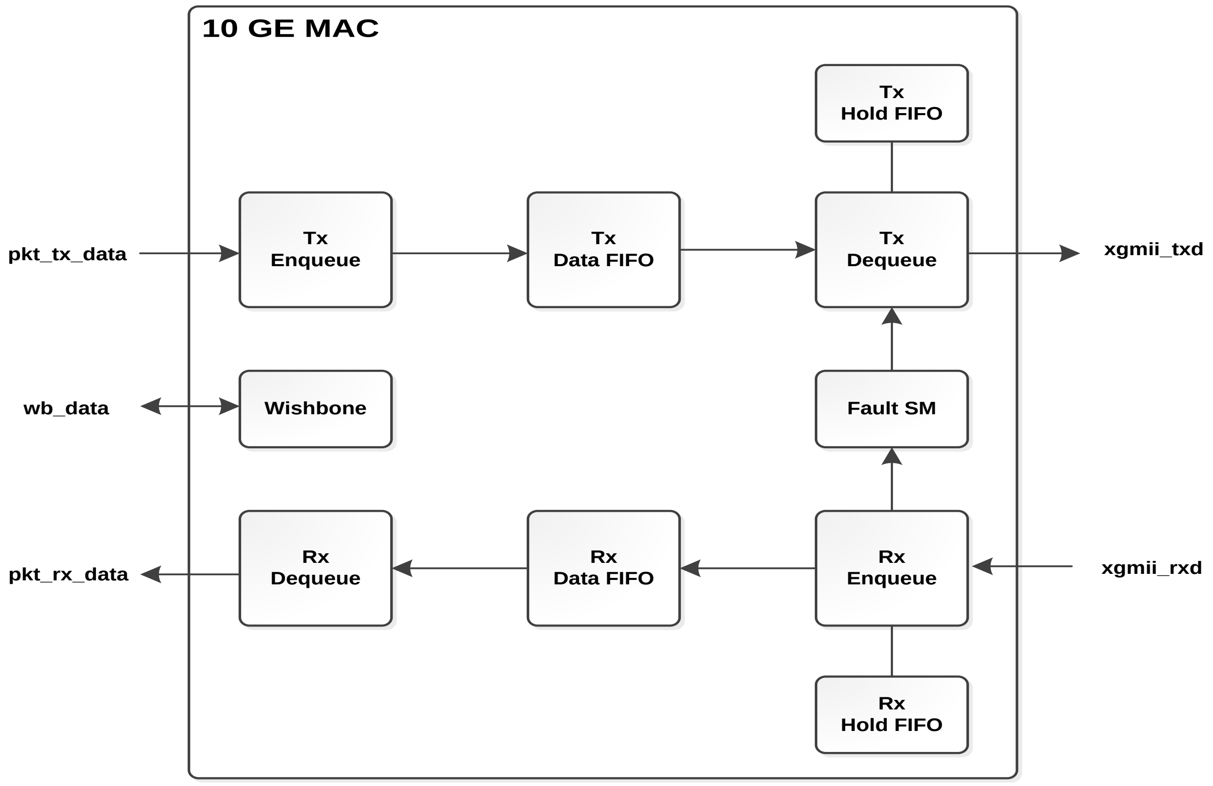
Chapter 11

Advanced Verification with SystemVerilog OOP Testbench Course Project

Objective:

This project is intended as a hands-on opportunity for student 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 10GE Mac design is an Open Source design. The RTL is available freely for anyone who wishes to use it, with the hope that the user community will improve upon the design by providing feedback, suggestions and ideas etc to the Open Source committee



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

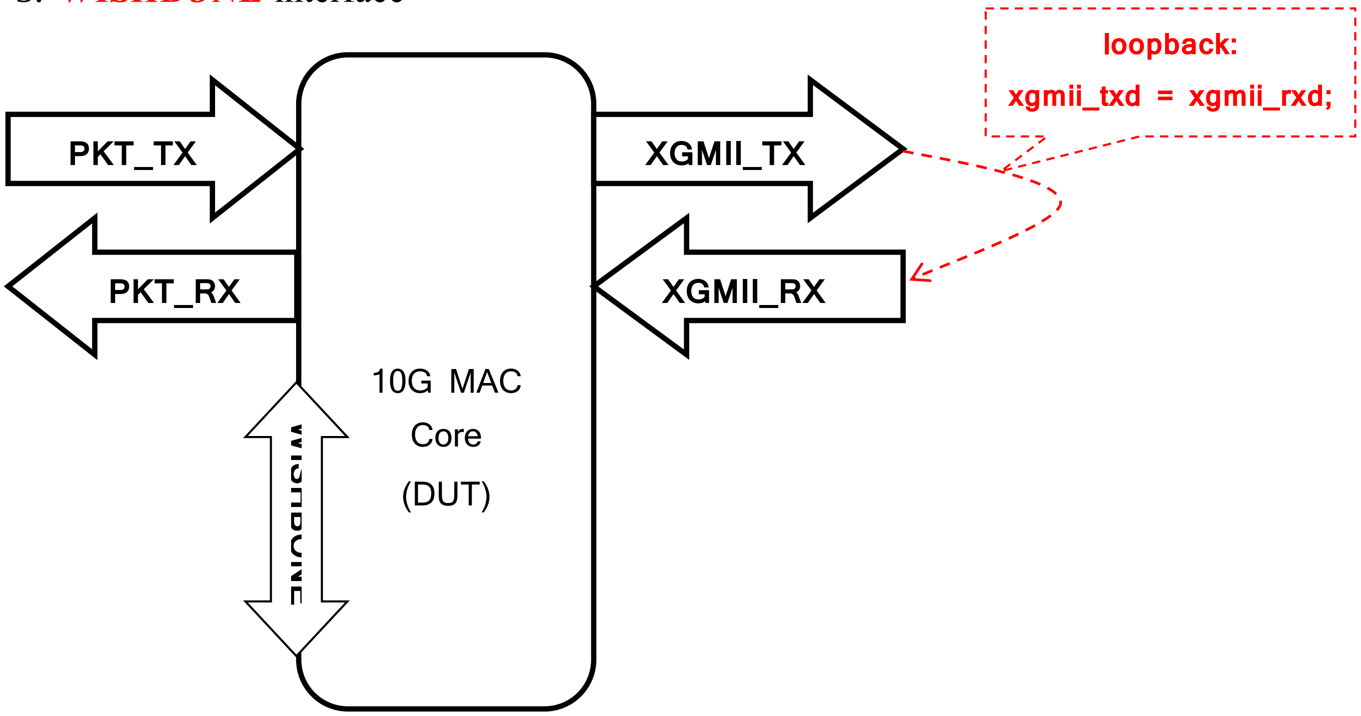
lab-project-10gEthernetMac

- doc/
- rtl/
    - include/ (has verilog defines parameters and other utilities)
    - verilog/ (contains RTL source code for the 10gEthernetMAC design)
- scripts/
    - This is where you will put your regression scripts, Makefile, or any other script-related utilities
- sim/
    - verilog/ (simulation directory)
        - runsim
        - CLEAN
        - You should bring up your testcase and simulation here
- testbench/
    - verilog/
        - The top-level testbench, “tb_xge_mac.sv” is located here
        - The packet content, “packets_txt.txt” is located here. The testbench reads this file to understand what packets to send. You will initially use this as a guide to build your verification environment. Eventually, we will do away with this, and generate packets using a packet.sv class
- testcases/
  This directory houses all your testcases
    - loopback/testcase.sv
        - The first testcase you will develop 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***
    - missing_eop/testcase.sv
    - missing_sop/testcase.sv
    - undersize_packet/testcase.sv
    - oversize_packet/testcase.sv
    - zero_ipg_packet/testcase.sv

In your Testplan, you must have a section that describes each of these testcases to some degree of depth. Your understanding of the design spec would therefore be reflected here as well

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 project is divided into a few mini phases:

1. Understanding the specs and creating a Testplan
2. Understanding existing verification environments and perform basic sanity check
3. Building OOP Testbench
4. Optional section on coverage
5. Summary & Conclusions

You may partner with another student to form a Project Team

PHASE0: RTDS – Read The Damn Spec

PHASE1: Testplan/Verification Plan

- Description of Verification Environment: OOP Testbench
- Block Diagram for Verification Environment
- Description of testcases (start with a loopback test)
- Description of Driver, Monitor, Scoreboard (use block diagrams to explain your environment)
- Coverage: Code Coverage, Functional Coverage, Assertions

PHASE2: Understanding existing environment – getting acquainted with the testbench

- Go to <DIR>/sim/verilog
- Execute the runsim script, observe the outputs
- Study the testbench, tb_xge_mac.sv, located in <DIR>/testbench/verilog
- Get yourself familiar with the "look and feel" of existing setup
- Create a waveform dump. Look at the design hierarchies
    - Pay special attention to the handshakes of the input stimulus: tx_valid, tx_sop, tx_eop etc
    - Also observe how outputs are produced:rx_avail, rx_val, rx_sop, rx_eop

Existing testbench environment is very rudimentary. The testbench has input stimulus co-mingled with instantiation of the Design Under Test. Specifically, the testbench has:

- An instantiation of the 10gEthernetMAC module
- initial blocks
- assign statements
- tasks for reading input files and processing packets

The testbench looks like a bowl of spaghetti – it is messy, noisy, and ugly. It feels very depressing! The first thing you need to do is to separate input stimulus from the actual testbench itself, just like what you have been doing in the homework

- Testbench should only have instantiation of the clocks, RTL module, testcase program block, and instantiation of interfaces
- Testcase should be self-contained inside a stand-alone testcase.sv program block
- Tasks/functions should be self-contained inside a stand-alone file, tasks.sv, or inside an interface if deemed appropriate

In this phase, the main goal is to move existing stimulus from the testbench to a stand-alone testcase.sv program block. Note that the purpose is to move the input stimulus from one place (testbench module) to another (testcase program block). There is no interface or clocking block involved

After you have done so, re-run your simulation, you should see similar outputs compared to the ones without the testcase.sv program block

PHASE3: Building Verification Environment

In this phase, you will start the build-up of class-based OOP Testbench

1. Remodeling existing testbench
   - Move existing stimulus into a stand-alone testcase.sv program block (done in PHASE2)
   - Add interface (use the diagram below to assist you)
   - Add clocking block
2. Create a stand-alone packet.sv class
3. Add driver.sv class. driver.sv must be a container class: it contains the packet.sv class
4. Add env.sv class. env.sv must be a container class
5. Add monitor.sv class
6. Add scoreboard.sv class

Under no circumstances should you simply copy and paste the existing testbench environment and use them as is. They are included just for your reference

For example: DO NOT read the content of the packets from a file, as is the case in the current environment. The existing example is provided to you simply as a reference

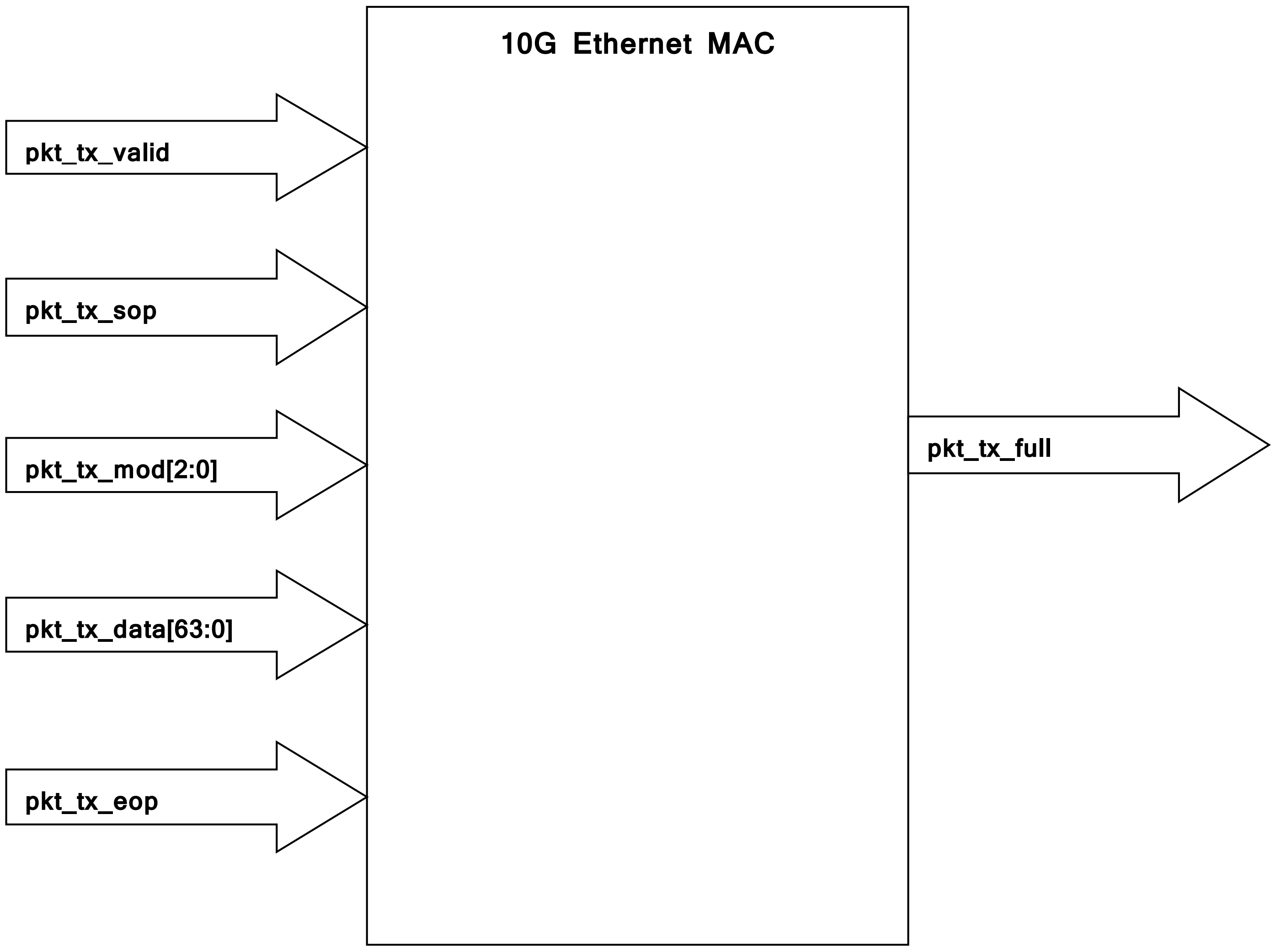
You should build a complete OOP Testbench from scratch

PHASE4: Optional/Extra Credit:

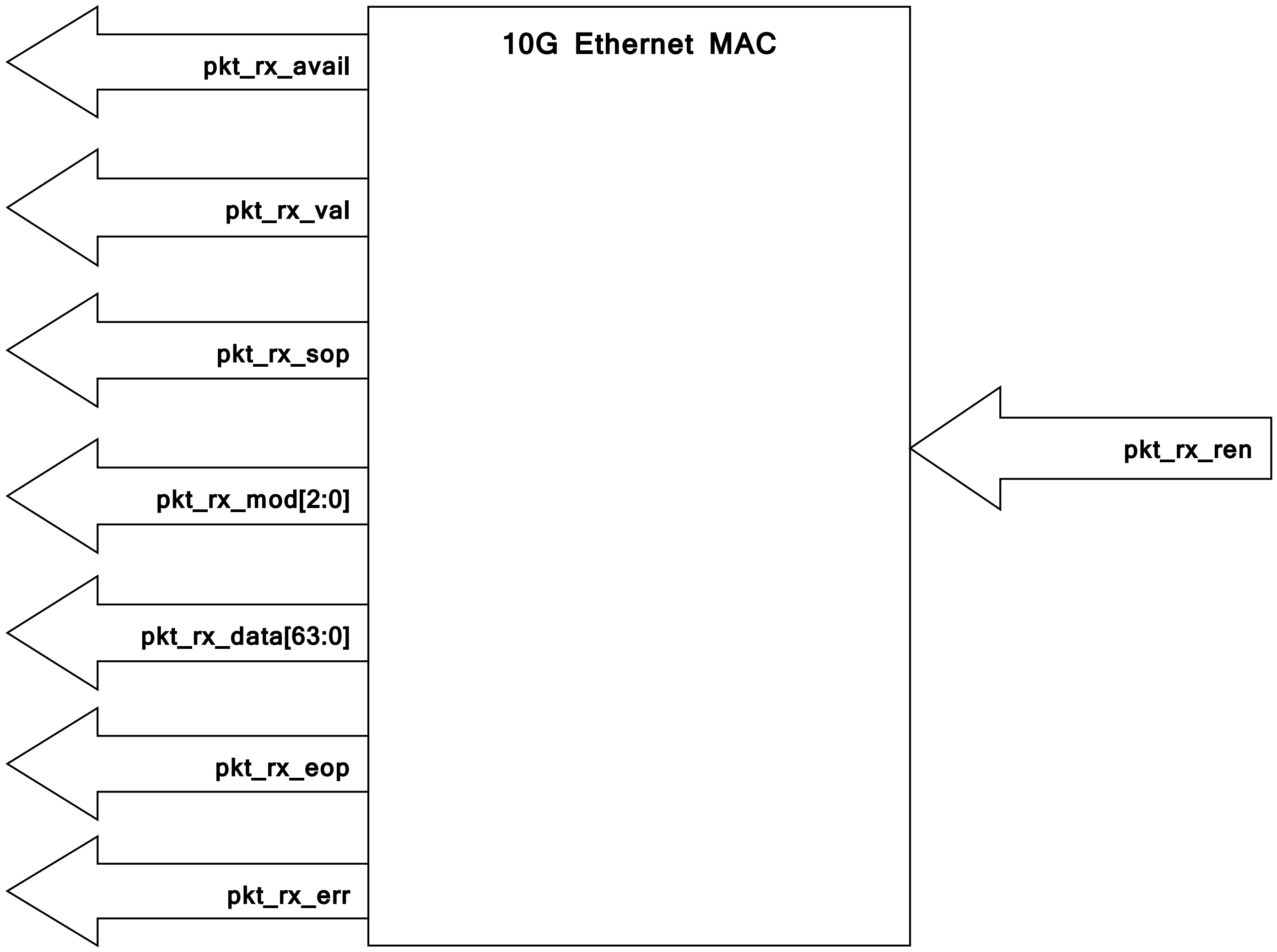
1. Add Coverage class, coverage.sv
2. Add Assertion
3. Regression scripts
4. Possible improvements to your OOP Testbench

10G Ethernet MAC Block Diagram: Transmit Path

Use the following diagram to create an interface. In this project, we will be dealing only with the TX and RX paths



10G Ethernet MAC Block Diagram: Receive Path



What you need to submit:

1. A Testplan in the form of a Word document. This document must contain all the items outlined in PHASE1. You must do 2 things:
   - Provide a hard copy to the instructor
   - Put a soft copy in lab-project-10gEthernetMac/doc/
2. Source code that implements a complete OOP Testbench, as outlined in PHASE2 and PHASE3 and optionally, PHASE4
   - Include a README file describing usages of regression scripts, output log files, summary etc
3. Submit your source code the same way you would your regular homework. Name your directory as "lab-project-10gEthernetMAC"

A sample testbench is provided to you as a reference. DO NOT simply copy and paste the code, rename a few things here and there, or otherwise use them as is. For example, the sample testbench reads the content of the packet from a file. *You MUST NOT do that!* Offenders will receive zero credit for the project

You must build the *OOP Testbench* from scratch!

Due Date:
Testplan – Last Day of Class
Source Code – Last Day of Class + 1 Week

Project Submission Policies:

The due dates for the project are strictly enforced. Late submissions are subject to 50% reduction in credits.

Due Dates:

1. **Testplan – last day of class**
2. **Source Code – last day of class + 1 week**