On-Line Software-Based Self-Test for Data TLBs

G. Theodorou, S.Chatzopoulos, N. Kranitis, A. Paschalis, D. Gizopoulos

Department of Informatics & Telecommunications, University of Athens, Greece

{gthe,nkran,paschali,dgizop}@di.uoa.gr

Abstract

In this paper, an SBST program development methodology is proposed for on-line testing of data TLB (D-TLB), both for data (SRAM) and tag (CAM) memory arrays. The proposed SBST methodology exploits existing special purpose instructions that modern Instruction Set Architectures (ISAs) implement to access the TLBs for debug-diagnostic purposes, termed hereafter Direct TLB Access (DTA) instructions, as well as, the trap handler mechanism to overcome testability challenges. Experimental results on the D-TLB arrays of OpenSPARC T1 show a significant improvement of test time (on average 40%) when such debug-diagnostic instructions are exploited.

1. Introduction

Most modern processors employ a small on-chip fully associative cache memory, commonly known as TLB to speed up the virtual memory translation. TLB arrays are small memory arrays that usually lack a MBIST scheme (e.g UltraSPARC T1 TLBs), since TLBs are limited size arrays (up to 128 entries).

Software-Based Self-Test (SBST) has recently emerged as a complementary solution for processor manufacturing [1] and periodic on-line testing [2]. In the case of on-line cache testing, SBST has increased flexibility to apply March tests [3]. Hence, SBST can be a viable solution for applying March tests to TLB arrays.

In this paper, we introduce an SBST program development methodology to apply March tests for on-line testing of both data (SRAM) and tag (CAM) D-TLB arrays. The proposed methodology can apply any March test to detect storage (both data and tag arrays) and comparison faults [4] (for tag array only). The methodology leverages the inherent power of moderns ISAs by exploiting special purpose instructions that we denote as Direct TLB Access (DTA) instructions and exploits the trap handler mechanism that is available in modern architectures, as well. The effectiveness of the proposed methodology is demonstrated on the D-TLB arrays of a modern processor, OpenSPARC T1.

2. DTA instructions in modern ISAs

DTA instructions are special instructions for debug-diagnostic purposes that provide direct controllability and observability to TLB arrays. These instructions are suitable for SBST implementation of March tests that target D-TLBs. Based on the TLB structure, an ideal DTA instruction has to contain the following fields:

- TLB Line Selection (LS) field
- Write/Read/Compare operation (WRC) selection field
- Array Selection (AS) field
- From/To data Address (A) field

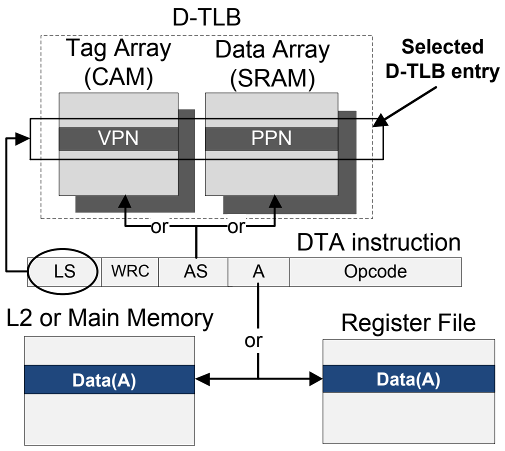

Figure 1: Ideal DTA instruction

In Figure 1 the format of an ideal DTA instruction and the way that every field is utilized to access a typical D-TLB is presented. In detail, the data or tag array of this D-TLB can be accessed by controlling the *AS* field. The March operation can be selected with the *WRC* field to write, read or compare a selected D-TLB entry. Compare operation is valid only for tag array. In case of D-TLBs that are fully associative arrays, a line selection field -*LS* field- is needed in the ideal DTA instruction to select a TLB entry in order to gain direct access to every D-TLB entry. Finally, *A* field contains an address of a general purpose register where the March test read outputs should be stored.

An ideal DTA instruction provides direct access to the D-TLB arrays and can overcome the testability challenges of applying March tests to D-TLBs arrays. In practice such an ideal DTA instruction is not present in ISAs but it can be indirectly implemented by combining a set of existing DTA instructions that cover in total all fields of the ideal one. Representative examples of such special purpose instructions, which can be characterized as DTA instructions, are present in RISC architectures, such as MIPS, ARM and SPARC architectures.

MIPS architectures implement special instructions for debug-diagnostic purposes that can directly access TLB data and tag arrays (*TLBWI*, *TLBR* and *TLBP* instructions). These instructions are part of the special coprocessor CP0 instructions that are implemented in modern MIPS architectures (e.g. MIPS R10000) and can write, read, and compare D-TLB contents with the content of either *ReadHi* or *ReadLo* registers respectively without executing store/load instructions. Hence, *TLBWI*, *TLBR* and *TLBP* instructions have similar fields with the ideal DTA instruction and can be considered as DTA instruction that can effectively implement March operations.

ARM architecture implements system control coprocessor (CP15) debug operations (MRC and MCR instructions) for accessing the D-TLB arrays (by utilizing register 15) for directly write, read and compare the D-TLB content. These instructions are executed in secure privilege mode and provide great visibility into the D-TLB by interrupting the program flow to execute them. They transfer the array contents from/to system array debug data registers without executing store/load instructions. Therefore, these instructions can be also considered as DTAs in order to implement March operations.

Finally, SPARC architecture implements alternate space identifier (ASI) store/load instructions (e.g. sta/lda instructions in LEON3 and stxa/ldxa instructions in OpenSPARC T1). These instructions are utilized to access embedded RAMs through a SPARC diagnostic access bus that bridges these embedded RAMs with the main memory or the processor's register file. Every embedded RAM, including D-TLB arrays, can be accessed by using ASI store/load instructions in order to implement write and read operations. Especially, for the D-TLB arrays, dedicated ASIs for both data and tag arrays are defined to access D-TLBs in hypervisor level. This way, the ASI fields of the ASI store/load instructions map to the *AS* field of the ideal instruction. Also, the ASI store/load instructions contain fields that map to *LS* and *A* fields of the ideal instruction, respectively. Hence, such ASI store/load instructions can be considered as DTA instruction and can be effectively used to implement March write and read operations.

3. D-TLB SBST methodology

The proposed SBST methodology implements low cost SBST March tests that target D-TLB arrays by taking advantage of existing debug-diagnostic instructions in modern ISAs to implement March write/read and compare operations (for CAM tag array). These instructions must cover in total the fields of the ideal DTA instruction to overcome D-TLB arrays testability challenges. If the ISA lacks debug-diagnostic instructions, generic store/load instructions can be utilized to cause a well-advised TLB miss of a memory page in order to implement March write/read operations by exploiting TLB miss & refill mechanism. Finally the methodology the trap handler mechanism to implement the March compare operation (for tag array) when the ISA lacks debug-diagnostic operations. The proposed SBST methodology is summarized in Figure 2.

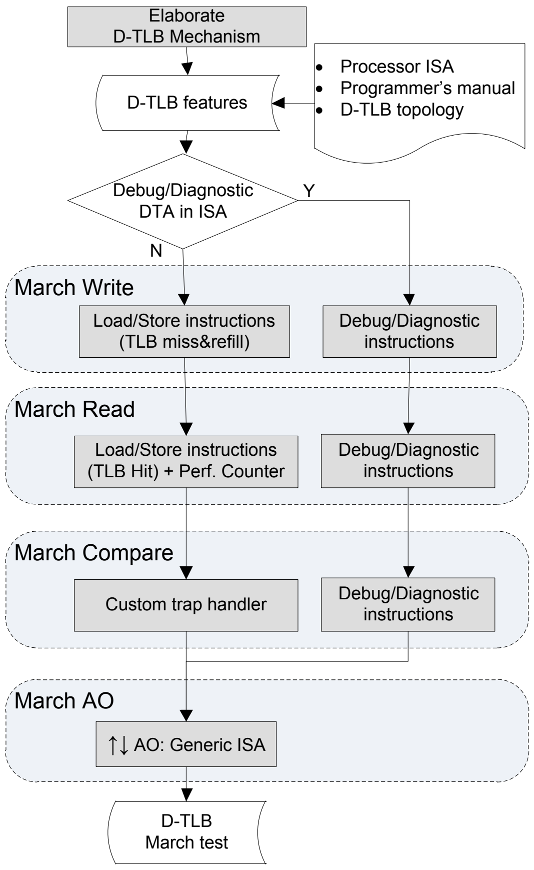
Elaborate D-TLB Mechanism
D-TLB features
Processor ISA
Programmer's manual
D-TLB topology
Debug/Diagnostic DTA in ISA
Y
N
March Write
Load/Store instructions (TLB miss&refill)
Debug/Diagnostic instructions
March Read
Load/Store instructions (TLB Hit) + Perf. Counter
Debug/Diagnostic instructions
March Compare
Custom trap handler
Debug/Diagnostic instructions
March AO
↑↓ AO: Generic ISA
D-TLB March test

Figure 2: SBST methodology for D-TLB arrays

4. Case study: OpenSPARC T1

The proposed SBST methodology has been applied to the D-TLB arrays of OpenSPARC T1 processor. OpenSPARC T1 implements a SPARC V9 ISA and includes privileged store/load instructions, denoted as alternate load/store (*ldxa/stxa* instructions). These instructions are considered as DTA instructions and can directly access the D-TLB arrays for debug/diagnostic purposes by specifying alternate space identifiers (ASIs) for both write and read access at supervisor level. We have exploited these alternate load/store instructions for March write/read operations for write and read March operations at low cost. SPARC V9 ISA does not implement a debug/diagnostic compare instruction for implementing the March compare operation. Therefore, a custom "miss-no-refill" D-TLB trap handler has been utilized to implement March compare operations to test CAM tag array for comparison faults. An assembly code snippet that shows how March operations are implemented for D-TLB arrays is shown in Figure 3.

Write/Read March operations - D-TLB Data array

```
!! %g4 contains a VA to access D-TLB data entries
wr %g0, ASI_DMMU_DATA_ACCESS, %asi
stxa %g6, [%g4] %asi        //March Write, %g6 contains DB
ldxa [%g4] %asi, %g7        //March Read, Data entry in %g7
```

Write/Read/Compare March operations - D-TLB Tag array

```
!! %g4 contains a VA to access D-TLB tag entries
wr %g0, ASI_DMMU_TAG_ACCESS %asi
stxa %g2, [%g1] %asi              //DB to Tag buffer from %g2
stxa %g2, [%g1+0x50] %asi         //Partition ID to PartID buffer
wr %g0, ASI_DMMU_DATA_ACCESS %asi
stxa %g6, [%g4] %asi              //March Write, %g6 any content
```

Figure 3: Code snippet for OpenSPARC T1 D-TLB arrays

In detail, alternate store (*stxa*) instructions have been used to implement March writes and alternate load (*ldxa*) instructions have been used to implement March reads. These instructions have direct access to the D-TLB arrays on PPN, VPN and control bits in every D-TLB entry by utilizing the appropriate address indexing and the corresponding ASI. Hence, the testability challenge of composing any data background and accessing control bits is addressed for storage faults. Note that March write operation for tag array is implemented by utilizing three *stxa* instructions. Tag entry (VPN, context ID and control bits) and Partition ID must firstly be placed in the Tag buffer and PartID buffer, respectively. Afterwards, March write operation for tag array is applied. The trap handler that we have programmed to implement March compare operation differs to the native one that handles D-TLB misses in two aspects: a. It does not refill the D-TLB when a miss occurs, b. It can access physical addresses beyond the 8GB limit (which is an OpenSPARC T1 convention for memory address space). This handler is enabled by utilizing a predefined value in register *%l6* and when a generic store/load instruction occurs it detects D-TLB hits and misses without refilling any D-TLB entry. Real addresses were utilized to avoid generating faults, for non-permitted physical addresses.

We have implemented a set of contemporary March tests for both storage (March C-, March MSS and March SS for both data and tag array) and comparison faults (March CFT [5] for tag array only). In order to demonstrate the effectiveness of the proposed methodology, we have implemented the March tests both by exploiting debug-diagnostic instructions as described in the proposed methodology and by utilizing the miss & refill

TLB mechanism (for architectures that lack debug-diagnostic instructions). It is clear that by exploiting debug-diagnostic instructions the test time is significantly improved for all the applied March tests by up to almost 40% (on average). The statistics for the D-TLB arrays (both for data and tag array) are shown in Table 1.

Table 1 : OpenSPARC T1 D-TLB: SBST routines statistics

March test	***TLB Miss & Refill Mechanism***		***Debug-Diagnostic Instructions (proposed)***		***Test Time Improv.***
	Test size (bytes)	***Test Time (cycles)***	***Test size (bytes)***	***Test Time (cycles)***	
March C-	480	43,908	424	26,412	39,8%
March MSS	648	56,484	520	33,972	39,9%
March SS	366	78,208	684	49,240	37,0%
March CFT [5]	600	43,160	556	22,764	47,3%

Finally, we evaluated the test effectiveness of the SBST routines with an in-house developed extended version of RAMSES fault simulator. The achieved fault coverage is complete (100%) for all the storage faults[1] that every March test guarantees for both data and tag D-TLB arrays when DTA instructions are exploited. It should be noted that when miss & refill mechanism is utilized, the fault coverage is lowered to 92% for the data D-TLB array and to 90% for the tag D-TLB array due to limitations on accessing the control bits for data backgrounds composition. For CAM comparison faults[2], the fault coverage is also slightly lowered to 92% for the D-TLB tag array due to the lack of a debug-diagnostic compare instruction and the utilization of the custom trap handler that addresses similar limitations in accessing the control bits.

References

[1] N. Kranitis, A. Paschalis, D. Gizopoulos, G. Xenoulis, "Software-Based Self-Testing of Embedded Processors", *IEEE Transactions on Computers*, Vol. 54, no. 4, pp. 461-475, 2005.

[2] A. Paschalis, D. Gizopoulos, "Effective software-based self-test strategies for on-line periodic testing of embedded processors", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, no.1, pp.88 – 99, Jan. 2005.

[3] G. Theodorou, N. Kranitis, A. Paschalis, D. Gizopoulos, "A Software-Based Self-Test Methodology for On-Line Testing of Processor Caches", in Proc. of *Int'l Test Conference (ITC)*, 2011

[4] K.J. Lin, C.W. Wu, "Testing content-addressable memories using functional fault models and march-like algorithms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.19, no.5, pp.577-588, May 2000

[5] P. Manikandan, B. B. Larsen, E. J. Aas, S. M. Reddy, "Test of Embedded Content Addressable Memories", in Proc. of *IEEE Int'l Symposium on Electronic System Design (ISED)*, 2010, pp.113-118.

[1]**Single Cell Storage Faults**: State (SF), Transition (TF), Write Destructive (WDF), Read Destructive (RDF), Deceptive Read Destructive (DRDF), Incorrect Read (IRF)
Coupling Storage Faults (2-cell): State (CFst), Disturb (CFds), Transition (CFtr), Write Destructive (CFwd), Read Destructive (CFrd), Deceptive Read Destructive (CFdr), Incorrect Read (CFir)

[2] **Compare Faults**: Stuck-Match (SMF), Stuck-MisMatch (SMMF), Conditional-Match (CMF), Partially-Match (PMF), Equivalence-MisMatch (EMMF), Inequivalence-Match (IMF), Cross-Match (XMF), Cross-Mismatch (XMMF)