LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-455

# A FAST MULTIPORT MEMORY BASED ON SINGLE-PORT MEMORY CELLS

R. Rivest
L. Glasser

July 1991

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

# A fast multiport memory based on single-port memory cells

Lance A. Glasser*
DARPA/ISTO
Arlington, Virginia 22209

Ronald L. Rivest†
MIT Laboratory for Computer Science
Cambridge, MA 02139

— July 10, 1991—

**Abstract**

We present a new design for dual-port memories that uses single-port memory cells but guarantees fast deterministic read/write access. The basic unit of storage is the word, rather than the bit, and addressing conflicts result in bit errors that are removed by correction circuitry. The addressing scheme uses Galois field arithmetic to guarantee that the maximum number of bit errors in any word accessed is one. These errors can be corrected every time with a simple correction scheme. The scheme can be generalized to an arbitrary number of ports.

## 1 Introduction

The purpose of a multiport memory is to provide several *simultaneous* communication paths to an array of data. Each "port" provides a separate independent access path for reading data from the array, or writing new data into the array. This array may be accessed in a "random-access" manner through each port—each such access may read or write any memory position, independent of which other positions are accessed at other times or through other ports. Thus a $p$-port multiport memory almost acts like $p$ independent data arrays, except that the contents of the arrays are always identical. Such a $p$-port memory can support $p$-way parallel access to the data array, allowing some computations to run up to $p$ times faster than if the accesses were processed sequentially. Dual-port memories are widely used in computer processors to implement an array of registers, so that an operation requiring the values of two registers can obtain both values with a single two-port read operation (see, for instance, [3], [6, Section 5.9], or [13]). Another common use is as a buffer between communicating finite-state machines. This paper addresses the question of how to efficiently implement a multiport memory.

*email address: glasser@darpa.mil

†Supported by NSF grant CCR-8914428 and DARPA contract N00014-89-J-1988. email address: rivest@theory.lcs.mit.edu

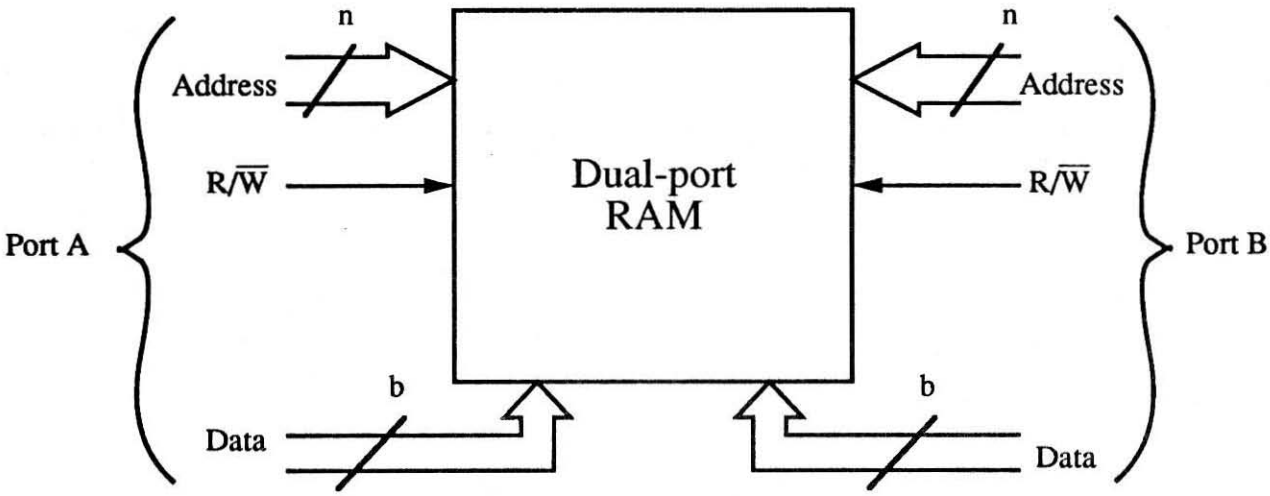


Figure 1: The block diagram of a dual-port RAM.

In the remainder of this introduction we introduce some notation that will be used throughout the paper.

We assume that the multiport memory is to contain an array of $2^n$ words, where each word is $b$ bits long, for some integer constants $n \geq 1$ and $b \geq 1$. The address of any memory word can thus be specified as a $n$-bit quantity. Many multiport memory designs implement a bit-array ($b = 1$); we gain some advantage by considering a larger word size ($b > 1$).

Figure 1 illustrates a dual-port memory (also called a dual-port RAM). The two input/output ports are labeled port A and port B. Each port consists of an input $n$-bit address bus, a $b$-bit input/output external data bus, and a read/write ($R/\overline{W}$) signal. On each memory cycle the user of a port can supply an address and indicate whether they wish to read or write the corresponding memory position. For a write operation, the user supplies a $b$-bit data word on the data bus; for a read operation the memory places the $b$-bit value of the specified word on the data bus.

We say that the memory system has a *conflict* if two ports attempt to access the same memory word at the same time. If the ports are both reading the same word, then there is really no conflict, and we expect both ports to be able to successfully read the word addressed. However, if one port is writing a word that another port is simultaneously accessing (either reading or writing), then it is not so clear what the multiport memory should do. There are a number of such subtle issues that arise in handling write operations, and we will defer discussion of them until our scheme has been presented. We present our scheme as if it were a multiport ROM scheme (read-only memory), and then return to issues arising from handling write operations in Section 3.4.

In Section 2 we review previous approaches to the design of multiport memories. Then Section 3 presents our new proposal for the design of dual-port memories.

# 2 Previous designs

In this section we review the two basic approaches that have been used in the design of multiport memories: replicated control circuits and multiplexing. The first approach is expensive in "space" (hardware) and the second is expensive in time. We also review some related approaches that have appeared.

## 2.1 Replicated control circuitry

With this approach there is a single array of memory words, but a separate decoder/selector path for each port. The cells that store the bits are multiported.

This design has the advantage that the ports can be used to read or write data without interacting (except when they address the same word). External devices connected to the various ports can thus share data conveniently with low latency and high bandwidth because the paths are independent. An example of such a design is the Texas Instrument integrated circuit SN74172, an 8-word by 2-bit dual-port register file [5]. More recent dual-port RAMs that use this approach are discussed in [7], [9], and [11].

Unfortunately this conceptually clean technique has two significant drawbacks. First, because almost all hardware except the basic memory cell must be replicated and the memory cell itself must have more wires and ports, the circuit area of a multiport RAM that takes this approach is significantly larger than that of a single-port RAM. This increases the cost and limits the maximum size of the device that can be built as a single integrated circuit. Second, because the device is larger, its internal wires must be longer. These wires thus have more capacitance, resulting in some speed degradation. This design thus yields multiport RAMs that tend to be slower than single-port RAMs of the same memory capacity.

## 2.2 Time-domain multiplexing

To avoid the area-related costs of the replication approach, a second general approach involves various forms of time-domain multiplexing, as described, for example, by Barber et al. [1]. These approaches fall into two general subclasses: deterministic and stochastic. In deterministic approaches, the speed penalty associated with time-domain multiplexing is incurred on every access, while with stochastic approaches, there is a degree of randomness involved, allowing this penalty to be avoided most of the time. We examine these approaches in order.

In deterministic multiport RAMs with time-domain multiplexing, one can use a RAM cell with fewer ports than the memory system. For instance, a dual-port RAM might use single-port cells. This saves significant quantities of area and wire. The problem is that the path to the data is now narrower and has less bandwidth. Thus the memory system must, overall, be slower. For instance, the straightforward way to build a dual-port RAM out of single-port cells is to timeshare the accessing path so that the ports alternate accesses. First one port is allowed to use the decoder and memory cells, and then the second port is allowed exclusive access to these resources, and then cycle repeats. The speed of the dual-port RAM of this type is thus seen to be about half that of the single-port RAM.

In the stochastic approach to time-domain multiplexing of a multiport RAM, one again builds a system where the data path is not wide enough to support the most general form of multiport access to save area and wire. The memory system is divided into subsystems which we will call bins. In the stochastic approach, part of the decoder path is duplicated. Take the example of a dual-port RAM. In this case, if Port A requests access to data in one bin and Port B requests access data in a different bin, then both service these requests simultaneously because separate paths exist between the bins and the ports. On the other hand, if both ports attempt to access two different memory locations in the same bin, then a conflict occurs. Since only the part of the path from the ports to the bins is duplicated, the ports will inevitably attempt to use the same bin for different purposes. Since both ports cannot obtain perfect data in the case of a conflict, it is standard practice to cause at least one of the ports to wait. Thus, when a conflict occurs, at least one of the ports sees additional delay. This delay not only increases the average delay, but its non-deterministic nature increases the complexity of the system that uses the multiport RAM. Also, the arbitration circuitry must designed with extreme care.

### 2.3 Other related work

Tanaka [12] describes a clever method of organizing a disk array for a page-memory and disk-cache system, wherein each disk stores a different word of each page, and the different ports access the words of each page in different, but compatible, orderings. His method does not apply here, since we access each word as an atomic operation; Tanaka has an advantage since the words of a page can be read in an arbitrary order. On the other hand, the techniques we apply here could conceivably be applied to work in the disk array application discussed by Tanaka.

## 3 The new design

The multiport memory design presented in this paper avoids the area penalty associated with the pure replicated control circuits, and also avoids delays associated with multiplexed approaches. It simplifies the use of the multiport RAM and improves its speed.

In this paper we illustrate how to build a multiport memory with a narrow internal data path, yet with nearly the same performance. The important characteristic of a narrower data path is that it cannot support, in the general case, completely independent access from the ports to the memory cells. Occasionally, several ports will try to simultaneously use a part of the data path that is too narrow to support all of the users. In this case there is an inevitable conflict and every port's request cannot be simultaneously serviced. The essence of this paper is a novel way to handle conflicts.

This invention is based on three observations.

1. One very seldom stores information in memory systems as single-bit objects but rather one stores larger abstract objects such as words, data packets, data structures, and database records. In this paper we will refer to all of these larger objects as words.

2. If a word is relatively long compared to a single bit, then relatively little overhead is incurred by additionally storing the parity-check information needed to correct bit errors in the word, provided the number of bit errors is kept small.

3. It is possible to store the words in memory in such a way that conflicts result in bit errors rather than whole word errors. Furthermore, one can keep the number of bit errors caused by conflicts small. Each port can thus use an error correction device that corrects all of the bit errors caused by conflicts, resulting in behavior that appears perfect to the user.

In Section 3.1 we begin with a description of the memory organization used, and then describe the addressing and error-correction techniques employed in Sections 3.2 and 3.3. Section 3.4 discusses the issues arising from writing to memory, and then Section 3.5 describes the generalization of this design to general multiport memories (i.e., where the number $p$ of ports may be larger than 2).

## 3.1 Memory organization

We describe the memory organization in three steps:

- First, we describe the organization of simple single-port memory that is partitioned into "bins."
- Second, we describe a straightforward attempt to generalize this design to a dual-port design, and note that conflicts are not handled adequately.
- Third, we describe how a modification of the bin addressing method can reduce the problem caused by conflicts to that of handling a small number of "erasure errors."

Section 3.2 then gives a fuller description of how the addressing logic can be constructed, and Section 3.3 describes how the error-correction logic can be implemented.

Figure 2 shows our starting point: a simple single-port RAM design for a memory of $2^n$ $b$-bit words. We suppose that each $n$-bit address $x$ is divided into an $n_0$-bit initial portion $x_0$ and an $n_1$-bit final portion $x_1$, where $n = n_0 + n_1$.

The memory is organized into $b$ columns. Each column is a "bit plane," so that the $y$th column is responsible for storing the $y$th bit of each data word, for $1 \leq y \leq b$. The $y$th column is thus attached to the $y$th wire of the data bus.

Each column stores $2^n$ bits, organized into $2^{n_0}$ "bins" of $2^{n_1}$ bits each. The value $x_0$ is used to select the correct bin, and the value $x_1$ is used as an offset within the bin to select the correct bit. The figure shows $x_0$ being used to select a row of bins with decoder/selector logic. The figure does not show $x_1$ being distributed to each bin; the bins could share the associated decoder/selector logic for decoding $x_1$. Also not shown is the distribution of the read/write signal.

Figure 3 illustrates a straightforward attempt to extend the single-port architecture of Figure 2 to a dual-port design. Another port (port B) has been added, including a new data bus, a new address bus, and a new decoder selector circuit for the new address. Note

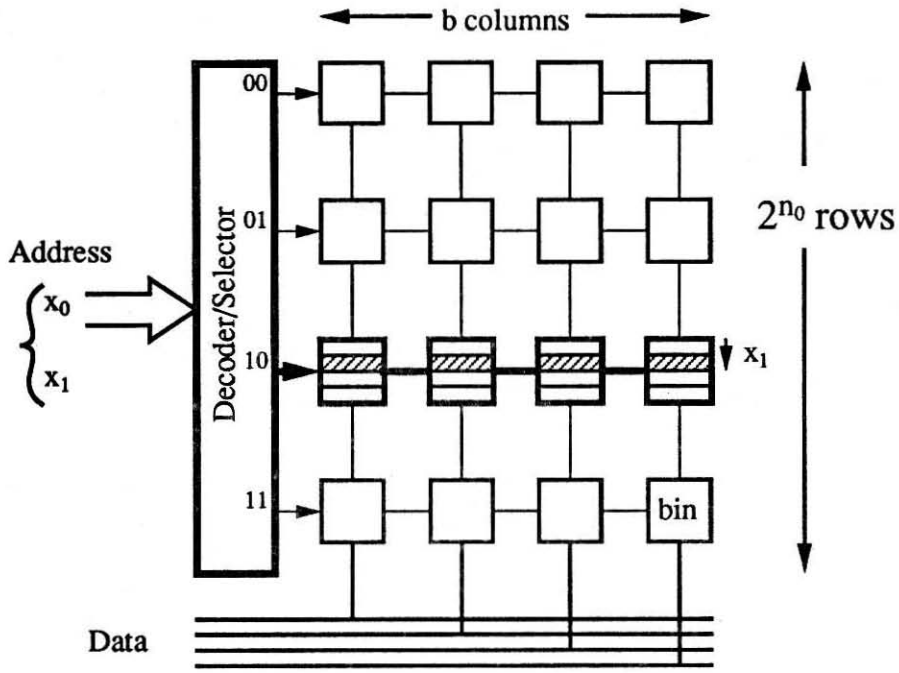


Figure 2: A simple single-port RAM architecture. This figure illustrates a 16-word 4-bit memory ($n = 4$, $b = 4$) with $n_0 = n_1 = 2$. The position of word at address $x = x_0x_1 = 1001$ is shown.

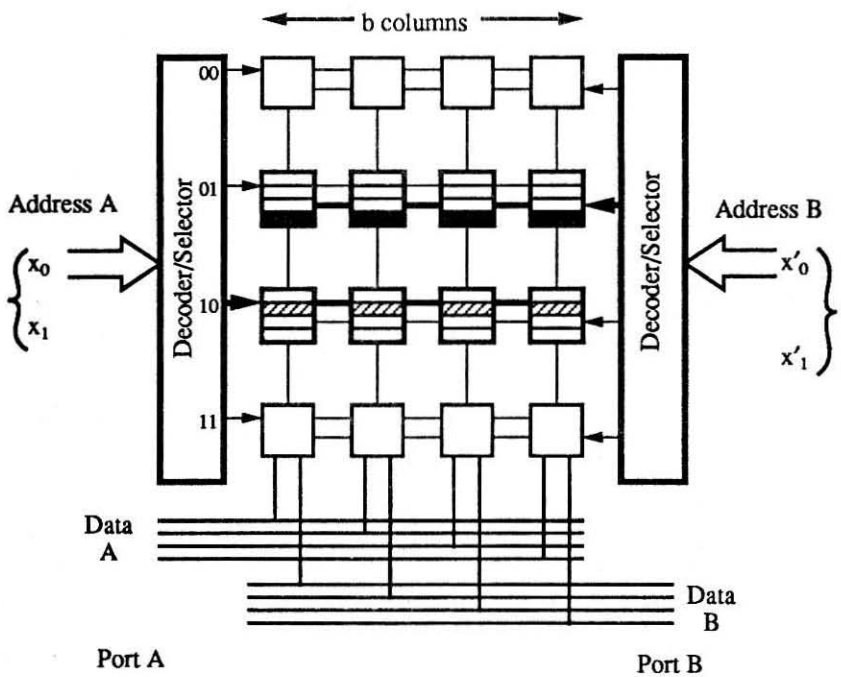


Figure 3: A dual-port RAM architecture based on the single-port RAM of Figure 2. This architecture does not handle conflicts well.

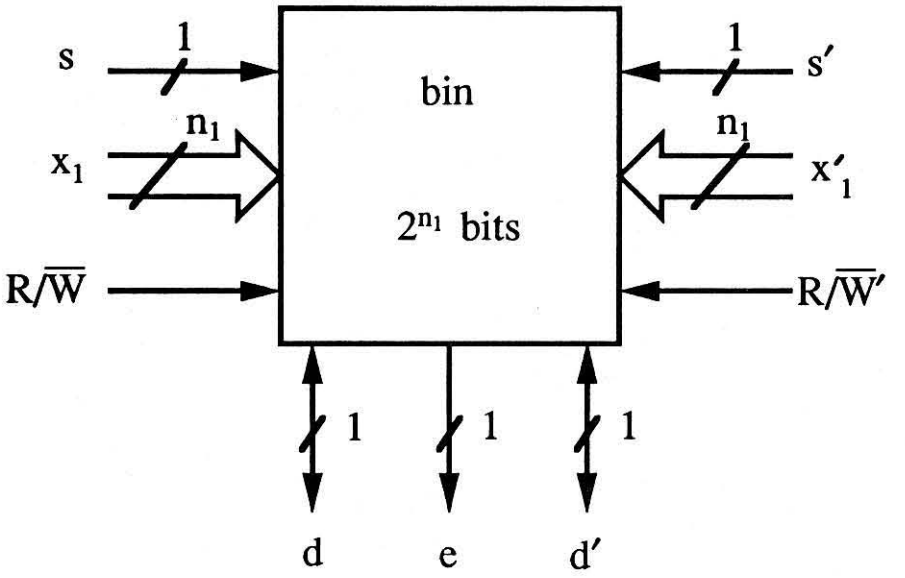


Figure 4: A dual-port bin.

the amount of circuit duplication here is far less than that called for in the full replication strategy described in Section 2.1, since the size of the decoder being replicated here is only $2^{n_0}$ rather than $2^n$. (The decoders for $x_1$ do not need to be duplicated, since a simple gating circuit can feed either $x_1$ or $x_1'$ to the decoder within each bin.) The bins are only slightly modified, not duplicated. The figure shows port A accessing word $x = x_0x_1 = 1001$ and port B accessing word $x' = x_0'x_1' = 0111$.

Figure 4 shows the configuration of the dual-port bin used in the architecture of Figure 3. This bin stores $2^{n_1}$ bits, and makes them available in a "restricted dual-port" mode as follows. Each such bin has a port A selector $s$, a port A address bus $x_1$, a port A read/write signal $R/\overline{W}$, and a port A data bus $d$. It has corresponding connections $s'$, $x_1'$, $R/\overline{W}'$ and $d'$ for port B. The bin also has an "erasure output" $e$. In typical operation, the bin is used either by port A or port B, but not both. For example, if the bin is selected by port B (signalled using $s'$) for a read operation (signalled using $R/\overline{W}'$), then the bit at address $x_1'$ within the bin is placed on data bus $d'$. Since $s$ is not active, the inputs $x_1$ and $R/\overline{W}$ are ignored, and the data bus $d$ is not affected. Such a bin is easily built out of a single-port memory array using simple multiplexors at the input/output connections.

What does such a dual-port bin do if there is a conflict (both ports attempting to use the bin)? In this case the bin only satisfies the request for port A, and ignores the port B request. It also raises the error signal $e$, however, to indicate that this bin could not satisfy the port B request.

Returning to Figure 3, we see that this simple dual-port architecture works correctly except when $x_0 = x_0'$, in which case the port B request will be totally ignored. The port A request will always be handled properly.

We thus see that the architecture of Figure 3 is quite economical (not much extra circuitry is required over the basic single-port design of Figure 2) but it is not a perfect dual-port design, since conflicts can cause a request on port B to be ignored.

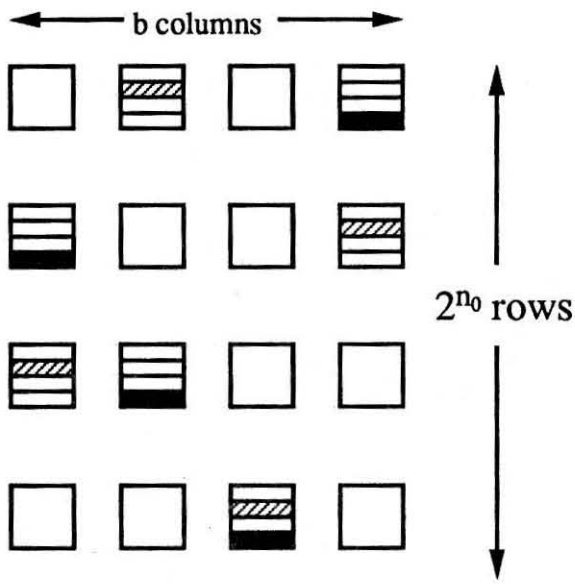


Figure 5: Illustration of the addressing principle.

Can we modify this design to handle conflicts properly? The answer is *yes*. We now examine how this may be achieved. There are two basic ideas:

1. We can modify the addressing scheme so that *any two distinct memory positions overlap in at most one bin.* Therefore, when a conflict occurs at most one bit position in the port B request is ignored. All other bit positions are read/written correctly. (Port A is always correct, as before.)

2. We can add some error-correction circuitry to the port B data bus to ensure that the damaged port B bit is correctly restored.

With these modifications we have a dual-port memory design that can handle any two simultaneous reads correctly, or a read and write operation concurrently. Two simultaneous write operations require additional work, however, as we shall see. In the next two sections we consider how the two ideas listed above are implemented.

## 3.2 Addressing method

It is an unusual characteristic of our memory design that the bits that make up a word are at different bins in each column, and any two distinct words utilize at most one bin in common. We call this the "unit-overlap principle." Figure 5 illustrates the unit-overlap principle. The cross-hatched word, selected by port A, is in bins 3, 1, 4, and 2 in columns 1, 2, 3, 4. The gray word, selected by port B, is in bins 2, 3, 4, and 1 in columns 1, 2, 3, and 4. Only one bin is in conflict (bin 4 in column 3) so that the port B word can be read with only one erasure error, in column 3.

Because of the unit-overlap principle, we can simultaneously read out any two words on the two ports, and be guaranteed that at most one bit of the word read out on port B is

incorrect. As before, all of the bits read out on port A are correct. Moreover, we know *which* bit of the port B word is potentially incorrect, since we know which bin had a conflict—this information is available from the $e$ bits in each column. (We assume that the $e$ bits of each column are combined to indicate which columns may have errors in the port B outputs.) Using simple error-correction techniques, as described in Section 3.3, we can correct this error and thus achieve simultaneous correct read-out of any two words in the memory.

In the rest of this section we examine how to arrange the words in memory so as to satisfy the unit-overlap principle.

Let $B(x_0, x_1, y)$ denote the index of the bin used in column $y$ to store the $y$th bit of the word with address $x = x_0x_1$. We need a formula for $B$ so that if address $x = x_0x_1$ is different from address $x' = x'_0x'_1$, then at most one value of $y$ satisfies the equation

$$B(x_0, x_1, y) = B(x'_0, x'_1, y) \ . \tag{1}$$

This condition guarantees that the addressing scheme satisfies the unit-overlap principle.

The computation of $B(x_0, x_1, y)$ is performed using finite field (or Galois field) arithmetic. (See Berlekamp [2], for example, for details of arithmetic in Galois fields.) This form of arithmetic requires no carries and can be quite fast.

Which finite field do we use? Since we want the result to be an $n_0$-bit number, we use the Galois field $GF(2^{n_0})$. This field contains $2^{n_0}$ elements, each of which corresponds to a bit string of length $n_0$. *We assume from now on that* $n_0 \geq n_1$ *and that* $n_0 \geq \lg b$. This assumption can be removed without undue difficulty, but it requires a rather more elaborate addressing scheme. (Details omitted.) This assumption means that $x_1$ can be interpreted as an element of $GF(2^{n_0})$, simply by appending $n_0 - n_1$ zeros to the end of $x_1$, so that it becomes a bit string of length $n_0$. Similarly, the column index $y$ (which requires $\lceil \lg b \rceil$ bits to represent) can similarly be interpreted as an element of $GF(2^{n_0})$. Thus all inputs to $B$ are elements of $GF(2^{n_0})$, and we can perform the computation of $B(x_0, x_1, y)$ in this field. Let "$\oplus$" and "$\otimes$" denote respectively the operations of addition and multiplication in the field $GF(2^{n_0})$.

We can thus express our formula for $B(x_0, x_1, y)$:

$$B(x_0, x_1, y) = x_0 \oplus (x_1 \otimes y) \ . \tag{2}$$

One can view the definition of $B(x_0, x_1, y)$, viewed as a function of $y$, as a line where $x_0$ is the intercept and $x_1$ is the slope. Since no two distinct lines can intersect at no more than one point, no two distinct addresses can conflict (same bin $B$) in more than one column $y$. This is just the unit-overlap principle, which also follows directly from the properties of fields (substituting the definition of $B$ into equation (1) gives an equation that is linear in $y$, and so can have at most one solution, given that the other values are fixed). Therefore, equation (2) provides an addressing scheme that satisfies the unit-overlap principle; any two distinct words overlap in at most one bin.

The implementation of our addressing scheme requires a modest amount of additional circuitry beyond that employed by the scheme of Figure 3. For each column $y$ of the memory, circuitry is needed to compute $B(x_0, x_1, y)$ (for port A) and $B(x'_0, x'_1, y)$ (for port B). We need $2b$ finite-field adders and $2b$ finite-field multipliers. Each adder requires $n_0$ exclusive-or

gates, and each multiplier requires less than $n_0^2$ exclusive-or gates. (See Berlekamp [2, Section 2.41], and note that for each column the value of $y$ is a fixed constant, so that multiplying by $y$ is multiplying by a constant, and the exclusive-or circuitry can be designed in advance as a function of $y$. If $n_0$ is small, ROMs can be used to store the multiplication and addition tables.)

## 3.3 Error-correction

In this section we consider the problem of correcting a single bit error in the word that has been read from port B. The unit-overlap principle described in Section 3.2 guarantees that at most one bit of the port B data will be in error.

The application of error-correction here is unusual in that we always know *where* (i.e., in which bit position) the potential error lies. We can thus view the output from port B as a binary word in which at most one bit position has been "erased" (replaced with the symbol "?"). The error output $e$ will be active for at most one column; that column is the one whose output is interpreted as a "?". As an example, for $b = 9$ we may have an output word such as

$$\texttt{01?110100}\,, \tag{3}$$

where the "?" indicates a potential error in position 3.

Correcting the erasure error is very simple, if we know what the overall parity of the word is supposed to be. For example, suppose that all words are stored with odd parity (every word is stored with an odd number of "1" bits); this may be accomplished by including a parity bit with each word stored. Then the missing bit in the word (3) above, for example, must be a "1".

We thus assume that each word is stored with an additional parity bit, so that single-bit erasures can be easily corrected. The word read from memory on port B is checked to see if it contains any erasures; if so, then the erasure symbol "?" is replaced by a "0" or "1" as necessary to satisfy the overall parity constraint.

Some versions of our scheme require the ability to correct more than single errors. The general theory of correcting erasure errors can be found in Elias [4], Peterson and Weldon [8], or Berlekamp [2]. While correcting a single erasure error only requires a single extra parity bit, correcting two or more erasure errors is significantly more work. In general, a linear error-correcting code with minimum weight of a code word equal to $w$ can be used to correct up to $w - 1$ erasure errors (see Peterson and Weldon [8, exercise 3.9]). (This is twice as good as the corresponding situation for random errors (that is, errors in unknown position), where only $(w - 1)/2$ errors can be corrected; one random error is as much trouble as two erasures.) This implies that for correcting multiple erasure errors each erasure error requires approximately an additional $\lg(b)$ parity symbols. The decoding circuitry required for two or more erasure errors is significantly more complicated than required for single erasure errors, but is not outside the realm of practicality.

## 3.4 Writing to memory

So far, we have presented our new design as if it were a multiport ROM. We now consider the complications that arise in handling write operations as well as read operations.

We begin by considering what restrictions arise if we wish to utilize the simplest single-error-correction scheme described above. The primary requirement is that each word is stored correctly with a correct parity bit. This implies that

1. the memory circuit generates the correct parity bit when writing a word into memory,
2. that at most one of the two ports be used for writing into memory on any cycle, and
3. if one port is writing and other is reading, then the port that is writing is given priority on any bin where the two operations conflict.

Restricting write operations to (say) only port A is not an uncommon feature; note, for example, that the Signetics 82S112 8×4 dual-port RAM [10], allows write operations through only one of its two ports. Condition (3) guarantees that no errors are introducing during the writing process itself because of conflicts. We believe this version of our scheme to be the most practical variation.

In some cases, however, it may be desirable to allow both ports to write simultaneously into memory. Encompassing this possibility is conceptually straightforward, though more expensive in hardware. Each word needs to stored with enough additional parity bits to enable correction of up to one erasure and one random error, because now the writing operation itself may introduce an erasure error, in addition to the error normally possible with a read operation, but unless the position of the erasure is also stored, this error will appear random when read later. That is, when two simultaneous write operations conflict, one of them can introduce an error anywhere in the word stored. Later on, when that word is read out, a second error may occur. The error-correction circuitry must then be able to correct both errors.

## 3.5 Generalization to more than two ports

The ideas described above can be generalized to handle more than two ports, as follows.

Suppose we implement a $p$-port memory, for some $p > 2$, using the addressing scheme as described in Section 3.2. Then $p$ words are accessed simultaneously, and each word can have up to $p - 1$ conflicts.

If we are implementing a $p$-port ROM, then we need to ensure that each word is stored with enough parity bits to permit the correction of up to $p - 1$ erasure errors. Without adding any additional parity bits, we can permit one port to write during each cycle (as long as that port gets priority in any conflicts).

A $p$-port RAM that allowed any port to write during any cycle would require that each word be stored with enough parity-correction bits to enable the correction of up to $2(p - 1)$ errors, since $p - 1$ can be introducing during writing and $p - 1$ during reading.

# 4 Conclusions

We have described a novel implementation of multiport memories, based on the use of single-port memories, an addressing scheme utilizing Galois field arithmetic, and error-correction. The scheme is particularly attractive from an engineering viewpoint for dual-port memories where at most one port is allowed to write during any cycle. Generalizations to more ports and a higher degree of parallelism during writes is also possible.

**Acknowledgments**

We thank Peter Elias and Tom Knight for helpful discussions.

# References

[1] F.E. Barber, D.J. Eisenberg, G.A. Ingram, M.S. Strauss, and T.R. Wik. A 2k×9 dual port memory. *International Solid-State Circuits Conf.*, pages 44–45, 302, 1985.

[2] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, 1968.

[3] Cypress. *Applications Handbook.* Cypress, 1989.

[4] Peter Elias. The noisy coding theorem for erasure channels. *American Mathematical Monthly*, 81(8):853–862, October 1974.

[5] Texas Instruments. *The TTL Data Book for Design Engineers.* Texas Instruments, 1976.

[6] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[7] Kevin J. O'Connor. The twin-port memory cell. *IEEE J. Solid-State Circuits*, SC-22(5):712–720, October 1987.

[8] W. Wesley Peterson and E. J. Weldon, Jr. *Error-Correcting Codes.* MIT Press, second edition, 1972.

[9] T. Sakurai, K. Nogami, K. Sawada, and T. Iizuka. Transparent-refresh DRAM (TReD) using dual-port DRAM cell. *IEEE 1988 Custom Integrated Circuits Conf.*, pages 4.3.1–4.3.5, 1988.

[10] Signetics. *Signetics Data Manual.* Signetics, 1976.

[11] B.A. Wooley T.-S. Yang, M.A. Horowitz. A 4-ns 4k×1-bit two-port BiCMOS SRAM. *IEEE J. Solid-State Circuits*, 23(5):1030–1040, October 1988.

[12] Yuzuru Tanaka. A multiport page-memory architecture and a multiport disk-cache system. *New Generation Computing*, 2:241–260, 1984.

[13] Neil H. E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design.* Addison-Wesley, 1985.