6.5 Verification and test Plan

The terms "verification" and "test" are often interchanged because they both deal with the concept of checking that the item in question is operational as intended. However, those two terms have different connotations as verification deals with the "what" to verify, and test deals with the "how" to implement the verification. Thus,

- **A verification plan** addresses the items to be verified, but without addressing the methodologies. For example, a verification plan for a CPU will address that the items to be verified include the ISA, the IOs, environment (e.g., ISA mix, memory types (fast/slow), application software written in X language, etc).
- **A Test plan** addresses how the items that need to be verified will be checked. For the CPU example, the test methodology may include simulation, emulation, use of assertions, use of UVM, constrained random tests, types of mixes, test application code, tools, instruments, etc.

The verification plan is a specification for the verification effort. It provides a strawman document that can be used by the design community to identify, early in the project, what needs to be verified. Early mistakes in the verification approach can be identified and corrected. A byproduct of the verification plan exercise is the revisit of the requirements. This enforces the process of verifying those requirements, thus helping in the identification of poorly specified or ambiguous requirements.

A test plan is a document that defines the following:

1. **The verification technologies**. The plan identifies the verification technologies that will be used for the project. These technologies include assertions, verification libraries, functional coverage, cross coverage, linting, code coverage, frameworks (e.g., UVM), simulators, emulators, formal verification, and tools. It should also identify how these technologies are used. For example, formal verification may be used at the subblock level, while simulators may be used at the chip level, and emulators may be used at the system / software verification level.

2. **Verification environment for the design-under-test**. This includes the structure of the testbench, and special instructions. The structure encompasses the component models (at the interface level), packages (at the declaration or higher level), and file structures. The verification environment will also include verification units to insure that the actual results produced by simulation of the design meet the expected results.

3. **Tests or transactions applied to the design**. These tests are used to verify the design's functional correctness as specified in the requirements specification. This includes tests at the top-level of the design as well as the subblocks. SystemVerilog Assertions can be used to specify assumptions about inputs, and transactions at the interfaces, along with expected results within a range of cycles, to allow for variations in the DUT cycle timing. Many of these transactions can be extracted from the requirements documents (system and architecture). A current trend in verification is to automatically generate stimulus/tests using the SystemVerilog Assertions and assumptions as a base for the definition of the constraints.

4. **Exit criteria**. These criteria identify when verification is complete, or at least achieved the goals. They may include code and functional coverage, as well as passing all tests and lint checks, etc. Code coverage may include line, branch, condition, toggle, path and FSM.[52] Functional coverage represents a user-defined model of functionalities of the design that the verification process should address. SystemVerilog has a rich set of constructs to capture the coverage model.

---

[52] Verification Methodology Manual http://vmmcentral.com/