Endianness and ARM® Processors for 32-bit

Overview

The terms little-endian and big-endian refer to the way in which a word of data is stored into sequential bytes of memory. The first byte of a sequence may store either as the least significant byte of a word (little-endian) or as the most significant byte of a word (big-endian). Typically, this is a detail that is of no relevance to the software engineer. In certain cases, however, the software engineer must take the endianness of the hardware into account.

Note: The majority of this application note is accurate only for ARM V4 and ARM V5 cores. In 2006, with the introduction of ARMv6 (ARM11), facilities were added for more flexibility in dealing with endianness issues. This article is still very useful in understanding the implications of endianness and the relationship between what the core does and the peripheral circuitry. When working with more recent cores such as the whole v7 family (Cortex) it should be remembered that there are instructions and monitors that can work around some of these issues in some circumstances. This is noted later. This note does not address 64-bit cores (ARMv8 architecture).

ARM® processors (v7 and earlier) use 32-bit words[1], and if the software accesses all data as 32-bit words, the issue of endianness is not relevant. However, if the software executes instructions that operate on data 8 or 16 bits at a time, and that data must be mapped at specific memory addresses (such as with memory-mapped I/O), then the issue of endianness will arise. The endianness of the system as a whole is implementation dependent and is determined by the circuitry that connects the processor to its peripheral components.

Programs that will execute directly from flash memory must be stored into flash in the correct endian format so that the code will be fetched correctly by the processor. For this reason, the Target Flash Programming feature of SourcePoint provides the user with the option to reverse the endianness of the flash code before programming it into a flash device.

What Is Endianness?

Endianness refers to how bytes and 16-bit half-words map to 32-bit words in the system memory. Given that a 32-bit word contains 4 bytes or 2 half-words, two possibilities exist for the ordering of bytes and half-words within the word. Suppose that a program must deal with a hexadecimal number of 0xBBCCDDEE. Figure 1 illustrates how that number appears in a register:

Bits 31-20	Bits19-16	Bits 15-08	Bits 07-00
BB	CC	DD	EE

Figure 1: Register Layout

If the memory location where this value is stored is displayed in 32-bit word format, the number will appear exactly as it appears in the register. When that number is accessed as bytes or half-words, the order of the subfields depends on the endian configuration of the system. If a program stores the above value at location 0x100 as a word and then fetches the data as individual bytes, two possible orders exist.

In the case of a little-endian system, the data bytes will have the order depicted by Figure 2. Note that the rightmost byte of the word is the first byte in the memory location at 0x100. This is why this format is called little-endian; the least significant byte of the word occupies the lowest byte address within the word in memory.

Address	0x100	0x101	0x102	0x103
Data	0xEE	0xDD	0xCC	0xBB

Figure 2: Little Endian Byte Order

If the program executes in a big-endian system, the word in Figure 1 has the following byte order in memory:

Address	0x100	0x101	0x102	0x103
Data	0xBB	0xCC	0xDD	0xEE

Figure 3: Big Endian Byte Order

The least significant byte of the word is stored in the high order byte address. The most significant byte of the word occupies the low order byte address, which is why this format is called big-endian. Operations with 16-bit half-words have similar consequences.

Endianness and ARM® Processors for 32-bit

When dealing with half-words, the memory address must be a multiple of two. Thus the value in Figure 1 will occupy two half-word addresses: 0x100 and 0x102. Figure 4 shows the layout for both endian configurations.

Address	0x100	0x102
Little Endian Data	0xDDEE	0xBBCC
Big Endian Data	0xBBCC	0xDDEE

Figure 4. Half Word Endian Orders

Note: Within the half-word, the bytes maintain the same order as they have in the word format. In little-endian mode, the least significant half-word resides at the low-order address (0x100) and the most significant half-word resides at the high-order address (0x102). For the big-endian case the layout is reversed.

Usually the issue of endianness is transparent to both programmers and users. However, the issue becomes relevant when data must cross between endian formats, such as may be the case for networks or distributed systems that use different platforms. The next section discusses how endianness relates to ARM processors.

Ramifications of Endianness and ARM Processors

The ARM Architectural Reference Manual states that ARM processors are bi-endian, meaning they can operate in either little-endian or big-endian modes. The ARM processor (v4 and v5) does not have any instructions or features that affect endianness. The endianness of the system as a whole is determined by the circuitry that connects the processor to its peripheral devices. In ARM v6 and beyond (all Cortex cores) the "setend" instruction was added to the ARM and Thumb instruction sets. It allows for dynamic changing of the CPSR E bit. Facilities were added to monitor the interaction of accesses/alignment and the E bit. For a description of the instructions, states, and monitors please refer to the current ARM Architecture Reference Manual.

In those ARM cores that contain a System Control coprocessor (coprocessor 15), Control Register 1 contains a bit that can be used to reverse the endian configuration of the system. The overall design of the system may require that this bit be set for proper operation. Check the user's manual for the board to determine the proper setting for this bit.

Since this bit is cleared at reset, the software engineer still must ensure that any code that executes either directly from flash or immediately after reset is formatted for the correct endian configuration, and the reset handler code must set this bit if required.

For those ARM cores, prior to v6, that do not include coprocessor 15, the software engineer has no control over the endian configuration of the system.

ARM specifications state that data values accessed in word format are invariant with respect to endianness. If a program stores a 32-bit value at a given memory address, then switches the endian configuration of the processor and reads back the 32-bit value at that same address, it will get the correct result. However, if data are read or written in smaller chunks (8 or 16 bits), this will no longer hold true. See "An Endian Experiment" below for some examples of what happens if the endian configuration is changed for 8- or 16-bit data.

Endian Issues When Programming Flash Devices

Today's flash memory devices are typically 8 or 16 bits wide. Some systems may implement a 32-bit-wide flash memory interface, but typically this actually consists of two interleaved 16-bit devices. Programming operations on these devices involve 8- or 16-bit data write operations at specific addresses within each device. For this reason, the software engineer must know and understand the endian configuration of the hardware in order to successfully program the flash device(s). This information typically is documented in the hardware reference manual for the board.

There are two main factors that must be considered to correctly program a flash device:

- A flash programming operation is initiated by placing the flash device into a special mode. This is typically accomplished by writing an 8- or 16-bit value to a particular address within the device. Where this address is mapped in the processor memory space will depend on the endian configuration of the system.
- Code which will be executed directly from an 8- or 16-bit flash device must be stored in a way that instructions will be properly recognized when they are fetched by the processor. This may be affected by the endian configuration of the system. Compilers typically have a switch that can be used to control the endianness of the code image that will be programmed into the flash device.

If the Target Flash Programming feature of SourcePoint is used to program the flash device, the user may select the "Swap Endian" option to reverse the endianness of the code image.

Endianness and ARM® Processors for 32-bit

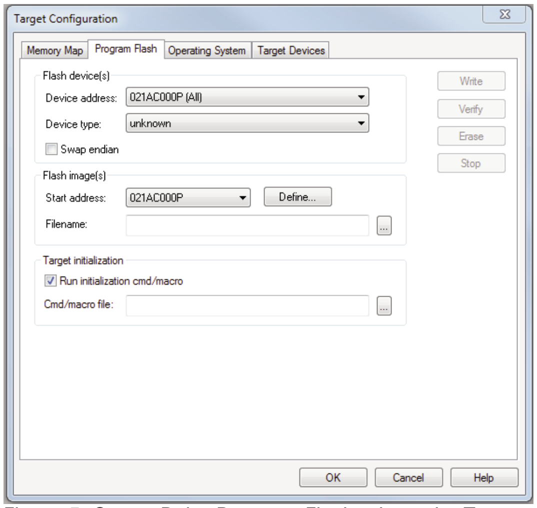

Figure 5: SourcePoint Program Flash tab on the Target Configuration dialog box

When selecting a device type from the "Device type" pull-down menu (Figure 5), the user must specify the endian configuration of the target. Currently, this menu provides the following choices:

- Intel 28 Series
- Intel 28 Series – Big Endian
- AMD 29 Series
- AMD 29 Series – Big Endian
- SST 39 Series
- SST 39 Series – Big Endian

If the endian configuration of the target does not match what the user has selected (e.g., big endian or not) the flash programming dialog will report that it was unable to locate or identify the flash device.

Also note that in many cases the user must provide a pre-flash initialization script to initialize the hardware into a state where a flash programming operation can be performed successfully.

An Endian Experiment

This section shows the results generated by a SourcePoint command file that writes an incrementing data pattern to target memory starting at address zero. The data are written to memory as 32-bit words, with the target in little endian mode. The data are then displayed in word, half word, and byte formats. The system is then switched to big endian mode and the data are displayed again. The entire experiment is then repeated with the target initially in big endian mode and switching to little endian mode.

Note that the byte order within each word does not change when the endian configuration is changed after data are written. However, the order of bytes within each word is affected by what endian mode is in effect when the data are written.

>//Load data in little endian mode
>control.b = 0
>load C:\ArmProjects\RomFiles\IncByteFilled.bin at 0
>//Display data in word format
>ord4 0 len 10

```
00000000   03020100   07060504   0B0A0908   0F0E0D0C
00000010   13121110   17161514   1B1A1918   1F1E1D1C
00000020   23222120   27262524   2B2A2928   2F2E2D2C
00000030   33323130   37363534   3B3A3938   3F3E3D3C
```

>//Display data in half word format
>ord2 0 len 20

```
00000000   0100   0302   0504   0706   0908   0B0A   0D0C   0F0E
00000010   1110   1312   1514   1716   1918   1B1A   1D1C   1F1E
00000020   2120   2322   2524   2726   2928   2B2A   2D2C   2F2E
00000030   3130   3332   3534   3736   3938   3B3A   3D3C   3F3E
```

>ord1 0 len 40
>//Display data in byte format

```
00000000 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
00000010 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
00000020 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
00000030 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
```

>//Switch to big endian mode
>control.b=1

>//Display data in word format
>ord4 0 len 10

```
00000000    03020100    07060504    0B0A0908    0F0E0D0C
00000010    13121110    17161514    1B1A1918    1F1E1D1C
00000020    23222120    27262524    2B2A2928    2F2E2D2C
00000030    33323130    37363534    3B3A3938    3F3E3D3C
```

>//Display data in half word format
>ord2 0 len 20

```
00000000   0302   0100   0706   0504   0B0A   0908   0F0E   0D0C
00000010   1312   1110   1716   1514   1B1A   1918   1F1E   1D1C
00000020   2322   2120   2726   2524   2B2A   2928   2F2E   2D2C
00000030   3332   3130   3736   3534   3B3A   3938   3F3E   3D3C
```

Endianness and ARM® Processors for 32-bit

```
>//Display data in byte format
>ord1 0 len 40
00000000 03 02 01 00 07 06 05 04 0B 0A 09 08 0F 0E 0D 0C
00000010 13 12 11 10 17 16 15 14 1B 1A 19 18 1F 1E 1D 1C
00000020 23 22 21 20 27 26 25 24 2B 2A 29 28 2F 2E 2D 2C
00000030 33 32 31 30 37 36 35 34 3B 3A 39 38 3F 3E 3D 3C

>//Load data in big endian mode
>load C:\ArmProjects\RomFiles\IncByteFilled.bin at 0
>//Display data in word format
>ord4 0 len 10
00000000 00010203 04050607 08090A0B 0C0D0E0F
00000010 10111213 14151617 18191A1B 1C1D1E1F
00000020 20212223 24252627 28292A2B 2C2D2E2F
00000030 30313233 34353637 38393A3B 3C3D3E3F

>//Display data in half word format
>ord2 0 len 20
00000000 0001 0203 0405 0607 0809 0A0B 0C0D 0E0F
00000010 1011 1213 1415 1617 1819 1A1B 1C1D 1E1F
00000020 2021 2223 2425 2627 2829 2A2B 2C2D 2E2F
00000030 3031 3233 3435 3637 3839 3A3B 3C3D 3E3F

>//Display data in byte format
>ord1 0 len 40
00000000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000010 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000020 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
00000030 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F

>//Switch to little endian mode
>control.b = 0
>//Display data in word format
>ord4 0 len 10
00000000 00010203 04050607 08090A0B 0C0D0E0F
00000010 10111213 14151617 18191A1B 1C1D1E1F
00000020 20212223 24252627 28292A2B 2C2D2E2F
00000030 30313233 34353637 38393A3B 3C3D3E3F

>//Display data in half word format
>ord2 0 len 20
00000000 0203 0001 0607 0405 0A0B 0809 0E0F 0C0D
00000010 1213 1011 1617 1415 1A1B 1819 1E1F 1C1D
00000020 2223 2021 2627 2425 2A2B 2829 2E2F 2C2D
00000030 3233 3031 3637 3435 3A3B 3839 3E3F 3C3D

>//Display data in byte format
>ord1 0 len 40
00000000 03 02 01 00 07 06 05 04 0B 0A 09 08 0F 0E 0D 0C
00000010 13 12 11 10 17 16 15 14 1B 1A 19 18 1F 1E 1D 1C
00000020 23 22 21 20 27 26 25 24 2B 2A 29 28 2F 2E 2D 2C
```

Summary

In many cases the endian configuration of a system is not relevant to the software engineer. However, in systems that contain memory-mapped peripheral devices (such as flash memory devices), endianness must be considered whenever the software must access data in bytes or half-words. See the user's manual for your board to determine the correct endian configuration.

The Target Flash Programming feature of SourcePoint allows the user to select the proper endian configuration of the target system before initiating a flash programming operation.

An initialization macro may be required to initialize the board into a state where it is ready for flash programming. This initialization macro may also need to set the endian control bit in the System Control Register (see the user's manual for your board).

17791 Fitch Avenue, Irvine, CA 92614 Voice: 949-517-4150 outside the US Fax: 714-731-6344 E-mail: ai-info@asset-intertech.com Web: www.asset-intertech.com