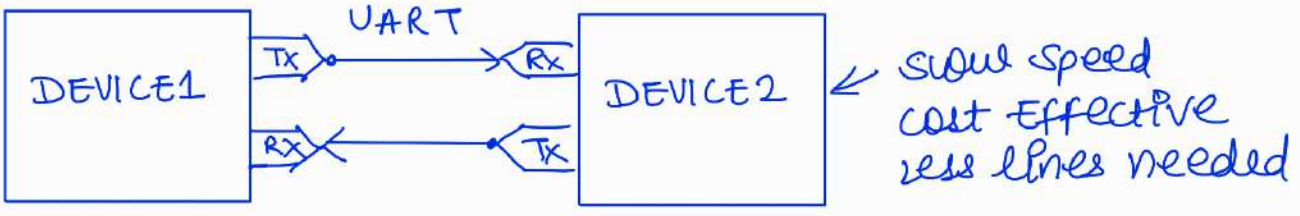
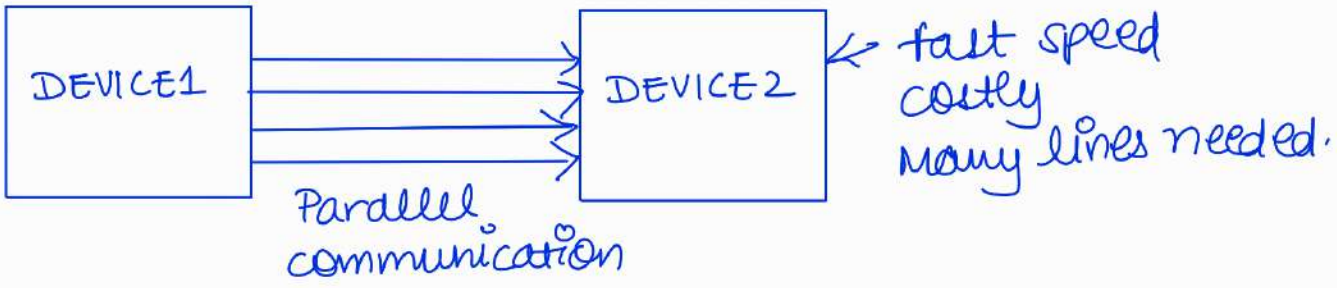
THEORETICAL BACKGROUND

→ UART {Universal Asynchronous Receiver Transmitter} protocol :

• BASICS OF UART → Universal Asynchronous Receiver/Transmitter.

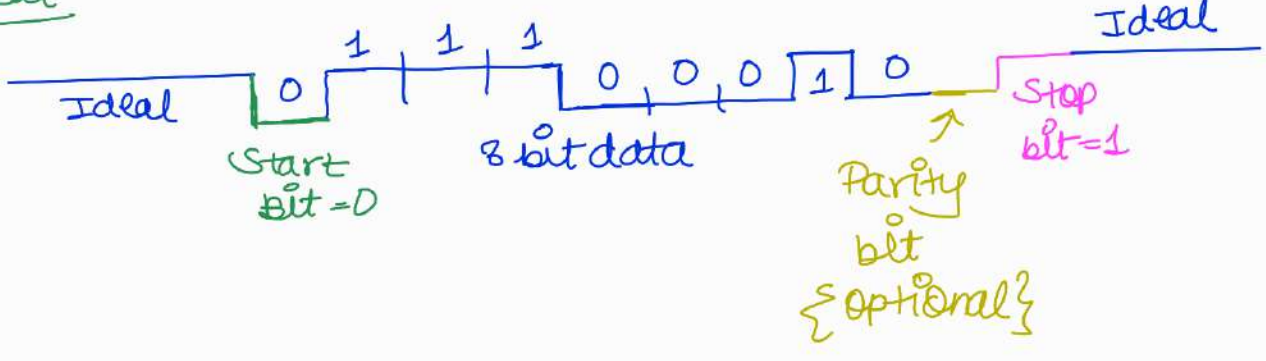
- UART is used for serial communication
- It is two wire communication protocol:
  - → one wire is for transmission
  - → second wire is for reception
- Data format & transmission speeds are configurable.
- Here, clock is not given for synchronisation
- UART transmission speed is slower compared to parallel communication but it needs less bus interface for communication.



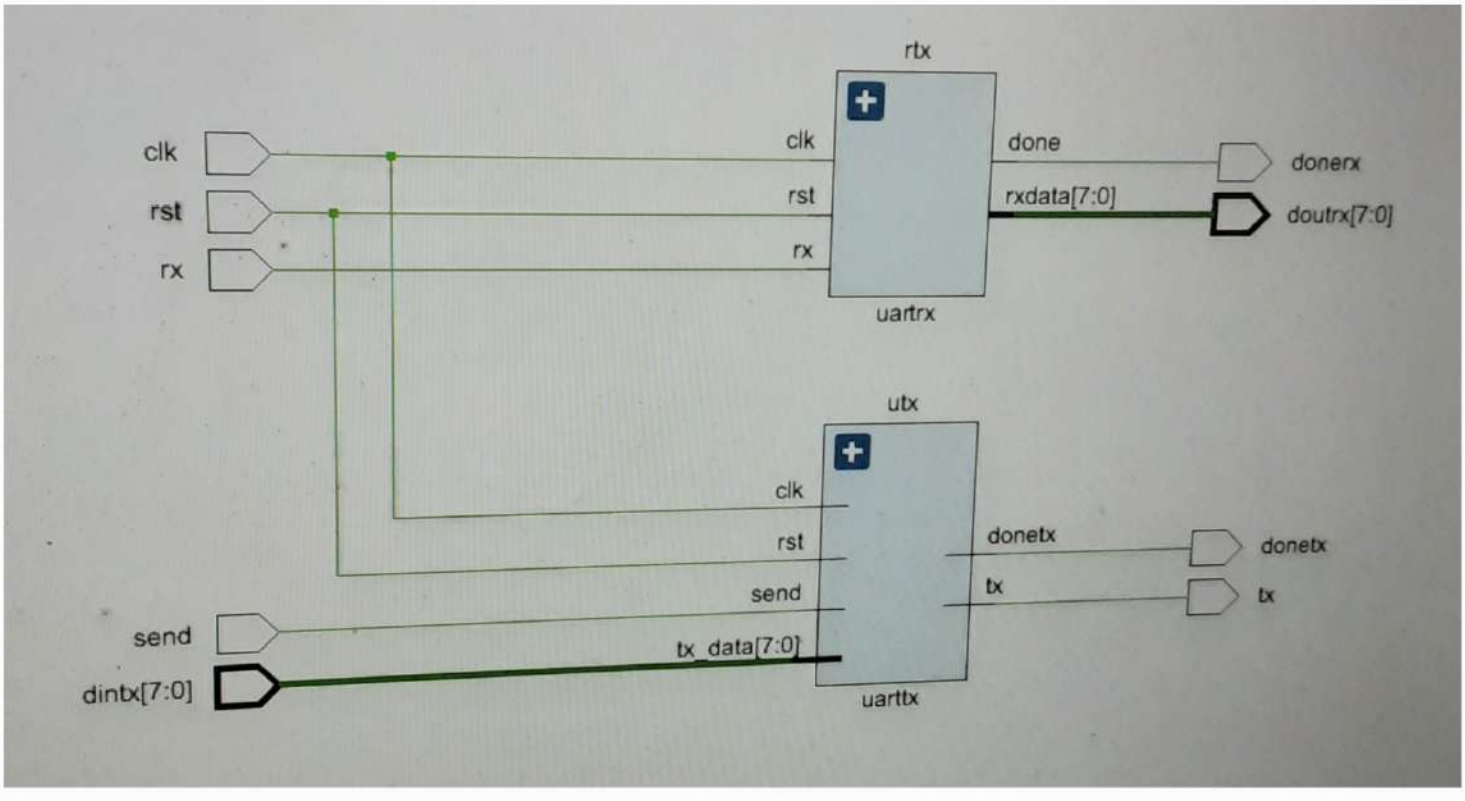


• Data format of UART :-

→ Ideal



- **Advantages of UART Protocol:**
  - → less physical interfacing.
  - → simple to configure
  - → Data size is configurable
  - → Speed is configurable.
  - → Full Duplex mode can be configure by two wires.
  - → Error identification Mechanism {one bit error can be identified}.
- **Disadvantages of UART Protocol:**
  - Low speed of communication
  - Start & stop bit required
  - Asynchronous communication
  - Redundant bits are present. {start bit + stop bits + parity bit}
  - **Schematic Design of UART :-**



→ **INPUT PINS OF OUR DESIGN :-**

→ Here clk & rst are global signal

→ rx pin:- whenever peripheral wish to communicate data to device uartrx it will sent data serially on rx pin. Here rx pin is input to our system

→ whenever user have a new data which it wants to communicate to the peripheral in that case first it'll make "send" high. So, whenever send is high master {user} have new data and at the same time we'll also be specify the data at port dintx [7:0].

→ OUTPUT PINS OF OUR DESIGN:

→ donetx and donerx: whenever we complete reception of a data, "donerx" will become high. similarly, when we complete the data transmission "donetx" will be high. These two are status pins.

→ The data that we'll communicate to a peripheral serially will be on tx pin

→ Data will be received on rx pin & will be given out on "doutrx [7:0]".

• SUBSYSTEMS OF OUR DESIGN:

→ our system consists of two subsystems one

is uartx { which handles the entire process of transmitting data to a peripheral } and other subsystem handles the entire process of receiving data from peripheral V.i.Z uartrx.

- CODE OF DESIGN :-

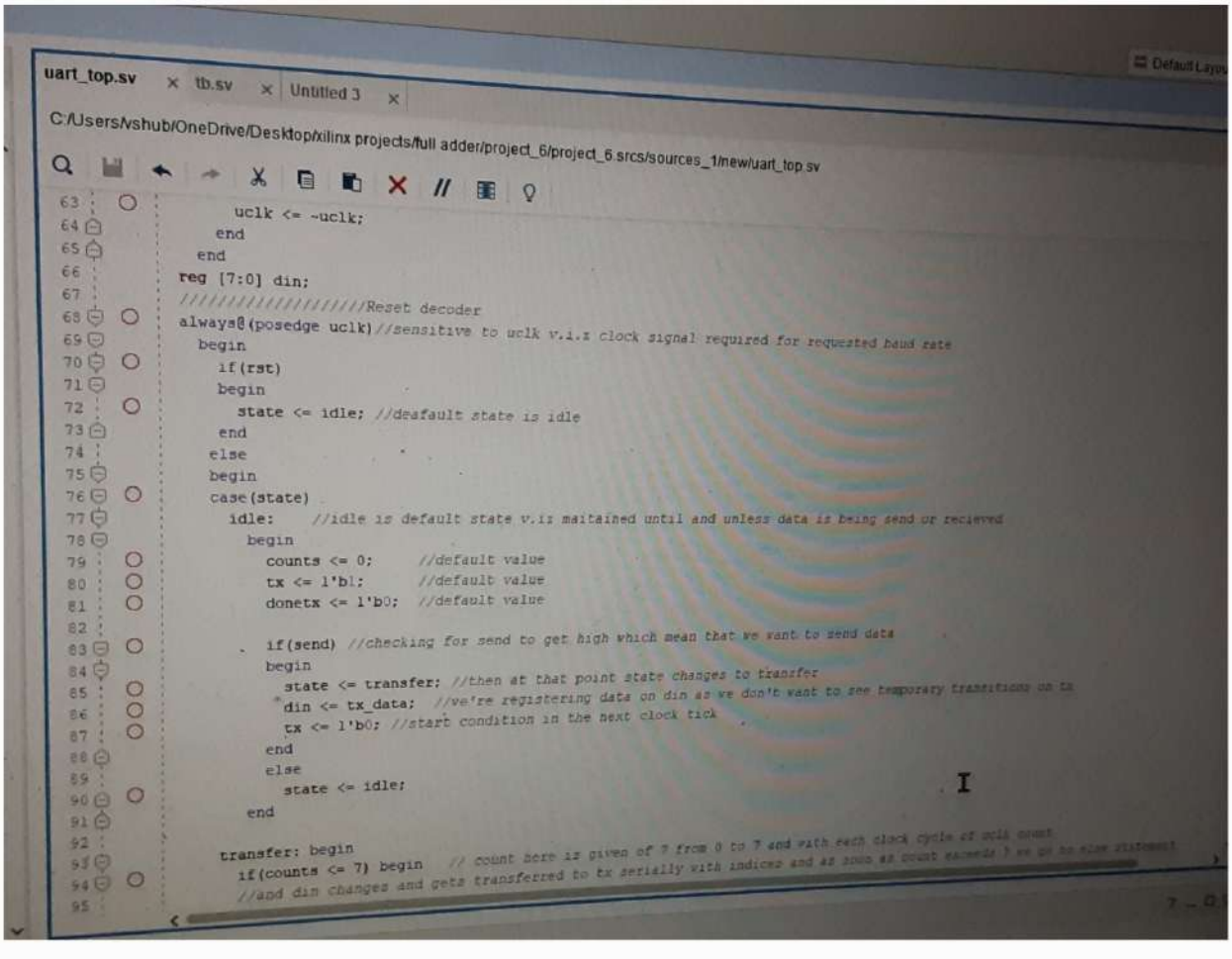
uart_top.sv × tb.sv × Untitled 3 ×

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sources_1/new/uart_top.sv

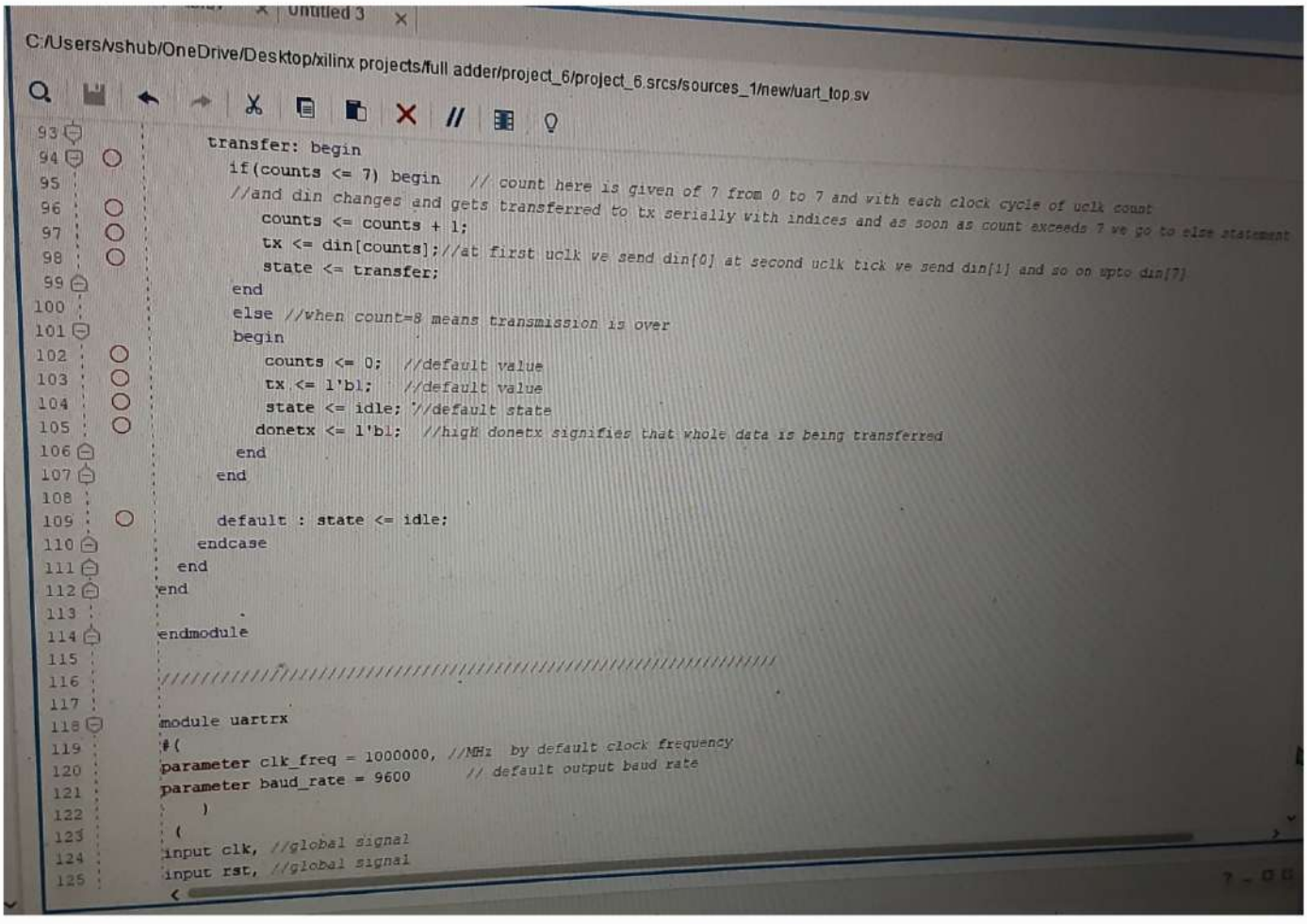
```
1   timescale 1ns / 1ps
2
3   module uart_top
4   #(
5   parameter clk_freq = 1000000,  //Mhz default frequency
6   parameter baud_rate = 9600
7   )
8   (
9     input clk,rst,  //global signals
10    input rx,
11    input [7:0] dintx,
12    input send,
13    output tx,
14    output [7:0] doutrx,
15    output donetx,
16    output donerx
17      );
18
19  uarttx
20  #(clk_freq, baud_rate)
21  utx   //name of instance of module uarttx
22  (clk, rst, send, dintx, tx, donetx);   //connecting ports
23
24  uartrx
25  #(clk_freq, baud_rate)
26  rtx //name of instance of module uartrx
27  (clk, rst, rx, donerx, doutrx); //connecting ports
28
29  endmodule
30
31  //////////////////////////////////////////////////////////////////
32
33  module uarttx
```

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sources_1/new/uart_top.sv

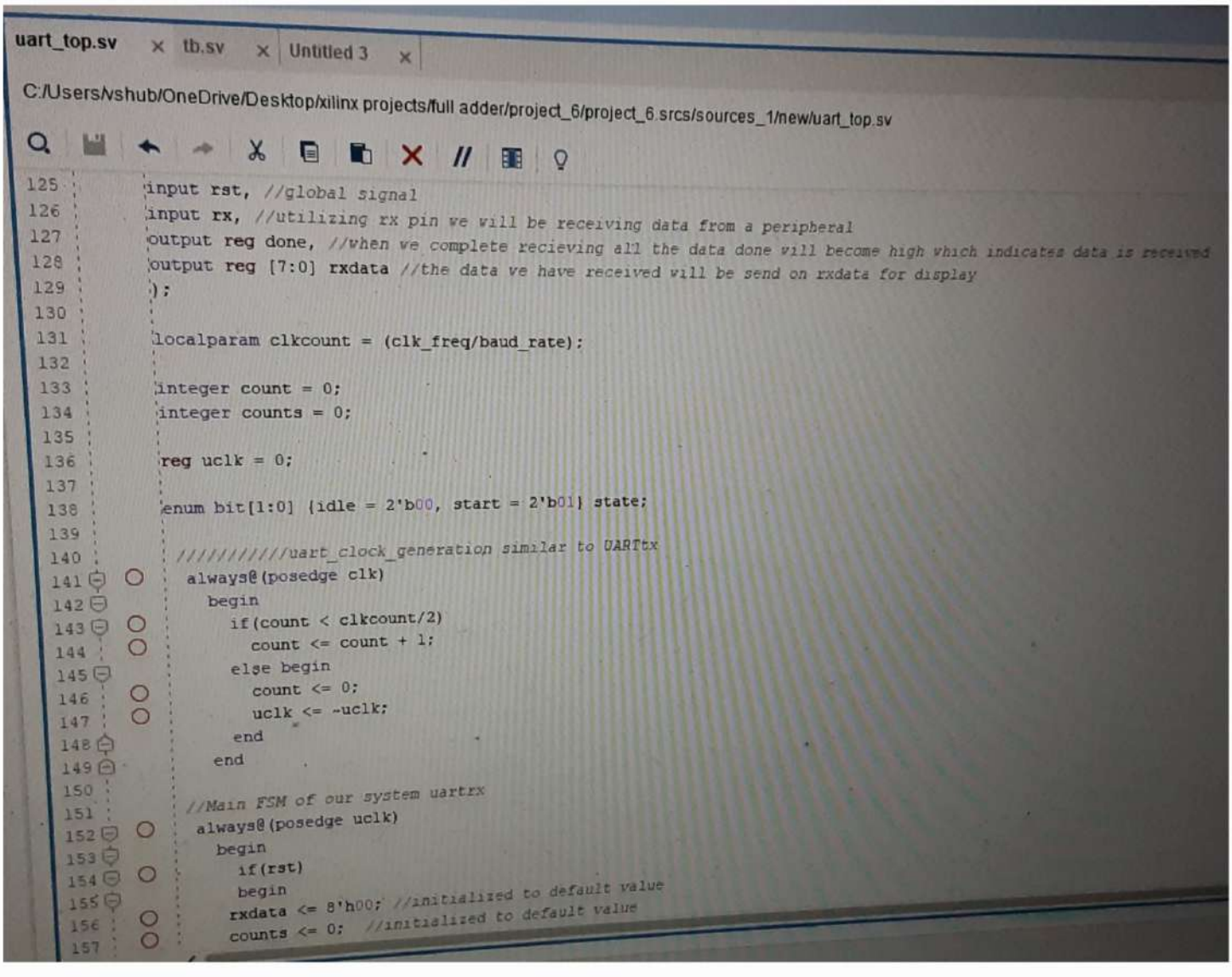
```
33  module uarttx
34  #(
35  parameter clk_freq = 1000000,  //MHz default frequency
36  parameter baud_rate = 9600
37  )
38  (
39  input clk,rst,   // Global signals
40  input send,      // Signifies that user wants to send data when it becomes 1(high)
41  input [7:0] tx_data,  // Data we want to send on tx pin is given at this tx_data pinv.1.s 8 bit data pin
42  output reg tx,  //output port on which data is send serially during transmission of data
43  output reg donetx  //when transmission of data completes this signals becomes high indicating that transmission is complete
44  );
45
46    localparam clkcount = (clk_freq/baud_rate); ///x helps in generation of clock signal for the specified baud rate
47
48  integer count = 0;
49  integer counts = 0;
50
51  reg uclk = 0; //uclk is kept in same state until and unless our count reaches clkcount/2 and
52  // after that it is inverted this is how we generate clock for specified baud rate v.1.z denoted by uclk
53
54  enum bit[1:0] {idle = 2'b00, start = 2'b01, transfer = 2'b10, done = 2'b11} state;
55
56  //////////uart_clock_generation
57  always@(posedge clk) //sensitive to onboard clock
58    begin
59      if(count < clkcount/2)
60        count <= count + 1;
61      else begin
62        count <= 0;
63        uclk <= ~uclk;
64      end
65    end
```



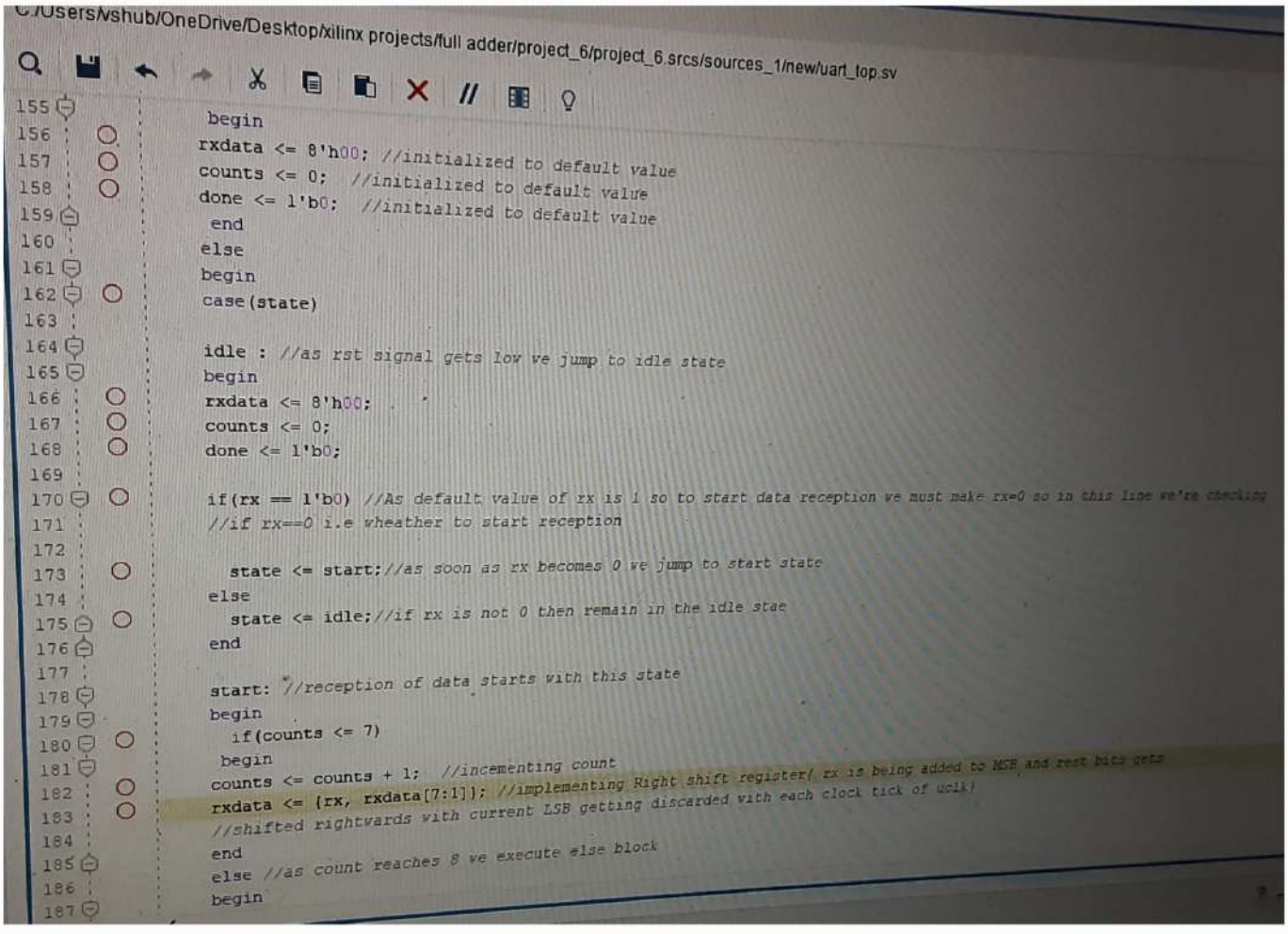

```
63        uclk <= ~uclk;
64      end
65    end
66  reg [7:0] din;
67  //////////////////Reset decoder
68  always@(posedge uclk)//sensitive to uclk v.1.z clock signal required for requested baud rate
69    begin
70      if(rst)
71      begin
72        state <= idle; //deafault state is idle
73      end
74      else
75      begin
76      case(state)
77        idle:    //idle is default state v.1z maitained until and unless data is being send or recieved
78          begin
79            counts <= 0;     //default value
80            tx <= 1'b1;      //default value
81            donetx <= 1'b0;  //default value
82
83            if(send) //checking for send to get high which mean that we want to send data
84            begin
85              state <= transfer; //then at that point state changes to transfer
86              din <= tx_data;  //we're registering data on din as we don't want to see temporary transitions on tx
87              tx <= 1'b0; //start condition in the next clock tick
88            end
89            else
90              state <= idle;
91          end
92
93      transfer: begin
94        if(counts <= 7) begin   // count here is given of 7 from 0 to 7 and with each clock cycle of uclk count
95        //and din changes and gets transferred to tx serially with indices and as soon as count exceeds 7 we go to else statement
```

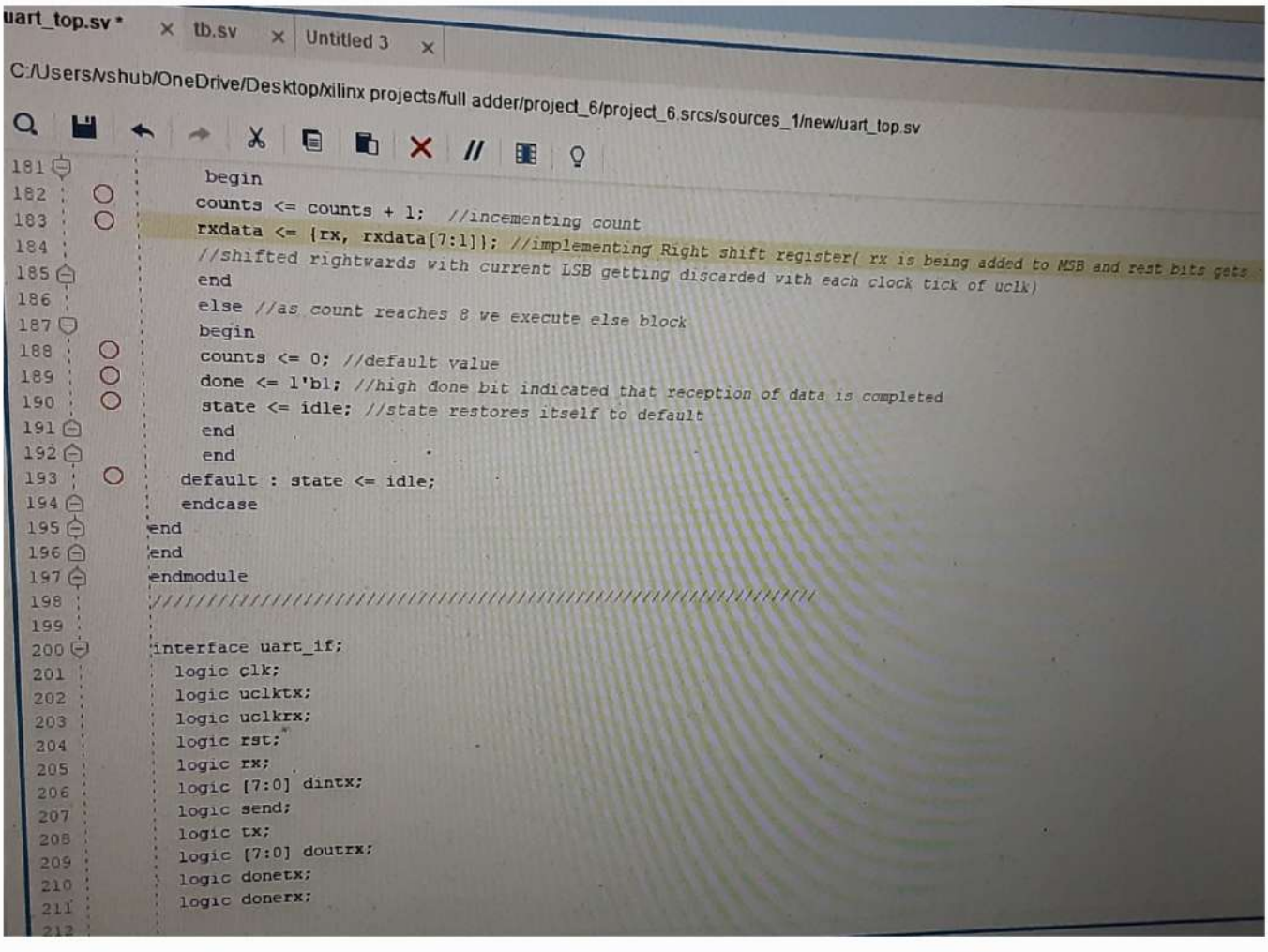


```
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sources_1/new/uart_top.sv

93      transfer: begin
94        if(counts <= 7) begin    // count here is given of 7 from 0 to 7 and with each clock cycle of uclk count
95        //and din changes and gets transferred to tx serially with indices and as soon as count exceeds 7 we go to else statement
96          counts <= counts + 1;
97          tx <= din[counts];//at first uclk we send din[0] at second uclk tick we send din[1] and so on upto din[7]
98          state <= transfer;
99        end
100       else //when count=8 means transmission is over
101       begin
102         counts <= 0;  //default value
103         tx <= 1'b1;   //default value
104         state <= idle; //default state
105        donetx <= 1'b1;  //high donetx signifies that whole data is being transferred
106       end
107     end
108
109     default : state <= idle;
110   endcase
111 end
112 end
113
114 endmodule
115
116 //////////////////////////////////////////////////////////////////////
117
118 module uartrx
119 #(
120 parameter clk_freq = 1000000, //MHz  by default clock frequency
121 parameter baud_rate = 9600      // default output baud rate
122   )
123   (
124 input clk, //global signal
125 input rst, //global signal
```



```
uart_top.sv  tb.sv  Untitled 3
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sources_1/new/uart_top.sv

125 input rst, //global signal
126 input rx, //utilizing rx pin we will be receiving data from a peripheral
127 output reg done, //when we complete recieving all the data done will become high which indicates data is received
128 output reg [7:0] rxdata //the data we have received will be send on rxdata for display
129 );
130
131 localparam clkcount = (clk_freq/baud_rate);
132
133 integer count = 0;
134 integer counts = 0;
135
136 reg uclk = 0;
137
138 enum bit[1:0] {idle = 2'b00, start = 2'b01} state;
139
140 //////////uart_clock_generation similar to UARTtx
141   always@(posedge clk)
142     begin
143       if(count < clkcount/2)
144         count <= count + 1;
145       else begin
146         count <= 0;
147         uclk <= ~uclk;
148       end
149     end
150
151 //Main FSM of our system uartrx
152   always@(posedge uclk)
153     begin
154       if(rst)
155       begin
156       rxdata <= 8'h00; //initialized to default value
157       counts <= 0;   //initialized to default value
```


C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sources_1/new/uart_top.sv
155 begin
156 rxdata <= 8'h00; //initialized to default value
157 counts <= 0; //initialized to default value
158 done <= 1'b0; //initialized to default value
159 end
160 else
161 begin
162 case(state)
163
164 idle : //as rst signal gets low we jump to idle state
165 begin
166 rxdata <= 8'h00;
167 counts <= 0;
168 done <= 1'b0;
169
170 if(rx == 1'b0) //As default value of rx is 1 so to start data reception we must make rx=0 so in this line we're checking
171 //if rx==0 i.e wheather to start reception
172
173 state <= start;//as soon as rx becomes 0 we jump to start state
174 else
175 state <= idle;//if rx is not 0 then remain in the idle stae
176 end
177
178 start: //reception of data starts with this state
179 begin
180 if(counts <= 7)
181 begin
182 counts <= counts + 1; //incementing count
183 rxdata <= {rx, rxdata[7:1]}; //implementing Right shift register(rx is being added to MSB and rest bits gets
184 //shifted rightwards with current LSB getting discarded with each clock tick of uclk)
185 end
186 else //as count reaches 8 we execute else block
187 begin


uart_top.sv *
tb.sv
Untitled 3
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sources_1/new/uart_top.sv
181 begin
182 counts <= counts + 1; //incementing count
183 rxdata <= {rx, rxdata[7:1]}; //implementing Right shift register(rx is being added to MSB and rest bits gets
184 //shifted rightwards with current LSB getting discarded with each clock tick of uclk)
185 end
186 else //as count reaches 8 we execute else block
187 begin
188 counts <= 0; //default value
189 done <= 1'b1; //high done bit indicated that reception of data is completed
190 state <= idle; //state restores itself to default
191 end
192 end
193 default : state <= idle;
194 endcase
195 end
196 end
197 endmodule
198 //
199
200 interface uart_if;
201 logic clk;
202 logic uclktx;
203 logic uclkrx;
204 logic rst;
205 logic rx;
206 logic [7:0] dintx;
207 logic send;
208 logic tx;
209 logic [7:0] doutrx;
210 logic donetx;
211 logic donerx;

→ SIMULATION SOURCE CODE (TESTBENCH):

uart_top.sv * | tb.sv * | Untitled 3

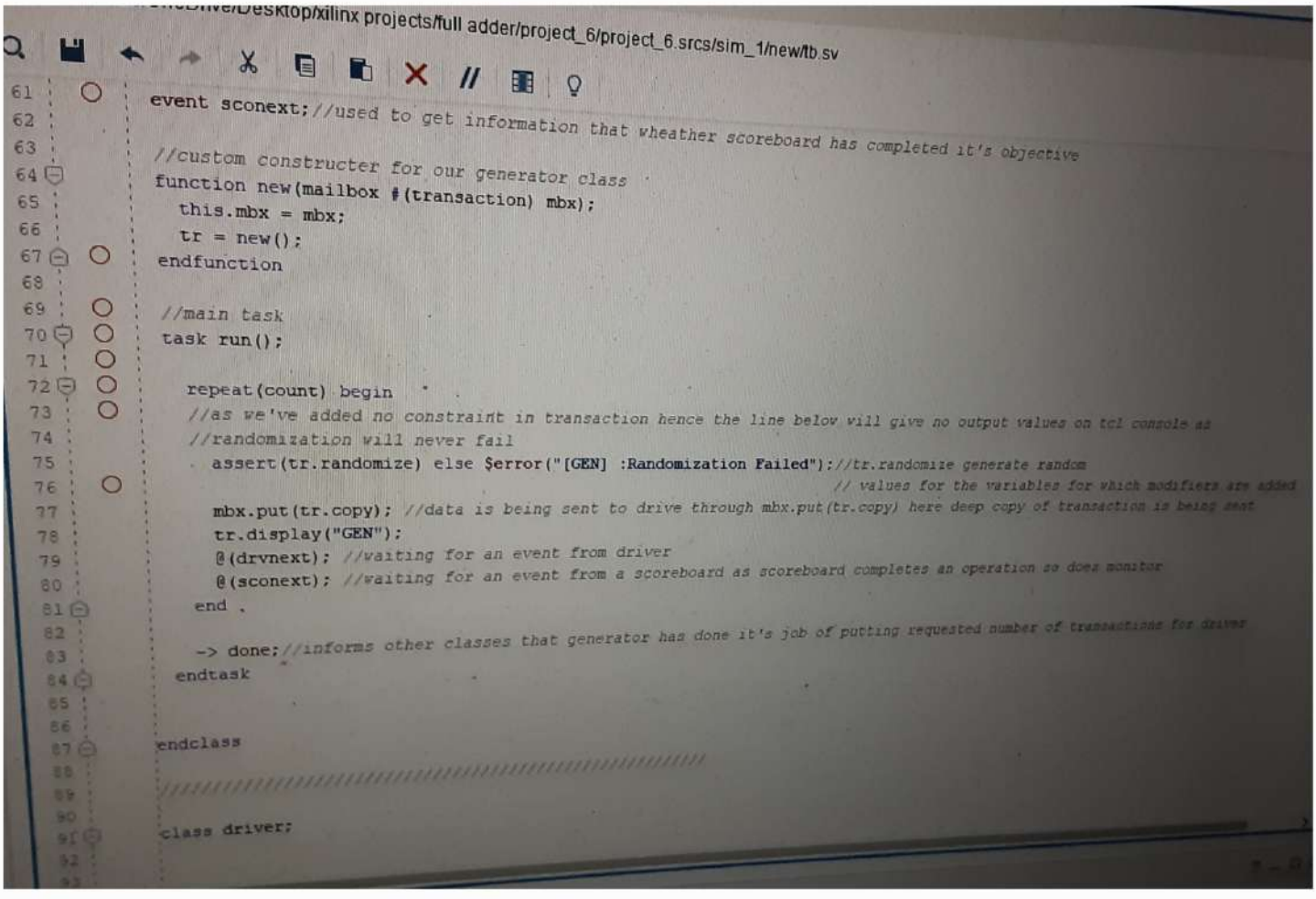
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

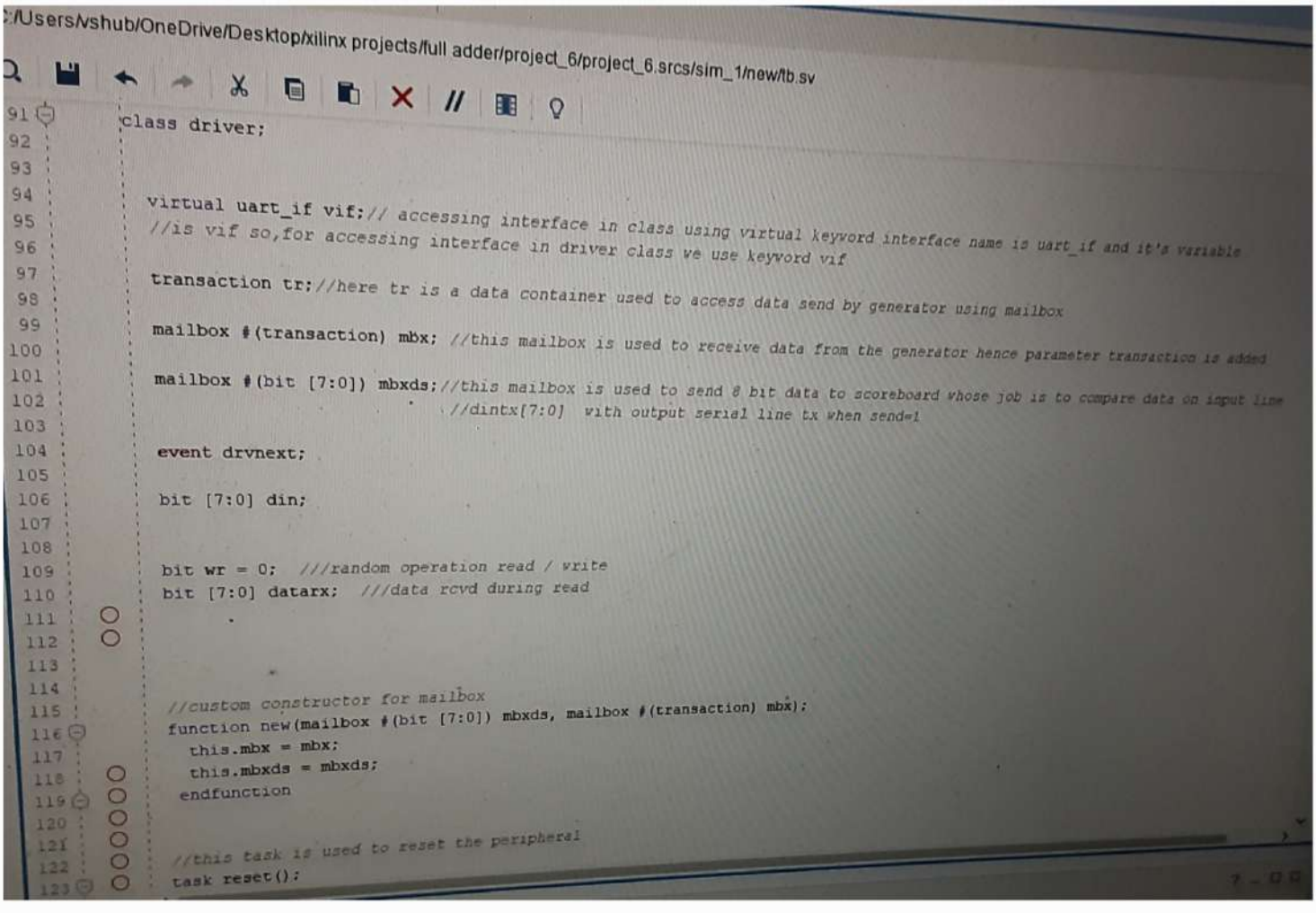
```
1  //half duplex
2
3  class transaction;//primary purpose of transaction class is to include variable for all the input and output ports
4                    //which are present in a design
5
6  typedef enum bit [1:0] {write = 2'b00 , read = 2'b01} oper_type;//enum type allows us to detect type of operation it
7                                                                  // has two values 0=write and 1=read
8   //write means we want to write data to a peripheral and read means we want to read data from a peripheral
9
10 randc oper_type oper;//oper is variable of type oper_type declared above.oper is randomize by putting rand modifier so it
11 // may be assigned 0 or 1 after we call randomize() global signals clk and rst won't be added in transaction class instead
12 //they'll be added in testbench top and driver respectively
13
14 bit rx;//rx is data we receive from peripheral as rx may be 0 or 1 so it is declared of bit type
15
16 //we may add rand modifier for other input signals namely rx,tx,send but number of iterations will exceed and tbe is oper_type
17 // we have to add 3rd state v.i.z randomized sate so inorder to avoid further complexity
18 rand bit [7:0] dintx;
19
20 bit send;
21 bit tx;
22
23 bit [7:0] doutrx;
24 bit donetx;
25 bit donerx;
26
27
28
29 function void display (input string tag);// tag represents class name sending data
30   $display("[%0s] : oper : %0s send : %0b TX_DATA : %b RX_IN : %0b TX_OUT : %0b RX_OUT : %b DONE_TX : %0b DONE_RX : %0b",
31    tag, oper.name(), send, dintx, rx, tx, doutrx, donetx, donerx);
32 endfunction// %0s is for string v.i.z for tag (class name sending data)
33
```

uart_top.sv * | tb.sv * | Untitled 3

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

```
31    tag, oper.name(), send, dintx, rx, tx, doutrx, donetx, donerx);
32 endfunction// %0s is for string v.i.z for tag (class name sending data)
33
34 function transaction copy();// performing deep copy of transaction
35   copy = new();
36   copy.rx = this.rx;
37   copy.dintx = this.dintx;
38   copy.send = this.send;
39   copy.tx = this.tx;
40   copy.doutrx = this.doutrx;
41   copy.donetx = this.donetx;
42   copy.donerx = this.donerx;
43   copy.oper = this.oper;
44 endfunction
45
46 endclass
47
48 class generator; //generator class
49
50  transaction tr; //creating handler for transaction
51
52  mailbox #(transaction) mbx; //mailbox is required for sending data from generator to a driver
53
54  //initializing events
55  event done;  //done is used to convey information to our testbench top that we have completed sending
56               //requested number of transactions
57
58  int count = 0;//stores the number of iterations user requests
59
60  event drvnext;//used to sense wheather driver have completed it's operation of triggering an interface
61  event sconext;//used to get information that wheather scoreboard has completed it's objective
62
63  //custom constructer for our generator class
```


Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv
event sconext;//used to get information that wheather scoreboard has completed it's objective
//custom constructer for our generator class
function new(mailbox #(transaction) mbx);
this.mbx = mbx;
tr = new();
endfunction
//main task
task run();
repeat(count) begin
//as we've added no constraint in transaction hence the line below will give no output values on tcl console as
//randomization will never fail
assert(tr.randomize) else $error("[GEN] :Randomization Failed");//tr.randomize generate random
// values for the variables for which modifiers are added
mbx.put(tr.copy); //data is being sent to drive through mbx.put(tr.copy) here deep copy of transaction is being sent
tr.display("GEN");
@(drvnext); //waiting for an event from driver
@(sconext); //waiting for an event from a scoreboard as scoreboard completes an operation so does monitor
end
-> done;//informs other classes that generator has done it's job of putting requested number of transactions for driver
endtask
endclass
class driver;

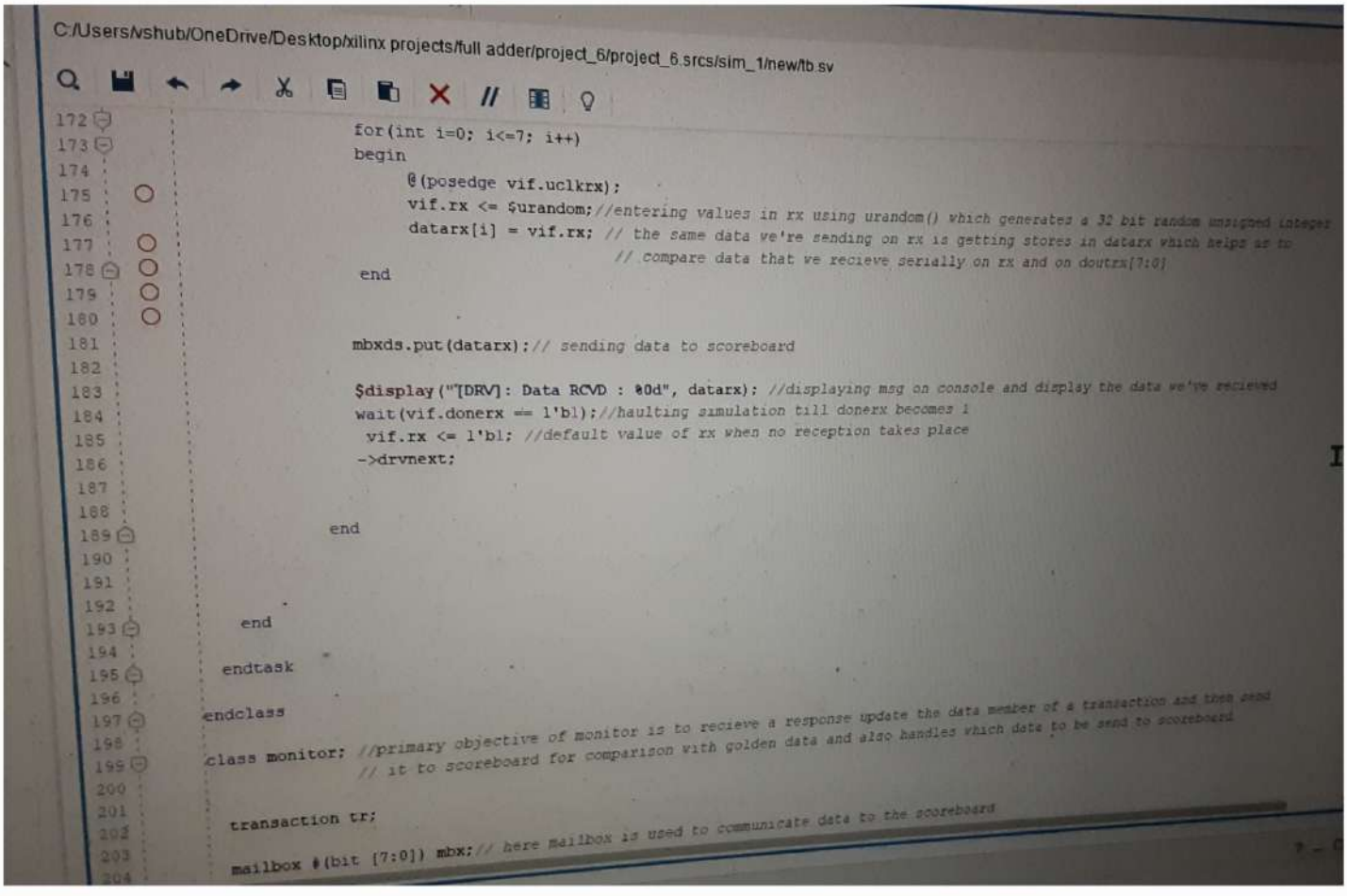

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv
class driver;
virtual uart_if vif;// accessing interface in class using virtual keyword interface name is uart_if and it's variable
//is vif so,for accessing interface in driver class we use keyword vif
transaction tr;//here tr is a data container used to access data send by generator using mailbox
mailbox #(transaction) mbx; //this mailbox is used to receive data from the generator hence parameter transaction is added
mailbox #(bit [7:0]) mbxds;//this mailbox is used to send 8 bit data to scoreboard whose job is to compare data on input line
//dintx[7:0] with output serial line tx when send=1
event drvnext;
bit [7:0] din;
bit wr = 0; ///random operation read / write
bit [7:0] datarx; ///data rcvd during read
//custom constructor for mailbox
function new(mailbox #(bit [7:0]) mbxds, mailbox #(transaction) mbx);
this.mbx = mbx;
this.mbxds = mbxds;
endfunction
//this task is used to reset the peripheral
task reset();

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

```
121
122     //this task is used to reset the peripheral
123     task reset();
124       vif.rst <= 1'b1;//reset is active high by default
125       vif.dintx <= 0; //default values
126       vif.send <= 0;  //default values
127       vif.rx <= 1'b1;  //default values
128       vif.doutrx <= 0;  //default values
129       vif.tx <= 1'b1;  //default values
130       vif.donerx <= 0; //default values
131       vif.donetx <= 0; //default values
132       repeat(5) @(posedge vif.uclktx); //waiting for 5 clock ticks of uclktx (a slower clock) and then apply 0 to vif.rst to it
133       vif.rst <= 1'b0;
134       @(posedge vif.uclktx);
135       $display("[DRV] : RESET DONE");
136     endtask
137
138
139     //in this task we decode the type of operations and apply it to a signal
140     task run();
141
142       forever begin
143         mbx.get(tr); //receiving transaction from a generator
144         //decoding type of operation
145         if(tr.oper == 2'b00)  ////data transmission ,performing write operation
146             begin
147             // v=at a time we can either write data to peripheral or read data from peripheral
148               @(posedge vif.uclktx);//waiting for posetive edge of uclktx
149               vif.rst <= 1'b0;
150               vif.send <= 1'b1;  ///start data sending op
151               vif.rx <= 1'b1;
152               vif.dintx = tr.dintx;// data send by generator tr.dintx is being applied to dintx
153               @(posedge vif.uclktx);//after data is received we make send signal low
```

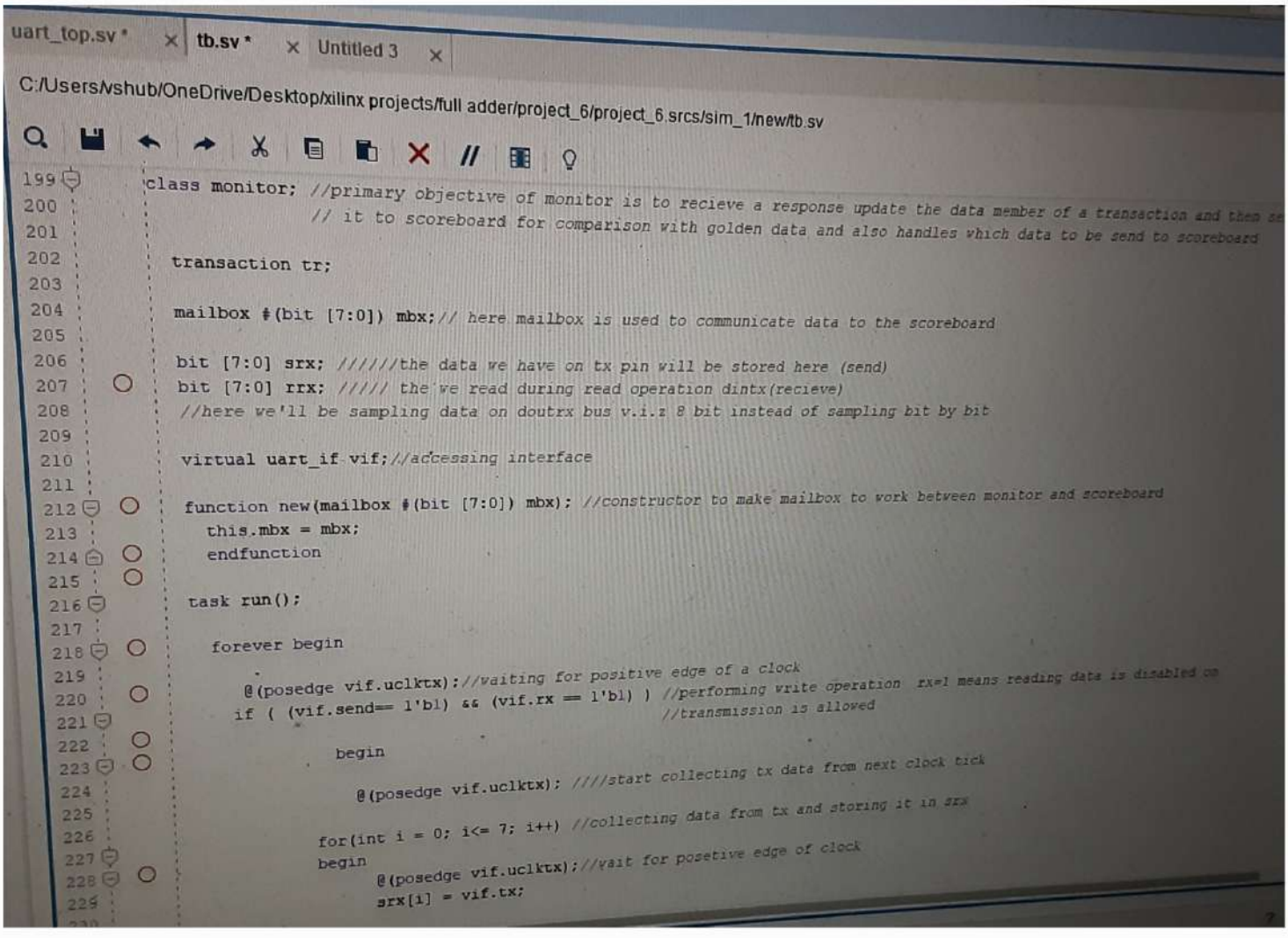
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

```
142       forever begin
143         mbx.get(tr); //receiving transaction from a generator
144         //decoding type of operation
145         if(tr.oper == 2'b00)  ////data transmission ,performing write operation
146             begin
147             // v=at a time we can either write data to peripheral or read data from peripheral
148               @(posedge vif.uclktx);//waiting for posetive edge of uclktx
149               vif.rst <= 1'b0;
150               vif.send <= 1'b1;  ///start data sending op
151               vif.rx <= 1'b1;
152               vif.dintx = tr.dintx;// data send by generator tr.dintx is being applied to dintx
153               @(posedge vif.uclktx);//after data is received we make send signal low
154               vif.send <= 1'b0;
155                 ////wait for completion
156               //repeat(9) @(posedge vif.uclktx);
157               mbxds.put(tr.dintx);//same data on dintx is being send to the scoreboard to be compared with data on output serial port to
158               $display("[DRV]: Data Sent : %0d", tr.dintx);
159                wait(vif.donetx == 1'b1);  //we'll hold our simulation till we get donetx=1
160                ->drvnext;  //this even trigger signifies that driver has completed it's job of triggering an interface
161           end
162
163       else if (tr.oper == 2'b01) //data reception,read operation
164               begin
165
166                 @(posedge vif.uclkrx);//waiting for posetive edge of uclkrx both clks uclktxand uclkrx are working at same frequency
167                  vif.rst <= 1'b0;
168                  vif.rx <= 1'b0; //start condition for read operation
169                  vif.send <= 1'b0;//as we're receiving data from peripheral hence send=0
170                  @(posedge vif.uclkrx);
171
172                 for(int i=0; i<=7; i++)
173                 begin
                      @(posedge vif.uclkrx);
```



C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

```
172                 for(int i=0; i<=7; i++)
173                 begin
174                     @(posedge vif.uclkrx);
175                     vif.rx <= $urandom;//entering values in rx using urandom() which generates a 32 bit random unsigned integer
176                     datarx[i] = vif.rx; // the same data we're sending on rx is getting stores in datarx which helps us to
177                                         // compare data that we recieve serially on rx and on doutrx[7:0]
178                 end
179
180
181                 mbxds.put(datarx);// sending data to scoreboard
182
183                 $display("[DRV]: Data RCVD : %0d", datarx); //displaying msg on console and display the data we've recieved
184                 wait(vif.donerx == 1'b1);//haulting simulation till donerx becomes 1
185                  vif.rx <= 1'b1; //default value of rx when no reception takes place
186                 ->drvnext;
187
188
189             end
190
191
192
193       end
194
195     endtask
196
197   endclass
198
199   class monitor; //primary objective of monitor is to recieve a response update the data member of a transaction and then send
200                  // it to scoreboard for comparison with golden data and also handles which data to be send to scoreboard
201
202     transaction tr;
203
204     mailbox #(bit [7:0]) mbx;// here mailbox is used to communicate data to the scoreboard
```



uart_top.sv * | tb.sv * | Untitled 3

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

```
199   class monitor; //primary objective of monitor is to recieve a response update the data member of a transaction and then se
200                  // it to scoreboard for comparison with golden data and also handles which data to be send to scoreboard
201
202     transaction tr;
203
204     mailbox #(bit [7:0]) mbx;// here mailbox is used to communicate data to the scoreboard
205
206     bit [7:0] srx; //////the data we have on tx pin will be stored here (send)
207     bit [7:0] rrx; ///// the we read during read operation dintx(recieve)
208     //here we'll be sampling data on doutrx bus v.i.z 8 bit instead of sampling bit by bit
209
210     virtual uart_if vif;//accessing interface
211
212     function new(mailbox #(bit [7:0]) mbx); //constructor to make mailbox to work between monitor and scoreboard
213       this.mbx = mbx;
214       endfunction
215
216     task run();
217
218       forever begin
219
220         @(posedge vif.uclktx);//waiting for positive edge of a clock
221         if ( (vif.send== 1'b1) && (vif.rx == 1'b1) ) //performing write operation  rx=1 means reading data is disabled on
222                                                      //transmission is allowed
223                 begin
224
225                   @(posedge vif.uclktx); ////start collecting tx data from next clock tick
226
227                 for(int i = 0; i<= 7; i++) //collecting data from tx and storing it in srx
228                 begin
229                     @(posedge vif.uclktx);//wait for posetive edge of clock
                      srx[i] = vif.tx;
```

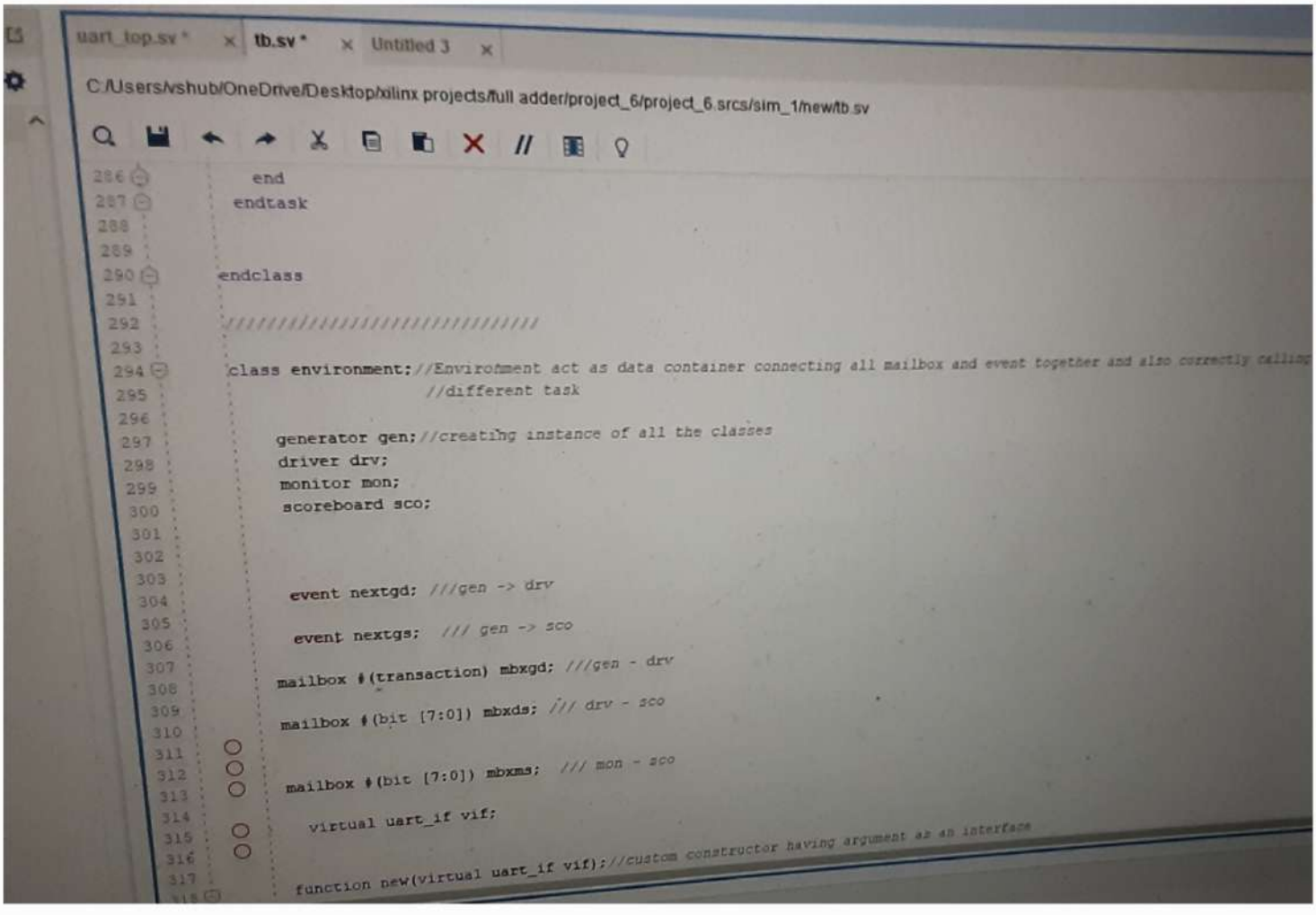
art_top.sv * | tb.sv * | Untitled 3

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

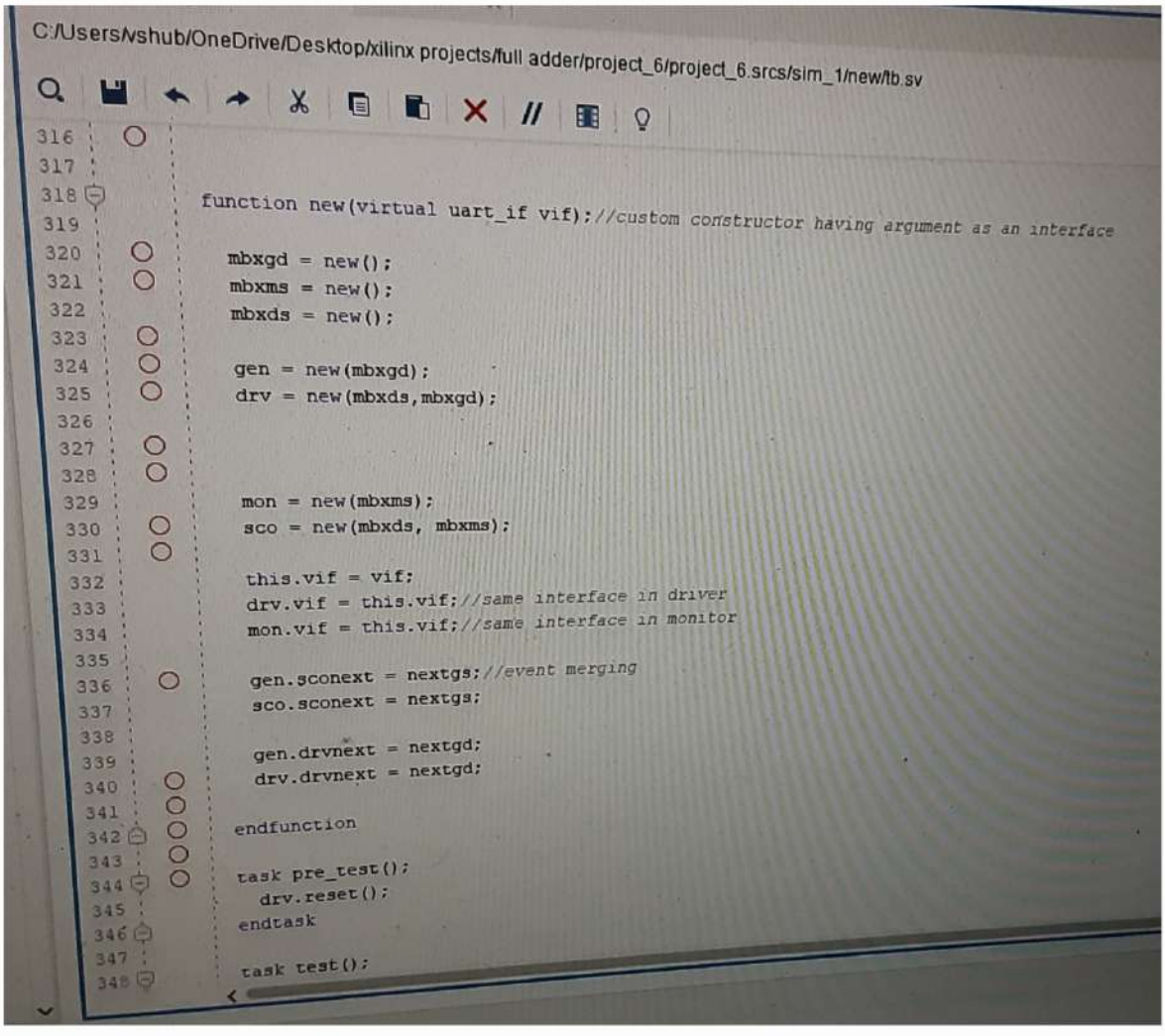
```
226
227
228             for(int i = 0; i<= 7; i++) //collecting data from tx and storing it in srx
229             begin
230                   @(posedge vif.uclktx);//wait for posetive edge of clock
231                   srx[i] = vif.tx;
232             end
233
234
235                $display("[MON] : DATA SEND on UART TX %0d", srx);
236
237                ////////////wait for done tx before proceeding next transaction
238               @(posedge vif.uclktx); //
239               mbx.put(srx);
240
241             end
242
243       else if ((vif.rx == 1'b0) && (vif.send == 1'b0) ) //rx=0 signifies data reception and during reception send=0
244         begin
245           wait(vif.donerx == 1);//waiting for completion of data reception
246           rrx = vif.doutrx;     //data available on 8 bit output bus is stored in rrx
247           $display("[MON] : DATA RCVD RX %0d", rrx);
248           @(posedge vif.uclktx);
249           mbx.put(rrx);//data is put in mbx so that it can be recieved by scoreboard
250       end
251     end
252   endtask
253
254
255   endclass
256
257
258
```

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

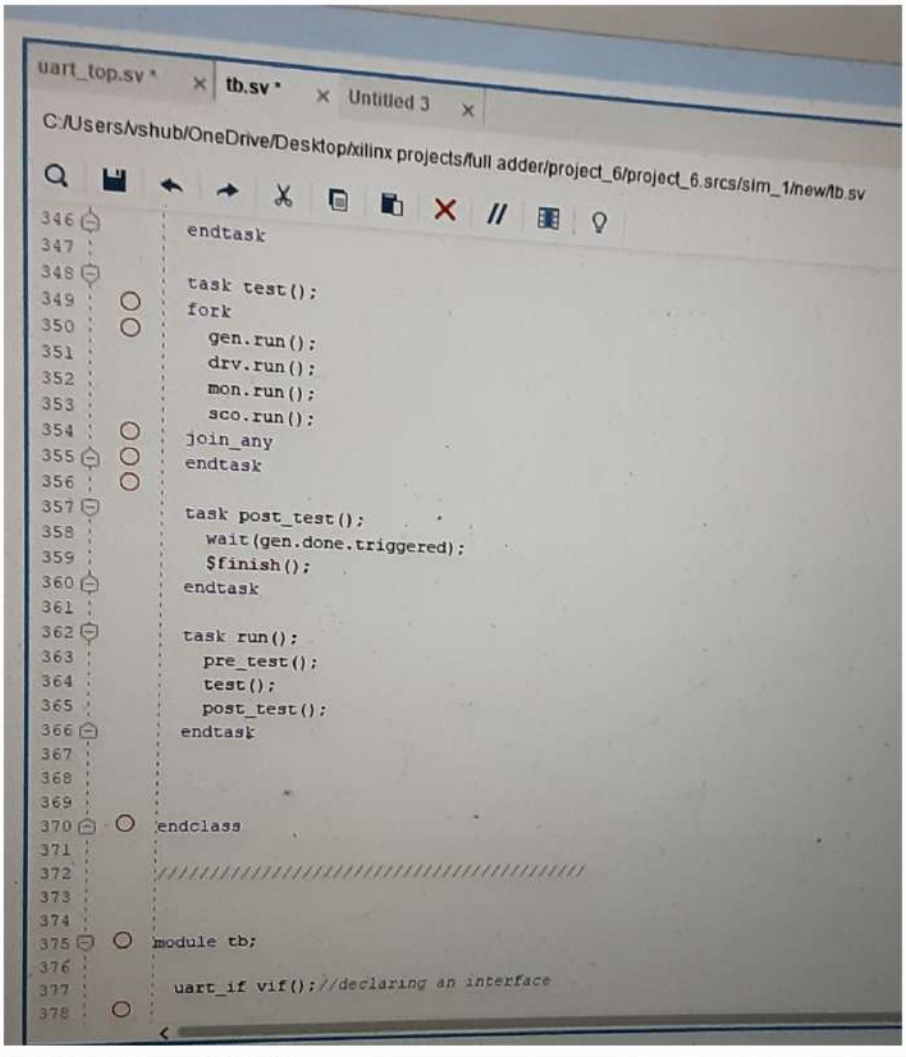
```
256
257   ////////////////////////////////////////////////////////////////////////////////
258
259   class scoreboard;
260     mailbox #(bit [7:0]) mbxds, mbxms;//scoreboard needs two mailbox one to recieve data from monitor and other to recieve data
261                                       // from driver
262     bit [7:0] ds; //data container
263     bit [7:0] ms; //data container
264
265      event sconext;//we need and event because before generator starting to send next transaction it should recieve trigger from sconext
266
267     function new(mailbox #(bit [7:0]) mbxds, mailbox #(bit [7:0]) mbxms);//constructor having 2 parameters from driver and scoreboard
268       this.mbxds = mbxds;
269       this.mbxms = mbxms;
270     endfunction
271
272     task run();
273       forever begin
274
275         mbxds.get(ds);//storing data from driver in ds
276         mbxms.get(ms);//storing data from monitor in ms
277
278         $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
279         if(ds == ms)
280           $display("DATA MATCHED");
281         else
282           $display("DATA MISMATCHED");
283
284        ->sconext; //triggering an event so that generator could start it's next transaction
285       end
286     endtask
287
288
```



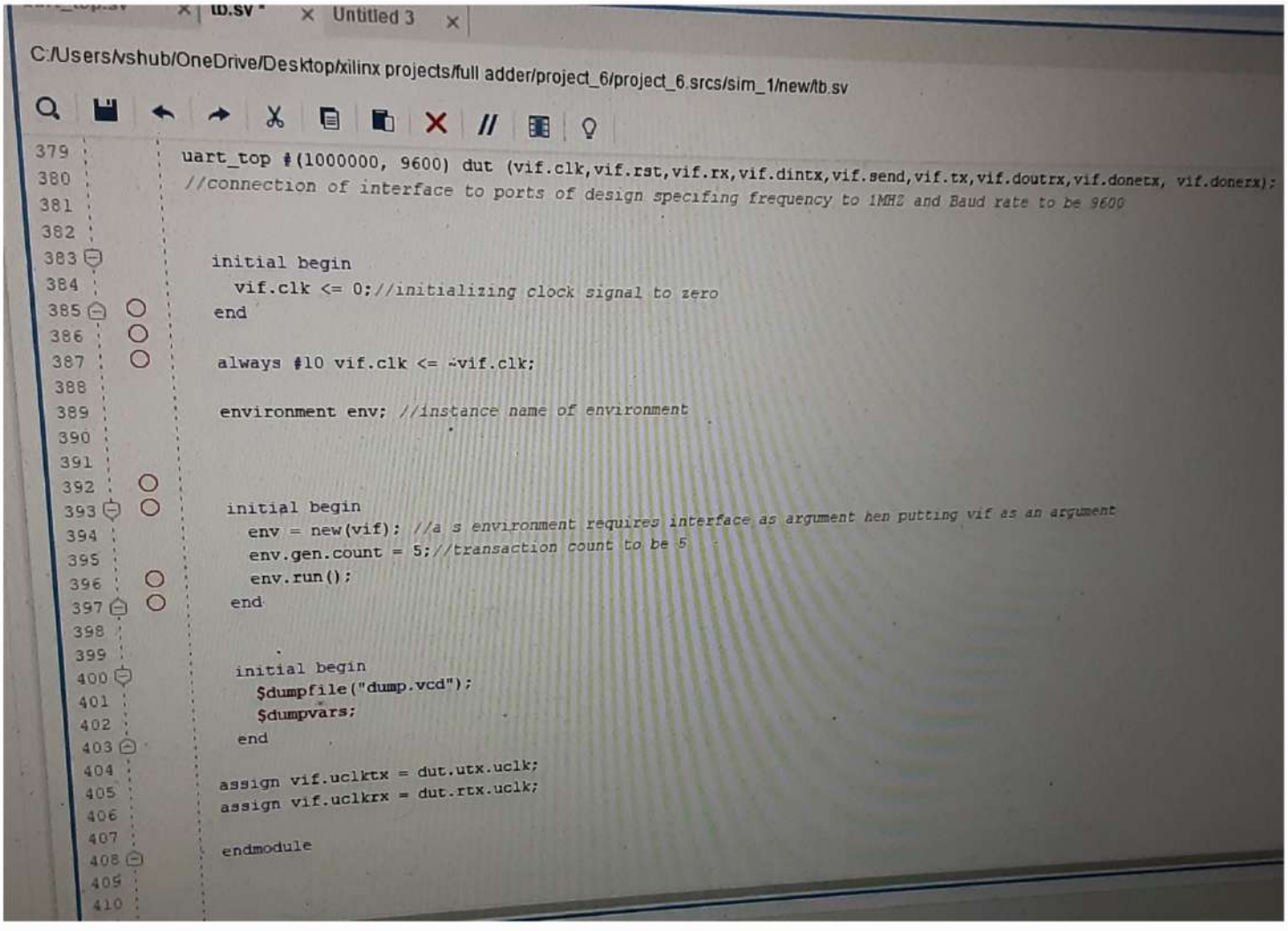
```
uart_top.sv *   tb.sv *   Untitled 3

C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

286        end
287      endtask
288
289
290    endclass
291
292    ////////////////////////////////
293
294    class environment;//Environment act as data container connecting all mailbox and event together and also correctly calling
295                      //different task
296
297      generator gen;//creating instance of all the classes
298      driver drv;
299      monitor mon;
300      scoreboard sco;
301
302
303       event nextgd; ///gen -> drv
304
305       event nextgs;  /// gen -> sco
306
307      mailbox #(transaction) mbxgd; ///gen - drv
308
309      mailbox #(bit [7:0]) mbxds; /// drv - sco
310
311
312      mailbox #(bit [7:0]) mbxms;  /// mon - sco
313
314        virtual uart_if vif;
315
316
317      function new(virtual uart_if vif);//custom constructor having argument as an interface
```



```
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv

316
317
318    function new(virtual uart_if vif);//custom constructor having argument as an interface
319
320      mbxgd = new();
321      mbxms = new();
322      mbxds = new();
323
324      gen = new(mbxgd);
325      drv = new(mbxds,mbxgd);
326
327
328
329      mon = new(mbxms);
330      sco = new(mbxds, mbxms);
331
332      this.vif = vif;
333      drv.vif = this.vif;//same interface in driver
334      mon.vif = this.vif;//same interface in monitor
335
336      gen.sconext = nextgs;//event merging
337      sco.sconext = nextgs;
338
339      gen.drvnext = nextgd;
340      drv.drvnext = nextgd;
341
342    endfunction
343
344    task pre_test();
345      drv.reset();
346    endtask
347
348    task test();
```

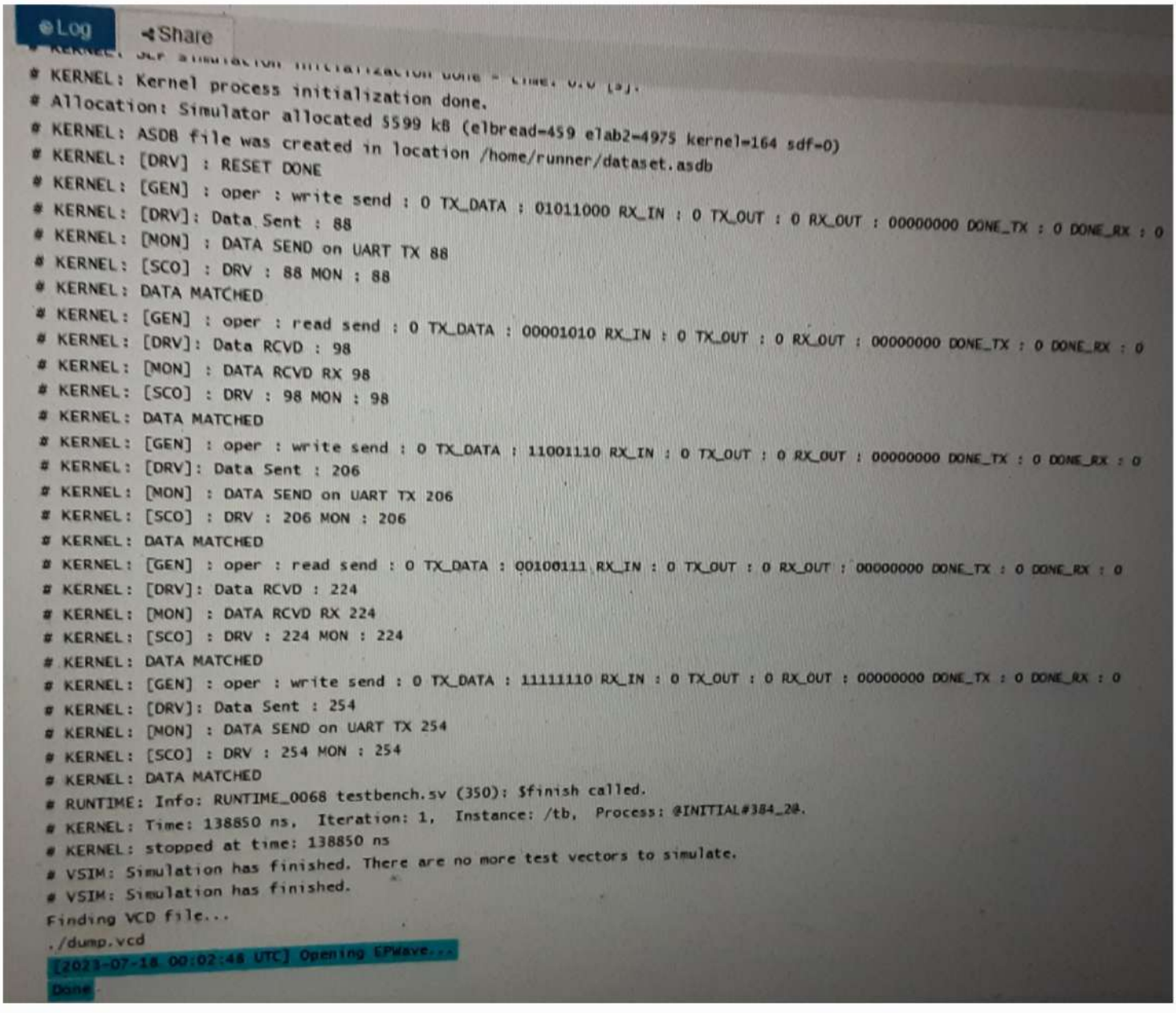


```
uart_top.sv *   tb.sv *   Untitled 3
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv
346     endtask
347
348     task test();
349     fork
350       gen.run();
351       drv.run();
352       mon.run();
353       sco.run();
354     join_any
355     endtask
356
357     task post_test();
358       wait(gen.done.triggered);
359       $finish();
360     endtask
361
362     task run();
363       pre_test();
364       test();
365       post_test();
366     endtask
367
368
369
370   endclass
371
372   ////////////////////////////////////////
373
374
375   module tb;
376
377     uart_if vif();//declaring an interface
378
```



```
tb.sv *   Untitled 3
C:/Users/vshub/OneDrive/Desktop/xilinx projects/full adder/project_6/project_6.srcs/sim_1/new/tb.sv
379     uart_top #(1000000, 9600) dut (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);
380     //connection of interface to ports of design specifing frequency to 1MHZ and Baud rate to be 9600
381
382
383       initial begin
384         vif.clk <= 0;//initializing clock signal to zero
385       end
386
387       always #10 vif.clk <= ~vif.clk;
388
389       environment env; //instance name of environment
390
391
392
393       initial begin
394         env = new(vif); //a s environment requires interface as argument hen putting vif as an argument
395         env.gen.count = 5;//transaction count to be 5
396         env.run();
397       end
398
399
400       initial begin
401         $dumpfile("dump.vcd");
402         $dumpvars;
403       end
404
405     assign vif.uclktx = dut.utx.uclk;
406     assign vif.uclkrx = dut.rtx.uclk;
407
408     endmodule
409
410
```

→ CONSOLE


Log
Share
KERNEL: Kernel process initialization done.
Allocation: Simulator allocated 5599 kB (elbread=459 elab2=4975 kernel=164 sdf=0)
KERNEL: ASDB file was created in location /home/runner/dataset.asdb
KERNEL: [DRV] : RESET DONE
KERNEL: [GEN] : oper : write send : 0 TX_DATA : 01011000 RX_IN : 0 TX_OUT : 0 RX_OUT : 00000000 DONE_TX : 0 DONE_RX : 0
KERNEL: [DRV]: Data Sent : 88
KERNEL: [MON] : DATA SEND on UART TX 88
KERNEL: [SCO] : DRV : 88 MON : 88
KERNEL: DATA MATCHED
KERNEL: [GEN] : oper : read send : 0 TX_DATA : 00001010 RX_IN : 0 TX_OUT : 0 RX_OUT : 00000000 DONE_TX : 0 DONE_RX : 0
KERNEL: [DRV]: Data RCVD : 98
KERNEL: [MON] : DATA RCVD RX 98
KERNEL: [SCO] : DRV : 98 MON : 98
KERNEL: DATA MATCHED
KERNEL: [GEN] : oper : write send : 0 TX_DATA : 11001110 RX_IN : 0 TX_OUT : 0 RX_OUT : 00000000 DONE_TX : 0 DONE_RX : 0
KERNEL: [DRV]: Data Sent : 206
KERNEL: [MON] : DATA SEND on UART TX 206
KERNEL: [SCO] : DRV : 206 MON : 206
KERNEL: DATA MATCHED
KERNEL: [GEN] : oper : read send : 0 TX_DATA : 00100111 RX_IN : 0 TX_OUT : 0 RX_OUT : 00000000 DONE_TX : 0 DONE_RX : 0
KERNEL: [DRV]: Data RCVD : 224
KERNEL: [MON] : DATA RCVD RX 224
KERNEL: [SCO] : DRV : 224 MON : 224
KERNEL: DATA MATCHED
KERNEL: [GEN] : oper : write send : 0 TX_DATA : 11111110 RX_IN : 0 TX_OUT : 0 RX_OUT : 00000000 DONE_TX : 0 DONE_RX : 0
KERNEL: [DRV]: Data Sent : 254
KERNEL: [MON] : DATA SEND on UART TX 254
KERNEL: [SCO] : DRV : 254 MON : 254
KERNEL: DATA MATCHED
RUNTIME: Info: RUNTIME_0068 testbench.sv (350): $finish called.
KERNEL: Time: 138850 ns, Iteration: 1, Instance: /tb, Process: @INITIAL#384_2@.
KERNEL: stopped at time: 138850 ns
VSIM: Simulation has finished. There are no more test vectors to simulate.
VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2023-07-18 00:02:48 UTC] Opening EPWave...
Done