ICT-2009.3.2-248603-**IP**

Modelling, Control and Management of Thermal Effects in Circuits of the Future

	WP no.	Deliverable no.	Lead participant
	WP6	**D6.3.3**	**OFFIS**

Report on integration of the individual optimization techniques

Prepared by	**Patrick Knocke (OFFIS)**
Issued by	**THERMINATOR Project Office**
Document Number	**THERMINATOR/D6.3.3/v1.3**
Dissemination Level	**Public**
Date	**2013-02-04**

Document

Title	Report on integration of the individual optimization techniques
Type	Deliverable PU
Ref	D6.3.3
Target version	V1_3
Current issue	V1_3
Status	
File	
Author(s)	Patrick Knocke (OFFIS) Sven Rosinger (OFFIS)
Reviewer(s)	Michele Carrano (ST) <michele.carrano@st.com> Domenik Helms (OFFIS) <domenik.helms@offis.de>
Approver(s)	G.Gangemi(ST)
Approval date	04/02/2012
Release date	06/02/2012

Distribution of the release

Dissemination level	PU
Distribution list	

History

Rev.	DATE	Comment
0.1	2012-11-15	Initial version
0.9	2012-12-12	Input of IMEC and OFFIS
1.0	2013-01-21	Revised version, approved by all partners
1.1	2013-01-28	Input of ST and UNIBO
1.2	2013-02-04	Update including changes from reviewers
1.3	2013-02-04	Check and ship out

References

[1] **Therminator Project Partners** *D6.3.1 - Specification and standardization of the thermal aware design, optimization and exploration flow*, 2011

[2] **Therminator Project Partners** *D6.3.2 - Presentation and evaluation of thermal-aware design techniques for 3D SiP stacks and 2D SoCs*, 2012

[3] **Therminator Project Partners** *D6.2.2 - Presentation and evaluation of an all level thermal simulator of 3D SiP stacks and 2D SoCs*, 2012

[4] **Therminator Project Partners** *D1.2.1 - Specification of internal design flows and environments and existing tool interfaces*, 2010

[5] **Therminator Project Partners** *D1.3.1 B - Technical specification of testcases and distribution to partners of concern*, 2012

[6] **H. H. Chan and I. L. Markov** *CompaSS (Compacting Soft and Slicing Packings),* http://vlsicad.eecs.umich.edu/BK/CompaSS/

[7] **H. H. Chan and I. L. Markov** *Practical Slicing and Non-slicing Block-Packing without Simulated Annealing*, Proc. Great Lakes Symposium on VLSI (GLSVLSI), Boston, Massachusetts, April 2004, pp. 282-287.

[8] **Sven Rosinger, Malte Metzdorf, Domenik Helms, Wolfgang Nebel** *Behavioral-Level Thermal- and Aging-Estimation Flow*, Proc. of 12th IEEE Latin American Test Workshop (LATW), 2011

[9] **Reef Eilers, Malte Metzdorf, Sven Rosinger, Domenik Helms, Wolfgang Nebel** *Phase space based NBTI model*, Proc. of International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2012

[10] **Therminator Project Partners** *D7.3.1 - Evaluation of thermal-aware design prototype tools*, 2012

This page was intentionally left blank.

Contents

General remarks

The presented work was executed in the context of Task 6.3, System-level thermal aware design, which started at month M1 and will end in month M36. This document covers the activities in period M31-M36 of all involved partners.

Throughout the document direct links are given to the previously defined measurable objectives:

- MO6.3.1 - Development of optimization algorithms for thermal, ageing and performance behaviour of 3D SiPs and 2D SoCs
- MO6.3.2 - Evaluation and calibration of newly created/extended tools and flows regarding thermal, ageing and performance optimization potential

1 Introduction

Task 6.3 is focused on the set up and implementation of a thermal aware design flow for analysis of temperature distributions and automated place and routing for 3D SiP and 2D SoCs. Each partner developed different types of optimization techniques unique to their own expertise. IMEC has investigated the optimization of 3D stack of Dies in SiPs by optimizing placing, routing and generation/positioning of TSVs. OFFIS used the developed high-level NBTI-aware thermal estimation flow to generate optimizing place and route constraints, while Unibo/ST optimized the floorplanning by examining at the placement of 3D system macro-blocks.

The general requirements and outlay for the design flow were described earlier in D6.3.1 [1], while the actual implementation and validation of the different optimization techniques was presented in D6.3.2 [2]. The current deliverable therefore shows the integration of the developed techniques/tools and their interfaces.

Section 2 focuses on the APIs used in the NBTI-aware Thermal Estimation flow developed by OFFIS, while section **Error! Reference source not found.** presents a brief overview of the design methodology and the tools that can be used for a holistic 2.5/3D IC implementation done by IMEC. Section 4 addresses the integration of the physical-level thermal simulation and optimization flow by ST and UNIBO.

The evaluation summary of the different implementations is presented in Section 5 and includes links and quantifications to the measurable objectives.

This report ends with the conclusion in Section 6.

2 NBTI-aware Thermal Optimization Flow Integration

OFFIS has integrated the proposed optimization techniques into a flow consisting of multiple tools and frameworks. As specified in [2] the flow consists of a synthesis tool (PowerOpt) to generate RT-Level descriptions of a Behavioral Level Design, an inhouse Place & Route solution to arrange the generated hardware on the die surface and a multi physics simulation framework to estimate the resulting temperature and aging behaviour taking into account the so called NBTI effect.

But the core program of the flow is the optimization tool called “Analyzer”. This binary generates all inputs and evaluates all outputs of the aforementioned other tools used in the flow and activates and controls them to implement the interdependency of the different phenomena and optimize the resulting design to decrease temperature spikes and therefore increase the reliability and lifetime of the device.

Figure 1 shows the possible information flow between the different tools. As it can be seen there are three specially marked interfaces which will be discussed in detail in the following subsections. The interface between the synthesis tool and the optimization tool is discussed in Section 2.1 Synthesis API, Section 2.2 Place and route API specifies the data formats and parameters used to control the floorplanning tool, while Section 2.3 Physics API will describe the same for the multi physics simulation framework. All examples in the following sections are based on test-case 5 specified in [5] of the Therminator project.

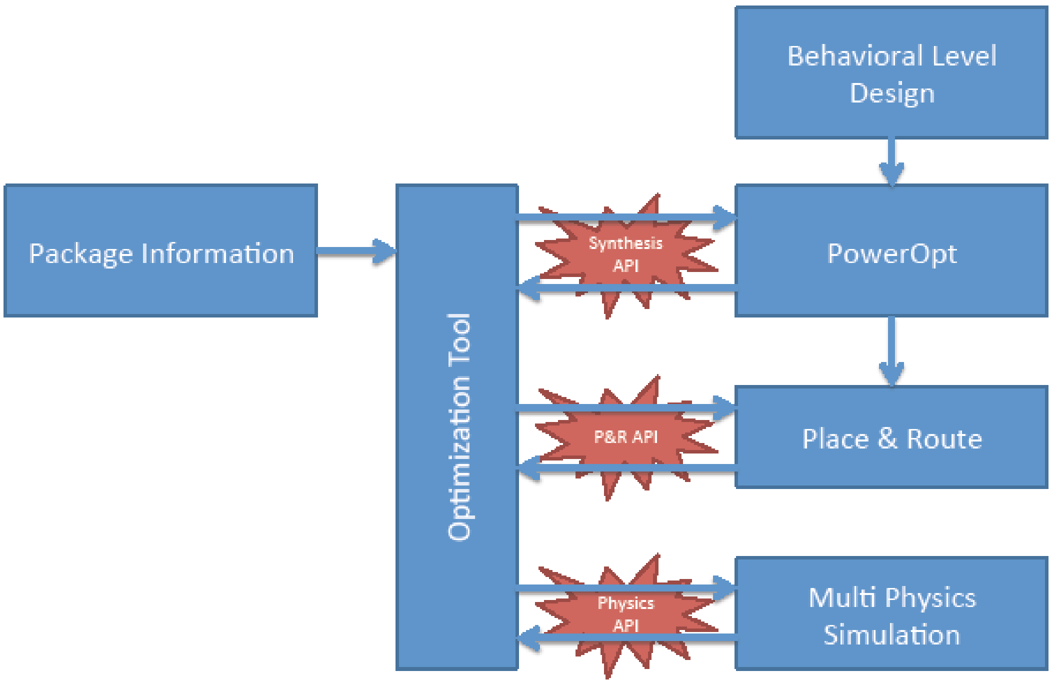


Figure 1 – Used interfaces and information flow in the high level thermal and reliability optimization flow

2.1 Synthesis API

The synthesis tool used in the implementation of the optimization flow is the former ChipVision owned software PowerOpt [4]. As ChipVision went bankrupt in the course of the project no changes to the tool could be made to integrate the optimization techniques. OFFIS decided to use the already available API of PowerOpt to modify the software’s behaviour by settings constraints on the synthesized design. This actually allows for a better encapsulation of the different tools and therefore increases the replace ability of them.

The following command is used in the flow to control and modify the synthesized design.

```
set_process ?-help? -processes... ?-voltage? ?-csteps_min? ?-csteps_max?
            ?-type? ?-min_resources? ?-max_resources?
            ?-sharing_threshold? ?-synthesize? ?-chaining?
            ?-loop_unrolling_max_nodes=-lumax? ?-fsm_encoding?
            ?-cdb_mapping?

Description:
This command can be used to set process constraints.

Keywords:
  -processes:     A list of unique process identifiers for all
                  processes. Available processes can be obtained via
                  calling 'get_process_list'. If you want a complete
                  hierarchy level of processes to be included you may
                  also submit the hierarchy name only.
  -voltage:       Sets the voltage [V] for the given processes.
                  Deprecated! Use set_hiearchy -synthesis_voltage
                  instead.
  -csteps_min:    Sets the desired minimum (positive) number of
                  control-states (csteps) in the schedule. The value
                  'auto' indicates no limit / preserve previously
                  specified pipelining
  -csteps_max:    Sets the desired maximum (positive) number of
                  control-states to use in the schedule. The value
                  'auto' indicates no limit / preserve previously
                  specified pipelining
  -type:          The name of a CDB component. E.g. Add_fast.
  -min_resources: Sets the minimum number of usable resources of the
                  given type for the given processes.
  -max_resources: Sets the maximum number of usable resources of the
                  given type for the given processes.
  -sharing_threshold:
                  Sets minimum input bitwidth a resource of given
                  type must have to be considered for sharing
  -synthesize:    Marks a process instance or hierarchy level as to
                  be synthesized. Usually this option is applied to
                  the top-level instance. Alternatively, you can
                  apply the C source code pragma 'poweropt toplevel'
                  to mark those parts of the source code which
                  belongs to the design to be synthesized. This
                  option takes values on=1 or off=0. The default is
                  'off'.
  -chaining:      Chaining mode advanced=1 or basic=0. The default is
                  'advanced'
  -loop_unrolling_max_nodes=-lumax:
                  Defines the maximum number of nodes in a process
                  instance after loop unrolling. Exceeding this limit
                  will prevent unrolling.
  -fsm_encoding:  Optimize the State Transition Graph in FSM for low
                  power. By default the STG encoding would be the one
                  targeted for low power. Possible values are
                  'one_hot' for One Hot Encoding, 'one_cold' for One
                  Cold Encoding, 'bin' for Binary Encoding, 'gray'
                  for Gray Encoding and 'low_power' for Low Power
                  State Transition Graph Encoding.
  -cdb_mapping:   Can be 'area' or 'timing'.
```

Output 1 – `set_process -help` information

The `set_process` command can be used to set global values like design wide supply voltage or scheduling constraints but can be also used to set much more fine granular settings. The optimizations presented in [2] need to control the number of used components to distribute the activity on the die surface and eliminate temperature hotspots. Using the `processes`, `type` and `min_resources` keywords it is possible to increase the number of a specific component on a per process instance level. If for example the number of small Adders in process `val_abs3` needs to be raised from two instances to three the following command can be issued to PowerOpt:

```
set_process –processes Design.process_image.val_abs3.this
–type DEFAULT::Add_small –min_recources 3
```

If there are enough operations to map on to the additional component PowerOpt will update the scheduling and binding accordingly.

To identify which components are used more than others and which temperature and aging behaviour is occurring at every component, detailed information about the used power and location of each component is necessary. As PowerOpt does not do a place and route step during synthesis only information about the used power and the area of each component is available.

To get the power of each component the command `report_power` can be used.

```
report_power ?-help? ?-precision? ?-instances? ?-resources? ?-components?
             ?-cdb_component? ?-memories? ?-noheader? ?-file?

Description:
This command prints a report of the power estimation results.

Keywords:
  -precision:     Integer value defining the precision of reported
                  numbers.
  -instances:     Report instance specific power numbers.
  -resources:     Report power numbers of individual datapath
                  resources like adders and registers.
  -components:    Report power consumption per component (like clock,
                  controller, etc.).
  -cdb_component: Reports the CDB component name of the resources.
                  This option can only be used together with option
                  '-resources'.
  -memories:      Report power per memory instance.
  -noheader:      Ommits the report header.
  -file:          If submitted, the output is stored into this file.
                  Otherwise, the report is written into the console.
```

Output 2 – `report_power -help` information

Using the parameters `resources`, `components`, `cdb_component` and `file` the following output can be obtained and saved in a file in the filesystem for later parsing in the optimization tool.

```
Resource report:

       component                  total   leakage  dynamic   CDB-component
--------------------------------------------------------------------------
Design.process_image.val_abs3.this
  Ctrl                           13.062u   2.037n  13.060u POWEROPT::Internal
  Reg_1                          16.715u  32.589n  16.682u DEFAULT::Reg
  Reg_6                           8.487u  16.295n   8.471u DEFAULT::Reg
  Reg_7                           8.415u  16.295n   8.399u DEFAULT::Reg
  Reg_5                           8.255u  16.295n   8.239u DEFAULT::Reg
  Reg_9                           7.828u  16.295n   7.811u DEFAULT::Reg
  Mux_Reg_1_data_in               7.244u  13.294n   7.231u DEFAULT::Selop_small
  Add_1                           6.966u  12.492n   6.953u DEFAULT::Add_small
  Add_2                           6.000u  10.721n   5.989u DEFAULT::Add_small
  Inc_1                           3.729u   7.535n   3.722u DEFAULT::Inc_small
  Mux_Inc_1_in00                  3.526u   4.898n   3.521u DEFAULT::Selop_small
  Neg_1                           3.514u   6.714n   3.507u DEFAULT::Neg_small
  Mux_Add_1_in01                  3.245u   4.898n   3.240u DEFAULT::Selop_small
  Mux_Reg_5_data_in               3.228u   4.898n   3.223u DEFAULT::Selop_small
  Mux_Add_1_in00                  3.026u   4.898n   3.021u DEFAULT::Selop_small
  Mux_Reg_6_data_in               2.964u   4.898n   2.959u DEFAULT::Selop_small
  Mux_Y_back_readWritePort_0.A0   1.655u   1.990n   1.653u DEFAULT::Selop_small
  Mux_Cmp_Ls_1_in01             789.688n   1.225n 788.463n DEFAULT::Selop_small
  Reg_3                         254.405n 509.204p 253.896n DEFAULT::Reg
  Reg_4                         250.271n 509.204p 249.762n DEFAULT::Reg
  Cmp_Ls_1                       62.308n 109.589p  62.199n DEFAULT::CmpLt_small
  Enc_1                           0.000    0.000    0.000  POWEROPT::Internal
  Sll_1                           0.000    0.000    0.000  DEFAULT::Sll_small
  Sll_2                           0.000    0.000    0.000  DEFAULT::Sll_small
  Sll_3                           0.000    0.000    0.000  DEFAULT::Sll_small
  Srl_1                           0.000    0.000    0.000  DEFAULT::Srl_small
```

Output 3 – Excerpt of the `report_power` output

As it can be seen the output consists of the component name, which can be merged with the process name to get the unique hierarchical name, the total power, the leakage and dynamic power and the used CDB component.

To get the location of each component on the die a place and route has to be done. In case of a commercially available place and route solution this means exporting a register transfer level description of the synthesized design and using this to further synthesize it down to a gate level placed and routed design. Most commercial tools support Verilog and/or VHDL descriptions of the design as input, which PowerOpt can export using the `export_verilog` or `export_vhdl` command.

The used inhouse solution on the other hand only needs to have the area of each component to generate the rough placement. PowerOpt can export the area values at different levels using the report_area command. Using the command

```
report_area -components -cdb_component
```

the output visible in Output 5 can be generated. As with the power report the generated area report consists of a (hierarchical) component name, the area of the component and underlying CDB component. The optimization tool can therefore match the area and the power values of each component and identify the used CDB component. This is key for the generation of the synthesis constraints.

```
report_area ?-help? ?-precision? ?-instances? ?-resources? ?-components?
            ?-cdb_component? ?-memories? ?-noheader? ?-file?

Description:
This command prints an area report based on a synthesis run.

Keywords:
  -precision:     Integer value defining the precision of reported
                  numbers.
  -instances:     Report instance specific area numbers.
  -resources:     Report area of individual datapath resources like
                  adders and registers.
  -components:    Report area per component (like clock, controller,
                  etc.).
  -cdb_component: Reports the CDB component name of the resources.
                  This option can only be used together with option
                  '-resources'.
  -memories:      Report area per memory instance.
  -noheader:      Ommits the report header.
  -file:          If submitted, the output is stored into this file.
                  Otherwise, the report is written into the console.
```

Output 4 – **`report_area –help`** information

```
Resource report:

      component                               area     CDB-component
----------------------------------------------------------------------
Design.process_image.val_abs3.this
  Reg_1                                     896.000  DEFAULT::Reg
  Mux_Reg_1_data_in                         563.200  DEFAULT::Selop_small
  Reg_5                                     448.000  DEFAULT::Reg
  Reg_6                                     448.000  DEFAULT::Reg
  Reg_7                                     448.000  DEFAULT::Reg
  Reg_9                                     448.000  DEFAULT::Reg
  Add_2                                     353.279  DEFAULT::Add_small
  Add_1                                     267.572  DEFAULT::Add_small
  Inc_1                                     161.368  DEFAULT::Inc_small
  Mux_Add_1_in00                            128.000  DEFAULT::Selop_small
  Mux_Add_1_in01                            128.000  DEFAULT::Selop_small
  Mux_Inc_1_in00                            128.000  DEFAULT::Selop_small
  Mux_Reg_5_data_in                         128.000  DEFAULT::Selop_small
  Mux_Reg_6_data_in                         128.000  DEFAULT::Selop_small
  Neg_1                                     111.033  DEFAULT::Neg_small
  Ctrl                                       67.200  POWEROPT::Internal
  Mux_Y_back_readWritePort_0.A0              52.000  DEFAULT::Selop_small
  Cmp_Ls_1                                   37.903  DEFAULT::CmpLt_small
  Mux_Cmp_Ls_1_in01                          32.000  DEFAULT::Selop_small
  Reg_3                                      14.000  DEFAULT::Reg
  Reg_4                                      14.000  DEFAULT::Reg
  Enc_1                                       1.200  POWEROPT::Internal
  Sll_1                                       0.000  DEFAULT::Sll_small
  Sll_2                                       0.000  DEFAULT::Sll_small
  Sll_3                                       0.000  DEFAULT::Sll_small
  Srl_1                                       0.000  DEFAULT::Srl_small
```

Output 5 – Excerpt of the **`report_area`** output

2.2 Place and route API

The inhouse blocklevel place and route solution is based on the open source block-packer CompaSS [6]. CompaSS is a well-known [7] tool that produces optimal slicing packings consisting of hard or soft blocks. It has been modified to support hierarchical designs and enable block tracking by setting unique IDs.

The input of the blocklevel floorplanner consists of a simple textfile. Using a simplified input file as an example the format of this file can be seen in Output 6.

```
Input file example:

[
[
[ 5002.756  0.5 2.00  // Design.process_image.val_abs3 ]
[ 5002.756  0.5 2.00  // Design.process_image.val_abs2 ]
[ 5002.756  0.5 2.00  // Design.process_image.val_abs ]
[ 4376.856  0.5 2.00  // Design.process_image.this ]
[ 5333.401  0.5 2.00  // Design.process_image.sum_image ]
[ 5255.659  0.5 2.00  // Design.process_image.sub_image ]
[ 7031.671  0.5 2.00  // Design.process_image.multiply ]
[ 5145.373  0.5 2.00  // Design.process_image.max_image ]
[ 12873.912  0.5 2.00  // Design.process_image.erosion ]
[ 12428.497  0.5 2.00  // Design.process_image.dilatation ]
[ 16221.904  0.5 2.00  // Design.process_image.convolution_rect2 ]
[ 16045.308  0.5 2.00  // Design.process_image.convolution_rect ]
[ 4813.722  0.5 2.00  // Design.process_image.binarisation2 ]
[ 7147.182  0.5 2.00  // Design.process_image.binarisation ]
]
[ 8835853.907  0.5 2.00  // GLOBAL ]
]

APART Design.process_image.sum_image Design.process_image.erosion
NEAR Design.process_image.erosion Design.process_image.val_abs
```

Output 6 – Place & Route simplified input file

The hierarchical information of the design is represented as nested squared brackets in the file. In the example shown in Output 6 the design consists of 15 sub-blocks. There are in total 13 sub-blocks representing sub-functions, one sub-block for the GLOBAL variables etc. and one sub-block for the logic of the main function (`Design.process_image.this`). Each leaf node in the hierarchy consists of an area value in μm^2, a minimum and maximum value for the aspect ratio of the generated block and the hierarchically unique ID used in PowerOpt.

Using this simple input format is sufficient for the inhouse blocklevel floorplanner to generate an unconstrained floorplan. But as it is essential to influence this result to get at better temperature distribution and therefore a better reliability value of the generated design, placement constraints can be added to the input file. Two of these constraints can be seen in Output 6 after the nested block information. Each constraint consists of three parts. First the type of constraint is specified (`APART` or `NEAR`), meaning the preferred distance, followed by the two IDs for which this constraint should take effect.

In the example given the block for `Design.process_image.erosion` should not be placed near to `Design.process_image.sum_image` but rather toegether with `Design.process_image.val_abs`.

The resulting floorplan is then exported into a textfile like the one shown in Output 7.

```
Output file example:

3064.23
2970.86
15
90.0608 48.5989 0 0 // Design.process_image.this
90.0608 53.4497 0 48.5989 // Design.process_image.binarisation2
90.0608 55.5487 0 102.049 // Design.process_image.val_abs3
90.0608 55.5487 0 157.597 // Design.process_image.val_abs2
90.0608 55.5487 0 213.146 // Design.process_image.val_abs
90.0608 57.1322 0 268.695 // Design.process_image.max_image
90.0608 58.3569 0 325.827 // Design.process_image.sub_image
90.0608 59.2201 0 384.184 // Design.process_image.sum_image
90.0608 78.0771 0 443.404 // Design.process_image.multiply
90.0608 79.3597 0 521.481 // Design.process_image.binarisation
90.0608 138.002 0 600.841 // Design.process_image.dilation
90.0608 142.948 0 738.842 // Design.process_image.erosion
90.0608 178.161 0 881.79 // Design.process_image.convolution_rect
90.0608 180.122 0 1059.95 // Design.process_image.convolution_rect2
2974.17 2970.86 90.0608 0 // GLOBAL
```

Output 7 – Place & Route simplified output file

The first two lines of the output specify the overall size of the design, while the third line lists how many blocks are following. Each following line consists of four decimal value representing the height, length of the block and the coordinates of its lower left corner. The already familiar unique ID generated from the PowerOpt hierarchy completes the line. A visualization of the generated floorplan can be found in [10].

2.3 Physics API

The multiphysics simulation has been implemented in the Matlab numerical computing environment. It consists of a very fast Green Function based thermal estimation and a NBTI Phase Space simulation. Both simulations have been published [8][9] and have been well received in the scientific community.

To use a green function in the thermal estimation this function first has to be calculated. In essence the green function is a package specific temperature map for a defined sample power dissipation. For the calculation of this temperature map different classic approaches can be used like the finite element or finite difference method. The finite difference method has been chosen to be used in the presented flow but can easily be replaced with another thermal estimation technique if desired. For the estimation the package is broken down into cuboids and information about their physical properties and their location in the package is given to the finite difference algorithm. This is done by generating a Matlab .m file which includes all parameters in form of simple variables or matrixes and calling Matlab to run a predefined script which calls the implemented finite difference algorithm and exports the generated green function.
As the calculation of this green function takes quite a long time compared to the runtime of the then possible thermal estimation using the green function it is beneficial that this only has

to be done once for each package. To further improve the reusability of this result it is saved under a filename which can be generated from the package information used. Therefore even if the flow has been used with different packages the correct green function can be found and used if it has been generated before.

For the actual temperature estimation the same approach as for the green function calculation has been chosen. The flow generates a Matlab .m file including the necessary parameters in form of simple variables and matrixes and then starts Matlab with a call to a predefined script. A shortened example of this generated .m file can be seen in Output 8. It mainly consists of the matrixes for the switched capacitances, leakage values, supply voltages, temperature values and information about the resistance of the supply voltage grid.

```
Input file example:

runID        = 1;
matrixSizeX  = 75;
matrixSizeY  = 50;

switchedCap  = [ 355440 … … … 268670 ];

leakageCur   = [  61709 … … …  59915 ];

VddMaps{1}   = [      1 … … …      1 ];

TmpMaps{1}   = [    298 … … …    298 ];

r            = [      0 … … …      0 ];
```

Output 8 – Multi Physics parameter file example (shortened)

After the estimation has been run the resulting temperature map and supply voltage map are then exported to a CSV file using the Matlab own `dlmwrite` function. This file can then be parsed and if necessary used to update the leakage values and a new iteration of the estimation can be started.

If the results have stabilized the NBTI aware reliability simulation is run the same way. A Matlab .m file including the parameters is generated, Matlab is called to run a predefined script and the results are again exported into CSV files. These results are again parsed and if necessary used to update the leakage values and the flow can start the thermal estimation with the updated parameters again.

As specified in [2] this is repeated until a stable temperature and aging estimation has been reached. An example of a stabilized estimation can be found in [3].

3 Design methodology and Tools for a holistic 2.5/3D IC Implementation

The following subsection 3.1 will present the created design flow of IMEC by first describing a global overview, followed by a short description of the used virtual physical prototyping and is completed by a presentation of the developed and assembled EDA tool flow.

Subsection 3.2 will then present the application of the presented flow on a memory-on-logic design case as a concrete example of 2.5 and 3D integration.

3.1 Design Flow

3.1.1 Global overview

The overall view of the proposed design flow is shown on Figure 2. The flow methodology is composed of 3 phases:

1) **Architectural exploration** — Here we focus on the application-driven micro-architectural design exploration without any concerns about the physical properties of the future integrated circuit. The goal is to establish the global architectural definition of the system that will serve as an input for the Virtual Prototyping phase (the next step). Known exploration techniques can be used during this phase: high-level instruction/memory/NoC simulators (e.g. Flexus, MacPat, …), SystemC behavioral models and whatever proprietary techniques one might have habit using. Since in the examples, the architecture of the system is known in advance, this step is not illustrated.
2) **Virtual physical prototyping** — Architectural description refined in the previous step is here augmented with physical properties of the design (basically the properties of the circuit). The system is described using RTL specification of the components (when available). Components for which synthesizable RTL does not exist at this stage are modeled as black boxes: that is a minimal set of data required to describe a physical portion of the silicon (typically we use area or gate count, power dissipation values, timing information, etc.). Also we can supply the constraints, for timing (through standard SDC files that typically capture the following constraints: operating conditions, wire load models, design rules, timing, area & power constraints, etc.) or manual constraints for floorplanning (physical placement constraints). After synthesis, place and route, we can extract relevant system parameters (area, timing, power, thermal, mechanical properties, etc.) to establish the quality of the given design point. Since the flow is fast, many design iterations are feasible to explore deeply design space.
3) **Standard design flow** — Here we use the standard design flow tools based on established EDA practices (e.g. Synopsys, Cadence). But after applying the virtual prototyping, the input is now a stable specification. The standard design flow is not used for design exploration but for design implementation.

The objective here is to replace the standard design practice, most of the time based on very approximate design exploration (using spreadsheet type back-of-the-envelope calculations) that directly feeds the Standard Design Flow to produce the final layout. Since design set-up and iteration time within the standard design flow is expensive in both man and CPU time, the

idea is to generate a mock-up of the standard flow that will trade-off the speed with accuracy: this is Virtual Design Prototyping practice.
Since the Virtual Prototyping is much less expensive in terms of designer/CPU time, we can enable more iterative flow, to manually explore various design space points. The objective is, by deep exploration of the design space, to come up with only selected design space points that will be then pushed and actually implemented using standard design flow tools. At the interface between the two we will find basically a very precise system specification.

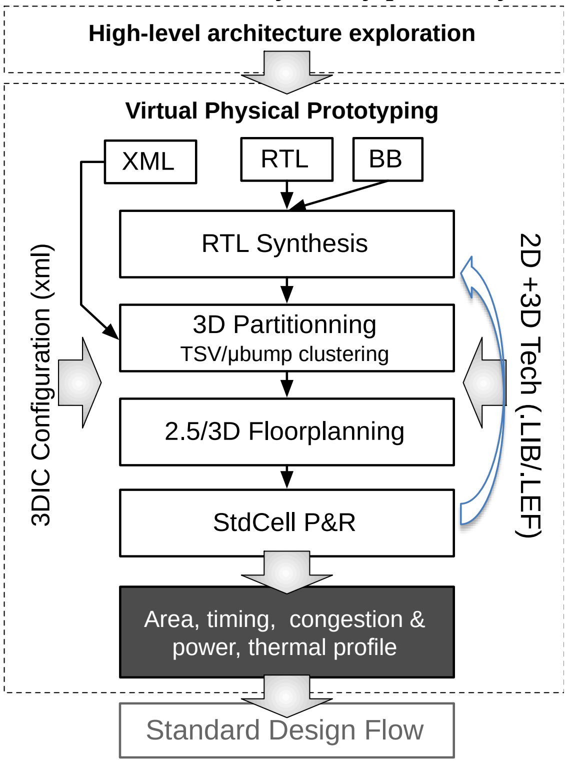


Figure 2 Three phases of the prototyping design flow: high-level exploration, virtual prototyping and standard design flow

3.1.2 Virtual Physical Prototyping

Virtual physical prototyping is a design practice that allows system architects to plan advanced packaging ICs in a holistic fashion and before the actual design flow. During this design phase, we typically perform the following steps: 3D design partitioning; TSV/µbumps array partitioning, clustering, place and route; 2D and 3D technology parameters choices and their co-optimization with the architectural design; 3D floorplanning with a standard cell placement and route; and an early mechanical, thermal and reliability assessment. Virtual prototyping does not aim to modify the current design methodology, but it is rather used before standard industrial design flow tools.
The input to the flow is an architecture described using synthesizable RTL (VHDL or Verilog) or in black-box RTL stubs. Since we use hard macros for the CPU core and L2 cache, these components are described as high-level, black-box (BB) models as shown in Figure 2. The black-box description provides a minimal set of information required to perform design flow iteration, and includes: the interface definition, the area, timing and power models. Besides the design data, the virtual prototyping system relies on library information from IP vendors (.LEF/.LIB) and technology, electrical, physical and design rule parameters from the foundry. The design is partitioned and floorplanned on a per-tier basis. After P&R, we extract the parasitics and run performance/area analyses to assess and verify the architectural and technology choices for a given design configuration. The choices are

based on metrics such as area, timing and congestion analysis both for the front- and back-side. The power distribution is fed to the virtual prototyping infrastructure to generate power density maps. This information, together with the 3D stack configuration (die, BEOL and the interface thickness and properties, the package/cooling thermal resistance, etc.) is forwarded to a thermal modeling tool that generates a thermal profile for each die in the system. In our analysis, we consider average power and power density distributions to generate the thermal profiles.

3.1.3 Assembled/developed EDA tool flow

The 3D SpyGlass Physical® tool from Atrenta is used for virtual physical prototyping of the 3D-stack (the tool is also supporting Silicon Interposer implementation). The standard 2D version of the tool has been extended to support the 3D integration requirements: the backside routing capability (including the congestion analysis and on-the-fly technology exploration of the TSV diameter, pitch and the Keep-Out Zone area size), support for easy TSV and µbump array partitioning, clustering and placement constraining. For the thermal analysis, we use the Compact Thermal Model, developed at IMEC that has been silicon-validated using similar 3D stack configurations. The whole flow process, that is die prototyping and thermal modeling are illustrated in Figure 3.
The overall run-time of the flow is short even for complicated designs that are fully described at the RTL level. The design set-up time is typically measured in hours and an average iteration time is measured in tens of minutes (including floorplanning, P&R and parameter extraction). A temperature profile, for a given floorplan or 3D stack configuration is extracted in just a few minutes.

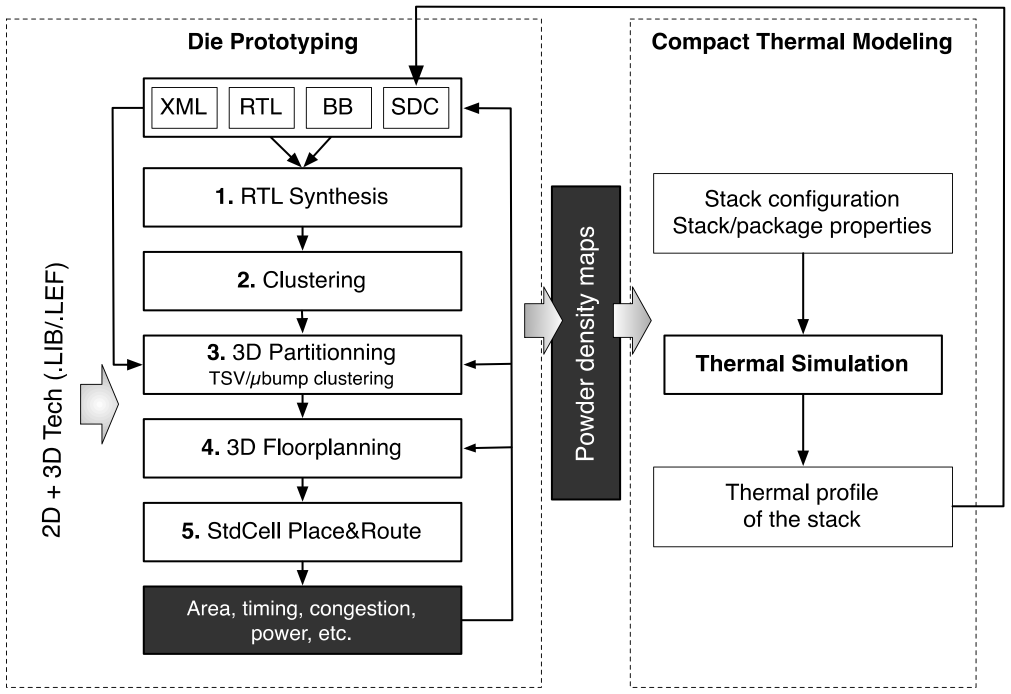


Figure 3 Die prototyping and Compact Thermal Model interaction

3.2 Examples of optimization

In the following we show:

1. How to co-optimize the choice of the TSV form factor (diameter/height) and RDL number width and pitch as function of the TSV array partitioning placement and route

2. Comparison of the logic to memory interface properties through wirelength estimations for 2.5 and 3D integration scenario
3. Congestion of the OpenSPARC core for advanced technology nodes

3.2.1 TSV array partitioning and optimization

First, in the case of the 3D integration we will not consider the trivial case where the TSVs are placed exactly below the µbumps (i.e. the case where TSVs and µbumps are aligned and have the same x, y coordinates). This means that the TSVs can be placed anywhere in the logic die. This can be important since it can relax the constraint of pre-placing the TSVs at a given position imposed by the position of µbumps and before placing the actual modules on the logic die.
In such configuration the connectivity between the TSVs and the µbumps will be made using redistribution layer (RDL). Further, the TSVs can be clustered in arbitrarily sized arrays. In this particular case using only one TSV array will not be a good design choice because even with very aggressive TSV pitches, accommodating 1044 connections into a single array would be difficult. First, it will generate a very strong constraint for the placement of the logic die. Then, because of the escape routing problem, the backside routing layers would either require very fine RDL pitches or fewer metal layers, which is to be avoided for cost reasons. It is therefore obvious that we need to co-optimize: a) the way TSV array is partitioned; b) the properties of the back side routing layers (namely the width/pitch of the tracks) and c) the number of RDL layers (direct impact on the manufacturing cost).
To understand the different trade-offs we analyze the feasibility of the routing, i.e. congestion analysis, for various TSV and RDL pitches (we assume TSV and µbump pitch of respectively 30, 20, 10µm and 10, 5 and 2µm) and only one routing layer. Figure 4 shows congestion maps for 4 different TSV/RDL pitch combinations (for sake of clarity we show only the routing on the backside). Red color code indicates very high congestion (design is basically not routable) with blue and green indicating feasible congestion. For a given example we can easily understand that for this particular combination a minimum of 5µm pitch is required to route TSVs with 20µm pitch. When TSV pitch increases to 30µm the RDL pitch could go up to 10µm for a similar congestion pattern.

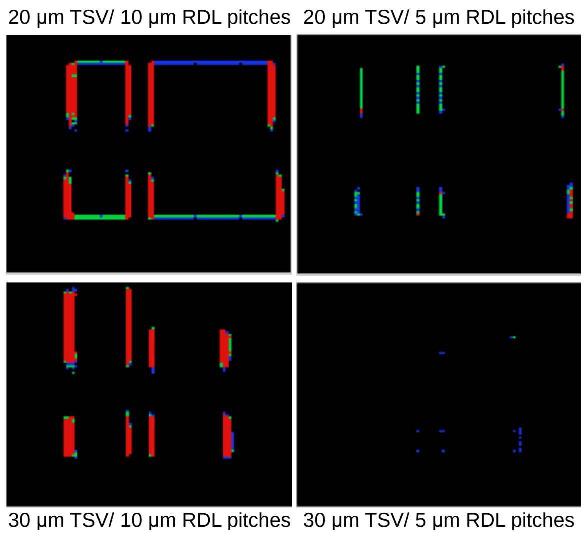

20 µm TSV/ 10 µm RDL pitches
20 µm TSV/ 5 µm RDL pitches
30 µm TSV/ 10 µm RDL pitches
30 µm TSV/ 5 µm RDL pitches

Figure 4 Congestion analysis of the backside using various TSV/RDL pitch values

3.2.2 Wirelength comparison

Design prototyping tool can extract, after synthesis, floorplanning, place&route on both front and backside of all dies, geometrical information about any net in the design. Typically, the geometrical information is the wirelength and the exact distribution of this length across different metal layers.

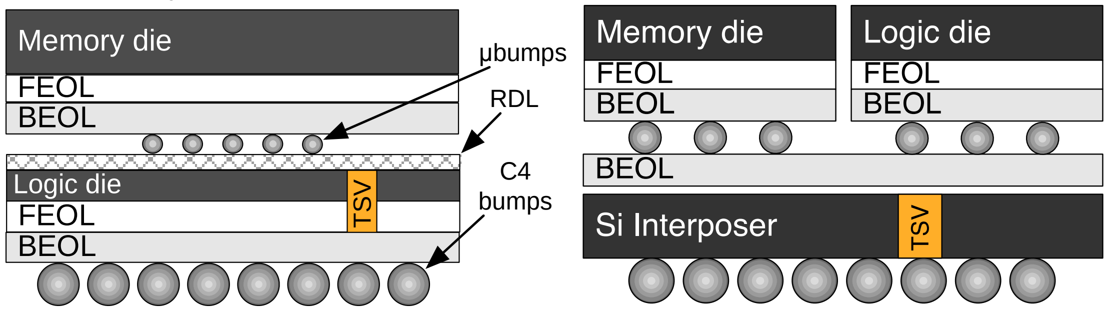


Figure 5 Cross section of the face to back stacked circuit and side by side integration using the interposer

For both 3D and 2.5D integration scenarios we extracted the minimum, the average, the maximum and the total wirelength, focusing only on inter-die nets (those between WideIO DRAM controller block on the logic die and the actual μbumps of the wide IO DRAM). Table 1 summarizes the results obtained.

Table 1 Wirelength statistics for the 3D and 2.5D Integration Scenarios

	Min	**Average**	**Max**	**Total**
3D	2775.24	4079.85	5206.26	4309327.00
2.5D	3801.48	7216.75	9968.16	7620890.00

The maximum (basically the most important one) and the total wirelength doubles when moving from 3D to the 2.5D case, which is normal, having in mind the sizes of the logic and the WideIO DRAM dies and the given position of the μbumps on the Wide IO DRAM side. There is very little room for the improvement (even if we consider best case scenario, where WideIO DRAM is rotated 90° clockwise). This information can be used, in conjunction with appropriate delay models to perform static timing analysis on the inter-die nets.

3.2.3 CPU core congestion analysis and area vs. number of metal layers optimization

Modern cores are complex circuits that require, as we move to more complex designs and more advanced technologies, more and more metal layers to accommodate complex wiring demands. Or, each wiring layer will add an extra cost, not only on the processing side, but also for mask preparation. It is therefore crucial that designers optimize the usage of the metal layers, especially for the advanced technology nodes.

In this example we show how one can mitigate, and co-optimize, relaxed area constraints versus the number of wiring layers used. To demonstrate this we will be using only one core, taken from the OpenSPARC multi-processor SoC. Figure 6 shows two floorplans (showing the top hierarchical level blocks only, standard-cells are not shown for image clarity reasons) obtained using industry standard 45nm and technology files scaled to 20nm. Note that the images shown are not to scale.

20nm technology node offers significant improvement in the area (3.6 times more to be more precise). The question is of course, what happens to the routing. To understand the impact of

the new technology choice on the routing, we performed congestion analysis and will express the end results as congestion score: the ratio of congested horizontal/vertical to the total edges of the die (expressed in %).
Empirical trials tells us that the congestion score of 10 (that is 10% of edges have more routes required by the design then what is actually provided by the technology — the ration between track demand and track supply is > 1) will be very difficult to route in reasonable amount of time using standard backend tools for layout generation.

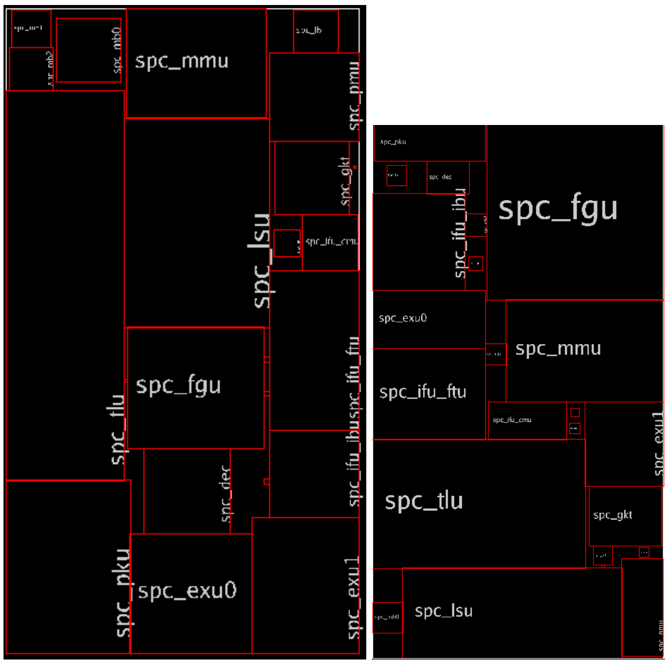


Figure 6 OpenSPARC core: floorplans for 45 and 20nm technologies

Figure 7 shows congestion maps for different system settings. The leftmost figure shows the industry standard 45nm library results with 9 metal layers. The global congestion score is less then 5, indicating that the routing is feasible. This is expected as this circuit has been implemented and results already presented in the literature.
However when we move to 20nm node, also with 9 metal layers, we see that the global congestion score goes up to 15, indicating that the actual routing at layout stage will be very difficult. Increasing the number of metal layers to 11, the rightmost picture, shows improvements in the congestion, since the global congestion score is now less then 10 (border line from the routability perspective).

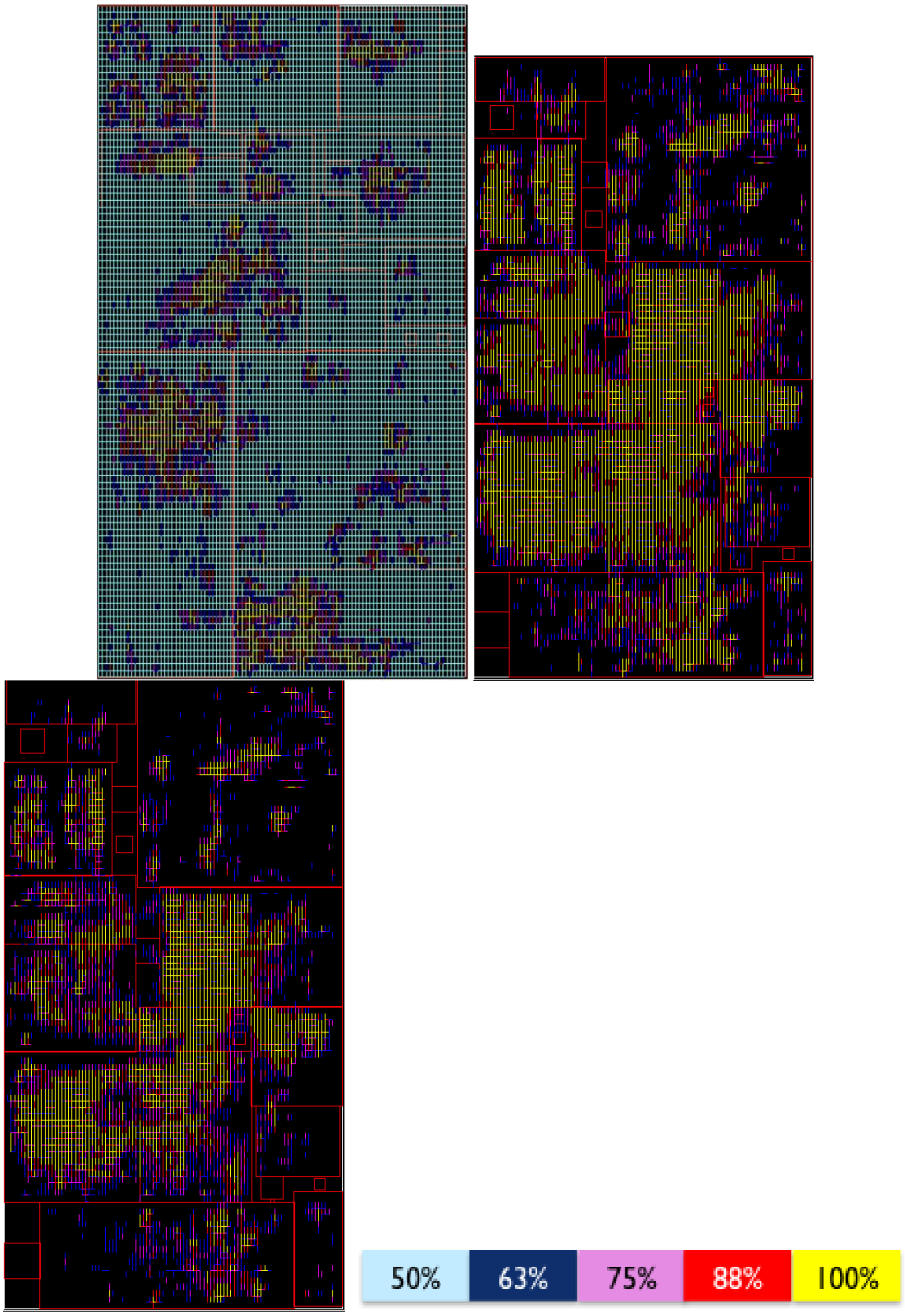


Figure 7 Congestion analysis: 45nm/9 metal layers (ML); 20nm/9 ML; 20nm 11 ML. The congestion legend is show on the bottom: yellow color code indicated the congestion above 100%

Another way of solving the congestion problem (other then modifying the actual core micro-architecture which is a non-preferred method having in mind the implications of the micro-architectural change) will be to allow more "empty" space around the standard cells. This can be easily achieved by lowering the area utilization parameter. Figure 8 shows how 20% of the area increase will reduce the global congestion score from 10 to 7, enabling much easier routing with 2 metal layers less.

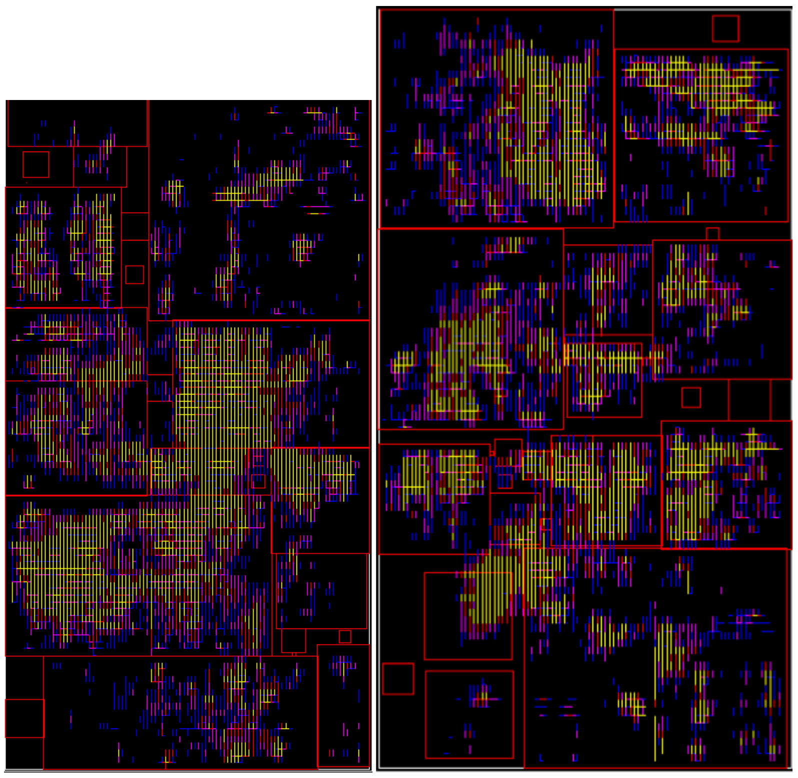


Figure 8 Impact of the area increase on the routing resources requirements: on the left 20nm with 11 metal layers and on the right 9 metal layers with 20% extra area.

4 Integration of the physical-level thermal simulation and optimization flow

This section describes the integration of the physical level thermal simulator within a standard digital design flow. While D6.2.2 and D6.3.2 focuses on the power and thermal flow and on its evaluation through the adoption of real-life test cases, the integration of the simulation environment and the file formats utilized as interfaces of the tools will be described in details in this section. The flow can be divided in three sub-tasks: the generation of the power maps utilized at the physical level for the floorplan and place & route evaluations, the generation of the CTM (Chip Thermal Model) containing the temperature-dependent power of each instance of a chip, and the thermal simulator that compute the thermal maps. All the described flows are integrated to the standard digital design flows. Physical and logical information of the design (e.g., LEF, DEF, GDS …) can be exported from the place & route flow (SoC Encounter has been utilized in this context) at all the steps of the design (e.g., post placement, post clock tree synthesis, post route optimization) according to the design and optimization flow described in D6.3.1, and they are linked to the thermal analysis flow through standard exchange formats:

- DEF (Design Exchange Format): specify instances (std cells, memories, pads, analogue components) and net positions within the design.
- verilog: specify logical connectivity among the instances of the design.

Moreover, the file formats required for the generation of the libraries are:

- GDSII file. Physical description of circuit elements. Utilized to create DEF views not available for custom macro blocks.
- LEF (Library Exchange Format): Physical description of library cells interfaces.
- LIB (synopsys-format library): logical descriptions of library cells, with power and timing tables.
- SPI: Spice netlist.

In the following, the listed files will be referred to as *chip database*. The first step required for the thermal analysis and optimization flow involves the generation of the power maps of the die or dies composing the system. Figure 9 provides a simplified view of the power maps generation flow.

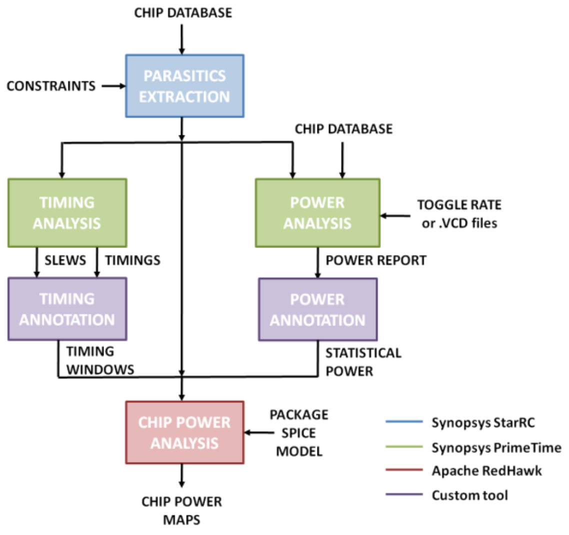
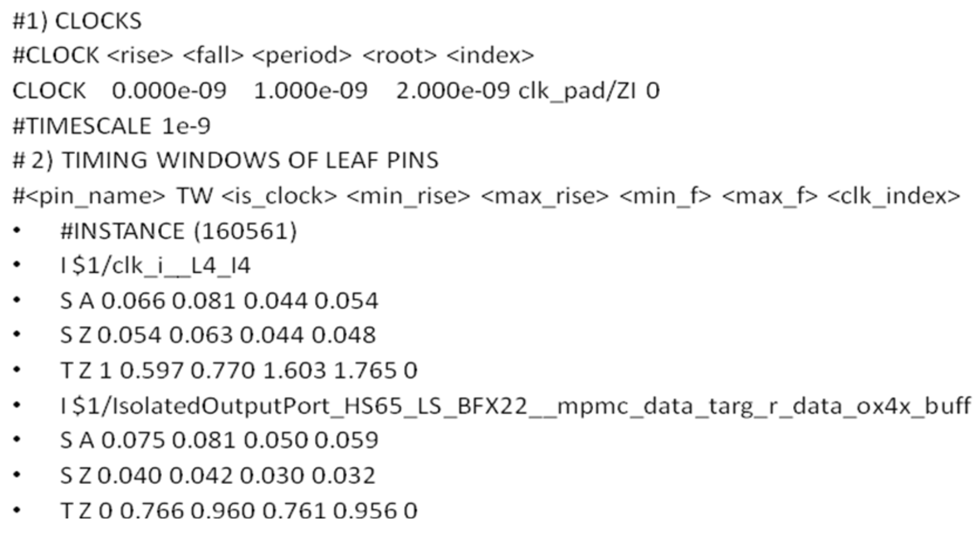


Figure 9: Simplified view of the chip power maps generation flow and integration.

The first step of this flow involves the parasitic extraction that uses as input the chip database and generates a parasitic annotation file in a *Standard Parasitic Exchange Format* (*SPEF*). This generated SPEF file, together with the chip database is utilized by both the timing analysis engine and the power analysis engine that generate two custom intermediate file format utilized in the flow:

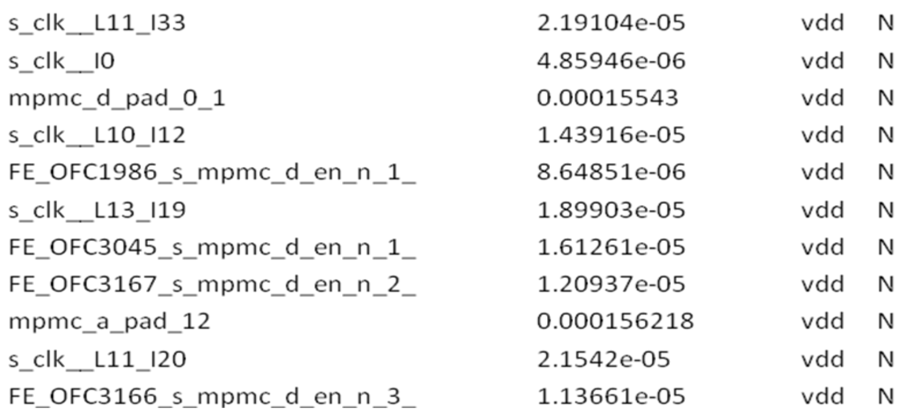
- A timing window file, including the parameters for each instance of the design to perform the dynamic updating of the power instance file.
- A power annotation file, that includes the power consumption of each instance of the design, computed statically.

Figure 10 shows an example of the timing window file. Within the timing window file, each pin of each instance within the design is annotated with four parameters: a min rise time, a max rise time, a min fall time and a max fall time. These parameters are computed to perform the dynamic power calculation as described in D6.2.1. Figure 11 shows an example of power annotation file.



```
#1) CLOCKS
#CLOCK <rise> <fall> <period> <root> <index>
CLOCK   0.000e-09   1.000e-09   2.000e-09 clk_pad/ZI 0
#TIMESCALE 1e-9
# 2) TIMING WINDOWS OF LEAF PINS
#<pin_name> TW <is_clock> <min_rise> <max_rise> <min_f> <max_f> <clk_index>
•   #INSTANCE (160561)
•   I $1/clk_i__L4_I4
•   S A 0.066 0.081 0.044 0.054
•   S Z 0.054 0.063 0.044 0.048
•   T Z 1 0.597 0.770 1.603 1.765 0
•   I $1/IsolatedOutputPort_HS65_LS_BFX22__mpmc_data_targ_r_data_ox4x_buff
•   S A 0.075 0.081 0.050 0.059
•   S Z 0.040 0.042 0.030 0.032
•   T Z 0 0.766 0.960 0.761 0.956 0
```

Figure 10: Example of timing windows file.



```
s_clk__L11_I33                      2.19104e-05       vdd   N
s_clk__I0                           4.85946e-06       vdd   N
mpmc_d_pad_0_1                      0.00015543        vdd   N
s_clk__L10_I12                      1.43916e-05       vdd   N
FE_OFC1986_s_mpmc_d_en_n_1_         8.64851e-06       vdd   N
s_clk__L13_I19                      1.89903e-05       vdd   N
FE_OFC3045_s_mpmc_d_en_n_1_         1.61261e-05       vdd   N
FE_OFC3167_s_mpmc_d_en_n_2_         1.20937e-05       vdd   N
mpmc_a_pad_12                       0.000156218       vdd   N
s_clk__L11_I20                      2.1542e-05        vdd   N
FE_OFC3166_s_mpmc_d_en_n_3_         1.13661e-05       vdd   N
```

Figure 11: Example of power annotation file.

Apache redhawk computes as input the files generated by the flow, together with the chip database, and it can generate as an output the power map, formatted as shown in Figure 12. The first line of the file indicates the coordinates of the chip boundaries (in microns), the second line indicates the number of points, and the others reports the average power associated to each point within the grid of the die.

```
0 0 3800 3800
10000
0       0       38      38      1.5825888e-10
38      0       76      38      1.5825888e-10
76      0       114     38      1.5440736e-10
114     0       152     38      8.1494208e-11
152     0       190     38      3.7743456e-10
190     0       228     38      1.0211808e-10
228     0       266     38      8.5076352e-11
266     0       304     38      9.0935712e-11
304     0       342     38      3.7943232e-10
342     0       380     38      8.8174848e-11
380     0       418     38      2.0860224e-10
418     0       456     38      3.1280352e-10
```

Figure 12: Example of Apache redhawk power map.

The second flow leads to the realization of the Chip Thermal Model (CTM), a binary file that contains the information about the temperature-dependent power consumption of each instance within a die. A block scheme of the flow is provided in Figure 13.

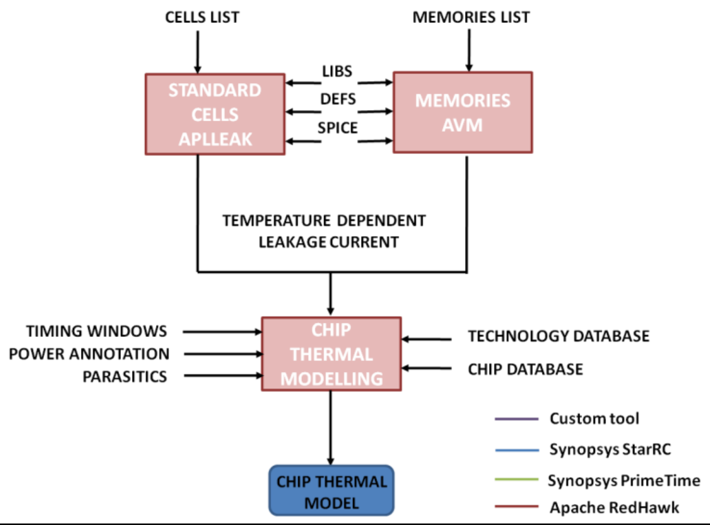


Figure 13: Block scheme of the CTM generation flow and integration.

As the objective of the task is the realization of the temperature-dependent characterization of a chip, the first step consists of the generation of the temperature dependent characterization of the cells, memories, and analog IPs (pads) composing the system. As this step is accomplished performing spice simulations of the library cells, the inputs required are the spice netlists of the cells, the .lib and .def files of the cells containing timing, power, and physical information utilized to calculate the power with interpolation in case of unavailability of the spice netlists. An example of leakage file is shown in Figure 14.

```
data_version cell.leakage 5v3
tool_name aplleak rail_slew
version 7.1 rel 0B
Released Date: 04/07/2012
data_tag asc 1351532745 Mon Oct 29 18:45:45 2012
cell_temp 864 5
25 50 75 100 125
HS65_LSS1_FA1X21 0 2 vdd 25.000000 1.70825e-09 1.71204e-09 50.000000 5.47131e-09 5.58549e-09
    75.000000 1.50661e-08 1.55872e-08 100.000000 3.64621e-08 3.81325e-08 125.000000 7.9247e-08
    8.36372e-08 vss 25.000000 1.7088e-09 1.73232e-09 50.000000 5.47765e-09 5.61219e-09
    75.000000 1.50804e-08 1.56246e-08 100.000000 3.64873e-08 3.81871e-08 125.000000 7.92887e-
    08 8.37172e-08
HS65_LSS1_FA1X35 0 2 vdd 25.000000 3.22481e-09 3.1414e-09 50.000000 1.03511e-08 1.0184e-08
    75.000000 2.85305e-08 2.82398e-08 100.000000 6.9098e-08 6.87147e-08 125.000000 1.50265e-07
    1.50029e-07 vss 25.000000 3.23926e-09 3.43367e-09 50.000000 1.03748e-08 1.10458e-08
    75.000000 2.85697e-08 3.05359e-08 100.000000 6.91577e-08 7.41758e-08 125.000000 1.5036e-07
    1.61783e-07
...
```

Figure 14: Example of leakage file containing temperature-dependent information of the cells power consumption.

The basic information included within the leakage file are the number of cells and the number of temperature points characterized (in this case 864 and 5, respectively), the temperatures utilized for characterization (in this case 25°C, 50°C, 75°C, 100°C and 125°C), and the temperature dependent leakage power of each cell characterized. The final step of the sub-task consists of the generation of the CTM, by updating the power information available from previous steps, with the temperature-dependent leakage information of each instance of the design.
The CTM act as interface toward the thermal simulation flow as well as power maps generated utilizing redhawk (without temperature-dependent leakage power computation) and hand-made power maps. An overview of the flow is shown in Figure 15.

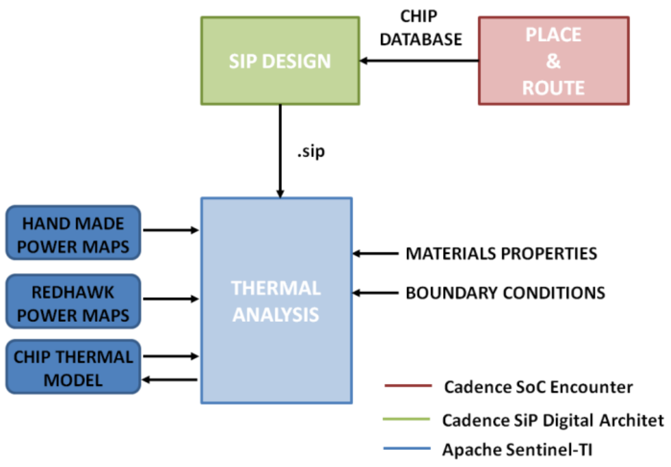
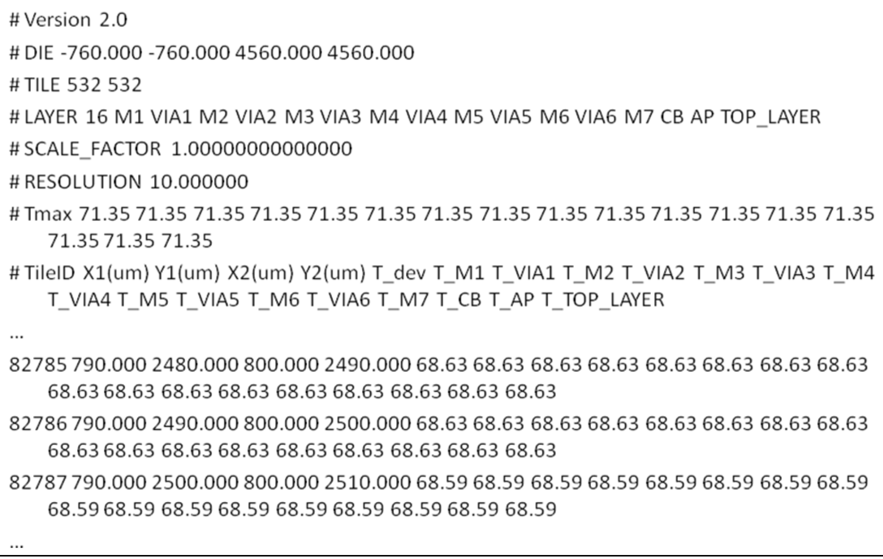


Figure 15: Overview of the thermal simulation framework integration.

An added value of the simulation flow is the full compatibility with tools for design of systems at package level. For this purpose Cadence SiP digital Architect has been selected. Such a tool enables the co-design at chip/package levels through the link with the Cadence Encounter place & route tool, enabling exploration of the optimal florplanning of basic blocks within system on chip, as well as optimal positioning of dies at package level. The exchange format between SiP Digital Architect and Apache Sentinel-TI is the .sip format.



```
# Version 2.0
# DIE -760.000 -760.000 4560.000 4560.000
# TILE 532 532
# LAYER 16 M1 VIA1 M2 VIA2 M3 VIA3 M4 VIA4 M5 VIA5 M6 VIA6 M7 CB AP TOP_LAYER
# SCALE_FACTOR 1.00000000000000
# RESOLUTION 10.000000
# Tmax 71.35 71.35 71.35 71.35 71.35 71.35 71.35 71.35 71.35 71.35 71.35 71.35 71.35 71.35
    71.35 71.35 71.35
# TileID X1(um) Y1(um) X2(um) Y2(um) T_dev T_M1 T_VIA1 T_M2 T_VIA2 T_M3 T_VIA3 T_M4
    T_VIA4 T_M5 T_VIA5 T_M6 T_VIA6 T_M7 T_CB T_AP T_TOP_LAYER
...
82785 790.000 2480.000 800.000 2490.000 68.63 68.63 68.63 68.63 68.63 68.63 68.63 68.63
    68.63 68.63 68.63 68.63 68.63 68.63 68.63 68.63 68.63
82786 790.000 2490.000 800.000 2500.000 68.63 68.63 68.63 68.63 68.63 68.63 68.63 68.63
    68.63 68.63 68.63 68.63 68.63 68.63 68.63 68.63 68.63
82787 790.000 2500.000 800.000 2510.000 68.59 68.59 68.59 68.59 68.59 68.59 68.59 68.59
    68.59 68.59 68.59 68.59 68.59 68.59 68.59 68.59 68.59
...
```

Figure 16: Example of chip thermal profile exchange format.

After the thermal simulation, Sentinel-TI generates a report file, reporting information about the peak temperature within each die, the thermal maps of the components composing the system (die, package, board), a chip thermal profile file, and the converged power in the case of the CTM-based flow. The chip thermal profile consist of a text file containing information about the temperature calculated within all the layers of the chip, and can be imported again within redhawk to compute more accurate power analysis and power integrity simulations (IR-drop, electro migration) considering the temperature. An example of chip thermal profile file is shown in Figure 16.

5 Evaluation including Measurable Objectives

The potential of the developed optimization techniques (MO6.3.1) and therefore the partial achievement of MO6.3.2 have mostly been already presented in D6.3.2 [2]. To reiterate some of the results: different optimization approaches have been developed and it was shown that the developed techniques can reduce the temperature hotspots by ~10K, decrease temperature gradients, improve the time to failure (based on NBTI considerations) by 16%, and result in a wire length reduction of up to ~35% for different examples.

To completely address MO6.3.2 the created optimization techniques have been integrated into different design flows, as it has been shown in this deliverable, which have been examined regarding their ease of use, their runtime, and their accuracy.

5.1 Ease of use

The created design flows are very easy to use as only small adaptations of already existing scripts are needed to make them compatible with the automatic flows here presented. In case of the NBTI-aware Thermal Optimization Flow the existing synthesis script of a design can be extended to include the needed placeholders for the generated constraints and the flow needs no further human interaction to generate an optimized design. For the presented holistic 2.5/3D IC implementation the adaptation of the existing scripts depend on the design complexity, but the time needed is measured in minutes for simpler design (less than a couple of 100k gates) and only up to a couple of hours for more complex designs such as the OpenSPARC T2 (millions of gates).

5.2 Runtime

The runtime of the presented flows highly depend on the complexity and size of the designs which are being optimized. It ranges from a couple of minutes to up to a few hours for more complex designs like the OpenSPARC T2 microprocessor. Compared to the runtime of standard EDA tools available before Therminator of up to a couple of days this is a huge improvement and makes the presented optimization techniques feasible in the first place. The speed up is obtained by the different approximation methods developed in Task 6.2 like the green function based temperature estimation, which reduces the calculation time of a temperature distribution from several minutes down to a fraction of a second, the newly create NBTI model which is 600 times faster than the formerly used reaction-diffusion model, or simplifying the routing by ignoring actual standard cell location.

5.3 Accuracy

All the used approximations to speed up the design flows and enabling the optimization techniques have an influence on the accuracy of the results. By comparing the design parameters like area, max delay, power etc. of the die prototype and the final layout generation the accuracy of the estimation was within ~15% of the final layout and therefore well within the precision needed for the planning purposes of the holistic 2.5/3D IC implementation. The evaluation of the NBTI-aware Thermal Optimization Flow on the specified testcase 5 of the Therminator project is being reported D7.3.1 [10].

6 Conclusion

In this deliverable three different design flows integrating the formerly presented optimization techniques have been presented. It was shown that the flows are very easy to use, have a runtime which is better than the available EDA tools and are within the necessary accuracy to be useful and reliable. Using these new flows enables the user to analyse temperature distributions and automatically place and route 3D SiP and 2D SoCs to greatly improve temperature hotspots, temperature gradients and reliability of the resulting product in an easy and fast way.

Section 5 showed that the quantification of the measurable objectives 6.3.1 and 6.3.2 are fulfilled and the Task 6.3 has been completed successfully.