Layout dependent Synthesis for Manufacturing Costs optimized 3D Integrated Systems

Andy Heinig
Fraunhofer Institute for Integrated Systems
IIS/EAS
Dresden, Germany
Email: andy.heinig@eas.iis.fraunhofer.de

***Abstract*—3D integration opens up entirely new perspectives in chip development, such as integration of different technologies in a stack with smaller form factor as with classical board design. It enables also the partitioning of large SOC designs into a stack with two or more dies. If the resulting 3D-System is optimized, its costs can be smaller than the costs for the manufacturing of the corresponding 2D-System. In this paper a new layout dependent synthesis method for manufacturing costs optimized 3D integrated systems is introduced. As its major part a 3D synthesis optimization method algorithm which used layout information from a floorplanner is presented. The flow was tested on a VLIW processor design, which demonstrates a cost reduction by 3D implementation.**

I. Introduction

In the last years there are two main drivers for 3D integration. The first is the combination of different silicon technologies (e.g. analog, CMOS, sensors) in a single package. For example [1] shows the integration of an image sensor with a high performance processor. The second driver results from the manufacturing costs of large chips such as SOCs or multi-processors. As shown in [2] the manufacturing of a die with $200mm^2$ costs 400 units, whereas the manufacturing of a $100mm^2$ die costs only 133 units. Assuming that the $200mm^2$ system can be partitioned into two dies of $100mm^2$, the total die costs is only 266 unit. Even if additional costs for TSV processing and stacking will occur, the total costs will be below the 400 cost units of the single die realization. Beside cost reduction, 3D integration can also improve the system performance. With the right placement, TSVs allows to reduce the interconnect length between functional blocks. Utilizing the cost and performance advantages of 3D integration well requires proper synthesis of functional blocks on a set of dies.

A classical 2D design is divided into different stages to translate the given RTL-Code to the final layout. The mayor stages of such a design flow are shown in figure 1. The flow is broken into these stages to bring the computation time into a range which is solvable in an acceptable time. In this work we are mainly interested to develop new 3D synthesis optimization methods and to integrate it into a 3D flow. So in the following the synthesis should be explained more in detail. As it is shown in figure 1 the synthesis is also divided into two steps. This two synthesis steps can be classified into two major categories: synthesis with or without layout information.

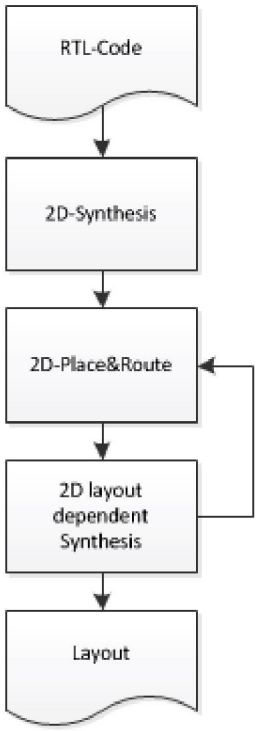

Fig. 1. 2D design flow with layout dependent synthesis optimization.

The algorithms from both categories where used to fulfill different task and optimization goals which are derived from the classical 2D design flow. In such a classical 2D design flow the RTL-code is firstly synthesized followed by the place and route. Such a hierarchal approach is a practical from the computational time. For old technologies such an approach is useable because the layout can be approximated with some simple models. Which models can be used in the synthesis step to guide them. In modern technologies such assumptions doesnt hold. To eliminate the use of layout approximation in the synthesis step a new category of synthesis algorithms was developed. In these algorithms the layout and synthesis flows are performed together. But the drawback of such algorithms are there long running time. So in practical applications a two-step approach is used.

The advantages of 3D integration such as shorter interconnect and reduced costs can be used only, if the complete design flow is adapted for 3D implementation. Especially the synthesis of the modules and the placement of functional blocks to the different dies in a stack and the insertion of related TSV is an important task that is not supported by current 2D design tools.

However, in this work a new developed 3D synthesis method should be presented. Because the presented is layout dependent the necessary environment is also presented. In this work the environment is given in form of a floorplanner from which the needed layout information can be derived. The developed design 3D design flow is shown in figure 2.

Similar to the classical 2D design flow the now developed 3D flow are also organized in some steps. Because TSVs uses significant die area (in a 45nm technology a TSVs has the same size as 20 nand gates) the number of TSVs should be reduced in all steps of the flow. In the first step of the 3D Flow a classical 2D synthesis is used to transform the RTL-Code into a first standard cell netlist. For complexity reason this netlist is then coarsened into so called supercells which contains a lot of standard cells. With these supercells the floorplanner plans a first floorplan of the 3D system. The TSV insertion step follows. Because the supercells compound a lot of smaller standard cells, so the TSVs can placed into the supercells. The additional area for the TSVs was already considered in the floorplanning step with extra area for the related supercell. With the results of the floorplanner and the TSV insertion steps the new developed 3D layout dependent 3D synthesis improves the former synthesis results. In this case improve means first of all a reduced number of TSVs under consideration of other constraints such as area or timing. With the new synthesis results an update of the floorplan is performed because a variation in the number of TSVs influences the floorplan in a great manner. These two steps are repeated until only very small changes in the synthesis are done. After that the coarsened supercells are refined and a standard 2D place and route tool is used for the place and route step. In a preparation step the original netlist is partitioned into individual netlists for the single dies. Furthermore scripts are generated which places the TSVs to their fixed positions. The standard cells within the supercells are placed first to the center of the supercell. Later their position is optimized by the 2D place and route tool. The timing constraints for every die are calculated by the 3D floorplanner and also set by a script. After the preparation step the standard 2D place and route tool places the standard cells to their best positions considering timing and design rules. With the fixed positions of the TSVs it is guaranteed that there is no mismatch between of the positions from the same TSV on different dies. After this step, was completed for all dies, the 3D system is placed and routed.

The overall optimization goal is a 3D-system with optimized manufacturing costs that also fulfills other constraints such as timing, power or electrical ones. The presented optimization goal and strategy is different to other 3D floorplanners such as [3], [4] which optimize the area of the system.

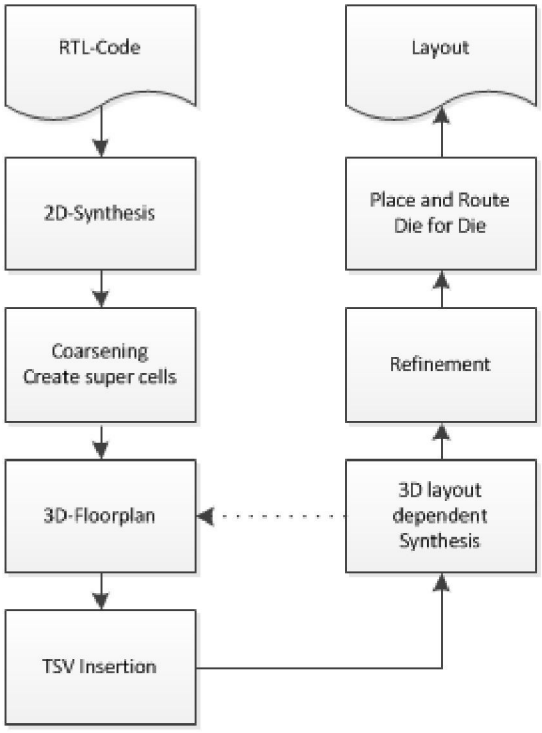

Fig. 2. Proposed 3D design flow.

3D floorplanning and the TSV insertion steps are most important for system optimization. After these steps are completed the place and route steps are done individual for every die by standard place and route tools whereas the positions of TSVs and an approximate position for the standard cells are given by the floorplanning and TSV insertion tool.

The reminder of the paper is organized as follow. In section II known 2D layout dependent synthesis optimization methods are introduced. Because there is not known such method for 3D integration we present in section III the new developed 3D layout dependent synthesis method. This method uses ideas from the 2D case but must consider different optimization goals such as number of TSVs. Also in this section the necessary environment for a layout driven optimization is presented. In our case the layout information there derived from a 3D floorplanner. The paper ends with the result section.

II. Related work

In the following some 2D layout dependent synthesis methods should be presented. In [5] is one of the first layout dependent synthesis optimization presented. In this work there is no repeated iteration between the placement and the layout dependent synthesis optimization. The algorithms works as follows: an initial netlist is synthesized but the latest step of the synthesis the mapping step isn't performed. With this synthesis result a global placement is done and with this position information the decomposition and the mapping is performed. Such an approach allows that the decomposition

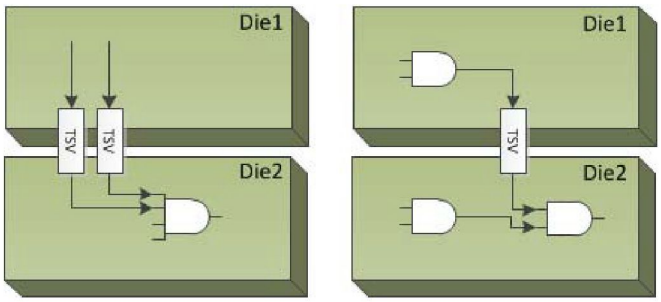

Fig. 3. Different mappings for the same logic but which results in different numbers of TSVs.

results in compact standard cells in areas which have a higher congestion. Otherwise in areas with lower congestion or if two cells are far away the resulting number of standard cells is increased but the cells himself are smaller.

This work is extend in [6] to consider congestion information do guide the layout dependent logic synthesis.

In [7] on other method for such an optimization is presented. The main idea behind this work is to reduce the congestion of an area in both step, placement and routing. This is done by some two major phases. In the first major phase a placement is done and is refined by a layout dependend synthesis optimization. This optimization is driven by the congestion of the area which is reduced by repeated iterative call of the both steps until a given threshold is reached. If the threshold is reached then the second major phase is started. In this phase the placement result are routed under congestion minimization. Then the resulting routing is refined by a layout dependent synthesis optimization. The routing step and the synthesis optimization steps are repeated until the routing congestion is lower as the given maximum routing congestion.

III. 3D FLOW WITH LAYOUT DEPENDENT SYNTHESIS

A. Remapping

The major task for the remapping is to reduce the number of TSVs in the 3D stack. So there are some configurations identified in which the number of TSVs can be reduced. The first of such configurations is shown in 3. In this configuration the 2D synthesis results in a mapping which contains only one standard cell with four input pins. In some parts of the design such an mapping can be a good trade-of between area, congestion and delay. If two of the four predecessors are on an other die at the 3D stack the floorplanner inserts two TSVs as depicted in the figure. For this configuration can another mapping of the graph to the standard cells results in a solution with three standard cells where each of them contains two inputs. This mapping consumes a little bit more placement area, but the advantage of this standard cell mapping is that only one TSV must be inserted by the floorplanner. The task for the 3D layout dependent logic synthesis is to find such configurations and apply the remapping with the use of the well-known DeMorgan rules as it is explained above. This search and remapping step should be repeated because for parts of the currently remapped structure maybe the same optimization can be applied.

In Algorithm 1 is the pseudo code for the remapping synthesis optimization method is depicted:

Algorithm 1 Pseudo code for remapping of standard cells with more than two TSVs in the same die.

```
for all standardcells do
  if standardcell with more than two TSVs at same die then
    remap with DeMorgan rules
    run fast flooplanning
    if floorplan not fulfill timing constraints then
      reject remap
    end if
    if floorplan not fulfill area ratio between dies then
      reject remap
    end if
  end if
end for
```

The area constraints are important because otherwise the area in the involved dies can be to different and this can results in some unusable place in one of the involved dies. Also the timing constraints are very important because the flooplanner optimizes the timing and the 3D synthesis shouldnt work against the floorplanner.

The above described configurations can be extended in a more general way as it is depicted in 4. In this more general situation the two TSVs which are the input of the cell in 3 can be more spread about the die. This spread cells can combined into one cell which has then the two or more TSVs as input as it is also shown in the figure 4 in the middle. If the situation which is shown in the middle is reached the approach is the same as it is used in 3.

To find such solutions a recursive search is applied for a standard cell. For all interconnects to predecessor cells the number of TSVs is counted and for the predecessor cells this approach is applied recursive. This recursive calls are only done to a given depth because most of the participated cells are moved from one die to the other die on and influences the area ratio between the dies. Also if the depth of the recursive calls is not limited the runtime of the algorithm is too long. In practice a depth of 4 or 5 is a good tradeoff between reducing the number of TSVs, the runtime and the area ratio.

In Algorithm 2 is the pseudo code for the general remapping synthesis optimization method is depicted:

B. 3D floorplanner

As described above the 2D synthesis results where partitioned into so called supercells. Because every interconnect between two supercells can result in a TSV if the concerned supercells are placed at different dies. So the coarsening step encourages the goal of lowering the number of TSVs by minimizing the number of cuts in interconnects between the supercells under consideration of balancing the total area of

Algorithm 2 Pseudo code for remapping of standard cells with more than two TSVs in the same die.

```
for all standardcells do
  C_TSV = count TSVs recursive in predecessors until given depth
  if C_TSV > 2 then
    C_cells = collect standard cells which in the path to the TSVs
    R_cells = remap C_cells with DeMorgan rules
    run fast flooplanning
    if floorplan not fulfill timing constraints then
      reject remap R_cells
    end if
    if floorplan not fulfill area ratio between dies then
      reject remap R_cells
    end if
  end if
end for
```

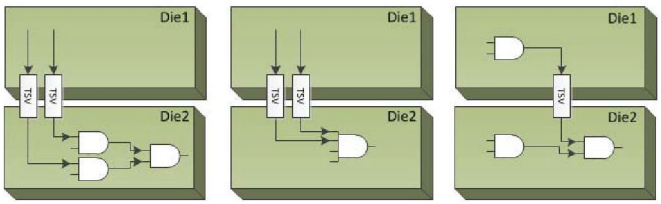

Fig. 4. Different mappings for the same logic but which results in different numbers of TSVs.

the supercells. Because the problem is NP-complete a heuristic strategy similar to the hMetis algorithm [8] is used to solve the problem.

Other floorplanners for 3D-systems such as described in [10] use the hierarchy of the design to derive the supercells. But this assignment from cells to supercells does not regard the number of TSVs in the resulting layout. Therefore in this work a repartitioning algorithm is used to create new supercells.

In [9] an analytical approach for place and route of 3D-Systems is introduced. Its optimization goal is a combination of die area and number of TSVs. This optimization goal induces two problems, if manufacturing costs should be optimized as it is done in this work. First the areas for the single dies must be given. However the area of the dies can only be determined if the number of TSVs is known because TSVs occupy a significant die area. But their number is not determined before 3D place and route. The result is a mutual dependency that can be solved only in a time consuming and possibly non converging iterative loop. The second problem is similar to the first. A strong relation between the number of TSVs and the area exists. If the relation between the dimensions of the different dies is adverse the number of TSVs grows strongly. Finding a good balance for both terms is very difficult and must be done by the designer.

C. Cost function

Stochastic optimization techniques use a cost function to evaluate a solution. It is defined as:

$$cost = x_1 * c_1 + x_2 * c_2 + ... + y_1 * p_1 + y_2 * p_2 + ...$$

The cost function consist of two parts $c_1, c_2, ...$ and $p_1, p_2, ...$ and there corresponding weighting factors $x_1, x_2, ...y_1, y_2, ...$ The first part describes the major optimization goal in the context of this work, the production costs. However every digital design must meet some constraints, as for example timing, power or electrical ones. In the second term of the cost function a penalty term is inserted for every constraint.

If shall the manufacturing cost of a 3D-System be optimized there is the need for a cost model. Different cost models for 3D chips exists, e.g.[10], [11], [12], [13]. We use a cost function for a 3D-System with k layers similar to[10]:

$$c_1 = \sum_{i=1}^{k} \frac{C_{die,i} + C_{TSV,i}}{(Y_{die,i} \cdot Y_{TSV,i} \cdot Y_B^{k-1})} + \frac{C_B \cdot (k-1)}{Y_B^{k-1}}$$

where $C_{die,i}$, $C_{TSV,i}$ are the costs to produce the i-th die and its TSVs. The yields of these processes are $Y_{die,i}$ and $Y_{TSV,i}$. C_B are the costs for stacking of two dies (with the yield Y_B) together.

The first penalty term is added to meet timing constraints. It consists of three regions. The first region describes the case that the floorplan violate the timing constraints whereas the second case is used if the timing is short below the constrain. The penalty is evaluated as follows:

$$p_{time} = \begin{cases} y_1^1 \cdot T & \text{if } T > T_{max} \\ y_1^2 \cdot T & \text{if } T_{buffer} > T \geq T_{max} \\ 0 & \text{otherwiese} \end{cases}$$

in which T is the timing which is calculated for the current floorplan with a statical timing analysis (STA), T_{max} is the maximal accepted timing, T_{buffer} is the aspired timing and y_1^1, y_1^2 are the corresponding penalty factors (with $y_1^1 > y_1^2$).

High number of TSVs implicates some problems. One of them is the increasing area needed for TSVs. In the standard cell area a number of TSVs can be placed in the given TSV grid. If the TSVs require more than the places given by the TSV grid the die area must be increased only for the TSVs. This leads to increased production costs. Another problem is the increasing of the wire length if a TSV cannot be placed in the enclosing box - which is the box given by the pins of the corresponding interconnect. If the number of TSVs is high they are placed often out of the enclosing box resulting in longer interconnects as estimated. The impact of the TSV number in cost function - resulting from the TSV-Yield $Y_{TSV,i}$- is small, so a penalty term p_{TSV} to the objective function is added:

$$p_{TSV} = y_2 \cdot (N_{TSV} - N_{max,TSV})^+$$

in which N_{TSV} ist the number of TSVs in current floorplan, $N_{max,TSV}$ is the desired number of TSVs, y_2 is a penalty factor and $n^+ = \max\{0, n\}$.

The complete cost function is then:

$$cost = x_1 \cdot c_1 + p_{time} + p_{TSV}$$

with the cost weighting factor g, the cost function C and the penalties S_{time} and S_{TSV}. It is important to balance the impact of the different parts of the objective function to find a good solution.

D. Floorplanning

The floorplanning step is the major part of the proposed design flow. At the floorplanning step the supercells will be placed to the individual dies and within the dies to their non-overlapping positions. To handle this mapping from supercells to positions at dies an efficient data structure is needed. Some work was done for developing data structures for the floorplanning of 2D chips. In the 3D domain these structures can be extended to either 2.5D structures or real 3D structures. Real 3D data structures are the 3D slicing tree [3], the 3D CBL [14], the sequence triple and sequence quintuple [15]. 3D data structures have the advantage to handle real 3D supercells which means supercells that occupies area at more than one die. However the supercells generated by the partitioning step will not be placed at different dies and the floorplanning considers only the area consumption of TSV. Therefore in this work we use a 2.5D data structure which holds the data for the die stack together a 2D data structure (e.g. sequence pair) for the individual dies.

Different stochastic optimization techniques such as threshold accepting algorithm [16], simulated annealing [17] or great deluge algorithm [18] are used for floorplanning. In previous work [**?**] we discovered that the threshold accepting algorithm delivers the best results for the production cost optimization floorplanning problem. Evolutionary or genetic algorithms are not considered because this class of algorithms compares - in contrast to the previous ones several different configurations in every optimization step. Storing these configurations in parallel is very memory consuming and therefore for real world examples not applicable.

Stochastic optimization techniques need operations to generate a new configuration from the current one. A new configuration should only have few changes from a current configuration regarding the costs resulting from evaluation of cost function. Six operations are defined in this work, to generate such a neighborhood configuration from the current one:

Stochastic optimization techniques need operation to generate a new configuration from the current configuration. The new configuration should only less differ from the current configuration regarding the costs resulting from the cost function. In this work where defined six operations to generate such a solution neighborhood configuration from the current one.

- Swap the position of two supercells at the same die
- Move the position of a supercell at the die
- Change the aspect ratio from one supercell
- Move one supercell to another die
- Swap two supercells between different dies
- Move the point of origin.

In the following the pseudo code of Algorithm 3 for the stochastic optimization method is explained:

Algorithm 3 Pseudo code for stochastic optimization techniques.

```
generate start solution l_0
i=0
while stop criterion is false do
  generate neighborhood solution l_N
  generate random number r
  if r ≤ P(accept l_N) then
    l_{i+1} = l_N
  else
    l_{i+1} = l_i
  end if
  i = i + 1
end while
```

At first a start configuration is chosen randomly. Following, in every optimization step a neighboring configuration l_N is generated by selecting and executing one of the six previous described operations on the current solution l_i. This solution would be accepted with a given probability.

The algorithm ends if the stop criterion is fulfilled. Different stop criterions are possible, for example ending either if a given number of iterations is reached ($i = N_{stop}$) or if almost all of the neighboring solutions are discarded.

The algorithms simulated annealing, threshold accepting an great deluge algorithm vary only in $P(\text{accept } l_N)$. It is clear that $P(\text{accept } l_N) = 1$ if the new solution is better than the current. But to overcome local minima it should be possible to accept worse configurations. So there are cases with $P(\text{accept } l_N) > 0$ although l_N is worse the l_i in the algorithms.

The value thresholds describe that the accepting barrier is lowered. In early phases this barrier is high and the search is not influenced by this. In later phases this barrier is lowered to zero and the search accepted only configurations with even or better costs.

The number of cycles for every threshold is determined empirical and in the order of 10000 to 100000.

E. Floorplanning update

One advantage of the stochastic optimization techniques is there opportunity to restart the algorithm on an older solution. This opportunity can be used for the update of the floorplan. Such an update is done by a restart of the stochastic optimization algorithm. The other advantage of the stochastic optimization techniques is there opportunity to balance between accuracy and runtime by justify the number of steps. This opportunity is also used by the update of the floorplan because in this case only some steps for the stochastic optimization algorithm are necessary.

IV. Results

The flow can't be tested and the well-known MCNC and GSRC benchmarks because for this benchmarks no cell in-

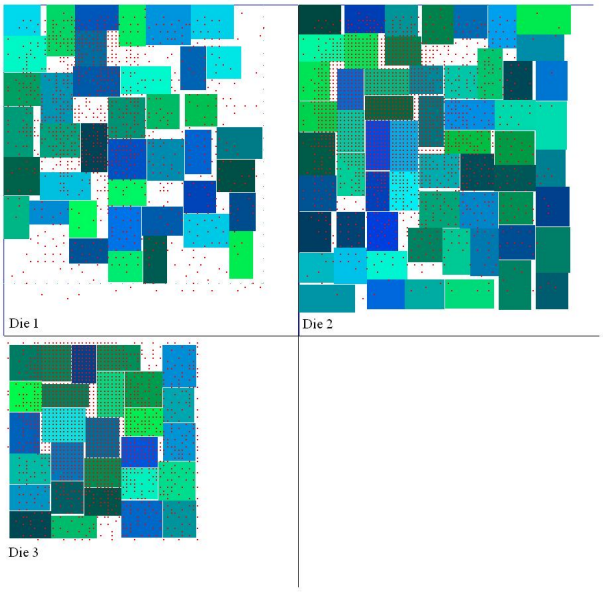

Fig. 5. Resulting layout for the VLIW-Processor example.

	SA		TA		GD	
Steps	TSVs	Delay	TSVs	Delay	TSVs	Delay
1	2470	2.97	2470	2.97	2470	2.97
5	2323	2.94	2324	2.94	2455	2.95
10	2286	2.89	2285	2.89	2436	2.93
15	2267	2.87	2267	2.87	2416	2.91
20	2256	2.87	2291	2.85	2393	2.89
25	2243	2.84	2276	2.83	2374	2.89
30	2211	2.84	2265	2.81	2364	2.87

TABLE I
RESULTS OF THE SYNTHESIS OPTIMIZATION FOR THE STOCHASTIC FLOORPLANNING METHODS SIMULATED ANNEALING (SA), THRESHOLD ACCEPTING (TA) AND GREAT DELUGE (GD).

formation or RTL-Code is available. However, the flow was tested on a high-scalable VLIW-processor. In a 2D-system the processor was placed at a die with $700.000um^2$ and total costs of 100 units. With the proposed method, the same system can be placed at a three dies stacked with a total area of $643.000um^2$. As it is shown in Figure 5 the cost optimal system consists of dies with different size. The dies cost 35 + 35 + 15 units. With additional 10 units for TSV insertion and stacking the total manufacturing costs are with 95 units below the 2D implementation. The results for the timing and the number of TSVs and the different stochastic floorplan optimization techniques are shown in table I. If it is shown the repeated using of the optimization reduces the number of TSVs and also the timing. The reduction of the timing is at most a result of the reduced number of TSVs which have a significant delay. Also it is shown that threshold accepting gives the best result for this example. In the table is also shown that after some steps there is no progress in the reduction of the timing or the number of TSVs. At this point the optimization loop can be canceled.

The longest path in the 2D-system has a delay of 3.0ps whereas in 3D the delay was reduced by 0.2ps to 2.8ps.

V. CONCLUSION AND OUTLOOK

A new layout dependent 3D synthesis method and there inclusion into a 3D design flow presented. It could be shown that 3D integration can provide advantages for large designs. In the future we will search for other configuration which also has a significant influence to the timing or the number of TSVs.

REFERENCES

[1] I. Limansyah, M. J. Wolf, A. Klumpp, K.Zoschke, R. Wieland, M. Klein, H. Oppermann, L. Nebrich, A. Heinig, A. Pechlaner, H. Reichl, and W. Weber, "3d image sensor sip with tsv silicon interposer," *ECTC 2009*, 2009.

[2] T. Karnik, "3d architectures and cad tools," *D43D Workshop*, 2010.

[3] L.Cheng, L.Deng, and M. Wong, "Floorplanning for 3d vlsi design," *Proceedings of IEEE/ACM ASP-DAC 2005*, pp. 504–511, 2005.

[4] J. Cong, G. Luo, J. Wei, and Y. Zhang, "Thermal-aware 3d ic placement via transformation," in *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 780–785. [Online]. Available: http://dx.doi.org/10.1109/ASPDAC.2007.358084

[5] T. Kutzschebauch and L. Stok, "Congestion aware layout driven logic synthesis," in *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, 2001, pp. 216 –223.

[6] ——, "Layout driven decomposition with congestion consideration," in *Proceedings of the conference on Design, automation and test in Europe*, ser. DATE '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 672–. [Online]. Available: http://dl.acm.org/citation.cfm?id=882452.874539

[7] D. Pandini, L. Pileggi, and A. Strojwas, "Global and local congestion optimization in technology mapping," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 498 – 505, apr 2003.

[8] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Mulitlevel hypergraph partitioning: Applications in vlsi domain," *34th Design and Automation Conference*, pp. 526 – 529, 1997.

[9] J.Cong and G.Luo, "A multilevel analytical placement approach for 3d ics," *Proceedings of the 14th ASP-DAC*, pp. 361–373, 2004.

[10] X. Dong and Y. Xie, "System-level cost analysis and design exploration for three-dimensional integrated circuits (3d ics)," *Asia and South Pacific Design Automation Conference*, pp. 234–241, Januar 2009.

[11] J. H. Lau, "Tsv manufacturing yield and hidden costs for 3d ic integration," *60th Electronic Components and Technology Conference*, pp. 1031–1042, June 2010.

[12] Y. Chen, D. Niu, Y.Xie, and K. Chakrabarty, "Cost-effective integration of three-dimensional (3d) ics emphasizing testing cost analysis," *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 471–476, November 2010.

[13] R. Weerasekera, L. Zheng, D. Pamunuwa, and H. Tenhunen, "Extending system-on-chip to the third dimension: Performance, cost and technological tradeoffs," *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 212–219, November 2007.

[14] Y.Ma, S. X.Hong, and C.K.Cheng, "3d cbl: an efficent algorithm for general 3dimensional packing problems," *Proceedings of the 48th MWSCAS*, pp. 1079–1082, 2005.

[15] S. H.Yamazaki, K.Sakanushi and Y.Kajitani, "The 3d-packaing by a meta data structure and packing heuristics," *IEICE Transactions on Fundamentals*, pp. 639–645, 2000.

[16] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, vol. 90, pp. 161–175, September 1990.

[17] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York: John Wiley and Sons, 1990.

[18] G. Dueck, "New optimization heuristics – the great deluge algorithm and the record-to-record travel," in *Technical reports*. Heidelberg: IBM Germany, 1989.