

## Momen Mostafa Mohamed Elzaghawy

### UART\_TX UVM

Design:

```
1  module UART (UART_if.DUT uartif);
2
3      reg [3:0] state = uartif.IDLE;
4      reg [3:0] counter = 0;
5      reg [7:0] DATA_reg;
6      reg PARITY_bit;
7  always @(posedge uartif.clk or negedge uartif.reset) begin
8      if (!uartif.reset) begin
9          state <= uartif.IDLE;
10         uartif.TX_OUT <= 1'b1;
11         uartif.Busy <= 1'b0;
12         counter <= 0;
13     end else begin
14         case (state)
15             uartif.IDLE: begin
16                 uartif.TX_OUT <= 1'b1;
17                 uartif.Busy <= 1'b0;
18                 counter <= 0;
19                 if (uartif.DATA_VALID) begin
20                     DATA_reg <= uartif.P_DATA;
21                     PARITY_bit <= (uartif.PAR_TYP == 0) ? ~^uartif.P_DATA : ^uartif.P_DATA;
22                     state <= uartif.START;
23                     uartif.Busy <= 1'b1;
24                 end
25             end
26             uartif.START: begin
27                 uartif.TX_OUT <= 1'b0;
28                 state <= uartif.DATA;
29                 counter <= 0;
30             end
31             uartif.DATA: begin
32                 uartif.TX_OUT <= DATA_reg[counter];
33                 counter <= counter + 1;
34                 if (counter == 7)
35                     state <= (uartif.PAR_EN) ? uartif.PARITY : uartif.STOP;
36             end
37             uartif.PARITY: begin
38                 uartif.TX_OUT <= PARITY_bit;
39                 state <= uartif.STOP;
40             end
41             uartif.STOP: begin
42                 uartif.TX_OUT <= 1'b1;
43                 state <= uartif.IDLE;
44             end
45         endcase
46     end
47 end
48 endmodule
```

Golden model:

```
1  module UART_golden_model(UART_if.GOLDEN uartif);
2      reg [3:0] counter = 0;
3      reg [7:0] DATA_reg;
4      reg  PARITY_bit;
5      reg [3:0]cs,ns;
6      // cs state
7      always @(posedge uartif.clk , negedge uartif.reset)begin
8          if (!uartif.reset) begin
9              cs <= uartif.IDLE;
10             counter <= 0;
11         end
12         else
13             cs <= ns;
14     end
15     //ns state
16     always @(*)begin
17         case(cs)
18             uartif.IDLE : begin
19                 if (uartif.DATA_VALID) begin
20                     ns = uartif.START;
21                 end
22             end
23             uartif.START : ns = uartif.DATA;
24             uartif.DATA :begin
25                 if (counter == 7)
26                     ns = (uartif.PAR_EN) ? uartif.PARITY : uartif.STOP;
27             end
28             uartif.PARITY : ns = uartif.STOP;
29             uartif.STOP : ns = uartif.IDLE;
30             default : ns = uartif.IDLE;
31         endcase
32     end
33 endmodule
```

```

33 //output
34 always @(posedge uartif.clk , negedge uartif.reset) begin
35     if (!uartif.reset) begin
36         uartif.TX_OUT_ex <= 1'b1;
37         uartif.Busy_ex <= 1'b0;
38         counter <= 0;
39     end
40     else begin
41         case (cs)
42             uartif.IDLE: begin
43                 uartif.TX_OUT_ex <= 1'b1;
44                 uartif.Busy_ex <= 1'b0;
45                 counter <= 0;
46                 if (uartif.DATA_VALID) begin
47                     DATA_reg <= uartif.P_DATA;
48                     PARITY_bit <= (uartif.PAR_TYP == 0) ? ~^uartif.P_DATA : ^uartif.P_DATA;
49                     uartif.Busy_ex <= 1'b1;
50                 end
51             end
52             uartif.START: begin
53                 uartif.TX_OUT_ex <= 1'b0;
54                 counter <= 0;
55             end
56             uartif.DATA: begin
57                 uartif.TX_OUT_ex <= DATA_reg[counter];
58                 counter <= counter + 1;
59             end
60             uartif.PARITY: begin
61                 uartif.TX_OUT_ex <= PARITY_bit;
62             end
63             uartif.STOP: begin
64                 uartif.TX_OUT_ex <= 1'b1;
65             end
66         endcase
67     end
68 end
69 endmodule

```

## SVA:

```
1 module UART_sva (UART_if.DUT uartif);
2
3     always_comb begin
4         if(!uartif.reset)
5             assert_reset: assert final ((uartif.TX_OUT)&&(!uartif.Busy)&&(UART.state==uartif.IDLE)&&(UART.counter==0));
6             cover_reset:cover final ((uartif.TX_OUT)&&(!uartif.Busy)&&(UART.state==uartif.IDLE)&&(UART.counter==0));
7         end
8         property no2;
9             @(posedge uartif.clk) disable iff(!uartif.reset)
10                ((UART.state==uartif.IDLE) && (!uartif.DATA_VALID) )|=>
11                (uartif.TX_OUT == 1) && (!uartif.Busy) && (UART.counter == 1'b0) ;
12        endproperty
13        property no3;
14            @(posedge uartif.clk) disable iff(!uartif.reset)
15                ((UART.state==uartif.IDLE) && (!uartif.DATA_VALID) )|=> (UART.state==uartif.IDLE) ;
16        endproperty
17        property no4;
18            @(posedge uartif.clk) disable iff(!uartif.reset)
19                (UART.state==uartif.IDLE)&&(uartif.DATA_VALID) && (uartif.PAR_TYP==0) |=>
20                (UART.PARITY_bit==^$past(uartif.P_DATA)) && (uartif.Busy) && (UART.DATA_reg==$past(uartif.P_DATA)) &&(UART.state==uartif.START);
21        endproperty
22        property no5;
23            @(posedge uartif.clk) disable iff(!uartif.reset)
24                (UART.state==uartif.IDLE)&&(uartif.DATA_VALID) && (uartif.PAR_TYP==1) |=>
25                (UART.PARITY_bit==^$past(uartif.P_DATA)) && (uartif.Busy) && (UART.DATA_reg==$past(uartif.P_DATA)) &&(UART.state==uartif.START);
26        endproperty
27
28        property no6;
29            @(posedge uartif.clk) disable iff(!uartif.reset)
30                (UART.state==uartif.IDLE)&&(uartif.DATA_VALID) |=> (UART.state==uartif.START);
31        endproperty
32        property no7;
33            @(posedge uartif.clk) disable iff(!uartif.reset)
34                (UART.state==uartif.START) |=>
35                (uartif.TX_OUT==0) && (UART.state==uartif.DATA) && (UART.counter==0);
36        endproperty
37        property no8;
38            @(posedge uartif.clk) disable iff(!uartif.reset)
39                ((UART.state==uartif.DATA) && (UART.counter != 7 )) |=>
40                ((uartif.TX_OUT)==UART.DATA_reg[$past(UART.counter)]) && (UART.counter==$past(UART.counter)+1);
41        endproperty
42        property no9;
43            @(posedge uartif.clk) disable iff(!uartif.reset)
44                ((UART.state==uartif.DATA) && (UART.counter != 7 )) |=> (UART.state==uartif.DATA) ;
45        endproperty
46
47        property no10;
48            @(posedge uartif.clk) disable iff(!uartif.reset)
49                (UART.state==uartif.DATA)&&(UART.counter==7)&&(uartif.PAR_EN==1) |=>
50                (UART.state==uartif.PARITY);
51        endproperty
52        property no11;
53            @(posedge uartif.clk) disable iff(!uartif.reset)
54                (UART.state==uartif.DATA)&&(UART.counter==7)&&(uartif.PAR_EN==0) |=>
55                (UART.state==uartif.STOP);
56        endproperty
57        property no12;
58            @(posedge uartif.clk) disable iff(!uartif.reset)
59                (UART.state==uartif.PARITY) |=>
60                (uartif.TX_OUT==$past(UART.PARITY_bit)) &&(UART.state==uartif.STOP);
61        endproperty
62        property no13;
63            @(posedge uartif.clk) disable iff(!uartif.reset)
64                (UART.state==uartif.STOP) |=>
65                (uartif.TX_OUT==1) &&(UART.state==uartif.IDLE);
66        endproperty
67
68        assert_2:assert property (no2);
69        cover_2:cover property (no2);
70        assert_3:assert property (no3);
71        cover_3:cover property (no3);
72        assert_4:assert property (no4);
73        cover_4:cover property (no4);
74        assert_5:assert property (no5);
75        cover_5:cover property (no5);
76        assert_6:assert property (no6);
77        cover_6:cover property (no6);
78        assert_7:assert property (no7);
79        cover_7:cover property (no7);
80        assert_8:assert property (no8);
81        cover_8:cover property (no8);
82        assert_9:assert property (no9);
83        cover_9:cover property (no9);
84        assert_10:assert property (no10);
85        cover_10:cover property (no10);
86        assert_11:assert property (no11);
87        cover_11:cover property (no11);
88        assert_12:assert property (no12);
89        cover_12:cover property (no12);
90        assert_13:assert property (no13);
91        cover_13:cover property (no13);
92
93    endmodule
```

Interface:

```
1 interface UART_if (clk);
2     input bit clk;
3     logic reset;
4     logic [7:0] P_DATA;
5     logic PAR_EN;
6     logic PAR_TYP;
7     logic DATA_VALID;
8     logic TX_OUT , TX_OUT_ex;
9     logic Busy , Busy_ex;
10
11     parameter IDLE = 0, START = 1, DATA = 2, PARITY = 3, STOP = 4;
12
13     modport DUT ( input clk , reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID ,
14                 | output TX_OUT , Busy);
15
16     modport GOLDEN ( input clk , reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID ,
17                     | output TX_OUT_ex , Busy_ex);
18
19     modport TEST ( output reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID ,
20                  | input clk , TX_OUT , Busy , TX_OUT_ex , Busy_ex);
21
22     modport MONITOR ( output clk , reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID , TX_OUT , Busy , TX_OUT_ex , Busy_ex);
23 endinterface
24
25
```

Top:

```
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import UART_test_pkg::*;
4
5 module UART_top();
6
7     bit clk;
8
9     initial begin
10         forever begin
11             #1 clk = ~clk;
12         end
13     end
14
15     UART_if uartif (clk);
16     UART dut (uartif);
17     UART_golden_model GM (uartif);
18     bind UART UART_sva UART_sva_inst(uartif);
19
20     initial begin
21         uvm_config_db#(virtual UART_if)::set(null, "uvm_test_top", "UART_IF", uartif);
22         run_test("UART_test");
23     end
24
25 endmodule
```

Test:

```
# UART_test.sv
1 package UART_test_pkg;
2 import UART_env_pkg::*;
3 import UART_config_pkg::*;
4 import UART_seq_pkg::*;
5 import uvm_pkg::*;
6 import UART_seq_item_pkg::*;
7 `include "uvm_macros.svh"
8
9 class UART_test extends uvm_test;
10     `uvm_component_utils(UART_test)
11
12     UART_env env;
13     UART_config UART_cfg;
14     virtual UART_if UART_vif;
15     UART_main_seq main_seq;
16     UART_reset_seq reset_seq;
17
18     function new(string name = "UART_test", uvm_component parent = null);
19         super.new(name, parent);
20     endfunction
21
22     function void build_phase(uvm_phase phase);
23         super.build_phase(phase);
24         env = UART_env::type_id::create("env", this);
25         UART_cfg = UART_config::type_id::create("UART_cfg");
26         main_seq = UART_main_seq::type_id::create("main_seq");
27         reset_seq = UART_reset_seq::type_id::create("reset_seq");
28
29         UART_cfg.is_active = UVM_ACTIVE;
30
31         if (!uvm_config_db#(virtual UART_if)::get(this, "", "UART_IF", UART_cfg.UART_vif))
32             `uvm_fatal("build_phase", "test - unable to get the virtual interface");
33
34         uvm_config_db#(UART_config)::set(this, "*", "CFG", UART_cfg);
35     endfunction
36
37     task run_phase(uvm_phase phase);
38         super.run_phase(phase);
39         phase.raise_objection(this);
40
41
42         `uvm_info("run_phase", "reset asserted", UVM_LOW)
43         reset_seq.start(env.agt.sqr);
44         `uvm_info("run_phase", "reset deasserted", UVM_LOW)
45
46
47         `uvm_info("run_phase", "stimulus generation started 1", UVM_LOW)
48         main_seq.start(env.agt.sqr);
49         `uvm_info("run_phase", "stimulus generation ended 1", UVM_LOW)
50
51
52         `uvm_info("run_phase", "reset asserted", UVM_LOW)
53         reset_seq.start(env.agt.sqr);
54         `uvm_info("run_phase", "reset deasserted", UVM_LOW)
55
56         phase.drop_objection(this);
57     endtask
58
59 endclass
60 endpackage
```

Agent:

```
# UART_agent.sv
1  package UART_agent_pkg;
2  import UART_driver_pkg::*;
3  import UART_monitor_pkg::*;
4  import UART_sequencer_pkg::*;
5  import UART_seq_item_pkg::*;
6  import UART_config_pkg::*;
7  import uvm_pkg::*;
8  `include "uvm_macros.svh"
9
10 class UART_agent extends uvm_agent;
11     `uvm_component_utils(UART_agent)
12
13     UART_sequencer sqr;
14     UART_driver drv;
15     UART_monitor mon;
16     UART_config UART_cfg;
17     uvm_analysis_port #(UART_seq_item) agt_ap;
18
19     function new (string name = "UART_agent", uvm_component parent = null);
20         super.new(name, parent);
21     endfunction
22
23     function void build_phase (uvm_phase phase);
24         super.build_phase(phase);
25         if (!uvm_config_db #(UART_config)::get(this, "", "CFG", UART_cfg))
26             `uvm_fatal("build_phase", "test - unable to get the configuration");
27
28         if (UART_cfg.is_active == UVM_ACTIVE) begin
29             sqr = UART_sequencer::type_id::create("sqr", this);
30             drv = UART_driver::type_id::create("drv", this);
31         end
32         mon = UART_monitor::type_id::create("mon", this);
33         agt_ap = new("agt_ap", this);
34     endfunction
35
36     function void connect_phase(uvm_phase phase);
37         mon.UART_vif = UART_cfg.UART_vif;
38         mon.mon_ap.connect(agt_ap);
39         if (UART_cfg.is_active == UVM_ACTIVE) begin
40             drv.UART_vif = UART_cfg.UART_vif;
41             drv.seq_item_port.connect(sqr.seq_item_export);
42         end
43     endfunction
44 endclass
endpackage
```

Env:

```
1  package UART_env_pkg;
2  import UART_agent_pkg::*;
3  import UART_scoreboard_pkg::*;
4  import UART_coverage_pkg::*;
5  import uvm_pkg::*;
6  `include "uvm_macros.svh"
7
8  class UART_env extends uvm_env;
9      `uvm_component_utils(UART_env)
10
11     UART_agent agt;
12     UART_scoreboard sb;
13     UART_coverage cov;
14
15     function new(string name = "UART_env", uvm_component parent = null);
16         super.new(name, parent);
17     endfunction
18
19     function void build_phase(uvm_phase phase);
20         super.build_phase(phase);
21         agt = UART_agent::type_id::create("agt", this);
22         sb = UART_scoreboard::type_id::create("sb", this);
23         cov = UART_coverage::type_id::create("cov", this);
24     endfunction
25
26     function void connect_phase(uvm_phase phase);
27         agt.agt_ap.connect(sb.sb_export);
28         agt.agt_ap.connect(cov.cov_export);
29     endfunction
30 endclass
31 endpackage
```

Config:

```
UART_config.sv
1  package UART_config_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4
5  class UART_config extends uvm_object;
6      `uvm_object_utils(UART_config)
7
8      virtual UART_if UART_vif;
9      uvm_active_passive_enum is_active;
10
11     function new(string name = "UART_config");
12         super.new(name);
13     endfunction
14 endclass
15 endpackage
```



Sequence item:

```
1  package UART_seq_item_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4
5  parameter IDLE = 0, START = 1, DATA = 2, PARITY = 3, STOP = 4;
6
7  class UART_seq_item extends uvm_sequence_item;
8      `uvm_object_utils(UART_seq_item)
9
10     bit clk;
11     rand logic reset;
12     rand logic [7:0] P_DATA;
13     rand logic PAR_EN;
14     rand logic PAR_TYP;
15     rand logic DATA_VALID;
16     rand int pattern_type;
17
18     logic TX_OUT , TX_OUT_ex;
19     logic Busy , Busy_ex;
20
21     function new(string name = "UART_seq_item");
22         super.new(name);
23     endfunction
24
25     function string convert2string();
26         return $sformatf("%s reset=%b P_DATA=%h PAR_EN=%b PAR_TYP=%b DATA_VALID=%b",
27             super.convert2string(), reset, P_DATA, PAR_EN, PAR_TYP, DATA_VALID);
28     endfunction
29
30     function string convert2string_stimulus();
31         return $sformatf("reset=%b P_DATA=%h PAR_EN=%b PAR_TYP=%b DATA_VALID=%b",
32             reset, P_DATA, PAR_EN, PAR_TYP, DATA_VALID);
33     endfunction
34
35     constraint reset_con {reset dist {0 := 3 , 1 := 97};}
36     constraint PAR_TYP_con {PAR_TYP dist {0 := 50 , 1 := 50};}
37     constraint PAR_EN_con {PAR_EN dist{0 := 75 , 1 := 25};}
38     constraint DATA_VALID_con {DATA_VALID dist {0 := 8 , 1 := 92};} ;
39     constraint P_DATA_LSB_con {P_DATA[0] dist {0 := 20 , 1:=80};}
40     constraint pattern_type_con { pattern_type dist {0 := 96 , 1 := 4 };}
41     constraint P_DATA_con { if (pattern_type) P_DATA inside{8'hFF , 8'h00 , 8'hAA};}
42
43 endclass
44 endpackage
```

Sequence:

```
# UART_seq.sv
1  package UART_seq_pkg;
2  import uvm_pkg::*;
3  import UART_seq_item_pkg::*;
4  `include "uvm_macros.svh"
5
6  class UART_reset_seq extends uvm_sequence #(UART_seq_item);
7      `uvm_object_utils(UART_reset_seq)
8      UART_seq_item seq_item;
9
10     function new(string name = "UART_reset_seq");
11         super.new(name);
12     endfunction
13
14     task body;
15         seq_item = UART_seq_item::type_id::create("seq_item");
16         start_item(seq_item);
17         seq_item.reset = 0;
18         seq_item.P_DATA = 0;
19         seq_item.PAR_EN = 0;
20         seq_item.PAR_TYP = 0;
21         seq_item.DATA_VALID = 0;
22         finish_item(seq_item);
23     endtask
24 endclass
25
26 class UART_main_seq extends uvm_sequence #(UART_seq_item);
27     `uvm_object_utils(UART_main_seq)
28     UART_seq_item seq_item;
29
30     function new(string name = "UART_main_seq");
31         super.new(name);
32     endfunction
33
34     task body;
35         seq_item = UART_seq_item::type_id::create("seq_item");
36         repeat (1000) begin
37             start_item(seq_item);
38             assert(seq_item.randomize());
39             finish_item(seq_item);
40         end
41     endtask
42 endclass
43 endpackage
44
```

Sequencer:

```
UART_sequencer.sv
1  package UART_sequencer_pkg;
2  import uvm_pkg::*;
3  import UART_seq_item_pkg::*;
4  `include "uvm_macros.svh"
5
6  class UART_sequencer extends uvm_sequencer #(UART_seq_item);
7      `uvm_component_utils(UART_sequencer);
8
9      function new (string name = "UART_sequencer", uvm_component parent = null);
10         super.new(name, parent);
11     endfunction
12 endclass
13
14 endpackage

```

Driver:

```
1 package UART_driver_pkg;
2 import uvm_pkg::*;
3 import UART_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class UART_driver extends uvm_driver #(UART_seq_item);
7     `uvm_component_utils(UART_driver)
8
9     virtual UART_if UART_vif;
10    UART_seq_item stim_seq_item;
11
12    function new(string name = "UART_driver", uvm_component parent = null);
13        super.new(name, parent);
14    endfunction
15
16    task run_phase(uvm_phase phase);
17        super.run_phase(phase);
18        forever begin
19            stim_seq_item = UART_seq_item::type_id::create("stim_seq_item");
20            seq_item_port.get_next_item(stim_seq_item);
21
22            UART_vif.reset = stim_seq_item.reset;
23            UART_vif.P_DATA = stim_seq_item.P_DATA;
24            UART_vif.PAR_EN = stim_seq_item.PAR_EN;
25            UART_vif.PAR_TYP = stim_seq_item.PAR_TYP;
26            UART_vif.DATA_VALID = stim_seq_item.DATA_VALID;
27
28            @(negedge UART_vif.clk);
29            seq_item_port.item_done();
30            `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
31        end
32    endtask
33 endclass
34 endpackage
35
```

Coverage:

```
1
2 package UART_coverage_pkg;
3 import UART_seq_item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class UART_coverage extends uvm_component;
8     `uvm_component_utils(UART_coverage)
9
10     uvm_analysis_export #(UART_seq_item) cov_export;
11     uvm_tlm_analysis_fifo #(UART_seq_item) cov_fifo;
12     UART_seq_item seq_item_cov;
13
14     covergroup cvr_gp ;
15     reset_cp : coverpoint seq_item_cov.reset {
16         bins zero = {0} ;
17         bins one = {1} ;}
18     PAR_TYP_cp : coverpoint seq_item_cov.PAR_TYP {
19         bins zero = {0} ;
20         bins one = {1} ;}
21     PAR_EN_cp : coverpoint seq_item_cov.PAR_EN {
22         bins zero = {0} ;
23         bins one = {1} ;}
24     P_DATA_cp : coverpoint seq_item_cov.P_DATA {
25         bins all_values = {[0:255]};}
26     DATA_VALID_cp : coverpoint seq_item_cov.DATA_VALID {
27         bins zero = {0} ;
28         bins one = {1} ;}
29     TX_OUT_cp : coverpoint seq_item_cov.TX_OUT {
30         bins zero = {0} ;
31         bins one = {1} ;}
32     Busy_cp : coverpoint seq_item_cov.Busy {
33         bins zero = {0} ;
34         bins one = {1} ;}
35     PAR_EN_TYPE_cross : cross PAR_EN_cp , PAR_TYP_cp ;
36     P_DATA_DATA_VALID_cross : cross P_DATA_cp , DATA_VALID_cp ;
37     TX_OUT_Busy_cross : cross TX_OUT_cp , Busy_cp ;
38
39     endgroup
40
41     function new(string name = "UART_coverage", uvm_component parent = null);
42         super.new(name, parent);
43         cvr_gp = new();
44     endfunction
45
46     function void build_phase(uvm_phase phase);
47         super.build_phase(phase);
48         cov_export = new("cov_export", this);
49         cov_fifo = new("cov_fifo", this);
50     endfunction
51
52     function void connect_phase(uvm_phase phase);
53         super.connect_phase(phase);
54         cov_export.connect(cov_fifo.analysis_export);
55     endfunction
56
57     task run_phase(uvm_phase phase);
58         super.run_phase(phase);
59         forever begin
60             cov_fifo.get(seq_item_cov);
61             cvr_gp.sample();
62         end
63     endtask
64 endclass
65 endpackage
66
```

## Scoreboard:

```
1  package UART_scoreboard_pkg;
2  import UART_seq_item_pkg::*;
3  import uvm_pkg::*;
4  `include "uvm_macros.svh"
5
6  class UART_scoreboard extends uvm_scoreboard;
7      `uvm_component_utils(UART_scoreboard)
8
9      uvm_analysis_export #(UART_seq_item) sb_export;
10     uvm_tlm_analysis_fifo #(UART_seq_item) sb_fifo;
11     UART_seq_item seq_item_sb;
12     int error_count = 0;
13     int correct_count = 0;
14
15     function new(string name = "UART_scoreboard", uvm_component parent = null);
16         super.new(name, parent);
17     endfunction
18
19     function void build_phase(uvm_phase phase);
20         super.build_phase(phase);
21         sb_export = new("sb_export", this);
22         sb_fifo = new("sb_fifo", this);
23     endfunction
24
25     function void connect_phase(uvm_phase phase);
26         super.connect_phase(phase);
27         sb_export.connect(sb_fifo.analysis_export);
28     endfunction
29
30     task run_phase(uvm_phase phase);
31         super.run_phase(phase);
32         forever begin
33             sb_fifo.get(seq_item_sb);
34             if ((seq_item_sb.TX_OUT != seq_item_sb.TX_OUT_ex) && [seq_item_sb.Busy != seq_item_sb.Busy_ex]) begin
35                 `uvm_error("run_phase", $sformatf("comparison failed while ref = %0d", seq_item_sb.TX_OUT_ex))
36                 `uvm_error("run_phase", $sformatf("comparison failed while ref = %0d", seq_item_sb.Busy_ex))
37
38                 error_count++;
39             end
40             else begin
41                 correct_count++;
42             end
43         end
44     endtask
45
46     function void report_phase(uvm_phase phase);
47         super.report_phase(phase);
48         `uvm_info("report_phase", $sformatf("correct = %0d", correct_count), UVM_MEDIUM)
49         `uvm_info("report_phase", $sformatf("error = %0d", error_count), UVM_MEDIUM)
50     endfunction
51 endclass
52 endpackage
```

Monitor:

```
1 package UART_monitor_pkg;
2 import uvm_pkg::*;
3 import UART_seq_item_pkg::*;
4 `include "uvm_macros.svh"
5
6 class UART_monitor extends uvm_monitor;
7     `uvm_component_utils(UART_monitor)
8
9     virtual UART_if UART_vif;
10     UART_seq_item rsp_seq_item;
11     uvm_analysis_port #(UART_seq_item) mon_ap;
12
13     function new (string name = "UART_monitor", uvm_component parent = null);
14         super.new(name, parent);
15     endfunction
16
17     function void build_phase (uvm_phase phase);
18         super.build_phase(phase);
19         mon_ap = new("mon_ap", this);
20     endfunction
21
22     task run_phase (uvm_phase phase);
23         super.run_phase(phase);
24         forever begin
25             rsp_seq_item = UART_seq_item::type_id::create("rsp_seq_item");
26             @(posedge UART_vif.clk);
27
28             rsp_seq_item.reset = UART_vif.reset;
29             rsp_seq_item.P_DATA = UART_vif.P_DATA;
30             rsp_seq_item.PAR_EN = UART_vif.PAR_EN;
31             rsp_seq_item.PAR_TYP = UART_vif.PAR_TYP;
32             rsp_seq_item.DATA_VALID = UART_vif.DATA_VALID;
33
34             mon_ap.write(rsp_seq_item);
35             `uvm_info("run_phase", rsp_seq_item.convert2string_stimulus(), UVM_HIGH)
36         end
37     endtask
38 endclass
39
40 endpackage
```

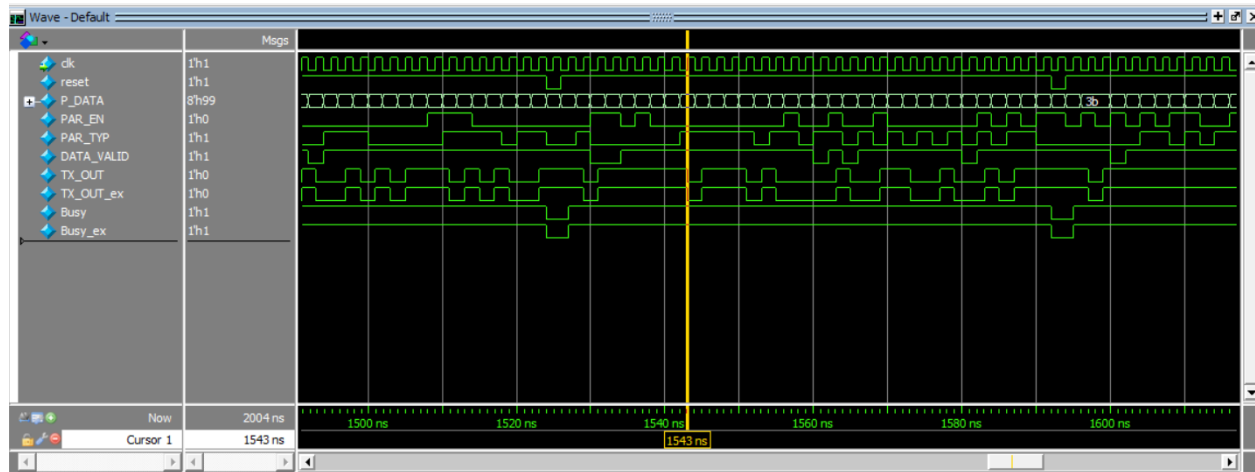
Do file:

```
1 vlib work
2 vlog -f src_files_UART.list
3 vsim -voptargs=+acc work.UART_top -classdebug -uvmcontrol=all
4 add wave /UART_top/uartif/*
5 run -all
```

Src\_files.list:

```
src_files_UART.list
1 UART_if.sv
2 UART.sv
3 UART_golden_model.sv
4 UART_config.sv
5 UART_seq_item.sv
6 UART_seq.sv
7 UART_sequencer.sv
8 UART_monitor.sv
9 UART_driver.sv
10 UART_scoreboard.sv
11 UART_coverage.sv
12 UART_agent.sv
13 UART_env.sv
14 UART_test.sv
15 UART_top.sv
16 |
```

## Waveform and transcript:



```

UVM_INFO UARI_test.sv(44) @ 2: uvm_test_top [run_phase] reset deasserted
UVM_INFO UARI_test.sv(47) @ 2: uvm_test_top [run_phase] stimulus generation started 1
UVM_INFO UARI_test.sv(49) @ 2002: uvm_test_top [run_phase] stimulus generation ended 1
UVM_INFO UARI_test.sv(52) @ 2002: uvm_test_top [run_phase] reset asserted
UVM_INFO UARI_test.sv(54) @ 2004: uvm_test_top [run_phase] reset deasserted
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 2004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO UARI_scoreboard.sv(48) @ 2004: uvm_test_top.env.sb [report_phase] correct = 1002
UVM_INFO UARI_scoreboard.sv(49) @ 2004: uvm_test_top.env.sb [report_phase] error = 0

```

--- UVM Report Summary ---

\*\* Report counts by severity

UVM\_INFO : 12

UVM\_WARNING : 0

UVM\_ERROR : 0

UVM\_FATAL : 0

\*\* Report counts by id

[Questa UVM] 2

[RNISTI] 1

[TEST\_DONE] 1

[report\_phase] 2

[run\_phase] 6

\*\* Note: \$finish : C:/questasim64\_2024.1/win64/./verilog\_src/uvm-1.1d/src/base/uvm\_root.svh(430)

Time: 2004 ns Iteration: 61 Instance: /UARI\_top

Break in Task uvm\_pkg/uvm\_root::run\_test at C:/questasim64\_2024.1/win64/./verilog\_src/uvm-1.1d/src/base/uvm\_root.svh line 430