Functional cache chip for improved system performance

by Richard E. Matick

The use of a cache to improve the performance of computing systems is becoming very pervasive, from microprocessors to high-end systems. The general approach has traditionally been to use ordinary fast RAM chips and interface these close to the processor for speed. However, this is far from the ideal solution. The stringent and often conflicting requirements on the cache bandwidth for servicing the processor and minimizing reload time can severely limit attainable performance. The cache need not be the performance-limiting factor if a properly integrated functional cache chip is used. This paper defines the basic requirements of a cache subsystem and shows how these have been or could be implemented in typical systems. Subsequently, the functional requirements of an optimal cache chip design are presented and illustrated.

Cache subsystem overview: Cache or no cache?

The access times of high-density memory chips are beginning to approach the cycle time of many current high-performance processors [1]. Under such circumstances, one could consider building a reasonably high-performance microprocessor without a cache, using instead the static column capability of typical memory chips as a type of simple cache. While this is a distinct possibility, it has certain inherent fundamental limitations which may not be acceptable. For systems over which a range of cost and performance is desired, a cache should be an inherent part of the system organization and architecture. Some of the limitations of a cacheless design are the following:

1. The system organization is dominated by the memory organization; e.g., adding a cache later requires significant redesign or else is very inefficient.
2. The system cycle time is likewise very strongly determined by the speed and organization of the memory chips.
3. As the memory capacity increases, it is increasingly difficult to maintain the same processor cycle time.
4. If adequate system busing is not provided from the start, it is difficult and costly to change later.

All of these elements lead to the processor design being driven by the memory design. Because the natural evolutionary path is for the processors to get faster, the gap between processor and memory cycle time tends to widen. When this happens, the processor throughput becomes limited entirely by the main memory [2]. Since the processor and main memory will likely be built from different technologies, or at least by different processes, the processor speed can increase and main memory can stay constant at any point in time. Since the cache chip can be produced

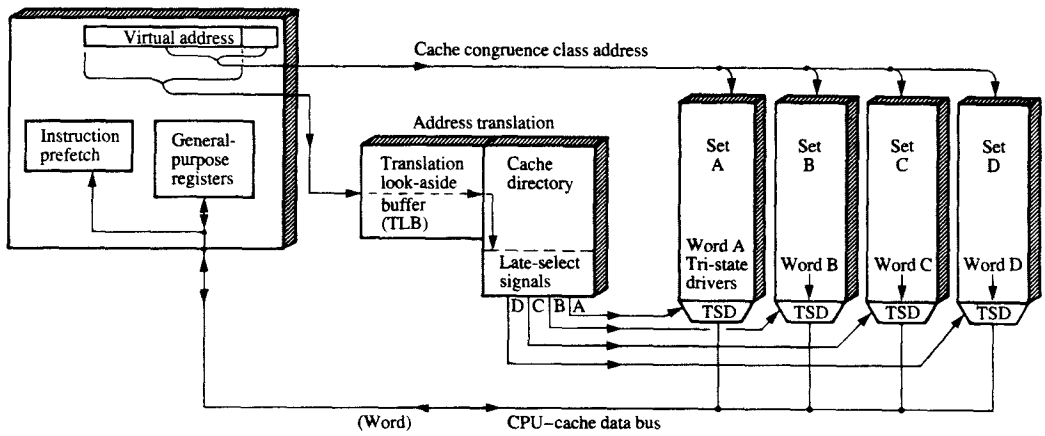

Figure 1

Schematic of a late-select, four-way set-associative cache organization.

from the same technology as the processor, the cache speed can track the processor speed. Hence, the use of cache allows the system performance to be only loosely dependent on the memory performance.

With a cache, we can devise one system architecture and a general system organization which can continually improve in performance as the processor (and cache) chip improves [3–6]. This eliminates the performance gap dependency between processor cycle time and main memory cycle time which continually occurs over time. This has been the historical trend, occurring repeatedly for fundamental reasons: namely, faster processors require more main memory, and larger memory is slower in the same technology. Any given system will migrate toward the largest memory affordable (see [2], Chapter 1).

Fundamentals of cache-accessing requirements

There are a number of conflicting requirements which a cache array must fulfill in order to provide the necessary functions with high performance. Achieving these with standard array designs typically leads to a rather complex system. The complexity and resulting high cost can be substantially reduced by understanding the functions which are required and properly integrating them into the array chips. In order to understand this, let us consider the fundamental accessing problems and typical methods of implementation. To do this, we use a very common type of cache organization as an example, implemented with relatively simple, single-port array chips, then a slightly more complex but still single-ported array chip, followed by a structure which is similar but uses a true two-port array (can support two simultaneous accesses to different addresses). This clearly shows the external complexity required to implement the full cache array structure. It is then shown how various parts of this complexity can be simplified by a more judicious design of the cache chip and, with the final embodiment, a simplification which can be used for micro-, mini-, or large computers. It will be seen that not only is a two-port array not needed, but also that a two-port array without other functions is inadequate as well as costly, and therefore a poor design/performance trade-off. (Note: The implicit assumption is that the path between cache and CPU is one word per cycle. For complex structures where multiple arguments are fetched simultaneously on the same cycle, a two-port array may be useful, but does not change the design issues with respect to the integrated functions for reload and performance as discussed here.)

We consider a late-select cache, organized as four-way set-associative, with an overall structure as shown schematically in **Figure 1**. During a normal read access, part of the virtual address is used to select the four possible words that could be correct, namely the congruence class, which consists of one word from each of sets A, B, C, and D.[1] Simultaneously, the

[1] These definitions make a set orthogonal to a congruence class and have been used widely for many years within IBM. They differ from those sometimes used in the literature, where a congruence class is called a set, which leads to confusion. See [7] for a definition of congruence class.

total virtual address is translated via the translation look-aside buffer (TLB) and cache directory to see which set, if any, is the correct one. If one is chosen, then "late" in the cycle the correct word is enabled from one of these sets by the appropriate late-select signal A, B, C, or D, and placed on the CPU–cache data bus.

Typically, the data-out ports of any chip are implemented with tri-state drivers, so the four words can be dot-ORed together as shown. These drivers have a DATA ENABLE input signal, allowing any one word to be selected and placed on the CPU–cache data bus. During a write access, a problem is encountered. Typically high-speed static FET memory chips require that the data be valid at the chip boundary before the chip access is initiated (this is also typically true for dynamic memory chips which might be used in a "main memory" version of this concept). For a late-select cache design, this presents a problem, since we wish to start the chip accesses in parallel with the translation, and the data cannot possibly be valid until the translation is complete. Typical caches use a read-modify-write operation which requires two cache cycles and reduces system performance. Functionally, it is desirable to have a cache which can perform a late-write operation. This can be done without impacting the cache performance, but requires special design of the chip.

Whenever the address translation indicates a cache miss, the cache block must be fetched from main memory and reloaded into the cache. For complex-instruction-set computers (CISC), which require a relative large number of processor cycles per instruction executed, there are often enough free cycles to allow this reload process to be relatively slow, with a (barely) tolerable degradation of system performance. However, the trend in processor design has been to reduce the number of processor cycles per instruction executed, and such designs place severe demands on the overall memory subsystem bandwidth. For instance, Figure 6 of [8] shows that for a high-performance processor pipeline designed to achieve an average of approximately 1.25 cycles per instruction executed, assuming an ideal memory system (e.g., infinite cache), the reload penalty for a finite cache at a typical design point can be an average of 5 percent reduction in millions of instructions per second (MIPS) for *each additional cycle of reload required.* Thus, for high-performance systems this typically requires that the reload take place as quickly as possible. Since the memory access time is generally some fixed value, additional performance is obtained by reloading multiple words on each cycle, once the first main memory access has started. Loading multiple words into the cache on a miss presents several problems. First, the reload requires that all the words be placed into contiguous logical locations in the "same set"; i.e., all the words go to set A, or all to set B, etc. For instance, if we reload four words on each cache cycle as in **Figure 2**, all the four-word I/O ports of Figure 1 must now

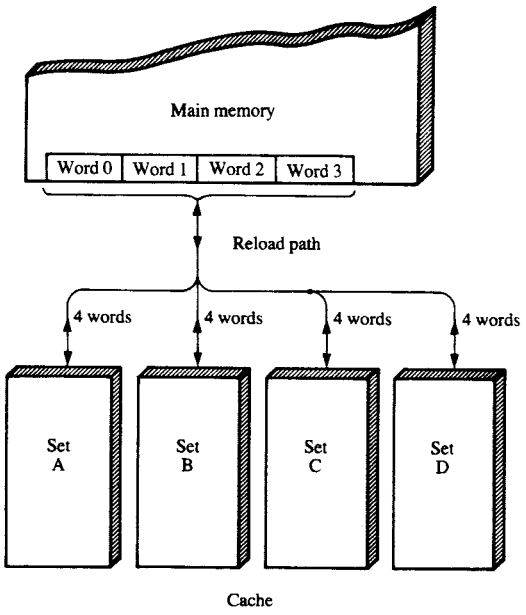

Figure 2

Main memory–cache data path for reload of four words each cycle.

be connected somehow to set A (or set B, C, or D if the replaced block is in that set). There are some additional complicating requirements; e.g., on a miss, the reload should start on the word that caused the miss so that it can be "loaded through" to the CPU for processing in parallel with the reload. All of these requirements are very different from the normal access in Figure 1, where one word from "each set" is accessed, on a word boundary. These conflicting accessing requirements create some problems in cache design. We now consider how these requirements can be and have been met, starting with simple arrays and progressing to more sophisticated designs and finally the "functionally integrated" design proposed here.

Case I: Single-port, single-word cache array modularity

Suppose we have available cache array chips which allow an array design of one word per unit chip as shown in **Figure 3**, where the unit making up the word can be one chip, as in part (a), or several chips, as, for example, four chips in part (b). The number of chips and the partitioning used to obtain a word are a function of the cache size and chip modularity which is not important here. In the following discussions, for simplicity, we use the representation of Figure 3(a) as a "chip

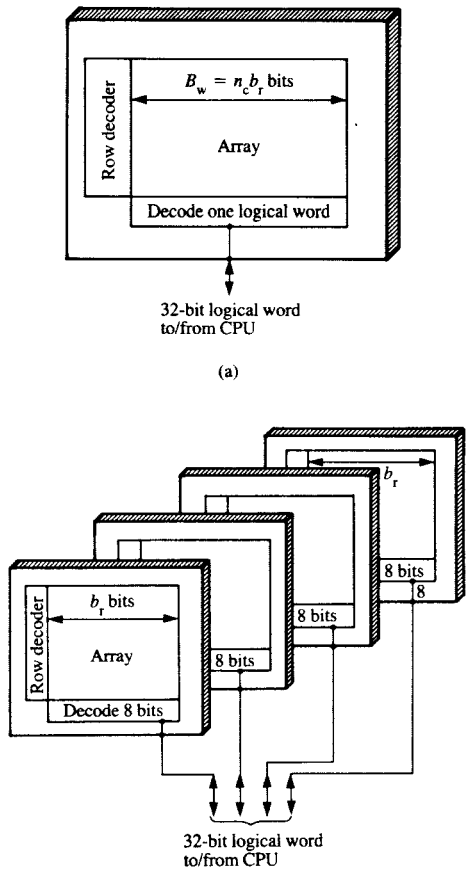

Figure 3

Schematic of chip unit comprising system logical word of 32 bits. (a) Chip unit = one logical word to/from CPU; n_c = number of chips in chip unit. (b) Chip unit of n_c = 4 chips, 8 bits per chip.

unit" which supplies one word of I/O to or from the CPU, but it should be understood that more than one chip can be implied, and in various configurations.

With such a chip unit, it is possible to build a four-way set-associative, late-select cache which can also reload multiple words on each cache cycle (e.g., two, four, or more words per cycle). The manner in which this is achieved is a function of the relation of the required overall cache capacity and number of bits per chip that the technology can provide, i.e., the modularity of the chips and system. In order to understand the problems and trade-offs, suppose we have a single-port chip unit as in Figure 3(a) with a one-word I/O port. Further suppose that the desired total cache capacity and available chip unit density are such that a total of eight chip units are required. If the set associativity is four-way, these eight chip units will map to two chip units per set of the associativity, as shown in **Figure 4**. In such a case, it is possible to reload a maximum of two words per reload cycle, since each set has two chip units and hence two independent I/O ports available to main memory. On a normal CPU access, the word address bit accesses one of the two rows of chip units, and one word from each of these four chip units is accessed and held at the edge of the chip unit, one word for each of the four sets, i.e., the congruence class. The late-select signal selects one of these four, and it is placed on the CPU data bus via the CPU MUX (multiplexor). This MUX is obviously necessary, since the I/O lines out of each chip unit cannot be dot-ORed except as shown, even though they are from tri-state drivers. The reason for this is obvious: namely, for reload, two separate words traverse between main memory and the array, one word for each row. Thus the off-chip MUX is necessary. Of course, one serious drawback is that this structure cannot support a simultaneous reload and CPU access because of the one-port design of the chip units. This results in access interferences and degradation of performance, which can be reduced by a separate interface to main memory for reload. However, another limitation which is not a result of the one-port design is that this configuration cannot support any more than two words of reload per cycle. For instance, in Figure 4, if a reload of four words per reload cycle were desirable, this configuration could not support it, even if the chip units were two-ported arrays with one of the ports used for a separate bus to main memory for reload. This results from the fact that each set A, B, C, or D is contained on only two chip units, with one-word I/O per chip unit or two words maximum per set for reload. Thus, a simple way to increase the reload path width would be to add additional chip units. The use of 16 chip units with four per set would provide a four-word reload path as desired. However, the cache capacity has been doubled, which increases the cost, package size, and delay, and is not typically acceptable. The fundamental design problem is that technology improvements increase the array bit density per chip faster than the system has increased the required cache capacity. The net result is that the *number of chip units per system has greatly decreased with time, and this trend will continue.* Thus, the designer of the "next" system typically has fewer chip units available and a potentially smaller reload path. For instance, suppose that for the next-generation design of the cache in Figure 4 the chip density increased by a factor of four, while the cache capacity increased by a factor of two. Hence, only four chip units are required instead of eight, so the organization of Figure 4 would provide only a one-word reload path, which is very undesirable. Obviously, if the I/O path of each chip unit were increased from one to two words, a two-word reload path would be possible. However, this would be achieved at considerable expense, since only a one-word path to the CPU is desired, thus

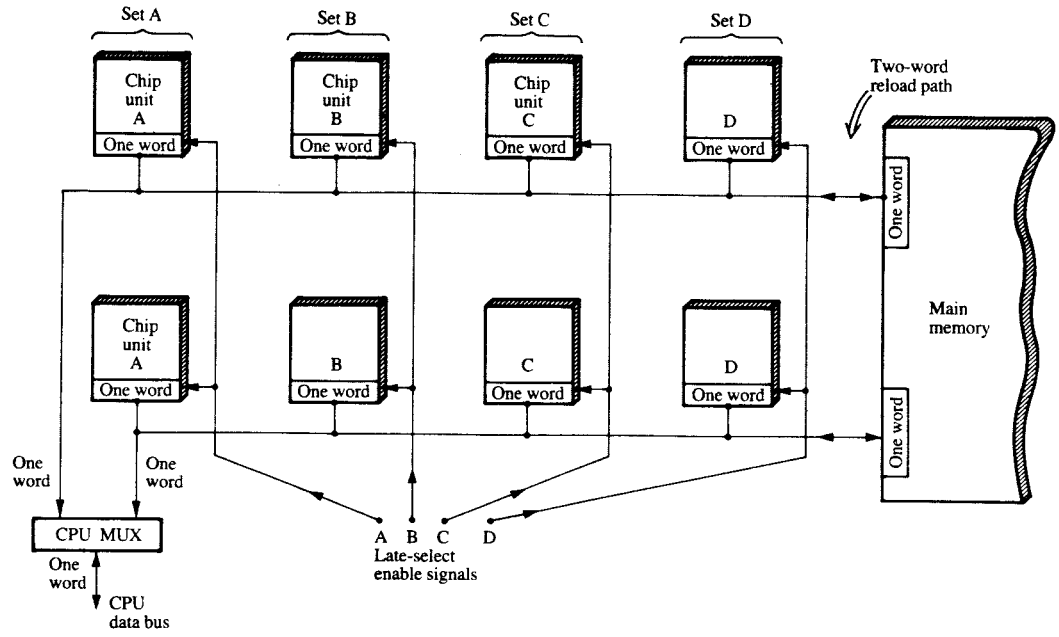

Figure 4

Late-select, four-way set-associative cache organized using single-port chip units with two-word reload path, requiring two chip units per set.

wasting the most important parameter, cache bandwidth. Other solutions are desirable and possible, as will be seen below.

Notice in Figure 4 that there are a total of eight chip units, so that during reload there are six potential I/O ports which are sitting idle. These could be used for increasing the reload path width if we could spread the words of each set over each chip unit. Then, during reload, one word could be reloaded to each chip unit for a maximum of eight possible words reloaded per cycle in this case. (Of course, fewer could be reloaded if desired.) However, this improvement requires a special type of mapping of the logical cache blocks (the replaceable unit) to the physical array structure, sometimes referred to as "Latin-square" mapping[2] [9, 10]. This mapping, using the cache chip unit of Figure 3(a), is shown in **Figure 5** for a four-way associative, late-select cache design which requires only four chip units. (Additional groups of four, similar to that in Figure 4, could be added above these with appropriate interfaces.) The need for this rather complex mapping arises from the lack of an adequate reload interface coupled with the small number of chip units needed for a typical cache. The design point at which such a mapping is required can be specified as follows. Let

N_{cu} = total number of chip units in cache,
n_{rl} = number of words reloaded to each chip unit per reload cycle,
W_{rl} = total reload path to cache in words per cycle,
S = set associativity of cache organization.

A late-select cache will require the Latin-square-type mapping of Figure 5 if

$$\frac{(N_{cu} \times n_{rl})}{S} < W_{rl}. \qquad (1)$$

For instance, in Figure 5, $N_{cu} = 4$, $n_{rl} = 1$, $W_{rl} = 4$, and $S = 4$. When these are substituted into Equation (1), the left-hand side yields 1, which is less than W_{rl} which is 4; hence the Latin-square mapping is needed. Note that this would still be true even if the set associativity were reduced to $S = 2$. The reload path width of four words can be obtained without this mapping only by increasing the number of chip units or by decreasing the set associativity. For example, in Figure 4, there are eight chip units and the reload path is only two words, so $N_{cu} = 8$, $n_{rl} = 1$, $S = 4$, and $W_{rl} = 2$. Substituting these into Equation (1) yields four for the left-hand side, which exactly equals W_{rl}. Thus a Latin-square mapping is not needed, as shown in Figure 4.

[2] A Latin square of order m is a $m \times m$ square array of m elements, with each row and column a permutation of the elements.

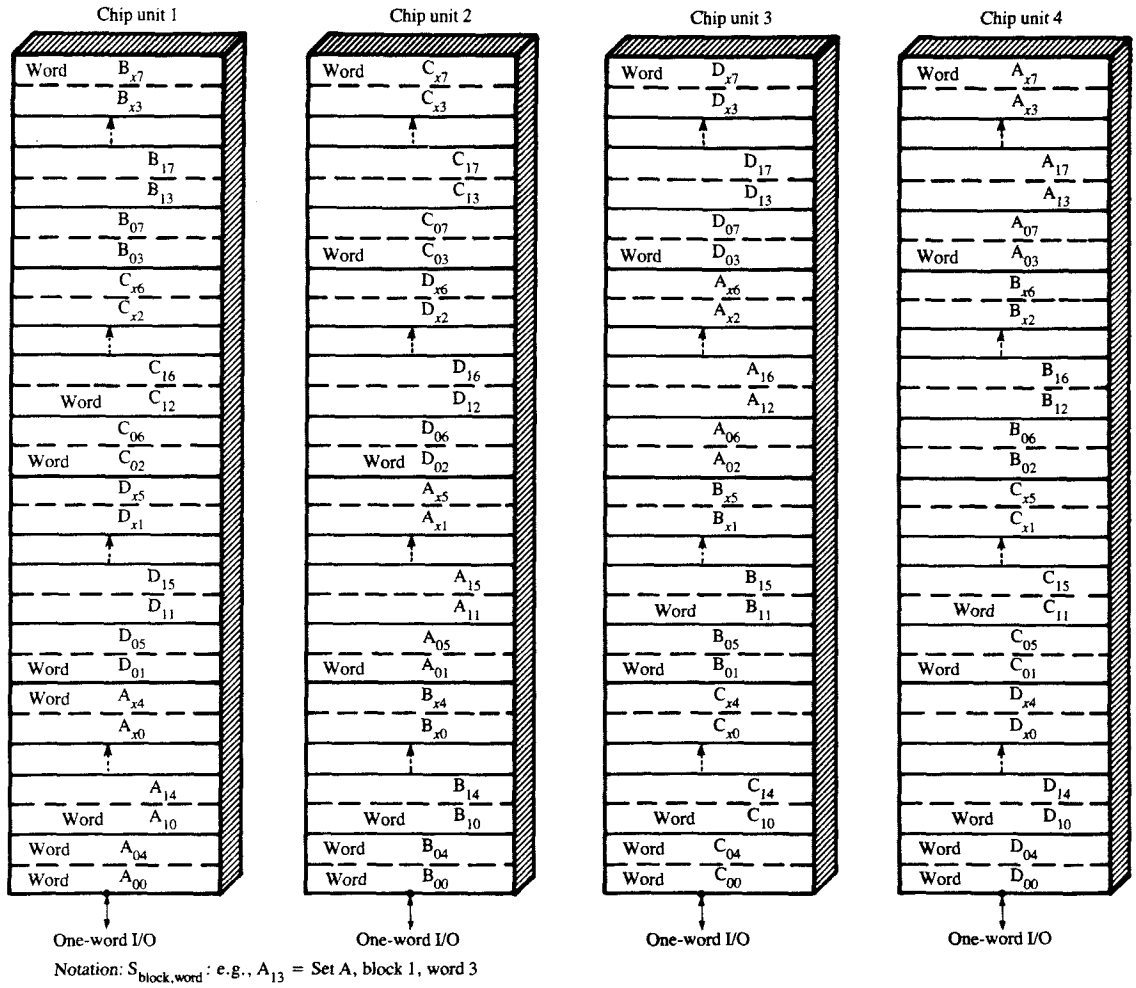

Figure 5

Logical-to-physical mapping for spreading each block across each chip unit in a late-select, four-way set-associative cache with eight logical words per block and x blocks total.

The mapping of Figure 5 is used in the following manner. During normal accesses, the same address is applied to all chip units in order to access the congruence class, composed of the corresponding word from each set, e.g., word 1 from each of sets A, B, C, and D. Thus, these four words must be at the same address on each chip unit, and the same is true for word 2, 3, 4, etc. However, during reload, only one word can be written to each chip unit, and if we wish to reload, say word A0, A1, A2, and A3 on the same cycle, obviously this can be done only if each of these words is on a different chip unit. The same holds true for all other groups of four words. The mapping of Figure 5 provides the proper distribution of words on the chip units, but two problems are encountered during reload. First, because contiguous words of any given block are stored at different addresses on each chip, each chip must receive a different (partially different) address. Since the starting address depends on which set is being reloaded, the addressing logic and bus for reload are more complex. A further complication arises in that a given word from main memory, e.g., word 1, can reside on any of the chip units, depending on the set being reloaded; hence, a ring-shift data aligner is required between the cache and main memory as shown in **Figure 6**. Additional complexity is introduced in the late-select logic. Since the words of any set can be on any chip unit, the late-select signal must not only choose the set, but also match the appropriate word and set according to the mapping, as shown in Figure 5, in order to enable the correct chip unit. A final complication

arises from the single-word data path into/out of each chip unit. During normal access, the four words must merge into one word to/from the CPU. On reload, the four words must be separate, allowing one-word I/O to each chip unit. For the assumed chip unit of Figure 3(a), this would require an off-chip multiplexor of some sort, such as that shown by the MUX in Figure 6. While there are many ways and places for providing this function, it must be included somewhere. If this function is placed on a separate chip, the extra chip crossing and multiplexor logic delay are added in the most critical access path, which is extremely undesirable. Ideally this multiplexor function should be done on the existing chips, and the delay should be overlapped with other, unavoidable delays. The proposed functionally integrated chip totally eliminates this multiplexor and delay, as will be seen later.

The additional circuitry required by this organization for accessing is only one aspect of the total problem. Another problem is that even though the reload bandwidth has been improved, it still is far from ideal. Since there is only one I/O port on each chip, then only one access, either for a normal CPU cycle or for a reload cycle, is possible on each system clock period. A miss and subsequent reload typically start at the word causing the miss—this word is immediately loaded through to the CPU and the CPU resumes processing. If the next CPU cycle requires a cache access, either the CPU or the reloading must wait, with appropriate logic for sensing and restarting. Regardless of which alternative is chosen, either CPU or reload-wait, the overall system performance is degraded. A further degradation is encountered from the same access interference problem if a store-in cache is used. This term means that the cache contains the latest copy of the correct data, so if any changes have been made to a block, it must first be written back to main memory before it can be removed from the cache. With high-performance systems, store-in cache is a better cost performance design, so this produces many more opportunities for access interference and degradation. Let us now consider how we might improve on the above design, first with simple additions to the cache chip.

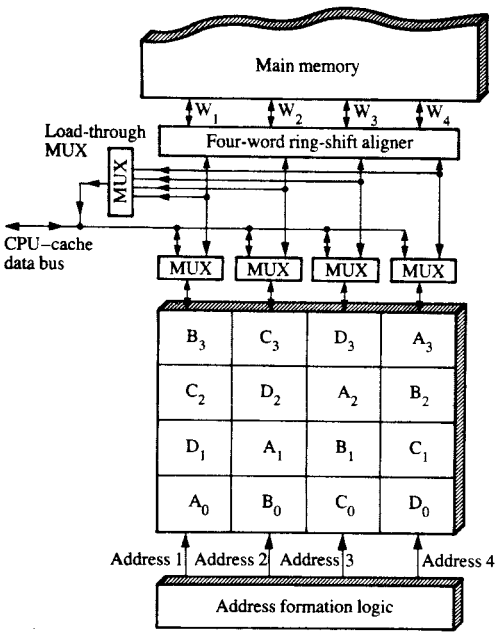

Figure 6

Some additional functions required to achieve an improved reload path via the mapping of Figure 5.

Case II: Single-port, single-word array access with on-chip bus multiplexing and load-through

To minimize critical path electrical delay while simplifying the overall busing, a few simple functions can be added to the cache chips without disturbing the array design; i.e., the functions are added entirely on the periphery of the array boundary. From the above discussion, it is clear that adding the multiplexor function of Figure 6, as well as the related "load-through" path, will substantially help to remove some of the limitations of the above organization. For instance, the chip unit in **Figure 7** has a one-word bus to main memory and a separate one-word bus to the CPU; however, the storage array itself is still only a one-port design to minimize cost and maximize bit density. The small amount of multiplexing necessary to achieve this is included on-chip as shown. (Note that the ring-shift aligner shown in Figure 6 is not on the chip but is part of the main memory interface.) In addition, if a load-through buffer (LTB) is also added, as in Figure 7, additional improvement can be obtained in some limited cases. For instance, if a "read" access to word A0 causes a miss and each chip has a load-through buffer, then the four words A0, A1, A2, and A3 are loaded into both the array and the buffer. Thus, these words are available from these buffers on subsequent cycles. Note that since four words are reloaded each cycle, only the first group of four is loaded into the LTBs; the subsequent words go only to the array. If sufficient logic is included to identify these first four words, any of them could be loaded through to the CPU via the load-through path, as needed. On the next cycle after loading-through word A0, if the CPU accesses word A1, A2, or A3, it can be fetched from the LTB on the appropriate chip, without interfering with the reload of the second group of four words from main memory to the cache. Of course, if the CPU writes to any of these words, or if the access is to a word other than one in any of the LTBs,

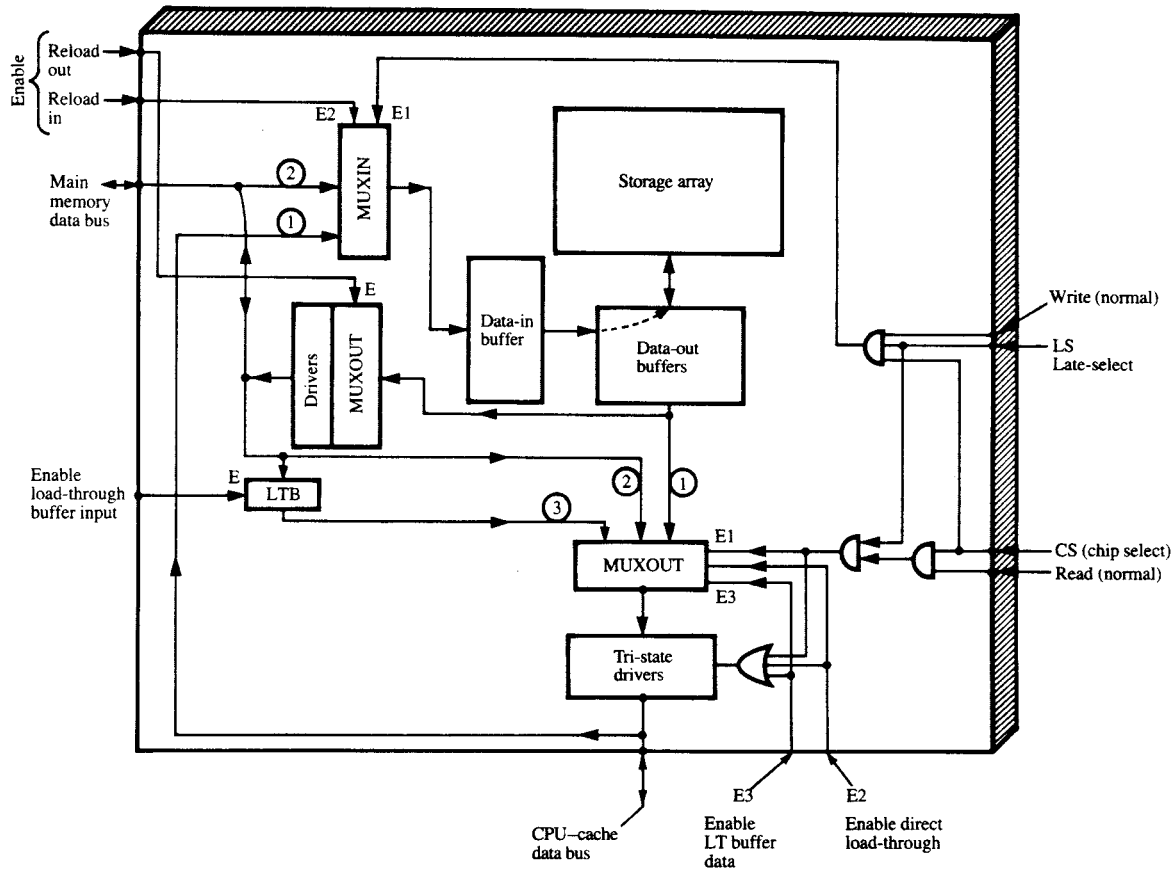

Figure 7

Modified static RAM chip for cache applications.

an interference is encountered. Such LTBs can be valuable for instruction fetches, which tend to be sequential; since data fetches tend to be more random, the load-through buffer will be of some, but limited, value. Even for sequential I-fetches, the LTB does not necessarily eliminate interference between reloading and CPU accesses. For instance, suppose the word causing the miss was A3. A subsequent reload will put A0, A1, A2, and A3 into the LTBs. A sequential I-fetch will next access A4, which is not in any LTB and must wait for the next reload cycle and access to the array itself. The next I-fetch to word A5 will not have this word in the LTB, hence an interference. Of course, complex logic could be used to latch the second set of four reloading words into the LTBs in this case, but the cost is high and the reward is small. The functionally integrated cache is a better design, as will be seen. In all cases, if access is permitted to partially loaded blocks, word-valid flags and logic in the CPU are required in order to know which words are accessible.

Case III: True two-port array with single-word access on each port

It should be clear from the above discussions that the interface between cache and main memory is quite different from that between cache and CPU. Since these two interfaces are best satisfied with two buses operating with different addresses, it would seem appropriate to use an array which is truly two-ported, allowing two simultaneous random accesses to the array. While this is possible, we will see that this provides more than what is required for some of the problems, but not enough to solve all the problems; in other words, it is not the ideal solution.

A cache organized similarly to that of Figure 6 but using a two-port chip unit is shown in **Figure 8**, with one port interfaced to main memory for reload and the other interfaced to the CPU for normal accesses. For reload, a separate ring-shift data aligner and shift logic, plus a separate address bus per chip unit with address logic, are still required, much as before. Since each I/O port has a separate address input, the CPU address can be separate, with one bus to all chip units as shown. The two ports have eliminated the need for separate multiplexors on the data bus—they are built into the additional complexity of the cells and separate word/bit lines and decoders. The load-through buffers of Figure 7 are no longer needed, since once a word has been loaded, we have random access to it via the CPU port if we maintain logic in the CPU for specifying which words have been reloaded. However, the load-through path in Figure 7 may still be needed, even though a two-ported array is used. The reason is that a two-port cell design is considerably simpler if a write and simultaneous read are not permitted to the same cells, i.e., do not read and write to the same word. If this is the case, then either a separate load-through path is required, or an extra cycle of delay is encountered before the CPU can restart. In addition, the logic for comparing the addresses for the two ports and granting access must be done in the CPU—the cache is a slave and will produce errors if used improperly. The store-back of modified blocks which added to the reload time of the case of Figure 7 is no different for the two-port organization of Figure 8. Considering the fact that a two-port cell/array design itself, without including the additional drivers, decoders, and other logic which is necessary, consumes approximately 30 to 50 percent more area than a one-port design and is slower, we have paid an enormous price and have gained very little in return. Thus, this type of two-port design is definitely a poor choice.

Simplification of reload path

In all the cases above, it should be clear that one major obstacle is the complex requirements of the reload path. In a typical, ordinary RAM chip there is only one decoder which

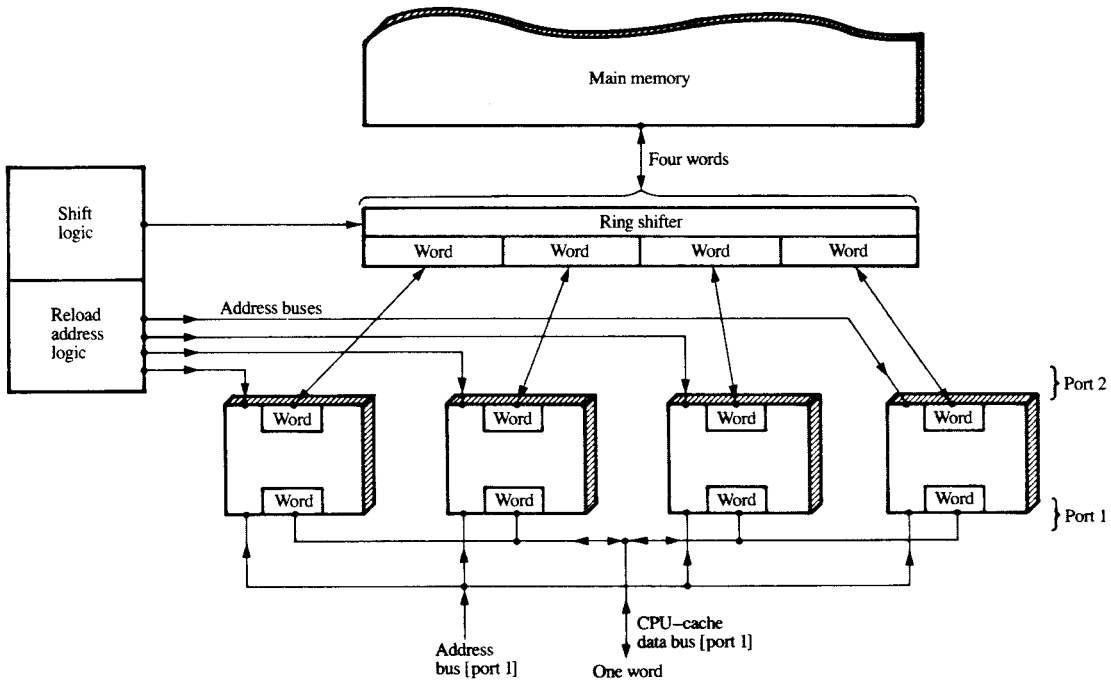

Figure 8

Late-select, four-way set-associative cache using true two-port arrays with single-word/chip unit access on each port and word reload via mapping of Figure 5.

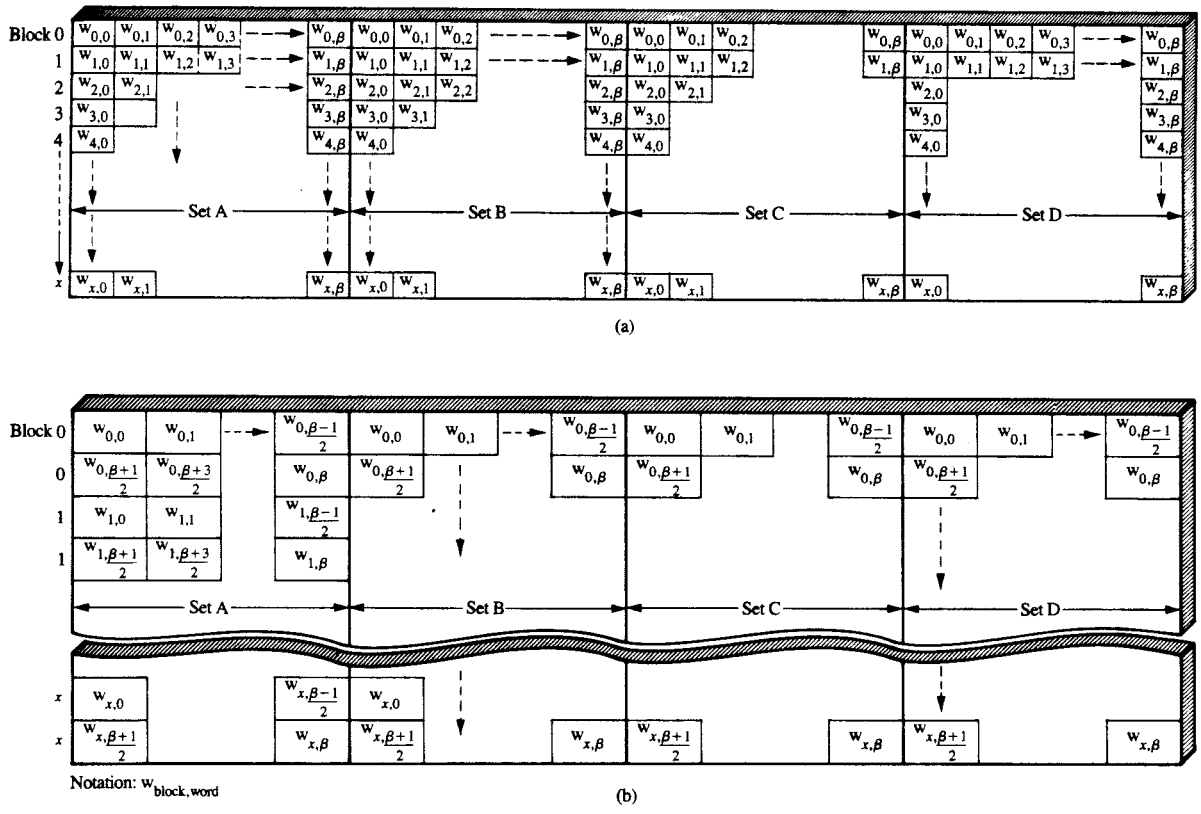

Figure 9

Mapping of sets, blocks, and words to chip units for late-select, four-way set-associative cache having $x + 1$ blocks of $\beta + 1$ words per block for (a) one and (b) two rows per congruence class.

selects the bit lines. As a result, there is only one fixed interface to the external world for any and all accesses. A RAM chip used for a cache, as described previously, requires a typical type of interface when communicating with the CPU, but a very different interface when reloading to/from main memory. A substantial improvement can be achieved by proper integration of the essential functions directly into the cache chip. In order to achieve this, a different and special mapping of the words and blocks to the physical arrays is required and used in conjunction with a cache array having an additional, relatively simple bit-line decoder. This mapping is special in that a physical word (row) address must access the multiple words of any given congruence class for normal access, and multiple words from the same set for reload. This requires that multiple words from each set must reside at the same row address. **Figure 9** shows this mapping for two chip configurations, each having $x + 1$ cache blocks with $\beta + 1$ words per block. In Figure 9(a), any given block resides at only one row address, while in Figure 9(b) any block resides at two adjacent row addresses. In Figure 9, any row address to a single chip will select the same bytes of the same words of each set (i.e., the congruence class). In addition, the same row address also accesses some portion of a cache block from each set as required for reload. The minimum number of bits per row required to achieve this mapping is a function of the cache organization, system, and chip parameters, as will be seen below. This mapping, when properly integrated onto the chip, allows a four-way set-associative, late-select design with multiple-word reload, the actual number depending on the number of pins and decoding provided on the chip. This mapping can be used with either a one- or two-port cell/array design, although the latter will be seen to be unnecessary unless, of course, the CPU itself needs to access two separate words. When properly implemented, this mapping eliminates the reload ring-shift data aligner,

requires only one address bus connecting all chips, and lends itself to inclusion of other simple functions on-chip to significantly assist reload, at small cost, with essentially no additional delay, and using a one-port array design.

The mapping of Figure 9 can be used with various levels of complexity on the cache chip to improve performance. At the lowest level, an additional bit-line decoder can be added to eliminate the ring-shifter and the separate address per chip unit. One simple example illustrating how this can be achieved is shown in **Figure 10**. The bit-sense lines are connected on each end to a different bit decoder. The top bit-line decoder interfaces to main memory for reloading, while the bottom one interfaces to the CPU for normal accessing. Note that while the chip itself has two separate data buses, one to main memory and one to the CPU, the cells and arrays are single-ported. This requires that both buses cannot be busy simultaneously (we later remove this restriction).

The importance of the mapping and its implementation requires considerably more detail. Suppose a 64K-byte, four-way set-associative, late-select cache is implemented from 128K-bit chips, arranged as 256 rows (physical word lines) by 512 columns (bits per word line) as in Figure 10. It is further assumed that the bottom column decoder on each chip decodes 4 × 16 bits out of 512, where each group of 16 bits is one half-word from each of the four sets, A B C D (i.e., the congruence class). These four half-words are held in the output latches until the late-select signal selects one of the four and dumps it on the CPU data bus through tri-state drivers. Each chip supplies half the word for a 32-bit logical word. The bottom decoder provides the selection for a maximum of four-way set associativity. The same physical structure could provide a two-way associative cache—the sets on each chip would be mapped A A B B or A B A B rather than A B C D, and the late-select signal would have to be combined externally with the appropriate logical word address (similarly for the reload path). The same chip in a modified organization could provide eight-way associativity, but is usually not necessary. The top decoder provides 32 out of 512, where the 32 bits are all contiguous bits and represent two contiguous half-words from the same set.

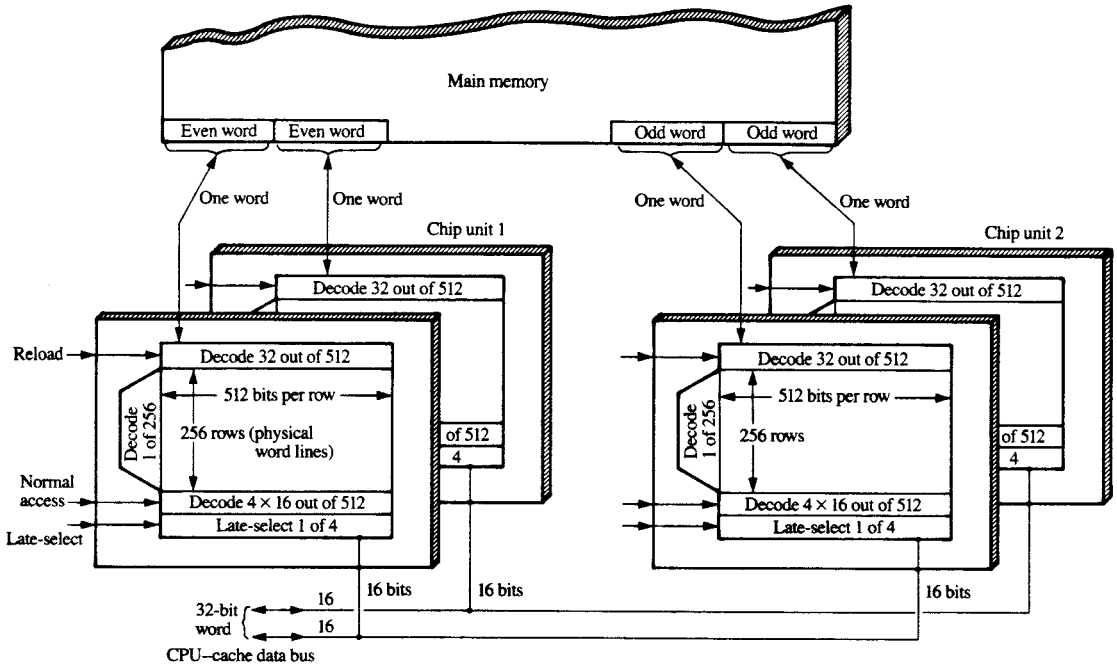

Figure 10

Elimination of ring-shift data aligner and data bus multiplexor by proper mapping and on-chip decoding, in a 64K byte, four-way set-associative, late-select cache with a four-word reload path.

For this assumed configuration, a 32-bit logical word length requires two chips per chip unit in Figure 3(a), and this chip unit has the capacity to reload two words per cycle from main memory. The total number of words which can be reloaded per cycle depends on the number of chip units in the cache. Assume that two chip units are required, as in Figure 10, for a total cache capacity of 64K bytes. The reload path can then be four words wide. One reasonable arrangement for reloading from main memory would be to put even words on one chip unit and odd words on the other (any other arrangement is also workable). The ring-shift alignment is done by the 32 out of 512 bit decoders on each chip. The need for a separate address bus to each chip is eliminated by the use of the mapping of Figure 9 with a control signal indicating "reload" or "normal access," since only one or the other path can be active at one time. The same address bus is used for CPU accesses and reload. For a reload, the upper decoder is activated on each chip, and the bit address selects four consecutive words on the four chips, where all these words are within one set as required. For a normal access, the lower decoder is activated and four words of 32 bits each, one from each set, are selected on one or the other of the two chip units, as determined by the even/odd word address. The late-select signal enables one of these to/from the CPU data bus.

Note that the organization of Figure 10 need not use the exact mapping shown in Figure 9. In fact, the minimum number of bits per physical word (row) on each chip is 128 (i.e., 4×32) bits in this case. The required minimum number of bits per row can be related to the other parameters as follows. Referring to Figure 9, let

b_r = number of bits per row on each chip,
L_{lw} = number of bits per logical word (32 bits in our examples),
W_{rl} = total number of logical words reloaded per cycle,
N_c = total number of chips in cache,
S = set associativity.

For the configuration of Figure 10, the *minimum* number of bits per row on each chip is given by

$$b_r \geq \frac{S \times L_{lw} \times W_{rl}}{N_c}. \qquad (2)$$

Of course, as the chip configuration is changed, the bit-line decoders must be changed accordingly. If the reload path is decreased to two words per cycle, while the set associativity remains at four, with four chips, from Equation (2) the minimum number of bits now becomes 64 bits per row. The I/O path to main memory is likewise reduced to 16 bits per chip.

While the implementation of Figure 10 is workable, neither the store-back of a modified cache block nor the reload can take place simultaneously with normal accesses because of address-bus as well as array-access contention, so there will be interference and significant performance degradation. The key is to design a chip using a one-port array while providing all the functions and not compromising the speed. All of these requirements can be achieved in the organization of Figure 10 by the inclusion of two very simple registers on-chip, namely a store-back buffer and a cache-reload buffer, along with a simplification of the top bit-line decoder, all described below.

Functionally integrated cache chip

A schematic of a cache chip unit which properly integrates all the above functions, while still using a one-port array design and requiring only one address bus to all chips, is given in **Figure 11** (only data-flow shown). This is achieved by the use of the mapping of Figure 9, properly integrated bit-line decoders, and two special buffers with associated addressing logic. In Figure 11, each chip unit interfaces to the CPU data bus via a bit-line decoder which selects one word from each of the four sets within the array, followed by a multiplexor which uses the late-select signals to select one of these four words, identical to Figure 10. However, the interface to main memory shown in Figure 10 is significantly improved as follows. The bit-line decoder at the top of the array selects, for any given set and block address, the total portion of that block which resides on that chip unit rather than just the one word decoded in Figure 10. This requires less decoding than the top decoder in Figure 10. Furthermore, the data bus in and out of this decoder does not go directly to main memory but rather is dot-ORed to the store-back buffer and cache-reload buffer as shown in Figure 11. These two buffers interface to main memory on one bus per chip unit as shown. The operation of these buffers can be made nearly transparent and can give the chip an appearance of a two-port array by including some of the controls on the chip. The special functions on this cache chip are the following:

1. Store-back buffer (SBB); since the cache is a "store-in" cache, a cache miss requires that any modified block be rewritten back to main memory. On one cache cycle, any block can be fully written into this temporary buffer to allow reload to proceed as fast as possible. The cache controller issues the necessary signals to achieve this action. These signals enable the top set of bit switches as well as the SBB, and the latter latches whatever data are on the SBB/CRB array bus. During reload, these data just remain latched in this buffer. After reload is completed, the SBB is written back to main memory, while the CPU can simultaneously access the cache array. The writeback is performed under the guidance of the cache controller in conjunction with some on-chip controls such as a two-bit counter which points to the next entry and is automatically incremented for each unload cycle.
2. (a) Load-through path; on a cache miss, the first word fetched from main memory and passed to the cache is

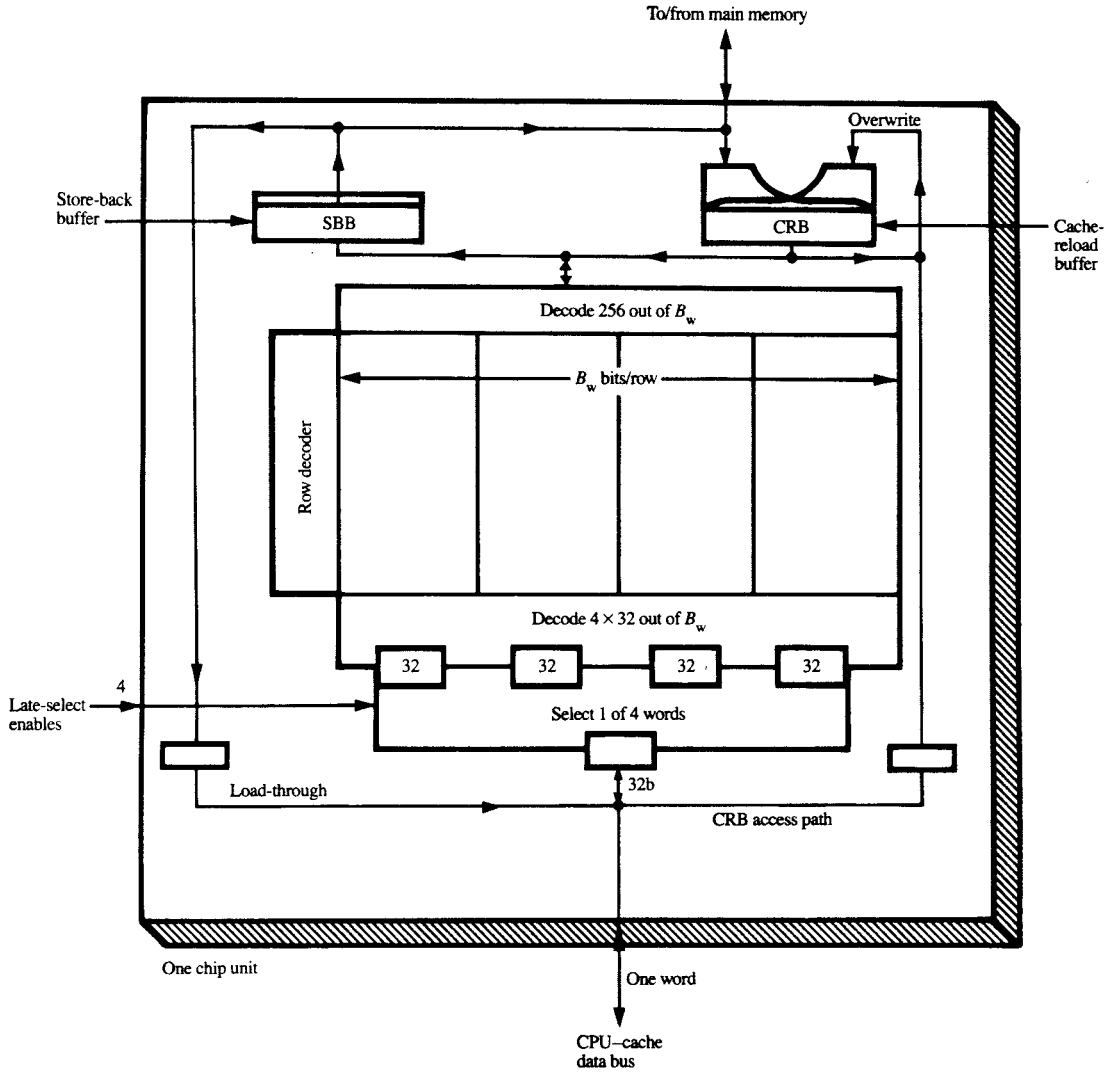

Figure 11

Functionally integrated chip unit using one-port array cells.

the word which caused the miss (either read or write miss). The cache chips will pass this first word from the main memory data bus directly to the CPU data bus via the load-through path for an immediate load on the same cycle. If the miss was for a write, the simplest design is still to place the data on the bus but not have it latched by any unit—this could be changed to load-through for only a read miss but is unnecessary. Only the first word is loaded through. Subsequent words can be obtained from the cache-reload buffer after they have been reloaded. Note that if W_{rl} words are reloaded on each cycle, on the cycle after the first word is loaded through, any of these W_{rl} words are available to the CPU. The functions required for load-through control can all be placed on-chip since they are quite simple.

For a WRITE miss, the word which caused the miss must first be loaded into the CRB, after which it can be overwritten as described below.

(b) Cache-reload buffer (CRB); reloading words from main memory go first into the CRB. While this is taking place, the CPU may read or overwrite any words which have already been loaded into the CRB via the CRB access path, or any other data already resident within the cache array. The additional controls required for these features are not very large nor complex. Some or all of them can be placed on the cache chip, depending on the system pipeline,

technology available for the cache, and the performance required. In general, higher performance is obtained by placing more of the controls directly on the chip to give the chip an external appearance of being a two-port array. In other words, simultaneous accesses from both the memory side and CPU side are handled by the chip, which decides whether the information is in the CRB or array, and delivers it with minimal outside control. The basic functions which must be provided, either on-chip or elsewhere, are as follows:

- Reload address register (RAR), a register which holds the address of the block and word-group W'_{rl} being reloaded.
- Set ID register, which holds the ID of a set which is being reloaded and contains either S bits (decoded, direct set enable) or s encoded bits, where $s = \log_2 S$.
- Word-group valid flag register (WGVF), a W'_{rl}-master/slave register which holds one valid flag for each word-group in a cache block; if there are W'_{bl} words per cache block, then there are a total of W'_{bl}/W'_{rl} word-groups and an equal number of such flags to specify which of the word-groups have been reloaded. If two words are reloaded each cycle and the cache block contains four words, then two such flags are required.
- Compare and CRB control circuits for automatically selecting the CRB or array as appropriate. The details of such control are technology- and design-dependent, thus are not shown, but are relatively straightforward.

The overall functioning of this chip unit is as follows. Suppose a cache access miss occurred and the block to be replaced has been modified. Assuming a store-in cache, this block must be removed from the cache before the new block overwrites it. However, if this block were totally written back to main memory before the reload process started, a very significant performance degradation would result, especially as the CPU design attempts to reduce the average number of cycles per instruction. Thus two actions are initiated simultaneously; first, the reload request to main memory is started, and at the same time the modified cache block which is to be replaced is temporarily stored in the store-back buffer (SBB) (the latter action only requires one cycle). As soon as main memory can start the reload, the incoming words are placed in the cache-reload buffer (CRB). The preferred design is a CRB which can hold an entire cache block, although a partial-block design is also feasible. A full-block design is assumed. Since typically only a part of a block is reloaded on each cycle, several cycles and some addressing/decoding are necessary to properly access the CRB. This is done by a very simple decoder which can be located on-chip or elsewhere. The address for the reloading words need be only the higher-order bits of the block index bits, which will typically be two to four bits and thus four to sixteen cycles for reload. These bits and other necessary bits are stored in the reload address register (RAR). Because these bits were present on the CPU–cache address bus at the time a miss occurred, they contain the address of the word that caused the miss and hence is used for control of both the load-through buffer and the CRB. Once the CRB has been fully loaded, it requires one cache cycle to load this into the array. This can be done immediately, or, preferably, after another miss is encountered. The latter can be done during the cache idle time while the next block is being fetched from main memory, thus avoiding a lost cycle.

- *Minimum mapping for functional chip*

For the configuration and assumptions used above, the chip-unit configuration requires a minimum of one entire block from each of four sets to reside at each row address. This results from the size of the cache block, the set associativity, and the specification that the SBB will be loaded from the array on one cycle and likewise that the CRB will be unloaded to the array on one cycle. We could relieve this restriction by specifying that the SBB be loaded from the array and CRB be unloaded to the array in multiple cycles, since there will typically be two or more cycles idle for each of these. If we choose two cycles for this, one entire block from each set can reside at two row addresses. This provides more flexibility in the overall cache specifications but adds a small amount to the on-chip controls. Note that the number of cycles to load the SBB could be different from that to unload the CRB to the array, but this has little advantage since the same decoder is used for both functions. The minimum number of bits per row on each chip for a generalized functional cache of the type in Figure 11 can be specified as follows. In Figure 9, let

b_r = number of bits per row on each chip,
α = number of cycles to load SBB/unload CRB to the array = number of rows per block,
b_{bl} = total number of bits per cache block,
N_c = total number of chips in cache,
S = set associativity.

For this general case, the *minimum* number of bits per row on each chip is thus given by

$$b_r \geq \frac{S \times b_{bl}}{\alpha \times N_c}. \tag{3}$$

For the ideal case, a value of $\alpha = 1$ is preferred if possible. For such a case, and with a four-way set associativity, 64 bytes (8×64 bits) per cache block, and four chips total, Equation (3) specifies a minimum of 512 bits per row. If we allow α to be two cycles, which maps to two rows per block as given by the mapping of Figure 9(b), or alternatively reduce the cache block to 32 bytes, a minimum of 256 bits

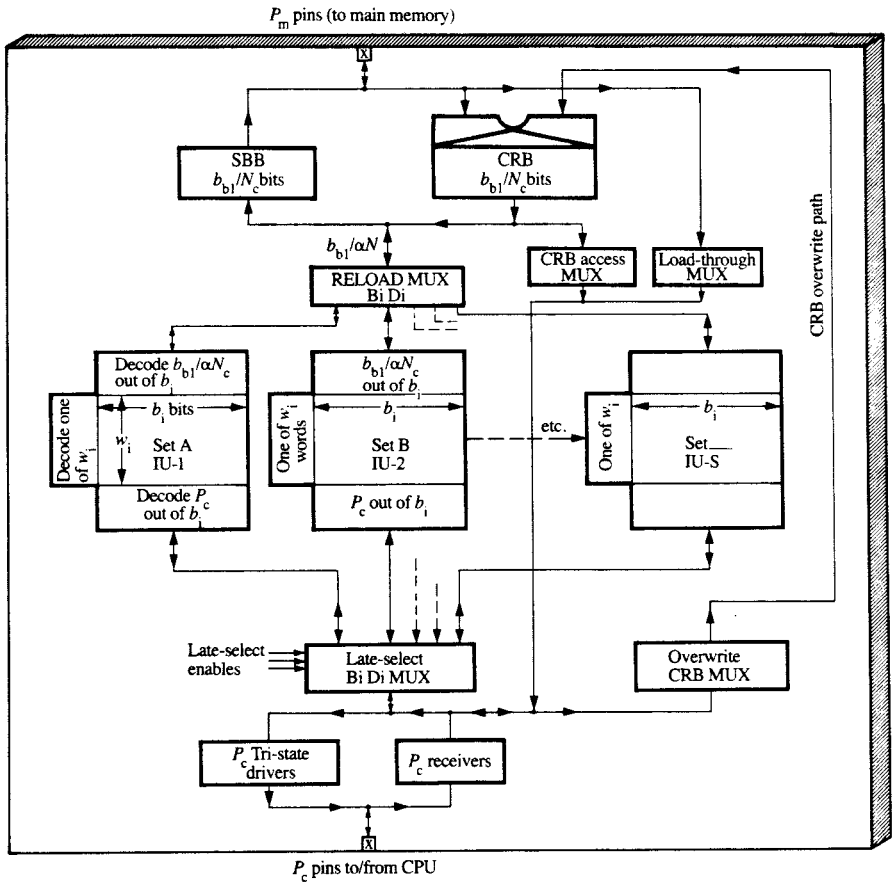

Figure 12

General chip structure and partitioning for an island-type design showing path widths at each interface.

per row are required. Obviously, a choice of both $\alpha = 2$ and a 32-byte block reduces the minimum number of bits per row to 128. In this manner, the chip designer is given more flexibility in choosing the more optimal array configuration for speed and layout, which is important.

• *Cache chip implementation*

Although the above functions can be implemented with macros on a relatively standard array design, the highest possible performance will require a custom design. Such a custom integrated approach will have many trade-offs in terms of circuit design, timing, and partitioning, which are highly technology-dependent. However, the logical structure of the chip will be basically the same, although the actual values of individual parameters will change. A very general logical partitioning in terms of islands which incorporates all the above functions is shown in **Figure 12** for a single chip.

Each set, or portion thereof, is mapped to one or more islands of this chip, and one or more chips as needed to form a chip unit. Assuming the cache capacity in bytes, C, set associativity, S, the technology in terms of bytes per chip, B_c, and hence number of chips in the cache, N_c, the length of the logical word, L_{lw}, the number of pins (bits) per chip for the CPU data bus, P_c, and the number of pins (bits) per chip for the main memory data bus P_m, have been specified, various relations among the remaining parameters for logical partitioning and reload path are as follows.

As defined previously, each chip unit is organized to supply one logical word to the CPU. The total number of chip units for each independent bank is

$$N_{cu} = N_c\left[\frac{P_c}{L_{lw}}\right] = \frac{C}{B_c} \times \frac{P_c}{L_{lw}}. \quad (4)$$

The average number of words reloaded per reload cycle,

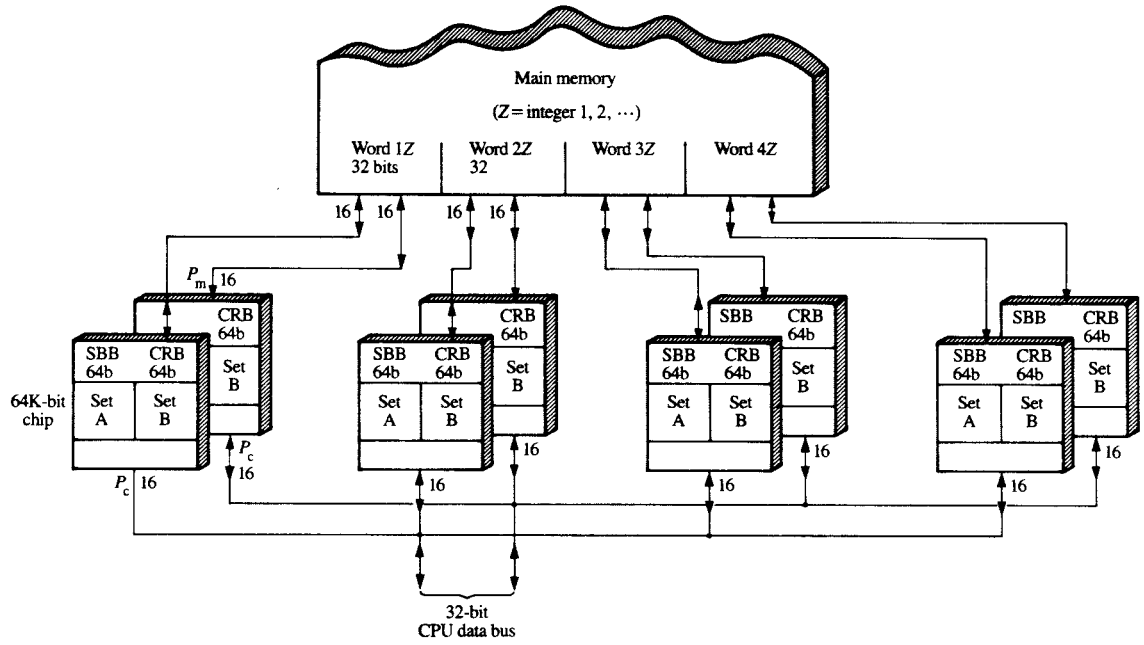

Figure 13

Partitioning of a 64K-byte cache using 64K-bit chips in a two-way set-associative, late-select design, with a four-word reload path: 64-byte blocks and SBB/CRB distributed across all eight chips.

assuming all P_m ports are used, is

$$W_{rl} = N_c \times \frac{P_m}{L_{lw}} \text{ in words per cycle.} \quad (5)$$

For a chip which is configured from independent islands such as that shown in Figure 12, the islands must be grouped into island units, IU, to match the given chip pin constraint, P_c, to the CPU and the set associativity, S. Ideally, there should be one island unit per set on each chip, and this will specify the amount of decoding needed per island and island unit. Note that an island unit is defined per chip, so for a partitioning which requires more than one chip per chip unit (i.e., per logical word) as in Figure 10, a logical word can be spread over multiple islands and multiple chips. In such a case, the bottom decoder on each island unit in Figure 12 which services the CPU must decode P_c out of b_i bits per logical island unit.

The island decoding on the top of the arrays in Figure 12 must take into account that the interface is to the SBB/CRB, which has a wider bus width than the individual chip bus width, P_m, to main memory. Ideally, each SBB/CRB will be spread over all N_c chips, so each chip will contain only b_{bl}/N_c bits of the cache block. If the SBB/CRB are loaded/unloaded to the array in α cycles, where $\alpha \geq 1$, the top decoder on each island unit must decode $b_{bl}/(\alpha \times N_c)$ out of b_i bits. Note that no top decoding is necessary per island unit if b_i exactly equals $b_{bl}/(\alpha \times N_c)$. Ideally, it is desirable to have $\alpha = 1$. However, if the maximum allowable island unit array bit width, b_i, is for instance one half of b_{bl}/N_c, using a value of $\alpha = 2$ will exactly match the two requirements.

Cache system: Partitioning example

An example of how a cache might be partitioned onto the preferred custom layout of Figure 11 is given below. It is assumed that a 64K-byte, two-way set-associative, late-select cache organization using 64 bytes per block (replaceable unit) is implemented with static, functional cache chips having 64K bits per chip, with a 16-bit primary data path to the CPU, a 16-bit path to the main memory for reload, and a 32-bit logical word to the CPU. Thus, the required parameters are $C = 64$K bytes, $b_{bl} = 512$ bits, $S = 2$, $B_c = 8$K bytes per chip, $P_c = 16$ pins, $P_m = 16$ pins, and $N_c = 8$ chips total. Substituting these parameters into the above equations, we find that the chip unit consists of two such chips, which gives a total of N_{cu} = four chip units for this bank of 64K bytes. Since each chip has a 16-pin interface for reloading, the entire cache can have a four-word interface to main memory for reload, as shown in **Figure 13**. Since the

SBB/CRB are distributed over all chips, each chip will only contain 64 bits (8 bytes) of each 64-byte block. For optimum performance, the array configuration on each chip should have a bit width b_i of at least 128 bits to match the 64-bit SBB/CRB in a two-way set-associative design. A four-way set-associative design under the same conditions would require 256 bits. The remaining overall structure is logically equivalent to those of Figures 11 and 12, with appropriate partitioning to match the given chip parameters.

Conclusions

The above discussions have shown that a cache system operates in two distinct modes—normal access and reload. During normal access, a typical high-performance processor can require, on average, between one- and two-word access per processor cycle at a continuous rate. Whenever an access miss occurs, the system bandwidth requirement becomes a single burst of multiple words (equal to the block size) loaded on one main memory cycle and occurring infrequently, at random intervals. Unfortunately, the latter requirement is very difficult and costly to implement, resulting in performance degradation in actual systems. This has led to various cache organizations in an attempt to reduce the reload penalty. Unfortunately, ordinary static RAM chips are optimized for normal access and introduce considerable complexity in the reload path. This complexity takes several forms due to the various functions which must be performed for reload, in combination with the cache organization. Various pieces of this complexity can be removed by identification of the proper function and its implementation. By judicious organization and implementation of the cache chip, most of the reload process can be made to appear as if occurring on one memory cycle, while still allowing a set-associative, late-select cache organization. The internal array need be only a one-port cell design, and the additional functions are easily integrated with the proper choice of mapping of the cache blocks to the physical array. The cache chip specified in this paper incorporates all of these features and achieves an optimal trade-off between the array/chip design and the functional requirements of the overall system.

Appendix A: Nomenclature

- α = number of cycles to load SBB/unload CRB to the array = number of rows per block in Figure 9
- b_i = island unit array bit-width in bits per island unit
- b_r = number of bits per row address on each chip
- b_{bl} = total number of bits per cache block
- B_{bl} = number of bytes per cache block
- B_c = number of bytes per chip
- B_w = total number of bits per row on a chip unit or $n_c b_r$
- C = capacity of each independent cache bank in bytes
- L_{lw} = length of logical word in bits per word (32 bits in our examples)
- I_c = number of islands per chip
- IU = number of island units per chip
- N_c = total number of chips in cache over which a block is distributed, i.e., the smallest independent unit of a total cache
- n_c = total number of chips in each chip unit
- N_{cu} = total number of chip units in cache
- P_c = number of data pins on each chip for CPU bus
- P_m = number of pins on each chip for reload bus from main memory
- S = set associativity
- n_{rl} = number of words reloaded to each chip unit per cycle
- W_{rl} = total reload path to cache in words per cycle
- $W_{bl} = b_{bl}/L_{lw}$ = total number of logical words per cache block

Acknowledgments

The author wishes to thank F. T. Tong and S. Chuang for their support and helpful discussions.

References

1. S. Schuster, B. Chappell, R. Franch, P. Greier, S. Klepner, F.-S. Lai, P. Cook, R. Lipa, R. Perry, W. Pokorny, and M. Roberge, "A 15-ns CMOS 64K RAM," *IEEE J. Solid State Circuits* **SC-21**, No. 5, 704–711 (1986).
2. R. E. Matick, *Computer Storage Systems and Technology*, John Wiley & Sons, Inc., New York, 1977.
3. D. Fier, R. Caulk, P. Torgerson, D. Breid, R. Bradley, and K. LeClair, "A 36/72b CMOS Micro-Mainframe Chip Set," *Digest of Technical Papers*, IEEE International Solid State Circuits Conference, February 1986, p. 26.
4. D. Alpert, D. Carberry, M. Yamamura, Y. Chow, and P. Mak, "32 Bit Processor Chip Integrates Major System Functions," *ELECTRONICS*, pp. 113–119 (July 14, 1983).
5. T. Watanabe, "An 8K Byte Intelligent Cache Memory," *Digest of Technical Papers*, IEEE International Solid State Circuits Conference, February 1987, p. 266.
6. C. Alsing, K. Holberger, C. Holland, E. Rasala, and S. Wallach, "Minicomputer Fills Mainframe's Shoes," *ELECTRONICS*, pp. 130–137 (May 22, 1980).
7. R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Syst. J.* **9**, No. 2, 78–117 (1970).
8. R. E. Matick and D. T. Ling, "Architecture Implications in the Design of Microprocessors," *IBM Syst. J.* **23**, No. 3, 264–280 (1984); (Figure 6).
9. H. B. Mann, *Analysis and Design of Experiments*, Dover Publications, New York, 1949.
10. M. Y. Hsiao, D. Bossen, and R. T. Chien, "Orthogonal Latin Square Codes," *IBM J. Res. Develop.* **14**, No. 4, 390–394 (1970).

Received July 7, 1988; accepted for publication October 3, 1988

Richard E. Matick *IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.*

Dr. Matick received his B.S., M.S., and Ph.D. degrees in electrical engineering from Carnegie Mellon University, Pittsburgh, in 1955,

1956, and 1958. He joined IBM in October 1958 and worked in the areas of thin magnetic films, memories, and ferroelectrics. As manager of the Magnetic Film Memory Group from 1962 to 1964, he received an Outstanding Invention Award for the invention and development of the thick-film read-only memory. He spent six months at IBM Hursley, England, developing this read-only memory for System/360 applications. Dr. Matick joined the technical staff of the IBM Director of Research in 1965 and remained until 1972, serving in various staff positions that included coordinator of Research Division plans and Technical Assistant to the Director of Research. He took a sabbatical in 1972 to teach at the University of Colorado and at IBM Boulder. During the summer of 1973 he taught at Stanford University while doing research there. He is currently working in the areas of VLSI functional memory chip and microprocessor design. In 1986 Dr. Matick received an IBM Outstanding Innovation Award as co-inventor of "video RAM," a new memory chip that is becoming popular for use in bit-buffered displays and is also used in the high-resolution display announced with the IBM PC RT. Dr. Matick is the author of the books *Transmission Lines for Digital and Communication Networks* and *Computer Storage Systems and Technology*. He is also the author of chapters on memories in *Introduction to Computer Architecture* and *Electronics Engineers' Handbook, Second Edition*. Dr. Matick has written numerous papers on magnetic devices and memories, semiconductor circuits, memory and logic, as well as virtual memory chips and systems. He is the holder of numerous patents and patent publications, and is a member of Eta Kappa Nu and a senior member of the Institute of Electrical and Electronics Engineers.