Report

On

Verification of 10G Ethernet Mac Core

Final Project: Verification and Functional Coverage of 10G Ethernet Mac Core

ELEN: 613 SOC Verification

Spring 2019

Professor

Dr. Bruce Greene

Submitted

At

By

Jonathan (Johnny) Trinh

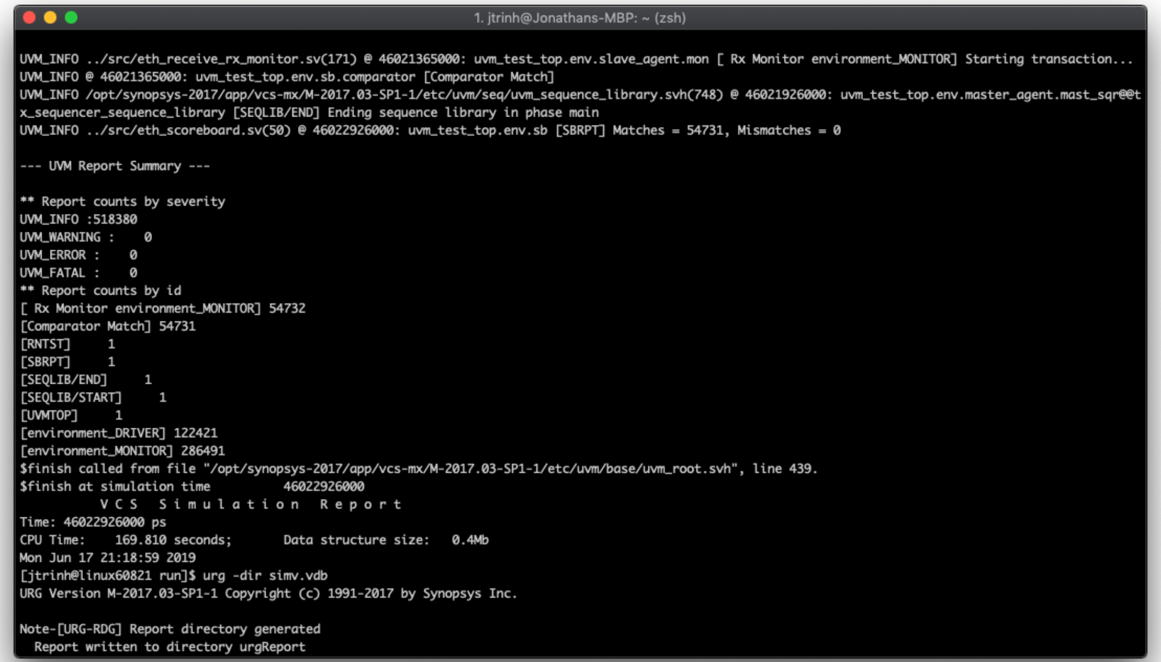
Student ID: W00001252098

Introduction:

In this project, we used the Universal Verification Methodology (UVM) to verify a 10G Ethernet Mac Core. We were given specs that included wishbone, xgmii, transmit, and receive signals. A basic covergroup was provided and additional coverpoints were written (i.e. packet length). We learned to employ object-oriented like programming to drive signals from the transmit side (via a driver) and monitor them (on both transmit and receive).

Procedure:

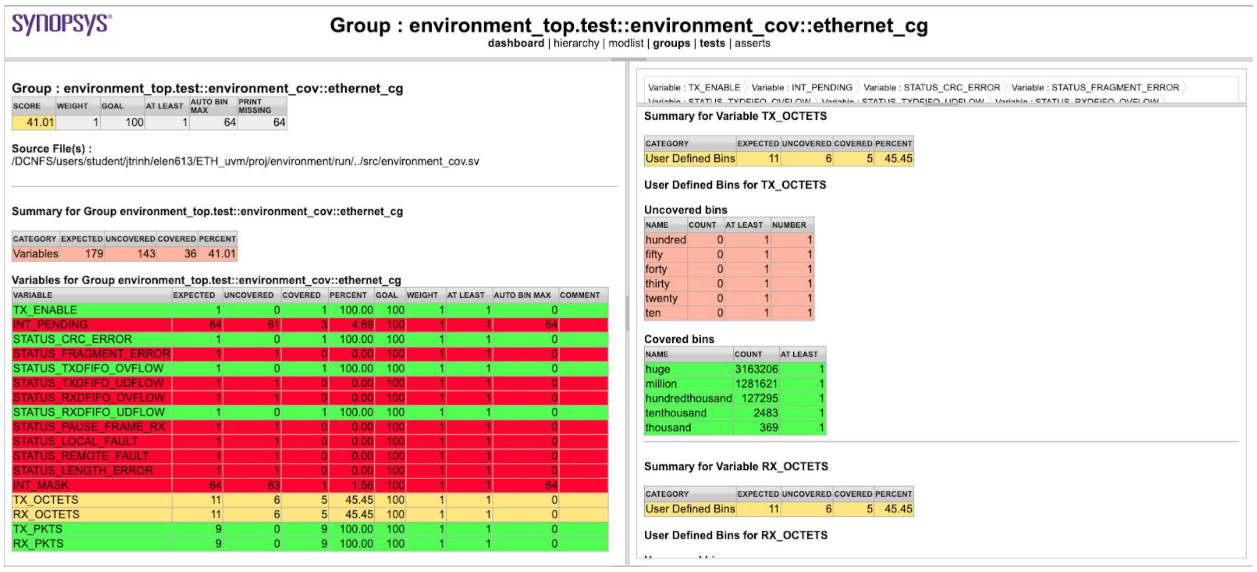
We set up the original environment with 'uvmgen' and generated an interface, driver, monitor, and sequencer for each agent (transmit and receive). A wishbone interface was also added to place down the DUT in the environment. The primary structure employed was a driver on the transmit side, that drove some stimulus onto the DUT. At the same time, a monitor was being used to see what was transmitted. The receive agent is passive so only the monitor was being used. A "data" class acts as a carrier or vehicle for the information going from the transmit to the receive. It packages the data from the transmit for each packet, and sends it to the receive where the data is unravelled from the dynamic array employed in the data class. With 20000 transactions occurring, we get the following:



```
1. jtrinh@Jonathans-MBP: ~ (zsh)

UVM_INFO ../src/eth_receive_rx_monitor.sv(171) @ 46021365000: uvm_test_top.env.slave_agent.mon [ Rx Monitor environment_MONITOR] Starting transaction...
UVM_INFO @ 46021365000: uvm_test_top.env.sb.comparator [Comparator Match]
UVM_INFO /opt/synopsys-2017/app/vcs-mx/M-2017.03-SP1-1/etc/uvm/seq/uvm_sequence_library.svh(748) @ 46021926000: uvm_test_top.env.master_agent.mast_sqr@@t
x_sequencer_sequence_library [SEQLIB/END] Ending sequence library in phase main
UVM_INFO ../src/eth_scoreboard.sv(50) @ 46022926000: uvm_test_top.env.sb [SBRPT] Matches = 54731, Mismatches = 0

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :518380
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[ Rx Monitor environment_MONITOR] 54732
[Comparator Match] 54731
[RNTST]     1
[SBRPT]     1
[SEQLIB/END]     1
[SEQLIB/START]     1
[UVMTOP]     1
[environment_DRIVER] 122421
[environment_MONITOR] 286491
$finish called from file "/opt/synopsys-2017/app/vcs-mx/M-2017.03-SP1-1/etc/uvm/base/uvm_root.svh", line 439.
$finish at simulation time          46022926000
           V C S   S i m u l a t i o n   R e p o r t
Time: 46022926000 ps
CPU Time:     169.810 seconds;       Data structure size:   0.4Mb
Mon Jun 17 21:18:59 2019
[jtrinh@linux60821 run]$ urg -dir simv.vdb
URG Version M-2017.03-SP1-1 Copyright (c) 1991-2017 by Synopsys Inc.

Note-[URG-RDG] Report directory generated
  Report written to directory urgReport
```

Figure 1: Screenshot of the VCS Simulation showing 54371 Matches

When generating the functional coverage for the environment_test, the test shows about a 41% coverage. This default test satisfies a portion of the coverpoints, but not all in the covergroup.



Group : environment_top.test::environment_cov::ethernet_cg

dashboard | hierarchy | modlist | groups | tests | asserts

Group : environment_top.test::environment_cov::ethernet_cg

SCORE	WEIGHT	GOAL	AT LEAST	AUTO BIN MAX	PRINT MISSING
41.01	1	100	1	64	64

Source File(s) :
/DCNFS/users/student/jtrinh/elen613/ETH_uvm/proj/environment/run/../src/environment_cov.sv

Summary for Group environment_top.test::environment_cov::ethernet_cg

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Variables	179	143	36	41.01

Variables for Group environment_top.test::environment_cov::ethernet_cg

VARIABLE	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT	AT LEAST	AUTO BIN MAX	COMMENT
TX_ENABLE	1	0	1	100.00	100	1	1	0	
INT_PENDING	64	61	3	4.69	100	1	1	64	
STATUS_CRC_ERROR	1	0	1	100.00	100	1	1	0	
STATUS_FRAGMENT_ERROR	1	1	0	0.00	100	1	1	0	
STATUS_TXDFIFO_OVFLOW	1	0	1	100.00	100	1	1	0	
STATUS_TXDFIFO_UDFLOW	1	1	0	0.00	100	1	1	0	
STATUS_RXDFIFO_OVFLOW	1	1	0	0.00	100	1	1	0	
STATUS_RXDFIFO_UDFLOW	1	0	1	100.00	100	1	1	0	
STATUS_PAUSE_FRAME_RX	1	1	0	0.00	100	1	1	0	
STATUS_LOCAL_FAULT	1	1	0	0.00	100	1	1	0	
STATUS_REMOTE_FAULT	1	1	0	0.00	100	1	1	0	
STATUS_LENGTH_ERROR	1	1	0	0.00	100	1	1	0	
INT_MASK	64	63	1	1.56	100	1	1	64	
TX_OCTETS	11	6	5	45.45	100	1	1	0	
RX_OCTETS	11	6	5	45.45	100	1	1	0	
TX_PKTS	9	0	9	100.00	100	1	1	0	
RX_PKTS	9	0	9	100.00	100	1	1	0	

Variable : TX_ENABLE 〉 Variable : INT_PENDING 〉 Variable : STATUS_CRC_ERROR 〉 Variable : STATUS_FRAGMENT_ERROR 〉

Summary for Variable TX_OCTETS

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	11	6	5	45.45

User Defined Bins for TX_OCTETS

Uncovered bins

NAME	COUNT	AT LEAST	NUMBER
hundred	0	1	1
fifty	0	1	1
forty	0	1	1
thirty	0	1	1
twenty	0	1	1
ten	0	1	1

Covered bins

NAME	COUNT	AT LEAST
huge	3163206	1
million	1281621	1
hundredthousand	127295	1
tenthousand	2483	1
thousand	369	1

Summary for Variable RX_OCTETS

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	11	6	5	45.45

User Defined Bins for RX_OCTETS

Figure 2: Screenshot of the Functional Coverage Report of Environment Test

Issues to Address:

While the environment_test that came with UVM took us as far as 40% in terms of coverage, much higher coverage was needed. To achieve such a goal, directed tests were to be written.

Approaches Taken:

Within the test module, an additional test was written, where the base_sequence class was subclassed and added to the sequence library. This test, however, used a different “data” class that was defined by “eth_data_oversized” which represents a packet that is too large. This “eth_data_oversized” (defined in eth_transmit_eth_data.sv) is a subclass of the regular “eth_data” class and has constraints that bring it outside the boundary of what is defined in the specifications.

The alternative is to simply define an inline constraint within virtual task body using the *uvm_do_with* function and specify the narrower constraint.

Further Thoughts:

Unfortunately, out of the 11 bins for TX_OCTETS and/or RX_OCTETS, only five were covered. These numbers represents the number of bytes transmitted and since the “large” bins were hit, it would be better to write a directed test with a smaller packet size (or perhaps limit the dynamic allocation to a certain size within the eth_data class). Additionally to address the overflow/underflow issues, one can transmit without enabling the receive side (since there was a signal called *pkt_rx_ren* or the enable). This allows the FIFO to fill up and overflow. Inversely, the DUT can receive while the FIFO is empty causing an underflow error. Achieving both requires a derived class of the monitor that overrides the receive function to simply not receive (in the first case) or receive faster than then transmitting (in the second case).

Takeaways:

1. UVM is a very robust tool for systematically generating every object needed (i.e. drivers, monitors, etc.) though many files need to be touched to maintain consistency.
2. Creation of an environment as well as a new one for every directed test can save changing original code
3. The object-oriented paradigm is extremely prevalent within UVM and is useful for writing separate test cases (whether it be creating a derived class of data, drivers, monitors, etc.)