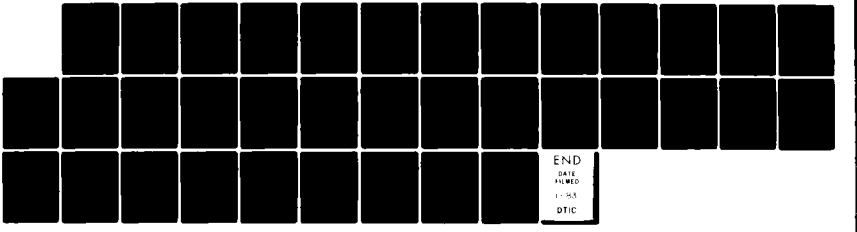
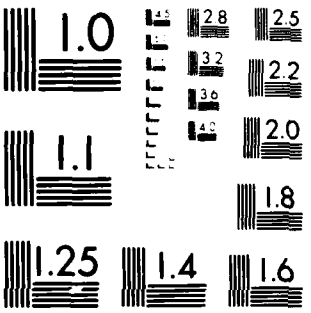
AD-A123 028 CONSTRUCTION AND EVALUATION OF CONTENT ADDRESSABLE MEMORIES(U) MASSACHUSETTS UNIV AMHERST C C FOSTER JUN 82 ARO-16088.8-EL DAAG29-79-G-0046 1/1

UNCLASSIFIED F/G 9/1 NL





(12)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1 REPORT NUMBER: 16088.8-EL	2 GOVT ACCESSION NO: AD A123028
3 RECIPIENT'S CATALOG NUMBER:	
4 TITLE (and Subtitle): Construction and Evaluation of Content Addressable Memories	5 TYPE OF REPORT & PERIOD COVERED: Final: 1 Jan 79 - 30 Jun 82
6 PERFORMING ORG. REPORT NUMBER:	
7 AUTHOR(s): Caxton C. Foster	8 CONTRACT OR GRANT NUMBER(s): DAAG29 79 G 0040
9 PERFORMING ORGANIZATION NAME AND ADDRESS: University of Massachusetts, Amherst, MA 01003	10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11 CONTROLLING OFFICE NAME AND ADDRESS: [illegible]	12 REPORT DATE: JUNE Dec 82
	13 NUMBER OF PAGES: 38
MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15 SECURITY CLASS (of this report): [illegible]
	15a DECLASSIFICATION DOWNGRADING SCHEDULE:

DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE JAN 4 1983 S D B

DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

KEY WORDS (Continue on reverse side if necessary and identify by block number)

content addresable memories
computers

ABSTRACT (Continue on reverse side if necessary and identify by block number)

The work on this grant has fallen into four distinct phases. First of all, we constructed a small (320-word) Content Addressable Memory (CAM) from commercially available components. Second, we selected a number of problems intended to span the spectrum of CAM applications and programmed these on our small machine. Third, we analyzed the strengths and weaknesses of our small machine as displayed in the solution of these problems. Finally, we designed, but have not yet constructed, a full-scale CAM that retained the strengths of our small machine and incorporated features intended to overcome its weaknesses. In parallel with the above work,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

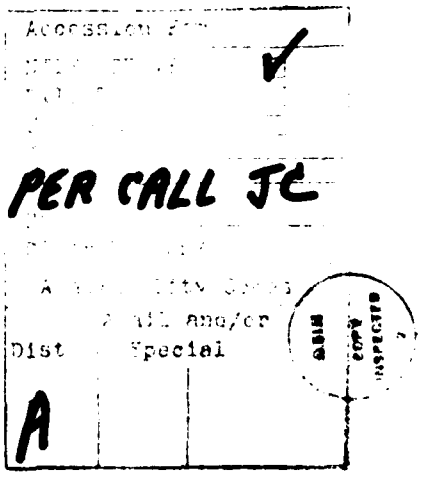
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060

20. ABSTRACT CONTINUED

in an attempt to aid in the selection of representative problems for study, we have spent a considerable amount of effort investigating the theoretical capabilities of CAMs.



CONSTRUCTION AND EVALUATION
OF
CONTENT ADDRESSABLE MEMORIES

FINAL REPORT

CAXTON C. FOSTER

JUNE 1982

U. S. ARMY RESEARCH OFFICE

GRANT DAAG 29-7[illegible]-G-0046

UNIVERSITY OF MASSACHUSETTS

List of Participating Scientific Personnel:

Caxton C. Foster

Alfred Hough

Steven Levitan

William Verts

Charles Weems

FINAL REPORT

Construction and Evaluation of Content Addressable Memories

ARO Grant DAAG 29-79-C-0046

Caxton C. Foster
University of Massachusetts
Amherst, Massachusetts 01003

INTRODUCTION

The work on this grant has fallen into four distinct phases. First of all, we constructed a small (320-word) Content Addressable Memory (CAM) from commercially available components. Second, we selected a number of problems intended to span the spectrum of CAM applications and programmed these on our small machine. Third, we analyzed the strengths and weaknesses of our small machine as displayed in the solution of these problems. Finally, we designed, but have not yet constructed, a full-scale CAM that retained the strengths of our small machine and incorporated features intended to overcome its weaknesses.

In parallel with the above work, in an attempt to aid in the selection of representative problems for study, we have spent a considerable amount of effort investigating the theoretical capabilities of CAMs.

Phase One: Construction

When we initially applied for this grant, we intended to construct a modest-sized CAM from standard TTL chips. We soon discovered that Semionics Associates was building what they called a Recognition Memory (REM) card that was not too far from what we wanted. In the interest of saving time and money and energy, we decided to go with the REM cards even though we knew that they were not ideal.

Each REM card contains 16 content addressable cells of 256 bytes (2048 bits) each. We purchased twenty of these cards giving us 320 memory cells. These were connected to a Cromemco microcomputer with a Decwriter and two disks and assorted other peripherals. Each REM card outputs a single bit that is zero if no words are responders and one if one or more words are responders. We connected these SOME/$\overline{\text{NONE}}$ bits to a priority encoder so that in one instruction time we could identify which REM card was the "first" one containing a responder. To then locate which word on the card was the first responder required a slow cell by cell scan of the sixteen cells on the card.

The other operation that was missing from the REM cards was the "count responders" instruction. To perform this we had to write the tag bits into an unused memory bit (we called the FLAG), and then in serial fashion go through each word and count the number of FLAGs equal to one.

The REM cards have many bits per word and only a few words per card. In a very real sense, each word of a CAM behaves like a processor and obviously one can not achieve more parallelism than there are processors. In our machine,

if we wanted to be able to content address more than 320 items, we had to conceptually divide our long (256 byte) words in half (or in thirds or quarters) and store two or more items in each word. Then, when we wanted to do a search we would have to "search the first set of items", then "search the second set", and so on, repeating the search as many times as required to examine all the items packed into each word.

While this repeated search was perfectly acceptable for an exploratory system, it led to an N-fold reduction in the potential speed up which would be absolutely devastating in a real-world application.

Consequently, our strongest conclusion as a result of this study is that "two shorter words are more valuable than one longer word." -- provided, of course, that they don't get too short or that there exists a way to couple two or more short words together when one really needs a long word. Two other conclusions may be drawn on the basis of our experience with the REM cards. The first is that the hardware must provide a convenient and rapid means of discovering whether or not any words are "responders". That is, there must be a SOME/NONE line that continuously reflects the OR of all the tag bits and an instruction that can do a conditional branch on the state of that line.

The other major recommendation is that there must exist a convenient non-sequential method of counting responders. A count of responders is very useful for guiding the higher levels of a search where you might choose to employ one strategy if there are only ten responders, and an entirely different one if there are one thousand. But, while this count of responders is very useful, it is not used with anything like the frequency that "branch on zero responders" is used. Consequently, the count responder operation may be considerably slower than the branch on zero responders.

Phase 2: Selection of Test Programs

There exist no widely recognized kernels of parallel programs that every CAM should execute rapidly. A person desiring to exercise a CAM must fall back on his or her own intuition and select test programs that meet one or more of the following criteria:

1. The "heart" of the problem is "obviously" one of great general interest and widespread applicability.
2. The problem is an abstraction of a real problem which is strongly compute-bound in a von-Neumann machine.
3. The problem can conceivably be completed in one semester.
4. The problem has inherent interests of its own.

Attached to this report there is a list of published and unpublished studies that we have done on this grant, and copies of all the papers.

Speaking generally, the availability of a CAM enables one to approach the solution of a problem in ways that would be rejected out of hand in a von-Neumann organization. Take for example, the solution of simple substitution cyphers reported in (4). There is no great trick to solving such puzzles; they are published in most daily newspapers and provide an enjoyable half-hour

diversion for the afficionado. You make some guesses about the value some ciphertext symbol might have, substitute it throughout the cryptogram and make further guesses on the basis of the emerging pattern about further letters. But this "guessing" process is highly directed by an excellent pattern-matching computer behind your eyes, and this approach is classically very hard to implement on a computer. Storing and searching a dictionary of several thousand common English words, Wall found that the CDC 6600 might take up to several hours to solve a single cryptogram. The same approach implemented on a (simulated) CAM could solve the same cryptogram in approximately half a second. Suddenly, an "obvious" approach that had been completely impractical becomes very attractive.

Consider also the identification of postage stamps. This is a problem that is representative of many database query systems. The user approaches the machine with a stamp in hand hoping to identify it. He can readily observe its size, shape, denomination, major color, country of origin, and perhaps give some description of the scene depicted on the stamp. What he wants is the Scott's number which uniquely identifies this stamp from all others. In a conventional machine, one would have to store lists of all "blue" stamps, all "red" stamps, all "two-penny" stamps, etc., etc., and, as the user entered descriptive information, cross-correlate these lists searching for intersections. In the CAM all these subsidiary lists disappear and a straight content search is carried out for all the stamps meeting the given descriptions.

But this is not all. Once a search of the database has been carried out in a CAM, we can count responders and tell the user immediately how many stamps meet his criteria of search so he can decide when to terminate his search. Further still, because the data is stored in a CAM, we can examine the unspecified criteria to determine which ones will most rapidly narrow down the list of responders and suggest to the user that he enter information about these criteria. Thus, the CAM enables us to help the user help the CAM to help the user in a most advantageous manner. But this is still not all. By storing key descriptive words in the CAM, we can let the user enter information in very nearly normal English, pick out the key words he uses, and do an incremental search while he is still inputting his description.

Consider the problem of garbage collection in LISP (1). One of the problems that inhibits the use of LISP for real-time applications is that of the unpredictable and arbitrarily long delays that occur when main memory becomes full. One sees similar "naps" occurring on personal computers when string space is all used and the operating system has to clean out old strings that are no longer referenced.

A typical LISP cell in a CAM might contain three fields (at least) called "garbage" (one bit long), "left" (CAR), and "right" (CDR). When a free cell is needed, we search for a cell whose garbage bit is "1". Before we can use this cell, we must discover if it is the only cell in memory now pointing to the cell named in the "left" field (call it TARGET). In a CAM we can ask if there exist any cells which point to TARGET in either their CAR or CDR. If there are none, then TARGET is also a garbage cell and we set its garbage bit. A similar search must be done for the cell pointed at by the "right" field. Once these four searches have been done and garbage bits set if required, we can take and use the original cell we found. This takes only four exact

match searches and for fifteen bit fields could easily be accomplished in under ten microseconds. Thus LISP becomes available for real-time use where arbitrary delays might be fatal.

Weems (9) has studied Conway's game of Life, in which the survival or death of a cell depends on the state of the eight immediate neighbors. What Weems does is to send a "counter" on a path which encircles its host cell (north, west, south, south, east, east, north, north, west, south), and at each neighboring cell adds one to the counter if the cell it is currently occupying is alive and adds zero to the counter if the current host is dead. When the counter eventually returns to its home cell it contains the count of how many live neighbors the home cell has. This is the number which determines the state of the home cell on the next generation. Every cell does its neighborhood search at the same time.

But the count of live neighbors is only a special case of a convolution in a 3 x 3 window, and is easy to generalize if the "add" is replaced by a more complicated operation. Local, window-type convolutions are very common in programs which process visual images and in many other areas of application. Indeed much of the interest in Fast Fourier Transforms is because once the transform has been carried out a simple multiplication followed by an inverse transform can be used to perform a convolution. In a CAM we can perform a simple convolution in about one hundred microseconds directly on the image in question without having to perform the transform and its inverse.

In the analysis of images taken by a moving camera one tries to identify "interesting points". Successive positions of an "interesting" point establish a "flow-vector" for that point and a set of such flow-vectors, when projected, can be used to identify the FOE (focus of expansion) which is the point toward which the camera is moving. Due to noise in the images and inherent digitization noise, the set of flow-vectors will not all intersect at a single point. Taken pairwise, the flow-vectors will intersect in a set of points and the center of mass of these intersection points is a good estimate of the FOE. Such a center of mass can be found in a CAM in under a millisecond (7), or roughly 1400 times faster than a conventional machine. What this means is that this simple-minded approach to motion analysis is useful in a CAM and useless in a conventional machine.

Phases 3 and 4: Analysis and Design

Our analysis of the REM and design for a large scale CAM are reported in 5. Two major reasons may be deduced for the failure and expensive cost overruns of many past innovative architectures. First of all, they tripped over technology and, second, once they were completed they sat in a corner because nobody wanted to expend the energy and money required to use them.

Our chip design is very conservative. It does not push the current VLSI technology at all. It can readily be fabricated using 2 1/2 micron technology on a chip less than 6mm square. Using optimized dynamic memory, the size of the chip could be reduced by perhaps a third.

We have designed for a memory that is 512 x 512 cells, but the chip design is such that any size memory from 8 x 8 cells up to 4096 x 4096, or even

larger, can be accommodated. All size-dependent functions reside in "edge cards" that must lie along the bottom edge of the memory for counting responders.

In regards to the second point, our department has active groups in vision analysis, robotics, motion analysis, database management, and others who are eagerly awaiting the arrival of the Titanic so that they can use it to speed up their heavily compute-bound problems.

Published Papers

1. Bonar, J. and Levitan, S., "Real-Time Lisp Using Content Addressable Memory", International Conference on Parallel Processing, Bellaire, Michigan, August 1981, pp.112-119

2. Levitan, S. and Foster, C., "Finding an Extremum in a Network", Nineth Annual Symposium on Computer Architecture, Austin, Texas, April 1982, pp.321-325.

3. Levitan, S., "Algorithms for a Broadcast Protocol Muktiprocessor", Third International Conference on Distributed Computing Systems, Miami, Florida, October 1982, pp.666-671.

4. Wall, R., "Decryption of Simple Substitution Cyphers With Word Divisions Using a Content Addressable Memory", Cryptologia, 4,2, April 1980, pp.109-115.

5. Weems, C., Levitan, S., and Foster, C., "Titanic: A VLSI Based Content Addressable Parallel Array Processor", Proceedings of IEEE International Conference on Circuits and Computers, September 1982, pp.236-239.

Unpublished Papers

6. Foster, C., Levitan, S., and Wall, R., "Using A REM to Identify Postage Stamps".

7. Foster, C., "Finding A Center of Mass With A CAM".

8. Foster, C., "A Two Rail Algorithm For Finding An Extremum".

9. Weems, C., "Life Is A CAM-Array, Old Chum".

10. Hough, A., "Tune Recognition Using A Content Addressable Memory".

11. Weems, C., "An Algorithm For A Simple Image Convolution On The Titanic Content Addressable Parallel Array Processor".

12. Weems, C., "A CAM Implementation Of The Naval Algorithm For Text-To-Speech Synthesis".

APPENDIX A

Description of The Titanic

Origins

Starting in the Summer of 1979, after acquiring a small Content Addressable Memory (CAM), we conducted an extensive exploration of applications of content addressability and parallelism. During the ensuing three years some thirty applications have been developed with over a dozen being programmed to completion. All have been analyzed with an eye toward the design of more useful hardware. Application areas have included data base retrieval, LISP garbage collection, text-to-speech synthesis, and image convolution. Some of the results of this work are presented here as a rationale for some of our architectural design decisions.

More Cells With Less Memory

One of our major findings is somewhat counterintuitive. Normally, CAM designers give a large amount of memory to each cell of the CAM. This is so that each cell may hold a large record of logically associated data. Such a design attempts to maximize the benefits of the CAM's associativity. We have found, however, that a majority of interesting CAM applications require only one to sixteen bytes of memory in each cell, and that these applications benefit much more from the added parallelism of having more cells. Further, we have found that those applications which require more memory in each cell will work adequately if an efficient means of moving data between cells is provided. Thus, we conclude that the resources required to construct large cell memories would be far better spent in

constructing more cells with less memory.

Need Fast Some/None and Find First

A common means of controlling loop execution in CAM algorithms is to continue processing until none or only one of the CAM's tag bits are turned on. It is thus essential that we have a fast means of determining this. The simplest way of doing this is to test whether any tags are on; if so, then we select one and turn it off, then repeat the some/none test. The find first operation is also used frequently when a search finds several data elements with the same key value. It then provides a way to select one of these for processing. These cases combine to emphasize the need for fast some/none and find first operations.

Slower Response Count is Acceptable

Many CAM designs devote much complex and expensive hardware to increasing the speed of the operation which counts the tag bits that are turned on. We have found, however, that the count of responding tags is used primarily as a way of gathering statistics for use at much higher levels of processing control to direct the strategic application of the CAM. It is thus rather infrequently applied as compared to operations such as comparisons and some/none tests. We thus feel that slower, simpler, less expensive response count hardware is quite acceptable. Further, we have designed a very simple response count system which runs only about an order of magnitude slower

than much more complex designs.

Convenient Additions

CAMs typically have only one tag bit per cell. We have found, however, that most algorithms need two to four tags. Usually this is simulated by storing tag bits in the memory, however this becomes a major inconvenience when the cells have small memories. It is thus convenient to have multiple tag bits in each cell. Although any CAM with bit select and multi-write can perform bit-serial addition (and, incidentally, is thus called a Content Addressable Parallel Processor -- CAPP -- see Foster [1]), it is far more convenient and several times faster to perform additions if each cell contains a full adder. Finally, we have also found it quite convenient if each cell is provided with a way of logically combining stored tag bits. When a CAPP is provided with these capabilities at the individual cell level, the result is a Single Instruction Multiple Data (SIMD) parallel processor of considerable power.

An Image Processing CAPP

By the Winter of 1981 we had begun to examine application of a CAPP to image processing. We soon found that we were dealing with two kinds of problem solutions. One kind worked independently of where pixel values were placed in the CAPP. An example of this truly associative type of solution is histogram directed feature extraction. The other kind required that pixel data be combined and it

was thus necessary to use inter-cell communication links to accomplish this. Although we had already considered a linear cell interconnect (as a way to simulate cells with larger memories), we were now faced with problems that required a rectangular interconnect (hexagonal and triangular interconnects were not considered because digitized images do not map well onto them). An example of this is contrast enhancing image convolution. We also discovered that the edges of an image require special processing. Our solution to this problem was to provide a four-way (N, S, E, W,) cell inerconnect network with three different edge treatments. The simplest edge treatment is dead-edging, that is making the edges of the grid act like a frame of inactive cells. Another treatment is circular-wrap. In this case each edge cell is logically connected back around to its counterpart on the opposite edge. The most complex treatment is spiral-wrap in which each edge cell is logically connected to a cell on the opposite edge that is offset by one row or column. This last treatment provides a way to turn an essentially rectangular CAPP into a linear structure and thus make it more general.

Some practical aspects of designing an image processing CAPP include the need to be able to load the memory with an image in one video frametime (1/30 second). This may seem like a long time, but remember that a 512x512 image contains 262,144 pixels. For sixteen-bit pixels this means a data transfer rate of about sixteen million bytes per second. We

have also considered types of secondary storage that will be needed to keep up with such transfer rates. Hardware testing and debugging have also been major concerns simply because of the large number of components involved.

Titanic

In the Summer of 1981 we started work on the design of a VLSI-based CAPP for image processing. Our intent was to produce a conservative design which would be simple enough for us to construct with reasonable confidence of success but which would also provide a significant advance in processing power. From the beginning we imposed a number of constraints on the design. For example, the CAPP would have to consist of no more than one hundred circuit boards and each board should have a maximum of one hundred off-board connections. As another example, the VLSI chips we designed would be restricted to no more that 40,000 devices, have a pin-out of no more than forty pins, and dissipate less that two watts.

We also set a number of goals which we hoped to achieve. It was decided that the CAPP should contain 262,144 cells arranged as a rectangular 512x512 array to facilitate image processing. Each cell would contain at least thirty-two bits of memory, multiple tags, and some bit serial processing power. One hundred nanoseconds was set as a goal for the minor clock cycle time. We also planned to meet as many of the design recommendations established by our CAM research as we could. Finally, it was decided that

the CAPP would be built to be driven by another machine, such as a Digital Equipment Corporation VAX. Once the goals and constraints were set, work on the design got under way and, for obscure reasons, the project was given the name "Titanic".

Titanic and Its Environment

The Titanic is divided into two main parts: the central control and the parallel processor. The central control is a simple, fast, fetch-ahead pipelined processor which will be built from MSI devices. It issues instructions to the parallel processor, controls loading and unloading of data in the parallel processor, serves as an interface to the VAX or other host computer and to other data sources and secondary storage devices. The central controller contains a ROM with a set of micro-coded subroutines for performing commonly needed higher level CAPP operations, and a writeable control store which allows users to add their own special microcoded instructions. Also contained in the central controller is a small program memory into which subroutines or entire programs may be loaded. The writeable control store and program memory are loaded directly by the VAX. Once these memories are loaded, the VAX can issue commands to the central controller which tell it to execute routines stored in the program memory, to single step through a stored routine, or to execute a single instruction passed as a literal by the VAX. Figure 1 shows Titanic and its environment.

The Parallel Processor

The Titanic parallel processor consists of an 8x8 array of processing circuit boards and a set of special purpose boards which control how the edges of the CAPP are treated, buffer broadcast signals, and perform other functions such as collecting the some/none signals to a single line. The parallel processor receives data and instructions broadcast to it by the central controller. Each parallel processor instruction operates in exactly one minor cycle time. Some operations do require multiple clock cycles, but these are taken care of by having the central control rebroadcast the instruction as many times as necessary. Figure 2 shows the structure of the parallel processor.

Each processor board consists of an 8x8 array of special CAPP integrated circuits plus some random buffer logic. A list of the sixty-three I/O lines on each board is given in Table 1. Our current design calls for all sixty-four processor circuit boards to be placed in four card racks (sixteen per rack) and interconnected by a broadcast backplane and ribbon cables.

The Titanic IC

The heart of the Titanic design is a special purpose nMOS VLSI CAPP integrated circuit. Each of these chips contains sixty-four CAPP cells, an instruction decoder, and other miscellaneous logic. The design of this IC is actually much further along than the rest of the project (this being mainly due to test chip fabrication time

constraints). To compensate for this somewhat bottom-up development we have designed the chip with as much generality as possible, knowing that such generality need not be fully used later on.

The Communications Interconnect

One of our biggest problems in designing Titanic was how to handle the rectangular interconnection of the cells. The number of wires required for such a network, even for bit serial communications, is staggering. This became most evident when we tried to design the IC communications interface. For sixty-four cells, the arrangement which gives the minimum number of external connections is an 8x8 grid. With a four-way N,S,E,W interconnect there are then only thirty-two neighboring cells to connect to. (We considered an eight-way N,S,E,W,NW,NE,SW,SE interconnect, but were forced to abandon it due to the wiring complexity.) By the time control, power, and clock signals were added to the thirty-two neighbor lines, we found that a sixty-four pin package would be required to hold the IC. Further examination also revealed that a full interconnect would require that each processor board have 256 ribbon cable communication lines -- in other words, a two foot wide swath of ribbon cable running between each pair of boards! Because this violated two of our main design constraints, we had to simplify the interconnect.

By 3:1 multiplexing the communications net as it crossed chip boundaries, we were able to reduce the IC pin count to twenty-two pins and the total board wire count to sixty-three (of which only thirty-two need to be run in ribbon cable). By going from sixty-four pin to twenty-two pin packages, the board size was also reduced significantly. Unfortunately, all of these benefits were paid for in a loss of speed. The new interconnect takes 0.8 microseconds to transfer one bit between cells (25.6 microseconds for thirty-two bits). We should also note here that the Titanic instruction set makes this multiplexing transparent to the user.

Some/None Logic

On-chip the Some/None signal is determined by feeding the output of the main tag bit into a sixty-four-way NOR with an inverter between its output and the Some/None pad driver. Once the signal goes off-chip, it passes through a four-level OR tree before reaching the central controller.

Count Responders

The count responders operation requires only three changes to be made to the CAPP circuitry to be feasible. Firstly, it must be possible to connect all of the response bits into a circular shift register. This is easily accomplished because the neighbor communication network already provides most of the necessary links. Secondly, the South multiplex register must be modified to include a

counter and a full adder. Finally, the cards that control the top-bottom edge treatment must be modified to include column summing registers and a final sum register.

The algorithm used to count responders is given in Figure 3. This method is reasonably fast (about twenty-six microseconds), inexpensive, and most importantly it can be used with any size of array without having to modify the IC -- only the bottom row circuit board needs to be changed.

Device Floorplan

Figure 4 shows the Titanic IC's floorplan. The unit cells are arranged in two columns of thirty-two. This arrangement was chosen because we found that the best compaction would be obtained if we could share control and memory select lines among as many cells as possible. Each cell is thus very long and narrow. A column of thirty-two cells is almost covered by a river of metal control and select lines which run vertically over it. These lines are simply duplicated and mirrored for the two columns. Control is generated by a PLA with no feedback loops while select signals are generated by a simple decoder. Communications multiplex and responder count hardware is provided by a small block of random logic. The overall size estimate of the active chip area (excluding pads and drivers) is 2400x2400 lambda. Thus if lambda is three microns, the central portion of the die would be roughly 285 mils on a side. This is somewhat large, but not unreasonable. Power dissipation is estimated at 1.5 watts, which is low enough

to allow forced-air cooling. Table 2 lists the pin functions of the Titanic IC.

The Unit Cells

A unit cell consists of thirty-two bits of fully static memory, four one-bit static tag "registers" called A, B, X, and Y, and a static carry bit "register" called Z. Each cell also contains an ALU which continuously generates X nand Y, X nor Y, and X + Y + Z. Finally, each cell contains logic for selecting some source of data (a register, memory, an ALU function, broadcast data, or a neighbor cell), possibly inverting the selected signal and storing it in a destination (memory or register). Neighbor communiation lines run vertically in a channel that divides the cell. The Z register is special in that it is not available for selection as a data source. It can be copied directly to the X register and can be loaded from the output of the selector. It also is loaded with the carry from X + Y + Z whenever that function is selected.

The X register is special in that its output is connected to the some/none logic and the neighbor communication network. In some sense it is the "main" tag bit.

The A register is also special. It controls whether the cell is active. If a cell is not active, it ignores all instructions broadcast by the central controller except a special few.

The A register is intended to be used for storing a second set of tag bits which may eventually be combined with other sets through the logical operations provided by the ALU.

The B register is intended as temporary storage for a second set of activity bits, essentially providing a single level of "subroutine call" or an alternative activity "screen".

Figure 5 shows the logical arrangement of a unit cell while Figure 6 shows its silicon floorplan.

Titanic IC Instruction Set

Table 3 lists the instruction set of the Titanic IC. Each instruction executes in one minor clock cycle (100 n s). This was done to avoid feedback loops in the decode PLA on the chip and to avoid special instruction states in the central controller. This means that the central controller must be programmed to re-issue some instructions several times. For example, transferring data to neighbor cells across chip boundaries requires eight individual transfers because of the 8:1 multiplex. The central control must therefore issue the shift instruction eight times in a row. This, of course, is encoded as a single operation in the controller's microcode ROM.

There are eight basic instructions recognized by the chip. Of these, six are memory transfer operations and use a five-bit address value to select the bit to be read or written. The other two instructions treat the address as a

sub-operation specifier. For the most part these are non-memory data source to register data transfer operations with one op code causing the data to be inverted before storage and the other causing a direct transfer. There are sixteen special sub-ops, however, which are reserved for unusual operations such as transferring data through the multiplexers or counting responders.

Some operations (those underlined in Table 3) are also designated as "jam transfers". This means that they are performed regardless of whether the A register contains a logic one. These provide a means of storing and retrieving different activity patterns and of applying global operations which ignore activity without the usual overhead of having to save the current activity pattern, and retrieve it later.

Current Status

As of this writing we have designed a sixteen-cell (4x4) test chip, and are negotiating for fabrication. Using a simple set of three micron design rules, we have succeeded in fitting the circuitry onto a 180x180 mil body area with considerable room to spare. The actual cell area occupies only 130x106 mils. Estimated power dissipation is only 300 milliwatts.

We have already written a number of programs for the Titanic and estimated their operation times by hand. For example, one special purpose convolution of interest in computer vision processing (a simple 3x3 mask) required only

112 microseconds for the entire 512x512 image. More complex convolutions take longer, of course, but most of interest can be performed in less than five milliseconds. We have also examined motion analysis and found the results to be quite encouraging.

Further Research

Based on the results of our test chip experience, we intend to proceed to full sixty-four cell ICs and, eventually, construction of the entire machine. Architectural changes which we intend to pursue are increasing the memory size to sixty-four bits per cell and perhaps going to an 8:2 communications multiplex (with a twenty-eight pin package) for a doubling in the data transfer rate.

We also plan to program a statistics gathering Titanic simulator which will allow us to experiment with software development and optimization.

Our work thus far has indicated that a Content Addressable Parallel Array Processor is extremely well suited for image processing, vision, and motion analysis. We intend to pursue further applications in these areas and also in new areas such as tactile object recognition in robotics.

Conclusion

Rationale and a design have been presented for a Content Addressable Parallel Processor suitable for both general use and image processing applications. The architecture of the processor is based in practical experience and the hardware design has been constrained to make it possible to construct using existing technology and with a high confidence of success. Despite these constraints, simulations have shown that such a machine would provide a significant increase in processing power over what is presently available.

References

[1] Foster, Caxton C., Content Addressable Parallel Processors, Van Nostrand Reinhold, New York, 1976.

List of Processor Board I/O Lines

Number of Lines	Function
8	Bidirectional North Neighbor Communications
8	Bidirectional South Neighbor Communications
8	Bidirectional East Neighbor Communications
8	Bidirectional West Neighbor Communications
8	Chip Column Select
8	Chip Row Select
4	Op Code
5	Bit Address or Sub Op Code
1	Broadcast Comparand Data
1	Some/None Output
2	Clock phases
1	Power
1	Ground
63	

Table 1

List of Titanic IC Pin Assignments

Pin	Function
1	West Neighbor Communications (Bidirectional)
2	Chip Select 1
3	South Neighbor Communications (Bidirectional)
4	Op Code Bit 1
5	Op Code Bit 2
6	Op Code Bit 3
7	Op Code Bit 4
8	Comparand in
9	Some/None out
10	Clock Phase 1
11	Ground
12	East Neighbor Communications (Bidirectional)
13	Chip Select 2
14	Spare (Test)
15	Address Bit 5
16	Address Bit 4
17	Address Bit 3
18	Address Bit 2
19	Address Bit 1
20	North Neighbor Communications (Bidirectional)
21	Clock Phase 2
22	Power

Table 2

Titanic IC Instruction Set

Memory Operations

Op Code	Write/Read 0	1
0	M := A	A := M
1	M := B	B := M
2	M := X	X := M
3	M := Y	Y := M
4	M := C	A := M
5	M := B	B := M

N = North neighbor data
S = South neighbor data
E = East neighbor data
W = West neighbor data
Z = Carry bit register
M = Memory bit specified by address
A = A tag register
B = B tag register
X = X tag register
Y = Y tag register
C = broadcast Comparand

Underline indicates jam transfer

Non-Memory to Register Operations

Op code 65D - Source transferred to destination
Op code 75D - Source inverted and transferred to destination

Destination	Source 0	1	2	3	4	5	6	7
0	A:=A	A:=B	A:=X	A:=Y	A:=$\overline{X+Y}$	A:=$\overline{X \wedge Y}$	R:=N	S
1	B:=A	B:=B	B:=X	B:=Y	B:=$\overline{X+Y}$	B:=$\overline{X \wedge Y}$	R:=S	P
2	X:=A	X:=B	X:=X	X:=Y	X:=$\overline{X+Y}$	X:=$\overline{X \wedge Y}$	R:=E	E
3	Y:=A	Y:=B	Y:=X	Y:=Y	Y:=$\overline{X+Y}$	Y:=$\overline{X \wedge Y}$	R:=W	C
4	A:=N	A:=S	A:=E	A:=W	A:=C	A:=$\overline{X \vee Y}$	Z:=C	I
5	B:=N	B:=S	B:=E	B:=W	B:=C	B:=$\overline{X \vee Y}$	A:=B	A
6	X:=N	X:=S	X:=E	X:=W	X:=C	X:=$\overline{X \vee Y}$	A:=X	L
7	Y:=N	Y:=S	Y:=E	Y:=W	Y:=C	Y:=$\overline{X \vee Y}$	A:=Y	

Special Operations

Op code	"Destination" 0	1	2	3	4	5	6	7
67	NM:=C	SM:=C	EM:=C	WM:=C	Z:=X	CRCR	SCR	PANS
77	MBN	MBS	MBE	MBW	X:=Z	NOP	NOP	NPO

NM = North Multiplexer buffer
SM = South Multiplexer buffer
EM = East Multiplexer buffer
WM = West Multiplexer buffer
MBN = Move Buffer North
MBS = Move Buffer South
MBE = Move Buffer East
MBW = Move Buffer West
CRCR = Clear Response Count Register
SCR = Shift and Count Responder
PANS = Piplined Add North to South
NOP = No Operation

Table 3

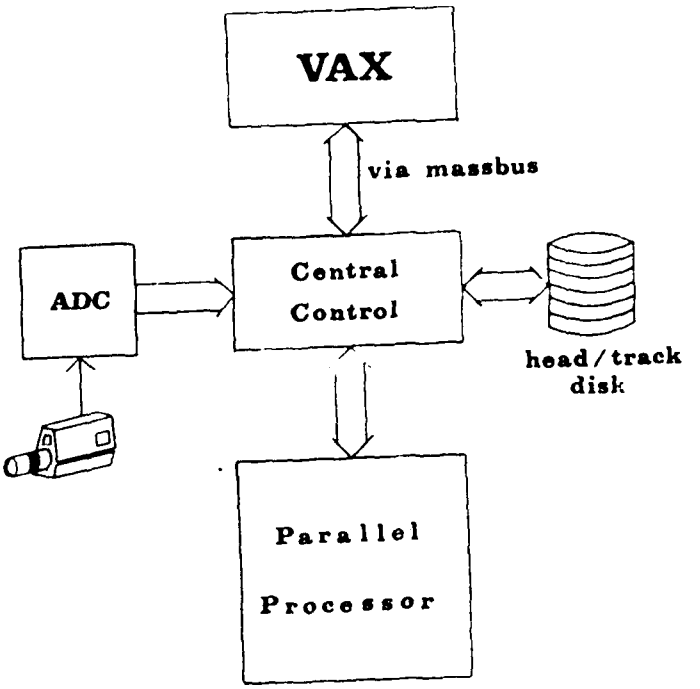

VAX
via massbus
ADC
Central
Control
head/track
disk
Parallel
Processor

Fig. 1

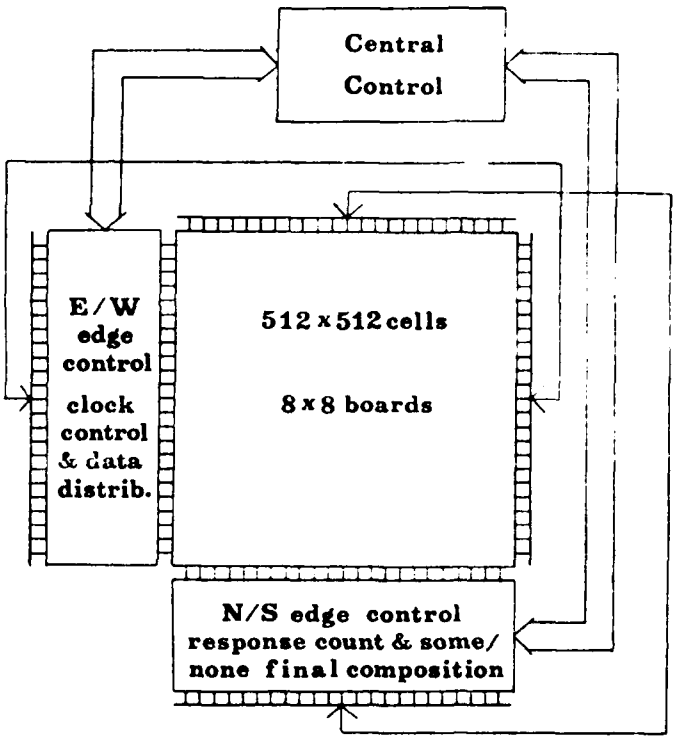

Central
Control
E/W
edge
control
clock
control
& data
distrib.
512 x 512 cells
8 x 8 boards
N/S edge control
response count & some/
none final composition

Fig. 2

```
Set all activity bits                                          0.1μS
Clear Response Count Register (CRCR)                           0.1μS
For I:=1 to 64 do
    Shift and Count Responder (SCR)                            6.4μS
Turn off all chip row select lines                             0.1μS
Turn on all chip column select lines                           0.1μS
For I:=1 to 64 do
    begin
        Turn on row select line I                             12.8μS
        Pipeline Add North to South (PANS)
    end
For I:= to 6 do (*Empty the pipeline*)
    Pipeline Add North to South (PANS)                         0.6μS
For I:=1 to 64 do
    Pipeline Add West to East on Bottom Row Board              6.4μS
                                                              ------

Response count is now available on Bottom Row
    Board                                                     26.6μS
```

Pipeline Add North to South (PANS) takes the low order bit from the response count register, adds it to a data bit input on the North line and outputs the result on the South line. The carry from the addition is stored in a temporary storage cell and used in the next PANS. The input and output operations are buffered and appropriately clocked to allow true pipelined operation. Row and column select lines are turned on and off by setting and clearing bits in registers on the edge control cards. Once a row is turned on, it remains on until it is explicitly turned off and vice versa.

Fig. 3

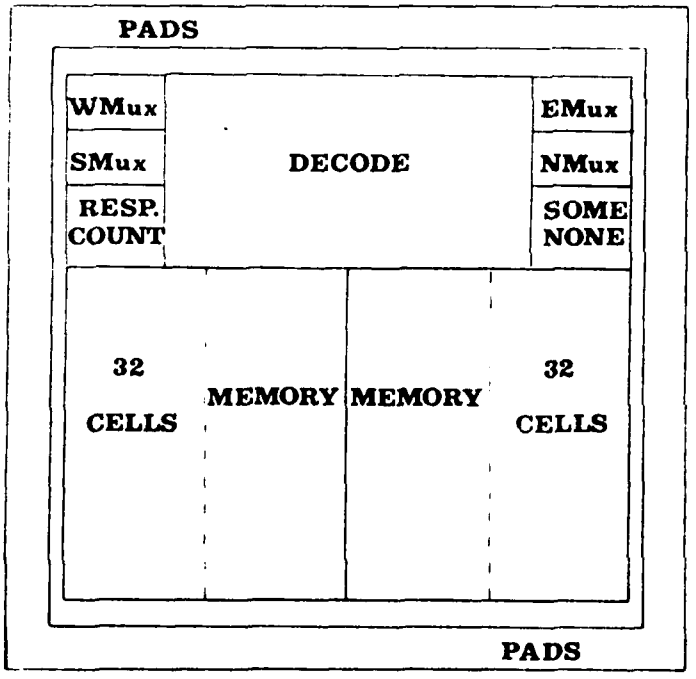

PADS
WMux
SMux
RESP. COUNT
DECODE
EMux
NMux
SOME NONE
32 CELLS
MEMORY
MEMORY
32 CELLS
PADS

Fig. 4

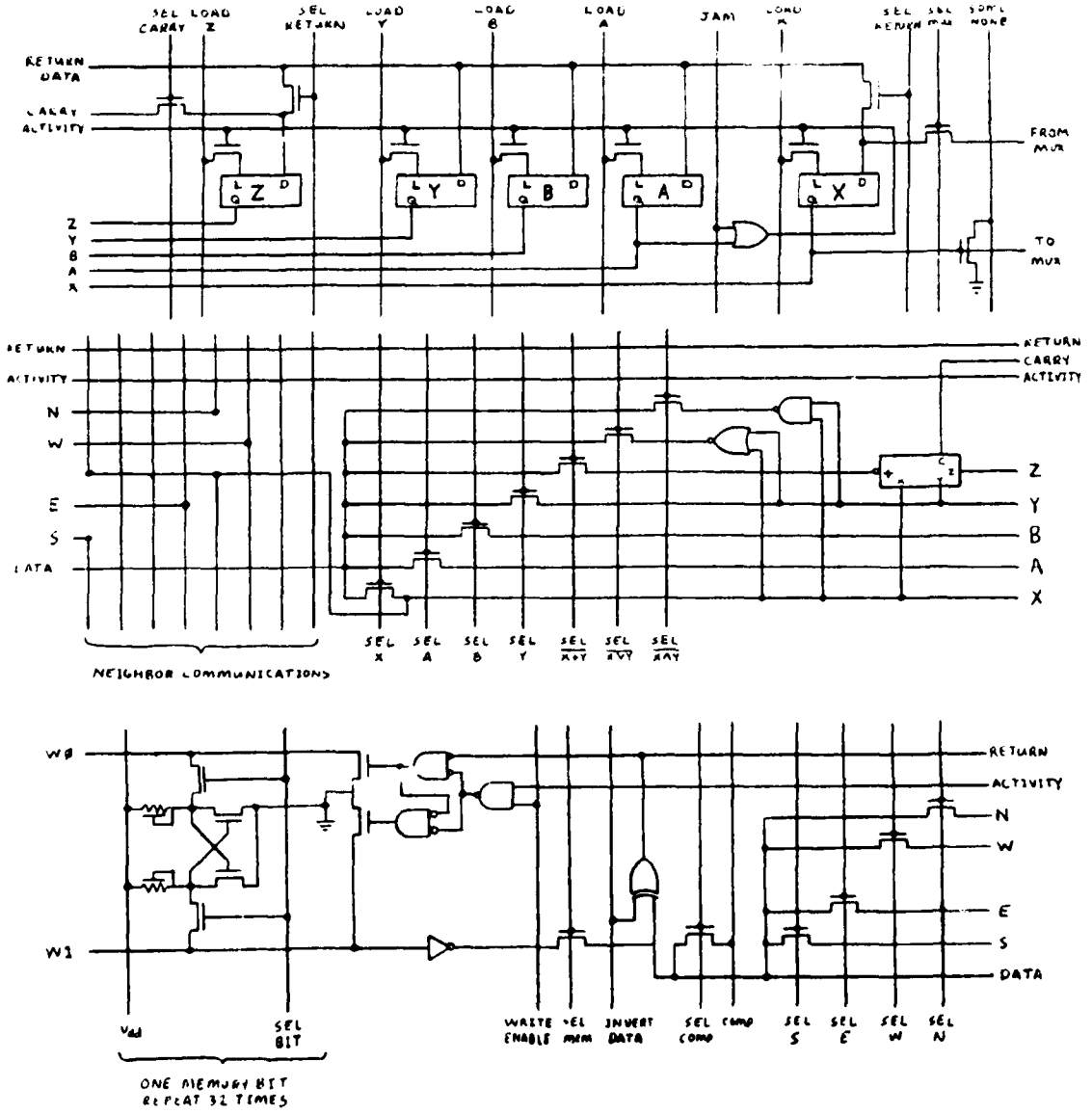

SEL CARRY
LOAD Z
SEL RETURN
LOAD Y
LOAD B
LOAD A
JAM
LOAD X
SEL RETURN
SEL MUX
SEL NONE
RETURN DATA
CARRY ACTIVITY
FROM MUX
TO MUX
Z
Y
B
A
X
RETURN
ACTIVITY
N
W
E
S
DATA
RETURN
CARRY
ACTIVITY
SEL X
SEL A
SEL B
SEL Y
NEIGHBOR COMMUNICATIONS
W0
W1
V_{dd}
SEL BIT
ONE MEMORY BIT
REPEAT 32 TIMES
WRITE ENABLE
SEL MEM
INVERT DATA
SEL COMD
SEL S
SEL E
SEL W
SEL N

Fig. 5

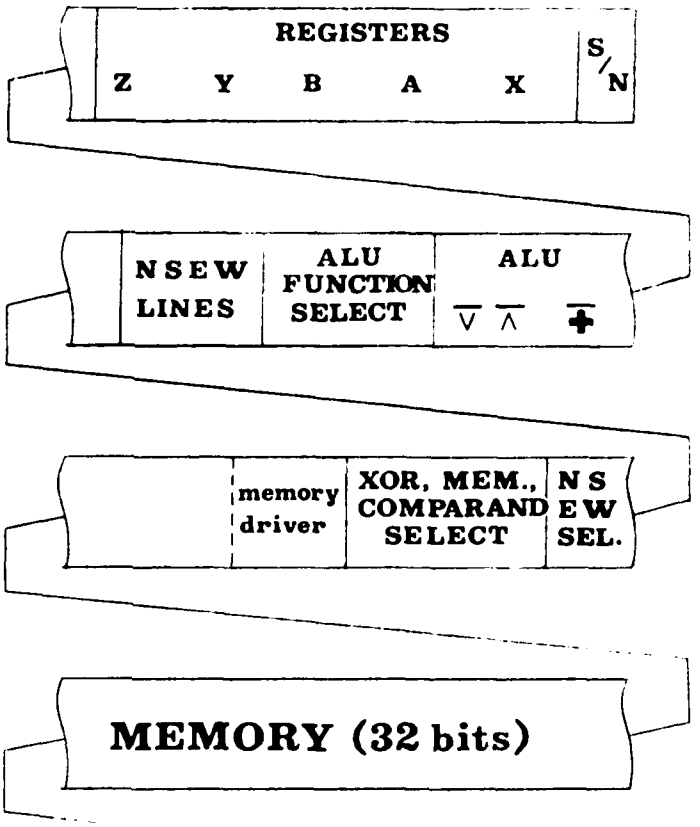
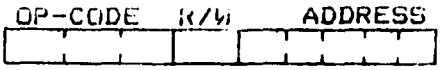

REGISTERS
Z
Y
B
A
X
S/N
NSEW LINES
ALU FUNCTION SELECT
ALU
memory driver
XOR, MEM., COMPARAND SELECT
N S E W SEL.
MEMORY (32 bits)

Fig. 6

Titanic IC Instruction Set

Memory Operations



Op	R/W	FUNCTION
0	0	M:=C
0	1	A:=M
1	0	M:=B
1	1	B:=M
2	0	M:=X
2	1	X:=M
3	0	M:=Y
3	1	Y:=M
4	0	M:=A!
4	1	A:=M!
5	0	M:=B!
5	1	B:=M:

! - Transfer ignores activity
A - Activity register
B - Secondary Activity register
C - Comparand
M - Memory
X - Main tag register
Y - Secondary tag register
Z - Carry register
N - Data from North
E - Data from East
W - Data from West
S - Data from South

Register Operations

OP CODE 6 - NORMAL
OP CODE 7 - INVERT SOURCE

Source \ Destination	0	1	2	3
0	X:=A!	A:=A!	B:=A!	Y:=A!
1	X:=B	A:=B	B:=B	Y:=B
2	X:=X	A:=X	B:=X	Y:=X
3	X:=Y	A:=Y	B:=Y	Y:=Y
4	X:=$\overline{X+Y}$	A:=$\overline{X+Y}$	B:=$\overline{X+Y}$	Y:=$\overline{X+Y}$
5	X:=$\overline{X \wedge Y}$	A:=$\overline{X \wedge Y}$	B:=$\overline{X \wedge Y}$	Y:=$\overline{X \wedge Y}$
6	X:=$\overline{X \vee Y}$	A:=$\overline{X \vee Y}$	B:=$\overline{X \vee Y}$	Y:=$\overline{X \vee Y}$
7	X:=C	A:=C	B:=C	Y:=C
8	X:=N	A:=N	B:=N	Y:=N
9	X:=E	A:=E	B:=E	Y:=E
10	X:=W	A:=W	B:=W	Y:=W
11	X:=S	A:=S	B:=S	Y:=S
12	X:=N~!	A:=C!	X:=CN~!	Z:=C
13	X:=E~!	A:=B!	X:=CE~!	(6)Z:=X (7)X:=Z
14	X:=W~!	A:=X!	X:=CW~!	(6)SCRR! (7)CRCR!
15	X:=S~!	A:=Y!	X:=CS~!	(6)SCRC! (7)PANS!

~ - Zig zag shift with data transfer in and out of chip
SCRR - Shift and count responders by rows
CRCR - Clear response count register
PANS - Pipelined add North to South
SCRC - Shift and count responders by columns


MED
83