A Methodology for Mapping Multiple Use-Cases onto Networks on Chips

Srinivasan Murali
CSL, Stanford University
Stanford, USA
smurali@stanford.edu

Martijn Coenen, Andrei Radulescu, Kees Goossens
Philips Research Laboratories
The Netherlands
{martijn.coenen,andrei.radulescu,kees.goossens}@philips.com

Giovanni De Micheli
LSI, EPFL
Switzerland
giovanni.demicheli@epfl.ch

Abstract

A communication-centric design approach, Networks on Chips (NoCs), has emerged as the design paradigm for designing a scalable communication infrastructure for future Systems on Chips (SoCs). As technology advances, the number of applications or use-cases integrated on a single chip increases rapidly. The different use-cases of the SoC have different communication requirements (such as different bandwidth, latency constraints) and traffic patterns. The underlying NoC architecture has to satisfy the constraints of all the use-cases. In this work, we present a methodology to map multiple use-cases onto the NoC architecture, satisfying the constraints of each use-case. We present dynamic re-configuration mechanisms that match the NoC configuration to the communication characteristics of each use-case, also accounting for use-cases that can run in parallel. The methodology is applied to several real and synthetic SoC benchmarks, which result in a large reduction in NoC area (an average of 80%) and power consumption (an average of 54%) compared to traditional design approaches.

Keywords: Networks on Chips, Systems on Chips, Use-Cases, Modes, Dynamic Re-Configuration.

1 Introduction

Systems on Chips are high-complexity, high-value chips that are used in a wide variety of areas such as cellular phones, TV processors, set-top boxes. As technology advances, it becomes cost-effective to integrate several different applications or use-cases onto a single SoC chip. As an example, the *PNX8550* (*Viper2*) set-top box SoC based on the Philips Nexperia platform has multiple resolution video processing capabilities (like high definition, standard definition), multiple picture modes (like split-screen, picture-in-picture), video recording features, high speed internet access, file transfer services, etc. [10], [11].

Current state-of-the-art SoCs also allow several of the use-cases to run in parallel. As an example, in a set-top box SoC, video display and recording applications can run in parallel, where the recorder could potentially record a different program than what is being displayed on the screen. We refer to such use-cases that run in parallel as *compound modes* (Figure 1). The transition between the single use-case mode to compound mode needs to be smooth. As an example, when we start a new function such as video-recording in a set-top box, the video display that is currently

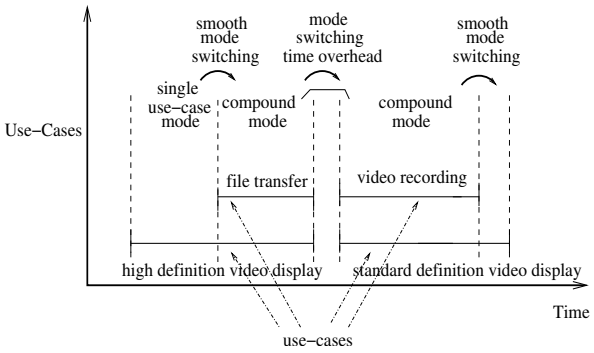


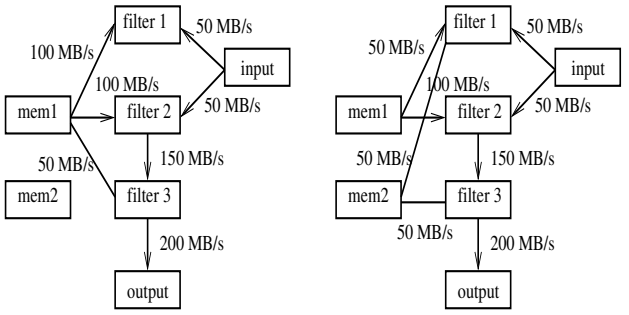
Figure 1. Use-cases and compound modes

going on should be unaffected. However, when there is a switching between compound modes, there can be a configuration time overhead to load the new set of use-cases, as shown in Figure 1.

As the communication requirements of the SoCs increase, current single or multiple bus-based solutions become inefficient in terms of throughput and performance [1]. A communication-centric design approach, *Networks on Chips (NoCs)*, has recently emerged as the design paradigm for designing a scalable communication infrastructure for future *SoCs* [1]-[5]. The NoC architecture for the design should closely match the traffic characteristics and performance requirements of the different use-cases. As the different use-cases have different functionalities, the communication characteristics can be very different across the use-cases. As an example, in Figure 2, a small fragment of the communication constraints for two different use-cases for the *Viper2* set-top box SoC is presented, where the bandwidth requirements for some of the traffic streams for the use-cases are quite different.

One of the important phases of the NoC design is to build a network that can support the communication constraints and traffic characteristics of each use-case. The design process includes mapping of the cores onto the NoC components, such as the switches, network interfaces (NIs) and configuration of the NoC to support the traffic flows. The NoC configuration has several sub-phases such as finding paths and reserving resources for the various traffic flows in the NoC. In most current SoC designs, the interconnect architecture is manually designed to support the different

3-9810801-0-6/DATE06 © 2006 EDAA



(a) Use-case 1 (b) Use-case 2

Figure 2. Example use-cases

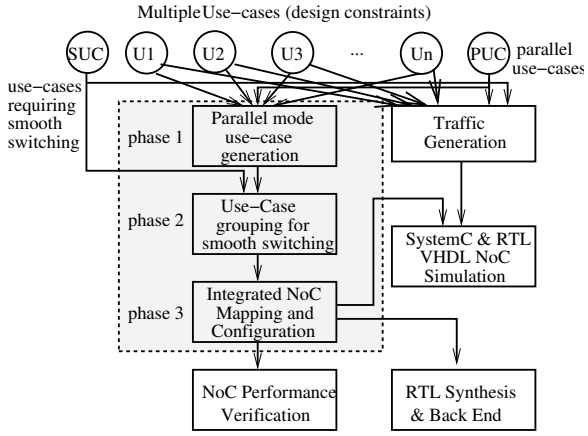


Figure 3. Multi use-case NoC design methodology

use-cases [12]. As the SoCs can have several hundred use-cases, manually checking whether the design constraints of the individual use-cases are satisfied by the NoC is a tedious process. Even if such checking can be done, converging to a single design for the NoC that supports the different use-cases is a non-trivial problem. If the NoC is designed to run only one of the use-cases, such as the most communication intensive use-case, it may not be able to support the constraints of the other use-cases.

In this work we present a design methodology for mapping, path selection and resource reservation in the NoC that satisfies the communication constraints of multiple use-cases of the SoC. We consider compound modes, where two or more use-cases run in parallel, and automatically compute the communication constraints for such modes from the constituent use-cases. When there is switching between the use-cases that are run, there is a possibility of changing the paths and resource reservations in the NoC across the use-cases. The dynamic network re-configuration can be applied when the use-case switching times are large and it helps in reducing the operating frequency and power consumption of the NoC. In our methodology, we pre-process the use-cases and identify the set of use-cases that need to share the same NoC configuration and use-case switching where the NoC configuration can be changed. We also explore the effect of dynamic voltage and frequency scaling (DVS/DFS) techniques for reducing the power consumption of the network across the different use-cases. We apply our methods to several SoC designs (set-top box, TV processor SoCs) and synthetic benchmarks to validate the design methodology. The methods are scalable to a large number of use-cases and are applicable even when the use-cases have very different communication characteristics.

2 Previous Work

A large body of research focuses on the architecture [5]-[8] and automating the design flow for NoCs [22], [23]. We refer the reader to [21] for an overview of the various issues in the design of NoCs.

In [9], the *Æthereal* architecture is presented. It supports Quality-of-Service (QoS) for applications by using Guaranteed Throughput (GT) connections for traffic streams that have bandwidth/latency constraints and by using Best Effort (BE) connections for the remaining traffic streams. Mapping and topology selection/generation for a single use-case has been explored by several researchers [13]-[20].

The multi-use case mapping problem is addressed in [25], where a method to map multiple use-cases onto NoCs is presented. The approach is based on building a synthetic worst-case use-case that includes the constraints of all the use-cases and to design and optimize the NoC based on the worst-case use-case. Such an approach results in a NoC that satisfies the design constraints of all the use-cases and performs well on SoCs that have similarity in the traffic patterns across the different use-cases and when there are a small number of use-cases. However, it performs poorly on systems where the traffic characteristics of the use-cases are very different or when the number of use-cases is large. This is because the worst-case use-case has highly over-specified constraints (as it is based on the worst-case combinations) and leads to a large NoC design. As an example, for a synthetic 20 use-case 20 core design, the size of the NoC designed using the method presented here is less than 10% of the size of the NoC produced using the method in [25] (details presented in Section 6.2 of this work). As the number of use-cases on a SoC is increasing with each new platform, scalability of the mapping process is critical. The methodology presented in this work is scalable to a large number of use-cases and is applicable even when the communication patterns in the use-cases are very different.

3 Design Methodology

The communication characteristics and constraints of the various use-cases of a SoC are the input to our design methodology ($U1 \cdots Un$ in Figure 3). The user specifies the set of use-cases that can run in parallel (PUC in the Figure). In the first phase of the design process, new use-cases are generated automatically to represent such parallel modes of operation.

For a multiple use-case SoC, when the system switches between use-cases, some timing overhead is incurred in loading the new use-case. This delay is mainly due to the fact that the new use-case's data and code need to be loaded, control signals need to be distributed to different parts of the design and the already running use-case need to be gracefully shut down. This switching time varies with different use-cases and depends on the underlying computational architecture. Some use-cases represent control sequences that are critical and are loaded and run quickly. For many other use-cases, the switching time is of the order of hundreds of micro-seconds to milli-seconds. In this time, we can reconfigure the paths and TDMA slot-tables in the NoC to match the communication characteristics of the use-cases. We can also scale the supply voltage and NoC frequency to match the use-case characteristics that can lead to a reduction in the NoC power consumption. A description of the ways to achieve the re-configuration and the associated overhead (in terms of time, resource usage, energy) is pre-

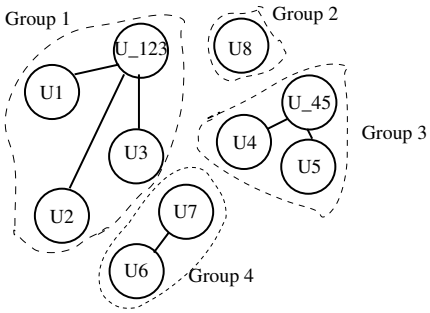


Figure 4. Smooth switching graph

sented in our earlier work [25].

When some use-cases can run in parallel, we require a smooth transition between the single use-case mode to the parallel use-case mode and the network configuration should not be changed. Other use-cases that require smooth switching between them are given as an input to the design flow (*SUC* in Figure 3). In the second phase of the design, we pre-process the use-cases identifying those set of use-cases that can have re-configuration and those that should share the same NoC configuration. The detailed description of this phase is presented in Section 4.

In the third phase of the design, we perform mapping and NoC configuration. The objective of the mapping process is to design the smallest size NoC (in terms of the number of the switches used) that satisfies the design constraints of all the use-cases. We assume that all of the use-cases utilize the same mapping of cores onto the NoC components and only the paths and TDMA slot-tables can be potentially re-configured across the different use-cases. This is because, if each individual use-case has a different mapping, then each core potentially needs to be connected to several different NIs, which may not be feasible because of physical layout restrictions and wiring complexity. The methods presented in this paper can be easily extended to support even limited re-configuration of the mapping across the different use-cases.

In the last phase of the design, the SystemC/VHDL code for the NoC design is generated and simulations of the design are performed. The NoC performance for the GT connections is also verified analytically in this step.

4 Use-Case Pre-Processing

The set of use-cases that can run in parallel is specified by the user as an input. As the number of combinations of the use-cases can be large, it is a tedious process for the user to manually create use-cases to represent the parallel modes. In the first phase of the methodology we automatically compute the bandwidth, latency requirements for such parallel modes from the individual use-cases. The bandwidth of a flow between two cores in such a compound mode is obtained by summing the bandwidth of the flows between the two cores across the use-cases that comprise the mode and the latency requirement of the flow is taken to be the minimum of the requirements of the flows across the different use-cases in the mode. Such compound modes are then taken as separate use-cases in the design flow.

To capture the constraints that certain use-cases need smooth switching between them (and hence should have the same NoC configuration), we obtain the set of such use-cases as an input from the user. We automatically consider those use-cases in a compound-mode to also require smooth-switching. Once the set of use-cases are obtained, we construct a switching graph SG:

Definition 1 *The switching graph is an undirected graph, $SG(SV, SE)$ with each vertex $sv_i \in SV$ representing an use-case and the undirected edge (sv_i, sv_j) (or (sv_j, sv_i)), representing the fact that the use-cases sv_i and sv_j require smooth switching between them.*

As an example, in Figure 4, a SG graph for 10 use-cases is presented. The use-cases U_123, U_45 are automatically generated by the first phase of the design flow to represent the compound modes of operation where use-cases 1, 2, 3 and 4, 5, respectively, run in parallel. We require a smooth switching between use-cases 6 and 7, as use-case 7 is considered to be critical. The set of use-cases that need to have the same NoC configuration have an edge between them in the SG graph.

To find the set of all use-cases that need to have the same NoC configuration, we use the algorithm presented in Algorithm 1. In the algorithm, the SG graph is traversed and those vertices that are reachable from each other are grouped. The vertices in the same group represent those use-cases that need to have the same NoC configuration. This is obtained by performing depth-first search of the SG graph, possibly multiple times, until all vertices are traversed. The set of vertices traversed in a single search are grouped together, as they are reachable from each other. During the mapping process, the set of use-cases that are in the same group utilize the same NoC configuration.

Algorithm 1 Use-Case Grouping

1. Initialize $sv_i \in SV, \forall i \in 1 \cdots |SV|$, unvisited.
2. Choose unvisited vertex $v \in SV$ and mark it visited.
3. Perform depth first search from v on SG. Group all vertices traversed in the search and mark them visited.
4. Remove visited vertices and their edges from SG.
5. Repeat steps 2-4 until all vertices in SG are visited.

5 Unified Mapping-NoC Configuration

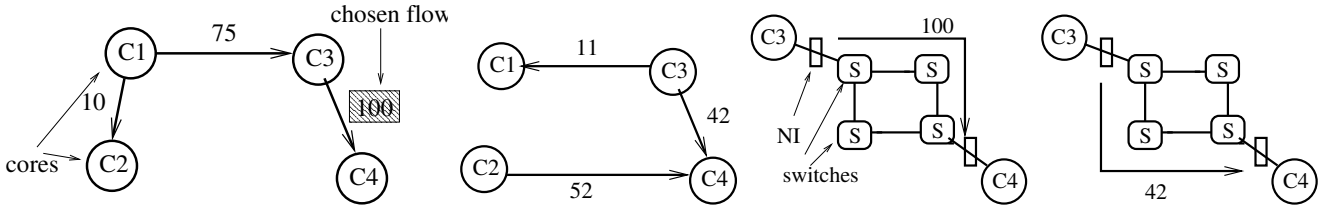
As graph mapping is an instance of the NP-Hard quadratic assignment problem [13], [16], we use a heuristic algorithm to perform the multi use-case mapping. The basic algorithm for a single use-case for the *Æthereal* architecture is presented in [20], and in this section, we extend the approach to consider multiple use-cases. Unlike previous works in the domain [13]-[19], in our approach, the selection of paths for the different traffic flows and the reservation of TDMA slot-table entries for the GT traffic flows are unified with the mapping process. Such a unified mapping-NoC configuration mechanism leads to quicker pruning of solution search space and results in quick convergence to a mapping that satisfies the design constraints.

To perform the mapping, let us formulate the following:

Definition 2 *Let the set of use-cases be U. The communication between set of all pairs of cores in an use-case $i, \forall i \in 1 \cdots |U|$, is represented by the set F_i. Each flow in the use-case i, $flow_{i,j}, \forall j \in 1 \cdots |F_i|$, is associated with a bandwidth, $bw_{i,j}$ and a latency constraint, $lat_{i,j}$.*

The *bandwidth* of the flow represents the maximum rate of traffic communicated in the flow and the *latency* of the flow represents the maximum delay by which a packet of the flow should reach the destination.

The mapping algorithm for multiple use-cases is presented in Algorithm 2. In the first step, a NoC topology is generated. The size of the topology is varied in the outer-loop until a valid mapping is obtained in the subsequent



(a) Use-Case 1 (b) Use-Case 2 (c) Path for use-case 1 (d) Path for use-case 2

Figure 5. (a), (b): Example use-cases with traffic flows annotated with bandwidth values (in MB/s). (c), (d): The paths chosen for the two use-cases for a flow between cores C3 and C4.

steps. When we vary the size, we assume that the topology structure is a mesh, although the mapping design methodology is applicable to any NoC topology. Initially, all the cores of the SoC are unmapped. In the second step of the algorithm, the traffic flows are sorted in a non-increasing order of their bandwidth values for all the use-cases in the design. Then the flow with the maximum bandwidth value is chosen across all the use-cases. The intuition behind choosing the flow that has the largest bandwidth value first is that it reduces bandwidth fragmentation and larger flows get to use shorter paths, which is desirable as it leads to lower power consumption [15]. While choosing a flow, we prefer to choose a flow from the already mapped nodes before other flows, as it further helps in satisfying the bandwidth constraints.

Each use-case maintains separate data structures that represent the available bandwidth and TDMA slots in the NoC for that use-case. Once the maximum available flow across all the use-cases is chosen, the source and destination cores of the flow, if they are not already mapped, are mapped onto the NoC, and all the use-cases use the same mapping of the cores onto the NoC. When performing the mapping of these cores, the path with the least cost (path cost is a combination of hop delay and residual bandwidth/slots [20]) satisfying the constraints of the flow is chosen and the resources (bandwidth, slots) are reserved for the flow for that use-case.

Algorithm 2 Unified Mapping and Path Selection

1. Generate a NoC topology with one switch.
2. Sort the flows $f_{i,j}$, $\forall i \in 1 \cdots |U|$, $j \in 1 \cdots |F_i|$, in non-increasing order of the bandwidth values.
3. Choose the flow in order of the bandwidth value, preferring flows that have source/destination vertices already mapped. Let $f_{m,n}$ be the flow chosen.
4. Choose a least cost path that satisfies the constraints for the flow $f_{m,n}$ in the use-case m. If the source, destination of the flow is unmapped, map them onto the NIs on the ends of the chosen path. Reserve bandwidth $bw_{m,n}$ and TDMA slots for the flow on the path.
5. For all other use-cases i, $\forall i \in 1 \cdots |U|$, $i \notin m$, choose the flow f, that has the same source and destination vertices as $f_{m,n}$, if such a flow exists.
6. Choose a least cost path in each use-case that satisfies the constraints and reserve resources. For use-cases in same group, choose path for that use-case in the group that has the maximum bandwidth value for the flow and reserve resources across the path in each use-case.
7. Remove mapped flows and repeat steps 3-6 until all flows are mapped.
8. If a valid mapping is not possible, increase the topology size and go to step 1.

For all the other use-cases (other than the one whose flow was chosen and mapped now), the flows that have the same source-destination nodes as the mapped flow are chosen and the paths are selected and resources are reserved.

Example 1 *Let us consider a small example of the procedure for 2 use-cases shown in Figures 5(a) and 5(b). The largest flow across the 2 use-cases is the flow between the cores C3 and C4 in use-case 1. A mapping of the cores C3, C4 onto the NoC topology, along with unified path selection and TDMA slot table reservation for the first use-case is performed (Figure 5(c)). The flow between C3 and C4 in the other use-case is selected next and a path for the flow is found in the NoC. Note that the other use-case uses the same mapping of the cores onto the topology as the first use-case, but can use a different path if NoC re-configuration is possible when the two use-cases switch. The residual capacity and time slots on the NoC links are updated separately for the two use-cases. The process is repeated for all the remaining flows in the use-cases.*

During the path selection step, we take in to account the groupings obtained from Algorithm 1, where the use-cases in the same group have the same path, slot-table reservations. In this scenario, the path and slot reservation are chosen for the flow that has the maximum bandwidth value across the different use-cases in the group and the same path, slot-table reservation is utilized in all the use-cases of the group.

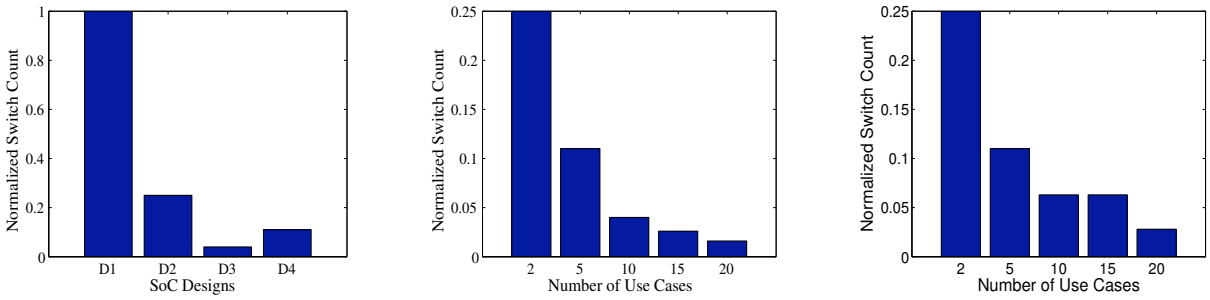
The above procedure is repeated until all the flows in all the use-cases are mapped. Note that once the initial mapping step is performed, the solution space can be explored further by considering swapping of vertices using simulated annealing or tabu search, as performed in [19]. We refer the interested reader to [20] for several optimizations carried out (during path selection, slot-table allocation), deadlock free path selection mechanisms, mathematical details of the objective function modeling, which are applied to a single use-case scenario. Such features are maintained in this multi use-case algorithm as well.

The key difference between this work and the earlier approach to multi-use case mapping presented in [25] (which is also based on [20]), is that instead of using a synthetic worst-case use-case that captures the design constraints of all the use-cases, here we maintain separate data-structures for the different use-cases. That is, during the mapping process we simultaneously consider the bandwidth, latency constraints of all the use-cases and when a flow is mapped, we update the data structures of the individual use-cases. As the worst-case use-case is over-specified, as it has to account for the worst constraints for every flow, such an approach is not scalable, while the method presented here scales much better with the number and variety of use-cases. The comparisons of the methods is presented in the next section.

6 Simulation Results

6.1 Experimental Benchmarks

To validate the performance of the multiple use-case mapping methodology, we perform experiments on existing SoC designs and synthetic benchmarks. We consider four simplified versions of real SoC designs: a set-top box



(a) SoC Designs (b) Spread (*Sp*) benchmarks (c) Bottleneck (*Bot*) benchmarks

Figure 6. The number of switches used for the current method normalized with respect to the the *WC* method

SoC with 4-use-cases [11] (*D1*), set-top box SoC with 20 use-cases (*D2*), a video processing SoC used in TVs with 8-use-cases (*D3*), and video processing SoC with 20-use-cases (*D4*). The designs *D2* and *D4* are based on scaled versions of the designs *D1* and *D3* for supporting more use-cases. Each use-case has a large number of (50 to 150) communicating pairs of components. The set-top box SoC and the TV processor have different functionalities and communication patterns. The set-top box design uses an external memory for storing and retrieving data and the amount of data communicated to the memory is very large when compared to the rest of the design. The video processor design uses a streaming architecture with local memories on the chip, thereby distributing the communication load across several components. We apply our design method to these SoCs with different architectures to validate the generality of the method.

We also generated synthetic benchmarks for testing the method with more number and variety of use-cases. The benchmarks are structured to follow the application patterns of real SoCs. We identify two classes of such benchmarks: (i) Spread communication benchmarks (*Sp*), where each core communicates to few other cores. These benchmarks represent designs such as the TV processor that has many small local memories with communication spread evenly in the design. (ii) Bottleneck communication benchmarks (*Bot*), where there are one or more bottleneck vertices to which most of the communication takes place. These benchmarks characterize designs using shared memory/external devices such as the set-top box example. We vary the bandwidth and latency constraints across the different traffic flows of the use-cases. Most of the video processing architectures have traffic flows that have bandwidth/latency values that fall in to few (around 3-4) clusters. As an example, the HD video streams have traffic flows with bandwidth requirements of few hundred MB/s, the SD video streams have few MB/S bandwidth needs, the audio streams have low bandwidth needs and the control streams have low bandwidth needs, but are latency critical. We capture such effects in the synthetic benchmarks generated, with the traffic parameters taking a cluster of values, with small deviations in the values within each cluster.

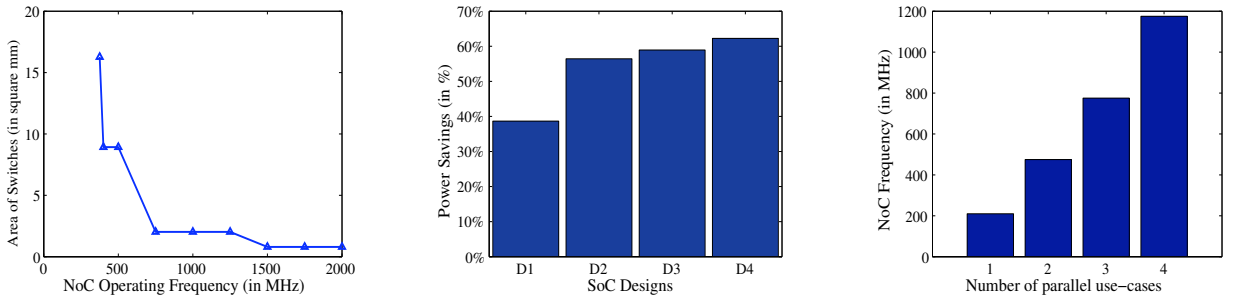
6.2 Effect of Mapping for SoC Benchmarks

In order to compare the quality of mappings produced by the design approach presented in this paper with the worst-case design method (*WC* method) presented in [25], we fix the operating frequency and link sizes of the NoC to be the same (500 MHz, 32 bits) for the methods. We apply the design methods and find the smallest size network that satisfies the constraints of the use-cases. We fix the number of cores to be same (equal to 20 with 60-100 connections between cores) for all the synthetic benchmarks and vary the number of use-cases across the benchmarks (from 2 to 40 use-cases) to evaluate the quality of the mappings. In Figure 6, the number of switches used in the mesh NoC for the current design methodology normalized with respect to the number of switches used in the *WC* method for the various benchmarks is presented. For the designs *D1*, *D2* and for the synthetic benchmarks with small number of use-cases, the *WC* method performs reasonably when compared to the method presented in this work. However, as the number of use-cases increase, the *WC* method starts to perform poorly, as the worst-case use-case becomes heavily over-specified and the resulting NoC design becomes big. The method presented here, on the other hand, performs well even for large number of use-cases and is scalable. As an example, for the *D3* design, the current methodology produced a successful mapping of the application onto a 2×2 mesh, while the *WC* method required a 11×11 mesh for the design. For the synthetic benchmarks (both *Sp* and *Bot*) with 40 use-cases, the current methodology resulted in a 2×2 mesh, while the *WC* method failed to produce a valid mapping even onto a 20×20 mesh topology (thus they not plotted in Figures 6(b) and 6(c)). Compared to the *Bot* benchmarks, for the *Sp* benchmarks the current method performs much better than the *WC*. This is attributed to the fact that the *Sp* benchmarks have more variations in the communication patterns across the different use-cases and the *WC* method is unable to adapt to such variations, while the current method does. For all the benchmarks, both the methods produced the results in less than few minutes when run on a Linux workstation.

6.3 Frequency-Area Tradeoffs

We can perform area-frequency trade-offs using the method presented in this work. When the NoC frequency is higher, the bandwidth and resources available across the NoC is higher and a smaller network can satisfy the constraints of the design. On the other hand, higher frequency of operation implies a higher power consumption in the network. In Figure 7(a), we present the Pareto curve for the area-frequency trade-off for the *D1* design. The area of the switches is obtained from layouts with back-annotated worst-case timing in 0.13 μm technology. At low operating frequencies ($\leq$ 350 MHz), the area of the NoC (which is taken to be the sum of the area of all the switches[1]) is large as more number of switches are needed to satisfy the design

[1]Here we assume that the NI area is taken to be part of the core area.



(a) Area-Frequency Trade-off (b) Effect of DVS/DFS (c) Frequency effects of parallel use-cases

Figure 7. (a) Area-Frequency trade-offs (b) The power savings achieved using DVS/DFS, (c) The impact of running use-cases in parallel

constraints. At very high-frequencies ($\geq$ 1.5GHz), the area of the NoC is very small. The optimum design point can be chosen based on the objectives of the designer from such a curve.

6.4 Dynamic Configuration

The switching time between most use-cases in a SoC is of the order of few milli-seconds. When the use-cases are expected to run for a long time, the frequency of operation of the NoC can be varied during this switching time to match the communication characteristics of the use-cases, thereby resulting in large power savings for the system. When the different use-cases require different NoC frequencies, the voltage of the NoC can also be dynamically changed to match the requirements of the use-cases. We use a conservative model for voltage scaling, where we assume that the square of the voltage scales linearly with the frequency [24]. The dynamic voltage and frequency scaling technique (DVS/DFS) results in an average of 54% reduction in power consumption for the different SoC designs when compared to the design where no DVS/DFS scheme is used (Figure 7(b)).

6.5 Parallel Use-Cases

As the number of use-cases that can run in parallel increases, the NoC size or frequency also increase. Our methodology can be applied by the designer to quickly perform trade-offs involving the number of use-cases that run in parallel with the size/frequency required for the NoC to support the parallel use-cases. As an example for a 20-core, 10 use-case *Sp* benchmark, the required NoC frequency as the number of use-cases run in parallel is varied is presented in Figure 7(c). Such a plot helps the designer in evaluating the trade-offs involved in the NoC for supporting multiple parallel use-cases.

7 Conclusions

The number of applications or use-cases integrated in a single SoC chip increases rapidly with each SoC platform. Designing an efficient NoC that supports the communication constraints of all the use-cases is a non-trivial problem. In this work, we presented mapping and NoC configuration methods for designing such high-end SoCs that support multiple use-cases and compound modes of operation, where multiple use-cases run in parallel. When the switching time between use-cases is large, the paths and TDMA slot tables of the NoC can be re-configured to match the use-case characteristics. In our methodology, we identify the use-case switchings where the NoC can be re-configured and consider this during the mapping process. We also explore the effect of dynamic voltage and frequency scaling (DVS/DFS) techniques for reducing the power consumption of the network across the different use-cases. We validate our design methodology on set-top box and TV SoC designs and on several synthetic benchmarks. The methodology is efficient and scalable to a large number of use-cases with varying communication patterns. In future, we plan to consider physical layout details to apply re-configuration of mappings across use-cases.

References

[1] L. Benini and G.De Micheli, "Networks on Chips: A New SoC Paradigm", IEEE Computers, pp. 70-78, Jan. 2002.
[2] W. J. Dally, B. Towles, "Route packets, not wires: on-chip interconnection networks", Proc. DAC 2001.
[3] D.Wingard,"MicroNetwork-Based Integration for SoCs", Design Automation Conference DAC 2001, pp. 673-677, Jun 2001.
[4] M. Sgroi et al. , "Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design", Proc. DAC 2001.
[5] F.Karim et al., "On-chip communication architecture for OC-768 network processors", Proc. DAC, pp. 678-678, June 2001.
[6] S.Kumar et al., "A network on chip architecture and design methodology", ISVLSI 2002, pp.105-112, Apr 2002.
[7] P.Guerrier, A.Greiner,"A generic architecture for on-chip packet switched interconnections", DATE 2000, pp. 250-256, March 2000.
[8] M. Dall'Osso et. al, "xpipes: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs", pp. 536-539, ICCD 2003.
[9] E. Rijpkema et al., "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip", DATE 2003.
[10] Philips, Nexperia PNX8550 Home Entertainment Engine, Dec. 2003.
[11] S. Dutta et al., "Viper: A multiprocessor SOC for advanced set-top box and digital TV systems", IEEE Design and Test of Computers, pages 21-31, Sept-Oct 2001.
[12] K. Goossens et al., "Interconnect and Memory Organization in SOCs for advanced Set-Top Boxes and TV — Evolution, Analysis, and Trends", Interconnect-Centric Design for Advanced SoC and NoC, Kluwer, April, 2004.
[13] J. Hu, R. Marculescu, "Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints", Proc. ASP-DAC 2003.
[14] J. Hu, R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures", Proc. DATE 2003.
[15] S. Murali, G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures", Proc. DATE 2004.
[16] S. Murali, G. De Micheli, "SUNMAP: a tool for automatic topology selection and generation for NoCs", Proc. DAC 2004.
[17] A. Pinto et al., "Efficient Synthesis of Networks On-Chip", Proc. ICCD, 2003.
[18] A. Pinto et al., "Constraint-Driven Communication Synthesis", Proc. DAC 2002.
[19] S. Murali et al., "Mapping and Physical Planning of Networks-on-Chip with Quality-of-Service Guarantees", Proc. ASPDAC 2005.
[20] A. Hansson et al., "A unified approach to constrained mapping and routing on network-on-chip architectures", pp. 75-80, Proc. ISSS 2005.
[21] A.Jantsch, H.Tenhunen, "Networks on Chip", Kluwer Academic Publishers, 2003.
[22] K. Goossens et al., "A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification", pp. 1182-1187, DATE 2005.
[23] D. Bertozzi et al., "NoC Synthesis Flow for Customized Domain Specific Multi-Processor Systems-on-Chip", IEEE Transactions on Parallel and Distributed Systems, Feb 2005.
[24] J. Rabaey et al., "Digital Integrated Circuits", Prentice Hall, 2002.
[25] S. Murali et al., " Mapping and Configuration Methods for Multi-Use-Case Networks on Chips", to appear in ASPDAC, 2006.