ECE-593 Fundamentals of Pre-Silicon Validation
System Verilog Test bench Hierarchy

System Verilog- Test bench Block Diagram

Testbench or Verification Environment 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. The block diagram is shown in fig. 1.

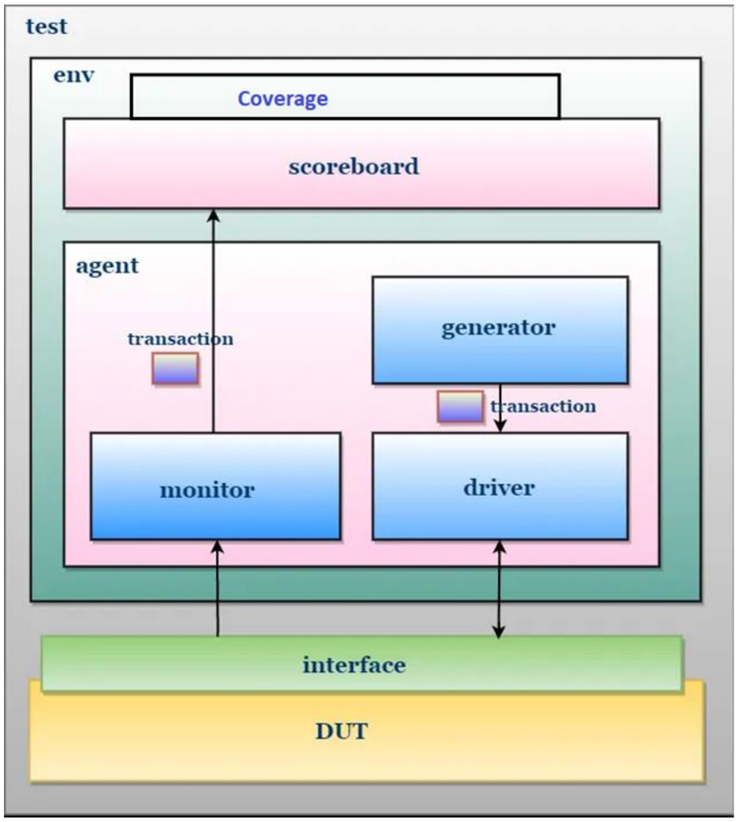
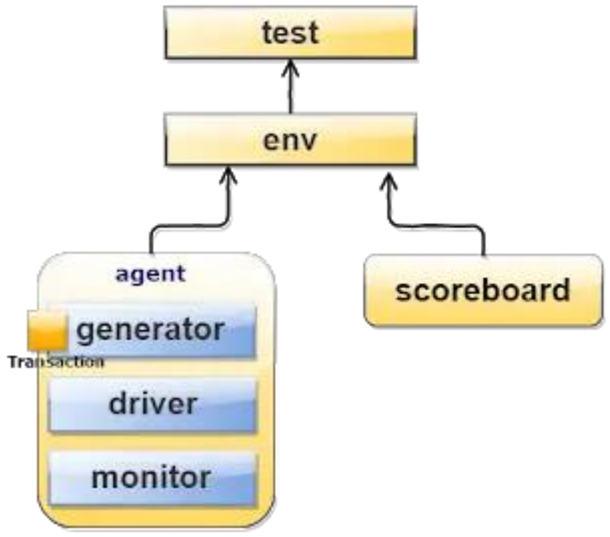


Figure 1: Block diagram System Verilog Verification Environment

Verification environment is a group of classes or components. where each component is performing a specific operation. i.e, generating stimulus, driving, monitoring, etc. and those classes will be named based on the operation.

System Verilog Testbench Hierarchy



Transaction Class

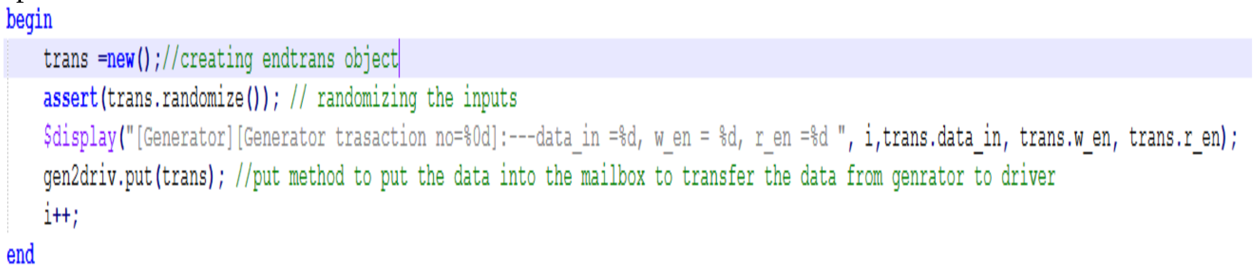
- Transaction classes represent individual transactions or data transfers within the verification environment.
- They encapsulate data and associated properties, providing a convenient way to model stimulus and responses exchanged between the testbench and the DUT.
- Transaction classes can include fields representing various aspects of the transaction, such as address, data, control signals, and timestamps.
- Constraints for read and write enable is mentioned in the transaction class. Weighted distribution is used to equally distribute the read and write enable.

```
//Constraints dor read and write enable
constraint wr {w_en dist {1:/50, 0:/(50)};}
constraint rd {r_en dist {0:/50, 1:/(50)};}
function void post_randomize();
r_en = ~w_en;
endfunction
```

Generator

- Generators are responsible for generating stimulus or test scenarios to drive the DUT.
- They create instances of transaction classes and configure them with appropriate values to stimulate the DUT under various conditions.

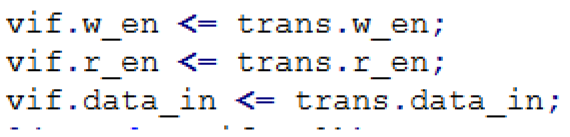
- Generators may operate in a directed manner, where specific test cases are predefined, or in a constrained random manner, where stimuli are generated dynamically based on specified constraints.



```
begin
    trans =new();//creating endtrans object
    assert(trans.randomize()); // randomizing the inputs
    $display("[Generator][Generator trasaction no=%0d]:---data_in =%d, w_en = %d, r_en =%d ", i,trans.data_in, trans.w_en, trans.r_en);
    gen2driv.put(trans); //put method to put the data into the mailbox to transfer the data from genrator to driver
    i++;
end
```

Driver

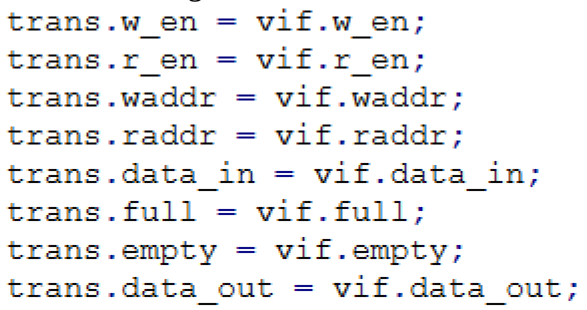
- Drivers are responsible for driving stimulus generated by the generator into the DUT.
- They translate transaction objects or stimulus generated by the testbench into appropriate signal-level activity compatible with the DUT's interface.
- Drivers handle timing considerations, such as clocking and synchronization, to ensure proper interaction with the DUT.



```
vif.w_en <= trans.w_en;
vif.r_en <= trans.r_en;
vif.data_in <= trans.data_in;
```

Monitor

- Monitors observe and capture the behavior of the DUT during simulation.
- They analyze signals or interfaces associated with the DUT and extract relevant information, such as input stimuli, output responses, and internal states.
- Monitors can translate raw signal activity into higher-level transactions for analysis and scoreboard comparison. Transfering the data from interface to mailbox

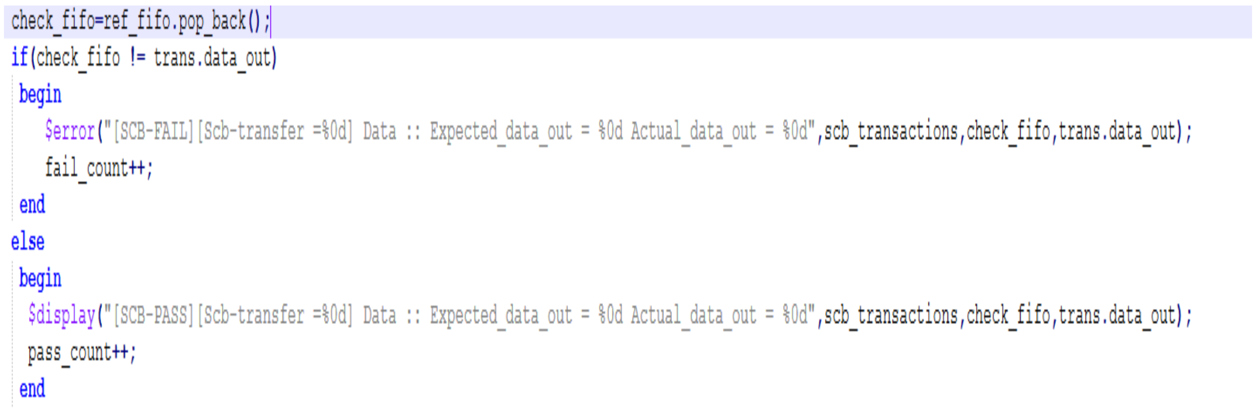


```
trans.w_en = vif.w_en;
trans.r_en = vif.r_en;
trans.waddr = vif.waddr;
trans.raddr = vif.raddr;
trans.data_in = vif.data_in;
trans.full = vif.full;
trans.empty = vif.empty;
trans.data_out = vif.data_out;
```

Scoreboard

- Scoreboards compare expected and observed results to verify the correctness of the DUT's behavior.
- They receive transaction data from both the generator and the monitor and perform checks to ensure that the DUT responds correctly to the provided stimuli.

- Scoreboards may also track coverage metrics to assess the completeness of verification.



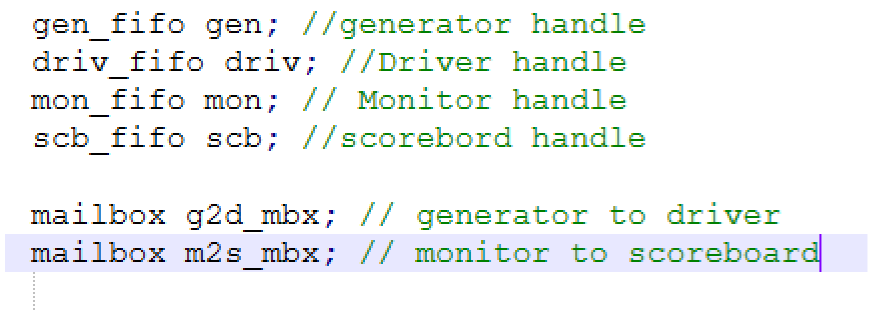
```
check_fifo=ref_fifo.pop_back();
if(check_fifo != trans.data_out)
 begin
   $error("[SCB-FAIL][Scb-transfer =%0d] Data :: Expected_data_out = %0d Actual_data_out = %0d",scb_transactions,check_fifo,trans.data_out);
   fail_count++;
 end
else
 begin
  $display("[SCB-PASS][Scb-transfer =%0d] Data :: Expected_data_out = %0d Actual_data_out = %0d",scb_transactions,check_fifo,trans.data_out);
  pass_count++;
 end
```

Coverage

- Coverage mechanisms track which parts of the design have been exercised during simulation.
- They monitor various aspects of the DUT's behavior, such as code coverage, functional coverage, and assertion coverage.
- Coverage analysis helps identify untested or poorly tested areas of the design, guiding verification efforts to improve coverage completeness.

Environment

The environment represents the primary container for organizing and managing the verification components. It includes agents, drivers, monitors, sequencers, scoreboard, and other necessary modules. The environment is responsible for creating and configuring these components and ensuring proper communication among them.



```
gen_fifo gen; //generator handle
driv_fifo driv; //Driver handle
mon_fifo mon; // Monitor handle
scb_fifo scb; //scorebord handle

mailbox g2d_mbx; // generator to driver
mailbox m2s_mbx; // monitor to scoreboard
```

Test:

Tests are sequences of stimuli applied to the DUT to verify its functionality and performance. They consist of a series of transactions generated by the testbench to exercise different features and scenarios of the DUT. Tests are typically organized into test cases or test suites, each targeting specific aspects of the DUT's behavior.

tb_top:

tb_top refers to the top-level module of the testbench hierarchy. It serves as the central control unit that instantiates and orchestrates various components of the verification environment, including generators, monitors, drivers, scoreboards, and coverage collectors. tb_top coordinates the execution of tests, manages simulation resources, and facilitates communication between different parts of the testbench. It provides a unified interface for interacting with the DUT and orchestrating the verification process.

IMPLEMENTATION:

System Verilog environment for asynchronous fifo is implemented by combining all the components of the hierarchy obtained the desired results.

- The constraints for read enable and write enable are mentioned in the transaction class in order to not assert both of them at the same time.
- The generator generated the randomized inputs such as data_in, w_en, r_en and is passed to the driver using transaction class.
- The driver drives the stimulus to the dut through virtual interface.
- The generated outputs and inputs from the dut is collected by the monitor and sent to the scoreboard using transaction.
- The scoreboard compares the data_in and data_out with the reference model and displays whether the test is passed or failed.
- Whenever write enable is asserted, the data will be written to the fifo memory and the burst details are displayed in each component. The write method which is called in the monitor is responsible for transferring the data to the scoreboard.
- Whenever read enable is asserted, the data will be read from the fifo memory and the burst details are displayed in each component. This will happen in the run phase of the scoreboard.

CONTRIBUTION OF TEAM MEMBERS:

- Bhavani: Implemented monitor, generator, transaction class successfully.
- Asritha: Implemented scoreboard, driver successfully.
- Vijay: Implemented Environment, Test, TestBench Top successfully.
- Asritha and Bhavani together tested the burst passing from generator-> driver-> monitor thoroughly.
- Vijay and Bhavani together tested the burst passing from the monitor ->scoreboard thoroughly.
- Asritha and Vijay : Implemented the coverage and verified the entire design thoroughly and wrote the verification plan document.

CHALLENGES FACED:

- During the initial run, 'x' was initially received as the value of data_out due to improper connection.

- The read address increment was incorrectly set, resulting in incorrect values.

- Correct full and empty values were not received after read and write transfers.

- In the monitor and driver, the updated data_out wasn't obtained after each read transfer.

- There was a lack of implementation of error correction and detection mechanisms, which is essential for verifying the asynchronous FIFO correctly.

- Race conditions between write and read pointers were observed.

- Difficulty was encountered in resetting back to normal operation.

CHALLENGES RESOLVED:

- Thoroughly examined the entire testbench, detecting improper connections between classes, and rectified the issue by establishing proper connections between components.
- Discovered incorrect values for the read address increment, attributed to excessive delays in certain testbench components. This was addressed by reducing delays and ensuring proper flow.
- Addressed challenges in achieving full and empty conditions by meticulously verifying connections between the design and testbench via interfaces. Removing clocking blocks revealed discrepancies, enabling corrections for receiving accurate values in the transcript.
- Noticed previous values persisting in the monitor for every two read operations, which was resolved by synchronizing delays between the driver and monitor.
- Uncovered discrepancies in values transferring from data_in to data_out, causing race conditions between read and write pointers. This was mitigated through thorough code examination and modification in both the design and testbench.
- Rectified an incorrect reset value assigned in the top-level testbench of the UVM verification hierarchy by accurately specifying the reset signal's value.