Byte-Order Issues for the NS32532

National Semiconductor
Application Note 591
George Grenley
July 1989

1.0 INTRODUCTION

There are two methods of numbering the bytes in a computer's memory in common use today. One method, known as ''Big Endian'', is derived from IBM® and is followed by Motorola in the 68000 family. The other method, known as ''Little Endian'', is derived from DEC and is followed by National in the Series 32000® family, and Intel in the 8086 family.

Although not usually a major problem, the differences between these two methods need to be considered when designing systems with processors of different types and when designing processors onto bus-based systems which use the opposite nomenclature for byte ordering, such as the National NS32532 on the VME bus.

2.0 TWO BYTE-ORDERING METHODS

As shown in *Figure 1*, when a 68000 references a word (16-bit), it uses the address which is numerically lower and the most significant byte (MSB). When it references a longword (32-bit), it uses the address which is numerically lowest and the most significant byte of the most significant word.

IBM/Motorola Method

D31 D24	D23 D16	D15 D08	D07 D00
Byte 0	Byte 1	Byte 2	Byte 3
Word 0		Word 1	
Longword 0			

FIGURE 1. ''Big Endian'' Byte Ordering

The VME specification follows this addressing method. Furthermore, the VME specification is quite explicit about which data lines should be active when a given byte, word, or long word is being referenced. Specifically, for byte and word transfers, even bytes are transferred on D8–D15, and odd bytes are transferred on D0-D7. For longword transfers, byte 0 is transferred on D24–D31, byte 1 is transferred on D16–D23, byte 2 on D8–D15, and byte 3 on D0–D7.

DEC/Intel/National Method

D31 D24	D23 D16	D15 D08	D07 D00
Byte 3	Byte 2	Byte 1	Byte 0
Word 1		Word 0	
Longword 0			

FIGURE 2. ''Little Endian'' Byte Ordering

This byte-ordering method is different than the system used by NSC in the Series 32000 family (see *Figure 2*). Specifically, IBM and VME are ''backwards'' with respect to how byte addresses are associated with data lines, compared to the NSC method. Word and longword addressing is similar, however, in that a word or longword is referenced by the numerically-lowest address contained therein. The difference is that these numerically lower addresses point to the least significant byte of the word or longword, instead of the most significant, as in the IBM/VME case. To summarize, both ''Big Endian'' and ''Little Endian'' byte-ordering schemes:

1. refer to a word or longword by the numerically lowest address within the word.
2. define D31 as the MSB (2^{31}), and D0 as the least significant byte LSB (2^{0}).
3. store text strings, in order, in numerically ascending locations in memory.

The only difference between these two methods is the numbering of bytes within the word (i.e., the assignment of byte addresses to data bus bits). Series 32000 devices define D0–D7 as byte 0, IBM/Motorola devices define D0–D7 as byte 3.

3.0 BYTE-ORDERING SOLUTIONS

There is no way to hook a Series 32000 processor to VME bus without creating certain compatibility problems. There are two ways to connect the processor, as shown in *Figures 3* and *4.* Note that both systems are internally consistent, since the data written by the processor in either case will be read back the same way. The compatibility arises only when the processor (such as an NS32532) transfers data to a slave unit (such as a 68000 on a peripheral board).

3.1 CORRECT BIT ALIGNMENT

Figure 3 is the most obvious solution. Suppose that the NS32532 is to read a 32-bit integer $0123ABCD stored at longword address 0. When the NS32532 reads the word, the device will correctly read $0123ABCD, with 0 as the most significant bit, and D as the least. Similarly, if the NS32532 passes the 32-bit pointer to a 68000 family slave processor, the slave will read it in the correct order.

However, suppose the 68000 family slave then stores the text string ''Unix'' beginning at location 0. The 68000 will put the first character at its byte 0, the next at byte 1, etc. When the NS32532 reads this string, it will see ''x'' at location 0, ''i'' at location 1, ''n'' at location 2, and ''U'' at location 3. This is, of course, backwards. The NS32532 will then have to swap the bytes to get the correct order. This method (shown in *Figure 3*) is called correct bit alignment. It preserves the bit position relationship.

RRD-B30M75/Printed in U. S. A.

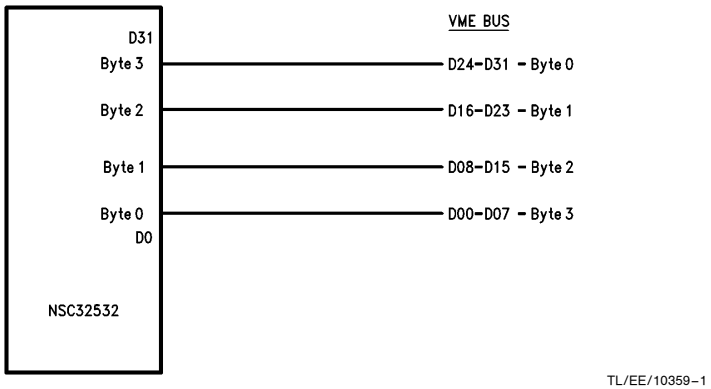

FIGURE 3. Correct Bit Alignment Method

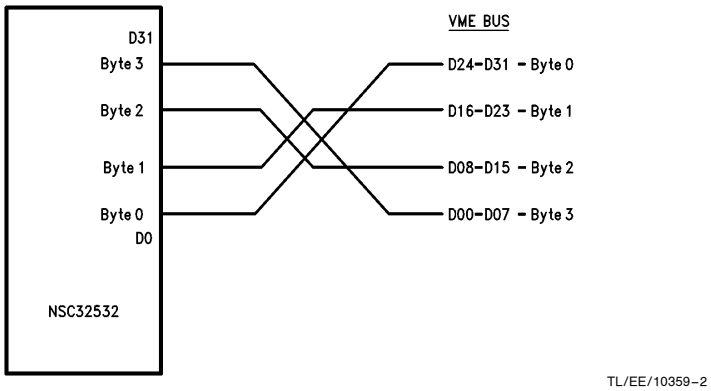

FIGURE 4. Correct Byte Alignment Method

3.2 CORRECT BYTE ALIGNMENT

The byte-swapped arrangement shown in *Figure 4* solves the problem of mis-ordered byte strings. When the 68000 slave stores the string ''Unix'' as above, the NS32532 will now read it correctly. However, this method fails when reading words and longwords. Using the example from above, the 32-bit integer is read back as $CDAB2301. This method is called correct byte alignment because it preserves the byte-sequence that strings are stored in memory. Note that in both cases, the order of bits within a byte is the same.

3.3 CHOOSING THE BEST SOLUTION

A dynamic byte-ordering mechanism would solve the dilemma caused by the two methods. Obviously, if one knew that integers were being read, the data path in *Figure 3* would be selected. If it were text, *Figure 4* would be selected. The problem is, there is no way for the hardware to know this—the size of the data transfer does not indicate the type of data being transferred.

VME boards with software-selectable byte swapping have been built. However, these have some major pitfalls. First, if one sets the swap bit, any further fetches of code from off-board memory will be garbled. In a system where caches are doing this independent of programmer control, there are major problems associated with keeping things under control. This can be partly solved by restricting swapped transfers to a certain address range, but the additional logic required to support this requires a lot of real-estate.

The second pitfall is that there is no agreed-upon convention among vendors of VME boards based 80x86 and Series 32000 family processors regarding when to swap (i.e., to stay compatible with VME specifications) and when it is okay not to swap (and thereby deviate from specifications). In short, any programmable or dynamic byte-swapping approach is likely to produce problems and system crashes, and is definitely not in compliance with VME specifications.

So, designers are faced with the necessity of making a choice, where neither option is ideal. There are, however, good reasons for picking the correct byte alignment method. The first reason is compatibility with VME addressing standard. VME states that when byte 0 is referenced, D24–D31 shall be active for 32-bit transfers, and D8–D15 for 16-bit transfers, etc.

The second reason is compatibility with virtually all I/O devices. The byte-ordering problem first became a major issue when computer networks became common and designers began hooking DEC equipment to IBM. As network protocols grew in complexity, network designers had to solve compatibility problems. Designers solve these problems by specifying all transfers as streams of byte oriented information. Any larger unit of data is defined in terms of the arrangement of bytes within it. For example, if the net-

work sends a 32-bit address, it might send four bytes, where byte 3 is defined as the most significant, byte 2 and 1 in the middle, and byte 0 the least significant. It is the responsibility of the processor interpreting this data to do so in accordance with the specification.

Using this methodology means that I/O control blocks (and other inter-processor data structures) can be specified in a machine-independent way. Each byte in the structure is assigned an address by VME, and referred to in all documentation by its byte address. Other structures are built up based on these addresses. This leaves it up to the programmer who is writing for a specific processor to be aware of its internal ordering, and to program accordingly. Note, however, that macros can be defined to handle the ordering for a specific processor, thus making it transparent to the programmer.

This means that when communicating to an I/O device, an NS32532 program would have to swap bytes on some control information, but not the data being transferred. For instance, in the case of a disk controller, the initialization parameters and address pointers would have to be swapped, but the blocks of data coming off the disk would not be. Since there is more data than control information, this approach is more efficient.

Because I/O devices handle text data as a byte ordered stream, whether it is disk, tape, or data communications, preserving correct byte alignment also means that tapes and disks holding text information will move from machine to machine without problems. To be able to write a backup tape on one type of machine and be able to read it properly onto another is a great advantage. One can also move binary executable files between NS32532-VME and NS32532-MULTIBUS® machines without problems; no byte-swapping is necessary. Similarly, PROMs from one type of machine will plug in and work properly on the other type by simply plugging them into the same numbered byte socket—0 into 0, etc.

4.0 SUMMARY

In summary, two methods can be used to solve byte ordering differences between Series 32000 processors and IBM/Motorola devices. The following points illustrate these two solutions and the effects of choosing one over the other.

1. Either method can be used for NS32532 systems since the processor will read back what it wrote.
2. There is no single way to hook an NS32532 processor to the VME bus without creating certain compatibility problems. Two possible solutions are correct bit alignment and correct byte alignment.
   a. Correct Bit Alignment (See *Figure 3*), is integer compatible but reverses text strings.
   b. Correct Byte Alignment (See *Figure 4*), is text compatible but reverses bytes within integers.
3. Software transparent dynamic byte swapping is not possible, because there is no way for hardware to know what type of data is being transferred, and therefore it cannot know whether to swap bytes.
4. Using the correct byte alignment method, the NS32532 reverses the order of bytes, giving compatibility with the VME world. Full text transfer capability (which is the majority of the inter-system communication) is also achieved without penalty. This means, for example, that a tape containing C language source files written on a 68000-based VME machine, or an NS32532 based MULTIBUS machine, will be read correctly on an NS32532-based VME machine, without requiring software to swap bytes. The only requirement involves I/O control blocks; a programmer must always byte-swap all 16- or 32-bit data types.

National recommends that the correct byte alignment method be used on all Series 32000-based VME products, and further recommends that the swapping be done directly at the processor's data lines. This way, the entire memory and I/O structure of the system is consistent, regardless of whether the memory and I/O is on or off card.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998