实验题目：串口收发器设计

班级：无 42

学号：2014011058

姓名：陈誉博

日期：2016 年 5 月 21 日

一、 实验目的

1．了解和掌握 UART 的工作原理；

二、 实验原理及设计思路

1. 串口基本原理

UART（Universal Asynchronous Receiver/Transmitter）是一种通用串行数据总线，用于异步通信。该总线双向通信，可以实现全双工传输和接收。在嵌入式设计中，UART用来与PC进行通信，包括与监控调试器和其它器件。与UART相关的一个概念是RS232-C标准，该标准由美国电子工业协会EIA(Electronic Industry Association)制定的一种串行物理接口标准，其规定了若干标准的数据速率，并且采用较高电平来保证20米以内的有线传输。

UART 是计算机与嵌入式系统中串行通信端口的关键部分，速率有规定的 9600 等波特率。在实际应用中，通用串口的电气特性兼容 RS232 规范信号，即逻辑“1”信号相对于地为-3 到-15 伏，而逻辑“0”相对于地为 3 到 15 伏。因此，当一个微控制器的 UART 与外界电路相连时，需要采用一个符合 RS232 标准的驱动器来将控制器管脚的 CMOS 电平或 TTL 电平转换为 RS232 标准电平。TTL 电平是 3.3V 的，而 RS232 是负逻辑电平，如果没有类似 MAX232 的驱动芯片进行电平转换，这么高的电压很可能会把芯片烧坏。

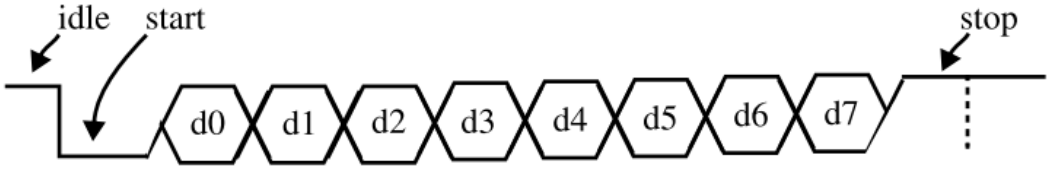


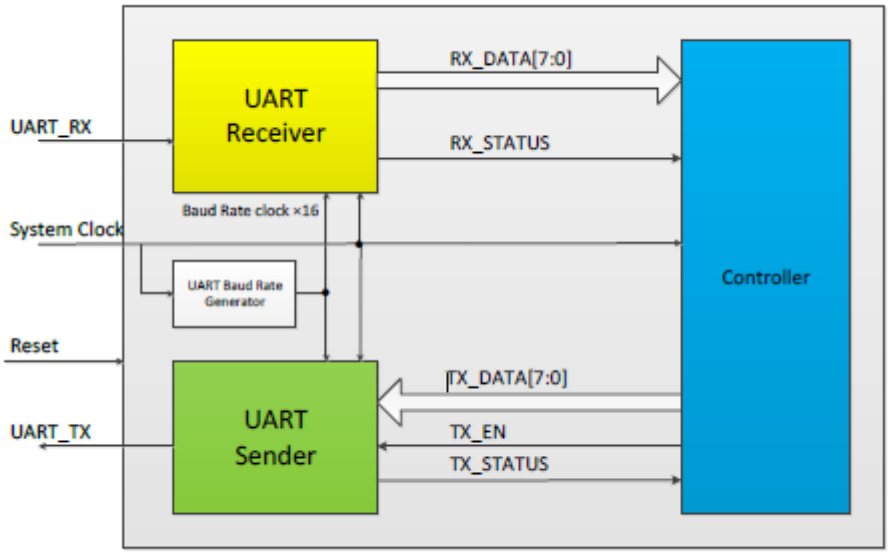
图 1 串口时序示意图

图1表明在异步传送中串行发送一个数据字节的位定时关系(图中没有包括奇偶校验位)。发送一个完整的字节信息，首先是一个作为起始位的逻辑 “0” 位，接着是8个数据位，然后是1个、1＋1/2个或2个停止位逻辑 “1” 位，数据线空闲时呈现为高或 “1” 状态。在字符的8位数据部分，先发送数据的最低位（LSB），最后发送最高位（MSB）。每位持续的时间是固定的，由发送器本地时钟控制，每秒发送的数据位个数，即为 “波特率” 。

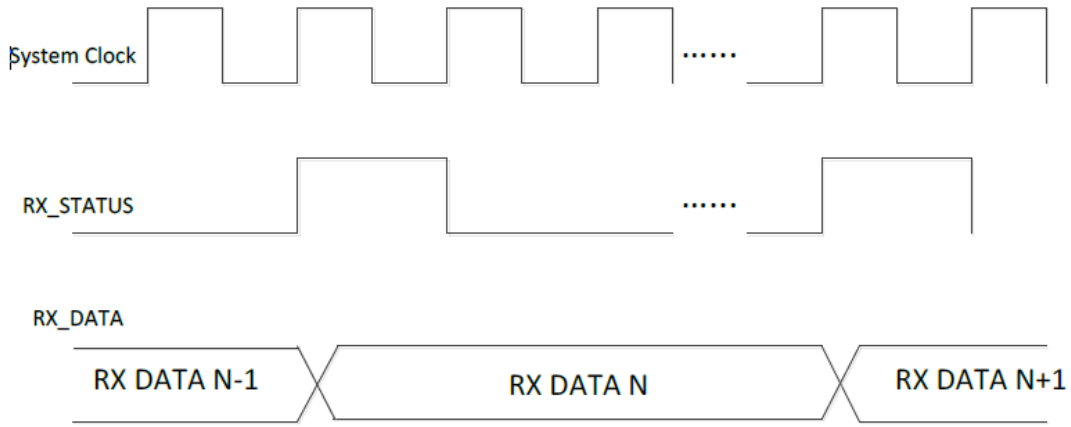
起始位和停止位起着很重要的作用。显然，他们标志每个字符的开始和结束，但更重要的是他们使接收器能把局部时钟与每个新开始接收的字符再同步。异步通信没有可参照的时钟信号，发送器随时都可能发送数据，需要从任何边沿的出现时刻开始正确地采样紧接着的10～11 位(包括开始位、数据位和停止位)。接收器的时钟与发送器的时钟不是同一个，因此，接收器采样点的间隔跟由发送器时钟所确定的位间隔时间不同，接收器设计不好可能会导致采样错误。

2. 实验设计原理

串口收发器包括发送器和接收器两个模块。首先，通过串口接收器模块从外部接收数据，并将接收到的数据送给控制器模块，同时控制器模块根据接收的串口数据产生发送数据，并通过串口发送器模块将数据发送到外部。

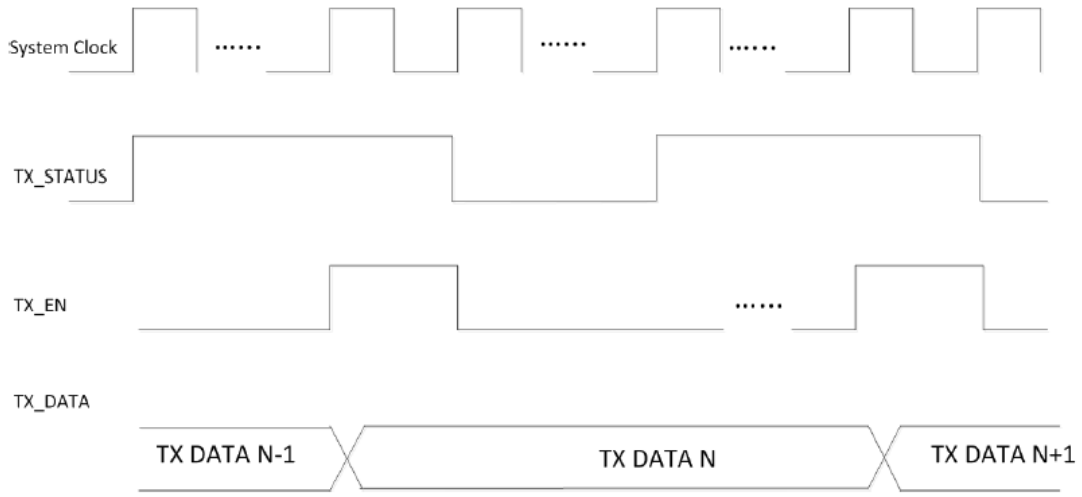


串口接收器（UART Receiver）模块负责从串口中接收串行数据流，并根据 UART 通讯协议提取接收到的数据并发送给控制器。每当串口接收器收到一个完整的数据，在 RX_STATUS 上输出一个高电平指示脉冲，并同时在 RX_DATA 上输出接收到的有效数据，RX_DATA 上的接收数据一直有效到下一个 RX_STATUS 脉冲位置。串口接收器与控制器之间的接口波形如此下图所示。



由于从线路上接收到的串行数据帧与接收模块的时钟是异步的，所以接收器功能实现中的关键是接收器时钟与每个接收字符的同步。一个有效的方法是接收器采用高速率时钟对串行数据进行采样，通常采样频率是位时钟频率的整数倍。理论上倍数越高接收数据各位的分辨率越高，实际中，一般最大选择16倍。波特率发生器(Baud Rate Generator)模块负责根据System clock时钟产生所需的16倍（或者其他倍数）波特率的接收时钟。

串口发送器（UART Sender）从控制器接收待发送数据，然后根据UART通讯协议串行发送出去。当控制器检测到TX_STATUS上出现高电平时，意味着此时串口发送器处于空闲状态可以接收一个新的发送数据，控制器在TX_DATA上输出待发送的数据，并同时在TX_EN上输出一个高电平脉冲，指示串口发送器启动一个新的数据发送。串口发送器与控制器之间的接口波形如下如所示。



3. 设计思路

1) 波特率发生器模块模块

产生 9600Hz 信号的基本思路是通过 100MHz 的时钟进行分频得到。由于需要波特率 16 倍的高速率采样信号，所以先从 100MHz 产生 9600*16=153600Hz 的信号，再由产生的信号进行 16 分频得到 9600Hz 的信号。

2) 接收器模块

为了使结果更加精确，使用系统时钟进行扫描。当检测到 UART_RX 信号中每一位数据的起始低电平时，将计数使能变为 1，开始计数。首先空置 24 个 153600Hz 的上升沿以保证之后的每次采样位于数据位的中心处，之后每 16 个上升沿进行一次采样，共采样八次并按照逆序将采集得到的数据存入 RX_DATA。当采集数据次数到达八次之后，计数使能变为 0，采集数据工作结束。

3) 控制器模块

输出按照实验要求，最高位为 1 时输出补码，最高位为 0 时透明传输。当检测到 RX_STATUS 信号为高电平时，说明此时有一数据被采集，此时检测 TX_STATUS 信号，如果该信号为高电平，则说明发送器模块空闲，则将 TX_EN 信号置为 1。经过约大于一个周期的时间（为了保证每次时钟的上升沿能够检测到 TX_EN 的高电平）后 TX_EN 变为 0.

4) 发送器模块

模块中设置了一个计数器，每当一个 9600Hz 时钟的上升沿来临，计数器加 1.TX_STATUS 在从 TX_EN 上升沿开始的第一个时钟周期之后就变为低电平，所以当计数器为 1 的时候将 TX_STATUS 置为 0；检查到 TX_EN 信号的高电平时，说明有一个数据从控制器发出，需要发送，此时使能信号置为 1，将 TX_DATA 的数据送到寄存器中。当 TX_EN 为低电平时，使能信号置为 0，将寄存器中的值按照计数器的顺序依次输出。当计数器的值为 11 时，数据发送完毕，计数器置 0.

三、 关键代码及源文件清单

1. UART 代码：UART2.v:

```
module UART_receiver(UART_RX,sysclk,RX_DATA,RX_STATUS);
input UART_RX;
input sysclk;
output [7:0]RX_DATA;
```

```
reg [7:0]RX_DATA;
output RX_STATUS;
reg RX_STATUS;

reg enable;
reg previous_enable;
reg [9:0]baud_rate_counter;
reg [4:0]time_center_counter;
reg [3:0]data_counter;
reg [4:0]status_counter;
reg [7:0]temp_data;
//set the first values
initial begin
    RX_DATA=8'b0;
    RX_STATUS=0;
    enable=0;
    previous_enable=0;
    baud_rate_counter=10'b0;
    time_center_counter=5'b0;
    data_counter=4'b0;
    temp_data=8'b0;
    status_counter=5'b0;
end
always @(posedge sysclk) begin
    previous_enable=enable;//record the status of the last clock
period
    if(enable==0) begin
        if(UART_RX==0) begin
            enable=1;//find the beginning 0 and set enable to 1
            RX_STATUS=0;
        end
    end
    if(enable==1&&previous_enable==0) begin
        baud_rate_counter=0;
    end
    else begin
        baud_rate_counter=baud_rate_counter+10'b00000_00001;
        if(baud_rate_counter==652)  begin  //find  a  posedge  of
153600Hz signal
            if(enable==1) begin
                time_center_counter=time_center_counter+5'b00001;
                if(time_center_counter==16) begin//find a posedge of
9600Hz
                    data_counter=data_counter+4'b0001;
```

```
                time_center_counter=0;
            end
            if(data_counter==9) begin
                data_counter=0;
                enable=0;
                RX_STATUS=1;
            end
            if(time_center_counter==8&&data_counter!=0) begin
                temp_data[data_counter-1]=UART_RX;
            end
        end
        if(RX_STATUS==1) begin
            status_counter=status_counter+5'b00001;
            if(status_counter==16) begin
                RX_STATUS=0;
                status_counter=0;
            end
        end
        baud_rate_counter=0;
    end
  end
end
always @(posedge sysclk) begin
  if(RX_STATUS==1) begin
    RX_DATA=temp_data;
  end
end
endmodule

module UART_sender(sample_clk,TX_DATA,TX_EN,TX_STATUS,UART_TX);
input sample_clk;
input [7:0]TX_DATA;
input TX_EN;
output TX_STATUS;
output UART_TX;
reg UART_TX;

reg enable;
reg [8:0]temp_data;
reg [3:0]counter;
initial begin
  UART_TX<=1;
  enable<=0;
  temp_data<=7'b0;
```

```
    counter<=4'b0;
end
assign TX_STATUS=counter==0?1:0;
always @(posedge sample_clk) begin
    if(TX_EN==1) begin
        enable=1;
    end
    if(enable==1&&TX_EN==0) begin
        temp_data[8]<=1;
        temp_data[7:0]<=TX_DATA;
        counter<=1;
        UART_TX<=0;
        enable<=0;
    end
    else begin
       if(counter==10)
           counter<=0;
       else if(counter!=0) begin
           if(counter<10) begin
               UART_TX=temp_data[counter-1];
           end
           counter<=counter+1;
       end
    end
end
endmodule

module
UART_control(RX_DATA,RX_STATUS,sysclk,TX_STATUS,TX_DATA,TX_EN);
input [7:0]RX_DATA;
input RX_STATUS;
input sysclk;
input TX_STATUS;
output [7:0]TX_DATA;
output TX_EN;
reg TX_EN;
reg receive_enable;//decide if the control mode have received a new
data
reg [13:0]counter;

assign TX_DATA[7]=RX_DATA[7];
assign TX_DATA[6:0]=RX_DATA[7]?(~RX_DATA[6:0]):RX_DATA[6:0];
initial begin
    TX_EN=0;
```

```
    receive_enable=0;
    counter=13'b0;
end
always @(posedge sysclk) begin
    if(RX_STATUS==1) begin
        receive_enable=1;
    end
    else if(TX_STATUS==1&&receive_enable==1) begin
        receive_enable=0;
        TX_EN=1;
    end
    if(TX_EN==1) begin
        counter=counter+13'b000000_0000001;
        if(counter==11000) begin
            counter=0;
            TX_EN=0;
        end
    end
end
endmodule

module Baud_Rate_Generator(sysclk,baud_rate_clk);
input sysclk;
output baud_rate_clk;//153600Hz signal
reg baud_rate_clk;
reg [9:0]counter;
initial begin
    counter=10'b0;
    baud_rate_clk=0;
end
always @(posedge sysclk) begin
    counter=counter+10'b00000_00001;
    if(counter==326) begin
        baud_rate_clk=~baud_rate_clk;
        counter=0;
    end
end
endmodule

module sample_generator(baud_rate_clk,sample_clk);
input baud_rate_clk;
output sample_clk;//9600Hz signal
reg sample_clk;
reg [3:0]counter;
```

```
initial begin
    sample_clk=0;
    counter=4'b0000;
end
always @(posedge baud_rate_clk) begin
    counter=counter+4'b0001;
    if (counter==8) begin
        sample_clk=~sample_clk;
        counter=0;
    end
end
endmodule

module UART(UART_RX,sysclk,UART_TX);
input UART_RX;
input sysclk;
output UART_TX;
wire [7:0]RX_DATA;
wire RX_STATUS;
wire [7:0]TX_DATA;
wire TX_STATUS;
wire TX_EN;
wire baud_rate_clk;
wire sample_clk;
UART_receiver ur1(UART_RX,sysclk,RX_DATA,RX_STATUS);
UART_control uc1(RX_DATA,RX_STATUS,sysclk,TX_STATUS,TX_DATA,TX_EN);
Baud_Rate_Generator brg1(sysclk,baud_rate_clk);
sample_generator sg1(baud_rate_clk,sample_clk);
UART_sender us1(sample_clk,TX_DATA,TX_EN,TX_STATUS,UART_TX);
endmodule

module UART_testbench;
reg UART_RX;
reg sysclk;
wire UART_TX;

wire [7:0]RX_DATA;
wire RX_STATUS;
wire [7:0]TX_DATA;
wire TX_STATUS;
wire TX_EN;
wire baud_rate_clk;
wire sample_clk;
UART_receiver ur1(UART_RX,sysclk,RX_DATA,RX_STATUS);
```

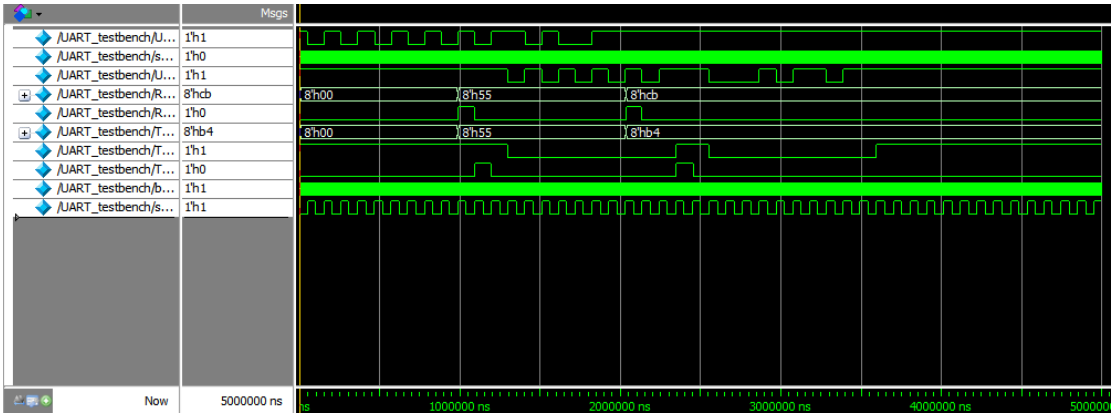
```
UART_control uc1(RX_DATA,RX_STATUS,sysclk,TX_STATUS,TX_DATA,TX_EN);
Baud_Rate_Generator brg1(sysclk,baud_rate_clk);
sample_generator sg1(baud_rate_clk,sample_clk);
UART_sender us1(sample_clk,TX_DATA,TX_EN,TX_STATUS,UART_TX);
initial begin
    forever #5 sysclk<=~sysclk;
end
initial begin
    sysclk<=0;
    UART_RX=1;
    #52083 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=1;
    #104166 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=0;
    #104166 UART_RX=0;
    #104166 UART_RX=1;
    #104166 UART_RX=1;
    #104166 UART_RX=1;
end
endmodule
```

2. 端口约束文件：constraints.xdc

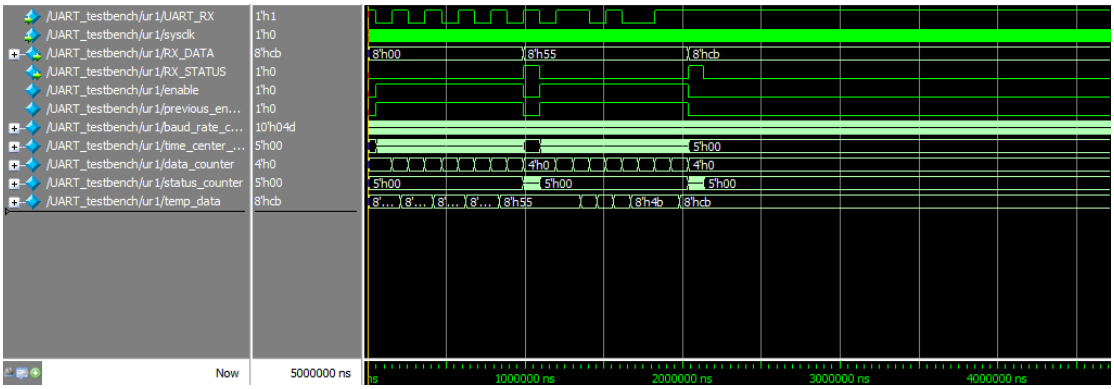
```
set_property PACKAGE_PIN B18 [get_ports {UART_RX}]
set_property PACKAGE_PIN A18 [get_ports {UART_TX}]
set_property PACKAGE_PIN W5 [get_ports {sysclk}]

create_clock -period 10.000 -name CLK -waveform {0.000 5.000}
[get_ports sysclk]
#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets sysclk]
set_property IOSTANDARD LVCMOS33 [get_ports {UART_RX}]
set_property IOSTANDARD LVCMOS33 [get_ports {UART_TX}]
set_property IOSTANDARD LVCMOS33 [get_ports {sysclk}]
```

四、 仿真结果与分析

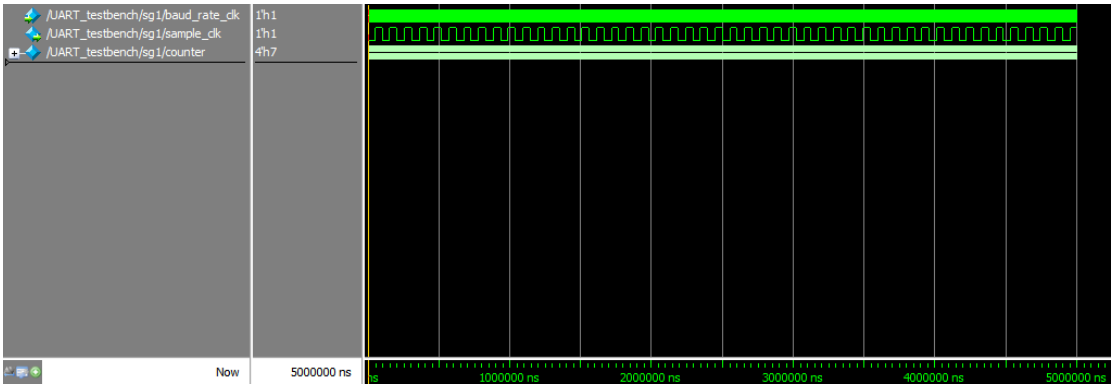
1. 整体仿真结果



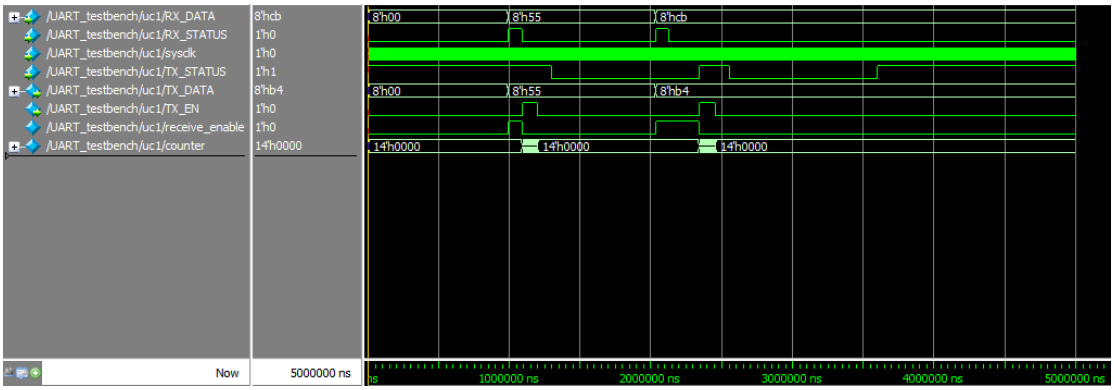
2. 接收器模块



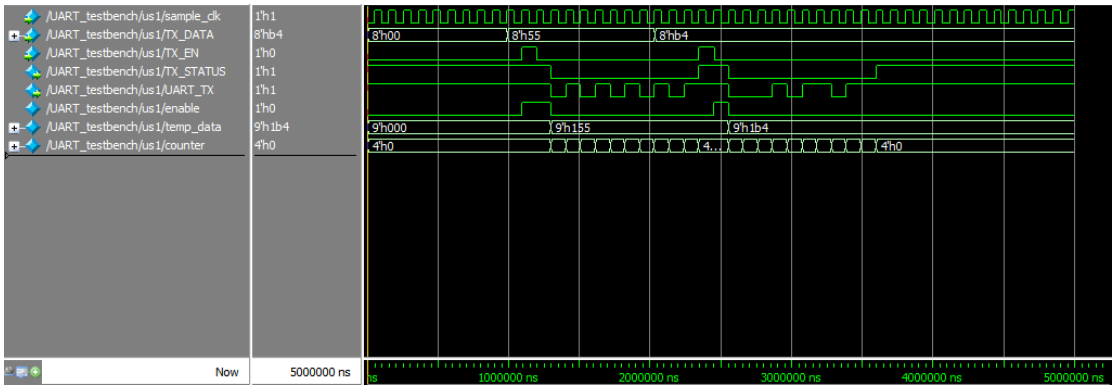
3. 波特率发生器



4. 控制器模块



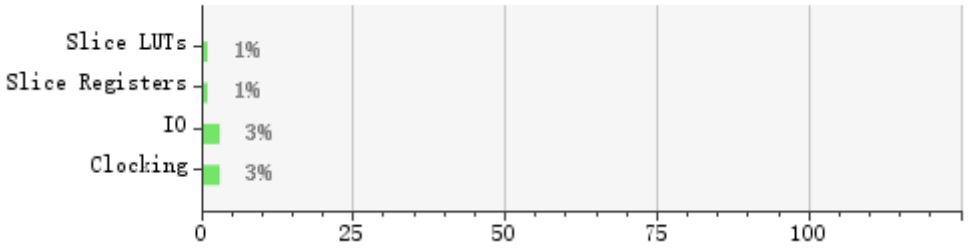
5. 发送器模块



五、 综合报告

1. 面积报告

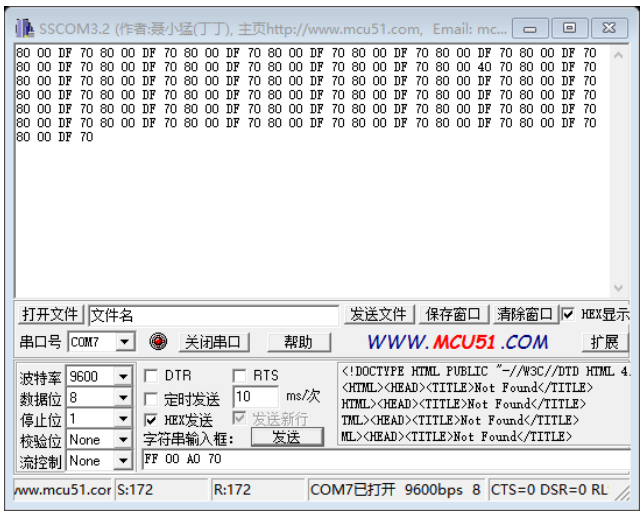
Resource	Utilization	Available	Utilizat...
Slice LUTs	100	20800	0.48
Slice Registers	88	41600	0.21
IO	3	120	2.50
Clocking	1	32	3.12



2. 时序报告

Name	Waveform	Period (ns)	Frequency (MHz)
CLK	{0.000 5.000}	10.000	100.000

六、 硬件调试情况



七、 实验总结与体会

此次实验作为最后一次实验，同时也是我个人认为难度最高的一次。这次实验中涉及到了异步时钟的通信问题，对收发器的逻辑设计要求很高。在实验中也出现了很多问题，首先是误码率非常高的问题。刚写完代码烧程序之后发现收到的字符误码率非常高，大概为 10%。为了解决这个问题，我重新设计了发送器部分的逻辑，经修改之后误码率大大降低。在此之后又出现了连续发送两位数据时只能收到一位的问题。经过仔细检查代码逻辑和方针波形之后，我发现了这个很隐蔽的错误：在修改发送器部分的逻辑之前，TX_EN 信号的持续时间恰好为 1 个 9600Hz 的时钟周期。但是当我修改了发送器的逻辑之后，TX_EN 信号的持续时间过短，以至于第二个数据发送的时候时钟并没有检测到 TX_EN 的高电平。在将 TX_EN 信号的持续时间提高了约 10%之后，问题得到了解决。此次实验的调试过程非常漫长，每一个模块的逻辑都需要思考和调试很久才能真正达到理想的效果。在调试程序的时候，老师说如果将数据连续发送的话，大多数人的代码应该都不能通过。通过这句话我也理解了异步通信的关键之处和难点。这次实验对于理解异步通信的原理和设计方法非常有帮助，同时也是一次很好的逻辑思维训练。感谢老师和助教的悉心指导。