

# Randomization Approach for Processor based SoC

Nilesh Patel



## Randomization Approach for Processor based SoC

Randomization has always been one of the effective parts of Verification Environment. It can perform a vital role for verification environment of highly configurable DUT, such as processor based SoC.

The randomization approach for processor based design platform and assembly program based test environment is discussed here.

- **Traditional method of Test Approach:**

Verification of System on Chip incorporated with processor and a large number of peripherals is becoming more time consuming in terms of development of tests with all possible configurations. This is because each peripheral contains tens of configuration registers which turn into a number of different configurations. However more of possible configurations are permutation combination of configuration fields instead of unique configuration of peripheral block.

What should be the value or set of values for a 14 bit clock divisor when going to draft for test-plan is a challenge and to cover all possible values is not a wise decision. Generally the verification team decides to have lower values of clock divisor which will make simulation faster and have one test scenario which contains higher value of clock divisor. This will not allow testing other values of configuration field.

The question arises, how to make sure that a reserved resource in configuration register doesn't affect the functionality? A bug can be induced into design if care of reserved configuration field is not taken into design.

Here randomization helps. Randomization relieves test developer from using hard coded value for configuration space and any assumption for it. However randomization approach requires one time development effort for verification environment. This is because configuration space doesn't allow test developer to configure any random value. Here randomization does not mean any random configuration whether it is valid or invalid.

General flow of the verification of processor based SoC is to configure registers of configuration space of module under test using processor specific assembly language.

The general randomization approach in verilog based verification environment is to drive the input using random data generator system task with input seed. This will limit the verification environment to configure only those registers which are possible to configure using verilog BFM (Bus Function Model). But this will not work for a processor based DUT which require test case in assembly language.

- **Contemporary Randomization Approach:**

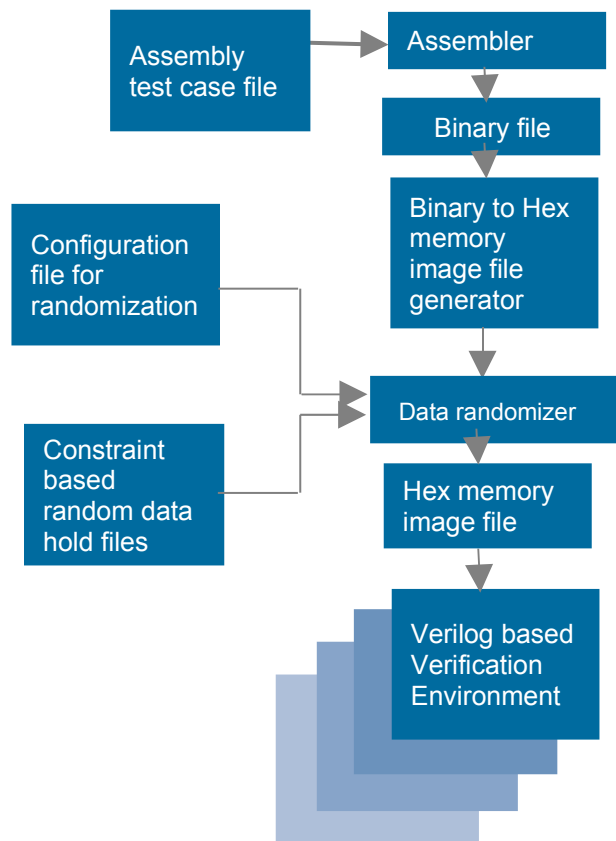
There are two ways to introduce randomization as a part of verification environment for processor based DUT. One is, generate random data using any means and put it into data section of assembly program. The other is to modify hex formatted binary image file of the assembly program using scripts.

In this section, the second approach is discussed. As it is known that randomization of configuration space register requires constraint based random data, one can generate database of constrained random data using various HVLs like Vera, Specman, SystemVerilog, SystemC

which brings pretty complex construct to support randomization. It's also possible to develop utility (e.g. Perl) which produce constraint based random data for specific requirement. Support of user defined constraint on output of random data generator helps to filter out invalid configuration particular to DUT. However this will require developer efforts to make sure that randomization block will generate unique sequence of configurations.

Verification environment for the processor based SoC requires test cases in assembly language particular to the processor being used in SoC. There is no construct or syntax in assembly language which allow constrained random value to be select for configuration space. However various intermediate stages of verification flow allow VE to modify data section of binary image file before passing it to verilog based VE.

The below depicted flow for passing binary image file, to Verilog based VE, generated by assembler, is common for processor based SoC. This flow introduces insertion of randomized data into hex formatted binary image file:



- Configuration file for randomization contains starting address, ending address and name of file which contains constraint based random data particular to peripheral block
- Constraint based random data hold files contain constraint random data particular to peripheral block.
- Data randomizer is a script which will decode the configurations supplied by configuration file for randomization and modify the hex image file by copying data from constraint based random data hold files.

Verification environment can add constrained randomized data into final hex image file before Verilog simulation start. Developer can use various available scripting languages to modify hex formatted binary image file of assembly language coded test case. This approach can work because current processor contains separate data space and code space.

- **Advantages:**

- It reduces number of directed test cases for highly configurable Design IP.
- Generates different and unique scenario every time one runs test.
- Allows to generate same scenario again by passing seed value.
- Helps to cover uncovered scenarios from directed tests.
- Reduces test development cycle
- Random data generator available as part of programming language.
- Requires one time framework which includes constraint on randomization and flow of randomization of configuration space.

- **Conclusion:**

There would always be tradeoffs to applying various approaches in verification. The contemporary randomization is best suit for processor based SoC. This is because it requires one time development effort only and reduces the compilation time as insertion of random data into data space is not part of test case. This makes development of test case independent of knowledge steps required to add randomization. Again randomization method will only be useful if randomized tests get run multiple times.