Reversing Data With Valid-Ready Protocol Verification Documentation

By: Omar Smri, Exalt Technologies

December, 18th, 2019

Table of Contents:

Valid-Ready Data Reverser - DUT

A circuit that implements reversing on the data fed. The data is inputted to the DUT according to valid-ready protocol. The DUT then does the reversing bits operation and sends the reversed data on the output. The reversing done in the DUT is bit reversing. The DUT takes address as well and does no operation on the address but passes it to the output according to valid-ready protocol. Figure 1 shows a black-box block diagram for the DUT.

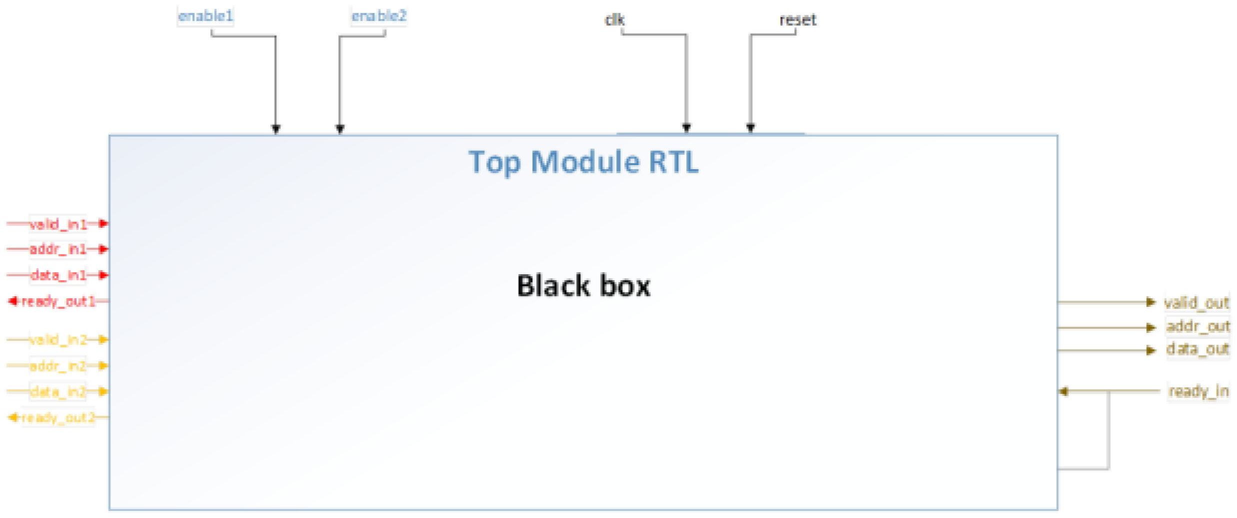


Figure 1. Black box block of DUT.

As shown in Figure1, the DUT has three groups of signals to provide communication with and through DUT. These groups of the signals are shown on both right and left sides of the DUT. Two groups on the left side represent the data signals that are connected to the master which sends data to the slave which communicates through the group of signals on the right side of the DUT. The DUT has two groups of signals on its left since it can be connected to two masters. These three groups of signals (**valid_in1, addr_in1,data_in1 and ready_out1)**, (**valid_in2, addr_in2, data_in2 and ready_out2**) and (**valid_out, addr_out, data_out, ready_in**). In which the two masters communicate through the first two groups and the slave communicates through the last.

The DUT also has clock and reset signals besides with two control signals. These two control signals are **enable1** and **enable 2**. enable1 and enable 2 are used to determine which master is sending data to one slave through the DUT. The cases are shown in Table 1.

enable1	enable2	input
0	0	0
0	1	Master 2 (signals` group 2)
1	0	Master 1 (signals` group 1)
1	1	Master 1 (signals` group 1)

Table 1. DUT input source cases.

The clk input of the DUT is connected to a frequency clk generator. The reset input is active-low input where resetting the DUT leaves all the outputs of the DUT reset but the ready_out1 and ready_out2 outputs which turn to any value on reset.

Figure2 shows an internal block diagram of the DUT. In which it contains two internal blocks used to implement the desired functionality.

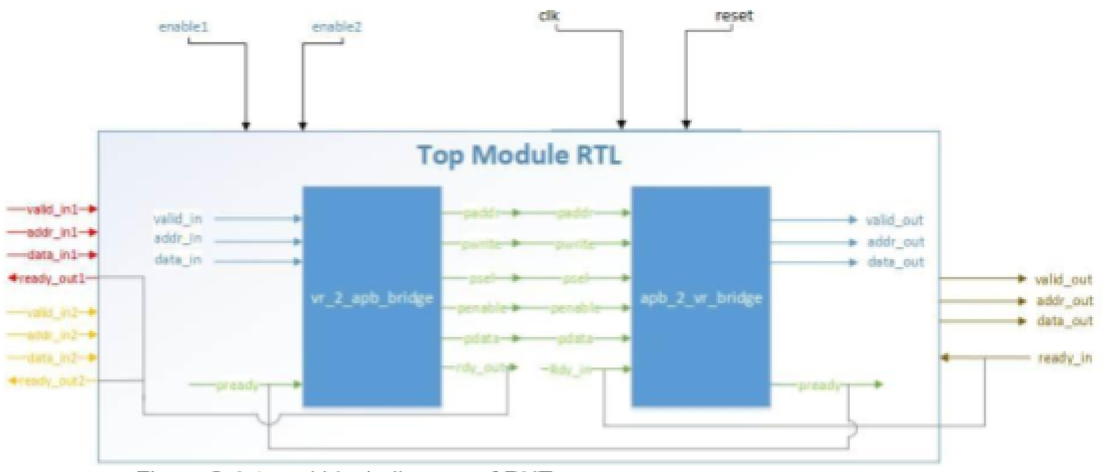
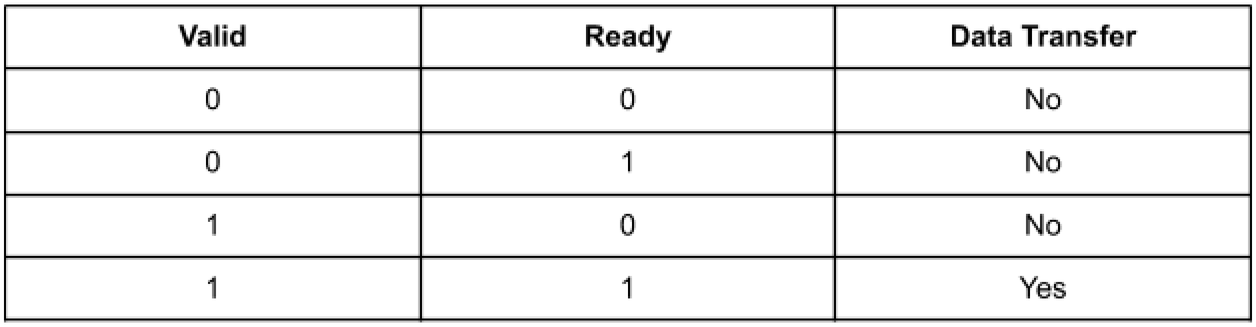


Figure 2. Internal block diagram of DUT

Valid-Ready Protocol

Valid-Ready protocol is a common hardware protocol for sending data between producer and consumer. In which the producer sends data to the consumer. Along with the data, it sends a **valid** signal. And the Consumer sends back a **ready** signal. Once both the valid and ready signals are active high, the data is received by the consumer. Table 2 shows the cases of the signals valid and ready and transferring data based on the values of these signals.



Valid	Ready	Data Transfer
0	0	No
0	1	No
1	0	No
1	1	Yes

Table 2. Truth table for data transferring in Valid-Ready protocol.

There are different approaches regarding the ordering of the valid and ready signals in the valid-ready protocol. This DUT follows the approach that valid shall be asserted before ready. This makes the case in the second row of the table where valid is active-low and ready is active-high not allowed.

Verification Environment Architecture

The DUT is to be verified using UVM methodology. To do so, a full UVM environment is to be built surrounding the DUT. The environment drives stimulus to the DUT, responds to the output of the DUT and checks the results in the scoreboard. Figure 3 shows the block diagram of the UVM environment to be built.

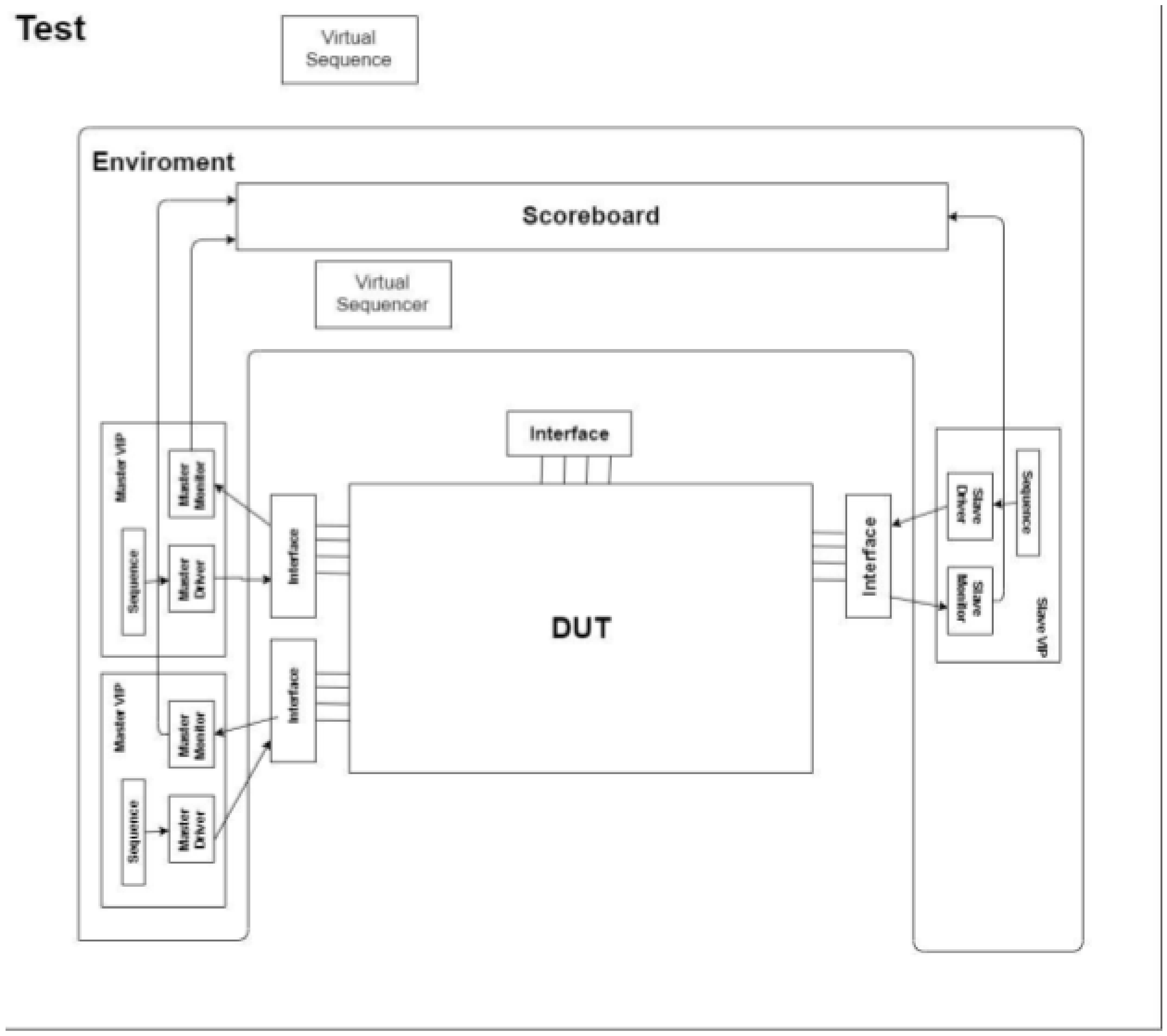


Figure 3. Verification eniroment block diagram

The UVM environment consists of many components that interact and communicate between each other to test the DUT.

Transaction

The transaction is a packet of data where this data is used in the process of testing and verifying the DUT. The data inside the transaction to be written consists of the following fields:

1. data: is array of bits of width defined in parameter in the test. This field is random and to be fed to the DUT.
2. addr: is an array of bits of width defined in parameter in test. This field is random and to be fed to the DUT.
3. randomDelay: is of type integer. This field is randomized and its value specifies the delay between driving two different transactions by specifying the delay between driving each valid signal from the master. In the case of the slave, this random delay is used to specify the delay between receiving the valid and driving the ready.

The transaction sets constraints on the random variables mentioned above. The only constraint here is on the value of random delay to be between 0 and the time of 5 cycles.

Interface

The verification of this DUT requires two types of interfaces, one for the masters and slaves and one for the control signals. Three instances of the first are used for the masters and slaves. One instance of the second is used for the control signals. The two interfaces are as following:

- control_signals_interface: this interface is parameterized by the **clk** and **rst** signals. It also contains a special fields to be connected to **enable1** and **enable2** sideband signals.

- Master_slave_interfac: three virtual instances of this interface are used in the verification environment. One of them connects the slave pins to the slave VIP. Two of them connect each master pins with its master VIP. This interface has fields for the signals of:
  - Valid.
  - Ready.
  - Data.
  - address.

Sequence

The sequence generates the stimulus which consists of a series of transactions that are sent to the driver via the sequencer. This operation is done through the ordinary steps of transferring the sequence elements to the driver through the sequencer. Starting by creating transaction. The sequence then wait for the grant from the sequencer. When the grant is received, the sequence randomizes the created transaction and sends it to the sequencer. The sequence then will wait for item done which will be sent from the sequencer once it passes the transaction to the driver.

In verifying the valid-ready date reverser, different instances of this sequence class will be used for the different masters and slave.

Sequencer

A simple sequencer will be used where all the details of the sequencer are implemented in the uvm_sequencer class. The sequencer extends uvm_sequencer.

Master Driver

The master driver is used to drive signals to the DUT through the groups of signals that are specified for the masters. The Driver drives the signals in the transactions generated in the sequence and passed to the driver by the sequencer.

The driver shall drive the signals according to the valid-ready protocol. This is done through the following steps:

1. Driving the random data and address from the transaction together with an active-high valid signal on a positive edge of the clock.

2. Waiting for the ready signal from the DUT to turn to active-high and then active-low again. This is done by checking the ready signal on the virtual interface instance in the driver.
3. After the condition in step 2 achieved, set the valid signal to active low and the data and address on the virtual interface to zero.
4. Wait a random delay specified in the transaction.
5. Repeat steps from 1 to 4 until finishing all the transactions.

Slave Driver

The slave driver is used to drive signals to the DUT through groups of signals that are specified for the slave. The driver drives the signals in the transactions generated in the sequence and passed to it by the sequencer.

Driving the data to the DUT must be done in a way that satisfies the valid-ready protocol. This is done following the steps:

1. Wait for a valid signal from the DUT to turn to active-high.
2. When the valid signal turns active-high, wait a random delay then drive an active-high ready signal.

Master Monitor

The monitor collects the data returned on the outputs of the DUT and puts them on a transaction-level object preparing to send them to the scoreboard. The monitor in the case of the master in the DUT tested in this test packs the data in an instance of the transaction class. It collects the address and the data received.

Slave Monitor

The monitor collects the data returned on the outputs of the DUT and puts them on a transaction-level object preparing to send them to the scoreboard. The monitor in the case of the slave in the DUT tested in this test packs the data in an instance of the transaction class. It collects the address and the data received.
Important note is that the slave monitor and master monitor are both instances of the same monitor.

Master VIP

The master VIP consists of instances of master driver, master monitor and sequence. These components are explained above. Two instances of this class are used to test this design, each one is connected to an interface of one slave.

Slave VIP
The slave VIP consists of instances of slave driver, slave monitor and sequence. These components are explained above. One instance of this class is used to test the DUT. It is connected to the slave interface.

Scoreboard

The scoreboard receives data from all the monitors in this verification environment. It checks the correctness of the data. The scoreboard basically compares two important fields, one is the address and the other is data. It compares the address received from the monitor of the slave with the address sent to the DUT by the master that currently communicating data to the DUT.

The other field is the data, where the data received from the outputs of the DUT through the monitor of the slave shall be an array that contains a reverse of the data sent to the DUT by the operating master. The data sent to the DUT is passed to the scoreboard by the master monitor.

The scoreboard can determine which master is operating by checking the enable1 and enable2 signals that are set in the test.

Virtual Sequence

The need to virtual sequence appears in this project due to the need of driving different sequences on different interface. Instances of each individual sequence are created inside the virtual sequence. These sequences are to be passed to their drivers through instances of sequencers which are instantiated in the virtual sequencers. The sequences are connected to their sequencers using the start macro. They are all started in the test by starting the virtual sequence using the virtual sequencer.

Virtual Sequencer

The virtual sequencer includes instance of each real sequencer inside all the VIPs of this environment. The virtual sequencer is used to pass the sequence items from the sequences instantiated inside the virtual sequence to their real sequencers in their VIPs.

Stimulus Flow

A typical test flow is described below:

- The selected test generates the sequences of items and connects these sequences inside the virtual sequence to the sequencers inside the virtual sequencers. This starts the master and slave sequences.
- The sequences contain random transactions with random delay to be added between each transaction.
- The transactions generated in the sequences move through the sequencers in the virtual sequencer to their drivers.
- The master drivers now have the data and addresses to be driven to the DUT. the slave driver has the random delay to be set between yielding the valid and driving the ready signals. The drivers have yielded these information from the transactions passed from the sequences by the sequencers.
- The master driver drives a valid signal together with the data and address to the DUT.
- The master monitor collects the driven data from the interface and sends it to the scoreboard.
- The DUT processes the data and signals and passes them with a valid signal to the slave.
- The slave driver yields a valid signal.
- The slave driver starts a random delay. After this random delay, it drives a ready signal. And the slave monitor reads the data and address received from the outputs of the DUT on the interface. Then it sends them to the scoreboard.
- The DUT passes the ready_in signal from the slave driver to the master driver.
- The master driver yields a ready_out signal and resets the valid, data and address signals to zero.
- The scoreboard calculates the correct results from the data it receives from the master monitor and compares it to the information it receives from the slave monitor. It reports the correctness of the data.
- Another sequence starts.
  Figure 4 shows the flow of the transaction in the environment and between the environment and DUT.

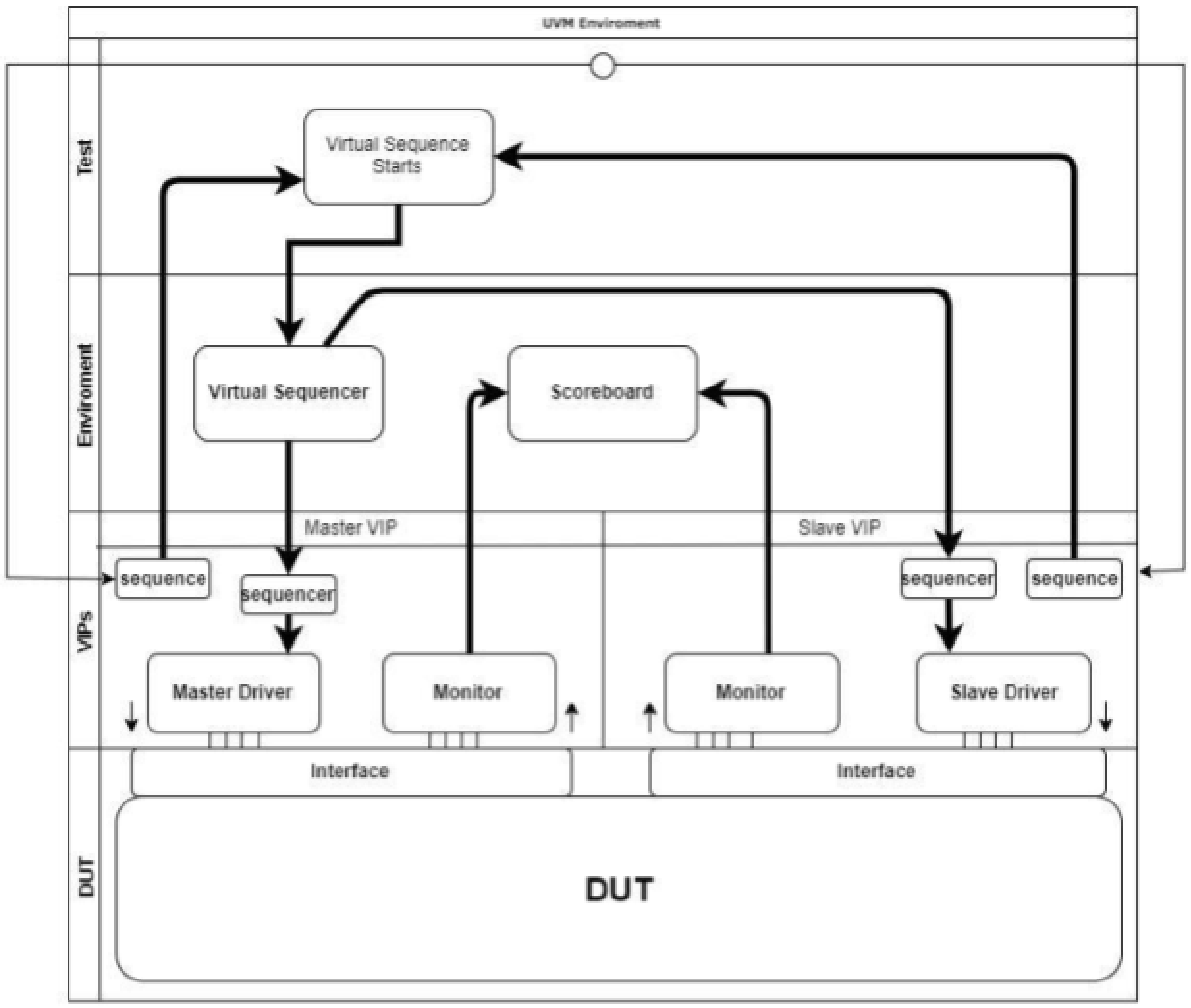


Figure 4. Flow chart diagram of the transaction

Features to be tested

- Reversing the data.
- The validity of the protocol is independent of the delay between transactions.
- The validity of the protocol is independent of the delay between the valid and ready signals.
- Passing the address without any manipulation.
- Reading the data and address from the right master based on the values of the signals enable1 and enable2.

Functional coverage

To be discussed.