Verification of Interconnect RTL Code for Memory-Centric Computing using UVM

Hyuk Je Kwon
Electronics and Telecommunication Research Institute, Daejeon, Korea
heavenwing@etri.re.kr

Myeong-Hoon Oh
Electronics and Telecommunication Research Institute, Daejeon, Korea
mhoonoh@etri.re.kr

Won-ok Kwon
Electronics and Telecommunication Research Institute, Daejeon, Korea
happy@etri.re.kr

2021 International Conference on Electronics, Information, and Communication (ICEIC) | 978-1-7281-9161-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICEIC51217.2021.9369792

***Abstract*—This document is about the verification of an interconnect (i.e. switch) RTL code that is based on Gen-Z protocol using Universal Verification Methodology (UVM). Ports in the switch for transmission packets are connected to virtual interfaces with UVM. The packets that are generated in the UVM environment are transmitted into the ports of the switch through the virtual interfaces. For verifying the switch logic, we designed sequence items and a virtual sequencer and simulated it.**

Keywords—UVM; virtual sequencer; Gen-Z Switch

I. INTRODUCTION

UVM (universal verification methodology) is for verifying DUT that has been getting complicated. In many cases, we designed a DUT in RTL and also coded a testbench for testing or verifying the DUT. After testing the DUT, we generally trashed the test code out. During the test and verifying the DUT, we could not catch some bugs even if those are in it if we use user-coded testbench. And we could be not sure that some systemic errors were found out, not bugs in the code. In the user-coded testbench, there are some limits for testing or verifying all user-designed code. Because users can not define all cases, the user-coded testbench has the limit not to find all bugs or systemic errors. To overcome it, many EDA companies have made tools for verification and reusable, and the tools have been used it now. The tools can make to generate some stimulus codes by several users simultaneously [1][2]. Recently, tools are unified into one, that is, UVM [3].

This paper is to verify the interconnect (i.e. switch) RTL code (based Gen-Z 1.0 protocol) for memory-centric computing, using UVM. The switch has several ports that transmit or receive packets to or from other memory-centric components. To put the stimulus into switch ports, we designed sequence items of UVM and verified switch codes. In this paper, we show the designed sequence item and virtual sequencer. And we show the structure that is for verifying switch RTL.

II. STRUCTURE OF GEN-Z SWITCH

The Gen-Z protocol is a universal system interconnect that supports high bandwidth and low latency. It supports byte-addressable memory access, block memory access, input/output (I/O) device access, messaging, and accelerator access to transparently connect all components to the Gen-Z fabric [4]. The Gen-Z switch relays end-to-end packets between an ingress interface and an egress interface. From a communications perspective, Gen-Z adopts packet communication to perform operations uniformly between various devices. Packets are classified into end-to-end (EE) packets and link-local (LL) packets. The EE packets are used for inter-component operations such as read and write. The LL packet operates between interfaces and handles link-specific operations such as flow control and physical layer management.

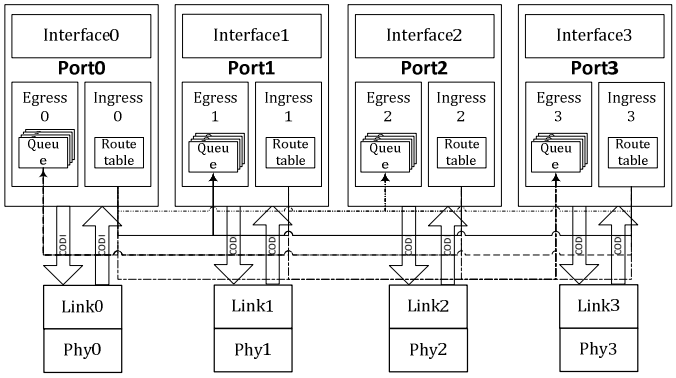

Fig. 1. Gen-Z Switch for memory-centric computing.

A. Ingress port

The switch's basic function is to transfer Gen-Z packets (EE) into Gen-Z components. To achieve it, the switch can have several links, and each link is composed of two-port, Ingress and Egress. If Gen-Z packets are received in ingress port, it checks two CRC on packets. Preluded CRC (PCRC) protects against errors that could cause the packet virtual circuit and length to be incorrectly interpreted. All Gen-Z packets contain a 24-bit End-to-end CRC (ECRC) field, which protects the entire packet. Unlike the PCRC validation, a packet can be relayed despite having a bad ECRC.

After checking CRC, the switch determines a link that is as a passage of the packet to transfer. To find the link, it uses a routing table that is as a lookup table. Each ingress port contains a unique route table. We will deal with it next section.

B. Egress port

An egress port has a queue for storing packets that come from each ingress port. In Gen-Z Switch, there are several ingress ports so that the queue exists in each egress port to store packets of an ingress port, exactly one-to-one. When the packets are memorized in the queue, there is a valid signal that indicates a packet is valid or not. The valid signal is also stored in the queue with the packet. A tail signal also indicates a packet's end. When the packet in the queue is selected to transmit to the link, the selection algorithm is the round-robin.

C. Route Table

A routing table is like an address book. In the Gen-Z packet, there is a DCID (destination component identification) which shows the final destination of the packet. The DCID is used for an address to search the routing table that has the number of the output port.

III. Stimulus for Gen-Z Switch

The switch testbench's hierarchy is shown in Fig. 2. An agent that is composed of a driver, sequencer, monitor, which can be multiple under an 'env' environment. A scoreboard is scoring the responses from DUT. Stimuli that are sequence items are managed and generated by sequence, the driver transmits some stimuli through a virtual interface to DUT.

A. Configuring Environment

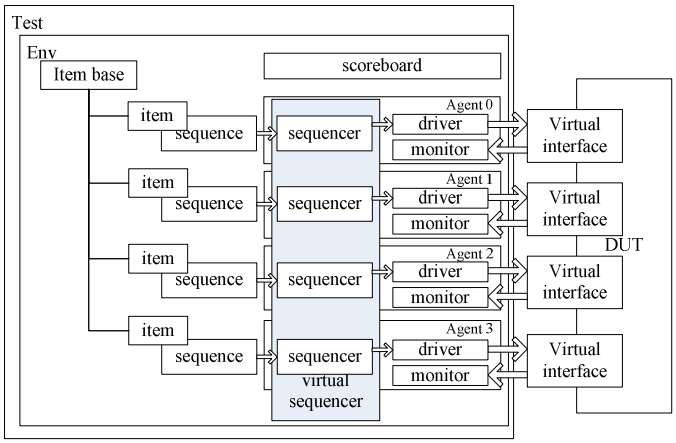

Fig. 2. Testbench's hierarchy and virtual sequencer. Under an environment 'Env', there can be placed multiple agents. Each agent has a driver and monitor, sequencer and it executes the sequencer individually. To drive the packets into all ports(virtual interface) of DUT simultaneously, all sequencers are combined into the virtual sequencer which can transfer the packets into all drivers.

The designed switch can have multiport but it has 4 ports in this simulation. Each port corresponds with one agent that has a driver, monitor, and sequencer. An agent can drive the packet that is configured in sequence into DUT, respectively. However, we want to simulate all ports on DUT concurrently with different packets. So, we use a virtual sequencer in order to do it. The configured virtual sequencer is shown in Fig. 3. The virtual sequencer is a set of sequences in each agent so that it can execute all sequences at the same time or at different times.

```
class switch_virtual_sequencer extends uvm_sequencer;
    `uvm_component_utils (switch_virtual_sequencer)
    function     new(string     name     ="switch_virtual
sequencer", uvm_component parent);
        super.new(name, parent);
    endfunction
    uvm_sequencer #(switch_pkt_header_port0_item)
                            sequencer_pkt_port0;
    uvm_sequencer #(switch_pkt_header_port1_item)
                            sequencer_pkt_port1;
    uvm_sequencer #(switch_pkt_header_port2_item)
                            sequencer_pkt_port2;
    uvm_sequencer #(switch_pkt_header_port3_item)
                            sequencer_pkt_port3;
endclass
```

Fig. 3. Sequencers of each agent are merged into a virtual sequencer

B. Making Stimulus data

In the Gen-Z packet, there are several data compromising the packet. The main goal of Switch is to route the packet according to DCIDs, so we make the DCIDs as a random number shown in Fig. 4. Address, Length, and Payload are also a random number for distinguishing the packets.

```
rand bit [11:0] dest_id;
constraint c_dest_id { dest_id>=1; dest_id<8; }
```

Fig. 4. DCID constraint. Reserved word, 'rand', 'constraint' in systemverilog

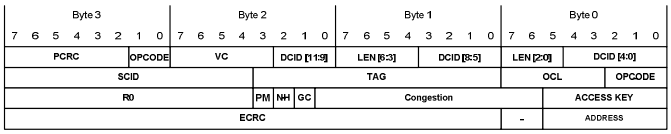

Fig. 5. Structure of Gen-Z packet for simulation.

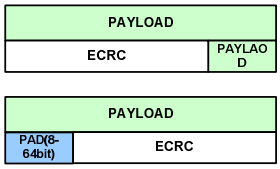

Fig. 6. The end of the packet depends on the size of the payload. A packet must be 4-byte alignment and PAD could be added if the size of the packet is not.

The Structure of the Gen-Z packet of simulation is shown in Fig. 5. The size of the Gen-Z packet must be 4-byte alignment and the length of the payload would be not 4-byte alignment. In this case, some pads could be added at the end of the packet.

C. Deliver Stimulus

'p_sequencer' is a point of virtual sequencer (i.e. `switch_virtual_sequencer` in Fig. 3) and each sequence that is made in the sequence item starts on the p_sequencer, `sequencer_pkt_port0.start(p_sequencer.sequencer_pkt_port0)`.

```
class      switch_transaction_virtual_sequence      extends
uvm_sequence;
  `uvm_object_utils(switch_transaction_virtual_sequence)
  `uvm_declare_p_sequencer (switch_virtual_sequencer)

  switch_sequence_pkt_port0 sequence_pkt_port0;
     .  .  .  .
```

```
  task body();
fork
     sequence_pkt_port0.start
          (p_sequencer.sequencer_pkt_port0);
  join
        .  .  .  .
endtask

  endclass : switch_transaction_virtual_sequence
```

Fig. 7. Virtual sequencer delivers to each driver a sequence.

D. Connection with DUT

```
initial  uvm_config_db#(virtual  codi_if#(.WIDTH(WIDTH)))::
set(uvm_root::get(),"*" , "vif0", link_sw_pkt_vif[0]);

initial  uvm_config_db#(virtual  codi_if#(.WIDTH(WIDTH)))::
set(uvm_root::get(),"*" , "svif0", sw_link_pkt_vif[0]);

switch_top switch_top_i (
 …
   .link_sw_pkt    (link_sw_pkt_vif    ),
   .sw_link_pkt    (sw_link_pkt_vif    ),
 …
);
```

Fig. 8. Interfacing to DUT(switch_top) using virtual interfaces in order to communicate with UVM in testbench top module. 'link_sw_pkt_vif', 'sw_link_pkt_vif' are virtual interfaces.

UVM is written by object-oriented programming, which is based on "Class" [1]. All components of DUT are static, which is also exchanged in the form of signals, wire, net at all levels. Hence the communication between the DUT and testbench cannot be like the one in traditional test benches i.e. port-based connection of the DUT ports to the testbench ports. The virtual interface acts as a medium to connect the DUT and the testbench (i.e. UVM) [5]. Testbench access the DUT signals via the virtual interface and vice versa.

IV. Simulation.

The Hierarchy of the Gen-Z switch's testbench is shown in Fig. 9.

```
----------------------------------------------------------------------
Name                        Type                       Size  Value
----------------------------------------------------------------------
uvm_test_top                switch_test                -  @2273
  env                       switch_env                 -  @2344
    agent_port0             switch_port0_agent         -  @2395
      driver_genz           switch_driver_genz_port0   -  @3238
        rsp_port            uvm_analysis_port          -  @3346
        seq_item_port       uvm_seq_item_pull_port     -  @3294
      monitor               switch_monitor             -  @4028
        ecrc_analysis_port uvm_analysis_port          -  @4140
        mon_analysis_port   uvm_analysis_port          -  @4090
      sequencer_pkt_port0   uvm_sequencer              -  @3379
        rsp_export          uvm_analysis_export        -  @3437
        seq_item_export     uvm_seq_item_pull_imp      -  @3997
        arbitration_queue   array                      0     -
        lock_queue          array                      0     -
        num_last_reqs       integral                   32  'd1
        num_last_rsps       integral                   32  'd1
    agent_port1             switch_port1_agent         -   @2426
                  .    .          .
    agent_port2             switch_port2_agent         -   @2457
                  .    .          .
    agent_port3             switch_port3_agent         -   @2488
                  .    .          .
    scoreboard              switch_scoreboard          -   @2519
      mon_analysis_export   uvm_analysis_imp           -   @7059
      port_in               uvm_analysis_imp_in        -   @7109
      port_out              uvm_analysis_imp_out       -   @7159
    scoreboard_ecrc         switch_scoreboard_ecrc     -   @2553
      ecrc_analysis_export  uvm_analysis_imp           -   @7213
    switch_virtual_seqr     switch_virtual_sequencer   -   @2586
      rsp_export            uvm_analysis_export        -   @2644
      seq_item_export       uvm_seq_item_pull_imp      -   @3206
      arbitration_queue     array                      0     -
      lock_queue            array                      0     -
      num_last_reqs         integral                   32  'd1
      num_last_rsps         integral                   32  'd1
```

Fig. 9. The Hierarchy of switch's Testbench using UVM

All of 'agent_port' is same each other, that is, it is reusable. So, if some ports on the switch are added and they have to be tested, several 'agent_port's can be just added without changing it.

There are 4 agents, one scoreboard, and one virtual sequencer(switch_virtual_seqr). The stimulus used in this simulation is shown in Fig. 10. There are four-type packets for PORT0~PORT3. Each packet has unique data and a different packet length.

```
[SEND PACKET PORT3]8ff133f1000fb4330000b009e5001083 HEAD
[SEND PACKET PORT2]c8065c17000476fa0000b009e5002002 HEAD
[SEND PACKET PORT1]842362c70006fd710000b009990010e3 HEAD
[SEND PACKET PORT3]0000000000000000c72d9268aac557ca PAYLOAD
[SEND PACKET PORT2]00000000000000081f52ef0d1942c01 PAYLOAD
[SEND PACKET PORT1]000000000000000009627e37e095961f7 PAYLOAD
[SEND PACKET PORT3]8f77040000000000c72d9268aac557ca ECRC
[SEND PACKET PORT2]0000000000000000b4763f5ecad8d48 PAYLOAD
[SEND PACKET PORT1]000000000000000512642e5b65fbe59 PAYLOAD
[SEND PACKET PORT0]6820ed850008a8cc0000b009990010e6 HEAD
[SEND PACKET PORT2]1154e600000000000b4763f5ecad8d48 ECRC
[SEND PACKET PORT1]  50b75e5623f74900512642e5b65fbe59 ECRC_END2
```

Fig. 10. Sending packets from each port. Port0 is composed of HEAD. Port2 HEAD, PAYLOAD, PAYLOAD, ECRC. Port1 HEAD, PAYLOAD, PAYLOAD, ECRC. Port3 HEAD, PAYLOAD, ECRC.

Each port0(✦) receives the Gen-Z packet, and it is routed by the receiving port's route table. The packet's DCID received on port0 is 0x006, and it is routed the packet to 'rtable_eid' (egress id) 0x001, i.e., egress port1. And the routed packet that is from port0 is shown in egress port1(✦). Different input packets received on each port are transmitted on routed ports at different times according to the routing algorithm of the egress port, even if the receiving time is the same.

V. Conclusion

This paper is about the UVM testbench for testing Gen-Z Switch logic in RTL. All sequencers on the agents are combined into the virtual sequencer simultaneously capable of sending packets to all drivers. The data for the Gen-Z packet made by sequence item is delivered to the driver through the virtual sequencer and the driver send to DUT (switch) through the virtual interfaces.

All packets on the ports of Gen-Z received route a specific port that is determined through the routing table using the packet's DCID as an address. The testbench can deliver multiple packets simultaneously without changing testbench codes and the content of the packets is also different.

Acknowledgment

This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00503, Researches on next generation memory-centric computing system architecture).

References

[1] Siddharth Raghuvanshi, Viswajeet Singh, "Review on Universal Verification Methodology(UVM) concepts for functional verification,"

International Journal of Electrical, Electronics and Data Communication, vol. 2, issue-3, pp. 101-107, March, 2014.

[2] Sharon Rosenberg, Kathleen A Meade, *A Practical Guide to Adopting the Universal Verification Methodology (UVM)*, Cadence Design Systems, Inc, 2010.

[3] Universal Verification Methodology (UVM) 1.2 Class Reference, Accelera System Initiative, June, 2014.

[4] Gen Z Core Specification version 1.0, 2018.

[5] Clifford E. Cummings, Heath Chambers, "SystemVerilog Virtual Classes, Method, Interfaces and their use in Verification and UVM," SNUG 2018 (Silicon Valley).

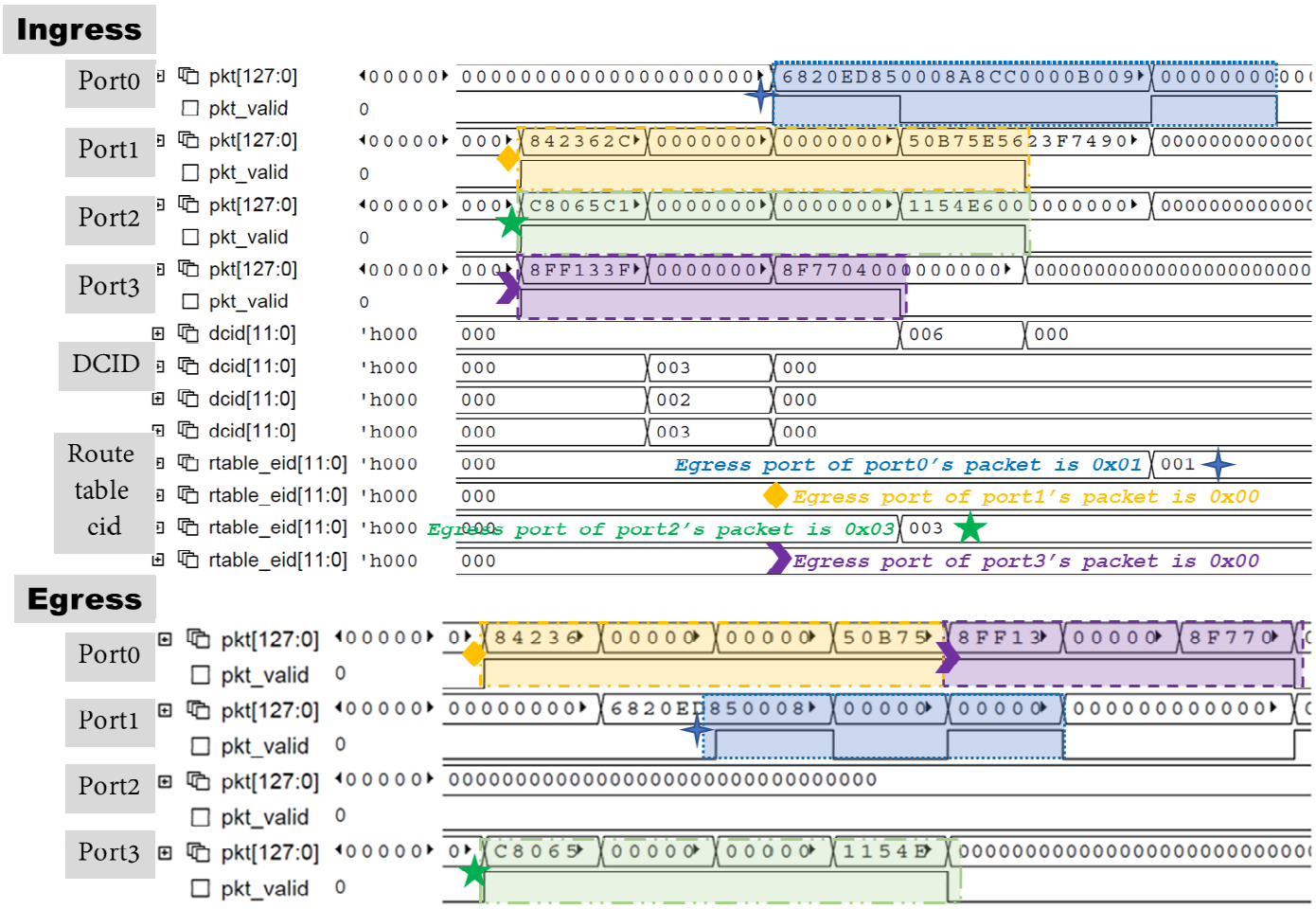

Fig. 11. Waveforms of Switching ingress ports to egress ports accroding to DCID. Ingress port0 data is 0x6820ED(✦) with 0x006(dcid), port1 data is 0x842362(◆) with 0x003(dcid), port2 data is 0xC8065C(★)with 0x002(dcid), port3 data is 0x8FF133(❯)with 0x003(dcid). Port0's egress port which is routed in the route table is 0x01, port1 0x00, port2 0x03, port3 0x00. Egress port0 relays the packets of ingress port1, port3. Egress port1 relays the packet of ingress port0 and egress port3 relays the packet of ingress port2. But egress port2 doesn't anything.