on substrate temperature. Consequently only thermal cells on the device layer dissipate power, which equals the total amount of power consumption in standard cells covered by a thermal cell.

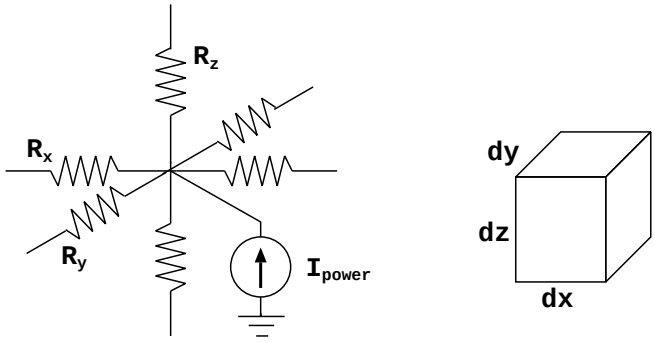


Figure 4.2: Equivalent RC model of a thermal cell.

To compute thermal resistance in each thermal cell, the value of thermal conductivity k is needed. However, on each layer thermal conductivities in the same direction can be different at different locations due to complex layout patterns. The metal layer, for example, is a non uniform mixture of materials such as metal and inter metal dielectric (IMD) with very different thermal properties. Discretization of a chip at submicron scale to align with material geometries is impractical due to the excessive problem size resulted from such fine granularity meshing. To facilitate an efficient yet accurate 3-dimensional thermal analysis, the equivalent average thermal conductivity for each layer can be used.

In our model, we adopted the layer stack-up and thermal coefficients in [77],

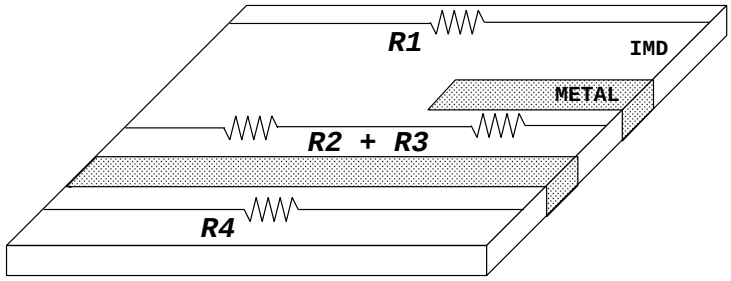


Figure 4.3: Example of computing average thermal conductivity in metal layer.

Layer	Thickness (μm)	k_x ($W/°C \cdot m$)	k_y ($W/°C \cdot m$)	k_z ($W/°C \cdot m$)
package	200	5.0	5.0	5.0
sub $1 \sim 4$	125	63.0	63.0	63.0
device	2	28.0	14.0	69.0
wire $1 \sim 2$	3.1	20.7	20.7	4.26
bump	200	0.05	0.05	0.25

Table 4.1: Layer stackup and thermal properties.

where a 90 nm[1] SoC design is analyzed. Figure 4.3 illustrates how thermal conductivity on a metal layer is derived. Thermal resistance in the x (horizontal) direction is calculated as combined resistance of R_1, $R_2 + R_3$ and R_4, where R_1 represents inter-metal dielectric (IMD), $R_2 + R_3$ represents IMD and the metal wire and R_4 is the metal wire only. The equivalent resistances for y and z directions are calculated in the same manner. In [77], the authors listed the thermal resistances (R_x, R_y and R_z) in a thermal cell on each layer. We derived the corresponding thermal conductivity (k) on each layer based on the relationship $R = L/kA$. The layer stack up and thermal conductivity factors are listed in Table 4.1 and used throughout our experiments. The only difference from [77] is that we chose a larger conductivity in the package layer so that thermal resistance per cm^2 equals to $0.4\,°C/W$, which is close to a commodity microprocessor's package.

The equivalent RC circuit obtained from FDM discretization can also be solved using general methods for linear equations. To compare the efficiency of SPICE against other linear solvers, we performed thermal simulations of the same prob-

[1]Although our design flow is based on a 65 nm standard cell library and we assume thermal coefficients are the same as the 90 nm design.

lem of 3600 thermal cells or unknowns using different methods. We implemented a few linear solvers to test against SPICE, namely direct methods including LU decomposition, Cholesky decomposition and iterative methods including Jacobi method, Gauss-Seidel method and Successive Over Relaxation (SOR) method. SPICE simulation and linear solvers result in the same temperature values, however, the runtime varies significantly for the same problem size. Table 4.2 lists the results, where the runtime is shown in seconds.

Method	*Runtime(s)*	*Ratio*
SPICE	21.99	1.0
LU dcmp	504.98	23.0
Cholesky dcmp	124.53	5.7
Jacobi	691.26	31.4
Gauss-Seidel	479.30	21.8
SOR	320.04	14.6

Table 4.2: Speed comparison between SPICE and linear solvers.

As can be seen in Table 4.2, SPICE is the fastest among all the methods. Even the best linear solver Cholesky decomposition, which takes advantage of the symmetric property of the conductance matrix, is more than five times slower. SPICE, on the other hand, can utilize advanced circuit analysis techniques and results in much less simulation time.

4.1.2 Impact of Interconnect in Substrate Heat Distribution

In CMOS processes, interconnect uses copper or aluminum and has a much larger thermal conductivity than bulk silicon and insulator materials such as SiO_2. At room temperature, thermal conductivity of copper is about 400 W/mK, three times larger than silicon which has a thermal conductivity of 130 W/mK. If large temperature difference exists on the two ends of an interconnect, it could potentially serve as a low resistance path and redistribute heat on the substrate layer as illustrated in Figure 4.4. We performed a preliminary analysis on the impact of interconnect using a simplified structure model which only consists a piece of silicon and a metal wire. For a detailed analysis, other factors such as self-heating in wires, material used in the via and the diffusion area, etc. also need to be considered.

We compared the ability of heat conduction between a wire in layer Metal Three (M3) and a piece of silicon of the same length in Table 4.3. M3 is typically used

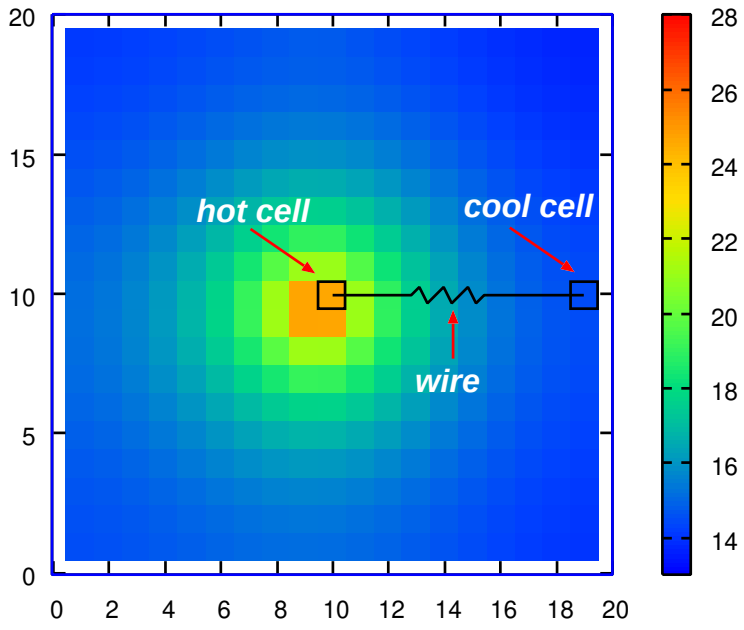


Figure 4.4: A cool cell connected to a hot cell through metal wire.

for local interconnection between standard cells. Copper is used in the metal layer and is sized twice wider than the minimum width required in the design manual.

	Ratio(Si/Cu)
Length(μm)	1
Width(μm)	13
Thickness(μm)	3.0
Thermal Conductivity(W/mK)	0.3
Thermal Resistance(K/W)	1/11.7

Table 4.3: Comparison of heat conductivity between copper and silicon.

It can be seen from Table 4.3 that although thermal conductivity of copper (Cu) is more than 3 times better than silicon (Si), the resistance in wire is actually much larger due to its small cross sectional area. In reality, the ratio of thermal resistance would be smaller than 1/11.7 because the effective resistance in silicon is smaller due to multiple parallel conducting paths between the two ends through adjacent thermal cells. In addition, most signal wires are sized using the minimum width, which is only half of the value used in the above comparison. Consequently, if a cool region gets 10 °C temperature rise due to heat diffusion in the substrate from a hot region, heat diffusion in interconnect only contributes to less than 1 °C. In other words, substrate is the dominating layer of horizontal heat diffusion.

To verify the above analysis using our model, we created a two dimensional grid of cells in a square shape where cells in the center have a high power consumption and all other cells have zero power consumption. In this way, temperature rise in inactive cells is solely due to heat diffusion from the active cell through substrate. Thermal simulation is performed to obtain temperature values in all cells.

The second step is adding a resistor to the equivalent RC circuit to model interconnects in M3 connecting the hot cell (in the center) and a cool cell on the boundary. The width of the wire is twice the minimum and length of the wire is the distance from the center of the hot cell to the center of the cool cell. This time the temperature rise in the cool cell includes contributions from heat diffusion through the interconnect.

Simulation results are shown in Table 4.4. $\triangle T$ is the extra temperature increase due to heat diffusion through interconnect.

When heat is only diffused through the substrate, temperature rise is 23.3 °C in the hot cell and 14.37 °C in the cool cell. When a single wire is considered,

	T_{rise}(°C)	$\triangle T$(°C)
substrate only	14.37	
substrate and 1 wire	14.63	0.26
substrate and 10 wires	16.32	1.95

Table 4.4: Temperature rise in the cool cell.

temperature in the cool cell increased by another 0.26 °C reaching 14.63 °C. For ten wires the interconnect contributes 1.95 °C, corresponding to about 12% of the total temperature rise.

In both cases, temperature increase due to wire is smaller than the value predicated from the analysis based on Table 4.3 (approx 1 °C). This is because heat can diffuse in the substrate layer to the cool cell not only through cells in a straight line source-sink, but also through the cells in adjacent rows. Therefore, the effective resistance in the substrate layer becomes smaller than the value used in the analysis in Table 4.3.

In conclusion, if wire density between thermal cells in different regions is not high, as in most cases, we can reasonably assume that the heat is mainly distributed in the substrate layer, and contribution of the interconnects is marginal.

4.1.3 Design Flow and Benchmarks

Unlike other fields in Electronic Design Automation such as placement and timing analysis, there are no standard cell based circuit benchmarks specifically designed for investigating thermal properties. Some works in the literature used the MCNC[2] and ISPD[3] standard cell placement benchmarks in their experiments. However, these circuits are composed of generic cells and information about power consumption inside these circuits are not available.

As described in Section 2.1, power consumption is closely related to the function of a circuit as its main component is the switching of load capacitance. In our work, benchmark circuits are synthesized from RTL descriptions into gate level netlist using commercial EDA tools. Since the patterns of circuits input are usually unknown, we apply random test vectors to obtain power consumption in these circuits.

[2]MCNC is the abbreviation of Microelectronics Center of North Carolina and the benchmark suite contains macro cell level, standard cell level and mixed designs.

[3]ISPD is the abbreviation of The International Symposium on Physical Design and the benchmark suite is for physical design applications.

Our design flow is based on an industrial 65 nm standard cell library, which is illustrated in Figure 4.5. We used Synopsys' VCS for logic simulation and switching activity annotation, Design Compiler for logical synthesis, IC compiler for physical placement and Power Compiler for power estimation. Power estimation is based on the annotated switching activity obtained from applying random test vectors to the benchmarks. The post placement netlist of standard cells and their power consumption are mapped to our SPICE based thermal simulator to obtain a thermal map.

The first step of the thermal simulation block in Figure 4.5 is to map the standard cells to a two dimensional array of thermal cells according to their locations. The size of each dimension depends on the circuit size and the granularity of analysis. A SPICE netlist of these thermal cells is then constructed. The current in each thermal cell equals to the total amount of power consumption from all mapped standard cells. The SPICE simulation result will return the voltage in all thermal cells, which is the equivalent temperature value according to the electrical-thermal analogy.

To gain some insight in thermal behavior within a circuit, we first tried the ISCAS benchmarks which are described in verilog and are widely used for placement and testing. In order to obtain thermal maps with significant temperature gradient, we also designed a synthetic benchmark circuit where we can force large variations of power density at different locations. In this way, we can intentionally create hotspots and thermal gradients to explore the correlation between functional and spatial hotspots. The benchmark circuit is composed of 8 identical $24\,bit \times 24\,bit$ Booth encoded multipliers and is synthesized with a clock period of 1 *GHz*. We choose multipliers as they usually dissipate quite a lot of power due to the partial product tree and reduction network.

4.1.4 Experiment Results

Circuit geometries and simulation results for the ISCAS circuits are shown in Table 4.5. In column 5, $MaxT_{rise}$ reports the maximum temperature rise above the ambient, and in column 6, $\triangle T$ reports the temperature difference between Max T_{rise} and the minimum temperature rise.

As can be seen from Table 4.5, the ISCAS benchmark circuits are small in size and flat in power profile. Due to the heat diffusion in the substrate layer, temperature in the cells are quite even. As a result, we did not observe significant thermal gradient ($\triangle T > 1\,^{\circ}\mathrm{C}$) in these circuits.

For our synthetic benchmark, the layout of the circuit occupies a square shape

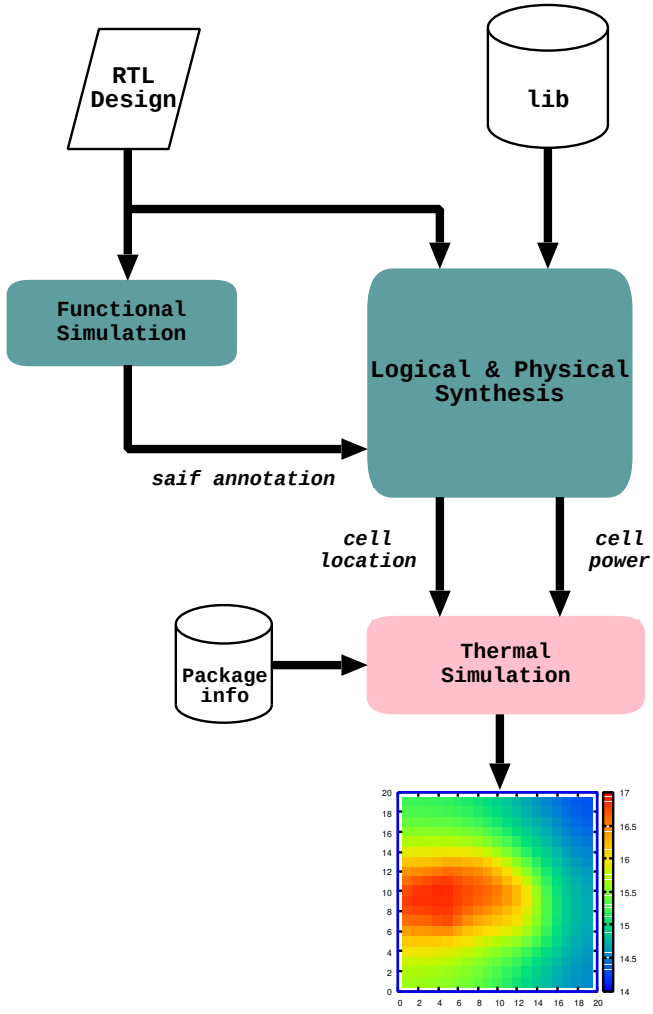


Figure 4.5: Standard cell design thermal simulation flow.

Circuit	# of Cells	Area(μm^2)	Total Power(mW)	Max T_{rise}(°C)	$\triangle T$(°C)
c432	210	27.2×28.6	0.069	4.49	0.00
c1355	313	40.0×41.6	0.456	15.20	0.02
c499	331	42.9×41.8	0.434	13.12	0.02
c1908	334	42.8×40.2	0.291	9.69	0.01
c880	353	40.0×39.0	0.089	2.96	0.00
c2670	529	47.0×49.4	0.212	6.44	0.01
c3540	969	62.0×62.4	0.372	5.20	0.01
c5315	1304	69.2×70.2	0.617	6.82	0.03
c7552	1443	75.4×73.5	0.793	7.86	0.02
c6288	2582	95.3×94.9	1.440	7.24	0.09

Power is measured at 1 GHz for all units.

Table 4.5: Results on ISCAS circuits.

of 400 μm on each side. The layout and position of each multiplier unit is shown in Figure 4.6. Cells within each multiplier are placed close together since the layout tool performs a timing driven placement.

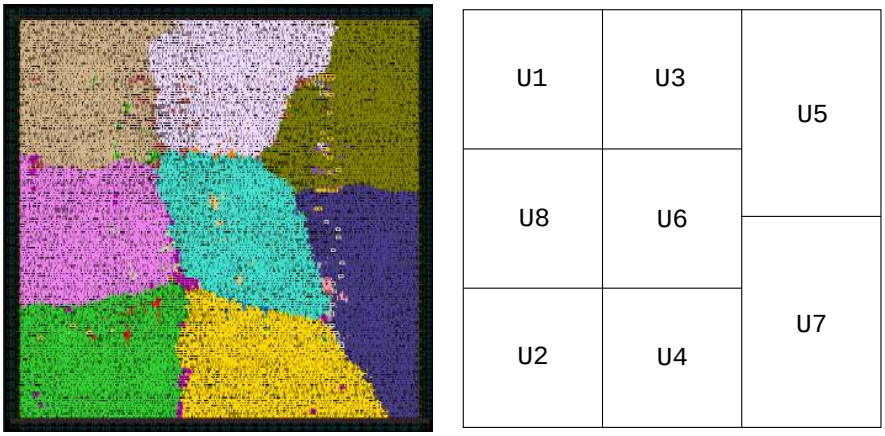
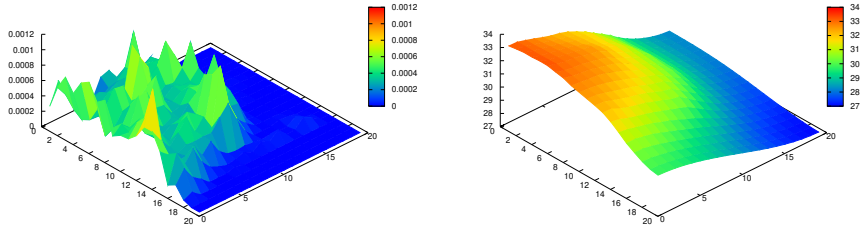


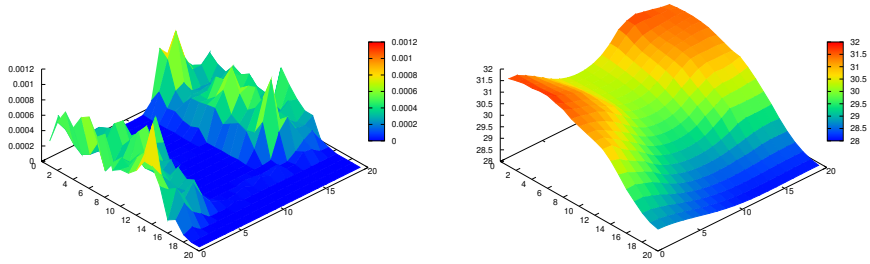
Figure 4.6: Layout of benchmark circuit.

We performed six experiments by activating different combinations of units. Active units dissipate both dynamic and static power and inactive units dissipate static power only. Table 4.6 lists all configurations used in the circuit and the experiment results.

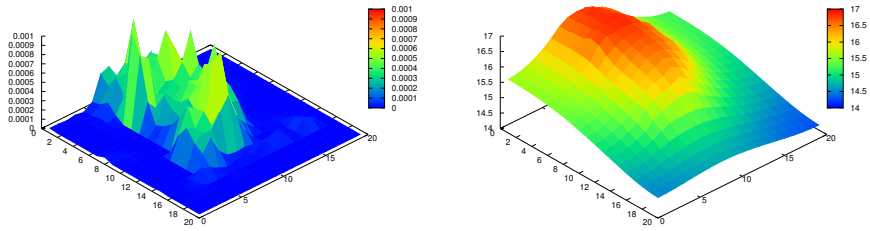
The experiments can be categorized into 4 groups according to the number of active units.



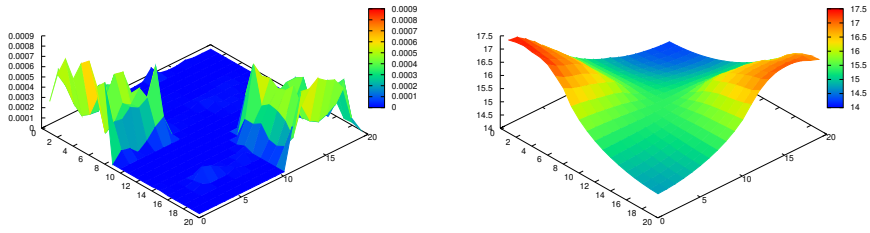
a) U2, U4, U6 and U8 are active.



b) U1, U2, U3 and U4 are active.



c) U6 and U8 are active.



d) U2 and U5 are active.

Figure 4.7: Power (left) and thermal (right) profiles of test2 and test3.

# of Act.Units	Active Units	Tot.Power(mW)	Max T_{rise}(°C)	$\triangle T$(°C)	ratio
8	All	158.0	60.0	1.39	
4	U2,U4,U6,U8	79.8	33.3	6.31	+68%
	U1,U2,U3,U4	79.7	31.8	3.76	
2	U2,U5	41.4	17.5	3.19	+23%
	U6,U8	41.0	16.9	2.59	
1	U8	21.4	9.7	2.50	

Power is measured at 1 GHz for all units.

Table 4.6: Peak temperature and gradient in the synthetic benchmark.

- **test1** all units are active.
- **test2** four units are active (with two different configurations).
- **test3** two units are active (with two different configurations).
- **test4** only one unit is active.

In test1, we activate all 8 units which results in the highest power consumption and peak temperature. The thermal profile is flat and has the smallest gradient since all units are dissipating power. In test4 we activate unit U8 only and the hotspot has the smallest peak temperature. Test2 and test3 are of more interest and we show their power and thermal maps in Figure 4.7. In the maps, unit of power values is W and unit of temperature rise above ambient is °C. In test2, we first make units U2, U4, U6 and U8 active which are placed close together in the lower left corner as shown in Figure 4.6. Then units U1, U2, U3 and U4 are made active, which are located on the two opposite sides of the circuit. The results showed that in the first case the peak temperature rise is higher. Similarly, in test3 we performed 2 experiments each activating 2 different units. The results in row 4 and 5 in Table 4.6 showed the similar thermal behavior. With the same amount of power consumption, the hotspot exhibits a higher temperature when active units are placed close and thus power density is larger. We also compared the difference in thermal gradient in test2 and test3 in the last column in Table 4.6. When hotspots are closer, thermal gradient is also larger which reflects the trend that peak temperature can increase faster than average temperature.

4.1.5 Analysis of Thermal Behavior

The experiment results provide us with insight and guidelines into some important aspects of thermal behavior and thermal aware design. The benchmark

circuit, although relatively small in area, already has significant temperature gradient (e.g. larger than 6 °C along a diagonal of 500 um long). Due to the short circuiting effect of heat diffusion in the substrate layer, the maximum thermal gradient is much smaller than the maximum power gradient. For example, an active unit dissipates 100 times more power than an inactive unit which only dissipates static power. The thermal gradient, even in the worst case in our experiments, is just 6.31 °C.

The results also show that the thermal profile is highly dependent on the relative location of high power consumption units. Functional hotspots should be placed away from each other as far as possible in order to reduce thermal coupling. Low power consumption units can be placed closer to hotspots to serve as their heat spreader. On the other hand, temperature sensitive units (e.g. analog blocks, leakage power dominant blocks) need to be placed carefully in order to minimize heat diffusion from hotspots.

From another point of view, we can also see how peak temperature can be reduced by using dynamic scheduling policies on systems with multiple execution units such as multicore architectures. Test1 and test4 can be seen as two extreme cases where a task is performed either on all units or only on one unit. The execution time is shorter when all units are active and work in parallel but the peak temperature is also much higher. Alternatively, if there is slack between execution time and deadline requirement or peak temperature becomes the primary concern once exceeding a threshold value, we can schedule the task on fewer units to avoid overheating.

4.2 Wire Delay Estimation under Substrate Temperature Variation

Interconnect delay increases steadily with technology scaling and global interconnects have already dominated path delays. In Chapter 2, we illustrated that a secondary heat conduction path exists from substrate layer towards metal layers. In nanometer technologies, in spite of an increase in the number of available metal layers, the top metal layers may still get closer to the substrate which results in a stronger thermal coupling between the substrate and the interconnects. Temperature in interconnects can reach very high due to the fact that metal layers are far away from heat sink, especially in global interconnects.

Electrical resistivity in metal increases linearly with temperature and consequently high temperature causes performance degradation in metal interconnects. Traditional physical design algorithms such as floorplanning assume re-

sistivity in interconnects is uniform and constant and wirelength is used as a metric to estimate signal delay in interconnects. However, in designs where the substrate has nonuniform thermal profile, the traditional way of estimating wire delay can lead to large errors. This is because interconnect performance decreases with an increase in temperature and the delay of two wires of the same length are no longer equal.

Although extensive work has been done in thermal aware floorplanning algorithms, all of them assume electrical resistivity in wires is constant and thermal gradients in the substrate has no impact on wire delay. This assumption is in general invalid and increasingly inaccurate in nanometer high performance designs where large temperature gradients already exist in the substrate. In this section, we first illustrate the impact of nonuniform interconnect thermal profile on the Elmore delay and then we show a new way of estimating wire delay in thermal aware floorplanning algorithms.

4.2.1 Nonuniform temperature dependent wire delay model

The electrical resistance of an interconnect line has a linear relationship with its temperature and can be written as:

$$R(x) = R_0(1 + \beta \cdot T(x)) \tag{4.5}$$

where R_0 is the resistance at reference temperature, β is the temperature coefficient($1/^\circ$C) and $T(x)$ is the temperature profile along the length of the interconnect.

According to the distributed RC Elmore delay model [2, 78], signal propagation delay through an interconnect line of length L can be written as follows:

$$D = R_d \left(C_L + \int_0^L c_0(x)dx \right) + \int_0^L r_0(x) \cdot \left(\int_x^L c_0(\tau)d\tau + C_L \right) dx \tag{4.6}$$

where R_d is the driver cell's ON resistance, $c_0(x)$ and $r_0(x)$ are the capacitance per unit length and resistance per unit length at location x and C_L is the load capacitance.

	Symbol	Value	Unit
temperature coefficient	β	3E-03	$1/°C$
sheet resistance at room temperature	r_{sh}	0.077	Ω/sq
sheet capacitance at room temperature	c_{sh}	0.2	fF/sq

Table 4.7: Electrical and thermal parameters for Al/Cu interconnects.

It can be assumed that the driver cell's ON resistance and the capacitance per unit length do not change with temperature variations [34]. By using Eq. (4.5), we can rewrite Eq. (4.6) as:

$$D = D_0 + (c_0 L + C_L) r_0 \beta \int_0^L T(x)dx - c_0 r_0 \beta \int_0^L x \cdot T(x)dx \tag{4.7}$$

where

$$D_0 = R_0(c_0 L + C_L) + \left(c_0 r_0 \frac{L^2}{2} + r_0 L C_L \right) \tag{4.8}$$

D_0 is the Elmore delay of the interconnect corresponding to the unit length resistance at reference temperature. Typical electrical and thermal parameters for Aluminum/Copper interconnects are given in Table 4.7.

Given a temperature profile and dimension of an interconnect we can calculate its delay from Eq. (4.7). In Figure 4.8, we plot the percentage of delay increase as temperature increases for wires of different lengths. It can be seen from Figure 4.8 that the delay of an interconnect at high temperature can be quite different from the delay at room temperature. The high temperature in an interconnect is mainly caused by self heating and heat diffusion from the substrate. According to [34], assuming the substrate has a uniform temperature profile, the temperature within an interconnect can be written as:

$$T(x) = T_{sub} + \frac{\theta}{\lambda^2} \left(1 - \frac{\sinh \lambda x + \sinh \lambda (L - x)}{\sinh \lambda} \right) \tag{4.9}$$

$$\lambda^2 = \frac{1}{k_m} \left(\frac{k^*_{ins}}{t_m t_{ins}} - \frac{I^2_{rms} \rho_i \beta}{w^2 t^2_m} \right) \tag{4.10}$$

$$\theta = \frac{I^2_{rms} \rho_i}{w^2 t^2_m k_m} \tag{4.11}$$

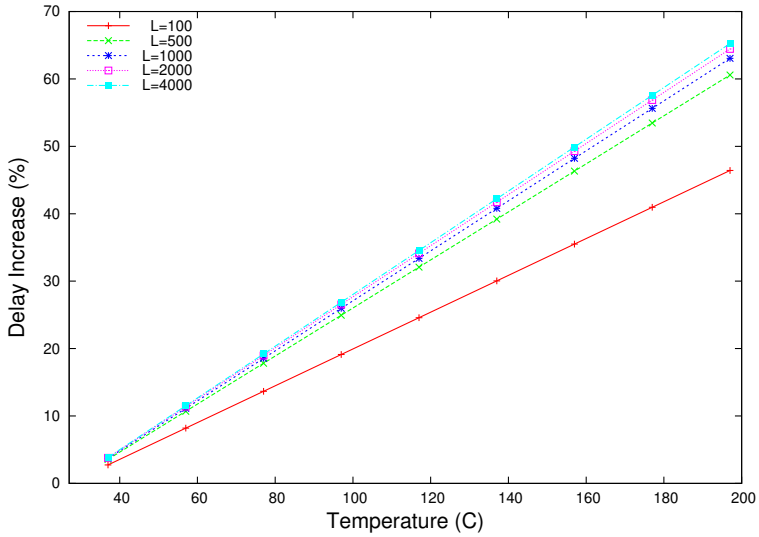


Figure 4.8: Percentage increase of signal delay with respect to nominal delay at room temperature (27 °C).

where θ and λ are constants for a chosen metal layer in a specific technology node. The peak temperature rise is equal to θ/λ^2 for interconnects whose lengths are larger than the heat diffusion length.

Based on 4.9, we plot the thermal profiles for a local, a semi-global and a global Cu interconnect of $1000\,\mu m$ long in a $50\,nm$ technology in Figure 4.9 with parameters provided in ITRS [6]. Temperature in the substrate is assumed to be uniform at 100 °C. Current density is $3.0 \times 10^6 A/cm^2$ for all three layers.

As can be seen in Figure 4.9, the global interconnect which is in the top most metal layer and thus the farthest away from the heat sink has the highest peak temperature.

4.2.2 Estimating temperature dependent wire delay in thermal aware floorplanning

Interconnects between architectural blocks are mostly assigned to semi global and global metal layers, where wirelength is usually large and signal delay is strongly affected by heat diffusion from the substrate. In addition, long wires are

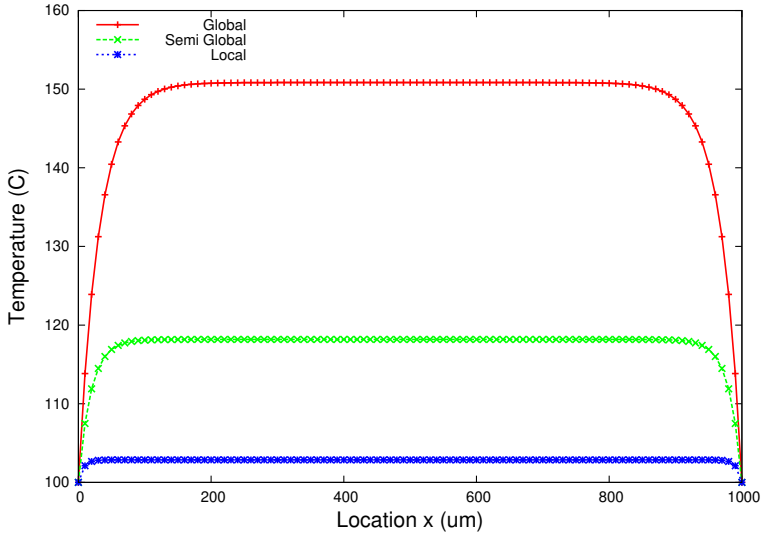


Figure 4.9: Temperature profiles along the length of interconnects on different metal layers.

very likely to be routed above several blocks with significant temperature variations. To estimate the temperature dependent interconnect delay, the nonuniform thermal profile on the substrate layer has to be considered.

At the floorplanning stage, detailed routing information of an interconnect is unknown and wirelength is usually estimated as the Manhattan distance between two connected blocks. Manhattan distance is measured as the half perimeter of the bounding box of two end points of an interconnect. The half perimeter can be either along the upper bend or the lower bend since resistance is constant along the interconnect and therefore delay is the same as long as wirelength is the same. For temperature dependent wire delay estimation, different routes of the same length can have different delay since they may be subjected to different thermal profile on the substrate.

In our work, we restrict the routing of an interconnect to be L-shape with an upper bend as illustrated in Figure 4.10. L-shape routing, also called 1-bend routing, is a type of pattern routing which uses the predefined pattern (of L-shape) to route two end points. Pattern routing can reduce the number of vias and has been shown to allow a more accurate prediction of wirelength and congestion at an early stage in the design flow [79]. In addition, we assume all signals propagate in the direction from left to right between two end-points.

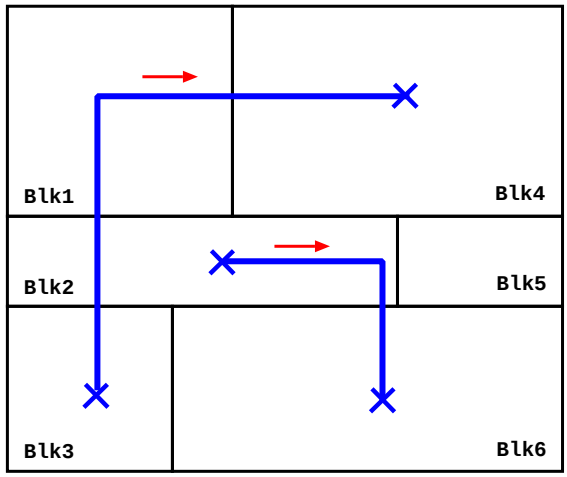


Figure 4.10: Signals propagate from left to right in upper bend L-shape routing.

All interconnects are routed in local ($< 100\mu m$), semi global ($< 500\mu m$) or global layers according to the wirelength. Once the routing of an interconnect is determined, we extract thermal profile of blocks along the route of the interconnect and compute the interconnect's thermal profile. Temperatures within each block usually exhibit some gradient also, but we assume a block has constant thermal profile for the sake of simplicity. The thermal profile of an interconnect is then used in Eq. (4.7) to obtain the propagation delay.

HotFloorplan [80], an architectural level thermal aware floorplanning tool, is used in our experiments to perform floorplanning for the MCNC benchmarks. The optimization process in HotFloorplan is based on simulated annealing and by default it uses a linear combination of total area, peak block temperature and total wirelength as the cost function. A new floorplan is generated by making random moves to the candidate floorplan (e.g. move a block to a new location, etc.). If the move results in an improvement in the cost, the new floorplan is accepted as the candidate, otherwise the new floorplan is accepted based on a probability function.

We used benchmark *ami49* throughout the experiments as a test case, as it contains 49 functional blocks and has the largest area in the MCNC benchmarks. Random power density is assigned to each block in *ami49* to obtain power consumption since the MCNC benchmark is designed for testing traditional floorplanning algorithms and does not contain any information about power consumption in each block.

In HotFloorplan, the connectivity information is stored in a wire density matrix, which is a 2-dimensional matrix where an element with index i and j represents the number of wires between block i and block j. To be compatible with the matrix representation, multiple terminal nets in the MCNC benchmarks are split up into pairwise connections. Position of pins are modeled at the center of the associated block and wire dimensions are derived from predictions of the $50nm$ process node.

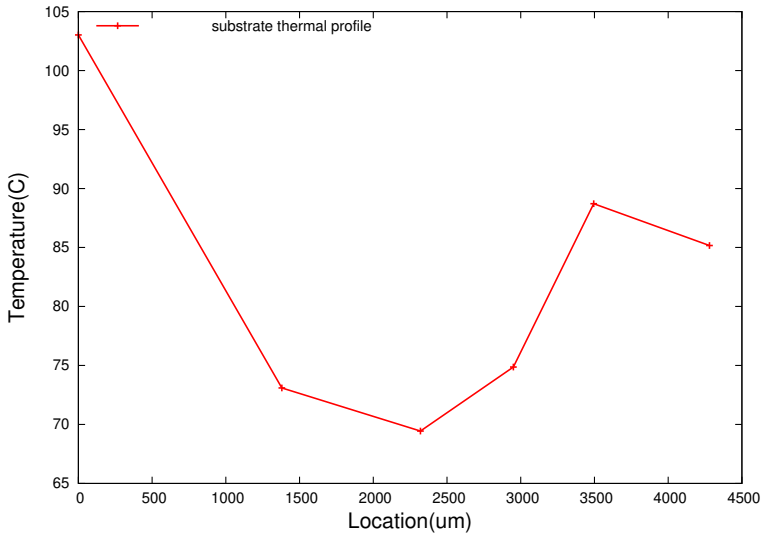


Figure 4.11: Substrate temperature profile along the length of an interconnect.

First, we illustrate our steps of performing temperature dependent wire delay estimation in an example. We extract the substrate thermal profile for one segment of a wire as illustrated in Figure 4.11 from one floorplan of *ami49*. Based on Eq. (4.9), the temperature profile of the global interconnect subject to substrate thermal profile in Figure 4.11 is computed and shown in Figure 4.12. We also included the maximum and average temperature of the interconnect for the purpose of comparison.

The temperature dependent delay (shown in Figure 4.13) is calculated using Eq. (4.9). The delay subject to maximum, average and room temperature are also shown in the figure. As can be seen in Figure 4.13, the wire delay obtained by using average and maximum temperature can be 25% over estimated from the real delay, while by using room temperature, obviously, resulted in significant underestimation.

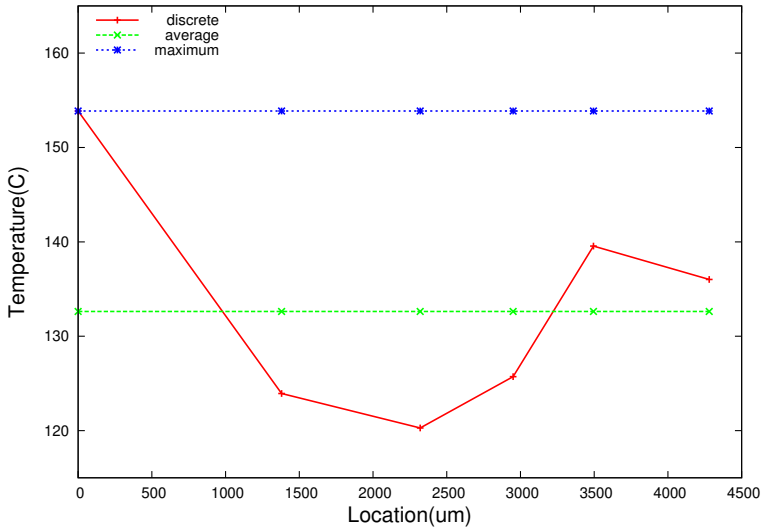


Figure 4.12: Interconnect temperature profile along the length.

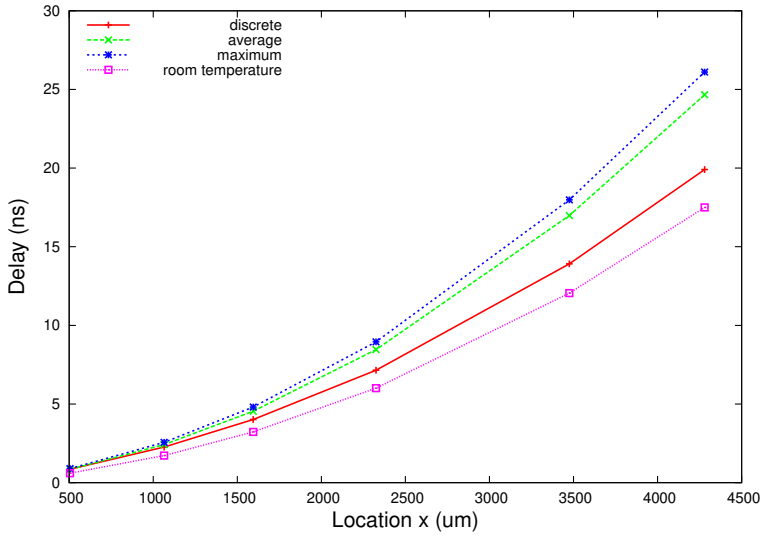


Figure 4.13: Temperature dependent wire delay along the length.

Next, we run HotFloorplan using traditional cost function with total wirelength as one of the evaluation metrics to collect statistics on thermal profiles in global interconnects (wires longer than $500\mu m$). In Figure 4.14, we show the statistics on the average temperature of all global interconnects. The height of a bar represents the number of interconnects having average temperature between one statistical point and the next. The substrate has a peak temperature of 104 °C and a temperature gradient of around 30 °C. Obviously, the interconnects have a significant higher average temperature than the substrate and a few of them even reached above 145 °C due to self-heating. For long wires, it is especially desirable to avoid routing above substrate regions at high temperature. Detouring around hotspot regions may increase wirelength and cause congestion, therefore an accurate overall analysis is necessary to assess different routing choices.

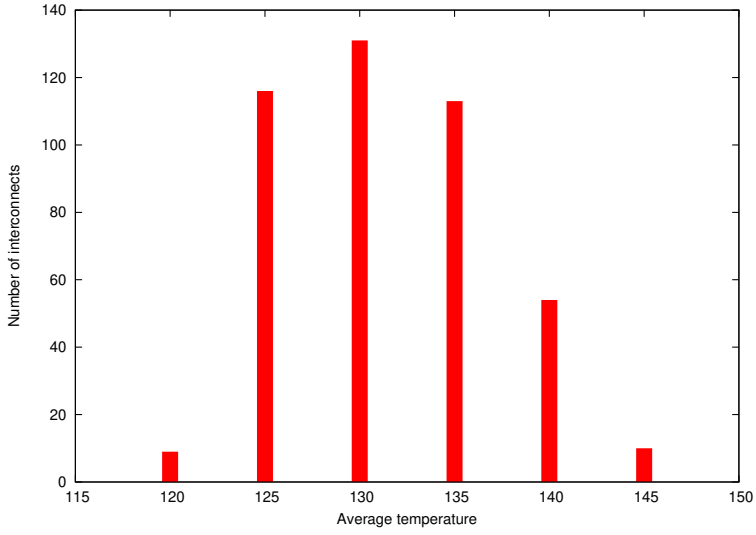


Figure 4.14: Statistics on average temperature in interconnects.

In Figure 4.15, we show the statistics on temperature gradient within global interconnects. Although temperature gradient in about half of the interconnects is less than 10 °C, more than 40% of interconnects do have a gradient larger than 20 °C. As we have described earlier, using average or maximum temperature can introduce a large error in delay estimation for these interconnects and therefore an accurate analysis for each wire is needed.

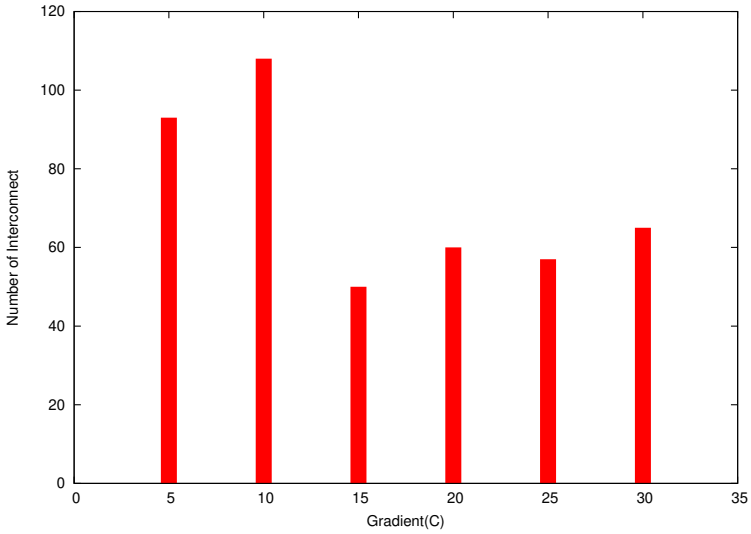


Figure 4.15: Statistics on temperature gradient in interconnects.

4.3 Summary

In this chapter, we described our SPICE simulation based temperature estimation method and discussed thermal behaviors and thermal ware design using a synthetic benchmark. High temperatures are caused by the large local power density in hotspots and therefore thermal coupling between hot cells and blocks need to be reduced in order to lower peak temperature. A preliminary analysis of the impact of interconnects on thermal distribution in the substrate is carried out and we conclude that the impact is marginal due to the small cross sectional area and thus large thermal resistance in interconnects.

In addition, we also presented a way to estimate temperature dependent delay in wires at an early design stage. Long wires, especially global wires, are subject to nonuniform temperature in the substrate. Consequently, a good wire plan at the floorplanning stage needs to take into account the temperature profile in wires as well.

Chapter 5

Power Density Reduction in Hotspots

As described in Equation2.7, junction temperature is largely dependent on power density. As described in Chapter 2, elevated temperatures in modern CMOS circuits are caused by increased power density (power consumption per unit area) as manufacturing technology scales to smaller geometries. The high power density in hotspots makes local temperature rises much faster than full chip heating. In order to lower peak temperature the power density in hotspots has to be reduced.

From the definition of power density, it can be seen that circuit designers can lower power density through reducing power consumption and/or increasing hotspot area. However, power consumption is dependent on circuit functionality and the room for reduction is not always large. In addition, low power techniques need to have large timing granularity in order to be effective for temperature reduction as we discussed in Section 2.5.

Alternatively, we approach the thermal problem by managing area. In this chapter, we describe two block level post placement temperature reduction techniques which reduce power density in hotspots through area management. Instead of uniformly increasing a block's area, the two techniques explicitly target area management in hotspots. We compare the efficiency of the two techniques

in reducing peak temperature against the general area enlarging method.

5.1 Motivation

Thermal management can be carried out at both design time (static) and run time (dynamic). In particular, physical level design time solutions such as thermal aware floorplanning and placement algorithms focus on reducing thermal coupling between blocks and cells that have a high power consumption. These algorithms try to spread high power consumption cells evenly on the die to avoid excessive local power densities that can lead to overheating. However, cells within hotspots are usually closely coupled and placing them apart inevitably increases the length of interconnect between these cells. As wire delay is gradually dominating cell delay in nanometer technologies, longer interconnect is making timing closure increasingly difficult.

Long wires not only introduce extra delay but also power due to the parasitic capacitance in wires. In Table 5.1, we list two net instances of different length to show the ratio between wire capacitance and pin capacitance. Pin capacitance refers to the capacitance associated with the output pin of the driving gate and the input pins of the load gate. All the values are reported from a design implemented in a 65 nm library using Synopsys' IC Comipler. Both nets have 3 pins, namely 1 driver pin and 2 load pins. Recall from Eq. (2.1) that dynamic power consumption is due to the charging and discharging of load capacitance. As can be seen in Table 5.1, wire capacitance is on the same order of magnitude as pin capacitance for short wires and dominate total capacitance for long wires. Capacitance in long wires can significantly impact the driving cell's power consumption.

Length(μm)	Pin cap.(fF)	Wire cap.(fF)	Ratio(Wire/Pin)
94.31	6.93	19.29	2.78
5.31	9.68	1.02	0.11

Table 5.1: Ratio between wire capacitance and pin capacitance.

Unlike design for low power, not many works have focused on smart management of area in the context of standard-cell designs with the explicit objective of reducing local power density. One possible reason is that in a traditional back-end design flow, a potential increase in area means increasing chip cost and reducing yield. As a result, most floorplaning and placement tools try to minimize total area by placing cells as compact as possible. For standard-cell designs, this is also made possible by the fine grain of the atomic elements of placement, i.e., library cells of the same height.

In modern design, on the contrary, the outline of a die is usually fixed while the component blocks and cells can be placed in a variable shape [81]. For the same total cell area, this means there are some whitespace[1] or area slack that can be exploited to alleviate the thermal problem. However, even a straightforward use of this area slack (e.g., by increasing the area or decreasing the row utilization factor[2] *UF* during placement) would result in a decrease in cell (and, in turn, power) density over the entire circuit block. Such a generalized, "blind" allocation, ignores the fact that peak temperature usually occurs in local hotspots which are clusters of cells having larger switching activity than the rest of the circuit. Consequently, it is desirable to reduce cell density mostly in hotspots instead of the whole circuit while maintaining (or even slightly increasing) cell density in cooler areas.

In this work, we propose two methods *empty row insertion* (*ERI*) and *hotspot diffusion* (*HSD*) for implementing a smart management of this additional area in such a way that peak temperature and temperature gradients can be reduced. Instead of devising completely new placement algorithms, we target the two approaches as plug-ins to mainstream industrial physical design tools. Modern design tools have already achieved good quality of placement for standard cell based designs. By optimizing the placed netlist we try to preserve circuit performance as much as possible while reducing peak temperature.

During the placement stage, performance driven algorithms find optimal locations for each cell to minimize critical path delay. As a result, coupling cells are placed close together, either on the same row or on neighboring rows to make interconnect wirelength as short as possible. This is illustrated in Figure 5.1 where we highlight a few cells and their connections along a signal path from a circuit layout.

To keep the increase in wirelength as small as possible when enlarging the area of hotspots, our methods try to preserve the relative positions of cells, which helps to avoid introducing significant overhead in circuit delay and power consumption. The two methods differ in the type of granularity at which the white space is allocated. In *ERI*, empty layout rows are inserted inside the hotspots, whereas in *HSD* individual "hot" cells are diffused into surrounding "cool" regions.

[1] Whitespace means space on a placement row that is not covered by functional cells.

[2] The utilization factor is defined as the ratio between core area and total area, thereby reducing utilization with the same core area will increase the total area.

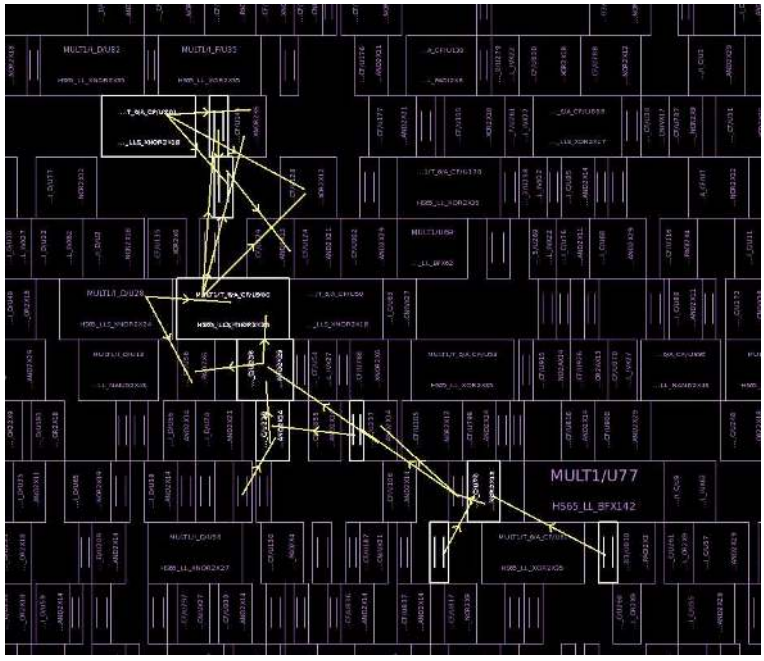


Figure 5.1: Coupled cells are placed close together.

5.2 Design Methodology

In this section we describe the two proposed schemes, *Empty Row Insertion (ERI)* and *HotSpot Diffusion (HSD)* as post-placement temperature reduction techniques. Both methods aim to reduce power density in hotspot regions, by reducing cell density while keeping delay and power overhead as little as possible. They work on synthesized and placed design, and can therefore exploit detailed spatial information about the cells, besides using accurate, post-layout estimation of area, delay, and power.

Figure 5.2 shows the flow of our methodology. The first step is to input the post placement netlist to the thermal simulator described in Section 4.1 to get an initial thermal map. The initial thermal map, together with the placed netlist and a user-specified area overhead, are processed by our area management tool which yields a modified placed netlist with better thermal properties using one of the two strategies. Our thermal management tool interacts directly with commercial physical design tools.

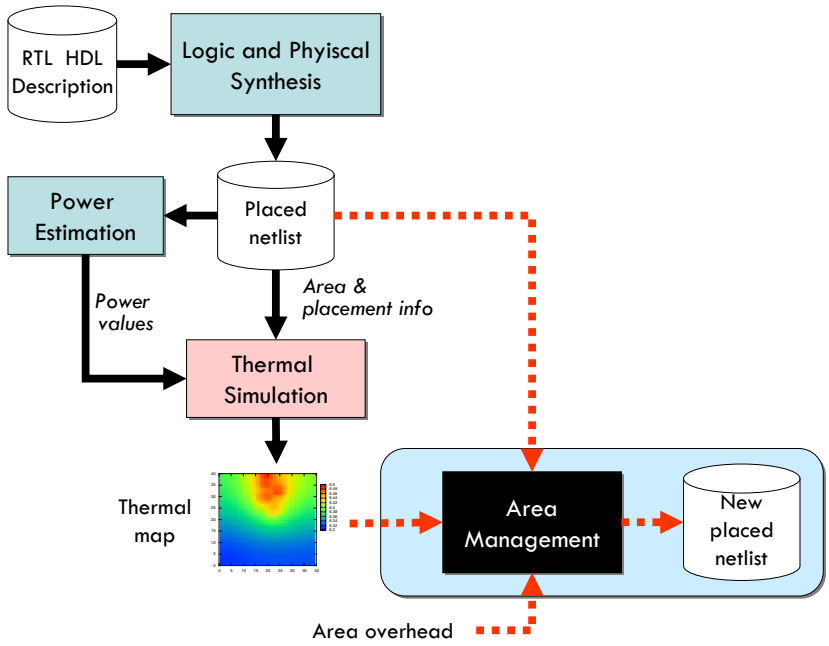


Figure 5.2: Synthesis and Post-Layout Flow of the Proposed Methodology.

In both methods, the available area overhead is filled with *filler cells* which do

not contain active transistors and consume zero power. They can guarantee electrical continuity of power and ground rails in each layout row. Filler cells are also designed to meet all the design rules imposed by the technology such as geometrical sizes, spacing and percentage of metal to guarantee a planar construction of stacked upper layers. This gives our methods a compliance with industrial semiconductor fabrication process.

Unlike other thermal aware placement algorithms which build their own timing and power estimation models, our methods interact directly with commercial physical design tools. We extract timing and power information and modify layout in Synopsys' IC Compiler, which enables our approaches a seamless integration into industrial design flow.

Moreover, the application of the proposed temperature reduction techniques does not limit the use of other thermal aware design methods. Instead they can be used as orthogonal methods which help to further reduce both peak temperature and temperature gradient.

5.2.1 Model Description and Problem Formulation

Figure 5.3 illustrates the layout and the corresponding thermal mesh. As mentioned in Chapter 4 a thermal cell usually covers several standard cells. In Table 5.2 and Table 5.3, we list some of the variables in the layout domain as well as the thermal domain. The characteristics of wide and concentrated hotspots are described below. M_{WHS} is the threshold ratio between the width of a hotspot and the total width W to define a wide hotspot.

Variable	*Description*
W	row width
H	row height
$R = \{r_0, ..., r_n\}$	rows in the layout
$G_i = \{g_0, ..., g_m\}$	standard gates in row_i
P_{g_i}	power consumption of gate i

Table 5.2: Variables in the layout (standard cells) domain.

Definition 5.1 *ADJACENT CELLS*
two thermal cells a and b, with coordinates (i_a, j_a) and (i_b, j_b) respectively, are adjacent if

$$(|i_a - i_b| = 1 \wedge |j_a - j_b| = 0) \vee (|i_a - i_b| = 0 \wedge |j_a - j_b| = 1) \qquad (5.1)$$

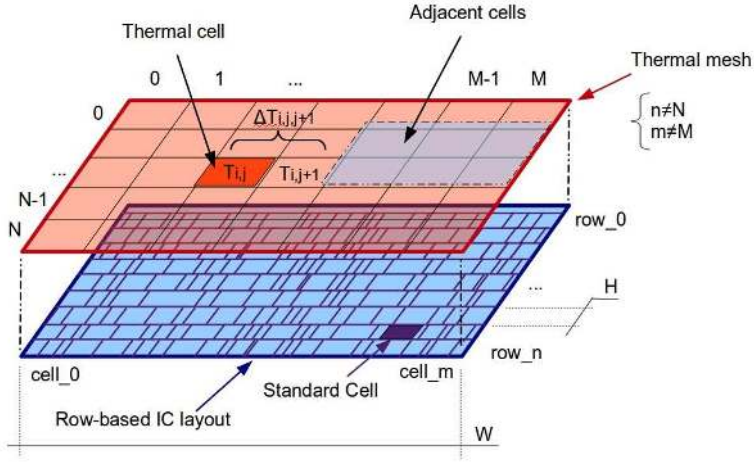


Figure 5.3: Thermal mesh VS layout.

Definition 5.2 *HOTSPOT*
set of adjacent thermal cells for which

$$T_{i,j} \geq \alpha T_{average} \quad with \quad \alpha > 1.0 \tag{5.2}$$

Definition 5.3 *WIDE HOTSPOT*
thermal region made up of $K \times L$ thermal cells, with $K > M_{WHS}$, adjacent thermal cells of coordinates (i, J) with $J = const \in [0, K]$ for which

$$T_{i,J} \geq \alpha T_{average} \quad with \quad \alpha > 1.0 \tag{5.3}$$

and

$$\Delta Th_{i,J} \leq \gamma T_{average} \quad with \quad \gamma < 1.0 \tag{5.4}$$

Definition 5.4 *CONCENTRATED HOTSPOT*
thermal region made up of $K \times L$, with $K < M_{WHS}$, adjacent thermal cells of coordinates (i, j) for which

$$T_{i,j} \geq \alpha T_{average} \quad with \quad \alpha > 1.0 \tag{5.5}$$

and

$$\Delta Th_{i,j} \leq \gamma T_{average} \quad with \quad \gamma < 1.0 \tag{5.6}$$

Variable	*Description*
A	section area of a thermal cell
M_{WHS}	threshold width ratio to define wide hotspot
$Pd_{i,j} = \frac{1}{A}\sum P_{gate_i}$; $gate_i \in cell_{i,j}$	power density of the thermal cell with coordinate i, j
$T_{i,j}$	temperature of the thermal cells i, j
$\Delta Th_{i,j} = \lvert T_{i,j} - T_{i,j+1} \rvert$	horizontal thermal gradient between adjacent thermal cells
$\Delta Tv_{i,j} = \lvert T_{i,j} - T_{i+1,j} \rvert$	vertical thermal gradient between adjacent thermal cells

Table 5.3: Variables in the thermaldomain.

$$\Delta Tv_{i,j} \leq \gamma T_{average} \quad with \quad \gamma < 1.0 \tag{5.7}$$

Problem Formulation Given a row-based IC layout L with area $A_L = (W \times n \times H)$, with n=*number of rows* (Figure 5.3), and the corresponding thermal mesh made up of $N \times M$ thermal cells; Let Q be the hot spot region, with $Q \subseteq L$, and size $(N_Q \times M_Q)$ thermal cells, allocate additional whitespace in order to minimize the power density of the hotspot region such that:

1. the introduced area overhead is lower than a user defined threshold: $A_L <= A_{Lnom}(1 + \delta_{area})$;

2. the delay overhead on the critical path is smaller than a user defined threshold: $Dp <= Dp_{nom}(1 + \delta_{timing})$.

5.2.2 Method 1: Empty Row Insertion for Wide Hotspots

Under this scheme, the granularity of the area slack insertion is a layout row. Conceptually it works as follows: in the area of a given hotspot, we insert an empty row between every adjacent row. The method is illustrtrated in Figure 5.4, where shaded rectangles colored in red indicate cells in the hotspot region and shaded rectangles colored in green indicate cells in the cool region. Figure 5.4.a shows the original layout with a wide hotspot region and Figure 5.4.b shows the layout after the *ERI* method is applied. As can be seen

in Figure 5.4.b, an empty row is inserted between each row in the hotspot region while the cool region remains the same as the original layout. This row of whitespace will be filled with filler cells which do not consume power such that we increase the area only of the hotspot region.

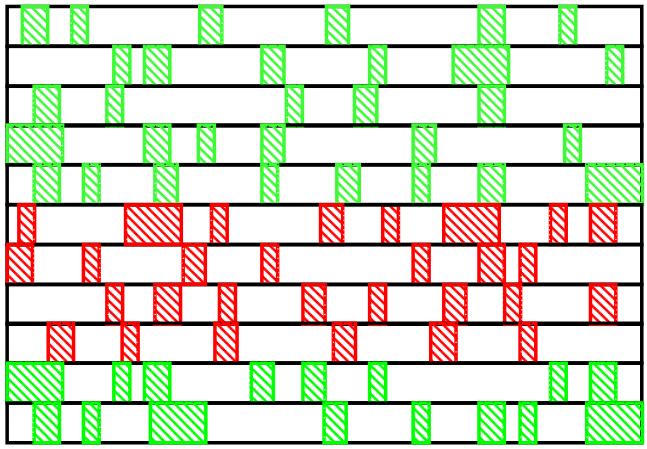
The algorithm of the method is illustrated in Algorithm 1. We first initialize the number of empty rows to the maximum available, which means the area overhead meets exactly with the area constraint. The increase in area might introduce too much extra wirelength causing delay constraint violation although the reduction in power density is the largest. In such a case, we revert to the original layout and decrease the number of rows to insert, which subsequently shrinks the area and reduces timing overhead. The procedure is repeated until timing constraint is not violated. In this way, we achieve power density reduction through empty row insertion without violating area or timing constraint.

Algorithm 1 Empty Row Insertion.

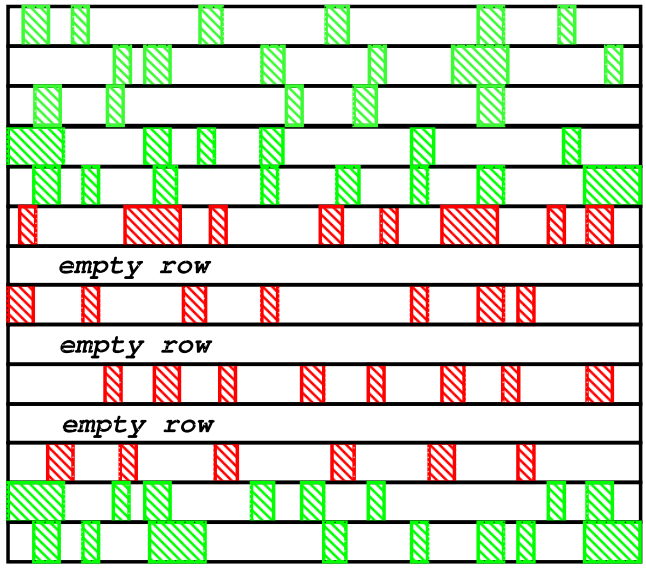
INPUT: Row-Based IC Layout
INPUT: Thermal-Map
INPUT: Timing Information
localize the wide hot-spot region Q
list all the rows belonging to the hot-spot region $row_i \in Q$
initialize number of empty rows $n_e = (A \times \delta_{area})/W$
insert n_e rows in the middle of hotspot
while Timing constraint is violated:
$\{Dp > Dp_{nom}(1 + \delta_{timing})\}$
do
  revert layout
  decrease number of empty rows n_e
  insert n_e rows in the middle of hotspot
  update power information
  update timing information
end while
generate new thermal map
OUTPUT: optimized layout

5.2.3 Method 2: Cell Diffusion for Concentrated Hotspots

For concentrated hotspots, only a small fraction of cells in a row belongs to the hotspot region, making the *ERI* method less efficient. Power density can be reduced by increasing the total circuit area, resulting in a uniform increase in area across the entire circuit. From a thermal point of view, it would be desirable to have a larger reduction in power density in hotspot regions than the cool regions.



a) original layout.



b) after *ERI*.

Figure 5.4: Empty row is inserted in between adjacent rows.

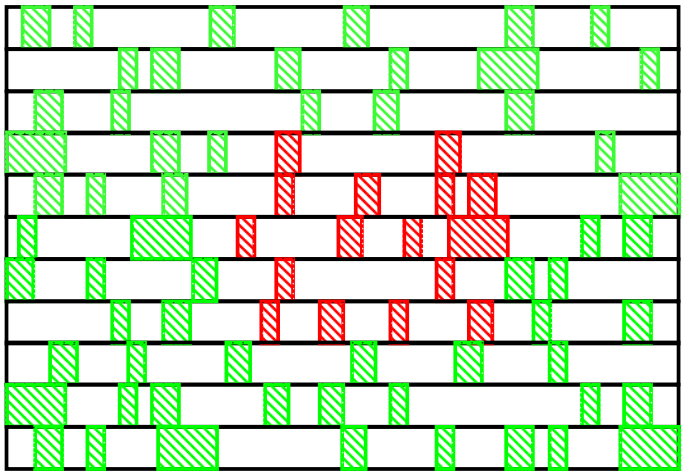
We first tried to push away the cells in the vicinity of a hotspot, which creates some whitespace around the hotspot. This can be achieved through the use of "bound" in placement tools. A placement "bound" only allows a set of user specified cells to be placed in a defined area, other cells will be moved out of the bound in a subsequent legalization step. This will enable cells in the hotspot to scatter over a larger area and power density is thus reduced. However, pushing a large number of cells away introduces significant timing overhead. This is because the legalization step does not perform any delay optimization but merely finds the nearest "legal" location for a cell pushed out of the bound.

Algorithm 2 HotSpot Diffusion.

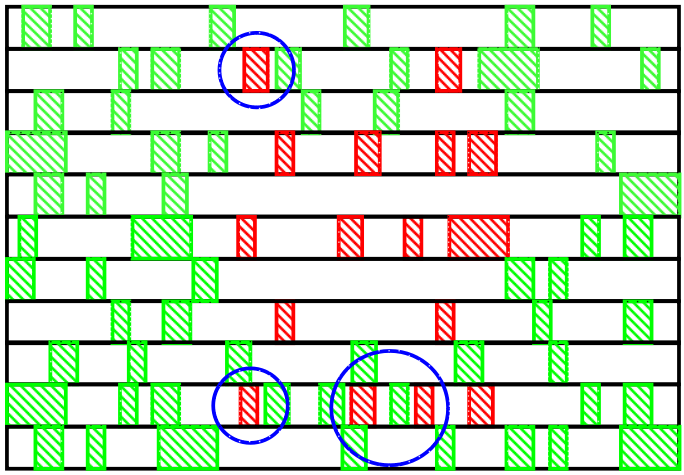
INPUT: Row-Based IC Layout
INPUT: Thermal-Map
INPUT: Timing Information
localize the concentrated hot-spot region, Q
list all the gates belonging to the hot-spot region, $g_i \in Q$
initialize number of rows to diffuse n_d
move n_d rows to the neighboring area
while Timing constraint is violated:
    $\{Dp > Dp_{nom}(1 + \delta_{timing})\}$
do
    revert layout
    decrease number of rows to diffuse n_d
    move n_d rows to the neighboring area
    update power information
    update timing information
end while
generate new thermal map
OUTPUT: optimized layout

For this reason, in this second method we increase the area of hotspot "in-site" of its original location, which means the cell cluster constituting a hotspot will grow in dimension as illustrated in Figure 5.5. Cells in the hotspot belonging to the same row are moved together and the original row ordering is preserved. In this way, we minimize the introduced wirelength between coupling cells that are placed in local clusters, either on the same row or adjacent rows.

The movement of hotspot cells might cause overlap with other cells in the cooler surrounding area. These overlaps are removed by performing placement legalization in the placement tool. As the legalization finds the nearest available location, the displacement of an overlapped cell from its original location is small. In Figure 5.5, the red cells within blue circles would otherwise cause overlaps if placement legalization is not performed. Consequently the hotspot grows in area and "diffuses" into neighboring area such that power density is reduced in well defined layout regions.



a) original layout.



b) after *HSD*.

Figure 5.5: Cells in the hotspot diffuse into surrounding area.

The algorithm of the *HotSpot Diffusion* method is described in Algorithm 2. As in the *Empty Row Insertion* method, we first try the maximum number of rows to diffuse, which will be vertically moved to neighboring regions in both directions. If the timing constraint is violated, we revert the layout and decrease the number of rows to diffuse.

5.3 Experiment Results

We used the design flow shown in Figure 5.2. The two methods, *Empty Row Insertion* and *HotSpot Diffusion* are implemented in Tcl scripts that can be executed directly in Synopsys' IC Compiler. The advantage of integrating the methods into physical design tools is that any changes in delay or power consumption can be immediately updated without the need to dump and import data files between different tools. Commands to interact with IC Compiler are shown in Appendix B.

To test our temperature reduction methodology, we performed a set of experiments using synthetic benchmarks, which are composed of several multipliers with different operand widths. The use of synthetic benchmarks makes it easy to control the location, shape and intensity of hotspots. The benchmark circuits are synthesized in the 65 nm standard cell library where cells have a uniform height of 2.6 μm, which is also the height of a placement row.

5.3.1 *Empty Row Insertion*

The first set of experiments are performed to compare the effectiveness of temperature reduction between *Empty Row Insertion* and increasing circuit area uniformly. We will refer to the latter as the *General* method. The benchmark circuit is designed to have hotspot of a rectangular shape (shown in Figure 5.6 colored in red). Table 5.4 lists the changes in critical delay, total area, peak temperature and dynamic power of the benchmark circuit. Temperature is reported as temperature rise above the ambient environment in °C.

The first row shows the reference implementation, where the circuit is floorplanned with the utilization factor (UF) set to 0.6. According to our experience, typical values for the UF is within the range of 0.4 to 0.7 in order to accommodate buffer cells and routing perturbations in the layout. With a UF larger than 0.7, the tool complains about over utilization and reports error in the placement stage. The peak temperature rise reached 21.7 °C above ambient environment

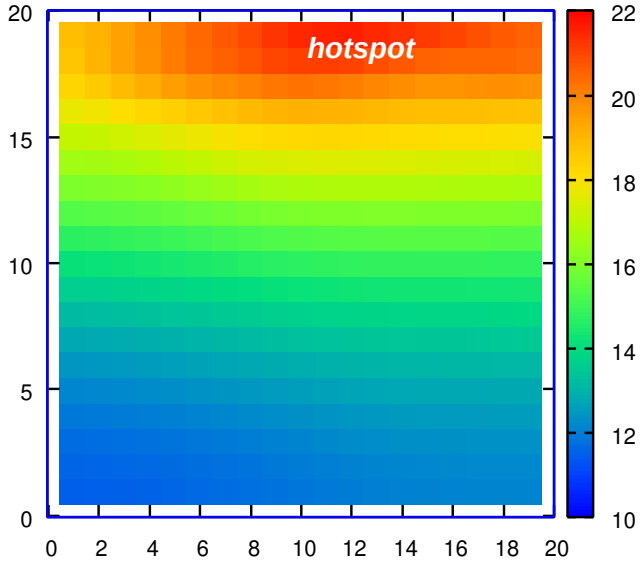


Figure 5.6: Thermal map of circuit with a wide hotspot. Temperatures are in °C.

in the reference design. In method 2 and 3 of Table 5.4 we relax the UF to 0.5 and 0.4, allowing larger area with the aim of reducing power density. As mentioned in the previous sections, relaxing the UF results in a uniform increase in area. The peak temperature in the hotspot dropped from 21.7 °C to 17.0 °C as the area increases. However, the decrease in temperature is achieved at the cost of a very large area overhead.

Method	*Delay*(*ns*)	$\Delta Delay$	*Area*(μm^2)	$\Delta Area$	*Temp*(°C)	$\Delta Temp$	ΔP_{dyn}
UF=0.6	0.95	-	413×413	-	21.7	-	-
UF=0.5	0.95	0.0%	453×453	20.0%	19.2	-11.5%	5.3%
UF=0.4	0.95	0.0%	506×506	49.5%	17.0	-21.7%	9.2%
ERI16	0.98	2.5%	413×453	9.4%	18.2	-16.3%	1.8%
ERI23	0.99	4.0%	413×473	14.5%	17.0	-21.5%	2.1%

Table 5.4: Temperature reduction in hotspot through *ERI*.

Next we apply the *ERI* method to the reference design. We performed two experiments inserting 16 and 23 rows which correspond to an area overhead of 9.4% and 14.5% respectively. In *ERI16*, the peak temperature dropped from 21.7 °C to 18.2 °C, resulting in a 16.3% reduction. In *ERI23*, the same amount of peak temperature reduction is achieved as in the case when the UF is set to

0.4. The area overhead, on the other hand, is much smaller for *ERI* showing the effectiveness of increasing area only in the hotspot region.

However, increasing the area may have a negative impact on circuit's performance as the interconnect between coupling cells can increase in length. One interesting observation that can be found in Table 5.4 is that as we relax the UF constraint, the critical delay stays more or less the same reflecting the effort made by the tool to meet timing constraint. For the *ERI* method, there is a slight increase in the circuit's delay as we are not performing any kind of delay optimization.

In fact, the extra wire length not only introduces signal delay but also dynamic power, as the wire capacitance is also increased. Although the increase in power is relatively small compared with the increase in area and does not cause temperature increase, we can still use it as one of the metrics to compare among the different methods. In Table 5.4, it can be seen that the increase in dynamic power for the *General* method is very significant while for the *ERI* method the increase is moderate. Especially, to achieve the same temperature reduction, the dynamic power overhead for the *General* method is almost five times more than the *ERI* method.

To compare the area efficiency of the *General* method and the *ERI* method, we can compute the ratio of $\Delta Temp$ to $\Delta Area$ in Table 5.4. For example, the ratio between $\Delta Temp$ to $\Delta Area$ for *ERI16* is 1.73 while for *UF=0.5* is 0.58. This means for every percentage of area increase, the *ERI* method get more than one percent of temperature reduction. The *General* method, on the other hand, is less efficient since the area is increased uniformly in the entire circuit.

5.3.2 *HotSpot Diffusion*

The second set of experiments are performed on a benchmark circuit having a concentrated hotspot. We use the *HotSpot Diffusion* method to reduce the power density in the hotspot region.

The experiment results are shown in Table 5.5. The reference circuit is implemented with the UF set to 0.5. We first relax the UF factor to 0.4 to increase the area. The peak temperature is reduced by 16.3% with an area overhead of 25.6% and a timing overhead of 2.91%. Then we apply the *HSD* method to further reduce the power density in the hotspot. As can be seen in Table 5.5, the peak temperature dropped by another 3% with total area stays the same. The delay increased by 2% due to the extra wirelength between cells.

$Method$	$Delay(ns)$	$\Delta Delay$	$Area(\mu m^2)$	$\Delta Area$	$Temp(°C)$	$\Delta Temp$
UF=0.5	1.03	-	232×232	-	2.94	-
UF=0.4	1.06	2.91%	260×260	25.6%	2.46	-16.3%
hsd	1.08	4.85%	260×260	25.6%	2.37	-19.4%

Table 5.5: Temperature reduction in hotspot through *HSD*.

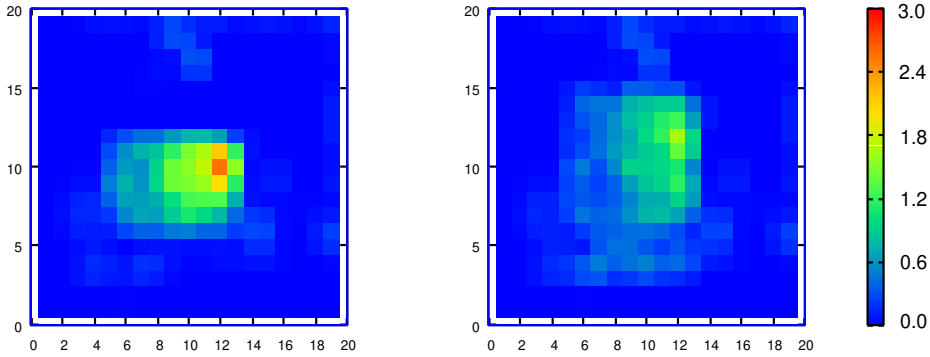


Figure 5.7: Comparison of power density before (left) and after (right) applying the *HSD* method. Power density is in W/mm^2.

The *HSD* method explicitly spreads cells in the hotspot over a larger area, which has a direct impact on the power profile of the circuit. In Figure 5.7, we show the power density map of the benchmark circuit before and after applying the *HSD* method. As cells of the hotspot are spreaded in a larger area, the peak power density is reduced as reflected by the lighter color in the right power density map.

The time complexity of the two proposed methods is mainly bounded by the update of timing and power information from the placement tool. For example in *ERI16* in Table 5.4, setting new locations of 16 rows (3,095 standard cells) in IC Compiler takes about 4 seconds while updating the timing and power information of the whole design (composed of 27,474 standard cells) takes approximately 12 seconds. In both methods, we start with the maximum number of available rows to move according to the area constraint. The row movement can induce performance degradation due to introduced wirelength. Therefore, if the timing budget is quite tight, to find the number of rows to move n_d that satisfy both the area and timing constraint will require more iterations, which results in long exploration time. Suppose 10 iterations are needed in the *ERI16* experiment, then the execution time of the method would be around 3 minutes.

5.4 Summary

In this chapter, we presented two post placement stage temperature reduction techniques, *ERI* and *HSD*. The two methods reduce power density in a hotspot by increasing its area in an efficient way. Due to the increase in distance between coupling cells, extra wirelength is inevitably introduced. By observing that coupling cells are placed on the same row or adjacent rows, our methods minimize the introduced wirelength through preserving relative cell positions in the final layout. The experiment results show that the two methods are both more efficient than generally increasing a block's area in reducing peak temperature.

Chapter 6

Energy and Thermal Aware Design in FPU

As we have described in Chapter 3, complex Floating-Point (FP) operations such as division can be implemented in several ways based on different algorithms. The FP division operation, although infrequent in percentage of instructions, could contribute a significant portion to the total energy consumption due to its long latency. In this chapter, we analyze and compare the power and energy consumption of different algorithms and implementations for FP division and discuss the potential of digit-recurrence dividers from a thermal perspective.

Based on instruction statistics from a scientific application, we compare the total energy consumption of all floating point operations between different implementations for division.

For applications having intensive FP operations, the placement of power efficient dividers could be exploited to reduce thermal diffusion from hotspots like the Fused Multiply-Add (FMA) unit to temperature sensitive blocks like on-chip cache. The power consumption in leakage dominant block can increase exponentially with temperature and thus can be reduced from lowering its temperature. Consequently, a larger gain in total energy can be achieved by including digit-recurrence dividers in the Floating-Point Unit (FPU) design.

We will first describe the common metric, energy per operation, which is used to compare and evaluate the different algorithms and implementations for division.

6.1 Energy Metrics

At the algorithm level of design abstraction, a problem can usually be approached by different methods. For example, an application can be implemented in different ways with different timing and latency. When power is a primary design constraint, a common measure of the power and energy dissipation is required in order to evaluate and compare different algorithms.

Because the algorithms are in general different and the latency of the operations varies from case to case, it is convenient to have a measure of the energy dissipated to complete an operation. This energy-per-operation is given by

$$E_{op} = \int_{t_{op}} vi \; dt \qquad [J] \tag{6.1}$$

where t_{op} is the time elapsed to perform the operation. Operations are usually performed in more than one cycle (in n cycles) of clock period T_C and the expression of t_{op} is typically $t_{op} = T_C \times n$. By dividing the energy-per-operation by the number of cycles we obtain the energy-per-cycle

$$E_{pc} = \frac{E_{op}}{n} \qquad [J]. \tag{6.2}$$

This term is proportional to the average power dissipation that can be expressed in its equivalent forms:

$$P_{ave} = \frac{E_{pc}}{T_C} = E_{pc} f = \frac{E_{op}}{t_{op}} = V_{DD} I_{ave} \quad [W] \tag{6.3}$$

where V_{DD} is the unit supply voltage and I_{ave} its average current. By combining (6.3) and t_{op} we obtain

$$E_{op} = P_{ave} \times T_C \times n \qquad [J] \tag{6.4}$$

The term P_{ave} has an impact on the sizing of the power grid in the chip and on the die temperature gradient, while the term E_{op} impacts the battery lifetime.

6.2 Implementation of the FP-units

To analyze the impact on power dissipation of the different units and to evaluate the different approaches to division, we implemented the four units for binary64:

- **FMA** is the fused multiply-add unit of Figure 3.7 modified to execute the Newton-Raphson (NR) division algorithm.
- **FMA soft** is the fused multiply-add unit of Figure 3.3 to execute the NR division algorithm in software.
- **r16div** is the radix-16 divide unit of Figure 3.6 completed with convert-and-round unit and sign and exponent computation and update.
- **Penryn** is the division unit of Figure 3.5 modified to handle binary64 only. That is, the recurrence is composed by two cascaded radix-4 stages (Figure 3.4.b) plus the same initialization, convert-and-round unit and sign and exponent processing as the **r16div**.

All units are synthesized using the design flow described in 4.1.3 to obtain the maximum speed. Because in our flow we do not use two-phase clocks, for the **Penryn** implementation we cascaded the two radix-4 stages of Figure 3.5 into a single clock cycle.

Power estimation is based on randomly generated input vectors conformed to IEEE 754 binary64 format. The synthesis results are summarized in Table 6.1, where T_c is the minimum clock period, $Cycles$ is the number of clock cycles to finish an FP operation and $Latency$ is the total delay from applying inputs to obtaining results, that is $T_c \times Cycles$. The average power dissipation P_{ave} is normalized for all units at 1 GHz. The power dissipation data for the FMA unit are divided by operation.

As described in Section 3.4, the FMA unit has four pipeline stages. For the three operations: ADD, MUL and MA fused, the power was measured with the pipeline full to get the worst case power dissipation necessary to characterize the thermal behavior (Section 6.4) of the units. For division operations, being an iterative algorithm, a new instruction has to wait until the previous instruction finished execution and the power was measured per operation. This explains why the value P_{ave} for FMA DIV is smaller than the other FMA cases such as MUL and MA fused.

From the data of Table 6.1, it can be seen that an ADD operation in a FMA consumes much less power than a MUL operation but the latency is the same.

Unit	T_c $[ns]$	*Cycles*	*Latency* $[ns]$	*Area* $[\mu m^2]$	P_{ave} $[mW]$	E_{op} $[pJ]$
FMA ADD	0.75	4	3.0	114,816	49.7	198.8
FMA MUL		4	3.0		205.2	820.8
FMA MA fused		4	3.0		223.6	894.4
FMA DIV		18	13.5		131.6*	2368.8
FMA DIV soft		41	30.8	94130	55.7*	2283.7
Penryn	0.75	18	13.5	21229	31.5	567.0
r16div	0.75	18	13.5	14054	20.8	374.4

P_{ave} is average power measured at 1 GHz.
* Iterative algorithm, pipeline not full.

Table 6.1: Results of implementations.

A more advanced FMA architecture designed to optimize latency for addition is discussed in [82], however in this work, we used the basic FMA architecture since our main purpose is to compare alternatives for implementing division. For floating-point division, it is clear that the digit-recurrence approach (**Penryn** and **r16div**) is much more convenient in terms of latency, area and power dissipation. For example, with the same latency, FMA DIV consumes more than 4 times power than **Penryn** and more than 6 times than **r16div**. The **r16div** scheme has the same latency as the **Penryn** unit for division. It can probably be clocked with the same scheme used in Intel Core2 FP-units and provide the same throughput at reduced area and power dissipation.

In terms of energy per operation, the results in Table 6.1 show that in a FMA unit, the ratio of E_{op} between ADD and MUL is about 1 / 4 and MA fused consumes slightly more than MUL operations. With the same latency, the energy per operation E_{op} is proportional to average power P_{ave}, thus implementing division in a FMA unit consumes much more energy than **Penryn** and **r16div**. On the other hand, although DIV operation in **Penryn** and **r16div** have the lowest power consumption, the energy consumed in these units are much larger than ADD operations due to the long latency in DIV operations. The latter observation motivates the optimization for power consumption in division.

The only argument in favor of the FMA DIV is that division is much less frequent than addition and multiplication and as a result a larger power dissipation for the operation can be tolerated. The software implementation of division in FMA has a even longer latency (as shown in Table 6.1), since each iteration has to go through all the pipeline stages and intermediate results have to be

saved in register files. The E_{op} for the hardware and software implementations of division in FMA is almost the same, but the former has a much shorter latency. Therefore in all the experiment results shown hereafter, we refer to the modified FMA with hardware support for division when comparing division by multiplication in a FMA and division by digit-recurrence approaches.

6.3 Energy Consumption in FP-operations

6.3.1 Instruction Mix

In [83], the average frequency of floating-point operations in the SPECfp92 benchmark suite is reported. The most common instructions are multiply and add with MUL accounting for 37% and ADD for 55%. Moreover, the FP adder consumes nearly 50% of the multiply results which explains why fused multiply-add units are often used in modern processors. Table 6.2 summarizes the instruction mix. The first column shows the mix when none of the MUL and ADD instructions are fused. The second column shows the mix when 50% of the MUL instructions are fused with ADD.

	not fused	*fused*
ADD	55.0%	44.8%
MUL	37.0%	22.7%
FMA	0%	22.7%
DIV	3.0%	3.7%
OTHERS	5.0%	6.1%

Table 6.2: Instruction mix.

Based on the implementation data in Table 6.1, we can obtain the clock cycle distribution for all FP operations (shown in Figure 6.1) with the instruction mix in Table 6.2. Due to the much longer latency of DIV operation, the percentage of cycles spent in DIV operation is significantly larger than its percentage of instructions, which emphasizes the importance of optimizing DIV operation in terms of delay, power and energy consumption.

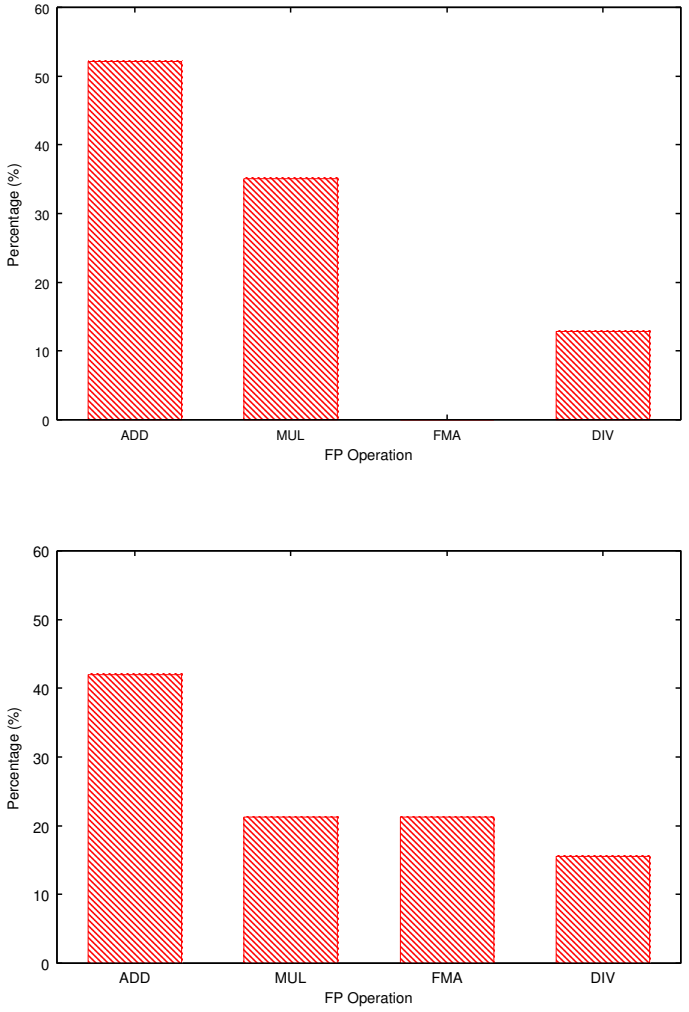


Figure 6.1: Clock cycle distribution for all FP operations: not fused (above) and fused (below).

6.3.2 Energy Consumption in a Scientific Application

To compare the energy consumption of units with different latencies a suitable metric is the energy consumption per operation (E_{op}), as described in Eq. (6.4).

Due to the significant reduction in E_{op} for division as shown in Table 6.1, we use a digit-recurrence (Penryn or r16div) divider for FP-division. To compare the energy savings in an application between division by FMA and by digit-recurrence, we use the SPICE benchmark which has a rather high percentage of divisions [84].

Since the percentage of MUL instructions that can be fused with ADD is not mentioned in [84], we list the results of the comparison in Table 6.3 to show the upper and lower bound of the energy consumption. In Table 6.3, the results are obtained by assuming two extreme situations that is when none of the MUL instructions can be fused with ADD (top), which gives the upper bound and when all MUL can be fused with ADD (bottom), which gives the lower bound. In all three setups (*Penryn*, *r16div* and *FMA*), the MUL and ADD operations are implemented by the FMA unit. The DIV operation, on the other hand, is implemented in Penryn, r16div or FMA, respectively.

	Percentage	*Penryn* [*pJ*]	*r16div* [*pJ*]	*FMA* [*pJ*]
DIV	8.0%	45.4	30.0	190.0
ADD	45.0%	90.3		
MUL	26.0%	213.8		
Fused MA	0.0%	–		
Total	79.0%	349.2	333.8	493.8

	Percentage	*Penryn* [*pJ*]	*r16div* [*pJ*]	*FMA* [*pJ*]
DIV	8.0%	45.4	30.0	190.0
ADD	19.0%	38.0		
MUL	0.0%	–		
Fused MA	26.0%	232.9		
Total	53.0%	316.3	300.9	460.9

Table 6.3: Energy consumption in SPICE without (top) and with (bottom) fused multiply-add.

Note that the comparison is based on the four FP arithmetic operations (ADD,

MUL, Fused MA, DIV) , which explains why *Percentage* of "Total" is not 100% in Table 6.3. Due to the reduction in the number of instructions by fusing MUL and ADD, there is a reduction in the total number of instructions of the whole application, which is reflected in a smaller *Percentage* of "Total" in Table 6.3 (bottom) since we use the top as baseline reference.

Although the division operation is much less frequent than addition and multiplication, it dissipates a significant proportion of total energy in all arithmetic operations (around 10% if implemented in digit-recurrence dividers and 40% if implemented in FMA). In both cases (fused MA or not) there is a significant reduction (around 30%) in energy consumption by using a digit-recurrence divider to implement binary64 division. Due to the low percentage of DIV instructions, the **r16div** unit only consumes slightly less energy than *Penryn* although its power consumption is 1/3 less than **Penryn**.

6.4 Thermal Analysis

In the previous section, we show that using digit-recurrence dividers for DIV operations consumes much less power and energy than using FMAs. From the thermal perspective, the low power divider can be placed near the FMA to reduce the heat flux in the FMA through lateral heat diffusion. In this way, the divider block help further reduce the high temperature rise in the FMA. Similar floorplan strategies can be found in high-end multicore processors where caches are placed beside cores [85] to partially mitigate thermal problems arose from the excessive heat generated inside the cores.

To perform thermal analysis, we use the model proposed in Chapter 4, which consists of a conventional RC model of the heat conduction paths around each thermal element. The FMA unit is laid out with an area of $437\,\mu m \times 437\,\mu m$ and the **r16div** unit is laid out with an area of $437\,\mu m \times 44\,\mu m$, which is about 1/10 of the FMA.

Figure 6.2 shows the impact on temperature distribution when a **r16div** is placed next to a FMA. Temperatures shown in the figure indicate the rise above the ambient temperature. The figure above is a thermal map of the FMA unit alone and the figure below shows the thermal map when a **r16div** unit is placed next to the FMA. Power consumption in both units are estimated based on workload characterized by the instruction mix with fused MA as shown in Table 6.2. When DIV operations are executed in the **r16div** unit, the average power consumption for all FP operations decreased from 132.44 mW to 115.18 mW resulting in a 13.0% reduction.

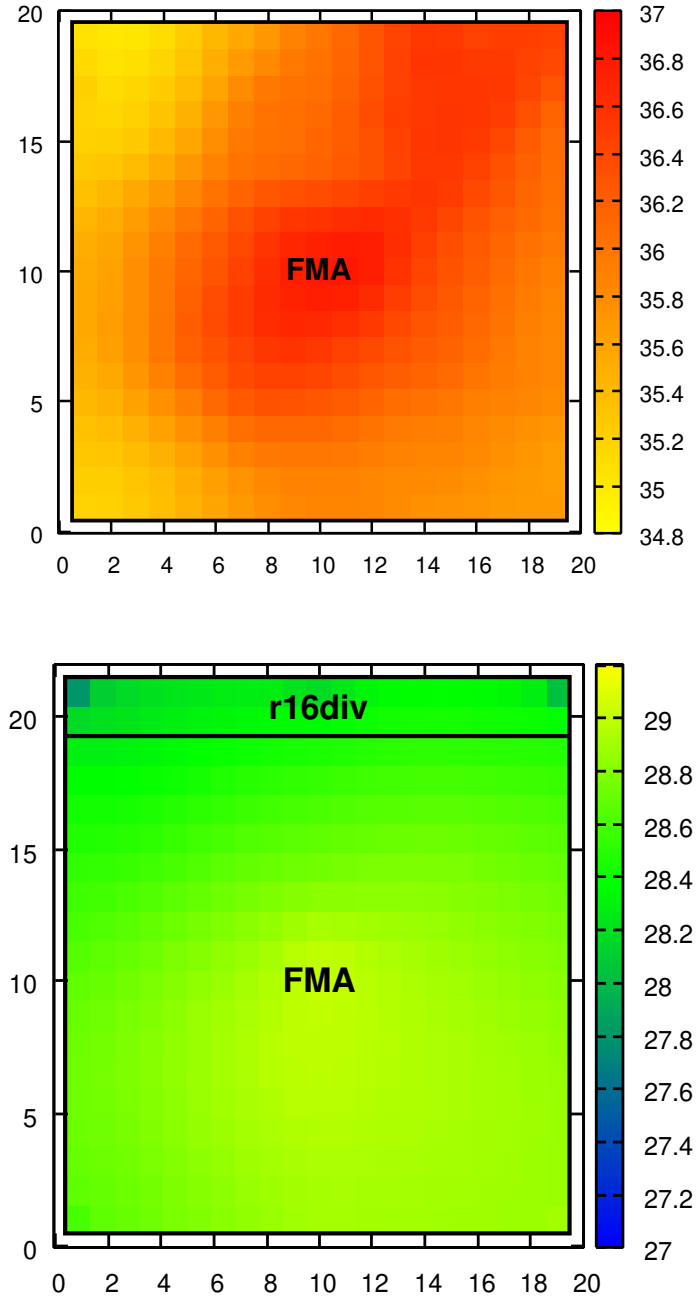


Figure 6.2: Comparison of thermal profiles: FMA alone (above) and FMA plus r16div (below). Temperatures are in °C. Note that the scale does not overlap.

In the FMA unit, the hotspots correspond to the tree multiplier and the CPA, which is reflected by the high temperature colored in red in the thermal map. The peak temperature rise reached 36.9°C above ambient. By putting the divider beside the multiplier the peak temperature rise dropped from 36.9°C to 29.6°C and the average temperature reduced from 36.0°C to 28.7°C.

The peak temperature reduction in the FMA is achieved from two factors. First, due to the offload of DIV operations to **r16div**, the average power consumption in the FMA is reduced, effectively lowering the power density in the FMA. Second, the **r16div** introduced a 10% increase in total area, which increased the cross sectional area of the heat conducting path and reduced the thermal resistance from junction to ambient. The above two factors result in a peak temperature reduction of 20%.

6.5 Leakage Optimization in Caches

In previous sections, we show that power and energy can be significantly reduced by using a digit-recurrence unit for division operations and the divider can also mitigate thermal problems in the FMA. In fact, the placement of these digit-recurrence units can be exploited to limit the amount of heat diffusion from hot blocks such as FMAs to caches as well. Leakage is the dominant fraction of power consumption in caches [86], so cache is more sensitive to temperature increase in terms of power. In this section, we use a leakage model to characterize the temperature reduction and energy savings in a cache block in a multi-core like architecture.

6.5.1 Leakage Power Model

The leakage power is mainly due to gate tunneling and subthreshold leakage and has a large temperature dependency as we have described in Section 2.1.

While gate leakage is relatively temperature independent, subthreshold leakage has an exponential dependence on temperature. The SPICE[1] BSIM4 model is a very elaborate model of modern transistors behavior. We use the BSIM4 models provided by the standard cell library to characterize the temperature dependency of the cell's leakage power.

[1]Here we refer to the actual electrical level simulations run by SPICE, not the benchmark used in Section 6.3.

The average power consumption of a 4-bit carry propagate adder is measured with the input fixed to obtain the static dissipation. Squares in Figure 6.3 show the normalized leakage reported by SPICE as temperature is swept from 20 °C to 150 °C, which is the typical range of operating temperatures. In order to approximate the exponential increase of leakage, we use a fourth order polynomial to accurately fit the SPICE reported data.

$$P_{leak} = P_{leak0}\{a(T-T_0)^4 + b(T-T_0)^3 + c(T-T_0)^2 + d(T-T_0) + e\} \quad (6.5)$$

The model shall be used in the next section to calculate the amount of leakage reduction obtained from decrease in temperature. The polynomial is plotted in Figure 6.3 using a curve. As shown in the figure, leakage power more than doubles for every 30 °C rise in temperature. At high temperatures the rate of increase in leakage power is very fast, which means that even a few degrees of increase in temperature can induce a large amount of leakage power. Consequently, containing the high temperature rise is very important in limiting static dissipation in leakage dominant blocks such as on-chip cache.

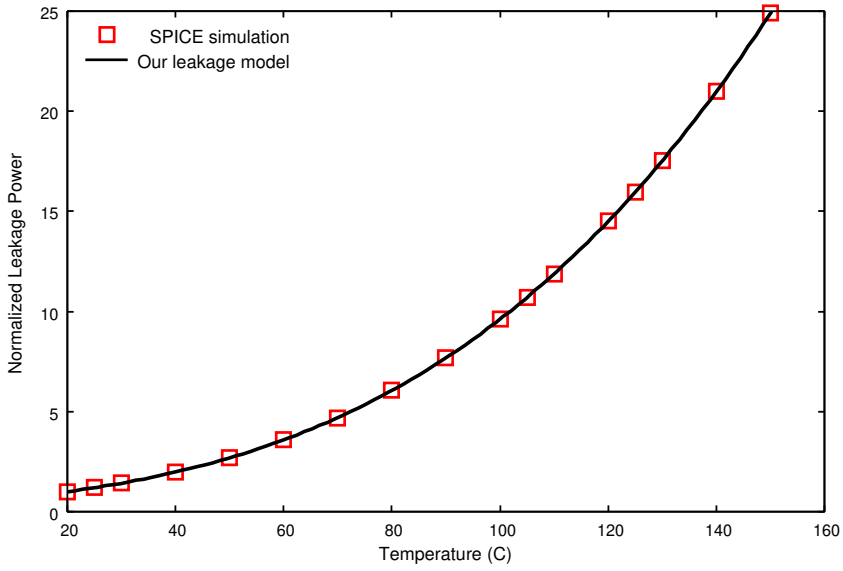


Figure 6.3: Temperature dependent leakage model derived from SPICE simulation.

6.5.2 Experiment Setup

In our experiments we made the assumption based on [86] that 70% of the total power consumption in cache is from leakage. We performed three experiments using five different configurations of FMA, **r16div** and a cache block. The configurations are summarized in Table 6.4 with config1 and config3 as baseline references.

	FMA	*r16div*
config1	2	0
config2	2	2
config3	4	0
config4	4	2
config5	4	4

Table 6.4: Number of units in each configuration.

The size of the cache block is 875 $\mu m \times$ 160 μm, which can accommodate approximately 16 KB in a 65 nm process according to our estimation[2]. The physical dimension of the FMA unit and the **r16div** unit are the same as in Section 6.4 with the FMA 10 times larger than the **r16div**.

6.5.3 Experiment Results

In Figure 6.4 we show the impact on temperature distribution when two **r16div** units are placed in between the FMAs and a cache block. Temperatures shown in the figure indicate the rise above the ambient temperature which is 50°C in our experiments. Again, power consumption in the FMA and divider units are estimated based on workload characterized by the instruction mix with fused MA as shown in Table 6.2. The right figure has more thermal cells in the grid due to its increased area.

The area occupied by the FMAs has a higher temperature which is reflected by the red (dark) color. The div units reduce the average temperature rise in the cache block from 23.1°C to 18.1°C. This means the absolute temperature in the cache block reduces from 73.1°C to 68.1°C. From our temperature dependent

[2]In [2], a 6T SRAM cell is reported to occupy an area of $20 \times 22\lambda$ in a 130 nm process, where λ is half of the channel length. In a 65 nm process, λ is approximately 33 nm and the area of a SRAM cell would be around 0.5 μm^2.

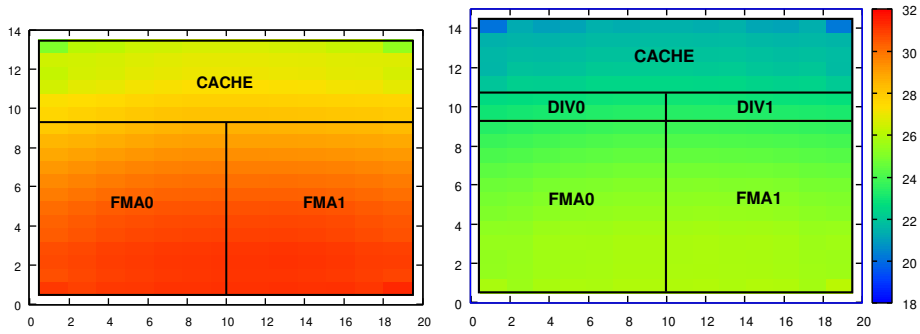


Figure 6.4: Thermal profiles in config1 and config2: FMAs alone (left) and FMAs plus 2 **r16div** (right). Temperatures are in °C.

leakage model, the decrease results in a 12.3% reduction in leakage. The total power consumption in the cache block is therefore reduced by 8.6%.

In Figure 6.5 we show a larger circuit composed of a cache block and four FMA units. The size of the cache block is the same as before. Again, we use **r16div** for division operations to save energy and reduce average temperature in cache. We first try to use two dividers as shown in Figure 6.5 left where the average temperature in the cache block dropped by 4.8°C and leakage is reduced by 11.7%. Next, we use four dividers in *config5* as shown in Figure 6.5 right. Due to the larger area introduced by the dividers, the average temperature rise in the cache block is reduced by 7.3°C from 25.8°C to 18.5°C. The leakage is reduced by 17.1% and the total power consumption decreased by 12.0% in the cache.

	Grid Size	T_{max}	T_{min}	T_{cache}	ΔP_{leak}	ΔP_{total}
config1	20×14	33.8	21.0	23.1	-	-
config2	20×15	28.3	16.5	18.1	-12.3%	-8.6%
config3	20×24	42.1	23.4	25.7	-	-
config4	20×25	35.8	19.2	20.9	-11.7%	-8.2%
config5	20×26	35.5	17.0	18.5	-17.1%	-12.0%

Table 6.5: Temperature and power reductions in cache.

Table 6.5 summarizes the experiment results where *Grid Size* is the size of the thermal grid when determining temperature distribution in the whole system. T_{max} is the peak temperature rise in the system and T_{min} is the minimum temperature rise in the system. T_{cache} is the average temperature rise in the cache block. ΔP_{leak} and ΔP_{total} are the percentage of leakage and total power

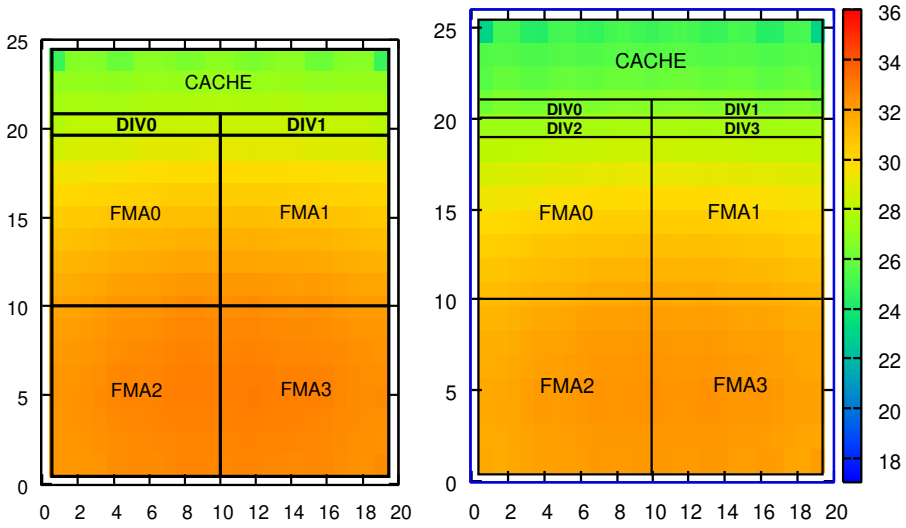


Figure 6.5: Thermal profiles in config4 and config5: FMAs plus 2 **r16div** (left) and FMAs plus 4 **r16div** (right). Temperatures are in °C.

reduction in the cache block due to the decrease in temperature.

It is obvious that by using more divider units we can obtain a larger reduction in cache temperature and thus leakage. However, the cost is increased die area which might not be desirable. It should be noted that the divider units are power efficient components for division operations instead of plain empty space. Given the power and thermal properties of the divider block, designers can decide the number of units to use based on the frequencies of division operations in the application.

6.6 Summary

In this chapter, we have compared different implementation of floating-point division in terms of power and energy per operation. Digit-recurrence dividers have been shown to dissipate much less power than multiplicative algorithms such as Newton-Raphson in a FMA. A significant amount of energy can be saved by including a digit-recurrence divider in the FPU due to the much longer latency of division operations.

We have also shown how the low power division unit can be used to mitigate

thermal problems in the FMA and the cache. Similar to the *Empty Row Insertion* method that we have described in Chapter 5, the **r16div** introduced area overhead but reduced power density in the hotspots. The difference is that the extra area from **r16div** is not dummy cells but power and energy efficient functional unit for division.

Chapter 7

Perspective

In this chapter, we will highlight some of the perspectives enabled through the work presented in the dissertation. In specific, we will discuss about temperature aware wire planning in the early design stage, delay overhead optimization in the proposed temperature reduction techniques and incorporating thermal analysis into existing Negative Biased Temperature Instability (NBTI) induced aging analysis modeling tools.

7.1 Thermal Aware Planning and Routing for Global Wires

The temperature dependent wire delay model as described in Section 4.2 can be incorporated into wire planning and global routing algorithms. The spatial-temporal variation in temperature has already caught attention in the area of clock tree synthesis and optimization. This is because the temperature induced clock skew, if ignored, can in the worst case result in system failure. As technology further scales, the peak temperature rise and temperature variation in global wires can become more significant.

The thermally induced delay variation in signal wires also has to be analyzed

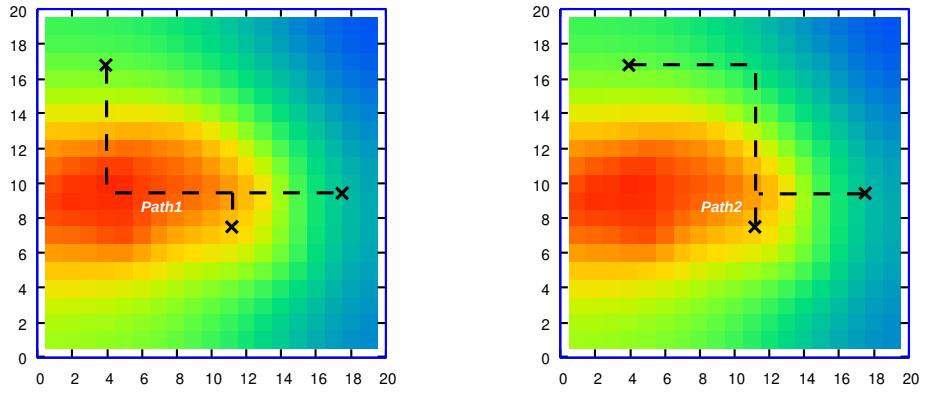


Figure 7.1: Thermal aware global routing.

and considered in routing algorithms to avoid performance degradation. At the early design stage, routes of long wires can be planned based on the layout and the initial thermal analysis, taking into account the temperature variation on the substrate and at the same time avoiding over congestion when detouring thermal hotspots. The exact route and dimension of wires can then be determined during the global and detailed routing stages. Figure 7.1 illustrates two routing solutions for a three terminal net with the crosses representing the terminals and dashed lines representing the routes. *Path1* (left) has the same wirelength as *Path2* (right), but it is subject to higher temperature. In terms of delay and reliability, *Path2* is more favorable than *Path1*.

7.2 Delay Overhead Optimization in *ERI* and *HSD* Methods

The two methods proposed in Chapter 5, namely *ERI* and *HSD*, results in a larger peak temperature reduction than the general area enlargement method. In addition, the introduced dynamic power overhead due to the increase in wirelength is smaller than the general method. There is, however, a small amount of timing overhead, although less than 5% even in the worst case.

By identifying timing critical paths, delay optimization can be performed on these paths to reduce the timing overhead. The delay overhead is mainly caused by the increase in wirelength, especially in wires connecting cells that were originally placed far away vertically. These paths need optimization because the

proposed methods increase the area occupied by the hotspot vertically, which exacerbates their delay. Optimization methods can include rearranging the location of cells on the critical paths, replacing driving cells of long wires with larger drive strength and other methods.

7.3 NBTI Analysis with Detailed Spatial Thermal Distribution

As we have discussed in Chapter 2 Section 2.4, the reliability and lifetime of microelectronic systems can be significantly degraded by the eleveated temperature and thermal gradients. In particular, the Negative Biased Temperature Instability (NBTI) effect can lead to a shift in the PMOS transistor's threshold voltage up to 50 mV over time and severely degrade circuit performance. The NBTI induced voltage shift is temperature sensitive and increases much faster at high temperatures. Information of a circuit's operating temperature is therefore fundamental to aging analysis caused by NBTI effect.

The scenario is very similar to the wire delay problem that we just discussed, ignoring the temperature variation along the signal paths can lead to large errors. Transistors subject to different temperatures exhibit different speed of NBTI induced aging, e.g. transistors in the hot region age faster than transistors in the cool region. The difference from the wire delay problem is that the target of analysis is the transistors on the substrate instead of wires in the metal layers. The accuracy of existing NBTI models, many of which use a single worst case temperature for the whole circuit, can be improved by utilizing the detailed temperature distribution to perform layout and thermal aware analysis.

Chapter 8

Conclusion

The goal of this work is to investigate power and thermal management techniques in nanometer technologies. The high junction temperature caused by the increased power density in modern VLSI chips has negative impact on many characteristics of a CMOS circuit such as delay, static power dissipation and reliability. The need for effective power and thermal management is ever increasing as manufacturing technology further scales to smaller geometries.

To lower peak temperature, the high local power density in hotspots has to be reduced. In turn, to lower power density, the power consumption (both dynamic and static) needs to be reduced using low power techniques. Alternatively, the area occupied by the hotspot can be increased to enlarge the cross sectional area of the thermal resistance from the junction to the ambient. This is especially useful when power can not be further reduced due to the realized functionality.

Increasing the area can have negative impact on power and delay due to the increase in wirelength. Cells that communicate with each other are usually laid out in local clusters by placement tools. We show that by maintaining the structure of these clusters the delay and power overhead can be minimized.

The design of Floating-Point Units (FPUs) was investigated as an example for power and thermal management. For floating-point division operations, at the algorithmic level, we used a digit-recurrence approach which consumes much

less power and energy than a multiplicative approach. The peak temperature in the FPU is reduced from two factors. First, the average power consumption in the FPU is less when division is implemented in a digit-recurrence divider. Second, the divider block introduced extra area, effectively reducing the power density of the FPU.

In specific, this work has resulted in the following contributions:

1. We developed a thermal modeling method for solving the equivalent RC circuit to obtain steady-state heat distribution within a circuit. The thermal simulator provided circuit designers a tool to study thermal distribution and optimization. For example, we observed a thermal gradient of 6 °C at a distance of 500 μm, which means that thermal gradients not only exist at chip level, or in large blocks, but also in blocks of a small size. With the shrink of transistor's feature size, thermal gradients within functional blocks can become more significant.

2. We provided a preliminary investigation of the impact of metal wires on the heat distribution in the substrate layers. The experiment results conclude that although metals like copper have a better thermal conductivity than silicon, the thermal resistance in wires is much larger than the substrate due to the much smaller cross sectional area. Therefore, it can be safely assumed that wires do not contribute significantly to heat conduction between hot and cool cells and lateral thermal diffusion mainly occurs in the substrate layer.

3. We proposed a temperature aware interconnect delay estimation method in the early physical design stage and evaluated signal delay in global wires subject to the nonuniform temperature distribution on the substrate layer. From the statistics obtained in the MCNC benchmark, it is shown that the average temperature in global wires can be significantly different and thermal gradients larger than 20 °C were also developed within 40% of all wires. Consequently, during the wire planning stage it is very important to consider the nonuniform thermal distribution on the substrate in the routing algorithm.

4. We proposed and evaluated two post placement stage temperature reduction techniques. Area management with the explicit goal of increasing area in the hotspot is an alternative to reducing power consumption for thermal management. The proposed methods are shown to be more effective and efficient than uniformly increasing a circuit's area.

5. We compared different algorithms and implementations for floating-point division in terms of power consumption and energy per operation. Al-

though division is less frequent than addition and multiplication, its contribution to power and energy consumption for all FP operations is significant due to its much longer latency. Digit-recurrence division units consume much less power than units using multiplicative algorithms, such as Newton-Raphson. For the *SPICE* application, a 30% reduction in energy can be achieved when division is implemented in a radix-16 digit-recurrence divider rather than in a FMA unit.

6. We analyzed the impact on the temperature distribution and leakage dissipation in the FPU when digit-recurrence dividers are used for division operations. The reduction in power not only reduces energy consumption but also results in a lower peak temperature in the FMA. By serving as heat spreader for the FMA, the divider further reduces the power density in the FPU and the peak temperature dropped as much as 7 °C. The power efficient dividers can also be utilized to limit heat diffusion from the FMA to the cache block and to reduce its temperature induced leakage. The experiments showed a 17% reduction in leakage power in the cache block is achieved.

Bibliography

[1] R. Mahajan, C. pin Chiu, and G. Chrysler, "Cooling a microprocessor chip," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1476–1486, Aug. 2006.

[2] N. H. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson Education, Inc, 2005.

[3] H. 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, pp. 468–473, Aug. 1984.

[4] A. Alvandpour, P. Larsson-Edefors, and C. Svensson, "Separation and extraction of short-circuit power consumption in digital CMOS VLSI circuits," *Proc. of the 1998 International Symposium on Low Power Electronics and Design*, pp. 245–249, Aug. 1998.

[5] S.-H. Jung, J.-H. Baek, and S.-Y. Kim, "Short circuit power estimation of static CMOS circuits," *Proc. of the 2001 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 545–549, 2001.

[6] *International Technology Roadmap for Semiconductors (ITRS)*, 2007, http://www.itrs.net/.

[7] S. Mukhopadhyay, A. Raychowdhury, and K. Roy, "Accurate estimation of total leakage in nanometer-scale bulk CMOS circuits based on device geometry and doping profile," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 363–381, Mar. 2005.

[8] F. Fallah and M. Pedram, "Standby and Active Leakage Current Control and Minimization in CMOS VLSI Circuits," *IEICE Transactions on Electronics*, 2005.

[9] *IEEE Standard for Design and Verification of Low Power Integrated Circuits*, IEEE Computer Society Std. 1801, 2009.

[10] "Si2 Common Power Format Specification." [Online]. Available: http://www.si2.org/?page=811

[11] W. Shen, Y. Cai, X. Hong, and J. Hu, "An Effective Gated Clock Tree Design Based on Activity and Register Aware Placement," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 12, pp. 1639–1648, Dec. 2010.

[12] H. Mahmoodi, V. Tirumalashetty, M. Cooke, and K. Roy, "Ultra Low-Power Clocking Scheme Using Energy Recovery and Clock Gating," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 33–44, Jan. 2009.

[13] *Power Compiler User Guide*, Version C-2009.06-SP2 ed., Synopsys, Inc.

[14] D. Chen, J. Cong, Y. Fan, and J. Xu, "Optimality study of resource binding with multi-Vdds," *Proc. of 43rd ACM/IEEE Design Automation Conference*, pp. 580–585, 2006.

[15] C. Xian, Y.-H. Lu, and Z. Li, "Dynamic Voltage Scaling for Multitasking Real-Time Systems With Uncertain Execution Time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1467–1478, Aug. 2008.

[16] Y. Cho and N. Chang, "Energy-Aware Clock-Frequency Assignment in Microprocessors and Memory Devices for Dynamic Voltage Scaling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1030–1040, Jun. 2006.

[17] K. Nowka, G. Carpenter, E. MacDonald, H. Ngo, B. Brock, K. Ishii, T. Nguyen, and J. Burns, "A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1441–1447, Nov. 2002.

[18] V. Sundararajan and K. Parhi, "Low power synthesis of dual threshold voltage CMOS VLSI circuits," *Proc. of the 1999 International Symposium on Low Power Electronics and Design*, pp. 139–144, 1999.

[19] X. He, S. Al-Kadry, and A. Abdollahi, "Adaptive leakage control on body biasing for reducing power consumption in CMOS VLSI circuit," *Proc. of 10th International Symposium on Quality of Electronic Design*, pp. 465–470, Mar. 2009.

[20] S. Kim, C. J. Choi, D.-K. Jeong, S. Kosonocky, and S. B. Park, "Reducing Ground-Bounce Noise and Stabilizing the Data-Retention Voltage of Power-Gating Structures," *IEEE Transactions on Electron Devices*, vol. 55, no. 1, pp. 197–205, Jan. 2008.

[21] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual: For System-on-Chip Design.* Springer Publishing Company, Inc., 2007.

[22] Y. Zhan, S. V. Kumar, and S. S. Sapatnekar, "Thermally Aware Design," *Foundations and Trends in Electronic Design Automation*, no. 3, pp. 255–370, 2008.

[23] Y. Yang, C. Zhu, Z. Gu, L. Shang, and R. Dick, "Adaptive multi-domain thermal modeling and analysis for integrated circuit synthesis and design," *Proc. of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, pp. 575–582, Nov. 2006.

[24] T.-Y. Wang, J.-L. Tsai, and C. Chung-Ping Chen, "Thermal and power integrity based power/ground networks optimization," *Proc. of Design, Automation and Test in Europe*, vol. 2, pp. 830–835, Feb. 2004.

[25] D. Chen, E. Li, E. Rosenbaum, and S.-M. Kang, "Interconnect thermal modeling for accurate simulation of circuit timing and reliability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 2, pp. 197–205, Feb. 2000.

[26] M.-N. Sabry, A. Bontemps, V. Aubert, and R. Vahrmann, "Realistic and efficient simulation of electro-thermal effects in VLSI circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 3, pp. 283–289, Sep. 1997.

[27] S. Wunsche, C. Clauss, P. Schwarz, and F. Winkler, "Electro-thermal circuit simulation using simulator coupling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 3, pp. 277–282, Sep. 1997.

[28] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: a compact thermal modeling methodology for early-stage VLSI design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[29] P. Li, L. Pileggi, M. Asheghi, and R. Chandra, "Efficient full-chip thermal modeling and analysis," *Proc. of the 2004 IEEE/ACM International Conference on Computer Aided Design*, pp. 319–326, Nov. 2004.

[30] B. Wang and P. Mazumder, "A Logarithmic Full-Chip Thermal Analysis Algorithm Based on Multi-Layer Green's Function," *Proc. of Design, Automation and Test in Europe*, vol. 1, pp. 1–6, Mar. 2006.

[31] ——, "Accelerated Chip-Level Thermal Analysis Using Multilayer Green's Function," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 325–344, Feb. 2007.

[32] Y. Zhan and S. Sapatnekar, "High-Efficiency Green Function-Based Thermal Simulation Algorithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1661–1675, Sep. 2007.

[33] N. Rinaldi, "Thermal analysis of solid-state devices and circuits: an analytical approach," *Solid-State Electronics*, vol. 44, no. 10, pp. 1789–1798, 2000.

[34] A. Ajami, K. Banerjee, and M. Pedram, "Modeling and analysis of nonuniform substrate temperature effects on global ULSI interconnects," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 849–861, Jun. 2005.

[35] J. C. Ku and Y. Ismail, "Area Optimization for Leakage Reduction and Thermal Stability in Nanometer-Scale Technologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 2, pp. 241–248, Feb. 2008.

[36] C. J. M. Lasance, "Thermally driven reliability issues in microelectronic systems: status-quo and challenges," *Microelectronics Reliability*, vol. 43, no. 12, pp. 1969–1974, 2003.

[37] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur, "Thermal Performance Challenges from Silicon to Systems," *Intel Technology Journal, (Q3)*, 2000.

[38] F. d'Heurle, "Electromigration and failure in electronics: An introduction," *Proceedings of the IEEE*, vol. 59, no. 10, pp. 1409–1418, Oct. 1971.

[39] J. Black, "Electromigration failure modes in aluminum metallization for semiconductor devices," *Proceedings of the IEEE*, vol. 57, no. 9, pp. 1587–1594, Sep. 1969.

[40] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, vol. 94, no. 1, pp. 1–18, Jul. 2003.

[41] D. Atienza, G. De Micheli, L. Benini, J. Ayala, P. Del Valle, M. DeBole, and V. Narayanan, "Reliability-aware design for nanometer-scale devices," *Proc. of the 2008 Asia and South Pacific Design Automation Conference*, pp. 549–554, Mar. 2008.

[42] J.-L. Tsai, C.-P. Chen, G. Chen, B. Goplen, H. Qian, Y. Zhan, S.-M. Kang, M. Wong, and S. Sapatnekar, "Temperature-Aware Placement for SOCs," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1502–1518, Aug. 2006.

[43] F. Zanini, D. Atienza, C. Jones, and G. De Micheli, "Temperature sensor placement in thermal management systems for mpsocs," *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 1065–1068, May 2010.

[44] E. Kursun and C.-Y. Cher, "Temperature variation characterization and thermal management of multicore architectures," *IEEE Micro*, vol. 29, no. 1, pp. 116–126, Jan.-Feb. 2009.

[45] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic Thermal Management through Task Scheduling," *IEEE International Symposium on Performance Analysis of Systems and software*, pp. 191–201, Apr. 2008.

[46] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," *Proc. of 7th International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 171–182, Apr. 2001.

[47] B. Schafer and T. Kim, "Hotspots Elimination and Temperature Flattening in VLSI Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 11, pp. 1475–1487, Nov. 2008.

[48] A. Gupta, N. Dutt, F. Kurdahi, K. Khouri, and M. Abadir, "LEAF: A System Level Leakage-Aware Floorplanner for SoCs," *Proc. of the 2007 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 274–279, Jan. 2007.

[49] C. Liu, J. Su, and Y. Shi, "Temperature-aware clock tree synthesis considering spatiotemporal hot spot correlations," *Proc. of 26th IEEE International Conference on Computer Design*, pp. 107–113, Oct. 2008.

[50] M. Cho, S. Ahmedtt, and D. Pan, "TACO: temperature aware clock-tree optimization," *Proc. of the 2005 IEEE/ACM International Conference on Computer-Aided Design*, pp. 582–587, Nov. 2005.

[51] A. Chakraborty, K. Duraisami, A. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino, "Dynamic Thermal Clock Skew Compensation Using Tunable Delay Buffers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 639–649, Jun. 2008.

[52] A. Gupta, N. Dutt, F. Kurdahi, K. Khouri, and M. Abadir, "Thermal Aware Global Routing of VLSI Chips for Enhanced Reliability," *Proc. of 9th International Symposium on Quality Electronic Design*, pp. 470–475, Mar. 2008.

[53] K. Lu and D. Pan, "Reliability-aware global routing under thermal considerations," *Proc. of 1st Asia Symposium on Quality Electronic Design*, pp. 313–318, Jul. 2009.

[54] A. Coskun, T. Rosing, and K. Whisnant, "Temperature Aware Task Scheduling in MPSoCs," *Proc. of Design, Automation and Test in Europe*, pp. 1–6, Apr. 2007.

[55] A. Coskun, T. Rosing, K. Whisnant, and K. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor socs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, pp. 1127–1140, Sep. 2008.

[56] A. Coskun, T. Rosing, and K. Gross, "Utilizing predictors for efficient thermal management in multiprocessor socs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1503–1516, Oct. 2009.

[57] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli, "Thermal balancing policy for multiprocessor stream computing platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 12, pp. 1870–1882, Dec. 2009.

[58] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," *Proc. of 30th Annual International Symposium on Computer Architecture*, pp. 2–13, Jun. 2003.

[59] *IEEE Standard for Floating-Point Arithmetic*, IEEE Computer Society Std. 754, 2008.

[60] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[61] S. Oberman, G. Favor, and F. Weber, "AMD 3DNow! technology: architecture and implementations," *IEEE Micro*, vol. 19, no. 2, pp. 37–48, Mar./Apr. 1999.

[62] T. Lang and J. Bruguera, "Floating-point multiply-add-fused with reduced latency," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 988–1003, Aug. 2004.

[63] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit Recurrence Algorithms and Implementations*. Kluwer Academic Publisher, 1994.

[64] H. Baliga, N. Cooray, E. Gamsaragan, P. Smith, K. Yoon, J. Abel, and A. Valles, "Improvements in the Intel Core2 Penryn Processor Family Architecture and Microarchitecture," *Intel Technology Journal*, pp. 179–192, Oct. 2008.

[65] N. Burgess and C. Hinds, "Design issues in radix-4 SRT square root amp; divide unit," *Conference Record of 35th Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1646–1650, 2001.

[66] G. Gerwig, H. Wetter, E. Schwarz, and J. Haess, "High performance floating-point unit with 116 bit wide divider," *Proc. of 16th IEEE Symposium on Computer Arithmetic*, pp. 87–94, Jun. 2003.

[67] A. Nannarelli and T. Lang, "Low-power division: comparison among implementations of radix 4, 8 and 16," *Proc. of 14th IEEE Symposium on Computer Arithmetic*, pp. 60–67, 1999.

[68] S. Oberman, "Floating point division and square root algorithms and implementation in the AMD-K7TM microprocessor," *Proc. of 14th IEEE Symposium on Computer Arithmetic*, pp. 106–115, 1999.

[69] NVIDIA. "Fermi. NVIDIA's Next Generation CUDA Compute Architecture". Whitepaper. [Online]. Available: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[70] D. DasSarma and D. Matula, "Measuring the accuracy of ROM reciprocal tables," *IEEE Transactions on Computers*, vol. 43, no. 8, pp. 932–940, Aug. 1994.

[71] D. De Caro, N. Petra, and A. Strollo, "A high performance floating-point special function unit using constrained piecewise quadratic approximation," *Proc. of the 2008 IEEE International Symposium on Circuits and Systems*, pp. 472–475, May 2008.

[72] S. Oberman and M. Siu, "A high-performance area-efficient multifunction interpolator," *Proc. of 17th IEEE Symposium on Computer Arithmetic*, pp. 272–279, Jun. 2005.

[73] W. Liu and A. Nannarelli, "Power dissipation in division," *Proc. of 42nd Asilomar Conference on Signals, Systems and Computers*, pp. 1790–1794, Oct. 2008.

[74] ——, "Appendix to power dissipation in division," IMM, Technical Report 2008-15. [Online]. Available: http://orbit.dtu.dk/All.external?recid=228622

[75] J. H. L. IV and J. H. L. V, *A Heat Transfer Textbook*. Phlogiston Press, 2008.

[76] R. L. Boylestad, *Introductory Circuit Analysis*. Pearson Education, Inc, 2002.

[77] T. Sato, J. Ichimiya, N. Ono, K. Hachiya, and M. Hashimoto, "On-chip thermal gradient analysis and temperature flattening for SoC design," *Proc. of the 2005 Asia and South Pacific Design Automation Conference (ASP-DAC)*, vol. 2, pp. 1074–1077, Jan. 2005.

[78] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, Jan. 1948.

[79] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Predictable routing," *Proc. of the 2000 IEEE/ACM International Conference on Computer Aided Design*, pp. 110–113, 2000.

[80] K. Sankaranarayanan, S. Velusamy, M. Stan, C. L, and K. Skadron, "A case for thermal-aware floorplanning at the microarchitectural level," *Journal of Instruction Level Parallelism*, vol. 7, 2005.

[81] S. Adya and I. Markov, "Fixed-outline floorplanning: enabling hierarchical design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[82] J. Bruguera and T. Lang, "Floating-point fused multiply-add: reduced latency for floating-point addition," *Proc. of 17th IEEE Symposium on Computer Arithmetic*, pp. 42–51, Jun. 2005.

[83] S. Oberman and M. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions on Computers*, vol. 46, no. 2, pp. 154–161, Feb. 1997.

[84] J. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitive Approach*. Morgan Kaufmann Publishers, 1995.

[85] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli, and S. Vora, "A 45 nm 8-Core Enterprise Xeon Processor," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 7–14, Jan. 2010.

[86] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Drowsy instruction caches. Leakage power reduction using dynamic voltage scaling and cache sub-bank prediction," *Proc. of 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35)*, pp. 219–230, 2002.

Appendix A

Source Code for the Thermal Simulation Tool

In this chapter, we list the source code for our SPICE simulation based thermal simulation tool. Each thermal cell in the RC equivalent netlist is an instance of a SPICE sub-circuit, which models the cuboid in Figure 4.2. The thermal cell model and the thermal coefficients for different layers are read into SPICE using the `include` statement in the netlist preamble. Both files are listed in Section A.1.

The mapping between standard cells in the layout domain and thermal cells in the thermal domain is performed by `mapv3.pl`, which reads the physical location and power consumption of each standard cell and maps them to a two dimensional array of thermal cells. The output is a list of power values in each thermal cell, which is read by `makespicev3.pl` to generate the SPICE netlist. These two scripts are listed in Section A.2 and Section A.3.

Some auxiliary scripts help with parsing the output of SPICE simulation and generating a colored thermal map. These scripts are listed in Section A.4.

A.1 SPICE Subcircuit Model for Thermal Cells

SPICE sub-circuit model for the thermal cells.

```
.SUBCKT cell N E S W T B
.param res_n='L/(2*(kn*(W*H)))'
.param res_s='L/(2*(ks*(W*H)))'
.param res_e='W/(2*(ke*(L*H)))'
.param res_w='W/(2*(kw*(L*H)))'
.param res_u='H/(2*(ku*(W*L)))'
.param res_d='H/(2*(kd*(W*L)))'
*.param cap='ro*cp*(W*L*H)'
R_N central N res_n
R_S central S res_s
R_W central W res_w
R_E central E res_e
R_U central T res_u
R_D central B res_d
*Ccentral central 0 cap
Ipower 0 central DC=Ipw
.ENDS cell
```

Global parameter file contains thermal coefficients for different layers.

```
* thermal conductivity silicon [k]=W/(C*m)
.param k_package=5
.param k_substrate=63
.param k_devicex=28
.param k_devicey=14
.param k_devicez=69
.param k_wirexy=20.7
.param k_wirez=4.26
.param k_bumpxy=0.05
.param k_bumpz=0.25
.param k_side=0
* density [ro]=kg/m^3
.param ro=2330
* specific heat [cp]=J/(Kg*C)
.param cp=712.8
* cell dimensions
.param L=20.0u
.param W=20.0u
* ambient temperature
.param Vam=0
```

A.2 Mapping from Standard Cells to Thermal Cells

```
#!/usr/bin/perl
use Data::Dumper;
use Storable;
use Getopt::Long;

########################################
#From cell placement and cell power files
#To thermal grid power
########################################

my $counter=0;
my $scale_factor=1;
my ($cell_name,$cell_power,$cell_area,$cell_width,
    $cell_height,$cell_x,$cell_y);
my $cell_height=2.6;  #2.6 um
my %allcells;
my ($maxwidth,$maxheight);
$maxwidth=0;
$maxheight=0;
my ($minX,$minY);
$minX=100;
$minY=100;
my ($origin_offset_X,$origin_offset_Y);
$origin_offset_X=0;
$origin_offset_Y=0;
my $totalpower;
my @grid;
my $xgrid_dimension=20;
my $ygrid_dimension=20;
my $xgridstep,$ygridstep;
my $verbose;
my $power_file, $physical_file;
my $print_power;

GetOptions("xmax=i" => \$xgrid_dimension,
       "ymax=i" => \$ygrid_dimension,
       "verbose" => \$verbose,
       "scale=f" => \$scale_factor,
       "height=f" => \$cell_height,
       "power=s" => \$power_file,
```

```
        "physical=s" => \$physical_file,
        "print_power" => \$print_power);

die("Please specify physical and power file names.\n")
    unless(defined($power_file) and defined($physical_file
    ));
die("Cannot open physical information file!") if(!open
    PHYSICAL, $physical_file);
die("Cannot open power information file!") if(!open POWER
    , $power_file);


while(<PHYSICAL>){
  if(/^(----)+$/){
    last;
  }
}
while(<PHYSICAL>){
  unless(/^(----)+$/){
    if(/(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s
       +\((-?[0-9]+\.[0-9]*), (-?[0-9]+\.[0-9]*)\)/){
      $cell_name=$1;
      $cell_area=$4;
      $cell_x=$6;
      $cell_y=$7;
      $cell_width=$cell_area/$cell_height;
      $maxwidth=$6+$cell_width if($6+$cell_width>=
          $maxwidth);
      $maxheight=$7+$cell_height if($7+$cell_height>=
          $maxheight);
      $minX=$cell_x if($cell_x<=$minX);
      $minY=$cell_y if($cell_y<=$minY);

      die("Cell $cell_name already exists!") if(exists
          $allcells{$cell_name});
      $allcells{$cell_name}->{'x'}=$cell_x;
      $allcells{$cell_name}->{'y'}=$cell_y;
      $allcells{$cell_name}->{'width'}=$cell_width;
      $allcells{$cell_name}->{'height'}=$cell_height;

      $counter++;
    }
  }else{
    last;
```

```
  }
}

print "Total: $counter\nWidth: $maxwidth\tHeight:
    $maxheight\nmin X: $minX\nmin Y: $minY\n" if defined
    $verbose;

$origin_offset_X=$minX;
$maxwidth-=$origin_offset_X;

foreach $cn (sort keys %allcells){
  $allcells{$cn}->{'x'}-=$origin_offset_X;
}

$origin_offset_Y=$minY;
$maxheight-=$origin_offset_Y;

foreach $cn (sort keys %allcells){
  $allcells{$cn}->{'y'}-=$origin_offset_Y;
}

print "Total: $counter\nWidth: $maxwidth\tHeight:
    $maxheight\n" if defined $verbose;

$counter=0;

while(<POWER>){
  if(/^(----)+$/){
    last;
  }
}
while(<POWER>){
  if(/Totals/){
    last;
  }elsif(/(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+\(\S+\)\s+(\S+)
      /){
    $cell_name=$1;
    $dyn_power = $4;
    $leak_power = $6;
    $cell_power=($dyn_power*$scale_factor+$leak_power)*1e
        -3;  # scaling and mW -> W

    if(exists $allcells{$cell_name}){
      $allcells{$cell_name}->{'power'}=$cell_power;
```

```
        $totalpower+=$cell_power;
        $counter++;
      }else{
        die("Cell $cell_name can not be recognized!\n");
      }
    }
}

my $pd=$totalpower/($maxwidth*$maxheight*1e-6);
print "Total: $counter\nPower: $totalpower\nPower Density
    : $pd\n" if defined $verbose;

$xgridstep=$maxwidth/$xgrid_dimension;
$ygridstep=$maxheight/$ygrid_dimension;

#initialize grid cells
for(my $i=0;$i<$xgrid_dimension;$i++){
  for(my $j=0;$j<$ygrid_dimension;$j++){
    $grid[$i][$j]->{'name'}="Unit${i}_${j}";
    $grid[$i][$j]->{'x'}=$i*$xgridstep;
    $grid[$i][$j]->{'y'}=$j*$ygridstep;
    $grid[$i][$j]->{'power'}=0;
  }
}

#output grid cell dimensions
print "xgridstep: $xgridstep\tygridstep: $ygridstep\n" if
     defined $verbose;

#put cells into grid
foreach $cn (sort keys %allcells){
  my $x=$allcells{$cn}->{'x'};
  my $y=$allcells{$cn}->{'y'};
  my $p=$allcells{$cn}->{'power'};
  my $xindex=sprintf("%d",$x/$xgridstep);
  my $yindex=sprintf("%d",$y/$ygridstep);
  print "$x\t$y\t$p\n" if defined $print_cell_power;
  $grid[$xindex][$yindex]->{'power'}+=$p;
}

my @biglist;
for(my $j=0;$j<$ygrid_dimension;$j++){
  for(my $i=0;$i<$xgrid_dimension;$i++){
    push(@biglist,$grid[$i][$j]->{'power'});
```

```
    print $grid[$i][$j]->{'power'}." \n";
  }
}

#print Dumper(\%allcells);
#print Dumper(\@grid);
```

A.3 Generating SPICE Netlist for the RC Equivalent Circuit

```
#!/usr/bin/perl
use Getopt::Long;
use Storable;

my $tincr1="1u";
my $tstop1="1u";
my $xnum_of_cells=20;
my $ynum_of_cells=20;
my $num_of_layers=9;
my $device_layer=5;
my $prefix="Xcell_";
my $power_file;
my @layers;
my %conductivity=(
        "package" => {
                        "x" => "k_package",
                        "t" => "200u",
                    },
        "substrate" => {
                        "x" => "k_substrate",
                        "t" => "12.5u",
                    },
        "device" => {
                        "x" => "k_devicex",
                        "y" => "k_devicey",
                        "z" => "k_devicez",
                        "t" => "2u",
                    },
        "wire" => {
                        "x" => "k_wirexy",
                        "z" => "k_wirez",
                        "t" => "3.1u",
```

```
                        },
        "bump" => {
                            "x" => "k_bumpxy",
                            "z" => "k_bumpz",
                            "t" => "200u",
                        },
);

my %layermap = (
        "layer1" => "package",
        "layer2" => "substrate",
        "layer3" => "substrate",
        "layer4" => "substrate",
        "layer5" => "substrate",
        "layer6" => "device",
        "layer7" => "wire",
        "layer8" => "wire",
        "layer9" => "bump",
);

GetOptions("xdimension=i" => \$xnum_of_cells,
       "ydimension=i" => \$ynum_of_cells,
       "layers=i" => \$num_of_layers,
       "tincr=s" => \$tincr1,
       "tstop=s" => \$tstop1,
       "power=s" => \$power_file);

create_cells();
read_power($power_file);

print "*Cell Based Thermal Model\n";
print ".include thermal.spi\n";
print ".include global_param\n";

print "\n********RC MODEL********\n";
print_cells($num_of_cells);
print "Vambient 1 0 DC Vam\n";
print "\n********SIMULATION********\n";
#print ".OP VOLTAGE\n";
print ".TRAN $tincr1 $tstop1\n";
print "\n********MEASURE********\n";
print ".OPTION POST PROBE\n";
print_measures();
print_probes();
```

```
print "\n.END\n";

sub create_cells{
  my $xlength=$xnum_of_cells;
  my $ylength=$ynum_of_cells;

  for(my $i=1;$i<=$xlength;$i++){
    for(my $j=1;$j<=$ylength;$j++){
      for(my $k=1;$k<=$num_of_layers;$k++){
        my $cell_name="XLayer${k}_${i}_${j}";
        $layers[$k][$i][$j]->{'name'}=$cell_name;
        $layers[$k][$i][$j]->{'power'}=0;
        $layers[$k][$i][$j]->{'initTemp'}=0;
      }
    }
  }
}

sub read_power{
  my ($fn)=@_;
  my $xlength=$xnum_of_cells;
  my $ylength=$ynum_of_cells;
  my $counter=0;

  die("Cannot open input power file!\n") if(!open POWER,
      $fn);
  while(<POWER>){
    chomp;
    my $i=1+$counter%$xlength;
    my $j=1+$counter/$xlength;
    # device layer is layer 6
    $layers[$device_layer][$i][$j]->{'power'}=$_;
    $layers[$device_layer][$i][$j]->{'initTemp'}=0;
    $counter++;
  }
  die("Number of cells $xnum_of_cells X $ynum_of_cells
      and number of powers $counter do not match!\n")
      unless($counter==$xnum_of_cells*$ynum_of_cells);
}

sub print_cells{
  my $ambient=1;
  my $xlength=$xnum_of_cells;
```

```
my $ylength=$ynum_of_cells;

for(my $k=1;$k<=$num_of_layers;$k++){
  my $line;
  my $material=$layermap{"layer${k}"};
  my $thickness=$conductivity{$material}->{"t"};

  for(my $i=1;$i<=$xlength;$i++){
    for(my $j=1;$j<=$ylength;$j++){
      my $north,$east,$south,$west;
      my $curval,$kn,$ke,$ks,$kw,$ku,$kd;
      my $top,$bottom;
      my $btmlayer=$k-1;

      $top="tbLayer${k}_${i}_$j";
      $bottom="tbLayer${btmlayer}_${i}_$j";

      if(defined $conductivity{$material}->{"z"}){
        $ku=$conductivity{$material}->{"z"};
        $kd=$conductivity{$material}->{"z"};
      }else{
        $ku=$conductivity{$material}->{"x"};
        $kd=$conductivity{$material}->{"x"};
      }

      if($k==1){
        $bottom=$ambient;
      }

      if($k==$num_of_layers){
        $top=$ambient;
      }

      $curval=$layers[$k][$i][$j]->{'power'};
      $tempval=$layers[$k][$i][$j]->{'initTemp'};

      #determine connections
      if($j>1 and $j<$ylength){
        my $w=$j-1;
        $west="ewLayer${k}_${i}_$w";
        $east="ewLayer${k}_${i}_$j";
        $kw=$conductivity{$material}->{"x"};
        $ke=$conductivity{$material}->{"x"};
      }elsif($j==1){
```

```
    $west=$ambient;
    $east="ewLayer${k}_${i}_$j";
    $ke=$conductivity{$material}->{"x"};
    $kw=$conductivity{$material}->{"x"};
    $kw="k_side" if($k!=1);      # package layer is
        treated differently
  }elsif($j==$ylength){
    my $w=$j-1;
    $west="ewLayer${k}_${i}_$w";
    $east=$ambient;
    $kw=$conductivity{$material}->{"x"};
    $ke=$conductivity{$material}->{"x"};
    $ke="k_side" if($k!=1);
  }

  if($i>1 and $i<$xlength){
    my $n=$i-1;
    $north="nsLayer${k}_${n}_$j";
    $south="nsLayer${k}_${i}_$j";
    if(defined $conductivity{$material}->{"y"}){
      $kn=$conductivity{$material}->{"y"};
      $ks=$conductivity{$material}->{"y"};
    }else{
      $kn=$conductivity{$material}->{"x"};
      $ks=$conductivity{$material}->{"x"};
    }
  }elsif($i==1){
    $north=$ambient;
    $south="nslayer${k}_${i}_$j";
    if(defined $conductivity{$material}->{"y"}){
      $kn=$conductivity{$material}->{"y"};
      $ks=$conductivity{$material}->{"y"};
    }else{
      $kn=$conductivity{$material}->{"x"};
      $ks=$conductivity{$material}->{"x"};
    }
    $kn="k_side" if($k!=1);
  }elsif($i==$xlength){
    my $n=$i-1;
    $north="nsLayer${k}_${n}_$j";
    $south=$ambient;
    if(defined $conductivity{$material}->{"y"}){
      $kn=$conductivity{$material}->{"y"};
      $ks=$conductivity{$material}->{"y"};
```

```
            }else{
              $kn=$conductivity{$material}->{"x"};
              $ks=$conductivity{$material}->{"x"};
            }
              $ks="k_side" if($k!=1);
          }
          $line.="$layers[$k][$i][$j]->{'name'} $north
              $east $south $west $top $bottom cell kn=$kn ke
              =$ke ks=$ks kw=$kw ku=$ku kd=$kd H=$thickness
              InitTemp=$tempval Ipw=$curval\n";
        }
      }
      print "$line\n";
    }
}

sub print_measures{
    print "\n";
    my $xlength=$xnum_of_cells;
    my $ylength=$ynum_of_cells;
    my $counter=1;

    for(my $j=1;$j<=$ylength;$j++){
      for(my $i=1;$i<=$xlength;$i++){
        print ".MEASURE V central$counter MAX V($layers[
            $device_layer][$i][$j]->{'name'}.central)\n";
        $counter++;
      }
    }
}

sub print_probes{
    print "\n";
    my $xlength=$xnum_of_cells;
    my $ylength=$ynum_of_cells;

    for(my $i=1;$i<=$xlength;$i++){
      for(my $j=1;$j<=$ylength;$j++){
        print ".PROBE V($layers[$device_layer][$i][$j]->{'
            name'}.central)\n";
      }
    }
}
```

A.4 Auxiliary Scripts to Generate a Thermal Map from SPICE Simulation Results

`readtemp.pl` reads the output of SPICE simulation and extracts the temperature values.

```
#!/usr/bin/perl
use Data::Dumper;
use Getopt::Long;

my $startflag=0;
my $flag=0;
my @temperatures;
my $sort;

GetOptions("sort" => \$sort);

while(<>){
  my $line;

  chomp;
  $line=$_;
  if($startflag==1){
    $flag=1;
  }
  if(/temper\s+alter/){
    $startflag=1;
  }
  if(/^\s+25.0000\s+1.0000\s+$/){
    $flag=0;
  }
  if($flag==1){
    my @values=split /\s+/,$line;
    push @temperatures,@values[1..4];
  }
}

my @sorted=sort {$b <=> $a} @temperatures;
if(defined $sort){
  print "@sorted\n";
}else{
  print "@temperatures\n";
}
```

```
#print Dumper(@temperatures);
```

`map3d.pl` takes the temperatures as input and outputs in tuple (x, y, value).

```
#!/usr/bin/perl
use Getopt::Long;

my $xmax=40, $ymax=40;
my $h;
my $counter=0;

GetOptions("xmax=i" => \$xmax,
        "ymax=i" => \$ymax,
        "hotspot" => \$h);
while(<>){
   chomp;
   my $y=sprintf("%d",$counter/$xmax) + 0.5;
   my $x=($counter % $xmax) + 0.5;
   my $val=$_;
   $val=$val-318 if defined $h;
   print "$x\t$y\t$val\n";
   $counter++;
}
```

`thermalplot.sh` is a wrapper for gnuplot scripts which read the data file containing tuples (x, y, value) and plot them in a thermal map.

```
#!/bin/sh

gnuplot -persist << EOF
set pm3d implicit
set size ratio 1.0
set view map
#set cbrange [20:35]
set palette rgbformulae 22,13,-31
set dgrid3d $1,$2
set ticslevel 0
set yrange [0 : $2]
set term postscript eps enhanced color
set output "$4.eps"
splot "$3.dat" w pm3d t ""
EOF
```

Appendix B

Synopsys Commands in the ERI and HSD Methods

B.1 Floorplanning in Synopsys' IC Compiler

In the benchmark circuits for the *Empty Row Insertion* and *HotSpot Diffusion* methods, we place multipliers in specific regions to force the location of hotspots. To achieve this, plan groups are used to restrict the placement of cells to a specific region of the core area. Following is an example, which places module `MULT2` in a rectangular region specified by the lower left and upper right coordinates.

```
create_plan_groups -rectangle {{$x0 $y0} {$x1 $y1}} {MULT2}
```

To remove a defined plan groups, simply use the `remove_plan_groups` command.

B.2 Commands for Information Retrieval and Cell Movement

Various information regarding cells, nets, timing paths can be retrieved with the `get_attribute` command. For example, the following command can be used to get the maximum delay, which returns the delay in nanoseconds in a scalar value that can be directly used by other scripts:

```
get_attribute [get_timing_paths -delay_type max] arrival
```

To get the maximum delay on paths through a certain cell:

```
get_attribute [get_timing_paths -through $cell_name] arrival
```

To get the timing slack of a specified path:

```
set mypath [get_timing_paths -through $cell_name]

get_attribute $mypath slack
```

To get the coordinates of a certain cell, one can use:

```
get_location $cell_name
```

To set the location of a given cell to specified coordinates:

```
set_attribute [get_cells -all $cell_name] origin {$x0 $y0}
```

To create placement bounds during the placement stage:

```
create_bounds -coordinate [list $x0 $y0 $x1 $y1] -exclusive $cells
```

The above command creates a rectangular bound, which only allows specified cells to be placed inside. Once the bound is created, the layout needs to be legalized with the command `legalize_placement`.

IC Compiler does not automatically recompute the changes in wire length and wire capacitance when the layout is changed by the user. Therefore, in order to obtain the updated timing and power reports, first make IC Compiler extract the resistance and capacitance associated with wires:

```
extract_rc
```

B.3 Scripts for the *Empty Row Insertion* Method

The *ERI* method is implemented in perl scripts utilizing the existing `mapv3.pl` listed in Chapter A. As a result, in the following we only list the code snippet that is different from `mapv3.pl`. The script parses the physical file and computes new coordinates for the cells in the hotspot region. Tcl commands to set new locations for these cells are output in a text file, which can be subsequently read into IC Compiler.

```
while(<PHYSICAL>){
  if(/^(----)+$/){
    last;
  }
}
while(<PHYSICAL>){
  unless(/^(----)+$/){
    if(/(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s
        +\((-?[0-9]+\.[0-9]*), (-?[0-9]+\.[0-9]*)\)/){
      my $cell_name=$1;
      my $cell_area=$4;
      my $cell_orient=$5; # 0, 0-mirror, 180, 180-mirror
      my $cell_x=$6;
      my $cell_y=$7;
      my $cell_originx=$cell_x;
      my $cell_originy=$cell_y;
      my $cell_width=$cell_area/$cell_height;

      if($cell_orient eq "0-mirror"){
        $cell_originx=$cell_x+$cell_width;
      }elsif($cell_orient eq "180"){
        $cell_originx=$cell_x+$cell_width;
        $cell_originy=$cell_y+$cell_height;
      }elsif($cell_orient eq "180-mirror"){
        $cell_originy=$cell_y+$cell_height;
      }

      $maxwidth=$6+$cell_width if($6+$cell_width>=
          $maxwidth);
      $maxheight=$7+$cell_height if($7+$cell_height>=
          $maxheight);
      $minX=$cell_x if($cell_x<=$minX);
      $minY=$cell_y if($cell_y<=$minY);
```

```
        die("Cell_$cell_name_already_exists!") if(exists
            $allcells{$cell_name});
        $allcells{$cell_name}->{'x'}=$cell_x;
        $allcells{$cell_name}->{'y'}=$cell_y;
        $allcells{$cell_name}->{'originx'}=$cell_originx;
        $allcells{$cell_name}->{'originy'}=$cell_originy;
        $allcells{$cell_name}->{'orient'}=$cell_orient;
        $allcells{$cell_name}->{'width'}=$cell_width;
        $allcells{$cell_name}->{'height'}=$cell_height;

        $allrows{$cell_y}->{$cell_name}=$allcells{
            $cell_name};

        $counter++;
      }
    }else{
      last;
    }
}

my $row_counter, $y_increment;
foreach my $row (sort { $a <=> $b } keys %allrows){
    if($row>=$starty and $row<=$endy){
      $row_counter++;
      $y_increment=$row_counter*2.6;
    }elsif($row<$starty){
      $y_increment=0;
    }

    foreach my $cell_name (sort { $allrows{$row}->{$b}->{'x
        '} <=> $allrows{$row}->{$a}->{'x'} } keys %{$allrows
        {$row}}){
      if($y_increment!=0){
        my $icc_command;
        my $old_y, $new_y, $old_originy, $new_originy;

        $old_y=$allcells{$cell_name}->{'y'};
        $old_originy=$allcells{$cell_name}->{'originy'};
        $new_y=$old_y+$y_increment;
        $new_originy=$old_originy+$y_increment;

        $allcells{$cell_name}->{'y'}=$new_y;
        $allrows{$row}->{$cell_name}->{'y'}=$new_y;
```

```
        $allcells{$cell_name}->{'originy'}=$new_originy;
        $allrows{$row}->{$cell_name}->{'originy'}=
            $new_originy;

        $icc_command="set_attribute [get_cells -all ${
            cell_name}] origin {$allcells{$cell_name}->{'
            originx'} $allcells{$cell_name}->{'originy'}}";
        print "$icc_command\n" if defined $command;
      }
    }
}

$maxheight+=$y_increment;
print "Total: $counter\nWidth: $maxwidth\tHeight:
    $maxheight\n" if defined $verbose;
```

B.4 Scripts for the *HotSpot Diffusion* Method

```
proc inRegion { cell_location Region } {
  set cellx [lindex $cell_location 0]
  set celly [lindex $cell_location 1]
  set Rx0 [lindex $Region 0]
  set Ry0 [lindex $Region 1]
  set Rx1 [lindex $Region 2]
  set Ry1 [lindex $Region 3]

  if {$cellx >= $Rx0 && $celly >= $Ry0 && $cellx < $Rx1
     && $celly < $Ry1} {
    return 1
  } else {
    return 0
  }
}

foreach_in_collection mycell [get_cells] {
  set cell_name [get_attribute $mycell full_name]
  set cell_location [get_location $mycell]
  set cellx [lindex $cell_location 0]
  set celly [lindex $cell_location 1]

  if {[inRegion $cell_location $HS]} {
```

```
    if {[info exists allrows($celly)]} {
      set l [llength $allrows($celly)]
      set i 0
      for {} {$i<$l} {incr i} {
        if {$cellx < $xarray([lindex $allrows($celly) $i
            ])} {
          break;
        }
      }
      set allrows($celly) [linsert $allrows($celly) $i
          $cell_name]
    } else {
      set allrows($celly) $cell_name
    }
    set xarray($cell_name) [lindex $cell_location 0]
  }
}

set rowheight 2.6
set max 14
set rows [lsort -real -decreasing [array names allrows]]
set extra 0
set i 0
for {} {$i<$max} {incr i} {
  set y1 [lindex $rows $i]
  set y1_1 [expr {$y1 + $rowheight*($max-$i+$extra)}]
  foreach mycell $allrows($y1) {
    set_cell_location $mycell -coordinates [list $xarray(
        $mycell) $y1_1]
  }
}
set rows [lsort -real -increasing [array names allrows]]
set i 0
for {} {$i<$max} {incr i} {
  set y1 [lindex $rows $i]
  set y1_1 [expr {$y1 - $rowheight*($max-$i)}]
  foreach mycell $allrows($y1) {
    set_cell_location $mycell -coordinates [list $xarray(
        $mycell) $y1_1]
  }
}

remove_bounds -all
legalize_placement
```