

STARAN parallel processor system hardware

by KENNETH E. BATCHER

Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

The parallel processing capability of STARAN* resides in n array modules ($n \leq 32$). 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, the other elements of STARAN, and the results of certain application studies.

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 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 STARAN array modules are bipolar. They exhibit read cycle times of less than 150 nsec and write cycle times of less than 250 nsec.

STARAN ARRAY MODULES

A STARAN array module (Figure 4) contains an 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;

* TM, Goodyear Aerospace Corporation, Akron, Ohio.

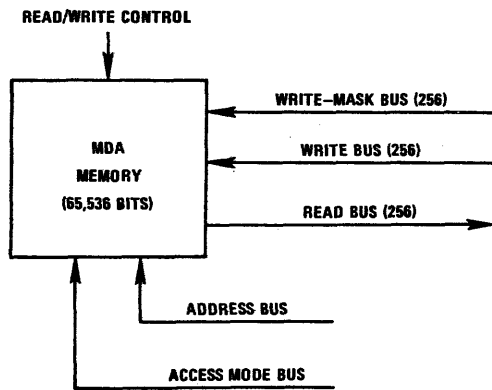


Figure 1—Multi-dimensional access memory

that is, if x_i is the state of the i th X-register bit, and f_i is the state of the i th flip network output, then:

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

where ϕ 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 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, ϕ , is applied to both registers:

$$\begin{aligned} x_i &\leftarrow \phi(x_i, f_i) \\ y_i &\leftarrow \phi(y_i, f_i) \end{aligned}$$

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

$$\begin{aligned} x_i &\leftarrow \phi(x_i, f_i) && \text{(where } y_i=1) \\ x_i &\leftarrow x_i && \text{(where } y_i=0) \end{aligned}$$

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

$$\begin{aligned} x_i &\leftarrow \phi(x_i, f_i) && \text{(where } y_i=1) \\ x_i &\leftarrow x_i && \text{(where } y_i=0) \\ y_i &\leftarrow \phi(y_i, f_i) \end{aligned}$$

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

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 ready from memory, a bit-slice (b) of Field B is read, and a bit-slice (s) of Field S is written

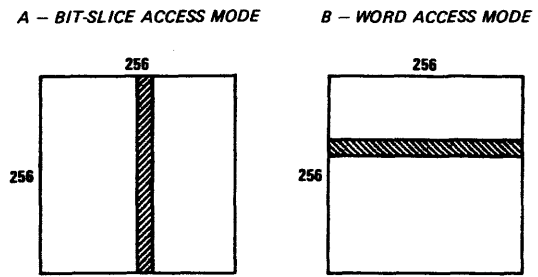


Figure 2—Bit-slice and word access modes

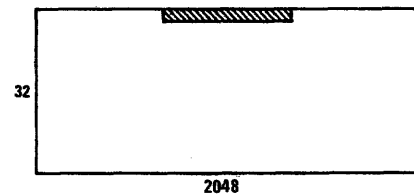
into memory. The operation starts at the least significant bits of the fields and steps through the fields to the most significant bits. At the beginning of each loop execution, the carry (c) from the previous bits is stored in Y, and X contains zeroes:

$$\begin{aligned} x_i &= 0 \\ y_i &= c_i \end{aligned}$$

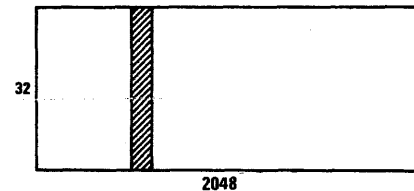
The loop has four steps:

Step 1: Read Bit-slice a and exclusive-or (\oplus) it to X selec-

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

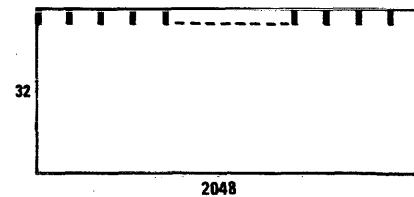


Figure 3—Accessing 256-byte records

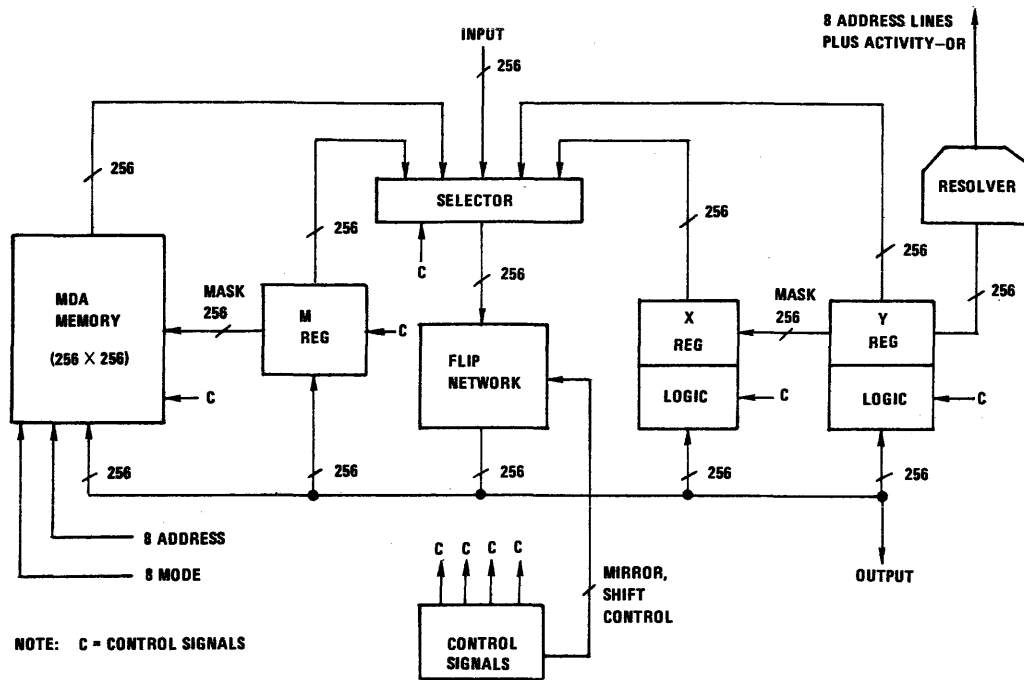


Figure 4—STARAN array module

tively 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:

$$x_i \leftarrow x_i \oplus x_i$$

$$y_i \leftarrow y_i \oplus x_i$$

$$x_i = 0$$

$$y_i = c'_i$$

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

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 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 (2) and other algorithms (3) 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.

OTHER STARAN ELEMENTS

Figure 5 is a block diagram of a typical STARAN system with four array modules. Each array module 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 comparand, 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.
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 for 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 subroutines. To satisfy the high instruction fetch rate of the control units (up to 7.7 million instructions per second), the control

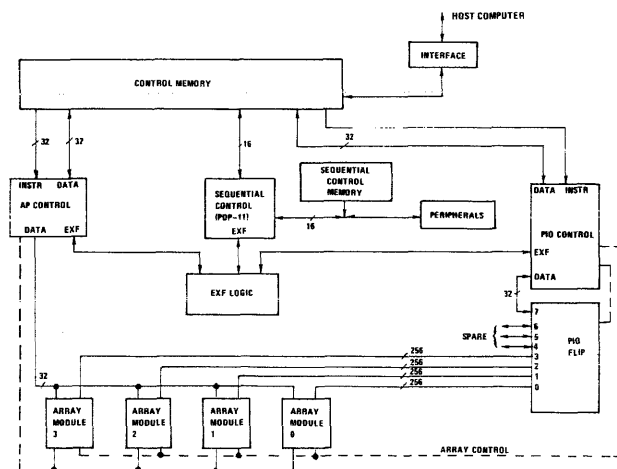


Figure 5—Typical STARAN block diagram

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 subroutines 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×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 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.

The design of STARAN allows it to be connected to other computers (host computers) as a special-purpose peripheral. The interface can take many different forms. One could connect to an I/O channel of the host. Alternately, one could connect to the memory bus of the host so that it can address STARAN memory directly and/or allow STARAN to address its memory directly. For example, the STARAN at Rome Air Development Center (4) is connected to an I/O channel of a Honeywell HIS-645 computer. At Goodyear Aerospace, another STARAN is interfaced to the direct memory access port of an XDS 5 computer.

APPLICATIONS

Several representative application areas—fast Fourier transforming, sonar post-processing, string search, file processing, and air traffic control—are discussed below. Other application-oriented work which has been performed under contract to various government agencies, include image processing, data management, position locating and reporting, bulk filtering and radar processing.

Fast fourier transform

The Fast Fourier Transform (FFT) is a basic operation in digital signal processing which is being widely used in

the real-time processing of radar and sonar signals. The structure of the FFT algorithm is such that it can be segmented into many similar concurrent operations. Parallel implementation of the FFT can provide orders of magnitude speed increases over sequential computer execution times. The organization of STARAN lends itself to efficient manipulation of data in the FFT.

The Air Force supplied real radar data (on tapes) to GAC to be transformed by the STARAN system. A 512-point, 16-bit FFT was performed on this real data in 2.7 milliseconds using only two MDA arrays. A 1024-point transform on real input data could be performed in about 3.0 milliseconds using all four arrays available at GAC's STARAN evaluation and training facility. For comparison purposes, the following is a list of reported execution times for a 1024-point, real input, FFT:

Sequential Computers

XDS Sigma 5	660 msec
IBM 360/67	446 msec
UNIVAC 1108	190 msec (complex)
UNIVAC 1108 (with array processor attachment)	29.2 msec (complex)

Special Purpose FFT Systems

Time/Data 90 System	28 msec
ELSYTEC 306/HFFT	18 msec
SPECTRA SYSTEM '900'	9.2 msec

Sonar post-processing

Sensor data processing can be split into two major categories—signal processing and post-processing. Signal processing is the area of the system where operations such as the FFT are performed; post-processing involves the sorting and editing of the signal processor output data to determine tactical information such as whether a real target is in the coverage area and where the target is.

The job of sorting the spectral lines that result from the FFT operations is a formidable task, especially in a multi-sensor case. The trend has been for increasing the sensitivity of signal processing systems. The acoustic signal line sorting task that accompanies any increased sensitivity can be staggering. For instance, a 6 db improvement in sensitivity, in a classified Navy sonar system, would result in increasing the target load by a factor of 16 and the computer processing load by a factor of 250 or more.

A digital sonar signal processing system, under development at the Naval Air Development Center (NADC), requires that subroutines operate on the target spectral lines (outputs from an FFT) and other input data to form outputs suitable for later use in classification algorithms. Since the system is a multi-sensor system, these subroutines must process a very large volume of data in real-time. The content addressability feature of STARAN provides the potential for significant performance gains due to the

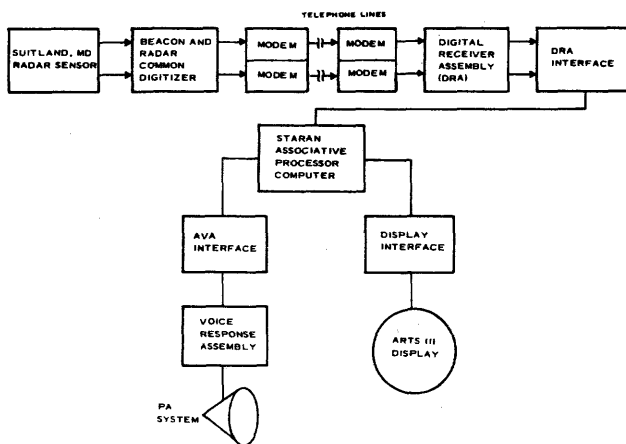


Figure 6—TRANSPRO '72 demonstration system

requirement for many searches in these post-processing subroutines.

As a consequence of this potential improvement, NADC issued a contract to GAC to assess the comparative run times for the STARAN versus a large-scale conventional computer (the CDC-6600). NADC-developed algorithms for the most time consuming operations in the post-processor system were programmed on the STARAN computer. Real data was then processed on both the STARAN and, by NADC, on a CDC-6600.

The STARAN executed the programs, using the real data, 200 times faster than the CDC-6600.

String search

A processing function used by several agencies for locating specific character strings (such as place names) in textual information, was developed for STARAN and tested on a sample data base. The same function was executed on a conventional computer (Sigma 5) for a timing comparison. The STARAN solution ran 100 times faster. This function is also applicable to nondefense applications such as patent, legal, and chemical information searches where cost of search may be a limiting parameter.

File processing

A personnel record file was used as a sample data base for demonstrating multiple-key searches. STARAN and a

parallel-head disk were used for the demonstration. This work demonstrated that a query, simple or complex, can be processed in less than 120 milliseconds and that several queries may be batched and processed in the same processing time period. The simplicity of the software for retrieval and update was also demonstrated.

Air traffic control

In May 1972 a complete terminal automation system was demonstrated at the TRANSPRO '72 exhibit at Dulles International Airport and later at private showings in Washington and Boston. Live radar and beacon sensor data were provided from the FAA site at Suitland, Maryland. The complete system is shown in Figure 6. The following ATC functions were demonstrated: beacon tracking, radar tracking, conflict prediction, conflict resolution, display processing, automatic voice advisory (AVA), and terrain avoidance.

The TRANSPRO demonstration illustrated the use of STARAN in a full repertoire of terminal automation functions including advanced features such as automatic track initiation of all the aircraft, automatic tag placement on the display, and automatic handoff from sector to sector. The live targets were supplemented with 256 simulated targets so that up to 400 targets (representative of larger terminal densities) could be provided.

Average execution times for the most important functions were:

conflict prediction	90 msec
conflict resolution	25 msec
tracking	100 msec
display processing	160 msec

The entire ATC program executed in less than 5 percent of real-time.

REFERENCES

1. Batcher, K. E., "Flexible Parallel Processing and STARAN," 1972 WESCON Technical Papers, Session 1.
2. Batcher, K. E., "Sorting Networks and Their Applications," 1968 Spring Joint Computer Conference, AFIPS Proceedings, Vol. 32, pp. 307-314.
3. Stone, H. S., "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, Vol. C-20, No. 2, February 1971, pp. 153-161.
4. Feldman, J. D., RADCAP: An Operational Parallel Processing Facility, Goodyear Aerospace Corporation.