Evaluation of Portability and Design Diversity of FabCache

Takahiro Sasaki*, Takaki Okamoto, Seiji Miyoshi, Yuki Fukazawa, Toshio Kondo
Graduate School of Information Technology Engineering, Mie University, Japan. 1577 Kurimamachiya-cho, Tsu city, Mie prefecture, 514-8507, Japan.

* Corresponding author. Tel.: +81-59-231-9780; email: sasaki@arch.info.mie-u.ac.jp
Manuscript submitted October 29, 2015; accepted May 8, 2016.
doi: 10.17706/ijcee.2016.8.3.185-196

Abstract: Single-ISA heterogeneous multi-core architecture which consists of diverse superscalar cores is becoming more importantly in the processor architecture. Using a proper superscalar core for characteristic in a program contributes to reduce energy consumption and improve performance. However, designing a heterogeneous multi-core processor requires a large design and verification effort. Therefore, we have proposed FabHetero which generates diverse heterogeneous multi-core processors automatically using FabScalar, FabCache, and FabBus which generate various designs of superscalar core, cache system, and flexible shared bus system, respectively. In our previous work, we estimated the physical design, delay, and power consumption only on a L1 instruction cache of a 32-bit processor. However, almost all modern processor has a L1 data cache, and nowadays there are many 64-bit processors to achieve high-performance computing. This paper shows availability and efficiency of the FabCache by estimating overheads, area, delay, and power consumption, of both instruction cache and data cache systems for both 32-bit and 64-bit processors. FabCache has good portability because bus communication system of generated cache system from FabCache employs AMBA4 protocol that is widely used in various architecture to communicate with other design and its connection logic can be parameterized such as individual bus width. To show an effectiveness and portability of FabCache, this paper applies FabCache to FabScalar-alpha which is a 64-bit processor, and evaluates availability and effectiveness of the generated cache system. According to the evaluation results, the cache systems generated by FabCache works perfectly and the increased area is about 1.7%, delay is 0.1ns, and power is 0.1% compared with hand designed cache system. Evaluation results show that the FabCache can generate reasonable cache system and it has good portability.

Key words: Cache generator, design automation, heterogeneous multi-core processor, VLSI design.

1. Introduction

The studies of single-ISA heterogeneous multi-core processors attract attention in various fields from mobile computing to high-performance computing [1]-[3]. Using a suitable superscalar core for characteristic in a program contributes a high-performance computing with low-energy consumption. However, designing a single-ISA heterogeneous multi-core requires a huge design and verification effort that is multiplied by the number of different cores, its cache systems, and complex shared bus system. To solve this problem, N. K. Choudhary *et al.* propose FabScalar [4], [5] that automatically generates Register-Transfer-Level (RTL) design of diverse superscalar cores. FabScalar can parameterize many microarchitectural diversity such as fetch/issue width, number of pipeline stages, size of tables, and

function unit mix to expose instruction level parallelism (ILP). FabScalar contributes to mitigate the design and verification effort that currently limits the amount of microarchitectural diversity that can be practically implemented for both research and commercial products. Although the current FabScalar generates diverse superscalar cores, each core on a single-ISA heterogeneous multi-core requires a suitable cache system and each differently-designed cache should be connected with a flexible shared bus system. As for the cache systems, we must consider the organization including capacity, line size, associativity, design hierarchy, and coherency protocol to optimize the design. Because of same reason as cores, the cache systems require a huge effort. To solve this problem, we have proposed FabHetero [6], which is a framework to design diverse heterogeneous multi-core processors automatically. FabHetero automatically designs not only processor cores but also an entire heterogeneous multi-core processor includes cache and shared bus system. FabHetero consists of three design automation tools: FabScalar for core design, FabCache [7] for cache design and FabBus [8] for shared bus design. All designs generated by FabCache can be synthesized with single-ported and dual-ported memory. Therefore, FabCache can be used for simulation with diverse cache systems and it is suitable for a standard cell-based ASIC design flow which does not support highly-ported memory, and for FPGA.

As a previous work [9], we estimated various design evaluations on FabScalar which is a 32-bit processor. However, the evaluations is performed only on a L1 instruction cache. Furthermore, nowadays, there are many 64-bit processors to achieve high-performance computing. To achieve the requirement, FabScalar-alpha which is a 64-bit processor and has been proposed as a branch project of FabScalar [6]. Therefore, we apply both L1 instruction and L1 data cache generated by FabCache into FabScalar-alpha to clear trend of overheads, area, delay and power consumption. We analyze the design results of both L1 instruction and L2 data cache for both 32bit and 64bit processors, and compare them with hand designed cache system to show reasonability of our FabCache. According to the results of L1 instruction and data cache, cache systems generated by FabCache works perfectly, and the increase area is about 1.7%, delay is 0.1ns, and power is 0.1%. The trend is almost all of the same with our previous work. Finally, we can say that FabCache has a good portability and can be used differently environmental through the porting to FabScalar-alpha and the evaluation results.

2. Related work

Single-ISA heterogeneous multi-core architecture which consists of diverse superscalar core types is becoming more importantly in the processor architecture. Using a proper superscalar core for characteristic in a program streamlines the execution of various programs and program phases. Many researches intend to achieve a high performance and improve energy- and area-efficiency by adapting to application heterogeneity [1]-[4], [10]. In these works, the cache structure is recognized as an important performance factor. In particular, B. de Abreu Silva *et al*. [10] focuses on heterogeneity of cache size. The heterogeneous paradigm results in the new requirement for designing diverse cache systems in a chip to fully exploit application heterogeneity. This aspect incurs a large cache design effort for a heterogeneous multi-core processor. No prior work addresses design effort to implement diverse cache systems, and FabCache is the first work to mitigate design effort for heterogeneous multi-core processors.

A cache generator is proposed to easily improve performance of a soft processor on an FPGA [11]. In order to be applied to various target systems, cache generator are required to produces cache systems with various associativities, latencies, and dimensions. However, this generator can produce only three types of associativities: direct mapping, two-way set associative and full-associative. More importantly, the cache hierarchy is inflexible and caches generated by this generator cannot be adapted to differently-designed superscalar cores because the generator does not support arbitrary fetch width. Leon4 [12] also proposes a

configurable cache system; though it targets a scalar pipeline processor and does not support arbitrary fetch width also. In addition, there are a lot of cache generators such as ref [13]-[15]. However, it is difficult to generate the cache systems to suit for diverse heterogeneous multi-core system because it is not parameterizable. In contrast, FabCache composes a cache of desired associativity, hierarchy, and fetch width and these factors are parameterizable.

3. FabHetero

Before we explain FabCache, we describe FabHetero which is the project of automatic generation of an entire heterogeneous multi-core processor. Developing an emerging heterogeneous multi-core processor will require to design a huge combinations of processor cores, cache systems, and a shared bus system. Therefore, automatically generating arbitrary processor cores, cache systems, and bus systems facilitates design of heterogeneous multi-core processors. We have proposed FabHetero as a tool-set for achieving the requirement. FabHetero is a framework to generate diverse heterogeneous multi-core processors automatically using FabScalar, FabCache and FabBus. Fig. 1 shows an example of the heterogeneous multi-core processor generated by FabHetero. There are three differently-designed cores (Core 0, Core 1, and Core 2), and each core has different cache structure. The shared bus (in the figure, Interconnect) connects the diverse cache systems with the shared memory (a last level cache or main memory). As for the cache systems, Core 0 has only L1 instruction and L1 data caches, Core 1 has unified L2 cache in addition to L1 caches, and Core 2 has dedicated L2 caches instead of unified L2 cache (each cache design may differ in cache capacity, line size and associativity). Both L1 and L2 cache hierarchies can be omitted if a core does not require cache system. Such non-cache design is valuable for in particular the field of embedded system in which a part of or all cache hierarchies are often removed. To generate various processor cores, cache systems and flexible shared bus system, FabHetero uses FabScalar, FabCache and FabBus, respectively. At first, we describe FabScalar and FabBus briefly, and then the detail of FabCache.

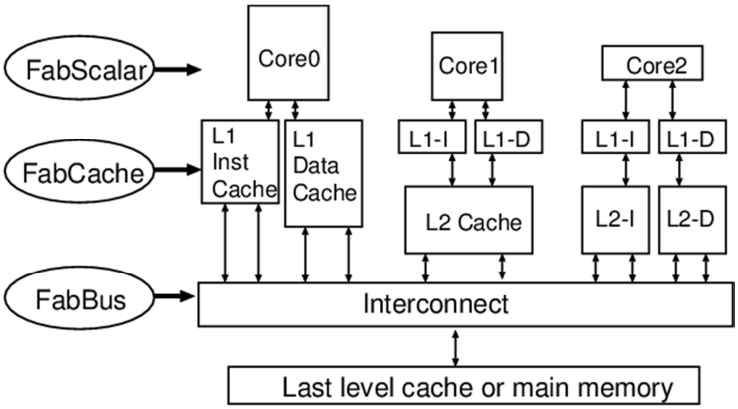

Fig. 1. An example of generating a heterogeneous multi-core processor of FabHetero.

3.1. FabScalar

N.K.Choudhary *et al.* propose FabScalar which is a tool-set to automatically generate synthesizable RTL designs of diverse superscalar cores. FabHetero uses FabScalar to design a suitable superscalar core to exploit ILP in a program and program phase. FabScalar can vary fetch width from one to eight, and an instruction fetch can start from any instruction, this means that FabScalar requires consecutive instructions even the instructions are not aligned on a cache line boundary like modern high-performance superscalar processors. Load store unit (LSU) in FabScalar dispatches speculative load instructions up to the number of load store queue (LSQ), a size of LUQ is also parameterizable, for effective out-of-order execution. Caches for FabScalar should satisfy these diversity whatever FabScalar's design space may be.

3.2. FabBus

FabBus is a tool to design flexible shared bus for heterogeneous multi-core processors. In the heterogeneous multi-core processor, the shared bus between caches and cores has been becoming more complex because a suitable implementation for each cache system and core is different. We have proposed FabBus to automate a designing of a flexible shared bus system which is required to design an entire heterogeneous multi-core processor. FabBus is based on AMBA protocol which is released by ARM Holdings and is commonly used in many embedded systems.

4. FabCache

FabCache is an automatic cache generator. And it has many parameters to change not only simple cache capacity but also more complex features such as hierarchy, bus burst transfer length and individual bus width from lower to higher memory hierarchy. Fig. 2 shows an outline of generation logic of FabCache. User can generate various cache system by modifying the parameter files only. And generated cache system can be ported into almost all of existing hardware. Portability of FabCache is explained in Section 6.

This section shows diversity of our FabCache and explains superset strategy which is key technology to implement FabCache effectively.

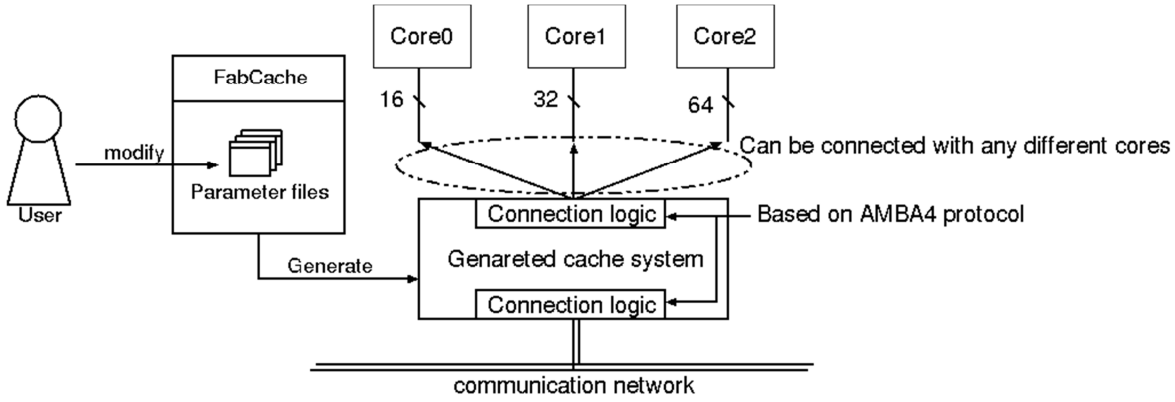

Fig. 2. An outline of generation logic of FabCache.

4.1. Design Diversity

Cache hierarchy: FabCache can generate diverse cache systems up to two-level for each core in a heterogeneous multi-core processor. Namely, FabCache can generate not only a two-level cache design but also one-level cache and non-cache design.

Cache dimensions: Features of caches in all hierarchies can be change such as cache capacity, line size, and associativity to optimize the trade-off among energy consumption, area, and performance.

Specific designs: Each cache hierarchy has specific microarchitectural designs. For example, as for L1 instruction cache, since FabScalar generates one to eight fetch width superscalar cores, L1 instruction cache must support the all fetch width (including not a power of two) in the FabScalar's design space. Another example is that we can choose two cache system types for L2 cache: dedicated L2 caches and unified L2 cache. These aspects help us to design a suitable cache implementation for each hierarchy.

Interface design: We can change the bus width between cache hierarchies or cache system and external design. Arbitrary width transmission helps us to solve the limitation of input/output (I/O) pins easily.

Split transaction: Many modern processors adopt lockup-free cache to hide memory access latency. To support this feature, Miss-Status-Handling-Registers (MSHRs) are introduced. However, MSHRs increase circuit scale of cache system. FabCache can generate simple lockup and high performance lockup-free cache to support wide range from ultra-low-power embedded processor to high performance processor.

4.2. Superset Strategy

Because FabCache is implemented as a superset code in SystemVerilog, all parameterized microarchitectural diversity shares the same source codes in RTL implementation. In contrast, P. Yiannacouras and J. Rose generate RTL codes using a generation script [11]. In this method, a generation script parses parameters and generates the target RTL code dedicated to the given parameters. Using generation script has the advantage of code optimization for each parameter compared with the superset strategy. However, using generation script has a critical problem, extensibility, when we (both developers and users) want to add a new microarchitectural approach or feature. Since we intend to use FabCache for developing a new microarchitecture for heterogeneous multi-core processors, providing extensibility is essential for our goal. Therefore, we adopted superset strategy to implement FabCache. By using superset strategy, adding a new microarchitecture becomes easier because we can directly implement it into RTL code while using generation script requires back-annotation to make the generation script after implementing it once. However, the superset strategy has inherent two concerns: (1) the code may diminish readability if the superset code naively includes various design choice. (2) the unintended hardware remains in the generated design. The former is not so serious problem for user, though we solve the problem to adopt specific implementation and it is available in Ref. [7]. The latter is an open problem, therefore, we have been implementing FabCache avoiding the concerns to adopt some specific implementation described in our previous paper [9], and verify the overhead of unintended hardware by comparing with hand-tuned cache in this paper.

5. FabScalar-Alpha

FabScalar originally supports Portable-Instruction-Set-Architecture (PISA) which is used in SimpleScalar [16], but currently it is ported to MIPS32 [17] ISA and Alpha ISA [18]. MIPS32 is widely used in embedded field, but it is similar to PISA. In our previous work, we evaluate FabCache on the FabScalar which implements 32bit MIPS32 ISA.

In the high-performance computing field, 64-bit processors such as Sun UltraSPARC, Intel IA-64, MIPS MIPS64, and IBM Power were introduced in the earlier time. And then 64-bit processor becomes the main stream in architecture field to achieve high-performance computing in these days. Currently, even mobile processors such as ARM [19] has 64bit capable. Therefore, as a branch project of FabScalar, there is FabScalar-alpha which adopts Alpha 21264 ISA. Alpha 21264 ISA is used in 64-bit processor and it can deal with multi-thread application.

To distinguish them clearly, we call a FabScalar which is used in our previous work “FabScalar-MIPS32”, and one which is introduced in this paper “FabScalar-alpha”.

6. Portability

Although FabCache has been implemented as a part of FabHetero, the cache systems generated by FabCache can be used in various computing fields from embedded to high-performance processor. In general, existing designs may differ its features such as fetch width and individual bus width. Therefore, designer must tune their design, especially in connection logic, to connect every existing design. In FabCache, most design and features of FabCache are parameterized and its bus communication system adopts AMBA4 protocol which is widely used in various architectures. In addition, design of FabCache has an adjustment module such as downsizer to be compatible with various existing hardware. Therefore, we can easily to connect other architecture only to change the bus communication systems by changing its parameter files only. Table 1 shows an example of FabCache’s changeable features. All of features can be changed by given parameters. And FabCache’s connection logic is subset of AMBA4 protocol completely

whatever the given parameter may be. Therefore, generated design by FabCache can connect easily with almost all of third-party hardware which employs AMBA4 protocol. As a first step to show the good portability, we connect FabCache to FabScalar-alpha and estimate design verifications. According to the estimation results, the trend of each result is almost all of the same as the result of our previous work. Estimation results are shown in following section. And therefore, we can decide the generated cache is reasonable. Finally, we can say FabCache has a good portability.

Table 1. An Example of FabCache's Customizable Features

Feature	Range
Cache associative	Direct-Mapping, N(any)-way set associative, Full associative
Cache hierarchy	0~2
Transfer policy	Blocking, Non-blocking
Cache capacity	Any(Power of 2)
Way size	Any
Line size	Any
Word size	Any
MSHR entry size	Any
Burst transfer length	Any
Data bus width	Any
Lifting alignment restriction	Supported
Banked-memory	2-banked memory supported

7. Design Verification and Evaluation

In this section, we show the physical design and power consumption results of caches generated by FabCache by comparing with hand-tuned cache.

7.1. Performance

To show the cache systems generated by FabCache work correctly, we execute 100 million instructions of SPEC2000INT benchmarks on a Verilog HDL simulator. We simulate FabCache with both FabScalar-MIPS32 and FabScalar-alpha, and we confirmed both simulations work perfectly. We also evaluate performance of each benchmark using FabScalar-alpha. Fig. 3 and Fig. 4 show the result of benchmarks with changing associativity from direct mapping (1-way set associative) to 16-way set associative. Each performance is normalized by the result of direct mapping cache of 1024KB. The results from 2-way to 16-way in Fig. 3 are almost same.

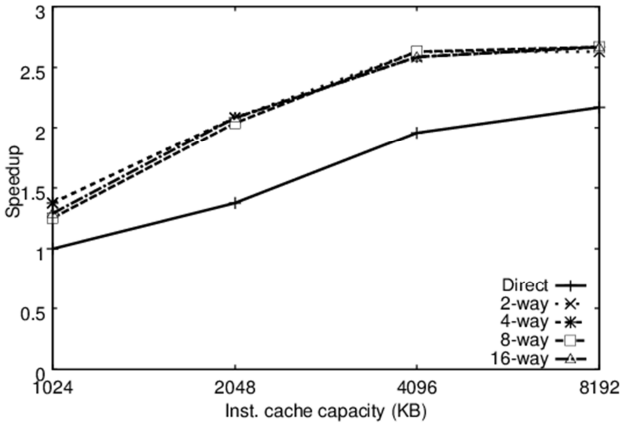

Fig. 3. L1 instruction cache size vs. performance (FabScalar-alpha).

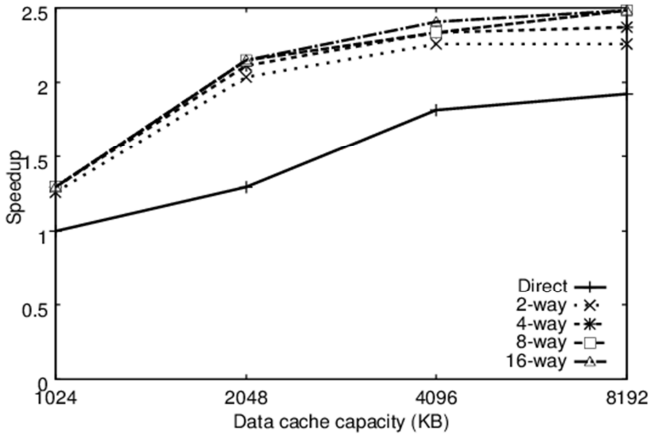

Fig. 4. L1 data cache size vs. performance (FabScalar-alpha).

According to the results in Fig. 3 and Fig. 4, cache systems generated by FabCache work correctly because performance increases as increasing the cache capacity. In addition, we also verify the trend of increasing performance is almost all of the same our previous results of FabScalar-MIPS32 [9].

7.2. Physical Design

7.2.1. L1 instruction cache

We performed physical design using the FabCache. Fig. 5 and Fig. 6 show chip layouts results of L1 instruction cache for FabScalar-MIPS32 and FabCache-alpha, respectively. We implemented L1 instruction cache as superset to keep an RTL code simple described in Section 4.2, thus an LRU memory and some control logics exist even if we implement a direct mapping cache. In this section, we show the physical design generated by FabCache is reasonable even if it has unused LRU memory and control logics.

Table 2. EDA Environment

Phase	EDA tool
Functional verification	Cadence NC-Verilog 09.20-S038
Logical synthesis	Synopsys Design Compiler 2013.03-SP2
Place & route	Synopsys IC Compiler 2012.06
Power estimation	Synopsys XA 2012.06-SP2

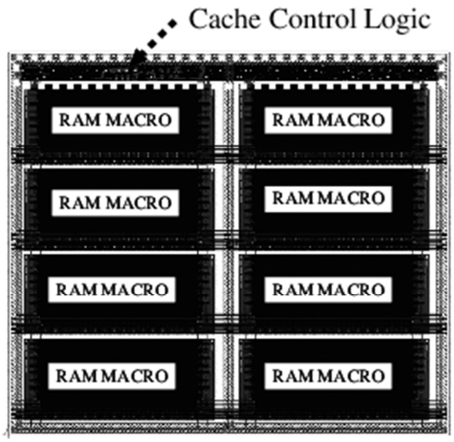

Fig. 5. Chip image of L1 instruction cache (FabScalar-MIPS32) in ref [9].

The both L1 instruction caches consist of two banks interleaved, 8KB capacity, 4 line size and direct mapping. We synthesized L1 instruction caches which generated by FabCache using Rohm 180 nm

technology and Kyoto University standard cell library [15]. Table 2 shows the EDA tools used for designing the physical design and power estimation. The RTL codes generated by FabCache is successfully synthesized, placed and routed. Cache control logic in the Fig. 5 and Fig. 6 are composed of LRU memory, interleaved memory control logic and RAM MACRO indicates a SRAM memory block. According to the layout result shown in Table 3. About L1 instruction cache in FabScalar-MIPS32, the area of cache control logic is 58,496.25um^2 and the whole cache area consists of cache control logic and SRAM memory is 1,668,016.62um^2, the ratio of cache logic area to whole cache area is 3.5%. About the L1 instruction cache in FabScalar-alpha (64bit processor), the area of cache control logic is 68,621.41um^2 and the whole cache area consists of cache control logic and SRAM memory is 1,678,141.78um^2, the ratio of cache logic area to whole cache area is 4.1%. Although the L1 instruction cache includes unused LRU logic, the control logic is enough small and unintended hardware can be negligible.

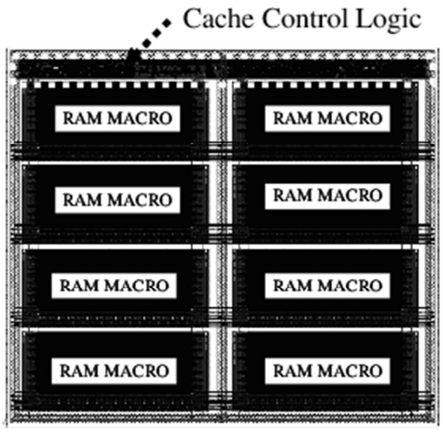

Fig. 6. Chip image of L1 instruction cache (FabScalar-alpha).

Table 3. Area of Control Logic of Physical Designs (FabScalar-MIPS and FabScalar-Alpha)

Design	Area of control logic	Entire cache area	Area ratio of control logic
FabScalar-MIPS	58,496.25um^2	1,668,016.62um^2	3.5%
FabScalar-Alpha	68,621.41um^2	1,678,141.78um^2	4.1%

7.2.2. L1 data cache

We also implemented L1 data cache as superset code. When we implement a blocking cache, MSHR and some control logics exist in the design though these units are wasteful. This section also show an unintended hardware is enough small and it is negligible. We performed a physical design of L1 data cache system for FabScalar-alpha. Fig. 7 shows a chip layout result of L1 data cache. The L1 data cache consists of blocking mechanism, 8KB capacity, 4 line size and direct mapping (1 way set associative). We synthesized L1 data cache which is generated by FabCache using Rohm 180 nm technology and Kyoto University standard cell library also. RTL design generated by FabCache is successfully synthesized, placed and routed. Cache control logic in the Fig. 7 is composed of one entry MSHR register and MSHR control logic, and RAM MACRO indicates SRAM memory. Table 4 shows the area and delay of L1 data cache. 'FabCache' design in the table indicates caches generated by FabCache automatically, 'Hand design' indicates optimized cache which has no overhead of automatic generating. According to the layout results, area and delay overhead of L1 data cache generated by FabCache is only 0.06% and 0.08ns, respectively, compared to hand design. Because the ratio of the cache control logic is very small although FabCache design has overhead of automatic generating, the area and delay is reasonable. We confirm the both FabScalar-MIPS32 and FabScalar-alpha physical design results have almost all of the same trend.

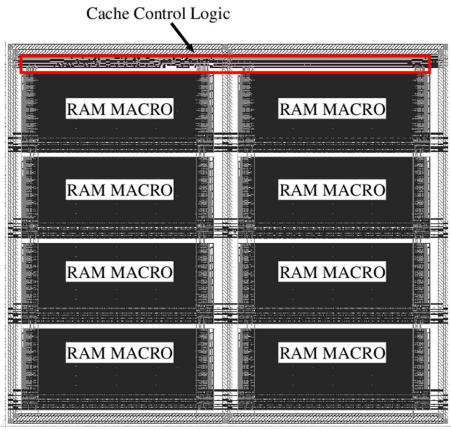

Fig. 7. Chip image of L1 data cache (FabScalar-alpha).

Table 4. Area and Delay of L1 data cache (FabScalar-alpha)

Design	Area of control logic	Entire cache area	Delay	Area ratio of control logic
FabCache	71,304.41um^2	1,680,824.78um^2	2.47ns	4.24%
Hand design	70,227.26um^2	1,679,747.63um^2	2.39ns	4.18%

7.3. Power Consumption

7.3.1. L1 instruction cache

As mentioned Section 7.2, generated instruction cache design from FabCache includes unused control logic. The extra circuits such as LRU memory may cause more power consumption than ordinary direct mapping cache. Therefore, we estimated overhead of power consumption in L1 instruction cache which automatic generated by comparing with that of hand-tuned cache. We execute 50 million instructions of SPEC2000 INT benchmarks and used Synopsys XA G-2012.06-SP2 which is the EDA tool for a power estimation tool. Fig. 8 shows power consumption of L1 instruction cache in FabScalar-MIPS32. Fig. 9 is additional result of this paper and shows power consumption of L1instruction cache in FabScalar-alpha. The value in the both figure are normalized by FabCache design. 'FabCache design' in Fig. 8 indicates direct mapping L1 instruction cache which includes LRU logic because it has an overhead of automatic generation, and 'Hand design' indicates optimized direct mapping L1 instruction cache by hand, this means that there no extra circuits. According to the estimation result, the increase power consumption is less than 0.1%. Therefore, our implementation is reasonable to maintain RTL code simply with negligible overhead.

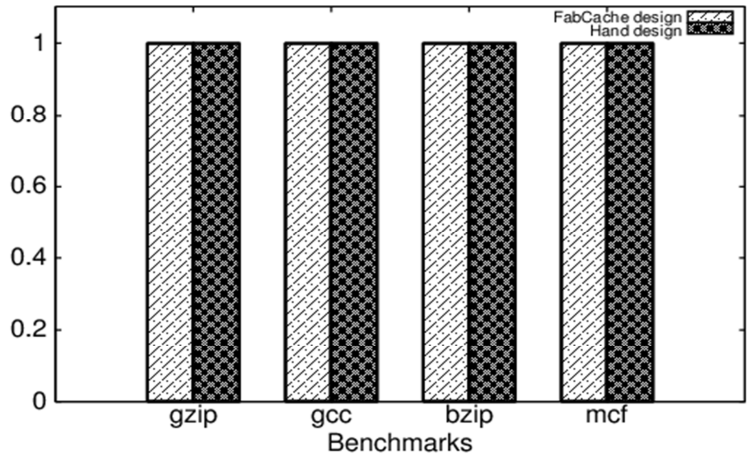

Fig. 8. Power consumption of L1 inst. cache (FabScalar-MIPS32) in ref [9].

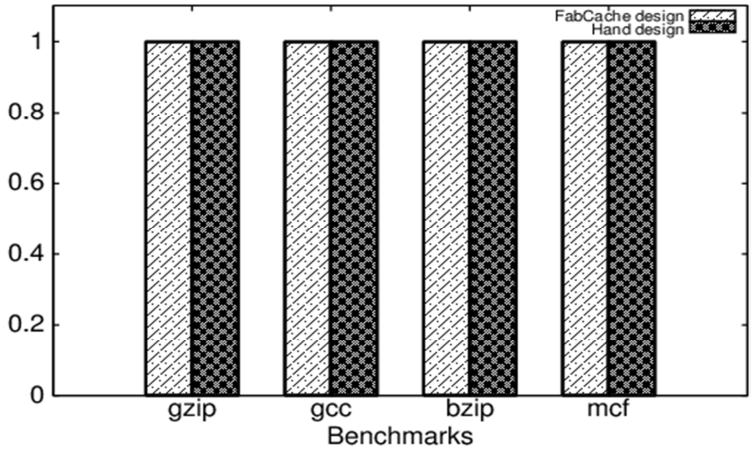

Fig. 9. Power consumption of L1 inst. cache (FabScalar-alpha).

7.3.2. L1 data cache

As mentioned Section 7.2, generated data cache design from FabCache includes unused control logic also. The extra circuits such as MSHR may cause more power consumption than optimized or hand-designed blocking cache. Therefore, we estimated power consumption of L1 data cache which automatic generated by comparing with hand-tuned cache to execute 50 million instructions of SPEC2000 INT benchmarks and Synopsys XA G-2012.06-SP2. Fig. 10 shows the power consumption of L1 data cache in FabScalar-alpha. 'FabCache design' indicates caches generated by FabCache and 'Hand design' indicates optimized cache by hand. According to the estimation results, the increase power which caused by extra circuits is only less than 0.1%. We judged the overhead is enough small to keep RTL code simple and modifiable is reasonable.

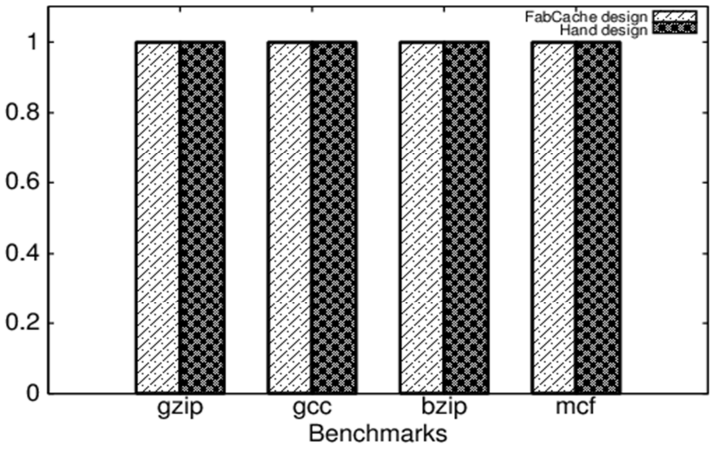

Fig. 10. Power consumption of L1 data cache (FabScalar-alpha).

8. Conclusion

In this paper, we present detail evaluation results and clarify portability of FabCache. According to various evaluation results, cache systems generated by proposed FabCache work correctly and the area, delay and power are reasonable. In particular, each evaluation result of 64-bit processor has almost all of the same trend compared with our previous work which evaluate performance on a 32-bit processor. Therefore, though we implemented FabCache as a part of FabHetero project to generate heterogeneous multi-core system, FabCache can also be used in other computer systems.

As our future work, we are preparing to open FabCache to other researcher and developer to accelerate the studies of not only heterogeneous multi-core system but also cache systems.

Acknowledgment

This work is supported by JSPS KAKENHI Grant Number 24700047 and 15K00074. This work is also supported by the VDEC of the University of Tokyo in collaboration with Synopsys, Inc., Cadence Design Systems, Rohm Corporation and Toppan Printing Corporation.

References

[1] Kumar, R., Tullsen, D. M., Ranganathan, P., Jouppi, N. P., & Farkas, K. I. (2004). Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. *Proceedings of 31st International Symposium on Computer Architecture* (pp. 64-75).

[2] Najaf-Abadi, H. H., & Rotenberg, E. (2008). Configurational workload characterization. *Proceedings of International Symposium on Performance Analysis of Systems and Software* (pp. 147-156).

[3] Greenhalgh, P. (2013). Big.LITTLE processing with ARM cortex-A15 & cortex-A7. ARM White Paper. From http://www.arm.com/ja/files/downloads/big.LITTLE Final.pdf

[4] Choudhary, N. K., Wadhavkar, S. V., Shah, T. A., Mayukh, H., Gandhi, J., Dwiel, B. H., Navada, S., Najaf-abadi, H. H., & Rotenberg, E. (2011). FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. *Proceedings of 38th IEEE/ACM International Symposium on Computer Architecture* (pp. 11-22).

[5] Choudhary, N. K., Wadhavkar, S. V., Shah, T. A., Mayukh, H., & Gandhi, B. J. (2012). FabScalar: Automating superscalar core design. *Micro, IEEE*, *32(3)*, 48-59.

[6] Nakabayashi, T., Sasaki, T., Rotenberg, E., Ohno, K., & Kondo, T. (2012). Research for transporting alpha ISA and adopting multi-processor to FabScalar. *Proceedings of Symposium on Advanced Computing Systems and Infrastructures* (pp. 374-381).

[7] Okamoto, T., Nakabayashi, T., Sasaki, T., & Kondo, T. (2013). FabCache: Cache design automation for heterogeneous multi-core processors. *Proceedings of the 1st International Symposium on Computing and Networking* (pp. 602-606).

[8] Seto, Y., Nakabayashi, T., Sasaki, T., & Kondo, T. (2013). FabBus: A bus framework for heterogeneous multi-core processor. *Proceedings of 28th International Technical Conferench on Circuits/Systems, Computers and Communications* (pp. 254-257).

[9] Okamoto, T., Nakabayashi, T., Sasaki, T., & Kondo, T. (2014). Detail design and evaluation of FabCache. *Proceedings of the Second International Symposium on Computing and Networking*.

[10] de Abreu Silva, B., Cuminato, L. A., & Bonato, V. (2012). Reducing the overall cache miss rate using different cache sizes for Heterogeneous multi-core processors. *Reconfigurable Computing and FPGAs* (pp. 1-6).

[11] Yiannacouras, P., & Rose, J. (2003). A parameterized automatic Cache generator for FPGAs. *Field-Programmable Technology* (pp. 324-327).

[12] Leon 4/GRLIB. (2014). From http://www.gaisler.com

[13] Thomas, D. T. (2012). Designing, verifying and building an advanced L2 Cache sub-system using SystemC. *Proceedings of the Design and Verification Conference and Exhibition*.

[14] Akgul, B. E. S., Mooney, & Parlak, V. J. (2003). Parametrized lock cache generator. *Proceedings of Design, Automation and Test in Europe Conference and Exhibition* (pp. 1138-1139).

[15] Onodera, H., Hirata, A., Kitamura, A., Kobayashi, K., & Tamaru, K. (1999). P2Lib: Process portable library and its generation system. *Journal of Information Processing*, *40(4)*, 1660-1669.

[16] Burger, D., & Austin, T. M. (1997). The SimpleScalar tool set, Version 2.0. *Sigarch Compututer Architecture News*, *25(3)*, 13-25.

[17] MIPS Technology Inc. (2005). MIPS Architecture for Programmers Volume I/II/III Revision 2.50.

[18] Kessler, R. E. (1999). The Alpha 21264 microprocessor. *IEEE Micro*, *19(2)*, 24-36.
[19] Yeung, A., Partovi, H., Harvard, Q., Ravezzi, L. Ngai, J., Homer, R., Ashcraft, M. & Favor, G. (2014). A 3GHz 64b ARM v8 processor in 40nm bulk CMOS technology. *Proceedings of IEEE Solid-State Circuits Conference Digest of Technical Papers* (pp. 110-111).

Takahiro Sasaki received the B.S., M.S. and Ph.D. degrees from Hiroshima City University in 1998, 2000, and 2003 respectively.

He is now assistant professor in the Graduate School of Engineering, Mie University. His research interests are in low-power and high-performance computing, multi-processor architecture, parallel processing and video codec hardware.

Dr. Sasaki is a member of the Institute of Electronics, Information and Communication Engineers and Information Processing Society of Japan.

Takaki Okamoto received the B.S. degrees from Mie University in 2013, and M.S. degree from Mie University in 2015. His current research interests are in low-energy cache memory.

He is currently working on manufacturing control in Sumitomo Wiring Systems, Ltd.

Seiji Miyoshi received the M.E. degree of information engineering from Mie Graduate University in 2016. His research interests are design automation and design method of heterogeneous multi-core processor.

He is currently working on car-electronics in Renesas System Design Co., ltd.

Yuki Fukazawa obtained his M.S. degree in information sciences from Hiroshima City University, Japan, in 2010, and Ph.D. degree in information sciences from Hiroshima City University in 2014.

He is currently a researcher at Graduate School of Engineering, Mie University, since 2014.

His interests are highly reliable BIST architecture and highly efficient operation structure with SIMD-based processor for motion estimation.

Toshio Kondo received the B.S., M.S. and Ph.D. degrees from Nagoya University in 1976, 1978 and 1996, respectively. In 1978, he joined the Musashino Electrical Communication Laboratories, Nippon Telegraph and Telephone Corporation (NTT), Tokyo, Japan, and had been engaged in research on SIMD parallel processors, character recognition systems and MPEG encoder LSIs. Since 2000, he has been a professor at Mie University, Mie, Japan. His current research interests include efficient motion estimators for UHDTV video compression, highly parallel processors for object recognition and processor instruction-set extensions for video processing. Dr. Kondo is a member of the IEEE Circuits and Systems Society, the Institute of Electronics, Information and Communication Engineers and Information Processing Society of Japan.