INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de Validez Oficial de Estudios de nivel superior según Acuerdo Secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976

Departamento de Electrónica, Sistemas e Informática

Maestría en Diseño Electrónico

Graphical Framework for Automatic Generation of Custom UVM Testbenches in SystemVerilog Applied for the Validation of a SerDes DUT

TRABAJO RECEPCIONAL para obtener el **GRADO** de

MAESTRO EN DISEÑO ELECTRÓNICO

Presentan:

César Fernando Limones Mora

Rogelio Rivas Villegas

Directores*: Manuel Salim, Alejandro Moreno*

Tlaquepaque, Jalisco. Febrero 2021

Acknowledgments

César would like to acknowledge this work mostly to the memory of his mother Silvia Mora who passed away from cancer before the culmination of this work was completed. Also, his father, brother, and girlfriend for their support and encouragement.

Rogelio would like to acknowledge his parents for supporting him throughout this journey, just like they have been doing in every single project in his life.

Both Rogelio and César would like to express the biggest gratitude to their tutors Manuel and Alejandro. They were a source of support, guidance, and encouragement. Last but not least, all the appreciation to Intel for all the support in resources that made the realization of this achievement possible.

Abstract

A novel graphical tool designed to assist Pre-Silicon validators in the creation of complete, functional, and compile-clean UVM testbenches is presented in this case study. A detailed description of the user-friendly interface is documented and demonstrated to auto-generate a validation environment template for the verification of an ALU and SerDes chip. The output obtained from the tool is later customized and optional sections are filled up to perform the full validation of the circuit.

For the SerDes DUT, this case study takes over from the work of the latest 2017 ITESO SerDes circuit design [1]. Both authors of this document worked on the 2016 iteration [2], [3] and are very familiar with the design, but this time instead of the actual design of the chip, the primary focus is how this new validation tool can be an essential asset to ensure the quality of the chip and to improve the efficiency of the verification process.

Table of Contents

Table of Figures

List of Tables

Chapter 1: Introduction

Over the last three decades, the semiconductor industry has undergone tremendous changes. As the industry moves forward into designing the latest, greatest, and fastest integrated circuits (ICs), design validation has become even more critical for SoCs to be successful. For increasingly complex digital designs, improved verification methodologies and tools had become a fundamental asset for all validation engineers.

To unify design and verification languages, SystemVerilog (SV) was introduced as an extensive enhancement to the IEEE 1364 Verilog standard. SystemVerilog has introduced many verification aides such as OOP concepts, assertion, and coverage to the verification environment.

Furthermore, to improve the reusability and automation, a uniform verification environment was required, and thus many standard methodologies have been widely used in the industry [4]. Such methodologies ensure that the testbench is layered and separated into individual independent files [5].

Currently, the most widely accepted methodology is Universal Verification Methodology (UVM), which is built on the verification constructs in SystemVerilog.

1.1. Problem Statement

Unfortunately, UVM is complex and usage is not always easy. The complexity of the methodology and the sheer size of the library make adoption difficult and challenging even for experts.

In the pre-Si verification field, they are not many practical exercises to help the students to understand how the UVM methodology works. UVM adopters can pass a hard time with how difficult is to generate UVM environments from scratch, there the vast number of classes and how they interact with each other can be very confusing.

1.2. Motivation

Developing testbenches in a standard modular and consistent manner has brought many benefits. In this case study, a tool aimed to address this quandary and accelerate the adoption and deployment of the methodology is proposed.

Currently, there is no software that can provide a graphical solution to create UVM testbenches similar to the one presented in this case study. When writing UVM testbenches, there are some tools than can be an aid to the user. For example, Eclipse DVT which allows the user to see graphically the environment that they are generating or Doulos´s *EasierUVM* [20] which can generate UVM testbenches by giving configurations files as input.

1.3. Research Goals

The goal of this research work is to create a tool that fulfills the necessities of validator engineers, students, and UVM enthusiasts that was stable and easy to use. The goal is achieved with the following objectives:

- To create a software to help with the creation of UVM testbenches.
- To proof that this tool grants major benefits by performing the validation of a simple design, for this case an ALU was chosen.
- To proof that this tool grants major benefits by performing Validation of a complex design, for this case the SerDes design from ITESO.
- To analyze the results of traditional UVM coding against using the files generated by the tool.

1.4. Contributions

The major contributions of this work include:

- Development of a graphical tool in C++ that generates complete UVM testbenches templates.
- Creation of a complete simple hierarchical UVM testbench that matches academic projects.
- Creation of a complete complex hierarchical UVM testbench that is similar to the used in the industry projects.

- Complete validation of an ALU design in UVM.
- Complete validation of the ITESO SerDes design.

1.5. Organization

The structure of the document is as follows:

- Chapter 1: This chapter covers the introduction.
- Chapter 2: This chapter provides the background of Pre-Silicon Verification Methodologies focusing on UVM methodology.
- Chapter 3: This chapter explains all the details related to the graphical UVM framework tool that was developed.
- Chapter 4: This chapter explains the testbenches generated for the ALU and the SerDes.
- Chapter 5: This chapter discusses the validation that was made for the ALU and the SerDes with the aid of the auto-generated output code from the tool.
- Chapter 6: This chapter comprises of the resultant simulations and findings from the tests and the tool implementation.
- Chapter 7: The conclusion and possible future work are briefly discussed in this chapter.

1.6. Definitions

- **Agent**: An abstract container used to emulate and verify DUT devices; agents encapsulate a driver, sequencer, and monitor.
- **Comma:** Special symbol in the 8b/10b encoding that are often used to indicate start-of-frame or end-of-frame conditions.
- **Component**: A piece of VIP that provides functionality and interfaces. Also referred to as a transactor.
- **Consumer**: A verification component that receives transactions from another component.
- **Driver**: A component responsible for executing or otherwise processing transactions, usually interacting with the device under test (DUT) to do so.

- **Element Group:** SW identifier to define an object polymorphic attributes and functions.
- **Element Type:** Attribute to map a UVM object with a canvas element.
- **Environment**: The container object that defines the testbench topology.
- **Export**: A transaction-level modeling (TLM) interface that provides the implementation of methods used for communication. Used in UVM to connect to a port.
- **Factory method**: A classic software design pattern used to create generic code by deferring, until run time, the exact specification of the object to be created.
- **Graphic Element:** C++ class to represent a TB component as a visual object.
- **Monitor**: A passive entity that samples DUT signals but does not drive them.
- **Port**: A TLM interface that defines the set of methods used for communication. Used in UVM to connect to an export.
- **QString:** QT Class used to provide a Unicode character string.
- **Request**: A transaction that provides information to initiate the processing of a particular operation.
- **Response**: A transaction that provides information about the completion or status of a particular operation.
- **Scoreboard**: The component used to dynamically predict the response of the design and check the observed response against the predicted response. Usually refers to the entire dynamic response-checking structure.
- **Sequence**: A UVM object that procedurally defines a set of transactions to be executed and/or controls the execution of other sequences.
- **Sequencer**: An advanced stimulus generator which executes sequences that define the transactions provided to the driver for execution.
- **Test**: Specific customization of an environment to exercise the required functionality of the DUT.
- **Testbench**: The structural definition of a set of verification components used to verify a DUT. Also referred to as a verification environment.
- **Transaction**: A class instance that encapsulates information used to communicate between two or more components.

- **UI:** The user interface (UI) is the point of human-computer interaction and communication in a device.
- **Virtual sequence**: A conceptual term for a sequence that controls the execution of sequences on other sequencers.
- **Widget**: A software widget is a relatively easy-to-use software application or component made for one or more different software platforms.

1.7. Conventions

The case study assumes that the reader has some knowledge of OOP, C++, verification testbenches, TLM, and UVM. It is not intended to go very deep in details other than those related to the UVM Framework tool.

- QT classes are described in purple and "`consolas`" font
- SV classes are written in "`courier`" font
- Graphic Element are referred starting with capital letter.
- UVM objects are referred with no capitalization.
- A subset of the full functions' suite is presented in the document. To see the full documentation of the program, please refer to the App Documentation in the References section.

1.8. Abbreviations

The following list documents the abbreviations used in this document.

API – Application programming interface
ASIC – Application-specific integrated circuit
AVM – Advanced Verification Methodology
CDV – Coverage driven verification
BFM – Bus Functional Model
DUT – Device Under Test
EDA – Electronic Design Automation

eRM – eLanguage Reuse Methodology

FIFO – First-in, first-out

FVM – Formal Verification Methodology

GUI – Graphical user interface

HDL – Hardware Description Language

HVL – High-level Verification Language

IP – Intellectual property

OOP – Object-Oriented Programming

OVM – Open Verification Methodology

PCI-E – Peripheral Component Interconnect Express

RTL – Register Transfer Level

RVM – Reference Verification Methodology

SoC – System-on-Chip

SV – SystemVerilog

SW – Software

TB – Testbench

TLM – Transaction-Level Modeling

UML – Unified Modeling Language

UVC – Universal Verification Component

UVM – Universal Verification Methodology

VIF – Virtual Interface

VIP – Verification IP

VMM - Verification Methodology Manual

Chapter 2: Pre-Si Verification Methodologies

In most engineering products, a significant amount of effort is spent on validation activities. Usually, 80% of the time scheduled in a project is dedicated to validation activities. Validation is defined by different authors, but usually, they agree on the main goal of performing tasks to verify that a product satisfies reference specifications, having correct functionality, and meeting customer needs [6].

2.1. SoC Life-cycle

During the development of a project, the cost of fixing a bug in a late-stage is much more than the cost of finding and fixing it in an earlier stage, such as the Pre-Si validation. This means that a bug found by any of the Pre-Si tasks might need only a few changes in the code and running a bunch of tests to ensure that anything else is broken. On the other hand, a bug in the Post-Silicon process can lead to a fatal fabrication error which could mean hundreds of millions of dollars are required to fix the issue [7].

That is why Pre-Si validation activities are a critical step in the SoC Life-cycle. A good verification effort gives confidence and enables the job of the consecutive stages of the development process.

2.2. Role of Verification

The main goal of Verification is defined around the Quality of the design. To improve and prove this metric there are a series of steps or activities that need to be performed, such as finding as many bugs as possible as quickly as possible, analyze performance, ensure target coverage is being hit, ensure correct functionality of the design and many other tasks.

This is only possible by doing a good planning stage, but also by having a robust Testbench and Validation infrastructure. If these stages are done correctly, the execution of the verification tasks can run smoothly.

2.3. Verification Methodologies

The continuous growth in the complexity of electronic design requires a modern, systematic, and automated approach for creating testbenches [6].

Various hardware verification languages (HVL) started to appear during the last decades. It was HVLs that enabled the transition from directed testing testbenches to constrained-random testbenches. When this adoption began, the existing validation techniques and procedures came short on supporting this new approach.

Before the verification methodologies came into the picture, each verification engineer used to write verification environments in their own way, there was no standard way to define any of the verification components. Even in the same companies and projects, there were variations in how verification testbenches were coded, resulting in very poor reuse.

The problems were there, how to standardize BFMs, checkers, base objects, etc. In an attempt to define how things should be done, and homogenize the industry, early frameworks alternatives started to emerge. Some of these are RVM (Reference Verification Methodology), eRM (e Reuse Methodology), URM (Universal Reuse Methodology), AVM (Advanced Verification Methodology), VMM (Verification Methodology Manual), and OVM (Open Verification Methodology).

While all these methodologies have their own value, and some of there were widely accepted with proven success on thousands of projects there is one that gradually started to dominate the verification landscape. Finally, it all came to UVM (Universal Verification Methodology), which is the most accepted methodology right now.

2.3.1. UVM

UVM stands for the Universal Verification Methodology, it is built on SystemVerilog constructs. UVM with SV supports constrained random, coverage-driven verification and provides a flexible and scalable infrastructure for testbench architectures.

UVM does not add any new keywords or capabilities to the SV language but uses existing capabilities to put together a powerful set of libraries. If a simulator supports the full SV

class-based capabilities, an engineer can run UVM without any additional licenses [8].

The UVM is a complete methodology that codifies the best practices for efficient and exhaustive verification. One of the key principles of UVM is to develop and leverage reusable components – also called UVM Verification Components (UVCs) [9].

UVM is an open-source SV base class library and is part of a standard by Accellera. As a standard, it is supported by all the major and minor simulator vendors.

2.3.2. Why UVM

UVM consists of a defined methodology for architecting modular testbenches, which means that the code will be consistent and uniform.

It´s been proved in several papers that UVM provides several advantages when compared to traditional verification, like such [22] and [23] as:

- Better Performance in terms of CPU time and memory usage.
- Much easier debugging
- Reusability of base classes, methods, environment
- Easier to synchronize the communication between verification components
- Very much structured testbench
- Fixed run flow because of UVM phases
- Easy to prepare the code
- Lesser time to build the testbench

With UVM, it is expected that all validation engineers follow the same guidelines and good practices when creating code. This can avoid many verification pitfalls by taking advantage of all the reuse that UVM brings to the table like re-using testbench components and stimulus within and across projects, development of Verification IP, easier migration from simulation to emulation, etc.

UVM knowledge is heavily spread around the globe, there is plenty of work going on into creating an infrastructure around UVM such as tools, editors, debuggers, graphical environments, and coding standards.

2.3.3. Drawbacks of learning UVM

The learning process of UVM is not easy, for anyone new to the methodology, the learning curve to understand all the details and the library is very steep. Before even trying to learn UVM, the verification engineer needs to be fluent in SV for verification HVL, which can be quite complex by itself. On top of that, learning UVM can be overwhelming, it is very challenging due to its approach and library complexity.

There are approximately 300 classes in the UVM library, and the documentation can have holes. In UVM there are many ways to do the same thing (in part due to the intrinsic polymorphism of the OOP) and that makes UVM particularly challenging for beginners to learn and use. Learning and using UVM is a very significant challenge and that can make it an obstacle for small companies and large companies alike.

2.3.4. UVM Class library

The UVM Class Library [10] contains the programming interface (API) that defines a standard for the creation, integration, and extension of UVM Verification Components (UVCs) and verification environments that scale from block to system. The library consists of base classes, utilities, and macros.

From the UVM class hierarchy in Figure 1, it can be seen that most of the classes in UVM are derived from a set of classes that are described below.

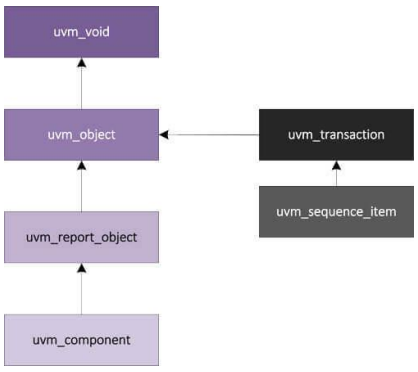


Figure 1 - UVM class hierarchy [11]

The *uvm_void* doesn´t have any purpose but serves as the base class for all UVM classes.

The *uvm_object classes´* main role is to provide a set of methods for common operations like create, copy, compare, print, and record. All major UVCs and transactions are derived

from this class.

The *uvm_report_object* provides an interface to the UVM reporting facility. All messages, warnings, errors issued by components go via this interface.

The u*vm_component* classes are intended to model permanent structural parts of the testbench. These are the basic building blocks for any verification environment. All major verification components such as monitors, scoreboards, drivers, etc. are derived from this class.

The *uvm_transaction* and *uvm_sequence_item* classes are intended to model stimulus and transactions. These class objects contain the actual data that is used to communicate.

Please note that the use of the *uvm_transaction* class as a base for user-defined transactions is deprecated. Its subtype, *uvm_sequence_item*, shall be used as the base class for all user-defined transaction types [11].

2.4. UVM Testbench and Verification Components

A UVM testbench is composed of reusable UVM-compliant universal verification components (UVCs). A UVC is an encapsulated, ready-to-use, and configurable verification environment intended for an interface protocol, a design sub-module, or even for software verification. Each UVC follows a consistent architecture and contains a complete set of elements for sending stimulus, as well as checking and collecting coverage information for a specific protocol or design [9].

UVM provides a comprehensive base class library supporting the construction and deployment of testbenches composed of reusable UVCs. It is explicitly simulation-oriented and intended to perform coverage-driven constrained random verification [12].

A typical UVM testbench is shown in Figure 2. The most common hierarchy goes like this, a Testbench top, which contains the UVM Test and the DUT. The UVM Test which in turn holds the sequences and one or more UVM Environments. Next, on the UVM Environment, the UVM scoreboards, UVM sequencers, and UVM Agents reside. Finally, the lower level of the

hierarchy, the UVM Agent, groups together the UVM Driver, UVM Monitor, and UVM Sequencers.

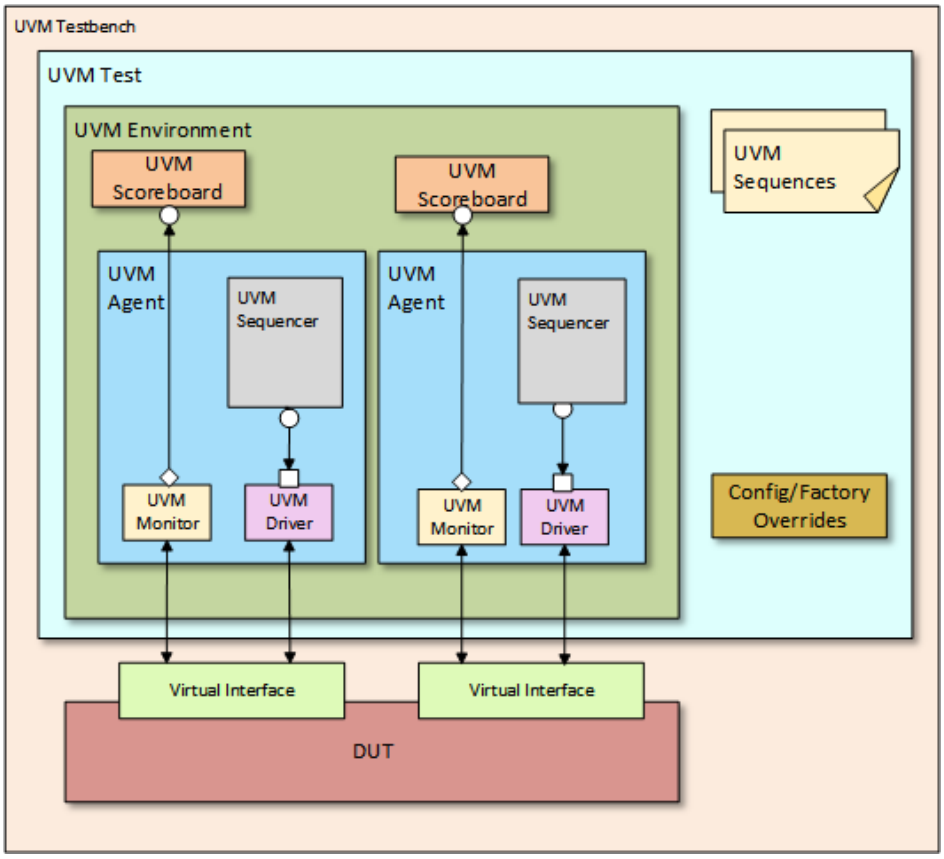


Figure 2 -Typical UVM testbench Architecture

The role of each testbench element is explained below [10], [13], [14], [15].

2.4.1. UVM Testbench TOP

The UVM testbench TOP typically instantiates the Design Under Test (DUT) module and interfaces. It is a static container to hold everything required to be simulated and becomes the root node in the hierarchy. It generates the clocks and resets and passes them to the interface handle; it also creates any DUT specific interface implementations. The UVM Test is invoked at run-time by the run_test() method, allowing the UVM Testbench to be compiled once and run with many different tests.

2.4.2. UVM Test

The UVM Test is the top-level UVC in the UVM testbench. It is a pattern to check and verify specific features and functionalities of a design. It typically performs three main functions:

- Instantiates the top-level environment

- Configures the environment
- Appy stimulus by invoking UVM sequences

2.4.3. UVM Environment

The UVM Environment is a component that groups together other UVCs that are interrelated. The most common components that are instantiated within the environment are UVM Agents, UVM scoreboards, or even other UVM Environments.

2.4.4. UVM Scoreboard

The UVM Scoreboard is where the checkers are coded. It will verify the functional behavior of the DUT and flag any errors. It usually receives transactions with data from the DUT interfaces via UVM Analysis ports and compares the data against expected results.
Typically, these expected results are obtained from a Reference model or a predictor that would mimic the functionality of the design.

2.4.5. UVM Agent

The UVM Agent is a component that encapsulate other UVCs that are dealing with a specific DUT interface. It typically includes a UVM sequencer to manage stimulus flow, a UVM Driver to apply stimulus on the DUT interface, and a UVM monitor to monitor the DUT interface.

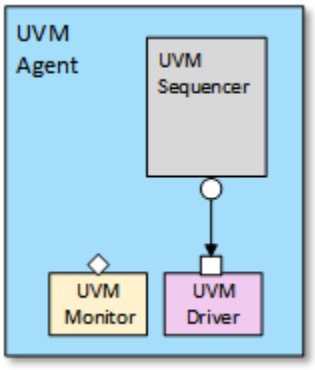


Figure 3 - Active Agent

The UVM Agent can be found with two modes of operations, active and passive. When the UVM Agent is in active mode shown in Figure 3, it can generate stimulus, on the other side, when it is in passive mode in Figure 4, it will only sample the interface without controlling it.

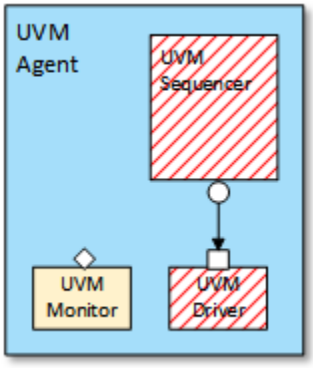


Figure 4 - Passive Agent

2.4.6. UVM Sequencer

The UVM Sequencer serves as an arbiter for controlling the flow of request and response sequence items from one or multiple sequences and sends it to the Driver for execution.

The base class is parameterized by either request or response item types, although by default the response and request types are the same, it can be specified by the user.

A request type (REQ) is used to send a request to the Sequencer to send sequence_item to the driver. While a response type (RSP) is used as a response from the driver when it completes a particular operation.

2.4.7. UVM Sequence

The UVM Sequence is an object that contains a behavior for generating stimulus. UVM Sequences are not part of the component hierarchy. A sequence generates a series of UVM Sequence Items that are executed by an assigned sequencer which then sends them to the driver. Hence, sequences make up the core stimuli of any verification plan.

2.4.8. UVM Driver

The UVM Driver receives individual UVM Sequence Item transactions that are obtained from the UVM Sequencer and then drives them to a particular interface of the design. It converts transaction-level stimulus into pin-level stimulus.

There is a driver/sequencer handshake mechanism that allows the driver to get a series of transaction items from the sequence and respond to the sequence after it has finished

driving the given item and continue to the next one.

2.4.9. UVM Monitor

The UVM Monitor is a passive component that samples the DUT interface and collects the information obtained in transactions that are sent out to the rest of the UVM Testbench for further analysis.

The UVM Monitor has access to the DUT signals via a virtual interface handle and it also has a TLM analysis port to broadcast the sampled transactions. The overall functionality should be limited to basic monitoring that is always required and any other processing on the transactions should be delegated to dedicated components connected to the monitor´s analysis port.

2.4.10. UVM Sequence Item

The sequence-item are UVM objects that consist of data fields required for generating the stimulus. To generate the stimulus, most of the data members in sequence items are generally defined as random using ‘rand’ or ‘randc’ type and generally can have constraints defined.

2.4.11. UVM Config Object

Configuration objects are an efficient, reusable means for organizing configuration variables. In a typical testbench there will generally be several configuration objects, each tied to a component. A configuration object is created as a subclass of uvm_object to encapsulate all related configuration variables for a given branch of the testbench structural hierarchy. There may also be a single, additional configuration object to hold global configuration variables. Each of the configuration variables within a configuration object may be declared as rand and consequently the configuration object may be randomized [16].

2.4.12. UVM Subscriber

Subscribers are basically listeners of an analysis port. They subscribe to a broadcaster and receive objects whenever an item is broadcasted via the connected analysis port.
This class is particularly useful when designing a coverage collector that attaches to a monitor.

2.4.13. UVM Phases

Each of the UVM components derived from the *uvm_component* class is aware of the phase concept. Each component goes through a pre-defined set of phases and cannot proceed to the next phase until all components finish their execution in the current phase. UVM Phases act as a synchronizing mechanism in the simulation´s life cycle.
The overall picture of the UVM Phasing is shown in Figure 5:

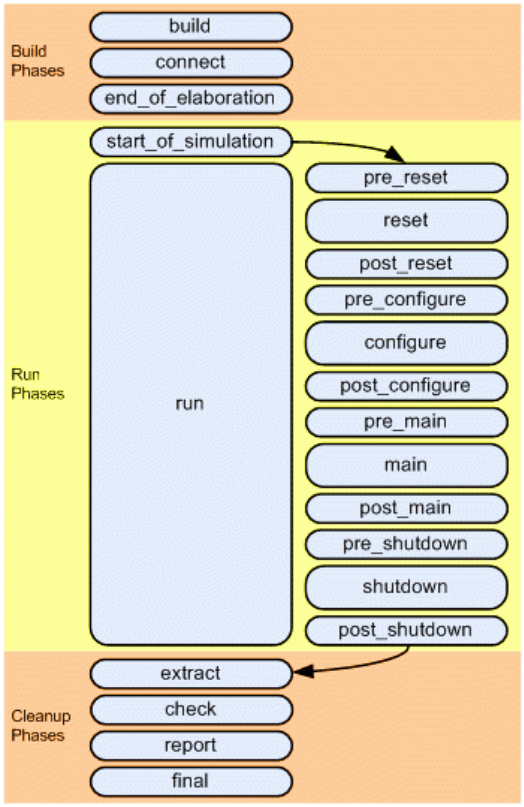


Figure 5 - UVM Phases [16]

The UVM Phases divided into three categories:

- Build Phases which contain 3 sub-phases
- Run Phases which contain 13 sub-phases
- Clean-up Phases which contains 4 sub-phases

The Build Phases are executed at the start of the UVM Testbench simulation and their overall purpose is to construct, configure, and connect the Testbench component hierarchy. All the build phase methods are functions and therefore execute in zero simulation time.

On the Run Phases, the UVM testbench stimulus is generated and executed during the run time phases which follow the build phases.

The Clean-up Phases extract information from Scoreboards and Functional Coverage Monitors to determine if the test has passed and/or reached its coverage goals. The Clean-up Phases are implemented as functions and therefore take zero time to execute [16].

2.4.14. Transaction-Level Modeling (TLM)

Transaction-Level Modeling (TLM) is a modeling style for building highly abstract models of components and systems. In this scheme, data is represented as transactions in which UVM components can communicate via special ports called TLM interfaces. This kind of abstraction has been shown to be necessary for today´s verification.

UVM provides a set of transaction-level communication interfaces and channels that can be used to connect components at the transaction level such that data packets can be transferred between them. Such that a component may communicate, by way of its interface to any other component which implements that interface [9].

The use of TLM interfaces isolates each component from changes in other components through the environment. This approach promotes reusability and flexibility by allowing any component to be swapped for another, as long as they have the same TLM interface.

Chapter 3: Graphical UVM Framework

UVM is very powerful and very flexible, but it is also very complex. It is built on object-oriented programming (OOP) concepts, including classes with methods that operate on data.

One metric for the complexity of UVM is the size and depth of the library. The latest release from *Accellera*, version 1.2, has a reference manual of 938 pages and a user's guide with 190 pages. Learning by heart a subset of constructions required to develop a working testbench is a significant hurdle for new adopters, and even be difficult to experienced users. A better approach is needed to take advantage of the benefits of UVM more easily.

In the paper, "A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities" [17], it is mentioned that one of the drawbacks of UVM is that: "it is very complicated, so it does not make sense with small projects". The UVM Framework tool address this problem by highly reducing the complexity so UVM can be used hastily for even small projects.

The continuous growth in the complexity of electronic designs requires a modern, systematic, and automated approach to creating testbenches [6]. The tool allows the engineer to worry more about the specifics of the validation IP and release some burden on the UVM constructs. Making it more efficient to write and debug the validation environment. The expectation is to reduce the learning time for new validators and save time and effort for common tasks. While also adopting a graphic appealing tool that anyone can use.

3.1. Background

Exhaustive research was done to find out the best approach to create this tool. One option was to create the code in java from scratch, but this would take a big effort. Another one was to take advantage of an existing open code repository that had some similarities to the necessities of this project.

It was decided to go with the latter to create the tool in order to take advantage of the infrastructure already developed for a similar software and modify as needed. This software is called *WiRED* Panda and it is a framework created to simulate electronic digital circuits and then generate Arduino code directly from the schematics. It is an open software developed in QT 5.10 by the engineers Davi Morales, Fábio Cappabianco, Lucas Lellis, and Rodrigo Torres, to whom the authors would like to express their complete gratitude for making this possible.

Parts of the graphic infrastructure from *WiRED Panda* were reused to create the custom tool that presented in this project, with the difference that this will generate UVM testbenches instead of electronic designs.

WiRED Panda is a free software designed to help students to learn about logic circuits and simulate them in an easy and friendly way [18].

The main features of the software are:

- Works on Windows, OSX, and Linux
- Real-time logic simulation.
- User-friendly interface.
- It is intuitive and easy to use.
- Export projects to Arduino

An example of what *WiRED Panda* can create is shown in Figure 6.

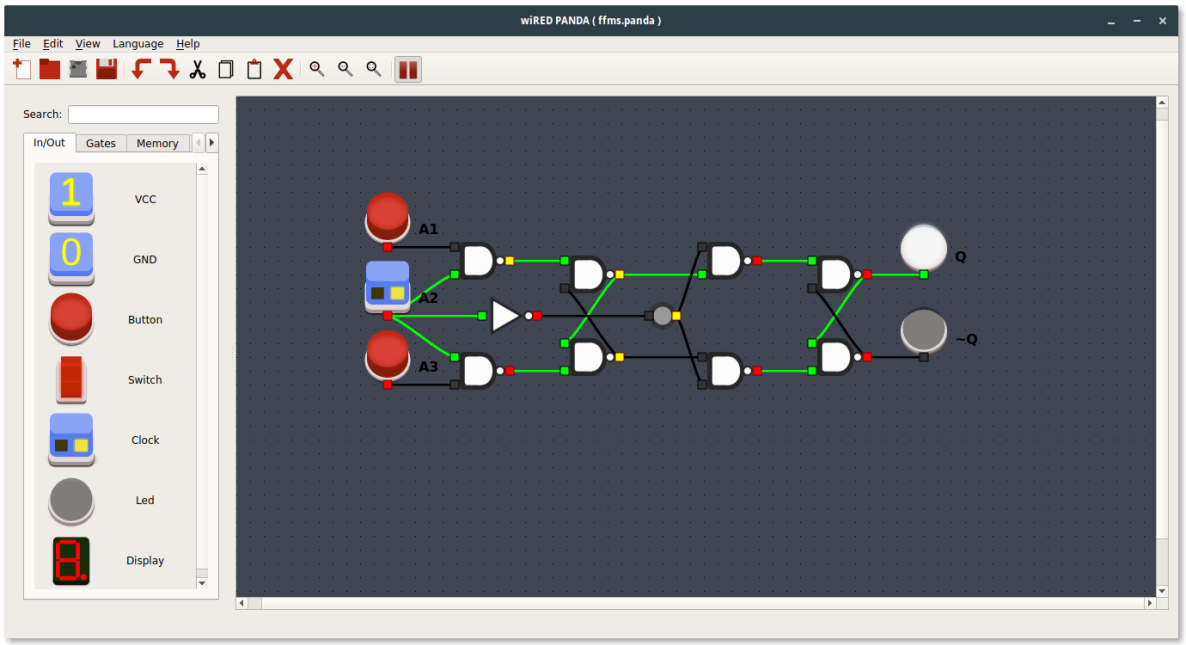


Figure 6 - WiRED Panda circuit

Several electronics projects have been developed at The **Federal University of São Paulo** (Portuguese: *Universidade Federal de São Paulo, UNIFESP*) using this tool and show clear evidence of how powerful this tool is. Some examples are:

- Genius Game
- Snake Game
- Panda Pong Game
- Crossing Led Game
- Battleship Game
- Asteroids Game
- Minefield Game
- Mastermind Game
- Guitar Panda
- Bowling Game
- Calculator
- Elevator controller
- Tetris Game

All the details of this software can be found in wiredpanda.org.

3.2. QT and C++

Qt is a free and open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android, or embedded systems with little or no change in the underlying codebase while still being a native application with native capabilities and speed [19].

Coding in QT offers many benefits:

- Allow multi-platform development
- Big support for the tool.
- Have commercial licenses but also free Open Source.

For the code generation tool, the QT Creator IDE offers a C++ compiler. This Object-Oriented Programming language is chosen to develop the code.

3.3. Implementation

In this section, the implementation of the program is explained in detail. However, not all the functions and attributes are described in the class or function tables. Only the more relevant features are included in this document.

3.3.1. Requirements

The main objective of this project is the development of didactic software for the creation of UVM testbenches.

The specific objectives of this project are:

- Create a user-friendly graphic interface for the edition and manipulation of UVM components.
- A code generation based on the graphic elements presented in the tool canvas.
- Smart connections between the UVM elements, having constraints between the possible connections.
- Set general requirements for the tool.

- Comply with the specific requirements for each UVM component.
- Creation of new functions specifics to UVM.

3.3.2. Specification

Following specifications are extracted based on the requirements and tools to be used for the framework development:

- The program will run on Windows and UNIX operating systems.
- Executable files are generated using QT Creator IDE.
- UI Forms are generated using QT Creator IDE.
- Source code is written in C++.
- The code generator output files are generated in SystemVerilog language.
- The code generator output files are compile-clean for VCS and QuestaSim compilers.
- The generated code is compliant with the UVM 1.2 version of the methodology defined by *Accellera*.

3.3.3. Use Cases

The validator is only one entity that interacts with the tool. However, the level of expertise of each user defines the subset of use cases that will be required to generate an adequate SV code using the framework.

Use cases identified are:

- Add a VC to the scene.
- Add a VC inside another VC.
- Remove VC from the scene.
- Update position of an existing VC in the scene.
- Create connections between two or more VCs.
- Remove an existing connection.
- Add connectivity ports in a VC.
- Modify port type and name.
- Remove connectivity ports in a VC.
- Change properties of a VC.
    - Class name
    - Parent class

    - Config object
    - Associated sequence.
    - Associated sequence item.
    - Signal list.
    - Covergroups list.
- Generate SV code.
- Save a project.
- Open a project.
- Copy and paste VCs.

3.3.4. GUI description

Graphical UVM framework main window layout consists of four sections: utilities area, component pane, scene, and code preview area. These sections are QT forms and their corresponding C++ classes that are instantiated in the main Class called `MainWindow` that extends from the QT type `QMainWindow`.

Class:	**MainWindow** : public QMainWindow
Description:	
Main Class that is called at the execution of the program. Extends from the QT Class QMain Window. Instantiates the main QT Form and the Editor class to modify the scene. Calls the SV Exporter action.	
Attributes:	Ui::MainWindow *ui Editor *editor;
Functions:	bool **ExportToSV** (QString fname);

Table 1 - MainWindow Class

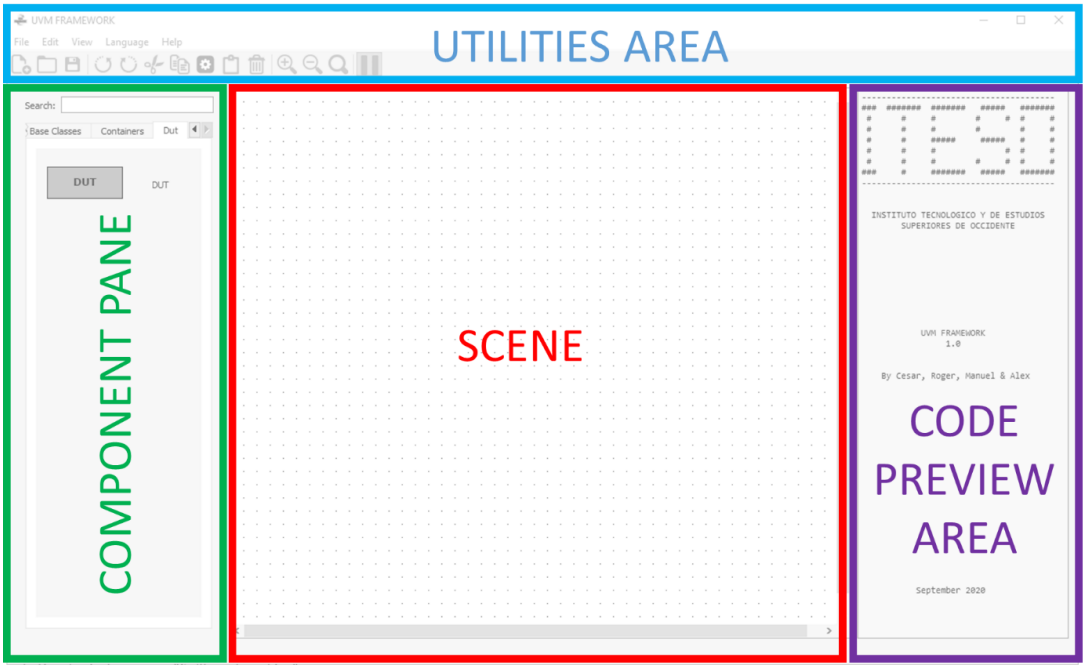


Table 2 - Pane Definition

The utilities area contains the main functions required to set up a project and the basic functionality of the program. In this section, the user can create new projects, set up preferences, adjust the view, change the language, and open the help menu.

The most relevant function of this menu is the invocation of the code generator to export the graphic view to compilable SV code. This action is a call to a file directory generator and the creation of the files of every component that is in the scene. This function is part of the MainWindow Class and it returns a Boolean in case the code is generated successfully.

Function:	bool **ExportToSV** (Qstring fname);
Description:	
This function calls the Code Generator functions to get the SV code from each element in the scene. It generates the directory, SV files, and corresponding package.	
Arguments:	Qstring with the path of the directory to generate the files.

Table 3 - ExportToSV function

More details on the Code Generator functions are explained in section 3.3.10.

This taskbar also contains a set of tools to manipulate the Graphic Elements and the scene,

such as create new a file, open an existing file, save a file, rotate elements, copy elements, cut elements, paste elements, show properties menu, remove an item, and adjust the view.

Class:	**ElementFactory** : public Qobject
Description:	
Left pane Class that obtains the initial information and functions to create an element and place it in the scene. It is also responsible for creating connections between elements.	
Attributes:	None.
Functions:	QPixmap **getPixmap**(ElementType type); GraphicElement* **buildElement**(ElementType type, Editor *editor, QgraphicsItem *parent = 0); QNEConnection* **buildConnection**(QGraphicsItem *parent = 0);

Table 4 - ElementFactory Class

The component pane left widget is also defined as the Element Factory. It is the first source to select and drag a component to the scene. Different tabs are added in this menu based on its UVM or SV attributes.

The classification of the components is:

- SystemVerilog modules: DUT and Interface shown in Figure 7.

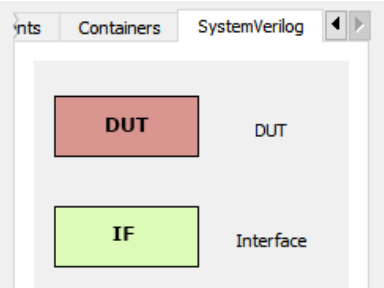


Figure 7 - SV modules

- Container elements: Top, Test, Env, and Agent shown in Figure 8.

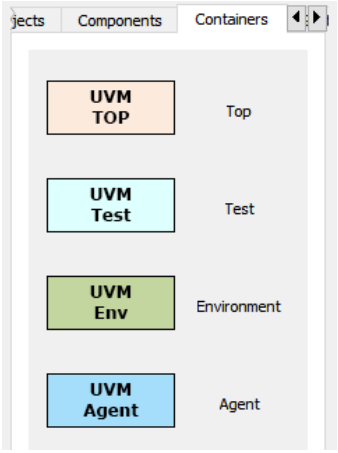


Figure 8 - Container Elements

- UVM component elements: Scoreboard, Subscriber, Monitor, Sequencer, and Driver shown in Figure 9.

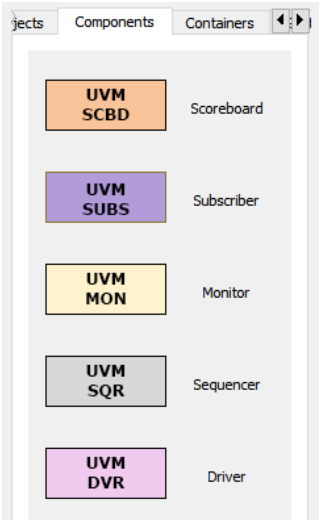


Figure 9 - UVM components

- UVM object elements: Config, Sequence, and Sequence Item shown in Figure 10.

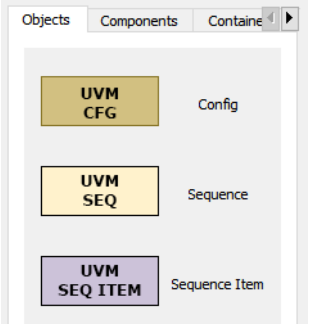


Figure 10 - UVM Objects

The central section of the GUI is the scene. This is the canvas where the UVM TB diagram is drawn. This section is essential because it is aware of all the components that are in the diagram and the connections between each other. It also registers the element positions and calls the painting methods of all the elements.

Class:	**Scene**: public QGraphicsScene
Description:	
Scene Class is a canvas that draws elements, ports, and connections. It allows element interactions and modifications from the User. Is aware of all the elements that are in the diagram.	
Attributes:	Canvas dimensions.
Functions:	QVector< GraphicElement* > **getElements**( );

Table 5 - Scene Class

Finally, the code preview area section is a text viewer that shows the SV code for a specific element.

Class:	**QTextBrowser** : public QTextEdit
Description:	
Text viewer that obtains the code for the selected element and displays a preview of the code to be generated.	
Attributes:	QString _elementCode
Functions:	void **setText**(const QString &text);

Table 6 - QTextBrowser Class

3.3.5. UVM models

All the VCs used in a UVM testbench extend from the common class UVM object. As the name suggests, they are SV special classes with a specific functionality. This modular approach allows the User to have both flexibility and scalability during the building process of the verification environment. This advantage is also use by the tool to model the VCs.

The UVM methodology employs a layered, object-oriented approach to testbench development that allows “separation of concerns” among the various team members. Each component in a UVM testbench has a specific purpose and a well-defined interface to the rest of the testbench to enhance productivity and facilitate reuse.

When these components are assembled into a testbench, the result is a modular reusable verification environment that allows the test writer to think at the transaction level, focusing on the functionality that must be verified, while the testbench architect focuses on how the test interacts with the Design Under Test (DUT) [21].

In UVM, there are many different types of classes, however, there are some objects that at the very least are needed to create a complete UVM environment. Even though there are many more classes and possibilities when designing a validation environment, the ones selected were considered as the most relevant for a complete design and were added UVM framework tool. With these, the user can create a complete environment from basic to rather complex designs, these are:

- Agent
- Interface
- Driver
- Environment
- Monitor
- Subscriber
- Scoreboard
- Sequence item
- Sequencer
- Sequence
- Config
- TB Top
- DUT
- Test

All these elements are described in detail in the following sections. However, all of them extend from a common class called Graphic Element. This class is the wrapper to enable all the QT capabilities to create graphic items in the scene.

Since UVM is all about configurability, different configuration options and properties have been added to each of these objects that can be accessed by right-clicking on them. In the following lines, each element will be aborded in detail.

3.3.6. Graphic Element

A common class for all the UVM elements is required to facilitate the usage of the QT functions. This parent class is the Graphic Element class. The main purpose of this Class is to set up basic graphic attributes, paint methods, but also create common methods for similar UVM objects.

Class:	**GraphicElement** : public QGraphicsObject
Description:	
Parent class of all the UVM elements required for a TB creation. It handles graphic attributes and virtual functions to be implemented in child classes. Assigns a UVM element type and group definition (container or element only). Stores the UVM object attribute, name of the element SV class, parent class (in case of customized inheritance), and instance. Handles the number of ports and connections associated with a UVM element.	
Attributes:	virtual ElementType ***elementType*** virtual ElementGroup ***elementGroup*** QString m_uvmType QString m_labelClass QString m_labelInstance QString m_inheritance QList<QNEConnection*> m_uvmConnections
Functions:	virtual void ***paint( )*** void **addUvmConnection**(QNEConnection* newConnection);\ virtual void ***updatePorts***( );

Table 7 - GraphicElement Class

3.3.7. Containers

The UVM Containers are a subset of UVM elements in which other objects can be added inside. Consider these internal elements as instances within the class. The containers included in the framework are:

- Top
- Test
- Environment
- Agent

Top

The top-level hierarchy of the environment is generated in “<dut_name>_tb_top.sv”. It is the top container of the testbench. It is a container that can only hold inside the DUT, the interface, and the Test. This file consists of the following:

- Clocking mechanism for the design.
- Reset mechanism.
- Virtual interface instantiation and registering.

- Wave dump
- Running the test. A test is usually started within testbench top by a task called run_test. This global task should be supplied with the name of the user defined UVM test that needs to be started. If the argument to run_test is blank, it is necessary to specify the test name via command-line to the simulator using +UVM_TESTNAME.
- This component is 100% autogenerated by the tool.
- It has the configuration options to change the frequency of the clocking mechanisms and to optionally choose if “finish on completion” is added after the test run. Reset active state is also configurable.
-

Class:	**Top:** public GraphicElement
Description:	
UVM Container that models a SystemVerilog TB top module. It can instantiate DUT, Interface, and Test elements.	
Attributes:	ElementType::TOP ElementGroup::CONTAINER int m_clockCycle int m_resetTime
Functions:	void **addInterfaceToContainer**() ; void **addDUTToContainer**(); void **addTestToContainer**();

Table 8 - GraphicElement (Container) Class

Test

A testcase is a pattern to check and verify specific features and functionalities of a design. Instead of writing the same code for different testcases, the entire testbench is put into a container called an “environment” and use the same environment with a different configuration for each test. Each testcase can override, tweak knobs, enable/disable agents, change variable values in the configuration table, and run different sequences on many sequencers in the verification environment.

So, for the tool, a container can only hold an environment and different sequences. Config elements are also added at this level.

Class:	**Test:** public GraphicElement
Description:	
UVM Container that models the UVM Test component. It can instantiate Config, Sequence, and Env elements. The Config and Sequence elements are configurable.	
Attributes:	ElementType::TEST ElementGroup::CONTAINER QString m_config QString m_sequence
Functions:	void **addSequenceToContainer**() ; void **addEnvToContainer**(); void **addConfigToContainer**();

Table 9 - Test Class

Env

Element that models an UVM Environment. This container contains all the UVM component models that are required to inject stimulus, add observability, and drive signals. It also contains scoreboards and checking mechanisms. Coverage collection is also performed inside this component.

Class:	**Env:** public GraphicElement
Description:	
UVM Container that models UVM environment component. It can instantiate Scoreboard, Subscriber, Agent, and child Env elements. The Config object is configurable.	
Attributes:	ElementType::ENV ElementGroup::CONTAINER QString m_config
Functions:	void **addScoreboardToContainer**() ; void **addEnvToContainer**(); void **addAgentToContainer**(); void **addSubscriberToContainer**();

Table 10 - Env Class

Agent

An agent encapsulates a sequencer, driver and monitor into a single entity by instantiating and connecting the components together via TLM interfaces. The configuration options added to the agent are:

- A knob to select the agent´s type, active or passive.
    - Active agent: instantiate all three components: sequencer, driver, and monitor. Enables data to be driven to DUT via driver.
    - Passive agent: only instantiate the monitor. Used for checking and coverage only. Useful when there´s no data item to be driven to DUT.

Class:	**Agent:** public GraphicElement
Description:	
UVM Container that models UVM agent component. It can instantiate Subscriber, Monitor, Sequencer and, Driver elements. The Config object is configurable. The Passive and Active modes are configurable.	
Attributes:	ElementType::AGENT ElementGroup::CONTAINER QString m_config QString m_activeMode
Functions:	void **addSequencerToContainer**() ; void **addDriverToContainer**(); void **addMonitorToContainer**(); void **addSubscriberToContainer**();

Table 11 - Agent Class

3.3.8. Elements

These elements are the graphic models of the rest of the UVM objects and components used in the validation. Not all of them can be added to every container.

Sequence

The Sequence is a special kind of element that behaves as a container in a special case. This element can only include Sequence Item child items but does not instantiate any internal object. Therefore, it is considered an element and not a container.

The Sequence is created at the Test hierarchy.

Class:	**Sequence:** public GraphicElement
Description:	
An Element that models the UVM sequence object. It can instantiate Sequence Item elements. The Config object is configurable.	
Attributes:	ElementType::SEQUENCE ElementGroup::OBJECT QString m_config QString m_seqItem
Functions:	void **addSequenceItemToContainer**() ;

Table 12 - Sequence Class

Sequence Item

Transactions between VCs are performed using sequence items. A Sequence Item element models the graphic representation of these transactions. The signal list is fundamental in this element, but also the definition of random fields and constraints.

Sequence Item is created at the Sequence hierarchy.

Class:	**Transaction:** public GraphicElement
Description:	
An Element that models UVM sequence item object. The Config object is configurable. The signal list with random fields is configurable. UVM utils for random fields is provided.	
Attributes:	ElementType::TRANSACTION ElementGroup::OBJECT QString m_config QList<QString> m_signalList
Functions:	None.

Table 13 - Transaction Class

Config

The configuration objects in UVM are an intrinsic feature of the methodology. The model for this object is called Config. This element is generic and is a wrapper for the User to add configurable parameters in the design. The Config is created at the Test hierarchy.

Class:	**Config:** public GraphicElement
Description:	
An Element that models UVM configuration object.	
Attributes:	ElementType::CONFIG ElementGroup::OBJECT
Functions:	None.

Table 14 - Config Class

Scoreboard

The checking and comparison between the expected and actual behavior of the DUT is mainly performed in a Scoreboard component. The model for the component receives transactions from other components. The number of ports is customizable. The Scoreboard is created at the Env hierarchy.

Class:	**Scoreboard:** public GraphicElement
Description:	
An Element that models UVM Scoreboard component. The Config object is configurable.	
Attributes:	ElementType::SCOREBOARD ElementGroup::COMPONENT QString m_config QString m_seqItem
Functions:	None.

Table 15 - Scoreboard Class

Subscriber

The approach to integrate coverage capabilities into the tool is applied at the Subscriber model. This element is a generic wrapper for a wide variety of tasks. However, the properties menu provides a field to generate the template code for covergroups.

Subscriber is created at the Agent hierarchy.

Class:	**Subscriber:** public GraphicElement
Description:	
An Element that models UVM subscriber component. The Config object is configurable. The Covergroup list is optional.	
Attributes:	ElementType::SUBSCRIBER ElementGroup::COMPONENT QString m_config QList<QString> m_covergroups
Functions:	None

Table 16 - Subscriber Class

Monitor

Observability at the interface level of the TB is performed using a Monitor element. This model is typically connected to the DUT interface and forwards transactions to the rest of the environment components.

The Monitor is created at the Agent hierarchy.

Class:	**Monitor:** public GraphicElement
Description:	
An Element that models UVM monitor component. The Config object is configurable. The Sequence Item is configurable.	
Attributes:	ElementType::MONITOR ElementGroup::COMPONENT QString m_config QString m_seqItem
Functions:	None

Table 17 - Monitor Class

Sequencer

The association and handling of the sequences injected into the TB are managed by the Sequencer model. It does not have any input ports. However, it has sequence objects associated with it. These sequences are started by the Test element.

Sequencer is created at the Agent hierarchy.

Class:	**Sequencer:** public GraphicElement
Description:	
An Element that models UVM sequencer component. The Config object is configurable. The Sequence to be executed is configurable.	
Attributes:	ElementType::SEQUENCER ElementGroup::COMPONENT QString m_config QString m_sequence
Functions:	None

Table 18 - Sequencer Class

Driver

Driving of the signals obtained from the virtual interface is made in the driver component. The Driver element models this object by receiving and sending transactions between the DUT and the sequencer. Graphically its ports are represented as inputs, however, the code allows to handle incoming and outgoing transactions.

It can be configured in requester or responder mode.

Driver is created at the Agent hierarchy.

Class:	**Driver :** public GraphicElement
Description:	
An Element that models UVM driver component. The Config object is configurable. The driving mechanism (REQ or RESP) is configurable.	
Attributes:	ElementType::DRIVER ElementGroup::COMPONENT QString m_config QString m_seqItem bool m_isResponder
Functions:	None

Table 19 - Driver Class

Interface

The connection between the DUT and TB is made via the interface module. An Interface element models this connection and sets all the signals required to achieve complete connectivity with the DUT, Monitor, and Driver elements. This is not a UVM component but

a SystemVerilog module.

The Interface is created at the TOP hierarchy.

Class:	**Interface:** public GraphicElement
Description:	
An Element that models SystemVerilog interface module. The signals list is optional, but strongly recommended.	
Attributes:	ElementType::INTERFAZ ElementGroup::SV_MODULE QList<QString> m_signalList
Functions:	None

Table 20 - Interface Class

Dut

A wrapper for the DUT module is modeled using a DUT element. This is an empty module that is handled as a dummy instance of the RTL code. This file is the only one that needs to be replaced with the User actual DUT file.

The DUT is created at the TOP hierarchy.

Class:	**Dut :** public GraphicElement
Description:	
An Element that models SystemVerilog DUT module. The Signals list is optional, but strongly recommended.	
Attributes:	ElementType::DUT ElementGroup::SV_MODULE QList<QString> m_signalList
Functions:	None

Table 21 - DUT Class

3.3.9. Connections

Interactions between the VCs are well defined in the UVM methodology. Thus, support for connectivity between elements is included in the tool.

The port class is a QT graphic item that is embedded in every element. This item is a structure that stores input and output descriptors that allow or restrict interactions between all the elements in the scene.

Class:	QNEPort(QGraphicsItem *parent) : QGraphicsPathItem(parent)
Description:	
A Graphic item that stores descriptors of the port behavior. Contains a pointer to its parent element.	
Attributes:	int ***type***
Functions:	bool **isOutput**( ); bool **isInput**( );

Table 22 - QNEPort Class

There are six different types of ports included in the tool:

- Input: Generic port that serves as an endpoint for a connection.
- Output: Generic port that serves as the start point for a connection.
- Analysis port: Output port. UVM analysis port to send transactions.
- Analysis import: Output port. UVM analysis port to send transactions. Receiving port should implement a write method.
- Analysis export: Input port. UVM analysis export to get transactions.
- Analysis TLM FIFO: Input port. UVM analysis FIFO to queue incoming transactions.

Port type and name can be changed from the scene as shown in Figure 11.

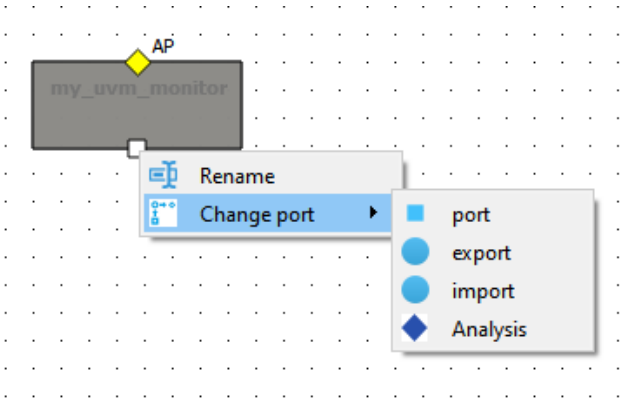


Figure 11 - Port typing change

A connection determines its state depending on the port types that the user tries to connect. The connection is marked as Valid if all the checks pass and invalid if any of the restrictions is found.

Furthermore, a connection is also rejected if the parent elements of the candidate ports are not compatible with the UVM methodology or the interaction between both elements is not aligned to the best-practices and congruency of the TB.

Restrictions are listed as follows:

- A Connection is forbidden between two outputs.
- A Connection is not allowed between two input ports.
- Analysis imports can be connected to Analysis Export ports or Analysis TLM FIFOs.
- Monitor outputs can be connected to Scoreboards and Subscribers.
- Sequencer outputs can be connected to Drivers.
- Interface ports can be connected to Drivers or Monitor elements.
- DUT ports can only be connected to Interface elements.
- Any other combination is not allowed.

If the candidate ports satisfy the conditions, a new QNEConnection object is created and assigned to the port's parent element.

Class:	**QNEConnection**: public QGraphicsPathItem
Description:	
Graphic Item that stores the attributes of an interaction between two elements. Contains graphic attributes to draw the connection in the scene. Stores logical information to associate each created connection with a common parent to add the corresponding code. Has pointers of the ports interacting in the connection attempt. Contains a pointer to its parent element.	
Attributes:	QNEPort* **port1**( ) const QNEPort* **port2**( ) const Status m_status
Functions:	void **setStatus**(const Status &status) void **setTree1**(QList<GraphicElement*> tree1); void **setTree2**(QList<GraphicElement*> tree2);

Table 23 - QNEConnection Class

3.3.10. SystemVerilog Code generator

UVM classes and methodology are a complex set of tools, especially for early adopters. UVM itself tries to address this issue by adding standard libraries and building blocks as the one described in the previous section. However, there are still some complications in the code implementation that are related to the different ways in which a single operation can be performed using SV. The methodology supports all this flexibility, but this implies a lack of cleanness and maintainability in the code.

There are some coding guidelines and best-known methods to create TBs. The code generator implemented in this case study follows all of them to create a template and automatically generate most of the lines of uniform, readable and well documented files. In some cases, the template itself represents 100% of the file content.

The code generator is intended to generate everything needed to get a UVM environment running. The tool is flexible enough to allow the user to generate the code from a single component to a whole testbench. The user just populate the canvas, and the code will be auto-generated on the go. Each connection, each signal, every change in the canvas will be automatically reflected on the code in the right window. And when the user is satisfied with the designed testbench, the tool allows the exportation of all the files to SV.

The final auto-generated code will be compile-clean and the only pending step would be for each user to gradually populate the missing pieces specific to each design to create a complete code.

The code generator consists of different functions that extract the element features, based on the properties set by the User, but also taking into account its interaction with other elements in the scene, expected behavior, and hierarchy level in which the object/module is instantiated. Several algorithms are put together to be aware of the dependencies of each element’s code.

Four base functions inspect each Graphic Element and generate parts of the code, taking branch decisions to add or exclude sections. This is performed following the best-known techniques and maintaining consistency throughout the TB.

Function:	QString ***populateHeader***()
Description:	
This function returns a QString with the header of the element. Obtains the instances and signals required for the current element.	
Arguments:	Graphic element pointer. Child elements pointers. Signal list.

Table 24 - QString populateHeader function

Function:	QString ***populateBuild***()
Description:	
This function returns a QString with the UVM build phase of the element. Obtains the config objects, virtual interfaces, instances, and creates the objects of the child items of the current element. It oversees the assigned statements for SystemVerilog modules.	
Arguments:	Graphic element pointer. Child elements pointers. Config object settings.

Table 25 - QString populateBuild function

Function:	QString ***populateRun***()
Description:	
This function returns a QString with the UVM run phase of the element. Obtains the sequences and starts them if required. It oversees the procedural blocks for SystemVerilog modules.	
Arguments:	Graphic element pointer. Associated Sequence pointer. Associated Sequence Item pointer.

Table 26 - QString populateRun function

Function:	QString ***populateConnect***()
Description:	
This function returns a QString with the connect phase of the element. Obtains the connections and ports that need to be added in the code.	
Arguments:	Graphic element pointer. Connections list with ports and graphic elements.

Table 27 - QString populateConnect function

A wrapper function arbitrates between each function call and picks the correct execution order depending on the element type detected.

Function:	QString ***populateCode***()
Description:	
This function returns a QString with the full code of the element. Enables function execution depending on the element type, connections, and properties.	
Arguments:	Graphic element pointer.

Table 28 - QString populateCode function

Furthermore, these functions generate a compilable code if all the properties are filled accordingly as described in section 3.4 depending on the element being manipulated.

3.3.11. UML Diagram

The modeling of the tool will use the following UML diagram in Figure 12.

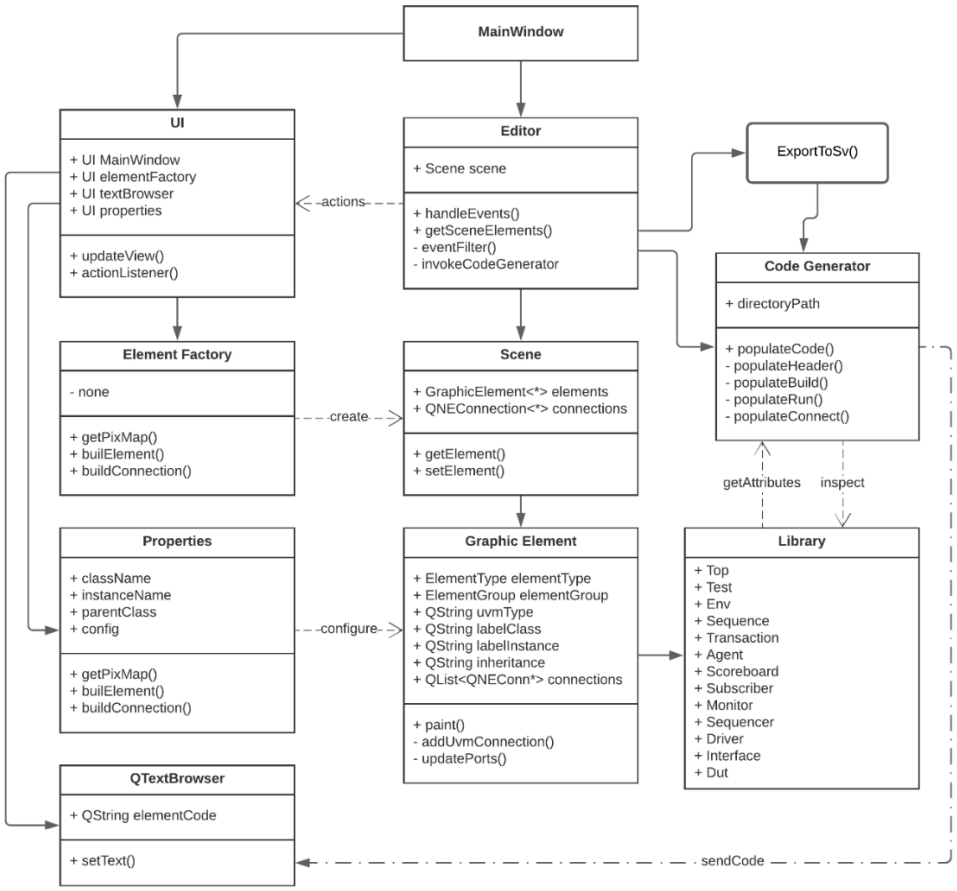


Figure 12 - UML Diagram

Chapter 4: Testbench Generation

In this chapter, the tool is used to demonstrate that it can provide a quick and effective way to create complete UVM environments. The design and generation of the SystemVerilog code for two testbenches is covered: one of a simple ALU and one for the ITESO´s SerDes. The first DUT addressed the ALU, as it is easier to show all the benefits in a rather simple design and afterward switch to a more complex one as it is the SerDes.

4.1. ALU Testbench

4.1.1. Overview

The first DUT for which the testbench is going to be generated is a simple ALU whose diagram is shown in Figure 13. The ALU is composed of two four bits entries A and B and a 3 bits selector, then based on the selected opcode the corresponding result is presented in the 5-bit output “result”.

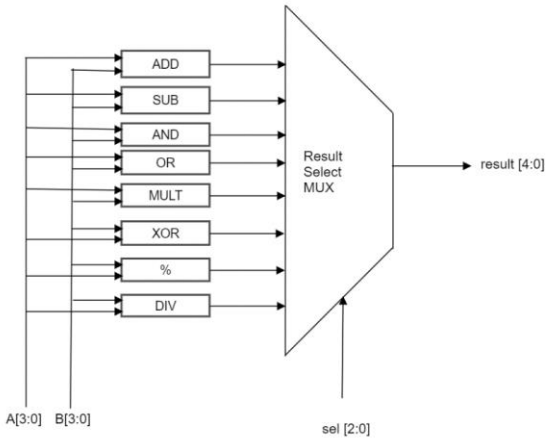


Figure 13 - ALU diagram

The ALU is capable of computing any of the following operators:

- Addition
- Subtraction
- Logical AND
- Logical OR
- Multiplication
- Logical XOR
- Modulo operator
- Division

As simple as this design looks, even for this ALU creating a complete UVM environment from scratch would be very time-consuming and is propense to have many human errors.

By using the “UVM framework” tool, the number of human errors is drastically reduced, and the validation process is accelerated.

As discussed in previous chapters, the components that this ALU requires to have for a complete environment are the following:

- DUT
- Agent
- Driver
- Environment
- Interface
- Monitor
- Scoreboard
- Sequence item
- Sequencer
- Sequence
- TB top
- Test
- EnvPkg

4.1.2. Testbench graphic design

The ALU TB design was generated starting from the top hierarchy, i.e. the TB top module. As a first step, a TOP component was dragged to the scene as shown in Figure 14, to set this graphic element as the parent item for the rest of the elements, ports, and connections.

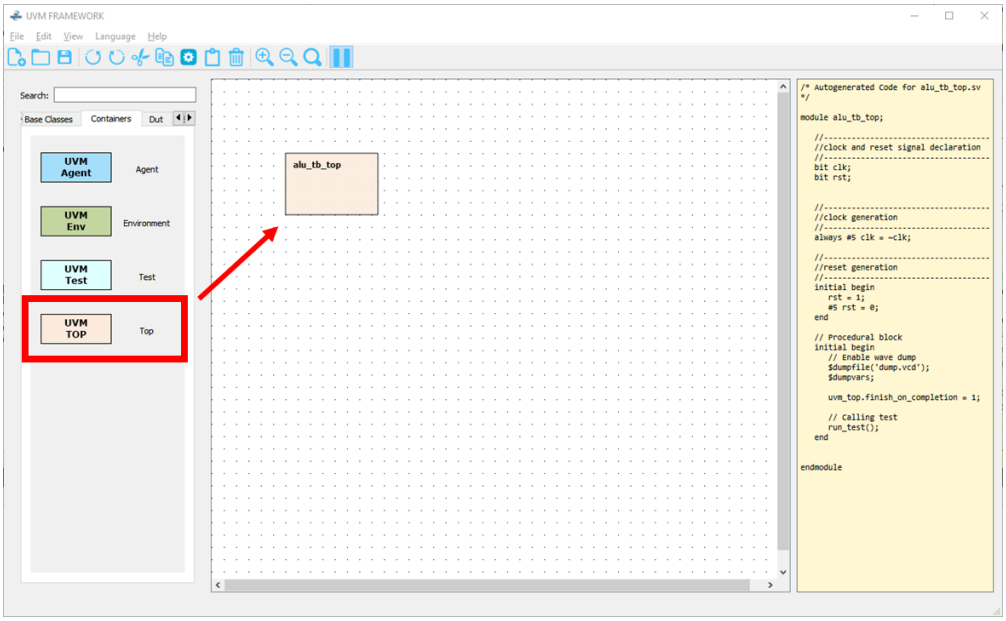


Figure 14 - Adding UVM TOP

The default name of the TOP element was replaced with the specific name for the ALU TB using the properties menu as shown in Figure 15. In this case, there is no need to fill in the

instance name since no other module instantiates the top module. A signal list capability is provided in case the user needs to add specific signals, but for this case, additional signals are not needed.

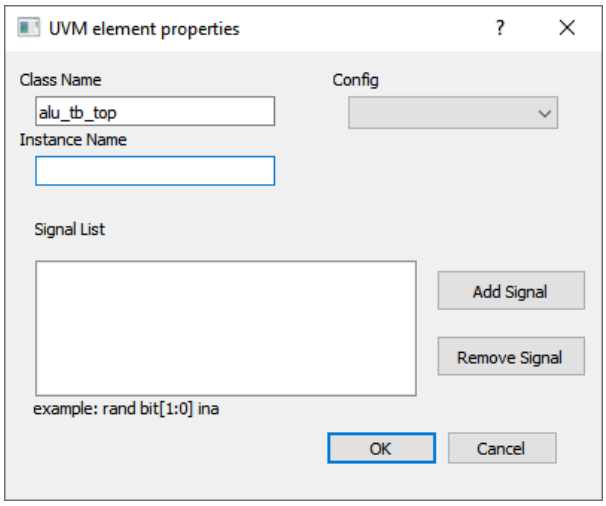


Figure 15 - UVM TOP properties menu

The TOP module contains the instances of the DUT and Interfaces. For the ALU TB, the internal elements are added using the right-click menu, and the field “Add Element...” as shown in Figure 16.

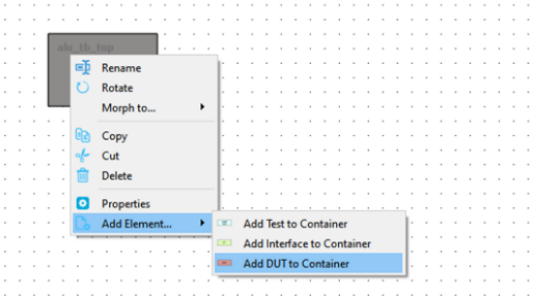


Figure 16 - UVM TOP Right-Click Menu

Once the elements are in place, an instance name is required to get a compilable code. Class name changes are optional, but in this case, the name provided by the golden model of the ALU is assigned. Signals are mandatory to make a correct instantiation and connection between the interface and DUT.

The interface properties are configured as shown in Figure 17.

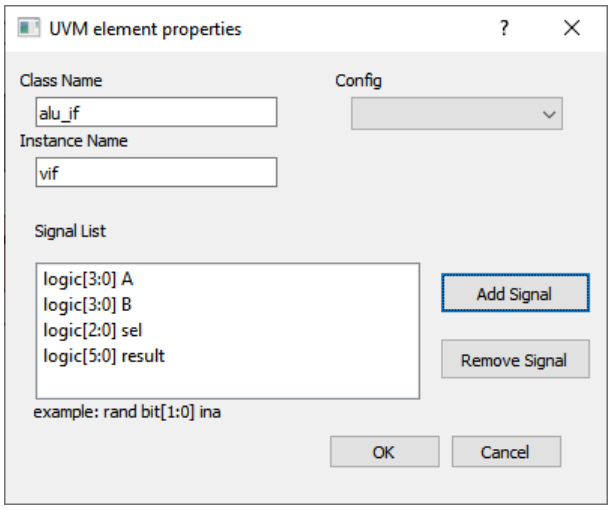


Figure 17 - Interface Properties

DUT properties are configured as follows in Figure 18.

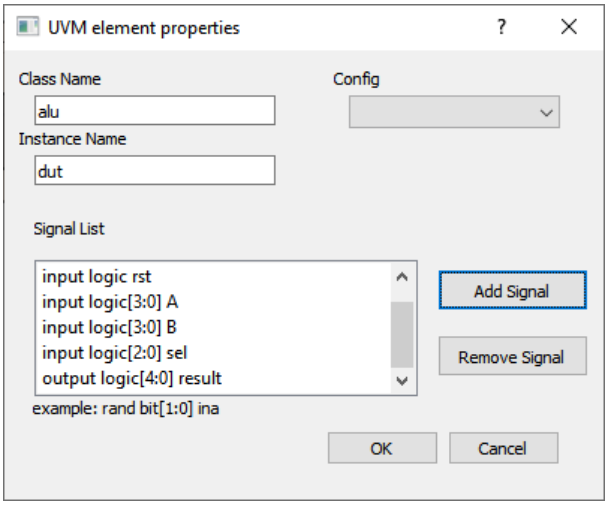


Figure 18 - DUT properties

Both elements are drawn in the scene as child items of the Top module and the code generated now contain both instances. This is reflected in Figure 19.

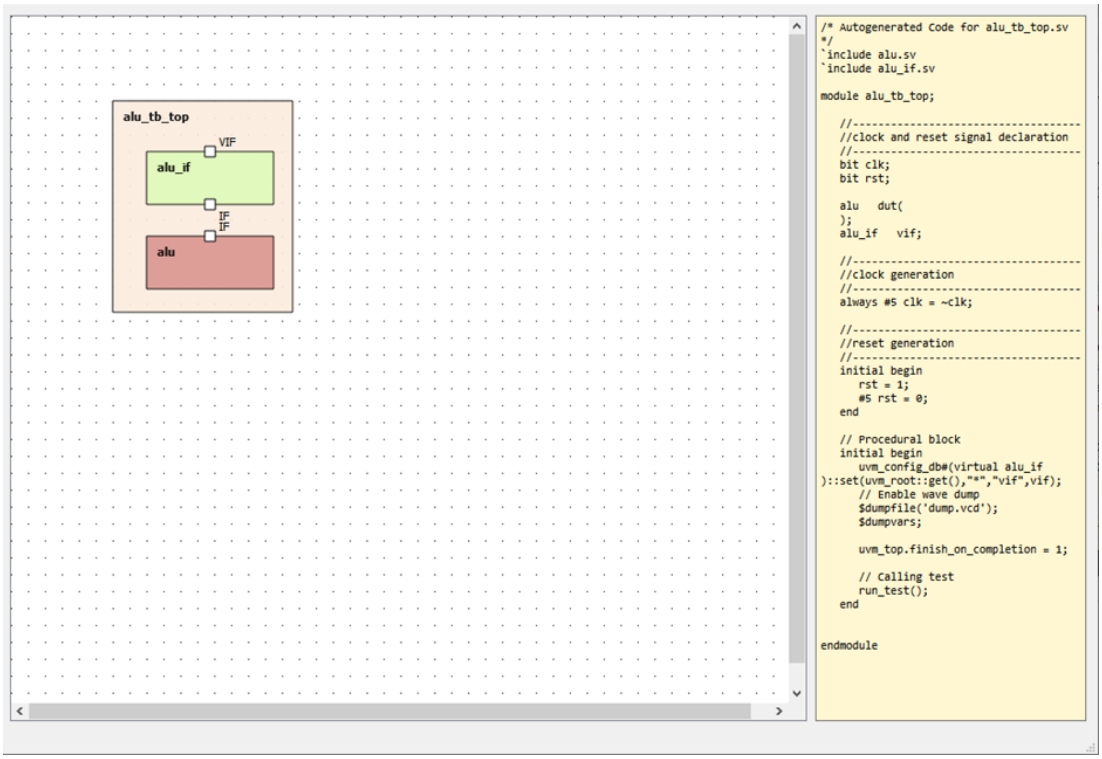


Figure 19 - UVM Top with DUT and IF

Port connections between the Interface and DUT are provided as an “IF” port which connects the physical signals from the ALU to the SV interface. This connection is made using the SV syntax when the wire is drawn as shown in Figure 20.

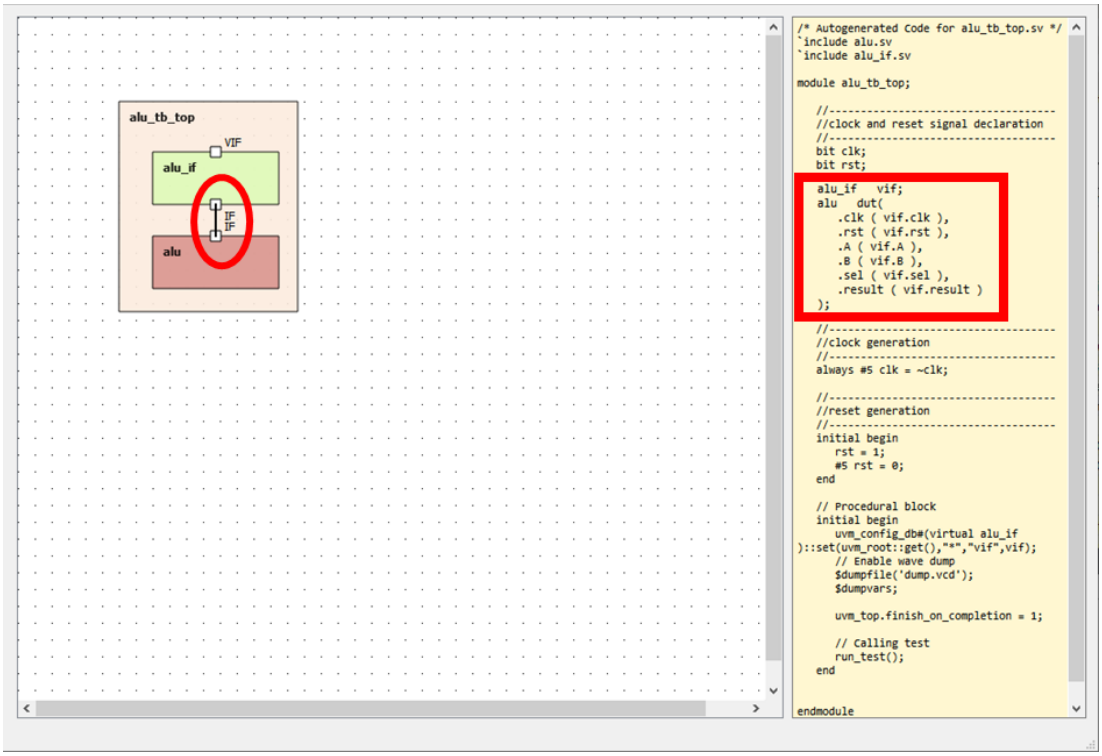


Figure 20 - Wiring IF and DUT

The last graphic element instantiated in the TOP module is the TEST element as shown in Figure 21. This is the second hierarchy in a typical UVM TB.

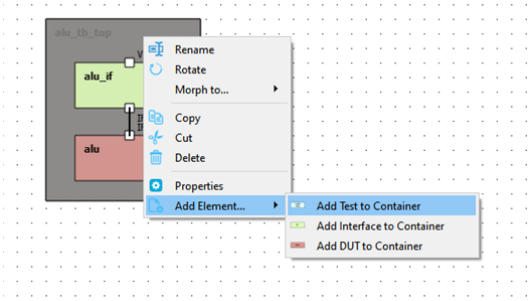


Figure 21 - Adding UVM Test to TB_TOP

A test element needs to have a class name but there is no mandatory instance name. Even when it is under the TOP module hierarchy, an explicit instantiation is not required in the code. The run_test() UVM call executes the corresponding test instead. A Sequence element must be selected to get a compile clean code, so a corresponding sequence will be assigned later in the process. The signal list is optional as shown in Figure 22.

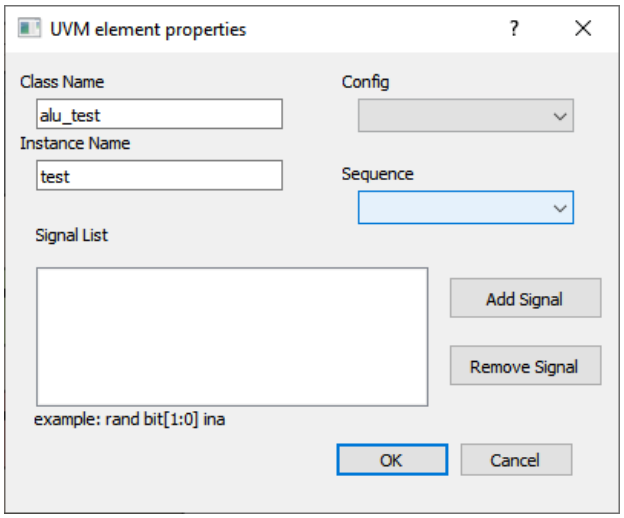


Figure 22 - UVM Test properties menu

The initial Test code and diagram is shown below in Figure 23.

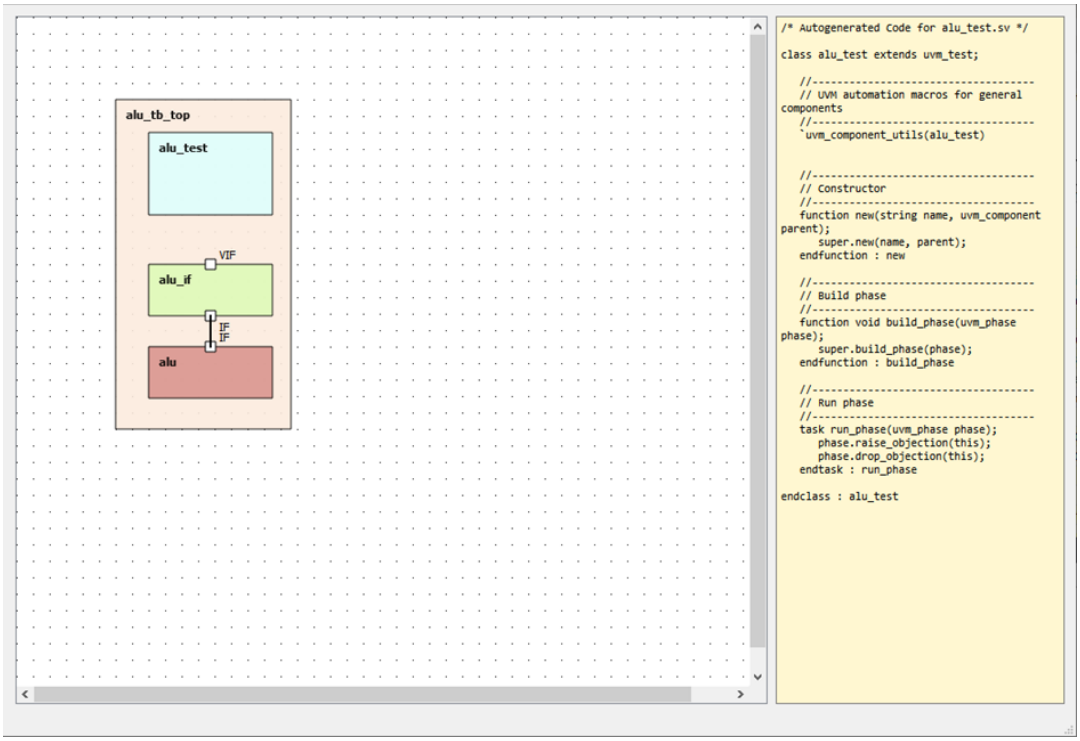


Figure 23 - TB TOP with IF, DUT, and UVM Test

The Test is the second level hierarchy element that contains most of the UVM elements required to complete the validation TB. Sequences, Environments, and Config elements can be added to a Test container.

As a first step, a Sequence is added to the Test as shown in Figure 24.

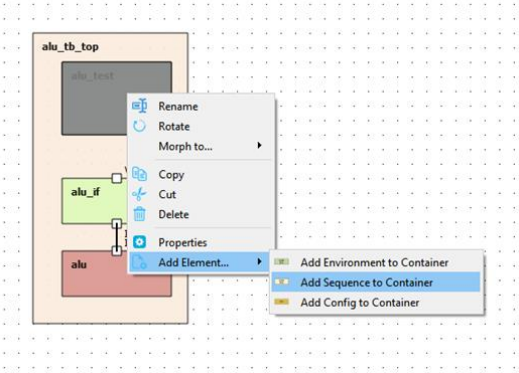


Figure 24 - Adding a UVM Sequence to the UVM Test

It is important to recall that the sequence Class Name and Instance Name will be started later using a function at the Test hierarchy. This means that an adequate instance name ‘req’ should be provided, even when it is not strictly required for the compilation process. This is shown in Figure 25.

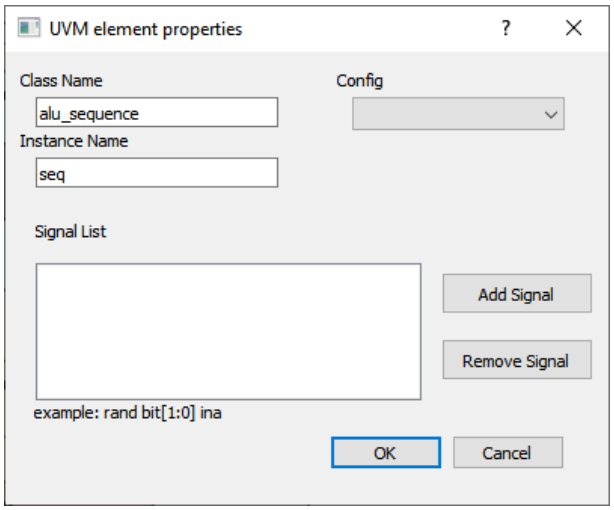


Figure 25 - UVM sequence properties menu}

Then, the Sequence element is added to the scene as is shown in Figure 26.

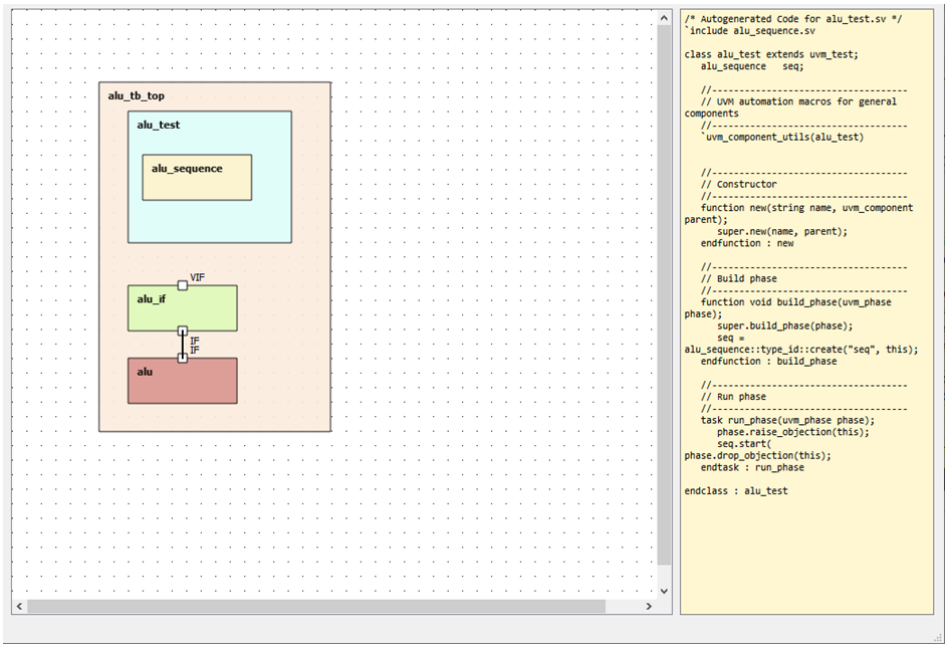


Figure 26 - Adding UVM sequence to UVM test

An Environment is being added as the main container for the UVM components as shown in Figure 27.

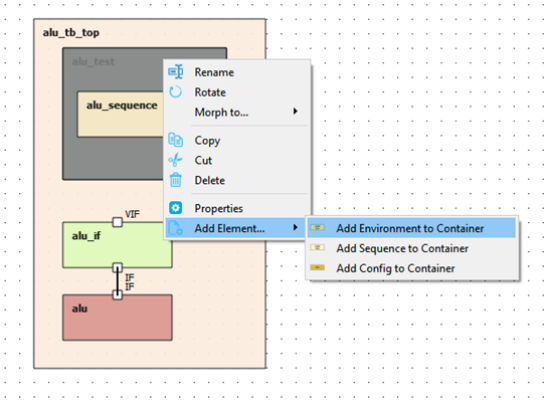


Figure 27 - Adding UVM Env to UVM Test

The Environment does not need to be aware of the Sequence or Sequence Item selected, so, in this case, the Class and Instance name are the only required attributes as shown in Figure 28.

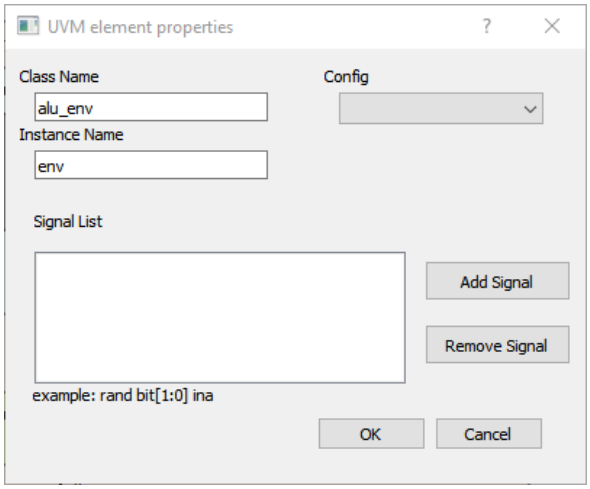


Figure 28 - UMV Env properties menu

Now, the ALU Test is complete. However, the Sequence selection is still pending. For this, a specific Sequence is now selected in the Test properties menu. There is one option now that points to the newly added Sequence ‘req’. Since, one single Sequence is needed, ‘req’ is selected as the Sequence to be started by the Test as shown in Figure 29.

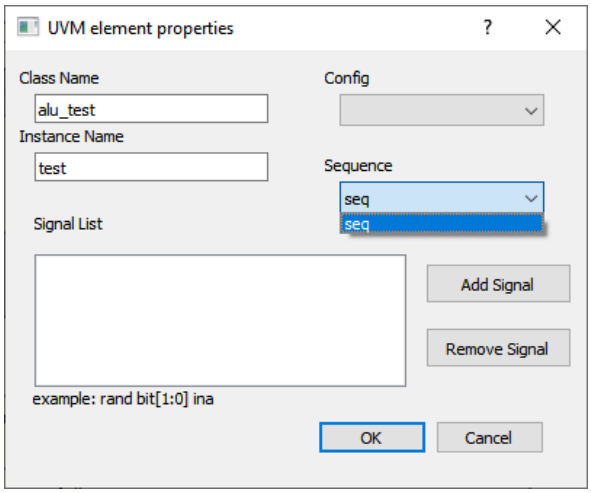


Figure 29 - UVM Test (Sequence selection)

The Scene updated and Test code with its elements is shown below in Figure 27.

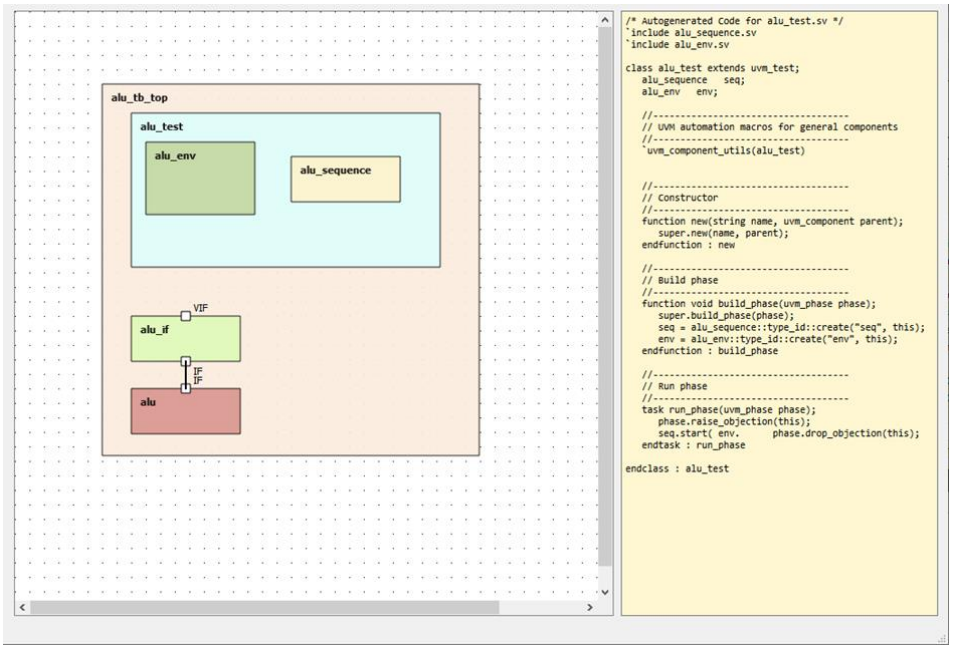


Figure 30 - Adding Env to Test

The sequence body will be completed by the user. However, it still needs to instantiate Sequence Item elements inside the Sequence. For this, a new Sequence Item element is added using the properties menu as it is displayed in Figure 31.

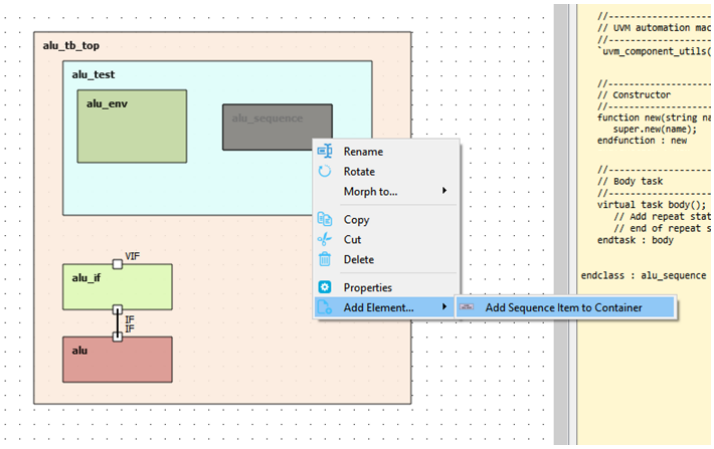


Figure 31 - Adding sequence item to UVM Sequence

The Sequence Item needs a Class Name, but not an Instance Name necessarily. The properties menu for the sequence is shown in Figure 32.

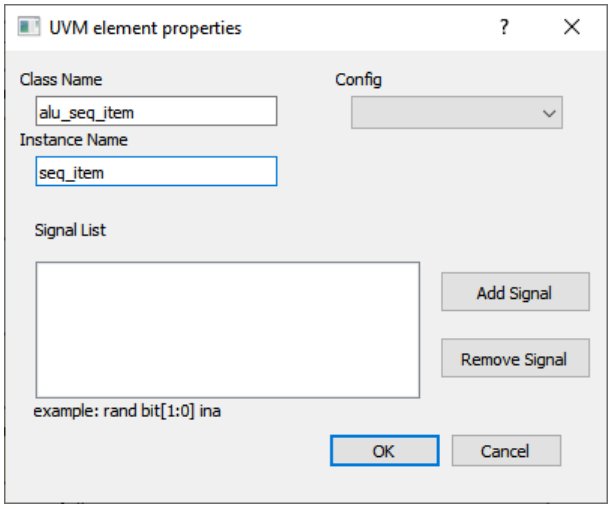


Figure 32 - Sequence item properties menu

The signal list is an important feature that allows the code generator to create the UVM fields macros to implement the randomization and configuration of UVM Object attributes. So, in this case, the signal list naming is relevant to differentiate the random signals from the rest of them. For the ALU TB, the random signals are the input operands and operation selector. This has been filled as an example in Figure 33. Both ‘rand’ and ‘randc’ keywords are supported.

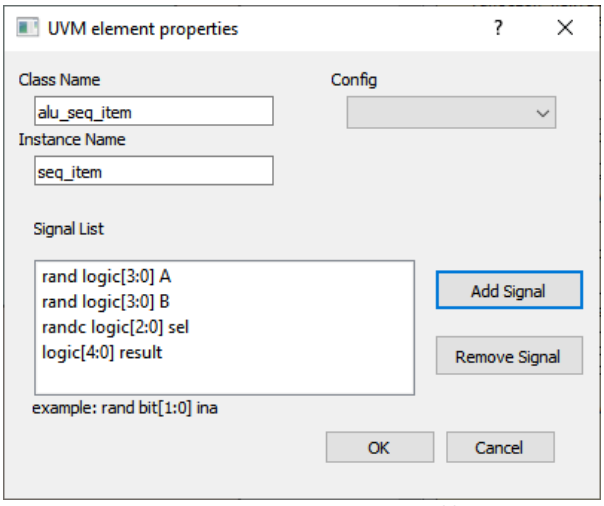


Figure 33 - Sequence item signal list

The code and diagram after the modifications for the Sequence Item are displayed in Figure 34.

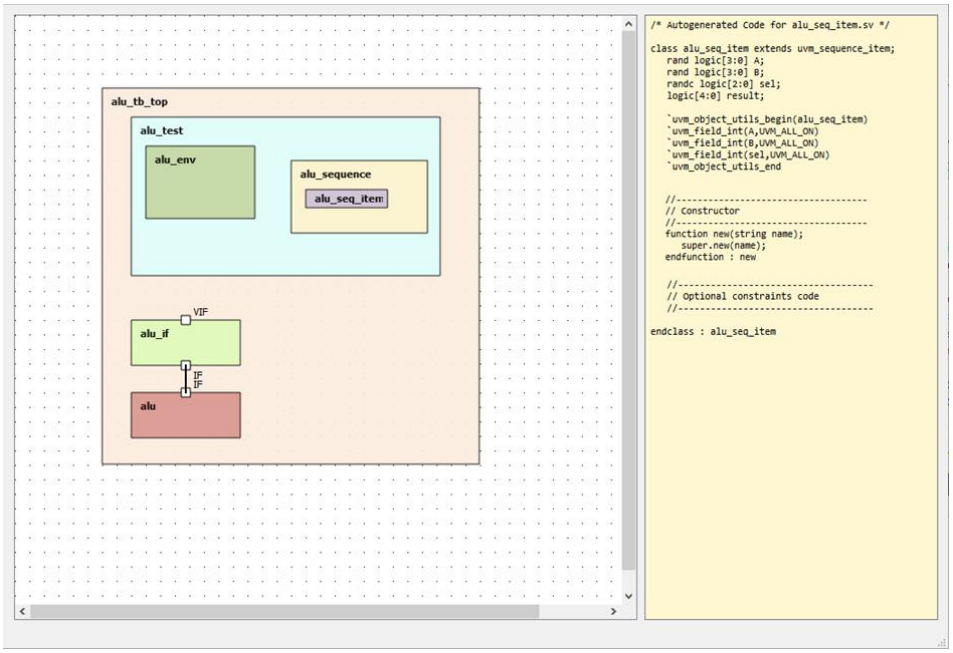


Figure 34 - Sequence item in UVM env

Additionally, the code in the Sequence gets updated with its corresponding parameter pointing now to the new Sequence Item as shown in Figure 35.

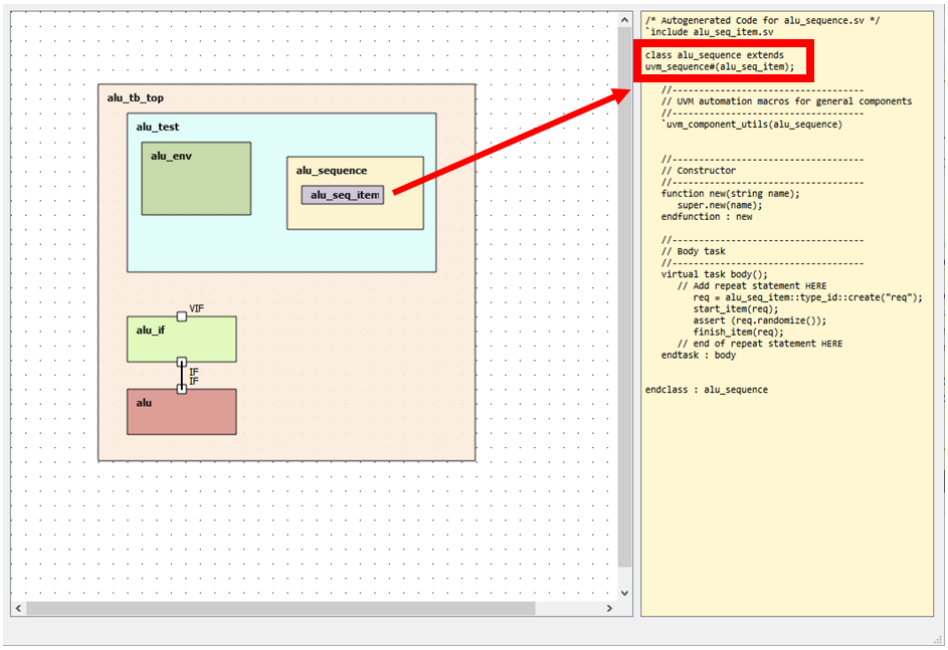


Figure 35 - UVM Sequence with UVM Sequence Item

Coming back to the Env element, the next component required is the Scoreboard. It can be added as shown in Figure 36.

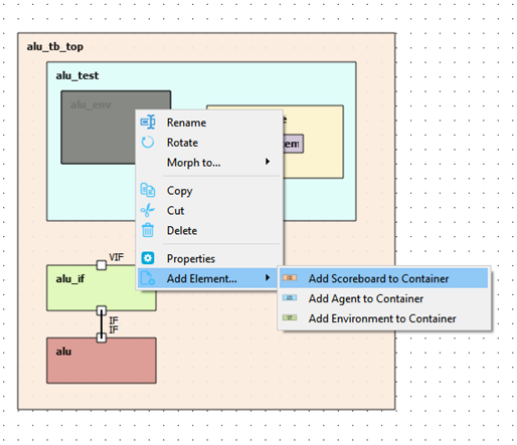


Figure 36 - Adding Scoreboard to UVM Env

A Sequence Item is optional for the Scoreboard, but it simplifies the code generation, so for this example, the 'alu_seq_item' is selected. The signal list is optional for the Scoreboard. Figure 37 presents the properties menu of the scoreboard.

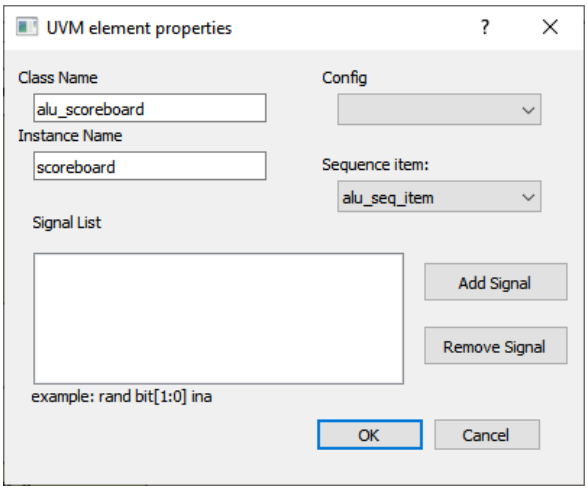


Figure 37 - UVM Scoreboard properties menu

Scoreboards are created by default with a UVM TLM analysis FIFO, which is represented by a circle in the diagram. This port is also included in the code.

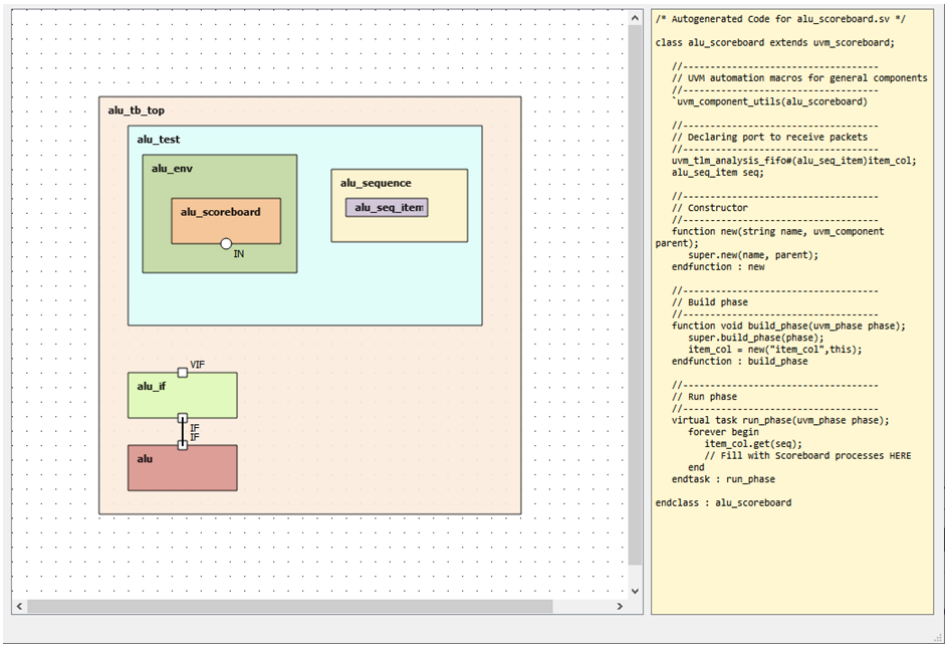


Figure 38 - UVM Env with UVM Scoreboard

A second important element in the Env is an Agent. This component will be added at the same level as the Scoreboard as shown in Figure 39.

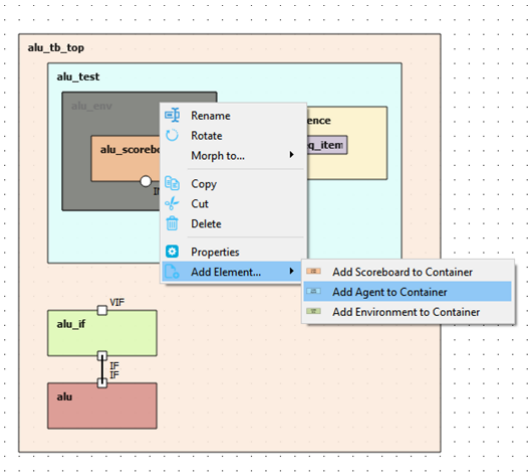


Figure 39 - Adding UVM Agent to UVM Env

The Agent does not need to be aware of the specific Sequence or Sequence Item that is been handled. The signal list, again, is optional as shown in Figure 40.

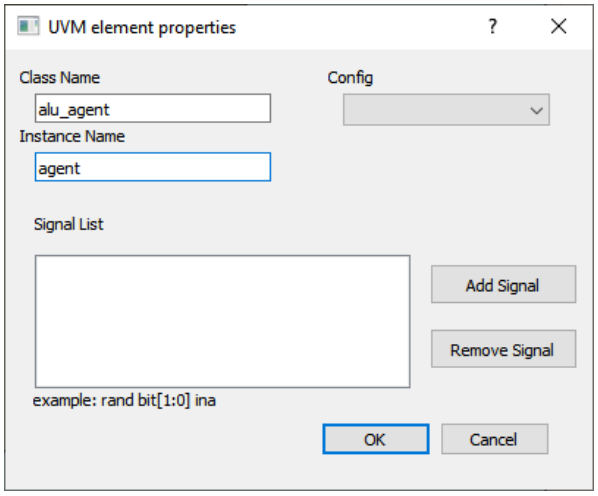


Figure 40 - UVM Agent - Properties menu

So, the Env element diagram and code are as follows in Figure 41.

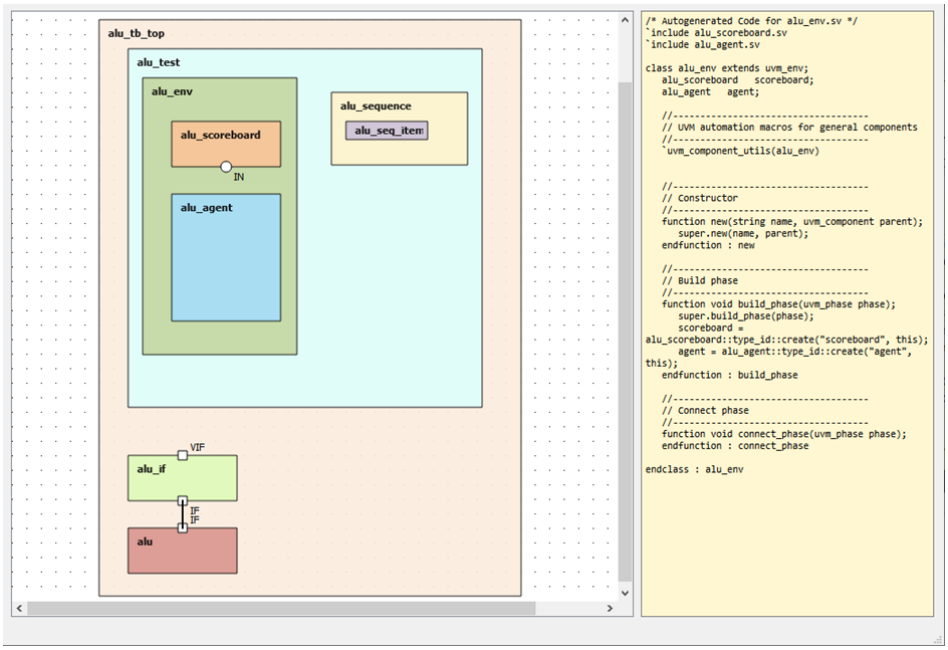


Figure 41 - UVM Env with Agent and Scoreboard

The Agent contains three different kinds of elements: Sequencer, Driver and, Monitor.

Monitor addition is shown below in Figure 42.

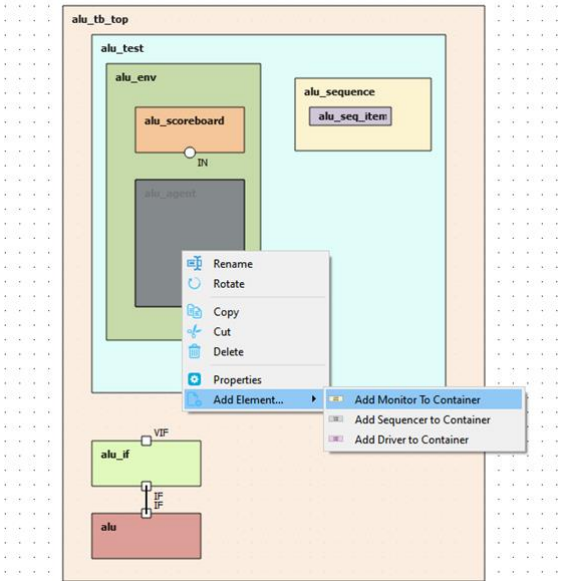


Figure 42 - Adding UVM Monitor to UVM Agent

A Monitor element needs to know the Sequence Item. An instance name is required for the code generation of upper hierarchies. The signal list is optional. The properties menu of the monitor is showcased in Figure 43.

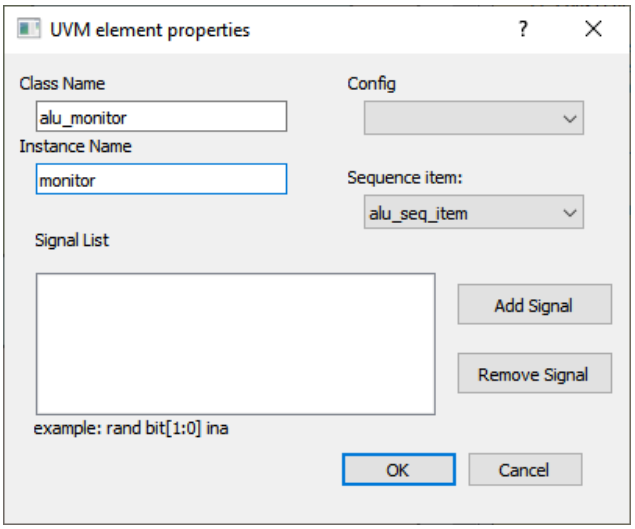


Figure 43 - UVM Monitor properties menu

Monitor code is shown in the following Figure 44.

```
/* Autogenerated Code for alu_monitor.sv */

class alu_monitor extends uvm_monitor;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(alu_monitor)

   //------------------------------------
   // Analysis port
   //------------------------------------
   uvm_analysis_port#(alu_seq_item)item_collected_port;

   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      item_collected_port = new("item_collected_port", this);
   endfunction : build_phase

   //------------------------------------
   // Run phase
   //------------------------------------
   virtual task run_phase(uvm_phase phase);
      alu_seq_item seq_item_collected =
alu_seq_item::type_id::create("alu_seq_item", this);
      forever begin
         // Fill with Monitor sampling process HERE
      end
   endtask : run_phase

endclass : alu_monitor
```

Figure 44 - UVM monitor auto-generated code

Sequencer addition is shown below in Figure 45.

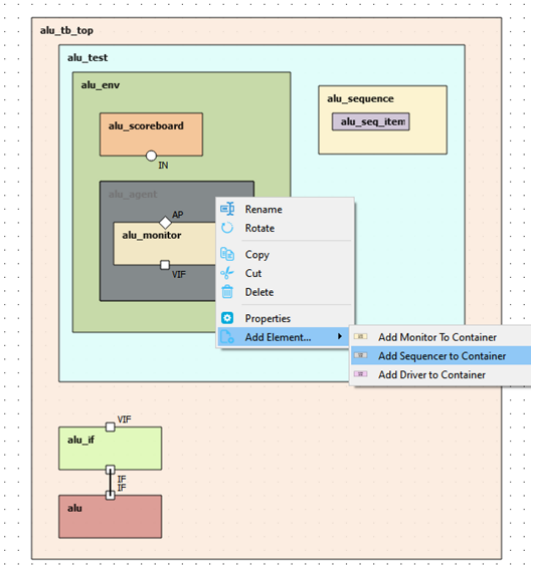


Figure 45 - Adding Sequencer to UVM Agent

In a similar case as the Monitor, the Sequencer only needs the instance name and Sequence Item. The signal list is optional. Again, the properties menu for this component can be seen in Figure 46.

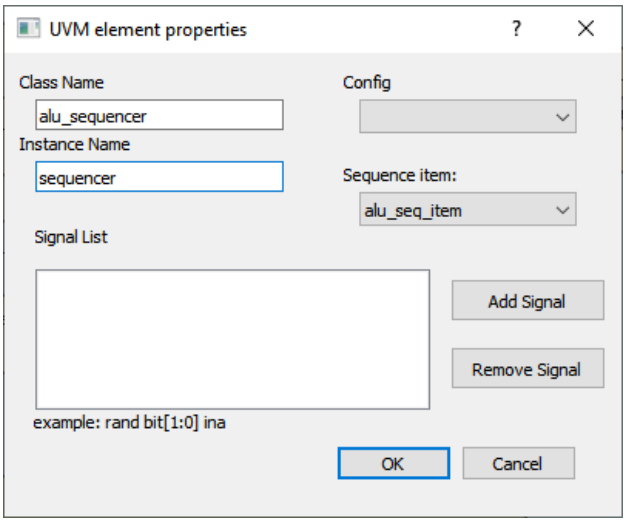


Figure 46 - UVM Sequencer properties menu

The Sequencer code is shown below in Figure 47.

```
/* Autogenerated Code for alu_sequencer.sv */

class alu_sequencer extends uvm_sequencer#(alu_seq_item);

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(alu_sequencer)

   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new


endclass : alu_sequencer
```

Figure 47- UVM Sequencer Code

Finally, a Driver element is added to the Agent as shown in Figure 48.

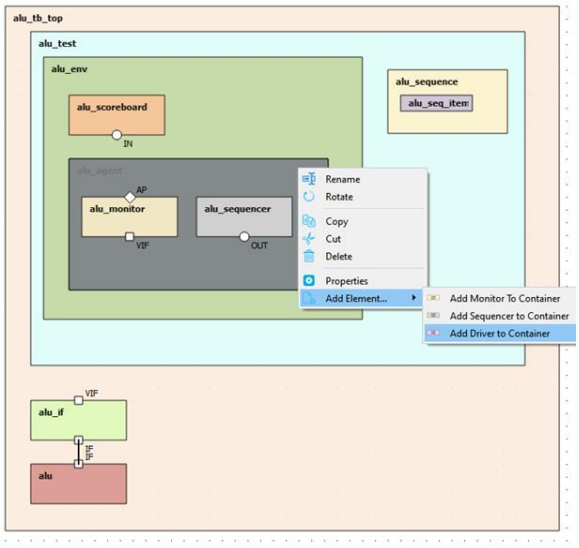


Figure 48 - Adding Driver to UVM Agent

Driver does not need signals specified. Instance name and Sequence Item is required as shown in Figure 49.

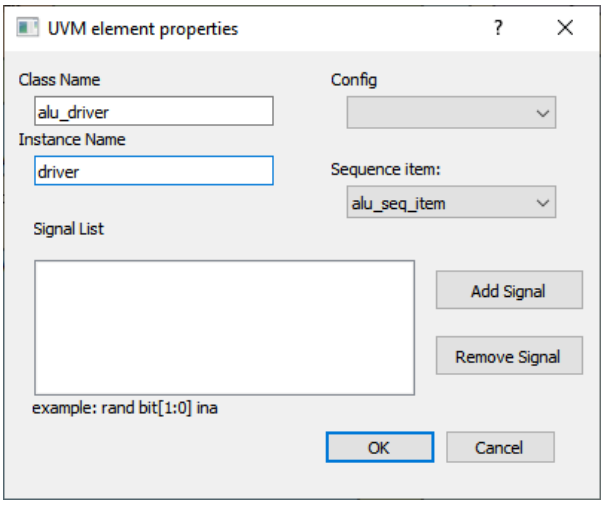


Figure 49 - UVM Driver properties menu

Driver generated code is shown below in Figure 50.

```
/* Autogenerated Code for alu_driver.sv */

class alu_driver extends uvm_driver#(alu_seq_item);

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(alu_driver)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
   endfunction : build_phase

   //------------------------------------
   // Run phase
   //------------------------------------
   virtual task run_phase(uvm_phase phase);
      forever begin
         seq_item_port.get_next_item(req);
         drive();
         seq_item_port.item_done();
      end
   endtask : run_phase

   //------------------------------------
   // Drive task
   //------------------------------------
   virtual task drive();
      // TODO fill with your driving procedures
   endtask : drive

endclass : alu_driver
```

Figure 50 - UVM Driver Auto-generated Code

Agent code with all its instances in place is as follows in Figure 51. Something to recall is the conditional create statements for Sequencer and Driver elements. This part of the code is enabled only if the Agent is configured as ACTIVE. For more details, please refer to section 2.4.5.

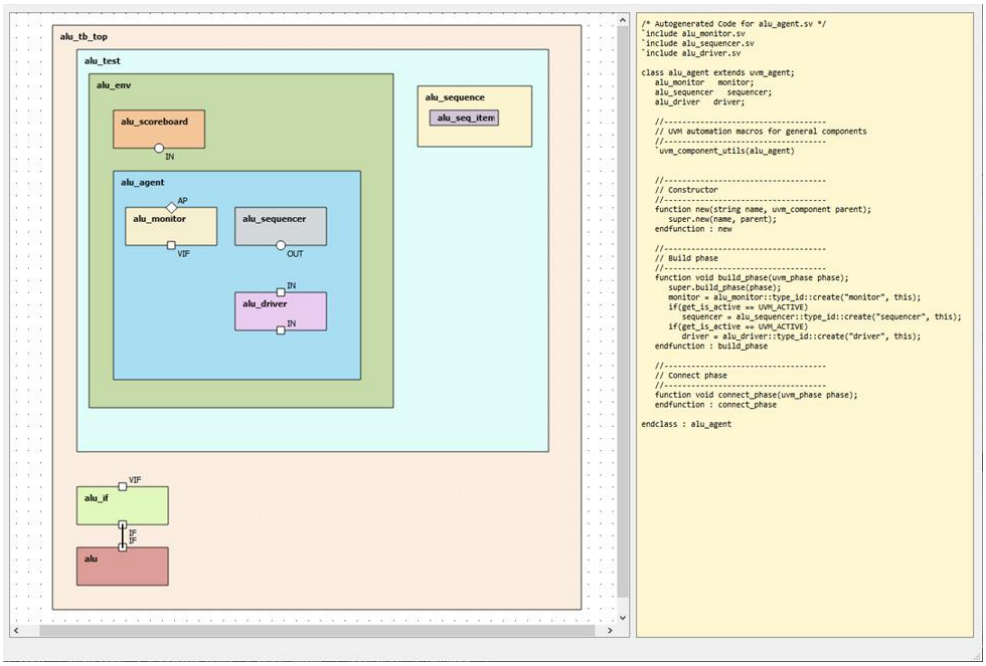


Figure 51 - UVM TB with Driver

Connections need to be added between the components in the scene. The only connection so far is between the Interface and DUT. A similar process to connect the rest of the elements will be described in detail in the next pictures.

The most internal connection is between the Sequence and the Driver. Both are internal elements of the Agent, so the connection is performed graphically in the same Graphic Element (Agent). The connection code is also included inside the Agent as shown in Figure 52 below.

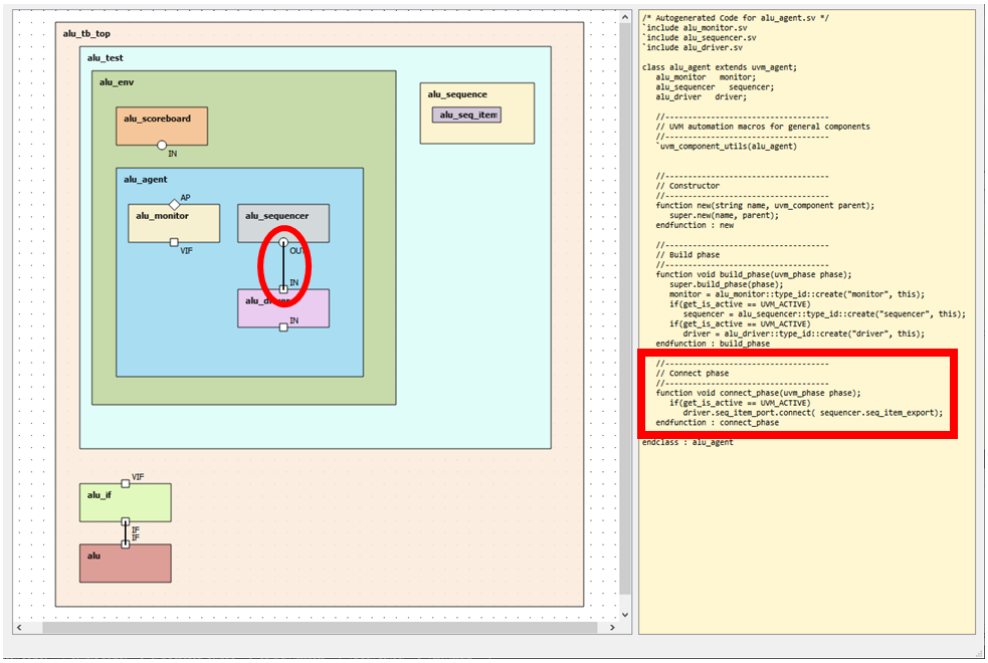


Figure 52 - Connection of Driver and Sequencer

The connection between the Monitor and the Scoreboard is made at the Env hierarchy since it is the common parent of both elements. There should be always a common parent between elements to get them properly connected. This connection is shown in Figure 53.

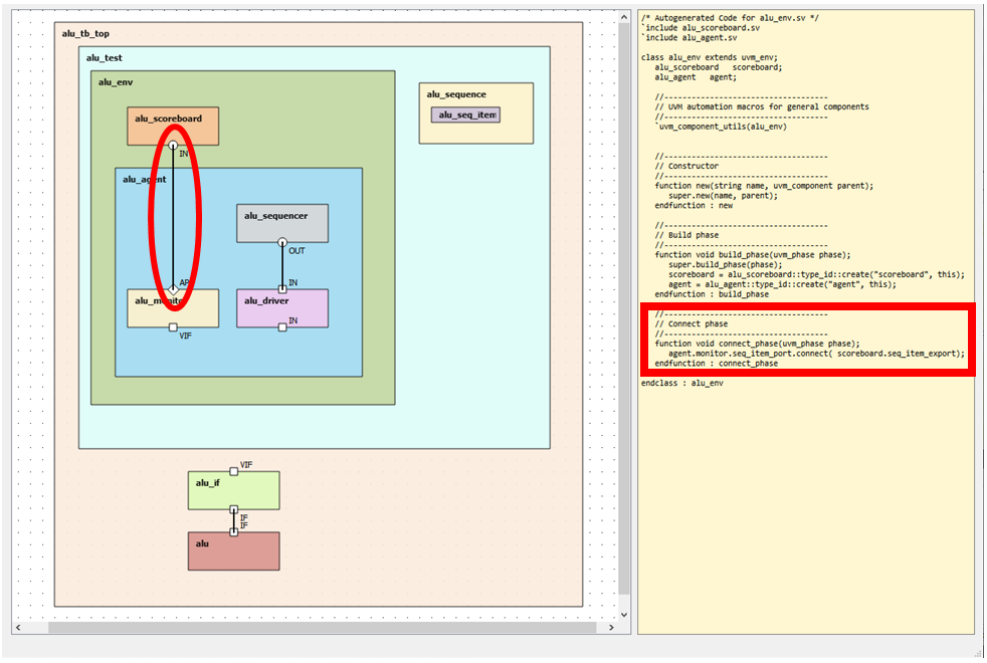


Figure 53 - Connection of Scoreboard and monitor

The last two connections are made between Monitor-Interface and Driver-Interface. The connection for the virtual interfaces is slightly different from the one made between UVM ports as shown in Figure 54.

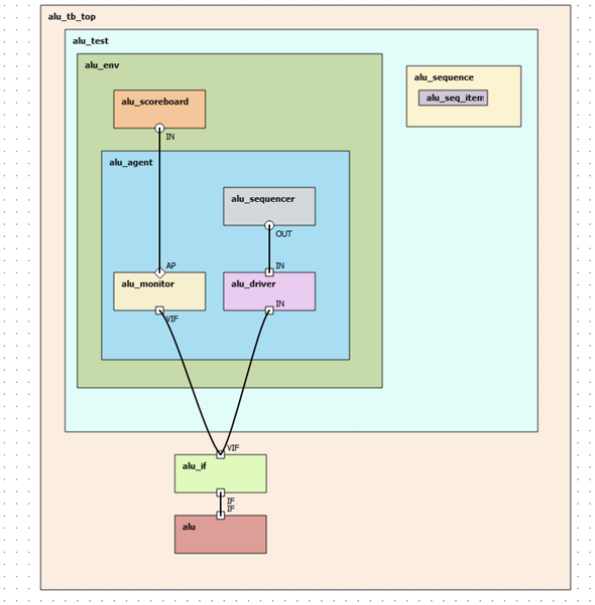


Figure 54 - Connection of Driver and Monitor with the IF

For the Monitor the resultant code in Figure 55 is as follows:

```
/* Autogenerated Code for alu_monitor.sv */

class alu_monitor extends uvm_monitor;
   //------------------------------------
   // Virtual Interface
   //------------------------------------
   alu_if   vif;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(alu_monitor)

   //------------------------------------
   // Analysis port
   //------------------------------------
   uvm_analysis_port#(alu_seq_item)item_collected_port;

   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      item_collected_port = new("item_collected_port", this);
      if(!uvm_config_db#(virtual alu_if)::get(this, "","vif",vif))
         `uvm_fatal("No_vif", {"Virtual interface must be set for:
get_full_name(),".vif"});
   endfunction : build_phase

   //------------------------------------
   // Run phase
   //------------------------------------
   virtual task run_phase(uvm_phase phase);
      alu_seq_item seq_item_collected =
alu_seq_item::type_id::create("alu_seq_item", this);
      forever begin
         // Fill with Monitor sampling process HERE
      end
   endtask : run_phase

endclass : alu_monitor
```

Figure 55 - UVM Monitor Auto-generated code

And the Driver has the following code in Figure 56:

```
/* Autogenerated Code for alu_driver.sv */

class alu_driver extends uvm_driver#(alu_seq_item);
   //------------------------------------
   // Virtual Interface
   //------------------------------------
   alu_if   vif;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(alu_driver)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db#(virtual alu_if)::get(this, "","vif",vif))
         `uvm_fatal("No_vif", {"Virtual interface must be set for:
get_full_name(),".vif"});
   endfunction : build_phase

   //------------------------------------
   // Run phase
   //------------------------------------
   task run_phase(uvm_phase phase);
      forever begin
         seq_item_port.get_next_item(req);
         drive();
         seq_item_port.item_done();
      end
   endtask : run_phase

   //------------------------------------
   // Drive task
   //------------------------------------
   virtual task drive();
      // TODO fill with your driving procedures
   endtask : drive

endclass : alu_driver
```

Figure 56 - UVM Driver Auto-generated Code

A Config element can be added optionally to the scene at the Test hierarchy. A config insertion is shown below in Figure 57, however, for the ALU example, this object is optional.

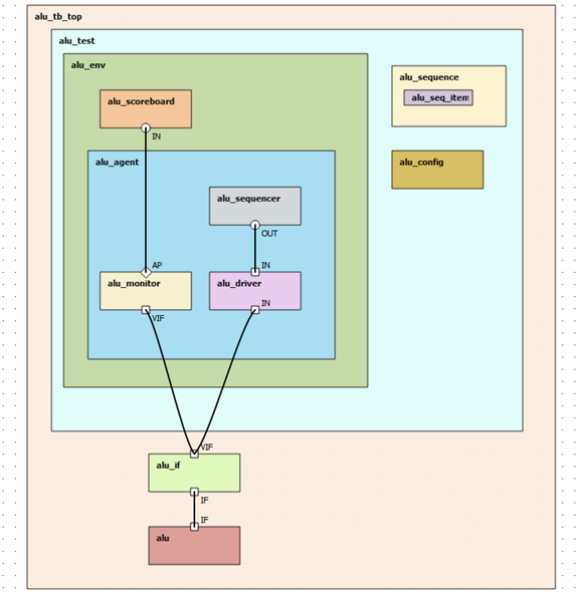


Figure 57 - Adding UVM Config to the Test

There is another component that improves the Quality of Validation and adds flexibility to the Environment. This component is the Subscriber, which consists of a generic component that can receive and process transactions at different levels of the TB. It also has the capability of forwarding transactions to other VCs. However, in this graphical implementation, the subscriber will be used as a wrapper element to introduce the coverage collection capabilities to the TB.

The Subscriber can be added at the Agent or Env hierarchies as shown in Figure 58. For this example, the Agent will contain the Subscriber element.

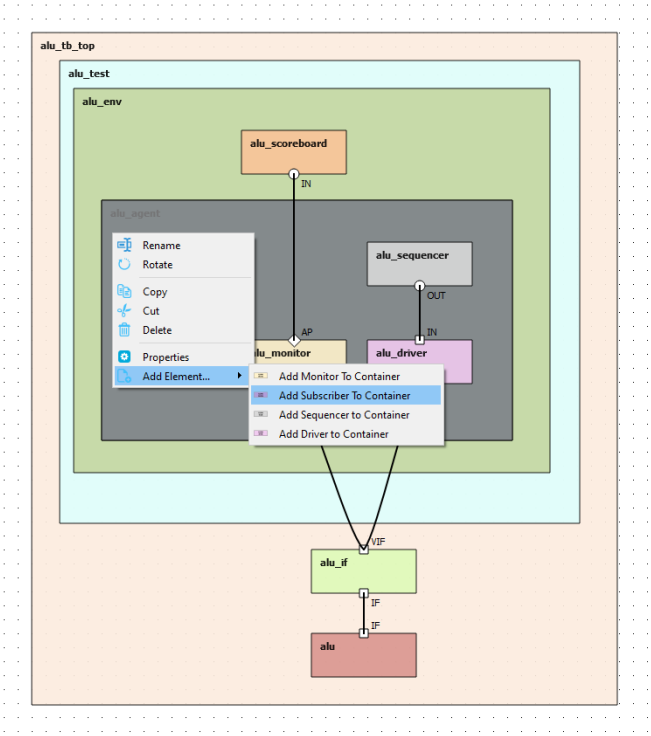


Figure 58 - Adding Subscriber to the TB

The Subscriber requires a Class Name and Instance Name. However, this element has an optional field that no other element includes. The Covergroup list in Figure 59 allows the user to generate a template to include covergroups and manage them from the graphical model.

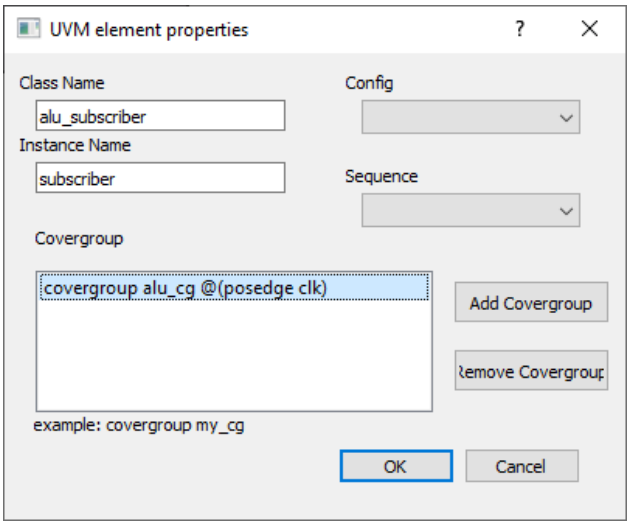


Figure 59 - UVM Subscriber properties menu

The Subscriber component is added close to the Monitor because most of the interactions are between both elements. A new diagonal wire is created between the Monitor and the Subscriber to generate the proper code in the Agent connect phase. This is displayed in Figure 60.

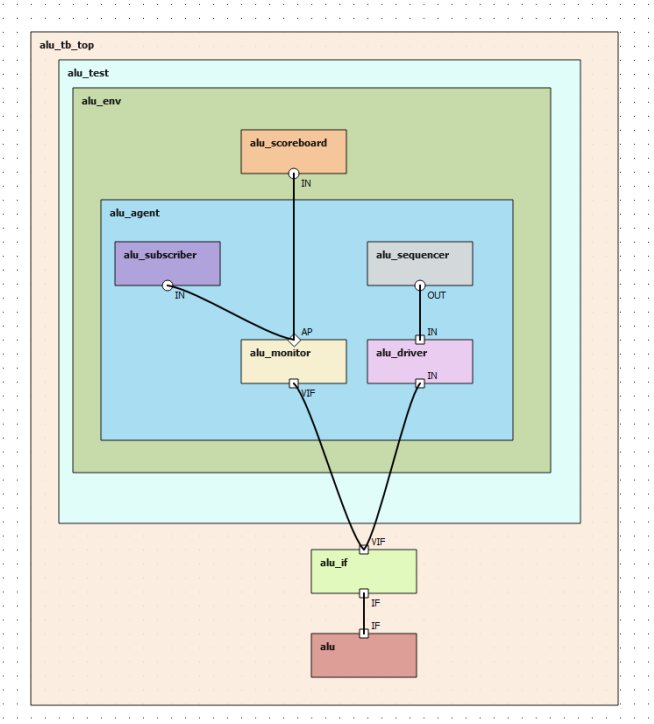


Figure 60 - ALU Testbench with subscriber

A final view of the ALU TB diagram is presented below in Figure 61.

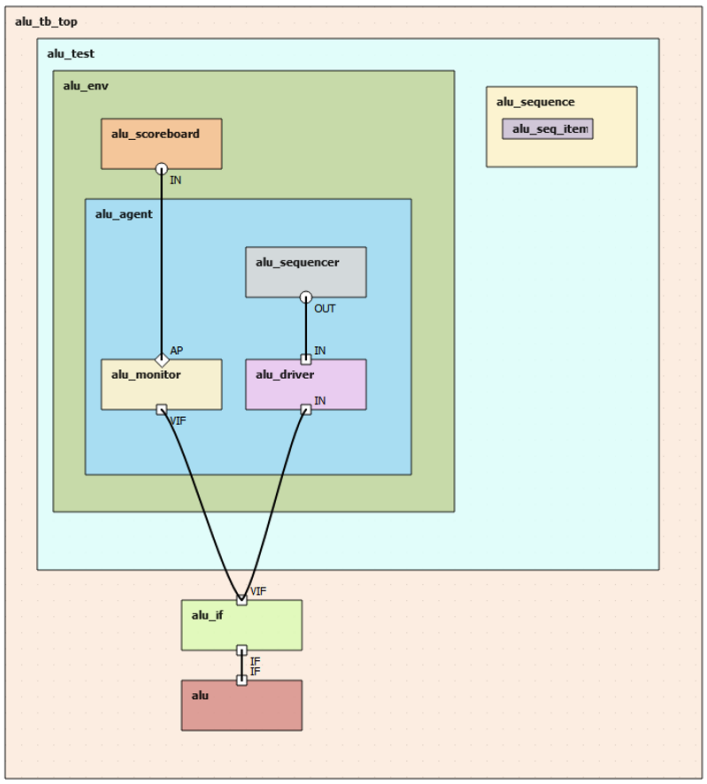


Figure 61 - Final ALU TB

4.1.3. Generated code output files

Once all the elements are in place, a final step consists of the output files generated in SV format. To generate the output files, the option “Export to SystemVerilog” is required as shown in Figure 62.

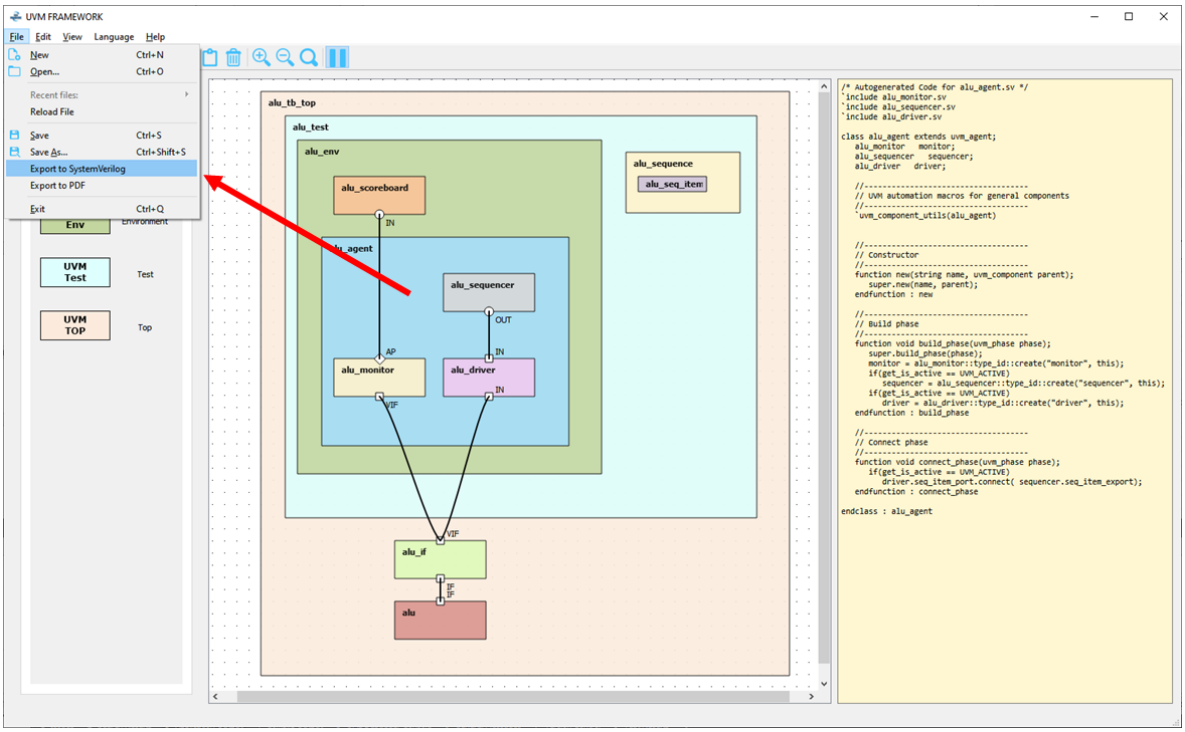


Figure 62 - Generating SV Code for the Design

A name for the project is also required to create the output directory and set the file paths as shown in Figure 63.

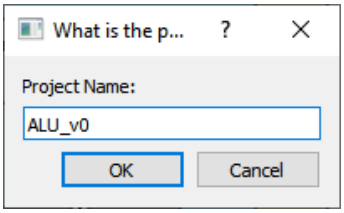


Figure 63 - Adding a Project Name

With the completion of this last step, the files are ready for compilation using the SV compiler preferred by the User as shown in Figure 64.

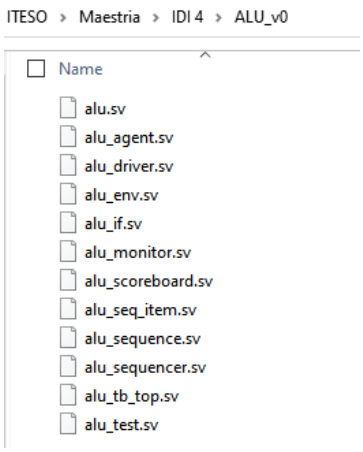


Figure 64 - Output generated files

4.2. SerDes Testbench

4.2.1. Overview

The SerDes device [1] that is validated in this case study is composed of four modules: Analog RX, Digital RX, Analog TX, and Digital TX modules. As shown in Figure 65.

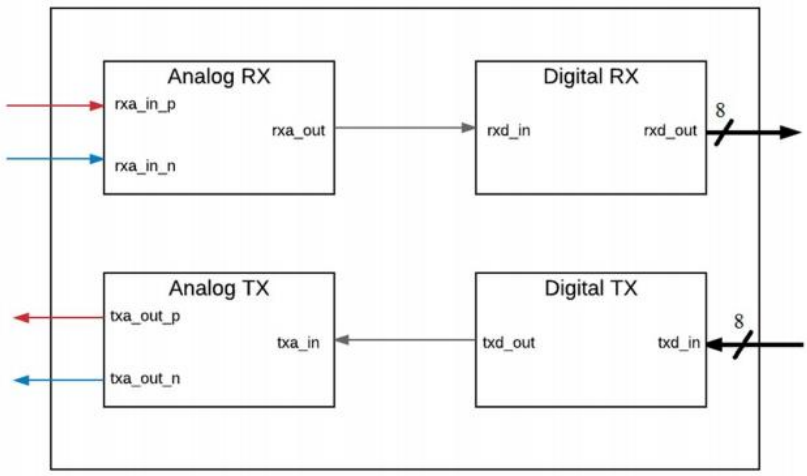


Figure 65 - SerDes General Structure

The receiver function starts with the analog block known as Analog RX, which receives a differential signal as input and produces a digital signal as the output. The digital block, Digital RX, takes the receiver analog block output as input and recovers the data byte encoded and converts the serial data to parallel, and finally, decodes the data.

The transmitter stage begins with the digital block Digital TX whose input is a parallel data byte, which is encoded and transmitted serially toward Analog TX input providing a differential signal as the output.

4.2.2. Testbench graphic design

Generation of the graphic model of the SerDes TB is presented in this section. It consists of a very similar process as the one described for the ALU DUT. However, there are significant differences that will be explained in detail over this section, including more complex cases, config object usage, inheritance, and components reuse.

As a first step, a Top element is added to the scene from the left pane menu as shown in Figure 66. This step is the same required for the ALU and, in general, for every TB that needs to be compiled in a stand-alone setup.

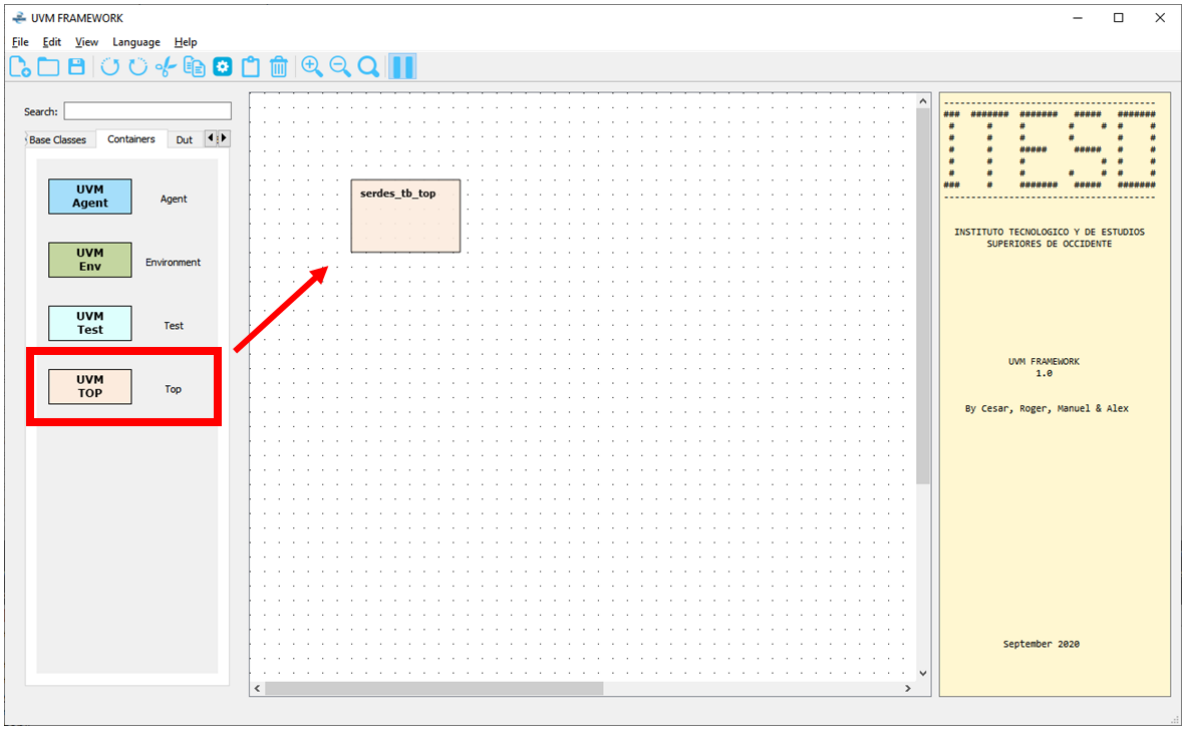


Figure 66 - Adding SerDes TB TOP to the scene

The second element that will be added is the DUT as shown in Figure 67.

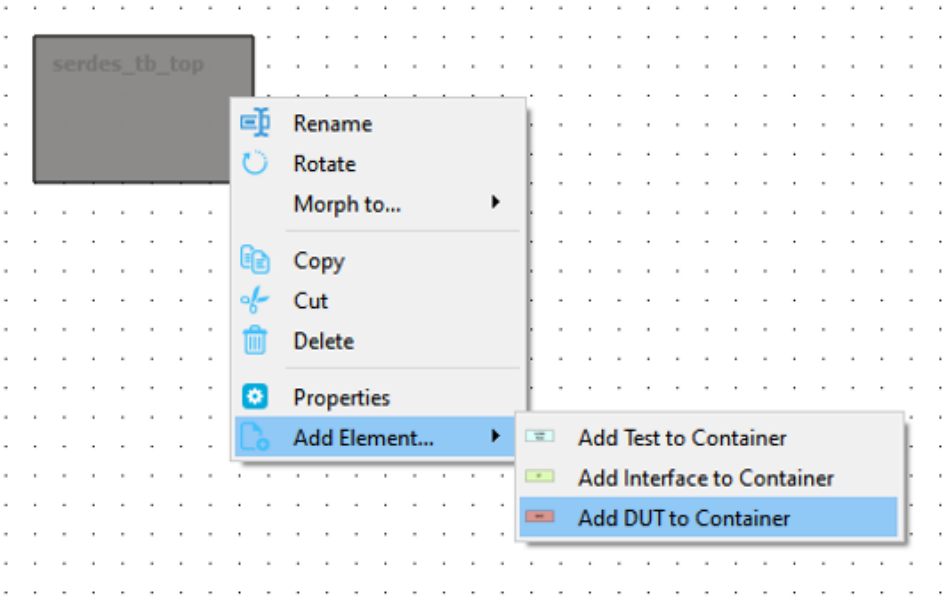


Figure 67 - Adding SerDes DUT to the TB_TOP Container

Dut Class name was not modified from the original version as in [1]. The Instance name was set as “serdes_top” and inputs and outputs get defined in the signal list corresponding field. It is important to remember that the signals are required to connect the Dut and the Interface with the rest of the system as shown in Figure 68.

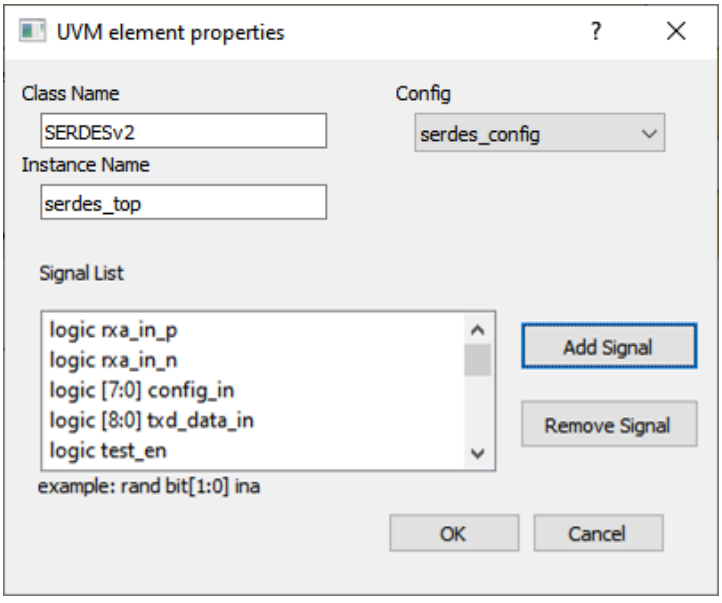


Figure 68 - SerDes DUT Properties menu

Interface instantiation is similar to in the ALU TB as shown in Figure 69.

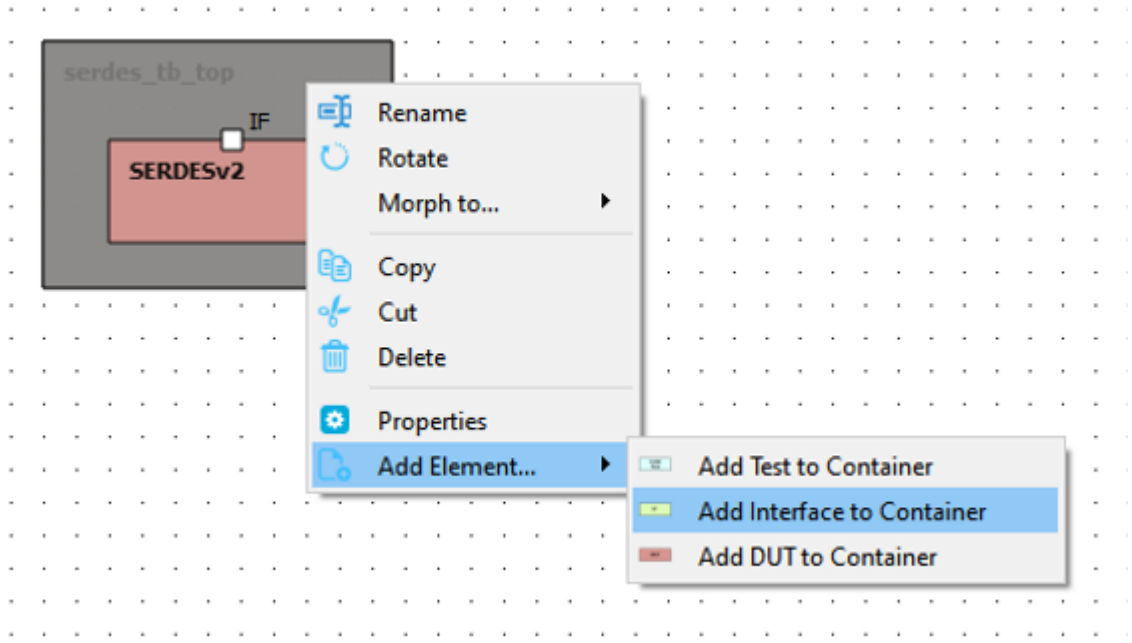


Figure 69 - Adding SerDes Interface to TB_TOP container

Signals are added as well to generate a proper connection as shown in Figure 70.

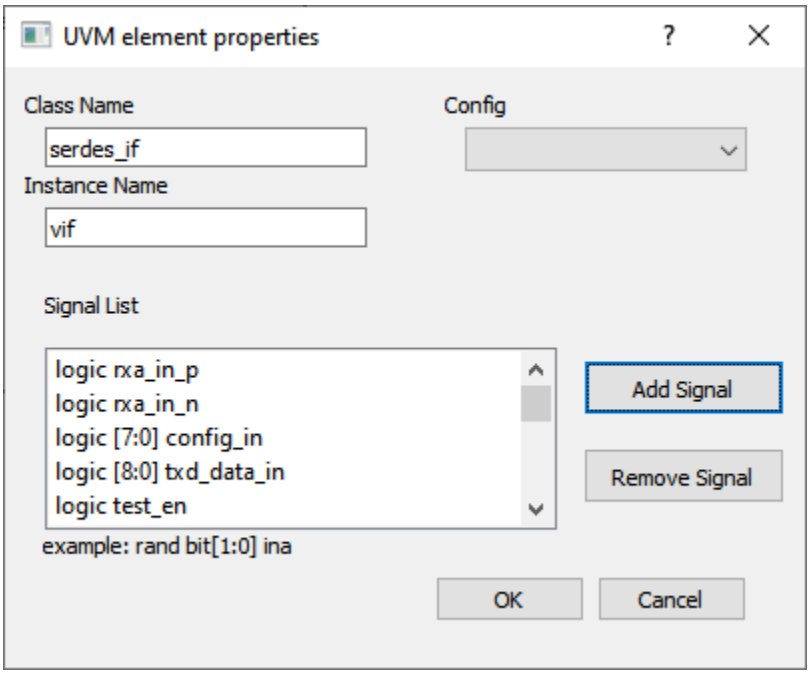


Figure 70 - SerDes IF properties menu

Interface code and diagram are shown in the following Figure 71.

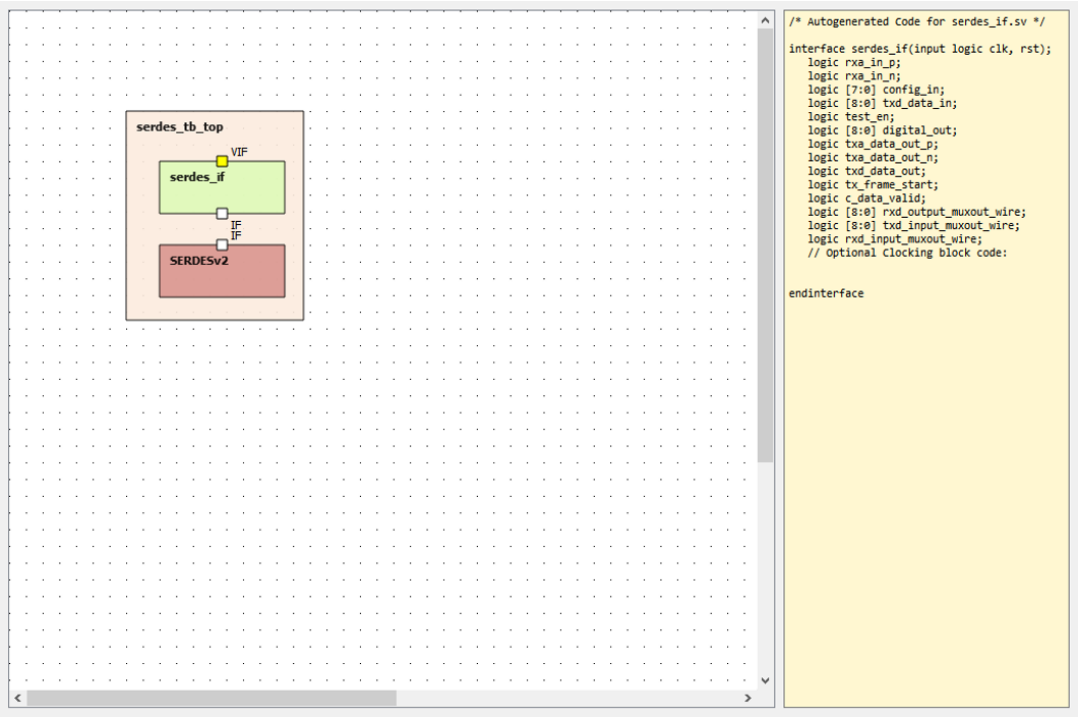


Figure 71 - Interface code displayed

Interface and Dut are connected using a wire and connection is created and added to the Top element code. This can be seen in Figure 72.

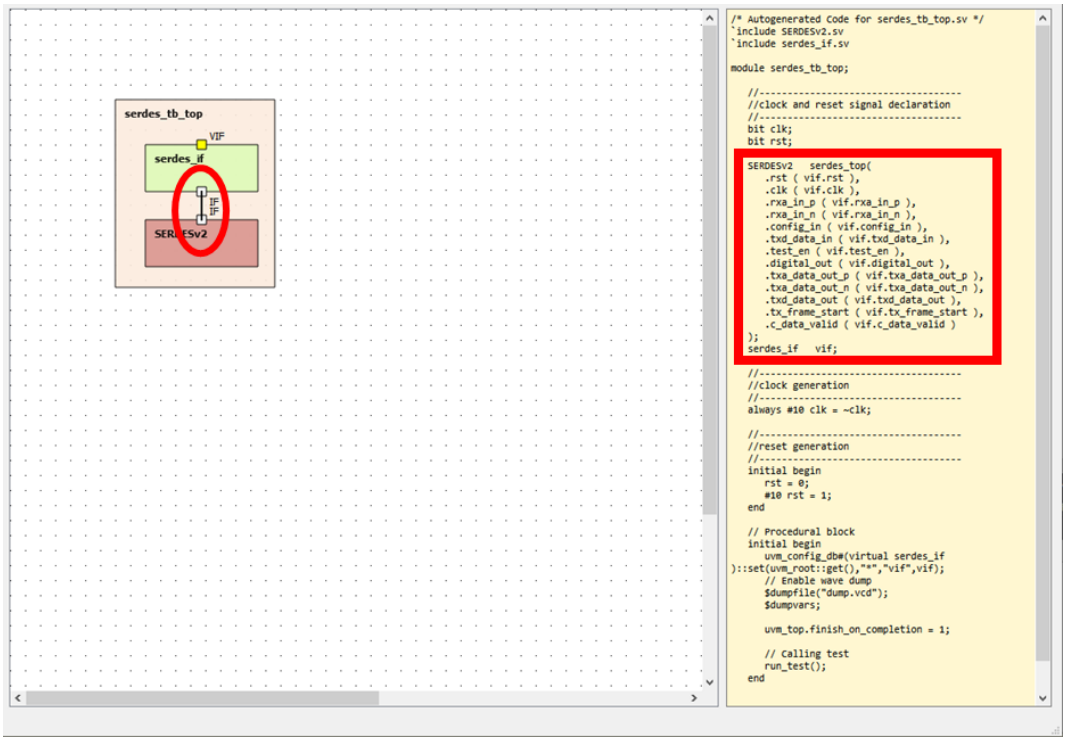


Figure 72 - SerDes Interface and DUT connection

One of the main changes in the TB generation starts at the test level. For the Test element, a new container is added. This component contains the Env and most of the rest of the VCs. However, this test will be a base class to generate the rest of the tests. In other words, all the tests that will be executed in the Validation section extend from this base test. The inheritance capability of the tool is demonstrated using this mechanism.

The first step is the addition of a new Test element to the Top module as shown in Figure 73.

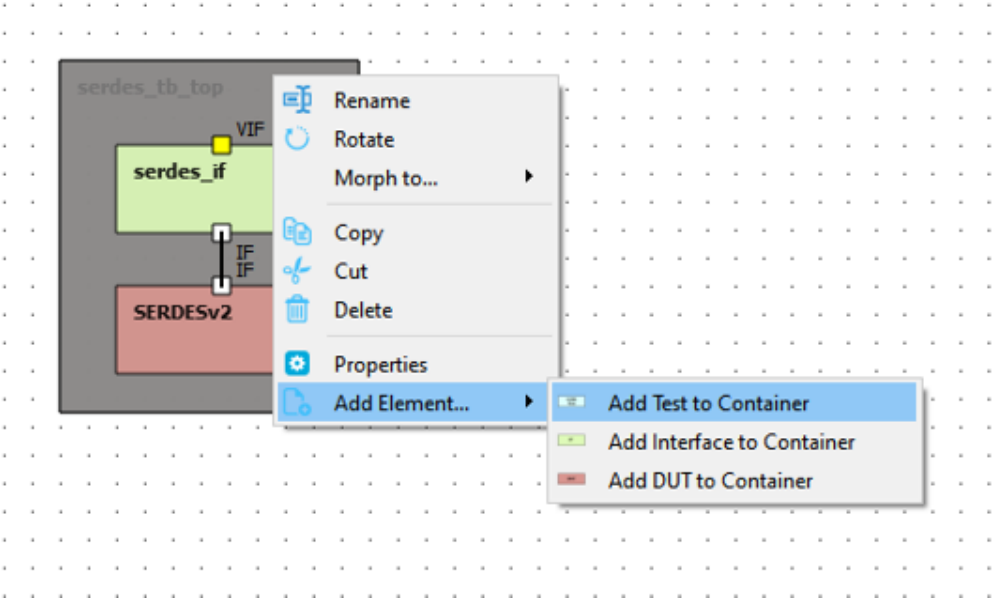


Figure 73 - Adding Test to SerDes TB_TOP container

This Test requires an Instance name. The sequence and signals are optional because this test will work only as of the base test. Furthermore, the Parent class field for this element is empty, meaning that the parent class to be used for this element is set directly from the UVM library as shown in Figure 74.

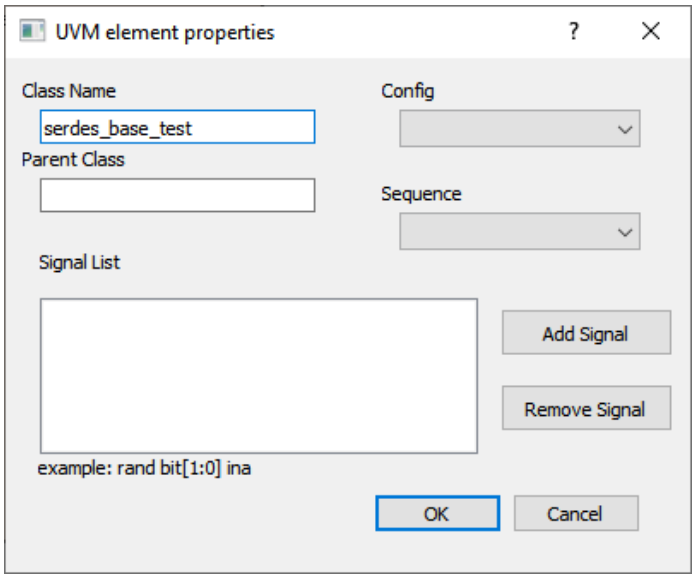


Figure 74 - SerDes base test properties menu

The first iteration of the Test code is shown in Figure 75.

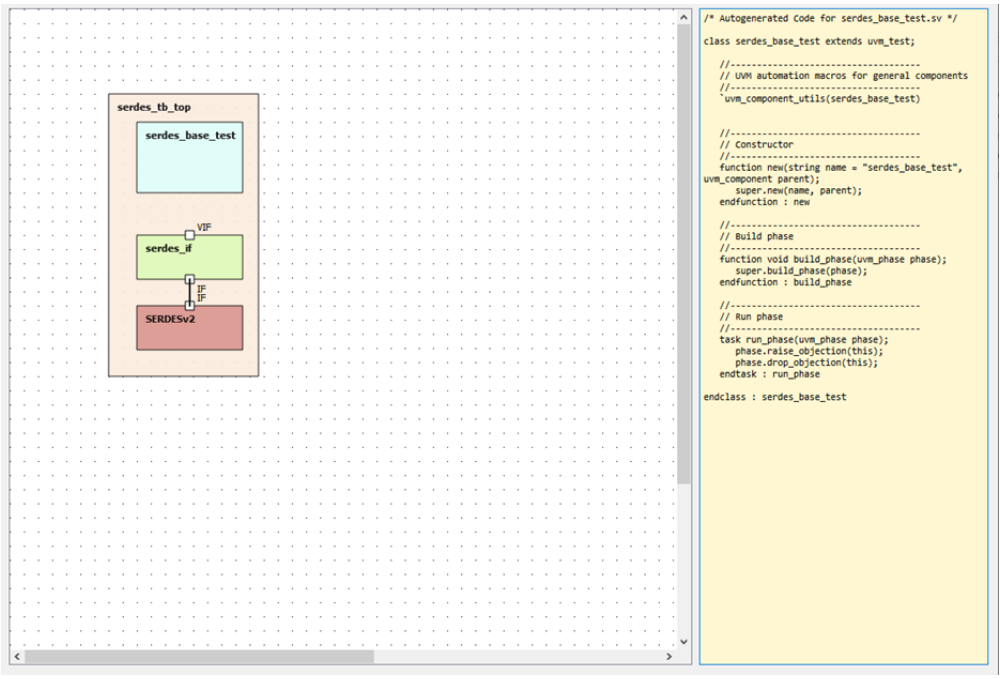


Figure 75 - SerDes base test code displayed

A Sequence element is then added to the Test as shown in Figure 76. This sequence will be the base sequence. The rest of the sequences will extend from this object.

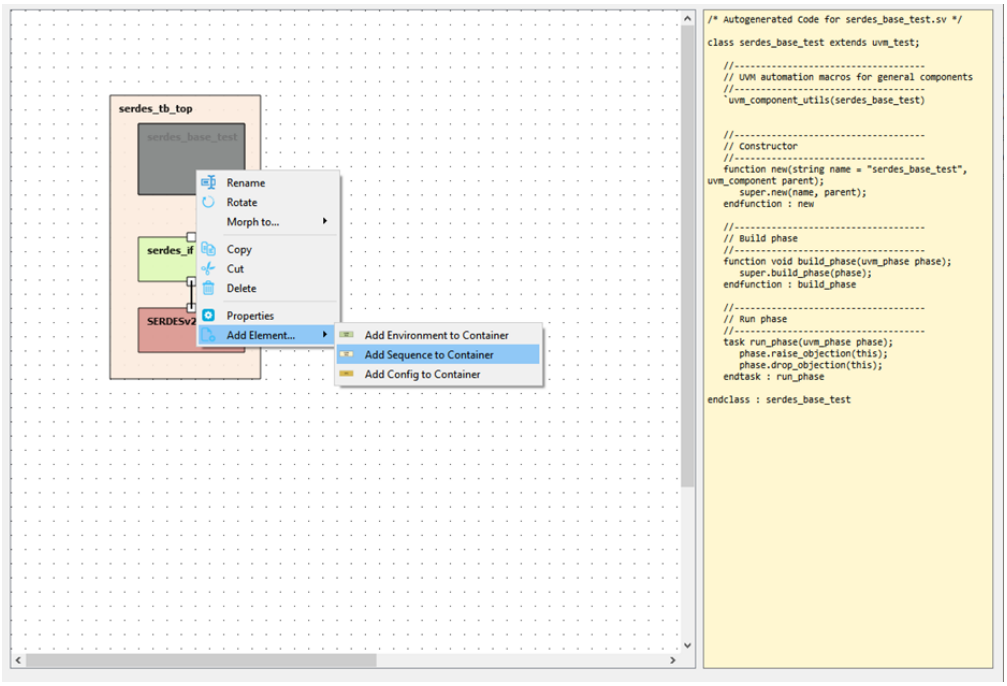


Figure 76 - Adding SerDes sequence to the base test

In this case, the only required field is the Instance Name. The properties menu of this component is shown in Figure 77.

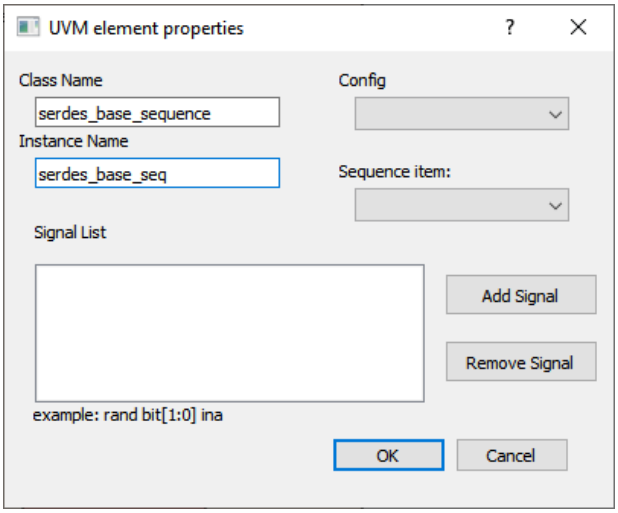


Figure 77 - SerDes base Sequence properties menu

The Sequence code and diagram are as follows in Figure 78.

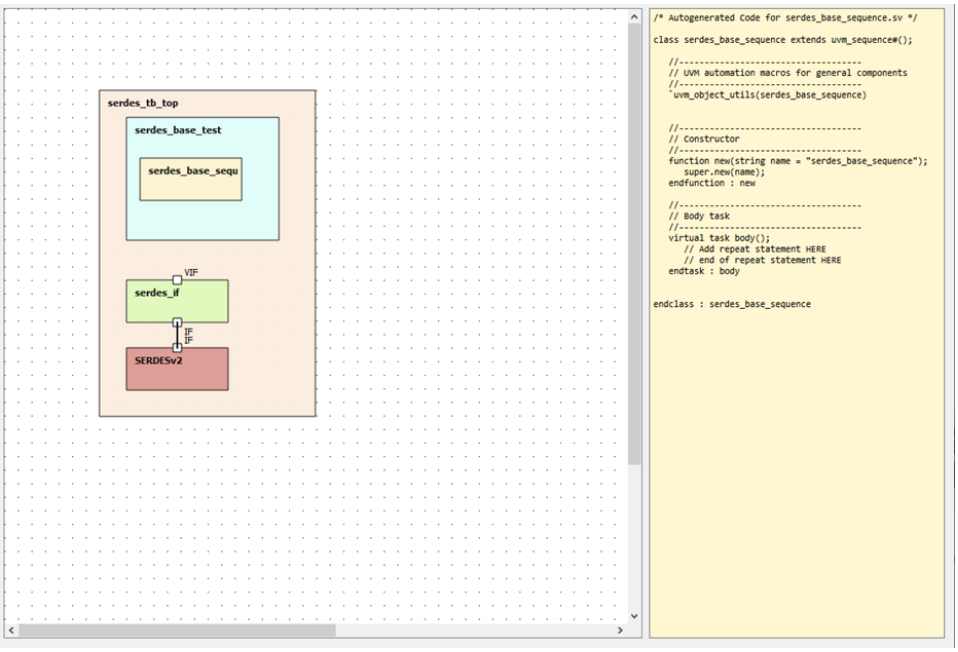


Figure 78 - SerDes sequence code and diagram

The Environment is then added to the Test as shown in Figure 79.

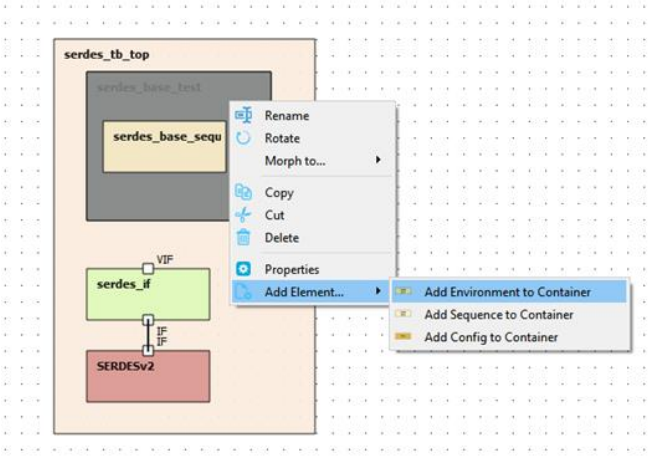


Figure 79 - Adding SerDes Env to the Test

This Env element requires an Instance name. Signal list is optional as shown in Figure 80.

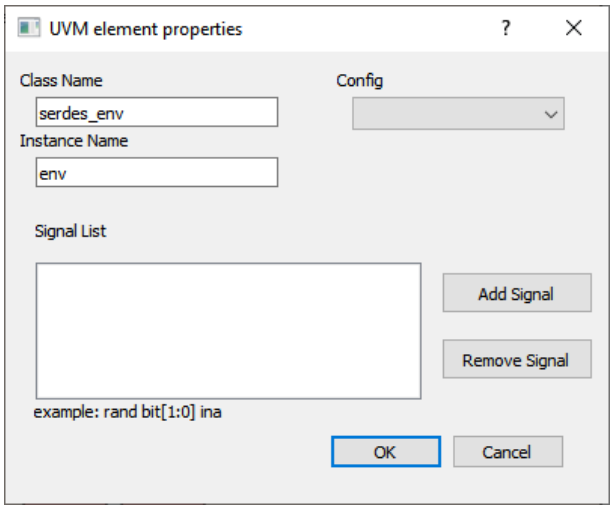


Figure 80 - SerDes Env properties menu

Env diagram and code is shown below in Figure 81.

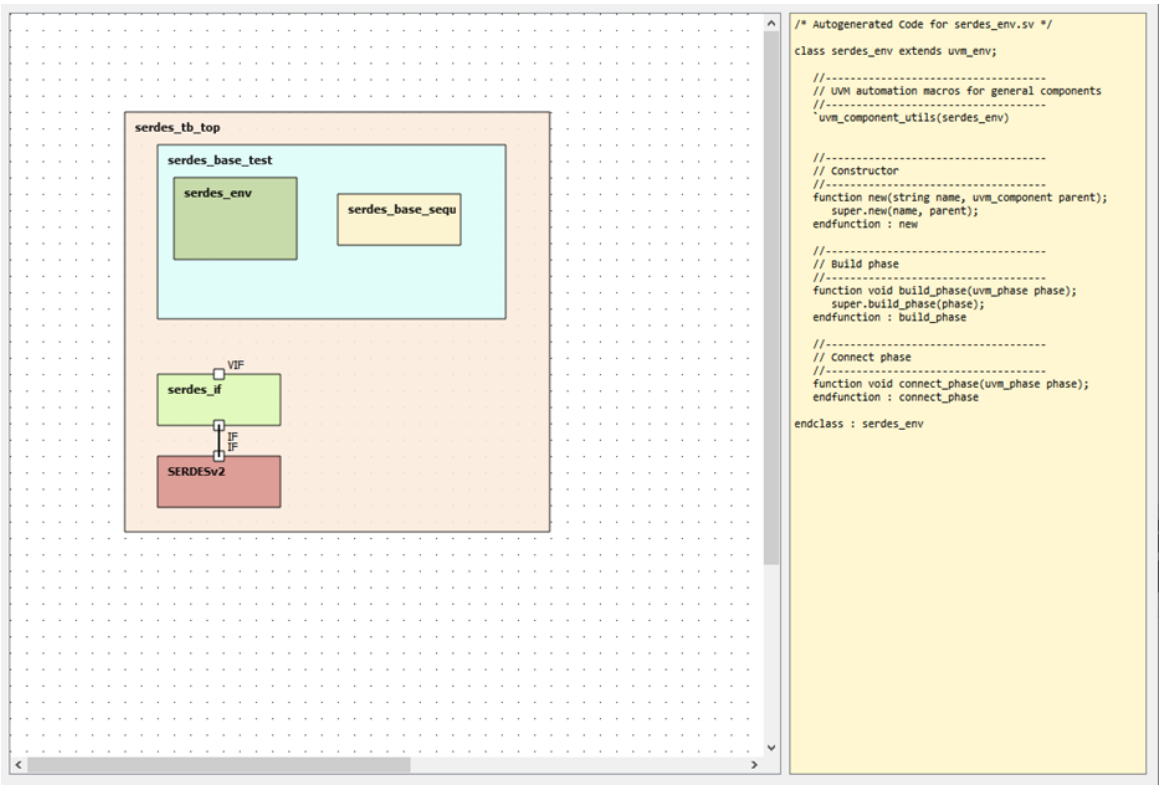


Figure 81 - SerDes Env diagram and code

The Sequence Item element is added to complete the basic objects required to model the SerDes transaction fields as shown in Figure 82.

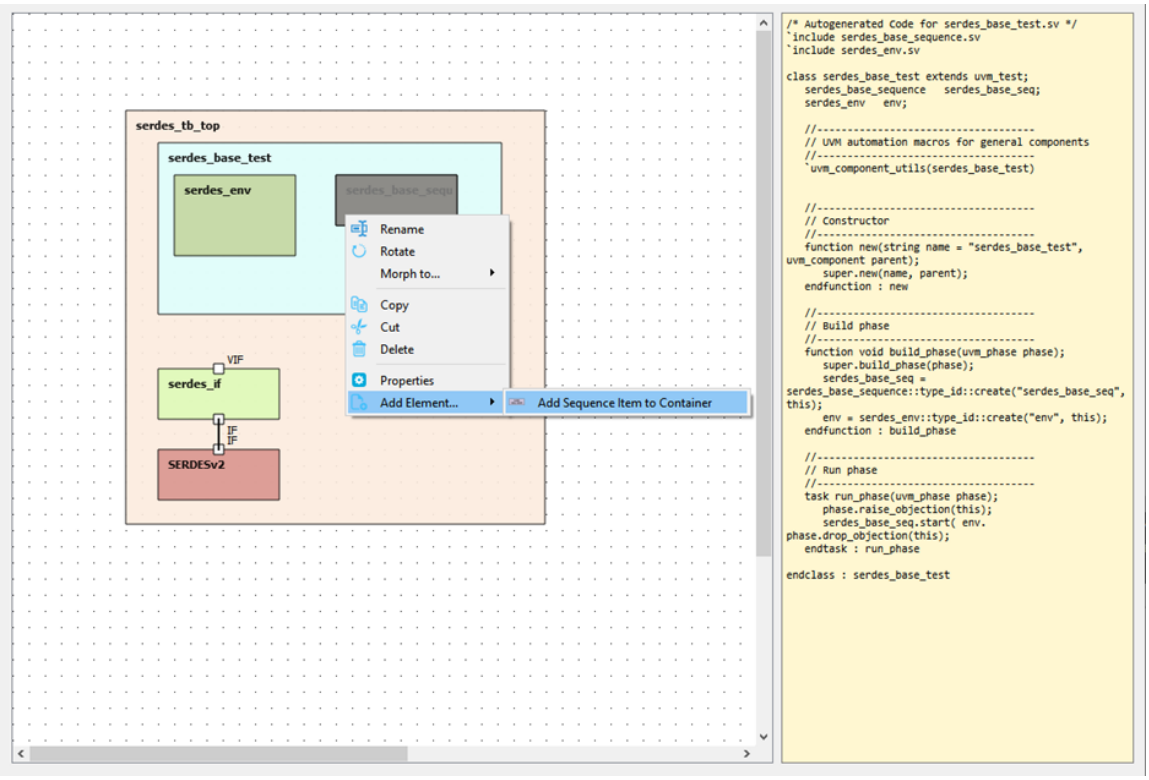


Figure 82 - Adding SerDes sequence Item to sequence

For the Sequence Item, the Class name field is required. The instance name is optional. The signal list is optional as well, however, the signal list will be filled with the required interface signals, adding the rand prefix to the signals that will be randomized or configured using the UVM utils macros as shown in Figure 83.

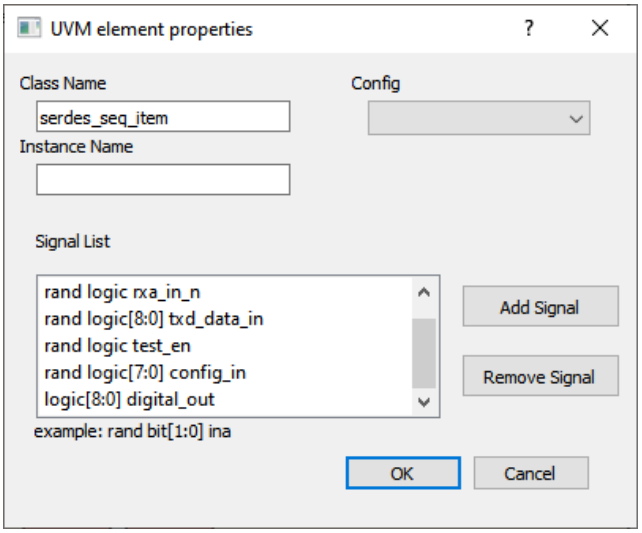


Figure 83 - SerDes sequence item properties menu

The Sequence Item diagram and code are shown in Figure 84.

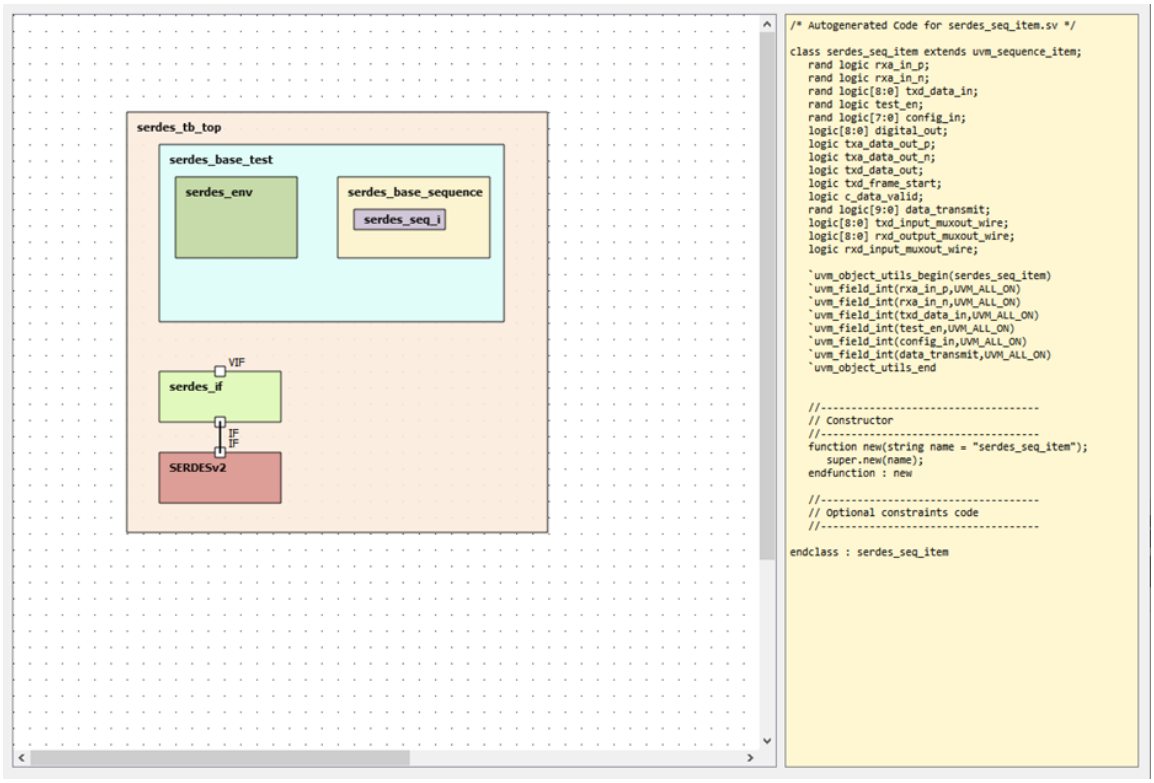


Figure 84 - SerDes sequence Item and diagram

Going deeper into the VCs hierarchy, another difference with respect to the ALU TB is demonstrated with the addition of the next elements.

Two separate Agent elements will be added at the Env hierarchy as shown in Figure 85.

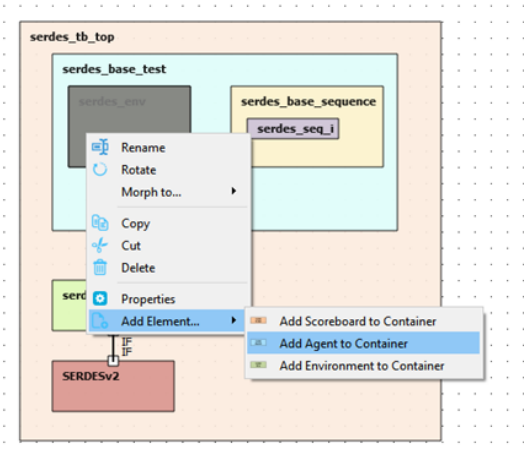


Figure 85 - Adding SerDes Agent to Env

One will be the TX Agent component in Figure 86 and the second one the RX Agent in Figure 87. This means that these components will have different behavior. The Class and the Instance name are required.

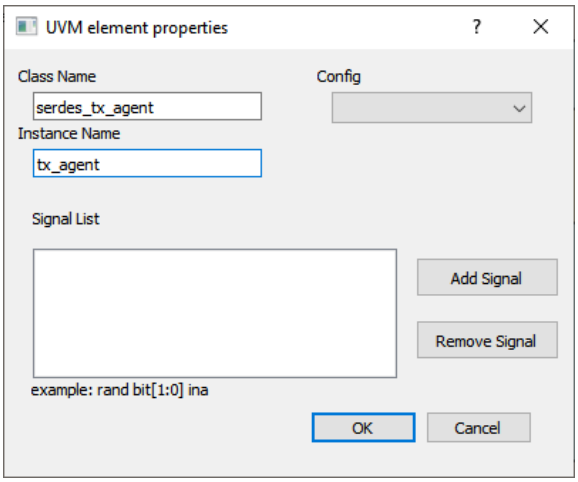


Figure 86- SerDes tx agent properties menu

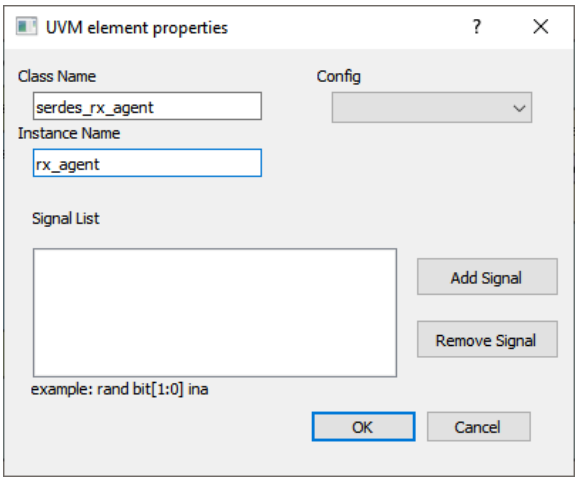


Figure 87 - SerDes RX agent properties menu

A single Scoreboard element is added to the Env. For the SerDes TB, one scoreboard will be used to facilitate the comparisons between the transactions received and sent from/to the SerDes DUT. Users can add as many scoreboards as required, but this depends entirely on the user's needs as shown in Figure 88.

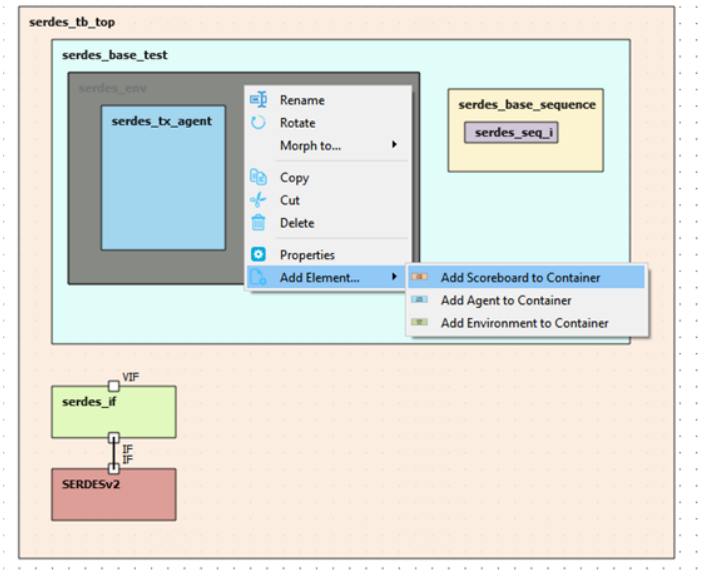


Figure 88 - Adding Scoreboard to SerDes base test

The Scoreboard requires a Class and Instance name. The Sequence item is selected by default as shown in Figure 89.

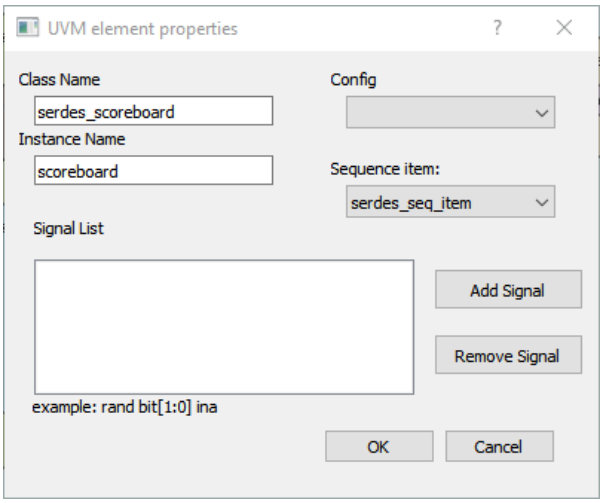


Figure 89 - SerDes scoreboard properties menu

The Scoreboard code is shown below in Figure 90.

```
/* Autogenerated Code for serdes_scoreboard.sv */

class serdes_scoreboard extends uvm_scoreboard;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(serdes_scoreboard)

   //------------------------------------
   // Declaring port to receive packets
   //------------------------------------
   uvm_tlm_analysis_fifo#(serdes_seq_item)item_col;
   serdes_seq_item seq;

   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      item_col = new("item_col",this);
   endfunction : build_phase

   //------------------------------------
   // Run phase
   //------------------------------------
   virtual task run_phase(uvm_phase phase);
      forever begin
         item_col.get(seq);
         // Fill with Scoreboard processes HERE
      end
   endtask : run_phase

endclass : serdes_scoreboard
```

Figure 90 - SerDes Scoreboard auto-generated code

The Env diagram and code with its elements are represented as follows in Figure 91.

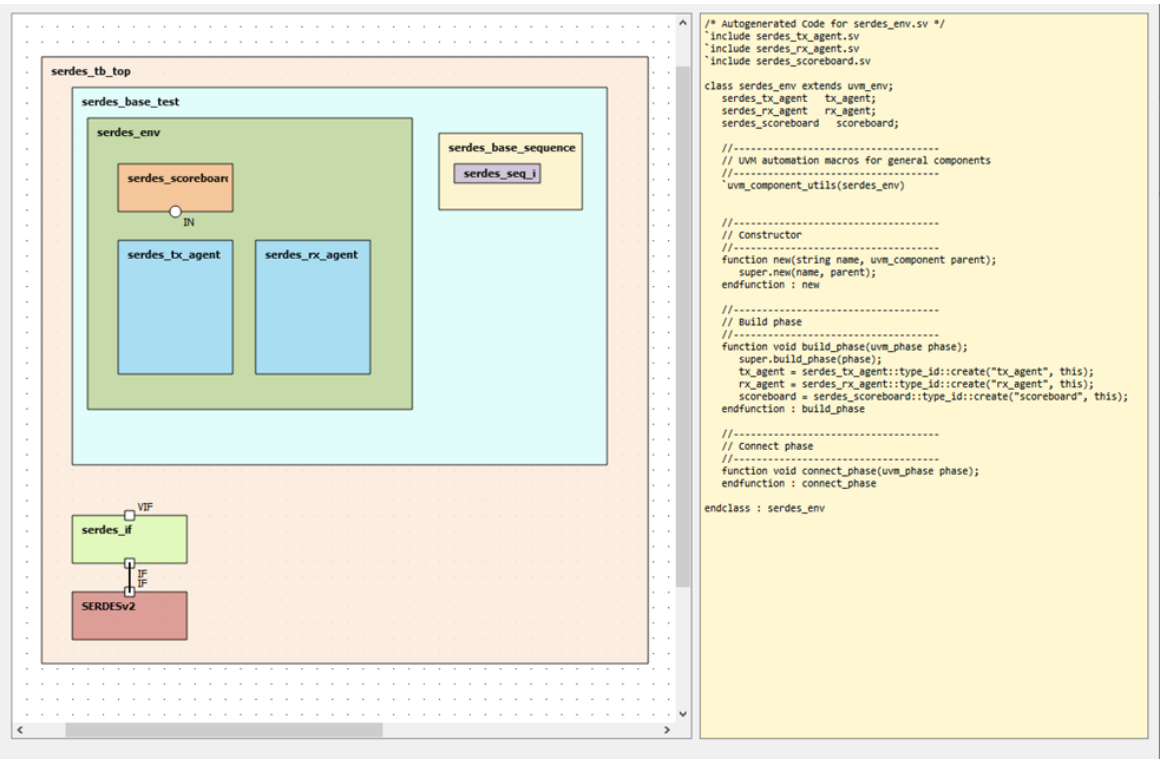


Figure 91 - SerDes Env code and diagram

The next step consists of the creation of the internal VCs that are part of the Agents.

A monitor is added as a first element to create the agent as shown in Figure 92.

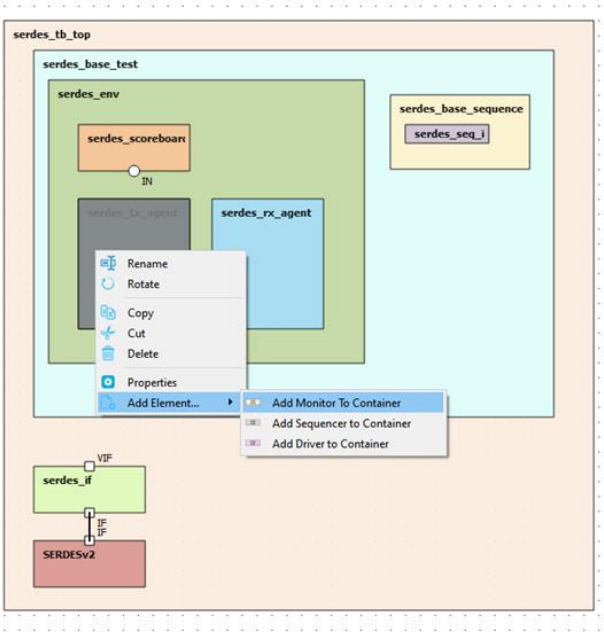


Figure 92 - Adding a monitor to SerDes Tx Agent

The monitor only requires a Class and Instance name. Sequence item is added by default. The signal list is optional as shown in Figure 93.

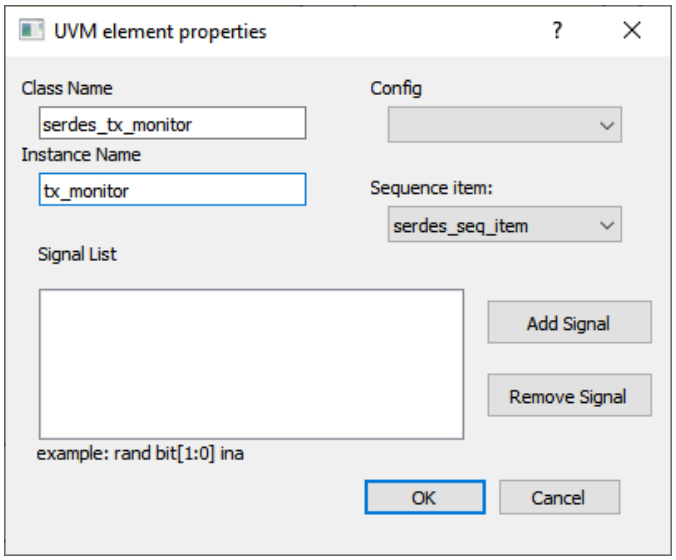


Figure 93 - SerDes tx monitor properties menu

The Monitor code is shown below in Figure 94.

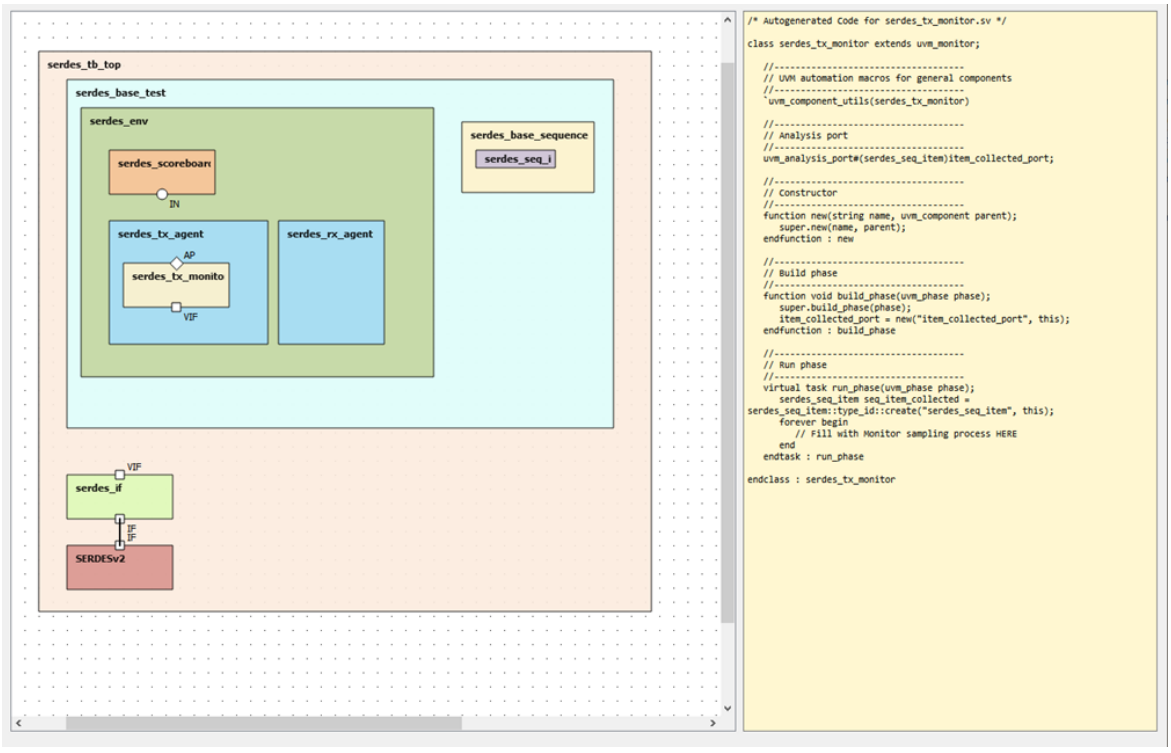


Figure 94 - SerDes tx monitor code and diagram

The second element of the TX agent is a Sequencer as shown in Figure 95.

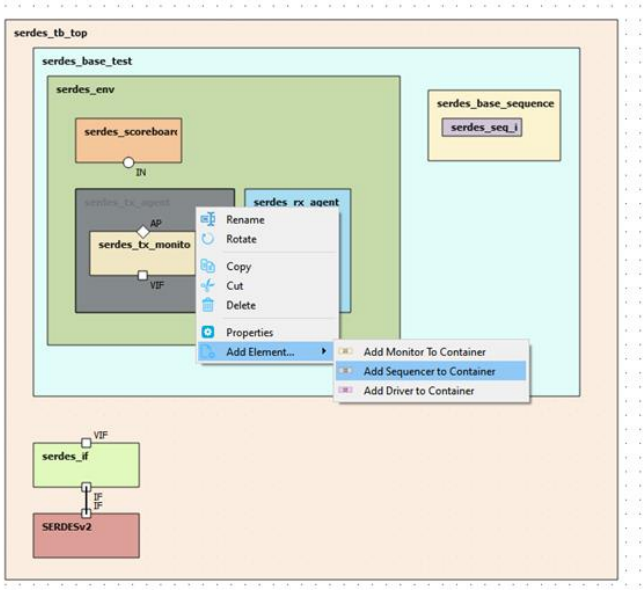


Figure 95 - Adding Sequencer to SerDes TX Agent

In a similar way as in the Monitor, the Sequencer requires a Class and Instance name. The Sequence item is selected by default. The signal list is optional as shown in Figure 96.

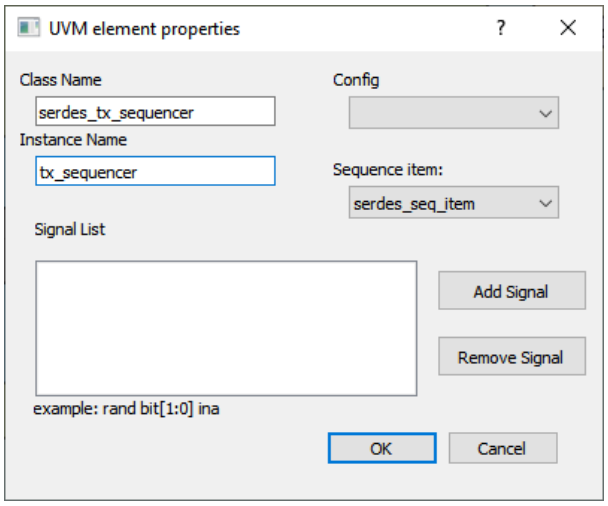


Figure 96 - SerDes Tx sequencer properties menu

The Sequencer diagram and code are shown in Figure 97.

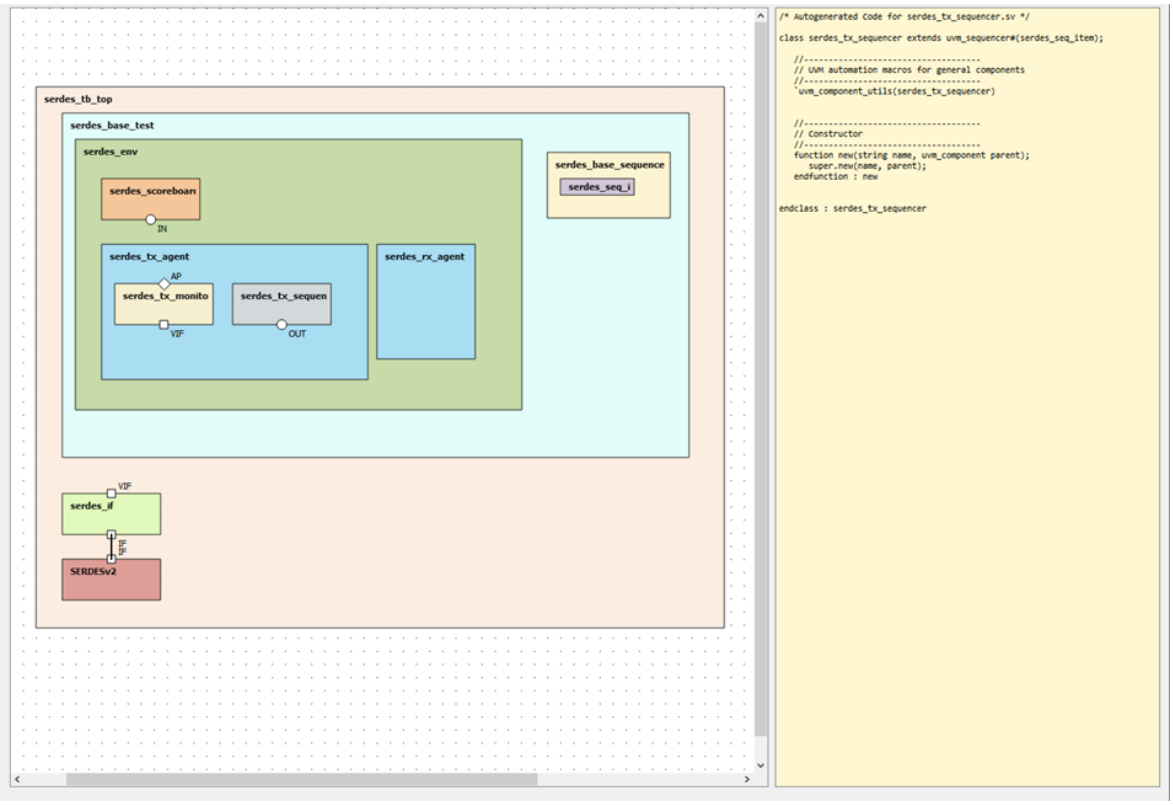


Figure 97 - SerDes TX sequencer code and diagram

To complete the TX Agent, a driver is added to the container as shown in Figure 98.

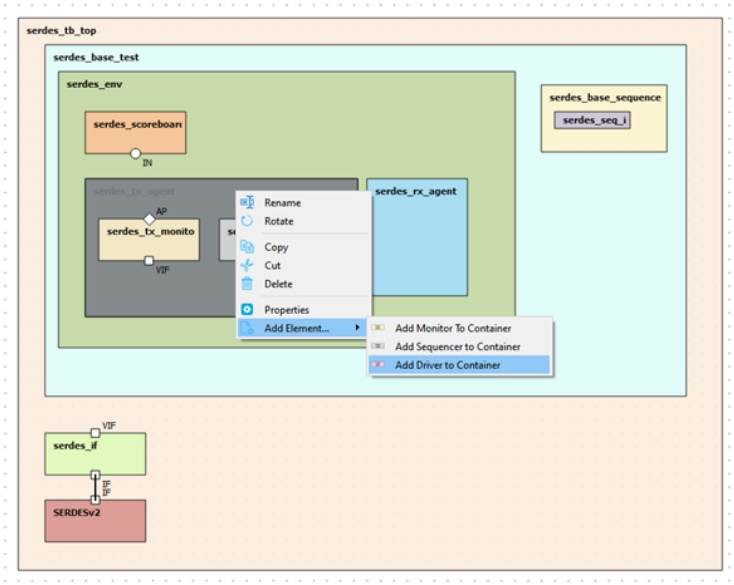


Figure 98 - Adding Driver to SerDes TX agent

The Driver requires a Class and Instance name. Sequence item is added by default. The signal list is optional as shown in Figure 99.

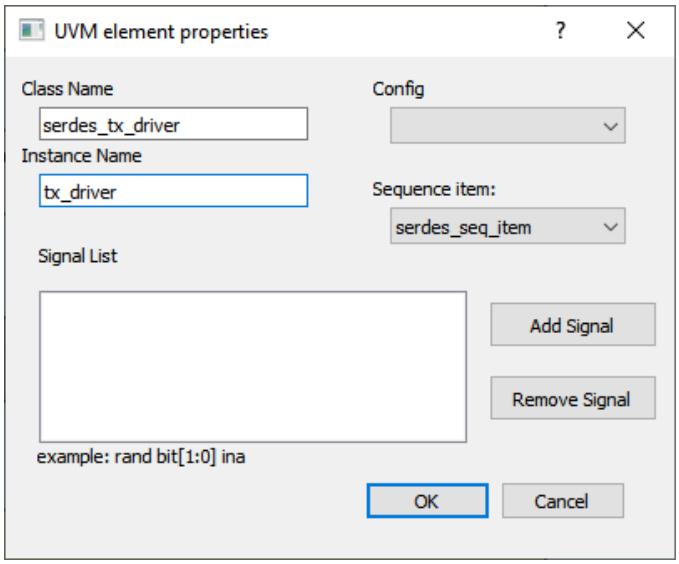


Figure 99 - SerDes TX driver properties menu

Driver diagram and code are shown in Figure 100.

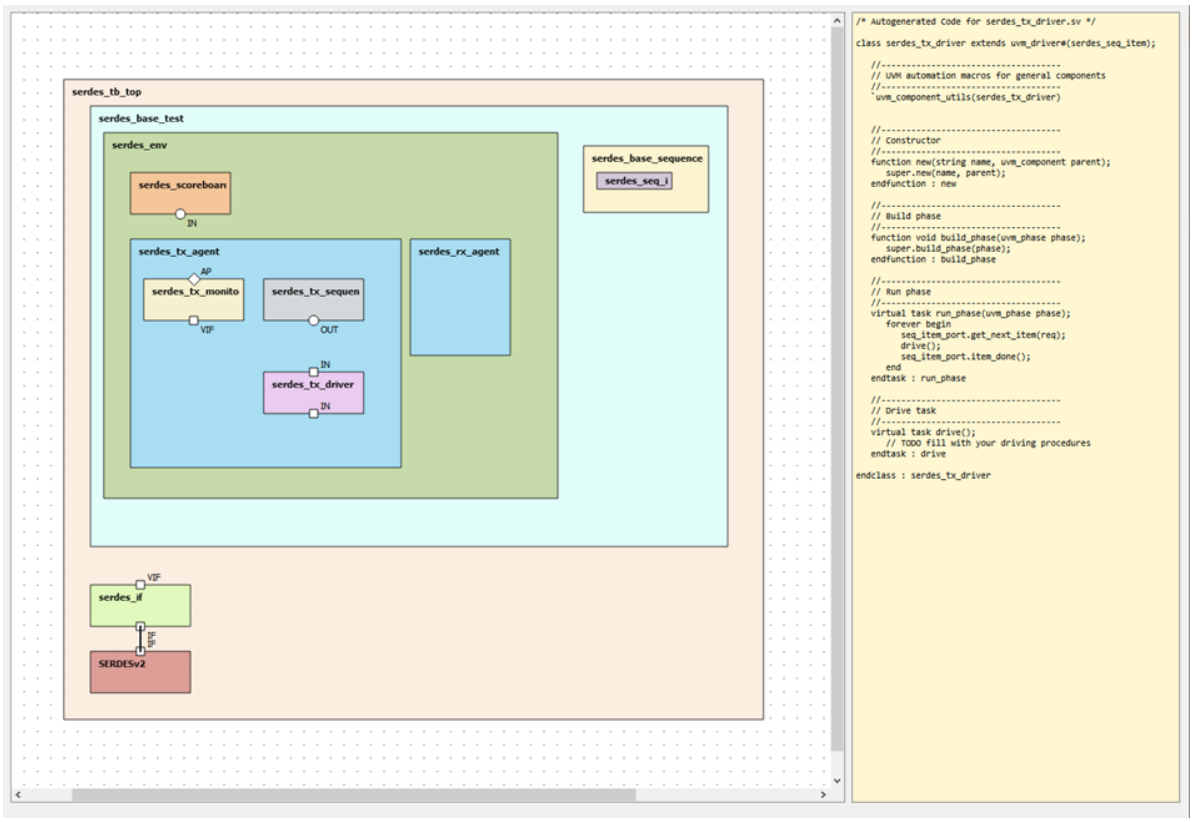


Figure 100 - SerDes TX driver code and diagram

The TX Agent is now complete as shown in Figure 101.

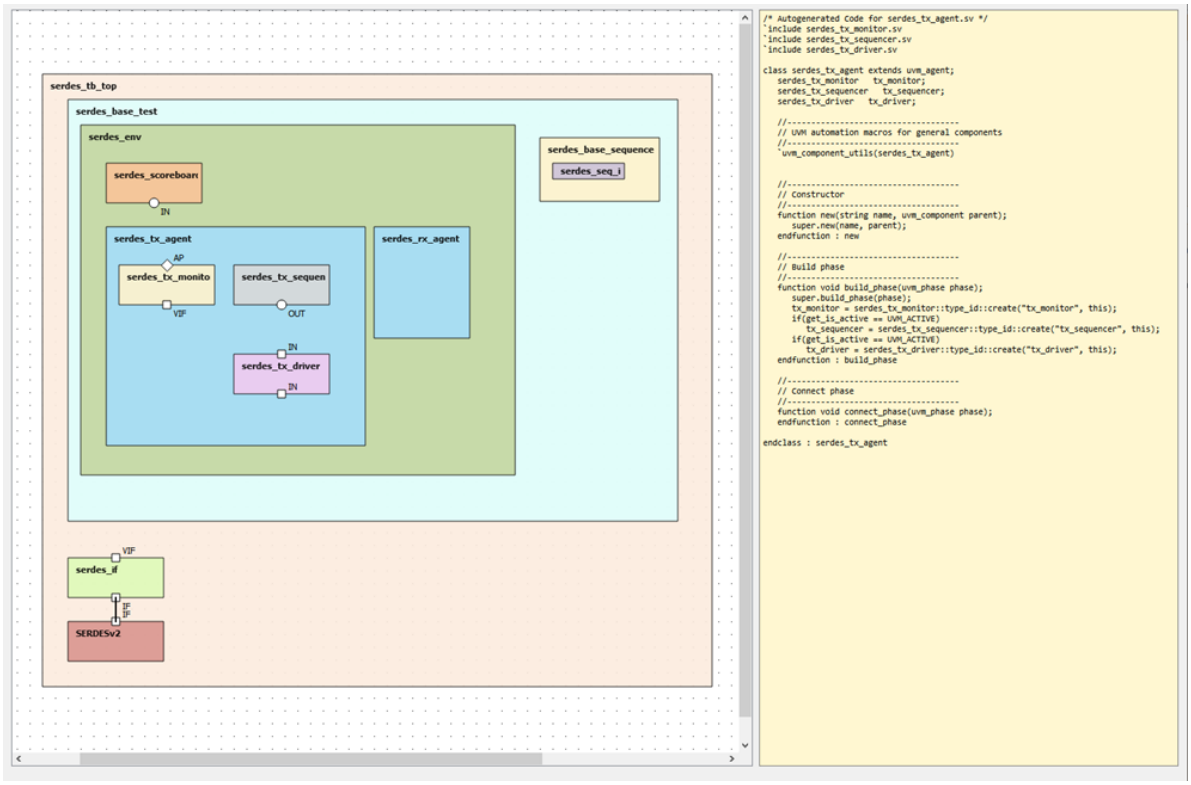


Figure 101 - SerDes TX Agent

Internal elements from this container can be copied from the TX Agent to generate the internal components of the RX Agent. Using this procedure and replacing the prefix “serdes_rx” with “serdes_tx” easily generates the elements that complete the RX Agent as shown in Figure 102.

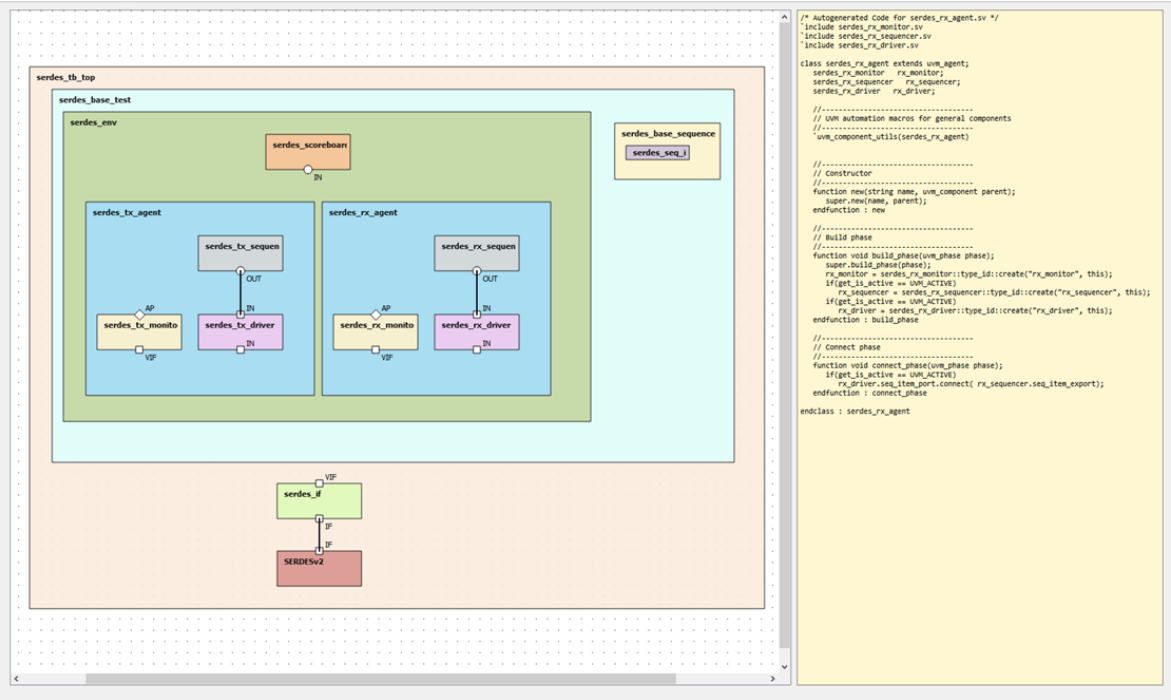


Figure 102 - SerDes RX and TX agents

All the base VCs required to complete the SerDes TB are in the scene. The final step to complete the code of the UVM components is the definition of the connections as shown in Figure 103.

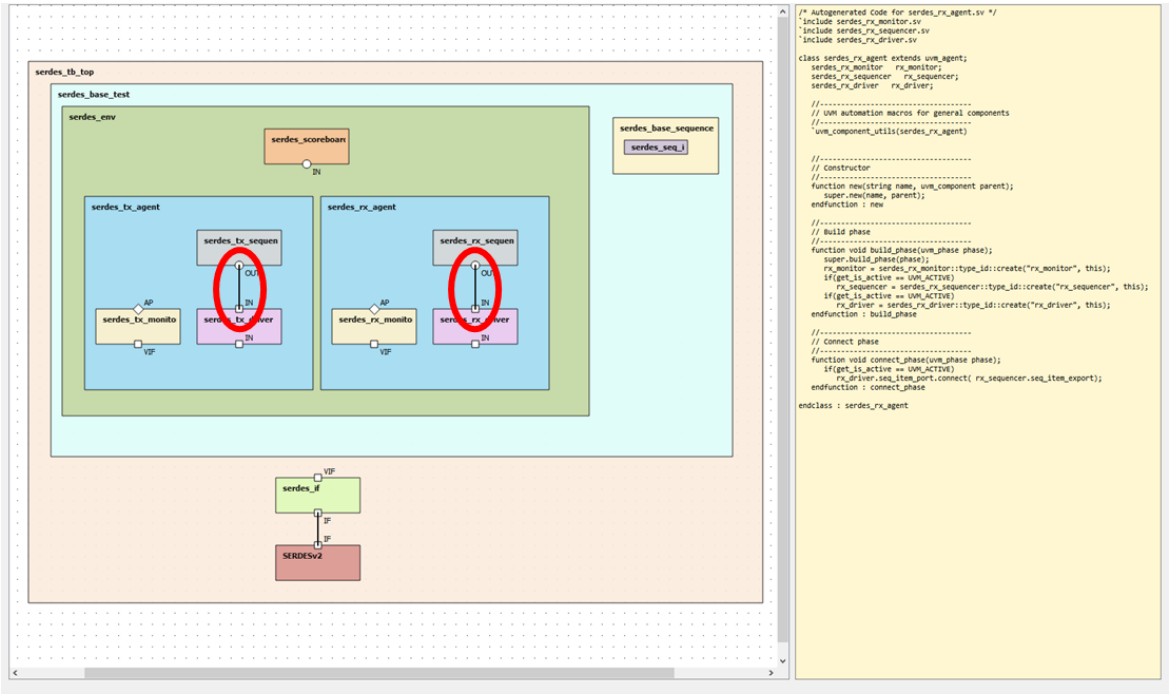


Figure 103 - SerDes TX and RX sequencers connections to their drivers

Internal TX Agent connection between TX Sequencer and TX Driver is demonstrated in the image below in Figure 104.

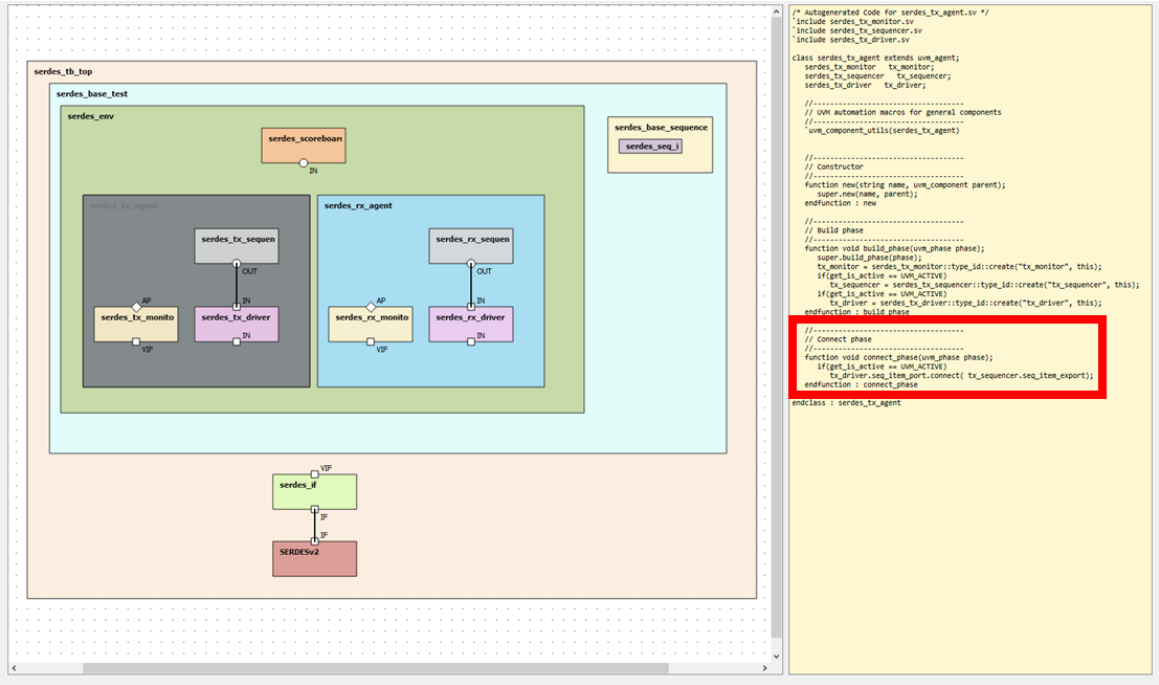


Figure 104 - SerDes TX Agent connection code displayed

The connections between internal RX Agent elements (RX Sequences and RX Driver) are shown in the following Figure 105.

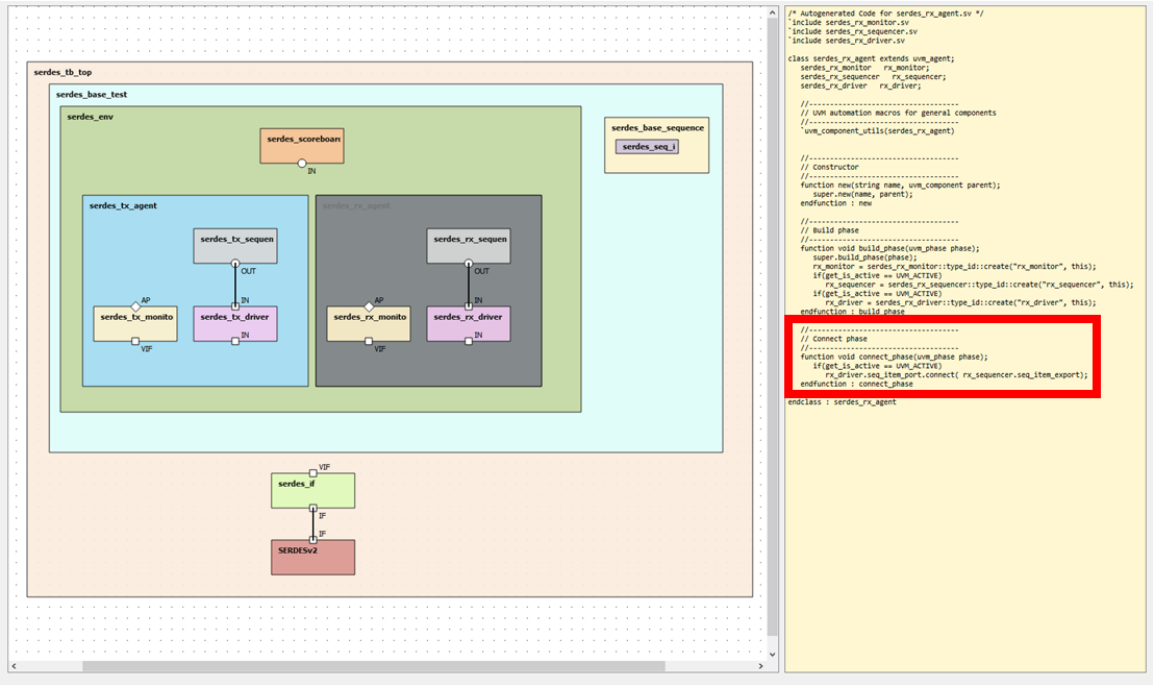


Figure 105 - SerDes RX Sequence and RX Driver connection code displayed

The connection between Interface and TX Driver is shown in the diagram below. The code updates in the driver are shown as well in Figure 106.

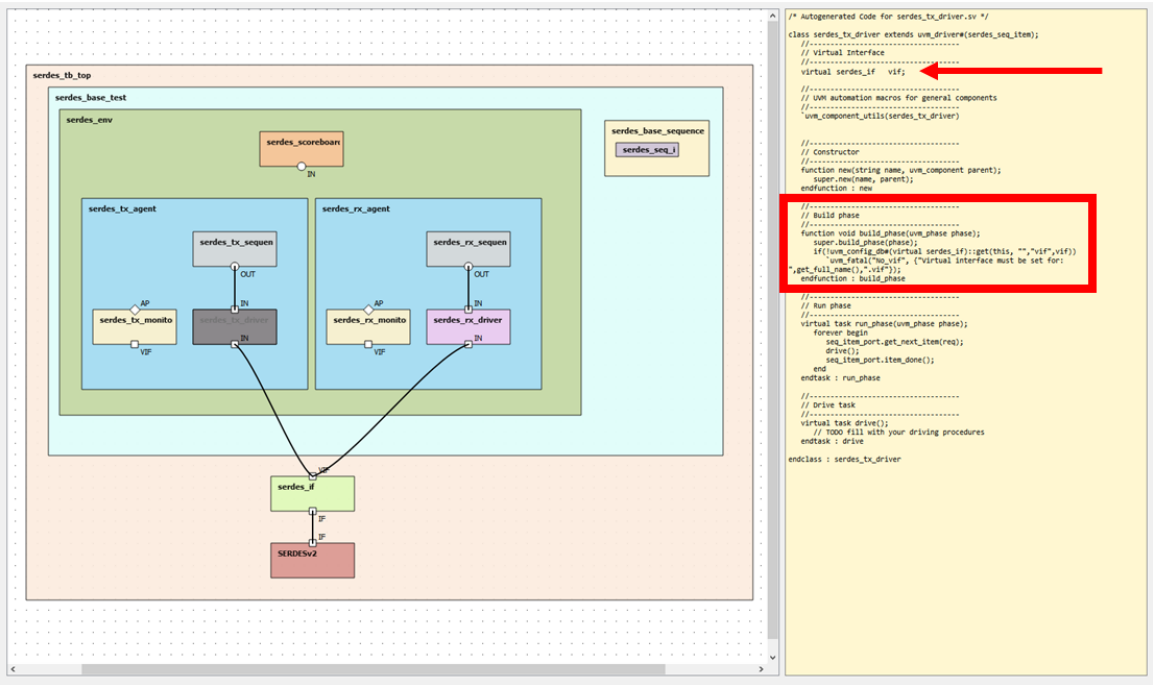


Figure 106 – SerDes TX Driver connection to the IF

The connection between Interface and RX Driver is shown in the diagram below. Code

updates in the driver are shown as well in Figure 107.

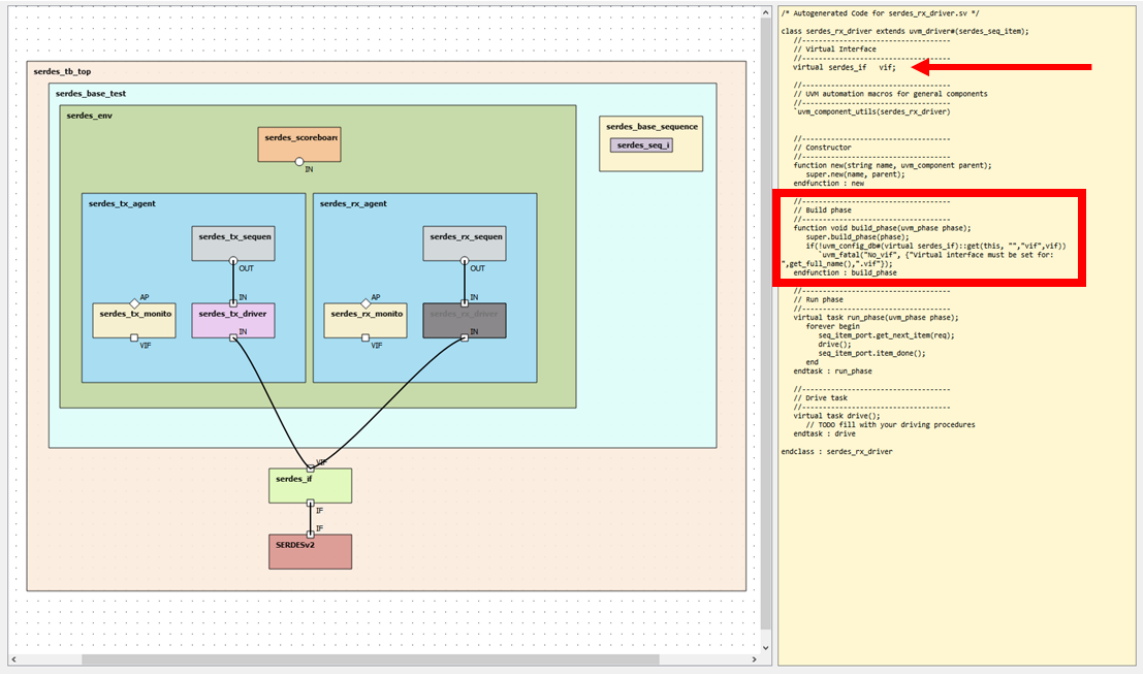


Figure 107 – SerDes RX Driver connection to the IF

A similar interconnection is required for the monitors. TX Monitor connection diagram and code are shown in Figure 108.

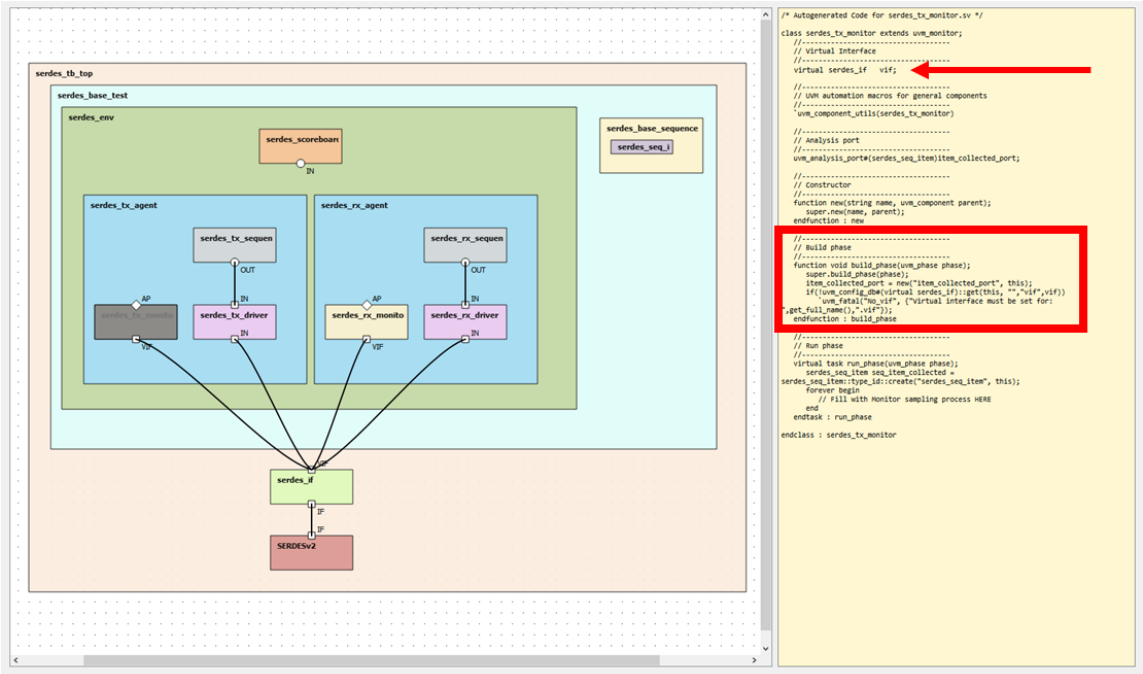


Figure 108 - SerDes TX monitor's connection to the IF

RX Monitor connection is shown in Figure 109.

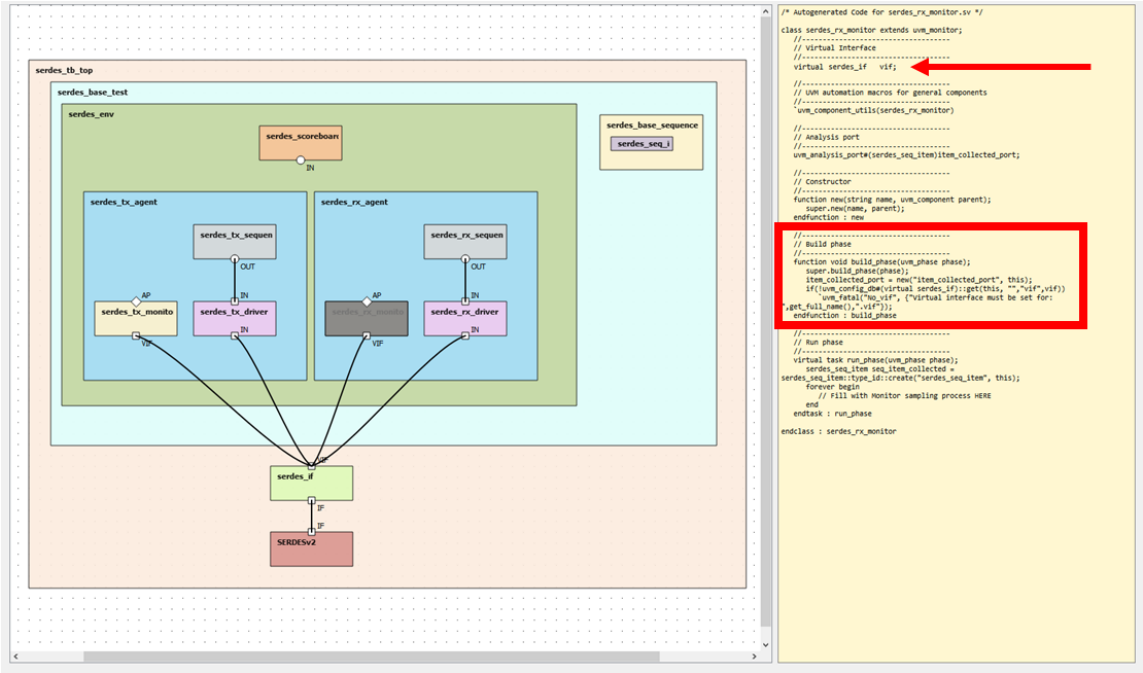


Figure 109 - SerDes RX monitor´s connection to the IF

The last connection is made between the Scoreboard and Monitors. To create this connection, two separate inputs are required to handle the incoming transactions from the different monitors (TX and RX).

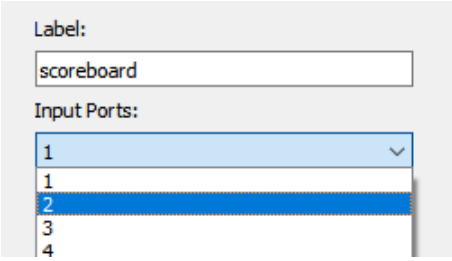


Figure 110 - SerDes Adding extra ports to the Scoreboard

Then, a new port is added to the Scoreboard as is shown in Figure 110. A new set of wires is also added from each monitor to their corresponding proper port.

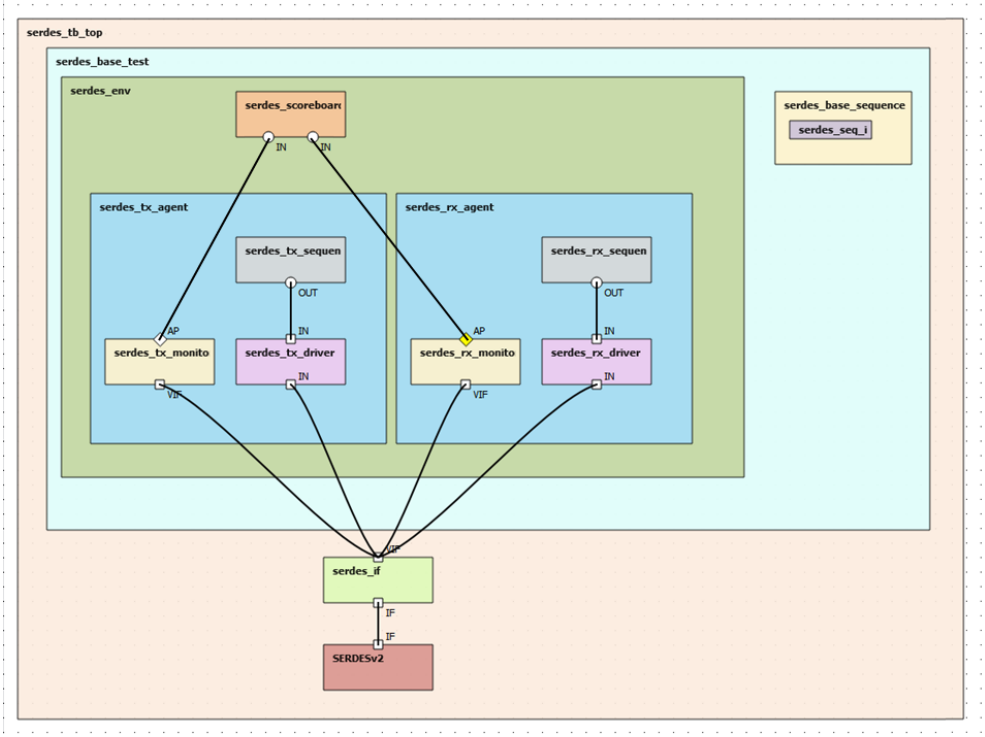


Figure 111 - SerDes Scoreboard connection to the monitors

With the Scoreboard connections, all the TB connections and VCs are complete as showcased in Figure 111.

So far, the TB can execute the base test. This test instantiates and builds all the VCs of the Environment and can inject a base sequence as well. For the ALU TB, this structure was fair enough to complete an adequate validation. However, the SerDes TB requires a more complex infrastructure due to the wide variety of stimuli that can be injected into the DUT.

The tool offers a powerful way to model inheritance in a graphical TB model. To demonstrate this feature applied to the SerDes TB, a set of tests are added and included in the testplan.

Starting from the Top element, a new Test is added as shown above. It is important to recall that a Test element requires a Class name but no Instance name. Now, a new field identified as “Parent Class” is incorporated into the properties menu and it contains the parent object of the current element. For this new test called *serdes_parallel_loopback_test* the *serdes_base_test* attributes are inherited, so the properties menu is filled with the corresponding information as shown in Figure 112 below.

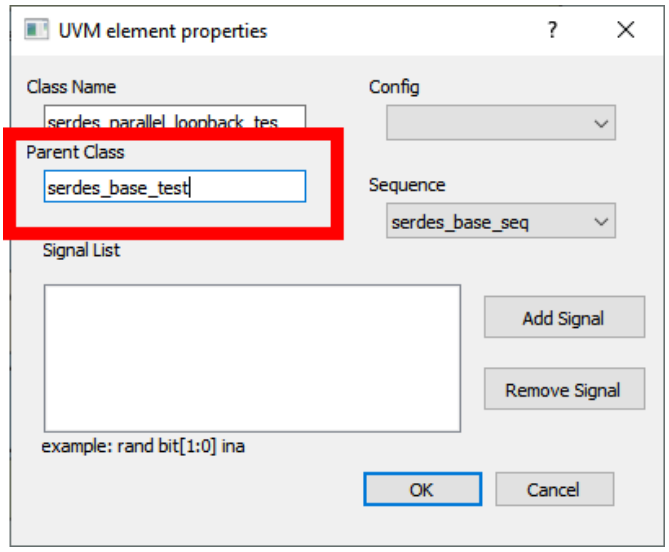


Figure 112 - SerDes base child test properties menu

In a similar way as in the previous cases, a Sequence element is added to the Test, configured as follows.

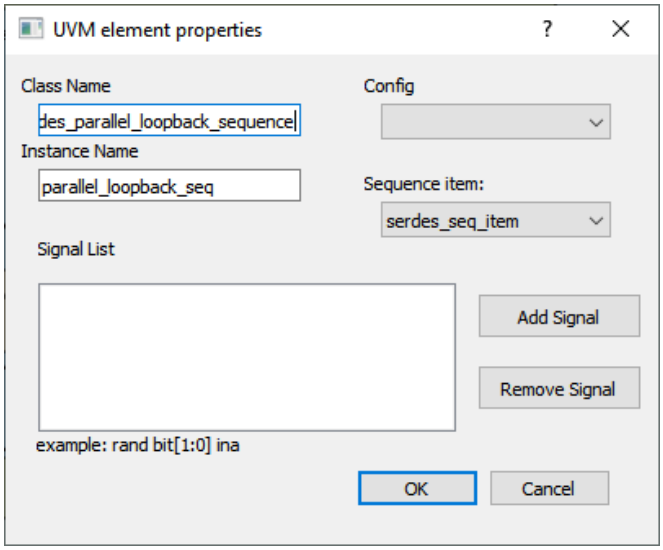


Figure 113 - SerDes child test properties menu

In this example, there is no need to add a different Sequence Item element, but the User can add another one if required as shown in Figure 113.

Going back to the new Test added, the corresponding Sequence is specified from the drop-down list as shown in Figure 114.

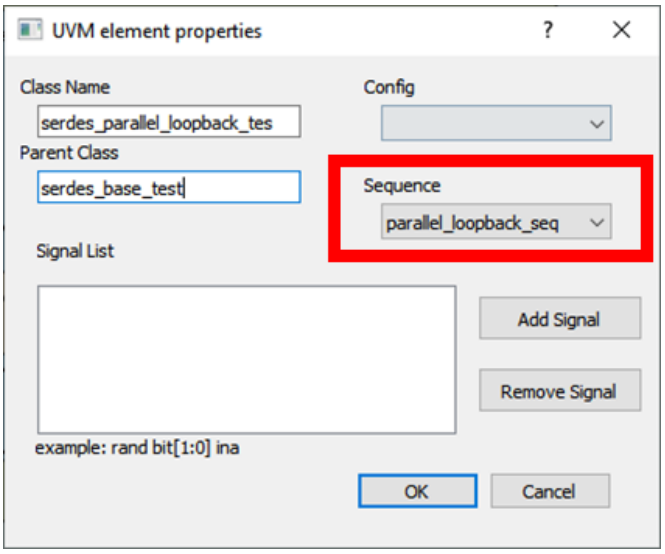


Figure 114 – SerDes Child Test dropdown menu

So, the diagram now shows at the Top hierarchy a new Test that seems to be apart from the serdes_base_test as shown in Figure 115. However, the code shows clearly the extends keyword, indicating that the base test inherits its properties and infrastructure capabilities to the newly generated serdes_parallel_loopback test.

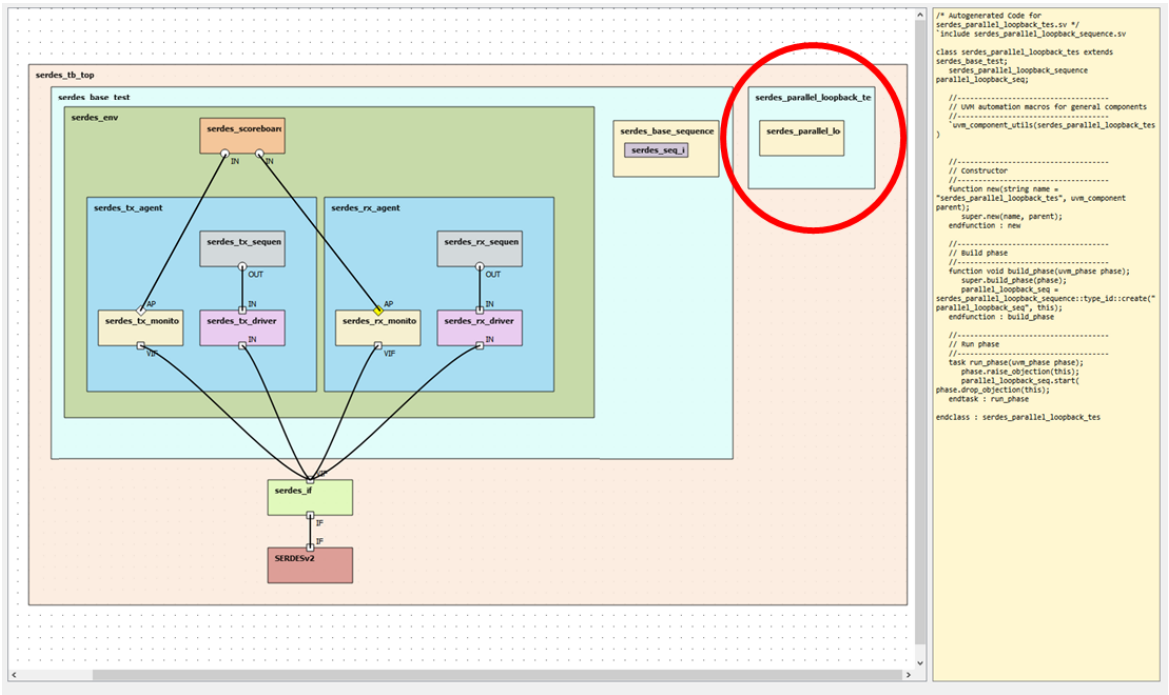


Figure 115 - SerDes Diagram with parallel loopback child test

Similarly, a second test for the serial loopback test mode is added as shown in Figure 116.

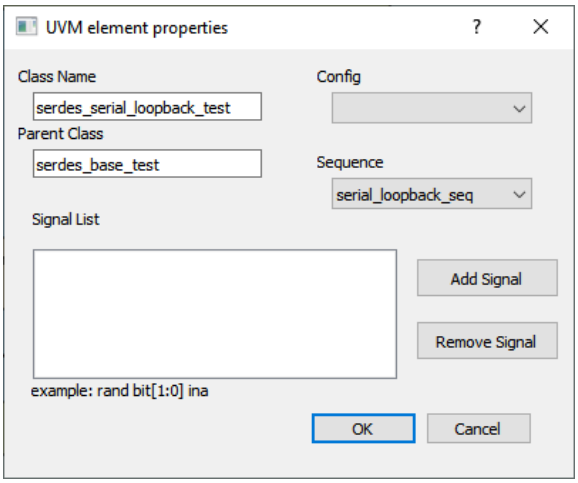


Figure 116 - SerDes serial loopback test properties menu

Again, the diagram in Figure 117 reflects the newly added Test and Sequence elements.

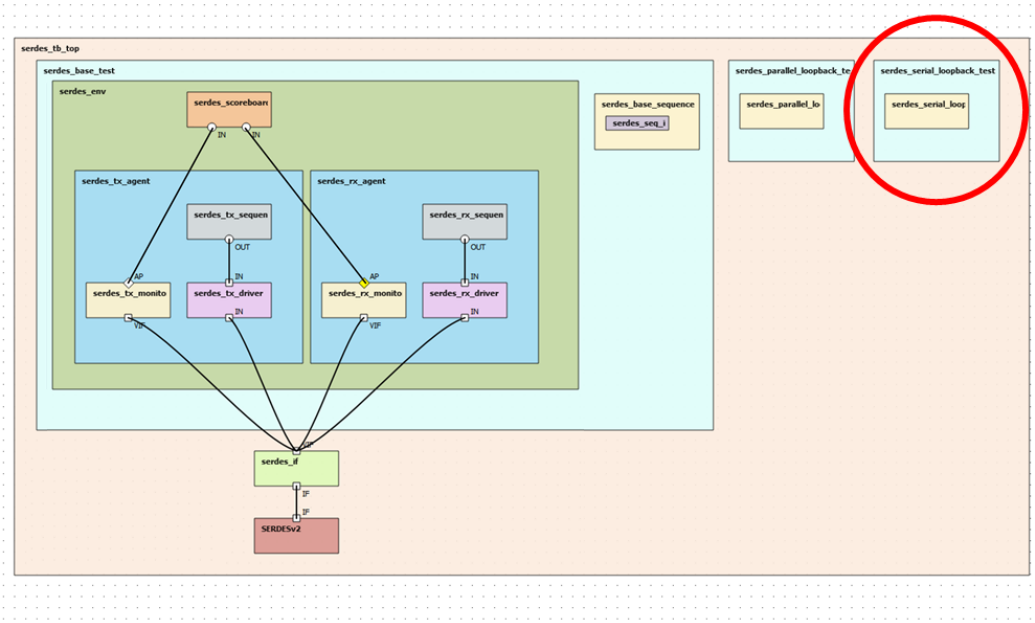


Figure 117 - SerDes TB diagram with two child tests

Following the same procedure, the rest of the Tests are added to complete the test library required for the SerDes validation.

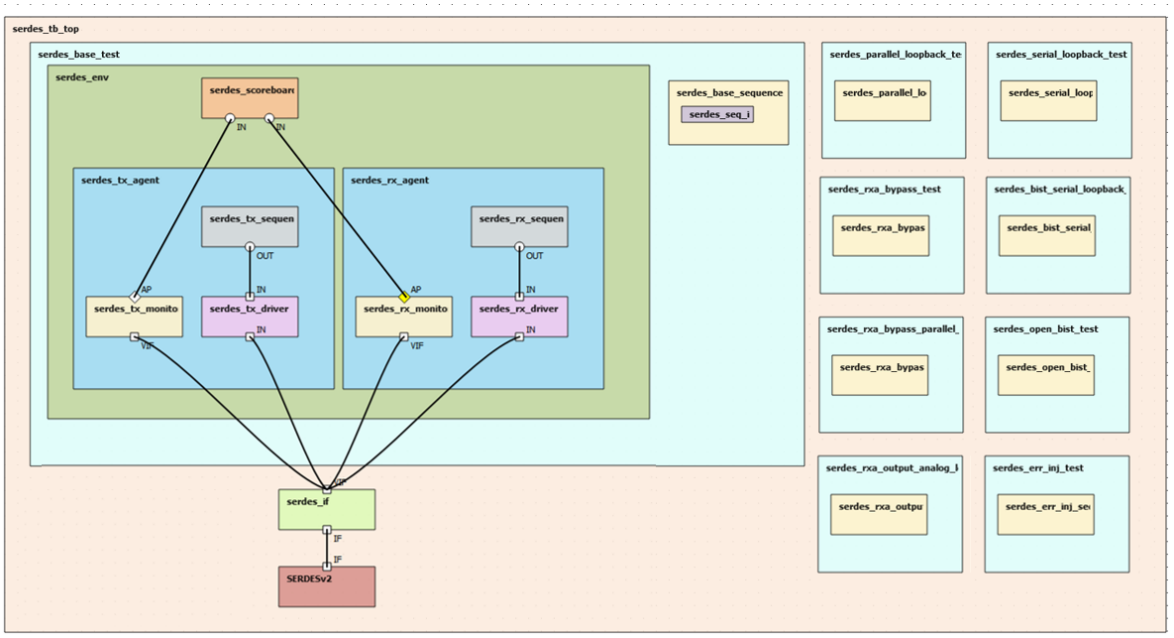


Figure 118 - SerDes TB diagram with full test suite

A full suite of tests and sequences are now added to the scene, thus, its code modeled by the diagram is also generated as shown in Figure 118.

Something relevant is that this Test library visual representation is a novel approach that was not found in the State of the Art. At the same time, the graphic representation of the TB maintains its clarity and scalability.

Even when the current model is a very complete representation of a typical SerDes validation TB there is still one additional element that is required to improve the flexibility of the code. Configuration objects are also supported by this infrastructure, so next, these elements are added at the test level as shown in Figure 119.

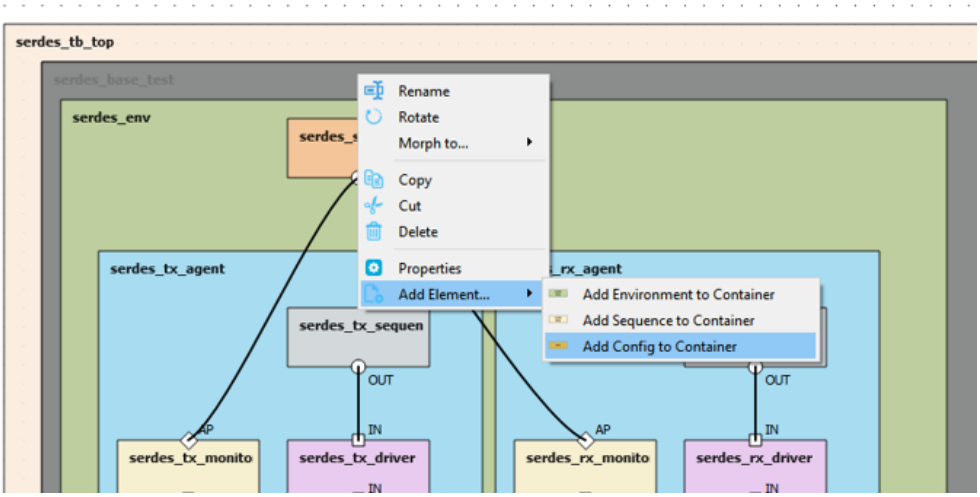


Figure 119 - SerDes adding cfg object

Two config objects shown in Figure 120 and Figure 121, are added to the scene, one for the TX operation and the second one for the RX operation.

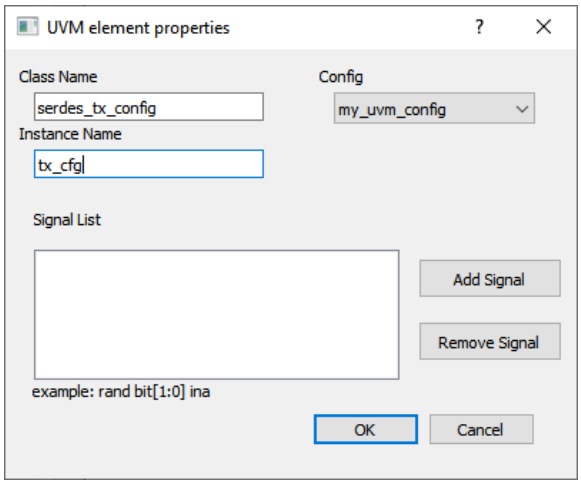


Figure 120 - SerDes TX config object

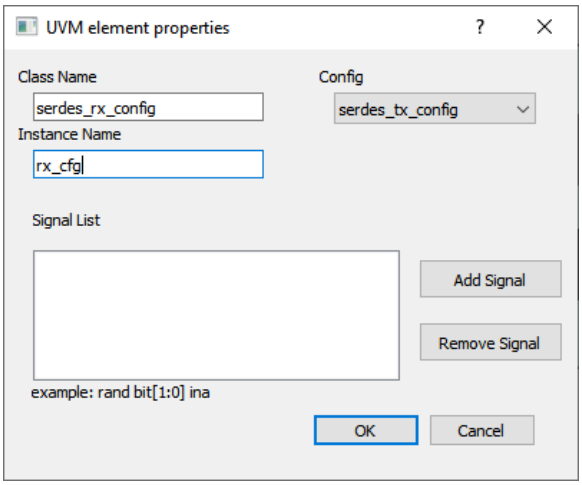


Figure 121 - SerDes RX config object

The diagram now shows two Config elements at the Test level. These config objects can be assigned to any element inside an Env container.

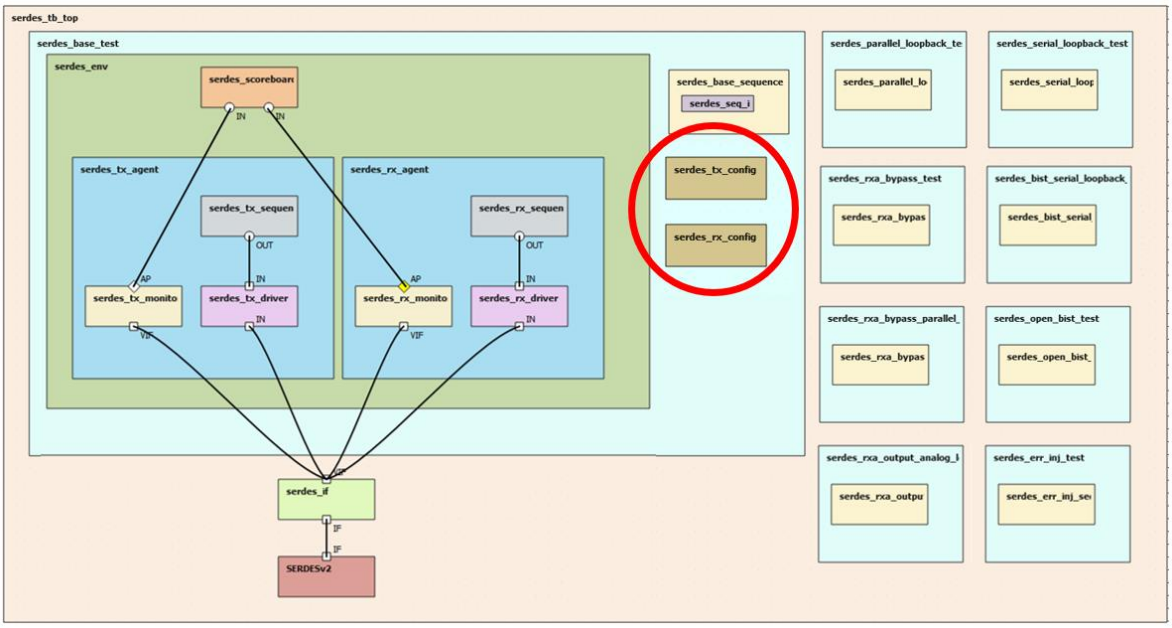


Figure 122 - SerDes TB with config objects

For the TX agent elements, the serdes_tx_config is selected in Figure 123. For the RX agent elements, the serdes_rx_config is selected in Figure 124.

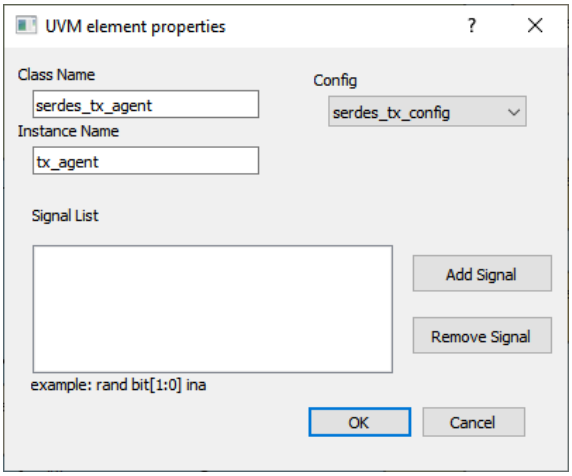


Figure 123 - SerDes TX agent with config

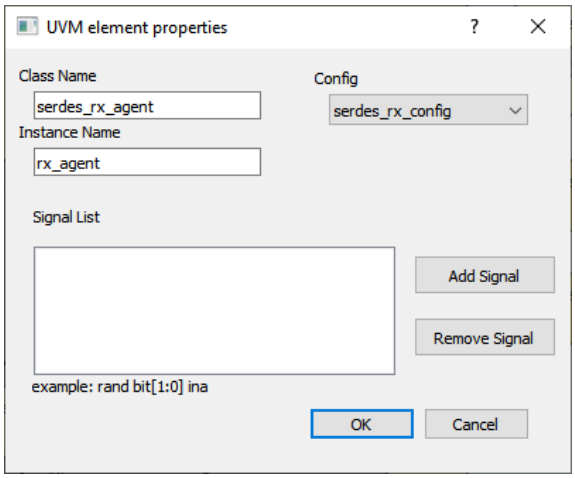


Figure 124 - SerDes RX agent with config

With this last step, the TB of the SerDes is complete. A final view of the model is shown in Figure 125 below.

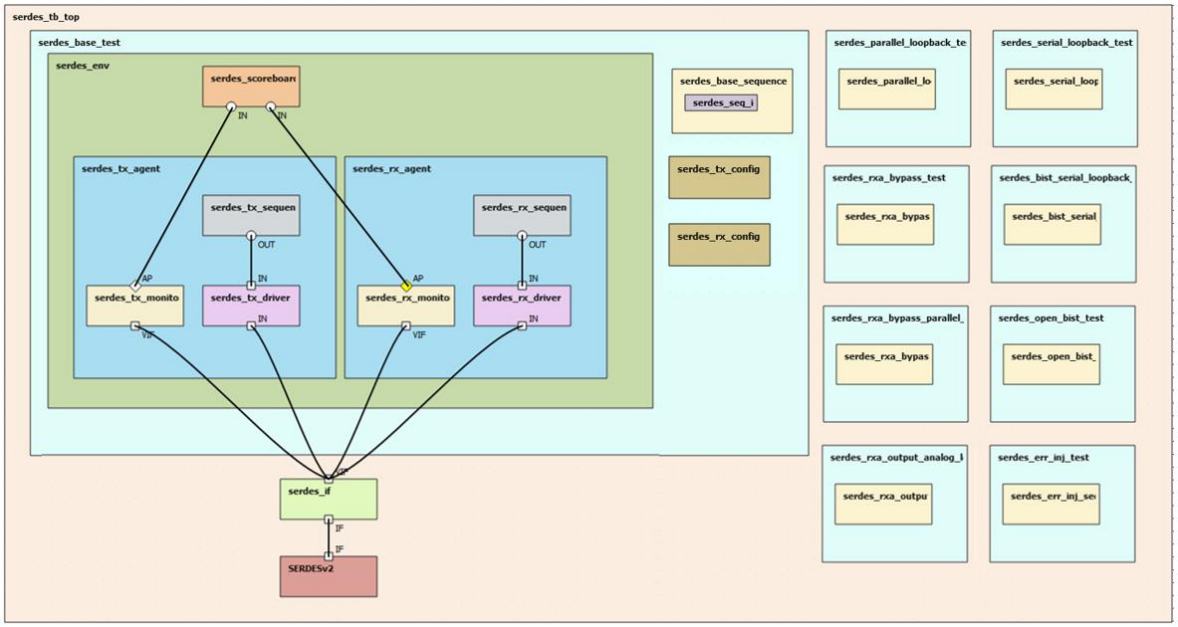


Figure 125 - SerDes complete testbench

Coverage collection for the RX and TX modules will be performed using a Subscriber element in a similar way as in the ALU example.

Two separate Subscriber elements will be added to each Agent container in Figure 126 and Figure 127. With this modular approach, the coverage collection has more flexibility and can be instantiated separately in other testbenches.

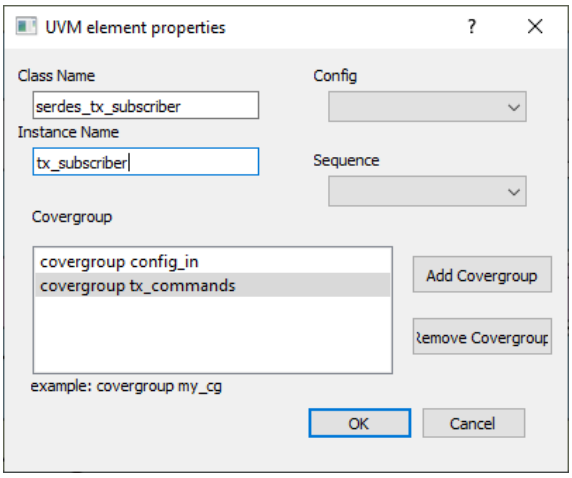


Figure 126 - SerDes TX subscriber properties menu

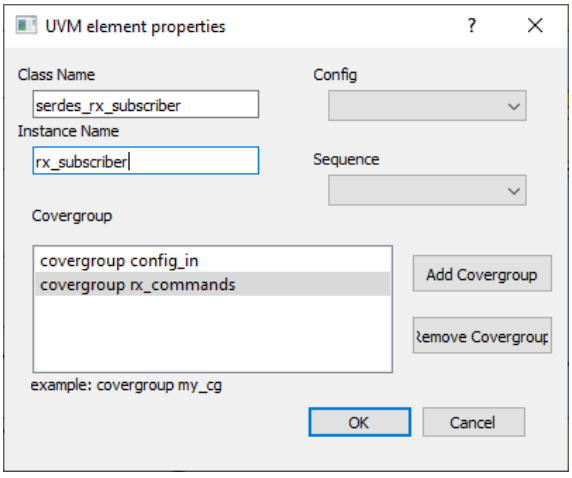


Figure 127 - SerDes RX subscriber properties menu

The new architecture enabled for coverage collection is shown in Figure 128.

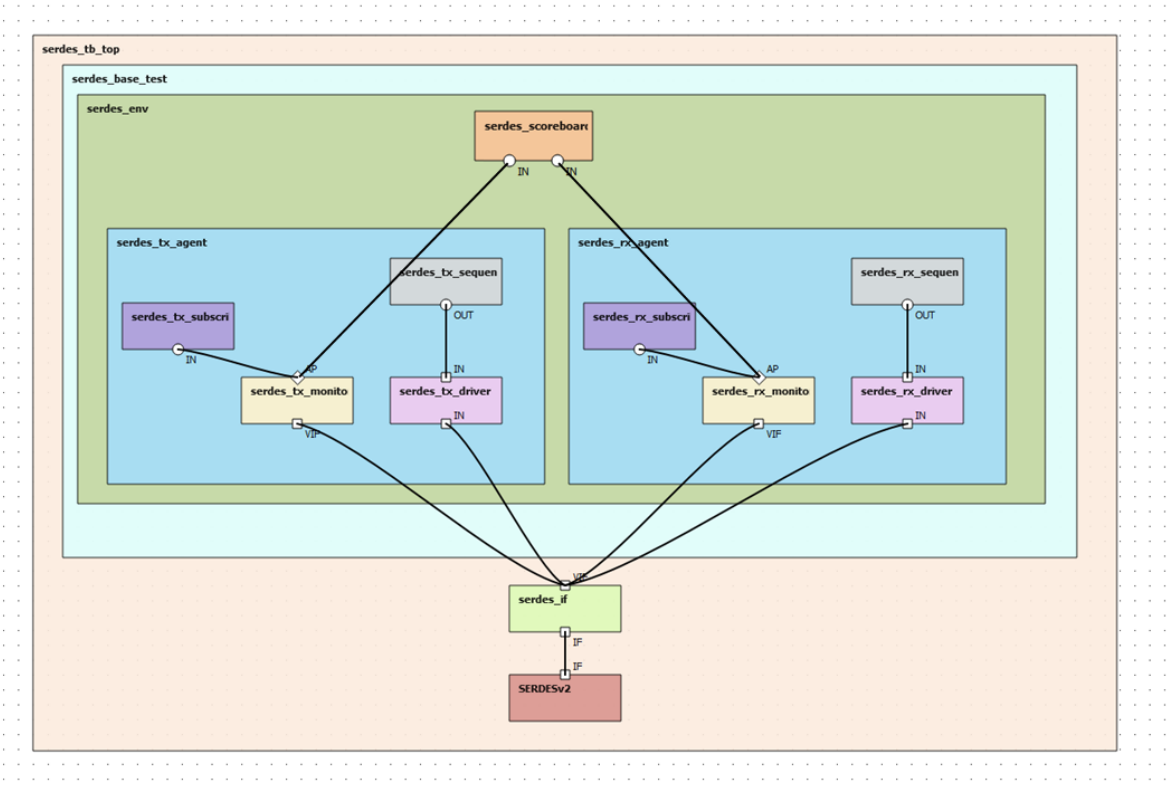


Figure 128 - SerDes TB with subscribers

4.2.3. Generated code output files

Once the TB model is ready, the code is generated by selecting the “Export to SystemVerilog” option from the task bar as shown in Figure 129.

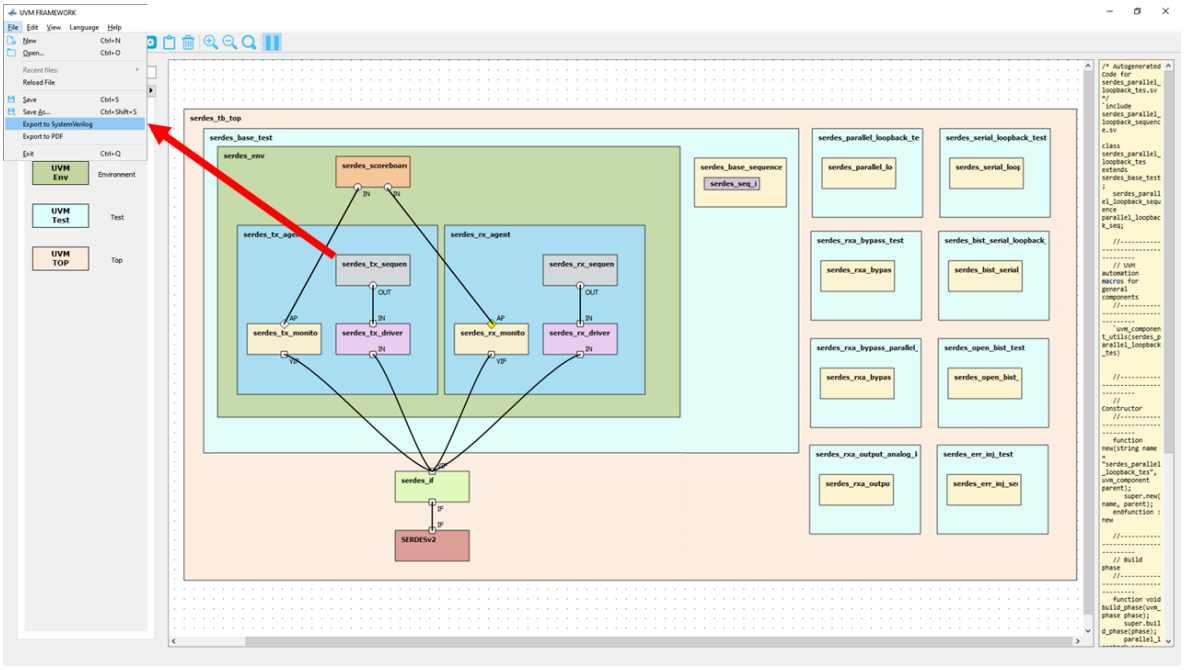


Figure 129 - SerDes generating SV code

The name of the project is also required as shown in Figure 130.

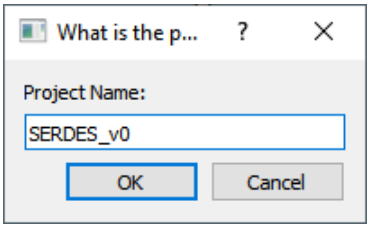


Figure 130 - SerDes project name

Finally, the files are created in the desired directory as shown in Figure 131.

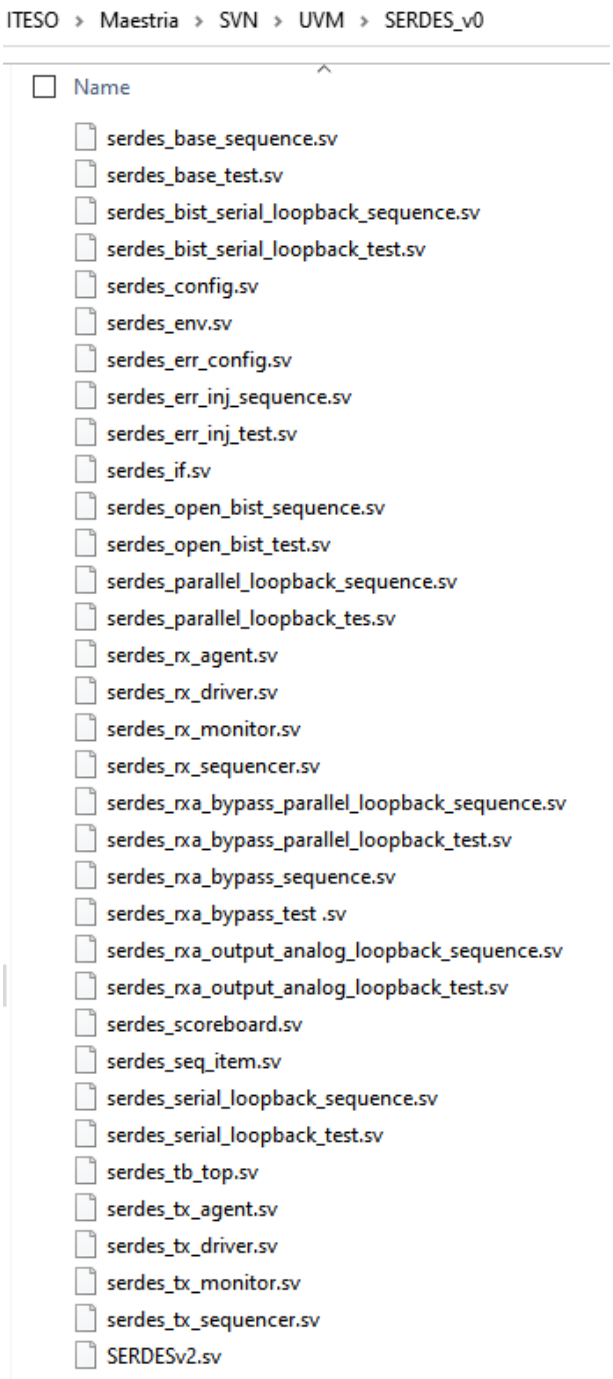


Figure 131 - SerDes output files

Chapter 5: Validation

This chapter covers the validation based on the output testbench code generated by the tool. It is divided into two parts, one for the ALU and one for the SerDes. The tool cannot generate the totality of the validation code, as each design is unique, however, the gaps that need to be filled are lesser as it will be shown throughout this chapter.

5.1. ALU

The RTL code of the ALU that is being validated is presented in Figure 132. It is intended to find as many errors as possible for complete validation. This ALU has two 4-bits input numbers “A” and “B” and a 3-bits opcode selector “sel”. The result of the selected operation is shown on the 5-bits output “result”.

```
01 module alu(
02   input logic clk,
03   input logic rst,
04   input logic[3:0] A, B,
05   input logic[2:0] sel,
06   output logic[4:0] result);
07
08   always@(posedge clk)
09     if(!rst)
10       begin
11         case(sel)
12           3'b000: result <= A + B ; //Addition
13           3'b001: result <= A + (~B + 1) ; //Subtraction
14           3'b010: result <= A & B ; //Logical AND
15           3'b011: result <= A | B ; //Logical OR
16           3'b100: result <= A * B ; //Multiplication
17           3'b101: result <= A ^ B ; //Logical XOR
18           3'b110: result <= A % B ; //Modulo operator
19           3'b111: result <= A / B ; //Division
20           default: $display("Invalid operation selected");
21         endcase
22       end
23      else
24       begin
25      result <= 5'bxxxxx;
26       end
27 endmodule
```

Figure 132 - ALU RTL Code

5.1.1. System Specifications

The ALU has the following system specifications:

ALU Specifications	
Operand length	4 bits
Output length	5 bits
Number of operands permitted	2
Required opcodes	Addition Subtraction Logical AND Logical OR Multiplication Logical XOR Modulo operator Division
Timing implementation	Synchronous
Minimum Coverage	80%

Table 29 - ALU System Specifications

5.1.2. Test Plan

The focus of the validation for the ALU is to provide input data and to verify that the resulting output data from the ALU is correct. To accomplish this validation, several approaches can be taken. The proposed verification architecture is based on a constrained random verification with UVM which consists of having the UVM testbench randomly generate input data A and B from the UVM Driver and the scoreboard will check the ALU output to verify correctness.

It is desired that all valid opcodes are tested and that corner cases are validated. The best way to guarantee this is by doing two things:

- Adding constraint randomness
- Adding a coverage block.

A minimum of 100% coverage fulfilled is required.

5.1.2.1. Required External Receivables:

Receivable	From Whom:
ALU Spec	Design
RTL DUT	Design
UVM testbench template	UVM Framework tool

5.1.2.2. Tests

There is only one test required for validating the ALU, it will be called alu_test.

Focused Test	Description
alu_test	The basic test for the ALU.

Table 30 - ALU Tests

5.1.2.3. Monitoring

The ALU will have a monitor to snoop through the virtual interface signals at every clock pulse.

5.1.2.4. Checks

To ensure the proper functioning of the ALU, all the opcodes should be checked. This will be done by a checker function in the scoreboard.

Check	Implementation Description	Type
Addition	A + B checker	Checker
Subtraction	A – B checker	Checker
Logical AND	A & B checker	Checker
Logical OR	A | B checker	Checker
Multiplication	A * B checker	Checker
Logical XOR	A ^ B checker	Checker
Modulo	A % B checker	Checker
Division	A / B checker	Checker

Table 31 - ALU Checks

5.1.2.5. Coverage

The following coverpoints were defined to guarantee that the ALU has complete testing. A 100% coverage is expected to be hit.

Cover point	Description	Type
SEL_cp	Covers all valid opcodes	Coverpoint
A_equal_B_cp	A equals B	Coverpoint
A_greater_B_cp	A > B	Coverpoint
B_greater_A_cp	B > A	Coverpoint
A_values_cp	Min and max values of A	Coverpoint
B_values_cp	Min and max values of B	Coverpoint
A_cross_SEL_cp	Min and max values of A for all valid opcodes.	Cross Coverpoint
B_cross_SEL_cp	Min and max values of B for all valid opcodes.	Cross Coverpoint

Table 32 - ALU Coverpoints

5.1.2.6. Stimulus

The stimulus is how the testing is driven. For the ALU, one drive task is enough to accomplish the validation goals.

Stimulus tasks	Description	Category
Drive	Randomized sequences sent.	Generic

Table 33 - ALU Stimulus Tasks

5.1.3. Testbench

The verification components used in this testbench will be described below. Figure 133 shows the relationship between the verification components. The ALU is the design-under-test (DUT) module.

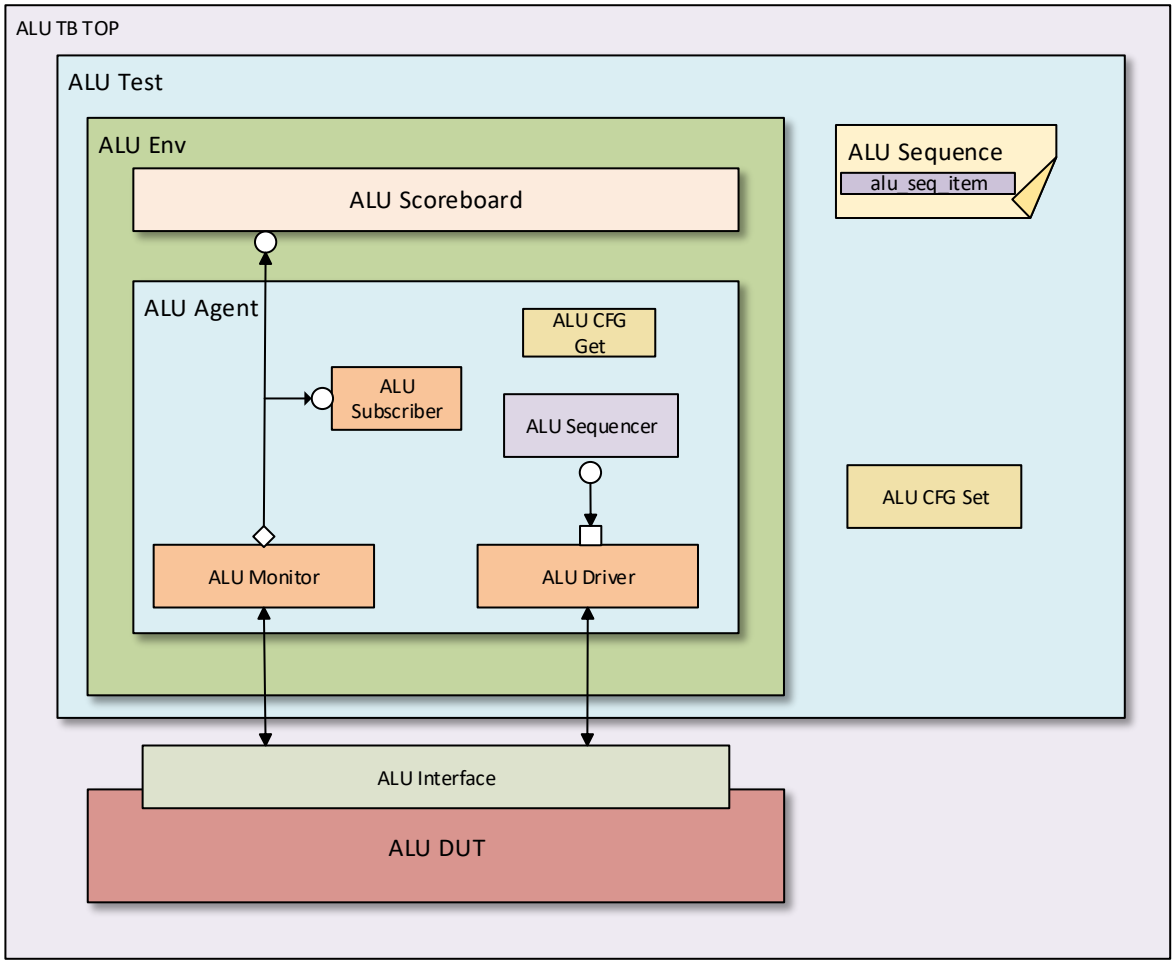


Figure 133 - ALU Testbench

5.1.3.1. ALU Interface

The interface for the ALU, shown in Figure 134, is very straight forward, it just has the same signals as the DUT.

```
01 /* Autogenerated Code for alu_if.sv */
02
03 interface alu_if(input logic clk, rst);
04    logic[3:0] A;
05    logic[3:0] B;
06    logic[2:0] sel;
07    logic[4:0] result;
08    // Optional Clocking block code:
09
10
11 endinterface
```

Figure 134 - ALU Interface Code

5.1.3.2. ALU Sequence Item

The alu_sequence_item consists of the data fields that are required for generating the stimulus of the ALU.

Inputs A and B (line 06 and 07) are the payload or data content of the transfer. These inputs are declared as “rand” and will later be controlled by defined constraints.
The input “sel” (line 08) is the configuration information of the transfer, it is being declared as “randc” as it is wanted to cycle through all the opcodes of the ALU. Variables declared with the “randc” keyword are random-cyclic variables that cycle through all the values in their declared range. The basic idea of defining “sel” as “randc” is such that no opcode is repeated within an iteration.

Lastly, the result (line 09) is the analysis information, this will be used for capturing the response of the ALU. This field shouldn´t be randomized and hence, it is not present on the utility and fields macros.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03
04
05 class alu_seq_item extends uvm_sequence_item;
06    rand logic[3:0] A;
07    rand logic[3:0] B;
08    randc logic[2:0] sel;
09    logic[4:0] result;
10
11    `uvm_object_utils_begin(alu_seq_item)
12    `uvm_field_int(A,UVM_ALL_ON)
13    `uvm_field_int(B,UVM_ALL_ON)
14    `uvm_field_int(sel,UVM_ALL_ON)
15    `uvm_object_utils_end
16
17
18    //-------------------------------------
19    // Constructor
20    //-------------------------------------
21  function new(string name= "alu_seq_item");
22     super.new(name);
23  endfunction
24
25    //-------------------------------------
26    // Optional constraints code
27    //-------------------------------------
28
29
30 endclass : alu_seq_item
```

Figure 135 - ALU Sequence Item code

5.1.3.3. ALU Sequencer

The alu_sequencer in Figure 180 will generate transactions as class objects alu_seq_item and send them to the alu_driver.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 //This file is the sequencer that will take the stimuli to the driver
04 import aluEnvPkg::*;
05 class alu_sequencer extends uvm_sequencer#(alu_seq_item);
06
07    //-------------------------------------
08    // UVM automation macros for general components
09    //-------------------------------------
10    `uvm_component_utils(alu_sequencer)
11
12    //---------------------------------------
13    //constructor
14    //---------------------------------------
15    function new(string name, uvm_component parent);
16      super.new(name,parent);
17    endfunction
18
19 endclass
```

Figure 136 - ALU Sequencer Code

5.1.3.4. ALU Driver

The alu_driver in Figure 137, will take the information from the alu_sequencer and create the transactions. These transactions are then passed to the ALU DUT through the alu_interface to compute the result.

The drive() task (line 48) will drive the inputs on the first clock and the next clock will drive the result computed by the ALU.

```
`include "uvm_macros.svh"
import uvm_pkg::*;
//This is the driver file
import aluEnvPkg::*;

class alu_driver extends uvm_driver#(alu_seq_item);
   //------------------------------------
   // Virtual Interface
   //------------------------------------
   virtual alu_if   vif;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(alu_driver)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db#(virtual alu_if)::get(this, "","vif",vif))
         `uvm_fatal("No_vif", {"Virtual interface must be set for: ",get_full_name(),".vif"});
   endfunction : build_phase

   //------------------------------------
   // Run phase
   //------------------------------------
   virtual task run_phase(uvm_phase phase);
      forever begin
         seq_item_port.get_next_item(req);
         drive();
         seq_item_port.item_done();
      end
   endtask : run_phase

   //------------------------------------
   // Drive task
   //------------------------------------
   virtual task drive();
    @(posedge vif.clk)
    vif.A <= req.A;
    vif.B <= req.B;
    vif.sel <= req.sel;
 `uvm_info("",$sformatf("Driver: A is %d B is %d Sel is %d", vif.A,vif.B,vif.sel), UVM_MEDIUM)
    repeat(2)
    @(posedge vif.clk)
    req.result <= vif.result;
   endtask : drive

endclass : alu_driver
```

Figure 137 - ALU Driver Code

5.1.3.5. ALU Monitor

Concurrently, as the transactions are being created by the alu_driver, the alu_monitor in Figure 138 will capture the operation and results. This information will be passed down to two components, the alu_subscriber which in this design is being used to capture functional coverage, and the alu_scoreboard, which will act as a checker for the design.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 //This is the monitor file
04 import aluEnvPkg::*;
05 class alu_monitor extends uvm_monitor;
06    //------------------------------------
07    // Virtual Interface
08    //------------------------------------
09    virtual alu_if   vif;
10
11    //------------------------------------
12    // UVM automation macros for general components
13    //------------------------------------
14    `uvm_component_utils(alu_monitor)
15
16    //------------------------------------
17    // Analysis port
18    //------------------------------------
19    uvm_analysis_port#(alu_seq_item)item_collected_port;
20
21    //------------------------------------
22    // Constructor
23    //------------------------------------
24    function new(string name, uvm_component parent);
25       super.new(name, parent);
26    endfunction : new
27
28    //------------------------------------
29    // Build phase
30    //------------------------------------
31    function void build_phase(uvm_phase phase);
32       super.build_phase(phase);
33       item_collected_port = new("item_collected_port", this);
34       if(!uvm_config_db#(virtual alu_if)::get(this, "","vif",vif))
35          `uvm_fatal("No_vif", {"Virtual interface must be set for: ",get_full_name(),".vif"});
36    endfunction : build_phase
37
38    //------------------------------------
39    // Run phase
40    //------------------------------------
41   virtual task run_phase(uvm_phase phase);
42       alu_seq_item seq_item_collected = alu_seq_item::type_id::create("alu_seq_item", this);
43       forever begin
44          // Fill with Monitor sampling process HERE
45       @(posedge vif.clk)
46       seq_item_collected.A = vif.A;
47       seq_item_collected.B = vif.B;
48       seq_item_collected.sel = vif.sel;
49       @(posedge vif.clk)
50       seq_item_collected.result = vif.result;
51       @(posedge vif.clk)
52       `uvm_info (get_type_name(),$sformatf("Monitor: A=4'b%b B=4'b%b
Result=5'b%b",seq_item_collected.A, seq_item_collected.B, seq_item_collected.result),UVM_MEDIUM)
53       item_collected_port.write(seq_item_collected);
54       end
55    endtask : run_phase
56
57 endclass : alu_monitor
58
```

Figure 138 - ALU Monitor Code

5.1.3.6. ALU Subscriber

This component in Figure 139 is connected or subscribed to the monitor's analysis port, in this case, it will be used as a functional coverage module that will record and total the transactions being computed by the ALU.

One covergroup (line 09 to line 34) is enough to make sure that all interesting scenarios are

being hit by the test content. It is desired, that all the operation opcodes are hit with key values (i.e. max. and min.), so cross coverpoints were defined on lines 32 and 33.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 import aluEnvPkg::*;
04 class alu_subscriber extends uvm_subscriber#( alu_seq_item );
05    `uvm_component_utils( alu_subscriber )
06
07    alu_seq_item seq;
08
09    covergroup alu_cg;
10       SEL_cp:      coverpoint seq.sel
11                    {
12                    bins Add  = {3'b000};
13                    bins Sub  = {3'b001};
14                    bins And  = {3'b010};
15                    bins Or   = {3'b011};
16                    bins Mult = {3'b100};
17                    bins Xor  = {3'b101};
18                    bins Mod  = {3'b110};
19                    bins Div  = {3'b111};
20                    }
21    A_equal_B_cp:   coverpoint seq.A == seq.B {bins hit = {1};}
22    A_greater_B_cp: coverpoint seq.A > seq.B {bins hit = {1};}
23    B_greater_A_cp: coverpoint seq.B > seq.A {bins hit = {1};}
24    A_values_cp:    coverpoint seq.A {
25                    bins min = {4'b0000};
26                bins max = {4'b1111};
27                 }
28    B_values_cp:    coverpoint seq.B    {
29                    bins min = {4'b0000};
30                bins max = {4'b1111};
31                 }
32    A_cross_SEL_cp: cross A_values_cp, SEL_cp;
33    B_cross_SEL_cp: cross B_values_cp, SEL_cp;
34    endgroup: alu_cg
35
36    function new( string name, uvm_component parent );
37       super.new( name, parent );
38       alu_cg = new;
39    endfunction: new
40
41
42    function void write( alu_seq_item t );
43       seq = t;
44     $display("my_cov_item obtained by my_coverage");
45       alu_cg.sample();
46    endfunction: write
47
48 endclass: alu_subscriber
```

Figure 139 - ALU Subscriber Code

5.1.3.7. ALU Scoreboard

The other component that is subscribed to the alu_monitor via the monitor's analysis port is the alu_scoreboard.

The alu_scoreboard is a component that checks that the ALU is responding correctly.
The scoreboard is the component that requires more lines of code to be written by the verification engineer. As the checks are very specific to each design, it is impossible to auto-generate with a tool.

For the ALU, there is a checker for each operation of the ALU (line 038 to line 117) making the scoreboard´s code to be too long that it was divided into two figures, Figure 140 and Figure 141.

```
001 `include "uvm_macros.svh"
002 import uvm_pkg::*;
003 import aluEnvPkg::*;
004 class alu_scoreboard extends uvm_scoreboard;
005
006    //-------------------------------------
007    // UVM automation macros for general components
008    //-------------------------------------
009    `uvm_component_utils(alu_scoreboard)
010
011    //-------------------------------------
012    // Declaring port to receive packets
013    //-------------------------------------
014    uvm_tlm_analysis_fifo#(alu_seq_item)item_col;
015    alu_seq_item seq;
016
017    //-------------------------------------
018    // Constructor
019    //-------------------------------------
020    function new(string name, uvm_component parent);
021       super.new(name, parent);
022    endfunction : new
023
024    //-------------------------------------
025    // Build phase
026    //-------------------------------------
027    function void build_phase(uvm_phase phase);
028       super.build_phase(phase);
029       item_col = new("item_col",this);
030    endfunction : build_phase
031
032    //-------------------------------------
033    // Run phase
034    //-------------------------------------
035    virtual task run_phase(uvm_phase phase);
036       forever begin
037          item_col.get(seq);
038          if(seq.sel == 3'b000) begin
039             `uvm_info(get_type_name(),$sformatf(" The selected operation is Addition"), UVM_LOW)
040            `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
041             if(seq.A + seq.B === seq.result) begin
042                `uvm_info(get_type_name(),$sformatf("Addition Pass "),UVM_LOW)
043             end else begin
044               `uvm_error(get_type_name(),$sformatf("Addition Failed "))
045             end
046             $display("--------------------------------------------------------------------------------");
047          end else if(seq.sel == 3'b001) begin
048             `uvm_info(get_type_name(),$sformatf(" The selected operation is Subtraction"),UVM_LOW)
049            `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
050             if(seq.A + (~seq.B + 1) === seq.result) begin
051               `uvm_info(get_type_name(),$sformatf("Substraction Pass "),UVM_LOW)
052             end else begin
053               `uvm_error(get_type_name(),$sformatf("Substraction Failed "))
054             end
055             $display("--------------------------------------------------------------------------------");
056          end else if(seq.sel == 3'b010) begin
057            `uvm_info(get_type_name(),$sformatf(" The selected operation is Logical AND"),UVM_LOW)
058            `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
059           if(seq.A & seq.B === seq.result) begin
060                `uvm_info(get_type_name(),$sformatf("Logical AND Pass "),UVM_LOW)
061             end else begin
062               `uvm error(get type name(),$sformatf("Logical AND Failed "))
063             end
064            $display("--------------------------------------------------------------------------------");
065          end else if(seq.sel == 3'b011) begin
```

Figure 140 - ALU Scoreboard Code (Pt. 1)

```
066             `uvm_info(get_type_name(),$sformatf(" The selected operation is Logical OR"),UVM_LOW)
067             `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
068             if(seq.A | seq.B === seq.result) begin
069                `uvm_info(get_type_name(),$sformatf("Logical OR Pass "),UVM_LOW)
070             end else begin
071                `uvm_error(get_type_name(),$sformatf("Logical OR Failed "))
072             end
073             $display("--------------------------------------------------------------------------------");
074          end else if(seq.sel == 3'b100) begin
075             `uvm_info(get_type_name(),$sformatf(" The selected operation is Multiplication"),UVM_LOW)
076             `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
077             if(seq.A * seq.B !== seq.result) begin
078                 `uvm_error(get_type_name(),$sformatf("Multiplication: Test Failed "))
079             end if (seq.A != 0 && seq.result / seq.A != seq.B) begin
080                 `uvm_error(get_type_name(),$sformatf("Multiplication Overflow: Test Failed "))
081             end else begin
082              `uvm_info(get_type_name(),$sformatf("Multiplication Pass "),UVM_LOW)
083             end
084             $display("--------------------------------------------------------------------------------");
085          end else if(seq.sel == 3'b101) begin
086             `uvm_info(get_type_name(),$sformatf(" The selected operation is Logical XOR"),UVM_LOW)
087             `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
088             if(seq.A ^ seq.B == seq.result) begin
089                `uvm_info(get_type_name(),$sformatf("Logical XOR Pass "),UVM_LOW)
090             end else begin
091                `uvm_error(get_type_name(),$sformatf("Logical XOR Failed "))
092             $display("--------------------------------------------------------------------------------");
093             end
094          end else if(seq.sel == 3'b110) begin
095             `uvm_info(get_type_name(),$sformatf(" The selected operation is Modulo Operator"),UVM_LOW)
096             `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
097             if(seq.A % seq.B == seq.result) begin
098                `uvm_info(get_type_name(),$sformatf("Modulo OP Pass "),UVM_LOW)
099              end else begin
100                `uvm_error(get_type_name(),$sformatf("Modulo OP Failed "))
101              end
102             $display("--------------------------------------------------------------------------------");
103          end else if(seq.sel == 3'b111) begin
104             `uvm_info(get_type_name(),$sformatf(" The selected operation is Division"),UVM_LOW)
105             `uvm_info(get_type_name(),$sformatf(" Value of A = %0d Value of B = %0d Value of Result =
%0d", seq.A, seq.B, seq.result),UVM_LOW)
106             if (seq.B == 0) begin
107            `uvm_error(get_type_name(),$sformatf("Division by zero "))
108        end else
109        if(seq.A / seq.B !== seq.result) begin
110                `uvm_error(get_type_name(),$sformatf("Division Failed "))
111             end else begin
112                `uvm info(get type name(),$sformatf("Division Pass "),UVM LOW)
113             end
114           $display("--------------------------------------------------------------------------------" );
115           end else begin
116        `uvm info(get type name(),$sformatf ("Default value initiated"),UVM LOW)
117           end
118       end
119
120    endtask : run phase
121
122 endclass : alu_scoreboard
```

Figure 141 - ALU Scoreboard Code (Pt. 2)

5.1.3.8. ALU Config Object

To enable or disable the coverage at will, a configuration object in Figure 142, was added to the alu_agent. This will adjust the structure of the alu_agent by setting a bit called “has_subscriber” (line 08).

This bit will control if the coverage module (alu_subscriber) is connected to the monitor or not. Usually, all configuration objects are specified (set) within the UVM test while there is flexibility as from where the information can be fetched using get().

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 import aluEnvPkg::*;
04 class alu_config extends uvm_object;
05 `uvm_object_utils( alu_config )
06   // Do not register config class with the factory
07
08   bit                      has_subscriber = 0;
09
10    //------------------------------------
11    // Constructor
12    //------------------------------------
13   function new(string name= "alu_config");
14      super.new(name);
15   endfunction
16
17
18 endclass : alu_config
```

Figure 142 - ALU Config Code

5.1.3.9. ALU Agent

The ALU Agent in Figure 143, encapsulates the alu_sequencer, alu_driver, alu_monitor, and alu_subscriber. The information contained in the configuration object is obtained via the get() function in line 31.

With this, it makes use of the variable “has_subscriber” to determine if the subscriber is created and subscribed to the alu_monitor or not. By default, all agents are active which will remain to be true for the ALU testbench.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 // This is agent file which will combine together the sequencer, driver and monitor
04 import aluEnvPkg::*;
05 class alu_agent extends uvm_agent;
06    alu_monitor   monitor;
07    alu_sequencer   sequencer;
08    alu_driver   driver;
09    alu_config cfg;
10    alu_subscriber subscriber;
11
12    //-------------------------------------
13    // UVM automation macros for general components
14    //-------------------------------------
15    `uvm_component_utils(alu_agent)
16
17
18    //-------------------------------------
19    // Constructor
20    //-------------------------------------
21    function new(string name, uvm_component parent);
22       super.new(name, parent);
23
24    endfunction : new
25
26    //-------------------------------------
27    // Build phase
28    //-------------------------------------
29    function void build_phase(uvm_phase phase);
30       super.build_phase(phase);
31       if(!uvm_config_db#(alu_config)::get(this, "", "alu_config",cfg))
32         `uvm_fatal("No_cfg", {"CFG object not found: ",get_full_name(),"cfg"});
33       monitor = alu_monitor::type_id::create("monitor", this);
34       if(get_is_active == UVM_ACTIVE)
35          sequencer = alu_sequencer::type_id::create("sequencer", this);
36       if(get_is_active == UVM_ACTIVE)
37          driver = alu_driver::type_id::create("driver", this);
38       if (cfg.has_subscriber)
39  subscriber = alu_subscriber::type_id::create("subscriber", this);
40
41    endfunction : build_phase
42
43    //-------------------------------------
44    // Connect phase
45    //-------------------------------------
46    function void connect_phase(uvm_phase phase);
47       if(get_is_active == UVM_ACTIVE)
48          driver.seq_item_port.connect( sequencer.seq_item_export);
49       if (cfg.has_subscriber) begin
50   `uvm_info (get_type_name(),$sformatf("Monitor: Coverage enabled"),UVM_MEDIUM)
51          monitor.item_collected_port.connect( subscriber.analysis_export );
52       end else
53          `uvm_info (get_type_name(),$sformatf("Monitor: Coverage disabled"),UVM_MEDIUM)
54    endfunction : connect_phase
55
56 endclass : alu_agent
```

Figure 143 - ALU Agent Code

5.1.3.10. ALU Environment

The ALU Environment in Figure 144 is a container that has alu_agent and the alu_scoreboard. The scoreboard will subscribe to the ALU monitor´s analysis port to obtain the transactions´ information (line 35).

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 import aluEnvPkg::*;
04
05 class alu_env extends uvm_env;
06    alu_scoreboard   scoreboard;
07    alu_agent   agent;
08
09    //-------------------------------------
10    // UVM automation macros for general components
11    //-------------------------------------
12    `uvm_component_utils(alu_env)
13
14
15    //-------------------------------------
16    // Constructor
17    //-------------------------------------
18    function new(string name, uvm_component parent);
19       super.new(name, parent);
20    endfunction : new
21
22    //-------------------------------------
23    // Build phase
24    //-------------------------------------
25    function void build_phase(uvm_phase phase);
26       super.build_phase(phase);
27       scoreboard = alu_scoreboard::type_id::create("scoreboard", this);
28       agent = alu_agent::type_id::create("agent", this);
29    endfunction : build_phase
30
31    //-------------------------------------
32    // Connect phase
33    //-------------------------------------
34    function void connect_phase(uvm_phase phase);
35     agent.monitor.item_collected_port.connect(scoreboard.item_col.analysis_export);
36    endfunction : connect_phase
37
38 endclass : alu_env
```

Figure 144 - ALU Env Code

5.1.3.11. ALU Sequence

The ALU sequence in Figure 145 used for this verification will randomize 100 transactions within the constraints established in the sequence item. With this many transactions, the validation goals are achieved.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 //This is file which will generate the sequence which will take the sequence items on the seqeuncer in
a particular sequence
04 import aluEnvPkg::*;
05 class alu_sequence extends uvm_sequence #(alu_seq_item);
06
07    //------------------------------------
08    // UVM automation macros for general components
09    //------------------------------------
10   `uvm_object_utils(alu_sequence)
11
12
13    //------------------------------------
14    // Constructor
15    //------------------------------------
16    function new(string name = "alu_sequence");
17       super.new(name);
18    endfunction : new
19
20    //------------------------------------
21    // Body task
22    //------------------------------------
23    virtual task body();
24     repeat(100) begin
25          req = alu_seq_item::type_id::create("req");
26          start_item(req);
27          assert (req.randomize());
28          finish_item(req);
29     end
30    endtask : body
31
32
33 endclass : alu_sequence
```

Figure 145 - ALU Sequence Code

5.1.3.12. ALU Test

The main function of a UVM Test is to call the sequences to be run. For the ALU Test in Figure 146, there is only one sequence that is called on the run_phase task (line 46 to line 50). The topology is also printed on line 40, as this is extra information that is nice to have in the test log. In the ALU test, the variable “has_subcriber” is being set in the configuration object. This will enable coverage for this particular test.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 import aluEnvPkg::*;
04
05 class alu_test extends uvm_test;
06    alu_sequence   seq;
07    alu_env   env;
08    alu_config cfg;
09
10    //-------------------------------------
11    // UVM automation macros for general components
12    //-------------------------------------
13    `uvm_component_utils(alu_test)
14
15
16    //-------------------------------------
17    // Constructor
18    //-------------------------------------
19    function new(string name, uvm_component parent);
20       super.new(name, parent);
21    endfunction : new
22
23    //-------------------------------------
24    // Build phase
25    //-------------------------------------
26    function void build_phase(uvm_phase phase);
27       super.build_phase(phase);
28       cfg = alu_config::type_id::create( "cfg" );
29       cfg.has_subscriber =1;
30       uvm_config_db #(alu_config)::set(this,"env.agent", "alu_config", cfg);
31       seq = alu_sequence::type_id::create("seq", this);
32       env = alu_env::type_id::create("env", this);
33    endfunction : build_phase
34
35   //-----------------------------------------
36   // end_of_elaboration phase
37   //-----------------------------------------
38   virtual function void end_of_elaboration();
39     //print's the topology
40     print();
41   endfunction
42
43    //-------------------------------------
44    // Run phase
45    //-------------------------------------
46    task run_phase(uvm_phase phase);
47       phase.raise_objection(this);
48       seq.start( env.agent.sequencer );
49       phase.drop_objection(this);
50    endtask : run_phase
51
52 endclass : alu_test
```

Figure 146 - ALU Test Code

5.1.3.13. ALU TB TOP

The ALU TB Top in Figure 147 is where the ALU RTL gets instantiated and connected to the virtual interface. It also contains the clock and reset generation. In the Top Testbench, also the virtual interface is set from the configuration database.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03 `include "tb/aluEnvPkg.sv"
04 import aluEnvPkg::*;
05
06 module alu_tb_top;
07
08    //-------------------------------------
09    //clock and reset signal declaration
10    //-------------------------------------
11    bit clk;
12    bit rst;
13
14    alu   dut(
15       .clk ( vif.clk ),
16       .rst ( vif.rst ),
17       .A ( vif.A ),
18       .B ( vif.B ),
19       .sel ( vif.sel ),
20       .result ( vif.result )
21    );
22    alu_if vif ( .clk(clk), .rst(rst) );
23    //-------------------------------------
24    //clock generation
25    //-------------------------------------
26    always #5 clk = ~clk;
27
28    //-------------------------------------
29    //reset generation
30    //-------------------------------------
31    initial begin
32       rst = 1;
33       #5 rst = 0;
34    end
35
36    // Procedural block
37    initial begin
38       uvm_config_db#(virtual alu_if )::set(uvm_root::get(),"*","vif",vif);
39       // Enable wave dump
40       $dumpfile("dump.vcd");
41       $dumpvars;
42        uvm_top.finish_on_completion = 1;
43
44       // Calling test
45       run_test();
46    end
47
48
49 endmodule
```

Figure 147 - ALU TB TOP Code

5.1.3.14. ALU Environment Package

A UVM Package is a complete collection of all Classes that are needed to build a verification Environment. These components, previously described in this chapter, required for the ALU Testbench are listed included in the aluEnvPkg in Figure 148.

```
01 `include "uvm_macros.svh"
02 import uvm_pkg::*;
03
04 package aluEnvPkg;
05 `include "tb/alu_config.sv"
06 `include "tb/alu_seq_item.sv"
07 `include "tb/alu_subscriber.sv"
08 `include "tb/alu_sequence.sv"
09 `include "tb/alu_sequencer.sv"
10 `include "tb/alu_driver.sv"
11 `include "tb/alu_monitor.sv"
12 `include "tb/alu_scoreboard.sv"
13 `include "tb/alu_agent.sv"
14 `include "tb/alu_env.sv"
15 `include "tb/alu_test.sv"
16
17
18
19
20 endpackage: aluEnvPkg
```

Figure 148 - ALU Env Pkg Code

5.1.4. Simulation

Many commercial digital simulators can be used to simulate SystemVerilog circuits with UVM. For the ALU, all simulations were done using Questa Sim from Mentor graphics.

The first step is making sure that the testbench is compiled clean as shown in Figure 149.

```
# Compile of alu.sv was successful.
# Compile of alu_if.sv was successful.
# Compile of alu_seq_item.sv was successful with warnings.
# Compile of alu_tb_top.sv was successful.
# Compile of aluEnvPkg.sv was successful.
# Compile of alu_sequence.sv was successful with warnings.
# Compile of alu_sequencer.sv was successful.
# Compile of alu_scoreboard.sv was successful.
# Compile of alu_driver.sv was successful.
# Compile of alu_monitor.sv was successful.
# Compile of alu_agent.sv was successful.
# Compile of alu_test.sv was successful.
# Compile of alu_env.sv was successful.
# Compile of alu_config.sv was successful.
# Compile of alu_subscriber.sv was successful.
# 15 compiles, 0 failed with no errors.

QuestaSim>
```

Figure 149 - ALU Compile

Then, the simulation can be run. The command used to start the simulation for the ALU is:

vsim work.alu_tb_top +UVM_TESTNAME=alu_test

The topology of the ALU is printed during the simulation in the *end_of_elaboration* phase and is shown in Figure 150.

```
# ----------------------------------------------------------------------
# Name                         Type                        Size  Value
# ----------------------------------------------------------------------
# uvm_test_top                 alu_test                    -     @472
#   env                        alu_env                     -     @488
#     agent                    alu_agent                   -     @502
#       driver                 alu_driver                  -     @627
#         rsp_port             uvm_analysis_port           -     @642
#         seq_item_port        uvm_seq_item_pull_port      -     @634
#       monitor                alu_monitor                 -     @511
#         item_collected_port  uvm_analysis_port           -     @666
#       sequencer              alu_sequencer               -     @518
#         rsp_export           uvm_analysis_export         -     @525
#         seq_item_export      uvm_seq_item_pull_imp       -     @619
#         arbitration_queue    array                       0     -
#         lock_queue           array                       0     -
#         num_last_reqs        integral                    32    'd1
#         num_last_rsps        integral                    32    'd1
#       subscriber             alu_subscriber              -     @650
#         analysis_imp         uvm_analysis_imp            -     @657
#     scoreboard               alu_scoreboard              -     @495
#       item_col               uvm_tlm_analysis_fifo #(T)  -     @680
#         analysis_export      uvm_analysis_imp            -     @719
#         get_ap               uvm_analysis_port           -     @711
#         get_peek_export      uvm_get_peek_imp            -     @695
#         put_ap               uvm_analysis_port           -     @703
#         put_export           uvm_put_imp                 -     @687
# ----------------------------------------------------------------------
```

Figure 150 - ALU's topology

The behavior of the testbench is working as expected as it can be shown in the following waveform in Figure 151:

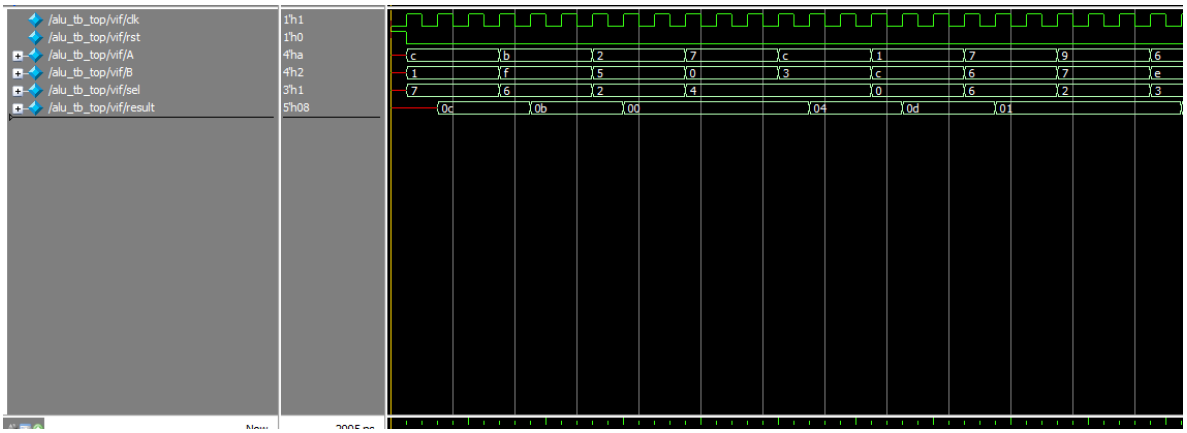


Figure 151 - ALU Waveforms

5.1.5. Coverage Report

The first run without constraints of 100 transactions showed the following coverage results:

A_equal_B_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
A_greater_B_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
A_values_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
B_greater_A_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
B_values_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
SEL_cp	8	8	0	**100.00%**	**100.00%**	**100.00%**
CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage%

Table 34 - ALU First Coverpoint Results

A_cross_SEL_cp	16	10	6	62.50%	**62.50%**	**62.50%**
B_cross_SEL_cp	16	9	7	56.25%	**56.25%**	**56.25%**
Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage%

Table 35 - ALU First Cross Coverpoint Results

Summary	Total Bins	Hits	Hit %
Coverpoints	15	15	100.00%
Crosses	32	19	59.37%

Table 36 - ALU First Coverage Summary

These results are very promising but not all the corner cases were hit. There are some misses on opcodes that weren´t tested with the maximum and minimum values of A and B. To increase coverage two things can be done:

- Increase the number of transactions tested. For the ALU it is possible to do without problems but for more complex designs it could be hard to have the compute resources available for big tests.
- Improve the constraints so that corner cases are often hit.

It was opted to improve the constraints by adding the following code in Figure 152. This will allow that 10% of the A and B goes zero and 10% goes to the maximum value of ´d15.

```
constraint A_max_min {A dist {0:=10, 15:=10};}
constraint B_max_min {B dist {0:=10, 15:=10};}
```

Figure 152 - Max and Min constraints

After these tweaks, the improvement is noticeable as 100% coverage was obtained.

A_equal_B_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
A_greater_B_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
A_values_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
B_greater_A_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
B_values_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
SEL_cp	8	8	0	**100.00%**	**100.00%**	**100.00%**
CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage%

Table 37 - ALU First Coverpoints Results

A_cross_SEL_cp	16	16	0	**100.00%**	**100.00%**	**100.00%**
B_cross_SEL_cp	16	16	0	**100.00%**	**100.00%**	**100.00%**
Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage%

Table 38 - ALU Final Cross Coverpoint Results

Summary	Total Bins	Hits	Hit %
Coverpoints	15	15	**100.00%**
Crosses	32	32	**100.00%**

Table 39 - ALU Final Coverage Summary

5.1.6. Bug Report

Even though the coverage looks promising the pass rate is not. The scoreboard showed 22 errors found which translates to a 78% pass rate. These errors can be divided into the following five buckets:

- **Modulo OP Error**

This error in Figure 153 is caused when the Modulo OP is selected and the input B is zero. This causes a division by zero, which causes to have an X in the result.

```
# ----------------------------------------------------------------------------------
# UVM_INFO tb/alu_driver.sv(53) @ 65: uvm_test_top.env.agent.driver [] Driver: A is  0 B is  0 Sel is 6
# UVM_INFO tb/alu_monitor.sv(52) @ 85: uvm_test_top.env.agent.monitor [alu_monitor] Monitor: A=4'b0000
B=4'b0000 Result=5'bxxxxx
# my_cov_item obtained by my_coverage
# UVM_INFO tb/alu_scoreboard.sv(95) @ 85: uvm_test_top.env.scoreboard [alu_scoreboard]  The selected
operation is Modulo Operator
# UVM_INFO tb/alu_scoreboard.sv(96) @ 85: uvm_test_top.env.scoreboard [alu_scoreboard]  Value of A = 0
Value of B = 0 Value of Result = x
# UVM_ERROR tb/alu_scoreboard.sv(100) @ 85: uvm_test_top.env.scoreboard [alu_scoreboard] Modulo OP Failed
# ----------------------------------------------------------------------------------
```

Figure 153 - Modulo OP Error

- **Logical AND Error**

The error shown in Figure 154 is a validation bug in the scoreboard, the problem is that the A and B have 4 bits and the result has 5 bits.

```
# ----------------------------------------------------------------------------------------
# UVM_INFO tb/alu_driver.sv(53) @ 95: uvm_test_top.env.agent.driver [] Driver: A is  0 B is 15 Sel is 2
run
# UVM_INFO tb/alu_monitor.sv(52) @ 115: uvm_test_top.env.agent.monitor [alu_monitor] Monitor: A=4'b0000
B=4'b1111 Result=5'b00000
# my_cov_item obtained by my_coverage
# UVM_INFO tb/alu_scoreboard.sv(57) @ 115: uvm_test_top.env.scoreboard [alu_scoreboard]  The selected
operation is Logical AND
# UVM_INFO tb/alu_scoreboard.sv(58) @ 115: uvm_test_top.env.scoreboard [alu_scoreboard]  Value of A = 0
Value of B = 15 Value of Result = 0
# UVM_ERROR tb/alu_scoreboard.sv(62) @ 115: uvm_test_top.env.scoreboard [alu_scoreboard] Logical AND
Failed
# ----------------------------------------------------------------------------------------
```

Figure 154 - Logical AND Error

The fix is to update the scoreboard to only take into account the first four bits of “result” and ignore the fifth.

```
if(seq.A & seq.B === seq.result[3:0]) begin
```

- **Multiplication Overflow Error**

The error shown in Figure 155 is a flaw in the design. The result is only 5 bits long isn´t enough to capture all the bits of multiplication with two 4 bits input. If the design is limited by and cannot increase the number of bits in the output, the validator could limit the inputs with constraints.

```
# ----------------------------------------------------------------------------------------
# UVM_INFO tb/alu_driver.sv(53) @ 485: uvm_test_top.env.agent.driver [] Driver: A is 15 B is 15 Sel is 4
# UVM_INFO tb/alu_monitor.sv(52) @ 505: uvm_test_top.env.agent.monitor [alu_monitor] Monitor: A=4'b1111
B=4'b1111 Result=5'b00001
# my_cov_item obtained by my_coverage
# UVM_INFO tb/alu_scoreboard.sv(75) @ 505: uvm_test_top.env.scoreboard [alu_scoreboard]  The selected
operation is Multiplication
# UVM_INFO tb/alu_scoreboard.sv(76) @ 505: uvm_test_top.env.scoreboard [alu_scoreboard]  Value of A = 15
Value of B = 15 Value of Result = 1
# UVM_ERROR tb/alu_scoreboard.sv(80) @ 505: uvm_test_top.env.scoreboard [alu_scoreboard] Multiplication
Overflow: Test Failed
# ----------------------------------------------------------------------------------------
```

Figure 155 - Multiplication Overflow Error

- **Division by zero**

The error displayed in Figure 156 is caused when a division by zero is occurring. The RTL can be updated to handle this scenario or if this is expected the validator can add a constraint to do waive this scenario and a exclude can be added to de coverpoint expecting this to be hit.

```
# ----------------------------------------------------------------------------------
# UVM_INFO tb/alu_driver.sv(53) @ 605: uvm_test_top.env.agent.driver [] Driver: A is 15 B is  0 Sel is 7
# UVM_INFO tb/alu_monitor.sv(52) @ 625: uvm_test_top.env.agent.monitor [alu_monitor] Monitor: A=4'b1111
B=4'b0000 Result=5'bxxxxx
# my_cov_item obtained by my_coverage
# UVM_INFO tb/alu_scoreboard.sv(104) @ 625: uvm_test_top.env.scoreboard [alu_scoreboard]  The selected
operation is Division
# UVM_INFO tb/alu_scoreboard.sv(105) @ 625: uvm_test_top.env.scoreboard [alu_scoreboard]  Value of A = 15
Value of B = 0 Value of Result = x
# UVM_ERROR tb/alu_scoreboard.sv(107) @ 625: uvm_test_top.env.scoreboard [alu_scoreboard] Division by
zero
# ----------------------------------------------------------------------------------
```

Figure 156 - Division by Zero Error

- **Subtraction Error**

Figure 157 is showcasing an error that happens when the ALU is subtracting the case of A is lesser than B. This is also a validation bug in which the checker needs to be updated to handle this corner case or a constraint to only allow A > B can be added.

```
# ----------------------------------------------------------------------------------
# UVM_INFO tb/alu_driver.sv(53) @ 2375: uvm_test_top.env.agent.driver [] Driver: A is  0 B is 15 Sel is 1
# UVM_INFO tb/alu_monitor.sv(52) @ 2395: uvm_test_top.env.agent.monitor [alu_monitor] Monitor: A=4'b0000
B=4'b1111 Result=5'b10001
# my_cov_item obtained by my_coverage
# UVM_INFO tb/alu_scoreboard.sv(48) @ 2395: uvm_test_top.env.scoreboard [alu_scoreboard]  The selected
operation is Subtraction
# UVM_INFO tb/alu_scoreboard.sv(49) @ 2395: uvm_test_top.env.scoreboard [alu_scoreboard]  Value of A = 0
Value of B = 15 Value of Result = 17
# UVM_ERROR tb/alu_scoreboard.sv(53) @ 2395: uvm_test_top.env.scoreboard [alu_scoreboard] Substraction
Failed
# ----------------------------------------------------------------------------------
```

Figure 157 - Subtraction Error

5.2. SerDes

The RTL code of the *SerDes*, found in the Appendix, is taken over from the work done in the 2017 thesis [1]. The authors encourage the reader to go through this document for an in-depth understanding of the *SerDes* functionality.

The *SerDes* comprises digital modules in SV HDL and analog modules designed through schematics employing analog components. For this case study, the analog modules are not being validated and just the digital ones are targeted. The analog modules in this case are just passthrough dummy modules.

This *SerDes* system has three main stages, transmission, reception, and testing. Both the transmission and the reception stages are composed of an analog and a digital block, while the testing stage is entirely digital.

The overall structure of the *SerDes* is presented in Figure 158.

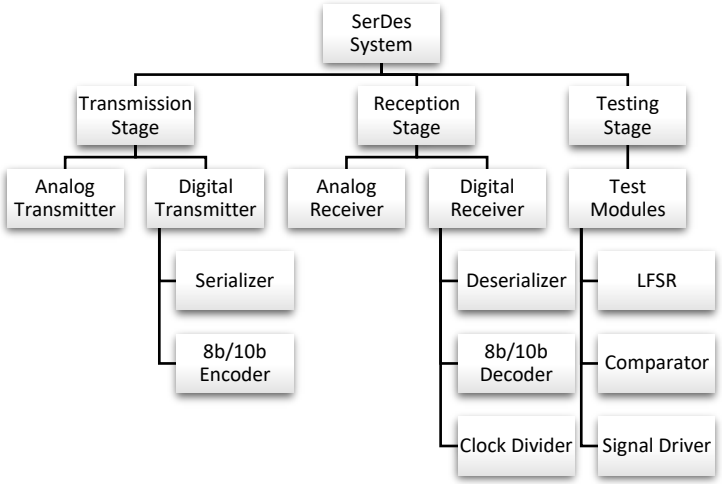


Figure 158 - SerDes structure

The transmission stage function is to receive a 9-bit parallel data, encode it to 8b/10b and transmit a serialized data for the PCI express protocol.

The digital block of the transmission can convert a parallel data bus into a serial data format. It encodes the data using 8b/10b codification, meets the specifications of speed, and provides a transmission clock signal to synchronize the circuit on when to send the data. It is composed of two modules: a serializer and an encoder.

The reception stage receives a serial differential input data encoded in 8b/10b and will provide an output of a parallel digital decoded 9-bit data.

The digital block of the reception can convert the received serial data to parallel data and align it with a system clock. It proposes a recovery scheme based on the clock and sampling data. As the received data is encoded, it also performs the 8b/10b decoding process while meeting the specifications of speed. It is composed of three modules: a de-serializer, an encoder, and a clock divider.

The testing stage adds verification capabilities to the chip´s architecture by using control signals and design-for-testability (DFT) techniques that allow testing of the final *SerDes* circuit once manufactured. It is composed of three modules: a comparator, a linear-feedback shift register (LFSR), and a signal driver.

5.2.1. System specifications

The SerDes chip has 40 pins, 4 of them are for power, and 36 are divided for input and output signals. The following Table 40 shows all the external inputs and outputs pins of the SerDes circuit.

Type	Signal name	Pins	Description
Inputs	rst	1	Global reset active high
	clk	1	Global clock
	rxa_in_p	1	Positive input for the analog receiver
	rxa_in_n	1	Negative input for the analog receiver
	config_in[7:0]	8	This is the input configuration for either the test modules or the analog transmitter. test_en = 0: analog configuration. test_en = 1: test configuration. config_in[2:0]:selects the testmode config_in[3]: input test patterns config_in[4]: errors_en. Applies for mode 4 and 7. {1]: test_data= {disp_err, code_err, dispout, num_errors[5:0]} {0}: {disp_err, code_err, dispout, rxa_out, bist_end, num_erros[3:0]} config_in[5]: test_out_mux_sel. Choose between the rxd output data or test data in the rxd output.
	txd_data_in [8:0]	9	Parallel data in for the digital transmitter
	test_en	1	Signal that enables the test inputs for different testing modes instead of the analog transmitter configuration pins.
Outputs	digital_out[8:0]	9	This can be either the error number or the digital receiver output depending on bit [4] of the config_in signal.
	txd_data_out	1	Digital transmitter output

	tx_frame_start	1	Sync signal of the digital serializer
	c_data_valid	1	Data valid of the digital receiver
	txa_data_out_n	1	Positive output of the analog transmitter.
	txa_data_out_p	1	Negative output of the analog transmitter.
Power	VDD	1	Analog positive power supply.
	VSS	1	Analog negative or grounded power supply.
	DVDD	1	Digital positive power supply.
	DVSS	1	Digital negative power supply.

Table 40 - SerDes pins

5.2.2. SerDes Operation Modes

The SerDes has a total of 8 operation modes listed in Table 41. Validating the behavior of all these modes is the key aspect of the SerDes validation.

Available test operation modes	
Mode	**config_in[2:0]**
0 – SerDes operation	3'b000
1 – Parallel loopback	3'b001
2 – Serial loopback	3'b010
3 – RXA bypass	3'b011
4 – BIST with serial Loopback	3'b100
5 – RXA bypass with parallel loopback	3'b101
6 – Open BIST	3'b110
7 – RXA output with analog loopback	3'b111

Table 41 - SerDes operation modes

5.2.2.1. Mode 0: Functional mode

The mode 0, shown in Figure 159, is the normal operation mode of the SerDes. This functional mode consists of a free run of the system that can receive transmission transactions in a full-duplex mode. This is the basic configuration of the SerDes.

The testing modes are described in the following sections, analyzing in detail all the possible flows supported by this mode of operation, such as the SerDes as the initiator of the transmission process, the SerDes as the final destination of a transaction, connections in loopback and the DFX capabilities. For this case, *test_en* is 1'b0 and the RX and TX stages

are working independently.

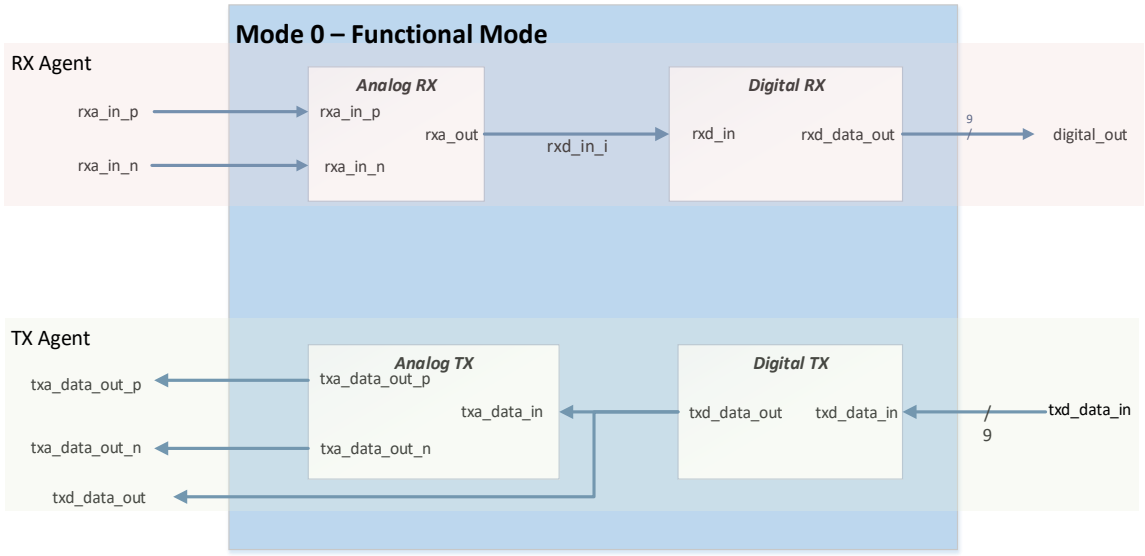


Figure 159 - Mode 0: functional mode

5.2.2.2. Mode 1: Parallel Loopback

In this mode, the SerDes has an internal parallel loopback created between the digital receiver output and the digital transmitter output. The input data is serially coming from the Analog RX and it is expected to match with the output data in the TX Agent. This mode is pictured in Figure 160.

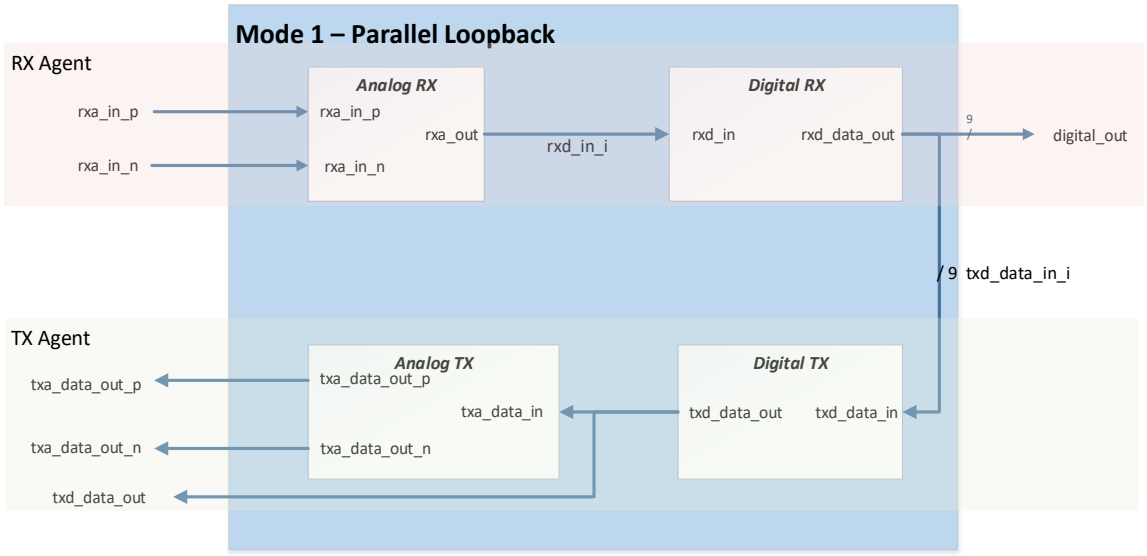


Figure 160 - Mode 1: Parallel loopback

5.2.2.3. Mode 2: Serial Loopback

In this mode the analog receptor module is isolated while the digital transmitter output is connected through internal serial feedback to the digital receiver input. The input data this time is coming parallelly through the Digital TX and it is expected to match with the data coming out of the digital RX. This mode is illustrated in Figure 161.

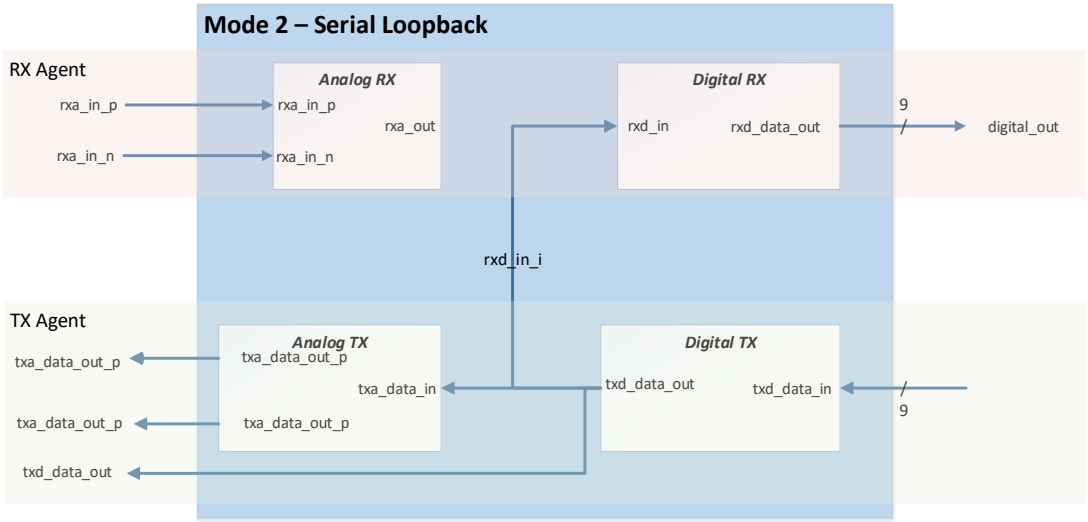


Figure 161 - Mode 2: Serial Loopback

5.2.2.4. Mode 3: RXA Bypass

The RXA Bypass mode of operation shown in Figure 162, was devised to overcome any malfunctioning of the Analog RX. It uses the config_in[3] pin as serial input data instead of the analog pins rxa_in_p and rxa_in_n. This way, the analog receiver is isolated.

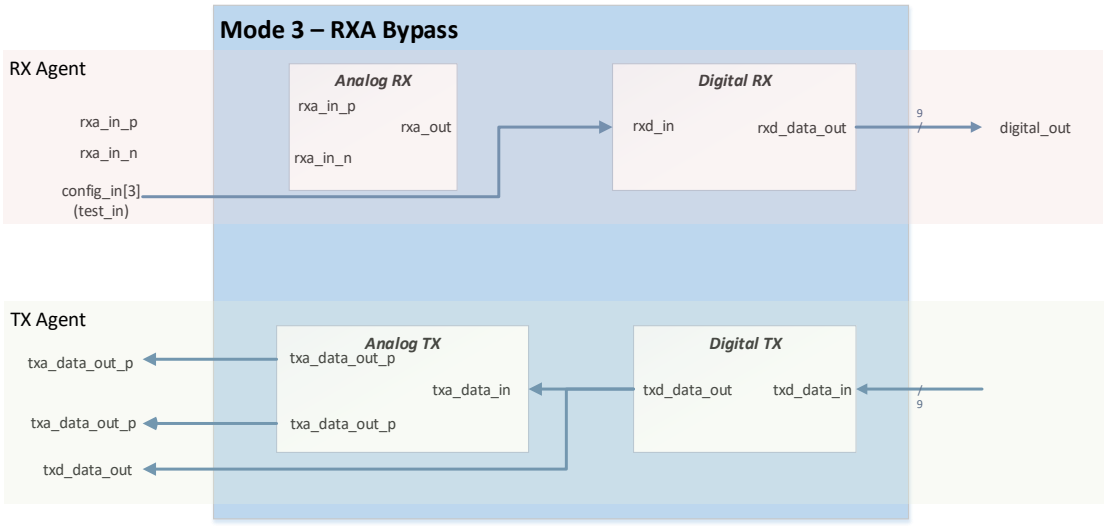


Figure 162 - Mode 3: RXA Bypass

5.2.2.5. Mode 4: BIST with serial loopback

In this mode shown in Figure 163, the LFSR generates pseudo-random parallel patterns that are injected as inputs to the digital transmitter, then the serial output of the Digital TX is connected as an input to the Digital RX. There is an internal comparator that will check if errors are presented within this loop. In this mode, the Analog RX is not used and maintained isolated.

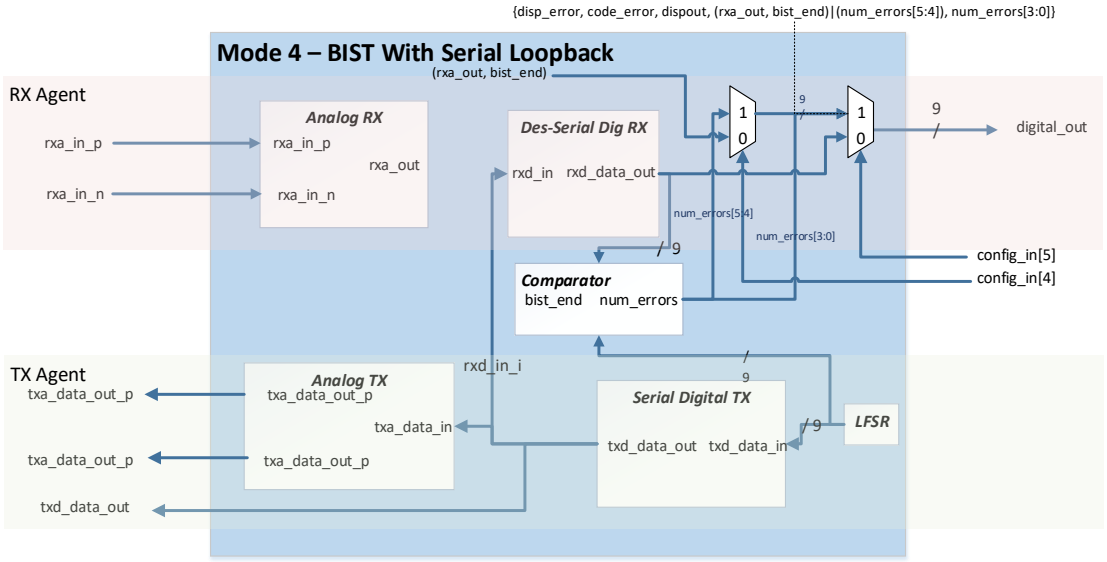


Figure 163 - Mode 4: BIST with serial loopback

5.2.2.6. Mode 5: RXA bypass with parallel loopback

This mode, illustrated in Figure 164, resembles a combination of operation mode 1 with operation mode 3. Therefore, the analog receiver is isolated and there is a connection of the parallel data from the digital receiver output to the digital transmitter input. In this case, the input from config_in[3] should match the output serial data on txa_data_out_p

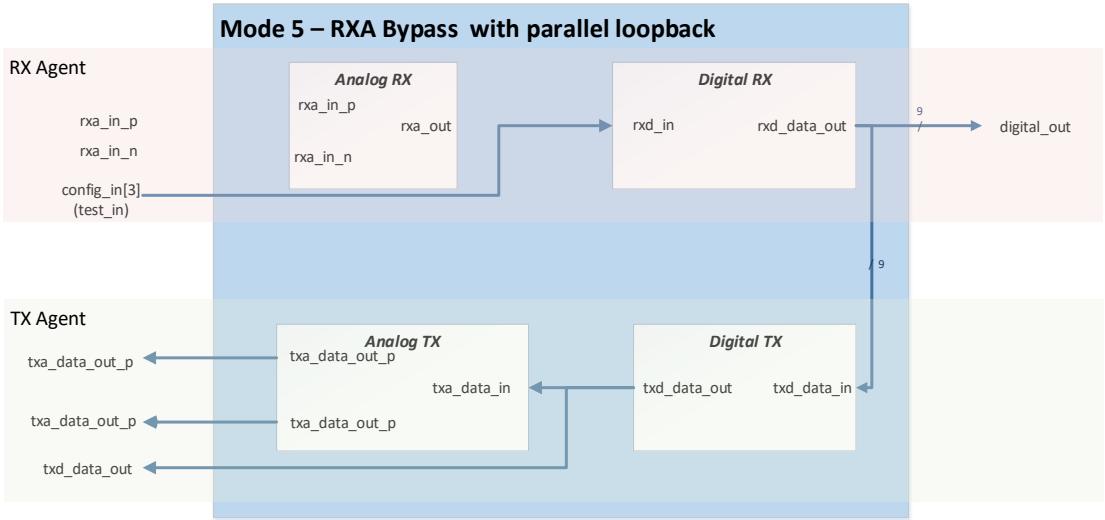


Figure 164 - Mode 5: RXA bypass with parallel loopback

5.2.2.7. Mode 6: Open BIST

This mode resembles the operation mode 4: BIST with serial loopback, however, as shown in Figure 165, this time there is no internal serial loopback and the analog receiver gets external data. For the testbench, the Analog TX output is connected to the Analog RX input.

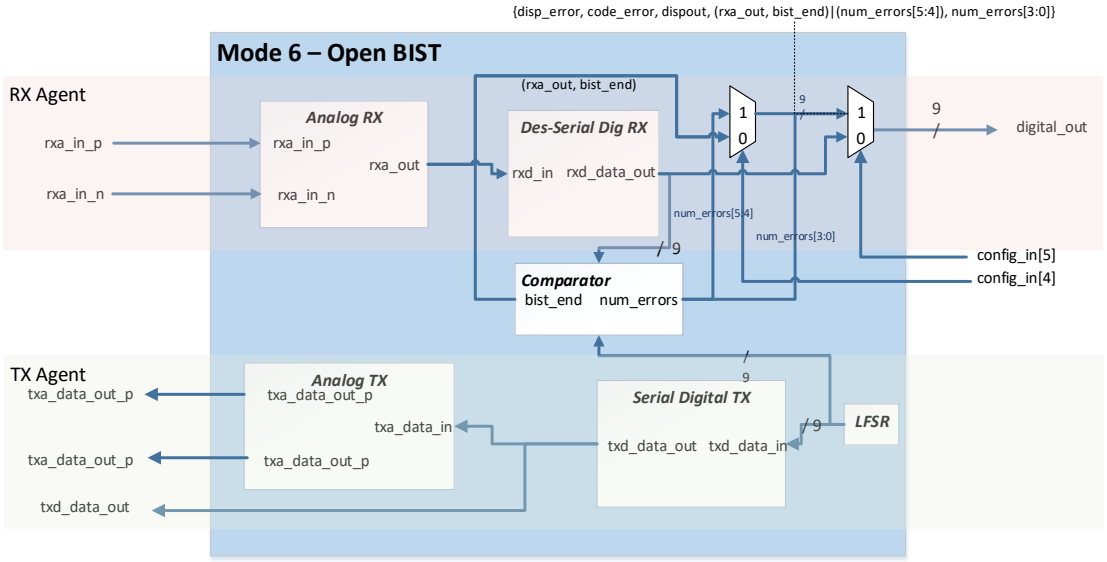


Figure 165 - Mode 6: Open BIST

5.2.2.8. Mode 7: RXA output with analog loopback

This mode, in Figure 166, is used to check the behavior of the analog modules. There is an internal analog loopback between the analog receiver output and the analog transmitter input. There is also the possibility to drive the Analog RX output by changing the configuration of pins config_in[4] and config_in[5].

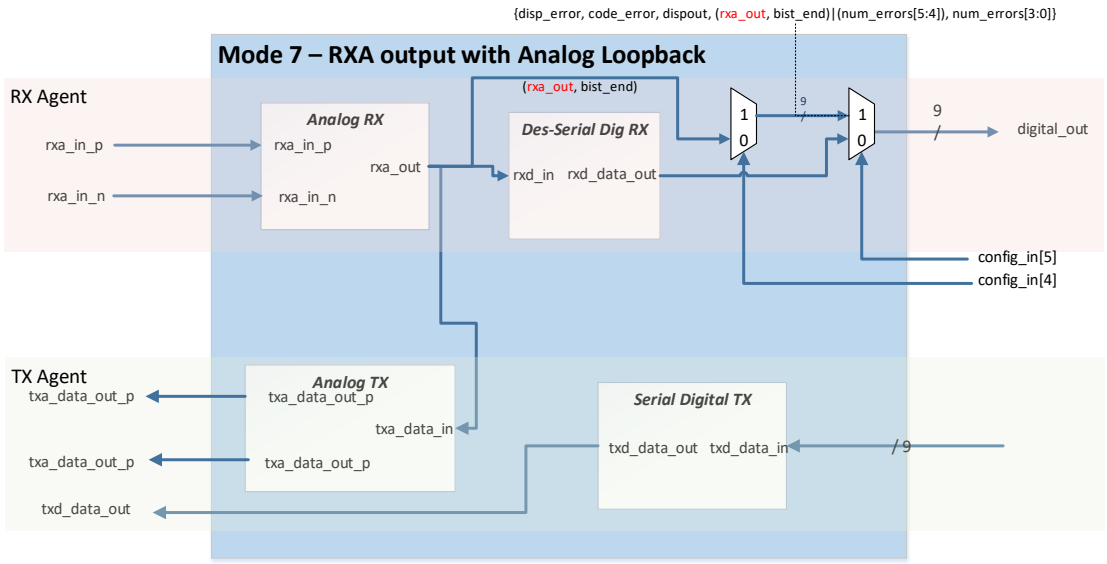


Figure 166 – Mode 7: RXA Output with Analog Loopback

5.2.3. Test Plan

The validation for the SerDes consists of focused tests for each of the operation modes defined in the specification. Parallel or serial data is being driven as input to the DUT and depending on the operation mode, the output data should match the expected values.

The verification architecture presented in this chapter is defined by a pair of UVM Agents connected to a single Scoreboard and the Virtual Interface. There is one Agent for managing all the receive transactions (RX Agent) and another Agent to take care of the transmit transactions (TX Agent). Each one of the mentioned is composed of a Monitor, a Sequencer, a Driver, and a Subscriber used for coverage.

All operation modes should be targeted by a focus test along with a custom checker to validate its behavior. Two coverage blocks were developed, but only a few covergroups were considered. Exhaustive coverage verification was left out of scope for this case study. Nevertheless, a 100% coverage is required.

5.2.3.1. Required External Receivables:

Receivable	From Whom
SerDes Spec	ITESO TV1 SerDes chip
RTL DUT	ITESO Design
UVM testbench template	UVM Framework tool

5.2.3.2. Tests

The SerDes is a complex design that has several operation modes. A test for each operation mode was developed.

Focused Test	Description
serdes_base_test	Base test of the SerDes.
serdes_functional_test	Functional mode of operation.
serdes_parallel_loopback_test	Parallel loopback mode of operation.
serdes_serial_loopback_test	Serial loopback mode of operation.
serdes_rxa_bypass_test	Bypass of the Analog RX.
serdes_bist_serial_loopback_test	BIST enabled with serial loopback.
serdes_rxa_bypass_parallel_loopback_test	Bypass of the Analog RX with parallel loopback.
serdes_open_bist_test	Open BIST mode.
serdes_rxa_output_analog_loopback_test	Analog loopback with Analog RX output.

Table 42 - SerDes Tests

5.2.3.3. Monitoring

The SerDes has two monitors (RX Monitor and TX Monitor) to snoop through the virtual interface signals, one for every agent (RX_Agent and TX_Agent). These will use the validation signals rx_transmit and tx_transmist respectively to control when to start snooping. Table 43 contains all the signals being monitored.

Monitored Signal	Monitor
tx_transmit	TX Monitor
tx_frame_start	TX Monitor
txa_data_out_p	TX Monitor
txa_data_out_n	TX Monitor
txd_data_out	TX Monitor
txd_data_in	TX Monitor
rx_transmit	RX Monitor

C_data_valid	RX Monitor
test_en	RX Monitor
Config_in	RX Monitor
Digital_out	RX Monitor
Rxd_output_muxout_wire	RX Monitor
Rxd_input_muxout_wire	RX Monitor
Rxa_in_p	RX Monitor
Rxa_in_n	RX Monitor

Table 43 - Monitored Signals

5.2.3.4. Checks

To ensure the proper functioning of the SerDes, all operation modes should be checked. There is one checker for each operation mode of the SerDes.

Check	**Mode (config_in[2:0])**	**Implementation Description**
Functional Mode	1'b000	
Parallel Loopback	1'b001	
Serial Loopback	1'b010	
RXA Bypass	1'b011	
BIST with serial Loopback	1'b100	
RXA bypass with parallel loopback	1'b101	
Open BIST	1'b110	
RXA output with Analog Loopback	1'b111	

Table 44 - SerDes Checks

5.2.3.5. Coverage

The following coverpoints were defined to assure that all typical scenarios of the SerDes are being exercised. A 100% coverage of the defined coverage should be hit.

Cover point	**Description**	**Type**	**Environment**
bist_end_cp	BIST end asserted.	Coverpoint	RX Agent
test_mode_cp	All test modes hit.	Coverpoint	RX Agent
config3_cp	Input data through config_in[3]	Coverpoint	RX Agent
comma_cp	Comma on the receiver.	Coverpoint	RX Agent
config_4_cp	config_in[4] mux hit.	Coverpoint	RX Agent
config_5_cp	config_in[5] mux hit.	Coverpoint	RX Agent

num_errors_cp	Number of errors hit	Coverpoint	RX Agent
comma_cp	Comma on the transmitter.	Coverpoint	TX Agent
test_mode_cp	All test modes hit.	TX Agent	TX Agent

Table 45 - SerDes Coverpoints

5.2.3.6. Stimulus

For driving the stimuli, some tasks were developed to drive a given data in parallel or serial.

Stimulus tasks	**Description**
Parallel	The RX Driver will inject the serial data into the DUT, this data can come through the RX Analog inputs or config_in[3] depending on the configuration mode.
Serial	The TX Driver will inject the parallel data into the DUT via the txd_data_in signal.

Table 46 - SerDes Stimulus Tasks

5.2.4. Testbench

Figure 167 shows the relationship between the verification components. The SerDes is the design-under-test (DUT) module. The verification components used in this testbench will be described below, however, the SystemVerilog code for them will be placed in the Appendix section on this document.

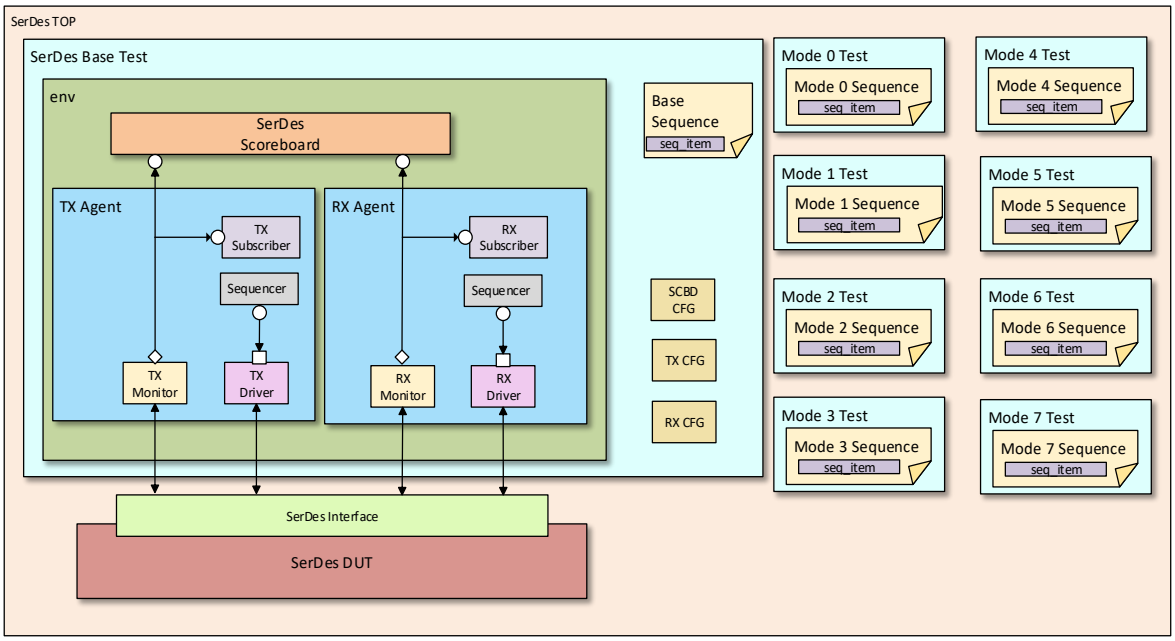


Figure 167 - SerDes Validation Testbench

5.2.4.1. SerDes Interface

The interface for the SerDes contains all the inputs and outputs of the SerDes DUT plus the addition of some extra signals, such as internal signals to be spied by the monitor and control signals such as *rx_transmit* and *tx_transmit* used as an indicator to the drivers' transmission status.

5.2.4.2. SerDes Sequence Item

The *serdes_sequence_item* consists of the data fields that traveling through the SerDes.
The input signals have been declared as rand to add the randomization capabilities while the output, control ,and spy signals do not need this declaration.
It is worth mentioning the field *data_transmit* is a helper used to drive serial signals more understandably.
The constraint added is such that the analog signals always have their inverted counterpart.

5.2.4.3. SerDes Sequencer

The SerDes sequencers are standard codes to send the sequence items to the drivers. There is for each agent: RX Sequencer and TX Sequencer.

5.2.4.4. SerDes Driver

The serdes_tx_driver drives the parallel data coming into the Digital TX. The data must remain active for 10 clocks to be valid. In this case, a configuration object was not required.

The serdes_rx_driver is in charge of all the serial transactions coming to the Analog RX. This driver contains a configuration object for such operation modes in which the serial incoming data is being driven by config_in[3] instead of the normal rxa_in_p and rxa_in_n.
The drive () task will use the slow clock of the SerDes to send a given 10-bits value serially.

5.2.4.5. SerDes Monitor

There are two monitors within the SerDes, one per agent, these are where the scoreboard will obtain information used for pass/fail criteria for each of the modes. In this testbench architecture, the subscribers will also obtain the information from the monitors for capturing coverage.
Both monitors are snooping parallel and serial data. TX Monitor all the information passing through the transmitter block and the RX Monitor all the data traffic of the receiver's modules. The TX Monitor normally gets the serial information in sets of 10 bits, but it can make use of the config object to monitor the serial input bit by bit. This is especially useful for mode 7.

5.2.4.6. SerDes Subscriber

The TB has two subscribers that are used for coverage collection. One subscribed to the RX Monitor and one subscribed to the TX Monitor. The coverage analysis done for the SerDes was basic but enough to prove how the tool can create these blocks.

5.2.4.7. SerDes Scoreboard

The SerDes scoreboard was the most challenging to code, as each test mode has different specific to be checked. Specific checks for each mode had to be coded and determined by a scoreboard configuration object (scbd_cfg).

The overall behavior for most cases is as follows. The incoming data was split in parallel or serial. Then this data can be golden (expected) or actual data. The golden data contains the transactions that are being injected into the DUT while the actual data is the processed data

coming out of the RTL. The golden data is treated as a reference to compare against the actual data. To keep the data to be compared, all the transactions received are being stored in analysis FIFOs. When these FIFOs got all incoming data, the transactions are compared to determine if there is a mismatch or not.

5.2.4.8. SerDes Config Object

The SerDes configuration object is used to set the three things: the testmode, if coverage is enabled, and if the respective driver is active or passive. All the config objects are set within the UVM Tests.

5.2.4.9. SerDes Agent

There are two SerDes agents in this testbench, one for the transmission blocks and one for the reception blocks.
The RX Agent encapsulates the rx_sequencer, the rx_driver, the rx_monitor, and the rx_subscriber. The TX Agent encapsulates the tx_sequencer, the tx _driver, the tx_monitor, and the tx_subscriber. Both have their respective configuration object to enable the coverage and to determine if the drivers are active or passive. In this validation infrastructure, only one driver is active at the same time.

5.2.4.10. SerDes Environment

The SerDes Env is a container that has both agents (RX Agent and TX Agent) and the Scoreboard. The scoreboard is connected to both monitor´s analysis port to obtain the transactions´ information.

5.2.4.11. SerDes Sequence

The SerDes Testbench will use a base sequence as a parent class from which all the other sequences will inherit. Each of the tests has its corresponding sequences. In the current testbench there no test case that uses more than one sequence.
The specific sequences for each test are very similar with minor adjustments to configure the correct mode being tested.

5.2.4.12. SerDes Test

Similar to the SerDes sequences, this testbench uses a base test as a parent class from

which all other test classes will inherit from. There is also one test for each of the configuration modes.

The base test contains all the configuration objects used for the validation, but the actual set of these config objects is done at each test individually.

5.2.4.13. SerDes TB TOP

The SerDes TB Top is where the RTL gets instantiated and connected to the virtual interface. It also contains the clock and reset generation. In the Top Testbench, also the virtual interface is set from the configuration database. This code connects some internal signals to the virtual interface so that they can be spied by all the UVCs.

5.2.4.14. SerDes Environment Package

A UVM Package is a complete collection of all Classes that are needed to build a verification Environment. All the testbench files required for the compilation of this testbench are listed in the serdesEnvPkg in Figure 168.

```
`include "uvm_macros.svh"
import uvm_pkg::*;

package serdesEnvPkg;
`include "tb/serdes_config.sv"
`include "tb/serdes_seq_item.sv"
`include "tb/serdes_tx_subscriber.sv"
`include "tb/serdes_rx_subscriber.sv"
`include "tb/serdes_sequence_base.sv"
`include "tb/serdes_parallel_loopback_sequence.sv"
`include "tb/serdes_serial_loopback_sequence.sv"
`include "tb/serdes_bist_serial_loopback_sequence.sv"
`include "tb/serdes_rxa_bypass_sequence.sv"
`include "tb/serdes_rxa_bypass_parallel_loopback_sequence.sv"
`include "tb/serdes_open_bist_sequence.sv"
`include "tb/serdes_rxa_output_analog_loopback_sequence.sv"
`include "tb/serdes_rx_sequencer.sv"
`include "tb/serdes_tx_sequencer.sv"
`include "tb/serdes_rx_driver.sv"
`include "tb/serdes_tx_driver.sv"
`include "tb/serdes_rx_monitor.sv"
`include "tb/serdes_tx_monitor.sv"
`include "tb/serdes_scoreboard.sv"
`include "tb/serdes_rx_agent.sv"
`include "tb/serdes_tx_agent.sv"
`include "tb/serdes_env.sv"
`include "tb/serdes_base_test.sv"
`include "tb/serdes_parallel_loopback_test.sv"
`include "tb/serdes_serial_loopback_test.sv"
`include "tb/serdes_bist_serial_loopback_test.sv"
`include "tb/serdes_rxa_bypass_test.sv"
`include "tb/serdes_rxa_bypass_parallel_loopback_test.sv"
`include "tb/serdes_open_bist_test.sv"
`include "tb/serdes_rxa_output_analog_loopback_test.sv"
endpackage: serdesEnvPkg
```

Figure 168 - SerDes EnvPkg Code

5.2.5. Simulation

In the Validation of the SerDes, all simulations were done in two different commercial simulators: Questa Sim from Mentor graphics and VCS from Synopsis. The results were similar in both cases.

The testbench has been compiled without any errors in both Questa Sim and Synopsis.

The command used to run the SerDes' simulations, depending on the chosen test is:

```
vsim work.serdes_tb_top +UVM_TESTNAME=<TEST_NAME> -voptargs=+acc=lprn+top
```

During the simulation, it is printed the topology of the SerDes in the end_of_elaboration phase and is shown in Figure 169.

The testbench works as expected and the signals are driven with the applied stimuli. The simulation for each test is presented below:

```
# ----------------------------------------------------------------------------------
# Name                          Type                                    Size  Value
# ----------------------------------------------------------------------------------
# uvm_test_top                  serdes_rxa_output_analog_loopback_test  -     @472
#   env                         serdes_env                              -     @482
#     rx_agent                  serdes_rx_agent                         -     @501
#       rx_driver               serdes_rx_driver                        -     @532
#         rsp_port              uvm_analysis_port                       -     @547
#         seq_item_port         uvm_seq_item_pull_port                  -     @539
#       rx_monitor              serdes_rx_monitor                       -     @664
#         rx_item_collected_port uvm_analysis_port                      -     @673
#       rx_sequencer            serdes_rx_sequencer                     -     @555
#         rsp_export            uvm_analysis_export                     -     @562
#         seq_item_export       uvm_seq_item_pull_imp                   -     @656
#         arbitration_queue     array                                   0     -
#         lock_queue            array                                   0     -
#         num_last_reqs         integral                                32    'd1
#         num_last_rsps         integral                                32    'd1
#     scoreboard                serdes_scoreboard                       -     @515
#       rx_fifo                 uvm_tlm_analysis_fifo #(T)              -     @704
#         analysis_export       uvm_analysis_imp                        -     @743
#         get_ap                uvm_analysis_port                       -     @735
#         get_peek_export       uvm_get_peek_imp                        -     @719
#         put_ap                uvm_analysis_port                       -     @727
#         put_export            uvm_put_imp                             -     @711
#       rx_item_collected_export uvm_analysis_export                    -     @688
#       tx_fifo                 uvm_tlm_analysis_fifo #(T)              -     @751
#         analysis_export       uvm_analysis_imp                        -     @790
#         get_ap                uvm_analysis_port                       -     @782
#         get_peek_export       uvm_get_peek_imp                        -     @766
#         put_ap                uvm_analysis_port                       -     @774
#         put_export            uvm_put_imp                             -     @758
#       tx_item_collected_export uvm_analysis_export                    -     @696
#     tx_agent                  serdes_tx_agent                         -     @508
#       tx_monitor              serdes_tx_monitor                       -     @800
#         tx_item_collected_port uvm_analysis_port                      -     @807
# ----------------------------------------------------------------------------------
```

Figure 169 - SerDes topology

5.2.5.1. Mode 0: Functional Mode

The validation of the functional mode is covered by executing all the test modes. The test modes include the functionality of this mode and verify the correctness of the system. All the modes from 1 to 7 validate exhaustively the features of the mode 0 and conform a subset of its behavior.

The selected test modes of the design execute in depth all the features and the different transactions flows that are done within the SerDes.

5.2.5.2. Mode 1: Parallel Loopback

In this mode the RX Agent is Active, and the TX Agent is Passive. This means that the data is being driven serially to the Analog RX via *rxa_in_p* and *rxa_in_n* pins. This data is divided into segments of 10 bits, being the first 10 bits sent a “comma”.

These transactions, which are considered golden, are stored in a FIFO to be compared later. Once the parallel loopback is completed, it is expected that the same data that got into the system matches with the Analog TX output serial data. This way it is ensured that the serialization and deserialization are successful. The simulation that shows this mode can be displayed in Figure 170.

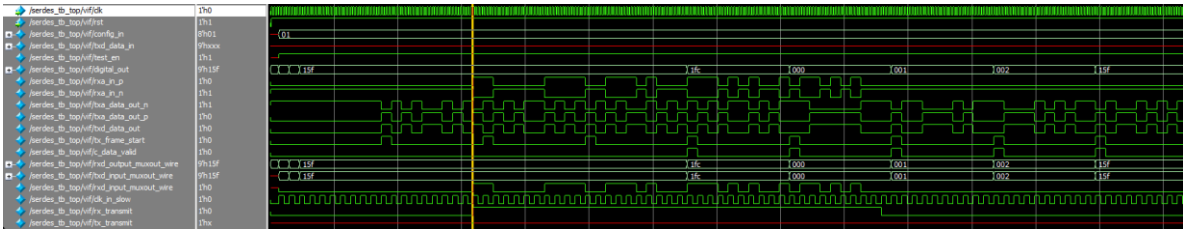


Figure 170 - Parallel loopback simulation

5.2.5.3. Mode 2: Serial Loopback

In this mode the TX Agent is Active, and the RX Agent is Passive. The data input data comes parallelly in the *txd_data_in* pin of the digital transmitter. This data is being stored in a FIFO and compared against the output parallel data in *digital_out* pins. This way the validator ensures that the serialization and deserialization process is equivalent. This is presented in Figure 171.

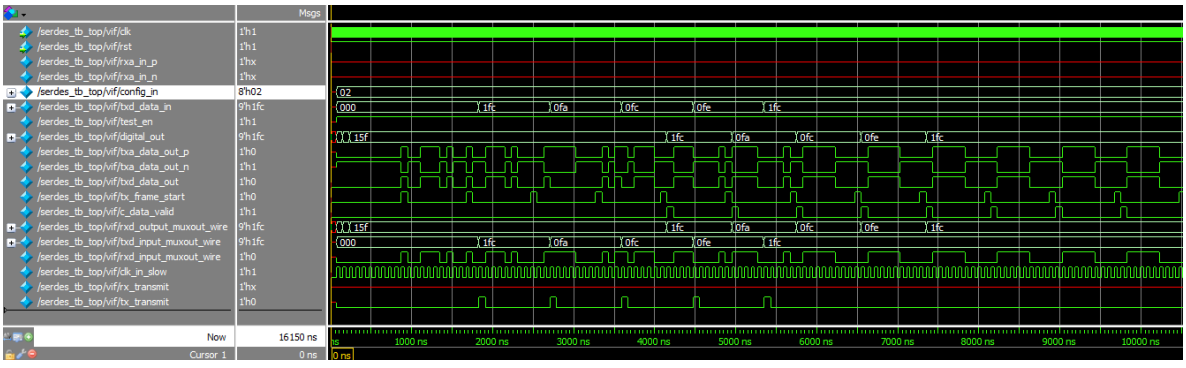


Figure 171 - Serial Loopback Simulation

5.2.5.4. Mode 3: RXA Bypass

In this mode, the RX Agent is Active, and the TX Agent is Passive. In this mode it is checked that the input to the digital received matches with the incoming data in from the *config_in[3]*

pin. This mode is showcased in Figure 172.

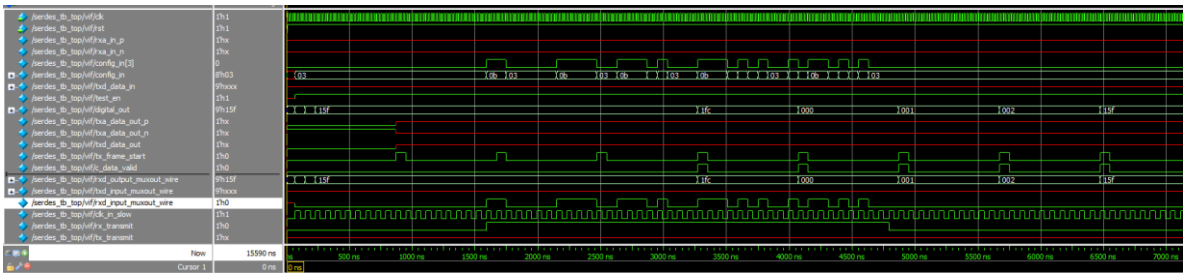


Figure 172 - RXA Bypass simulation

5.2.5.5. Mode 4: BIST with Serial Loopback

In this mode the TX Agent is Active, and the RX Agent is Passive. In this mode, the checker is similar to the one in mode 2, but the main difference is that instead of checking the input data from the *txd_data_in*, the monitor is snooping the internal signal *rxd_input_muxout_wire*, which in the output of the LFSR. The checker will the data and compare when the LFSR has completed its cycle of 64 transactions. Figure 173 presents the simulation of this configuration mode and Figure 174 shows a segment of the simulation log.

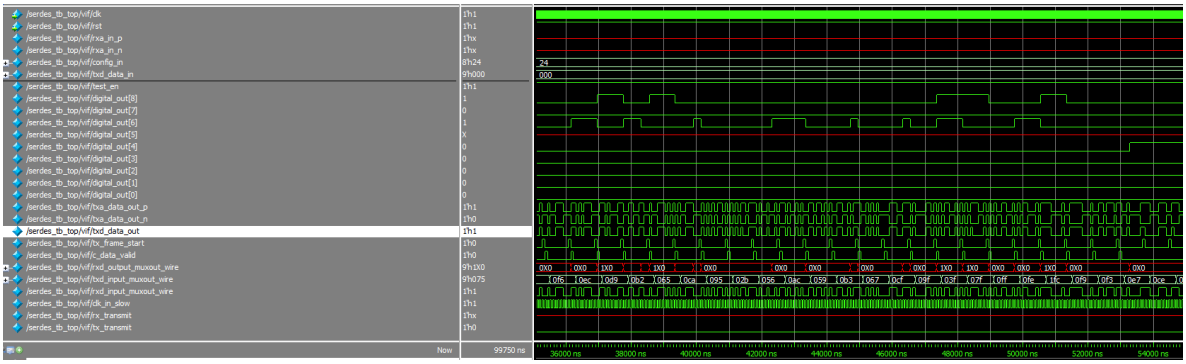


Figure 173 - BIST with Serial Loopback simulation

```
# UVM_INFO tb/serdes_scoreboard.sv(266) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] PAR GOLDEN Q SIZE:          64
# UVM_INFO tb/serdes_scoreboard.sv(267) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] PAR ACTUAL Q SIZE:          64
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 1fc Actual: 1fc
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0f9 Actual: 0f9
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0f3 Actual: 0f3
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0e7 Actual: 0e7
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0ce Actual: 0ce
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 09d Actual: 09d
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 03a Actual: 03a
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 075 Actual: 075
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0eb Actual: 0eb
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0d7 Actual: 0d7
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0af Actual: 0af
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 05e Actual: 05e
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0bc Actual: 0bc
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 079 Actual: 079
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0f2 Actual: 0f2
# UVM_INFO tb/serdes_scoreboard.sv(276) @ 53030: uvm_test_top.env.scoreboard [serdes_scoreboard] Stream match. Expected: 0e5 Actual: 0e5
```

Figure 174 - BIST with Serial Loopback log

5.2.5.6. Mode 5: RXA Bypass with Parallel Loopback

In this mode, the RX Agent is Active, and the TX Agent is Passive. In this mode the same checks for the parallel loopback are in place, with the only difference that the input data is coming from the *config_in[3]* pin instead of the *rxa_in_p* and *rxa_in_n*. This can be observed in Figure 175

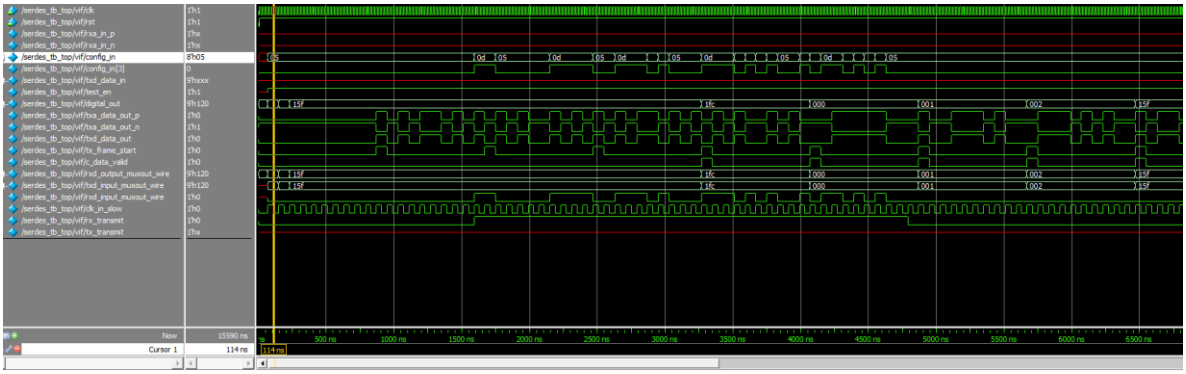


Figure 175 - RXA Bypass with Parallel loopback Simulation

5.2.5.7. Mode 6: Open BIST

In this mode the RX Agent is Active, and the TX Agent is Passive. This mode validates that the test modules' comparator is correctly behaving by detecting mismatches in the parallel data. The used approach is to send a "comma" followed by a fixed data value in the Analog RX input for the 62 transactions that the LFSR is generating data and a final transaction with a "comma" to assert *bist_end*. Reaching a total of 63 transactions sent by the LFSR.

Apart from the "commas", the data is going to be different from the LFSR output data, so the number of errors presented in the *digital_out[6:0]* should be a known value of 61. If there aren´t 61 errors when *bist_end* is asserted, then the test will fail.

It is needed to configure the *digital_out* to show all 6 bits of *num_errors* so the signals *config_in[4]* and *config_in[5]* should be set to 1'b1. Figure 176 presents the behavior of this mode. With this test, it is guarantee that the comparator is working correctly.

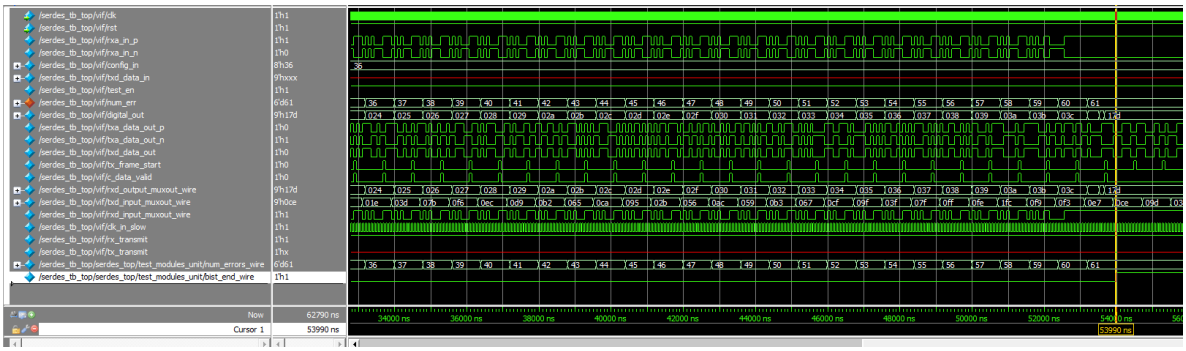


Figure 176 - Open BIST Simulation

5.2.5.8. Mode 7: RXA output with analog loopback

In this mode the RX Agent is Active, and the TX Agent is Passive. This mode checks that the analog loopback is correct, meaning that both analog inputs *rxa_in_p* and *rxa_in_n* are matching with *txa_data_out_p* and *txa_data_out_n* respectively.

This mode also configures the DFX multiplexors using config_in[5] to drive the analog output *rxa_out* through the *digital_out[5]* pin. This is also checked by the scoreboard. The waveform from Figure 177 represents this mode.

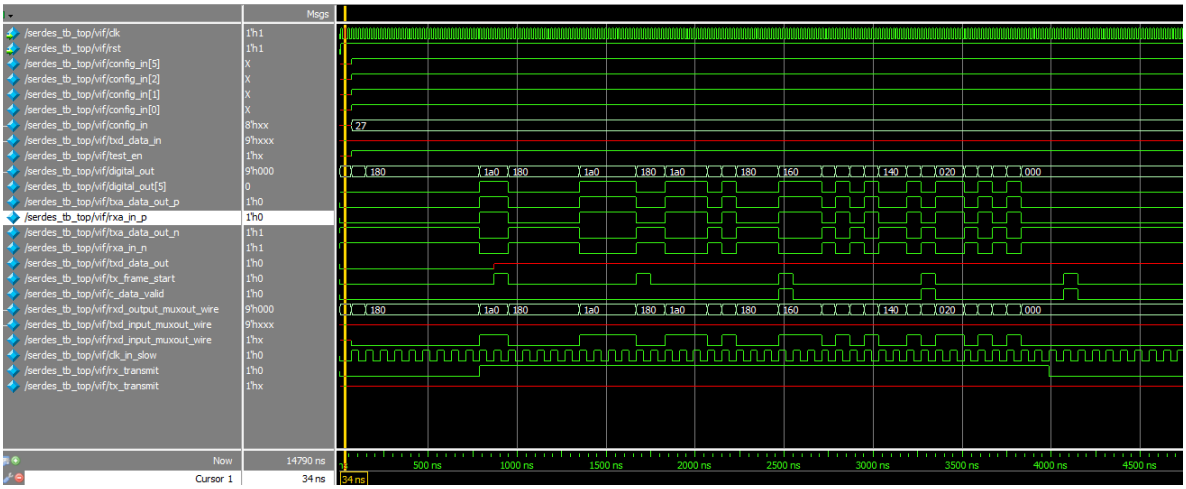


Figure 177 - RXA output with analog loopback simulation

5.2.6. Coverage Report

It is desired to collect coverage from the complete set of tests. In Questa Sim, the database for each test needs to be saved independently before merging the complete database. When all the databases are ready, they can be merged in a single file which will have the collective coverage of all the tests.

The QuestaSim command to save the coverage database for each test:

```
coverage save -assert -directive -cvg -codeAll <test_name>.ucdb
```

The QuestaSim command to merge the databases is:

```
vcover merge -64 merge.ucdb *.ucdb
```

The QuestaSim command to generate the coverage report from the merged database:

```
vcover report -details -html merge.ucdb
```

The overall coverage collected for the SerDes is 95.75% which is almost enough to hit the coverage target. However, the misses are analyzed to be due to the err_count of the

comparator being stopped on purpose before reaching the max count, so these misses can be waived. Let´s just keep in mind that the coverage definition was basic and a more in-depth coverage analysis of the SerDes was left as future work. The details of the coverage hit/miss for each covergroup can be found below:

Score	Total	CVG
TOTAL	**95.75%**	**95.75%**
serdesEnvPkg	**95.75%**	**95.75%**
serdes_tx_subscriber	**93.75%**	**93.75%**
serdes_rx_subscriber	**97.76%**	**97.76%**

Table 47- SerDes Total Coverage Summary

- **TX_CG:**

Summary	Total Bins	Hits	Hit %
Coverpoints	9	9	**100.00%**

Table 48 - SerDes TX CG Summary

comma_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
test_mode_cp	8	7	1	**100.00%**	**100.00%**	**100.00%**
CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage%

Table 49 - SerDes TX CG Details

- **RX _CG:**

Summary	Total Bins	Hits	Hit %
Coverpoints	81	79	**97.53%**

Table 50 - SerDes RX CG Summary

bist_end_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
comma_cp	1	1	0	**100.00%**	**100.00%**	**100.00%**
config3_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
config4_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
config5_cp	2	2	0	**100.00%**	**100.00%**	**100.00%**
num_error_cp	64	62	2	**96.87%**	**96.87%**	**96.87%**
test_mode_cp	8	8	0	**100.00%**	**100.00%**	**100.00%**
CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage%

Table 51 - SerDes RX CG Details

5.2.7. Bug Report

- **Encoded vs decoded comma**

In the Parallel Loopback mode, it was found that the commas received and sent are not matching. The encoded comma is 10'h07C and the decoded comma is 9'h1FC. In this simulation, an encoded comma of 10h'07C is expected but a 10'h383 value is received instead. This value is the inverted value of 10'h07C. Although the RTL can receive "commas" in both channels p and channels n, the expectation is that the received value stays the same.

```
# UVM_ERROR tb/serdes_scoreboard.sv(161) @ 4870: uvm_test_top.env.scoreboard
[serdes_scoreboard] Stream mismatch. Expected: 383 Actual: 07c
```

- **Infinite BIST Loop**

The *bist_end* signal is expected to assert when the data comparison between upon reception of two "commas". The Open BIST mode showed that this works fine unless there are errors in the commas received. This will cause the LFSR to send transactions in an infinite state even if the LFSR loop has been reached. The RTL fix to avoid this issue is to assert bist_end not when the "commas" have been compared but when the LFSR loop is completed.

- **Number of errors overflow**

If there are more errors than the maximum number permitted (63), like in the above case where BIST didn´t catch the two commas and didn´t stop sending data. The errors will eventually overflow, and the actual number of errors will be lost to the user. If the maximum number of errors were reached, the counter should remain with the maximum count or an

additional flag should be used to highlight this scenario.

- **Test_out multiplexor**

The *test_out* pin was removed in one of the last RTL fixes during the SerDes integration. It was decided to replace this pin by adding two multiplexors to select this signal in the outputs. While this works correctly, it is a design flaw as it prevents the user from having visibility of *bist_end* and the maximum number of errors at the same time, which is important for debug. When testing BIST it is more important to have the *bist_end* pin available than the *disp_error*, *code_error*, or *dispou*t. Any of these other bits being muxed would have a lesser impact.

- **Code error and parity errors**

For the BIST operation modes, the LFSR will start generating parallel patterns, however, these patterns are not compliant with the 8b/10b encoding. This causes the parity errors *disp_error* and *code_error* to be asserted in the design. Ideally, the LFSR should be updated to generate patterns that have the same number of 0's and 1's to be compliant with this encoded.

- **SerDes always sending data**

This is a big design flaw in the Serialization process, the design is done in such a way that *tx_frame_start* will always go high every 10 clocks, irrespective of if there is data available or not. This causes to have random traffic in the channels which can be interpreted as noise. *Tx_frame_start* should only start toggling when a "comma" has been received and stop entirely upon receipt of the second "comma".

Chapter 6: Results and Discussion

The UVM framework provides several contributions to the validation process. All the improvements are documented in this chapter starting with the KPIs that represent a measurable indicator to adopt the tool during the development of the Pre-Si validation of any VLSI product.

A separate set of advantages was identified apart from the KPIs. These are not strictly tied to a numeric metric but are proven to be helpful during the TB design, implementation, debug, and validation iterations. The inputs for these results are surveys and experiments from validators in the industry.

The following benefits were identified while integrating this tool into the Validation process.

- Speeds up the UVM learning ramping up process.
- Provides guidelines for best-practice coding.
- Reduce human errors, such as typos or wrong interpretations.
- Reduces the UVM TB creation and improves TB flexibility.
- Enables validation execution tasks faster.
- Introduces consistency throughout the different validation levels.

6.1. Metrics and indicators

Measurement is fundamental for improvements. Indicators and metrics show the potential and benefits of the UVM framework. One of the key aspects would be the time invested for generating the same code and the number of lines of code that can be auto-generated instead of hand-coded.

The total number of lines of code that are necessary to implement a verification suite can be an effective measure of the effort required in implementing it. This metric can be used to compare the productivity offered by new verification technologies or methods. If they can reduce the number of lines of code that need to be written, then they should reduce the effort required to implement the verification [22].

6.2. KPI´s

Although there are many benefits with the UVM Framework tool, the Key Performance Indicators (KPI's) that are the most critical are described in the following lines:

- TB infrastructure coding time reduction.
- Number of hand-coded lines reduction.
- Minimization of human errors and typos in the environment.
- Improvement to the TB simulation performance.
- Automatic diagram generation.

6.3. Testbench Template Results Analysis

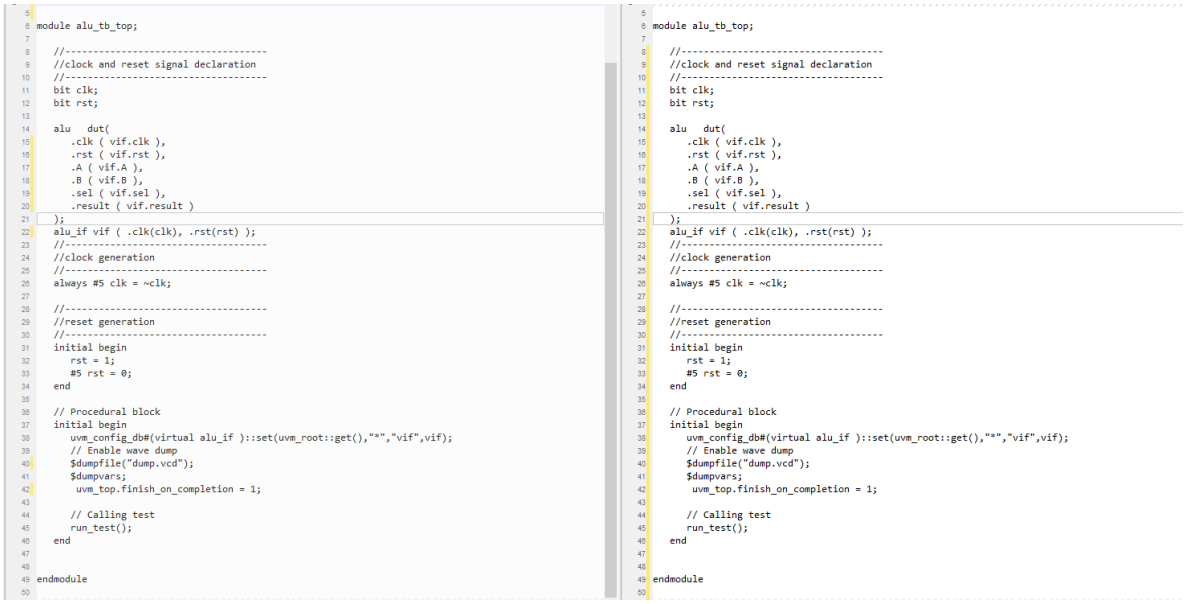
The main advantage of using the proposed platform is the creation of a template that has most of the code used in a complete SV tesbench with UVM. This template is ready to compile and only few gaps are remaining to be completed with the custom validator's code.

6.3.1. ALU Testbenches comparison

In the following lines, a comparison between the output files of the auto-generated testbenches against the actual testbenches used for the validation of the ALU is performed.

6.3.1.1. ALU TB TOP

The generated code for the Alu testbench top compared against the final SV code used in the validation testbench is shown in Figure 178. As can be appreciated in the comparison, the core of the file is 45 lines, and all of them were generated by the tool. So, 100% of the code was automatically generated.

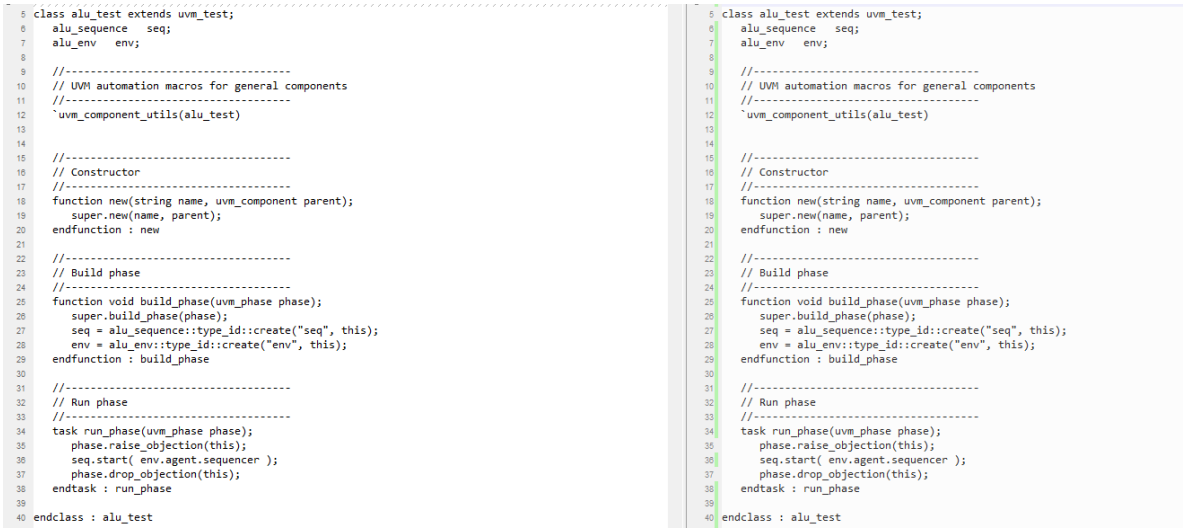


```
module alu_tb_top;

   //-------------------------------------
   //clock and reset signal declaration
   //-------------------------------------
   bit clk;
   bit rst;

   alu   dut(
      .clk ( vif.clk ),
      .rst ( vif.rst ),
      .A ( vif.A ),
      .B ( vif.B ),
      .sel ( vif.sel ),
      .result ( vif.result )
   );
   alu_if vif ( .clk(clk), .rst(rst) );
   //-------------------------------------
   //clock generation
   //-------------------------------------
   always #5 clk = ~clk;

   //-------------------------------------
   //reset generation
   //-------------------------------------
   initial begin
      rst = 1;
      #5 rst = 0;
   end

   // Procedural block
   initial begin
      uvm_config_db#(virtual alu_if )::set(uvm_root::get(),"*","vif",vif);
      // Enable wave dump
      $dumpfile("dump.vcd");
      $dumpvars;
       uvm_top.finish_on_completion = 1;

      // Calling test
      run_test();
   end


endmodule
```

Figure 178 – alu_tb_top.sv: Generated vs Final

6.3.1.2. ALU Test

The alu_test generated code produced 35 lines of code; this is shown in Figure 179. All of them were reused without modification for the final testbench. So again, 100% of the code for this file was auto-generated.



```
class alu_test extends uvm_test;
   alu_sequence   seq;
   alu_env   env;

   //-------------------------------------
   // UVM automation macros for general components
   //-------------------------------------
   `uvm_component_utils(alu_test)


   //-------------------------------------
   // Constructor
   //-------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //-------------------------------------
   // Build phase
   //-------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      seq = alu_sequence::type_id::create("seq", this);
      env = alu_env::type_id::create("env", this);
   endfunction : build_phase

   //-------------------------------------
   // Run phase
   //-------------------------------------
   task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      seq.start( env.agent.sequencer );
      phase.drop_objection(this);
   endtask : run_phase

endclass : alu_test
```

Figure 179 – alu_test.sv: Generated vs Final

6.3.1.3. ALU Sequencer

The sequencer is arguably the simplest UVC used in the UVM testbenches. For the ALU, 15 lines of code were sufficient to complete the code, and the framework generated 100% of them, as is shown in Figure 180.

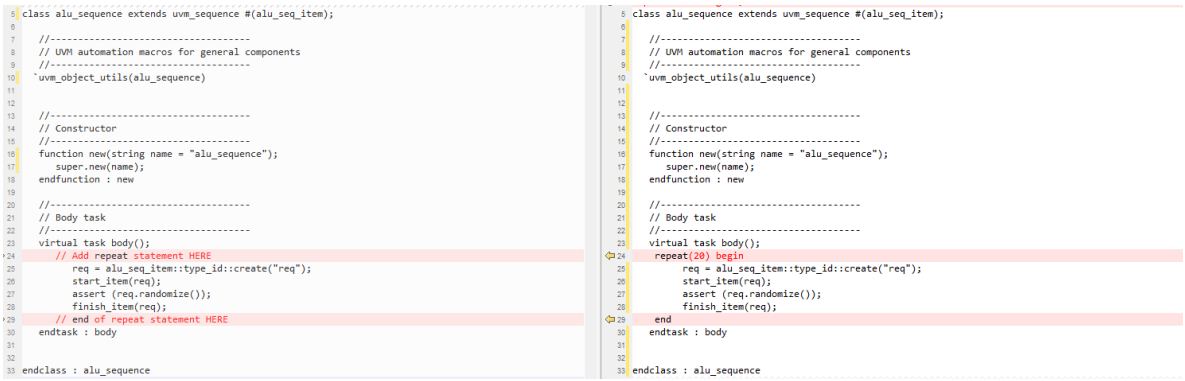
```
class alu_sequencer extends uvm_sequencer#(alu_seq_item);

  //-------------------------------------
  // UVM automation macros for general components
  //-------------------------------------
  `uvm_component_utils(alu_sequencer)

  //---------------------------------------
  //constructor
  //---------------------------------------
  function new(string name, uvm_component parent);
    super.new(name,parent);
  endfunction

endclass
```

```
class alu_sequencer extends uvm_sequencer#(alu_seq_item);

  //-------------------------------------
  // UVM automation macros for general components
  //-------------------------------------
  `uvm_component_utils(alu_sequencer)

  //---------------------------------------
  //constructor
  //---------------------------------------
  function new(string name, uvm_component parent);
    super.new(name,parent);
  endfunction

endclass
```

Figure 180 - alu_sequencer.sv: Generated vs Final

6.3.1.4. ALU Sequence

For the ALU sequence, 28 lines were generated. For which, the code will work completely without modifications as is, but being strict 2 lines were modified to adjust to the number of sequence items that are going to be sent. In this case, 92.85% remained the same as the output file from the tool. In this case, 20 sequence items are sent so the repeat code was modified to do so. So dependent on each user, the final code will be between 90% to 100% auto-generated.



```
class alu_sequence extends uvm_sequence #(alu_seq_item);

  //-------------------------------------
  // UVM automation macros for general components
  //-------------------------------------
  `uvm_object_utils(alu_sequence)


  //-------------------------------------
  // Constructor
  //-------------------------------------
  function new(string name = "alu_sequence");
     super.new(name);
  endfunction : new

  //-------------------------------------
  // Body task
  //-------------------------------------
  virtual task body();
     // Add repeat statement HERE
        req = alu_seq_item::type_id::create("req");
        start_item(req);
        assert (req.randomize());
        finish_item(req);
     // end of repeat statement HERE
  endtask : body


endclass : alu_sequence
```

```
class alu_sequence extends uvm_sequence #(alu_seq_item);

  //-------------------------------------
  // UVM automation macros for general components
  //-------------------------------------
  `uvm_object_utils(alu_sequence)


  //-------------------------------------
  // Constructor
  //-------------------------------------
  function new(string name = "alu_sequence");
     super.new(name);
  endfunction : new

  //-------------------------------------
  // Body task
  //-------------------------------------
  virtual task body();
   repeat(20) begin
        req = alu_seq_item::type_id::create("req");
        start_item(req);
        assert (req.randomize());
        finish_item(req);
   end
  endtask : body


endclass : alu_sequence
```

Figure 181 - alu_sequence.sv: Generated vs Final

6.3.1.5. ALU Sequence item

For the sequence item, the mandatory code was generated entirely. It is up to each validator if they want to add or not any constraints. This code, shown in Figure 182, is completely optional. In the ALU few constraints were added as an example of what can be done. For the ALU, 24 lines of code out of 27 in total were generated. These are optional so for this validation between 88.88% to 100% of the code completion was achieved.

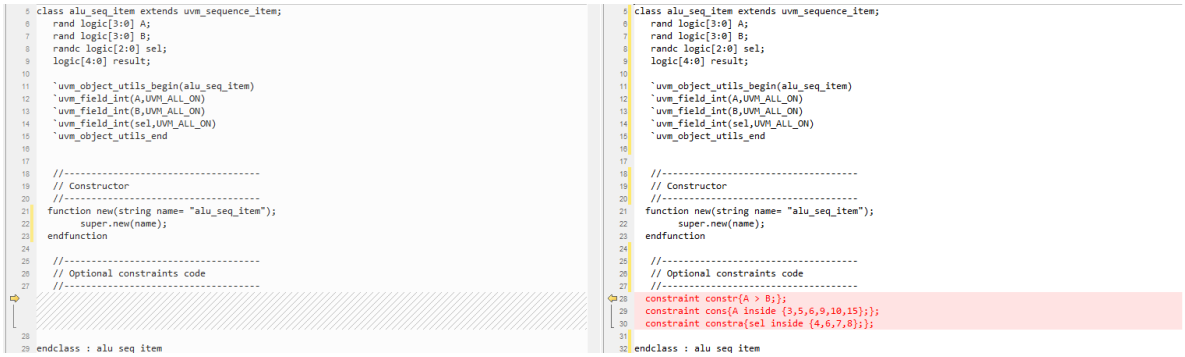


Figure 182 - alu_sequence_item.sv: Generated vs Final

6.3.1.6. ALU Scoreboard

The scoreboard can be as complex as the engineer want to code it. This code is the most project by project dependent and needs to be coded specifically per-project basis. For the ALU´s scoreboard, the total of lines coded was 114, from which only 38 were generated by the tool. In this case, 33.33% of the total lines were auto-generated.

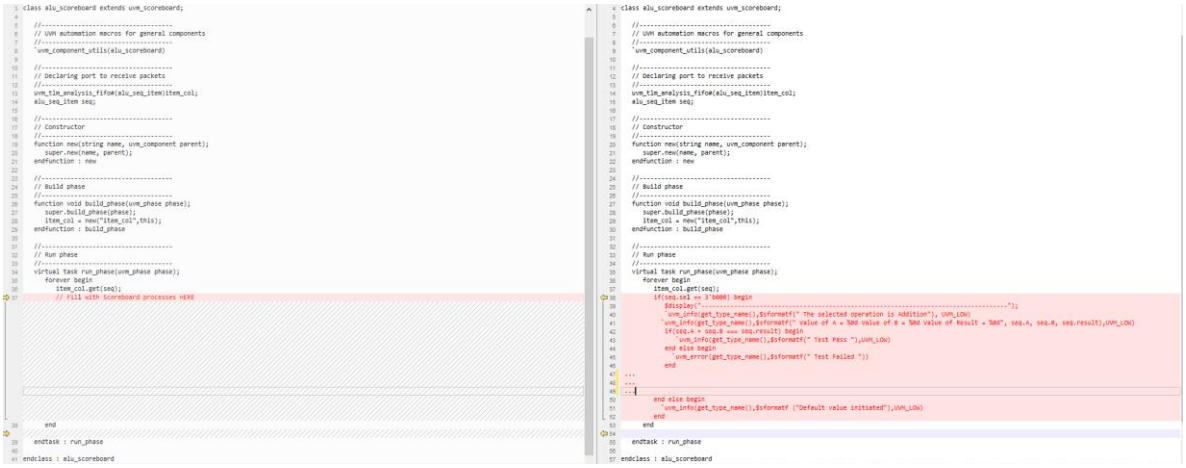


Figure 183 - alu_scoreboard.sv: Generated vs Final

6.3.1.7. ALU Monitor

Most of the Monitor's code was generated by the tool. For the ALU, 41 out of 52 lines coded were auto-generated. This indicates 78% of tool-generated code.

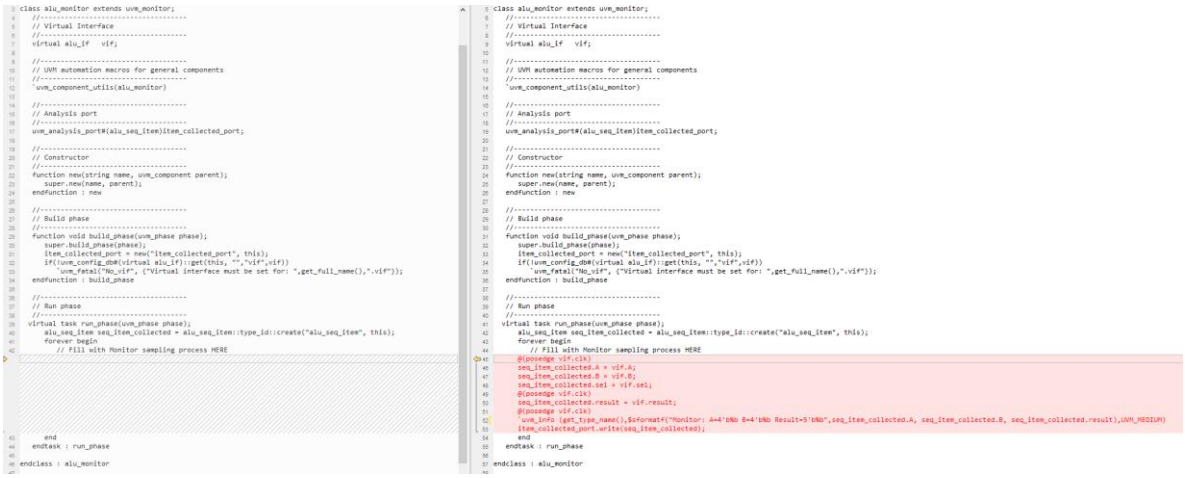


Figure 184 - alu_monitor.sv: Generated vs Final

6.3.1.8. ALU Env

The Alu Env was completely generated by the tool. This makes 100% of code generation.

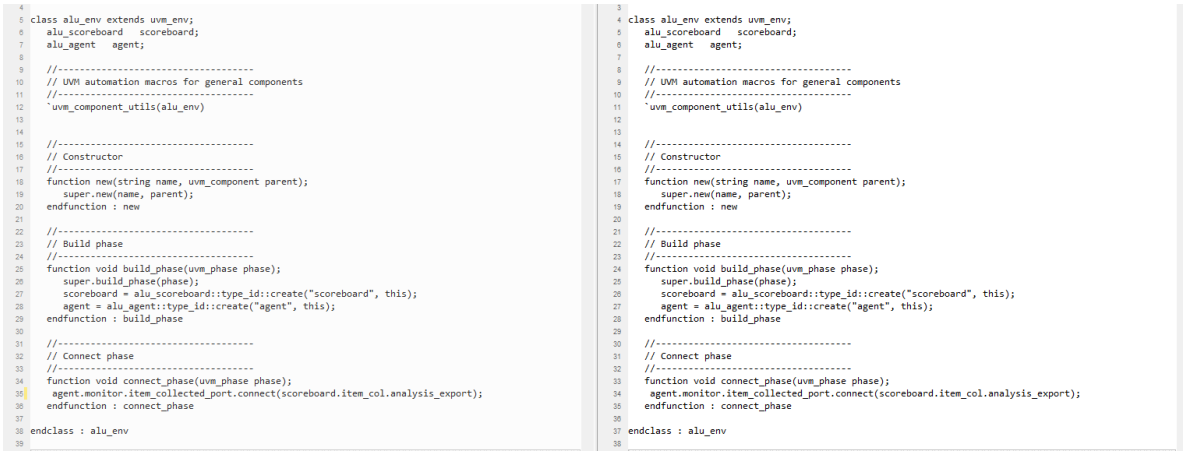


Figure 185 - alu_env.sv: Generated vs Final

6.3.1.9. ALU Driver

Out of 53 lines of code for the ALU Driver, 43 were generated automatically. This consists of 81% of the code.

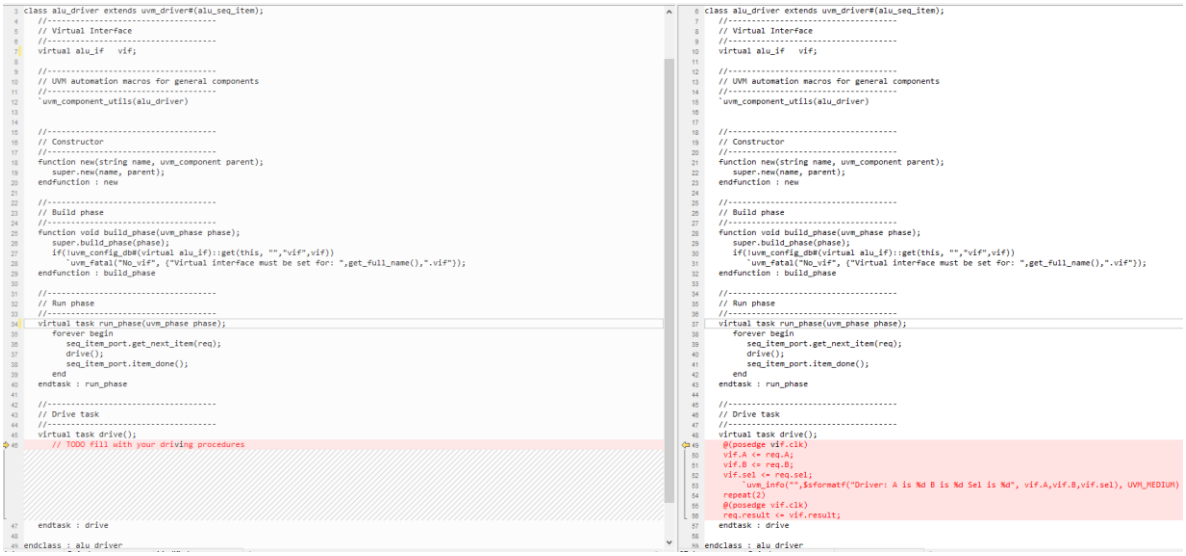


Figure 186 - alu_drive.sv: Generated vs Final

6.3.1.10. ALU Interface

The Alu interface was also generated entirely by the tool. This makes 100% of code generation.

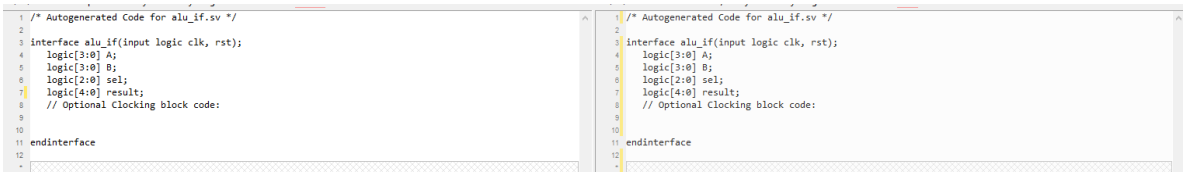


Figure 187 - alu_if.sv: Generated vs Final

6.3.1.11. ALU Subscriber

For the subscriber used in the ALU, only around 50% of the code was reused, as every coverpoint is project-specific. The comparison is available in Figure 188.

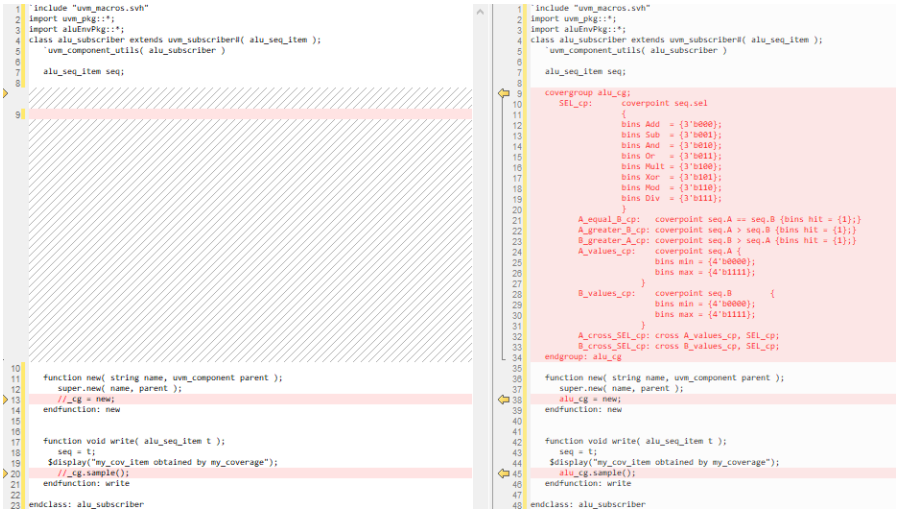


Figure 188 - alu_subscriber.sv: Generated vs Final

6.3.1.12. ALU Agent

For the ALU Agent, the 100% lines of code were produced by the UVM Framework software.

Figure 189 - alu_agent.sv - Generated vs Final

6.3.2. SerDes Testbenches comparison

Next, the output files of the auto-generated testbenches are compared against the actual testbenches used for the validation of the SerDes.

6.3.2.1. SerDes TB TOP

The generated code for the SerDes TB TOP compared against the final SV code used in the validation testbench is shown in Figure 190. This class is composed of 58 lines, with only a difference of lines between the code that was autogenerated. These lines are specific to the SerDes project as they are internal signals of the SerDes that were added to the interface. In this case, 91.37% of the code was generated by the tool

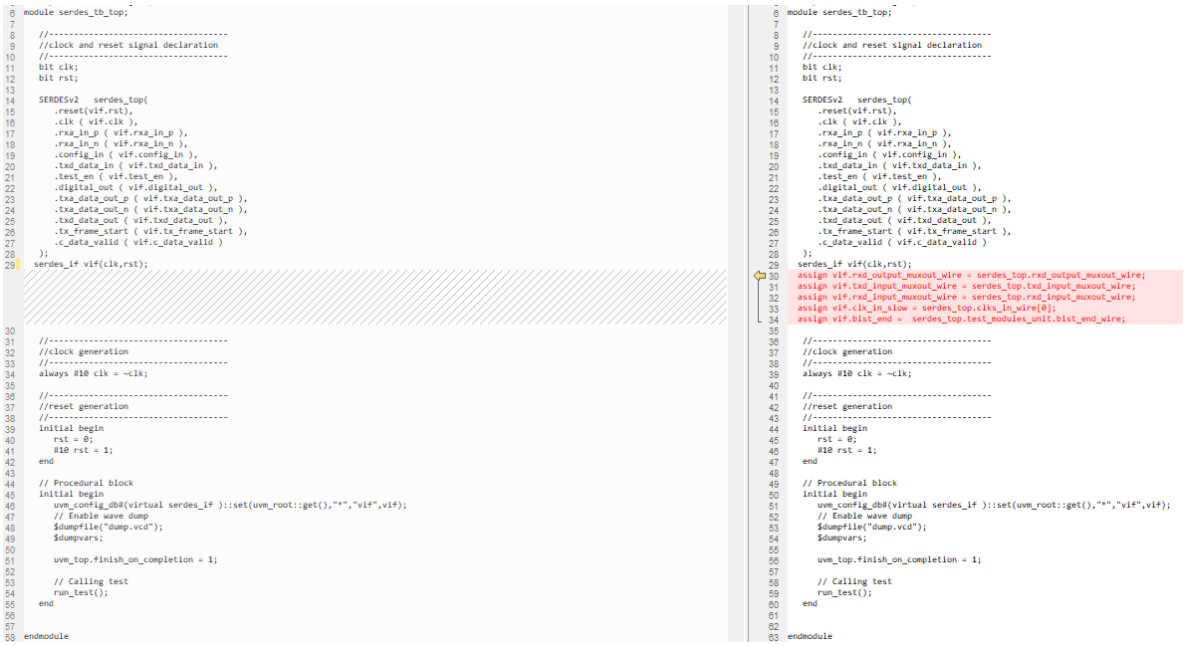


Figure 190 - serdes_tb_top.sv: Generated vs Final

6.3.2.2. SerDes Base Tests

The SerDes Base Test code used in the validation environment is a bit more different than the one generated due to the addition of more custom configuration objects. Also, the base test used in the validation had some additional information printed in the end_of_elaboration phase and the report_phase. This addition is completely optional. And finally, the run_phase was moved into the child tests, so there is no need to have that in the base test. Approximately, 50% of the code in this class was generated by the tool. This comparison can be found in Figure 191.

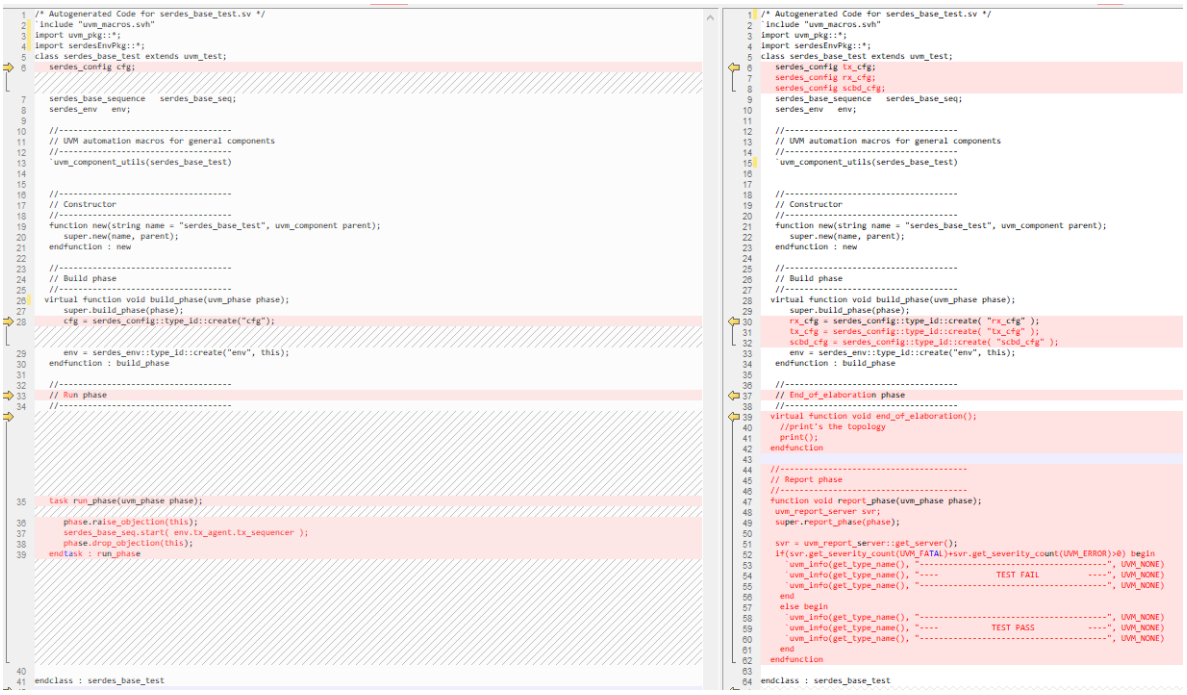


Figure 191 - serdes_base_test.sv: Generated vs Final

6.3.2.3. SerDes Focus Tests

Most of the code for the seven focus tests was generated by the tool, however, the specifics of each mode and setting the configuration objects in the database had to be coded manually. Approximately 70 % of the code was generated automatically for the focused tests of the SerDes. This comparison of the open BIST test is available in Figure 192, the rest of the tests are very similar to this.

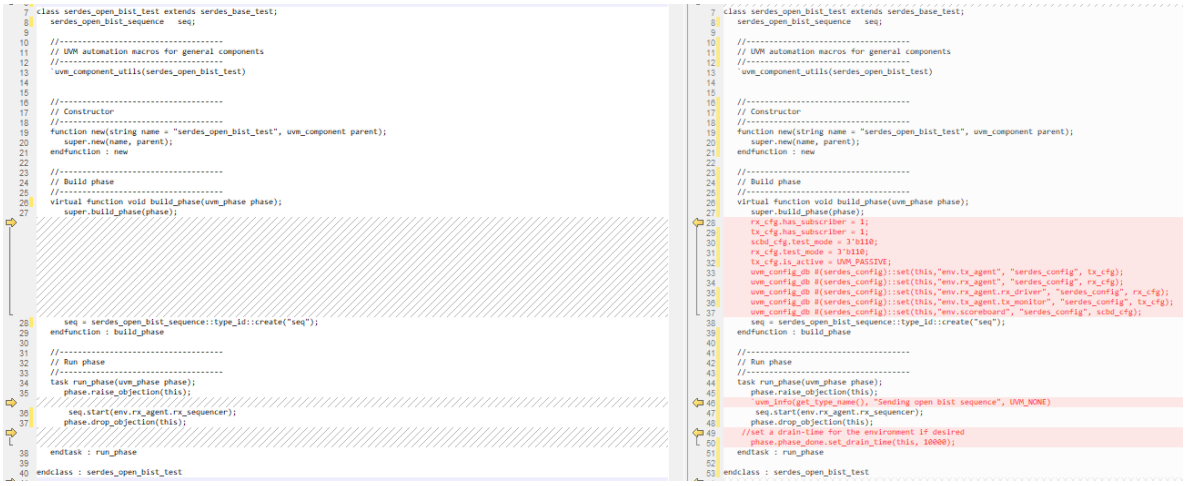


Figure 192 - serdes_*_test.sv: Generated vs Final

6.3.2.4. SerDes Monitors

Figure 193 shows the differences between the RX monitor used in the validation against the code that was generated by the tool. In this case, 65% of the generated code was reused, although most of the new codes are `uvm_info print statements. This is similar to the other monitor TX Monitor.

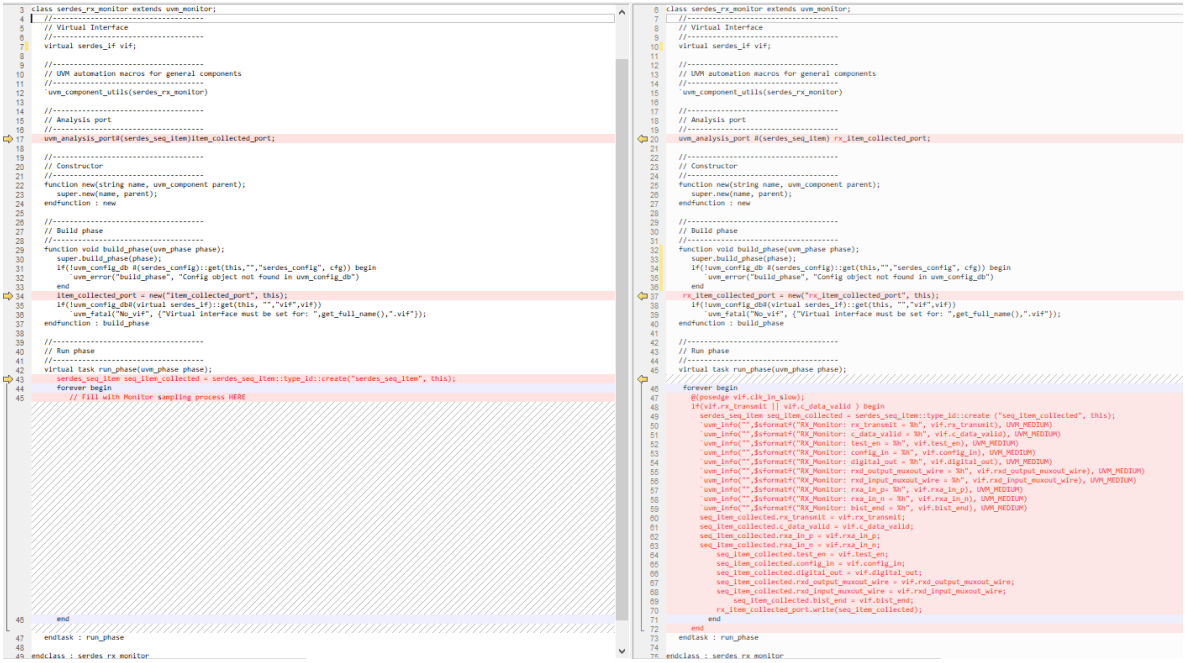


Figure 193 - serdes_rx_monitor.sv: Generated vs Final

6.3.2.5. SerDes Env

As is shown in Figure 194,100% of the code was achieved by using the tool.

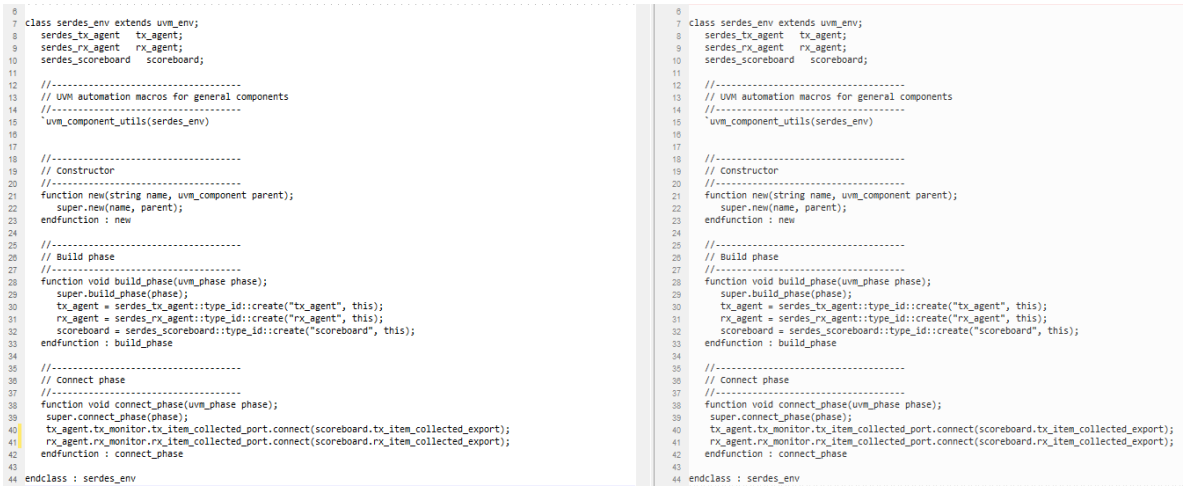


Figure 194 - serdes_env.sv: Generated vs Final

6.3.2.6. SerDes Drivers

Approximately 70% of the code was able to be reused for the TX Driver and the RX Driver. This comparison can be seen in Figure 195.

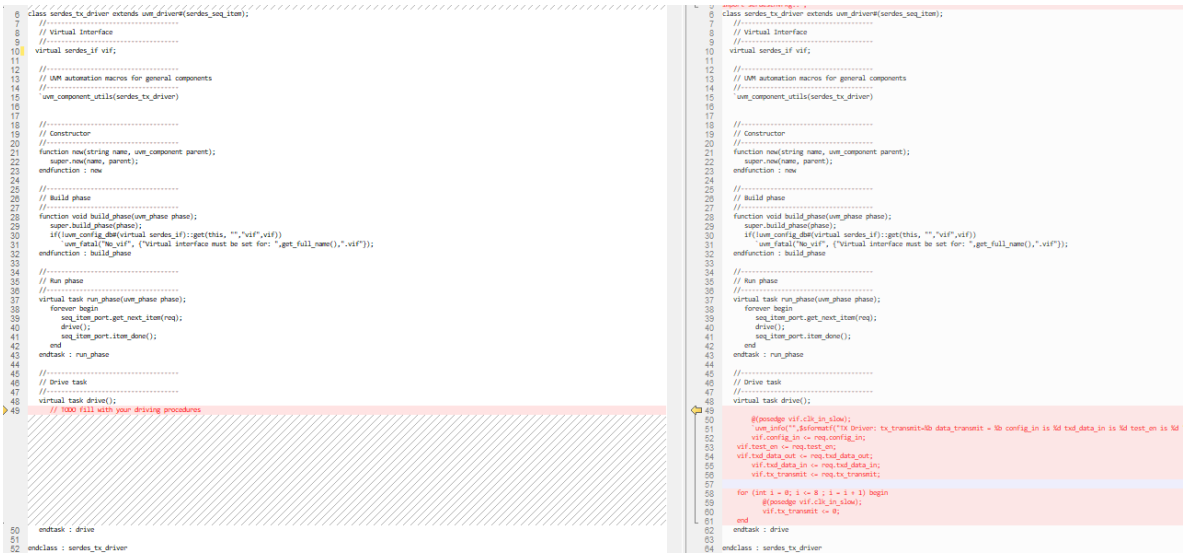


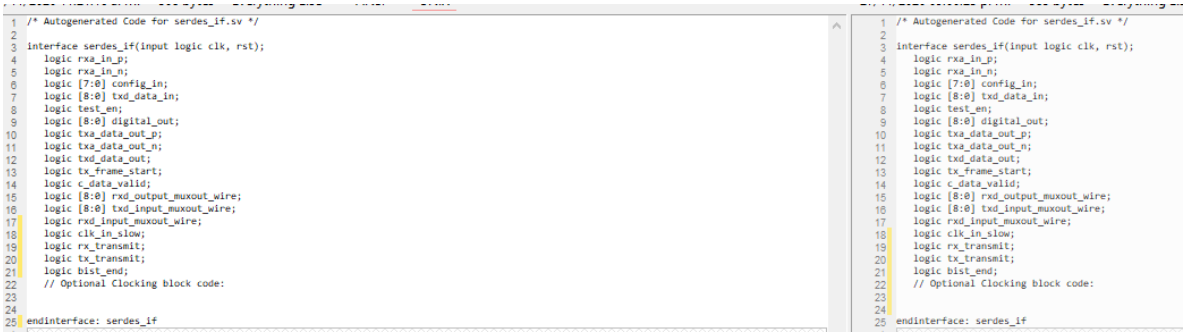
Figure 195 - serdes_tx_driver.sv: Generated vs Final

6.3.2.7. SerDes Config

For the configuration object, the re-used code was near 90%. Just the fields that are unique to each project had to be added to this code.

6.3.2.8. SerDes Interface

For the SerDes interface, 100% of the code was obtained by the tool. Although, keep in mind that every signal in the interface had to be added in the tool by hand. The comparison is displayed in Figure 196.



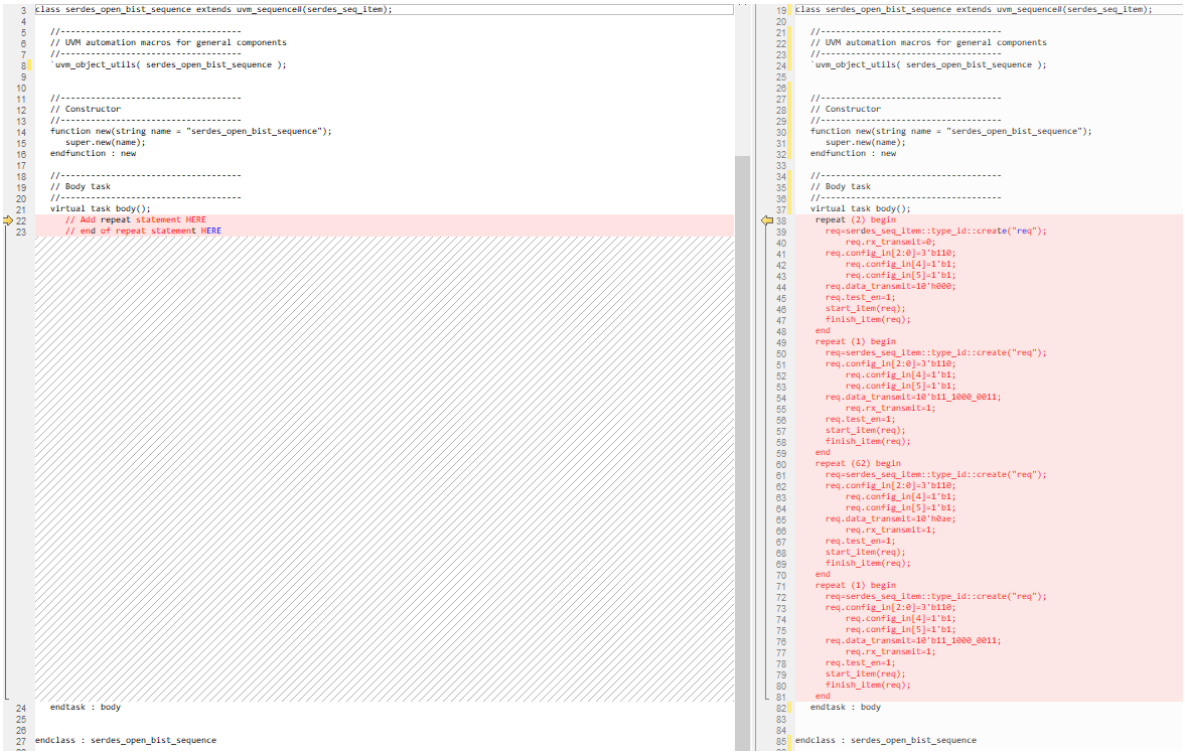
```
/* Autogenerated Code for serdes_if.sv */

interface serdes_if(input logic clk, rst);
   logic rxa_in_p;
   logic rxa_in_n;
   logic [7:0] config_in;
   logic [8:0] txd_data_in;
   logic test_en;
   logic [8:0] digital_out;
   logic txa_data_out_p;
   logic txa_data_out_n;
   logic txd_data_out;
   logic tx_frame_start;
   logic c_data_valid;
   logic [8:0] rxd_output_muxout_wire;
   logic [8:0] txd_input_muxout_wire;
   logic rxd_input_muxout_wire;
   logic clk_in_slow;
   logic rx_transmit;
   logic tx_transmit;
   logic bist_end;
   // Optional Clocking block code:


endinterface: serdes_if
```

```
/* Autogenerated Code for serdes_if.sv */

interface serdes_if(input logic clk, rst);
   logic rxa_in_p;
   logic rxa_in_n;
   logic [7:0] config_in;
   logic [8:0] txd_data_in;
   logic test_en;
   logic [8:0] digital_out;
   logic txa_data_out_p;
   logic txa_data_out_n;
   logic txd_data_out;
   logic tx_frame_start;
   logic c_data_valid;
   logic [8:0] rxd_output_muxout_wire;
   logic [8:0] txd_input_muxout_wire;
   logic rxd_input_muxout_wire;
   logic clk_in_slow;
   logic rx_transmit;
   logic tx_transmit;
   logic bist_end;
   // Optional Clocking block code:


endinterface: serdes_if
```

Figure 196 - serdes_if.sv: Generated vs Final

6.3.2.9. SerDes Sequences

For the sequences, the percent of reused code is variable. On average, around 60% of the final code was generated by the tool. Figure 197 presents a comparison of the open BIST sequence.



```
class serdes_open_bist_sequence extends uvm_sequence#(serdes_seq_item);

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_object_utils( serdes_open_bist_sequence );


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name = "serdes_open_bist_sequence");
      super.new(name);
   endfunction : new

   //------------------------------------
   // Body task
   //------------------------------------
   virtual task body();
      // Add repeat statement HERE
      // end of repeat statement HERE
   endtask : body


endclass : serdes_open_bist_sequence
```

```
class serdes_open_bist_sequence extends uvm_sequence#(serdes_seq_item);

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_object_utils( serdes_open_bist_sequence );


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name = "serdes_open_bist_sequence");
      super.new(name);
   endfunction : new

   //------------------------------------
   // Body task
   //------------------------------------
   virtual task body();
    repeat (2) begin
      req=serdes_seq_item::type_id::create("req");
         req.rx_transmit=0;
      req.config_in[2:0]=3'b110;
         req.config_in[4]=1'b1;
         req.config_in[5]=1'b1;
      req.data_transmit=10'h000;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b110;
         req.config_in[4]=1'b1;
         req.config_in[5]=1'b1;
      req.data_transmit=10'b11_1000_0011;
         req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (62) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b110;
         req.config_in[4]=1'b1;
         req.config_in[5]=1'b1;
      req.data_transmit=10'h0ae;
         req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b110;
         req.config_in[4]=1'b1;
         req.config_in[5]=1'b1;
      req.data_transmit=10'b11_1000_0011;
         req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
   endtask : body


endclass : serdes_open_bist_sequence
```

Figure 197 - serdes_*_sequence.sv: Generated vs Final

6.3.2.10. SerDes Sequence Item

The SerDes sequence item can be generated in almost it is totality by the code generator. The only thing that must be added manually is the constraints section, which is specific per-project basis. Around 98% matches with the code from the code generator, Figure 198 shows this comparison.

Figure 198 - serdes_seq_item.sv: Generated vs Final

6.3.2.11. SerDes Scoreboard

Figure 199 presents only a fragment of the comparison of the former scoreboard against the final one. In this case, the scoreboard was restructured with several changes after the obtained code. This is due to the complexity of the design. For this module, less than 10% of the code stayed the same.

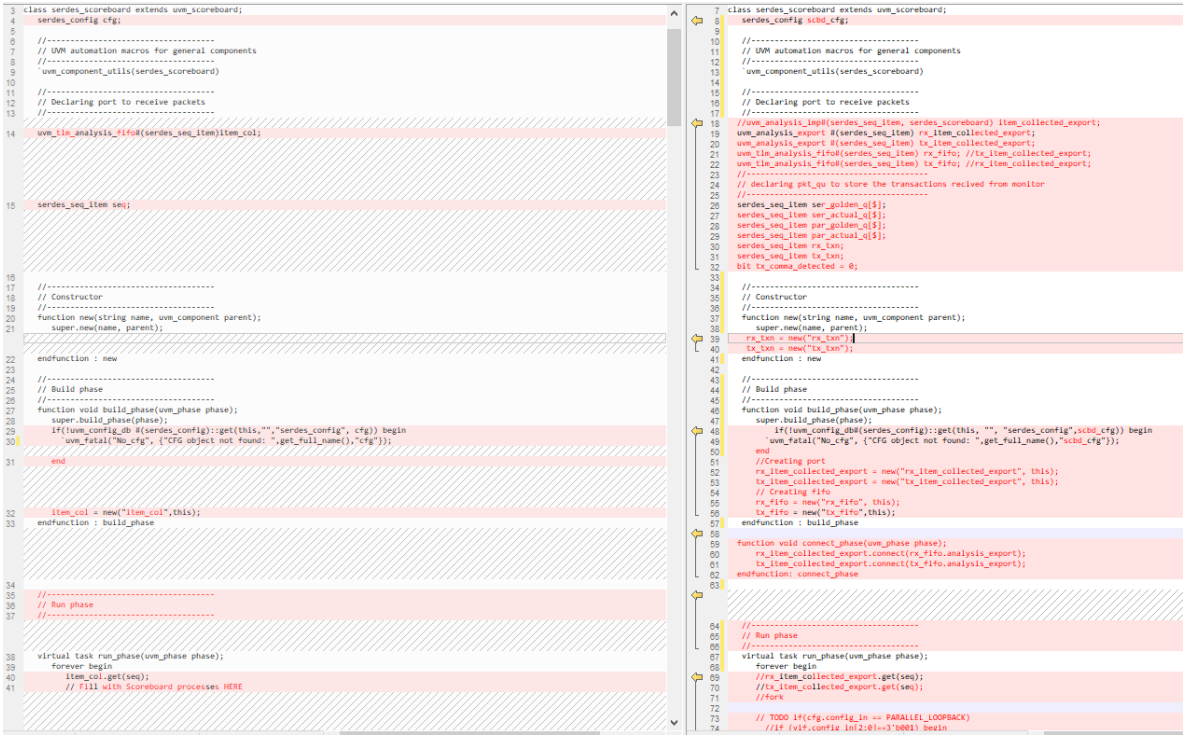


Figure 199 - serdes_scoreboard.sv: Generated vs Final

6.3.2.12. SerDes Subscriber

For both RX and TX subscribers the auto-generated code average is about 50%. This is because the covergroup definition cannot be added automatically within the tool. The RX Subscriber comparison can be found in Figure 200.

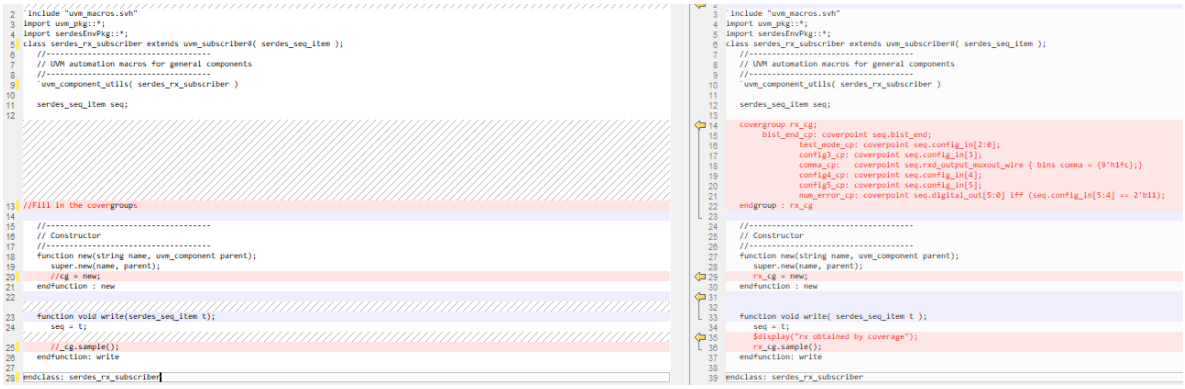


Figure 200- serdes_subscriber.sv: Generated vs Final

6.3.2.13. SerDes Sequencers

Figure 201 presents the comparison of the auto-generated code against the one used in the validation of the SerDes. For both sequencers, it was completely reused 100%.

Figure 201 - serdes_tx_sequencer.sv: Generated vs Final

6.3.2.14. SerDes Agents

For the SerDes Agents, around 70% of the code created by the tool was used as and the rest was added for the validation. This comparison is displayed in Figure 202.

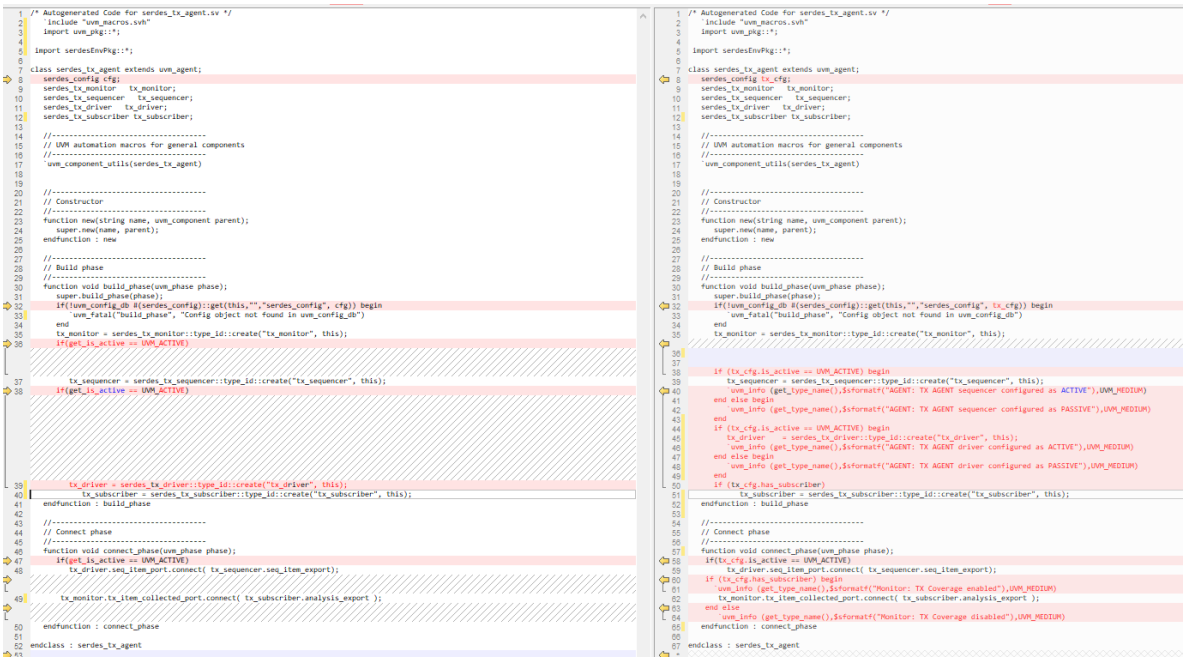


Figure 202 - serdes_tx_agent.sv: Generated vs Final

6.3.3. Lines of Code Generation Summary

Next, Table 52 shows the summary of the overall benefit that can be obtained by using the tool to create the Testbench. This measurement is achieved by comparing the number of lines generated vs the lines that were hand-coded.

Component	Simple Design (ALU)	Complex Design (SerDes)	Average
TB TOP	100%	90%	95%
Interface	100%	100%	100%
UVM Base Test	N/A	50%	50%
UVM Test	100%	70%	85%
UVM Sequencer	100%	100%	100%
UVM Sequence	90%	60%	75%
UVM Sequence Item	85%	95%	90%
UVM Scoreboard	33%	10%	20%
UVM Driver	80%	70%	75%
UVM Monitor	75%	65%	70%
UVM Agent	100%	70	85%
UVM Cfg	N/A	90%	90%
UVM Env	100%	100%	100%
UVM Subscriber	50%	50%	50%
Total	84%	73%	78%

Table 52 - Lines of code Summary

In this analysis, the percentages were rounded down to showcase that even in a worst-case

scenario the results are very appealing. Most of the components require minor to non-existing changes to the output files of the tool. This will allow the user to focus on coding the gaps of the components that require project-specific customization such as subscribers, sequences, or scoreboards.

6.4. Efficiency

The usage of the framework provides numerous implicit advantages to improve the Validation Team's efficiency.

This section consists of a survey based on observations made from the authors, but also from early adopters of the tool that were asked to provide feedback about the program.

For this analysis, a group of validation engineers was selected as the early adopters. They were chosen based on their grade of expertise in the verification field. The sample is not representative, collaborators were mainly chosen from Mexico and the United States. The participants were classified into three sub-groups:

- Beginner user profile: Junior validator or recently graduated. Typically, with less than a year of experience working in the validation industry with zero knowledge of UVM methodologies.
- Intermediate user profile: Intermediate validator with 2-3 years of experience in the validation field but with basic knowledge of the UVM methodology.
- Advanced user profile: Senior validator with more than 5 years in the industry with a year or more working on the UVM based TB development.

The following feedback was collected by the survey. Additional advantages were identified, some of them were not even part of the proposed KPIs.

Beginner user's observations:

- Improves the learning process by reducing the number of total objects and components available in the UVM library to only a subset of the basic elements required to set up the base TB infrastructure.
- Visualization of the TB is easier as it is constructed in a graphical canvas as a representation of the UVM database and TB hierarchies.

- Part of the code documentation is automatically generated.

Intermediate user's observations:

- Improves scalability of the project due to its modular approach and container elements creation and properties menu.
- Improves portability of the code with the creation of the uvmfr file types and the copy and paste features of the elements.
- Adds flexibility to a TB infrastructure because it is easy to modify the full architecture with a few clicks.
- Connectivity changes are easy to perform and integrate at any level of the UVM hierarchy.
- Interface and sequence item signals can be modified maintaining consistency throughout the design.

Advanced user's observations:

- Naming conventions are easy to maintain and update for all the VCs in the canvas.
- Inheritance and coverage concepts are integrated into a graphical tool.
- Reduces the time spent on the ramping up process of new team members.
- Complex UVM concepts and knowledge transfer is simplified by the usage of the tool.
- Version control can be added to the file headers easily.
- Coding guidelines and best-known methods for UVM methodology are implemented smoothly.

A table comparing the time taken to write of the testbench, considering just the template's code, without exhaustive project specific details a per the UVM skill of the user can be found in Table 53.

User	Simple Design (ALU)	Complex Design (SerDes)
Beginner	10 – 12 hours	50 – 60 hours
Intermediate	6 – 8 hours	30 – 40 hours
Advanced	2 – 4 hours	8 – 12 hours
Average	~7 hours	~ 33 hours

Table 53 – Template Developing Time by hand as per UVM Skill

6.5. Developing time

Depending on how prominent the user with the tool is; the amount of time used to develop the testbench template can be drastically improved. Considering a newly adapt to the tool (novice), someone that has some understanding of how to use the tool (medium), and someone that completely understand how to create testbenches using the tool (expert), the Table 54 shows how much time it takes to complete the testbench from scratch up to the step of generating the templates in SV.

User	Simple Design (ALU)	Complex Design (SerDes)
Novice	~ 45 – 60 min	~ 90 – 120 min
Medium	~ 15 – 30 min	~ 45 – 60 min
Expert	~ 6 – 7 min	~ 20 – 30 min
Average	27 min	60 min

Table 54 - Template Developing time with the framework as per tool proficiency

The benefit of using the tool shows up when compared against hand coding the UVM testbench template. Table 55 showcases how much times faster can be achieved in average by using the tool against hand coding the template.

		Simple Design (ALU)			Complex Design (SerDes)		
	USER	UVM Skills					
		Beginner	Intermediate	Advanced	Beginner	Intermediate	Advanced
Tool Usage	Novice	12x	8x	3x	30x	20x	6x
	Medium	29x	18x	8x	63x	40x	11x
	Expert	101x	64x	27x	132x	84x	24x

Table 55 - ALU template coding comparison

From this table, it is observed that there is more gain when the user is prominent with the tool but his skill in UVM is limited. If the user has more skill in UVM, the tool benefit diminishes, however, the typical user is most likely to be at the same level of expertise in UVM and the tool usage. This is highlighted in the green diagonal, showcasing that the tool gives several improvements as the design increases in complexity.

In all cases, using the tool to generate the UVM template of the testbench represents a significant time reduction that implies an efficiency improvement.

Chapter 7: Conclusions

A novel tool to design and implement Pre-Silicon validation testbenches using the UVM methodology was presented in this document. Discussion about the existing validation methodologies and tools to improve the auto-generation of the TB was analyzed and compared with the proposed hypothesis.

Once the specifications and requirements were defined, a description of the graphical software development was explained in detail. The relevant libraries were mapped to their functionality in the graphic scene. Code generator algorithm was defined and exposed as the final step of a typical usage case.

Two separate DUTs were taken as a case of study and their testbenches were generated using the tool. A step by step solution was illustrated to showcase the functional capabilities of the platform for different levels of user expertise with both, the tool and the UVM methodology.

Using the files generated by the tool, the rest of the TB code was completed manually to obtain a reference model of a full validation environment. Basic validation was done to demonstrate the tool output and its impact on the verification process.

Finally, a comparison between the traditional way to create a TB infrastructure and the usage of the tool as the TB template generator was presented, showing an average percentage of 78% of code generated automatically, reducing human errors and facilitating the TB modeling requirements.

7.1. Contributions

The major contributions of this project were as below:

- A graphical tool for the automated creation of UVM testbenches.
- A UVM based layered testbench designed to verify the functionality of simple and complex IPs using the UVM Framework tool.

7.2. Social impact

Once launched, the UVM Framework could be a great asset in the learning of UVM methodologies for university students, validation engineers, RCGs (Recent College Graduates), and any UVM enthusiasts.

The creation of a solid validation team is essential for any Pre-Si verification organization. This tool contributes to a quick integration of new team members and facilitates knowledge transfer. This is a critical outcome during this time since new silicon designs and their validation requirements to speed up their processes due to the increase of the cloud services, remote-working, teleconferencing, and new requirements that appeared with the covid-19 pandemic.

7.3. Future work

All the key features of the project were covered in this case study. However, a set of additional capabilities that were not defined in the original scope turned to be interesting areas for exploration, such as an exhaustive SW validation for the tool performed by a complete team of validation engineers (the typical paradox, who validates the validation tool?).

Coverage capabilities can be improved by adding a dedicated element for this purpose. Port types in UVM can be explored deeper to add flexibility to the tool, however, this addition may obstacle the didactic purposes of the tool so, a trade-off would be required to include this in a release depending on the user's goals. Portability and compatibility with similar apps seem to be a fascinating topic that can be explored in partnership with other engineering teams in the industry.

A lot more customization options can be listed in this section, but for this release, the typical and more relevant were selected to include a wide variety of users with different levels of knowledge, experience, and curiosity.

References

[1] M. M. Hoil-Loria, “Design for Testability in a SerDes System,” Nov. 2017, Accessed: Nov. 02, 2020. [Online]. Available: https://rei.iteso.mx/handle/11117/5137.

[2] C. F. Limones-Mora, “Test Modules Design for a SerDes Chip in 130 nm CMOS technology,” Jul. 2016, Accessed: Nov. 02, 2020. [Online]. Available: https://rei.iteso.mx/handle/11117/3892.

[3] R. Rivas-Villegas, “Design, Implementation and Verification of a Deserializer Module for a SerDes Mixed Signal System on Chip in 130 nm CMOS Technology,” Aug. 2016, Accessed: Nov. 02, 2020. [Online]. Available: https://rei.iteso.mx/handle/11117/3990.

[4] W. Ni and J. Zhang, “Research of reusability based on UVM verification,” in *2015 IEEE 11th International Conference on ASIC (ASICON)*, Nov. 2015, pp. 1–4, doi: 10.1109/ASICON.2015.7517189.

[5] M. S. M. Dass, “Design and Verification of a Dual Port RAM Using UVM Methodology,” p. 161.

[6] C. Spear and G. Tumbush, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*, 3rd ed. Springer US, 2012.

[7] A. Ziv, “Challenges and Solutions in Post-Silicon Validation of High-end Processors (Invited Tutorial),” in *2019 Formal Methods in Computer Aided Design (FMCAD)*, Oct. 2019, pp. 1–1, doi: 10.23919/FMCAD.2019.8894258.

[8] C. E. Cummings and H. Chambers, “SystemVerilog Virtual Classes, Methods, Interfaces and Their Use in Verification and UVM,” p. 27, 2018.

[9] H. Height, *A Practical Guide to Adopting the Universal Verification Methodology (UVM) Second Edition*. Lulu.com, 2010.

[10] Accelera, *UVM Class Reference Manual 1.2*. 2014.

[11] “Base Classes,” *ChipVerify*. https://www.chipverify.com/uvm/base-classes (accessed Nov. 09, 2020).

[12] J. Francesconi, J. Rodriguez, and P. Julian, “UVM based testbench architecture for unit verification,” Jul. 2014, pp. 89–94, doi: 10.1109/EAMTA.2014.6906085.

[13] Accelera, “Universal Verification Methodology (UVM) 1.2 User’s Guide.” Oct. 08, 2015.

[14] “UVM Tutorial for Beginners.” https://www.chipverify.com/uvm/uvm-tutorial (accessed Nov. 08, 2020).

[15] “UVM Introduction - Verification Guide.” https://verificationguide.com/uvm/ (accessed Nov. 08, 2020).

[16] “UVM Phasing | Universal Verification Methodology.” https://www.learnuvmverification.com/index.php/2016/04/29/uvm-phasing/ (accessed Nov. 09, 2020).

[17] K. Salah, “A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities,” pp. 94–99, Feb. 2015, doi: 10.1109/IDT.2014.7038594.

[18] W. Team, “WiRed Panda - Learn about logic circuits and simulate them in an easy and friendly way.” https://wiredpanda.org/ (accessed Oct. 08, 2020).

[19] “Qt (software),” *Wikipedia*. Nov. 19, 2020, Accessed: Nov. 20, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Qt_(software)&oldid=989500203.

[20] “Doulos.” https://www.doulos.com/knowhow/systemverilog/uvm/easier-uvm/ (accessed Nov. 01, 2020).

[21] M. Horn, H. van der Shoot, M. Peryer, T. Fitzpatrick, and J. Stickley, “Universal Verification Methodology (UVM) Cookbook,” *2018*, p. 547.

[22] J. Bergeron, *Writing Testbenches using SystemVerilog*. Springer US, 2006.

Appendix

9.1. SerDes RTL Codes

The following RTL codes are listed in alphabetical order.

9.1.1. Analog_RX.sv

```
/*****************************************************************************
*Name:
* Analog_RX.v
*Description:
* This block is a behavioural model for the analog
* front end of the Deserializer. The analog front end transforms
* a weak differential signal into a CMOS single ended signal.
*Editor:
* Miguel Mihail Hoil Loria.
*Changes:
* Ports named after the LEF file ports and declared as a black box.
*****************************************************************************/
module Analog_RX
  (
    //inout \subc!,
    //------------Inputs--------
    input rx_in,
    input rx_inb,
    //------------Outputs--------
    output rx_out,
    output rx_outb
  );
  //Uncomment for tests with verilog
  assign rx_out = rx_in;
  assign rx_outb = rx_inb;
endmodule
```

9.1.2. Analog_TX.sv

```
/*****************************************************************************
*Name:
* Analog_TX.v
*Description:
* This block is a wrapper for the analog transmitter.
*Editor:
* Miguel Mihail Hoil Loria.
*Changes:
* Instances the analog transmitter black box.
*****************************************************************************/
module Analog_TX
(
//------------Inputs--------
input txa_data_in,
input [7 : 0] tx_config,
//------------Outputs--------
output txa_data_out_p,
output txa_data_out_n
);
/*TXA_FINAL
txa_final_unit (
 .\sub! (1'b0) ,
//------------Inputs--------
 .DATA(txa_data_in),
```

```
 .ZA(tx_config[0]),
 .ZB(tx_config[1]),
 .ZC(tx_config[2]),
 .ZD(tx_config[3]),
 .AMP_CTRL_1(tx_config[4]),
 .AMP_CTRL_2(tx_config[5]),
 .PRE_CTRL_1(tx_config[6]),
 .PRE_CTRL_2(tx_config[7]),
 .TEST_DATA(1'b0),
 .DATA_SELECTOR(1'b0),
//------------Outputs--------
 .TX(txa_data_out_p),
 .TXBar(txa_data_out_n),
 .DIRECT_DATA_TX(),
 .DIRECT_DATA_TXBar()
);*/
assign txa_data_out_p = txa_data_in;
assign txa_data_out_n = !txa_data_in;
endmodule
```

9.1.3. CDR.sv

```
module CDR(rst, clks_in, a_rx, samp_test);
  input rst;
  input [3:0] clks_in;
  input a_rx;
  //output reg c_rx;
  output reg samp_test;
  reg [3:0] c_rx_upsampled;
  reg [3:0] c_rx_upsampled_reg;
  reg [3:0] c_rx_upsampled_2reg;
  reg [3:0] c_rx_upsampled_3reg;
  reg [3:0] c_rx_upsampled_4reg;
  wire [3:0] a_xor;
  reg [3:0] a_xor_reg;
  //wire [3:0] a_and;
  reg [3:0] a_and_reg;
  reg [1:0] best_samp;
  reg [1:0] best_samp_reg;
  reg a_rx_reg;
  wire [2:0] resA1;
  wire [2:0] resA2;
  wire [3:0] num_of_ones_in_rx;
  wire clk;
  //wire edge_detected;
  // Use the phase 0 clk as the system clk
  assign clk = clks_in[0];
  assign a_xor[3]= c_rx_upsampled[1] ^ c_rx_upsampled[2];
  assign a_xor[0]= c_rx_upsampled[2] ^ c_rx_upsampled[3];
  assign a_xor[1]= c_rx_upsampled[1] ^ c_rx_upsampled[0];
  assign a_xor[2]= c_rx_upsampled[0] ^ c_rx_upsampled[1];
  // CDR
  // Up-sample de serial input with clks_in at different phases.
  genvar index;
  generate
    for (index=0; index < 4; index=index+1) begin : gen_upsample
      always @(posedge clks_in[index] or negedge rst) begin
        if (rst == 1'b0) begin
          c_rx_upsampled[index] <= 1'b0;
          //a_rx_reg <= 1'b0;
        end else begin
          c_rx_upsampled[index] <= a_rx;
          //a_rx_reg <= a_rx;
        end
      end
    end
  endgenerate
  // Transition tracking
  always@(posedge clk or negedge rst)
    begin
```

```
      if(rst == 1'b0) begin
        a_xor_reg <= 4'h0;
        c_rx_upsampled_reg <= 4'h0;
        c_rx_upsampled_2reg <= 4'h0;
        c_rx_upsampled_3reg <= 4'h0;
        c_rx_upsampled_4reg <= 4'h0;
      end else begin
        a_xor_reg <= a_xor;
        c_rx_upsampled_reg <= c_rx_upsampled;
        c_rx_upsampled_2reg <= c_rx_upsampled_reg;
        c_rx_upsampled_3reg <= c_rx_upsampled_2reg;
        c_rx_upsampled_4reg <= c_rx_upsampled_3reg;
      end
    end
  always@(posedge clk or negedge rst)
    begin
      if(rst == 1'b0) begin
        best_samp <= 2'h0;
        best_samp_reg <= 2'h0;
      end else begin
        best_samp_reg <= best_samp;
        //if(edge_detected_Xreg) begin
        casex(a_xor_reg)
          4'b1xxx: best_samp <= 2'h3 ;
          4'b01xx: best_samp <= 2'h2 ;
          4'b001x: best_samp <= 2'h1 ;
          4'b0001: best_samp <= 2'h0 ;
          default: best_samp <= best_samp ;
        endcase
        //end
      end
    end
  // Sample only in the best timing according to the
  // transition detector output
  always@(posedge clk or negedge rst)
    begin
      if(rst == 1'b0) begin
        samp_test <= 1'b0;
      end else begin
        samp_test <= c_rx_upsampled_4reg[best_samp_reg];
      end
    end
endmodule
```

9.1.4. clock_divider.sv

```
/*****************************************************************************
*Name:
* clock_divider.v
* This block takes a clock reference and divides it by 8.
* It will generate 8 clocks at equidistant phases.
/*****************************************************************************/
module clock_divider(
  input rst,
  input clk,
  output reg [3 : 0] clks_out
);
  reg [1 : 0] div_by_4_q;
  reg [1 : 0] div_by_4_reg;
  wire clk_div_by_4;
  assign clk_div_by_4 = div_by_4_reg[1];
  always@(posedge clk or negedge rst) begin
    if(rst == 1'b0) begin
      div_by_4_q <= 2'b0 ;
      div_by_4_reg <= 2'b0 ;
    end else begin
      div_by_4_q <= {div_by_4_q[0], ~div_by_4_q[1]} ;
      div_by_4_reg <= div_by_4_q ;
    end
  end
```

```
  always@(posedge clk or negedge rst) begin
    if(rst == 1'b0) begin
      clks_out <= 4'd0;
    end else begin
      clks_out <= {clks_out[2 : 0], clk_div_by_4};
    end
  end
endmodule
```

9.1.5. Comparator.sv

```
/*************************************************************
* Name:
* Comparator.v
* Description:
* This module returns 1 when the 2 input n bits variable is wired.
* Combinational implementation
* Inputs:
* Data A
* Data B
* Outputs
* Comp_out
* Versión:
* 1.0
* Author:
* Christian Aparicio Zuleta
* Fecha:
* 09/09/2017
*************************************************************/
module Comparator
  #(
    parameter WORD_LENGTH = 8
  )
  (
    // Input ports
    input [WORD_LENGTH-1 : 0] Data_A,
    input [WORD_LENGTH-1 : 0] Data_B,
    // Output Ports
    output Comp_out
  );
  //Internal wires
  wire [WORD_LENGTH-1 : 0] XOR_out;
  wire OR_out;
  //Combinational logic
  assign XOR_out = Data_A ^ Data_B;
  assign OR_out = | XOR_out;
  assign Comp_out = !OR_out;
endmodule
```

9.1.6. ComparatorP.sv

```
/*************************************************************************
*Name: ComparatorP.v
*Description: This module compares the data transmitted with the received
one. Is composed of 3 submodules, one controls the transmitted data
recording, other module makes reads the memory data and compares it with the
received data. The last submodule is a memory to saving all the transmitted
data due to the difference in time with the transmission and recepcion.
*Inputs:
* clk: The input for the global clock.
* rst: The global reset signal.
* dataA_in: the data being transmitted.
* dataB_in: received data.
* lfsr_en: indicates when the transmitter starts sending new data.
* data_valid_pipe: indicates when the digital receiver output is valid.
*Outputs:
* bist_end: indicates when the data comparison received between two commas
```

```
has ended.
* num_errors: number of mismatches.
*Version:
* 1.0
*Author:
* Miguel Mihail Hoil Loria.
*Date:
* 08/10/2017
**********************************************************************************/
module ComparatorP #(
  parameter WIDTH = 9,
  parameter MEM_SIZE = 8,
  parameter [8 : 0] COMMA = 9'b1_1111_1100,
  parameter ERRORS_WIDTH = 6
)
  (
    //------------Inputs--------
    input clk,
    input rst,
    input [ WIDTH -1 : 0 ] dataA_in,
    input [ WIDTH -1 : 0 ] dataB_in,
    input lfsr_en,
    input data_valid_pipe,
    //------------Outputs--------
    output bist_end,
    output [ERRORS_WIDTH-1 : 0 ] num_errors
  );
  localparam CNTR_SIZE = $clog2(MEM_SIZE);
  wire [CNTR_SIZE - 1 : 0] cntA_wire, cntB_wire;
  wire [ERRORS_WIDTH-1 : 0 ] num_errors_wire;
  wire bist_end_wire, write_enable_wire;
  wire [WIDTH-2 : 0] MemData_wire; //0 -> WIDTH-2 = 8 data bits without Ko
  dataA_save
  #(
    .WIDTH(WIDTH),
    .DECODED_COMMA(COMMA),
    .MEM_SIZE(MEM_SIZE)
  )
  dataA_Unit (
    //Inputs
    .clk(clk),
    .reset(rst),
    .lfsr_en(lfsr_en),
    .dataA(dataA_in),
    //Outputs
    .cntA(cntA_wire),
    .we(write_enable_wire)
  );
  simple_dual_port_ram_single_clock #(
    .DATA_WIDTH(WIDTH-1),
    .ADDR_WIDTH(CNTR_SIZE)
  )
  MemoryData_Unit (
    //INPUTS
    .clk(clk),
    .data(dataA_in[WIDTH-2 : 0]),
    .write_addr(cntA_wire),
    .we(write_enable_wire),
    .read_addr(cntB_wire),
    //OUTPUTS
    .q(MemData_wire)
  );
  dataB_compare #(
    .WIDTH(WIDTH),
    .DECODED_COMMA(COMMA),
    .MEM_SIZE(MEM_SIZE),
    .ERRORS_WIDTH(ERRORS_WIDTH)
  )
  dataB_Unit (
    //Inputs
    .clk(clk),
```

```
    .reset(rst),
    .data_valid_pipe(data_valid_pipe),
    .dataB(dataB_in),
    .MemData(MemData_wire),
    //Outputs
    .cntB(cntB_wire),
    .bist_end(bist_end_wire),
    .num_errors(num_errors_wire)
  );
  assign bist_end = bist_end_wire;
  assign num_errors = num_errors_wire;
endmodule
```

9.1.7. Control_serial.sv

```
/************************************************************
* Name:
* Control_serial.v
* Description:
* This module delivers enable signals, based in comparators
* Parameters:
* COUNT_LENGTH: size of the counter
* Inputs:
* Counter
* Outputs:
* eight, nine, ten
* Versión:
* 1.0
* Author:
* Christian Aparicio Zuleta
* Date:
* 24/04/2017
************************************************************/
module Control_serial
  #(
    parameter COUNT_LENGTH
  )
  (
    input [COUNT_LENGTH-1:0] counter,
    output eight, nine, ten
  );
  Comparator #(COUNT_LENGTH) Comp_Counter_eight(4'd8,counter,eight);
  Comparator #(COUNT_LENGTH) Comp_Counter_nine(4'd9,counter,nine);
  Comparator #(COUNT_LENGTH) Comp_Counter_ten(4'd10,counter,ten);
endmodule
```

9.1.8. dataA_save.sv

```
/************************************************************************
*Name:
* dataA_save.v
*Description:
* This module controls memory recording for data transmitted.
*Inputs:
* clk: The input for the global clock.
* reset: The global reset signal.
* dataA: the data being transmitted.
* lfsr_en: indicates when the transmitter starts sending new data
*Outputs:
* cntA: write memory pointer
* we: enables memory writing
*Version:
* 1.0
*Author:
* Miguel Mihail Hoil Loria.
*Date:
* 31/05/2017
*-Changes:
```

```
* 02/09/2017
* -Finite state machine was changed for combinational logic.
* 20/10/2017
* -Registers for input data were added.
**********************************************************************/
module dataA_save #(
  parameter WIDTH = 9,
  parameter [8 : 0] DECODED_COMMA = 9'b1_1111_1100,
  parameter MEM_SIZE = 8
)
  (
    //Inputs
    input clk,
    input reset,
    input  lfsr_en,
    input [ WIDTH -1 : 0 ] dataA,
    //Outputs
    output [$clog2(MEM_SIZE) - 1 : 0] cntA,
    output we
  );
  localparam CNTR_SIZE = $clog2(MEM_SIZE);
  localparam IDLE = 2'd0;
  localparam WAIT = 2'd1;
  localparam SAVE = 2'd2;
  reg [1 : 0] State;
  wire lfsr_en_RegOut_wire;
  wire [WIDTH -1 : 0] dataA_RegOut_wire;
  wire [1 : 0] nxtSte_wire;
  wire notSt1_St0_wire;
  wire enable_register_wire;
  wire dataA_is_dcdComma_wire;
  wire [CNTR_SIZE - 1 : 0] cntA_wire, cntA_nxt_wire;
  // - * - * - Code Starts - * - * -
  // - - - Register data Input - - -
  Registro #(1)
  LFSRen_register (
    // Input Ports
    .clk(clk),
    .reset(reset),
    .enable(1'b1),
    .Data_Input(lfsr_en),
    //Output Ports
    .Data_Output(lfsr_en_RegOut_wire)
  );
  Registro #(WIDTH)
  DataA_register (
    //Input Ports
    .clk(clk),
    .reset(reset),
    .enable(lfsr_en),
    .Data_Input(dataA),
    //Output Ports
    .Data_Output(dataA_RegOut_wire)
  );
  // - * - Sequential state assignment - * -
  always@(posedge clk or negedge reset) begin
    if(reset == 1'b0)
      State <= IDLE;
    else
      State <= nxtSte_wire;
  end
  // - * - Combinational control signals assignment - * -
  assign notSt1_St0_wire = (!State[1]) & State[0];
  assign enable_register_wire = State[1] & (!State[0]);
  assign nxtSte_wire[1] = notSt1_St0_wire & lfsr_en_RegOut_wire &
    (!dataA_is_dcdComma_wire);
  assign nxtSte_wire[0] = enable_register_wire | (notSt1_St0_wire &
    (!lfsr_en_RegOut_wire))
    | ((!State[0]) & lfsr_en_RegOut_wire & dataA_is_dcdComma_wire);
  // - * - Structural signals assignment - * -
  Registro #(CNTR_SIZE)
```

```
  CounterA_register
  (
    //Input Ports
    .clk(clk),
    .reset(reset),
    .enable(enable_register_wire),
    .Data_Input(cntA_nxt_wire),
    //Output Ports
    .Data_Output(cntA_wire)
  );
  assign dataA_is_dcdComma_wire = dataA_RegOut_wire == DECODED_COMMA;
  assign cntA_nxt_wire = cntA_wire + 1'b1;
  assign cntA = cntA_wire;
  assign we = enable_register_wire;
endmodule
```

9.1.9. dataB_compare.sv

```
/**********************************************************************
*Name:
* dataB_compare.v
*Description:
* This module makes the comparison reading the memory data and comparing
it with the received data.
*Inputs:
* clk: The input for the global clock.
* reset: The global reset signal.
* data_valid_pipe: indicates when the digital receiver output is valid.
* dataB: received data.
* MemData: memory data read.
*Outputs:
* cntB: read memory pointer
* bist_end: indicates when the data comparison received between two commas
has ended.
* num_errors: number of mismatches.
*Version:
* 1.0
*Author:
* Miguel Mihail Hoil Loria.
*Date:
* 31/05/2017
*-Changes:
* 02/09/2017
* -Finite state machine was changed for combinational logic.
* 12/09/2017
* -D flip-flops stages added
* 20/10/2017
* -Registers for input data were added. The registers surrounding
* the combinational logic were reduced to one stage.
**********************************************************************/
module dataB_compare #( parameter WIDTH = 9, parameter [8 : 0] DECODED_COMMA
                        = 9'b1_1111_1100, parameter MEM_SIZE = 8, parameter ERRORS_WIDTH = 6
)
  (
    //Inputs
    input clk,
    input reset,
    input data_valid_pipe,
    input [WIDTH-1 : 0] dataB,
    input [WIDTH-2 : 0] MemData,
    //Outputs
    output [$clog2(MEM_SIZE) - 1 : 0] cntB,
    output bist_end,
    output [ERRORS_WIDTH-1 : 0 ] num_errors
  );
  // * * * * Variables * * * *
  localparam CNTR_SIZE = $clog2(MEM_SIZE);
  localparam IDLE = 2'd0;
  localparam CLEAR = 2'd1;
  localparam WAIT = 2'd2;
```

```
  localparam COMPARE = 2'd3;
  reg [1 : 0] State;
  wire [1 : 0] nxtSte_wire;
  wire [WIDTH-1 : 0] dataB_RegOut_wire;
  wire [WIDTH-2 : 0] MemData_RegOut_wire;
  wire data_valid_pipe_RegOut_wire;
  wire flag_end_wire;
  wire syncRst_registers_wire;
  wire enable_cntrB_register_wire;
  wire S1_notS0_wire;
  wire data_is_comma_wire;
  wire data_equal_wire;
  wire [ERRORS_WIDTH-1 : 0 ] errors_next_wire, errors_wire;
  wire [CNTR_SIZE - 1 : 0] cntB_wire, cntB_nxt_wire;
  wire bist_end_wire;
  wire enable_errors_register_wire;
  wire enable_BistEnd_register_wire;
  wire enable_cntrB_register_pipeB_wire, syncRst_registers_pipeB_wire;
  wire data_is_comma_pipeB_wire;
  wire enable_BistEnd_register_pipeB_wire;
  wire enable_errors_register_pipeB_wire;
  // - * - * - Code Starts - * - * -
  // - - - Register input data - - -
  Registro #(1) DataValidPipeIn_register (
// Input Ports
    .clk(clk),
    .reset(reset),
    .enable(1'b1),
    .Data_Input(data_valid_pipe),
// Output Ports
    .Data_Output(data_valid_pipe_RegOut_wire)
  );
  Registro #(WIDTH-1) MemData_register (
   // Input Ports
    .clk(clk),
    .reset(reset),
    .enable(data_valid_pipe),
    .Data_Input(MemData),
// Output Ports
    .Data_Output(MemData_RegOut_wire)
  );
  Registro #(WIDTH) DataB_register (
   // Input Ports
    .clk(clk),
    .reset(reset),
    .enable(data_valid_pipe),
    .Data_Input(dataB),
// Output Ports
    .Data_Output(dataB_RegOut_wire)
  );
  // - - - Equality comparators - - -
  Comparator #(.WORD_LENGTH(WIDTH) ) DataIsComma_Comparator (
   // Input Ports
    .Data_A(dataB_RegOut_wire),
    .Data_B(DECODED_COMMA),
   // Output Ports
    .Comp_out(data_is_comma_wire)
  );
  Comparator #(.WORD_LENGTH(WIDTH-1)) DataIsEqual_Comparator (
   // Input Ports
    .Data_A(dataB_RegOut_wire[WIDTH-2 : 0]),
    .Data_B(MemData_RegOut_wire),
   // Output Ports
    .Comp_out(data_equal_wire)
  );
  // - * - Combinational control signals assignment - * -
  assign S1_notS0_wire = State[1] & (!State[0]);
  assign flag_end_wire = State[1] & State[0];
  assign syncRst_registers_wire = (!State[1]) & State[0];
  assign enable_cntrB_register_wire = flag_end_wire & (!data_is_comma_wire);
  assign nxtSte_wire[1] = syncRst_registers_wire | enable_cntrB_register_wire |
```

```
  S1_notS0_wire;
assign nxtSte_wire[0] = ((!State[0]) & data_valid_pipe_RegOut_wire &
                         data_is_comma_wire)
  | (S1_notS0_wire & data_valid_pipe_RegOut_wire);
assign enable_errors_register_wire = enable_cntrB_register_wire &
  !data_equal_wire | syncRst_registers_wire;
assign enable_BistEnd_register_wire = flag_end_wire & data_is_comma_wire |
  syncRst_registers_wire;
// - - - Flops Stage B - - -
Registro #(1) EnableRegister_register (
 // Input Ports
  .clk(clk),
  .reset(reset),
  .enable(1'b1),
  .Data_Input(enable_cntrB_register_wire),
  .Data_Output(enable_cntrB_register_pipeB_wire)
);
Registro #(1) SyncRst_register (
 // Input Ports
  .clk(clk),
  .reset(reset),
  .enable(1'b1),
  .Data_Input(syncRst_registers_wire),
 // Output Ports
  .Data_Output(syncRst_registers_pipeB_wire)
);
Registro #(1) DataIsComma_PipeB_register (
 // Input Ports
  .clk(clk),
  .reset(reset),
  .enable(1'b1),
  .Data_Input(data_is_comma_wire),
 // Output Ports
  .Data_Output(data_is_comma_pipeB_wire)
);
Registro #(1) Enable_BistEnd_register (
 // Input Ports
  .clk(clk),
  .reset(reset),
  .enable(1'b1),
  .Data_Input(enable_BistEnd_register_wire),
 // Output Ports
  .Data_Output(enable_BistEnd_register_pipeB_wire)
);
Registro #(1) Enable_Errors_register (
 // Input Ports
  .clk(clk),
  .reset(reset),
  .enable(1'b1),
  .Data_Input(enable_errors_register_wire),
 // Output Ports
  .Data_Output(enable_errors_register_pipeB_wire)
);
// - * - Sequential state assignment - * -
always@(posedge clk or negedge reset) begin
  if(reset == 1'b0)
    State <= IDLE;
  else
    State <= nxtSte_wire;
end
// - * - Structural signals assignment - * -
Registro #(CNTR_SIZE) CounterB_register (
 // Input Ports
  .clk(clk),
  .reset(reset),
  .enable(enable_cntrB_register_pipeB_wire),
  .Data_Input(cntB_nxt_wire),
 // Output Ports
  .Data_Output(cntB_wire)
);
RegistroSyncRst #(6) Errors_register (
```

```
    // Input Ports
     .clk(clk),
     .reset(reset),
     .sync_reset(syncRst_registers_pipeB_wire),
     .enable(enable_errors_register_pipeB_wire),
     .Data_Input(errors_next_wire),
    // Output Ports
     .Data_Output(errors_wire)
   );
   RegistroSyncRst #(1) Bist_end_register (
   // Input Ports
     .clk(clk),
     .reset(reset),
     .sync_reset(syncRst_registers_pipeB_wire),
     .enable(enable_BistEnd_register_pipeB_wire),
     .Data_Input(data_is_comma_pipeB_wire),
   // Output Ports
     .Data_Output(bist_end_wire)
   );
   //assign data_is_comma_wire = dataB_RegOut_wire == DECODED_COMMA;
   //assign data_equal_wire = dataB_RegOut_wire[WIDTH-2 : 0] ==   MemData_RegOut_wire;
   assign errors_next_wire = errors_wire + 1'b1;
   assign cntB_nxt_wire = cntB_wire + 1'b1;
   // - - Outputs assignment - -
   assign cntB = cntB_wire;
   assign num_errors = errors_wire;
   assign bist_end = bist_end_wire;
endmodule
```

9.1.10. decodePipe.sv

```
// Chuck Benz, Hollis, NH   Copyright (c)2002
//
// The information and description contained herein is the
// property of Chuck Benz.
//
// Permission is granted for any reuse of this information
// and description as long as this copyright notice is
// preserved.  Modifications may be made as long as this
// notice is preserved.

// per Widmer and Franaszek

// Pipelining techniques added by Rogelio Rivas

module decodePipe (rst, clk, data_valid, datain, dispin, dataout, data_valid_pipe, dispout,
disp_err, code_err, ko) ;
  input rst;
  input clk;
  input data_valid;
  input [9:0]   datain ;
  input     dispin ;
  output [7:0]  dataout ;
  output    dispout ;
  output    code_err ;
  output    disp_err ;
  output ko ;
  output data_valid_pipe ;

  wire ai = datain[0] ;
  wire bi = datain[1] ;
  wire ci = datain[2] ;
  wire di = datain[3] ;
  wire ei = datain[4] ;
  wire ii = datain[5] ;
  wire fi = datain[6] ;
  wire gi = datain[7] ;
  wire hi = datain[8] ;
  wire ji = datain[9] ;
```

```
reg ai_reg ;
reg bi_reg ;
reg ci_reg ;
reg di_reg ;
reg ei_reg ;
reg ii_reg ;
reg fi_reg ;
reg gi_reg ;
reg hi_reg ;
reg ji_reg ;
reg dispin_reg ;

reg ai_2reg ;
reg bi_2reg ;
reg ci_2reg ;
reg di_2reg ;
reg ei_2reg ;
reg ii_2reg ;
reg fi_2reg ;
reg gi_2reg ;
reg hi_2reg ;
reg ji_2reg ;
//reg aeqb_reg  ;
//reg ceqd_reg ;
reg p22_reg ;
reg p13_reg ;
reg p31_reg ;
reg p40_reg ;
reg p04_reg ;
reg dispin_2reg ;

reg p22bceeqi_reg ;
reg p22bncneeqi_reg ;
reg p13in_reg ;
reg p31i_reg ;
reg p13dei_reg ;
reg p22aceeqi_reg ;
reg p22ancneeqi_reg ;
reg p13en_reg ;
reg anbnenin_reg ;
reg abei_reg ;
//reg cdei_reg ;
reg cndnenin_reg ;
reg disp6a_reg ; // pos disp if p22 and was pos, or p31.
reg disp6a2_reg ;  // disp is ++ after 4 bits
reg disp6a0_reg ; // -- disp after 4 bits
reg feqg_reg ;
reg heqj_reg ;

// non-zero disparity cases:
//reg p22enin_reg ;
//reg p22ei_reg ;
//wire p13in = p12 & !ii ;
//wire p31i = p31 & ii ;
//reg p31dnenin_reg ;
//wire p13dei = p13 & di & ei & ii ;
//reg p31e_reg ;

reg ai_3reg ;
reg bi_3reg ;
reg ci_3reg ;
reg di_3reg ;
reg ei_3reg ;
reg ii_3reg ;
reg fi_3reg ;
reg gi_3reg ;
reg hi_3reg ;
reg ji_3reg ;
reg dispin_3reg ;
reg disp6p_reg ;
```

```
reg disp6n_reg ;
reg p40_2reg ;
reg p04_2reg  ;
reg p31_2reg ;
reg p13_2reg ;

reg ai_4reg ;
reg bi_4reg ;
reg ci_4reg ;
reg di_4reg ;
reg ei_4reg ;
reg ii_4reg ;
reg fi_4reg ;
reg gi_4reg ;
reg hi_4reg ;
reg ji_4reg ;
reg compa_reg ;
reg compb_reg ;
reg compc_reg ;
reg compd_reg ;
reg compe_reg ;
//reg k28_reg ;
reg k28p_reg ;
reg dispin_4reg ;
reg disp6p_2reg ;
reg disp6n_2reg ;
reg dispaux_a_reg ;
reg code_aux_a_reg ;
reg code_aux_b_reg ;
reg code_aux_c_reg ;
reg code_aux_d_reg ;
reg p13_3reg ;
reg p31_3reg ;

reg disp6b_reg ;
reg fghj22_reg ;
reg fghjp13_reg ;
reg fghjp31_reg ;

reg ao_reg ;
reg bo_reg ;
reg co_reg ;
reg do__reg ;
reg eo_reg ;
reg fo_reg ;
reg go_reg ;
reg ho_reg ;

reg ko_reg ;

reg dispout_reg;
reg disp_err_reg;
reg code_err_reg;

reg data_valid_1pipe;
reg data_valid_2pipe;
reg data_valid_3pipe;
reg data_valid_4pipe;
reg data_valid_5pipe;

  wire aeqb = (ai_reg & bi_reg) | (!ai_reg & !bi_reg) ;
wire ceqd = (ci_reg & di_reg) | (!ci_reg & !di_reg) ;
wire p22 = (ai_reg & bi_reg & !ci_reg & !di_reg) |
       (ci_reg & di_reg & !ai_reg & !bi_reg) |
       ( !aeqb & !ceqd) ;
wire p13 = ( !aeqb & !ci_reg & !di_reg) |
       ( !ceqd & !ai_reg & !bi_reg) ;
wire p31 = ( !aeqb & ci_reg & di_reg) |
       ( !ceqd & ai_reg & bi_reg) ;

wire p40 = ai_reg & bi_reg & ci_reg & di_reg ;
```

```
  wire p04 = !ai_reg & !bi_reg & !ci_reg & !di_reg ;

  wire disp6a = p31_reg | (p22_reg & dispin_2reg) ; // pos disp if p22 and was pos, or p31.
   wire disp6a2 = p31_reg & dispin_2reg ;  // disp is ++ after 4 bits
   wire disp6a0 = p13_reg & ! dispin_2reg ; // -- disp after 4 bits

  wire disp6b = (((ei_3reg & ii_3reg & ! disp6a0_reg) | (disp6a_reg & (ei_3reg | ii_3reg)) |
disp6a2_reg |
          (ei_3reg & ii_3reg & di_3reg)) & (ei_3reg | ii_3reg | di_3reg)) ;

  // The 5B/6B decoding special cases where ABCDE != abcde

  wire p22bceeqi = p22_reg & bi_2reg & ci_2reg & (ei_2reg == ii_2reg) ;
  wire p22bncneeqi = p22_reg & !bi_2reg & !ci_2reg & (ei_2reg == ii_2reg) ;
  wire p13in = p13_reg & !ii_2reg ;
  wire p31i = p31_reg & ii_2reg ;
  wire p13dei = p13_reg & di_2reg & ei_2reg & ii_2reg ;
  wire p22aceeqi = p22_reg & ai_2reg & ci_2reg & (ei_2reg == ii_2reg) ;
  wire p22ancneeqi = p22_reg & !ai_2reg & !ci_2reg & (ei_2reg == ii_2reg) ;
  wire p13en = p13_reg & !ei_2reg ;
  wire anbnenin = !ai_2reg & !bi_2reg & !ei_2reg & !ii_2reg ;
  wire abei = ai_2reg & bi_2reg & ei_2reg & ii_2reg ;
  //wire cdei = ci_2reg & di_2reg & ei_2reg & ii_2reg ;
  wire cndnenin = !ci_2reg & !di_2reg & !ei_2reg & !ii_2reg ;

  // non-zero disparity cases:
  //wire p22enin = p22_reg & !ei_2reg & !ii_2reg ;
  //wire p22ei = p22_reg & ei_2reg & ii_2reg ;
  //wire p13in = p12 & !ii ;
  //wire p31i = p31 & ii ;
  //wire p31dnenin = p31_reg & !di_2reg & !ei_2reg & !ii_2reg ;
  //wire p13dei = p13 & di & ei & ii ;
  //wire p31e = p31_reg & ei_2reg ;

  wire compa = p22bncneeqi_reg | p31i_reg | p13dei_reg | p22ancneeqi_reg |
        p13en_reg | abei_reg | cndnenin_reg ;
  wire compb = p22bceeqi_reg | p31i_reg | p13dei_reg | p22aceeqi_reg |
        p13en_reg | abei_reg | cndnenin_reg ;
  wire compc = p22bceeqi_reg | p31i_reg | p13dei_reg | p22ancneeqi_reg |
        p13en_reg | anbnenin_reg | cndnenin_reg ;
  wire compd = p22bncneeqi_reg | p31i_reg | p13dei_reg | p22aceeqi_reg |
        p13en_reg | abei_reg | cndnenin_reg ;
  wire compe = p22bncneeqi_reg | p13in_reg | p13dei_reg | p22ancneeqi_reg |
        p13en_reg | anbnenin_reg | cndnenin_reg ;

  wire ao = ai_4reg ^ compa_reg ;
  wire bo = bi_4reg ^ compb_reg ;
  wire co = ci_4reg ^ compc_reg ;
  wire do_ = di_4reg ^ compd_reg ;
  wire eo = ei_4reg ^ compe_reg ;

  wire feqg = (fi_2reg & gi_2reg) | (!fi_2reg & !gi_2reg) ;
  wire heqj = (hi_2reg & ji_2reg) | (!hi_2reg & !ji_2reg) ;

  wire fghj22 = (fi_3reg & gi_3reg & !hi_3reg & !ji_3reg) |
        (!fi_3reg & !gi_3reg & hi_3reg & ji_3reg) |
        ( !feqg_reg & !heqj_reg) ;
  wire fghjp13 = ( !feqg_reg & !hi_3reg & !ji_3reg) |
         ( !heqj_reg & !fi_3reg & !gi_3reg) ;
  wire fghjp31 = ( (!feqg_reg) & hi_3reg & ji_3reg) |
         ( !heqj_reg & fi & gi_3reg) ;

  wire dispout_pre = (fghjp31_reg | (disp6b_reg & fghj22_reg) | (hi_4reg & ji_4reg)) &
(hi_4reg | ji_4reg) ;
  wire ko_pre = ( (ci_4reg & di_4reg & ei_4reg & ii_4reg) | ( !ci_4reg & !di_4reg & !ei_4reg
& !ii_4reg) |
        (p13_3reg & !ei_4reg & ii_4reg & gi_4reg & hi_4reg & ji_4reg) |
        (p31_3reg & ei_4reg & !ii_4reg & !gi_4reg & !hi_4reg & !ji_4reg)) ;

  /*wire alt7 =   (fi & !gi & !hi & // 1000 cases, where disp6b is 1
        ((dispin & ci & di & !ei & !ii) | ko |
```

```
          (dispin & !ci & di & !ei & !ii))) |
        (!fi & gi & hi & // 0111 cases, where disp6b is 0
         (( !dispin & !ci & !di & ei & ii) | ko |
          ( !dispin & ci & !di & ei & ii))) ;*/

  //wire k28 = (ci_3reg & di_3reg & ei_3reg & ii_3reg) | ! (ci_3reg | di_3reg | ei_3reg |
ii_3reg) ;
  // k28 with positive disp into fghi - .1, .2, .5, and .6 special cases
  wire k28p = ! (ci_3reg | di_3reg | ei_3reg | ii_3reg) ;
  wire fo = (ji_4reg & !fi_4reg & (hi_4reg | !gi_4reg | k28p_reg)) |
        (fi_4reg & !ji_4reg & (!hi_4reg | gi_4reg | !k28p_reg)) |
        (k28p_reg & gi_4reg & hi_4reg) |
        (!k28p_reg & !gi_4reg & !hi_4reg) ;
  wire go = (ji_4reg & !fi_4reg & (hi_4reg | !gi_4reg | !k28p_reg)) |
        (fi_4reg & !ji_4reg & (!hi_4reg | gi_4reg |k28p_reg)) |
        (!k28p_reg & gi_4reg & hi_4reg) |
        (k28p_reg & !gi_4reg & !hi_4reg) ;
  wire ho = ((ji_4reg ^ hi_4reg) & ! ((!fi_4reg & gi_4reg & !hi_4reg & ji_4reg & !k28p_reg)
| (!fi_4reg & gi_4reg & hi_4reg & !ji_4reg & k28p_reg) |
                (fi_4reg & !gi_4reg & !hi_4reg & ji_4reg & !k28p_reg) | (fi_4reg & !gi_4reg
& hi_4reg & !ji_4reg & k28p_reg))) |
        (!fi_4reg & gi_4reg & hi_4reg & ji_4reg) | (fi_4reg & !gi_4reg & !hi_4reg &
!ji_4reg) ;

  wire disp6p = (p31_reg & (ei_2reg | ii_2reg)) | (p22_reg & ei_2reg & ii_2reg) ;
  wire disp6n = (p13_reg & ! (ei_2reg & ii_2reg)) | (p22_reg & !ei_2reg & !ii_2reg) ;
  /*
  wire disp4p = fghjp31 ;
  wire disp4n = fghjp13 ;*/

  wire code_aux_a = p40_2reg | p04_2reg | (fi_3reg & gi_3reg & hi_3reg & ji_3reg) |
(!fi_3reg & !gi_3reg & !hi_3reg & !ji_3reg) |
            (p13_2reg & !ei_3reg & !ii_3reg) | (p31_2reg & ei_3reg & ii_3reg) |
            (ei_3reg & ii_3reg & fi_3reg & gi_3reg & hi_3reg) | (!ei_3reg & !ii_3reg &
!fi_3reg & !gi_3reg & !hi_3reg) ;

  wire code_aux_b = (ei_3reg & !ii_3reg & gi_3reg & hi_3reg & ji_3reg) | (!ei_3reg & ii_3reg
& !gi_3reg & !hi_3reg & !ji_3reg) |
            (!p31_2reg & ei_3reg & !ii_3reg & !gi_3reg & !hi_3reg & !ji_3reg) |
            (!p13_2reg & !ei_3reg & ii_3reg & gi_3reg & hi_3reg & ji_3reg) ;

  wire code_aux_c = (((ei_3reg & ii_3reg & !gi_3reg & !hi_3reg & !ji_3reg) |
              (!ei_3reg & !ii_3reg & gi_3reg & hi_3reg & ji_3reg)) &
             ! ((ci_3reg & di_3reg & ei_3reg) | (!ci_3reg & !di_3reg & !ei_3reg))) ;

  wire code_aux_d = (ci_3reg & di_3reg & ei_3reg & ii_3reg & !fi_3reg & !gi_3reg & !hi_3reg)
|
            (!ci_3reg & !di_3reg & !ei_3reg & !ii_3reg & fi_3reg & gi_3reg & hi_3reg) ;

  wire code_err_pre = code_aux_a_reg |
            code_aux_b_reg |
            code_aux_c_reg |
            (disp6p_2reg & fghjp31_reg) | (disp6n & fghjp13_reg) |
            (ai_4reg & bi_4reg & ci_4reg & !ei_4reg & !ii_4reg & ((!fi_4reg & !gi_4reg) |
fghjp13_reg)) |
            (!ai_4reg & !bi_4reg & !ci_4reg & ei_4reg & ii_4reg & ((fi_4reg & gi_4reg) |
fghjp31_reg)) |
            (fi_4reg & gi_4reg & !hi_4reg & !ji_4reg & disp6p_2reg) |
            (!fi_4reg & !gi_4reg & hi_4reg & ji_4reg & disp6n_2reg) |
            code_aux_d_reg ;

  assign dataout = {ho_reg, go_reg, fo_reg, eo_reg, do__reg, co_reg, bo_reg, ao_reg} ;
  assign data_valid_pipe = data_valid_5pipe ;
  assign dispout = dispout_reg ;
  assign disp_err = disp_err_reg ;
  assign code_err = code_err_reg ;
  assign ko = ko_reg ;

  // my disp err fires for any legal codes that violate disparity, may fire for illegal
codes
  wire dispaux_a = (dispin_3reg & disp6p_reg) | (disp6n_reg & !dispin_3reg) |
```

```
            (dispin_3reg & !disp6n_reg & fi_3reg & gi_3reg) |
            (dispin_3reg & ai_3reg & bi_3reg & ci_3reg) |
              (!dispin_3reg & !disp6p_reg & !fi_3reg & !gi_3reg) |
            (!dispin_3reg & !ai_3reg & !bi_3reg & !ci_3reg);
wire disp_err_pre = ( dispaux_a_reg |
            (dispin_4reg & !disp6n_2reg & fghjp31_reg) |
            (!dispin_4reg & !disp6p_2reg & fghjp13_reg) |
            (disp6p_2reg & fghjp31_reg) | (disp6n_2reg & fghjp13_reg)) ;

always@(posedge clk or negedge rst) begin   // First Pipe Stage
  if(rst == 1'b0) begin
      ai_reg <= 0 ;
      bi_reg <= 0 ;
      ci_reg <= 0 ;
      di_reg <= 0 ;
      ei_reg <= 0 ;
      ii_reg <= 0 ;
      fi_reg <= 0 ;
      gi_reg <= 0 ;
      hi_reg <= 0 ;
      ji_reg <= 0 ;
      data_valid_1pipe <= 0 ;
      dispin_reg <= 0 ;
  end else begin
      data_valid_1pipe <= data_valid ;
      dispin_reg <= dispin ;
      if (data_valid) begin
          ai_reg <= ai ;
          bi_reg <= bi ;
          ci_reg <= ci ;
          di_reg <= di ;
          ei_reg <= ei ;
          ii_reg <= ii ;
          fi_reg <= fi ;
          gi_reg <= gi ;
          hi_reg <= hi ;
          ji_reg <= ji ;
      end else begin
          ai_reg <= ai_reg ;
          bi_reg <= bi_reg ;
          ci_reg <= ci_reg ;
          di_reg <= di_reg ;
          ei_reg <= ei_reg ;
          ii_reg <= ii_reg ;
          fi_reg <= fi_reg ;
          gi_reg <= gi_reg ;
          hi_reg <= hi_reg ;
          ji_reg <= ji_reg ;
      end
  end
end

always@(posedge clk or negedge rst) begin   // Second Pipe Stage
  if(rst == 1'b0) begin
      ai_2reg <= 0 ;
      bi_2reg <= 0 ;
      ci_2reg <= 0 ;
      di_2reg <= 0 ;
      ei_2reg <= 0 ;
      ii_2reg <= 0 ;
      fi_2reg <= 0 ;
      gi_2reg <= 0 ;
      hi_2reg <= 0 ;
      ji_2reg <= 0 ;
      //aeqb_reg <= 0 ;
      //ceqd_reg <= 0 ;
      p22_reg <= 0 ;
      p13_reg <= 0 ;
      p31_reg <= 0 ;
      p40_reg <= 0 ;
      p04_reg <= 0 ;
```

```
      data_valid_2pipe <= 0 ;
      dispin_2reg <= 0 ;
  end else begin
     data_valid_2pipe <= data_valid_1pipe ;
      ai_2reg <= ai_reg ;
      bi_2reg <= bi_reg ;
      ci_2reg <= ci_reg ;
      di_2reg <= di_reg ;
      ei_2reg <= ei_reg ;
      ii_2reg <= ii_reg ;
      fi_2reg <= fi_reg ;
      gi_2reg <= gi_reg ;
      hi_2reg <= hi_reg ;
      ji_2reg <= ji_reg ;
      //aeqb_reg <= aeqb ;
      //ceqd_reg <= ceqd ;
      p22_reg <= p22 ;
      p13_reg <= p13 ;
      p31_reg <= p31 ;
      p40_reg <= p40 ;
      p04_reg <= p04 ;
      dispin_2reg <= dispin_reg ;
  end
end

  always@(posedge clk or negedge rst) begin   // Third Pipe Stage
  if(rst == 1'b0) begin
      ai_3reg <= 0 ;
      bi_3reg <= 0 ;
      ci_3reg <= 0 ;
      di_3reg <= 0 ;
      ei_3reg <= 0 ;
      ii_3reg <= 0 ;
      fi_3reg <= 0 ;
      gi_3reg <= 0 ;
      hi_3reg <= 0 ;
      ji_3reg <= 0 ;
      p22bceeqi_reg <= 0 ;
      p22bncneeqi_reg <= 0 ;
      p13in_reg <= 0 ;
      p31i_reg <= 0 ;
      p13dei_reg <= 0 ;
      p22aceeqi_reg <= 0 ;
      p22ancneeqi_reg <= 0 ;
      p13en_reg <= 0 ;
      anbnenin_reg <= 0 ;
      abei_reg <= 0 ;
      //cdei_reg <= 0 ;
      cndnenin_reg <= 0 ;
      //p22enin_reg <= 0 ;
      //p22ei_reg <= 0 ;
      //p31dnenin_reg <= 0 ;
      //p31e_reg <= 0 ;
      data_valid_3pipe <= 0 ;
      feqg_reg <= 0 ;
      heqj_reg <= 0 ;
      disp6a_reg <= 0 ;  // pos disp if p22 and was pos, or p31.
      disp6a2_reg <= 0 ;  // disp is ++ after 4 bits
      disp6a0_reg <= 0 ;
      disp6p_reg <= 0 ;
      disp6n_reg <= 0 ;
      dispin_3reg <= 0 ;
      p40_2reg <= 0 ;
      p04_2reg <= 0 ;
      p31_2reg <= 0 ;
      p13_2reg <= 0 ;
  end else begin
      data_valid_3pipe <= data_valid_2pipe ;
      ai_3reg <= ai_2reg ;
      bi_3reg <= bi_2reg ;
      ci_3reg <= ci_2reg ;
```

```
    di_3reg <= di_2reg ;
    ei_3reg <= ei_2reg ;
    ii_3reg <= ii_2reg ;
    fi_3reg <= fi_2reg ;
    gi_3reg <= gi_2reg ;
    hi_3reg <= hi_2reg ;
    ji_3reg <= ji_2reg ;
    p22bceeqi_reg <= p22bceeqi ;
    p22bncneeqi_reg <= p22bncneeqi ;
    p13in_reg <= p13in ;
    p31i_reg <= p31i ;
    p13dei_reg <= p13dei ;
    p22aceeqi_reg <= p22aceeqi ;
    p22ancneeqi_reg <= p22ancneeqi ;
    p13en_reg <= p13en ;
    anbnenin_reg <= anbnenin ;
    abei_reg <= abei ;
    //cdei_reg <= cdei ;
    cndnenin_reg <= cndnenin ;
    //p22enin_reg <= p22enin ;
    //p22ei_reg <= p22ei ;
    //p31dnenin_reg <= p31dnenin ;
    //p31e_reg <= p31e ;
    feqg_reg <= feqg ;
    heqj_reg <= heqj ;
    disp6a_reg <= disp6a; // pos disp if p22 and was pos, or p31.
    disp6a2_reg <= disp6a2;  // disp is ++ after 4 bits
    disp6a0_reg <= disp6a0;
    disp6p_reg <= disp6p ;
    disp6n_reg <= disp6n ;
    dispin_3reg <= dispin_2reg ;
    p40_2reg <= p40_reg ;
    p04_2reg <= p04_reg ;
    p31_2reg <= p31_reg ;
    p13_2reg <= p13_reg ;
end
end

  always@(posedge clk or negedge rst) begin   // Fourth Pipe Stage
if(rst == 1'b0) begin
    ai_4reg <= 0 ;
    bi_4reg <= 0 ;
    ci_4reg <= 0 ;
    di_4reg <= 0 ;
    ei_4reg <= 0 ;
    ii_4reg <= 0 ;
    fi_4reg <= 0 ;
    gi_4reg <= 0 ;
    hi_4reg <= 0 ;
    ji_4reg <= 0 ;
    compa_reg <= 0 ;
    compb_reg <= 0 ;
    compc_reg <= 0 ;
    compd_reg <= 0 ;
    compe_reg <= 0 ;
    //k28_reg <= 0 ;
    k28p_reg <= 0 ;
    data_valid_4pipe <= 0 ;
    disp6b_reg <= 0;
    fghj22_reg <= 0 ;
    fghjp13_reg <= 0 ;
    fghjp31_reg <= 0 ;
    disp6p_2reg <= 0 ;
    disp6n_2reg <= 0 ;
    dispin_4reg <= 0 ;
    dispaux_a_reg <= 0 ;
    code_aux_a_reg <= 0 ;
    code_aux_b_reg <= 0 ;
    code_aux_c_reg <= 0 ;
    code_aux_d_reg <= 0 ;
    p31_3reg <= 0 ;
```

```
        p13_3reg <= 0 ;
    end else begin
        data_valid_4pipe <= data_valid_3pipe ;
        ai_4reg <= ai_3reg ;
        bi_4reg <= bi_3reg ;
        ci_4reg <= ci_3reg ;
        di_4reg <= di_3reg ;
        ei_4reg <= ei_3reg ;
        ii_4reg <= ii_3reg ;
        fi_4reg <= fi_3reg ;
        gi_4reg <= gi_3reg ;
        hi_4reg <= hi_3reg ;
        ji_4reg <= ji_3reg ;
        compa_reg <= compa ;
        compb_reg <= compb ;
        compc_reg <= compc ;
        compd_reg <= compd ;
        compe_reg <= compe ;
        //k28_reg <= k28 ;
        k28p_reg <= k28p ;
        disp6b_reg <= disp6b ;
        fghj22_reg <= fghj22 ;
        fghjp13_reg <= fghjp13 ;
        fghjp31_reg <= fghjp31 ;
        disp6p_2reg <= disp6p_reg ;
        disp6n_2reg <= disp6n_reg ;
        dispin_4reg <= dispin_3reg ;
        dispaux_a_reg <= dispaux_a ;
        code_aux_a_reg <= code_aux_a ;
        code_aux_b_reg <= code_aux_b ;
        code_aux_c_reg <= code_aux_c ;
        code_aux_d_reg <= code_aux_d ;
        p31_3reg <= p31_2reg ;
        p13_3reg <= p13_2reg ;
    end
  end

        always@(posedge clk or negedge rst) begin   // Fifth Pipe Stage
    if(rst == 1'b0) begin
        ao_reg <= 0 ;
        bo_reg <= 0 ;
        co_reg <= 0 ;
        do__reg <= 0 ;
        eo_reg <= 0 ;
        fo_reg <= 0 ;
        go_reg <= 0 ;
        ho_reg <= 0 ;
        data_valid_5pipe <= 0 ;
        dispout_reg <= 0 ;
        disp_err_reg <= 0 ;
        code_err_reg <= 0 ;
        ko_reg <= 0 ;
    end else begin
        data_valid_5pipe <= data_valid_4pipe ;
        ao_reg <= ao ;
        bo_reg <= bo ;
        co_reg <= co ;
        do__reg <= do_ ;
        eo_reg <= eo ;
        fo_reg <= fo ;
        go_reg <= go ;
        ho_reg <= ho ;
        dispout_reg <= dispout_pre ;
        disp_err_reg <= disp_err_pre ;
        code_err_reg <= code_err_pre ;
        ko_reg <= ko_pre;
    end
  end

endmodule
```

9.1.11. deserializer.sv

```
// deserializer.v
// This block samples an asynchronous signal. and transforms it into
// a source synchronous parallel bus.
// It uses up-sampling data recovery and a shift register to transform the
// serial input stream into a parallel bus.
// This block does not decode or encode any of the inputs.
/*****************************************************************************
*Editor:
* Miguel Mihail Hoil Loria.
*****************************************************************************/
module deserializer #(
  parameter ENCODED_COMMA = 10'h07C
)
  (rst, clks_in, a_rx, c_parallel_out, clk_out, disparity_d, disparity_q,
   c_data_valid);
  input rst;
  input [3:0] clks_in;
  input a_rx;
  input disparity_d ;
  output reg [9:0] c_parallel_out;
  output reg clk_out;
  output reg disparity_q;
  output c_data_valid;
  wire c_rx;
  reg comma_detected_reg;
  wire comma_detected;
  wire clk;
  reg [9:0] shift_2reg;
  reg [3:0] cycle_count;
  wire samp_test;
  wire comma_detected_regOut_wire;
  wire c_data_valid_wire;
  wire restart_counter_wire;
  // Use the phase 0 clk as the system clk
  assign clk = clks_in[0];
  //CDR
  CDR CDR1(
    .rst(rst),
    .clks_in(clks_in),
    .a_rx(a_rx),
    .samp_test(samp_test)
  );
  // Use a flip flop to remember the running disparity in the decoder
  always @(posedge clk or negedge rst) begin
    if (rst == 1'b0)
      disparity_q <= 1'b0;
    else
      disparity_q <= disparity_d;
  end
  // 10 bit shift register
  always@(posedge clk or negedge rst) begin
    if (rst == 1'b0)
      begin
        shift_2reg <= 10'h000;
      end
    /*This code clears the shift registers except the MSB which registers the
sampled bit. Thus, there
won't be a incorrect check of the new data, this is very important to
catch the comma value.*/
    else if(comma_detected | restart_counter_wire)
      begin
        shift_2reg <= {samp_test, 9'b0};
      end
    else begin
      shift_2reg <= {samp_test, shift_2reg[9:1]};
    end
  end
  // Look for comma symbol
```

```
  assign comma_detected = shift_2reg == ENCODED_COMMA || shift_2reg ==
    ~ENCODED_COMMA;
  // Use a 10 cycle counter to generate a data_valid signal.
  // rst the counter if a special sync character, such as a comma is
  // identified
  always @(posedge clk or negedge rst) begin
    if (rst == 1'b0) begin
      cycle_count <= 4'd9;
      clk_out <= 1'b0;
    end
    else begin
      if(comma_detected_regOut_wire) begin //starts the count when a comma is catched
(ENABLE the counter)
        if(restart_counter_wire)
          cycle_count <= 4'd9;// Restart
        else
          cycle_count <= cycle_count - 1'b1;//Countdown
      end
      /* * * clk_out duty cycle * * */
      // Assert clk_out one cycle after data_valid
      // De-assert clk_out in cycle 5
      if(cycle_count == 4'd5)
        clk_out <= 1'b0;
      else if (cycle_count == 4'd0 && (!comma_detected))
        clk_out <= 1'b1;
    end
  end
  /* * * Data is valid when the count down expires. * * */
  Registro #(1)
  C_data_valid_register
  (
    // Input Ports
    .clk(clk),
    .reset(rst),
    .enable(1'b1),
    .Data_Input(comma_detected | restart_counter_wire),
    // Output Ports
    .Data_Output(c_data_valid_wire)
  );
  /* * * * comma detected register * * * */
  /*With a flip-flop T is possible to assert the register with one comma
and with a second one revert it to 0*/
  FlipFlopT #(1)
  CommaDetected_registerFFT
  (
    // Input Ports
    .clk(clk),
    .reset(rst),
    .enable(comma_detected),
    .Data_Input(comma_detected),
    // Output Ports
    .Data_Output(comma_detected_regOut_wire)
  );
  /* * * * data out register * * * */
  always @(posedge clk or negedge rst) begin
    if(rst == 1'b0) begin
      c_parallel_out <= 10'd0;
    end
    else begin
      if (comma_detected | restart_counter_wire) begin
        c_parallel_out <= shift_2reg;
      end
    end
  end
  assign restart_counter_wire = (cycle_count == 4'd0);
  assign c_data_valid = c_data_valid_wire;
endmodule
```

9.1.12. digitalRX.sv

```
module digitalRX
  #(
    parameter ENCODED_COMMA = 10'h07C
  )
  (
    input rst,
    input a_rx,
    input disparity_d,
    output [7:0] dataout,
    output dispout,
    output code_err,
    output disp_err,
    output data_valid_pipe,
    output clk_4f,
    output ko,
    input [3:0]clks_in
  );
  wire dispin ;
  wire [9:0] c_parallel_out;
  wire clk_out;
  wire c_data_valid;
  wire disparity_q;
  wire a_rx_buff;
  wire a_rx_diff_buff;
  assign clk_4f = clks_in[0];
  deserializer #(
    .ENCODED_COMMA(ENCODED_COMMA)
  )
  deserializer1(rst, clks_in, a_rx, c_parallel_out, clk_out, disparity_d,
                disparity_q, c_data_valid);
  decodePipe decode1(rst, clks_in[0], c_data_valid, c_parallel_out,
                     disparity_q, dataout, data_valid_pipe, dispout, disp_err, code_err,
ko);
endmodule
```

9.1.13. encode.sv

```
/*****************************************************************
* Name:
* Encode.v
* Description: This modules executes de codification from 8 bits to 10b
depending in the last Run Disparity
* Fully combinational
* Inputs:
* data_in: 8 bit word input to transmit and 1 bit for control signals
* disp_in: 0 means last disparity was negative
* Outputs
* dataout: 10 bit output word
* disp_out: disparity of the actual word (10 bit)
* Version:
* 2.0
* Author:
* Chuck Benz, Hollis, NH Copyright (c)2002
* Permission is granted for any reuse of this information and description as
 * long as this copyright notice is preserved. Modifications may be made as
 * long as this notice is preserved.
 * Review by:
 * Christian Aparicio Zuleta
 * Date:
 * 24/04/2017
 *****************************************************************/
module encode (datain, dispin, dataout, dispout);
      input [8:0] datain ;
      input dispin ; // 0 = neg disp; 1 = pos disp
      output [9:0] dataout ;
      output dispout ;
      wire ai = datain[0] ;
```

```
        wire bi = datain[1] ;
        wire ci = datain[2] ;
        wire di = datain[3] ;
        wire ei = datain[4] ;
        wire fi = datain[5] ;
        wire gi = datain[6] ;
        wire hi = datain[7] ;
        wire ki = datain[8] ;
        wire aeqb = (ai & bi) | (!ai & !bi) ;
        wire ceqd = (ci & di) | (!ci & !di) ;
        wire l22 = (ai & bi & !ci & !di) |(ci & di & !ai & !bi)| ( !aeqb & !ceqd) ;
        wire l40 = ai & bi & ci & di ;
        wire l04 = !ai & !bi & !ci & !di ;
        wire l13 = ( !aeqb & !ci & !di) | ( !ceqd & !ai & !bi) ;
        wire l31 = ( !aeqb & ci & di) | ( !ceqd & ai & bi) ;

        // The 5B/6B encoding
        wire ao = ai ;
        wire bo = (bi & !l40) | l04 ;
        wire co = l04 | ci | (ei & di & !ci & !bi & !ai) ;
        wire d0 = di & ! (ai & bi & ci) ;
        wire eo = (ei | l13) & ! (ei & di & !ci & !bi & !ai) ;
        wire io = (l22 & !ei) |(ei & !di & !ci & !(ai&bi)) | // D16, D17, D18
        (ei & l40) |(ki & ei & di & ci & !bi & !ai) | // K.28
        (ei & !di & ci & !bi & !ai) ;
        // pds16 indicates cases where d-1 is assumed + to get our encoded value
        wire pd1s6 = (ei & di & !ci & !bi & !ai) | (!ei & !l22 & !l31) ;
        // nds16 indicates cases where d-1 is assumed - to get our encoded value
        wire nd1s6 = ki | (ei & !l22 & !l13) | (!ei & !di & ci & bi & ai) ;
        // ndos6 is pds16 cases where d-1 is + yields - disp out - all of them
        wire ndos6 = pd1s6 ;
        // pdos6 is nds16 cases where d-1 is - yields + disp out - all but one
        wire pdos6 = ki | (ei & !l22 & !l13) ;
        // some Dx.7 and all Kx.7 cases result in run length of 5 case unless
        // an alternate coding is used (referred to as Dx.A7, normal is Dx.P7)
        // specifically, D11, D13, D14, D17, D18, D19.
        wire alt7 = fi&gi&hi&(ki|(dispin ? (!ei & di & l31) : (ei & !di & l13))) ;

        wire fo = fi & ! alt7 ;
        wire go = gi | (!fi & !gi & !hi) ;
        wire ho = hi ;
        wire jo = (!hi & (gi ^ fi)) | alt7 ;
        // nd1s4 is cases where d-1 is assumed - to get our encoded value
        wire nd1s4 = fi & gi ;
        // pd1s4 is cases where d-1 is assumed + to get our encoded value
        wire pd1s4 = (!fi & !gi) | (ki & ((fi & !gi) | (!fi & gi))) ;
        // ndos4 is pd1s4 cases where d-1 is + yields - disp out - just some
        wire ndos4 = (!fi & !gi) ;
        // pdos4 is nd1s4 cases where d-1 is - yields + disp out
        wire pdos4 = fi & gi & hi ;
        wire compls6 = (pd1s6 & !dispin) | (nd1s6 & dispin) ;
        wire disp6 = dispin ^ (ndos6 | pdos6) ;
        wire compls4 = (pd1s4 & !disp6) | (nd1s4 & disp6) ;
        assign dispout = disp6 ^ (ndos4 | pdos4) ;
        assign dataout = {(jo ^ compls4), (ho ^ compls4),
                          (go ^ compls4), (fo ^ compls4),
                          (io ^ compls6), (eo ^ compls6),
                          (d0 ^ compls6), (co ^ compls6),
                          (bo ^ compls6), (ao ^ compls6)} ;
endmodule
```

9.1.14. FlipFlopT.sv

```
/**********************************************************************************
*Name:
* FlipFlopT.v
*Description:
* This module is the behavioral description for a T flip flop.
*Inputs:
* clk: The input for the global clock.
* reset: The global reset signal.
* enable: Disable o enable the register.
* Data_Input: The data input.
*Outputs:
* Data_Output: The register data output.
*Version:
* 1.0
*Author:
* Miguel Mihail Hoil Loria.
*Date:
* 18/02/2017
**********************************************************************************/
module FlipFlopT
  #(
    parameter WORD_LENGTH = 8
  )
  (
    // Input Ports
    input clk,
    input reset,
    input enable,
    input [WORD_LENGTH-1 : 0] Data_Input,
    // Output Ports
    output [WORD_LENGTH-1 : 0] Data_Output
  );
  reg [WORD_LENGTH-1 : 0] Data_reg;
  always@(posedge clk or negedge reset) begin
    if(reset == 1'b0)
      Data_reg <= 0;
    else if(enable == 1'b1)
      Data_reg <= Data_Input ^ Data_Output;
  end
  assign Data_Output = Data_reg;
endmodule
```

9.1.15. LFSR.sv

```
/**********************************************************************************
* Name:
* LFSR.v
* Author:
* Cesar Limones 2016
*Description:
* This module is a linear feedback shift
* register that will generate parallel data to be sent by
* the transmitter when BIST mode is enabled. It has the
* initial value fixed to a comma symbol.
/**********************************************************************************/
module LFSR #( parameter seed = 10'b11_1111_1100 )
  (
    //------------Inputs--------
    input rst,
    input clk,
    input lfsr_en,
    //------------Outputs--------
    output [8 : 0] lfsr_parallel_out
  );
  //------------Internal Variables--------
  wire linear_feedback;
  wire linear_feedback_i;
```

```
  reg [8 : 0] lfsr_data_out_reg;
  reg [9 : 0] lfsr_parallel_out_reg;
  //-------------Code Starts Here-------
  assign linear_feedback_i = lfsr_parallel_out_reg[5] ^
    lfsr_parallel_out_reg[9];
  assign linear_feedback = lfsr_parallel_out_reg[4] ^ linear_feedback_i;
  assign lfsr_parallel_out = lfsr_data_out_reg;
  always@(posedge clk or negedge rst) begin
    if(rst == 1'b0) // active low reset
      lfsr_parallel_out_reg <= seed;
    else if (lfsr_en) begin
      lfsr_parallel_out_reg <= {lfsr_parallel_out_reg[8 : 0],
                                linear_feedback};
    end
  end
  always@(lfsr_parallel_out_reg) begin
    if (lfsr_parallel_out_reg[9 : 0] == seed)
      lfsr_data_out_reg = lfsr_parallel_out_reg[8 : 0];
    else //If there is no comma on the data send a zero on the k bit.
      lfsr_data_out_reg = {1'b0, lfsr_parallel_out_reg[7 : 0]};
  end
endmodule
```

9.1.16. Mux2to1.sv

```
/**********************************************************************************************
******
*Description:
*    This module is the behavioral description for a parameterized 2 to 1 Multiplexer.
*Inputs:
*    Selector: The signal to choose the input to have in the output.
*    Data_0: The signal to choose with LOW level logic.
*    Data_1: The signal to choose with HIGH level logic.
*Outputs:
*   Mux_Output: The output chosen according to "Selector".
*Version:
*    2.0
*Author:
*    Miguel Mihail Hoil Loria.
*Date:
*    23/03/2017
**********************************************************************************************
*****/

module Mux2to1
#(
    parameter WORD_LENGTH = 8
)
(
    //Input Ports
    input Selector,
    input [WORD_LENGTH-1 : 0] Data_0,
    input [WORD_LENGTH-1 : 0] Data_1,

    //Output Ports
    output [WORD_LENGTH-1 : 0] Mux_Output
);

wire [WORD_LENGTH-1 : 0] Sel = {WORD_LENGTH{Selector}};
assign Mux_Output = ~Sel & Data_0 | Sel & Data_1;

endmodule
```

9.1.17. Register.sv

```
/////////////////////////////////////////////////////////////////////
* Name:
* Register.v
* Description:
* This module is a register with parameter.
* Inputs:
* clk: Clock signal
* reset: reset signal
* Data_Input: Data to lache data
* Outputs:
* Mux_Output: Data to provide lached data
* Versi□n:
* 1.0
* Author:
* Jos□ Luis Pizano Escalante
* Fecha:
* 07/02/2013
/////////////////////////////////////////////////////////////////////*/
module Register
  #(
    parameter WORD_LENGTH = 1
  )(
    // Input Ports
    input clk,
    input reset,
    input enable,
    input [WORD_LENGTH-1 : 0] Data_Input,
    // Output Ports
    output [WORD_LENGTH-1 : 0] Data_Output
  );
  reg [WORD_LENGTH-1 : 0] Data_reg;
  //08-10-17 "or negedge reset"
  always@(posedge clk or negedge reset) begin
    if(~reset) Data_reg <= 0;
    else begin
      if (enable)Data_reg <= Data_Input;
      else Data_reg <= Data_reg;
    end
  end
  assign Data_Output = Data_reg;
endmodule
```

9.1.18. Registro.sv

```
/////////////////////////////////////////////////////////////////////////////////////////////
******
*Description:
*    This module is the modified register from Moodle taught in class.
*Inputs:
*    clk: The input for the global clock.
*    reset: The global reset signal.
*    enable: Disable o enable the register.
*    Data_Input: The data input.
*Outputs:
*    Data_Output: The register data output .
*Version:
*    2.0
*Author:
*    Miguel Mihail Hoil Loria.
*Date:
*    06/02/2017
/////////////////////////////////////////////////////////////////////////////////////////////
*****/

module Registro
#(
```

```
    parameter WORD_LENGTH = 8
)

(
    // Input Ports
    input clk,
    input reset,
    input enable,
    input [WORD_LENGTH-1 : 0] Data_Input,

    // Output Ports
    output [WORD_LENGTH-1 : 0] Data_Output
);

reg  [WORD_LENGTH-1 : 0] Data_reg;

always@(posedge clk or negedge reset) begin
    if(reset == 1'b0)
        Data_reg <= 0;
    else if(enable == 1'b1)
        Data_reg <= Data_Input;
end

assign Data_Output = Data_reg;

endmodule
```

9.1.19. RegistroSyncRst.sv

```
/**********************************************************************************
*Name:
* RegistroSyncRst.v
*Description:
* This module is the modified register from Moodle taught in class.
*Inputs:
* clk: The input for the global clock.
* reset: The global reset signal.
* sync_reset: Reset the register on a clock transition.
* enable: Disable o enable the register.
*sync_reset
* Data_Input: The data input.
*Outputs:
* Data_Output: The register data output.
*Version:
* 1.0
*Author:
* Miguel Mihail Hoil Loria.
*Date:
* 28/02/2017
**********************************************************************************/
module RegistroSyncRst #(
  parameter WORD_LENGTH = 8
)
  (
    // Input Ports
    input clk,
    input reset,
    input sync_reset,
    input enable,
    input [WORD_LENGTH-1 : 0] Data_Input,
    // Output Ports
    output [WORD_LENGTH-1 : 0] Data_Output
  );
  reg [WORD_LENGTH-1 : 0] Data_reg;
  always@(posedge clk or negedge reset) begin
    if(reset == 1'b0)
      Data_reg <= 0;
    else if(enable == 1'b1)
      begin
        if(sync_reset == 1'b1)
```

```
          Data_reg <= 0;
        else
          Data_reg <= Data_Input;
      end
  end
  assign Data_Output = Data_reg;
endmodule
```

9.1.20. SERDES_chip.sv

```
/*************************************************************************
*Description:
* This module instatiates the SerDes core and connect the ports to their
* correspondig pad.
*Version:
* 1.0
*Author:
* Miguel Mihail Hoil Loria.
*Date:
* 07/11/2017
*************************************************************************/
module SERDES_chip
  (
    //------------Inputs--------
    input VDD, VSS,
    input DVDD, DVSS,
    input rst,
    input clk,
    input rxa_in_p,
    input rxa_in_n,
    input [7 : 0] config_in,
    input [8 : 0] txd_data_in,
    input test_en,
    //------------Outputs--------
    output [8 : 0] digital_out,
    output txa_data_out_p,
    output txa_data_out_n,
    output txd_data_out,
    output tx_frame_start,
    output c_data_valid
  );
  wire VDD_wire, VSS_wire, DVDD_wire, DVSS_wire;
  wire rst_wire, clk_wire;
  wire rxa_in_p_wire, rxa_in_n_wire;
  wire txa_data_out_p_wire, txa_data_out_n_wire;
  wire test_en_wire, txd_data_out_wire, tx_frame_start_wire,
  c_data_valid_wire;
  wire [7 : 0] config_in_wire;
  wire [8 : 0] txd_data_in_wire, digital_out_wire;
  SERDESv2
  SERDES_core
  (
    //------------Inputs--------
    .reset(rst_wire),
    .clk(clk_wire),
    .rxa_in_p(rxa_in_p_wire),
    .rxa_in_n(rxa_in_n_wire),
    .config_in(config_in_wire),
    .txd_data_in(txd_data_in_wire),
    .test_en(test_en_wire),
    //------------Outputs--------
    .digital_out(digital_out_wire),
    .txa_data_out_p(txa_data_out_p_wire),
    .txa_data_out_n(txa_data_out_n_wire),
    .txd_data_out(txd_data_out_wire),
    .tx_frame_start(tx_frame_start_wire),
    .c_data_valid(c_data_valid_wire)
  );
  assign VDD_wire = VDD;
  assign VSS_wire = VSS;
```

```
  assign DVDD_wire = DVDD;
  assign DVSS_wire = DVSS;
  //------------ POWER ------------
  // core power ring
  PVDD pad_VDD (.VDD(VDD_wire));
  PVSS pad_VSS (.VSS(VSS_wire));
  // IO power ring
  PDVDD pad_DVDD (.DVDD(DVDD_wire));
  PDVSS pad_DVSS (.DVSS(DVSS_wire));
  //------------ Input PADS ------------
  PICSD pad_rst(.P(rst), .IE(1'b1), .PD(1'b1), .Y(rst_wire));
  PIC pad_clk(.P(clk), .IE(1'b1), .Y(clk_wire) );
  PIC pad_rxa_in_p(.P(rxa_in_p), .IE(1'b1), .Y(rxa_in_p_wire) );
  PIC pad_rxa_in_n(.P(rxa_in_n), .IE(1'b1), .Y(rxa_in_n_wire) );
  PIC pad_config_in0(.P(config_in[0]), .IE(1'b1), .Y(config_in_wire[0]));
  PIC pad_config_in1(.P(config_in[1]), .IE(1'b1), .Y(config_in_wire[1]));
  PIC pad_config_in2(.P(config_in[2]), .IE(1'b1), .Y(config_in_wire[2]));
  PIC pad_config_in3(.P(config_in[3]), .IE(1'b1), .Y(config_in_wire[3]));
  PIC pad_config_in4(.P(config_in[4]), .IE(1'b1), .Y(config_in_wire[4]));
  PIC pad_config_in5(.P(config_in[5]), .IE(1'b1), .Y(config_in_wire[5]));
  PIC pad_config_in6(.P(config_in[6]), .IE(1'b1), .Y(config_in_wire[6]));
  PIC pad_config_in7(.P(config_in[7]), .IE(1'b1), .Y(config_in_wire[7]));
  PIC pad_txd_data_in0(.P(txd_data_in[0]), .IE(1'b1), .Y(txd_data_in_wire[0]));
  PIC pad_txd_data_in1(.P(txd_data_in[1]), .IE(1'b1), .Y(txd_data_in_wire[1]));
  PIC pad_txd_data_in2(.P(txd_data_in[2]), .IE(1'b1), .Y(txd_data_in_wire[2]));
  PIC pad_txd_data_in3(.P(txd_data_in[3]), .IE(1'b1), .Y(txd_data_in_wire[3]));
  PIC pad_txd_data_in4(.P(txd_data_in[4]), .IE(1'b1), .Y(txd_data_in_wire[4]));
  PIC pad_txd_data_in5(.P(txd_data_in[5]), .IE(1'b1), .Y(txd_data_in_wire[5]));
  PIC pad_txd_data_in6(.P(txd_data_in[6]), .IE(1'b1), .Y(txd_data_in_wire[6]));
  PIC pad_txd_data_in7(.P(txd_data_in[7]), .IE(1'b1), .Y(txd_data_in_wire[7]));
  PIC pad_txd_data_in8(.P(txd_data_in[8]), .IE(1'b1), .Y(txd_data_in_wire[8]));
  PIC pad_test_en(.P(test_en), .IE(1'b1), .Y(test_en_wire) );
  //------------ Output PADS ------------
  POC4C pad_txa_data_out_p(.A(txa_data_out_p_wire), .P(txa_data_out_p) );
  POC4C pad_txa_data_out_n(.A(txa_data_out_n_wire), .P(txa_data_out_n) );
  POC4C pad_txd_data_out(.A(txd_data_out_wire), .P(txd_data_out) );
  POC4C pad_tx_frame_start(.A(tx_frame_start_wire), .P(tx_frame_start) );
  POC4C pad_c_data_valid (.A(c_data_valid_wire), .P(c_data_valid) );
  POC4C pad_digital_out0(.A(digital_out_wire[0]), .P(digital_out[0]));
  POC4C pad_digital_out1(.A(digital_out_wire[1]), .P(digital_out[1]));
  POC4C pad_digital_out2(.A(digital_out_wire[2]), .P(digital_out[2]));
  POC4C pad_digital_out3(.A(digital_out_wire[3]), .P(digital_out[3]));
  POC4C pad_digital_out4(.A(digital_out_wire[4]), .P(digital_out[4]));
  POC4C pad_digital_out5(.A(digital_out_wire[5]), .P(digital_out[5]));
  POC4C pad_digital_out6(.A(digital_out_wire[6]), .P(digital_out[6]));
  POC4C pad_digital_out7(.A(digital_out_wire[7]), .P(digital_out[7]));
  POC4C pad_digital_out8(.A(digital_out_wire[8]), .P(digital_out[8]));
  //------------ CORNERS ------------
  PCORNER sw_pcorner ();
  PCORNER se_pcorner ();
  PCORNER nw_pcorner ();
  PCORNER ne_pcorner ();
endmodule
```

9.1.21. SERDESv2.sv

```
////////////////////////////////////////////////////////////////
//
// Module: SerDes
//
// Description: This block is the top level wrapper of the
// SerDes system, it contains the digital and analog transmitter,
// the digital and analog receiver, the test modules and the
// clock divider.
//
////////////////////////////////////////////////////////////////
module SERDESv2 #(
  parameter WIDTH = 9,
  parameter DECODED_COMMA = 9'h1FC,
  parameter ENCODED_COMMA = 10'h07C
```

```
)
  (
    //------------Inputs--------
    input reset, //reset
    input clk, //main system clock
    input rxa_in_p,//external Positive input for Analog Receiver
    input rxa_in_n, //external Negative input for Analog Receiver
    input [7 : 0] config_in, //Analog transmitter configuration
    input [WIDTH - 1 : 0] txd_data_in, //external digital transmission input
    input test_en,
    //------------Outputs--------
    output [WIDTH - 1 : 0] digital_out,
    output txa_data_out_p,
    output txa_data_out_n,
    output txd_data_out,
    output tx_frame_start,
    output c_data_valid
  );
  //------------Internal Variables--------
  wire rxa_out_p_wire;
  wire rxa_out_n_wire;
  wire txa_input_muxout_wire;
  wire [7 : 0] txa_config_wire;
  wire txa_data_out_p_wire;
  wire txa_data_out_n_wire;
  wire [3 : 0] clks_in_wire;
  wire [WIDTH - 1 : 0] txd_input_muxout_wire;
  wire txd_data_out_wire;
  wire frame_wire, frame_regOut_wire;
  wire rxd_input_muxout_wire;
  wire [7:0] rxd_data_out_wire;
  wire Ko_wire;
  wire code_err_wire;
  wire disp_err_wire;
  wire dispout_wire;
  wire data_valid_wire;
  wire [WIDTH - 1 : 0] rxd_output_muxout_wire;
  //------------Module instantiation --------
  ////////////////////////
  // Analog modules //
  ////////////////////////
  //Analog receiver
  Analog_RX
  analog_receiver_unit(
//CESAR    .\subc! (1'b0) ,
    // INPUTS
    .rx_in(rxa_in_p),
    .rx_inb(rxa_in_n),
    // OUTPUTS
    .rx_out(rxa_out_p_wire),
    .rx_outb(rxa_out_n_wire)
  );
  //Analog Transmitter
  Analog_TX
  analog_transmitter_unit(
    // INPUTS
    .txa_data_in(txa_input_muxout_wire),
    .tx_config(txa_config_wire),
    // OUTPUTS
    .txa_data_out_p(txa_data_out_p_wire),
    .txa_data_out_n(txa_data_out_n_wire)
  );
  ////////////////////////
  // Test modules //
  ////////////////////////
  test_modules_v2
  #(
    .DECODED_COMMA(DECODED_COMMA)
  )
  test_modules_unit
  (
```

```
  //------------INPUTS--------
  .rst(reset),
  .ser_clk(clks_in_wire[0]),
  .test_en(test_en),
  .lfsr_en(frame_regOut_wire),
  .config_in(config_in),
  /* RXA OUT */
  .rxa_out(rxa_out_p_wire),
  .rxa_out_n(rxa_out_n_wire),
  /* RXD OUT */
  .rxd_data_out({Ko_wire, rxd_data_out_wire}),
  .c_data_valid(data_valid_wire),
  .code_err(code_err_wire),
  .disp_err(disp_err_wire),
  .dispout(dispout_wire),
  /* TXD */
  //-INPUT-
  .txd_data_in(txd_data_in),
  //-OUTPUT-
  .txd_data_out(txd_data_out_wire),
  //------------OUTPUTS--------
  .rxd_input_muxout(rxd_input_muxout_wire),
  .txd_input_muxout(txd_input_muxout_wire),
  .txa_input_muxout(txa_input_muxout_wire),
  .rxd_output_muxout(rxd_output_muxout_wire),
  .tx_config(txa_config_wire)
);
////////////////////////
// Digital modules //
////////////////////////
clock_divider
clock_divider_unit(reset, clk, clks_in_wire);
//Digital receiver
digitalRX
#(
  .ENCODED_COMMA(ENCODED_COMMA)
)
DigitalRX_Unit
(
  //Inputs
  .rst(reset),
  .a_rx(rxd_input_muxout_wire),
  .disparity_d(1'b0),
  //Outputs
  .dataout(rxd_data_out_wire),
  .dispout(dispout_wire),
  .code_err(code_err_wire),
  .disp_err(disp_err_wire),
  .data_valid_pipe(data_valid_wire),
  .clk_4f(),
  .ko(Ko_wire),
  .clks_in(clks_in_wire)
);
//Digital Transmitter
Serializer
DigitalTX_Unit
(
  //Inputs
  .clk(clks_in_wire[0]),
  .rst(reset),
  .data_in(txd_input_muxout_wire),
  //Outputs
  .data_out(txd_data_out_wire),
  .frame(frame_wire)
);
// Signal frame_wire is early
Registro #(1)
txd_frame_register
(
  // Input Ports
  .clk(clks_in_wire[0]),
```

```
    .reset(reset),
    .enable(1'b1),
    .Data_Input(frame_wire),
    // Output Ports
    .Data_Output(frame_regOut_wire)
  );
  assign txa_data_out_p = txa_data_out_p_wire;
  assign txa_data_out_n = txa_data_out_n_wire;
  assign digital_out = rxd_output_muxout_wire;
  assign c_data_valid = data_valid_wire;
  assign txd_data_out = txd_data_out_wire;
  assign tx_frame_start = frame_regOut_wire;
endmodule
```

9.1.22. Serialization.sv

```
/*****************************************************
* Name:
* Serialization.v
* Description:
* This module takes a parallel input and delivers the same data in serial at
the frequency given by it´s clock
* Parameters:
* DATA_LENGTH: size of the word that is going to be serialized
* Inputs:
* Data_in
* clk: clock signal
* reset: reset signal (Active Low)
* rd_in: run disparity received from encoder
* Outputs:
* rd_out: bypasses rd_in when count = 1
* serial_out: This port will give the pulses of the information to
transmit
* start_frame: Gives a pulse 1 clock cycle before the data begins sending
when the
* Versión:
* 2.0
* Author:
* Efrain Arrambide
* Review by:
* Christian Aparicio Zuleta
* Date:
* 24/04/2017
*****************************************************/
module Serialization
  #(
    parameter DATA_LENGTH,
    parameter COUNT_LENGTH = CeilLog2(DATA_LENGTH)
  )(
    input [DATA_LENGTH-1:0] Data_in,
    input clk, reset, rd_in,
    output rd_out, serial_out, start_frame, nine
  );
  reg [DATA_LENGTH-1:0] par_reg;
  reg [COUNT_LENGTH-1:0] counter;
  reg disparity;
  wire ten, eight;
  always @(posedge clk or negedge reset) begin
    if (~reset) {counter, par_reg} <=
    {{COUNT_LENGTH{1'b0}},{DATA_LENGTH{1'b0}}};
    else begin
      if (!ten) {counter, par_reg} <= {counter +
        1'b1,1'b0,par_reg[DATA_LENGTH-1:1]};
      else {counter, par_reg} <= {{COUNT_LENGTH-1'b1{1'b0}},1'b1,Data_in};
      if (eight) disparity <= rd_in;
      else disparity <= 0;
    end
  end
  //Combinational logic
  Control_serial #(COUNT_LENGTH) Control_PISO(counter, eight, nine, ten);
```

```
  assign rd_out = disparity;
  assign serial_out = par_reg[0];
  assign start_frame = (ten)? 1'b1: 1'b0;
  /*************************************************************************/
  /*Log Function*/
  function integer CeilLog2;
    input integer data;
    integer i,result;
    begin
      for(i=0; 2**i < data; i=i+1)
        result = i + 1;
      CeilLog2 = result;
    end
  endfunction
  /*************************************************************************/
endmodule
```

9.1.23. Serializer.sv

```
/*************************************************************************
* Name:
* Serializer.v
* Description:
* Top module of a 10 bit serializer that uses an 8b10b encoder
* Inputs:
* clk: clock signal
* rst: reset signal
* data_in: data bus at the input
* Outputs:
* data_out: serial data out
* frame: indicates the start of a transaction
* Versión:
* 1.0
* Author:
* Christian Aparicio Zuleta
* Date:
* 04/13/2017
*************************************************************************/
module Serializer #(parameter DATA_LENGTH=10) (clk, rst, data_in, data_out,
                                               frame);
  input clk, rst;
  input [8:0] data_in;
  output data_out, frame;
  wire rd_out_encode, rd_in_encode, count_nine;
  wire [8:0] data_in_encode;
  wire [9:0] dataout;
  wire current_rd;
  //Instances by order (No .<port name needed>)
  encode encode1(data_in_encode, current_rd,dataout,rd_out_encode);
  Serialization #(DATA_LENGTH) Serialization1 (dataout, clk, rst,
                                               rd_out_encode,
rd_in_encode,data_out,frame,count_nine);
  Register #(10) Data_Input (clk, rst, count_nine, {data_in,rd_in_encode},
                             {data_in_encode,current_rd});
endmodule
```

9.1.24. SignalDriverS2.sv

```
////////////////////////////////////////////////////////////
// Module: signal_driver
//
// Author: Cesar Limones 2016
//
// Description: This module functionality is to gather all
// the signals from the other modules and drive them to
// their respective counterpart depending on the test mode
```

```
// that is being executed.
//////////////////////////////////////////////////////////////
/*********************************************************************************
*Name:
* SignalDriverS2.v
*Editor:
* Miguel Mihail Hoil Loria.
*Changes:
* 24/05/2017
* -Output port names changed:
* rxd_in_i > rxd_input_muxout
* txd_data_in_i > txd_input_muxout
* txa_data_in_i > txa_input_muxout
* digital_out > rxd_output_muxout
* -test_out is always rxa_out except for the BIST modes.
* -The default mode and the functional mode are the same.
* -The input c_data_valid is not required due to the output pin that
works the same.
* 09/06/2017
* -behavioral description changed to structural description
* 05/11/2017
* -bist_end and rxa_out are selected with two multiplexors, the
first chooses between them and the two
* num_errors MSBs, this output is the input for the multiplexor that
selects the digital receiver output.
* In both cases the selectors do not have any combinational logic.
*********************************************************************************/
module SignalDriverS2 #( parameter WIDTH = 9)
  (
    //------------Inputs--------
    input [2 : 0] mode, //config_in[2 : 0]
    input test_in, //config_in[3]
    input errors_en, //config_in[4]
    input test_out_muxsel, //config_in[5]
    /* RXA OUT */
    input rxa_out,
    /* RXD OUT */
    input [WIDTH - 1 : 0] rxd_data_out,
    input code_err,
    input disp_err,
    input dispout,
    /* TXD */
    input txd_data_out,
    input [ WIDTH - 1 : 0 ] txd_data_in,
    /* LFSR OUT */
    input [ WIDTH - 1 : 0 ] lfsr_parallel_out,
    /* Comparator OUT */
    input bist_end,
    input [5 : 0] num_errors,
    //------------Outputs--------
    output rxd_input_muxout, txa_input_muxout,
    output [WIDTH - 1 : 0] txd_input_muxout,
    output [WIDTH - 1 : 0] rxd_output_muxout
  );
  //------------Internal Variables--------
  wire rxd_input_muxout_wire, txa_input_muxout_wire;
  wire [WIDTH-1 : 0] txd_input_muxout_wire;
  wire [1 : 0] test_out_wire;
  wire [WIDTH-1 : 0] rxd_output_muxout_wire;
  wire rxd_input_bitZero_muxsel_wire, rxd_input_bitOne_muxsel_wire;
  wire rxd_muxout_A_wire, txd_input_bitZero_muxsel_wire;
  wire txd_input_bitOne_muxsel_wire, txa_input_muxsel_wire;
  wire [WIDTH-1 : 0] txd_muxout_A_wire;
  wire [WIDTH-1 : 0] rxd_muxinput_wire;
  wire mZero_inv, mOne_Xor_mTwo, mZeroInv_And_mTwo;
  //-------------Code Starts Here-------
  /* * * * rxd_input_mux * * * */
  Mux2to1 #( .WORD_LENGTH(1) ) RXD_input_A_MUX (
    //Input Ports
    .Selector(rxd_input_bitZero_muxsel_wire),
    .Data_0(rxa_out), .Data_1(txd_data_out),
```

```
    //Output Ports
    .Mux_Output(rxd_muxout_A_wire) );
  Mux2to1 #( .WORD_LENGTH(1) ) RXD_input_B_MUX (
    //Input Ports
    .Selector(rxd_input_bitOne_muxsel_wire),
    .Data_0(rxd_muxout_A_wire), .Data_1(test_in),
    //Output Ports
    .Mux_Output(rxd_input_muxout_wire) );
  /* * * * txd_input_mux * * * */
  Mux2to1 #( .WORD_LENGTH(WIDTH) ) TXD_input_A_MUX (
    //Input Ports
    .Selector(txd_input_bitZero_muxsel_wire),
    .Data_0(txd_data_in), .Data_1(rxd_data_out),
    //Output Ports
    .Mux_Output(txd_muxout_A_wire) );
  Mux2to1 #( .WORD_LENGTH(WIDTH) ) TXD_input_B_MUX (
    //Input Ports
    .Selector(txd_input_bitOne_muxsel_wire),
    .Data_0(txd_muxout_A_wire), .Data_1(lfsr_parallel_out),
    //Output Ports
    .Mux_Output(txd_input_muxout_wire) );
  /* * * * txa_input_mux * * * */
  Mux2to1 #( .WORD_LENGTH(1) ) TXA_input_MUX (
    //Input Ports
    .Selector(txa_input_muxsel_wire),
    .Data_0(txd_data_out), .Data_1(rxa_out),
    //Output Ports
    .Mux_Output(txa_input_muxout_wire) );
  /* * * * test_out_mux * * * */
  Mux2to1 #( .WORD_LENGTH(2) ) Test_out_MUX (
    //Input Ports
    .Selector(errors_en),
    .Data_0({rxa_out, bist_end}), .Data_1(num_errors[5 : 4]),
    //Output Ports
    .Mux_Output(test_out_wire) );
  /* * * * rxd_output_mux * * * */
  Mux2to1 #( .WORD_LENGTH(WIDTH) ) RXD_output_MUX (
    //Input Ports
    .Selector(test_out_muxsel),
    .Data_0(rxd_data_out), .Data_1(rxd_muxinput_wire),
    //Output Ports
    .Mux_Output(rxd_output_muxout_wire) );
  //-------------Wire assignations-------
  assign rxd_muxinput_wire = {disp_err, code_err, dispout, test_out_wire,
                              num_errors[3 : 0]};
  assign mZero_inv = !mode[0];
  assign mOne_Xor_mTwo = mode[1] ^ mode[2];
  assign mZeroInv_And_mTwo = mZero_inv & mode[2];
  // rxd input muxtiplexor 3-1 selectors
  assign rxd_input_bitZero_muxsel_wire = mZero_inv & mOne_Xor_mTwo;
  assign rxd_input_bitOne_muxsel_wire = mode[0] & mOne_Xor_mTwo;
  // txd input muxtiplexor 3-1 selectors
  assign txd_input_bitZero_muxsel_wire = mode[0] & !mode[1];
  assign txd_input_bitOne_muxsel_wire = mZeroInv_And_mTwo;
  // txa input muxtiplexor 2-1 selector
  assign txa_input_muxsel_wire = &mode;
  // Assign output ports
  assign rxd_output_muxout = rxd_output_muxout_wire;
  assign rxd_input_muxout = rxd_input_muxout_wire;
  assign txd_input_muxout = txd_input_muxout_wire;
  assign txa_input_muxout = txa_input_muxout_wire;
endmodule
```

9.1.25. simple_dual_port_ram_single_clock.sv

```
// Quartus II Verilog Template
// Simple Dual Port RAM with separate read/write addresses and
// single read/write clock
/*
* 30/05/17
* -The read is now asynchronous
*/
module simple_dual_port_ram_single_clock
  #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=6)
  (
    input [(DATA_WIDTH-1) : 0] data,
    input [(ADDR_WIDTH-1) : 0] read_addr, write_addr,
    input we, clk,
    output [(DATA_WIDTH-1) : 0] q
  );
  // Declare the RAM variable
  reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];
  always@(posedge clk)
    begin
      // Write
      if (we)
        ram[write_addr] <= data;
    end
  assign q = ram[read_addr];
endmodule
```

9.1.26. test_modules_v2.sv

```
////////////////////////////////////////////////////////////////
// Module: test_modules
// Author: Cesar Limones 2016
// Description: This block contains all the testing modules
// for a SerDes. It contains the LFSR, a comparator
// and a signal driver.
////////////////////////////////////////////////////////////////
/*****************************************************************************
*Name:
* test_modules_v2.v
*Editor:
* Miguel Mihail Hoil Loria.
*>Inputs:
* ser_clk: The input for the global clock.
* rst: The global reset signal.
* test_en: enables the test operating modes
* lfsr_en: enables the lfsr, it should be the frame output from digital
transmitter
* config_in: analog transmitter configuration, in test mode the 5 less
significant bits
* are used for test configuration
* rxa_out: analog receiver positive output (RXA OUT)
* rxa_out_n: analog receiver negative output (RXA OUT)
* rxd_data_out: digital receiver output (RXD OUT)
* c_data_valid: indicates when the digital receiver output is valid (RXD
OUT)
* code_err: indicates an invalid 10 bit data received (RXD OUT)
* disp_err: indicates a running disparity error (RXD OUT)
* dispout: running disparity (RXD OUT)
* txd_data_in: digital transmitter input
* txd_data_out: digital transmitter output
*<Outputs:
* rxd_input_muxout: digital receiver real input
* txd_input_muxout: digital transmitter real input
* txa_input_muxout: analog transmitter real input
* rxd_output_muxout: SerDes output, can be the digital receiver output
or the comparator errors
* test_out: SerDes output for the analog receiver output (RXA OUT)
* tx_config: final analog transmitter configuration
```

```
*-Changes:
* 24/05/2017
* -Output port names changed:
* rxd_in_i > rxd_input_muxout
* txd_data_in_i > txd_input_muxout
* txa_data_in_i > txa_input_muxout
* digital_out > rxd_output_muxout
* 08/10/2017
* -Two registers were added to delay the LFSR data generation
allowing enough time for register
* the actual output from the LFSR
* 05/11/2017119
* -test_out output removed because now is multiplexed in the signal
driver in order to save one pin.
**********************************************************************/
module test_modules_v2 #( parameter WIDTH = 9, parameter DECODED_COMMA =
                          9'h1FC )
  (
    //------------Inputs--------
    input rst,
    input ser_clk,
    input test_en,
    input lfsr_en,
    input [ 7 : 0 ] config_in,
    /* RXA OUT */
    input rxa_out,
    input rxa_out_n,
    /* RXD OUT */
    input [WIDTH - 1 : 0] rxd_data_out,
    input c_data_valid,
    input code_err,
    input disp_err,
    input dispout,
    /* TXD */
    input [ WIDTH - 1 : 0 ] txd_data_in,
    input txd_data_out,
    //------------Outputs--------
    output rxd_input_muxout, txa_input_muxout,
    output [ WIDTH - 1 : 0 ] txd_input_muxout,
    output [ WIDTH - 1 : 0 ] rxd_output_muxout, //RXD OUTPUT
    output [ 7 : 0 ] tx_config );
  //------------Internal Variables--------
  wire bist_on_wire, txd_frame_start_wire, txd_frame_start_regOutA_wire;
  wire txd_frame_start_regOutB_wire, rxd_data_ready_wire, bist_end_wire;
  wire [ WIDTH - 1 : 0 ] lfsr_parallel_out_wire;
  wire [ 5 : 0 ] num_errors_wire;
  reg [2 : 0] mode_reg;
  reg test_in_reg, errors_en_reg, test_out_sel_reg;
  reg [7 : 0] tx_config_reg;
  //-------------Code Starts Here-------
  always@(test_en or config_in) begin
    if (test_en) begin
      mode_reg = config_in[2:0];
      test_in_reg = config_in[3];
      errors_en_reg = config_in[4];
      test_out_sel_reg = config_in[5];
      tx_config_reg = 8'b0;
    end
    else begin
      mode_reg = 3'b000;
      test_in_reg = 1'b0;
      errors_en_reg = 1'b0;
      test_out_sel_reg = 1'b0;
      tx_config_reg = config_in;
    end
  end
  //------------- Module instantiation -------
  //Buffers?
  Registro #(1) buffer_rxa_out_p_register (
    .clk(ser_clk), .reset(rst), .enable(1'b1), .Data_Input(rxa_out),
    .Data_Output() );
```

```
  Registro #(1) buffer_rxa_out_n_register (
    .clk(ser_clk), .reset(rst), .enable(1'b1), .Data_Input(rxa_out_n),
    .Data_Output() );
  //Signal driver
  SignalDriverS2 #(9)
  signal_driver_unit (
    // INPUTS
    .mode(mode_reg), .test_in(test_in_reg), .errors_en(errors_en_reg),
    .test_out_muxsel(test_out_sel_reg), .rxa_out(rxa_out),
    .rxd_data_out(rxd_data_out), .code_err(code_err), .disp_err(disp_err),
    .dispout(dispout), .txd_data_in(txd_data_in),
    .txd_data_out(txd_data_out), .lfsr_parallel_out(lfsr_parallel_out_wire),
    .bist_end(bist_end_wire), .num_errors(num_errors_wire),
    // OUTPUTS
    .rxd_input_muxout(rxd_input_muxout),.txd_input_muxout(txd_input_muxout),
    .txa_input_muxout(txa_input_muxout),
    .rxd_output_muxout(rxd_output_muxout) );
  // Register for delay the data generation and allow current data recording.
  Registro #(1) txd_frame_start_registerA (
    .clk(ser_clk), .reset(rst), .enable(bist_on_wire),
    .Data_Input(txd_frame_start_wire),
    .Data_Output(txd_frame_start_regOutA_wire) );
  Registro #(1) txd_frame_start_registerB (
    .clk(ser_clk), .reset(rst), .enable(bist_on_wire),
    .Data_Input(txd_frame_start_regOutA_wire),
    .Data_Output(txd_frame_start_regOutB_wire) );
  //Pseudo Alleatory pattern Generator LFSR
  LFSR #( .seed({1'b1,DECODED_COMMA}) )
  lfsr_unit (
    .rst(rst), .clk(ser_clk), .lfsr_en(txd_frame_start_regOutB_wire),
    .lfsr_parallel_out(lfsr_parallel_out_wire) );
  //Comparator
  ComparatorP #( .WIDTH(WIDTH), .COMMA(DECODED_COMMA) )
  comparator_unit (
    // INPUTS
    .clk(ser_clk),
    .rst(rst),
    .dataA_in(txd_input_muxout),
    .dataB_in(rxd_data_out),
    .lfsr_en(txd_frame_start_wire),
    .data_valid_pipe(rxd_data_ready_wire),
    // OUTPUTS
    .bist_end(bist_end_wire),
    .num_errors(num_errors_wire)
  );
  // The next two signals are important for recording and comparison function
  //in the comparator
  assign txd_frame_start_wire = lfsr_en & bist_on_wire; //This signal
  //permits check and store the incoming txd data.
  assign rxd_data_ready_wire = c_data_valid & bist_on_wire; //This signal
  //allows the comparation between the store data and actual rxd data.
  assign bist_on_wire = test_en & mode_reg[2] & !mode_reg[0];
  assign tx_config = tx_config_reg;
endmodule
```

9.1.27. TXA_FINAL.sv

```
/*******************************************************************************
*Name:
* TXA_FINAL.v
*Description:
* This module is the black box for the MACRO with the same name in the LEF
file.
*Version:
* 1.0
*Author:
* Miguel Mihail Hoil Loria.
*Date:
* 07/11/2017
```

```
*************************************************************************/
module TXA_FINAL
  (
    inout \sub! ,
    //------------Inputs--------
    input DATA,
    input ZA,
    input ZB,
    input ZC,
    input ZD,
    input AMP_CTRL_1,
    input AMP_CTRL_2,
    input PRE_CTRL_1,
    input PRE_CTRL_2,
    input TEST_DATA,
    input DATA_SELECTOR,
    //------------Outputs--------
    output TX,
    output TXBar,
    output DIRECT_DATA_TX,
    output DIRECT_DATA_TXBar
  );
endmodule
```

9.2. SerDes Testbench Codes

The following testbench codes are listed as included in the SerDes Env package.

9.2.1. serdesEnvPkg.sv

```
`include "uvm_macros.svh"
import uvm_pkg::*;

package serdesEnvPkg;
`include "tb/serdes_config.sv"
`include "tb/serdes_seq_item.sv"
`include "tb/serdes_tx_subscriber.sv"
`include "tb/serdes_rx_subscriber.sv"
`include "tb/serdes_sequence_base.sv"
`include "tb/serdes_parallel_loopback_sequence.sv"
`include "tb/serdes_serial_loopback_sequence.sv"
`include "tb/serdes_bist_serial_loopback_sequence.sv"
`include "tb/serdes_rxa_bypass_sequence.sv"
`include "tb/serdes_rxa_bypass_parallel_loopback_sequence.sv"
`include "tb/serdes_open_bist_sequence.sv"
`include "tb/serdes_rxa_output_analog_loopback_sequence.sv"
`include "tb/serdes_rx_sequencer.sv"
`include "tb/serdes_tx_sequencer.sv"
`include "tb/serdes_rx_driver.sv"
`include "tb/serdes_tx_driver.sv"
`include "tb/serdes_rx_monitor.sv"
`include "tb/serdes_tx_monitor.sv"
`include "tb/serdes_scoreboard.sv"
`include "tb/serdes_rx_agent.sv"
`include "tb/serdes_tx_agent.sv"
`include "tb/serdes_env.sv"
`include "tb/serdes_base_test.sv"
`include "tb/serdes_parallel_loopback_test.sv"
`include "tb/serdes_serial_loopback_test.sv"
`include "tb/serdes_bist_serial_loopback_test.sv"
`include "tb/serdes_rxa_bypass_test.sv"
`include "tb/serdes_rxa_bypass_parallel_loopback_test.sv"
`include "tb/serdes_open_bist_test.sv"
`include "tb/serdes_rxa_output_analog_loopback_test.sv"
endpackage: serdesEnvPkg
```

9.2.2. serdes_tb_top.sv

```
/* Autogenerated Code for serdes_tb_top.sv */
   `include "uvm_macros.svh"
   import uvm_pkg::*;
   `include "tb/serdesEnvPkg.sv"
   import serdesEnvPkg::*;
module serdes_tb_top;

   //-------------------------------------
   //clock and reset signal declaration
   //-------------------------------------
   bit clk;
   bit rst;

   SERDESv2 serdes_top(
      .reset(vif.rst),
      .clk(vif.clk),
      .rxa_in_p(vif.rxa_in_p),
      .rxa_in_n(vif.rxa_in_n),
      .config_in(vif.config_in),
      .txd_data_in(vif.txd_data_in),
      .test_en(vif.test_en),
      .digital_out(vif.digital_out),
      .txa_data_out_p(vif.txa_data_out_p),
      .txa_data_out_n(vif.txa_data_out_n),
      .txd_data_out(vif.txd_data_out),
      .tx_frame_start(vif.tx_frame_start),
      .c_data_valid(vif.c_data_valid)
  );
  serdes_if vif(clk,rst);
  assign vif.rxd_output_muxout_wire = serdes_top.rxd_output_muxout_wire;
  assign vif.txd_input_muxout_wire = serdes_top.txd_input_muxout_wire;
  assign vif.rxd_input_muxout_wire = serdes_top.rxd_input_muxout_wire;
  assign vif.clk_in_slow = serdes_top.clks_in_wire[0];
  assign vif.bist_end =  serdes_top.test_modules_unit.bist_end_wire;

   //-------------------------------------
   //clock generation
   //-------------------------------------
   always #10 clk = ~clk;

   //-------------------------------------
   //reset generation
   //-------------------------------------
   initial begin
      rst = 0;
      #10 rst = 1;
   end

   // Procedural block
   initial begin
      uvm_config_db#(virtual serdes_if )::set(uvm_root::get(),"*","vif",vif);
      // Enable wave dump
      $dumpfile("dump.vcd");
      $dumpvars;

      uvm_top.finish_on_completion = 1;

      // Calling test
      run_test();
   end


endmodule
```

9.2.3. serdes_if.sv

```
/* Autogenerated Code for serdes_if.sv */

interface serdes_if(input logic clk, rst);
   logic rxa_in_p;
   logic rxa_in_n;
   logic [7:0] config_in;
   logic [8:0] txd_data_in;
   logic test_en;
   logic [8:0] digital_out;
   logic txa_data_out_p;
   logic txa_data_out_n;
   logic txd_data_out;
   logic tx_frame_start;
   logic c_data_valid;
   logic [8:0] rxd_output_muxout_wire;
   logic [8:0] txd_input_muxout_wire;
   logic rxd_input_muxout_wire;
   logic clk_in_slow;
   logic rx_transmit;
   logic tx_transmit;
   logic bist_end;
endinterface: serdes_if
```

9.2.4. serdes_config.sv

```
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_config extends uvm_object;

   //-------------------------------------
   // UVM automation macros for general components
   //-------------------------------------
   `uvm_object_utils(serdes_config)


   uvm_active_passive_enum  is_active = UVM_ACTIVE;
   bit                      has_subscriber = 0;
   bit [2:0]                test_mode;
   //-------------------------------------
   // Constructor
   //-------------------------------------
  function new(string name = "serdes_config");
    super.new(name);
   endfunction : new


endclass : serdes_config
```

9.2.5. serdes_seq_item.sv

```
/* Autogenerated Code for serdes_seq_item.sv */

`include "uvm_macros.svh"
import uvm_pkg::*;
class serdes_seq_item extends uvm_sequence_item;
  rand bit       rxa_in_p;
  rand bit       rxa_in_n;
  rand bit [8:0] txd_data_in;
  rand bit       test_en;
  rand bit [7:0] config_in;
       bit [8:0] digital_out;
       bit       txa_data_out_p;
       bit       txa_data_out_n;
```

```
        bit       txd_data_out;
        bit       tx_frame_start;
        bit       c_data_valid;
  rand bit [9:0] data_transmit;
        bit       rx_transmit;
        bit       tx_transmit;
        bit [8:0] txd_input_muxout_wire;
        bit [8:0] rxd_output_muxout_wire;
        bit       rxd_input_muxout_wire;
        bit       bist_end;

    `uvm_object_utils_begin(serdes_seq_item)
    `uvm_field_int(rxa_in_p,UVM_ALL_ON)
    `uvm_field_int(rxa_in_n,UVM_ALL_ON)
    `uvm_field_int(txd_data_in,UVM_ALL_ON)
    `uvm_field_int(test_en,UVM_ALL_ON)
    `uvm_field_int(config_in,UVM_ALL_ON)
    `uvm_field_int(data_transmit,UVM_ALL_ON)
    `uvm_object_utils_end


    //-------------------------------------
    // Constructor
    //-------------------------------------
    function new(string name = "serdes_seq_item");
       super.new(name);
    endfunction : new

    //-------------------------------------
    // Optional constraints code
    //-------------------------------------
   constraint rxa_in_n_c { rxa_in_n == !rxa_in_p; };

endclass : serdes_seq_item
```

9.2.6. serdes_tx_subscriber.sv

```
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_tx_subscriber extends uvm_subscriber#(serdes_seq_item);
    //-------------------------------------
    // UVM automation macros for general components
    //-------------------------------------
    `uvm_component_utils( serdes_tx_subscriber )

    serdes_seq_item seq;

    covergroup tx_cg;
        comma_cp:  coverpoint seq.txd_data_in
                {
                bins comma = {9'b1_1111_1100};
                }
        test_mode_cp: coverpoint seq.config_in[2:0];
    endgroup : tx_cg

    //-------------------------------------
    // Constructor
    //-------------------------------------
    function new(string name, uvm_component parent);
       super.new(name, parent);
       tx_cg = new;
    endfunction : new

    function void write(serdes_seq_item t);
       seq = t;
     $display("tx obtained coverage");
       tx_cg.sample();
```

```
   endfunction: write

endclass : serdes_tx_subscriber
```

9.2.7. serdes_rx_subscriber.sv

```
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rx_subscriber extends uvm_subscriber#( serdes_seq_item );
   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils( serdes_rx_subscriber )

   serdes_seq_item seq;

   covergroup rx_cg;
        bist_end_cp: coverpoint seq.bist_end;
        test_mode_cp: coverpoint seq.config_in[2:0];
        config3_cp: coverpoint seq.config_in[3];
        comma_cp:   coverpoint seq.rxd_output_muxout_wire { bins comma = {9'h1fc};}
        config4_cp: coverpoint seq.config_in[4];
        config5_cp: coverpoint seq.config_in[5];
        num_error_cp: coverpoint seq.digital_out[5:0] iff (seq.config_in[5:4] == 2'b11);
   endgroup : rx_cg

   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
      rx_cg = new;
   endfunction : new


   function void write( serdes_seq_item t );
      seq = t;
      $display("rx obtained by coverage");
      rx_cg.sample();
   endfunction: write

endclass: serdes_rx_subscriber
```

9.2.8. serdes_sequence_base.sv

```
/* Autogenerated Code for serdes_base_sequence.sv */
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_sequence_base extends uvm_sequence #(serdes_seq_item);

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
  `uvm_object_utils( serdes_sequence_base );

   //------------------------------------
   // Constructor
   //------------------------------------
   function new( string name = "serdes_sequence_base" );
      super.new(name);
   endfunction : new

   //------------------------------------
   // Body task
   //------------------------------------
   virtual task body();
```

```
    repeat(20) begin
         req = serdes_seq_item::type_id::create("req");
         start_item(req);
         assert (req.randomize());
         finish_item(req);
    end
   endtask : body


endclass : serdes_base_sequence
endclass
```

9.2.9. serdes_parallel_loopback_sequence.sv

```
//------------------------------------------------------------------------
//                   UVM FRAMEWORK
//------------------------------------------------------------------------
// Project         : SERDES
// Unit            : serdes sequence mode 1
// File            : serdes_parallel_loopback_sequence.svh
//------------------------------------------------------------------------
// Creation Date   :
//------------------------------------------------------------------------
// Description: this class defines the base top level sequence used
//    for the serdes simulations. It is used as the top
//    level sequence for test_top. It is extended to create
//    specific test scenarios.
//
//------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_parallel_loopback_sequence extends uvm_sequence #(serdes_seq_item); //poner
como default uvm_sequence_item

  `uvm_object_utils( serdes_parallel_loopback_sequence );

  //---------------------------------------
  //Constructor
  //---------------------------------------
  function new( string name = "serdes_parallel_loopback_sequence" );
     super.new( name );
  endfunction

  //Creating the sequence
  virtual task body();
    repeat (2) begin
      req=serdes_seq_item::type_id::create("req");
      req.rx_transmit=0;
      req.config_in=8'b0000_0001;
      req.data_transmit=10'h000;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0001;
      req.data_transmit=10'b11_1000_0011;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0001;
      req.data_transmit=10'h0b9;
```

```
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0001;
      req.data_transmit=10'h0ae;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0001;
      req.data_transmit=10'h0ad;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0001;
      req.data_transmit=10'h000;
      req.rx_transmit=0;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end

  endtask


endclass
```

9.2.10. serdes_serial_loopback_sequence.sv

```
//------------------------------------------------------------------------
//                    UVM FRAMEWORK
//------------------------------------------------------------------------
// Project         : SERDES
// Unit            : serdes sequence mode 2
// File            : serdes_serial_loopback_sequence.svh
//------------------------------------------------------------------------
// Creation Date   :
//------------------------------------------------------------------------
// Description: this class defines the base top level sequence used
//    for the serdes simulations. It is used as the top
//    level sequence for test_top. It is extended to create
//    specific test scenarios.
//
//------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_serial_loopback_sequence extends uvm_sequence #(serdes_seq_item); //poner como
default uvm_sequence_item

  `uvm_object_utils( serdes_serial_loopback_sequence );

  //---------------------------------------
  //Constructor
  //---------------------------------------
  function new( string name = "serdes_serial_loopback_sequence" );
     super.new( name );
```

```
  endfunction

  //Creating the sequence
  virtual task body();
    repeat (2) begin
      req=serdes_seq_item::type_id::create("req");
      req.tx_transmit=0;
      req.config_in=8'b0000_0010;
      req.txd_data_in=9'h000;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      //req.randomize();
      //send_request(req);
      req.config_in=8'b0000_0010;
      req.txd_data_in=9'b1_1111_1100; // Sending the first comma to start
      req.tx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0010;
      req.txd_data_in=9'h0fa;
      req.tx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0010;
      req.txd_data_in=9'h0fc;
      req.tx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0010;
      req.txd_data_in=9'h0fe;
      req.tx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in=8'b0000_0010;
      req.txd_data_in=9'b1_1111_1100;//Sending a second commma to end
      req.tx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
  endtask


endclass
```

9.2.11. serdes_bist_serial_loopback_sequence.sv

```
//------------------------------------------------------------------------
//                    UVM FRAMEWORK
//------------------------------------------------------------------------
// Project         : SERDES
// Unit            : serdes sequence mode 4
// File            : serdes_bist_serial_loopback_sequence.svh
//------------------------------------------------------------------------
// Creation Date   :
//------------------------------------------------------------------------
// Description: this class defines the base top level sequence used
//    for the serdes simulations. It is used as the top
//    level sequence for test_top. It is extended to create
//    specific test scenarios.
//
//------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_bist_serial_loopback_sequence extends uvm_sequence #(serdes_seq_item); //poner
como default uvm_sequence_item

  `uvm_object_utils( serdes_bist_serial_loopback_sequence );

  //---------------------------------------
  //Constructor
  //---------------------------------------
  function new( string name = "serdes_bist_serial_loopback_sequence" );
     super.new( name );
  endfunction

  //Creating the sequence
  virtual task body();
    repeat (2) begin
      req=serdes_seq_item::type_id::create("req");
      req.rx_transmit=1;
      req.config_in[2:0]=3'b100;
      req.config_in[5]=1'b1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (100) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b100;
      req.config_in[5]=1'b1;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
  endtask


endclass
```

9.2.12. serdes_rxa_bypass_sequence.sv

```
//------------------------------------------------------------------------
//                    UVM FRAMEWORK
//------------------------------------------------------------------------
// Project          : SERDES
// Unit             : serdes sequence mode 3
// File             : serdes_rxa_bypass_sequence.svh
//------------------------------------------------------------------------
// Creation Date    :
//------------------------------------------------------------------------
// Description: this class defines the base top level sequence used
//    for the serdes simulations. It is used as the top
//    level sequence for test_top. It is extended to create
//    specific test scenarios.
//
//------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rxa_bypass_sequence extends uvm_sequence #(serdes_seq_item);

  `uvm_object_utils( serdes_rxa_bypass_sequence );

  //---------------------------------------
  //Constructor
  //---------------------------------------
  function new( string name = "serdes_rxa_bypass_sequence" );
     super.new( name );
  endfunction

  //Creating the sequence
  virtual task body();
    repeat (2) begin
      req=serdes_seq_item::type_id::create("req");
      req.rx_transmit=0;
      req.config_in[2:0]=3'b011;
      req.data_transmit=10'h000;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b011;
      req.data_transmit=10'b11_1000_0011;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b011;
      req.data_transmit=10'h0b9;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b011;
      req.data_transmit=10'h0ae;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
```

```
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b011;
      req.data_transmit=10'h0ad;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b011;
      req.data_transmit=10'h000;
      req.rx_transmit=0;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end

  endtask


endclass
```

9.2.13. serdes_rxa_bypass_parallel_loopback_sequence.sv

```
//-----------------------------------------------------------------------
//                    UVM FRAMEWORK
//-----------------------------------------------------------------------
// Project         : SERDES
// Unit            : serdes sequence mode 3
// File            : serdes_rxa_bypass_sequence.svh
//-----------------------------------------------------------------------
// Creation Date   :
//-----------------------------------------------------------------------
// Description: this class defines the base top level sequence used
//    for the serdes simulations. It is used as the top
//    level sequence for test_top. It is extended to create
//    specific test scenarios.
//
//-----------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rxa_bypass_parallel_loopback_sequence extends uvm_sequence #(serdes_seq_item);

  `uvm_object_utils( serdes_rxa_bypass_parallel_loopback_sequence );

  //---------------------------------------
  //Constructor
  //---------------------------------------
  function new( string name = "serdes_rxa_bypass_parallel_loopback_sequence" );
     super.new( name );
  endfunction

  //Creating the sequence
  virtual task body();
    repeat (2) begin
      req=serdes_seq_item::type_id::create("req");
      req.rx_transmit=0;
      req.config_in[2:0]=3'b101;
      req.data_transmit=10'h000;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b101;
```

```
      req.data_transmit=10'b11_1000_0011;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b101;
      req.data_transmit=10'h0b9;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b101;
      req.data_transmit=10'h0ae;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b101;
      req.data_transmit=10'h0ad;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b101;
      req.data_transmit=10'h000;
      req.rx_transmit=0;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end

  endtask


endclass
```

9.2.14. serdes_open_bist_sequence.sv

```
//------------------------------------------------------------------------
//                    UVM FRAMEWORK
//------------------------------------------------------------------------
// Project         : SERDES
// Unit            : serdes sequence mode 6
// File            : serdes_open_bist_sequence.svh
//------------------------------------------------------------------------
// Creation Date   :
//------------------------------------------------------------------------
// Description: this class defines the base top level sequence used
//    for the serdes simulations. It is used as the top
//    level sequence for test_top. It is extended to create
//    specific test scenarios.
//
//------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
```

```
class serdes_open_bist_sequence extends uvm_sequence #(serdes_seq_item);

  `uvm_object_utils( serdes_open_bist_sequence );

  //---------------------------------------
  //Constructor
  //---------------------------------------
  function new( string name = "serdes_open_bist_sequence" );
     super.new( name );
  endfunction

  //Creating the sequence
  virtual task body();
    repeat (2) begin
      req=serdes_seq_item::type_id::create("req");
      req.rx_transmit=0;
      req.config_in[2:0]=3'b110;
      req.config_in[4]=1'b1;
      req.config_in[5]=1'b1;
      req.data_transmit=10'h000;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b110;
      req.config_in[4]=1'b1;
      req.config_in[5]=1'b1;
      req.data_transmit=10'b11_1000_0011;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (62) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b110;
      req.config_in[4]=1'b1;
      req.config_in[5]=1'b1;
      req.data_transmit=10'h0ae;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b110;
      req.config_in[4]=1'b1;
      req.config_in[5]=1'b1;
      req.data_transmit=10'b11_1000_0011;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
  endtask


endclass
```

9.2.15. serdes_rxa_output_analog_loopback_sequence.sv

```
//------------------------------------------------------------------------
//                    UVM FRAMEWORK
//------------------------------------------------------------------------
// Project          : SERDES
// Unit             : serdes sequence mode 7
// File             : serdes_rxa_output_analog_loopback_sequence.svh
//------------------------------------------------------------------------
// Creation Date    :
//------------------------------------------------------------------------
// Description: this class defines the base top level sequence used
//    for the serdes simulations. It is used as the top
//    level sequence for test_top. It is extended to create
//    specific test scenarios.
//
//------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rxa_output_analog_loopback_sequence extends uvm_sequence #(serdes_seq_item);

  `uvm_object_utils( serdes_rxa_output_analog_loopback_sequence );

  //---------------------------------------
  //Constructor
  //---------------------------------------
  function new( string name = "serdes_rxa_output_analog_loopback_sequence" );
     super.new( name );
  endfunction

  //Creating the sequence
  virtual task body();
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.rx_transmit=0;
      req.config_in[2:0]=3'b111;
      req.config_in[5]=1'b1;
      req.data_transmit=10'h000;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b111;
      req.config_in[5]=1'b1;
      req.data_transmit=10'b11_1000_0011;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b111;
       req.config_in[5]=1'b1;
      req.data_transmit=10'h0b9;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b111;
      req.config_in[5]=1'b1;
      req.data_transmit=10'h0ae;
      req.rx_transmit=1;
      req.test_en=1;
```

```
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b111;
      req.config_in[5]=1'b1;
      req.data_transmit=10'h0ad;
      req.rx_transmit=1;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end
    repeat (1) begin
      req=serdes_seq_item::type_id::create("req");
      req.config_in[2:0]=3'b111;
      req.config_in[5]=1'b1;
      req.data_transmit=10'h000;
      req.rx_transmit=0;
      req.test_en=1;
      start_item(req);
      finish_item(req);
    end

  endtask


endclass
```

9.2.16. serdes_rx_sequencer.sv

```
/* Autogenerated Code for serdes_rx_sequencer.sv */

`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rx_sequencer extends uvm_sequencer#(serdes_seq_item);

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(serdes_rx_sequencer)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new


endclass : serdes_rx_sequencer
```

9.2.17. serdes_tx_sequencer.sv

```
/* Autogenerated Code for serdes_tx_sequencer.sv */

`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_tx_sequencer extends uvm_sequencer#(serdes_seq_item);

   //------------------------------------
```

```
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(serdes_tx_sequencer)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new


endclass : serdes_tx_sequencer
```

9.2.18. serdes_rx_driver.sv

```
//------------------------------------------------------------------------------
//                       UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rx_driver extends uvm_driver #(serdes_seq_item);
   serdes_config rx_cfg;
   //------------------------------------
   // Virtual Interface
   //------------------------------------
   virtual serdes_if   vif;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(serdes_rx_driver)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db #(serdes_config)::get(this,"","serdes_config", rx_cfg)) begin
         `uvm_fatal("build_phase", "Config object not found in uvm_config_db")
      end
      if(!uvm_config_db#(virtual serdes_if)::get(this, "","vif",vif))
         `uvm_fatal("No_vif", {"Virtual interface must be set for:
",get_full_name(),".vif"});
   endfunction : build_phase

   //------------------------------------
   // Run phase
   //------------------------------------
   virtual task run_phase(uvm_phase phase);
      forever begin
         seq_item_port.get_next_item(req);
         drive();
         seq_item_port.item_done();
      end
   endtask : run_phase
```

```
   //-----------------------------------
   // Drive task
   //-----------------------------------
   virtual task drive();
    fork
    forever begin
    @(posedge vif.clk_in_slow);
    `uvm_info("",$sformatf("RX Driver: data_transmit = %b rxa_in_p is %d config_in is %d
test_en is %d ", req.data_transmit, vif.rxa_in_p,vif.config_in, vif.test_en), UVM_MEDIUM)
    vif.config_in[2:0] <= req.config_in[2:0];
    if (!(rx_cfg.test_mode == 3'b011 || rx_cfg.test_mode == 3'b101)) begin
         vif.config_in[3] <= 0;
    end
    vif.config_in[7:4] <= req.config_in[7:4];
    vif.test_en <= req.test_en;
    end
    for (int i = 0; i <= 8 ; i = i + 1)
    begin
      `uvm_info("",$sformatf("RX Driver: rxa_in_p is %0d data_transmit[%0d] = %0d
",vif.rxa_in_p, i, req.data_transmit[i]), UVM_MEDIUM)
       vif.rx_transmit<= req.rx_transmit;
       if (rx_cfg.test_mode == 3'b011 || rx_cfg.test_mode == 3'b101) begin
         vif.config_in[3] <= req.data_transmit[i];
       end else begin
         vif.rxa_in_p<= req.data_transmit[i];//serial data
         vif.rxa_in_n <= !req.data_transmit[i];
       end
       @(posedge vif.clk_in_slow);
    end
    join_any
   endtask : drive

endclass : serdes_rx_driver
```

9.2.19. serdes_tx_driver.sv

```
/* Autogenerated Code for serdes tx driver.sv */

`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_tx_driver extends uvm_driver#(serdes_seq_item);
   //-----------------------------------
   // Virtual Interface
   //-----------------------------------
  virtual serdes_if vif;

   //-----------------------------------
   // UVM automation macros for general components
   //-----------------------------------
   `uvm_component_utils(serdes_tx_driver)


   //-----------------------------------
   // Constructor
   //-----------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //-----------------------------------
   // Build phase
   //-----------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db#(virtual serdes_if)::get(this, "","vif",vif))
         `uvm_fatal("No_vif", {"Virtual interface must be set for:
",get_full_name(),".vif"});
   endfunction : build_phase
```

```
   //-------------------------------------
   // Run phase
   //-------------------------------------
   virtual task run_phase(uvm_phase phase);
      forever begin
         seq_item_port.get_next_item(req);
         drive();
         seq_item_port.item_done();
      end
   endtask : run_phase

   //-------------------------------------
   // Drive task
   //-------------------------------------
   virtual task drive();

    @(posedge vif.clk_in_slow);
    `uvm_info("",$sformatf("TX Driver: tx_transmit=%b data_transmit = %b config_in is %d
txd_data_in is %d test_en is %d ", req.tx_transmit, req.data_transmit,
vif.config_in,vif.txd_data_in, vif.test_en), UVM_MEDIUM)
    vif.config_in <= req.config_in;
    vif.test_en <= req.test_en;
    vif.txd_data_out <= req.txd_data_out;
    vif.txd_data_in <= req.txd_data_in;
    vif.tx_transmit <= req.tx_transmit;

    for (int i = 0; i <= 9 ; i = i + 1) begin
       @(posedge vif.clk_in_slow);
       vif.tx_transmit <= 0;
    end
   endtask : drive

endclass : serdes_tx_driver
```

9.2.20. serdes_rx_monitor.sv

```
/* Autogenerated Code for serdes_rx_monitor.sv */
   `include "uvm_macros.svh"
   import uvm_pkg::*;
   import serdesEnvPkg::*;

class serdes_rx_monitor extends uvm_monitor;

   //-------------------------------------
   // Virtual Interface
   //-------------------------------------
  virtual serdes_if vif;

   //-------------------------------------
   // UVM automation macros for general components
   //-------------------------------------
   `uvm_component_utils(serdes_rx_monitor)

   //-------------------------------------
   // Analysis port
   //-------------------------------------
   uvm_analysis_port #(serdes_seq_item) rx_item_collected_port;

   //-------------------------------------
   // Constructor
   //-------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //-------------------------------------
```

```
   // Build phase
   //-------------------------------------
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    rx_item_collected_port = new("rx_item_collected_port", this);
      if(!uvm_config_db#(virtual serdes_if)::get(this, "","vif",vif))
         `uvm_fatal("No_vif", {"Virtual interface must be set for:
",get_full_name(),".vif"});
   endfunction : build_phase

   //-------------------------------------
   // Run phase
   //-------------------------------------
   virtual task run_phase(uvm_phase phase);
    forever begin
      @(posedge vif.clk_in_slow);
      if(vif.rx_transmit || vif.c_data_valid ) begin
        serdes_seq_item seq_item_collected = serdes_seq_item::type_id::create
("seq_item_collected", this);
        `uvm_info("",$sformatf("RX_Monitor: rx_transmit = %h", vif.rx_transmit), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: c_data_valid = %h", vif.c_data_valid),
UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: test_en = %h", vif.test_en), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: config_in = %h", vif.config_in), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: digital_out = %h", vif.digital_out), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: rxd_output_muxout_wire = %h",
vif.rxd_output_muxout_wire), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: rxd_input_muxout_wire = %h",
vif.rxd_input_muxout_wire), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: rxa_in_p= %h", vif.rxa_in_p), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: rxa_in_n = %h", vif.rxa_in_n), UVM_MEDIUM)
        `uvm_info("",$sformatf("RX_Monitor: bist_end = %h", vif.bist_end), UVM_MEDIUM)
        seq_item_collected.rx_transmit = vif.rx_transmit;
        seq_item_collected.c_data_valid = vif.c_data_valid;
        seq_item_collected.rxa_in_p = vif.rxa_in_p;
        seq_item_collected.rxa_in_n = vif.rxa_in_n;
        seq_item_collected.test_en = vif.test_en;
        seq_item_collected.config_in = vif.config_in;
        seq_item_collected.digital_out = vif.digital_out;
        seq_item_collected.rxd_output_muxout_wire = vif.rxd_output_muxout_wire;
        seq_item_collected.rxd_input_muxout_wire = vif.rxd_input_muxout_wire;
        seq_item_collected.bist_end = vif.bist_end;
        rx_item_collected_port.write(seq_item_collected);
      end
      end
   endtask : run_phase

endclass : serdes_rx_monitor
```

9.2.21. serdes_tx_monitor.sv

```
/* Autogenerated Code for serdes_tx_monitor.sv */
   `include "uvm_macros.svh"
   import uvm_pkg::*;
   import serdesEnvPkg::*;

class serdes_tx_monitor extends uvm_monitor;
   serdes_config tx_cfg;
   //-------------------------------------
   // Virtual Interface
   //-------------------------------------
  virtual serdes_if vif;

   //-------------------------------------
   // UVM automation macros for general components
   //-------------------------------------
   `uvm_component_utils(serdes_tx_monitor)
```

```
   //-------------------------------------
   // Analysis port
   //-------------------------------------
   uvm_analysis_port #(serdes_seq_item) tx_item_collected_port;

   //-------------------------------------
   // Constructor
   //-------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //-------------------------------------
   // Build phase
   //-------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db #(serdes_config)::get(this,"","serdes_config", tx_cfg)) begin
          `uvm_fatal("build_phase", "Config object not found in uvm_config_db")
      end
      tx_item_collected_port = new("tx_item_collected_port", this);
      if(!uvm_config_db#(virtual serdes_if)::get(this, "","vif",vif))
          `uvm_fatal("No_vif", {"Virtual interface must be set for:
",get_full_name(),".vif"});
   endfunction : build_phase

   //-------------------------------------
   // Run phase
   //-------------------------------------
   virtual task run_phase(uvm_phase phase);
    forever begin
      @(posedge vif.clk_in_slow);
      if (tx_cfg.test_mode[2:0] == 3'b111) begin
          if (vif.rx_transmit) begin
         serdes_seq_item seq_item_collected = serdes_seq_item::type_id::create
("seq_item_collected", this);
        `uvm_info("",$sformatf("TX_Monitor: tx_transmit = %h", vif.tx_transmit), UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: tx_frame_start = %h", vif.tx_frame_start),
UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: txa_data_out_p = %h", vif.txa_data_out_p),
UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: txa_data_out_n = %h", vif.txa_data_out_n),
UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: txd_data_out = %h", vif.txd_data_out),
UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: txd_data_in = %h", vif.txd_data_in), UVM_MEDIUM)

        seq_item_collected.tx_frame_start = vif.tx_frame_start;
        seq_item_collected.tx_transmit = vif.tx_transmit;
        seq_item_collected.test_en = vif.test_en;
        seq_item_collected.config_in = vif.config_in;
        seq_item_collected.txa_data_out_p = vif.txa_data_out_p;
        seq_item_collected.txa_data_out_n = vif.txa_data_out_n;
        seq_item_collected.txd_input_muxout_wire = vif.txd_input_muxout_wire;
        seq_item_collected.txd_data_out  = vif.txd_data_out;
        seq_item_collected.txd_data_in = vif.txd_data_in;
        tx_item_collected_port.write(seq_item_collected);
        end
      end else begin

      if(vif.tx_transmit || vif.tx_frame_start ) begin
        serdes_seq_item seq_item_collected = serdes_seq_item::type_id::create
("seq_item_collected", this);
        `uvm_info("",$sformatf("TX_Monitor: tx_transmit = %h", vif.tx_transmit), UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: tx_frame_start = %h", vif.tx_frame_start),
UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: txa_data_out_p = %h", vif.txa_data_out_p),
UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: txa_data_out_n = %h", vif.txa_data_out_n),
UVM_MEDIUM)
```

```
        `uvm_info("",$sformatf("TX_Monitor: txd_data_out = %h", vif.txd_data_out),
UVM_MEDIUM)
        `uvm_info("",$sformatf("TX_Monitor: txd_data_in = %h", vif.txd_data_in), UVM_MEDIUM)
        seq_item_collected.tx_frame_start = vif.tx_frame_start;
        seq_item_collected.tx_transmit = vif.tx_transmit;
        seq_item_collected.test_en = vif.test_en;
        seq_item_collected.config_in = vif.config_in;
        seq_item_collected.txa_data_out_p = vif.txa_data_out_p;
        seq_item_collected.txa_data_out_n = vif.txa_data_out_n;
        seq_item_collected.txd_input_muxout_wire = vif.txd_input_muxout_wire;
        seq_item_collected.txd_data_out  = vif.txd_data_out;
        seq_item_collected.txd_data_in = vif.txd_data_in;
        tx_item_collected_port.write(seq_item_collected);
        if( vif.tx_frame_start ) begin
          repeat ( 9 ) @(posedge vif.clk_in_slow) begin
            serdes_seq_item subseq_item_collected = serdes_seq_item::type_id::create
("subseq_item_collected", this);
            `uvm_info("",$sformatf("TX_Monitor: tx_transmit = %h", vif.tx_transmit),
UVM_MEDIUM)
            `uvm_info("",$sformatf("TX_Monitor: tx_frame_start = %h", vif.tx_frame_start),
UVM_MEDIUM)
            `uvm_info("",$sformatf("TX_Monitor: txa_data_out_p = %h", vif.txa_data_out_p),
UVM_MEDIUM)
            `uvm_info("",$sformatf("TX_Monitor: txa_data_out_n = %h", vif.txa_data_out_n),
UVM_MEDIUM)
            `uvm_info("",$sformatf("TX_Monitor: txd_data_in = %h", vif.txd_data_in),
UVM_MEDIUM)
            subseq_item_collected.tx_frame_start =  1 ; //vif.tx_frame_start;
            subseq_item_collected.tx_transmit = vif.tx_transmit;
            subseq_item_collected.test_en = vif.test_en;
            subseq_item_collected.config_in = vif.config_in;
            subseq_item_collected.txa_data_out_p = vif.txa_data_out_p;
            subseq_item_collected.txa_data_out_n = vif.txa_data_out_n;
            subseq_item_collected.txd_input_muxout_wire = vif.txd_input_muxout_wire;
            subseq_item_collected.txd_data_out  = vif.txd_data_out;
            subseq_item_collected.txd_data_in = vif.txd_data_in;
            tx_item_collected_port.write(subseq_item_collected);
          end
        end
      end
      end
    end
   endtask : run_phase

endclass : serdes_tx_monitor
```

9.2.22. serdes_scoreboard.sv

```
//------------------------------------------------------------------------
//                       UVM FRAMEWORK
//------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_scoreboard extends uvm_scoreboard;
   serdes_config scbd_cfg;

   //-------------------------------------
   // UVM automation macros for general components
   //-------------------------------------
   `uvm_component_utils(serdes_scoreboard)

   //-------------------------------------
   // Declaring port to receive packets
   //-------------------------------------
  //uvm_analysis_imp#(serdes_seq_item, serdes_scoreboard) item_collected_export;
  uvm_analysis_export #(serdes_seq_item) rx_item_collected_export;
  uvm_analysis_export #(serdes_seq_item) tx_item_collected_export;
  uvm_tlm_analysis_fifo#(serdes_seq_item) rx_fifo; //tx_item_collected_export;
```

```
  uvm_tlm_analysis_fifo#(serdes_seq_item) tx_fifo; //rx_item_collected_export;
  //---------------------------------------
  // declaring pkt_qu to store the transactions recived from monitor
  //---------------------------------------
  serdes_seq_item ser_golden_q[$];
  serdes_seq_item ser_actual_q[$];
  serdes_seq_item par_golden_q[$];
  serdes_seq_item par_actual_q[$];
  serdes_seq_item rx_txn;
  serdes_seq_item tx_txn;
  bit tx_comma_detected = 0;

   //---------------------------------------
   // Constructor
   //---------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
    rx_txn = new("rx_txn");
    tx_txn = new("tx_txn");
   endfunction : new

   //---------------------------------------
   // Build phase
   //---------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db#(serdes_config)::get(this, "", "serdes_config",scbd_cfg)) begin
        `uvm_fatal("No_cfg", {"CFG object not found: ",get_full_name(),"scbd_cfg"});
      end
      //Creating port
      rx_item_collected_export = new("rx_item_collected_export", this);
      tx_item_collected_export = new("tx_item_collected_export", this);
      // Creating fifo
      rx_fifo = new("rx_fifo", this);
      tx_fifo = new("tx_fifo",this);
   endfunction : build_phase

  function void connect_phase(uvm_phase phase);
      rx_item_collected_export.connect(rx_fifo.analysis_export);
      tx_item_collected_export.connect(tx_fifo.analysis_export);
  endfunction: connect_phase

   //---------------------------------------
   // Run phase
   //---------------------------------------
   virtual task run_phase(uvm_phase phase);
      forever begin

      logic[9:0] ser_golden_stream = '0;
      logic[9:0] ser_actual_stream = '0;
      logic[8:0] par_golden_stream = '0;
      logic[8:0] par_actual_stream = '0;

    ////////////////////////////////////////////////////////////////////////////
    ////////////////////MODE 0 - SERDES FUNCTIONAL OPERATION ///////////////////////
    ////////////////////////////////////////////////////////////////////////////
    if (scbd_cfg.test_mode==3'b000) begin
        `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 0 - Functional Operation
:: ------"),UVM_LOW)
        fork
        begin
        rx_fifo.get(rx_txn);
        $display("SCOREBOARD GOT RX TXN");
        end
        begin
        tx_fifo.get(tx_txn);
        $display("SCOREBOARD GOT TX TXN");
        end
        join_any
    end
    ////////////////////////////////////////////////////////////////////////////
```

```
    ////////////////////MODE 01 - PARALLEL LOOPBACK ////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
    if (scbd_cfg.test_mode==3'b001 ) begin
        `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 01 - Parallel loopback
:: ------"),UVM_LOW)
      tx_fifo.get(tx_txn);
      $display("SCOREBOARD GOT TX TXN");

      if( tx_txn.tx_frame_start ) begin  // TX MONITOR captures serial output // FIXME
        ser_actual_q.push_back( tx_txn );
      end
      if( tx_txn.tx_transmit ) begin  // TX MONITOR snoops interface and captures parallel
data
        par_golden_q.push_back( tx_txn );
      end

      `uvm_info(get_type_name(),$sformatf("SER GOLDEN Q SIZE:
%d",ser_golden_q.size()),UVM_LOW)
      `uvm_info(get_type_name(),$sformatf("SER ACTUAL Q SIZE:
%d",ser_actual_q.size()),UVM_LOW)
      `uvm_info(get_type_name(),$sformatf("PAR GOLDEN Q SIZE:
%d",par_golden_q.size()),UVM_LOW)
      `uvm_info(get_type_name(),$sformatf("PAR ACTUAL Q SIZE:
%d",par_actual_q.size()),UVM_LOW)

      if( ser_actual_q.size()>0 && ser_actual_q.size()%10 == 0 ) begin
        //if( ser_actual_q.size()>0 && ser_actual )
        while( ser_actual_q.size() > 0 ) begin
          //ser_actual_stream = { ser_actual_stream[8:0],
ser_actual_q.pop_front().txd_data_out } ;
          ser_actual_stream = { ser_actual_q.pop_front().txd_data_out,
ser_actual_stream[9:1] } ;
        end
        `uvm_info(get_type_name(),$sformatf("Received a full actual txn:
%h",ser_actual_stream),UVM_LOW)
        if( ser_actual_stream == 10'b11_1000_0011 || ser_actual_stream == 10'b00_0111_1100 )
begin
          tx_comma_detected = 1;

          repeat(9) begin
          if( tx_comma_detected ) begin
             rx_fifo.get(rx_txn);
             $display("SCOREBOARD GOT RX TXN");
             if( rx_txn.rx_transmit ) begin  // RX MONITOR snoops interface and captures
serial data
               ser_golden_q.push_back( rx_txn );
             end
             if( rx_txn.c_data_valid ) begin  // RX MONITOR captures digital output
               par_actual_q.push_back( rx_txn );
             end
          end
          end
        end
      end

      if( tx_comma_detected ) begin
        rx_fifo.get(rx_txn);
        $display("SCOREBOARD GOT RX TXN");
        if( rx_txn.rx_transmit ) begin  // RX MONITOR snoops interface and captures serial
data
          ser_golden_q.push_back( rx_txn );
        end
        if( rx_txn.c_data_valid ) begin  // RX MONITOR captures digital output
          par_actual_q.push_back( rx_txn );
        end
      end

      if( ser_golden_q.size()>0 && ser_golden_q.size()%10 == 0 ) begin
        //if( ser_actual_q.size()>0 && ser_actual )
        while( ser_golden_q.size() > 0 ) begin
```

```
          ser_golden_stream = { ser_golden_q.pop_front().rxa_in_p, ser_golden_stream[9:1] }
;
        end
        `uvm_info(get_type_name(),$sformatf("Received a full golden txn:
%h",ser_golden_stream),UVM_LOW)
            if ( tx_comma_detected && ser_actual_stream !=ser_golden_stream ) begin
                `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",ser_golden_stream, ser_actual_stream))
            end else begin
                `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",ser_golden_stream, ser_actual_stream),UVM_LOW)
            end
      end

    end
    ////////////////////////////////////////////////////////////////////////////////
    ////////////////////MODE 02 - SERIAL LOOPBACK //////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
    if (scbd_cfg.test_mode==3'b010) begin
      `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 02 - Serial loopback       ::
------"),UVM_LOW)
      fork
      begin
      tx_fifo.get(tx_txn);
      $display("SCOREBOARD GOT TX TXN");
          if( tx_txn.tx_transmit ) begin  // TX MONITOR snoops interface and captures
parallel data
          par_golden_q.push_back( tx_txn );
                    `uvm_info(get_type_name(),$sformatf("tx_transmit
%d",tx_txn.tx_transmit),UVM_LOW)

          end
      end
      begin
      rx_fifo.get(rx_txn);
      $display("SCOREBOARD GOT RX TXN");
          if( rx_txn.c_data_valid ) begin  // RX MONITOR captures digital output
          par_actual_q.push_back( rx_txn );
          end
      end
      join_any

    `uvm_info(get_type_name(),$sformatf("PAR GOLDEN Q SIZE:
%d",par_golden_q.size()),UVM_LOW)
    `uvm_info(get_type_name(),$sformatf("PAR ACTUAL Q SIZE:
%d",par_actual_q.size()),UVM_LOW)

     if (par_golden_q.size()>0 && par_actual_q.size()>0 && (par_actual_q.size() ==
par_golden_q.size()) ) begin
              par_golden_stream = par_golden_q.pop_front().txd_data_in;
              par_actual_stream = par_actual_q.pop_front().digital_out;
              if (par_actual_stream != par_golden_stream) begin
                 `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",par_golden_stream, par_actual_stream))
              end else begin
                 `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",par_golden_stream, par_actual_stream),UVM_LOW)
              end
        end
    end
    ////////////////////////////////////////////////////////////////////////////////
    ////////////////////MODE 03 - RXA bypass ///////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
    if (scbd_cfg.test_mode==3'b011 ) begin
        `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 03 - RXA bypass       :: --
----"),UVM_LOW)
        rx_fifo.get(rx_txn);
        $display("SCOREBOARD GOT RX TXN");
        if( rx_txn.rx_transmit ) begin  // RX MONITOR snoops interface and captures serial
data
          ser_golden_q.push_back( rx_txn );
```

```
        end
        if ( rx_txn.config_in[3] != rx_txn.rxd_input_muxout_wire ) begin
                `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",rx_txn.config_in[3], rx_txn.rxd_input_muxout_wire))
         end else begin
        `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",rx_txn.config_in[3], rx_txn.rxd_input_muxout_wire),UVM_LOW)
        end
    end
    ////////////////////////////////////////////////////////////////////////////////
    ////////////////////MODE 04 - BIST WITH SERIAL LOOPBACK ////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
    if (scbd_cfg.test_mode==3'b100) begin
      `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 04 - BIST with serial
loopback       :: ------"),UVM_LOW)
      fork
      begin
      tx_fifo.get(tx_txn);
      $display("SCOREBOARD GOT TX TXN");
          if( tx_txn.tx_transmit) begin  // TX MONITOR snoops interface and captures
parallel data
          par_golden_q.push_back( tx_txn );
                    `uvm_info(get_type_name(),$sformatf("CESAR txd_input_muxout_wire
%d",tx_txn.txd_input_muxout_wire),UVM_LOW)

          end
      end
      begin
      rx_fifo.get(rx_txn);
      $display("SCOREBOARD GOT RX TXN");
          if( rx_txn.c_data_valid && !rx_txn.bist_end ) begin  // RX MONITOR captures
digital output
          par_actual_q.push_back( rx_txn );
          end
      end
      join_any

    `uvm_info(get_type_name(),$sformatf("PAR GOLDEN Q SIZE:
%d",par_golden_q.size()),UVM_LOW)
    `uvm_info(get_type_name(),$sformatf("PAR ACTUAL Q SIZE:
%d",par_actual_q.size()),UVM_LOW)

     if (par_golden_q.size()==64 && par_actual_q.size() == 64) begin
            while( par_golden_q.size() > 0 ) begin
             par_golden_stream = par_golden_q.pop_front().txd_input_muxout_wire;
             par_actual_stream = par_actual_q.pop_front().digital_out;
             if (par_actual_stream != par_golden_stream) begin
                `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",par_golden_stream, par_actual_stream))
             end else begin
                `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",par_golden_stream, par_actual_stream),UVM_LOW)
             end
             end
        end
    end
    ////////////////////////////////////////////////////////////////////////////////
    ////////////////////MODE 05 - RXA BYPASS WITH PARALLEL LOOPBACK ////////////////
    ////////////////////////////////////////////////////////////////////////////////
    if (scbd_cfg.test_mode==3'b101) begin
        `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 05 - RXA bypass with
parallel loopback       :: ------"),UVM_LOW)
      tx_fifo.get(tx_txn);
      $display("SCOREBOARD GOT TX TXN");
      if( tx_txn.tx_frame_start ) begin  // TX MONITOR captures serial output
        ser_actual_q.push_back( tx_txn );
      end

      `uvm_info(get_type_name(),$sformatf("SER GOLDEN Q SIZE:
%d",ser_golden_q.size()),UVM_LOW)
```

```
      `uvm_info(get_type_name(),$sformatf("SER ACTUAL Q SIZE:
%d",ser_actual_q.size()),UVM_LOW)

      if( ser_actual_q.size()>0 && ser_actual_q.size()%10 == 0 ) begin
        while( ser_actual_q.size() > 0 ) begin
          ser_actual_stream = { ser_actual_q.pop_front().txd_data_out,
ser_actual_stream[9:1] } ;
        end
        `uvm_info(get_type_name(),$sformatf("Received a full actual txn:
%h",ser_actual_stream),UVM_LOW)
        if( ser_actual_stream == 10'b11_1000_0011 || ser_actual_stream == 10'b00_0111_1100 )
begin
          tx_comma_detected = 1;

          repeat(9) begin
          if( tx_comma_detected ) begin
             rx_fifo.get(rx_txn);
             $display("SCOREBOARD GOT RX TXN");
             if( rx_txn.rx_transmit ) begin  // RX MONITOR snoops interface and captures
serial data
               ser_golden_q.push_back( rx_txn );
             end
             if( rx_txn.c_data_valid ) begin  // RX MONITOR captures digital output
               par_actual_q.push_back( rx_txn );
             end
          end
          end
         end
      end

      if( tx_comma_detected ) begin
        rx_fifo.get(rx_txn);
        $display("SCOREBOARD GOT RX TXN");
        if( rx_txn.rx_transmit ) begin  // RX MONITOR snoops interface and captures serial
data
          ser_golden_q.push_back( rx_txn );
        end
        if( rx_txn.c_data_valid ) begin  // RX MONITOR captures digital output
          par_actual_q.push_back( rx_txn );
        end
      end

      if( ser_golden_q.size()>0 && ser_golden_q.size()%10 == 0 ) begin
        while( ser_golden_q.size() > 0 ) begin
          ser_golden_stream = { ser_golden_q.pop_front().config_in[3],
ser_golden_stream[9:1] } ;
        end
        `uvm_info(get_type_name(),$sformatf("Received a full golden txn:
%h",ser_golden_stream),UVM_LOW)
            if ( tx_comma_detected && ser_actual_stream !=ser_golden_stream ) begin
                `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",ser_golden_stream, ser_actual_stream))
            end else begin
                `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",ser_golden_stream, ser_actual_stream),UVM_LOW)
            end
        //end
      end

    end
    ////////////////////////////////////////////////////////////////////////////////
    ////////////////////MODE 06 - OPEN BIST ////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
    if (scbd_cfg.test_mode==3'b110) begin
        `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 06 - Open BIST       :: ---
---"),UVM_LOW)
        rx_fifo.get(rx_txn);
        $display("SCOREBOARD GOT RX TXN");
        if (rx_txn.bist_end) begin
                $display("SCOREBOARD GOT BIST END");
           if ( rx_txn.digital_out[5:0] != 'd61  ) begin
```

```
                `uvm_error(get_type_name(),$sformatf("Expected number of errors mismatch.
Expected: %d Actual: %d", 61, rx_txn.digital_out[5:0]))
             end else begin
             `uvm_info(get_type_name(),$sformatf("Expected number of erros match. Expected: %d
Actual: %d",61, rx_txn.digital_out[5:0]),UVM_LOW)
             end
          end
      end
      ////////////////////////////////////////////////////////////////////////////////
      ////////////////////MODE 07 - RXA OUTPUT WITH ANALOG LOOPBACK //////////////////////
      ////////////////////////////////////////////////////////////////////////////////
      if (scbd_cfg.test_mode==3'b111) begin
          `uvm_info(get_type_name(),$sformatf("------ :: TEST MODE 07 - RXA output with analog
loopback       :: ------"),UVM_LOW)
          rx_fifo.get(rx_txn);
          $display("SCOREBOARD GOT RX TXN");
          tx_fifo.get(tx_txn);
          $display("SCOREBOARD GOT TX TXN");
          if ( rx_txn.rxa_in_p != rx_txn.digital_out[5]  ) begin
                `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",rx_txn.rxa_in_p, rx_txn.digital_out[5]))
           end else begin
          `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",rx_txn.rxa_in_p, rx_txn.digital_out[5]),UVM_LOW)
          end
          if ( rx_txn.rxa_in_p != tx_txn.txa_data_out_p  ) begin
                `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",rx_txn.rxa_in_p, tx_txn.txa_data_out_p))
           end else begin
          `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",rx_txn.rxa_in_p, tx_txn.txa_data_out_p),UVM_LOW)
          end
          if ( rx_txn.rxa_in_n != tx_txn.txa_data_out_n ) begin
                `uvm_error(get_type_name(),$sformatf("Stream mismatch. Expected: %h Actual:
%h",rx_txn.rxa_in_n, tx_txn.txa_data_out_n))
           end else begin
          `uvm_info(get_type_name(),$sformatf("Stream match. Expected: %h Actual:
%h",rx_txn.rxa_in_n, tx_txn.txa_data_out_n),UVM_LOW)
          end
      end
      end
     endtask : run_phase

endclass : serdes_scoreboard
```

9.2.23. serdes_rx_agent.sv

```
/* Autogenerated Code for serdes_tx_agent.sv */
   `include "uvm_macros.svh"
   import uvm_pkg::*;

 import serdesEnvPkg::*;

class serdes_rx_agent extends uvm_agent;
   serdes_config rx_cfg;
  //----------------------------------------
  // Declaring Agent components
  //----------------------------------------
   serdes_rx_monitor   rx_monitor;
   serdes_rx_sequencer   rx_sequencer;
   serdes_rx_driver   rx_driver;
   serdes_rx_subscriber rx_subscriber;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(serdes_rx_agent)
```

```
   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db #(serdes_config)::get(this,"","serdes_config", rx_cfg)) begin
         `uvm_fatal("build_phase", "Config object not found in uvm_config_db")
      end
      rx_monitor = serdes_rx_monitor::type_id::create("rx_monitor", this);
    if (rx_cfg.is_active == UVM_ACTIVE) begin
      rx_driver    = serdes_rx_driver::type_id::create("rx_driver", this);
         `uvm_info (get_type_name(),$sformatf("AGENT: RX AGENT driver configured as
ACTIVE"),UVM_MEDIUM)
      end else begin
         `uvm_info (get_type_name(),$sformatf("AGENT: RX AGENT driver configured as
PASSIVE"),UVM_MEDIUM)
      end

    if (rx_cfg.is_active == UVM_ACTIVE) begin
      rx_sequencer = serdes_rx_sequencer::type_id::create("rx_sequencer", this);
         `uvm_info (get_type_name(),$sformatf("AGENT: RX AGENT sequencer configured as
ACTIVE"),UVM_MEDIUM)
      end else begin
         `uvm_info (get_type_name(),$sformatf("AGENT: RX AGENT sequencer configured as
PASSIVE"),UVM_MEDIUM)
      end
    if (rx_cfg.has_subscriber)
    rx_subscriber = serdes_rx_subscriber::type_id::create("rx_subscriber", this);
  endfunction : build_phase

   //------------------------------------
   // Connect phase
   //------------------------------------
  function void connect_phase(uvm_phase phase);
    if(rx_cfg.is_active == UVM_ACTIVE)
      rx_driver.seq_item_port.connect(rx_sequencer.seq_item_export);
    if (rx_cfg.has_subscriber) begin
      `uvm_info (get_type_name(),$sformatf("Monitor: RX Coverage enabled"),UVM_MEDIUM)
       rx_monitor.rx_item_collected_port.connect( rx_subscriber.analysis_export );
    end else
       `uvm_info (get_type_name(),$sformatf("Monitor: RX Coverage disabled"),UVM_MEDIUM)
  endfunction : connect_phase

endclass : serdes_rx_agent
```

9.2.24. serdes_tx_agent.sv

```
/* Autogenerated Code for serdes_tx_agent.sv */
   `include "uvm_macros.svh"
   import uvm_pkg::*;

 import serdesEnvPkg::*;

class serdes_tx_agent extends uvm_agent;
   serdes_config tx_cfg;
  //----------------------------------------
  // Declaring Agent components
  //----------------------------------------
   serdes_tx_monitor   tx_monitor;
   serdes_tx_sequencer   tx_sequencer;
```

```
   serdes_tx_driver   tx_driver;
  serdes_tx_subscriber tx_subscriber;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(serdes_tx_agent)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if(!uvm_config_db #(serdes_config)::get(this,"","serdes_config", tx_cfg)) begin
         `uvm_fatal("build_phase", "Config object not found in uvm_config_db")
      end
      tx_monitor = serdes_tx_monitor::type_id::create("tx_monitor", this);
      if (tx_cfg.is_active == UVM_ACTIVE) begin
         tx_driver    = serdes_tx_driver::type_id::create("tx_driver", this);
         `uvm_info (get_type_name(),$sformatf("AGENT: TX AGENT driver configured as
ACTIVE"),UVM_MEDIUM)
      end else begin
         `uvm_info (get_type_name(),$sformatf("AGENT: TX AGENT driver configured as
PASSIVE"),UVM_MEDIUM)
      end

      if (tx_cfg.is_active == UVM_ACTIVE) begin
         tx_sequencer = serdes_tx_sequencer::type_id::create("tx_sequencer", this);
         `uvm_info (get_type_name(),$sformatf("AGENT: TX AGENT sequencer configured as
ACTIVE"),UVM_MEDIUM)
      end else begin
         `uvm_info (get_type_name(),$sformatf("AGENT: TX AGENT sequencer configured as
PASSIVE"),UVM_MEDIUM)
      end
      if (tx_cfg.has_subscriber)
        tx_subscriber = serdes_tx_subscriber::type_id::create("tx_subscriber", this);
  endfunction : build_phase

   //------------------------------------
   // Connect phase
   //------------------------------------
  function void connect_phase(uvm_phase phase);
    if(tx_cfg.is_active == UVM_ACTIVE)
         tx_driver.seq_item_port.connect( tx_sequencer.seq_item_export);
    if (tx_cfg.has_subscriber) begin
      `uvm_info (get_type_name(),$sformatf("Monitor: TX Coverage enabled"),UVM_MEDIUM)
       tx_monitor.tx_item_collected_port.connect( tx_subscriber.analysis_export );
    end else
       `uvm_info (get_type_name(),$sformatf("Monitor: TX Coverage disabled"),UVM_MEDIUM)
  endfunction : connect_phase

endclass : serdes_tx_agent
```

9.2.25. serdes_env.sv

```
/* Autogenerated Code for serdes_env.sv */

   `include "uvm_macros.svh"
   import uvm_pkg::*;
   import serdesEnvPkg::*;

class serdes_env extends uvm_env;
   serdes_tx_agent   tx_agent;
   serdes_rx_agent   rx_agent;
   serdes_scoreboard   scoreboard;

   //------------------------------------
   // UVM automation macros for general components
   //------------------------------------
   `uvm_component_utils(serdes_env)


   //------------------------------------
   // Constructor
   //------------------------------------
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction : new

   //------------------------------------
   // Build phase
   //------------------------------------
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      tx_agent = serdes_tx_agent::type_id::create("tx_agent", this);
      rx_agent = serdes_rx_agent::type_id::create("rx_agent", this);
      scoreboard = serdes_scoreboard::type_id::create("scoreboard", this);
   endfunction : build_phase

   //------------------------------------
   // Connect phase
   //------------------------------------
   function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    tx_agent.tx_monitor.tx_item_collected_port.connect(scoreboard.tx_item_collected_export);
    rx_agent.rx_monitor.rx_item_collected_port.connect(scoreboard.rx_item_collected_export);
   endfunction : connect_phase

endclass : serdes_env
```

9.2.26. serdes_base_test.sv

```
//------------------------------------------------------------------------------
//                       UVM FRAMEWORK
//------------------------------------------------------------------------------

`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_base_test extends uvm_test;
  serdes_config tx_cfg;
  serdes_config rx_cfg;
  serdes_config scbd_cfg;
  serdes_env env;

  //---------------------------------------
  // UVM automation macros for general components
  //---------------------------------------
  `uvm_component_utils(serdes_base_test)
```

```
  //---------------------------------------
  // constructor
  //---------------------------------------
  function new(string name = "serdes_base_test",uvm_component parent=null);
    super.new(name,parent);
  endfunction : new

  //---------------------------------------
  // build_phase
  //---------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    rx_cfg = serdes_config::type_id::create( "rx_cfg" );
    tx_cfg = serdes_config::type_id::create( "tx_cfg" );
    scbd_cfg = serdes_config::type_id::create( "scbd_cfg" );
    env = serdes_env::type_id::create("env", this);
  endfunction : build_phase


  //---------------------------------------
  // end_of_elaboration phase
  //---------------------------------------
  virtual function void end_of_elaboration();
    //print's the topology
    print();
  endfunction

  //---------------------------------------
  // end_of_elaboration phase
  //---------------------------------------
 function void report_phase(uvm_phase phase);
   uvm_report_server svr;
   super.report_phase(phase);

   svr = uvm_report_server::get_server();
   if(svr.get_severity_count(UVM_FATAL)+svr.get_severity_count(UVM_ERROR)>0) begin
     `uvm_info(get_type_name(), "---------------------------------------", UVM_NONE)
     `uvm_info(get_type_name(), "----            TEST FAIL          ----", UVM_NONE)
     `uvm_info(get_type_name(), "---------------------------------------", UVM_NONE)
    end
    else begin
     `uvm_info(get_type_name(), "---------------------------------------", UVM_NONE)
     `uvm_info(get_type_name(), "----           TEST PASS           ----", UVM_NONE)
     `uvm_info(get_type_name(), "---------------------------------------", UVM_NONE)
    end
  endfunction

endclass : serdes_base_test
```

9.2.27. serdes_parallel_loopback_test.sv

```
//------------------------------------------------------------------------------
//                      UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_parallel_loopback_test extends serdes_base_test;

   `uvm_component_utils(serdes_parallel_loopback_test)

  //---------------------------------------
  // sequence instance
  //---------------------------------------
  serdes_parallel_loopback_sequence seq;

  //---------------------------------------
  // constructor
```

```
  //----------------------------------------
  function new(string name = "serdes_parallel_loopback_test",uvm_component parent=null);
    super.new(name,parent);
  endfunction : new

  //----------------------------------------
  // build_phase
  //----------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
      rx_cfg.has_subscriber = 1;
      tx_cfg.has_subscriber = 1;
      scbd_cfg.test_mode = 3'b001;
      rx_cfg.test_mode = 3'b001;
      tx_cfg.is_active = UVM_PASSIVE;
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent", "serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent", "serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent.rx_driver",
"serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent.tx_monitor",
"serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.scoreboard", "serdes_config",scbd_cfg);

    // Create the sequence
      seq = serdes_parallel_loopback_sequence::type_id::create("seq");
  endfunction : build_phase

  //----------------------------------------
  // run_phase - starting the test
  //----------------------------------------
  task run_phase(uvm_phase phase);

    //Start the sequence
    phase.raise_objection(this);
      `uvm_info(get_type_name(), "Sending parallel loopback sequence", UVM_NONE)
    seq.start(env.rx_agent.rx_sequencer);
    phase.drop_objection(this);

    //set a drain-time for the environment if desired
    phase.phase_done.set_drain_time(this, 10000);
  endtask : run_phase

endclass : serdes_parallel_loopback_test
```

9.2.28. serdes_serial_loopback_test.sv

```
//------------------------------------------------------------------------------
//                      UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_serial_loopback_test extends serdes_base_test;

   `uvm_component_utils(serdes_serial_loopback_test)

  //----------------------------------------
  // sequence instance
  //----------------------------------------
  serdes_serial_loopback_sequence seq;

  //----------------------------------------
  // constructor
  //----------------------------------------
  function new(string name = "serdes_serial_loopback_test",uvm_component parent=null);
    super.new(name,parent);
  endfunction : new
```

```
  //---------------------------------------
  // build_phase
  //---------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
      rx_cfg.has_subscriber = 1;
      tx_cfg.has_subscriber = 1;
      scbd_cfg.test_mode = 3'b010;
      rx_cfg.test_mode = 3'b010;
      rx_cfg.is_active = UVM_PASSIVE;
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent", "serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent", "serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent.rx_driver",
"serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent.tx_monitor",
"serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.scoreboard", "serdes_config",scbd_cfg);

    // Create the sequence
      seq = serdes_serial_loopback_sequence::type_id::create("seq");
  endfunction : build_phase

  //---------------------------------------
  // run_phase - starting the test
  //---------------------------------------
  task run_phase(uvm_phase phase);

    //Start the sequence
    phase.raise_objection(this);
      `uvm_info(get_type_name(), "Sending serial loopback sequence", UVM_NONE)
    seq.start(env.tx_agent.tx_sequencer);
    phase.drop_objection(this);

    //set a drain-time for the environment if desired
    phase.phase_done.set_drain_time(this, 10000);
  endtask : run_phase

endclass : serdes_serial_loopback_test
```

9.2.29. serdes_bist_serial_loopback_test.sv

```
//------------------------------------------------------------------------------
//                      UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_bist_serial_loopback_test extends serdes_base_test;

   `uvm_component_utils(serdes_bist_serial_loopback_test)

  //---------------------------------------
  // sequence instance
  //---------------------------------------
  serdes_bist_serial_loopback_sequence seq;

  //---------------------------------------
  // constructor
  //---------------------------------------
  function new(string name = "serdes_bist_serial_loopback_test",uvm_component parent=null);
    super.new(name,parent);
  endfunction : new

  //---------------------------------------
  // build_phase
  //---------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
```

```
      rx_cfg.has_subscriber = 1;
      tx_cfg.has_subscriber = 1;
      scbd_cfg.test_mode = 3'b100;
      rx_cfg.test_mode = 3'b100;
      rx_cfg.is_active = UVM_PASSIVE;
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent", "serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent", "serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent.rx_driver",
"serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent.tx_monitor",
"serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.scoreboard", "serdes_config",scbd_cfg);

    // Create the sequence
      seq = serdes_bist_serial_loopback_sequence::type_id::create("seq");
  endfunction : build_phase

  //---------------------------------------
  // run_phase - starting the test
  //---------------------------------------
  task run_phase(uvm_phase phase);

    //Start the sequence
    phase.raise_objection(this);
      `uvm_info(get_type_name(), "Sending bist serial loopback sequence", UVM_NONE)
    seq.start(env.tx_agent.tx_sequencer);
    phase.drop_objection(this);

    //set a drain-time for the environment if desired
    phase.phase_done.set_drain_time(this, 10000);
  endtask : run_phase

endclass : serdes_bist_serial_loopback_test
```

9.2.30. serdes_rxa_bypass_test.sv

```
//------------------------------------------------------------------------------
//                       UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rxa_bypass_test extends serdes_base_test;

   `uvm_component_utils(serdes_rxa_bypass_test)

  //---------------------------------------
  // sequence instance
  //---------------------------------------
  serdes_rxa_bypass_sequence seq;

  //---------------------------------------
  // constructor
  //---------------------------------------
  function new(string name = "serdes_rxa_bypass_test",uvm_component parent=null);
    super.new(name,parent);
  endfunction : new

  //---------------------------------------
  // build_phase
  //---------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
      rx_cfg.has_subscriber = 1;
      tx_cfg.has_subscriber = 1;
      super.scbd_cfg.test_mode = 3'b011;
      super.rx_cfg.test_mode = 3'b011;
      super.tx_cfg.is_active = UVM_PASSIVE;
```

```
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent", "serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent", "serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent.rx_driver",
"serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent.tx_monitor",
"serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.scoreboard", "serdes_config",scbd_cfg);

    // Create the sequence
      seq = serdes_rxa_bypass_sequence::type_id::create("seq");
  endfunction : build_phase

  //-------------------------------------
  // run_phase - starting the test
  //-------------------------------------
  task run_phase(uvm_phase phase);

    //Start the sequence
    phase.raise_objection(this);
      `uvm_info(get_type_name(), "Sending rxa bypass sequence", UVM_NONE)
    seq.start(env.rx_agent.rx_sequencer);
    phase.drop_objection(this);

    //set a drain-time for the environment if desired
    phase.phase_done.set_drain_time(this, 10000);
  endtask : run_phase

endclass : serdes_rxa_bypass_test
```

9.2.31. serdes_rxa_bypass_parallel_loopback_test.sv

```
//------------------------------------------------------------------------------
//                      UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rxa_bypass_parallel_loopback_test extends serdes_base_test;

   `uvm_component_utils(serdes_rxa_bypass_parallel_loopback_test)

  //-------------------------------------
  // sequence instance
  //-------------------------------------
  serdes_rxa_bypass_parallel_loopback_sequence seq;

  //-------------------------------------
  // constructor
  //-------------------------------------
  function new(string name = "serdes_rxa_bypass_parallel_loopback_test",uvm_component
parent=null);
    super.new(name,parent);
  endfunction : new

  //-------------------------------------
  // build_phase
  //-------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
      rx_cfg.has_subscriber = 1;
      tx_cfg.has_subscriber = 1;
      scbd_cfg.test_mode = 3'b101;
      rx_cfg.test_mode = 3'b101;
      tx_cfg.is_active = UVM_PASSIVE;
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent", "serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent", "serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent.rx_driver",
"serdes_config",rx_cfg);
```

```
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent.tx_monitor",
"serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.scoreboard", "serdes_config",scbd_cfg);

    // Create the sequence
      seq = serdes_rxa_bypass_parallel_loopback_sequence::type_id::create("seq");
  endfunction : build_phase

  //---------------------------------------
  // run_phase - starting the test
  //---------------------------------------
  task run_phase(uvm_phase phase);

    //Start the sequence
    phase.raise_objection(this);
      `uvm_info(get_type_name(), "Sending rxa bypass with parallel loopback sequence",
UVM_NONE)
    seq.start(env.rx_agent.rx_sequencer);
    phase.drop_objection(this);

    //set a drain-time for the environment if desired
    phase.phase_done.set_drain_time(this, 10000);
  endtask : run_phase

endclass : serdes_rxa_bypass_parallel_loopback_test
```

9.2.32. serdes_open_bist_test.sv

```
//------------------------------------------------------------------------------
//                       UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_open_bist_test extends serdes_base_test;

   `uvm_component_utils(serdes_open_bist_test)

  //---------------------------------------
  // sequence instance
  //---------------------------------------
  serdes_open_bist_sequence seq;

  //---------------------------------------
  // constructor
  //---------------------------------------
  function new(string name = "serdes_open_bist_test",uvm_component parent=null);
    super.new(name,parent);
  endfunction : new

  //---------------------------------------
  // build_phase
  //---------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
      rx_cfg.has_subscriber = 1;
      tx_cfg.has_subscriber = 1;
      scbd_cfg.test_mode = 3'b110;
      rx_cfg.test_mode = 3'b110;
      tx_cfg.is_active = UVM_PASSIVE;
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent", "serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent", "serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent.rx_driver",
"serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent.tx_monitor",
"serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.scoreboard", "serdes_config",scbd_cfg);
```

```
    // Create the sequence
      seq = serdes_open_bist_sequence::type_id::create("seq");
  endfunction : build_phase

  //---------------------------------------
  // run_phase - starting the test
  //---------------------------------------
  task run_phase(uvm_phase phase);

    //Start the sequence
    phase.raise_objection(this);
      `uvm_info(get_type_name(), "Sending open bist sequence", UVM_NONE)
    seq.start(env.rx_agent.rx_sequencer);
    phase.drop_objection(this);

    //set a drain-time for the environment if desired
    phase.phase_done.set_drain_time(this, 10000);
  endtask : run_phase

endclass : serdes_open_bist_test
```

9.2.33. serdes_rxa_output_analog_loopback_test.sv

```
//------------------------------------------------------------------------------
//                       UVM FRAMEWORK
//------------------------------------------------------------------------------
`include "uvm_macros.svh"
import uvm_pkg::*;
import serdesEnvPkg::*;
class serdes_rxa_output_analog_loopback_test extends serdes_base_test;

   `uvm_component_utils(serdes_rxa_output_analog_loopback_test)

  //---------------------------------------
  // sequence instance
  //---------------------------------------
  serdes_rxa_output_analog_loopback_sequence seq;

  //---------------------------------------
  // constructor
  //---------------------------------------
  function new(string name = "serdes_rxa_output_analog_loopback_test",uvm_component
parent=null);
    super.new(name,parent);
  endfunction : new

  //---------------------------------------
  // build_phase
  //---------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
      rx_cfg.has_subscriber = 1;
      tx_cfg.has_subscriber = 1;
      scbd_cfg.test_mode = 3'b111;
      rx_cfg.test_mode = 3'b111;
      tx_cfg.test_mode = 3'b111;
      tx_cfg.is_active = UVM_PASSIVE;
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent", "serdes_config",
super.tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent", "serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.rx_agent.rx_driver",
"serdes_config",rx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.tx_agent.tx_monitor",
"serdes_config",tx_cfg);
      uvm_config_db #(serdes_config)::set(this,"env.scoreboard", "serdes_config",scbd_cfg);

    // Create the sequence
      seq = serdes_rxa_output_analog_loopback_sequence::type_id::create("seq");
```

```
  endfunction : build_phase

  //---------------------------------------
  // run_phase - starting the test
  //---------------------------------------
  task run_phase(uvm_phase phase);

    //Start the sequence
    phase.raise_objection(this);
      `uvm_info(get_type_name(), "Sending RXA output analog loopback sequence", UVM_NONE)
    seq.start(env.rx_agent.rx_sequencer);
    phase.drop_objection(this);

    //set a drain-time for the environment if desired
    phase.phase_done.set_drain_time(this, 10000);
  endtask : run_phase

endclass : serdes_rxa_output_analog_loopback_test
```