


This electronic document has been provided by the Kent State University Interlibrary Loan Department. This material may be protected by United States Copyright Law (Title 17, U.S. Code).

University of Oklahoma ILL

ILLiad TN: 435328 

ILL Number: 89127265

■■■■■■■■■■

Borrower: KSU

Print Date: Tuesday, March 27, 2012

Lending String: \*OKU,IRU,UIU,PMC EEM

Patron: Yuan, Man

Journal Title: Proceedings of the ... Sagamore  
Computer Conference on Parallel Processing

Volume: Issue:

Month/Year: 1973Pages: 147-152

Article Author:

Article Title: Batcher, K. E · STARAN/RADCAP  
Hardware Architecture

ISSN:

Imprint: [Syracuse, NY] . Dept. of Electrical & C

ILL

Call #: MICROFICHE SERIAL  
1679

Location: Engineering Library  
Microforms Area

Tues  
Apr 6-15

ODYSSEY ENABLED

Charge

Maxcost: \$50IFM

Shipping Address:

Kent State University

Libraries & Media Services

ILL

1125 Risman Drive

Kent, OH 44242-0001

Odyssey: 206.107.43.20

Email: lms-ill@kent.edu

Ariel: 131.123.20.192

Not On Shelf

Exceeds Copy Limit (Say No)

ILLiad - 50 Pg Limit

Not Found As Cited

STARAN/RADCAP HARDWARE ARCHITECTURE

Kenneth E. Batcher  
Goodyear Aerospace Corporation  
Akron, Ohio 44315

**Summary:** Hardware architecture is described for RADCAP, the operational associative array processor (AP) facility installed at Rome Air Development Center (RADC), N. Y. Basically, this facility consists of Goodyear Aerospace STARAN<sup>(a)</sup> parallel processor and various peripheral devices interfaced with a Honeywell Information Systems (HIS) 645 sequential computer, which runs under the Multics time-shared operating system. The hardware of STARAN/RADCAP is described with particular emphasis on the parallel processing elements.

Introduction

Companion papers presented at this conference describe the potential use of the RADCAP facility and its software (1) (2). The STARAN associative array (parallel) processor (3) employed in RADCAP has been modified to include a custom parallel input/output (PIO) unit and an interface to the 645 computer.

The parallel processing capability of STARAN resides in four array modules. Each array module contains 256 small processing elements (PE's). They communicate with a multi-dimensional access (MDA) memory through a "flip" network, which can permute a set of operands to allow inter-PE communication. This gives the programmer a great deal of freedom in using the processing capability of the PE's. At one stage of a program, he may apply this capability to many bits of one or a few items of data; at another stage, he may apply it to one or a few bits of many items of data.

The remainder of this paper deals with the MDA memories, the STARAN array modules, and the STARAN/RADCAP elements.

Multi-Dimensional Access (MDA) Memories

A common implementation of associative processing is to treat data in a bit-sequential manner. A small one-bit PE (processing element) is associated with each item or word of data in the store, and the set of PE's accesses the data store in bit-slices; a typical operation is to read Bit *i* of each data word into its associated PE or to write Bit *i* from its associated PE.

The memory for such an associative processor could be a simple random-access memory with the data rotated 90 deg, so that it is accessed by bit-slices instead of by words. Unfortunately, in most applications, data come in and leave the processor as items or words instead of as bit-slices. Hence,

rotating the data in a random-access memory complicates data input and output.

To accommodate both bit-slice accesses for associative processing and word-slice accesses for STARAN input/output (I/O), the data are stored in a multi-dimensional access (MDA) memory (Figure 1). It has wide read and write busses for parallel access to a large number (256) of memory bits. The write-mask bus allows selective writing of memory bits. Memory accesses (both read and write accesses) are controlled by the address and access mode control inputs; the access mode selects a stencil pattern of 256 bits, while the address positions the stencil in memory.

For many applications, the MDA memory is treated as a square array of bits, 256 words with 256 bits in each word. The bit-slice access mode (Figure 2A) is used in the associative operations

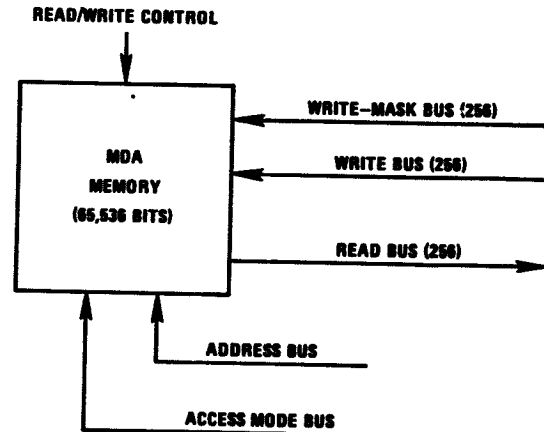


Figure 1. Multi-Dimensional Access Memory

A - BIT-SLICE ACCESS MODE      B - WORD ACCESS MODE

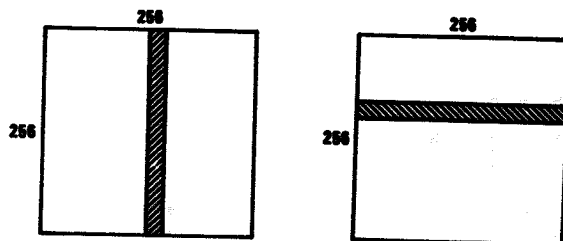


Figure 2. Bit-Slice and Word Access Modes

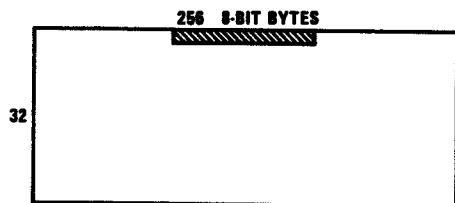
(a) TM, Goodyear Aerospace Corporation, Akron, Ohio.

to access one bit of all words in parallel, while the word access mode (Figure 2B) is used in the I/O operations to access several or all bits of one word in parallel.

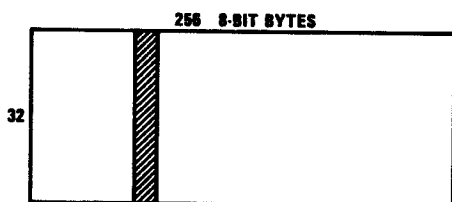
The MDA memory structure is not limited to a square array of 256 by 256. For example, the data may be formatted as records with 256 8-bit bytes in each record. Thirty-two such records can be stored in an MDA memory and accessed several ways. To input and output records, one can access 32 consecutive bytes of a record in parallel (Figure 3A). To search key fields of the data, one can access the corresponding bytes of all records in parallel (Figure 3B). To search a whole record for the presence of a particular byte, one can access a bit from each byte in parallel (Figure 3C).

The MDA memories in the RADCAP array modules are bipolar. They exhibit read cycle times of less than 150 nsec and write cycle times of less than 250 nsec.

A - ACCESS TO 32 CONSECUTIVE BYTES OF A RECORD



B - ACCESS TO CORRESPONDING BYTES OF ALL RECORDS



C - ACCESS TO ONE BIT OF EVERY BYTE IN A RECORD

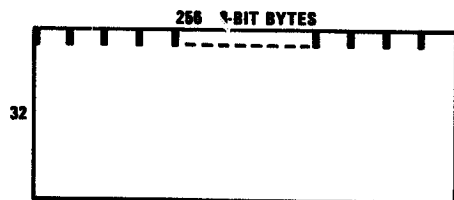


Figure 3. Accessing 256-Byte Records

### STARAN Array Modules

A STARAN array module (Figure 4) contains a MDA memory communicating with three 256-bit registers (M, X, and Y) through a flip (permutation) network. One may think of an array module as having 256 small processing elements (PE's), where a PE contains one bit of the M register, one bit of the X register, and one bit of the Y register.

The M register drives the write mask bus of the MDA memory to select which of the MDA memory bits are modified in a masked-write operation. The MDA memory also has an unmasked-write operation that ignores M and modifies all 256 accessed bits. The M register can be loaded from the other components of the array module.

In general, the logic associated with the X register can perform any of the 16 Boolean functions of two variables; that is, if  $x_i$  is the state of the  $i^{\text{th}}$  X-register bit, and  $f_i$  is the state of the  $i^{\text{th}}$  flip network output, then:

$$x_i \leftarrow \phi(x_i, f_i) \quad (i = 0, 1, \dots, 255)$$

where  $\phi$  is any Boolean function of two variables. Similarly, the logic associated with the Y-register can perform any Boolean function:

$$y_i \leftarrow \phi(y_i, f_i) \quad (i = 0, 1, \dots, 255)$$

where  $y_i$  is the state of the  $i^{\text{th}}$  Y-register bit. The programmer is given the choice of operating X alone, Y alone, or X and Y together.

If X and Y are operated together, the same Boolean function,  $\phi$ , is applied to both registers:

$$x_i \leftarrow \phi(x_i, f_i)$$

$$y_i \leftarrow \phi(y_i, f_i)$$

The programmer also can choose to operate on X selectively using Y as a mask:

$$x_i \leftarrow \phi(x_i, f_i) \quad (\text{where } y_i = 1)$$

$$x_i \leftarrow x_i \quad (\text{where } y_i = 0)$$

Another choice is to operate on X selectively while operating on Y:

$$x_i \leftarrow \phi(x_i, f_i) \quad (\text{where } y_i = 1)$$

$$x_i \leftarrow x_i \quad (\text{where } y_i = 0)$$

$$y_i \leftarrow \phi(y_i, f_i)$$

In this case, the old state of Y (before modification by  $\phi$ ) is used as the mask for the X operation.

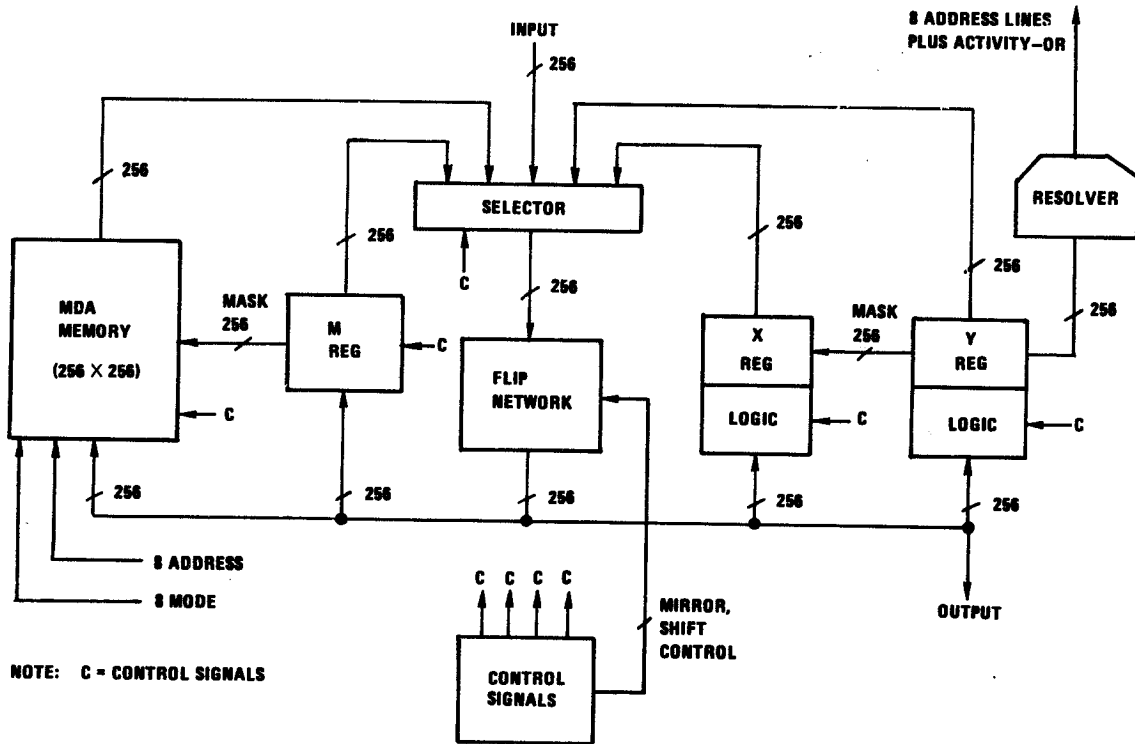


Figure 4. STARAN Array Module

For a programming example, the basic loop of an unmasked add fields operation is selected. This operation adds the contents of a Field A of all memory words to the contents of a Field B of the words and stores the sum in a Field S of the words. For n-bit fields, the operation executes the basic loop n times. During each execution of the loop, a bit-slice (a) of Field A is read from memory, a bit-slice (b) of Field B is read, and a bit-slice (s) of Field S is written into memory. The operation starts at the least significant bits of the fields and steps through the fields to the most significant bits. At the beginning of each loop execution, the carry (c) from the previous bits is stored in Y and X contains zeroes:

$$x_i = 0$$

$$y_i = c_i$$

The loop has four steps:

**Step 1:** Read Bit-slice a and exclusive-or ( $\oplus$ ) it to X selectively and also to Y:

$$x_i \leftarrow x_i \oplus y_i a_i$$

$$y_i \leftarrow y_i \oplus a_i$$

The states of X and Y are now:

$$x_i = a_i c_i$$

$$y_i = a_i \oplus c_i$$

**Step 2:** Read Bit-slice b and exclusive-or it to X selectively and also to Y:

$$x_i \leftarrow x_i \oplus y_i b_i$$

$$y_i \leftarrow y_i \oplus b_i$$

Registers X and Y now contain the carry and sum bits:

$$x_i = a_i c_i \oplus a_i b_i \oplus b_i c_i = c'_i$$

$$y_i = a_i \oplus b_i \oplus c_i = s_i$$

**Step 3:** Write the sum bit from Y into Bit-slice s and also complement X selectively:

$$s_i \leftarrow y_i$$

$$x_i \leftarrow x_i \oplus y_i$$

The states of X and Y are now:

$$x_i = c_i' \oplus s_i$$

$$y_i = s_i$$

**Step 4:** Read the X-register and exclusive-or it into both X and Y:

$$x_i \leftarrow x_i \oplus x_i$$

$$y_i \leftarrow y_i \oplus x_i$$

This clears X and stores the carry bit into Y to prepare the registers for the next execution of the loop:

$$x_i = 0$$

$$y_i = c_i'$$

Step 3 takes less than 250 nsec, while Steps 1, 2, and 4 each take less than 150 nsec. Hence, the time to execute the basic loop once is less than 700 nsec. If the field length is 32 bits, the add operation takes less than 22.4 microsec plus a small amount of setup time. The operation performs 256 additions in each array module. This amounts to 1024 additions, if all four array modules are enabled, to achieve a processing power of approximately 40 MIPS (million-instructions-per-second).

The array module components communicate through a network called the flip network. A selector chooses a 256-bit source item from the MDA memory read bus, the M register, the X register, the Y register, or an outside source. The bits of the source item travel through the flip network, which may shift and permute the bits in various ways. The permuted source item is presented to the MDA memory write bus, M register, X register, Y register, and an outside destination.

The permutations of the flip network allow inter-PE communication. A PE can read data from another PE either directly from its registers or indirectly from the MDA memory. One can permute the 256-bit data item as a whole or divide it into groups of 2, 4, 8, 16, 32, 64, or 128 bits and permute within groups.

The permutations allowed include shifts of 1, 2, 4, 8, 16, 32, 64, or 128 places. One also can mirror the bits of a group (invert the left-right order) while shifting it. A positive shift of mirrored data is equivalent to a negative shift of the unmirrored data. To shift data a number of places, multiple passes through the flip network may be required. Mirroring can be used to reduce the number of passes. For example, a shift of 31 places can be done in two passes: mirror and shift 1 place on the first pass, and then remirror and shift 32 places on the second pass.

The flip network permutations are particularly useful for fast-fourier transforms (FFT's). A  $2^n$  point FFT requires  $n$  steps, where each step pairs the  $2^n$  points in a certain way and operates on the two points of each pair arithmetically to form two new points. The flip network can be used to rearrange the pairings between steps. Bitonic sorting (4) and other algorithms (5) also find the permutations of the flip network useful.

Each array module contains a resolver reading the state of the Y register. One output of the resolver (activity-or) indicates if any Y bit is set. If some Y bits are set, the other output of the resolver indicates the index (address) of the first such bit. Since the result of an associative search is marked in the Y register, the resolver indicates which if any words respond to the search.

#### STARAN/RADCAP Elements

Each of the four array modules in STARAN/RADCAP (Figure 5) contains an assignment switch that connects its control inputs and data inputs and outputs to AP (associative processor) control or the PIO (parallel input/output) module.

The AP control unit contains the registers and logic necessary to exercise control over the array modules assigned to it. It receives instructions from the control memory and can transfer 32-bit data items to and from the control memory. Data busses communicate with the assigned array modules. The busses connect only to 32 bits of the 256-bit-wide input and output ports of the array modules (Figure 4), but the permutations of the array module flip networks allow communication with any part of the array. The AP control sends control signals and MDA memory addresses and access modes to the array modules and receives the resolver outputs from the array modules.

Registers in the AP control include:

1. An instruction register to hold the 32-bit instruction being executed.
2. A program status word to hold the control memory address of the next instruction to be executed and the program priority level.
3. A common register to hold a 32-bit search command, an operand to be broadcast to the array modules, or an operand output from an array module.
4. An array select register to select a subset of the assigned array modules to be operated on.
5. Four field pointers to hold MDA memory addresses and allow them to be incremented or decremented for stepping through the bit-slices of a field, the words of a group, etc.
6. Three counters to keep track of the number of executions of loops, etc.

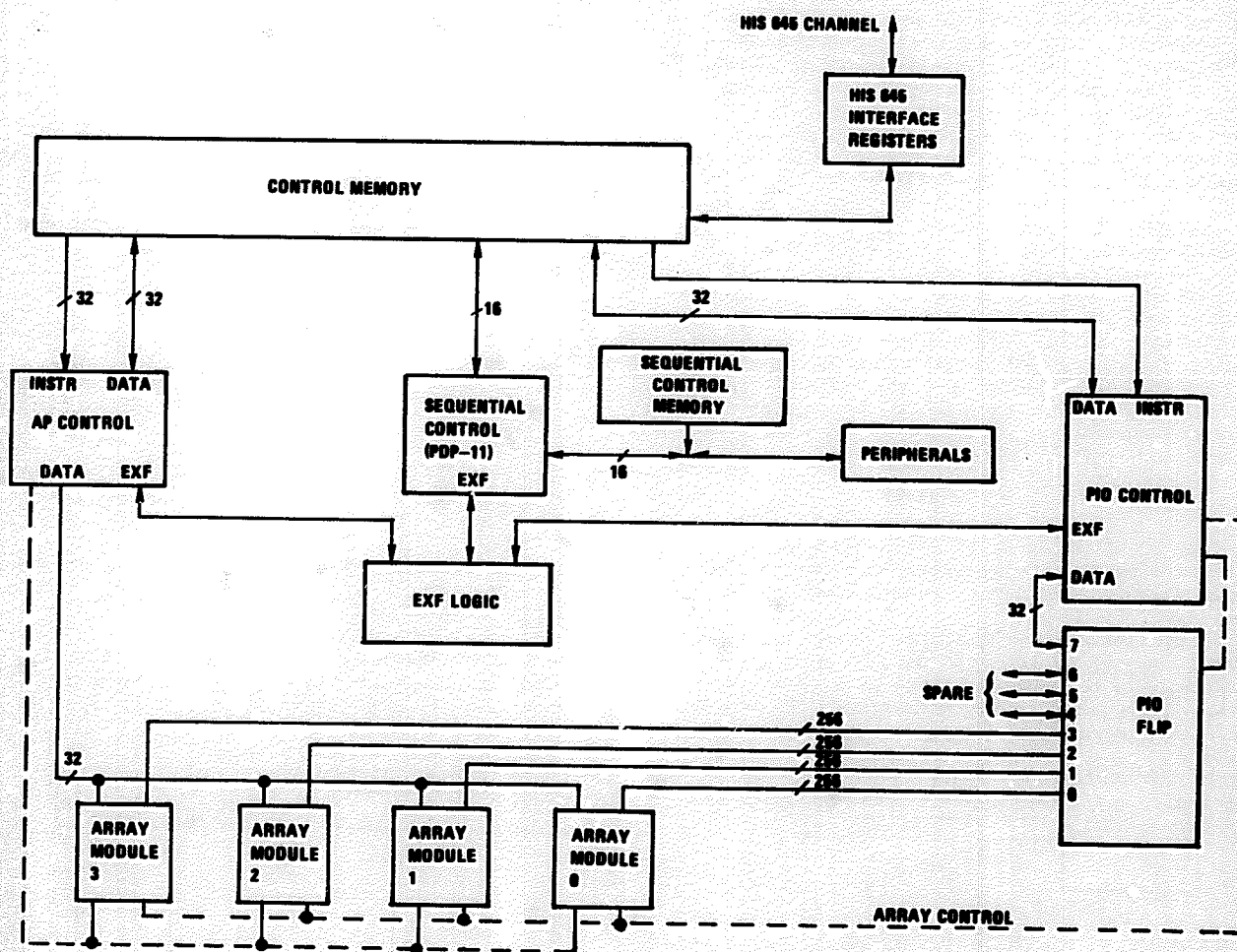


Figure 5. STARAN/RADCAP Block Diagram

7. A data pointer to allow stepping through a set of operands in control memory.

8. Two access mode registers to hold the MDA memory access modes.

The parallel input/output (PIO) module contains a PIO flip network and PIO control unit (Figure 5). It is used for high bandwidth I/O and inter-array transfers.

The PIO flip network permutes data between eight 256-bit ports. Ports 0 through 3 connect to the four array modules through buffer registers. Port 7 connects to a 32-bit data bus in the PIO control through a fan-in, fan-out switch. Ports 4, 5, and 6 are spare ports intended for future connections to high bandwidth peripherals, such as parallel-head disk stores, sophisticated displays, and radar video channels. The spare ports also could be used to handle additional array modules. High bandwidth inter-array data transfers up to 1024

bits in parallel are handled by permuting data between Ports 0, 1, 2 and 3. Array I/O is handled by permuting data between an array module port and an I/O port. The PIO flip network is controlled by the PIO control unit.

The PIO control unit controls the PIO flip network and the array modules assigned to it. While AP control is processing data in some array modules the PIO control can input and output data in the other array modules. Since most of the registers in the AP control are duplicated in PIO control, it can address the array modules associatively.

The control memory holds AP control programs, PIO control programs, and microprogram sub-routines. To satisfy the high instruction fetch rate of the control units (up to 7.7 million instructions per second), the control memory has five banks of bipolar memory with 512 32-bit words in each bank. Each bank is expandable to 1024 words. To allow for storage of large programs, the control memory

also has a 16K-word core memory with a cycle time of 1 microsec. The core memory can be expanded to 32K words. Usually the main program resides in the core memory and the system microprogram sub-routines reside in bipolar storage. For flexibility, users are given the option of changing the storage allocation and dynamically paging parts of the program into bipolar storage.

A Digital Equipment Corporation (DEC) PDP-11 minicomputer is included to handle the peripherals, control the system from console commands, and perform diagnostic functions. It is called sequential control to differentiate it from the STARAN parallel processing control units. The sequential control memory of 16K 16-bit words is augmented by a 8K X 16-bit "window" into the main control memory. By moving the window, sequential control can access any part of control memory. The window is moved by changing the contents of an addressable register.

The STARAN/RADCAP peripherals include a disk, card reader, line printer, paper-tape reader/punch, console typewriter, and a graphics console.

Synchronization of the three control units (AP control, sequential control, and PIO control) is maintained by the external function (EXF) logic. Control units issue commands to the EXF logic to cause system actions and read system states. Some of the system actions are: AP control start/stop/reset, PIO control start/stop/reset, AP control interrupts, sequential control interrupts, and array module assignment.

RADCAP connects to a common peripheral channel of a 645 computer. Channel characters are 6 bits wide. Instead of interfacing the channel to one of the three control units in RADCAP, the channel interface is assigned a set of control memory addresses so it can be addressed by any control unit. The interface has a 30-character first-in, first-out, (FIFO) queue to buffer the data transfer between the two machines. To reduce the number of queue accesses, the control units transfer queue data by character-pairs, 12 bits at a time.

#### References

- (1) J. D. Feldman and O. A. Reimann, RADCAP: An Operational Parallel Processing Facility, GER-15946, Goodyear Aerospace Corporation and Rome Air Development Center (22 August 1973).
- (2) E. W. Davis, Jr., STARAN/RADCAP System Software, GER-15948, Goodyear Aerospace Corporation (22 August 1973).
- (3) K. E. Batcher, "Flexible Parallel Processing and STARAN," 1972 WESCON Technical Papers, Session 1.
- (4) K. E. Batcher: "Sorting Networks and Their Applications," 1968 Spring Joint Computer Conference, AFIPS Proceedings, Vol 32, pp 307-314.
- (5) H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, Vol C-20, No. 2, February 1971, pp 153-161.