Programmable indexing networks

by KENNETH JAMES THURBER
Honeywell Incorporated
St. Paul, Minnesota

INTRODUCTION

One of the most important functions that must be performed in a digital machine is the handling and routing of data. This may be done in routing logic (computers), in permutation switching networks (computers and telephone traffic), sorting networks, etc. In some parallel processing computers being envisioned the handling of large blocks of data in a parallel fashion is a very important function that must be performed. For a special-purpose machine a fixed-wire permutation network could be acceptable for the handling of data; however, for a general-purpose machine more sophisticated reprogrammable networks are required.

The permutation network problem has been previously studied by Benes, Kautz et al., Waksman, Thurber, and Batcher. This paper introduces and defines a new network to be considered. This is the generalized indexing network. This network can perform an arbitrary mapping function and is easily reprogrammable to perform any other arbitrary map with \( n \) inputs and \( m \) outputs, and has many potential areas of use. The most interesting possible area of application is the processing of data while routing the data. If the network is used as routing logic, it can perform many simple data manipulation routines while routing the data e.g., matrix transposition.

Some of the solutions presented are significant improvements on the shift register permuters suggested by Mukhopadhyay. The solutions suggested here are programmable (utilizing the output position mask), as fast, and utilize less hardware than the previously suggested shift registers permuters.

FORMULATION OF THE PROBLEM

Previously, most researchers have considered the problem of permuting a set of \( n \) input lines \( X_1, X_2, \ldots, X_{n-1}, X_n \) onto a set of \( n \) output lines \( Y_1, Y_2, \ldots, Y_{n-1}, Y_n \) by means of a device called a permuter. A permuter produces a one to one mapping from the \( n \) input lines to the \( n \) output lines of the network. The permuter can perform a very limited set of functions. As currently studied, the permutation networks can only transfer lines of data. In this paper the networks will be utilized to transfer words of data.

Limitations of permutation networks are that input words cannot be repeated or deleted at the output. Also, blanks cannot be inserted into the output and the number of input words and the number of output words must be equal. The indexing network differs from the permuter in that input words can be repeated or deleted and blanks can be inserted in the output. Also, for an indexing network the number of input words \( n \) has no special relation to the number of output words \( m \). The non-blank output words may appear in many contiguous subsets of the output words (these subsets could be empty). Figure 1 shows some examples of possible permutation networks. Figure 2 shows some examples of possible indexing networks.

In this paper \( X_i \) means a word of input data (instead

\[
\begin{align*}
X_1 &= Y_2 \\
X_2 &= Y_1 \\
X_3 &= Y_4 \\
X_4 &= Y_3
\end{align*}
\]

**Figure 1—Permuter**

*The terminology indexing network and generalized indexing network will be taken to have the same meaning.*
transfers are arranged such that a transfer pulse to the input set of registers causes the simultaneous parallel cyclic transfer of the contents of the registers; i.e., $n \rightarrow n - 1, n - 1 \rightarrow n - 2, \ldots, 2 \rightarrow 1, 1 \rightarrow 0$, and $0 \rightarrow n$ simultaneously. A transfer pulse to the output set of registers (and to the OPM) causes the simultaneous parallel transfer of the contents of the registers; i.e., $n \rightarrow n - 1 (\text{OPM}(n) \rightarrow \text{OPM}(n - 1)), \ldots, 2 \rightarrow 1 (\text{OPM}(2) \rightarrow \text{OPM}(1)), \text{and } 1 \rightarrow n (\text{OPM}(1) \rightarrow \text{OPM}(n))$. The previously specified functions are performed by the Input Cyclic Control (ICC) and Output Cyclic Control (OCC) respectively. The Transfer Control (TC) performs the function of transferring data from input position 0 to output position 1. There is no output position 0.

Figure 3 shows the clocking hardware used to read the OPM and produce the desired control pulse for the TC. It is assumed that the clocking hardware contains a clock with clock rate $c/p$, where $c$ is the clock rate of the sorter and $p$ is a suitable positive integer. Binary constants $c_2, c_1,$ and $c_0$ placed on the input lines to the network produce an output from the network after $(c_2(4) + c_1(2) + c_0(1))$ units of delay. One unit of delay is equal to the time period between indexing clock pulses (the clock rate of the indexing network is $c$ so a unit delay is $c^{-1}$ second). The clocking hardware is used to advance the input registers to a position selected by the OPM.

Figure 4 shows a general setup for an indexing network and a complete indexing network for $n = 5$ and $m = 4$. The words are 4-bit words in this example. The indexing network consists of an input set of registers and associated ICC and TC hardware, an output set of registers and the associated OCC and OPM hardware, and the clocking and control hardware.

The clock rate of the indexing network is $c$ per second
and the clock rate of the clocking hardware is $c/8$ per second. In general the clock rate for the clocking hardware is $c/n + 3$ per second.* No provisions have been shown for connecting the network to other hardware, but this should be obvious. A blank (binary 0) is placed in register 0 of the set of input registers.

---

* $c/n + 3$ are needed instead of $c/n$ because (1) a time period is needed for shifting $n + 1$ input values instead of just $n$ input values, (2) a time period is needed to transfer the data, and (3) a time period is needed to shift the output registers and OPM.
Figure 7—Comparator indexing network for \( n = 6 \) and \( m = 4 \)

The details of the operation are as follows:

1. The input data and the OPM are inputted into the network.
2. The input data is cycled until the input code equals the current value of the OPM.
3. The input word is transferred to the output register.
4. The output register and the OPM are advanced one position unless the output register is full in which case go to (6).
5. Go to (2).
6. Output the data in the output register.
7. Stop.

EXTENSIONS OF THE SOLUTION GIVEN IN PREVIOUS SECTION

The solution given in the previous section is interesting in that there are several other methods by which it can be implemented in a more sophisticated manner. Since the solution given previously does not require as much hardware as some of the other solutions it is interesting to consider what can be done with the addition of some extra hardware.

As with the solution given in the third part of this paper, the solution given in the previous section can be implemented in a form such as in Figures 5 and 6. Also, it could be implemented in any form that "lies" between the solutions given in Figures 5 and 6.

The following solutions require that the set of input registers be able to shift cyclically backwards (0 \( \rightarrow \) 1, 1 \( \rightarrow \) 2, \( \ldots \), \( n - 1 \rightarrow n \), \( n \rightarrow 0 \)) as well as forwards (1 \( \rightarrow \) 0, 2 \( \rightarrow \) 1, \( \ldots \), \( n \rightarrow n - 1 \), 0 \( \rightarrow n \)).

One method of improving the solution given previously is to make more than just a comparison of the two numbers for equality. A solution is to check and see whether the number contained in the OPM is greater than, equal to, or less than the number designating the current state of the input. If the OPM number is larger shift the input register forward, if the OPM number is smaller shift the input register backwards, and if the numbers are equal then transfer the information. The actual shifting can be implemented as in the previous section (a comparison after every input shift) or as in the third section (this would require a subtraction to determine the number of needed periods of delay) using the clocking hardware in Figure 3 to produce the transfer pulse.

Another improvement that can be made is based upon the following observation; i.e., if the set of registers can cycle both forwards and backwards then there are cases where it is shorter time wise to go around one of
the "ends" of the set of input registers. For example, if \( n = 10 \) and the network is at 9 and needs to go to 1 then the shortest way is 9 \( \rightarrow \) 10, 10 \( \rightarrow \) 1 (instead of 9 \( \rightarrow \) 8, 8 \( \rightarrow \) 7, \ldots, 2 \( \rightarrow \) 1). This solution can be implemented by calculating and comparing \( n + 1 - | p - q | \) to \( | p - q | \) were \( p \) and \( q \) are the current location and the desired location. Again this solution could be built as in the previous section (comparison after each input shift) or as in the third section (using clocking delays); however, it is probably best implemented using clocking hardware (such as in Figure 3) because the minimum of \( n + 1 - | p - q | \) and \( | p - q | \) give the number of time delays to be produced by the clock. Therefore, after the comparison has been made, the minimum value can be used as input data into clocking hardware and the register cycled in the proper direction (forward or backward).

THE SPLITTER

This section presents a solution to the generalized data indexing problem based upon an input decision called the input position map. This solution utilizes a modular construction and seems most interesting in the case in which a lot of different indexings must be produced in rapid succession. A major advantage of this type of network is that it is capable of simultaneously processing many indexings at the same time.

The input position map (IPM) is a set of binary codes associated with the input data of a network that specifies the position (or positions) that the data is to be transferred to in the set of output registers. In the case of the design of a splitter it will be assumed that the input data and the binary code contained in the IPM associated with that input data are contained in an extended register as shown in Figure 8.

Figure 10 shows the general block diagram of several splitter networks organized to perform a permutation function. Each module in the splitter takes the \( n \) inputs (assume \( n \) is even) and groups of these \( n \) inputs into two \( n/2 \) input groups based upon the mapping information contained in the mapping information portion of the node. The splitter is most useful in constructing sorting networks that have \( n = 2^k \).

The permutation network shown in Figure 10 can be built in various sizes so that it can be configured as shown in Figure 9. The mapping information inputted to this network would be the binary value of the position in the set of output registers that the data was destined for so that an arbitrary input register would contain DATA and DESTINATION OF DATA where the destination of the data is between 0 and \( n - 1 \). The first splitter encountered \( (n \to n/2) \) would sort the information based upon the binary value contained in the highest order digit; whereas, the last group of
splitters \((2 \rightarrow 1)\) would read the lowest order digit. The values being read would be inputted to the AND gates as shown in Figure 10. The full word of data would be transferred to the appropriate output register in parallel and the appropriate output register and the input register would be advanced one position each. The next word is then processed in the identical manner. To split \(n\) elements into two \(n/2\) element groups requires \(n\) clock periods. The bit that the AND gate reads is different at each level, but begins with the high order digit and proceeds to the low order digit.

The IPM for a permuter constructed by the splitter method is just the binary output destinations of the data. It is a little harder to construct a generalized indexing network using this concept. The permuter was easy because it needed a one to one and onto mapping function. A generalized indexing network is a little harder but not impossible. It will be slightly harder to compute the IPM than it was for the permuter, but the following method and the hardware shown in Figure 11 configured as in Figure 9 will produce a generalized indexing network. One modification of the network is that in the first splitter, the data must be broken from \(n\) into two groups of \(m/2\) elements. From that point on each group of \(m/2\) elements is split into two groups of \(m/2 + 1\) elements. The mapping information for the network can be furnished by the following observations. Each element of input data can be categorized as to where it is transferred by means of a two-bit binary map (byte). The high order byte specifies the split \(n \rightarrow m/2\); whereas, the low order byte specifies the split \(2 \rightarrow 1\). There are exactly four distinct possibilities that can happen to a piece of data; i.e., the data not transferred to either output register, the data transferred to one but not the other output register (two possible cases), or the data transferred to both output registers. These are indicated in Figure 12 and the necessary hardware shown in Figure 11. This design allows design of a generalized indexing network if the output registers are all set to the blank (0) value before they receive any data. In order to make the splitter work utilizing two bit bytes, the mapping information must be introduced at each stage of the process as shown in Figure 13. If the mapping information was completely specified with the data in stage 1 there would be no way to produce the indexing \((X_1, 00, X_i)\) because the second byte would have to be 10 and 01 simultaneously. \((X_1, 00, X_i)\) could be produced by the map 11 associated with \(X_i\) at stage 1 the map 10 associated with the value of \(X_i\) in stage 2 \((A')\), and the map 01 associated with \(X_i\) a stage 2 \((B')\) in Figure 13. The difficulty encountered in constructing the
maps for the splitter is balanced by two advantages of the splitter; i.e., (1) the designer can get by with only two bits of mapping information in each data word at every stage of the process (this has not been done in Figure 13, but the reader can clearly see why it can be done by looking at Figure 13), and (2) since previously used mapping information is no longer needed, many different indexings can be in process at the same time.

The IPM can be constructed by tracing the desired data output back through the network. Figure 13 is a generalized indexing network for \( n = 8, m = 4 \) constructed using the splitter concept. It is conceivable to combine the networks using only one portion to replace the portions marked \( AA' \) and \( BB' \), thereby eliminating some transfer hardware and \( A(A') \) and \( B(B') \) at the expense of more complex clocking and logic. By changing the size of the bytes it is conceivable to construct many different IPM's, but the previously explained IPM seems to be a very good one to use.

This network can be built to provide very high rates of throughput since the level \( m/2 \) splitter takes half as much time to operate as the \( m \) level splitter. With some sophisticated clocking it is conceivable to “time share” the \( m/2 \) level splitter with two \( m \) level splitters and thereby maximize throughput.

**DISTRIBUTED INDEXING NETWORKS**

This section presents the final two solutions to the indexing network problem considered in this paper. These two networks are characterized by a highly parallel operation, high speed, and unique timing arrangement. Each network has one comparator (or clocking hardware unit) and one transfer control unit for each word of desired output. In some cases \( (m > n + 1) \) this requires the addition of several extra blank input registers as in Figure 16. It is assumed that the clock controlling the cycling of the input register has a long enough time between pulses to allow the comparison and transfer of data.

Both of these networks are based upon the observation that in a complete cycling of the input registers, all data passes through every register. When the correct word is recognized it is immediately transferred.

In the comparator solution in Figure 14, at every clock period the data currently occupying input positions 0, 1, 2, ..., \( M - 1 \) is compared to the OPM and the appropriate transfers made. This solution (and the solution shown in Figure 15) requires the larger of \( m \) or \( n + 1 \) clock pulses for the indexing of the input data.

The solution shown in Figure 15 requires a modification of the OPM. The value of the \( i \)th position of the OPM is not the binary value of the input data desired, but the number of clock pulses before the input data is in the \( i \)th position; i.e., if \( Y_i = X_j \) then

\[
\text{OPM} (i) = \begin{cases} 
  j - i & \text{if } j \geq i \\
  m - (i - j) & \text{if } j < i \text{ and } m > n + j \\
  n + 1 - (i - j) & \text{if } j < i \\
  & \text{and } n + 1 \geq m
\end{cases}
\]
CONCLUSION

A new class of networks was presented in this paper. These networks have the ability to arbitrarily reorder a set of \( n \) input cells into \( m \) output cells with the repetition or deletion of any cell allowed. Blank cell values may be arbitrarily placed in any of the output cells, thereby allowing the construction of arbitrary contiguous sets of data separated by blanks in the output. These networks have as special cases previously studied permutation and sorting networks. The networks described here are extremely general in nature and should have many different areas of application, particularly, in areas needing networks for routing and transferring data.

The ease of programmability of the indexing networks described is a feature that is extremely unique. Almost all previously studied permutation and sorting networks have long set up and programming times that tend to make them useless in problems in which the destination of the data has to be changed between each set of data inputted. The manner in which the programs are inputted into the network and the simplicity of the program are other features that are unique to the approach followed in this paper. Another unique feature of the solutions presented is the range of tradeoffs they cover. The designer can easily make tradeoff comparisons between the solutions and has many possible different ways to configure each type of network to obtain various speed and hardware comparisons. Hybrid solutions may be extremely attractive.

An interesting solution to consider is utilization of the splitter to go from \( n \) to \( n/2^k \) followed by utilization of \( 2^k \) non-splitter networks (like a comparator network). In this manner the large input block of data can be broken down for high-speed “parallel” sorting by other networks.

It is suggested that future research consider construction of high-speed routing networks utilizing the previously described sorting networks. These networks seem to be particularly attractive for the routing and re-arranging of data in parallel processors. Another topic that might be of interest is the investigation of the possibilities of performing logic operations on the data while it is being routed (indexed). Consideration might be given to the use of these networks as memories. Some of the logic might be able to be used to convert from a indexing network to a memory.

Some possible applications for the generalized indexing networks are: sorting of data, routing of data, permutation networks, multi-access memories with the number of words of memory accessed a controllable variable, associative memories, multi-access associative memories, reconfigurable multi-processors for real-time users, associative multiprocessors, and any other applications which require the manipulation and re-configuration of large amounts of data.

BIBLIOGRAPHY

1 K E Batcher
   Sorting networks and their applications
2 E V Benes
   Mathematical theory of connecting networks and telephone traffic
   Academic Press New York 1965
3 W H Kautz K N Levitt A Waksman
   Cellular interconnection arrays
4 A Waksman
   A permutation network
5 K J Thurber
   Design of cellular data switching networks
   Submitted for Publication
6 A Mukhopadhyay G Schmitz K J Thurber K K Roy
   Minimization of cellular arrays
   Final Report NSF Grant GJ-158 September 1969 Montana State University
   Bozeman Montana