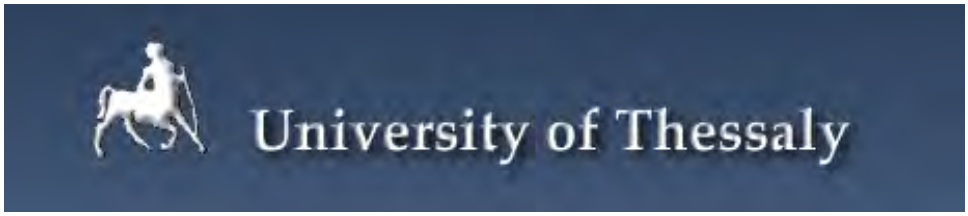


Master Thesis

Thermal Analysis of 3D Integrated Circuits

Educational Institution: Department of Computer And Communication Engineering, University of Thessaly

Supervisor: George Stamoulis, Nestoras Eumorfopoulos

Student: Alexia Marnari

Acknowledgement

First of all, I would like to express my greatest thanks to my professors and supervisors of this thesis, Prof. George Stamoulis and Prof. Nestoras Eumorfopoulos, who were always available to offer their help and advice, who encouraged me and induced me to try more.

I would also like to express my sincere gratitude to Konstantis Daloukas, P.H.D. student at the Department of Computer and Communications Engineering at the University of Thessaly, for his valuable guidance, help and suggestions.

Finally, I should thank my family and friends for their support, understanding and patience. Certainly, I should refer my friend and postgraduate student at the same Department Athanasio Bareka, for his insightful suggestions on critical moments.

Περίληψη

Η θερμική ανάλυση είναι μία από τις πιο κρίσιμες προκλήσεις που συνεπάγεται η τεχνολογική εξέλιξη. Η συνεχής προσπάθεια για μικρότερα μεγέθη και μεγαλύτερη απόδοση, καθώς και η νέα τρισδιάστατη δομή των ολοκληρωμένων κυκλωμάτων, μαζί με τη χρήση των low-k διηλεκτρικών, προκαλούν μη αμελητέα αύξηση της θερμοκρασίας απειλώντας την σωστή λειτουργία των ολοκληρωμένων κυκλωμάτων. Η απόδοση, αξιοπιστία και η κατανάλωση ισχύος των συσκευών τίθενται σε κίνδυνο, ενώ το φαινόμενο Joule heating, επιδεινώνεται. Αναμφίβολα επομένως, τα θερμικά ζητήματα απαιτούν την προσοχή μας και θα πρέπει να λαμβάνονται σοβαρά υπόψιν.

Σκοπός αυτής της εργασίας είναι να προσδιοριστεί ο ρόλος των διασυνδέσεων στην αύξηση της θερμοκρασίας. Εκμεταλλευόμενοι την πληροφορία από τα αρχεία .def/.lef, μπορούμε να βρούμε την ακριβή θέση των μετάλων στην περιοχή πάνω από το υπόστρωμα. Μοντελοποιώντας την περιοχή αυτή ως ένα πλέγμα από μπλοκ διαστάσεων 1x1 micron, σχηματίζεται ένα δίκτυο αντιστάσεων. Με τη χρήση της μεθόδου Finite Differences, παράγεται ένα σύστημα γραμμικών εξισώσεων το οποίο μπορεί να λύθει εύκολα μέσω ενός παράλληλου “precondition” μηχανισμού και να προκύψουν τα επιθυμητά αποτελέσματα.

Abstract

Temperature consideration is one of the most critical challenges that come with technological evolution. The continuous effort for smaller sizes and greater performance as well as the new 3D structure of integrated circuits together with the use of low-k dielectrics, cause non-negligible temperature rises menacing the proper functionality of ICs. The performance, reliability and power consumption of the devices are set under risk, while Joule heating effect worsens. Undoubtedly then, thermal issues demand our concern and have to seriously been taken into account.

The purpose of this thesis is to define the role of interconnections on temperature increase. Exploiting the information from .def/.lef files, we can find the exact position of metals on the upper substrate area. Modeling it as a grid of 1x1 micron blocks, a network of resistances is formed. The use of Finite Differences Method produces a linear system of equations that can be fastly solved with a parallel precondition mechanism, presenting us the desired results.

Contents

1 Chapter - Thermal Effects

1.1 Heat Generation due to technological evolution

Since the appearance of the first technological achievements, speed and performance have been the main targets inspiring inventors to evolve their previous attainments. More specifically, in semiconductor industry, there is a relentless effort for higher CMOS performance and functionality, greatly pushed by the customer needs and the competition between manufacturers. Undoubtedly, the electronics industry plays a leading role in economic, social and political development throughout the world. Therefore, the attempt for rises in integrated circuits integration density and speed will continue to keep a dominant position in experts minds [12].

Direct consequence of the above presented attempt is the scaling of CMOS, thus a great miniaturization of device sizes has been observed throughout the years. However, even if the catchword *Smaller and Faster* constitutes a formidable driving force for such an aggressive technology scaling, we have to be careful with the challenges and risks that it brings into the surface [21].

A significant issue accompanying the deeper entrance in nanometer sizes is the increased power density. The latter leads in elevated on chip temperature which puts the desired performance under risk, menacing the proper functionality of the devices. In addition to this, as chips warm up in a non-uniform way, local hot spots and spatial gradients are generated, with higher power densities and consequently higher local temperatures [24].

The thermal menace also worsens due to the nowadays used multilayer 3D stacking, as well as the use of low-k dielectrics (Fig. 1).Stacking multiple layers in a 3D volume promises density and performance. However, it requires extensive thermal consideration as the power density and temperature of these architectures can be quite high [26] [21]. In this case we need cooling a volume consisting of units placed on top of each other, instead of just cooling a planar surface. In addition to this, deep submicron technologies such as low-k dielectric materials have been introduced in order to achieve better performance, reducing interconnect capacitance and thus delay as well as cross talk noise. However, due to their lower thermal conductivity they are more susceptible to thermal effects [5].

Temperature increase is an inevitable aspect of the continuous scaling

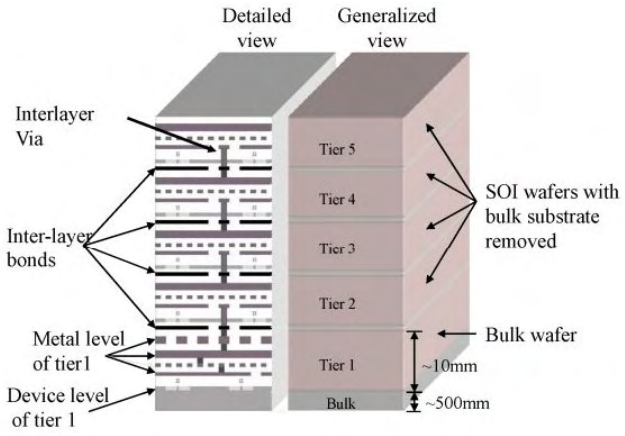


Figure 1: 3D Integrated Circuit structure.

trend. Nevertheless, high temperature has significant impacts on chip performance and reliability. Leading to slower transistor speed, more leakage power consumption, higher interconnect resistance and reduced reliability, thermal issues constitute a major challenge that has to be seriously taken into consideration. Management of them remains key factor for future microprocessors and ICs. Great proof of their significance is the extensive existing research on thermal analysis models [26] [8] [17] [25] [16] [22] [9] [28] [18], while many research laboratories keep working on this subject trying to present more accurate thermal simulators for VLSI designs.

1.2 Thermal Issues

As mentioned above, thermal issues affect integrated circuits'performance, power and reliability in a critical way. It is therefore essential to take a closer look at the way they are influenced.

1.2.1 Performance

The required time to accomplish one task using a specified amount of an available resource is defined as the performance of a computer. Concerning a chip, it's performance is directly associated with it's clock frequency and consequently the circuit delay [7].

Temperature rises may bring into the surface two conflicted challenges. On one hand, higher temperature leads to a reduction in charge-carrier mobility which means lower drive current of a transistor and finally increased circuit delays. On the other hand, higher temperature leads to a reduction in the transistor threshold voltage which means increased drive current and finally reduced circuit delays.

According to the prevalent effect there will be negative temperature dependence (increased delays), positive temperature dependence (decreased delays), or mixed temperature dependence [31].

1.2.2 Power Consumption

The sources of power consumption in VLSI circuits come from devices and interconnects. The power consumption from devices can be decomposed into dynamic power, short circuit power and static power. The power consumption in interconnects comes from Joule effect.

Dynamic power is caused by charging and discharging events during voltage transitions and has a negligible dependence on temperature. Short-circuit power is caused by current from supply to ground during switching and also has insignificant dependence on temperature.

Static power however, appears due to leakage and has exponential dependence on temperature. Thus, temperature variations are significally depicted on leakage which in many cases owns a leading position on the total on chip power [7] [31] [30] [20].

1.2.3 Joule heating

Power dissipation results not only from dynamic, short circuit and static power but from Joule heating as well. Joule heating is the process by which the passage of an electric current through a conductor releases heat [4]. Therefore, the flow of current through metal wires dissipates power and generates heat increasing the wire temperature [27]. Temperature rise in the interconnects due to Joule heating effect can be important because of the new scaling trends. According to the stacked 3D model, interconnects are located away from the silicon substrate and the heat sink, separated by layers of insulating materials with lower thermal conductivities than that of silicon [6]. Low-k dielectrics have reduced the material conductivities aggravating Joule effect. As the wire resistivity is temperature dependent:

$\rho(T) = \rho_0[1 + b(T - T_0)]$, higher densities lead to higher temperatures and larger interconnect delays [27] [25].

Consequently, the perpetual scaling increases current density and Joule heating effect becomes significant factor to the total device temperature rise, menacing device's reliability and performance.

1.2.4 Reliability

The ability of a system to maintain its proper functionality under even unexpected circumstances during its lifetime is called reliability. Concerning semiconductors, the term reliability regards the premature aging of semiconductor materials and the consequences it entails to the operation of the devices. Some of the most important temperature related effects affecting reliability are: Bias temperature Instability and Electromigration [14].

Bias Temperature Instability

It is a complex electro thermal phenomenon which occurs at high temperatures and causes threshold voltage shifts over long periods of time. It is distincted in NBTI concerning pMOS and PBTI concerning nMOS. NBTI (Negative Bias Temperature Instability) is caused by the generation of interface traps which are unsaturated silicon dangling bonds. Temperature rises accelerate the generation of these traps while the resulted trapped holes increase threshold voltage and decrease drain current.

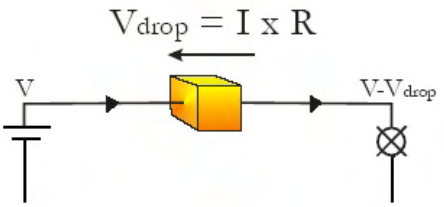
A dual effetct is PBTI (Positive Bias Temperature Instability) with lower impact than NBTI but quite important [14] [31].

ElectroMigration

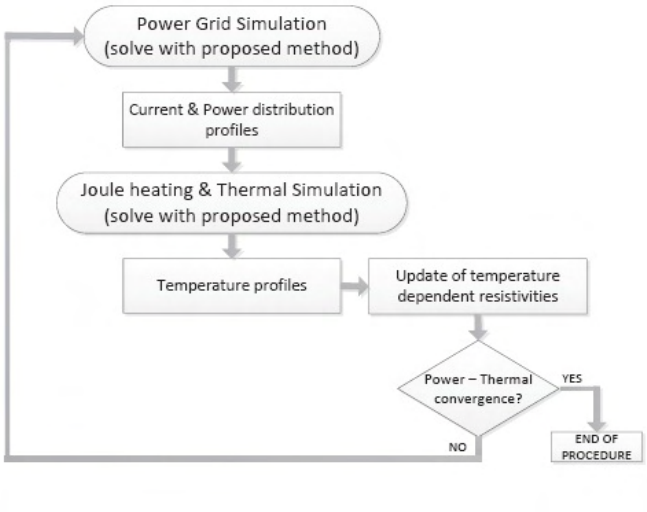
Electromigration refers to the migration of metal atoms over time to regions where the current density is high , forming voids in the metal wire. Black's equation describes electromigration in the following way:

$$MTTF = AJ^{-n}e^{Q/kT}$$

where the role of the temperature is that increased temperature reduces MTTF and thus degrades chip lifetime. Heat by devices but also heat by Joule effect affect the temperature of a wire [14] [31].



(a) IR Drop phenomenon.



(b) Electrical-Thermal coSimulation process.

Figure 2: Explanation scheme for IR-Drop and the Electrical-Thermal coSimulation process.

1.2.5 The IR-drop problem

The IR-Drop problem appears due to the electrical resistance of a conductor. As shown in Fig. 2(a), a voltage $V_{drop} = R * I$ is created between the two ends of the conductor. As a consequence, there is a drop in the voltage available at the load devices, where $V_{load} = V_{supply} - V_{drop}$. Due to Joule heating effect, the flow of current through metal wires increases wire temperature while as electrical resistivity is temperature - dependent, it increases with temperarure rises. This leads to greater V_{drop} and finally lower V_{load} [11]. All these facts, greatly also explain why an electrical-thermal co-simulation

is required when considering the on chip power delivery network analysis as shown in Fig. 2(b) and presented in [15].

2 Chapter - Full Chip Thermal Analysis

2.1 The thermal PDE

There are three modes of heat transfer: conduction, convection and radiation. As we are interested in heat transfer in solids, we focus on the equation of heat conduction according to Fourier's law [19]:

$$q = -k_t \nabla T \tag{1}$$

which means that the flow of heat at a point per unit area and per unit time, is proportional to the temperature gradient at that point and the heat flows in the direction of decreasing temperatures. In equation (1), q is the heat flux, k_t is the thermal conductivity of the material and T the temperature.

Analyzing (1) we conclude to the following parabolic PDE:

$$\begin{aligned} &\nabla q = -k_t \nabla^2 T = g(r,t) - \rho C_p \frac{\partial T(r,t)}{\partial t} \Rightarrow \\ &\rho C_p \frac{\partial T(r,t)}{\partial t} = k_t \nabla^2 T(r,t) + g(r,t) \end{aligned} \tag{2}$$

where t the time, g the power density, C_p the heat capacity, ρ the density of the material and r the spatial coordinate of the point at which the temperature is being determined. Assuming steady state analysis, all derivatives with respect to time equal to zero so equation (2) amounts to Poisson's equation:

$$\nabla^2 T(r) = -\frac{g(r)}{k_t} \tag{3}$$

In many cases steady state analysis is sufficient, however, in many other cases transient analysis is required [29] [13]. Without loss of generality we focus on steady state thermal analysis.

The next step is the discretization of equation (3). There is a variety of methods used for such purpose. Finite Difference Method (FDM), Finite Element Method (FEM), Green Functions are more usual.

2.2 Finite Element Method

According to the Finite Element Method (FEM) [10], the design space is first discretized/meshed into elements such as tetrahedra or hexahedra.

Temperatures are then calculated at the nodes of the element, while temperatures elsewhere within the element are interpolated using the following function (for an hexahedral element):

$$T(x,y,z) = \sum_{i=1}^{8} N_i(x,y,z)T_i \tag{4}$$

where T_i the temperature at node i and N_i the shape function for node i. If (x_c, y_c, z_c) the center of the element and w, d, h the width, height and depth of the element respectively, $N_i(x,y,z)$ can be written as:

$$\begin{aligned} &N_i(x,y,z) = \\ &(\frac{1}{2} + \frac{2(x_i - x_c)}{w^2}(x - x_c)) \times (\frac{1}{2} + \frac{2(y_i - y_c)}{d^2}(y - y_c)) \times \\ &(\frac{1}{2} + \frac{2(z_i - z_c)}{h^2}(z - z_c)) \end{aligned} \tag{5}$$

Using equation (4), the thermal gradient g can be calculated as:

$$g = \begin{bmatrix} \frac{\partial T}{\partial x} \\ \frac{\partial T}{\partial y} \\ \frac{\partial T}{\partial z} \end{bmatrix} = BT \tag{6}$$

where $B = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \dots & \frac{\partial N_8}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \dots & \frac{\partial N_8}{\partial y} \\ \frac{\partial N_1}{\partial z} & \frac{\partial N_2}{\partial z} & \dots & \frac{\partial N_8}{\partial z} \end{bmatrix}$

Subsequently, stamps (in FEM they are called stiffness matrices K) are created for each element, using the variational method for an arbitrary element type:

$$K = \iiint_V B^T D B \mathrm{d}V \tag{7}$$

where V the volume of the element and $D = \begin{bmatrix} k_{t,x} & 0 & 0 \\ 0 & k_{t,y} & 0 \\ 0 & 0 & k_{t,z} \end{bmatrix}$ with $k_{t,i}, i \in x, y, z$ the thermal conductivities along x, y, z axis.

According the boundary condition case (convective, conductive etc), these stamps are accordingly calculated and together with the stamps from various

elements they are superposed to obtain the global stiffness matrix K_g in order to be incorporated to the global set of equations:

$$K_g T = P \tag{8}$$

where T the vector of all the unkonown temperatures and P the vector of power at the corresponding node [23].

2.3 Finite Difference Method

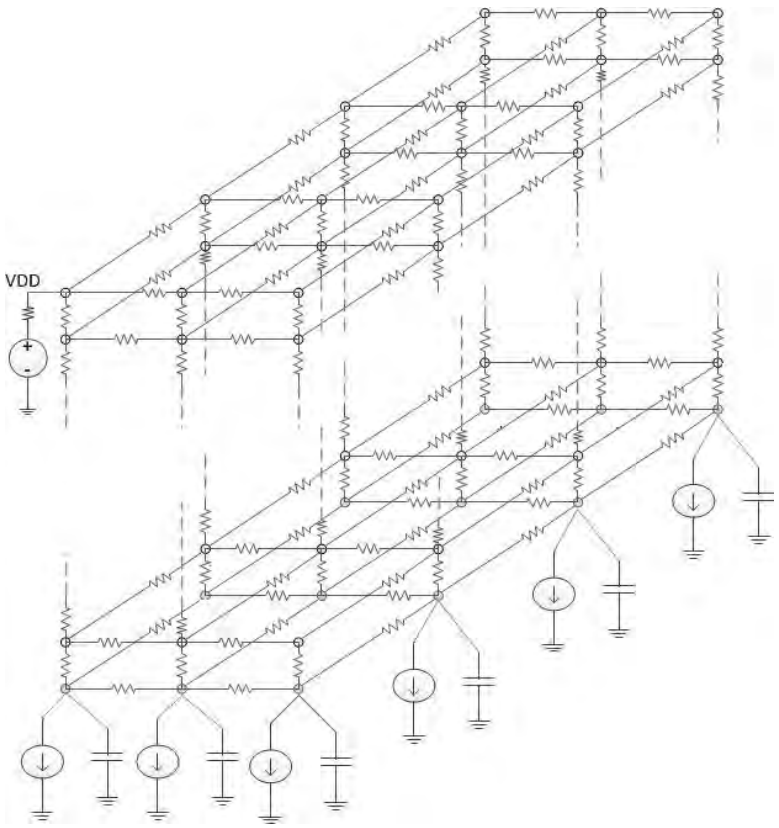


Figure 3: Modeling of a chip as a grid of resistances

This method discretizes the entire chip forming a system of linear equations relating temperature distribution with power density distribution. The basic philosophy behind FDM is the conversion of governing equations from continuous functions into their discretely sampled counterparts. The result

is a system of algebraic equations solvable for dependent variables at discrete grid points. Therefore the chip can be discretized into cuboidal regions with sides along the x, y, z axis. Consequently, the spatial derivative of temperature T can be written as a finite difference in rectangular coordinates. Assuming a region in which a vertex is at the origin of the first octant and $T_{i,j,k}$ the temperature at node $(i\Delta_x, j\Delta_y, k\Delta_z)$, in x-direction we have:

$$\frac{\partial^2 T(r)}{\partial^2 x} \simeq \frac{\frac{T_{i-1,j,k}-T_{i,j,k}}{\Delta_x} - \frac{T_{i,j,k}-Ti+1,j,k}{\Delta_x}}{\Delta_x} \left[\frac{T_{i-1,j,k}-T_{i,j,k}}{R_{i-1.j.k}} - \frac{T_{i,j,k}-T_{i+1,j,k}}{R_{i,j,k}}\right] \cdot \frac{\Delta_y \Delta_z}{k_t \Delta_x} \tag{9}$$

where $R_{i-1,j,k} = \frac{\Delta_x}{k_t \Delta_y \Delta_z}$ and $\Delta_{\{x,y,z\}}$ the length of the rectangle in each direction.

Similarly, the respective equations arise for y and z direction, while the following equation is finally produced:

$$\begin{aligned} &\left[\frac{T_{i-1,j,k}-T_{i,j,k}}{R_{i-1,j,k}} + \frac{T_{i+1,j,k}-T_{i,j,k}}{R_{i,j,k}}\right] + \\ &+ \left[\frac{T_{i,j-1,k}-T_{i,j,k}}{R_{i,j-1,k}} + \frac{T_{i,j+1,k}-T_{i,j,k}}{R_{i,j,k}}\right] + \\ &+ \left[\frac{T_{i,j,k-1}-T_{i,j,k}}{R_{i,j,k-1}} + \frac{T_{i,j,k+1}-T_{i,j,k}}{R_{i,j,k}}\right] = -g_{i,j,k}\Delta V \end{aligned} \tag{10}$$

with $g_{i,j,k}\Delta V$ the total power and $\Delta V = A_x\Delta_x = A_y\Delta_y = A_z\Delta_z$

This discretization reflects the well known thermal-electrical duality where each node in the discretization corresponds to a node in the circuit. Considering a network where the electrical resistances are mapped to thermal resistances connecting adjacent nodes and power sources are mapped to current sources, the thermal problem is equal to the solution of a circuit of linear resistors and current sources.

Finally, using MNA formulation we produce the following system of linear equations:

$$Gt = p \tag{11}$$

where t the temperature vector at each point, p the vector containing the total power generated within each element and G a sparse and symmetric matrix which contains the thermal resistors $G_x = \frac{k_t \Delta_y \Delta_z}{\Delta_x}, G_y = \frac{k_t \Delta_x \Delta_z}{\Delta_y}$ and $G_z = \frac{k_t \Delta_x \Delta_y}{\Delta_z}$ forming a grid of resistances as shown in fig (3)

Unfortunately, the use of FDM method fosters the risk of wasteful computations, as discretizing the entire chip a huge linear system has to be solved. However, GPUs promise to greatly alleviate this issue.

2.4 Green Functions

The basic concept is that the Green function G(r, r') finds the temperature at the point r when a power source is placed at location r'. Green function algorithms consider field and source planes. Field planes are the points where temperatures have to be calculated and source planes are the power sources. For more than one pair source-field plane, the temperature distribution can be obtained through superposition. More specifically, for a point (x,y) on the field plane and a point (x', y') on the source plane, the Green function can be written as :

$$G(x, y, x', y') = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} C_{mn} cos(\frac{m\pi x}{a}) cos(\frac{n\pi y}{b}) cos(\frac{m\pi x'}{a}) cos(\frac{n\pi y'}{b}) \quad (12)$$

Due to the fact that equation (12) contanis infinite sumations that may lead to long runtimes, equation (12) has to be further considered and analyzed [23].

2.5 Preferred Algorithm

In an attempt to compare the thermal simulation methods we can observe that FDM and FEM discretize the entire chip promising very high accuracy in thermal analysis. Handling complicated geometries such as nonuniform structures they become very flexible. However, accurate 3D thermal analysis using these methods in a full chip scale implies the solution of a huge linear system of equations with many unknowns, becoming very expensive. The basic difference between FDM and FEM is that FDM discretizes the differential operator while FEM discretizes the temperature field.

On the other hand, the Green function method analyzes only the layers whose temperature interests us. Consequently, the size of the resulting problem is relatively small and can be solved very fast. The basic problem is that this method usually assumes uniform distributions.

Finally, FDM appears the more preferable method for our purposes. In conjuction with the massively parallel architectures (such as GPUs) we bring

off an accurate thermal analysis model with near optimal computational complexity and low memory requirements [9] [23].

2.6 Solution Methods for Linear Systems

As the produced linear system in (11) is very large and sparse, a suitable method for its solution has to be chosen. Direct methods have been preferred till now for solving linear systems, mainly due to their robustness and predictable behavior. However, due to their execution time and memory requirements in case of large scale grids, there is a notable shift toward iterative methods. Exploiting matrices' attributes such as SPD (Symmetric and Positive Definite), efficient methods such as Conjugate Gradient can be used being more computationally and memory efficient for large sparse linear systems.

The biggest challenge with iterative methods is their convergence rate which scales according the properties of the matrix. For fast convergence an appropriate preconditioner, according to each specific problem and type of system matrix, has the responsibility to "smoothen" system's properties. In fact, the preconditioner involves a solver step $Mz = r$ in every iteration and solves the system $M^{-1}Ax = M^{-1}b$ instead of $Ax = b$. The role of the preconditioner can be enhanced exploiting parallel computations. Therefore, matrices produced by the thermal grid can be approximated by Fast Transform Solvers who minimize the number of iterations for the solution of the systems $Mz = r$ in a parallel environment [15].

2.7 Graphics Processing Units - GPUs

As mentioned above, 3D thermal analysis requires the solution of a huge linear system of equations with many unknowns. However, GPU-based parallel computing tranquilizes this issue.

To begin with, a GPU is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the building of images in a frame buffer intended for output to a display. Graphic chips started as fixed function graphics processors but became increasingly programmable and computationally powerful which led NVIDIA to introduce the first GPU. They are used in embedded systems, mobile phones, personal computers and game consoles. Due to their highly parallel structure they distinct for their effectiveness [3] [18]. They process large blocks of data in parallel. GPUs

have been widely used for scientific computation in the past few years thank to NVIDIA CUDA. GPUs can provide more than 10X higher computing capability than multicore CPUs [9]. Therefore, GPU computing is growing rapidly, becoming competitive in speed, programmability and price and promises great achievements due to its high computational throughput via it's massively parallel architecture.

3 Chapter - Interconnect Thermal Modeling

3.1 Overview

The purpose of this thesis is to investigate the influence of interconnections on temperature rise. Therefore, we begin examining LEF/DEF files and we parse the parts of the files that interest us. Considering the die area as a 3D grid of 1x1 micro we calculate the percentage of the block that each interconnect/via covers. As a next step, we model each block as 6 resistsances forming a general network of resistances. Finally, using Finite Differences Method, we calculate a matrix containing the necessary coefficients that we then insert to a parallel fast preconditioner.

3.2 LEF/DEF Files

3.2.1 DEF File

A DEF (Design Exchange Format) file is an open specification for representing physical layout of an integrated circuit in an ASCII format. It contains the design-specific information of a circuit and is a representation of the design at any point during the layout process. DEF was developed by Cadence Design Systems. DEF conveys logical design data to, and physical design data from, place-and-route tools. Logical design data can include internal connectivity (represented by a netlist), grouping information, and physical constraints. Physical data includes placement locations and orientations, routing geometry data, and logical design changes for backannotation.They are usually generated by place and route tools. It is used in conjuction with the LEF File [1] [2].

3.2.2 LEF File

A LEF (Library Exchange Format) file is an open specification for representing physical layout information on components of an integrated circuit in an ASCII format. The LEF contains library information for a class of designs. Library data includes layer, via, placement site type, macro cell definitions. The LEF file is strongly connected with the DEF file [1] [2].

3.3 Description of the files implemented

3.3.1 File: global.h

Global variables visible to all files are defined here:
x, y: variables of type integer which define the x and y coordinates of the die area
units: is of type integer and it represents the size of the die area
maxLayerNumber is a variable of type integer used to count the number of metal layers in each circuit
kSi and **kCu**: variables of type int representing the thermal conductivities of Si and Cu
A struct gArrayInfo is also defined here for a two dimensional and a three dimensional array of structs gArrayInfo. Each struct contains information about the coordinates of a block at the grid, its six conductivities along x, y, z axis and the percentages of metal influence.

3.3.2 File: layerMetalList.h

It is used for the parsing of the .lef file of a circuit.

The structs and the prototypes of the functions implemented in layerMetalList.c are defined here. The information needed and stored from the .lef file is about the type of layer: metal, via or default via (an array of vias). If we examine a metal we are interested in its name(for example metal1) in order to know the metal's number, its width and pitch, its direction (horizontal or vertical), its offset, thickness and height. If we examine a via, we are interested in its name (for example via1) in order to know the via's number, its spacing and width. If we examine a default via which is a rectangle area of vias, we are interested in its name, the according via layer name, the metals it connects, the coordinates of the rectangle area etc.

Finally, a struct Layers is defined which constitutes the node of a list storing information about metals, vias and defaultVias. Each node of this list is a pointer to a struct metalLayer, a struct viaLayer or a struct defaultVia.

3.3.3 File: layerMetalList.c

It is used for the parsing of the .lef file of a circuit.

The functions defined in layerMetalList.h and implemented here are the following:

struct Layers *addLayer(struct Layers *head, struct metalLayer *mLayer, struct viaLayer *vLayer, struct defaultVia *defVia)
This function adds a node to the list of structs of type Layers

int findThicknessUsingMetalLayerNumber(struct Layers *head, int layerNumber):
This function takes as input the layer number of the metal and returns its thickness

int findViaThicknessUsingMetalLayerNumber(struct Layers *head, int layerNumber):
This function takes as input the layer number of the via and returns its thickness

void findWidthSpacing(struct Layers *head, char *viaName, double *width, double *spacing):
This function takes as input a viaName (via1 for example) and returns its width and spacing

void printLayers(struct Layers *head):
This function prints the contents of the list

void findWidth(struct Layers *head, char *metalLayerNumber, double *width):
This function takes as input the layerName (metal1 for example) runs the list and returns the width of this metal

char *findViaName(struct Layers *head, char *defaultviaName):
This function takes as input the viaName (via1-4 for example) and returns the according via layer (via1 for example)

void clearLayerMetalList(struct Layers *lHead),
void removeViaLayerItem (struct viaLayer *vLayer),
void removeDefaultViaItem (struct defaultVia *dVia),
void removeMetalLayerItem(struct metalLayer *mLayer),
struct Layers *removeLayerMetalListItem(struct Layers *lItem)
These functions are all used to free the nodes of the list.

3.3.4 File: lefParser.h

It is used for the parsing of the .lef file of a circuit.

3.3.5 File: lefParser.c

It is used for the parsing of the .lef file of a circuit.
The functions defined in lefParser.h and implemented here are the following:

void parseLefFile(char *filename):
This function uses the following auxiliary functions in order to parse the .lef file. According to the type of layer, the appropriate information is stored into the list.

char *parseSecondToken(char *line):
This function takes as input a line of the form LAYER metal2 and returns the second token

void parseSecondIntToken(char *line, double *term):
This function reads a line and returns the second token in double

void parseEnclosure(char *line, double *x, double *y):
This function reads a line of the form ENCLOSURE 0.035 0 and returns the x, y coordinates

void parseSpacing2(char *line, double *start, double *step):
This function reads a line of the form SPACING 0.15 BY 0.15 and returns the two double values

void parseOffset(char *line, double *offsetX, double *offsetY):
This function reads a line of the form OFFSET 0.095 0.07 and returns the two double values

void parseRect(char *line, double *x1, double *y1, double *x2, double *y2):
This function reads a line of the form RECT -0.4 -0.4 0.4 0.4 and returns the 4 double values

3.3.6 File: metalList.h

It is used for the parsing of the .def file of a circuit.
It contains the definitions of the structs and the prototypes of the functions implemented in metalList.c. The information needed and stored from the .def file is about the type of metal: specialNetInterconnection, netInterconnection, specialNetVia, netVia. According to each of these types the appropriate infomation is stored into a list of structs of type Metal.

3.3.7 File: metalList.c

It is used for the parsing of the .def file of a circuit.
The functions defined in metalList.h and implemented here are the following:

void printMetals(struct Metals *head):
This function prints the contents of the list

struct Metals *add(struct Metals *head, struct Interconnections *inter, struct Vias *v, struct NetInterconnections *nInter, struct NetVias *nV):
This function adds a struct of type Metal to the list

void removeViaItem(struct Vias *viaItem),
void clearMetalList(struct Metals *mHead),
void removeNetViasItem(struct NetVias *nViaItem),
struct Metals *removeMetalListItem(struct Metals *mItem),
void removeInterconnectionsItem(struct Interconnections *interItem),
void removeNetInterconnectionItem(struct NetInterconnections *nInterItem)
These functions are all used to free the nodes of the list.

3.3.8 File: parser.h

It is used for the parsing of the .def file of a circuit.
All the functions implemented in parser.c are defined here.

3.3.9 File: parser.c

It is used for the parsing of the .def file of a circuit.
Here are the implementations of all the functions for the parsing of .def file.

void parseFile(char *filename):
This function uses the following auxiliary functions in order to parse the .def file and accordingly store information into the list.

void parseNets(char *line):
This function reads a line and whether it refers to a netVia or a netInterconnection it calls the according functions parseNetVias and parseNetInterconnections.

void parseUnits(char *line):
This function reads a line of the form UNITS DISTANCE MICRONS 2000 and sets the global variable units to the distance number read.

void parseNetVias(char *line):
This function stores all the necessary information concerning a netVia into the list.

void parseDieArea(char *line):
This function reads a line of the form DIEAREA (0 0) (29290 22880) and sets the global x and y coordinates accordingly.

char *parseViaRule(char *line):
This function takes as input a line of the form VIARULE Via1Array-0 and returns the viaArrayName (for example Via1Array-0)

void parseNetInterconnections(char *line):
This function stores all the necessary information concerning a netInterconnection into the list.

void parseVia(char *position, char *line):
This function stores all the necessary information concerning a specialnet via into the list.

void parseInter(char *position, char *line):
This function stores all the necessary information concerning a specialnet interconnection into the list.

void parseMetalWire(char *position, char *line):
This function reads the input line and whether it concerns wire or via, it calls the function parseInter(char *position, char *line) or parseVia(char *position, char *line)

void parseViaCutSize(char *line, int *cutSizeX, int *cutSizeY):
This function reads the input line and isolates the x, y values of the cut size.

void parseViaRowCol(char *line, int *numCutRows, int *numCutCols):
This function reads the input line and isolates the number of rows and columns of the via array.

void parseViaMetalLayer(char *line, int *layerFrom, int *layerTo):
This function stores the layers that a via connects.

void parseViaCutSpacing(char *line, int *cutSpacingX, int *cutSpacingY):
This function stores the x and y spacing coordinates between cuts.

void parseViaEnclosure(char *line, int *botEnclX, int *botEnclY, int *topEnclX, int *topEnclY):
This function stores the x and y enclosure values for the bottom and top metal layers of vias.

3.3.10 File: viaList.h

This file includes definitions of structs and the prototypes of functions implemented in the file viaList.c

3.3.11 File: viaList.c

Here is the implementation of the functions used for controlling vias.

struct viaInfo *addViaInfo(struct viaInfo *head, char *viaRule, int layerFrom, int layerTo, int cutSizeX, int cutSizeY, int cutSpacingX, int cutSpacingY, int botEnclX, int botEnclY, int topEnclX, int topEnclY, int numCutRows, int numCutCols):
This function adds a node containing infoprmation about a via into the list.

void findUsingArrayName(struct viaInfo *head, char *viaRule, int *from, int *to, int *spacingX, int *spacingY, int *sizeX, int *sizeY, int *botX, int *botY, int *topX, int *topY, int *numRows, int *numCols):
This function runs the list of vias and stores basic information about a viaArray given as input.

void printViaInfo(struct viaInfo *head):
This function prints the contents of the list.

void clearViaList(struct viaInfo *vHead):
struct viaInfo *removeViaListItem(struct viaInfo *item):
These functions are all used to free the nodes of the list.

3.3.12 File: percentages.h

This file includes the prototypes of the functions implemented in the file percentages.c

3.3.13 File: percentages.c

This file implements the functions defined in the file percentages.h

void definePercentage():
This function runs the list of sructs of type Metals and whether it finds pointer to struct of type Interconnections, NetInterconnections, NetVias or

Vias, makes the essential transformations and calls the function calculateArea().

void calculateArea(char *type, char *typeOfMetal, char *direction, int layerNumber, int thickness, int x1, int y1, int x2, int y2):
This function first according to the coordinates of each metal and the coordinates of the block of the die area that it covers calculates the percentage of the metal on the block. The result is stored into a list of structs of type layerNum.

3.3.14 File: percentagesList.h

It contains the definitions of the structs and the prototypes of the functions implemented in percentagesList.c.

3.3.15 File: percentagesList.c

struct layerNum *addLayerNum(struct layerNum *head, struct layerList *layerPointer,int layer):
This function adds a node to the list of structs of type layerNum.

void printlayerNum(struct layerNum *head):
This function prints the contents of a list of structs of type layerNum.

struct layerList *addLayerList(struct layerList *head, char *type, char *typeOfMetal, char *direction, int dieBlockX1, int dieBlockY1, int dieBlockX2, int dieBlockY2, int thickness, double percentage):
This function adds a node to a list of structs of type layerList.

void printlayerList(struct layerList *head):
This function prints the contents of a list of structs of type layerList.

void addElement(int layer, char *type, char *typeOfMetal, char *direction, int dieBlockX1, int dieBlockY1, int dieBlockX2, int dieBlockY2, int thickness, double percentage, struct layerNum *layerNumHead):
This function links a node of the list of structs of type layerNum with a list

of structs of type layerList.

void printAll(struct layerNum *layerNumHead):
This function prints the contents of a list of structs of type layerNum with each node linked with a list of structs of type layerList.

void clearLayerNumList(struct layerNum *layerNumHead),
struct layerNum *removeLayerNumItem(struct layerNum *lNItem),
void clearLayerList(struct layerList *lListHead),
struct layerList *removeLayerListItem(struct layerList *item):
These functions are all used to free the nodes of the lists.

3.3.16 File: resistancesList.h

This file includes the definitions of the functions implemented in resistancesList.c.

3.3.17 File: resistancesList.c

struct gArrayInfo * create3DByteMatrix(int layers, int cols, int rows)**:
This function creates a 3D array of dimensions: layers x cols x rows and initializes it with zeros. It is an array of structs of type gArrayInfo, thus this function also sets the fields dieBlockX1, dieBlockY1, dieBlockX2, dieBlockY2 accordingly in order to have blocks of 1x1 microns concerning the value units of the die area. The other fields are set to zero.

void printArray3D(struct gArrayInfo * array, int layers, int cols, int rows)**:
This function prints the contents of the 3D array.

struct gArrayInfo * add3DCoord1(struct gArrayInfo ***array, int i, int j, int k, int dieBlockX1, int dieBlockY1, int dieBlockX2, int dieBlockY2, double gxR, double gxL, double gyU, double gyD, int metalInfluence)**:
This function sets values into the fields gxR, gxL, gyU, gyD and metalInflu-

ence. It is called when an interconnection is found.

struct gArrayInfo * add3DCoord2(struct gArrayInfo ***array, int i, int j, int k, int dieBlockX1, int dieBlockY1, int dieBlockX2, int dieBlockY2, double gzR, double gzL, int viaInfluence)**:
This function sets values into the fields gzR, gzL and viaInfluence. It is called when a via is found.

3.3.18 File: calculateResistances.h

This file includes the definition of the function implemented in calculateResistances.c.

3.3.19 File: calculateResistances.c

void fillBlocks():
This function creates the global 3D array of structs of type gArrayInfo. It then calculates the values of the conductancies along x, y, z axis and sets the according fileds of the array. The calculation of the conductancies is as follows: If we have interconnection in horizontal position, the x conductance is influenced by the metal and y conductance by the oxide. Therefore, if the metal fully covers the block: $g_x = g_y$ = 2 * kCu * thickness / percentage, otherwise g_x = 2 * kSi * thickness / (1-percentage) + 2 * kCu * thickness / percentage and g_y = 2 * kSi * thickness / 1. If two or more metals exist in the same block we consider their in parallel connection, so we add the partial conductivities.
If we have interconnection in vertical position, the y conductance is influenced by the metal and x conductance by the oxide. If the metal fully covers the block: $g_x = g_y$ =2 * kCu * thickness / percentage, otherwise g_y = 2 * kSi * thickness / (1-percentage) + 2 * kCu * thickness / percentage and g_x = 2 * kSi * thickness / 1.
When a via is found in a block, we have to calculate the gz conductance. If it fully covers the block, g_z=2 * kCu * thickness / percentage, otherwise g_z=2 * kSi * thickness / (1-percentage) + 2 * kCu * thickness/percentage.
In each case, when no interconnection is met, $g_x = g_y$ =2*kSi*thickness and when no via is met, g_z=2*kSi*thickness.

Finally this function prints the 3D array.

3.3.20 File: csparse.h, csparse.c

These files are used for sparse arrays which are used in Finite Differences Method. They define and implement the essential functions.

3.3.21 File: matrix.h

This file includes the definitions of the functions implemented in matrix.c.

3.3.22 File: matrix.c

void thermal-sim():
The basic function that calls the partial following functions in order to create the fdm matrix. First, it calls initializeThermal3D() and then it creates the fdm matrix.

void initializeThermal3D(int *dimensions):
This function initializes some constants used for fdm matrix creation.

void create-fdm-matrix-simple3D (int *dimensions, cs *M):
This function creates the fdm matrix.

int expand3D(int x, int y, int z, int *dimensions):
This function converts a 3D node to 1D.

3.3.23 File: main.c

This function parses the .lef and .def files, calculates the percentage of the metal at each 1x1 micron block and finally calls thermal-sim to create the fdm matrix.

3.4 Assumptions during parsing of LEF/DEF files

3.4.1 Assumptions concerning specialNet Interconnections

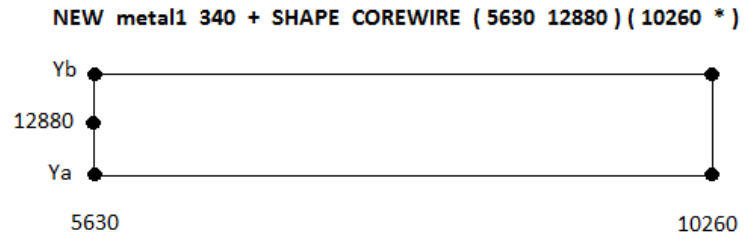


Figure 4: Case for an horizontal specialnet interconnection

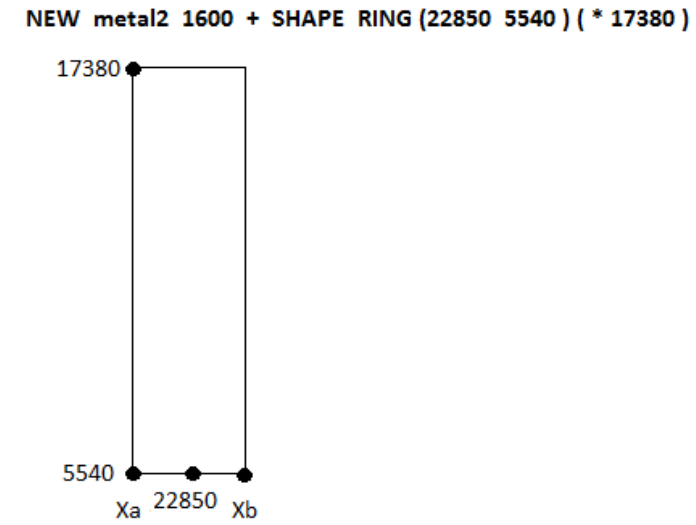


Figure 5: Case for a vertical specialnet interconnection

Horizontal SpecialNet Interconnection

Consider the example of the following row in the .def file:
NEW metal1 340 + SHAPE COREWIRE (5630 12880) (10260 *)
as shown in fig (4).

To begin with, deciphering the coordinates given in this line, X1 = 5630, Y1 = 12880, X2 = 10260 and Y2 = Y1. When the metal is HORIZONTAL (Y1 = Y2), the center line coordinate is Y1 = Y2. Therefore, in order to find the two limits Ya, Yb:
Ya = Y1 - width/2, Yb = Y1 + width/2 where width = 340.

Vertical SpecialNet Interconnection

Consider the example of the following row in the .def file:
NEW metal2 1600 + SHAPE RING (22850 5540) (* 17380)
as shown in fig (5).

When the metal is VERTICAL (X1 = X2), the center line coordinate is X1 = X2. Therefore, in order to find the two limits Xa, Xb:
Xa = X1 - width/2, Xb = X2 + width/2 where width = 1600.

3.4.2 Assumptions concerning specialNet Vias

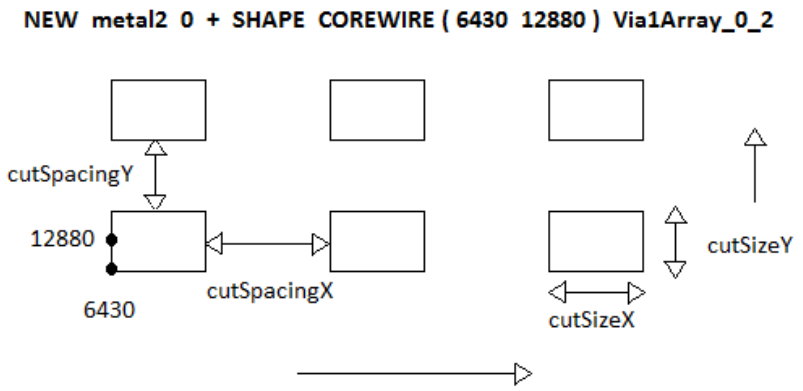


Figure 6: Case for a specialnet via

First, to distinguish a specialnet via from a specialnet interconnection, we consider the width. If its value is 0, then we have a via. Consider the example of the following row in the .def file:
NEW metal2 0 + SHAPE COREWIRE (6430 12880) Via1Array-0-2
as shown in fig (6).

According to Via1Array-0-2 and its information on the relevant part in the .def file we find the cutSize, cutSpacing and RowCol values. The RowCol

value defines the number of rows and columns that form the arrray of vias. According to the coordinates given, the vias are created from bottom to up and from left to right. The cutSizeX, cutSizeY, cutSpacingX, cutSpacingY values are considered as shown in fig (6).

3.4.3 Assumptions concerning net Interconnections

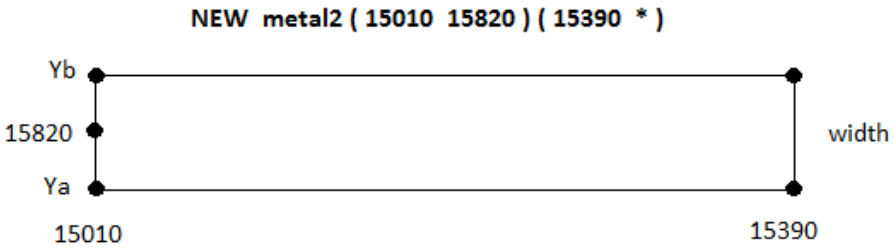


Figure 7: Case for a net interconnection

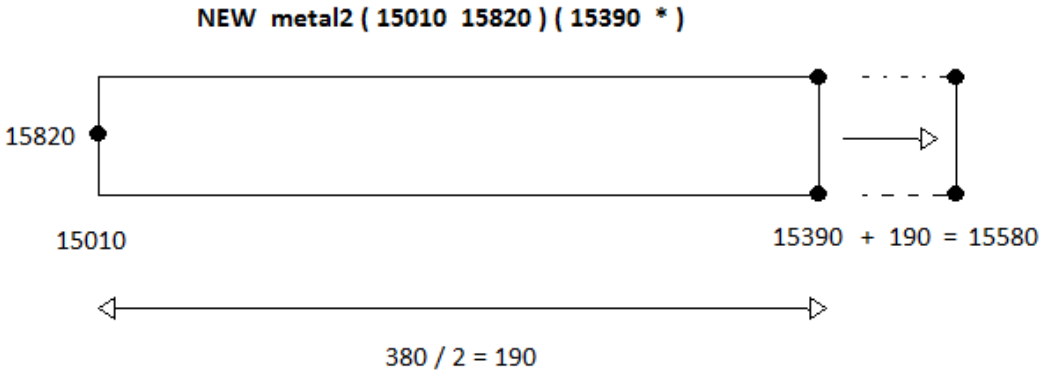


Figure 8: Case for a net interconnection considering extValue

Consider the example of the following row in the .def file:
NEW metal2 (15010 15820) (15390 *)
as shown in fig (7).

In this case, in order to find the width, we run the .lef file and we store the width of the corresponding metal (for example metal2). Therefore:

Ya = 15820 - width/2, Yb = 15820 + width/2.
In the case of NETS we have to also consider the extValue. If extValue is not given, its default value is length/2. In horizontal mode, extValue concerns only x values. For the first endpoint (X1, Y1), the default extValue is ignored as length/2 value will be surely involved inside length value. For the second endpoint (X2, Y2) X2 will be extended by length/2 where length = X2 - X1, as shown in fig (8).

Similarly, in vertical mode, we extend on y - axis.

3.4.4 Assumptions concerning net Vias

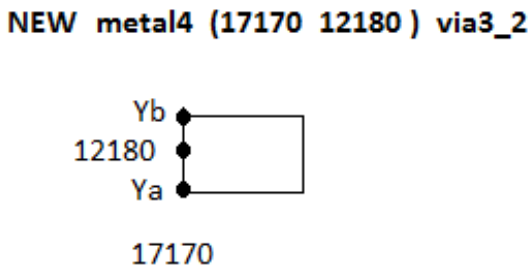


Figure 9: Case for a net via

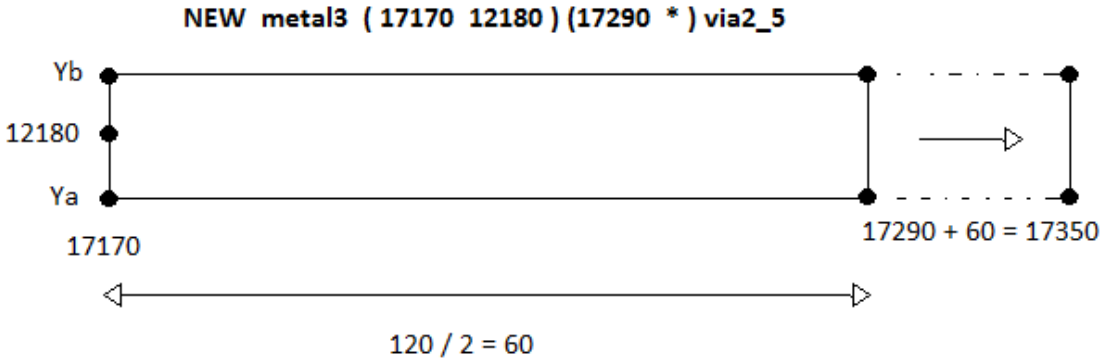


Figure 10: Case for a net via considering extValue

Via with one pair of coordinates

Consider the example of the following row in the .def file:
NEW metal4 (17170 1582012180) via3-2
as shown in fig (9).
Considering via3-2 we run the .lef file in order to find the layer that it corrresponds to(via2 for example). We then find the width of this via layer. The width is given in microns so we have to multiply it with units. According to width we calculate Ya and Yb. We also suppose that vias are in horizontal mode, thus the center line coordinate concerns the y axis. In this case we ignore extValue.

Via with two pairs of coordinates

Consider the example of the following row in the .def file:
NEW metal3 (17170 12180) (17290 *) via2-5
as shown in fig (10).

According to the via2-5 we find the via that it corresponds to, its width and spacing. Finding the width we can then calculate Ya and Yb. We take into account extValue, so X2 or Y2 will be extended according to the logic discussed in previous case. In the rectangular formed by the coordinates given, we conside vias of width*width by a distance of spacing.

3.5 Calculation of resistances forming the R model of the chip

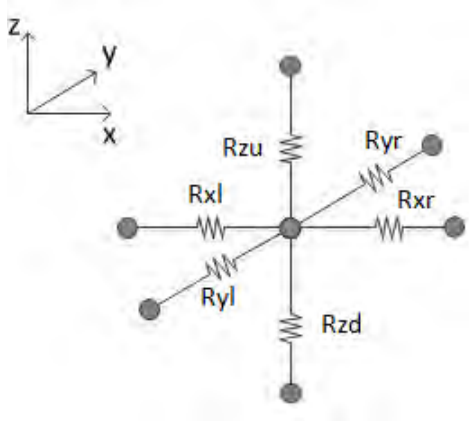


Figure 11: Modeling of a 1x1 micron block as 6 resistances along x, y, z axis

As mentioned above, each block 1x1 micron is modeled as 6 resistances (fig 11). The type for the thermal resistance is given by:
$R_{thermal} = \frac{l}{k*t*w}$
In general, the thermal resistance along each axis will be calculated as the parallel combination of the thermal resistance influenced by metal and the thermal resistance influenced by oxide. If metal in horizontal position exists in a block, it contributes to x resistance while to y resistance contributes only the oxide. Respectively, if metal in vertical position exists in a block, it contributes to y resistance, while to x resistance contributes the oxide. Finally, if many metal contributions exist into a block we calculate the resistances and take their parallel combination.

References

[1] *LEF/DEF Language Reference*.

[2] *Wikipedia, Design Exchange Format, Library Exchange Format*.

[3] *Wikipedia, Graphics Processing Unit*.

[4] *Wikipedia, Joule heating effect*.

[5] K. Banerjee, A. Mehrotra, A. Sangiovanni-Vincentelli, and C. Hu, "On thermal effects in deep sub-micron vlsi interconnects," in *Design Automation Conference, 1999. Proceedings. 36th*.

[6] K. Banerjee, M. Pedram, and A. H. Ajami, "Analysis and optimization of thermal issues in high-performance vlsi," in *in ACM/SIGDA Int. Symp. Physical Design (ISPD*, 2001, pp. 230–237.

[7] X. Chen, "Performance, power, and thermal modeling and optimization for high-performance computer systems," Ph.D. dissertation, University of Michigan, 2011.

[8] Y.-K. Cheng, P. Raha, C.-C. Teng, E. Rosenbaum, and S.-M. Kang, "Illiads-t: an electrothermal timing simulator for temperature-sensitive reliability diagnosis of cmos vlsi chips," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 8, pp. 668 –681, 1998.

[9] Z. Feng and P. Li, "Fast thermal analysis on gpu for 3d-ics with integrated microchannel cooling," in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, 2010, pp. 551 –555.

[10] B. Goplen and S. Sapatnekar, "Efficient thermal placement of standard cells in 3D ICs using a force directed approach," in *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, 2003.

[11] C. Halford, "Ir-drop analysis," Advanced Layout Solutions Ltd, Tech. Rep.

[12] C. Hu, "Future cmos scaling and reliability," *Proceedings of the IEEE*, vol. 81, no. 5, pp. 682 –689, 1993.

[13] W. Huang, "Hotspota chip and package compact thermal modeling methodology for vlsi design," Ph.D. dissertation, University of Virginia, 2007.

[14] A. A. Khan, "Design tools for reliability analysis."

[15] N. E. G. S. P. T. Konstantis Daloukas, Alexia Marnari, "Fast Electrical-Thermal Co-Simulation of Large-Scale Power Delivery Networks on Massively Parallel Architectures ."

[16] P. Li, L. Pileggi, M. Asheghi, and R. Chandra, "Ic thermal simulation and modeling via efficient multigrid-based approaches," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 9, pp. 1763 –1776, 2006.

[17] P. Liu, H. Li, L. Jin, W. Wu, S. X.-D. Tan, and J. Yang, "Fast thermal simulation for runtime temperature tracking and management," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 2882 –2893, 2006.

[18] X.-X. Liu, Z. Liu, S.-D. Tan, and J. Gordon, "Full-chip thermal analysis of 3d ics with liquid cooling by gpu-accelerated gmres method," in *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, 2012, pp. 123 –128.

[19] M. N. Ozisik, "Heat transfer: A basic approach." NY:McGraw-Hill.

[20] S. Pan, N. Chang, and J. Zheng, "A new methodology for ic-package thermal co-analysis in 3d ic environment," in *Electrical Design of Advanced Packaging Systems Symposium (EDAPS), 2010 IEEE*, 2010, pp. 1 –4.

[21] M. Pedram and S. Nazarian, "Thermal modeling, analysis, and management in vlsi circuits: Principles and methods," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1487 –1501, 2006.

[22] H. Qian and S. Sapatnekar, "Fast poisson solvers for thermal analysis," in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, 2010, pp. 698 –702.

[23] S. S. Sapatnekar, "Thermally aware design (chapter 2)."

[24] A. Sridhar, A. Vincenzi, M. Ruggiero, and D. Atienza, "Neural network-based thermal simulation of integrated circuits on gpus," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 1, pp. 23 –36, 2012.

[25] B. Wang and P. Mazumder, "Accelerated chip-level thermal analysis using multilayer green's function," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 325 – 344, 2007.

[26] T.-Y. Wang and C. C.-P. Chen, "3-d thermal-adi: a linear-time chip level transient thermal simulator," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 12, pp. 1434 – 1445, 2002.

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

[28] H. Xu, V. Pavlidis, and G. De Micheli, "Analytical heat transfer model for thermal through-silicon vias," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, 2011, pp. 1 –6.

[29] S. S. S. Yong Zhan, Sanjay V.Kumar, "Thermally aware design (chapter 1)."

[30] T. yuan Wang, Y. min Lee, C. Chen, and C. C. ping Chen, "3d thermal-adi - an efficient chip-level transient thermal simulator," in *ACM International Symposium on Physical Design (ISPD), ACM 1581136501/03/0004*, 2003, pp. 10–17.

[31] S. Zhan, Kumar, "Thermally aware design (chapter 3)."