MAPPING MULTIMODE SYSTEM COMMUNICATION TO A NETWORK-ON-A-CHIP (NoC)

A Thesis

by

PRAVEEN SUNDER BHOJWANI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2003

Major Subject: Computer Engineering

MAPPING MULTIMODE SYSTEM COMMUNICATION TO A NETWORK-ON-A-CHIP (NoC)

A Thesis

by

PRAVEEN SUNDER BHOJWANI

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Rabi N. Mahapatra (Chair of Committee)	Duncan M. H. Walker (Member)
A. L. Narasimha Reddy (Member)	Valerie E. Taylor (Head of Department)

December 2003

Major Subject: Computer Engineering

ABSTRACT

Mapping Multimode System Communication to a
Network-on-a-Chip (NoC). (December 2003)
Praveen Sunder Bhojwani, B. Tech.(Honors),
Indian Institute of Technology, Kharagpur, India
Chair of Advisory Committee: Dr. R. N. Mahapatra

Decisions regarding the mapping of system-on-chip (SoC) components onto a NoC become more difficult with increasing complexity of system design. These complex systems capable of providing multiple functionalities tend to operate in multiple modes of operation. Modeling the system communication in these multi*modes* aids in efficient system design. This research provides a heuristic that gives a flexible mapping solution of the multimode system communications onto the NoC topology of choice. The solution specifies the immediate neighbors of the SoC components and the routes taken by all communications in the system. We validate the mapping results with a network-on-chip simulator (*NoCSim*). This thesis also investigates the cost associated with the interfacing of the components to the NoC. With the goal of reducing communication latency, we examine the packetization strategies in the NoC communication. Three schemes of implementations were analyzed, and the costs in terms of latency, and area were projected through actual synthesis.

To my parents Neela and Sunder.

ACKNOWLEDGMENTS

Thanks are first due to Dr. Rabi Mahapatra. His insight and guidance have been instrumental in the completion of this research.

I would like to thank the members of the Network-On-Chip group, past and present, Narayanan, Subrata and Brenna, for all their help in furthering research in this domain.

I would further like to thank Siddharth, Junyi and Anand, for the endless hours of relief from the monotony of research.

Thanks also go to Lee Vick at Tensilica Corp. for aiding this research by helping with the synthesis of the TIE code and providing the important results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

With on-chip physical interconnections becoming a limiting factor for performance and possibly energy consumption in modern day system-on-chips (SoC), designers face the challenge of utilizing suitable interconnect architectures to guarantee reliable operation of the interacting components. The shared bus, which is today's dominant interconnect template, will not meet the performance requirements of tomorrow's systems [6]. A suitable replacement in the form of an on-chip packet-switched interconnection template has been suggested by many researchers [6, 3, 8]. These networks-on-a-chip (NoC) are well suited for heterogeneous communication among cores in a SoC environment and would address the performance and the scalability requirements of the SoCs.

Researchers have suggested the usage of regular layouts for the cores/resources (processor cores, memory cores, etc.) constituting the SoC [3, 8], i.e. utilizing topologies like the folded torus or the mesh. The communication architecture for such systems consists of the basic building block, the *network tile*. These tiles are connected to an on-chip network that routes packets between them. Each tile may consist of one or more cores and would also have routing logic responsible for routing and forwarding the packets, based on the routing policy of the network. Fig.1 below illustrates the generic network-on-chip architecture that has the folded torus topology. Given the on-chip interconnect architecture, the system designer needs to make an important decision as to how the resources in the SoC are to be interconnected, i.e. mapping the resources to the NoC topology, thereby deciding which resources are to

---

The journal model is *IEEE Transactions on Automatic Control*.

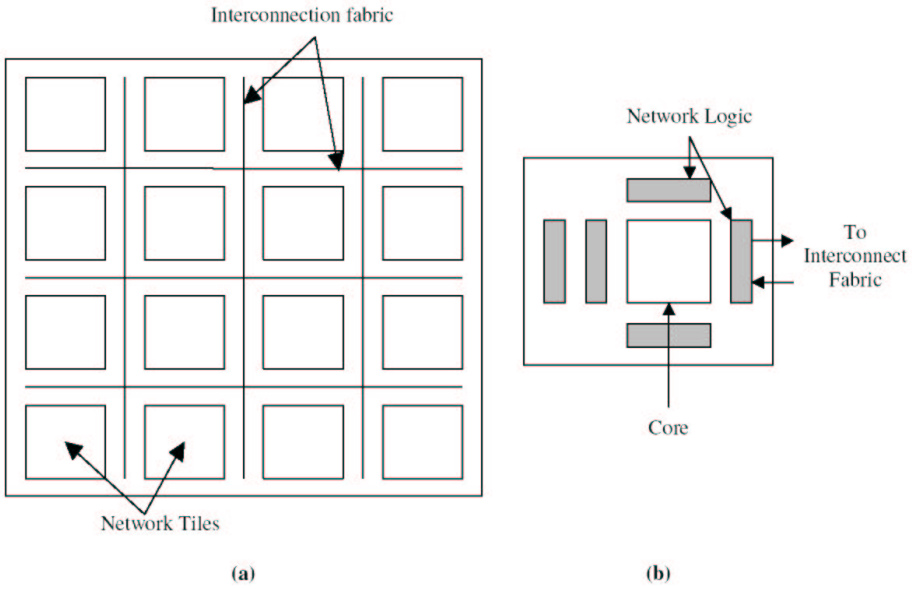

Fig. 1. (a) Generic Network-on-Chip architecture (b) Network tile structure

be placed next to one another in the NoC and how the communication in the SoC is routed amongst these resources. The obvious drawback of cementing the on-chip interconnect network is that a resource in the system may not have a direct communication link to the other resources that it may communicate with. These extraneous communications will have to be routed through the switching logic of the network tiles that are directly connected to it. So the selection of the immediate neighbors of a resource now becomes crucial. The decision of how each resource is connected to the rest of the resources will affect the performance of the system being developed.

The decision making process is further complicated when the systems being developed are conglomerates of functionalities. These *multifunctional systems*, while providing multiple functionalities, also have a consequential increase in the communication complexity within the system and make decisions on the immediate neighbors more complex.

For a better portrayal of the system communication we consider the operation

of the system in different modes for its multiple functionalities. Traditionally, when a system is referred to as being multimode, the mode is associated with the mode of operation of the resources that constitutes the system [13]. For example, consider a multifunctional system such as the present day cell phone, which can behave as a MP3 player, a digital recorder, a PDA and a cell phone. This system is not only capable of operating in the four basic modes of operation that have been listed here, but can also operate as an MP3 player and a PDA at the same time. So the number of possible modes depends on the number of functionality that the system provides. The resources that constitute the multifunctional system, are capable of operating in more than a single mode of the system and the operation characteristics, such as execution times and deadlines may differ in the different modes of the system.

It is evident from above that the communication requirements of the resources may be different in the various modes of its operation. By providing the ability to model the communication in the system based on the different modes, we get a clearer picture of the communication characteristics and requirements of the on-chip interconnect. We refine the original concept of the multimode system, to now reflect the modes of communication in the system. The availability of this mode-based communication model supplements a system development strategy that will decide the interconnections amongst the resources of the SoC.

This research provides the mode-based communication model that allows for the abstraction of the multimode communication within the system being developed. The information provided by the model aids in reducing the design space explored and yields a solution that conforms to the communication restrictions set by the design specifications. A design heuristic to aid in determining the immediate neighbors of a resource in a NoC topology and the routing of the communication within the system is also presented here.

The solution obtained does not provide the exact mapping of resources onto the NoC. It only determines the immediate neighbors of the resources in the SoC and the routes taken by all the communications in the system. This flexible solution provided can then be mapped onto the NoC template, depending on placement constraints that the system designer may have. A scenario where such flexibility may be essential is when a particular resource may need to be placed along the chip boundary for I/O operations. Hence by not providing the exact mapping, we give the system designer the flexibility to place the resources anywhere, while guaranteeing latency-performance constraints, as long as the neighbors are set according to those specified by the given solution.

The solution obtained through the mapping heuristic is verified through a network-on-a-chip simulator, *NoCSim* [15]. When operating in a trace-based mode, this simulator generates packets using execution traces of the resources in the SoC. The packets are injected into the NoC to obtain latency-performance characteristics.

Another issue that has to be addressed before such a system can be deployed is the on-chip communication latency. We need to reduce this latency as much as possible, at every stage of the data communication. The communication comprises of three stages, the packet assembly, packet transmission and the packet disassembly and delivery. This research examines the latency characteristics in the packet assembly stage of the on-chip communication.

The different packetization strategies that have been investigated in this research are

1. Software library based,
2. On-core module based,
3. Wrapper based.

The implementations vary depending on the reconfigurability and programmability of the core in question. This research investigates the suitability of these three methods and determines the subsequent performance differences between them. The results will provide crucial information to the system designer at the time of *core-network interface design*.

CHAPTER II

PRESENT STATUS OF THE QUESTION

The concept of SoC network communication in the form of packet-switched communication was suggested by Guerrier and Greiner [6]. They proposed a generic interconnection template that addressed the performance and scalability requirements of system-on-chip using integrated switching networks. In [3, 8], the authors discussed the advantages of using regular NoC architectures.

Micheli and Benini [5] proposed that on-chip micro-networks, designed with layered methodology, would meet the distinctive challenges of providing functionally correct, reliable operation of interacting system-on-chip components. This idea was also suggested by Sgroi et.al. [12]. They suggested a formal approach to system-on-chip design. Their approach considered communication between components as important as the computations they performed.

The mapping problem is a well studied problem and has been very important with respect to multiprocessor systems. Most of the work in this regard has dealt with assigning tasks to processors and the focus has been on minimizing the communication overhead. A graph theoretic approach in [9], assumes fixed cost communciation edges. The algorithm then determines a n-way cut, aggregating the cost of the communication to be sum of the communication across the cuts. Formulation of the mapping problem as an optimization problem, has also been done by numerous authors. Bokhari [1] uses the number of task graph edges that fall onto the host graph edges, as a mapping evaluation function. Also, the algorithm assigns a uniform traffic intensity to all pairs of communicating processes. This is an unreasonable restiction considering the possibility of variation in the communication profiles. Using simulated annealing, the authors in [2], frame the optimization problem as a two-phase strategy.

In the first phase, they assign the tasks to the processors and in the later phase, they schedule traffic connections over network data links to reduce interprocess communication conflicts. Simulated annealing was also used by Steele [14], but the network contention issues were ignored. In [10], the authors provide a mapping heuristic that is guided by an evaluation function that approximates the total completion time of the given assignment, by taking into consideration the communication delays caused by the network contention. [11] provides a heuristic to reduce the channel contention when mapping.

To the best of our knowledge, no past work provides a flexible mapping solution of the resources to the NoC by determining the immediate neighbors of the resources constituting the system. No previous work addresses this mapping process for the case of multifunctional systems with multiple modes of operation. The absence of a suitable model to accommodate the multimode communications of complex heterogeneous systems and a design heuristic to determine the mapping of the resources onto the desired NoC topology were the factors driving this research.

The need for a test bed to simulate systems built on the NoC backbone led to the development of *NoCSim*, a network-on-chip simulator [15]. This simulator was developed using SystemC [16] and was initially designed with the intention of injecting packets generated by random sources that would have parameters to control the packet injection rate characteristics. One of the drawbacks of this system was the lack of support for simulating real-life system communication characteristics. NoCSim was upgraded to provide it with the ability to generate NoC traffic based on the execution traces of the target applications. The following section describes the multimode system model.

Current research does not address design methodologies that are NoC-aware, i.e. they do not consider NoC issues that directly affect system performance. The

absence of a design validation environment hampers the ability of researchers from analyzing NoC related concepts. Research is also required to examine the various costs associated with interfacing cores to the NoC environment. This research will address these issues.

CHAPTER III

MAPPING MULTIMODE SYSTEM COMMUNICATION TO A NOC

A. Multimode System Communication

A multimode system is defined by a set of a fixed number of modes. Each mode is characterized by a set of communicating resources (i.e. processing elements (PEs), memory elements, etc.) and their corresponding communication characteristics. A graph representation M_i(R,T) - termed as the *mode graph*, contains resources (R) as the nodes, the communication traffics (T) between these resources as the edges and operates in mode i. This mode graph describes the resources that operate and contribute to the communication traffics in the mode being represented in the graph. The communication edges are characterized by the communication parameters, i) *inter-arrival time* (IA) - the minimum inter-arrival time between (outgoing communication requests) requests for communication to another resource, ii) *deadline* (DL) - for the completion of the communication and iii) the *size of data* being communicated (DS) - to determine the traffic load of this communication edge.

With the aid of the individual mode graphs, the communication requirements of the system in each mode are highlighted. Combining the individual mode graphs yields the combined mode graph (M*), which will provide system developers with the communication requirements of the system.

$$M* = \bigcup_{i \forall MODES} M_i(R_i, T_i)$$

where $M_i(R_i, T_i)$ is the mode graph for mode i, with resource set R_i and communication traffic set T_i. The resource set may be the same for different mode graphs. The variable factor will be the communication edges.

Let us now define some notations that can be used to specify the communications in these multimode systems. The communication traffic present in a multimode system is characterized by a set of traffics T, such that $T_{i,j}^{R,S} \epsilon\ T$, where R is the resource from where the communication originates, S is the sink for this communication, i is the mode of the traffic patterns and j ranges from 1 to k, where k is the number of traffics originating from R in mode i. The attributes of the traffic are the inter-arrival time (IA), deadline (DL) and the size of the data (DS). For example consider the following traffic set:

$$T = \{T_{1,1}^{A,B}, T_{1,2}^{A,C}, T_{2,1}^{A,B}, T_{1,1}^{B,A}, T_{2,1}^{C,A}, T_{2,2}^{C,B}\}$$

From this traffic set, we can determine the following information:

1. This multimode system has 2 modes (mode 1 and 2),
2. It has 3 resources (A, B and C).
3. Resource A has 3 traffics originating from it, 2 of them are in mode 1 and 1 in mode 2.
4. Resource B has only 1 traffic, in mode 1.
5. Resource C has 2 traffics, both in mode 2.

This traffic set can further be partitioned into set of traffics associated with a particular mode.

$$T_1 = \{T_{1,1}^{A,B}, T_{1,2}^{A,C}, T_{1,1}^{B,A}\} - traffic\ in\ mode\ 1$$

$$T_2 = \{T_{2,1}^{A,B}, T_{2,1}^{C,A}, T_{2,2}^{C,B}\} - traffic\ in\ mode\ 2$$

Fig. 2 above shows a graphical representation of the combined mode graph of the

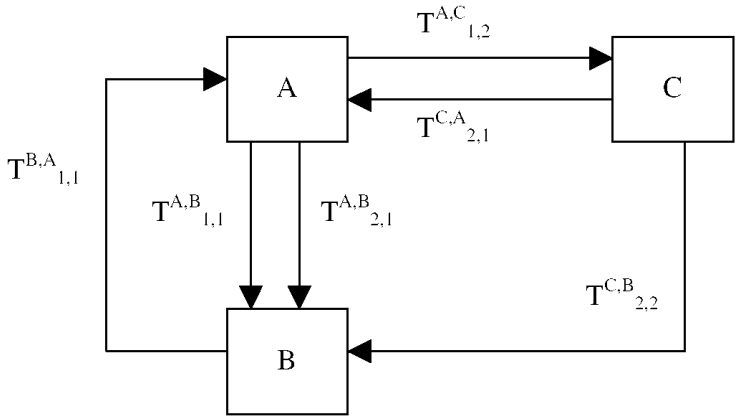

Fig. 2. An example combined mode graph with the traffics in the different mode being highlighted

example above. The mode graph for this example is:

$$M* = \{M_1(R_1, T_1) \bigcup M_2(R_2, T_2)\}$$

$$R_1 = \{A, B, C\} \qquad T_1 = \{T^{A,B}_{1,1}, T^{A,C}_{1,2}, T^{B,A}_{1,1}\}$$

$$R_2 = \{A, B, C\} \qquad T_2 = \{T^{A,B}_{2,1}, T^{C,A}_{2,1}, T^{C,B}_{2,2}\}$$

It is important to note that the communication traffic in one mode may not conflict with the communication traffic in another mode. The information provided by the mode-based communication model aids in reducing the design space explored and enumerates the necessary attributes needed by the mapping heuristic presented in the following section.

B. Mapping Multimode System Communication to a NoC

A crucial step in the system design methodology, the mapping stage facilitates the determination of the immediate neighbors of the resources in the NoC topology. The reason for its importance is that it is not always possible to provide a direct com-

munication channel between a particular resource and all the resources that it may communicate with. The NoC topology constrains the number of immediate neighbors that a resource (present in a network tile) may have when placed on the interconnect template. So the communication traffics of the resource can now be partitioned into two sets - (i) communication to the immediate NoC tile neighbors of the resource and (ii) communication to the rest of the resources. The latter set of communication will have to be routed through the routing logic of its immediate neighbors. Hence an important design decision has to be made to select the resources critical enough to be placed as immediate neighbors, and route the rest of the communication through the switching logic of these immediate neighbors. We provide a heuristic that aids in determining the immediate neighbors of each resource in the system, without providing the exact placement, and also the routes taken by communications to the rest of the resources in the system.

1. Problem Formulation

The essential idea that has been used here to solve the problem, involves the transformation of the arbitrary cardinality *combined mode graph* (or resource interconnection graph), into a *network graph* of fixed cardinality. As described in the previous section, the combined mode graph (M*) provides the communication characteristics of each resource in all modes of operation of the system. The network graph (N) shows how the resources of the SoC are interconnected in the NoC topology, and can further be mapped onto the NoC topology selected for the system. The network graph is characterized by a *resource set* (NR), which is the node set of the graph and is equivalent to the resource set of the combined mode graph (M*). The network graph's edge set is characterized by *network edge set* (NE). It has a restriction set on it by the target topology of the NoC being utilized for the SoC. The restriction limits the number of

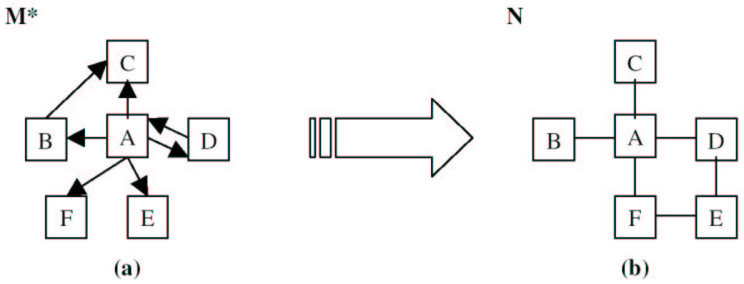

Fig. 3. (a) Combined mode graph (b) network graph

outgoing edges from a *network resource nr* ϵ NR. For example, if the NoC topology to be used in the SoC is the folded torus, then the number of outgoing links possible on the network resource is 4. Another property of the network edges is that if a resource A has a network edge to resource B, then resource B must also have a network edge to resource A. Each network edge in the network graph has associated with it a set of communication traffics that flow through it. It also has associated with it, the average communication delay offered by the network edge in each mode of operation of the system. This network graph is clearly a representation of the resources and their communications in the NoC, with the only exception being that the position of the resources has not been fixed.

Fig. 3 shown above summarizes the crux of the transformation operation. The input to the heuristic is a combined mode graph. If the target NoC architecture has a folded torus topology the number of interconnections on a resource is constrained to four. The mapping heuristic attempts to determine the four immediate neighbors in the network. Once these have been decided, it also determines the suitable candidates of these four that will support the communication requirements of the extraneous edges.

Example: In the example in Fig. 3(a), we note that resource A is connected to resource B, C, D, F and E. When mapped to the folded torus topology, the heuristics

decides that the 4 immediate neighbors have to be B, C, D, and F. The communication between A and E has to now be routed either through C, D, B, or F. The heuristic ranks the candidate immediate neighbors and selects the best candidate to merge the extraneous communications with, thereby deciding on the route that this communication will take. In this case it decides that communication between A and E can be supported by either F or D. Fig. 3(b) provides the final mapping of the immediate neighbors of all resources in the system.

2. Preliminaries

Before we present the mapping heuristic, the reader must be made aware of some of the terminologies used in the mapping heuristic.

Definition 1: A *candidate* for transformation is a node in the combined mode graph that has a cardinality greater than the constraint set by the target NoC topology, i.e. we attempt to solve the most difficult problem first.

$$Candidate = max\{\mid T_i \mid \ : \ i \ \epsilon \ R\}$$

where R is the resource set of the system and — T_i — is the size of the communication traffic set of resource i.

Definition 2: A *secondary edge* is a communication edge on a resource in the combined mode graph, that has not yet been merged with a network edge of the resources. The attributes associated with a secondary edge are the communication traffics $T_{i,j}^{R,S}$, i.e. the inter-arrival time, deadline and the amount of data being transmitted.

Definition 3: A *critical edge* in the secondary edge set is the next edge that is selected by the heuristic to merge with one of the network edges. The selection of the critical edge is made from the traffic set T_R, where R is the candidate resource being

considered. The heuristic examines the traffic in all the modes and then decides on the critical edge that will it attempt to place.

We now define the operations that are performed by the mapping heuristic.

Operation 1: In the *merge* operation we assign the secondary edges to the selected network edge and update the delay values on that network edge. The merge operation has three possible scenarios.

1. Case 1: If the network edge has no destination assigned yet, we set the destination of the network edge to be the same as that of the critical secondary edge.

2. Case 2: If the network edge destination is the same as that of the secondary edge, we add the secondary edge to the network edge and update the delay in the mode that is affected.

3. Case 3: If the network edge destination is not the same as that of the secondary edge, we need to add this secondary edge to the network edge and propagate it forward. We also update the delay in the mode that is affected.

Operation 2: In the *rank* operation, the heuristic assigns scores to the candidate network edges. The ranking results are used to determine the network edge to merge with. The scores are assigned based on four criteria:

1. Criteira 1: If the network edge has already been visited as a solution for this edge. This condition is checked to prevent cyclic passing of of secondary edges when we merge-and-propagate. This helps keep a check on the cyclic propagation of secondary edges amongst the resources.

2. Criteira 2: If the network edge was backtracked from.

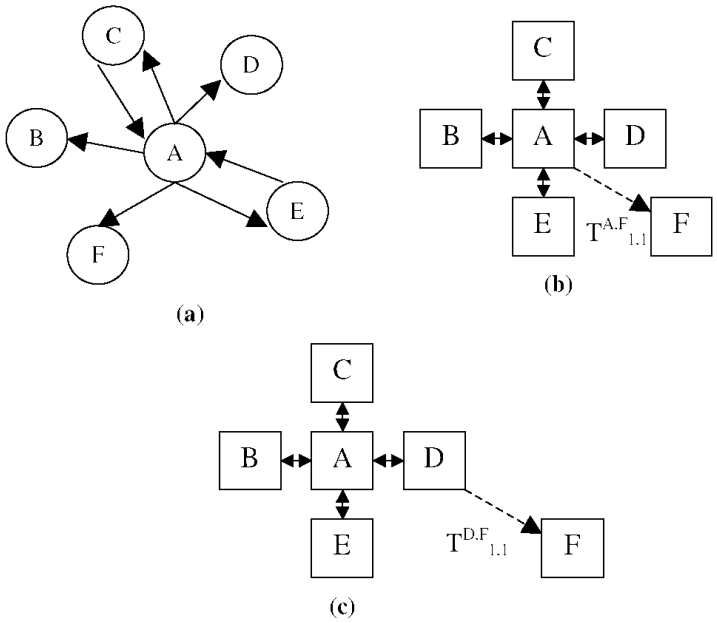

Fig. 4. An example demonstrating the merge operation

3. Criteira 3: The number of secondary edges that need to be assigned at the resource pointed to by the network edge

4. Criteira 4: The number of secondary edges - at the resource pointed to by the network edge - whose communication clashes with the communication mode of the candidate secondary edge.

Example: In the example in Fig. 4, consider the given mode graph (assume edge labels in the form of $T_{i,j}^{R,S}$, they have been ignored here so as to provide a neater figure). We are to map this resource interconnection graph to the folded torus network topology. We need to reduce the cardinality of resource A to 4 and route the extra edge through one of the other resources. The heuristics selects the four most critical outgoing edges of resource A and assigns them to the four vacant outgoing network edges. In this case, it has selected the edges to B, C, D and E to be the critical edges and hence places B, C, D and E as the immediate neighbors of A. The edge to F

($T_{1,1}^{A,F}$) needs to be routed through either of the four outgoing edges, i.e. we need to merge the communication of A to F through either B, C, D or E. From Fig. 4(c) we notice, that the heuristic selects resource D to be the candiate through which this connection is to be routed. This selection is done based on the results obtained by the ranking operation. So we now merge $T_{1,1}^{A,F}$ to the network edge from A to D. The communication from A to F, has now been decomposed into communication from A to D and from D to F. So we now need to have a secondary edge at D, pointing to F ($T_{1,1}^{D,F}$). This new traffic edge that is added to D will now have updated traffic information that considers the delay incurred on network edge.

3. Heuristic

The heuristic presented in this paper takes in the combined mode graph of the system as input and transforms the arbitrary cardinality graph, into a fixed cardinality network graph. This network graph provides the necessary interconnections of the resources in the system. The solution provided can then be mapped to the NoC topology of choice. Table 3 enumerates the steps in the heuristic.

The heuristic that has been provided here is iterative in nature. In each of the iterations we consider a resource of the system and attempt to make decisions on its immediate neighbors in the NoC. This resource is termed as the candidate. It has a set of communication edges, called the secondary edges. These secondary edges are of the form $T_{i,j}^{R,S}$ and belong to T, the communication set of the combined mode graph. The objective of the heuristic is to merge these secondary edges with the network edges. These network edges are nothing more than the network links between tiles in the NoC network.

When we attempt to merge a secondary edge with a network edge, there are three possible scenarios. These scenarios were have already been highlighted in the

Table I. Heuristic for mapping the combined mode graph of a system onto a NoC topology.

```
while (resource graph has more candidates for transformation) {
    candidate = getNextCandidate( )
    SE = {set of secondary edges of resource candidate}
    NE = {set of network edges of resource candidate}
    while (SE has more secondary edges to merge with the network edges of candidate){
        se = getCriticalSE( )
        if (a network edge to the destination of se already exists) {
            if (merge with this network edge is possible) {
                Merge se with this network edge
                update the slacks and the edge routes for each edge
                Rip up the old routes through this network edge and update their values
            }
        else {
            if (an empty network edge to connect to the destination of se is available) {
                if (a return edge to candidate is possible) {
                    merge se with this empty network edge
                    update the slacks and routes for this edge
                    set the return edge to point to candidate
                }
            else {
                rank network edges to select best candidate for merge
                select the best possible candidate network edge to merge with
                merge se with this network edge
                update the slacks and the edge routes of the affected edges
            }
            if (a merge is not possible) {
                we need to backtrack and pass back the violating edge back to its  predecessor
            }
        }
    }
}
```

previous section. In case 1 of the merge operation, when the network edge has no destination specified yet, the merge operation will set the destination resource of the secondary edge as an immediate neighbor of the candidate. In case 2 of the merge operation, the network edge has the same destination as that of the secondary edge. Here we just merge the communication and update the delays for the different modes of communication on this network edge. In case 3 of the operation, the destination of the network edge is not the same as that of the secondary edge. In this case too, we merge the communication with the network edge. Aside from that we also add a secondary edge to the destination resource pointed too by the network edge. This new secondary edge would have the same destination as that of the critical secondary edge being merged. The only difference is that the communication characteristics on this edge have been updated after delay updations on the network edge. These steps are performed for each un-merged secondary edge in each resource in the system.

Example: In Fig. 5, an example execution of the heuristic is demonstrated. In (a), we present the initial configuration of the resource interconnection graph. In (b), we have selected resource A as the candidate (since it has the highest cardinality). Resource A now becomes a network graph node. In (c), we select the communication to G to be a critical communication edge. The decision to make G an immediate neighbor to A is made. After three more iterations, we select the next three critical communications. B, C and H are selected and assigned as the immediate neighbors of A in (d). In (e), we address the communication edge from A to F. This edge is merged with the edge to G, and the secondary edge is propagated to G. In (f), resource C is selected as the candidate node. In (g), D and E are set as the immediate neighbors of C. In (h), we finally assigned F as the immediate neighbor of G. We now have the complete network graph. This graph provides us with the immediate neighbors of all the resources in the NoC.

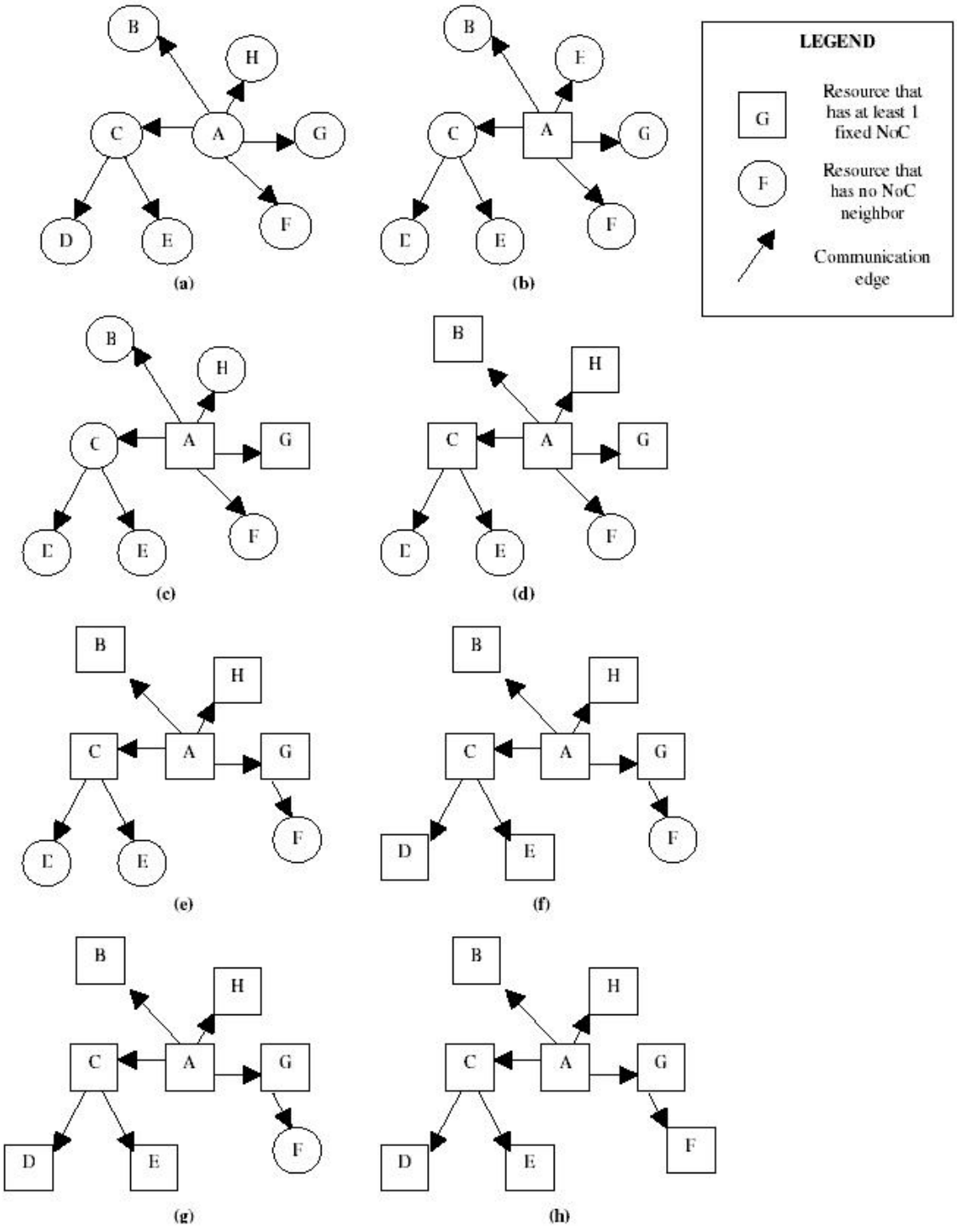

Fig. 5. Sample execution of the mapping heuristic

C. NoCSim - A Verification Test Bed for Network-on-Chips

NoCSim is a network-on-chip system simulator developed using the popular system-level design programming language SystemC. The simulator operates at the flit-level and utilizes a virtual-channel based flow control scheme [4]. It has some parameterizable parameters, such as the number of virtual channels, the buffer depth, and the size of the mesh network. Presently, the simulator has two modes of traffic generation. In the first mode it can generate traffic with a constant bit rate (CBR) and random Poisson distribution. The user using configuration files can specify the traffic generation parameters. In the second mode, the traffic generated is trace-based, i.e. the traffic is generated using exact communication traces. These communication traces provide the information needed by a traffic generator, i.e. the time at which to generate the packet and the amount of data to transmit over the network. The traffic generated is routed from the sources to the destinations for which it is configured. It then provides the end-to-end latency information for each flit that can be used to evaluate the performance of the system designed. The results obtained from the mapping heuristic will be used to setup NoCSim and run simulations. These simulations will provide latency details that will validate the results of the mapping heuristic.

D. Experiments and Results

To test the mapping heuristic, we developed two test cases. The first test case consists of a set of random mode graphs that are to be mapped onto a 4 x 4 folded torus topology. We chose to create the random data set because of the lack of a complex real-life system to suitably test the mapping heuristic. As shown in Fig. 6 below, the target system is capable of operating in three modes. The mode graphs are input to the mapping heuristic to obtain the immediate neighbor information for each resource

in the system. This information is used to map the resources onto the folded torus topology, as can be seen in Fig. 6. The mapped solution is used to configure NoCSim and run simulations. We compare the latency results obtained from NoCSim to those set as constraints in the mode graphs. This comparison will aid in validating the mapping solution.

Table II. Sample set of the latency results obtained from NoCSim vs those provided as constraints to the mapping heuristic.

S.No.	Comm. Source(S)	Comm. Destination(T)	$T_{i,j}^{S,T}$			NoCSim latency results
			IA	DS	DL	
1	3	14	6	64Kb	9	3
2	15	4	7	64Kb	9	4
3	4	10	9	64Kb	9	10 *
4	14	1	9	64Kb	7	7
5	6	9	9	64Kb	8	6

The results obtained from NoCSim provide the latency characteristics of all communications that take place in the NoC configured. Table II above shows a sample set of the results obtained. This sample set shows the communication characteristics ($T_{i,j}^{S,T}$) of some of the communication traffics in the system. The last column of the table shows the latency results obtained from NoCSim. We compare the latency constraints set on the communication traffics to that obtained from NoCSim. We note that the communication constraints are met for most of the traffic. For this test case, only a single communication constraint was violated (* in the above table). This violation is due to the fact that the mapping heuristic approximates the delays on the network edges in the NoC. However, our experiments on other test cases have shown that the error in approximation is limited to 12%, and can further be reduced with better delay models.

For our second test case, we developed a hypothetical multimode system capable of providing four functionalities and capable of operating in three modes. The four selected functionality were JPEG encoder, MPEG encoder, ADPCM and MP3

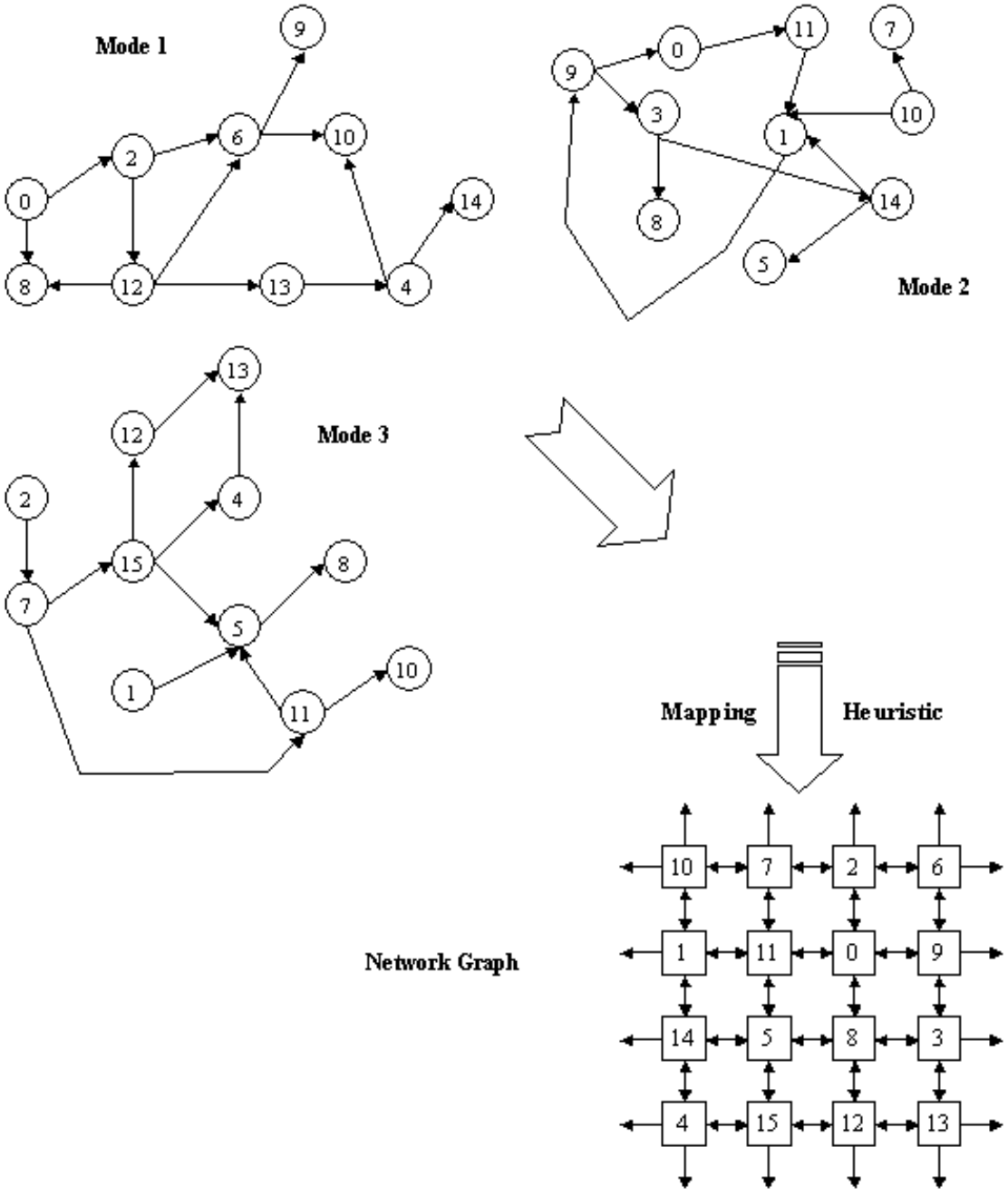

Fig. 6. The result obtained after mapping the 3 input mode graphs onto a 4 x 4 folded torus topology. The final layout of the resources in the NoC has also been shown

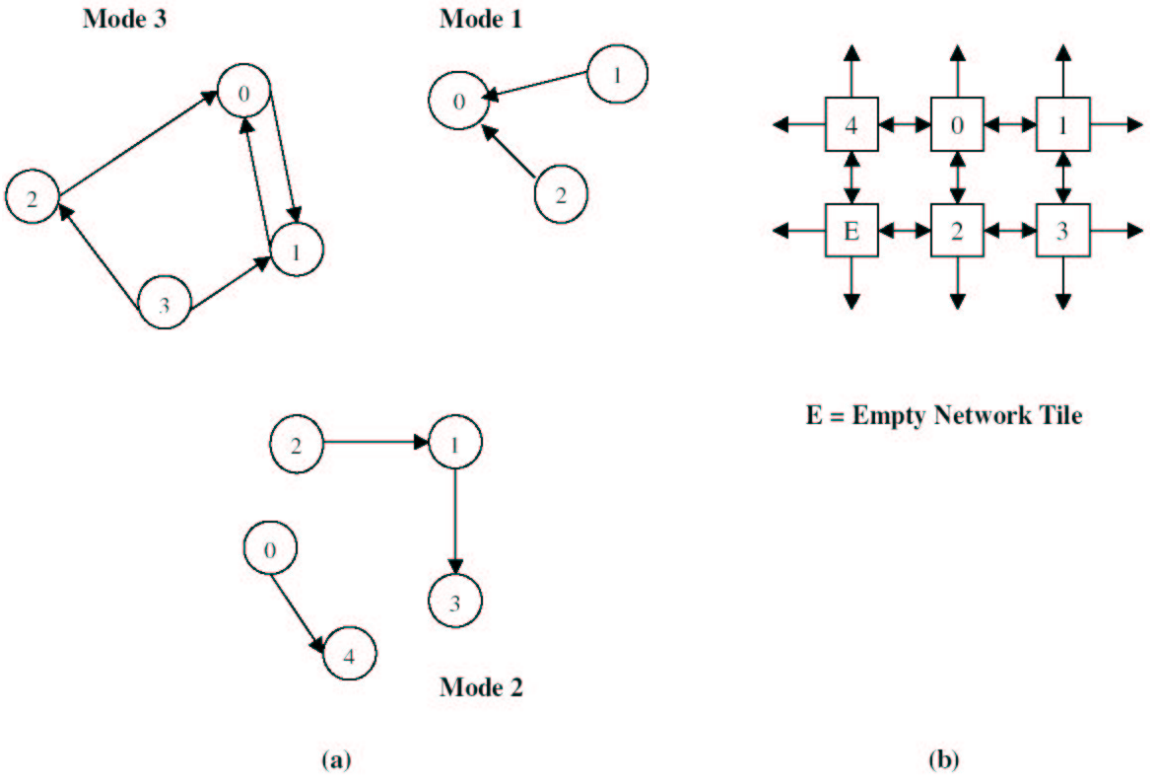

Fig. 7. (a) The mode graphs of the system being developed. The three mode graphs provide four functionalities (JPEG, MPEG, ADPCM, MP3). (b) This shows the mapping of the four resources on to the NoC

Table III. Sample set of the latency results obtained from NoCSim vs those provided as constraints to the mapping heuristic.

S.No.	Comm. Source(S)	Comm. Destination(T)	$T_{i,j}^{S,T}$			NoCSim latency results
			IA	DS	DL	
0	0	4	17	96Kb	60	50
1	2	3	63	192Kb	100	98
2	1	0	2.82	40Kb	25	22
3	1	3	1076	128Kb	75	67
4	2	0	260	256Kb	128	132 *

decoder. We partitioned the modules constituting the functionalities, into four resources. These would be placed on the NoC and used to provide the above functionalities. Fig. 7 above shows the mode graphs input to the mapping heuristic and the final mapping of the resources on the NoC. The mapping obtained was verified through NoCSim and a sample set of the latency results obtained, have been presented in Table III. In this test case, most of the latency constraints were met. Only one of the constraints was violated (* in Table III).

CHAPTER IV

INTERFACING CORES WITH ON-CHIP PACKET-SWITCHED NETWORKS

A. Introduction

With the onset of packet-switched networks being a possible mode of communication on SoCs, various aspects of the communication need to be evaluated and optimized to provide the required quality of service (QoS). In order to reduce the packet-switched network on-chip communication latency, several schemes are possible, starting at the compiler-level where the compiler will place instructions - requiring communication medium usage - earlier in the sequence of execution, to have the controllers such as those of memory pre-fetching and transmitting data to the consumers to reduce latency. But these can only be addressed once such a network is deployed.

In this work, we address the core-network logic interface issues. The packet communication process is a target for the reduction of latency. The packet communication process has essentially three stages - packet preparation, packet transmission and packet handling at receiver. Primarily we look at the packet preparation stage of the communication over the network. This is the period from where the processor knows that it has to communicate with an external component (w.r.t. to its tile), to the time it delivers the packet to the network logic of that tile, which eventually delivers it to the destination component. Since this stage could be a possible bottleneck, apart from the latency of the communication channel, we try to reduce the latency exhibited by the system at the beginning of the communication process. These results can also be used for analyzing the system for the final stage of communication too, the packet handling stage. Since these stages are essentially complementary, they will

exhibit similar tendencies.

Another important issue that arises here is of whether the core should be aware of the network or not. The pros for a network-aware core are:

- Reduced latency, because the core directly provides the packet, once it is informed of the packet format.
- Reduced complexity of the network interface of the core.

The cons of a network-aware core are:

- Specification of packet parameters to the core.
- Core requires a certain degree of programmability.
- Need to modify existing cores to make them network-aware.

The experimental scenario considered for this research, was of a simple distributed memory environment. The system consists of a core that can access separate memory cores spread through the on-chip network. To the software executing on the processor core, the memory is one contiguous block present at a single location. The processor core is aware of the distributed nature of the memory space. When the software attempts to access a memory location, the destination core has to be identified and accessed. We shall demonstrate how this is implemented in the three different schemes. Before we examine the packet preparation steps and methods, we take a look at the generic structure of the packets. The structure of the packet can be tuned for a particular network, so as to reduce the overhead of packetizing the data. A packet essentially consists of 3 parts, the packet header, the packet data and the packet tail. The packet header contains the necessary routing and network control information. These will be the destination and source addresses. When source routing is used, the

Table IV. Generic packet structure.

Packet Header	Packet Data	Packet Tail

destination address will be ignored. It is replaced with a route field that will specify the route to the destination. A disadvantage of source routing is the added overhead of including the route field in the packet header. But the inclusion of such a field reduces the complexity of the routing logic on the cores on the network. It simplifies their routing decisions and their task will be to just look at the route field and route the packet over the specified output port. The packet data consists of essentially two types of information. The first is the control information that will indicate to the receiving memory core about the type of memory request being made. The second will be the actual data, i.e. the memory address being accessed. The packet tail contains error-checking code and error-correcting code. But this part of the packet is optional. The inclusion of this information will depend on the error probability of the underlying network. The packet structure utilized for this research was tuned to the corresponding implementation strategy. With the simple distributed memory environment scenario in mind, we identified the generic operational steps that need to be performed when an address at a memory core needs to be addressed. Fig. A illustrates these steps. The set of operations stated above are executed at different locations, depending on the type of packetization strategy. The location will be incumbent on the configurability and programmability characteristic of the core in question.

1. Software Library-based Strategy: The software implementation of the packet preparation provides the user with a library of instructions that can be used

Table V. Generic packetizing process for a simple distributed memory model.

Step 1:	Translate address by determining which memory core needs to be accessed, and determine the effective address at that memory core.
Step 2:	Prepare packet header by setting the source address and the route to the destination.
Step 3:	Examine program instruction requiring memory access, and set control flags in the packet data.
Step 4:	Set effective address in packet data.
Step 5:	If using error-checking codes and error-correcting codes, calculate the values and set them in the packet tail.
Step 6:	Assemble packet and deliver to the network logic of the core.

to access a memory address in a distributed memory space. The cost of this implementation will be considered in terms of the size of the library and the execution time overhead for each instruction.

2. On-core module-based Strategy: In this implementation, a processor with an on-core packetization module will have to be developed. The facility to be able to add on a co-processor to an existing processor, is provided by the Tensilica's Xtensa Core. This configurable, extensible and synthesizable processor core was designed specifically to address Embedded System-on-Chip (SoC) applications. This processor can be molded by the system designer to suit the application. The designer can also describe additional data-types, instructions and execution units using the Tensilica Instruction Extension (TIE) language. Using this core, it is possible to develop an application specific core for packetization.

3. Wrapper-based Strategy: A wrapper compliant with a standard core interface - typically the VSI Alliance's Virtual Component Interface (VCI) Standard - will be developed. This standard defines the basic characteristics of the Virtual Component Interface (VCI). The HDL implementation will be synthesized to provide for the performance costs. The tool used here will be the Synopsys Design Analyzer.

Table VI. Expected characteristics of the packetization schemes.

Type	Area	Latency	Complexity	Flexibility
Software Library (on-core)	Low on HW area, but increases code size (increased instructions to packetize)	High	Increased code size.	Requires programmable cores.
RTL (HW) implementation (on-core)	Additional register and logic to packetize	Low	Additional registers and logic and an increase in instruction set.	Requires programmable cores or development of modified cores.
Wrapper RTL (HW) Implementation (off-core)	Additional control, registers and logic to packetize.	Low	Additional control, registers and logic.Ability to understand core operation	Can use existing cores. Modify wrappers for plug-and-play into different networks.

The trade-offs here would be of latency, area, complexity and flexibility. Table A provides a tabular representation of these features in the three possible implementations. As mentioned in Chapter 1, one of the focuses of this research has been the analysis of the alternative packet preparation methods available to the system designer. For our research, we used the Xtensa Processor Core from Tensilica [17]. This configurable, extensible and synthesizable processor core was designed specifically to address Embedded System-on-Chip (SoC) applications. This processor can be molded by the system designer to suit the application. The designer can also describe additional data-types, instructions and execution units using the Tensilica Instruction Extension (TIE) language. Using this core, it is possible to develop an application specific core for packetization. In the following section we shall discuss three implementations of the packetizing modules.

B. Implementation Details

1. Software Library for Packetization

The software implementation of the packet preparation provides the user with a library of instructions that can be used to access a memory address in a distributed memory space. The library requires three configuration files. These files provide important network associated properties. Fig. 8 provides an overview of the configuration file structure. The *socnet.conf* configuration file specifies the address of the host. It also provides the route information to the network elements. The packet structure used is specified in *packet.conf* and this specifies the fields in the packet and their corresponding size in terms of bits. *mem_alloc.conf* contains the memory allocation information, i.e. the address space of the memory cores in the environment. When the user issues a memory access instruction, the corresponding packet is prepared according to the steps highlighted in the figure A. The sample code to test the library was executed on the basic Xtensa Processor Core. The core was configured with a 128-bit processor interface. The cycle count for the execution of the packetization instruction was determined by using the profiling tool - *xt-gprof* - included in the Xtensa toolset. The area results for this strategy are the size of the software library code.

2. On-core Module for Packetization

In this implementation we utilized the Xtensa Processor Core's configurability and its Tensilica Instruction Extension (TIE) language to define instructions for preparing the packets. This program. The Xtensa toolset has tools that allow the profiling of the executed instructions. From the profile one can obtain the required results such as the cycle-count for the executed instructions The TIE compiler also generates the

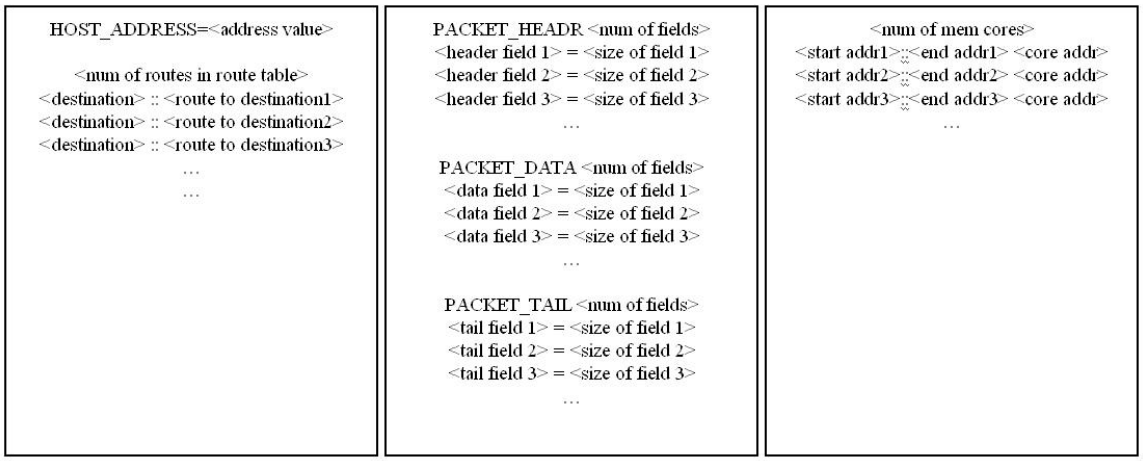

Fig. 8. Configuration file structures

required Verilog/VHDL files that are then analyzed using Synopsys Design Analyzer, to obtain the timing and area costs. The TIE definition used for our research, included the specification of the stages listed in Fig. A, in terms of the TIE language. The TIE code was successfully compiled with the TIE compiler and the execution of the custom instruction was tested on the Xtensa processor. The packet structure used in this implementation is equivalent to the one shown in Fig. A. The cycle count for this implementation was obtained using the Instruction Set Simulator (ISS), provided with the Xtensa tool set. The ISS provides detailed information on the contents of the registers in use and the output available at the processor interface.

3. Wrapper Logic for Packetization

For cores that are neither programmable nor reconfigurable, the only option for interfacing with the networking logic of the tile is to utilize a wrapper, which would have the responsibility of packetizing and de-packetizing the cores requests and responses. The wrappers have the responsibility of (i) receiving the contents from the

core interface, preparing the packets and dispatching them to the network logic of the tile and (ii) receiving the packets from the networking logic and presenting the contents to the core interface. For our experiment, we designed the packetizer module of the wrapper, which was compliant with VSI Alliance's Virtual Component Interface (VCI) Standard Version 2 [18]. This standard defines the basic characteristics of the Virtual Component Interface (VCI). It provides detailed information on the different complexity interfaces, the Peripheral VCI (PVCI), the Basic VCI (BVCI) and the Advanced VCI (AVCI). We developed the wrapper that would be compliant with the BVCI standard. The implementation details are not provided here due to the restrictive nature of the standards document. The VSIA vision is to dramatically improve the productivity of SoC development by specifying open standards and specifications that facilitate the integration of software and hardware VCs from multiple sources. This was the reason we chose to develop a wrapper compliant with the VCI standard because we believe that most future cores will have well-defined interfaces similar to or be VCI standard compliant. The packetizing module attempts to optimize the packets being generated for the on-chip network. The packet structure in this implementation was dependent on the signals used in the BVCI interface (details cannot be provided due to non-disclosure agreement). The packetizer module maintains the address translation information, i.e. the mapping of memory addresses to destination core addresses. It analyses the content of the core request and tries to optimize on the amount of data being sent over the on-chip network, by filtering the redundant information from the packets. The timing, and area analysis for this implementation was obtained using the Synopsys' Design Analyzer.

C. Results

The results obtained for the analysis carried out evaluates the performance of the packetization schemes in terms of latency, and area. Table C provides a summary of the results that were obtained for the latencies experienced. The latency in the case of the software library, was determined using the cycle count obtained from the Instruction Set Simulator (ISS) and the clock frequency. This result will vary with different processors and implementations of the packetization library. In the case of the on-core packetization, the latency was determined in a similar way. However, the clock frequency was obtained through synthesis of the TIE specification. It should be noted here that, these two schemes were implemented on the Xtensa Processor Core. The lower clock frequency is due to the slow-down caused by the TIE logic that was incorporated into the processor core. This is an acceptable trade-off, in light of the performance improvement. This conservative result was obtained by using the TSMC 0.18micron and slow libraries. With a little more effort and better libraries it is possible to have the Xtensa processor operate at its normal clock frequency of 200MHz and will further reduce the latency. The result for the wrapper implementation is obtained using the 0.35micron technology library. With better technology, there will be a further reduction in the latency. The latency result in this case provides the developer with the time taken through the longest path in the wrapper, and will enable him to decide on the clocking rate for the interface.

Table C, provides the area costs of the three schemes that were implemented. The area cost of the three implementations cannot be compared quantitatively. The results provide a measure of the cost that the system designer would experience using a particular strategy. The area for the software implementation was determined in terms of the code size of the software library. To determine the area of the TIE logic,

Table VII. Latency results.

Packetization Strategy	Cycle Count	Clock Frequency	Latency
Software Library	47	193 MHz	243.5ns
On-core Packetization	2	185 MHz	10.8ns
Wrapper Packetization	-	-	3.02ns

Table VIII. Area results.

Packetization Strategy	Area	Remarks
Software Library	118 KB	Code size
On-core Packetization	13K	Gate count using 0.18 micron technology
Wrapper Packetization	4K	Gate count using 0.35 micron technology

for the on-core packetization, the TIE specifications were synthesized following the regular steps (i.e. compilation of the TIE specification and synthesis of the compiler output). The 13K gate count is an overly conservative estimate. The silicon area appeared to be under 0.2 square mm. The area for the wrapper was determined using Synopsys' Design Analyzer. It cannot be directly compared to the one obtained for the Xtensa Core, as the technologies used in both are considerably different.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

The need for a high-performance on-chip interconnect architecture to meet the demands of the complex modern day systems has spurred the on-chip interconnect research community to research into probable solutions. Using technology originated in parallel computing, and merging its ideas with those of the networking domain has presented the network-on-chip as the plausible solution. Minimizing the associated costs requires research into design methodlogy issues.

One such issue that has been addressed here, is the mapping of the heterogenous communicating cores onto the selected topology. This thesis provides a flexible mapping solution that attempts to guarantee latency performance constraints. Other immediate issues that crop up, are those of power budgetting. With power becoming a crucial factor in the design of mobile systems, the mapping will also need to consider the energy constraints of the on-chip interconnects.

Once the on-chip interconnect architecture is selected for a particular system, one cannot just "plug-in" IPs and expect the system to function. The IPs need to be interfaced with the network. With three possible scenarios possible for the interfacing, the system designer needs to select the appropriate core-network interface. The results provided in this thesis will aid system designers in fathoming the costs associated with the possible implementations. Future work in this regard would look to provide more cost effective solutions for core-network interfacing.

REFERENCES

[1] S. H. Bokhari, "On the mapping problem", *IEEE Trans. on Computers*, Vol. 30, pp. 207-214, March 1981.

[2] S. W. Bollinger and S. F. Midkiff, "Heuristic technique for processor and link assignment in multicomputers", *IEEE Trans. on Computers*, Vol. 40, pp. 325-333, March 1991.

[3] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks", in *Proc. DAC*, 2001, pp. 684-689.

[4] W. J. Dally, "Virtual-channel flow control", *IEEE Trans. Parallel and Distributed Systems*, Vol. 3, pp. 194-205, March 1992.

[5] G. De Micheli and L. Benini, "Networks on chip: A new SOC paradigm", *IEEE Computer*, Vol. 35, pp. 70 -78, Jan 2002.

[6] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections", in *Proc. DATE*, 2000, pp. 250-256.

[7] A. Jantsch and H. Tenhunen, *Networks on Chip*, Dordrecht, The Netherlands: Kluwer Academic Publishers, 2003.

[8] S. Kumar, A. Jantsch, J-P. Soininen, M. Forsell, M. Millberg, et al., "A network on chip architecture and design methodology", in *Proc. IEEE Computer Society Annual Symposium on VLSI*, April 2002, pp. 117-124.

[9] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems", *IEEE Trans. on Computers*, Vol. 37, pp. 1384-1397, November 1988.

[10] R. Perego and G. De Petris, "Minimizing network contention for mapping tasks onto massively parallel computers", in *Proc. Euromicro Workshop Parallel and Distributed Processing*, January 1995, pp. 210-218.

[11] L. Schwiebert and D. N. Jayasimha, "Mapping to reduce contention in multiprocessor architectures", in *Proc. Parallel Processing Systems*, April 1993, pp. 248-253.

[12] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, et al., "Addressing the system-on-a-chip interconnect woes through communication-based design", in *Proc. DAC*, 2001, pp. 667-672.

[13] Y. Shin, D. Kim, and K. Choi, "Schedulability-driven performance analysis of multiple mode embedded real-time systems", in *Proc. DAC*, 2000, pp. 495-500.

[14] C. S. Steele, "Placement of communicating processes on multiprocessor networks", Technical Report 5184:TR:85, California Institute of Technology, Pasedena, California, April 1985.

[15] N. Swaminathan, P. Bhojwani and R. Mahapatra, "Communication synthesis for on-chip networks", Technical Report #TR-CS-2002-08-0, Texas A&M University, College Station, 2002.

[16] SystemC 2.0.1, *White paper*, www.systemc.org/projects/sitedocs/document/v201_White_Paper/en/1, [Accessed June 2002].

[17] Tensilica, *Xtensa Core Product Brief*, www.tensilica.com/Xtensa_PB_91102.pdf, [Accessed May 2002].

[18] VSI Alliance, *Virtual Component Interface Standard Version 2 (OCB 2 2.0)*, April 2001, www.vsi.org/resources/datasheets/ocb2ds.pdf [Accessed May 2002].

VITA

Praveen Bhojwani was born in Sharjah, United Arab Emirates (U.A.E.) on the 30th of July, 1979. After completing his schooling at The Modern High School, Dubai, U.A.E., he went on to attain his Bachelor of Technology (Honors) in Computer Science and Engineering at the Indian Institute of Technology, Kharagpur, India, in May 2001.

Permanent Address:
A/41 Archana Co-op. Housing Society,
New Versova Link Road,
Andheri(W),
Mumbai 400 053,
India.

The typist for this thesis was the author.