A Non-Cacheable Address Designating Scheme in MMU-less Embedded Microprocessor Systems

***Yong-Seok Lim, **Woon-Sik Suh, *Suki Kim**

* Department of Electronics Engineering, Korea Univ, ** Samsung Electronics Co. Ltd

E-mail: yslim@ulsi.korea.ac.kr

ABSTRACT

This paper proposes a novel scheme of designating non-cacheable addresses of memories in embedded systems of multi-master architectures without a Memory Management Unit (MMU). As a solution for data coherency problem between external memories and a cache memory, we proposes a cache masking scheme by allocating the most significant bit of address not used in 32-bit address system as indicator bit to designate non-cacheable address. As this scheme enables non-cacheable area designation every address, the simpler in the aspect of hardware and more flexible size of non-cacheable area can be obtained.

1. INTRODUCTION

In general, the procedure of instruction execution consists of the pipelining stages sequentially that fetching an instruction from memory, decoding the instruction, execution instruction and lastly writing back result. According to instructions, there is the procedure fetching operand from memory and saving again the calculation result in the memory. In the course of this series of memory access, if operand changes any time by the multiprocessor system sharing one bus or organized multiple masters, the coherency between cache and memory shall be considered carefully. General solutions for this cache coherency protocol are as follows [1];

- BUS snooping protocol for shared bus system
- Directory-based protocol
- Software-based protocol
- Static protocol (shared writable data is non-cache)

Firstly, for BUS snooping protocol techniques, cache controller continuously compares the tag of its cache with the address on memory and system bus, and monitors memory and system bus whether I/O systems or other bus masters read and write. The bus snooping hardware of the cache controller maintains information about the state of every cache line and takes appropriate action to maintain coherency [2]. Secondly, Directory-based protocol divides physical memory into fixed-size blocks and saves entry for each cache in the directories of blocks. Directory entries can be distributed so that different requests can go to different memories, thereby reducing contention and allowing a scalable design. By using hardware, these schemes secure the merit of software transparency to make cache not be seen in the operating system and software, while these are complex in the aspect of hardware and may not be suitable for the embedded systems for special purposes. Thirdly, Software-based protocol, relies on either laborious programming or elaborate compilers, insert cache control instructions at compile-time to maintain coherence. This scheme could force the cache to flush its contents to main memory whenever a critical section of code was exited [1]. Lastly, static protocol (shared writable data is non-cache) is the simplest cache coherency strategy among these schemes; it maintains coherency by non-cache tagging at a block to be changeable by multi-master and making it exist only in main memory. Cache can save any data in the whole system memory area, however, if it is designated as non-cacheable, it cannot be cached when a read-miss occurs. This strategy is much used in the embedded system due to its simple and plain feature of hardware implementation.

Besides, there are many variations on the cache coherency that are much more complicated models. The one found on both Pentium-Pro and PowerPC is called MESI [2][3], but is beyond the scope of this paper.

This paper is to examine what existing schemes were applied to the MMU-less embedded systems for the static protocol strategy and then to describe the schemes proposed in this paper below [4][5][6].

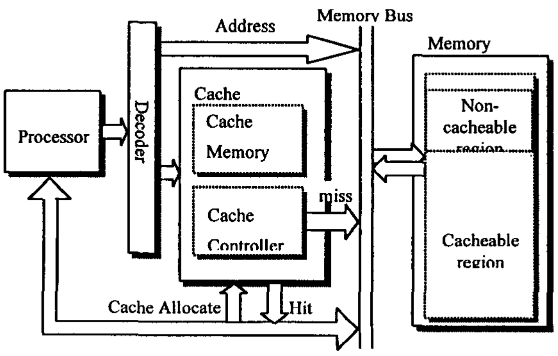

Fig 1. Block diagram of static protocol having an external cache

2. GENERAL NON-CACHEABLE AREA DESIGNATING SCHEMES

Fig 1 is a general static protocol block diagram. As the mention above, static protocol is usually used in embedded systems due to its simple characteristic. And, in the aspect of hardware can be easily constructed - simple decoder, some registers and so on. In Embedded systems, this scheme can be applied as following.

2.1 Scheme 1 - Specification start point and end point to indicate non-cacheable areas

The first scheme is setting non-cacheable areas by writing to register starting points and end points of the areas as shown in Fig 2 [4]. But, it has a limitation of non-cacheable area designation as the number of non-cacheable area designating registers is affected

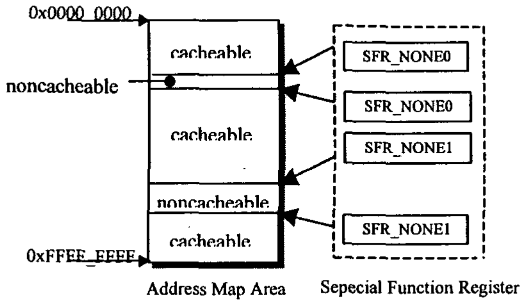

Fig 2. Scheme of using start point and end point to designating non-cacheable areas

2.2 Scheme 2- Non-cacheable area designating scheme having fixed size

The second scheme is setting non-cacheable areas in the non-cacheable pointer registers and comparing these with addresses from CPU using comparator. This scheme saves a part of physical address in the register and this part decides size of the non-cacheable area. Fig.3 shows whether to perform caching as the address from CPU is compared with address set in the non-cacheable pointer register.

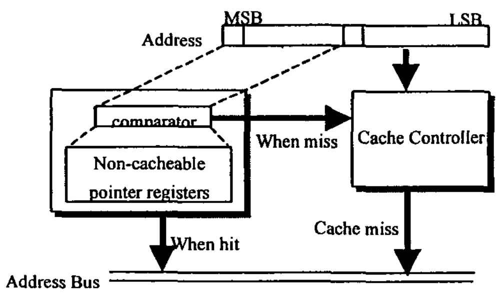

Fig 3. Block diagram of scheme having a fixed

This scheme, compared to the one specifying a start point and an end point to designate non-cacheable areas, can reduce the number of registers, but resolution –the size of non-cacheable area- is limited and the number of non-cacheable area designations depends on the number of registers as the scheme specifying a start point and an end point.

The resolution setting is as shown in Fig.4. That is, each non-cacheable area has the $2^{(32-m-n)}$ value in size.

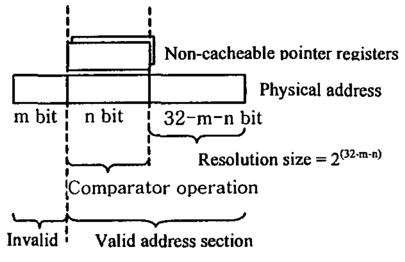

Fig 4 Resolution decision method

3. PROPOSED NON-CACHEABLE ADDRESS DESIGNATING SCHEME

-Mirroring scheme

The main idea of this proposed paper is using an address that is not used on the physical address to mask a cache operation. It means that the MSB of the address becomes a non-cacheable indicator whether to perform the cache operation or not. If this bit is 1, although cache miss cycle, cache does not operate the miss cycle. On the contrary, if this bit is 0, cache performs the normal operation including the cache miss cycle. Prior to this paper, it is impossible to specify the non-cacheable address by the unit of addresses or variables, to access hardware controlling registers called special function registers (SFR). And also those schemes are not available for masking the small size non-cacheable area such as char-type variables. In this proposed scheme, the cases such as the variable, data structure variables size and hardware controlling register accesses can be applied.

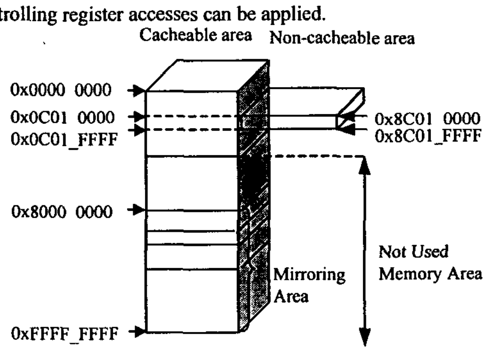

Fig 5. Example of mirroring Scheme

To achieve this scheme, not only hardware implementations controlling the cache system with the not-used bit of addresses but also software implementations using the mirroring area through the variable assignment should be considered. Fig.5 shows how the most significant bit specifies the non-cacheable areas from 0x0C01_0000 to 0x0C01_FFFF in the memory with the physical area of 1G Byte. The logical address space is separated into mirroring area and normal area in the figure. This mirroring area is made use of the non-cacheable channel on the software. That is, to designating the non-cacheable address, the mirroring area from 0x8000_0000 to 0xFFFF_FFFF should be used. The value of the MSB address bit depends on the software programming. When the address is targeting the physical address space, it means that the value of MSB is 0 and the cache normally operates. On the other hand, if the address is targeting the mirroring address space exceeding the physical address, the value of MSB is 1, and the hardware masks cache operation. Because this MSB is not propagated to other below logics, irrelevantly masking the cache, the other logics operate normally.

Fig.6 shows the actual implementation results of hardware and includes descriptions of the operation on the software in the following Example. If a variable for specifying the non-cacheable address in software is pBUF_B, the address is allocated in the mirroring area to include the non-cacheable indicator (line3 in Example) and just this variable on the program like main routine in Example is used. This rule is essential to fully exclude a caching status not intended, which may occur in non-caching of a variable with data size less than a block size of cache.

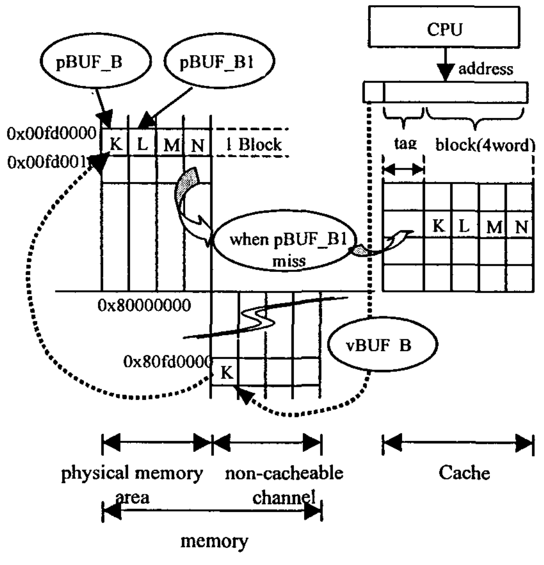

Fig 6. Example of data transferring in Mirroring Scheme

A caching state of the variable specifying a non-cacheable address and an example of fully solving it are illustrated in Example in detail. For instance, like the case of mark 1 in Example, a condition that the content of pBUF_B is cached physically in accessing pBUF_B1, a variable adjacent to non-cacheable variable pBUF_B occurs. After cache read miss cycle occurs, if address 0x80fd0000 included in the mirroring space is issued at next cycle, the MSB of address masks the cache operation and memory controller gives data K to CPU instead of cache controller. In this case of accessing data just with vBUF_B without using pBUF_B on the program, in any event, data K inside the cache is never accessed. (mark 2 in Example). Fig.6 shows this procedure. Also this process is applied to the write cycle. When CPU issues address 0x80fd0000 through variable vBUF_B which designates a mirroring area, cache controller recognizes that value of the MSB is 1, and then masks the cache. However, memory controller receives address 0x00fd0000 from the address bus and writes data k at the place of address 0x0fd0000.

```
#include <stdio.h>
#define pBUF_B (*(volatile unsigned *)0x00fd0000)
        /* Physical Address for Non-Cacheable */
#define pBUF_B1 (*(volatile unsigned *)0x00fd0004)
        /* Physical Address for Cacheable */
#define vBUF_B (*(volatile unsigned *)0x80fd0000)
        /* Non-Physical Address for Non-Cacheable */

void main(void)
{
        unsigned char Cac_B1, Non_B;

        pBUF_B1 = 0x66;
        vBUF_B = 0x77;

        Cac_B1= pBUF_B1;            // mark 1
        Non_B = vBUF_B;             // mark 2
        return 0;
}
```

```
main
0x000000: e3a00066.... : MOV  r0,#0x66
0x000004: e3a018fd .... : MOV  r1,#0xfd0000
0x000008: e5810004.... : STR  r0,[r1,#4]
0x00000c: e3a02077....: MOV  r2,#0x77
0x000010: e1810f00....: ORR  r0,r1,r0,LSL #30
0x000014: e5802000....: STR  r2,[r0,#0]
0x000018: e5911004....: LDR  r1,[r1,#4]
0x00001c: e5900000....: LDR  r0,[r0,#0]
0x000020: e3a00000....: MOV  r0,#0
[illegible]
```

Example. Software example of non-cacheable application.
Assemble test code is made of ARM instruction [7] sets and generated by ARM SDT.

Example shows assembly language code by applying the mirroring scheme to prove that physical memory area is different just in only upper one bit from virtual address area (mirroring area). In this procedure, vBUF_B, as a variable of mirroring area, is non-cacheable channel of pBUF_B, a variable in the physically existing area.

The assemble code in Example suggests the process of data access through mirroring channel as the assemble result of C code.

The hardware configuration by this scheme is implemented more simply compared to existing ones and secure the availabilities. Compared to existing ones, that is to say, this proposed scheme does not have to any registers to store addresses of designating non-cacheable areas and any comparator, either. But nevertheless this scheme can be obtained more flexible size of non-cacheable area and specify non-cacheable area every addresses.

4. CONCLUSIONS

Non-cacheable area designation schemes are usually used at the MMU-less embedded systems due to their simple characteristics. In this proposed paper, by allocating the variables for accessing non-cacheable areas to mirroring channel which is the address over the specifiable actual range of memory map -mirroring area- and making a most significant bit of CPU address which is actually not used as a non-cacheable indicator bit, we can obtain much flexibility in assigning non-cacheable area and designate non-cacheable area in byte address unit and need not have any extra registers to specify these non-cacheable areas. The idea of this paper, although it has a bit of elaborateness of software (non-cacheable physical memory access through mirroring channel), can be implemented by allocating only address of a variable for cache masking to the mirroring area, following the general flow of software development.

REFERENCES

[1] Scott E. Crawford and Ronald F. DeMara "Cache Coherence in a Multiport Memory Environment" Massively Parallel Computing Systems, 1994, Proceedings of the First International Conference on, 1994

[2] Nikitas Alexandridis " Design of microprocessor – based systems" prentice hall, 1993

[3] David A. Patterson and John L. Hennessy "Computer Organiztion & Design" Morgan Kaufmann, 1997

[4] Samsung's S3C44B0X (KS32C41100) 16/32-bit RISC MCU Data Book, Samsung electronics.

[5] Eisner, C.; Shitsevalov, I.; Hoover, R.; Nation, W.; Nelson, K.; Valk, K "A methodology for formal design of hardware control with application to cache coherence protocols". Design Automation Conference, 2000. Proceedings 2000, 2000, Page(s): 724 –729

[6] Tomasevic, M.; Milutinovic, *V* "Hardware approaches to cache coherence in shared-memory multiprocessors. 2". IEEE Micro, Volume: 14 Issue: 6, Dec. 1994 Page(s): 61 -66

[7] Steve Furber, ARM System-On-Chip Architecture. Addison-Wesley, ADDISON-WESLEY, 2000