

UART 串口收发的原理与 Verilog 实现

一、 软件平台与硬件平台

软件：Vivado 2017

硬件：龙芯 FPGA 实验箱（不知道啥型号）

二、 原理介绍

串口是串行接口（serial port）的简称，也称为串行通信接口或 COM 接口。串口通信是指采用串行通信协议（serial communication）在一条信号线上将数据一个比特一个比特地逐位进行传输的通信模式。串口按电气标准及协议来划分，包括 RS-232、RS-422、RS485 等。其中最常用的就是 RS-232 接口。

RS-232 接口有以下三个特性：

- 1、用了一个 9 针的连接器"DB-9"(早期的电脑有用 25 针的连接器"DB-25")
- 2、允许全双工通信(即通过串口发送数据和接收数据可以同时进行)
- 3、通信的最大速率大约在 10KBytes/s 左右

虽然 DB-9 接头一共有 9 根线，但是实现串口通信只需要其中的 3 根线就可以了，分别是：

- 1、pin-2: RXD(receive data)，接收串行数据
- 2、pin-3: TXD(transmit data)，发送串行数据
- 3、pin-5: GND(ground)，地线

在串口通信中，数据在 1 位宽的单条线路上进行传输，一个字节的数据要分为 8 次，由低位到高位按顺序一位一位的进行传送,这个过程称为数据的"串行化(serialized)"过程。由于串口通信是一种异步通信协议，并没有时钟信号随着数据一起传输，而且空闲状态(没有数据传输的状态)的时候，串行传输线为高电平 1，所以发送方发送一个字节数据之前会先发送一个低电平 0，接收方收到这个低电平 0 以后就知道有数据要来了，准备开始接收数据从而实现一次通信。串口通信的时序如下图所示：

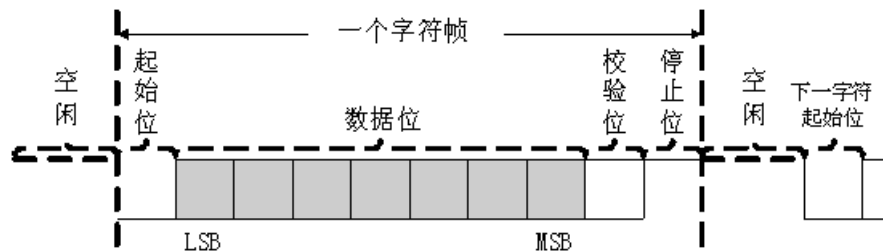


图 1

串口通信的规范如下：

- 1、空闲状态(没有数据传输的状态)下，串行传输线上为高电平 1
- 2、发送方发送低电平 0 表示数据传输开始，这个低电平表示传输的起始位
- 3、8-bit 的数据位(1 Byte)是从最低位开始发送，最高位最后发送
- 4、数据位的最高位发送完毕以后的下一位是奇偶校验位，这一位可以省略不要，同时，当不发送奇偶校验位的时候接收方也相应的不接收校验位
- 5、最后一位是停止位，用高电平 1 表示停止位

下面以发送字节 0x55 为例来说明整个的发送过程：

先把 0x55 转化成二进制为：01010101。显然 0x55 的最低位 bit 0 是 1，次低位 bit 1 是 0，……，最高位 bit 7 是 0，由于串口是从最低位开始发送一个字节，所以 0x55 各个位的发送顺序是 1-0-1-0-1-0-1-0，波形如下图所示

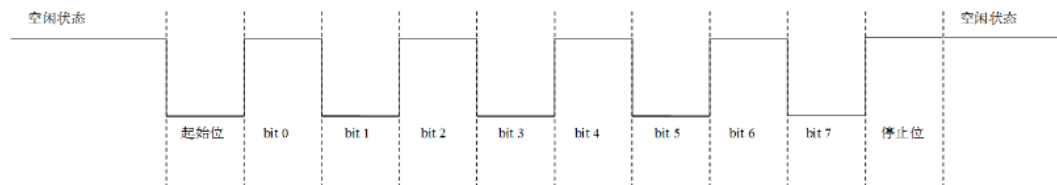


图 2

下面在给出一个波形，根据上面的规则也可以很容易判断这是发送字节 0x13 的波形

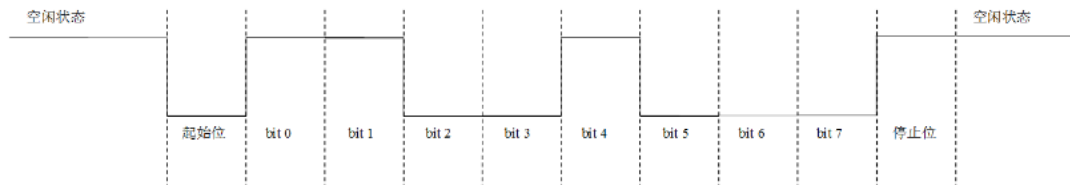


图 3

接下来的最后一个问题是：串口传输的速度是多少？

实际上，串口传输的速度用波特率(baudrate)来指定。波特率表示的是每秒发送的比特数，单位是 bps(bits-per-seconds)，例如，1000 bauds 表示 1 秒钟发送了 1000 个比特，或者说每个比特持续的时间是 1ms。关于串口发送的波特率是有一组标准的规定的，并不是随便一个数字。常用的波特率标准有：

- 1、1200 bps
- 2、9600 bps (常用)
- 3、38400 bps
- 4、115200 bps (常用，而且通常情况下是我们能用的最快的波特率)

波特率为 115200 bps 时，每个比特持续的时间为 $(1/115200)=8.7\mu s$ ，所以发送 8 个 bit(1 Byte)需要的时间是 $8 \times 8.7\mu s = 69\mu s$ 。在不考虑奇偶校验位的情况下，发送一个字节还需要发送额外的 1 个起始位和 1 个停止位，所以发送 1 个字节实际所需要的最少时间是 $10 \times 8.7\mu s = 87\mu s$ ，这意味着 1s(1000000us)中能发送的字节数为 $(1000000/87) = 11494$ ，所以在波特率为 115200bps 的情况下，串口传输数据的速率约为 11.5KB/s。而有些电脑的串口有时候需要一个更长的停止位，比如 1.5 位或 2 位的停止位，那么发送一个字节所需要的时间比只有一个

比特停止位的情况所耗费的时间更长，在这种情况下，串口的传输速率会低于 10.5KB/s。

通过上面一系列的总结以后，可以得出 FPGA 与 PC 之间的串口通信主要包括三个模块：波特率产生模块、发射模块和接收模块。

三、 目标功能

1、编写发送模块的 verilog 代码，并往 PC 上连续不断发送 0x00~0xff 这些数据，PC 上用串口调试助手进行接收并以 16 进制显示出来

2、在第一个功能的基础上编写接收模块的 verilog 代码，接收模块接收到第一个功能中发送模块发送的数据以后，用接收到的并行数据的低四位驱动板上的四个 LED 灯

3、编写一个顶层模块把发送模块和接收模块均例化进去，然后从 PC 的串口调试助手上发送数据到 FPGA，FPGA 接收到数据以后把接收的数据返回给串口调试助手显示。

四、设计思路与 Verilog 代码编写

4.1、发送模块波特率时钟的设计与实现

本节以波特率为 115200bps 为例来说明波特率模块设计方法，其余波特率可以以此类推。由于我的开发板上的时钟为 50MHz，周期 $T=20\text{ns}$ ，而波特率为 115200bps，所以 1 个 bit 持续的时间是 $8.7\mu\text{s}$ ，那么每个 bit 占用的周期数 $N=(8.7\mu\text{s} / 20\text{ns}) = 434$ ，所以可以定义一个计数器，每当计数器从 0 计数到 433 的时候就把计数器清零，然后在计数值为 1(这个计数值最好比 433 的一半要小，这篇博客的最后一部分分析了原因)的情况下产生一个高脉冲。发射模块只要检测到这个高脉冲的到来就发送一个 bit，这样就实现了波特率为 115200bps 的串口数据发送。

而接收模块的波特率时钟产生逻辑与发送的波特率时钟相比稍有不同。不同之处在于当接收模块检测到 `I_rs232_rxd` 的下降沿以后，表示有数据过来，准备开始接收数据了，由于一个 bit 持续的时间为 434 个时钟周期，所以为了保证接收模块接收数据的准确性，我们需要在 $434/2=217$ 个周期，也就是数据的正中间位置的时候把输入的数据接收并存起来。也就是说接收模块的波特率时钟要比发射模块的波特率时钟滞后数个周期

波特率产生模块的框图如下图所示

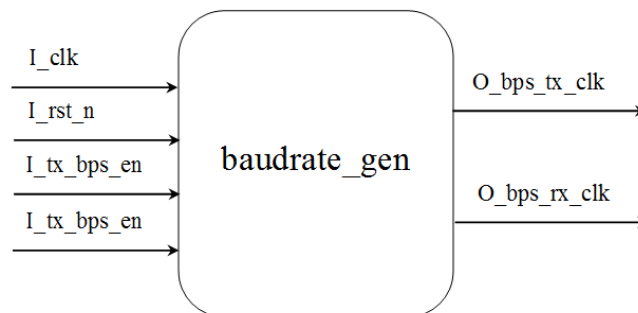


图 4

其中：

I_clk 是系统时钟；

I_rst_n 是系统复位；

I_tx_bps_en 是发射模块波特率使能信号，当 I_tx_bps_en 为 1 时 O_bps_tx_clk 才有时钟信号输出；

I_rx_bps_en 是接收模块波特率使能信号，当 I_rx_bps_en 为 1 时 O_bps_rx_clk 才有时钟信号输出。

波特率模块的完整代码如下：

```
module baudrate_gen
(
    input    I_clk                , // System 50MHz clk
    input    I_rst_n              , // Reset
    input    I_bps_tx_clk_en      , // Transmit Enable
    input    I_bps_rx_clk_en      , // Receive Enable
    output   O_bps_tx_clk         , // 发送模块波特率产生时钟
    output   O_bps_rx_clk         , // 接收模块波特率产生时钟
);

parameter    C_BPS9600          = 5207      ,
              C_BPS19200         = 2603      ,
              C_BPS38400         = 1301      ,
              C_BPS57600         = 867       ,
              C_BPS115200        = 433       ;

parameter    C_BPS_SELECT       = C_BPS115200 ;

reg [12:0] R_bps_tx_cnt;
reg [12:0] R_bps_rx_cnt;

//////////////////////////
// transmit
//////////////////////////
always @(posedge I_clk or negedge I_rst_n)
begin
    if(!I_rst_n)
        R_bps_tx_cnt <= 13'd0 ;
    else if(I_bps_tx_clk_en == 1'b1)
        begin
            if(R_bps_tx_cnt == C_BPS_SELECT)
                R_bps_tx_cnt <= 13'd0 ;
            else
                R_bps_tx_cnt <= R_bps_tx_cnt + 1'b1 ;
        end
end
```

```

        end
    else
        R_bps_tx_cnt <= 13'd0 ;
    end

assign O_bps_tx_clk = (R_bps_tx_cnt == 13'd1) ? 1'b1 : 1'b0 ;

/////////////////////////////////////////////////////////////////
// receive
/////////////////////////////////////////////////////////////////
always @(posedge I_clk or negedge I_rst_n)
begin
    if(!I_rst_n)
        R_bps_rx_cnt <= 13'd0 ;
    else if(I_bps_rx_clk_en == 1'b1)
        begin
            if(R_bps_rx_cnt == C_BPS_SELECT)
                R_bps_rx_cnt <= 13'd0 ;
            else
                R_bps_rx_cnt <= R_bps_rx_cnt + 1'b1 ;
            end
        end
    else
        R_bps_rx_cnt <= 13'd0 ;
    end

assign O_bps_rx_clk = (R_bps_rx_cnt == C_BPS_SELECT >> 1'b1) ? 1'b1 : 1'b0 ;

endmodule

```

运行的仿真结果为：

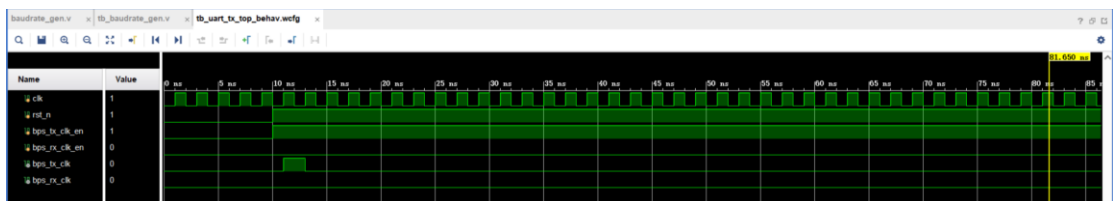


图 5

4.2、发送模块的设计与实现

有了波特率时钟以后，就可以开始编写发送模块的内部逻辑了。发送模块的结构框图如下图所示

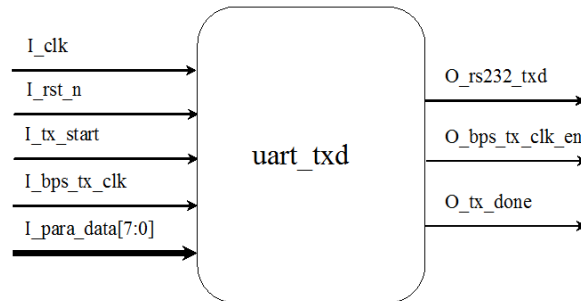


图 6

其中：

I_clk 是系统时钟；

I_rst_n 是系统复位；

I_tx_start 是开始发送信号，当检测到 I_tx_start 为高电平时，立马把输入 I_para_data[7:0] 的数据串行化成单 bit 的发出去；

I_bps_tx_clk 是发送模块波特率时钟信号，当检测到 I_bps_tx_clk 为高的时候就发送 1 个 bit；

I_para_data[7:0] 是并行的 8-bit 数据；

O_rs232_txd 是串行的 bit 数据流；

O_bps_clk_en 是发射波特率时钟启动信号，当它为 1 是波特率产生模块才能产生发射模块的波特率时钟；

O_tx_done 是发送 1 字节数据完成的标志位，当一个字节发送完毕以后，O_tx_done 产生一个高脉冲。

以发送字节 0x55 为例，发送模块几个关键信号的时序图如下图所示

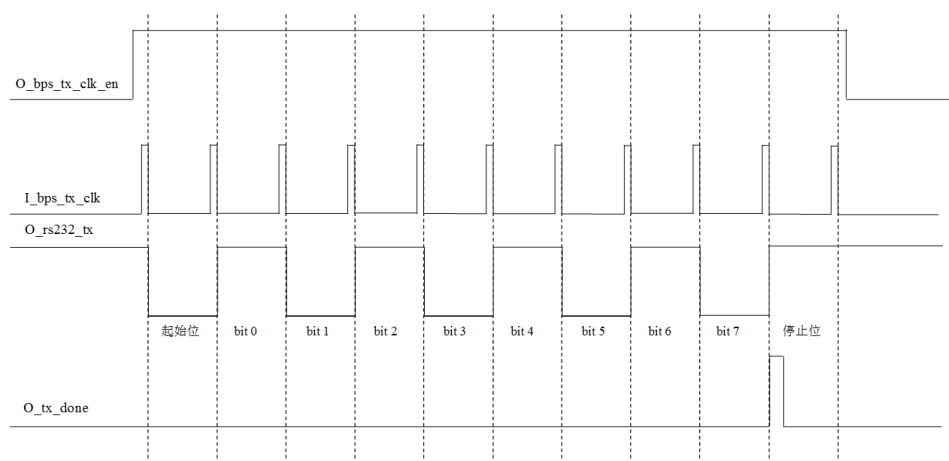


图 7

发送模块的代码如下：

```
module uart_txd
(
    input          I_clk          ,//50MHz
    input          I_rst_n        ,// 系统全局复位
    input          I_tx_start     ,// 发送使能信号
    input          I_bps_tx_clk   ,// 发送波特率时钟
    input  [7:0]    I_para_data    ,// 要发送的并行数据
    output reg     O_rs232_txd    ,// 发送的串行数据，在硬件上与串口相
连
    output reg     O_bps_tx_clk_en ,// 波特率时钟使能信号
    output reg     O_tx_done      // 发送完成的标志
);

reg [3:0] R_state ;

reg R_transmitting ; // 数据正在发送标志

/////////////////////////////////////////////////////////////////
////
// 产生发送 R_transmitting 标志位
/////////////////////////////////////////////////////////////////
////
always @(posedge I_clk or negedge I_rst_n)
begin
    if(!I_rst_n)
        R_transmitting <= 1'b0 ;
    else if(O_tx_done)
        R_transmitting <= 1'b0 ;
    else if(I_tx_start)
        R_transmitting <= 1'b1 ;
end

/////////////////////////////////////////////////////////////////
////
// 发送数据状态机
/////////////////////////////////////////////////////////////////
////
always @(posedge I_clk or negedge I_rst_n)
begin
    if(!I_rst_n)
        begin
            R_state      <= 4'd0 ;
```



```

        O_rs232_txd <= 1'b1 ;
        O_tx_done   <= 1'b0 ;
        O_bps_tx_clk_en <= 1'b0 ;
    end
else if(R_transmiting)
    begin
        O_bps_tx_clk_en <= 1'b1 ;
        if(I_bps_tx_clk)
            begin
                case(R_state)
                    4'd0 :
                        begin
                            O_rs232_txd <= 1'b0 ;
                            O_tx_done   <= 1'b0 ;
                            R_state      <= R_state + 1'b1 ;
                        end
                    4'd1 : //I_para_data[0]
                        begin
                            O_rs232_txd <= I_para_data[0] ;
                            O_tx_done   <= 1'b0 ;
                            R_state      <= R_state + 1'b1 ;
                        end
                    4'd2 : //I_para_data[1]
                        begin
                            O_rs232_txd <= I_para_data[1] ;
                            O_tx_done   <= 1'b0 ;
                            R_state      <= R_state + 1'b1 ;
                        end
                    4'd3 : //I_para_data[2]
                        begin
                            O_rs232_txd <= I_para_data[2] ;
                            O_tx_done   <= 1'b0 ;
                            R_state      <= R_state + 1'b1 ;
                        end
                    4'd4 : //I_para_data[3]
                        begin
                            O_rs232_txd <= I_para_data[3] ;
                            O_tx_done   <= 1'b0 ;
                            R_state      <= R_state + 1'b1 ;
                        end
                    4'd5 : //I_para_data[4]
                        begin
                            O_rs232_txd <= I_para_data[4] ;
                            O_tx_done   <= 1'b0 ;

```

```

        R_state      <= R_state + 1'b1    ;
    end
4'd6 : //I_para_data[5]
    begin
        O_rs232_txd  <= I_para_data[5]    ;
        O_tx_done    <= 1'b0              ;
        R_state      <= R_state + 1'b1    ;
    end
4'd7 : //I_para_data[6]
    begin
        O_rs232_txd  <= I_para_data[6]    ;
        O_tx_done    <= 1'b0              ;
        R_state      <= R_state + 1'b1    ;
    end
4'd8 : //I_para_data[7]
    begin
        O_rs232_txd  <= I_para_data[7]    ;
        O_tx_done    <= 1'b0              ;
        R_state      <= R_state + 1'b1    ;
    end
4'd9 : //end
    begin
        O_rs232_txd  <= 1'b1 ;
        O_tx_done    <= 1'b1 ;
        R_state      <= 4'd0 ;
    end
    default :R_state      <= 4'd0 ;
endcase
end
else
    begin
        O_bps_tx_clk_en <= 1'b0 ;
        R_state          <= 4'd0 ;
        O_tx_done        <= 1'b0 ;
        O_rs232_txd      <= 1'b1 ;
    end
end

endmodule

```

其中当检测到输入信号 I_tx_start 为高电平以后，发送模块立即把 R_transmitting 信号拉高，表示开始要发送数据了，在 R_transmitting 为高电平的期间，打开波特率时钟使能信号并且在波特率时钟的控制下通过一个状态机把并行数据发送出去，并产生发送完成信号 O_tx_done，等 O_tx_done 为高以后再把 R_transmitting 拉低表示一次发送结束。

为了实现功能 1 的效果还需要编写一个顶层模块把波特率模块和发送模块例化进去并产生发送的信号，顶层模块的代码如下：

```
module uart_tx_top
(
    input          I_clk          , // 系统 50MHz 时钟
    input          I_rst_n        , // 系统全局复位
    output         O_rs232_txd    // 发送的串行数据，在硬件上与串口相连
);

wire             W_bps_tx_clk;
wire             W_bps_tx_clk_en;
wire             W_tx_start;
wire             W_tx_done;
wire [7:0]       W_para_data;

reg [7:0]        R_data_reg;
reg [31:0]       R_cnt_1s;
reg              R_tx_start_reg;

assign W_tx_start    =    R_tx_start_reg;
assign W_para_data   =    R_data_reg;

/////////////////////////////////////////////////////////////////
// Generate the data
/////////////////////////////////////////////////////////////////
always @(posedge I_clk or negedge I_rst_n)
begin
    if(!I_rst_n)
        begin
            R_cnt_1s          <= 31'd0;
            R_data_reg        <= 8'd0;
            R_tx_start_reg <= 1'b0;
        end
    else if(R_cnt_1s == 31'd24_999_999)
        begin
            R_cnt_1s          <= 31'd0;
            R_data_reg        <= R_data_reg + 1'b1;
            R_tx_start_reg <= 1'b1;
        end
    end
end
```

```

else
begin
    R_cnt_1s          <= R_cnt_1s + 1'b1;
    R_tx_start_reg    <= 1'b0;
end
end

uart_txd U_uart_txd
(
    .I_clk              (I_clk),
    .I_rst_n            (I_rst_n),
    .I_tx_start         (W_tx_start),
    .I_bps_tx_clk       (W_bps_tx_clk),
    .I_para_data        (W_para_data),
    .O_rs232_txd        (O_rs232_txd),
    .O_bps_tx_clk_en    (W_bps_tx_clk_en),
    .O_tx_done          (W_tx_done)
);

baudrate_gen U_baudrate_gen
(
    .I_clk              (I_clk),
    .I_rst_n            (I_rst_n),
    .I_bps_tx_clk_en    (W_bps_tx_clk_en),
    .I_bps_rx_clk_en    (          ),
    .O_bps_tx_clk       (W_bps_tx_clk),
    .O_bps_rx_clk       (          )
);

endmodule

```

将 R_cnt_1s 修改为 4999，即 5000 个时钟周期传输一个消息
仿真结果为：

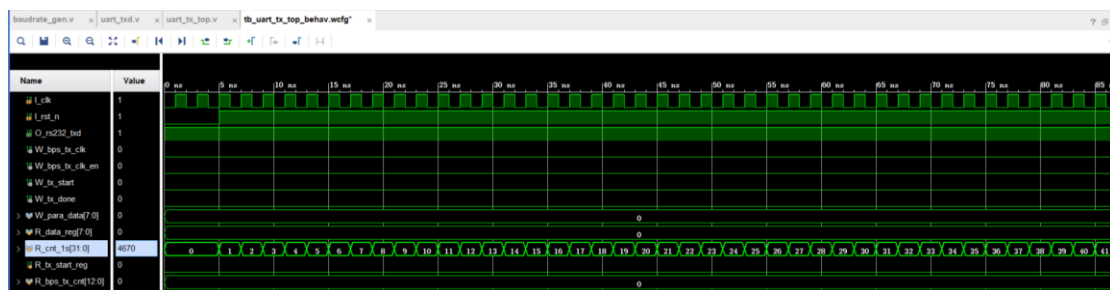


图 8

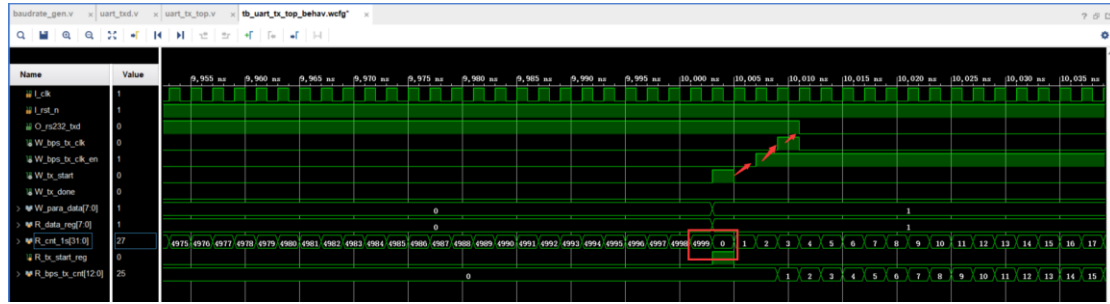


图 9

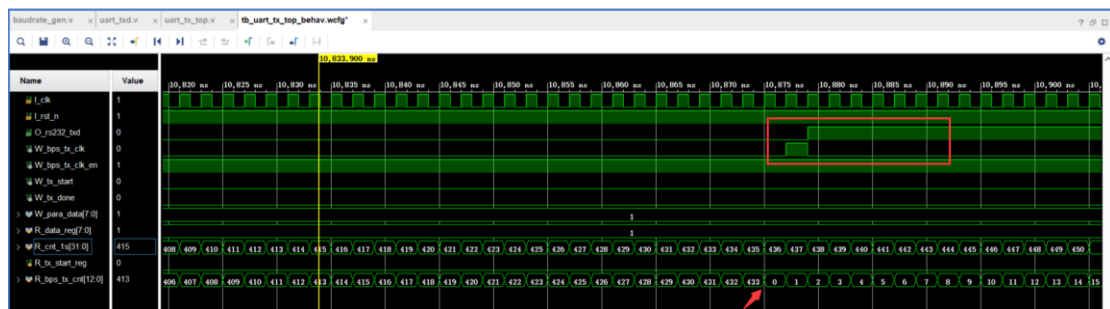


图 10

在本地仿真完成后，便要接入板上测试，我们编写约束文件：



```
set_property PACKAGE_PIN H7 [get_ports {O_led_out[3]}]
set_property PACKAGE_PIN D5 [get_ports {O_led_out[2]}]
set_property PACKAGE_PIN A3 [get_ports {O_led_out[1]}]
set_property PACKAGE_PIN A5 [get_ports {O_led_out[0]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {O_led_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_led_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_led_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_led_out[3]}]
```

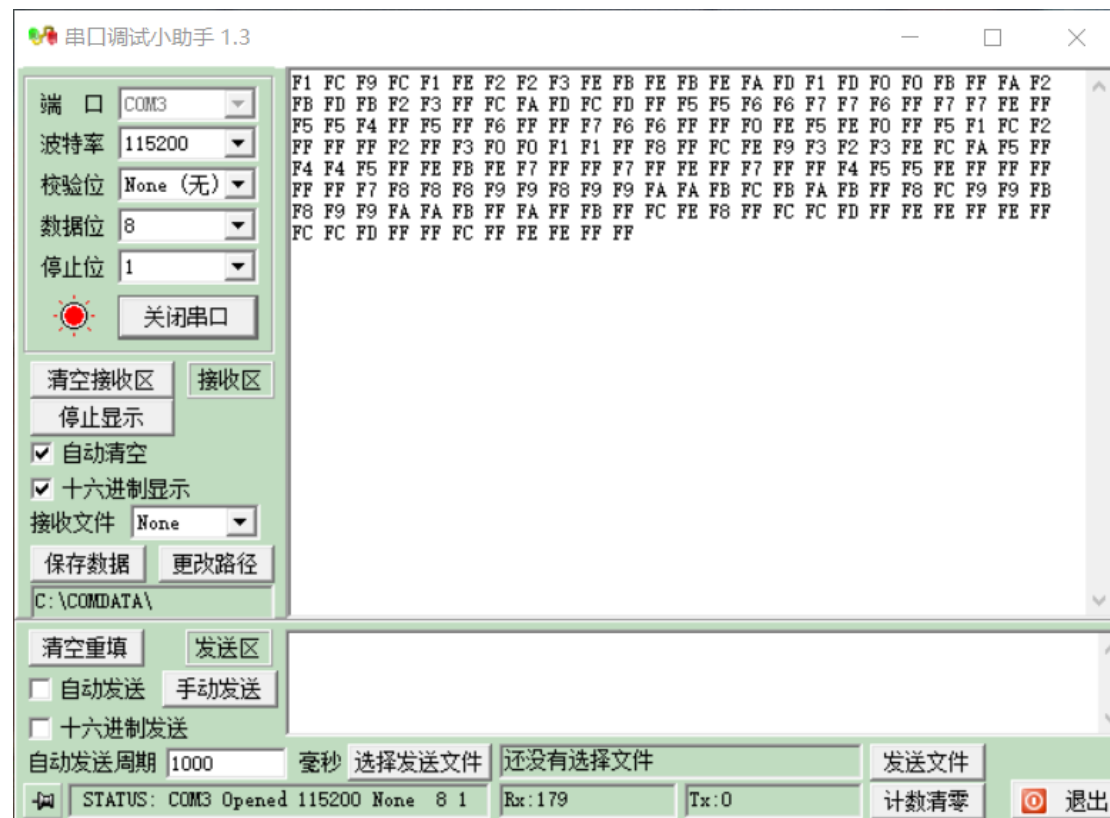
```
set_property PACKAGE_PIN AC19 [get_ports I_clk]
set_property PACKAGE_PIN Y3 [get_ports I_rst_n]
set_property IOSTANDARD LVCMOS33 [get_ports I_clk]
set_property IOSTANDARD LVCMOS33 [get_ports I_rst_n]
set_property PACKAGE_PIN H19 [get_ports O_rs232_txd]
set_property IOSTANDARD LVCMOS33 [get_ports O_rs232_txd]
set_property PACKAGE_PIN F23 [get_ports I_rs232_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports I_rs232_rxd]
```

选取芯片：**xc7a200tfbg676-1 (active)**

注意：约束文件一个字都不能打错，一个空格都不能少

然后点击  Run Implementation ,  Generate Bitstream 生成比特流文件，时间较长，耐心等待。

最后接入 **FPGA** 实验箱，安装驱动（都是自动的），指的提醒的是，在比特流文件拷入实验箱之前（包括进行调试的时候，实验箱都不要断电，因为实验箱使用 **ROM**，断电就会丢失文件），最后测试成功的截图：



还有接收模块、串口回显功能等还没有完成，以后有空会继续实现。

参考：[【接口时序】3、UART 串口收发的原理与 Verilog 实现](#)（[jgliu 的博客园](#)）