DUAL-PORT SRAMs WITH SEMAPHORE ARBITRATION

APPLICATION NOTE AN-14

By Michael J. Miller

Introduction

Due to their high bandwidth and message access flexibility, dual-port SRAMs are used to link multiple high-performance processors and systems. Integrated Device Technology makes dual-port SRAMs in many configurations, all of which consist of one SRAM with two sets of address, data and control signals. This allows two processors to share the same block of physical memory in their respective address spaces. The two processors can access data in two memory locations simultaneously and asynchronously. This approach clearly outperforms a discrete part's design where two processors must synchronize through arbitration for access to a bus which is used to access one location at a time in a standard single-port RAM.

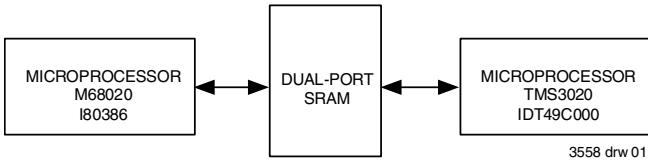


Figure 1. Dual-Port RAMs Link High-Performance Processors

IDT's dual-access approach removes synchronization requirements at the memory's bus access level. Nevertheless, synchronization must be performed at other levels to ensure data integrity and proper system operation. This application note addresses several approaches to solving the mutual exclusion problem and gives a detailed discussion of the semaphore capability provided by the IDT71342.

Arbitration

Consider a multiple-processor system where each processor has access to the same data. Arbitration schemes are necessary to resolve the situation when multiple processors want the same piece of data at the same time. Different approaches to the arbitration issue have different tradeoffs and are best-suited for different applications. These solutions vary from no arbitration, hardware solutions, software solutions, and combinations thereof.

Seemingly, the simplest solution is to employ no arbitration at all. This approach works if the application guarantees that two processors will not access the same location simultaneously or, if they do, then the indeterminate results are acceptable. Sometimes handshaking can be employed through I/O ports or interrupt mechanisms. This approach provides a high-performance, low overhead design but is restricted to certain applications. If arbitration is not required, the IDT7134 can be used. It is a 4K x 8 dual-port RAM with no arbitration. This part can also be used in large dual-port designs where one hardware arbiter is used for a whole array composed of many IDT7134s. The interrupt handshake mechanism can be achieved by using devices like the IDT7130/7140.

Most applications cannot sacrifice data integrity and utilize the dual-port memory as a collection of individual memory locations which require a finite access time. In this case, arbitration at memory location resolution is required. For example, the IDT7130 and IDT7132 use an address comparison mechanism which provides a $\overline{BUSY}$ signal to the losing side. When the two processors try to access the very same location, the arbitration asserts the $\overline{BUSY}$ signal to the processor which attempted access last. When access attempts are within 5ns of each other, a side is chosen arbitrarily. The $\overline{BUSY}$ outputs are suitable for attachment to the READY or DTACK inputs of most microprocessors. This approach is very straightforward and flexible and has the benefit that a processor cannot be locked out of the RAM longer than the access period of the other processor.

The features of the IDT7130/7132 and other device with $\overline{BUSY}$ that make them a superb solution in many designs may create problems in other applications. The fact that $\overline{BUSY}$ lines are used and that arbitration resolution is at the level of individual locations can be a major limitation in some instances. Many significant controllers, such as the IDT 8031 and IDT8051, are not equipped with READY or DTACK input pins. Of those that are equipped, a penalty is often paid in the higher performance versions if they require "seeing" the $\overline{BUSY}$ signal faster than the IDT7130/7132 can supply it (16MHz 68020 requires 25ns $\overline{AS}$ to $\overline{DSACK}$). In these cases, wasteful wait cycles are required. In other applications, software constraints may require mutual exclusion at the software data structure level rather than at the memory cell location level. For this reason, Integrated Device Technology developed devices like the IDT71342.

Instead of comparing addresses on every cycle, and occasionally asserting $\overline{BUSY}$ status, the IDT71342 employs circuitry to support a software mechanism called semaphores. Here, every memory cycle is equally as short as the next and arbitration is handled at the software level.

The semaphore concept was pioneered by E.N. Dijkstra in 1968. He developed a test and set approach for single processor multi-tasking systems. The task tests a memory location (a semaphore) for a particular value and, on the next cycle, the task sets the same location a unique value. If the semaphore was already set, then the current task knows that another task has access. If the value was not present, then the task knows that it has permission to proceed and all other tasks are blocked because the semaphore is not set. Only one task at a time has permission via the semaphore. Semaphores are used like locks to resources such as disk buffers, message queues, critical code sections, shared access to communication controllers, etc.

Because the test and set operation requires that the two memory accesses are indivisible in time, the IDT7130/7132 will not support semaphores for many processors and systems. This occurs because one processor may test the semaphore and, before it can set it, the other processor might test it, too. In this case, both processors "believe" they

have the semaphore. The IDT71342 employs a twist by using set and test. The "set" corresponds to a request and the "test" checks to see if the request was granted. The indivisible double access requirement is avoided because, as soon as a request is made by one processor on one side, the grant is blocked on the other side. Some processors support test and set operations through a read/modify/write operation, but the memory bus design must support the processor in such a way that the address and the chip select remain constant. When the test and set instruction is used, arbitration must take place. As will be seen, semaphore operation without hardware busy arbitration has many advantages.

The IDT semaphore scheme employs a software/hardware approach which provides a secure method of resource allocation with the flexibility of software configuration and control and the resolution of hardware. Since there is no hardware relationship between semaphores and dual-port memory locations, the block sizes, locations and semaphore association are defined by the software. The semaphores can also be used to allocate other resources such as I/O devices. This offers the system designer considerable flexibility.

As an example, a dual-port SRAM might be shared by a disk controller processor and a host processor. When the controller is accessing a buffer in memory (e.g. when writing a sector in a track), the main processor cannot be allowed to interrupt or delay the controller. By setting the semaphore, the controller has exclusive access to the disk buffer. When done, it releases the semaphore and therefore provides access to the disk buffer by the processor on the other side.

Because the processors must test and set a semaphore with multiple bus cycles, the semaphore arbitration scheme has a longer arbitration latency than the address comparison scheme. Since arbitration is most often used for access to multiple locations in memory the overhead can be amortized across multiple accesses. In systems that require mutual exclusion of access to data structures over a period longer than one memory cycle, this trade-off is irrelevant.

Functional Description of the IDT71342

The IDT71342 is a fast dual-port 4K x 8 CMOS static SRAM with semaphore logic, packaged in a 52-pin PLCC and 64-pin TQFP. The semaphore logic can be used to allocate portions of the dual-port SRAM to one side or the other and is used in place of the address arbitration logic used in other dual-port designs. Semaphores are software-controlled. Therefore, this approach provides several advantages including allocation of multiple blocks of arbitrary size and no processor WAIT states or $\overline{BUSY}$ logic.

Like other IDT dual-port SRAMs, the IDT71342 allows access to a common set of SRAM cells from two independent ports. Each port is functionally identical to that of a conventional static RAM. Both ports are completely independent and asynchronous in operation. Reading or writing on one port does not affect the operation or timing of read/write operations on the other port. Unlike the IDT7130/7132, the IDT71342 does not employ hardware arbitration which blocks write access. If one port is writing to a location while the other port is reading that same location, the data will change during the read. If both ports attempt to write to the same location at the same time, the result will be some combination of the two data words being written. If both ports are reading, however, there is no interaction because the data does not change.

How the Semaphore Flags Work

The semaphore logic is provided by a set of eight latches. These latches can be used to pass a flag, or token, from one port to the other to indicate that a block of SRAM is in use. The internal circuitry prevents the flag from being passed in both directions at the same time. The semaphores provide a hardware assist for a use assignment method called "token passing allocation". In this method, the state of the semaphore latch is used as a token indicating that a block of SRAM is in use. If the processor on the L port wants to use a block of SRAM, it attempts to set the latch, requesting

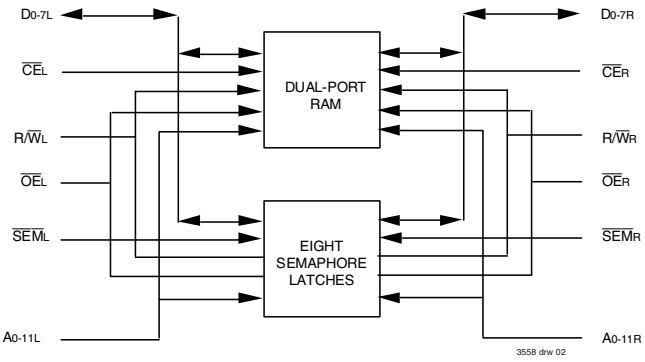


Figure 2. Functional Block Diagram of Dual-Port SRAM with Semaphores

the token. The processor then checks the latch to see if it was successful in setting the semaphore. If it was, the processor proceeds to read and/or write in the block. If the processor was not successful in setting the latch, it means that the R port had set it first, has the token and is using the block. The L port then continues to test until it is successful, indicating that the R port has released the token and is no longer using the block.

The semaphore logic is independent of the dual-port SRAM. These eight latches can be accessed from either port by enabling the semaphore chip enable (SEM = V_{IL}), which is separate from the SRAM chip enable. When the semaphore logic is enabled on a port, one of the eight latches can be read or written from that port. The latch is selected by the three least significant address pins for the port and the data for reading and writing uses the D_0 data pin.

A semaphore latch is read or written in the same manner as an SRAM cell. The latch is written to a "1" or "0" by activating the semaphore logic enable, selecting the latch with the three least significant address bits, activating the write enable and putting a "1" or "0", respectively, on the D_0 data pin. The latch may be read by activating the semaphore enable, selecting the latch, holding the write enable HIGH and reading the data on D_0, For the user's convenience, all eight of the data lines are set to the same value as D_0 during read. In other words, the data lines will contain all "1"s or all "0"s when D_0 is a "1" or a "0", respectively. In this way, branch zero testing can be employed.

The semaphore read logic latches the readout state of the semaphore flag during the read. This prevents the value seen by the reading port from changing during the read, even though the state of the latch may be changing internally due to write activity on the other port. The latch goes into the hold mode when both semaphore enable and output enable are active. In order to see the latch change, either the semaphore enable or output enable must be disabled, and then enabled. This means that read operations must be cyclic; it is not possible to enable the semaphore and

output enable continuously and wait for the latch value being read to change.

The semaphore logic is active LOW. An access token is requested by writing a "0" to the semaphore latch and is released by writing a "1". To request a token, an attempt to write a "0" to the semaphore is made and the semaphore is read to determine if the "0" was successfully written. If a "0" is read, the token request was granted. If a "1" is read, the request was denied and the other port has the token.

The critical case of semaphore timing occurs when both ports request the token by writing a "0" at the same time. The semaphore logic is specially designed to resolve this problem—if requests are made simultaneously, the logic guarantees that only one side receives the token. In this case, the token assignment will be made arbitrarily to one port or the other.

Figure 2 shows the internal logic circuitry for one semaphore "latch" cell. It is composed of multiple latches and cross-coupled AND gates which serve as an arbiter to guarantee that only one side at a time receives a grant signal. A typical sequence of semaphore operations is listed in Table 1. The D_0 columns represent the logic value that would be read on that side. The "Request F/F's are the internal flip-flops which store the state of requests.

Use of Semaphores

Semaphores provide useful solutions for various problems at both the hardware and software levels. The following selections highlight a few of the semaphore benefits which range from increasing performance to providing functionality not available with other designs.

High-Performance Dual-Port Design

To gain a deeper understanding of the trade-offs between semaphore and non-semaphore dual-port SRAM designs, the following example compares both approaches. Dual-port memory system design requires a key awareness of the microprocessor's memory access time requirements. Figure 3 is a read cycle timing diagram of a 20MHz 68020. Two timings are critical: A 45ns address to data size acknowledge ($\overline{DSACK}$) to guarantee no wait states and a 95ns address to data. It is also important to examine a typical design. Figure 4 shows the interface between a single processor and one side of the dual-port. For simplification, the other port interface was omitted from the drawing. This example shows the address bus which is decoded by a comparator (IDT74FCT521A) and an address decoder (IDT74FCT138A). The address interface chooses which dual-port SRAM to enable. After the chip enable is enabled, chip enable arbitration (available on all IDT DPRAMs except for the IDT7014) and data access can begin.

In a tightly-coupled system (i.e., the 68020 processor and dual-port are on the same board), chip select can be generated from address in 13ns. In the best case, the data acknowledge is tied to the 68020 through a NAND gate (to include other acknowledges). The NAND gate will

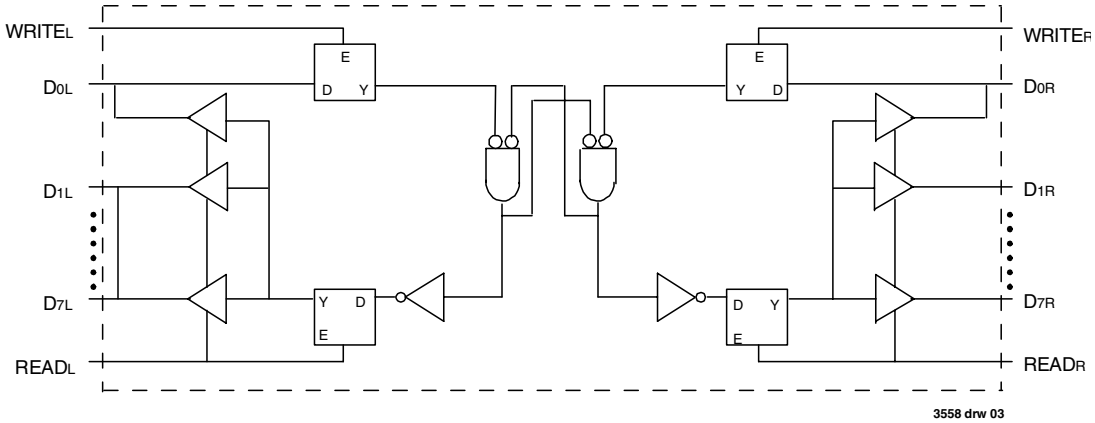


Figure 3. Simplified Diagram of One Semaphore Cell

Function	Left		Right		Function
	D0	Request F/F	Request F/F	D0	
No action	1	1	1	1	Semaphore Free
L port writes 0	0	0	1	1	L port has token
R port writes 0	0	0	0	1	No change; L port keeps token
L port writes 1	1	1	0	0	Semaphore freed; R port gets it
R port writes 1	1	1	1	1	Semaphore free
L port writes 0	0	0	1	1	L port has token
L port writes 1	1	1	1	1	Semaphore free

Table 1. Semaphore Function Table

3558 tbl 01

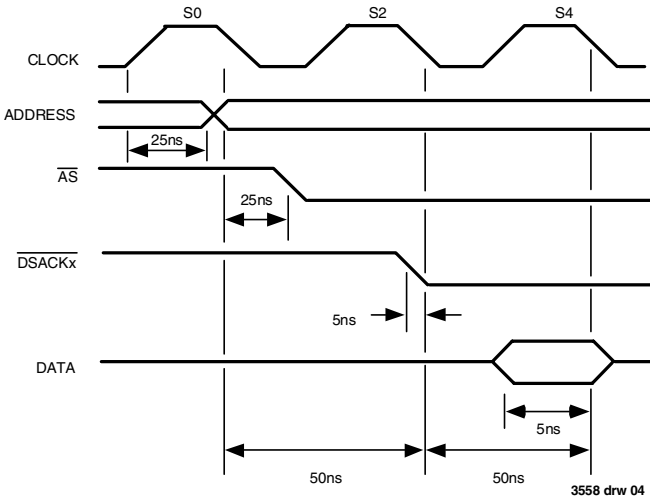


Figure 4. Read Cycle Timing for 20Mhz 68020

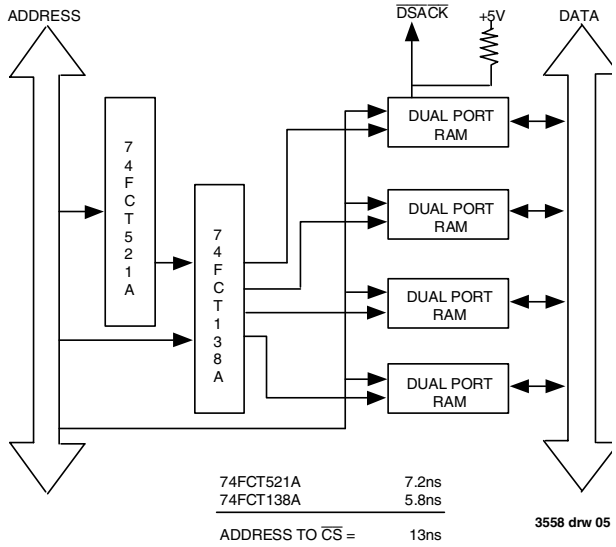


Figure 5. Memory Interface to One Port of a Dual-Port RAM System

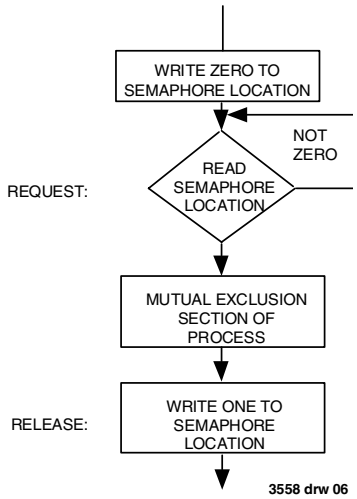
introduce another 5ns delay. This leaves 26.9ns to generate the acknowledge (DSACK) and meet the 5ns setup time to guarantee that a wait state will not be inserted. In a less rigorous design where the dual-port and CPU are on separate boards, 10ns or more may be required for on/off board buffers and bus delay, etc. This leaves 16ns or less to generate acknowledge.

Considering the timing constraints, the designer can choose from several options. In applications which require arbitration resolution to the memory cell level, 26.9ns is not enough time to generate DSACK from CE using the IDT7130L55. One solution involves adding logic to the BUSY/DSACK path so that a wait state is always inserted until the dual-port can respond with BUSY. This will slow down the system whenever the dual-port is accessed. If block arbitration or higher memory cycle performance are required, the designer should utilize the IDT71342. This configuration would only be constrained to the 95ns address to data access time, minus any address and data buffer time. The IDT71342 provides high enough performance for use with the 25MHz 68020. Some software overhead is required for semaphore access but, given the fact that the semaphore arbitration is for a block of locations, the arbitration latency can be amortized across multiple higher speed accesses. Consequently, the semaphore approach provides a higher performance solution if block arbitration is desirable or acceptable.

A Software View of Semaphores

The dictionary defines semaphore as "signaling by flags". A semaphore is implemented as a specialized type of memory location which can be accessed by either processor in a dual-port design. Two different operations are performed on the semaphore: the request operation which attempts to gain access and the release operation which signals the termination of access. These operations are used to guarantee mutual exclusion, meaning that only one processor is accessing a resource at any given time. This occurs from the time a request is granted until the time that the semaphore is released.

A semaphore is chosen which both processors associate with one resource. First the processor requests the semaphore by attempting to write a "0" to the semaphore location. Then it reads the location. If it receives a non-zero value (i.e. a "1"), it loops back and reads the semaphore location again. It will continue to read the location until it receives a "0". The software may be written in such a way that useful work may be performed while waiting. When a "0" is read, the processor can access the resource for as long, and as many times, as desired. The processor must release



Flow Chart 1. Sequence of Operations on Semaphore to Guarantee Mutual Exclusion

the semaphore when it is finished with the resource. This is achieved by writing a "1" to the semaphore location.

Using Semaphores at the Software Level

One example of where semaphores might be applied involves two processors working together to generate a video display for animated images. The "MASTER" processor generates a picture layout in the form of a display list. The "SLAVE" processor reads the display list, interprets it and generates an image in a display buffer. As the image is displayed, the video buffer is cleared. The displayed list is reinterpreted and

displayed. If the display list is changed, the image appears as though it has moved, giving the illusion of animation.

A dual-port SRAM is used to store the display list. The SLAVE interprets one display list repeatedly to generate the display buffer image, while the MASTER generates and updates another display list. The SLAVE processor continuously updates the video display buffer since the buffer is wiped clean when its contents are dumped to the video screen.

In this particular application, the dual-port SRAM is broken up into three areas. The first area contains common information concerning which display list is being accessed and which one is being updated. It is locked with the semaphore $\overline{SEM}_0$. Two buffers comprise the other areas and are locked by semaphores $\overline{SEM}_1$ and $\overline{SEM}_2$. At any given time one buffer is used for the display list currently being interpreted and the other is used for the list being built. The common area stores the pointer which indicates which buffer is being updated.

The key to the effectiveness of this approach lies at the software level. The flow chart for the master processor begins with a buffer request via a semaphore. Once granted, it builds a display list. Then it releases the buffer through the semaphore mechanism. Next it calls a routine to inform the SLAVE processor to switch over to the new buffer. It then loops back to request access to the other buffer.

The SLAVE processor functions by first fetching the current buffer number. Then it requests the buffer via the semaphore mechanism (involving $\overline{SEM}_1$ or $\overline{SEM}_2$). Once the SLAVE gains access to the buffer, it builds the display from the list. After releasing the buffer, it goes back to fetching the current buffer/number. This is necessary because the MASTER processor may have switched buffers. Fetching the current buffer/number requires access to the common area which is achieved by obtaining the semaphore $\overline{SEM}_0$. After accessing the data, the SLAVE releases $\overline{SEM}_0$ which allows the MASTER to come in and update the common area.

The software code for the MASTER and SLAVE processors is listed on the following pages. It is in the form of a pseudo-"C" language-type program. The request for a semaphore is made by the WHILE statements accessing a variable called SEM. The semaphore is released by writing a "1" to that variable.

Semaphores and Caches

In high-performance dual-port systems, semaphores can be used with caches to achieve valid data synchronization. The use of caches is an established method of speeding up access between a processor and main memory. Main memory may be slower due to the use of lower cost, higher density DRAMs or system bus latency. The cache operates by

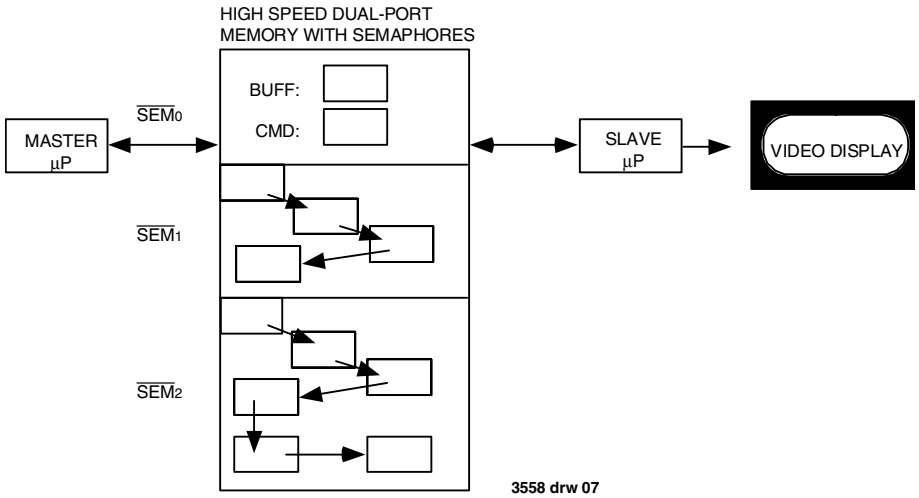
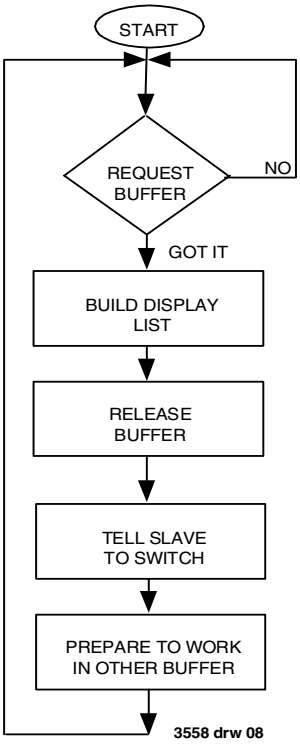
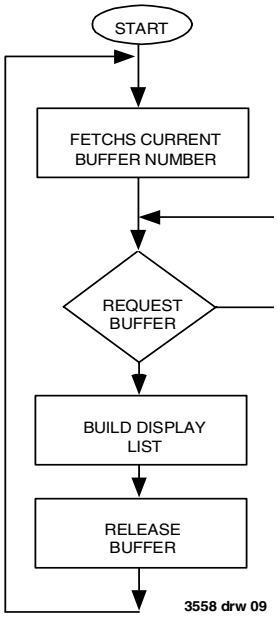


Figure 5B. Software Block Diagram of Video Display System for Animation



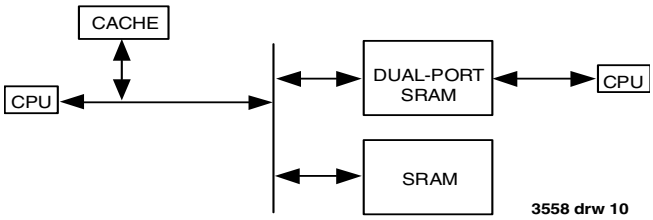
Flow Chart 2. Sequence of Operations for Master Processor



Flow Chart 3. Sequence of Operations for Slave Processor

monitoring data transfer between the processor and memory. When write operations are performed, the cache remembers the data and location. When a read is performed it compares the address of the request with a list of locations it has data for. If the address matches, the cache supplies the data and aborts the main memory access. If no match occurs, the cache allows the main memory access to proceed and notes the data and location.

One might first assume that the dual-port SRAM can always be used with cached memory accesses. However, extra considerations must be made. When data is written to a memory location in dual-port SRAM, the cache stores the acquired value and its associated location. The next time that location is read, the cache will register a "match" and bypass reading from the location in dual-port SRAM. This might result in an error if a processor on the other port has written new data to the location.



Flow Chart 4. Dual-Port SRAM in a Cached Memory Environment

One way to remedy the situation is to put the dual-port SRAM into non-cached I/O address space and block data transfer between the dual-port SRAM and cached address space where standard SRAM exists. To make this approach work, semaphores must be employed to lock a buffer in the dual-port SRAM while the data is in the cached SRAM. In this way a "check out" procedure can be implemented to ensure data integrity. The semaphore latches must be addressed through non-cached I/O space in order for the request and release mechanism to function correctly.

Conclusion

There are a number of ways to handle dual-port SRAM arbitration. Choice of the most efficient technique concerns what granularity of address arbitration is required, whether a processor must be locked out of a block of memory for multiple accesses from the other processor and what constraints are imposed by the memory access cycle timing. Semaphores provide an alternative which can result in higher performance systems and provide functions which are not otherwise achievable. The following is a quick summary.

No Busy Logic- Some applications guarantee by definition that the two processors will not access the same locations simultaneously or, if they do, it doesn't matter. The IDT7134 is also ideal for use in large dual-port designs where one arbiter is used for an array of dual-port devices.

Interrupt Logic - Interrupt logic provides a signaling method from one processor to the other to provide a mechanism for handshaking.

Hardware Busy Logic- Hardware busy logic provides the lowest latency overhead when accessing multiple individual unrelated memory locations. The MASTER/SLAVE concept was introduced by IDT to provide a single arbiter, thus avoiding deadlocks encountered with multiple arbiters when using more than one dual-port in wide bus applications.

Semaphore Logic- Semaphore logic provides the best overhead trade-off when accessing a block of data comprised of multiple related locations. This facility may also be required in high performance applications where one of the processors does not have a ready/busy input or the overhead of wait states cannot be tolerated.

Semaphores provide a mechanism for one processor to bar the other processor from seeing an incomplete update of a block of data. This is achieved through a software mechanism supported by on-chip circuitry which provides a test and set facility that arbitrates between simultaneous requests.

CODE FOR MASTER PROCESSOR

```
MAIN     (      )    {
                   /*            code  to  initialize  */

          FOREVER          {
                 SEM  (CUR_BUF):=  0
                 UNTIL  (SEM  (CUR_BUF)  =  0);                  /*request*/
                 BUILD_DISPLAY  (CUR_BUFF);                      /*Build  new  display  list*/
                 SEM  (CUR_BUFF):=  1                            /*release*/
                 SWITCH_BUFF  (CUR_BUFF);
                 IF  (CUR  -=  BUFF  =  1)
                        CUR_BUFF:=  2;
                 else  CUR_BUFF:=  1;
                 }
          }                                                 /*end  MAIN*/

SWITCH_BUFF  (NBUFF)    {
          SEMO:=  0
          UNTIL  (SEMO  =  0); /*request*/
          BUFF:=  NBUFF;
          CMD:=  NEW;
          SEM:=  1;                          /*release*/
          RETURN  (  )
          }
```

CODE FOR SLAVE PROCESSOR

```
MAIN  (  )      {
          FOREVER      {
                 CUR_BUFF:=  FETCH_BUFF  (  );
                 PROCESS  (CUR_BUFF);
                 }
          }

FETCH_BUFF  (  )       {
          SEM  0:=  0;
          UNTIL  (SEMO  =  0);                              /*request*/
          A  BUFF:=  BUFF;
          CMD:=  OLD;
          RETURN  (ABUFF);
          SEMO:=  1;                                        /*release*/
          }

PROCESS  (BUFF)       {
          SEM  (BUFF):=  0;
                 UNTIL  (SEM  (BUFF)  =  0);                /*request*/
                 REFRESH  (BUFF):                 /*code  to  refresh  display*/
                 SEM  (BUFF):=  1;                          /*release*/
                 }
```

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use o any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.