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

Original
A unifying formalism to support automated synthesis of SBSTs for embedded caches / DI CARLO, Stefano; Gambardella, Giulio; Indaco, Marco; Rolfo, Daniele; Prinetto, Paolo Ernesto. - STAMPA. - (2011), pp. 39-42. (Intervento presentato al convegno IEEE East-West Design & Test Symposium (EWDTS) tenutosi a Sebastopoli, UA nel 9-12 Sept. 2011) [10.1109/EWDTS.2011.6116421].

Availability:
This version is available at: 11583/2460388 since:

Publisher:
IEEE Computer Society

Published
DOI:10.1109/EWDTS.2011.6116421

Terms of use:
This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)
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.

N.B. This is a copy of the ACCEPTED version of the manuscript. The final PUBLISHED manuscript is available on IEEE Xplore®:

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6116421

DOI: 10.1109/EWDTS.2011.6116421

© 2011 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
A unifying formalism to support automated synthesis of SBSTs for embedded caches

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, 1-10129, 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 not always affordable in embedded systems [2], where instead power, timing and area overhead. This kind of overhead is built-in-self-test (MBIST), that may unfortunately occur in implementation of testing infrastructures, such as memory

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 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].

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.

†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.

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.

Table I summarises the available rules.

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

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
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 III
TARGET MEMORY DESCRIPTION RULES

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Rule description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>LSBs that identify a specific word into the cache line</td>
</tr>
<tr>
<td>I</td>
<td>MSBs that identify the content of the directory array</td>
</tr>
<tr>
<td>T</td>
<td>Policy: Specify if the cache is Write-Back or Write-Through</td>
</tr>
<tr>
<td>Word size</td>
<td>Data-width inside the target system</td>
</tr>
<tr>
<td>Base Address</td>
<td>Base address for memory</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>March element operation</th>
<th>SBST-Methoodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>None</td>
</tr>
<tr>
<td>w0</td>
<td>WC (DT,DB)</td>
</tr>
<tr>
<td></td>
<td>WC (DTn,DB)</td>
</tr>
<tr>
<td>w1</td>
<td>WC (DT,DB)</td>
</tr>
<tr>
<td></td>
<td>WC (DTn,DB)</td>
</tr>
<tr>
<td>r0</td>
<td>RC (DTn,DB)</td>
</tr>
<tr>
<td>r1</td>
<td>RC (DT,DB)</td>
</tr>
</tbody>
</table>

### 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 operand. 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

<table>
<thead>
<tr>
<th>MetaISA</th>
<th>nML description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register32write</td>
<td></td>
</tr>
<tr>
<td>MemoryWrite</td>
<td></td>
</tr>
<tr>
<td>MemoryRead</td>
<td></td>
</tr>
<tr>
<td>Enable Cache</td>
<td></td>
</tr>
<tr>
<td>Invalidate Cache Line</td>
<td></td>
</tr>
</tbody>
</table>

**Register32write**

\[
\text{opRegister32write}(\text{data: value32}, i, \text{index, reg : R}) \\
\text{action} = \{ \\
\text{reg[i] = data}; \\
\} \\
\]

**MemoryWrite**

\[
\text{opMemoryWrite}(\text{reg : R, i, \text{index, a : addr, mem : M})} \\
\text{action} = \{ \\
\text{mem[a] = reg[i]}; \\
\} \\
\]

**MemoryRead**

\[
\text{opMemoryRead}(\text{mem : M, i, \text{index, a : addr, reg : R})} \\
\text{action} = \{ \\
\text{reg[i] = mem[a]}; \\
\} \\
\]

**Enable Cache**

\[
\text{opEnableCache}() \\
\text{action} = \{ \\
\text{MSR = ENABLE_CACHE}; \\
\} \\
\]

**Invalidate Cache Line**

\[
\text{opInvalidateCacheLine}(j, \text{index, i, \text{index, reg : R})} \\
\text{action} = \{ \\
\text{DCache_Tag = reg[i]}; \\
\text{DCache_Data = reg[j]}; \\
\} \\
\]
In Target processor ISA description all the Meta ISA are described using the assembly instruction of the processor.

<table>
<thead>
<tr>
<th>Meta ISA</th>
<th>NiosII</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register32write</td>
<td>movhi &lt; regA &gt;, %hi(&lt; value &gt;); ori &lt; regA &gt;, &lt; regA &gt;, %lo(&lt; value &gt;);</td>
</tr>
<tr>
<td>MemoryWrite</td>
<td>stw &lt; regA &gt;, &lt; IMM &gt; (&lt; regB &gt;);</td>
</tr>
<tr>
<td>MemoryRead</td>
<td>ldw &lt; regA &gt;, &lt; IMM &gt; (&lt; regB &gt;);</td>
</tr>
<tr>
<td>Enable Cache</td>
<td>None</td>
</tr>
<tr>
<td>Invalidate Cache Line</td>
<td>flushd &lt; IMM &gt; (&lt; regA &gt;);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Meta ISA</th>
<th>MicroBlaze</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register32write</td>
<td>ori &lt; regA &gt;, &lt; regB &gt;, &lt; value &gt;;</td>
</tr>
<tr>
<td>MemoryWrite</td>
<td>sw &lt; regA &gt;, &lt; regB &gt;, &lt; regC &gt;;</td>
</tr>
<tr>
<td>MemoryRead</td>
<td>lw &lt; regA &gt;, &lt; regB &gt;, &lt; regC &gt;;</td>
</tr>
<tr>
<td>Enable Cache</td>
<td>msrclr &lt; regA &gt;, &lt; IMM &gt;;</td>
</tr>
<tr>
<td>Invalidate Cache Line</td>
<td>wdc &lt; regA &gt;, &lt; regB &gt;;</td>
</tr>
</tbody>
</table>

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


