TESTPLAN DOCUMENT FOR 10GE MAC CORE

Prepared By:
MRINAL MATHUR

Contents

1. Project Overview

This project is intended as a hands-on opportunity to build a SystemVerilog class-based verification environment, otherwise known as the OOP Testbench, on a 10Gb Ethernet MAC Core design. The source code for the 10GE Mac Core is provided as Verilog RTL. Furthermore, a rudimentary testbench and simulation environment is also provided.

The first step in the project is to get the understanding of the specification of the design. Next step will be to understand existing verification environments. Once the basic sanity checks are performed and testbench is working as expected, building of the OOP testbench is started.

The main purpose of the project is to build a OOP infrastructure where each component has a specialized job. This document explains all the components involved in creating the testbench by giving a brief description and providing reference to the actual blocks in the design.

In addition to the provided testcase, five other testcases were added to the project. These test cases tests the different scenarios of the design and the randomization technique is used to explore the corner cases.

The master script is also added to the project. The script is designed to execute all the test cases as well as perform coverage and in the provides log files and displays summary.

2. Unit Under Test

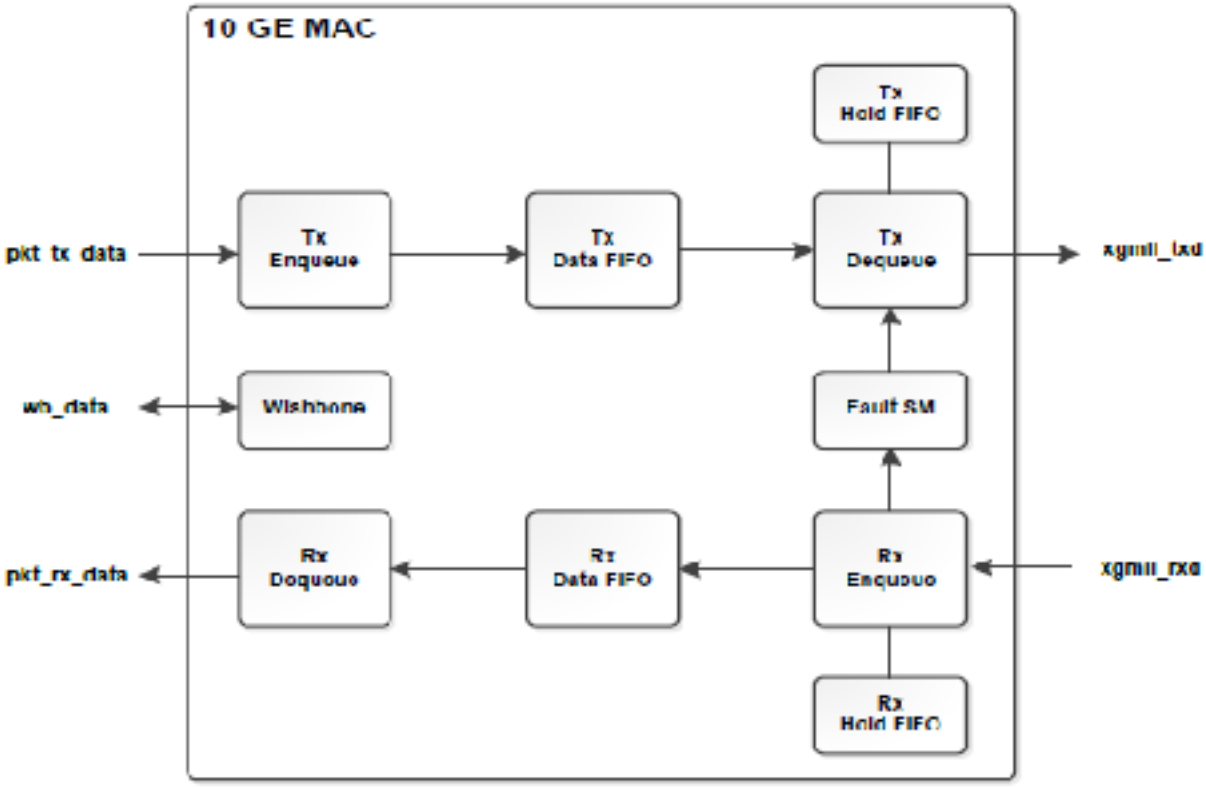
The 10GE Mac design is an Open Source design. The RTL is available freely for anyone who wishes to use it.

There are 3 different interfaces associated with the 10gEthernetMAC design:
1. PKT_TX and PKT_RX interface
2. XGMII_RX and XGMII_TX interface
3. WISHBONE interface

In this project, we will be dealing only with the TX and RX interface (loopback). This is to say that the XGMII interface will be configured in loopback mode.

The wishbone interface present is only initialized and is not been used to verify.

The top level diagram of the design is as shown below:



3. Ethernet Frame

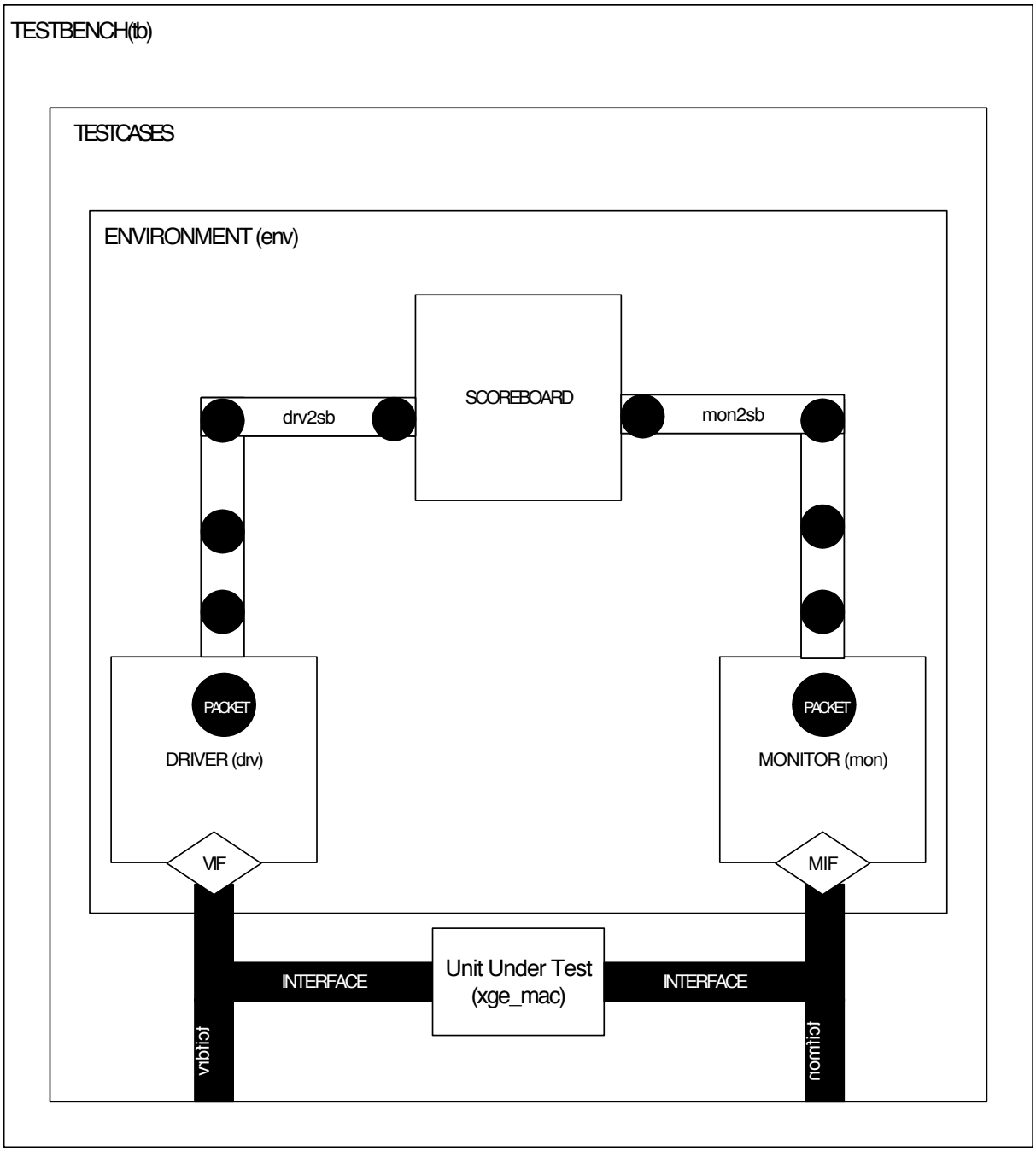
SOP	DST_ADDR	SRC_ADDR	ETHER_TYPE	PAYLOAD	EOP
1 Octet	6 Octet	6 Octet	2 Octet	46-1500 Octets	1 Octet

The structure of ethernet frame according to the IEEE standards are as shown above:

SOP (Start of Packet): The ethernet packet starts with SOP bit which is set high when a new packet transfer starts.
DST_ADDR (Destination Address): This is the Destination MAC address.
SRC_ADDR (Source Address): This is the Source MAC address.
Ether_Type (Ether Type): This denotes the ethernet protocol used in the payload of the frame.
Payload : The minimum payload is 46 octets and maximum is 1500 octets. When the actual payload is less, padding bytes are added by the design. Jumbo frames are the ones with payload greater then 1500 octets.
IPG (Inter Packet Gap): This is the idle time between packets. This is ideally between 10-50 octets of idle time.

4. Testbench Architecture Diagram

Below depicts the architecture of the testbench used in this project.



5. Testbench Components

Verification Environment or **Testbench** is used to check the functional correctness of the Design Under Test (**DUT**) by generating and driving a predefined input sequence to a design, capturing the design output and comparing with-respect-to expected output.

Verification environment is formed by grouping the several components performing specific task/operation.

In verification environment, separate class's will be written to perform specific operation i.e, generating stimulus, driving, monitoring etc. and those class will be named based on the operation. Below is the list of class's that the verification environment have,

Name	Type	Description
driver	class	Generates the stimulus and drives the packet level data inside transaction into pin level (to DUT).
monitor	class	Observes pin level activity on interface signals and converts into packet level which is sent to the components such as scoreboard.
scoreboard	class	Receives data items from monitors and compares with expected values. Expected values can be either golden reference values or generated from reference model.
environment	class	The environment is a container class for grouping higher level components like driver, monitor and scoreboard.
Testcase	program	Test is responsible for, • Configuring the testbench. • Initiate the testbench components construction process. • Initiate the stimulus driving.
tb_xge_mac	module	This is the top most file, which connects the DUT and TestBench. It consists of DUT, Test and interface instances, interface connects the DUT and TestBench.
xge_Interface	interface	This is the interface module, which contains bundled wires connecting testcase, driver, monitor and UUT.
xge_mac	module	This is the top level DUT
packet	class	Creates packet with all its attributes.
Coverage	class	Take care of code coverage and functional coverage.

6. Testcases

1) loopback/testcase.sv
   The first testcase is the “loopback” scenario. This is the exact same testcase as the one that comes with this rudimentary testbench. The main purpose of this particular testcase is to bring up the testbench environment.

2) missing_eop/testcase.sv
   This testcase is a scenario where EOP is missing in the packet. The design is expected to handle it gracefully and receive the next packet normally.

3) missing_sop/testcase.sv
   This testcase is a scenario where SOP is missing in the packet. The design is expected to handle it gracefully by skipping the current packet and receive the next packet normally.

4) undersize_packet/testcase.sv
   This testcase is a scenario where packet size is smaller then the design specs. The design is expected to handle it gracefully by appending 0’s in the end to make it of minimum specification.

5) oversize_packet/testcase.sv
   This testcase is a scenario where packet size is bigger then normal. The design is expected to handle it gracefully by receiving it normally.

6) zero_ipg_packet/testcase.sv
   This testcase is a scenario where inter packet gap is zero. The design is expected to function normally by receiving all the packets.

5. OOP Testbench Directory Tree

The following directories contain all design and testbench related files and documents:

lab-project-10gEthernetMac

- doc/
  - This is where specifications doc, and the testplan document is located.
- rtl/
  - include/ (has verilog defines parameters and other utilities)
  - verilog/ (contains RTL source code for the 10gEthernetMAC design)
- scripts/
  - This is where regression scripts is located.
- sim/
  - verilog/(simulationdirectory)
  - runsim
  - CLEAN
- testbench/
  - verilog/ The top-level testbench, “tb_xge_mac.sv” is located here . Also the component classes are located here as well. The packet class is

located here.

- testcases/ This directory houses all your testcases and VCS logs.
    - o loopback/testcase.sv
    - o missing_eop/testcase.sv
    - o missing_sop/testcase.sv
    - o undersize_packet/testcase.sv
    - o oversize_packet/testcase.sv
    - o zero_ipg_packet/testcase.sv

6. Running Script

In the project directory, master script (regression.py) is present in the scripts directory. This is a python script and needs to be run in the same directory as it has a dependency on library present (Argparse) in the same directory.

The script is designed to execute all the test cases as well as perform coverage and provides log files and displays summary.

Directory location: sf100212@vlsi.ucsc-extension.edu:SV_Project/scripts/regression.py

For help regarding running the script type: python regression.py -h

Example for regressing all testcases: python regression.py all_tests

Example for running individual testcase: python regression.py loopback

For testcases name, refer above section (5).

7. Summary and Conclusion

The design was verified by building the Constrained-Random SystemVerilog OOP testbench environment. The component classes were created as part of Testbench infrastructure. The data variable such as packets were used during runtime to drive and monitor the design. Scoreboarding technique was used to verify the actual packets data versus the expected data. Coverage class was designed to perform functional as well as code coverage of the testbench.

The course and the project provided the necessary skills required for building the SystemVerilog testbench. The professor went in to details of all the testbench components as well as specified the Do's and Don'ts of the programming techniques. The assignments after each class helped in eventually building a complete infrastructure for the project.