A unifying formalism to support automated synthesis of SBSTs for embedded caches

A unifying formalism to support automated synthesis of SBSTs for embedded caches

Authors: Di Carlo S., Gambardella G., Indaco M., Rolfo D., Prinetto P.,

Published in the Proceedings of the IEEE East-West Design & Test Symposium (EWDTS), 9-12 Sept. 2011, Sebastopoli, UA.

DOI: 10.1109/EWDTS.2011.6116421

A unifying formalism to support automated synthesis of SBSTs for embedded caches[1]

Stefano Di Carlo*, Giulio Gambardella†, Marco Indaco*, Daniele Rolfo†, Paolo Prinetto*
†CINI
Via Ariosto 25, 00185 Roma, Italy
Email: {*FirstName.LastName*}@consorzio-cini.it
*Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
Email: {*FirstName.LastName*}@polito.it

***Abstract*—The paper presents a new unifying formalism introduced to effectively support the automatic generation of assembly test programs to be used as SBST (Software Based Self-Testing) for both data and instruction cache memories. In particular, the new formalism allows the description of the target memory, of the selected March Test algorithm, and the way this has to be customize to adapt it to the selected cache.**
Since the related synthesis tool is multi-platform, the Instruction Set Architecture of the target processor must be properly described, as well. The newly introduced formalism supports such a description at the different abstraction levels.
Several applications examples are provided.

I. Introduction

In modern microprocessors, memories have become the most critical part of the global design. In particular, caches are the most widely used and their testing has become mandatory, for both instruction and data, owing to their presence in many embedded designs [1].
Various types of solutions have been proposed to tackle this problem. Typical approaches consists of hardware implementation of testing infrastructures, such as memory built-in-self-test (MBIST), that may unfortunately occur in power, timing and area overhead. This kind of overhead is often not affordable in embedded systems [2], where instead SBST (*Software-Based Self-Testing*) plays an important role as an alternative testing methodology. SBST does not imply increases in terms of area and it consists of an algorithm, usually written in assembly language, exploiting the instruction set of the target processor, thus allowing at-speed test.
State-of-the-art approaches for memory testing mostly refer to March Tests, [3], that have been defined for both bit-oriented and word-oriented memories [4].
Compared to Static Random Access Memories (SRAMs), cache memories have peculiar characteristics and properties, including, among the others, the addressing style and the limited accessibility due to their behaviour within the microprocessor system. In fact, different write policies can be set-up, depending on design needs. Nevertheless, being the physical implementation of cache memories mostly based on SRAMs, the two families of memories are affected by the same kind of faults [5].
Several march test methodologies for caches were presented in literature [6] [7] [8] [9].
Notwithstanding the definition of valuable solutions, each approach is expressed resorting to different formalisms, thus causing a lack of a consistent scheme for defining the translation's approach.
This paper aims at overcoming this problem, introducing a unifying formalism to support the overall SBST cache testing methodology.
Despite the significant presence of cache testing methods, at our best knowledge no automated SBST generation tool have so far been presented. The assembly program formulation is left to the test engineer that could introduce errors. To fill the gap we presented in [10] MarciaTesta, an automatic assembly generation tool. The formalism presented in this paper has been used as input to MarciaTesta.
The paper is organized as follows: section II presents the MarciaTesta tool in which the formalism is used, while section III introduces the formalism used for the software-based self-test methodology, the March Test description, and the memory configuration. Section IV describes the methodology of the processor instruction set definition, based the nML formalism [11]. Section V eventually concludes the paper.

II. The MarciaTesta tool

MarciaTesta is a general purpose multi-platform tool able to generate assembler programs that implement the chosen March Test for the target processor data cache.
Three translation levels are set up for the *ASM* generation (Figure 1). The first step aims at transforming a generic March Test into a data cache one, exploiting a predefined *SBST methodology*. The description of the *SBST methodology* allows to translate a given march test into a new one, suitable for cache memories. The *MT2CACHE* program performs this first step of translation.

[1]This projects is partially funded by Ansaldo STS SpA and FinMeccanica within the "Iniziativa Software" (II ediz) framework

The translated March Test is then processed by *CACHE2C* program, using the *Target memory configuration* information, that transforms it into a *C/C++* program. This intermediate output is particularly suited for the test emulation on the target cache architecture.

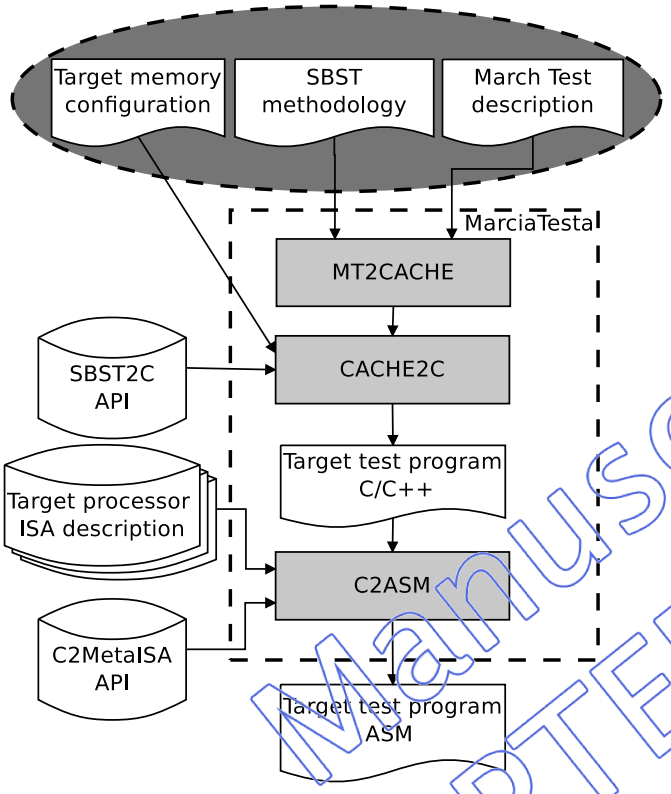

Fig. 1. MarciaTesta tool architecture

The third step translates the *C/C++* cache test implementation into the assembly program. *C2ASM* script parses the *C/C++* program extracting the key parameters (e.g., addresses and data background patterns) and writes the assembly algorithm, targeted to the chosen microprocessor.
These levels of translation set up in MarciaTesta tool requires straightforward definitions of the rules to be applied.
The first translation, performed by *MT2CACHE* program, needs an unambiguous definition of both inputs, i.e., (a) the *SBST methodology* that lists the rules for the March Test translation and (b) the March Test to be translated. The second step requires the *Target memory configuration* description, listing the main properties of the target architecture.
Finally a clear description of the processor instruction set (*Target processor ISA description*) is needed to write the assembly program correctly.
In the next sections the new formalisms are described in details.

III. SBST Methodology Rules

MarciaTesta is characterized by a very high flexibility and a remarkable user-friendliness. These include, among the other, the possibility offered to the user of freely selecting the preferred *SBST Methodology*. To fully support such a peculiar feature, a new formalism had to be introduced.
This new formalism is defined in terms of a set of rules, grouped into *SBST Methodology rules*, capable of supporting the majority of the proposed *SBST Methodologies*.
Table I summarises the available rules.

TABLE I
SBST Methodology Rules

Rule Name	Rule description
EC	Enable cache
DC	Disable cache
FC	Flushes cache (Invalidates all cache lines)
FCCL	Invalidates current cache line
WC (*tag,d*)	Writes data *d* in cache
RC (*tag,d*)	Reads data from cache and verifies if it is equal to *d*
WM (*tag,d*)	Writes data *d* in main memory
U{...}	Repeats in-brackets rules with upward addressing
D{...}	Repeats in-brackets rules with downward addressing

The last two rules (*U* and *D*) are used just when the methodology requires one or more initialization march elements, since otherwise the addressing order is defined by each march element.
The *tag* parameter is useful to specify the value written into the cache memory directory array and it can get three different values:

- *DT*: data background associated with the cache directory array test
- *DTn*: complemented data background associated with the cache directory array test
- *Any*: cache directory array assumes the value given by the addressing order defined by input

The parameter *d* specifies the data written in cache and it can assume five different values:

- *DB*: data background written in cache
- *DBn*: complemented data background written in cache
- *pDB*: last written data background
- *pDBn*: complemented value of the last written data background

Table II lists the set of available rules that are necessary to describe all the proposed March Test.

The third input of MarciaTesta tool is the *Target memory configuration* file, which details all the significant features of the target cache. Thanks to this approach, the tool is able to customize the test to the target architecture, fulfilling the properties and the constraints of the cache memory.

To describe the *SBST Methodology* we implement an interface with a set of methods, named *March element operations*. In his/her *SBST Methodology* description, the user can define each method (*Init*, $w0$, $w1$, $r0$ and $r1$) resorting to a set of

TABLE II
MARCH TEST DESCRIPTION RULES

Rule Name	Rule description
W0	Write of 0 (*DBn*)
W1	Write of 1 (*DB*)
R0	Read and verify to have 0 (*DBn*)
R1	Read and verify to have 1 (*DB*)
U(...)	Execution of march elements with ascending addressing order
D(...)	Execution of march elements with descending addressing order
A(...)	Execution of march elements with any addressing order

TABLE III
TARGET MEMORY DESCRIPTION RULES

Rule Name	Rule description
O	LSBs that identify a specific word into the cache line
I	Middle bits that identify the cache line
T	MSBs that identify the content of the directory array
Policy	Specify if the cache is Write-Back or Write-Through
Word size	Data-width inside the target system
Base Address	Base address for memory

available rules, in order to implement correctly the methodology.
For a better comprehension, Table IV shows two examples based on different SBST methodologies, presented in [6] and [9].

TABLE IV
EXAMPLE OF SBST METHODOLOGY DESCRIPTIONS FOR TESTING DIRECTORY ARRAY OF WRITE-BACK DATA CACHE

March element operation	**SBST Methodology**	
	[6]	[9]

init	U{
	WC (DT,DB)
	}
	U{
	WC (DTn,DB)
	}
		None

$w0$	FCL WC (DTn,pDBn)	WC (DTn,DB) DC WM (DTn,DBn) EC
$w1$	FCL WC (DT,pDBn)	WC (DT,DB) DC WM (DT,DBn) EC
$r0$	RC (DTn,pDB)	RC (DTn,DB)
$r1$	RC (DT,pDB)	RC (DT,DB)

IV. META-ISA FORMALISM

The main output of the MarciaTesta tool is the generated assembly program (*Target test program ASM*). In order to automatically generate this program, a standard input is required for the tool. This input has to list the *ASM* instructions of the target processor.
The first idea for this multi-platform tool was to make a library for each microprocessor with the assembly implementation of each operands. Nevertheless, the library composing would be very difficult, and request a big effort whenever a new processor library is described. Therefore a new approach had to be studied, in order to reduce the effort for the library description. For this purpose a set of *Meta ISA* have been outlined. This list include all the required and sufficient instructions for the translation of the operations required during a cache test. Each of them are then described using a *nML* description [12], as in Table V.
Thanks to this new technique, that moves the intelligence of the assembly algorithm generation to the tool, MarciaTesta users can easily compose libraries for their target processors.
The identified Meta ISAs are:

- *Register32write*: write operation in a register of a 32 bit immediate.
- *Register16write*: write operation in a register of a 16 bit immediate.
- *MemoryWrite*: write operation in central memory.
- *MemoryRead*: read operation from central memory.
- *LoopInstruction*: instruction for a loop operation.
- *AddInstruction*: addition operation on a register of a 16 bit immediate.
- *SubInstruction*: subtract instruction on a register of a 16 bit immediate.
- *Enable Cache*: instruction that enables the cache.
- *Disable Cache*: instruction that disables the cache.
- *Invalidate Cache Line*: instruction that invalidates a cache line.

TABLE V
EXAMPLES OF META ISAS NML DESCRIPTION

MetaISA	nML description
Register32write	$opRegister32write(data : value32, i : index, reg : R)$ $action = \{$ $reg[i] = data;$ $\}$
MemoryWrite	$opMemoryWrite(reg : R, i : index, a : addr, mem : M)$ $action = \{$ $mem[a] = reg[i];$ $\}$
MemoryRead	$opMemoryRead(mem : M, i : index, a : addr, reg : R)$ $action = \{$ $reg[i] = mem[a];$ $\}$
Enable Cache	$opEnableCache()$ $action = \{$ $MSR = ENABLE\_CACHE;$ $\}$
Invalidate Cache Line	$opInvalidateCacheLine(j : index, i : index, reg : R)$ $action = \{$ $DCache\_Tag = reg[i];$ $DCache\_Data = reg[j];$ $\}$

In *Target processor ISA description* all the Meta ISA are described using the assembly instruction of the processor.

TABLE VI
META ISAS TRANSLATION FOR NIOSII PROCESSOR

Meta ISA	***NiosII***
Register32write	$movhi < regA >, \%hi(< value >);$ $ori < regA >, < regA >, \%lo(< value >);$
MemoryWrite	$stw < regA >, < IMM > (< regB >);$
MemoryRead	$ldw < regA >, < IMM > (< regB >);$
Enable Cache	*None*
Invalidate Cache Line	$flushd < IMM > (< regA >);$

TABLE VII
META ISAS TRANSLATION FOR MICROBLAZE PROCESSOR

Meta ISA	***MicroBlaze***
Register32write	$ori < regA >, < regB >, < value >;$
MemoryWrite	$sw < regA >, < regB >, < regC >;$
MemoryRead	$lw < regA >, < regB >, < regC >;$
Enable Cache	$msrclr < regA >, < IMM >;$
Invalidate Cache Line	$wdc < regA >, < regB >;$

V. CONCLUSION

During the MarciaTesta tool design, many inputs has been described. An intelligent approach is to isolate each of them and study a flexible solution that allows their description.
In order to create a general purpose tool, the inputs description of MarciaTesta has been fully described during preliminary analysis. Each of them has been accurately described using formalisms that gives flexibility to the user.
First of all, in section III the software-based self-test description gives leave to use the tool with different cache testing approaches, and frees the user to select the best for his purpose. Then the formalism explain how to define the March Test operation in the SBST methodology and how to define correctly the memory configuration.
Finally the straightforward description of meta-ISAs given in section IV, is really essential for the *Target processor ISA library* establishment.

ACKNOWLEDGMENT

The authors would like to express their sincere thanks to the whole design team of Ansaldo STS SpA for their helpful hints and guidelines.

REFERENCES

[1] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *Proc. 30th Annual Internation Symposium on Computer Architecture*, no. 7, pp. 136–146, 2003.
[2] S. Hamdioui, G. Gaydadjiev, and A. Van de Goor, "The state-of-art and future trends in testing embedded memories," in *Proc. Records of the IEEE International Memory Technology, Design and Testing Workshop*, no. 7, pp. 54–59, 2004.
[3] A. Van De Goor, "Using march tests to test srams," *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 8–14, 1993.
[4] A. Van De Goor, I. Tlili, and S. Hamdioui, "Converting march tests for bit-oriented memories into tests for word-oriented memories," in *Proc. Records of the IEEE International Memory Technology, Design and Testing Workshop*, pp. 46–52, 1998.
[5] S. Di Carlo and P. Prinetto, *Models in Hardware Testing*, ch. Models in Memory Testing, From functional testing to defect-based testing, pp. pp. 157–185. Springer, 2010.
[6] S. Di Carlo, P. Prinetto, and A. Savino, "Software-based self-test of set-associative cache memories," vol. 60, no. 7, pp. 1030–1044, 2011.
[7] S. M. Al-Harbi and S. K. Gupta, "A methodology for transforming memory tests for in-system testing of direct mapped cache tags," in *Proc. 16th IEEE VLSI Test Symposium*, pp. 394–400, 1998.
[8] J. Sosnowski, "In-system testing of cache memories," in *Proc. IEEE International Test Conference*, pp. 384–393, 1995.
[9] J. Sosnowski, "Improving software based self - testing for cache memories," in *Proc. 2nd International Design and Test Workshop*, pp. 49–54, 2007.
[10] S. D. Carlo, G. Gambardella, M. Indaco, P. Prinetto, and D. Rolfo, "Marciatesta: an automatic generator of test programs for microprocessors data caches," in *Submitted to 20th Asian Test Symposium*, 2011.
[11] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nml," in *Proc. European Design and Test Conference*, pp. 503–507, 1995.
[12] TestGroup, "Meta-isa description using nml formalism." Internal Report.