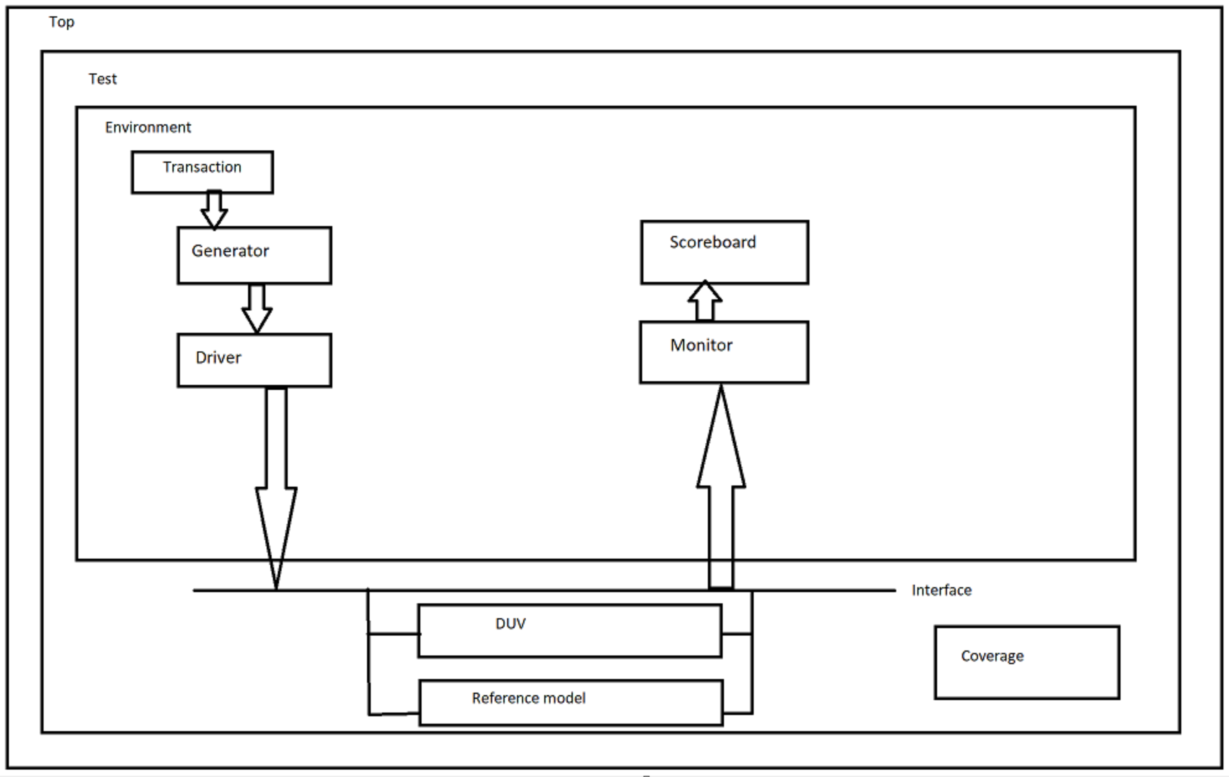
Portland State UNIVERSITY

ECE593: PROJECT REPORT
Verification of JTAG TAP controller

Team members:
1. Mangala Bhat (PSUID: 969559278)
2. Piyush Purwat (PSU ID : 941496418)
3. Venkata Sai Pranav Twarakavi (PSU ID: 951484914)

Abstract: In this project,we verified different instruction execution using the JTAG tap controller , verification included driving the DUV to states in order to program the instruction register in the design and make sure that appropriate state is then tms is driven to facilitate the required data register.

Testbench architecture:



In the testbench , we emply the architecture shown above , the tms sequence intended to be driven is taken by extracting each element of a randomized array ,constrained according to the instruction we are using.The test program runs the environment method which in turn builds and runs generator ,driver, monitor and scoreboard methods.The generator drives data through the interface , after processing these inputs , the output of the DUV is collected through the monitor and compared with the reference model in the scoreboard , the run ends after all data is driven and all checks in scoreboard are done.

Communication in Testbench:
Communication between the classes present in the testbench takes place through mailbox.In this testbench it is used in two places .
i)Generator and Driver
ii)Monitor and scoreboard

Coverage:

Coverage is implemented in a separate module , we bind this module to the design under verification using bind construct

Technique applied for stimulus:
we extend the transaction class from within a test program and write constraints for required fields and drive the environment task from this file.
To make generating stimulus generation easy , we randomize a dynamic array(to hold all scan chain values and tms values) and iterate through this array to provide stimulus to the interface through the driver .Additionally, free running clock and reset is provided through top module

Results :

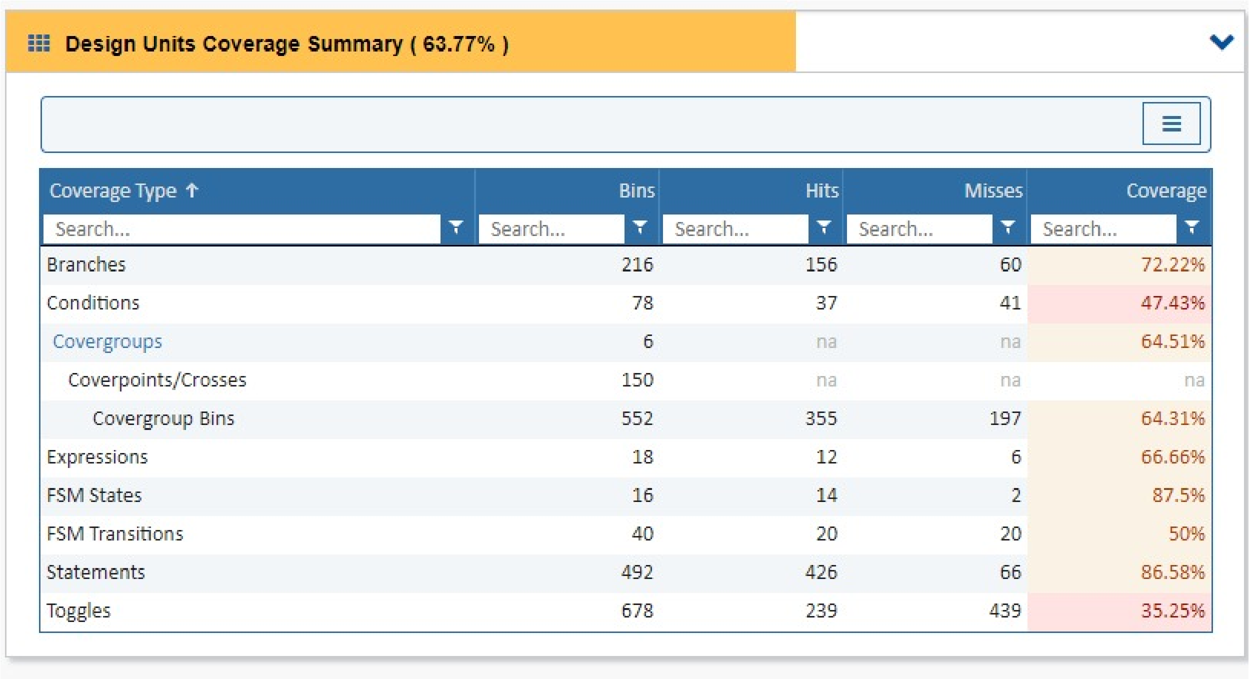
Testcases are passing for valid instructions.

Description of test bench/tests executed:

In test program present in the top module is instantiated and the following actions take place:
1.Test program contains classes extended from the transaction class we apply constrains Here, subsequently environment class in instantiated and its method is run
2.In the env run method we further run the methods of all classes contained by the Environment and take care of required connections between the classes
3.First to run is the generator,it is required for generating the constrained stimulus required

4.Second to run is the driver , it takes data from the monitor and drives it to DUV through the Interface.
5.Third to run is the monitor method that is responsible for collecting the output stimulus From the DUV through the interface.
6.Fourth to run is the scoreboard method that is responsible for comparing the output Signatures from both the reference model and the DUV.
7.Last to run is the end of test task that terminates the simulation once the testcase is Complete.

Assessment of the coverage :


Design Units Coverage Summary (63.77%)

Coverage Type ↑	Bins	Hits	Misses	Coverage
Branches	216	156	60	72.22%
Conditions	78	37	41	47.43%
Covergroups	6	na	na	64.51%
Coverpoints/Crosses	150	na	na	na
Covergroup Bins	552	355	197	64.31%
Expressions	18	12	6	66.66%
FSM States	16	14	2	87.5%
FSM Transitions	40	20	20	50%
Statements	492	426	66	86.58%
Toggles	678	239	439	35.25%

Instruction testcases provide large coverage interms of FSM states , nearly 87.5% is covered except for pause states.since design traves through all states for instruction execution.

Toggles have a lower coverage since these are bit values

Above coverage is obtained by running a random test for 100000 transactions over 100000030 ns simulation time.

Coverage for conditional statements is lower and shows that random tests do Not triggermore than half the control values .

The possibility of getting more coverage is higher depending on the instruction executed and will be lower ifthe number of instructions defined is less.

Next possible developments:

Debugging added Scan chain and integration with multiple designs

check if the boundary values can be controllable and observable.

Run coverage for all combinations of instructions

effort required:

90 hrs

Challenges encountered:

1.Giving stimulus to TAP inputs:

Our prior implementation plan included taking a probe signal from DUV to driver to drive TDI only when the we reached the required internal state .This implementation did not wor
Current implementation resolves this issue by using randomized arrays and constraining it t
drive valid data at required states depending On the tms sequence provided.

2.Limited time to debug and execute scancells and additional rtl logic .Integration and Testing for the same is incomplete.

3.working together on different files led to lot of debugging while integrating them,common

Included bad reference to handles and synchronization issues.

4.Ramping up about testability and why we require observalbility and controllability in real Hardware ,Identify faults that instructions are capable of capturing and thinking about how Verify test infrastructure using pre-silicon techniques.

pre-existing code/ideas :

References from class material,cfv textbook and websites like chipverify,verifcation to Architecture the testbench structure.

Tests run:

- 1 random testcase , providing large amount of stimulus to all primary input pins
- Constrained random testcases to execute all instructions and set up tap for different modes
- Contrained random testcase to drive a sequence of two instructions
- Constrained random testcase to keep design in IR states for a large amount of simulation time.
- Constrained random testcase to keep design in DR states for a large amount of simulation time
- Testcase to check extensilibility by modifying tap defines in design
- Testcase to traverse states not used in instructions and drive random tms sequence at that point.

Bugs Found: None
Expected response obtained for given stimulus

Injecting bugs:

-Invalid instructions to check whether the design is executing the
Default options
-Modify state variables and run against reference model(testcase fails in this case)

Next steps:

1.Enhance design and build scan chains and employ multiple JTAG controllers to Verify different instructions.

2.Define own instructions for the instruction register and verify them with related testcases