

Chapter 9

3D Network on Chip Topology Synthesis: Designing Custom Topologies for Chip Stacks

Ciprian Seiculescu, Srinivasan Murali, Luca Benini and Giovanni De Micheli

9.1 Introduction

Today, many integrated circuits contain several processor cores, memories, hardware cores and analog components integrated on the same chip. Such *Systems on Chips* are widely used in high volume and high-end applications, ranging from multimedia, wired and wireless communication systems to aerospace and defense applications. As the number of cores integrated on a SoC increases with technology scaling, the two-dimensional chip fabrication technology is facing lot of challenges in utilizing the exponentially growing number of transistors.

As the number of transistors and the die size of the chip increase, the length of the interconnects also increases. With smaller feature sizes, the performance of the transistors have increased dramatically. However, the performance improvement of interconnects has not kept pace with that of the transistors [1]. With reducing geometries, the wire pitch and cross section also reduce, thereby increasing the RC delay of the wires. This coupled with increasing interconnect length leads to long timing delays on global wires. For example, in advanced technologies, long global wires could require up to 10 clock cycles for traversal [2]. Another major impact of increased lengths and RC values is that the power consumption of global interconnects become significant, thereby posing a big challenge for system designers.

9.1.1 3D-Stacking

Recently, 3D-stacking of silicon layers has emerged as a promising solution that addresses some of the major challenges in today's 2D designs [1, 3–8]. In the 3D

C. Seiculescu (✉)

Doctoral-assistent in Integrated Systems Laboratory, Swiss Federal Institute of Technology Lausanne (EPFL),
EPFL IC ISIM LS11 INF 339 (Bâtiment INF),
Station 14, 1015 Lausanne, Switzerland
Tel.: +41 21 693 0916
e-mail: ciprian.seiculescu@epfl.ch

stacked technology, the design is partitioned into multiple blocks, with each block implemented on a separate silicon layer. The silicon layers are stacked on top of each other. Each silicon layer has multiple metal layers for routing of horizontal wires. Unlike the 3D packaging solutions that have been around for a long time (such as the traditional system-in-package), the different silicon layers are connected by means of on-chip interconnects.

The 3D-stacking technology has several major advantages: (1) the foot-print on each layer is smaller, thus leading to more compact chips (2) smaller footprints lead to shorter wires within each layer. Inter layer connections are obtained using efficient vertical connections, thereby leading to lower delay and power consumption on the interconnect architecture (3) allows integration of diverse technologies, as each could be designed as a separate layer. A detailed study of the properties and advantages of 3D interconnects is presented in [1] and [9].

There are several methods for performing 3D integration of silicon layers, such as the *Die-to-Die*, *Die-to-Wafer* and *Wafer-to-Wafer* bonding processes. In the *Die-to-Die* bonding process, individual dies are glued together to form the 3D-IC. In the *Die-to-Wafer* process, individual dies are stacked on top of dies which are still not cut from the wafer. The advantages of these processes are that the wafers on which the different layers of the 3D stack are produced can be of different size. Another advantage is that the individual dies can be tested before the stacking process and only “known-good-dies” can be used, thereby increasing the yield of the 3D-IC. In the *Wafer-to-Wafer* bonding, full wafers are bonded together. The vertical interconnection is usually achieved using *Through Silicon Vias (TSVs)*. For connection from one layer to another, a TSV is created in the upper layer and the vertical interconnect passes through the via from the top layer to the bottom layer. Connections across non-adjacent layers could also be achieved by using TSVs at each intermediate layer. The integration of the different layers could be done with either face to face or face to back topologies [10]. A die’s face is considered to the metal layers and the back is the silicon substrate. The copper half of the TSV is deposited on each die and the two dies are bonded using thermal compression. Typically, the dies are thinned to reduce the distance between the stacked layers. Several researches have addressed 3D technology and manufacturing issues [1, 4, 11]. Several industrial labs, CEA-LETI [12], IBM [13], IMEC [14] and Tezzaron [15], to name a few, are also actively developing methods for 3D integration.

In Fig. 9.1, we show a set of vertical wires using TSVs implemented in SOI and bulk silicon technologies [11]. We also show the schematic representation of a bundle of TSV vias in Fig. 9.2. In [11], a $4 \times 4 \mu\text{m}$ via cross section, $8 \mu\text{m}$ via pitch, $1 \mu\text{m}$ oxide thickness and $50 \mu\text{m}$ layer thickness are used.

The use of 3D technology introduces new opportunities and challenges. The technology should achieve a very high yield and commercial CAD tools should evolve to support 3D designs. Another major concern in 3D chips is about managing heat dissipation. In 2D chips, the heat sink is usually placed at the top of the chip. In 3D designs, the intermediate layers may not have a direct access to the heat sink to effectively dissipate the heat generated. Several researchers have been working on all these issues and several methods have been proposed to address them. For example, the problem of partitioning and floorplanning of designs for 3D integration has been

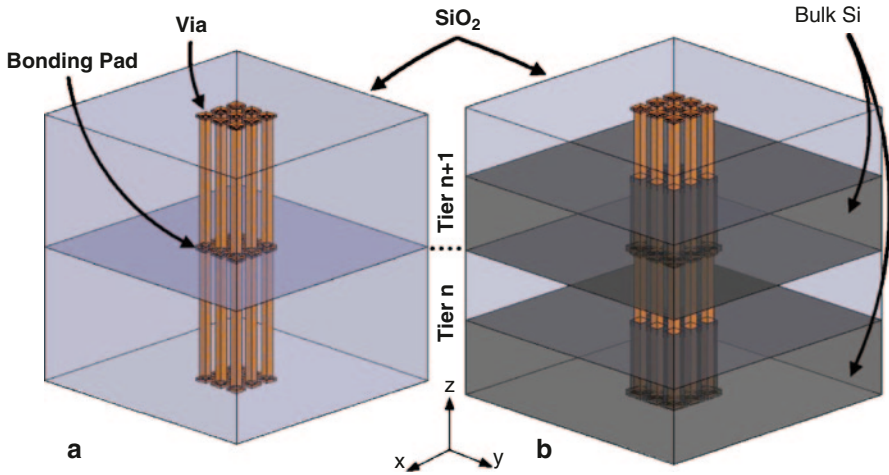


Fig. 9.1 An example set of nine vertical links

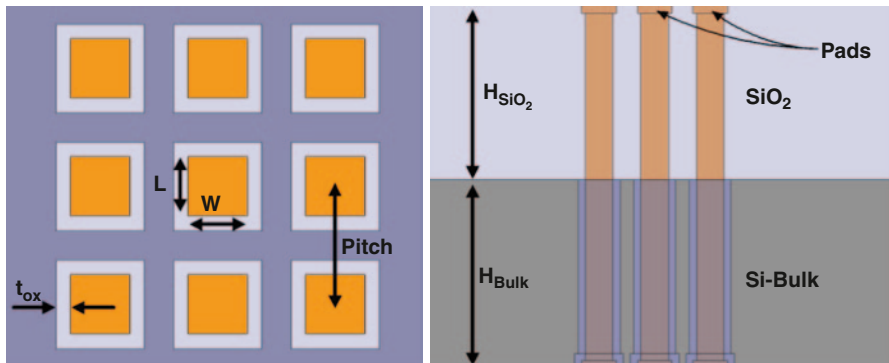


Fig. 9.2 3D bundle cross-section

addressed in [4–8]. Today, several 3D technologies have matured to provide high yield [16]. Many methods have been presented for achieving thermally efficient 3D systems. The methods span from architectural to technology level choices. At the architectural level, works have addressed efficient floorplanning to avoid thermal hot-spots in 3D designs [17]. At the circuit level, use of thermal vias for specifically conducting heat across the different silicon layers has been used [4]. In [13], use of liquid cooling across the different layers is presented.

9.1.2 Networks on Chips for 3D ICs

One of the major challenges that designers face today in 3D integration is how to achieve the interconnection across the components within a layer and across the layers

in a scalable and efficient manner. The use of *Networks on Chips (NoCs)* has emerged as the solution to the 3D integration problem. The NoC paradigm has recently evolved to address the communication issues on a chip [2, 18, 19]. NoCs consist of switches and links and use circuit or packet switching technology to transfer data inside a chip. NoCs have several advantages, including achieving faster design closure, higher performance and modularity. An example NoC architecture is shown in Fig. 9.3. A NoC consists of set of switches (or routers), links and interfaces that packetize data from the cores. A detailed introduction to NoC principles can be found in [19].

NoCs differ from macro-networks (such as the wide area networks) because of local proximity and predictable behavior. The on-chip networks should have low communication latency, power consumption and could be designed for particular application traffic characteristics. Unlike a macro-network, the latency inside a chip should be in the order of few clock cycles. Use of complex protocols will lead to large latencies, NoCs thereby require streamlined protocols. Power consumption is a major issue for SoCs. The on-chip network routers and links should be highly power efficient and occupy low area.

The use of NoCs in SoCs has been a gradual process, with the interconnects evolving from single bus structures to multiple buses with bridges, crossbars and a packet-switching network. Compared to traditional bus based systems, a network is clearly more scalable. Additional bandwidth can be obtained by adding more switches and links. Networks are inherently parallel in nature with distributed arbitration for resources. Thus, multiple transactions between cores take place in parallel in different parts of a NoC. Whereas, a bus-based system use centralized arbitra-

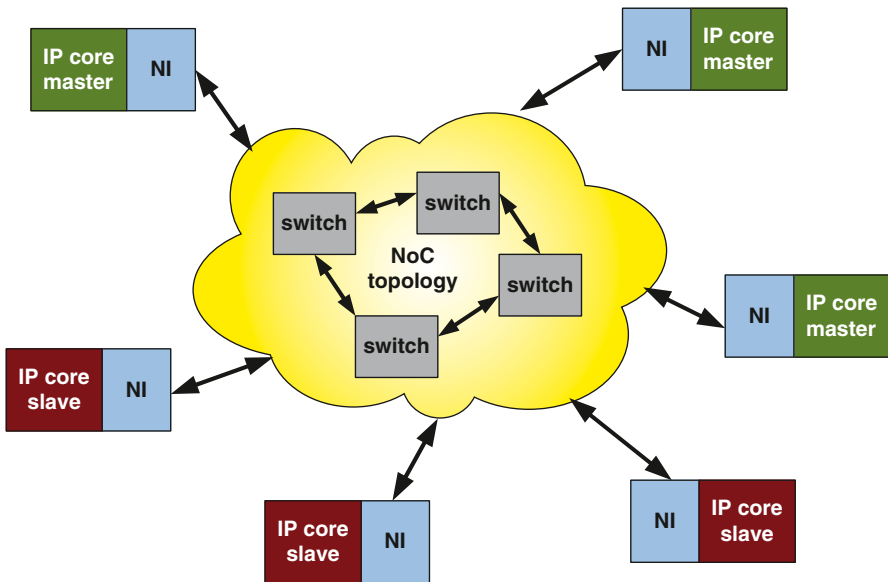


Fig. 9.3 Example NoC design

tion, thereby leading to large congestion. Also, the structure and wiring complexity can be well controlled in NoCs, leading to timing predictability and fast design closure. The switches segment long global wires and the load on the links are smaller, due to their point-to-point nature.

NoCs are a natural choice for 3D chips. A major feature of NoCs is that a large degree of freedom is available that can be exploited to meet the requirements. For example, the number of wires in a link (i.e. the link data width) can be tuned according to the application and architecture requirements. The data can be efficiently multiplexed on a small set of wires if needed. This is unlike bus based systems, that require several set of wires for address, data and control. Thus, communication across layers can be established with fewer vertical interconnects and TSVs. NoCs are also scalable, making the integration of different layers easy. Several different data streams from different sources and destinations can be transferred in parallel in a NoC, thereby increasing performance. The combined use of 3D integration technologies and NoCs introduces new opportunities and challenges for designers.

Several researchers are working on building NoC architectures for 3D SoCs. Router architectures tuned specifically for 3D technologies have been presented in [20] and [21]. Using NoCs for 3D multi-processors has been presented in [22]. Cost models for 3D NoCs, computed analytically has been presented in [23]. Designing regular topologies for 3D has been addressed in [24].

9.1.3 Designing NoCs for 3D ICs

Designing NoCs for 3D chips that are application-specific, with minimum power-delay is a major challenge. Successful deployment of NoCs require dedicated solutions that are tailored to specific application needs. Thus, the major challenge will be to design hardware-optimized, customizable platforms for each application domain. The designed NoC should satisfy the bandwidth and latency constraints of the different flows of the application.

Several works have addressed the design of bus based and interconnect architectures for 2D ICs [25, 26]. Several methods have addressed the mapping of cores on to NoC architectures [27–31]. Custom topology synthesis for 2D designs has been addressed in [32–39].

Compared to the synthesis of NoCs for 2D designs, the design for 3D systems present several unique challenges. The design process needs to support the constraints on the number of TSVs that can be established across any two layers. In some 3D technologies, only connections across adjacent layers can be supported, which needs to be considered. Finally, the layer assignment and placement of switches in each layer need to be performed.

The yield of a 3D IC can be affected by the number of TSVs used, depending on the technology. In Fig. 9.4, we show how the yield for different processes vary with the number of TSVs used across two layers. The graphs show a trend that after a threshold, the yield decreases with increasing number of TSVs. Thus, the topology

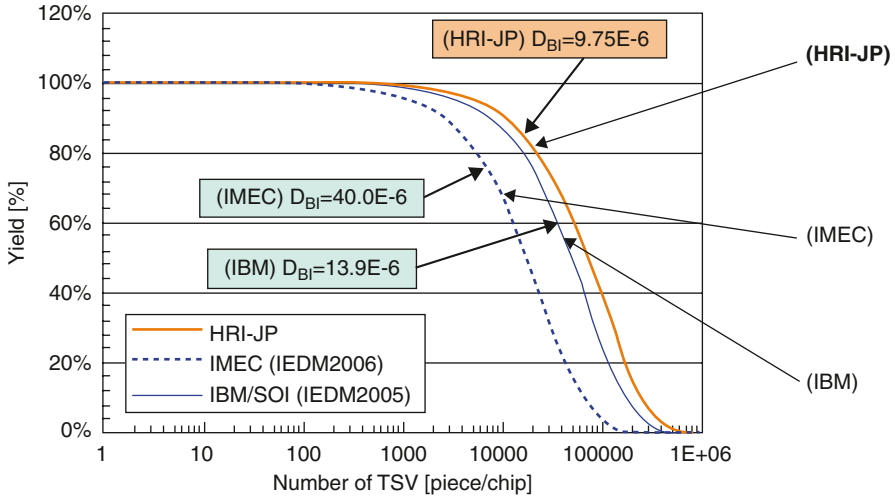


Fig. 9.4 Yield vs. TSV count

synthesis process should be able to design valid topologies meeting a specific yield and hence a TSV constraint. Moreover, with increasing TSV count, more area needs to be reserved for the TSV macros in each layer. Thus, to reduce area, a bound on allowed TSV is important. In [3], the pitch of a TSV is reported to be between 3 and 5 μm . Reserving area for too many TSVs can cause a considerable reduction in the active silicon area remaining for transistors.

The TSV constraint can significantly impact the outcome of the topology synthesis process. Intuitively, we can see that when more TSVs are permitted, more vertical links (or larger data widths) can be deployed. The resulting topologies could have lower latencies, while using more area for the TSVs. On the other hand, a tight TSV constraint would force fewer inter-layer links, thereby increasing congestion on such links and affecting performance. In Figs. 9.5 and 9.6, we show the best topologies synthesized by our flow for the same benchmark for two different TSV constraints. In the first case 13 interlayer links are used and in the second only eight inter-layer links are used.

Building power-efficient NoCs for 3D systems that satisfy the performance requirements of applications, while satisfying the technology constraints, is an important problem. To address this issue, new architectures and design methods are needed. In this chapter, we present synthesis methods for designing NoCs for 3D ICs. The objective of the synthesis process is to obtain the most power efficient topology for the NoC for the 3D design. The process has to meet the 3D design constraints and the application performance requirements. We also take the floorplan of each layer of the 3D design, without the interconnect, as an optional input to the design process. In our design process, we also compute the position of the switches in the floorplan and place them, while minimally perturbing the position of the other cores. We apply our methods to several SoC benchmarks which show large power and latency improvements when compared to the use of standard topologies.

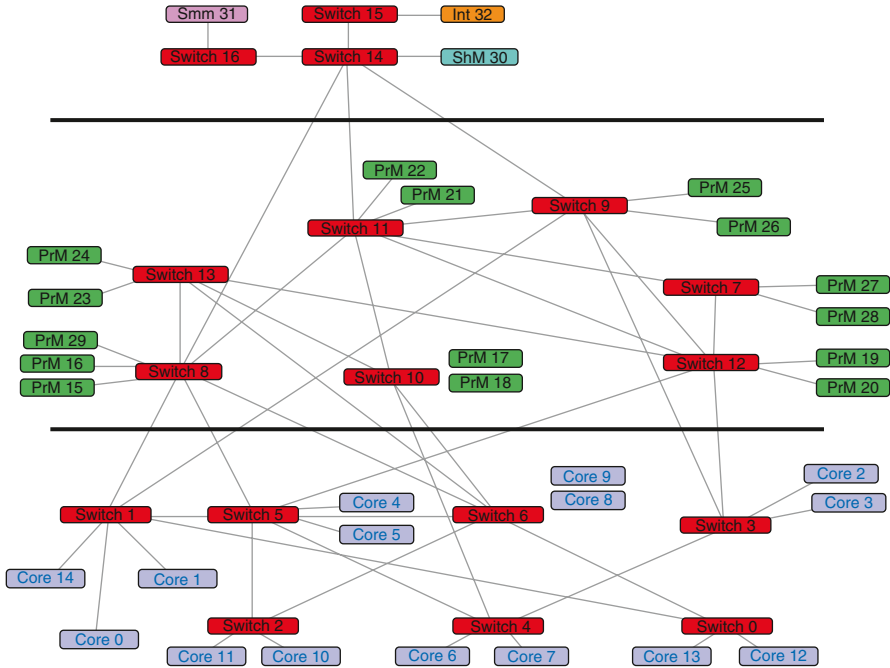


Fig. 9.5 Topology using 13 inter-layer links

While many works have addressed the architectural issues in the design of NoCs for 3D ICs, relatively fewer works have focused on the design aspects. In [17], the authors have presented methods for mapping cores on to 3D NoCs with thermal constraints. We have presented our methods to design NoCs for 3D ICs in [40, 41].

We also make a comparative study of NoC designs for the corresponding 2D implementation of the benchmarks. The objective is to evaluate the actual power and latency advantages when moving to a 3D implementation. For this study, we apply a flow developed by us earlier for NoC design for 2D systems [39]. Our results show that a 3D design can significantly reduce the interconnect power consumption (38% on average) when compared to the 2D designs. However, the latency savings is lower (13% on average), as the number of additional links that require pipelining in 2D were few.

We use TSVs to establish the vertical interconnections. In Fig. 9.7, an example of how a vertical link is established across two layers is presented. In our architecture, the TSV needs to be drilled only on the top layer, and the interconnect uses horizontal metal at the bottom layer. In our synthesis flow, we allocate area for a TSV macro at the top layer for the link during the floorplanning phase. The TSV macro is actually placed directly at the output port of the corresponding switch. For links that go across multiple silicon layers, we also place TSV macros in each intermediate layer. In Fig. 9.8, we show an example link that spans multiple layers.

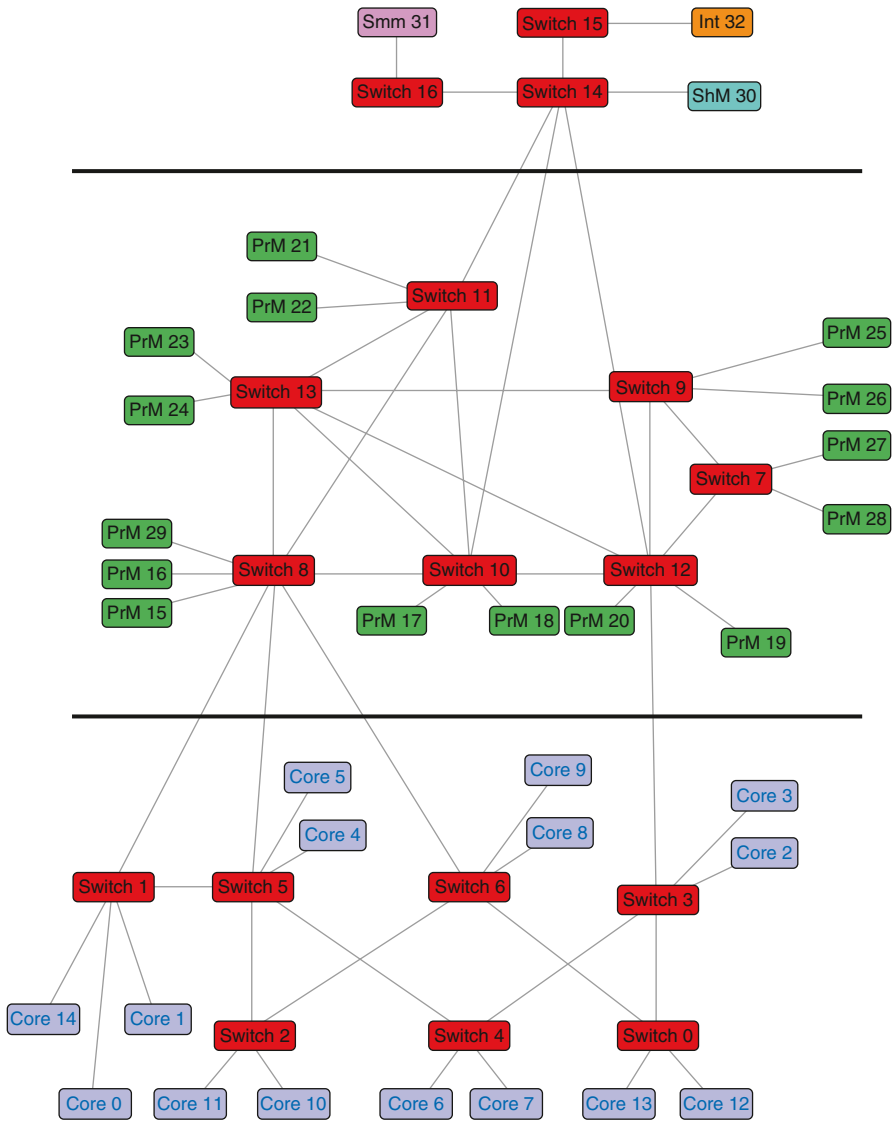


Fig. 9.6 Topology using eight inter-layer links

9.2 3D Architecture and Design Flow

If there is a core that is connected to a switch that is in another layer below the core layer, then the network interface that translates the core communication protocol to the network protocol will be the one that will contain the necessary TSV macros. If there are intermediate layers among the core's network interface and the switch it is connected to then TSV macros will be added in the intermediate layers just as in the

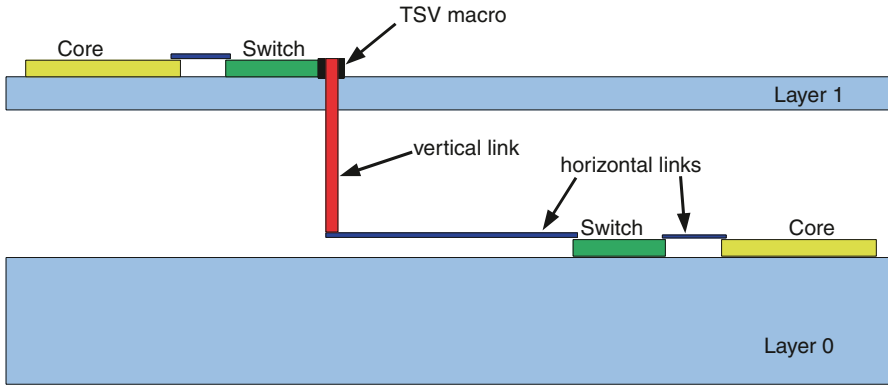


Fig. 9.7 Example vertical link

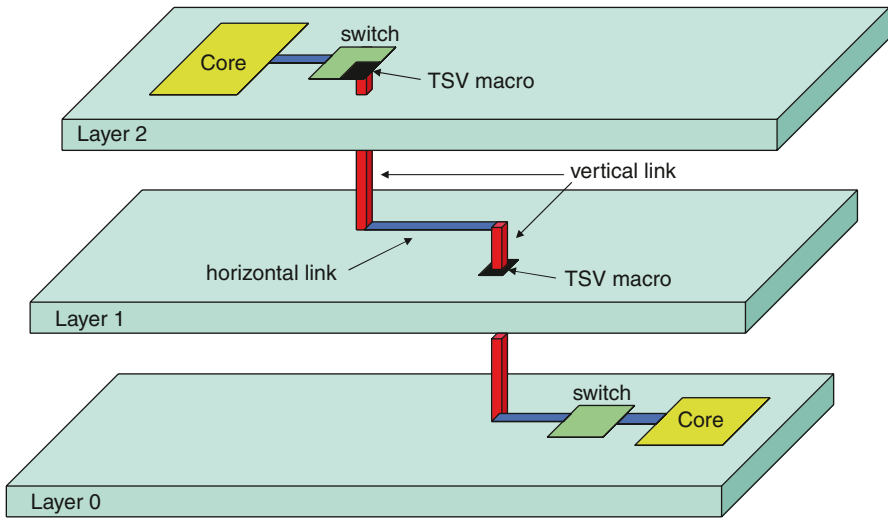


Fig. 9.8 Example vertical link

case of the switch to switch link from Fig. 9.8. Active silicon area is lost every time a TSV macro is placed as the area reserved by the macro will be used to construct the TSV.

9.3 Design Flow Assumptions

In the design approach we use several realistic assumptions.

- The computation architecture is designed separately from the communication architecture. Several works (such as [42]) have shown the need to separate compu-

tation from communication design to tame the complexity. We assume hardware/software partitioning of the design has been performed and tasks are statically assigned to the cores. For the communication architecture design, we assume that the hardware/software partition of application tasks onto the processor/hardware cores has been performed.

- The assignment of the cores to the different layers of the 3D are performed using existing methods/tools. There have been several works that address this issue and our work is complementary to them.
- The floorplan of the cores in each layer (without the NoC) has been performed by existing methods. We use the floorplan estimates as inputs to our design flow to better estimate wire delay and power consumption.
- Though the synthesis methods presented in this chapter are general and applicable to wide range of NoCs, we use a particular architecture ([43]) to validate the approach.

9.4 Design Approach

Our design flow for topology synthesis is presented in Fig. 9.9. The topology synthesis procedure produces a set of valid design points that meet the application performance and 3D technology constraints, with different power, delay and area values. From the Pareto curves, the designer can then choose the best design point.

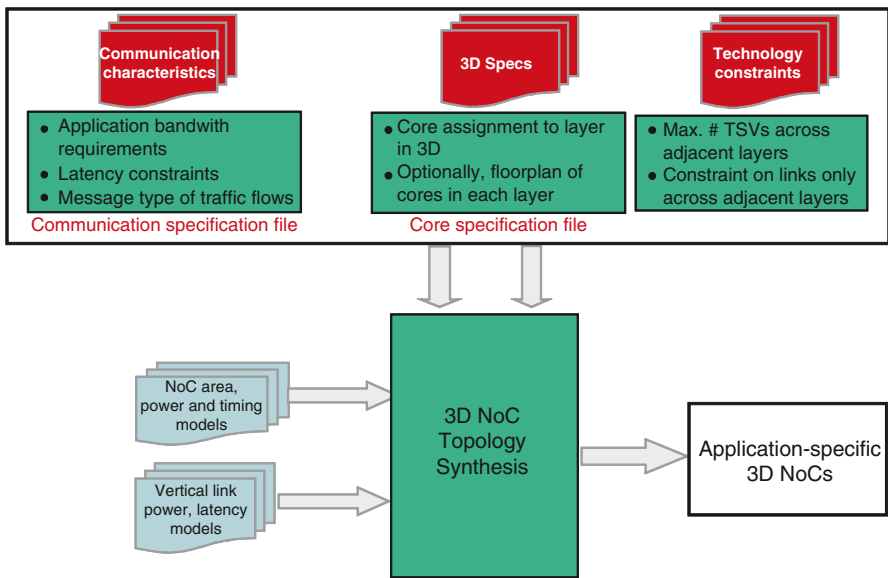


Fig. 9.9 Design flow

The placement and floorplan of the switches and TSVs on each layer is also produced as an output.

The different steps in the topology synthesis process are outlined as follows. In the outer most loop, the architectural parameters, such as the NoC frequency of operation, data width are varied and for each architectural point the topology synthesis process is repeated. Then, the number of switches in the design are varied. When fewer switches are used to connect the cores, the size of each switch is large and the inter-switch links are longer. However, the packets traverse a shorter path. On the other hand, when more switches are used, the size of each switch is smaller, but packets travel more hops. Depending on the application characteristics, the optimal power point in terms on the number of switches used varies. Hence, our synthesis tool sweeps a large design space, building topologies with different switch counts.

For a chosen architectural point and switch count, we establish connectivity across the cores and switches. Then, we also determine the layer assignment for each of the switches. If there is a tight constraint on the TSVs or when the design objective is to minimize the number of vertical connections, a core in a layer can be constrained to be connected to a switch in the same layer. In this way, core to switch links will not require vertical connections. On the other hand, this will require inter-layer traffic flows to traverse at least two switches, thereby increasing latency and power consumption. This choice is application-specific and should be chosen carefully.

To address this issue, we present a two-phase method for establishing core to switch connectivity. In the first phase, a core can be connected to a switch in any layer, while in the second phase, cores can be connected to only those switches in the same layer. The second phase is invoked when TSV constraints are not met in Phase 1 or when the objective is to minimize the number of vertical interconnections used. These phases are explained in detail in the next sections.

Several inputs are obtained for the topology synthesis process. The name, size and position of the different cores, assignment of cores to the 3D layers, the bandwidth and latency constraints of the different flows are obtained. A constraint on the number of TSVs that can be established is also taken as an input. In some 3D technologies, vertical interconnects can established only across adjacent layers. This is also taken as an input. We model the maximum TSV constraint by constraining the number of NoC links that can be established across adjacent layers. We denote this by *max_ill*. For a chosen link width, the value of *max_ill* can be computed easily from the TSV constraint. For the synthesis procedure, the power, area and timing models of the NoC switches and links are also taken as inputs. For the experimental validation, we use the library of network components from [43] and the models are obtained from layout level implementations of the library components. The design process is general and models for other NoC architectures can also be easily integrated with the design flow. The delay and power consumption values of the vertical interconnects are also taken as inputs. We use the models from [11] for the vertical interconnects.

9.5 Algorithm

We will now go on to describe the algorithm for synthesizing applications specific Noc topologies for 3D ICs. We will start by formally defining the inputs to the algorithm. The first input is the *core specification* which describes the number of cores, their position and the layer assignment. The core specification is defined as follows:

Definition 1 *For a design with n cores. The x and y co-ordinate positions of a core i are represented by xc_i and yc_i respectively, $\forall i \in 1 \dots n$. The assignment of core i to the 3D layer represented by $layer_i$.*

The second input is the *communication specification* which describes the communication characteristics of the application and it is represented by a graph [28, 30, 31]. The graph is defined as follows:

Definition 2 *The communication graph is a directed graph, $G(V, E)$ with each vertex $v_i \in V$ representing a core and the directed edge (v_i, v_j) representing the communication between the cores v_i and v_j . The bandwidth of traffic flow from cores v_i to v_j is represented by $bw_{i,j}$ and the latency constraint for the flow is represented by $lat_{i,j}$.*

Setting of several NoC architectural parameters can be explored, like the frequency at which the topology has to run and the data width of the links. The ranges in which the design parameters are varied are taken as inputs. The algorithm sweeps the parameters in steps, and designs the best topology points for each setting. For each architectural point, the algorithm performs the steps shown in Algorithm 1. The algorithm will create a list of all the switch counts for which topologies will be generated (steps 2–5). By default, the switches are varied from one to the maximum number of cores in the design or in each layer. However, the designer can also manually set the range of switch counts to be explored.

The objective function of topology synthesis is initially set to minimize power consumption. However, for each topology point, if the 3D technology constraints are not met, the objective function is slowly driven to minimize the number of vertical interconnections. For this purpose, we use the scaling parameter θ . To obtain designs with lower inter-layer links, θ is varied from θ_{min} to θ_{max} in steps of θ_{scale} , until the maximum number of inter-layer links constraints is met. After several experimental runs, we determined that varying θ from 1 to 15 in steps of 3 gives good results. In step 7, the algorithm tests if inter-layer links can cross multiple layers, and if not, then phase one is skipped and phase 2 is used directly. In step 8, the parameter θ used for setting the importance of the 3D constraints is set to the minimum value to try to optimize for power. The function to build topologies is called in step 10 on the initial list of switch counts to be explored. A detailed description of the *BuildTopologyGeneral* is given in the Sect. 9.5.1. If the *Unmet* set is not empty, then some topology points may not have met the technology constraints. Thus, θ is increased and the function is called again.

Phase 2 of the algorithm detailed in Sect. 9.5.2 is more restricted, as cores can only be connected to switches in the same layer of the 3D stack. Topologies built

using this restriction are usually consume more power, as more switches are required. Also the average hop count increases, as inter-layer flows traversing different layers have to go through at least two hops. The advantage of the method from phase 2 is that it can build topologies with a very tight constraint on the number of inter-layer links. In step 15, the algorithm tests if there are entries in the *Unmet* set for which topologies were not built in phase 1. This could be either because the constraints on the maximum number of inter-layer links were too tight or because the technology did not allow for inter-layer links to cross more than one layer and phase one was skipped completely. If *Unmet* is not empty then in step 16, the algorithm calls *BuildTopologyLayerByLayer* function which tries to build topologies using the restrictive approach.

Algorithm 1 Explore design points

```

1: {Initialize set Unmet with the switch counts for which topologies are explored}
2: Unmet =  $\phi$ 
3: for  $i = 1$  to  $|U|$  do
4:   Unmet = Unmet  $\cup$   $i$ 
5: end for
6: {If TSVs can cross multiple layers try general method}
7: if TSVmulti_layer = allowed then
8:    $\theta = \theta_{min}$ 
9:   while (Unmet  $\neq \phi$ ) & ( $\theta \leq \theta_{max}$ ) do
10:    BuildTopologyGeneral(V, E,  $\theta$ , Unmet)
11:     $\theta = \theta + \theta_{scale}$ 
12:   end while
13: end if
14: {If general method was not used or there are still design points for which
    solutions were not found use layer by layer method}
15: if (Unmet  $\neq \phi$ ) then
16:   BuildTopologyLayerByLayer(V, E, Unmet)
17: end if

```

9.5.1 Phase 1

Since different switch counts are explored and the number of switches rarely equals the number of cores, the first problem that arises is to decide how to connect the cores to switches. The assignment of cores to switches can have a big impact on the power consumption, but also on the number of inter-layers links required as switches from different layers can be assigned to the same switch. As multiple cores have to be assigned to the same switch, we partition the cores in as many blocks as there are switches. For this, we define the *Partitioning Graph* as follows:

Definition 3 *The partitioning graph is a directed graph, $PG(U, H, \alpha)$, that has same set of vertices and edges as the communication graph. The weight of the edge (u_i, u_j) , defined by $h_{i,j}$ is set to a combination of the bandwidth and the latency constraints of the traffic flow from core u_i to u_j : $h_{i,j} = \alpha \times bw_{i,j}/max_bw + (1 - \alpha) \times min_lat/lat_{i,j}$,*

where max_bw is the maximum bandwidth value over all flows, min_lat is the tightest latency constraint over all flows and α is a weight parameter.

Algorithm 2 *BuildTopologyGeneral(U, H, θ , Unmet)*

```

1: Build partitioning graph PG(U, H,  $\alpha$ )
2: if  $\theta > \theta_{min}$  then
3:   Build scaled partitioning graph, SPG(W, L,  $\theta$ )
4:   PG = SPG
5: end if
6: {For each switch count in Unmet build topology}
7: for Each  $i \in Unmet$  do
8:   Perform  $i$  min-cut partitions of PG.
9:   {Let the set  $partition_j$  be set of vertices in  $j^{th}$  partition,  $\forall j \in 1 \dots i$ }
10:  {Compute layer assignment for each switch;}
11:  for  $j = 1 \dots nsv$  do
12:    Assign to  $layer\_sw_j$  the value that averages the layer values of all
    cores  $\in partition_j$ 
13:  end for
14:  CheckConstraints(cost)
15:  Compute paths for inter-switch flows using cost
16:  if Paths were found then
17:    {If all paths were found remove current point from the set of unexplored
    solutions}
18:     $Unmet = Unmet \setminus i$ 
19:    Save solution
20:  end if
21: end for

```

The weights on the edges in the partitioning graph are calculated as a linear combination of the bandwidth required by the communication flow and the latency constraint. The parameter α can be used to make trade-offs between power and latency. The intuition is that when α is large, cores that have high bandwidth communication flows will be assigned to the same switch. This will minimize the switching activity in the NoC and therefore reduce the power consumption. On the other hand, when α is small, cores that have tight latency constraints will be assigned to the same switch minimizing the hop count. The parameter α is given as input or can be varied in a range as well to explore the trade-offs between power consumption and latency. However, the partitioning graph has no information on the layer assignment of the cores and cannot be used if the number of inter-layer links has to be reduced. For this purpose, we define the it Scaled partitioning Graph:

Definition 4 *A scaled partitioning graph with a scaling parameter θ , $SPG(W, L, \theta)$, is a directed graph that has the same set of vertices as PG. A directed edge $l_{i,j}$ exists between vertices i and j , if $\exists(u_i, u_j) \in P$ or $layer_i = layer_j$.*

In the scaled partitioning graph, the edges that connect vertices that correspond to cores that are in different layers are scaled down. This way cores that are on dif-

ferent layers will be assigned to different switches. This can lead to a reduction in the inter-layer links, because the links that connect switches can be reused by many flows while links that connect cores to switches can only be used by the communication flows of that core.

As the parameter θ scales, to drive the partitioner to cluster cores that are on the same layer, edges between the vertices that correspond to cores in the same layer are added. It is important that these edges have lower weight than the real edges. If too much weight is given to the new edges, then the clustering is no more communication based and it will lead to an increase in the power consumption. Equation 9.1 shows how the weights are calculated in the SPG. The weight from the new edges is calculated based on the maximum weight of the edge in the PG and it is denoted by \max_wt .

$$l_{i,j} = \begin{cases} h_{i,j} & \text{if } (u_i, u_j) \in PG \ \& \ layer_i = layer_j \\ \frac{h_{i,j}}{\theta \times |layer_i - layer_j|} & \text{if } (u_i, u_j) \in PG \ \& \ layer_i \neq layer_j \\ \frac{\theta \times \max_wt}{10 \times \theta_{max}} & \text{if } (u_i, u_j) \notin PG \ \& \ layer_i = layer_j \\ 0 & \text{otherwise} \end{cases} \quad (9.1)$$

From the definition, we can see that the newly added edges have at most one-tenth the maximum edge weight of any edge in PG, which was obtained experimentally after trying several values.

The steps of the *BuildTopologyGeneral* function are presented in Algorithm 2. In the first step, the partitioning graph is build. If θ is larger than the initial value (step 2), it means that feasible topologies could not be built for all switch counts using the core to switch assignment based on power and latency only. Therefore in step 3, the scaled partitioning graph is built from the partitioning graph using the current value of θ and replaces the partition graph in step 4. The design points from the *Unmet* set are explored in step 7. For each switch count that is explored, the cores are partitioned in as many blocks as the value of the switch count for the current point (step 8). Once the cores are connected to switches, the switch layer assignment can be computed. Switches are assigned to layers in the 3D stack based on the layer assignment of the core it connects to. A switch is placed at the average distance in the third dimension among all the cores it connects (steps 11–13). For the current core to switch assignment, the inter-switch flows have to be routed (steps 14, 15). The function *CheckConstraints(cost)* enforces the constraints imposed by the upper bound on inter-layer links. A more detailed description on how the constraints are enforced and how the routes are found is provided in Sect. 9.5.3. If paths for all the inter switch flows were found with the given constraints, then the topology for the design point is saved and the entry corresponding to the current switch count is removed from the *Unmet* set (steps 18 and 19).

9.5.2 Phase 2

As previously stated, phase 2 is more conservative in the sense that cores can only be connected to switches in the same layer. To ensure that the blocks that result from the partitioning do not contain cores that are assigned to different layers on the 3D stack, the partitioning is done layer by layer. To do a layer by layer partitioning, we define the *Local Partitioning Graph* as follows:

Definition 5 *A local partitioning graph, $LPG(Z, M, ly)$, is a directed graph, with the set of vertices represented by Z and edges by M . Each vertex represents a core in the layer ly . An edge connecting two vertices is similar to the edge connecting the corresponding cores in the communication graph. The weight of the edge (m_i, m_j) , defined by h_{ij} , is set to a combination of the bandwidth and the latency constraints of the traffic flow from core m_i to m_j : $h_{ij} = \alpha \times bw_{ij}/max_bw + (1 - \alpha) \times min_lat/lat_{ij}$, where max_bw is the maximum bandwidth value over all flows, min_lat is the tightest latency constraint over all flows and α is a weight parameter. For cores that do not communicate with any other core in the same layer, edges with low weight (close to 0) are added between the corresponding vertices to all other vertices in the layer. This will allow the partitioning process to still consider such isolated vertices.*

A local partitioning graph is built for each layer and the partitioning and therefore the core to switch assignment is done layer by layer. Another restriction is imposed on the switches, as they can be connected to other switches in the same layer, but only to switches that are in adjacent layers when it comes to connections in the third dimension.

Since there can be different number of cores on the different layers, it is important to be able to distribute the switches to the layers of the 3D stack unevenly. Therefore, the algorithm in phase 2 starts by determining the minimum number of switches in each layer necessary to connect the cores. The operating frequency determines the maximum size of a switch, as the critical path in a switch depends on the number of input/output ports. The maximum switch size is determined in step 1 based on switch frequency models given as inputs. In steps 2–5, the minimum number of switches in each layer is determined from the number of cores in the layer and the maximum supported size for the switches at the desired operating frequency. In step 5, the local partitioning graphs are built, one per layer. Then for each design point remaining in the *Unmet* set, the algorithm distributes the switches on the different layers (step 8). Then we calculate the actual number of switches to be used in each layer, starting from the minimum number of switches in each layer previously calculated (steps 12–16). We also make sure that the number of switches in each layer does not grow beyond the number of cores. For the calculated switch count on each layer, the local partitioning graphs are partitioned in as many blocks as the switch count (step 17). Once the cores are assigned to switches, the *CheckConstraints(cost)* is called to enforce the routing constraints and paths are found for the inter switch flows (steps 19, 20). If paths are found for all flows, the topology for the design point is saved.

Algorithm 3 *BuildTopologyLayerByLayer(V, E, Unmet)*

```

1: Obtain maximum switch size  $max\_sw\_size$  for current frequency
2: for each layer  $j \in 1 \dots lr$  do
3:   {Find the minimum number of switches needed in each layer}
4:    $ni_j = \lceil \text{number of cores in } layer_j / max\_sw\_size \rceil$ 
5: end for
6: Build  $LPG(Z, M, j)$  for each layer  $j$ .
7: for each  $i \in Unmet$  do
8:    $i = i / lr$ 
9:   {Calculate the number of switches in each layer starting from minimum
10:  number of switches in a layer}
11:  for each layer  $j \in 1 \dots lr$  do
12:    {The maximum number of switches in a layer cannot be larger than
13:    the number of cores in the layer}
14:    if  $ni_j + i \leq |LPG(Z, M, j)|$  then
15:       $np = ni_j + i$ 
16:    else
17:       $np = |LPG(Z, M, j)|$ 
18:    end if
19:    Obtain  $np$  min-cut partitions of  $LPG(Y, M, j)$ 
20:  end for
21:  CheckConstraints(cost)
22:  Compute paths for inter-switch flows using cost
23:  if Paths were found then
24:    Save solution
25:  end if
26: end for

```

9.5.3 Find Paths

When routing the inter switch flows, new physical links have to be opened between switches as in the beginning the switches are not connected among themselves. To establish the paths and to generate the physical connectivity for the inter switch flows, a similar procedure is used as in the 2D case [39]. The procedure finds minimum cost paths and the cost is based on the power increase generated by routing the new flow on that path. By using marginal power as the cost metric, the algorithm minimizes the overall power consumption. The full description of finding paths is beyond the scope of this work and we refer the reader's attention to [39]. The work also shows how to find deadlock free routes in the design, which can also be used in 3D. However, in 3D we must take care of the maximum number of inter-layer links constraint (max_ill) together with the constraint on the maximum switch size imposed by the operating frequency. Therefore, in this section we will focus on how these constraints can be enforced by modifying the cost on which the paths are calculated. The routine to check and enforce the constraints is presented in Algorithm 4. Before describing the algorithm we have to make the following definitions:

Definition 6 Let nsw be the total number of switches used across all the layers and let $layer_i$ be the layer in which switch i is present. Let $ill(i, j)$ be the number of vertical links established between layers i and j . Let the $switch_size_inp_i$ and $switch_size_out_i$ be the number of input and output ports of switch i . Let $cost_{i,j}$ be the cost of establishing a physical link between switches i and j .

Algorithm 4 *CheckConstraints(cost)*

```

1: Define  $soft\_max\_switch\_size$ ,  $soft\_max\_ill$ ,  $SOFT\_INF$ 
2: Define  $max\_switch\_size$ ,  $max\_ill$  and  $INF$ 
3: {For each pair of switches check if a link can be opened without violating the
   constraints}
4: for  $i = 1$  to  $nsw$  do
5:   for  $j = 1$  to  $nsw$  do
6:     {Check TSV constraint}
7:     if (TSV are not allowed to cross multiple layers and  $|layer_i - layer_j| \geq 2$ )
       or ( $ill$  count between  $layer_i$  and  $layer_j$  is  $\geq max\_ill$ ) then
8:        $cost_{i,j} = INF$ 
9:     else if  $ill$  count between  $layer_i$  and  $layer_j$  is  $\geq soft\_max\_ill$  then
10:       $cost_{ij} = SOFT\_INF$ 
11:     else if  $switch\_size\_inp_i + 1 \geq max\_switch\_size$  or
        $switch\_size\_out_j + 1 \geq max\_switch\_size$  then
12:       $cost_{i,j} = INF$ 
13:     else if  $switch\_size\_inp_i + 1 \geq soft\_max\_switch\_size$  or
        $switch\_size\_out_j + 1 \geq soft\_max\_switch\_size$  then
14:       $cost_{i,j} = SOFT\_INF$ 
15:     end if
16:   end for
17: end for

```

In the algorithm, we use two types of threshold. One type refers to hard thresholds that are given by the constraints and the other type is the soft thresholds which are set to be just a bit less than the hard constraints. While violating a hard threshold means that it is impossible to build a topology, soft thresholds can be violated. These allow the algorithm to reserve the possibility to open new links for special flows that otherwise cannot be routed due to other constraints (e.g. to enforce deadlock freedom). The algorithm tests if a link can be opened between every pair (i, j) of switches (steps 3, 4). First, the constraint on the number of vertical links are checked. In the case of phase 2, when inter-layer links cannot cross multiple layers and the distance in the third dimension between switch i and switch j is larger than 1, then the cost for that pair is set to INF . Also if the number of inter-layer links between the layer containing switch i and switch j reached the max_ill value, then the cost is also set to INF (steps 7, 8). By setting the cost to INF , we make sure that when finding paths, we will not open a new link between switch i and j . If only the hard threshold is used, then the algorithm would be able open links until reaching the limit and abruptly hit

an infeasible point. In a similar manner to the hard constraints, the soft constraints are enforced by setting the cost to *SOFT_INF* when the number of inter-layer links is already close to the hard constraint (steps 9, 10). The *SOFT_INF* value is chosen to be several orders of magnitude larger than the normal cost based on power. The constraints to limit the size of the switches are very similar to the constraints on the maximum number of inter-layer links and are enforced in steps 11–15.

When paths are computed, if it is not feasible to meet the *max_switch_size* constraints, we introduce new switches in the topology that are used to connect the other switches together. These indirect switches help in reducing the number of ports needed in the direct switches. Due to space limitations, in this chapter, we do not explain the details of how the indirect switches are established.

If we look at Algorithm 1, we can see that many design points are explored, especially when the constraint on the maximum number of inter-layer links is tight. Several methods can be employed to stop the exploration of design points when it becomes clear that a feasible topology cannot be built. To prune the search space, we propose three strategies. First, as the number of input/output ports of a switch increases, the maximum frequency of operation that can be supported by it reduces, as the combinational path inside the crossbar and arbiter increases with size. For a required operating frequency of the NoC, we first determine the maximum size of the switch (denoted by *max_sw_size*) that can support that frequency and determine the minimum number of switches needed. Therefore, design points where the switch count is less than that can be skipped. Second, for phase 2 we initialize the number of switches layer by layer as above. Thus, the starting design point can have different number of switches in each layer. The third strategy is applied after partitioning. The number of inter-layer links used to connect the cores to the switches is evaluated, before finding the paths. If the topology requires more inter-layer links than the threshold, we directly ignore the design point.

9.5.4 Switch Position Computation

In modern technology nodes, a considerable amount of power is used to drive the wires. To be able to evaluate more accurately the power consumption of a topology point, we have to estimate the power used to drive the links. In order to evaluate the lengths of the links, we have to find positions for the switches and to place them in the floorplan. While the positions of the cores is given as input, the switches are added by the algorithm and their positions has to be calculated. Since a switch is connected to several cores, one way to calculate the switch position is to minimize the distance between the switch and the cores it is connected to. This can easily be done by averaging the coordinates of the cores the switches is connected to. This is a simple strategy and can provide good results and can be further improved by weighing the distance between the switch and cores with the bandwidth that the core generates, so that links that carry more bandwidth would be shorter. However, this strategy does not take into consideration that the switch can be connected to

other switches as well and minimizing the distance between switches is desirable. To achieve this, a strategy that uses a linear program formulation that minimizes the distance between the cores and switches at the same time is presented in [40]. If an inter-layer link crosses more than one layer then macros have to be placed on the floorplan to reserve space to create the TSVs. However, finding the position for TSV macros is much easier because the TSV macro is connected between only two components (core to switch or switch to switch). Therefore the TSV macro can be placed anywhere in the rectangle defined by the two component as it would not increase the wire length (Manhattan distance is considered).

Placing the switches and TSV macros at the computed position may result in overlap with the existing cores. For most real designs, moving the cores from their relative positions is not desirable as there are many constraint to be satisfied.

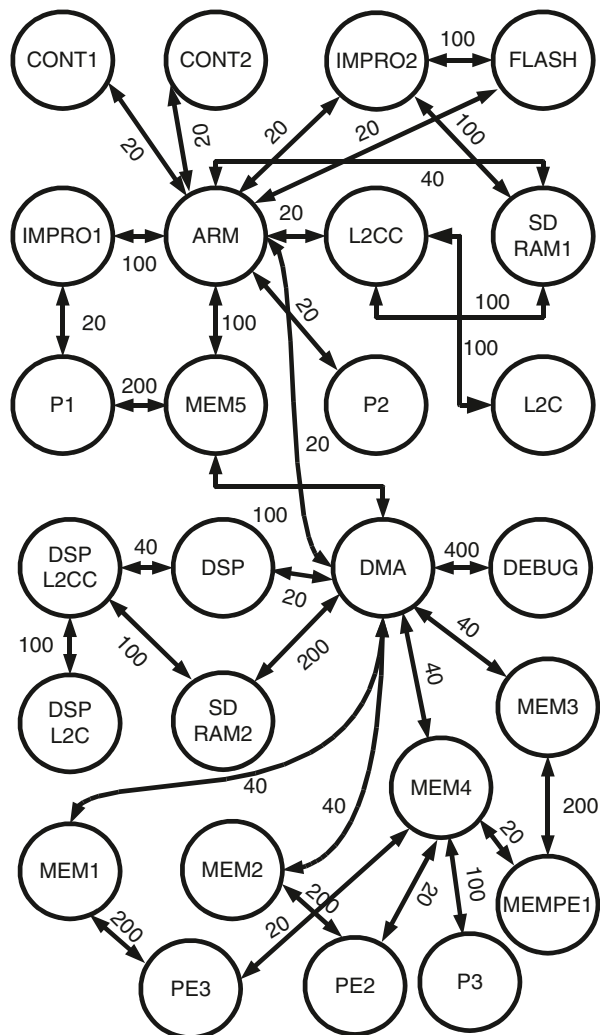


Fig. 9.10 D26 media communication graph

A standard floorplanner can be used, but it can produce poor results if it is not allowed to swap the cores. A custom routine designed to insert the NoC component in the existing floorplan can give better results removing the overlap. The routine considers one switch or TSV macro at a time. It tries to find a free space near its ideal location to place it. If no space is available, we displace the already

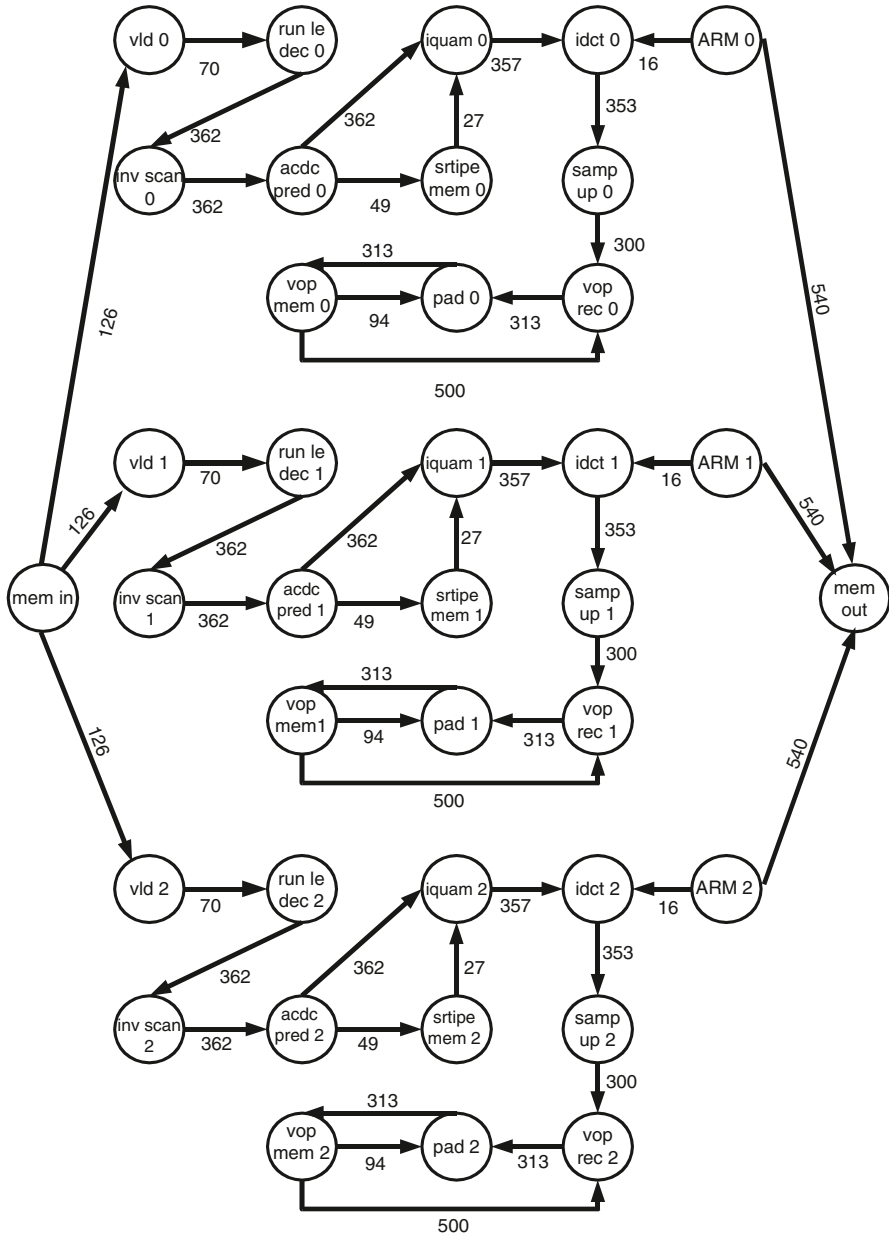


Fig. 9.11 *D38_tvpd* communication graph

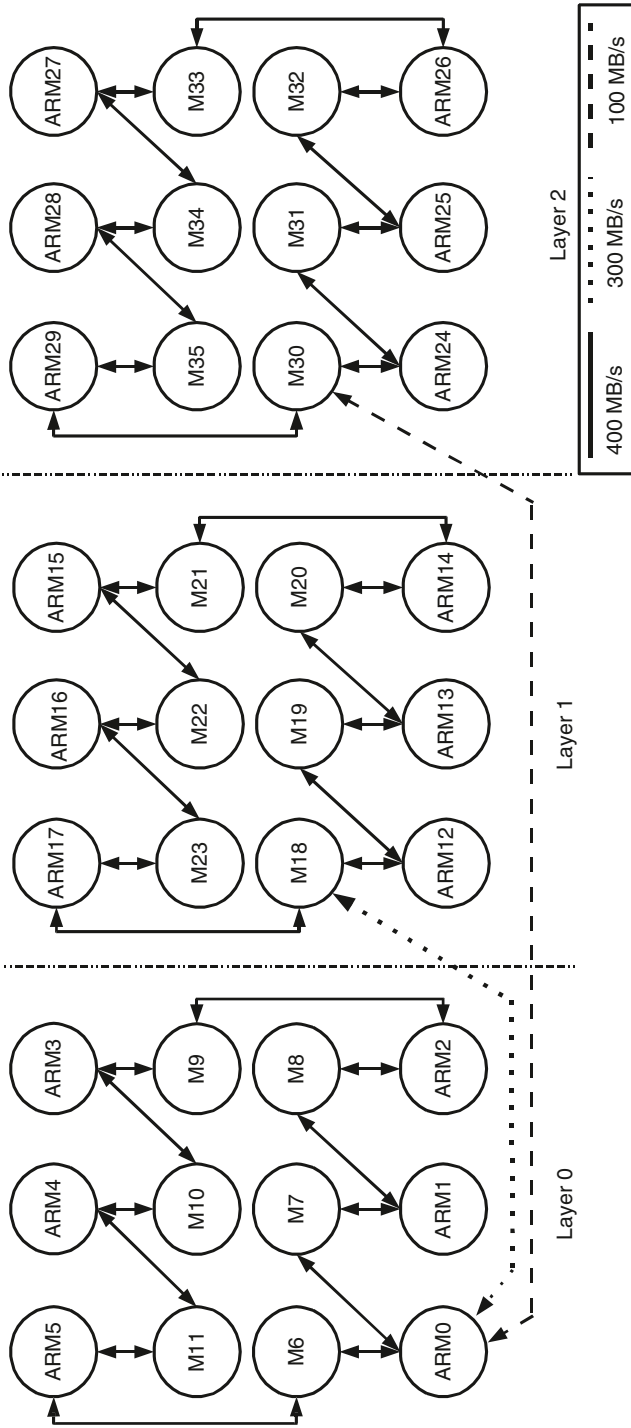


Fig. 9.12 D36_4 communication graph

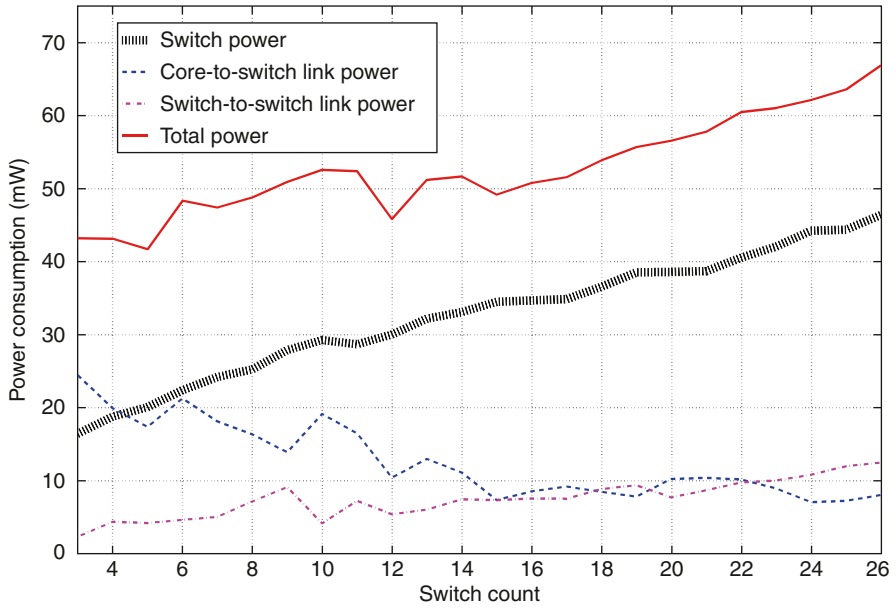


Fig. 9.13 Power consumption in 2D

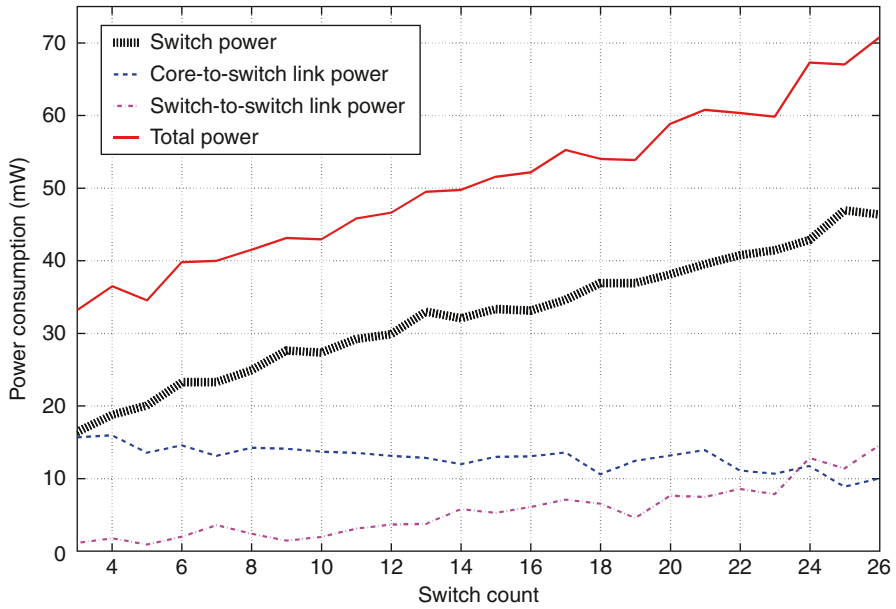


Fig. 9.14 Power consumption in 3D

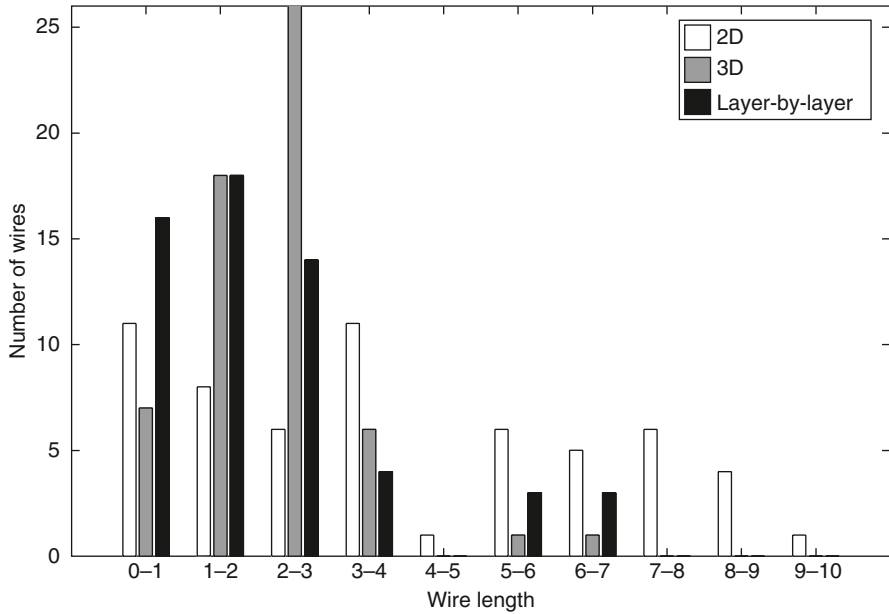


Fig. 9.15 Wire length distributions

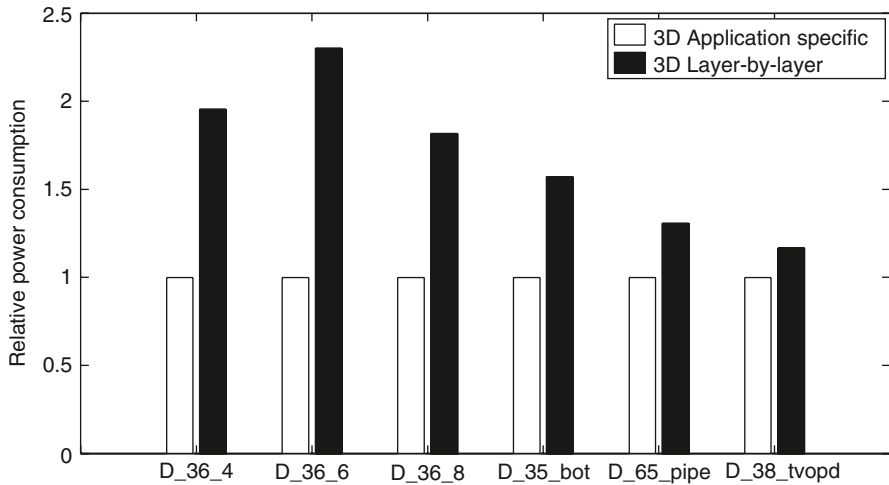


Fig. 9.16 Comparison with layer-by-layer

placed blocks from their positions in the x or y direction by the size of the component, creating space. Moving a block to create space for the new component can cause overlap with other already placed blocks. We iteratively move the necessary blocks in the same direction as the first block, until we remove all overlaps. As more components are placed, they can re-use the gap created by the earlier components (See Figs. 9.10 to 9.18).

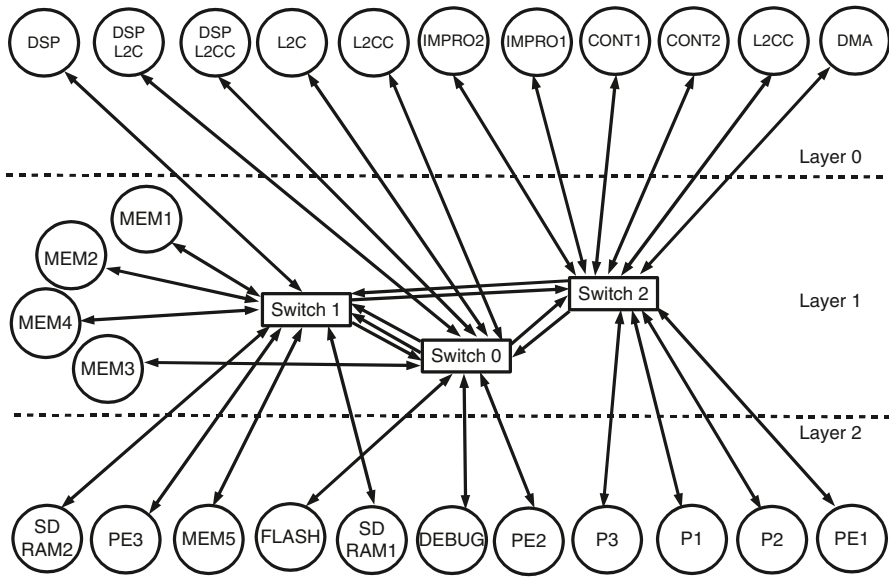


Fig. 9.17 Most power-efficient topology

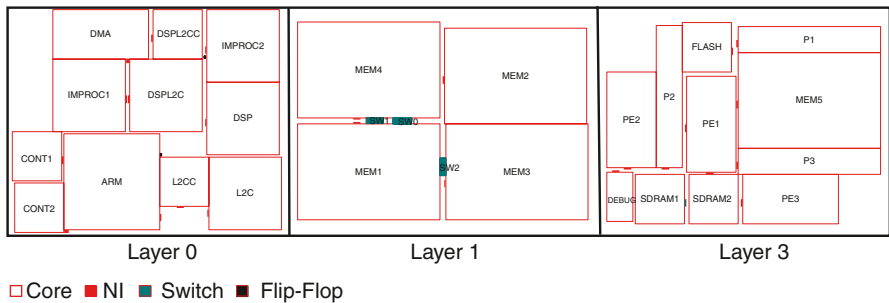


Fig. 9.18 Resulting 3D floorplan with switches

9.6 Experiments and Case Studies

To show how the algorithm performs, we will show different results on a realistic multimedia and wireless communication benchmark. We will also make a comparison on topologies built with the two phases of the described algorithm to show the advantages and disadvantages of each phase. We will also make a comparison between NoCs designed for different applications for 2D-ICs and 3D-ICs. Experiments showing the advantage of custom topologies over regular ones are also presented. In order to better estimate power, the NoC component library from [43] is used. The power and latency values of the switches and links of the library are

determined from post-layout simulations, based on 65 nm low power libraries. The vertical links are shown to have an order of magnitude lower resistance and capacitance than a horizontal link of the same dimension [11]. This translates to a traversal delay of less than 10% of clock cycle for 1 GHz operation and negligible power consumption on the vertical links.

9.6.1 Multimedia SoC Case Study

For the case study, a realistic multimedia and wireless communication SoC was chosen as benchmark. The communication graph for the benchmark (denoted as *D_26_media*) is shown in Fig. 5.4. From the figure, it can be observed that there are 26 cores in the SoC. A part of the SoC which is constructed around an ARM processor is used for multimedia applications and it is aided by hardware video accelerators and controllers to access external memory. The other part of the SoC built around a DSP is used for wireless communication. The communication between the two parts is facilitated by several large on-chip memories and a DMA core. A multitude of peripherals are also present for off-chip communication.

To compare between NoC designed for 2D-ICs and 3D-ICs, we performed 2D floorplans of the cores as well as 3D floorplan of the cores distributed to three layers using existing tools [44]. The assignment of cores to the layers of the 3D stack was performed manually, but there are solutions presented for 3D floorplanning that can give also the assignment of cores to layers. The tool from [39] was used to generate the application specific topologies for the 2D case. To data width of the links was set to 32 bits to correspond to the data width of the cores and the frequency was set to 400 MHz (this being the lowest frequency at which topologies can be designed to support the required bandwidth for the chosen data width). The *max_ill* constraint was set at 25. The impact of the constraint on power is analyzed later on. The power consumption on the different components of the NoCs, as well as the total power consumption is presented in Fig. 9.9 for the 2D-IC and in Fig. 9.10 for 3D. The plots represent the power consumption for topologies generated for different switch counts.

Both plots start with topologies containing three switches. Because there are 26 cores in the design, topologies with less than three switches could not be built, as they would require the use of switches too large to support the operating frequency. These design points are pruned from exploration to reduce the run time, as explained in Sect. 9.5. The power consumption on individual components: switches, switch to switch links and core to switch links are presented in the figures. Several trends can be observed. The switch power grows as the number of switches grow. The core to switch link power goes down with more switches as the switches are placed closer to cores and the wire length decreases. The switch to switch link power consumption grows with the number of switches as the number of such links increases. However, the switch to switch link power does not increase as fast as the switch power with the switch count. The trends are similar for both 2D and 3D cases, but the absolute values for the link power in the 3D case is less than the ones

for 2D, as long and power hungry links from the 2D layout are replaced by short and efficient vertical links. For this particular benchmark, a power saving of 24% is achieved in 3D over 2D due to shorter wires. To give a better understanding, we show the wire-length distribution of the links in 2D and 3D cases in Fig. 9.11. From the figure, as expected, the 2D design has many long wires. In Fig. 9.12 the topology designed using phase 1 for the best power point is shown and in Fig. 9.13 the floorplan of the cores and network components in 3D for the corresponding topology is presented.

For a more complete comparison between topologies for 2D-ICs and 3D-ICs, we designed topologies using different SoC benchmarks. We consider three distributed benchmarks with 36 cores (18 processors and 18 memories): *D_36_4* (communication graph in Fig. 9.14), *D_36_6* and *D_36_8*, where each processor has 4, 6 and 8 traffic flows going to the memories. The total bandwidth is the same in the three benchmarks. We consider a benchmark, *D_35_bot* that models bottleneck communication, with 16 processors, 16 private memories (one processor is connected to one private memory) and 3 shared memories to which all the processors communicate. We also consider two benchmarks where all the cores communicate in a pipeline fashion: 65 core (*D_65_pipe*) and 38 core designs (*D_tvopd*) (communication graph in Fig. 5.4). In the last two benchmarks, each core communicates only to one or few other cores.

We selected the best power points for both the 2D case and the 3D case and we report the power and zero load latency in Table 9.1. As most of the power difference comes from the reduced wire length in 3D, the power savings differs from benchmark to benchmark. For the benchmarks with spread traffic and many communication flows the power savings are considerable, as they benefit from the reduction of the long wires of the 2D design. In the bottleneck benchmark, there are many long wires that go to the shared memory in the 2D case. Even though the traffic to shared memories is small, we can still see a reasonable power saving when moving to 3D. For the pipeline benchmarks, most of the links are between neighboring cores, so links are short even in 2D. So, going for a 3D design does not lead to large savings. The average power reduction is 38% and the average zero load latency reduction is 13% for the different benchmarks when comparing 3D to a 2D implementation.

Table 9.1 2D vs 3D NoC comparison

Benchmark	Power (mW)						Latency (cyc)	
	Link power		Switch power		Total power		2D	3D
	2D	3D	2D	3D	2D	3D		
<i>D_36_4</i>	150	41.5	65	70.5	215	112	3.28	3.14
<i>D_36_6</i>	154.5	43.5	76.5	82	230	125.5	3.57	3.5
<i>D_36_8</i>	215	55.5	105	104.5	320	160	4.37	3.65
<i>D_35_bot</i>	68	36.2	48	43.3	116	79.5	6.04	4.2
<i>D_65_pipe</i>	106	104	63	58	169	162	2.53	2.57
<i>D_38_tvopd</i>	52.5	22.67	37	38.11	89.5	60.78	4	3.6

In Fig. 9.13, we show the power consumption of the topologies synthesized using Phase 2 of the algorithm, with respect to topologies synthesized using Phase 1 for the different benchmarks. Since in Phase 2 cores in a layer are connected to switches in the same layer, the inter-layer traffic needs to traverse more switches to reach the destination. This leads to an increase in power consumption and latency. As seen from Fig. 9.13, Phase 1 can generate topologies that lead to a 40% reduction in NoC power consumption, when compared to the Phase 2. However Phase 2 can generate topologies with a much tighter inter-layer link constraint.

9.6.2 Impact of Inter-layer Link Constraint and Comparisons with Mesh

Limiting the number of inter-layer links has a great impact on power consumption and average latency. Reducing the number of TSVs is desirable for improving the yield of a 3D design. However, a very tight constraint on the number of inter-layer links can lead to a significant increase in power consumption. To see the impact of the constraint, we varied the value of max_ill constraint and performed topology synthesis for each value, for the D_36_4 benchmark. The power and latency values for the different max_ill design points are shown in Figs. 9.19 and 9.20. When there is a tight constraint on the inter-layer links, the cores are connected to switches in the same layer, so that only switch-to-switch links need to go across layers. This results in the use of more switches in each layer, increasing switch power consumption and average latency. Please note that our synthesis algorithm also allows the designers to perform such power, latency trade-offs for yield, early in the design cycle.

Custom topologies that match the application characteristics can result in large power-performance improvement when compared to the standard topologies, such as mesh and torus [39]. A detailed comparison between a custom topol-

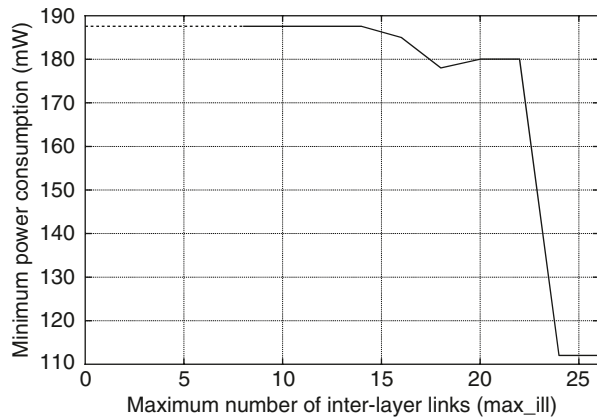
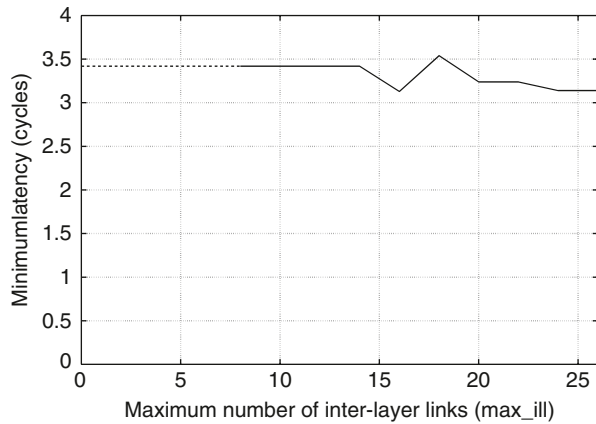


Fig. 9.19 Impact of max_ill on power

Fig. 9.20 Impact of *max_ill* on latency



ogy and several standard topologies for different benchmarks for the 2D case has been presented in [39]. For completeness, we compared the application specific topologies generated by our algorithm with an optimized 3D mesh topology (*3D Opt-mesh*), where core placement is optimized such that cores that communicate are connected to nearby switches. The power consumption value for the topologies for different benchmarks is presented in Fig. 9.21. The custom topologies result in large power reduction (average of 51%) when compared to the 3D mesh topology.

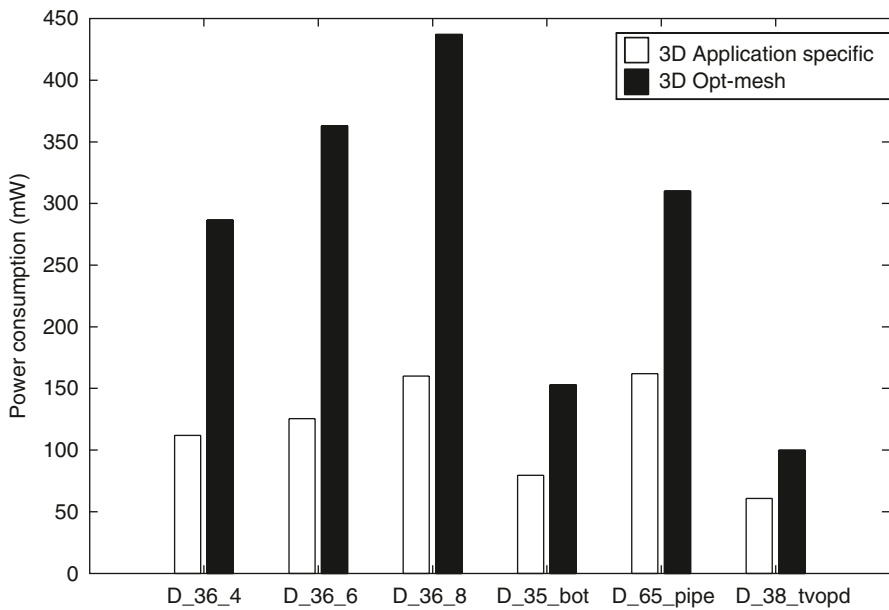


Fig. 9.21 Comparisons with mesh

9.7 Conclusions

Networks On Chips (NoCs) are a necessity for achieving 3D integration. One of the major design issues when using NoCs for 3D is the synthesis of the NoC topology and architecture. In this chapter, we presented synthesis methods for designing power-efficient NoC topologies. The presented methods not only address classic 2D issues, such as meeting application performance requirements, minimizing power consumption, but also the 3D technology constraints. We showed two flavors of the general algorithm, one for achieving low power solution and the other to achieve tight control on the number of vertical connections established. We also presented comparisons with 2D designs to validate the benefits of 3D integration for interconnect delay and power consumption.

Acknowledgment We would like to acknowledge the financial contribution of CTI under project 10046.2 PFNM-NM and the ARTIST-DESIGN Network of Excellence.

References

1. K. Banerjee et al., “3-D ICs: A Novel Chip Design for Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration”, Proc. of the IEEE, vol. 89, no. 5, p. 602, 2001.
2. L. Benini and G. De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computers, vol. 35, no. 1, pp. 70–78, Jan. 2002.
3. E. Beyne, “The Rise of the 3rd Dimension for System Integration”, International Interconnect Technology Conference, pp. 1–5, 2006.
4. B. Goplen and S. Sapatnekar, “Thermal Via Placement in 3D ICs”, Proc. Intl. Symposium on Physical Design, p. 167, 2005.
5. J. Cong et al., “A Thermal-Driven Floorplanning Algorithm for 3D ICs”, ICCAD, Nov. 2004.
6. W.-L. Hung et al., “Interconnect and Thermal-Aware Floorplanning for 3D Microprocessors”, Proc. ISQED, March 2006.
7. S. K. Lim, “Physical Design for 3D System on Package”, IEEE Design & Test of Computers, vol. 22, no. 6, pp. 532–539, 2005.
8. P. Zhou et al., “3D-STAF: Scalable Temperature and Leakage Aware Floorplanning for Three-Dimensional Integrated Circuits”, ICCAD, Nov. 2007.
9. R. Weerasekara et al., “Extending Systems-on-Chip to the Third Dimension: Performance, Cost and Technological Tradeoffs”, ICCAD, 2007.
10. G. H. Loh, Y. Xie, and B. Black. “Processor Design in 3D Die-Stacking Technologies”, IEEE Micro Magazine, vol. 27, no. 3, pp. 31–48, May--June 2007.
11. I. Loi, F. Angiolini, and L. Benini, “Supporting Vertical Links for 3D Networks on Chip: Toward an Automated Design and Analysis Flow”, Proc. Nanonets, 2007.
12. C. Guedj et al., “Evidence for 3D/2D Transition in Advanced Interconnects”, Proc. IRPS, 2006.
13. <http://www.zurich.ibm.com/st/cooling/interfaces.html>
14. IMEC, http://www2.imec.be/imec_com/3d-integration.php
15. <http://www.tezzaron.com>
16. N. Miyakawa, “A 3D Prototyping Chip Based on a Wafer-level Stacking Technology”, ASP-DAC, 2009.
17. C. Addo-Quaye, “Thermal-Aware Mapping and Placement for 3-D NoC Designs”, Proc. SOCC, 2005.

18. P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet Switched Interconnections", Proc. DATE, 2000.
19. G. De Micheli and L. Benini, "Networks on Chips: Technology and Tools", Morgan Kaufmann, San Francisco, CA, First Edition, July 2006.
20. J. Kim et al., "A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3d Architectures", ISCA, 2007.
21. D. Park et al., "MIRA: A Multi-Layered On-Chip Interconnect Router Architecture", ISCA, 2008.
22. F. Li et al., "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory", ISCA, 2006.
23. V. F. Pavlidis and E. G. Friedman, "3-D Topologies for Networks-on-Chip", IEEE TVLSI, 2007.
24. B. Feero and P. P. Pande, "Performance Evaluation for Three-Dimensional Networks-on-Chip", Proc. ISVLSI, 2007.
25. J. Hu et al., "System-Level Point-to-Point Communication Synthesis Using Floorplanning Information", Proc. ASPDAC, 2002.
26. S. Pasricha et al., "Floorplan-Aware Automated Synthesis of Bus-Based Communication Architectures", Proc. DAC, 2005.
27. S. Murali and G. De Micheli, "An Application-Specific Design Methodology for STbus Crossbar Generation", Proc. DATE, 2005.
28. S. Murali and G. De Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs", Proc. DAC, 2004.
29. S. Murali and G. De Micheli, "Bandwidth Constrained Mapping of Cores on to NoC Architectures", Proc. DATE, 2004.
30. J. Hu and R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures", Proc. DATE, 2003.
31. S. Murali et al., "Mapping and Physical Planning of Networks on Chip Architectures with Quality-of-Service Guarantees", Proc. ASPDAC, 2005.
32. A. Pinto et al., "Efficient Synthesis of Networks on Chip", ICCD 2003, Oct. 2003.
33. W. H. Ho and T. M. Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns", HPCA, 2003.
34. T. Ahonen et al., "Topology Optimization for Application Specific Networks on Chip", Proc. SLIP, 2004.
35. K. Srinivasan et al., "An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks", ICCAD, 2005.
36. J. Xu et al., "A Design Methodology for Application-Specific Networks-on-Chip", ACM TECS, 2006.
37. A. Hansson et al., "A Unified Approach to Mapping and Routing on a Combined Guaranteed Service and Best-Effort Network-on-Chip Architectures", Technical Report No: 2005/00340, Philips Research, Apr. 2005.
38. X. Zhu and S. Malik, "A Hierarchical Modeling Framework for On-Chip Communication Architectures", ICCD, 2002.
39. S. Murali et al., "Designing Application-Specific Networks on Chips with Floorplan Information", ICCAD, 2006.
40. S. Murali et al., "Synthesis of Networks on Chips for 3D Systems on Chips", ASPDAC, 2009.
41. C. Seiculescu, S. Murali, L. Benini, and G. De Micheli, "SunFloor 3D: A Tool for Networks on Chip Topology Synthesis for 3D Systems on Chip", Proc. DATE, 2009.
42. K. Keutzer et al., "System-Level Design: Orthogonalization of Concerns and Platform-Based Design", IEEE TCAD, 2000.
43. S. Stergiou et al., "xpipesLite: a Synthesis Oriented Design Library for Networks on Chips", Proc. DATE, 2005.
44. S. N. Adya and I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design", IEEE TVLSI, 2003.