|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 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

1 – Introduction to Digital Verification

*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](#_Toc127216182)

[1.1. ASIC Digital Design Flow 1](#_Toc127216183)

[1.1.1. Design Specifications (specs) 1](#_Toc127216184)

[1.1.2. RTL Design 1](#_Toc127216185)

[1.1.3. Functional Verification 2](#_Toc127216186)

[1.1.4. Logic Synthesis 2](#_Toc127216187)

[1.1.5. Formal Verification (Formality) 2](#_Toc127216188)

[1.1.6. Placement & Routing 2](#_Toc127216189)

[1.1.7. Sign Off 2](#_Toc127216190)

[1.2. Major Digital Electronics Careers 2](#_Toc127216191)

[1.2.1. Digital Design Engineer 2](#_Toc127216192)

[1.2.2. Digital Verification Engineer 2](#_Toc127216193)

[1.2.3. Physical Implementation Engineer 2](#_Toc127216194)

[2. Introduction to Verification 3](#_Toc127216195)

[2.1. Why is Verilog not preferred? 3](#_Toc127216196)

[2.2. What are the features of SystemVerilog? 3](#_Toc127216197)

[2.3. How to perform functional verification? 3](#_Toc127216198)

[2.4. Basic Testbench Functionality and Structure 4](#_Toc127216199)

[3. OOP in Verification 4](#_Toc127216200)

[3.1. The Four Pillars of OOP 5](#_Toc127216201)

[3.1.1. Encapsulation 5](#_Toc127216202)

[3.1.2. Abstraction 5](#_Toc127216203)

[3.1.3. Inheritance 5](#_Toc127216204)

[3.1.4. Polymorphism 5](#_Toc127216205)

[3.2. why we use OOP? 5](#_Toc127216206)

[4. Layered SystemVerilog Testbench 5](#_Toc127216207)

[4.1. Signal Layer 6](#_Toc127216208)

[4.2. Command Layer 6](#_Toc127216209)

[4.3. Functional Layer 6](#_Toc127216210)

[4.4. Scenario Layer 6](#_Toc127216211)

[4.5. Test Layer 7](#_Toc127216212)

[5. Constrained Random Coverage Driven Verification 7](#_Toc127216213)

[5.1. Direct Stimulus 7](#_Toc127216214)

[5.2. Pure Random 7](#_Toc127216215)

[5.3. Constrained Random Stimulus 7](#_Toc127216216)

[6. Coverage 8](#_Toc127216217)

[6.1. Code Coverage 8](#_Toc127216218)

[6.2. Functional Coverage 8](#_Toc127216219)

[7. References 9](#_Toc127216220)

# 1. Introduction

In this report, we will give a brief explanation of each of the following topics

* ASIC digital design flow
* Different digital electronics specializations
* Role of digital verification engineer
* Why do we need OOP and SystemVerilog
* The most common layered SystemVerilog testbench

Before starting to learn the technical skills of verification, we need to understand its role and position (puzzle piece) in the cycle of digital IC design flow. In addition, understand why the HDL language “Verilog” isn’t enough for current requirements of verification of complex designs and why people moved to a new hardware verification language called “SystemVerilog” to build testbenches.

## 1.1. ASIC Digital Design Flow

It can be best explained by a flowchart of the typical steps needed for the fabrication of an IC

Design Specifications

RTL Design

Functional Verification

Logic Synthesis

Formal Verification

Floor Planning

Place & Route

Sign Off

Fabrication

### 1.1.1. Design Specifications (specs)

Typically, in industry field the chip functionality is described in “Requirement & Specification Document” by the system engineer according to agreement with the customer.

### 1.1.2. RTL Design

The Design Engineer starts to plan the architecture blocks of the design to meet the sytem requirements. Then model it using a hardware description language like Verilog/VHDL but currently Verilog is the most commonly used in the industry. This language has to be synthesizable as the code must be compatible with the synthesis tool to represent logic gates.

### 1.1.3. Functional Verification

The Verification Engineer builds the verification environment to test the functionality of the design. Making sure the RTL meets all the specs. By simulating multiple test cases and reporting functional coverage. If either a bug is found or the design not achieving a spec then report back to design engineer.

### 1.1.4. Logic Synthesis

A process that translates the HDL code into a gate-level netlist, optimized for a set of timing, power and area constraints.

### 1.1.5. Formal Verification (Formality)

Formality is a tool which verify the functionality equivalence of the gate netlist against the RTL.

### 1.1.6. Placement & Routing

The process that places the cells of the chip and plan routing between blocks that target minimizing the total interconnect length.

### 1.1.7. Sign Off

After the final layout is ready. In SignOff, we check LVS (Layout Verses Schematic), DRC (Design Rule Constraint) and Timing Closure of the netlist. If all successful then generate the IC layout in the form of a GDS file which is sent to fabrication.

## 1.2. Major Digital Electronics Careers

There are many puzzle pieces to form the cycle of digital design which are modelling, design, verification, integration, backend and layout. We will mention 3 major specializations

### 1.2.1. Digital Design Engineer

Responsible for writing a synthesizable & optimized RTL using HDL language as Verilog/VHDL to meet the specs.

### 1.2.2. Digital Verification Engineer

Responsible for building a verification environment by first developing a test plan containing all needed test cases and scenarios to make sure the design works as intended.

### 1.2.3. Physical Implementation Engineer

Responsible for implementing the design floor planning, pin placement, power planning, cells placement and routing. Preforms static timing analysis and timing closure, DFT insertion and GDS generation.

These are 3 of the major roles of a digital electronics engineer. Our project is categorized under the role of the verification engineer. Next, we will dig into the basics of verification.

# 2. Introduction to Verification

The goal of verification isn’t to prove that the design is bug-free as no design can be entirely bug free for all scenarios. What our focus lies on is to make sure the design can accomplish its task successfully according to specifications.

Currently designs’ complexity is growing more and more. So, the process of chip fabrication is very time consuming and costs millions. Any defects in the design if caught at an early stage will save time and cost of repeating many processes in the flow.

## 2.1. Why is Verilog not preferred?

Back in the 1990’s, Verilog was the primary language to perform functional verification of designs but it was small and not very complex at the time. As design complexity increases, better tools are needed which lead to moving to currently SystemVerilog and research for better and simpler verification methods is continuous.

## 2.2. What are the features of SystemVerilog?

It’s basically Verilog with some added features. Some of these features are:

* Compatible with Object-Oriented Programming
* Generation of constrained-random stimulus
* Functional coverage
* Transaction level modeling

There are many other useful features, but these allow us to create testbenches at a higher level of abstraction than with an HDL or a programming language such as C.

## 2.3. How to perform functional verification?

A hardware design consists of several Verilog (.v) files with one top module that instantiate lower level submodules.

An environment which can be called a testbench is required for the verification of a design. The idea is to drive the design with different stimuli to observe its outputs and compare it with expected values.

In order to do this, the top module is instantiated within the testbench environment and it’s given a name DUT which stands for design under test, and connect the design input/output ports with the appropriate testbench component signals. Drive the inputs to the design with certain values then observe and compare the outputs of the design.

Testbench

DUT

addr

wr\_en

wdata

rdata

rd\_en

clk

clk

## 2.4. Basic Testbench Functionality and Structure

The purpose of a testbench is to determine the correctness of the DUT. This is accomplished by the following steps.

* Generate stimulus
* Apply stimulus to the DUT
* Capture the response
* Check for correctness
* Measure progress against the overall verification goals

The usual structure of a SystemVerilog testbench consists of multiple layers of abstractions each level contains classes that have a certain role.

This can be achieved by the help of OOP which will be explained next.

# 3. OOP in Verification

Object-oriented programming is a different approach of coding from the known procedural programming where data and functions are stored in different memory locations.

**Procedural Program**

**Module**

**Module**

**Subprogram**

Data

Functions

**Subprogram**

Data

Functions

**OOP Program**

**Module**

**Module**

**Object**

**Object**

Data + Functions

Data + Functions

OOP brings encapsulation and abstraction grouping related data and functions inside an object or class and storing them in the same memory space.

## 3.1. The Four Pillars of OOP

Part of the reason why software and verification engineers love OOP is that it provides many useful features such as.

### 3.1.1. Encapsulation

Wrapping up of data and information under a single unit.

### 3.1.2. Abstraction

Displaying only essential information and hiding the details.

### 3.1.3. Inheritance

Allows classes to be derived from other classes inheriting its properties and methods.

### 3.1.4. Polymorphism

Allows subclasses to define their own unique behaviors and yet share some of the same behavior of the parent class.

## 3.2. why we use OOP?

Features of OOP benefits engineers in many ways

* Code reuse (Reusability)
* Code Maintainability
* Memory Management

In general, because nowadays the designs are very complex. Building an environment consumes times so it’s very useful if some classes of an old environment can be reused. OOP allows us to build a modular, scalable, configurable and re-usable testing environment.

# 4. Layered SystemVerilog Testbench

A key concept for any modern verification methodology is the layered testbench. Although this process may seem to make the testbench more complex, it actually helps to make the task easier by dividing the code into smaller pieces (classes) that can be developed separately. With a higher-level transactions and sequences transferred between these classes. In addition, a layered approach allows reuse and encapsulation of Verification IP (VIP) which are OOP concepts.

A general layered testbench and its component are explained as follows.

**Functional Coverage**

**Driver**

**Assertions**

**DUT**

**Monitor**

**Agent**

**Checker**

**Scoreboard**

**Generator**

**Test**

**Environment**

Signal

Functional

Command

Scenario

it consists of 4 major layers either layer drive the lower layer and doesn’t know all the pin level details of the DUT.

## 4.1. Signal Layer

Contains the DUT and the signals that connect it to the testbench.

## 4.2. Command Layer

It deals directly with the DUT using pin-level commands

* Driver: converts transactions to pin-level activity
* Monitor: observe pin-level activity without having an effect on the DUT

## 4.3. Functional Layer

Abstract layer that feeds down command layer with basic commands

* Agent: receives higher level transactions and sends them to driver and scoreboard
* Scoreboard: predicts the results of the transaction received from agent
* Checker: compared the expected results from scoreboard with observer results of the monitor

## 4.4. Scenario Layer

Drives the functional layer according to required scenario or sequence which is produced by the generator or sequencer.

## 4.5. Test Layer

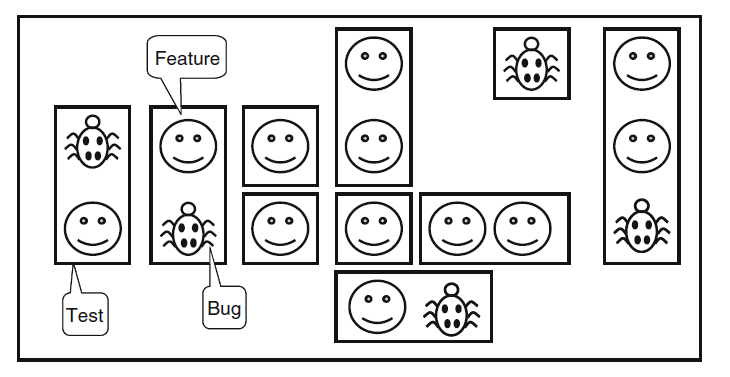
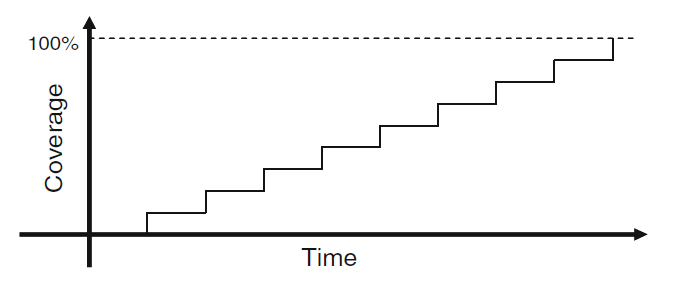
Top level used to control the generator and measure coverage of verification plan test cases. It acts as the conductor in an orchestra.

Now all what is left is how we drive the stimulus and how to measure the coverage.

# 5. Constrained Random Coverage Driven Verification

To explain what constrained random means. We have to mention 3 different approaches to drive the stimulus

## 5.1. Direct Stimulus

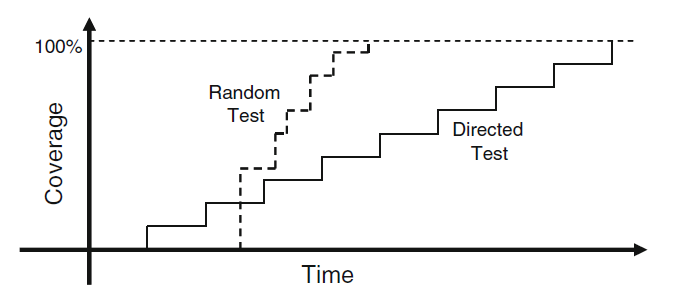
 Manually writing all the tests to cover all cases which is highly time consuming with poor scalability and maintainability which isn’t preferred for an environment

It discovers bugs that we expect from the testcases and no ability to discover new areas in the design.

## 5.2. Pure Random

Randomize stimulus with no constraints which may generate new corner testcases that wouldn’t normally be tested. But there is no way to prevent it from generated illegal stimulus as what might break the protocol of the design for example.

## 5.3. Constrained Random Stimulus

Constrain the randomization within a certain bounds and probability to benefit testing interesting scenarios to hit all possible corner cases.

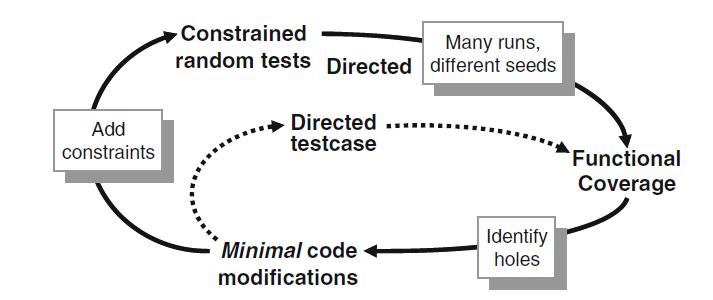
It takes longer time to ramp up compared to direct testing but reaches 100% coverage a lot faster.

There might be few cases the randomization can’t reach so they are written using direct testing.

This approach is the best for scalability and maintainability.

Finally, we can measure the verification progress by collecting coverage which is explained next.

# 6. Coverage

We measure how much functional coverage is achieved by adding some code in the testbench to monitor all stimulus going into the device and its reaction and response.

## 6.1. Code Coverage

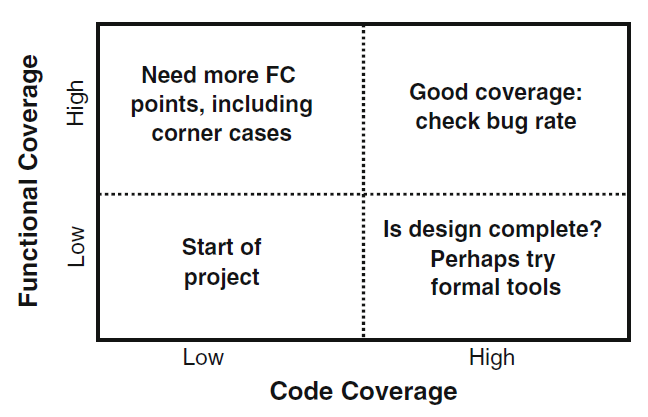
* Automatically extracted from design code
* Measures how much of the design code lines is covered
* Generated by the simulation tool and not the engineer

We need to measure code/line coverage to know identify holes in the verification plan itself.

100% code coverage means that the verification plan covered all the design code but it doesn’t give any indication about if the designer wrote RTL of all the features of the specs.

## 6.2. Functional Coverage

It is used to ensure the test meets all specs and features.

High functional coverage means the test met all the features listed in the verification plan. But it is no indication that the design is bug free, we check bugs later after simulation. Low code coverage means that we need to write more testcases.

# 7. References

* [SystemVerilog for Verification by Chris Spear](https://cutt.ly/G3RowBL)
* [ChipVerify](https://www.chipverify.com/systemverilog/systemverilog-tutorial)
* Si-Vision Academy SystemVerilog Tutorial