On construction of 0-1 sorters from 2×2 switches

SO Kin Tai

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Information Engineering

June 2000

Abstract

A *0-1 sorter* means a sorter of single-bit inputs. It sends the signals 1 to the upper output addresses. By treating the single-bit signal as the activity indicator of input, the 0-1 sorter functions as a *concentrator*, which is an important device in telecommunication switching.

A *compressor* is a network which, under proper switching control, can not only function as a 0-1 sorter, but also as a *cyclic 0-1 sorter*, which can switch signals 1 to any segment of circularly consecutive output addresses. A *2X-interconnection network* constructs a compressor out of compressors (see Definition 3.3.3 and Theorem 3.3.6 of [9]), and thereby can be recursively invoked for the construction of indefinitely large compressors. One instance of a recursive 2X-construction is the baseline network appended with the *swap exchange* (Definition 3.4.9 of [9]).

Iterative cells [13] are 2×2 switching cells controlled by both in-band signals and a single-bit external signal called the *running parity*. In the special case of the baseline network appended with swap exchange, when the 2×2 switching cells in it are controlled as iterative cells, the resulting network can be controlled as a cyclic 0-1 sorter by a proper set of the running parities. The starting position of the circularly consecutive output addresses determines the initial values of running parities. We derive an explicit formula for this determination.

Although all recursive 2X-constructions give functionally equivalent

switches in the strong sense (Corollary 4.1.16 of [9]), they are not equally useful. For one thing, they incur different complexities in laying out the interstage exchanges. In fact, the baseline network is among those that incur the highest complexity in terms of the 2-layer Manhattan layout, while the *divide-and-conquer networks* (Definition 3.5.9 in [9]) achieve the lowest complexities (Theorems 4.2.15 in [9]). In view of this, we are interested in the generic 2X-interconnection rather than just those involved in the baseline network appended with swap exchange. We derive an explicit formula for the initialization of running parities that control a generic 2X-construction network into a cyclic 0-1 sorter.

摘要

一個 0-1 排序器表示只接收單位元輸入的排序器。它把信號'1'送到較上的輸出。當那個單位元輸入是活動位時，0-1 排序器便成爲一個集中器。集中器在電訊交換中是一個重要的部份。

一個壓縮器是一個網絡。在正確的交換控制之下，壓縮器不單可以控制成 0-1 排序器，更可成循環 0-1 排序器 --- 它可把信號'1'輸出到任可一段循環而連續的位址。2X 互連網絡利用壓縮器建造壓縮器(詳見[9]的定義 3.3.3 與定理 3.3.6)，因此可以造出無限大的壓縮器。遞歸 2X 建造法的其中一個例子就是基線網絡加接一個對調交換(見[9]的定義 3.4.9)。

循環單元[13]是一種由同頻帶信號與一個外來單位元信號所控制的交換單元，而我們定義那一個外來信號爲運行奇偶位元。當所有基線網絡加接對調交換中的單元爲循環單元時，只要正確設定當中的運行奇偶位元，這一個網絡便可成爲一個循環 0-1 排序器。而運行奇偶位元的設定是取決於那一段循環而連續位址的首個位址。我們尋出一組顯式方程式去決定運行奇偶單元的始值。

雖然由遞歸 2X 建造法，可以得到作用上相同的各個交換網絡(見[9]的推論 4.1.16)，它們不是同樣有效的。其中，排與排之間不同的交換有不同的複雜性。事實上，基線網絡有最高的複雜性，而 divide-and-conquer 網絡(見[9]的定義 3.5.9)的則最低(見[9]的定理 4.2.15)。因此，我們除了基線網絡之外，更研究類屬 2X 互連網絡，我們尋出一租顯性方程式控制類屬 2X 互連網絡，令它成爲一個循環 0-1 排序器。

Acknowledgement

I would like to thank Prof. S. – Y. R. Li for his thoughtful guidance. Also, I would like to thank all the members in Telephone Lab for their willingness to share the knowledge.

Table of contents

List of Figures

Chapter 1 Introduction

The *0-1 sorter* plays an important role in many applications, for example, in concentration of active packet in telecommunication switching. In this chapter, 0-1 sorter and concentrator are defined. In addition, there will be a review of literatures covering sorting algorithms and constructions of 0-1 sorters. Finally, there will a section on the comparison of those constructions reviewed.

1.1 The of 0-1 sorter and concentrator

0-1 sorter serves as a significant module in telecommunication and switching. For example, it is stated [14] that the very high throughput required in broadband packet switches can be met by adopting banyan self-routing interconnection networks. Relatively high traffic levels are expected on the packet channels terminating on the inlets of these networks. The low efficiency of packet channels at the user-network interfaces requires the process of concentration to interface the broadband network with the low-rate inlets. The basic function of a 0-1 sorter in broadband switching is to multiplex traffic streams offered at the user-network interface, so that high utilization is achieved in the packet channels entering the broadband network. The activity of a packet is usually represented by the activity bit, which is either zero or one, so that a 0-1 sorter is capable of selecting those active ones among all the input channels for multiplexing. As a result, it has been widely applied for separating active packets from inactive packets [2, 3, 11 and 15].

Beside the 0-1 sorter, there is another switch called *compressor* [9]. A compressor can be viewed as a general form of 0-1 sorter that the starting point of the concentrated packet string could be any among all the output ports for compressor, rather than the first output port. A compressor receives active packets from the input ports and distributes them in cyclic fashion across the output buffers, which are located at the output ports of the network. The number of packets in each output buffer could be differ at most by one among all output ports, thereby achieving an efficient use of the buffer space. A similar device that is called "distributor" is described in [5 and 6]. Although the compressor plays an important role in the efficient usage of buffers, there is not many literatures cover on compressor. With the usefulness of compressor regarding the lack of literature available, it is worth studying further into the implementation and architecture of it.

Before going into the review of those previously proposed 0-1 sorters and concentrators, clarification have to be made. It may be mixed up that 0-1 sorter is equivalent to a concentrator, actually they are not.

Definition 1.1.1 By an M-to-N concentrator, $M \geq N$, it means an M-input-M-output device for routing single-bit signals such that the upper N outputs are all greater than or equal to the lower M-N outputs.

In Definition 1.1.1, it can be seen there is an ordering among the inputs.

Since it is a device for routing single-bit signals, the ordering is 1 (active) > 0 (idle), where the bit is representing the activity of the signal. Actually, the definition of 0-1 sorter is built upon the definition of concentrator, and it is given as follow.

Definition 1.1.2 An $M \times M$ device is a 0-1 sorter if it is an M-to-N concentrator for *all* $N \leq M$. An $M \times M$ binary sorter will also be referred as an M-input concentrator in the thesis.

In another words, a 0-1 sorter is similar to an M-to-N concentrator, except that N can be any value, provided that N is not larger than M. For example, an 8-to-4 concentrator guarantees that the first four outputs must be larger than or equal to the last four outputs. Suppose that five 1's are presented at the input side of the 8-to-4 concentrator, there is no guarantee that the fifth output of the 8-to-4 concentrator must be a 1. On the other hand, for the same pattern presented at the input side of an 8-input concentrator, the first five outputs must be 1's while the last three must be 0's.

1.2 Review of literature on constructions

In this section, some previously proposed sorting algorithms as well as constructions of concentrators and 0-1 sorters are reviewed. They are odd-even merging network, the fast knockout algorithm, reverse banyan network

prepended by running sum adder and the recursive construction of 0-1 sorter. Of the final three of the architectures, the latter two can be regarded as 0-1 sorters, while the fast knockout can't be. The fact that a fast knockout concentrator is not qualified to be a 0-1 sorter will be illustrated by an example.

1.2.1 Odd-even merging network

Merging is the process of arranging two descendingly-ordered lists of number into one descendingly-ordered list. A sorting algorithm was presented in [1] by invoking the iterative rule for odd-even merging, and this iterative rule is illustrated by Figure 1.1. Suppose we have two ordered sequences, and we want to form a single ordered sequence out of the two sequences. It can be seen in Figure 1.1 that the odd elements of the two sequences are diverted to a merger of smaller size, while the even elements are diverted to another one. The output of each of the two smaller mergers is the sorted sequence of their respective inputs. Finally, the two small sorted sequences are interleaved and passed through a single stage of comparison, then the output would be the sorted sequence.

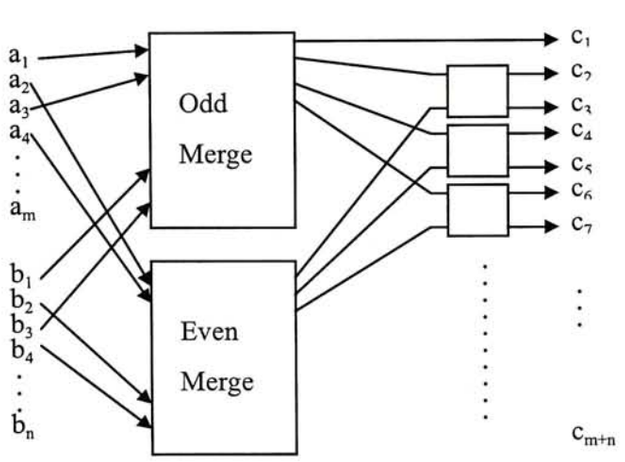


Figure 1.1 Iterative rule for odd-even merging networks

In Figure 1.1, it is not necessary for m equals n, i.e. the two input sequences maybe of different lengths. For the two smaller mergers, they can be constructed by invoking the same iterative rule again. According to the *zero-one principle* [7], the odd-even merging network is able to sort all numbers other than zero and one. An 8×8 odd-even merging network constructed using 2×2 cells are illustrated in Figure 1.2.

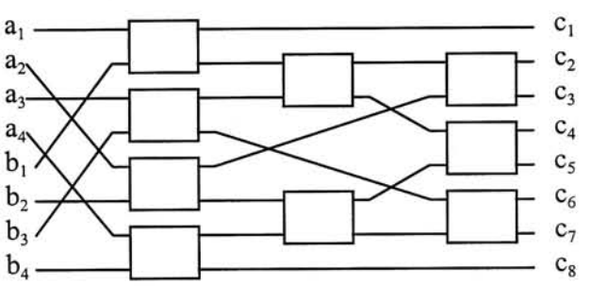


Figure 1.2 8×8 odd-even merging network

1.2.2 Fast Knockout algorithm

In many human sport competitions, there is a knockout tournament to determine the champion among all participants. In a knockout tournament, M participants are divided into $M/2$ pairs, and each pair of participants will have a match to determine who can advance to the next stage. In each stage, half of the participants advance to the next stage while half of them are eliminated. The process continues until a champion of the tournament is determined. Simply speaking, it is the selection of champion through a binary tree of comparisons. In the knockout tournament with total M participants, log_2M stages of matches are required to determine the winner among the M participants.

An M-to-N knockout concentrator [15] is a device with M inputs and M outputs, although we only care about the first N. The inputs to the knockout concentrator, which maybe the activity bits of the incoming signals, are compared so that the N largest among the M inputs are separated out. Although we are talking about sorting zero and one in the knockout, as well as the fast knockout sorting algorithm in latter part, by *zero-one principle* [7], the two algorithms are able to sort all numbers. In the knockout concentrator, the N winners are selected one by one, each through a knockout tournament such that losers of the kth tournament enter the tournament competing for the $(k+1)$st place.

The fast knockout algorithm [10] is an improved version of the knockout algorithm, in terms of stages required to separate the N winners out of the M inputs. This improvement is gained by making use of the transitive law on the ordering of numbers. In the fast knockout algorithm, many small and partially ordered sets are compared and merged. Finally, a partially ordered set containing all the competing numbers is formed. Suppose we have two disjoint and partially ordered sets of numbers. In one of the two sets, there is a number X such that it is smaller than x numbers in its own set. In another set, there is a number Y such that it is smaller than y numbers in its own set. Since the two sets are partially ordered, the loser of the comparison between X and Y is sure to be smaller than $(x+y+1)$ numbers in the merged set. Hence, the loser compete for the $(x+y+2)$nd place in the next round of comparison.

It is defined in [10] that an element of set S, where certain information about the ordering among the number in the set is known, is called an *i-element* if it is known to be smaller than at least i other elements. In addition, the set S is said to be in the state $\{x_0, x_1, \ldots, x_{n-1}\}$ if it can be partitioned into a disjoint union $S_0 \cup S_1 \cup \ldots \cup S_n \cup \ldots$ such that S_i consists of x_i i-elements of S for all i, for a fixed n. Furthermore, the disjoint union of z sets in the state $\{x_0, x_1, \ldots, x_{n-1}\}$ is represented as $z \times \{x_0, x_1, \ldots, x_{n-1}\}$.

In human knockout tournament, two competitors have a match to determine who advance to the next stage. In the fast knockout concentrator, two numbers are feed into the sorting cell to determine which one is the winner while the other one is loser, and they will be competing for different places in the next stage. At the start of the concentration process, there is no information on the ordering of the numbers within the set and it is represented as

$$M \times \{1, 0, 0, \ldots, 0\}.$$

Two disjoint sets of the same state are to be merged into one set. Each i-element is compared with an i-element in another set. According to the transitive law on the ordering of the numbers, the loser in the comparison becomes a $(2i+1)$-element in the merged set. And the stage change is as follow,

$$2z \times \{x_0, x_1, \ldots, x_{2i}, x_{2i+1}, \ldots\}$$

$$\rightarrow z \times \{x_0, x_1+x_0, \ldots, x_{2i}, x_i+x_{2i+1}, \ldots\},$$

and this is called the *fast knockout* stage.

Suppose M equals 2^m, then m Fast Knockout stages are needed to merge M

disjoint sets into one set. Once all the numbers are in one set, then the *conventional knockout* stages are needed to transform the set into $\{1, 1, \ldots, 1\}$. The conventional knockout stage is as follow:

$$\{x_0, x_1, \ldots, x_{n-1}\}$$

$$\rightarrow \left\{ \left\lceil \frac{x_0}{2} \right\rceil, \left\lfloor \frac{x_0}{2} \right\rfloor + \left\lceil \frac{x_1}{2} \right\rceil, \ldots, \left\lfloor \frac{x_{i-1}}{2} \right\rfloor + \left\lceil \frac{x_i}{2} \right\rceil, \ldots, \left\lfloor \frac{x_{n-2}}{2} \right\rfloor + \left\lceil \frac{x_{n-1}}{2} \right\rceil \right\}$$

In another word, we start the concentration with $M \times \{1, 0, 0, \ldots, 0\}$. After $log_2 M$ fast knockout stages, they are merged into one ordered set. And then invoke the conventional knockout stage until the set becomes $\{1, 1, 1, \ldots, 1\}$ and concentration is done.

An 8-to-4 fast knockout concentrator is shown in Figure 1.3. Also, the stage evolution of an 8-to-4 fast knockout concentrator is shown in Figure 1.4. Here we adopt the notation in [7] that a vertical arrow represents a comparison between two elements and the larger one will go to the tail of the arrow. Using 2×2 sorting cell can perform the comparison.

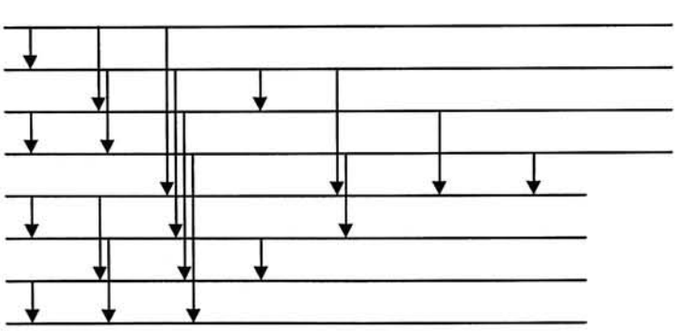


Figure 1.3 An 8-to-4 fast knockout concentrator

Stage	State
	8×{1, 0, 0, 0}
1	4×{1, 1, 0, 0}
2	2×{1, 2, 0, 1}
3	1×{1, 3, 0, 3}
4	1×{1, 2, 1, 2}
5	1×{1, 1, 2, 1}
6	1×{1, 1, 1, 2}
7	1×{1, 1, 1, 1}

Figure 1.4 State changes of the 8-to-4 fast knockout

An *M*-to-*N* fast knockout concentrator is not qualified to be a 0-1 sorter, and this will be illustrated in the following example. Consider an 8-to-3 fast knockout concentrator that is shown in Figure 1.5. Also shown in Figure 1.5 are five 1's presented at the input side.

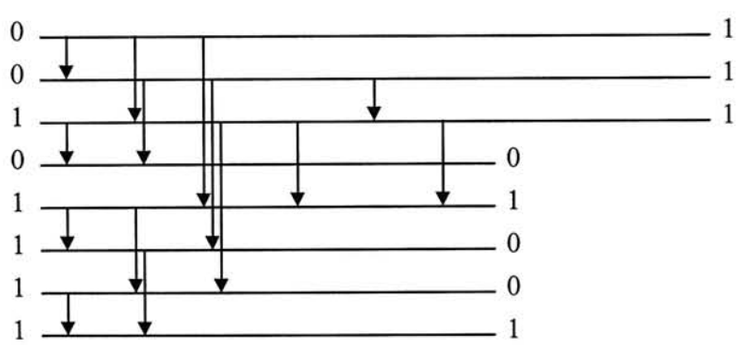


Figure 1.5 An 8-to-3 fast knockout concentrator with five 1's at the input side

At the output side, although the first three outputs are 1's, there is no guarantee that the fourth and fifth outputs are also 1's. It can be seen that for the last two 1's, they appear at the fifth and the eighth outputs, instead of the fourth and the

fifth. With this example, it can be seen that while fast knockout concentrator is an M-to-N concentrator, it is not qualified to be a 0-1 sorter.

1.2.3 Reverse Banyan network preceded by a running sum adder

It was proposed in [3 and 14] that a *reverse banyan network* preceded by a *running sum adder* is capable to be a 0-1 sorter. Before talking about the cascade of reverse banyan network and running sum adder, the mirror image of reverse banyan network, a banyan network, will be introduced and its non-blocking conditions will be presented and discussed. An 8×8 banyan network is illustrated in Figure 1.6.

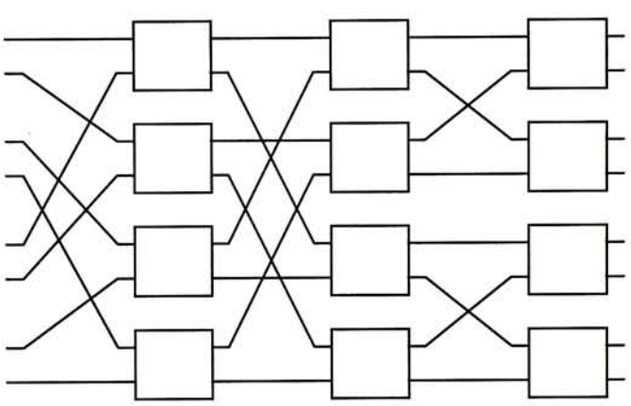


Figure 1.6 An 8×8 banyan network

Banyan network is a unique-path routing network, i.e. from a particular input to a particular output, there is one and only one path connecting them. In addition, a $N\times N$ banyan network consists of log_2N stages of switching cells, and there are $N/2$ 2×2 switching cells in each stage. For a $N\times N$ banyan network, there are $(N/2)log_2N$ 2×2 switching cells in total.

Since Banyan-type networks are unique-path routing networks, it is

intuitive to think that they incur a very high chance of blocking even though the destination addresses of the arriving packets are unique. However, if the arriving packets satisfy certain conditions, it is guaranteed that the packets can be routed without any internal conflict within the Banyan-type network. The banyan network is non-blocking if the active inputs $x_1, x_2, \ldots, x_m$ ($x_j > x_i$ if $j > i$) and their corresponding output addresses y_k satisfy the following:

1. $y_1 < y_2 < \ldots < y_m$ *or* $y_1 > y_2 > \ldots > y_m$

2. *If* $x_i < w < x_j$, *then w should also be an active input.*

In another words, for the banyan network to be non-blocking, the corresponding destination addresses of the inputs have to be either monotonically increasing or decreasing. In addition, the incoming packets should be compact or concentrated, i.e. leaving no idle input between any two active inputs.

Conforming to the above two conditions in a banyan network, paths connecting a set of concentrated inputs and their corresponding outputs, which are either monotonically increasing or decreasing, must can be established. With this fact, a reverse banyan network is capable of being a 0-1 sorter, and the rationale is as follow: A reverse banyan network is no more than a mirror image of the banyan network. The inputs to the banyan network become the outputs of the reverse banyan network, and outputs of the banyan network become the inputs to the reverse banyan network. With this exchange of roles between input and output, the non-blocking conditions for banyan network serves as a supporting argument that a reverse banyan network can served as a 0-1 sorter.

Analogous to the non-blocking conditions in the banyan network, any set of inputs in the reverse banyan network must can be connected to the set of consecutively-located outputs, provided that the output addresses of the corresponding inputs are either monotonically increasing or decreasing [6]. Shown in Figure 1.7 is the scenario that active inputs are routed to the concentrated output ports in the reverse banyan network.

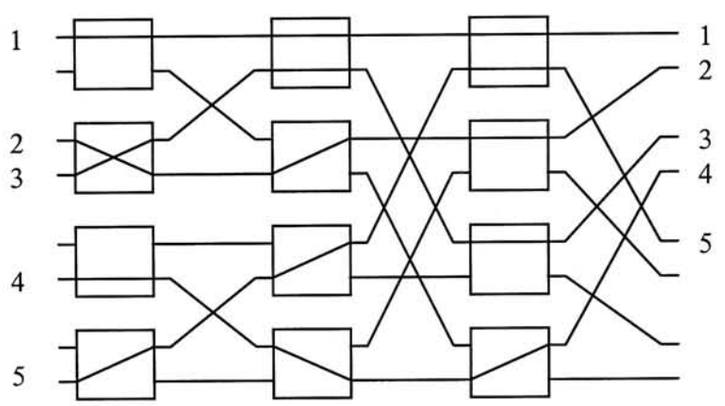


Figure 1.7 An 8×8 reverse banyan network served as a 0-1 sorter

In the above, it can be seen that a reverse banyan network possesses the capability for being 0-1 sorter. However, the routing of signals in the reverse banyan network remains a problem to be solved. By adding a running sum adder, the function for the reverse banyan network to be a 0-1 sorter is completed. Let $N = 2^n$. Define the *running sum* of an $N \times N$ switching device as a sequence of A_0, A_1, …, A_{N-1} such that each A_k counts the number of active input ports between addressed 0 and k-1. A running sum adder calculates each A_k. If the calculated value is binary $(b_1 \ldots b_n)$, then the n-bit routing tag $b_n b_{n-1} \ldots b_1$ is attached in front of the packet that enters input port k of the reverse banyan network for self-

routing. A running sum adder is depicted in Figure 1.8. It can be seen that a running sum adder consists of only simple adders.

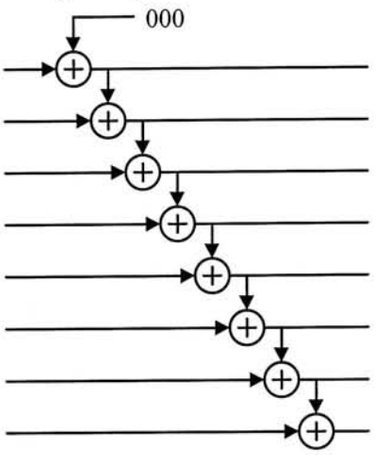


Figure 1.8 An 8-input running sum adder

Suppose the activity bit '1' represents an active input while '0' represents an idle one. Since signals are to be concentrated to consecutively located outputs in the reverse banyan network, the destination address for two successive active inputs differ by one only. Running sum adder computes a set of new dummy destination tags for the active inputs and the individual bits of the tags are used to control the state of the 2×2 switching cells in the reverse banyan network. For example, in Figure 1.5, dummy destination tag '000' will be attached to the first active inputs. In addition, the result of the addition between '000' and '1' (the activity bit) will be '001', and it will in turn be the dummy destination tag for the next active input. The set of dummy destination tags generated by the running sum adder will be consecutive, so that the active inputs will be routed in a non-blocking way in the reverse banyan network.

In Figure 1.8, eight adders are used in the 8-input running sum adder. However, there is another construction for the running sum adder with the same

dimension. This implementation will be discussed in section 1.2.5 in more detail.

1.2.4 Recursive construction using iterative cells

In [13], a recursive construction for 0-1 sorter is proposed. Although the author of [13] regards it as a concentrator, it is actually a 0-1 sorter according to definition 1.1.2. Since the process of concentration makes use of the activity bits of the signals, which are either '0' or '1', the device is no more than separating the signals with activity bits '1' from those with activity bits '0'.

The recursive construction is depicted in Figure 1.9.

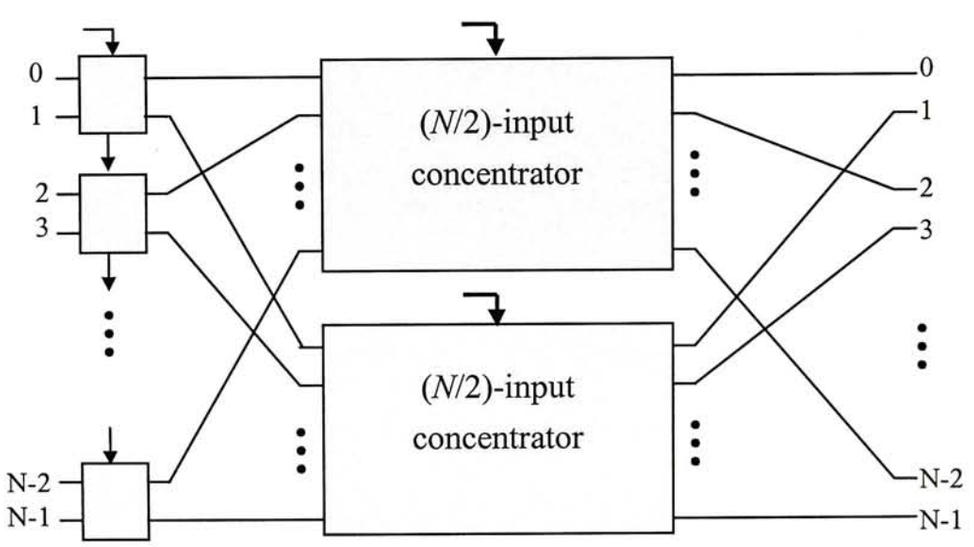


Figure 1.9 Recursive construction of N-input concentrator

It can be seen in Figure 1.9 that the recursive construction consists of two stages. First of all, the first stage is a set of 2×2 switching elements. The states of the front-end 2×2 switching elements are set so that one-half of the active inputs are routed to the upper (N/2)-concentrator. On the other hand, the remaining of the

active inputs are routed to the lower ($N/2$)-concentrator. In the case that odd number of active inputs are presented at the input side, the upper ($N/2$)-concentrator will have one more active input than the lower one. For the two ($N/2$)-input concentrators in the second stage of an N-input concentrator, they can further be constructed using the same recursive algorithm. Finally, the outputs from the two (N/2)-input concentrators are interleaved to obtain the final concentrator output.

As mentioned, the first stage of 2×2 switching elements is needed to direct the active inputs to the upper and lower ($N/2$)-input concentrators alternatively. In order to achieve this, a special 2×2 switching element, called *iterative cell*, is employed. Unlike standard 2×2 switching elements such as those used in Figure 1.6 and Figure 1.7, each 2×2 element here is equipped with an extra non-data input for control signaling. The control signal entering a 2×2 element is the *running parity* (RP), i.e. the running sum modulo 2. The 2×2 element uses this RP in its switching control and, at the same time, integrates the activity bits of its two data inputs into the RP before passing the RP down to the rest 2×2 elements at the same stage. The logic circuitry is shown in Figure 1.10.

Similar to the case in 1.2.3 that the activity bit '1' represents an active input while '0' represents an idle one. Let the upper and lower outputs of the 2×2 element be labeled as 0 and 1, respectively. If exactly one of the two inputs is active and the received RP is x, x is either 0 or 1, then the active input is

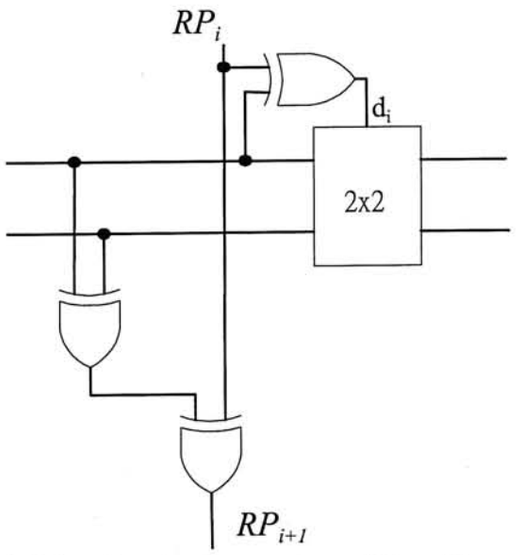


Figure 1.10 Logical circuitry of iterative cell

connected to output x. When both inputs are active, a logic 1 of x sets the 2×2 switching element to cross state while a logic 0 sets it to bar state. Actually, the state is controlled by signal d_i in Figure 1.10. It is bar state if d_i is '1', or cross state if d_i is '0'. The eight possible input patterns, and the corresponding state of the 2×2 elements, are shown in Figure 1.11. It should be noted that the eight possible input patterns and the corresponding state of the 2×2 element can be realized by the circuitry in Figure 1.10.

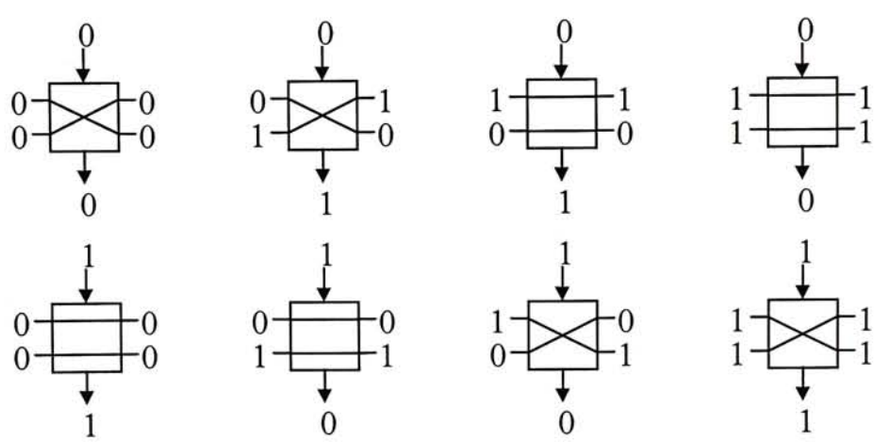


Figure 1.11 The eight possible patterns of an iterative cell

According to the recursive construction, the two (N/2)-input concentrators

are further constructed using the same algorithm, giving rise to four ($N/4$)-input concentrators, and so on. Unfolding the recursion, the concentrator is built solely with iterative cells. In Figure 1.12, an 8-input concentrator is illustrated.

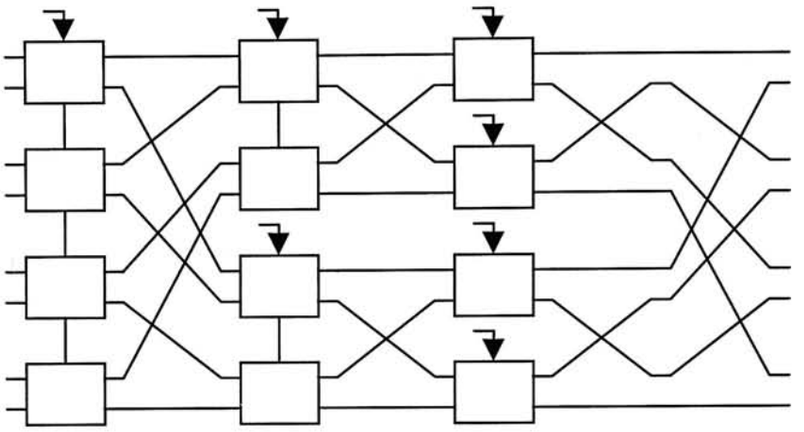


Figure 1.12 A recursively constructed 8-input concentrator

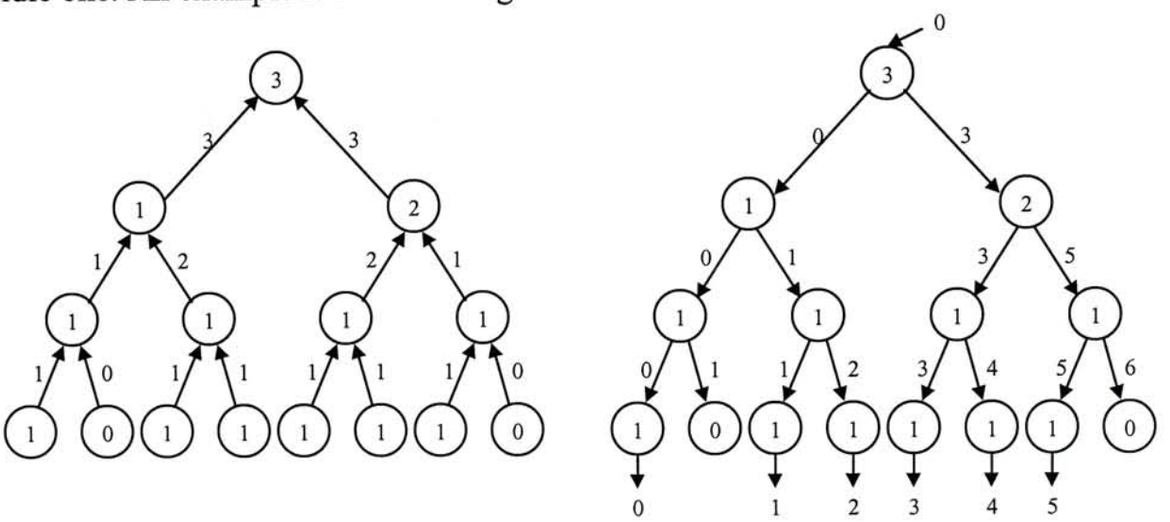
In each of the recursion, there is a stage of front-end iterative cells. For this stage of front-end iterative cells, the RP has to be initialized and it is passed down throughout all the iterative cells within the same stage. In addition, for the second recursion, there are two (N/2)-input concentrators, hence two sets of front-end iterative cells are formed, and they are within the same stage. For the network to be a 0-1 sorter, all the RP's have to be initialized as '0', and the active inputs (with activity bit '1') will then concentrated at the output side, start stacking at the top output. For the final exchange of the network, it is actually the composition of two shuffle-type exchanges. In the 8-input concentrator, it is the composition of two 4×4 shuffle exchanges, which are stacked in parallel, with a 8×8 shuffle exchange.

1.2.5 Comparison of 0-1 sorters

A running sum adder is used, together with a reverse banyan network (section 1.2.3), to form a 0-1 sorter. The running sum adder is used to generate the routing tags, which are to be used for the routing in the following reverse banyan network. An implementation of running sum adder is shown in Figure 1.8. In that implementation, it can be seen that the running sum is transmitted serially from top to bottom. As the routing tags are generated and transmitted one by one, from top to bottom, serially, there will be an eight-gate-time lag between the generation of routing tags for the first input and the eighth input. It can be said that the adders are arranged in eight stages, as they are operating in serial.

Beside the serially constructed running sum adder as shown in Figure 1.8, there is another construction for running sum adder. This implementation is based on a binary tree [9]. Each leaf of the tree corresponds to an input to the running sum adder. The calculation of running sum is in two phases. In phase 1, each leaf will send an '1' upward if the corresponding input is active. At the same time, each internal node receives a number L from the left-son and a number R from the right-son. The internal node will send the sum of L and R to its predecessor, while keeping the value of L in its own node. In phase 2, each internal node will receive a value from its predecessor. This value will be sent to the left son of the internal node. On the other hand, the sum of value from predecessor and the value kept in that internal node in phase 1 will be sent to the right son. The operation for the root is the same as those internal nodes, except

the value from its predecessor is fixed at zero, as the root has no predecessor. For the leaves, they will receive a value from its predecessor, and this value will in turn be the routing tag for the corresponding input, provide that the input is active. On the other hand, the value received will be neglected if the input is an idle one. An example is shown in Figure 1.13.



Phase 1 Phase 2

Figure 1.13 Two phases for generating the running sum in the binary tree

The two-phase algorithm, associated with the binary tree, for generating the running sum can be realized by using adders. A running sum adder based on the two-phase algorithm is illustrated in Figure 1.14.

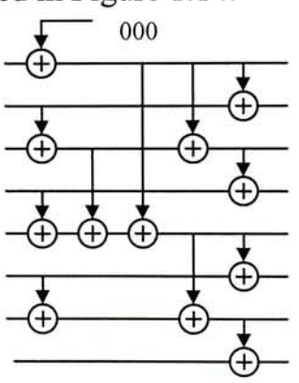


Figure 1.14 An 8-input running sum adder associated with binary tree

For the 8-input running sum adder in Figure 1.14, there are five stages of adders.

In general, an N-input running sum adder would require $(2log_2N\text{-}1)$ stages of adders. Comparing to the serially connected adders, the above implementation would shorten the time lag between the generation of the first and last routing tags. This save in time would be significant when the dimension of the running sum adder is large. This is crucial for high-speed data network, since synchronization is needed before entering the reverse banyan network. If the time lag between the generation of the first and last routing tags is large, the synchronization would be hard to maintain. It is not shown in Figure 1.14 that there are some elements for delay so that synchronization is kept in each stage of the running sum adder.

While the running sum adder generate the routing tags for all inputs before sending them to the reverse banyan network, the recursively-constructed 0-1 sorter (section 1.2.4) generates the routing tags within the network, each bit per stage, by mean of running parity. Consider the scenario in Figure 1.15, in which a recursively-constructed 0-1 sorter with three active inputs is shown.

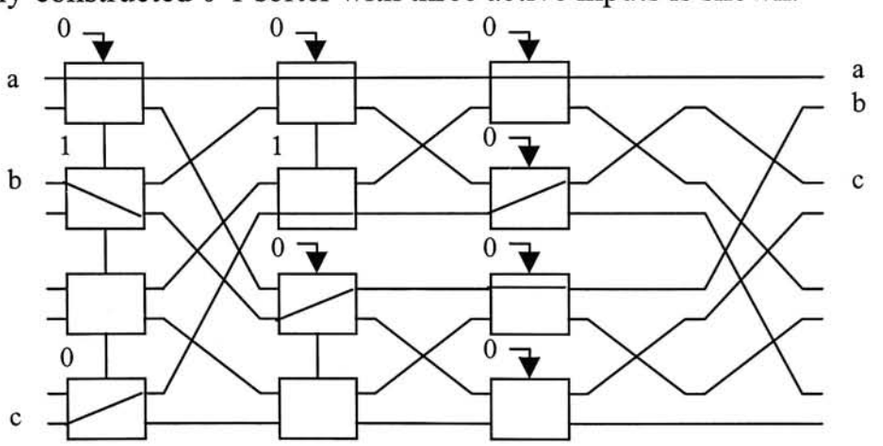


Figure 1.15 A recursively constructed 8-input concentrator with three active inputs

Signal *a*, *b* and *c* are the three active signals, which are presented at input ports 0, 2 and 7, respectively. After passing through the three stages of iterative cells, they will be concentrated at the top three output ports. In another words, the destination addresses of signal *a*, *b* and *c* will be 000, 001 and 010 respectively. Consider signal *b* in particular. The running parities to those iterative cells which signal *b* goes through are 1, 0 and 0, from the first to the third stage. Actually, this is the reverse of the destination address for signal *b*. It is similar for *a*, *c*, and all other signals which have to be routed through the 0-1 sorter. The running parities can be viewed as the routing tag within the 0-1 sorter, and the use of this routing tag started at the least significant bit at the first stage to the most significant bit at the final stage. Comparing to the running sum adder, running parities are generated within the routing network, and there is no need for an extra device for the running sum adder. In addition, the routing tags in the recursively-constructed 0-1 sorter are generated and consumed at the same time. On the contrary, the routing tags are generated in the running sum adder and they are then used in the reverse banyan network. The performance of the running-parity approach would degrade if the dimension of the 0-1 sorter is large as synchronization is needed to be maintained within the same stage of cells. For the complexity, the running-parity approach saves the space for the extra running sum adder as mentioned. However, the wiring for the running parity within the same stage would complicate the layout and complexity is larger than a baseline network of normal 2×2 switching cells.

Chapter 2

Compressor based on baseline-swap network

In those 0-1 sorters discussed in Chapter 1, the core elements of them are the 2×2 switching cells (iterative cell can be viewed as a variation of 2×2 switching cell). In most cases, several 2×2 switching cells are aligned in a stage, and there are a number of stages within the 0-1 sorter. Successive stages of 2×2 cells are connected by *exchange*. In this chapter, the class of *bit permutation induced exchanges* [9] is presented. The bit permutation induced exchange is important in proving that the topology of a recursively-constructed 0-1 sorter is no more than a baseline network appended with a special exchange. In addition, the definition of a switch called *compressor*, which can be functioned as *cyclic 0-1 sorter* under proper switching control, will be introduced. In particular, the recursively-constructed 0-1 sorter can be modified to be a cyclic 0-1 sorter by initializing the running parities to a particular set of values. The explicit formula for this initialization will be given in this chapter.

2.1 Bit permutation induced exchange

In the previous chapter, four construction algorithms for concentrators and 0-1 sorters are revisited. It is common in all of them that the basic building blocks are the 2×2 switching cells. In addition, for the baseline network preceded by a running sum adder and the recursively-constructed 0-1 sorter, the

2×2 cells are aligned in stage. For example, an 2^k-input 0-1 sorter based on the baseline network consists k stages of 2×2 cells, and there are 2^{k-1} 2×2 cells in each stage. That's also exactly the case for the recursively-constructed 0-1 sorter. In this situation, a network is different from another by having a different connection pattern between the 2×2 cells in successive stages. The connection pattern between two stages of 2×2 cells is called *exchange*. There are many possible exchanges. However, in most cases, the exchanges that are commonly seen (e.g. banyan exchange and shuffle exchange) are in the class of bit permutation induced exchange [9].

Before talking about the bit permutation induced exchange, the permutation and representation of it will be covered first. A permutation on integers from 1 to n means a one-to-one function from the set of these n integers into itself. The most succinct notation for permutations is the cycle representation in group theory: For example, the cycle (125) represents the permutation such that $1\rightarrow2$, $2\rightarrow5$, $5\rightarrow1$ and $k\rightarrow k$ for all numbers other than 1,2 and 5. When two or more permutations are applied successively, the result is the same as applying the multiplication, from left to right, of the permutations. Hence, for example, the permutation (125)(234) is the same as (13425).

For a 2^k-input network, there are exactly 2^k input and output ports at every stage of 2×2 cells. Each of the input and output ports can be represented by a k-bit number in their stages. The interstage exchange is the connection between the output ports of one stage to the input ports of the following stage. Actually,

this exchange is a one-to-one mapping induced by the permutation among the k-bit numbers. The mapping would be represented by X_σ, where σ is the permutation among number from 1 to k.

Definition 2.1.1 A $2^k \times 2^k$ exchange X is a one-to-one mapping among all k-bit numbers.

Definition 2.1.2 The *order* of a non-identity permutation σ means the smallest number m such that $\sigma(m) \neq m$.

With this notation on exchanges, the connection pattern between stages can be expressed in a more precise way. In addition, the result of the combination of several exchanges can be worked out much more easily, since the combination is no more than the multiplication of the permutations. The definitions of some commonly seen exchanges are given below.

Definition 2.1.3 Fixed an integer n. The exchange $X_{(n\ n\text{-}1\ \ldots\ d)}$ will be called the $2^n \times 2^n$ *shuffle exchange of order d* for $1 \leq d < n$ and denoted as $SHUF^{(n)}_d$. It can be simplified as $SHUF_d$ when there is no ambiguity. In particular, the $2^n \times 2^n$ shuffle exchange of order 1 is simply the $2^n \times 2^n$ shuffle exchange and denoted as $SHUF^{(n)}$ or SHUF.

Figure 2.1 shows a $SHUF^{(3)}$ and the permutation of a $SHUF^{(3)}$ is (321). For each output port, they are connected to the input port of the following stage with the address such that it is the left shift of one bit of the originating output port address.

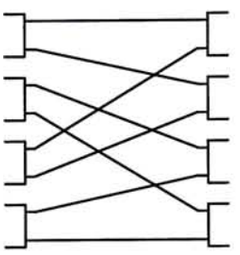


Figure 2.1 A $SHUF^{(3)}$ exchange

Definition 2.1.4 Fix an integer n. The exchange $X_{(n\ d)}$ will be called the $2^n \times 2^n$ *banyan exchange of order d* and denoted as $BANY^{(n)}_d$, or simply $BANY_d$. In particular, the $2^n \times 2^n$ banyan exchange of order 1 is simply called the $2^n \times 2^n$ banyan exchange and denoted as $BANY^{(n)}$ or simply BANY.

Figure 2.2 shows a $BANY^{(3)}$ and the permutation of it is (31). Each output port is connected to an input port of the following stage, where the input port address is obtained by exchanging the first and the third bits of the originating output port address.

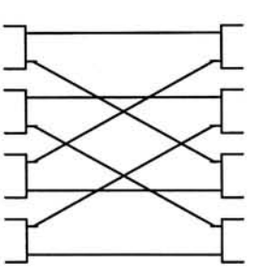


Figure 2.2 A $BANY^{(3)}$ exchange

Definition 2.1.5 Fix an integer n. The exchange $X_{(d\ n)(d+1\ n-1)\ldots\left(\left\lfloor\frac{k-d+1}{2}\right\rfloor+d-1\right.}$

$k-\left\lfloor\frac{k-d+1}{2}\right\rfloor+1)$ will be called the *$2^n \times 2^n$ swap exchange of order d* and denoted as $\text{SWAP}^{(n)}_d$, or simply SWAP_d. In particular, the $2^n \times 2^n$ swap exchange of order 1 is simply called the $2^n \times 2^n$ swap exchange and denoted as $\text{SWAP}^{(n)}$ or simply SWAP.

An 8×8 swap exchange is the same as an 8×8 banyan exchange. Figure 2.3 shows a 16×16 swap exchange.

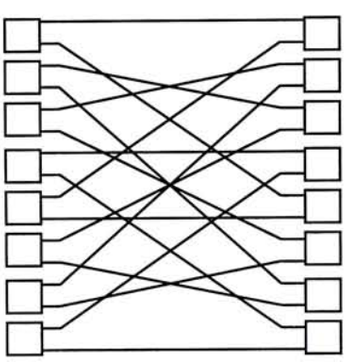


Figure 2.3 A $\text{SWAP}^{(4)}$ exchange

Swap exchange is not as popular as the banyan and shuffle exchanges, as it doesn't draw much attention as the other two in literatures. However, swap exchange will be part of the focus in the later part of the thesis as it is the exchange appended to the baseline network after unfolding the recursively-constructed 0-1 sorter.

2.2 Compressor

For the four 0-1 sorters and concentrators introduced in Chapter 1, they are

all devices with the same number of inputs and outputs. The definitions of concentrator and 0-1 sorter are recited as follow.

By an M-to-N concentrator, $M \geq N$, it means an M-input-M-output device for routing single-bit signals such that the upper N outputs are all greater than or equal to the lower M-N outputs.

The single-bit signal may stand for the activity of an input port. A binary data stream trails the activity bit and it is irrelevant to the switching inside the concentrator.

An $M \times M$ device is a 0-1 sorter if it is an M-to-N concentrator for *all* $N \leq M$. An $M \times M$ binary sorter will also be called an M-input concentrator.

In all those previously seen 0-1 sorters, they compress active inputs to consecutive outputs starting at fixed output address, normally the first output port. On the other hand, there is a switching device called *compressor*. Instead of start stacking active inputs at a fixed output address, compressor concentrates active inputs to circularly consecutive outputs starting at any given output address. The compressor with proper switching control will be called *cyclic 0-1 sorter*.

Definition 2.2.1 A device is called a compressor if it can accommodate every combination of concurrent connections subjected to the following two conditions on active input/output address:

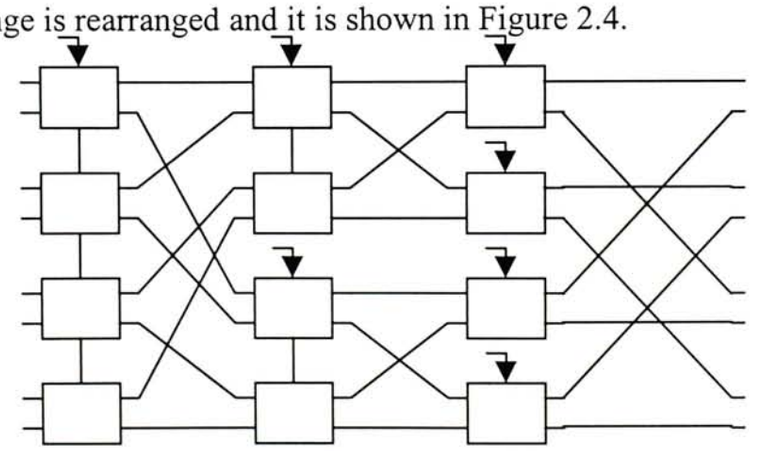
(a) The active output addresses are circularly consecutive;

(b) The mapping from the active input addresses to their corresponding output addresses is monotonically increasing or decreasing.

Both the strings "00011110" and "11000011" are circular consecutive while "00110110" is not. According to this definition, the 0-1 sorters discussed in Chapter 1 are just a special case of compressor with particular control. With this definition, the cyclic 0-1 sorter constructed can be used in various aspects in switching. For example, it can be used as a *distributor* [5] in buffer allocation so that traffic among the buffers can be evenly distributed.

2.3 Introduction to Baseline-swap network

The 8-input recursively-constructed 0-1 sorter is shown in Figure 1.12. The final exchange is rearranged and it is shown in Figure 2.4.



For the 8-input 0-1 sorter in Figure 2.4, it is actually a baseline network
Figure 2.4 Recursively constructed 8-input 0-1 sorter

appended with a banyan exchange after the rearrangement. It is not surprising that the first half of the network is a baseline network. Since the construction of the 0-1 sorter is a recursive one and it is similar to the recursive 2-stage construction, which generates a baseline network, except that the recursive construction of 0-1 sorter appends a shuffle exchange at the end in each step of the recursion.

Theorem 2.3.1 When two $2^{k-1}\times 2^{k-1}$ swap exchanges are vertically stacked together, their concatenation with a $2^k\times 2^k$ shuffle exchange yields the $2^k\times 2^k$ swap exchange.

Proof

For two $2^{k-1}\times 2^{k-1}$ swap exchanges that are vertically stacked together, the whole exchange is $\mathrm{SWAP}^{(k)}_2$. The associated permutation is $(2\ k)(3\ k\text{-}1)\ldots(\left\lfloor\frac{k-1}{2}\right\rfloor+1\ k-\left\lfloor\frac{k-1}{2}\right\rfloor+1)$. The associated permutation of the $\mathrm{SHUF}^{(k)}$ is $(k\ k\text{-}1\ \ldots\ 1)$. The associated exchange of the concatenation is the product of the previous two permutations, which is $(1\ k)(2\ k\text{-}1)\ldots(\left\lfloor\frac{k}{2}\right\rfloor\ \ k-\left\lfloor\frac{k}{2}\right\rfloor+1)$. This permutation would induce the exchange $\mathrm{SWAP}^{(k)}$. ■

Theorem 2.3.2 Unfolding the recursive construction of the 2^k-input concentrator in section 1.2.4 would yield a $2^k\times 2^k$ baseline network appended with a $2^k\times 2^k$

swap exchange.

Proof

The baseline network is trivial and it is similar to the result of the recursive 2-stage construction. The final exchange is just a functional composition of $SWAP^{(k)}_{k-1}$, $SHUF^{(k)}_{k-2}$, ... $SHUF^{(k)}_{2}$, $SHUF^{(k)}$. By Theorem 2.3.1, the results follow. ■

The composition of a $SWAP^{(4)}_{2}$ and a $SHUF^{(4)}$ is shown in Figure 2.5.

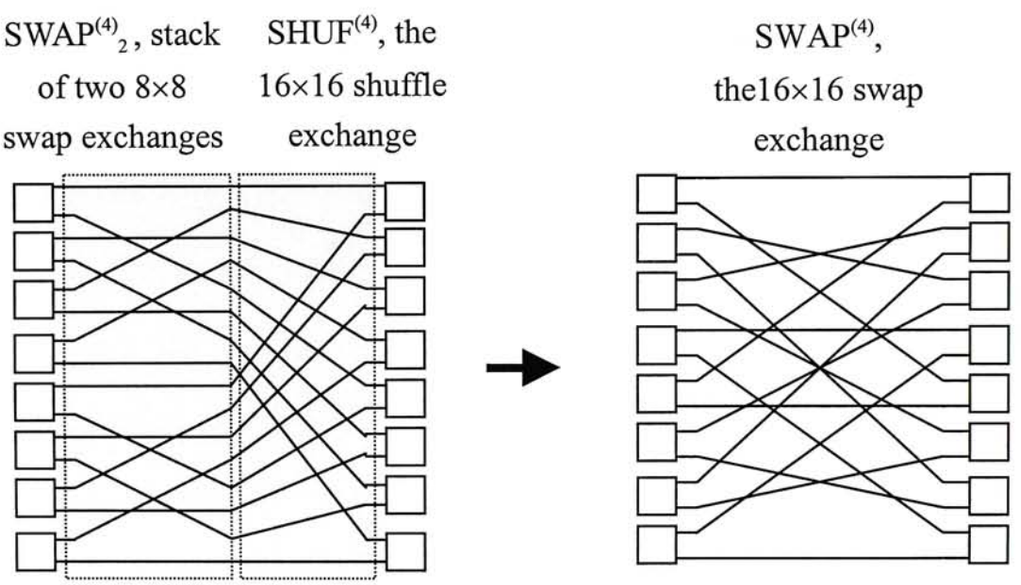


Figure 2.5 $SWAP^{(4)}_{2} \times SHUF^{(4)} = SWAP^{(4)}$

It should be noted that the 0-1 sorter in Figure 2.4 is a baseline network appended with a banyan exchange. However, a $BANY^{(3)}$ is identical to a $SWAP^{(3)}$. The topology of a baseline network appended with a swap exchange at the end will be referred as baseline-swap network in the remaining of the thesis.

2.4 New algorithm for running parity initialization

In the $2^k \times 2^k$ baseline-swap network design, the 2^k-1 values of the running parity RP_0, which is fed to the first iterative cell at each front-end stage in the recursive construction, are hard-wired to zero. As a result, all the active signals are concentrated at the upper output ports from top to bottom. In both Figures 1.8 and 1.14, it can be seen that an initial address '000' is fed to the running sum adder. Hence the concentrated outputs start stacking at the top output port. Actually, the 0-1 sorter constructed from a running sum adder with a reverse banyan network is able to send active inputs to circularly consecutive outputs starting at any given output address, once we offer an initial value, instead of '000', for the computation of running sum. In section 1.2.5, the running sum adder and running parity are compared, and they are of similar function. Hence, it is expected that the baseline-swap network is able to be a cyclic 0-1 sorter by setting the RP_0 to appropriate values.

The discussion of cyclic 0-1 sorter in this chapter will be based on $2^n \times 2^n$ baseline-swap network constructed with iterative cells. And it will be shown, in Chapter 3, that the baseline network appended with a swap exchange is just one member of a large family of compressor. The compression is done by suitably controlling the states of 2×2 switching cells within the cyclic 0-1 sorter. By the construction of cyclic 0-1 sorter based on iterative cells, a recursive algorithm [8] was derived to initialize the running parities in order to function as a distributor

and this is briefly discussed as follow. Suppose there are m active inputs to be concentrated starting at the output port d. If d is an even number, RP_0 of the first stage is set to zero, otherwise, one is set. After RP_0 of the first stage is determined, the outputs are disinterleaved to be two output patterns of the smaller 0-1 sorters. By looking at the starting addresses of the two output patterns, the RP_0's of the respective 0-1 sorters can be determined in the similar fashion to the RP_0 of the first stage. By invoking the process recursively, all the RP_0's can be obtained.

Notations: Considering the N-input ($N=2^n$) compressor, $RP_{i,j}$ is used to represent the initial value of RP's at stage i, $i = 0, 1, \ldots, n-1$, and the j^{th} one, $j = 0, 1, \ldots, 2^i-1$. In the algorithm, j will be expressed in binary format for manipulation and it is represented as $j_{i-1}j_{i-2}\ldots j_0$ with j_{i-1} as the *most significant bit* (MSB). Symbols "$\vee$", "$\wedge$" and "$\oplus$" are representing logical function OR, AND and XOR, respectively. An 8-input compressor composed with an 8×8 baseline network and an 8×8 SWAP exchange, together with different RP's is shown in Fig. 2.6.

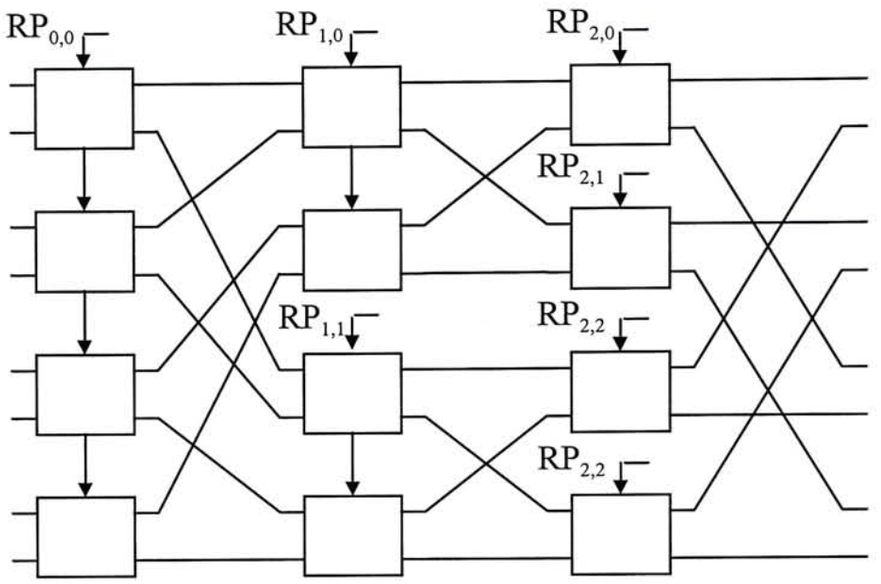


Figure 2.6 8-input compressor together with its RP's

The starting point of the circularly consecutive outputs is given by D_n, $0 \le D_n < N$, which may be represented in n-bit binary form as $d_{n-1} \ldots d_1 d_0$ with d_0 being the *least significant bit* (LSB) and d_{n-1} being the MSB. If a k-bit binary number is represented in the form of a k-dimension binary vector with the MSB at the top, then

$$D_n = \begin{pmatrix} d_{n-1} \\ \vdots \\ d_3 \\ d_2 \\ d_1 \\ d_0 \end{pmatrix}.$$

Lemma 2.4.1

A 2×2 cell is itself a compressor.

Lemma 2.4.1 is clear and the proof is omitted here.

Thus, when $N = 2$, the N-input compressor corresponds to a single iterative cell. The function of RP is to determine which output port the first active input will go. Thus RP is just simply the destination address of the first active input, d_0. When $N>2$, D_n is applied to the N-input compressor for obtaining the initial values of the RP's. By the recursive construction of the network, the given D_n has to be translated to E_1, F_{n-1} and G_{n-1}, respectively. It is shown in Figure 2.7, where E_1=$RP_{0,0}$, F_{n-1} and G_{n-1} are starting points of the respective circularly consecutive outputs of the two (N/2)-input compressors.

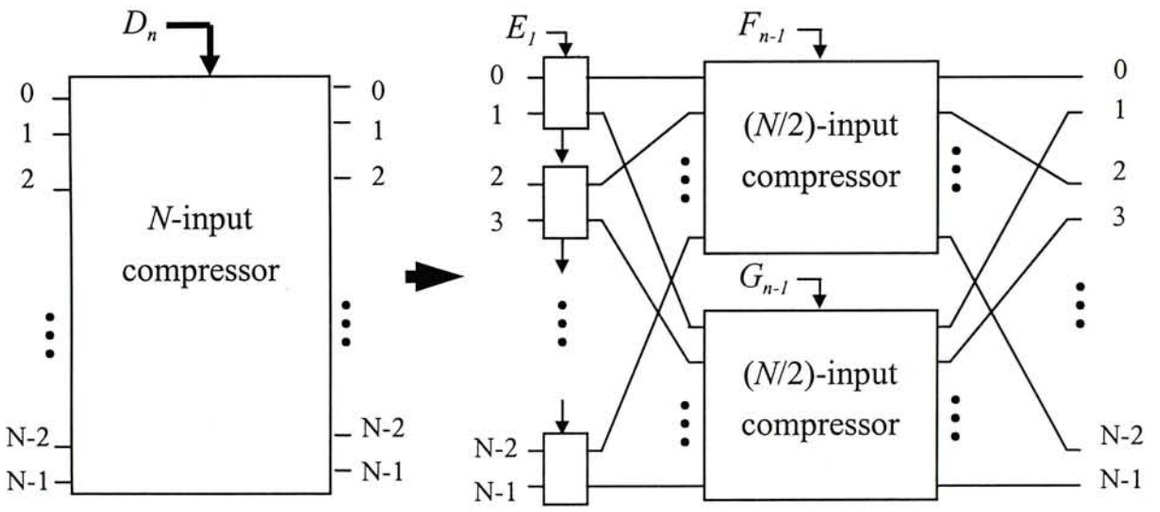


Figure 2.7 Translation of D_n into E_1, F_{n-1} and G_{n-1}

Refer to Figure 2.7, the first active signal (from top to bottom) will be directed to the upper (N/2)-input compressor in the second stage if E_1 is zero, or the lower (N/2)-input compressor if E_1 is one otherwise. Obviously, it can be seen that the upper (N/2)-input compressor is connected with even-indexed output ports 0, 2, ..., N-2 while the lower one is connected with odd-indexed output ports 1, 3, ..., N-1. With this observation, E_1 should be determined by d_0,

the LSB in D_n, whose value (0 or 1) indicates whether D_n is odd or even. After the first stage, the first and second active signals are separated into two independent (N/2)-input compressors, and the two active signals become the first signals to be considered in their respective (N/2)-input compressors. In the second stage, the respective starting points, F_{n-1} and G_{n-1}, of the two (N/2)-input compressors are needed to be determined and they will be shown to be obtained by the remaining bits in D_n, together with the LSB.

It is obvious that one of the F_{n-1} and G_{n-1} must be $d_{n-1}...d_1$, depending on which (N/2)-input compressor received the first active input signal. The rationale is as follow: After the inverse shuffle exchange between the two stages, the destination address of the first active input D_n becomes $d_0d_{n-1}...d_1$. Thus, once the respective (N/2)-input compressor routes the signal to its output port $d_{n-1}...d_1$, the appended shuffle exchange will send the signal to the destination $d_{n-1}...d_1d_0$ (See the permutation induced by $\mathrm{SHUF}^{(n)}$). From the definition of compressor, the second active signal at the input side must be routed to the output address that is by 1 increment of D_n, hence another address is ($d_{n-1}\oplus(d_0\wedge d_1\wedge\ldots d_{n-2})$, …, $d_2\oplus(d_0\wedge d_1)$, $d_1\oplus d_0$, $d_0\oplus 1$). The second LSB is $d_1\oplus d_0$ as the carry from the addition of 1 to the LSB has to be considered, and similar for other more significant bits.

Suppose that $d_0 = 1$, the first active signal is routed to the lower (N/2)-input compressor (G_{n-1} is $d_{n-1}...d_1$), and the addition of 1 to D_n would affect the more significant bits, and hence F_{n-1} should be $(d_{n-1}\oplus(d_0\wedge d_1\wedge\ldots d_{n-2}),\ldots,$

$d_2 \oplus (d_0 \wedge d_1)$, $d_1 \oplus d_0$).

On the other hand, if $d_0 = 0$, then F_{n-1} is $d_{n-1} \ldots d_1$. The LSB of G_{n-1} is $d_1 \oplus d_0 = d_1$, the same as F_{n-1}'s, as the carry has no effect on it. As a result, $F_{n-1} = G_{n-1}$ in this situation. Hence, we have the following statement:

$$\text{If } D_n = \begin{pmatrix} d_{n-1} \\ \vdots \\ d_3 \\ d_2 \\ d_1 \\ d_0 \end{pmatrix}, \text{ then clearly } E_1 = d_0,\ G_{n-1} = \begin{pmatrix} d_{n-1} \\ \vdots \\ d_3 \\ d_2 \\ d_1 \end{pmatrix},$$

$$\text{and } F_{n-1} = \begin{cases} G_{n-1}; & d_0 = 0 \\ G_{n-1} + 1 \quad mod\ 2^{n-1}; & d_0 = 1 \end{cases}.$$

$$\therefore F_{n-1} = d_0 + G_{n-1}\ mod\ 2^{n-1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ d_0 \end{pmatrix} + \begin{pmatrix} d_{n-1} \\ \vdots \\ d_3 \\ d_2 \\ d_1 \end{pmatrix} = \begin{pmatrix} (d_0 \wedge \cdots \wedge d_{n-2}) \oplus d_{n-1} \\ \vdots \\ (d_0 \wedge d_1 \wedge d_2) \oplus d_3 \\ (d_0 \wedge d_1) \oplus d_2 \\ d_0 \oplus d_1 \end{pmatrix}.$$

The translation of D_n into E_1, F_{n-1} and G_{n-1} can be expressed in Figure 2.8.

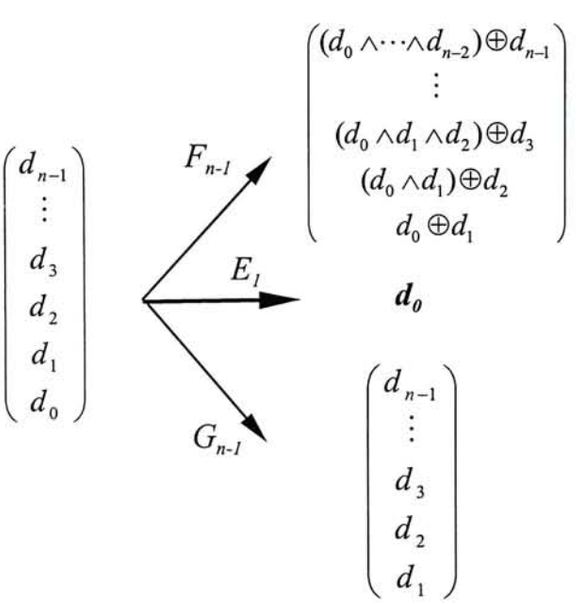


Figure 2.8 Translation from D_n to E_1, F_{n-1} and G_{n-1} in vector expression

Now we unfold the recursive construction of Figure 1.9 into an n-stage network of 2×2 iterative cells. For $1 \leq i \leq n$, there are 2^{i-1} RP's to be initialized in the i^{th} stage. By the recursive application of the formula in Figure 2.8, all the $RP_{i,j}$'s can be determined from the D_n (see Figure 2.9). Therefore, we have the following algorithm. The algorithm is different from that of [8], in which the RP's are recursively obtained.

Algorithm 2.4.2 (Expression for $RP_{i,j}$)

If (i equals zero)

$RP_{0,0} = d_0$;

Else if (j_{i-1} equals zero)

$RP_{i,j} = d_0 \diamondsuit_0 d_1 \diamondsuit_1 \ldots \diamondsuit_{i-2} d_{i-1} \oplus d_i$, *where the operator* $\diamondsuit_k = \begin{cases} \vee; j_{i-k-2} = 0 \\ \wedge; j_{i-k-2} = 1 \end{cases}$

Else

Let l equals to the number of consecutive "1" in the first run from j_{i-1}.

$RP_{i,j} = d_l \diamondsuit_0 d_{l+1} \diamondsuit_1 \ldots \diamondsuit_{i-l-2} d_{i-1} \oplus d_i$, *where* $\diamondsuit_{k-l} = \begin{cases} \vee; j_{i-k-2} = 0 \\ \wedge; j_{i-k-2} = 1 \end{cases}$

* *In the expression, the precedence of operations is always from left to right. Also in the algorithm, $d_0 \diamondsuit_0 d_1 \diamondsuit_1 \ldots \diamondsuit_{i-2} d_{i-1} \oplus d_i$ is the final expression for $RP_{i,j}$ where $\diamondsuit_k$ represents a logical operation, which is either an OR operation or AND operation.*

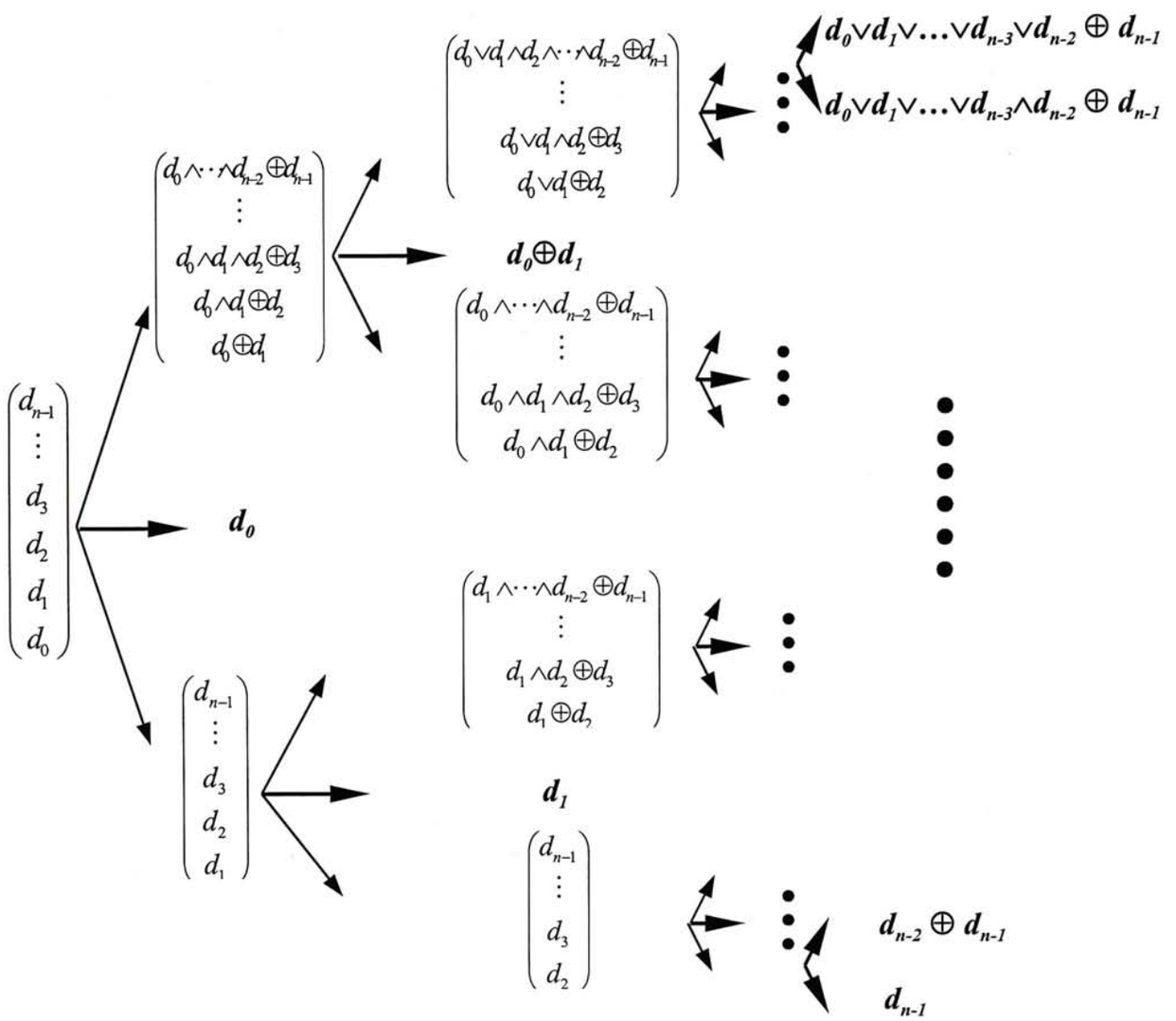


Figure 2.9 Vector expressions for respective concentrators of $N = 2^n$. Notice that the array of initial values of RP's is in boldface expressions.

Proof

The proof is obvious according to the above statements and Figure 2.9. ■

The mechanism of the algorithm is described as follow:

1. If i is equal to zero, $RP_{i,j} = d_0$ and the algorithm stops here.
2. Let l be the number of "1" in the first run of digits in $j_{i-1}j_{i-2}\ldots j_0$, starting from

the MSB. Omit $d_0, d_1, \ldots, d_{l-2}, d_{l-1}$ in the final expression for $RP_{i,j}$.

3. Delete the first l digits (the deleted l digits are all "1" if $l \neq 0$) from $j_{i-1}j_{i-2}\cdots j_0$, then the string belongs to one of the three cases:

   (a) No digit left in j, i.e. j is an all one string:

   $RP_{i,j} = d_i$;

   (b) Only j_0 left and it is equal to zero:

   $RP_{i,j} = d_{i-1} \oplus d_i$;

   (c) A string $j_{i-(l+1)}j_{i-(l+2)}\cdots j_0$ left, and $j_{i-(l+1)}$ equals zero:

      (i) Ignore $j_{i-(l+1)}$.

      (ii) Let $x = l$. Starting with d_x, append "$\vee$" if $j_{i-(x+2)}$ equals zero and "$\wedge$" if $j_{i-(x+2)}$ equals one. Delete the digit having been considered in j. Append d_{x+1}. Increment x by 1. Repeat Step (ii) until j is exhausted.

      (iii) Append "$\oplus d_i$" at the end of the expression.

Example 1

Suppose we want to find out the expression for $RP_{4,10}$. j equals ten and it is expressed as

$$j_3j_2j_1j_0 = 1010.$$

The number of "1" in the first run of digit is one, hence $l = 1$. It can be seen that d_0 will not be involved in the final expression for $RP_{4,10}$. After deleting j_3 from j, the expression for j becomes

$$j_2 j_1 j_0 = 010.$$

Starting with x equals l, which is one in this example, d_1 is appended with a "$\wedge$" as j_1 equals one. With a "$\oplus\ d_4$" is appended at the end, the final expression becomes

$$RP_{4,10} = d_1 \wedge d_2 \vee d_3 \oplus d_4.$$

Example 2

In this example, the expression for $RP_{6,3}$ has to be worked out. Since i is not equal to zero in this example, j has to be expressed in binary format as

$$j_5 j_4 j_3 j_2 j_1 j_0 = 000011.$$

The first run of digits of the above j is four "*0*", hence l equals zero as the first run of digits is not a consecutive sequence of "*1*". Up to this step, it can be concluded that all d_0 to d_6 will in involved in the expression for $RP_{6,3}$. As l equals zero, none of the digits have to be deleted from j. The first digit, j_5 in this example, has to be ignored. Starting with x equals l, which is zero in this example, d_0 is appended with a "$\vee$" as j_4 is equal to zero. Since both j_3 and j_2 are zero, the expression is appended with two more "$\vee$" with d_1 and d_2 in between. The partial expression of $RP_{6,3}$ is now "$d_0 \vee d_1 \vee d_2 \vee d_3$" after j_2 is considered. With both j_1 and j_0 are equal to one, two "$\wedge$" are added to the expression together with d_4 and d_5. In the final step, "$\oplus d_6$" is appended, and the final expression of $RP_{6,3}$ becomes

$$RP_{6,3} = d_0 \vee d_1 \vee d_2 \vee d_3 \wedge d_4 \wedge d_5 \oplus d_6.$$

With the algorithm to determine all the RP's with a straightforward

approach in a compressor, the combinatorial logic circuitry can be realized for implementation. It can be seen in the expressions for RP's that only the operations of AND, OR and XOR are involved, and the circuitry for a 16×16 compressor is shown in Figure 2.10.

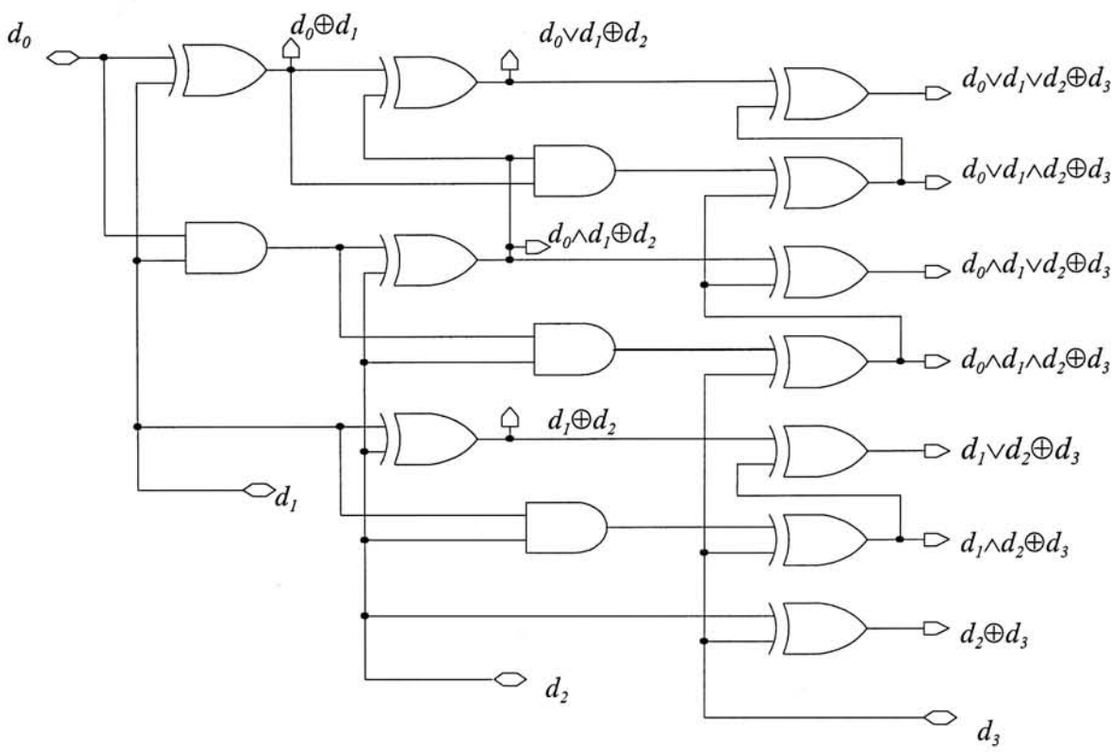


Figure 2.10 Circuitry for a 16 × 16 compressor

While it is much more desirable to derive all the RP's to different stages in a direct and explicit approach, the generation of some RP's may depends on other RP's generated in earlier stages. By reusing signals, the total gate count for the whole circuitry can be reduced.

Corollary 2.4.3 When the active packets are compressed into a cyclic consecutive sequence, the idle ones are also compressed with the mapping of the

idle input addresses to their corresponding output addresses is monotonically decreasing.

2.5 Input fairness

In all the 0-1 sorters discussed thus far, considering the M-to-N concentration process where only the first N outputs are concerned in the 0-1 sorter. If the total number of active inputs is no more than N, all the active packets can be routed, otherwise, overflow occurs and some active packets must be queued or discarded. Just like the running sum adder, there exists the inherent bias against the lower inputs in the baseline-swap 0-1 sorter. The related issue is the *input fairness*. As the signal RP is traversing from top to bottom, an active signal is considered and routed to the next stage first before any signal lower than it get to be routed to the following smaller 0-1 sorter. Active signals at lower input ports have a higher chance of being blocked or have to be buffered than those at the upper input ports under heavy traffic situations. It can be seen at the output side that the top down sequence of active signals at the input side is preserved. In this case, cyclic running sum adder network is introduced to achieve the input fairness in reverse banyan network. In the cyclic structure, partially computed running sum from the lowest adder is fed to the top adder. A statistical approach is proposed in this section to cope with the issue of input fairness in baseline-swap 0-1 sorter.

Two sets of control are applied to the baseline-swap network alternately

from time slot to time slot, so that in one time slot precedence will be given to signals at upper ports while signals at the lower ports will be considered first in the following time slot. The mechanism is as follow.

In the baseline-swap 0-1 sorter, all the RP's are set to zero so that the packets will be concentrated at the upper part of the output ports. On the contrary, we can take complement of all the RP's, i.e. logic one, and then the compressor stacks the active packets upward from bottom to top. Notice that it is also one of the output pattern of the compressor. To all of the compressors discussed so far, the compression property is also tenable when the mapping from the active input addresses to their corresponding output addresses is monotonically decreasing.

The input fairness is achieved by:

a. In the ith time slot, all the RP's are set to zero so that packets at upper ports are concentrated and the network is just a compressor for concentration;

b. In the $(i+1)$th time slot, the number of active packets, k, is used as the starting output address in the compressor and it is expressed in binary form as before. All the RP's are obtained by using algorithm 2.4.2. However, complement of each calculated RP is taken before applying to the network, i.e. logic one is applied to the iterative cell if zero is the calculated value.

A compressor being operated in the mode for $(i+1)$th time slot is described in Figure 2.11.

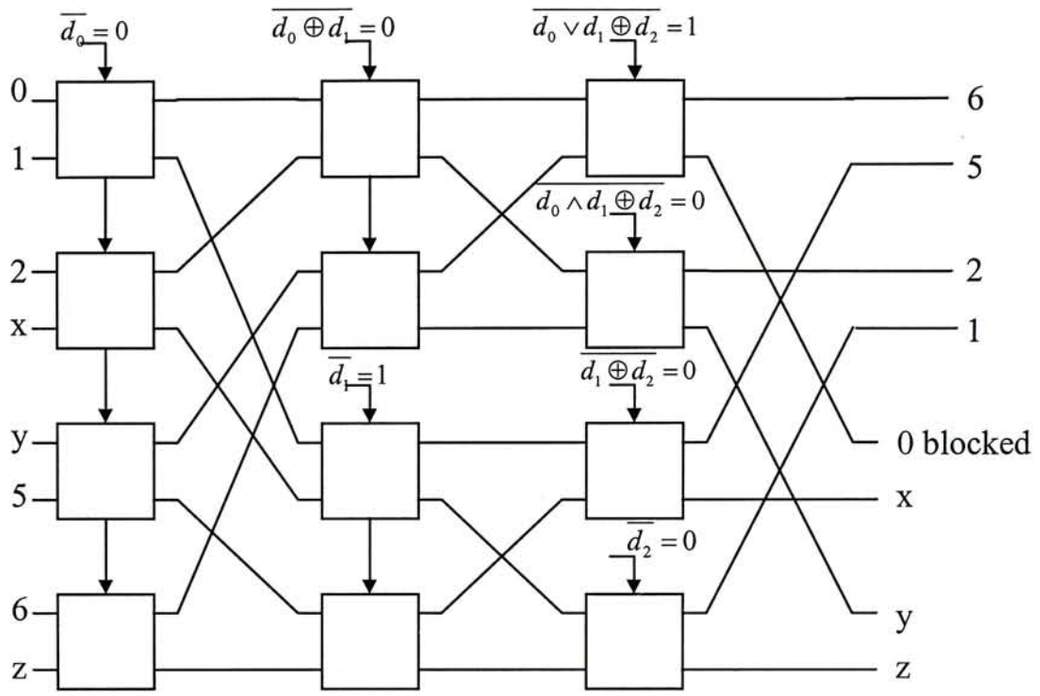


Figure 2.11 A compressor together with a reverse set of control to the original mode

Suppose there are packets at input ports 0, 1, 2, 5 and 6. There are total 5 active inputs and hence k equals 101 in binary form. With k equals 101, all the RP's are calculated and complements are taken before applying to the iterative cells. The routing of idle inputs are also shown in Figure 2.11 to illustrate the fact that the mapping of them is the opposite of those active one.

Figure 2.11 shows that active signals at lower input ports achieve higher priority under the operation of step *b*. The reasoning is as follow. By corollary 2.4.3, the input-output address mapping of idle inputs is monotonically decreasing while the active ones is monotonically increasing in the compressor if all the RP's are set to zero. If all the RP's are set to one instead, the direction of mapping is just reversed and the first active signal at the input will be at the

lowest output.

Obviously, the network is also a compressor to the idle inputs. If the mapping from the idle input addresses to their corresponding outputs is monotonically increasing (here, the number of active packets is used as the starting output address), the active ones are also compressed with the mapping from their input addresses to corresponding output addresses which is monotonically decreasing. Therefore, considering the idle inputs, $\overline{RP}$'s are the initial values to the compressor.

With this fact, the RP's are slightly modified so that from top to bottom the last active signal at input side will be presented at the first output ports, and then the second last active signal and so on. Following the above two steps in every consecutive time slots, signals at lower input ports should have a similar chance of blocking with signals at upper input ports after long time span.

Chapter 3 The general architecture of 0-1 sorter

In the previous chapter, it is shown that the recursively-constructed 0-1 sorter is actually a baseline network appended with a swap exchange. The baseline-swap network is capable to be a compressor. Actually, the baseline-swap network is just a member of the family call *2X-networks* which are obtained by *recursive 2X-construction* [9], and it is the general architecture of 0-1 sorter bases on. The baseline network is among those that incur the highest complexity in terms of the 2-layer Manhattan layout, while the *divide-and-conquer networks* (Definition 3.5.9 in [9]) achieve the lowest complexities (Theorems 4.2.15 in [9]). In view of this, we are interested in the generic 2X-interconnection rather than just those involved in the baseline network appended with swap exchange. As the recursive 2X-construction itself also preserves the compressor properties, network built based on the general architecture can be viewed as a module in a much larger 0-1 sorter. With the recursive construction, a 0-1 sorter with a large number of input ports can be realized by connecting various modules as switching blocks in the 2X-network. We derive an explicit formula for the initialization of running parities that control a generic 2X-construction network into a cyclic 0-1 sorter, and it will be introduced in this chapter.

3.1 Recursive 2X-construction

The recursive 2X-construction network consists of two stages of switching

modules, the first stage and the second stage, as well as an exchange appended to the second stage of switching modules. A 2X-network is shown in Fig. 3.1. Actually, the name of 2X-network is originated from the "2" stages of switching module and the final exchange (X) appended.

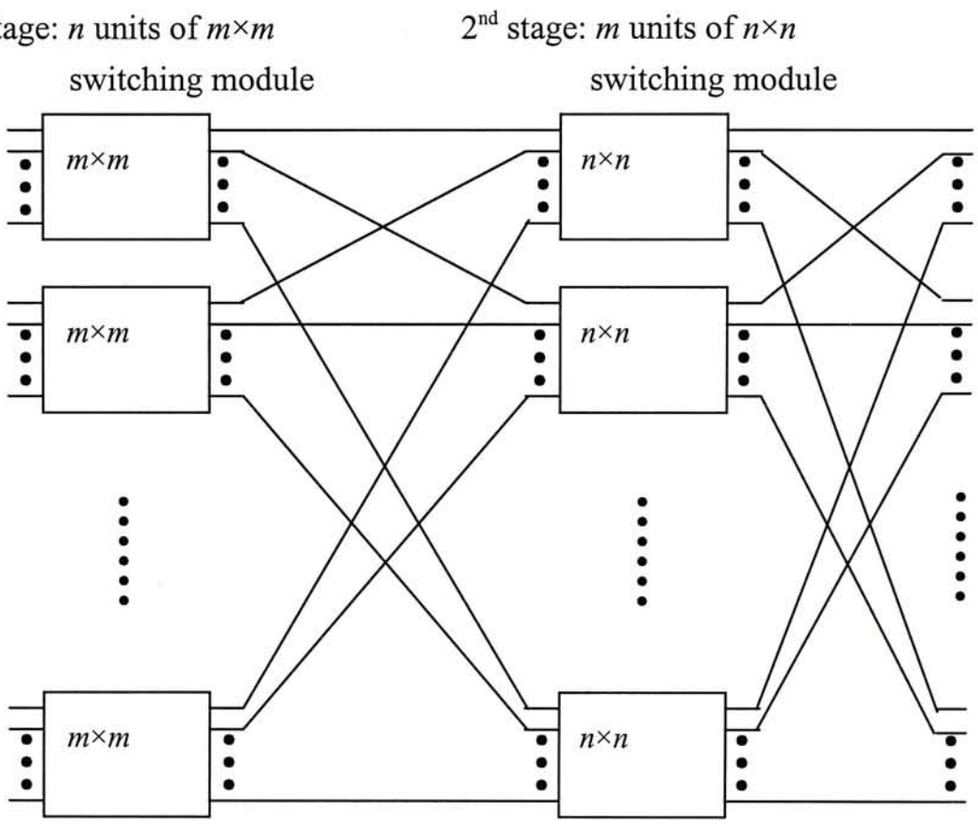
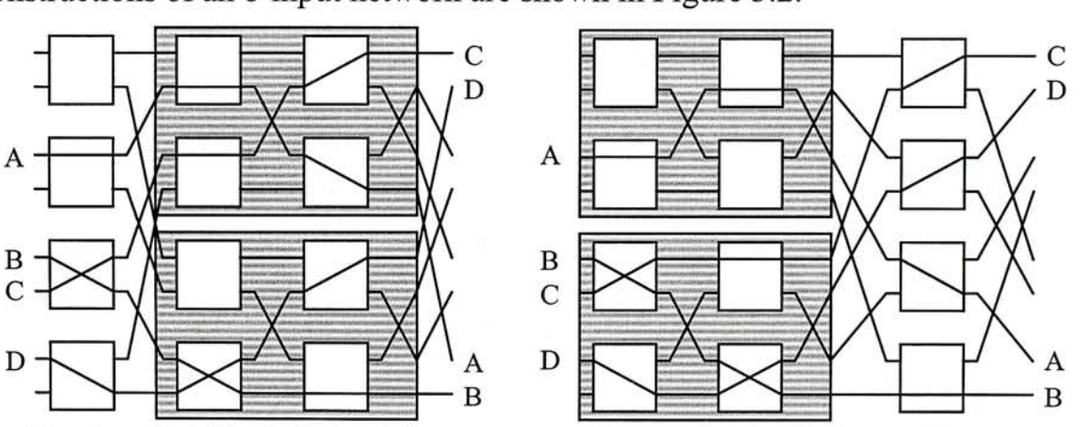


Figure 3.1 A 2X-network

The topology of the network is described as follow. Suppose network with a total of mn (i.e. m times n) input ports and the same number of output ports is to be built, where both m and n are any positive integers other than one. The first stage will be n units of switching module, each of them is a $m \times m$ one. For the second stage, there will be m units of switching module and each of them is a $n \times n$ one. Switching modules in the first and the second stages are labeled from 0

to n-1 and 0 to m-1 from top to bottom, respectively. Then the j^{th}-output port of the i^{th} module in the first stage is connected to the i^{th}-input port of j^{th} module in the second stage. In addition, there is an exchange appended to the second-stage modules. The final exchange is just the inverse, or the mirror image, of the exchange between the two stages of switching modules in the 2X-network, i.e. the shuffling effect will be cancelled if the two exchanges are appended directly.

The recursive 2X-construction allows flexibility, as networks with the same number of input ports can be of various possible topologies, depending on how the (m, n) pairs are chosen. For example, the (m, n) pair of a 16-input 2X-network can be either one of (2, 8), (4, 4) and (8, 2). The (m, n) pair of an 8-input 2X-network can be either (2, 4) or (4, 2). Both the possible 2X-constructions of an 8-input network are shown in Figure 3.2.



(a) (m, n) = (2, 4) for an 8-input network

(b) (m, n) = (4, 2) for an 8-input network

Figure 3.2 Two 8-input 2X-networks

In Figure 3.2(a), the first-stage switching modules are 2×2 switching cells while those in the second stage are 4×4 switching modules, which are in the shaded boxes. In the shaded boxes, the 2X-construction method is applied until

all the switching modules within the network are 2×2 switching cells. The same occurs in Figure 3.2(b). When the 2X-construction method is applied recursively to all the switching modules within the network so that all the switching modules are 2×2 switching cells, the networks with the same number of input ports are actually functionally equivalent. It means a network resulting from recursive 2X-construction can be transformed into another, of the same dimension, by shuffling 2×2 switching cells within the same stages. For example, in Figure 3.2, network in Figure 3.2(a) can be transformed into Figure 3.2(b) simply by swapping the second and third 2×2 switching cells in both the second and third stages.

It should be noticed that the network in Figure 3.2(a) is the same as the one in Figure 1.12. After the final exchange in Figure 3.2(a) is rearranged, it becomes a baseline-swap network. In general, the baseline-swap network topology is a special case of the 2X-network. It is constructed by applying the recursive 2X-construction, choosing m equals two in every step, until all the switching modules in the network are 2×2 switching cells.

It is also depicted In Figure 3.2 the sets of connections. Signals are presented at input ports 3, 5, 6 and 7, and they are routed to output ports 7, 8, 1 and 2, respectively. In this connection assignment, the output addresses are circularly consecutive. Also, the mapping from the active input addresses to their corresponding output addresses is monotonically increasing. Since the 2X-networks in Figure 3.2 allow the above connection assignments, it is illustrated

that 2X-network preserves the compressor properties. It is proved [9] that if all the switching modules within the network are compressors, the 2X-construction preserves the compressor properties as well.

3.2 Control a 2X-interconnection network as a cyclic 0-1 sorter

In the previous section, it is stated that a 2X-network is capable of being a compressor, provided that every switching module within is compressor. However, interaction between switching modules is required for the 2X-network to function properly as a compressor. In this section, the control algorithm to the network, which is mainly the interaction between modules of the same stage, will be introduced.

As mentioned, compressor can route the signals to any set of circularly consecutive outputs. Hence, out-band control to a compressor is needed to inform the device which set of output ports should the signals be routed to. In addition, according to Definition 2.2.1, suppose we have chosen that the mapping from the active input addresses to their corresponding output addresses is monotonically increasing. Then the only information for compressor to know is the output address of the *first* (starting from the top) active input. With the output address of the first active input, the other signals will be routed to from a circularly consecutive set while the mapping from the active input addresses to their corresponding output addresses is monotonically increasing.

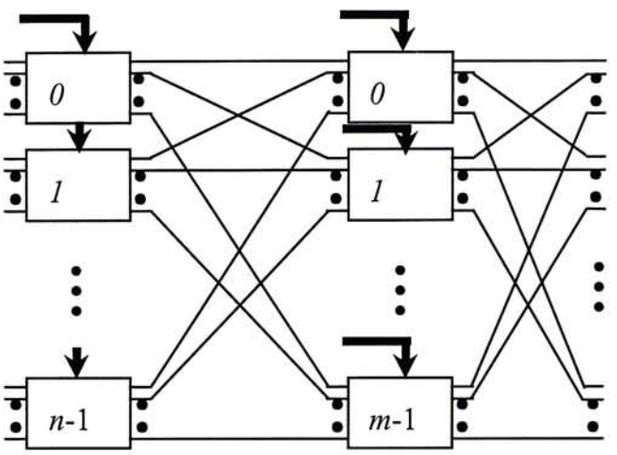


Figure 3.3 2X-network with the external control on both stages

The network in Figure 3.1 is recited in Figure 3.3 for the illustration of the control. Different from Figure 3.1, Figure 3.3 shows that out-band control signal is required for both first stage and second stage switching modules. The external controls to the two stages are different. In the first stage, all the modules are *serially* connected as out-band control is generated and passed down from top to bottom. On the other hand, out-band controls to the second stage modules are independent from each other.

The control algorithm is described as follow. Suppose the output address of the first active input is D, $D \in \mathbf{Z}_{mn}$. As D is any integer from 0 to mn-1, it can be written as $D = xm+y$, where $0 \le x < n$, $0 \le y < m$. The value y is fed into the top most switching module, module 0, in the first stage. Since the modules in the first stage are serially connected, the out-band control signal has to be passed down to the following module. Suppose the number of active input presented at each of the first stage modules are k_p, $0 \le p < n$. Then the value $(y+k_0)\ mod\ m$ will be passed down as out-band control to module 1. In general, module i receives

$(y+\sum_{i=0}^{i-1} k_i)\ mod\ m$ as out-band control and passes down $(y+\sum_{i=0}^{i} k_i)\ mod\ m$ to the following stage. Simply speaking, the out-band control in the first stage is the running sum modulo m, with an initial value y. As stated, it is required that all the switching modules within are compressors for the 2X-network to be a compressor. Similar to D to the 2X-network, $(y+\sum_{i=0}^{i-1} k_i)\ mod\ m$ is actually the starting address of the first active input among the m inputs of the switching module i on the first stage. Once $(y+\sum_{i=0}^{i-1} k_i)\ mod\ m$ is fed in as the out-band control to module i of the first stage, signal will be routed in a way that they start stacking at output port $(y+\sum_{i=0}^{i-1} k_i)\ mod\ m$ of module i. Also, the mapping of the active input addresses to their corresponding output addresses is monotonically increasing. In the second stage, every out-band control to each of the switching module are independent. It is obtained by comparing y with the position of the module. The second stage modules are labeled from 0 to m-1 from top to bottom. If $i \geq y$ the out-band control to module i will be x. On the other hand, if $i < y$, the out-band control to module i will be $(x+1)\ mod\ n$.

Algorithm 3.2.1 (Control to 2X-network with (m, n) as parameters)

D is the output address of the first active input, $D \in \mathbf{Z}_{mn}$, and it is expressed as $D = xm+y$, where $0 \leq x < n$, $0 \leq y < m$. The number of active input presented at each of the first stage modules are k_p, $0 \leq p < n$.

Control of module i of stage 1:

If (i equals 0)

Control = y;

Else Control $= (y + \sum_{i=0}^{i-1} k_i)$ *mod m.*

Control of module i of stage 2:

If (i ≥ y)

Control = x;

Else Control = (x+1) mod n.

Example 1

A 24×24 compressor is to be constructed, and the (*m*, *n*) pair is (6, 4). Suppose *D* equals 20 and there are 5 active inputs, located at 0, 7, 13, 20 and 21. The network is shown in Figure 3.4.

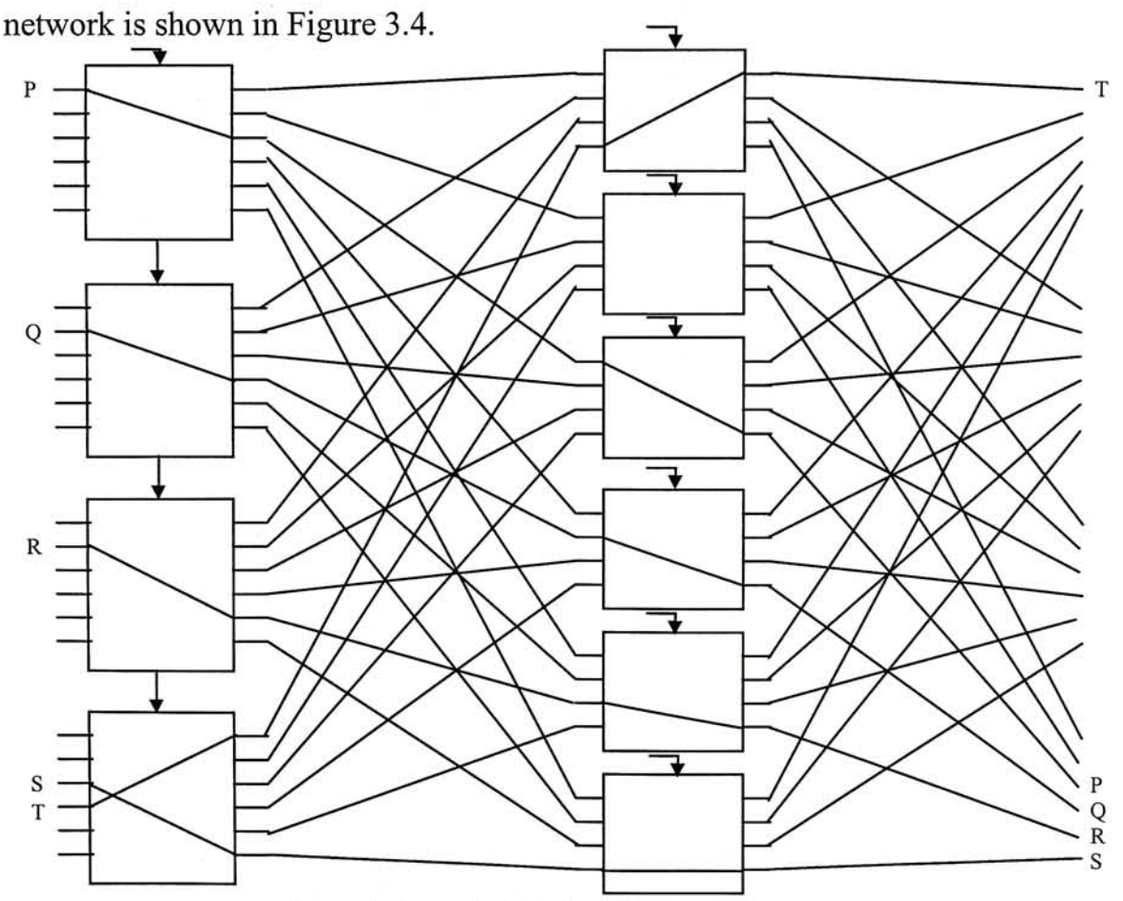


Figure 3.4 A 24×24 2X-network

As D equals 20, it can be expressed as $D = 6x+y$, where x and y equal 3 and 2, respectively, in this case. As y equals 2, signal P at module 0 of the first stage will be routed to output 2. The running sum modulo m is passed down to module 1, but now with value 3 since 1 is added as there is one active input presented at module 0. Signal Q is routed to output 3 as the running sum modulo m is now 3. The procedure is applied to each of the first stage module until the running sum modulo m is passed down to the last module. Hence running sums modulo m to modules 2 and 3 are 4 and 5 respectively. After the routing of the first stage, signal P, Q, R, S and T are now presented at the input sides of modules 2, 3, 4, 5 and 0 of the second stage, respectively. As y equals 2, the out-band control signal to modules 2, 3, 4 and 5 will be x, which is 3 in this case. On the other hand, the out-band control signal to modules 0 and 1 will be $(x+1)$ *mod* n, which is 0 in this case. It can be seen at the output side of the network that the signals from a circularly consecutive sequence, and the mapping of input addresses to their corresponding output addresses are monotonically increasing.

Example 2

It is going to show in this example the control to a $2^k \times 2^k$ 2X-network, which is a little bit simpler than those in non-$2^k \times 2^k$ 2X-networks. Consider a 16×16 2X-network, where k equals 4 and it is shown in Figure 3.5.

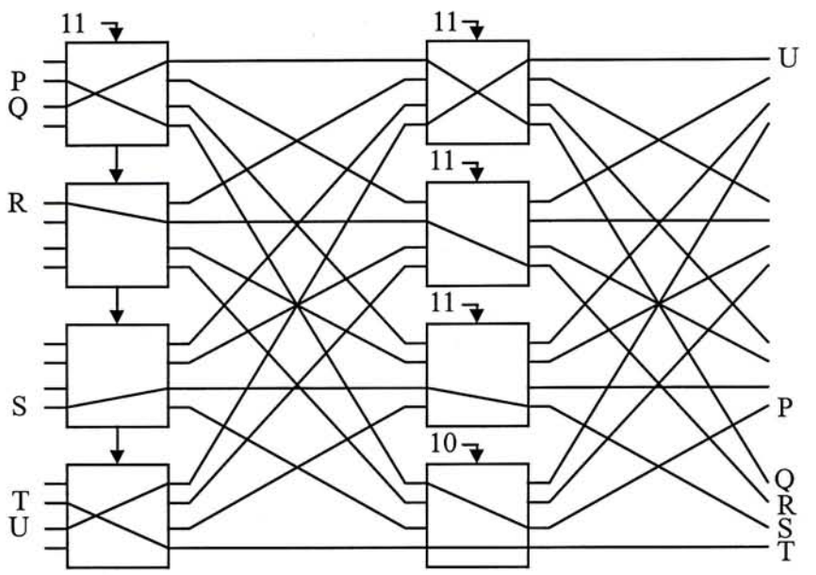


Figure 3.5 A 16×16 2X-network

Suppose D equals 11 in this case. Different from the previous example, D is only needed to be expressed in binary format. D is expressed as $1011_{(2)}$ in this case. The out-band control signal to the first stage of switching module is the last two bits of D. In general, for a $2^k \times 2^k$ 2X-network with a particular (m, n) pair, the last $log\ m$ bits of D can be viewed as y while the first $log\ n$ bits can be viewed as x in the above mentioned control algorithm. $11_{(2)}$ will be fed into module 0 of the first stage in this example. Since there are two active inputs presented at module 0, the running sum modulo 4 will be $(11_{(2)}+10_{(2)})$ mod 4, which equals $01_{(2)}$, and this value will in turn be fed into module 1. This process is carried on until the running sum reaches the last module of the first stage. Since the initial value of the running sum modulo 4 to the first stage is $11_{(2)}$, the out-band control value to module 3 in the second stage will be $10_{(2)}$ while the other three are all $11_{(2)}$. This result simply follows algorithm 3.2.1 where x equals $10_{(2)}$ and y equals $11_{(2)}$. Once all the values are determined, the signals can

be routed to the particular set of addresses.

3.3 Recursive construction of large 0-1 sorter

A 2X-network can be used to further construct a larger compressor with more input and output ports. For instance, by using the 24×24 and 16×16 compressors in the above two examples as building blocks, a 384×384 compressor can be built, where the 24×24 and 16×16 compressors are used as the switching modules in the 384×384 compressor. As mentioned, first stage switching modules are serially connected and each of the module has to further generate the out-band control signal, which is simply the running sum modulo m, to the following module. Hence, in building a large 2X-network using smaller 2X-networks as switching modules, those smaller 2X-networks should be capable of further generating the out-band control signal beside receiving them. Actually, the generation of out-band control signal by the 2X-network is very simple, and it is described as follow. The running sum modulo m after the last module of the first stage will be $[(y + \sum_{i=0}^{n-1} k_i) \ mod \ m]$. For the second stage, we need to consider the last module only. The out-band control to the last module of the second stage must be x. Let the number of signals presented at the input side of this last module is k'. Then like those first stage modules, the running sum modulo n of this last module is $(x+k') \ mod \ n$. Then the out-band control signal, D', will be $D' = [(x+k') \ mod \ n]\times m + [(y + \sum_{i=0}^{n-1} k_i) \ mod \ m]$. If the compressor is a

$2^k \times 2^k$ network, D' is simply the concatenation of the two running sums of the two stages. For example, the last modules of both first and second stages, in Figure 3.5, together with the signals, are illustrated in Figure 3.6.

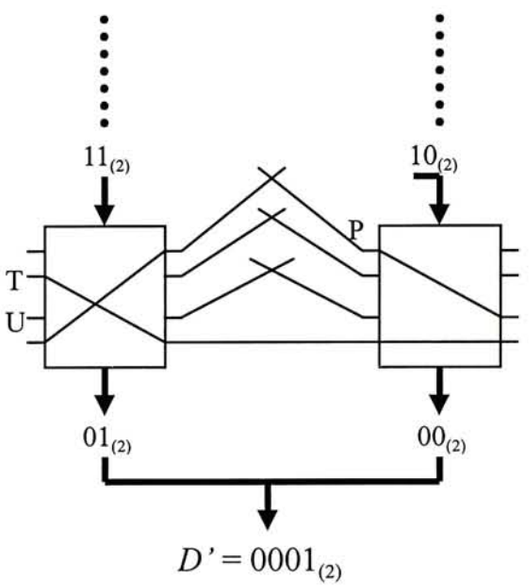


Figure 3.6 The generation of D' on the network level

In the concatenation, it can be seen that the less significant bits are from the first stage, while the more significant bits are from the second stage. In Figure 3.6, $01_{(2)}$ is the running sum modulo 2 from the first stage and $00_{(2)}$ is the running sum modulo 2 from the second stage. Their concatenation, $0001_{(2)}$, will be treated as D in the following module and the first signal in it will be routed to output port $0001_{(2)}$.

As mentioned, the baseline-swap network in [13] is an example of 2X-network. The baseline-swap network is composed of iterative cells, and the out-band control associated with the iterative cells also resembles the control algorithm in the 2X-network indeed. In the 0-1 sorter, since the concentrated

signals should start stacking at the first output port, D equals to zero. For a $2^k \times 2^k$ 0-1 sorter, it is constructed by choosing (m, n) pair as $(2, 2^{k-1})$ and the $2^{k-1} \times 2^{k-1}$ network is further constructed by 2X-construction. 2X-construction is recursively applied until all the elements within are all 2×2 iterative cells. The iterative cell can be viewed as a 2×2 compressor, receiving the running parity so that it will guide whether the upper or the lower output port the first active signal should go. With this recursive 2X-construction, a large 0-1 sorter can be built using smaller compressors as the building blocks.

Chapter 4 Epilogue

In Chapter 2, the baseline-swap network has been introduced and its usage as a compressor has been investigated. Furthermore, it is generalized in chapter 3 that baseline-swap network is just a member in the family of 2X-networks. For the algorithms proposed in chapter 2 and 3 concerning about the routing of the signals and generation of running parities, they are theoretically feasible. In this chapter, the implementation problems and the further directions of research will be investigated.

4.1 Directions of further studies

4.1.1 Synchronization within the same stage

Consider a $2^k \times 2^k$ baseline-swap network constructed using iterative cells. In the first stage, running parity is propagating from the top iterative cell to the last iterative cell of the same stage. In this implementation, there maybe the situation that packets are ready to be sent to the second stage at the upper half of iterative cells, while those iterative cells at the lower half are still waiting for the running parity to determine the bar/cross state of the cells. Synchronization would be hard to maintain. This problem would be accentuated if k is very large and the number of iterative cells per stage increases, as the running parity has to pass through more iterative cells in serial.

One of the possible solutions to this is to add dummy content to the packet, i.e. a training sequence is added prior to the real processing of data packet. With

the dummy content, it can be guaranteed that all the running parities reach the last iterative cells of their respective stages and the routing is determined before the real data enters the first stage of iterative cells. Hence synchronization can be maintained.

In section 1.2.5, it is mentioned that the running sum adder can be constructed using the two-phase algorithm, which is associated with the binary tree. The main advantage of this implementation over the serially connected running sum adder in Figure 1.8 is that the running sum adder associated with the binary tree maintains a better synchronization. A similar technique can be applied to the baseline-swap network of iterative cells. Instead of passing the running parity from top to bottom, the iterative cells of the same stage can be viewed as the leaves of a binary tree and the running sum of each iterative cell is passed up to the root. Then the running parities to different iterative cells are passed down from the root, and the iterative cells would receive them at the same time as they are all at the lowest level of the binary tree.

4.1.2 Layout complexity

It is stated in Chapter 3 that a 0-1 sorter of certain dimension can be constructed with different topologies by choosing different (m, n) pairs in the recursive 2X-construction. With different (m, n) pairs in each of the recursive step, the 0-1 sorter constructed would induce different complexities. For interconnection network of 2×2 cells, the major component of the layout

complexity depends on the exchange between any two stages. For all interconnection networks of 2×2 cells, the lower bound for the layout complexity is associated with the *divide-and-conquer network* [9]. For the 0-1 sorter by recursive 2X-construction, a similar network can be obtained by choosing appropriate (m, n) pairs in different steps. However, for such a 0-1 sorter, it would induce higher complexity than a divide-and-conquer network. The higher complexity is due to the extra wiring for carrying the running parities within the same stage of iterative cells, which cannot be found in a general interconnection network.

As the explicit set of formulae for determining the running parities in baseline-swap network is derived, it may be possible to derive a similar set for any other network resulting from recursive 2X-construction. However, they are not as simple as the case for baseline-swap network. Consider the reverse banyan network constructed by choosing n equals 2 in every step in the recursive 2X-costruction, which is illustrated in Figure 4.1. For the baseline-swap network, since m equals 2 in the first stage and switching modules are serially connected. As a result, the one-bit running parity can be transmitted from top to bottom at every front-end stage of iterative cells. However, in the construction of reverse banyan network, the first-stage switching modules are not 2×2 cells while the second-stage elements are. Hence, the running sum modulo m is transmitted serially, which is a signal of $log\ m$ bits. In order to obtain the explicit set of formulae similar to the one for baseline-swap network

in any other 2X-networks such as reverse banyan network, single-bit signals have to be combined to form the running sum modulo m before they can be transmitted to the next module. The transmission of more than one bit signal complicates the process of finding the explicit formulae. The derivation of the explicit formulae for running parities depends on the running sum of those previous modules, while those formulae for baseline-swap depends on D_n solely.

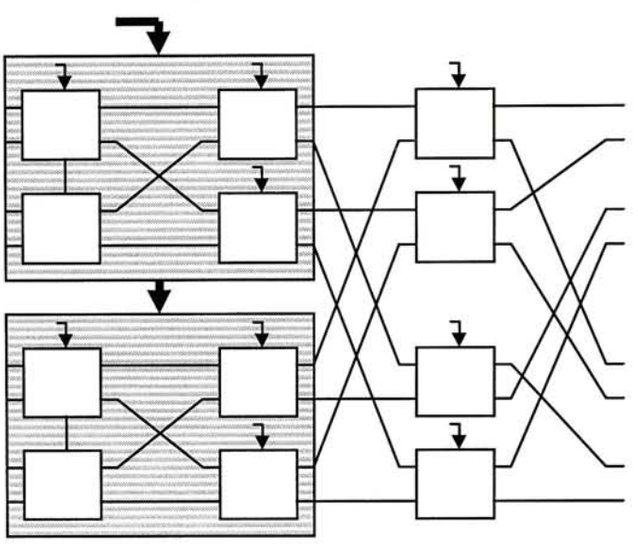


Figure 4.1 An 8×8 reverse banyan network together with external control as cyclic 0-1 sorter

4.1.3 Statistical initialization of running parity

According to algorithm 3.2.1, it can be seen that the running parities to the switching module in the second stage of the 2X-network is either x or x+1. If only one value is obtained and applied to all the second-stage modules, the workload for computation would be less, and this maybe seen as a statistical way of concentration. With statistical concentration, the performance of the 0-1 sorter can be investigated and see whether it is acceptable.

4.2 Conclusion

Compressor is a concentration network that compresses active inputs to circularly consecutive outputs starting at any given output address. In many ATM switching design, this kind of network plays an essential role as a distributor in buffer allocation and building of virtual FIFO queue [4]. A compressor can be constructed from iterative cells whose bar/cross state controlled by in-band signals from the two inputs plus an external signal, the running parity. First of all, the family of bit permutation induced exchanges is introduced in the thesis. The family contains many commonly seen exchanges, such as banyan exchange and shuffle exchange. Swap exchange is also defined, according to the bit permutation induced exchange. It is shown that the recursively-constructed 0-1 sorter is actually a baseline network appended with a swap exchange at the end. As this network topology is capable to be a compressor, the thesis investigates an explicit and immediate algorithm for initializing each RP in the baseline-swap network based on iterative cells. This algorithm is deduced from the calculation of the running sum stage by stage. With this algorithm, a practical logical circuitry can be realized. Besides, the construction of the baseline-swap network is later generalized to be the recursive 2X-construction. With the recursive 2X-construction, 0-1 sorter of the same dimension can be of various topologies. Furthermore, the 2X-construction can be applied recursively, so that 0-1 sorter of large dimension can be obtained by

connecting smaller 0-1 sorter according to the 2X-construction. The algorithm for the initialization of RP in any 2X-network for it to be compressor is also proposed, which is essential in building a large 0-1 sorter.

REFERENCES

[1] K. Batcher, "Sorting networks and their applications," *Proc. Spring Joint Computer conference, 1968*, pp. 307-314.

[2] J. Giacopelli et al., "Sunshine: A high performance self-routing broadband packet switch architecture," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1289-1298, Oct. 1991.

[3] A. Huang and S. Knauer, "Starlite: A wide band digital switch," *Proc. GLOBECOM'84*, Nov. 84, pp. 121-125.

[4] H. S. Kim, "Multichannel ATM switch with preserved packet sequence," *Proc. ICC'92*, pp. 1634-1638.

[5] H. S. Kim and A. Leon-Garcia, "A self-routing multistage switching network for broadband ISDN," *IEEE J. Select. Areas Commun.*, vol. 8, no. 3, pp. 459-466, Apr. 1990.

[6] H. S. Kim and A. Leon-Garcia, "Nonblocking property of reverse banyan networks," *IEEE Trans. Commun.*, vol. 40, no. 3, pp. 472-476, Mar. 1992.

[7] D. E. Knuth, *The Arts of Computer Programming, Vol 3: Sorting and Searching*, Addison-Wesley, 1973.

[8] J. G. Lee and B. G. Lee, "A new distribution network based on controlled switching elements and its applications," *IEEE/ACM Trans. Networking*, vol. 3, pp. 70-81, Feb. 1995.

[9] S.-Y. R. Li, *Switching Theory and Broadband Applications*, Academic Press, Sep. 1, 2000.

[10] S.-Y. R. Li and C. M. Lau, "Concentrators in ATM switching," *Proc. GLOBECOM'95*, vol. 3, pp. 1746-1750.

[11] S. C. Liew and K. W. Lu, "A 3-stage interconnection structure for large packet switches," *Proc. ICC'90*, pp. 316.7.1-316.7.7.

[12] P. S. Min, H. Saidi, and M. V. Hegde, "A nonblocking architecture for broadband multichannel switching," *IEEE/ACM Trans. Networking,* vol. 3, no. 2, pp. 181-198, Apr. 1995.

[13] M. J. Narasimha, "A recursive concentrator structure with applications to self-routing switching networks," *IEEE Trans. Commun.*, vol. 42, pp. 896-

898, Feb/Mar/Apr. 1994.

[14] A. Pattavina, "Design of a packet concentrator for broadband networks," *Proc. ICC'90*, vol. 3, pp. 817-821.

[15] Y.-S. Yeh, M. G. Hluchyj, and A. S. Acampora, "The knockout switch: A simple, modular architecture for high-performance packet switching," *IEEE J. Select. Areas Commun.*, pp. 1274-1283, Oct. 1987.