INTRODUCTION TO IDT's FourPort™ SRAM

APPLICATION NOTE AN-45

By John R. Mick

Introduction

Integrated Device Technology is continuing to pioneer higher speed and higher density static RAMs. As IDT has improved its CMOS technology, new SRAM architectures and additional features have become feasible. The end result is that design engineers now have powerful new integrated circuits available for demanding applications. Two such devices are the IDT7052 and the IDT7054 FourPort™ SRAM. The 7052 is a Four-port 2K by 8-bit Static RAM built using a 12-transistor four-ported static RAM cell. Each of the four ports is independent in terms of the byte that it can read or write.

The IDT7052/IDT7054 FourPort™ SRAM provides the system architect with better ways to look at computer system design. For example, the IDT7052 can be used in a multiprocessor environment to provide a common memory among several processors. An example of such an architecture is shown in Figure 1. Here we see each of the four SRAM ports connected to a high performance microprocessor. These processors could also be intelligent controllers, DSP engines, or a combination of the two. The FourPort™ SRAM can be used in such computer architectures as hypercubes and parallel processing machines for storage and movement of data. It offers unheard of opportunities in digital signal processing (DSP) where new architectures for Fast-Fourier-Transforms (FFTs), recursive and non-recursive digital filters, windowing functions, and special purpose algorithms can take advantage of multiple ports into a shared memory. The IDT7052/IDT7054 FourPort™ SRAM can increase system performance and reduce parts count by providing simultaneous access to the data by more than one processor at a time.

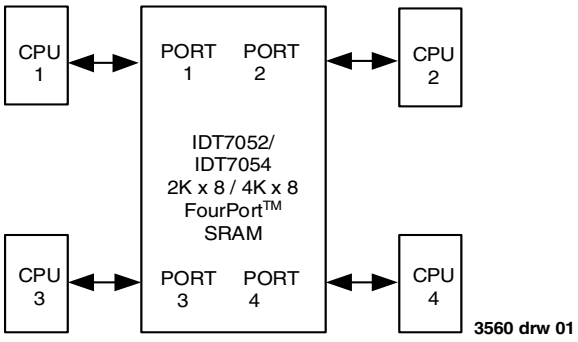

Figure 1. Four-Port™ SRAM Providing Common Memory to Four CPU's

Understanding the FourPort™ SRAM

In order to effectively design with the IDT7052/IDT7054 FourPort™ SRAM, it is important for the design engineer to understand its construction and architectural features. This is most easily accomplished by starting with a simple single port SRAM cell and evolving its architecture into the FourPort™ structure. Figure 2 shows a typical single port static RAM built using a four-transistor cell. This architecture is commonly used by most static RAM manufacturers to build static RAMs because it offers high density, good speed and low power.

In its simplest description, the device consists of two N-channel transistors (Q_1) and two resistors (R_1) that are connected so as to form two simple cross-coupled inverters. This gives a regenerative action such that one N-channel transistor is ON and the other N-channel transistor is OFF. Thereby, a single bit of memory is formed. In order to interface to this cross-coupled pair of inverters, two additional N-channel transistors (Q_2) are connected between the inverter outputs and the bit-lines. The gates of these two N-channel transistors are connected to a line called the row select. These Q2 transistors connected between the cell and the bit lines are usually called transmission gates. The result is that when a particular row of cells in the SRAM is addressed, these two transistors are turned

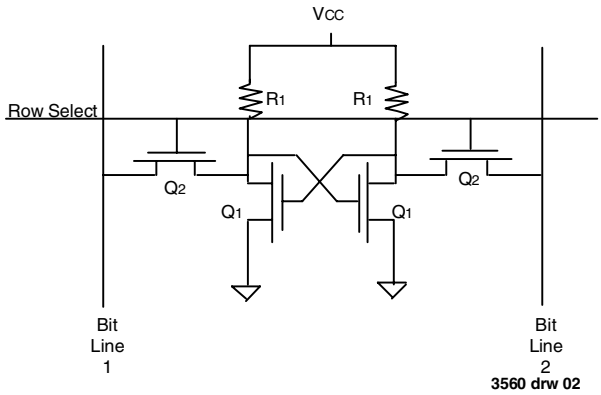

Figure 2. Typical Four-Transistor SRAM Cell

JUNE 1999

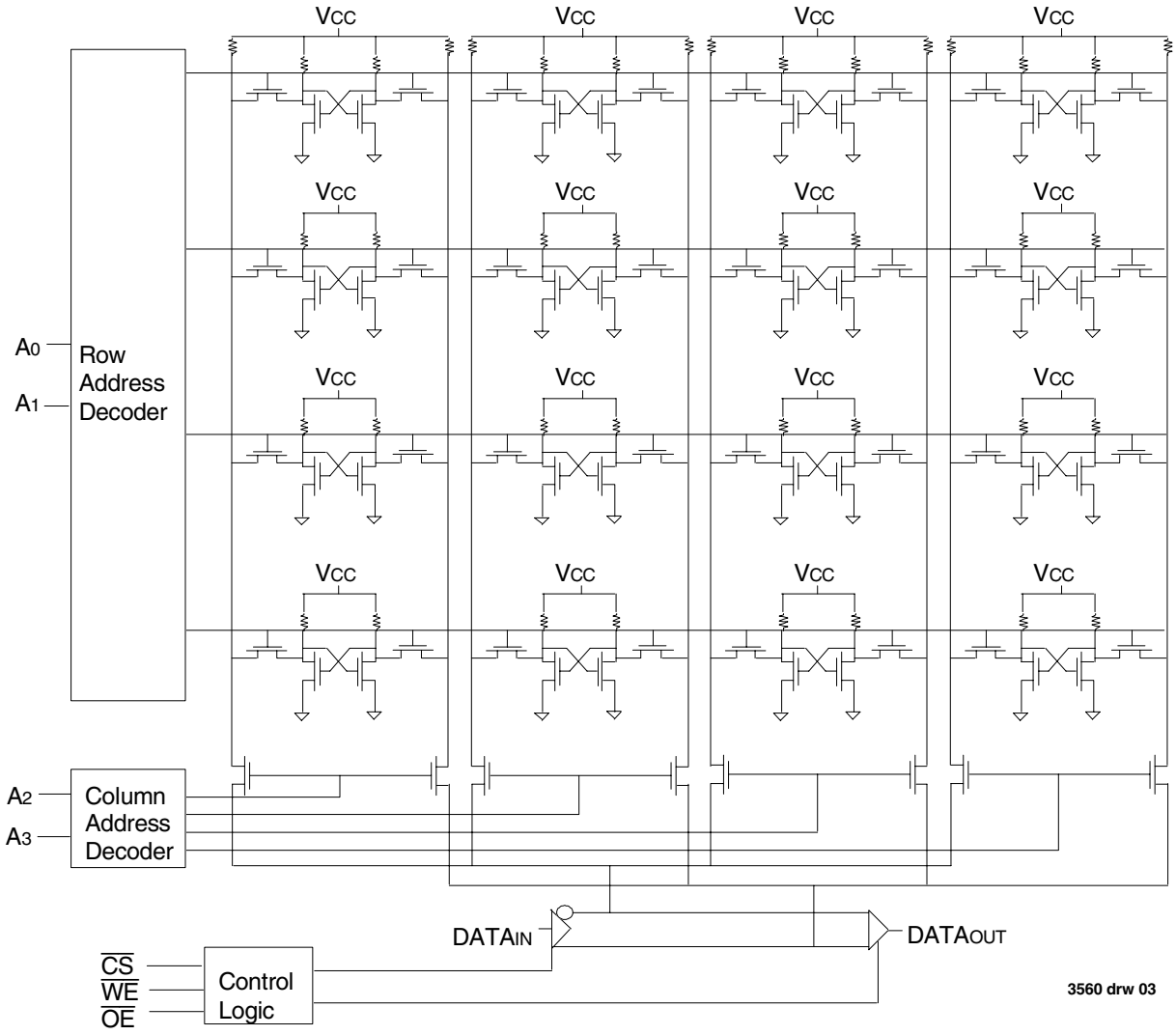

Figure 3. An Example Four-Transistor Cell for a 16-bit SRAM.

on and one bit-line will reflect a HIGH and the other bit-line will reflect a LOW as determined by the current state of the static RAM cell.

An expanded example of this SRAM architecture is shown in Figure 3. Here we see a 16-bit SRAM, organized four-rows by four columns internally, in a more complete form. The bit-lines of the cells are connected to the inputs of a sense amplifier by means of N-channel switches. These switches are controlled by the column address decoder. The sense ampilifier will detect whether the state of the bit is a logic one or a logic zero depending on the relative polarity of the two bit-lines going into the differential sense amplifier. We usually call the transistors connected between the sense amplifier and bit-lines a data multiplexer or data selector.

When we wish to write the simple SRAM as shown in Figure 3, a row address line is selected by the row address decoder and the N-channel pass transistors (Q_2 of Figure 2) connected to the bit lines are turned on by pulling their gates high. Now however, the write amplifier driven by the Data-In line (Figure 3) is turned on by the write enable signal via the control logic. The write amplifier will drive one bit-line HIGH and the other bit-line LOW as determined by the logic state of the data input. The output of the write amplifier is more powerful than the inverter transistors (Q_1 in Figure 2) in the SRAM cell and it easily overpowers these inverter transistors if it is necessary to flip the static RAM bit. In its simplest form, this is all there is to the circuitry of a static RAM. Functions such as chip enable are used to simply enable or disable the entire operation of the SRAM. Output enable on a static RAM is used to turn "on" the outputs during a read cycle and turn "off" the outputs during write cycles. It can be used to solve timing problems in high-speed applications.

A variation on the standard four-transistor static RAM call is the six-transistor static RAM cell as shown in Figure 4. In this Figure we see that the two pull-up resistors (R_1 of Figure 2) have been replaced by two P-channel transistors. The operation of such a six-transistor cell is identical to the four-transistor cell previously described. The difference between the two approaches is that the physical size of the cell with the P-channel transistors is larger than the cell with the resistors. The standby power can

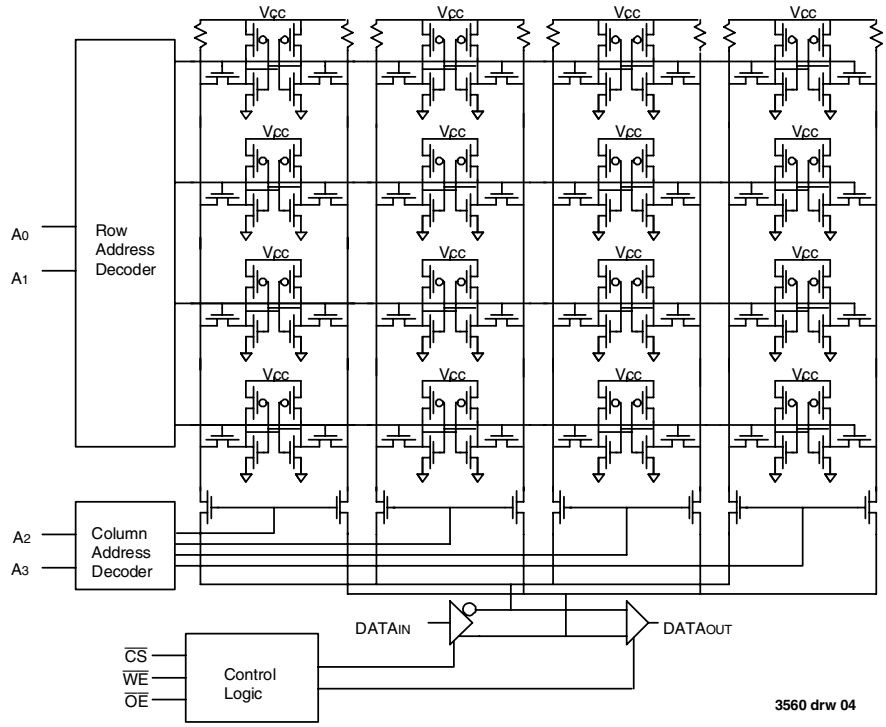

Figure 4. An Example of a Six-Transistor 16-bit SRAM.

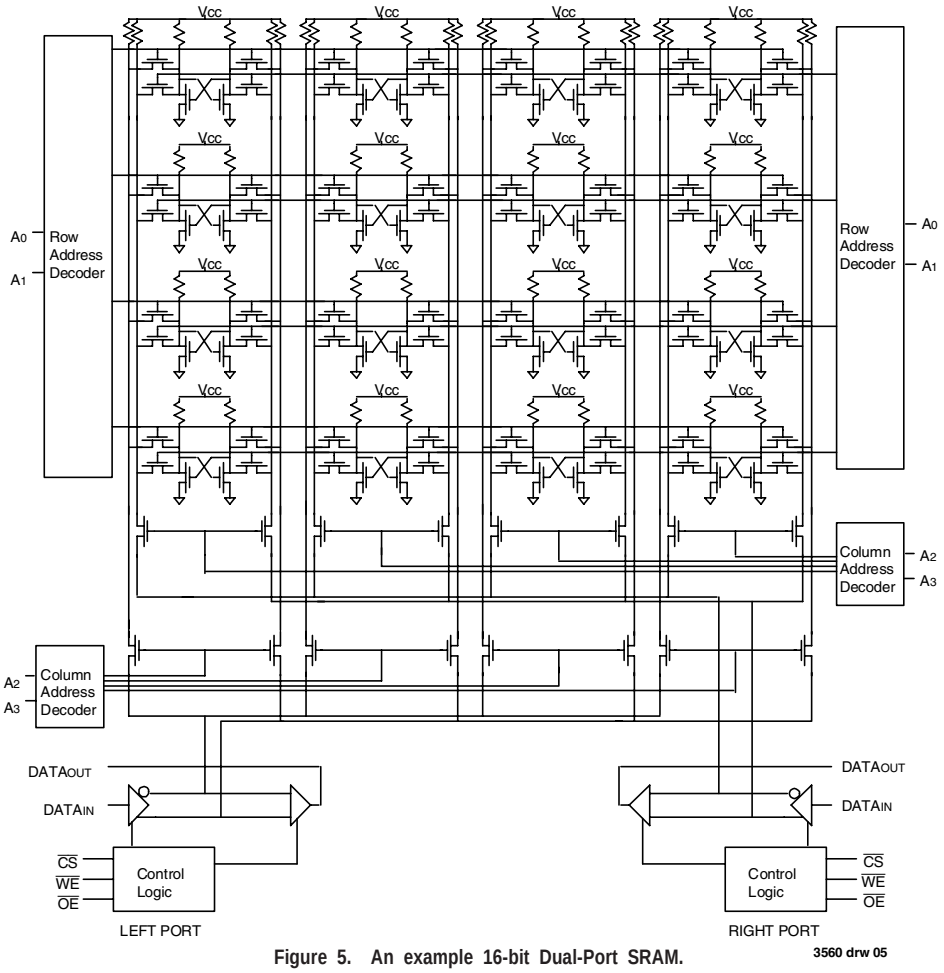

Figure 5. An example 16-bit Dual-Port SRAM.

be lower for the six-transistor cell because there is no power being dissipated. In a four-transistor cell, one of the the pull-up resistors is always dissipating power since one transistor of the cell is always ON. The six-transistor cell can have higher radiation hardened characteristics than the four-transistor cell because the voltage swings in the cell are larger. This is because the internal node in the cell that is high is pulled to the +5V rail by the P-channel transistor. In addition, the six-transistor cell provides higher internal noise margins in the circuit for this same reason. Most manufacturers of static RAMs use the four-transistor cell because it allows static RAMs of higher density to be fabricated with smaller die sizes.

Next, let's look at a typical dual-port SRAM such as the IDT7027, a 32K by 16-bit device. An example schematic diagram showing a sixteen-bit two-port SRAM is shown in Figure 5. Here we see our standard cross-coupled inverter pairs using two N-channel transistors with resistor pull-ups (Q_1 and R_1 of Figure 2) to form the sixteen memory bits. Notice however, now there are two pairs of N-channel transmission gates connected to each SRAM cell's true and compliment outputs and two pairs of bit-lines associated with each cell. Each pair of bit-lines is a read/write port into the dual-port SRAM. Each pair of transmission gates has its own row address control so that Port A can select any memory cell in the SRAM and Port B can select any memory cell in the SRAM. This is the technique used in IDT dual-port SRAMs to provide total independent access to individual bytes. Each pair of bit-lines is connected to a sense amplifier and a write buffer via a data multiplexer so that each port on the 2-port SRAM can read or write data at its selected address.

Now for the FourPort™ SRAM operation. Figure 6 shows a minimal schematic diagram for the IDT7052 12-transistor FourPort™ SRAM cell. The two inverters making up the basic memory cell are fabricated using two N-channel pulldown transistors and two P-channel pullup transistors. They are connected in the normal cross-coupled inverter fashion to make a single memory cell. Four individual memory ports are achieved by using four pairs of N-channel pass transistors to connect to four pairs of bit-lines. Four individual row addresses are used to select each pair of transmission gates connected between the SRAM cell outputs and the bit-line pairs. Four sense-amplifier/write-buffers are used to provide individual read/write paths from each port to all the cells in the SRAM.

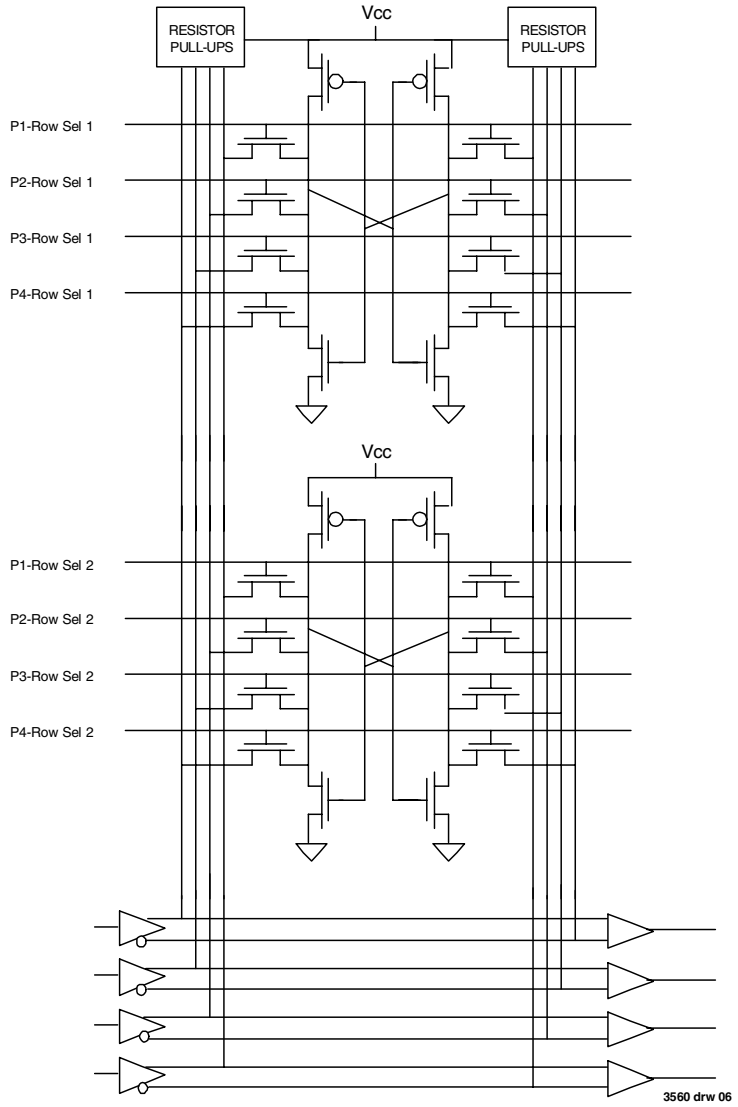

Figure 6. A Simple Example of a Twelve Transistor FourPort™ SRAM Configuration.

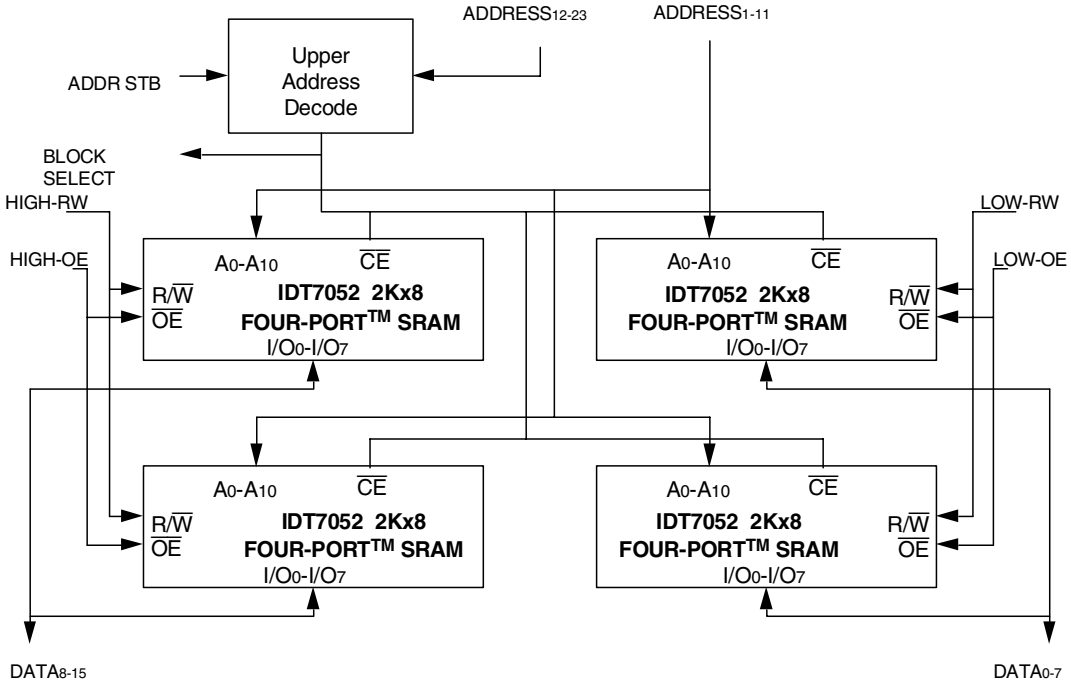

Figure 7. A 4K x 16-bit FourPort $\overline{CE}$ Controlled RAM

From this discussion, the design engineer should understand the mechanism used to implement a FourPort™ SRAM. As described, we can see how we can make each port of the FourPort™ SRAM totally independent from the other ports. Do not confuse this statement to mean that independent reads and writes can always be performed without data corruption. If two ports write to the same byte at the same time, one or both values may be lost. Likewise, if one port writes to a byte at the same time another part is reading the byte, the read may be corrupted even though the byte write is completed correctly. This application note does not discuss issues of data integrity in the case of multiple accesses to the same location, when one of the asynchronous accesses is a write cycle. These problems are discussed in detail in Application Note 02 and will not be further discussed here. Suffice it to say that the IDT7054 and IDT7052 FourPort™ SRAMs have a BUSY input to allow external hardware or software arbitration schemes to be implemented to meet the specific needs of the designer's system. The BUSY input serves only to block write cycles from the port to which this signal is applied. It has no effect on a read cycle. Note that in the following applications were not using the BUSY input of the FourPort™ SRAM so it should be tied HIGH. This will not be mentioned again, so please keep this in mind to avoid inadvertent write inhibits into the FourPort™ SRAM.

Once the rules are understood however, only engineering creativity is needed to visualize new architectural opportunities for FourPort™ SRAMs. This powerful new memory technology will provide increased performance in future electronic processing systems.

Cascading FourPort™ SRAM

Perhaps the most easily understood techniques in designing with static RAMs are width and depth expansion. Width expansion of the FourPort™ SRAM is straightforward. No additional parts are needed to build 16, 24 or 32 bit wide or wider memories. Any port of the FourPort™ SRAM can be viewed the same as a simple single port static RAM. All the same rules apply and they can be applied individually to each port of the FourPort™ SRAM.

Depth expansion of the FourPort™ SRAM is also quite simple. If one port is viewed as a static RAM, it is expanded similar to a single port device. Lower addresses are connected between devices and upper addresses are decoded by means of a standard decoder such as an IDT74FCT138 or IDT74FCT139. The outputs of the decoders can be used either to control the chip selects or control the write-enable and output-enable individually. Simple examples of expansion of one port of a FourPort™

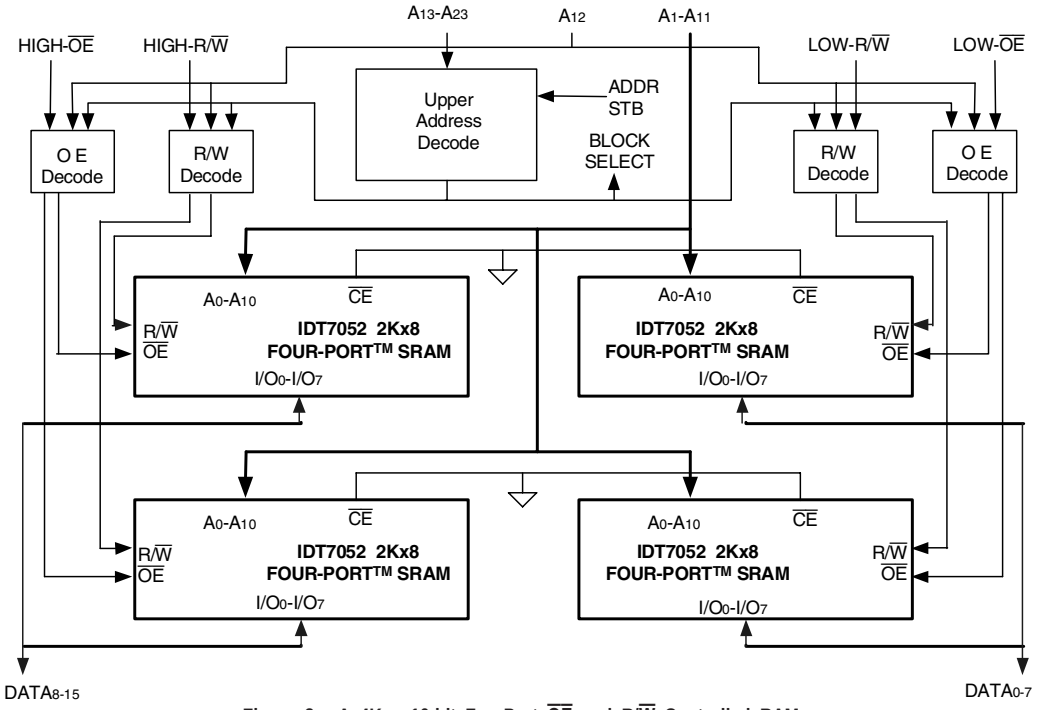

Figure 8. A 4K x 16-bit FourPort $\overline{OE}$ and R/$\overline{W}$ Controlled RAM

SRAM to a 4K-word by 16-bit configuration are shown in Figure 7 and Figure 8. Figure 7 shows the Chip Enable expansion method while Figure 8 shows write-enable, output-enable expansion. The two schemes are similar, but, sometimes one can have a timing advantage over the other. This is usually a function of the actual timing signals that are available or have already been generated.

Once the depth expansion is understood, we can view the CPU interconnect schemes by simply looking at a one deep FourPort™ SRAM. We recognize that deeper versions can be realized as just described.

Connecting the FourPort™ SRAM to CPUs a Z80A Example

Probably the easiest interface of the IDT7052 FourPort™ SRAM is to a Z80A. This processor still provides a great price-performance tradeoff! By using four Z80As with the IDT7052 FourPort™ SRAM, significant performance advantages can result. For example, no time need be lost due to DMA channels. The data placed in memory by one Z80A on one port is instantly available to another Z80A on another port. In a similar fashion, parallel processing can be performed by multiple processors working on the data in shared memory.

The typical connection scheme for the IDT7052 (or IDT7054 4Kx8 FourPort RAM) to a Z80A is shown in Figure 9. Here we see the eleven address lines, A0-A10, of the FourPort™ SRAM are connected to the A0-A10 lines of the Z80A. This places the FourPort™ SRAM in a contiguous 2K address space of the Z80A. The 2K byte segment actually used is determined by upper address decode circuit. A PAL or an IDT74FCT521

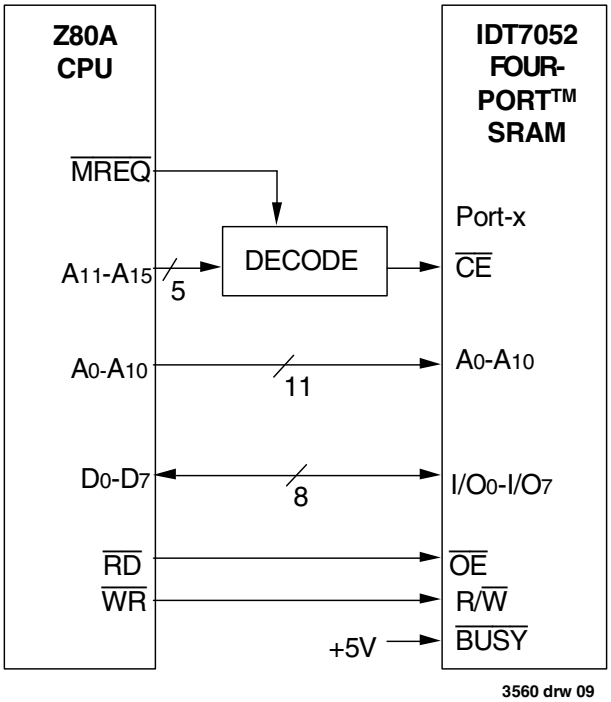

Figure 9. Interfacing the Z80A to One Port of the FourPort™ SRAM.

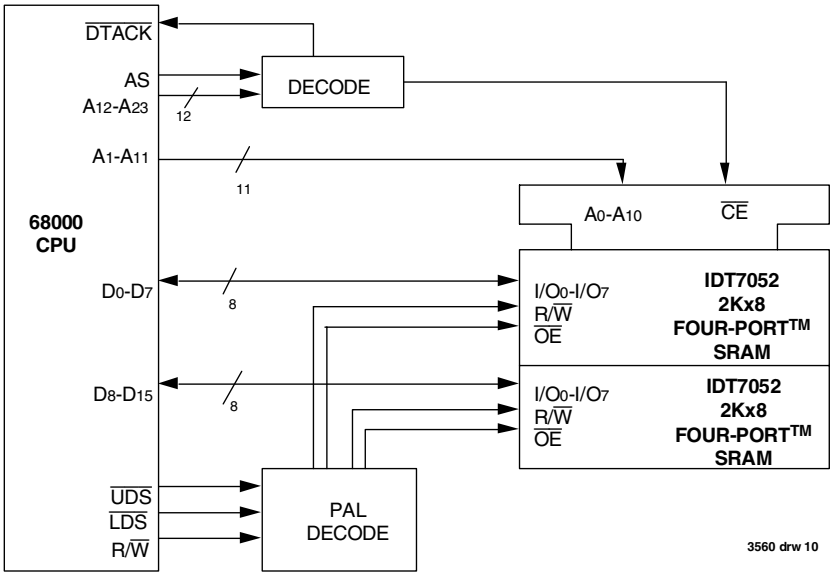

Figure 10. A 16-bit FourPort™ SRAM with the 68000 CPU.

could be used to perform this function. The data lines are connected between the processor and the SRAM. The Z80A has a $\overline{RD}$ line that can be connected to the FourPort™ $\overline{OE}$ and a $\overline{WR}$ line that can be connected to the FourPort™ R/W input. This works along the lines of the a Chip Enable expansion method just described. When the Z80A addresses the FourPort™ SRAM, either a read or write will be performed depending on the instruction being executed. If $\overline{RD} = V_{IL}$, the FourPort™ SRAM will output data from the addressed byte. If $\overline{WR} = V_{IL}$, the FourPort™ SRAM will write data into the addressed byte.

A 68000 Connection Example

If we wish to build a 16-bit microprocessor interface to one port of the IDT7052 FourPort™ SRAM, a typically interface might be as shown in Figure 10. Here we see two IDT7052s used in a 16-bit configuration. One FourPort™ SRAM is connected to the lower eight data bits (D_0-D_7) and the other FourPort™ SRAM is connected to the upper eight data bits (D_8-D_{15}). This completes a 16-bit data bus. Address lines A_0-A_{10} of the FourPort™ SRAM are connected between RAMs and also connected to address lines A_1-A_{11} respectively of the 68000. Remember, the 68000 does not have an A_0 address line but uses Upper-Data-Strobe ($\overline{UDS}$) and Lower-Data-Strobe ($\overline{LDS}$) to control the upper and lower byte selection. These two signals in conjunction with the R/$\overline{W}$ signal are decoded in a PAL to generate the individual FourPort™ SRAM R/$\overline{W}$ and $\overline{OE}$ control signals. Figure 11 shows the truth table needed for the PAL. It has been my experience when working with the 68000, that once these signals are generated, they are useful throughout the design to control other peripherals, etc. Basically, however, in this example we simply have a lower byte

INPUTS				OUTPUTS			
R/$\overline{W}$	$\overline{UDS}$	$\overline{LDS}$		$\overline{URW}$	$\overline{LRW}$	$\overline{UOE}$	$\overline{LOE}$
X	1	1		1	1	1	1
1	0	0		1	1	0	0
0	0	0		0	0	1	1
1	1	0		1	1	1	0
0	1	0		1	0	1	1
1	0	1		1	1	0	1
0	0	1		0	1	1	1

3560 tbl 01

Figure 11. 68000 16-bit Control PAL Truth Table.

FourPort™ SRAM and an upper byte FourPort™ SRAM.

The upper address lines of the 68000, A_{23}-A_{12} in this case, are used to position the 2K bytes of FourPort™ SRAM in continuous address space of the 68000. The actual location can be anywhere from 0x000000 to 0xFFFFFF as long as the overall range is on 2K byte boundaries. Usually we include address strobe ($\overline{AS}$) in the decoding as it can solve some timing problems. A timing review will show if it is needed. An output of the decode circuit can be used to generate the data acknowledge ($\overline{DTACK}$) if it is needed. Usually design engineers have an overall plan for generating the memory $\overline{CE}$s and $\overline{DTACK}$, so what is shown here is only to remind you of solving the overall problem.

How about 8-Bits, 16-Bits and 32-Bits in the Same System!!!

This is perhaps the most interesting example to talk about. We will use an 8-bit Z80A, a 16-bit 68000 and a 32-bit R3000 RISC microprocessor to discuss the design techniques. We have chosen the three processors because they are typical, they are fun to work with and they have had broad acceptance in the microprocessor world. First, let's look at Figure 12 to understand memory addressing and "memory space". All three of our selected microprocessors are "byte" addressable machines. That means they can address bytes as well as words in the case of the 68000 and R3000. The 68000 is a Big-Endian machine and the R3000 will be operated in Big-Endian mode to keep things simple. (DEC and Intel fans can make the appropriate transformation. In fact, the FourPort™ SRAM might make a really exciting byte-ordering problem solver between machines by connecting one port as Big-Endian and another port as Little-Endian to the same microprocessor and similarity for the second processor.)

Since we are talking about byte addressable machines, Figure 12

Address of Bytes in a Big-Endian 16-bit Word

Word Address	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
0x0024	0x0024	- etc -	- etc -	- etc -
0x0020	0x0020	0x0021	0x0022	0x0023
0x001c	0x001c	0x001d	0x001e	0x001f
0x0018	0x0018	0x0019	0x000a	0x000b
0x0014	0x0014	0x0015	0x0016	0x0017
0x0010	0x0010	0x0011	0x0012	0x0013
0x000c	0x000c	0x000d	0x000e	0x000f
0x0008	0x0008	0x0009	0x000a	0x000b
0x0004	0x0004	0x0005	0x0006	0x0007
0x0000	0x0000	0x0001	0x0002	0x0003

3560 tbl 02

Address of Bytes in a Big-Endian 16-bit Word

Word Address	Bits 15-8	Bits 7-0
0x0014	0x0014	- etc -
0x0012	0x0012	0x0013
0x0010	0x0010	0x0011
0x000e	0x000e	0x000f
0x000c	0x000c	0x000d
0x000a	0x000a	0x000b
0x0008	0x0008	0x0009
0x0006	0x0006	0x0007
0x0004	0x0004	0x0005
0x0002	0x0002	0x0003
0x0000	0x0000	0x0001

3560 tbl 03

Address of Bytes in a Big-Endian 8-bit Word

Word Address	Bits 7-0
- etc -	- etc -
0x0010	0x0010
0x000f	0x000f
0x000e	0x000e
0x000d	0x000d
0x000c	0x000c
0x000b	0x000b
0x000a	0x000a
0x0009	0x0009
0x0008	0x0008
0x0007	0x0007
0x0006	0x0006
0x0005	0x0005
0x0004	0x0004
0x0003	0x0003
0x0002	0x0002
0x0001	0x0001
0x0000	0x0000

3560 tbl 04

Figure 12. Memory Map for 8, 16, and 32-bit Ordering.

shows the byte addresses of an 8-bit machine, the byte addresses of a 16-bit machine and the byte addresses of a 32-bit machine. Likewise, word addresses of 16-bit and 32-bit machines are shown. What is intended here is to point out that we want the consecutive byte ordering of all of the machines to remain constant. By doing this, we keep the ability to do indexing into an array of bytes from any of the processors as a simple task. For example, a 40 byte index from any byte address is the same in all processors talking to each other through the FourPort™ SRAM. We can look at Figure 12 as representing the Z80A, 68000 and R3000 respectively.

Next, let's look at the interface needed for each of our three processors. We will build on our previous examples in this application note but there are differences needed to allow proper addressing. Let's begin by looking at the R3000. The reader should refer to IDT's wealth of information on the IDT79R3000 RISC microprocessor if you are not familiar with the standard CPU, FPA, Cache and I/O interface. We will use four of the IDT7052 FourPort RAMs to give a 32-bit wide memory for this example. We assume the first port is connected to the R3000, the second port is connected to the 68000 and the third port is connected to the Z80A. The fourth port could be connected to a second one of any of these processors or a wide selection of other things.

A typical R3000 interface is shown in Figure 13. The key element here is to understand that we are interfacing to a 32-bit data bus, 32-bit address bus with byte encoded control signals and to an R3000 interface. We are able to implement the required byte control per Figure 12 by using the "BYTE PAL" shown in Figure 13. The truth table for this PAL is shown in Figure 14. Using this decoding, we are able to do all of the required operations. This includes 32-bit word operations, 24-bit three-byte operations, 16-bit half-word operations and 8-bit byte operations. The signals available are A_0, A_1, AccessType0, and AccessType1, all from the R3000 address register, and we assume a R/W input from the "Control PAL" shown in Figure 13. There may be other options here, but this R/W signal must be realized in some fashion. The upper address bits from the R3000 are decoded in the fashion previously discussed to locate the total 8K bytes of FourPort™ SRAM in the R3000 address space.

Working out the timing of the R3000 interface is most of the work. Remember that at this interface point there are several flexibilities in the final timing. With an R3000 running at 16 MHz, a data transfer cycle is in multiples of 62.7 nanoseconds, 20 MHz gives 50 nanoseconds, and 25 MHz allows 40 nanoseconds. Thus depending on the processor speed and the FourPort™ SRAM speed selected block refill may or may not be desired. In any case, we should be able to run with zero, one or two stall cycles. As mentioned before, the design engineer usually has a plan for address decoding and control handshake which is more closely tied to the overall system design. From this standpoint, interfacing to one port of the FourPort™ SRAM is no different than interfacing to an EPROM, DRAM, SRAM, or peripheral.

Next, let's look at the 16-bit 68000 interface in a 32-bit memory system. A detailed block diagram is shown in Figure 15. The key thing to notice here is that four of the IDT7052 FourPort™ SRAMs are used. Notice that two of the devices are connected to the D_0-D_7 data bus and two of the devices are connected to the D_8-D_{15} databus on the 68000. Address line A_1 will be used to select which pair of FourPort™ SRAMs that the processor will read or write.

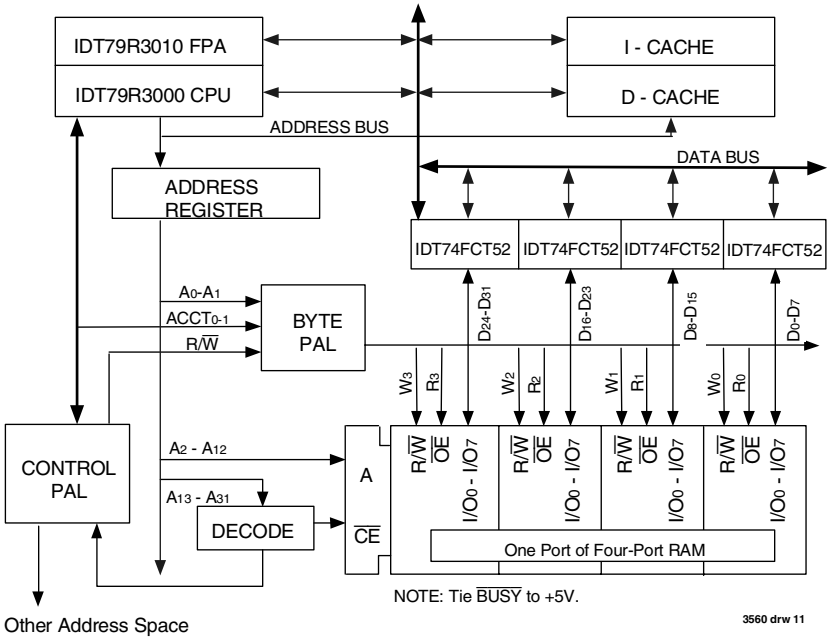

Figure 13. Using a 32-bit Wide FourPort Memory with the R3000.

INPUTS			OUTPUTS								COMMENTS
ACCT0	A1	A0	W3	W2	W1	W0	R3	R2	R1	R0	
1	0	0	1	1	1	1	0	0	0	0	Word Read
1	0	0	0	0	0	0	1	1	1	1	Word Write
0	0	0	1	1	1	1	0	0	0	0	Tri-Byte Read
0	0	0	1	1	1	1	1	0	0	0	Tri-Byte Read
0	0	1	0	0	0	1	1	1	1	1	Tri-Byte Write
0	0	1	1	0	0	0	1	1	1	1	Tri-Byte Write
1	0	0	1	1	1	1	0	0	1	1	Half-Word Read
1	0	0	1	1	1	1	1	1	0	0	Half-Word Read
1	1	0	0	0	1	1	1	1	1	1	Half-Word Write
1	1	0	1	1	0	0	1	1	1	1	Half-Word Write
0	0	0	1	1	1	1	0	1	1	1	Read Byte 0
0	0	1	1	1	1	1	1	0	1	1	Read Byte 1
0	1	0	1	1	1	1	1	1	0	1	Read Byte 2
0	1	1	1	1	1	1	1	1	1	0	Read Byte 3
0	0	0	0	1	1	1	1	1	1	1	Write Byte 0
0	0	1	1	0	1	1	1	1	1	1	Write Byte 1
0	1	0	1	1	0	1	1	1	1	1	Write Byte 2
0	1	1	1	1	1	0	1	1	1	1	Write Byte 3

3560 tbl 05

Figure 14. 32-bit R3000 Control PAL Truth Table (Big-Endian).

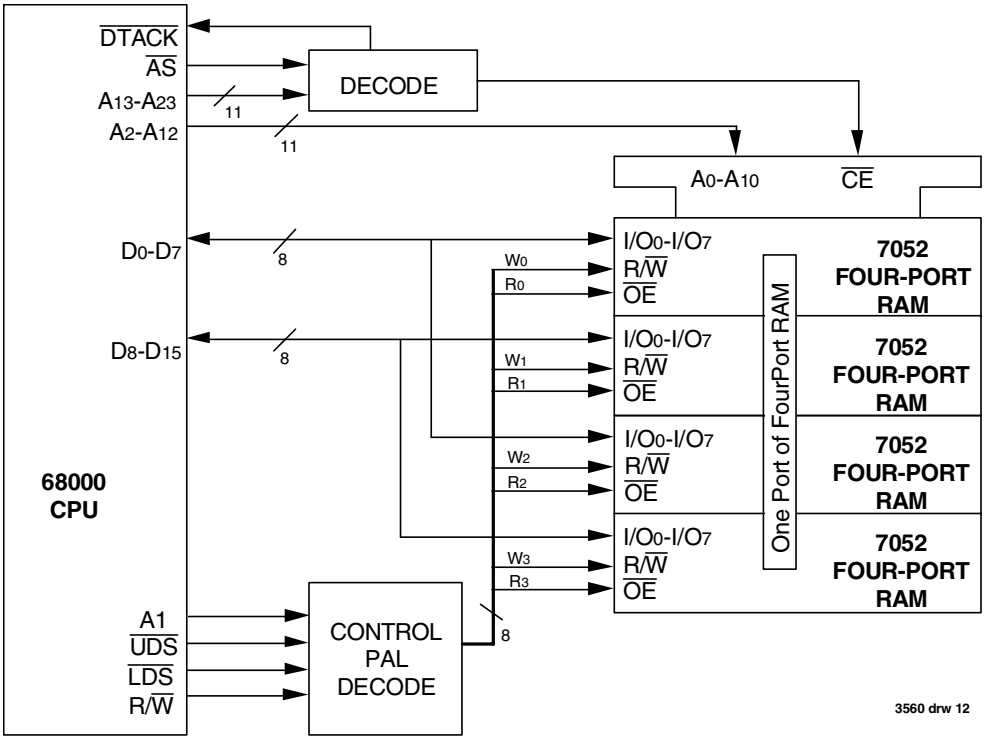

Figure 15. A 32-bit FourPort™ SRAM with the 68000 CPU.

INPUTS					OUTPUTS								COMMENTS
R/$\overline{W}$	A1	LDS	UDS		W3	W2	W1	W0	R3	R2	R1	R0	
1	0	0	0		1	1	1	1	0	0	1	1	Word Read
1	1	0	0		1	1	1	1	1	1	0	0	Word Read
0	0	0	0		0	0	1	1	1	1	1	1	Word Write
0	1	0	0		1	1	0	0	1	1	1	1	Word Write
1	0	1	0		1	1	1	1	0	1	1	1	Read Byte 0
1	0	0	1		1	1	1	1	1	0	1	1	Read Byte 1
1	0	1	0		1	1	1	1	1	1	0	1	Read Byte 2
1	0	0	1		1	1	1	1	1	1	1	0	Read Byte 3
0	1	1	0		0	1	1	1	1	1	1	1	Write Byte 0
0	1	0	1		1	0	1	1	1	1	1	1	Write Byte 1
0	1	1	0		1	1	0	1	1	1	1	1	Write Byte 2
0	1	0	1		1	1	1	0	1	1	1	1	Write Byte 3

3560 tbl 06

Figure 16. 32-bit 68000 Configuration Control PAL Truth Table.

For example, when A_1 is LOW, control signals W3, W2, R3 and R2 will be enabled. When A_1 is HIGH, control signals W1, W0, R1 and R0 will be enabled. This is shown in complete detail in the truth table of Figure 16. If we study this truth table, we see how we accomplish both 16-bit word (half-word) reads and writes as well as 8-bit byte reads and writes. All of this is consistent with the memory map shown in Figure 12. The technique here is actually to use A_1 to select either the lower half-word or the upper half-word in a 32-bit FourPort™ SRAM memory system. Every thing else about the design is the same as the previous 68000 example.

Lastly, let's look at the 8-bit interface to the Z80A microprocessor. It also should be viewed as being hooked into a 32-bit memory system. A detailed block diagram is shown in Figure 17. Notice that all four of the IDT7052 FourPort™ SRAMs are connected to the D_0-D_7 data bus. Address lines A_1 and A_0 will be used to select the device to which the Z80A processor

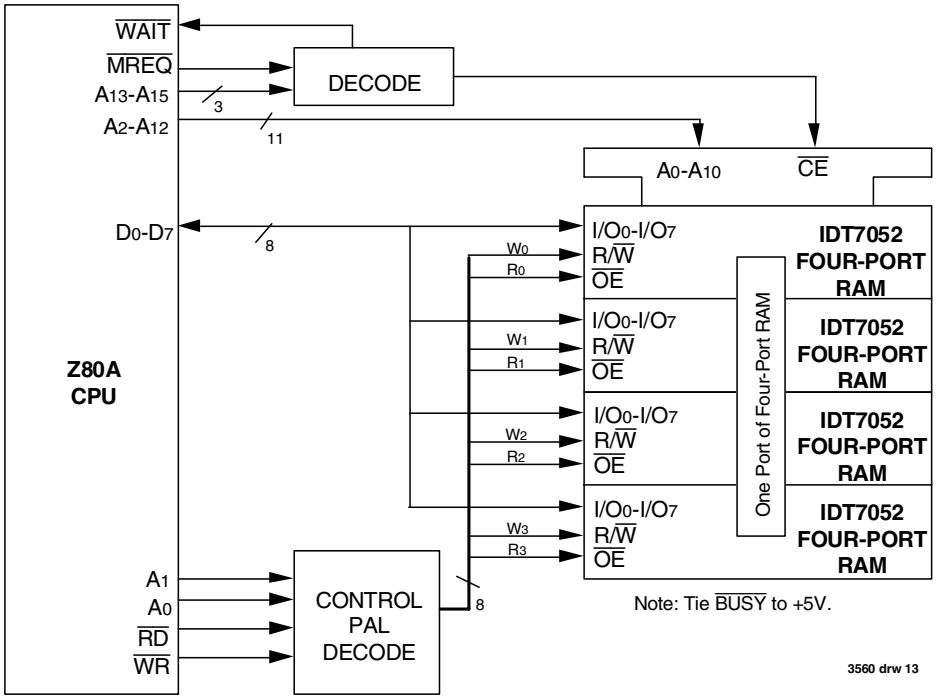

Figure 17. A 32-bit FourPort RAM with a Z80A CPU.

INPUTS				OUTPUTS								COMMENTS
$\overline{WR}$	$\overline{RD}$	A1	A0	W3	W2	W1	W0	R3	R2	R1	R0	
1	0	0	0	1	1	1	1	0	1	1	1	Read Byte 0
1	0	0	1	1	1	1	1	1	0	1	1	Read Byte 1
1	0	1	0	1	1	1	1	1	1	0	1	Read Byte 2
1	0	1	1	1	1	1	1	1	1	1	0	Read Byte 3
0	1	0	0	0	1	1	1	1	1	1	1	Write Byte 0
0	1	0	1	1	0	1	1	1	1	1	1	Write Byte 1
0	1	1	0	1	1	0	1	1	1	1	1	Write Byte 2
0	1	1	1	1	1	1	0	1	1	1	1	Write Byte 3

3560 tbl 07

Figure 18. 32-bit Z80A Control PAL Truth Table.

will talk. In fact, A_1, A_0, $\overline{RD}$ and $\overline{WR}$ are inputs to the control PAL decode. The truth table to be implemented is detailed in Figure 18. This processor is only capable of performing byte reads or writes s o the decoding is straightforward. A_1 and A_0 are used to do byte selection. Thus, the FourPort™ SRAM A_0-A_{10} address inputs are connect to the A_2-A_{12} address lines of the Z80A. This keeps the byte addressing as desired in the memory map of Figure 12. Again the remaining part of the design is as shown in the previous Z80A example.

The key in all of this discussion is to keep track of the data bus width being used in the design. Similarly, the decoding and processor address connections must take this into account. This is one point that the design engineer usually does not have to deal with when working with single port memories.

The purpose of this three processor example is to show a few interconnect schemes to typical microprocessors. From this discussion, the design engineer should be able to extend the concepts presented here to other 8-bit, 16-bit and 32-bit microprocessors. Just keep the techniques in mind and work out the desired memory mapping and timing.

System Design Ideas

Now that we have discussed how the the FourPort™ SRAM is built and we have a good idea of how to connect it to many processors, let's look at some system level uses for this type of FourPort™ SPAM.

Digital Signal Processing (DSP)

Digital signal processing applications have been expanding as new developments in semiconductor technology provide increased packing density and new architectures in integrated circuits. The IDT7052 and IDT7054 FourPort™ SRAM are another in the continuing growth of integrated circuits that allow design engineers to realize new system designs.

One of the simplest DSP algorithms that can be implemented is the finite-impulse-response (FIR) filter. In this type of algorithm, the impulse response of the filter has nonzero values only for a finite duration. These types of filters are easily implemented using only multiplication and summation. Figure 19 shows a block diagram of a DSP machine that takes advantage of the FourPort™ SRAM to interface to a multiplier-

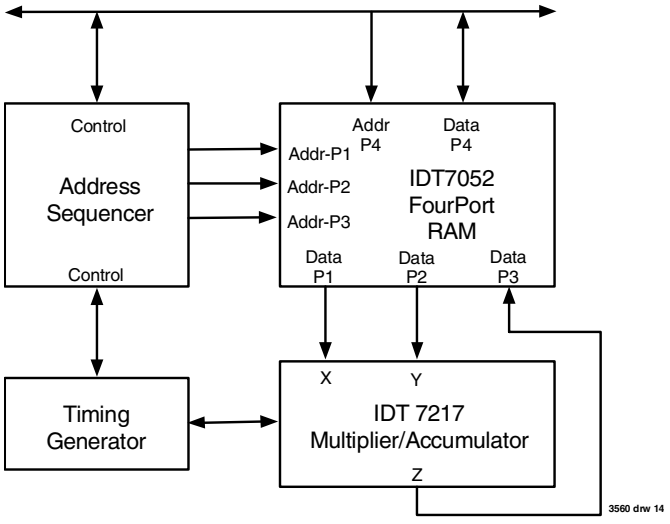

Figure 19. A Simple DSP Engine Using a FourPort RAM

accumulator (MAC) such as the IDT7210. In this example, two of the four ports of the FourPort™ SRAM are used to feed data to the MAC inputs and a third port of FourPort™ SRAM is used to receive completed results from the MAC output. The fourth port of the FourPort™ SRAM is connected to a local data-address bus to interface to the remainder of the system.

In the actual operation of such a processor as shown in Figure 19, data is loaded into the FourPort™ SRAM via Port 4. The algorithm usually needs coefficients and these are also loaded into the FourPort™ SRAM using Port 4. An address sequencer has the responsibility of providing the correct sequence of addresses to Ports 1, 2 and 3. This unit operates in conjunction with the timing generator to execute the algorithm. Let's look at an example. Suppose our algorithm is:

$$y(n) - A_0 * x(n) + A_1 * x(n-1) + A_2 * x(n-2) + A_3 * x(n-3)$$

We could read this as the current processed value is equal to the current sample times A_0, plus the first past sample times A_1, plus the second past sample times A_2, plus the third past sample times A_3. This already shows its potential as a FourPort™ SRAM application.

Now, we initialize our system at power-up by putting the values A_0, A_1, A_2, and A_3 into the FourPort™ SRAM. We would most likely clear the four locations for the data, then we start taking data. Each time we receive a data value, we can overwrite the fourth past sample. For each new sample, we will compute a new y(n) and put it into the FourPort™ SRAM. At some point we will extract the sequence of values for y(n) that we have computed. As can be seen, we can have several operations happening on the same clock cycle. SRAM ports 1 and 2 could be outputting data, SRAM port 3 could be inputting data, and SRAM port 4 could be inputting or outputting data, all simultaneously. The speed implications are obvious. Needless to say, this is a simple example for the purpose of demonstration. But if we wanted to work on 1024 samples with 128 coefficients and a more complex algorithm, all we have to do is follow the same methodology. See IDT application note 42 for a more detailed example of how to use this method to implement a matrix multiplication.

CPU to CPU to CPU to CPU

Referring to Figure 1 where we started this application note, we see that we have a FourPort™ SRAM connected between four CPUs. How do we efficiently communicate each processor's status to the other processors? You will need to work an acceptable software semaphore scheme or accomplish hardware handshaking using external circuitry. The most obvious software scheme is token passing. After each processor has determined its order in the token passing scheme, the token passing protocol boils down to each processor taking its turn. This can be achieved by reading one memory location to see who is master. Usually multiple reads and compares are performed to avoid any data corruption problems. A good example of this mechanism is detailed in IDT application note 43.

Let's take an example. The byte at address zero contains the token. The current value is one. CPU 1 is master of the FourPort™ SRAM and can read or write data. When finished, it writes a two at address zero. Every so often CPU 2 checks address zero and when it sees a two, it knows it is master. It performs any needed data reads or writes. When it is finished, it writes a three at address zero. CPU 3 writes a four and CPU 4 writes a one. Thus, a simple token passing scheme. "Fail safe" mechanisms can be implemented to keep the token moving if there is any failure.

Another obvious scheme is to set up a simple software semaphore path between each pair of processors. This technique can be used to pass data between processor pairs. The semaphore for each processor can use a different byte (or word) address for each semaphore in each direction. By using this method, many different software handshake techniques can be implemented. Rather than use a test and set instruction for semaphores in this application, another interlocking mechanism, like separate locations dedicated to the status of each processor, should be used to guarantee clean communications between tasks.

In addition, several hardware approaches are usually available in most multiprocessor environments. These include individual interrupts between processors as well as broadcast interrupt approaches. In either case, after the data has been set up in a private buffer, processor A can interrupt processor B to notify it of the pending message. The data structures used in such an environment can include pointer passing and linkage conventions consistent with modern day software techniques.

Summary

The FourPort™ SRAM is a truly new innovative integrated circuit memory that offers now communications methods for computing machines. It provides exceptional speeds because of its opportunity for parallelism. The IDT7052 2Kx 8-bit FourPort™ SRAM and the IDT7054 4K x 8-bit FourPort™ SRAM are the first in a series of memories that will pioneer these new architectural frontiers. At speeds as fast as some of the fastest standard static RAMs, they bring now performance dimensions to parallel communication between tasks of a computing machine. These devices utilize the latest in IDTs CMOS technology to provide the design engineer with an economical high performance, low power, small size and highly reliable "Speciality Memory" for todays performance-driven designs.

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 who are 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 to develop 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 from 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 of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

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-us/.

Trademarks

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