A Low-Power Crossroad Switch Architecture and Its Core Placement for Network-On-Chip

ABSTRACT

As the number of cores on a chip increases, power consumed by the communication structures takes significant portion of the overall power-budget. The individual components of the SoCs will be heterogeneous in nature with widely varying functionality and communication requirements. The communication topology should possibly match communication workflows among these components. In this paper, we first propose a novel interconnect architecture for SoC, which uses *crossroad switches* to dynamically construct a dedicated communication path between any two cores. We then present a design methodology for constructing network on chip (NoC) for application-specific computer systems with profiled communication characteristics. We design a core placement tool, which automatically maps cores to a communication topology such that the total communication energy can be minimized. Experimental results show that the design methodology can generate optimized on-chip networks with fewer resources than meshes and tori, and the power saving approximates to 40%.

1. INTRODUCTION

As parallel chip architectures scale in size, on-chip networks have becoming the main communication architecture, replacing dedicated interconnects and shared buses. NoC architectures have to deliver good latency-throughput performance in the face of very tight power and area budgets. These trends make on-chip network design to be one of the most challenging and significant design problems.

In [11, 13], they design mapping algorithms for regular topologies, which are suitable for interconnecting homogeneous cores in a chip multiprocessor. However, varied core functionality, core size and communication requirements are involved in recent SoCs designs. If a regular interconnect such as mesh or tori is designed to match the requirements of few high communicative components, it maybe waste much resources with respect to the needs of other components. So it is attractive for most current SoCs to use irregular topologies (e.g. Figure 1(d)) to design dedicated high-speed links between high communicative cores.

Some applications have high point-to-point communications. If it is not well controlled, large intercommunications across switches could consume large significant energy of NoCs. Take a core flow graph for application suite in Figure 1(a) as an example. Each node in the core flow graph represents a core, while each weighted edge represents the communication frequency between a pair of cores. Two possible NoC topologies for the core flow graph are shown in Figure 1(c) and Figure 1(d). The topology in Figure 1(c) is a initial core placement, and there are 27 intercommunications. The topology in Figure 1(d) is an improved core placement, and there are 15 intercommunications. We can observe that the power consumption of random placement will be larger than the well-controlled placement, because of the high frequency of inter-communications.

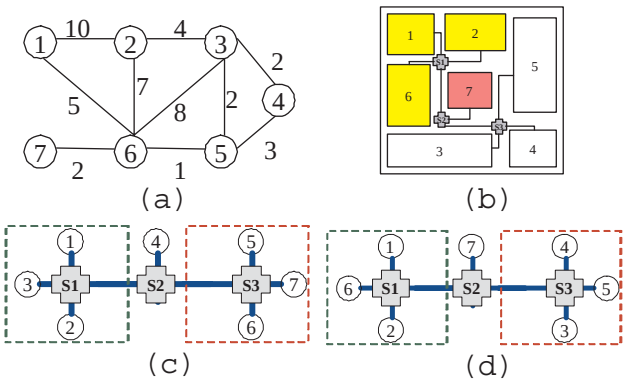

Figure 1: (a) Core flow graph. (b) Final NoC block (c) Initial core placement (d) Improved core placement

This paper makes two contributions as follows:

- First, we propose a novel bus architecture named *crossroad interconnect architecture*. The architecture consists of several communication blocks, which include a group of cores, four way bus lines and a local crossroad switch. There is also a global arbiter to coordinate the communications between crossroad switches when necessary. By coordinating those arbiters, we can organize several algorithms either to enhance the performance or to minimize the power consumption for communications.
- Second, we also present a design methodology and a core placement tool to automatically build application-specific communication topologies, which supports low intercommunications for well-known communication patterns. It starts from application specifications, continues through the mapping of the application onto topologies and ends up with selection of a topology.

The rest of the paper is organized as follows. Section 2 summarizes the related work. In Section 3, we describe the proposed crossroad interconnect architecture and characteristics of this design. Section 4 describes the design methodology and the power aware core placement algorithm. We explain the experimental environment and show the results of our work in Section 5. Finally, we summarize our findings in Section 6.

2. RELATED WORK

VLSI design for power optimization to satisfy the power budget is an important research issue [2]. The bus topology can be changed so that different segments can be split for the purpose of reducing the power consumption [4, 8, 15]. The early works in [5] pointed out the need of more scalable architectures for on-chip communication and, therefore, to progressively replace shared busses with on-chip networks.

Many NoC architectures have been proposed in the literature so far, but in most cases, the design methodologies and tools are still at the early stage. The problem of mapping cores onto NoC architectures is addressed in [1, 10, 11, 13, 14]. In [10], a branch-and-bound algorithm is used to map cores onto a mesh-based architecture with the objective of minimizing energy and satisfying the bandwidth constraints of the NoC. Murali and De Micheli [13] presented an algorithm that maps cores, or components of a SoC, onto a mesh NoC architecture, minimizing the average communication delay. Hu and Marculescu [11] presented an algorithm for mapping IPs onto a generic regular NoC architecture consisting of a network of tiles, each consisting of a processing core and a router. In [3, 9, 11], the assignment and scheduling of tasks onto cores were performed first, then they apply profiling to derive the communication patterns of the application used in the topology synthesis and routing algorithms.

However, most of these researches were focused on regular topologies (e.g. mesh, tori, hypercube), which require large scale redundant switches in order to meet application requirements of burst point-to-point communication. These regular topologies do not fit for application-specific SoC developments, because they are sometimes over-designed. It is also hard to automatically construct optimized NoCs by a tool if the switches are heterogeneous.

3. CROSSROAD INTERCONNECT ARCHITECTURE

In this section we present a brief description of the proposed bus architecture, *Crossroad Interconnect*, in the aspect of switches, links and networks. The crossroad interconnect architecture is similar to the segmented bus architecture [4]. However, it isolates the requirements of long data swing so as to reduce power consumption. The key idea of the crossroad interconnect architecture is to partition all cores on chip as well-organized irregular topologies and use crossroad switches to dynamically control active paths for those connections needed between any two cores. In addition to power optimization, our crossroad interconnect architecture also gives better performance and parallel communication by providing two separated virtual paths for different crossroad blocks at the same time.

3.1 The Proposed Architecture

The basic communication unit of the architecture is the *crossroad communication block (CCB)* as shown in Figure 2(a). A communication block comprises a crossroad switch with an arbiter, a group of cores, and four bus directions for data transmissions. More than one route can go through the switch at the same time as long as its input and output ports in the switch are not occupied by another route. Every crossroad switch only takes care of the requests from four different ways (up, down, left and right), so the control of the crossroad switch is independent and scalable, and the design complexity is low. In this case, users can construct different structure styles (topology) of the crossroad communication architecture based on the requirements (power, performance, area) of their SoCs.

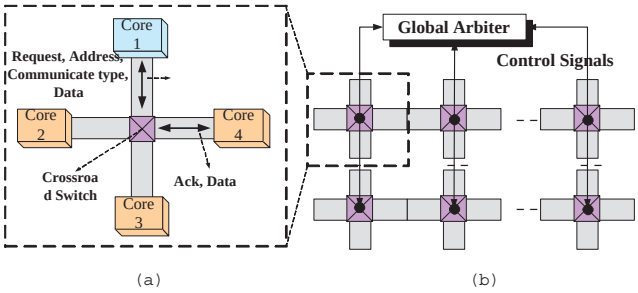

Figure 2: (a)Crossroad communication block (CCB) (b)The crossroad interconnect architecture

The overall communication architecture is shown as Figure 2(b). Several communication blocks can be connected to each other to construct a large scale communication network. Only the edge sides of the crossroad switches can be connected with cores. If two switches connect to each other, the bus line between the two switches cannot plug any cores, because it only provides data path between two communication blocks. The global arbiter is used for special purposes, such as deadlock-free algorithms, routing algorithms, contention-free algorithms, etc. We can design several algorithms to coordinate all switches to enhance the performance or low-power communication depending on the characteristics of different applications. For example, adaptive routing can dynamically select the exchange path between two modules to solve the interconnect contention problems. The overall advantages can be summarized explained in the following subsections.

3.2 Fully Configurable

The switch in a communication block only takes care of the control and the data exchanges between four ways. When a master requests a slave in the local block, it passes the signals to the target slave. If a master requests a slave in another remote block, the local arbiter will act as a master in the remote block and sends the request to the remote switch. Every crossroad switch only cares about the requests from four different ways, so the basic element structure and control can be very regular and simple.

When two switches connect to each other, one of the two blocks can be viewed as a master or a slave module for another block. And this feature makes the overall architecture more scalable and highly configurable. Our design can easily combine the communication block to make up our irregular network topologies according to the different application characteristics, as shown in Figure 3. We can employ with a placement algorithm to explore the best solutions of the intercommunication for purposes of high performance, low power consumption and low cost.

3.3 Power Optimization by Localization

In shared-bus architectures, every data transfer is broadcast, meaning the data must reach each possible receiver at great energy cost. Because the power consumption of each bus segment is proportional to the number of devices connecting to the segment in splitting bus architecture. In our design, only one sender, one receiver and one or more than two switches are involved at each data communication, data exchanges among devices will result in minimum power consumption. Due to the full programmability, the placement of all cores will be optimized by profiling the communication traffic characteristics of applications. We can connect high communicative cores into several CCBs to provide separate high-frequency blocks where data are total independently transmitted, as shown in Figure 3. In this case, it can save more energy consumption due to the reduction of long-distance communications.

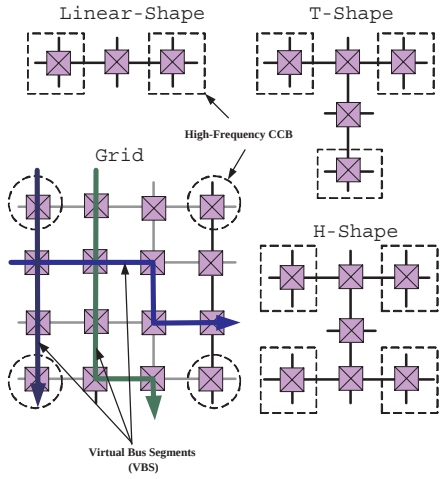

Figure 3: Interconnect topologies

3.4 Better Communication Parallelism

We design a "overpass" mechanism that can construct multiple segments to let different masters communicate with their target slaves at the same time, as shown in Figure 4(a). Figure 4(b) shows the three possible combinations of parallel communications in a local communication block. If those master-slave pairs are different, they may use different channels to communicate in the crossroad bus architecture. This behavior is suitable for multiprocessor or multithreading, where each processor or thread can work in their local regions and communicate to other regions by coordinating crossroad switches in each block. Because the architecture may provide separate *virtual bus segments (VBS)* where data are total independently transmitted, as shown in Figure 3, it results in more communication parallelism than conventional shared-bus architectures.

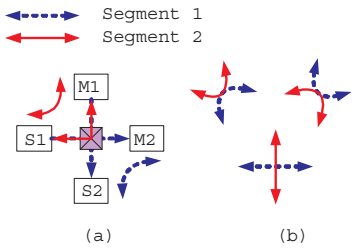

Figure 4: Three available combinations for parallel communications

4. DESIGN METHODOLOGY

Crossroad NoC is a structured interconnect architecture such that it can be integrated into a design flow easily, as shown in Figure 5. First, the communication characteristics between cores can be derived by profiling embedded applications. Then, we can construct suitable communication topologies according to the profiling results to meet specific requirements (power, performance, area). Using the proposed partition algorithms, we can decide which cores should be put in the same CCB. The hardware specification specifies the cores (e.g. processing and storage elements) that are to be connected in the system-on-chip design as well as the requirements on bandwidth, latency, etc. put on the communication hardware to support the necessary transfers. The optimized code can then be used in the behavioral simulator to ensure correctness and to generate a number of different benchmarking information to make sure that the specification is fulfilled. Designers can get the area, performance and power estimations of the crossroad communication architecture to estimate their design. Finally behavioral verification by the structural description is used as a part in the integration of the system-on-chip design to achieve the final implementation.

Because crossroad interconnect architecture is a simple structure, it is easy to automate the design and even build a library of IP cores for crossroad interconnects with various self-routing encoding and performance. Also the crossroad interconnect architecture gives more space on performance, power, and area along with other useful features such that designs can be more flexible.

In this section, we present an algorithm to support a power-aware placement for the crossroad architecture. We will define the system model in Section 4.1, and present the core placement algorithm in Section 4.2.

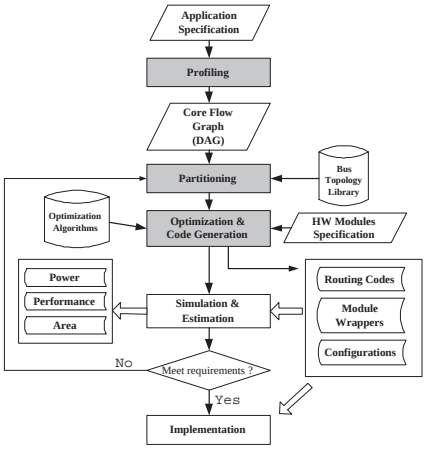

Figure 5: Design flow for crossroad NoC

4.1 System Model

Hu et al. [11] proposed a model for power consumption of tiled-based NoC architectures. The average energy consumption of sending one bit of data from tile t_i to t_j is:

$$E_{bit}^{ti,tj} = n_{hops} \times E_{S_{bit}} + (n_{hops} - 1) \times E_{L_{bit}}, \qquad (1)$$

where $E_{S_{bit}}$, $E_{L_{bit}}$, n_{hops} represent the energy consumption of a switch, the energy consumption of interconnection wires and the number of switches the bit passes, respectively. The basic idea of the paper is to group high communicative cores into several CCBs, as shown in Figure 3, such that

data exchanges among cores will result in minimum power consumption calculated by Equation 1. To formulate this problem more formally, we define the following terms:

Definition 4.1. The *Core Flow Graph (CFG)* is a undirected graph, G(V,E), where each vertex $v_i \in$ V represents a core and the directed edge (v_i, v_j), denots as $e_{i,j} \in$ E, representing the communication between the cores v_i and v_j . The weight of the edge $e_{i,j}$, denoted by $flow_{i,j}$, represents the communication flow or communication frequency between v_i and v_j .

Definition 4.2. The *Switch Topology Tree (STT)* is a tree, $T(V, E)$ with each vertex $v_i \in V$ representing a switch with at most 4 degrees and the edge (v_i, v_j), denoted as $e_{i,j} \in E$, representing the communication between the switches v_i and v_j . The weight of the edge $e_{i,j}$, denoted by $flow_{i,j}$, represents the communication flow or communication frequency between v_i and v_j .

In order to achieve low-power design for NoC topologies, we have to minimize the number of communications traversed through bus lines and switch hops for each pair of cores. For this purpose, we define communication distance as follows:

Definition 4.3. The *Communication Distance (CDist)* between two switches s_i, s_j, denoted CDist(T, s_i, s_j), on a switch topology tree T is the number of switch hops and edges traversed from s_i to s_j on T, where

$$CDist(T, s_i, s_j) = \sum_{e \in Path_{s_i,s_j}} E_{L_{bit}} + \sum_{switch \in Path_{s_i,s_j}} E_{S_{bit}} \quad (2)$$

For example, as shown in Figure 1(d), cores 1 and 2 are connected to the same switch, but separated from others by other switches. When core 1 sends a signal to core 2 (intra-block communicaiton), the signal only flows within the same switch, instead of all switches, so this avoids charging or discharging the unnecessary part of the entire NoC system. Based on Equation 1, the power savings can be very significant if most data exchanges are performed by intra-block communication. However, several switches must be involved for inter-block communication, e.g., switch 1, 2 and 3 of Figure 1(d) must pass signals if core 6 sends a signal to core 5. The number of switches to be involved for each inter-block communication depends on the topological relationships among the cores and the NoC architecture. Thus, it is important to organize the bus architecture such that most data exchanges will be performed within CCBs or near CCBs locally as possible. The NoC topology construction problem is defined as follows:

Definition 4.4. Given a Core Flow Graph $G = (V, E)$, the *NoC Topology Construction* problem is to identify a Switch Topology Tree $T(V, E)$ whose total communication cost Cost(T,G) can be minimized. The formula of Cost(T,G) is defined as follows:

$$Cost(T, G) = \sum_{\substack{e(i,j) \in E[G] \\ and \\ s_i, s_j \in V[T]}} weight(e) \times CDist(T, s_i, s_j), \quad (3)$$

where s_i and s_j are the switches connected by core i and core j, respectively.

A construction example is shown in Figure 1. In Figure 1(a), each node in the core flow graph represents a core, while each weighted edge represents the communication frequency between a pair of cores. Two possible NoC topologies for the core flow graph of Figure 1(a) are shown in Figure 1(c) and Figure 1(d). The topology in Figure 1(c) constructed by initial core placement, and its communication cost is ([(10+4) + (2+1)] $\times$ $E_{S_{bit}}$ + [(2+3)] $\times$ ($2E_{S_{bit}}$+$E_{L_{bit}}$) + [(5+7+8+2)] $\times$ ($3E_{S_{bit}}$+$2E_{L_{bit}}$) = $93E_{S_{bit}}$ + $49E_{L_{bit}}$). The topology in Figure 1(d) constructed by the power-aware core placement, and its communication cost is ([(10+7+5) + (2+2+3)] $\times$ $E_{S_{bit}}$ + [(2)] $\times$ ($2E_{S_{bit}}$+$E_{L_{bit}}$) + [(1+4+8)] $\times$ ($3E_{S_{bit}}$+$2E_{L_{bit}}$) = $72E_{S_{bit}}$ + $28E_{L_{bit}}$). Obviously the cost of the initial core placement is large than the cost of the power-aware core placement, because the inter-communication flow is too large. Our key idea is to profile the characteristics of applications and to allocate high communicative cores in CCBs or near CCBs to minimize the $Cost(T, G)$. Figure 1(b), shows the placement result block based on the organization of Figure 1(d).

4.2 Power-Aware Core Placement (PACP)

The objective of the *power-aware core placement* is to minimize the inter-communications between cores such that the power can be saved. In order to find efficient methods, we followed a multi-phase approach, where each phase addresses a limited instance of the general problem. The successive steps are outlined as follows: **core clustering**, **cluster optimizing**, and **physical switch mapping**.

4.2.1 Phase 1: Core Clustering

The first phase to construct a NoC topology is to characterize the hardware structure that can be mapped into a graph, called a "backbone graph" or "switch topology tree." It can be obtained by applying Gomory and Hu [7]. That is, given a weighted and undirected graph, the Gomory-Hu algorithm can find a tree whose communication cost is minimum and the entire process is guaranteed to be finished in polynomial time. Figure 6 gives an example of the Gomory Hu cut tree of Figure 1(a). It can be proven that the tree generated has the minimum Cost(T,G) using polynomial computing time.

Next, we have to collect high communicative cores into the same clusters to minimize the intercommunications. Systematically partitioning the Gomory Hu cut tree into smaller clusters is the key point of the clustering. By the min-cut theorem, we can select the minimum cut to make the GH-tree into two clusters, shown as Figure 7(a). By this way, we can make sure that the flow of the backbone between the two clusters C1 and C2 is minimum. By recursively applying the above process to clusters C1 and C2 until each cluster contains at most 3 nodes, we can finally have the cluster tree as shown in Figure 7(b).

Because each switch can connect at most 3 cores and the other link connects to another switch. We can merge adjacent clusters if there are sufficient empty core connections in one of the adjacent clusters, shown as Figure 7(b). Finally, we can get the cluster tree (Figure 7(c)), which can be used to allocate switches to connect all cores.

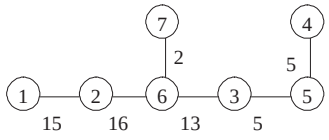

Figure 6: GH-cut tree derived from Figure 1(a)

4.2.2 Phase 2: Cluster Optimization

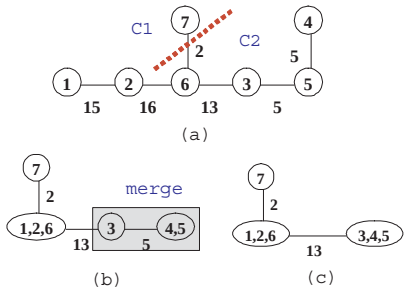

Figure 7: (a) Recursively cut the GH-cut tree into several clusters (b) Merge adjacent super nodes (c) The final cluster tree

In previous phase, the high communicative cores can be mapped into low-intercommunication clusters, however, the flows between clusters may not be minimum. There are some special cases, where clustering results are not the optimal when the weights of edges are too close. We have to optimize these clusters to get more power savings. For example, the Figure 8(b) is the clustering result topology of Figure 8(a), and its flow is 6. However, we can see another topology in Figure 8(c), its flow is 5. This is because that more than two adjacent clusters of the topology in Figure 8(b) can be merged into a cluster to make more intracommunications. We name this behavior as *cluster shifting*. If there are more than two non-fulfilled clusters (less than 3 nodes), we will try to adjacency-pair shift combination one cluster at a time to merge some adjacent clusters into less clusters.

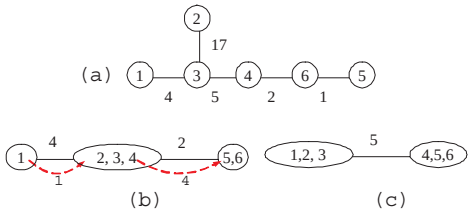

Figure 8: (a)A GH-cut tree example (b)Cluster shifting (c)Results after cluster shifting

4.2.3 *Phase 3: Physical Switch Mapping*

Because a switch only has four connection links, we have to allocate switches to clusters carefully. From the cluster tree, we can regard each cluster as a super node and recursively select three adjacent clusters containing the maximum weight sum as a super cluster. By the way, there will generate several layered super clusters containing at most three clusters. Finally, we assign switches to clusters and super clusters to connect all cores, as shown in Figure 9(a). Next, we merge adjacent switches which are not full connected until no adjacent switches can be merged, shown as Figure 9(b). Figure 9(c) is the final NoC infrastructure. Since the switch just have four way to connect cores or other switches, our topology looks like 3-nary tree structure, as shown in Figure 10. The internal nodes are switches and the leaf nodes are cores. If the number of cores is C_n, the minimum number of needed switches(SW_n) to construct the topology is $SW_n = \lceil (C_n - 2)/2 \rceil$.

5. EXPERIMENTAL EVALUATION

We use Modelsim to simulate our NoC RTL code design, and verify the correctness by some experiments. Then, we utilize Synopsys Design Analyzer to synthesize our crossroad

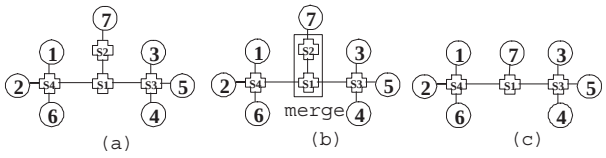

Figure 9: (a) Assign switches to clusters and super clusters (b) Combine adjacent switches (c) The final NoC infrastructure

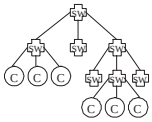

Figure 10: 3-nary tree structure

bus components and get the netlist schematic file. Before we start the simulation of power consumption, we must translate the netlist schematic file to SPICE model and use the simulation patterns that automatically generated by Modelsim. Next, we use Nanosim to do detailed low level simulation and gain the average power consumption.

Figure 11 shows the experiment NoC topology, which included two masters and three slaves. M1, M2 and M2 will do the "read" and "write" operations both once to S1, S2 and S3, respectively. For the power estimations, we first model another bus architecture, wishbone [16], and apply the same workload to compare with the estimated crossroad architecture. We use the simplescalar to simulate the MPEG4 decoder, and collect the 15 frames' cache access information. These frames include one I-frame and 40 P-frames. We also perform the evaluation by choosing two application cases: Video Object Plane Decoder (VOPD) and Multi-Window Displayer (MWD), presented in [12, 6], to show the resource improvement for the custom NoC. The two evaluation metrics are "power" and "performance". The power is estimates by Nanosim, and the performance is the cycles for completing the workloads.

5.1 Comparison of Power and Performance

The experimental results of the experiment NoC topology and the wishbone are shown in Table 1. Because the energy consumption is power × cycles, and the crossroad bus architecture is 507uW × 130 = 65910 uW, and wishbone is 528 uW × 180 = 95040 uW. The crossroad bus architecture can save the energy consumption approximated to 31%.

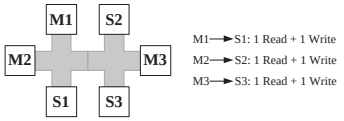

Figure 11: Experimental topology of the crossroad NoC

The profiled core flow graph of application MPEG4 is shown as Figure 12(a). The Figure 12(b) is our compared initial generated topology, and the PACP topology generated by our power-aware placement tool is shown as Figure 12(c). The experimental results for power consumption and performance of placement algorithms are shown in Table 2. The ratio of the power saving approximates to 40%. Obviously it saves power consumption if we carefully construct the NoC topology.

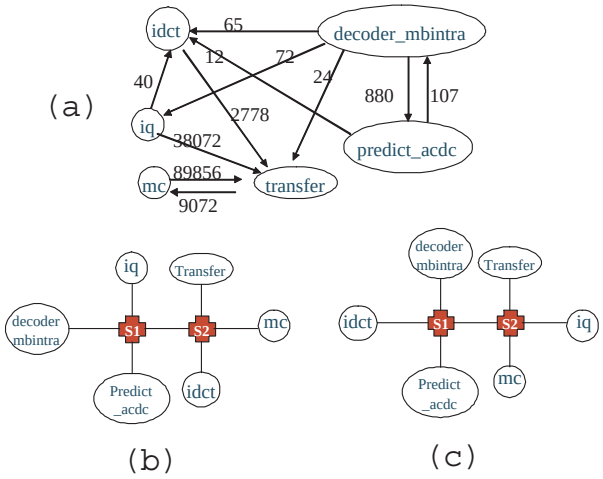

Figure 12: (a)Core flow graph of the MPEG4 decoder(b)Initial Topology(c)Power-aware Topology

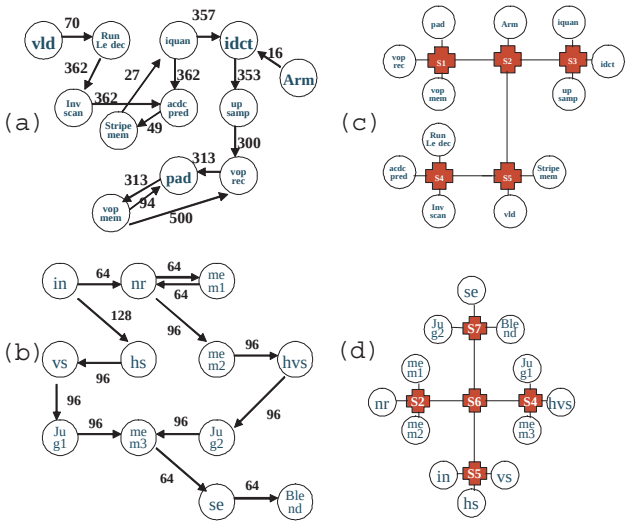

Figure 13: (a)Core flow graph of application VOPD(b)Core flow graph of application MWD(c)VOPD NoC topology(d)MWD NoC topology

Table 1: Estimates of bus architectures

	Simulation Metric	
Bus Architecture	Power (uW)	Performance (cycles)
Crossroad NoC	507.39	130
Wishbone	528.23	180

5.2 Comparison of Resources

The communication characteristics of applications VOPD and MWD are shown in Figure 13(a) and Figure 13 (b), respectively. We automatically developed customized application-specific topologies by our tool that closely matches the applications' communication characteristics. Figure 13(c) and Figure 13(d) show the generated topologies for applications VOPD and MWD, respectively. We can observe that although the cores of the two applications are both 12, however the topology will optimized for each specific-application communication characteristics. For both applications we need relatively small number of switches (5 crossroad switches) compared to mesh network (16 switches). We can obtain significant resource improvement for the custom NoC. Because we group high communicative cores into CCBs, the parallel communication can be achieved to enhance the communication performance.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented crossroad interconnection architecture and its design methodology. The key point of the design flow is to profile the application characteristics, and to connect high communicative cores in a local communication block to reduce the intercommunications as less as possible. The network architectures are highly optimized for the particular NoC design, providing savings in power, switches for example designs. In the future, we will continue to build a tool chain, which automatically instantiates an application-specific NoC in SystemC and verilog, and then the system can automatically completes the whole design flow and simulations for heterogeneous NoCs.

Table 2: Estimates of core placement algorithms

	Simulation Metric	
Placement Approach	Power (uW)	Performance (cycles)
Random placement	365.27	315994
PACP placement	260.59	310487

7. REFERENCES

[1] N. K. Bambha and S. S. Bhattacharyya. Joint application mapping/interconnect synthesis techniques for embedded chip-scale multiprocessors. *IEEE Transaction on Parallel and Distributed Systems*, 16(2):99–112, Feb. 2005.

[2] A. Bellaouar, I. Abu-Khater, and M. I. Elmastry. An ultra-low-power cmos on-chip interconnect architecture. In *Symposium on Low Power Electronics. Digest of Technical Papers*, pages 52–53, Oct. 1995.

[3] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergio, L. Benini, and G. D. Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transaction on Parallel and Distributed Systems*, 16(2):113–129, Feb. 2005.

[4] J. Y. Chen, W. B. Jone, J. S. Wang, H. I. Lu, and T. F. Chen. Segmented bus design for low-power systems. *IEEE Transactions on VLSI Systems*, 7(1):25–29, Mar. 1999.

[5] W. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of Design and Automation Conference DAC 2001*, pages 684–689, June 2001.

[6] E. V. der Tol and E. Jaspers. Mapping of mepg-4 decoding on a flexible architecture platform. In *SPIE2002*, pages 1–13, Jan. 2002.

[7] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of Soc. Industrial Appl. Math.*, 9(4):551–569, Dec. 1961.

[8] C.-T. H. and M. P. Architectural energy optimization by bus splitting. *IEEE Transactions on Computer-Aided Design on Integrated Circuits and Systems*, 21(4):408–414, Apr. 2002.

[9] W. Ho and T. Pinkston. A methodology for designing efficient on-chip interconnects on well-behaved communication patterns. In *Proceedings of Ninth Int'l Symp. High-Performance Computer Architecture*, pages 377–388, Feb. 2003.

[10] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 233–23, Jan. 2003.

[11] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In *Proceedings of DATE Conference*, Mar. 2003.

[12] E. Jaspers and P. de With. Chip-set for video display of multimedia information. *IEEE Transaction on Consumer Electronics*, 45(3):707–716, Aug. 1999.

[13] S. Murali and G. D. Micheli. Bandwidth constrained mapping of cores onto noc architectures. In *Proceedings of Conference DATE*, 2004.

[14] S. Murali and G. D. Micheli. Sunmap: A tool for automatic topology selection and generation for nocs. In *Proceedings of Design Automation Conference*, 2004.

[15] J. Plosila, T. Seceleanu, and P. Liljeberg. Implementation of a self-timed segmented bus. *IEEE Journals on Design and Test of Computers*, 20(6):44–50, 2003.

[16] Silicore Corporation. *WISHBONE System-On-Chip Interconnection Architecture for Portable IP Cores*, 2001.