UVM testbench to verify Utopia interface for ATM protocol

Divyansh Jain, Pranav Gangwar, Rishav Karki

1) Device we tested in a UVM environment :

In this project, we used Chris Spear's SystemVerilog Testbench for the Utopia DUT as a reference, and modified all the components of the testbench into a UVM testbench. UVM offers a standardized and efficient methodology for verification, enabling higher productivity, improved quality, and increased confidence in the correctness of digital designs.

Summary of key features in the DUT :

Utopia

Utopia stands for Universal Test and Operations PHY Interface for Asynchronous transfer mode (ATM). It has two modes: master and slave mode. In master mode, the controller oversees transfers while external PHY controls transfers in slave mode.

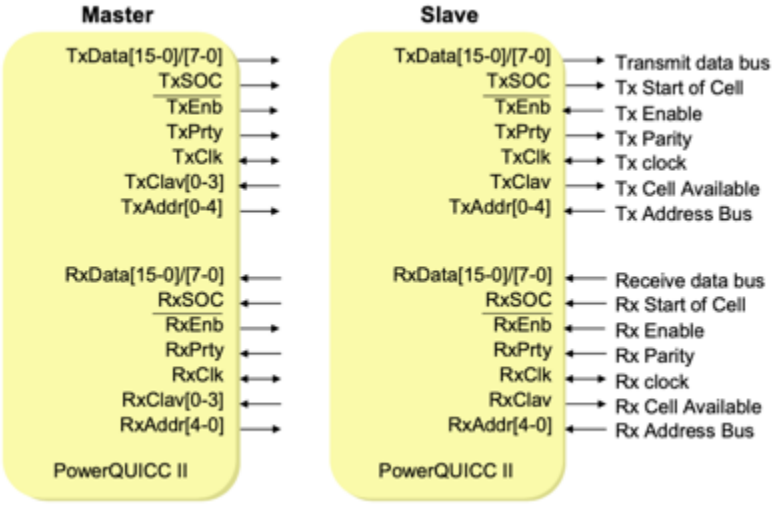


Figure 1: Utopia defining interface between PowerQUICC II and the PHY.

PHY is the name given to the physical device interfacing between the PowerQUICC II and the external system that converts the digital signals of the UTOPIA to those required externally and handles all the line signaling. The receiver data bus is identified

as RXD 0-7 or 15 and are the connections for data transferred from the PHY to the FCC. The Transmitter data bus is identified as TXD 0-7 or 15 and are the connections for data transferred from the FCC to the PHY. When the PHY has a cell ready to transfer to the FCC, it indicates it by asserting the RX cell available signal. The FCC indicates to the PHY that it will sample the receive data and receiver start of cell signals on the next cycle by asserting RX enable.

RXSOC indicates that the receiver data lines contain the first valid byte of the cell. It's a tri- stateable signal for multi-PHY operation. The PHY asserts the transmit cell available signal, TXCLAV to indicate that it has room for a cell to transmit. The FCC indicates to the PHY that valid data is on the transmit data bus with the transmit enable signal, TXENB, and indicates that the first byte of the cell is on the bus with the transmit start of cell, TXSOC.

AAL5 Cell

An AAL5 cell is a basic ATM cell consisting of a 48-octet AAL PDU (Protocol Data Unit) and a 5 octet cell header. Figure 2 shows an example Cell.

Cell Description:

- The AAL5 cell is a standard ATM cell consisting of a header and a 48 byte payload.
- The header consists of the Header Error Control and four other fields supplied by the user in the TCT.
- A "place holder" HEC is supplied by the ATM controller. The actual HEC is calculated and supplied by the PHY.

Header Description:

- GFC - generic flow control
- VPI - Virtual Path Identifier: used with the VCI to form a virtual circuit identifier.
- VCI - Virtual Channel Identifier: used with VPI to form a virtual circuit identifier.
- PT - payload type: identifies cell function. For example, a management cell.
- C - Cell loss priority: if C=1, the cell is subject to being discarded by the network.
- HEC - Header Error Control: an error check CRC on the other 4 bytes of the header; can correct a 1-bit error.

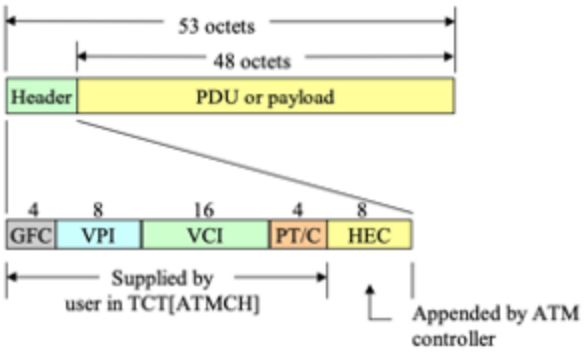


Figure 2: AAL5 Example Cell

AAL5 Frame

An AAL5 frame is one or more cells with an appended trailer in the last cell. The last cell of a frame consists of the fields shown in Figure 3. Padding is required if there is not enough data to fill the last cell. CPCS-UU+CPI is supplied by the user for transmission.

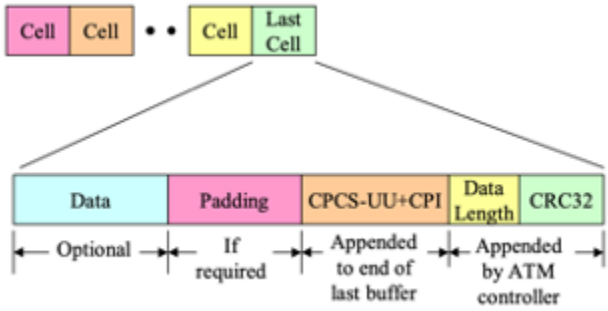


Figure 3: AAL5 Example Frame

Formats of the ATM Cell Header

The ATM standards groups have defined two header formats. The User-Network Interface (UNI) header format is defined by the UNI specification, and the Network-to-Network Interface (NNI) header format is

defined by the NNI specification. The UNI specification defines communications between ATM endpoints (such as workstations and routers) and switch routers in private ATM networks. The format of the UNI cell header is shown in Figure 4.

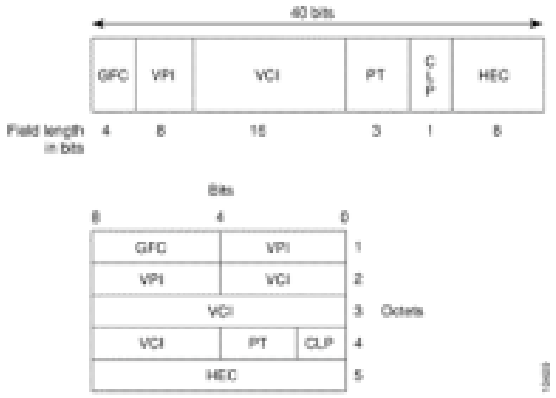


Figure 4: UNI header cell

The UNI header consists of the following fields:
GFC—4 bits of generic flow control that are used to provide local functions, such as identifying multiple stations that share a single ATM interface. The GFC field is typically not used and is set to a default value.
VPI—8 bits of virtual path identifier that is used, in conjunction with the VCI, to identify the next destination of a cell as it passes through a series of switch routers on its way to its destination. VCI—16 bits of virtual channel identifier that is used, in conjunction with the VPI, to identify the next destination of a cell as it passes through a series of switch routers on its way to its destination. PT—3 bits of payload type. The first bit indicates whether the cell contains user data or control data. If the cell contains user data, the second bit indicates congestion, and the third bit indicates whether the cell is the last in a series of cells that represent a single AAL5 frame.
CLP—1 bit of congestion loss priority that indicates whether the cell should be discarded if it encounters extreme congestion as it moves through the network.
HEC—8 bits of header error control that are a checksum calculated only on the header itself.

The NNI specification defines communications between switch routers. The format of the NNI header is shown in Figure 5.

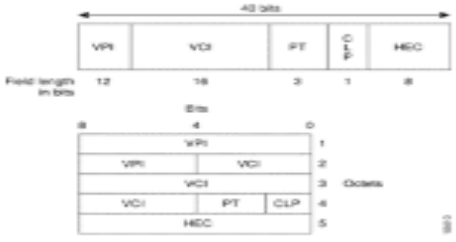


Figure 5: NNI Header Format

2) Features of our UVM environment :

The **UVM environment** includes multiple **Utopia agents** and a **CPU agent,** which is extended from the **UVM agent** class. We used separate agents for Utopia Transmitters (TX) and Receivers (RX). Each **RX agent** has a **Utopia driver** (extended from **UVM driver** class) that converts transactions into pin level activity on the **Utopia RX interface**, **Utopia RX monitor** (extended from **UVM monitor** class) that converts pin level activity from the **Utopia RX interface** into transactions for **coverage analysis** (a separate class is created for coverage analysis that is extended from **UVM subscriber** class), and a **Utopia sequence**r (extended from **UVM sequencer** class). Each **TX agent** has a **Utopia TX monito**r(extended from **UVM monitor** class) that converts pin level activity from the **Utopia TX interface** into transactions and sends them to the scoreboard, but it doesn't have a UVM driver or sequencer since it's a **passive agent**. We instantiated an array of Utopia RX and Utopia TX interfaces (1 for each Rx port and Tx port) in the top level testbench, and set the virtual interfaces (Utopia RX in driver, Utopia TX in monitor) using the **UVM config db**. We defined **Utopia sequence items** - UNI cell and NNI cell (extended from **UVM sequence item** class), where the properties of the classes are ported from Chris Spear's SV testbench. We defined a **Utopia sequence** base class (extended from the **UVM sequence** class) that randomizes the properties of the UNI cell and NNI cell. The UVM environment also has a **scoreboard** (extends from UVM scoreboard) that communicates with all the monitor Utopia interfaces (RX and TX) using **UVM TLM analysis FIFOs,** and a **utopia_coverage** object that also uses **UVM TLM analysis FIFOs** to communicate with the RX monitor. The scoreboard checks if the actual contents of the UNI cell (from the TX interface) matches the expected contents (from the RX interface) of the cell. Additionally, the number of expected and actual cells for a TX port are also compared in the scoreboard.

The **CPU agent** has a **CPU driver** (extended from **UVM driver** class) that drives the signals of the CPU interface to an LUT register file (contains port forwarding address), and a **CPU sequencer** (extended from **UVM sequencer** class) .

The Utopia environment class is instantiated in a **base test class** (extended from **UVM test class**) , where the UVM sequences are also defined. Each testcase extends from the base test class. In the run phase task, these sequences are launched on the sequencer handles for the corresponding Utopia RX interface. A **utopia_config** object (extends from **uvm_object**) contains the configuration parameters that can be set from the test case, and then accessed by the uvm components .

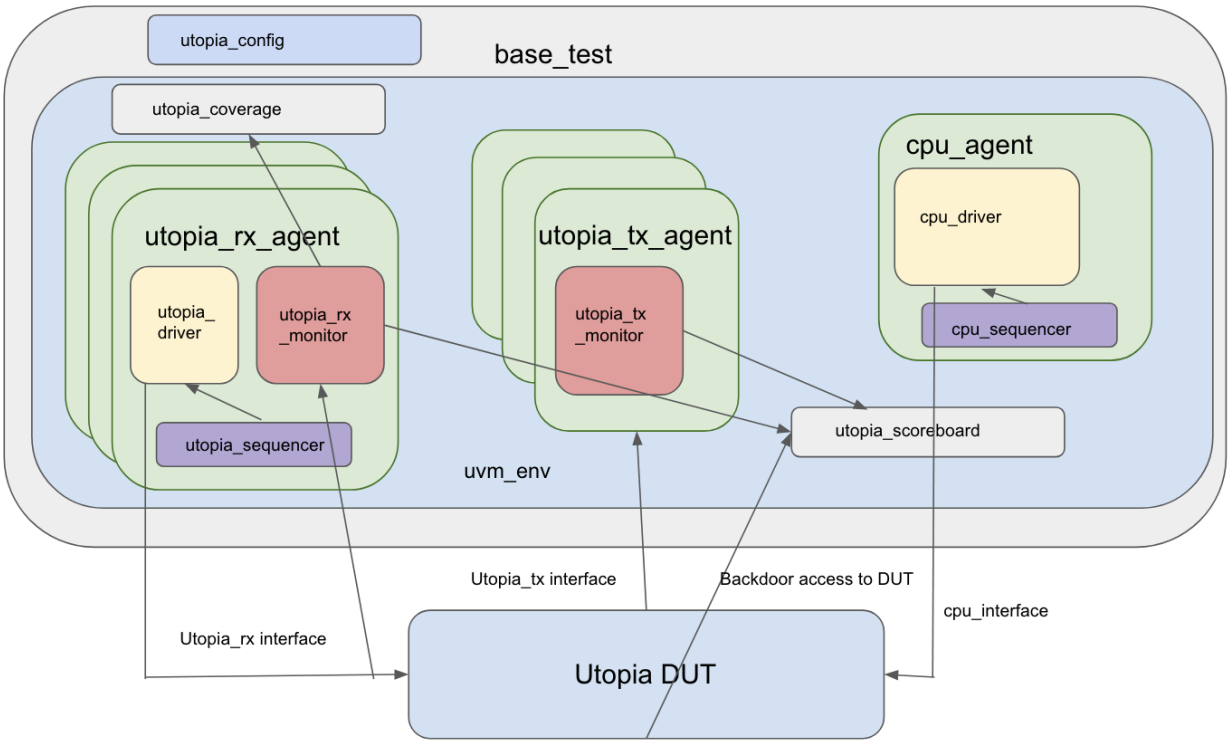


Fig 1: All the static components of the UVM testbench. Sequences and sequence items are dynamically created and launched on the sequencer.

The details of the test plan and coverage metrics will be explained in the next section.

3) Test plan -- what sorts of bugs are we trying to detect?

List of test cases :

1) **uni_seq_test** : Launch UNI base sequences on all the sequencers (RX sequencers in total for RX agents) sequentially, such that the attributes of the UNI cell are randomized, and the port forwarding address is also randomized in the cpu sequence. Covers all scenarios except port forwarding to all TX ports, which is handled separately.
   These constrained randomized test cases should ideally cover packet transmission (one-to-one , one-to-many transmission) from all the RX ports to all the TX ports. Connectivity bugs can be detected in case tests fail. This test is repeated multiple times before moving to the next test.
   Launch UNI base sequences on all the sequencers (RX sequencers in total for RX agents) sequentially, such that the attributes of the UNI cell are randomized,

and the port forwarding address is also randomized in the cpu sequence. An additional constraint is placed - the data must be forwarded to all TX ports. These constrained randomized test cases should ideally cover packet transmission (one-to-all transmission) from all the RX ports to all the TX ports. Connectivity bugs can be detected in case tests fail. This test is repeated multiple times and the final coverage report is generated.

2) **uni_parallel_test :** Launch UNI base sequences on all the sequencers (RX sequencers in total for RX agents) in parallel (using fork join), such that the attributes of the UNI cell are randomized, and the port forwarding address is also randomized in the cpu sequence. Covers all scenarios except port forwarding to all TX ports, which is handled separately. These constrained randomized test cases should ideally cover packet transmission (one-to-one , one-to-many transmission) from all the RX ports to all the TX ports. By launching the sequences in parallel on the RX ports, we want to ensure there are no arbitration issues in the DUT when packets from different RX ports are being forwarded to the same TX ports. Connectivity, arbitration bugs can be detected in case tests fail. This test is repeated multiple times before moving to the next test.
   Launch UNI base sequences on all the sequencers (RX sequencers in total for RX agents) in parallel (using fork join), such that the attributes of the UNI cell are randomized, and the port forwarding address is also randomized in the cpu sequence. An additional constraint is placed - the data must be forwarded to all TX ports. These constrained randomized test cases should ideally cover packet transmission (one-to-all transmission) from all the RX ports to all the TX ports. By launching the sequences in parallel on the RX ports, we want to ensure there are no arbitration issues in the DUT when packets from different RX ports are being forwarded to the same TX ports. Connectivity, arbitration bugs can be detected in case tests fail. This test is repeated multiple times before moving to the next test.
3) Additional directed test cases based on functional coverage holes will be added to the plan in the final report - Since we were able to achieve 100% functional coverage on the Forward ports and RX source ports, we didn't add any other directed test cases.

Results :

We used Mentor Questa 2021.3 simulator on edaplayground to run our simulations.

A detailed log file (Utopia_simulation_log_with_all_packet_transmission_messages.txt) has been generated that shows the UVM testbench topology and all the debug messages during UNI cell packet transmission.

UVM Topology copied from the simulation log :

```
# ----------------------------------------------------------------
# Name                       Type                   Size  Value
# ----------------------------------------------------------------
# uvm_test_top               uni_seq_test           -     @383
#   env                      utopia_env             -     @395
#     cpu_agnt               cpu_agent              -     @716
#       cpu_drv              cpu_driver             -     @725
#         rsp_port           uvm_analysis_port      -     @742
#         seq_item_port      uvm_seq_item_pull_port -     @733
#       cpu_sqncr            cpu_sequencer          -     @751
#         rsp_export         uvm_analysis_export    -     @759
#         seq_item_export    uvm_seq_item_pull_imp  -     @865
#         arbitration_queue  array                  0     -
#         lock_queue         array                  0     -
#         num_last_reqs      integral               32    'd1
#         num_last_rsps      integral               32    'd1
#     utopia_agnt_Rx[0]      utopia_rx_agent        -     @415
#       utopia_drv           utopia_driver          -     @881
#         rsp_port           uvm_analysis_port      -     @898
#         seq_item_port      uvm_seq_item_pull_port -     @889
#       utopia_mon           utopia_rx_monitor      -     @1030
#         item_collected_port      uvm_analysis_port      -     @1038
#         item_collected_port_cov  uvm_analysis_port      -     @1047
#       utopia_sqncr         utopia_sequencer       -     @907
#         rsp_export         uvm_analysis_export    -     @915
#         seq_item_export    uvm_seq_item_pull_imp  -     @1021
#         arbitration_queue  array                  0     -
#         lock_queue         array                  0     -
```

```
#       num_last_reqs          integral                32    'd1
#       num_last_rsps          integral                32    'd1
#     utopia_agnt_Rx[1]          utopia_rx_agent           -     @423
#       utopia_drv             utopia_driver           -     @1064
#       rsp_port             uvm_analysis_port         -     @1081
#       seq_item_port          uvm_seq_item_pull_port    -     @1072
#       utopia_mon             utopia_rx_monitor         -     @1213
#       item_collected_port    uvm_analysis_port         -     @1221
#       item_collected_port_cov  uvm_analysis_port         -     @1230
#       utopia_sqncr           utopia_sequencer          -     @1090
#       rsp_export           uvm_analysis_export       -     @1098
#       seq_item_export        uvm_seq_item_pull_imp     -     @1204
#       arbitration_queue      array                   0     -
#       lock_queue           array                   0     -
#       num_last_reqs          integral                32    'd1
#       num_last_rsps          integral                32    'd1
#     utopia_agnt_Rx[2]          utopia_rx_agent           -     @431
#       utopia_drv             utopia_driver           -     @1247
#       rsp_port             uvm_analysis_port         -     @1264
#       seq_item_port          uvm_seq_item_pull_port    -     @1255
#       utopia_mon             utopia_rx_monitor         -     @1396
#       item_collected_port    uvm_analysis_port         -     @1404
#       item_collected_port_cov  uvm_analysis_port         -     @1413
#       utopia_sqncr           utopia_sequencer          -     @1273
#       rsp_export           uvm_analysis_export       -     @1281
#       seq_item_export        uvm_seq_item_pull_imp     -     @1387
#       arbitration_queue      array                   0     -
#       lock_queue           array                   0     -
#       num_last_reqs          integral                32    'd1
#       num_last_rsps          integral                32    'd1
#     utopia_agnt_Rx[3]          utopia_rx_agent           -     @439
#       utopia_drv             utopia_driver           -     @1430
#       rsp_port             uvm_analysis_port         -     @1447
#       seq_item_port          uvm_seq_item_pull_port    -     @1438
#       utopia_mon             utopia_rx_monitor         -     @1579
#       item_collected_port    uvm_analysis_port         -     @1587
#       item_collected_port_cov  uvm_analysis_port         -     @1596
#       utopia_sqncr           utopia_sequencer          -     @1456
#       rsp_export           uvm_analysis_export       -     @1464
```

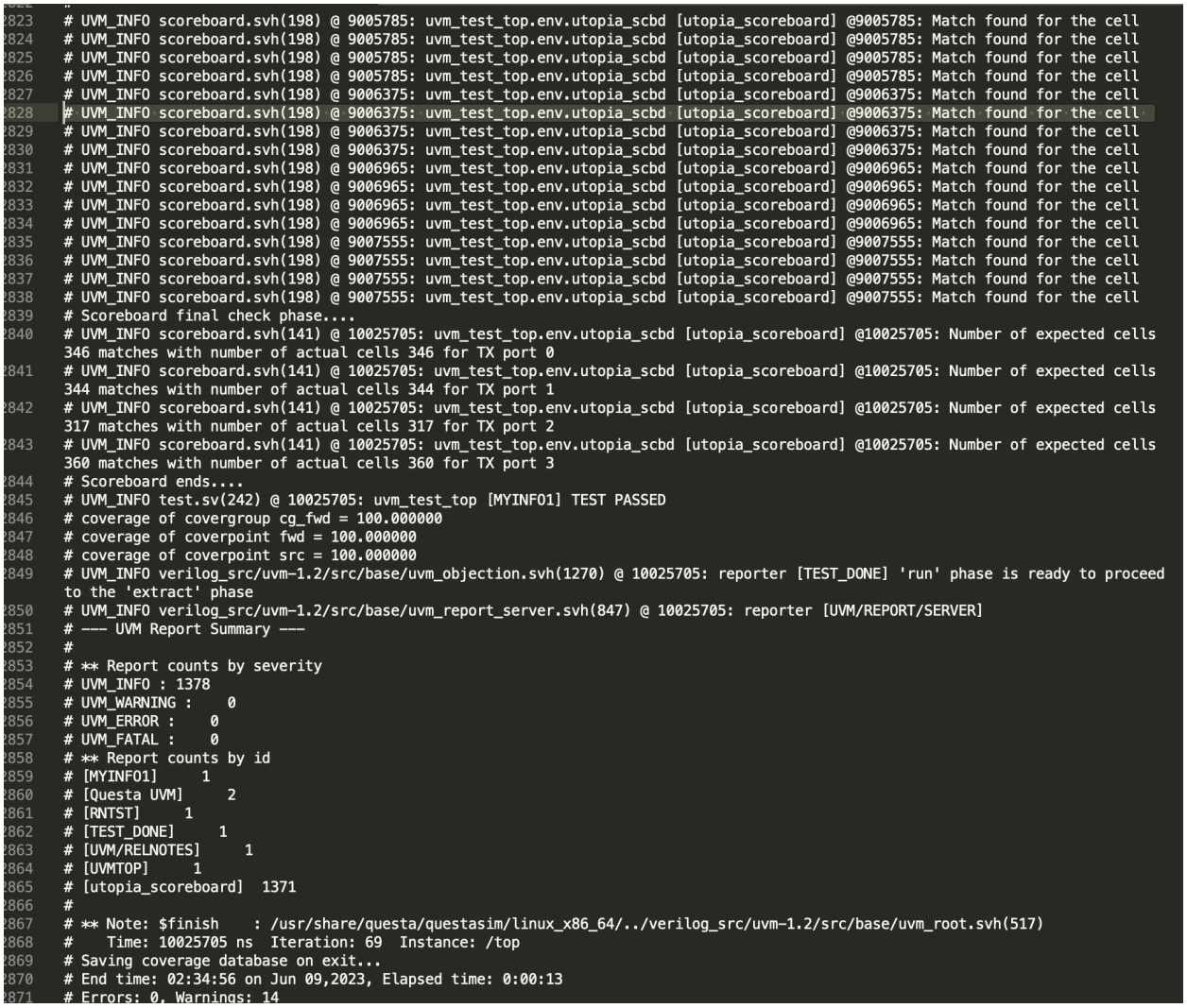
```
#       seq_item_export        uvm_seq_item_pull_imp      -    @1570
#       arbitration_queue      array                 0    -
#       lock_queue             array                 0    -
#       num_last_reqs          integral              32   'd1
#       num_last_rsps          integral              32   'd1
#     utopia_agnt_Tx[0]        utopia_tx_agent         -    @447
#       utopia_mon             utopia_tx_monitor       -    @1613
#       item_collected_port    uvm_analysis_port       -    @1621
#     utopia_agnt_Tx[1]        utopia_tx_agent         -    @455
#       utopia_mon             utopia_tx_monitor       -    @1632
#       item_collected_port    uvm_analysis_port       -    @1640
#     utopia_agnt_Tx[2]        utopia_tx_agent         -    @463
#       utopia_mon             utopia_tx_monitor       -    @1651
#       item_collected_port    uvm_analysis_port       -    @1659
#     utopia_agnt_Tx[3]        utopia_tx_agent         -    @471
#       utopia_mon             utopia_tx_monitor       -    @1670
#       item_collected_port    uvm_analysis_port       -    @1678
#     utopia_cov               utopia_coverage         -    @487
#       analysis_imp           uvm_analysis_imp        -    @495
#       tlm_a_fifo_cov[0]      uvm_tlm_analysis_fifo #(T) -    @504
#       analysis_export        uvm_analysis_imp        -    @548
#       get_ap                 uvm_analysis_port       -    @539
#       get_peek_export        uvm_get_peek_imp        -    @521
#       put_ap                 uvm_analysis_port       -    @530
#       put_export             uvm_put_imp             -    @512
#       tlm_a_fifo_cov[1]      uvm_tlm_analysis_fifo #(T) -    @557
#       analysis_export        uvm_analysis_imp        -    @601
#       get_ap                 uvm_analysis_port       -    @592
#       get_peek_export        uvm_get_peek_imp        -    @574
#       put_ap                 uvm_analysis_port       -    @583
#       put_export             uvm_put_imp             -    @565
#       tlm_a_fifo_cov[2]      uvm_tlm_analysis_fifo #(T) -    @610
#       analysis_export        uvm_analysis_imp        -    @654
#       get_ap                 uvm_analysis_port       -    @645
#       get_peek_export        uvm_get_peek_imp        -    @627
#       put_ap                 uvm_analysis_port       -    @636
#       put_export             uvm_put_imp             -    @618
#       tlm_a_fifo_cov[3]      uvm_tlm_analysis_fifo #(T) -    @663
#       analysis_export        uvm_analysis_imp        -    @707
```

```
#      get_ap              uvm_analysis_port        -    @698
#      get_peek_export       uvm_get_peek_imp       -    @680
#      put_ap              uvm_analysis_port        -    @689
#      put_export            uvm_put_imp            -    @671
#    utopia_scbd            utopia_scoreboard       -    @479
#     tlm_a_fifo_rcvd[0]      uvm_tlm_analysis_fifo #(T) -    @1688
#      analysis_export       uvm_analysis_imp        -    @1732
#      get_ap              uvm_analysis_port        -    @1723
#      get_peek_export       uvm_get_peek_imp       -    @1705
#      put_ap              uvm_analysis_port        -    @1714
#      put_export            uvm_put_imp            -    @1696
#     tlm_a_fifo_rcvd[1]      uvm_tlm_analysis_fifo #(T) -    @1741
#      analysis_export       uvm_analysis_imp        -    @1785
#      get_ap              uvm_analysis_port        -    @1776
#      get_peek_export       uvm_get_peek_imp       -    @1758
#      put_ap              uvm_analysis_port        -    @1767
#      put_export            uvm_put_imp            -    @1749
#     tlm_a_fifo_rcvd[2]      uvm_tlm_analysis_fifo #(T) -    @1794
#      analysis_export       uvm_analysis_imp        -    @1838
#      get_ap              uvm_analysis_port        -    @1829
#      get_peek_export       uvm_get_peek_imp       -    @1811
#      put_ap              uvm_analysis_port        -    @1820
#      put_export            uvm_put_imp            -    @1802
#     tlm_a_fifo_rcvd[3]      uvm_tlm_analysis_fifo #(T) -    @1847
#      analysis_export       uvm_analysis_imp        -    @1891
#      get_ap              uvm_analysis_port        -    @1882
#      get_peek_export       uvm_get_peek_imp       -    @1864
#      put_ap              uvm_analysis_port        -    @1873
#      put_export            uvm_put_imp            -    @1855
#     tlm_a_fifo_sent[0]      uvm_tlm_analysis_fifo #(T) -    @1900
#      analysis_export       uvm_analysis_imp        -    @1944
#      get_ap              uvm_analysis_port        -    @1935
#      get_peek_export       uvm_get_peek_imp       -    @1917
#      put_ap              uvm_analysis_port        -    @1926
#      put_export            uvm_put_imp            -    @1908
#     tlm_a_fifo_sent[1]      uvm_tlm_analysis_fifo #(T) -    @1953
#      analysis_export       uvm_analysis_imp        -    @1997
#      get_ap              uvm_analysis_port        -    @1988
#      get_peek_export       uvm_get_peek_imp       -    @1970
```

```
#      put_ap              uvm_analysis_port        -    @1979
#      put_export            uvm_put_imp            -    @1961
#    tlm_a_fifo_sent[2]      uvm_tlm_analysis_fifo #(T) -    @2006
#      analysis_export       uvm_analysis_imp       -    @2050
#      get_ap              uvm_analysis_port        -    @2041
#      get_peek_export       uvm_get_peek_imp       -    @2023
#      put_ap              uvm_analysis_port        -    @2032
#      put_export            uvm_put_imp            -    @2014
#    tlm_a_fifo_sent[3]      uvm_tlm_analysis_fifo #(T) -    @2059
#      analysis_export       uvm_analysis_imp       -    @2103
#      get_ap              uvm_analysis_port        -    @2094
#      get_peek_export       uvm_get_peek_imp       -    @2076
#      put_ap              uvm_analysis_port        -    @2085
#      put_export            uvm_put_imp            -    @2067
# ------------------------------------------------------------------
#
```

```
  covergroup CG_fwd();

    option.per_instance = 1;

    SRC: coverpoint src {
      bins src_bin[] = {[0:`RxPorts - 1]};
      //option.weight = 0;
      bins bad_values  = default;
    }

    FWD : coverpoint fwd {
      bins fwd_bin[] = {[0:`TxPorts - 1 ]};
      //option.weight = 0;
      bins bad_values = default;
    }

    CROSS : cross src,fwd;

endgroup : CG_fwd
```

Fig 2 : Covergroup and coverpoints for functional coverage. Cover all RX ports (src), TX ports (fwd) and cross combination of RX,TX ports.



```
#
# UVM_INFO scoreboard.svh(198) @ 9005785: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9005785: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9005785: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9005785: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9005785: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9005785: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9005785: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9005785: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006375: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006375: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006375: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006375: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006375: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006375: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006375: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006375: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006965: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006965: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006965: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006965: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006965: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006965: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9006965: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9006965: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9007555: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9007555: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9007555: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9007555: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9007555: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9007555: Match found for the cell
# UVM_INFO scoreboard.svh(198) @ 9007555: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @9007555: Match found for the cell
# Scoreboard final check phase....
# UVM_INFO scoreboard.svh(141) @ 10025705: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @10025705: Number of expected cells
346 matches with number of actual cells 346 for TX port 0
# UVM_INFO scoreboard.svh(141) @ 10025705: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @10025705: Number of expected cells
344 matches with number of actual cells 344 for TX port 1
# UVM_INFO scoreboard.svh(141) @ 10025705: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @10025705: Number of expected cells
317 matches with number of actual cells 317 for TX port 2
# UVM_INFO scoreboard.svh(141) @ 10025705: uvm_test_top.env.utopia_scbd [utopia_scoreboard] @10025705: Number of expected cells
360 matches with number of actual cells 360 for TX port 3
# Scoreboard ends....
# UVM_INFO test.sv(242) @ 10025705: uvm_test_top [MYINFO1] TEST PASSED
# coverage of covergroup cg_fwd = 100.000000
# coverage of coverpoint fwd = 100.000000
# coverage of coverpoint src = 100.000000
# UVM_INFO verilog_src/uvm-1.2/src/base/uvm_objection.svh(1270) @ 10025705: reporter [TEST_DONE] 'run' phase is ready to proceed
to the 'extract' phase
# UVM_INFO verilog_src/uvm-1.2/src/base/uvm_report_server.svh(847) @ 10025705: reporter [UVM/REPORT/SERVER]
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 1378
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [MYINFO1]     1
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVM/RELNOTES]     1
# [UVMTOP]     1
# [utopia_scoreboard]  1371
#
# ** Note: $finish    : /usr/share/questa/questasim/linux_x86_64/../verilog_src/uvm-1.2/src/base/uvm_root.svh(517)
#    Time: 10025705 ns  Iteration: 69  Instance: /top
# Saving coverage database on exit...
# End time: 02:34:56 on Jun 09,2023, Elapsed time: 0:00:13
# Errors: 0, Warnings: 14
```

Fig 3 : Snapshot of simulation log, UVM info messages display the contents of the UNI cell, and the scoreboard messages after comparing the expected and actual data cells. Finally, the number of expected and actual data cells is compared. The simulation passes with 0 UVM_ERROR and 0 UVM_FATAL. Functional coverage is 100%.

We ran both test cases uni_seq_test and uni_parallel_test and they are passing without any errors.

```
194     uvm_config_db #(virtual
    cpu_ifc)::set(null,"uvm_test_top.env.cpu_agnt.cpu_drv", "mif", mif);
195
196   end
197
198   initial begin
199
200     $dumpfile("dump.vcd");
201     $dumpvars(3,top);
202     $set_coverage_db_name("utopia_cov");
203
204     //run_test("uni_seq_test");     // choose your test here
205
206     run_test("uni_parallel_test");
207     //run_test("uni_error_injection_test");
208
209     uvm_top.print_topology();
210
211   end
212
213
214 endmodule : top
215
```

Fig 4: The test case can be selected in top.sv in the code snippet shown above. We have two tests - uni_seq_test and uni_parallel_test.

The key feature of our UVM environment is the way we have connected a configurable number of RX and TX utopia agents. We used UVM TLM Analysis FIFOs to connect the monitors with the scoreboard, instead of using the more conventional analysis export implementation and write function in the scoreboard, because using an array of TLM FIFOs allows us to scale the number of 'listening channels' on the scoreboard. Using write functions in the scoreboard, we could only have connected one monitor analysis port with an analysis export of the scoreboard. Using TLM Analysis FIFOs also gives us the flexibility to store expected transactions in tlm_a_fifo_sent and the actual transactions in tlm_a_fifo_rcvd as and when the DUT sends them. The FIFOs get filled asynchronously, and we can compare the transactions at a later time in the simulation, instead of doing it immediately as in the case of the Write function implementation.

```
class utopia_scoreboard extends uvm_scoreboard;

 `uvm_component_utils(utopia_scoreboard)

  uvm_tlm_analysis_fifo  #(NNI_cell) tlm_a_fifo_rcvd[`TxPorts];
  uvm_tlm_analysis_fifo  #(NNI_cell) tlm_a_fifo_sent[`RxPorts];
```

Fig 5: Declaring an array of UVM TLM analysis FIFOs in Scoreboard

```
  virtual function void connect_phase(uvm_phase phase);

    super.connect_phase(phase);

//utopia_agnt.utopia_mon.item_collected_port.connect(utopia_scbd.item_collect
ed_export);

    for(l=0;l<`TxPorts;l++)
utopia_agnt_Tx[l].utopia_mon.item_collected_port.connect(utopia_scbd.tlm_a_fi
fo_rcvd[l].analysis_export);

    for(k=0;k<`RxPorts;k++)
utopia_agnt_Rx[k].utopia_mon.item_collected_port.connect(utopia_scbd.tlm_a_fi
fo_sent[k].analysis_export);


    for(m=0;m<`RxPorts;m++)
utopia_agnt_Rx[m].utopia_mon.item_collected_port_cov.connect(utopia_cov.tlm_a
_fifo_cov[m].analysis_export);
```

Fig 6: Connecting the Monitor's analysis port to the TLM analysis FIFO's analysis export. Similar connections are made for the coverage TLM analysis FIFOs.

We share the rest of our results in the 'results' directory.

Conclusion : We developed a UVM testbench framework to verify the Utopia Interface for ATM protocol. We took a reference SystemVerilog testbench and ported the testbench to include UVM compatible classes. By porting to UVM, we managed to decouple the testbench architecture from the testcases. We developed a test plan to verify the key features of the Utopia packet crossbar switcher and defined functional coverage metrics. We were able to set up a scoreboard that compares all the transactions. All the planned test cases passed and we were able to achieve 100% functional coverage on the IP.