```
`include "uvm_macros.svh"


package uvm_svid_monitor_package;
   import uvm_pkg::*;
   //---------------------------------------------------------------
   //svid_transmit_packet_configuration
   //---------------------------------------------------------------
class svid_transmit_packet_configuration extends uvm_object;
   `uvm_object_utils( svid_transmit_packet_configuration )

   function new( string name = "" );
      super.new( name );
   endfunction: new
endclass // svid_transmit_packet_configuration


   //---------------------------------------------------------------
   //svid transmit packet class;
   //---------------------------------------------------------------
class svid_transmit_packet extends uvm_sequence_item;

   typedef enum bit [2:0] {IDLE,SET_VID_FAST, SET_VID_SLOW,
SET_VID_DECAY, SET_PS, GET_REG} svid_cmd;
   typedef enum int       {LOOP0, LOOP1} svid_loop;
   typedef enum bit [2:0] {INVALID0, INVALID1, START_FRAME, END_FRAME,
                           INVALID2, INVALID3, INVALID4, INVALID5
                           } frame_limiter;

   //Frame specific parmeters
   frame_limiter start_frame;
   svid_loop addr;
   svid_cmd     command;
   bit [7:0]     payload;
   bit           parity;
   frame_limiter end_frame;

   function new(string name=" ");
      super.new(name);
   endfunction // new

   `uvm_object_utils_begin(svid_transmit_packet)
      `uvm_field_enum( frame_limiter, start_frame,  UVM_ALL_ON )
      `uvm_field_enum( svid_cmd,      command,      UVM_ALL_ON )
      `uvm_field_enum( svid_loop,     addr,         UVM_ALL_ON )
      `uvm_field_int(  payload,                     UVM_ALL_ON )
      `uvm_field_int(  parity,                      UVM_ALL_ON )
      `uvm_field_enum( frame_limiter, end_frame,    UVM_ALL_ON )
   `uvm_object_utils_end



endclass // svid_transmit_packet
```

```
   //-------------------------------------------------------------
   //svid transmission monitor; this monitor retrives the packet
   //from the ingress interface and put it to the analysis port
   //-------------------------------------------------------------
class svid_transmit_packet_monitor extends uvm_monitor;
   `uvm_component_utils(svid_transmit_packet_monitor)

   uvm_analysis_port #(svid_transmit_packet) svid_ap;
   virtual svid_packet_intf ingress_intf; //from wired interface

   function new( string name, uvm_component parent );
      super.new( name, parent );
   endfunction: new

   function void build_phase(uvm_phase phase);
      super.build_phase( phase );
      void'(uvm_resource_db#(virtual svid_packet_intf
)::read_by_name(.scope("ifs"), .name("svid_packet_intf_ingress"),
.val(ingress_intf)));
      svid_ap = new( .name( "svid_ap" ), .parent( this ) );
   endfunction // build_phase

   task run_phase(uvm_phase phase);
      forever begin
       svid_transmit_packet svid_packet;
       @ingress_intf.cb_out;
       if(ingress_intf.cb_out.command != svid_transmit_packet::IDLE)begin
          svid_packet =
svid_transmit_packet::type_id::create(.name("svid_packet"));
          svid_packet.start_frame =
svid_transmit_packet::frame_limiter'(ingress_intf.cb_out.start_frame);
          svid_packet.addr        =
svid_transmit_packet::svid_loop'(ingress_intf.cb_out.addr);
          svid_packet.command     =
svid_transmit_packet::svid_cmd'(ingress_intf.cb_out.command);
          svid_packet.payload     = ingress_intf.cb_out.payload;
          svid_packet.parity      = ingress_intf.cb_out.parity;
          svid_packet.end_frame   =
svid_transmit_packet::frame_limiter'(ingress_intf.cb_out.end_frame);
          svid_ap.write(svid_packet);
       end
      end // forever begin
   endtask // run_phase

endclass // svid_transmit_packet_monitor
   //-------------------------------------------------------------
   //svid agent; instantiates sequencer (NOT), monitor and driver
   //-------------------------------------------------------------
class svid_transmit_packet_agent extends uvm_agent;
   `uvm_component_utils(svid_transmit_packet_agent)

   uvm_analysis_port#(svid_transmit_packet) svid_ap;
   virtual svid_packet_intf ingress_intf; //from DUT
```

```
   svid_transmit_packet_monitor        packet_monitor;

   function new( string name, uvm_component parent );
      super.new( name, parent );
   endfunction: new

   function void build_phase( uvm_phase phase );
      super.build_phase( phase );

      svid_ap = new( .name( "svid_ap" ), .parent( this ) );
      packet_monitor    = svid_transmit_packet_monitor::type_id::create(
.name( "packet_monitor" ), .parent( this ) );

   endfunction // build_phase


   function void connect_phase(uvm_phase phase);
      super.connect_phase( phase );
      packet_monitor.ingress_intf = ingress_intf;
      packet_monitor.svid_ap.connect(svid_ap);
   endfunction // connect_phase

endclass // svid_transmit_packet_agent
   //------------------------------------------------------------
   //svid_functional_coverage_subscriber
   //------------------------------------------------------------
class svid_functional_coverage_subscriber extends
uvm_subscriber#(svid_transmit_packet);
   `uvm_component_utils(svid_functional_coverage_subscriber)

   svid_transmit_packet svid_packet;

   covergroup svid_packet_cg;
      start_frame_cp: coverpoint svid_packet.start_frame;
      addr_cp:        coverpoint svid_packet.addr;
      command_cp:     coverpoint svid_packet.command;
      end_frame_cp:   coverpoint svid_packet.end_frame;
   endgroup // svid_packet_cg

   function new( string name, uvm_component parent );
      super.new( name, parent );
      svid_packet_cg = new;
   endfunction // new

   function void write(svid_transmit_packet t);
      svid_packet = t;
      svid_packet_cg.sample();
   endfunction // write

endclass // svid_functional_coverage_subscriber
   //------------------------------------------------------------
   //   svid_scoreboard_subscriber
   //------------------------------------------------------------
   typedef class svid_packet_scoreboard;
```

```
class svid_packet_scoreboard_subscriber extends
uvm_subscriber#(svid_transmit_packet);
   `uvm_component_utils(svid_packet_scoreboard_subscriber)


  function new( string name, uvm_component parent );
      super.new( name, parent );
  endfunction: new

   function void write( svid_transmit_packet t );
      svid_packet_scoreboard svid_packet_sb;
      $cast( svid_packet_sb, m_parent );
      svid_packet_sb.check_svid_packet( t );
   endfunction: write

endclass: jelly_bean_sb_subscriber
   //---------------------------------------------------------------
   //   svid_scoreboard
   //---------------------------------------------------------------
class svid_packet_scoreboard extends uvm_scoreboard;

   `uvm_component_utils( svid_packet_scoreboard )

   uvm_analysis_export#( svid_transmit_packet )
svid_packet_analysis_export;
   local svid_packet_scoreboard_subscriber svid_packet_sub;

   function new( string name, uvm_component parent );
      super.new( name, parent );
   endfunction: new

   function void build_phase( uvm_phase phase );
      super.build_phase( phase );
      svid_packet_analysis_export = new( .name(
"svid_packet_analysis_export" ), .parent( this ) );
      svid_packet_sub =
svid_packet_scoreboard_subscriber::type_id::create( .name(
"svid_packet_sub" ), .parent( this ) );
   endfunction: build_phase

   function void connect_phase( uvm_phase phase );
      super.connect_phase( phase );
      svid_packet_analysis_export.connect(
svid_packet_sub.analysis_export );
   endfunction: connect_phase

   virtual function void check_svid_packet(svid_transmit_packet t);

      ////////////////////////////////////////////////////////////////
      //Put in the packet checking mechanism here !!!!
      ////////////////////////////////////////////////////////////////

   endfunction
```

```
endclass // svid_packet_scoreboard
   //------------------------------------------------------------
   // svid_packet_checking_environment  -- Putting it all together !!!
   //------------------------------------------------------------
class svid_packet_checking_env extends uvm_env;
   `uvm_component_utils(svid_packet_checking_env)

   svid_transmit_packet_agent          svid_agent; //sequencer, driver
and monitor
   svid_functional_coverage_subscriber svid_fc_sub;
   svid_packet_scoreboard              svid_sb;
   virtual svid_packet_intf ingress_intf;

   function new( string name, uvm_component parent );
      super.new( name, parent );
   endfunction: new

   function void build_phase( uvm_phase phase );
      super.build_phase( phase );
      svid_agent    = svid_transmit_packet_agent::type_id::create( .name(
"svid_agent"  ), .parent( this ) );
      svid_fc_sub   =
svid_functional_coverage_subscriber::type_id::create( .name(
"svid_fc_sub" ), .parent( this ) );
      svid_sb       = svid_packet_scoreboard::type_id::create( .name(
"svid_sb"     ), .parent( this ) );
    endfunction: build_phase

   function void connect_phase( uvm_phase phase );
      super.connect_phase( phase );
      svid_agent.ingress_intf = ingress_intf;
      svid_agent.svid_ap.connect( svid_fc_sub.analysis_export );
      svid_agent.svid_ap.connect( svid_sb.svid_packet_analysis_export );
   endfunction: connect_phase

endclass // svid_packet_checking_env

endpackage // uvm_svid_monitor_package
```