optiMap: A Tool for Automated Generation of NoC Architectures using Multi-Port Routers for FPGAs

Balasubramanian Sethuraman, Ranga Vemuri
{sethurb, ranga}@ececs.uc.edu
Department of ECECS, University of Cincinnati, Cincinnati, OH 45221-0030, USA.

Abstract

Networks-on-Chip (NoC) way of system design has been introduced to overcome the communication and the performance bottlenecks of a bus based system design. Area is at a premium in FPGAs. In this research, we propose to reduce network area overhead by reducing the number of routers, by making the router handle multiple logic cores. We implement an improved multi-local port router design with variable number of local ports. In addition to substantial area savings, we observe significant performance improvement. We discuss the issues involved in the use of multi-local port routers for NoC design in FPGAs. We observe an average of 36% *area savings (maximum of* 47.5%*) on* $XC2VP30$ *FPGA and significant performance gain (*30% *average compared to single-local port version) with a multi-local port router. Mapping of cores onto such a non-traditional NoC architecture is a complex task. We present an algorithm which optimally maps the cores based on the given set of objectives. For the given task graph and the set of constraints, the algorithm finds the optimal number of routers, configuration of each router, optimal mesh topology and the final mapping. We test the algorithm on a wide variety of benchmarks and report the results.*

1 Introduction

FPGAs are increasingly being used in applications with low to medium volume than their ASIC counterparts, due to shorter design cycles and reduced associated costs. Modern FPGAs [15] with embedded processors can be used for System-on-Chip (SoC) designs. SoC designs are built as a complex interconnection of various functional elements. In the GigaBits era of SoC design, interconnection of cores using bus architectures present a communication bottleneck [7]. Also, they do not present a scalable solution to existing problems in the communication. Networks-on-Chip (NoC) has been proposed as a new design paradigm to solve the communication bottlenecks [10, 3]. The main idea is to avoid shared-bus and implement interconnection of various Intellectual Property (IP) cores using on-chip packet-switched networks [14]. NoCs provide a scalable and modular architecture and help independent design of IP cores and its re-use. A detailed survey of NoC is given in [8].

Motivation: In an FPGA, area is available at a premium and hence the on-chip communication network should be as small as possible. This ensures that the maximum area can be utilized by the logic while maintaining the performance of the on-chip network. Also, reduction in the logic blocks used in FPGAs has a direct impact on the power consumption and the timing [1]. The central component of an NoC architecture is a router and hence it is prudent to make its area smaller. The network area can be reduced as follows.

- Using a simple router supporting complete functionality, without sacrificing the performance.
- Reducing the number of routers, without reducing the number of communicating logic cores.

This paper proposes to use the second strategy by introducing multiple local ports (LP). We present a modified router architecture and analyze the issues involved in the use of the proposed approach for NoC design in FPGAs.

Mapping: Mapping of cores onto such complex multi-local port routers presents a great challenge. It is no longer just a simple nearest neighbor or shortest path finding algorithm. Hence, we present an algorithm which maps the input task graph optimally, minimizing the overall execution time. This is an exhaustive search algorithm and is guaranteed to find the optimal configuration and hence a formal proof is redundant. For a given set of constraints and objectives, the algorithm finds the optimum number of routers, the configuration of each router, the optimum mesh topology and the the best possible mapping of cores onto the NoC architecture. The algorithm effectively automates the NoC design cycle by finding the optimum mesh topology and the final mapping for the given task communication graph. Ideas explained in this research, though being restricted to FPGAs, can be extended and applied to ASICs. To summarize, the contributions of the paper are,

1. An architecture for a multi local port router capable of handling multiple logic cores simultaneously, without sacrificing the performance.
2. An algorithm to find the optimum mesh based NoC configuration (using multiple local port routers) and the final mapping, reducing the overall execution time.

The rest of the paper is organized as follows: Section 2 describes related work. The architecture overview, the merits and issues involved in the proposed approach are given in Section 3. We present a mapping algorithm for the proposed architecture in Section 4. The experimental results for various benchmarks are analyzed in Section 5. Section 6 presents the conclusion and future work.

3-9810801-0-6/DATE06 © 2006 EDAA

2 Related Work

For reconfigurable computing platforms, several designs have been reported to reduce the size of router [1, 2, 5]. The authors present the smallest router called *LiPaR* [1] for FPGAs, with no sacrifice in the performance. It is a parallel router capable of establishing connections between various ports, simultaneously. Since, the connections can be established without any clock penalty, we obtain this router [1] to experiment our strategy. In the literature, we have different mapping strategies for mesh networks [4, 6, 11]. In the above works, the authors use shortest path routing (non XY routing schemes) to minimize cost. In this context, the authors make an over-simplistic assumption about the router design and its capability and do not discuss the overheads involved in the design. Router overheads cannot be ignored, as it affects the final system performance.

To the best of our knowledge, this is the first work to propose a multi-local port router design for regular mesh networks with XY routing in FPGAs and an optimum mapping onto regular mesh architectures having routers with multiple local ports. We design the multi-local port router (referred as multi-port router, interchangeably) and test the approach based on the constraints for FPGAs.

3 Architecture

In an Network-on-Chip (NoC) design, systems are built by integrating different design cores in a particular defined fashion. The term *Design Core* refers to Intellectual Property (IP) cores that are pre-built and verified for functionality, performance and other constraints.

Topology: We choose the mesh topology in this research because of the following advantages.

- The two-dimensional mesh topology is reported to be efficient in terms of area and power [4].
- Mesh topology is best suited for an FPGA because of lesser routing overheads and reduced use of expensive global wires.
- Due to pin limitations and memory restrictions in an FPGA, IP cores have to be spread across the FPGA. In a mesh style, cores can be distributed effectively complementing the above requirement.

Routing and Flow Control: We use the XY routing and store-and-forward flow control. XY routing is one of the efficient forms of routing for a mesh based NoC. It imposes lesser overhead on the part of a router, as the decoding logic for routing the packets is simple. Store-and-forward, though having some buffer requirements, can result in increased channel utilization and a simple router [1].

Modified Architecture: LiPaR router has an 8 bit header to identify the X and Y coordinates of the NoC system. We modify the router header and decoding scheme to enable the router handle multiple logic cores, at the same time. A part of the header is reserved to identify the local port number (LID) (Figure 1(a)). The address part (A) stores the X and the Y co-ordinates of the destination router.

Adapted Decoding Logic: XY routing scheme is followed till data packet reaches the destination router. When the packet reaches the destination router, the LID is used to identify the local port to which the packet is addressed. Based on LID, the cross point matrix (made of Multiplexer (Mux) and Demultiplexer (Demux)) select signals are appropriately set. This establishes the transfer to the correct local port, when more than one local ports are available. But for the extra decoding logic (to correctly select one among many logic (local) ports), the rest are same as the traditional mesh based NoC system. Since there is no clock penalty in the establishment of connections, this approach does not degrade the performance. Figure 1(b) shows the block diagram of a 4 local port (LP) router. We see the 8 connections between different input and output channels that can be established simultaneously. As a general case, in an n LP router with 4 directional ports, $n + 4$ connections can be established simultaneously. This is possible because an arbiter is present at each of the output channels (no central arbiter) in LiPaR [1] and the connections can be established independent of other channels.

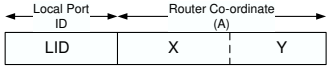
We implement an important design optimization on part of Network Interface (NI) design. If a node has to send data to multiple nodes (out-degree > 1), we give precedence of sending based on the distance of the receiving node. This increases the pipelining rate of flow and serves to reduce the overall delay by abstraction. It is to be noted that there is no additional overhead on part of the header of each packet. Also, there is no reduction in the total number of addressable logic IP cores. This scheme mainly aims to replace the inter router channel communication with intra-channel communication. This scheme can be extended to wormhole flow control. Also, this scheme will help to optimize the buffer size of each channel in each router.

3.1 Architectural Advantages

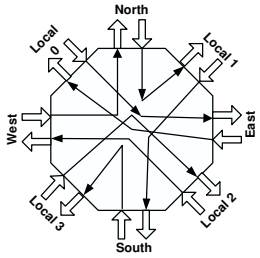
Area Reduction: We observe an average area savings of 36%, with maximum savings of 47.5% in a 4 LP version in Xilinx $XC2VP30$ Virtex II Pro FPGA (Figure 1(c)). The huge area savings are due to the fact that for every single LP router that is removed (and the corresponding logic core added to a multi LP router), we save upon 8 channel buffers (of 4 directional ports) and the associated decoding and routing logic. Furthermore, we save on the routing area between single LP routers.

Power Savings: LiPaR consumes $824.25mW$, out of which $797.5mW$ is the quiescent power [1]. Therefore, area reduction in term of the number of routers significantly reduces the number of slices consumed, thereby, reducing the power consumption considerably.

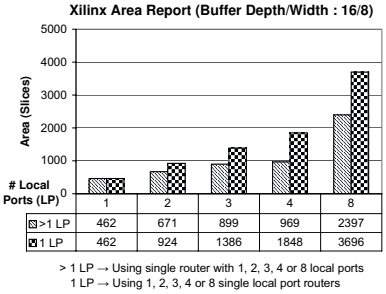
Congestion Reduction: Congestion is an inherent problem in a shared network. As the intercommunication between the cores increase, there can be *heavy congestion as a lot of paths will be shared*. Multi LP router achieves communication in the form of intra-channel communication replacing inter-router-channel communication. This in turn reduces the number of shared paths, thereby, reducing the overall congestion in the network.



(a) Modified Header Packet



(b) 4 LP Router *(having 8 Parallel connections)*



(c) Xilinx ISE Synthesis Results

Figure 1: Modified Router Architecture

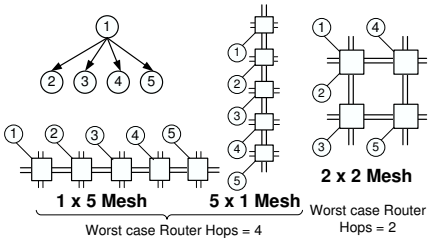


Figure 2: Mapping of Five cores

Transit Time Reduction: The transit time of the packets is one of the performance bottlenecks in an NoC. For example, consider a case where a core is communicating with more than four cores (out-degree > 4). The fifth receiving core, at the best, can only be reached in 2 router hops. Increased number of router hops increases the overall time. With use of multi LP routers, the total number of hops are significantly reduced. For every local port added, we do away with two complete channel hops, which reduces the number of clock cycles by $2 \times \#flits$. This is because the inter-router channel transfers are now replaced by intra-channel transfers and this effect is more pronounced in routers using store-and-forward type of flow control.

Improving NoC Design by reducing router count: Consider the case where five cores are communicating in a particular fashion (Figure 2). With traditional mesh design, we have either a 5×1 or a 1×5 mesh, having a worst-case of 4 router hops. But, with the use of just one 2 LP router, we can implement a 2×2 design, reducing the worst-case router hops to 2. This is a very effective strategy and with large number of cores and complex interconnection patterns, the savings will be much larger.

3.2 Design Issues

The proposed approach gives gains in terms of area, power and performance. Intuitively, a single router with n local ports seems to be the best option. But, following are some issues that may limit the maximum number of local ports in the NoC design.

Critical Path: Addition of more local ports to a single router increases size of the decoding logic of the router. Also, increased interconnect count and length within a router are inevitable. This can impact the critical path of final design. But, we observe that the variation in the timing due to addition of local ports (till 9) to the routers in $XC2VP30$ FPGA is not very significant. The router designs are reported to be operating close to $90MHz$. With increased local port count, we can expect a larger variation.

Input/Output (I/O) Constraints: Pins are limited in an FPGA. I/O requirements dictate the placement of cores. Hence, to avoid performance degradation due to larger interconnects, routers having lesser LP are preferred.

Buffer: Buffer requirements of router and logic favor the placement of cores and its corresponding router near the bRAMs of Xilinx FPGA, thus favoring smaller LP routers.

Routing Resources Congestion: A larger LP router needs larger number and complex form of interconnects, which may create problems for the FPGA Place-And-Route (PAR) tool. But, for a 9 LP router, PAR was successful.

Arbitration: Our router design has an arbiter at each of the output channel. The access grant of an output channel is given inside a single FSM state (`if-then-else` construct) and hence has a fixed priority service scheme. With this approach, there are no extra cycles wasted for arbitration. But the flip side is that when there are multiple channels requesting the same output channel, the fixed priority scheme may lead to an unfair service.

Based on the above factors, we investigate the alternate router architecture with multiple local ports against the traditional NoC implementation. It is to be noted that except for the critical path constraint (which affects frequency of operation), others dictate the upper bound on the search space of the algorithm.

4 optiMap: The Mapping Algorithm

Mapping of cores onto a non-traditional NoC architecture using multi local port routers, is not a straight forward process and presents a great challenge. Under certain constraints, traditional NoC mapping may be better in some cases. Even in that situation, there can be a better mapping compared to the nearest-neighbor based mapping. Let us consider the example shown in Figure 3, to give a flavor of mapping on multi local port routers. Here we choose a 1×2 mesh each having a two local port router. Let us consider a simple cost function where we sum number of hops. We increment the cost for each channel access, to get the final cost. We observe from Figure 3 that the cost is reduced

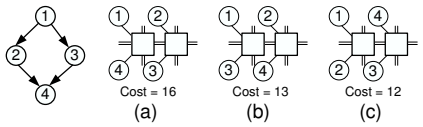


Figure 3: Mapping using with two local ports

Algorithm 1: optiMap Algorithm

Input: Given a system level task graph, $G(T, E)$ with n logic cores
Input: Input the placement constraint, routing constraint, I/O constraint and buffer constraint
Output: The optimum NoC configuration
Analyze the constraint file and set the upper bound of the router configuration, thereby, defining the search space of the algorithm
repeat
  γ: Generate the set of all possible partition configurations for n for the defined search space
  Γ: Generate all possible combinations of γ
  foreach *Partition configuration in* Γ **do**
    Define the # of routers for the present configuration
    Define the configuration of each router
    Υ: Generate all possible mesh connection topologies for the partition configuration
    foreach *Mesh and Partition configuration in* Γ, Υ **do**
      φ: Generate all possible ways of mapping of cores
      foreach *Mapping of given Mesh and Partition configuration in* Γ, Υ, φ **do**
        foreach *Edge in the Task graph* **do**
          Identify the source router (i_1, j_1) and the destination router (i_2, j_2)
          κ: Decompose in terms of the intra/inter-channel communications between (i_1, j_1) and (i_2, j_2) using XY routing scheme
          Update the communication times of all channels
          Calculate the Queue times (Q_t) at all channels and update the transit/arrival times appropriately
          $C_f = \alpha \times$ Total Execution Time of cores (transit, core execution and arbitration times) $+\beta \times Q_t$
          if $C_f <$ *Best.Config.Cost* **then**
            Best.Config $\leftarrow$ Current.Config
          end
        end
      end
    end
  end
until *All the configurations are evaluated*

by 25%, with proper mapping. This represents the simplest of cases, where the queuing effect at channels and other cost parameters are not considered. For a complex NoC system, a mapper that finds an optimum NoC configuration is required. Hence, we present a mapping algorithm, capturing the effects explained in Section 3.1 and 3.2. The algorithm does an exhaustive search and is guaranteed to find the optimal configuration and hence a formal proof is redundant. The mapping algorithm can be easily extended to incorporate additional cost parameters, thereby, giving a multi-objective based mapping algorithm. The algorithm efficiently maps the cores of the given task graph for various objectives and constraints. Also, the algorithm finds the optimum number of routers, the configuration of each router and an optimum mesh topology for a given task graph. We test the algorithm on a wide variety of benchmarks and report the results.

Problem definition: Given a system level task graph, $G(T, E)$ and the set of constraints, find the optimum NoC configuration, that is, find the mesh topology, number of routers, configuration of each router and the final mapping of logic cores, reducing the cost function.

In Algorithm 1, the problem of finding the different partition configurations for the given task graph corresponds to the partitioning of an integer problem. We permute and find

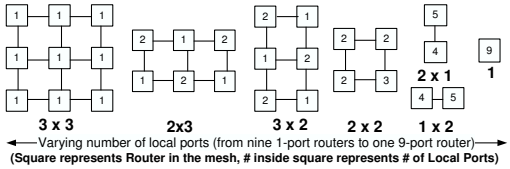


Figure 4: Mapping Search Space for Cores

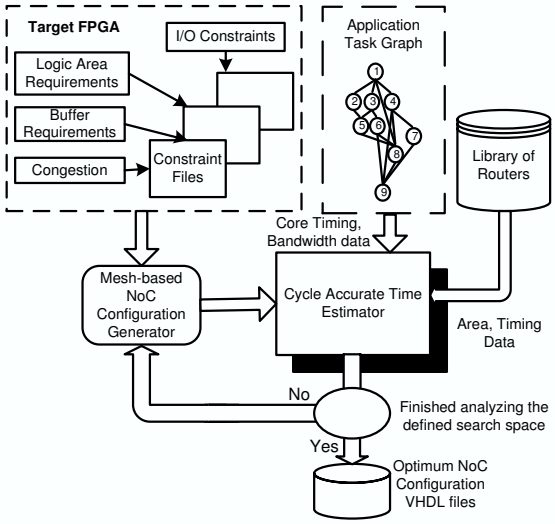


Figure 5: Algorithm Flow of optiMap

all possible ways of partitioning the value n (which is the number of nodes in the graph) to get the set Γ. Based on Γ, we define the number of routers and the configuration of each router, based on the constraints. We generate all possible mesh topologies for all the partition configurations to get set Υ. After this, we map the cores onto each of the configuration in all possible ways (φ) and evaluate the cost function. Figure 4 shows the search space of the mapping algorithm for a specific case of 9 cores. The cost function is the weighted sum of total execution time (which includes transit time, core execution time and arbitration time) and queue delay (due to simultaneous access of paths/channels). The factors explained in the Section 3.2 like routing density, placement constraints, I/O constraints, buffer requirements, etc. dictate the upper bound of the search space, that is, the maximum number of local ports that can be added to a router. We build a cycle accurate simulator to calculate the execution and queue times, for the overall data transfer. In other words, we find the overall execution time of the given task (application). In the end, the algorithm outputs the best NoC configuration (including the best partition, configuration of each router, the mesh topology, and the optimum mapping of the cores). In short terms, we are doing an exhaustive search of all possible NoC configurations (Figure 5). Analytically, for a task graph with n nodes, the total number of configurations analyzed is (2^{n-1} partitions) $\times$ (# mesh configurations) $\times$ ($n!$ ways of mapping n cores). The # of mesh configurations (k) for a given value of partition (p) is k += t (where t=1, if (p/i)=0; else t=0;), where i = 1,...,p.

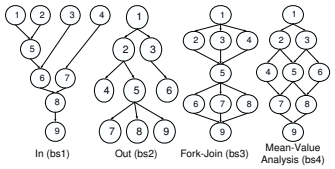


Figure 6: Basic Task Graphs [9]

5 Experiment Results

A typical SoC design is described as a task communication graph. The application is described as a Directed Acyclic Graph $G(T, E)$ (referred to as *Task Graph* hereon), where T represents the vertices (tasks) and E is the set of directed edges describing the precedence, the dependence, the timing and the bandwidth constraints in the task graph. Most applications, including Fast Fourier Transform, Discrete Cosine Transform (DCT) and Auto Regressive Filter, can be described in the form of task graphs.

Benchmarks: Kwok and Ahmad [9] use a set of different task graph types for studying various scheduling algorithms for multiprocessors, including Out-Tree, In-Tree, Fork-Join and Mean-Value-Analysis. Figure 6 shows the basic graph structures [9]. These basic task graph types represent high level task structures that are commonly encountered in parallel applications. We use a tool called Task Graphs for Free (TGFF) [13] to generate the basic task graphs having nine nodes ($bs1, bs2$) by fixing the in-degree, out-degree, and dependence width/depth. The fork-join ($bs3$) and mean-value-analysis ($bs4$) graph structures are manually created, as TGFF was not capable of generating these graph structures. We then write a C++ program that takes in the set of basic task graph structures that were already generated and outputs an application task graph, G. The application task graph G is formed by a random combination of the basic graph structures, by varying the dependence degree, the width and the depth across different levels of nodes. It is to be noted that most of the application task structures can be generated using the combination of the basic graph structures and hence the program generates a variety of the system task graphs. We set the upper limit on the number of tasks in the graphs to nine and develop a set of 18 benchmarks (including the 4 basic graphs). We fix the number of nodes at 9 because it represents a 3×3 NoC system, a typical case. We analyze this system with 9 single LP routers, against the new scheme. The algorithm presented in Section 4 explains the strategy to find an optimum mapping in the new scheme. Figure 7 shows the benchmark set that were generated using our tool. We omit the specific details of the nodes and edges of each of the benchmarks due to space limitations. The benchmarks $b1$ and $b2$ represent the simple graphs. The cases $e1$ and $e2$ have large out-degree. Benchmarks $p1$ - $p4$ represent packed structures [13], while $r1$ and $r2$ are two random cases. Benchmarks $pa1$ and $pa2$ have high degree of parallelism. LU Decomposition (lu) and Laplace Equation Solver(les) represent the real time examples [13]. The set of eighteen benchmarks cover a wide variety of graph types and the proposed methodology can be easily extended to cover applications like MPEG4, JPEG and DCT.

Experiment Platform: We use the $ML310$ board provided by Xilinx to functionally verify the various versions of the stand alone router and the NoC system. The test board has Virtex II Pro family ($XC2VP30$) of FPGA [15]. We use the Xilinx ISE $6.2i$ [15] to synthesize the designs. Modelsim $5.8c$ [12] is used to simulate the model and generate activity data of the Placed-And-Routed (PAR) models. The FloorPlanner tool of the Xilinx ISE $6.2i$ is used to implement placement constraints on the NoC system. We use the XPower tool of the Xilinx ISE $6.2i$ to get the power estimate values of the designs. The mapping algorithm is written in C++ using the Standard Template Library (STL) vectors and dynamic arrays. The optiMap algorithm is executed in a Sun-Blade 1000 workstation having dual processors operating at $750MHz$ and $2GB$ RAM. The average execution time varied between 5 and 6 hours. The large execution time is due to the fact that the algorithm does an exhaustive search of a very large design space (refer Section 4), to find the optimum NoC configuration.

Analysis of the Results: In this research, we make the algorithm to search the entire search space, while caching the results for different constraints. We perform this to present a broad picture of the effectiveness of the algorithm. For experimentation purposes, we assume equal execution times for all nodes in the graph (communication time from NI to logic is also abstracted into this lumped time) and equal bandwidth constraints for the edges. We discuss the results of randomly selected benchmarks in each of the cases below.

Upper Bound on # Routers: It is to be noted that for a case where the upper bound on the number of routers is k, the optimum configuration can contain combination of router(s) with $\leq k$ local ports and the timing of the maximum LP router used is taken as the timing of NoC configuration. It is seen in all of the benchmarks that reduction in the number of routers increases performance. Interestingly, from the Figure 9(a), we see that having 8 routers is beneficial than having 7 routers. This is due to the fact the maximum number of hops (diagonal length) is reduced from 6 to 4. Overall, we infer that it is better to reduce as many routers as possible and introduce multi LP routers.

Upper Bound on # LP: In Figure 9(b), we fix an upper bound on the number of local ports (timing data in Table 1). We see that increasing the number of local ports increases performance. But, it is seen in most of the cases that the 7 LP version is not better than the 6 LP version. Also, for the specific case of $p4$, the 8 LP version is better than the 9 LP version. The best NoC configurations obtained by the optiMap algorithm for lu, les and $p4$ (for different upper bounds on number of LP) are shown in Figure 8, along with the single LP versions. It is to be noted that the algorithm finds the optimum NoC configuration even for the single LP router version. Interestingly, in Figure 8(c), for a 3 LP upper bound case, the optimum NoC configuration uses 4 routers (instead of three 3 LP routers). This is due to the fact that the algorithm tries to reduce the overall hop count (thereby reducing overall time) and the inter-communication pattern of $p4$ dictates this configuration. We observe an average performance

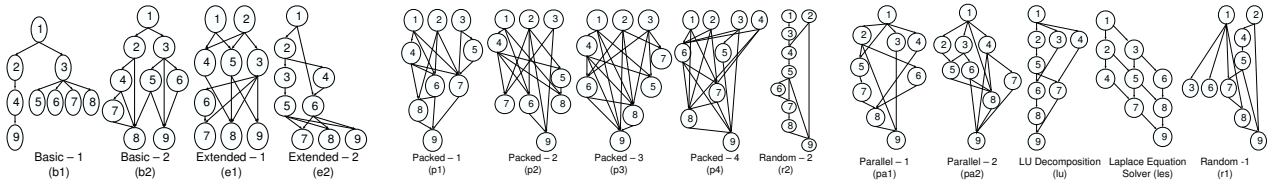
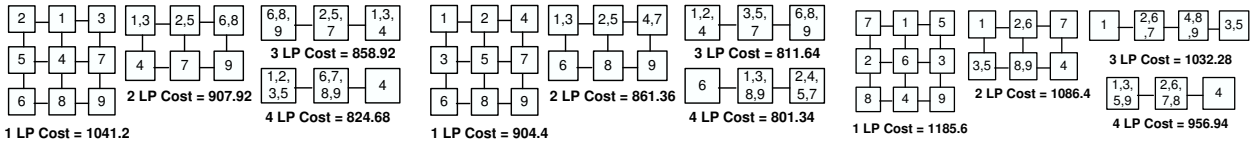
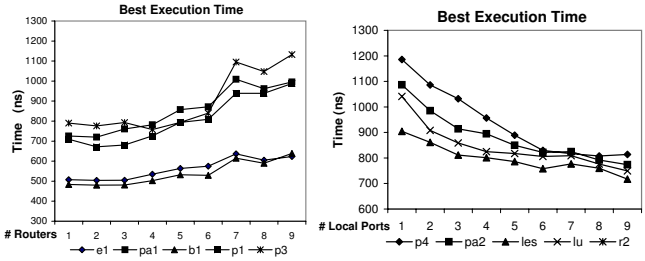


Figure 7: Benchmark Set (lu,les [9])



(a) LU Decomposition *(lu)* (b) Laplace Equation Solver *(les)* (c) Packed-4 *(p4)*

Figure 8: Optimum NoC configuration with varying upper bound on # local ports(LP) - the mapped cores are inside square (router) *(shown till # LP = 4, due to space constraints)*



(a) Upper bound: # routers (b) Upper bound: # LP

Figure 9: Overall Execution Time (ns)

Max local port count	Benchmark Execution Time (ns)				
	p4	pa2	les	lu	r2
1	1185.6	1086.8	904.4	1041.2	1451.6
2	1086.4	985.52	861.36	907.92	1241.6
3	1032.28	914.08	811.64	858.92	1158.36
4	956.94	894.7	801.34	824.68	1089.2
5	889.28	849.58	786.06	817.82	1071.9
6	829.92	821.94	758.1	805.98	1077.3
7	817.02	825.03	776.97	809.01	1081.35
8	808	792	760	776	1024
9	814.06	773.76	717.34	749.58	1023.62

Table 1: Execution Time - Upper bound on # LP

improvement of 30% across the set of benchmarks, compared to a single LP design. To summarize, the optimum number of routers and the router configuration (and the final mapping) depends on the given application task graph and the maximum frequency of operation of the NoC. In all the benchmarks, it is observed that choosing a single LP router design is never optimal, thus, validating our proposed approach.

6 Conclusion & Future Work

We present an approach of incorporating multi local port routers in regular mesh based Networks-on-Chip design, in FPGAs. We analyze the merits and the constraints involved in using such a design methodology. We present an algorithm that finds the optimum NoC Configuration. We experiment with a wide set of benchmarks and report the results. The results show significant area savings and improvement in performance, thereby, validating the proposed approach. As a future work, we are developing a heuristic based algorithm to quickly output the NoC configuration, based on the results and insight got from the current work.

Acknowledgements

We acknowledge the contributions of Prasun Bhattacharya in obtaining the various VHDL models.

References

[1] Balasubramanian Sethuraman et al. LiPaR: A Light-Weight Parallel Router for FPGA-based Networks-on-Chip. In *15th Great Lakes Symposium on VLSI (GLSVLSI'05)*, 2005.

[2] C.A. Zerferino et al. ParIS: A Parametric and Scalable Network on Chip. In *SBCCI'2004*, 2004.

[3] W. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *DAC*, 2001.

[4] D.Bertozzi et al. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *Parallel and Distributed Systems, IEEE Trans. on*, 16(2):113–129, 2005.

[5] Fernando Moraes et al. A Low Area Overhead Packet-switched Network On Chip: Architecture and Prototyping. In *IFIP VLSI-SOC 2003*, pages 318–323, 2003.

[6] J. Hu and R. Marculesu. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In *DATE'03*, pages 688–693, 2003.

[7] International Sematech. International Technology Roadmap for Semiconductors. In *http://public.itrs.net*, 2002.

[8] N. Kavaldjiev and G. J. Smit. A survey of efficient on-chip communications for SoC. In *PROGRESS 2003 Embedded Systems Symposium*, October 2003.

[9] Y.-K. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *Parallel Distributed Systems, IEEE Transactions on*, 7(5):506–521, 1996.

[10] L. Benini and G. De Micheli. Networks on Chips: A New SOC Paradigm. In *IEEE Computer*, pages 70–78, Jan 2002.

[11] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Euromicro Symposium on Digital System Design*, 2003.

[12] MentorGraphics Inc. http://www.mentorgraphics.com.

[13] R.P. Dick et al. TGFF: Task Graphs For Free. In *6th International Workshop on Hardware/Software Codesign*, 1998.

[14] Shashi Kumar et al. A Network on Chip Architecture and Design Methodology. In *Annual Symposium on VLSI'2002, IEEE CS Press*, pages 105–112, 2002.

[15] Xilinx Inc. http://www.xilinx.com, 2004.