# Wrapper Module Test Plan and Results

Hongkuan Yu

Jan 2025

# Contents

# 1 DUT

This module is simple, so I include the SystemVerilog code of `dut.sv` below.

```systemverilog
module dut(clk,
        rst_n,
        rxd,
        rx_dv,
        txd,
        tx_en);
input clk;
input rst_n;
input[7:0] rxd;
input rx_dv;
output [7:0] txd;
output tx_en;

reg[7:0] txd;
reg tx_en;

always @(posedge clk) begin
    if(!rst_n) begin
        txd <= 8'b0;
        tx_en <= 1'b0;
    end
    else begin
        txd <= rxd;
        tx_en <= rx_dv;
    end
end
endmodule
```

This `dut.sv` serves as **a simple data wrapper**. The I/O ports of `dut.sv` and their functions are shown in Table 1 below.

| input (wire) | clk | 1 bit | Global clock |
|---|---|---|---|
| input (wire) | rst_n | 1 bit | Active-low reset |
| input (wire) | rxd | 8 bits | Data received |
| input (wire) | rx_dv | 1 bit | Valid for data received |
| output (reg) | txd | 8 bits | Data transmitted |
| output (reg) | tx_en | 1 bit | Valid for data transmitted |

Table 1: I/O ports of `dut.sv` and their functions

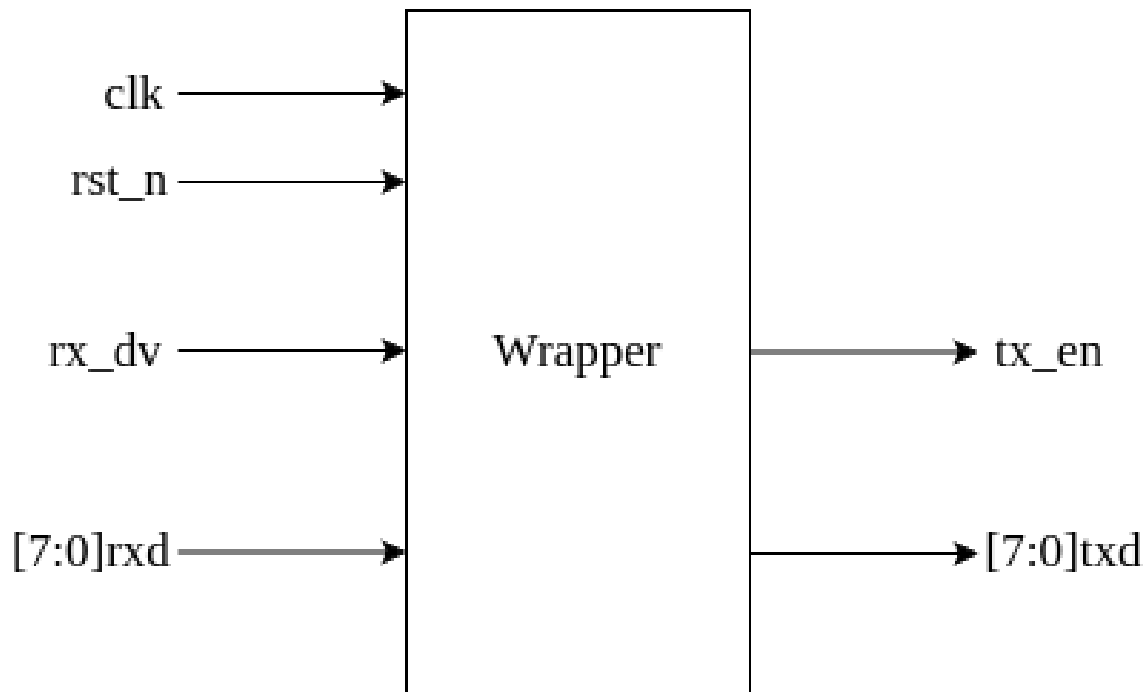The module's I/O graphic overview is shown in Figure 1 below.



Figure 1: Wrapper Module's I/O Overview

3

# 2    Simulation Testbench (Dynamic)

Create `sim_tb` folder holding simulation testbench. Write a testbench `dut_tb.sv` to verify the `dut.sv` dynamically by Synopsys VCS.

Create `sim_tb_script` folder for necessary scripts. Create `filelist.f` and `Makefile` containing normal simulation commands.

Create `sim_tb_results` folder for simulation testbench results. The results of the simulation testbench meet expectations: the output signals `tx_en` and `txd` successfully get the correct value of `rx_dv` and `rxd` after one clock period, separately. The result waveform is shown below in Figure 2 with decimal representation.
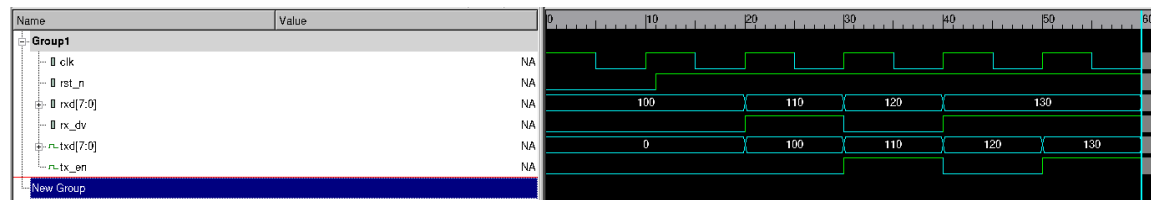


Figure 2: Simulation Testbench Result

# 3 Formal Verification (Static)

Create `sva` folder for formal verification. Write a formal property verification `dut_sva.sv` to verify `dut.sv` statically.

Create `sva_script` folder for necessary scripts. Create `filelist.f` and `Makefile` containing operation commands of formal verification tool.

Create `sva_results` folder for formal verification results.

Since I currently do not have access to Synopsys VC Formal or Cadence Jasper-Gold, I skipped this section, leaving unproved `dut_sva.sv` and template `Makefile` for VCS only.

# 4  UVM (1.1d)

Create `uvm` folder holding all UVM components. The UVM components include (from top to bottom): a base test, environment, model, scoreboard, agent, sequencer, transaction, driver, and monitor. Two different test cases serve as two different sequences. An interface is included for portable design. `top_tb.sv` connects them all.

Create `uvm_script` folder for necessary scripts. Create `filelist.f` and `Makefile` containing UVM simulation commands.

Create `uvm_results` folder for UVM results.

The structure of this UVM platform is shown in Figure 3 below. This platform does not contain the UVM register model.
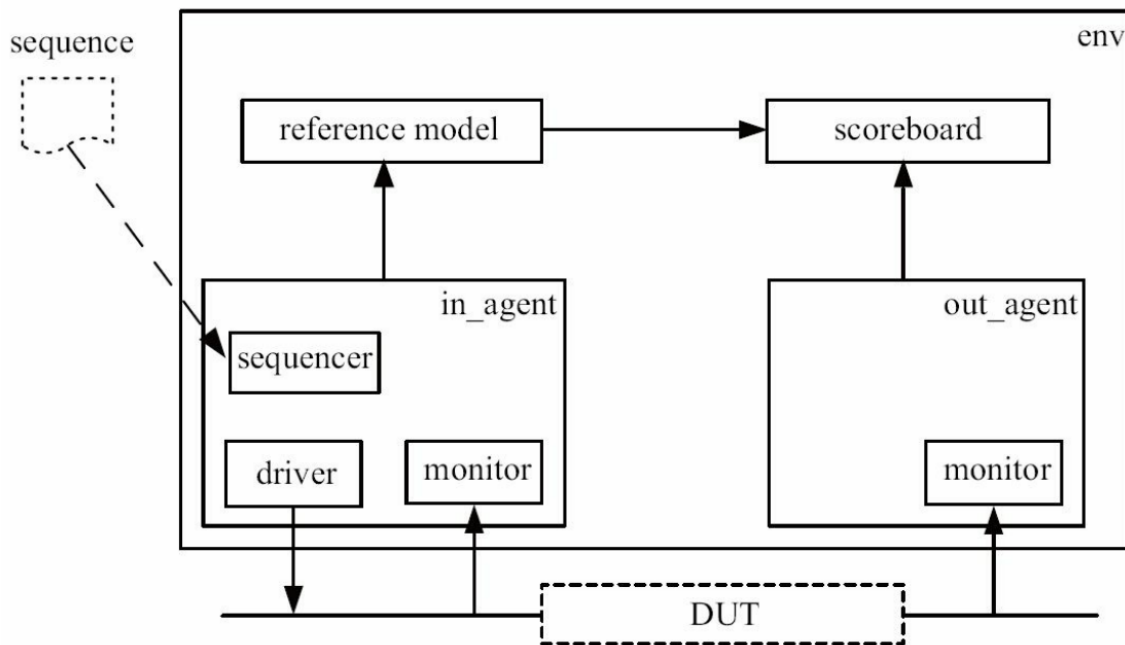


Figure 3: Structure of UVM Platform

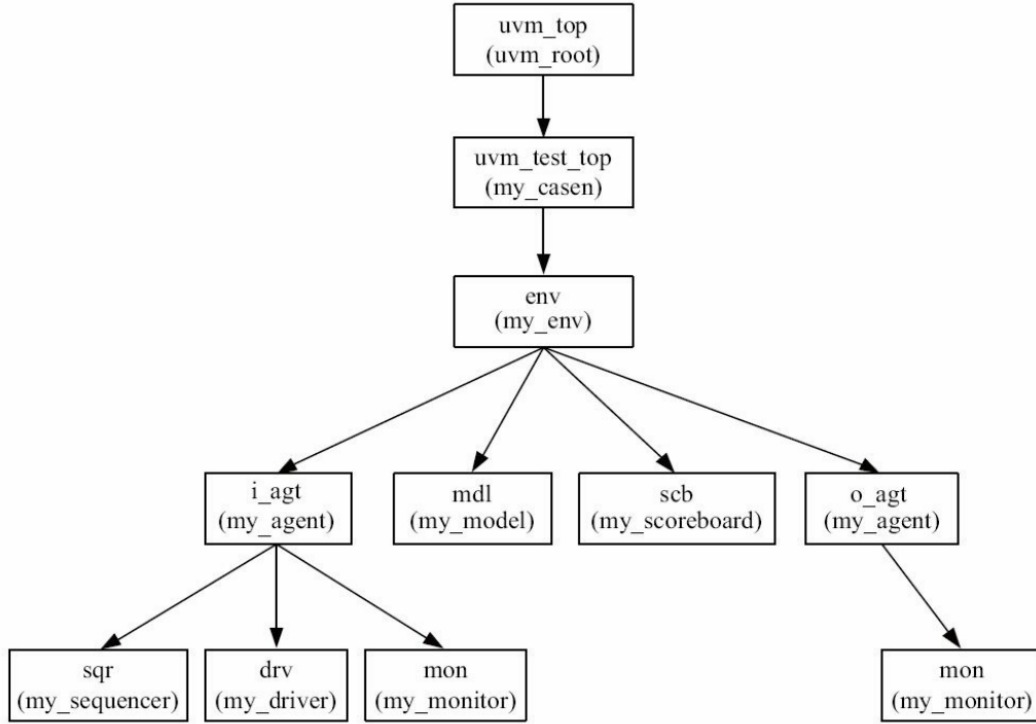This platform's UVM tree is shown in Figure 4 below.



Figure 4: UVM Tree

The transaction mimics the bitstream of an ethernet connection by generating random MAC addresses and other related information. Test case 0 does not restrict the bitstream length. Test case 1 restricts the bitstream length to 60 bytes. Both test cases passed. Please refer to the `.log` file respectively in the `uvm_results` folder.

# References