UVM VERIFICATION PLAN

For

Asynchronous FIFO

Fundamentals of Pre-Silicon Validation  
Winter -2024

By

**Abram Fouts & Yunus Syed**

**ECE-593: Fundamentals of Pre-Silicon Validation – Venkatesh Patil**

**February 24, 2024**

Table of Contents

[**Overview**](#_lal29e6vjrcb) **3**

[**The UVM Testbench Architecture**](#_urf667599p24) **3**

[**UVM Testbench**](#_vwjzvsjjdee) **4**

[**UVM Test**](#_b1yf8araxqj) **5**

[**UVM Environment**](#_rs6crli215w9) **5**

[**UVM Scoreboard**](#_d72d3u58excc) **5**

[**UVM Agent**](#_4454q5nbg2fx) **5**

[**UVM Sequence Item**](#_uhfhwb5gn8ti) **6**

[**UVM Sequence**](#_rbkek096e85b) **6**

[**UVM Sequencer**](#_2znts1xg188h) **7**

[**UVM Driver**](#_4o7opuex5zc5) **7**

[**UVM Monitor**](#_bq8iigxo9w1r) **8**

[**Coverage**](#_4lhhodhyoq9h) **9**

# Overview

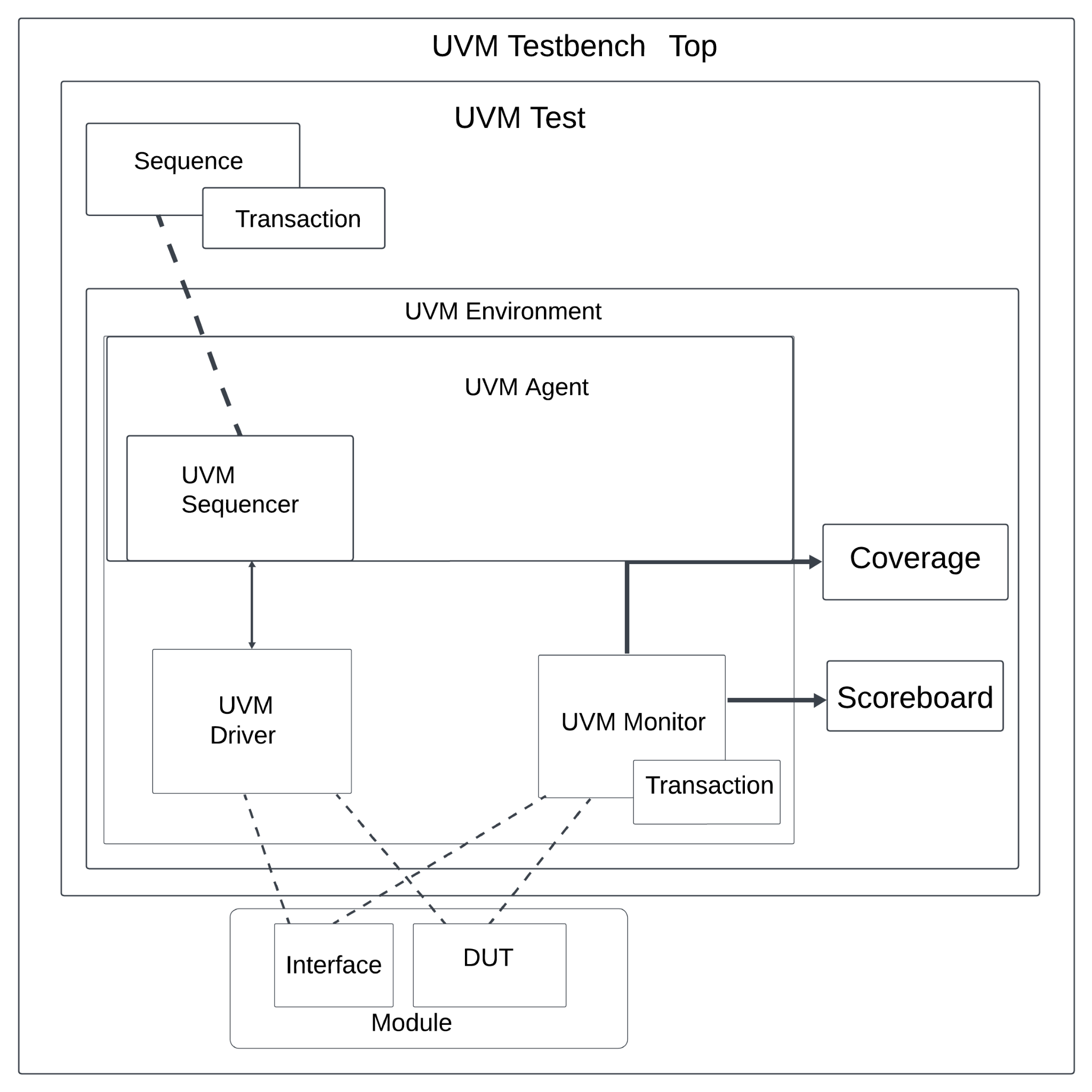
This documentation gives quick overview of explanation of UVM testbench architecture and how we implemented the architecture.

The UVM code for the asynchronous fifo can be found here:

<https://github.com/abfouts/ece_593_asynchronous_fifo/tree/main/verify/uvm>

# The UVM Testbench Architecture

The generic utilities provided by the UVM class library are transaction library mode(TLM), configuration database, component hierarchy, etc.., The Below block diagram shows the terminology and typical block diagram of UVM testbench architecture.



**Fig 1: UVM Testbench Architechture.**

# UVM Testbench

The UVM testbench instantiates the DUT (Design Under Test) module and test class, configuring the connection between them. UVM test is dynamically instantiated at runtime, allowing testbench to be compiled and run with different tests.

The testbench or *top* modules will generate the producer and consumer clocks, instantiate the dut and create the random test that extends from uvm test and register it with the factory. The top-level module creates the interface and places it on the uvm\_config\_db for all uvm components to receive.

# UVM Test

The UVM Test is the top-level component in the UVM testbench, it performs the main functions like instantiating the top environment, configuring the environment using factory overrides, and applying stimulus by UVM sequences through environment DUT.

The UVM test being used is specific for a random value test that will be run many times. The test will send many random burst transactions back to back with random resets produced throughout.

# UVM Environment

It is a hierarchical component that groups other verification components that are connected. Components that are instantiated inside the UVM environment are UVM Agents, UVM Scoreboard, or other environments. The top-level environment target remains similar to DUT.

The UVM environment extends from the built in UVM ENV, and it instantiates the agent and the scoreboard. In the future (next milestone), the environment will also instantiate the coverage. In the connect phase the environment will connect the agent analysis port to the scoreboard analysis port.

# UVM Scoreboard

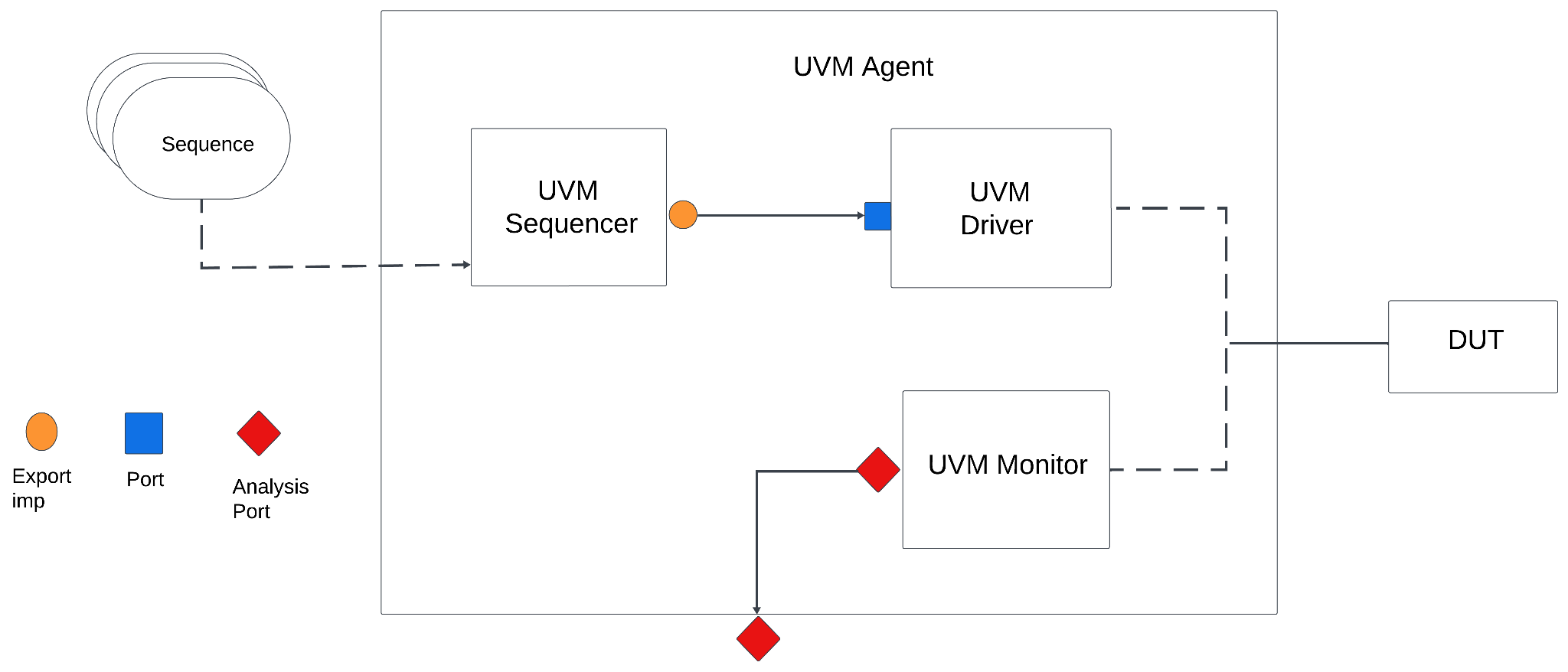
The Scoreboard's main function is to check the behavior of DUT, in our condition we are writing the mailboxes for the scoreboard, the data is generated where the mailboxes are used for communication between parts of the UVM testbench, and the random constraint makes sure it is in the specified range of burst entries.

The scoreboard will extend from the UVM scoreboard, and it uses the macro “uvm\_analysis\_imp\_decl” to create a custom port name for the write function. The comparison will be performed inside of the write function as it is not streaming individual transactions, but instead receiving the entire burst packet. It can be designed in other ways, but this allows the scoreboard to know exactly how many items were written and read from the asynchronous fifo.

# UVM Agent

UVM Agent is a hierarchical component that contains other verification components which are dealing with specific DUT interfaces. A UVM agent contains a UVM sequencer to manage stimulus flow, a UVM Driver to apply stimulus on the DUT interface, and a UVM monitor to observe the DUT interface. It might include other components like Checkers and TLM models, etc.

There is one agent and it extends from UVM agent and it instantiates the driver, monitor, sequencer, and config. The config sets if the UVM agent is active or not, but is not necessary. For the case of this class the agent will be active, and then creates an analysis port, and gets the ccd vif from the uvm\_config\_db. In the connect phase the monitor analysis port is connected to the agent analysis port to simplify the connection to the scoreboard, and the sequencer is connected to the driver seq\_item\_port.



**Fig 2: UVM Agent**

# UVM Sequence Item

The sequence item is used to define the type of item transaction that will take place in the sequence, sequencer and most of the times the analysis ports. It acts as a packet of data that can contain the inputs and outputs of the DUT.

The sequence item extends from the UVM sequence item and contains the input data, output data, reset and all random variables that are going to be used for randomizing the data. A single sequence item will act as an entire randomly generated write vector or read vector for the monitor. This contains the number of idle write and read cycles and any other information a sequence may need to successfully write to the asynchronous fifo.

# UVM Sequence

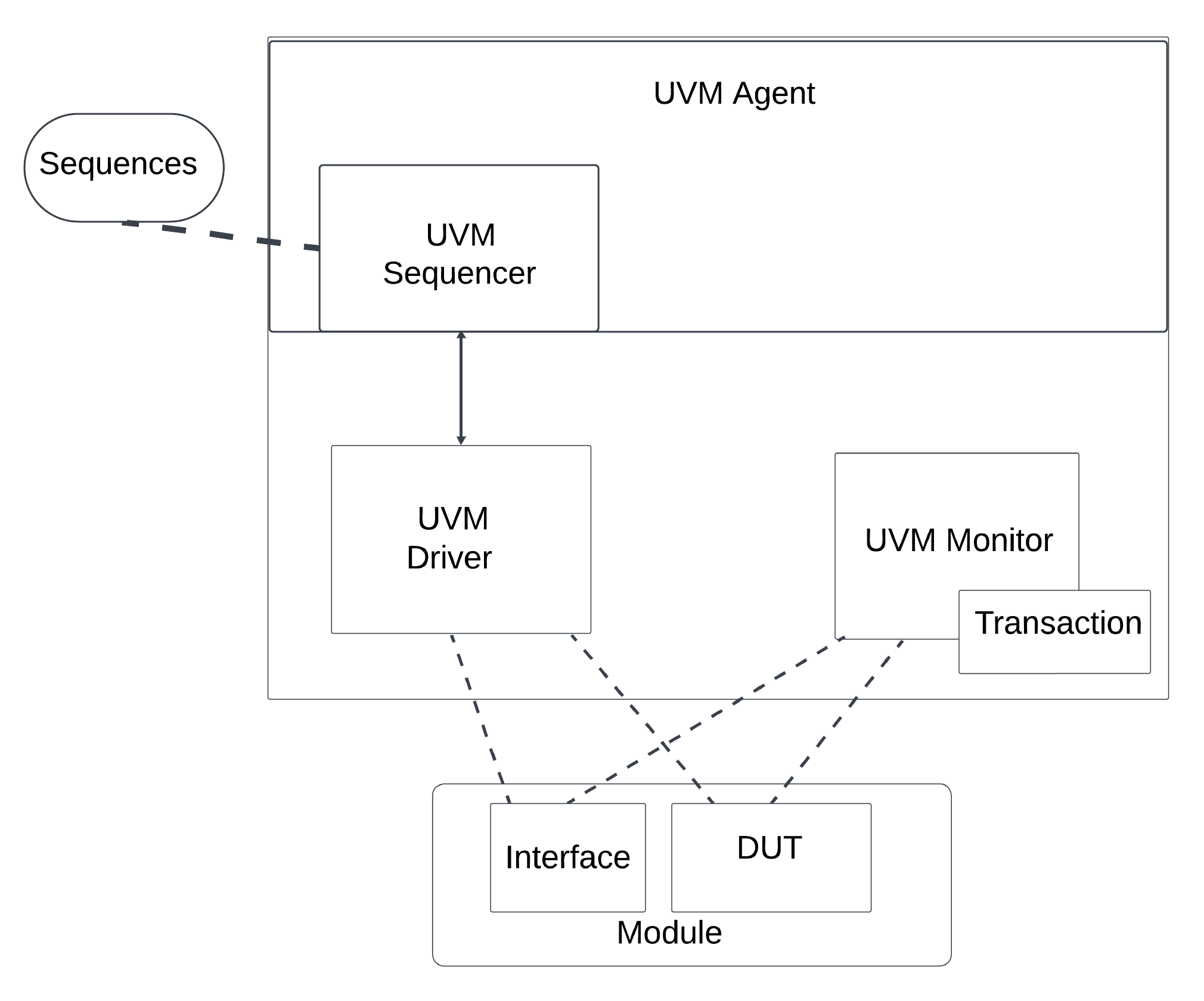
It is an object that contains the behavior of generating stimulus and is not a part of the component hierarchy. It can come from a single transaction and may drive stimulus for the time of simulation. It can invoke another sequence.

There are two sequences. The base sequence for reset that extends from UVM sequence and expects the sequence item defined above, and a random sequence that extends from base sequence. The base sequence randomizes the data but asserts the reset, so nothing should be inputted or outputted to/from the DUT except for the memory being reset. The random sequence contains the method to generate a random number of data writes, and dynamically allocate random bytes to the vectors.

# UVM Sequencer

The UVM Sequencer serves as an arbiter controlling transaction from multiple stimulus sequences, in more detail it controls the flow of sequence items transactions generated by one or more UVM sequences. In our code sequencer generates sequences of random transactions for testing DUT.

The sequencer extends from the UVM sequence.



**Fig 3: Sequencer**

# UVM Driver

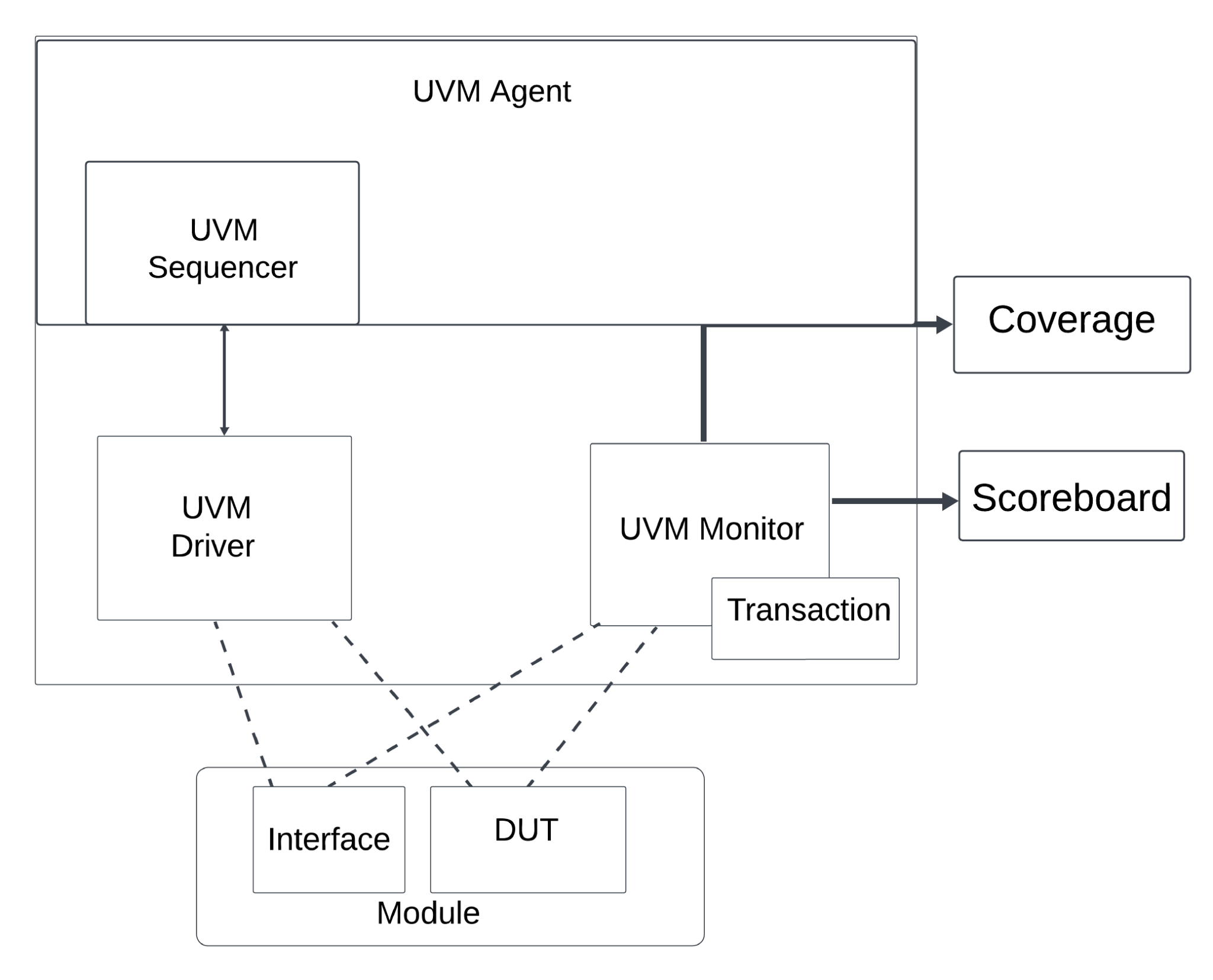
It receives individual UVM sequence items transactions from the UVM sequencer from the sequencer and drives to DUT. Driver converts transaction level stimulus to pin level stimulus. It can also have a TLM port to receive transactions as a burst from a sequencer to access DUT to drive the signals. In our code “start “ can be used from a test sequence to initiate transactions where it specifies the number of write operations and data to write. During simulation tasks will perform write and read operations on DUT.

The driver extends from the UVM driver class and contains the method to write the sequence item data to the DUT. The sequence item contains the entire byte vectors used. This method controls both read enable and write enable signals. The read and writes are forked off to run in parallel. Both loop for the expected number of writes, and on the nth +1 cycle it strobes write enable and read enable signals, and the write provides the expected byte onto the data in bus. The respective sections wait for the FIFO full or FIFO empty signals.

# UVM Monitor

UVM Monitor samples the DUT interface and captures the information there in transactions which are sent out to the rest of the UVm testbench for further analysis. It is similar to Driver as it converts the pin-level activity to transactions.

The monitor extends from the UVM monitor and provides the monitored sequence item to the analysis port of the scoreboard for comparison. The monitor keeps track of when a transaction occurs, so when it receives a write when it has not received any transactions previously and it is not in reset mode starts the transactions. It contains 4 forks (branches) one for both read and write control to keep track if the device is done writing or done reading. The other two are used for read and write transactions, when the WR\_EN signal is strobed it places the DATA IN data into an indexed position of a sequence item. At the same time, the RD\_EN signal places DATA OUT into an indexed position of a sequence item. Once both read and write are done receiving / sending data the sequence item is sent to the scoreboard via the analysis port.



**Fig 4: UVM Monitor having Transaction, Coverage, and Scoreboard**

# Coverage

Coverage will be used to function verify the asynchronous fifo. It will instantiate the environment while monitoring the virtual interface.