A Trace-Capable Instruction Cache for Cost-Efficient Real-Time Program Trace Compression in SoC

Chun-Hung Lai, *Student Member*, *IEEE*, Fu-Ching Yang, *Student Member*, *IEEE*, and Ing-Jer Huang, *Member*, *IEEE*

Abstract—This paper presents a novel approach to make the on-chip instruction cache of a SoC to function simultaneously as a regular instruction cache and a real-time program trace compressor, named trace-capable cache (TC-cache). It is accomplished by exploiting the dictionary feature of the instruction cache with a small support circuit attached to the side of the cache. Compared with related work, this work has the advantage of utilizing the existing instruction cache, which is indispensable in modern SoCs, and thus saves significant amount of hardware resource and power consumption. The TC-cache can be configured to work simultaneously as the instruction cache and the trace compressor, named the online mode, or exclusively as the trace compressor, named the bypass mode. The RTL implementation of a 4 KB trace-capable instruction cache, a 4 KB data cache, and an academic ARM processor core has been accomplished. The experiments show that the TC-cache achieves average compression ratio of 90 percent with a very small hardware overhead of 3,652 gates (1.1 percent). It takes only 0.2 percent additional system power for the online mode operation. In addition, the trace support circuit does not impair the global critical path. Therefore, the proposed approach is a highly feasible on-chip debugging/monitoring solution for SoCs, even for cost-sensitive ones such as consumer electronics. Furthermore, the same concept can be applied to the data cache to compress the data address trace as well.

Index Terms—Program trace, compression, cache, real time.

1 Introduction

To debug or analyze a software program running in a processor-based system-on-chip (SoC), it is often necessary to collect the program execution trace directly inside the SoC in real time. However, the problem is that the volume of the real-time trace grows so rapidly that it is impractical to store the trace on chip or send it outside through limited I/O pins. Therefore, various techniques have been proposed to reduce the trace volume directly in hardware. A straightforward approach is to employ a dedicated hardware compressor, which implements some compression algorithms such as the Lempel-Ziv (LZ) algorithm [1]. However, the hardware cost is very high.

On the other hand, we have observed that the on-chip cache is almost an indispensable component in modern SoC. In this paper, we propose a cost efficient approach to reuse the existing on-chip instruction cache to compress the program execution trace with only minor additional support circuitry. The motivation is that when an instruction address is looked up from the instruction cache, the cache performs a table (or dictionary) lookup operation. When the lookup is successful (called cache hit), it is more economical to record the table index than to record the instruction address. This ***trace-capable instruction cache*** (***TC-cache***) is capable of serving as a regular instruction cache and a program trace compressor at the same time, making it more viable than dedicated hardware compressor approaches. To the best of our knowledge, this is the first work to extend the cache's functionality to support real-time program trace compression.

The rest of this paper is organized as follows: Section 2 reviews the related work of hardware-based compression techniques. Section 3 describes the proposed trace-capable instruction cache architecture. Section 4 discusses the data flow of different operation modes of the cache. Section 5 presents the decompression methods. Section 6 presents the experimental results. Section 7 concludes this paper.

2 Related Work

We review the related work from two perspectives: hardware-based trace compression techniques and multi-purposed cache organizations.

2.1 Hardware-Based Trace Compression Techniques

In the first perspective, the hardware-based compression techniques can be characterized into two classes. The first class includes the techniques tailored specific to microprocessors. Most works in this class use the filtering mechanisms to select only certain instructions in the trace for recording. The IEEE-ISTO 5001-2003 Nexus Consortium defines a set of specification for on-chip debugging [2]. The Nexus trace module selects only discontinuous instructions, such as branches, jumps, interrupts, etc., for recording. Many

• *The authors are with the Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung 80424, Taiwan, R.O.C. E-mail: ghlai@eslab.cse.nsysu.edu.tw, fcyangesl@gmail.com, ijhuang@cse.nsysu.edu.tw.*

Manuscript received 30 July 2009; revised 1 May 2010; accepted 13 Aug. 2010; published online 28 Sept. 2010.
Recommended for acceptance by R. Marculescu.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-07-0353.
Digital Object Identifier no. 10.1109/TC.2010.194.

modern microprocessors have incorporated Nexus-compliant modules, such as Tensilica's reconfigurable processor cores with TRAX-PC macrocell [3], Freescale's MPC565 [4], NEC's V850, etc. ARM's Embedded Trace Macrocell (ETM) [5] also uses a similar approach. Hsieh and Huang adopt another similar approach in an embedded debug/trace module for a DSP processor [6]. This module compresses both the program trace and the data trace. The typical compression ratio of the program trace achieved by these related works is around 80-90 percent. To further enhance the compression performance, Hopkins and McDonald-Maier proposed a P2G interface [7] which requires the processor core to provide more signals to the trace unit such as sequential instructions, direct/indirect branch, context switch, etc., for better filtering to achieve a higher compression ratio of 97 percent.

The second class consists of hardware implementations of lossless data compression algorithms. Among all lossless data compression algorithms, dictionary-based approaches such as LZ-related algorithms (e.g., [8], [9]) and other algorithms (e.g., X-MatchPROv4 [10]) are very suitable for hardware implementation [11]. The advantage of this class of hardware is that it is capable of compressing a broad range of data. The disadvantage is that its hardware overhead is higher and its typical compression ratio (30-55 percent) is lower than the first class techniques.

Note that it is possible to adopt a hybrid approach by combining techniques from the two classes. An example is proposed by Kao et al. [12], in which three phases, including a branch/target filter, a slicing module, and a LZ-based hardware compressor are adopted to achieve a much higher compression ratio (99.7 percent).

The above techniques focus on maximizing the compression ratio at the cost of higher hardware overhead. On the other hand, the goal of this paper is to develop a more cost effective solution by reusing the existing hardware resource (the instruction cache) while achieving a considerable compression ratio.

2.2 Multipurposed Cache Organizations

The on-chip cache occupies a significant portion of the total chip area and power consumption. For example, the cache occupies one half of the chip area and consumes 43 percent of power in StrongARM SA-110 [13]. Therefore, it is highly motivated to achieve a better use of the cache. A most common practice is to modify the cache organization such that it can be configured as a scratch-pad memory (SPM) or as a combination of a smaller cache and a smaller SPM. There are some situations where it is better for the cache to serve as a SPM. For example, in an application specific environment where the software behavior is well predictable, it might be more efficient in performance and energy for the software to control which portion of the software and data to be brought on chip in the SPM rather than by the cache mechanism. Many commercial microprocessor/DSP cores such as Samsung S3C4510B [14], Analog Devices ADSP-BF537 [15], and Texas Instruments TMS320C6711 [16] have such kind of cache organization.

Many researchers have been working on the cache-SPM issues. For example, Ranjan et al. [17] and Chiou et al. [18] proposed techniques to partition between the cache and SPM. Hillenbrand and Henkel proposed to use a SPM to emulate a cache with the help of a direct memory access (DMA) and a small controller [19]. Baron et al. proposed a real-time cache implementation which can be used as a cache mode or a standard memory mode [20]. On the other hand, Cullison and Wagner proposed a multipurposed cache memory which can serve as either a boot memory or a cache at different times in a multiprocessor environment [21].

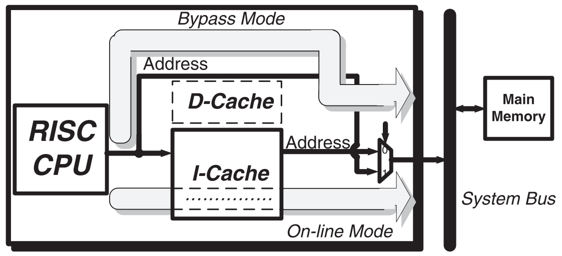


Fig. 1. Generic processor and cache integration and the online and bypass modes.

Among the multipurposing efforts for cache, our work is the first approach, to the best of our knowledge, to use the cache memory to perform real-time program trace compression.

3 Trace-Capable Instruction Cache Architecture

Depending on the specific need of the application program running on the CPU, the instruction cache could be turned on or off as shown in Fig. 1. To simplify the discussion and illustration, we will focus on RISC CPU's (having constant instruction word width) throughout the rest of the paper. The instruction cache is usually turned on to speed up the memory operations. In this case, the cache can be used simultaneously for the program execution and the trace compression, which is called the ***online mode***. On the other hand, the cache may be turned off for some special applications, such as a real-time system which cannot tolerate the performance uncertainty caused by the cache operations [22]. When the cache is turned off, all of the memory references will bypass the instruction cache and it can then be used solely for the trace compression purpose. This case is called the ***bypass mode***.

The architecture of the TC-cache is illustrated in Fig. 2. The shaded area depicts the ***trace-capable cache support circuit*** (***TCS circuit***), which is integrated with the CPU and the instruction cache. The TCS circuit consists of a trace cache mode register, a branch/target identifier, and a trace generator. The trace cache mode register defines the two operation modes for the instruction cache: the bypass mode (i.e., trace $\text{cache mode} = 1$) and the online mode (i.e., trace $\text{cache mode} = 0$).

The data cache operates as usual; it does not involve in the program trace compression. The rest of the components in the figure are described in the following sections:

3.1 Branch/Target Identifier

The program trace consists of segments of consecutive instruction addresses. It suffices to record only the head (target) address and the tail (branch) address, in order to

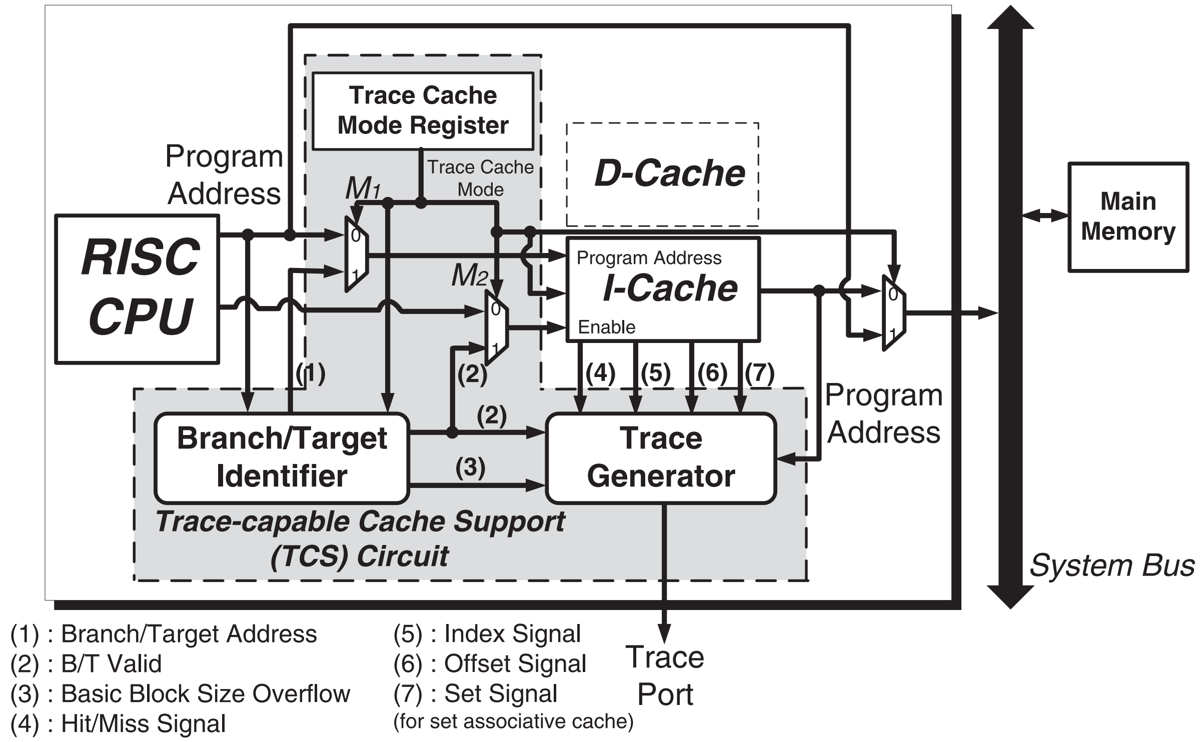


Fig. 2. The proposed trace-capable instruction cache architecture.

reduce the trace size. The ***branch address (B address)*** is the address of a branch instruction. The ***target address (T address)*** is the address of the target of a branch instruction. The purpose of the B/T identifier is to select the B address and the T address. Fig. 3 shows the B/T identifier for a RISC CPU. If the offset of previous address and the current address is not equal to a constant offset, then a branch has been identified and the branch signal is asserted. This signal is latched to identify the coming target address. For a CISC CPU (having variable instruction word width), a help signal from the instruction decoder is needed to identify the B address and the B/T identifier in Fig. 3 also needs to be modified to accommodate this help signal.

The ***basic block size overflow signal (BBO signal)*** in Fig. 3 is used to decide whether the consecutive program address references cause cache wrapping around. The counter will increment upon every incoming address, and will be reset when a branch occurs. The max size denotes the maximal length of consecutive address stream that the instruction cache can accommodate without causing the

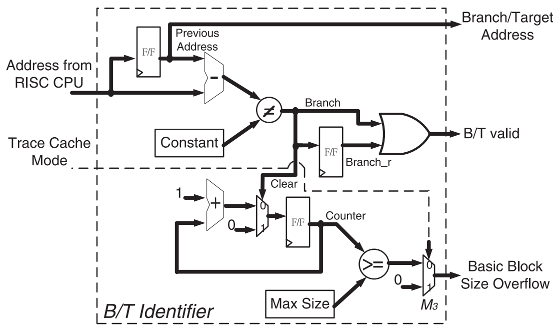


Fig. 3. Branch/target identifier for an RISC CPU.

wrapping around. The B/T identifier asserts a BBO signal when the counter value is greater than or equal to the max size. The BBO signal is fixed at zero in the bypass mode as indicated in the multiplexer M_3 of Fig. 3. The use of the BBO signal will be explained in Section 4.2.

3.2 Cache Module

The input of the program address of the instruction cache comes from two possible sources. The address comes from the CPU when in the online mode or from the B/T identifier when in the bypass mode. The selection is made by the multiplexer M_1 in Fig. 2. The multiplexer M_2 selects the sources of the cache enable input using the same condition as M_1. The enable signal comes from the CPU internal cache control register when in the online mode or from the B/T valid signal of the B/T identifier when in the bypass mode. Four internal signals of the instruction cache are exported to the trace generator: hit/miss signal, (cache line) index signal, (cache line) offset signal, and set signal (only for set associative cache).

When in the online mode, the instruction cache operates normally. Each instruction address is sent to the cache and the CPU reads the corresponding instruction from the cache. The cache performs a refill from the main memory when a read-miss occurs.

When in the bypass mode, the instruction cache operates differently from the normal case. For the CPU, the instruction cache is bypassed. The instructions are directly read from the main memory. On the other hand, only the addresses generated by the B/T identifier are sent to the cache to perform the read operations. The instruction cache is active only when the B/T valid signal of the B/T identifier is true. In addition, only the ***tag*** RAM of the instruction cache is operative while the data RAM of the instruction cache is not

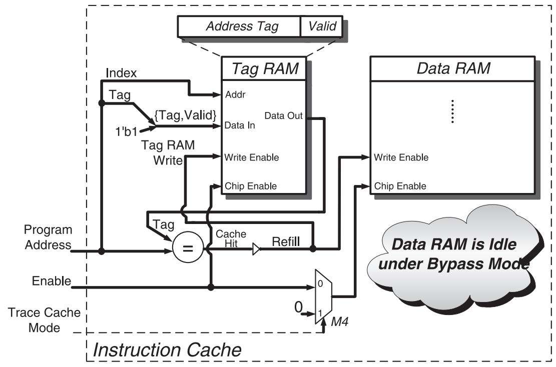


Fig. 4. The operations of tag RAM and data RAM for the instruction cache.

used. As indicated in the multiplexer M_4 of Fig. 4. The data RAM of the instruction cache is turned off when in the bypass mode, and this is made by setting the chip enable signal of the data RAM to zero. When there is a cache hit, the cache hit data are simply ignored. When there is a miss, the tag of the address is written to the tag portion of the corresponding cache line to perform a bookkeeping operation. Next time, when the tag of an incoming address reference is matched in the cache, a cache hit occurs despite the incoming address is first accessed.

3.3 Trace Generator

The trace generator is shown in Fig. 5, the cache lookup result is recorded by the trace packing module whenever the B/T valid signal output from the B/T identifier is true. The BBO signal output from the B/T identifier is used to select the hit/miss signal exported from cache or the forced miss signal. When the BBO signal is true, the trace generator is forced to record a miss whether the cache lookup result is a cache hit or miss as indicated in the multiplexer M_5. On the other hand, when a cache miss occurs or the BBO signal is true, the trace generator records the full address; otherwise, it records index, offset, set as indicated in the multiplexer M_6.

Therefore, when a cache hit occurs, the trace generator records a hit and the index, offset, set values. If a cache miss occurs or the BBO signal is true, the trace generator records a miss and the full address. The trace record length when cache hits depends on the cache configuration, in the basic setting (i.e., direct map cache and cache line size is equal to one), only the index value is recorded. According to this result, the trace file format will be like as follows: M(address), M(address), H(index), H(index), M(address), etc., where **H** denotes hit and **M** denotes miss.

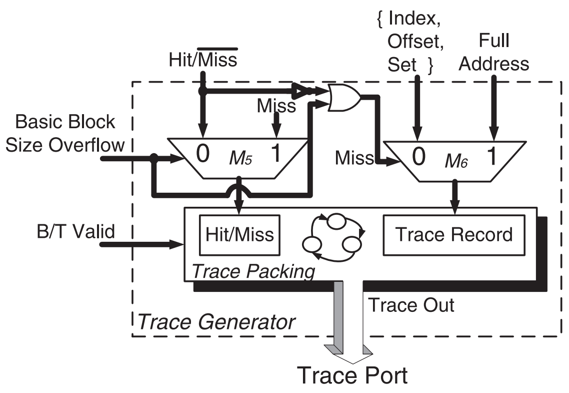


Fig. 5. Trace generator.

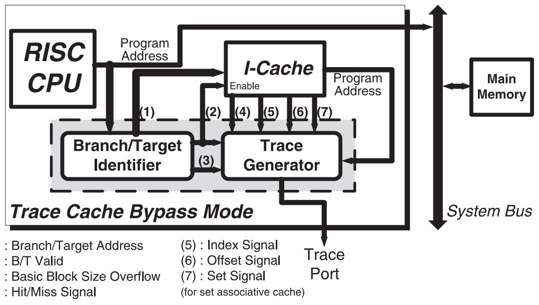


Fig. 6. Data path for the bypass mode ($\mathrm{M1} = 1$, $\mathrm{M2} = 1$).

Trace generator will transmit the trace information to off-chip host to save these information as trace files. Trace generator could transmit the data to the host via JTAG port or other transmission protocols such as IEEE-ISTO Nexus 5001 AUX port.

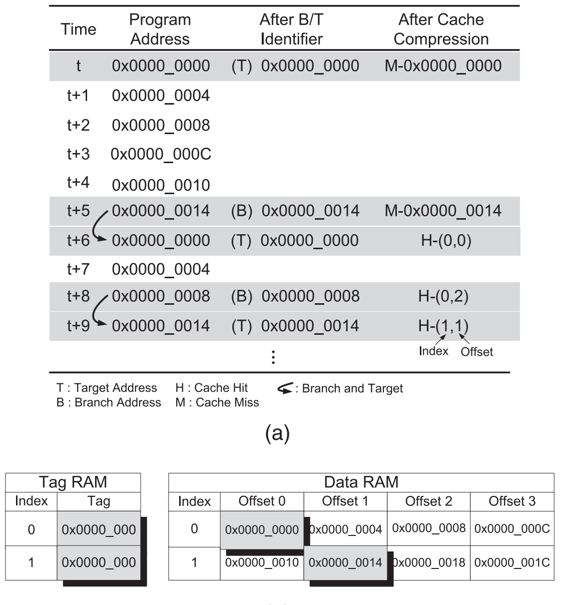
4 Data Paths and Operations of the Bypass Mode and the Online Mode

4.1 The Bypass Mode

The cache operated under the bypass mode is configured by setting the multiplexers M_1 and M_2 to 1. The corresponding data path is shown in Fig. 6. Therefore, the cache address reference input is the B/T address, and the cache enable input is the B/T valid signal output from the B/T identifier. This indicates that the cache is active only whenever a B/T address reference is identified by the B/T identifier. At the same time, the cache lookup result is processed by the trace generator when the B/T valid signal is active.

An example of the compressed trace information in the bypass mode is shown in Fig. 7a. Column 1 is the instruction sequence. Column 2 is the original program address trace sent by the CPU. Column 3 is the B/T address trace after ignoring the contiguous addresses. Column 4 is the address trace after cache compression. When a cache miss occurs, the trace generator records a miss and the full address, such as the M-0x0000_0000 record at instruction sequence t. If a cache hit occurs, the trace generator records a hit and (index,offset) value, such as the H-(0,0) at $t + 6$.

The cache content after the execution of these address references is shown in Fig. 7b. The cache configuration in the figure is a 32-byte direct map cache whose line size is 16 bytes. Note, the data RAM of the instruction cache is not used for the bypass mode because only the tag hit and miss information generated by the tag RAM is necessary for trace compression in the bypass mode. In the bypass mode, the cache is active only when the current address reference is a T address or a B address. The cache misses at instruction sequence t and $t + 5$ (in Fig. 7a) will update the tag portion of the corresponding cache line $index_0$ and $index_1$, respectively, as in the gray shade of Fig. 7b. When the tag of an incoming address reference is matched in the cache, a cache hit record occurs, such as at instruction sequence $t + 6, t + 8$, and $t + 9$. The content of the data RAM represents that instruction addresses 0x0000_0000 through 0x0000_000C and 0x0000_0010 through 0x0000_001C occupy the cache line $index_0$ and $index_1$, respectively.



Time	Program Address	After B/T Identifier	After Cache Compression
t	0x0000_0000	(T) 0x0000_0000	M-0x0000_0000
t+1	0x0000_0004		
t+2	0x0000_0008		
t+3	0x0000_000C		
t+4	0x0000_0010		
t+5	0x0000_0014	(B) 0x0000_0014	M-0x0000_0014
t+6	0x0000_0000	(T) 0x0000_0000	H-(0,0)
t+7	0x0000_0004		
t+8	0x0000_0008	(B) 0x0000_0008	H-(0,2)
t+9	0x0000_0014	(T) 0x0000_0014	H-(1,1)
		⋮	

Index
Offset
T : Target Address
B : Branch Address
H : Cache Hit
M : Cache Miss
: Branch and Target
(a)

Tag RAM	
Index	Tag
0	0x0000_000
1	0x0000_000

Data RAM				
Index	Offset 0	Offset 1	Offset 2	Offset 3
0	0x0000_0000	0x0000_0004	0x0000_0008	0x0000_000C
1	0x0000_0010	0x0000_0014	0x0000_0018	0x0000_001C

Fig. 7. (a) Example trace for the bypass mode; (b) cache content of the example trace.

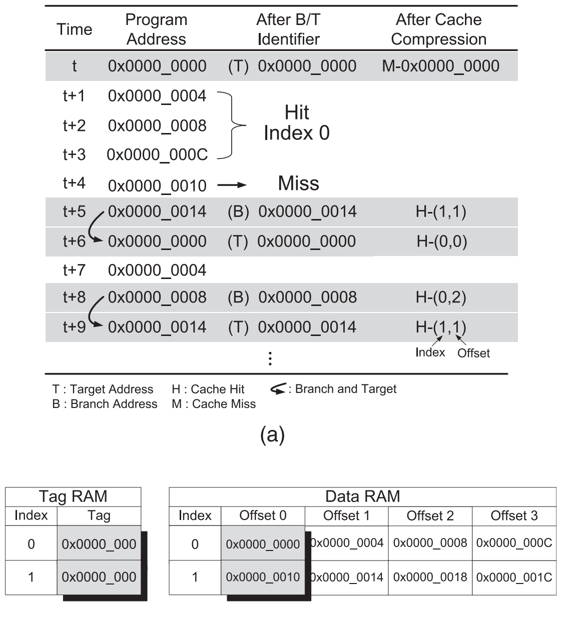
The trace size in Fig. 7 is analyzed as follows: with the full address being 32 bits, the Hit/Miss signal being one bit, the index signal being one bit, and the offset signal being two bits. Therefore, the original full trace file size will be $(32 \times 10) = 320$ bits, and the compressed trace file size will be $(33 \times 2) + (4 \times 3) = 78$ bits, with a compression ratio of 75.6 percent ((320-78)/320 percent).

4.2 The Online Mode

The cache operated under the online mode is configured by setting the multiplexers M_1 and M_2 to zero. The corresponding data path is shown in Fig. 8. Therefore, the cache address reference input is the CPU instruction address bus, and the cache enable input is the cache enable bit of the cache control register. This indicates that the memory address references are used as the input to B/T identifier and cache at the same time. Whenever the B/T identifier outputs the B/T valid signal as true, the cache lookup result will be processed by the trace generator.

The compressed trace information and the cache content in the online mode are shown in Fig. 9. The meaning of each column and the cache configuration is the same as described in Section 4.1. For illustration, the data RAM here is also intended to indicate the occupation of the cache line, and the content represents a specific instruction address rather than the corresponding data. In the online mode, the cache content update is different from the bypass mode. The cache is active during the program execution, and the cache content is updated as long as a cache miss occurs, even when the current address reference is neither a T address nor a B address. It implies that the B/T addresses could be brought into cache in advance before they are accessed. As in the gray shade of Fig. 9b, the cache miss at instruction sequence $t + 4$ (in Fig. 9a) will update the tag portion and data portion of the corresponding cache line $index_1$, causing a cache hit at instruction sequence $t + 5$. This is different from the case in the bypass mode in which a cache miss is resulted at $t + 5$ instead, due to the fact that the cache is inactive and the cache content is not updated at $t + 4$, as shown in Fig. 7a. The compression ratio of this online mode example is 84.6 percent ((320-49)/320).

The BBO signal in Fig. 3 is only used in the online mode, and Fig. 10 illustrates an example. As shown in Fig. 10a, the cache miss at instruction sequence t will cause cache hits at $t + 1, t + 2$, and $t + 3$ in the cache line $index_0$, and the cache miss at instruction sequence $t + 4$ will cause cache hits at $t + 5, t + 6$, and $t + 7$ in the cache line $index_1$. And again, the cache miss at instruction sequence $t + 8$ will cause a cache hit at ***t+9*** in the cache line $index_0$. However, the cache refill at instruction sequence $t + 8$ will cover the original content in the cache line $index_0$ as shown in the gray shade of Fig. 10b. In this case, the cache has duplicately used the same index and offset value within a basic block, for a basic block is too large which causes the cache refill wrapping around. And this will lead to the trace reconstruction error. Consequently, the trace generator is forced to record a miss and the full address at instruction sequence $\mathbf{t + 9}$.



Time	Program Address	After B/T Identifier	After Cache Compression
t	0x0000_0000	(T) 0x0000_0000	M-0x0000_0000
t+1	0x0000_0004		
t+2	0x0000_0008		
t+3	0x0000_000C		
t+4	0x0000_0010		
t+5	0x0000_0014	(B) 0x0000_0014	H-(1,1)
t+6	0x0000_0000	(T) 0x0000_0000	H-(0,0)
t+7	0x0000_0004		
t+8	0x0000_0008	(B) 0x0000_0008	H-(0,2)
t+9	0x0000_0014	(T) 0x0000_0014	H-(1,1)
		⋮	

Hit
Index 0
Miss
Index
Offset
T : Target Address
B : Branch Address
H : Cache Hit
M : Cache Miss
: Branch and Target
(a)

Tag RAM	
Index	Tag
0	0x0000_000
1	0x0000_000

Data RAM				
Index	Offset 0	Offset 1	Offset 2	Offset 3
0	0x0000_0000	0x0000_0004	0x0000_0008	0x0000_000C
1	0x0000_0010	0x0000_0014	0x0000_0018	0x0000_001C

Fig. 9. (a) Example trace for the online mode; (b) cache content of the example trace.

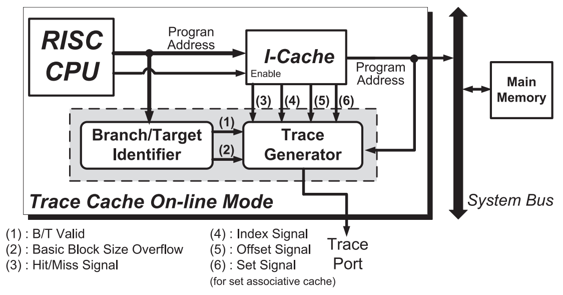
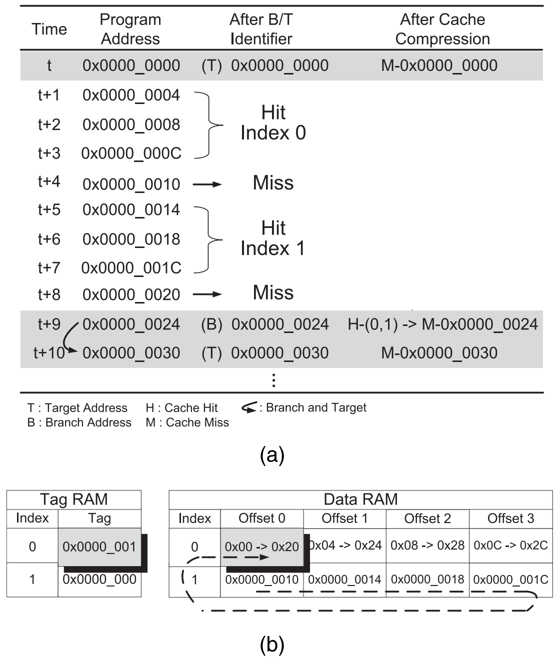


Fig. 8. Data path for the online mode (M1=0, M2=0).

4.3 Comparison of the Bypass Mode and the Online Mode

The difference in the cache behaviors of the bypass mode and the online mode is analyzed in Table 1. The instruction cache



Time | Program Address | After B/T Identifier | After Cache Compression
t 0x0000_0000 (T) 0x0000_0000 M-0x0000_0000
t+1 0x0000_0004
t+2 0x0000_0008
t+3 0x0000_000C
Hit Index 0
t+4 0x0000_0010 Miss
t+5 0x0000_0014
t+6 0x0000_0018
t+7 0x0000_001C
Hit Index 1
t+8 0x0000_0020 Miss
t+9 0x0000_0024 (B) 0x0000_0024 H-(0,1) -> M-0x0000_0024
t+10 0x0000_0030 (T) 0x0000_0030 M-0x0000_0030
T : Target Address
B : Branch Address
H : Cache Hit
M : Cache Miss
: Branch and Target
(a)
Tag RAM
Index Tag
0 0x0000_001
1 0x0000_000
Data RAM
Index Offset 0 Offset 1 Offset 2 Offset 3
0 0x00 -> 0x20 0x04 -> 0x24 0x08 -> 0x28 0x0C -> 0x2C
1 0x0000_0010 0x0000_0014 0x0000_0018 0x0000_001C
(b)

Fig. 10. (a) Overflow problem in the online mode; (b) cache wrap around.

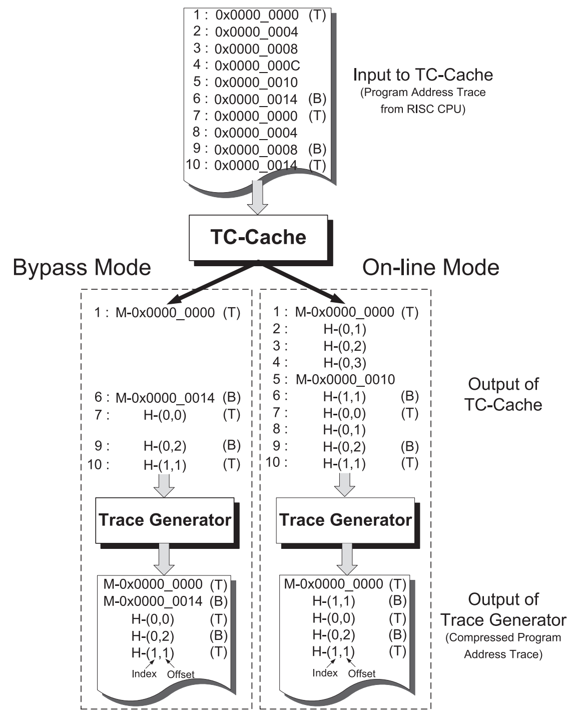


Fig. 11. Comparison of the program address traces in the bypass mode and the online mode.

is looked up only for B/T addresses in the bypass mode, whereas is looked up for every address in the online mode. The cache content is updated only for misses caused by the B/T addresses, whereas is updated for misses caused by both B/T and non-B/T addresses in the online mode. However, the cache output signals are considered valid as the final compressed trace record only when the addresses are branches or targets (determined by the B/T identifier in Fig. 3) for both the cases of the bypass and online modes.

Fig. 11 compares the difference in the compression flow between the two modes by combining examples from Figs. 7 and 9. At the top of the figure, there is the program trace from the CPU, which consists of 10 instructions. The output of the TC-cache in the bypass mode and the online mode is shown in the left and right sides, respectively. For the online mode, there are 10 records coming out of the the cache, out of which five final records are produced by the trace generator. For the bypass mode, there are only five records out of the cache, and all these five records are output as the final records by the trace generator. Note that although the two modes have the same number of final records, they are different in terms of their natures as hits and misses: the online mode has one miss record and four hit records, whereas the bypass mode has two miss records and three hit records. The reason is due to the difference in the cache operations as compared in Table 1. Therefore, the decompression mechanism is different for the two modes and will be explained in the next section.

TABLE 1
Comparison of Operations of the
Bypass Mode and the Online Mode

	Bypass Mode	On-line Mode
Cache Lookup Activity	B/T Address	Every Address Reference
Cache Content Update	Miss of B/T Address	Miss occur
Cache Output Validity (4,5,6,7 signals in Fig. 2)	B/T Address	B/T Address

4.4 Extension to Data Trace

The proposed TC-cache architecture can be extended to trace the addresses of data access by connecting the TCS circuit in Fig. 2 between the data address bus of the CPU and the input of the data cache. There are two ways to accomplish the connection. First, we can use only one copy of the TCS circuit and multiplex it between the instruction address bus and the data address bus. The advantage is that only copy of the TCS circuit is used. The disadvantage is that only one type of trace, either the program address or the data address, can be captured at a time. On the other hand, we can use one TCS circuit for the instruction cache and another TCS circuit for the data cache. This configuration allows that both the program address trace and the data address trace can be captured simultaneously. Either way, the data cache can serve as both a regular data cache and a data trace compressor.

There is one limitation when applying our technique to the data cache: our technique captures only the data address trace but not the data value trace. However, this is not considered as a significant limitation. Since the data values are difficult to compress, due to their less degree of regularity as compared to addresses, the data value trace capability is usually considered as optional, as in ARM ETM [23] and Hopkins and McDonald-Maier [7]. On the other hand, because of the nature of lower compression efficiency of the data value trace, the Nexus tracer even makes no attempt to compress it [7].

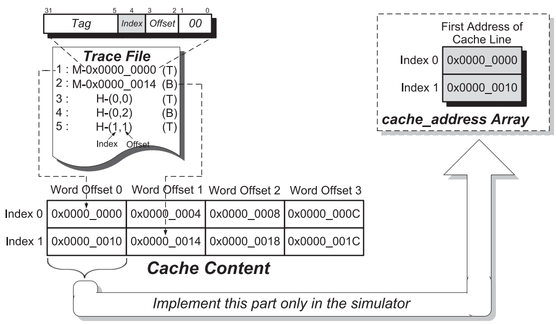


Fig. 12. Reconstruction of the TC-cache for the bypass mode trace (cache: a 32-byte direct map cache with line size of four words (16 bytes)).

5 Decompression for the Compressed Program Trace

Once the compressed program trace is read back to a host computer, it can be decompressed by software for further analysis. The goal of the program trace decompression is to reconstruct the program addresses that are output from the CPU. To simplify our task, it suffices to reconstruct only the B/T addresses because the addresses between a target and its corresponding branch can be mechanically derived by incrementing the address of the target until the address of the branch.

The fundamental of the decompression is to reconstruct the content of TC-cache. Therefore, the decompression software is basically a TC-cache simulator. However, it operates quite differently from a regular cache simulator which takes in the address references then outputs the cache hit/miss information. The TC-cache simulator takes in the cache miss and hit records from the trace file, reconstructs the cache content according to the records, and then outputs the B/T addresses accordingly. Note that the content of the simulated cache is the addresses instead of instructions since we are only interested in the addresses.

Let's first look at the decompression of the compressed bypass mode trace using the example in Fig. 11. The reconstruction of the TC-cache is illustrated in Fig. 12 where the TC-cache is a 32-byte direct map cache with a line size of four words. The first record in the compressed trace is a miss record, and therefore, its address information is entirely encapsulated in the record 0x0000_0000. Accordingly, the first decompressed B/T address is 0x0000_0000. In addition, this address is mapped to the cache line in $index_0$. At this point, we know that this line accommodates instruction addresses 0x0000_0000 through 0x0000_000C according to the specification of this TC-cache. The second record is also a miss record 0x0000_0014, which is the second decompressed B/T address. This address is mapped to the cache line in $index_1$, which accommodates the addresses ranging from 0x0000_0010 through 0x0000_001C. At this point, the entire content of the TC-cache is known. The third record is a hit record, with information of `(index=0, offset=0)`. So, we can look up the TC-cache at the corresponding cache line and offset, and find that the corresponding B/T address is 0x0000_0000. Similarly, the fourth and fifth records are both hit records and their corresponding addresses are 0x0000_0008 and 0x0000_0014, respectively.

Note that the TC-cache simulator can be simplified by keeping only the first address of each cache line since the addresses are consecutive within a cache line. The data structure to keep such first addresses is called cache_address array in our simulator, shown in the upper right side of Fig. 12. When the TC-cache is a set associative cache, each set is treated the same as a direct map cache, with the set being selected by the set information in the hit record. For a miss record, the corresponding address is written to a set according to the replacement policy of the TC-cache.

In the following sections, the decompression algorithms for the bypass mode and the online mode are presented, respectively.

5.1 Decompression Flow of the Bypass Mode

The decompression flow for the bypass mode is shown in Fig. 13. At the top, there is the trace file, consisting of branch and target records which interleave with each other as shown in the figure. It is assumed that the trace file starts with a target record. At $step_1$ and $step_2$, a record is read in and checked for its type. For a target record, $step_3$ checks if it is a hit or a miss. For a miss target, its address is encapsulated in the miss information in the current target

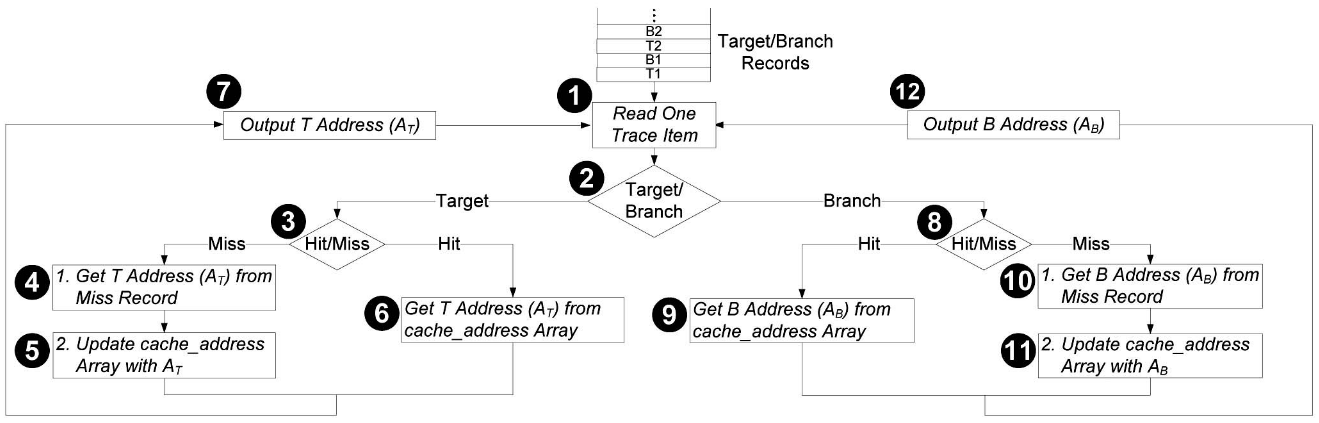

B2
T2
B1
T1
Target/Branch Records
1
Read One Trace Item
7
Output T Address (A_T)
12
Output B Address (A_B)
2
Target/ Branch
Target
Branch
3
Hit/Miss
Miss
Hit
4
1. Get T Address (A_T) from Miss Record
5
2. Update cache_address Array with A_T
6
Get T Address (A_T) from cache_address Array
8
Hit/Miss
Hit
Miss
9
Get B Address (A_B) from cache_address Array
10
1. Get B Address (A_B) from Miss Record
11
2. Update cache_address Array with A_B

Fig. 13. Decompression flow diagram for the bypass mode.

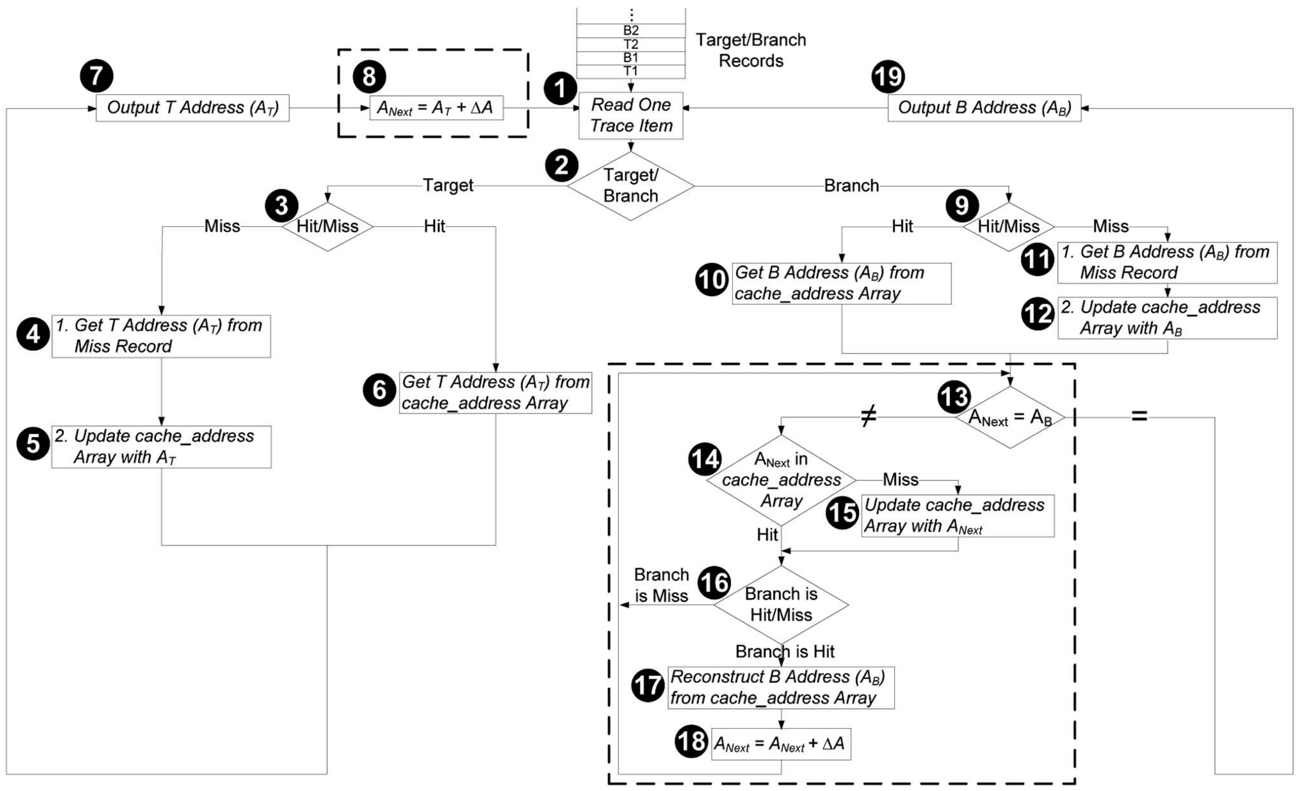

B2
T2
B1
T1
Target/Branch Records
7 Output T Address (A_T)
8 A_Next = A_T + ΔA
1 Read One Trace Item
19 Output B Address (A_B)
2 Target/ Branch
Target
Branch
3 Hit/Miss
Miss
Hit
9 Hit/Miss
Hit
Miss
11 1. Get B Address (A_B) from Miss Record
10 Get B Address (A_B) from cache_address Array
12 2. Update cache_address Array with A_B
4 1. Get T Address (A_T) from Miss Record
6 Get T Address (A_T) from cache_address Array
5 2. Update cache_address Array with A_T
13 A_Next = A_B
≠
=
14 A_Next in cache_address Array
Miss
15 Update cache_address Array with A_Next
Hit
Branch is Miss
16 Branch is Hit/Miss
Branch is Hit
17 Reconstruct B Address (A_B) from cache_address Array
18 A_Next = A_Next + ΔA

Fig. 14. Decompression flow diagram for the online mode.

record. Consequently, $step_4$ looks up the T address from the miss information and then $step_5$ updates the cache_address array with the T address. On the contrary, for a hit target which infers its address has existed in the cache_address array, then $step_6$ looks up the T address from the cache_address array. $Step_7$ completes the processing of the current target record by outputting the T address and then goes back to $step_1$ to process the next record. For a branch record, $step_8$ through $step_{12}$ handle it in a way similar to the case of the target record.

5.2 Decompression Flow of the Online Mode

The decompression flow of the online mode is illustrated in Fig. 14. The steps which are different from those of the bypass mode are marked by the dashed boxes, i.e., $step_8$, and $step_{13}$ through $step_{18}$.

The processing of the target record is similar to the bypass mode. However, at the end of the processing, $step_8$ updates a temporary variable A_{Next} by adding an instruction address increment constant ΔA to the T address A_T. After this step, A_{Next} points to the next instruction address of the current T address. This variable is used to track the missing sequential address references (i.e., instructions 2 through 5 in Fig. 11) between a target instruction and a branch instruction, and to consider the effect of these references which update the cache content accordingly to reconstruct a valid B address.

For a branch record, $step_9$ checks whether it is a hit or a miss. If it is a miss branch, similar to the case of the bypass mode, the B address can be obtained from the miss information in the branch record in $step_{11}$. If it is a hit branch, the B address is looked up from the cache_address array in $step_{10}$. However, it might not be a correct one and will go through further modification by subsequent $step_{13}$ through $step_{18}$. The purpose of these modification steps is to reconstruct the instruction addresses (and their effects on the cache), which are missing in the online mode trace, between a target instruction and a branch instruction.

$Step_{13}$ checks if A_{Next} falls within the address range of the preceding target instruction and the current branch instruction. If yes, it then goes into an inner loop ($step_{14}$ through $step_{18}$). $Step_{14}$ checks if the cache line which accommodates A_{Next} exists in the cache_address array. If not, which infers a cache miss occurs and then the array is updated with A_{Next} in $step_{15}$. These two steps are used to emulate that the cache is looked up and the cache content is updated constantly between a T address and a B address. For a hit branch, after considering the effect of A_{Next} on cache in $step_{15}$, the B address is regenerated in $step_{17}$. This step is used to generate a valid B address that refers to an identical cache content as the branch instruction executing on the target hardware. $Step_{18}$ updates A_{Next} by pointing it to the next address and then goes back to $step_{13}$ to check for the necessity of the next loop iteration.

Once the loop has completed, the exit path of the $step_{13}$ is taken. Now, the variable A_B has the correct B address and is the output in $step_{19}$.

From the illustration above, let's consider the compressed online mode trace in Fig. 11. The first record is a miss target, the T address is obtained from the miss information 0x0000_0000, and the cache line in $index_0$ is

TABLE 2
Gate Count Analysis of the
Trace-Capable Cache (UMC 0.18-μm)

Original CPU + Cache*	N_ARM[24]	51917
	Cache[25]	270000
	Total (T_1)	321917
Trace-capable Cache Support (TCS) Circuit	Cache Modification	872
	B/T Identifier	1133
	Trace Generator	1265
	Mode Register	8
	Mux	374
	Total (T_2)	3652 (1.1% w.r.t. T1)

* 4KB direct map instr. cache, 4KB direct map data cache; both with line size of 4 words

TABLE 3
Timing Analysis of the Trace-Capable Cache

	Timing (ns)	
	Global Critical Path	Instr. Cache Path
CPU + Cache	7.3	6.0
CPU + Trace Capable Cache	7.3	6.3

TABLE 4
Hit Size of the Specific Cache Configuration

Cache Size	Hit Size (Bits)		
	Cache Line Size (Words)		
	2	4	8

1K	Index = 7 Bits
	Offset = 1 Bit
		Index = 6 Bits
		Offset = 2 Bits
			Index = 5 Bits
			Offset = 3 Bits

2K	Index = 8 Bits
	Offset = 1 Bit
		Index = 7 Bits
		Offset = 2 Bits
			Index = 6 Bits
			Offset = 3 Bits

4K	Index = 9 Bits
	Offset = 1 Bit
		Index = 8 Bits
		Offset = 2 Bits
			Index = 7 Bits
			Offset = 3 Bits

8K	Index = 10 Bits
	Offset = 1 Bit
		Index = 9 Bits
		Offset = 2 Bits
			Index = 8 Bits
			Offset = 3 Bits

16K	Index = 11 Bits
	Offset = 1 Bit
		Index = 10 Bits
		Offset = 2 Bits
			Index = 9 Bits
			Offset = 3 Bits

* Example of a direct map cache

filled with instruction addresses 0x0000_0000 through 0x0000_000C in $step_5$. In addition, A_{Next} is updated to 0x0000_0004 to point to the next instruction address (i.e., instruction 2 in the figure). The second record is a hit branch with (`index` $= 1$, `offset` $= 1$), causing an ***'invalid'*** B address to be looked up from the cache_address array in $step_{10}$, since the cache line in $index_1$ has not been filled with any valid address yet. However, at this point, A_{Next} is not equal to A_B in $step_{13}$ and will go into $step_{14}$ through $step_{18}$. $Step_{14}$ checks that the current A_{Next} exists in the cache_address array and the array is not updated. $Step_{18}$ updates A_{Next} to 0x0000_0008 (i.e., instruction 3 in the figure) and then goes back to $step_{13}$. The inner loop $step_{14}$ through $step_{18}$ operates iteratively and handles A_{Next} 0x0000_0008 and 0x0000_0000C in a similar way. When A_{Next} points to 0x0000_00010, $step_{14}$ checks that the current A_{Next} does not exist in the cache_address array and the cache line in $index_1$ is filled in $step_{15}$. A valid B address is regenerated in $step_{17}$ and finds that the corresponding B address is 0x0000_0014. Then, $step_{18}$ updates A_{Next} to 0x0000_0014 and goes back to $step_{13}$. At this point, A_{Next} is equal to A_B and the exit path of $step_{13}$ is taken. Finally, $step_{19}$ outputs the B address and completes the processing of the current branch record.

6 Experimental Results

6.1 Hardware Implementation

The TC-cache has been implemented in RTL and integrated with an academic ARM-like CPU [24]. The cache components are modified from the caches of LEON2 processor [25]. Both the instruction and data caches have the same configuration: 4 KB direct map cache with line size of four words (16 bytes). The design is synthesized with UMC 0.18-μm standard cell library.

The analysis of the hardware gate counts is listed in Table 2. The TCS circuit costs only 3,652 gates, which is only 1.1 percent of the CPU+Cache system integration. Furthermore, the TCS circuit does not impair the system performance. The circuit delays are listed in Table 3. The TCS circuit causes the delay of the instruction cache to increase slightly from 6.0 ns to 6.3 ns. However, such minor increase does *not* impair the system since it is still less than the global critical path (7.3 ns), which is in the CPU. The CAD tool did not bother to optimize the paths related to the cache. If it is necessary to further improve the speed of the instruction cache, we can add a pipeline register between the cache and the trace generator to decouple the delay of the trace generator from the delay of the cache.

6.2 Compression Quality

For each benchmark, the *compression ratio* of its program trace is defined as follows:

$$\left(1 - \frac{Compressed\ Trace\ File\ Size}{Full\ Trace\ File\ Size}\right) \times 100\% \tag{1}$$

where the *compressed trace file size* can be obtained from

$$\#\, of\ Hit\ \times\ (1 +\ Hit\ Size)\ +\ \#\, of\ Miss\ \times\ (1 +\ 32) \tag{2}$$

and the *hit size* is composed by

$$Bidwidth\ of\ \{Index\ +\ Offset\ +\ Set\}.$$

Table 4 analyzes how the hit size is affected by the cache size and cache line size for the direct map cache. The analysis shows that the hit size depends solely on the cache size regardless of its cache line size. Therefore, the hit size is 8/9/10/11/12 bits for 1/2/4/8/16 KB cache, respectively. For set associative caches, the hit size also depends solely on the cache size, regardless of the cache line size and set number.

The compression ratios of five application benchmarks, with each benchmark running for 1,000,000 instructions, are measured for a direct map cache with various cache sizes in Tables 5 and 6 for the bypass and online modes, respectively. The result shows that a small cache, ranging from 1 to 4 KB, is good enough to produce the best

TABLE 5
Compression Ratio versus Cache Size (the Bypass Mode)

Application	Compression Ratio				
	Cache Size (Bytes)				
	1K	2K	4K	8K	16K
DCT	90.8%	90.4%	90.1%	89.4%	88.7%
FFT	89.8%	91.4%	92.2%	91.8%	91.9%
JPEG Encoder	91.1%	90.4%	91.4%	90.6%	89.9%
Fibonacci Sequence	83.4%	81.6%	79.7%	77.9%	76.1%
Tower of Hanoi	92.8%	92.0%	91.2%	90.4%	89.6%

* Instr. cache configuration is a direct map cache with line size of 4 words (16 bytes)

TABLE 6
Compression Ratio versus Cache Size (the Online Mode)

Application	Compression Ratio				
	Cache Size (Bytes)				
	1K	2K	4K	8K	16K
DCT	90.9%	90.8%	90.4%	89.6%	88.9%
FFT	90.1%	91.2%	92.2%	91.9%	91.8%
JPEG Encoder	89.3%	88.7%	91.4%	90.7%	89.9%
Fibonacci Sequence	83.4%	81.6%	79.7%	77.9%	76.1%
Tower of Hanoi	92.8%	92.0%	91.2%	90.4%	89.6%

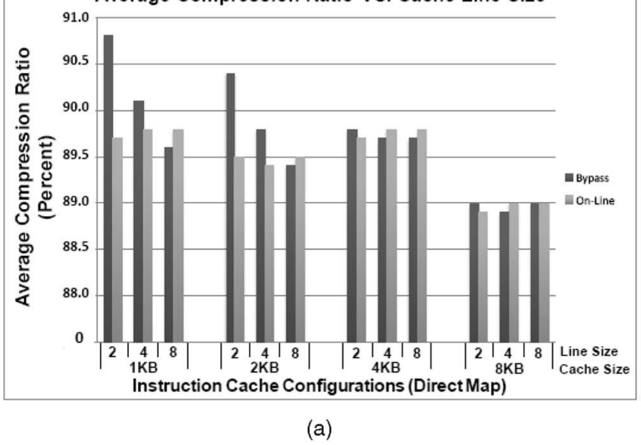
* Instr. cache configuration is a direct map cache with line size of 4 words (16 bytes)

compression ratio. A larger cache does not result in better compression ratio. The reason is that as the cache size increases, the hit size increases more significantly than the cache miss rate decreases. This result indicates that our proposed TC-cache is a very cost effective solution; a good quality of compression ratio can be easily achieved with a small cache.

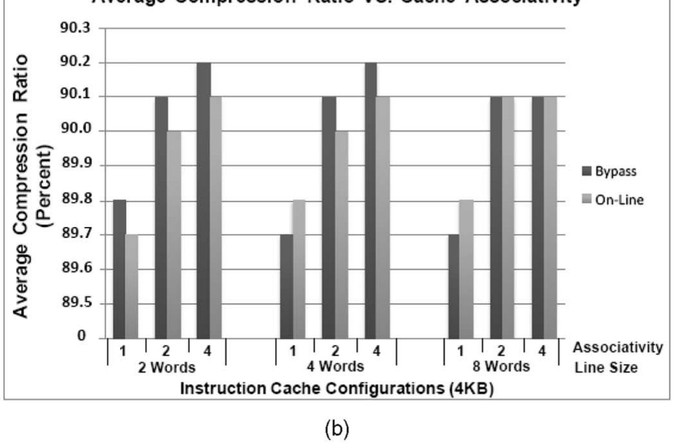
Fig. 15, left, shows the compression ratios of the bypass mode and the online mode when varying the cache line size ranging from two words to eight words. The result shows that the compression ratio of the bypass mode increases as the cache line size decreases, especially for a small cache. The reason is that the number of cache lines is increased as the line size decreases for a fixed-size cache. Further, the instruction cache is used solely for storing the B/T instructions in the bypass mode, and hence the cache hit rate can be increased with a greater number of cache lines, since more B/T instructions can be recorded in the cache. On the contrary, the compression ratio of the online mode tends to increase as the line size increases, since a larger line size has higher probability to bring in the B/T instructions in advance by non-B/T instructions, thus increasing the hit rate of the B/T instructions. In addition, a larger line size enables the spatial locality of the normal program execution to be utilized, this is beneficial to increase the probability of cache hits for the B/T instructions as well.

Fig. 15a also compares the compression ratio between two modes for different cache line sizes and cache sizes. The result shows that the bypass mode outperforms the online mode when the line size is small. The online mode is slightly worse than the bypass mode, due to a thrashing-like situation in which the B/T instructions compete with other non-B/T instructions for the same cache locations. Such effect is significant especially for a small cache. In contrast, the bypass mode can avoid such competition since only the B/T instructions are eligible to be stored in the cache. However, the online mode can utilize the property of hit on first access (caused by bringing in B/T in advance) when the line size increases, which amortizes such competition problem and increases the compression quality. Therefore, the online mode is better than the bypass mode for a larger line size. It should be noted that the bypass mode significantly benefits from a smallest line size, it is capable to achieve an impressive compression ratio with only a small amount of cache. For example, a compression ratio of 90.8 percent is achieved for a 1 KB direct map cache with line size of two words.

Fig. 15b shows the compression ratios for a set associative cache with different associativity ($N_{WAYS} = 1$, 2, 4) in both the bypass mode and the online mode. The result shows that increasing the cache associativity improves the compression ratio for both modes. In this case, the cache hit rate increases with the cache associativity for



(a)



(b)

Fig. 15. Average compression ratio as functions of cache line size and cache associativity.

TABLE 7
Comparison with Related Work

	Trace Capability		Average Program Trace Compression Ratio	Overhead Gate Count
	Program Trace	Data Trace		
Our Work	√	√ (address only)	89.8%*	3652 or 7304^{Δ} (UMC 0.18-μm)
Kao *et al.* [12]	√		99.7%	51678 (UMC 0.18-μm)
Hsieh *et al.* [6]	√	√	88.3%	28662 (0.13-μm)
ARM7 ETM [5]	√	√	80.0% [26]	30720 (n/a)
Generic Nexus [7]	√	√	88.5%	$60120^{\dagger}$ (0.18-μm)
Hopkins *et al.* [7]	√	√	97.0%	$30181^{\dagger}$ (UMC 0.18-μm)
PDLZW + AHDB [9]	√	√	37.8% (exe. file)	133120 (0.35-μm)
X-MatchPROv4 [10]	√	√	49.0%	9039 Tile's 70% of a A500K130-BG45 FPGA

* = 1 - geometric mean of trace reduction in 4 KB cache in Table 6
Δ = 3652 gates when the TCS circuit is shared; 7304 gates when the TCS circuit is not shared.
† = Original data are published as the chip area. We translate the area information into the equivalent NAND2 gate counts (one gate = 9.98μm^2) [12] for comparison.

the decrease of the cache conflict. Consequently, increasing the cache associativity has the positive impact on compression ratio. However, increasing the associativity beyond two ways yields little or no benefit. In addition, the online mode significantly benefits from an increase in the associativity than in the line size. For example, the compression ratio for a 2-word cache line is improved from 89.7 to 90.0 percent when increasing the associativity from one way to two ways, versus from 89.7 to 89.8 percent when increasing the line size of a direct map cache from two words to eight words.

6.3 Power Analysis

Table 8 lists the power consumption of major hardware components (CPU, RAM blocks in caches, and the TCS circuit) under different operation modes. The power consumption is estimated with Synopsys's PrimePower [27] using the JPEG encoder application. The technology library used is UMC 0.18-μm standard cell library, with the operating voltage of 1.62 V and frequency of 100 MHz. The CPU is on in all configurations which consumes 29.8 mW power. The single-port synchronous SRAM used in our experiment supports standby/read/write modes. The energy of the RAM blocks is saved by entering the standby state when they are turned off. This can be done by controlling the chip enable signal of the RAM block in Fig. 4. Only the leakage power is dissipated under the standby mode. On the other hand, the average read/write current is 12.006 mA under the assumption of 50 percent read/write operations [28] for a 64-word by 30-bit memory module. In the table, the cache consumes 7.3 mW in the standby (disabled) mode and 304.8 mW in the active (enabled) mode. For the TCS circuit, the clock is gated when it is not used. The table shows that it consumes 0.1 mW in the gated clock mode and 0.7 mW in the active mode. The cache is the dominant power contributor when it is enabled.

The CPU-$ is the basic configuration, consuming 37.2 mW, in which both the cache and the tracing support are disabled, with only the CPU as the active component. When the cache is enabled to speed up memory operations, it becomes the CPU+$ configuration, consuming 334.7 mW due to the entire cache being on.

The CPU+$ configuration is very typical in modern SoC applications. When the tracing support is enabled for CPU+$, it becomes the CPU+TC$_OL (online mode) configuration, consuming 335.3 mW, an increase of merely 0.6 mw (0.2 percent) because the entire cache has already been on in the CPU+$ configuration. This indicates that the online tracing is very power efficient since it utilizes most of the cache components and their existing operations in CPU+$.

When the tracing support is enabled for CPU-$, it becomes the CPU+TC$_BP (bypass mode) configuration, consuming 110.1 mW, which is an increase of 72.9 mW. Such increase is larger than the increase in the case of the online mode because the cache is entirely disabled in the CPU-$ configuration. However, such an increase is still much less than the increase (297.5 mW) of going from the CPU-$ configuration to the CPU+$ configuration, because in CPU+$ the entire cache is enabled, whereas in CPU+TC$_BP only the instruction tag RAM needs to be turned on, which consumes just 26 percent of the entire cache.

TABLE 8
Power Consumption of Each Operation Mode (JPEG Encoder)

Hardware Components		CPU + Cache + Tracer Configurations			
		CPU-$	CPU+$	CPU+TC$_BP	CPU+TC$_OL
Power (mW)	CPU	29.810	29.810	29.810	29.810
	(Status)	*On*	*On*	*On*	*On*
	Cache	7.295	304.809	79.594	304.809
	(Status)	*Off*	*Entirely On*	*Only Tag RAM of I$ is On*	*Entirely On*
	TCS Circuit	0.083	0.063	0.671	0.658
	(Status)	*Off*	*Off*	*On*	*On*
	Total	37.188	334.682	110.075	335.277

1. Instr. and data cache configuration are both 2 KB 2-way set associative cache with line size of 8 words (32 bytes)
2. CPU + TC$_BP : Instr. cache with bypass tracing mode
3. CPU + TC$_OL : Instr. cache with on-line tracing mode

6.4 Comparison with Related Work

Table 7 compares our work with related work. Most of the work, including ours, is capable of capturing both program and data traces, except Kao et al. Note that our work does not support the data value when capturing the data trace (details being discussed in Section 4.4). Since the program trace is our major goal, we will focus on the program trace subsequently.

For the average program trace compression ratio, our work (89.8 percent) is better than the processor-related solutions (ARM7 ETM, Hsieh et al., and generic Nexus module, with ratios from 80.0 to 88.5 percent) and is far better than the general purpose data compression solutions (PDLZW+AHDB and X-MatchPROv4, with ratios 37.8 percent and 49.0 percent, respectively). The approaches of Kao et al. (99.7 percent) and Hopkins et al. (97 percent) achieve a higher average compression ratio than our work at the cost of much more complex hardware schemes. On the other hand, ours hardware overhead (3,652 gates if the TCS circuit is shared; 7,304 gates if the TCS circuit is duplicated) is far lower than those of all related approaches (from 28,662 to 1,33,120 gates). In summary, our work is very cost effective; it achieves a very good compression ratio with a minimal hardware cost, making it a highly feasible debugging/monitoring solution to SoCs, even for cost-sensitive ones such as consumer electronics.

7 Conclusion

We have presented a novel TC-cache which could function simultaneously as a regular instruction cache and as a real-time program trace compressor with minor modification to the original processor-cache integration circuit. The proposed cache is very flexible by being capable of working in two operation modes: the online mode where the cache is used for both tracing and memory performance optimization and the bypass mode where the cache is used solely for tracing but not memory performance optimization. Synthesis results have shown that the modification does not impair the system performance, and the hardware overhead is just 3,652 gates while achieving a significant average compression ratio of 90 percent. In addition, the TC-cache is very power efficient: in the online mode it requires only 0.2 percent of additional system power, and in the bypass mode it requires only 26 percent of the cache power. Compared with related work, our work has the lowest hardware overhead and achieves an impressive compression quality. Experiments also show that only a small amount of cache, such as 1 KB, is sufficient to achieve high-compression ratio. Therefore, our approach is a highly feasible debugging/monitoring solution to SoCs, even for cost-sensitive ones such as consumer electronics. Furthermore, the same concept can be applied to the data cache to compress the data address trace as well.

To the best of our knowledge, our approach is the first work to incorporate the tracing capability into the cache. In the future, we would like to investigate the possibility of partitioning the cache into smaller blocks and organizing them in a more intelligent way to make the cache more versatile in ever diversifying developments and applications of modern SoCs.

References

[1] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Information Theory*, vol. IT-23, no. 3, pp. 337-343, May 1977.

[2] *IEEE-ISTO Nexus 5001 Forum*, http://www.nexus5001.org/, 2010.

[3] *Xtensa Processor Real-Time Trace*, Tensilica Inc., http://www.tensilica.com/products/xtensa/xtensalx/traceLX.htm, 2010.

[4] *MPC565 Reference Manual, Chapter 22, Development Support*, Freescale Semiconductor Inc., Nov. 2005.

[5] *Embedded Trace Macrocell Architecture*, ARM Ltd., http://www.arm.com/products/solutions/ETM.html, 2010.

[6] M.-C. Hsieh and C.-T. Huang, "An Embedded Infrastructure of Debug and Trace Interface for the DSP Platform," *Proc. IEEE Design Automation Conf.*, pp. 866-871, June 2008.

[7] A. Hopkins and K. McDonald-Maier, "Debug Support Strategy for Systems-on-Chips with Multiple Processor Cores," *IEEE Trans. Computers*, vol. 55, no. 2, pp. 174-184, Feb. 2006.

[8] J.-M. Chen and C.-H. Wei, "VLSI Design for High-Speed LZ-Based Data Compression," *IEE Proc. Circuits, Devices, and Systems*, vol. 146, no. 5, pp. 268-278, Oct. 1999.

[9] M.-B. Lin, J.-F. Lee, and G.E. Jan, "A Lossless Data Compression and Decompression Algorithm and Its Hardware Architecture," *IEEE Trans. VLSI Systems*, vol. 14, no. 9, pp. 925-936, Sept. 2006.

[10] J. Nunez and S. Jones, "Gbit/s Lossless Data Compression Hardware," *IEEE Trans. VLSI Systems*, vol. 11, no. 3, pp. 499-510, June 2003.

[11] S. Kasera and N. Jain, "A Survey of Lossless Data Compression Techniques," technical report, 2004.

[12] C.-F. Kao, S.-M. Huang, and I.-J. Huang, "A Hardware Approach to Real-Time Program Trace Compression for Embedded Processors," *IEEE Trans. Circuits and Systems I*, vol. 54, no. 3, pp. 530-543, Mar. 2007.

[13] J. Montanaro, R. Witek, and K. Anne, "A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor," *IEEE J. Solid-State Circuits*, vol. 31, no. 11 pp. 1703-1714, Nov. 1996.

[14] *S3C4510B Data Sheet*, Samsung Electronic.

[15] *ADSP-BF537 Data Sheet*, Analog Devices Inc., Feb. 2009.

[16] *TMS320C6711D Floating-Point Digital Signal Processor (Rev. B)*, Texas Instruments Inc., June 2006.

[17] P.P. Ranjan, N.D. Dutt, and A. Nicolau, *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration*. Kluwer Academic Publishers, 1999.

[18] D. Chiou, P. Jain, L. Rudolph, and S. Devadas, "Application-Specific Memory Management for Embedded Systems Using Software-Controlled Caches," *Proc. IEEE Design Automation Conf.*, pp. 416-419, June 2000.

[19] D. Hillenbrand and J. Henkel, "Block Cache for Embedded Systems," *Proc. IEEE Design Automation Conf. Asia and South Pacific*, pp. 322-327, Mar. 2008.

[20] N. Baron, P. Marino, A. Goren, and E. Melanmed-Cohen, *Real Time Cache Implemented by On-Chip Memory Having Standard and Cache Operating Modes*, US Patent, 1996.

[21] D.L. Cullison and T.A. Wagner, *Multi-Purpose Cache Memory Selectively Addressable Either as a Boot Memory or as a Cache Memory*, US Patent, 1992.

[22] S. Basumallick and K. Nilsen, "Cache Issues in Real-Time Systems," *Proc. ACM SIGPLAN Workshop Language, Compiler and Tool Support for Real-Time Systems*, May 1994.

[23] *Embedded Trace Macrocell ETMv1.0 to ETMv3.4 Architecture Specification, Chapter 4.6, Data Trace*, ARM Ltd., July 2007.

[24] Y.-T. Lin and I.-J. Huang, "Enhanced 32-bit Microprocessor-Based SoC for Energy Efficient MP3 Decoding in Portable Devices," *Proc. Int'l Conf. Consumer Electronics*, pp. 1-2, Jan. 2007.

[25] *LEON2 Processor User's Manual*, Gaisler Research, May 2004.

[26] C. MacNamee and D. Heffernan, "Emerging On-Chip Debugging Techniques for Real-Time Embedded Systems," *Computing and Control Eng. J.*, vol. 11, pp. 295-303, Dec. 2000.

[27] *PrimePower Manual*, Synopsys Inc., June 2005.

[28] *UMC 0.18 μm Process High-Speed Single-Port SRAM (HS-SRAM-SP) Generator User Manual*, Artisan Components Inc., Aug. 2000.

Chun-Hung Lai received the BS from the Department of Information Engineering and Computer Science, Feng-Chia University, Taiwan, in 2004, and the MS degree from the Department of Computer Science and Engineering, National Sun Yat-Sen University, Taiwan, in 2007. He is currently working toward the PhD degree at the Department of Computer Science and Engineering, National Sun Yat-Sen University. His research interests include memory issues in embedded system design, embedded microprocessors, and hardware/software codesign. He is a student member of the IEEE.

Fu-Ching Yang received the BS, MS, and PhD degrees from the Department of Computer Science and Engineering, National Sun Yat-Sen University, Taiwan, in 2003, 2004, and 2009, respectively. His research interests include SoC debugging methodology, trace compression algorithm development, and microprocessor design/verification. He ia a student member of the IEEE.

Ing-Jer Huang received the BS degree in electrical engineering from National Taiwan University in 1986, and the MS and PhD degrees in computer engineering from the University of Southern California in 1989 and 1994, respectively. He is currently a professor at the Department of Computer Science and Engineering, National Sun Yat-Sen University, Taiwan. His research interests include microprocessors, SoC design, design automation, system software, embedded systems, and hardware/software codesign. He has been actively involved in academic, educational, and industrial activities. He has extensive collaboration with several IC-related industries. He is a member of the IEEE and the ACM.