**PES University Electronic City Campus**

Hosur Rd, Konappana Agrahara, Electronic City, Bengaluru, Karnataka 560100



A Project Report on

*“Layered Testbench for S R latch with enable”*

Submitted by:

* **G Nikhil Kumar Reddy (PES2UG20EC152)**
* **Darshan (PES2UG20EC143)**

Under the guidance of:

**Assistant professor Vinay Reddy Sir**

**Department of Electronics and Communication Engineering**

**PES UNIVERSITY**

Electronic City, Bangalore-560100

2021 - 2022

**Introduction**

The layered testbench is the heart of the verification environment:

• The lowest layer is the signal layer that connects the testbench to

the RTL design. It consists of interface, clocking, and modport

constructs.

• The command layer contains lower-level driver and monitor

components, as well as the assertions (properties) that check

design intent. This layer provides a transaction-level interface to

the layer above and drives the physical pins via the signal layer.

• The functional layer contains higher-level driver and monitor

components, as well as the self-checking structure that

determines whether tests pass or fail. Additional checking, for

example protocol checkers, can span the command and

functional layers.

• The scenario layer uses generators to produce streams or

sequences of transactions that are applied to the functional layer.

The generators have a set of weights, constraints or scenarios

specified by the test layer. The randomness of constrained

random testing is introduced within this layer.

• Finally, the test layer is where the tests are located. The tests can

define new sequences of transactions using the scenario layer,

synchronize multiple transaction streams, generate sequences by

interacting directly with the functional or command layers, or

supply directed stimulus directly to the command layer.

Although this layered testbench is designed primarily for using

constrained-random stimulus generation, it supports manual directed

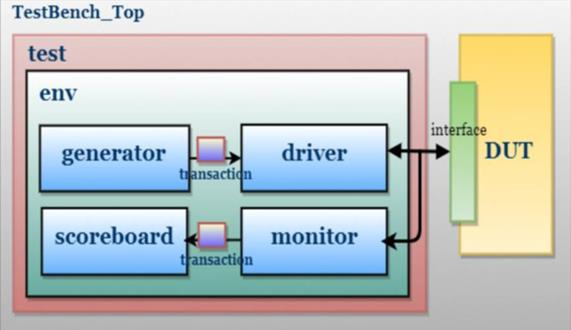
tests as well. The upper-left portion of Figure 1 shows a path running

directly from the tests to the driver, bypassing the generator entirely.

This allows a verification engineer to generate transactions directly

without setting up constrained-random scenarios.

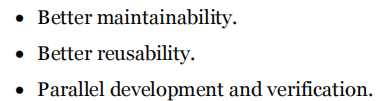
**TESTBENCH ARCHITECTURE**



**OBJECTIVE**

Our objective is to write a layered testbench for S R latch with enable as input by giving random stimiulus.

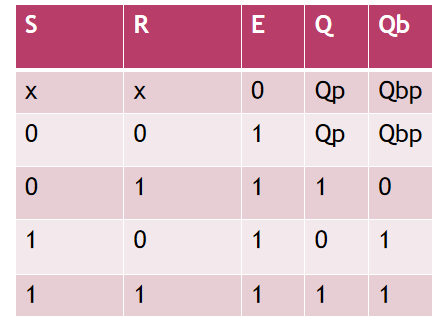
**BENEFITS OF LAYERED TESTBENCH**



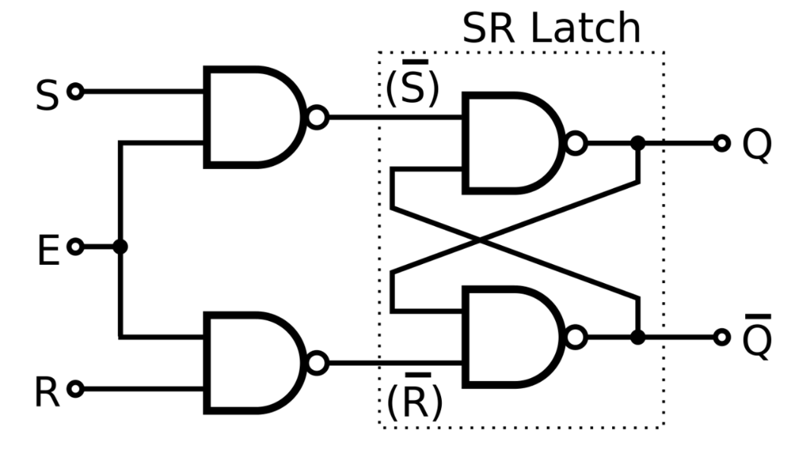
**DESIGN**

**The gated SR latch is a simple extension of the SR latch which provides an Enable line which must be driven high before data can be latched. Even though a control line is now required, the SR latch is not synchronous, because the inputs can change the output if the enable line is held high at length.**

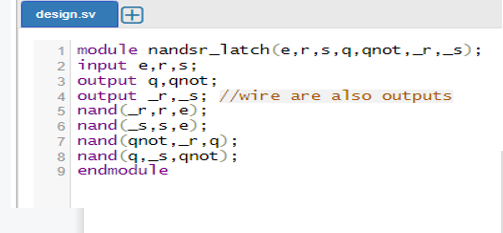
**TRUTH TABLE**



**LOGIC DIAGRAM OF GATED SR LATCH**



**DESIGN System Verilog code**



**Transaction Class**

* It is a packet with signals of random variables passed to all the classes.

• Field required to generate stimulus are declared in transaction

class

• Transaction class can also be used as placeholder for the activity

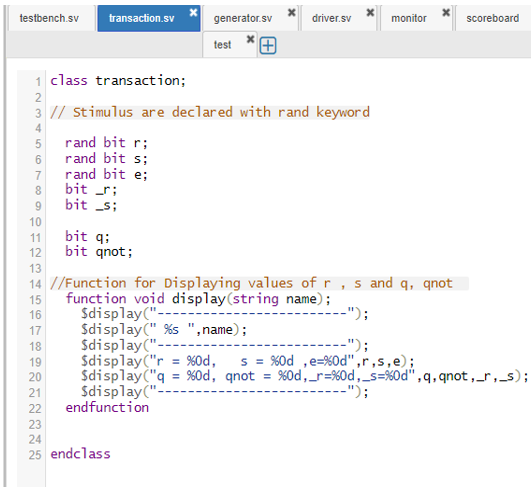
monitored by monitor on DUT signals

• Generator sends transactions to be interpreted by the driver to

provide stimulus to the DUT through the interface

• Monitor interprets outputs of the DUT into transaction then

pushes them to the scoreboard



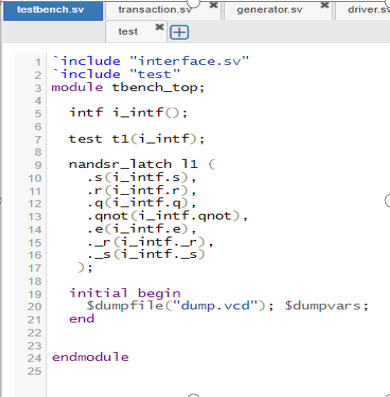
Testbench

• Test Bench is the top-level entity and a module. Instantiates the

DUT, interface and the test

• The test is a module that instantiates the environment and calls

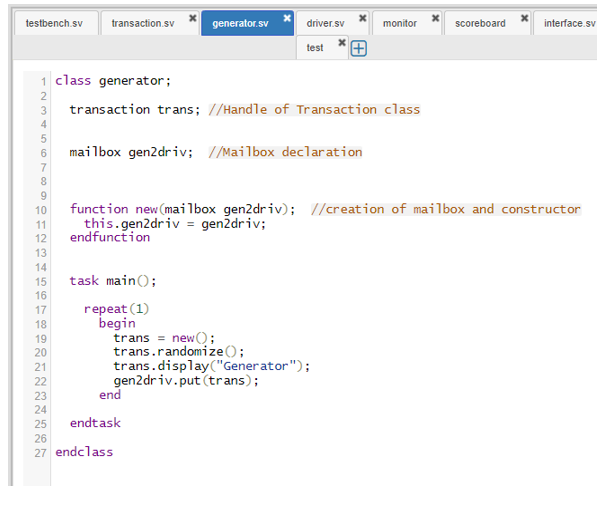
its run() procedure.



**GENERATOR**

**- Generates the random stimulus for DUT.**

• Sends generated transactions to driver over mailbox.

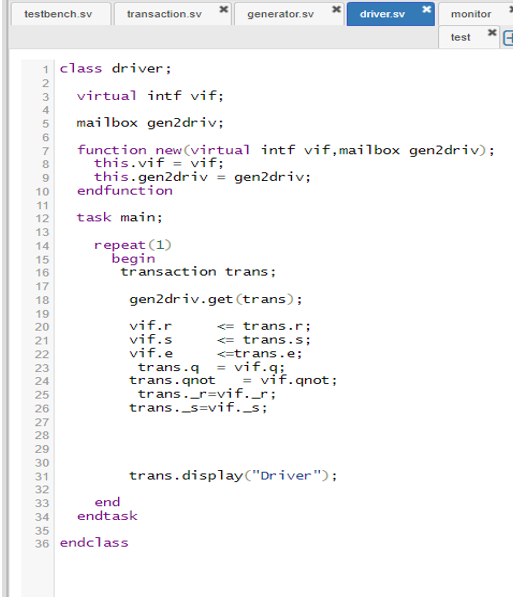


DRIVER

• Driver Class receives the stimulus from a generator in the form of

a transaction through a shared mailbox

• Driver converts transaction level data to signal level activity



**INTERFACE**

• SystemVerilog construct to encapsulate communication between

modules

• Can contain modports which provides direction information for

module interface ports and controls the user of tasks and controls

the uses of tasks and functions within certain modules

• Shared by multiple entities (driver, monitor, etc.)

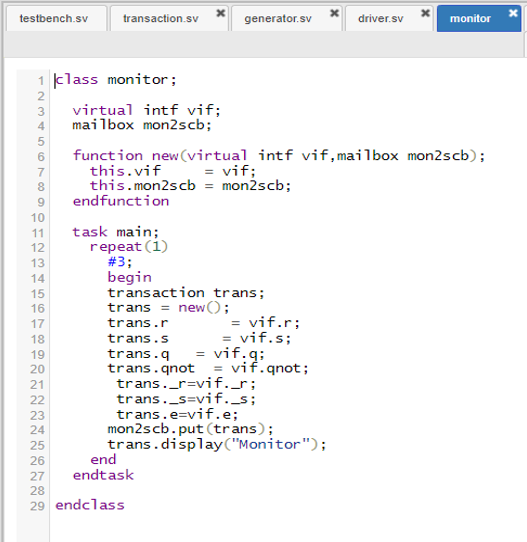


Monitor

• Samples the Interface signals and convert signal level activity to

transaction level

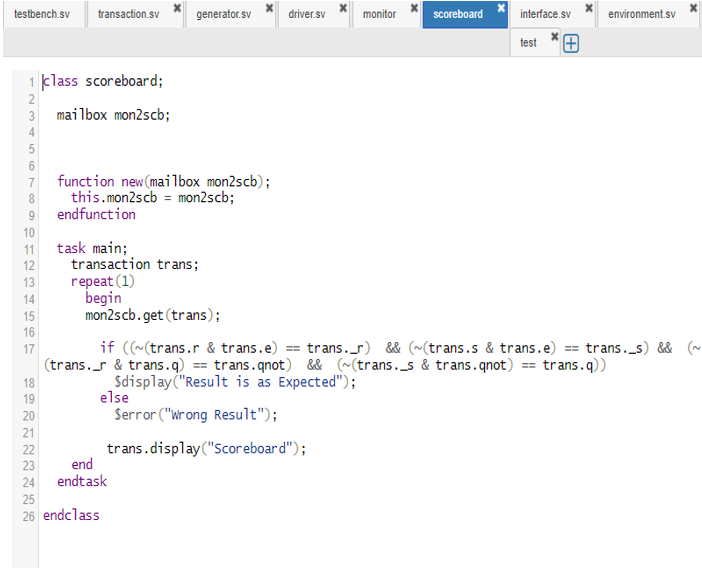
• Sends transaction to scoreboard via mailbox



SCOREBOARD

• Scoreboard receives the sampled transaction from the monitor

• Used to perform the actual verification, is the output correct?



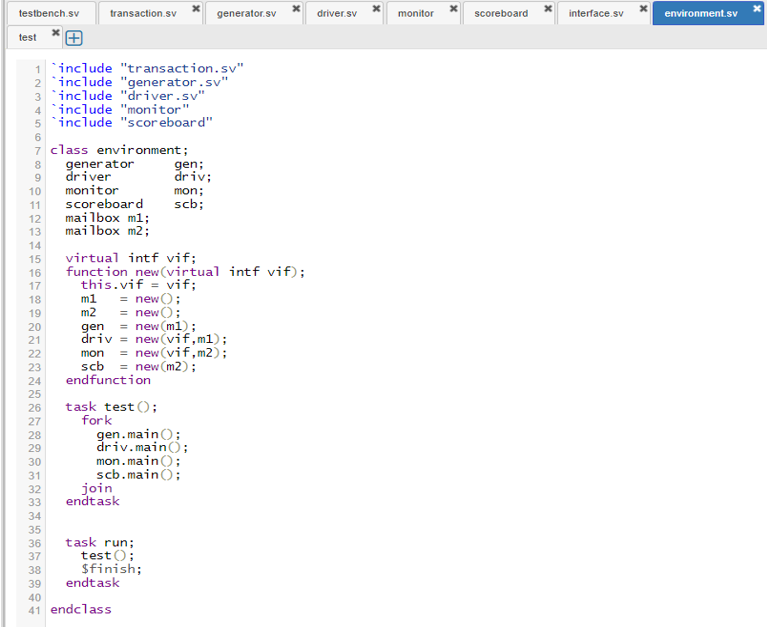
**ENVIRONMENT**

• Contains instances of the generator, driver, monitor, and

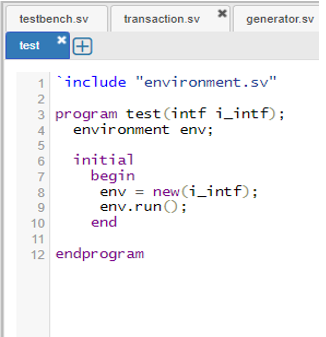
scoreboard, connects generators to drivers and monitors to

scoreboard with mailboxes, and provides instance of the interface

to all

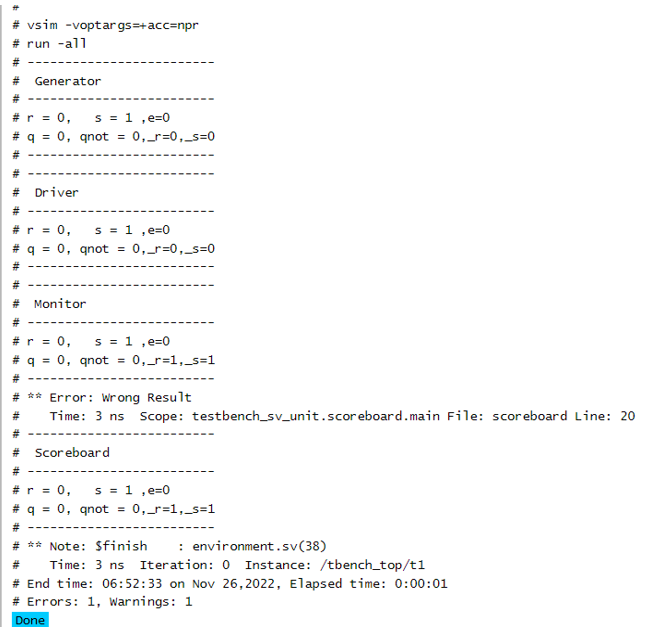


TEST module

• The test is a module that instantiates the environment and calls

its run() procedure.

**OUTPUT**



CONCLUSION

Since the enable signal is low u get the result wrong.Hence to get the proper results the Enable input must always be High

**CODE LINK**

https://www.edaplayground.com/x/qQhS

EDA playground is the tool used.