Do not forget to 'Cover' your SystemC code with UVMC

Vishal Baskar

Siemens Industry Software, 46871 Bayside Parkway, Fremont, CA 94538|Vishal.Baskar@siemens.com

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

ABBREVIATIONS

SC: SystemC
SV: SystemVerilog
TLM: Transaction Level Modelling
UVMC: Universal Verification Methodology Connect

INTRODUCTION

Coverage is important in the verification cycle. But what if we are faced with a design that involves mixed language like SV and SC with UVMC? Do we collect functional coverage from SC models? Or do we ignore them and use coverage from the SV side in a verification environment?

SystemC is arguably becoming a standard and is widely being adopted by the industry for system-level modeling, hence it makes sense that the verification is extensively performed. However, when it comes to performing **cover** checks in the SystemC model directly, currently there is no standard. Further investigation and research are needed to ensure IEEE-1647 e functional coverage language features are supported in developing such libraries that would eventually become a standard in the future. In this paper we have produced a familiar solution of utilizing functional coverage usage by exporting them from SC to SV each time a sampling event is called on the SC side.

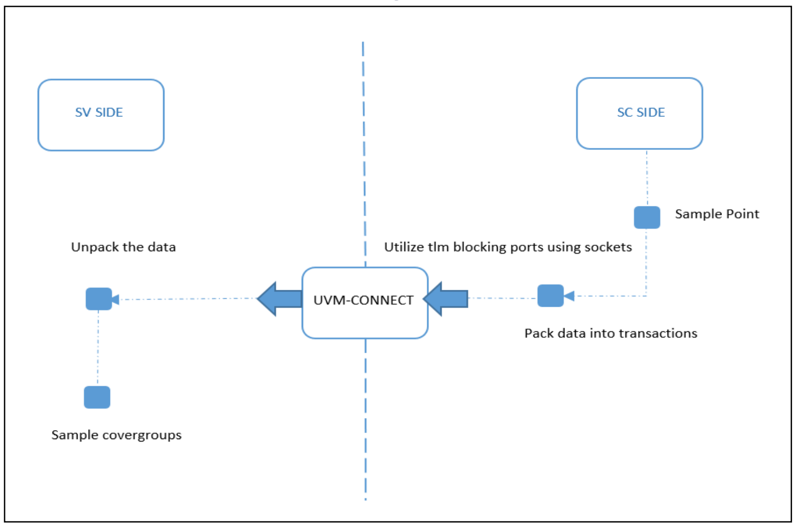
COVERAGE BASICS

A coverage model includes one or more cover groups, which represent a data set to be sampled under certain conditions, typically with a sampling event. For each occurrence of the sampling event, each cover item samples one value and assigns that value into bins that the user has set.

The approach would be to send packets of data from the SC side to the SV side which can be used for sampling functional coverage each time the **sample event** is triggered. This way, verification of highly abstract components and system models can be done using tlm ports and sockets and the data can then be de-serialized on the SV side for functional coverage.

Name	Missing Bins	Total Bins	% Hit	Coverage	Status
/user_pkg/consumer/consumer__1/sv_header	0	10	100.00%	100.00%	
ADD	0	1	100.00%	100.00%	
b1				1	
illegal zero				0	
CMD	0	1	100.00%	100.00%	
all				1	
ignore_bin nop				0	
env.cons.sv_header	0	10	100.00%	100.00%	
VAL0	0	1	100.00%	100.00%	
VAL1	0	1	100.00%	100.00%	
VAL2	0	1	100.00%	100.00%	
VAL3	0	1	100.00%	100.00%	

DIAGRAM



This paper discusses two examples. One would deal with utilizing TLM sockets and generic payloads and the other with transaction packets made from the template specialization class.

EXAMPLE 1

In this example, we have a producer_uvm class, which derives from our generic SC producer. We then pack that class that connects to the SV side by using a simple initiator socket and we send a **generic payload** in the form of a transaction using TLM 2.0.

In the SV side, a simple SV consumer TLM model prints received transactions of type **uvm_tlm_generic_payload**. The class uvm_tlm_generic_payload enables it to be generated in sequences and transported to drivers through sequencers. It consist of many data types like **m_command, m_data[ ]**, m_byte_enable,q etc.

```
void run() {
  tlm_generic_payload gp;
  char unsigned data[8];//Original
  gp.set_data_ptr(data); //Original
  sc_time delay;
  while (num_trans--) {
  .
  .
   gp.set_command(TLM_WRITE_COMMAND);
   }
   out->b_transport(gp,delay); //Sampling point triggered
   ap.write(gp);
```

```
class consumer extends uvm_component;
  uvm_tlm_b_target_socket #(consumer) in;
   uvm_tlm_gp t; //TLM Generic Payload on the receiver side.
  covergroup sv_header with function sample(uvm_tlm_gp t);
        TYPE : coverpoint t .command {
        ignore_bins nop = {0};
        bins all = {[1:5]};  }
       ADDR : coverpoint t.m_address {
        bins b1 = {[0:'hfffff]};
        bins b2 = {[20'hfffff:28'hfffffff]};
     }         ap.write(t);
               sv_header.sample(t);
```

A sampling event is triggered each time the **b_transport** from the SC side sends a TLM transaction and on the SV side, the coverage function is **sampled**.

EXAMPLE 2

In the SC side, we use a **converter class**, which means the transaction definitions are allowed to be completely different. The transaction classes do not have to have the same declaration number, type, and order in both SC and SV. The converter can adapt different transaction definitions and at the same time serialize the data. Functional coverage is applied when the transaction is sent as **TLM transactions** from SC to SV. The producer class contains the input transactions which are **packed** and sends it to the SV side. The producer sends TLM transactions in the form of blocking transport to SV every time. A **sampling** event is reached when the packets are sent to the SV side.

```
template <class T>
 class producer : public sc_module {
  public:
   sc_port<tlm_blocking_transport_if<T> > out;
   }
   SC_HAS_PROCESS(producer);
   void run() {
    .
    packet::cmd_t cmd;
    int unsigned addr;
    vector<char> data;
    .
     t.cmd  = cmd;
     t.addr = addr;
     out->b_transport(t, delay); //Sampling point triggered
    }
```

```
covergroup sv_header with function sample(T t);
    option.per_instance = 1;

    ADD : coverpoint t.addr {
         illegal_bins zero = {0};
         bins b1 = {[0:32'hffffffff]};
         bins b2 = {[32'hffffffff:40'hffffffffff]};
         bins max = {[40'hffffffffff:64'hffffffffffffffff]};
     }

     VAL0 :  coverpoint t.data[0]{bins a0 = {[0:'hff]};}
     VAL1 :  coverpoint t.data[1]{bins a1 = {[0:'hff]};}
     .
     VAL7 :  coverpoint t.data[7]{bins a7=  {[0:'hff]};}
endgroup
  #(delay.get_realtime(1ns,1e-9));
  sv_header.sample(t);//Sample the coverage
```

CONCLUSION AND FUTURE WORK

In preparation for this approach, we attempted the use of tlm sockets and ports present in examples 1 and 2, respectively.

- In example 1, we utilized **the uvm_tlm_generic_payload** transaction which had data members **like address, data, command**, etc. Once we were able to identify the width and bins to cover, we were getting good results as seen in the results sections.
- In the second example, we used a template class <T> to send packets containing addresses and data to the SV side from the SC side. We were able to capitalize on the **do_pack** and **do_unpack** methods of UVM and the coverage results show that all bins in the **coverpoint** were covered.
- Rather than having a class-based transaction passed as an object to a covergroup, utilizing constraint randomization and assertion on tlm_generic_payloads would certainly be more helpful. It would also help improve the overall coverage goal and to identify holes in coverage space. This can also be scaled up to signal processing models that are designed in SC. It would be very beneficial to identify how various signals that are sent in, are sampled and to verify them using coverage, constraint randomization, etc