Routing Algorithm for Symmetric Rearrangeable Networks and Emerging Applications

(IJCSNT) International Journal of Communication Systems and Network Technologies
ISSN Number: 2053-6283
Volume 2, Issue No. 1, 1–10

https://journals.mirlabs.in/index.php/ijcsnt
10.18486/ijcsnt/2.1.016

Amitabha Chakrabarty[1]

Abstract

Routing algorithms for symmetric rearrangeable networks have been discussed in greater detail in the literature. Major focus of these algorithms was on designing algorithms for full occupancy networks. One of the major issues with those algorithms was the required time to setup a switch of size N. Very little effort have been put to design algorithms which compromise between time complexity and switch throughput. In this paper we propose a new routing algorithm that uses two different routing methods in routing from input stage to output stage. One of the methods is called deterministic or optimal routing and other one is called adaptive routing. Deterministic method assures that each input request will have a definite path in the network and adaptive method makes sure that the path establishment takes place as fast as possible. This new algorithm works for both full and partial permutations without any modifications to the method. An optimal routing algorithm is used in the outermost stages of the network and in the innermost stages the decision making is based on the state of the switching elements. This new algorithm is called as hybrid routing algorithm. The required execution time of this new algorithm is much faster than optimal algorithm, and it has better scaling properties than suboptimal routing algorithms. This paper also addresses some of the emerging applications built using symmetric rearrangeable network class.

Keywords

Rearrangeable Networks, Permutation, Interconnection Networks, Routing Tags, Complexity.

Received: 29 December 2012; **Revised:** 31 March 2013; **Accepted:** 09 April 2013;
Published: 30 April 2013;

[1]Department of Computer Science and Engineering (CSE), BRAC University, Mohakhali, Dhaka, Bangladesh.

Corresponding author:
Email: amitabha@bracu.ac.bd

Introduction

There has been lots of research focused on symmetric rerrangeable networks. Design of efficient routing algorithms for these classes of networks was always a major research domain. The Major focus of recent communication researches in electrical or optical domain is to increase the throughput of each switching ports rather than increasing number of ports[2,3]. Increasing the port capacity after a certain boundary will introduce crosstalks to the system. In this paper we focus on Beneš network[1] where to increase the capacity of the switch means increasing number of ports rather than increasing port's throughput. Waksman[6] proposed a recursive algorithm for setting the switching element state in the Beneš network for uni-processor system. He showed that Beneš network is the shorter depth 2×2 rearrangeable network. The algorithm proposed by Opferman and Tsao-Wu[12], called the looping algorithm, works from the outer stage towards the center stage. It works by dividing the entire network into smaller networks and recursively setting paths in the smaller networks, there by setting the complete path. Later Andreson[13] provided an extended version of the looping algorithm for base 2^t networks[4,5].

Nassimi and Sahni[8,10] proposed a parallel self-routing method for a particular class of permutations. Nassimi and Sahni[11] proposed their way to implement Waksman's[6] approach in a parallel processing mode. To reduce the switch setting time in Beneš networks, Feng and Seo[2,3] proposed Inside-out routing method. In that method they developed a new way to setup connecting paths for input/output request starting from the middle stage to the outward direction. Kim[7] showed that Inside-out routing method needs back-tracking and even after back tracking it's not fully blocking free. Lee[4,5], proposed a non-recursive algorithm, where she divided the network in two parts: NS1 and NS2. Her algorithm works on a single stage of the network from left to right. Another algorithm proposed by Çam and Fortes.[19], used a parallel machine called a PRAM (Parallel Random Access Machine)[20] to reduce the required number of processors for determining the routing tags for the network in parallel. Each instruction stream takes unit time in the PRAM structure regardless of the processors number and each processor has a flag that indicates whether the processor is busy or idle. The PRAM model is not a realistic model of available hardware. One of the biggest drawbacks with PRAM is its constant memory access time, as this model suggests processor number p to be a large number, so in the physical implementation these processors will occupy some physical space and the location of the memory access time cannot be the same for all processors. Another issue is with the concurrent reads and writes operation mentioned in PRAM model[8]. With current memory structure, there is a limit on the number of concurrent read and write, which also suggest that it is not possible to perform read and write simultaneously by all the processors. A third issue is that the shared memory has a capacity of $O(N)$, where N is the number of network inputs and outputs. If the memory is on a shared bus, this will limit the speed of execution. If the memory is implemented with an independent path from each processor to each memory element, the complexity of the required interconnect equals that of the network to which the routing algorithm is being applied. Since PRAM model is not practical, so the method of routing in[19] is yet to prove itself to be work efficient[10].

Our Contribution

The algorithms cited in Section I have serial time complexity of $O(N \log N)$ except the inside-out routing[2,3]. Inside-out routing has a time complexity of $O(N)$ (N is total number of inputs in the network), but it has blocking characteristic and to reduce the blocking ratio time complexity approaches to $O(N \log N)$. $O(N \log N)$ is the shortest possible time required to setup a Beneš network, as the network

has $O(\log N)$ depth and has $O(N)$ switching elements. Our contribution in this paper is to derive an algorithm that can achieve this minimum time complexity and when simulated gives faster execution time than existing algorithms. This work is an extension of the work presented in[9]. In our previous work we compared the result of our proposed algorithm with random routing and showed that our routing method has better throughput than random routing. In this paper we compare the result of our proposed algorithm with looping algorithm and also with a modified version of our proposed routing which we call adaptive routing[14]. Looping algorithm has been chosen for comparison because of the method's wide acceptance in the research community. An analysis of the time complexity of the algorithm is presented in this paper along with addressing some new emerging application where the proposed algorithm can be applied[15].

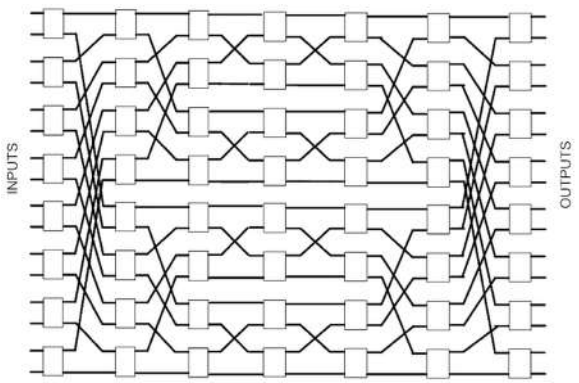

Figure 1. 16×16 Beneš Network.

Basic Routing Algorithm

The networks under observation have a total of $(2 \log N - 1)$ stage. From these stages first $(\log N - 1)$ stages follows deterministic routing and the remaining stages are bit controlled. In this paper we make routing decision in the distributed half of the network using the method proposed in this paper and bit controlled part routes the other half. An "outer" algorithm, which is applied to the outermost k stages, $0 \leq k \leq (\log N - 2)$, of the network (for this paper $k = 0$). An "inner" algorithm, which is applied in all the remaining stages. This algorithm trades off performance against execution time[16]. The design of the inner algorithm follows recursive routing. In the distributed part of network the algorithm selects paths depending on the state of the switching element. If the switching element is in idle state, it is set to the straight through state and the signal goes to the next stage[17]. In straight through state of a switching element at stage k, signal at input port i exits from output port $j = i$ of the switching element and goes to stage $k + 1$. If the switching elements are already set to a state, the unused output port is used to goto the next stage. In case where the switching element is set to a cross state, signal coming to input port i exit from output port $j = i - 1$ if i is odd else $j = i + 1$. Once the signal finds a center stage switching element, the rest of the routing is bit controlled. For any conflict that may occur in bit-controlled routing is resolved by choosing an alternative connecting path, by going back to stage $k = (\log N - 1)$. In case of any unsuccessful path search, start with a new search from stage $k = k - 2$ iff $k = 0$ for that input.

For $k = 0$ the algorithm drops the request and start routing with a new request and the routing process continues for rest of the available inputs[18]. For implementing adaptive routing we excluded the looping algorithm part from our proposed method. Elimination of the looping algorithm force adaptive algorithm to make switching elements status based decision making between stages $k = 0$ to $k = (\log N - 2)$[21].

Preliminaries

Some notation used to describe the state variables in the algorithm code is presented below. For simplicity we assume that $a = (\log N - 1)$ and $b = (2 \log N - 2)$.

Definition 2.1: Input Permutation is the set of one to one requests between the switch inputs and outputs. In more mathematical term, a mapping of an input to an output is an element in the input permutation. Let us assume that $P_{0:(N-1)}$ is a given permutation such that, $P_{0:(N-1)} = \{x_i \mid x_i \in \{0 \ldots (N-1)\}\}$, where $x_i \neq x_j$, and $0 \leq (i, j) \leq (N-1)$. The mapping $P : i \rightarrow x_i$ indicates that input port i is requesting for the output port x_i.

$$P_{0:7} = [0\ 7\ 3\ 2\ 4\ 1\ 5\ 6]^T \tag{1}$$

For example, an 8×8 network with an input permutation $P_{0:7} : (0\ 7\ 3\ 2\ 4\ 1\ 5\ 6)$, maps $0 \rightarrow 0$, $1 \rightarrow 7$ and so on. A binary representation of these permutation for given by the above permutation matrix, where each row number corresponds to an input port and bits correspond to the requested output port, which can be used to expressed in binary the switching element settings[22].

Definition 2.2: (STATE). STATE is an array that holds the state of all the 2×2 switching elements in the network. A switching element can occupy one of three states. The state of the switching element at position $[j, k]$ in the network is recorded as STATE$[j, k]$ where $j = \lfloor \frac{i}{2} \rfloor$ where $0 \leq i \leq N-1$, and $0 \leq k \leq b$. If STATE$[j, k] = NULL$, the switching element is actually un-configured. If STATE$[j, k] = 0$ the switching element is set to perform straight through switching and when STATE$[j, k] = 1$ it will perform a cross operation (i.e. will connect its lower input to its higher output and vice versa).

Definition 2.3: (Forward Routing.) Forward routing is the establishment of a routing path from an input i, $0 \leq i \leq (N-1)$, to the requested output port $P(i)$, where P is the input permutation. Fig 2(a) shows the routing from input i to output $P(i)$, $0 \leq i \leq (N-1)$, where the nodes represent the switching elements in the input and output stages, and the path through the stages is represented by a straight line.

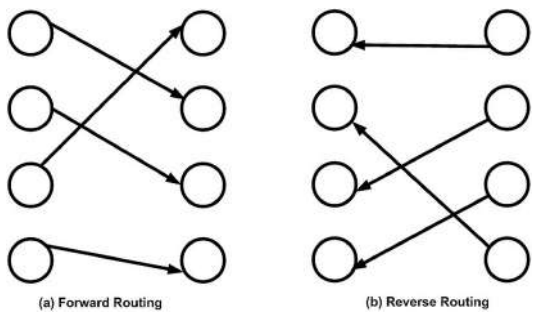

Figure 2. Forward and reverse routing

Definition 2.4: (Reverse Routing.) The establishment of a path from output $P^{-1}(i)$ to input i, $0 \leq i \leq (N-1)$, is termed reverse routing. Fig 2(b) shows the routing from output port $P^{-1}(i)$ to input i, $0 \leq i \leq N-1$.

Definition 2.5: (Neighbour Port.) The neighbor port $Ne(i)$ to port i is the port adjacent to it in a switching element. $Ne(i) = i+1$, if i is even otherwise it is $(i-1)$.

Definition 2.6: (Port Mapping.) As a signal passes through the network, its address changes, in the sense that the port number in a given stage at which the signal is present differs, starting with address i at the network input, and ending with address $P(i)$ at the network output. The changes in address can be regarded as due to port mapping. Two types of port mapping can occur, those caused by the switching elements, and those caused by the link patterns. A signal presenting at input i_k in stage k will emerge at output O_k where, the input will map to the output with the same address if the relevant switching element is in the straight configuration, and to its neighbour output if the switching element performs a cross operation.

The mapping performed by the link patterns differ in the first a stages of the network and the remaining stages, because of the symmetric arrangement of the link patterns. During the mapping between output port O_k of stage k and the corresponding input port in stage $k+1$ as $M_k(O_k)$ and using a binary representation for the perfect shuffle and inverse perfect shuffle, it follows that where the binary representation of O_k is $(b_l b_{l-1} \ldots b_0)$ and $l = a$.

The reverse mapping must be known to perform reverse routing. This maps input port i_{k+1} to the corresponding input port in stage k. Evidently this is port $M^{-1}(i_{k+1})$.

Hybrid Routing Algorithm

The design of the algorithm follows the concept of looping algorithm, which follows recursive routing. Recursive routing is divided into two parts forward routing and reverse routing. In the distributed part of the network proposed algorithm selects paths depending on the state of the switching element. For an idle state, switching element is set to the straight through state and the signal goes to the next stage. If the switching elements is already set to a state, the unused output port is used to go to the next stage. Once the signal finds a center stage switching element, the rest of the routing is bit controlled. Any conflict that may occur is resolved by choosing an alternative connecting path.

Forward Routing

This section describes forward routing for first a stages. Rest of the routing is bit controlled.

1. Set $k = 1$
2. Find a unconnected input i_1. Set $i = i_1$
3. Set $\left[\left\lfloor \frac{i}{2} \right\rfloor\right]$, if it is not already set.
4. If $\left[\left\lfloor \frac{i}{2} \right\rfloor\right]$ is already set, apply port mapping defined in section 2.6 and go from stage k to $k+1$
5. Continue Steps 3–4 for first $(a-1)$ stages
6. From stage a to $(b-1)$ follow bit controlled routing
7. At $k = (b-1)$ set $i = Ne(P(i))$. If i is not connected start reverse routing, else goto stage 1 and set any non connected input $i_1 = i$, goto step 3

Reverse Routing

Once the forward routing is completed, reverse routing starts from the neighbour port of $P(i)$. If neighbour port is already connected routing goes to stage 1 and start forward routing with any unused input port.

1. Depending on value of $\left[\lfloor \frac{i}{2} \rfloor\right]$ apply reverse port mapping and goto from k to $k-1$
2. Continue step 1 for $k = (b-1)$ to $k = (a-1)$
3. From stage $k = a$ to $k = 1$ apply bit controlled routing
4. At stage $k = 1$, set $i = Ne(i)$ and start forward routing

Conflict Resolution

The process of forward and reverse routing might encounter a situation where there is no path available for a request $i \rightarrow P(i)$. In those situations a conflict resolution process executes to overcome the conflict. Conflict resolution starts from the stage just before stage where bit controlled routing started and searches for alternative path. Any unsuccessful path search start a new search from stage $k = (k-1)$ for forward routing or $k = (k+1)$ for reverse routing. If $k = 0$ or $k = b$ request i is dropped and forward routing start with any unconnected input port.

1. For forward routing any conflict at stage k, go back to stage $k = (a-1)$. Use the alternative output port $\left[\lfloor \frac{i}{2} \rfloor\right]$ of is any and search for a new routing path. If no new path is available then go back to stage $k = (a-2)$ and repeat the path search. For every unsuccessful search apply $k = k-1$ and repeat the search until $k = 0$. If $k = 0$ drop the request, and start forward routing with any unconnected port.
2. For reverse routing, any conflict at stage k, go back to stage $k = a+1$, and use the alternative input port $\left[\lfloor \frac{i}{2} \rfloor\right]$ of is available and search for alternative routing path. If there is no unconnected input port in the switching element then go back to stage $k = k+1$ and repeat the path searching process. In any new conflict arises, continue the search process till $k = (b+1)$. If $k = (b+1)$, drop the request, set $k = 1$ and start forward routing with any unconnected input.

Adaptive Routing Algorithm

This section modifies the algorithm proposed in Section III. The modification eliminates the use of deterministic routing algorithm in the outer most stages, i.e. the looping algorithm. This makes the modified algorithm fully adaptive, which means the routing decision for first a stages depends on the status of the switching elements only. This gives the algorithms bigger scope of searching for alternative routing paths. This is due to the fact that in adaptive routing method algorithm only drops a request when the condition $k < 0$ satisfies. The objective of adaptive routing method is to see is it possible to have a better blocking performance and execution time than the hybrid routing method. Section V shows a detail comparison between hybrid routing, adaptive routing and also random routing.

Simulation Results

To compare results of the proposed algorithms with existing literature two other algorithms have been chosen. One of them is the looping algorithm because of its wide acceptance among the research

community as a non blocking algorithm. The other one is the random routing algorithm, which is being accepted as a very fast routing algorithm but with limited throughput. The performance of the methods described is measured using three different metrics: blocking probabilities, required path search and setup time. Fig 3 shows the performance graphs for a full input occupancy network. In other words, all the inputs are active. All the simulations have been tested in an Intel(R) Core(TM) 2 Quad 2.40 GHz CPU computer with a memory of 8GB. All simulation codes are written using C++ programming language. The results show that hybrid and adaptive have similar blocking probabilities throughout the observation window. The blocking probabilities shows little variation for smaller values of N where hybrid routing has a slight edge over the adaptive approach. Comparing these two methods with the Random routing shows that both hybrid and adaptive outperforms Random routing for all values of N by a large margin. So for full a occupancy network when probability of blocking is the performance measuring tool, hybrid and adaptive give similar performance with few exceptions, but overall they are always superior than Random routing.

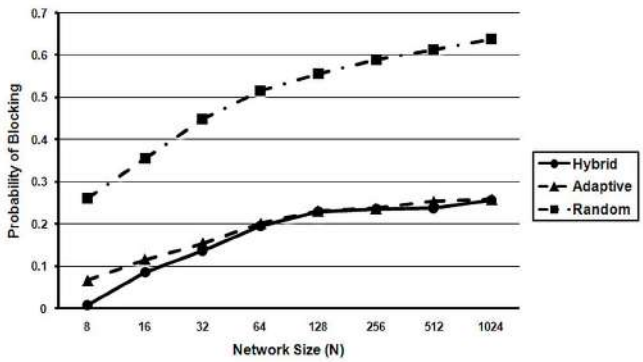

Figure 3. Performance of three different methods for full input occupancy.

The required number of path searches is another performance analysis metric. Only hybrid routing and adaptive have been considered in this analysis as Random routing does not use alternative path searching in the case of internal blocking. The path searching count is an important tool to measure algorithm complexity as it is a measure of the time it takes to configure the network. To continue with the results, Fig 4 shows the curves indicating the average required number of path searches for both methods for a full occupancy network. Unlike the blocking probability curves, these two curves show huge differences in path search count. For example for $N = 512$, adaptive routing requires almost two times as many path searches as hybrid routing.

The execution time of the two methods is another important measure. Here the execution time of looping algorithm is also taken into consideration.

Fig. 5 shows the result when the two algorithms applied in a network with full input occupancy. The figure suggests that for smaller networks (for example $N = 64$) there is almost no observable difference in the two routing algorithms. But significant differences can be seen for larger networks. For example, when $N = 1024$ the time difference of more than 150 ms. Also the hybrid algorithm takes less time to execute for larger network than the looping algorithm.

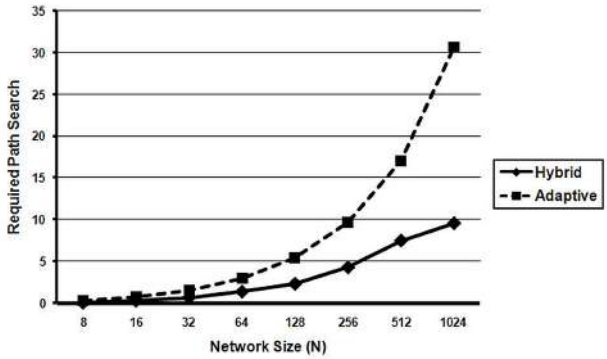

Figure 4. Path search graph for a full occupancy network.

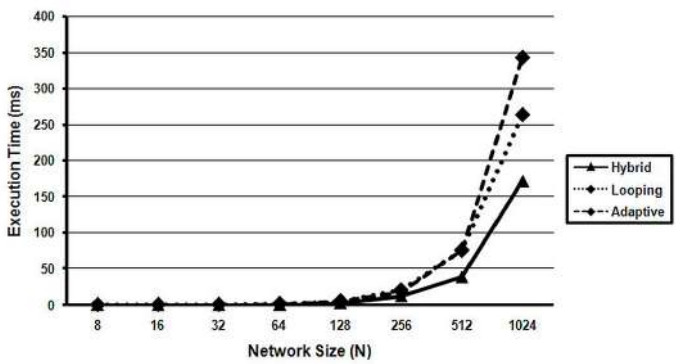

Figure 5. Execution time for full occupancy network.

Emerging Application

New applications have emerged for symmetric rearrangeable class of networks in the field of system on chip (SoC) and network on chip (NoC). SoC is an arrangement of two or more complex microelectronic components in a single chip[14–17]. Complex functionalities that required heterogeneous components attached to a PCB are replaced by SoCs. Advancements in the silicon technologies allow large functional unit to be built in a single chip. A typical SoC contains processors, on chip memories, accelerated functional units, signal processing units, logic circuits etc. The primary advantages of SoC is low cost, smaller in size and fast performance. Because of the SoC today's hand held devices are smaller in size compared to the bulky old versions. NoC overcomes the scalability and performance issues related to the bus based or point to point communication structure in the SoC[15,18,21,22]. The obvious choice for NoC in an SoC is the crossbar networks as they give superior performance than bus base models, but this network also suffers from scalability issue after a certain input number along with low network utilization[23]. So the solution to these is multistage interconnection networks having switching elements arranged in rows

and columns and each switching element is connected to the next stage via some fixed link patterns. These networks have better scalability property along with equal path distance between source and destination which make then viable for SoC.

Conclusion

This paper demonstrates the benefit of using a routing algorithm that is not optimal but effective. Compromising a fraction of the throughput compared to optimal routing algorithms, faster network setup time can be achieved. Previously available sub-optimal routing algorithms lack the efficiency hence not applicable in any real world application. New application domains have been addressed in this paper. The proposed algorithm achieves a balance between throughput and execution time. As a result this algorithm is of potential interest for application such as NoC where fully deterministic algorithms are still preferred. This algorithm is designed to work for serial domain applications hence not suitable for full parallel implementations. In future paper from the authors will discuss a detail time complexity analysis of the algorithm. Also prospect of partial parallel implementation of the algorithm and effect on overall execution time will be address in future.

References

[1] E. Benes. Mathematical Theory of Connecting Networks and Telephone Traffic. New York: Academic Press, 1965.

[2] S.-W. Seo, T.-Y. Feng, and H.-I. Lee. Permutation Realizability and Fault Tolerance Property of the Inside-Out Routing Algorithm. IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 9, pp. 946-957, 1999.

[3] T.-Y. Feng and S.-W. Seo. A New Routing Algorithm for a Class of Rearrangeable Networks. IEEE Trans. Computers, vol. 43, no. 11, pp. 1270-1280, 1994.

[4] K. Y. Lee. On the rearrangeablity of $2(\log_2 N - 1)$ stage permutation networks. IEEE Trans. Comput, vol. C-34, no. 5, pp. 412-425, May 1985.

[5] K. Y. Lee. A new Benes network control algorithm and Parallel Permutation Algorithm. IEEE Trans. Comput., vol. C-30, no. 5, pp. 157-161, May 1981.

[6] A. Waksman. A Permutation Network. J. ACM, vol. 15, no. 1, pp. 159-163, Jan. 1968.

[7] M.K. Kim, H. Yoon, and S.R. Maeng. On the Correctness of Inside-Out Routing Algorithm. IEEE Trans. Computers, vol. 46, no. 7, pp. 820-823, July 1997.

[8] D. Nassimi and S. Sahni. A self-routing Benes network and Parallel Permutation Algorithm. IEEE Trans. Comput., vol. C-30, no. 5, pp. 332-340, May 1981.

[9] A. Chakrabarty, M. Collier, S. Mukhopadhyay. Adaptive Routing Strategy for Large Scale Rearrangeable Symmetric Networks. International Journal of Grid and High Performance Computing (IJGHPC), vol. 2(2), pp. 53-63, 2010.

[10] D. Nassimi and S. Sahni. A self-routing Benes network. Proceedings of the 7th annual symposium on Computer Architecture. La Baule, United States Rep. pp: 190-195, May 1980.

[11] D. Nassimi and S. Sahni. Parallel Algorithms to Set Up the Benes Permutation Network. IEEE Trans. Comput., Vol. c-31, No. 2, February 1982.

[12] D. C. Opferman and N.T. Tsao-Wu. On a Class of Rearrangeable Switching Networks, Part I: Control Algorithm. Bell System Technical J., vol. 50, pp. 1,579-1,600, 1971.

[13] S. Andresen. The looping algorithm extended to base 2^t rearrangeable switching networks. IEEE Trans. Commun., vol. COM-25, no. 10, pp. 1057-1063, Oct. 1977.

[14] Steve Furber. ARM System-on-Chip Architecture. Addison-Wesley Longman Publishing Co. Inc, 2000.

[15] Bjerregaard, Tobias and Mahadevan, Shankar. A survey of research and practices of Network-on-chip. ACM Comput. Surv., vol. 38, June, 2006.

[16] Drew Wingard. MicroNetwork-Based Integration for SoCs. In Proceedings of the 38th Design Automation Conference, pp. 673-677, 2001.

[17] Andreas Gerstlauer, Gunar Schirner, Dongwan Shin, Junyu Peng, Rainer Domer, Daniel D. Gajski. System-on-Chip Component Models. Technical Report, University of California, Irvine, 2006.

[18] Pierre Guerrier Alain and Alain Greiner. A Generic Architecture for On-Chip Packet-Switched Interconnections. Proceedings of the conference on Design, automation and test in Europe (DATE), pp. 250-256, 2000.

[19] Hasan Çam and Jose A.B. Fortes. Work-Efficient Routing Algorithms for Rearrangeable Symmetrical Networks. IEEE Trans. on Parallel and Distributed Systems, Vol. 10, No. 7, July 1999.

[20] R. Cypher, J. L. C. Sanz, L. Snyder. An EREW PRAM Algorithm for Image Component Labeling. IEEE Trans On Pattern Analysis And Machine Intelligence. Vol. 11, No. 3. March 1989.

[21] Dally, William J. and Towles, Brian. Route packets, not wires: on-chip inteconnection networks. Proceedings of the 38th annual Design Automation Conference, pp. 684-689, 2001.

[22] Benini, L. and De Micheli, G. Networks on chips: a new SoC paradigm. Computer, vol. 35, pp. 70-78, 2002.

[23] Xu, Jiang and Wolf, Wayne and Henkel, Joerg and Chakradhar, Srimat and Lv, Tiehan. A Case Study in Networks-on-Chip Design for Embedded Video. Proceedings of the conference on Design, automation and test in Europe, vol. 2, 2004.