# 实验四　综合电路模块设计

实验日期：2020/6/6
实验人员：（杨泽群/18374192/180323）
实验室及桌号：28
任课教师：彭朝琴

## 一、任务目的

1. 熟练掌握 FPGA 开发流程及调试方式方法。
2. 加强锻炼综合化、智能化、信息化的功能模块设计。
3. 学习并掌握异步系统设计。
4. 全面掌握 FPGA 开发板硬件资源的使用方法。

## 二、任务功能描述及分析

### 1. 实验内容

开发 RS232 总线接口，实现 UART 通信协议。初步设计参数为波特率固定、8 位数据位、无奇偶校验、1 位停止位。（选做）串口参数具有波特率、数据位数、奇偶校验、停止位均可配置功能。

本实验对于波特率，奇偶校验均可配置。

### 2. 实验要求

（1）描述综合电路模块功能，分析并划分电路子模块，罗列各个模块输入输出信号关系及功能定义，设计各个模块输入输出操作流程。

（2）设计代码实现，分模块进行功能仿真，以及最后的总模块仿真。

（3）输入输出选择对应硬件接口，绑定引脚，下载程序验证设计是否成功。

## 三、功能模型的程序流程图

### 设计思想：

#### 1 串行接口 RS232 工作原理

串口用来连接 FPGA 和 PC 机，RS-232 允许全双工通信，即计算机在接收数据的同时可以发送数据。串口按位（bit）发送和接收字节。通常以 8 位数据为 1 组，先发送最低有效位，最后发送最高有效位。尽管比按字节（byte）的并行通信慢，但是串口可以在使用一根线发送数据的同时用另一根线接收数据。

通信使用 3 根线完成：（1）地线，（2）发送，（3）接收。由于串口通信是异步的，端口能够在一根线上发送数据同时在另一根线上接收数据。其他线用于握手，但不是必须的。数据的传输没有时钟信号，接收端必须采取某种方式，使之与接收数据同步。

1）串行线缆的两端先约定好串行传输的参数（传输速度、传输格式等）；

2）当没有数据传输的时候，发送端向数据线上发送"1"；

3）每传输一个字节之前，发送端先发送一个"0"来表示传输已经开始，这样接收端便可以知道有数据到来了；

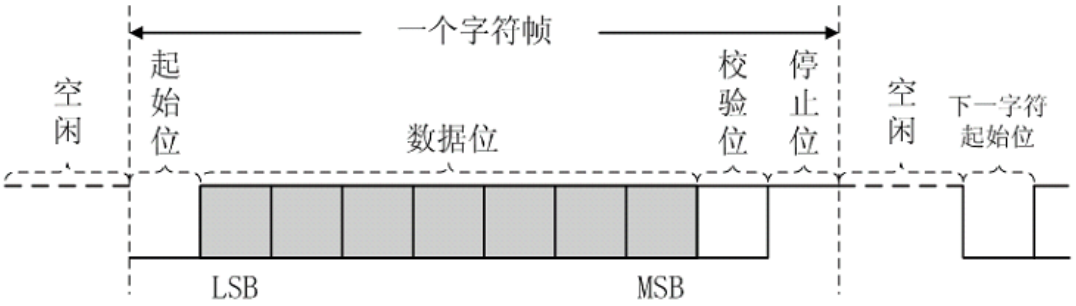



图 1 数据帧结构

4）开始传输后，数据以约定的速度和格式传输，所以接收端可以与之同步；

5）在串口总线上‘高电平’是默认的状态，当一帧数据开始传输必须先拉低电平，这就是起始位，起始位之后是 8 位数据位，最后是校验位和停止位（可不加校验位）。传输完成一个字节之后，都在其后发送一个停止位("1")。

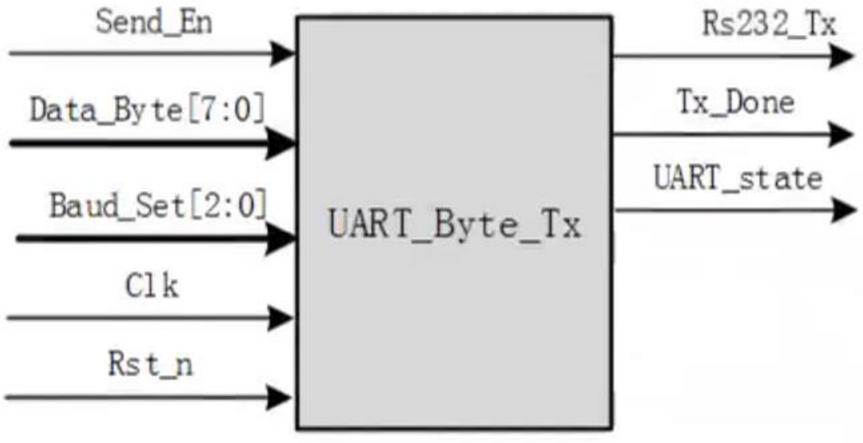
**2 波特率发生器**

波特率是串口传输的传输速度；在微观上就是一个位的周期。常用的波特率有 9600bps 和 115200bps。“9600bps”表示每秒可以传输 9600 位。本次实验我所选用的传输速率为 9600bps。由于我们的 FPGA 通常运行在远高于 9600Hz 的频率上（100MHz），因此需要分频产生接近 9600Hz 的时钟信号。若 FPGA 时钟为 50MHz，则需要$\frac{50M}{9600}=5207$个时钟周期置位一次就可以得到$9600Hz$的时钟。

## 模块设计

### 发送模块

其串口模块示意图如下，其中 Send_En 为使能端，其输入正脉冲即可使编码开始工作；Data_Byte 是输入八位的并行数据，是编码的主要成分；Baud_set 是波特率选择输入，在本题中给出了 5 种波特率的调整，分别为 9600，19200，38400，76800，153600，输入为 0-4 即可获得相应的波特率。Clk 信号为时钟信号，Rst_n 是输入复位信号。输出有三个，Rs232_Tx 是输出的数据；Tx_Done 高电平表示完成了一次编码输出；UART_state 表示其是否在工作，工作时输出高电平，其余时间输出低电平。




其发送字节时序图如下。

图 2 串口模块结构体

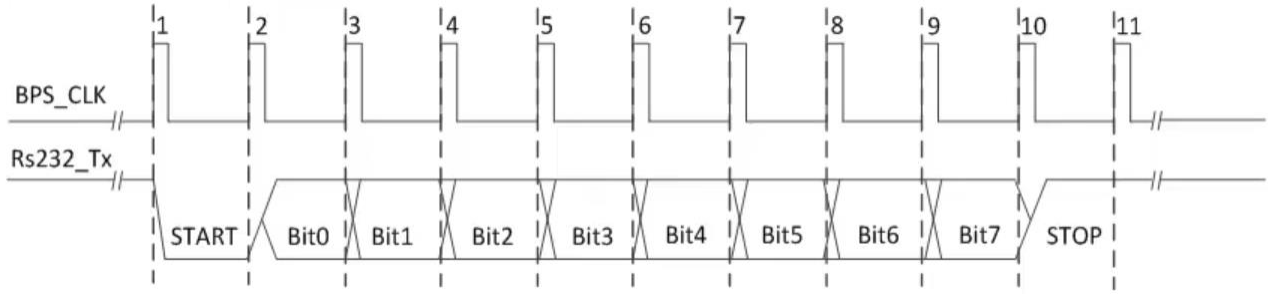



图 3 UART 发送字节时序图

其第一个字节为起始位，输出为低电平；第 2-9 个字节为 8 位数据位；第 10 个字节为停止位，11 个字节上升沿即为该模块的结束。在这里我在第 10 个位置插入了奇偶校验位，使得其可以起到检验输出的作用。

其流程如下：首先对于整个模块进行初始化

```
always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
                uart_state <= 1'b0;
        else if(send_en)
                uart_state <= 1'b1;
        else if(Tx_Done)
                uart_state <= 1'b0;
        else
                uart_state <= uart_state;

always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
                r_data_byte <= 8'd0;
        else if(send_en)
                begin
                        r_data_byte <= data_byte;
                        Check <= data_byte[0]+data_byte[1]+data_byte[2]+data_byte[3]+data_byte[4]+data_byte[5]+data_byte[6]+data_byte[7];
                end
        else
                begin
                        r_data_byte <= r_data_byte;
                        Check <= Check;
                end
```

可以看出，在 send_en 为 1 时数据传入 r_data_byte，并且将奇偶校验位填充，等待下一步波特率的生成和进一步处理。

接下来是波特率产生模块：

```
                end
always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
                bps_DR <= 16'd5207;
        else
                begin
                        case(baud_set)
                                0:bps_DR <= 16'd5207;
                                1:bps_DR <= 16'd2603;
                                2:bps_DR <= 16'd1301;
                                3:bps_DR <= 16'd867;
                                4:bps_DR <= 16'd433;
                                default:bps_DR <= 16'd5207;
                        endcase
                end

//counter
always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
                div_cnt <= 16'd0;
        else if(uart_state)
                begin
                        if(div_cnt == bps_DR)                   //从0记到波特率
                                div_cnt <= 16'd0;
                        else
                                div_cnt <= div_cnt + 1'b1;
                end
        else
                div_cnt <= 16'd0;

// bps_clk gen
always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
                bps_clk <= 1'b0;
        else if(div_cnt == 16'd1)
                bps_clk <= 1'b1;
        else
                bps_clk <= 1'b0;
```

先通过输入 baud_set 设置波特率，然后用 16 位寄存器开始计数，每当记录到波特率时清零；而波特率时钟当计数器记到 1 时自动加一，即可以实验波特率时钟计时功能。

下面是波特率计数和数据输出模块。

```
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
        bps_cnt <= 4'd0;
else if(Tx_Done)                        //记到结束清零
        bps_cnt <= 4'd0;
else if(bps_clk)
        bps_cnt <= bps_cnt + 1'b1;
else
        bps_cnt <= bps_cnt;

always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
        Tx_Done <= 1'b0;
else if(bps_cnt == 4'd12)
        Tx_Done <= 1'b1;
else
        Tx_Done <= 1'b0;
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
        Rs232_Tx <= 1'b1;
else begin
        case(bps_cnt)
                0:Rs232_Tx <= 1'b1;
                1:Rs232_Tx <= START_BIT;
                2:Rs232_Tx <= r_data_byte[0];
                3:Rs232_Tx <= r_data_byte[1];
                4:Rs232_Tx <= r_data_byte[2];
                5:Rs232_Tx <= r_data_byte[3];
                6:Rs232_Tx <= r_data_byte[4];
                7:Rs232_Tx <= r_data_byte[5];
                8:Rs232_Tx <= r_data_byte[6];
                9:Rs232_Tx <= r_data_byte[7];
                10:Rs232_Tx <= Check[0];
                11:Rs232_Tx <= STOP_BIT;
                default:Rs232_Tx <= 1'b1;
        endcase
end
```

首先是个波特率计数器，一个计数时间输出一个数据。当输出第 12 个数据时即停止，Tx_Done 输出 1。而下面是一个对于波特率时钟的 case 语句，当它为 1-11 的每个值时对应着相应的输出，即实现了波特率编码的功能。

**接收模块**

串口接收模块结构体如下。

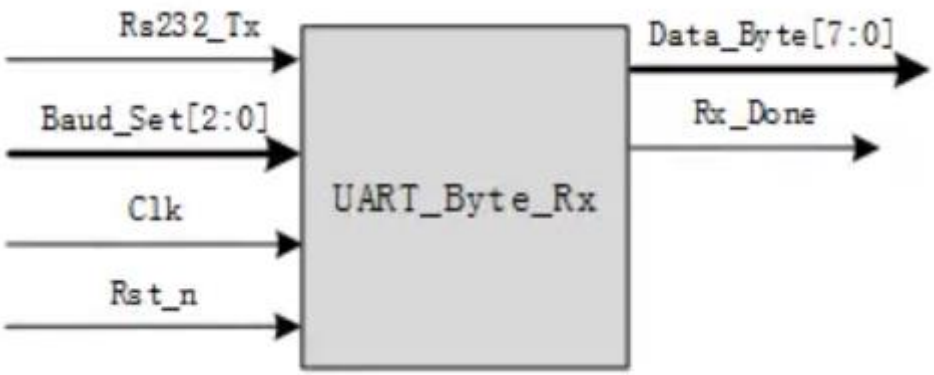


图 5 串口接收模块结构体

其中，Rs232_Tx 是发送模块输出的数据，其余三个含义与发送模块相同。输出 Data_Byte 是接收到的数据，理论上应和输入数据相同；Rx_Done 是标记接收有没有完成的端口。

在这里信号接收时，通常取一个波特率周期内中间的点。在这里我将 Bit_x 平均划分成 16 分，取得中间 6 份进行加权平均，这样可以得到更稳定的数据。其大致实现见图 6。

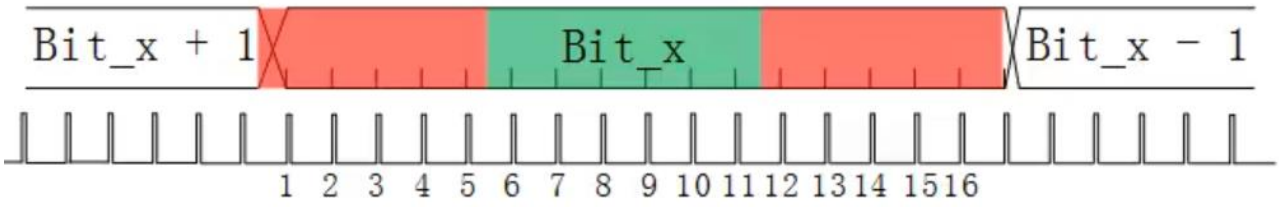



图 6 数据接收取样

其流程如下：

```
//同步寄存器，消除亚稳态
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)begin
        s0_Rs232_Rx <= 1'b0;
        s1_Rs232_Rx <= 1'b0;
end
else begin
        s0_Rs232_Rx <= Rs232_Rx;
        s1_Rs232_Rx <= s0_Rs232_Rx;
end

//数据寄存器
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)begin
        tmp0_Rs232_Rx <= 1'b0;
        tmp1_Rs232_Rx <= 1'b0;
end
else begin
        tmp0_Rs232_Rx <= s1_Rs232_Rx;
        tmp1_Rs232_Rx <= tmp0_Rs232_Rx;
end

assign nedege = !tmp0_Rs232_Rx & tmp1_Rs232_Rx;
```

首先进行初始化，存下输入数据，并用 tem 的临时存储，利用此时的数据和上一时刻的数据确定串行输入数据的上升沿。

接下来是波特率计数模块，与发送端情况类似，不再赘述。此时波特率为发送端波特率除以 16，为把一个波特率时钟分为 16 份取样。

```
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
        bps_DR <= 16'd324;
else begin
        case(baud_set)
                0:bps_DR <= 16'd324;
                1:bps_DR <= 16'd162;
                2:bps_DR <= 16'd80;
                3:bps_DR <= 16'd53;
                4:bps_DR <= 16'd26;
                default:bps_DR <= 16'd324;
        endcase
end

//counter
always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
                div_cnt <= 16'd0;
        else
        if(uart_state)
                begin
                        if(div_cnt == bps_DR)                   //到达波特率
                                div_cnt <= 16'd0;
                        else
                                div_cnt <= div_cnt + 1'b1;
                end
        else
                div_cnt <= 16'd0;

// bps_clk gen
always@(posedge Clk or negedge Rst_n)           //bps_clk时钟
        if(!Rst_n)
                bps_clk <= 1'b0;
        else if(div_cnt == 16'd1)
                bps_clk <= 1'b1;
        else
                bps_clk <= 1'b0;

//bps counter
always@(posedge Clk or negedge Rst_n)                   //波特率计数
if(!Rst_n)
        bps_cnt <= 8'd0;
else if(bps_cnt == 8'd175 | (bps_cnt == 8'd12 && (START_BIT > 2)))
        bps_cnt <= 8'd0;
else if(bps_clk)
        bps_cnt <= bps_cnt + 1'b1;
else
        bps_cnt <= bps_cnt;

always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
                Rx_Done <= 1'b0;
        else if(bps_cnt == 8'd175)
                Rx_Done <= 1'b1;
        else
                Rx_Done <= 1'b0;
```

当波特率时钟不为 0 时，这里使用 case 语句，对于 bps_cnt 的计数情况进行了分类，较好的达到了将数据存在对应地方的功能。

```
else if(bps_clk)begin
        case(bps_cnt)
                0:begin
                        START_BIT = 3'd0;
                        r_data_byte[0] <= 3'd0;
                        r_data_byte[1] <= 3'd0;
                        r_data_byte[2] <= 3'd0;
                        r_data_byte[3] <= 3'd0;
                        r_data_byte[4] <= 3'd0;
                        r_data_byte[5] <= 3'd0;
                        r_data_byte[6] <= 3'd0;
                        r_data_byte[7] <= 3'd0;
                        STOP_BIT = 3'd0;
                end
            6,7,8,9,10,11:START_BIT <= START_BIT + s1_Rs232_Rx;
            22,23,24,25,26,27:r_data_byte[0] <= r_data_byte[0] + s1_Rs232_Rx;
            38,39,40,41,42,43:r_data_byte[1] <= r_data_byte[1] + s1_Rs232_Rx;
            54,55,56,57,58,59:r_data_byte[2] <= r_data_byte[2] + s1_Rs232_Rx;
            70,71,72,73,74,75:r_data_byte[3] <= r_data_byte[3] + s1_Rs232_Rx;
            86,87,88,89,90,91:r_data_byte[4] <= r_data_byte[4] + s1_Rs232_Rx;
            102,103,104,105,106,107:r_data_byte[5] <= r_data_byte[5] + s1_Rs232
            118,119,120,121,122,123:r_data_byte[6] <= r_data_byte[6] + s1_Rs232
            134,135,136,137,138,139:r_data_byte[7] <= r_data_byte[7] + s1_Rs232
            150,151,152,153,154,155:Check <= Check + s1_Rs232_Rx;
            166,167,168,169,170,171:STOP_BIT <= STOP_BIT + s1_Rs232_Rx;
            default:
                    begin
                        START_BIT = START_BIT;
                        r_data_byte[0] <= r_data_byte[0];
                        r_data_byte[1] <= r_data_byte[1];
                        r_data_byte[2] <= r_data_byte[2];
                        r_data_byte[3] <= r_data_byte[3];
                        r_data_byte[4] <= r_data_byte[4];
                        r_data_byte[5] <= r_data_byte[5];
                        r_data_byte[6] <= r_data_byte[6];
                        r_data_byte[7] <= r_data_byte[7];
                        STOP_BIT = STOP_BIT;
                    end
        endcase
```

然后对于计算出的每个 r_data_byte 数据，可以认为其第二位(>=4)值就是其计算出的平均值。这样就完成了程序的设计。

```
always@(posedge Clk or negedge Rst_n)        //取>=4的位LOL
        if(!Rst_n)
                data_byte <= 8'd0;
        else if(bps_cnt == 8'd175)
        begin
                data_byte[0] <= r_data_byte[0][2];
                data_byte[1] <= r_data_byte[1][2];
                data_byte[2] <= r_data_byte[2][2];
                data_byte[3] <= r_data_byte[3][2];
                data_byte[4] <= r_data_byte[4][2];
                data_byte[5] <= r_data_byte[5][2];
                data_byte[6] <= r_data_byte[6][2];
                data_byte[7] <= r_data_byte[7][2];
                check1 <= Check[0];
        end
```

## 四、任务难点、创新点等学习总结

本次实验难度较大，对于串口的理解至关重要。在理解了 Rs232 协议能等内容后，程序就较为容易了。其中对于模块的端口的设置至关重要，其收发的结构基本一致，思路也很清楚，就是对于并行数据跟随波特率转化为串行数据的过程，再由串行数据译码出并行数据。

此任务的创新点在于加了波特率可选的输入和奇偶校验位，可以看出其效果较好，可以满足使用要求。

## 五、程序源代码及仿真波形

### 发送模块

```
module uart_byte_tx(
Clk,
Rst_n,
data_byte,
send_en,
baud_set,

Rs232_Tx,
Tx_Done,
uart_state
);

input Clk;
input Rst_n;
input [7:0]data_byte;
input send_en;
input [2:0]baud_set;

output reg Rs232_Tx;
output reg Tx_Done;
output reg uart_state;                                  //状态:1 为工作状态

reg bps_clk;                                            //波特率时钟

reg [15:0]div_cnt;//分频计数器

reg [15:0]bps_DR;//分频计数最大值

reg [3:0]bps_cnt;//波特率时钟计数器

reg [7:0]r_data_byte;

reg [2:0]Check;
localparam START_BIT = 1'b0;                            //输出，开始时上低电平
localparam STOP_BIT = 1'b1;                             //输出，
```

结束时上高电平

```
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
uart_state <= 1'b0;
else if(send_en)
uart_state <= 1'b1;
else if(bps_cnt == 4'd11)
uart_state <= 1'b0;
else
uart_state <= uart_state;

always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
r_data_byte <= 8'd0;
else if(send_en)
r_data_byte <= data_byte;
else
r_data_byte <= r_data_byte;

always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
bps_DR <= 16'd5207;
else
begin
case(baud_set)
0:bps_DR <= 16'd5207;
1:bps_DR <= 16'd2603;
2:bps_DR <= 16'd1301;
3:bps_DR <= 16'd867;
4:bps_DR <= 16'd433;
default:bps_DR <= 16'd5207;
endcase
end

//counter
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
div_cnt <= 16'd0;
else if(uart_state)
begin
if(div_cnt == bps_DR)                                                    //从
0 记到波特率
div_cnt <= 16'd0;
else
div_cnt <= div_cnt + 1'b1;
end
```

```
        else
        div_cnt <= 16'd0;

        // bps_clk gen
        always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
        bps_clk <= 1'b0;
        else if(div_cnt == 16'd1)
        bps_clk <= 1'b1;
        else
        bps_clk <= 1'b0;

        //bps counter
        always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
        bps_cnt <= 4'd0;
        else if(bps_cnt == 4'd12)                                                    //记
到结束清零
        bps_cnt <= 4'd0;
        else if(bps_clk)
        bps_cnt <= bps_cnt + 1'b1;
        else
        bps_cnt <= bps_cnt;

        always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
        Tx_Done <= 1'b0;
        else if(bps_cnt == 4'd11)
        Tx_Done <= 1'b1;
        else
        Tx_Done <= 1'b0;
        always Check <=
r_data_byte[0]+r_data_byte[1]+r_data_byte[2]+r_data_byte[3]+r_data_byte[4]+r_data_byte[5]+r_data_byte[6]+r
_data_byte[7];
        always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
        Rs232_Tx <= 1'b1;
        else begin
        case(bps_cnt)
        0:Rs232_Tx <= 1'b1;
        1:Rs232_Tx <= START_BIT;
        2:Rs232_Tx <= r_data_byte[0];
        3:Rs232_Tx <= r_data_byte[1];
        4:Rs232_Tx <= r_data_byte[2];
        5:Rs232_Tx <= r_data_byte[3];
        6:Rs232_Tx <= r_data_byte[4];
        7:Rs232_Tx <= r_data_byte[5];
```

```
        8:Rs232_Tx <= r_data_byte[6];
        9:Rs232_Tx <= r_data_byte[7];
        10:Rs232_Tx <= Check[0];
        11:Rs232_Tx <= STOP_BIT;
        default:Rs232_Tx <= 1'b1;
        endcase
        end

        endmodule
```

## 接收模块

```
module uart_byte_rx(
        Clk,
        Rst_n,
        baud_set,
        Rs232_Rx,

        data_byte,
        check1,
        Rx_Done
);

        input Clk;
        input Rst_n;
        input [2:0]baud_set;
        input Rs232_Rx;

        output reg [7:0]data_byte;
        output reg Rx_Done;
        output reg check1;

        reg s0_Rs232_Rx,s1_Rs232_Rx;//同步寄存器

        reg tmp0_Rs232_Rx,tmp1_Rs232_Rx;//数据寄存器

        reg [15:0]bps_DR;          //分频计数器计数最大值
        reg [15:0]div_cnt;   //分频计数器
        reg bps_clk;//
        reg [7:0]bps_cnt;

        reg [2:0]Check;
        reg uart_state;

        reg [2:0] r_data_byte [7:0];
        reg [7:0] tmp_data_byte;
        reg [2:0] START_BIT,STOP_BIT;
```

```
wire nedege;

//同步寄存器，消除亚稳态
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)begin
    s0_Rs232_Rx <= 1'b0;
    s1_Rs232_Rx <= 1'b0;
end
else begin
    s0_Rs232_Rx <= Rs232_Rx;
    s1_Rs232_Rx <= s0_Rs232_Rx;
end

//数据寄存器
always@(posedge Clk or negedge Rst_n)
if(!Rst_n)begin
    tmp0_Rs232_Rx <= 1'b0;
    tmp1_Rs232_Rx <= 1'b0;
end
else begin
    tmp0_Rs232_Rx <= s1_Rs232_Rx;
    tmp1_Rs232_Rx <= tmp0_Rs232_Rx;
end

assign nedege = !tmp0_Rs232_Rx & tmp1_Rs232_Rx;

always@(posedge Clk or negedge Rst_n)
if(!Rst_n)
    bps_DR <= 16'd324;
else begin
    case(baud_set)
        0:bps_DR <= 16'd324;
        1:bps_DR <= 16'd162;
        2:bps_DR <= 16'd80;
        3:bps_DR <= 16'd53;
        4:bps_DR <= 16'd26;
        default:bps_DR <= 16'd324;
    endcase
end

//counter
always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)
        div_cnt <= 16'd0;
    else
    if(uart_state)
        begin
```

```
                if(div_cnt == bps_DR)          //波特率计数
                    div_cnt <= 16'd0;
                else
                    div_cnt <= div_cnt + 1'b1;
            end
        else
            div_cnt <= 16'd0;

// bps_clk gen
always@(posedge Clk or negedge Rst_n)      //bps_clk 时钟
        if(!Rst_n)
            bps_clk <= 1'b0;
        else if(div_cnt == 16'd1)
            bps_clk <= 1'b1;
        else
            bps_clk <= 1'b0;

//bps counter
always@(posedge Clk or negedge Rst_n)          //波特率计数
if(!Rst_n)
        bps_cnt <= 8'd0;
else if(bps_cnt == 8'd175 | (bps_cnt == 8'd12 && (START_BIT > 2)))
        bps_cnt <= 8'd0;
else if(bps_clk)
        bps_cnt <= bps_cnt + 1'b1;
else
        bps_cnt <= bps_cnt;

always@(posedge Clk or negedge Rst_n)
        if(!Rst_n)
            Rx_Done <= 1'b0;
        else if(bps_cnt == 8'd175)
            Rx_Done <= 1'b1;
        else
            Rx_Done <= 1'b0;


always@(posedge Clk or negedge Rst_n)      //取>=4 的位 LOL
        if(!Rst_n)
            data_byte <= 8'd0;
        else if(bps_cnt == 8'd175)
        begin
            data_byte[0] <= r_data_byte[0][2];
            data_byte[1] <= r_data_byte[1][2];
            data_byte[2] <= r_data_byte[2][2];
            data_byte[3] <= r_data_byte[3][2];
            data_byte[4] <= r_data_byte[4][2];
```

```
            data_byte[5] <= r_data_byte[5][2];
            data_byte[6] <= r_data_byte[6][2];
            data_byte[7] <= r_data_byte[7][2];
            check1 <= Check[0];
        end

always@(posedge Clk or negedge Rst_n)
if(!Rst_n)begin
        START_BIT = 3'd0;
        r_data_byte[0] <= 3'd0;
        r_data_byte[1] <= 3'd0;
        r_data_byte[2] <= 3'd0;
        r_data_byte[3] <= 3'd0;
        r_data_byte[4] <= 3'd0;
        r_data_byte[5] <= 3'd0;
        r_data_byte[6] <= 3'd0;
        r_data_byte[7] <= 3'd0;
        Check <= 3'd0;
        STOP_BIT = 3'd0;
end
else if(bps_clk)begin
        case(bps_cnt)
            0:begin
                        START_BIT = 3'd0;
                        r_data_byte[0] <= 3'd0;
                        r_data_byte[1] <= 3'd0;
                        r_data_byte[2] <= 3'd0;
                        r_data_byte[3] <= 3'd0;
                        r_data_byte[4] <= 3'd0;
                        r_data_byte[5] <= 3'd0;
                        r_data_byte[6] <= 3'd0;
                        r_data_byte[7] <= 3'd0;
                        STOP_BIT = 3'd0;
                    end
            6,7,8,9,10,11:START_BIT <= START_BIT + s1_Rs232_Rx;
            22,23,24,25,26,27:r_data_byte[0] <= r_data_byte[0] + s1_Rs232_Rx;
            38,39,40,41,42,43:r_data_byte[1] <= r_data_byte[1] + s1_Rs232_Rx;
            54,55,56,57,58,59:r_data_byte[2] <= r_data_byte[2] + s1_Rs232_Rx;
            70,71,72,73,74,75:r_data_byte[3] <= r_data_byte[3] + s1_Rs232_Rx;
            86,87,88,89,90,91:r_data_byte[4] <= r_data_byte[4] + s1_Rs232_Rx;
            102,103,104,105,106,107:r_data_byte[5] <= r_data_byte[5] + s1_Rs232_Rx;
            118,119,120,121,122,123:r_data_byte[6] <= r_data_byte[6] + s1_Rs232_Rx;
            134,135,136,137,138,139:r_data_byte[7] <= r_data_byte[7] + s1_Rs232_Rx;
            150,151,152,153,154,155:Check <= Check + s1_Rs232_Rx;
            166,167,168,169,170,171:STOP_BIT <= STOP_BIT + s1_Rs232_Rx;
            default:
                begin
```

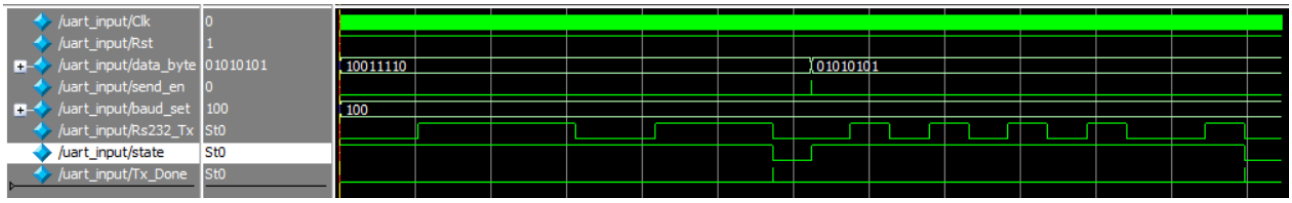
```
                        START_BIT = START_BIT;
                        r_data_byte[0] <= r_data_byte[0];
                        r_data_byte[1] <= r_data_byte[1];
                        r_data_byte[2] <= r_data_byte[2];
                        r_data_byte[3] <= r_data_byte[3];
                        r_data_byte[4] <= r_data_byte[4];
                        r_data_byte[5] <= r_data_byte[5];
                        r_data_byte[6] <= r_data_byte[6];
                        r_data_byte[7] <= r_data_byte[7];
                        STOP_BIT = STOP_BIT;
                    end
            endcase
        end

        always@(posedge Clk or negedge Rst_n)           //uart_state 判断
        if(!Rst_n)
            uart_state <= 1'b0;
        else if(nedege)
            uart_state <= 1'b1;
        else if(Rx_Done || (bps_cnt == 8'd12 && (START_BIT > 2)))
            uart_state <= 1'b0;
        else
            uart_state <= uart_state;

endmodule
```

顶层模块

```
module uart(
        Clk,        //50MHz
        Rst,        //复位信号
        data_byte,      //输入信号
        send_en,        //使能端,高脉冲启动
        baud_set,       //波特率设置端（9600/19200/...）

        Rs232_Tx,       //1 输出
        data_w,   //2 读入
        check1          //字符检查
);

input Clk;
input Rst;
input [7:0]data_byte;
input send_en;
input [2:0]baud_set;

output Rs232_Tx;
```

```
output [7:0]data_w;
output check1;

wire led;
wire Rx_Done;

    uart_byte_tx uart_byte_tx(
        .Clk(Clk),
        .Rst_n(Rst),
        .data_byte(data_byte),
        .send_en(send_en),
        .baud_set(3'd0),

        .Rs232_Tx(Rs232_Tx),
        .Tx_Done(),
        .uart_state(led)
    );


    uart_byte_rx uart_byte_rx(
        .Clk(Clk),
        .Rst_n(Rst),
        .baud_set(3'd0),
        .Rs232_Rx(Rs232_Tx),

        .data_byte(data_w),
        .check1(check1),
        .Rx_Done(Rx_Done)
    );
endmodule
```

波形仿真




第一个数据为 01111001，其奇偶校验为 1；
第二个数据为 10101010，其奇偶校验为 0。较好的实现了题设功能。