```
module UART_TX_RX_MASSIV_MODULE
#(
   parameter UART_BAUD_RATE             =  9600,//битрейт передачи
   parameter CLOCK_FREQUENCY            =  50000000,//частота сигнала IN_CLOCK
   parameter PARITY                     =  1,//параметр бита четности
   parameter NUM_OF_DATA_BITS_IN_PACK   =  8,//кол-во информационных бит в элементарной транзакции
   parameter NUMBER_STOP_BITS           =  2,//кол-во стоп-битов
   parameter TX_MASSIV_DEEP             =  4,//глубина буфера TX
   parameter RX_MASSIV_DEEP             =  4,//глубина буфера RX
   parameter RX_MASSIV_DEEP_LOG_2=$clog2(RX_MASSIV_DEEP),//определение размерности соответствующих регистров
   parameter TX_MASSIV_DEEP_LOG_2=$clog2(TX_MASSIV_DEEP)
)
(
   input IN_CLOCK,//входной тактовый сигнал
   input wire [NUM_OF_DATA_BITS_IN_PACK*TX_MASSIV_DEEP-1:0] IN_TX_DATA_MASSIV,//входной массив данных для передачи

   input [TX_MASSIV_DEEP_LOG_2:0] IN_TX_NUMBER_OF_PACKS_TO_SEND,//число пакетов, которые нужно отправить при следующей инициализации транзакции
   input IN_TX_LAUNCH,//сигнальная линия инициализации транзакции

   output reg OUT_TX_ACTIVE,//сигнальная линия занятости узла TX
   output reg OUT_TX_DONE,//сигнальная линия окончания передачи модулем TX

   input IN_RX_CLEAR_BUFFER,//сигнальная линия для очистки буфера принятых бит. Должен быть коротким.
   output reg [NUM_OF_DATA_BITS_IN_PACK*RX_MASSIV_DEEP-1:0] OUT_RX_DATA_MASSIV,//выходной вектор принятых данных
   output reg OUT_RX_ERROR,//сигнальная линия ошибки приема
   output reg [RX_MASSIV_DEEP_LOG_2:0] OUT_RX_NUM_OF_DATA_PACKS_READY,//число принятых пакетов с момента последней очистки буфера

   output      TX_PORT,//TX
   input       RX_PORT//RX

);
   wire [NUM_OF_DATA_BITS_IN_PACK-1:0] IN_UART_TX_DATA;
   wire [NUM_OF_DATA_BITS_IN_PACK-1:0] OUT_UART_RX_DATA;
   UART_TX_RX_MODULE
   #(
      .UART_BAUD_RATE(UART_BAUD_RATE),
      .CLOCK_FREQUENCY(CLOCK_FREQUENCY),
      .PARITY(PARITY),
      .NUM_OF_DATA_BITS_IN_PACK(NUM_OF_DATA_BITS_IN_PACK),
      .NUMBER_STOP_BITS(NUMBER_STOP_BITS)
   )
   UART
   (
      .IN_CLOCK(IN_CLOCK),
      .IN_TX_LAUNCH(IN_UART_TX_LAUNCH),
      .IN_TX_DATA(IN_UART_TX_DATA),
      .OUT_TX_ACTIVE(OUT_UART_TX_ACTIVE),
      .OUT_TX_DONE(OUT_UART_TX_DONE),
      .OUT_TX_STOP_BIT_ACTIVE(OUT_UART_TX_STOP_BIT_ACTIVE),
      .OUT_TX_START_BIT_ACTIVE(OUT_UART_TX_START_BIT_ACTIVE),
      .OUT_RX_DATA_READY(OUT_UART_RX_DATA_READY),
      .OUT_RX_DATA(OUT_UART_RX_DATA),
      .OUT_RX_ERROR(OUT_UART_RX_ERROR),
      .IN_RX_SERIAL(RX_PORT),
      .OUT_TX_SERIAL(TX_PORT)

   );
   //состояния автомата для передачи пакетов
   localparam STATE_WAIT            =  2'b00;
   localparam STATE_WRITE_PACKS     =  2'b01;
   localparam STATE_WAIT_UART_DONE  =  2'b10;

   localparam NUM_OF_DATA_BITS_IN_PACK_LOG_2=$clog2(NUM_OF_DATA_BITS_IN_PACK);

   reg   [1:0]    REG_TX_FSM_STATE;//состояние автомата для переачи данных
   reg [RX_MASSIV_DEEP_LOG_2:0] REG_RX_PACK_COUNT;//счетчик приема пакетов
   reg [TX_MASSIV_DEEP_LOG_2:0] REG_TX_PACK_COUNT;//счетчик отправки пакетов
   reg   [NUM_OF_DATA_BITS_IN_PACK*TX_MASSIV_DEEP-1:0] REG_TX_DATA_MASSIV ;
   reg   [TX_MASSIV_DEEP_LOG_2:0] REG_TX_NUMBER_OF_PACKS_TO_SEND;
   reg FIRST_CLOCK_AFTER_WRITE_PACK;
   reg [NUM_OF_DATA_BITS_IN_PACK-1:0] REG_UART_TX_DATA;
   reg REG_UART_TX_LAUNCH;
```

```
 73      assign IN_UART_TX_DATA=REG_UART_TX_DATA;
 74      assign IN_UART_TX_LAUNCH=REG_UART_TX_LAUNCH;
 75      initial begin
 76         OUT_TX_ACTIVE=0;
 77         OUT_TX_DONE=0;
 78         REG_TX_FSM_STATE=STATE_WAIT;
 79         REG_RX_PACK_COUNT=0;
 80         REG_TX_PACK_COUNT=0;
 81         FIRST_CLOCK_AFTER_WRITE_PACK=0;
 82         OUT_RX_NUM_OF_DATA_PACKS_READY=0;
 83         OUT_RX_ERROR=0;
 84         OUT_RX_DATA_MASSIV=0;
 85      end
 86      always @(posedge IN_CLOCK)
 87      begin
 88         case(REG_TX_FSM_STATE)
 89            STATE_WAIT:
 90            begin
 91               if(FIRST_CLOCK_AFTER_WRITE_PACK)
 92               begin
 93                  FIRST_CLOCK_AFTER_WRITE_PACK<=0;
 94                  OUT_TX_DONE<=1;
 95               end
 96               else
 97                  OUT_TX_DONE<=0;
 98               OUT_TX_ACTIVE<=0;
 99               REG_TX_PACK_COUNT<=0;
100               if(IN_TX_LAUNCH&&IN_TX_NUMBER_OF_PACKS_TO_SEND!=0)
101               begin
102                  REG_TX_FSM_STATE<=STATE_WRITE_PACKS;
103                  OUT_TX_ACTIVE<=1;
104                  REG_TX_DATA_MASSIV=IN_TX_DATA_MASSIV;
105                  if(IN_TX_NUMBER_OF_PACKS_TO_SEND>TX_MASSIV_DEEP)
106                     REG_TX_NUMBER_OF_PACKS_TO_SEND=TX_MASSIV_DEEP;
107                  else
108                     REG_TX_NUMBER_OF_PACKS_TO_SEND<=IN_TX_NUMBER_OF_PACKS_TO_SEND;
109                  REG_UART_TX_DATA=IN_TX_DATA_MASSIV[NUM_OF_DATA_BITS_IN_PACK-1:0];
110                  REG_TX_PACK_COUNT<=0;
111               end
112            end
113            STATE_WRITE_PACKS:
114            begin
115               if (REG_TX_PACK_COUNT==0)
116               begin
117                  REG_UART_TX_LAUNCH<=1;
118                  REG_TX_PACK_COUNT=1;
119               end
120               if(OUT_UART_TX_START_BIT_ACTIVE)
121                  REG_UART_TX_LAUNCH<=0;
122               if(REG_TX_PACK_COUNT<=REG_TX_NUMBER_OF_PACKS_TO_SEND)
123                  begin
124                     if(OUT_UART_TX_STOP_BIT_ACTIVE)
125                     begin
126                           REG_UART_TX_DATA=sel_part_vector(REG_TX_DATA_MASSIV,
    REG_TX_PACK_COUNT);
127                           REG_TX_FSM_STATE<=STATE_WAIT_UART_DONE;
128                     end
129                  end
130                  else
131                  begin
132                     REG_TX_FSM_STATE<=STATE_WAIT;
133                     FIRST_CLOCK_AFTER_WRITE_PACK<=1;
134                     REG_UART_TX_LAUNCH<=0;
135                  end
136            end
137            STATE_WAIT_UART_DONE:
138            begin
139               if(!OUT_UART_TX_ACTIVE)
140               begin
141                  if(REG_TX_NUMBER_OF_PACKS_TO_SEND!=REG_TX_PACK_COUNT)
142                     REG_UART_TX_LAUNCH<=1;
143                  REG_TX_PACK_COUNT<=REG_TX_PACK_COUNT+1;
144                  REG_TX_FSM_STATE<=STATE_WRITE_PACKS;
145               end
146            end
147         endcase
148      end
149      always@(posedge OUT_UART_RX_DATA_READY or posedge IN_RX_CLEAR_BUFFER)
150      begin
```

```
151         if(IN_RX_CLEAR_BUFFER)
152         begin
153            OUT_RX_NUM_OF_DATA_PACKS_READY<=0;
154            OUT_RX_ERROR<=0;
155            OUT_RX_DATA_MASSIV<=0;
156         end
157         else
158         begin
159            if (OUT_RX_NUM_OF_DATA_PACKS_READY<RX_MASSIV_DEEP)
160               OUT_RX_NUM_OF_DATA_PACKS_READY=OUT_RX_NUM_OF_DATA_PACKS_READY+1;
161            else
162               OUT_RX_NUM_OF_DATA_PACKS_READY=1;//для избежания переполнения буфера
163            OUT_RX_DATA_MASSIV=ins_pack_in_vector(OUT_RX_DATA_MASSIV,OUT_UART_RX_DATA,
    OUT_RX_NUM_OF_DATA_PACKS_READY-1);
164            OUT_RX_ERROR=OUT_UART_RX_ERROR|OUT_RX_ERROR;
165         end
166      end
167      //функция- сепаратор пакета из вектора
168      function [NUM_OF_DATA_BITS_IN_PACK-1:0] sel_part_vector;
169         input [NUM_OF_DATA_BITS_IN_PACK*TX_MASSIV_DEEP-1:0] vector;
170         input [TX_MASSIV_DEEP_LOG_2:0] index;
171         reg [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] i;
172         reg [NUM_OF_DATA_BITS_IN_PACK-1:0] buffer;
173         begin
174            for(i=0;i<NUM_OF_DATA_BITS_IN_PACK;i=i+1)
175               buffer[i]=vector[i+index*NUM_OF_DATA_BITS_IN_PACK];
176            sel_part_vector=buffer;
177         end
178      endfunction
179      //функция- интегратор пакета в вектор
180      function [NUM_OF_DATA_BITS_IN_PACK*RX_MASSIV_DEEP-1:0] ins_pack_in_vector;
181         input [NUM_OF_DATA_BITS_IN_PACK*RX_MASSIV_DEEP-1:0] vector;
182         input [NUM_OF_DATA_BITS_IN_PACK-1:0] pack;
183         input [TX_MASSIV_DEEP_LOG_2:0] index;
184         reg [NUM_OF_DATA_BITS_IN_PACK*RX_MASSIV_DEEP-1:0] vector_buf;
185         reg [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] i;
186         begin
187            vector_buf=vector;
188            for(i=0;i<NUM_OF_DATA_BITS_IN_PACK;i=i+1)
189               vector_buf[i+index*NUM_OF_DATA_BITS_IN_PACK]=pack[i];
190            ins_pack_in_vector=vector_buf;
191         end
192      endfunction
193   endmodule
194
```