1. Connection And Sorting Networks

Most of the material presented here is covered, often from a different point of view, in [Hui,90].

Consider the parallel computer model in which multiple processors have access to multiple memories. Each processor can access each memory and various combinations of processors can have simultaneous access to various combination of memories. To be feasible, this requires fast and reasonably inexpensive hardware, to connect, or, more precisely, send signals from N input lines (processors) to M outputs (memories) and vice-versa. One candidate is the simple **crossbar switch** (left side of figure 1-1). It has N inputs and M outputs and provision for connecting each input to any output through an independent contact or **crosspoint**. The crosspoint count, NM, is a measure of the cost of such a switch. The crossbar switch has several desirable properties:

1. Any M of the N inputs can be connected in any order (**permutation**) to its M outputs.
2. The algorithm for assigning cross points for closure, given the outputs to which each of M inputs is to be connected, is very simple and fast.
3. Less important, but also interesting, is the fact that the crossbar switch can be used to connect a number of inputs to the same output or outputs.

There are a number of designs for connection networks whose costs are substantially less, even order-wise, than that of the simple crossbar. Generally such networks are built of stages of crossbar switches of size $< < NM$. In trade for their lower cost, such networks include some which only lack the third property above, others which lack both the second and third property, and some, the least expensive, which have none of these three properties.

1.1. Classes Of Multistage Connection Networks

Generally connection networks establish many independent paths from inputs to outputs along which information is passed. In the connection switching mode such a path may be maintained continuously while the message is delivered. Alternatively the first link of a path can be established, then a information passed through that link to an electronic node where it is stored until the next link in its path is available, and it is passed on. This message switching mode is useful when network blockage is likely and messages can be organized into small packages. Switched connection networks typically consist of a series of stages, in each of which one link of a path is established. In the first stage switches the first link of each independent path is established, these are connected to the second stage where an independent link is established in second stage switches, etc. This arrangement is ideal for pipelining short messages. After a set of messages have passed through the links in stage i they pass to stage i+1 and a *next* set of messages may enter stage i. However there may be messages of different length and the necessary link in stage i+1 for a particular path may not be available when a message has passed through the link in stage i. This leads to "store-and-forward" operation in which the pipelining becomes asynchronous, supported by storage necessary to hold messages as they wait for the next state links to be available. This is an effective way of operating even if the network is not easily organized in stages, as long as it support multiple independent paths through sequences of links.

The networks initially surveyed are non-blocking or nearly so and viewed as used for making multiple connections simultaneously and holding connections for relatively long time--like a network for connecting local telephone calls (this is the context in which they were originally discovered). The topology of the networks and their applicability apply to both the connection and transmission

model. To get to a store-forward model, NET', from a given connection model NET:

1. Each switch input port of NET is replaced by a transmitter, which receives that ports input signal and pass it onward through any one of the crosspoints connected directly at that port. And each switch output port receives its inputs from any of a number crosspoints and sends it on to a single input port. (Each output port which receives signals from a set of crosspoints can be combined with the one possible recipient of its output; an input port from which outputs can be transmitted to a set of crosspoints. So these combined receiver-transmitters are the basic components of the store and forward NET'. The first stage input ports are only a transmitter and the last output port are only receivers. So the number of receivers equals the number of transmitters.

2. permanently closing these crosspoints, and interpreting the closing of a crosspoint in NET as transmission along that, now closed, crosspoint in NET'.

The number of transmitters is proportional to the number of switch input ports in a network, as is the number of crosspoints. If the complexity of a transmitter is proportional to the number of its outputs then its complexity is O(number of crosspoints in NET). And since the transmitters are each paired with a receiver, O(number of crosspoints in NET) is also the number of receiver-transmitters. (see figure 1-2)

The networks are assumed to be passing signals in one direction. They can be made two-way by adding another independent network going in the opposite direction, or placing receiver transmitters in the opposite direction at each node and sharing the lines two-way, between pairs of receiver-transmitters.

The first class of networks considered are associated in the literature with Bell Labs scientists **Clos** and **Benes**, both of whom made significant contributions to the elucidation of their properties [Benes,65]. We call them **classical non-blocking networks**. The connections to be set up in an N-by-M network, assuming $N \geq M$, can be represented by a permutation of the first M integers and an ordered selection of M of the N inputs. If the permutation is $<p_1, \ldots, p_M>$, and the ordered selection from the N inputs have indices $<s_1, \ldots, s_M>$, then the $s_i th$ input is to be connected to the $p_i th$ output. If $M = N$ then only the permutation is required, for then the $i th$ input is to be connected to the $p_i th$ output. A network is non-blocking if all $M!$ permutations of the outputs can be set up in the network for each possible selection of M of the N inputs.

Here are two classes of non-blocking networks.

1. **Oblivious or strictly or pure non-blocking** : Each permutation can be set up by connecting its members, one at a time *in any order*, with no knowledge of those not yet connected, without the need to rearrange any connection already made.

2. **Rearrangeable non-blocking** [Paull, 64]. Each permutation can be set up looking at its members one a time in any order, but there is no algorithm which guarantees that previously made connections will not have to be rearranged. If the entire permutation is known before any connections are made the connections can be planned so that when set up one at a time they will not require rearrangement.

The class of networks considered now is pictured in figure 1-1. This three stage configuration representation is recursive: in the $1st$ and $3rd$ stages each block represents a crossbar switch, the central stage blocks (switches), called **CSSs** each may represent single crossbar switches or entire networks, themselves built out of smaller crossbars in an similar 3 stage configuration. Along with the network configuration there is a Connection Matrix which is useful in the recording and validity

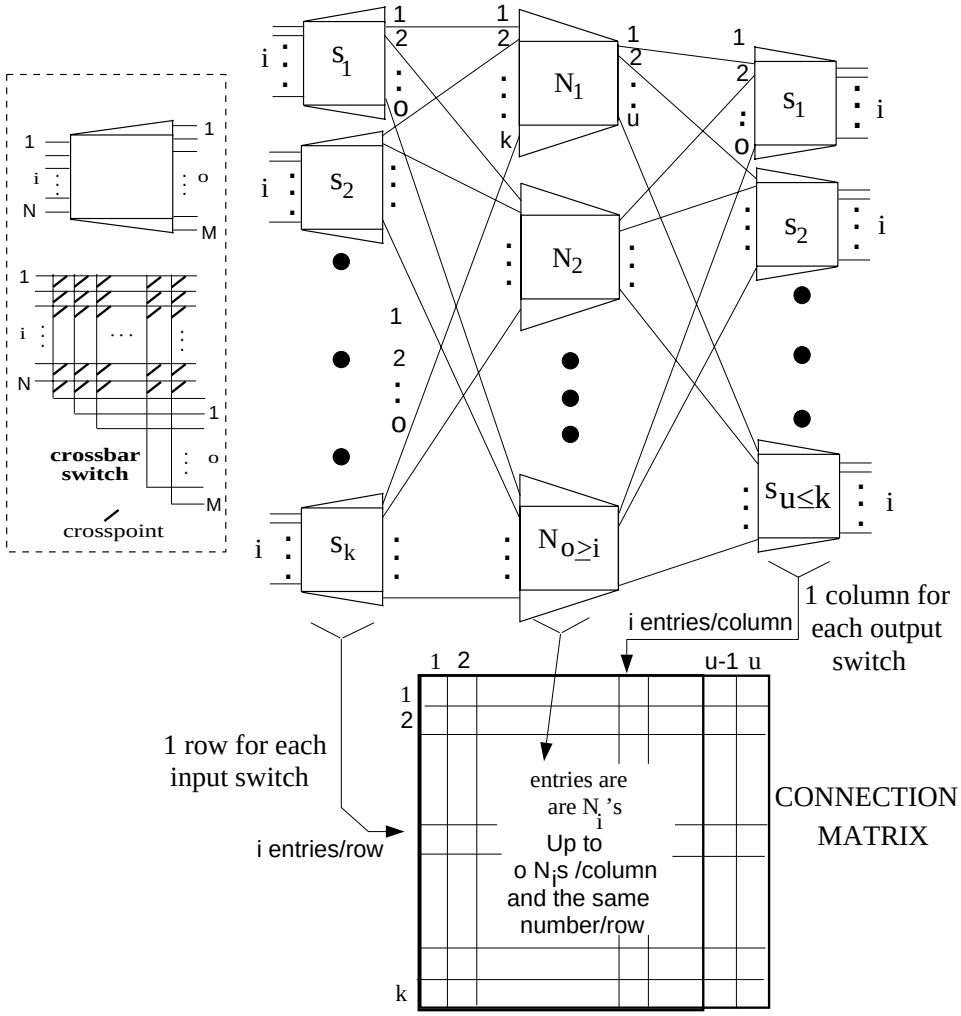


Figure 1-1: A Class Of Non-Blocking Networks Parameters: i, o, k, and u

analysis of the connections established in the configuration. The entire class is characterized by the following properties

1. Each $1st$ stage switch has one output to each of the o CSS.
2. Each $3rd$ stage switch has one input from each of the o CSS.
3. Each CSS has one output to each of the u $3rd$ stage switch.
4. Each CSS has one input from each of the k $1st$ stage switch.
5. The Connection Matrix has k rows and u columns and each connection from a network input switch, say in, to network output switch, say out, there is a CSS designation entered at a matrix position in, out. There are o such designations available, no two in any column or any row may be the same. There are at most i such entries in any one column or one row.

A particular network is determined from this schema by the setting of the parameters $i \leq o$, and $k \geq u$, and by the number of recursive application to the CSSs are carried out together with the settings of parameters for each such application. These parameters are:

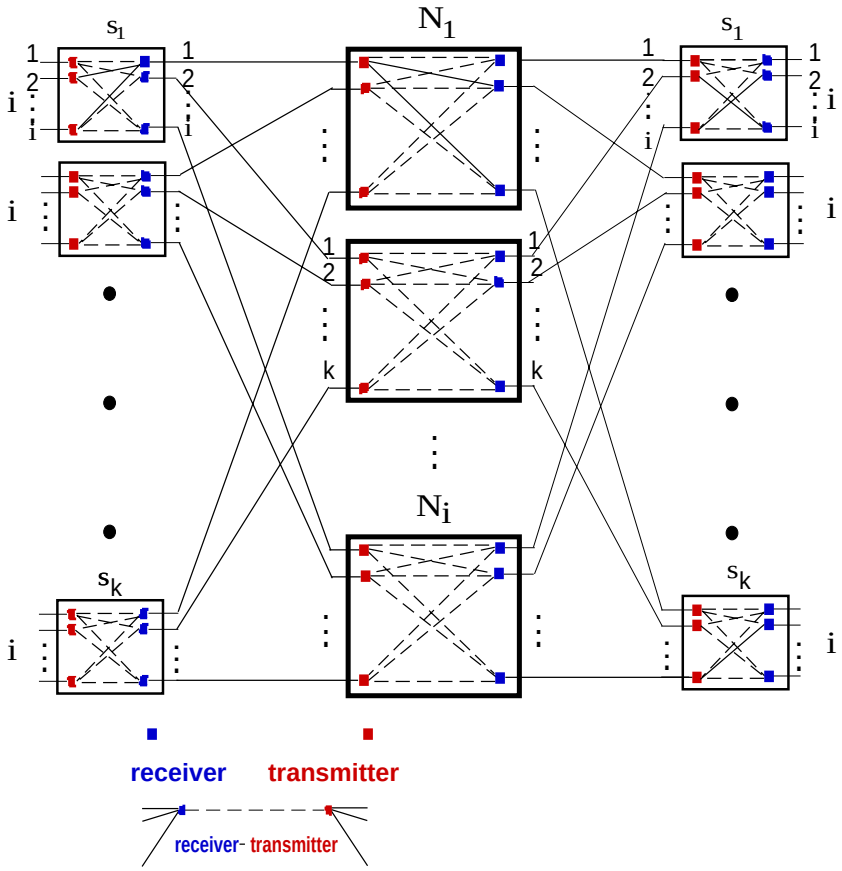


Figure 1-2: Store And Forward Network

1. i = the number of inputs to each $1st$ stage switch
   = the number of outputs of each $3rd$ stage switch
   = the maximum number of entries of CSS designations allowed in a row/column in the Connection Matrix.

2. o = the number of outputs of each $1st$ stage switch
   = the number of CSSs
   = the number of inputs to each $3rd$ stage switch
   = the number of CSS designation available to be entered in the Connection Matrix.

3. k = the number of $1st$ stage switches
   = the number of inputs to each CSS
   = the number of rows in the Connection Matrix.

4. u = the number of $3rd$ stage switches
   = the number of outputs of each CSS
   = the number of columns in the Connection Matrix.

Two extreme subclasses of these networks are shown in figure 1-3 For both of these networks the total number of inputs is ik. Also $u = k$ and each CSS has the same number of inputs as outputs. In summary in the (a) network $u = k = i = o$. This is a the simplest class of rearrangeable non-blocking networks. In figure 1-4 the (a) network is the also a simplest class of rearrangeable

non-blocking network, but in this case $u < k$ and the total number of outputs $ik < iu$. The (b) network in both figures is a schema for strictly or pure non-blocking network, in which $o = 2i - 1$, and $k = u$ In figure 1-3 (b) the total number of network inputs = the total number of outputs = ik. There are the same total number of inputs in figure 1-4 (b), but the number of outputs is less. That total is = iu, with $u < k$.

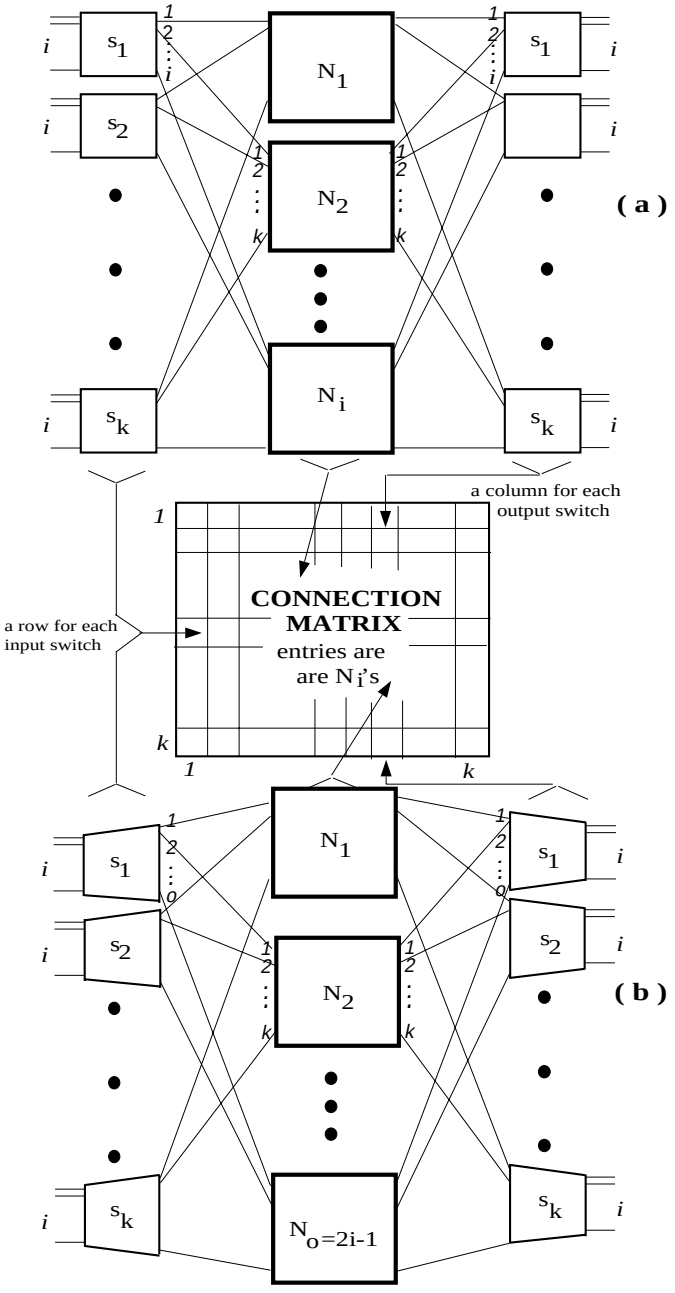


Figure 1-3: Classical Connection Networks N X N

An Example

Network definitions are illustrated in the rearrangeable network in figure 1-5.

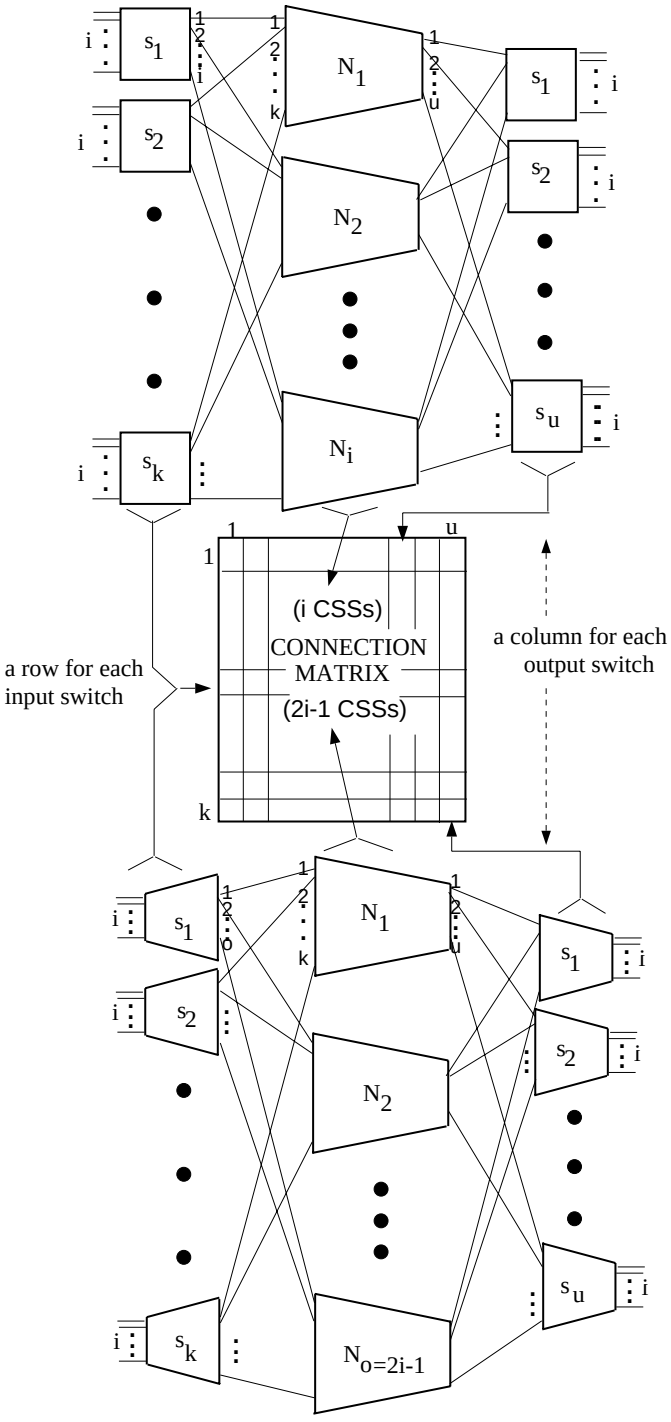


Figure 1-4: More Classical Networks N X M, N > M

1. Total number of inputs and outputs, $N = M = 8$
2. Size of first, last stage, switches, number of CSS*s*, $i = o = 2$
3. Number of first, last stage, switches, number of inputs-outputs to each CSS, $k = 4$.
4. Designation of CSS*s* A and B.

The CSS*s* A and B are themselves recursively constructed of 3 stages of switches. The CSS*s* in the subnetwork A are A1 and A2, and in B, B1 and B2. In A as well as B, $N = M = 4$, $i = o = k = 2$. The network then finally consist of 5 stages each with 4 2–*by*–2 switches.

The network's 8 inputs and outputs are each designated, top to bottom, with 000 through 111. The 4 input and output switches in the 1*st* and *last* stages of the network are designated 00 through 11. The network connection matrix below the network in the figure has a row and a column respectively for each first and each last stage switch. This connection matrix shows a B in position <00, 00> and an A at position <10, 01> indicating that there are connections from input switch 00 to output switch 00 through CSS B, and another from input switch 10 to output switch 01 through CSS A. The exact input, 000, and output involved in the connection through B is not recorded in the matrix. The information in the matrix, given the interstate wiring configuration, is sufficient to indicate that to compete the connection through B a connection from the topmost, input, on its topmost input switch, 0, in the 1*st* stage in B, to the 0*th* output switch in B, must be completed. As shown in the connection matrix on the right this uses CSS B2 of subnetwork B. switch.

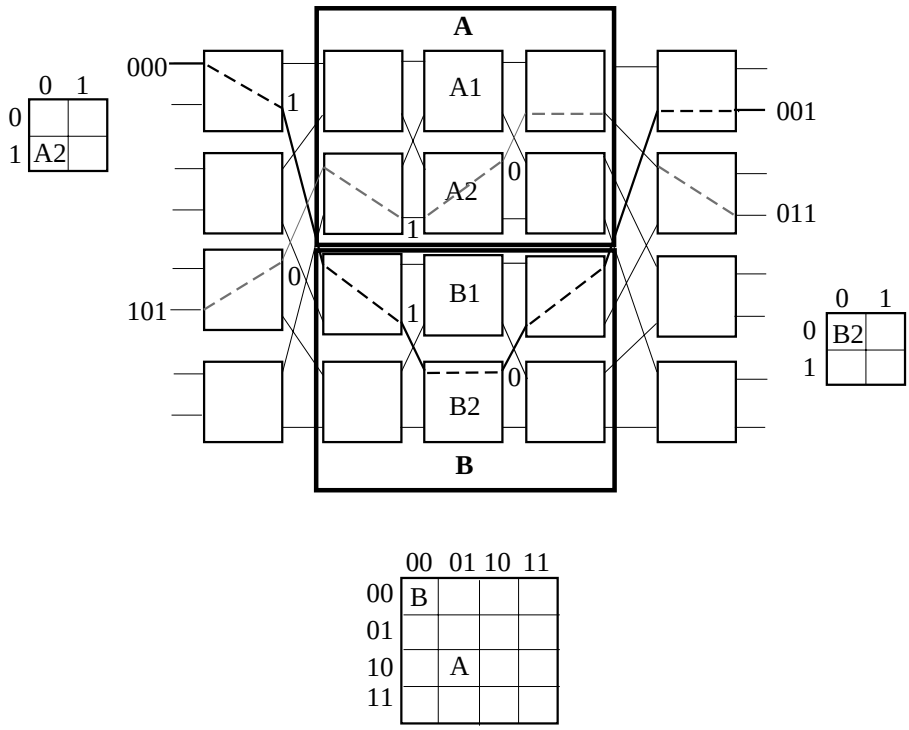


Figure 1-5: Connection Patterns: Connection Matrices At Two Levels

Now returning to the general situation. Consider a connection to be completed between an input *I*, on switch s(*I*), and an output *O*, on output switch s(*O*). At the row and column, identified respectively with s(*I*) and s(*O*), a mark is made. Each mark identifies a CSS (N_j) to which *I*, through s(*I*) will be connected on the left, and *O* through s(*O*) will be connected on the right by closing the appropriate crosspoints in s(*I*) and s(*O*). There are constraints on these assignments. There are at most *i* different CSS*s* in each row and in each column (there are *i* inputs and *i* outputs per input and output switch respectively), but there are as many as *o* different CSS*s* available to ($\geq i$). No two CSS*s* in any row or in any column can be the same because each input and each output switch

have one connection to each CSS*s*.

The question then is: can the i entries in every row and every column of the matrix, be assigned CSS*s* that will satisfy the constraint that no pair of CSS*s* in any row or any column will be the same? Consider two ways to make these assignments:

If the assignments are made systematically no rearrangement is necessary, certainly for the case that $i = o = 2$. First assign all instances requiring 2 connections, both in the same table entry. No other connections are possible in the row and column of such entries. Then choose a remaining row and assign two different CSS*s* there. This will typically force other assignments in columns of the assigned connections. Make all such forced connections so as to meet the constraints of two different CSS*s* in each row and each column. If there are still entries to be made an un-assigned row is chosen and the process of the previous sentences repeated, etc.

The second approach can be used for assigning CSS*s* one at a time in an arbitrary order--it generally requires redoing earlier assignments to make a new assignment possible and is unfortunately the more realistic model. Given the i entries in every row and column we proceed to assign CSS*s* in any way we like taking care not to violate the one/column-one/row rule, until we find a position at which we cannot assign a CSS*s* without violating that rule. What does such a conflict look like?

There is a **violation** if it is necessary to assign a CSS to position <r_0,c_0>, called the **initial target**, and all o CSS*s* already are distributed in r_0 and c_0. In either r_0 or c_0 there can be at most $i - 1$ different CSS*s*. There are o different ones available. So, if $o = 2(i - 1) + 1$ there must be enough CSS*s* available so no such violation can occur--there are always enough CSS*s* to assign to each entry without violations or rearrangement. The network in figure 1-3 (b) with $o = 2i + 1$ is such a strictly non-blocking classical network, discovered by Clos.

In all cases of N-by-N classical networks in which $2i - 2 \geq o \geq i$ the violation can occur. However, in all these cases, with <r_0, c_0> being the initial target, there is at least one CSS which appears in c_0 but not in r_0, say A, and another which appears in r_0 but not in c_0, say B. To show this:

Let the number of CSS*s* in r_0 be N_r, the number in c_0 be N_c, and the number which are in both r_0 and c_0 be N_{rc}.

$N_c \leq i - 1$ and $N_r \leq i - 1$ since there can not be as many as i in either r_0 or c_0 or <r_0,c_0> could not in any case sustain an entry.

$N_c + N_r - N_{rc} = o$ = the total number of different CSS*s* altogether in r_0 and c_0. If there were fewer there would be a CSS available for <r_0, c_0> not in either. r_0 or c_0. So by subtraction $N_r - N_{rc} \geq o - i + 1$ and $N_c - N_{rc} \geq o - i + 1$.

These are respectively the number of CSS*s* in r_0-not in c_0, and the number in c_0-not in r_0. If $o = i$ then this number is ≥ 1, and if $o = 2i - 2$ it is $i - 1$. Notice that if $o = 2i - 1$, the strictly non-blocking case, this number is i, which is impossible if an entry in <r_0,c_0> is to be possible.

In summary the number of CSS*s* in r_0 not in c_0, and the number in c_0 not in r_0 are each $\geq 1 \leq i - 1$

Q.E.D.

So there is a violation at <r_0, c_0>, and an A is in c_0, at r_1, and not in r_0, and a B is in r_0 and not in c_0. Our strategy is to put A in <r_0, c_0>, and try to remove duplications that result. (An alternative

strategy in which the initial target is replaced by B thus creating a conflict in r_0 follows an analogous course and is just as viable.)

Change the A at <r_1, c_0> to a B and make this B the target.

Now there may already be a B in the target row r_1 at c_1.

Then change that B target to an A and make it the target.

Now there may already be an A in the target column c_1 at r_2.

etc.

This strategy continues as long as the changing target conflicts with a duplicate. Will it end? Figure 1-6 illustrates the outcome of this strategy by showing the connection matrix initially and after the first and final step of the algorithm, for a few different initial states.

The path of changes which are made by that procedure can not reenter a row or column in which there has already been a change because that implies that there exists a duplicate in one of these columns or rows. (Assume the contrary. Then the first change to occur in a column or row already processed, say a row, r, is not at the points that the entries of A and B had already been placed. So that point, p, is somewhere else in r. Since there is an A or B at p there must have been more than one A, or more than one B in that row. This can not be the case when the process leaves a row. Figure 1-6 illustrates this fact.) So new rows and columns are visited until there is no conflict with the target. Notice that the second change in a row is because the first change was from A to B so the second change in the row, if there is one changes a B. But there can be no B in any row in the column in which the B in r_0 is located. So the rearrangement must terminate after 1 change in c_0, and at most 2 changes in every column except the one which has B in the r_0. This means that there can be changes (2) in every column except c_0, and that containing in r_0 a total of $2k-3$ (not counting the entry of A at <r_0, c_0>.)

Note: Assuming there were actually $2k-3$ changes with the initial target A, then if B instead of A were the initial target in <r_0, c_0> then only 1 change would be needed. In general when $i = 0$ there are two alternative ways of doing the rearrangements, and the sum of the number of changes necessary is no greater than $2k-2$. So the largest number of changes is not more than $k-1$. When $o > i$ this number may be decreased, it certainly is when $o = 2i - 1$.

1.1.0.1. The Number Of Stages, And Cost Of The Classical Networks

This development is valid for figure 1-3 where $o \geq i$.

Number Of Stages

Consider the cost when $o \geq i$ and $k = i^{n-1}$ and the CSS*s* are recursively replaced so that finally all switches are *i–by–i*. Assume that the total number of inputs to the network is a power of i, that is $N = i^n$.

Stages(i^n) = the number of stages in networks of with i^n inputs.

Certainly:

Stages(i^1) = 1 a single crossbar switch

And the recursive structure of the network gives:

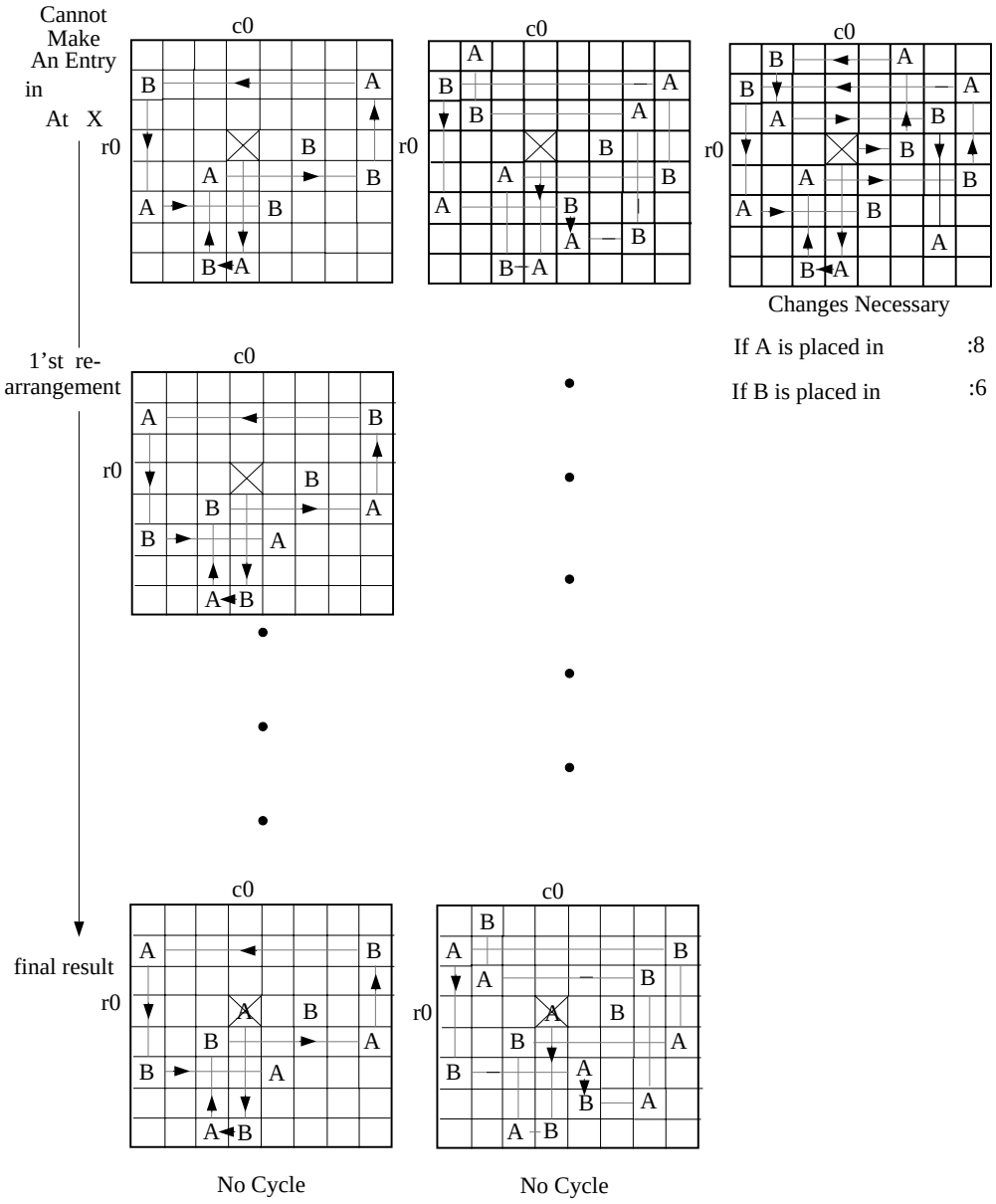


Figure 1-6: Representation Of Rearrangeability

The number of stages for a network with i^n inputs and outputs equals the number of stages of a network with i^{n-1}(the CSS*s*) + 2(the 1*st* and *last* stages)., i.e.:

$Stages(i^n) = Stages(i^{n-1}) + 2$

This is satisfied by

$Stages(i^n) = 2n - 1 = 2log_i N - 1$

Cost In Crosspoints

when $o = i$

When the CSS*s* are recursively replaced so that finally all switches are *i–by–i* it is easily

verified that *every stage has the same number of* i*–by–*i *switches* as the first stage. So every stage costs i^{n+1} and from above there are $2n-1$ stages. So the total cost is:

$$cost(i^n) = cost(N) = i^{n+1}(2n-1) = 2\,i\,N\log_i N - i\,N$$

when $o \geq i$

More generally when the CSS*s* are recursively replaced so that finally all switches are i*–by–*i.

$$Cost(i^n) = 2\,i^n\,o\,(1 + (o/i) + (o/i)^2 \cdots (o/i)^{n-2}) + o^n\,i$$

$$Cost(i^n) = 2\,i^n\,o\,(((o/i)^{n-1} - 1)/((o/i) - 1)) + o^n\,i$$

$$Cost(i^n) = Cost(N) = 2\,N\,o\,(((o/i)^{n-1} - 1)/((o/i) - 1)) + o^n\,i$$

and when $i = 2$ $o = 3$

$$2\,N\,3\,(((3/2)^{n-1} - 1)/((3/2) - 1)) + 3^n\,i$$

$$6\,N((1.5^{n-1} - 1)/.5) + 2^n(1.5^n)2$$

$$(8\,N[2^{\log_2 1.5}]^n) - 12\,N + 2\,N[2^{\log_2 1.5}]^n$$

$$12\,N(2^n)^{\log_2 1.5})/1.5 - 12\,N + 2\,N(2^n)^{\log_2 1.5}$$

$$8\,N\,N^{.585} - 12\,N + 2\,N\,N^{.585}$$

$$10\,N^{1.585} - 12\,N$$

1.1.0.2. Path Algorithm For Benes And Other Classical Networks

It is possible to do establish the path in a Benes Network with a sequential algorithm in time $O(\log N)$, provided the path is not blocked. Suppose the input and output switches are numbered 0 through $N-1 = 2^n - 1$, and a connection is to be made from input i to output o. The path is developed in two parts: as a left to right, *input path* from i to a central stage switch, and from o as a right to left *output path* to the same central stage switch. The paths enter switches and emerge from either the 0 (upper) exit or 1(lower) exit. The $0\,1$ exit is on the right of each switch in the input path and on the left in the output path. The left and right paths must start through their respective first switches, L_1 and R_1, with the same exits, either both 0 or both 1. This follows from the requirement in the connection matrix associated with the recursive formulation of the Benes Network that, at the intersection of the row and column representing respectively the first switches of an input and output path pair, there is an entry of a CSS. This implies that input and output paths take the same exit from L_1 and R_1. This will only be possible without disturbing existing paths if the existing connections in L_1 and R_1 are not to different CSS, for then there would be no exit that was unused in both. As long as there is no such blocking the paths can be developed further. Let L_j and R_j respectively be the $j\,th$ switch on the input and output path. It follows from the recursive definition of the Benes Network that, in order to complete them, the paths must take the same exits from L_j and R_j. This is only possible if these switches are not blocked.

An analogous algorithm holds for all the classical networks, including the Clos Network in which there is guaranteed to be no blocking and for which consequently the algorithm always produces a path.

The complexity of the algorithm is proportional to the number of stages, $O(\log N)$, in the path since it need only make the constant time determination of an available common exit in L_j and R_j.

The algorithm requires information of exits used in every switch. It could be implemented within each switch if communication was established between the input and output paths as they progress through each switch to establish the common exits.

1.1.0.3. Small Variations

In rearranging the entries in the connection matrix, it is always possible to keep the entries in one row/column fixed and still make any desired connection from an input to an output switch. Therefore the inputs to one of the $1st/3rd$ switches may be connected directly (hard wired) to the CSSs, eliminating that $1st/3rd$ stage switch to which they were connected. This gives a slight savings in crosspoints if the eliminated switch is small and may increase the number of rearrangements in a rearrangements needed to free up a connection. In fact the number of rearrangements necessary for a connection to the hard wired lines can require $2n - 2$ such rearrangements--even though without the switch removal $n - 1$.

The maximum number of rearrangements required are determined by the dimensions of the connection matrix, that is by the number of $1st$ and $3rd$ stage switches. If there are N of the former M of the latter, $N > M$ it is the smaller M that determines the maximum number of rearrangements.

Generally, then it is not the total number of network inputs or outputs that determines the maximum number of rearrangements needed, since a few many-input $1st/3rd$ stage switches can handle the same total number of network inputs.

A network which illustrates these points is shown if figure 1-7. In this network at most $2 = 2(2)–2$ rearrangements are necessary to make an initially blocked connection, one of the $1st$ stage switches can be removed as shown and if one wants to recurse to make a network with more stages of smaller switches one ought to do so on the $1st$ and $3rd$ stage switches.

1.1.1. Banyan And Omega Networks And The Perfect Shuffle

For passing information between memories and processors a connection network ought, at least, to be able to connect each single input to any single output. The **Benes** network which is that in figure 1-3 with $i = o = 2$, is the basis for an efficient network satisfying these minimal requirements. It is obtained by using the first $log_i n$ = the first half + 1 of the stages of the Benes network. Viewed left to right this is a **Banyan** network, and right to left it is an **Omega** network. The properties of shown now for the Banyan also for the Omega network.

These network have the minimal property and an elegant addressing scheme, but support only a small fraction of all possible permutations.

In figure 1-8 (a) a completely expanded network with $2–by–2$ switches and 2^3 input and outputs is shown with two paths (one with lines). It is the one developed in 1-5. The two outputs from each $2–by–2$ switch are labeled 0 and 1. The interconnections between the $3rd$ and $4th$ stages have been lengthened and labeled from 000 to 111 because the paths to be considered now are those from the input on the left and to these lengthened lines. This illustrates that to get to the Banyan output $b_1 b_2 b_3$from *any* input one need only connect that input to the b_1 output of its $1st$ stage switch, then to the b_2 output of the $2nd$ stage switch and finally to the b_3 output of stage 3. In particular, on one of the darkened paths individual switch settings of 1 in the first stage, 1 in the second stage, and 0 in the third stage establish a path to the the lengthened lines labeled 110. This path starts at the uppermost network input, but the same switch settings starting at any of the

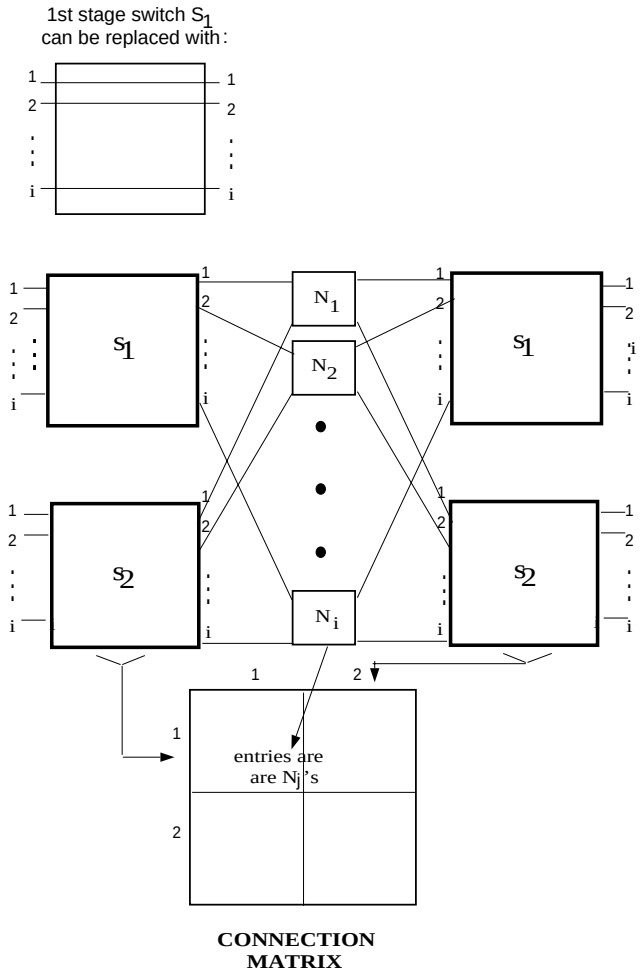


Figure 1-7: Many Network Inputs/Outputs With a 2X2 Connection Matrix

network inputs would establish a path to the same point.

In part (b) of figure 1-8 all paths from one Banyan input to all outputs are shown with darkened lines. These form a tree and such a tree would be found tracing all paths from any of the inputs. This is the reason the elegant addressing scheme exists.

In general, for Benes network, a connection through the network can be completely specified by giving the position, 0 or 1 or... or $i-1$ (0 or 1, for the example) on the output of the switch at each stage in its path in the network. There is a symmetry in this specification because the connection matrix shows that each choice of a CSS, N_i, effectively determines two switch settings the same distance from the input and output. It is a mirror symmetry. The settings of the first r switches listed in order are p_i, $i = 1, 2, \ldots, r$, with p_i taking a value from 0 to $i-1$. If r is 1, there are, from each input, i different paths to the output of a stage 1 switch specified by $<p_1>$. If r is 2 then there are i^2 possible paths to the output of a stage 2 switch specified by $<p_1, p_2>$. If r is $log_i N$ then $i^{log_i} = N$ different positions at the output of stage $log_i N$ that can be specified for each input. This is a sufficient specification since N the number of outputs of the Benes network. Since the total number of stages of that network is $2log_i N - 1$, there is a path available from any input to any output using only the first one-half stages $+1$ (the center stage) of a Benes and there is also an elegant

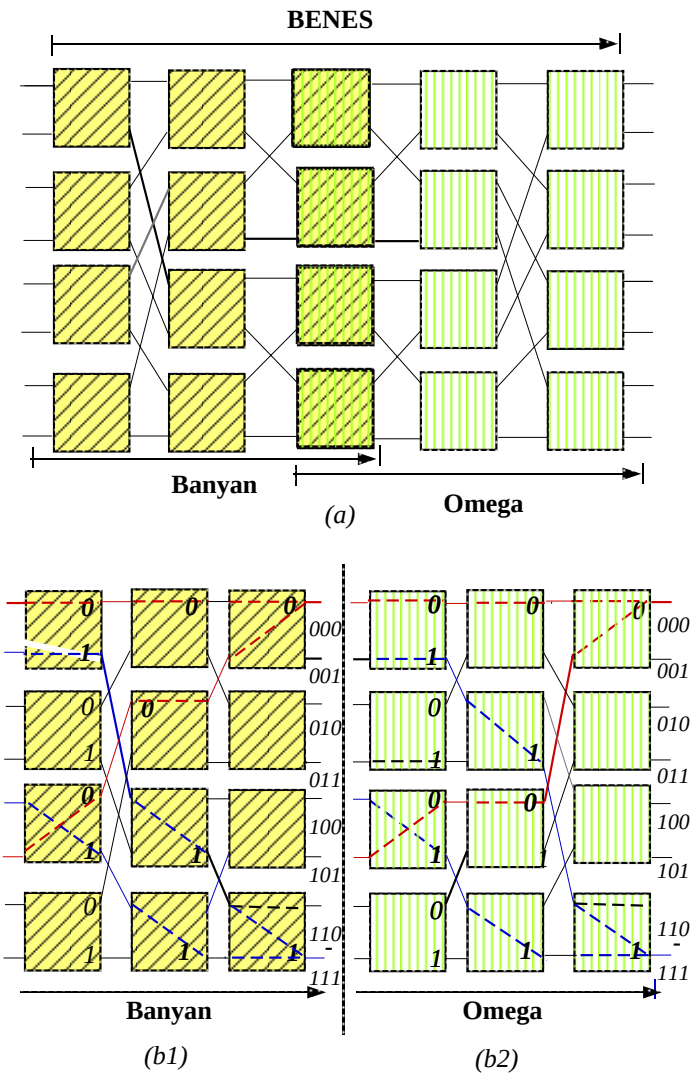


Figure 1-8: Banyan Network Addressing

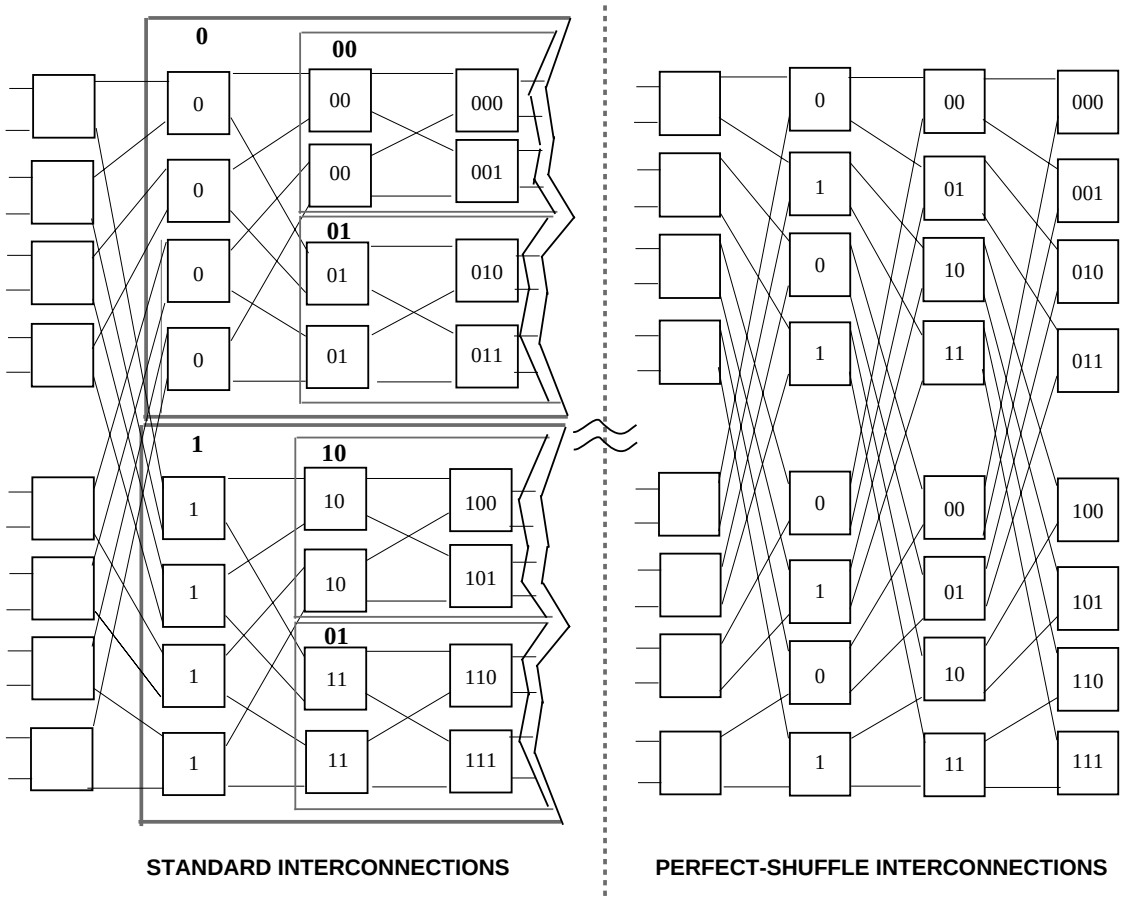
addressing scheme. Actually using the first half + 1 stage of any of the networks in figure 1-3 an analogous addressing scheme exists: for $i = 0 > 2$, the base of the addresses simply becomes i.

The Banyan type network is easily blocked, for example it is impossible to connect any two inputs to the same input switch to the two outputs of any single output switch.

Banyan Path Algorithm

If the destination of an input to a Banyan, is sent along with the information in the input, that address can be used to guide the information to its destination (or discover the impossibility to reach that destination--Banyan is easily blocked). This allows the connection paths to be computed in a time proportional to the number of stages of the network ($O(\log n)$).

1.1.1.1. Perfect Shuffle



FIRST HALF + ONE BENES NETWORK

Figure 1-9: Example Of The Benes And Corresponding Perfect Shuffle

The Benes networks has N inputs and outputs to and from each stage and there are $N/2$ $2X2$ switches per stage.

With $i = o = 2$, and k a power of 2, the network schema in figure 1-3 (a) is the recursive definition for the Benes. When the CSSs are filled by 3 stage schemas again and again until all CSSs remaining are $2X2$ switches the result is a $2k$ input-output Benes network. As for all the networks in this class, the outputs of the $1st$ stage and the inputs to the last stage switches are connected to central subnetworks. As long as the connection are to the correct subnetwork they may be made to any input/output switches within the subnetwork, without alteration of the properties shown to hold for these networks. So a Benes networks, when fully expanded, may be pictured many ways, all topologically and thus functionally equivalent.

In order to have the properties including rearrangeability and Banyan addressing through the first $logN$ stages ascribed to them, it is necessary and sufficient that their connectivity satisfy the following conditions:

The Defining Properties Of The Benes Network

The recursive definition of the Benes Network has 2 CSSs. These are labeled 0 and 1. Then the $0th$ CSS is expanded according to the recursive definition--it has 2 CSSs, and these are labeled 00 and 01. Similarly in group 1, the CSSs are labeled 10 and 11. The first stage switches in the $0th$ CSSs are also labeled 0, and the first stage switches in CSS 1 are labeled 1. In general a $1st$ stage switch in a CSS gets the same label as that CSS.

Switches in the $1st$ stage of the network have one output to each $2nd$ stage switch of the network labeled 0, and one to each of those with label 1. Note that the $2nd$ stage of the network is the $1st$ stage of a CSS.

Each $2nd$ stage switch labeled ξ (where $\xi = 0$ or 1) has an output to one $3rd$ stage CSS labeled $\xi 0$, and one to a third stage CSS labeled $\xi 1$.

Generally each kth stage switch labeled ξ, where ξ is a $k-1$ bit binary number, has an output to one $k+1st$ stage switch labeled $\xi 0$, and to one labeled $\xi 1$.

The number of switches in a stage is a power of 2, $N = 2^r$. So in the second stage there are 2^{r-1} switches labeled 0 and the same number labeled 1. In the $3rd$ stage the switches are labeled $00, 01, 10, 11$ and there are 2^{r-2} with each label. In the kth stage the switches are labeled with each $k-1$ bit binary numbers and there are s^{r-k+1} with each label. Assume these label conditions are true up to the $k-1st$ stage. In the kth stage each switch is labeled with a $k-1$ bit binary numbers since the labels of this stage are formed from those of the $k-1th$ stage where each switch is labeled with $k-2$ bit number, by concatenating a 0 or a 1. Also there are in stage k there are 2^{r-k+1} switches with the same label since 2^{r-k} in the $k-1st$ stage. So there are now twice as many labels but still the same number of switches.

The Perfect Shuffle Connection Between Stages

As will be shown, there is a single connection pattern, the **Perfect Shuffle**, that satisfies the Benes properties between each pair of stages from 1 to $log_2 N$, and thus is applicable to a Banyan network, i.e., allows efficient determination of which switches to close to make any input-output connection. With the uniformity of interconnections in these stages the perfect shuffle network is iterative which is an advantage for fabrication, and also allows time-reduction along the horizontal axis, which means one copy of a stage with connections back to itself through a delay can simulate the Benes-Banyan network. The same argument can be applied to the last $log_2 N$ stages of the Benes to show they two can also have the same interconnection pattern between stages, but this is not the perfect shuffle--it is its reverse. Thus a second perfect shuffle network could perform as the second half of the Benes Network--which together with that perfect shuffle version of the first half could act as the Benes Network.

The first through $(n/2)th$ output in stage k are connect successively to each switch in stage $k+1$ (output i to switch i). And similarly the $(n/2+1$th$\}$ to nth outputs go to the first through $n/2th$ switch in stage $k+1$. (the $(n/2+i)th$ output to the switch i).

Figure 1-9 shows the first four stages of a 16 input Benes network with the standard geometry of interconnections between stages, and on its left, the same four stages with the same topology but different geometry of interconnections between stages. The interconnections between every pair of stages on the left has the same interconnection geometry. This geometry is that of the Perfect Shuffle. The network is labeled to show that it satisfies the Benes property.

Consider the first stage of a perfect shuffle network. Each switch has connections to two

adjacent $2nd$ stage switches so if the $2nd$ stage switches are numbered alternately 0 and 1, then each $1st$ stage switch has one output to the 0 group and 1 to the 1 group, satisfying the Benes property. Now the each $2nd$ stage switch again has connections to two successive $3rd$ stage switch so if the switches are labeled in sequence 00, 01 -- having inputs from the first switch in group 0 of stage 2 -- then 10, 11, having inputs from the second stage 2 switch labeled 1. Subsequent switches are named in sequence $00, 01, 10, 11$, and that pattern is repeated till all $3rd$ stage switches are exhausted, and thus this set of interstage connections has the Benes property. In general let the kth stage switches be labeled with $k-1$ bit binary numbers in sequence, with the sequence repeated as many times as necessary to exhaust all kth stage switches, and the $k+1st$ stage switches be labeled with k bit binary numbers in sequence, with the sequence repeated as many times as necessary to exhaust all $k+1th$ stage switches. Now the perfect shuffle results in the first switch in stage k, labeled [. . . 0], ([. . .] is a sequence of length $k-1$) having connections to first two switches, labeled [0 . . . 0]0 and [0 . . . 0]1, and in general with the switch labeled [ξ] of stage k being connected to the switches [ξ]0 and [ξ]1 of stage $k+1$. Thus the connection between stages k and $k+1$ has the Benes property.

See figure 1-10 for an illustration of how the perfect shuffle exhibits the Benes property.

Number the left side of the perfect shuffle from 0 at the top to 2^n-1 and the right side similarly. The perfect shuffle then connects left side numbered i to right side numbered $2i$ for $i=0$ to $2^{n-1}-1$ and the left side numbered $2^{n-1}+j$ to the right side numbered $2j+1$ for $j=0$ to $2^{n-1}-1$.

It follows that the Perfect Shuffle Network allows Banyan addressing.

1.2. Sorting Networks And Self Steering Connection

A hardware sorter with n distinct inputs, and $n|$ distinct outputs, when presented at its inputs with the integers 1 through n in any order will send 1 to the first output, 2 to the second output, and so on. A message may be attached to each the integers 1 through n at the input. The sorter inputs then consist of two parts, the *address part*, (the integers) which establish a path for each second or *messagepart* through the network. At each *min–max* block the address part determines to which output of that block the address part, and now also the associated message part will exit the block. This establish a path through the network along which the message travels from the sorting network input to the addressed output through the sorter. So a hardware sorter of this type can in principal act as a self-steering connection network. It differs from a simpler connection network in that it requires that all possible output addresses (1 through n) appear at its input. If an address is missing it will typically either have a default of 0 or ∞ which will result in the specified input addresses, sorted in with the default value, going to other than the addressed outputs.

1.2.0.1. Basic Theorem On Sorters Built Using Min-Max Comparators

Let the set of n inputs be X.

Let $max(a,b) = a+b$

Let $min(a,b) = a \cdot b$

These functions are completely analogous to the functions "or" (+) and "and" (·) in propositional logic.

This is suggested by the behavior of *min* and *max* on 0 and 1, which is identical to that for **or**

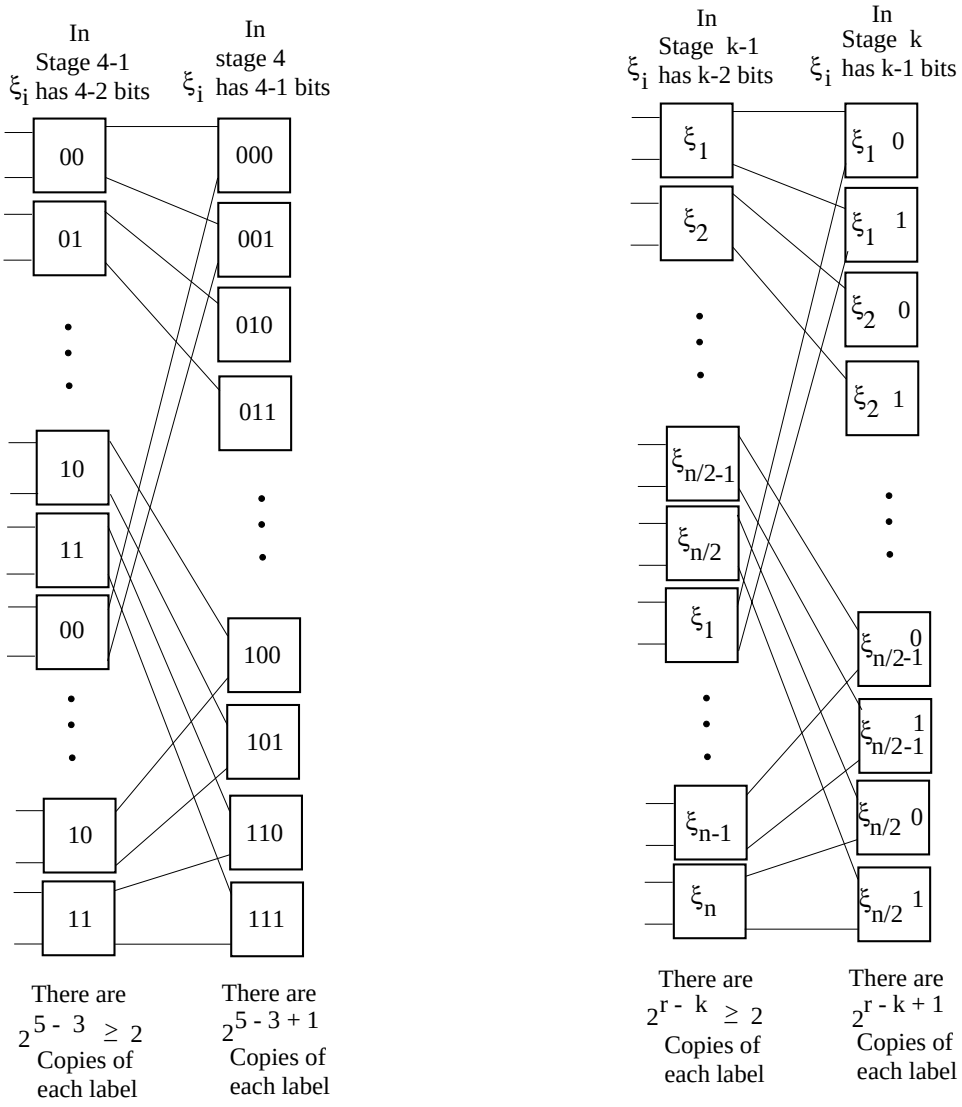


Figure 1-10: The Perfect Shuffle Has the Benes Property

and **and**.

a b	*min*	*max*
00	0	0
01	0	1
10	0	1
11	1	1

Figure 1-11: Minimum And Maximum Functions

It follows that all the associative, distributive, etc. laws which apply to **or** and **and** also apply to *min* and *max* as long as variables are only $0\,s$ and $1\,s$.

Therefore any *min–max* function has a disjunctive canonical form. These form a subset of all propositional logic canonical forms--being only those from which all negations can be removed by simplification. In any case it follows that the number of different *min–max* functions of n variables is finite, and different functions must be distinguished by their behavior on all possible combinations of $1s$ and $0s$. They must have different truth tables. Only the set of unique *min–max* functions on n variables, with one for the smallest output, one for the next to smallest, etc. will sort every input of n $0s$ and $1s$. (For example: the *min–max* function giving the kth highest input, and that giving the $k+\varepsilon th$ $\varepsilon \geq 1$ highest behave differently on every 0–1 with exactly k $1s$.

So:

Theorem:

If it is possible to build a sorter which consists of a network of comparators (it is), then a given comparator network will sort any numerical inputs iff it sorts all permutations of 0 *and* 1.

It is possible to build such a sorter. We have previously shown sorting networks made only of mins and maxs --so there is a a min-max expression on n variables for the kth highest, $k \geq 1 \leq n$. If all n inputs (variable values) are 1 or 0,in fact, the kth largest output is the maximum of the minimums of all combinations of k variables. This is so because the kth largest is 1 iff there is at least one combination of k variables all of whom are 1. The same expression then must give the kth largest if the inputs can take on any values.

1.2.1. ODD-EVEN MERGE SORT

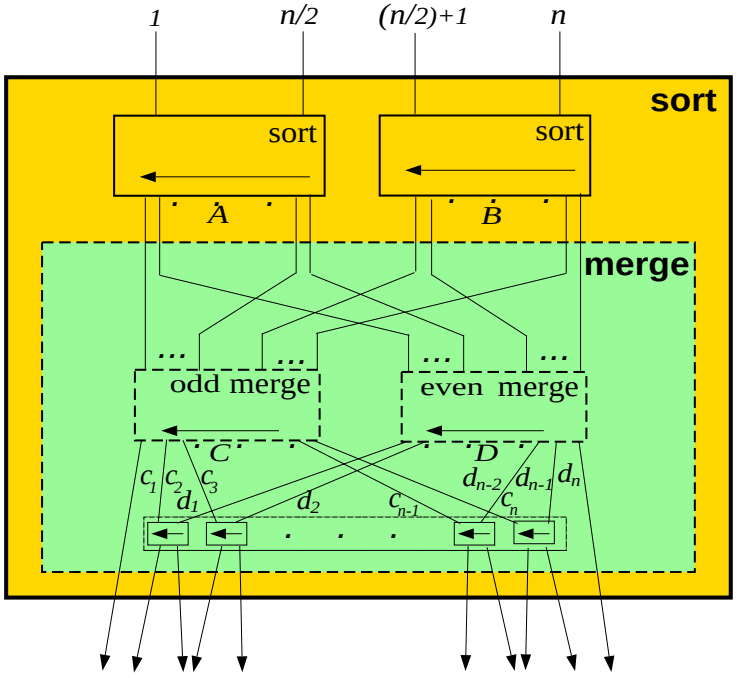


Figure 1-12: Odd-Even Merge Sorter

At the top of figure 1-12 the structure of ODD-EVEN merge sorter with n inputs to a sorter and

merger is pictured recursively, based on the existence of sorters and mergers with $n/2$ inputs. The same information is given algebraically below.

Definitions

$sort_{oe}(X)$ is the **Odd-Even merge sort** of the set of numbers X

$sort_{oe}(X)$ is the output of a *min–max* function with inputs X_1 and X_2 where $X = \{X_1, X_2\}$

$merge_{oe}(X,Y)$ is the **Odd-Even merge** of two sorted sets X and Y. This merge has the usual result, two sorted lists are joined in one sorted list of all their elements. The way it accomplishes that however is not in the usual way.

$merge_{oe}(X,Y)$ is the output of a *min–max* function with inputs X_1 and Y_1 and $X = \{X_1\}$ and $Y = \{Y_1\}$

$odd(<x_1, x_2, \ldots, x_n>) = <x_1, x_3 \ldots, x_{n \text{ or } n-1}>$

$even(<x_1, x_2, \ldots, x_n>) = <x_2, x_4, \ldots, x_{n \text{ or } n-1}>$

Then to sort the list consisting of every element in A followed by every element in B, called AB.

$sort_{oe}(AB) = merge_{oe}(sort_{oe}(A), sort_{oe}(B))$

where this odd even merge is accomplished as follows:

$merge_{oe}(sort_{oe}(A), sort_{oe}(B)) =$

$w(merge_{oe}(odd(sort_{oe}(A)), odd(sort_{oe}(B))), merge_{oe}(even(sort_{oe}(A)), even(B')))$

Let C and D be sorted sequences and

Let C and D be sorted sequences and

$w(C, D) = E = <e_1, e_2, \ldots \ldots, e_n>$ where e_i is defined as follows:

$e_1 = c_1$,
and
for $1 < i < n$
$e_i = min(c_i, d_{i-1})$
$e_{i+1} = max(c_i, d_{i-1})$
end
and
$e_n = d_n$.

A key part of the proof is to show that the outputs of the comparators which each receive one input from the odd outputs of A and B merged, and one from the even outputs of A and B merged is in sorted order as they emerge at the bottom of the ODD-EVEN sorter. Taking advantage of the 0–1, theorem we need only do this for inputs of 0 and 1 from A and B.

The number of $0s$ at the odd indexes of A, $odd(A)$, can be 1 more than those in $even(A)$, similarly for those in odd and $even(B)$.

The possibilities are illustrated in figure 1-13 at the top. Here these differences in the numbers $0's$ is seen to depend on the parity (odd or even) of the index of the last 0 of A and B.

So we have one of the following cases.

1. Number of $0s$ in $(odd(A) \cup odd(B)) = 2 +$ the number in $(even(A) \cup even(B))$.
2. Number of $0s$ in $(odd(A) \cup odd(B)) = 1 +$ the number in $(even(A) \cup even(B))$.

3. Number of $0s$ in $(odd(A) \cup odd(B))$ = in $(even(A) \cup even(B))$.

Figure 1-13 illustrates the proof of these results.

The sorted binary output $D = <d_1, d_2, \ldots, d_n>$ is shown below that of $C = <d_1, d_1, \ldots, d_n>$. D has been shifted one position to the right. The C and D outputs go pairwise to comparators. So the outputs e_{2i} and e_{i+1} are the outputs of the comparator whose two inputs are c_{i+1} and d_i. Thus the two inputs to each comparator are both 0 from $i = 1$ to $i = x$, then there is one comparator with a 0 and 1 input making $e_{2x} = 0$, $e_{2x+1} = 1$. Thereafter every comparator has two $1s$ for input. So clearly the output consists of $0s$ followed by $1s$. This is a sorted binary output.

The critical point is that at which the input to a comparator from the shifted D is 1 and that from C is 0. If there was more than one "critical" position at which the inputs to a comparator differed (this would be so if D were no shifted) then the output would have a 0 after the first 1.

As shown at the bottom of figure 1-13, when the difference in the number of $0s$ in C and is 1 there would be no comparator which received a 1 and 0 (with D shifted), and with the difference 0 there would again be only one comparator that received a 1 (from C) and 0 (from the shifted D).

1.2.2. BITONIC SORT

A finite sequence of numbers is **bitonic** if it

1. is non-decreasing
2. non-increasing

Or it is one of the above followed by the other.

The following are bitonic: $<1,3,5,9,4,2>$ and $<9,4,2,1,3,5>$

It follows that a sequence of $0s$ and $1s$ is bitonic if the $0s$ and $1s$ form contiguous subsequence, or can be given this form by a circular rotation, ex., $<0,0,1,1,1,0,0,0>$ or $<1,0,0,0,0,0,1,1>$ can be rotated to give $<0,0,0,0,0,1,1,1>$

Note that any sequence of length 1, 2 or 3 numbers is bitonic.

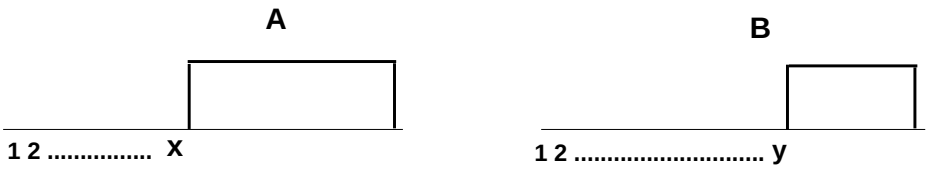
Before further description of the bitonic sorter it may be helpful to sort out some definitions.

Both a **sorter** and a **merger** have n inputs $i_1, \ldots, i_n$, and outputs, $o_1, \ldots, o_n$. The outputs from both have the property $o_1 \le o_2 \cdots \le o_n$. The range of possible input arrangements however differ. The sorter produces its ordered output no matter what the order relation between any two of its inputs. A **merger** only produces its ordered outputs if its inputs themselves have some ordering. For a standard merger the input relations must $i_1 \le i_2 \cdots \le i_{n/2}$, and independently $i_{n/2+1} \le i_2 \cdots \le i_n$, in which case the ordered output is guaranteed. For a **bitonic merger** the inputs running from i_1 to i_n must be a bitonic sequences for the ordered output to be guaranteed. (This is different than the two independent sorted lists which comprise the input to the usual merger.)

1.2.2.1. The Explanation

The explanation is inductive.

Assume the existence of a bitonic sorter and a bitonic merger, each with $n/2$ inputs. They form the basic blocks from which the n input bitonic sorter is constructed. (A 2-input sorter is realized with a single min–max block, as is A 2-input merger.)



x	y	# of 0's in C (odd merge)	# of 0's in D (even merge)	difference
even	even	x/2 + y/2	x/2 + y/2	0
even	odd	x/2 + (y-1)/2 + 1	x/2 + (y-1)/2	1
odd	even	(x-1)/2 + y/2 + 1	(x-1)/2 + y/2	1
odd	odd	(x-1)/2 + (y-1)/2 + 2	(x-1)/2 + (y-1)/2	2

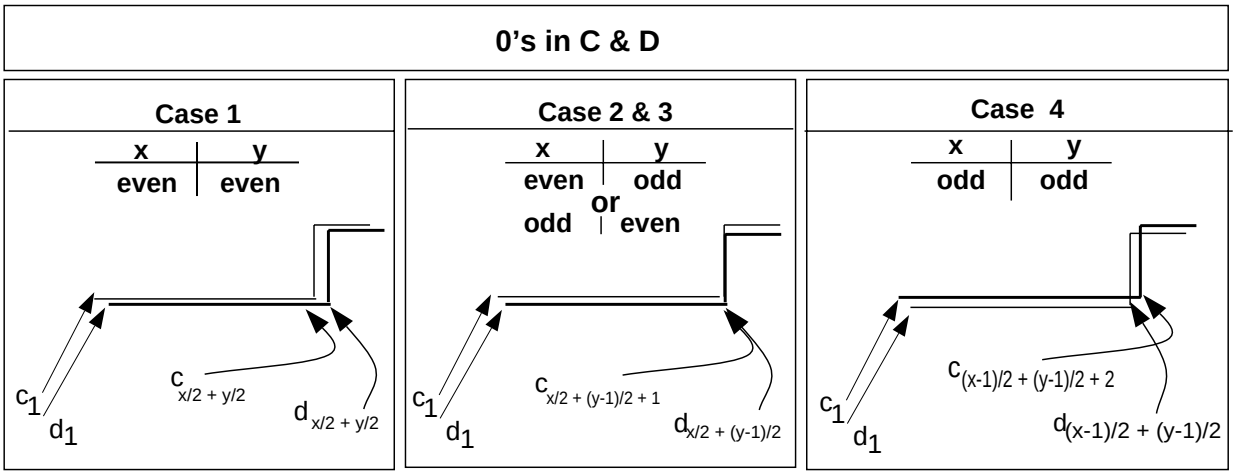


Figure 1-13: 0-1 Proof: Odd-Even Merge Sort

This construction is shown in figure 1-14. The sorters are in heavily lined block and the bitonic mergers in the dashed lines . The order of the outputs of the blocks is given by the arrows indicating the output values run from low at the tail to high at the head of the arrows.

By sorting both halfs of the input, and concatenating their outputs, the $A1$ and $A2$ sequences, a bitonic input is provided for the n input bitonic merger.

Then the inputs to each min-max block in the **filtering line**, starting from the maximum of $A1$ and the minimum of $A2$, and continuing, putting pairs of members of $A1$ and $A2$ into neighboring blocks, those in $A1$ being of decreasing magnitude and those of $A2$ increasing magnitude. Thus producing C, the high outputs of blocks in the filtering line, and D the low outputs in the filtering line. Two properties of C and D are to be shown

1. $c \in C$ and $d \in D$ then $c \geq d$

2. C and D are each bitonic sequences

These two properties will be demonstrated, assuming each input to the n input sorter are

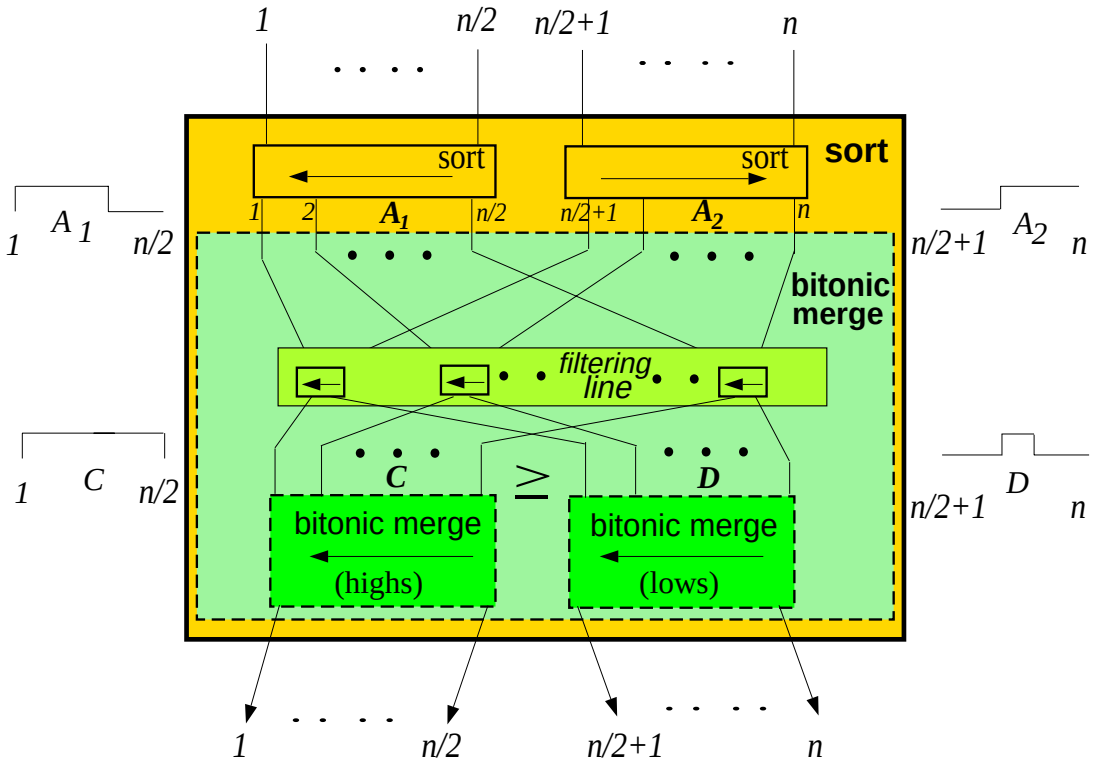


Figure 1-14: Bitonic Sorter

either 0 or 1. Since the resulting n input sorter is easily show by induction, to consist entirely of interconnected min-max blocks it follows that the sorter will work for any numerical inputs.

If 1. and 2. above are true then, because of the assumed property of the $n/2$ input bitonic mergers which produce a sort of the bitonic inputs C and D (by 2.), and given 1. , the entire output of the n input bitonic sorter in the figure is in numerical order.

Now 1. and 2. are demonstrated:

First the inputs to the filtering line:

We can represent a possible bitonic sequence formed with A_1 non-increasing to A_2 non-decreasing on a circle as in figure 1-15. Throughout the figure the darkened part of the circumferences represent $1s$ and the non-darkened $0s$. The two possibilities are represented in I (a), with $\leq 1/2$ the circumference 1 and in II (a) with $\geq 1/2$ equal to 1. In I (b) the running clockwise on the semi-circles designated A_1, from i to ii, and A_2 from ii back to shows two possible sequences of A_1 and A_2 each represented by half of the circumference. Without changing the number of $1s$ and $0s$ on the circumference the different possibilities for A_1 and A_2 can be represented by choosing different diameters to separate the two. Two such diameter choices are shown in I (b) and the linear values of for A_1 and A_2 resulting shown below each. $A1$ can start anywhere from segment $0,1,\ldots,8$ through, respectively, segments $7,8,\ldots,15$,with $A2$ continuing clockwise through the following remainder of the circumference. This range of possibilities maintains the required property

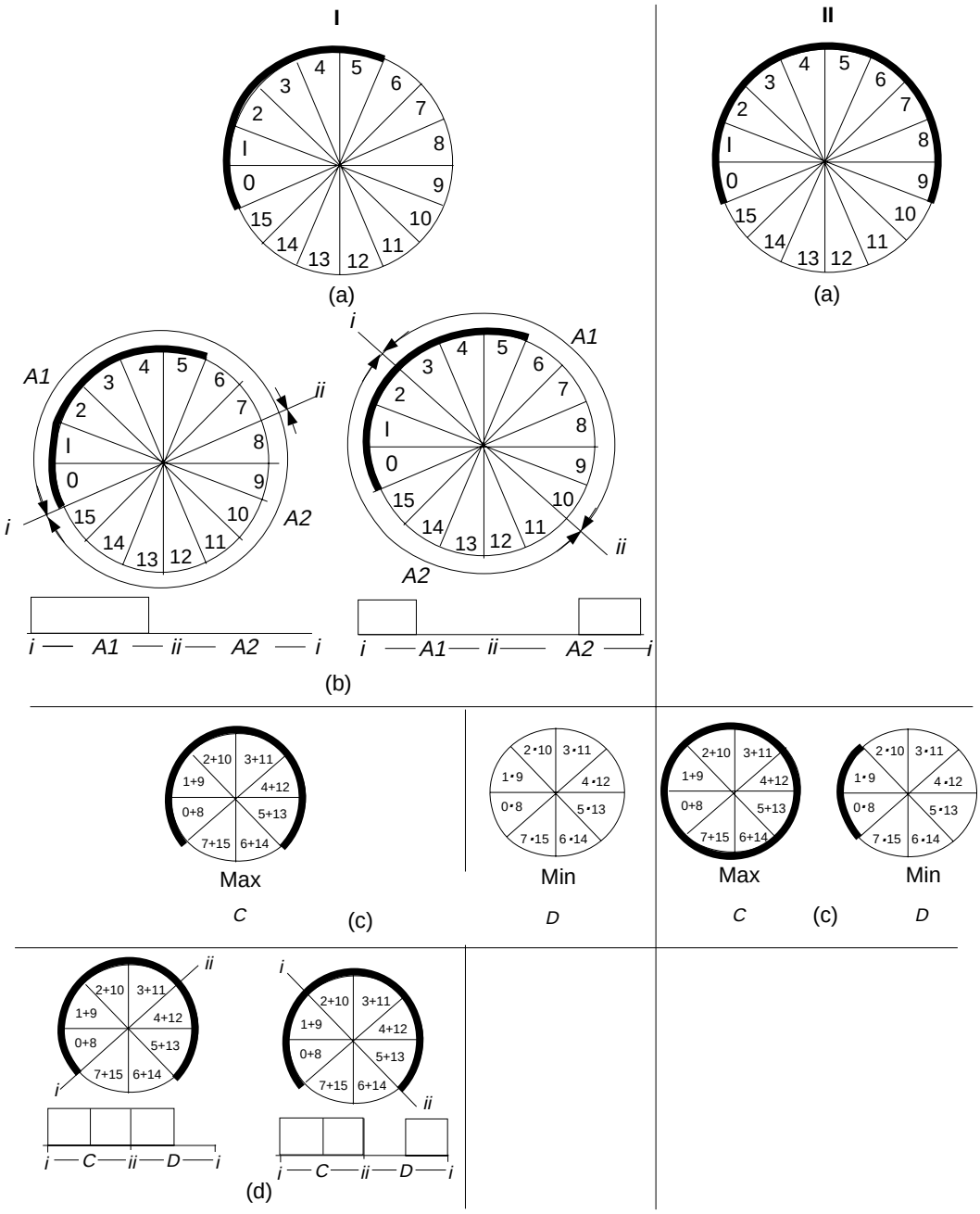


Figure 1-15: Each High Is At Least As Large As All Lows, Both are Bitonic

that $A1$ is non-increasing and $A2$ non-decreasing. Similarly all the possibilities for II(a) are given by the same ranges of diameter placements for defining $A1$ and $A2$ so as to maintain the non-increasing property of $A1$ and $A2$ the non-decreasing one of $A2$.

Given these possibilities for A_1 and A_2 the shapes of C and D can be determined. The filter line results in the maximum (+) of the arc values of opposite sectors give the values that C will take, and the minimum (**.**) of the arc values of opposite sectors give values that D will take. The two circles in I(c) represent these maximums of C, and minimums of D. Note that all minimums are $0s$ and therefore $c \in C$ and $d \in D$ implies $c \geq d$. As with A_1 and A_2, all possibilities for C and D are

represented by rotations of the diameter as illustrated by the two cases in I (d) with the linear sequences of $0\,s$ and $1\,s$ shown below. As illustrated by these example C and D are always bitonic. The other cases for which this is also true is shown in II (c). The argument in this case is analogous, but the maximums have a sequence of $1\,s$ which cover more than $1/2$ of the circumference. So the maximums, C form a sequence of all $1\,s$ and again if $c \in C$, $d \in D$ then $c \geq d$.

1.2.2.2. Number Of Stages

Referring to 1-14, the number of stages in the fully developed bitonic merge, $smerge_b(2^n)$, with $N = 2^n$ is given by:

$smerge_b(2^1) = 1$ (a single *min–max* block)

$smerge_b(2^n) = smerge_b(2^{n-1}) + 1$ (1 for the filtering line)

This is satisfied by $smerge_b(2^n) = n$

Then again referring to 1-14, the number of stages in the fully developed bitonic sort, $ssort_b(2^n)$, is given by:

$ssort_b(2^1) = 1$ (a single *min–max* block)

$ssort_b(2^n) = ssort_b(2^{n-1}) + smerge_b(2^n)$

$ssort_b(2^n) = ssort_b(2^{n-1}) + n$

This satisfied by $ssort_b(2^n) = n^2/2 + n/2$ and since $N = 2^n$ and so $n = log\,N$

$ssort_b(N) = (log^2\,N + log\,N)/2$

Again the cost in *min–max* blocks:

$cmerge_b(2) = 1$

$cmerge_b(N) = N/2 + 2\,smerge_b(N/2)$

$cmerge_b(N) = (N/2)\,log\,N$

$csort_b(2) = 1$

$csort_b(N) = (N/2)\,log\,N + 2\,ssort_b(N/2)$

$csort_b(N) = (N/4)(log^2 N + log\,N)$

EXAMPLE

The development of some small bitonic sort networks from the recursive definition in figure 1-14 is given in figure 1-16. In the next section equivalent time-division sorting networks are considered.

1.2.3. Time Division Connection Networks

The development of this section follows that of [Paull, 66]. It is based on application of general time-space equivalents of networks applied to connection networks.

A continuous signal S can be sampled periodicly and the samples transmitted to a destination--then at the destination the samples can be passed through a filter to recreate the

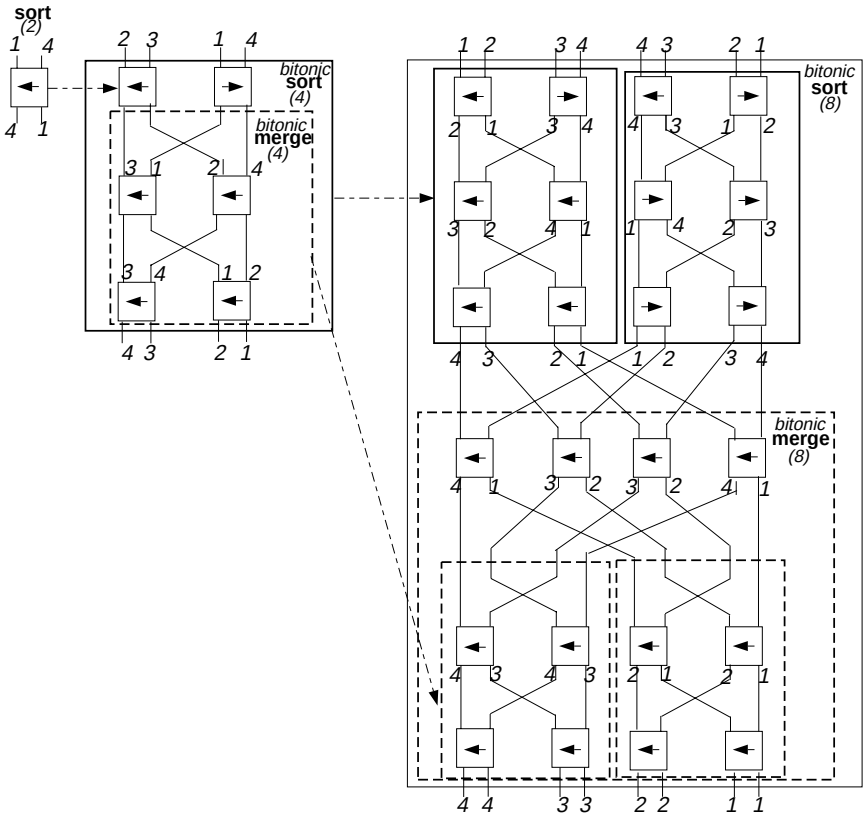


Figure 1-16: Development Of Bitonic Sorting Networks; 2, 4, and 8 Inputs

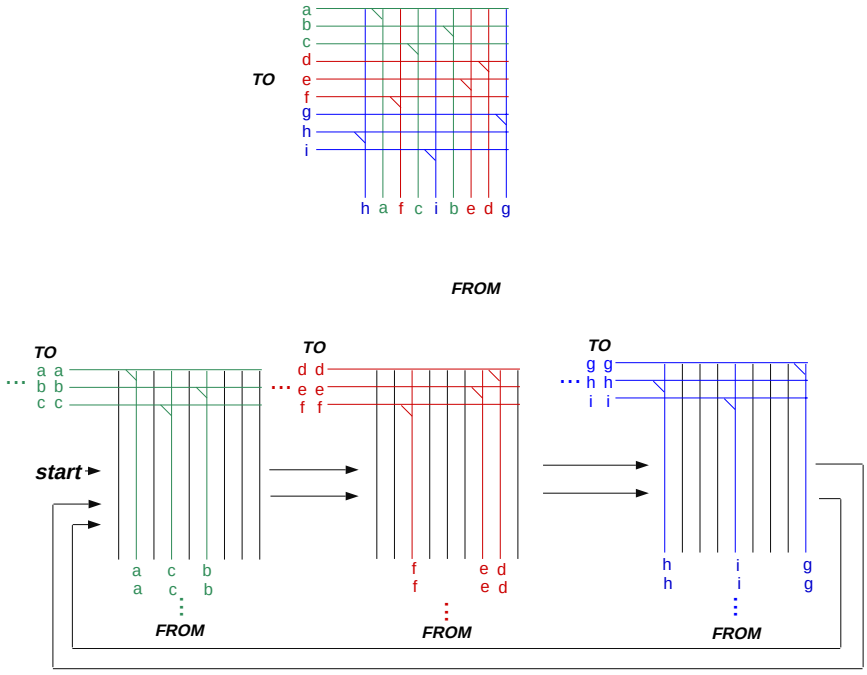


Figure 1-17: Time Division Crossbar Type Switch

continuous signal S. The frequencies in S determine the sampling rate necessary to reconstitute S at the destination. If a number of such sample sequences are to be sent through a switching

network to a set of destinations a simplification in the network is possible because the sampled sequences can be interspersed on the same input and the switch settings changed as the interspersed signals identity changes. In fact whenever pulse sequences are to be sent to different destinations and there is enough time to intersperse a number of such pulse sequences this saving in switches is possible. A very simple example is provided by a simple NXN crossbar switch. In figure 1-17 a $9X9$ crossbar with connections established from each input to an output shown. Assuming the pulse rate required allows interspersing three signals a crossbar of $3X9$ is sufficient to transmit the signals to the desired destinations as shown in (b). If there is sufficient time for 9 sample sequences to be interspersed all could be placed on the same input line which is switched to the one of 9 outputs between successive samples.

The same possibilities for time sharing is available for multi-stage networks. These must compete in cost with the simple single stage crossbar time division network. It should be noted that the ratio of the cost of a single stage and multi-stage (rearrangeable) network is $N^2 / NlogN = N/logN$. The ratio increases with N so only for large networks do multi-stage networks compete. When time division is used one can divide the costs of both by k at the expense of the added control and delays needed--and this expense is greater with the multi-stage network. However the ratio in cost doesn't change. On the other hand the difference in cost is divided by k, the value of the multi-stage time division network competes with the single stage network for very large networks with a k which is not very large.

The connection networks in figure 1-3 have considerable uniformity, enough in fact to make timed equivalents possible. Though not obvious, the structure of the inter-stage connections (perfect shuffle) is unchanged from stage to stage in the Banyan Network. This allows a single stage network with the perfect shuffle connecting its output to its input, operating in time, to simulate a Banyan Network. Also, though perhaps not so obviously, there is a similarity between the switches in the top row with those in every other row of the network. This allows the construction of a single row of switches operating in time, to simulate the space network. These timed networks contain delays represented in the figures by a circled number d giving the size of the delay. The correspondence between the space and timed networks is governed by simple general rules which can be applied to many parrallel networks.

In the timed network the single row of switches operates in t time slots. t is the number of switch rows in the equivalent space network. The switch settings must be changed at the kth time slot to that in the kth row of the equivalent space network. Inputs to the networks can be any number, say n, of bits, but each path through the network must be of width n. The inputs can be presented serially so that the same input receives a sequence of n bit words. In the timed network a serial input is presented with one word following another with period t at that input. If an analog signal is sampled at intervals of t and digitalized the sequence of digitalized samples (words) can form an input to the timed network. At the output established by the switch connections the output samples arrive in the same order in which they were sent, words, after the first, arrive at intervals of t. So provided t is large enough, the analog signal can be reconstructed at that output.

There is a delay experienced by the first signal passing through the timed network. This is determined by the delays through which that signal must pass which is, in turn, determined by the switch settings.

EXAMPLES

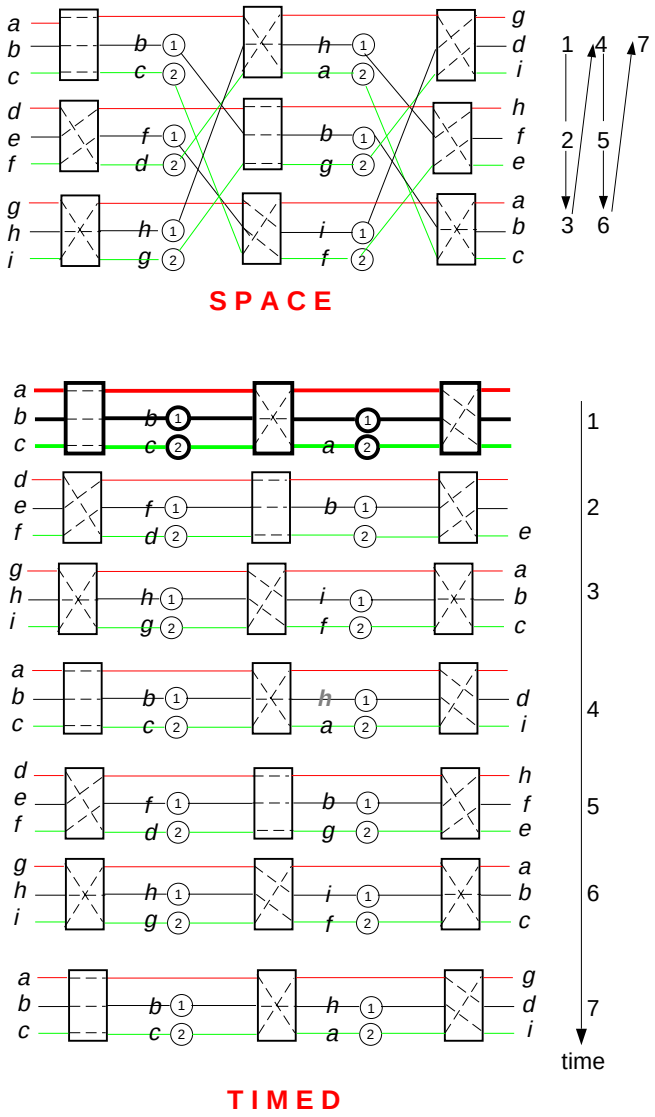


Figure 1-18: Three Stage Connection Networks--Space And Timed Equivalents

In figure 1-18 a three stage rearrangeable ND space network with *nine* 3–*by*–3 switches is shown with *nine* inputs, a through i propagating through it. A circled numbers, i in a line is only there to indicate the number of rows of switches that are skipped by that line before it reaches its destination. The levels are counted circularly from the top as shown at the right of the ND network. These determine the delays in the equivalent timed T network consisting of *three* 3–*by*–3 switches shown in boldface just below the ND network. This is also used to show the state of the timed network during in the first of a series of seven time slots, the state of the timed network at the succeeding 6 time slots is shown below. The same inputs presented to the space network are shown propagating in time through the timed network. The switch settings in the timed network at time slot k are the same as those in the space network in row k.

This correspondence between these timed and space networks can be derived using principles established earlier. Starting with the T network which is in fact *acyclic*. In general switch setting, and inputs can be different in each of the first three time slots, after which they repeat-they cycle. It follows that the resultant outputs will also cycle, and that when the iterative equivalent ND

network is constructed after every three inputs state inputs and network outputs will repeat. Finally the iterative ND network can be condensed to three blocks (each containing three 3–*by*–3 switches) by taking the state outputs of the third and feeding them back to the appropriate state input locations in the second and first. This results in the ND space network at the top of the figure.

In the timed network an input can travel to the output without experiencing any delays. On the other hand it could be delayed by as much as 4 time slots in its *first* passage through the network, 4 being the maximum delay that can be experienced in such a passage. This is the maximum priming time of the network. Input a is delayed by 2 time slots, while g is delayed by the maximum 4. But the time between appearances at the output of a, of g, or of any other input is 3 or the number of time slots before an input is repeated--the output rate must equal the input rate.

In figure 1-19 a five stage rearrangeable network with *forty-five* 3–*by*–3 switches is shown. This space version has *twenty-seven* inputs propagating through the network. The timed network equivalent require scheduled switching of some delays. The state of the timed network is shown with the same inputs as the space network, propagating over time, in figure 1-20. It consists of *five* 3–*by*–3 switches (one row of switches in the figure). Its state is shown during a sequence of nineteen time slots. Note that every third copy of these rows of 3–*by*–3 switches have the same delays, but the second and third rows have delays different from each other, and from the first. To achieve this difference the network needs additional scheduling switches, to periodically select delays. The required scheduling switches are shown only in the bottom row of 3–*by*–3 switches. They are switched to get delays of 1 and 2 (switches as shown), then 1 and 8 (bottom switch down) and 7 and 8 (both switches down) on successive time slots. The time between any two appearances at the network output of each input is, as in all such timed networks, the number of time slots required by the network to receive copies of all its inputs. It is the same for all inputs and is 9 in this example. The large saving in hardware here is balanced by the long maximum possible delay from the time an input first appears to its arrival at the output of the network (different than the time *between* arrivals) . For this network that time is $6 + 8 + 8 + 6 = 28$.

One problem with the rearrangeable networks considered here is the relatively long time required to calculate the connection switch closings necessary to make a desired set of connections, even with a parallel algorithm. This can be alleviated by using a Clos network in which this computation is faster. The Clos network requires $2n - 1$ central stage switches, where there are n input switches. This however means that the uniformity needed to produce a timed equivalent is lost. So a central stage switch can be added ($2n$ now, the fast connection algorithm available to set connection switches is preserved, while the need uniformity is maintained.)

In figure 1-21 a *three* stage Clos network with *six* 3–*by*–6 and *six* 3–*by*–3 switches is shown in both the space and timed version. The space version has *nine* inputs propagating through the network. The central stage 3–*by*–3 switches used in establishing the connections between first and last stage 3–*by*–6 switches is shown at the top. The timed network equivalent is shown at the bottom. It consists of *two* 3–*by*–6 and *one* 3–*by*–3 switches. The sequence in which inputs are presented to, and received from the timed network are shown respectively at the inputs and outputs of the network.

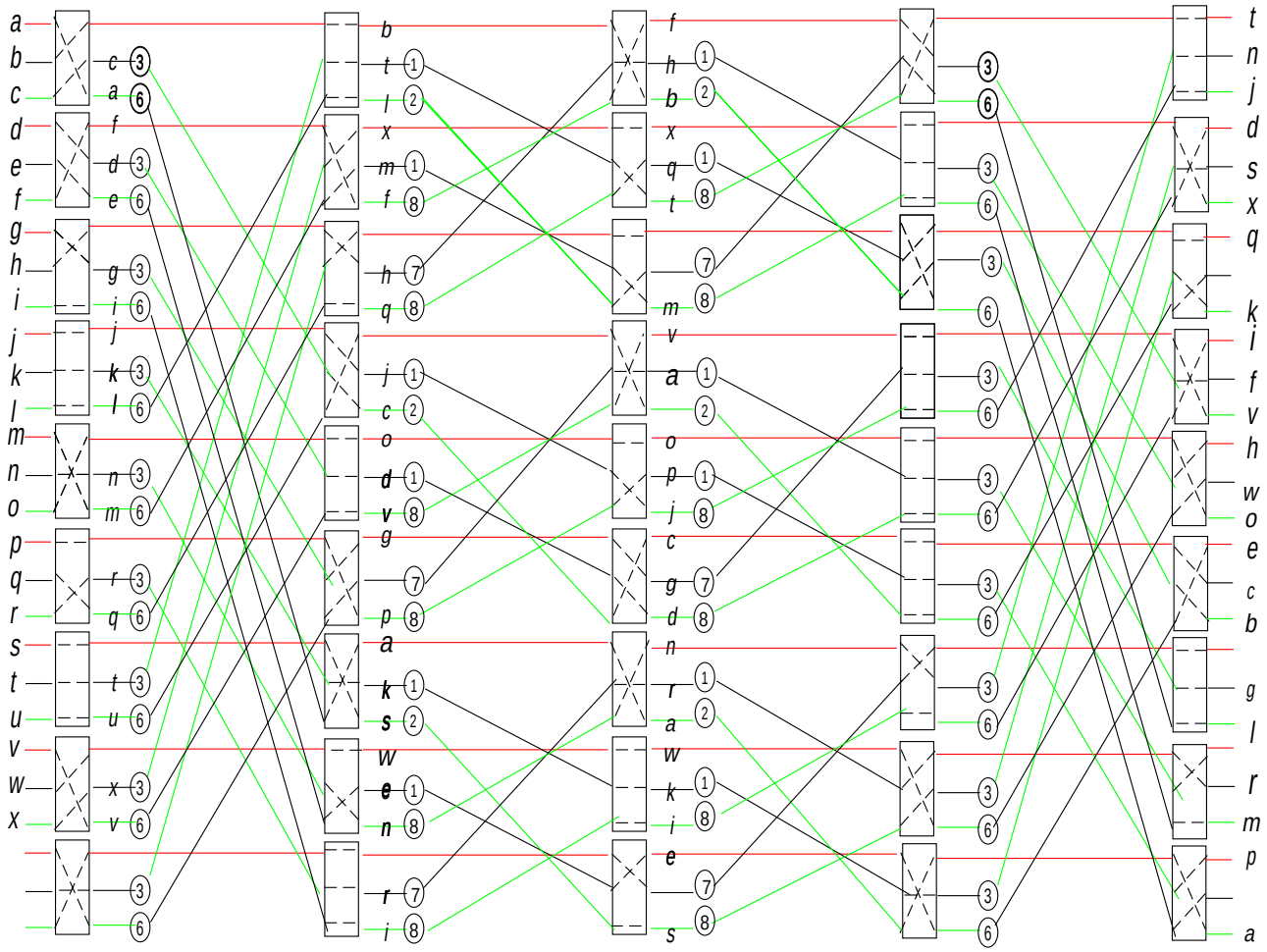


Figure 1-19: Five Stage Connection Networks--Space Equivalent

1.2.3.1. Time Division Sorting-Connection Networks

For the connection networks considered earlier any of a 3–*by*–3 switches inputs will pass through correctly, independent of the time its other inputs arrive. For simulating sorting networks with time division it is necessary that both inputs that will eventually arrive at a *min*–*max* block be present simultaneously to get the correct output. This fact and non uniformities in connection pattern make the time division sorting networks more complex and also perhaps more interesting. They require delays, and periodic switching of delays, of connections, and of block functions. Also priming is necessary. Furthermore, when used for connection networks all inputs must be specified.

If the sorting network (space or time) is used simply to sort the inputs then any input which is not specified is conventionally assumed to be ∞ or zero and all is well. However, if it is used as a steering connection network, each input has a destination and and information part. As noted earlier such a network will not make correct connections unless the destination part for *all* possible destinations appear at the network input. So, though connections from only a few network inputs are to be made, the other inputs must have specified and different destination parts, an awkward requirement. In figure 1-22 the time division sorting network during its first four time slots is shown at the top. These four incarnations are repeated periodically. To achieve the four incarnations, the inter-block delays and connections must be changed (switched) periodically between time slots.

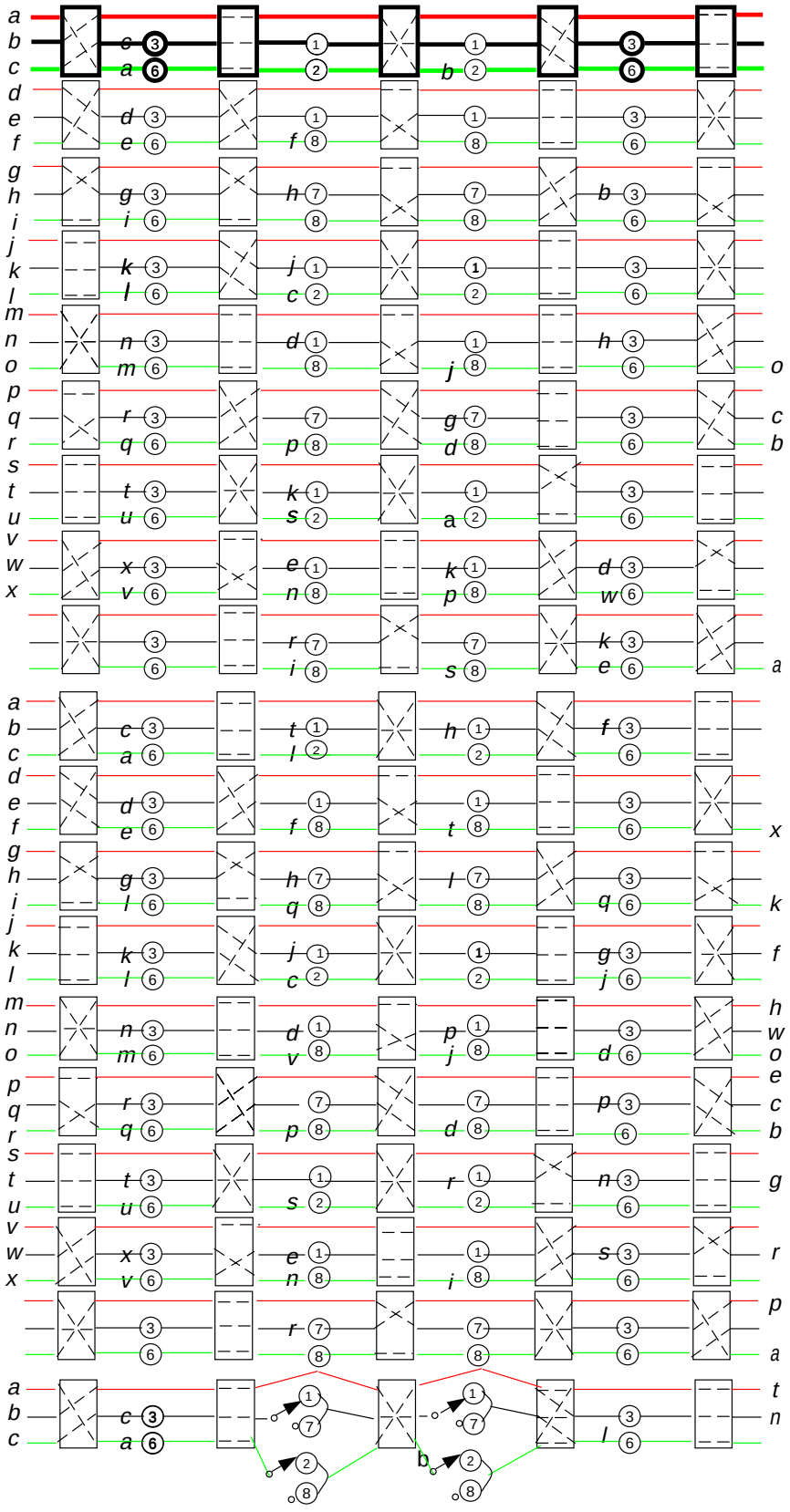


Figure 1-20: Five Stage Connection Networks--Time Division

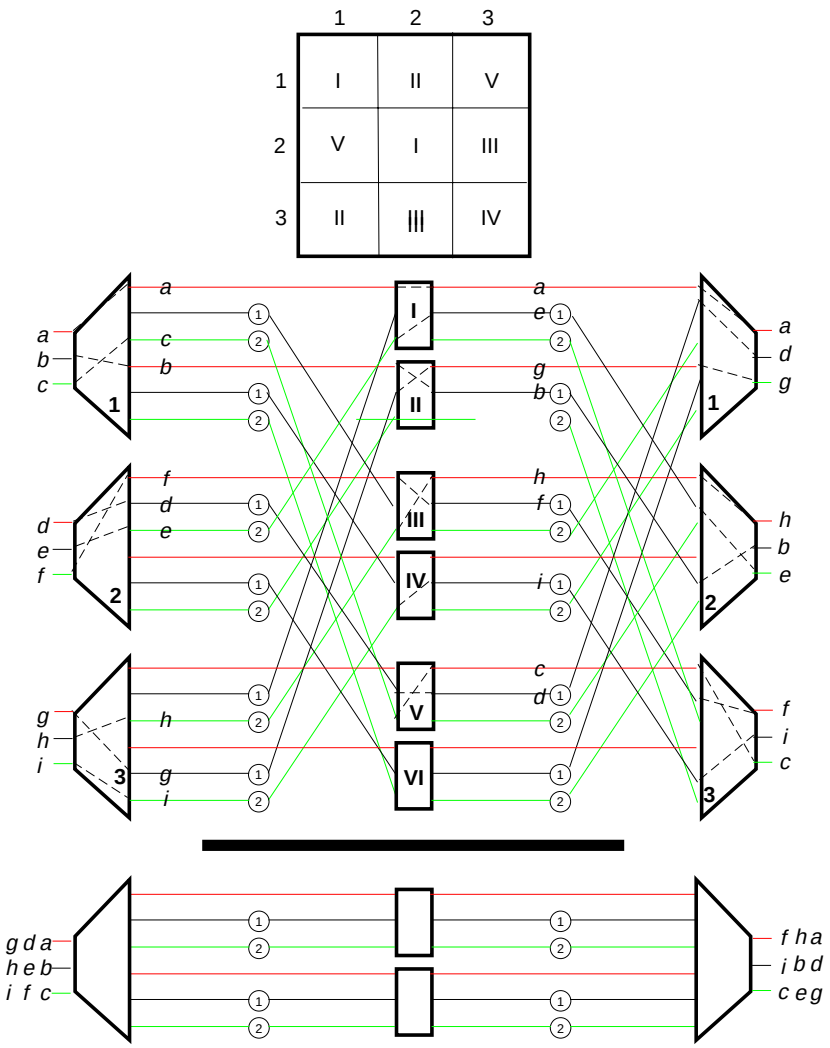


Figure 1-21: Clos Network, Connection Matrix-Space Connections, Timed Equivalent

Also the *min–max* blocks need periodic change (arrow up/down).

Figure 1-23 shows the timed network used only for sorting its inputs. The set of inputs are repeated periodically. The numbers 1 through 4 are each presented twice as network inputs to be sorted. The flow of these numbers through the network at seventeen successive time-slots is given. At the early time slots the numbers produce outputs at *min–max* blocks only as far as the early stages, ex., the second block in time slot 2, and third in time slot 5. These blocks are marked with darkened right sides in the figure. The design requires that blocks with less than two inputs at a given time slot present no outputs. Only blocks with two inputs present during the same time slot have outputs. In the last 4 time slots illustrated all blocks in the time division network have legitimate outputs. They are sorted as shown with each number appearing twice as they were at the input. The smallest appearing at the representation of the first output block.

If the network were used as a connection network the destination part of each input steers the information part through the network. Unlike the example in figure 1-23 inputs in the figure no two destination parts could be the same; they should also be between 1 and 8. The same principal illustrated in the figure would steer these eight numbers through the network with their

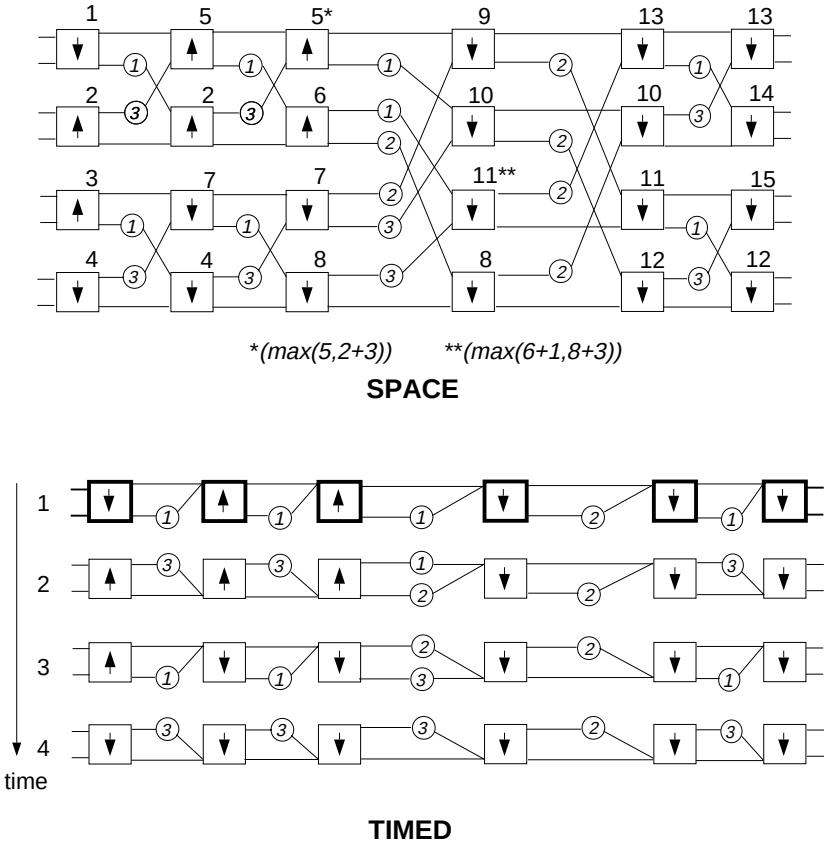


Figure 1-22: Bitonic Sorting Networks--Timed Equivalent

accompanying messages.

In calculating sorting or connection network timing it is assumed that the time taken by a *min–max* block is 0, as is the time taken on any interconnection which does not contain a delay. The delay before useful output occurs is the same independent of the distribution of the eight destinations presented to the network input. This is also true if the network is used for sorting only. This is different than the situation with the timed connection networks. In the timed connection networks any of the connections from input to output can be made one switch after the other without waiting for any other connection to arrive at the switches--i.e, independently. The maximum time to get an input through the network depends on the path with the maximum possible delay, and this is simply the sum of the largest delays between successive switch stages. This time is only incurred if there is a path using these delays and one may reasonably speak of the average delay to make a connection.

For the sorting connection network however a *min–max* block is inoperative during a time slot unless both of its inputs are present--independent of the size of those destination parts. Therefore the structure of the network alone determines the times. These can be determined for example by presenting $1s$ at every input as the number to be sorted. Alternatively one can determine these times by calculation using a version of the space network with the corresponding delay times recorded on the interconnections. This calculation is illustrated in the lower part of figure 1-22. For this network the time at which the computation of a block first appears at the output of a block is shown above each block. The calculation is based the following considerations.

If the time of first completion of block x at stage k is $t_k(x)$ and the delay on a line from a kth stage block x to block b in stage $k+1$ is $d_k(x,b)$ then the first completion of block b at stage $k+1$ is $max_{x\ connected\ to\ b}(t_k(x)+d_k(x,b))$. In addition to being illustrated in figure 1-22, the result is demonstrated in figure 1-23 by marking the last active block at each time unit. The last two outputs of the space network first appears at time 12. Then the first two outputs of the space network are seen at time 13, the next two at time 14, the next two at 15, and the final two at 16 of the timed network. The outputs continue to appear in the same order until the inputs of the network cease, and then they withdraw in time from the outputs.

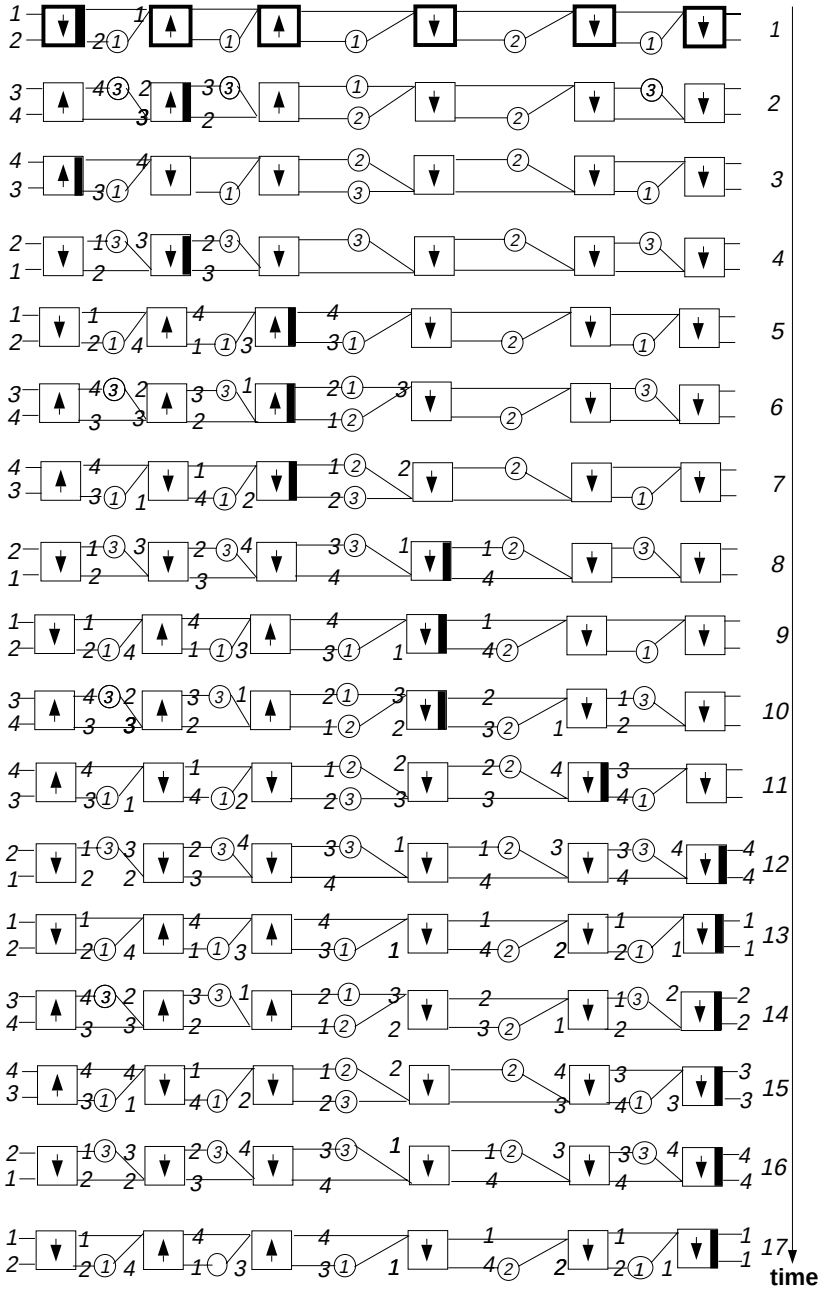


Figure 1-23: Bitonic Sorting Network Signal Transmission Over Time

REFERENCES

[Benes,65] V. E. Benes, "Mathematical Theory of Connecting Networks and Telephone Traffic, Academic Press, New York and London, 1965

[Hui,90] Joseph Y. Hui, "Switching And Traffic Theory For Integrated Broadband Networks", Kluwer Academic Publishers, Boston/Dordrecht/London, 1990

[Paull, 64] Paull, M.C., A System For Controlling A Switching Network, Patent No. 3129407 1964

[Paull, 66] Properties Of Time Division Switching Networks and Their Relation to Space Division Networks, Bell Laboratories Technical Memorandum, 1966.

Index

Table of Contents

List of Figures