**Introduction and history**

During the past decade, the time spent by the verification engineers to verily the functionality of the designs rose to more than 60% of the total project time. Even developers of smaller chips and FPGAs are having problems with the past verification approaches. It has become more difficult to get our goal of verification using conventional verification techniques.

Due to the huge functional space of a processor, processor verification is considered to be one of the most challenging problems facing verification engineers. The functional verification of a processor is the process of raising the confidence level in the compliance of a processor design to its specifications. The verification process begins with creating a verification plan that defines what properties and functionalities need to be verified, what are the methods and approaches that will be used in processor testing and what is the expected behavior of the design. As, it is the process of defining functional coverage models and functional specifications of the verification. Furthermore, the testing strategy is one of the major decisions taken in the verification planning phase. Directed testing is well suited for testing single functionalities, but it is hard to hit more complex scenarios using only directed testing. On the other hand, constrained-random verification (CRV) can be very effective in tackling processor verification challenges, such as: complex instruction sets, multiple pipeline-stages, in-order or out-of-order execution strategies, instruction parallelism and multi-precision operations. The most important module of a CRV environment is the test-case generator, which plays a very important role in most of the recent approaches towards developing automated processor verification environments. A test-case generator generates a large set of valid test cases in a pseudo-random way controlled/guided by constrained randomness. The development of such test generators has started to catch attention of functional verification engineers, and researchers since the early 2000s.

Due to the poor features of Hardware Description Languages (HDLs) available back then, Verilog and VHDL, in terms of verification and software, the development of these generators have been categorized as a software problem, tackled either by building them as software applications [6] or by designing new scenario-level languages, such as Test-Template Language. However, recent efforts have been exerted towards the utilization of System Verilog features as a Hardware Verification Language (HVL) to improve stimulus generation quality. The Universal Verification Methodology (UVM) gradually dominates the verification world, as it covers these needs. UVM is a powerful verification methodology that was designed to be able to verify a wide range of design sizes and design types. It is a standardized methodology for verifying SoC designs. UVM is derived mainly from the OVM (Open Verification Methodology) and the (VMM) Verification Methodology Manual. It is an open source SystemVerilog library allowing creation of flexible, reusable verification components and assembling powerful test environments utilizing constrained random stimulus generation and functional coverage methodologies.