Hierarchical design flow methodology reuse and standardisation

Vaassen, A.W.P.

Award date:
2003

TU/e technische universiteit eindhoven

Faculty of Electrical Engineering
Section Design Technology For Electronic Systems (ICS/ES)
ICS-ES 836

Master's Thesis

HIERARCHICAL DESIGN FLOW METHODOLOGY.

Reuse and standardisation.

A.W.P. Vaassen

Supervisor:	prof.dr.ir. P.R. Groeneveld
Coach:	ir. J. Huisken (Philips Research Laboratories)
Date:	December 2003

Authors' address data: ing. A.W.P Vaassen WDA21; ad.vaassen@philips.com

Title:	Hierarchical Design Flow Methodology Reuse and standardisation.
Author(s):	Ing. A.W.P. Vaassen WDA21 Ad.vaassen@philips.com
Reviewers(s):	Prof. Dr. ir. P.Groeneveld, Ir. J. Huisken.
Technical Note:	2003/677
Version:	06, 28/09/2003 **Status:** On-going

Project:	100cube, MoSAIC, Master of Science Thesis
Customer:	Philips Research, Eindhoven University of Technology

Keywords: Floorplan, CAD, hierarchical, design flow, clustering, partitioning, block-based, tile-based, flat.

Abstract: In the IC design community, there is a productivity gap for large System-On-Chip (SoC) designs. This gap is the difference between design complexity and design productivity. Physical design is one of topics responsible for that gap.

For physical design, flat and block-based design methodologies are commonly applied. When looking at these methodologies, it can be seen that flat do not scale with complexity, and block-based methodologies only scale with increase in human and hardware resources. This means that neither of them will bring the solution. From automation and performance point of view flat is the better methodology, but as chips get larger and larger it will run into capacity problems. To solve this capacity problem without losing the automation and performance advantages, a tile-based design methodology is developed.

These standard tiles will look and behave as standard cells but significantly larger. This method is based on the work that was carried out in the seventies, when people changed from transistors to standard cells. Through this move automatic place and route, and synthesis came available. At this moment, the same kind of change could help to reduce the productivity gap and to make automatic floorplanning possible.

The most important question this report addresses will be the size and construction of the tiles.

Conclusions: At this moment a complete automated hierarchical design flow is made to convert a gate level netlist into a CoReUse database with gds2, timing and physical verification, as a final result. This flow is based on Cadence tools and contains about 30 makefiles, 2 Philips internal tools, and 150 scripts. The processor array of the Xetal project is used as an experiment. The result was that this design could be made 3 times quicker compared to the flat layout style in the same area. This gain in run-time gives a loss 17% in timing closure on the worst-case path. For the rest of the paths the tile-based method is about 8% better. Besides that also a flat run was done with minimum area. When comparing the tile-based method against this result, it is still 3.5 times quicker with the same timing result for the critical path. For the rest of the paths the result is 15% better. On area the tile-based method is about 23% worse. One reason for that is due to a bad area utilisation of the optimisation tiles. If this could be solved without wiring congestion problems the area is somewhat the same.

One of most important conclusions is that the tile-based method can handle larger designs than flat. Besides that the tile-based method provides the possibility to do design-space exploration.

Recommendations:

- More and different design cases,
- Clustering algorithm,
- Abstract modelling of layout,
- Optimal aspect ratio of tiles,
- Buffer insertion for pins,
- Choice between bottom-up or top-down design,
- Toplevel optimisation with regards to:
    - Timing,
    - Cross talk.

Contents

Acknowledgement

Even though I had just graduated from the 'Hogere Technische School Eindhoven' in 1996, I still had a strong desire to continue studying electronics. At that time, I preferred a job instead of a study at the University of Technology. After three years, the desire was so strong that I still started this Master of Science study. The combination of work and study has given me a better understanding of the lectures.

I would like to express my gratitude to my girlfriend, family and friends for their support during the course of this study. Besides my family there are a number of people that I would like to thank for realising this study. First of all, Engel Roza and Philips Research for giving me the opportunity to start this study aside my normal work. During the last 4 years I have very much enjoyed the Fridays that I spent at the university attending lectures. Secondly, I would like to thank Erwin Waterlander, Marino Strik, John Dielissen, Marc Heijligers, Paul Wielage, and Alexander Danilin for all the valuable discussions. A special word of thanks goes to my supervisors Jos Huisken of Philips Research and Patrick Groeneveld of the University of Technology Eindhoven for their confidence, cooperation, and guidance throughout the graduation project. Last but not least, I would like to thank the reviewers of my report for their time.

Ad Vaassen
Eindhoven, September 2003

1. Introduction

This is the final report of a M. Sc. thesis project of Eindhoven University of Technology, Department of Electrical Engineering, Information and Communication Systems group. The research work is conducted within Philips Research Laboratory Eindhoven from October 2002 until July 2003 as part of the MoSAIC project, which focuses on the semiconductor engineering productivity gap and design regularity.

1.1. Problem description

In the IC design community, there is a widening productivity gap for large SoC designs. This gap is the difference between design complexity and design productivity. The design complexity is driven by the growth of integrating more and more functionality in a product. This gives great pressure on the total time to create a chip for products, because they should not be obsolete before being brought to the market. Investment in technology improvements has dominated product creation resources. But design productivity has not been able to keep track with technology improvements such as transistor density growth. (See Figure 1.) One of the reasons for this is that broader ranges of design factors have to be taken into account when designing in a new technology. For example timing, power, reliability, manufacturability, signal integrity and testability requirements.

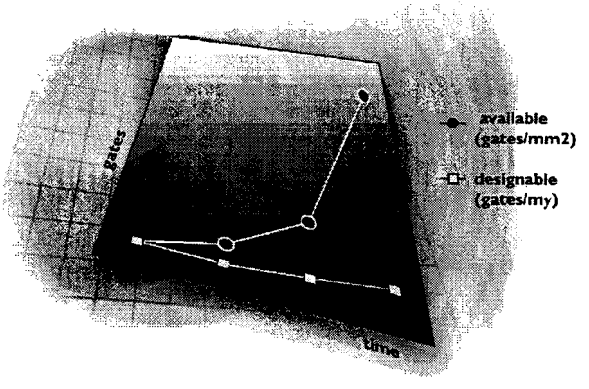

Figure 1: Design Gap

In the International Technology Roadmap for Semiconductors 1999 edition [1], figures are given of 58% complexity growth against 21% productivity growth per year. Most important reasons for this growing gap are functional verification, logic synthesis and physical design.

For physical design, flat and block-based design methodologies are commonly applied. Where most CAD vendors are working on flat, only a few put emphasis on block-based (2-level) design methodologies to succeed in designing (larger) chips. Instead of designing with a large number of cells, a small number of blocks are used, making manual floorplanning necessary. From automation and performance point of view flat is the better methodology, but as chips get larger and larger it will run into capacity problems. The number of designable objects will exceed the maximum and the total CPU time needed for design will be extremely large. Other observations that hamper this methodol-

ogy are the reuse of hard layout modules (IP) and the fact that design teams are becoming multi-site, where each site designs a part of the layout. This implies that physical design can only start if all the design teams have finished their part of the complete system. Block-based is at this moment the only alternative, but is not very well automated. Main reason is the absence of standardization. Besides that also more and different algorithms are needed, for partitioning, pin assignment, and relative placement.

As a conclusion, flat methodologies do not scale with complexity, and block-based methodologies only scale with increase in human and hardware resources. This means that neither of them will bring the solution to the productivity gap. Flat provides the most automated simplest methodology with the best density result. Main reason is that global and local optimisation can be performed at the same time. Block-based designs can address the capacity issue. However blocks are firm or hard which makes the floorplan a puzzle; compacting, and timing closure are more difficult to achieve. It also requires a floorplan methodology that is complex and inflexible because of heterogeneous sized blocks, time budgeting and block estimations.

1.2. Objective

The objective of this thesis is to investigate the viability of a hierarchical based design flow methodology based on reuse and standardisation, combining best of both worlds. Trying to use automation, density, timing, and methodology properties of flat and the capacity property of block-based. Main characteristic will be to use a standardized way of design, together with a divide and conquer strategy, to break down the problem into smaller suitable pieces (tiles). Most interesting question for this approach is the size and construction of these tiles.

1.3. Approach overview

A complete design methodology contains an enormous number of design steps depending on the layout approach. For this thesis the method of standard tiles will be used. More information can be found in Chapter 2.3. These standard tiles will look as standard cells but significantly bigger. This way of designing is based on work that was done in the seventies, when people changed from transistors to standard cells. Through this move automatic place, route, and synthesis came available. At this moment, the same kind of change could help to solve the productivity gap and to make automatic floorplanning possible.

In general, the enormous number of design steps can be clustered into a few basic topics: planning, implementation, assembly and verification. Planning consists of multiple tasks of which design partitioning is the most important step. Design partitioning breaks the design down into partitions, taking into consideration chip performance requirements, size, hard and firm core IP, and packaging. Besides partitioning also some library cells are identified that are needed for global optimisation. In Chapter 3, a closer look will be taken on this partitioning task.

After planning, work on the individual clusters and optimisation cells starts. The timing

constraints and size are determined for the implementation phase of each tile. Besides that the technology also gives some extra constraints to the tile implementation. Specific information about the tile implementations is described in Chapter 4.

When the tiles are ready, block models are created for toplevel assembly. See Chapter 5 for more information on the kind of models are needed and how they are created.

In the final step, the individual tiles and chip I/Os are placed and connected using the flat design style again. This toplevel assembly phase also encompasses power planning, clock routing, cross talk reduction, and global optimisation. This cross talk reduction and global optimisation is done via the use of optimisation tiles. More information on tile-based assembly can be found in Chapter 2.3.

The last phase is verification to checks if all the physical, and electrical rules are implemented in the right way. And to verify if the original netlist and the final layout have the same functional behaviour. Chapter 6 shows results of using the hierarchical tile-based design methodology versus the flat methodology. Finally, in Chapter 7 conclusions are stated and recommendations for future work are given.

1.4. Related work

At this moment, two other companies are also using comparable techniques a proposed: Ammocore, and Telairity. Telairity tries to solve this by mapping designs on a fixed library of functional (already physical implemented) blocks [12]. (See Figure 2)

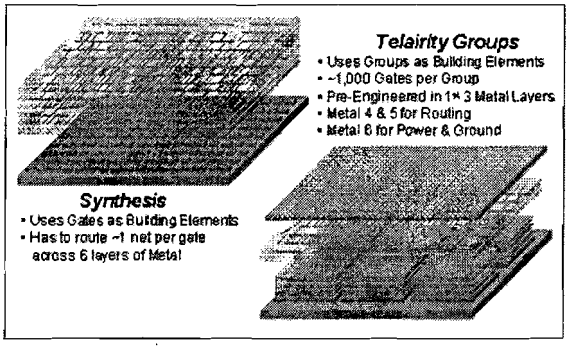

Figure 2: Telairity design style

Ammocores strategy [13][14][15], looks similar to what is going to be presented in this thesis. First they partition the design by using the Fiduccia-Mattheyses (FM) partition algorithm. After that, these partitions obtain an estimated area with a fixed height constraint for that block; this fixed height is needed allowing every block to be placed on a row-based grid. When this is done all, the toplevel constraints such as power connection, pins assignment/placement and timing are propagated down onto the block. After that, all blocks are made in parallel and incorporated back into the toplevel for optimisation and final routing. Main difference between Ammocore and this thesis' approach is the design flow, netlist partitioning, and tile implementation. The design flow that is used by Ammocore is a hierarchical block-based methodology. (See Chapter 2.2.)

2. Digital Design flow

As mentioned in Chapter 1.3, the design flow is adapted to the layout style. In this Chapter flat, hierarchical block-based and tile-based layout styles, with their associating flow are shown. At the end, pros and cons per approach are summarized.

2.1. Flat approach

The flat approach is the easiest way of designing a chip. It is relatively straightforward and consists of only a few steps. See Figure 3.

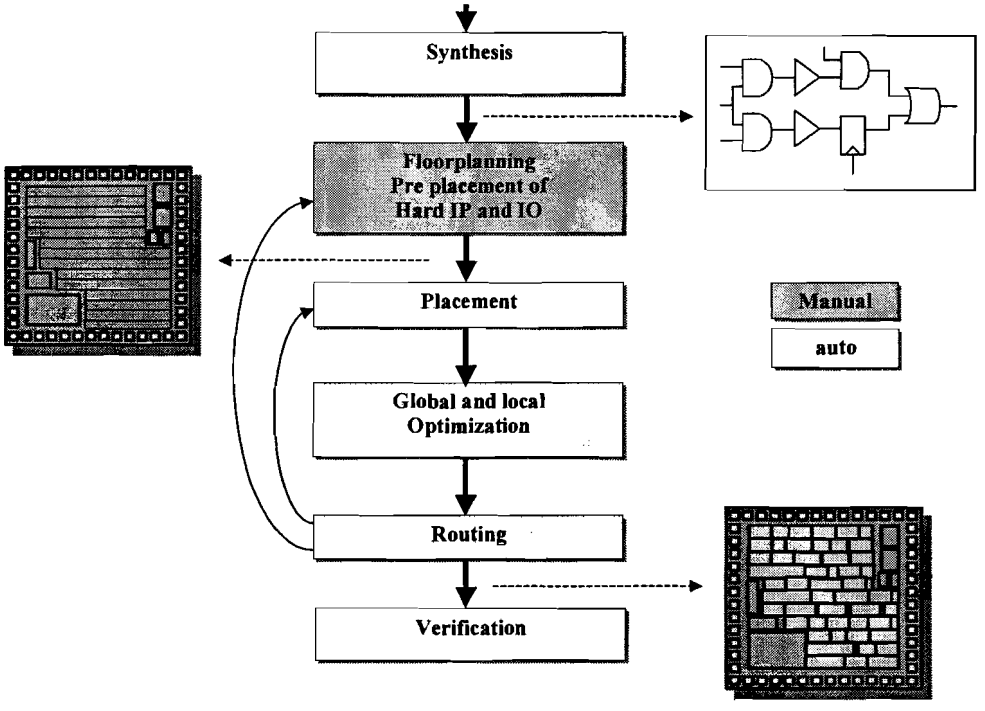

Figure 3: Flat design flow

The first step is to synthesize the Register Transfer Level (RTL) description to a gate level netlist using chip constraints such as area and/or timing. This mapping is done on standard cells, which is the level of granularity for this layout style. Besides standard cells the netlist could also contain memories, custom layouts or analogue IP blocks called hard macros. After synthesis, floorplanning has to be done. Which is the most essential step during backend design. It is manual work and requires information such as the power strategy, clock strategy and resources for place and route. After this step, the size of the floorplan is determined and with that the cost of the chip.

Next step is placement of the standard cells on rows. After that, the drive strengths of the cells are optimised. This is needed because the wire length estimation that was used during synthesis was based on a statistical model. At this point in the flow the Steiner

tree calculation gives a better estimation. The design can now be routed such that all the connections between cells and /or macros are made. Next step is design extraction to allow verification of power, signal integrity and timing. If needed the violating wires can be rerouted. If this did not solve the problem, the information can also be fed back higher into the design cycle hoping that with this extra information the flow will converge to design closure. The last step is to check the functionality of the layout against the original description.

2.2. Hierarchical block based approach

The hierarchical block approach has a lot of similarities with the flat approach. The flat approach is used for each block in the implementation phase. See Figure 4. Besides this part the hierarchical flow also has toplevel flow containing a planning, assembly and verification step. In the planning step: partitioning, time budgeting, synthesis and floorplanning are done, to start the divide and conquer strategy.

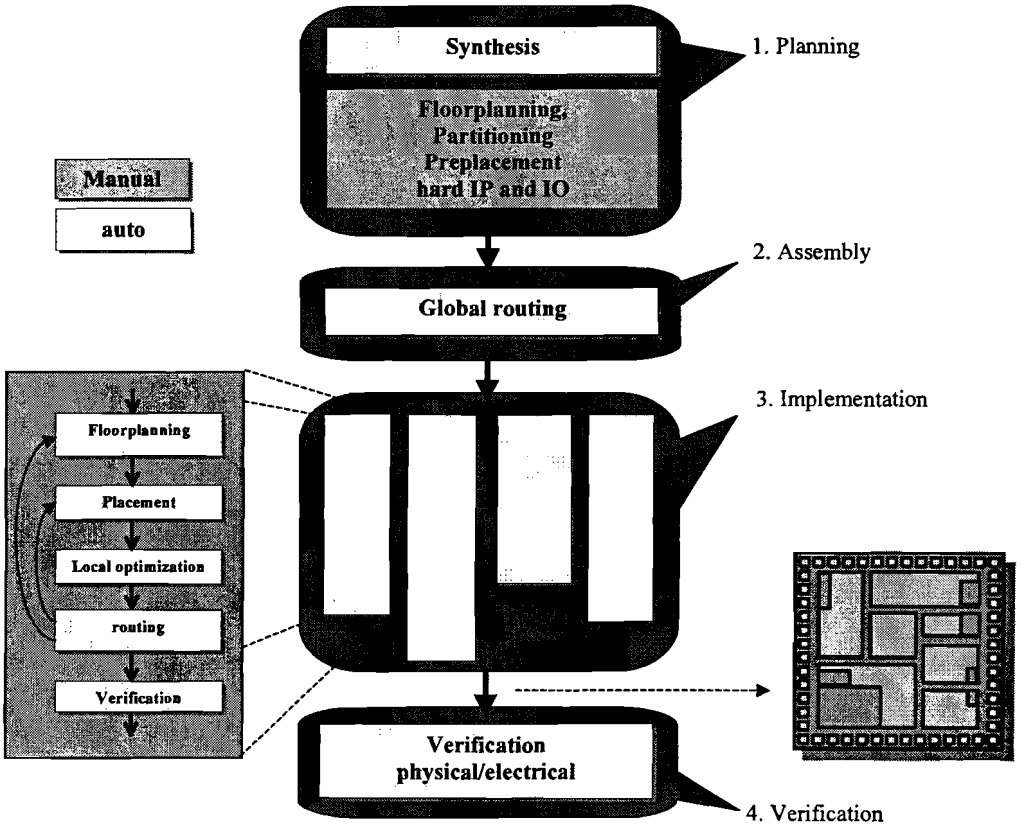

Figure 4: Hierarchical block-based design flow

The common approach for partitioning is to make the partitions as big as possible, preferable on the edge of what CAD tools can handle flat. This limit is about 1.2 Million library objects; in basic gates this would be around 5 Million.

Before floorplanning can start, estimations have to be done for each partition on area, power, and timing. The estimation of the design area for each partition is the most difficult part, because this needs accurate models of all design steps. If the estimations are not accurate enough the total area could be too large or too small, leading to wasted area or to no solution at all. In both cases the floorplan has to be redone. This means that some or all the partitions have to be done over again.

During floorplanning each partition obtains a shape and position within the total design area. To guarantee minimum influence between global routing and local routing, channels for global routing are reserved between the partitions. The shape and location of these channels are based on the number of connections that run through a certain area. After that, the pins of each partition get assigned to its boundary. To verify this assignment, toplevel is routed and for each of the partitions the boundary constraints and netlist is implemented using the flat layout approach.

The flat layout part of the hierarchical is extended with an additional step for generating abstract views needed for toplevel design. The generated views resemble the standard cell views. In which timing, power, signal integrity and physical layout are modelled in a compact way, such that it will lead to an information reduction for toplevel. Only the local information that is important for the global view is incorporated.

The blocks are read back and toplevel layout is ready. At the end, everything is verified on toplevel. This verification can be done hierarchical or flat. Normally this is done flat because of the CAD tools and accuracy. But this is not a necessity.

2.3. Hierarchical tile-based approach

The tile-based method is a hybrid form based on small-standardized blocks, which are called tiles. The implementation of the tiles should be in such a way that it avoids problems for toplevel, block level, or between both levels. Each Tile represents a small part of the complete netlist and can be a unique part or a repetitive part. Since most physical design problems are related to distance, the most important questions for this approach will be on the size and the physical implementation of the tiles. For determining the size of a tile some parameters could be used such as design hierarchy, timing, cross talk, and power grid constraints. More on this topic will be explained in Chapter 3.

The tile-based flow starts with a hierarchical synthesis followed by an analysis. (See Figure 7.) For all hierarchical levels parameters are calculated and together with heuristics and physical design constraints the proper clusters are chosen. After that, constraints per cluster are derived and each cluster is flattened and synthesized again for a better and more accurate result. The clusters are then implemented in a standardized way to form a new library of standard tiles. More detailed information can be found in Chapter 4. Besides these tiles, also specific optimisation tiles are needed for global timing improvement. (See Figure 5.) When tile A is communicating with tile C the wire is routed over a certain distance and has to be buffered to meet the toplevel timing or cross-talk constraints. So the best location for this buffering has to be found, for example location B. Inserting this buffer can be done into an already designed tile or in a special to create optimisation tile or on a different placement grid for optimisation. Inserting in a

designed tile can be difficult because of tile utilisation and timing influence. The use of a second placement grid is not that easy because then also power and ground has to be routed on that grid. And this would complicate the flat design approach on toplevel. The use of special optimisation tiles influences the flow the least.

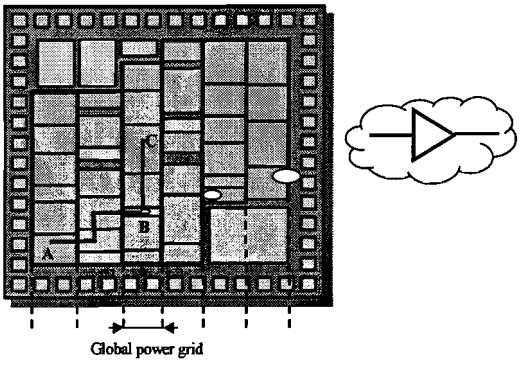

Figure 5: Toplevel buffer placement

To be able to solve global timing issues, a toplevel synthesis with a wire load model (WLM) is done up front, to see which buffers and inverters have to be created as tiles. With this new library of standard tiles and optimisation tiles, the total chip can be designed using the flat design style. Major reason is that the new library looks and behaves as a standard cell library. So that the flat design approach can be reused at toplevel. However, hard macros and analogue IP (Intellectual Property) still have to be placed manually. Note that there is a possibility to convert hard macros into tiles as well. This method is close to optimal if the area range of the macros and clusters are similar. However for the chips that are designed at this moment, this is not the case. Most of the designs have a few big-shared memories on chip instead of many small-distributed memories. For the future, this number of small memories will increase because of bandwidth and power requirements. Transporting data is more and more power inefficient in newer technologies because of the wire capacitance increase. Wires are dominating more and more.

After placing the hard macros, the rows are created, and placement of the tiles can be done. One of the interesting possibilities is to see if the power grid on toplevel can be reused as a parameter for the width of a tile. (See Figure 5) Each tile will then have a fixed width and a variable height. (See Figure 6) In standard cells this is somewhat the same but then the other way around, fixed height and variable width.

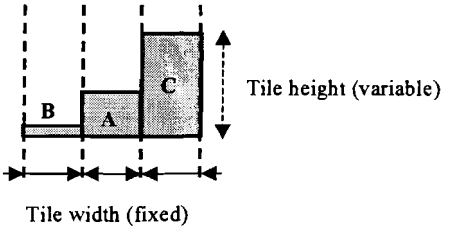

Figure 6: Tile size variation

The lengths of the global wires are estimated with this placement and optimisation can be done. For the timing optimisation and clock-trees synthesis the buffer and inverter tiles can be used. Each design tile has its own clock-tree if that was needed, which is modelled in a timing library file. This is done via an insertion delay and a skew value. The toplevel clock-tree task is to unbalance the clock pins of the block in such a way that the endpoints of the blocks are balanced over the complete chip.

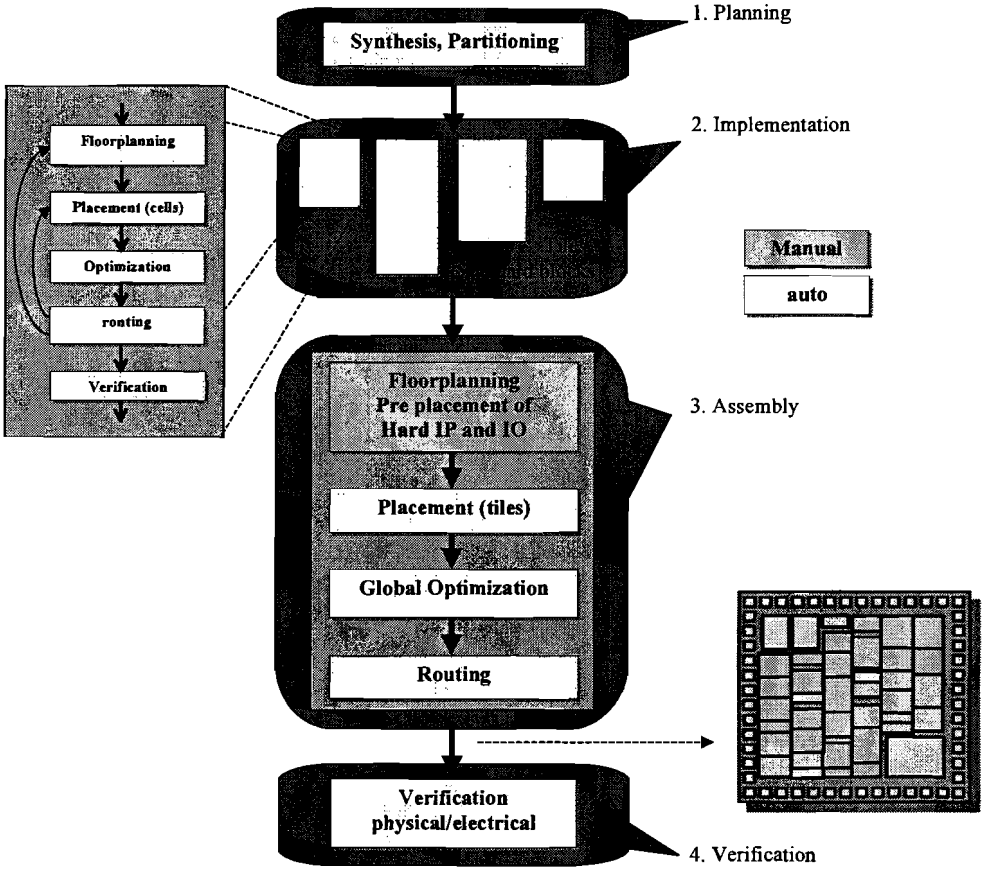

Figure 7: Hierarchical tile-based design flow

The toplevel routing is done over the tiles by using the top most routing layers that are not used inside the tile. See Figure 8. If necessary also the rest of the routing layers can be used, which means that the wires could run through the tile. As a draw back this routing could affect the internal timing of the block. To verify the toplevel the tiles do not have to be verified because they where already verified when designed. Only the connections to and from the tiles have to be verified. Also design, electrical and antenna rule checks (DRC, ERC, ARC) have to be done to guaranty manufacturability.

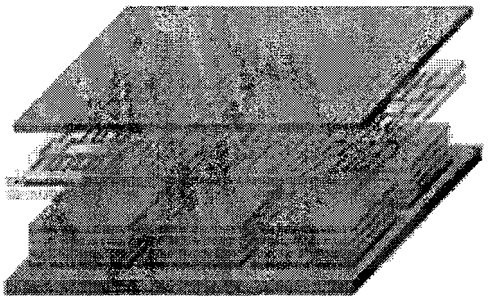

Figure 8: Toplevel routing over tiles

2.4. Design style summary

When looking at the three layout styles it can be summarized that the flat layout style is the easiest automated flow with the highest performance and density figures. The disadvantage is, that run-time increases, more than linear and sometimes even quadratic with the number of objects or design parameters. For the hierarchical layout style this is not a problem, because the netlist can be split up in smaller blocks. The disadvantage however is that the search space is decreasing. As a result, this might reduce performance or increase the area. In addition, engineering change orders (ECO) are more difficult because they could affect multiple blocks. See Table 1.

	Automation	Density / area	Timing	Capacity / run-time
Flat	+	+	+	-
Hierarchical block-based	+/-	-	-	+
Hierarchical tile-based	+	+/-	+/-	+

Table 1: Design style overview

The hierarchical tile-based design flow is a compromise of the flat and block-based flow. It takes care of the flat disadvantage and tries to solve it by using knowledge of the hierarchical block based design flow. Difference with the hierarchical block based approach is that the layout construction is bottom-up, first the tiles are made and after that toplevel. In this way, problems are not propagated down but up. The advantage is that the optimisation space is much larger. And if needed, the critical tile can be redesigned.

3. Partitioning / Clustering

The essence of partitioning is to divide a netlist into clusters/partitions in such a way that it saves runtime and/or allows better search of the design space. Before the clusters can be located some partitioning constraints are necessary. The choices for the clusters should be in such a way that it avoids electrical, and physical problems for toplevel, blocks, or between them. Some implementation issues that could help the partitioning are signal integrity, power grid structures, area, timing closure, and design hierarchy. In this Chapter some possibilities are evaluated.

3.1. Rents rule

Naveed Sherwani [6], shows that an important class of placement algorithms is based on partitioning. One of the most popular algorithms is the quadratic placement procedure. This is one of the algorithms used by Cadence. Another very popular algorithm in standard cell placement is the bisection placement procedure. Recent trends show that placement algorithms have to optimise besides the traditional area and routability, also power and signal integrity. As a result, algorithms that estimate wire lengths are becoming more important, because they could help to fix problems in the placement step instead of in the routing step.

The quality of a block-based layout is depending on the generated partitions. In [2], Rents parameter is used for analysing partitioning algorithms. To see which of the tested algorithms performed best for layout. In this paper [2], it is shown that spectral based partitioning algorithms are better than the traditional iterative methods such as the Fiduccia-Mattheyses (FM) approach. Rents parameter (p) is the exponential part of Rents rule and shows a power-law relation between T, the average number of pins/terminals per module and C, the average number of cells/blocks per module in that same cluster/partition. *Tpc* is the average number of terminals/pins per cell.

$$T = \overline{tpc} * C^P$$

Equation 1: Rents rule

In 2002, P. Christie [3] gives an equation for optimised cell placement. Which is an optimised Rents rule Equation.

$$T_{cl} = \left(\overline{tpc} - \frac{T_{top}}{C_{tot}} \right) C_{cl}^P \left(1 - \frac{C_{cl}}{C_{tot}} \right)^P + T_{top} \frac{C_{cl}}{C_{tot}}$$

Equation 2: Optimised Rents rule

T_{cl} and C_{cl} are respectively the number of terminals and the number of cells of a certain cluster. T_{top} and C_{tot} are the total number of terminals and cells of the complete design.

This final solution is a better approximation than Rents rule because it does not neglect edge effects and pins, as is shown in Figure 9. The coloured points are extracted points from the netlist; each colour gives a different starting point before growing to the edge of the netlist. The average of these coloured points is the red line (curved line). The dashed line is Rents rule and the dotted line is the optimised Rents rule. It can be seen that Rents rule for the complete netlist is too optimistic. Main reason for this is that the netlist is not homogeneous as expected and that the netlist is not infinite in size. When assumed that the netlist would be infinite, Rents rule could be extracted. This also holds for Equation 2, when C_{tot} goes to infinite this equation is equal to Rents rule. Leading to the blue dashed line.

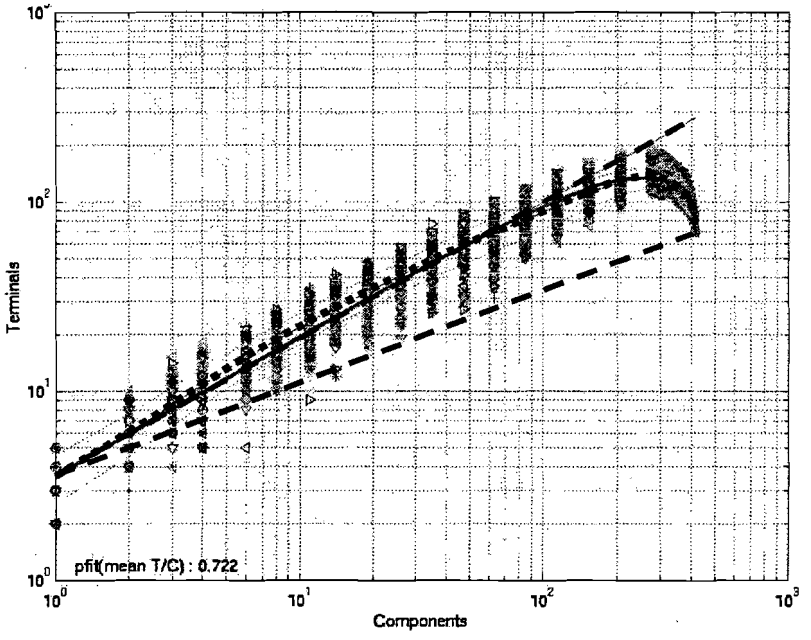

Figure 9: Industry netlist example for Rent parameter extraction.

When using the optimised Rents rule on a hierarchical netlist, for each hierarchical level the Rent parameter (p) can be calculated. It could also be done via Rents rule but it would not be so accurate. The only problem with the optimised formula is that it cannot calculate the toplevel Rent Parameter. It can only be fitted to the average extracted curve. Meaning that each cell in the netlist is used as a starting point for growing to the edge. For a simple netlist this can be done.

Besides that Rents parameter can also be used as an indictor for placement, the lower the Rent value of a cluster the closer the cells are placed together. This can be seen in the combination of Table 2 and Figure 10. For a first test, 5 clusters where selected from the Philips Network-on-Chip (NoC) router design [17] to give an impression. When looking at the Rent parameter it seems that the lower the Rent parameter the closer the cluster is placed together. Each region is a cluster in the design hierarchy.

	Name	Rent exponent	Rent improved
P_0	Switch	0.64	0.66
P_1	Fastq $_{(3\text{-}0\text{-}8)}$	0.19	0.25
P_2	Ftfifo $_{(34\ by\ 24)}$	0.37	0.27
P_3	Ftfifo $_{(34\ by\ 3)}$	0.49	0.47
P_4	hpu	0.60	0.58
$P_{background}$	toplevel	0.46	-

Table 2: Rent exponents per selected cluster

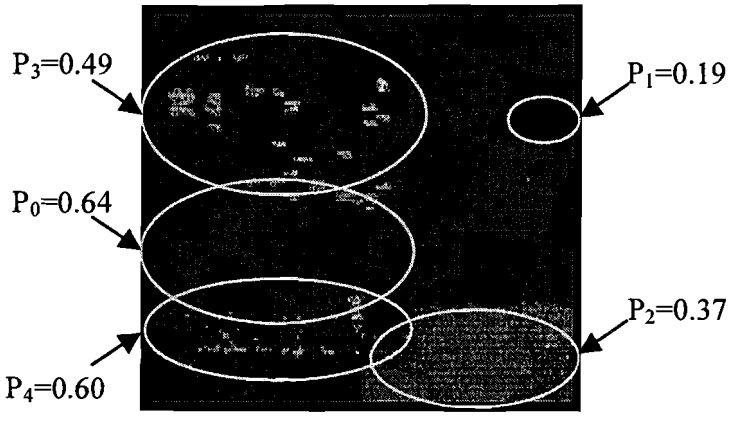

Figure 10: Rdt_router_top_a6_c8 NoC design

Some other experiments on different kind of designs where done and it seems that this method of cross-linking is possible but not perfect. However in general, this method might not be true for all cases, because it is highly dependent on the placement algorithm. In our case qplace from cadence was used. Based on these experiences the method of search, create, and design is introduced. So that the designed clusters can be used to design toplevel, to get almost the same placement result as doing it completely flat. Disadvantage here is that this method could cost some extra area, because the complete design could not be broken into clusters that have a rent exponent lower than toplevel. To minimize this disadvantage these clusters should be as small as possible. As conclusion it can be seen that the chance is higher that the cells of a certain cluster will be placed close together if the Rent parameter is low but there are exception.

3.2. Signal integrity constraint

Signal integrity is getting more and more important when feature sizes decrease. The decrease in wire spacing and wire width, gives an increase in wire height because of the current density. This leads to smaller bottom capacitances but bigger cross coupling capacitances. To control the signal integrity the length of a cluster wire should not be longer than a certain maximum length. This length also gives an indication on the size of the cluster. A possible first order cross talk calculation is to look at the cross coupling capacitance in relation to the ground capacitance of 2 wires floating above substrate.

$$X_{talk} = \frac{Z_{victim}}{Z_{cross} + Z_{victim}},$$

$$Z_{cross} = \frac{1}{2\pi f C_{cross}},$$

$$Z_{victim} = X_{Ctot} \,//\, X_{Cload} \,//\, R_{driver},$$

$$Z_{victim} = \frac{R_{driver}}{1 + 2\pi f R_{driver}(C_{load} + C_{tot})},$$

$$X_{talk} = \frac{1}{1 + \frac{1}{2\pi f R_{driver} C_{cross}} + \frac{C_{load} + C_{tot}}{C_{cross}}},$$

$$C_{cross} = \alpha * C_{tot},$$

$$C_{load} = fanout * C_{in}$$

Equation 3: Cross talk

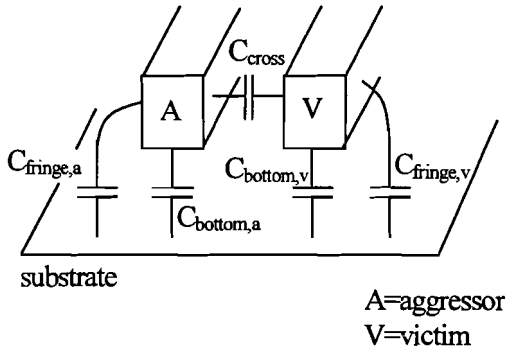

Figure 11: Wire configuration

The X_{talk} variable is the cross coupling value between aggressor and victim wire. In the manual of Celtic, a tool for cross talk analysis, the default maximum value for X_{talk} is 20%. This means that one-fifth of aggressor voltage is allowed to be injected on the victim. C_{cross} is the cross coupling capacitance between the two wires, C_{tot} is the sum capacitance of the bottom, fringe and cross capacitance. C_{in} is the input capacitance of a cell.

Based on Equation 3 and the fact that capacitance grows linear with the distance, the maximum wire length can be calculated for a certain technology to prevent cross talk problems between global wires (toplevel wires) and/or cluster wires. Routing on a larger pitch can solve cross talk problems on toplevel. The average number of terminals per wire minus one can replace the fanout variable. To find the average number of terminals per wire the netlist has to be analysed. See Equation 4. T_{cl}^{io} is the number of terminals for cluster *(cl)* and tpn_{cl} is the average number of terminal per net for that cluster.

$$\overline{tpn}_{cl} = \frac{T_{cl}^{io} + \overline{tpc}_{cl} * C_{cl}}{N_{nets}},$$

$$fanout = \left(\overline{tpn}_{cl} - 1\right)$$

Equation 4: Average terminals per net

With the given information, the maximum wire length for a certain technology can be calculated depending on the aggressor signal slew (τ_{slew}). Equation 5 shows the relation, for a two-wire configuration, between aggressor slew and the length of the wire for a certain X_{talk}. It is derived from Equation 3 and the fact that wire capacitance grows linear

with the wire length.

$$l = \frac{\tau_{slew}}{2\pi R_{driver}\left[C_{cross}\left(\frac{1}{X_{talk}}-1\right)-C_{tot}\right]} + \frac{C_{load}}{C_{cross}\left(\frac{1}{X_{talk}}-1\right)-C_{tot}}$$

Equation 5: Slew versus wire length for certain X_{talk}

Given the following parameters (TSMC 0.12µ process, minimum width and minimum spacing of the wires metal {2,3,4,5,6}, *Ccross*=111fF, *Ctot*=125fF, *Cin*=9fF, *fanout*=3, *Rdriver*=200Ω, *Xtalk*=20%.), Figure 12 shows the relation between aggressor signal slew and the wire length. When taking the default slew of the library (τ_{slew}=0.27ns) and the least drive-strength for the victim (ivx05 R_{driver} = 1k8Ω) the maximum wire length would be 104µm. When calculating the wire length in the same situation but with the fastest slew (τ_{slew}=15ps) the length is 33µm.

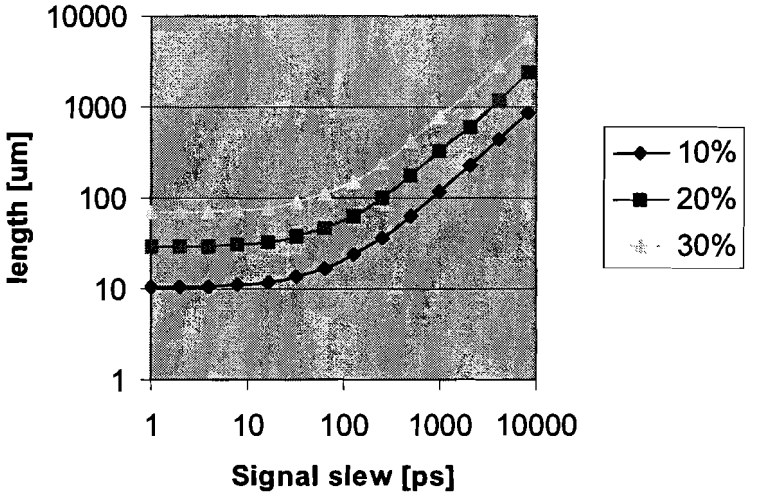

Figure 12: Signal slew versus wire length (log-log scale)

3.3. Power grid constraint

Regarding the power grid constraints for a tile, Philips EMC guidelines [4] are used. They provide rules for supply bounce and global power dimensions. For a TSMC 0.12µ process, the EMC guidelines state that each standard cell power rail can supply a maximum length of 164µm. Meaning that this power rail can supply an N number of cells that in total have a width of 164µm. Beyond this 164µm the power rail has to be stitched to the global power grid. (See Figure 13.) So this length can be used as a width constraint for the tile size. In this way the toplevel power grid can be used as placement grid for the tiles.

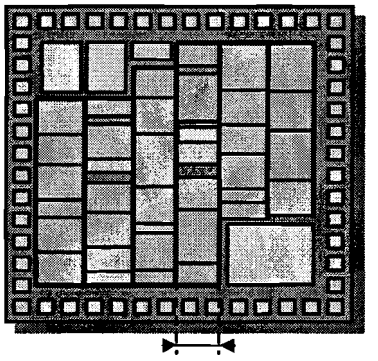

Figure 13: global power grid

3.4. Area ratio constraint

When searching for clusters the area ratio of the smallest and biggest cluster should be as small as possible because some placement algorithms are using move-based procedures for solving the placement problem. Additionally small clusters have the advantage that the use of a wire-load model is much more accurate. The ratio of the wire lengths is then much smaller. At least our partitioning algorithm should try to find a solution that is in the same range as the standard cell library. In this case the tools are performing somewhat the same on standard cell level as on tile level. When looking at different standard cell libraries the following ratios where found between the smallest and largest cell.

Process	.35µ	.25µ	.18µ	.12µ	90n
Ratio	15.5	19.5	29	40	52.5

Table 3: Maximum standard cell area ratios in CMOS libraries

3.5. Clustering algorithm

Partitioning is a very complex topic with lots of room for optimisation. However, since this was not the main focus of my thesis I decided to use a pragmatic partitioning rule. Mainly based on technology parameters and the design hierarchy. From Chapter 3.2 it can be seen that the maximum length of the cluster in relation to signal integrity is 104µm for a .12µ process. Chapter 3.3 shows a maximum tile width of 164µm because of the power rail within the standard cell library. This means that the minimum of these two values determines the tile width. This leads to a maximum area of $10816\mu m^2$. When dividing this maximum area through the basic gate area for that technology, the maximum number of gates per tile should be less then 1300. From Chapter 3.4 it can be seen that the minimum number of gates per tile should be at least 25 basic gates, which is around 7 standard cells. Table 4 shows for different designs the total number of leaf clusters, and the number of basic gates for the largest and smallest leaf cluster. A leaf cluster is defined as a group of basic cells that do not contain any children, only library components.

Design name	CMOS Technology [µm]	Number of leaf clusters [#]	Largest leaf cluster [basic gates]	Smallest leaf cluster [basic gates]
rdt_demod	0.12	252	1289	1
rdt_aeonic_core	0.12	11155	1184	0
rdt_xetal_pparray	0.18	24	436	4

Table 4: Leaf cluster overview

When partitioning the netlist, only the existing design hierarchy is used. Meaning that no partitioning is done within a leaf cluster. Current status is that the Rent parameter of the toplevel netlist is used to select all the hierarchical levels that have a lower rent parameter than toplevel. After that the maximum numbers of cells is used to look which of the selected clusters still comply with the both rules (Rent, maximum gates per tile). Finally the list is checked to see which combinations of clusters form the complete netlist again. If the result of this combination check fails, the Rent parameter of the toplevel is increased and again the search process is started until a result is found. When multiple solutions are possible with list the one that needs the lowest number of clusters is selected.

4. Tile implementation

After dividing the netlist into clusters, these clusters have to be implemented physically to form a library of standard tiles. In this way toplevel floorplanning becomes a normal standard cell place and route problem. However, some boundary constraints have to be taking into account, for making this methodology feasible. Looking at the standard cell library, some specific properties can be observed that are important. The most important properties are: area, fixed height of the building blocks, IO pin locations, and locations for power connections. In normal hierarchical block-based designs methodologies these properties are the most difficult ones to determine. Based on choices, the design will become routable or not. In the next Chapters these basic properties are explained in more detail and how they are estimated and implemented for the tile-based approach.

4.1. Timing propagation

Before an area estimate can be done the clusters have to be synthesized. For that, the toplevel timing constraints has to be propagated down. This means that the clock period and IO timing have to be divided over the clusters depending on cluster area and logic depth of paths and wire load models. One technique to do this is to place register elements at the output of each tile[1]. For the tile-based method we used the time budgeting option of cadence. Background information on this command was not available.

4.2. Area estimation

A layout area is always described as a rectangle. The width divided by the height is the aspect ratio. Aspect ratios vary between 2 and ½, depending on the layout package. For most of the chips, the aspect ratio is around one, because chip packages are often square. For our cluster we also have to estimate the best aspect ratio. Shmuel Wimer et.al. [9] and Ralph Otten [10] present what the best aspect ratios for blocks are in a non-slicing and a slicing floorplan. For our test case the aspect ratio of the tiles is chosen to be on average, square. The column width or row height can then be calculated with Equation 6. The *Rowutil* variable is the utilization factor for that cluster, to reserve area that is needed for optimisation, clock-tree generation, decoupling and routability. An interesting way for calculating the row utilization is to combine the average number of pins and nets per unit of area. Also the use of the wire length distribution model based on Rents rule [7][8] gives extra information. How to use this is however still subject of further investigations.

[1] The disadvantage is a possible extra cycle and the way designers have to write their RTL description. It is even not always possible to do registered output because of test infrastructure. This forces some outputs to have multiple functions.

$$H_{row} = W_{column} = \sqrt{\frac{1}{N_{cluster}} \sum_{n=1}^{N_{cluster}} A_n * Rowutil_n}$$

Equation 6: Row height / Column width

In our case, the column width has to be calculated instead of the row height, because the clusters are placed on vertical rows (columns). The design of the cluster is done flat. Therefore, inside the block the rows are horizontal because of the preferred routing direction of metal1. These metal1 stripes are needed for connecting power and ground to the standard cells. (See Figure 16)

4.3. Pin placement

An issue that arises due to the tile-based methodology is the pin locations within the tile. In a flat approach, these pins are somewhere on the wire but they do not have any physical location. When making a tile, the pins should be placed on positions where they have minimal influence on placement and routing of the tile. Besides that the influence of the internal wires connected to those pins should be negligible when looking at the toplevel wires. Some experiments were done to see how these pins influence placement and routing. In the left pictures of Figure 14 all the pins are placed on the top boundary, and in the right picture all the pins are placed on top of the connecting cell. The colour in both cases gives the Rent parameter calculated when starting at that cell position. It can be observed as a routing congestion map for the placed cluster.

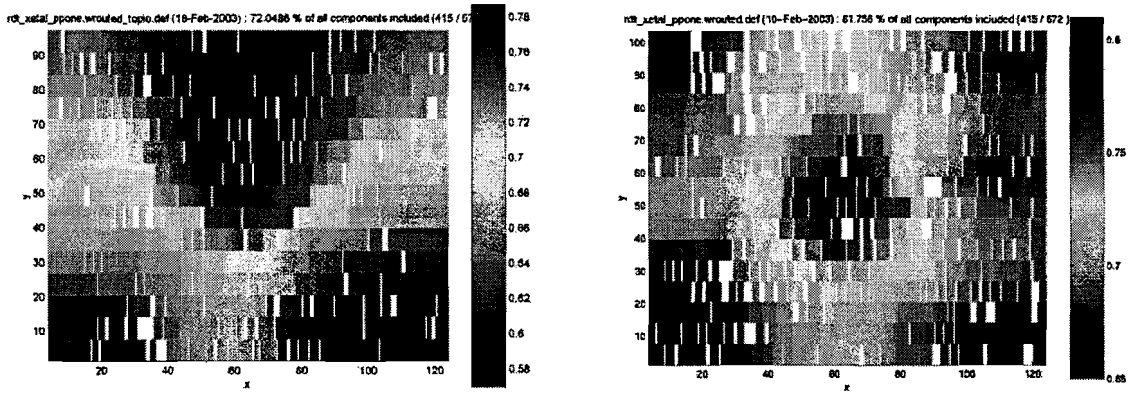

Figure 14: Pin placement influence.

It can be seen that putting the pin on top of the connected standard cell gives an easier routing solution. This method is similar to what is done in standard cells. It also eases modelling of timing for that pin.

The pins are always placed on the highest metal layer that is used for designing the tile. At this moment, 3 layers are used for routing within a tile. In this way, it is always

possible to connect the pins of the tile. Important for this method is that all the pins are connected to only one cell.[2] If this would not be the case the complete wire is becoming a pin and will have more influence on the toplevel timing. (See Figure 15) The grey lines are toplevel wires, black lines are internal wires, black dots are standard cell ports and the grey dots are ports from inserted buffers. Figure 15a shows that the complete tile wire becomes a pin for toplevel routing. In this way toplevel routing will connect somewhere on the wire to try to minimize the toplevel wiring. In this case the influence of the internal wires are dominating the total wire. In Figure 15b all the internal wires with more than one connection have an extra buffer for disconnecting the pin from the wire. In this way, the influence of internal wire is minimal in relation to the toplevel wire. The internal wire for the pin connection is very small in relation to the toplevel global wire. This also makes modelling of the tile timing easier; the information of the last cell can be used directly.

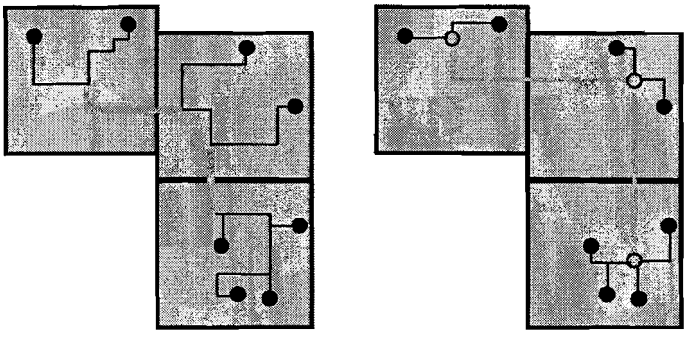

Figure 15: Pin influence

The worst pin location is in the centre of the tile, due to the fact that the drive strength should be big enough to drive at least a wire that is going to the boundary of the tile. At the boundary the wire will be buffered. For the worst-case situation the driver could be the smallest cell of the library. This means that this cell has to drive a certain input load (worst-case the biggest of the library) and the wire that is routed to this cell. This wire is then at least half a tile length long. Equation 7 can be used as a first order estimate to calculate how long such wire is allowed to be.

$$l_{wire} = \frac{C_{driver\,max} - C_{load\,input}}{C_{wire/mm}}$$

Equation 7: Maximum distance to tile boundary constraint.

For a TSMC 0.12µ process, minimum width and minimum spacing using metal3, 4, 5 or 6, (C_{embed}=262.1fF/mm, $C_{in,ivx18}$=102.8fF, $C_{driver,ivx05}$=120.2fF.) a maximum wire length of

[2] If this is not the case inserting a buffer can solve this. When multiple pins are connected to one cell, the pins will be placed with a certain minimum distance (DRC rule). (See Figure 16.)

66µm is calculated. From the timing library a delay of about 1.2ns was then found for this ivx05 cell.

4.4. Power strategy

The power and ground connections of a tile are similar to that of a standard cell. Meaning that power and ground stripes are placed on the boundary, in the column directions. Within cadence the connections for power and ground in rows are made via a special command called *sroute followpins*. Meaning that following the direction of the pins connects the power. In the tile implementation the power and ground wires/pins are placed vertical, on the left and right sides of the rows. (See Figure 16.)

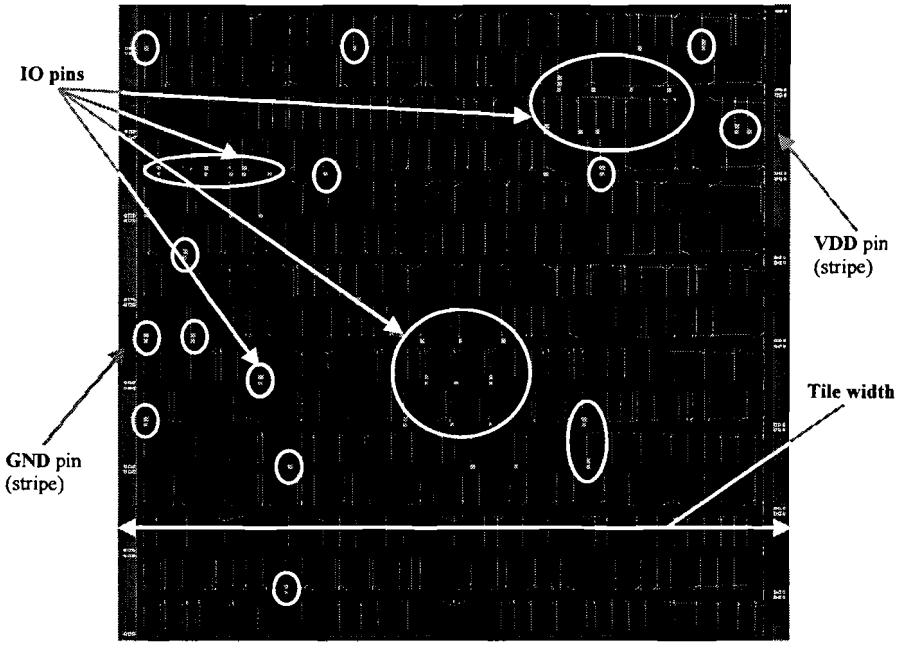

Figure 16: Abstract view of rdt_xetal_pplogic cluster

In this way, the designed cluster does not have to be rotated by 90 degrees before it can be placed on rows again. Although it could be done it can influence yield, since not all structures/polygons are laid out in their preferred directions.[5] Besides the impact on yield, the use of rotated clusters, influences the routing resources. To solve this problem the library package file has to be changed for each level of the hierarchical design flow methodology, which is cumbersome.

The power and ground wires are in metal 2 or metal 4 depending on the total area of the chip. Metal 4 is the better choice because the resistance is lower than that of metal2. A lower resistance gives a lower supply dip. But when using metal4, the number of obstructions within the block is larger, since metal3 and metal2 are also blocked. The width of these metal 4 or metal 2 stripes is dependent on the number of cells. The width is also dependent on the maximum allowed supply bounce on these power tracks.

Within Philips, EMC guidelines [4] provide rules for supply bounce and global power dimensions. The width of the global power grid can be calculated with the formula from appendix B of the EMC guidelines, for a TSMC 0.12µ process.

For a block of 750µm height, signal activity of $\alpha.f_{clock}$=60MHz and voltage drop of ΔV=10mV the width of the global wire should be at least 15µm. For the tiles this means that the width of power tracks within, is half of the calculated global wire width. When using the flip and abut rows at floorplan level two rows together form the global width again.

4.5. Clock strategy

The clock strategy is the same for all levels, and based on an H tree for minimum skew or zero skew. Each tree will be modelled as an insertion delay in the timing model of the block; by doing this toplevel clock tree synthesis is just as doing clock insertion on block level. As a consequence not all the cluster clock trees have to be matched; this can be solved on toplevel or one hierarchical level higher.

4.6. Timing optimisation

On toplevel, optimisation is carried out to get timing closure. Main reason for this is the fact that a WLM model was used, which is not accurate enough for large blocks/areas. This means that extra buffers have to be inserted between the clusters. Because we only have one row description, the buffers cells have to be implemented in a new buffer tile for toplevel. Therefore we have to make an optimisation tile with one row to place the buffers or inverters cells. A better solution would be not to place these optimisation cells separately but to put these cells directly between the cluster rows. It is also possible to place more than one buffer or inverter in the special optimisation cell. But if these extra cells are not used, the inputs have to be connected to a constant since leaving them unconnected violates design rules. An other solution could be to flatten the layout after the tiles are placed and to perform an optimisation run on this layout.

5. Model generation

Models are needed for toplevel implementation and they need to be equivalent to standard cells, to enable reuse of the standard cell flow on toplevel. Besides that, the models should be as accurate as possible and only leave out information that is not needed at toplevel. The minimum views that are needed are the layout/physical, timing, and functional view.

5.1. Layout model

The easiest way of modelling tiles is to make a detailed abstract. This detailed abstract models all the routing, pins and obstructions of the standard cells, leading to a lot of information, which can be reduced significantly. For instances if two wires in a cluster are routed on minimum distance, they can be merged into one obstruction. Also routing area that cannot be used on toplevel can be merged into one polygon. Because the pins of the cluster are in metal3, most of the toplevel wiring will also be in the higher metal layers, metal1 could be blocked in total. This saves information and processing time because the obstruction can be modelled into one polygon.

5.2. Timing model

Inserting metal fills, to improve manufacturability, on locations that are valid for global routing, is a way of modelling global wires making timing verification/modelling of the tile more accurate. However, this insertion should only be done for timing extraction (DSPF or rSPF). After metal filling, the wires can be extracted into a rSPF model. RSPF stands for reduced Standard Parasitic Format. With this rSPF and the timing models of the standard cells, the timing of a tile can be verified. All paths that are connected to pins have to be modelled for toplevel. The internal paths that are not connected to external pins are verified and they do not need to be modelled. (See Figure 17)

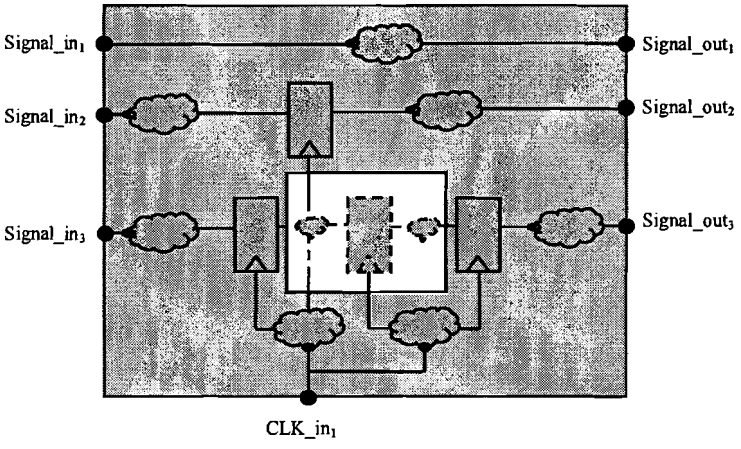

Figure 17: Timing model

But for paths that are connected to IO pins, these still have to be modelled for different loads and different input slews and for different Process, Voltage and Temperature (PVT) corners. One of the possible solutions is to extend the set-up and hold times of the register elements to the boundary. For the combinatorial paths, this is not possible and those paths have to be modelled as a single element with the proper function, delay and drive strength. Besides these IO paths, also the clock has to be modelled for toplevel clock-tree insertion. This means that the clock tree in this cluster has to be modelled as an insertion delay with a certain deviation or minimum insertion delay together with a skew value.

6. Design Results

Because the tile-based method can give the flat layout style a new impulse, the tile-based method is compared to the flat layout style on multiple topics, such as run-time, area, wire- length, capacitance and timing closure. The most important topics are area, run time, and timing closure. For timing closure it is also important to look at the histogram of all the paths, to get a better understanding of the result. To measure all these values, the following approach is used: First the tile-based method is used to create a starting point for the comparison. This point is then used to create a flat result. After that the netlist is again made with the flat approach but then in the smallest possible area. The tools and computer infrastructure used for both methods are the same, meaning same tool versions and same computer queues. Unfortunately, the number of CPUs used per computer could not be fixed.

6.1. SIMD processor array

The first design that was made was a Single Instruction Multiple Data (SIMD) processor array. This array is part of a design project called Xetal within Philips Research [16]. The array is made out of 320 processors and 1 global controller all running at 16.7MHz. The design contains in total 120K CMOS18 standard cells, equal to about 500K basic gates. When using the analysed parameters before and after synthesis the processor and controller where found as tiles. (See Figure 18.)

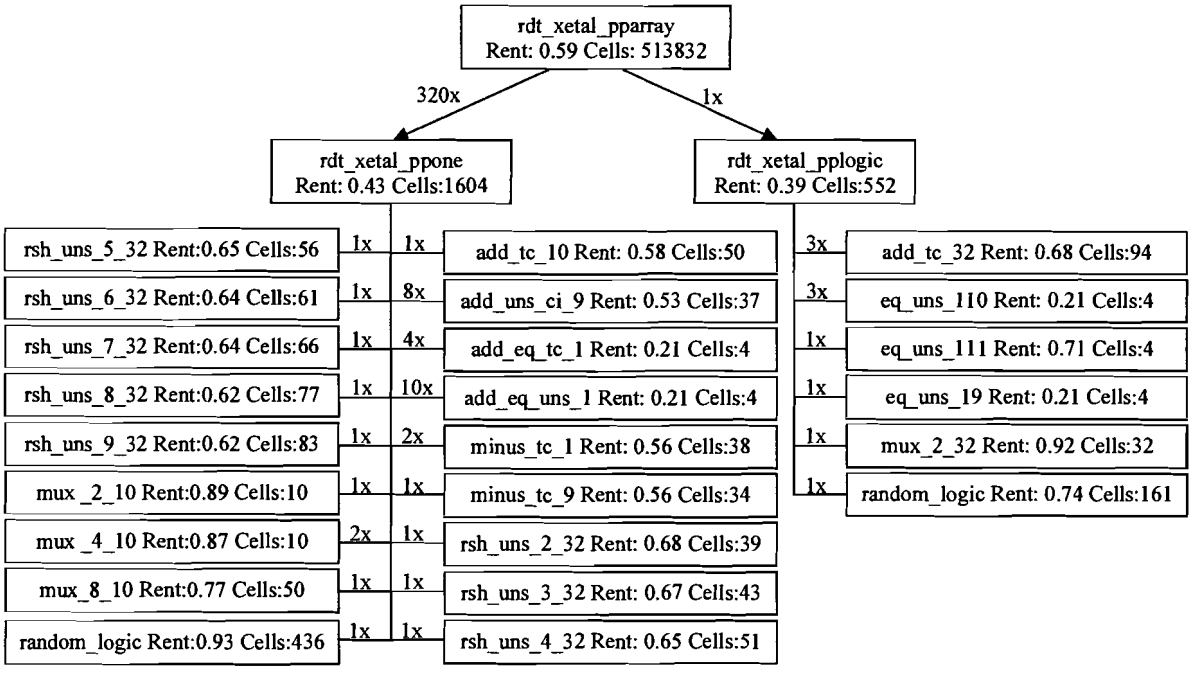

Figure 18: Hierarchy processor array

After synthesis of the toplevel also four extra blocks had to be made for optimisation and clock tree synthesis. Figure 19 and Figure 20 respectively show the flat and tile-based layout and both are plotted on the same scale.

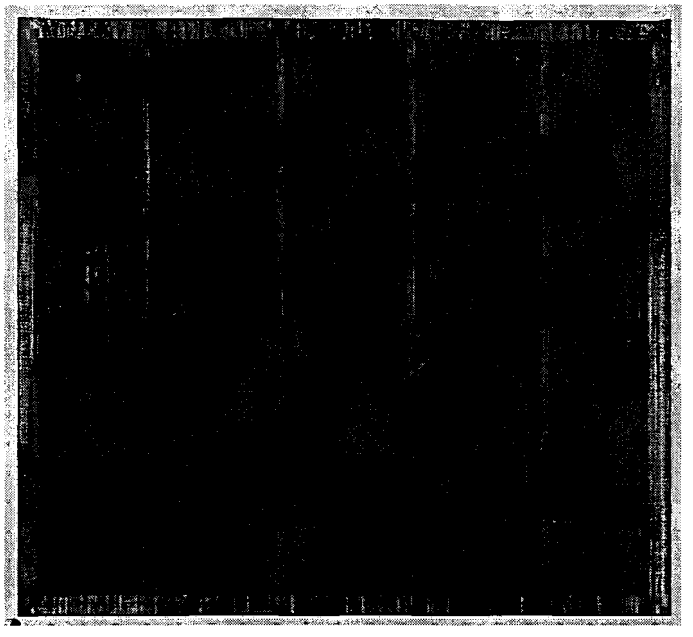

Figure 19: Flat processor array layout

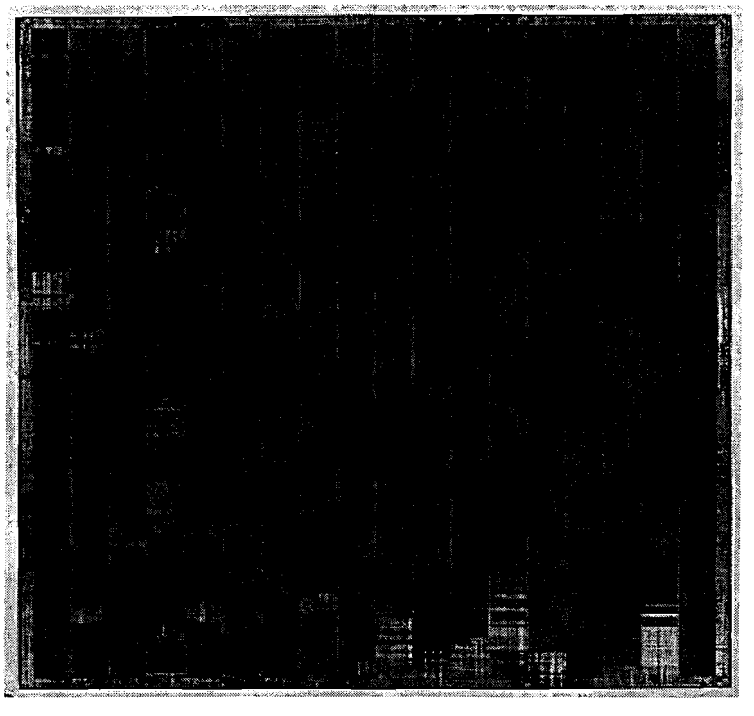

Figure 20: Tile-based processor array layout

For comparing the tile-based with the flat layout approach on run-time there are multiple possibilities depending on the computer infrastructure. In Table 5, the run-time is calculated depending on the number of computers and tiles. In case 2 and 4 each

rdt_xetal_ppone processor is made unique. In case 3 all the *rdt_xetal_ppone* processors are physical identical. Case 1 is flat in minimum area and case 5 is flat using the tile-based area.

Case	Layout style	Area [mm^2]	Area [%]	Tiles #	Computers #	Cpu-time [s]	[h]
1	Flat	4.23	100	120K	1	24990	6.94
2	Tile-based	5.21	123	325	325	7416	2.06
3	Tile-based	5.21	123	6	1	7834	2.18
4	Tile-based	5.21	123	325	1	160724	44.65
5	Flat	5.29	125	120K	1	22052	6.13

Table 5: Cpu-times overview

When designing with tiles instead of flat the cpu-time is about 7 times worse, but by using the repetitive properties of the ppone or by using more computer resources the gain can be turned around to 3 times better under the same conditions. Most of the time in the tile-based layout style is used for toplevel, the design of the blocks is done in around 450 seconds. The rdt_xetal_ppone takes about 480 seconds and 420 seconds for the rdt_xetal_pplogic. These 450 seconds are divided into two major parts, 250 seconds is needed for implementation and the rest is taken by the verification and model generation. Toplevel implementation takes the other 7000 seconds.

From Table 5 it can be seen that with the same timing result the tile-based layout needs 23% more area. Most of this extra area is caused by the way the toplevel optimisation blocks are made. Each optimisation block is made out of one row containing only one buffer or inverter and filled up with decoupling cells or filler cells. This means that the row utilization of that optimisation block is very low, resulting in a lot of area loss. The advantage of doing optimisation blocks, is that you only need one row description for toplevel. The connections of the power pins is then done automatic via the *sroute follow pins* command. The 1677 needed optimisation blocks at toplevel cost about 1.3mm^2. If these blocks could be made with minimum overhead and still fit in an automated flow, maximal area reduction to 1.26mm^2 could be achieved. This is only true when there are enough routing resources on top of the tiles.

Case		1	3	5
areautil	[%]	80	65	63
Slack [setup_wc, path 1]	[ns]	10.86	10.88	13.19
Slack [setup_wc, path 2]	[ns]	28.92	33.19	30.87

Table 6: Timing characteristics.

When extracting timing numbers for set-up and hold times some strange result where seen. The critical path of the design is a wired OR over the 320 processors. This means that the critical path in both layout styles do not differ that much. But if you take a look at the setup slack histogram in Figure 21 of both cases (left flat and right tile-based) the difference in timing is better visible. The tile-based method gives a 14% better result than case 1 and a 7% better result than case 5. In overall the total slack in the tile-based method is much higher. This slack could be sacrificed for reducing the total area.

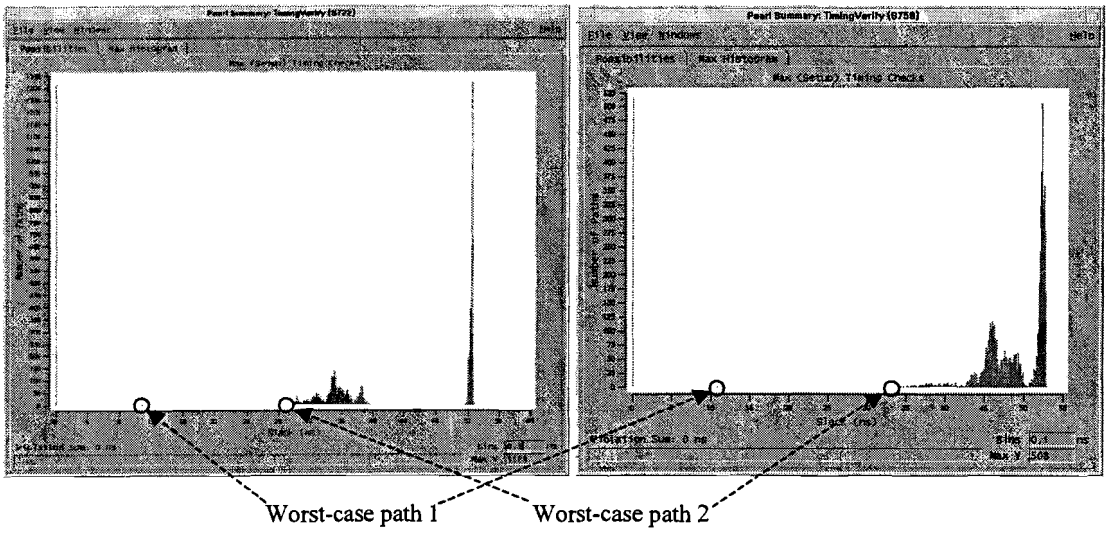

Figure 21: Slack histograms under worst-case conditions (left flat, right tile-based)

Besides timing and area also some design characteristics are extracted from the layouts. Table 7 shows for case 1 the real values, and these values represent 100%. For the cases 3 and 5 the values are given in percents in relation to case 1. It can be seen that the tile-based method uses more metal2, metal3 and metal4. This could be explained by the fact that standard blocks are only made with metal1, 2 and 3. Note that the standard cells use most of the metal 1 routing resources. Leaving only metal 2 and 3 for interconnect routing. (See Table 8.) For toplevel the same holds for metal1 and metal2 in relation to blocking routing resources, but a low area utilization helps the interconnect routing. This can also be seen when looking at design case 5.

Case		1		3	5
Total signal capacitance	[pF]	4512.57	[%]	95.0	104.1
Total wire length	[m]	13.92	[%]	98.7	107.6
vias	#	1195875	[%]	70.7	93.5
Metal1	[um]	685125	[%]	77.2	114.2
Metal2	[um]	1901676	[%]	125.1	119.6
Metal3	[um]	3507721	[%]	109.1	109.9
Metal4	[um]	4122462	[%]	100.2	116.6
Metal5	[um]	2342918	[%]	95.2	106.6
Metal6	[um]	1360487	[%]	47.3	57.9

Table 7: Wire characteristics.

Table 8 shows a more detailed distribution for each level in the tile-based method. On toplevel also metal1 is used. These metal1 wires are for solving local routing congestion problems, and for connecting pins of tiles. This is possible because the tile pins are also visible in metal2 and metal1. From the 13.74 meters of total wiring, 75% is used on toplevel. These 75% is equivalent to 5% wires in the netlist. The total capacitance of the tiles is equal to the toplevel capacitance. On toplevel 1677 components are used for

timing optimisation, which is about 1.4% of all the components in the design.

Case		pparray (toplevel)	ppone	pplogic
Total signal capacitance	[%]	48.8	51.1	0.1
Total wire length	[%]	73.9	25.6	0.5
vias	[%]	19.4	80.5	0.1
Metal1	[%]	38.3	61.6	0.1
Metal2	[%]	19.8	80.1	0.1
Metal3	[%]	64.7	35.3	0.0
Metal4	[%]	100	0	0
Metal5	[%]	100	0	0
Metal6	[%]	100	0	0
Synthesis results				
Nets	[%]	5.6	94.3	0.1
Components	[%]	0	99.9	0.1
Place and route results				
Nets	[%]	6.9	93.0	0.1
Components	[%]	1.4	98.5	0.1

Table 8: Tile-based internal characteristics

When analysing the toplevel placement results for both approaches it can be seen that the overall Rent exponent is lower in the tile-based method. See Figure 22. The red line (marked A) is the found average for that layout style. The left picture is from flat and the right from the tile-based method.

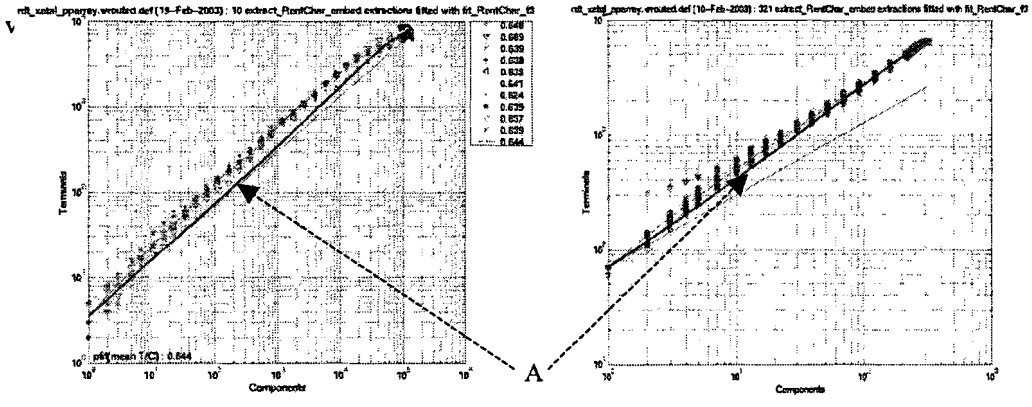

Figure 22: Rent exponent extraction

On the horizontal axes the number of instances are plotted and on the vertical axes the number of IO terminals for that given cluster. For the flat design the Rent parameter was extracted from the average of 200 starting points. In both cases the Rent parameter was about 0.643. When using Rents rule the calculated exponent is 0.636. This can be explained by the fact that the netlist is very regular and has a lot of IO pins.

7. Conclusion

At this moment a complete automated hierarchical design flow is made to convert a gate level netlist into a CoReUse database with gds2, timing and physical verification, as a final result. This flow is based on Cadence tools and contains about 30 makefiles, 2 Philips internal tools, and 150 scripts. The processor array of the Xetal projects was used as design case. The result is that this design could be made 3 times quicker in run-time compared to the flat layout style, in the same area. This gain in run-time gives a loss of 17% in timing closure on the worst-case path. For the rest of the paths the tile-based method is about 8% better. Besides that also a flat run was done with minimum area. When comparing the tile-based method against this result, it is still 3.5 times quicker in run-time with the same timing for the critical path. For the rest of the paths the result is 15% better. On area the tile-based method is about 23% worse. One reason for that is because the optimisation cells have a very bad area utilisation. If this can be solved efficiently, and assuming that this would not give a routing congestion problem, the area is comparable.

One of most important conclusions is that the tile-based method can handle larger designs than flat. Besides that the tile-based method provides the possibility to do faster design-space exploration. For more reliable conclusions, exercises with the automated hierarchical design flow on different designs are encouraged.

8. Future work

Some interesting topics for further research are:

- More and different design cases,
- Clustering algorithm,
- Abstract modelling of layout,
- Optimal aspect ratio of tiles,
- Buffer insertion for pins,
- Choice between bottom-up or top-down design,
- Toplevel optimisation with regards to:
    - Timing,
    - Cross talk.

Besides these topics the partition step could be done much higher in the design flow. At this moment it is done after a hierarchical synthesis step, but it could also be done on a generic netlist or on RT level. The scripts that are available have the possibility for partitioning a generic netlist instead of a synthesised verilog. But some experiments showed strange results. Main reason for that was the absence of technology information and the way the generic netlist was created. To better show the value of this methodology, more regular and non-regular designs have to be verified.

Glossary

IC	Integrated Circuit
MoSAIC	Methodologies on Silicon Architecting for Integrated Circuits
SoC	System-on-a-chip
ITRS	International Technology Roadmap for semiconductors
ECO	Engineering change order
EMC	Electro-Magnetic Compatibility
SIMD	Single Instruction Multiple Data
ACIVS	Advanced Concepts for Intelligent Vision Systems
CAD	Computer Aided Design
IP	Intellectual Property
I/O	Input-Output
RTL	Register Transfer Level
WLM	Wire Load Model
DRC	Design rule Check
ERC	Electrical rule Check
ARC	Antenna rule Check
LVS	Layout Versus Schematic
ECO	Engineering Change Order
NoC	Networks-On-Chips
TSMC	Taiwan Semiconductor Manufacturing Company Ltd.
DSPF	Detailed Standard Parasitic Format
rSPF	Reduced Standard Parasitic Format
TLF	Timing Library File
PVT	Process, Voltage, Temperature
CPU	Central Processing Unit
CoReUse	Core Repetitive Use
DATE	Design, automation and Test conference in Europe

References

[1] Semiconductor Industry Association,
International Technology Roadmap for Semiconductors 1999.
http://public.itrs.net/

[2] Hagen, L., A.B. Kahng, F.J. Kurdahi and C. Ramachandran.
On the intrinsic parameter and spectra-based partitioning methodologies.
IEEE Transactions on computer aided design, january 1994, Vol. 13, p. 27-37

[3] Christie, P.
A differential equation for placement analysis.
IEEE Transactions on VLSI systems, December 2001, Vol. 9. nr. 6, p. 913-921.

[4] Wiel, P van de.
SI-EMC design rules and guidelines.
Philips Internal report: RWR-562-PVDW-02110-PvdW,16 July 2002.

[5] Liebmann, L., G. Northrop, J. Culp, L. Signal, A. Barish and C. Fonseca.
Layout optimisation at the pinnacle of optical lithography.
Proceedings of SPIE, Santa Clara, USA, Vol. 5042, 27 February 2003

[6] Sherwani, N.
"algorithms for VLSI physical design automation".
ISBN:65465475754, 3rd ed., Kluwer, 1999, Chapter 7.4, p. 236-240.

[7] Davis, J.A., K de Vivek and J.D. Meindl.
A stochastic wire-length distribution for gigascale integration (GSI)-part 1.
IEEE Transactions on electron devices, March 1998, Vol. 45, p.580-597.

[8] Dambre, J., P. Verplaetse, D. Stroobandt and J. van Campenhout.
Getting more out of Donath's hierarchical model for interconnect prediction.
In: Proceedings of SLIP'02, San Diego, USA, 6-7 april 2002, p. 9-16.

[9] Wimer, S. and I. Koren, I. Cederbaum
Optimal aspect ratios of building blocks in VLSI.
IEEE Transactions on computer aided design, 1989, Vol. 8, p. 139-145.

[10] Otten, R.H.J.M.
Efficient floorplan optimisation.
IEEE International Conference on Computer Design, 1983, pp. 499-502.

[11] Groeneveld, P. and L. van Ginneken
"Method of designing a constraint-driven integrated circuit layout".
Patent WO9952049, 14 October 1999.

[12] www.telairty.com

[13] www.ammocore.com

[14] Katsioulas, A., S. Chow, J. Avidan and D. Fotakis.
"integrated circuit architecture with standard blocks".
Patent US6467074, 15 October 2002.

[15] Liu, D.
"Integrated circuit block model representation hierachical handling of timing exceptions".
Patent US6493864-B1, 10 December 2002.

[16] Abbo, A.A. and R.P. Kleihorst.
"Smart Cameras: Architectural challenges".
In: Proceedings of ACIVS-2002, Ghent, Belgium, cd-rom, 9-11 Sep. 2002.

[17] Rijpkema, E., K. Goossens, A. Radulescu et.al.
"Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip".
In: Proceedings of DATE, Munich, Germany, p. 350-355, March 2003.

A. Analysis/Clustering tcl script

```
set_global echo_commands false
set tcl_precision 10

proc largest_leaf_cluster { mod max_cells } {
  if {[llength [all_children $mod]] != 0} {
   foreach i [all_children $mod] {
    set max_cells [largest_leaf_cluster $i $max_cells ]
   }
  } else {
   echo [get_attribute $mod FOUND_cellsnr_flat] " cells in =>" [get_name $mod]
>> largest_leaf_cluster_info
   if { [string range [get_name $mod] 0 5] != "cells_" } {
    if { [get_attribute $mod FOUND_cellsnr_flat] > $max_cells } {
     set max_cells [get_attribute $mod FOUND_cellsnr_flat]
     echo $max_cells "-- was bigger --" [get_name $mod] >> larg-
est_leaf_cluster_info
    }
   }
  }
  return $max_cells
}

proc toplevel_synthesis { mod } {
 set_current_module  $mod
 set_top_timing_module  $mod
 do_optimize -priority area -effort medium
 set children [all_children $mod]
 set instances [find -instances *]
 set instances_new ""
 foreach i $children {
  set child_name [get_info $i name]
  foreach j $instances {
   set instance_name [get_info [get_info $j cellref] name]
   if { $child_name != $instance_name } {
    lappend instances_new $j
   }
  }
  set instances $instances_new
  set instances_new ""
 }
 if { [llength $instances] != 0} {
  set opt_cells_ref ""
  set opt_cells_instance ""
  foreach i $instances {
   set instance_cellref [get_info $i cellref]
   if { [lsearch $opt_cells_ref $instance_cellref] == -1 } {
    lappend opt_cells_ref $instance_cellref
    lappend opt_cells_instance $i
   }
  }
  set db_units [get_attribute [find -top] db_units]
  set db_pitch [get_attribute [find -top] db_pitch]
  set db_height [get_attribute [find -top] db_height]
  set height [get_attribute [find -top] db_cluster_height]
  set nearest_width [get_attribute [find -top] db_cluster_nearest_width]
  for {set i 0} {$i < [llength $opt_cells_instance]} {incr i 1} {
   do_create_hierarchy -module rdt_xetal_[get_name [lindex $opt_cells_ref $i]]
[lindex $opt_cells_instance $i] -no_feedthrough
   set_current_module $mod
```



```
   set_top_timing_module $mod
   set rows [expr ceil((([get_area -module rdt_xetal_[get_name [lindex
$opt_cells_ref
$i]]]*$db_units*$db_units)/$nearest_width)/($db_height*$db_units))]
   set width [expr round($db_height*$db_units*$rows+2*$db_pitch*$db_units)]
   puts [concat coreuse_cluster [find -module rdt_xetal_[get_name [lindex
$opt_cells_ref $i]]] $height $width rdt_xetal_[get_name [lindex $opt_cells_ref
$i]]]
   coreuse_cluster [find -module rdt_xetal_[get_name [lindex $opt_cells_ref
$i]]] $height $width
   set_current_module $mod
   set_top_timing_module $mod
  }
 }
}

proc design_cluster { target_libname mod } {
 set cluster_area 0
 set tot_rep_factor 0
 foreach i $mod {
  set rep_factor 0
  foreach j [all_parents $i] {
   set rep_factor [expr $rep_factor+[get_attribute $i FOUND_REP_$j]]
   set tot_rep_factor [expr $tot_rep_factor+[get_attribute $i FOUND_REP_$j]]
  }
  set cluster_area [expr $cluster_area+$rep_factor*[get_area -module $i]]
 }
 set db_tech_info [get_site_info ../lef/[lindex $target_libname 0].lef]
 set db_units [lindex $db_tech_info 0]
 set db_pitch  [lindex $db_tech_info 1]
 set db_height [lindex $db_tech_info 2]

 set tech_site [expr $db_units*$db_pitch]
 set square_length [expr ceil(sqrt([expr $clus-
ter_area/$tot_rep_factor])*$db_units)]
 set nearest_width [expr round(ceil($square_length/$tech_site)*$tech_site) ]
 set xio [expr round(6*$tech_site)]
 set height [expr round($nearest_width + 2*$xio)]

 set_attribute [find -top] db_units $db_units
 set_attribute [find -top] db_pitch $db_pitch
 set_attribute [find -top] db_height $db_height
 set_attribute [find -top] db_cluster_height $height
 set_attribute [find -top] db_cluster_nearest_width $nearest_width

 #puts [concat $nearest_width $height $util $xio $db_tech_info [get_name $mod]]
 foreach i $mod {
  set rows [expr ceil((([get_area -module
$i]*$db_units*$db_units)/$nearest_width)/($db_height*$db_units))]
  set width [expr round($db_height*$db_units*$rows+2*$db_pitch*$db_units)]
  puts [concat coreuse_cluster $i $height $width [get_name $i] ]
  coreuse_cluster $i $height $width
 }
}

proc get_site_info { filename } {
 set fileId [open $filename r 0600]
 set found_site_core 0
 set db_units -1
 set db_pitch -1
 set db_height -1
 while {[gets $fileId line] >= 0 } {
  if { [lsearch $line "DATABASE"] == 0} {
   set db_units [lindex $line 2]
```

```
  }
  if {[llength $line] == 2 & [lsearch $line "SITE"] == 0 & [lsearch $line
"CORE"] == 1 } {
   set found_site_core 1
  }
  if {[llength $line] == 5 & [lsearch $line "SIZE"] == 0 & $found_site_core ==
1 } {
   set found_site_core 0
   set db_pitch [lindex $line 1]
   set db_height [lindex $line 3]
  }
 }
 close $fileId
 return [concat $db_units $db_pitch $db_height]
}

proc coreuse_cluster { mod height width} {
 puts "------------- CoReUse DATA creating "
 set path [pwd]
 puts $path
 mkdir -p ../../../[get_name $mod]
 cd ../../../[get_name $mod]
 ln -s /home/piramid/paradice/2.2h/flow/Makefile_coreuse Makefile
 puts "------------- CREATING BLOCK LEVEL COREUSE DATA"
 gmake data LEVEL=block
 set_current_module $mod
 set_top_timing_module $mod
 puts "------------- GCF AND VERILOG IS CREATED"
 write_verilog NETLIST/[get_name $mod]_netlist_scn_nr.v
 write_gcf_assertions -version 1.4 CONSTRAINTS/[get_name $mod].gcf
 cd sedsm ;
 puts "------------- RUN SUBBLOCK FLOW"
 set rowutil "Y $width"
 puts "gmake change_cmos18to12"
 gmake change_cmos18to12_tm QUEUE=lopes HOST=lion
 puts "gmake subblock BLOCK_HEIGHT=$height HEIGHT_UTIL=$rowutil QUEUE=lopes
HOST=lion"
 gmake subblock LEFVERSION=_5.3 BLOCK_HEIGHT=$height HEIGHT_UTIL=$rowutil
QUEUE=lopes HOST=lion
 cd $path
}

proc coreuse_cluster_cmos12 { mod height width} {
 puts "------------- CoReUse DATA creating "
 set path [pwd]
 puts $path
 mkdir -p ../../../[get_name $mod]
 cd ../../../[get_name $mod]
 ln -s /home/piramid/paradice/2.2h/flow/Makefile_coreuse Makefile
 puts "------------- CREATING BLOCK LEVEL COREUSE DATA"
 gmake data LEVEL=block
 set_current_module $mod
 set_top_timing_module $mod
 puts "------------- GCF AND VERILOG IS CREATED"
 write_verilog NETLIST/[get_name $mod]_netlist_scn_nr.v
 write_gcf_assertions -version 1.4 CONSTRAINTS/[get_name $mod].gcf
 cd sedsm ;
 puts "------------- RUN SUBBLOCK FLOW"
 set rowutil "Y $width"
 puts "gmake change_cmos18to12"
 gmake change_cmos18to12 QUEUE=lopes HOST=lion
 puts "gmake subblock BLOCK_HEIGHT=$height HEIGHT_UTIL=$rowutil QUEUE=lopes
HOST=lion"
```



```
 gmake subblock LEFVERSION=_5.3 BLOCK_HEIGHT=$height HEIGHT_UTIL=$rowutil
QUEUE=lopes HOST=lion
 cd $path
}

proc coreuse_cluster_cmos18 { mod height width} {
 puts "------------- CoReUse DATA creating "
 set path [pwd]
 puts $path
 mkdir -p ../../../[get_name $mod]
 cd ../../../[get_name $mod]
 ln -s /home/piramid/paradice/2.2h/flow/Makefile_coreuse Makefile
 puts "------------- CREATING BLOCK LEVEL COREUSE DATA"
 gmake data LEVEL=block
 set_current_module $mod
 set_top_timing_module $mod
 puts "------------- GCF AND VERILOG IS CREATED"
 write_verilog NETLIST/[get_name $mod]_netlist_scn_nr.v
 write_gcf_assertions -version 1.4 CONSTRAINTS/[get_name $mod].gcf
 cd sedsm ;
 puts "------------- RUN SUBBLOCK FLOW"
 set rowutil "Y $width"
 puts "gmake subblock BLOCK_HEIGHT=$height HEIGHT_UTIL=$rowutil QUEUE=lopes
HOST=lion"
 gmake subblock BLOCK_HEIGHT=$height HEIGHT_UTIL=$rowutil QUEUE=lopes HOST=lion
 cd $path
}

proc cluster_synthesis { mod } {
 set top [get_current_module]
 foreach i $mod {
  set_current_module $i
  set_top_timing_module $i
  reset_dont_modify $i
  limit 900 do_optimize -priority area -effort medium
  set_dont_modify $i
 }
 set_current_module $top
}

proc dissolve_clusters { mod } {
 set top [get_current_module]
 foreach i $mod {
  set_current_module $i
  do_dissolve_hierarchy -hierarchical
 }
 set_current_module $top
}

proc rent_clusters { mod debug max_size clusters } {
 if { $clusters != "NO_SOLUTION" } {
  if {[llength [all_children $mod]] != 0} {
   foreach i [all_children $mod] {
    if { $clusters != "NO_SOLUTION" } {
    if { [string range [get_name $i] 0 5] != "cells_" } {
     if { [get_attribute $i FOUND_candidate] == 1 && $max_size >=
[get_attribute $i FOUND_cellsnr_flat]} {
      if { $clusters == "" } {
      set clusters $i
      echo "Candidate oke, list empty     : " [get_name $i]  >>
rent_clusters_info
      echo [get_name $clusters] >> rent_clusters_info
      } else {
      set clusters [linsert $clusters 0 $i]
```

```
      echo "Candidate oke, list not empty : " [get_name $i] >>
rent_clusters_info
      echo [get_name $clusters] >> rent_clusters_info
      }
     } else {
     echo "Candidate not oke (size or rent) descend : " [get_name $i] >>
rent_clusters_info
     echo $clusters >> rent_clusters_info
     set clusters [rent_clusters $i $debug $max_size $clusters]
     }
    } else {
     echo "SPECIAL CLUSTER FOUND, STILL HAS TO BE CHECKED :" [get_name $i] >>
rent_clusters_info
     if { [get_attribute $i FOUND_candidate] == 1 } {
      if { $clusters == "" } {
      set clusters $i
      echo "Candidate oke, list empty     : " [get_name $i]  >>
rent_clusters_info
      echo [get_name $clusters] >> rent_clusters_info
      } else {
      set clusters [linsert $clusters 0 $i]
      echo "Candidate oke, list not empty : " [get_name $i] >>
rent_clusters_info
      echo [get_name $clusters] >> rent_clusters_info
      }
     } else {
     echo "Candidate not oke (size or rent) descend : " [get_name $i] >>
rent_clusters_info
     echo $clusters >> rent_clusters_info
     set clusters [rent_clusters $i $debug $max_size $clusters]
     }
    }
     }
   }
  } else {
   set clusters "NO_SOLUTION"
   echo "No children available !!!!!! Stop searching. < NO_SOLUTION >" >>
rent_clusters_info
  }
 }
 return $clusters
}

#     if { $clusters == "NO_SOLUTION" || $clusters == ""} {

proc cluster_selection { mod debug cell_numbers } {
 foreach i [all_children $mod] {
  if { [get_attribute $i FOUND_candidate] != 1} {
   cluster_selection $i $debug $cell_numbers
  } else {
   lappend cell_numbers [get_attribute $i FOUND_cellsnr_flat]
  }
 }
 set multiplier [lsort $cell_numbers]
 if { [expr [lindex $multiplier [expr [llength $multiplier] - 1]] / [lindex
$multiplier 0]] <= 30 } {
  if {$debug == 1} {
   echo "FOUND CLUSTERS FOR " [get_name $mod] " : " [get_names [all_children
$mod]] >> cluster_selection_info
   echo "cluster numbers : " $multiplier >> cluster_selection_info
  }
 }
}
```



```
proc rent_candidates_update { mod debug rentplus} {
 foreach i [all_children $mod] {
  rent_candidates_update $i $debug $rentplus
 }
 if {$mod != [find -top]} {
  set FOUND_rent [get_attribute $mod FOUND_rent]
#  if {$FOUND_rent != 0 } {
   if { [expr [get_attribute [find -top] FOUND_rent] + $rentplus] >=
$FOUND_rent } {
    set_attribute $mod FOUND_candidate 1
    set_attribute [find -top] FOUND_candidates [linsert [get_attribute [find -
top] FOUND_candidates] 0 $mod]
   } else {
    set_attribute $mod FOUND_candidate 0
   }
#  } else {
#   set_attribute $mod FOUND_candidate 0
#  }
  if {$debug == 1} {
   echo [get_name $mod] [get_attribute $mod FOUND_rent] [get_attribute $mod
FOUND_candidate] >> candidates_update_info
  }
 }
}

proc lib_names { mod } {
 set_current_module $mod
 foreach i [find -instances *] {
   echo [get_info $i name] >> lib_names
   echo [get_info [get_info $i cellref] name] >> lib_names
  echo [get_info [get_info $i cellref] library] >> lib_names
 }
}

proc lib_names_hier { mod } {
 set_current_module $mod
 foreach i [find -instances -hierarchical *] {
   echo [get_info $i name] >> lib_names_hier
   echo [get_info [get_info $i cellref] name] >> lib_names_hier
  echo [get_info [get_info $i cellref] library] >> lib_names_hier
 }
}

proc rent_candidates { mod debug } {
 foreach i [all_children $mod] {
  rent_candidates $i $debug
 }
 if {$mod != [find -top]} {
  set FOUND_rent [get_attribute $mod FOUND_rent]
#  if {$FOUND_rent != 0 } {
   if { [get_attribute [find -top] FOUND_rent] >= $FOUND_rent } {
    set_attribute $mod FOUND_candidate 1
    set_attribute [find -top] FOUND_candidates [linsert [get_attribute [find -
top] FOUND_candidates] 0 $mod]
   } else {
    set_attribute $mod FOUND_candidate 0
   }
#  } else {
#   set_attribute $mod FOUND_candidate 0
#  }
  if {$debug == 1} {
   echo [get_name $mod] [get_attribute $mod FOUND_rent] [get_attribute $mod
FOUND_candidate] >> candidates_info
  }
```

```
 }
}

proc create_new_hier { mod debug } {
 foreach i [all_children $mod] {
  create_new_hier $i $debug
 }
 if { [llength [all_children $mod]] != 0 } {
  set_current_module $mod
  if {$debug == 1} {
   echo "old. ------------------------------------" >> create_hier_info
   echo [get_name $mod] >> create_hier_info
   echo "children = " [get_names [all_children $mod]] >> create_hier_info
   echo "parents  = " [get_names [all_parents $mod]] >> create_hier_info
  }
  set children [all_children $mod]
  set instances [find -instances *]
  set instances_new ""
  foreach i $children {
   set child_name [get_info $i name]
   foreach j $instances {
    set instance_name [get_info [get_info $j cellref] name]
    if { $child_name != $instance_name } {
     lappend instances_new $j
    }
   }
   set instances $instances_new
   set instances_new ""
  }

  if { [llength $instances] != 0} {
   do_create_hierarchy -module cells_[get_names $mod] $instances
  }

  if {$debug == 1} {
   echo "new." >> create_hier_info
   echo [get_name $mod] >> create_hier_info
   echo "children = " [get_names [all_children $mod]] >> create_hier_info
   echo "parents  = " [get_names [all_parents $mod]] >> create_hier_info
  }
 }
}

proc hier_analysis { mod debug cell_numbers} {
 rm -rf repetition_info
 rm -rf create_hier_info
 rm -rf pins_cells_info
 rm -rf problem_info
 rm -rf traverse_info
 rm -rf rents_rule_info
 rm -rf rents_rule_info_short
 rm -rf candidates_info
 rm -rf cluster_selection_info

 if {$debug == 1} {
  puts "Creating new hierarchy ............."
 }
 create_new_hier  [find -module $mod] $debug
 if {$debug == 1} {
  puts "Searching for repetition ..........."
 }
 search_rep   [find -module $mod] $debug
 if {$debug == 1} {
  puts "Adding design properties ..........."
```



```
 }
 pins_cells   [find -module $mod] $debug
 if {$debug == 1} {
  puts "Searching for design info ........."
 }
 traverse_hier  [find -module $mod] $debug
 if {$debug == 1} {
  puts "Calculating Rents rule ............."
 }
 rents_rule   [find -module $mod] $debug
 if {$debug == 1} {
  puts "Searching for Rent candidates ......"
 }
 rent_candidates  [find -module $mod] $debug
 if {$debug == 1} {
  puts "Cluster selection on instances ....."
 }
# cluster_selection [find -module $mod] $debug $cell_numbers

 if {$debug == 1} {
  mv repetition_info    repetition_info_$mod
  mv pins_cells_info    pins_cells_info_$mod
  mv problem_info       problem_info_$mod
  mv traverse_info      traverse_info_$mod
  mv rents_rule_info    rents_rule_info_$mod
  mv rents_rule_info_short   rents_rule_info_short_$mod
  mv create_hier_info    create_hier_info_$mod
  mv candidates_info    candidates_info_$mod
#  mv cluster_selection_info   cluster_selection_info_$mod
 }
}

proc rents_rule { mod debug } {
 foreach i [all_children $mod] {
  rents_rule $i $debug
 }
 set nivo_cells [get_attribute $mod FOUND_cellsnr_flat]
 set nivo_term  [get_attribute $mod FOUND_npins]
 set nivo_tpc   [get_attribute $mod FOUND_tpc]
 if {$debug == 1} {
  echo " " >> rents_rule_info
  echo "module      :" [get_name $mod] >> rents_rule_info
  echo "module      :" [get_name $mod] >> rents_rule_info_short
  echo "nivo_cells  :" $nivo_cells  >> rents_rule_info
  echo "nivo_term   :" $nivo_term   >> rents_rule_info
  echo "nivo_tpc    :" $nivo_tpc    >> rents_rule_info
 }
 if { $nivo_tpc == 0 } {
  set_attribute $mod FOUND_rent 0
  if {$debug == 1} {
   echo "Rents rule  : ERROR (FOUND_tpc = 0)" >> rents_rule_info
  }
 } else {
  if { $nivo_term == 0 } {
   set_attribute $mod FOUND_rent 0
   if {$debug == 1} {
    echo "Rents rule  : ERROR (FOUND_npins = 0)" >> rents_rule_info
   }
  } else {
   if { $nivo_cells == 1} {
    set_attribute $mod FOUND_rent 1
    if {$debug == 1} {
     echo "Rents rule  : ERROR (FOUND_cellsnr_flat = 1)" >> rents_rule_info
    }
```



```
   } else {
    set_attribute $mod FOUND_rent [expr
(log($nivo_term/$nivo_tpc)/log($nivo_cells))]
    if {$debug == 1} {
     echo "Rents rule  :" [get_attribute $mod FOUND_rent] >> rents_rule_info
     echo "Rents rule  :" [get_attribute $mod FOUND_rent] >>
rents_rule_info_short
    }
   }
  }
 }
 if {$mod != [find -top]} {
  set top_tpc     [get_attribute [find -top] FOUND_tpc]
  set top_npins   [get_attribute [find -top] FOUND_npins]
  set top_cells   [get_attribute [find -top] FOUND_cellsnr_flat]
  if {$debug == 1} {
   echo "top_tpc     :" $top_tpc         >> rents_rule_info
   echo "top_npins   :" $top_npins        >> rents_rule_info
   echo "top_cells   :" $top_cells        >> rents_rule_info
  }
  if { $top_cells == 0 } {
   set_attribute $mod FOUND_chris 0
   if {$debug == 1} {
    echo "Rent_impro  : ERROR (FOUND_cellsnr_flat = 0)" >> rents_rule_info
   }
  } else {
   set chris_denom [expr ($nivo_cells-(($nivo_cells*$nivo_cells)/$top_cells))]
   if { $chris_denom == 1 || $chris_denom <= 0 } {
    set_attribute $mod FOUND_chris 0
    if {$debug == 1} {
     echo "Rent_impro  : ERROR (chris_denom = " $chris_denom ")" >>
rents_rule_info
    }
   } else {
    set chris_num_denom [expr ($top_tpc-($top_npins/$top_cells))]
    if { $chris_num_denom == 0 } {
     set_attribute $mod FOUND_chris 0
     if {$debug == 1} {
      echo "Rent_impro  : ERROR (chris_num_denom = " $chris_num_denom ")" >>
rents_rule_info
     }
    } else {
     set chris_num_num [expr ($nivo_term-($top_npins*$nivo_cells/$top_cells))]
     if { $chris_num_num == 0 } {
      set_attribute $mod FOUND_chris 0
      if {$debug == 1} {
       echo "Rent_impro  : ERROR (chris_num_num = " $chris_num_num ")" >>
rents_rule_info
      }
     } else {
      set chris_num_num_denom [expr ($chris_num_num/$chris_num_denom)]
      if { $chris_num_num_denom <= 0 } {
       set_attribute $mod FOUND_chris 0
       if {$debug == 1} {
        echo "Rent_impro  : ERROR (chris_num_num_denom = " $chris_num_num_denom
")" >> rents_rule_info
       }
      } else {
       set_attribute $mod FOUND_chris [expr
log($chris_num_num/$chris_num_denom)/log($chris_denom)]
       if {$debug == 1} {
        echo "Rent_impro  :" [get_attribute $mod FOUND_chris]  >>
rents_rule_info
```



```
        echo "Rent_impro  :" [get_attribute $mod FOUND_chris]  >>
rents_rule_info_short
       }
      }
     }
    }
   }
  }
 }
}

proc traverse_hier { mod debug } {
 foreach i [all_children $mod] {
  traverse_hier $i $debug
 }
 if {$debug == 1} {
  echo " " >> traverse_info
  echo "module          :" [get_names $mod] >> traverse_info
  echo "childs          :" [get_names [all_children $mod]] >> traverse_info
  echo "parents         :" [get_names [all_parents $mod]] >> traverse_info
 }
 set view_number [lsearch [get_names [get_info $mod views]] "netlist"]
 if {$view_number != -1 } {
  set view [lindex [get_info $mod views] $view_number]
  set_attribute $mod FOUND_npins [llength [get_info $view ports]]
  if {$debug == 1} {
   echo "views           :" [get_names $view] >> traverse_info
   echo "# _opt_inst_count :" [get_attribute $view _opt_inst_count] >> trav-
erse_info
   echo "# ports         :" [get_attribute $mod FOUND_npins] >> traverse_info
  }
  if { [llength [all_children $mod]] == 0} {
   set_attribute $mod FOUND_cellsnr_flat [get_attribute $mod FOUND_cellsnr]
   set_attribute $mod FOUND_cellpins_flat [get_attribute $mod FOUND_cellpins]
  } else {
   set child_cells [get_attribute $mod FOUND_cellsnr]
   set child_cellpins [get_attribute $mod FOUND_cellpins]
   foreach j [all_children $mod] {
    set child_pinsflat [expr [get_attribute $j FOUND_REP_$mod]*[get_attribute
$j FOUND_cellpins_flat]]
    set child_flat [expr [get_attribute $j FOUND_REP_$mod]*[get_attribute $j
FOUND_cellsnr_flat]]
    set child_cells [expr $child_cells+$child_flat]
    set child_cellpins [expr $child_cellpins+$child_pinsflat]
   }
   set_attribute $mod FOUND_cellsnr_flat $child_cells
   set_attribute $mod FOUND_cellpins_flat $child_cellpins
  }
  if {$debug == 1} {
   echo "# level std     :" [get_attribute $mod FOUND_cellsnr]   >> trav-
erse_info
   echo "# HIER std CALC :" [get_attribute $mod FOUND_cellsnr_flat]  >> trav-
erse_info
   echo "# flat pin      :" [get_attribute $mod FOUND_cellpins]   >> trav-
erse_info
   echo "# HIER pin CALC :" [get_attribute $mod FOUND_cellpins_flat]  >> trav-
erse_info
  }
  if { [get_attribute $mod FOUND_cellsnr_flat] != 0 } {
   set_attribute $mod FOUND_tpc [expr double([get_attribute $mod
FOUND_cellpins_flat])/double([get_attribute $mod FOUND_cellsnr_flat])]
   if {$debug == 1} {
    echo "# tpc(hier)CALC :" [get_attribute $mod FOUND_tpc]   >> traverse_info
   }
```

```
   } else {
    set_attribute $mod FOUND_tpc 0
    if {$debug == 1} {
     echo "# tpc(hier)CALC : NO CELL(S)ON THIS HIERACHICAL LEVEL !" >> trav-
erse_info
    }
   }
  } else {
   if {$debug == 1} {
    echo "NO NETLIST VIEW AVAILABLE !"  >> traverse_info
   }
  }
}

proc pins_cells { mod debug } {
 foreach i [all_children $mod] {
  pins_cells $i $debug
 }
 set cellpins 0
 set teller 0
 set_current_module $mod
 set children [all_children $mod]
 set instances [find -instances *]
 set instances_new ""
 foreach i $children {
  set child_name [get_info $i name]
  foreach j $instances {
   set instance_name [get_info [get_info $j cellref] name]
   if { $child_name != $instance_name } {
    lappend instances_new $j
   }
  }
  set instances $instances_new
  set instances_new ""
 }
 foreach i $instances {
  set cellpins [expr $cellpins+[llength [get_info $i pins]]]
  set teller [expr $teller+1]
 }
 set_attribute $mod FOUND_cellpins $cellpins
 set_attribute $mod FOUND_cellsnr $teller
 if {$debug == 1} {
  echo " " >> pins_cells_info
  echo [get_name $mod] >> pins_cells_info
  echo "children = " [all_children $mod] >> pins_cells_info
  echo "cellpins = " $cellpins >> pins_cells_info
  echo "cellsnr  = " $teller   >> pins_cells_info
  echo "_opt_inst_count = " [get_attribute [lindex [get_info $mod views]
[lsearch [get_names [get_info $mod views]] "netlist"]] _opt_inst_count] >>
pins_cells_info
  if { [llength [all_children $mod]] == 0} {
   if { $teller != [get_attribute [lindex [get_info $mod views] [lsearch
[get_names [get_info $mod views]] "netlist"]] _opt_inst_count] } {
    echo [get_name $mod] >> problem_info
    echo "children = " [all_children $mod] >> problem_info
    echo "cellpins = " $cellpins >> problem_info
    echo "cellsnr  = " $teller   >> problem_info
    echo "_opt_inst_count = " [get_attribute [lindex [get_info $mod views]
[lsearch [get_names [get_info $mod views]] "netlist"]] _opt_inst_count] >>
problem_info
    echo "PROBLEM CELL : " [get_name $mod]   >> problem_info
   }
  }
 }
```



```
}

proc search_rep { mod debug } {
 foreach i [all_children $mod] {
  set count 0
  foreach j [all_children -instances $mod] {
   if { [get_names [get_info $j cellref]] == [get_names $i] } {
    set count [expr $count+1]
   }
  }
  if {$debug == 1} {
   echo " " >> repetition_info
   echo [get_names $mod] "=" $count "times" [get_names $i] >> repetition_info
  }
  set_attribute $i FOUND_REP_$mod $count
  search_rep $i $debug
 }
}

set_global echo_commands true

##### END of clustering and analysis procedures #####

#####   Begin of cadence ambit toplevel script  #####

set BLOCK_NAME cell_name
set flow "map_asic"
source ../cmd/rtl_list_block.tcl
source ../cmd/ac_setup.tcl
read_adb   ${database_area}/${BLOCK_NAME}_generic.adb

set_current_module  ${BLOCK_NAME}
set_top_timing_module  ${BLOCK_NAME}
source    ${constraints_area}/tech_setup.tcl
source    ${constraints_area}/constraints.tcl
do_optimize -effort medium -dont_uniquify -target_slack 0.1
write_adb    -hierarchical  ${database_area}/${BLOCK_NAME}_mapped_dont.adb
write_verilog  -hierarchical  ./${BLOCK_NAME}_mapped_dont.v

read_adb   ${database_area}/${BLOCK_NAME}_mapped_dont.adb

source ../cmd/hdfm_functions.tcl;
set debug 1
set cell_numbers " "
set_global echo_commands false
hier_analysis ${BLOCK_NAME} $debug $cell_numbers
set_global echo_commands true

rm -rf largest_leaf_cluster_info
set tech_size 1400
set max_size [expr [largest_leaf_cluster ${BLOCK_NAME} $tech_size ] + 1 ]
echo "Maximum leaf cluster without taking into account cells_-clusters :"
$max_size >> largest_leaf_cluster_info

set max_size [expr ceil([get_attribute [find -top] FOUND_cellsnr_flat]/15)]
set clusters "NO_SOLUTION"
for {set delta_rent 0.00} {$clusters == "NO_SOLUTION"} {set delta_rent [expr
$delta_rent+0.5]} {
      echo "delta_rent is :" $delta_rent
      rent_candidates_update [find -module ${BLOCK_NAME}] $debug $delta_rent
      set clusters [rent_clusters [find -module ${BLOCK_NAME}] $debug $max_size
""]
      set needed_extra_rent $delta_rent
}
```



```
echo "clusters found are :" [get_names $clusters] "with extra rent value of :"
$needed_extra_rent

echo [get_attribute [find -top] FOUND_rent]  [get_attribute [find -top]
FOUND_cellsnr_flat] "INFO FROM : " [get_name [find -top]] >> FOUND_SOLUTION.txt
foreach i $clusters {
       echo [get_attribute $i FOUND_rent] [get_attribute $i FOUND_cellsnr_flat]
"INFO FROM : " [get_name $i] >> FOUND_SOLUTION.txt
}

write_adb    -hierarchical  ${database_area}/${BLOCK_NAME}_before_dissolve.adb

dissolve_clusters $clusters

set_current_module  ${BLOCK_NAME}
set_top_timing_module  ${BLOCK_NAME}
source    ${constraints_area}/tech_setup.tcl
source    ${constraints_area}/constraints.tcl

do_time_budget
set_global echo_commands false
cluster_synthesis $clusters
set_global echo_commands true

set_current_module  ${BLOCK_NAME}
set_top_timing_module  ${BLOCK_NAME}
do_xform_map

do_change_name -use_rules -verbose -log ../log/change_name.log

# design the clusters, start the layout of the subblocks
#set_global echo_commands false
source ../cmd/hdfm_functions.tcl;
design_cluster $target_libname $clusters
#set_global echo_commands true

set_current_module  ${BLOCK_NAME}
set_top_timing_module  ${BLOCK_NAME}

write_adb    -hierarchical  ${database_area}/${BLOCK_NAME}_mapped.adb
write_verilog  -hierarchical  ${ver-
ilog_gates_final_area}/${BLOCK_NAME}_netlist_syn.v
write_verilog  -hierarchical ${ver-
ilog_gates_final_area}/${BLOCK_NAME}_netlist_scn_nr.v
write_gcf_assertions ${gcf_area}/${BLOCK_NAME}.gcf

report_hierarchy > ${reports_area}/${BLOCK_NAME}_mapped_hier.rpt
report_area -cells -hierarchical > ${re-
ports_area}/${BLOCK_NAME}_mapped_area.rpt
report_timing -late  -nworst 30 > ${reports_area}/${BLOCK_NAME}_mapped_late.rpt
report_timing -early -nworst 30 > ${re-
ports_area}/${BLOCK_NAME}_mapped_early.rpt

cp ac_shell.log ../log/ac_shell_map_first.log
cp ac_shell.cmd ../log/ac_shell_map_first.cmd

set_current_module  ${BLOCK_NAME}
set_top_timing_module  ${BLOCK_NAME}

foreach targetlib $target_libname {
       set_cell_property dont_utilize true  -lib ${targetlib} [find -cellref *]
       set_cell_property dont_utilize false -lib ${targetlib} [find -cellref
bf*]
```



```
      set_cell_property dont_utilize false -lib ${targetlib} [find -cellref
iv*]
}

toplevel_synthesis ${BLOCK_NAME}

set BLOCKS [get_names [all_children ${BLOCK_NAME}]]
mv ../cmd/rtl_list.tcl ../cmd/rtl_list.tcl_map_asic_used
set path [pwd]
cd ..;
gmake change_rtl_list BLOCKS=$BLOCKS
cd $path

mv ac_shell.log ../log/ac_shell_map.log
mv ac_shell.cmd ../log/ac_shell_map.cmd
quit

#####   END of cadence ambit toplevel script  #####
```