Generic Verification Methodology for Soft Processors Based on UVM

***Abstract*—Processor verification has always been a challenging problem for verification engineers. The ever-growing complexity of processor designs increases the verification complexity, and the gap between a verification plan and the available technologies to implement it. This increases the time spent on the implementation, and leads to a questionable quality of the verification process. In this paper, we propose UVM,The Universal Verification Methodology is a powerful verification methodology that was architected to be able to verify a wide range of design sizes and design types.Also it's an efficient method for verification due to its re-usability and constrained randomization. The main contribution that this paper introduces is implementing a generic UVM i.e: building one verification environment that can be used to verify any microprocessor with the same specifications and features of the cores that are used in this project to prove the idea. The whole design of the environment is nearly similar to the basic UVM design but with some differences to be suitable for the idea of generic UVM. According to the differences between cores and their supported ISA, interface is changeable with respect to the core under verification, each core has its own package and sequence item. the design has more than one test in order to support all the available instructions with their different nature. More details about design of the test bench and its implementation will be discussed later throughout the paper.**

Keywords–generic, SoC, soft processor, verification, UVM

Chapter I

I. Introduction

The increased complexities of SoCs have led to significant increase in the verification efforts that are imperative to meet the design specifications. It is realized that traditional simulation methods no longer suffice SoC design verification for complete system level verification, such that, the design verification is the process of checking that a given design correctly implements the specification and the required functionality [1].

70 percent of the development cycle is dedicated to design verification. Based on that, verification team to Design team ratio ranges from 2:1 to 3:1. As the main goal of verification is to ensure that the design implementation matches the specified specifications. An efficient methodology is needed in order to conquer the challenges resulted by the enormous state space of a processor. The challenge is increased further when the methodology needs to be able to cope with verification of several processor families, especially when they are based on completely different instruction set architectures (ISAs). The Universal Verification Methodology (UVM) is a standardized methodology for verifying SoC designs. UVM is derived mainly from the Open Verification Methodology (OVM) and the Verification Methodology Manual (VMM). SoC can be verified effectively using a simulation test bench that provides data to the chip inputs and checks resulting data on the chip outputs. A modern test bench-based verification environment automatically generates randomized stimulus for the chip inputs under control of user-specified constraints and checks the results of each test automatically. UVM is the best verification methodology that has been created to develop constrained-random test benches in a uniform fashion and to permit limited reuse of test bench components [2].

In this paper, a generic methodology for processor verification using the Universal Verification Methodology and SystemVerilog capabilities is proposed. A case study is presented demonstrating our proposed methodology on three open source processors: Riscy, LEON 2.4, and Amber23.

The rest of the paper is organized as follows: In chapter II, the proposed solution and methodology. In chapter III, the implementation. In chapter IV, the validation and results. And the paper is concluded in section V.

II. History And Background

During the past decade, the time spent by the verification engineers to verify the functionality of the designs rose to more than 60 percent of the total project time. Even developers of smaller chips and FPGAs design are having problems with the past verification approaches. Our goal of verification becomes more difficult to get using conventional verification techniques.

Due to the complex design 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 demonstrating the functional correctness of a processor design with respect to its specifications. The beginning of the verification process is 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 behaviour 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 more convenient 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 efficient 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 significant 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 and guided by constrained randomness. The development of such test generators has started to get the attention of functional verification engineers, and researchers since the early 2000s.

Due to the poor and weak 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 or by designing new scenario-level languages, such as Test-Template Language. However, recent efforts have been spent towards the utilization of System Verilog features as a Hardware Verification Language (HVL) to improve stimulus generation quality. The UVM gradually prevails the verification world, as it covers these needs. UVM is a powerful verification methodology that was designed to be able to verify a variety of different design sizes and design types. 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 [3]. There are two types of cores or processors: soft processors and hard processors. Soft processors is a microprocessor core that can be wholly implemented using logic synthesis. It can be implemented via different semiconductor devices containing programmable logic (e.g., ASIC, FPGA, CPLD).A hard-core processor is a processor that's actually physically implemented as a structure in the silicon.

III. Problem definition

It becomes possible for electronic system designers to assemble complete systems-on-chips due to the ever increasing advances in the integrated circuit technology. At the same time shrinking time to market leaves little room for errors in the design. So functional verification has become one of the main tasks in committing chips to fabrication. The need of standalone, pre-verified verification infrastructure is arisen so that verification does not become the bottleneck for the designers .The Verification Intellectual Property (Verification IP) is an important component of such infrastructure which can be easily inserted in the simulation-based tests. Any IP (e.g., a microprocessor) life-cycle from the verification process point of view can be divided into three phases: pre-silicon verification, post-silicon validation and runtime verification. Processor development takes a long time (2 years or more) and it is important to be able to detect bugs at all stages of processor development. Verifying a processor takes longer than design: the long tail of processor development is developing new tests for the processor and fixing any bugs. It is important that useful results can be obtained even in the early stages of verification — before the complete test infrastructure has been developed. Any verification technique requires significant investment so reusability not only of the technique but also of the infrastructure is critical [4]. UVM is used as it has Improved Verification quality due to constrained random verification. Test benches are Reusable where The verification components or agents get instantiated within a verification environment inside a project, and they may require some modification to suit the requirements of this verification environment, also The overall verification environment could also be used and modified according to the requirements of a certain test.

As a result to all the above reasons, there's a great demand for a reusable generic environment to verify microprocessors. Verification of IPs or even microprocessors using generic UVM becomes common and applicable but the problem of verifying microprocessors by using separate verification environments or different approaches is still found, this problem cost the product a lot of time and effort to get into the market, so the need to make one verification environment or generic verification method for microprocessors is increasing every day. UVM is the best method to be used to make this generic verification environment due to its simplicity, reusability and its power in coverage and function testing.

IV. objective

In this project, the main objective is to implement only one generic UVM used to verify the functionality of different cores(soft processors). these cores are mainly different in the instruction set architecture, the micro-architecture itself like the pipeline stages , the behaviour of cache memories,...etc. Our generic UVM tries to prove this idea using three different microprocessors which are:

- Amber 23 based on ARM-V2a ISA with three pipeline stages.
- Riscy core based on RISC-V ISA with four pipeline stages.
- LEON 2.4 based on SPARC-V8 ISA with five pipeline stages.

Our test bench can be used in verifying any core based on a certain ISA compatible with that of the three microprocessors used in proving the concept. Compatibility refers to have nearly the same specifications,,bus data width.

V. related work

For the best knowledge of the authors, this is the first Contribution in verifying different microprocessors (soft processors cores) using only one verification environment.All the related works are based on the idea of implementing a UVM test bench to verify only one IP or even a microprocessor but it's first time to build one generic environment to verify different cores with the same testbench without any tweaking in the main components of the testbench itself.

Chapter II

A. Challenges lead to the proposed solution

This section introduces the challenges that lead to the proposed methodology including the verification plan. In section B, the proposed verification methodology (including the UVM) is discussed.

Building a generic environment for verifying different processors becomes a great demand but for reaching an efficient and professional test bench, verification steps must be done in an effective manner to get the required results, thus the verification Steps are:

1) Understanding the design specifications.
2) Creating a verification plan.
3) Identifying the verification methods required, test benches, coverage, and stimuli.
4) Building the Verification environment (this part will be covered in details).
5) Executing a plan: The development and running tests, find bugs, regress the test suite, create functional coverage.

In this chapter the first three points in the verification steps will be discuss clearly.

The design specification in our case can be interpreted into studying the architecture of the cores well. The case study here is three different open-source cores based on different ISA and microarchitectures. Then building the verification plan that has to include all the previously studied specifications.

First of all, a general definition of the verification plan should be discussed which states that, it is a specification document for verification effort and a mechanism to ensure all essential features are verified as needed; it shows steps and effort needed to verify a specific design, So it should include the following criteria:

1) What to verify? Including the interesting design cases.
2) Methodologies on how to verify.
3) The required stimulus, checkers and coverage.
4) Priority for features to be verified in a specific design under test (DUT).

Based on that, our plan has to demonstrate the technique that will be used in the implementation, or the different scenarios that must be taken into consideration to test all the functions. In other words, it clarifies the test cases that can grouped into one function or even the different test cases among the cores. Figure 1 shows the flow of the verification plan.
As described in figure 1, the verification plan for our test bench is divided into two main parts:

- The First part includes all the interface signals (all input and output signals from and to the core) taking into consideration that we deal with the tested DUTs as a black box. Our implementation is built such that, the interface is dynamic (changeable according to the used DUT) and the environment is fixed (this point will be discussed in details in the Implementation chapter).
- The second part of the plan is the functionality test description for the ISA of each proposed core. Where the instructions needed to be verified are divided into functions. As the main challenge for implementing a generic test bench is that making it compatible with different ISA's as much as possible to save time and effort during the design verification process. Therefore, the main advantage of this plan is to analyze the similarities and differences between the three cores to act as a proof that we can verify different processors using the same environment.

For more clarification, the main challenges that we faced during designing the plan are:

- The execution of the instruction in each core is different from the other.
- Mapping the instruction format is also different in each core.
- The instructions access to certain flags is different from each core.

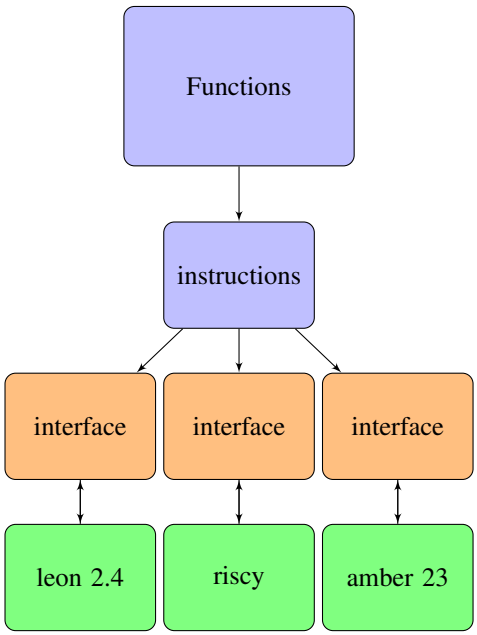

Figure 1. The verification plan flow

B. The Proposed Methodology

UVM is used to improve the verification quality due to the presence of constrained random verification. As the design complexity of SOCs and ASICs is increasing, so traditional test benches started failing the primary verification objective

of bugs catching. Because, it was becoming impossible to simulate all the possible scenarios with the traditional test benches. As a result, the constrained random verification concept is basically allowing the user to generate random test vectors, which provided a way of exercising the DUT with more combinations of inputs in less simulation time. Constrained random verification depends on Checkers, Coverage and Constraints. Each of these "three C's" plays a key role in the verification process and is supported by SystemVerilog language. Furthermore, UVM enables the test Bench Re-usability on two main sectors:
a. The verification components or agents get instantiated within a verification environment inside a project, and they may require some modification to suit the requirements of this verification environment.
b. The overall verification environment could also be used and modified according to the requirements of a certain test.

The main goal of our methodology is to obtain a generic test bench for processor core verification, in other words, implementing a general and reusable verification methodology for any core with ISA compatible with the proposed ISAs, as our target is to verify the supported instruction set by the core. This is achieved by building the UVM test bench in a generic way then adding the core testing, that mainly includes the instructions of the tested core and anything related only to that core, and the tested core interface, described below. The main benefit of this methodology is that it enables the test writer to develop the tests based on a generic and reusable way, while keeping the microarchitectural claims and the instruction set details included in the added package and interface. Based on that, the proposed test bench can be used for both simple and complex design tests. This test bench is built in a layered way, keeping the microarchitectural claims at the lower level, and letting the test writer uses the proposed test bench for different test scenarios. The proposed methodology is shown in figure 2.

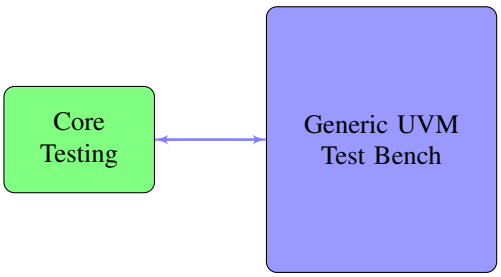

Figure 2. The proposed methodology

The structure of the methodology shown in figure 2 consists of:

1) Generic UVM test bench: The generic UVM test bench includes all the UVM components for the processor core testing.
2) The core testing: it includes
   - core package:It includes all the processor instructions that to be verified.
   - The processor interface: It includes the input and output ports of DUT. The interface has functions that the driver uses to drive instructions and data to the DUT, moreover, it deals with the clock and timing of each core regarding the timing constraints.
   - the processor target sequence item including its format, Opcodes,...etc.
   - top module of the Core including signals' instantiation.
   - Core defines and its own parameters.

To sum up the base that our idea is built on: verification of the soft processor core is done by connecting our UVM testbench to the target core's interface, Also its own package including all the supported ISA that is compatible with our cores and any features relate to this core. In the next chapter, all of that will be discussed clearly and the mechanism of doing that, in addition to that, the different scenarios and tests that has been done.

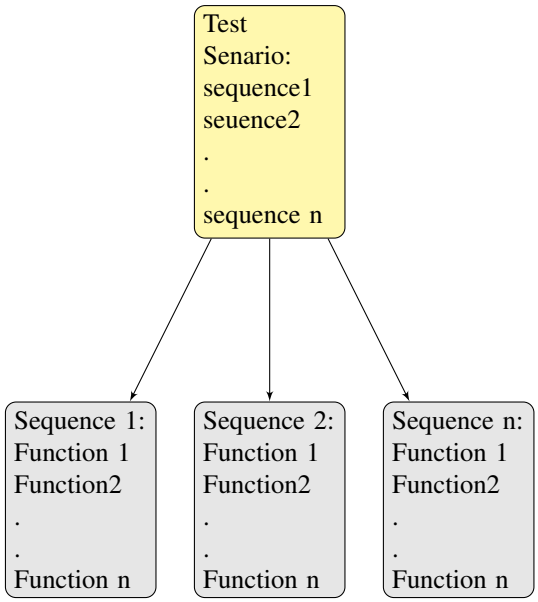

Figure 3. The test layered solution.

According to the mentioned challenges that were faced during the core verification, the proposed methodology includes a layered solution for the random generated stimulus for verification, as the proposed test bench enables different testing scenarios, each test enables different sequence scenarios for the group of instructions that have related functions, for example, a scenario of functional testing, includes the core instructions testing, which has different sequence scenarios for different group of instructions, like a sequence of testing the

Arithmetic Logic Unit (ALU) instructions that have variety of functions, for example, Add, Subtract, Shift,....etc. Figure 3 describes the proposed solution for the stimulus generation. The layered solution shown in Figure 3 consists of:

- Test Scenario:
  It is the desired scenario for testing the DUT, which is the test for functional verification in our case.
- Sequence:
  It includes the stimulus of the group of instructions that have related functions, for example, a sequence for ALU instructions.
- Instruction's function:
  It is the instruction function to be verified, for example, adding functionality is done by Add instruction.

REFERENCES

[1] "What's the Deal with SoC Verification?", Electronic Design, 2020. [Online]. Available: https://www.electronicdesign.com/technologies/dsps/article/21795646/whats-the-deal-with-soc-verification.

[2] "VLSI design: Free online UVM training from Aldec", Eeherald.com, 2020. [Online]. Available: http://www.eeherald.com/section/news/onws2013020689.html.

[3] Khairallah, Mustafa and Ghoneima, Maged. (2014). Reusable Processor Verification Methodology Based on UVM. 10.13140/2.1.3936.9926.

[4] Alastair Reid, Rick Chen, Anastasios Deligiannis, David Gilday, David Hoyes, Will Keen, Ashan Pathirane, Owen Shepherd, Peter Vrabel, and Ali Zaidi," End-to-End Verification of ARM® Processors with ISA-Formal ",Computer Aided Verification, 2016.