

Brief Contributions

A Fast VLSI-Efficient Self-Routing Permutation Network

Hasan Cam and Jose A.B. Fortes

Abstract — A multistage self-routing permutation network is presented. This network is constructed from concentrators and digit-controlled 2×4 switches. A destination-tag routing scheme is used to realize any arbitrary permutation. The network has $O(\log^2 N)$ gate-delay and uses $O(N^2)$ VLSI-area, where N is the number of inputs.¹ Assuming packet-switching is used for message transmission, the delay and VLSI-area of the network are smaller than those of any self-routing permutation network presented to date.

Index Terms — Concentrator, permutation, network, radix-sorting, self-routing.

I. INTRODUCTION

An interconnection network (IN) with $N = 2^n$ inputs/outputs is called a permutation network (or rearrangeable network) if it realizes every one of the $N!$ possible permutations in a single pass. If each switch in a network determines its own setting dynamically by examining the routing tag bits of the inputs, then the network is said to be self-routing. This paper presents a multistage self-routing permutation network, called PN, constructed from concentrators and digit-controlled conflict-free switches. Packet-switching is assumed for message transmission.

A sorting network can be used as a self-routing permutation network, although not every rearrangeable network is a sorting network [1]. Sorting networks are often constructed using two-input, two-output comparators that send the smaller of their two inputs to the upper output. When a sorting network is used as a self-routing permutation network, the routing algorithm becomes very simple because packets are simply sorted according to their destination addresses. The best known sorting networks are the odd-even merger and the bitonic sorters presented by Batcher [2]. For N inputs/outputs, each of these networks consists of $\log^2 N$ stages of $N/2$ comparators each. These two networks have the same time and area complexities, so they are collectively called "Batcher's network" in this paper. In order to compare the performance and cost of Batcher's network with those of PN, these measures are computed below with respect to two different computation models.

A sorting network can be analyzed in two different models of computation: bit model and word model [3], [4]. In the bit model, a comparator compares and passes the numbers bit by bit from left to right, most significant bits first. In this model, when a comparator receives two numbers, it passes the leading bits through as long as they

are equal and sets itself according to the first different bits of the numbers. Thus, numbers pass through comparators in a bit-serial pipelined fashion. This implies that a comparator can be considered a 2×2 digit-controlled switch in the bit-model. Therefore, with respect to this model, Batcher's network has $O(\log^2 N)$ gate-delay and uses $O(N \log^2 N)$ switch-area. In the word model, it takes a single step for each comparator to compare and transmit the *whole* incoming words (numbers). With respect to the word model, Batcher's network has $O(\log^2 N)$ comparator-delay and uses $O(N \log^2 N)$ comparator-area. The gate-delay (also referred to as bit-level delay) can be determined by multiplying the comparator-delay by $\log N$ [5] because a comparator can be replaced by $\log N$ 1-bit wide comparators. Similarly, the asymptotic cost of sorting networks can be expressed in gate-area by multiplying their cost by $\log N$ [5]. Hence, Batcher's network operating in the word model has $O(\log^3 N)$ gate-delay and uses $O(N \log^3 N)$ gate-area.

When a sorting network operating in the bit-level mode is used as a self-routing permutation network, a dedicated path is established for every destination and messages are sent along this path in a bit-serial pipelined fashion. Therefore, a sorting network operating in the bit-level model can be used as a self-routing permutation network if the circuit switching is adopted as a routing technique. When a sorting network operates according to the word model, a message and its destination address move together from stage to stage. Therefore, it can be used as a self-routing permutation network if packet switching is adopted as a routing technique. In this paper packet switching is assumed and PN is compared with Batcher's network operating in the word model.

Koppelman and Oruc [5] described a self-routing permutation network that has $O(\log^3 N)$ gate-delay and uses $O(N \log^3 N)$ gate-area. The self-routing permutation network presented by Jan and Oruc [6] has $O(\log^3 N)$ gate-delay and uses $O(N \log^2 N)$ gate-area. When VLSI-area is taken as a measure of cost (thus accounting for the area of connection links), Jan and Oruc's network uses $O(N \log^2 N)$ VLSI-area as a consequence of the results of [7], [8], [9], [10], [11] which show that a banyan network requires $O(N^2)$ VLSI-area. The same is true for Batcher's network. On the other hand, the self-routing permutation network presented in this paper has $O(\log^2 N)$ gate-delay and $O(N^2)$ VLSI-area.

Section II relates radix-sorting of N different numbers to how messages are routed in PN. In Section III, the configuration, routing scheme, concentrators, delay, and cost of PN are described. Possible shortcomings and alternative implementations of multichip concentrators for very large permutation networks are discussed in Section IV.

II. RADIX-SORTING

Let $S = [s_1 s_2 \dots s_n]$ be an $N \times N$ binary (or 0-1) matrix that represents a permutation s on N numbers $0, 1, \dots, N-1$. The way radix-sorting sorts the rows of S is very closely related to how the destination addresses of messages are sorted through the stages of PN during the implementation of permutation s . Radix-sorting sorts the rows of S recursively in ascending order as follows: in the first iteration, S is partitioned into two submatrices S_1 and S_2 , such that the most significant bits of all the rows of S_j , ($j = 1, 2$), are the same. In the

Manuscript received May 26, 1992; revised February 1, 1994.

H. Cam is with King Fahd University of Petroleum and Minerals (KFUPM), Computer Engineering Dept., Box 1826, Dhahran 31261 Saudi Arabia.

J.A.B. Fortes is with the School of Electrical Engineering, Purdue University, Electrical Engineering Bldg., West Lafayette, IN. 47907; e-mail fortes@ecn.purdue.edu.

IEEECS Log Number C95009.

¹ All logarithms are in base 2 unless stated otherwise.

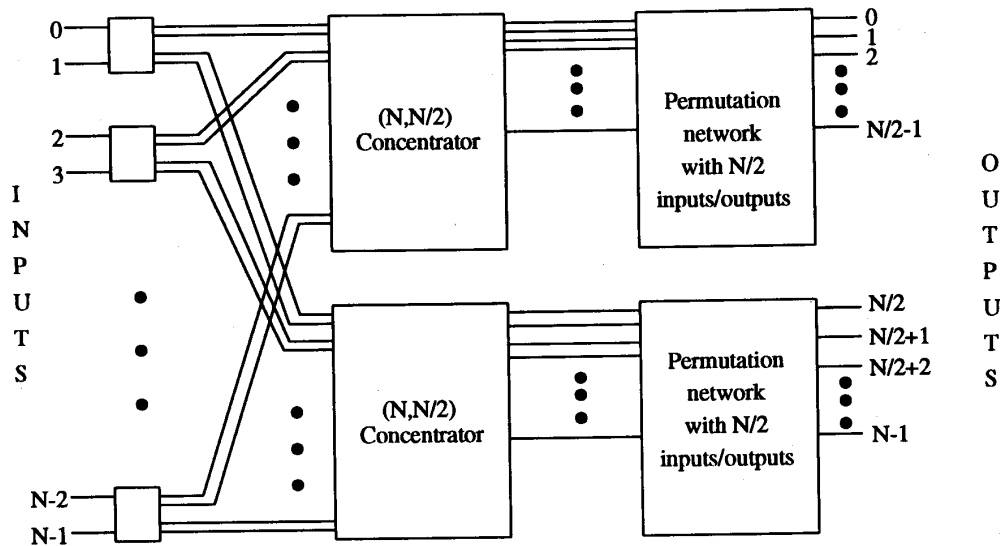


Fig. 1. The first iteration of the recursive construction of the self-routing permutation network PN.

second iteration, the submatrix S_j is partitioned into two submatrices, say S_{j1} and S_{j2} , such that the second most significant bits of all the rows of S_{jk} , ($k = 1, 2$), are the same. This partitioning process is iterated for the third most significant bit, the fourth most significant bit, and up to the least significant bit.

In a destination-tag routing scheme [12], [13] the address of the destination of the i th input is used as the routing tag for the i th input. A 2×2 switch at stage k examines the k th bit of the routing tag of the incoming input: If the k th bit is 0, then the upper output of the switch is taken; otherwise, the lower output is taken. (These k th bits are called control bits of the switch.) Using this scheme, PN sorts the N different destination addresses in the same way radix-sorting sorts the rows of S . The first stage partitions the messages into two subsets such that the most significant bits of the destination addresses of all the messages at the upper (respectively, lower) half of the second stage equal 0 (respectively, 1). Similar to the iterations of the radix-sorting, the k th stage of the network partitions every incoming subset of messages into two outgoing subsets determined by the k th most significant bits of their destination addresses.

III. SELF-ROUTING PERMUTATION NETWORK

Cube-type networks cannot realize all permutations due to "conflicts" in switches whose control bits constitute either the set $\{0, 0\}$ or the set $\{1, 1\}$. The approach of this paper replaces 2×2 switches with 2×4 switches where conflicts can never occur even if their control bits are identical. This is done at the expense of creating additional messages which are discarded by concentrators before reaching the next stage. These additional messages are herein labeled *invalid* whereas those to be permuted by the network are *valid* messages.

An (N, M) concentrator is an IN that has N inputs X_1, X_2, \dots, X_N and $M < N$ outputs Y_1, Y_2, \dots, Y_M , and can establish M disjoint paths from any set of M inputs to the M outputs [14], [15], [16]. An IN with N inputs and N outputs is called an N -hyperconcentrator if there are R disjoint paths from each set of R inputs to the first R outputs Y_1, Y_2, \dots, Y_R for any $R, 1 \leq R \leq N$ [15]. This implies that any N -hyperconcentrator can be used as an (N, M) concentrator by simply choosing the first M outputs of the hyperconcentrator as the M out-

puts of the concentrator. In this paper, the hyperconcentrator of Cormen and Leiserson [14] is used as a concentrator.

A. Network Configuration

Let $PN(N)$ denote a PN with N inputs and N outputs. It can be constructed recursively in $\log N$ iterations. The first iteration of the recursive construction is shown in Fig. 1. It consists of an input stage of $N/2$ digit-controlled 2×4 switches followed by two copies of an $(N, N/2)$ concentrator in parallel and an output stage that consists of two copies of a $PN(N/2)$ in parallel. The recursive construction of PN ends when the decomposition of permutation subnetworks results in $N/2$ switches at the output stage. (2×2 switches can be used in the last stage.) Let the $\log N$ stages of $PN(N)$ be labeled from left to right starting with 1. For $1 \leq k \leq n-1$ the stage k consists of 2^{n-1} 2×4 switches followed by $2^k(2^{n+1-k}, 2^{n-k})$ concentrators numbered from 0 to $2^k - 1$. The stage n consists of 2^{n-1} 2×2 switches. As an example, Fig. 2 illustrates $PN(16)$.

B. Routing Scheme

The network proposed here (PN) is primarily designed for bit-serial messages in a packet format. Each packet is divided into header and data sections. All bits of the data section of an invalid message have value of zero. The header contains the destination address and a so-called valid bit whose value is either 1 or 0 depending on whether the packet contains a valid or invalid message, respectively. A valid bit of 1 is "prepended" to every valid message at the input stage of PN before routing starts. The valid bits of messages are used during setup of the conducting paths of concentrators. Any concentrator of PN routes only valid messages; that is, it discards all invalid messages. Therefore, any switch at any stage of PN receives only valid messages. Two valid messages received by a switch are routed to those outputs to which the messages desire to go. Thus, two outputs of a switch transfer the incoming valid messages to concentrator(s), while the other two outputs "generate" and send invalid messages of all 0s including bit 0 as the valid bit.

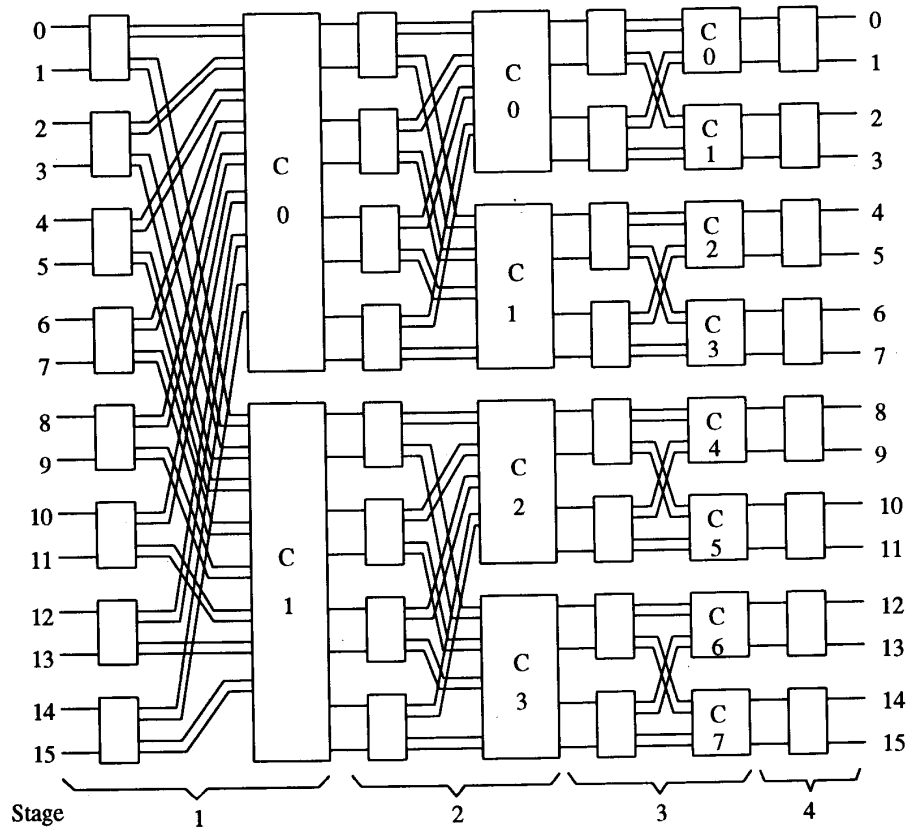


Fig. 2. The self-routing permutation network, PN(16). The letter C in the boxes stands for concentrator.

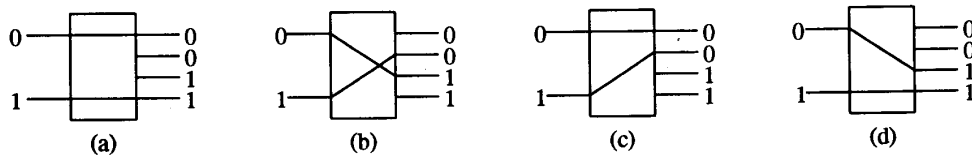


Fig. 3. The four valid states of a switch of PN. (a) Straight connection when upper and lower control bits are 0 and 1, respectively. (b) Cross connection when upper and lower control bits are 1 and 0, respectively. (c) Upper connection when both control bits equal 0. (d) Lower connection when both control bits equal 1.

The switches and the concentrators of PN are "self-setting" when inputs are presented to them. No matter what the inputs are, the switches never have a conflict because both inputs can be routed to the same desirable concentrator. In PN, a destination-tag routing scheme is used to realize any permutation. Consider a 2×4 switch at the k th stage of PN(N). The input links of a 2×4 switch are labeled by 0 and 1, while the output links are numbered 0, 0, 1, and 1 from top to bottom. As shown in Fig. 3, this switch can be set up in four different ways by examining the k th most significant bits of the destination addresses of the inputs (i.e., the control bits of the switch). If the upper control bit is 0, then the upper input is routed to the first output from top, and to the third output otherwise. If the lower control bit is 0, then the lower input is routed to the second output, and to the fourth output otherwise. If both control bits are 0 (respectively, 1), then both inputs are routed to the upper (respectively, lower) two outputs.

At the last stage of PN(N) messages arriving at the same switch cannot have the same destination address and, hence, their n th bits

form the set $\{0, 1\}$. This implies that no conflict can occur in any of these switches even if they are replaced by 2×2 switches. In a 2×2 switch, the input with control bit 0 (respectively, 1) is routed to the upper (respectively, lower) output.

C. Concentrators Used in PN

For $M = 2^m$ and $2 \leq m \leq n$, the M -hyperconcentrator, presented by Cormen and Leiserson [14], is used in PN as an $(M, M/2)$ concentrator by simply choosing the first $M/2$ outputs of the M -hyperconcentrator as the concentrator outputs. Due to the fact that any $(M, M/2)$ concentrator of PN receives exactly $M/2$ valid and $M/2$ invalid messages, every output of the concentrator carries (or transfers) a valid message to the next stage. The M -hyperconcentrator consists of $\log M$ stages of merge boxes, each of which merges two sorted sets of messages by their valid bits into one set of messages,

TABLE I

Network	Propagation delay (gate-delay)	Cost (VLSI-area)
Batcher's network	$O(\log^3 N)$	$O(N^2 \log N)$
Jan and Oruc's network	$O(\log^3 N)$	$O(N^2 \log N)$
PN (this paper)	$O(\log^2 N)$	$O(N^2)$

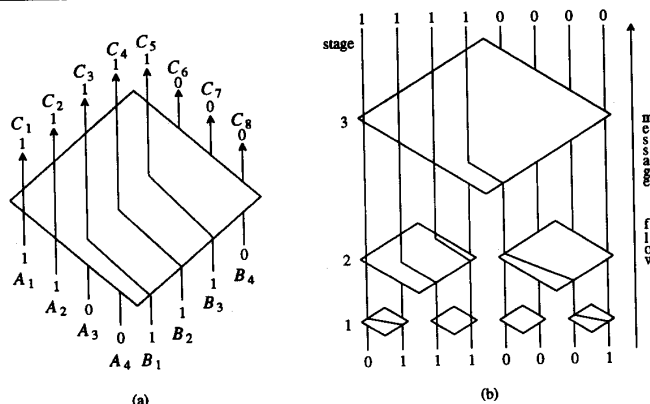


Fig. 4. [14] (a) Two sorted sets of inputs As and Bs are routed to the outputs Cs in a merge box of size 8×8 . (b) A block diagram of the 8-hyperconcentrator where inputs are at the bottom.

such that all the valid messages are routed to the first outputs of merge box. (Fig. 4 shows a merge box with eight inputs/outputs and a block diagram of the eight-hyperconcentrator.) Like Batcher's network, the hyperconcentrator of Cormen and Leiserson also uses recursive merging but, by taking the advantage of high fan-in NOR gates in nMOS and domino CMOS technologies, each merge takes only two gate-delays. The M -hyperconcentrator, therefore, routes all the valid messages in exactly $2 \log M$ gate delays.

In the previous sections, the configuration, components, and routing scheme of PN are presented. It remains to prove that PN is a self-routing permutation network.

Theorem 1: *PN is a self-routing permutation network.*

Proof: Due to 1) any switch of PN at the i th stage determining its setting dynamically by examining the i th bits of the destination addresses of the incoming messages, and 2) the hyperconcentrator of [14] setting itself up when messages are present, PN is a self-routing network.

To show that PN is also a permutation network consider the first iteration of the recursive construction of PN (shown in Fig. 1). For any given permutation, at the inputs there obviously exist N given (valid) messages with distinct destination addresses such that the most significant bits of these addresses constitute a set of $N/2$ 0s and $N/2$ 1s. The 2×4 switches at the first stage transfer all those $N/2$ messages whose destination addresses's most significant bit equal 0 (respectively, 1) to the upper (respectively, lower) concentrator. Then, each concentrator transfers all the incoming $N/2$ valid messages to the next $PN(N/2)$. Thus, the first stage of PN sends N given messages to the correct $PN(N/2)$ s. The above routing process can be repeated for every iteration of PN because PN is constructed recursively in $\log N$ iterations. In the final step, any resulting $PN(2)$ is the same as a 2×2 crossbar switch. Therefore, PN is also a permutation network, in addition to being self-routing. \square

D. Performance and Cost Analysis

In this section, performance is measured by how long it takes to realize a permutation on PN in terms of gate delays, and cost is measured as the VLSI-area (or layout area) of a physical implementation of PN. Both the cost and delay of any switch used in PN are taken as 1. Let $A_{PN}(N)$ and $D_{PN}(N)$ denote the cost and the propagation delay of $PN(N)$, respectively.

D.1. Propagation Delay of PN

From the recursive construction of $PN(N)$, the delay of PN (shown in Fig. 1) can be described by the recurrence equation:

$$D_{PN}(N) = D_{PN}(N/2) + D_C(N, N/2) + D_S,$$

where $D_C(N, N/2)$ is the delay of an $(N, N/2)$ concentrator, and D_S is the delay of a switch. Recall from Section III.C that PN uses the N -hyperconcentrator of Cormen and Leiserson as an $(N, N/2)$ concentrator and, hence, $D_C(N, N/2) = 2 \log N$ gate delays. So, the equation can be rewritten as

$$D_{PN}(N) = D_{PN}(N/2) + 2 \log N + 1.$$

It is clear that $D_{PN}(2) = 1$ because $PN(2)$ is a 2×2 switch. Thus, the solution with the boundary condition $D_{PN}(2) = 1$ is

$$D_{PN}(N) = \log^2 N + 2 \log N - 2.$$

So, a permutation through $PN(N)$ takes $O(\log^2 N)$ gate-delays.

D.2. The Cost of PN

Similar to $D_{PN}(N)$, $A_{PN}(N)$ can be described by the recurrence equation:

$$A_{PN}(N) = 2A_{PN}(N/2) + 2A_C(N, N/2) + A_{st},$$

where $A_C(N, N/2)$ is the VLSI-area of the $(N, N/2)$ concentrator, and A_{st} is the VLSI-area of $N/2$ switches and all the interconnection links at a stage of PN. Due to the fact that the N -hyperconcentrator uses $O(N^2)$ VLSI-area [14], [17], $A_C(N, N/2) = N^2$. According to [11], the VLSI-area of an N -node shuffle-exchange stage grows as $O(N^2/\log^{3/2} N)$. But, A_{st} is conservatively assumed to be equal to N^2 because $A_C(N, N/2)$ equals N^2 and, hence, is the dominant factor in determining $A_{PN}(N)$. So, when $A_C(N, N/2)$ and A_{st} are substituted in the equation we obtain

$$A_{PN}(N) = 2A_{PN}(N/2) + 2N^2 + N^2$$

The solution to this recurrence with the boundary condition $A_{PN}(2) = 1$ is

$$A_{PN}(N) = 6N^2 - 23N/2.$$

Hence, PN uses $O(N^2)$ VLSI-area.

D.3. Comparison of PN with Some Networks

Jan and Oruc [6] described a *self-routing* permutation network with the best known time and area complexities to date: $O(\log^3 N)$ gate-delay and $O(N \log^2 N)$ gate-area. The VLSI-area (instead of gate-area) of this and Batcher's network are computed below and compared with the VLSI-area of PN.

The evaluation of the cost of a network without considering the link crossovers and connection paths can make sense only if they have negligible cost compared to switch costs. However, it is shown in [8] and [10] that a $\log N$ -stage network, such as banyan network, can have VLSI-area complexity as large as $O(N^2)$ which is the same as VLSI-area of a crossbar. (But, although there is little that can be done to improve the VLSI-area of a crossbar, there are a number of parameters to save up to 50% of the VLSI-area of banyan network [7].) Batcher's network, which consists of $\log^2 N$ -stages with $N/2$ comparators at each stage, requires $O(N^2 \log N)$ VLSI-area (one can think of it as $\log N$ networks of $\log N$ stages, each network using $O(N^2)$ area). On the other hand, the cost of Jan and Oruc's network [6] which equals $O(N \log N)$ is measured in terms of elementary devices, simple binary adders, and comparators rather than in terms of VLSI-area. Therefore, it is assumed in [6] that a concentrator that contains cube network uses $O(N)$ gate-area, although it uses $O(N^2)$ VLSI-area. This implies that the network of Jan and Oruc uses $O(N^2 \log N)$ VLSI-area. The area and delay complexities of these networks are tabulated in Table I.

IV. DESIGN OF PN FOR A LARGE NUMBER OF INPUTS

As shown in Fig. 4(b) for $N = 8$ and explained in Section III.C, the hyperconcentrator of [14] is constructed recursively by cascading the merge boxes into $\log N$ stages. Note that the last merge box at the last stage of the N -hyperconcentrator has N inputs/outputs. For instance, for $N = 64K$, the last merge box of a 64K-hyperconcentrator would have 64K inputs/outputs. This may require multiple chips. However, partitioning it among several chips is not cost-effective because the bisection width of the hyperconcentrator grows linearly with the number of inputs. So, the technique of [14] is not cost-effective to build a hyperconcentrator with a large number of inputs/outputs. Therefore, PN cannot

have a very large number of inputs/outputs unless those hyperconcentrators with a very large number of inputs are built efficiently. Next, an alternative design for a Cormen and Leiserson's hyperconcentrator is discussed in case its implementation requires more than one chip. This new design enables PN to remain more cost-effective and faster than Batcher's networks.

One way of building a very large hyperconcentrator is to replace a very large merge box of [14] by Batcher's bitonic sorter or the network called merger [18], each of which can sort bitonic sequences of 0s and 1s. Of course, the larger the number of inputs of PN, the more these replacements need to take place. Although these replacements increase the cost and delay of PN, the resulting network is still better than an entire Batcher's network. A more efficient technique (than using a Batcher's sorter or merger) is described next for building a hyperconcentrator whose number of inputs is twice the number of inputs of the largest available hyperconcentrator of [14].

Let CL_k be the largest 2^k -hyperconcentrator of Cormen and Leiserson that can be built efficiently on a chip. Let S_{k+1} denote a stage of 2^k switches with 2^{k+1} inputs/output. The upper and lower inputs (respectively, outputs) of the top h th switch are connected to the stage inputs (respectively, outputs) h and $2^{k+1} - h - 1$ for $h = 0, 1, \dots, 2^k - 1$. Each 2×2 switch in the stage S_{k+1} routes the input with valid bit 1 (respectively, 0) to the upper (respectively, lower) output. It is shown below that an efficient 2^{k+1} -hyperconcentrator whose inputs form a sequence of 2^k 0s and 2^k 1s can be constructed from two CL_k s followed by S_{k+1} .

Theorem 2: *Let M be a network that consists of two copies of a CL_k in parallel followed by the stage S_{k+1} . If M receives 2^k valid messages and 2^k invalid messages, then it can be used as a 2^{k+1} -hyperconcentrator.*

Proof: Let CL_k^u and CL_k^l denote the upper and lower CL_k s of M , respectively. The valid bits of the valid messages form the one-zero sequence $1^i 0^j$ at the outputs of CL_k for $0 \leq i, j \leq 2^k$ and $i + j = 2^k$ because CL_k routes only valid messages to the first outputs. This implies that, if the one-zero output sequences of CL_k^u and CL_k^l are denoted by $1^i 0^{j_1}$ and $1^{i_2} 0^{j_2}$, respectively, then $0 \leq i_1, i_2, j_1, j_2 \leq 2^k$ and

$$i_1 + j_1 = i_2 + j_2 = 2^k. \quad (1)$$

If M receives 2^k valid messages and 2^k invalid messages, then

$$i_1 + i_2 = j_1 + j_2 = 2^k \quad (2)$$

From (1) and (2), we obtain

$$i_1 = j_2 \quad \text{and} \quad i_2 = j_1 \quad (3)$$

Equation (3) and the fact that the h th switch of S_{k+1} is connected to the inputs h and $2^{k+1} - h - 1$ imply that the valid bits of the inputs of the h th switch constitute the set $\{0, 1\}$. Therefore, no conflict occurs in any switch of S_{k+1} . Also, recall from the definition of S_{k+1} that any switch of S_{k+1} routes the message with valid bit of 1 (respectively, 0) to the upper (respectively, lower) output which is connected to the output h (respectively, $2^{k+1} - h - 1$) of S_{k+1} . Therefore, all the upper half outputs of S_{k+1} receive 1s (valid messages), while the lower half receive 0s (invalid messages). It follows that the network M can be used as a 2^{k+1} -hyperconcentrator. \square

The 2^{k+1} -hyperconcentrator of Theorem 2 is efficient because two CL_k s are utilized and a merge box of 2^{k+1} inputs/outputs is replaced by only a stage S_{k+1} of 2^{k+1} inputs/outputs instead of a sorter.

ACKNOWLEDGMENTS

The authors thank A. Yavuz Oruc for his constructive detailed comments. This research was supported in part by the U.S. Office of Naval Research under contract No. 00014-90-J-1483 and in part by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization, and was administered through the Office of Naval Research under contract No. 00014-88-k-0723.

REFERENCES

- [1] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, 1973.
- [2] K. Batcher, "Sorting networks and their applications," *AFIPS Spring Joint Computer Conf.*, vol. 32, pp. 307-314, 1968.
- [3] F.T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Computers*, vol. 34, no. 4, pp. 344-354, Apr. 1985.
- [4] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, 1992.
- [5] D.M. Koppelman and A.Y. Oruc, "A self-routing permutation network," *J. of Parallel and Distributed Computing*, vol. 10, pp. 140-151, 1990.
- [6] C. Jan and A.Y. Oruc, "Fast self-routing permutation networks," *Proc. 1991 Int'l Conf. Parallel Processing*, pp. I-263-I-269.
- [7] H.T. Szymanski, "A VLSI comparison of switch recursive banyan and crossbar interconnection networks," *Proc. 1986 Int'l Conf. Parallel Processing*, pp. 192-199.
- [8] P. Mazumder, "Evaluation of three interconnection networks for CMOS VLSI implementation," *Proc. 1986 Int'l Conf. on Parallel Processing*, pp. 200-207.
- [9] M.A. Franklin, D.F. Wann, and W.J. Thomas, "Pin limitations and partitioning of VLSI interconnection networks," *IEEE Trans. Computers*, vol. 31, no. 11, pp. 1109-1116, Nov. 1982.
- [10] M.A. Franklin, "VLSI performance comparison of banyan and crossbar communications networks," *IEEE Trans. Computers*, vol. 30, no. 4, pp. 283-291, Apr. 1981.
- [11] F.T. Leighton, M. Lepley, and G.L. Miller, "Layouts for the shuffle-exchange graph based on the complex plane diagram," *SIAM J. Alg. Disc. Meth.*, vol. 5, no. 2, pp. 202-215, June 1984.
- [12] D.H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Computers*, vol. 24, no. 12, pp. 1145-1155, Dec. 1975.
- [13] M.C. Pease, "The indirect binary n-cube multiprocessor array," *IEEE Trans. Computers*, vol. 26, no. 5, pp. 458-473, May 1977.
- [14] T.H. Cormen and C.E. Leiserson, "A hyperconcentrator switch for routing bit-serial messages," *Proc. 1986 Int'l Conf. Parallel Processing*, pp. 721-728.
- [15] L.G. Valiant, "Graph-theoretic properties in computational complexity," *JCSS*, vol. 13, no. 3, pp. 278-285, Dec. 1976.
- [16] C.D. Thompson, "Generalized connection networks for parallel processor intercommunication," *IEEE Trans. Computers*, vol. 27, no. 12, pp. 1119-1125, Dec. 1978.
- [17] T.H. Cormen, "Efficient multichip partial concentrator switches," *Proc. 1987 Int'l Conf. Parallel Processing*, pp. 525-532, 1987.
- [18] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, pp. 642-650, MIT Press, 1991.

Fast Evaluation of the Elementary Functions in Single Precision

W.F. Wong and E. Goto

Abstract — In this paper we will introduce a new method for the fast evaluation of the elementary functions in single precision based on the evaluation of truncated Taylor series using a difference method. We assume the availability of large and fast (at least for read purposes) memory. We call this method the ATA (Add-Table lookup-Add) method. As the name implies, the hardware required for the method are adders (both two/ and multi/operand adders) and fast tables. For IEEE Single Precision numbers our initial estimates indicate that we can calculate the basic elementary functions, namely reciprocal, square root, logarithm, exponential, trigonometric and inverse trigonometric functions, within the latency of two to four floating point multiplies.

Index Terms — Digital arithmetic, elementary functions, error analysis, table-based algorithms.

I. INTRODUCTION

Memory density has experienced an exponential growth in recent years. It now seems possible to use large (megabyte) tables for the purpose of speeding up elementary function computations [1][7]. In this paper we explore the limits of such an approach using a novel technique we call ATA (Add-Table lookup-Add) for computing the elementary functions in the IEEE single precision floating point [3]. The proposed ATA method will best contribute to the area of real time signal processing, and computer graphics.

II. THE ATA METHOD

Let X be a fixed point input. Our intention is for X to represent the mantissa of an IEEE Single Precision Floating Point number which is 24 bits in length. However, due to special requirements in handling the exponent, it is sometimes possible for X to take values outside the $[1, 2)$ interval in which the IEEE single precision floating number's mantissa is obliged to remain. We therefore assume X to be in different ranges depending on the function. The functions which will be considered in this paper and the ranges of the mantissa of X for each function assumed in this section are given in Table I.

Divide X into the following chunks, i.e.,

$$X = x_0 + \lambda x_1 + \lambda^2 x_2 + \lambda^3 x_3 \quad (1)$$

where $\lambda = 2^{-6}$, $x_i < 1$, $i = 1, 2, \text{ or } 3$, are 6 bits in length. On the other hand, depending on the function, we have x_0 anywhere in $[0, 4)$. Let f be a function we wish to compute and $f^{(n)}$ be the n th derivative of f . We can write the Taylor polynomial for f as follows

$$f(X) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0 + \lambda x_1)}{n!} (\lambda^2 x_2 + \lambda^3 x_3)^n \quad (2)$$

Our aim is to approximate $f(X)$ by neglecting terms smaller than λ^5 . Expanding (2) we have

Manuscript received March 15, 1993; revised January 26, 1994 and August 14, 1994.

W.F. Wong is with Dept. of Information System and Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore 0511; e-mail wongwf@iscs.nus.sg.

E. Goto is with Goto Laboratory, Institute of Physical and Chemical Research, 2-1, Hirosawa, Wako-shi, Saitama 351-01, Japan.

IEEECS Log Number C95014.