# Processor Shield for L1 Data Cache Software-Based On-line Self-testing

Ching-Wen Lin

Institute of Computer and
Communication Engineering,
National Cheng Kung University,
Tainan, Taiwan, R.O.C.
kliolin@mail.ee.ncku.edu.tw

Chung-Ho Chen

Institute of Computer and
Communication Engineering,
National Cheng Kung University,
Tainan, Taiwan, R.O.C.
chchen@mail.ncku.edu.tw

**Abstract - Conventional software-based cache self-tests typically ignore system related testing issues, such as physical memory layout, virtual memory mapping, and isolating faulty effects, especially for on-line testing. We propose an architectural support for data cache software-based self-testing (SBST): Processor Shield, which can tackle difficult-to-test issues during on-line SBST. The proposed processor shield includes a software framework and design for testing (DFT) hardware, which enables SBST program to run without influencing other processes and on-bus devices even if a cache test fails. The proposed SBST process can be iteratively executed and cooperate with dynamic voltage frequency scaling (DVFS) system to calibrate the required guardbands to accommodate transistor aging effects. Finally, we present a case study that performs SBST programs under Linux kernel on an ARMv5-compatible processor system. Our method can successfully switch between the SBST process and the kernel process and achieve the expected high fault coverages for cache control logic and RAM module testing.**

## I. Introduction

On-line testing is an effective method to capture operational faults, detect the transistor aging, and improve the system reliability. The cache system plays an important role in a modern processor by bridging the speed gap between the processor and the main memory, and it typically dominates the power consumption of a processor. In a low-power system design, dynamic voltage frequency scaling (DVFS) system is the most common method to balance the performance and power consumption.

Transistor aging may increase the threshold voltage and then degrade the max working frequency (e.g., the NBTI aging). A conventional solution is to use guardbanding, which reduces the working frequency or increases the operating voltage to tolerate the degradation during the chip lifetime [1]. The higher guardbands imply the lower performance or the higher power consumption. On-line self-test not only can capture the operational faults, but it is also able to provide the feedback to the DVFS system to calibrate the required guardbands against the transistor aging.

Software-based self-testing (SBST) is the most common on-line testing method since it can perform at-speed testing and critical path testing with no design modification and no additional power consumption. This paper presents a methodology to perform an on-line D-cache SBST under the operating system by preserving the contexts of other processes and on-bus devices unaffected even if a test fails.

To test memory cells, March algorithms are widely used for memory cell built-in self-testing (BIST) and SBST [3], [4], [5], [7], [8]. Various March algorithms are implemented to perform a cache SBST in a non-OS platform. The authors propose many methods to resolve encountered challenges during executing the March sequences on their target cache. However, the system related issues of on-line testing are typically ignored since the manufacturing test is the main purpose of previous works.

The first system related issue when conducting an on-line test is the system memory mapping, including physical memory layout and the virtual memory. The system memory mapping places limits on accessing the memory for on-line testing. The physical memory layout decides how a physical address is mapped to the main memory, the memory-mapped I/O (MMIO), or the reserved space, and this mapping typically depends on the hardware or platform configuration. The virtual memory is managed by the operating system, and the test process has to request the required memory regions from the operating system.

The above system memory mappings place limits of memory addresses during on-line testing, and a test process may have no right to access the required addresses. Accessing the required addresses in sequence is an essential operation for testing; however, these limits may make these accesses illegal. We call these addresses, which are required for testing but limited by the system memory mapping, **shielded addresses**, and an on-line SBST has to redirect these shielded addresses to the available memory space acquired from the operating system; otherwise, the test may result in a system crash.

The second system related issue is the faulty effect isolation which is necessary to keep other processes and on-bus devices unaffected during an on-line cache SBST. For a data cache test, the major syndrome, which interferes with other processes or on-bus devices, is the faulty address, and it comes from testing physical tag RAM or the controller. For instance, when the cache cleans a dirty line, the corresponding content of physical tag RAM is used to build the full physical address to directly access the system bus. If there is a fault in the RAM cell, the faulty address may cause irreversible damages to other processes or result in a system crash. Therefore, how to prevent the processor or a bus master, for example, the cache controller, from accessing the bus using a faulty address is the another challenge of performing an on-line cache SBST.

This work is supported by the Ministry of Science and Technology, Taiwan, R.O.C., under Grant MOST 105-2218-E-006-024 -

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

In this paper, we address the system related issues during a data cache on-line SBST, and we propose an architectural approach called **processor shield**, including the software framework and the design for testing (DFT) hardware. The software framework constructs a non-preemptive privileged process that can provide the required environment for data cache SBST and return the control to the operating system after the test is done. The DFT hardware blocks the illegal accesses due to faulty addresses and on the other hand, redirects the shielded addresses to the available memory space prepared by the software framework.

We present a case study which runs the data cache SBST program under Linux kernel on an ARMv5-compatible processor system. The proposed method can seamlessly switch the SBST process and the kernel process, even if a test fails. Based on various environmental conditions, including the temperature, the system workload, the operating voltage, and the frequency, the expected fault coverage, depending on the controller test program and the March algorithm used, can be obtained, and the test results can be used as the feedback to the DVFS system.

The rest of this paper is organized as follows. Section II describes the background and the related work. Section III presents the proposed processor shield. Section IV shows the software-based March algorithm development. Section V provides a case study and the experimental result. Finally, we make a conclusion for this paper.

## II. Background and Related Work

A March algorithm is a sequence of March elements composed of pre-defined read/write operations to the test target memory cells by the special addressing order. In the word-oriented March algorithm [2], the data that are marched for read/write are called data backgrounds (DBs) or their complementary data backgrounds (CDBs). After a March element has been applied to one memory cell, the element has to then be applied to the next one according to the pre-defined addressing order which can be ascending (⇑), descending (⇓), or either (⇕). As shown in Fig. 1, the word-oriented March C- algorithm contains six March elements with the write/read order specified from left to right. The least number of the required DBs and CDBs to test the intra-word faults of an L-bit memory cells (word) is defined as follows:

$$\{2 * (\lceil \log_2 L \rceil + 1)\} \text{ --- Eq. (1)}$$

⇕(wDB); ⇑(rDB, wCDB); ⇑(rCDB, wDB); ⇓(rDB, wCDB); ⇓(rCDB, wDB); ⇕(rDB)

Fig. 1. Word-oriented March C- Algorithm

The caches must cope with the translation of a virtual address from the processor to a physical address to access the system bus. The caches can be classified into four types: virtual index virtual tag (VIVT), virtual index physical tag (VIPT), physical index physical tag (PIPT), and physical index virtual tag (PIVT). An L1 cache implementation can be constructed as a VIVT or VIPT architecture to reduce the cache access latency. For a data cache, the VIVT architecture usually builds a physical tag associated with each cache line so that a dirty cache line can be written back to the next level memory without querying the MMU. In this paper, we focus on the on-line SBST of the cache system that has both virtual tag RAM and the physical tag RAM. The proposed method can be easily extended to the other architecture that uses only either physical tag RAM or virtual tag RAM.

Testing cache using SBST has been proposed by lots of researches. Carlo et al. [3] have tried to overcome the difficulties of set-associative cache testing. They deal with the issues successfully without using the hardware or special instruction support; however, there is no detailed discussion on performing the SBST under the operating system. In the other work, Theondorou et al. have proposed the direct cache access instructions (DCA instructions) that can directly read/write the desired value from/to any location of data RAM, tag RAM, and physical tag RAM in the cache [4]. Unfortunately, not every ISA has these special cache control instructions that can fully support cache testing.

## III. Proposed Processor Shield Design

In order to deal with the system related issues, we propose an architecture mechanism called processor shield, which can prevent the system from entering an unrecoverable status during testing and then return the control to the operating system. The unrecoverable statuses include the system crash, the violated contexts of other processes, and the modified contests of the MMIOs. All these statuses are caused when the test process accesses the shielded addresses limited by the system memory mapping, and these accesses may come from the test process or the faulty effects.

The proposed processor shield design includes the software framework and the DFT hardware. Before the processor executes the cache SBST body to test the RAM cells and the controller, the software framework has to prepare the required resources and initialize the test environment. During the cache SBST, the processor shield DFT has to prevent other processes and MMIOs from being affected by the test procedure and the faulty effects. Then, the SBST process has to return the control to the operating system as if the test program were never executed. In this paper, the data cache is assumed to be implemented in monolithic SRAM modules, including a data RAM bank, a virtual tag RAM bank, and a physical tag RAM bank. If the target bank is divided into several blocks, the proposed method can be applied to each block separately.

### *A. Challenge in Virtual Memory System*

The virtual memory system places limitations on accessing the cache so the on-line cache SBST process has to overcome two challenges on the virtual memory platform. To achieve the high fault coverage of the cache controller test, the SBST process has to execute cache control instructions and perform many read/write operations under various cache policies, including the write hit/write miss policy. Since the cache policy is typically set in the page descriptor, this requires to test the massive pages which have different virtual addresses, physical addresses, and cache attributes. However, the accesses to these pages may violate the memory access rights or protection overseen by the operating system. This is the first challenge to conquer for data cache on-line testing.

The second challenge is the misalignment problem when starting the March test from a non-zero cache index. These accessing sequences cannot meet the March addressing order since it requires to start from a zero index. Fig. 2 shows the example of indexing a direct-mapped cache by a paged virtual address. We assume that the length of the page number is P-bit, the index length is I-bit, and the tag length is T-bit. When the cache size is larger than the page size, the (P-T) least-significant bits of the page number are regarded as a part of the index field which is used to select a cache line. This situation occurs regardless of the virtual index or physical index cache. According to the March algorithm, the test process must perform the read/write operations from index 0 to index $2^I$-1, and vice versa. Therefore, these (P-T) least-significant bits of the virtual page number of the starting page must be zero.

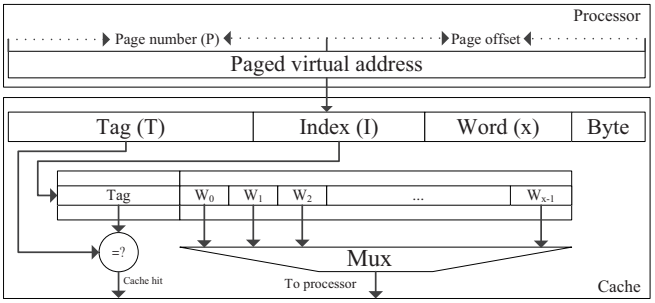


Fig. 2. Indexing a direct-mapped cache by a paged virtual address

When performing a March test on a non-OS platform, these (P-T) least-significant bits can be freely set to zeros to meet the March accessing order. However, the pages allocated by the operating system to store the test data may lead to the misalignment problem. By the way, if the cache is n-way set associative and the one-way size is larger than the page size, the misalignment problem still exists in the platform.

### *B. Processor Shield Software Framework*

A SBST process has to get the supervisor privilege for security requirements so adding a system call to initiate the test process is an attractive way to perform the on-line test for the system users. Fig. 3 shows the execution flow of the proposed SBST system call. The major functions for the cache test environment initialization include the test data allocation, the manuscript page table generation for testing, the SBST process context backup for recovery, the processor control setting, the DVFS control setting, and recording of the operating environment variables.

First, the SBST process requests free pages from the operating system to store the test data for testing the data cache, and then the corresponding page descriptors are recorded. The page descriptors can be obtained from special system calls, e.g., the virt_to_phys Linux system call which can provide the physical page number. Another way is to let the SBST process itself perform the page table walk to get the descriptors. These descriptors will be used to construct the manuscript testing-oriented page table.

The SBST process is implemented as a system call, and its page table is dispatched by the operating system. The manuscript page table for testing is constructed by the SBST process itself, and the comparison between these two page tables is shown in Fig. 4. Each table entry includes the virtual page number, the physical page number, the cacheable bit, the cache policy, and the other attributes.

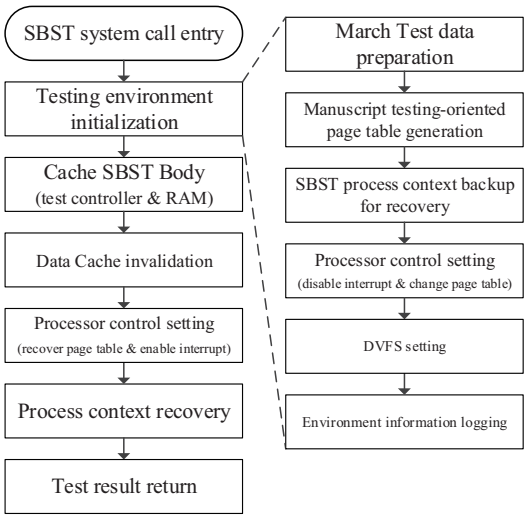


Fig. 3. The SBST system call exectuion flow

In order to seamlessly switch between the OS dispatched page table and the manuscript page table, the SBST process has to build shadow pages for the code segment and the control-dependent data segment in the manuscript page table. Most of the page descriptors are directly copied from the OS dispatched pages except for the cacheable bit. If the pc-relative constants or variables are used in the SBST program, the code pages must be non-cacheable; otherwise, they can be cacheable. All control-dependent data pages must be set non-cacheable.

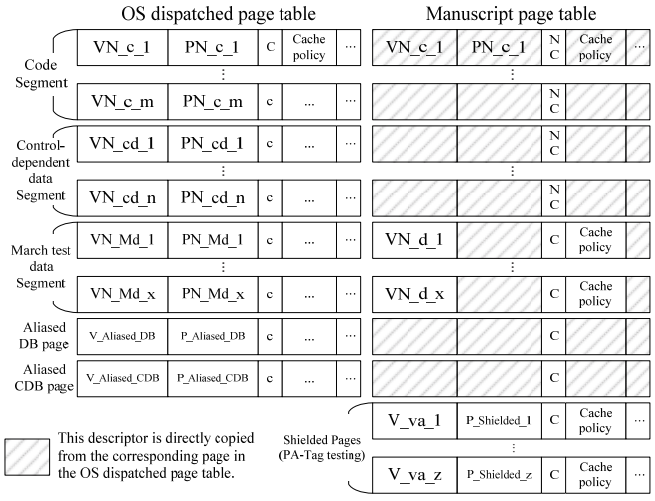


Fig. 4. Comparsion between two page tables

To test virtual tag RAM and data RAM, the March test data segment provides the required memory space to perform the March read/write operations. In the manuscript page table, the virtual page number (VN_d_x), the cacheable bit (C), and the cache policy of data RAM pages are set manually. To avoid the non-zero index problem, the (P - T) least-significant bits

of the virtual page number of the starting page must be zeros, and the virtual page numbers of data RAM pages have to be a continuous sequence.

To test physical tag RAM and the controller, the SBST process has to access various physical pages through the physical addresses which conform to the March DBs or the March CDBs. These physical pages may be shielded pages, and the SBST process needs to build a table entry for each shielded page in the manuscript page table. Then, the SBST process requests two physical pages from the operating system, and we call these pages **aliased DB page** and **aliased CDB page** respectively. All the shielded physical pages will be redirected to the aliased DB page or the aliased CDB page. The redirection function is provided by the processor shield DFT hardware. How to use the aliased pages to test physical tag RAM will be described in the section III-C.

After the manuscript testing-oriented page table is prepared, the SBST process has to back up its process context, including all register contents, processor status, and MMU status. Then, the SBST process sets processor controls, including disabling the interrupt, switching to the manuscript page table, and cleaning the TLB. When these processor controls have been set in sequence, the SBST process becomes a non-preemptive process which uses the manuscript testing-oriented page table.

Before entering the SBST body which includes the controller and the RAM tests, the SBST process can set the desired operating voltage and frequency through the DVFS mechanism. First, if the test fails on the maximum operating voltage and the minimum frequency, the faults are likely operational faults. As shown in Fig. 5, the operating voltage is set to the minimum, and the frequency is set to the expected maximum, which is workable previously. If the test fails, the SBST process decreases the frequency and iterates the test. Until the test is passed, the frequency is the current workable maximum, which may be degraded by the transistor aging. The maximum workable frequency on the any operating voltage can be obtained by performing the above iteration.

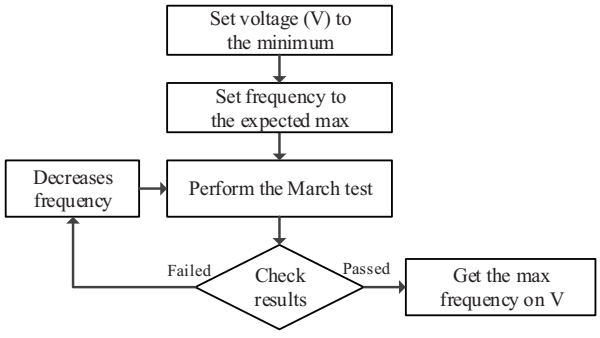


Fig. 5. How to get the maximum workable frequency on the minimum operating voltage

During the chip lifetime, the DVFS system can reduce the aging guardbands in the beginning and then increase the required guardbands according to the maximum workable frequencies on various operating voltages. The lower guardbands imply the lower operating voltage or the higher working frequency, and the desired balance between the power and performance can be achieved effectively. The SBST process can record the environment information, including the temperature, the operating voltage and frequency. The information and the test result are passed to the DVFS system to calibrate the required guardbands.

After setting of the operating voltage and frequency, the SBST process can perform the test body, including the controller and RAM tests. When the test body is finished, the SBST process needs to invalidate all the cache lines. The SBST process sets the processor controls to return the control to the OS. The processor control sequence includes switching to the OS dispatched page table, cleaning the TLB, and enabling the interrupt.

According to the proposed software framework, the SBST process can perform an on-line D-cache test under the operating system and cooperate with the DVFS system to calibrate the required guardbands. The remaining system related issues are the shielded address redirection and the faulty effect isolation. The software framework can block the part of faulty effects to prevent the system from crashing.

The RAM of the data cache can be classified into three modules, including data RAM, virtual tag RAM, and physical tag RAM. The SBST process has arranged all the constants and control variables in the non-cacheable pages so a faulty data RAM cell cannot change the program flow since the required variables come from the main memory rather than the D-cache. A faulty virtual tag RAM cell only increases the cache misses since the virtual tag is used to distinguish whether the target line is a hit or not. If there is a cache miss, the virtual address used to query the MMU comes from the processor rather than virtual tag RAM.

However, the values of physical tag RAM are directly used to access the next level memory when the cache performs the clean or replacement on a dirty cache line. Consequently, the DFT hardware is necessary to block faulty accesses when testing physical tag RAM.

### *C. The Physical Tag RAM Testing*

To test physical tag RAM, the SBST process writes the target cache line a datum which is different from that set in the same address of the next level memory. This cache line is marked as dirty. When the SBST process performs a clean operation on the target dirty line, its physical page number in physical tag RAM is directly used to copy the dirty line to the next level memory. If the copied-back datum matches the one in the next level memory, the cached physical page number is deemed correct.

According to the word-oriented March algorithm [2], there are several pairs of the March DB and CDB, and each March DB/CDB represents a physical page number. The SBST process has to perform a complete March execution flow for each pair of the March DB and CDB. Before performing a March run, the SBST process has to configure the processor shield DFT with two target physical page numbers, i.e., the March DB and CDB. The March DB page and the March CDB page will be redirected to the aliased DB page and the aliased CDB page, respectively. Other physical pages will be blocked by the DFT hardware since they are the faulty targets generated by the faulty physical tag RAM.

### D. Processor Shield DFT Hardware

Fig. 6 shows the processor shield DFT which is inserted between the processor and the system bus to block faulty accesses and redirect shielded addresses. The processor shield DFT works like a bus wrapper and can be implemented as either an MMIO device or a coprocessor. An MMIO implementation requires a bus slave interface and the system memory mapping. Consequently, to implement the processor shield DFT hardware like a coprocessor is a relative low-cost method. For the DFT hardware, the configuration parameters are passed through the coprocessor interface, and each specified coprocessor instruction triggers the corresponding function of the DFT hardware.

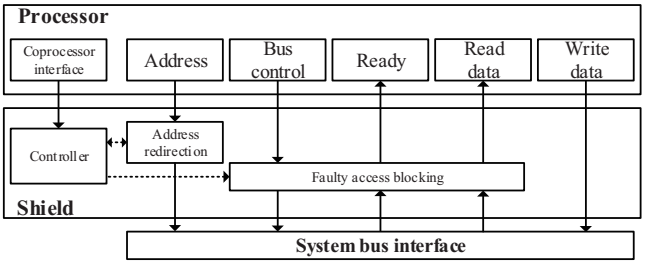


Fig. 6. Processor shield DFT hardware architecture

During on-line testing, a faulty physical tag RAM will lead to faulty accesses which use unexpected physical addresses. The DFT hardware has to block these faulty accesses to keep other processes unaffected. When a faulty access comes, the DFT hardware directly replies a success signal to the processor without accessing the system bus. Thus, this faulty access can't appear on the system bus, and the SBST process can go on the expected execution flow.

## IV. Data Cache Software-Based March Development

In this section, we present the basic pseudo test instructions, and show how to use these instructions to build the March read/write operations for each memory module.

### A. Pseudo Test Instructions

Pseudo test instructions, which can be easily translated into any target ISA, are used to improve portability of developed test programs [5]. To index the data cache, the entire virtual address can be divided into tag, index and word, and we use T, I, and x to denote tag, index, and word fields, respectively. The symbol [T,I,x] denotes a virtual address. Table 1 shows all the pseudo instructions used in this paper and their functions. For all the pseudo read instructions, including R, R, and RM, the value D is the expected value, but it may be changed by a faulty RAM.

We assume that the cache content remains unchanged after executing the disable cache (DC) instruction. The ADB/ACDB instruction passes the aliased DB/CDB page number to the DFT hardware. For instance, the ADB can be translated to two native instructions. The first instruction fetches the aliased DB page number, which is stored the control-dependent data segment, to a general register, and the second delivers the content of the register to the DFT hardware through the coprocessor interface.

Table 1. Pseudo instructions and their functions

| Inst. | Function |
|---|---|
| R([T,I,x],D)<br>R([T,I,x],D) | Read value D from the cache, address [T,I,x]<br>R : cache hit; R : cache miss |
| W([T,I,x],D)<br>W([T,I,x],D) | Write value D to the cache, address [T,I,x]<br>W : cache hit; W : cache miss |
| RM([T,I,x],D)<br>WM([T,I,x],D) | Directly read/write from/to the main memory<br>(They are only effective when the cache is disabled) |
| INV(I) | Invalidate the I-th line of the cache |
| CLEAR(I) | Clear the I-th line of the cache |
| EC / DC | Enable/disable cache |
| EPS / DPS | Enable/disable processor shield DFT hardware |
| ADB(PN)<br>ACDB(PN) | Deliver the aliased DB/CDB page number (PN) to the DFT |
| SDB(PN)<br>SCDB(PN) | Deliver the March DB/CDB page number(PN) to the DFT |

### B. March Read/Write Sequences for Each RAM Module

In this subsection, we introduce the March read/write implementation for each RAM module, and the widths of the March DBs depend on the line size of each memory module.

■ Data RAM:

```
wDB: DC; WM([T,I,x],DB); EC; INV(I); R([T,I,x],DB)
rDB: R([T,I,x],DB)
```

The SBST process writes a DB to memory and performs a read miss to move a DB into the target cache line. Then, the following read is a read hit, and the expected result is the DB.

■ Virtual Tag RAM:

```
wDB: DC; WM([DB,I,x],D); EC; INV(I); R([DB,I,x],D)
rDB: DC; WM([DB,I,x],!D); EC; R([DB,I,x],D)
```

To test virtual tag RAM, the March DB is the virtual page number, and it is applied to the tag (T) field instead of the data line. A read miss let the cache fill the target data line with the value D, and the March DB is written to the tag RAM. Before performing a read which is expected as a read hit, the SBST process writes the value !D to the same address in the main memory. If the target tag RAM is faulty, the read becomes a read miss and its result is the value !D.

■ Physical Tag RAM:

```
wDB: EPS; EC; INV(I); W([T,I,x],D); DPS
rDB: EPS; DC; WM([T,I,x],!D); EC; CLEAR(I); DC; RM([T,I,x],D); DPS
```

To test physical tag RAM, the March DB is a physical page number, and its mapped virtual page number is applied to the virtual tag (T) field. The cache write-miss policy is assumed to be write-allocate. A write miss let the cache write the value D to the target data line, and this line is marked as dirty. Before performing the clean operation on the target dirty line, the SBST process writes the value !D to the same address in the main memory. If the target RAM is faulty, the result of read from memory is the value !D instead of the value D.

## V. Case Study and Experimental Results

In this section, we present a case study that executes the SBST program under Linux kernel on an ARMv5-compatible pipelined processor, which has 16KB direct-mapped data cache. In this platform, the processer and the cache system are

implemented in Verilog, and the other system devices, including the system bus, the main memory, the interrupt controller, the timer, and other peripherals, are implemented in SystemC. We capture the stimuli for the test result analysis during performing the co-simulation, and then we use RAMSES simulator [6] to run the RAM fault simulation and Syntest Turboscan to execute the controller fault simulation.

We synthesize the processor core, the cache controller, and the processor shield DFT by TSMC 40nm technology library, and the maximum working frequency is set at 1GHz. The area results are shown in Table 2. The total hardware overhead is 2.1% compared to the logic part of the ARMv5-compatible processor. The additional address latency between the processor bus and the system bus is 0.06ns due to the insertion of the processor shield DFT hardware.

Table 2. Hardware Synthesis Results

| | Core | Cache controller | Processor Shield DFT |
|---|---|---|---|
| Area | 70,880 | 2,731 | 1,488 |

### *A. Memory Module Test Results*

The processor performs the software-based March algorithm on each target memory module. We implement the March C- algorithm which can effectively capture stuck-at-fault (SAF), transition fault (TF), address decoder fault (AF), state coupling fault (CFst), inversion coupling fault (CFin), and idempotent coupling fault (CFid). According to the fault coverage reported by RAMSES, the proposed on-line SBST run in Linux can achieve the same fault coverage, i.e., 100%, as of that runs on a non-OS platform.

### *B. Test Results of the Cache Logic Devices*

The cache controller implements seven control functions for the data cache: enable/disable, invalidate, clear, write-back, write-through, write-allocate, and write-around policy. We feed the netlist and the stimuli to Syntest Turboscan for the fault coverage simulation. The proposed on-line SBST process can capture 41699 of 42125 collapsed faults of the cache logic devices (98.99% fault coverage). Table 3 shows test results of the cache controller and other logic devices.

Table 3. Test results for the collapsed stuck-at faults

| Module | Detected faults | Undetected + Untestable faults | Total faults | Fault coverage |
|---|---|---|---|---|
| Controller | 1,437 | 108 | 1,545 | 93.01 |
| Multiplexer | 1,034 | 9 | 1,043 | 99.14 |
| Valid unit | 12,980 | 86 | 13,066 | 99.34 |
| Dirty unit | 25,681 | 199 | 25,880 | 99.23 |
| Others | 567 | 24 | 591 | 95.94 |
| Total | 41,699 | 426 | 42,125 | 98.99 |

### *C. SBST Process Statistics*

Table 4 shows the code size, memory usage, and execution times of each component of the data cache. The total code size is about 33KB, and the execution time is nearly 1.25 million processor cycles. The total memory usage is 75.5 KB, including 32KB March test data pages and 8 KB aliased pages, and this means the proposed SBST process can easily be applied to small embedded systems. Since processor shield redirects all shielded pages to aliased pages, only two physical pages are required to test physical tag RAM and logic devices.

Table 4. Data cache SBST process statistics

| Target | Code size (KB) | Memory usage (KB) | | Execution time (CPU cycle) |
|---|---|---|---|---|
| Data RAM | 0.69 | 0.99 | 32 KB March test pages | 327,432 |
| Tag RAM | 2.77 | 3.21 | | 132,543 |
| Phy. Tag RAM | 5.24 | 6.17 | 8 KB Aliased Pages | 157,331 |
| Logic device | 24.11 | 25.13 | | 631,521 |
| Total | 32.81 | 75.5 | | 1,248,737 |

## VI. Conclusion

This paper has addressed system related issues during on-line L1 data cache SBST, including performing the test non-preemptively, constructing the manuscript testing-oriented page table, redirecting *shielded physical pages*, blocking the faulty effects, and returning the control to the operating system. The proposed processor shield is an architectural solution, including the software framework and the DFT hardware. The processor shield can recover the SBST process context as if the test were never executed. Thus, the SBST process can be iteratively executed under various conditions, including the voltage, the frequency, and the temperature, even if the test fails. The processor shield can cooperate with the DVFS system to calibrate the required grandbands against the transistor aging. The SBST process can completely run under the operating system without affecting other processes and on-bus devices. We demonstrate the methodology using a case study that executes the SBST program under the Linux kernel on an ARMv5-compatible processor system. Our method can successfully switch between the SBST process and the kernel process and achieve the expected fault coverage as of that run on a non-OS platform. The hardware overhead of the processor shield is nearly 2.1% compared to the logic part of the processor.

## References


[1] J. Abella, X. Vera, and A. Gonzalez, "Penelope: The NBTI-aware processor," in *Proc. IEEE/ACM Int'l Symp. Microarchitecture (Micro)*, Dec. 2007, pp. 85–96.

[2] A. J. van de Goor, I. B. S. Tlili, and S. Hamdioui, "Converting march tests for bit-oriented memories into tests for word-oriented memories," in *Proc. Int. Workshop on Memory Technology Design and Testing (MTDT)*, Aug. 1998, pp. 46–52.

[3] S. Di Carlo, P. Prinetto, and A. Savino, "Software-based self-test of set-associative cache memories," *IEEE Trans. on Computer*, vol. 60, no. 7, pp. 418–423, July 2011.

[4] G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Software-based self test methodology for on-Line testing of L1 Caches in multithreaded multicore architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 4, pp. 786–790, Apr. 2013.

[5] Y.-C. Lin, Y.-Y. Tsai, K.-J. Lee, C.-W. Yen, and C.-H. Chen, "A softwarebased test methodology for direct mapped data cache," in *Proc. Asian Test Symp. (ATS)*, Nov. 2008, pp. 363–368.

[6] C.-F. Wu, C.-T. Huang, and C.-W. Wu, "RAMSES: a fast memory fault simulator," in *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Syst. (DFT)*, Nov. 1999, pp. 165–173.

[7] G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Software-based self-test for small caches in microprocessors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 12, pp. 1991–2004, Dec. 2014.

[8] S. Di Carlo, G. Gambardella, M. Indaco, D. Rolfo, and P. Prinetto, "MarciaTesta: An automatic generator ot test programs for microprocessors' data caches," in *Proc. Asian Test Symp. (ATS)*, Nov. 2011, pp. 401–406.