

```

1  class transaction;
2
3  typedef enum bit [1:0] {write = 2'b00 , read = 2'b01} oper_type;
4
5  randc oper_type oper;
6
7  bit rx;
8
9  rand bit [7:0] dintx;
10
11 bit send;
12 bit tx;
13
14 bit [7:0] doutrx;
15 bit donetx;
16 bit donerx;
17
18
19
20 function void display (input string tag);
21     $display("[%0s] : oper : %0s send : %0b TX_DATA : %b RX_IN : %0b TX_OUT : %0b RX_OUT
        : %b DONE_TX : %0b DONE_RX : %0b", tag, oper.name(), send, dintx, rx, tx, doutrx,
        donetx, donerx);
22 endfunction
23
24 function transaction copy();
25     copy = new();
26     copy.rx = this.rx;
27     copy.dintx = this.dintx;
28     copy.send = this.send;
29     copy.tx = this.tx;
30     copy.doutrx = this.doutrx;
31     copy.donetx = this.donetx;
32     copy.donerx = this.donerx;
33     copy.oper = this.oper;
34 endfunction
35
36 endclass
37
38
39 /*
40 module tb;
41     transaction tr;
42
43
44     initial begin
45         tr = new();
46         tr.display("TOP");
47
48     end
49
50
51 endmodule
52
53 */
54
55
56 class generator;
57
58     transaction tr;
59
60     mailbox #(transaction) mbx;
61
62     event done;
63
64     int count = 0;
65
66     event drvnnext;
67     event sconext;

```

```

68
69
70     function new(mailbox #(transaction) mbx);
71         this.mbx = mbx;
72         tr = new();
73     endfunction
74
75
76     task run();
77
78         repeat(count) begin
79             assert(tr.randomize) else $error("[GEN] :Randomization Failed");
80             mbx.put(tr.copy);
81             tr.display("GEN");
82             @(drvnxt);
83             @(sconxt);
84         end
85
86         -> done;
87     endtask
88
89
90 endclass
91
92 //////////////////////////////////////////////////
93
94 class driver;
95
96
97     virtual uart_if vif;
98
99     transaction tr;
100
101     mailbox #(transaction) mbx;
102
103     mailbox #(bit [7:0]) mbxds;
104
105
106     event drvnxt;
107
108     bit [7:0] din;
109
110
111     bit wr = 0; ///random operation read / write
112     bit [7:0] datarx; ///data rcvd during read
113
114
115
116
117
118     function new(mailbox #(bit [7:0]) mbxds, mailbox #(transaction) mbx);
119         this.mbx = mbx;
120         this.mbxds = mbxds;
121     endfunction
122
123
124
125     task reset();
126         vif.rst <= 1'b1;
127         vif.dintx <= 0;
128         vif.send <= 0;
129         vif.rx <= 1'b1;
130         vif.doutrx <= 0;
131         vif.tx <= 1'b1;
132         vif.donerx <= 0;
133         vif.donetx <= 0;
134         repeat(5) @(posedge vif.uclktx);
135         vif.rst <= 1'b0;
136         @(posedge vif.uclktx);

```

```

137     $display("[DRV] : RESET DONE");
138 endtask
139
140
141
142 task run();
143
144     forever begin
145         mbx.get(tr);
146
147         if(tr.oper == 2'b00)  ////data transmission
148             begin
149                 //
150                 @(posedge vif.uclktx);
151                 vif.rst <= 1'b0;
152                 vif.send <= 1'b1;  ///start data sending op
153                 vif.rx <= 1'b1;
154                 vif.dintx = tr.dintx;
155                 @(posedge vif.uclktx);
156                 vif.send <= 1'b0;
157                 ////wait for completion
158                 //repeat(9) @(posedge vif.uclktx);
159                 mbxds.put(tr.dintx);
160                 $display("[DRV]: Data Sent : %0d", tr.dintx);
161                 wait(vif.donetx == 1'b1);
162                 ->drvnext;
163             end
164
165         else if (tr.oper == 2'b01)
166             begin
167
168                 @(posedge vif.uclkrx);
169                 vif.rst <= 1'b0;
170                 vif.rx <= 1'b0;
171                 vif.send <= 1'b0;
172                 @(posedge vif.uclkrx);
173
174                 for(int i=0; i<=7; i++)
175                     begin
176                         @(posedge vif.uclkrx);
177                         vif.rx <= $urandom;
178                         datarx[i] = vif.rx;
179                     end
180
181
182                 mbxds.put(datarx);
183
184                 $display("[DRV]: Data RCVD : %0d", datarx);
185                 wait(vif.donerx == 1'b1);
186                 vif.rx <= 1'b1;
187                 ->drvnext;
188
189
190             end
191
192
193     end
194 endtask
195
196 endclass
197
198
199
200 /*
201
202 module tb;
203     generator gen;
204     driver drv;
205     event next;

```

```

206     event done;
207     mailbox #(transaction) mbx;
208     mailbox #(bit [7:0]) mbxt;
209
210     uart_if vif();
211
212     uart_top #(1000000, 9600) dut
213     (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);
214
215
216     initial begin
217         vif.clk <= 0;
218     end
219
220     always #10 vif.clk <= ~vif.clk;
221
222
223
224     initial begin
225         mbx = new();
226         mbxt = new();
227         gen = new(mbx);
228         drv = new(mbxt,mbx);
229         gen.count = 20;
230         drv.vif = vif;
231
232         gen.drvnext = next;
233
234         drv.drvnext = next;
235     end
236
237     initial begin
238
239         fork
240             gen.run();
241             drv.run();
242         join_none
243         wait(gen.done.triggered);
244         $finish();
245     end
246
247     initial begin
248         $dumpfile("dump.vcd");
249         $dumpvars;
250     end
251
252
253
254     assign vif.uclktx = dut.utx.uclk;
255     assign vif.uclkrx = dut.rtx.uclk;
256
257
258
259     endmodule
260
261     */
262
263     class monitor;
264
265         transaction tr;
266
267         mailbox #(bit [7:0]) mbx;
268
269         bit [7:0] srx; //send
270         bit [7:0] rrx; //recv
271
272
273

```

```

274     virtual uart_if vif;
275
276
277     function new(mailbox #(bit [7:0]) mbx);
278         this.mbx = mbx;
279     endfunction
280
281     task run();
282
283         forever begin
284
285             @(posedge vif.uclktx);
286             if ( (vif.send== 1'b1) && (vif.rx == 1'b1) )
287                 begin
288
289                     @(posedge vif.uclktx); ////start collecting tx data from next clock
290                     tick
291
292                     for(int i = 0; i<= 7; i++)
293                         begin
294                             @(posedge vif.uclktx);
295                             srx[i] = vif.tx;
296
297                         end
298
299                         $display("[MON] : DATA SEND on UART TX %0d", srx);
300
301                         ///////////////wait for done tx before proceeding next
302                         transaction
303                         @(posedge vif.uclktx); //
304                         mbx.put(srx);
305
306                         end
307
308                     else if ((vif.rx == 1'b0) && (vif.send == 1'b0) )
309                         begin
310                             wait(vif.donerx == 1);
311                             rrx = vif.doutrx;
312                             $display("[MON] : DATA RCVD RX %0d", rrx);
313                             @(posedge vif.uclktx);
314                             mbx.put(rrx);
315                         end
316
317                     end
318
319                 endtask
320
321             endclass
322
323             ////////////////////////////////////////////
324
325             /*
326             module tb;
327                 generator gen;
328                 driver drv;
329                 monitor mon;
330
331                 event sconext;
332                 event drvnnext;
333
334                 event done;
335
336                 mailbox #(transaction) mbx;
337                 mailbox #(bit [7:0]) mbxds;
338                 mailbox #(bit [7:0]) mbxms;
339
340                 uart_if vif();
341
342                 uart_top #(1000000, 9600) dut

```

```

341 (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx,vif.donerx);
342
343
344     initial begin
345         vif.clk <= 0;
346     end
347
348     always #10 vif.clk <= ~vif.clk;
349
350
351
352     initial begin
353         mbx = new();
354         mbxds = new();
355         mbxms = new();
356
357
358         gen = new(mbx);
359         drv = new(mbxds,mbx);
360         mon= new(mbxms);
361
362         gen.count  = 10;
363         drv.vif = vif;
364         mon.vif = vif;
365
366         gen.drvnex = drvnex;
367
368         drv.drvnex = drvnex;
369
370         gen.sconex = sconex;
371
372         mon.sconex = sconex;
373     end
374
375     initial begin
376
377         fork
378             gen.run();
379             drv.run();
380             mon.run();
381         join_none
382         wait(gen.done.triggered);
383         $finish();
384     end
385
386     initial begin
387         $dumpfile("dump.vcd");
388         $dumpvars;
389     end
390
391
392
393     assign vif.uclktx = dut.utx.uclk;
394     assign vif.uclkrx = dut.rtx.uclk;
395
396
397
398 endmodule
399 */
400
401 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
402
403
404
405 class scoreboard;
406     mailbox #(bit [7:0]) mbxds, mbxms;
407
408     bit [7:0] ds;

```

```

409     bit [7:0] ms;
410
411     event sconext;
412
413     function new(mailbox #(bit [7:0]) mbxds, mailbox #(bit [7:0]) mbxms);
414         this.mbxds = mbxds;
415         this.mbxms = mbxms;
416     endfunction
417
418     task run();
419         forever begin
420
421             mbxds.get(ds);
422             mbxms.get(ms);
423
424             $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
425             if(ds == ms)
426                 $display("DATA MATCHED");
427             else
428                 $display("DATA MISMATCHED");
429
430             ->sconext;
431         end
432     endtask
433
434
435 endclass
436
437 //////////////////////////////////////////////////
438
439 class environment;
440
441     generator gen;
442     driver drv;
443     monitor mon;
444     scoreboard sco;
445
446
447
448     event nextgd; ///gen -> drv
449
450     event nextgs; /// gen -> sco
451
452     mailbox #(transaction) mbxgd; ///gen - drv
453
454     mailbox #(bit [7:0]) mbxds; /// drv - sco
455
456
457     mailbox #(bit [7:0]) mbxms; /// mon - sco
458
459     virtual uart_if vif;
460
461
462     function new(virtual uart_if vif);
463
464         mbxgd = new();
465         mbxms = new();
466         mbxds = new();
467
468         gen = new(mbxgd);
469         drv = new(mbxds,mbxgd);
470
471
472
473         mon = new(mbxms);
474         sco = new(mbxds, mbxms);
475
476         this.vif = vif;
477         drv.vif = this.vif;

```

```

478     mon.vif = this.vif;
479
480     gen.sconext = nexttgs;
481     sco.sconext = nexttgs;
482
483     gen.drwnext = nexttgd;
484     drv.drwnext = nexttgd;
485
486 endfunction
487
488 task pre_test();
489     drv.reset();
490 endtask
491
492 task test();
493 fork
494     gen.run();
495     drv.run();
496     mon.run();
497     sco.run();
498 join_any
499 endtask
500
501 task post_test();
502     wait(gen.done.triggered);
503     $finish();
504 endtask
505
506 task run();
507     pre_test();
508     test();
509     post_test();
510 endtask
511
512
513
514 endclass
515
516 //////////////////////////////////////////
517
518
519 module tb;
520
521     uart_if vif();
522
523     uart_top #(1000000, 9600) dut
524         (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);
525
526
527     initial begin
528         vif.clk <= 0;
529     end
530
531     always #10 vif.clk <= ~vif.clk;
532
533     environment env;
534
535
536
537     initial begin
538         env = new(vif);
539         env.gen.count = 5;
540         env.run();
541     end
542
543
544
545     initial begin
546         $dumpfile("dump.vcd");

```



```
546         $dumpvars;
547     end
548
549     assign vif.uclktx = dut.utx.uclk;
550     assign vif.uclkrx = dut.rtx.uclk;
551
552     endmodule
553
554
555
556     //////////////////////////////////////
557
558
```