Understanding Asynchronous Dual-Port RAMs

This application note examines the evolution of multi-port memories and explains the operation and benefits of Cypress's asynchronous dual-port RAMs. It also explores the benefits of using dual-port RAMs over single-port RAMs in multiprocessor systems.

A dual-port RAM 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 using two independent sets of address, data, and control lines.

Dual-Port Memory Using Single-Port RAM

Before the dual-port memory cell existed, designers created dual-port RAMs from single-port RAMs by adding a multiplexer between the RAM and the two entities that shared the RAM. *Figure 1* illustrates a block diagram of such an arrangement. Two processors, MP1 and MP2, share the RAM. If each processor has access to the RAM half the time, the resource is shared equally and is said to be allocated according to a fairness doctrine.

This time division multiplexing assures that there is no contention for the RAM. However, performance suffers if the RAM's access time does not equal 1/2 or less of the processors' clock period, assuming that the processors are clocked from the same source.

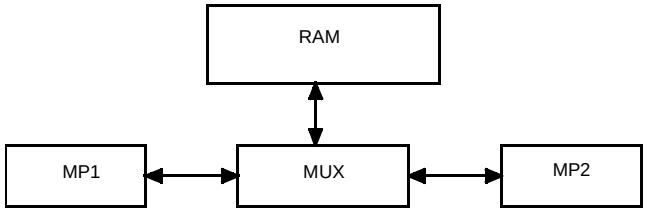

Figure 1. Dual-Port Memory Using Single-Port RAM

This approach requires either single-port memories with access times equal to one half of a bus cycle time, or requires that each processor access the RAM on every other cycle. Often these requirements are limiting due to speeds of available RAMs and difficulties with interleaving transactions between processors. Note also that this approach requires both processors to be running from the same clock.

By using dual-ported RAMs, the efficiency of memory accesses can essentially be doubled. Also since Dual-port RAMS do not have restrictions regarding accesses between the two ports, each processor can be operating at different clock rates. This inherent synchronization property proves to be one of the largest benefits of using dual-port RAMs.

Dual-Port RAM Applications

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

Virtual Dual-Port RAM

Communication between systems does not require physical dual-port RAMs. Instead, a conventional RAM memory is partitioned into virtual data-storage areas (buffers), usually to store at least two data packets. These buffers are shared between the communications controller and the intelligent element that assembles the packets and stores them (usually a microprocessor). The communications controller can also be a microprocessor. It reads the data from memory, converts the data from parallel to serial form, encodes the data, converts the data to analog form, and sends the data out over the communications channel on the transmit side. If the system contains only one processor, the 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 whose addresses have been previously agreed upon by the two processors.

This simple software-based buffer example requires a second level of control—a mechanism or procedure that prevents the two microprocessors from getting in each other's way. In other words, the system needs a procedure control mechanism.

Another way of analyzing this requirement introduces the concept of data ownership. Say, for example, that processor A assembles and stores messages and thus owns the data while performing these tasks. Likewise, the communications processor B owns the data while performing its tasks. The procedure control mechanism amounts to a technique for transferring data ownership between processor A and B.

In large systems, where many processors perform many different operations, the processing of the information is called a job or a procedure. The procedure is divided into many tasks, which can be performed by different processors. The tasks can either be scheduled and assigned by a processor dedicated to that 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.

Synchronizing sequential processes is the cornerstone of concurrent programming, which applies to multi-tasking, single-processor systems; distributed-processor networks; and tightly coupled multiprocessor systems.

Message Passing

In the two-processor system under consideration, synchronization can be achieved by using a lockword or lock variable. The lock variable can apply either to data (as in this example) or to executable instructions.

The lock variable is a location in shared memory that is operated upon using two synchronization primitives: LOCK (v) and UNLOCK (v), where (v) is the location operated upon. These are simple binary switch operations. If a processor wishes to lock or own a critical section of code or data, the processor indivisibly sets the lock variable if testing shows the lock variable to be zero. If the lock variable is not zero, then the operation is repeated until the lock variable is zero. To unlock the critical section, a processor sets the lock variable to zero and continues.

Most modern processors have indivisible read/modify/write instructions, also called test and set (TAS) instructions. In Reference 1, however, E. W. Dijkstra shows that lock variables can be implemented without using a read/modify/write instruction. And in Reference 2 he develops the semaphore, a technique for managing a queue of tasks waiting for a resource. Lock variables surround or bracket semaphores and thus provide entry and exit control on a mutual-exclusion basis.

Typical TAS Instruction

The current example assumes that the processors have a TAS instruction. A typical TAS instruction operates as follows: read, test, and 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, the contents 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 lock variable 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 lock variable, the processor sets the lock variable's value to zero. The system is initialized with all lock variables set to zero.

In the current example, processor A performs a TAS operation on the lock variable and, finding the lock variable to be zero, sets the lock variable to a one. This tells processor B that the message is in the process of being assembled in the memory buffer area and is not ready to be transmitted. Processor A then assembles the message. After the message is assembled, processor A clears the lock variable, 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 lock variable; finding the lock variable to be zero, processor B sets it to a two. This tells processor A that the message is in the process of being transmitted. Processor B then transmits the message and clears the lock variable. Processor B sends processor A a message that the transmission task has been completed. After receiving the message from processor B, processor A performs a TAS operation on the lock variable; finding the lock variable 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 lock variable, 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.

The Deadly Embrace

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 approximately the same time, it is possible to have one port of 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 small 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.

This condition is the simplest example of the deadly embrace. As far as the external world is concerned, both ports are busy, and the system remains locked up indefinitely, with each port waiting to be released by the other. 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, and 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.

For example, if processor A owns resource X and processor B owns resource Y, and both resources are required to accomplish the task, a stalemate occurs in which each processor waits for the other to relinquish the required resource. This is the simplest example. The concept extends to *n* processors and *m* resources.

The solution to the deadly embrace depends upon whether the system is autocratic or egalitarian, the tasks' priorities, etc., and is beyond the scope of this discussion. In the case of dual-port RAMs, however, the solution is simple: Do not cascade two masters in width; use a master and a slave.

Cypress Dual-Port RAM Operation

A simplified block diagram of the Cypress dual-port RAM appears in *Figure 2*. The device interface includes three types of signals: address, data, and control. There are two sets of these signals: those of the left port and those of the right port. Each signal has either the subscript L or R to designate left or right, respectively.

The address pins are designated A0 through A9 (1024 x 8) and A0 through A10 (2048 x 8), where A0 is the least significant bit (LSB) and A9 or A10 is the most significant bit (MSB). The address pins are unidirectional inputs to the device; their states specify the memory location to be read from or written into.

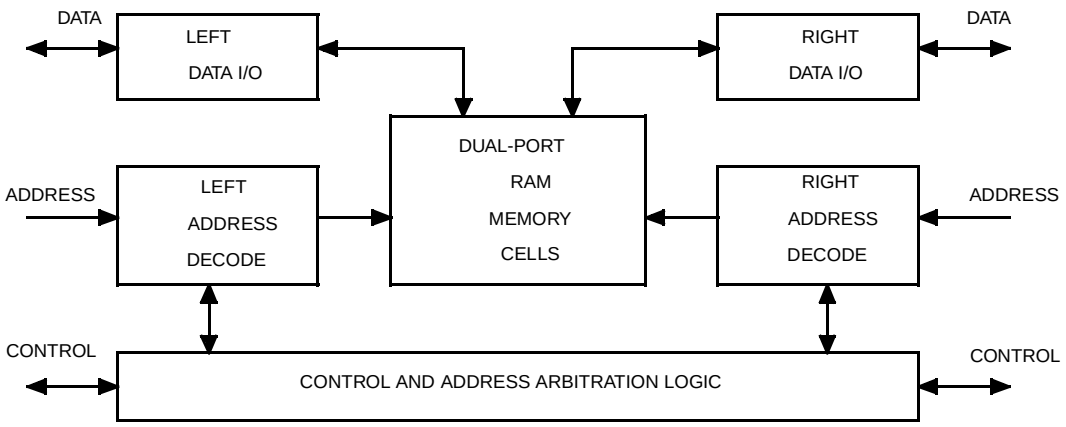

Figure 2. Dual-Port RAM Block Diagram

The data pins are designated I/O0 through I/O7, where I/O0 is the LSB and I/O7 is the MSB. The data pins are bidirectional; their states represent either the data to be written or the data to be read.

The control pins are chip enable ($\overline{CE}$), read/write (R/$\overline{W}$), and output enable ($\overline{OE}$). A semaphore enable control pin ($\overline{SEM}$) is included on dual-port RAMs with semaphores. Two flags are also provided, $\overline{INT}$ and $\overline{BUSY}$; both have open-drain outputs and require external pull-up resistors. A LOW on the chip enable input allows that port to become functional. Data is either read from the internal dual-port RAM array or written into it, depending upon the state of the read/write signal; a LOW initiates a write operation. The three-state data output drivers are enabled by a LOW output enable.

When one port writes to a pre-determined mailbox, an interrupt to the other port is generated. When the interrupted port reads that memory location, the interrupt is reset.

When both ports address the same memory location and both chip enables are active (LOW), contention occurs for that address. An arbitration is then performed, and ownership of the memory location is assigned to the winner. An active (LOW) busy signal notifies the loser of the arbitration.

Dual-Port RAM Functional Description

An important aspect of the Cypress dual-port RAMs is their interrupt logic. A simplified logic diagram of this logic appears in *Figure 3*, with the chip enables deleted. A port's chip enable must be asserted for the port to either read from or write to any location, including the mailboxes. Note that you can use the mailbox locations as conventional memory by not connecting the interrupt line to the appropriate processor.

The upper two memory locations (7FF and 7FE for 2K x 8; 3FF and 3FE for 1K x 8) can be used for message passing. The highest memory location serves as the mailbox for the right processor. When the left processor writes to this mailbox, the interrupt (request) to the right processor, $\overline{INTR}$, goes LOW. When the right processor reads its mailbox, the flip-flop is reset, and $\overline{INTR}$ goes HIGH.

The second highest memory location serves as the mailbox for the left processor. When the right processor writes to this mailbox, the interrupt (request) to the left processor, $\overline{INTL}$, goes LOW. When the left processor reads its mailbox, the flip-flop is reset, and $\overline{INTL}$ goes HIGH.

Note that each port can read the other port's mailbox without resetting the associated flip-flop. If your application does not require message passing, leave the appropriate pin open. Do not connect a pull-up resistor to the pin, and do not connect the pin to the processor's interrupt request pin.

Note that the active state of the busy signal prevents a port from setting the interrupt to the winning port. Additionally, an active busy signal to a port prevents that port from reading its own mailbox and thus resetting the interrupt. These operations are ramifications of the data-ownership concept.

If both ports address the same memory location at the same time, the master performs an arbitration, so that one port wins and the other loses. Because each of the two ports can be in either the reading or writing state, there are four possible combinations of ports and states (*Table 1*).

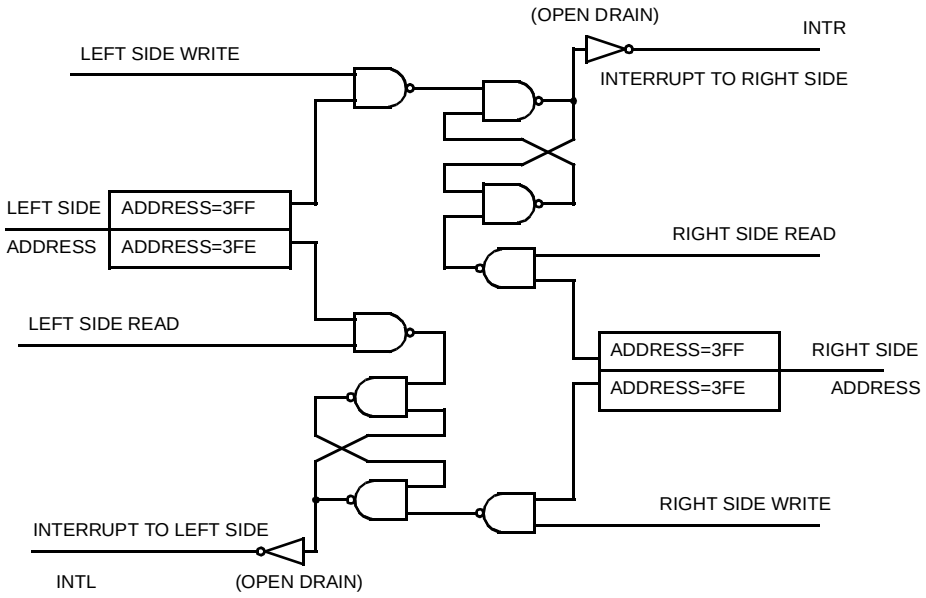

Figure 3. Interrupt Logic

Table 1. Functional Operation of Dual-Port Masters

Operation			Result of Operation after Arbitration
Case	Left Port	Right Port	(Master)
1	Read	Read	Both ports read.
2	Read	Write	Loser is prevented from writing. If loser is reading and ports are asynchronous, data read might not be valid.
3	Write	Read	
4	Write	Write	

Both Ports Reading

If both ports of a dual-port IC read the same location at the same time, you can assume that both ports read the same data. When arbitration occurs as a result of contention in a Cypress dual-port RAM, the port that wins the arbitration gets temporary ownership of the memory location. The losing port can read the memory location but the busy signal tells it that it lost the arbitration.

To guarantee data integrity in a multiprocessor system, it is standard practice to apply the concept of data ownership. This ownership can apply to executable code, data, or control locations in memory. The control locations in memory can be associated with a resource, such as a printer, tape drive, disk drive, or communications port.

One Port Reading, the Other Writing

The result of arbitration will allocate priority to either the reading or the writing port. In Cypress dual-port RAMs, if the losing port is attempting to write data, the write is inhibited so that the data in memory is not corrupted. The $\overline{\text{BUSY}}$ flag to the losing port signals that the write was not performed.

If the losing port is attempting to read data, it is possible for the data to be old data, new data, or some random combination of the two. The $\overline{\text{BUSY}}$ flag to the losing port signals that the old data is still being read on the losing port's data lines. The old data will remain undisturbed for an access time after either $\overline{\text{BUSY}}$ on the losing port goes HIGH, the losing port's address is toggled, $\overline{\text{CE}}$ for the losing port is toggled, or R/$\overline{\text{W}}$ for the losing port is toggled during a valid read.

If the new data is needed, the $\overline{\text{BUSY}}$ flag can be used to generate a delay until the new data is present or can signal a processor to attempt the read again after $\overline{\text{BUSY}}$ is cleared.

Both Ports Writing

The losing port is prevented from writing so that the data cannot be corrupted. $\overline{\text{BUSY}}$ is asserted to the losing port, indicating that the write operation was unsuccessful.

Arbitration Logic

Figure 4 shows the arbitration logic used in Cypress dual-port RAM masters. The arbitration logic has three functions: to decide which port wins and which loses if the addresses are equal simultaneously, to prevent the losing port from writing, and to provide a busy signal to the losing port.

The arbitration logic consists of left and right address equality comparators with their associated delay buffers; the arbitra-

tion latch formed by the cross-coupled, three-input NAND gates labeled L and R; and the gates that generate the busy signals.

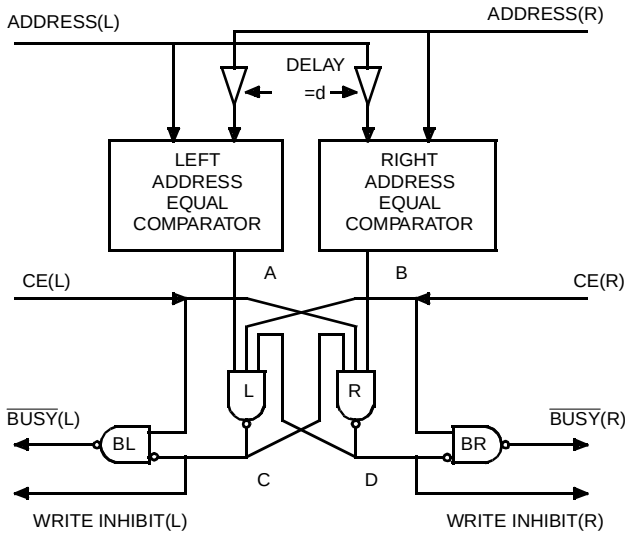

Figure 4. Arbitration Logic

Operation With Unequal Addresses

When the addresses of the right and left ports are not equal, the outputs of the address comparators (nodes A and B) are both LOW, and the outputs of the gates labeled L and R (nodes C and D) are both HIGH. This condition forces both $\overline{\text{BUSY}}$ signals HIGH and both $\overline{\text{Write Inhibit}}$ signals HIGH. The arbitration latch does not function as a latch.

Left Port Camped on an Address

Next, consider the condition where the left-port address and chip enable are quiescent, and the right-port address changes to an address equal to that of the left port. Nodes A and B are initially LOW.

Because the right-port address does not go through the delay buffer, the output of the right-address comparator (node B) goes HIGH before node A goes HIGH by a delay interval, d. The delay must be greater than the delay through the R gate, so that when node B goes HIGH, node D goes LOW, causing node C to remain HIGH. CE(R) and CE(L) are both HIGH; they are the inverse of the chip enable inputs. Node D going LOW causes the output of the BR gate to go LOW, which tells the right port that the memory location it just addressed belongs to the left port. A write inhibit signal is also generated that prevents the right port from writing into the addressed memory location.

In summary, when the right port addresses a memory location that is already being addressed by the left port, a delay occurs that equals the sum of the propagation delays of the right-address comparator, the R gate, the BR gate, and the output driver (not shown in the diagram). Then the busy signal to the right port is asserted. Nodes A, B, and C are now HIGH, and node D is LOW. $\overline{\text{BUSY}}$ is asserted to the right port.

Due to the symmetry of the arbitration logic, the device operates the same when either the right or left ports are camped on an address.

Right and Left Addresses Equal Simultaneously

In the general case, it is possible to have both ports access the same memory location simultaneously, unless this is guaranteed not to occur by the design of the system. When nodes A and B go from LOW to HIGH at exactly the same instant, the arbitration latch settles into one of two states and determines which port wins and which port loses. The latch is designed such that its two outputs are never LOW at the same time. It also has a very fast switching time.

The dual-port RAM imposes a minimum time difference between either of two events: the two chip enables going from inactive to active and the two sets of addresses going from mismatch to equal. If the events are close together in time, the probability of each port either winning or losing the arbitration is approximately equal. This parameter is called port set-up time for priority and is abbreviated as t_{PS} on the datasheets. The specified value is 5 ns. (Note, though, that Cypress product engineers have measured t_{PS} at room temperature and nominal V_{CC} (5V) and found a value of approximately 200 ps.) In other words, if one port addresses a memory location 5 ns before the other port, the first port is guaranteed to win. If not, the result of the subsequent arbitration is unpredictable.

Other Key $\overline{\text{BUSY}}$ Parameters

Several other key parameters are specified with respect to the busy signal. For example, $\overline{\text{BUSY}}$ LOW from address match, t_{BLA}, is the maximum time it takes busy to go LOW, as measured from the time the two port addresses are the same. This is the time from an address match until the losing port is notified that it has lost the arbitration. Obviously, the sooner this occurs the better. If the value of t_{BLA} is greater than the memory cycle time, another cycle must be added to detect the condition, which can severely reduce performance. This time is less than the minimum cycle time for all speed grades of all Cypress dual-port RAMs.

Another parameter, $\overline{\text{BUSY}}$ HIGH from address mismatch, t_{BHA}, is the maximum time it takes $\overline{\text{BUSY}}$ to go from LOW to HIGH, as measured from the time the two port addresses do not match until the $\overline{\text{BUSY}}$ signal goes HIGH. The comments of the preceding paragraph also apply here.

The next two parameters are similar to the preceding two. The difference is that the chip enable controls the busy signal. The parameters are $\overline{\text{BUSY}}$ LOW from $\overline{\text{CE}}$ LOW, t_{BLC}, and $\overline{\text{BUSY}}$ HIGH from $\overline{\text{CE}}$ HIGH, t_{BHC}. Both of these parameters are less than the minimum cycle time for all speed grades of all Cypress dual-port RAMs.

$\overline{\text{BUSY}}$ HIGH to valid data, t_{BDD}, is the maximum time it takes the data to become valid to the losing port after $\overline{\text{BUSY}}$ goes away. This parameter's value equals the address access time, t_{AA}, because a read cycle is initiated to the losing port when its $\overline{\text{BUSY}}$ signal transitions from LOW to HIGH. An action by either port can cause the busy transition. The winning port can either change its address or deassert its chip enable.

To illustrate the last two parameters, *Figure 5* shows the timing for the right port performing a write operation and the left port asynchronously moving to the same address and attempting to perform a read operation. The first parameter of interest is t_{DDD}, which is the maximum time between the stabilization of the data to be written by the winning port and that same data becoming valid at the outputs of the port that received the $\overline{\text{BUSY}}$. The second parameter of interest is t_{WDD},

which is the maximum time between the HIGH-to-LOW transition of the winning port's write strobe and the data becoming valid at the outputs of the port that received the $\overline{BUSY}$.

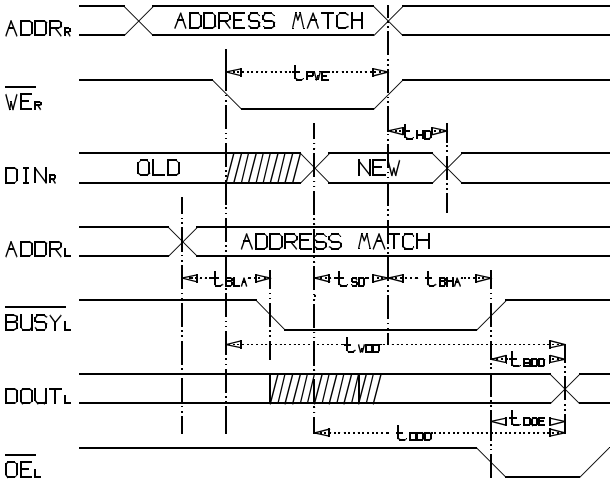

Figure 5. $\overline{BUSY}$ Timing

It is possible for the losing port to read either the old data, the new data, or some random combination of the two under these circumstances: the two ports are operating asynchronously (i.e., with independent clocks), and the conditions illustrated in *Figure 5* occur (winning port writing and losing port reading). If the read occurs early with respect to the write, old data is read. If the read occurs late with respect to the write, new data is read. And, if the read occurs at the same time the data is changing from old to new, the data read is not predictable. However, all is not lost. There are two general solutions. Both use the fact that the busy signal is asserted to the losing port, telling the port in this instance that the data it is reading might not be valid.

One solution is to use the HIGH-to-LOW transition of the busy signal to the losing port to generate an interrupt to the processor (or state machine) so that operation can be repeated. The drawback of this technique is that a snapshot of the states of the losing port's address lines and read/write line must be taken, so that the processor can tell what load/store operation caused the interrupt. Taking this snapshot requires latches or flip-flops for the data and control logic for doing the sampling, and the technique uses up an interrupt line. The processor must also be able to read the sampled data later.

A second solution is to use the LOW level of the $\overline{BUSY}$ signal to the losing port to prompt one of three types of delays: delay the reading of data until the data becomes valid, which occurs an access time after the LOW-to-HIGH transition of $\overline{BUSY}$; insert wait states until $\overline{BUSY}$ goes HIGH; or stretch the clock until $\overline{BUSY}$ goes HIGH. Any of these methods probably require less hardware and control logic than the preceding approach. Use of these methods does mean that the $\overline{BUSY}$ signal must eventually go from LOW to HIGH. This happens when the winning port either changes its address or deasserts its chip enable. For this reason, as well as for system noise immunity and power-saving considerations, it is recommended that blocks of addresses be decoded to generate chip enables for the dual-port RAMs.

Because the losing port has no control over the winning port in the general case, however, a question arises: What can the losing port do to successfully read the data just written, assuming the winning port does not change its address, write, or chip enable signals? There are two possible operations:

1. Change an address line to a different address, then change back to the original address. This toggles the $\overline{BUSY}$ signal to the losing port.
2. Change the state of the chip enable. This also toggles the $\overline{BUSY}$ signal to the losing port.

Hardware Semaphores

Cypress offers dual-port RAMs with eight on-chip hardware semaphore latches that are independent from RAM memory locations. Semaphore signaling is a popular method of allocating mutually exclusive accesses to blocks of memory that are shared among several processors. Exclusive processor control guarantees data integrity in sensitive applications such as shared I/O buffers. Semaphore signaling can also improve the efficiency of block memory accesses by preventing delays and processor stalls due to a memory location being busy from another processor access.

Traditional semaphore signaling has been implemented in software using dedicated memory locations to hold the semaphore signals. A processor could attempt to gain control of a semaphore by using an indivisible test and set instruction to test if the semaphore was set by another processor. If the semaphore is free, the processor sets the semaphore and gains exclusive control of a block of memory.

Cypress dual-port RAMs have on-chip hardware semaphores that are independent from RAM memory locations.

Hardware semaphores eliminate the need to use a processor with an indivisible test and set instruction. Semaphore control requests are handled using a standard write to the semaphore latch followed by a read instruction. There is no requirement to lockout other processor accesses to the semaphore between the write and read.

The hardware semaphores provide flexible software configuration of shared memory. The semaphores operate independent of any memory in the RAM allowing software to allocate block addresses and block sizes.

Cypress hardware semaphores implement a "token passing" scheme allowing the port in possession of the token to have exclusive access to a block of shared memory. Possession of the token can only be relinquished by the port with possession. A port's request for possession of the token will be denied if the token is held by the other port.

Possession of a token is indicated by the state of a semaphore latch formed from two cross-coupled NOR gates (see *Figure 6*). The latch can be set so that only one port controls the semaphore at a time. Additional input latches on the semaphore ports are used to hold requests to set or clear the latch. An output latch on each port is used to prevent the output from changing during a read from the port.

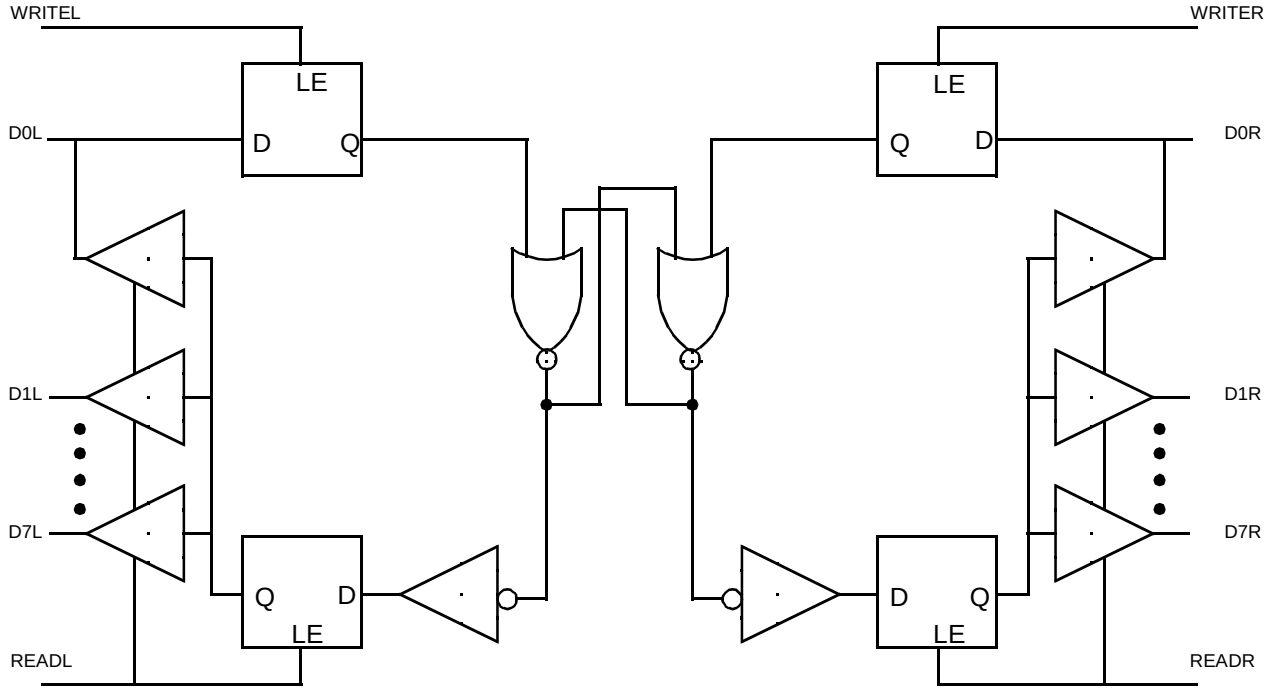

Figure 6. Semaphore Latch Cell

The semaphore latches are accessed through the data and address ports the same way as a RAM cell access. The semaphore enable line ($\overline{SEM}$ = LOW) initiates a semaphore access cycle. The A0–2 lines select which semaphore latch is accessed. Only the data on D_0 is latched into the semaphore during a write. The other data lines are ignored. During a read, the semaphore drives all the data lines (D0 through D7, D8) with the semaphore signal.

A processor requests control of a semaphore by writing a 0 to the D0 port of the semaphore addressed by A0–2. The 0 is latched into the port's input register and held until another write attempts to set it to 1. If the semaphore is free at the time of the request, the port will immediately be granted control of the semaphore. If the semaphore is controlled by the other port, the request for control will be denied. If control of the semaphore is relinquished by the other port while the 0 is still pending, then the requesting port will gain control of the semaphore. Control of the semaphore can only be relinquished by the controlling port by writing a 1 to the semaphore.

To see if a request for control of the semaphore was successful, a read of the semaphore is performed. A port controls the semaphore if 0 is read out on D0. The port does not control the semaphore if a 1 is read. The semaphore outputs drive all of the data lines with the state of the semaphore, so D0–7 will be "00000000" when control is granted and will be "11111111" when control is denied. The state of the internal semaphore latches may change during a read, but the output latch prevents the changes from propagating to the data lines. A new read cycle must be performed in order to update the port's output lines.

If both ports attempt to write a 0 within t_{SPS} of each other while the semaphore is free, semaphore arbitration logic will guarantee that only one side gains control of the semaphore.

Address Transition Detection

Why does changing the address or chip enable allow a losing port to read data successfully? All Cypress dual-port RAMs, both masters and slaves, use a circuit design technique called Address Transition Detection (ATD) to improve performance and reduce power dissipation.

ATD improves performance by equilibrating differential paths, pre-charging critical nodes, and forcing the outputs to a high-impedance state. Equilibration and pre-charging will bias critical nodes to voltage levels approximately in the mid-point of the small-signal operating range; when the data is sensed, it takes a shorter amount of time to transition to the 0 or 1 level. Forcing the outputs to their high-impedance states improves speed slightly, but more importantly, the technique reduces output switching noise by eliminating crowbar current and separating the output current into two pulses instead of one.

ATD minimizes power consumption because it turns on power-hungry circuits only when they are required. Slightly over 50 percent of a RAM's circuits are linear, and approximately 70 percent of the power is dissipated in the sense amplifiers during a read operation. When the RAM is operating at its maximum frequency, the ATD circuits are constantly triggered, so the power savings are minimal. At lower speeds or smaller duty cycles, however, the power savings are significant.

A diagram representing a typical ATD sequence is illustrated in *Figure 7*. The event that triggers the ATD sequence for either port is the transition of any address, chip-enable, or

read/write signal. Equilibration and pre-charging are performed next, followed by either turning on the sense amplifiers and latching the data (read operation) or pulling the BIT and $\overline{BIT}$ lines to the required levels (write operation) at the addressed location. The master clock pulse lasts from 7 to 11 ns, depending upon temperature, supply voltage, and the distribution of IC processing parameters. At the end of the pulse, the data is latched and the appropriate circuits are turned off.

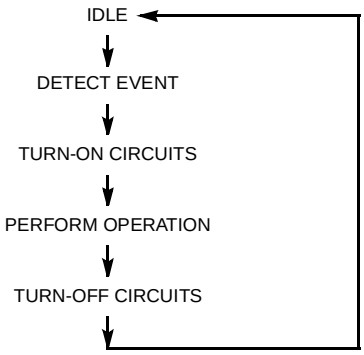

Figure 7. Simplified ATD Sequence

Master Standalone Operation

Figure 8 presents a block diagram of a system using two 8-bit microprocessors, the Cypress CY7C132 dual-port RAM, static RAM, and EPROM. The address lines of each microprocessor are decoded to generate the chip enables to the dual-port RAM, the SRAM, and the EPROM. Note that pull-up resistors are required on the interrupt requests to the microprocessors and the busy signals, which go to the microprocessors' wait inputs.

Slave Word-Width Expansion

The block diagram in *Figure 9* shows how to interconnect a CY7C132 (2K x 8) master and a CY7C142 (2K x 8) slave to form a 16-bit-wide word. The diagram does not show the interfaces to the processors or the connections for the interrupt signals. As previously explained, the interrupt outputs are not available at the 2K x 8 level in the 48-pin DIP due to pin limitations. In the LCC and PLCC packages, the interrupt outputs are available from both the master and the slave devices. You can use either one. You do not have to tie the corresponding interrupt pins of the master and the slave together.

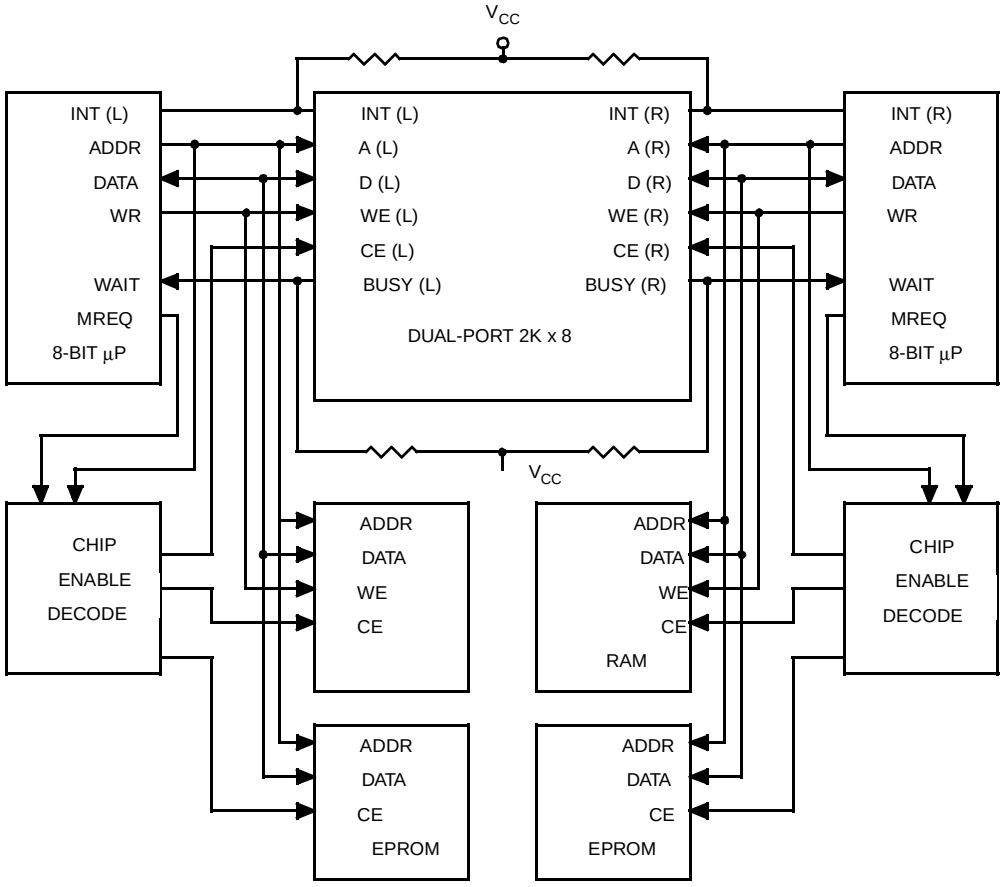

Figure 8. Typical 8-Bit Microprocessor

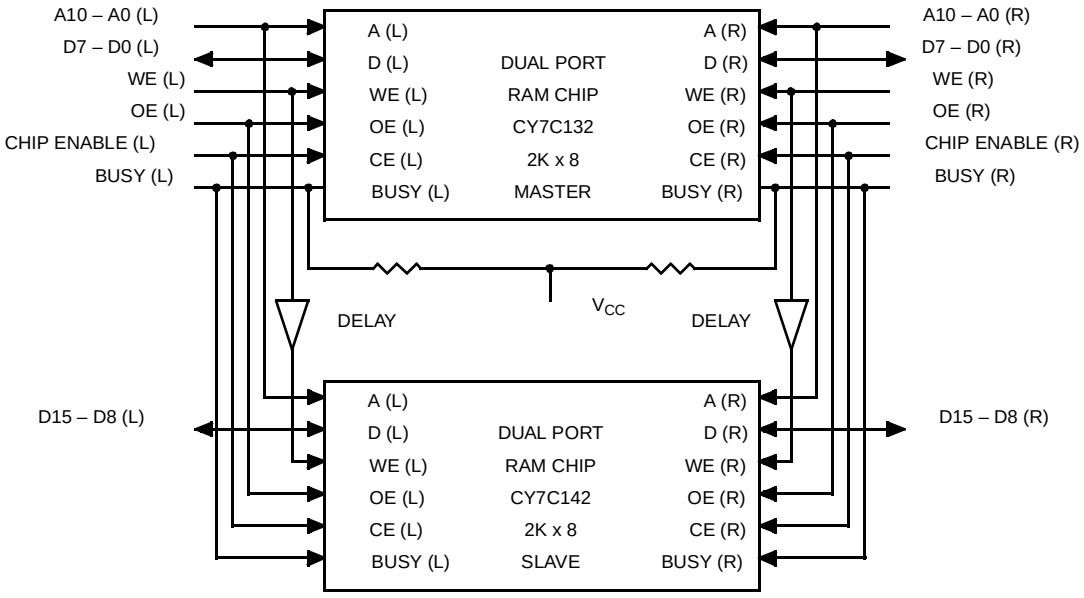

Figure 9. Expansion (2K x 16) with Slave

Delaying the Write Strobe

In width expansion, the write signals to the slave devices must be delayed by an interval at least equal to t_{BLA}, which is the time required for the master to assert the busy signal to the slave after an address match. The delay prevents the slave data at the address in contention from being overwritten. Both the write and read cycle times must be increased by this amount of time. In equation form:

$$t_{WC} = t_{PWE} + t_{BLA} \quad \text{Eq. 1}$$

where the delay must be at least equal to t_{BLA}.

Note that if you add more slaves to make a wider word, (e.g., 24 or 32 bits) the delay elements' outputs can connect directly to the write strobe inputs. Additional delay elements are not required.

Slave Standalone Operation

Some applications might require that you give one port permanent and absolute priority over the other. You can easily do this by implementing the memory using only slave dual-port RAMs. The $\overline{\text{BUSY}}$ input to the priority port must be tied HIGH by either connecting it directly to V_{CC} or to V_{CC} through a 10-KΩ pull-up resistor. You can connect the low-priority port's $\overline{\text{BUSY}}$ input to the high-priority port's read/write input.

In this configuration, the busy (read/write) signal to the lower-priority port always prevents the port from writing when the high-priority port is writing to any location. The data of the lower-priority port is overwritten when the two ports operate asynchronously, the lower-priority port is writing, and the higher-priority port simultaneously writes. This is not a very elegant solution because the $\overline{\text{BUSY}}$ input to the low-priority port is not qualified by comparing the addresses of the two ports or their chip enables. However, this approach suggests how the slave dual-port RAMs can be used with external arbitration logic. The busy inputs can be used by control logic or under program control to dynamically change the port priorities.

If the lower-priority port is read only, you can tie its $\overline{\text{BUSY}}$ input HIGH by either connecting it directly to V_{CC} or to V_{CC} through a pull-up resistor.

Dual-Port Design Example

The following design example illustrates the methodology to follow when designing with Cypress dual-port RAMs. In this example, a dual-port memory is used for message passing and bus snooping for many bus masters on a 32-bit-wide system bus. The dual-port RAMs interface to a 32-bit system bus on the right side and a 16-bit processor on the left side. From the right port, the memory appears as 8K 32-bit words, and from the left port the memory appears as 16K 16-bit words.

The memory has the following characteristics:

1. The memory location corresponding to address 0 for both ports is the same.
2. The data read from and written to the memory from both ports is in the same order. Thus, D0 of the right port corresponds to D0 of the left port. Additionally, D16 of the right port appears as D0 of the left port in address location 2048.
3. The minimum cycle time is 35 ns.
4. To conserve power, blocks of addresses are decoded to generate the required chip selects.
5. The CY7C132 and CY7C142 dual-port RAMs are used. Part of the design task is to specify the number of masters and slaves required and the way they must be interconnected.
6. The appropriate $\overline{\text{BUSY}}$ signals must be generated to the correct port when contention occurs.
7. All possible mailbox locations that can be used for message passing are used.

8. The right port signals are AR0...AR12, DR0...DR31, $\overline{CER}$, and $\overline{BUSYR}$. The left port signals are AL0...AL13, DL0...DL15, $\overline{CEL}$, and $\overline{BUSYL}$.

A simplified logic diagram of the memory appears in *Figure 10*. A total of 16 2K x 8 dual-port RAMs are required. The devices labeled MA (master, bank A) through MD (master, bank D) are CY7C132 masters. The devices labeled SU (slave, upper half-word) and SL (slave, lower half-word) are CY7C142 slaves. The memory consists of four masters and twelve slaves, along with the required control logic.

From the right port the memory is configured as 8K 32-bit words, with a master controlling three slaves. The one-of-four decoder labeled RB (right bank) generates chip enable signals for each bank of 2K 32-bit words. Data is written (sampled) on the bus side, and the only reads performed are from the mailbox locations.

A general-purpose, right-port, control-logic block generates control signals that conform to the timing diagram shown in *Figure 11*. The diagram does not show the generation of the output enable control signals, but they are similar to the RB decoder signals. If your application does not require message passing to the right port, you can tie the right-port output enable pins of all of the dual-port RAMs directly to V_{CC}.

From the left port, the memory is configured as 16K 16-bit words. For this organization, you might think that the slave dual-port RAMs in the second column from the right in *Figure 10* should be masters. If this were the case, however, you would have to defeat the arbitration logic in them when the right port addressed the same address; this would add logic, reduce the speed, and complicate the design. Therefore, this design uses a combination of left-bank decoding (LB, 1-of-4 decoder) and upper-lower 16-bit word decoding (UL, 1-of-8 decoder) to cause the bank master to arbitrate when the right port is addressing the same bank as the left port (more on this later).

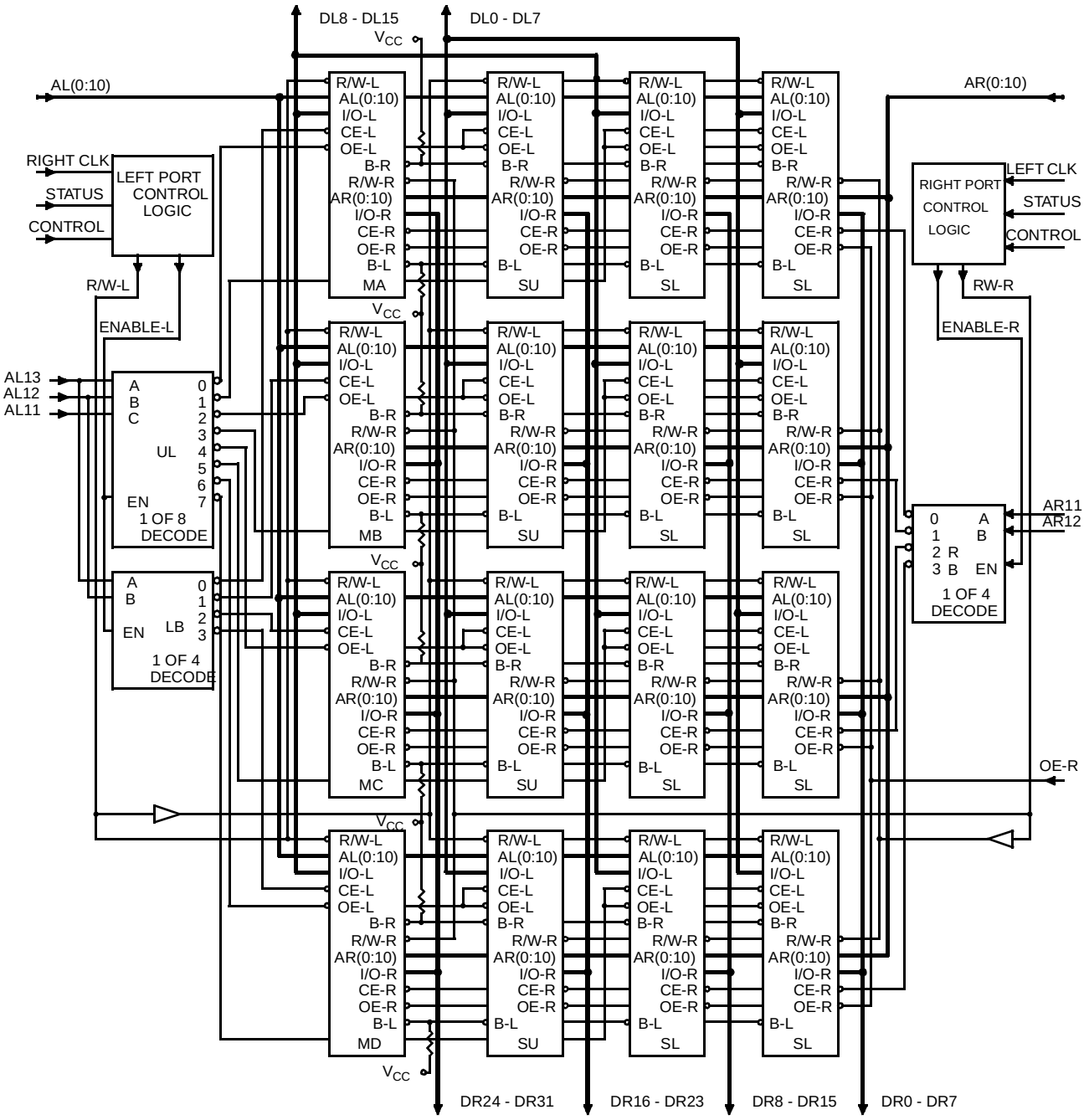

Figure 10. Logic Diagram for Dual-Port Example

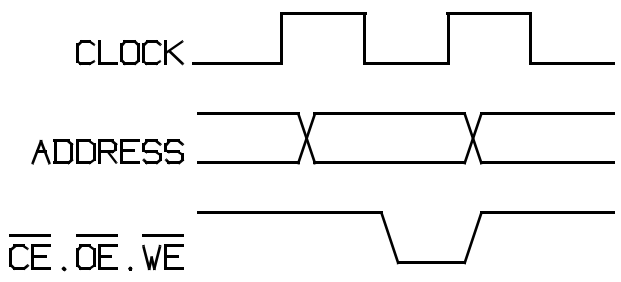

Figure 11. Timing for Dual-Port Example

Right-Port Operation

For purposes of this discussion, "word" refers to the 32-bit word at the right-port system-bus interface. At the 16-bit processor interface, the 32-bit word is referred to as either the lower half word (right-port bits 0 through 15) or the upper half-word (right-port bits 16 through 31).

The bank-selection process employs the chip enables. Specifically, the 1-of-4 RB decoder decodes the four combinations of the upper two right-port address-bus signals and generates four active-LOW chip enables to each bank of four dual-port RAMs. Bank A contains addresses 0 through 2047, bank B contains addresses 2048 through 4095, bank C contains addresses 4096 through 6143, and bank D contains addresses 6144 through 8191. In other words, bank A addresses 0 to 2K, bank B 2K to 4K, bank C 4K to 6K, and bank D 6K to 8K.

The lower 11 right-port address lines, AR(0:10), are connected to the A0 through A10 right-port address pins of all the dual-port RAMs.

Figure 11 does not show the generation of the write strobe, but does show the signal's timing. The write enable is applied directly to all the masters in parallel, then buffered, and then applied to all the slaves. The minimum propagation delay of the buffer must be at least as large as t_{BLA}, which is the time required for the master to assert the busy signal to the slaves after an address match occurs.

Note that all the right-port output-enable pins are connected together. These pins should be driven if reading is required; otherwise connect them to V_{CC}.

The open-drain busy outputs of the right port masters must be pulled up to V_{CC} using resistors. A value of 330Ω is recommended. The master busy outputs connect to all the right-port slave busy inputs for each bank.

For the data bus interface, the I/O pins of each RAM column connect to their respective I/O pins on each bank. This OR-tie connection is allowed because the bank-selection chip enable causes the output buffers of the unselected banks to go to the high-impedance state.

Left-Port Operation

The 1-of-4 decoder labeled LB performs bank selection for the left port. The upper two left-port address lines, AL13 and AL12, decode bank-select chip enable signals for the four masters only. Bank A corresponds to addresses 0 through 4095, bank B corresponds to addresses 4095 through 8191, bank C corresponds to addresses 8192 through 12,287, and bank D corresponds to addresses 12,288 through 16,383.

To perform upper and lower half-word selection, the 1-of-8 decoder labeled UL decodes the upper three right-port address signals. The decoder then generates eight chip enable signals with a resolution of 2048. The chip enables connect to the slaves' chip-enable and output enable pins (2048 resolution) and to the masters' output enable. Because the master chip enable resolution is 4096, the master arbitrates for two blocks of 2048 16-bit half words.

The lower eleven left-port address lines, AL(0:10), connect to left-port address pins A0 through A10 of all the dual-port RAMs.

At the 16-bit interface, writing is only required if the left port wishes to send a message to the right port. Otherwise, you can connect the left-port write pins of all the dual-port RAMs to V_{CC}.

To implement the left-port data bus interface, the left port's data I/O pins are connected together in the same manner as those of the right port for all RAMs in the same column. In addition, to multiplex a 32-bit data word to a 16-bit half word, the least-significant bytes and the most-significant bytes of each 2048-word group are connected together. The UL decoder that controls the left-port output enable performs the selection.

If you use the masters' interrupt pins, pull them up to V_{CC} through a 330Ω resistor and connect them to the processor interrupt-request input. You can leave the slaves' interrupt pins unconnected.

If the control signal connections from their source to the dual-port memory constitute electrically long lines, they might require proper termination to avoid voltage reflections due to impedance mismatches. Refer to Cypress's application note titled "Systems Design Considerations When Using Cypress CMOS Circuits."

References

1. Dijkstra, E.W., "Solution of a Problem in Concurrent Programming Control." *CACM*, Vol 8, no.9, Sept. 1965, p 569.
2. Dijkstra, E.W., "Co-operating Sequential Processes." *Programming Languages*, F. Genyus (Ed.) Academic Press, New York, 1968, pp 43 – 112.

Notes

1. The Interrupt function is not available at the 2K x 8 level in a 48-pin package.