September 28, 2010

3D-ICE 1.0

This document provides a brief summary of the usage of 3D-ICE 1.0. This includes illustrative examples of generating the input stack and floorplan files, and the various functions used for printing the results.

User Guide

Table of Contents

1. License and Copyright

This file is part of 3D-ICE-1.0.

3D-ICE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.

3D-ICE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with 3D-ICE. If not, see http://www.gnu.org/licenses/ .

Authors:

Arvind Sridhar[*]
Alessandro Vincenzi[*]
Martino Ruggiero[*]
Thomas Brunschwiler[†]
David Atienza[*]

Contact Information:

EPFL-STI-IEL-ESL
Bâtiment ELG, ELG 130
Station 11
1015 Lausanne, Switzerland

Email: 3d-ice@listes.epfl.ch
(SUBSCRIPTION NECESSARY!)
URL: http://esl.epfl.ch/3d-ice.html

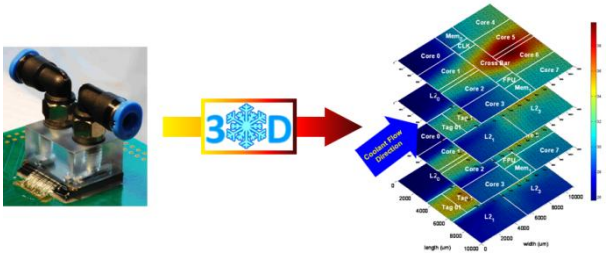
This research has been partially funded by the Nano-Tera RTD project CMOSAIC (ref.123618) - which is financed by the Swiss Confederation and scientifically evaluated by SNSF, and the PRO3D project- financed by the European Community 7th Framework Programme (ref.FP7-ICT-248776).

[*] Embedded Systems Laboratory, Department of Electrical Engineering, EPFL, Lausanne, Switzerland.
[†] Advanced Thermal Packaging Group, IBM Research Laboratory, Zurich, Switzerland.

2. Who needs 3D-ICE?

3D-ICE, or "**3D Interlayer Cooling Emulator**", is a Linux based *generic simulation platform* written in C to simulate the transient thermal behavior of 3D IC structures with inter-tier microchannel heat sinks. It is intended for various purposes including, but not limited to,

- Performing thermal analysis of 2D or 3D ICs during early stages of VLSI circuit/architecture design by electronic engineers,
- Simulating run-time thermal management strategies such as DVFS, dynamic work allocation, variable coolant flow rate etc.,
- Testing of microchannel heat-sink performances by microfabrication engineers and heat-sink designers,
- Evaluating accuracies of new and existing heat transfer correlations by experimental heat transfer engineers.



3D-ICE is based on the conventional compact modeling of heat transfer by conduction in solids, and advances a novel compact modeling methodology, called the Compact Transient Thermal Modeling (CTTM), for heat transfer by convection in microchannels. The user is free to use microchannel heat sinks of any dimension with the corresponding heat transfer performance data depending upon the accuracy/speed needs of the user. This simulator is ideal for situations where a quick estimate of chip temperatures is required, when the electronic designer is still iterating between various floorplanning and operating strategies in order to optimize for electronic performance and thermal safety/reliability of the final system.

In addition, the format of inputs, outputs and the problem construction/solving in 3D-ICE have been modeled on the popular compact modeling simulator **HotSpot**, making it easier for users who are familiar with this tool. With numerous functions to access a variety of thermal data during the simulation, the user can reach deep into the heart of the thermal simulation, use the data to interface with other (popular or custom) tools and automate any kind of design/run-time optimization algorithms using 3D-ICE. More functionality will be added in the future versions of 3D-ICE to make this interfacing even more automatic and easy.

For more details on the theory and discussions about the accuracy/speed of the modeling technique, please refer to the publication "3D-ICE_ICCAD2010.pdf", published in ICCAD 2010, available with this library.

3. Before you start

The 3D-ICE library has been written and developed using:

- bison 2.4.1
- flex 2.5.35
- gcc 4.1.2

Make sure that these tools are installed on your system before compiling and that the corresponding variables in makefile.def point to the respective binary file. To use 3D-ICE, you must also download the SuperLU library, available at http://crd.lbl.gov/~xiaoye/SuperLU/ .

A. Compile SuperLU

Before compiling 3D-ICE, you must compile the SuperLU library by executing the following commands:

```
$ wget http://crd.lbl.gov/~xiaoye/SuperLU/superlu_4.0.tar.gz

$ tar xvfz superlu_4.0.tar.gz

$ cd SuperLU_4.0/

$ cp MAKE_INC/make.linux make.inc
```

Next, check and edit the `SuperLUroot` variable in `./make.inc` and select the `blas` library before compiling. You can either use a `blas` library installed on you system or the `blas` library supplied by the authors of SuperLU (see the README file). If you decide to use the former then the variable BLASDEF must be set and `BLASLIB` must point to your `blas` library. Then compile SuperLU with

```
$ make
```

For the latter case, `BLASDEF` must be unset and BLASLIB must point to the `./libblas.a` archive. Then compile SuperLU with

```
$ make blaslib

$ make
```

These are the operations that can be done when compiling SuperLU on a generic Linux platform. In case of a different architecture, please reference to the README file.

B. Compile 3D-ICE

Check and edit the `SLU_MAIN` variable in `./makefile.def` to make it point to the main folder of SuperLU. Next, select the value of `SLU_LIBS` according to the choice done above when compiling SuperLU. You can then compile 3D-ICE with:

```
$ tar xvfz 3D-ICE-1.0.tar.gz

$ cd 3D-ICE

$ make
```

4. Inputs to 3D-ICE

This chapter describes the writing of input files for 3D-ICE. See the Stack Description File and Floorplan files in the `./examples` folder for reference.

A. Convention and Terminology

3D-ICE is based on the compact modeling of heat flow in solids and liquids applied to a 3D-IC structure with microchannel cooling. As quick recap, the structure is divided into cuboidal *thermal cells* based on the discretization parameters you provide. Next, thermal conductance for heat flow through each face of the cuboid is calculated and connected to the neighboring cells at these faces. Also, a capacitance representing the heat capacity of the cell is calculated. In 3D-ICE, both Cartesian coordinates and cardinal directions are used to describe the location of cells/nodes and direction of heat flow in the structure. The indices of cells/nodes along the *x* direction (WEST-EAST) are sometimes referred to as *columns*, the indices along the *y* direction (SOUTH-NORTH) are sometimes referred to as *rows*, and the indices along the *z* direction (BOTTOM-TOP) sometimes referred to as *layers*. Also, the word *length* is used primarily to refer to dimensions in the *x* direction, the term *width* is used for the *y* direction and the term *height* is used for the *z* direction unless otherwise specified. A typical solid thermal cell along with the coordinate systems used in the library is shown in Fig. 1.

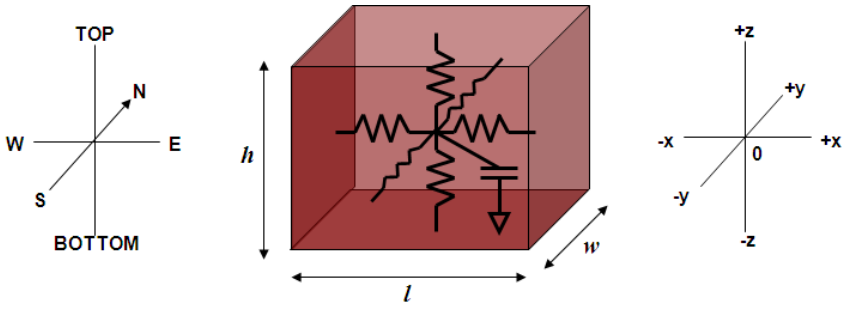
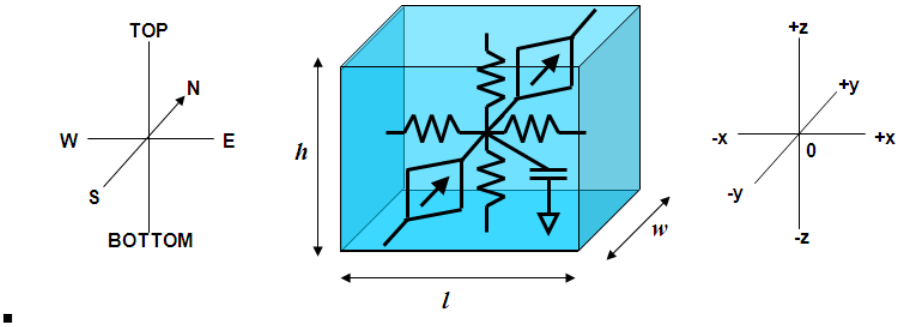


Figure 1: A typical solid thermal cell

The corresponding model for a liquid cell is shown in Fig. 2. Note that the current sources shown here correspond to the fluid flow in the microchannels. In this library, only flow direction NORTH or *+y* is supported, and hence, when microchannels used in a 3D-IC structure, they are always laid out facing SOUTH-NORTH (with the inlet being at the southern end and the outlet at the northern end of the channels). Hence, the northern edge of the IC is expected to be the hottest and the southern end of the IC is expected to be the coldest in any analysis. It is up to you to decide your floorplanning of the ICs accordingly.

3D-ICE accesses all the information needed to emulate a 3D IC from two different types of input file:

- Stack Description File
- Floorplan File



- **Figure 2: A typical microchannel thermal cell**

The syntaxes used in this document for describing how these input files must be written are based on the following convention:

`[: … :]`	POSIX characters class
`… | …`	OR- either of the two elements must be used
`[ … ]?`	an optional element
`[ … ]+`	one or more of this element must be used
`[ … ]*`	zero or more of this element must be used

Within the files, keywords must be written in low case and all the white spaces belonging to

```
[:space:]
```

will be skipped during the parsing. Identifiers (referred to as ID) must match the following expression:

```
[:alpha:] [ _ | [:alnum:] ]*.
```

Floating points values (referred to as DVALUE) must belong to

```
[+|-]? [:digit:]+ [ \. [:digit:]+ [ [e|E] [+|-]? [:digit:]+ ]? ]? .
```

Please refer to the flex sources in `3D-ICE/flex` for more details. It is also possible to insert comments at the end of a line (`//`) or to comment an entire block (`/* ... */`) of the input files (similar to C or Java).

B. Stack Description File

The stack description file (*.stk) is a netlist that specifies all the physical and geometrical properties of the 3D-IC for the simulation. The extension of the file is not relevant- it will be parsed independent of its presence or content.

The stack description file contains six main sections (mandatory and optional) and they must be declared following this order:

1. Materials
2. Conventional Heat Sink

3. Channel
4. Dies
5. Stack
6. Dimensions

The following description of each of these sections is not in the above mentioned order for ease of presentation and coherence.

Materials

The first section of the file contains the list of materials and their properties to be used in the simulation. At least one material must be declared. Materials are declared with the syntax,

```
material MATERIAL_ID :

       thermal conductivity     DVALUE ;

       volumetric heat capacity DVALUE ;
```

where

- `MATERIAL_ID` is a unique identifier to refer to this material,
- `thermal conductivity` is expressed in W/µm K ,
- `volumetric heat capacity` is expressed in J/µm^{3}K .

Materials declared here but not used in the following sections (channel, dies or stack) will be reported with a warning messages (`stderr`).

Example

```
material SILICON :
       thermal conductivity     1.30e-4 ;
       volumetric heat capacity 1.628e-12 ;
```

Dies

A die is a group of layers stacked together to form a single entity that is used when declaring the sequence of stacked elements of the 3D IC later in the file. This can represent an actual IC die in the stack. You can declare multiple dies, and use a single die multiple times during the stack description. The Dies section is a mandatory section and must contain at least one die element. A die must contain one *source* layer (the term *source* layer is used to denote those layers of the stack which contain active electronic components, and hence, provide the heat source for the simulation) and zero or more passive layers. The source layer can be placed at any location in the stack of layers in a die.

```
die DIE_ID :

       [ layer  IVALUE MATERIAL_ID ; ]*

         source IVALUE MATERIAL_ID ;

       [ layer  IVALUE MATERIAL_ID ; ]*
```

where

- `DIE_ID` is the unique identifier used to refer to the declared die;
- `IVALUE` is the height of the layer (in µm);
- `MATERIAL_ID` is the (previously declared) identifier of the material composing the layer.

The order of the layers within the die reflects their vertical disposition in the 3D IC, i.e., the first layer declared is the top most layer in the die (closer to the ambient) while the last one is the one at the bottom (closer to the PCB). Two examples of die declaration and their illustrations (not to scale) are shown below.

Example

```
die TOP_IC:
       source 2 SILICON;
       layer 50 SILICON;
```

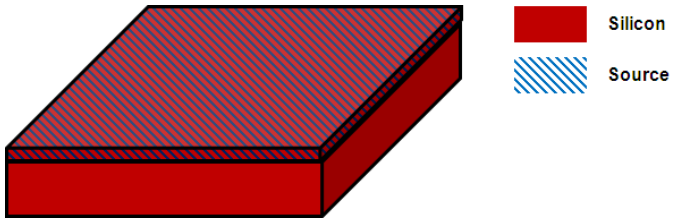


Figure 3: A 2-layer die

```
die BOTTOM_IC:
       layer 10 BEOL;
       source 2 SILICON;
       layer 50 SILICON;
```

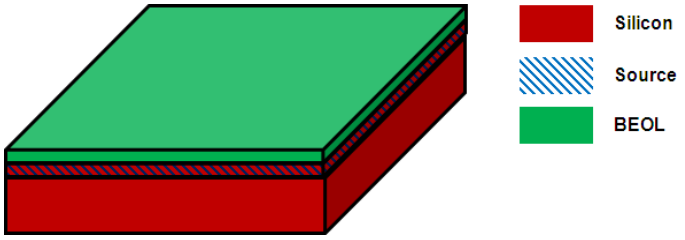


Figure 4: A 2-layer die

Conventional Air-Cooled Heat Sink

This is an optional section which includes a conventional air-cooled heat sink in the 3D IC. All the faces of the 3D IC stack are modeled as adiabatic walls by default. When the Conventional Heat Sink is specified, the top surface of the stack is connected to the ambient via a thermal resistance.

```
conventional heat sink :

       heat transfer coefficient DVALUE ;

       ambient temperature DVALUE ;
```

where

- `heat transfer coefficient` of the heat sink is expressed in W/µm^{2}K;
- `ambient temperature` is the ambient temperature expressed in K.

Example

```
conventional heat sink:
       heat transfer coefficient 1.0e-7;
       ambient temperature 300;
```

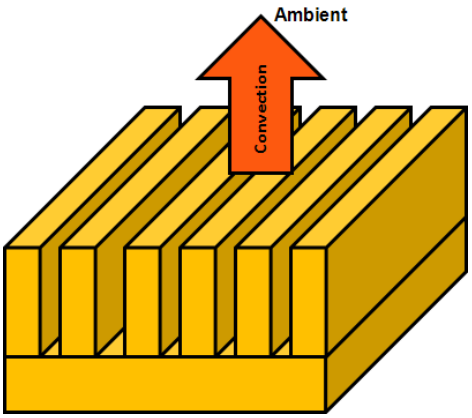


Figure 5: Air-cooled heat sink

Channel

The Channel section provides 3D-ICE information about the microchannel heat sink. Since in a given 3D IC design, all microchannel cavities are mostly designed identically, only one declaration of channel is allowed. As mentioned earlier, the channels are always laid out in the SOUTH-NORTH or *+y* direction. This section can be omitted for simulating a 3D IC without liquid cooling (solid only).

```
channel :

        height IVALUE ;

        channel length IVALUE ;

        wall    length IVALUE ;

        [ first wall length IVALUE ; ]?

        [ last  wall length IVALUE ; ]?

        wall material MATERIAL_ID  ;

        coolant flow rate DVALUE ;

        coolant heat transfer coefficient [ DVALUE |  side DVALUE ,
                                                      top  DVALUE ,
                                                      bottom DVALUE ] ;

        coolant volumetric heat capacity DVALUE ;

        coolant incoming temperature DVALUE ;
```

where

- `height` (in µm) corresponds to the height of the channel layer. This must exactly correspond to the microchannel height because of the CTTM modeling requirements. Any solid walls bounding

the top and bottom faces of the microchannel would constitute new layers that should be declared separately.

- `channel length` and `wall length` are the cross sectional widths (in µm) of channel and the walls separating them respectively. During the discretization of the system (see the Dimensions section for more details), 3D-ICE automatically starts with a wall at the eastern most end of the layer, and alternate channels and walls along the *x* direction. But the user must ensure that dimensions are such that the layer always ends with a wall (in other words, the number of columns must always be an odd number).
- `first wall length` and `last wall length` (in µm) are optional properties that represents the length of the western-most (the first column) and eastern-most (the last column) walls. This option has been included since, during the fabrication of microchannels on the back of the substrate, although the etching mask pattern is predominantly regular everywhere, there are chances of irregularities at the ends, or the deliberate use of different dimensions for the first and the last wall to preserve uniformity and symmetry of heat transfer coefficient. If one of these two dimensions (or both) is not declared, then the `wall length` will be used at its corresponding location.
- `MATERIAL_ID` is the identifier of the material composing the walls (the ID must be previously declared in the materials section).
- `coolant flow rate` is expressed (in ml/min) and it refers to the volume of coolant flowing per unit time per channel layer (cavity) in the stack. If you have multiple layers of microchannels, the total flow rate must be divided by the number of channel layers and given as a single input.
- `coolant heat transfer coefficient` is the Heat Transfer Coefficient of convective heat removal from the walls into the coolant (in $W/\mu m^2K$). It is possible to specify a single value of HTC for all the wetted surfaces of the microchannel or specify three different values- one of the *side* wall surfaces, one for the *top* and one for the *bottom* wall surface of the microchannel.
- `coolant volumetric heat capacity` is expressed in $J/\mu m^3K$.
- `coolant incoming temperature` is the inlet coolant temperature expressed in K .

Example

```
channel:
       height 100 ;
       channel length 50;
       wall    length 50;
       first wall length 25;
       last  wall length 25;
       wall material SILICON;
       coolant flow rate 42;
       coolant heat transfer coefficient
                          side 2.7132e-8,
                           top 5.7132e-8,
                        bottom 4.7132e-8;
       coolant volumetric heat capacity
                              4.172e-12;
       coolant incoming temperature
                                    300;
```

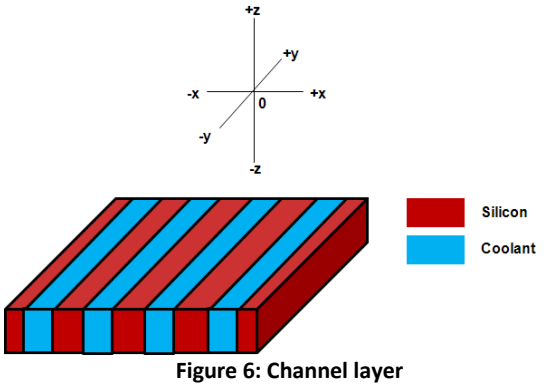


Figure 6: Channel layer

If neither the conventional nor the microchannel heat sink is declared in the Stack Description File, then the parsing will end with a warning message. Note that this would cause the temperatures to blow up unbounded with time in the presence of non-zero heat sources.

Stack

This section builds the vertical structure of the stack. The stack is composed of Dies (as previously declared) and layers and/or channels.

```
stack :

        [ layer   LL_ID  DVALUE MATERIAL_ID ; ]*

        [ channel CC_ID ; ]*

        [ die     DD_ID  DIE_ID floorplan "PATH" ; ]+
```

where

- `LL_ID`, `CC_ID` and `DD_ID` are identifiers used to name the stack elements and they can be used in the simulator code to refer to the corresponding element. They must be unique for each element.
- `MATERIAL_ID` is the identifier of the material (as previously declared) composing the declared layer.
- `DVALUE` is its height of the layer (in µm).
- `DIE_ID` is the identifier of a die (as previously declared) and `PATH` is the path to the floorplan file. This floorplan will be placed on the declared source layer in the definition of the die. The floorplan files contain information of the location and power dissipation activity of various floorplan components for the given die (see the description of Floorplan Files for more details). The same `DIE_ID` can be used multiple times (with different identifiers `DD_ID`) in a stack with the same or different floorplans, if identical/similar dies exist in a single IC.

The above grammar for the stack section is not the representative of a typical stack description and is only given for simplicity. Just remember that your final stack sequence must satisfy the following:

- there must be at least one die
- it cannot begin or finish with a channel (i.e., microchannel cavities can't be the bottommost or the topmost layers in a stack)
- there cannot be two consecutive channels
- channels can be used only if previously declared
- layers are optional

Declaring a layer in a stack is not mandatory but we left this option to support stacks with irregular patterns of dies and channels or to build auxiliary layers (such as a bonding layer).

As in the case of defining dies, the final sequence of stack elements reflects their vertical disposition. The first stack element corresponds to the topmost element while the last element declared is closest to the PCB.

Example

```
material SILICON :
       thermal conductivity     1.30e-4;
       volumetric heat capacity 1.628e-12;
material BEOL :
       thermal conductivity     2.25e-6;
       volumetric heat capacity 2.175e-12;

conventional heat sink:
       heat transfer coefficient 1.0e-7;
       ambient temperature 300;

channel:
       height 100 ;
       channel length 50;
       wall    length 50;
       first wall length 25;
       last  wall length 25;
       wall material SILICON;
       coolant flow rate 420;
       coolant heat transfer coefficient 5.713e-8;
       coolant volumetric heat capacity 4.172e-12;
       coolant incoming temperature 300;

die TOP_IC:
       source 2 SILICON;
       layer 50 SILICON;
die BOTTOM_IC:
       layer 10 BEOL;
       source 2 SILICON;
       layer 50 SILICON;


stack:
       die     MEMORY_DIE     TOP_IC    floorplan "./mem.flp";
       channel TOP_CHANNEL;
       die     CORE_DIE       BOTTOM_IC floorplan "./core.flp";
       channel BOTTOM_CHANNEL;
       layer   BOTTOM_MOST    10        BEOL;
```

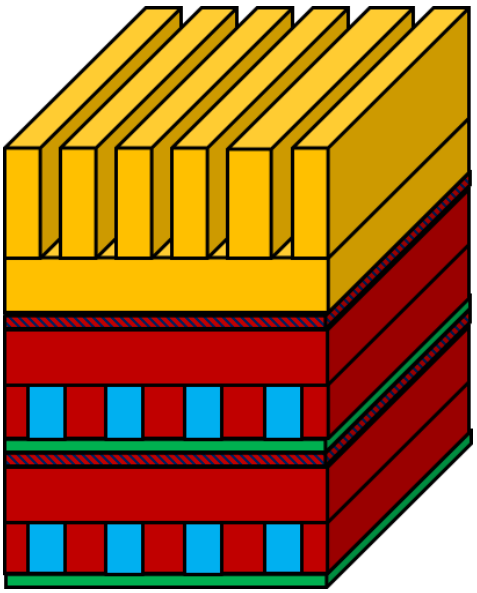


Figure 7: Complete stack

Dimensions

The last section of the Stack Description File declares the *xy* dimensions of the entire chip and the discretization sizes for the thermal cells (all in µm).

```
       dimensions :

              chip length DVALUE , width DVALUE ;

              cell length DVALUE , width DVALUE ;
```

The entire chip is discretized based on the same cell length (along *x* direction) and width (along *y* direction) values. The discretization along the *z* direction is not specified, since the height of a thermal

cell is taken to be the same as the height of the layer in which it exists as shown for a layer in Fig. 8. However, if you want a finer discretization than that along the *z* direction, then you will have to split the layer into multiple layers of the same material stacked on the top of each other in the declaration of die/stack, as shown in Fig. 9 (here $h_1+h_2=h$).

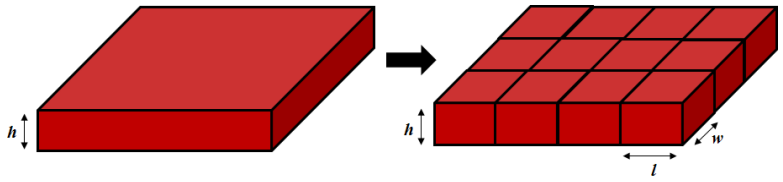


Figure 8: Discretization of a single layer

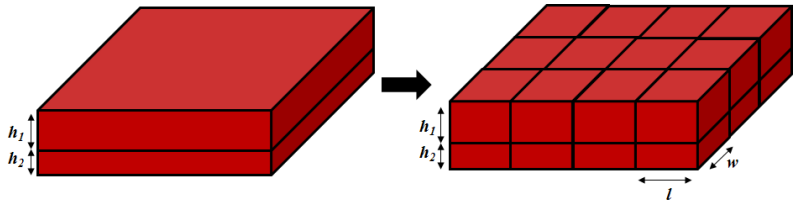


Figure 9: Discretization of a single layer split into 2 layers

Example

```
dimension :
        chip length 10000, width 10000;
        cell length 50, width 50;
```

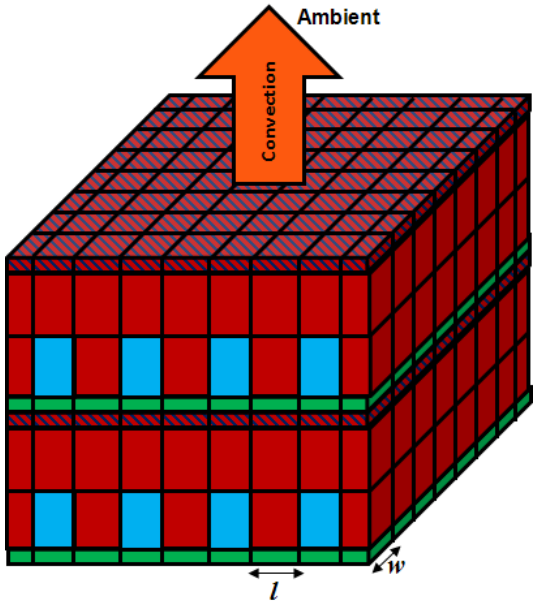


Figure 10: Discretized computational domain for the stack shown in Fig. 7

In the presence of channels, the cell length is determined SOLELY based on the cross sectional dimensions of channel and wall in the Channel section of the Stack Description File. This is because the CTTM modeling used in 3D-ICE requires that the entire cross section of the microchannel be a part of the thermal cell. This means that the cell length at some position along the *x* direction in the model is dependent upon the channel or the wall cross sectional width at that point. Note that given different values of `channel length`, `wall length`, `first wall length` and `last wall length`, the discretization along this direction will be non-uniform in the simulator.

For the purpose of illustration, the discretized domain corresponding to the stacked structure built in Fig. 7 is shown in Fig. 10. Here, *l* may vary from one cell to the other while *w* is a constant specified by `cell width` above. The `cell length` MUST STILL BE DECLARED in the presence of channel layers, in spite of the fact that it will be ignored during the parsing.

Problem Complexity

Remember that the dimensions of the chip, together with the dimensions of the thermal cells will directly affect the performance of the simulation. Indeed, the number of cells (nodes) influences both the amount of memory used by the library and the time needed to solve the linear system (see "3D-ICE_ICCAD2010.pdf"). The main computational effort of the simulator is incurred during the execution of SuperLU and `blas` libriries. Specifically, the LU factorization of the system matrix is the most time/memory intensive. Hence, for large problem sizes, the availability of memory must be ensured to prevent this step from failing during the simulation.

C. Floorplan File

Every die in the stack must be related to a "Floorplan File" (*.flp), which essentially provides the power dissipation profile (or heat sources) for the simulation. Each Floorplan file must contain the list of functional blocks (cores, caches, memories, etc), their positions, and the power dissipation as a function of time.

Every functional block, here called *floorplan element*, is a rectangular area inside the die, laid out in the source layer. Each floorplan element has a unique identifier- the name it is assigned. In addition, the position and the dimensions of each floorplan element are given (in µm) based on the same Cartesian coordinates that was used for building the stack, with the origin at the SOUTH-WEST corner of the source layer. An example floorplan of a 1cmX1cm die with the reference coordinates is shown in Fig. 11. All the distances shown here are in µm.

A floorplan element in the Floorplan File is declared using the following syntax.

```
IDENTIFIER :

        position  DVALUE , DVALUE ;

        dimension DVALUE , DVALUE ;

        power values DVALUE [ , DVALUE ]* ;
```

where

- `IDENTIFIER` is the unique identifier used to name the floorplan element. This string must be unique within the floorplan file it belongs to but it can be used on a different file.
- `position`, expressed in (in µm), is the (x,y) coordinate of the SOUTH-WEST corner of the floorplan element.
- `dimension` is the (length, width) dimensions of the floorplan element (in µm).
- The `DVALUE`(s) against the keyword `power values` are the list of power dissipation values (expressed in W) of the floorplan element for each *time slot* (scroll down for the explanation of time slots in 3D-ICE) separated by commas.

As an example, the declaration for the Cross Bar in Fig. 11 with 5 time slots would be as follows:

Example

```
Cross_Bar :
        position 2500, 3500;
        dimension 5000, 3000;
        power values 0.3, 0.3, 0.4, 0.2, 0.3;
```

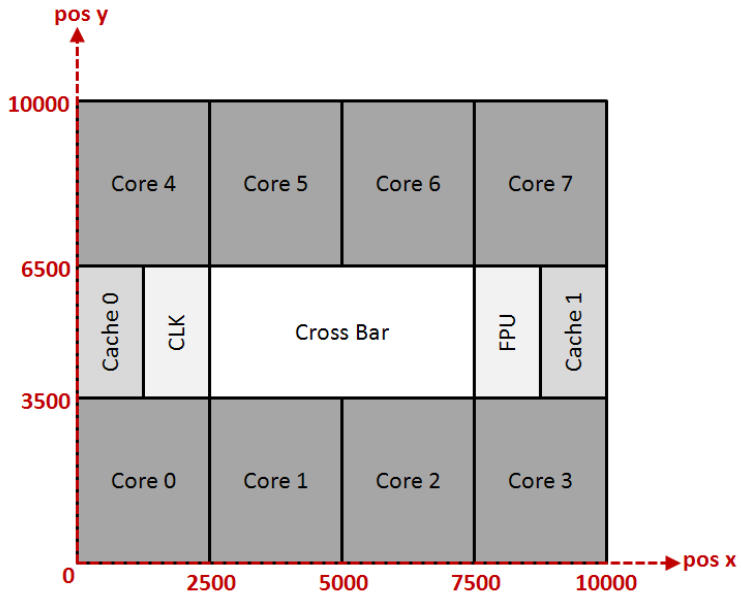


Figure 11: An example floorplan for a 1cmX1cm IC die

The following points must be kept in mind while writing the floorplan files:

- Floorplan elements must not overlap. During the parsing of the Floorplan File, the values describing the elements are checked to verify that all the elements are inside the chip and that they do not overlap.
- When the stack structure is discretized based on the given thermal cell dimensions, the power dissipated by a floorplan element is uniformly divided among the thermal cells contained *within*

its borders. A thermal cell is considered to be *within the borders* of a given floorplan element if the CENTER of the thermal cell is inside the floorplan element. If the center of a thermal cell happens to be right on the border of 2 or more floorplan elements, then it is thrown into the floorplan on the NORTH/EAST by default. The distribution of thermal cells into different floorplan elements is illustrated in Fig. 12: the thermal cells highlighted in blue belong to the Cross Bar, the cells in green go to Core 1 and the cells in orange belong to Core 0. If you want to see exactly which thermal cells are covered by each floorplan element, you can use the `print_floorplan` or `print_all_floorplans` functions in the Library to print the range of rows (*y* indices of thermal cells) and columns (*x* indices of the thermal cells) covered by each floorplan element, after the input files have been parsed. For more details on these routines, please see Chapter 5.

- The same Floorplan File can be assigned to 2 or more dies in the Stack Description File if they happen to have identical structure and behavior in the design. Each floorplan element in the entire design, in that case, is uniquely identified by the `DD_ID` identifier of the corresponding die element and the `IDENTIFIER` of the floorplan element.
- If two dies in the stack have the same floorplan but during the simulation they have a different power dissipation activity, then 2 different Floorplan Files must be created for each die and assigned to the corresponding die element declarations in the Stack Description File. This is because the power dissipations are directly linked to each floorplan element in the Floorplan File.
- It is possible to have gaps in the floorplan- regions where there is no floorplan element and hence, no power dissipation. The thermal cells in these regions will simply not be assigned any source value during the solving of system equations.

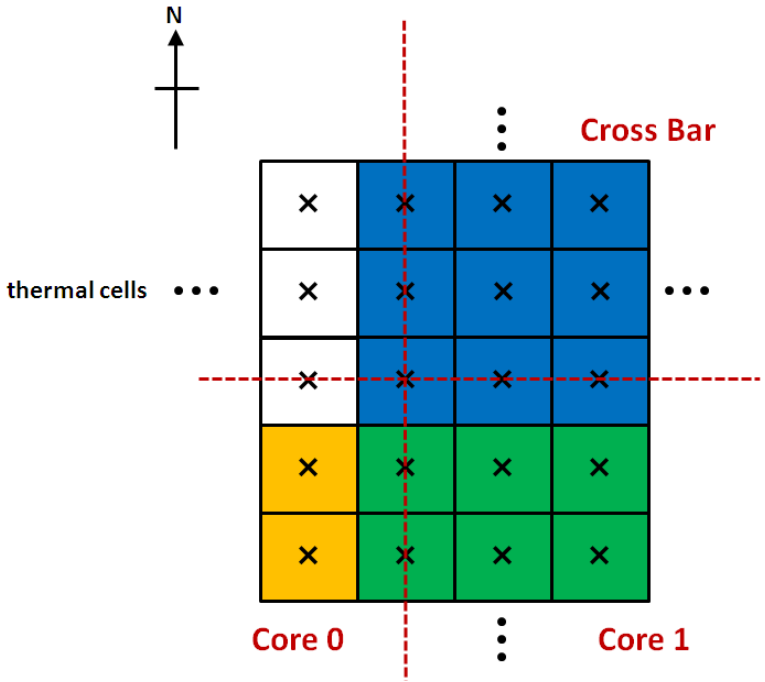


Figure 12: Distribution of thermal cells vis-à-vis floorplan elements

Time Slots

The entire time-interval of simulation (ToS) in 3D-ICE is divided into *time slots*- the minimum time duration for which the switching activity of the floorplan elements has been resolved. For example, if for a given design the switching activity (a measure of how much a floorplan component is active, directly related to its power dissipation) is sampled every 200ms during a 1 second ToS, then there are 5 time slots for the 3D-ICE simulation. And hence, there must be 5 values of power dissipation for each floorplan element declaration in the Floorplan File. Conversely, the number of power values for the floorplan element is interpreted as the number of time slots (NoTS) for the simulation. In all 3D-ICE simulations, it is assumed that the power dissipation in a particular thermal cell is CONSTANT during the period of a time slot- calculated based on the power value of the corresponding floorplan element, in which the thermal cell exists, for that time slot (see Fig. 13).

Note that time slot is different from *time step*, which is the discretization length of the time-domain for the numerical integration of the system of differential equations representing the entire thermal circuit created inside 3D-ICE. The exact durations of both time slot and time step are given when running the simulator. For further details, see Chapter 5.

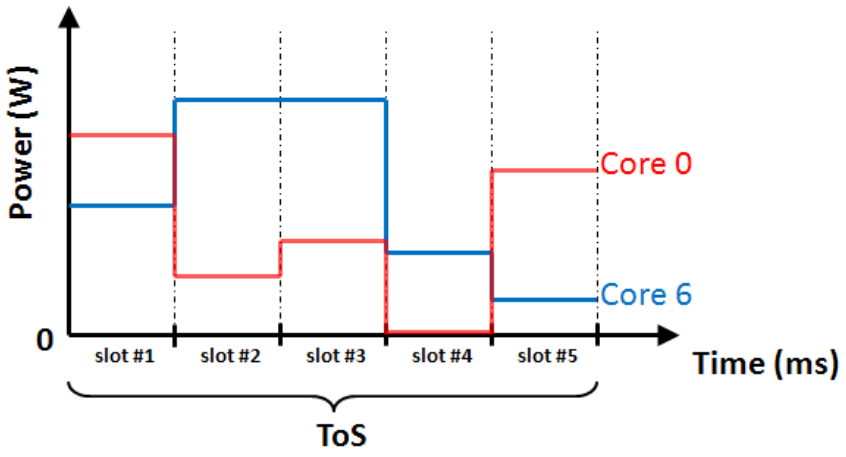


Figure 13: Power dissipation profile of Core 0 and Core 6 in Fig. 11

Since uniform sampling of the switching activity for all the components in an IC is assumed, the number of power values given for each floorplan element in the Floorplan File MUST BE THE SAME. Even when there is zero power dissipation for a particular floorplan element during some time slot (Core 0 at slot #4 in Fig. 13) or if a particular floorplan element has constant power dissipation during some or all time slots (Core 6 during slot #2 and slot #3 in Fig. 13), the corresponding power values must be mentioned EXPLICITY for each time slot. The function `fill_stack_description` in 3D-ICE parses the Stack Description File first and then parses all the Floorplan File(s) starting from the bottom most die in the stack. Therefore, the number of power values (and hence NoTS) declared in the first floorplan element in that die will be used as a reference to check all the others elements and also, will eventually determine the ToS. Power values in excess of this reference number of slots will be discarded and reported with a warning message, while the lack of them will be considered as an error.

5. Running 3D-ICE

In the `./examples` folder, a simulation project `Emulate3DICe.c` has been created as an example to show hot to use the 3D-ICE library. It has been written to parse and analyze the Stack Description File `2dies1channel.stk` and the corresponding Floorplan Files placed in the same folder. Once compiled with the make command, run the simulation using the following command.

```
$./Emulate3DICe 2dies1channel.stk time_slot_DVALUE delta_DVALUE
```

where

- `2dies1channel.stk` is the path to the Stack Description File containing the description of the 3D-IC.
- `time_slot_DVALUE` is the duration of each time slot (in seconds) for which power values specified in the Floorplan File(s) are held constant. This value, multiplied by NoTS (from the Floorplan File) gives the ToS.
- `delta_DVALUE` is the time step value (in seconds) for the numerical integration of the system equations. This must be LESS THAN or equal to `time_slot_DVALUE`, if you want the simulator to resolve the power switching accurately.

3D-ICE uses backward Euler method with constant time-stepping to solve the system equations. Hence, the solution is always numerically unconditionally stable. However, accuracy can be increased by reducing the time step value. The local truncation error of backward Euler method behaves as $\boldsymbol{O(h^2)}$, where h is the time step. However, given a ToS, the number of time steps in the entire simulation is a $\boldsymbol{O(1/h)}$ function. Hence, the upper bound of the total accumulated error at the final time point in the simulation behaves as

$$\boldsymbol{O(h^2).\ O(1/h)=O(h)}.$$

In other words, the total final error is approximately a linear function of the time step. For RC circuits, such as the thermal circuit that 3D-ICE solves, it is common practice to have at least 5 time steps for the duration of a rise time of the output temperature (defined as the time duration for the rise of temperature from 10% to 90% of its steady state value) to resolve the transients accurately.

A. Essential components in a Project

Every simulation project built using 3D-ICE can be prepared following three steps.

Step 1: First, the problem must be initialized and the necessary data structures containing the 3D IC information must be filled. For this, the two main header files are to be included in the main file to access the homonym data structures.

- `stack_description.h`
- `thermal_data.h`

For details about their content please refer to the description given in these files.

StackDescription

This data structure type collects all the data pertaining to the 3D IC structure and floorplans. An instance of this data type will then be related to the Stack Description File. The functions available to initialize, fill and empty an instance of a `StackDescription` variable are:

- `init_stack_description (StackDescription*)`
- `fill_stack_description ( StackDescription*, String_t)`
- `free_stack_description (StackDescription*)`

The functions to access the information in this data structure are

- `print_stack_description (FILE*, String, StackDescription*)`
- `print_all_floorplans (StackDescription*)`
- `get_number_of_remaining_power_values (StackDescription*)`
- `get_number_of_floorplan_elements (StackDescription*, String_t)`
- `get_number_of_channel_outlets (StackDescription*)`

ThermalData

This data structure type collects all the data needed for the thermal simulation, such as the values of conductances in each cell, matrices representing the system equations, etc. The functions to initialize, fill and empty a `ThermalData` variable are

- `init_thermal_data (ThermalData*, Temperature_t, Time_t, Time_t)`
- `fill_thermal_data ( ThermalData*, StackDescription*)`
- `free_thermal_data (ThermalData*)`

Just as an instance of `StackDescripiton` is tied to a Stack Description File, an instance of `ThermalData` depends upon an instance of `StackDescription`. Hence it is necessary to initialize and fill a `StackDescription` variable before filling a `ThermalData` variable. And for the rest of the simulation project, the two variables must be used in tandem since they refer to the same problem.

Step 2: Once the data structures have been filled, the next step is to execute the simulation. It can be done using two different functions.

- `emulate_step (ThermalData*, StackDescription*)`
- `emulate_slot (ThermalData*, StackDescription*)`

`emulate_step` increments the simulation time by a single time step and then terminates; while emulate_slots advances the simulation by a time slot, by simulating the system for all the time steps contained in that time slot (remember that time slot is the duration for which source power values are kept constant), when the power values indicating the activities of the floorplan elements will be updated. Both the functions can be called iteratively in a loop until the end of the ToS, controlled using the return variables of these functions which indicating whether or not the simulation has reached certain epochs (see the description in the files containing these functions for more details).

Step 3: Finally, outputs must be read from the thermal analysis during, and at the end of the simulation. The following last set of functions has been written to access to the time/temperatures during the simulation (refer to the example project for more details on the usage).

- `get_current_time (ThermalData*)`
- `get_max_temperature_of_floorplan_element ( … )`
- `get_min_temperature_of_floorplan_element ( … )`
- `get_avg_temperature_of_floorplan_element ( … )`
- `get_min_avg_max_temperature_of_floorplan_element ( … )`
- `get_all_max_temperatures_of_floorplan ( … )`
- `get_all_min_temperatures_of_floorplan ( … )`
- `get_all_avg_temperatures_of_floorplan ( … )`
- `get_all_min_avg_max_temperatures_of_floorplan ( … )`
- `get_temperature_of_channel_outlet ( … )`
- `get_all_temperatures_of_channel_outlets`
- `get_cell_temperature ( … )`
- `print_thermal_map ( … )`

These functions make it possible to read to the thermal state of a single floorplan element, an entire floorplan, a single channel outlet or all the coolant outlets in a given channel layer at any time during the simulation. It is also possible to read the temperature of a single thermal cell or print directly the thermal map (a matrix) for a specific layer in the stack. Some of these functions require you to refer to flooplan elements, layers etc. using the corresponding identifiers declared in the Stack Description File.

The above steps are essentially the same for any 3D ICE thermal simulation project. To simulate a new problem you can reuse the example file `Emulate3DICe.c` by doing the following:

- change the Stack Description File and the corresponding Floorplan Files,
- change the solver controls in your main file to suit your requirements
- change the output printing functions according to the new stack/floorplan structure.

B. Debugging of Simulation

For the purpose of debugging, several pre-processing options can be enabled to directly check the values computed during the construction of thermal data (thermal grid/circuit, system matrices, sources etc) before the simulation even starts. These options can be activated uncommenting the corresponding line in the file `3D-ICE/sources/Makefile` and running the `make` command again (run the `make clean` command before building). As a consequence, messages will be redirected to `stderr`.

The `FillThermalData` executable in the `3D-ICE/examples` folder has been created to illustrate this debug process. Since each of the following debug option prints a line for every thermal cell in the grid, it is suggested that you redirect at the command line the `stderr` stream to a specific file and observe it at the end of the execution. `FillThermalData` is run in the same way as `Emulate3DICe`.

```
$./FillThermalData "mystack.stk" time_slot_DVALUE delta_DVALUE
```

The debug options that are available are:

- `PRINT_CONDUCTANCES` prints for every cell, the *cell ID* -defined as (layer, row, column, index)-, its dimensions and the 6 conductances (NORTH, SOUTH, EAST, WEST, TOP and BOTTOM). The ID of the material related of the cell and the c_{conv} constant (in case of liquid cells, see "3D-ICE_ICCAD2010.pdf" more details) are also printed at the beginning of a new layer of cells.
- `PRINT_CAPACITIES` prints for every cell, the cell ID, its dimensions and its (heat capacity/time step) value, which is used in the construction of system matrices for backward Euler numerical integration. Also, the parameters used to evaluate it (`volumetric heat capacity` and `delta_DVALUE`) are printed. The ID of the material of the cell is also printed at the beginning of a new layer of cells.
- `PRINT_SYSTEM_MATRIX` prints the content of the system matrix while it is filled. For every column in the system matrix (***G***+***C***/*h*, see "3D-ICE_ICCAD2010.pdf" for more details), it prints the cell ID of the corresponding cell and the list of the nonzero coefficients indicating representing its neighbors. For every neighbor, it also prints the cell ID.
- `PRINT_SOURCES` prints for every cell in a source layer, the cell ID, its dimensions, the source value for that cell, and also, the values used to evaluate that source value (floorplan `power values`, surface fraction of the floorplan element occupied by the cell and floorplan element `IDENTIFIER`). For every channel inlet cell (the southernmost cell along a channel), it prints the cell ID and the source value corresponding to the boundary condition, and also the parameters used to evaluate that source (`coolant incoming temperature` and c_{conv} constant). Remember that this debugging information will be printed every time the power values of the floorplan elements are read at the beginning of a time slot.