A Study of the Permutation Capapability of Modified Omega Network

by

Phanthip Thaiyoo

Faculty of Engineering
June 2003

A Study of The Permutation Capability of A Modified Omega Network

A Thesis
Submitted to the Faculty of Engineering

By

Phanthip Thaiyoo

In partial fulfillment of the requirements
for the degree of
Master of Engineering in Broadband Telecommunications

Advisor: Dr. Gennady Veselovsky

Assumption University
Bangkok, Thailand
June 2003

"A Study of The Permutation Capability of A Modified Omega Network"

by

Ms.Phanthip Thaiyoo

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Engineering
Majoring in Broadband Telecommunications

Examination Committee:

1.	Dr.Gennady	Veselovsky	(Advisor)	[illegible]
2.	Dr. Thiraphong	Charoenkhunwiwat	(Member)	[illegible]
3.	Asst.Prof.Dr.Putchong	Uthayopas	(Member)	[illegible]
4.	Asst.Prof.Dr.Tubtim	Angkaew	(MUA Representative)	[illegible]

Examined on: May 20, 2003
Approved for Graduation on: June 9, 2003

Faculty of Engineering, Assumption University
Bangkok, Thailand

A Study of the Permutation Admissibility of A Modified Omega Network

By

Phanthip Thaiyoo

Abstract

The class of fault-tolerant redundant path (R-path) multistage networks was derived from Omega networks by Padmanabhan and Lawrie who proved that the new network class retained all the connection properties of the parent networks in the absence of faults. In the thesis a simple window method that allows determining the admissibility of any BPC (Bit Permutation Complement) permutation to an R-path Omega network is introduced. A study of the permutation admissibility of an R-path Omega network is based on a computational exploration with C language program and deals with the variable sizes of a network and switching element. It is shown that the permutation capability of R-path Omega networks is much better than that of the parent networks. For example such powerful and frequently used in parallel programming permutation as perfect shuffle is admissible to all possible configurations of R-path Omega networks being non-admissible to 1-path Omega networks.

Acknowledgements

I would like to thank my advisor, Dr. Gennady Veselovsky, for all of his advises and motivation throughout the completion of this thesis. Also, I would like to thank Asst. Prof. Dr. Kittiphan Techakittiroj for his suggestion and guidance. Furthermore, I would like to thank Dr. Sudhiporn Patumtaewapibal, Dean of Faculty of Engineering, and all other faculty members of Broadband Telecommunications department and laboratory for their support.

Also, I would like to thank all of my classmates for their support and encouragement. Finally, I would like to thank my family for their support, encouragement, and understanding.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The ***Omega network or 1-path Omega network*** was originally proposed by Duncan H. Lawrie [1] as an alignment network between processors and memories in the array processor systems. In general, an $N \times N$ 1-path Omega network as shown in Figure 1.1 can connect N processors to N memories by using a small number of stages, and each stage composed of fewer than N switches. The 1-path Omega networks are attractive for scalable shared-memory architectures like single instruction stream multiple data stream (SIMD) computers because their use of multiple simple switches are able to ensure feasibility of system and their use of a few stages with many switching modules precludes needing a long, inefficient interconnection network pathway.

A 1-path Omega network has a self-routing algorithm for establishing the path of any given source and destination. This algorithm always chooses one and only one path through the network between any source-destination pair, as the result, the ability of fault-tolerance of 1-path Omega network is limited; the whole system will be destroyed if there is any failure in a link or in a particular switch. Additionally, 1-path Omega network is a blocking network, i.e., some permutations cannot be established by the network for one pass. Figure 1.1 shows the paths established for the mapping 000→000, 100→010. These paths share a common connection at the output of the first stage, and a conflict occurs.

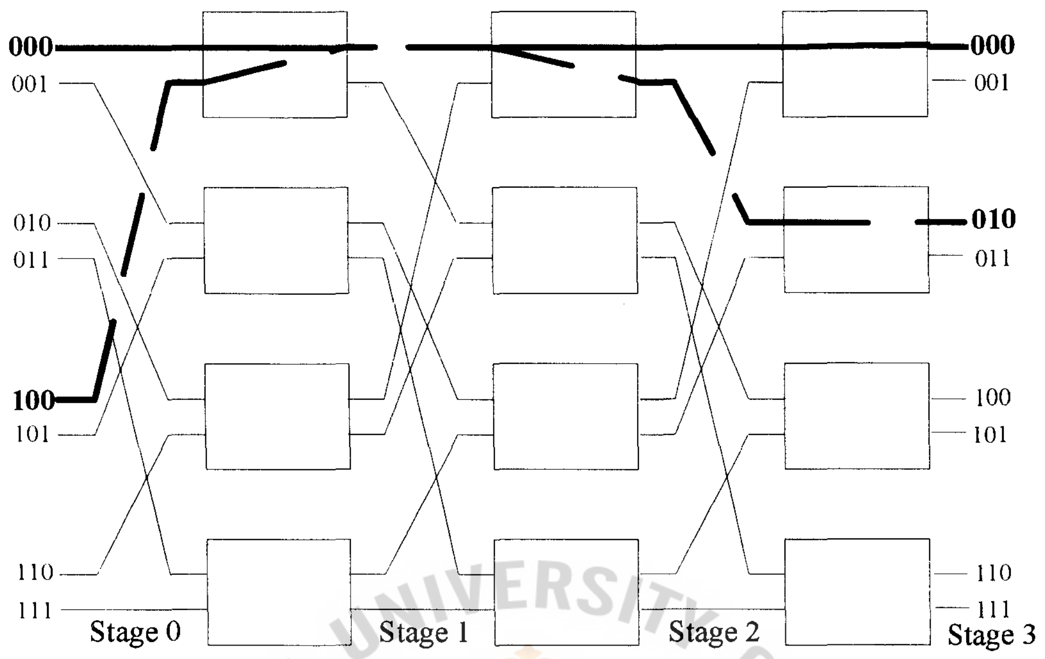


Figure 1.1 Path for (000,000), (100,010) in a 1-path 8 x 8 Omega network

To overcome the fault-tolerance limitation of the Omega network, K. Padmanabhan and D. Lawrie proposed its modification which is called ***Modified Omega Network or R-path Omega network*** [2]. The close relation to the 1-path Omega topology sustains R-path Omega network maintainable all connection properties of the standard 1-path Omega network.

The R-path Omega network achieves the fault-tolerance in the form of providing multiple disjoint paths between every source and every destination. For example, Figure 1.2 shows two different paths from source $S = 01000$ to destination $D = 01100$ in 32 x 32 2-path Omega network. Since the only one of R set is needed to provide connectivity between any source-destination pair, the network can tolerate the breakdown of any link or a partial failure of an intermediate switch. As an example in Figure 1.2, the failure in one of these two sets of intermediate switches or links will not prevent source S = 01000 from accessing destination D = 01100, because the leftover set that does not break down can be used instead.

Refer to [2], the authors of R-path Omega network proved that "**the multipath network would pass any permutation (π) that 1-path network passes**", i.e., no conflict could occur in the multipath network if none occurred in the 1-path network.

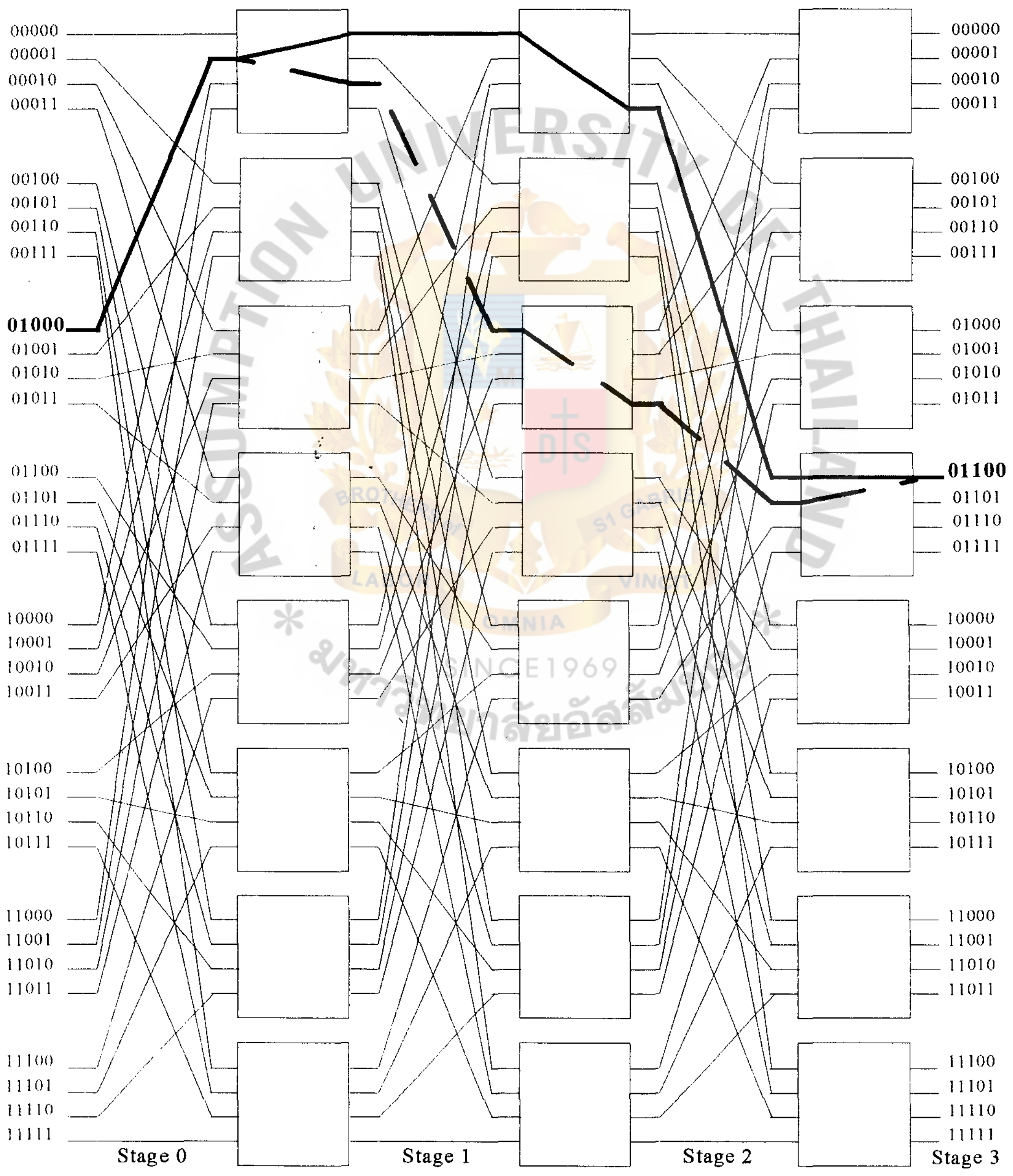


Figure 1.2 A 2-path Omega network of 32 inputs and 32 outputs. The solid and dashed edges indicate the two sets of disjoint paths

1.2 Goal of Thesis

The main objective of this thesis is to explore how the presence of multiple paths in R-path Omega network improves its permutation capability in comparison with that of a 1-path Omega network. The thesis seeks technique for testing admissibility of BPC permutations to R-path Omega network.

1.3 Relevant Literature

Before proceeding further, it will be recalled that a permutation is called a BPC permutation if the destination address can be obtained of the source address by permuting its bits $(s_0 s_1 s_2 \cdots s_{n-2} s_{n-1})$ and/or complementing some or all of its bit positions. The BPC permutation admissibility problem, which is being solved in this thesis, is most closely to the problem of admissibility solved by Xiaojun Shen [12]. However Shen's investigations are in concern with another class of interconnection network, namely k-Extra Stage Multistage Cube-Type Networks (k-EMCTN), where redundant disjoint paths between any pair of source and destination are provided by adding k more stages in front of a multistage cube-type network. In addition, our approach to solving the problem of admissibility is different. Therefore, it can be said that we are the first who explore admissibility of the most frequently used Bit Permutation Complement (BPC) permutations to R-path Omega network with the routing algorithm developed by us.

1.4 Thesis Overview

Section 2.1 considers on the general concept of interconnection network: static network, and dynamic network. Section 2.2 emphasizes on Multistage Interconnection Networks (MINs). The connection capability and main properties of MINs are described in section 2.2.1 and 2.2.2 respectively. While section 2.2.3 depicts MINs application in both B-ISDN and parallel computers environments. Chapter 3 considers 1-path and R-path Omega networks. The detail of their structures and main properties are discussed. The examples of BPC permutations that we used to explore are given in section 3.3. Chapter 4 is the exploration part, our admissibility algorithm concept and its pseudo code and flowchart are shown in section 4.1, and section 4.2 respectively. The frame format of C language program for simulating the exploration is shown in Section 4.3 with pseudo code and flowchart form as well. Section 4.4 shows the summary of permutation capability results. The last, chapter 5 concludes the overall of thesis, and the recommendation of future work.

Chapter 2

Fundamental Theory and Basic Concept

2.1 Interconnection Networks

Formally, Interconnection Network (IN) topology refers to the layouts of links and switch boxes that establish interconnections. The links are essentially physical wires (or channels); the switch boxes are devices that connect a set of input links to a set of output links. For high performance system, the interconnection network topologies tend to be regular and can be grouped into two categories: static and dynamic.

2.1.1 Static Networks

The static networks provide the means of fixed connections between N processors and N memory modules; the links between nodes are unchangeable and cannot be easily reconfigured [4]. There are various types of static networks, all of which can be classified according to the dimensions required for layout. Figure 2.1 shows one-dimensional, two-dimensional, and three-dimensional topologies. One-dimensional topology, Figure 2.1(a), is the linear array. It is used for some pipeline architectures. Two-dimensional topologies, Figures 2.1(b) through 2.1(f), include the ring, star, tree, mesh, and systolic array. Three-dimensional topologies, Figure 2.1(g) through 2.1(j), include the completely connected, chordal ring, 3 cube, and 3-cube-connected-cycle networks.

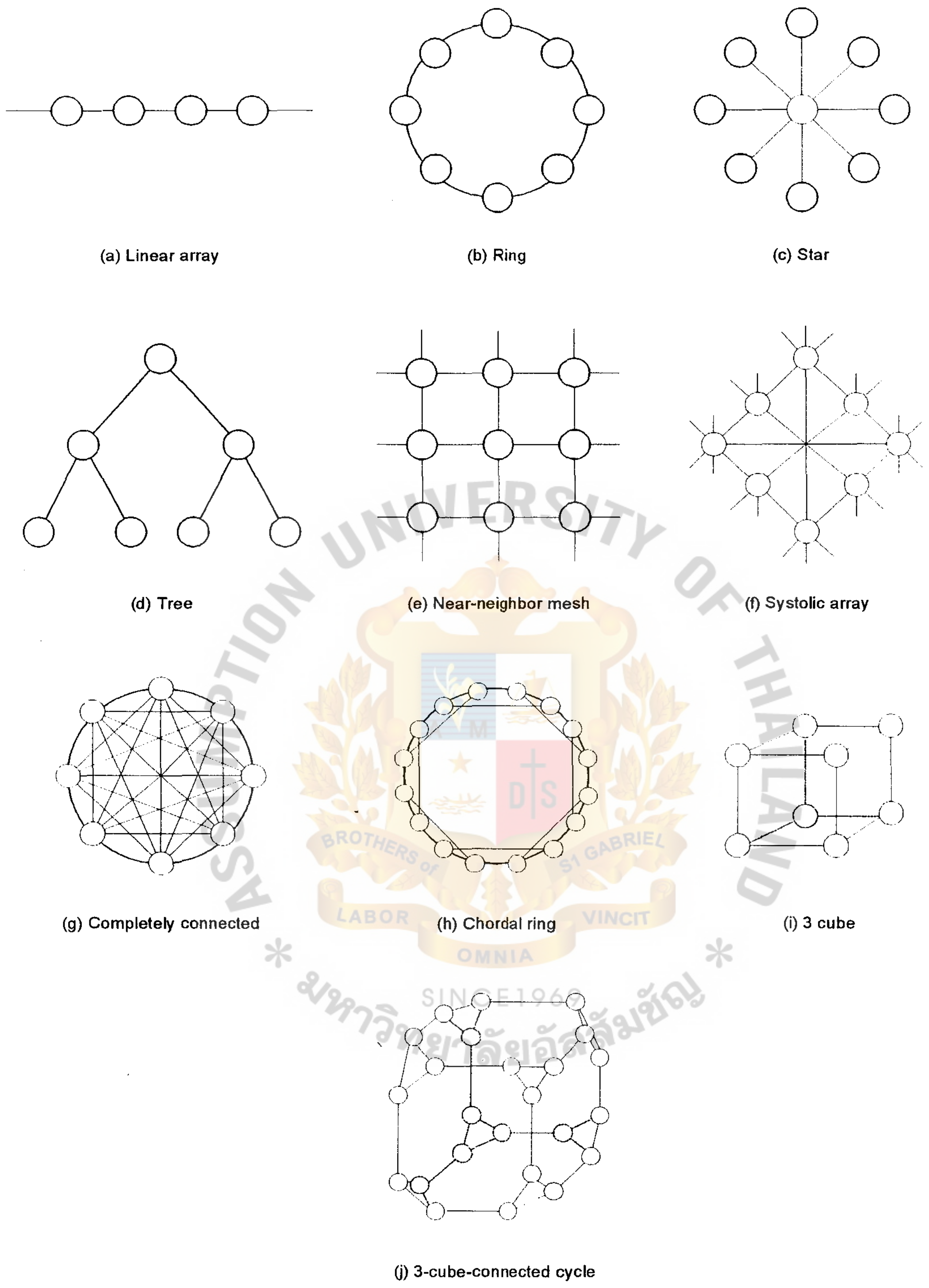


Figure 2.1 Static interconnection network topologies

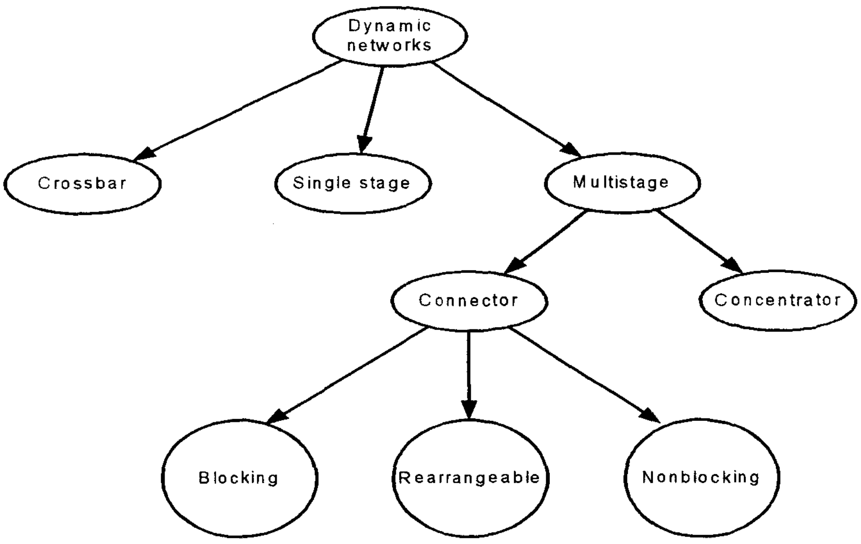


Figure 2.2 Classification of dynamic networks

2.1.2 Dynamic Networks

In contrast to the static network, links between nodes in dynamic topologies provide reconfigured connections [4]. The classification of dynamic networks is presented in Figure 2.2. First level of the hierarchy lists three forms of dynamic interconnection networks: (1) crossbar, (2) single-stage, and (3) multistage.

Firstly, the crossbar network is the oldest type and has served as the basic switching structure or fabric for many years. In a crossbar, the inputs and outputs are connected to set of switching points called crosspoints. Since each input-output combination has an individual crosspoint, any input can be connected to any output at any time with no possibility of blocking connection. For example, Figure 2.3(a) shows input 001 is connected to output 011, and at the same time input 010 is connected to output 001. Although the connection of crossbar is made on fly, any source can connect to any desired destination, the crossbar is impractical for large network because the cost of crossbar network is driven largely by the crosspoint count. For example, the cost of a crossbar then is NM for a $N \times M$ network or N^2 for a square network.

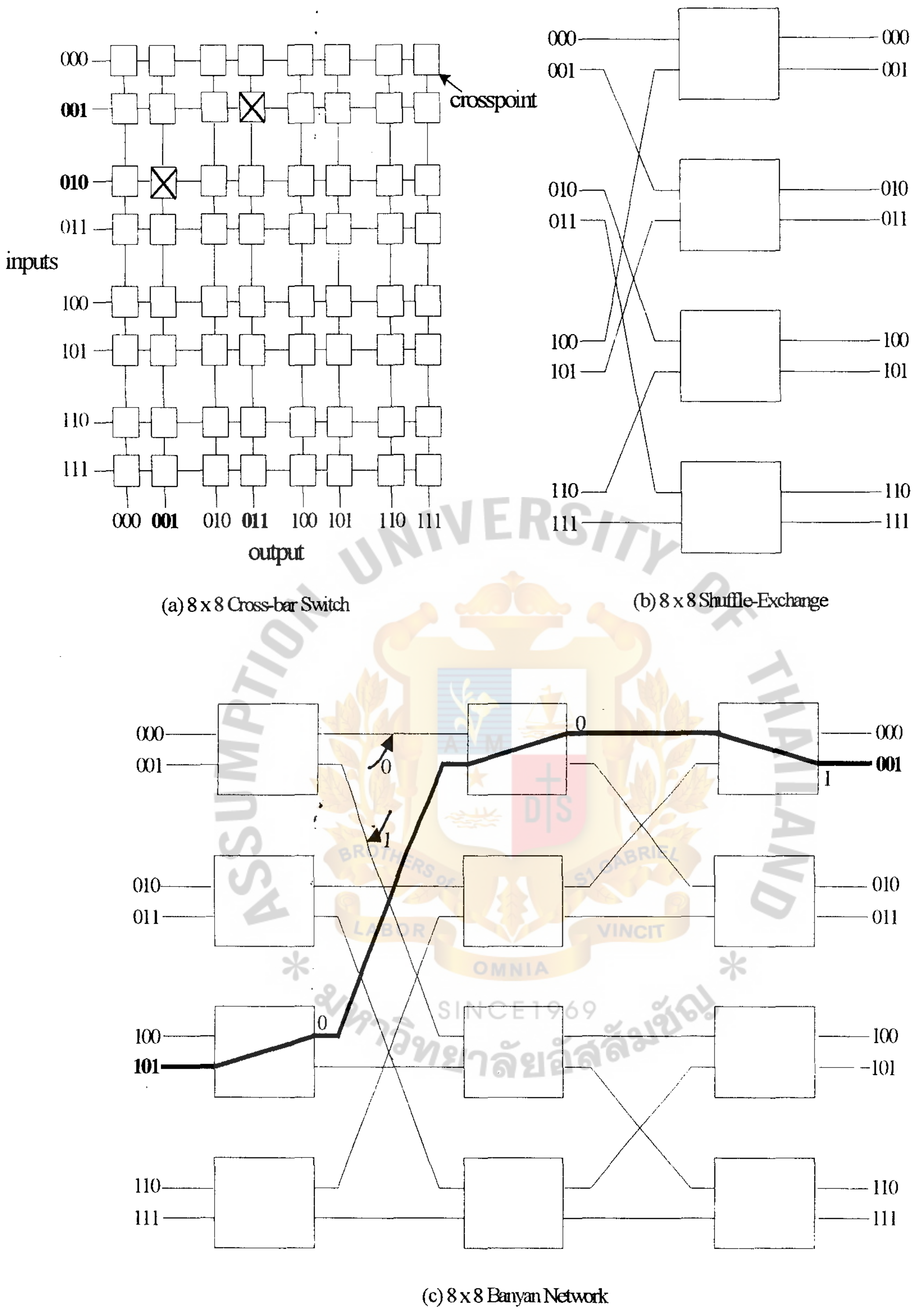


Figure 2.3 Dynamic networks: (a) crossbar switch, (b) single-stage, (c) multistage

Second is the single-stage network. In this case, data must pass repeatedly through the network in order to route them from an input to an output. In other words, the data must enter the network, be passed through the network for the number of times necessary to route them to the correct output, and then exit the network. One well-known example is the "Shuffle-exchange" [Lang 1975; Lang and Stone1976; Lawrie 1975; Stone 1971], as shown in Figure 2.3(b). The network connects each of the inputs on the left side to some outputs on the right side through a single layer of binary switches represented by the rectangles. The binary switches can direct the message on the left-side input to one of two possible outputs that is on the right side. Clearly, the single-stage network topology is limited, however this type of network has become a basis for multistage network, for instance, if we cascade enough single-stage networks together, these networks will form a completely connected multistage network.

The last is the multistage network; most switching networks on now a day consists of several stages, which range from as few as two stages, through some very powerful three-stage networks, or up to networks with a very large number of stages. Formally, the "staging" is as a result from several repercussions. Staging decreases immediately the number of crosspoint required for large N crossbar network. However staging generate the increasing delay, which becomes significant in the many-staged networks that require interconnecting large numbers of inputs and outputs.

One well-known example of multistage networks is Banyan network shown in Figure 2.3(c). The Banyan type network consists of n stages where $N = 2^n$ (N represents the number of input and output nodes). Therefore, each stage may use $N/2$ switch elements, and each switch element is base 2 (i.e., 2×2 switching element). The interconnection patterns from stage to stage determine the network topology. Each stage is connected to the next stage by at least N paths. The network delay is proportional to the number n of stages in a network. The cost of a size $N \times N$ network is proportional to $N \log_2 N$. This class of interconnection networks is also known as a class of cube-type networks.

In addition, the switching elements of Banyan network are controlled as self-routing. This control algorithm determines how the state of each switch will be set. Refer to Figure 2.3(c), it shows the switching decision at an 2×2 elementary switch in a given stage for connecting source S = 101 to destination D = 001 of a Banyan network. To route the signal, the algorithm simply uses the bits of the destination address, 001, as instructions for dynamically selecting a path through the switches. For illustration, the destination address says to set the first two switches to 0 followed by setting the third switch to 1. This path leads to the desired destination. If the bit is 0, the switching element will be set to connect the upper output of switch. In contrast, if the bit is 1, the switching element will be set to connect the lower output of switch. A path from any input side to all destination address from 000 to 111 can be obtained in this fashion. Note that the general state for each switch element can be assumed either the straight or the exchange states. In other words, each switch can be in any one of four legitimate states: straight, exchange, upper broadcast, and lower broadcast which are shown in Figure 2.4.

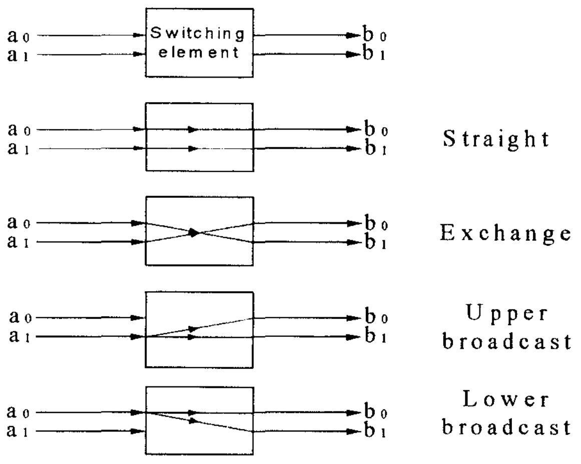


Figure2.4 A two-by-two switching box and its four interconnection states.

2.2 Multistage Interconnection Networks (MINs)

Refer to Figure 2.2, classification of dynamic networks, the multistage networks are further divided into concentrators, i.e., the network interconnects a specific idle input to an arbitrary idle output, and connectors, i.e., the network establishes a path from a specific input to a specific output. However, in this thesis, the multistage networks as connectors are considering, they are called Multistage Interconnection Networks (MINs)

2.2.1 MINs Connection Capability

Formally, MINs can be classified by their ability to establish desired connections between any input and any output. To a large extent, this is the ability to add a new arbitrary connection to a network with arbitrary existing connections. In other words, it is the ability that either a network may allow any arbitrary connection to be established at any time or the establishment of some connection may be blocked by an existing connection which uses a path or contact pair the new connection needs. As depicted in Figure 2.2 MINs are classified further to *nonblocking*, *rearrangeable*, and *blocking* networks.

Firstly, *nonblocking* network, any desired connection between unused ports can be always established immediately without interference from any arbitrary existing connections. All possible permutations (i.e., the connection of a set of sources to a set of destinations) can be provided in a nonblocking network, which sometimes is called the universal network. Secondly, *rearrangeable* network is also a universal network; however, this network may not be always possible to connect an idle pair of terminals without disturbing from any existing connections. That means when the rearrangeable network is given any set of connections in progress and any pair of idle terminals, the existing connections can be reassigned new routes (if necessary) so as to make it possible to connect the idle pair at any time. Lastly,

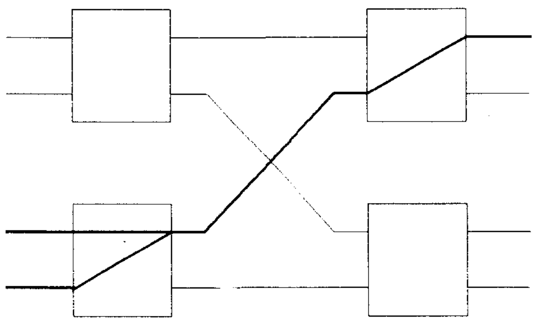


Figure 2.5 Path conflict in the simplest MIN

blocking network, in contrast to universal network, the *blocking* network may not be possible to connect of an idle pair of terminals in any way because in the blocking network, there is one and only one path for each pair. Figure 2.5 shows a path conflict in the simplest MIN.

2.2.2 Main Properties of MINs

A $N \times N$ multistage interconnection networks (MINs) is a communication network with N input terminals (sources) and N output terminals (destinations) composed of a certain number of stages of switching elements (with 2×2 switching elements, the number of stages is usually not less than $\log_2 N$). Each stage is connected to the next stage by at least N data transmission lines between any "source-destination" pair. Each switching element may select from two or more output lines when establishing a connection with an input line.

The simplest MIN (as shown in Figure2.5) consists of the two-input switching elements. It has been shown theoretically that such a two-input switching element ensures the least number of connection points. In addition, the control algorithms for two-input element networks are the simplest. However, component availability considerations may suggest using a MIN in which the number of switchable channels is different from two. Thus, there are large MINs with 16×16 switching elements.

The important properties of MINs include 1) blocking of information in the network; 2) speed, i.e., the rate of transmission of a message from the source to the destination; 3) ease of use, i.e., the degree to which connections are automatically established in the network; 4) partitionability, i.e., the possibility of partitioning the system into subsystems of different size; 5) modularity, i.e., the possibility of constructing the system from a limited number of basic modules; 6) LSI compatibility, i.e., the possibility of implementing the modules on an LSI chip; 7) scalability, i.e., amount the changes needed in order to make the system work with a greater number of inputs/outputs; 8) fault-tolerance, i.e., the ability of the system to remain functional even when some components are faulty (possibly with some degradation of performance).

2.2.3 MINs Application

As it has been noted in the abstract, multistage interconnection networks (MINs) initially were developed for needs of telephony as an alternative to a crossbar, which is impractical for large networks. For the past few decades, the theory and practical implementation of MINs has evolved dramatically. They are widely used as ATM switch fabrics in Broadband Integrated Services Digital Networks (B-ISDN) environment that of supporting a wide range of audio, video, and data services within the same network. Moreover, they are used to be interconnection facilities in parallel computers for delivering information streams to interactive devices.

2.2.3.1 ATM Switches in B-ISDN Environment

The basic concept behind Broadband Services Integrated Digital Network (B-ISDN) is that of supporting a wide range of audio, video, and data services within the same network. ATM (Asynchronous Transfer Mode) has been designated as the target transfer mode approach to providing the desired integration of the various traffic types to be supported by B-ISDN. It is essentially a packet-switched mode of transfer through the network, using short, fixed-length, 53-octet (byte) packets called cells. The cells are assigned on demand at the user network interface (UNI).

Once in the network, each ATM cell moves along the virtual connection associated with the virtual path established end-to-end. The routing function required to move the cells along their respective virtual paths is carried out at the network nodes shown in Figure 2.6. These nodes are called switches in the ATM switches. The prime purpose of the ATM switch is to switch incoming cells arriving on a particular input link to the output link associated with the appropriate virtual path (route). Figure 2.7 is the schematic portrayal of this generic function. In the example of Figure 2.7, cells following VP1 are shown arriving on link 1 and being switched to output link *N*.

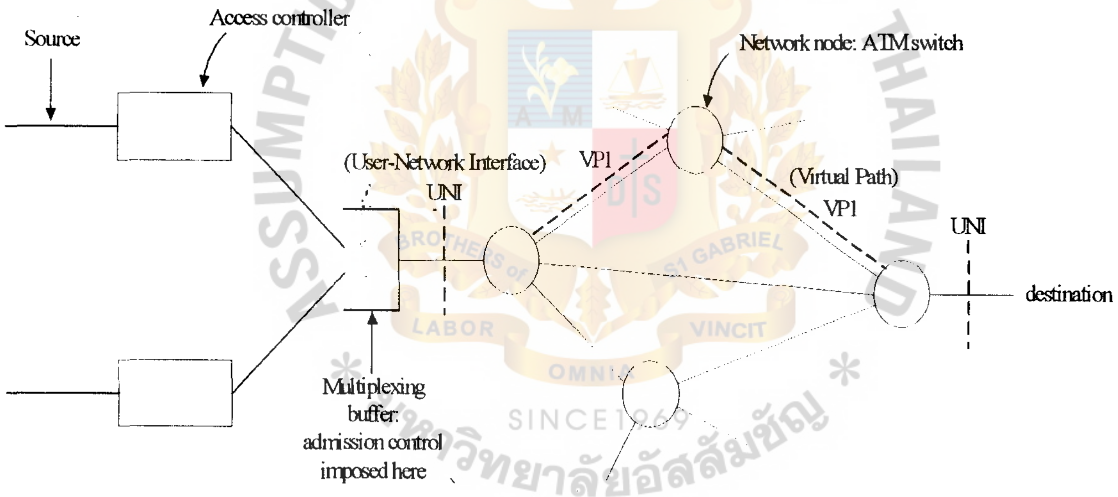


Figure 2.6 Rudimentary ATM network

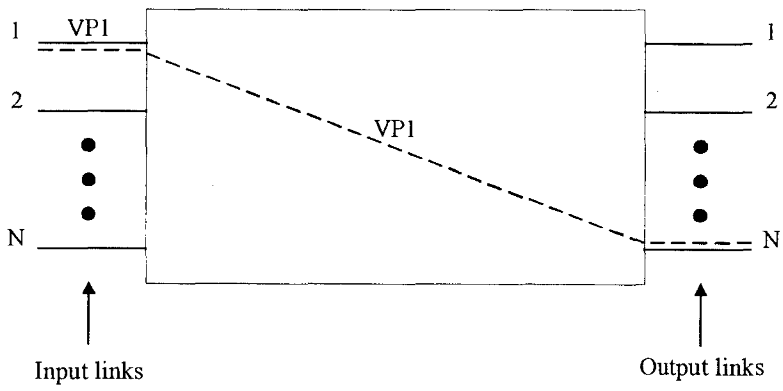


Figure 2.7 ATM switch

Furthermore, the examples of an ATM switch are found in [17], one of them is Batcher-Banyan architecture. In this case, Batcher sorter, which arranges incoming requests to the proper sequence, is followed by a Banyan network. As a while such configuration combines self-routing with non-blocking property.

2.2.3.2 Interconnection Facilities in Parallel Computers

Single instruction stream-multiple data stream (SIMD) multiprocessor computing systems occupy an important place among parallel computing system architectures. In these systems, *N* processors or "processor elements" (PE) execute the same instruction in a synchronized fashion on many data values, where each processor operates on a different subset or partition of the data [5]. For example, if the instruction were **SUB C, D** each PE would subtract its own value of **D** from its own value of **C**. The class of problem that an SIMD machine is especially designed to perform is vector computations over matrices or arrays of data.

A SIMD computer may assume one of two slightly different configurations, as illustrated in Figure 2.8 [16].Configuration I (Figure 2.8(a)), each processor element (PE) has its own individual memory module (M) and is linked by interconnection network to other identical processor elements. On the other hand, configuration II (Figure 2.8(b)), the interconnection network is interposed between the processor elements and the shared memory modules of all the processor elements. It is noticed that each processor element in configuration II can access to any arbitrary shared memory, but each processor in configuration I must access to its individual memory only. However, in configuration II, there is a possibility of memory collision when two or more processors try to access the same memory module, and its interconnection network can introduced the additional delay when accessing the memory.

Additionally, the permutation requests for SIMD computers are typical therefore, the interconnection network, which provides for communication among the processors and memory modules, in SIMD sometimes referred to as an alignment network or permutation network. Note that permutation networks are those that can connect their inputs to their outputs in any arbitrary way as long as no two inputs want the same output, i.e., for an *N* output network, there are *N!* possibilities.

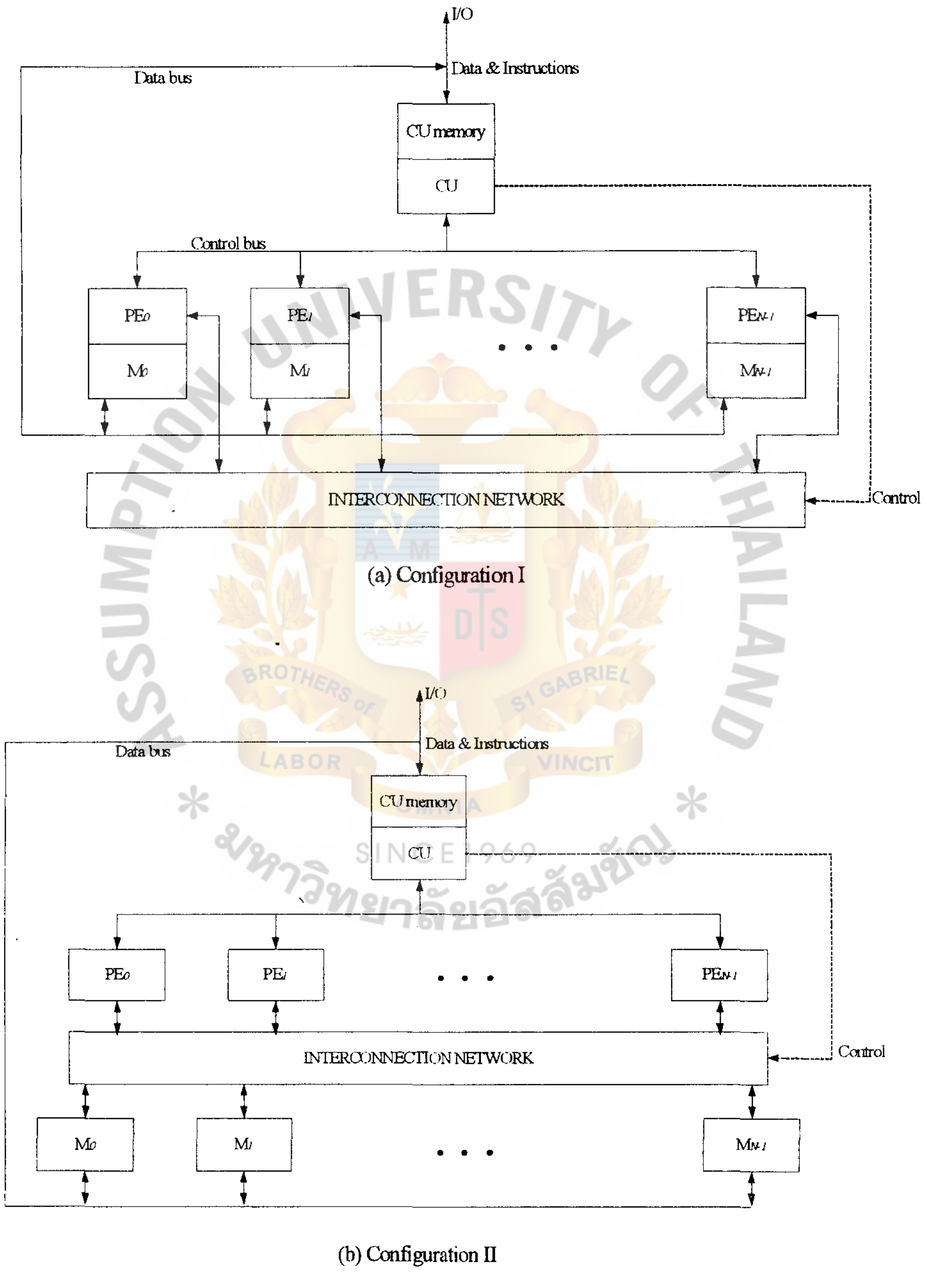


Figure 2.8 Architectural configurations of SIMD computers

Chapter 3

Omega Networks and BPC Permutations

3.1 The 1-path Omega Network

The 1-path Omega network was originally proposed by Duncan H. Lawrie [1] as an alignment network between processors and memories in the array processor systems. It was introduced as an alternative alignment network of the costly crossbar switch and the complicated rearrangeable Benes network (i.e., the rearrangeable, nonblocking multistage interconnection network). One possibility for an alignment network is the traditional *N x M* crossbar switch. This switch can perform any one-to-one of inputs to outputs and with slight modification, it can do one-to-many mapping. The time required to do this mapping is *O(log N)* or *O(log M)* gate delays. However, the number of gates in an *N x M* crossbar is proportional to *N x M*, and this is overly expensive in terms of gates and reliability for using in large systems. Benes network is an another possibility for an alignment network. It has the same capability as a crossbar but only *O (N log N)* gates for an *N x N* network. The time to pass through the network is *O (log N)*. However, it is not easy to set up this algorithm for doing these time units and this is too long to be practical in this application.

In contrast to crossbar and Benes networks, as an alignment in parallel computer, an *N x N* 1-path Omega network can connect *N* processors to *N* memories by using a small number of stages, and each stage composed of fewer than *N* switches. In addition, its use of multiple simple switches can ensure system feasibility, and its use of a few stages with many switching modules can preclude needing a long, inefficient interconnection network pathway.

As it has been noted in introduction, the self-routing algorithm to establish the path for any given source and destination of 1-path Omega network is efficient. Each switch is set itself according to a particular bit in the destination address. However, the ability of fault-tolerance (i.e., the ability of the system to remain functional even when some components failed) of 1-path Omega network is limited because it can provide only one path through the network between any source-destination pair. For example, if there is only failure in a link or a partial failure of a switch the whole system will be destroyed. Furthermore, 1-path Omega network is a blocking network, some permutations, i.e., the connection of a set of sources to a set of destinations, cannot be established by the network.

3.1.1 Network Structure

An $N \times N$ 1-path Omega network consists of $n = log_2N$ identical stages, where N is the number of inputs (outputs). Each stage consists of a *perfect shuffle* interconnection followed by $N/2$ switching elements. The perfect shuffle interconnection has the property taking an input at a position whose binary representation is $s_0\ s_1 \ldots s_{n-1}$, and moving it to position $s_1 \ldots s_{n-1}\ s_0$. For illustration, Figure 3.1 and Figure 3.2 represent the perfect shuffle interconnection and the 1-path Omega network for $N = 8$ respectively.

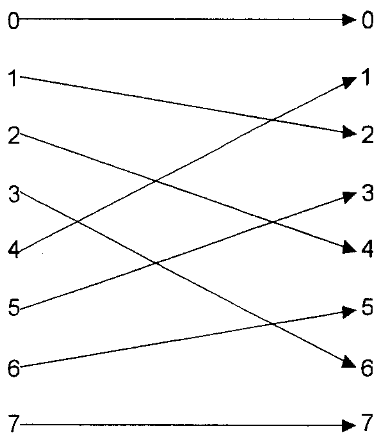


Figure 3.1 The perfect shuffle interconnection

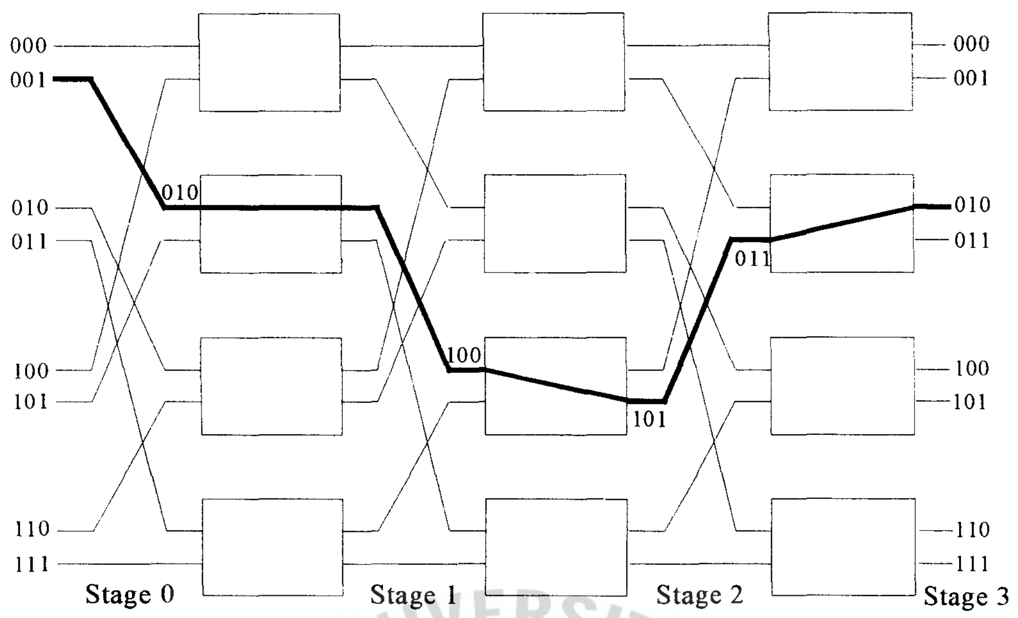


Figure 3.2 A 1-path Omega network with N=8

3.1.2 Routing Tag Control

The switching elements in 1-path Omega network are controlled individually. In order to pass data from the networks inputs to the network outputs, it is necessary to have the routing algorithm for setting the stages of the switches. Refer to Figure 2.4, each switch can have one of four possible connection states. The network may send its input *straight* through, *interchange* the inputs or *broadcast* one of the inputs to both outputs. However, to switch both inputs to the same output is not allowed because that means data are in a conflict.

As it has been mentioned, the 1-path Omega network is a subclass network of Banyan type. They have an equivalent topology which is basing on 2×2 switching elements of the self-routing multistage network (Note that the only difference between 1-path Omega network and Banyan network is their internal switch connections).Therefore, the self-routing algorithm of Banyan network is maintained as the one of 1-path Omega network

Let $S = s_0 s_1 \ldots s_{n-1}$ be the *source tag*, i.e., the binary representation of the input number, and let $D = d_0\, d_1 \ldots d_{n-1}$ be the *destination tag*, i.e., the binary representation of the desired output number. The self-routing algorithm is as simply uses the bits of the desired output number as instructions to control the successive stages in order. The input of the switch on the i^{th} stage, where i is between 0 and n-1, is connected to the upper output if $d_i = 0$; and to the lower output if $d_i = 1$. For an illustration, Figure 3.2, shows a path between source $S = 001$ and destination $D = 010$.

3.1.3 Information Blocking

To set up a particular path from any source to any destination in 1-path Omega network, it just simply follows the above procedure simultaneously for all sources or all destinations. It can be observed that the algorithm will choose the unique set of paths in the network for any given mapping. In addition, since the paths in a set will not necessarily be disjoint, a "blocking" situation (due to conflicts) may be presented to any source-destination pair. Note that, to continue accessing the blocked message when a conflict arising in the network, the blocked message must be waited until the other has completed its transmission.

A blocking condition generally happens when the messages that exist on the two input links of a switch want to go out the same output link, as the result, two different signals sharing a common wire. In fact, the two different signals are not allowed to share a common wire, they are allowed to share a common wire if and only if both signals have a common source, in which case, it can be said that they are identical. Figure 3.3 shows the conflicted paths which are established for the mapping 001$\rightarrow$000, 101$\rightarrow$011, which are sharing a common connection at the output of the first stage.

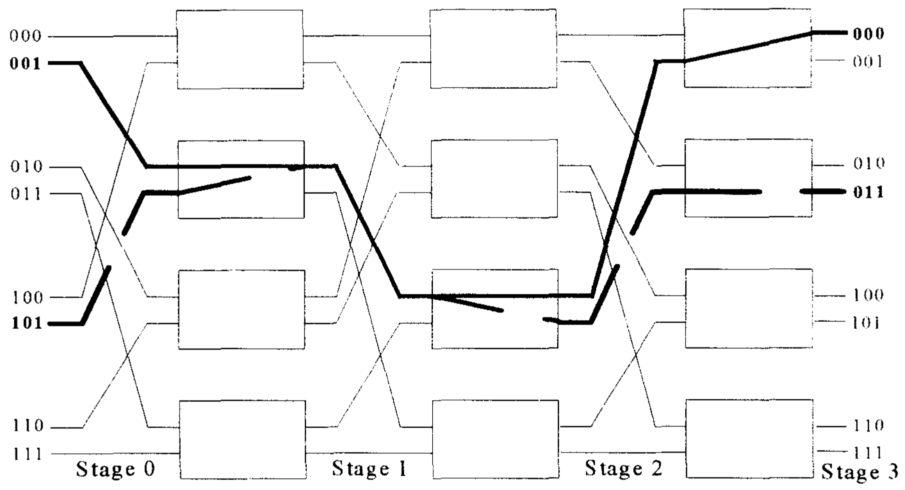


Figure 3.3 Path for (001,000), (101,011) in A 1-Path 8 x 8 Omega network

3.2 The R-Path Omega Network

K. Padmanabhan and D. Lawrie proposed ***Modified Omega Network or R-path Omega network*** [2], in order to overcome the fault-tolerance limitation of the 1-path Omega network by using basic switches with the number of input and output greater than 2. The R-path Omega network has been derived from the 1-path Omega network; therefore, it can maintain all connection properties of the original 1-path Omega network. The fault-tolerance of R-path Omega network is achieved in the form of being a multipath network, i.e., R-disjoint connection paths for every source-destination pair are provided through the *R* sets of intermediate switches and links. Figure 3.4 shows two different paths from source $S = 010$ to destination $D = 011$ in 8 x 8 2-path Omega network. The failure in one of these two sets of intermediate switches or links will not prevent source $S = 010$ from accessing destination $D = 011$. For the permutation admissibility, the authors of R-path Omega network proved that the permutation capability of a multipath network is not worse than that of a 1-path Omega network; R-path network would pass any permutation (π) that the 1-path

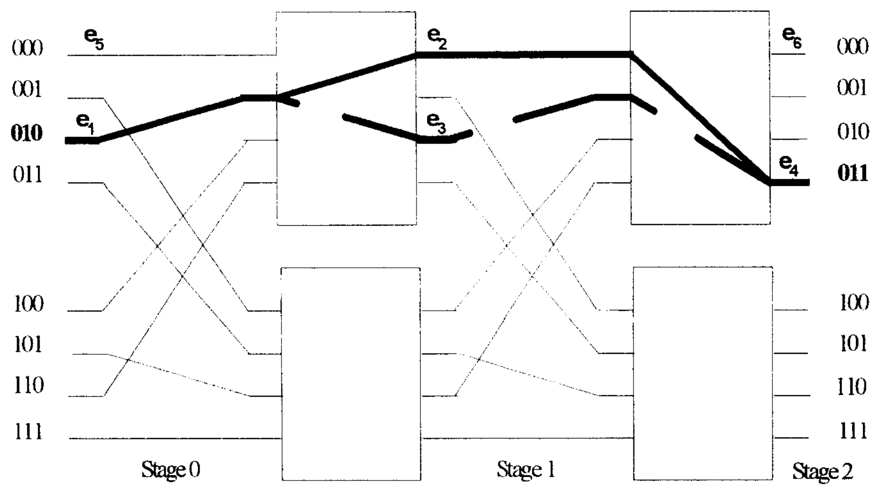


Figure 3.4 Path for (010,011) in a 2-Path 8 x 8 Omega network

network passes, and no conflict could occur in R-path network if none occurred in 1-path network.

3.2.1 Network Structure

An $N \times N$ R-path Omega network consists of $\lceil \log_B N \rceil$ identical stages of $B \times B$ switching elements, where each stage consists of N/B switches, it is a uniform network with all switches of size B. In fact, the R-path Omega network does not require N to be a power of B, because when $N = B^d$ the network will equivalent to 1-path Omega network. Stages of R-path Omega network are interconnected by the B * N/B shuffle. A B * N/B shuffle is the permutation of elements as defined in equation (3.1), where i is any input *terminal*, and $\pi(i)$ is the output terminal to which is linked the input terminal in a given stage (or between two columns of switches). For example, Figure 3.4 shows that in stage 0 the input terminal $i = 2$ (or 010) is linked to the output terminal $\pi(2) = 1$ (or 001).

$$\pi(i) = \left(Bi + \left\lceil \frac{i}{N/B} \right\rceil \right)_{\text{mod}\,N} \tag{3.1}$$

3.2.2 Establishing A Path

In an R-path Omega network, every source will keep track of the set of paths, which is using to get to the destinations, but when the source receives notice of a fault in one of those paths, the alternative of (R-1) sets of paths is selected. Let $N = 2^n$ and $B = 2^b$. The N x N R-path Omega network using with B x B switching element has R redundant paths where $R = B^{\lceil n/b \rceil}$. The expression (3.2) shows the general pattern of an entire path of every source-destination pair.

$$s_0 s_1 \ldots s_{n-1} \otimes_1 \otimes_2 \ldots \otimes_r \, d_0 d_1 \ldots d_{n-1} \tag{3.2}$$

It is easily seen that the entire path in (3.2) consists of source tag $s_0 s_1 \ldots s_{n-1}$ (i.e., the binary representation of the input number), extra bits $\otimes_1 \otimes_2 \ldots \otimes_r$ (i.e., the binary representation that use to set a switch), and destination tag $d_0 d_1 \ldots d_{n-1}$ (i.e., the binary representation of the output number to which input number S is connected). The extra bits (denoted by $\otimes$'s) are used to set a switch in one stage in order to selecting one route among the set of alternative redundant path for a given input-output pair. The number of the extra bits, r, equals $\log_2 R$. Note that, there is no extra bit consisting in 1-path Omega network because there are no redundant path providing connectivity in 1-path Omega network.

$$s_0 s_1 \ldots \boxed{s_{bi} s_{bi+1} \ldots s_{n-1} \otimes_1 \otimes_2 \ldots \otimes_r \, ..d_{bi-r-1}} \ldots d_{n-1} \tag{3.3}$$

The output terminal (of a switch) that a path occupies at stage $0 \le i \le \lceil \log_B N \rceil$ is generally given by the n-bit window with starting at bit position bi as indicated in the box shown in (3.3). It can be noticed that the $\log_2 R$ extra bits ($\otimes_1 \otimes_2 \ldots \otimes_r$) are as a part of every window except the input and output window [2]. For illustration, refer to Figure 3.4, the terminals occupy at each stage of the path $e_1 - e_2 - e_4$, and $e_1 - e_3 - e_4$, which connecting source S = 010 to destination D = 011 are shown in Table I, and Table II. Note that the extra bit, $\otimes$, is set to be 0 and 1 for path $e_1 - e_2 - e_4$ in Table I, and path $e_1 - e_3 - e_4$ in Table II respectively.

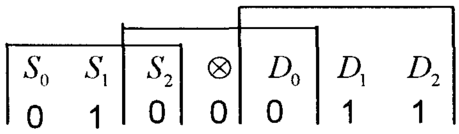


TABLE I

THE TERMINALS AT DIFFERENT STAGES OF PATH (01000, 00110) WHEN EXTRA BIT IS 0

Stage (i)	Terminal
0 (source)	010 $(e_1 = S_0S_1S_2)$
1	000 $(e_2 = S_2 \otimes D_0)$
2 (destination)	011 $(e_4 = D_0D_1D_2)$

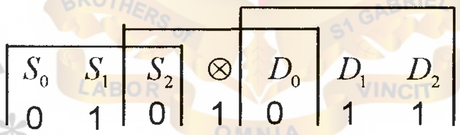


TABLE II

THE TERMINALS AT DIFFERENT STAGES OF PATH (01000, 00110) WHEN EXTRA BIT IS 1

Stage (i)	Terminal
0 (source)	010 $(e_1 = S_0S_1S_2)$
1	010 $(e_3 = S_2 \otimes D_0)$
2 (destination)	011 $(e_4 = D_0D_1D_2)$

It is noted that when a source is forced to take an alternate path because a "regular" one is down, it could interfere with (or block) another input that would normally have taken that path. Therefore, to avoid this from happening under no-fault conditions, it has to be ensured that certain sources follow only certain paths and not others. For instance in Figure 3.4, when the connections 0-0 and 2-3 are desired, the two connections cannot simultaneously use the paths $e_5 - e_2 - e_6$ and $e_1 - e_2 - e_4$ because there will be the conflict at link e_2.

The procedure of the design of an R-path $N \times N$ Omega network is given in [2]. The possible values of redundancy that can be obtained in a uniform $N \times N$ network and the sizes of the switches to be used are presented in Table III.

TABLE III

POSIIBLE VALUES OF REDUNDANCY IN UNIFORM NETWORKS

N	*R*	*B*(smallest)
2	1	2
4	1	2
8	2,1	4,2
16	4,1	8,2
32	8,2,1	16,4,2
64	16,4,1	32,16,2
128	32,8,4,2,1	64,32,8,4,2
256	64,16,4,2,1	128,64,32,8,2
512	128,32,8,2,1	256,128,16,4,2
1024	256,64,16,4,1	512,256,128,8,2
2018	512,128,32,8,2,1	1024,512,256,128,4,2

3.2.3 Permutation Capability

Permutation capability, which can loosely defined as the blocking probability of the network, of R-path Omega is concerned in [2]. There are two aspects to the operation of the R-path network under a no-fault situation. First, an R-path network should be able to pass every permutation that a 1-path network does. Second, the blocking (due to conflicts) in a multipath network should be no more than that in the corresponding 1-path network.

The authors of *R*-path Omega network had an observation on the Omega permutation capability that one way to ensure that a *R*-path Omega network passes every permutation which a 1-path Omega network does is to ensure that an input-output path occupies the same terminal at the output of first stage in the *R*-path Omega network as it does in the 1-path Omega network. In other words, in an R-path network, a packet (or path being created) on leaving the source could fork in one of R ways at the first stage. Upon leaving the first stage of switching elements, there is exactly one path the packet has to follow to reach the destination. If this portion of the path can be made identical to that in the 1-path network, the permutation will be passed by the multipath network. For example, consider 1-path and 2-path Omega network with size $N = 8$. Path between input S and output D in the two networks are given by the followings.

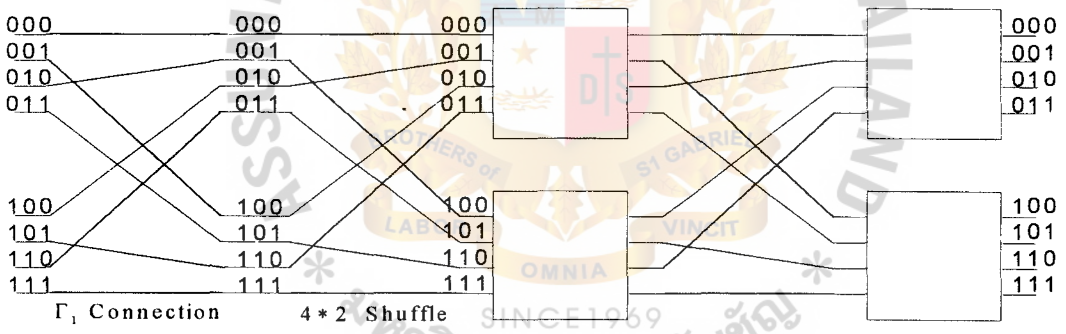
$$s_0\boxed{s_1s_2d_0}d_1d_2 \qquad \text{1-path network}$$
$$s_0s_1\boxed{s_2 \otimes d_0}d_1d_2 \qquad \text{2-path network}$$

Terminals that these paths occupy at the output of first stage are indicated by the boxes. From this, it can be seen that if the source address bits in the 2-path network were rotated right by one position ($s_0s_1s_2 \Rightarrow s_2s_0s_1$) and set $\otimes = s_2$, the two windows would match as follow.

$$s_0\boxed{s_1s_2d_0}d_1d_2 \qquad \text{1-path network}$$
$$s_2s_0\boxed{s_1s_2d_0}d_1d_2 \qquad \text{2-path network}$$

In addition, all following windows would also match in the two paths meaning the paths would be identical in the two networks beyond this point. Rotating the source bits is a fixed permutation that can be accomplished by a connection in front of the network. With this permutation, source $s_0s_1s_2$ is connected to terminal $s_2s_0s_1$ at the input to the first stage of the network. If this source sets the extra tag bit $\otimes$ to s_2, the path from the input of the network to the output is given by $s_2s_0s_1d_0d_1d_2$ and this path occupies the same terminals at the output of each stage as does the path in the 1-path network.

From the above idea, authors of R-path Omega network generated the theorem. Denote the permutation obtained by a k-bit right rotation by Γ_k. The following theorem explains that an R-path Omega network (with a bit–rotate permutation in front) in a fault-free situation will be able to emulate a 1-path network if sources use their r least significant bits as the extra tag bits. A $B*N/B$ shuffle is equivalent to rotating the source address bits b positions to the left while a Γ_k permutation rotates it r positions to the right. Hence, the net result is a b-r bit left rotate in front of the first stage of switches. If $R = B/2$ ($r = b$-1), this is the perfect shuffle. An example of a 2-path network realizing the identity permutation is shown in Figure 3.5. It can be seen that when tags are set as in theorem, the settings of the switches (in all stages except the first) are identical to those in the corresponding 1-path network.



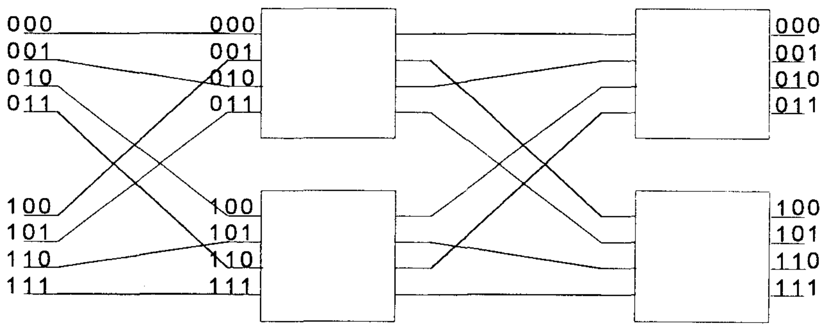


Figure 3.5 A 2-path 16 x16 Omega network with switches set up to realize the identify permutation

Theorem: Let $\log R = r$, where R is the number of redundant path of network. If a multipath Omega network is constructed with a Γ_k permutation in front and every source $s_0 s_1 \cdots s_{n-1}$ employs the tag $s_{n-r} \cdots s_{n-1} d_0 d_1 \cdots d_{n-1}$ to access the destination $d_0 d_1 \cdots d_{n-1}$, then the multipath Omega network will, in a no-fault condition, pass every permutation that the corresponding 1-path Omega network does.

Proof: Consider a permutation $\pi = \{(S, D)\}$ that is passed by the 1-path Omega network, and a path $S - D$. The Γ_k permutation in the multipath network takes source $s_0 s_1 \cdots s_{n-r} \cdots s_{n-1}$ to $s_{n-r} \cdots s_{n-1} s_0 \cdots s_{n-r-1}$. Following this permutation, a path in the network is given by following, where the extra tag bits $\otimes$'s are set as in the statement of the theorem.

$$s_{n-r} \cdots s_{n-1} s_0 s_1 \cdots \underbrace{s_{n-r} \cdots s_{n-1}}_{\otimes\otimes\cdots\otimes} d_0 \cdots d_{n-1}$$

As observed in the proof of Theorem, the output terminal such a path will occupy in stage 1 is given by the ***n***-bit window.

$$s_{b-r} \cdots s_{n-1} d_0 \cdots d_{b-r-1} \tag{3.4}$$

It can be verified that $b - r = \log k$ where k is $k = \frac{N}{B^{\lfloor \log_B N \rfloor}}$, so that (3.4) is also the terminal that the path from S to D would occupy in a 1-path Omega network at the output of stage 1. This is true of every output terminal (and every input-output path) in stage 1 so that no two paths would occupy the same terminal in stage 1. Successive windows corresponding to terminals in the remaining stages are identical in the two networks so that no conflict can occur in the multipath network if none occurs in the 1-path network. Thus, the multipath network will pass any permutation π that the 1-path network passes.

3.3 BPC Permutation

As it was stated above, a permutation refers to the connection of a set of sources to a set of destinations such that each source is connected to a single destination. A permutation is said to be admissible to a network, only when all N source-destination pairs are conflict free. Otherwise, such permutation is non-admissible and the conflict or blocking occurs.

The class of permutation that we are considering is ***Bit Permutation Complement (BPC)***. Recall that a permutation is called BPC permutation if the destination tag can be obtained of the source tag by permuting and/or complementing some or all of its bit positions [13]. Frequently used permutations of this class usually have names and they are listed together with their names, as below where each equation shows the mapping of a source $(s_0 s_1 s_2 \cdots s_{n-2} s_{n-1})$ to the destination.

Bit reversal:

$$\pi_{\text{bit reversal}} = (s_{n-1} s_{n-2} \cdots s_1 s_0)$$

Matrix transposition:

$$\pi_{\substack{\text{matrix}\\ \text{transposition}}} = \begin{cases} s_l s_{l+1} \cdots s_{2l-1} s_0 s_1 \cdots s_{l-1} & \text{if } n = 2l \\ s_l s_{l+1} \cdots s_{2l} s_0 s_1 \cdots s_{l-1} & \text{if } n = 2l+1 \end{cases}$$

Perfect shuffle:

$$\pi_{\text{perfect shuffle}} = s_1 s_2 \cdots s_{n-1} s_0$$

Vector reversal:

$$\pi_{\text{vector reversal}} = \overline{s}_0 \overline{s}_1 \cdots \overline{s}_{n-2} \overline{s}_{n-1}$$

Bit shuffle:

$$\pi_{\substack{\text{bit}\\ \text{shuffle}}} = \begin{cases} s_0 s_2 \cdots s_{n-2} s_1 s_3 \cdots s_{n-1} & \text{if } n = 2l \\ s_0 s_2 \cdots s_{n-1} s_1 s_3 \cdots s_{n-2} & \text{if } n = 2l+1 \end{cases}$$

Unshuffle:

$$\pi_{\text{unshuffle}} = \left(s_{n-1}s_0 \cdots s_{n-3}s_{n-2}\right)$$

Shuffle row major:

$$\pi_{\substack{\text{shuffle}\\ \text{row major}}} = \begin{cases} s_0 s_l s_1 s_{l+1} \cdots s_{l-1} s_{2l-1} & \text{if } n = 2l \\ s_0 s_{l+1} s_1 s_{l+2} \cdots s_{l-1} s_{2l} s_l & \text{if } n = 2l+1 \end{cases}$$

Butterfly:

$$\pi_{\text{butterfly}} = \left(s_{n-1}s_1 \cdots s_{n-2}s_0\right)$$

Exchange:

$$\pi_{\text{exchange}} = \left(s_0 s_1 \cdots s_{i-1}\overline{s}_i s_{i+1} \cdots s_{n-2}s_{n-1}\right)$$

Chapter 4

Exploring of BPC Permutation Admissibility to R-path Omega Network

4.1 BPC Permutation Admissibility

The problem, which is being solved in this section, is most closely resemble the problem of permutation admissibility (*PA*) solved by Xiaojun Shen in [12]. However Shen's investigations are in concern with another class of interconnection network, namely with *k*-Extra Stage Multistage Cube-Type Networks (*k*-EMCTN), where redundant disjoint paths for any source-destination pair are provided by adding *k* more stages in front of an Omega network implemented of switches with size 2×2. In this works mentioned above Shen resorts to the graph theory. We base our conclusions on rather simple and evident considerations.

As it has been already noted in section 3.3, for a BPC (Bit Permute Complement) permutation, the destination address is obtained by permuting the bits in the binary representation of the source in accordance with a given rule and/or complementing some of these bits. Thus for a BPC permutation π the transition sequence in a general case looks as follow

$$S_0 S_1 \cdots S_{n-2} S_{n-1} \otimes \cdots \otimes S_{\pi(0)} S_{\pi(1)} \cdots S_{\pi(n-2)} S_{\pi(n-1)}$$

with some or all components of the destination, part of the sequences may be complemented but for our approach, it makes no difference as it can be seen from the further discussion. It should be recalled that when routing each window defines terminal numbers in one of stages of the network. So to avoid blocking each window should allow to form different terminal numbers in a given stage for all input-output pairs but it is possible only if there are no components with the same subscripts inside the window, otherwise the

complete set of terminal numbers cannot be formed. Therefore, we receive a straightforward way to check the admissibility of any BPC permutation to an R-path Omega network of a given structure. It is quite evident that in this case only replica of components with the same subscripts within a window is of importance, it makes no difference whether such two components are mutual complements or not.

For illustration, we shall try *perfect shuffle* permutation, i.e., the frequently used permutation in parallel programming destination address that are derived from source addressed by cyclic shifting the latter to the left by one bit position, on both 1-path and 2-path Omega networks with using the foregoing approach.

Example 4-1 The 1-path Omega Network

Test the *perfect shuffle* admissibility to a 1-path Omega network using with $N = 32$ and $n = 5$, $B = 2$ and $b = 1$ respectively. Number of stage K equals 5, and the transition sequence does not contain extra bits as follows

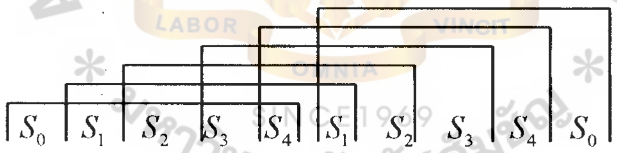


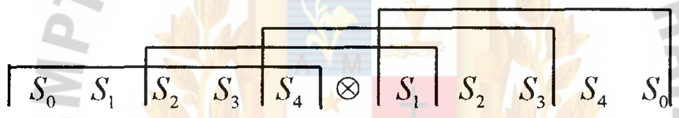
TABLE IV
THE TERMINALS AT DIFFERENTSTAGES OF *PERFECT SHUFFLE* PERMUTATION FOR 1-PATH OMEGA NETWORK WITH N = 32 AND B = 2

Stage (i)	Terminal
0	$S_0S_1S_2S_3S_4$
1	$S_1S_2S_3S_4S_1$
2	$S_2S_3S_4S_1S_2$
3	$S_3S_4S_1S_2S_3$
4	$S_4S_1S_2S_3S_4$
5	$S_1S_2S_3S_4S_0$

Application of routing with moving window to this case results in Table IV. (Stage 0 in what follows will be taken to mean input terminals whereas stages consisting of switches begin with stage 1). From the Table IV, it can be seen that terminals in stages 1, 2, 3, and 4 contain components with coinciding transcripts. Therefore, in this case of 1-path Omega network, *perfect shuffle* permutation is *non-admissible*.

Example 4-2 The 2-path Omega Network

Test *perfect shuffle* admissibility to a 2-path Omega network using the foregoing approach. Now let $N = 32$ or $n = 5$, $R = 2$, $B = 4$ or $b = 2$. Number of stages K here equals 4 and only one extra bit is needed. The transition sequence for perfect shuffle in this case looks as follows.



$$S_0 \; S_1 \; S_2 \; S_3 \; S_4 \otimes S_1 \; S_2 \; S_3 \; S_4 \; S_0$$

TABLE V
THE TERMINALS AT DIFFERENTSTAGES OF *PERFECT SHUFFLE* PERMUTATION FOR 2-PATH OMEGA NETWORK WITH $N = 32$ AND $B = 4$

Stage (i)	Terminal
0	$S_0S_1S_2S_3S_4$
1	$S_2S_3S_4 \otimes S_1$
2	$S_4 \otimes S_1S_2S_3$
3	$S_1S_2S_3S_4S_0$

Moving the window of length five in the above transition sequence starting at each step in $2i$ position results in symbolic terminal numbers shown in Table V. Here, no one terminal number contains components with the same subscript so the proper choice of extra bit values may provide the complete set of 5-bit binary numbers corresponding to the physical terminals in each stage. Therefore, in this case of 2-path Omega network, *perfect shuffle* permutation is *admissible*.

For convincing, Figure 4.1 displays occurrence of blocking when trying to realize the perfect shuffle permutation with 1-path 32x32 Omega network. The same permutation is admissible to 2-path 32x32 Omega network (Figure 4.2). Our analysis shows that it is admissible to any R-path Omega network with R>1. This finding is of fundamental importance: perfect shuffle is commonly used in parallel programming for matrix transposition, for fast Fourier transform, for various sorting procedures, for polynomial evaluation etc.

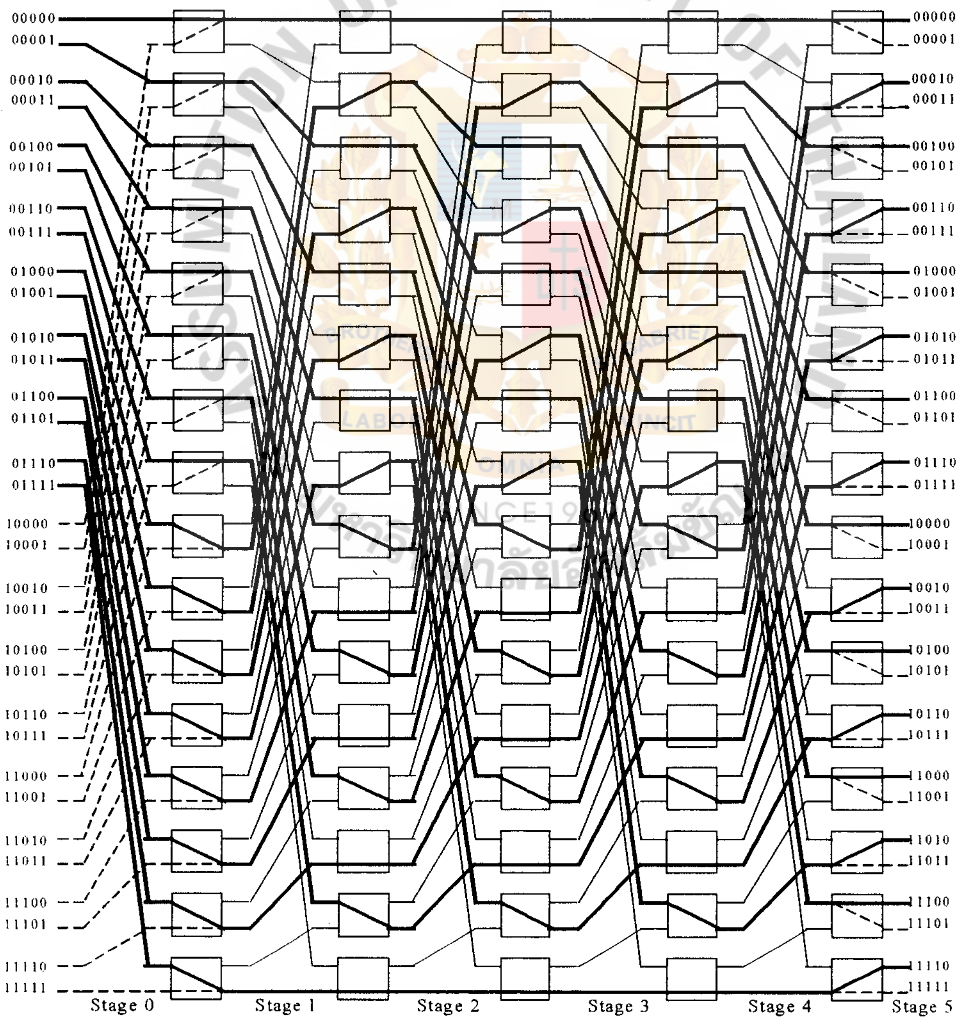


Figure 4.1 A 1-path 32x32 Omega network when realize perfect shuffle permutation

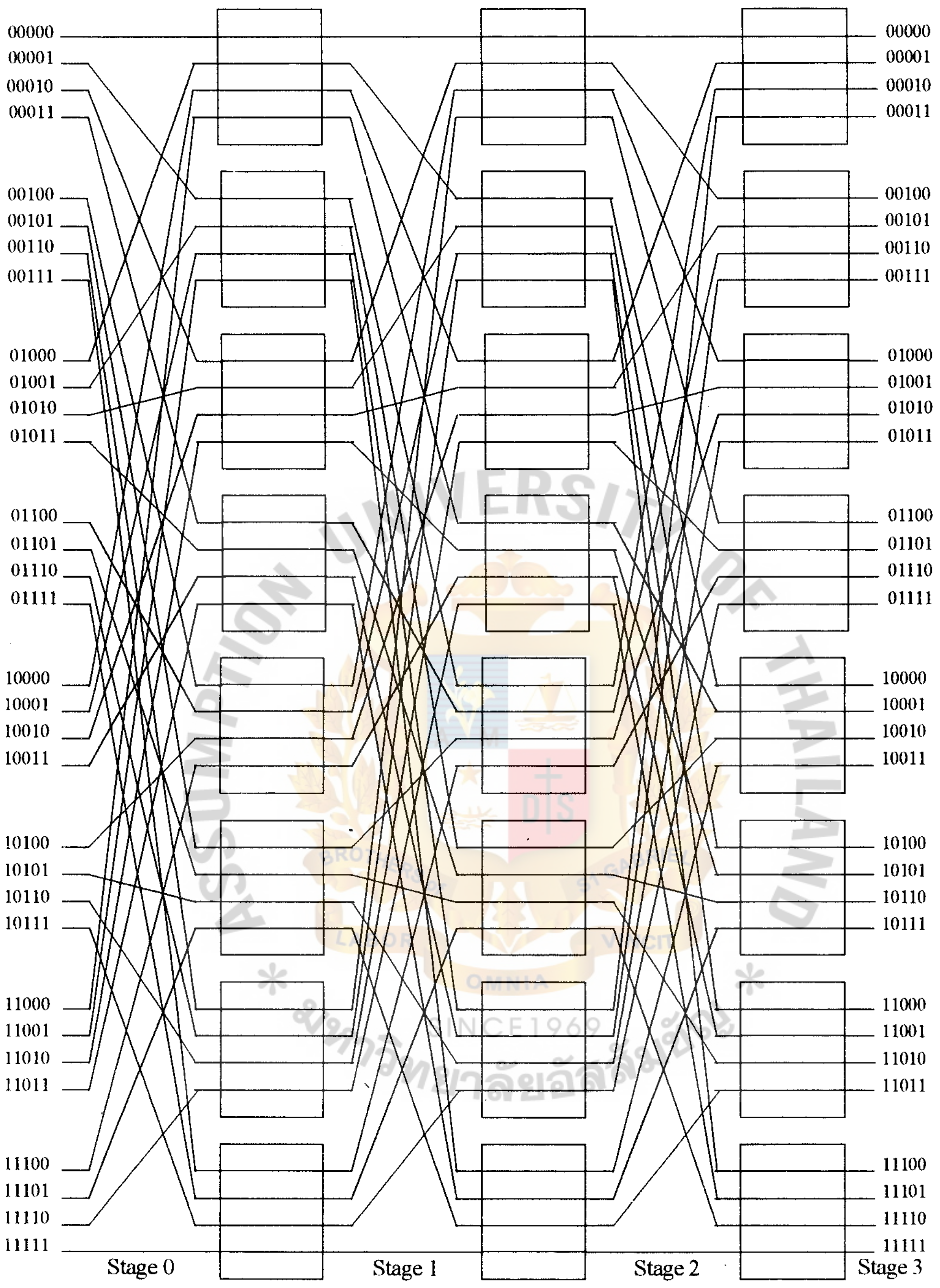


Figure 4.2 A 2-path 32x32 Omega network when realize perfect shuffle permutation

4.2 Admissibility Algorithm

The transition sequence for a BPC permutation can be rewritten as $s_0 s_1 \ldots \boxed{s_{bi} s_{bi+1} \ldots s_{n-1} \otimes_1 \otimes_2 \ldots \otimes_r \ldots d_{bi-r-1}} \ldots d_{n-1}$. The sequence as indicated in the box is the general tag sequence for every window except input and output windows. It can be observed that s_{bi} is the first of source tags, s_{n-1} is the last of source tags, and the last of destination tags is d_{bi-r-1}. Note that i is the indicator of the window number of each stage, where $0 \leq i < K$. B is size of switch, where $B = 2^b$. N is size of network, where $N = 2^n$. R is number of redundant path, where $R = 2^r$, and K is the total number of identical stage of R-path Omega network, where $K = \lceil \log_B N \rceil$.

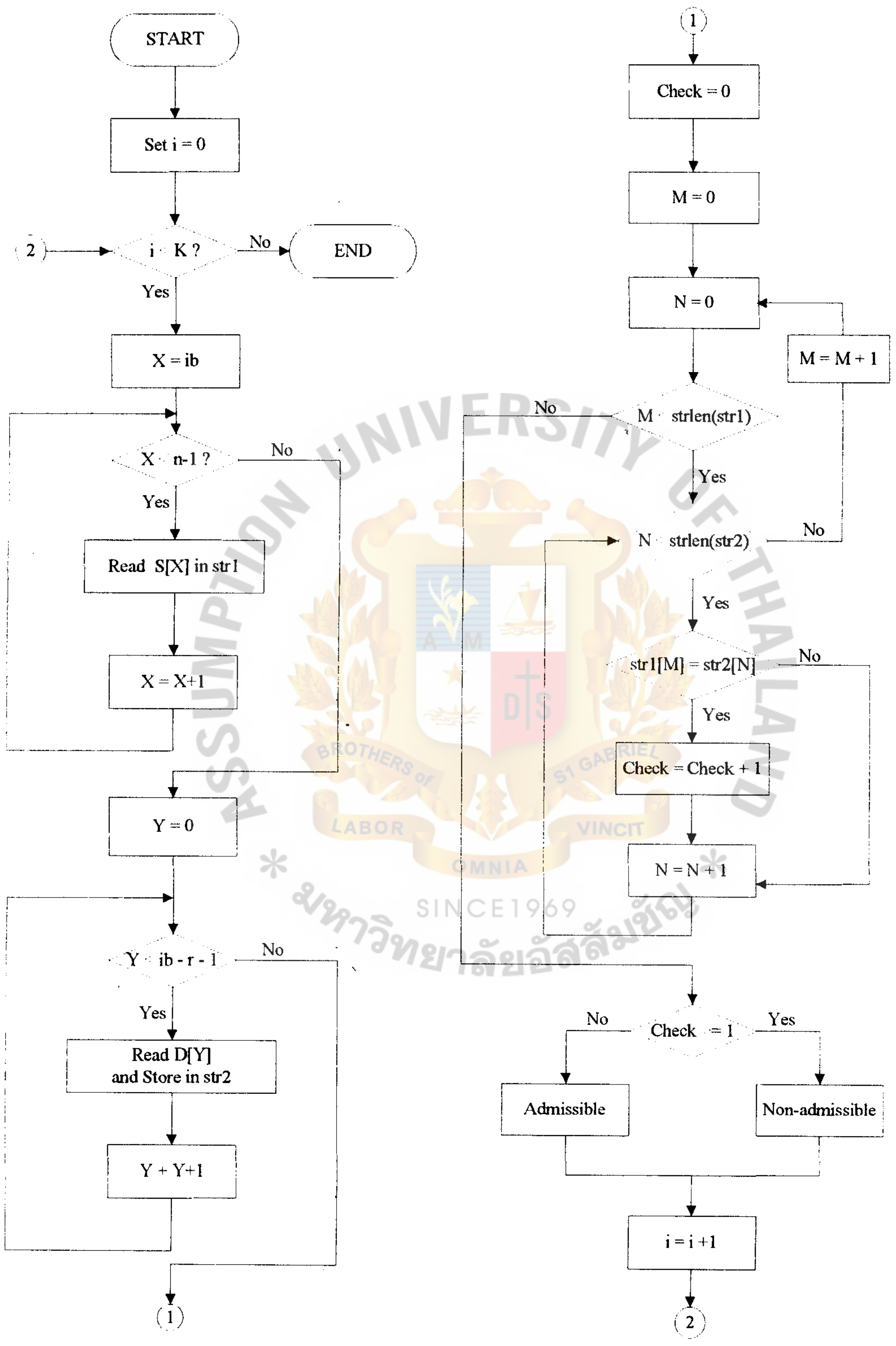
To check permutation admissibility, it is starting at window i = 0. While i is less than K, the bit of source tags, $s_{bi} s_{bi+1} \ldots s_{n-1}$ are read and stored into str1 one by one. At the same time, each bit of destination tags, which always end at d_{bi-r-1}, is read and stored into str2. Next step, the subscripts of source and destination tags inside the window i are checked one to one whether they are same. In checking, the subscript of first source tag (M = 0), which has been in str1, is foremost compared to the first destination tag (N = 0), which has been in str2. If their subscripts are same, Check = Check +1. If else, the subscript of the same first source tag will continue comparing to one of the next destination tag (N = N+1), and this will keep on until the last destination tag is investigated. After that, the next source tag (M = M+1) is counted, and the above comparing procedure is repeated until subscript of the last source tag is compared to one of the last destination tag. Finally, the result is concluded. If Check $\geq$ 1 that means the permutation is non-admissible, if else, it is admissible. This procedure is repeated for the next window ($i = i+1$) and ended up when all components in the last window ($i = K$-1) is inspected.

Algorithm: *Admissibility*

```
Input:  array P[$s_0 s_1 \ldots s_{bi} s_{bi+1} \ldots s_{n-1} \otimes_1 \otimes_2 \ldots \otimes_r \ldots d_{bi-r-1} \ldots d_{n-1}$]

Output: "admissible", "non-admissible"

Set  $i = 0$

While ($i < K$)

          x = bi

          While (x $\leq$ n-1)

                    Read s[x] and Store in str 1

                    x = x + 1

          End Do

          Set  y = 0

          While (y $\leq$ bi-r-1)

                    Read d[y] and Store in str 2

                    y = y + 1

          End Do

                    Set  check = 0

                    Set  M = 0

                    Set  N = 0

                    While (M < strlen (str1))

                              Do

                                        If  str1[M] = str2[N]

                                        Then  check = check + 1

                                        End if

                                        N = N+1

                              While (N < strlen (str2))
```

```
                End Do
                M = M+1
                Set  N = 0
                If (check ≥ 1)
                Then  Non-admissible
                Else   Admissible
                End if
            End Do
            $i = i+1$
        End Do
End
```

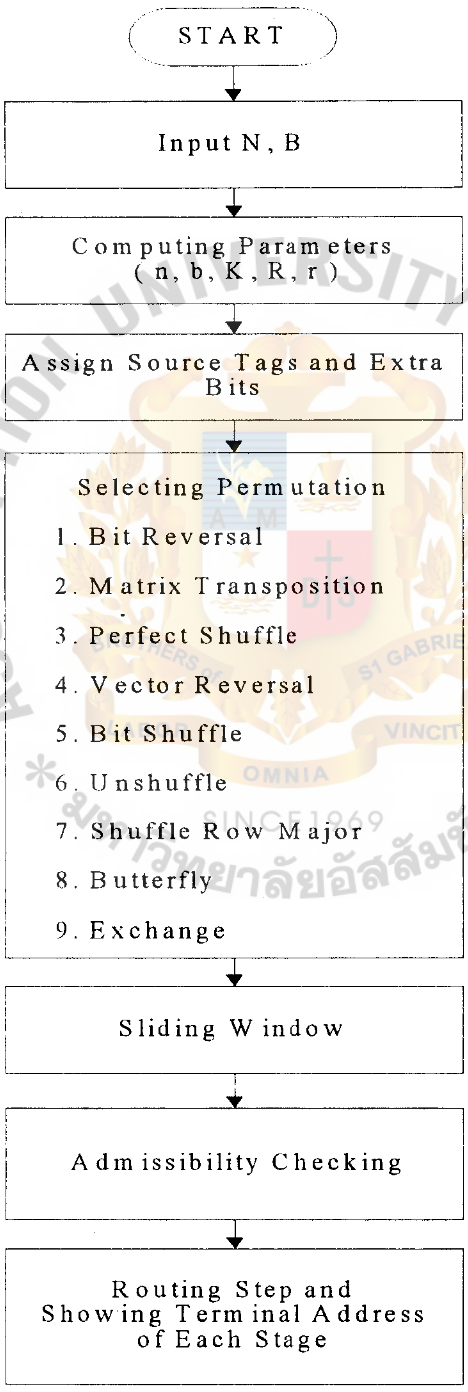
Flowchart: *Admissibility*



4.3 Frame Format of Program Design

The BPC admissibility on R-path Omega network is simulated in C language program with the variable sizes of network and switching elements as mentioned in Table.III of section 3.2.2.

Flowchart *Procedure*



Algorithm *Procedure*

```
1. Compute Parameters ( n, b, K, R, r )
        Input: size of network ( N ), size of switch ( B )
        Output: n, b, K, R, r
        Read ( N ), ( B ) to compute n, b, K, R, r by following equations
                $n = \log_2 N;$
                $b = \log_2 B;$
                $K = \lceil \log_B N \rceil;$
                $R = B^{[n/b]};$
                $r = \log_2 R;$
2. Assign Source Tags $(S = s_0 s_1 \cdots s_{n-2} s_{n-1})$ and Extra Bits $(\otimes_1 \otimes_2 \cdots \otimes_r)$
    /* This step is to assign the arrays of source tags and extra bits */
        Input: a number of bit ( n ), extra bit number ( r )
        Output: array S[0...n-1], array X[1...r]
        For i=0 to n-1;
        Do array of source tag S[i] = $s_0, s_1, \ldots, s_{n-2}, s_{n-1}$;
        For j=1 to r;
        Do array of extra bit X[j] = $\otimes_1 \otimes_2 \cdots \otimes_r$;
3. Select Permutation and Show the Entire Path
    /* This step performs the permutation selection and then the entire path of
the selected permutation is shown $s_0 \ldots s_{n-1} \otimes_1 \ldots \otimes_r d_0 \ldots d_{n-1}$ */
        Input: CaseNumber (x)  /* To select the case of permutation*/
        Output: EntirePath ( $s_0 \ldots s_{n-1} \otimes_1 \ldots \otimes_r d_0 \ldots d_{n-1}$ )
        For x=1 to 9; switch to case (x)
```

Case x=1; Bit_Reversal_Permutation

$$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow s_{n-1} s_{n-2} \cdots s_1 s_0$$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_{n-1} s_{n-2} \cdots s_1 s_0$ */

Case x=2; Matrix_Transposition_Permutation

$$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow \begin{cases} s_l s_{l+1} \cdots s_{2l-1} s_0 s_1 \cdots s_{l-1} & \text{if } n = 2l \\ s_l s_{l+1} \cdots s_{2l} s_0 s_1 \cdots s_{l-1} & \text{if } n = 2l+1 \end{cases}$$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_l s_{l+1} \cdots s_{2l-1} s_0 s_1 \cdots s_{l-1}$ when *n* bit number equals to *2l* and to destination $s_l s_{l+1} \cdots s_{2l} s_0 s_1 \cdots s_{l-1}$ when *n* bit number equals to *2l* + 1.*/

Case x=3; Shuffle_Permutation

$$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow s_1 s_2 \cdots s_{n-1} s_0$$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_1 s_2 \cdots s_{n-1} s_0$ */

Case x=4; Vector_Reversal_Permutation

$$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow \overline{s}_0 \overline{s}_1 \cdots \overline{s}_{n-2} \overline{s}_{n-1}$$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $\overline{s}_0 \overline{s}_1 \cdots \overline{s}_{n-2} \overline{s}_{n-1}$ */

Case x=5; Bit_Shuffle_Permutation

$$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow \begin{cases} s_0 s_2 \cdots s_{n-2} s_1 s_3 \cdots s_{n-1} & \text{if } n = 2l \\ s_0 s_2 \cdots s_{n-1} s_1 s_3 \cdots s_{n-2} & \text{if } n = 2l+1 \end{cases}$$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_0 s_2 \cdots s_{n-2} s_1 s_3 \cdots s_{n-1}$ when *n* bit number equals to *2l* and to destination $s_0 s_2 \cdots s_{n-1} s_1 s_3 \cdots s_{n-2}$ when *n* bit number equals to *2l* + 1.*/

Case x=6; Unshuffle_Permutation

$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow s_{n-1} s_0 \cdots s_{n-3} s_{n-2}$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_{n-1} s_0 \cdots s_{n-3} s_{n-2}$ */

Case x=7; Shuffle_Row_Major_Permutation

$$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow \begin{cases} s_0 s_l s_1 s_{l+1} \cdots s_{l-1} s_{2l-1} & \text{if } n = 2l \\ s_0 s_{l+1} s_1 s_{l+2} \cdots s_{l-1} s_{2l} s_l & \text{if } n = 2l+1 \end{cases}$$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_0 s_l s_1 s_{l+1} \cdots s_{l-1} s_{2l-1}$ when *n* bit number equals to *2l* and to destination $s_0 s_{l+1} s_1 s_{l+2} \cdots s_{l-1} s_{2l} s_l$ when *n* bit number equals to *2l* + 1.*/

Case x=8; Butterfly_Permutation

$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow s_{n-1} s_1 \cdots s_{n-2} s_0$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_{n-1} s_1 \cdots s_{n-2} s_0$ */

Case x=9; Exchange_Permutation

$s_0 s_1 \cdots s_{n-2} s_{n-1} \Rightarrow s_0 s_1 \cdots s_{i-1} \overline{s}_i s_{i+1} \cdots s_{n-2} s_{n-1}$

/* Source $s_0 s_1 \cdots s_{n-2} s_{n-1}$ goes to destination $s_0 s_1 \cdots s_{i-1} \overline{s}_i s_{i+1} \cdots s_{n-2} s_{n-1}$ */

Print("EntirePath") according to the selected permutation

4. Sliding Window

```
Input:    K, b, n
Output:   Window Address
Read( K, b, n );
    For i = 0 to K
    WindowAddress = data from bit ib to ( ib + n );
```

```
        Print( "Window Address stage  = ", i, WindowAddress);
```

5. Admissibility Checking

```
    Input:     array P[$s_0 s_1 \ldots s_{bi} s_{bi+1} \ldots s_{n-1} \otimes_1 \otimes_2 \ldots \otimes_r \ldots d_{bi-r-1} \ldots d_{n-1}$]
    Output:    "admissible", "non-admissible"
    Check:
        For $i$ = 1 to $K$
        For x = $ib$ to $n$-1
        ReadList( S[x] );
        If $i \neq K$
        For y = 0 to $ib$-$r$-1
        ReadList( D[y] );
        If( any of list S[x] =  any of list D[y] )
            Print("non-admissible");
        Else
            Print("admissible");
```

6. Routing Step and Showimg Terminal Address of Each Stage

```
    Input: bit number (n), bit number (b), stage number (K)
    Output: Terminal Address
      For i=0 to $K$ Do
        Read data from $ib$ position to ($ib$+$n$) position.
        /* Therefore, we obtain the general address of each stage,
        $s_{bi} s_{bi+1} \ldots s_{n-1} \otimes_1 \ldots \otimes_r \ldots d_{xb-r-1}$ */
      For source_address = 0 to N-1
        Do the following method to arrange extra bit
```

1. Sort order of source address from 0 to N-1 in the binary representation.
2. Consider the bits in that stage excluding the extra bit that are $s_{bi}s_{bi+1}\ldots s_{n-1}\ldots d_{xb-r-1}$.
3. Set extra bit of the first address equal to 0 (in binary representation).
4. Then we consider the next address, if it is distinct from the previous one, the extra bit will be set to 0.
5. If not, the extra bit will be increased by one in binary representation number.
6. The process will continue until source address = N-1.

4.4 Summary of Permutation Capability Results

TABLE VI

SUMMARY OF PERMUTATION CAPABILITY RESULTS

Note: "✗" means "non-admissible", "✓" means "admissible"

Networks	Bit Reversal	Matrix Transposition	Perfect Shuffle	Vector Reversal	Bit Shuffle	Unshuffle	Shuffle Row Major	Butterfly	Exchange
1-path Omega with N = 8	✗	✗	✗	✓	✗	✗	✗	✗	✓
2-Path Omega with N = 8	✗	✓	✓	✓	✓	✗	✓	✗	✓
1-path Omega with N = 16	✗	✗	✗	✓	✗	✗	✗	✗	✓
4-Path Omega with N = 16	✗	✓	✓	✓	✓	✗	✓	✗	✓
1-path Omega with N = 32	✗	✗	✗	✓	✗	✗	✗	✗	✓
2-Path Omega with N = 32	✗	✗	✓	✓	✗	✗	✓	✗	✓
8-path Omega with N = 32	✗	✓	✓	✓	✓	✗	✓	✗	✓
1-path Omega with N = 64	✗	✗	✗	✓	✗	✗	✗	✗	✓
4-Path Omega with N = 64	✗	✗	✓	✓	✓	✗	✓	✗	✓
16-path Omega with N = 64	✗	✓	✓	✓	✓	✗	✓	✗	✓
1-path Omega with N = 128	✗	✗	✗	✓	✗	✗	✗	✗	✓
2-Path Omega with N = 128	✗	✗	✓	✓	✗	✗	✗	✗	✓
4-Path Omega with N = 128	✗	✗	✓	✓	✓	✗	✓	✗	✓
8-path Omega with N = 128	✗	✓	✓	✓	✓	✗	✓	✗	✓
32-path Omega with N = 128	✗	✓	✓	✓	✓	✗	✓	✗	✓

TABLE VI

SUMMARY OF PERMUTATION CAPABILITY RESULTS

Note: "✗" means "non-admissible", "✓" means "admissible"

Networks	Bit Reversal	Matrix Transposition	Perfect Shuffle	Vector Reversal	Bit Shuffle	Unshuffle	Shuffle Row Major	Butterfly	Exchange
1-path Omega with N = 256	✗	✗	✗	✓	✗	✗	✗	✗	✓
2-Path Omega with N = 256	✗	✗	✓	✓	✗	✗	✗	✗	✓
4-Path Omega with N = 256	✗	✗	✓	✓	✓	✗	✓	✗	✓
16-path Omega with N = 256	✗	✓	✓	✓	✓	✗	✓	✗	✓
64-path Omega with N = 256	✗	✓	✓	✓	✓	✗	✓	✗	✓
1-path Omega with N = 512	✗	✗	✗	✓	✗	✗	✗	✗	✓
2-Path Omega with N = 512	✗	✗	✓	✓	✗	✗	✗	✗	✓
8-Path Omega with N = 512	✗	✗	✓	✓	✗	✗	✓	✗	✓
32-path Omega with N = 512	✗	✓	✓	✓	✓	✗	✓	✗	✓
128-path Omega with N = 512	✗	✓	✓	✓	✓	✗	✓	✗	✓

Chapter 5

Conclusion

5.1 Thesis Summary

Multistage interconnection networks are of interest for use in large-scale parallel computer and telecommunication systems. The class of multistage, self-routing networks which, in its basis form requires $\log_2 B$ stages of 2x2 switches to connect N inputs to N outputs with $N/2$ 2x2 switches, are needed in each stage is known as the Banyan or cube-type class of networks. This class includes Omega network [1]. The problem with this topology is that there is only one path from a given network input to a given output. Therefore, it is vulnerable to component faults. To achieve the problem, R-path Omega network with multiple disjoint paths for each input-output pair was presented [2].

The objective of the thesis is to explore how the presence of redundant paths of R-path Omega network improves the permutation capability in comparison with a 1-path Omega network. We introduce a simple algorithm that determines the admissibility of various kinds of regular BPC (bit-permute-complement) permutations to R-path Omega networks. The investigations were based on computational experiment with C language program and were done for the variable sizes of network and switching elements.

At the end of exploration, it was proved that permutation capability of R-path Omega networks is much better than of 1-path Omega networks. Of nine most frequently used in parallel programming permutations listed in section 3.3, 1-path Omega can realized for one pass only two permutations, whereas R-path Omega realizes up to six permutations (see section 4.4). This fact is of importance for parallel computer hardware and software designers. The findings of the done research are also of interest for designers of switching systems in telecommunication. E.g. R-path Omega network may be considered as an alternative to the Banyan router in Batcher-Banyan architecture of ATM switches.

5.2 Recommendation for Future Work

It can be recommended to expand the developed algorithm in this work approach to *k*-Extra-Stage multistage cube-type networks (*k*-EMCTN), where redundant disjoint paths for any source-destination pair are provided by adding k more stages in front of an Omega network implemented of switches with size 2×2.

Appendix A
Source Code

```
#include "c:\project\project.cpp"
void main(void)
{
        char ch;
        int i;
        do
        {
                printf("\n1.Input Parameter.");
                printf("\n2.Select Permutation.");
                printf("\n3.Check Admissibility.");
                printf("\n4.Routing Step.");
                printf("\n5.Show All Addresses.");
                printf("\n6.Show All Parameters And Write Data.");
                printf("\n7.Exit.");
                printf("\nSelect => ");
                ch = getche();
                switch(ch)
                {
                        case '1':
                                Parameter();   getch();              break;
                        case '2':
                                Input_Per();  getch();         break;
                        case '3':
                                char* added = (char*)malloc(sizeof(char)*r);
                                for(i=0;i<r;i++)
                                added[i] = 'X';
                                added[r] = '\0';
                                BlockingCheck(ch_Per,added);         break;
                        case '4':
                                Sort_First();
                                GenAdded();
```

```
                    showall();
                    getch();              break;
               case '5':
                    SearchSort();
                    showall();
                    getch();              break;
               case '6':
                    ShowParameter();
                    WriteOutput();                  break;
               case '7':
                    clearnode();                    break;
          }
     }
     while(ch!='7');
}

#include "c:\project\projhead.h"
int N,n,B,b,R,r,K,Total_length;
//int check=0;
char lp_Per[256],ch_Per='\0';
node *hnode,*wnode,*nnode;
node *newnode()
{
     return (node*)malloc(sizeof(node));
}

void delnode(node *p)
{
     if(p->lp==NULL && p->rp!=NULL)
     {
          hnode = p->rp;          wnode = hnode;
     }
     else
          if(p->lp!=NULL && p->rp!=NULL)
```

```
                wnode = p->rp;
        else
                if(p->lp!=NULL && p->rp==NULL)
                wnode = p->lp;
        else
        {
                hnode = NULL;           wnode = hnode;
        }
        free((void*)p);
}

node *firstnode(node *head,int x)
{
        head = newnode();
        head->index = x;
        head->lp = NULL;
        head->rp = NULL;
        return head;
}

void insertnode(node *start,int x)
{
        if(hnode==NULL)
        {
                hnode = firstnode(hnode,x);
                start = hnode;
        }
        else
        {
                while(start->rp!=NULL)
                start = start->rp;
                node *newn;
                newn = newnode();
                newn->index = x;
                if(start->rp==NULL)
                        newn->rp = NULL;
                else
                {
                        newn->rp = start->rp;
                        newn->rp->lp=newn;
                }
                newn->lp = start;
                start->rp = newn;
        }
        wnode = start;
}

void deletenode(node *current)
{
        node *after,*before;
        if(current != NULL)
        {
                after = current->rp;
                before = current->lp;
                delnode(current);
                before->rp = after;
                after->lp = before;
        }
}

void clearnode()
{
        wnode = hnode;
        nnode = wnode->rp;
        while(wnode!=NULL)
        {
                deletenode(wnode);
                wnode = nnode;
                nnode = nnode->rp;
        }
}

void shownow()
{
        if(wnode != NULL)
                printf("\n %3d = %s    %s\n",wnode->index,wnode->source,wnode-
>dest);
         else
                printf("\nNo Data in List\n");
                printf("----------------------------------------------------\n");
}

void showall()
{
        node *x,*y;
        x = hnode;
        y = hnode;
        for(int i=0;i<(N/2);i++)
                y = y->rp;
                printf("\n");
        while(y!=NULL)
        {
                if(x==wnode)            printf("->");
                else                    printf("  ");
                printf("%4d = %s %s %s",x->index,x->source,x->added,x->dest);
                x=x->rp;
                if(y==wnode)
                        printf("\t->%4d = %s %s %s\n",y->index,y->source,y-
                        >added,y->dest);
                else
                         printf("\t  %4d = %s %s %s\n",y->index,y->source,y-
                        >added,y->dest);
                y=y->rp;
        }
}

void swapnode(node *n1,node *n2)
{
        node temp;
        temp.index = n1->index;
        strcpy(temp.source,n1->source);
        strcpy(temp.added,n1->added);
        strcpy(temp.dest,n1->dest);
        strcpy(temp.databit,n1->databit);
        n1->index = n2->index;
        strcpy(n1->source,n2->source);
        strcpy(n1->added,n2->added);
        strcpy(n1->dest,n2->dest);
        strcpy(n1->databit,n2->databit);
        n2->index = temp.index;
        strcpy(n2->source,temp.source);
        strcpy(n2->added,temp.added);
        strcpy(n2->dest,temp.dest);
        strcpy(n2->databit,temp.databit);
}

long C1(char *Bin)
{
        long sum=0;
        int i;
        for (i = 0; i < strlen(Bin); i++)
        {
                if(Bin[i] >= '0' && Bin[i] < '2')
                        sum += ((Bin[i]-'0') * pow(2,strlen(Bin)-i-1));
                else
                {
                    printf("ERROR\n");   getch(); sum=0;
               }
          }
          return(sum);
}

char *ItoChar(int Dec,int number)
{
          char *Bin,str[11];
          int i;
          Bin = (char*)malloc(number+1);
          Bin[n] = '\0';
          for(i=0;i<number;i++)
               Bin[i]=' ';
               Bin[i]='\0';
               i=0;
               do
               {
                    Bin[i]=(Dec%10)+'0';
                    Dec=Dec/10;
                    i++;
               }
          while(Dec>0);
               strcpy(str,strrev(Bin));
               free((char*)Bin);
               return str;
}

char *ItoChar(int Dec,int x,int number)
{
          char *Bin;
          int i;
          Bin = (char*)malloc(number+1);
          Bin[r] = '\0';
```

```
        for(i=0;i<number;i++)
                Bin[i]='0';
                Bin[i]='\0';
                i=0;
                do
                {
                        Bin[i]=(Dec%x)+'0';
                        Dec=Dec/x;
                        i++;
                }
                while(Dec>0);
                return strrev(Bin);
}

void Sort(int type)
{
        long data1,data2;
        int ch,start=b;
        node* a = hnode;
        node* b = a->rp;
         if(type == '4')
        {
                do
                {
                        clrscr();
                        printf("There are %d stages.",K);
                        printf("\nWhich stage do you want to sort?\n=> ");
                        scanf("%d",&ch);
                }
                while(!(ch>=1 & ch<=K));
                        start = ch*start;
                while(a!=NULL)
                {
                        while(b!=NULL)
                {
                        char* strTmp1 = (char*)malloc(sizeof(char)*n);
                        char* strTmp2 = (char*)malloc(sizeof(char)*n);
                        for(int i=0;i<n;i++)
                        {
                                strTmp1[i] = a->databit[i+start];
                                strTmp2[i] = b->databit[i+start];
                        }
                        strTmp1[n]='\0';        strTmp2[n]='\0';
                        data1 = C1(strTmp1); data2 = C1(strTmp2);
                        free(strTmp1);
                        free(strTmp2);
                        if(data1 > data2)
                                swapnode(a,b);
                        else
                        if(data2 == data1 & a->index > b->index)
                                swapnode(a,b);
                                b = b->rp;
                }
                a = a->rp;
                b = a->rp;
        }
        ShowStage(ch);
         getch();
        return;
}
else
while(a!=NULL)
{
        while(b!=NULL)
        {
                switch(type)
                {
                case '1':
                    data1 = a->index;       //data1 = C1(a->source);
                    data2 = b->index;       //data2 = C1(b->source);
                    break;
                case '2':
                    data1 = C1(a->added);           data2 = C1(b->added);
                    break;
                case '3':
                    data1 = C1(a->dest);    data2 = C1(b->dest);
                    break;
                }
                if(data2 < data1)
                swapnode(a,b);
                b = b->rp;
            }
            a = a->rp;
            b = a->rp;
        }
}

void SearchSort()
{
        int ch;
        do
        {
            clrscr();
            printf("1.Sort by Index or Source.\n");
            printf("2.Sort by Added Bit.\n");
            printf("3.Sort by Destination.\n");
            printf("4.Sort by Stage.\n");
            printf("You select :> ");
            ch = getche();
        }
        while(!(ch>='1' & ch<='4'));
        Sort(ch);
        clrscr();
}

int Num_Route(int B,int n,int b)
{
        float a,x;
        int number;
        if((n%b))
                a = (n/b+1)-(1.0*n/b);
        else
                a = 0.0;
                if(B==2 || N==B)
                        return 1;
                if(N==8 && B==4)
                        return 2;
                        number = pow(B,a);
                if(number%2)
                        return number+1;
                else
                        return number;
}

void Parameter(void)
{
        clrscr();
        printf("Enter Node of Network: ");   scanf("%d",&N);
        printf("Enter Size of switch: ");   scanf("%d",&B);
        //   N = 2^n                        //   B = 2^b
        n = log10(N)/log10(2);              b = log10(B)/log10(2);
        int Buf = log10(N)/log10(B)*100;
        if(Buf%100)
                K = log10(N)/log10(B)+1;
        else
                K = log10(N)/log10(B);
```

```
            R = Num_Route(B,n,b);
            r = log10(R)/log10(2);
            Total_length = 2*n+r;
            printf("\nn=%d",n);  printf("\nb=%d",b);
            printf("\nR=%d",R);   printf("\nr=%d",r);
            printf("\nK=%d",K);
}

void ShowStage(int x)
{
        x--;
        int i;
        char s[11],d[11],str1[11],str2[11],str[11];
        node *a,*y;
        a = hnode;
        y = hnode;
        for(i=0;i<(N/2);i++)
            y=y->rp;
            printf("\n");
            while(a!=NULL)
            {
                for(i=b+(x*b);i<n;i++)
                    str1[i-(b+(x*b))] = a->source[i];
                    str1[i-(b+(x*b))] = '\0';
                if( K != x+1)
                {
                    for(i=0;i<b+(x*b)-r;i++)
                        str2[i] = a->dest[i];
                        str2[i] = '\0';
                }
                else
                    strcpy(str2,a->dest);
                    strcpy(str,"");
                    strcat(str,str1);
```

```
                        if(K != x+1)
                                strcat(str,a->added);
                                strcat(str,str2);
                                printf("\n  Stage%d:   node %2d:   %s %s %s",x+1,a-
                                >index,str1,a->added,str2);
                                a=a->rp;
                }
}

void GenAdded()
{
        int i=0,x=pow(2,r);
        node* a=hnode;
        clrscr();
        while(a!=NULL)
        {
                for(i=x-1;i>=0;i--)
                {
                        strcpy(a->added,ItoChar(i,2,r));
                        strcpy(a->databit,a->source);
                        strcat(a->databit,a->added);
                        strcat(a->databit,a->dest);
                        a=a->rp;
                }
        }
        //  printf("\n\n .....Complete.....\n\n");
        //  getch();
        clrscr();
}

void Bit_reversal(char *strSource,char *strDestination)
{
        strcpy(strDestination,strSource);
        strrev(strDestination);
        strDestination[strlen(strSource)] = '\0';
        strcpy(lp_Per,"Permutation : Bit reversal");
         return;
}

void Matrix_transposition(char *strSource,char *strDestination)
{
        char *s1,*s2;
        int l,x=0;
        if(strlen(strSource)%2)
        {
                l = (strlen(strSource)-1)/2;
                s1 = (char*)calloc(l+1,sizeof(char));
         }
        else
        {
                l = strlen(strSource)/2;
                s1 = (char*)calloc(l,sizeof(char));
         }
        s2 = (char*)calloc(l,sizeof(char));
        for(int i=l;i<strlen(strSource);i++)
        {
                s1[x] = strSource[i];
                x++;
        }
        s1[x] = '\0';
        for(i=0;i<l;i++)
        s2[i] = strSource[i];
        s2[i] = '\0';
        strcpy(strDestination,s1);
        strcat(strDestination,s2);
        free(s1);
        free(s2);
        strcpy(lp_Per,"Permutation :  Matrix transposition");
```

```
}

void Perfect_shuffle(char *strSource,char *strDestination)
{
        char ch = strSource[0];
        for(int i=0;i<strlen(strSource)-1;i++)
        strDestination[i]=strSource[i+1];
        strDestination[i] = ch;
        strDestination[i+1] = '\0';
        strcpy(lp_Per,"Permutation :  Perfect shuffle");
}

void Vector_reversal(char *strSource,char *strDestination)
{
        for(int i=0;i<strlen(strSource);i++)
        {
                if(strSource[i]=='0')
                        strDestination[i] = '1';
                else
                        strDestination[i] = '0';
        }
        strDestination[i] = '\0';
        strcpy(lp_Per,"Permutation :  Vector reversal");
}

void Bit_shuffle(char *strSource,char *strDestination)
{
        char *s1,*s2;
        int x=0,y=0,i=0;
        s1 = (char*)calloc(strlen(strSource),sizeof(char));
        s2 = (char*)calloc(strlen(strSource),sizeof(char));
        for(i=0;i<strlen(strSource);i++)
        {
                if(!(i%2)){ s2[x] = strSource[i];  x++; }
                else   { s1[y] = strSource[i];  y++; }
        }
        strcpy(strDestination,strcat(s2,s1));
        free(s1);
        free(s2);
        strcpy(lp_Per,"Permutation :  Bit shuffle");
}

void Unshuffle(char *strSource,char *strDestination)
{
        strDestination[0] = strSource[strlen(strSource)-1];
        for(int i=0;i<strlen(strSource)-1;i++)
        strDestination[i+1] = strSource[i];
        strDestination[i+1] = '\0';
        strcpy(lp_Per,"Permutation :  Unshuffle");
}

void Shuffle_row_major(char *strSource,char *strDestination)
{
        int l,len=strlen(strSource);
        strcpy(strDestination,"");
        l = (strlen(strSource))/2;
        for(int i=0;i<l;i++)
        {
                if(strlen(strSource)%2)
                {
                        strDestination[i*2]=strSource[i];
                        strDestination[i*2+1]=strSource[l+i+1];
                        strDestination[len-1] = strSource[l];
                }
                else
                {
                        strDestination[i*2]=strSource[i];
                        strDestination[i*2+1]=strSource[l+i];
                }
        }
        strDestination[len] = '\0';
        strcpy(lp_Per,"Permutation :  Shuffle_row_major");
}

void Butterfly(char *strSource,char *strDestination)
{
        strcpy(strDestination,strSource);
        strDestination[0] = strSource[strlen(strSource)-1];
        strDestination[strlen(strSource)-1] = strSource[0];
        strDestination[strlen(strSource)] = '\0';
        strcpy(lp_Per,"Permutation :  Butterfly");
}

void Exchange(char *strSource,char *strDestination)
{
        int l = strlen(strSource)/2-1;
        if(strlen(strSource)%2)
                l=strlen(strSource)/2;
                strcpy(strDestination,strSource);
        if(strSource[l] == '0')
                strDestination[l] = '1';
        else
                strDestination[l] = '0';
                strcpy(lp_Per,"Permutation :  Exchange");
}

char showmenu()
{
        char ch;
        do
        {
                clrscr();
                printf("1. Bit reversal.\n");
                printf("2. Matrix transition.\n");
                printf("3. Perfect Shuffle.\n");
                printf("4. Vector Reversal.\n");
                printf("5. Bit Shuffle.\n");
                printf("6. Unshuffle.\n");
                printf("7. Shuffle Row Major.\n");
                printf("8. Butterfly.\n");
                printf("9. Exchange.\n");
                printf("You select :> ");
                ch = getche();
        }
        while(!(ch>='1' & ch<='9'));
        return ch;
}

void Input_Per()
{
        ch_Per = showmenu();
        char* added = (char*)malloc(sizeof(char)*r);
        for(int i=0;i<r;i++)
        added[i] = '0';
        added[r] = '\0';
        //  BlockingCheck(ch_Per,added);
        if(hnode != NULL)
        {
                clearnode();
                hnode = NULL;
        }
        node *a;
        //clrscr();
        for(i=0;i<N;i++)
        {
                if(hnode==NULL)
```

```
          {
                    hnode = firstnode(hnode,i);
                    a = hnode;
          }
          else
          insertnode(a,i);
}
while(a!=NULL)
{
          // if(a==wnode) printf("->");
          // else   printf("\n  ");
          strcpy(a->source,ItoChar(a->index,2,n));
          switch(ch_Per)
          {
                    case '1':  Bit_reversal(a->source,a->dest);
                              break;
                    case '2':  Matrix_transposition(a->source,a->dest);
                              break;
                    case '3':  Perfect_shuffle(a->source,a->dest);
                              break;
                    case '4':  Vector_reversal(a->source,a->dest);
                              break;
                    case '5':  Bit_shuffle(a->source,a->dest);
                              break;
                    case '6':  Unshuffle(a->source,a->dest);
                              break;
                    case '7':  Shuffle_row_major(a->source,a->dest);
                              break;
                    case '8':  Butterfly(a->source,a->dest);
                              break;
                    case '9':  Exchange(a->source,a->dest);
                              break;
          }
          strcpy(a->added,added);
```

```
            strcpy(a->databit,"");
            strcat(a->databit,a->source);
            strcat(a->databit,a->added);
            strcat(a->databit,a->dest);
            //          printf("%4d = %s %s %s",a->index,a->source,added,a->dest);
            a=a->rp;
        }
        free(added);
        clrscr();
        printf("\n .....Complete.....\n");
        showall();
}

void BlockingCheck(char ch,char* added)
{
        char s[11],d[11],str1[11],str2[11];
        int i,j,check;
        for(i=0;i<n;i++)
        {
            if(i<9)
                    s[i] = i+49;
            else
                    s[i] = i+65-10;
        }
        s[n] = '\0';
        switch(ch)
        {
            case '1':  Bit_reversal(s,d);      break;
            case '2':  Matrix_transposition(s,d);    break;
            case '3':  Perfect_shuffle(s,d);     break;
            case '4':  Vector_reversal(s,d);     break;
            case '5':  Bit_shuffle(s,d);        break;
            case '6':  Unshuffle(s,d);          break;
            case '7':  Shuffle_row_major(s,d);       break;
        case '8':  Butterfly(s,d);          break;
        case '9':  Exchange(s,d);           break;
    }
    printf("\n\nPermutation = %s %s %s",s,added,d);
    for(int x=1;x<=K;x++)
    {
        for(i=(x*b);i<=n-1;i++)
            str1[i-(x*b)] = s[i];
        str1[i-(x*b)] = '\0';
        if( K!= x)
        {
            for(i=0;i<(x*b)-r;i++)
                str2[i] = d[i];
            str2[i] = '\0';
        }
        else
            strcpy(str2,d);
        if(K!=x)
            printf("\n\nStage %d = %s%s%s",x,str1,added,str2);
        else
            printf("\n\nStage %d = %s",x,str2);
        check=0;
        for(i=0;i<strlen(str1);i++)
            for(j=0;j<strlen(str2);j++)
                if(str1[i] == str2[j])
                    check++;
        if(check>=1)
        // printf("\nDuplicate %d Position.BLOCKING!!!\n",check);
        printf("\nDuplicate %d Position.NON-ADMISSIBLE!!!\n\n",check);
        else
            printf("\nNo Duplicate .ADMISSIBLE!!!\n\n");
    }
}
```

```
void ShowParameter()
{
        clrscr();
        printf("\nN=%d",N);  printf("\nB=%d",B);
        printf("\nn=%d",n);  printf("\nb=%d",b);
        printf("\nR=%d",R);   printf("\nr=%d",r);
        printf("\nK=%d\n",K);
        getch();
}

void WriteOutput()
{
        node *x;
        char str[256]="",str1[11],str2[11],ch='\0',buffer[256];
        char DirName[100] = "C:\\Project\\File\\";
        if(hnode != NULL)
        {
                do
                {
                        clrscr();
                        printf("Do you want write DATA ???\n");
                        printf("Select y/n ==> ");
                        ch = getche();
                }
                while(!(ch=='y' | ch=='n'));
        }
        if(ch == 'n')
        {
                return;
        }
        flushall();
        char* fileName = (char*)calloc(9,sizeof(char));
                do
                {
                    printf("\nEnter file name for write ==> ");
                    gets(fileName);
          }
          while(!strlen(fileName));

     strncat(DirName,fileName,8);
     strcat(DirName,".txt");
     free(fileName);
     FILE *fp = fopen(DirName,"w");
/////////////////////////////////
     strcpy(str,"Size of Network(N)=");    strcat(str,ItoChar(N,2));
     strcat(str,"\nSize of Switch(B)=");    strcat(str,ItoChar(B,2));
     strcat(str,"\nNumber of Stage(K)="); strcat(str,ItoChar(K,2));
     strcat(str,"\nNumber Bit of Source(n)=");     strcat(str,ItoChar(n,2));
     strcat(str,"\nOfset of Shift Stage(b)=");        strcat(str,ItoChar(b,2));
     strcat(str,"\nNumber of Redundant Path(R)=");        strcat(str,ItoChar(R,2));
     strcat(str,"\nNumber Bit of Added Bit(r)="); strcat(str,ItoChar(r,2));
     strcat(str,"\n\n");
     fwrite(&str,strlen(str),1,fp);
/////////////////////////////////
     char *lp_source = (char*)calloc(n,sizeof(char));
     char *lp_added  = (char*)calloc(r,sizeof(char));
     char *lp_dest   = (char*)calloc(n,sizeof(char));
     for(int i=0;i<r;i++)
          lp_added[i] = 'X';
          lp_added[r] = '\0';
     for(i=0;i<n;i++)
     {
          if(i<9)
          lp_source[i] = i+49;
          else lp_source[i] = i+65-10;
     }
     lp_source[n] = '\0';
     switch(ch_Per)
```

```
    {
            case '1':  Bit_reversal(lp_source,lp_dest);       break;
            case '2':  Matrix_transposition(lp_source,lp_dest);    break;
            case '3':  Perfect_shuffle(lp_source,lp_dest);     break;
            case '4':  Vector_reversal(lp_source,lp_dest);     break;
            case '5':  Bit_shuffle(lp_source,lp_dest);         break;
            case '6':  Unshuffle(lp_source,lp_dest);           break;
            case '7':  Shuffle_row_major(lp_source,lp_dest);        break;
            case '8':  Butterfly(lp_source,lp_dest);           break;
            case '9':  Exchange(lp_source,lp_dest);            break;
    }
    strcpy(str,lp_Per);       strcat(str," ==> ");
    strcat(str,lp_source);   strcat(str," ");
    strcat(str,lp_added);    strcat(str," ");
    strcat(str,lp_dest);     strcat(str,"\n\n");
    fwrite(&str,strlen(str),1,fp);
////////////////////////////////////
    int check;
    for(int q=0;q<K;q++)
    {
            check=0;

            for(i=b+(q*b);i<n;i++)
                    str1[i-(b+(q*b))] = lp_source[i];
                    str1[i-(b+(q*b))] = '\0';
            if( K != q+1)
            {
                    for(i=0;i<b+(q*b)-r;i++)
                            str2[i] = lp_dest[i];
                            str2[i] = '\0';
            }
            else
                    strcpy(str2,lp_dest);
                    strcpy(str,"Stage ");
```

```
                strcat(str,ItoChar(q+1,2));     strcat(str," ");
                strcpy(buffer,str1);
            if(K != q+1)
                strcat(buffer,lp_added);
                strcat(buffer,str2);
//          printf("\n\nStage %d = %s",q+1,str);
            for(i=0;i<n;i++)
                for(int j=i+1;j<n;j++)
                    if(buffer[i] != 'X')
                        if(buffer[i] == buffer[j])
                            check++;
            strcat(str,buffer);
            if(check>=1)
            {
            //printf("\nDuplicate %d Position.NONADMISSIBLE!!!",check);
                strcat(str," >> Duplicate ");
                strcat(str,ItoChar(check,2));
                strcat(str," Position.NON-ADMISSIBLE!!!\n");
            }
            else
            //printf("\nNo Duplicate .ADMISSIBLE!!!");
                strcat(str," >> No Duplicate .ADMISSIBLE!!!\n");
                fwrite(&str,strlen(str),1,fp);
        }
        fwrite("\n",1,1,fp);
        free(lp_source);
        free(lp_added);
        free(lp_dest);
////////////////////////////////
        x = hnode;
        while(x!=NULL)
        {
            strcpy(str,"");
            strcat(str,ItoChar(x->index,4));
```

```
                strcat(str," ");
                strcat(str,x->source);
                strcat(str," ");
                strcat(str,x->added);
                strcat(str," ");
                strcat(str,x->dest);
                strcat(str,"\t");
///////////////////////////////////////
                for(int j=0;j<K;j++)
                {
                        for(int i=b+(j*b);i<n;i++)
                        str1[i-(b+(j*b))] = x->source[i];
                        str1[i-(b+(j*b))] = '\0';
                        if( K != j+1)
                        {
                                for(i=0;i<b+(j*b)-r;i++)
                                        str2[i] = x->dest[i];
                                        str2[i] = '\0';
                        }
                        else
                                strcpy(str2,x->dest);
                                strcat(str,str1);
                        if(K != j+1)
                                strcat(str,x->added);
                                strcat(str,str2);
                                strcat(str,"\t");
                }
///////////////////////////////////////
                strcat(str,"\n");
                fwrite(&str,strlen(str),1,fp);
                //          printf("%4d = %s",x->index,x->databit);
                x=x->rp;
        }
        fclose(fp);
        clrscr();
        gotoxy(15,11);
        printf("Write DATA in file %s Complete.....",DirName);
        getch();
}

void Sort_First()
{
        long data1,data2;
        int ch,start=b;
        node* a = hnode;
        node* b = a->rp;
        while(a!=NULL)
        {
                while(b!=NULL)
                {

                        char* strTmp1 = (char*)malloc(sizeof(char)*n);
                        char* strTmp2 = (char*)malloc(sizeof(char)*n);

                        for(int i=0;i<n;i++)
                        {
                                strTmp1[i] = a->databit[i+start];
                                strTmp2[i] = b->databit[i+start];
                         }
                        strTmp1[n]='\0';strTmp2[n]='\0';
                        data1 = C1(strTmp1); data2 = C1(strTmp2);
                        free(strTmp1);
                        free(strTmp2);
                        if(data1 > data2)
                                swapnode(a,b);
                        else
                                if(data2 == data1 & a->index > b->index)
                                swapnode(a,b);
```

```
                    b = b->rp;
            }
            a = a->rp;
            b = a->rp;
        }
}
```

Appendix B

Results of Computational Experiments

1-Path Omega Network with Size of Network (N) = 8, and Size of Switch (B) = 2

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages				Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3 (Destination)	
1. Bit Reversal $s_0s_1s_2$ $s_2s_1s_0$	$s_0s_1s_2$	$s_1s_2s_2$	$s_2s_2s_1$	$s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2$ $s_1s_2s_0$	$s_0s_1s_2$	$s_1s_2s_1$	$s_2s_1s_2$	$s_1s_2s_0$	✗
3. Perfect Shuffle $s_0s_1s_2$ $s_1s_2s_0$	$s_0s_1s_2$	$s_1s_2s_1$	$s_2s_1s_2$	$s_1s_2s_0$	✗
4. Vector Reversal $s_0s_1s_2$ $\overline{s}_0\overline{s}_1\overline{s}_2$	$s_0s_1s_2$	$s_1s_2s_0$	$s_2\overline{s}_0\overline{s}_1$	$\overline{s}_0\overline{s}_1\overline{s}_2$	✓
5. Bit Shuffle $s_0s_1s_2$ $s_0s_2s_1$	$s_0s_1s_2$	$s_1s_2s_0$	$s_2s_0s_2$	$s_0s_2s_1$	✗
6. Unshuffle $s_0s_1s_2$ $s_2s_0s_1$	$s_0s_1s_2$	$s_1s_2s_2$	$s_2s_2s_0$	$s_2s_0s_1$	✗
7. Shuffle Row Major $s_0s_1s_2$ $s_0s_2s_1$	$s_0s_1s_2$	$s_1s_2s_0$	$s_2s_0s_2$	$s_0s_2s_1$	✗
8. Butterfly $s_0s_1s_2$ $s_2s_1s_0$	$s_0s_1s_2$	$s_1s_2s_2$	$s_2s_2s_1$	$s_2s_1s_0$	✗
9. Exchange $s_0s_1s_2$ $s_0\overline{s}_1s_2$	$s_0s_1s_2$	$s_1s_2s_0$	$s_2s_0\overline{s}_1$	$s_0\overline{s}_1s_2$	✓

2-Path Omega Network with Size of Network (N) = 8, and Size of Switch (B) = 4

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2 (Destination)	
1.Bit Reversal $s_0s_1s_2 \otimes s_2s_1s_0$	$s_0s_1s_2$	$s_2 \otimes s_2$	$s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2 \otimes s_1s_2s_0$	$s_0s_1s_2$	$s_2 \otimes s_1$	$s_1s_2s_0$	✓
3. Perfect Shuffle $s_0s_1s_2 \otimes s_1s_2s_0$	$s_0s_1s_2$	$s_2 \otimes s_1$	$s_1s_2s_0$	✓
4. Vector Reversal $s_0s_1s_2 \otimes \overline{s}_0\overline{s}_1\overline{s}_2$	$s_0s_1s_2$	$s_2 \otimes \overline{s}_0$	$\overline{s}_0\overline{s}_1\overline{s}_2$	✓
5. Bit Shuffle $s_0s_1s_2 \otimes s_0s_2s_1$	$s_0s_1s_2$	$s_2 \otimes s_0$	$s_0s_2s_1$	✓
6. Unshuffle $s_0s_1s_2 \otimes s_2s_0s_1$	$s_0s_1s_2$	$s_2 \otimes s_2$	$s_2s_0s_1$	✗
7. Shuffle Row Major $s_0s_1s_2 \otimes s_0s_2s_1$	$s_0s_1s_2$	$s_2 \otimes s_0$	$s_0s_2s_1$	✓
8. Butterfly $s_0s_1s_2 \otimes s_2s_1s_0$	$s_0s_1s_2$	$s_2 \otimes s_2$	$s_2s_1s_0$	✗
9. Exchange $s_0s_1s_2 \otimes s_0\overline{s}_1s_2$	$s_0s_1s_2$	$s_2 \otimes s_0$	$s_0\overline{s}_1s_2$	✓

1-Path Omega Network with Size of Network (N) = 16, and Size of Switch (B) = 2

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages					Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4 (Destination)	
1.Bit Reversal $s_0s_1s_2s_3$ $s_3s_2s_1s_0$	$s_0s_1s_2s_3$	$s_1s_2s_3s_3$	$s_2s_3s_3s_2$	$s_3s_3s_2s_1$	$s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3$ $s_2s_3s_0s_1$	$s_0s_1s_2s_3$	$s_1s_2s_3s_2$	$s_2s_3s_2s_3$	$s_3s_2s_3s_0$	$s_2s_3s_0s_1$	✗
3. Perfect Shuffle $s_0s_1s_2s_3$ $s_1s_2s_3s_0$	$s_0s_1s_2s_3$	$s_1s_2s_3s_1$	$s_2s_3s_1s_2$	$s_3s_1s_2s_3$	$s_1s_2s_3s_0$	✗
4. Vector Reversal $s_0s_1s_2s_3$ $\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}$	$s_0s_1s_2s_3$	$s_1s_2s_3\overline{s_0}$	$s_2s_3\overline{s_0}\overline{s_1}$	$s_3\overline{s_0}\overline{s_1}\overline{s_2}$	$\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}$	✓
5. Bit Shuffle $s_0s_1s_2s_3$ $s_0s_2s_1s_3$	$s_0s_1s_2s_3$	$s_1s_2s_3s_0$	$s_2s_3s_0s_2$	$s_3s_0s_2s_1$	$s_0s_2s_1s_3$	✗
6. Unshuffle $s_0s_1s_2s_3$ $s_3s_0s_1s_2$	$s_0s_1s_2s_3$	$s_1s_2s_3s_3$	$s_2s_3s_3s_0$	$s_3s_3s_0s_1$	$s_3s_0s_1s_2$	✗
7. Shuffle Row Major $s_0s_1s_2s_3$ $s_0s_2s_1s_3$	$s_0s_1s_2s_3$	$s_1s_2s_3s_0$	$s_2s_3s_0s_2$	$s_3s_3s_0s_1$	$s_0s_2s_1s_3$	✗
8. Butterfly $s_0s_1s_2s_3$ $s_3s_1s_2s_0$	$s_0s_1s_2s_3$	$s_1s_2s_3s_3$	$s_2s_3s_3s_1$	$s_3s_3s_1s_2$	$s_3s_1s_2s_0$	✗
9. Exchange $s_0s_1s_2s_3$ $s_0\overline{s_1}s_2s_3$	$s_0s_1s_2s_3$	$s_1s_2s_3s_0$	$s_2s_3s_0\overline{s_1}$	$s_3s_0\overline{s_1}s_2$	$s_0\overline{s_1}s_2s_3$	✓

4-Path Omega Network with Size of Network (N) = 16, and Size of Switch (B) = 8

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2 (Destination)	
1.Bit Reversal $s_0s_1s_2s_3 \otimes\otimes s_3s_2s_1s_0$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_3$	$s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3 \otimes\otimes s_2s_3s_0s_1$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_2$	$s_2s_3s_0s_1$	✓
3. Perfect Shuffle $s_0s_1s_2s_3 \otimes\otimes s_1s_2s_3s_0$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_1$	$s_1s_2s_3s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3 \otimes\otimes \overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes \overline{s}_0$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3$	✓
5. Bit Shuffle $s_0s_1s_2s_3 \otimes\otimes s_0s_2s_1s_3$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_0$	$s_0s_2s_1s_3$	✓
6. Unshuffle $s_0s_1s_2s_3 \otimes\otimes s_3s_0s_1s_2$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_3$	$s_3s_0s_1s_2$	✗
7. Shuffle Row Major $s_0s_1s_2s_3 \otimes\otimes s_0s_2s_1s_3$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_0$	$s_0s_2s_1s_3$	✓
8. Butterfly $s_0s_1s_2s_3 \otimes\otimes s_3s_1s_2s_0$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_3$	$s_3s_1s_2s_0$	✗
9. Exchange $s_0s_1s_2s_3 \otimes\otimes s_0\overline{s}_1s_2s_3$	$s_0s_1s_2s_3$	$s_3 \otimes\otimes s_0$	$s_0\overline{s}_1s_2s_3$	✓

1-Path Omega Network with Size of Network (N) = 32, and Size of Switch (B) = 2

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages						Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4$ $s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_4$	$s_2s_3s_4s_4s_3$	$s_3s_4s_4s_3s_2$	$s_4s_4s_3s_2s_1$	$s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4$ $s_2s_3s_4s_0s_1$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_2$	$s_2s_3s_4s_2s_3$	$s_3s_4s_2s_3s_4$	$s_4s_2s_3s_4s_0$	$s_2s_3s_4s_0s_1$	✗
3. Perfect Shuffle $s_0s_1s_2s_3s_4$ $s_1s_2s_3s_4s_0$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_1$	$s_2s_3s_4s_1s_2$	$s_3s_4s_1s_2s_3$	$s_4s_1s_2s_3s_4$	$s_1s_2s_3s_4s_0$	✗
4. Vector Reversal $s_0s_1s_2s_3s_4$ $\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4\overline{s}_0$	$s_2s_3s_4\overline{s}_0\overline{s}_1$	$s_3s_4\overline{s}_0\overline{s}_1\overline{s}_2$	$s_4\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4$ $s_0s_2s_4s_1s_3$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_0$	$s_2s_3s_4s_0s_2$	$s_3s_4s_0s_2s_4$	$s_4s_0s_2s_4s_1$	$s_0s_2s_4s_1s_3$	✗
6. Unshuffle $s_0s_1s_2s_3s_4$ $s_4s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_4$	$s_2s_3s_4s_4s_0$	$s_3s_4s_4s_0s_1$	$s_4s_4s_0s_1s_2$	$s_4s_0s_1s_2s_3$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4$ $s_0s_3s_1s_4s_2$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_0$	$s_2s_3s_4s_0s_3$	$s_3s_4s_0s_3s_1$	$s_4s_0s_3s_1s_4$	$s_0s_3s_1s_4s_2$	✗
8. Butterfly $s_0s_1s_2s_3s_4$ $s_4s_1s_2s_3s_0$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_4$	$s_2s_3s_4s_4s_1$	$s_3s_4s_4s_1s_2$	$s_4s_4s_1s_2s_3$	$s_4s_1s_2s_3s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4$ $s_0s_1\overline{s}_2s_3s_4$	$s_0s_1s_2s_3s_4$	$s_1s_2s_3s_4s_0$	$s_2s_3s_4s_0s_1$	$s_3s_4s_0s_1\overline{s}_2$	$s_4s_0s_1\overline{s}_2s_3$	$s_0s_1\overline{s}_2s_3s_4$	✓

2-Path Omega Network with Size of Network (N) = 32, and Size of Switch (B) = 4

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages				Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4 \otimes s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_4$	$s_4 \otimes s_4s_3s_2$	$s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4 \otimes s_2s_3s_4s_0s_1$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_2$	$s_4 \otimes s_2s_3s_4$	$s_2s_3s_4s_0s_1$	✗
3. Perfect Shuffle $s_0s_1s_2s_3s_4 \otimes s_1s_2s_3s_4s_0$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_1$	$s_4 \otimes s_1s_2s_3$	$s_1s_2s_3s_4s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4 \otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes \bar{s}_0$	$s_4 \otimes \bar{s}_0\bar{s}_1\bar{s}_2$	$\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4 \otimes s_0s_2s_4s_1s_3$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_0$	$s_4 \otimes s_0s_2s_4$	$s_0s_2s_4s_1s_3$	✗
6. Unshuffle $s_0s_1s_2s_3s_4 \otimes s_4s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_4$	$s_4 \otimes s_4s_0s_1$	$s_4s_0s_1s_2s_3$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4 \otimes s_0s_3s_1s_4s_2$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_0$	$s_4 \otimes s_0s_3s_1$	$s_0s_3s_1s_4s_2$	✓
8. Butterfly $s_0s_1s_2s_3s_4 \otimes s_4s_1s_2s_3s_0$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_4$	$s_4 \otimes s_4s_1s_2$	$s_4s_1s_2s_3s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4 \otimes s_0s_1\bar{s}_2s_3s_4$	$s_0s_1s_2s_3s_4$	$s_2s_3s_4 \otimes s_0$	$s_4 \otimes s_0s_1\bar{s}_2$	$s_0s_1\bar{s}_2s_3s_4$	✓

8-Path Omega Network with Size of Network (N) = 32, and Size of Switch (B) = 16

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_4$	$s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_2s_3s_4s_0s_1$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_2$	$s_2s_3s_4s_0s_1$	✓
3. Perfect Shuffle $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_1s_2s_3s_4s_0$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_1$	$s_1s_2s_3s_4s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4 \otimes\otimes\otimes \overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes \overline{s}_0$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_0s_2s_4s_1s_3$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_0$	$s_0s_2s_4s_1s_3$	✓
6. Unshuffle $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_4s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_4$	$s_4s_0s_1s_2s_3$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_0s_3s_1s_4s_2$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_0$	$s_0s_3s_1s_4s_2$	✓
8. Butterfly $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_4s_1s_2s_3s_0$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_4$	$s_4s_1s_2s_3s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4 \otimes\otimes\otimes s_0s_1\overline{s}_2s_3s_4$	$s_0s_1s_2s_3s_4$	$s_4 \otimes\otimes\otimes s_0$	$s_0s_1\overline{s}_2s_3s_4$	✓

1-Path Omega Network with Size of Network (N) = 64, and Size of Switch (B) = 2

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages							Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5$ $s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_5$	$s_2s_3s_4s_5\ s_5s_4$	$s_3s_4s_5\ s_5s_4s_3$	$s_4s_5\ s_5s_4s_3s_2$	$s_5\ s_5s_4s_3s_2s_1$	$s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5$ $s_3s_4s_5s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_3$	$s_2s_3s_4s_5\ s_3s_4$	$s_3s_4s_5\ s_3s_4s_5$	$s_4s_5\ s_3s_4s_5s_0$	$s_5\ s_3s_4s_5s_0s_1$	$s_3s_4s_5s_0s_1s_2$	✗
2. Perfect Shuffle $s_0s_1s_2s_3s_4s_5$ $s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_1$	$s_2s_3s_4s_5\ s_1s_2$	$s_3s_4s_5\ s_1s_2s_3$	$s_4s_5\ s_1s_2s_3s_4$	$s_5\ s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5s_0$	✗
4. Vector Reversal $s_0s_1s_2s_3s_4s_5$ $\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ \bar{s}_0$	$s_2s_3s_4s_5\ \bar{s}_0\bar{s}_1$	$s_3s_4s_5\ \bar{s}_0\bar{s}_1\bar{s}_2$	$s_4s_5\ \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3$	$s_5\ \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4$	$\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5$ $s_0s_2s_4s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_0$	$s_2s_3s_4s_5\ s_0s_2$	$s_3s_4s_5\ s_0s_2s_4$	$s_4s_5\ s_0s_2s_4s_1$	$s_5\ s_0s_2s_4s_1s_3$	$s_0s_2s_4s_1s_3s_5$	✗
6. Unshuffle $s_0s_1s_2s_3s_4s_5$ $s_5s_0s_1s_2s_3s_4$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_5$	$s_2s_3s_4s_5\ s_5s_0$	$s_3s_4s_5\ s_5s_0s_1$	$s_4s_5\ s_5s_0s_1s_2$	$s_5\ s_5s_0s_1s_2s_3$	$s_5s_0s_1s_2s_3s_4$	✗
. Shuffle Row Major $s_0s_1s_2s_3s_4s_5$ $s_0s_3s_1s_4s_2s_5$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_0$	$s_2s_3s_4s_5\ s_0s_3$	$s_3s_4s_5\ s_0s_3s_1$	$s_4s_5\ s_0s_3s_1s_4$	$s_5\ s_0s_3s_1s_4s_2$	$s_0s_3s_1s_4s_2s_5$	✗
8. Butterfly $s_0s_1s_2s_3s_4s_5$ $s_5s_1s_2s_3s_4s_0$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_5$	$s_2s_3s_4s_5\ s_5s_1$	$s_3s_4s_5\ s_5s_1s_2$	$s_4s_5\ s_5s_1s_2s_3$	$s_5\ s_5s_1s_2s_3s_4$	$s_5s_1s_2s_3s_4s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5$ $s_0s_1\bar{s}_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5\ s_0$	$s_2s_3s_4s_5\ s_0s_1$	$s_3s_4s_5\ s_0s_1\bar{s}_2$	$s_4s_5\ s_0s_1\bar{s}_2s_3$	$s_5\ s_0s_1\bar{s}_2s_3s_4$	$s_0s_1\bar{s}_2s_3s_4s_5$	✓

4-Path Omega Network with Size of Network (N) = 64, and Size of Switch (B) = 16

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2 (Destination)	
1.Bit Reversal $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_5s_4$	$s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_3s_4s_5s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_3s_4$	$s_3s_4s_5s_0s_1s_2$	✗
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_1s_2$	$s_1s_2s_3s_4s_5s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5 \otimes\otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes \bar{s}_0\bar{s}_1$	$\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_0s_2s_4s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_0s_2$	$s_0s_2s_4s_1s_3s_5$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_5s_0s_1s_2s_3s_4$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_5s_0$	$s_5s_0s_1s_2s_3s_4$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_0s_3s_1s_4s_2s_5$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_0s_3$	$s_0s_3s_1s_4s_2s_5$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_5s_1s_2s_3s_4s_0$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_5s_1$	$s_5s_1s_2s_3s_4s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5 \otimes\otimes s_0s_1\bar{s}_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5$	$s_4s_5 \otimes\otimes s_0s_1$	$s_0s_1\bar{s}_2s_3s_4s_5$	✓

16-Path Omega Network with Size of Network (N) = 64, and Size of Switch (B) = 32

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2 (Destination)	
1.Bit Reversal $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_5$	$s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_3s_4s_5s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_3$	$s_3s_4s_5s_0s_1s_2$	✓
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_1$	$s_1s_2s_3s_4s_5s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes \overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes \overline{s}_0$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_0s_2s_4s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_0$	$s_0s_2s_4s_1s_3s_5$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_5s_0s_1s_2s_3s_4$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_5$	$s_5s_0s_1s_2s_3s_4$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_0s_3s_1s_4s_2s_5$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_0$	$s_0s_3s_1s_4s_2s_5$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_5s_1s_2s_3s_4s_0$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_5$	$s_5s_1s_2s_3s_4s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5 \otimes\otimes\otimes\otimes s_0s_1\overline{s}_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5$	$s_5 \otimes\otimes\otimes\otimes s_0$	$s_0s_1\overline{s}_2s_3s_4s_5$	✓

1-Path Omega Network with Size of Network (N) = 128, and Size of Switch (B) = 2

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages								Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6$ $s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_6	$s_2s_3s_4$ s_5s_6 s_6s_5	$s_3s_4s_5$ s_6 $s_6s_5s_4$	$s_4s_5s_6$ $s_6s_5s_4s_3$	s_5s_6 s_6 $s_5s_4s_3s_2$	s_6 s_6s_5 $s_4s_3s_2s_1$	$s_6s_5s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6$ $s_3s_4s_5s_6s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_3	$s_2s_3s_4$ s_5s_6 s_3s_4	$s_3s_4s_5$ s_6 $s_3s_4s_5$	$s_4s_5s_6$ $s_3s_4s_5s_6$	s_5s_6 s_3 $s_4s_5s_6s_0$	s_6 s_3s_4 $s_5s_6s_0s_1$	$s_3s_4s_5s_6s_0s_1s_2$	✕
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6$ $s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_1	$s_2s_3s_4$ s_5s_6 s_1s_2	$s_3s_4s_5$ s_6 $s_1s_2s_3$	$s_4s_5s_6$ $s_1s_2s_3s_4$	s_5s_6 s_1 $s_2s_3s_4s_5$	s_6 s_1s_2 $s_3s_4s_5s_6$	$s_1s_2s_3s_4s_5s_6s_0$	✕
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6$ $\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ $\overline{s_0}$	$s_2s_3s_4$ s_5s_6 $\overline{s_0}\overline{s_1}$	$s_3s_4s_5$ s_6 $\overline{s_0}\overline{s_1}\overline{s_2}$	$s_4s_5s_6$ $\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}$	s_5s_6 $\overline{s_0}$ $\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}$	s_6 $\overline{s_0}\overline{s_1}$ $\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}$	$\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6$ $s_0s_2s_4s_6s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_0	$s_2s_3s_4$ s_5s_6 s_0s_2	$s_3s_4s_5$ s_6 $s_0s_2s_4$	$s_4s_5s_6$ $s_0s_2s_4s_6$	s_5s_6 s_0 $s_2s_4s_6s_1$	s_6 s_0s_2 $s_4s_6s_1s_3$	$s_0s_2s_4s_6s_1s_3s_5$	✕
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6$ $s_6s_0s_1s_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_6	$s_2s_3s_4$ s_5s_6 s_6s_0	$s_3s_4s_5$ s_6 $s_6s_0s_1$	$s_4s_5s_6$ $s_6s_0s_1s_2$	s_5s_6 s_6 $s_0s_1s_2s_3$	s_6 s_6s_0 $s_1s_2s_3s_4$	$s_6s_0s_1s_2s_3s_4s_5$	✕

1-Path Omega Network with Size of Network (N) = 128, and Size of Switch (B) = 2

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages								Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7 (Destination)	
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6$ $s_0s_4s_1s_5s_2s_6s_3$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_0	$s_2s_3s_4$ s_5s_6 s_0s_4	$s_3s_4s_5$ s_6 $s_0s_4s_1$	$s_4s_5s_6$ $s_0s_4s_1s_5$	s_5s_6 s_0 $s_4s_1s_5s_2$	s_6 s_0s_4 $s_1s_5s_2s_6$	$s_0s_4s_1s_5s_2s_6s_3$	✕
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6$ $s_6s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_6	$s_2s_3s_4$ s_5s_6 s_6s_1	$s_3s_4s_5$ s_6 $s_6s_1s_2$	$s_4s_5s_6$ $s_6s_1s_2s_3$	s_5s_6 s_6 $s_1s_2s_3s_4$	s_6 s_6s_1 $s_2s_3s_4s_5$	$s_6s_1s_2s_3s_4s_5s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6$ $s_0s_1s_2\overline{s}_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6$	$s_1s_2s_3$ $s_4s_5s_6$ s_0	$s_2s_3s_4$ s_5s_6 s_0s_1	$s_3s_4s_5$ s_6 $s_0s_1s_2$	$s_4s_5s_6$ $s_0s_1s_2\overline{s}_3$	s_5s_6 s_0 $s_1s_2\overline{s}_3s_4$	s_6 s_0s_1 $s_2\overline{s}_3s_4s_5$	$s_0s_1s_2\overline{s}_3s_4s_5s_6$	✓

2-Path Omega Network with Size of Network (N) = 128, and Size of Switch (B) = 4

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages					Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_6$	$s_4s_5s_6 \otimes s_6s_5s_4$	$s_6 \otimes s_6s_5s_4s_3s_2$	$s_6s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6 \otimes s_3s_4s_5s_6s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_3$	$s_4s_5s_6 \otimes s_3s_4s_5$	$s_6 \otimes s_3s_4s_5s_6s_0$	$s_3s_4s_5s_6s_0s_1s_2$	✗
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_1$	$s_4s_5s_6 \otimes s_1s_2s_3$	$s_6 \otimes s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5s_6s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes \overline{s_0}$	$s_4s_5s_6 \otimes \overline{s_0}\overline{s_1}\overline{s_2}$	$s_6 \otimes \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}$	$\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes s_0s_2s_4s_6s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_0$	$s_4s_5s_6 \otimes s_0s_2s_4$	$s_6 \otimes s_0s_2s_4s_6s_1$	$s_0s_2s_4s_6s_1s_3s_5$	✗
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes s_6s_0s_1s_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_6$	$s_4s_5s_6 \otimes s_6s_0s_1$	$s_6 \otimes s_6s_0s_1s_2s_3$	$s_6s_0s_1s_2s_3s_4s_5$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6 \otimes s_0s_4s_1s_5s_2s_6s_3$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_0$	$s_4s_5s_6 \otimes s_0s_4s_1$	$s_6 \otimes s_0s_4s_1s_5s_2$	$s_0s_4s_1s_5s_2s_6s_3$	✗
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6 \otimes s_6s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_6$	$s_4s_5s_6 \otimes s_6s_1s_2$	$s_6 \otimes s_6s_1s_2s_3s_4$	$s_6s_1s_2s_3s_4s_5s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5s_6 \otimes s_0s_1s_2\overline{s_3}s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6$	$s_2s_3s_4s_5s_6 \otimes s_0$	$s_4s_5s_6 \otimes s_0s_1s_2$	$s_6 \otimes s_0s_1s_2\overline{s_3}s_4$	$s_0s_1s_2\overline{s_3}s_4s_5s_6$	✓

4-Path Omega Network with Size of Network (N) = 128, and Size of Switch (B) = 8

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages				Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage 2	Stage3 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_6$	$s_6 \otimes\otimes s_6s_5s_4s_3$	$s_6s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_3s_4s_5s_6s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_3$	$s_6 \otimes\otimes s_3s_4s_5s_6$	$s_3s_4s_5s_6s_0s_1s_2$	✗
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_1$	$s_6 \otimes\otimes s_1s_2s_3s_4$	$s_1s_2s_3s_4s_5s_6s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes \bar{s}_0$	$s_6 \otimes\otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3$	$\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_0s_2s_4s_6s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_0$	$s_6 \otimes\otimes s_0s_2s_4s_6$	$s_0s_2s_4s_6s_1s_3s_5$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_6s_0s_1s_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_6$	$s_6 \otimes\otimes s_6s_0s_1s_2$	$s_6s_0s_1s_2s_3s_4s_5$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_0s_4s_1s_5s_2s_6s_3$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_0$	$s_6 \otimes\otimes s_0s_4s_1s_5$	$s_0s_4s_1s_5s_2s_6s_3$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_6s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_6$	$s_6 \otimes\otimes s_6s_1s_2s_3$	$s_6s_1s_2s_3s_4s_5s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes s_0s_1s_2\bar{s}_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6$	$s_3s_4s_5s_6 \otimes\otimes s_0$	$s_6 \otimes\otimes s_0s_1s_2\bar{s}_3$	$s_0s_1s_2\bar{s}_3s_4s_5s_6$	✓

8-Path Omega Network with Size of Network (N) = 128, and Size of Switch (B) = 32

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_6s_5$	$s_6s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_3s_4s_5s_6s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_3s_4$	$s_3s_4s_5s_6s_0s_1s_2$	✓
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_1s_2$	$s_1s_2s_3s_4s_5s_6s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes \bar{s}_0\bar{s}_1$	$\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_0s_2s_4s_6s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_0s_2$	$s_0s_2s_4s_6s_1s_3s_5$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_6s_0s_1s_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_6s_0$	$s_6s_0s_1s_2s_3s_4s_5$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_0s_4s_1s_5s_2s_6s_3$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_0s_4$	$s_0s_4s_1s_5s_2s_6s_3$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_6s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_6s_1$	$s_6s_1s_2s_3s_4s_5s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes s_0s_1s_2\bar{s}_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes s_0s_1$	$s_0s_1s_2\bar{s}_3s_4s_5s_6$	✓

32-Path Omega Network with Size of Network (N) = 128, and Size of Switch (B) = 64

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_6$	$s_6s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_3s_4s_5s_6s_0s_1s_2$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_3$	$s_3s_4s_5s_6s_0s_1s_2$	✓
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_1$	$s_1s_2s_3s_4s_5s_6s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ \overline{s_0}$	$\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}$	✓
5. Bit Shuffle] $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_0s_2s_4s_6s_1s_3s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_0$	$s_0s_2s_4s_6s_1s_3s_5$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_6s_0s_1s_2s_3s_4s_5$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_6$	$s_6s_0s_1s_2s_3s_4s_5$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_0s_4s_1s_5s_2s_6s_3$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_0$	$s_0s_4s_1s_5s_2s_6s_3$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_6s_1s_2s_3s_4s_5s_0$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_6$	$s_6s_1s_2s_3s_4s_5s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5s_6 \otimes\otimes\otimes\otimes\ s_0s_1s_2\overline{s_3}s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6$	$s_5s_6 \otimes\otimes\otimes\otimes\ s_0$	$s_0s_1s_2\overline{s_3}s_4s_5s_6$	✓

1-Path Omega Network with Size of Network (N) = 256, and Size of Switch (B) = 2

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages									Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7\ s_7$	$s_2s_3s_4s_5$ $s_6s_7\ s_7s_6$	$s_3s_4s_5s_6$ $s_7\ s_7s_6s_5$	$s_4s_5s_6s_7$ $s_7s_6s_5s_4$	$s_5s_6s_7\ s_7$ $s_6s_5s_4s_3$	$s_6s_7\ s_7s_6$ $s_5s_4s_3s_2$	$s_7\ s_7s_6s_5$ $s_4s_3s_2s_1$	$s_7s_6s_5s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_4s_5s_6s_7s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7\ s_4$	$s_2s_3s_4s_5$ $s_6s_7\ s_4s_5$	$s_3s_4s_5s_6$ $s_7\ s_4s_5s_6$	$s_4s_5s_6s_7$ $s_4s_5s_6s_7$	$s_5s_6s_7\ s_4$ $s_5s_6s_7s_0$	$s_6s_7\ s_4s_5$ $s_6s_7s_0s_1$	$s_7\ s_4s_5s_6$ $s_7s_0s_1s_2$	$s_4s_5s_6s_7s_0s_1s_2s_3$	✕
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7\ s_1$	$s_2s_3s_4s_5$ $s_6s_7\ s_1s_2$	$s_3s_4s_5s_6$ $s_7\ s_1s_2s_3$	$s_4s_5s_6s_7$ $s_1s_2s_3s_4$	$s_5s_6s_7\ s_1$ $s_2s_3s_4s_5$	$s_6s_7\ s_1s_2$ $s_3s_4s_5s_6$	$s_7\ s_1s_2s_3$ $s_4s_5s_6s_7$	$s_1s_2s_3s_4s_5s_6s_7s_0$	✕
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7$ $\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7\ \overline{s}_0$	$s_2s_3s_4s_5$ $s_6s_7\ \overline{s}_0\overline{s}_1$	$s_3s_4s_5s_6$ $s_7\ \overline{s}_0\overline{s}_1\overline{s}_2$	$s_4s_5s_6s_7$ $\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3$	$s_5s_6s_7\ \overline{s}_0$ $\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4$	$s_6s_7\ \overline{s}_0\overline{s}_1$ $\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5$	$s_7\ \overline{s}_0\overline{s}_1\overline{s}_2$ $\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_0s_2s_4s_6s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7\ s_0$	$s_2s_3s_4s_5$ $s_6s_7\ s_0s_2$	$s_3s_4s_5s_6$ $s_7\ s_0s_2s_4$	$s_4s_5s_6s_7$ $s_0s_2s_4s_6$	$s_5s_6s_7\ s_0$ $s_2s_4s_6s_1$	$s_6s_7\ s_0s_2$ $s_4s_6s_1s_3$	$s_7\ s_0s_2s_4$ $s_6s_1s_3s_5$	$s_0s_2s_4s_6s_1s_3s_5s_7$	✕
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_7s_0s_1s_2s_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7\ s_7$	$s_2s_3s_4s_5$ $s_6s_7\ s_7s_0$	$s_3s_4s_5s_6$ $s_7\ s_7s_0s_1$	$s_4s_5s_6s_7$ $s_7s_0s_1s_2$	$s_5s_6s_7\ s_7$ $s_0s_1s_2s_3$	$s_6s_7\ s_7s_0$ $s_1s_2s_3s_4$	$s_7\ s_7s_0s_1$ $s_2s_3s_4s_5$	$s_7s_0s_1s_2s_3s_4s_5s_6$	✕

1-Path Omega Network with Size of Network (N) = 256, and Size of Switch (B) = 2

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages									Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8 (Destination)	
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_0s_4s_1s_5s_2s_6s_3s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7$ s_0	$s_2s_3s_4s_5$ s_6s_7 s_0s_4	$s_3s_4s_5s_6$ s_7 $s_0s_4s_1$	$s_4s_5s_6s_7$ $s_0s_4s_1s_5$	$s_5s_6s_7$ s_0 $s_4s_1s_5s_2$	s_6s_7 s_0s_4 $s_1s_5s_2s_6$	s_7 $s_0s_4s_1$ $s_5s_2s_6s_3$	$s_0s_4s_1s_5s_2s_6s_3s_7$	✕
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_7s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7$ s_7	$s_2s_3s_4s_5$ s_6s_7 s_7s_1	$s_3s_4s_5s_6$ s_7 $s_7s_1s_2$	$s_4s_5s_6s_7$ $s_7s_1s_2s_3$	$s_5s_6s_7$ s_7 $s_1s_2s_3s_4$	s_6s_7 s_7s_1 $s_2s_3s_4s_5$	s_7 $s_7s_1s_2$ $s_3s_4s_5s_6$	$s_7s_1s_2s_3s_4s_5s_6s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7$ $s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4$ $s_5s_6s_7$ s_0	$s_2s_3s_4s_5$ s_6s_7 s_0s_1	$s_3s_4s_5s_6$ s_7 $s_0s_1s_2$	$s_4s_5s_6s_7$ $s_0s_1s_2\overline{s}_3$	$s_5s_6s_7$ s_0 $s_1s_2\overline{s}_3s_4$	s_6s_7 s_0s_1 $s_2\overline{s}_3s_4s_5$	s_7 $s_0s_1s_2$ $\overline{s}_3s_4s_5s_6$	$s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	✓

2-Path Omega Network with Size of Network (N) = 256, and Size of Switch (B) = 8

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages				Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage3 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_7s_6$	$s_6s_7 \otimes s_7s_6s_5s_4s_3$	$s_7s_6s_5s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_4s_5s_6s_7s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_4s_5$	$s_6s_7 \otimes s_4s_5s_6s_7s_0$	$s_4s_5s_6s_7s_0s_1s_2s_3$	✕
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_1s_2$	$s_6s_7 \otimes s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5s_6s_7s_0$	✓
4. Vector Reversa l $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}\overline{s_7}$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes \overline{s_0}\overline{s_1}$	$s_6s_7 \otimes \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}$	$\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}\overline{s_7}$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_0s_2s_4s_6s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_0s_2$	$s_6s_7 \otimes s_0s_2s_4s_6s_1$	$s_0s_2s_4s_6s_1s_3s_5s_7$	✕
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_7s_0s_1s_2s_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_7s_0$	$s_6s_7 \otimes s_7s_0s_1s_2s_3$	$s_7s_0s_1s_2s_3s_4s_5s_6$	✕
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_0s_4s_1s_5s_2s_6s_3s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_0s_4$	$s_6s_7 \otimes s_0s_4s_1s_5s_2$	$s_0s_4s_1s_5s_2s_6s_3s_7$	✕
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_7s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_7s_1$	$s_6s_7 \otimes s_7s_1s_2s_3s_4$	$s_7s_1s_2s_3s_4s_5s_6s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes s_0s_1s_2\overline{s_3}s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_3s_4s_5s_6s_7 \otimes s_0s_1$	$s_6s_7 \otimes s_0s_1s_2\overline{s_3}s_4$	$s_0s_1s_2\overline{s_3}s_4s_5s_6s_7$	✓

4-Path Omega Network with Size of Network (N) = 256, and Size of Switch (B) = 32

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_7s_6s_5$	$s_7s_6s_5s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_4s_5s_6s_7s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_4s_5s_6$	$s_4s_5s_6s_7s_0s_1s_2s_3$	✕
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_1s_2s_3$	$s_1s_2s_3s_4s_5s_6s_7s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes \overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes \overline{s}_0\overline{s}_1\overline{s}_2$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_0s_2s_4s_6s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_0s_2s_4$	$s_0s_2s_4s_6s_1s_3s_5s_7$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_7s_0s_1s_2s_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_7s_0s_1$	$s_7s_0s_1s_2s_3s_4s_5s_6$	✕
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_0s_4s_1s_5s_2s_6s_3s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_0s_4s_1$	$s_0s_4s_1s_5s_2s_6s_3s_7$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_7s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_7s_1s_2$	$s_7s_1s_2s_3s_4s_5s_6s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_5s_6s_7 \otimes\otimes s_0s_1s_2$	$s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	✓

16-Path Omega Network with Size of Network (N) = 256, and Size of Switch (B) = 64

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_7s_6$	$s_7s_6s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_4s_5s_6s_7s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_4s_5$	$s_4s_5s_6s_7s_0s_1s_2s_3$	✓
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_1s_2$	$s_1s_2s_3s_4s_5s_6s_7s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ \overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ \overline{s}_0\overline{s}_1$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_0s_2s_4s_6s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_0s_2$	$s_0s_2s_4s_6s_1s_3s_5s_7$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_7s_0s_1s_2s_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_7s_0$	$s_7s_0s_1s_2s_3s_4s_5s_6$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_0s_4s_1s_5s_2s_6s_3s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_0s_4$	$s_0s_4s_1s_5s_2s_6s_3s_7$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_7s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_7s_1$	$s_7s_1s_2s_3s_4s_5s_6s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\ s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_6s_7 \otimes\otimes\otimes\otimes\ s_0s_1$	$s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	✓

64-Path Omega Network with Size of Network (N) = 256, and Size of Switch (B) = 128

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capacity
	Stage 0 (Source)	Stage 1	Stage2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_7$	$s_7s_6s_5s_4s_3s_2s_1s_0$	✗
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_4s_5s_6s_7s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_4$	$s_4s_5s_6s_7s_0s_1s_2s_3$	✓
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_1$	$s_1s_2s_3s_4s_5s_6s_7s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ \overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ \overline{s}_0$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_0s_2s_4s_6s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_0$	$s_0s_2s_4s_6s_1s_3s_5s_7$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_7s_0s_1s_2s_3s_4s_5s_6$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_7$	$s_7s_0s_1s_2s_3s_4s_5s_6$	✗
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_0s_4s_1s_5s_2s_6s_3s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_0$	$s_0s_4s_1s_5s_2s_6s_3s_7$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_7s_1s_2s_3s_4s_5s_6s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_7$	$s_7s_1s_2s_3s_4s_5s_6s_0$	✗
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7$	$s_7 \otimes\otimes\otimes\otimes\otimes\otimes\ s_0$	$s_0s_1s_2\overline{s}_3s_4s_5s_6s_7$	✓

1-Path Omega Network with Size of Network (N) = 512, and Size of Switch (B) = 2

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages										Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_8s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_8$	$s_2s_3s_4s_5$ $s_6s_7s_8s_8s_7$	$s_3s_4s_5s_6$ $s_7s_8s_8s_7s_6$	$s_4s_5s_6s_7$ $s_8s_8s_7s_6s_5$	$s_5s_6s_7s_8$ $s_8s_7s_6s_5s_4$	$s_6s_7s_8s_8$ $s_7s_6s_5s_4s_3$	$s_7s_8s_8s_7$ $s_6s_5s_4s_3s_2$	$s_8s_8s_7s_6$ $s_5s_4s_3s_2s_1$	$s_8s_7s_6s_5$ $s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_4s_5s_6s_7s_8s_0s_1s_2s_3$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_4$	$s_2s_3s_4s_5$ $s_6s_7s_8s_4s_5$	$s_3s_4s_5s_6$ $s_7s_8s_4s_5s_6$	$s_4s_5s_6s_7$ $s_8s_4s_5s_6s_7$	$s_5s_6s_7s_8$ $s_4s_5s_6s_7s_8$	$s_6s_7s_8s_4$ $s_5s_6s_7s_8s_0$	$s_7s_8s_4s_5$ $s_6s_7s_8s_0s_1$	$s_8s_4s_5s_6$ $s_7s_8s_0s_1s_2$	$s_4s_5s_6s_7$ $s_8s_0s_1s_2s_3$	✕
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_1s_2s_3s_4s_5s_6s_7s_8s_0$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_1$	$s_2s_3s_4s_5$ $s_6s_7s_8s_1s_2$	$s_3s_4s_5s_6$ $s_7s_8s_1s_2s_3$	$s_4s_5s_6s_7$ $s_8s_1s_2s_3s_4$	$s_5s_6s_7s_8$ $s_1s_2s_3s_4s_5$	$s_6s_7s_8s_1$ $s_2s_3s_4s_5s_6$	$s_7s_8s_1s_2$ $s_3s_4s_5s_6s_7$	$s_8s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_0$	✕
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7\overline{s}_8$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8\overline{s}_0$	$s_2s_3s_4s_5$ $s_6s_7s_8\overline{s}_0\overline{s}_1$	$s_3s_4s_5s_6$ $s_7s_8\overline{s}_0\overline{s}_1\overline{s}_2$	$s_4s_5s_6s_7$ $s_8\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3$	$s_5s_6s_7s_8$ $\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4$	$s_6s_7s_8\overline{s}_0$ $\overline{s}_1\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5$	$s_7s_8\overline{s}_0\overline{s}_1$ $\overline{s}_2\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6$	$s_8\overline{s}_0\overline{s}_1\overline{s}_2$ $\overline{s}_3\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7$	$\overline{s}_0\overline{s}_1\overline{s}_2\overline{s}_3$ $\overline{s}_4\overline{s}_5\overline{s}_6\overline{s}_7\overline{s}_8$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_0s_2s_4s_6s_8s_1s_3s_5s_7$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_0$	$s_2s_3s_4s_5$ $s_6s_7s_8s_0s_2$	$s_3s_4s_5s_6$ $s_7s_8s_0s_2s_4$	$s_4s_5s_6s_7$ $s_8s_0s_2s_4s_6$	$s_5s_6s_7s_8$ $s_0s_2s_4s_6s_8$	$s_6s_7s_8s_0$ $s_2s_4s_6s_8s_1$	$s_7s_8s_0s_2$ $s_4s_6s_8s_1s_3$	$s_8s_0s_2s_4$ $s_6s_8s_1s_3s_5$	$s_0s_2s_4s_6$ $s_8s_1s_3s_5s_7$	✕
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_8s_0s_1s_2s_3s_4s_5s_6s_7$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_8$	$s_2s_3s_4s_5$ $s_6s_7s_8s_8s_0$	$s_3s_4s_5s_6$ $s_7s_8s_8s_0s_1$	$s_4s_5s_6s_7$ $s_8s_8s_0s_1s_2$	$s_5s_6s_7s_8$ $s_8s_0s_1s_2s_3$	$s_6s_7s_8s_8$ $s_0s_1s_2s_3s_4$	$s_7s_8s_8s_0$ $s_1s_2s_3s_4s_5$	$s_8s_8s_0s_1$ $s_2s_3s_4s_5s_6$	$s_8s_0s_1s_2$ $s_3s_4s_5s_6s_7$	✕

1-Path Omega Network with Size of Network (N) = 512, and Size of Switch (B) = 2

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages										Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9 (Destination)	
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_0s_5s_1s_6s_2s_7s_3s_8s_4$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_0$	$s_2s_3s_4s_5$ $s_6s_7s_8s_0s_5$	$s_3s_4s_5s_6$ $s_7s_8s_0s_5s_1$	$s_4s_5s_6s_7$ $s_8s_0s_5s_1s_6$	$s_5s_6s_7s_8$ $s_0s_5s_1s_6s_2$	$s_6s_7s_8s_0$ $s_5s_1s_6s_2s_7$	$s_7s_8s_0s_5$ $s_1s_6s_2s_7s_3$	$s_8s_0s_5s_1$ $s_6s_2s_7s_3s_8$	$s_0s_5s_1s_6$ $s_2s_7s_3s_8s_4$	✕
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_8s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_8$	$s_2s_3s_4s_5$ $s_6s_7s_8s_8s_1$	$s_3s_4s_5s_6$ $s_7s_8s_8s_1s_2$	$s_4s_5s_6s_7$ $s_8s_8s_1s_2s_3$	$s_5s_6s_7s_8$ $s_8s_1s_2s_3s_4$	$s_6s_7s_8s_8$ $s_1s_2s_3s_4s_5$	$s_7s_8s_8s_1$ $s_2s_3s_4s_5s_6$	$s_8s_8s_1s_2$ $s_3s_4s_5s_6s_7$	$s_8s_1s_2s_3$ $s_4s_5s_6s_7s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7s_8$ $s_0s_1s_2s_3\overline{s}_4s_5s_6s_7s_8$	$s_0s_1s_2s_3$ $s_4s_5s_6s_7s_8$	$s_1s_2s_3s_4$ $s_5s_6s_7s_8s_0$	$s_2s_3s_4s_5$ $s_6s_7s_8s_0s_1$	$s_3s_4s_5s_6$ $s_7s_8s_0s_1s_2$	$s_4s_5s_6s_7$ $s_8s_0s_1s_2s_3$	$s_5s_6s_7s_8$ $s_0s_1s_2s_3\overline{s}_4$	$s_6s_7s_8s_0$ $s_1s_2s_3\overline{s}_4s_5$	$s_7s_8s_0s_1$ $s_2s_3\overline{s}_4s_5s_6$	$s_8s_0s_1s_2$ $s_3\overline{s}_4s_5s_6s_7$	$s_0s_1s_2s_{34}$ $\overline{s}s_5s_6s_7s_8$	✓

2-Path Omega Network with Size of Network (N) = 512, and Size of Switch (B) = 4

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages						Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_8s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_8$	$s_4s_5s_6s_7s_8 \otimes s_8s_7s_6$	$s_6s_7s_8 \otimes s_8s_7s_6s_5s_4$	$s_8 \otimes s_8s_7s_6s_5s_4s_3s_2$	$s_8s_7s_6s_5s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_4s_5s_6s_7s_8s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_4$	$s_4s_5s_6s_7s_8 \otimes s_4s_5s_6$	$s_6s_7s_8 \otimes s_4s_5s_6s_7s_8$	$s_8 \otimes s_4s_5s_6s_7s_8s_0s_1$	$s_4s_5s_6s_7s_8s_0s_1s_2s_3$	✕
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_1s_2s_3s_4s_5s_6s_7s_8s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_1$	$s_4s_5s_6s_7s_8 \otimes s_1s_2s_3$	$s_6s_7s_8 \otimes s_1s_2s_3s_4s_5$	$s_8 \otimes s_1s_2s_3s_4s_5s_6s_7$	$s_1s_2s_3s_4s_5s_6s_7s_8s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6\bar{s}_7\bar{s}_8$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes \bar{s}_0$	$s_4s_5s_6s_7s_8 \otimes \bar{s}_0\bar{s}_1\bar{s}_2$	$s_6s_7s_8 \otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4$	$s_8 \otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6$	$\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6\bar{s}_7\bar{s}_8$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_0s_2s_4s_6s_8s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_0$	$s_4s_5s_6s_7s_8 \otimes s_0s_2s_4$	$s_6s_7s_8 \otimes s_0s_2s_4s_6s_8$	$s_8 \otimes s_0s_2s_4s_6s_8s_1s_3$	$s_0s_2s_4s_6s_8s_1s_3s_5s_7$	✕
5. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_8s_0s_1s_2s_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_8$	$s_4s_5s_6s_7s_8 \otimes s_8s_0s_1$	$s_6s_7s_8 \otimes s_8s_0s_1s_2s_3$	$s_8 \otimes s_8s_0s_1s_2s_3s_4s_5$	$s_8s_0s_1s_2s_3s_4s_5s_6s_7$	✕
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_0s_5s_1s_6s_2s_7s_3s_8s_4$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_0$	$s_4s_5s_6s_7s_8 \otimes s_0s_5s_1$	$s_6s_7s_8 \otimes s_0s_5s_1s_6s_2$	$s_8 \otimes s_0s_5s_1s_6s_2s_7s_3$	$s_0s_5s_1s_6s_2s_7s_3s_8s_4$	✕
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_8s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_8$	$s_4s_5s_6s_7s_8 \otimes s_8s_1s_2$	$s_6s_7s_8 \otimes s_8s_1s_2s_3s_4$	$s_8 \otimes s_8s_1s_2s_3s_4s_5s_6$	$s_8s_1s_2s_3s_4s_5s_6s_7s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes s_0s_1s_2s_3\bar{s}_4s_5s_6s_7s_8$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_2s_3s_4s_5s_6s_7s_8 \otimes s_0$	$s_4s_5s_6s_7s_8 \otimes s_0s_1s_2$	$s_6s_7s_8 \otimes s_0s_1s_2s_3\bar{s}_4$	$s_8 \otimes s_0s_1s_2s_3\bar{s}_4s_5s_6$	$s_0s_1s_2s_3\bar{s}_4s_5s_6s_7s_8$	✓

8-Path Omega Network with Size of Network (N) = 512, and Size of Switch (B) = 16

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages				Permutation Capability
	Stage 0 (Source)	Stage 1	Stage 2	Stage3 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_8s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_8$	$s_8 \otimes\otimes\otimes\ s_8s_7s_6s_5s_4$	$s_8s_7s_6s_5s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_4s_5s_6s_7s_8s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_4$	$s_8 \otimes\otimes\otimes\ s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8s_0s_1s_2s_3$	✕
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_1s_2s_3s_4s_5s_6s_7s_8s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_1$	$s_8 \otimes\otimes\otimes\ s_1s_2s_3s_4s_5$	$s_1s_2s_3s_4s_5s_6s_7s_8s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}\overline{s_7}\overline{s_8}$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ \overline{s_0}$	$s_8 \otimes\otimes\otimes\ \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}$	$\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}\overline{s_7}\overline{s_8}$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_0s_2s_4s_6s_8s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_0$	$s_8 \otimes\otimes\otimes\ s_0s_2s_4s_6s_8$	$s_0s_2s_4s_6s_8s_1s_3s_5s_7$	✕
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_8s_0s_1s_2s_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_8$	$s_8 \otimes\otimes\otimes\ s_8s_0s_1s_2s_3$	$s_8s_0s_1s_2s_3s_4s_5s_6s_7$	✕
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_0s_5s_1s_6s_2s_7s_3s_8s_4$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_0$	$s_8 \otimes\otimes\otimes\ s_0s_5s_1s_6s_2$	$s_0s_5s_1s_6s_2s_7s_3s_8s_4$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_8s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_8$	$s_8 \otimes\otimes\otimes\ s_8s_1s_2s_3s_4$	$s_8s_1s_2s_3s_4s_5s_6s_7s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_0s_1s_2s_3\overline{s_4}s_5s_6s_7s_8$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_4s_5s_6s_7s_8 \otimes\otimes\otimes\ s_0$	$s_8 \otimes\otimes\otimes\ s_0s_1s_2s_3\overline{s_4}$	$s_0s_1s_2s_3\overline{s_4}s_5s_6s_7s_8$	✓

32-Path Omega Network with Size of Network (N) = 512, and Size of Switch (B) = 128

Note: "✕" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capability
	Stage 0 (Source)	Stage 1	Stage2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_8s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_8s_7$	$s_8s_7s_6s_5s_4s_3s_2s_1s_0$	✕
2. Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_4s_5s_6s_7s_8s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_4s_5$	$s_4s_5s_6s_7s_8s_0s_1s_2s_3$	✓
3. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_1s_2s_3s_4s_5s_6s_7s_8s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_1s_2$	$s_1s_2s_3s_4s_5s_6s_7s_8s_0$	✓
4. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes \bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6\bar{s}_7\bar{s}_8$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes \bar{s}_0\bar{s}_1$	$\bar{s}_0\bar{s}_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6\bar{s}_7\bar{s}_8$	✓
5. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_0s_2s_4s_6s_8s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_0s_2$	$s_0s_2s_4s_6s_8s_1s_3s_5s_7$	✓
6. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_8s_0s_1s_2s_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_8s_0$	$s_8s_0s_1s_2s_3s_4s_5s_6s_7$	✕
7. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_0s_5s_1s_6s_2s_7s_3s_8s_4$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_0s_5$	$s_0s_5s_1s_6s_2s_7s_3s_8s_4$	✓
8. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_8s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_8s_1$	$s_8s_1s_2s_3s_4s_5s_6s_7s_0$	✕
9. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes s_0s_1s_2s_3\bar{s}_4s_5s_6s_7s_8$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes s_0s_1$	$s_0s_1s_2s_3\bar{s}_4s_5s_6s_7s_8$	✓

128-Path Omega Network with Size of Network (N) = 512, and Size of Switch (B) = 256

Note: "✗" means "non-admissible", "✓" means "admissible"

Permutation	The Terminal at Different Stages			Permutation Capability
	Stage 0 (Source)	Stage 1	Stage2 (Destination)	
1. Bit Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_8s_7s_6s_5s_4s_3s_2s_1s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_8$	$s_8s_7s_6s_5s_4s_3s_2s_1s_0$	✗
2.Matrix Transposition $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_4s_5s_6s_7s_8s_0s_1s_2s_3$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_4$	$s_4s_5s_6s_7s_8s_0s_1s_2s_3$	✓
2. Perfect Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_1s_2s_3s_4s_5s_6s_7s_8s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_1$	$s_1s_2s_3s_4s_5s_6s_7s_8s_0$	✓
3. Vector Reversal $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ \overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}\overline{s_7}\overline{s_8}$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ \overline{s_0}$	$\overline{s_0}\overline{s_1}\overline{s_2}\overline{s_3}\overline{s_4}\overline{s_5}\overline{s_6}\overline{s_7}\overline{s_8}$	✓
4. Bit Shuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_0s_2s_4s_6s_8s_1s_3s_5s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_0$	$s_0s_2s_4s_6s_8s_1s_3s_5s_7$	✓
5. Unshuffle $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_8s_0s_1s_2s_3s_4s_5s_6s_7$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_8$	$s_8s_0s_1s_2s_3s_4s_5s_6s_7$	✗
6. Shuffle Row Major $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_0s_5s_1s_6s_2s_7s_3s_8s_4$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_0$	$s_0s_5s_1s_6s_2s_7s_3s_8s_4$	✓
7. Butterfly $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_8s_1s_2s_3s_4s_5s_6s_7s_0$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_8$	$s_8s_1s_2s_3s_4s_5s_6s_7s_0$	✗
8. Exchange $s_0s_1s_2s_3s_4s_5s_6s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_0s_1s_2s_3\overline{s_4}s_5s_6s_7s_8$	$s_0s_1s_2s_3s_4s_5s_6s_7s_8$	$s_7s_8 \otimes\otimes\otimes\otimes\otimes\otimes\otimes\ s_0$	$s_0s_1s_2s_3\overline{s_4}s_5s_6s_7s_8$	✓

References

1. Duncan H. Lawrie "Access and Alignment of Data in an Array Processor", IEEE Transactions on Computer, December 1975

2. Krishnan Padmanabhun and Duncan H. Lawrie, "A Class of Redundant Path Multistage Interconnection Networks", IEEE Transactions on Computer, Vol. C-32, No.12, December 1975

3. Robert J. McMillen, "A Survey of Interconnection Networks", Proceedings of Globecom, 1984, pages 105-113

4. Hwang, Kai, and Briggs, Faye Alaye, "Computer Architecture and Parallel Processing", 1st ed., New York:Mcgraw-Hill, 1985

5. G.G. Veselovskii, M. F. Karavai, and S.M. Kuzneckik, "Switching Networks for SIMD Multiprocessor Computing Systems", Automatika i Telemekhanika, No.2, pp.3-39, February, 1989

6. Howard Jay Siegel, Wayne G. Nation, Clydie P. Kruskal, and Leonard M. Napolitano, Jr., "Using the Multistage Cube Network Topology in Parallel Supercomputers", Proceedings of the IEEE, Vol.77, No.12, December 1989

7. Lewis, Ted G. El-Rewini, Hesham and Kim, In-Kyu, "Introduction to parallel computing", 1st ed., Englewood Cliffs, NJ:Prentice Hall, c1992

8. Harrison, Peter G. Patel, and Naresh M, "Performance Modelling of Communication Networks and Computer Architectures", 1st ed., Wokingham : Addison-Wesley, c1993

9. Schutzer, Daniel, "Parallel Processing and the Future Data Center:Computing in the Land of the Lilliputians", 1st ed., New York:Van Nostrand Reinhold, 1994

10. Suresh Chalasani, C.S. Raghavendra, and Anujan Varma, "Fault-Tolerant Routing in MIN-based Supercomputers", Journal of Parallel and Distributed Computing 22(2): Pages 154-167 (1994)

11. Xiaojun Shen, Mao Xu, and Xiangzu Wang, "An optimal Algorithm for Permutation Admissibility to Multistage Interconnection Networks", IEEE Transactions on Computers, Vol.44, No.4, April 1995

12. Xiaojun Shen, "Optimal Realization of Any BPC Permutation on K-Extra-Stage Omega Networks",IEEE Transactions on Computers, Vol.44, No.5, May 1995

13. Xiaojun Shen, "An optimal O(NlgN) Algorithm for Permutation Admissibility to Extra-Stage Cube-Type Networks", IEEE Transactions on Computers, Vol.44, No.9, September 1995

14. Qing Hu, Xiaojun Shen and Weifa Liang, "Optimally Routing LC Permutation on K-Extra-Stage Cube-Type Networks", IEEE Transactions on Computers, Vol.45, No.1, January 1996

15. Zomaya, Albert Y., ed. "Parallel Computing:Paradigms and Applications", 1st ed., London:International Thomson Computer Pr., c1996

16. Zargham, Mehdi R., "Computer Architecture:Single and Parallel Systems", 1st ed., Upper Saddle River, NJ:Prentice Hall., c1996

17. Schwartz, M., "Broadband Integrated Networks", 1st ed., Upper Saddle River, NJ:Prentice Hall., c1996