```
module UART_FPGA_RX
  #(
  parameter UART_BAUD_RATE                    =9600,
  //baud
  parameter CLOCK_FREQUENCY                   =50000000,
  //frequency IN_CLOCK
  parameter PARITY                            =1,
  //parameter of parity bit in package
  //PARITY==0  :  package without parity bit
  //PARITY==1  :  package contains parity bit
  //PARITY==2  :  package contains odd bit
  parameter NUM_OF_DATA_BITS_IN_PACK          =8,//number of data bits in package
  parameter CLKS_PER_BIT_LOG_2                =$clog2(CLOCK_FREQUENCY/UART_BAUD_RATE),
  //the number of bits for the register of the main counter
  parameter NUM_OF_DATA_BITS_IN_PACK_LOG_2    =$clog2(NUM_OF_DATA_BITS_IN_PACK)
  //the number of bits for the register of bit counter
  )
  (
   input                                      IN_CLOCK,          //input clock
   input                                      IN_RX_SERIAL,      //UART RX port
   output reg                                 OUT_RX_DATA_READY,//set brifely when a data package is received
   output reg   [NUM_OF_DATA_BITS_IN_PACK-1:0] OUT_RX_DATA,      //received data package
   output reg                                 OUT_RX_ERROR       //read error indicator
   );
  localparam CLKS_PER_BIT       = CLOCK_FREQUENCY/UART_BAUD_RATE ;
  //the number of IN_CLOCK cycles of the main generator
  //for the transmission of one data bit

  //finit state machine
  localparam STATE_WAIT         = 3'b000;//state wait start bit
  //состояние ожидания старт-бита на линии
  localparam STATE_RX_START_BIT = 3'b001;//state wait half start bit to chack bus status again
  //состояние приема старт-бита
  localparam STATE_RX_DATA_BITS = 3'b010;//state package read
  //состояние считывания информационных битов
  localparam STATE_RX_STOP_BIT  = 3'b011;//state wait stop bit
  //состояние считывания стоп-бита
  localparam STATE_RX_PARITY_BIT= 3'b100;//state wait and check parity bit
  //состояние считывания бита четности/нечетности
  //internal registers//внутренние регистры

  reg [CLKS_PER_BIT_LOG_2:0]                  REG_CLOCK_COUNT;  //main counter
  reg [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0]      REG_BIT_INDEX;    //bit index counter
  reg [2:0]                                   REG_STATE;        //this register contains FSM state
  //определение начальных начальных значений регистров
  initial begin
      //initial output registers
      OUT_RX_DATA_READY                       =0;
      OUT_RX_DATA                             =0;
      OUT_RX_ERROR                            =0;
      //initial internal registers
      REG_CLOCK_COUNT                         =0;
      REG_BIT_INDEX                           =0;
      REG_STATE                               =STATE_WAIT;
  end

  always @(posedge IN_CLOCK)
  begin
      case (REG_STATE)
         STATE_WAIT:
         begin
            OUT_RX_DATA_READY           <= 1'b0;
            OUT_RX_ERROR                <= 0;
            if (IN_RX_SERIAL == 1'b0)          // start bit detected//обнаружен старт-бит
               REG_STATE <= STATE_RX_START_BIT;
         end
         STATE_RX_START_BIT :
         begin
            if (REG_CLOCK_COUNT == CLKS_PER_BIT/2-2)
            /*
            Повторное считывание состояние линии для
            проверки действительности начала передачи.
            Если на линии все еще низкий уровень сигнала, то
            приемник продолжает работать в стандартном режиме,
            тогда автомат переходит в состояние чтения информационных бит.
            Если на линии высокий сигнал, то оъявляется ошибка инициализации
```

```
77                транзакции. Приемник переходит в состояние ожидания.
78                */
79                begin
80                   if (IN_RX_SERIAL == 1'b0)
81                      begin
82                         REG_CLOCK_COUNT <= 0;
83                         REG_STATE       <= STATE_RX_DATA_BITS;
84                      end
85                   else
86                         REG_STATE       <= STATE_WAIT;
87                end
88                else REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
89             end
90             STATE_RX_DATA_BITS://состояние считывания информационных бит
91             begin
92                if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)//классический счетчик
93                 REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
94                else
95                begin
96                   REG_CLOCK_COUNT             <= 0;
97                   OUT_RX_DATA[REG_BIT_INDEX] <= IN_RX_SERIAL;//считывается очередной бит
98                   if (REG_BIT_INDEX < NUM_OF_DATA_BITS_IN_PACK-1)
99                      REG_BIT_INDEX <= REG_BIT_INDEX + 1;
100                  else
101                  begin
102                     REG_BIT_INDEX <= 0;
103                     if(PARITY!=0)//следующие состояние зависит от параметра четности
104                        REG_STATE   <= STATE_RX_PARITY_BIT;
105                     else
106                        REG_STATE   <= STATE_RX_STOP_BIT;
107                  end
108               end
109            end
110            STATE_RX_PARITY_BIT://чтение бита четности
111            begin
112               if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)//классический счетчик
113                  REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
114               else
115               begin
116                  REG_CLOCK_COUNT <= 0;
117                  REG_STATE   <= STATE_RX_STOP_BIT;
118                  case(PARITY)   //проверка правильности приема пакета по биту четности
119                     1:OUT_RX_ERROR<=((sum_of_bits(OUT_RX_DATA)+IN_RX_SERIAL)%2==0) ?0:1;
    //если последний бит-бит четности
120                     2:OUT_RX_ERROR<=((sum_of_bits(OUT_RX_DATA)+IN_RX_SERIAL)%2==0)?1:0;
    //если последний бит-бит нечетности
121                  endcase
122               end
123            end
124            STATE_RX_STOP_BIT://состояние считывания стоп-бита
125            begin
126               if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)
127                  REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
128               else
129               begin
130                  if(IN_RX_SERIAL)
131                  begin
132                     if(!OUT_RX_ERROR)
133                        OUT_RX_DATA_READY    <= 1'b1;
134                     REG_CLOCK_COUNT         <= 0;//обнуление счетчика
135                     REG_STATE               <= STATE_WAIT;
136                  end
137                  else
138                  begin
139                     OUT_RX_ERROR            <=1;//если на линии не высокий сигнал, то ошибка
140                     REG_STATE               <=STATE_WAIT;//автомат переходит в состояние
    ожидания
141                     REG_CLOCK_COUNT         <=0;//обнуляется счетчик
142                  end
143               end
144            end
145            default :
146               REG_STATE <= STATE_WAIT;//при аномальном значении REG_STATE переход в
    состояние ожидания
147         endcase
148      end
149      function [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] sum_of_bits;//функция суммирует биты в
    регистре
150      //this function sums the bits in a register
```

```
151         input [NUM_OF_DATA_BITS_IN_PACK-1:0] value;
152         reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] sum;
153         reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] i;
154         begin
155            sum=0;
156            for (i=0;i<NUM_OF_DATA_BITS_IN_PACK;i=i+1)
157               sum=sum+value[i];
158            sum_of_bits=sum;
159         end
160      endfunction
161   endmodule
```