TESTING PARAMETRIC CORES: A MULTI-LAYER TEST PROGRAM TO IMPROVE AND AUTOMATE THE EDA-ATE LINK

P. Bernardi[3], A.Bertuzzi[1], M. Grosso[3], V. Tancorre[1], S.Tritto[2]

[1]STMicroelectronics, Agrate, Italy
[2]Agilent Technologies, Cernusco sul Naviglio, Italy
[3]Politecnico di Torino, Torino, Italy

Abstract

In this paper we discuss about a "multi-layer" test program approach. A good scenario where this kind of technique can be used is the handling of parametric cores. Such cores may require several re-executions of the test procedures, varying every time a well defined non-deterministic part of the vectors. This approach allows obtaining precise diagnosis information. Even if for these cases CAD tools providers see a large room for automating test flow generation, currently this kind of cores still requires a considerable amount of custom tools (or even manual work) both on the Electronic Design Automation (EDA) and on the Automatic Test Equipment (ATE) side.

We'll show how a test program could be based on parametric procedures in order to be reused in the design of System-on-Chips (SoCs), and to be easily modified for implementing several different tests or diagnostics algorithms. As highlighted at the end of this paper, such a flexible test program structure would provide the best advantage when inserted in an automated flow for test generation, closing the loop between EDA and ATE environments [1].

Key words: SoC, DfT, Parametric Cores, Cost of Testing, Time to Manufacturing, EDA-ATE link, IEEE 1450.x standard, Test Flow.

I. Introduction

The increasing complexity and multi-functionality of SoCs have a twofold impact on the devices maturity curve, starting from chip design and leading to mass volume production.

On the design side, a multi-core approach is the best fit for building such complex SoCs, also enabled by present and emerging standards (e.g.: STIL or CTL) [2]; the main advantage of this approach is the possibility to rely on consolidated IP cores bringing their own test access mode, beside structural information.

On the testing side, more diagnostic capability and faster feedback to design environment are a key factor for shortening Time-to-Volume and Time-to-Market in the device lifespan.

Up to now, ATE equipment doesn't offer an adequate programming environment that allows users to properly address new testing concepts. In this field, design to manufacturing chain has got a weak link, resulting in the need for moving most of the 'test intelligence' directly on the IC. Moreover, ATE environments capable to support the needed DfT test strategies in a modular fashion are required. In this direction, it is known that in general ATE hardware and programming software are structured in such a way that the tester itself acts as a master (determining the test execution) and the device is the 'slave'.

On the contrary, for all the above reasons, the trend moves towards having devices driving their own test flows, so reversing the master/slave prospective and then using low cost/low performance testers with reduced test data volume and improving time-to-manufacturing window.

The more diagnosis capability is enhanced, the more debugging activity is straightforward in highlighting punctual or structural failures of logic cores.

By moving diagnosis capabilities to silicon, it is possible to define a multi-layer structured algorithm where the lower layer only deals with the ATE, while the others are oriented to the data management. Considering such a structure, device- or algorithm-dependent variables can be programmed in a parametric way, finally no longer requiring the users to rewrite the test program for either changing the device or the diagnostic

approach, but just to modify the layer(s) concerned.

Nowadays, users update these parameters manually. The closer target is to insert this multi-layer tool into a more general automated test environment IEEE 1450.x standard based, allowing both the core parameters description and the test flow development. The reduction of the time for validation and debugging phases allow users to focus on silicon performance.

In Section II a memory BIST case is depicted as parametric core example while in Section III the multi-layer approach is presented. Test time performances and experimental results obtained on a test chip are reported in Section IV. Finally, in Section V, an example of generalization of this approach as a possible framework for automating test flow generation is proposed and conclusions are drawn.

II. BIST of embedded memories: a parametric core example

A "parametric core" is a core whose test procedure is based on a mono- or bi-directional information exchange (parameters) between the ATE and the core. In practical terms, two parametric cores, having the same function, can be described by two different sets of parameters. This kind of core, because of its characteristics, is an ideal candidate to create an automatic test environment that is able to auto-generate a test flow starting from design level device description. A parametric cores example is given by memory arrays, equipped with their own BIST (*Built-In Self-Test*) structures. The test of embedded memories (*eMem*) is mainly approached as the test of the corresponding stand-alone devices (that is applying stimuli and collecting test results for reconstructing fault bitmaps); the difference resides in the solution adopted for applying the test algorithm (usually belonging to the March family) to the cell matrix: BISTs are nowadays popular for test of eMem as they provide an embedded way to generate the test sequences and compressing the outputs.

Unfortunately, BIST architectures are strongly stiff: new design of the BIST architecture is necessary each time the implemented test algorithm has to be changed. Particularly, in the case of embedded memories, where the variety of faults is really high, flexibility in terms of applied test is required.

In order to avoid frequent redesigns we introduced a new architecture: the Micro Programmable BIST (MPB) [3]. MPB architecture relies on a small microprocessor based on an ad-hoc instruction set suitable for memory access. A test program describing a word-oriented algorithm is stored in a RAM module and re-loaded by the user with the code describing the chosen March test. Flexibility is guaranteed from two points of view: easy reuse of the introduced structures and re-programmability of the memory test algorithm.

To make memory diagnostic powerful, DfT features have to be added into BIST design that usually provides only go-no-go signals, but generally doesn't yield information about failure locations. The introduction of hardware architecture compliant with the proposed standard IEEE 1500 [4] allows optimizing the Embedded Core Testing in a System-On-Chip device [5].

Moreover, the use of a standard IEEE 1149.1 TAP is desirable because commercial automation tools generally support it and this interface represent a valid bi-directional way for ATE-DUT communication using only five pins.

III. Multi-Layer Test Program Architecture

Product Engineers (PE) often spend a lot of time in non-value-added activities, such as converting design data into tester pattern data or test programs developing, instead of focusing earlier on silicon performances. Sometime PEs have to update a part of the test program already existing because of changing ATE platform or because DUT has been modified in some features like dimensions or data bus width (in case of memory for example). For these reasons, taking a closer look at the portability concept and aiming at developing a flexible software able to run tests on parametric cores, a multi-layer architecture has been developed; it is able to address all major needs in terms of software re-use and quickness during development phase.

As test case, for our multi-layer architecture, we chose the memory BIST (hardware or programmable) and categorized the parameters to run a certain BIST for a particular IC in the following classes:

1. *Algorithm* includes all parameter describing the algorithm (e.g. March-n) that tests the logic.
2. *Core* describes the size of the logic core to be tested.
3. *BIST* has the information concerning vector cycle numbers of input and output data stream inside the pattern.

4. *Device* includes information that is device dependent such as access pin name.
5. *ATE* that the *information stored* depends on the specific tester platform.

Using a multi-layer architecture, each of the five classes highlighted above can be stored in a correspondent module (or layer) creating a kind of library.

Following this approach, module *Core* can be seen as a library containing several entries that describe different cuts of logic cores (core-1, core-2, ..., core-n). In the same way, module *Algorithm* is the library that contains several items (e.g. march-1, march-2, ..., march-n), each of them describing a specific algorithm ensuring a specific test coverage.

Code-wise, these libraries are arrays of structures containing the main features of each class, as shown in Fig. 1.

```
struct core_type
{
  char          name[..];
  int           DIM_MEMORY;
  int           DIM_MULTIPLEXER;
  int           NBR_ROW;
  int           NBR_COL;
  int           WORD_LENGHT;
  ...               ...
}

struct algorithm_type
{
  char          bist[..];
  char          name[..];
  char          label[..];
  int           start_cycle_result_at;
  int           start_cycle_result;
  int           start_cycle_input;
  ...               ...
}
```

Fig. 1: Example of *Core* and *Algorithm* structures

Besides libraries in each module there are also functions closely linked to module management.

Regarding the *ATE* dependent module, all the routines devoted to manage ATE firmware commands were developed. Goal of the trial was to rely on the smallest set of ATE hardware options and compensate the consequent limitations with a dedicated software tool. Let's call this "*reducing the cost of test*".

Pure digital ATEs have a basic structure tailored to optimize the development of test flow mainly based on launching of patterns and comparing device signals with expected waveforms. Let's call this "*deterministic scenario*".

On the contrary, the output of BIST for logic core is based on a data stream that depends on the functionality of the logic under test. After a processing step, information encoded in the output data stream allows to re-run the algorithm focusing the BIST coverage on the highlighted failing part and starting the punctual diagnosis phase.

In this process ATE needs to deal with the BIST pattern which changes depending on BIST result. Let's call this "*not-deterministic scenario*".

As previously said, pure digital ATEs are not usually designed to handle such a not-deterministic pattern. In order to properly run a state of the art BIST it is necessary to enhance this kind of ATEs with the basic concepts of writing and reading *logical values*.

The enhanced capability of an ATE to deal encoded information with the device is a powerful feature that allows the deep investigation of the silicon by means of few low level functions (ATE dependent) implementing two elementary operations: reading and writing from and to DUT.

The structure of this multi-layer test program can be seen as a master function that links the five class modules, Fig.2. Users, calling the master function during test flow, can specify test environment just by means of five input parameter (see section IV below).

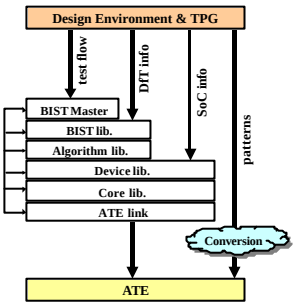

Fig. 2: Multi-layer test program structure for BIST test

Looking at the top level of this structure, the five class modules need also to talk each other because of functions having multi-class dependent arguments. One function might depend on both the core size and the complexity of the algorithm instance.

After the creation of five libraries containing several descriptions of algorithms, ATE platforms, BISTs and DUT information, users can directly insert this package into their test flows and run the BIST test by calling the master function which takes five parameters as input. These *parameters are pointers* to arrays of structure elements. For instance, if users need to change ATE platform or BIST type or March algorithm, the only modification involves the corresponding pointer and, in case the specified library doesn't contain the new

instance, then only the "weak" layer will have to be updated.

IV. Experimental Analysis

In this Section a very simple test chip is proposed as case study allowing evaluating the effectiveness of the depicted technique.

- *Case Study*

The parametric cores analyzed (manufactured using an HCMOS 0.13 µm library) were two SRAM embedded in a simple SoC device:

[core-1] SRAM 4Kx128
[bist-1] hardware BIST (HWB)
and
[core-2] SRAM 8Kx32
[bist-2] MPB with 64x4 RAM-code module for storing test algorithm.

March 12N was hardwired to test core-1 instead March 12N and March 6N algorithms were chosen to test core-2. Total area overhead introduced by the programmable BIST is about 2% of the memory area. In Tab. 1 is reported the total number of gates of the additional logic for test and diagnosis: this value is comparable with the one introduced by a hardwired BIST approach [6] for the same memory type. The TAP Controller and the TAP have not been considered since they are not related to a single core, but shared among multiple cores present in the SOC. Anyway, their size amounts to about 800 gates.

The programmable BIST approach does not introduce any timing overhead and guarantees an at-speed test.

Component	# of gates
Wrapper	1,921
Processor	3,090
Ram module	162
TOTAL	**5,173**

Tab. 1: MPB area overhead

In order to evaluate the effectiveness of the proposed approach the Agilent 93K SOC tester platform was selected.

For the sake of this trial the concerned ATE was considered just as a pure digital system and capture memory option was not considered because writing operations on vector memory and reading operations from fail memory were managed by basically firmware command in ATE layer of our Multi-layer test program architecture.

In practical terms, the high level concept of passing information to the device is built on the capability of ATE to change the parametric value of a certain number of vector cycles inside the pattern starting at a given vector cycle. In the same way the high level concept of retrieving information from the device is built on the ATE capability of recording the parametric value of vector cycles in a given window starting at a certain point of the pattern. Data output from the DUT can be retrieved through the tester error map (a dedicated memory for storing PASS/FAIL information). The vectors for data comparison have to be set to their PASS values (as from a golden device), then any failing cycle will be recorded in this error map and can be post-processed in order to decode DUT BIST report. In Fig. 3, the described diagnostic flow for a parametric memory BIST is visually represented.

```
error_detected = True;
Step = Full;
while (error_detected)
{
    for (i=0 ; i<Step ; i++)
        March(i);
    Read (Status)
    If (Status <> 0)
        {
        read (Err, Result);
        Step = Result.step – 1;
        }
    else
        error_detected = False;
}
```

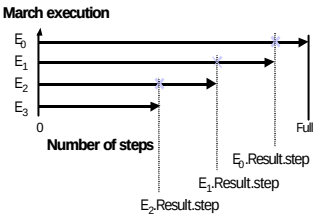

Fig. 3: parameter based re-executions for memory diagnosis

HWB and MPB have different input/output data stream structures concerning data comparing window to get results and input data to re-run the BIST to focus on diagnosis.

The multi-layer approach in designing the BIST algorithm allows dealing with all aspects of this application in a straightforward way.

Top layer (ATE interface layer) of this structure is dedicated to controlling the overall BIST test program main flow and multi-site testing program directives. Main flow algorithm can be described as in figure 4.

Layer 2,...,n-1 are devoted to give out macro functions into basic building blocks down to ATE interface layer.

- **launch_bist_pattern**: set up and run the test, get PASS/FAIL information;
- **check_bist**: verifies that only BIST dependent pins fails;
- **get_result**: reads error map and extract BIST result;
- **collect_data**: elaborates BIST result;

- **new_input_bist**: new BIST input to move BIST focus on failing parts;
- **run_diagnosis**: performs diagnosis;
- **class_device**: bins device.

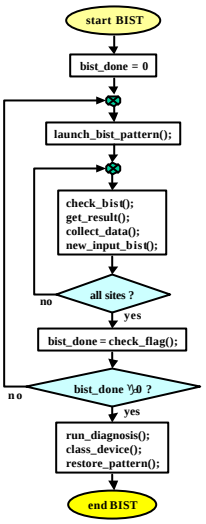

Fig. 4: BIST test flow

Bottom layer includes all functions that are ATE dependent. These functions drive the flow of firmware command to/from the ATE. Main functions are capturing data from the device and write data into the device.

Of course, also the way the BIST procedure is entered through a testing program call and the way the BIST procedure is concluded by sending the binning to the test program are ATE dependent. From a functional concept point of view ATE firmware command are both on the very top level and bottom level of this structure.

- *Results*

Generally speaking, a BIST test, in our multi-layer test program vision, can be described as follow:

BIST_test-n = master_function(device-n, core-n, bist-n, algorithm-n, tester-n).

That means that it's possible to identify a test by a string of five parameters. So in our case study we had three different scenarios:

1. hwb test: **master_function(device-1, core-1, bist-1, alg-1, tester-1).**
2. mpb1 test: **master_function(device-1, core-2, bist-2, alg-2, tester-1).**
3. mpb2 test: **master_function(device-1, core-2, bist-2, alg-1, tester-1).**

Where:
hwb is an hardware BIST (March 12N)
mpb1 is a Micro Prog. BIST (March 6N)
mpb2 is a Micro Prog. BIST (March 12N)

Using an Agilent platform (SOC 93000 C400) equipped with a workstation HP C3750 (2 Gbytes of RAM and processor running at 850 MHz) we tested six lots (EWS). Timing results for good dice are showed in Tab. 2.

Patterns' frequency: 100MHz					
Average Timing Table: BIST test on good dice					
Lot	Good	Go/noGo [ms]		Multi-layer Alg. [ms]	
		hwb	**mpb2**	**hwb**	**mpb2**
1	938	1.614	2.630	2.065	3.097
2	1042	1.630	2.637	2.073	3.096
3	1063	1.655	2.656	2.112	3.096
4	883	1.611	2.633	2.056	3.050
5	894	1.631	2.640	2.073	3.081
6	1138	1.648	2.667	2.085	3.090

Tab. 2: Average timing for BIST test

Notice that previous timings refer to a single run, so these measures highlight that, testing SRAMs by means of our multi-layer algorithm, which includes diagnostic features, leads to a test time overhead of about 0.450 ms with respect to a simple go/nogo test using the same pattern implemented in a traditional way (i.e. just PASS/FAIL without any diagnostic result).

When a memory fails a BIST test, the BIST has to be run several times for collecting failures and then giving diagnostic result. In order to evaluate the average time overhead for each run when a fail is present in the memory the mpb2 test in diagnostic mode was used. Tab. 3 reports some cases analysed.

Patterns' frequency: 100MHz				
MPB Multi-layer Algorithm : Failing dice (mpb2)				
Fail words	runs	Diag. Result	Test Time [ms]	
			Data collect.	Diagnosis
8	8	spot	34.488	0.413
14	14	spot	60.256	0.487
14,336	20	cluster	86.040	0.492
10,752	20	column	85.960	0.498

Tab. 3: Test time for diagnosis by Multi-layer algorithm

From the previous table it is possible to derive that the number of runs depends from number of fails (upper limit was fixed to 20) and the average test time is about 4.3 ms/run using Multi-layer algorithm while by means of a traditional go/nogo test using the same algorithm (March 12N) the value is about 2.6 ms/run.

Practically using our Multi-layer algorithm the time overhead is about 1.7ms/run (1.2 ms/run due for storing fails during data collection phase), instead the time for diagnosis is negligible and of course fail-independent: about 0.5 ms.

V. Next developments and Conclusions

The target of this work is to improve the link between CAD design simulations and ATE development environment. Efforts need to be focused on automatism.

The set of parameters that describes the device and silicon functionality find a common language description in CTL (Core-Test-Language: IEEE 1450.6 standard) [7]. CTL is a standard format to describe intellectual property (IP) core and SoC test information. Moreover, another standard language (IEEE 1450.4) [2] contains the definition of the data blocks necessary to specify the sequence of activities that are to be performed on each device in order to "test" it. ATE environment should take advantage from these descriptions in trying to customize the testing procedures on which the testing program is based. On ATE side, to enable this process, efforts need to be focused on IEEE 1450.x standards reader tools. Referring to multi-layer approach, if a CTL description of parametric core is available, CTL reader should provide parameters' values for updating libraries and 1450.4-reader tool would define procedures being in each layer. In this way the EDA-ATE link would be really automated. Design of so–called "structural testers" is based on these concepts; so, on this kind of equipments, test programs development should be ATE independent and the ATE-link module in our multi-layer architecture would be useless.

In this paper we proposed a simple multi-layer algorithm structure that, paying the price of 1.7 ms/run test time overhead (worst case), can diagnose topological memory fails (few runs needed) and, most important, it was built in a DUT-independent and ATE-independent way to be compliant with new IEEE 1450.x standards in order to automate the EDA-ATE link.

VI. References

[1] D. Appello, P. Bernardi, M. Grosso, M. Rebaudengo, M. Sonza Reorda, V. Tancorre, "STAT: a tool for supporting the test of complex SoCs", Proposed to IEEE International Test Conference ITC 2005

[2] IEEE 1450 Standard Test Interface Language(STIL),http://grouper.ieee.org/groups/1450/

[3] D. Appello, P. Bernardi, M. Rebaudengo, M. Sonza Reorda, V. Tancorre, "On the diagnosis of embedded memory cores through Programmable BIST", IEEE International Workshop on Test Resource Partitioning, 2004

[4] Draft standard for Embedded Core Test IEEE P1500/D0.7, IEEE, 2003

[5] D. Appello, F. Corno, M. Giovinetto, M. Rebaudengo, M. Sonza Reorda, "A P1500 compliant architecture for BIST-based Diagnosis of embedded RAMs", IEEE Asian Test Symposium, 2001, pp. 97–102

[6] D. Appello, A. Fudoli, F. Corno, M. Rebaudengo, M. Sonza Reorda, V. Tancorre, "A BIST-based Solution for the Diagnosis of Embedded Memories Adopting Images Processing Techniques", IEEE International On-Line Testing Workshop, 2002, pp.112-116

[7] R. Kapur, "CTL For Test Information Of Digital ICs", Kluwer Academic Publishers, USA, 2003

Biography

Paolo Bernardi received the M.S. degree in Computer Science Engineering from the Politecnico di Torino in 2002. He is currentlya Phd student in Computer Science and Automation. His main interests includes SoC testing & diagnosis, fault tolerant systems and programmable logics reliability evaluation.

Adriano Bertuzzi has been working as testing engineer mainly on mixed signal Ics and DFT activities for VLSI products. Present activity is focused on product engineering.

Michelangelo Grosso is currently a PhD student in Computer Science and Automation at the Politecnico di Torino, where he received the electronical engineering degree (*summa cum laude*) in 2004. His research interests include testing and diagnosis of electronic systems, in particular DfT design and software tools development for SoCs test and debug.

Vincenzo Tancorre is a design-to-test engineer in the Testing Technology Center at STMicroelectronics. His research interests include test-related process monitoring using diagnostic solutions for memories and unstructured logic. Tancorre has a BS in electrical engineering from Politecnico di Bari (Italy).

Simondavide Tritto graduated in Electronic Engineering at the University of Pisa in 1995, is working at Agilent Technologies as an Application Engineer on 93000 SOC tester since 2001, mainly focusing on DfT techniques, RF and High-speed digital devices