A Tool for SoC Test Network Design Automation

Aram, Khzarjyan
Virage Logic
Yerevan, Armenia
e-mail: Aram.Khzarjyan@viragelogic.com

ABSTRACT
Embedded memory size for SoC has to be increased to satisfy the memory-hungry applications. In the aspect of manufacturing yield, embedded memories have enormous influence on the SoC. For improving yield the embedded memory test and repair is necessary. The built-in test and repair solution is preferable because it has many advantages and does not require expensive ATE. Each type of embedded memory requires a specific test and repair solution, which is easy to realize as dedicated engine. On of possible ways to manage those from a unified point of view are to merge them in one test network. The automation of test network generation, insertion and verification becomes very important.

This paper introduces a tool, called Integrated Tool Environment (ITE) for SoC Test Network Design, a test network example generated by the environment and experimental results.

Keywords
Embedded memory, SoC, BIST, BIRA, BISR, ATE, ITE.

1. INTRODUCTION
Very deep sub-micron technology makes it possible to increase the transistor count on a die and allows packing more complex circuits in one chip. A system is composed from several components that were placed in different chips before. It is also possible to place in one chip called SoC . The main advantages of using SoC are miniaturization of the device size; cheapening product cost, reduced Time to Market. SoC is used in various well-known products such as cell phone, digital multimedia players, game console and other consumer electronic devices . SoC generally contains various cores that could be designed by different IP vendors. Cores can be customized and reused in many SoC designs . SoC computational power and embedded memory size has to be increased to be enough for memory-hungry applications. As a result, embedded memory becomes the major component of SoC that will occupy more than 94% SoC area in the year 2014 . In the aspect of manufacturing yield, embedded memories are more inclined to defects than other SoC components. For improving the yield, the embedded memories should be armed with redundancy . Anyway, memory cannot test and repair itself.

Build in test and repair solution is preferable, because it does not require expensive ATE, can perform at speed testing, offers vertical testability at all levels of testing started wafer manufacturing ended system-level in the field, has high diagnostic resolution. In general, SoC obtains the BIST that is used to perform only testing. For performing repair of embedded memories BIRA and BISR components of the engine are necessary. BIRA allocates redundancy based on detected defects of embedded memories , while BISR is used to reconfigure memories for further repair . In the In the considered example BIST, BIRA and BISR are realized as a processor, called Memory Test and Repair Processor (MTRP).

It is generally preferable to use one MTRP for a group of embedded memories, because having one MTRP per each memory leads to overhead and at having one MTRP for all memories it is too hard to specify all embedded memories properties . The design of MTRP, that is supposed to support different type embedded memory interfaces, is a complex task. To simplify this task, each embedded memory is placed into an appropriate wrapper which has a standard interface to connect with MTRP . Embedded memories can be grouped by different parameters, such as type, power domain, clock domain, placement location, etc. To manage all MTRPs, included in SoC, they are connected in one network, called Test Network (TN). Besides MTRPs, TN also contains Non-Volatile Storage (NVS) and Central Test Processor (CTP) . The main role in TN lays on the CTP that has dual-purpose functionality (manufacturing and common usage). During the manufacturing phase, CTP runs MTRPs, collects repair information, compresses and saves it into the NVS. In the real life, CTP repairs embedded memories when power goes on .

Each component in TN can be adopted for different SoC designs. Manually customization is a hard work. Instead of this each component can be generated by special compilers, called Template Based RTL Compiler . RTL compilers form hierarchical infrastructure and have a great role in the TN generating during SoC design. TN infrastructure usage flow is usually defined on a general sample as a design flow . The general scheme of the design flow for the test network is presented in Figure 1.

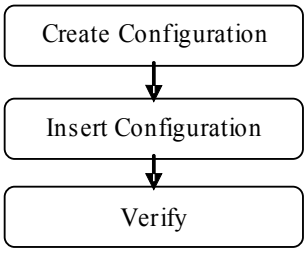

Figure 1: TN infrastructure design flow

Let us consider the flow in details:

- Create a Configuration.

Determine, the set of embedded memories for a particular SoC design, group them by several criteria. Generate embedded memories and related TN with its components. Place appropriate memory instances into SoC design.

- Insert into SoC design TN components. Insert memory wrappers, insert MTRPs and connect them with wrappers, insert CTP, NVS and connect them with MTPRs, connect JTAG interface with CTP .
- Verification has to check SoC functionality invariance under TN insertion, TN inner connections and its functionality.

TN designing, inserting it into SoC and testing is a complex task, which requires RTL Compiler hierarchy and software tool infrastructures. Each of them has its own environment, interface and parameter set. For entire infrastructure its organization complexity grows exponentially. The solution is an automation of design flows.

In Figure 2 a TN example is adduced which contains 6 embedded memories, 3 MTRPs, CTP and NVS. All memories have wrappers around. First MTRP connect with one wrapper, second with two wrappers and third with three wrappers.

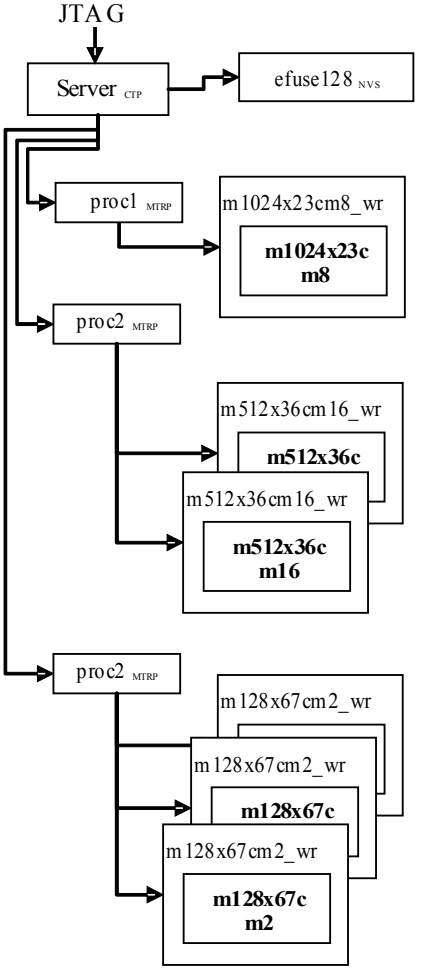

Figure 2: TN infrastructure

2. Related Works

Many results have been obtained in the area of design automation. Let us consider several of them.

In VPerl is introduced for verilog code generation. It is shown that VPerl reduces the Verilog code up to 5 times. However, VPerl is intended only for the verilog and is not extendable.

Another idea of automation tool is proposed for an Application Specific Instruction Set Processor (ASIPs) . The tool basis is LISA that is an Architect Description Language (ADL). It has an ability to quickly explore the ASIP optimum solution. However, it is designed to be used in a narrow sphere and it is very difficult to adopt for other type of tasks.

Another tool, called HDLGen , generates pipelined processors from ADL EXRESSION in functional level and supports various types of architectures (DSP, VLIW, EPIC, superscalar). EXPRESSION is a rich ADL, but does not allow describing TN components such as MTRP, CPT.

However, the main interest was in specific components and architectural types, while the peculiarities of each component were not considered in general. No tool has been developed by now for the TN infrastructure automation to cover every aspect of TN design.

The paper proposes an automated tool environment that can be used for a flexible TN design. It can easily be used for designs having different complexity (from very simple designs up to large and complex designs).

3. TN DESIGN AUTOMATION

The TN design is an iterative process, which can be repeated many times, because the SoC design is an incremental process and it needs to be redesigned due to the improvement necessity of different properties, such as embedded memory size, quantity, grouping criteria, etc. (Figure 3).

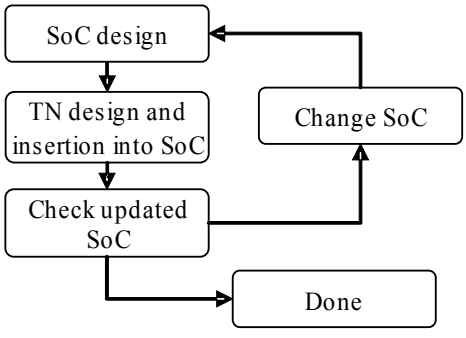

Figure 3: Life-cycle of TN design

To avoid extra work and possible mistakes, the automation of TN generation, insertion and verification becomes highly prior for all SoC designs. The automation is difficult to realize because the environments, which are used for TN infrastructure, are very different. There are two possible ways of automation. The first is to automate each particular design. The second is to define the general environment that can be used for all cases. The second solution gives more generic solution and is more preferable. This paper presents a tool that allows implementing the second solution.

4. INTEGRATED TOOL ENVIRONMENT

ITE is a template based system in which an appropriate template defines the corresponding TN component or tool, its relation with others, etc. It means that ITE has a library of predefined templates. On the other hand, ITE gives an opportunity to add new templates into library. New templates

can be created based either on the existing one or on a completely new one. Each component template has to define the component key features such as Power Management, Repairability, At Speed testing, Diagnostic, Technology Process, Low Area, etc. It forms user input data for ITE based on the key features.

Key component types used in ITE are the following:

Component type	*Description*
Memory	Embedded memory
Wrapper	Memory wrapper
Mtrp	MTRP
Ctp	CTP

Table 1: ITE key components

Let us introduce an example of embedded memory definition in Figure 4. In memory definition after keyword memory follows the memory name, the next in curly brackets declared memory parameters. The memory parameters are Number of Bits in memory word, Number of Words in memory and Column MUX size in memory.

```
memory <memory_name> {

        NB <number_of_bits>;
        NW <number_of_words>;
        CM <column_mux>;

}
```

Figure 4: Memory description

Another type of templates is defined to describe usage flows of TN components. There are several well-known flows of TN design, such as Configuration selection, verification, insertion; System (subsystem) verification, Design for testability, Production, Diagnosis, Third Party Memory, etc .

Depending on the selected flow template ITE suggests the predefined set of component types required for the selected flow. For example, the generation flow assumes the definition of memory, wrapper, mtrp, ctp. In Figure 6 is presented the sample of TN definition. This sample based on diagram placed in Figure 2. There are three memory definitions which have names m1024x23cm8, m512x36cm16 and m128x67cm2. There have Number of Words 1024, 512, 128; Number of Bits 23, 36, 67 and Column MUX 8, 16, 2 accordingly. Then follow three wrappers definitions, where the last one type is a serial. Defined MTRPs have names proc1, proc2 and proc3. The proc1 has one instance of wrapper, the proc2 has two m512x36cm16_wr wrapper instances and the proc3 has three m128x67cm2_wr wrapper instances. In the last defined CTP with name server. The server has NVS with name efuse128 and MTRP instance list. If some component has more than one instance then the component name included into additional curly brackets and with instance count.

ITE is wrapping all particular tool environments used during TN design. As a result ITE suggests one universal environment for all tools. It is important to notice that the presented description of TN infrastructure is easy to define and the probability of making errors is lower then in manual way. ITE gives an opportunity to work in one common environment without having deep knowledge of all necessary tools. As a result, the design time and TN design iteration are decreased.

```
# Memory declarations
memory m1024x23cm8 {
        NW 1024;
        NB 23;
        CM 8;
};
memory m512x36cm16 {
        NW 512;
        NB 36;
        CM 16;
};
memory m128x67cm2 {
        NW 128;
        NB 67;
        CM 2;
};
# Wrapper declarations
wrapper m1024x23cm8_wr {
        memory m1024x23cm8;
};
wrapper m512x36cm16_wr {
        memory m512x36cm16;
};
wrapper m128x67cm2_wr {
        type serial;
        memory m128x67cm2;
};
# MTRP declarations
mtrp proc1 {
        wrapper_list {m1024x23cm8_wr};
};
mtrp proc2 {
        wrapper_list { {m512x36cm16_wr 2} };
};
mtrp proc3 {
        wrapper_list { { m128x67cm2_wr 3} };
};
# Central Test Processor
ctp server {
        nvs efuse128;
        mtrp_list {proc1 proc2 proc3};
};
```

Figure 6: Verification flow sample

5. CONCLUSION

Experiments carried out on several real customers and test projects resulted is the design time reduction for the first iteration approximately from 1 hour to 45 minutes and for the next iterations from 45 minutes to 15 minutes. Mistakes made during the design are reduced dramatically. Generation, Insertion, Verification phases run automatically and the consumed time depends only on the computer performance, there are no human influences and delays.

The most important challenge now is to determine the range of application for the suggested approach. Corresponding results will be reflected in our future publication.

REFERENCES

[1] Yen-Kuang Chen , S. Y. Kung, “Trend and Challenge on System-on-a-Chip Designs”, Journal of Signal Processing Systems, v.53 n.1-2, p.217-229, November 2008

[2] Erik Jan Marinissen , Betty Prince , Doris Keitel-Schulz , Yervant Zorian, “Challenges in Embedded Memory Design and Test”, Proceedings of the conference on

Design, Automation and Test in Europe, p.722-727, March 07-11, 2005
[3] Gabe Moretti , Tom Anderson , Janick Bergeron , Ashish Dixit , Peter Flake , Tim Hopes , Ramesh Narayanaswamy, “Your core— my problem? (panel session): integration and verification of IP”, Proceedings of the 38th conference on Design automation, p.170-171, June 2001, Las Vegas, Nevada, United States [doi>10.1145/378239.378395]
[4] The National Roadmap for Semiconductors, 2000. Semiconductor Industry Association.
[5] Shoukourian, S.; Vardanian, V.; Zorian, Y, “SoC Yield Optimization via an Embedded-Memory Test and Repair Infrastructure”, IEEE Design & Test, v.21 n.3, p.200-207, May 2004 [doi>10.1109/MDT.2004.19]
[6] Rei-Fu Huang , Chao-Hsun Chen , Cheng-Wen Wu, Economic Aspects of Memory Built-in Self-Repair, IEEE Design & Test, v.24 n.2, p.164-172, March 2007
[7] S. Shoukourian , V. Vardanian , Y. Zorian, “An Approach for Evaluation of Redunancy Analysis Algorithms”, Proceedings of the International Workshop on Memory Technology, Design, and Testing (MTDT'01), p.51, August 06-07, 2001s
[8] Yervant Zorian, “Embedding infrastructure IP for SOC yield improvement”, Proceedings of the 39th conference on Design automation, June 10-14, 2002, New Orleans, Louisiana, USA [doi>10.1145/513918.514098]
[9] Yervant Zorian , Samvel Shoukourian, “Embedded-Memory Test and Repair: Infrastructure IP for SoC Yield”, IEEE Design & Test, v.20 n.3, p.58-66, May 2003 [doi>10.1109/MDT.2003.1198687]
[10] Yervant Z. “What is Infrastructure IP?”, IEEE Design &Test of Computers, May–June 2002. – P. 5–7.
[11] Zorian, Y. Torjyan, G. Nenni, D.,"Improving Yield With Retooling and Robust Infrastructure",EVALUATION ENGINEERING -CHICAGO-,Bibliographic details 2008, VOL 47; NUMB 4, pages 50-54, A VERNER NELSON USA, ISSN 0014-3316
[12] Walter Tibboel , Victor Reyes , Martin Klompstra , Dennis Alders, System-level design flow based on a functional reference for HW and SW, Proceedings of the 44th annual conference on Design automation, June 04-08, 2007, San Diego, California [doi>10.1145/1278480.1278488]
[13] Yan XL, Yu LL, Wang JB.,"A front-end automation tool supporting design, verification and reuse of SOC.",Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China.,J Zhejiang Univ Sci. 2004 Sep;5(9):1102-5. ISSN 1009-3095
[14] Oliver Schliebusch , A. Chattopadhyay , R. Leupers , G. Ascheid , H. Meyr , Mario Steinert , Gunnar Braun , Achim Nohl, “RTL Processor Synthesis for Architecture Exploration and Implementation”, Proceedings of the conference on Design, automation and test in Europe, p.30156, February 16-20, 2004
[15] A. Kejariwal, P. Mishra, J. Astrom, and N. Dutt. HDLGen: Architecture Description Language driven HDL Generation for Pipelined Processors. Technical Report CECS Technical Report 03-04, University of California, Irvine, 2003.
[16] Rochit Rajsuman, System-on-a-Chip: Design and Test, Artech House, Inc., Norwood, MA, 2000, ISBN:1580531075
[17] V. Ratford, "Make Your SoC Design a Winner: Select the Right Memory IP," date, pp.0015, 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02), 2002