Through-Silicon-Via Management during 3D Physical Design: When to Add and How Many?

Mohit Pathak, Young-Joon Lee, Thomas Moon, and Sung Kyu Lim
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, Georgia, USA
{mohitp, yjlee, tmoon, limsk}@gatech.edu

Abstract— In 3D integrated circuits through silicon vias (TSVs) are used to connect different dies stacked on top of each other. These TSVs occupy silicon area and have significantly larger area than regular gates. In this paper, we address two critical aspects of TSV management in 3D designs. First, we address the problem of how many TSVs to add in a design. Since TSVs occupy significant silicon area, a general tendency has been to use a minimum number of TSVs in 3D circuits. We show that such an approach does not give us the best possible result. Second, we address the problem of TSV insertion. Because TSVs occupy silicon area, their location is decided during the placement stage of 3D design. However, we show that this is not the best possible stage for TSV insertion. We propose a change in the physical design flow for 3D integrated circuits to address the limitations of existing TSV placement methodology. All our algorithms are integrated with commercial tools, and our results are validated based on actual GDSII layouts. Our experimental results show the effectiveness of our methods.

I. INTRODUCTION

Technology scaling has enabled smaller and faster devices thus increasing integration density and decreasing intrinsic gate delay. Higher integration density requires a greater number of longer interconnects. Therefore, as the device delay is reduced, the performance of the circuit becomes dominated by the interconnect delay. In addition, other interconnect related issues, such as power consumption and signal integrity, have become more pronounced with technology scaling. To manage these issues three dimensional integration has been proposed [1]. The emerging 3D die stack technology enables the integration of multiple planar integrated circuits in the vertical direction with high density interconnect using TSVs.

Various methods exist for the fabrication of TSVs for stacked 3D ICs [1]. The two most popular methods are (i) via first, and (ii) via last. In the via first method, blind vias, which do not go all the way through the wafer, are formed in the wafer before transistors are created or immediately after. The wafer is thinned from the backside until the TSVs are exposed, then the next wafer can be attached to the back of the thinned wafer. In the via last approach, individual chips with transistors and interconnects already formed are stacked on top of each other. The vias are then etched from the back of the face-down chip reaching down to the bond pads of the front. TSVs with 2\(\mu\)m diameter and 7\(\mu\)m pitch have been fabricated using the via last method thus allowing large number of TSVs [1]. TSVs are several times larger than logic gates and memory cells as illustrated in Figure 1. Therefore it is critical to consider the impact of TSVs while designing 3D ICs. In this paper we consider the via first fabrication process.

A large amount of work has been done on placement in 3D integrated circuits. The authors of [2] extend the analytical placement in 2D ICs to 3D ICs, however they do not consider the area impact of through silicon vias resulting in a large number of TSVs. The works done by authors in [3], [4] and [5] also ignore the impact of through silicon vias on area and report unacceptably large via count. The authors in [6] discuss the impact of partitioning on 3D floor planning and the authors in [7] use partitioning driven 3D IC placement. However, both these works fail to show the impact of through silicon via size on 3D ICs. The authors in [8] provide the first detailed placement and routing results for 3D ICs that demonstrates GDSII based layout. The authors show that through silicon vias can have a significant impact on the area and wire length of a circuit. They first perform a simple partitioning of the circuit followed by force directed 3D placement. In some cases the authors claim that 3D ICs result in wire-length worse than 2D ICs. However, they do not provide details of how they perform initial circuit partitioning into different dies.

3D technology can help in reducing the problem of long interconnect lengths in 2D ICs. However determining the number and locations of TSVs is very important. TSVs typically have much larger size than standard cells. Thus using too
many TSVs can actually cause performance to deteriorate.
Partitioning is a critical stage in the design of 3D integrated circuits as it determines the number of 3D nets (nets that have pins in multiple dies) present in the design. Having too few 3D nets may result in under exploitation of the advantages of 3D die stacking. On the other hand having too many 3D nets may have a negative impact due to the area occupied by TSVs. Thus finding the right amount of TSVs is critical in 3D IC design. We also show that for 3D nets the number of TSVs used per net is important and using just one TSV per die for each 3D net does not give the best results. The contributions of this paper are as follows:

- We demonstrate the impact of TSV management and 3D design on gate level and multi-core level 3D partitioned circuits.
- We examine different 3D min-cut exploration scheme to determine the number 3D nets (nets that have pins in multiple dies) in a design.
- We propose a change in the physical design flow to determine the location of through silicon vias.
- We provide accurate results for various metrics such as wire-length and timing and our results are based on an actual GDSII layouts.

II. MOTIVATION

TSVs are used to connect signals between different dies stacked on top of each other in 3D designs. These TSVs occupy significant silicon area (25\(\mu\)m\(^2\) for 5\(\mu\)m \(\times\) 5\(\mu\)m TSVs) as compared to standard cells (2.5\(\mu\)m\(^2\) each). Thus it is critical to effectively determine how many TSVs to use and where to place them.

The number of TSVs used is dependent on two things (i) the number of 3D nets (nets that have pins in multiple dies), and (ii) the number of TSVs used for each net. At one extreme we can minimize the number of 3D nets; this means we would use the least amount of silicon area for TSVs. At the other extreme we can have a large number of 3D nets; this would require more silicon area reserved for TSVs. In Section III we show how to explore and best determine the number of 3D nets in a design.

Another important aspect of TSV management is to determine how many TSVs should be used for a given 3D net and where to place them. In [8] the location of TSVs is determined during placement. The total number of TSVs is determined by partitioning and then TSVs are placed together with the gates. However, the location of TSVs are best determined once the location of gates are already known. Even for 2-pin nets, placing TSVs with gates may produce a bad solution. This happens because the TSV is placed outside the bounding box of the standard cells. While additional constraints may be added to keep the TSV within the bounding box of the gates, such constraints are not easy to add and may spoil the quality of the placement solution. Another approach could be to place TSVs during routing. The benefit in this case is that the location of gates is known. However, the major challenge is the impact of TSVs on the layout itself. Since TSVs are larger than gates, TSV’s may be inserted only in the white space. However, there is no guarantee that white space is available at the desired location during the routing stage as the location of gates is fixed. To overcome this difficulty we propose a two stage approach which changes the physical design flow for 3D integrated circuits slightly as compared to 2D circuits. In the new design flow we first perform global placement of gates only, which gives us the rough location of gates. We then perform global 3D routing where, TSVs are inserted during routing topology generation. Finally, we perform detailed placement followed by detailed routing. This approach allows us to determine the location of TSVs more accurately and also allows us to determine the number of TSVs to be used for each 3D net.

III. PARTITIONING DRIVEN PLACEMENT AND 3D NETS

Placement by recursive partitioning is one of the oldest approaches to the placement problem in VLSI design. A great deal of work has been done previously [9], [10], [11]. The placement is created by performing a series of cuts along different directions to divide the problem into smaller sub-problems until the required granularity is achieved. Traditional 2D placement consist of \(V\)-cut (cut along y-axis) and \(H\)-cut
(cut along x-axis) which divide a region into smaller sub-regions. In 3D integrated circuits an additional cut direction exists which divides a given 2D die into multiple dies called Z-cut. Figure 3 shows the possible cut directions that may exist for 3D ICs. In the following discussion we show that the order of the cut operations is extremely important for 3D designs.

At one extreme we can do the Z-cut first, followed by a sequence of V-cuts and H-cuts, such a cut-sequence would produce the minimum number of 3D nets. At the other extreme we can perform all the H-cuts and V-cuts and do the Z-cut at the end, such a cut-sequence would produce the maximum number of 3D nets. We show that for 3D designs the best solution can exist between these two extremes. We first perform a set of experiments to show that indeed for a large class of circuits there may exist a cut sequence between the two extremes as discussed above that may provide us with the best results. Later we propose a quick exploration scheme which helps us choose the best cut sequence.

A. Impact of cut sequences

In this section we explore the impact of cut-sequences on global placement of 3D ICs. A global placement for a 3D IC with \( R \) rows, \( C \) columns and \( D \) layers can be obtained by a cut sequence that may look like \( \text{V,H,V...H,...Z,...V...H} \). Where the number of V-cuts is equal to \( C \) and the number of H-cuts is equal to \( R \). The Z cut represents a series of cuts which divides the current 2D partitioning into \( D \) dies. Figure 3 shows two examples of when the Z-cut could be performed.

The authors in [12] showed how optimal cut sequences could be generated for partitioning-driven placement based on using Rent’s rule to approximate the number of nets cut. Rent’ rule states that on average, a block of \( C \) cells will have \( T = k \times C_P \) propagated terminals. \( k \) is the average cardinality of cells in the net-list and \( p \) is an experimentally derived constant (generally between 0.3 and 0.7). Based on Rent’s rule the authors determined that if a block containing \( C_0 \) cells is divided into two sub-regions containing \( C_1 \) and \( C_2 \) cells then the number of nets split between them is given by

\[
Nets_{\text{cut}} = 0.5 \times k \times (C_P^0 + C_P^2 - C_P^0)
\]

The derivation was discussed in [12]. To determine the wire-length of the cut nets an approximation function was used based on the width and height of the region being sub-divided. The main result derived by [12] was that the best results were based on using Rent’s rule to approximate the number of nets cut. Based on our discussion in the previous section, we observe that performing the Z-cut first may not be the best way to perform partitioning-driven placement for 3D ICs. However, we cannot use Rent’s rule and the model described above directly to obtain the best cut sequence. This is because the Rent’s parameters \( k \) and \( p \) can vary for each circuit and even

![Fig. 4](image1.png)

Fig. 4. The number of nets in 3D can be approximated based on the previously performed x-y cuts. The total number of 3D nets is approximately given by the sum of the nets cut in partitions A,B,C and D.

![Fig. 5](image2.png)

Fig. 5. The impact on wire length of using Rent’s-rule-based estimation for 3D circuits. In terms of gate count, ckt1 is smallest, whereas ckt5 is largest.

on the number of 3D nets we can estimate the area needed for the TSVs. We assume this area is spread uniformly across the die.

In our experiments we change the order in which the Z-cut is performed. To choose between the V-cut and the H-cut we use the observations made in [12]. If the number of available rows in a region is greater, then we perform an H-cut, otherwise we perform a V-cut. The authors in [12] showed that such a cut orientation tends to give the best results. We thus compute the total wire-length of a 3D design based on when the Z-cut is performed. The results are shown in Figure 5. We observe that, for all benchmarks performing the Z-cut first does not give the best results. However, doing too many \( V \) and \( H \) cuts may also degrade wire length due to the area impact of the TSV’s. In addition we observe that the number of cuts needed to obtain the optimum location increases with larger circuit size.

B. Early exploration for 3D IC cut sequence

Based on our discussion in the previous section, we observe that performing the Z-cut first may not be the best way to perform partitioning-driven placement for 3D ICs. However, we cannot use Rent’s rule and the model described above directly to obtain the best cut sequence. This is because the Rent’s parameters \( k \) and \( p \) can vary for each circuit and even
a small error in the value of these parameters results in a big error in actual results. We thus propose an early exploration method to determine which cut sequence gives the best results.

Assuming we need a global placement with \( R \) rows, \( C \) columns and \( D \) dies. We generate multiple global placement solutions for a given circuit. The way we generate these global placement solutions vary on the number of cuts we perform along the x and y axes before we perform a Z-cut. For example, to generate a global placement of size \( 32 \times 32 \times 4 \) (i.e. 32 rows, 32 columns and 4 dies). We first generate multiple solutions along x and y dimensions of size \( 1 \times 1, 2 \times 2, 4 \times 4, 8 \times 8 \), and \( 16 \times 16 \) where \( r_i \times c_i \) represents the number of rows and columns generated before the net-list is divided into multiple dies. We then perform cuts along the z-direction and then finally perform the final cuts to obtain the desired global placement.

Once we have these multiple global placement results we can perform a quick global routing to determine the quality of the solution obtained. In addition, rough timing analysis could also be performed at this stage to further check the quality of the solution obtained.

IV. TSV Placement

In this section we discuss our TSV placement flow. The traditional physical synthesis flow requires placement to be done before routing. We showed in Section II that such an approach suffers from several drawbacks. The first concern is that at this stage, only the minimum possible number of TSVs can be added for each net. Figure 2 shows a case where this may produce a bad result. In addition, the location of the TSVs may lay outside the bounding boxes of the standard cells, thus leading to bad results. In this section we propose a new physical synthesis 3D design flow that helps in overcoming these limitations. In our new flow, we perform (1) global placement, (2) global routing, (3) detailed placement, and (4) detailed routing in this sequence. Performing global placement in step (1) provides us with the rough location of standard cells. Based on this rough location we can perform global routing in step (2). The location of the TSVs and their number is determined during routing topology generation for 3D nets. Detailed placement in step (3) followed by detailed routing in step (4) is then performed based on the global placement and routing results obtained above.

In the first stage we perform global placement considering only the standard cells in the design. After global placement is finished, a global routing graph \( G = (V, E) \) is generated, where \( v \in V \) corresponds to each global placement bin, and \( e \in E \) corresponds to edges connecting these bins. The capacity of the horizontal and vertical edges is based on the capacity of the metal layers present in the region. The capacity of the edges in the z-direction (i.e. connecting bins in different dies) is based on the whitespace available for TSV placement in the bin. The weight associated with all edges is based on the capacitance associated with each edge. For horizontal and vertical edges this is based on the length of the edge and unit capacitance of the wire connecting them, whereas for z-direction edges it is based on the capacitance of the TSV.

Global routing is performed one net at a time. For each net a Steiner tree is constructed based on the minimum spanning tree (MST) of that net. Once the MST has been generated, pairs of nodes in the MST are connected one at a time using a maze router. Different weights on z-direction edges can control the numbers of TSVs used for each 3D net.

V. Multi-Core Design

In this section we briefly describe the details of our multi-core design. Our target design consists of 16 processor cores of LEON3 [13]. The LEON3 processor is a configurable and synthesizable 32-bit processor compliant with the SPARC V8 architecture. It contains an advanced 7-stage pipeline with a hardware multiply, divide, and multiply-accumulate (MAC). The LEON3 design also has configurable caches and local instruction and data scratch memories. We used a single configuration for all the cores, which is described in Table I. In this work, we set up the design to have 16 cores connected by AMBA (Advanced Micro-controller Bus Architecture) buses. In addition we have a memory controller for the SDRAM (Synchronous Dynamic Random Access Memory) and an AMBA bus controller. The 16 processor cores act as masters and the memory controller acts as a slave on the AMBA AHB (Advanced High Performance Bus) bus. The arbiter in the AMBA bus controller determines which master gets access and broadcasts its selection to all the slaves. Before a bus transfer is started, a master must be granted access to the bus. A prioritization scheme is used to choose a master requesting the bus. In our design, round-robin is used for the prioritization scheme. The arbiter has an interface with every other master using request and grant signals.

In the 2D layout of the 16-core design, the cores are placed as macro blocks on a single die. The standard cells, including the memory controller and the AMBA bus controller, are placed by Cadence Encounter. Most of the standard cells are placed around the center as shown in Figure 6(a). In the 3D layout of the design, it is partitioned at the core level. Each die has 4 cores and the total number of dies used is 4. The layout for the 3D design is shown in Figure 6(b). The standard cells of the AMBA bus controller and the memory controller are all located on Die3 as shown in Figure 6(b). All global IO signals are present in Die1. Signals generated by the bus controller may need to connect to all the cores. These signals are propagated across dies using TSVs.

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARCHITECTURE CONFIGURATION FOR THE LEON3 DESIGN.</strong></td>
</tr>
<tr>
<td>Number of cores</td>
</tr>
<tr>
<td>Instruction cache</td>
</tr>
<tr>
<td>Data cache</td>
</tr>
<tr>
<td>Register file</td>
</tr>
<tr>
<td>Multiplier/Divider</td>
</tr>
<tr>
<td>Core-core communication</td>
</tr>
</tbody>
</table>
VI. COMMERCIAL TOOL INTEGRATION

In this section we discuss the details of the integration of our placement and routing driven package with the commercial tool to obtain detailed placement and routing. Once we obtain global placement results from our partitioning driven placement we perform detailed placement and routing for each die using the commercial tool (Cadence Encounter) one die at a time.

We begin the placement of the dies starting from the top-most die. Through silicon vias are represented by a macro cell in our 45nm LEF (Library Exchange Format) files. Once the top-most die has been placed, the die below it is placed.

A. 3D timing analysis

Our 3D static timing analysis (STA) is performed using Synopsys PrimeTime. First, we prepare the Verilog net-list files of all four dies and the SPEF files containing extracted parasitic values for all the nets of the dies. Then, we create a top-level Verilog netlist that instantiates each die’s design and connects the 3D nets using TSV connections. We also create a top level SPEF file that has a parasitic model of the TSVs. After that, we run PrimeTime to get the 3D timing analysis results.

B. 3D timing budgeting

Timing budgeting is the process of allocating the path slack to the timing graph edges of the path. We use Synopsys Design Compiler to perform 3D timing budgeting. Since timing budgeting is done on the timing graph, the physical dimension (2D vs. 3D) does not make any difference. The 3D system can be considered as a hierarchical design where each die is a sub-design of the whole.

C. 3D timing optimization

Timing optimization is performed on each die separately using Cadence Encounter. The timing constraint file generated in the budgeting step is used for each die. These constraints include setting up (i) correct load capacitance and required signal delays on output ports, and (ii) signal arrival times and driving cell information on input ports. Once each die has been optimized we perform 3D timing analysis and parasitic extraction. The process of timing budgeting and optimization is then repeated until we observe negligible improvement.

VII. EXPERIMENTAL RESULTS

In this section we discuss in detail our experimental results. We use IWLS [12] 2005 benchmarks and several industrial circuits. The details of the circuits are shown in Table II. We also use 45nm technology and TSV cell size of $4.94 \mu m \times 4.94 \mu m$. The size of the TSV was chosen such that its height is equal to the height of two standard cell rows.

A. Impact of Cut Sequences on 3D ICs

In this section we discuss the impact of cut sequences on wire-length and performance of 3D ICs. We perform placement for all circuits on a placement grid with size $32 \times 32 \times 4$ (32 rows, 32 columns and 4 dies). We generate different cut sequences with size $r_i \times c_i$ where $r_i$ represents the number of rows and $c_i$ represents the number of columns generated before the net-list is divided into multiple dies. We first compare the number of TSVs used with different cut sequences for each circuit. The results are shown in Figure 7, which plots the number of TSVs used versus the corresponding cut sequences for different circuits. As expected by our discussion in Section III the number of TSVs needed increases as we perform more cuts in the x-y direction before performing cuts in the z-direction.

We plot the variation of wire-length and longest path delay (normalized with respect to their corresponding 2D design
values). The results are shown in Figure 8. We observe that for all designs initially as the number of TSVs increases we see an improvement in wire length and performance. However, as the number of TSVs becomes too large, the area impact of TSVs causes wire length and performance to degrade. The delay numbers were calculated based on TSV parasitic resistance of $10\Omega$, and capacitance of $25\, \text{fF}$. We observe that all circuits tend to show a region where a given number of TSVs leads to the best results. Based on our results we observe that cut-sequences can have a significant impact on 3D IC wire-length and performance metric. In addition we show that using the minimum number of TSVs is not the best possible approach for 3D ICs and our exploration method can help in identifying the best possible solution for 3D designs.

B. Impact of Multiple TSVs

In this section we compare the impact of using a single TSV versus multiple TSVs on wire-length and performance of 3D circuits. In single TSV mode we allow a single TSV across die-pairs for a 3D net, whereas for multiple TSVs we allow multiple TSVs. We show the effect on wire-length for 3D nets and on the longest path delay for the entire design. The results are shown in Figure 9. The results for multiple TSVs are normalized with respect to the results for single TSV.

We observe that using multiple TSVs helps in reducing wire-length for 3D nets. This shows the advantage of our proposed method to perform global placement, followed by global routing to determine location and number of TSVs used. It should be noted that the authors of [8] used the more traditional flow where TSV and gate placement is done before routing. This allows the flow in [8] to use a single TSV for a 3D net. We observe wire length is reduced by 4 to 20% for 3D nets. The wire length reduction largely happens due to the presence of multi-pin 3D nets; for two-pin 3D nets a single TSV is sufficient. In addition we observe that using multiple TSVs helps in improving the performance of most designs significantly.

C. Comparison of 3D vs 2D Designs

In this section we compare our 3D (partitioning based) results with our 2D (partitioning based), and commercial 2D tool (Cadence Encounter) to show the advantages of 3D design. We use wire-length and longest path delay to compare
our results. The results are shown in Figure 10. To compute the timing for 3D we use the TSV parasitic as follows: 10Ω, 25fF (tsv). We observe that for all test cases the commercial 2D beats our 3D results in terms of wire-length, except for ckt5. In addition the commercial 2D beats our 2D placement results in all circuits with difference in wire length ranging from 2 to 35%. However for larger circuits we observe that our 3D beats our 2D in terms of wire length by about 10 to 15%.

In terms of longest path delay before optimization, Figure 10 (b), shows that commercial 2D beats our 2D in all cases except ckt5. In addition, for the larger circuits (ckt2, ckt3, ckt4, ckt5) our 3D timing numbers are significantly better than those of the commercial 2D (40 to 80%).

As seen in our results, 3D design shows no timing improvement for ckt1 with 170K gates in it. However, for larger designs 3D circuits can show significant improvement in timing numbers. While 3D layouts show about 10 to 15% reduction as compared to our 2D, the change in longest path delay is much higher. We believe this happens because the reduction in global wire length is much higher for 3D than 2D. A similar trend is seen when comparing 3D to commercial 2D, while the wire-length in 3D is higher in some test circuits (ckt2, ckt3, ckt4), the longest path delay is significantly smaller due to smaller global interconnect lengths. We believe that even though 3D may not show significant improvement compared to 2D in terms of total wire length, it can show a significant advantage in terms of global wires and performance. In addition, to make the best use of 3D, the circuit size needs to be large because for small designs the area penalty of TSVs can cause 3D to perform worse than 2D (ckt1).

We finally show the results obtained after timing optimization in Figure 10 (c). To optimize the 2D design we used Cadence Encounter. To optimize the 3D design we used Synopsys Design Compiler to generate the timing budgeting constraints and Cadence Encounter to optimize each die individually, as discussed in Section VI. We observe that in this case the commercial 2D beats our 2D results in all circuits. In addition our 3D timing results are worse than our 2D and commercial 2D in almost all circuits. We strongly believe that the 3D layouts lose in terms of timing after optimization largely because the optimization is not true-3D aware. When the timing budgeting constraints are made using Synopsys Design Compiler the constraints are not aware of the white space available in the design. We discuss these results more in section VII-E

D. Multi-core Results

In this section we compare our results obtained for the multi-core design. We compare the results obtained using a 2D design and a 3D design. In addition, we show the impact that multiple TSVs have on wire length and delay for a 3D design. The 2D layout has a footprint of 6144µm × 6144µm, whereas the 3D layout has a footprint of 3544µm × 3544µm.

The wire-length numbers compared here are computed as the wire-length of the logic needed to connect the cores and does not include the wire-length of the cores which is the same across different cores. The wire length of the 2D design is \(3.49 \times 10^6\) whereas for 3D design it is \(2.16 \times 10^6\) a reduction
of 39%. The longest path delay for the 2D design is 71.84\text{n}s whereas for the 3D design it is 18.3\text{n}s, a reduction of 75%. The results are shown in Figure 11. The longest path delay after optimization for 2D design is 7.47\text{n}s, whereas for 3D the delay is 3.92\text{n}s. We can also see the impact of using multiple TSVs on the 3D design. Multiple TSVs reduce the longest path delay after optimization for the 3D design by 16\% and reduce the wire-length by 12\%. Figure 12 shows the layout of the top-die of 16 core LEON3 design implemented in 4-die 3D IC. We also show the layout of the top die ofckt2 implemented in 4-die 3D IC.

E. Gate-level vs Core-level Timing Results

In this section we briefly analyze the results obtained for timing analysis for gate-level and multi-core level circuits. As discussed in previous sections, we observe that longest path delay for both gate-level and multi-core level circuits tend to have better results in the 3D design as compared to 2D before optimization. Also, the multi-core circuit shows better timing results compared to the 2D design after optimization. However, for gate-level circuits the delay results are better for the 2D design after optimization. We believe that this happens due to two reasons. First the 16-core LEON3 design has a much larger foot-print size 6144\,\mu m \times 6144\,\mu m (in 2D) versus the largest gate level footprint 2800\,\mu m \times 2800\,\mu m (ckt5). As observed previously 3D designs tend to show better results with larger circuits. Secondly the timing optimization method used for optimizing 3D designs is not really 3D aware. The budgeting algorithm used in Section VI to set the timing constraints is not aware of the available white-space in different dies. In addition, when optimizing the 16-core LEON3 design the budgeting algorithm looks at a very small net-list (only a few thousand nets) that connects the different cores together, at this stage of the design the individual core design is fixed and not subject to further changes. However, for gate-level circuits the budgeting algorithm sees the entire net-list (a few hundred thousand nets). This huge change in the complexity of net-list handled by the budgeting technique in Section VI can also have a significant impact. We firmly believe that by using a true 3D aware optimization engine the gate-level circuits would have shown better results after optimization.

VIII. Conclusion

In this paper we showed how to manage TSVs for 3D ICs. We demonstrated a method for controlling the number of TSVs in a 3D IC to improve its performance, and proposed a modification to the physical synthesis flow to find a good way to place the TSVs. The effectiveness of our methods is demonstrated in our experimental results. Future work is to find a more rigorous method to determine the number of TSVs needed in a particular 3D design.

REFERENCES


Fig. 12. Layout (full and close-up shots) of the top-die of ckt2 (= FFT) and 16 core LEON3 design implemented in 4-die 3D IC