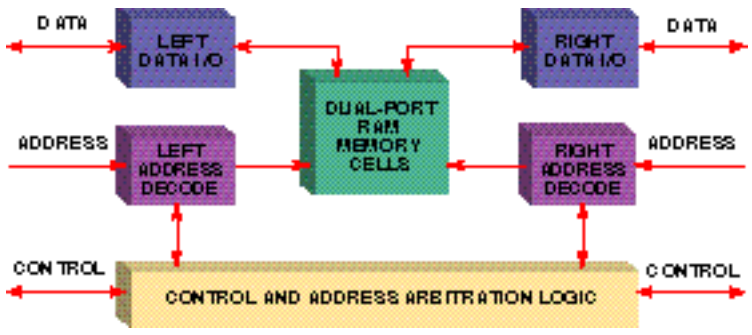
Understanding Specialty Memories, Part 1: Dual-Port RAMs

A dual-port RAM—generally classified with FIFOs as a "specialty" memory—is a random-access memory that can be accessed simultaneously by two independent entities. In digital ICs, this implies a dual-port memory cell that can be accessed at the same time by two independent sets of address, data, and control lines.

The first applications for dual-port memories were for CPU register files. Dual-port RAMs can also serve as data or instruction cache memories. But the largest usage of dual-port RAMs is in communications, which includes the exchange of data between processors, processes, and systems.



Fundamentals. Adual-port RAM uses three types of signals–data, address, and control-and uses two sets of them, one for the left port and one for the right port. The first dual-ports to use a true dual-port RAM cell were introduced in 1983; early "dual-port" RAMs were created from single-port RAMs by multiplexing the RAM between the two entities that shared it.

Background

Communication between systems does not require physical dual-port RAMs. Instead, you can partition a conventional RAM memory into virtual data-storage areas (buffers), usually to store at least two data packets. The buffers are shared between the communications controller and the intelligent element (usually a microprocessor) that assembles the packets and stores them.

The communications controller can also be a microprocessor, which reads the data from memory and converts it from parallel to serial form; encodes the data and converts it to analog form; and sends the data out over the communications channel on the transmit side. Note that in a single-processor system, data buffers are not shared and the system needs neither a virtual nor a physical dual-port RAM.

Control information associated with each data buffer tells the communications controller the number of words in the buffer and the starting address of the data in the buffer. The control information resides in one or more memory locations; in a two-processor system, the addresses have been previously agreed upon by both processors. This simple software-based buffer example requires a second level of control, however—a way to prevent the two microprocessors from getting in each other's way. In other words, the system needs a procedure-control mechanism.

Analyzing the procedure-control requirement another way introduces the concept of data ownership. Assume processor *A* assembles and stores messages and thus "owns" the data while performing these tasks. Likewise, communications processor *B* owns the data while performing its tasks. The procedure-control mechanism amounts to a technique for transferring data-ownership between processors *A* and *B*.

In large systems, where many processors perform many different operations, the processing of information is divided into many tasks that can be done by the different processors. The tasks can either be scheduled and assigned by a processor dedicated to the task or be performed by any available processor. These alternatives are referred to as *autocratic* and *egalitarian* systems, respectively. The term egalitarian implies that the processors are treated equally. In either case, the processors must have access to a shared-memory location used for message passing.

Message passing. Our two-processor system can achieve synchronization—essential to concurrent programming—by using a *lockword* or *lockvariable*. The lockvariable is a location in shared memory that is operated upon using two synchronization primitives—simple binary switch operations. If a processor wishes to lock or own a critical section of code or data, the processor indivisibly sets the lockvariable if testing shows the lockvariable to be zero; if the lockvariable is not zero, the operation is repeated until the lockvariable is zero. To unlock the critical section, a processor sets the lockvariable to zero and continues.

Although most modern processors have indivisible read/modify/write instructions, also called *test-and-set (TAS)* instructions, lockvariables can be implemented without using such instructions. Also, lockvariables surround or bracket *semaphores*—a technique for managing a queue of tasks waiting for a resource—and thus provide entry and exit control on a mutual-exclusion basis.

About TAS instructions. The two-processor example assumes that each processor has a TAS instruction. Such instructions typically operate as follows: read, test, set to X. The addressed memory location is read, and if its contents are zero, the value X is written into that location. If the contents are not zero, they are returned to the processor and the value in the memory location is not disturbed.

The usual convention is that a value of zero in the lockvariable means that the resource associated with it is available. A non-zero value means that another processor temporarily owns the resource and that the resource is not available. After performing the task associated with the lockvariable, the processor sets the lockvariable's value to zero. The system initializes with all lockvariables set to zero.

In our example, processor *A* performs a TAS operation on the lockvariable and, finding the lockvariable to be zero, sets the lockvariable to a one. This tells processor *B* that the message is being assembled in the memory buffer area and not ready to be transmitted. Processor *A* then assembles the message, after which it clears the lockvariable, sends a message to processor *B* saying that the message is ready to be transmitted, and gives the data's location and the number of bytes to be sent. Processor *B* reads the message from processor *A* and performs a TAS operation on the lockvariable; finding the lockvariable to be zero, processor *B* sets it to a two. This tells processor *A* that the message is being sent. Processor *B* then transmits the message and clears the lockvariable, and sends processor *A* a message that the transmission has been completed. After receiving the message from processor *B*, processor *A* performs a TAS operation on the lockvariable; finding the lockvariable to be zero, processor *A* concludes that the message has been successfully transmitted.

Note that this procedure does not require the use of a dual-port RAM. The procedure does require each processor to perform a TAS instruction, clear the lockvariable, and send a message to the other processor. Sending a message implies writing to a location in shared memory. To know that a message is waiting, the processor receiving the message must either read the memory location periodically (referred to as polling a mailbox), or the act of writing to the mailbox must generate an interrupt to the receiving processor. The interrupt-driven alternative is usually preferred, because the receiving processor does not have to waste time in a polling sequence.

Masters and slaves

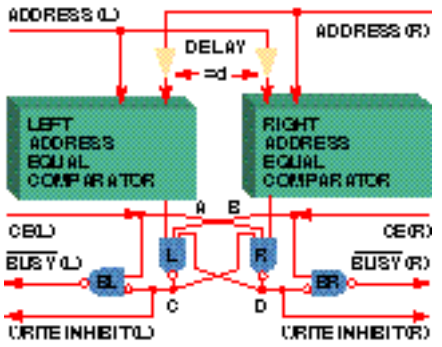
The first dual-port RAMs to use a dual-port RAM cell were introduced in 1983. They had two mailboxes for message passing. When written to from one port, a mailbox generates an interrupt to the opposite port. Additionally, on-chip arbitration logic generates a BUSY signal to the loser when both left and right ports address the same memory location. If the loser was attempting to write, the write is suppressed.

Most of the dual-port RAMs on the market today are functionally equivalent to the original products. The new features added to some dual-port RAMs from several manufacturers, including Cypress, include dedicated semaphore registers. Hardware semaphores provide efficient means of allocating exclusive priority accesses to blocks of shared memory locations in dual-port RAMs.

Slave dual-ports, as companions to master dual-ports, appeared in 1985. A slave device provides word-width expansion. Further, a slave contains no arbitration logic, and one master can drive many slaves.

A lose-lose situation

The arrangement between master and slave dual-ports just described avoids the so-called "deadly embrace" problem. The deadly embrace can occur when two masters are connected in parallel to make a wider word. If the left and right port addresses match, and the left and right port chip-enables then become active to both chips at about the same time, it is possible to have one port of



Arbitration logic. Cypress dual-port RAM masters incorporate arbitration logic that decides (should the addresses be equal simultaneously) which port wins and which loses, prevents the losing port from writing, and provides a BUSY signal to the losing port. The logic consists of left and right address-equality comparators with their associated delay buffers; the arbitration latch formed by the cross-coupled, three-input NAND gates (labeled Land R); and the gates that generate the BUSY signals.

one master lose and the opposite port of the other master also lose.

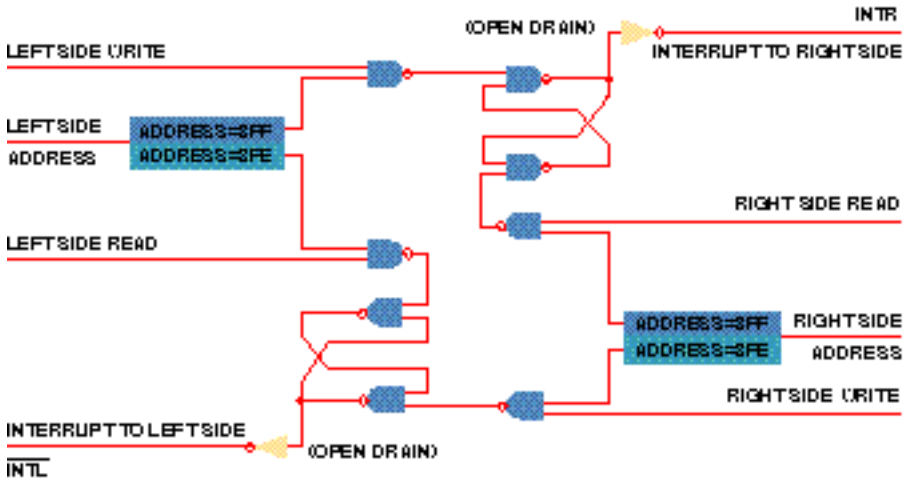
In other words, if an address match occurs and both ports are enabled during a brief time window or an aperture of uncertainty, the dual-port RAM cannot determine which port wins or loses. Under these conditions, if the corresponding left and right port BUSY pins are connected together, both ports of both masters are active (*low*). This condition occurs because the BUSY outputs are open-drain and the loser pulls the node *low*, and it is the simplest example of the deadly embrace. To the outside world, both ports are busy. The system remains locked up indefinitely—each port waits to be released by the other, and each master's arbiter section thinks it has lost the arbitration and is waiting to be released by the other.

In general, the deadly embrace occurs under two conditions: a processor requires one or more resources to perform a task; or, one or more of the required resources is temporarily owned by another processor, which requires one or more of the same resources to perform its task. The concept of the deadly embrace extends to *n* processors and *m* resources.

The solution to the deadly embrace problem in general is very complex and depends upon whether the system is autocratic or egalitarian, the tasks' priorities, and more. For dual-port RAMs in particular, however, the solution is simple: do not cascade two masters in width—instead, simply use a master and a slave.

Cypress dual-port operation

Dual-port RAMs use three types of signals—address, data, and control—and use two sets of them, one for the left port and another for the right port. Address signals are unidirectional into the dual-port; their states specify the memory location to be



Interrupt logic. Aport's chip enable must be asserted for the port to read from or write to any location, including the mailboxes. For the two-processor example in the text, the highest memory location is the mailbox for the right-side processor. When the left-side processor writes to this mailbox, the interrupt request to the right processor (INTR) goes low. When the right processor reads its mailbox, the flip-flop is reset and INTR goes high. The second-highest memory location is the mailbox for the left-side processor. When the right processor writes to this mailbox, the interrupt request to the left processor (INTR) goes low. When the left processor reads its mailbox, the flip-flop is reset and INTL goes high. For more detail, see the text. (Chip enables not shown.)