**TestPlan for 10 Gb/s Ethernet MAC block**

**Final Class Project for USCS 18966: SystemVerilog OOP Testbench (Instructor: Benjamin Ting)**

**Date: 20 Aug 2015**

**Student (author of testplan): John Hubbard** [**jhubbard@nvidia.com**](mailto:jhubbard@nvidia.com)

# Table of Contents

[Table of Contents 2](#_Toc427856629)

[Description of Test Plan 3](#_Toc427856630)

[Description of Test Cases 3](#_Toc427856631)

[Loopback testcase 3](#_Toc427856632)

[Missing EOP (end of packet) testcase 4](#_Toc427856633)

[Missing SOP (start of packet) testcase 4](#_Toc427856634)

[Oversize packet 4](#_Toc427856635)

[Undersize packet 4](#_Toc427856636)

[Zero IPG (inter-packet gap) test case (as time and understanding allow) 4](#_Toc427856637)

[Description of Driver, Monitor, Scoreboard (with block diagrams) 5](#_Toc427856638)

[Driver class 6](#_Toc427856639)

[Monitor class 6](#_Toc427856640)

[Scoreboard class 6](#_Toc427856641)

[Coverage: Code Coverage, Functional Coverage, Assertions 6](#_Toc427856642)

# Description of Test Plan

This test plan exercises the 10 Gb/s Ethernet MAC block. That project may be found at <http://opencores.org/project,ethmac10g> . Throughout this document, the block will be referred to as either the “DUT” (Device Under Test), or the “Ethernet MAC block”.

At a high level, the Ethernet MAC block simply translates 64-bit packet data (plus control lines) to and from a serial bit stream, as shown in Figure 1:



**Fig. 1: Block Diagram of the verification environment**

In order to test such a device, one of the first things to do it to put it into “loopback” mode, by cross-connecting the serial transmit and receive lines. Then the DUT will appear to send packets to itself, and we can verify that the parallel (non-serial) side works properly.

It is also important to verify that the serial bitstream (xgmii, in Figure 1) is in accordance with specifications. However, doing so that is beyond the scope of this document, as our purpose in this course is primarily to build and use an OOP (Object Oriented Programming) test bench in SystemVerilog.

# Description of Test Cases

The following test cases exist:

## Loopback testcase

As shown in Figure 1, simply send packets to the DUT, and receive them back from the DUT. Verify that the packet contents are indentical. Use randomization on:

* Number of packets sent
* Packet data (contents)
* Packet length

## Missing EOP (end of packet) testcase

Send packets as in the loopback testcase, but omit the EOP signal that the DUT requires. Verify that the DUT responds as expected: pkt\_rx\_err should be asserted. Use randomization on:

* Number of packets sent
* Packet data (contents)
* Packet length

## Missing SOP (start of packet) testcase

Send packets as in the loopback testcase, but omit the SOP signal that the DUT requires. Verify that the DUT responds as expected: no packets should be received. Use randomization on:

* Number of packets sent
* Packet data (contents)
* Packet length

## Oversize packet

Send packets as in the loopback testcase, but omit the SOP signal that the DUT requires. Verify that the DUT responds as expected: pkt\_rx\_err should be asserted. Use randomization on:

* Number of packets sent
* Packet data (contents)
* Packet length (constrained to an oversize value)

## Undersize packet

Send packets as in the loopback testcase, but force. Verify that the DUT responds as expected: it should basically work the same as the loopback test. Use randomization on:

* Number of packets sent
* Packet data (contents)
* Packet length (constrained to an undersize value)

## Zero IPG (inter-packet gap) test case (as time and understanding allow)

Send packets as in the loopback testcase, but omit the inter-packet gap that the DUT requires (this seems to require working at the XGMII interface, I’m not yet sure about that). Verify that the DUT responds as expected (I’m not yet sure how it should behave). Use randomization on:

* Number of packets sent
* Packet data (contents)
* Packet length

# Description of Driver, Monitor, Scoreboard (with block diagrams)



**Figure 2: Block diagram of the testbench classes**

The test system consists of a number of directories, each of which contain a testcase.sv file. Each testcase.sv file instantiates a program of the same name: “testcase”. The class relationships are as shown above. Class responsibilities:

## Driver class

Sends stimulus to the DUT (device under test).

## Monitor class

Collects the outputs from the DUT (device under test).

## Scoreboard class

Compares what the Driver class sent, the Monitor class received, and any expected transformations that the DUT should have applied. Provides a pass/fail result for the individual packet.

# Coverage: Code Coverage, Functional Coverage, Assertions

These may be done if time allows, but for now, they are still just a TODO item.