Use-Flow Diversity and Toolset for Test and Repair of Embedded Memory

Aram Khzarjyan
Synopsys
Yerevan, Armenia
e-mail: aramkh@synopsys.com

ABSTRACT

The development of a modern System-on-Chip (SoC) requires usage of embedded IP blocks from different vendors. The embedded memory, the IP block widely used in SoC, usually occupies an essential die area. Meanwhile, in difference to other SoC components embedded memories are more defect-prone.

STAR Memory System (SMS) is an infrastructural IP solution for built-in test and repair engines of embedded memories. It is widely adopted now by a variety of customers which development flows essentially differ from each other.

To cover the diversity of requests for user maintenance implying from difference in development flows we suggest a new approach basing on a library of SMS standard use flows implemented in a form of templates and a special toolset for their modification and verification . The implemented library of templates assists to design new flows quickly through retrieving and customizing specific examples. User can extend the library via insertion of new templates. A formal verification approach used already for business processes is successfully applied to the built library. The application is illustrated on some use flow examples.

Keywords

SMS, use flow, formal verification, SoC, template library

1. INTRODUCTION

Each new semiconductor technology node provides further miniaturization and higher performance. On the other hand, the growth in demand for System-on-Chips (SoCs) stimulates better, faster, smaller chips. The creation of such SoCs necessitates usage of several embedded IP blocks from different vendors.

Embedded memories IPs become the major component of SoC that will occupy more than 94% SoC area in the year 2014 [1]. In the aspect of manufacturing yield, embedded memories have more influence than other SoC components. Memory with repair ability will improve the SoC yield [2]. Anyway, memory cannot test and repair itself. Build in test and repair one of the solutions which can perform at speed testing, offers vertical testability and high diagnostic resolution. In general, SoC obtains the BIST that is used to perform only testing. BIRA and BISR engine components are necessary for performing repair of embedded memories. BIRA allocates redundancy based on detected defects of embedded memories [2], while BISR is used to reconfigure memories for further repair [3]. In the considered example BIST, BIRA and BISR are realized as a processor, called Memory Test and Repair Processor (MTRP).

STAR Memory System (SMS) is one of well-known MTRP solution [4] which is a complete set of infrastructural IP compilers and supporting software tools. Users are usually using different IP blocks with a wide range of SMS components to build their SMS use flows. It means the user has to learn all SMS components taking into account all specific details of their. As a result, the use flow design can become time consuming and an error prone process. One of use flow design optimization possible ways is the encapsulation of its complexity by providing a standard Use Flow Template Library (UFTL) and toolset of SMS usage.

The paper presents an approach of UFTL construction. Suggested UFTL templates are well-known usage flows of SMS. Five basic use flow templates, forming basic units of more complex use flow templates, are defined per each basic SMS tool. An approach of UFTL formal verification is suggested to verify the correctness of use flow. The illustration of formal verification approach application is provided on one of UFTL templates. The modification of the mentioned flow is a user-driven case. The formal verification has been applied to the flow to check the correctness of the modified use flow template.

2. STAR MEMORY SYSTEM

SMS allows usage of MTRP for a group of embedded memories. 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 [4]. 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 to one network, called Test Network (TN). Besides MTRPs, TN also contains Non-Volatile Storage (NVS) and Central Test Processor (CTP) [4]. 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 is on [4].

Each component in TN can be adopted for different SoC designs. Manual 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 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 and parallelism grows exponentially. The solution is an automation of use flows of TN Design and Verification Process (DVP).

3. ENGINEERING USE FLOWS

SMS Use flow is usually defined as a common design flow which can be implemented through performing the following five steps:

- *Preparation* is assigned to support different IPs with SMS interface.
- *Generation* is assigned to configure and create SMS components with IP blocks based on SoC design requirements.
- *Insertion* is assigned to integrate SMS components into SoC design and to connect signals between different hierarchies of TN blocks.

•*Design Verification* is assigned to check the correctness of SoC functionality, completeness of connections between IPs and test capability of IPs.

•*Post Manufacturing verification* is assigned to identify SoC die possible defects.

Each step can be performed manually using SMS family tools and compilers. All tools and compilers support a wide range of configuration options and workflows. Each SMS usage flow manual customization has many disadvantages such as routine repeating similar operations, extra time consuming, requirement for verifications per each change, etc. The automation of the mentioned process is one of the possible ways to avoid extra work and to save time and efforts.

The corresponding steps are documented, verified and suggested as recommended flows of SMS usage. A general reference flow to combine the mentioned steps for end to end usage of SMS package is suggested as well. The analysis of various users use flows has identified some standard use flows of SMS usage. These use flows are the customization of SMS usage flow templates that are similar to well-known ITIL templates [5-7]. Similarly, SMS use flow templates can form a template library which can be provided to ease the users' work. A repository of process templates assists to design new processes more quickly allowing them to retrieve and customize specific examples. User can also extend the library through insertion of new templates into the provided library.

The analysis of various SMS use flows has identified several features listed below:

- Parallel execution branches are available within one use flow (parallel generation of wrappers, MTRPs, etc.).
- Synchronization points can be required for a use flow (TN generation requires successful generation of all MTRPs, etc.)
- Single execution can be forked by several parallel branches(several post-manufacturing verifications)
- Each node is a function with the corresponding input and output variables
- Use flow contains also a data flow.

The provided features of SMS use flows are similar to the workflow processes features. That is why IBM's MQSeries Workflow model has been selected as a formal model of defining SMS use flows [8]. The main model components are ***activities*** and ***connectors***. The activities are associated with a context being defined as data passing to an activity. It is called an ***input container***. An activity also returns data called an ***output container***. Some output container elements of an activity can be passed to the input container elements of other activities or to the ***external memory***. All data elements are collected in the set **V.** Control and data connectors provide connections between the activities. A control connector has an associated Boolean predicate called a ***transition condition***. A directed graph based on sets of activities and control connectors is called a ***control flow*** of a workflow process. Full details can be found in [9, 10].

4. USE FLOW TEMPLATE LIBRARY

Taking into account the specifics of each recommended and customer-driven use flows, the paper presents an approach of SMS Use Flow Template library with the possibility to modify and extend the existing ones by the usage of provided UFTL supporting toolset. Any formal verification algorithm of workflow processes can be selected as a verification algorithm. Assertions have to be defined to check the correctness of input data and the correctness of execution per use flow.

There are five basic use flows used as a basis for any use flow template construction: preparation, generation, insertion, design and post manufacturing verification.

The preparation of different IPs to support SMS standard interface can be performed by interface preparation tool whose template is presented in Figure 1.

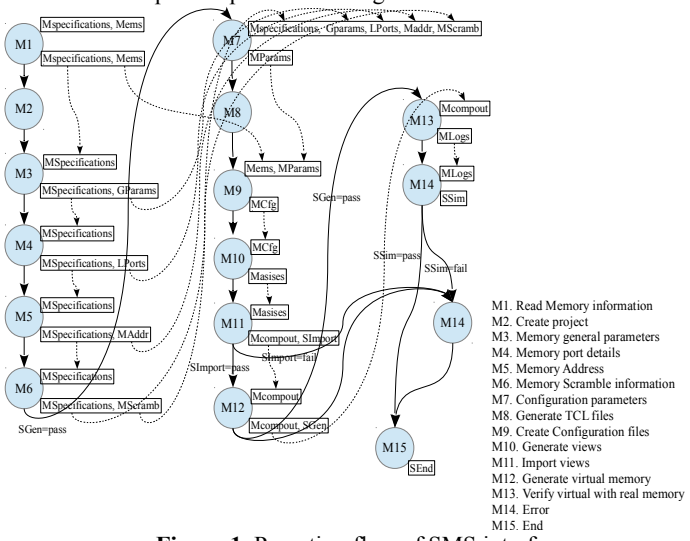

Figure 1. Prepation flow of SMS interface

The generation of SMS components can be performed by generation tool whose template is presented in Figure 2.

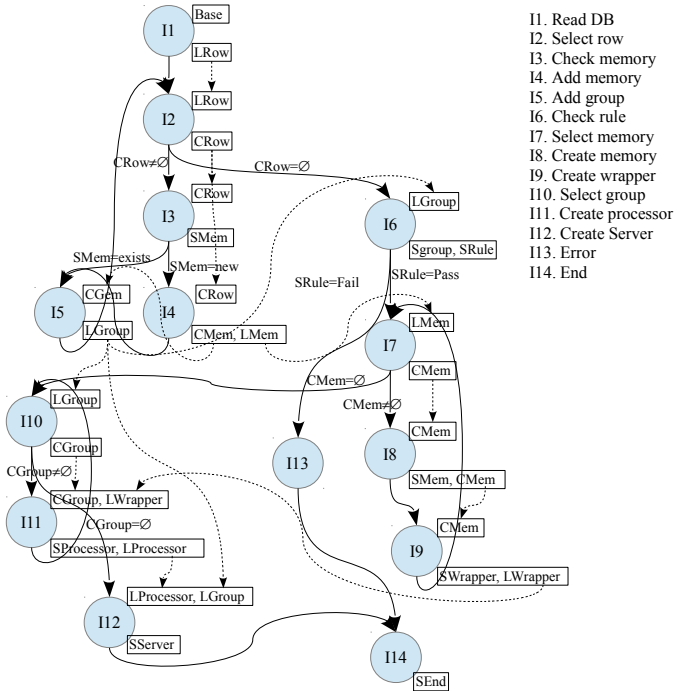

Figure 2. Generation flow of SMS components

The insertion of SMS components into design can be performed by insertion tool whose template is presented in Figure 3. The design verification of SMS functionality and completeness can be performed by verification tool whose template is presented in Figure 4. The post manufacturing verification generates and applies test patterns on die and processes log data files by using appropriate tool whose template is presented in Figure 5.

The mentioned steps can be combined depending on customer requirements. Each of them can be used as a separate use flow as well as can be a part of general use flow. The recommended flow of SMS complete solution use flow is a combination of all steps within a use flow illustrated in Figure 6.

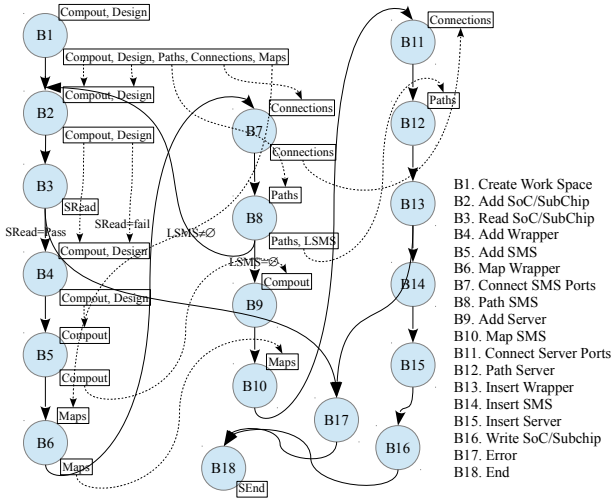

Figure 3. Insertion flow of SMS integration into SoC

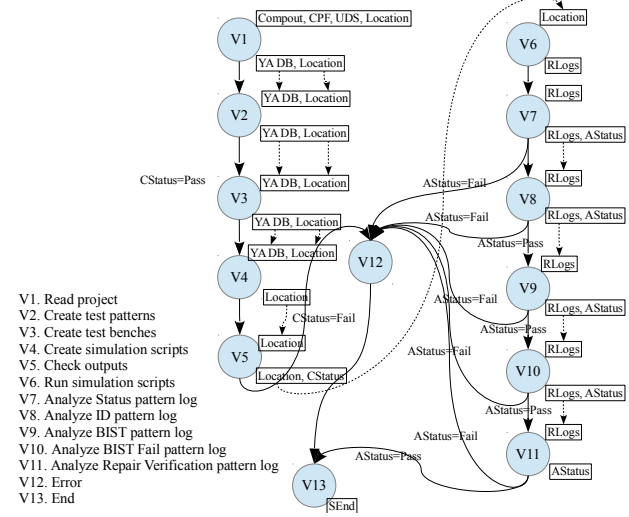

Figure 4. Verification flow of SMS functionality

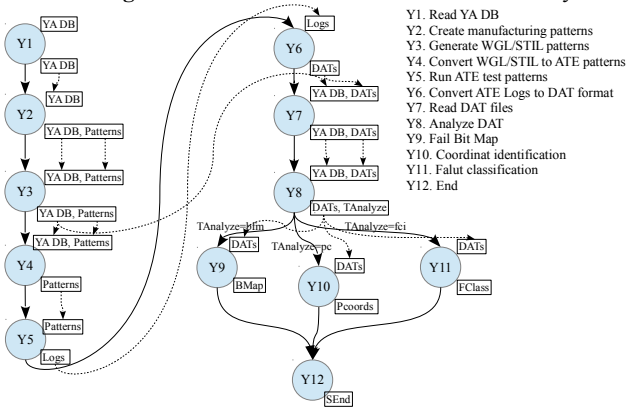

Figure 5. Post manufacturing flow of verification

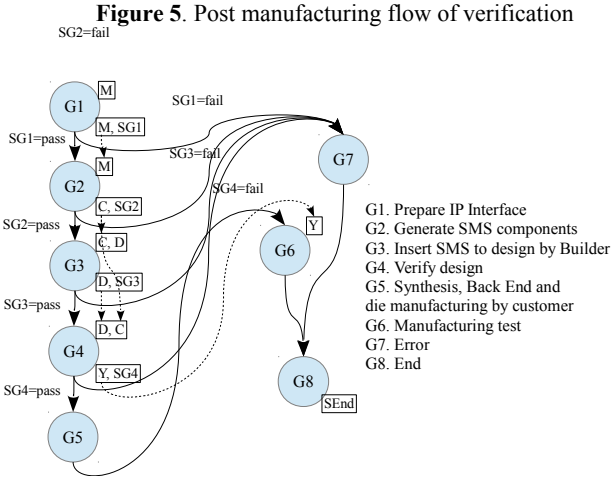

Figure 6. General flow of SMS use

Note: The suggested flow is a recommended flow only. User can modify it adding new activities or deleting the existing ones. Variations of the suggested compete use flow make 32 (5 steps, each step has 2 statuses – exists, removed).

5. FORMAL VERIFICATION ALGORITHM

The formal verification requires knowledge about the underlying workflow process (internal structure of tasks, data flow, etc.). The formal verification has to take into account not only peculiarities of process structures, but also data dependencies and, particularly, the data flow graph of a given process [11, 12, 13]. Two assertions have to be specified for the process. First, the process input assertion, called precondition, has to be satisfied prior to the process execution. Second, the process output assertion, called postcondition, has to be checked at the end of each process execution. The process is considered to be correct if the value of postcondition is "true" for all possible executions. This paper is using an extension of the acyclic process verification algorithm [11] for verification of cyclic workflow processes. It is based on the idea of reducing the cyclic workflow process to the acyclic one, and then applying the formal verification approach of acyclic processes. The algorithm of the described approach, suggested for the verification of cyclic business processes, is presented in details in [9]. Figure 2 is presenting the generation use flow template that is the most useful by users. For the verification of the given process, precondition and postcondition should be specified for the process [9, 11]. The specific conditions are created based on the needs of verification against definite aspects of process behavior. Generation flow cannot be executed without DB. Precondition has to be specified to check that DB information is applicable. It means that I1 activity input variables have to be defined.

$PreC = i(I1).Base \neq \perp,$

where $\perp$ denotes the unknown value of the variable. Generation process has to be finalized in case of either successful generation or error. The TRUE value of I13 activity Serror output variable means that the process has a generation error in some step. The PASS value of I14 activity Sserver output variable means that all components are generated successfully. The postcondition has to check overall behavior of the process for both cases.

PostC = (Serror = TRUE OR Sserver = PASS) AND Send = TRUE

The application of formal verification algorithm on the described process will identify the presence of cycles in it. The detailed analysis of cycles will determine that process cycles are intervals. The first step of algorithm will reduce the cyclic graph to the acyclic one [12]. It will initially construct the set of first order intervals - S_C={<*I2, I3, I4, I5*>, <*I7, I8, I9*>, <*I10, I11*>. The next step is an replacement of intervals by the corresponding equivalent activities <*I2', I4', I5'*> [9]. Exit transitions of new activities have to contain branching information the corresponding cycle that is interpreted in terms of branching state registers. They are cycle invariants. For instance, *I2'* activity has a transition to *I3'*. Its transition condition is formulated from transition condition of *I2* to *I6* with addition of branching register BrSr. *BrSr = 1* is presenting execution path <*I1, I2, i6,…*> and *BrSr = 2* is presenting execution path <*I1, I2, I3,…*>. Transition conditions of other new activities are constructed similarly. Reduction of the mentioned intervals by equivalent activities will result in a new process (Figure **7**). Second phase analysis of the graph will determine that it is acyclic. The acyclic process verification algorithm [11] has

to be applied to the reduced process. After the execution of the verification algorithm and after checking the correctness condition, we get that the process is *correct*.

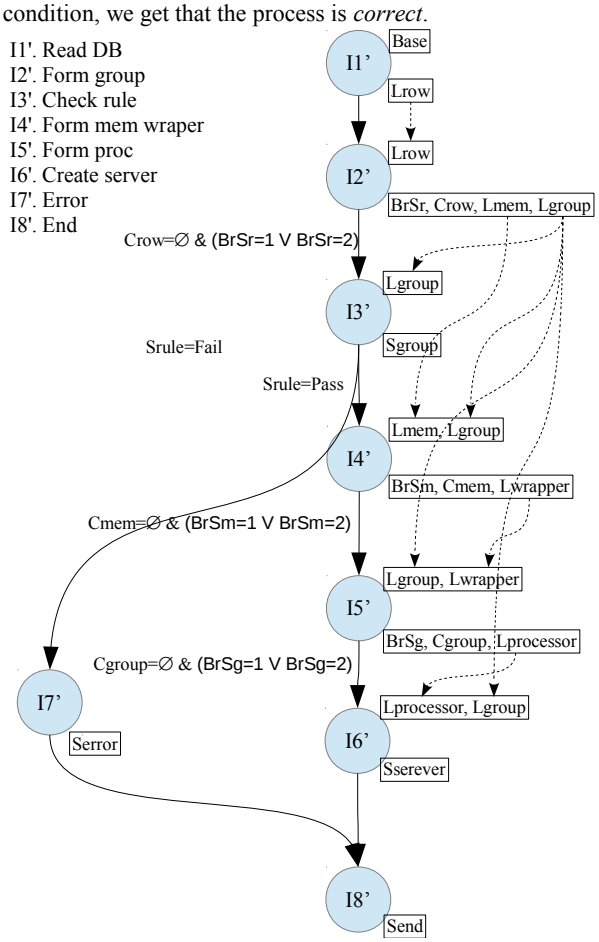

Figure 7. Reduced process

Some of the presented activities can be modified by the user. They *are I4', I5', I6', I8'* activities.

To improve the overall quality of this process *Analyze* and *Notify* additional tasks are added to the process Figure 8 for one of our customers. This activity analyzes if everything execution of a process. In case of an abnormally executed process, the activity *Notify* would notify about it. In addition, the postcondition has to check the state variables of analysis activity to be sure that analysis procedure passed successfully.

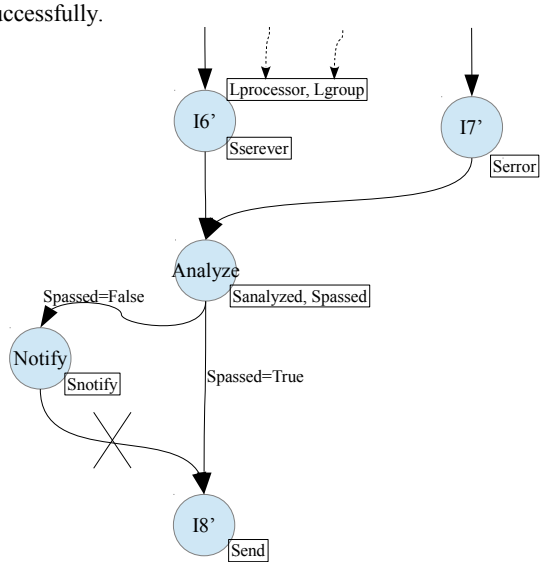

Figure 8. Modified section of the process

New postcondition:

PostC = (Serror = TRUE OR Sserver = PASS) AND Sanalyzed=TRUE AND Spassed=TRUE AND Send = TRUE

As a result of applying the similar steps of the process transformation and verification, the condition of **incorrect** processes would be satisfied [11]. The control connector between activities *Notify* and *End* has to be removed to correct the modified template logic. The postcondition has to be also changed to:

PostC = (Serror = TRUE OR Sserver = TRUE) AND Sanalyzed =TRUE AND ((Spassed =TRUE AND Send = TRUE) OR (Spassed =FALSE AND Snotify = TRUE)).

Applying the algorithm to the corrected process will result in the satisfaction of the correct process condition [10].

6. CONCLUSION

The use flow template library (UFTL) is suggested that contains SMS standard use flows. Specific use flows can be designed by customer through the modification of suggested templates by the toolset. A formal verification algorithm of workflow processes has been suggested to verify the correctness of customers' use flows after the modification. The application of the presented approach is illustrated on the SMS generation use flow template. It is also planned to extend UFTL for other type of IP cores in the future.

7. REFERENCES

[1] The National Roadmap for Semiconductors, 2000.

[2] Rei-Fu Huang; Chen Chao-Hsun; Wu Cheng-Wen; , "Economic Aspects of Memory Built-in Self-Repair ", 2007, Volume: 24, Page(s): 164 - 172.

[3] Shoukourian, S.; Vardanian, V.; Zorian, Y., "An approach for evaluation of redundancy analysis algorithms ", Memory Technology, Design and Testing, IEEE International Workshop on, 2001. , Page(s): 51 - 55 , 2001.

[4] Shoukourian, S.; Vardanian, V.; Zorian, Y., "SoC yield optimization via an embedded-memory test and repair infrastructure ", Design & Test of Computers, IEEE, 2004, Volume: 21, Page(s): 200 - 207.

[5] IT Infrastructure Library, IT Service Management, Office of Government Commerce, http://www.itil.co.uk/.

[6] , Microsoft Corporation "The Infrastructure Optimization Journey", 2008

[7] IBM, "Introducing the IBM Process Reference Model for IT" , January 2007, Second Edition.

[8] IBM 2001. "IBM MQSeries Workflow: Concepts and Architecture", Version 3.3, GH12-6285-03, Product No. 5697-FM3, Mar. 2001, pp. iii-58.

[9] Kostanyan, A.; V. Matevosyan; S. Shoukourian; A. Varosyan, "An Approach for Formal Verification of Business Processes". BIS'09, SpringSim'09, San Diego, USA, 2009

[10] Leymann, F. and D. Roller. 2000. "Production Workflow: Concepts and Techniques", Prentice Hall Press, 2000

[11] Kostanyan, A., Varosyan, A., 2008, "Partial Recognizing Algorithm for Verification of Workflow Processes", FUBUTEC'2008, Porto, Portugal, pp. 89-94

[12] Allen,F.E., Cocke, J."A program dataflow analysis procedure", Communications of the ACM, March 1976, 19(3):137-147.

[13] P. Ammann and J. Offutt. "Introduction to Software Testing. Cambridge University Press", 2008. ISBN 978-0-521-88038-1