IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 10, Issue 2, Ver. IV (Mar - Apr.2015), PP 15-21
www.iosrjournals.org

A Generic Randomization Framework Architecture for Test Execution in Automated Testing Of SoC

Subinoy Das[1], Surendra Shamanna[2]
[1](*IFIN ATV PSV, Infineon Technologies India Pvt. Ltd., Bangalore, India*)
[2](*IFIN ATV PSV, Infineon Technologies India Pvt. Ltd., Bangalore, India)*

Abstract : *Conventional directed tests lack the flexibility to cover different variations of test configurations. To overcome the problem and obtain better coverage, randomization is adopted. Generic randomization framework for test execution in automated testing of SoC is effective for generating real time stimulus that covers variations in test configurations such as different functional modes, register configurations, and network data packets. It provides a way to capture the state of the DUT (Device Under Test) at randomly generated parameter configurations. It prevents the selection bias and also accidental bias. It eradicates the origin of bias and provides better bug fixation in automated testing of SoC in post-silicon validation.*

Keywords: *Automated Validation Execution Flow, Post Silicon Validation, Randomization Framework, Test Automation*

I. Introduction

Due to high complexity of modern designs and increasing pressure to reduce their time-to-market, bugs can escape the pre silicon verification environment. Therefore in order to check for extreme cases and escaped bugs post silicon environment is used effectively. In order to achieve the "zero defect SoC" (System on Chip), it is required to check whether all the possible configurations/features of the SoC are working as expected or not. This involves testing of SoC using huge number of tests manually and consumes a lot of time, and is error prone as well. Thus, automation reduces the process time for the validation of SoC. This can be done by using an automation tool to execute the test flows and writing a suitable process for controlling the test execution on the DUT (Device Under Test). Randomization framework forms a part of the automated SoC validation process. The randomization automation framework can configure test parameters and pass them to the DUT, and generate status report.

Generic randomization framework provides a way to capture the state of the DUT. It provides randomly generated test configurations for test execution. Absence of bias means more reliable tests for automated SoC validation process. Ultimate goal of randomization is to ensure that each configuration of parameters is equally likely to be assigned to the test execution, so that extreme cases are checked and better coverage is obtained.

II. Scope Of Work

Generic randomized test framework architecture automates the generation of random test configurations for test execution. It provides a way for passing execution parameters to the DUT and save the recorded CPU state build up. This saved state buildup of the DUT is used to bring it back to the last known state just before failure and then execute the smaller set of execution parameters to the point of failure, thus requiring lesser execution time. Randomization framework enables the execution of smaller set of configurations more frequently and requires shorter reproduction time. Generic randomization framework will generate a random set of reproducible parameters and pass it to the test executable. After accepting the parameters of the framework, the test will carry out its operations for validation of SoC. Generic randomization framework is solely responsible for the passing of random reproducible parameter values to the test executable at run time.

III. Methodology

Randomization framework consists of designing a process that can be run through the automation tool and it will start execution of the test on the DUT. "Fig. 1" shows the basic block diagram of the randomization framework. "Fig 4" shows the features/functions in the randomization framework.

Selection of parameters will be random in nature and constant for a particular seed number and seed type, so that the effect is reproducible and repeatable. The set of selected parameters/test configurations will be passed to the DUT, and then the DUT will be reconfigured repeatedly with/without performing any reset for the next iteration cycles based on the number of seed iterations.

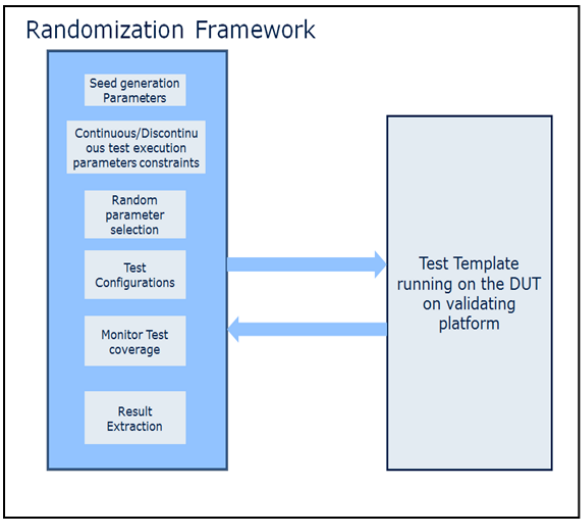


Figure 1. Randomization framework block diagram

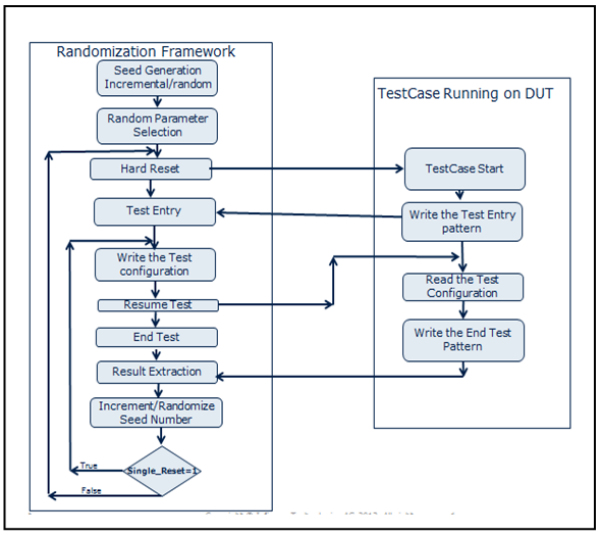
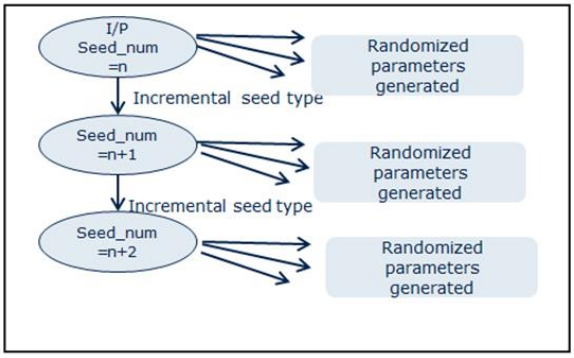


Figure 4. Functional flow for the randomization framework

Random value generation- Generation of randomization parameters includes obtaining the random numbers through a random number generator based on a specific seed number [Input Value] ,so that the event is reproducible. The randomly generated parameter values are then constrained by user-defined continuous/discontinuous limits. And selected parameter values are passed to the test execution. This technique maintains complete randomness of the assignment of parameters to a particular test executable configuration. Seed number forms the nodes for the chain of random parameters generated. If seed number and seed type are known at any node point, the set of events can be reproduced again.



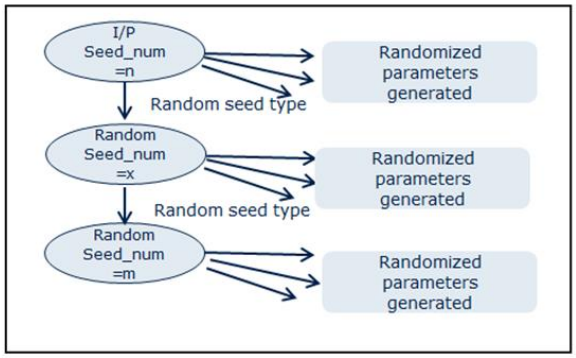


Figure 2. Random number generation using incremental seed type

Figure 3. Random number generation using random seed type

Incremental / Random Seed Type - Incremental seed number introduces a single level of randomization. In this, seed number is incremental in nature. Based on the seed number, the random parameters are selected from user-defined parameter constraints forming single level of randomization. "Fig 2" shows the generation of random parameters using incremental seed type. Random seed number introduces two levels of randomization. In the first level, seed number is randomized and, in the second level depending on seed number the random parameters are selected from user-defined constraints. "Fig 3" shows the generation of random parameters using random seed type.

DUT Reset once / Reset on every test configuration - Reset once /Reset on every test configuration can be controlled by the "single reset" flag option. If "single reset" flag is set, then DUT will reset only once, the next test configurations will be passed without resetting the device. Resetting only once ensures CPU state buildup. If 'single reset' flag is not set, then DUT will reset on every iteration before passing the test configurations. Resetting on every iteration ensures the recovery of the DUT from failure, before passing test configurations.

Synchronization between Test execution and Automation flow - For maintaining synchronization between test execution and automation flow, shared addresses of DUT are used. Using automation tool's built-in functions for memory reading and writing, control words can be used for establishing a semaphore mechanism

to pass configurations from automation flow to the test execution, and get results back from test execution to the automation flow.

Test considered - For validating the generic randomization framework, a use-case of SENT (Single Edge Nibble Transmission) Normal Frame is considered whose intent is to check for all possible frequencies and data values for the SENT Normal Frame Format by configuring the GTM (Generic Timer Module) for generating these SENT frames. "Fig. 5" shows the automation tool's GUI (Graphical User Interface) for the randomization framework, test parameters can be entered in continuous/discontinuous parameter format. Continuous/Discontinuous parameters can be specified in [min-max] /[value1, value2, value3…] format.

Table 1 shows the "Randomization framework field values" which is same for all tests. Table 2 shows the "Test execution parameters" which are specific to the SENT Normal test.

Considered Test's Flow -

Step [1]. Begin.
Step [2]. Initialization of GTM.
Step [3]. Test accepts selected parameters and test iteration from the Randomization framework through shared addresses, using semaphore mechanism.
Step [4]. Then test performs frame and CRC (Cyclic Redundancy Check) calculations, and loads data into the FIFO (First In First Out). Then GTM and SENT are configured to send the data.
Step [5]. It waits for 500 interrupts, for the data to be received in the SENT RD (SENT Receive Data) register.
Step [6]. After that counter and ATOM (Advanced Routing Unit connected Timer Output Module) channel is disabled, compare and shadow registers are cleared.
Step [7]. Then, based on the test iteration which is being passed by the Automation flow process, step[3]-step-[6] is performed again in an iterative way.
Step [8]. Exit.

Table 1: Randomization Framework Fields

Field Name	*Field Description*
Config file	Configuration file with respect to the DUT
Hex file	Binary file to be executed on DUT
MC boot config file	Boot configuration file for the MC(Management Control)flow
TimeOut	Timeout time for test execution
Program execution by	Selection of core [0/1/2] for downloading test executable.
Is Serial Prints Enabled?	Enable/Disable Serial prints for the result extraction during test execution
Seed Type?	Incremental/Random Seed type for random number generation
Seed Number	Based on this value, random parameter values will be generated for the test
Seed Iteration	Number of seeds used for test execution.
Parameter[1-30]	Test execution parameters.[min-max] /[value1, value2,…..]
Single Reset	DUT Reset once/Reset on every test configuration

Table 2: Sent Normal Frame" Test Execution Parameters

Test Parameter Number	*Test Parameter Name*	*Test Acceptable Range*
1	Data Nibble	0x1-0xf
2	Data Nibble	0x1-0xf
3	Data Nibble	0x1-0xf
4	Data Nibble	0x1-0xf
5	Data Nibble	0x1-0xf
6	Data Nibble	0x1-0xf
7	Clock Ticks	0x1-0x5a
8	Divider mode	0x0-0x1
9	SENT Channel	0x0-0x9
10	SENT ALT. Channel	0x1-0x2

Randomization framework Process flow -

Step [1]. Begin.
Step [2]. Fetch the seed number, seed type, "single reset" and seed iteration for the randomization framework and hex-file, configuration file, MC boot configuration file for the SENT Normal Frame test from the user.
Step [3]. Fetch the test parameters from the framework.
Step [4]. The parameters in [min-max]/[value1,value2,value3….]format is separated and converted to hexadecimal format for transferring it to memory.
Step [5]. If "single reset" field is set, then seed iteration becomes equal to the test iteration, and if this field is not set then test iteration becomes equal to unity.
Step [6]. Downloading of the test executable for SENT Normal Frame test takes place in the SoC, and just after that pre-configurations like initializations for clock, and address definitions are done.
Step [7]. Based on the "single reset" flag set or not in the framework hard reset is being performed for the SoC .If "single reset" flag is set, hard reset is performed only once otherwise reset is performed on every seed iteration.

Step [8]. The test writes a control word [Entry pattern] in the shared memory to indicate beginning of the test execution, and reads number of test iteration which is being passed from the automation tool process through shared memory address.
Step [9]. Test polls for the control word [resume pattern] in the same memory address which is going to be written from the automation flow, to resume the test.
Step [10]. Based on the seed number and seed type, a random parameter value will be selected in the user specified parameter range.
Step [11]. The random values generated, are written to the shared memory addresses based on the number of parameters from the automation process.
Step [12]. After that, the control word for resuming test is written from the automation process .With this, the test resumes and reads all the randomized parameter data values from the shared memory addresses.
Step [13]. The test executes for all the specified parameter configurations from the framework and the data is transmitted from one port to another
Step [14]. The functional coverage of the test can be monitored using prints from the test directly on to automation tool's log file, which is shown in the result section
Step [15]. After the test is executed, the test will write a control word to indicate the end of the of the test execution.
Step [16]. Based on the number of seed iterations and "single reset" flag being set or not, process iterations are performed. If the "single reset" flag is set, the steps 9-15 are repeated. And if the flag is not set, the steps from 7-15 are performed in an iterative way.
Step [17]. Exit.

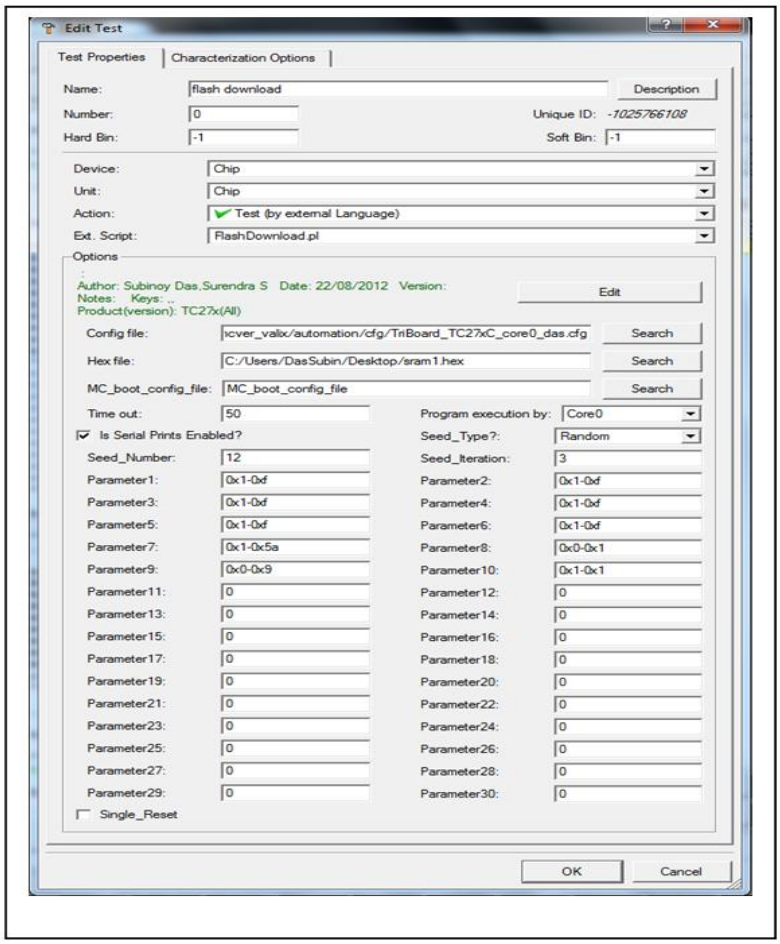


Figure 5: Automation tool's Randomization Framework GUI

IV. Result Analysis

Randomization framework has been successfully implemented and randomized parameters are being selected between the range specified by the user, and are passed from automation tool's GUI to the test execution. See the appendix section for results in more detail.

V. Conclusions

Randomization framework solved the problem of biasing in test execution for validation of SoC. Instead of user selecting any parameter value, it is required to provide the range within which values are required to be generated as per the data specifications. This is better than the standalone framework as it offers more coverage features in validation.

Thus, generic randomization framework for DUT would help to stress SoC components and gives user run control through automation software setup. It also helps to extract coverage and debug information from time to time through execution of test.

VI. Future Scope

The work has a potential to be extended by adding some more features, and its application on other areas as follows:

[1] Cases of interdependent parameters and scenario randomization where different test execution scenarios can be randomized.
[2] Application of the randomization framework in areas of communication between two systems using different randomized communication protocol.
[3] Application of Randomization Framework for Memory testing-where different memory blocks are considered randomly for reading/writing to test different memory accesses.

Acknowledgements

Thanks to Infineon Technologies India Pvt. Ltd., Bangalore for providing an opportunity to carry out this work. Thanks to Sadashivaiah Shivaprasad for his overall management for carrying out this project work. Thanks to Pammi Sesha (IFIN ATV PSV) for his support. Thanks to Mr. Rahul Thati (IFIN ATV PSV) for validation of the framework using SENT test. Our special Thanks to Ramasamy Subramanian for his valuable inputs towards the project work.

References

[1]. D.Ghosh, R.Subramaniam, V.Murthy, "A Randomized Methodology for Post-Silicon Validation of CAN and other Communication Modules", IN: Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on 22-25 Aug. 2013.

Appendix

In the "Fig. 6(a)-(b)" and "Fig. 7(a)-(b)" corresponding markings indicate-

[1] Downloading of the test executable to the DUT.
[2] Seed number and [3] Seed type.
[4] Selection of random parameters based on the seed type –incremental/random, and seed number.
[5] Control word [entry pattern]- written from the test .
[6] Writing the parameters to the shared memory locations.
[7] Control word [resume pattern]-written from the automation process.
[8] Prints from the test on the automation tool's result log file.
[9] Control word [end pattern] –written from the test.
[10] DUT reset once/reset on every iteration.

"Fig 6(a)" and "Fig 6(b)" show the result log obtained while downloading the test executable in flash using single board randomization framework with incremental seed type for three seed iterations and resetting the DUT on every test iteration, to ensure the recovery of the DUT from failure.

"Fig.7(a)" and "Fig.7(b)" shows result log generated in randomization framework for single board randomization framework with random seed type for three seed iterations without resetting the DUT more than once, to ensure a CPU state buildup. Random seed type is different from the incremental seed type, as the seed numbers generated are random in nature depending on their previous seed number.

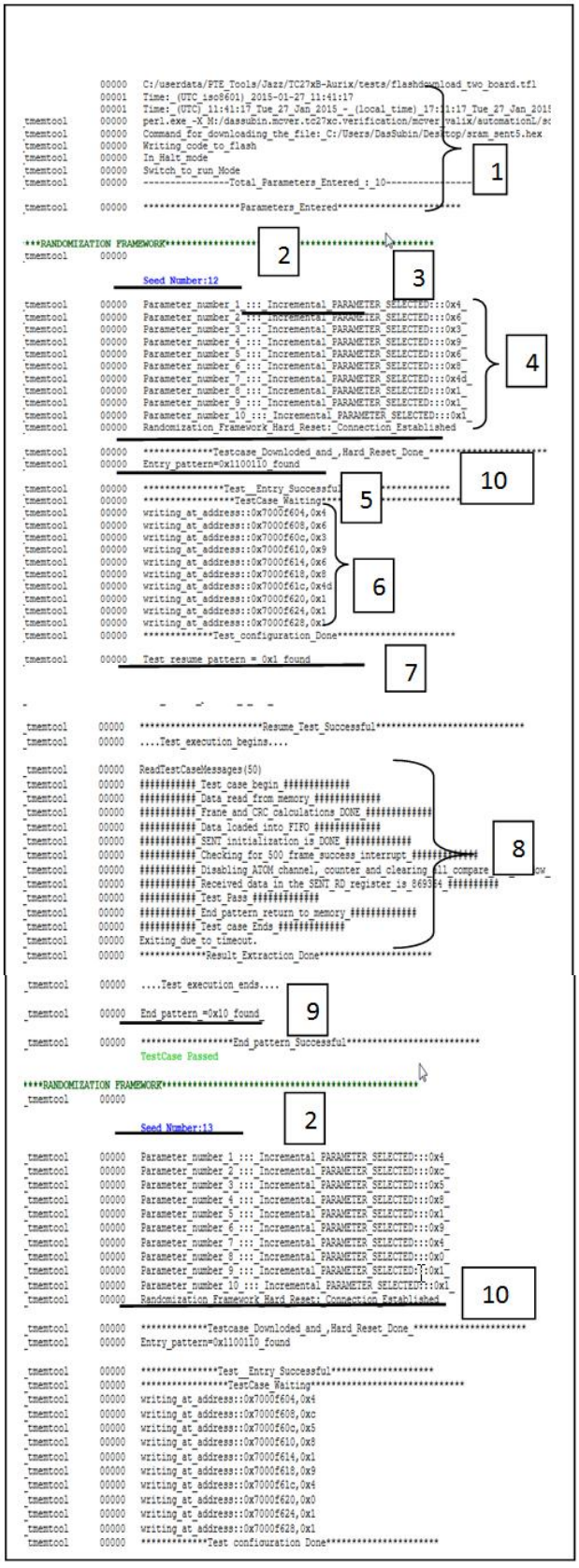


Figure.6 (a) Result log obtained for randomization framework ”SENT Normal Frame Test” using Incremental seed type and “single reset” unset.

Figure.6(b) Result log obtained for randomization framework ”SENT Normal Frame Test” using incremental seed type and “single reset” flag unset.

Figure.7(a) Result log obtained for randomization framework ”SENT Normal Frame Test” using random seed type and “single reset” flag set.

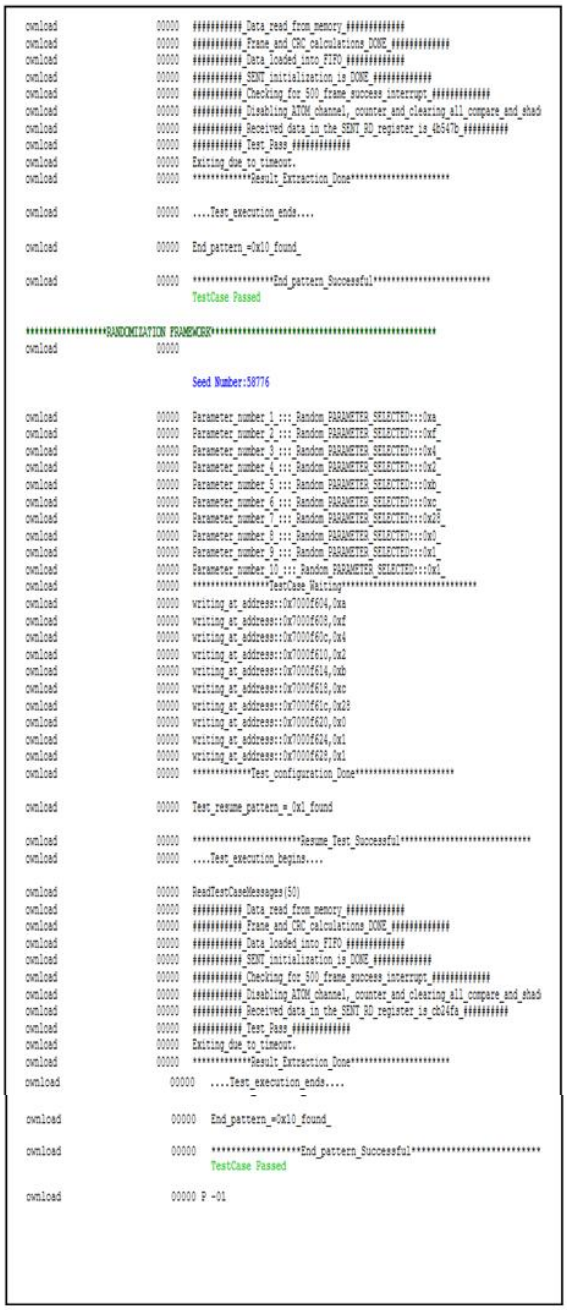


Figure.7(b) Result log obtained for randomization framework ”SENT Normal Frame Test” using random seed type and “single reset” flag set .