Designing Application-Specific Networks on Chips with Floorplan Information

Srinivasan Murali[⋆], Paolo Meloni[§], Federico Angiolini[‡], David Atienza[⋆+], Salvatore Carta[¶], Luca Benini[‡], Giovanni De Micheli[⋆], Luigi Raffo[§]
[⋆] LSI, EPFL, Switzerland,{srinivasan.murali, david.atienza, giovanni.demicheli}@epfl.ch
[§]DIEE, University of Cagliari, Italy, {paolo.meloni, luigi}@diee.unica.it
[‡]DEIS, Univerity of Bologna, Italy, {fangiolini, lbenini}@deis.unibo.it
[+]DACYA, Complutense University of Madrid (UCM), Spain
[¶]DMI, University of Cagliari, Italy, salvatore@unica.it

ABSTRACT

With increasing communication demands of processor and memory cores in *Systems on Chips (SoCs)*, scalable *Networks on Chips (NoCs)* are needed to interconnect the cores. For the use of NoCs to be feasible in today's industrial designs, a custom-tailored, application-specific NoC that satisfies the design objectives and constraints of the targeted application domain is required. In this work, we present a design methodology that automates the synthesis of such application-specific NoC architectures. We present a floorplan aware design method that considers the wiring complexity of the NoC during the topology synthesis process. This leads to detecting timing violations on the NoC links early in the design cycle and to have accurate power estimations of the interconnect. We incorporate mechanisms to prevent deadlocks during routing, which is critical for proper operation of NoCs. We integrate the NoC synthesis method with an existing design flow, automating NoC synthesis, generation, simulation and physical design processes. We also present ways to ensure design convergence across the levels. Experiments on several SoC benchmarks are presented, which show that the synthesized topologies provide a large reduction in network power consumption ($2.78\times$ on average) and improvement in performance ($1.59\times$ on average) over the best mesh and mesh-based custom topologies. An actual layout of a multimedia SoC with the NoC designed using our methodology is presented, which shows that the designed NoC supports the required frequency of operation (close to 900 MHz) without any timing violations. We could design the NoC from input specifications to layout in 4 hours, a process that usually takes several weeks.

Keywords

Systems on Chips (SoCs), Networks on Chips (NoCs), deadlock-free routing, topology, application-specific, floorplan

1. INTRODUCTION

With technology scaling, the number of processor, memory and hardware cores on a chip is increasing. This has resulted in increased computation and communication complexity of the design, and scalable approaches are needed to design the system. *Networks on Chips (NoCs)* have emerged as the paradigm for designing scalable communication architecture for *Systems on Chips (SoCs)* [4], [5]. In NoCs, instead of the traditional non-scalable buses, on-chip micro-networks are used to interconnect the various cores. NoCs have better modularity and design predictability when compared to bus based systems.

Some of the most important phases in designing the NoC are the synthesis of the topology or structure of the network and setting of various design parameters (such as frequency of operation or link-width). The standard topologies (mesh, torus, etc.) that have been used in macro-networks result in poor performance and have large power and area overhead when used for SoCs. Such topologies are required for on-chip systems where the traffic characteristics of the system cannot be predicted statically, as in chip-multiprocessors. However, for most SoCs the system is designed with static (or semi-static) mapping of tasks to processors and hardware cores and hence the communication traffic characteristics of the SoC can be obtained statically. This is true from SoC designs that are small to state-of-the art SoCs, such as, the Philips Nexperia platform [1], ST Nomadik [2], TI OMAP [3], etc.

Another motivation for the use of NoCs is the fact that the interconnect structure and wiring complexity can be well controlled. When the interconnect is structured, the number of timing violations that occur during the physical design (floorplanning and wire routing) phase is minimum. Such design predictability is critical for today's SoCs for achieving timing closure. It leads to faster design cycle, reduction in the number of design re-spins and faster time-to-market. As the wire delay as a fraction of gate delay is increasing with each technological generation, having shorter wires is even more important for future SoCs. Early works on NoC topology design assumed that using regular topologies (such as mesh) would lead to regular and predictable layouts [16]. While this may be true for designs with homogeneous processing cores and memories, this is not true for most SoCs as they are typically composed of heterogeneous cores. This is due to the fact that the core sizes of the SoC are highly non-uniform and the floorplan of the design does not match the regular, tile-based floorplan of standard topologies [7]. An application-specific NoC with structured wiring, which satisfies the design objectives and constraints is important to have feasible NoC designs.

As a motivating example, the network power consumption (switch and link power consumption), hop-count, wire-length and design area of two different NoC topologies for a video processor SoC with 42 cores is presented in Table 1. The first topology is a

Table 1: Topology Comparisons

Parameter	Mesh	Application-specific
Power (mW)	301.78	79.64
Hop-Count	2.58	1.67
Total wire-length (mm)	185.72	145.37
Design Area (mm^2)	51.0	47.68

mesh, while the second is a custom topology generated using the methodology presented in this paper. The wire-lengths and design area are obtained from floorplanning of the NoC designs. The detailed explanation of the topologies and the floorplanning process is described later in this paper (Sections 5, 6 B). The custom topology leads to a $3.8\times$ reduction in network power consumption, a $1.55\times$ reduction in average hop-count and a $1.28\times$ reduction in total length of wires when compared to the mesh.

In this work, we present a methodology to design the best topology that is tailor-made for a specific application and satisfies the communication constraints of the design. Our topology design process supports two objective functions: minimizing network power consumption and hop-count for data transfer. The designer can optimize for one of the two objectives or a linear combination of both. The topology design process supports constraints on several parameters such as the hop-count (when the objective is power minimization), network power consumption (when the objective is hop-count minimization), design area and total wire-length. The topology synthesis process uses a floorplanner to estimate the design area and wire-lengths. The wire-length estimates from the floorplan are used to evaluate whether the designed NoC satisfies the target frequency of operation and to compute the power consumption of the wires.

As deadlock-free routing is critical for proper operation of custom topologies, we integrate methods to find deadlock free paths during the topology design process. We have built accurate analytical models for power consumption and area of the network components. The power consumption values are obtained from layouts with back-annotated resistance, capacitance information and from the switching activity of the components. We also automatically tune several NoC architectural parameters (such as the NoC operating frequency, link-width) in the design process. The methodology can be streamlined with existing tool flows for instantiation, synthesis, FPGA emulation and layout of the NoC design. We present ways to close the design gap across the various levels of the flow: from topology design to floorplanning to simulation of the design. The methodology also supports manual intervention, if needed, at several levels (like manually setting up frequency, link-width).

To the best of our knowledge, this is the first work that presents a streamlined design methodology for NoC topology synthesis that is completely integrated with the state-of-the commercial tools for back-end physical design. Unlike all earlier works (please refer to Section 2), we present a floorplan aware topology design method for NoCs that leads to detecting timing violations on the NoC links early in the design cycle, with the resulting designs fully verified for timing correctness using standard place&route tools. This is also the first work on custom NoC topology synthesis that guarantees a complete deadlock-free network operation without requiring special hardware mechanisms, which is critical for using NoCs in real designs. Our topology synthesis process is integrated with NoC architectural parameter setting and uses accurate switch area, power models and link power models that are obtained from layouts of the components. The presented topology synthesis process is both performance and power consumption aware, which are two of the important design objectives in SoC design. Finally, the topology design process is integrated with an existing design flow and we present ways to ensure design convergence across the levels. The tool flow presented automates the entire NoC design process, including topology synthesis, routing and path computation, RTL code generation and layout generation; thereby bridging an important gap in the design of application-specific NoCs.

An actual layout obtained from an industrial tool (Cadence SoC Encounter [37]) of a 30-core multi-media SoC with the NoC designed using our methodology is presented in Sub-section 6 A. At the layout level, the designed NoC supports the required frequency of operation (close to 900 MHz) without any timing violations. We could design the NoC architecture from input specifications to layout in 4 hours, a process that used to take several weeks. A layout level comparison with a hand-designed architecture for this example is also presented, which shows that our automatic design methodology produces excellent results (in terms of power consumption and performance), matching those of carefully hand-crafted designs. Experiments on several SoC benchmarks show large power, performance and wire-length improvements when compared to standard topologies. Despite the very large design space considered, due to the use of fast algorithms and tools, the design process completes in reasonable time for all the experiments (see Sub-section 6 B).

2. PREVIOUS WORK

A large body of research works exists in synthesizing and generating bus-based systems [9]-[14]. A floorplan-aware point-to-point link design and bus design methodologies are presented in [15] and [14]. While some of the design issues in the NoCs are similar to bus based systems (such as link-width sizing), a large number of issues such as finding the number of required switches, sizing the switches, finding routes for packets, etc. are new in NoCs.

Methods to collect and analyze traffic information that can be fed as input to the bus and NoC design processes have been presented in [12] and [13]. Mappings of cores onto standard NoC topologies have been explored in [16]-[19]. In [17], [19] a floorplanner is used during the mapping process to get area and wire-length estimates. Unlike the method presented here, these works only select topologies from a library of standard topologies. In [18], a unified approach to mapping, routing and resource reservation has been presented. However, the work does not explore topology design process. The NoC design process for supporting multiple applications has been presented in [20]. This research complements our work and its methods can be applied here to support multiple applications as well.

Important research in macro-networks has considered the topology generation problem [21]. As the traffic patterns on these networks are difficult to predict, most approaches are tree-based (like spanning or Steiner trees) and only ensure connectivity with node degree constraints [21]. Hence, these techniques cannot be directly extended to address the NoC synthesis problem. Application-specific custom topology design has been explored earlier in [22]-[25]. The works from [22], [23] do not consider the floorplanning information during the topology design process. In [24], a physical planner is used during topology design to reduce power consumption on wires. However, the work does not consider the area and power consumption of switches in the design. Also, the number and size of network partitions are manually fed. In [25], a slicing tree based floorplanner is used during the topology design process. This work assumes that the switches are located at the corners of the cores and it does not consider the network components (switches, network interfaces) during the floorplanning process. Also, deadlock free routing, which is critical for custom NoC designs is not

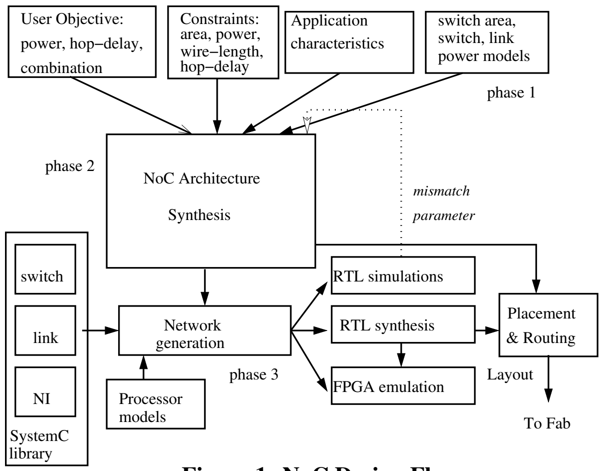
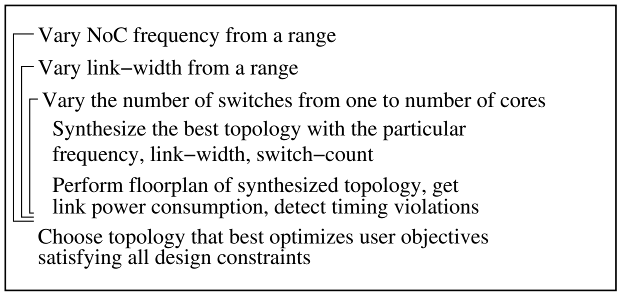

User Objective: power, hop−delay, combination
Constraints: area, power, wire−length, hop−delay
Application characteristics
switch area, switch, link power models
phase 1
phase 2
NoC Architecture Synthesis
mismatch parameter
switch
link
NI
SystemC library
Network generation
Processor models
phase 3
RTL simulations
RTL synthesis
FPGA emulation
Placement & Routing
Layout
To Fab

Figure 1: NoC Design Flow



```
Vary NoC frequency from a range
  Vary link−width from a range
    Vary the number of switches from one to number of cores
      Synthesize the best topology with the particular
      frequency, link−width, switch−count
      Perform floorplan of synthesized topology, get
      link power consumption, detect timing violations
Choose topology that best optimizes user objectives
satisfying all design constraints
```

Figure 2: NoC architecture synthesis (phase 2 of design flow)

supported in the work. Moreover, a complete design space exploration, from architectural parameter setting to simulation is not presented.

Several works exist on automatically generating the *Register Transfer Level (RTL)* code of a designed topology for simulation and synthesis [26]-[28]. These works again complement ours, as the input to them is a designed topology. Building area, power models for on-chip networks has been addressed in [29]-[32].

3. DESIGN FLOW

Our flow for designing NoCs is presented in Figure 1. In the first phase, the user specifies the objectives and constraints that should be satisfied by the NoC. The application traffic characteristics, size of the cores, and the area and power models for the network components are also obtained (see Section 4).

In the second phase of the flow, which is the main contribution of this work, the NoC architecture that optimizes the user objectives and satisfies the design constraints is automatically synthesized. The different steps in this phase are presented in Figure 2. The steps are explained in detail in Section 5. In the outer iterations, the key NoC architectural parameters (NoC frequency of operation and link-width) are varied in a set of suitable values. The bandwidth available on each NoC link is the product of the NoC frequency and the link-width. During the topology synthesis, the algorithm ensures that the traffic on each link is less than or equal to its available bandwidth value.

The synthesis step is performed once for each set of the architectural parameters. In this step, several topologies with different number of switches are explored, starting from a topology where all the cores are connected to one switch, to one where each core is connected to a separate switch. The synthesis of each topology includes finding the size of the switches, establishing the connec-

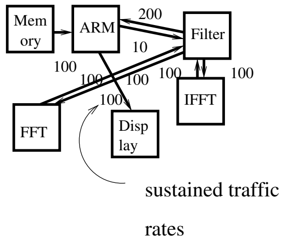


Figure 3: Filter application

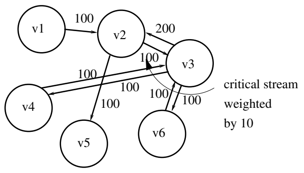


Figure 4: Core graph with sustained rates and critical streams

tivity between the switches and connectivity with the cores, and finding deadlock-free routes for the different traffic flows. In the next step, to have an accurate estimate of the design area and wire-lengths, the floorplanning of each synthesized topology is automatically performed. The floorplanning process finds the 2D position of the cores and network components used in the design. For this, we use Parquet, a fast and accurate floorplanner [35]. Based on the frequency point and the obtained wire-lengths, the timing violations on the wires are detected and the power consumption on the links is obtained. In the last step, from the set of all synthesized topologies and architectural parameter design points, the topology and the architectural configuration that best optimizes the user's objectives, satisfying all the design constraints is chosen. Thus, the output of phase 2 is the best application-specific NoC topology, its frequency of operation and the width of each link in the NoC.

In the last phase of the design (phase 3 in Figure 1), the RTL (SystemC) code of the switches, network interfaces and links for the designed topology is automatically generated. For this, we use the $\times$pipes library [8], [34], a library of soft macros for the network components and the associated tool $\times$pipesCompiler [26] to interconnect the network elements with the cores. At this phase, we also obtain a synthesizable RTL design that can also be emulated on FPGA. From the floorplan specification of the designed topology, the synthesis engine automatically generates the inputs for placement&routing. The placement&routing of the design is performed using SoC Encounter [37] for obtaining the layout, including the global and detailed routing of wires. The output of this phase is a complete layout of the NoC design that can be sent to a foundary.

As the flow has several steps, it is important to close the design gap across the different steps. To ensure that the designed topology will satisfy the timing constraints after place&route, we evaluate the wire-lengths for detecting timing violations early in the design process, i.e. during the topology synthesis phase itself. To bridge the gap between the initial traffic models and the actual observed traffic after simulating the designed NoC, we use a *mismatch* parameter. The parameter is read as part of the input specifications by the topology synthesis engine. The user can manually tune the parameter and re-design the NoC to suit the actual traffic characteristics (explained in Sub-section 6 C). Several other options are also supported by the topology synthesis engine, such as support for cores with fixed locations in the layout (due to pin/pad constraints). Due to lack of space, here we only present the major features of the synthesis process.

4. INPUT MODELS

The traffic characteristics of the application are represented by a graph [16], [17], [19], defined as follows:

DEFINITION 1. *The core 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)*, denoted as* $e_{i,j} \in E$*, representing the communication between the cores* v_i *and* v_j*. The weight of the edge* $e_{i,j}$*, denoted*

Table 2: Component Area-Power

Component	Parameter	Analytical	Experimental
4x4 switch	area(mm^2)	0.036	0.035
	power(mW)	22.16	22.54
5x5 switch	area(mm^2)	0.048	0.047
	power(mW)	28.38	28.70
link (2mm)	power(mW)	0.57	0.57

by $comm_{i,j}$, represents the sustained rate of traffic flow from v_i to v_j weighted by the criticality of the communication. The set F represents the set of all traffic flows, with value of each flow, f_k, $\forall k \in 1 \cdots |F|$, representing the sustained rate of flow between the source (s_k) and destination (d_k) vertices of the flow.

The core graph for a small filter example (Figure 3) is shown in Figure 4. The edges of the core graph are annotated with the sustained rate of traffic flow, multiplied by the criticality level of the flow, as done in [19].

We built accurate analytical models for the power consumption and area of the network components, based on the $\times$pipes architecture [8]. To get the power estimates, the place&route of the components is performed using SoC Encounter and accurate wire capacitances and resistances are obtained, as back-annotated information from the layout, with 0.13μm technology library. The switching activity in the network components is varied by injecting functional traffic. The capacitance, resistance and the switching activity report are combined to estimate power consumption using Synopsys PrimePower [38].

A huge number of implementation runs were performed, varying several parameters such as the number of input, output ports, link-width and the amount of switching activity at the layout level. Linear regression was used to build analytical models for the area and power consumption of the components as a function of these parameters. Due to the intrinsic modularity and symmetry of NoC components, the models built are very accurate (with maximum and mean error of less than 7% and 5%, respectively) when compared to the actual values. Power consumption on the wires is also obtained at the layout level. The analytical and experimental area, power consumption values for some components (with 900 MHz frequency, link-width of 32 bits, buffer depth of 3 in the switches) are presented in Table 2.

5. DESIGN ALGORITHMS

The algorithms for the topology design process are explained in this section. In the first step of Algorithm 1, a design point θ is chosen from the set of available or interesting design points ϕ for the NoC architectural parameters. In our current implementation, the synthesis engine automatically tunes two critical NoC parameters: operating frequency ($freq_\theta$) and link-width (lw_θ). As both frequency and link-width parameters can take a large set of values, considering all possible combinations of values would be infeasible to explore. The system designer has to trim down the exploration space and give the interesting design points for the parameters. The designer usually has knowledge of the range of these parameters. As an example, the designer can choose the set of possible frequencies from minimum to a maximum value, with allowed frequency step sizes. Similarly, the link data widths can be set to multiples of 2, within a range (say from 16 bits to 128 bits). Thus, we get a discrete set of design points for ϕ, as done in [14]. In all our experiments, we support 8 frequency steps and 4 link-width steps, providing 32 discrete design points in the set ϕ. The rest of the topology design process (steps 3-15 in Algorithm 1) is repeated for each design point in ϕ.

As the topology synthesis and mapping problem is NP-hard [22], we present efficient heuristics to synthesize the best topology for the design. For each design point θ, the algorithm synthesizes topologies with different numbers of switches, starting from a design where all the cores are connected through one big switch until the design point where each core is connected to only one switch. The reason for synthesizing these many topologies is that it cannot be predicted beforehand whether a design with few bigger switches would be more power efficient than a design with more smaller switches. A larger switch has more power consumption than a smaller switch to support the same traffic, due to its bigger crossbar and arbiter. On the other hand, in a design with many smaller switches, the packets may need to travel more hops to reach the destination. Thus, the total switching activity would be higher than a design with fewer hops, which can lead to higher power consumption.

For the chosen switch count i, the input core graph is partitioned into i min-cut partitions (step 3). The partitioning is done in such a way that the edges of the graph that are cut between the partitions have lower weights than the edges that are within a partition (refer to Figure 5(a)) and the number of vertices assigned to each partition is almost the same. Thus, those traffic flows with large bandwidth requirements or higher criticality level are assigned to the same partition and hence use the same switch for communication. Hence, the power consumption and the hop-count for such flows will be smaller than for the other flows that cross the partitions. For partitioning, we use Chaco, an efficient hierarchical graph partitioning tool [36].

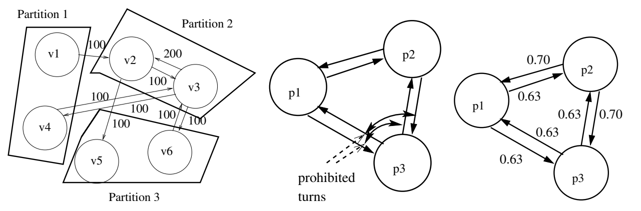
At this point, the communication traffic flows within a partition have been resolved. In steps 5-9, the connections between the switches are established to support the traffic flows across the partitions. In step 5, the *Switch Cost Graph (SCG)* is generated.

DEFINITION 2. *The SCG is a fully connected graph with i vertices, where i is the number of partitions (or switches) in the current topology.*

Please note that the SCG does not imply the actual physical connectivity between the different switches. The actual physical connectivity between the switches is established using the SCG in the *PATH_COMPUTE* procedure, which is explained in the following paragraphs.

In NoCs, wormhole flow control [39] is usually employed to reduce switch buffering requirements and to provide low-latency communication [6], [7]. With wormhole flow control, deadlocks can happen during routing of packets due to cyclic dependencies of resources (such as buffers) [39]. We pre-process the SCG and prohibit certain turns to break such cyclic dependencies. This guarantees that deadlocks will not occur when routing packets. For finding the set of turns that need to be prohibited to break cycles, we use the turn prohibition algorithm presented in [33], [18]. The algorithm has polynomial time complexity (very fast in practice, see Section 6) and guarantees that at most 1/3 of the total number of turns would be prohibited to remove cycles. The algorithm also guarantees connectivity between all nodes in the SCG after prohibiting the turns. From the algorithm, we build the *Prohibited Turn Set (PTS)* for the SCG, which represents the set of turns that are prohibited in the graph. To provide guaranteed deadlock freedom, any path for routing packets should not take these prohibited turns. These concepts are illustrated in the following example:

EXAMPLE 1. *The min-cut partitions of the core graph of the filter example (from Figure 3) for 3 partitions is shown in Figure 5(a). The SCG for the 3 partitions is shown in Figure 5(b). After*



(a) Min-cut partitions (b) SCG graph (c) Path selection

Figure 5: Algorithm examples

applying the turn prohibition algorithm from [33], the set of prohibited turns is identified. In Figure 5(b), the prohibited turns are indicated by circular arcs in the SCG. For this example, both the turns around the vertex P3 are prohibited to break cycles. So no path that uses the switch P3 as an intermediate hop can be used for routing packets.

Our topology synthesis process also supports freedom from another type of deadlock, known as message-level deadlock [39], by routing the traffic flows of the different message types in the design onto different physical links. Due to lack of space, we do not explain this in detail in this paper.

Algorithm 1 Topology Design Algorithm

```
1:  Choose design point $\theta$ from $\phi$: $freq_\theta$, $lw_\theta$
2:  for $i$ = 1 to $|V|$ do
3:      Find i min-cut partitions of the core graph
4:      Establish a switch with $N_j$ inputs and outputs for each partition, $\forall j \in 1 \cdots i$. $N_j$ is the number of vertices (cores) in partition $i$. Check for bandwidth constraint violations.
5:      Build Switch Cost Graph (SCG) with edge weights set to 0
6:      Build Prohibited Turn Set (PTS) for SCG to avoid deadlocks
7:      Set $\rho$ to 0
8:      Find paths for flows across the switches using function PATH_COMPUTE(i, SCG, $\rho$, PTS, $\theta$)
9:      Evaluate the switch power consumption and average hop-count based on the selected paths
10:     Repeat steps 8 and 9 by increasing $\rho$ value in steps, until the hop-count constraints are satisfied or until $\rho$ reaches $\rho_{thresh}$
11:     If $\rho_{thresh}$ reached and hop-count not satisfied, go to step 2.
12:     Perform floorplan and obtain area, wire-lengths. Check for timing violations and evaluate power consumption on wires
13:     If target frequency matches or exceeds $freq_\theta$, and satisfies all constraints, note the design point
14: end for
15: Repeat steps 2-14 for each design point available in $\theta$
16: For the best topology and design point, generate information for ×pipesCompiler and Cadence SoC Encounter
```

The actual physical connections between the switches are established in step 8 of Algorithm 1 using the *PATH_COMPUTE* procedure. The objective of the procedure is to establish physical links between the switches and to find paths for the traffic flows across the switches. Here, we only present the procedure where the user's design objective is to minimize power consumption. The procedure for the other two cases (with hop-count as the objective and with linear combination of power and hop-count as objective) follow the same algorithm structure, but with different cost metrics.

An example illustrating the working of the *PATH_COMPUTE* procedure is presented in Example 2. In the procedure, the flows are ordered in decreasing rate requirements, so that bigger flows are assigned first. The heuristic of assigning bigger flows first has been shown to provide better results (such as lower power consumption and more easily satisfying bandwidth constraints) in several earlier works [17], [18]. For each flow in order, we evaluate the amount of power that will be dissipated across each of the switches, if the traffic for the flow used that switch. This power dissipation value on each switch depends on the size of the switch, the amount of traffic already routed on the switch and the architectural parameter point (θ) used. It also depends on how the switch is reached (from what other switch) and whether an already existing physical channel will be used to reach the switch or a new physical channel will have to be opened. This information is needed, because opening a new physical channel increases the switch size and hence the power consumption of this flow and of the others that are routed through the switch. These marginal power consumption values are assigned as weights on each of the edges reaching the vertex representing that switch in the SCG. This is performed in steps 8 and 11 of the procedure. When opening a new physical link, we also check whether the switch size is small enough to satisfy the particular frequency of operation. As the switch size increases, the maximum frequency of operation it can support reduces (as the critical path inside the switch gets longer) [8]. This information is obtained from the placement&routing of the switches, taken as an input to the algorithms.

Algorithm 2 PATH_COMPUTE(i, SCG, ρ, PTS, θ)

```
1:  Initialize the set $PHY(i1, j1)$ to false and $Bw\_avail(i1, j1)$ to $freq_\theta \times lw\theta, \forall\, i1,\ j1 \in 1 \cdots i$
2:  Initialize $switch\_size\_in(j)$ and $switch\_size\_out(j)$ to $N_j$, $\forall\ j \in\ 1 \cdots i$. Find $switching\_activity(j)$ for each switch, based on the traffic flow within the partition.
3:  for each flow $f_k$, $k \in 1 \cdots |F|$ in decreasing order of $f_c$ do
4:      for i1 from 1 to i and j1 from 1 to i do
5:          {Find the marginal cost of using link i1, j1}
6:          {If physical link exists and can support the flow}
7:          if $PHY(i1, j1)$ and $Bw\_avail(i1, j1) \geq f_c$ then
8:              Find $cost(i1, j1)$, the marginal power consumption to re-use the existing link
9:          else
10:             {We have to open new physical link between i1, j1}
11:             Find $cost(i1, j1)$, the marginal power consumption for opening and using the link. Evaluate whether switch frequency constraints are satisfied.
12:         end if
13:     end for
14:     Assign $cost(i1, j1)$ to the edge $W(i1, j1)$ in SCG
15:     Find the least cost path between the partitions in which source ($s_k$) and destination ($d_k$) of the flow are present in the SCG. Choose only those paths that have turns not prohibited by PTS
16:     Update $PHY$, $Bw\_avail$, $switch\_size\_in$, $switch\_size\_out$, $switching\_activity$ for chosen path
17: end for
18: Return the chosen paths, switch sizes, connectivity
```

Once the weights are assigned, choosing a path for the traffic flow is equivalent to finding the least cost path in the SCG. This is done by applying Dijkstra's shortest path algorithm [40] in step 15 of the procedure. When choosing the path, only those paths that do not use the turns prohibited by PTS are considered. The size of the switches and the bandwidth values across the links in the chosen path are updated and the process is repeated for other flows.

EXAMPLE 2. *For the SCG from Example 1, let us consider routing the flow of value* 100 *between the vertices* $v1$ *and* $v2$*, across* *the partitions $p1$ and $p2$. Initially no physical paths have been established across any of the switches. If we have to route the flow across a link between any two switches, we have to first establish the link. The cost of routing the flow across any pair of switches is obtained from step 11 of the PATH_COMPUTE procedure. The SCG with the edges annotated with the costs is presented in Figure 5(c). The costs on the edges from $p2$ are different from the others due to the difference in initial switching activity in $p2$ compared to the other switches. This is because the switch $p2$ has to support flows between the vertices $v2$ and $v3$ within the partition. The least cost path for the flow, which is across switches $p1$ and $p2$ is chosen. Now we have actually established a physical path between these switches and this is considered when routing the other flows. Also, the size and switching activity of these switches have changed, which is noted.*

The *PATH_COMPUTE* procedure returns the sizes of the switches, connectivity between the switches and the paths for the traffic flows. The objective function for establishing the paths is initially set to minimizing power consumption in the switches. Once the paths are established, if hop-count constraints are not satisfied, the algorithm gradually modifies the objective function to minimize the hop-count as well, using the parameter ρ (in steps 7, 10 and 11 of Algorithm 1). The upper bound for ρ, denoted by ρ_{thresh}, is set to the value of power consumption of the flow with maximum rate, when it crosses the maximum size switch in the SCG. At this value of ρ, for all traffic flows, it is beneficial to take the path with least number of switches, rather than the most power efficient path. The ρ value is varied in several steps until the hop-count constraints are satisfied or until it reaches ρ_{thresh}.

In the next step (step 12, Algorithm 1), the algorithm invokes the floorplanner to compute the design area and wire-lengths. The floorplanner minimizes a dual-objective function of area and wire-length, with equal weights assigned to both. The floorplanner used [35] also supports soft cores, fixed pin/pad locations and aspect ratio constraints for the generated design. From the obtained wire-lengths, the power consumption across the wires is calculated. Also, the length of the wires is evaluated to check any timing violations that may occur at the particular frequency ($freq_{\theta}$). In the end, the tool chooses the best topology (based on the user's objectives) that satisfies all the design constraints. At the last step, for the synthesized topology, the algorithm automatically generates the information required for the ×pipesCompiler tool for network instantiation and the SoC Encounter tool to perform placement&routing.

The presented NoC synthesis process scales polynomially with the number of cores in the design. The number of topologies evaluated by the methodology also depends linearly on the number of cores. Thus, the algorithms are highly scalable to a large number of cores and communication flows. The synthesis time for several different SoC benchmarks is presented in Section 6 B.

6. EXPERIMENTS AND CASE STUDIES

6.1 Layout-level Comparisons

We had earlier manually developed a NoC design for a SoC that runs multi-media benchmarks [34]. The design consists of 30 cores: 10 ARM7 processors with caches, 10 private memories (a separate memory for each processor), 5 custom traffic generators, 5 shared memories and devices to support inter-processor communication. The hand-designed NoC has 15 switches connected in a 5x3 quasi-mesh network (2 cores connected to each switch), shown in Figure 6(a). The design is highly optimized, with the private memories being connected to the processors across a single switch and the shared memories distributed around the switches. The layout of the design (presented in Figure 6(b)) was performed using SoC Encounter and the mesh structure was maintained in the layout. Each of the cores has an area of 1 mm^2 [34] in the design. The entire process, from topology specification to layout generation took several weeks. The post-layout NoC could support a maximum frequency of operation of 885 MHz, which is determined by the critical path in the switch pipeline. The power consumption of the topology for functional traffic has been evaluated to be 368 mW.

We apply our topology synthesis process with the objective of minimizing power consumption, to automatically synthesize the NoC for this application. We set the design constraints and the required frequency of operation to be the same (885 MHz) as that of the hand-designed topology. The synthesized NoC topology and the layout obtained using SoC Encounter are presented in Figures 6(c) and 6(d). The synthesized topology has fewer switches (8 switches) than the hand-designed topology. It can support the same maximum frequency of operation (885 MHz), without any timing violations on the wires. As we considered the wire-lengths during the synthesis process to estimate the frequency that could be supported, we could synthesize the most power efficient topology that would still meet the target frequency. To reach such a design point manually would require several iterations of topology design and place&route phases, which is a very time consuming process.

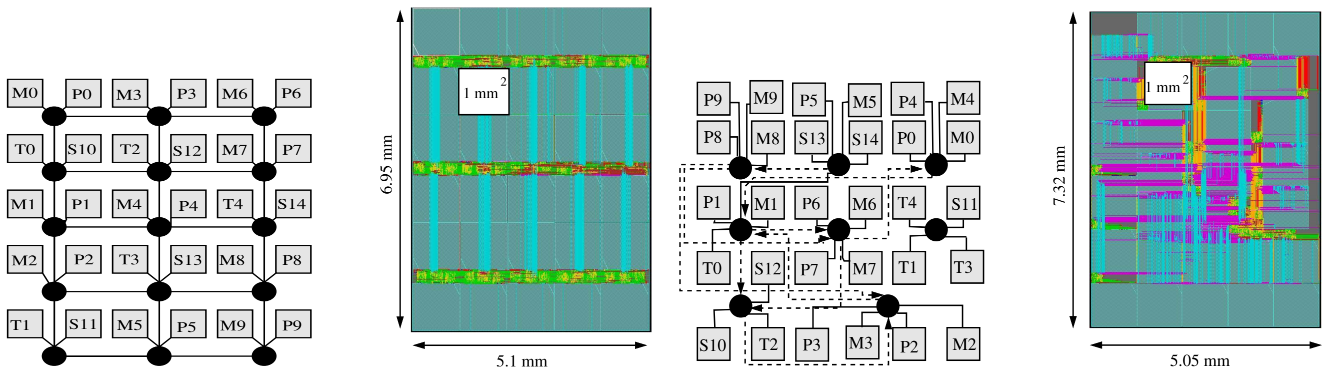
Layout level power consumption calculations on functional traffic show that the synthesized topology has 277 mW power consumption, which is $1.33\times$ lower than the hand-designed topology. Given the fact that the hand-designed topology is highly optimized, with much of the communicating traffic (which is between the ARM cores and their private memories) traversing only one switch, these savings are achieved entirely from efficiently spreading the shared memories around the different switches. The layout of the hand-designed NoC was manually optimized to a large extent (by moving switches, network interfaces) to reduce the area of the design. The layout of the synthesized topology is obtained completely automatically, and still the area of the design is close to that of the manual design (only a marginal 4.3% increase in area).

We perform cycle-accurate simulations of the hand-designed and the synthesized NoCs for two multimedia benchmarks. The total application time for the benchmarks (including computation time) and the average packet latencies for read transactions for the topologies are presented in Figures 7(a) and 7(b). The custom topology not only matches the performance of the hand-designed topology, but provides an average of 10% reduction in total execution time and of 11.3% in packet latency.

6.2 Experiments on SoC Benchmarks

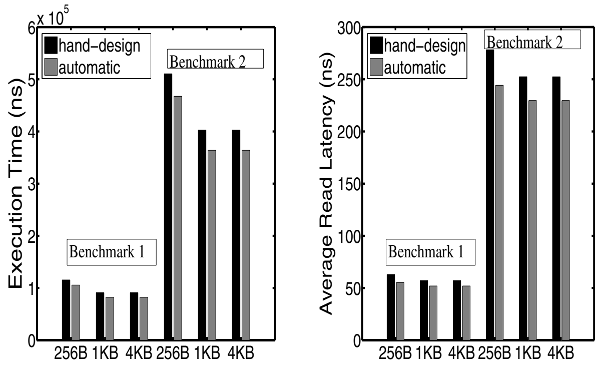
We have applied our topology design procedure to six different SoC benchmarks: *video processor (VPROC-42 cores), MPEG4 decoder (12 cores), Video Object Plane Decoder (VOPD-12 cores), Multi-Window Display application (MWD-12 cores), Picture-in-Picture application (PIP-8 cores) and IMage Processing application (IMP-23 cores).* We refer the readers to [7] for the communication characteristics of some of these benchmarks.

For comparison, we have also generated mesh topologies for the benchmarks by modifying the design procedure to synthesize NoCs based on mesh structure. To obtain mesh topologies, we generate a design with each core connected to a single switch and restrict the switch sizes to have 5 input/output ports. We also generated a variant of the basic mesh topology: *optimized mesh (opt-mesh)*, where those ports and links that are unused by the traffic flows are removed.



(a) Hand-designed topology (b) Layout (c) Automatically synthesized (d) Layout

Figure 6: (a), (b) Hand-designed topology and layout. M: ARM7 processors, T: traffic generators, P, S: private and shared slaves (c), (d) Automatically synthesized topology and layout. In Figure (c), bi-directional links are solid and uni-directional links are dotted.



(a) Execution time (b) Average read latency

Figure 7: Run time and latency for different cache sizes

The core graph and the floorplan for the custom topology synthesized by our tool for one of the benchmarks (VOPD) are shown in Figure 8. The network power consumption (power consumption across the switches and links), average hop-count and design area results for the different benchmarks are presented in Table 3. Note that the average hop-count is the same for mesh and opt-mesh, as in the opt-mesh only the unused ports and links of the mesh have been removed and the rest of the connections are maintained. The custom topology results in an average of $2.78\times$ improvement in power consumption and $1.59\times$ improvement in hop-count when compared to the standard mesh topologies. The area of the designs with the different topologies is similar, thanks to efficient floorplanning of the designs. It can be seen from Figure 8 that only very little slack area is left in the floorplan. This is because we consider the area of the network elements during the floorplanning process, and not after the floorplanning of blocks. The total run time of the topology synthesis and architectural parameter setting process for the different benchmarks is presented in Table 3. Given the large problem sizes and very large solution space that is explored (8 different frequency steps, 4 different link-widths, 42 cores for VPROC and several calls to the floorplanner) and the fact that the NoC parameter setting and topology synthesis are important phases, the run-time of the engine is not large. This is mainly due to the use of hierarchical tools for partitioning and floorplanning and our development of fast heuristics to synthesize the topology.

We also performed comparisons of synthesized topology against several other standard topologies. For mapping the cores onto the standard topologies, we use the tool from [17]. As the power libraries used for switches, links in the tool are different from the

Table 3: Comparisons with standard topologies

Appl	Topol.	Power (mW)	Avg. Hops	Area mm^2	Time (mins)
VPROC	custom	79.64	1.67	47.68	68.45
	mesh	301.8	2.58	51.0	
	opt-mesh	136.1	2.58	50.51	
MPEG4	custom	27.24	1.5	13.49	4.04
	mesh	96.82	2.17	15	
	opt-mesh	60.97	2.17	15.01	
VOPD	custom	30.0	1.33	23.56	4.47
	mesh	95.94	2.0	23.85	
	opt-mesh	46.48	2.0	23.79	
MWD	custom	20.53	1.15	15	3.21
	mesh	90.17	2.0	13.6	
	opt-mesh	38.60	2.0	13.8	
PIP	custom	11.71	1	8.95	2.07
	mesh	59.87	2.0	9.6	
	opt-mesh	24.53	2.0	9.3	
IMP	custom	52.13	1.44	29.66	31.52
	mesh	198.9	2.11	29.4	
	opt-mesh	80.15	2.11	29.4	

ones used in the synthesis process, we optimized the topologies for performance, subject to the design constraints. The comparisons against 5 standard topologies (mesh, torus, hypercube, Clos and butterfly) for an image processing benchmark with 25 cores is presented in Figure 9. The custom topology synthesized by our method shows large performance improvements (an average of $1.73\times$) over the standard topologies.

As an interesting observation, we found that prohibiting certain turns to avoid deadlocks during routing had a negligible impact on the power and performance results for all of the benchmarks. This was because, even if some turns were avoided, the path computation procedure could easily find other paths with low cost, as several alternative low cost paths exist between each source and destination in the SCG (refer to Section 5).

6.3 Handling Dynamic Effects

When the designed NoC is simulated, there can be some mismatch between the observed traffic patterns and the initial traffic estimates. This may be either because of inaccurate traffic models or because of dynamic effects, such as congestion. Note that it will be too time consuming to simulate each topology during the synthesis process. To bridge the gap between topology synthesis and simulation, we use the *mismatch* parameter; the input traffic rates are multiplied by the value of this parameter. The parameter is fed as an input to the synthesis engine. It is initially set to 1 and the user can manually tune the parameter and re-design the NoC, until the simulations satisfy the required performance level. The

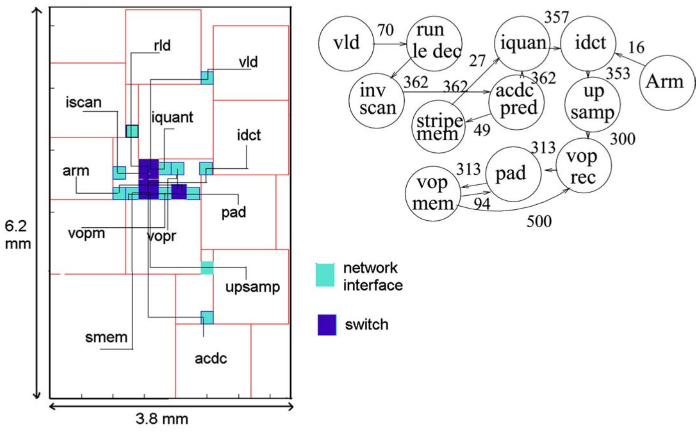


Figure 8: VOPD custom topology floorplan and core graph

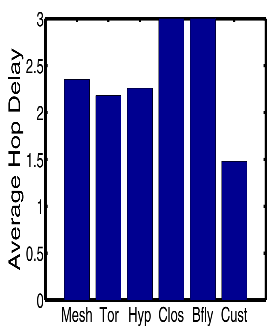


Figure 9: Performance comparisons

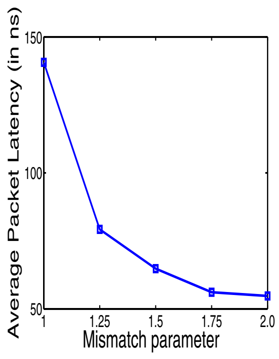


Figure 10: Dynamic effects

effect of increasing the parameter on performance for the MPEG4 NoC is presented in Figure 10. Extensions of the concept to handle localized congestion effects in the NoC are currently underway.

7. CONCLUSIONS

To have a power and latency efficient design, the communication architecture should closely match the application traffic characteristics, satisfying the different design constraints. Synthesizing such *Network on Chip (NoC)* architecture is non-trivial, given the large design space that needs to be explored. In this work, we have presented a methodology that automates the process, generating efficient NoCs that satisfy the design constraints of the application. To have fewer design re-spins and faster time-to-market, we consider fast and accurate floorplan information early in the design cycle. This leads to detecting timing violations on the NoC links during the NoC synthesis phase, thereby leading to timing closure with quicker convergence between the high level design and the physical design phases. We use accurate switch and link power models that are based on layouts of the components and accurate link power estimates based on the wire-lengths obtained from floorplanning. We also integrate deadlock free routing methods in the NoC synthesis process, which is critical for proper NoC operation. Experiments on several SoC benchmarks show that the synthesized topologies are much better (an average of $2.78\times$ power reduction, $1.59\times$ hop-count reduction) than the best mesh topology and mesh-based custom topologies for our case studies.

8. REFERENCES

[1] S. Dutta et al., "Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems", IEEE D&T, Sep/Oct 2001, pp. 21-31.
[2] http://www.st.com
[3] http://www.ti.com.
[4] L.Benini and G.De Micheli, "Networks on Chips: A New SoC Paradigm", IEEE Computers, pp. 70-78, Jan. 2002.
[5] D.Wingard,"MicroNetwork-Based Integration for SoCs", Proc. DAC, pp. 673-677, Jun 2001.
[6] K. Goossens et al., "A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification", DATE 2005.
[7] D. Bertozzi et al., "NoC Synthesis Flow for Customized Domain Specific Multi-Processor Systems-on-Chip", IEEE TPDS, Feb 2005.
[8] S. Stergiou et al., "×pipesLite: a Synthesis Oriented Design Library for Networks on Chips", pp. 1188-1193, Proc. DATE 2005.
[9] J. Daveau et al., "Synthesis of system-level communication by an allocation based approach", Proc. ISSS, pp. 150-155, Sept. 1995.
[10] M. Gasteier, M. Glesner, "Bus-based communication synthesis on system level", ACM TODAES, vol.4, no.1, pp. 1-11, 1999.
[11] K. Ryu, V. Mooney, "Automated Bus Generation for Multiprocessor SoC Design", Proc. DATE, pp. 282-287, March 2003.
[12] K.Lahiri et al., "Design Space Exploration for Optimizing On-Chip Communication Architectures", IEEE TCAD, vol.23, no.6, pp. 952- 961, June 2004.
[13] S. Murali, G. De Micheli, "An Application-Specific Design Methodology for STbus Crossbar Generation", pp. 1176-1181, Proc. DATE '05.
[14] S. Pasricha et al., "Floorplan-aware automated synthesis of bus-based communication architectures", Proc. DAC '05.
[15] J. Hu et al., "System-Level Point-to-Point Communication Synthesis Using Floorplanning Information", Proc. ASPDAC '02.
[16] J. Hu, R. Marculescu, 'Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures', Proc. DATE, March 2003.
[17] S. Murali, G. De Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs", Proc. DAC 2004.
[18] 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, April 2005.
[19] S. Murali et al., "Mapping and Physical Planning of Networks on Chip Architectures with Quality-of-Service Guarantees", Proc. ASPDAC 2005.
[20] S. Murali et al., "A Methodology for Mapping Multiple Use-Cases onto Networks on Chips", pp. 1-6, Proc. DATE, 2006.
[21] R. Ravi et al., "Approximation algorithms for degree-constrained minimum-cost network design problems", Algorithmica, 31(1): 58-78, 2001.
[22] A.Pinto et al., "Efficient Synthesis of Networks on Chip", ICCD 2003, pp. 146-150, Oct 2003.
[23] W.H.Ho, T.M.Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns", HPCA 2003, pp. 377-388, Feb 2003.
[24] T. Ahonen et al. "Topology Optimization for Application Specific Networks on Chip", Proc. SLIP 04.
[25] K. Srinivasan et al., "An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks", Proc. ICCAD '05.
[26] A. Jalabert et al., "×pipesCompiler: A tool for instantiating application specific networks-on-chip", pp. 884-889, Proc. DATE 2005.
[27] D.Siguenza-Tortosa, J. Nurmi, "Proteo: A New Approach to Network-on-Chip", in CSN 02, Sep. 2002.
[28] X.Zhu, S.Malik, "A Hierarchical Modeling Framework for On-Chip Communication Architectures", ICCD 2002, pp. 663-671, Nov 2002.
[29] T. T. Ye et al., "Analysis of power consumption on switch fabrics in network routers", Proc. DAC '03.
[30] H-S Wang et al., "Orion: A Power-Performance Simulator for Interconnection Network", Proc. Micro, Nov 2002.
[31] N. Banerjee et al., "A power and performance model for network-on-chip architectures", Proc. DATE '04.
[32] G. Palemoro, C. Silvano, "PIRATE: A Framework for Power/Performance Exploration of Network-On-Chip Architectures", PATMOS 2004
[33] D. Starobinksi et al., "Application of network calculus to general topologies using turn-prohibition", IEEE/ACM Transactions on Networking, Vol. 11, Issue 3, pp. 411-421, June 2003.
[34] F. Angiolini et al., "Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness", pp. 124-129, Proc. DATE 2006.
[35] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning : Enabling Hierarchical Design", IEEE Trans. on VLSI Systems, vol 11(6), pp. 1120-1135, Dec 2003. URL: http://vlsicad.eecs.umich.edu/BK/parquet/
[36] B. Hendrickson, R. Leland, "The Chaco User's Guide: Version 2.0", Sandia Tech Report SAND94–2692, 1994. URL: //www.cs.sandia.gov/b̃ahendr/chaco.html
[37] www.cadence.com
[38] www.synopsys.com
[39] W. J. Dally, B. Towles, "Principles and Practices of Interconnection Networks", Morgan Kaufmann , Dec 2003.
[40] T. H. Cormen et al., "Introduction to Algorithms", The MIT Press, June 1990.