Freescale Semiconductor
Application Note

AN2285
Rev. 2.2, 3/2008

Data Movement Between Big-Endian and Little-Endian Devices

by *Kyle Aubrey, Field Technical Leader*
Freescale Semiconductor, Inc.
San Jose, CA

Ashan Kabir, System Engineering
Freescale Semiconductor, Inc.
Austin, TX

This document addresses problems programmers and designers may encounter when working with big-endian Power Architecture™ processors in conjunction with other devices that are little-endian. The problems arise because a big-endian (BE) system stores the most significant byte (MSB) of the data at the starting (or lowest) address, but a little-endian (LE) system stores the least significant byte (LSB) of the data at the starting address. Examples and explanations of data versus address invariance policies are provided throughout this document. The terms ‘PCI bus’ and ‘agent’ are used synonymously because an agent add-in card (a little-endian agent is considered in this application note) and the PCI bus of a motherboard into which the agent is inserted are both little-endian, so data appears identically on both.

Contents

1 CPU Bus Versus PCI Bus Data Interpretation 2
2 Data Movement: Big-Endian Host to Little-Endian Agent 3
3 Address and Data Invariance 4
4 Hardware Byte Swapper 5
5 Conclusion 8
6 Revision History 8

freescale™ semiconductor

1 CPU Bus Versus PCI Bus Data Interpretation

Figure 1 shows data on the CPU bus (big-endian) and on the PCI bus (little-endian).

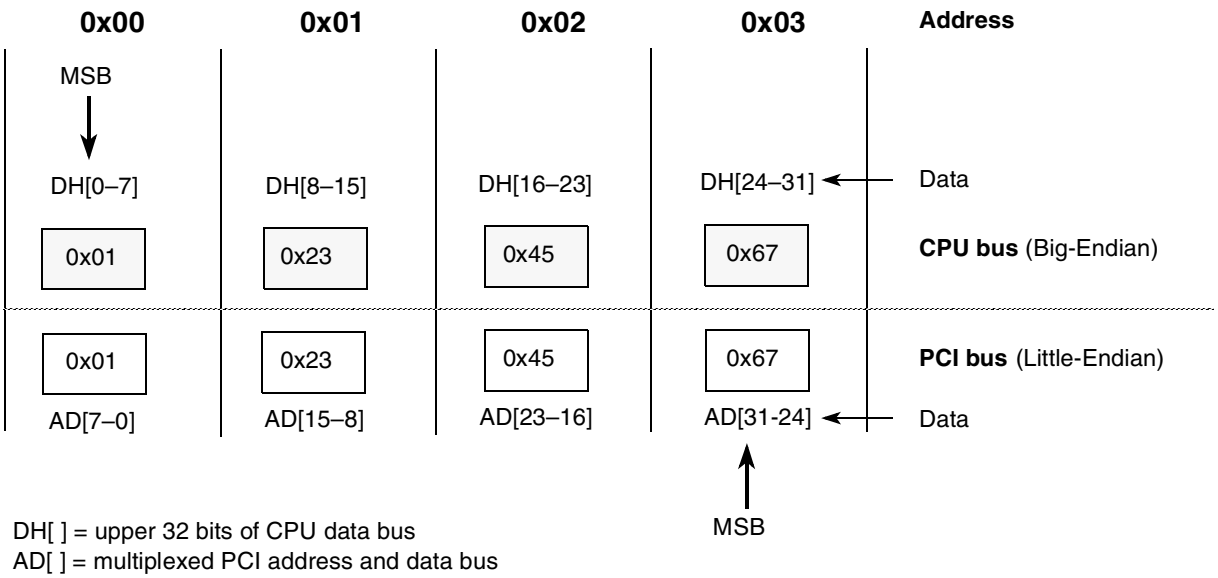


Figure 1. Data on Processor Bus and PCI Bus

As Figure 1 shows, the endianness of a bus determines whether the MSB is put into the lowest address (big-endian) or in the highest address (little-endian). Also, note that the PCI bus AD[31–0] signals are shown as discontinuous (that is, AD[7–0], AD[15–8], AD[23–16], and AD[31–24]) because of the bit ordering required by the PCI specification. Regardless of a device's endianness, data is interpreted from MSB to LSB. As shown in Figure 2 and Figure 3, this results in different interpretations of the data represented in Figure 1.

Figure 2. Interpretation of Processor Data on the CPU Bus (MSB to LSB)

Note that the interpretation of the data on the CPU bus appears reversed on the PCI bus.

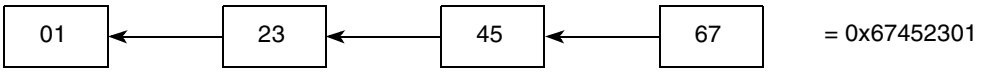


Figure 3. Interpretation of Data on the PCI Bus (MSB to LSB)

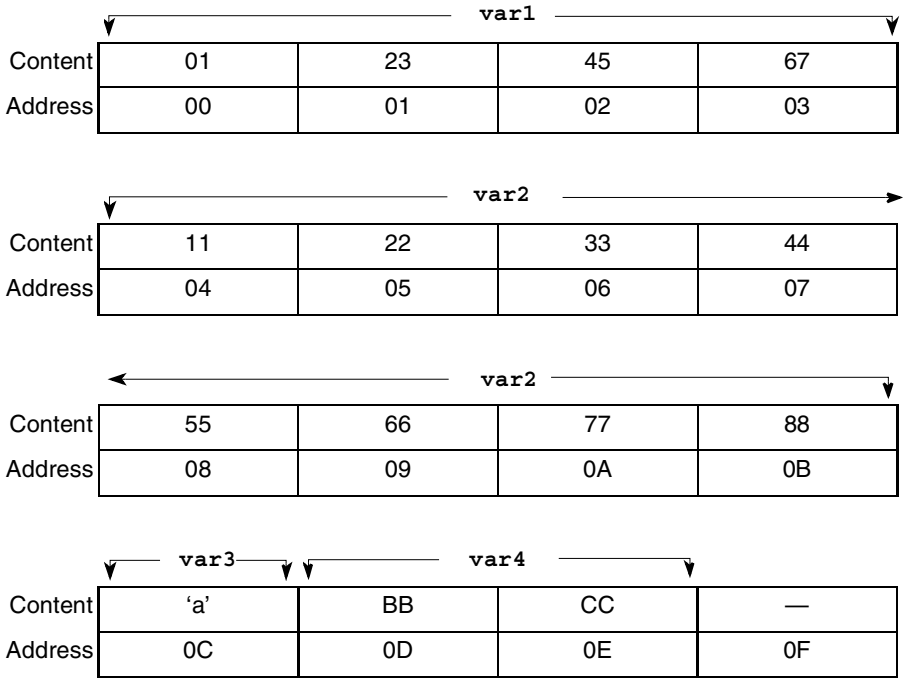
2 Data Movement: Big-Endian Host to Little-Endian Agent

Consider a host system in which the host is a big-endian Freescale processor with an integrated PCI bridge that allows data movement from the processor bus to a PCI bus. The following initialized data structure is placed into the memory of the big-endian host. Note that the `__packed__` switch is used to ensure that there is no inter-member padding and that natural alignment is not enforced. Because of the `__packed__` switch, `var2` resides in memory immediately following `var1`, `var3` immediately follows `var2`, and so on.

```
typedef __packed__ struct {
unsigned long          var1; //4Bytes; var1=0x01234567
unsigned long long     var2; //8Bytes; var2=0x1122334455667788
unsigned char          var3; //1Byte;  var3='a'
unsigned short         var4; //2Bytes; var4=0xBBCC
}FRAME;
```

Table 1 shows how the data structure FRAME appears on the processor bus of the host device.

Table 1. Example Data Structure FRAME on Processor Bus (Big-Endian)—No Padding



2.1 Big-Endian Host Interpretation of Data

Because data is interpreted from MSB to LSB, the values of the variables from the example data structure shown in Table 1 are as follows:

```
var1 = 0x01234567
var2 = 0x1122334455667788
```

```
var3 = 'a'

var4 = 0xBBCC
```

2.2 Little-Endian Agent Interpretation of Data

If data is transferred from its location on the host to the same location on the agent (data located in 0x0 of the host bus is transferred to 0x0 of the agent and so on), the data content of the agent memory appears as shown in Table 1. This is the default policy for dealing with endianness conversion, which is called address invariance, and it is defined later in this document. However, because the agent is a little-endian system, the interpretation of the data changes—data in the lowest location—is treated as the LSB. Therefore, the interpretation of the data in the agent little-endian system is as follows:

```
var1 = 0x67452301 (Reversed. See Figure 1.)

var2 = 0x8877665544332211 (Reversed. See Figure 1.)

var3 = 'a' (Restored. endianness concept does not apply for a single byte.)

var4 = 0xCCBB (Reversed. See Figure 1.)
```

In summary, the interpretation of data transferred from a big-endian host to a little-endian agent is sometimes reversed (`var1`, `var2`, and `var4`) and sometimes restored (`var3`).

2.3 Restoration of Reversed Data

Without the aid of a byte swapper that can interpret the software data structure, software must restore the reversed data. Because the interpretation of `var1`, `var2`, and `var4` in the little-endian agent is reversed, software must first byte-swap the value of `var1` obtained from the big-endian memory before transferred data can be read, as follows:

```
Temp = Byte_Swap_Long (var1);
```

`Byte_Swap_Long()` in this case takes a long value, swaps its byte locations, and returns the byte-swapped value. After the routine is called, `Temp` holds the true interpretation of `var1` in a little-endian system, which is 0x01234567. This process can then be repeated for subsequent variables.

3 Address and Data Invariance

Invariance is defined as not varying, constant, or preserved. There are two approaches to endian mapping: address invariance and data invariance. Address invariance preserves the addressing of the byte in memory. Data invariance preserves the relative byte significance but translates the byte addressing.

3.1 Address Invariance

The default policy for dealing with endianness conversion in Freescale Power Architecture processors is the policy of address invariance. This policy preserves the byte address of each byte on a particular I/O interface as it is placed into main memory or moved to Power ISA registers by swapping the byte-lanes. As described in Section 2.2, “Little-Endian Agent Interpretation of Data,” this policy can have the effect of reversing the order of significance of bytes (most significant to least significant) or a particular datum, but it has the benefit of preserving the format of the general data structures so that software that is aware of both endianness and data structure format can properly access the data structure.

3.2 Data Invariance

An alternative policy is data invariance. This policy preserves the exact value of data by swapping the byte addresses of each byte within the datum. Although this preserves values for a given size of datum (usually 4 bytes), it has the undesirable effect of scrambling the addresses and formats of the datum or data structures of other sizes. For instance with data invariance, byte 0 is still the most significant byte in the data structure but is now located at address 0x03 in memory rather than address 0x00.

The integration of a data invariance mode into a PCI bridge (as well as PCI-Express or HyperTransport bridge) is referred to as a "Hardware Byte Swapper."

4 Hardware Byte Swapper

This section describes the hardware byte swapper, or data invariance mode, built into certain bridges.

4.1 Common Misconception

There is a common misconception that if the PCI block has a byte swapper operating on-the-fly, the interpretation of a software data structure can be restored without any software intervention in the destination. This idea is false because the byte swapper of a PCI block can swap bytes only on aligned boundaries (for example, a 4-byte word boundary). When the byte swapper is turned on, it takes a word from the source memory, byte-swaps the word, and places the word into the PCI interface. The hardware byte swapper cannot interpret a data structure containing different types of variables, such as byte, half word, word, and doubleword.

4.2 Byte Swapper Usage

Table 1 in Section 2, "Data Movement: Big-Endian Host to Little-Endian Agent," shows the data content of the memory of a big-endian host. If the host device has a byte swapper in its PCI hardware block and if the byte swapper is turned on when the data moves from the host memory to the PCI bus, the byte swapper takes the first row in Table 1, byte-swaps the content, and places the swapped data on the PCI bus (little-endian). Then it repeats this process for rows 2, 3, and 4. Eventually, the data on the PCI bus appears as shown in Table 2.

Table 2. Example Data Structure Frame on PCI Bus (Little-Endian)

Table 3. with Byte Swapper Turned On

Content	01	23	45	67
Address	03	02	01	00

Content	11	22	33	44
Address	07	06	05	04

Table 2. Example Data Structure Frame on PCI Bus (Little-Endian)

Table 3. with Byte Swapper Turned On (continued)

Content	55	66	77	88
Address	0B	0A	09	08

Content	‘a’	BB	CC	—
Address	0F	0E	0D	0C

Based on Table 2, Table 4 shows the interpretation of `var1`, `var2`, `var3`, and `var4`.

Table 4. Interpretation of Byte Swapped Data on a Little-Endian Agent

Variable	Size	LSB	MSB	Interpretation [1]	Notes
`var1`	4 bytes	0x0	0x03	0x01234567	Because the byte swapper is turned on when data is transferred from a big-endian host to a a little-endian agent, **var1 is restored** and there is no need for software intervention.
`var2`	8 bytes	0x04	0x0B	0x5566778811223344	This is not the original interpretation of var2 in the big-endian host (see Table 1). Even though the byte swapper is turned on when data is transferred from a big-endian host to a little-endian agent, the data interpretation of **var2 is not restored**.
`var3`	1 byte	0x0F	0x0F	—	Endianness does not affect data that is only a single byte wide, so the data interpretation of **var3 is restored**. However, software should be aware that the byte swapper has moved **var3** from address 0x0C to address 0x0F.
`var4`	2 bytes	0x0D	0x0E	0xBBCC	Because the byte swapper is turned on when data is transferred from a big-endian host to a little-endian agent, **var4 is restored** and there is no need for software intervention. Note that the value for **var4** is restored only because it straddles the 2-byte (halfword) boundary. If **`var4`** were at big-endian address 0x0C, it would be byte swapped to the little-endian address of 0x0E.

[1] Interpretation of the data is always from MSB to LSB, regardless of the endianness.

Figure 4 shows an example of a 4-byte write to PCI memory when the hardware byte swapper (also known as data invariance) is enabled.

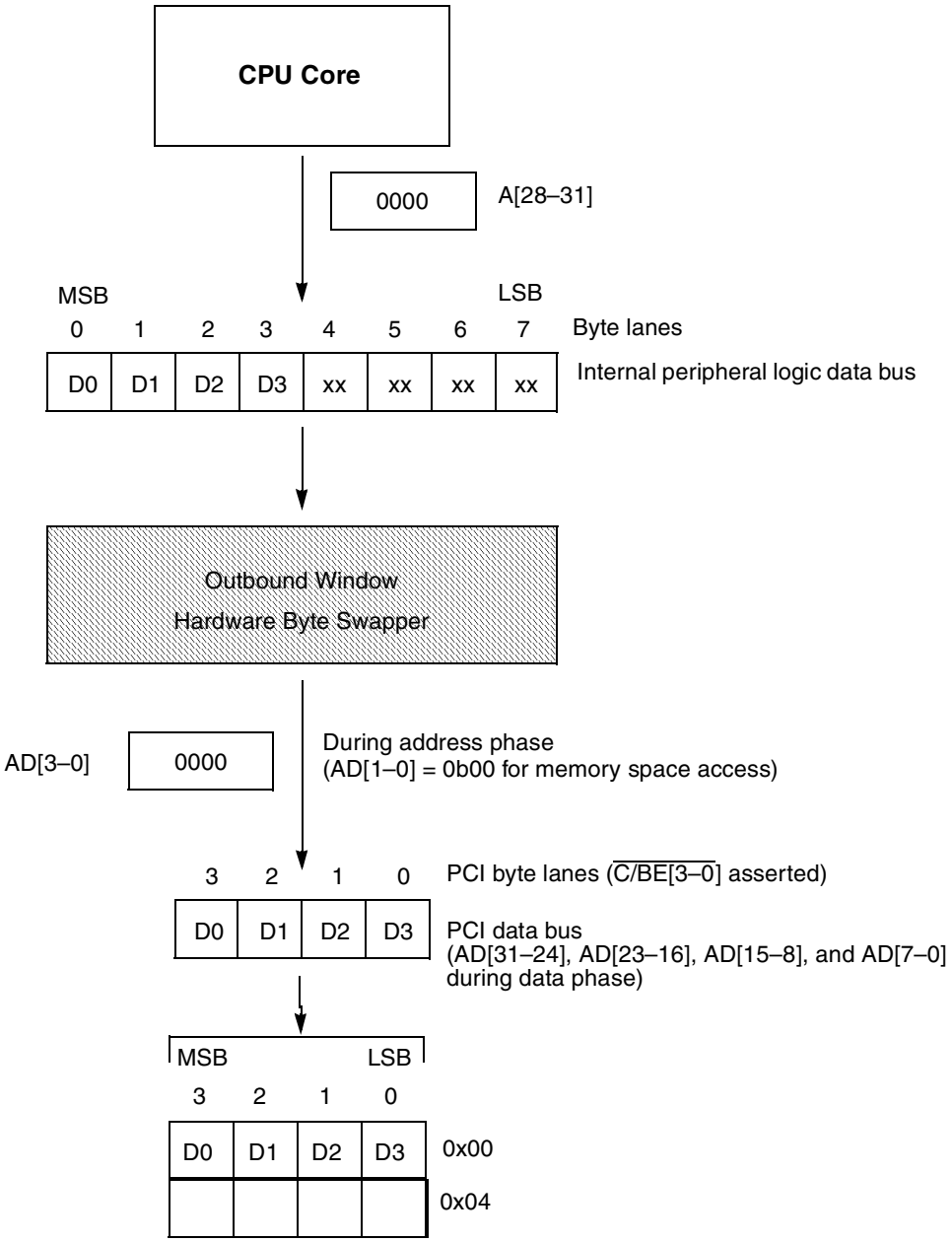


Figure 4. Four-Byte Transfer to PCI Memory Space With Hardware Byte Swapper Enabled

Note that when the byte swapper is enabled, the MSB on the internal peripheral logic bus, D0, is placed on byte lane 3(AD[31–24]) on the PCI bus. This occurs so D0 appears at address 0x*nnnn_nn*03 and not at address 0x*nnnn_nn*00 in the PCI space.

5 Conclusion

This application note demonstrates that when multiple data types occur in a single data structure, the use of a byte swapper does not maintain data interpretation across devices with different endian characteristics. In an ideal case, when all variables in the data structure are words, turning on the byte swapper restores data interpretation. Therefore, when the endianness of the host and agent differs, a software solution at the destination end is the most efficient solution to restore data interpretation.

6 Revision History

Table 5 provides a revision history of this application note.

Table 5. Revision History

Revision	Date	Substantive Changes
2.2	3/2008	Converted PowerPC to Power Architecture Minor formatting changes
2.1	11/2007	Put trademark after first mention of PowerQUICC II Pro and PowerQUICC III. Minor formatting changes
2	6/2006	Overhaul of most sections. Added section on invariance Various formatting Internal Release
1	5/2006	Formatting Changes Altered Fallacy Section to cover the Byte Swapper Feature more in-depth Added Figure 4 Added Section 7and 8

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
1-800-521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN2285
Rev. 2.2
3/2008