|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Academic Year: | 2022/2023 | Term: | Spring 2023 |  |
| Course Code: | ELC 4802 | Course Title: | Project (2) |

Graduation Project  
Digital Verification of OpenHMC Controller IP

- Sponsored by Si-Vision -

Cairo University

Faculty of Engineering

Electronics and Communications Engineering Department – 4th Year

2 – Universal Verification Methodology

*Submitted to: Dr. Ibrahim Qamar*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | BN | Sec | ID | رقم الجلوس |
| رحمه حمدى عبد الله عبد اللطيف | 16 | 2 | 9190636 | 34073 |
| علاء ربيع سالم على | 3 | 3 | 9191445 | 34115 |
| علي ماهر عبدالسلام نبيه | 5 | 3 | 9190067 | 34117 |
| محمد احمد طه السيد | 30 | 3 | 9191043 | 34142 |
| محمد حسام عثمان يسن | 35 | 3 | 9191083 | 34147 |
| محمود موسى جاد موسى | 8 | 4 | 9190971 | 34176 |
| ندى عثمان محمد بركات الحمراوى | 39 | 4 | 9191568 | 34207 |

Table of Contents

[1. Introduction 1](#_Toc127485833)

[1.1 What is UVM? 1](#_Toc127485834)

[1.2 Why is UVM useful? 1](#_Toc127485835)

[2. Basic UVM Testbench 2](#_Toc127485836)

[2.1. Testbench for one interface 2](#_Toc127485837)

[2.1.1. Transaction 2](#_Toc127485838)

[2.1.2. Sequence 2](#_Toc127485839)

[2.1.3. Sequencer 3](#_Toc127485840)

[2.1.4. Driver 3](#_Toc127485841)

[2.1.5. Monitor 3](#_Toc127485842)

[2.1.6. Agent 3](#_Toc127485843)

[2.1.7. Scoreboard 3](#_Toc127485844)

[2.1.8. Coverage Collector 3](#_Toc127485845)

[2.1.9. Environment 3](#_Toc127485846)

[2.1.10. Test 3](#_Toc127485847)

[2.2 Scaling the Testbench 4](#_Toc127485848)

[3. References 5](#_Toc127485849)

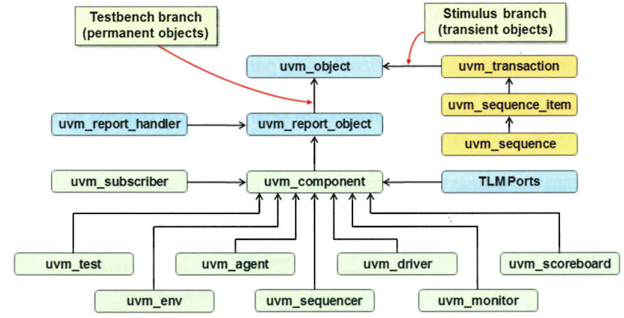
# 1. Introduction

In this report, we will define UVM and why does the industry need it. In addition, we will give a few examples of the architecture of the UVM testbench and how it can be modular, reusable and scalable without digging into the details of coding.

## 1.1 What is UVM?

Universal verification methodology (UVM) is a standard to enable faster development and reuse of verification environments and verification IP throughout the industry.

It’s a library of basic open-source classes we don’t have to know everything about but just include it and extend from it while writing our testbench. For example, think of C coding language if you need to use strings all you have to do is include the <string.h> library without caring about its content.

It is mainly derived from Open Verification Methodology (OVM) and is supported by multiple EDA vendors like Synopsys, Cadence, Mentor, etc.

## 1.2 Why is UVM useful?

The main idea behind UVM is to help companies develop modular, reusable and scalable testbench structures.

Its class library provides generic utilities like configuration database, TLM and component hierarchy in addition to data automation features like copy, print and compare.

It brings in a layer of abstraction where every component in the environment has one specific role and its code can be written separately.

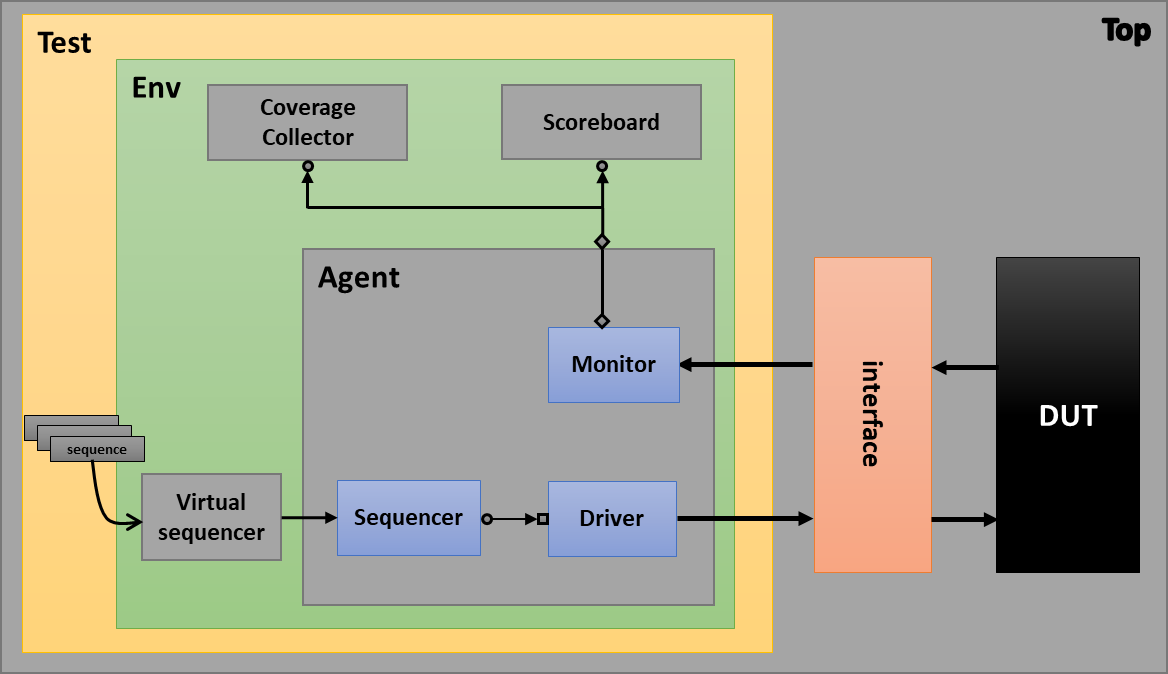
Hence, all testbench environments has the same structure and components for every IP and can be reused in different projects.

# 2. Basic UVM Testbench

First, we will define the different basic uvm components of the testbench for only one interface. Then see how can we scale it and reuse it to be a part of a larger design.

## 2.1. Testbench for one interface

We will break down and explain the major components of a simple UVM testbench hierarchy starting down to up.



### 2.1.1. Transaction

Extended from UVM class “uvm\_sequence\_item”. In the testbench not all blocks must have access to the pin level of the DUT. Transactions adds a level of abstraction so blocks transfers data between each other in the form of an object that groups all the pin level details into a single transaction.

### 2.1.2. Sequence

Extended from UVM class “uvm\_sequence”. If we want to test a certain scenario or a sequence of commands of the stimulus, we can write these scenarios in separate files and then the test passes the desired scenarios to the environment’s virtual sequencer which has a handle to the agent’s sequencer.

### 2.1.3. Sequencer

Extended from UVM class “uvm\_sequencer”. It is responsible for passing the sequences to the driver. Connected to the driver through a TLM port.

### 2.1.4. Driver

Extended from “uvm\_driver” class. Responsible for pin level operations to drive the DUT directly through the interface.

### 2.1.5. Monitor

“uvm\_monitor” it is a passive component doesn’t affect the DUT and only observe it’s input and output. Writes these data on an analysis port to whichever might be subscribed to this port.

### 2.1.6. Agent

“uvm\_agent” is like a wrapper that groups all components related to a specific interface or communicating with the same transaction.

### 2.1.7. Scoreboard

In a simple testbench it is extended from “uvm\_subscriber” to the monitor. But for more complicated testbenches it is extended from “uvm\_scoreboard” and the scoreboard is broken into 2 components. As it performs 2 operations.

* Observe the input from the monitor then predict its own output from calculations within the scoreboard
* Then compare the predicted output with the output observed from the monitor. If both are equal then the test is successful

### 2.1.8. Coverage Collector

It’s a subscriber to the monitor. We write all scenarios needed to be covered by the stimulus to the DUT.

### 2.1.9. Environment

At the top of the hierarchy. Instantiates the agent and have a virtual sequencer which contains the handle of agent’s sequencer

### 2.1.10. Test

Start the test by passing a certain sequence or sequences to the environment.

Next is how can we scale this small testbench with little edits.

## 2.2 Scaling the Testbench

Only difference is we can add a configuration option that when we can communicate with multiple interfaces. Just by taking an agent already set from an old environment and add it to be a part of a bigger testbench as follows.

The environment controls the configuration of each agent choosing which is active and which is passive by setting the “is\_active” handle to the state of the agent. So many different scenarios can be tested with passing sequences to each agent and monitor the DUT.

# 3. References

* [UVM Standard](https://www.accellera.org/downloads/standards/uvm)
* [The UVM Primer by Ray Salemi](https://cutt.ly/M3GIFlw)
* [ChipVerify](https://www.chipverify.com/uvm/uvm-tutorial)
* Si-Vision Academy UVM Tutorial
* [Paper: Using UVM Virtual Sequencers & Virtual Sequences](https://cutt.ly/n3GI4iz)
* [Paper: OVM/UVM Scoreboards - Fundamental Architectures](https://cutt.ly/w3GPebe)
* [Paper: SystemVerilog Assertions](https://cutt.ly/43GP1tQ)
* [Paper: Configuration in UVM – The Missing Manual](https://cutt.ly/q3GAajX)
* [Paper: Mastering Reactive Slaves in UVM](https://cutt.ly/g3GALWt)