A Study of Transparent On-chip Instruction Cache for NV Microcontrollers

Dahoo Kim, Itaru Hida, Eric S. Fukuda, Tetsuya Asai and Masato Motomura

Graduate School of Information Science and Technology
Hokkaido University
Sapporo, Hokkaido, Japan
Email: { kim@lalsie., hida@lalsie., fukuda@lalsie., asai@, motomura@ } ist.hokudai.ac.jp

***Abstract*—Demands for low energy microcontrollers have been increasing in recent years. Since most microcontrollers achieve user-programmability by integrating non-volatile (NV) memories, such as flash memories for storing their programs, the large power consumption required in accessing an NV memory has become a major problem. This problem becomes even critical when lowering the power-supply voltage of NV microcontrollers to achieve power and energy reduction. In this paper, we try to solve this problem by introducing an instruction cache and thus reducing NV memory access frequency. Unlike general-purpose microprocessors, it is important for microcontrollers used for real time applications in embedded systems that the program execution time can be calculated accurately prior to its execution. Therefore, we introduce a "transparent" instruction cache, which does not change the existing NV microcontroller's cycle-level execution time, for reducing power and energy consumption, but not for improving the processing speed. We have conducted detailed microarchitecture design based on a major industrial microcontroller architecture, and studied, as a preliminary evaluaion, hit rates of several instruction cache configurations.**

Keywords–embedded system; micro-controller; instruction cache; non-volatile; low power design.

I. INTRODUCTION

In recent years, sensor networks have been widely studied as a fundamental technology to realize the "Smart Society" [1]. In order to implement sensor networks in various application fields, sensor nodes which can operate for a long time with a small energy source are required. Therefore, it is necessary to reduce power consumption of the microcontroller operating a sensor node.

Meanwhile, NV microcontrollers (microcontrollers integrated with non-volatile memories) are widely used due to its convenience to develop embedded system's software. However, the power consumption of the non-volatile memory dominates the total power consumption of the microcontroller [2]. Furthermore, it is hard to reduce the power consumption of non-volatile memories in microcontrollers. Focusing on this point, the purpose of our work is to reduce power and energy consumption by introducing an instruction cache to the microcontroller, reducing access to the non-volatile memory.

Besides, as shown in Table I, the traditional cache architecture research has aimed at improving the performance of the microprocessor by introducing the high-speed cache memory (SRAM) between the main memory and the datapath by reducing the memory access time [3]. On the other hand, in the case of NV microcontroller used for real time applications in embedded systems, it is important that the program execution time can be calculated in advance. Furthermore, the change of execution time due to cache misses should be avoided since such a change can cause problems to the system such as real time applications. Thus, it is necessary to introduce an instruction cache, which does not cause cache miss penalties, while leaving the speed of the memory access at cache hits unchanged.

TABLE I. CONCEPT OF THIS WORK

Traditional Cache Architecture Research	**This work**
Micro Processor Datapath I-Cache (SRAM) D-Cache (SRAM) Main Memory (SRAM, DRAM)	NV Micro Controller for Embeded System Datapath IM (NVM) DM (SRAM) Datapath I-Cache (SRAM) IM (NVM) DM (SRAM)
High Performance	Low power & Low Energy Transparent (i.e., doesn't change cycle-level timing)

Therefore, in this paper, we aim at lowering power and energy consumption rather than improving the performance by introducing a transparent instruction cache, which does not change cycle level timing of existing NV microcontrollers.

The rest of this paper is organized as follows: Section 2 describes the features of our research. Section 3 describes the architecture of the proposed instruction cache employed to the NV microcontroller. Section 4 discusses preliminary evaluation of the proposed instruction cache. Section 5 summarizes future works.

II. FEATURES OF OUR RESEARCH

The Features of our Research are as follows:

1. **Examination of instruction cache suitable for microcontroller deployments** - As described in Section 1, in the case of a microcontroller, it is important to prolong the battery run time than to reduce the processing time. For this reason, the instruction cache that we propose in this paper is intended to reduce the power consumption rather than to improve the processing speed.

2. **Evaluation based on a realistic microcontroller architecture** - We evaluated the power and energy consumption of our system built on a base microcontroller, Renesas Electronics Corporation's 78K0R. Since not all of 78K0R's specifications

 ISBN: 978-1-61208-379-7

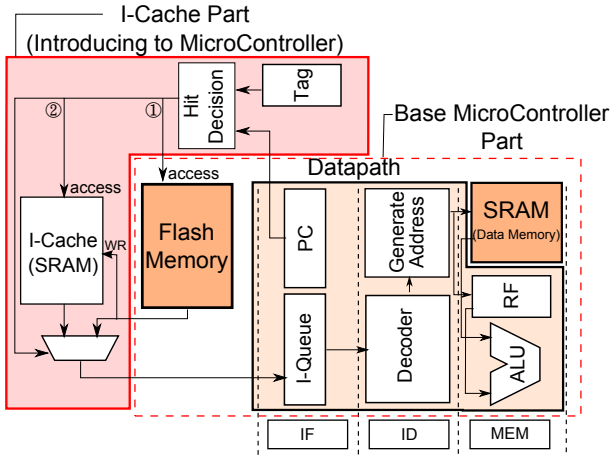

Figure 1. Concept of 1-word-per-line instruction cache architecture

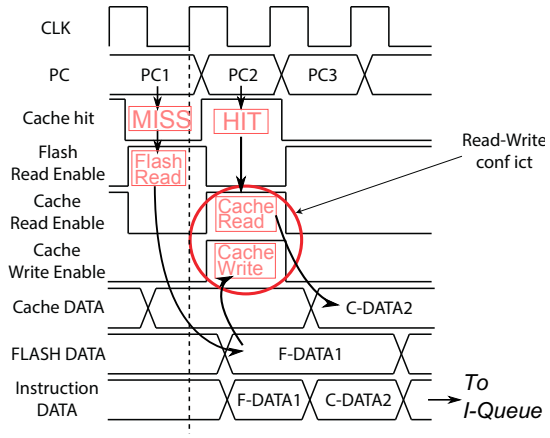

Figure 2. Instruction memory access timing in the case of cache hit immediately after cache miss

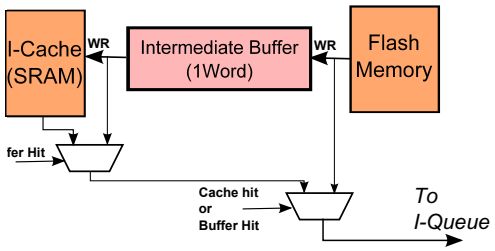

Figure 3. Intermediate buffer insertion method for 1-word-per-line instruction cache architecture

are open, we implemented the base microcontroller using only the publicly available information [5] [6] [7].

III. ARCHITECTURE

A. Base Microcontroller

Our microcontroller is based on 78K0R, which is widely used in various industrial fields [4], whose block diagram is shown as a part of Figure 1. 78K0R has a flash memory as its NV instruction memory and an SRAM as its data memory [5].

This architecture has a pipeline structure of three-stages (IF stage, ID stage, MEM stage) [6]. In the IF stage, the microcontroller provides an address from the program counter (PC) to the instruction memory (flash memory) and fetches an instruction sequence from the instruction memory. This instruction sequence is stored in an instruction queue (I-Queue). In the ID stage, the microcontroller decodes the instruction that has been fetched in the IF stage and extracts data memory's (SRAM) and RF's (Register File) addresses to be accessed. In the MEM stage, the microcontroller retrieves the data from the data memory and executes the instruction.

Also, the base microcontroller adopts the CISC architecture [6]. The number of the instructions of the base microcontroller is 915, and the instruction length is 1 byte to 5 bytes. Base microcontroller has four-byte (1 word) instruction queue (I-Queue) that contains an instruction sequence fetched from the instruction memory. Therefore, if a valid instruction exists in the I-Queue, there is no need to access the flash memory.

B. 1-word-per-line instruction cache

The base microcontroller fetches one word from the instruction sequence in the flash memory in one cycle [6]. Thus, as the first step to introduce the instruction cache to the base microcontroller, we designed a 1-word-per-line instruction cache architecture, as shown in Figure 1. We assumed that the operation timing of the instruction cache to be the same as the timing of the base microcontroller's access to the flash memory: The instruction sequence is read from flash memory in the subsequent cycle of the access to the flash memory.

Actual operation of the instruction cache is as follows: (1) In the case of a cache miss, the microcontroller accesses the flash memory and fetches 1 word from the instruction sequence. (2) In the case of a cache hit, the microcontroller accesses the instruction cache (I-Cache) and fetches 1 word from the instruction sequence in the instruction cache as well as the flash memory. Also, in the case of a cache miss, the instruction sequence which is read from the flash memory is written to the instruction cache. Since the bit widths of the instruction cache and the flash memory are the same, there is no penalty for writing to the instruction cache.

Since the instruction cache we propose does not allow cache miss penalty, there is a problem that a read access and a write access to the instruction cache can occur coincidentally. For example, as shown in Figure 2, when a cache miss occurs at PC1, microcontroller reads the flash memory. Then, in the next cycle, the instruction sequence is read from the flash memory. Thus, the instruction sequence for the cache miss (F-DATA1) must be written to the instruction cache in this cycle (the next cycle of the cache miss). In the cycle immediately after the cache miss, when a cache hit occurs at PC2, an instruction is fetched from the instruction cache. In this case, the collision of a read access and a write access to the instruction cache occurs.

For solving this problem, we designed **intermediate buffer insertion method** for the transparent instruction cache. This is a method to delay the write access to the cache by inserting an intermediate buffer to which the instruction sequence in the cache miss is written between the instruction cache and the flash memory. Figure 3 shows its architecture. In this method, the timing of writing the instruction sequence to the instruction cache is a cycle that does not access the instruction cache. If a valid instruction is still in the I-Queue of the datapath as described in the base microcontroller architecture, and if a

 ISBN: 978-1-61208-379-7

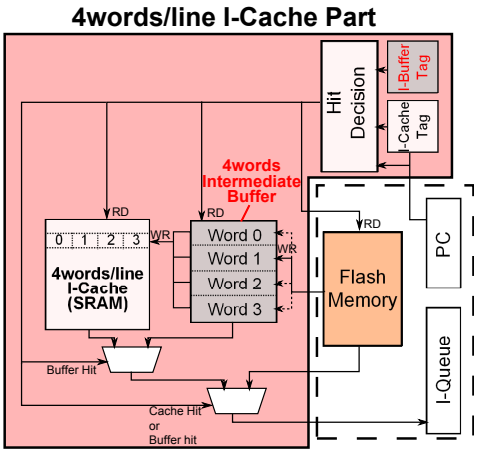

Figure 4. 4-word-per-line instruction cache architecture

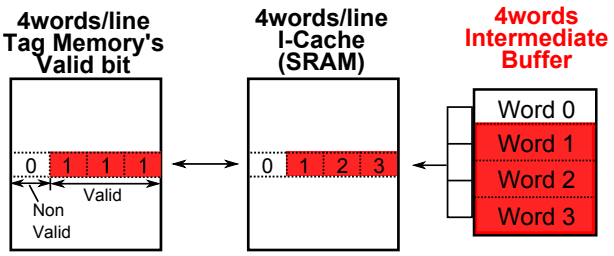

Figure 5. Writing operation of 4-word-per-line instruction cache

cache miss occurred, the microcontroller does not have access to the instruction cache.

Particularly, when a cache miss occurred, the instruction sequence that has been present in the intermediate buffer is written to the instruction cache and the instruction sequence that has been read from the flash memory is newly written to the intermediate buffer. Therefore, the intermediate buffer will not overflow with the buffer with only 1 word. Also, the *intermediate buffer insertion method* determines a cache hit in the intermediate buffer, and enables the instruction sequence to be read from the intermediate buffer.

C. 4-word-per-line I-Cache

As the second step to integrate the instruction cache to the base microcontroller, we designed 4-word-per-line instruction cache architecture that can take advantage of spatial locality.

The flash memory (NV memory) of the base microcontroller (existing NV microcontroller) reads one word from the instruction sequence in one cycle. Therefore, in order to implement the 4-word-per-line instruction architecture, a buffer for storing four words is required. We have implemented 4-word-per-line instruction cache by extending the number of words of the intermediate buffer, as shown in Figure 4. The instruction sequence that is read from the flash memory in one cycle is stored in the intermediate buffer one word per cycle. When all the four entries of the intermediate buffer are filled, the contents are sent to the 4-word-per-line instruction cache memory.

For example, when a miss occurs in the first cycle, the instruction sequence that is read from the flash memory is stored to one of the entries of the intermediate buffer by referring to the lower 2 bits of the instruction address (held in the PC). In the next cycle, if the flash memory is not accessed, in other words, if there are cache hits, buffer hits or I-Queue hits, the next instruction sequence is read from the flash memory by incrementing the PC and is stored to the next entry of the intermediate buffer. In this way, when four words are collected in the intermediate buffer, four words of the instruction sequence will be written to the instruction cache memory.

However, when a miss occurs before collecting the four words in the intermediate buffer, it is necessary to store the instruction sequence that has been read from the flash memory to the intermediate buffer. In this case, the existing instruction sequences in the intermediate buffer is written to the instruction cache, and the instruction sequence that has been read from the flash memory on a miss is stored to the intermediate buffer. For example, as shown in Figure 5, if a miss occurs when only three words of instruction sequence are stored in the intermediate buffer, the three words are written to the instruction cache, invalidating the 0-th word. In this paper, we extend the 4-word-per-line tag memory's valid bits to 4 bits, and make them indicate the valid word of each line.

However, when writing the line which has invalid words to the instruction cache, such as the case shown in Figure 5, it is expected that there is a case of a low hit rate, due to the deletion of the valid word that existed in one line of the instruction cache. We will evaluate our system in this regard.

D. Associativity

As the third step to integrate the instruction cache to base microcontroller, we have increased the associativity to 2 and 4 from 1. It is expected that the power consumption consumed by the control unit of the tag memory is increased because the control unit of the tag memory becomes complicated by increasing the associativity. Yet, there is a possibility that the hit rate will rises. We will evaluate our system in this regard as well. Also, we adopt a pseudo least recently used (LRU) replacement algorithm.

IV. Preliminary Evaluation

As the first step to evaluate the effect of proposed instruction cache on reducing power and energy consumption, we evaluated a hit rate on a few benchmark programs. It is expected that the effect increases with an increase in hit rate because of decreased access to a flash memory. Benchmark programs and each program's size are shown in Table II.

First, the hit rates of 1-word-per-line instruction cache for each benchmark programs are shown in Figure 6. Because the program sizes are small, they, from bubble sort program to factorial program, showed high hit rates for every cache size.

TABLE II. BENCHMARK PROGRAM AND PROGRAM'S SIZE

Evaluation program	size[Byte]
bubble sort	614
Celsius to Fahrenheit	585
Checksum	535
Copy verify	602
Factorial	606
EEMBC Coremark	11701

 ISBN: 978-1-61208-379-7

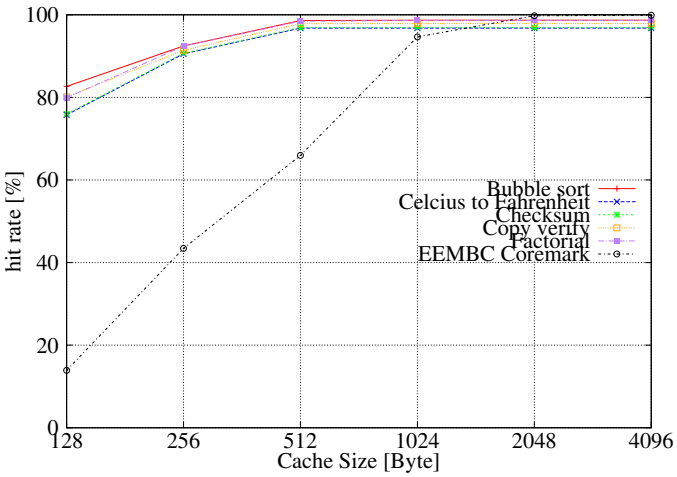

Figure 6. Cache hit rate by cache size for each programs (1-word-per-line)

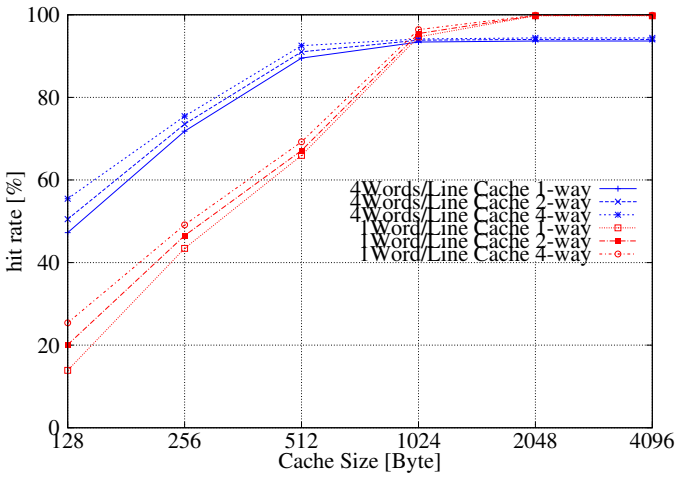

Figure 7. Cache hit rate by cache size (Evaluation program : EEMBC Coremark)

Hereafter, we will evaluate more detailed power consumption using EEMBC Coremark program.

In EEMBC Coremark program, the hit rate of each cache size is shown in Figure 7. With a large sized instruction cache (1 Kbytes or more), 1-word-per-line instruction cache has higher hit rate than 4-word-per-line instruction cache. This is because there is also a case that a valid data which was in the instruction cache in 4-word-per-line instruction cache has been discarded as described in 4-word-per-line instruction cache architecture. In the case of instruction cache of less than 1 Kbytes, 4-word-per-line instruction cache has a higher hit rate. This is because 4 word-per-line instruction cache can take advantage of spatial locality, even if the valid data has been discarded. Also, if the associativity is high, hit rate was high in both methods. Especially, when the cache size was small, its effect became significant.

Based on the above results, we will evaluate a more detailed power and energy consumption.

V. Conclusion and Future Work

In this paper, we proposed the transparent instruction cache architecture for reducing power and energy consumption of a practical NV microcontroller architecture. Unlike traditional cache architecture researches, we intended to reduce power and energy consumption rather than to improve processing speed. This is because, in the case of NV microcontroller used for real time application in embedded systems, it is important to prolong the battery run time and not to change the existing NV microcontroller's cycle-level execution time.

Our future work is to evaluate the effect of proposed instruction cache architecture by estimating more detailed power and energy consumption of this architecture. In order to estimate a more detailed power and energy consumption, we will develop an RTL description of the base microcontroller, and will integrate the proposed instruction cache architecture to the RTL description. We can generate back-annotated netlists after logic synthesis and placement/routing, then, using which power and energy reduction of the proposed architectures can be evaluated fairly accurately.

References

[1] V. C. Gungor, Bin Lu, and G. P. Hancke, "Opportunities and Challenges of Wireless Sensor Networks in Smart Grid", Industrial Electronics, IEEE Transactions on, pp. 3557-3564 , October, 2010.

[2] H. G. Lee, and N. Chang, "Energy-aware memory allocation in heterogeneous non-volatile memory systems", in Proc. of the 2003 international symposium on Low power electronics and design, pp. 420-423 , August, 2003.

[3] J. R. Goodman, "Using cache memory to reduce processor-memory traffic", in Proc. of the 10th annual international symposium on Computer architecture, pp. 124-131 , June, 1983.

[4] K. Oba, K. Kawai, R. Matsushita, K. Ishihara, and K. Eto, "Development of 16-bit All Flash Microcomputers " 78K0R/Kx3-L "Featuring Ultralow Power Consumption", Nec Technical Journal, 4(1), 35: , March, 2009.

[5] Renesas Electronics Corporation, "Renesas Demonstration Kit for RL78G14 User's Manual", http://documentation.renesas.com/doc/products/tools/r20ut2534eu0200_yrdkrl78g14_um.pdf , October, 2013.

[6] Renesas Electronics Corporation, "RL78 Family User's Manual: Software", http://documentation.renesas.com/doc/products/mpumcu/doc/rl78/r01us0015ej0210_rl78.pdf, January, 2014.

[7] Renesas Electronics Corporation, "78K0R/Hx3 User's Manual: Hardware", http://documentation.renesas.com/doc/products/mpumcu/doc/78k/r01uh0260jj0300_78k0rhx3.pdf, September, 2013.

 ISBN: 978-1-61208-379-7