FREE –UVM FIFO Verification IP User’s Manual

Version 2.0, Jun 2016

YanSolutions®

Revision History

Name	Version	Date	Comment
haykp	1.0	05/04/2016	First Main Release, full explanation of the FIFO VIP TB.
haykp	2.0	06/04/2016	Added the Coverage and Assertions Part

Contents

1. General

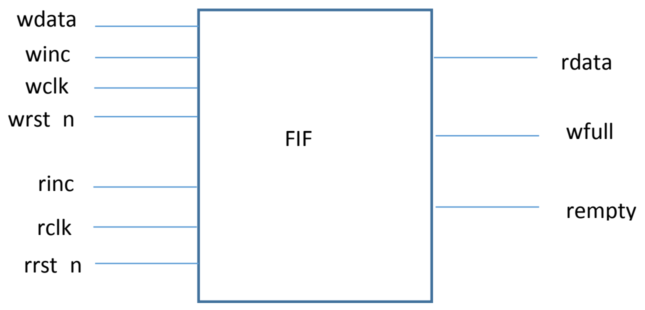
This is Verification plan for the FIFO devices. The plan analyses all possible cases and scenarios for FIFO testing.

2. Acronyms

Acronym	Meaning	Other
DUT	Design Under Test	
TB	Test-bench	

3. FIFO Basics

Below is shown the basic symbol of the asynchronous FIFO:



The FIFO detail description and RTL code can be found from Clifford E. Cummings "Simulation and Synthesis Techniques for Asynchronous FIFO Design" Article. Reference link to that article is:

http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf

3.1 Possible Configurations of the FIFO

For this test-bench we will use asynchronous FIFO as a DUT. However the developed test-bench should also support all possible configurations of the FIFO. In general following FIFO combinations exist

1. Asynchronous WRITE and READ FIFO
2. Synchronous WRITE and READ FIFO
3. Synchronous WRITE and READ FIFO, without WE input
4. Asynchronous WRITE and READ FIFO, without full and empty outputs

For all these possible FIFOs, the developed test-bench must easily support with a little changes.

3.2 FIFO Parameters

The FIFO which is used as DUT has following parameters

a. Input data size
b. Internal fifo memory address size

4. Possible Test Scenarios

In this section all possible test scenarios of the FIFO module will be analyzed and discussed.

In general for FIFO testing 2 main test flows are implemented:

Test Name	Description
DataFlowTest	Verifies 1) Data writing, data reading 2) Write, read same and different clocks 3) Resetting during write and read
FullEmptyTest	Verifies: 1) Writes until full, reset from the full state 2) Read until empty, reset in empty state 3) Fast reading, slow writing 4) Fast Writing, slow reading

To support the foregoing mentioned test flows, following small test scenarios were created.

Test Scenario Name	Explanation
WriteDataInFIFO	Checks the data writing in FIFO
ReadDataFromFIFO	Checks the data reading from FIFO
WriteReadWithDifferenClocks	Checks the data writing and reading with different clock for write and read operation
WriteReadWithSameClock	Checks the data writing and reading with same clock for write and read operation
ParallelWriteRead	Checks simultaneous writing and reading in DUT
WriteUntilFull	Writes data until FIFO becomes full
ReadUntilEmpty	Reads the data until FIFO becomes empty
ResetDuringWrite	Resets the FIFO during Write operation
ResetDuringRead	Resets the FIFO during Read operation

4.1 Resetting FIFO

Usually the TB needs to reset the DUT at the beginning of the test – to bring DUT into initial state. However to have thorough test, we will have an option in our TB to reset the DUT anytime we want.

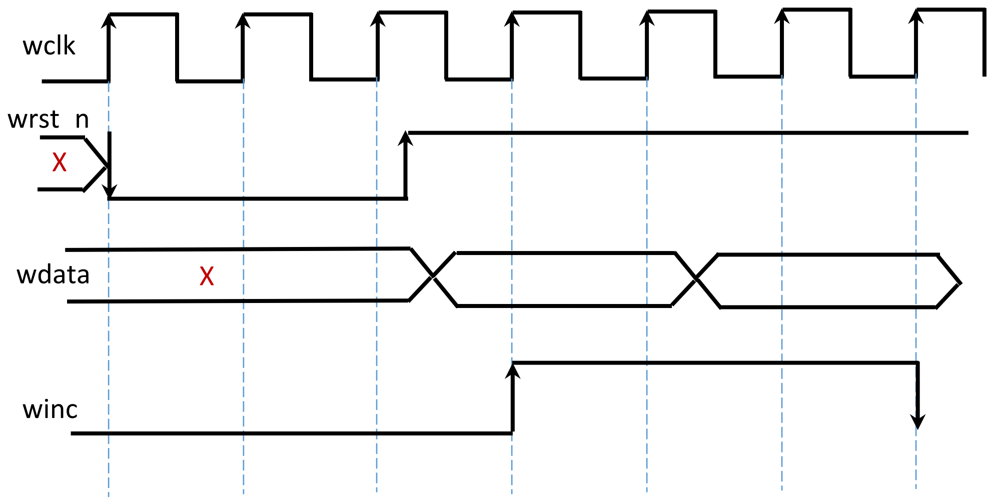
There reset can be done either WRITE pointer or READ pointer, or both of them.

4.2 Write Data into FIFO

This is the most basic operation that comes to the mind when trying to use the FIFO. FIFO is a memory which can remember the order of the data written into it. Therefore writing the data inside FIFO is one of its basic operations.

The TB should generate random input data and write that data into the FIFO.

The waveform diagram of the FIFO inputs for writing data into it is following:

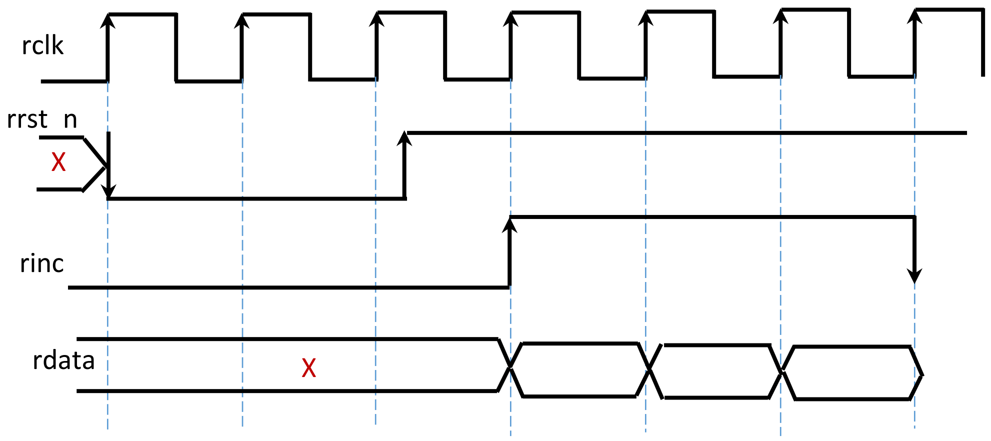


The data writing can be done either for single bit or for multiple bits i.e. multiple times write a data. Tb should support both, also in TB there must be e mechanism to randomize the number of times that data should be written in DUT.

Note: When writing into the FIFO the `rinc` input should be tied to constant value – 1’b0. Otherwise if we leave it in X state, than the value of write pointer would be also X and whole simulation will fail.

4.3 Read Data from FIFO

The next basic operation with FIFO, is reading the data from it. The waveform diagram for reading data from FIFO is following:



4.4 Write Data and read data with different clocks

This is the basic operation of the FIFO – writing and reading the data. TB should generate random data, than write that data into the FIFO and read the written data. After which the TB should compare the read data with the written data, and make sure that they match.

Also TB should check that the order of writing data and reading data is the same.

TB should also generate the clocks of write and read operations. TB should test different scenarios for write and read clocks. The best way would be having some random ratio variable, randomizing which will change the ratio between the write and read clocks.

The Writing and Reading of the data can be either for 1 data or for multiple data. So in TB option should be to control the number of times that TB writes/reads data from DUT.

Also this test scenario can be extended, TB can write/read using one clock than on fly, without DUT resetting change the clocks ratio and continue writing and reading in DUT.

4.5 Write Data and read data with the same clocks

This is subset of 2.2 point, here write and read operations clocks are the same, in other words FIFO is working in synchronous mode.

4.6 Parallel write and read data

In this scenario the TB performs write and read into the DUT in parallel. One time using the same clock for both read and write, than changing the clocks.

4.7 Write until Full

In this scenario the TB writes data into FIFO until it becomes full. The main target of this scenario is testing the wfull signal generation in FIFO.

4.8 Read until Empty

Here the TB reads the data from FIFO until it becomes empty. In this scenario we the FIFO `rempty` signal.

4.9 Reset during write phase

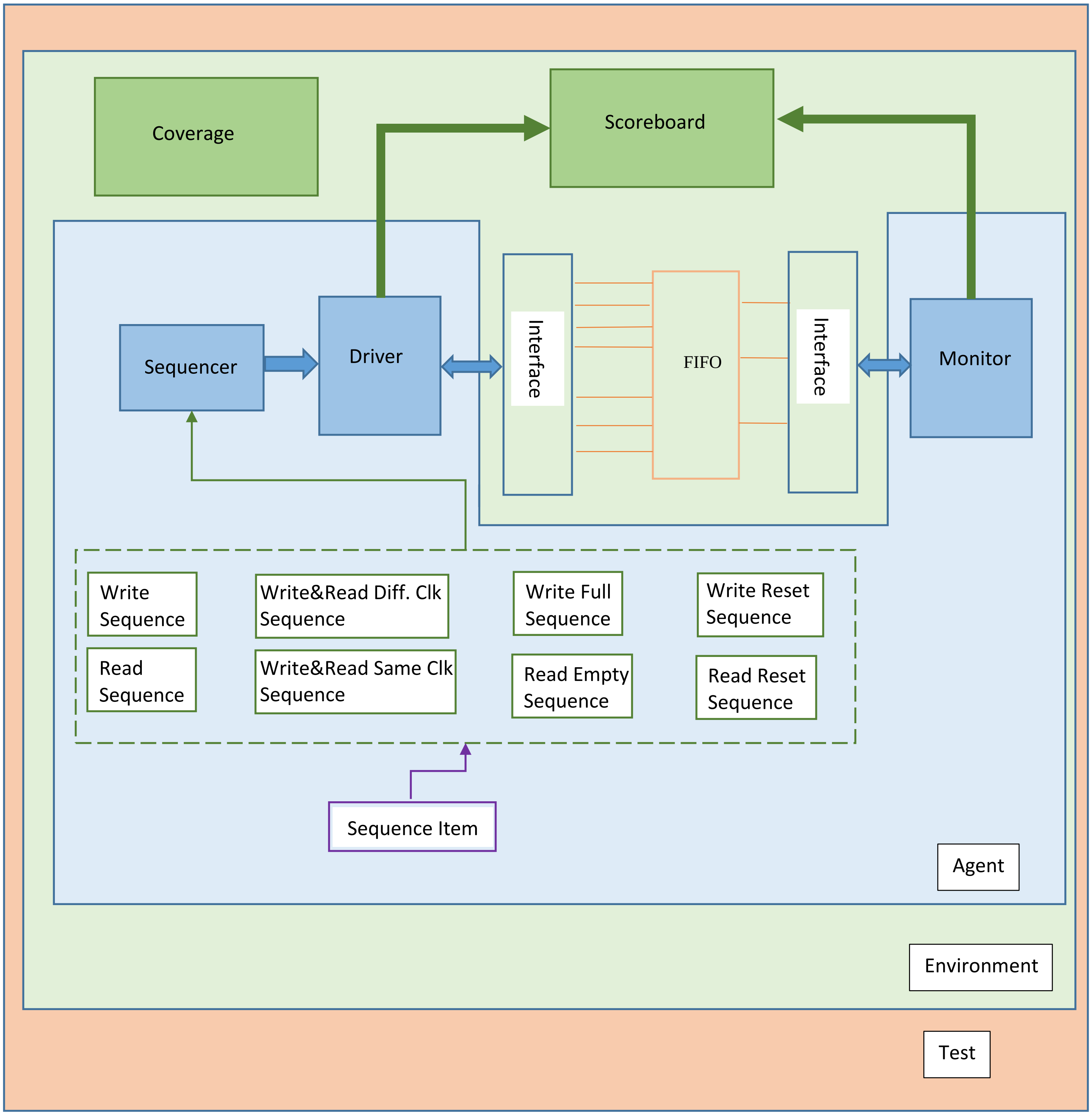
This scenario is prepared for testing the `wrstn` input, during write phase TB will randomly active the reset signal to see whether the FIFO is being reset or not.

4.10 Reset during read phase

This scenario is prepared for testing the `rrstn` input, during read phase TB will randomly active the reset signal to see whether the FIFO is being reset or not.

5. UVM TB Structure

Below is the TB structure

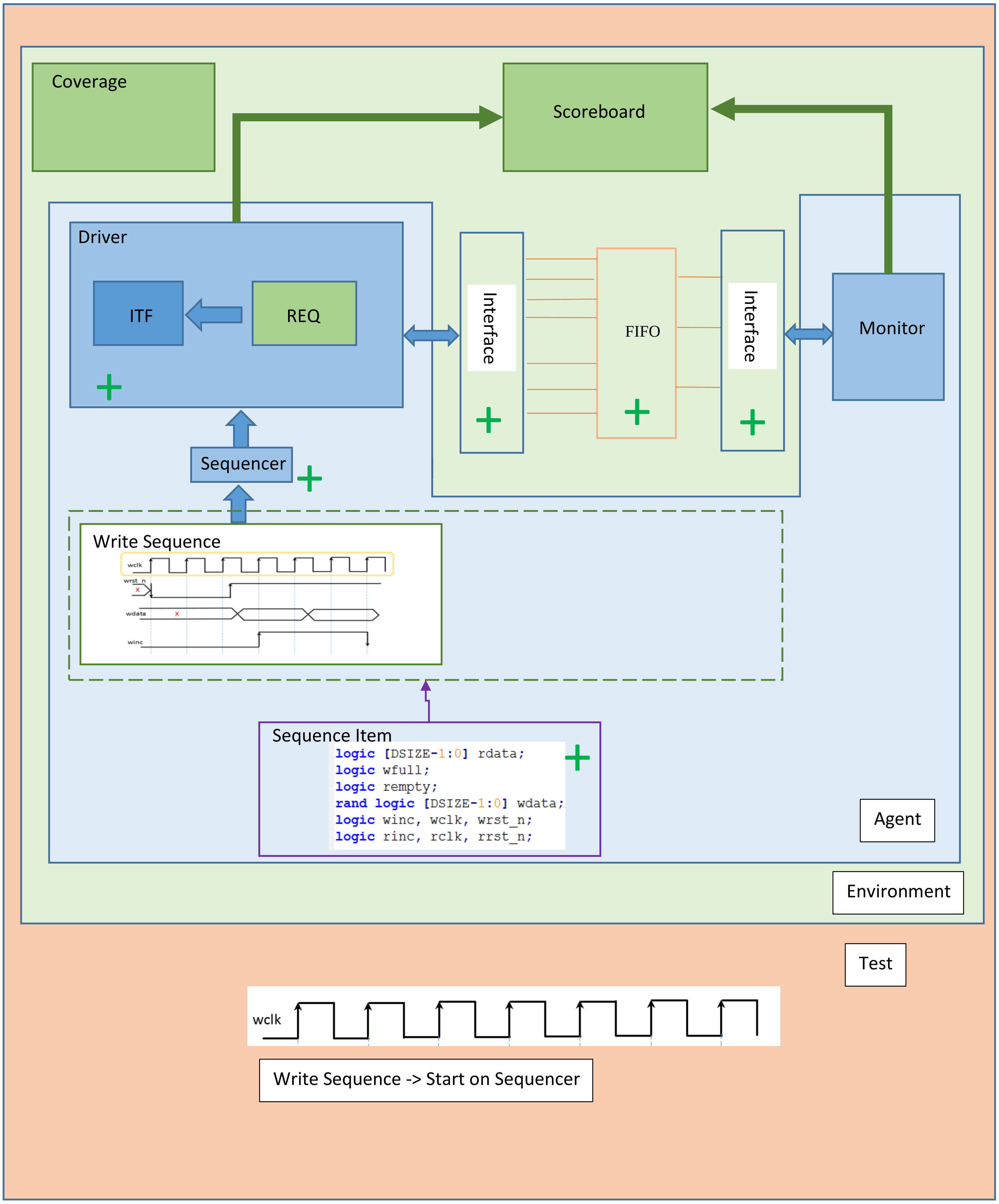

Coverage
Scoreboard
Sequencer
Driver
Interface
FIFO
Interface
Monitor
Write Sequence
Write&Read Diff. Clk Sequence
Write Full Sequence
Write Reset Sequence
Read Sequence
Write&Read Same Clk Sequence
Read Empty Sequence
Read Reset Sequence
Sequence Item
Agent
Environment
Test

6. Test Implementation Details

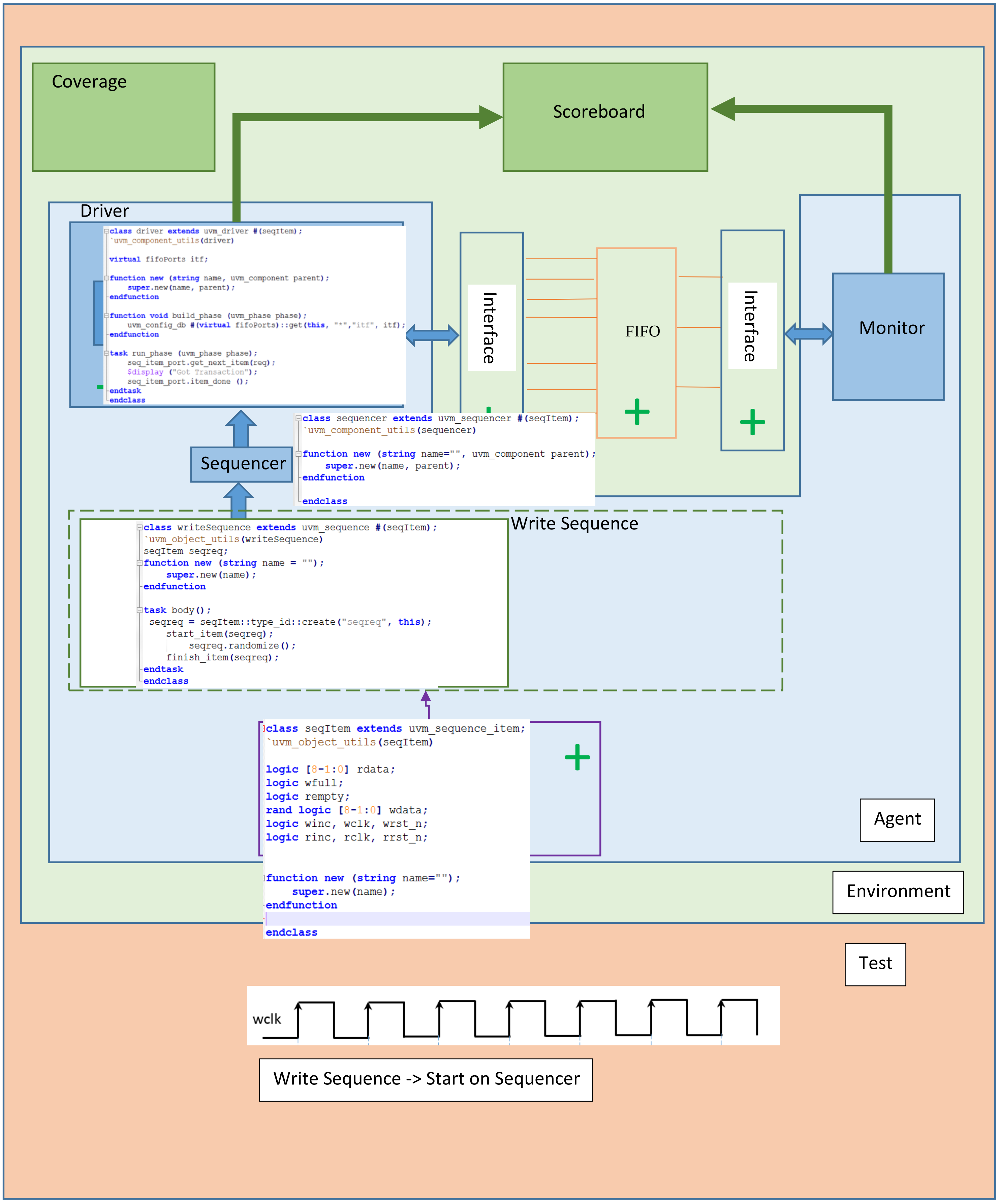
6.1 Write Data into FIFO

Here in detail will be discusses the write data into FIFO test steps, data generation and so on.

YanSolutions



```
logic [DSIZE-1:0] rdata;
logic wfull;
logic rempty;
rand logic [DSIZE-1:0] wdata;
logic winc, wclk, wrst_n;
logic rinc, rclk, rrst_n;
```



```
class driver extends uvm_driver #(seqItem);
`uvm_component_utils(driver)

virtual fifoPorts itf;

function new (string name, uvm_component parent);
    super.new(name, parent);
endfunction

function void build_phase (uvm_phase phase);
    uvm_config_db #(virtual fifoPorts)::get(this, "*","itf", itf);
endfunction

task run_phase (uvm_phase phase);
    seq_item_port.get_next_item(req);
    $display ("Got Transaction");
    seq_item_port.item_done ();
endtask
endclass
```

```
class sequencer extends uvm_sequencer #(seqItem);
`uvm_component_utils(sequencer)

function new (string name="", uvm_component parent);
    super.new(name, parent);
endfunction

endclass
```

```
class writeSequence extends uvm_sequence #(seqItem);
`uvm_object_utils(writeSequence)
seqItem seqreq;
function new (string name = "");
    super.new(name);
endfunction

task body();
  seqreq = seqItem::type_id::create("seqreq", this);
    start_item(seqreq);
        seqreq.randomize();
    finish_item(seqreq);
endtask
endclass
```

```
class seqItem extends uvm_sequence_item;
`uvm_object_utils(seqItem)

logic [8-1:0] rdata;
logic wfull;
logic rempty;
rand logic [8-1:0] wdata;
logic winc, wclk, wrst_n;
logic rinc, rclk, rrst_n;


function new (string name="");
    super.new(name);
endfunction

endclass
```

7. Clock Generation in TB

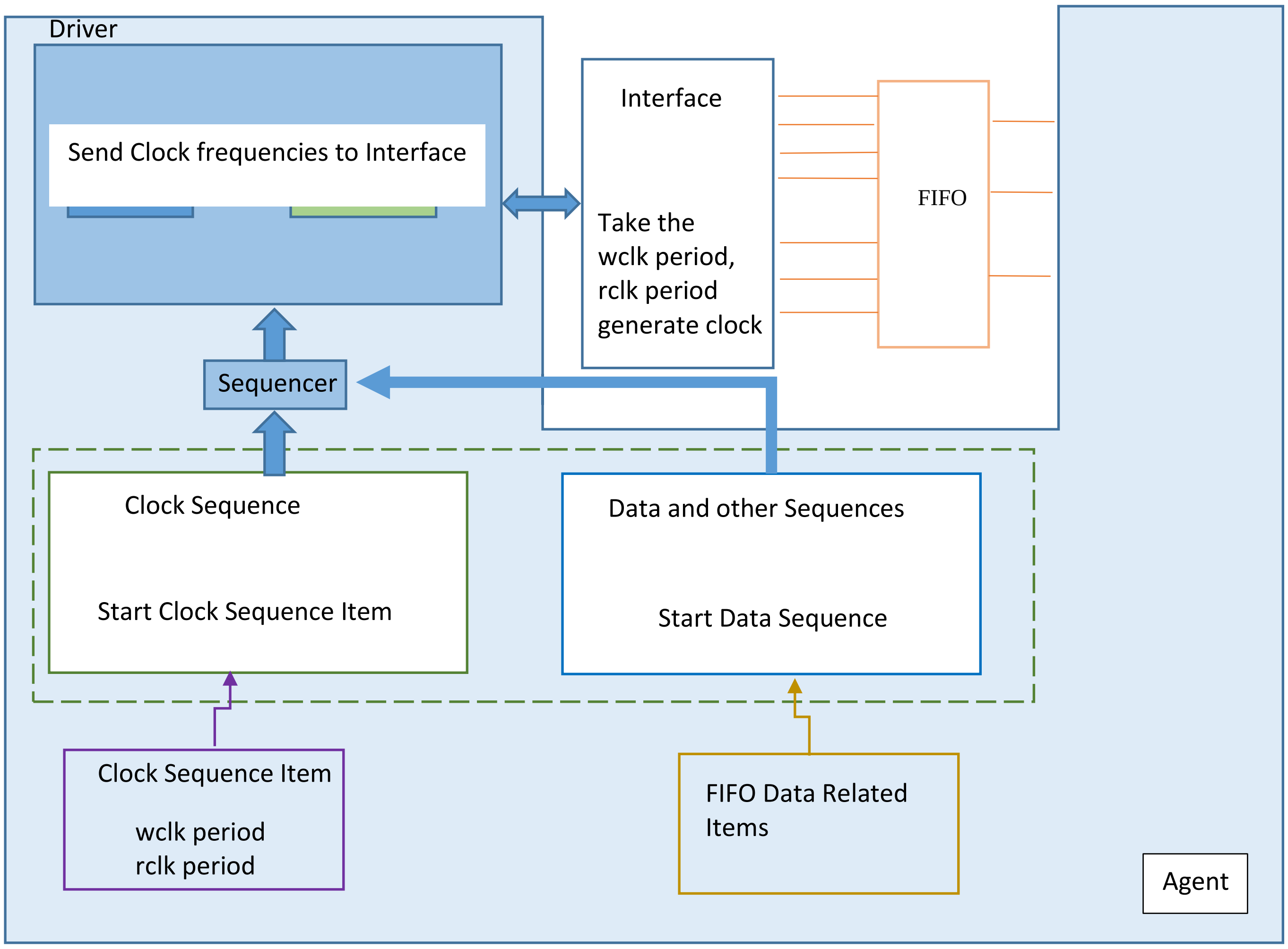
In our TB we have 2 independent clocks write clock (wclk) and read clock (rclk). For some test scenarios they can be the same, however in general TB should drive these clocks independent. Also there should be a mechanism to randomize these clock frequencies and ratio between them.

These are the requirements that exists on the clocks of the TB

1) TB should be able to modify the periods of wclk and rclk
    a. TB should be able to change the ratio between wclk and rclk periods
2) wclk should be separate from rclk, no any relation should be between them. The periods of wclk/rclk can be controlled by random variable.

The clocks generation scenario for FIFOs if following:

a) Both wclk and rclk will be generated in the interface
b) There would be 1 sequence which will contain the clock periods of rclk/wclk. That sequence will be started at the begging of the simulation.
c) For testing other functionalities of FIFO, other sequences will be created.



8. Reset mechanism in TB

For DUT reset testing, following scenarios should be executed by TB

1) Initial reset – write and read reset, before starting the test → This can be done in driver reset_phase
2) Write reset with data in FIFO
3) Read reset with data in FIFO
4) Write reset during read operation
5) Read reset during write operation
6) Random Reset – write or read, anytime

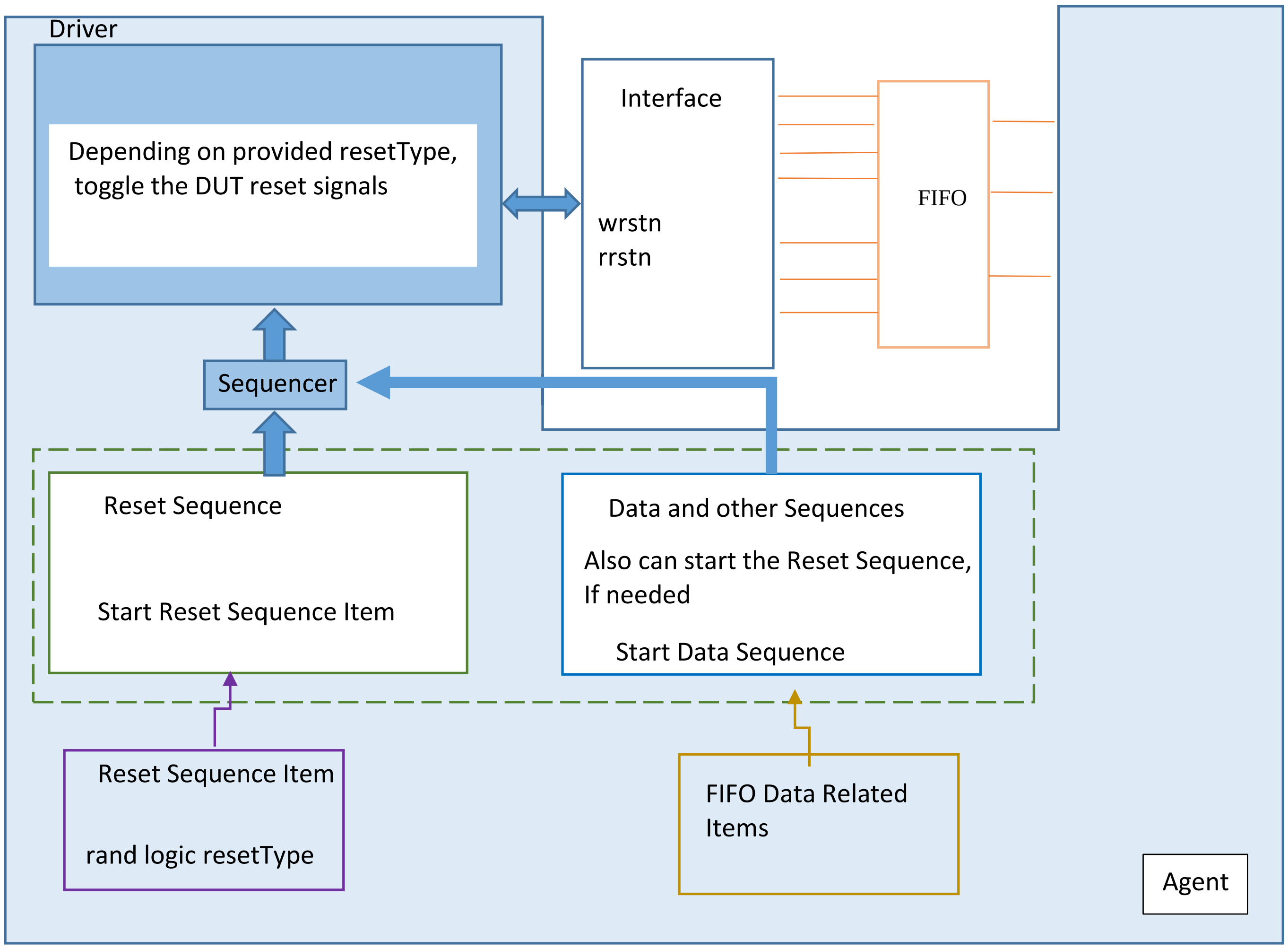
1) Initial Reset

This reset is automatically done in the Driver `reset_phase`.

2) Write Reset with data in FIFO

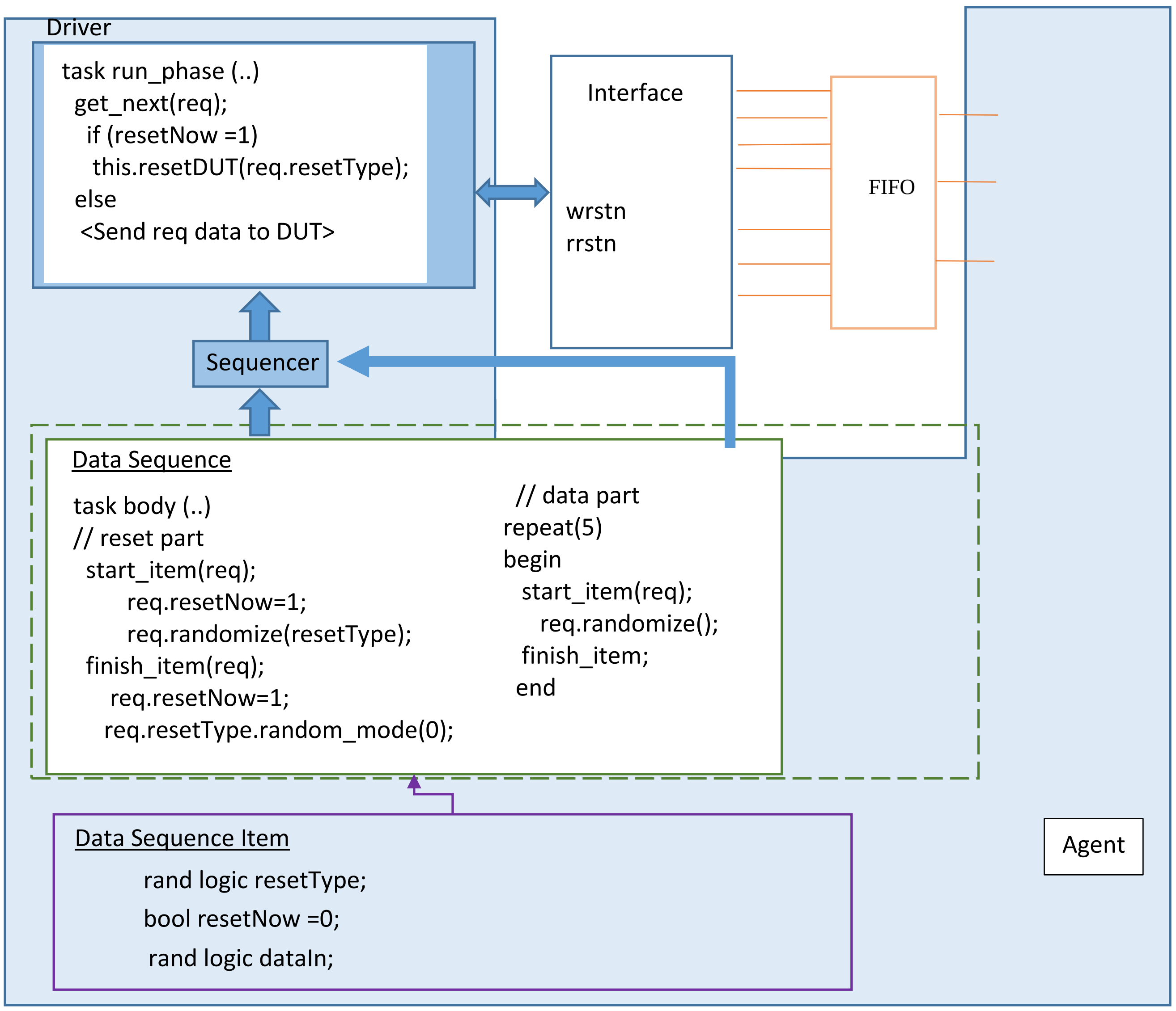
There can be 2 scenarios for Reset implementation.

Case1: Separate Sequence Item for Reset



<u>Case2:</u> Reset control part is in the same sequence Item where the FIFO data is

Data Sequence Item



Driver:

```
task run_phase (..)
  get_next(req);
    if (resetNow =1)
     this.resetDUT(req.resetType);
  else
   <Send req data to DUT>
```

Data Sequence

```
task body (..)
// reset part
  start_item(req);
        req.resetNow=1;
        req.randomize(resetType);
  finish_item(req);
      req.resetNow=1;
     req.resetType.random_mode(0);
```

```
 // data part
repeat(5)
begin
   start_item(req);
      req.randomize();
   finish_item;
 end
```

Data Sequence Item

```
rand logic resetType;
bool resetNow =0;
 rand logic dataIn;
```

9. List of Sequences in TB

The whole TB consists of sequences, which run in parallel or sequentially to test the DUT. Here the full list of sequences is presented

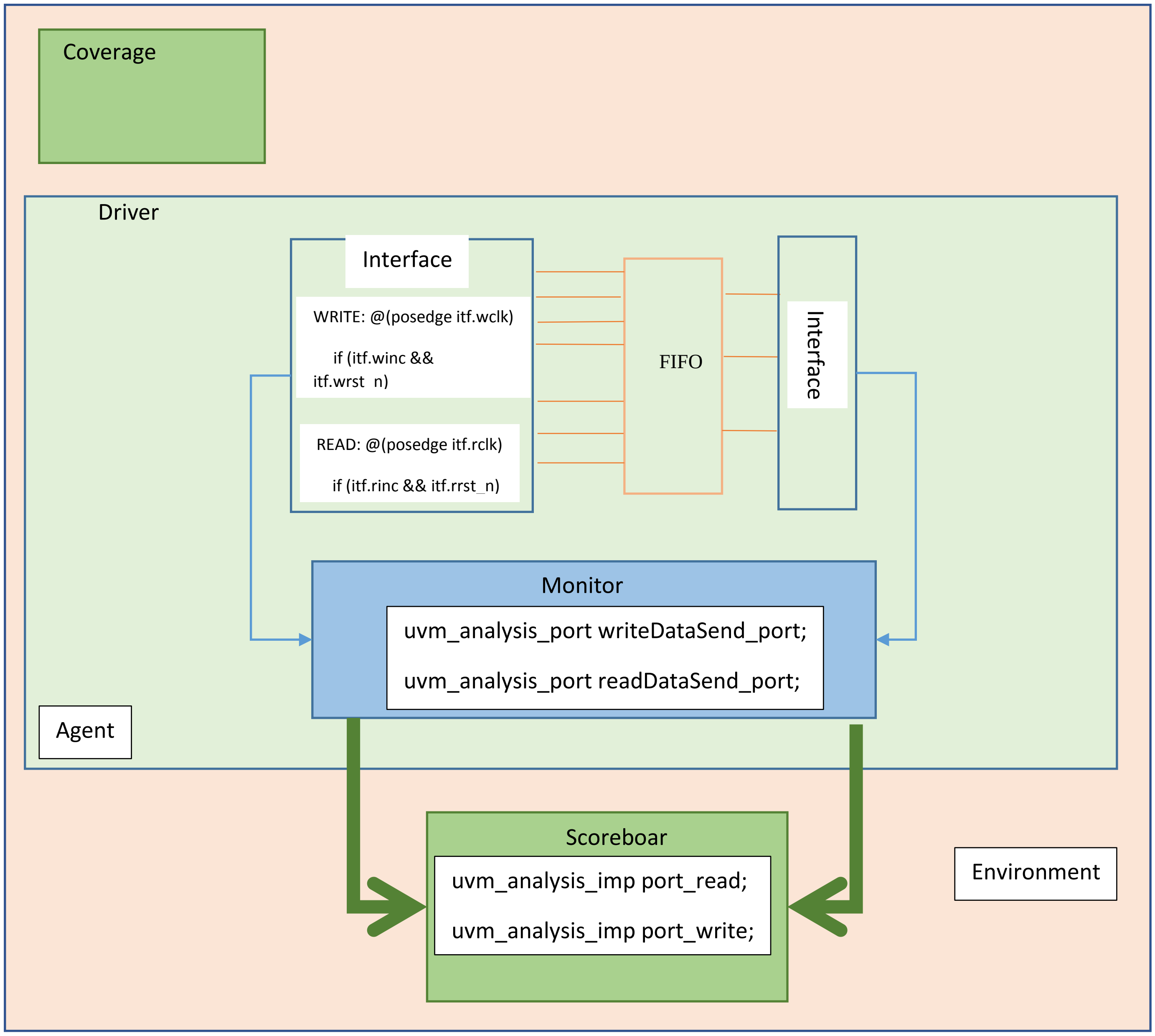
Sequence Name	Function
FullResetSeq	Performs write and read reset
WriteReset	Performs write reset
ReadReset	Performs read reset
WriteData	Writes Random data in DUT
ReadData	Reads data from DUT
WriteReadData	Simultaneously writes and reads data from DUT

10. Monitor and Scoreboard Class

There monitor class implemented for FIFO, one is called `monitor`, which monitors the FIFO input and output data.

Main task of input monitor is to understand wheher read or write operartion is being performed on the FIFO, after which create a package and send to the scorebaord class. The monitor starts collecting the FIFO input data when either `winc or winc` FIFO input goes to high. After that foreach clock posedge it takes FIFO corresponding input and output, makes them as a transaction and send through its analysis port.

There is also Scorebaord in the FIFO VIP. The main role of that class is comparing whether the read data equal to previously wrote data or not. Also scoreboard can understand if FIFO is empty ot full. Based on that it will check if FIFO wfull or rempty signals are behaving correctly or not. The monitors structure is shown in the below picture.


Coverage
Driver
Interface
WRITE: @(posedge itf.wclk)
if (itf.winc &&
itf.wrst_n)
READ: @(posedge itf.rclk)
if (itf.rinc && itf.rrst_n)
FIFO
Interface
Monitor
uvm_analysis_port writeDataSend_port;
uvm_analysis_port readDataSend_port;
Agent
Scoreboar
uvm_analysis_imp port_read;
uvm_analysis_imp port_write;
Environment

In FIFO there are 2 main operations write and read, and these operations are independent. So TB doesn't know how many times WRITE will be done in FIFO and also don't know how many times READ will be done in FIFO. The scoreboards should start comparing the data when

a) WRITE operation has be done (at least once)

b) After that READ operation has been done

Until there is no READ operation done the WRITE data should be collected in the `tlm_fifo` and scoreboard should wait.

1. To test the FIFO full state:

The scoreaboard should have write counter, which counts the number of write operations. And when the number of WRITE operations equals to the size of FIFO, SB should check the “`wfull`” signal value and make sure that it is high.

In case when Read operation is done, than the write counter value is being decreased by 1.

During FIFO

2. To test the FIFO empty state:

Scoreabord should have read counter, which counts the number of read operations. When the number of read operations is higher than the FIFO size, than SB should check the value of `rempty`. Check that signal is high.

In case when WRITE operation is done, than the read counter value should be decreased by 1.

These 2 check should be active during the whole test, as in real DUT the full/empty signals are generating anytime when DUT becomes full/empty.

11. Coverage

The functional coverage was developed to check the basic functionality of the FIFO design. The coverage checks the following functions to be tested in the FIFO during verification

1. FIFO Write enable port toggling
2. FIFO Read enable port toggling
3. Full signal transition from passive state to active state
4. Empty signal transition from passive state to active state

At this stage there are several separate FIFO tests, which are not combined into one test. As a result of this there is no way to measure the combined coverage of all tests. And for each test we have low coverage, but if we run all tests, than the coverage will be high.

The coverage commands are implemented in a separate package, after which that package is included in the top module of the TB. In that top module the coverage objects were created.

12. Assertion Checks

The basic principle of assertions development that is used here – is that assertions should not check the same thing that test-bench is checking. The assertions should try to check the items different than the TB is checking.

Also FIFO VIP has some assertions developed for the TB. To control the TB flow

FIFO Assertions:

1. Check that on reset all pointers and full/empty signals are going to their low states
2. Check that fifo_empty is asserted when the fifo read counter is 0
3. Check the fifo_full is asserted when the fifo write counter is bigger than FIFO width is
4. Check that if fifo is full, and you try to write than write pointer does not change
5. Check that if fifo is empty, and you try to read than read pointer does not change
6. There is a property that warns when you try to write in a full fifo
7. There is a property that warns if you read from the empty fifo
8. Make sure that in WRITE mode the `rinc` input has defined state – 0 or 1. It should not be X. Same for READ mode
9. Check setup hold times of the clocks
10. Check data writing to be done in memory
11. Check data reading

TB Assertions:

1. TB stars testing the FIFO, only after resetting it
2. TB should change the seed value of randomization, otherwise the warning message should be printed
3. The message should be printed showing the read/write clock frequencies

Note: The best way to add assertions to TB using SystemVerilog checker construct. However As the simulator that were used for this VIP creation, did not support checker, hence the SVAs were created in the module.

Please note that concurrent assertions cannot be created in the class, the main reason of this is that class is dynamic object, but concurrent assertions are static object i.e. they exists during whole simulation time.

5. Future Work

These are the items which will be updated in Future.

6. At this stage transaction data and address width are fixed, however in future these will become parametric. And user can control the FIFO size in TB using parameter.
7. The FIFO read and write test will contain randomization for the number of writes and number of reads. So user will not bother of how many times to write or read, this all will be randomly controller by the TB.

Create new coverage for following states of the FIFO

8. Same clock speed

9. Different clock speed
10. Run FIFO verification using synthesized netlist and SDF timing anotation

Make a top level test, which will run all sub-tests and measure FIFO coverage. In this case we will know the exact coverage value of all our tests.