Chapter 2
Designing Crossbar Based Systems

Over the last decade, the communication architecture of SoCs has evolved from single shared bus systems to multi-bus systems. Today, state-of-the-art bus based systems, such as the AMBA AXI [2] or the STBUS platform [3] supports the instantiation of crossbar matrices, where multiple buses operate in parallel, providing a high bandwidth communication infrastructure. While methodologies that target the design of NoCs are required in the long run, providing design support for the state-of-the-art crossbar based bus designs pose an immediate and pressing problem.

A crossbar matrix can be viewed as an evolutionary NoC architecture, where a single switch is used for the communication traffic flows. As the design process for building a general NoC is more complex in nature, synthesis of crossbar-based communication architectures is an ideal starting point for illustration of the design methods.

Despite some similarities, there is one important difference between the design of a crossbar matrix and a general NoC architecture. As the crossbar matrix design is simpler, exact algorithms can be utilized to build the system, thereby leading to fully optimum solutions. Even in cases where completely optimum solutions cannot be obtained, a large portion of the design space can be explored. Thus, we can design the crossbar system to handle more efficiently the local variations in traffic rates and burstiness in traffic flows, when compared to a general NoC system.

Even in complex NoCs, the communication architecture will be hierarchical in nature, with local cores communicating through crossbars and the global communication taking place through a scalable network. Thus, it is important to have efficient methods to design such crossbar systems.

In this chapter, we present the design of state-of-the-art crossbar based bus systems. We present methods to automatically design the *most power efficient crossbar* configuration for a MPSoC, satisfying the performance characteristics of the applications [52]. The communication architecture for the design should closely match the application traffic characteristics and performance requirements.

As an example, let us consider an image-processing MPSoC (detailed explanation of the MPSoC and experimental set-up is presented later in Section 2.5) with three different communication architectures used to connect the cores: a shared bus (all the cores are connected to a single bus), a full crossbar (each core is connected to a separate bus), and a partial crossbar (some of the cores share a bus). In Table 2.1, the average and maximum latencies incurred for a transaction (transfer of a single data word), obtained from SystemC simulation of the design using the different communication architectures are presented. The sizes of the crossbars (in terms of number of components used) normalized with respect to the size of the shared bus are also presented in the table. As seen from the table, as expected, both the average

S. Murali, *Designing Reliable and Efficient Networks on Chips,*
Lecture Notes in Electrical Engineering 34,

Table 2.1 Crossbar performance and cost for an example image-processing MPSoC

Type	Average latency (cycles)	Maximum latency (cycles)	Size ratio
Shared bus	35	51	1
Full crossbar	6	9	11
Partial crossbar	10	20	4

and the maximum transaction latencies are much higher for a single shared bus than the partial or full crossbars. However, it is interesting to note that an optimal partial crossbar gives almost the same performance as a full crossbar, even though it uses fewer resources than a full crossbar.

The proposed design methodology is based on actual functional traffic analysis of the application, and the generated crossbar configuration is validated by cycle-accurate SystemC simulation of the application using that crossbar. Most previous works on bus generation and NoC topology generation (which are somewhat similar to crossbar generation) are either based on average communication traffic flow between the various cores or based on statistical traffic generating functions [43–51]. While the former approaches fail to capture local variations in traffic patterns (as the average bandwidth of communication is a single metric that is calculated based on the entire simulation time), the latter approaches are only based on approximations to the functional traffic.

The proposed design methodology differs from existing approaches [43–50] in the fact that it is based on the analysis of simulated traffic patterns in windows. The entire simulation period is divided into a number of fixed-sized windows. The crossbar is designed such that, within each window, the application communication requirements (such as the bandwidth requirements) are met. Moreover, the overlap among traffic streams mapped onto the same resource is minimized, thereby reducing the latency for data transfer. The criticality and real-time requirements of streams are also considered and the overlapping critical streams are mapped onto different crossbar resources.

The methodology spans an entire design space spectrum with the analysis based on average communication traffic (as done in many previous works [43–50]) and on-peak bandwidth (as done in [51]) being the two extreme design points. Thus, the methodology also applies to cases where application traces are not available and only rough estimates of the traffic flows between the various cores are known. The design point in the spectrum is varied by controlling the window size used for the traffic analysis and design.

We also integrate the setting up of several communication architecture parameters (such as the frequency of operation) with the crossbar synthesis phase. The wiring complexity of the interconnect should also be considered during the communication architecture synthesis procedure. For this, the floorplan of the design is performed, where the accurate physical locations of the cores and the crossbar matrix are determined. From the resulting floorplan, the wire-lengths in the design are

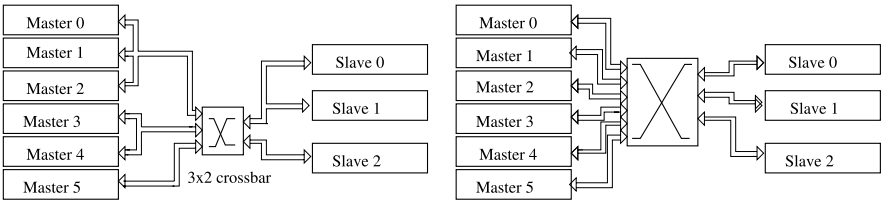
obtained. Based on the length of the wires and the operating frequency of the crossbar (which is automatically tuned by the synthesis procedure), any timing violations on the wires are obtained early in the design cycle. Thus, the crossbar architecture generated by the procedure is also validated for timing correctness, which is a key step to bridge the gap between the higher level architectural models and the actual physical design models of the crossbar architecture. From the wire-length estimates, accurate estimates of the power consumption of the interconnect wires are also obtained. The crossbar matrix power consumption values are based on the synthesis of the RTL models of the design, obtained using industry standard tools. From the wire and crossbar matrix power consumption, the total communication architecture power consumption is obtained, which is used to guide the synthesis procedure to obtain the most power efficient crossbar architecture.

We present experiments on several different MPSoC designs that show large reduction in power consumption of the communication architecture (45.3% on average) and total wire-length of the crossbar buses (38.0% on average) when compared to the traditional full crossbar based design approaches. Compared to the existing design methods, the proposed methodology results in crossbar platforms that lead to large reduction in transaction latencies (up to 7×). The experiments also show that the proposed approach is highly scalable to a large number of cores and to a large number of simulation windows in the design.

2.1 Problem Motivation and Application Traffic Analysis

2.1.1 Problem Motivation

There are three possible ways in which a crossbar can be instantiated: as a shared bus, a partial crossbar, or a full crossbar. The partial and full crossbars are actually composed of many buses to which the processor/memory cores are connected. Examples of partial and full crossbars are presented in Figure 2.1. In the partial crossbar architecture, some of the cores (such as the Master 0 and Master 1) share the same bus, while in the full crossbar, each core is connected to a separate bus. The objective of the crossbar synthesis procedure is to obtain an efficient clustering



(a) Partial crossbar (b) Full crossbar

Fig. 2.1 STbus crossbars

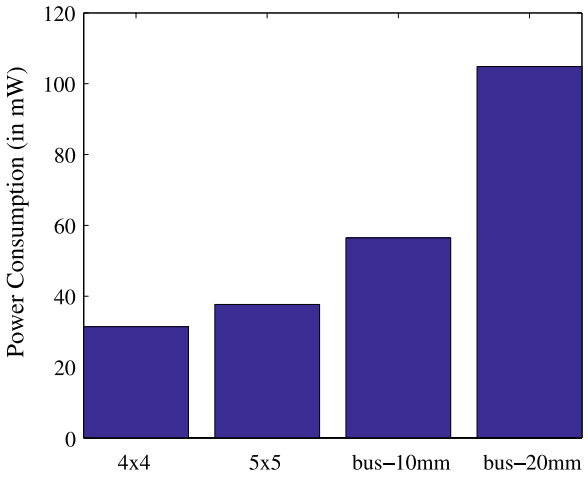
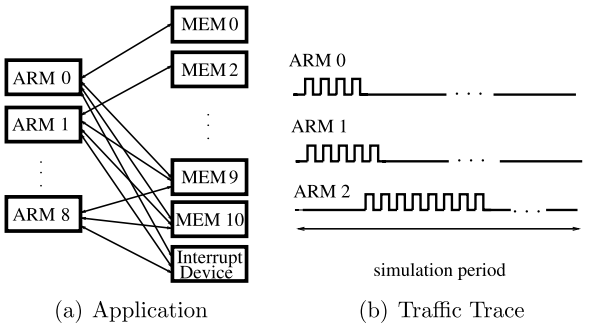


Fig. 2.2 Power consumption of switch matrix and wires

of the master and slave cores onto the crossbar buses, such that a communication architecture with low power consumption is obtained.

When choosing the most power efficient crossbar configuration, it is also important to account for the wiring complexity of the different configurations. As an example, the power consumption of the crossbar components (switch matrix and arbiters) for two different configurations and the power consumption of the wires for two different total wire-lengths (assuming a design with 30 cores and data width of 32-bits for the crossbar buses) are presented in Figure 2.2. For most MPSoC designs, the total length of the wires of the crossbar buses is of the order of few tens of millimeters. For the power consumption values presented in the figure, a 130 nm process technology is used, with an operating frequency of 500 MHz and an operating voltage of 1.2 V. The methods and assumptions used for estimating the power consumption of the crossbar matrices and wires are presented in detail in the experimental section. From the figure, we can infer that the wire power consumption is a significant fraction of the total communication architecture power consumption for crossbar based systems. Thus, it is important to consider the length of wires during the synthesis process, as the design point can be far from the optimum design point if such information is not accounted for. In order to have accurate wire-length estimates, we need to have accurate floorplan information of the design.

Another point worth noting is that in many crossbar architectures, the underlying protocol may not support pipelining of the buses (as an example, the Type 1 protocol of STbus [3]). In this case, the frequency of operation of the communication architecture is limited by the length of the longest bus in the design. For a chosen frequency point, it is then important to evaluate whether the length of the wires are lower than the threshold limit, so that they can be traversed in one clock cycle. We would also require the accurate floorplan and wire-length estimates to apply such feasibility checks.



(a) Application (b) Traffic Trace

Fig. 2.3 Application traffic analysis

2.1.2 Application Traffic Analysis

In this subsection, we explore the traffic characteristics of applications to model the performance constraints to be satisfied by the crossbar designed for the system. As an example, consider the 21-core image processing application, shown in Figure 2.3(a). In this example, there are 9 ARM cores, 11 on-chip memories, with some of the memories used for interprocessor communication and an interrupt device. The ARM cores act as masters and the memory cores act as slaves. The ARM cores run a set of image processing benchmarks that involve accesses to different memories. A cycle-accurate simulation of the system with a full crossbar design was performed, using the STbus crossbar architecture. A small trace of the traffic to three of the cores is shown in Figure 2.3(b).

Even though the aggregate traffic (measured over the entire simulation period) to the three cores is lower than that can be supported by a single bus, using a single bus to connect all three cores will lead to high average and peak latency due to overlap in traffic patterns during some regions of the simulation. Another related point is that if overlaps are not considered, connecting ARM 0 and ARM 1 on to the same bus is better than connecting ARM 0 and ARM 2 onto the same bus, as the former results in lower bandwidth needs. However, the latter solution will result in better performance (reduced transaction latency) while still satisfying the bandwidth needs. Note that using peak bandwidth instead of the average bandwidth will solve this problem, but lead to an over-design of the crossbar (in terms of number of buses needed or their frequency of operation). The design methodology needs to consider overlap among the various traffic streams into account and should consider local variations in traffic rates. Also, some of the traffic streams can be critical and to facilitate providing real-time guarantees; real-time traffic streams that overlap in time should not be mapped onto the same crossbar bus.

2.2 Design Methodology

The design flow for the crossbar design is shown in Figure 2.4, which consists of four distinct phases. In the first phase, the application is initially designed using a

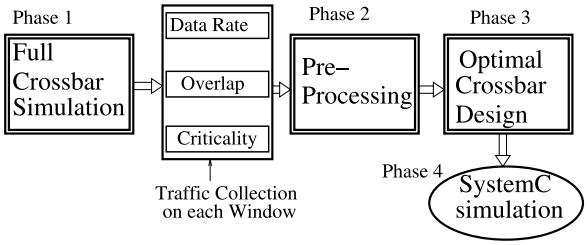


Fig. 2.4 Crossbar design methodology

full crossbar communication architecture and a SystemC simulation of the design is carried out. As the full crossbar architecture is nonblocking in nature (no contention between the cores if they are accessing different cores), it helps in modeling the application traffic requirements under ideal operating conditions. For the simulations, we use the *MPARM* simulation environment [57] that allows interconnection of ARM cores to several interconnection platforms (such as AMBA, STbus, ...) and to perform cycle accurate simulations for a variety of benchmark applications.

To effectively capture local variations in traffic patterns and to perform overlap calculations, we define a window-based traffic analysis. The entire simulation period is divided into a number of windows and the traffic characteristics to the various cores in each window are obtained. The traffic characteristics recorded include: the amount of data sent and received by each core in every window, amount of pair-wise overlap between the traffic streams between the different cores in every window, the real-time requirements of traffic streams, etc. Without loss of generality, in the rest of this chapter, we assume that all the windows are of equal size, although the methodology also applies to windows with varying sizes. The size of the window is parameterizable and depends on the application characteristics and performance requirements.

After the data collection phase, a preprocessing phase is carried out in which the cores that have traffic flows with large overlaps in any window and need to be put on different buses are identified. In this phase, the overlapping critical streams that need to be on separate buses are also identified.

In the next phase, the optimal crossbar configuration for the application, satisfying the performance constraints is synthesized. To generate the optimal crossbar configuration, we use the traffic information collected in each window and check whether the bandwidth, overlap, and criticality constraints are satisfied in each window. In the final phase, the designed crossbar matrix is instantiated in the MPARM environment and SystemC simulations are carried out.

The details of the crossbar synthesis phase are presented in Figure 2.5. In the outer loop of the synthesis process, the communication architectural parameters (such as the frequency of operation and bus width) are varied in several user defined steps. The interesting range for the parameters are obtained from the user. For each architectural parameter point, the most power efficient crossbar configuration is synthesized. For synthesis, we present two approaches: one approach is based on solving the problem exactly using *Integer Linear Program* (*ILP*) formulation, which is applicable for small problem instances, and the other is a more scalable approach based on fast and efficient heuristics. In the next step of the synthesis phase,

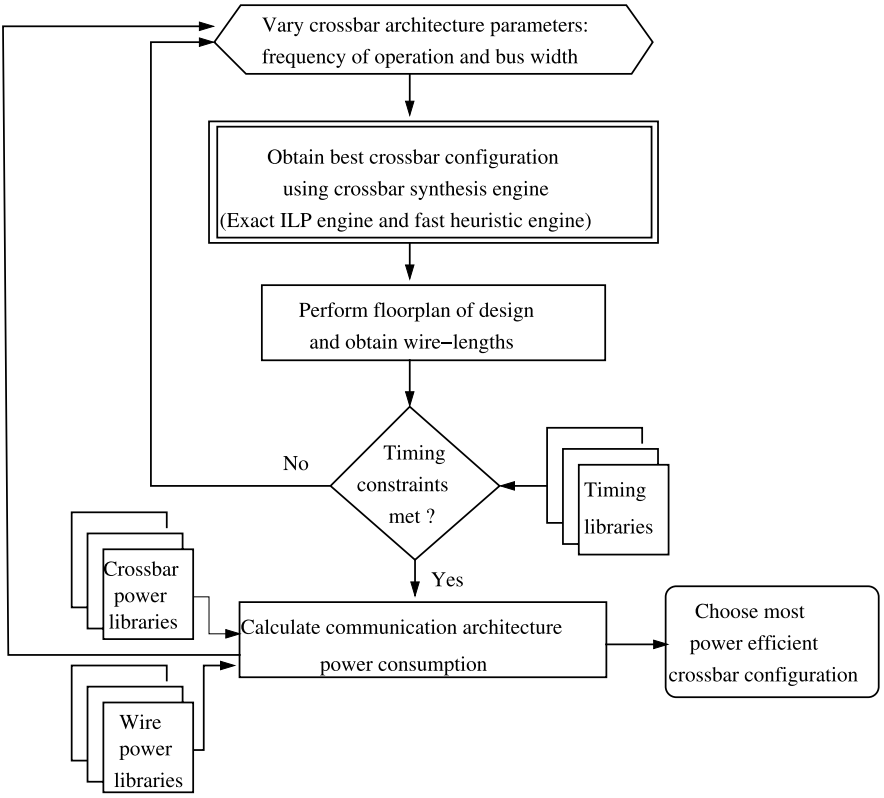


Fig. 2.5 The crossbar synthesis phase

floorplan of the synthesized design is performed. Floorplanning is the process of determining the exact 2D positions of the different cores and the switch matrix in the design. For obtaining the floorplan, we use Parquet [92], a fast and accurate floorplanner that minimizes the design area as well as the average wire-length. As the cores in the MPSoC are usually predesigned hardware blocks, we realistically assume that the size of the cores (either the width and height or the aspect ratio and area) are provided as an input to the synthesis process.

From the floorplan of the design, the length of the wires (based on the Manhattan distance), and hence the power consumption on the wires are obtained. In the next step, for the chosen frequency point, the wire-lengths are checked to see whether the maximum wire-length exceeds the length that the data can traverse in a single clock cycle. In the next step, from the switch matrix power consumption and the wire power consumption, the power consumption of the synthesized communication architecture is obtained. From the set of generated crossbar architectures for each architectural design point, the most power efficient architecture that satisfies the performance and timing constraints is chosen.

2.3 Exact Approach to Crossbar Synthesis

In this section, we formulate the mathematical models of the crossbar design problem and present the exact ILP formulation to synthesize the most efficient architecture for a chosen architectural parameter design point.

2.3.1 Problem Formulation

Definition 1 The set of all cores in the design is represented by the set T. The set of all windows used for traffic analysis is represented by the set W, with the bandwidth available (product of frequency of operation and bus width) in each window represented by WS. The set of buses used in the crossbar is represented by the set B.

Definition 2 The bandwidth requirement of each core t_i, $\forall i \in 1, \ldots, |T|$, in every window m, $\forall m \in 1, \ldots, |W|$, is represented by $comm_{i,m}$.[1] The amount of data overlap between every pair of cores (t_i, t_j) in each window m is represented by $wo_{i,j,m}$.

The overlap between every pair of cores t_i and t_j, over the entire simulation period is obtained by summing the overlap between them in all the windows and represented by the entries of the overlap matrix OM:

$$om_{i,j} = \sum_{m} wo_{i,j,m} \quad \forall i, j \tag{2.1}$$

In the preprocessing phase of the design flow (refer to Figure 2.4), those pair of cores that have overlap exceeding the threshold value (which is parameterizable) in any window are identified. By mapping the traffic flows of such cores onto separate buses, the maximum and average latency of data transmission can be reduced, and in some cases can also speed up the process of finding the optimal crossbar configuration. Also, in this preprocessing step, the real-time traffic streams that overlap with each other in any window are identified. Such cores with overlapping real-time streams should not be placed on the same bus, as real-time communication guarantee to the streams cannot be given in this case. Also, as noted earlier, most crossbar architectures do not allow masters and shared slaves of the design to be mapped onto the same bus. The set of all cores that cannot be on the same bus by the conflict matrix is defined by:

$$c_{i,j} = \begin{cases} 1, & \text{if } t_i \ \& \ t_j \text{ should be on different buses} \\ 0, & \text{otherwise} \end{cases} \quad \forall i, j \tag{2.2}$$

The performance constraints that need to be satisfied by the crossbar configuration in each window are modeled as constraints of an ILP.

[1] In the rest of this chapter, we follow the convention that variables i and j are defined for $1, \ldots, |T|$, variable k is defined for $1, \ldots, |B|$, and m for $1, \ldots, |W|$.

Definition 3 The set X represents the set of binding variables $x_{i,k}$, such that $x_{i,k}$ is one when core t_i is connected to the bus b_k and zero otherwise.

In the crossbar design, each core has to be connected to a single bus (while a single bus can connect multiple cores). This is implemented by the following constraint:

$$\sum_k x_{i,k} = 1 \quad \forall i \tag{2.3}$$

In every window used for traffic analysis, the individual buses of the crossbar have to support the traffic through them in that window. By evaluating the bandwidth constraints over a smaller sample space of a window (which is typically a few hundred or thousand cycles), instead of the entire simulation sample space (which can be millions of cycles), we are better able to track the local variations in the traffic characteristics.

This window-based bandwidth constraint is represented by the equation:

$$\sum_i comm_{i,m} \times x_{i,k} \leq WS \quad \forall k, m \tag{2.4}$$

Definition 4 The set SB represents the set of sharing variables $sb_{i,j,k}$, such that $sb_{i,j,k}$ is one when cores t_i and t_j share the same bus b_k and zero otherwise. The set S represents the set of sharing variables $s_{i,j}$, such that $s_{i,j}$ is one when cores t_i and t_j share any of the buses of the crossbar and zero otherwise.

The $sb_{i,j,k}$ can be computed as a product of $x_{i,k}$ and $x_{j,k}$. However, this results in nonlinear (quadratic) equality constraints. To break the quadratic equalities into linear inequalities, the following set of inequalities are used:

$$\begin{aligned} & sb_{i,j,k} \in \{0,1\} \\ & x_{i,k} + x_{j,k} - 1 \leq sb_{i,j,k} \\ & 0.5x_{i,k} + 0.5x_{j,k} \geq sb_{i,j,k} \quad \forall i, j, k \end{aligned} \tag{2.5}$$

and the $s_{i,j}$ are computed using the equation:

$$s_{i,j} = \sum_k sb_{i,j,k} \quad \forall i, j \tag{2.6}$$

The condition that certain cores are forbidden to be on the same bus, obtained from equation (2.2), is represented by:

$$c_{i,j} \times s_{i,j} = 0 \quad \forall i, j \tag{2.7}$$

The fact that all the integer variables introduced above take values of either 0 or 1 only, is represented by:

$$x_{i,k}, s_{i,j}, c_{i,j} \in \{0,1\} \quad \forall i, j, k \tag{2.8}$$

2.3.2 Exact Crossbar Synthesis Algorithm

The exact algorithm for the crossbar design has two major steps: the first is to find the best crossbar configuration that satisfies the performance constraints (that were presented in the above subsection) and the second step is to find the optimal binding of the cores to the chosen crossbar configuration.

In order to find the best crossbar configuration, we vary the number of buses in the design, from the maximum number (equal to the number of cores in the design, modeling a full crossbar) to one (modeling a single shared bus), in a binary search manner. For each configuration of bus count, we check whether a feasible solution that satisfies the constraints of the ILP (formed by the set of inequalities from equations (2.3) to (2.8)) exists. Once the minimum number of buses have been identified from applying the ILP, possibly multiple times, the buses used by the masters and slaves of the design are separated, thereby generating the optimal crossbar configuration.

Once the best crossbar configuration is obtained in the next step, the optimal binding of the cores onto buses of the crossbar is obtained. A binding of cores to the buses that minimizes the amount of overlap of traffic on each bus will result in lower average and peak latency for data transfer.

For this, the above ILP is solved with the objective of reducing the maximum overlap on each of the bus (the maximum overlap over all the buses is represented by the variable *maxov*), and satisfying the performance constraints as follows:

$$\begin{aligned} &\min \ maxov \\ &\text{s.t.} \quad \sum_i \sum_j om_{i,j} \times sb_{i,j,k} \leq maxov \quad \forall k \\ &\quad \text{and subject to equations (2.3) to (2.8)} \end{aligned} \tag{2.9}$$

By splitting the problem into two ILPs, the execution time of the algorithm is reduced, as solving ILP 1 for feasibility check is usually faster than solving the ILP 2 with objective function and additional constraints. The ILPs are solved using the CPLEX package [59].

2.4 Heuristic Approach to Crossbar Synthesis

As the exact ILP approach is not scalable to large problem instances, either when the number of cores in the design is large or when the number of simulation windows used for analysis is large, in this section we present fast and efficient heuristic approach for crossbar synthesis.

The problem of assigning cores to the minimum number of buses, subject to the performance constraints is a special instance of the general problem of *constrained bin-packing* [60]. There are several efficient heuristics that have been developed for

the bin-packing problem [60]. In this work, we use an approach that is based on the *first-fit* heuristic to bin-packing. We chose this heuristic for several reasons. When the performance constraints are removed, the heuristic procedure is theoretically guaranteed to provide solutions that are within two times the optimum solution that would be obtained by an exact algorithm [60]. Practically, we found that the solutions obtained by the heuristic are close to the optimum solution possible for experiments on several SoC benchmarks. Moreover, the heuristics are relatively simple to implement and have a very low run-time complexity, making the approach scalable to large designs and allowing the use of large number of simulation windows for analysis.

The heuristic algorithm for crossbar synthesis is presented in Algorithm 1. In the first step of the algorithm, the bandwidth available in each simulation window is calculated. In the next step, all the cores are initialized as unmapped, as they are yet to be mapped onto buses. Then the number of buses in the crossbar is initialized to zero (step 5). In steps 6 to 25, the assignment of the cores onto the buses of the crossbar is performed. The basic approach used is the following: We try to map as many cores as possible onto a single bus. While mapping the cores, from the set of all cores that satisfy the bandwidth and conflict constraints, we choose the one that minimizes the pair-wise traffic overlap with the cores that have been already mapped onto the current bus. When no more cores can be assigned to the current bus, either because the bandwidth of the bus in any of the simulation window has been saturated, or because of conflicts with the cores already mapped onto the bus, a new bus is instantiated. The process is repeated until all the cores in the design have been mapped onto a bus.

From the resulting number of buses, the buses onto which masters are attached and those onto which the slaves are attached are separated. From this, the efficient crossbar configuration for the design is obtained.

Example 1 Let us consider a small example with 5 cores, with 3 of them being masters and the rest being slaves. For illustrative purposes, let us assume that two simulation windows are used for analysis (although in real systems usually several thousand windows are used). The communication traffic rates for each of the cores (in MB/s) for the two simulation windows are presented in Table 2.2 and the amount of traffic overlap between the different cores over all the windows is presented in Table 2.3. Let us assume that the current frequency design point is 100 MHz and the bus width is 32 bits, which are automatically tuned by the crossbar synthesis procedure (as presented in Figure 2.5). In the first step of the heuristic algorithm, the bandwidth of the bus in each simulation window is calculated to be 400 MB/s (frequency $\times$ data-width). Initially, a single bus is instantiated and core_0 is chosen to be mapped onto the bus, as it has the maximum bandwidth requirements of the different cores, across all the simulation windows (see Figure 2.6(a)).

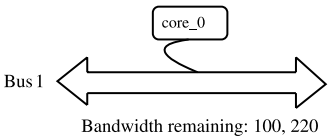
Then from the set of all cores, those cores that satisfy the bandwidth and conflict constraints are chosen. As cores that are masters and slaves are not allowed to be mapped onto the same bus (specified as part of the conflict constraints), the set of assignable cores to the bus are core_1 and core_2. From these two, core_2 is

Algorithm 1 Heuristic-synthesis(*frequency*, *buswidth*)

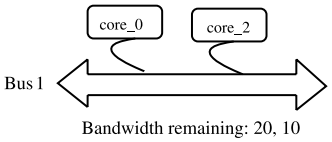
1: Bandwidth available in each window, $WS = frequency \times buswidth$
2: **for** $i = 1, \ldots, |T|$ **do**
3: mapped$(i) =$ false
4: **end for**
5: Initialize number of buses used, k to 0
6: **while** $\exists i \in 1, \ldots, |T|$, such that mapped$(i) =$ false **do**
7: Increment the bus count k by 1 and instantiate new bus. Initialize bandwidth available on bus on all windows: $BW(k, m) = WS, \forall m \in 1, \ldots, |W|$
8: Choose unmapped core $i, \forall i \in 1, \ldots, |T|$, with maximum bandwidth requirements on any window and map it onto bus k
9: Initialize the set *chosen_set* to ϕ
10: **for** $i = 1, \ldots, |T|$ **do**
11: **if** mapped$(i) =$ false and core i does not have conflicts with cores already mapped onto bus k **then**
12: $bw\_satisfied =$ true
13: **for** $m = 1, \ldots, |W|$ **do**
14: **if** $BW(k, m) < comm_{i,m}$ **then**
15: $bw\_satisfied =$ false
16: **end if**
17: **end for**
18: **if** $bw\_satisfied =$ true **then**
19: $chosen\_set = chosen\_set \cup i$
20: **end if**
21: **end if**
22: **end for**
23: Choose core $i, \forall i \in 1, \ldots, |chosen\_set|$, with minimum overlap with cores mapped onto bus k and map it to bus k. Update available bus bandwidth as: $BW(k, m) = BW(k, m) - comm_{i,m}, \forall m \in 1, \ldots, |W|$
24: Repeat steps 9–23 until *chosen_set* is empty
25: **end while**
26: Separate the buses onto which masters and slaves are mapped and generate the crossbar configuration

Table 2.2 Communication requirements of example system: (M-Master, S-Slave)

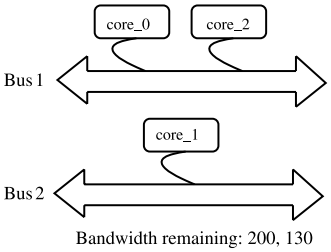
Name	Type	BW (win 1) MB/s	BW (win 2) MB/s
core_0	M	300	180
core_1	M	200	270
core_2	M	80	210
core_3	S	60	110
core_4	S	150	70



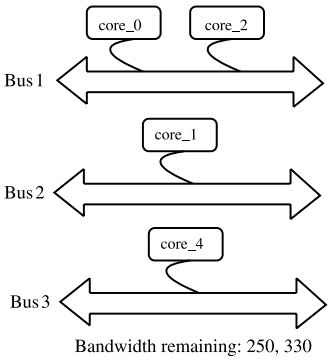
(a) Step 1. core 0 is chosen



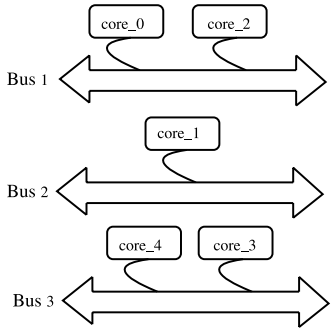
(b) Step 2. From emaining cores, core 2 is chosen.



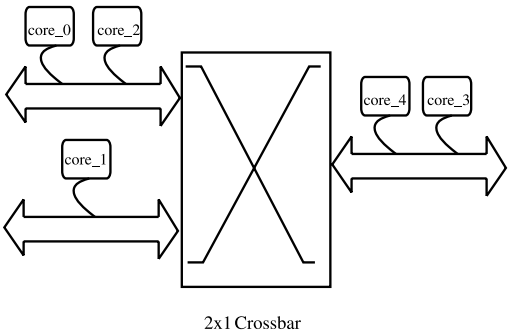
(c) Step 3. New bus is instantiated, core 1 mapped.



(d) Step 4. Another bus is instantiated.



(e) Step 5. The remaining core, core 4 is mapped.



(f) Step 6. The crossbar configuration, 2x1, is obtained from the instantiated buses.

Fig. 2.6 Example application of the heuristic algorithm

Table 2.3 Amount of traffic overlap between cores (in MB/s) of example system

	core_0	core_1	core_2	core_3	core_4
core_0	×	30	10	×	×
core_1	30	×	27	×	×
core_2	10	27	×	×	×
core_3	×	×	×	×	15
core_4	×	×	×	15	×

chosen, as it has minimum overlap with the cores already mapped onto the bus (i.e., with core_0) and assigned onto this bus (Figure 2.6(b)). When no more cores can be assigned to the current bus, a new bus is instantiated. The different steps of the procedure for the 5-core example are presented in Figures 2.6(a)–(f). At the end of the procedure, those buses that are used by the masters and those that are used by the slaves are separated, which gives the best crossbar configuration. In this example, we have 2 buses used by the masters and 1 used by the slaves, resulting in a 2×1 crossbar design, as shown in Figure 2.6(f).

2.5 Experiments and Case Studies

In this section, we present the experimental case studies performed to validate the proposed crossbar design methodology.

2.5.1 Experimental Platform and Power Models

For performing the SystemC simulations on MPSoC benchmarks, we use the MPARM simulation platform [57]. The platform is a representative of a large class of multiprocessor SoC platforms and consists of a configurable number of 32-bit ARM processors, memory cores, hardware devices or traffic generators, and a hardware interrupt unit. The platform allows the use of different interconnect architectures, such as the AMBA, STbus to interconnect the various hardware cores. It also supports a variety of MPSoC benchmarks that have been efficiently parallelized to run on the ARM cores.

For power consumption estimations of the switch matrix, we implemented several configurations of the AMBA multilayer crossbar, varying the number of input and output ports of the matrix. The different configurations were implemented using the AMBA DesignWare libraries obtained from Synopsys CoreAssembler tool [98]. The tool generates RTL code of the different configurations, which were then synthesized using Synopsys Design Compiler [98]. For synthesis, we utilize a 130 nm process technology, an operating voltage of 1.2 V and an operating frequency of 500 MHz. Based on the power consumption values obtained from the synthesis process, analytical models for the switch matrix power consumption are built using linear regression. During the crossbar design process, the power numbers from the analytical models are linearly scaled, based on the crossbar operating frequency (which is automatically tuned by the design process). We estimate the wiring capacitance and wire power consumption based on the models from [58]. The power consumption values of some of the crossbar components were presented earlier in Figure 2.2.

2.5.2 Application Benchmark Analysis

We apply the crossbar design methodology on several SoC designs implemented using the MPARM platform: *IMage Processing design 1* (*IMP1*-25 *cores*), *IMage Processing design-2* (*IMP2*-21 *cores*), *FFT based SoC* (*FFT*-29 *cores*), *Data Processing SoC* (*DP*-15 *cores*) *and SoC implementing a DES encryption system* (*DES*-19 *cores*). The traffic characteristics of the applications were scaled to project the traffic requirements of future MPSoCs, as presented in [31]. For traffic analysis, we use 1000 simulation windows for the different designs, with each simulation window accounting for few hundred simulation cycles.

The interesting range of operating frequencies and bus data-widths are obtained as inputs from the designer. Practically, the data-width of the bus is set-up based on the data-widths of the different processors in the design. In the SoC designs used here, all the processors have the same data-width (of 32-bits), and hence we feed this value as an input to the synthesis engine. The interesting range of operating frequencies are defined to be between 100 MHz to 500 MHz, with each frequency point being a multiple of 100 MHz. With this set-up, we apply the heuristic synthesis engine to design the crossbar architecture for the designs.

We first briefly analyze the quality of the crossbar design obtained for the IMP2 SoC design. The communication between the cores of the IMP2 design was presented earlier in Figure 2.3(a). The communication requirements of some of the cores for the first few simulation windows are presented in Table 2.4. In this benchmark, there are 9 ARM cores (ARM 0 to ARM 8), 11 memory cores (MEM 0 to MEM 10), and an interrupt device (INT). The ARM cores act as masters and the others are slave cores that respond to the requests of the masters. There is substantial temporal overlap between the traffic flows from the various ARM cores to the memories, as the ARM cores perform similar computations, and thus access their memories at almost the same time. The power consumption of the synthesized crossbar designs for the different frequency design points are plotted in Figure 2.7. As the maximum bandwidth requirements of most of the cores were above 800 MB/s, the minimum frequency design point that gives a feasible solution is 300 MHz (at 200 MHz, the available bus bandwidth of 800 MB/s cannot support the requirements of most cores). At lower operating frequencies (such as 300 MHz), a larger crossbar

Table 2.4 Traffic characteristics of IMP2

Core	Win. 1 (MB/s)	Win. 2 (MB/s)
ARM 0	810	210
ARM 1	740	234
MEM 0	790	150
MEM 1	730	220
MEM 9	180	50
MEM 10	180	50

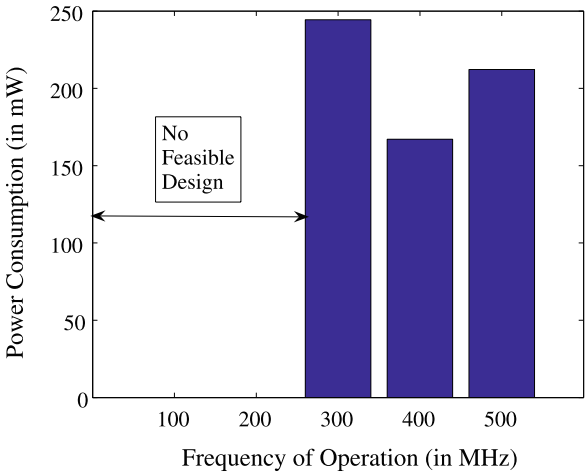


Fig. 2.7 Power consumption for different crossbar frequencies

configuration is required to satisfy the bandwidth constraints. A larger crossbar configuration usually also leads to an increased wiring complexity. These two factors coupled together results in larger power consumption for the communication architecture. At very high operating frequencies, the power consumption of the communication architecture is higher, as the power consumption increases linearly with the operating frequency of the system. For the IMP2 design, the crossbar architecture with lowest power consumption is obtained at 400 MHz.

The synthesized crossbar architecture (a 5×6 crossbar) for the IMP2 design is presented in Figure 2.8. In order to satisfy the window bandwidth constraints, only few of the cores can share a single bus, and thus each of the buses used in the crossbar have at most 2 cores attached to them. The bindings are such that the cores with highly overlapping streams are placed on different buses. As a result, the designed crossbar has acceptable performance (in terms of average and maximum latency constraints) with $1.9\times$ reduction in the number of buses used, when compared to a full crossbar. The floorplan of the IMP2 SoC with the designed crossbar, as obtained from the Parquet floorplanner is presented in Figure 2.9.

The size and power consumption of the synthesized crossbar architectures for the different SoC designs and for full crossbar configurations are reported in Table 2.5. The power consumption of both the switch matrix and the crossbar bus wires are reported in the table. The methodology results in a large reduction in the crossbar architecture power consumption (45.3% on average) when compared to the traditional full crossbar based systems. The synthesized crossbar configurations also lead to large reduction in the total length of the buses used in the design (38.0% on average, refer to Figure 2.11), as there are fewer buses in the design. Reducing wiring congestion is essential to have a faster physical design process and to achieve faster design closure.

The normalized average and maximum read/write transaction latencies (to read or write one data word) for the designs obtained using the methodology based on average traffic flows and using the proposed methodology (referred to as "slot" in the

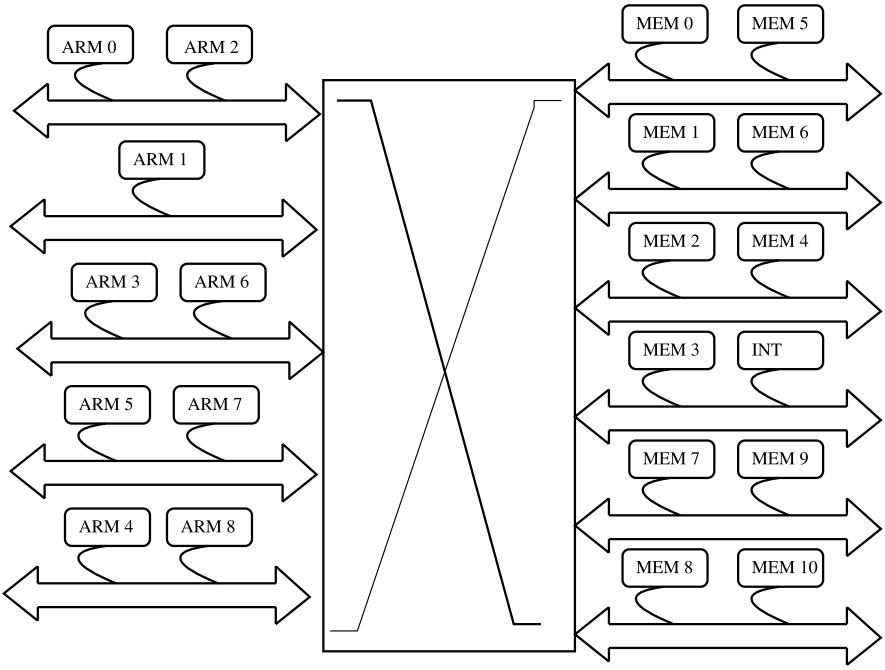


Fig. 2.8 Synthesized crossbar for the IMP2 SoC design

Fig. 2.9 Generated floorplan and buses for the IMP2 SoC design

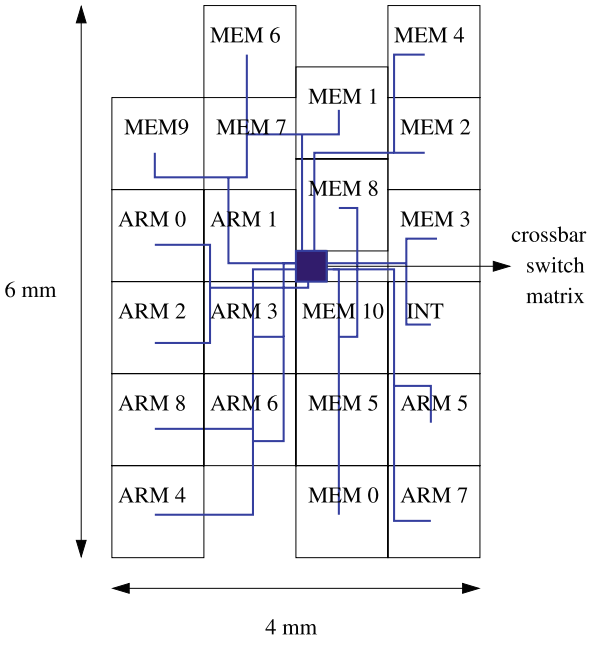
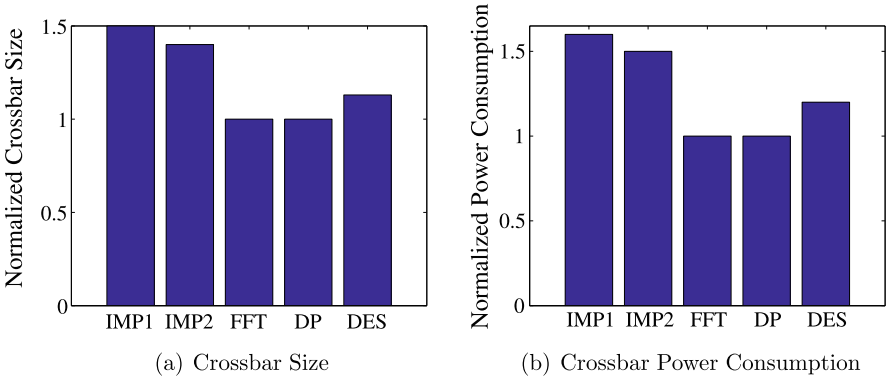


Table 2.5 Crossbar size and power consumption for SoC designs

Design	Full crossbar size	Synthesized crossbar size	Full crossbar power consumption (mW)			Synthesized crossbar power consumption (mW)		
			Matrix	Wire	Total	Matrix	Wire	Total
IMP1	11×14	6×7	156.7	228.0	384.7	60.2	146.1	206.3
IMP2	9×12	5×6	128.4	198.2	326.6	45.2	125.0	170.2
FFT	13×16	7×8	175.1	301.4	476.5	75.9	191.8	276.7
DP	6×9	3×5	38.7	51.3	90.0	12.1	36.9	49.0
DES	8×11	4×6	56.0	82.1	138.1	18.8	54.1	72.9



(a) Crossbar Size (b) Crossbar Power Consumption

Fig. 2.10 Comparisons of heuristic engine vs. exact engine

figures, signifying the use of the proposed slot or window based methodology), are presented in Figures 2.12(a) and (b). As seen from the figures, the latencies incurred by crossbar designs based on average traffic flows are $4\times$ to $7\times$ higher than the crossbars designed using the presented scheme. Also, the latencies incurred in the designs generated by our scheme are within acceptable bounds from the minimum possible latencies (of a full crossbar). Moreover, depending on the design objective, crossbar size-performance trade-offs can be explored in this approach by tuning the analysis parameters (such as the window size, overlap threshold, etc.), as explained in further subsections.

2.5.3 Comparisons of Heuristic Engine with the Exact Engine

In this subsection, we explore the quality of the solutions produced by the heuristic engine with respect to the exact ILP engine. As the exact engine takes several hours to compute solutions for designs with more than few hundred windows, we

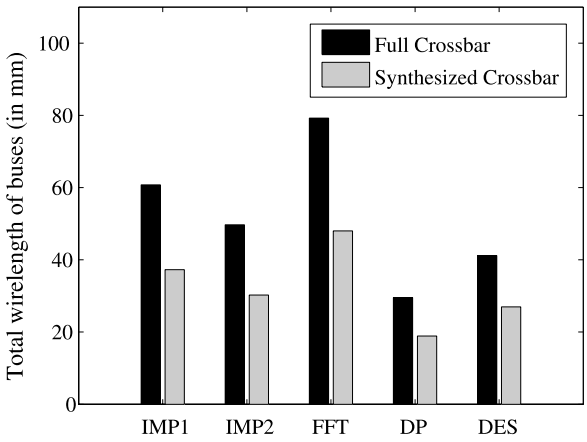


Fig. 2.11 Average wire-length of the crossbar buses for the designs

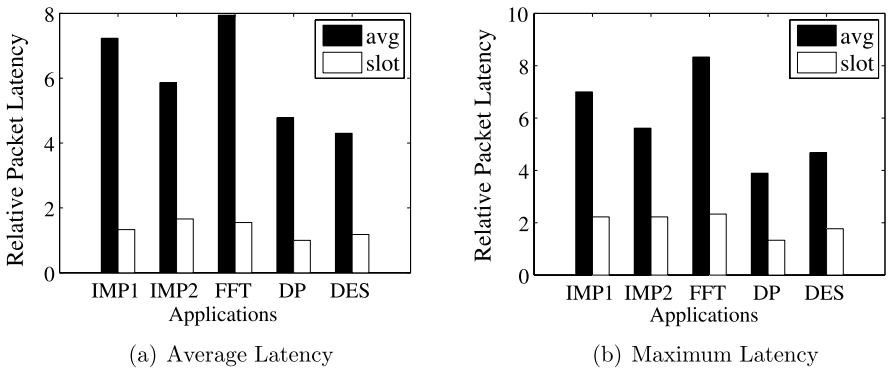


Fig. 2.12 Application relative latencies

reduced the number of windows to 100 for the designs and applied the two engines for the SoC designs. The size (total number of buses) of the crossbar synthesized by the heuristic engine normalized with respect to the size of the crossbar synthesized by the exact engine for the different designs is presented in Figure 2.10(a). The normalized power consumption of the synthesized crossbar designs for the different SoC designs is presented in Figure 2.10(b). Compared to the exact solutions, the solutions obtained by the heuristic engine incur only a modest increase in crossbar size (1.21× on average) and power consumption (1.26× on average).

The total run-time of the heuristic engine (including the time for performing floorplanning) for the biggest SoC design (the FFT SoC) for different number of window sizes is presented in Table 2.6. The experiments were performed on a Linux workstation, with 3.2 GHz processor, and 4 GB RAM. The run-time also includes the time to perform the sweep over the architectural parameters (frequency of operation and bus width) of the crossbar design. As seen from the table, the algorithms have a very low run-time complexity even for large designs and when large number

Table 2.6 Heuristic procedure run-time for FFT design

Number of windows	Run-time (in s)
1000	4.85
10000	5.46
50000	8.31
100000	11.91
500000	41.40

Table 2.7 Run-time for different number of cores

Number of cores	Run-time (in s)
29	41.40
40	67.22
50	96.73
60	130.03

of windows are used for analysis. On the other hand, the exact ILP procedure did not produce results in reasonable time for the design when more than few hundred windows were used for analysis. To show the scalability of the heuristic procedure with the number of cores in the design, we produced synthetic benchmarks based on the scaled versions of the FFT SoC. The execution times of the engine for the different benchmarks (with the number of windows set to 500,000) are presented in Table 2.7. From the table, we see that even for a very large design (60 cores with 500,000 windows used for analysis), the heuristic process completes in few minutes, thereby showing the scalability of the procedure.

2.5.4 Window Sizing

The size of the window used during the design process is an important parameter that determines the efficiency of the design methodology to capture the application performance parameters. A small window size results in much finer control of the application performance parameters and the resulting crossbars have lower latencies. However, a very small window size will lead to over-design of the network components. On the other hand, a large window size results in lesser control over the performance parameters of the application, but results in a more conservative design approach where higher transaction latencies can be tolerated.

To illustrate these effects, we applied the design methodology with different window sizes for a synthetic benchmark with 20 cores. Please note that we use a synthetic benchmark for this experiment (instead of the real SoC designs), so that we can vary the burst sizes (we refer to a burst as a stream of words generated by the

same core) in the application to study its impact on the crossbar synthesis process. The typical burst sizes for the benchmark is initially set to 100 cycles. When the window size is much smaller than the burst size, the size of the crossbar generated is very close to that of a full crossbar (refer Figure 2.13). When the window size is around few times that of the burst size (from 1–4 times), the synthesized crossbar has much smaller size (typically around 25%) and acceptable latencies (around 1.5×) of that of a full crossbar. For aggressive designs, the window size can be set closer to the burst size and for conservative designs (where larger transaction latencies can be tolerated), the window size can be set to few times the typical burst size. The acceptable window sizes for various burst sizes is presented in Figure 2.14. It can be seen from the plot that the window size varies almost linearly with the burst size, consolidating the above arguments.

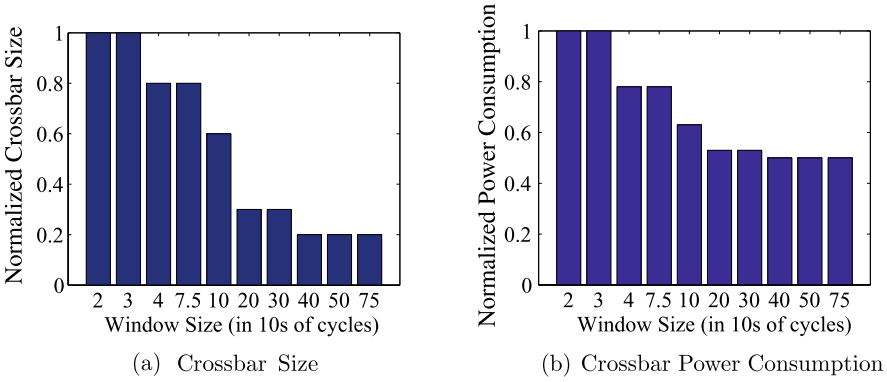
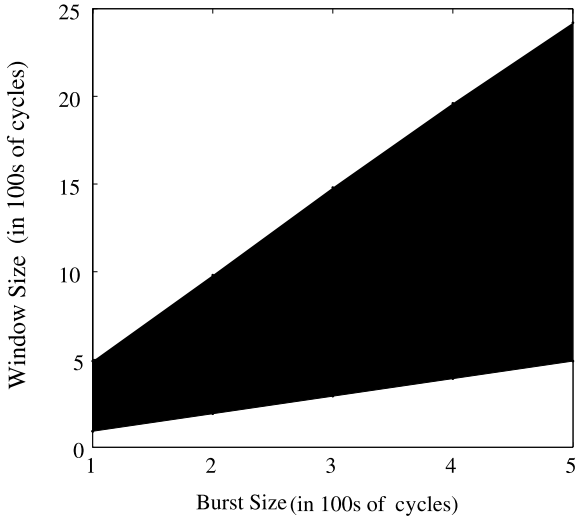


Fig. 2.13 Effect of window size on crossbar size and power consumption

Fig. 2.14 Burst vs. window size



2.5.5 Real-Time Streams & Effect of Binding

In each simulation window, the critical traffic streams that require real-time guarantees are recorded. During the preprocessing step of the design flow (refer to Figure 2.4), the real-time traffic streams that overlap with each other in any window are identified. In order to provide real-time guarantees to such streams, the cores with critical streams that have temporal overlap are placed onto separate buses of the crossbar. Experimental results on the benchmark applications show a very low transaction latency (almost equal to the latency of perfect communication using a full crossbar) for such streams. Please note that in order to provide hard real-time guarantees, the underlying crossbar architecture should also provide support for having priorities for the different traffic streams, so that the real-time streams are given higher priorities over other streams. In many crossbar architectures, such as the ST-bus, such support is provided in the crossbar architecture by utilizing priority based arbitration mechanisms.

After finding the best crossbar configuration, we do an optimal binding of the cores onto the buses of the crossbar, minimizing the total overlap on each bus. By minimizing the overlap on each bus, the transaction latencies reduce significantly. To illustrate this effect, we compare the crossbars designed using the proposed approach with two binding schemes: random binding of cores onto the buses, satisfying the design constraints (equations (2.3)–(2.8)) and optimal binding that minimizes overlap on each bus, satisfying the design constraints. The average latency incurred by the random binding scheme for the benchmark applications was on average 2.1× higher than that incurred by the optimal binding scheme.

2.5.6 Overlap Threshold Setting

By varying the two parameters: window size and overlap threshold, the crossbar can be designed such that the average and the maximum transaction latencies incurred in the design are acceptable. The effect of the overlap threshold parameter on the size and power consumption of the crossbar generated for the synthetic benchmark are presented in Figures 2.15(a) and (b). The crossbar size and power numbers are normalized with respect to the case when the overlap threshold is set to 0%, which leads to a full crossbar configuration (as no two cores can share a bus in this case). The plots end at 50% overlap between cores because, if the pair-wise overlap between two cores exceeds 50% of the window size (in any of the windows), then the window bandwidth constraints cannot be satisfied. So, the maximum value of the overlap parameter can be set at 50% of the window size. This will also speed-up the process of finding the best crossbar configuration, as such overlapping cores will be identified in the preprocessing phase (refer to Figure 2.4) and will be forbidden to be on the same bus of the crossbar. From experiments, we found that for aggressive designs (where there are tight requirements on the maximum latencies) the threshold can be set to around 10% and for conservative designs, the threshold can be set to 30%–40% of the window size.

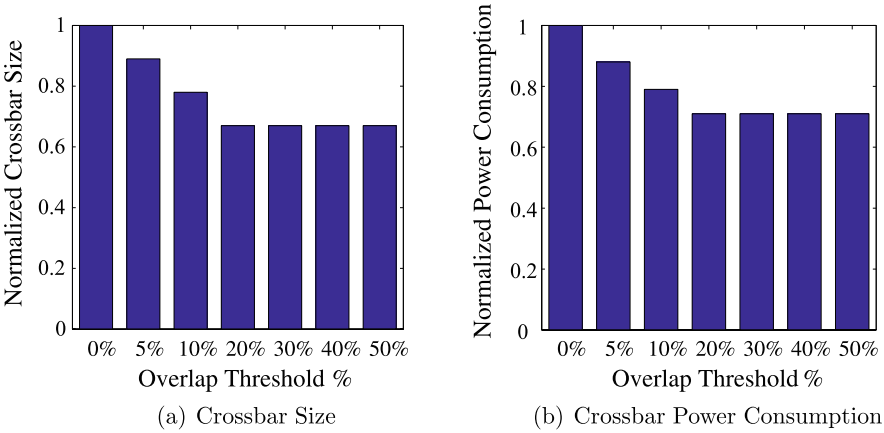


Fig. 2.15 Effect of overlap threshold parameter

2.6 Summary

Today, a streamlined methodology to design crossbar based architectures is not yet fully developed. Toward this end, in this chapter, we have presented methods that address this important problem of designing optimal crossbar based systems for SoCs. The approaches consider the real application traffic, accounting for the local variations and temporal overlap of the traffic streams. As a crossbar system is wiring dominated, it is important to consider the wiring complexity during the design of the architecture. We consider this by utilizing physical design aware methods that consider the layout of the design. We have validated the methods using two state-of-the-art industrial platforms: STBus and AMBA AXI, that are widely deployed in several industrial designs. Equipped with this knowledge of designing bus-based systems, in the subsequent chapters, we will proceed to design general NoC systems that can handle the global traffic requirements of SoCs.

http://www.springer.com/978-1-4020-9756-0

Designing Reliable and Efficient Networks on Chips
Murali, S.
2009, X, 198 p., Hardcover
ISBN: 978-1-4020-9756-0