

Verification of Configuration  
Registers

*By Kunal Shah*



## Verification of Configuration Registers

The functionality of Complex SOC and ASIC designs depends on the bits of the configuration registers. As insignificant and trivial as it may seem to be, improper verification of configuration registers may result in major time loss due to faulty outputs and regressions. It may take a few hours to a couple of weeks to correct these errors, seriously affecting the Project Schedule. However, certain set of guidelines allow risk-free and least time-consuming verification.

The article describes a proposed verification flow for verifying configuration registers.

### **Register Verification Flow**

#### **Test plan development:**

Larger designs have hundreds or thousands of registers with different attributes. The complete register verification plan can be divided in different test groups:

- Register default value verification:

The test case (or test cases) under this test group should cover the default value testing of all registers.

- Register attribute verification:

The test case (or test cases) under this test group should cover attribute testing of all registers. The attributes may be R (Read Only), RW (Read/Write), U (Update by software/core), S (Set) and C (Clear).

(Note that in a single register different fields (bits) may have different attributes.)

- Register functional verification:

All the registers are not only used to read and write all the times. Sometimes based on its current value the core may have to perform some functions. For instance, whenever application software sets the 'soft reset' bit, the design has to start re-initialization. Register functional verification test group covers these scenarios, wherein, based on the value of the register, the design has to perform some function.

### **Verification Strategy:**

The common operations for verification of registers are:

- Generate Read Transaction and manually check the default value
- Generate Write Transaction with any directed/random value
- Generate Read Transaction and compare the return value with the predicted one

Manual testing of configuration registers is the most risky and tedious task. Let's understand how the risk is involved in manual/directed testing. Generally, in manual testing 'Write Transaction' are generated with some directed value and then are returned to the 'Read Transaction', allowing manual prediction of appropriate value as per the attributes of the register.

Due to manual prediction it isn't possible to perform random testing. However, during attribute testing, all the fields, i.e. each and every bit of all the registers must be tested with 0 and 1. Verification aims at achieving 100% functional coverage for register attribute testing. With directed testing it is not always possible, subsequently, causing bugs.

### **Register Model Utility:**

The best way to get 100% functional coverage with least hassle for register attribute testing is to use a register model. Usage of register model in verification environment eliminates tedious and error-prone processes of manually managing registers and enables design, verification and firmware teams to work more efficiently for a consistent and synchronized chip design flow.

The register model can be hooked up at a proper place, such as a 'monitor', in any verification environment. Minor modification in monitor enables auto verification of all the configuration registers for the generated Read/Write transaction.

### **Test case development:**

There are multiple ways of developing test cases for configuration registers. If the verification engineer develops one test case for each register, then we may find hundreds or thousands of test cases. The total test case count may help in impressing the client (designers) but it has certain drawbacks also.

For each test case, the design/link needs to be initialized before the register testing starts. If the design/link initialization time is around 1 minute, (it may be higher for more complex/larger designs) and there are thousands of registers with multiple regressions considered, the total simulation time could run into weeks! Rather, a better approach would be to define a single test case that can verify the default value of all the registers.

Another test case could verify attributes of all the registers. Here, the design/link is not getting initialized while testing each register. Instead, the design/link is initialized only once and all the registers are tested. This way we can drastically reduce the simulation time and subsequently, regression time. It may reduce regression time of couple of days into few hours. In some cases like functional verification test group, it may be required to define separate test cases for each scenario. For default value and attribute testing, it may be better to reduce the number of tests.

### **Considering corner scenarios:**

For register verification there may be different kinds of corner scenarios. One of them is mentioned here which is very common but most of the time neglected.

For instance, there is an error in encoding/decoding of the register addresses. The two register addresses are, say, 0x1000 and 0x2000. A bug in the design causes both the address locations to swap. Whenever WR transaction is generated for 0x1000, due to the error in address encoding, the design writes @0x2000 location. Similarly, whenever RD transaction is generated for 0x1000 the design reads and

returns data from 0x2000 location. In the same way, any WR/RD transaction generated for 0x2000 locations would be responded to/from 0x1000. It makes it all the more challenging to debug the design. Even the usage of register utility proves to be ineffective. These scenarios need manual testing.

**Conclusion:**

In a nutshell, to get better results and success in register verification a specific strategy needs to be defined by keeping in mind below issues:

- Definition of test plan with well defined test groups
- Usage of register model utility in VE to get rid of manual testing
- Considering simulation time while creating test cases
- Considering corner scenarios