第三讲 串口发送模块设计及收发整合

最近更新日期：2016/11/21

相信很多朋友在学习完第二讲中所讲到的串口接收模块后，一定非常想学习如何设计串口的发送模块，Kevin 也非常明白大家的好学之心，所以在第三讲，Kevin 会带着大家一起来完成串口发送模块的设计，并且会将第二讲中已经设计好的串口接收模块与这一讲即将完成的串口发送模块进行整合，来实现一个完整的串口收发工程。

本讲主要内容如下：

1) 串口发送模块时序设计；
2) 串口发送模块代码编写；
3) 串口发送模块仿真验证；
4) 串口收发模块整合及板级验证。

一、串口发送模块时序设计

对于串口发送模块的时序，其实与接收模块的时序大同小异，只是发送模块是要构造出 RS232 协议的波形，而接收模块是如何在接收到 RS232 协议的波形后正确的检测出数据。

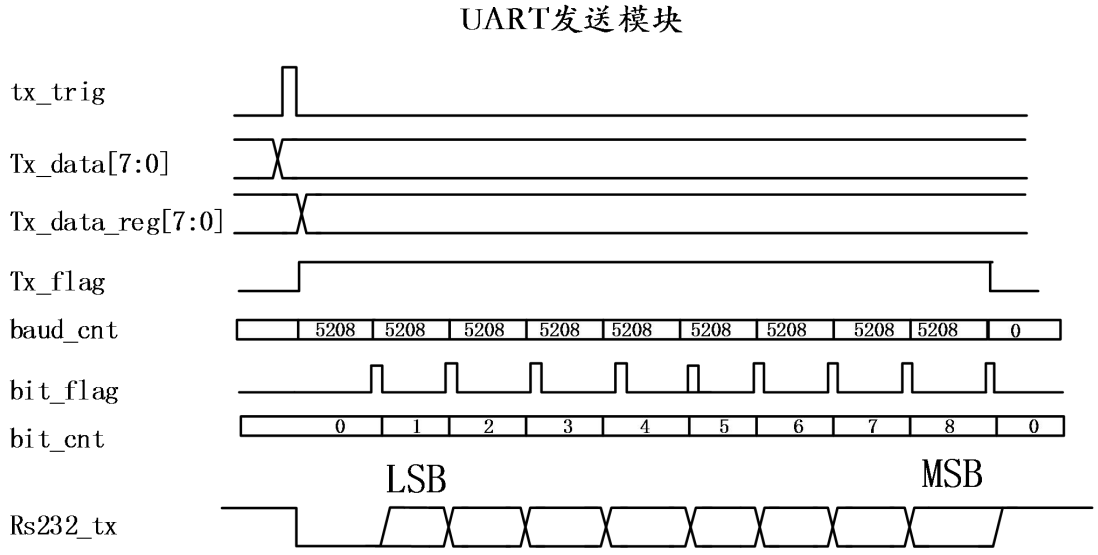

图 1 串口发送模块时序图

下面对时序图中的各个信号作简单的介绍：

① Tx_trig 表示串口发送模块工作的触发信号，也就是相当于让 FPGA 开始

通过串口发送数据的一个触发信号；

② Tx_data 为即将发送的数据；

③ Tx_data_reg 是保存待发送数据的寄存器，而该寄存器的值只会在 tx_trig 为高时进行改变；

④ Tx_flag 是串口发送模块处于工作状态的一个标志信号，为高则表示 FPGA 正在发送数据。

⑤ Baud_cnt 依然是进行 5208 个时钟周期的计数，大家一定要注意，该计数器只会在 tx_flag 为高时进行工作，其余状态为 0；

⑥ Bit_flag 在串口发送模块中，是在 baud_cnt 计数到最大值（5207）时拉高，与串口接收模块中稍微有些差别；

⑦ Bit_cnt 在串口接收模块中是用来计数已接收到串口数据的 bit 数，而在这（串口发送模块）则表示已发送 bit 数量；

⑧ Rs232_tx 则表示串口的发送端，如何构造 RS232_tx 发送的数据呢？我们完全可以利用 bit_cnt 来完成，在 bit_cnt 为不同的值时，则发送不同的数据位。

由于串口发送模块的时序与接收模块的时序类似，Kevin 就不作详细的解释了，相信大家也是可以完全理解的。如果有疑问，也可以配合视频教程来一起学习。

二、串口发送模块代码编写

不知道大家有没有发现，Kevin 在视频中写代码，一般都是对照着时序图来进行的。因为在我们进行时序设计时，其实就是在整理自己思路的一个过程，时序图画出来了，代码可以写得似行云流水般飞快。

所以呢，Kevin 也建议大家先不看下边 Kevin 贴出来的代码，先根据给出的时序图将代码写出来。

根据设计好的时序图，写出的代码如下：

```
// ********************************************************************************
// Project Name :
// Author       : Kevin
// Email        : deng.kanwen@163.com
// Create Time  : 2016/8/29 14:18:53
// File Name    : .v
// Module Name  :
// Called By    :
// Abstract     :
//
```

```
// CopyRight(c) 2016, OpenSoc Studio..
// All Rights Reserved
//
// ********************************************************************************
// Modification History:
// 1. initial
// ********************************************************************************/
// *************************
// MODULE DEFINITION
// *************************

//`define SIM

module  uart_tx(
        // system signals
        input                   sclk                    ,
        input                   s_rst_n                 ,
        // UART Interface
        output  reg             rs232_tx                ,
        // others
        input                   tx_trig                 ,
        input           [ 7:0]  tx_data
);

//=================================================================\
// ********** Define Parameter and Internal Signals *************
//=================================================================/
`ifndef SIM
localparam      BAUD_END        =       5207                    ;
`else
localparam      BAUD_END        =       56                      ;
`endif
localparam      BAUD_M          =       BAUD_END/2 - 1          ;
localparam      BIT_END         =       8                       ;

reg     [ 7:0]                  tx_data_r                       ;
reg                             tx_flag                         ;
reg     [12:0]                  baud_cnt                        ;
```

```
    reg                             bit_flag                        ;
    reg     [ 3:0]                  bit_cnt                         ;
//=============================================================================
// ***************      Main    Code     ****************
//=============================================================================
    // tx_data_r
    always  @(posedge sclk or negedge s_rst_n) begin
            if(s_rst_n == 1'b0)
                    tx_data_r       <=      'd0;
            else if(tx_trig == 1'b1 && tx_flag == 1'b0)
                    tx_data_r       <=      tx_data;
    end

    // tx_flag
    always  @(posedge sclk or negedge s_rst_n) begin
            if(s_rst_n == 1'b0)
                    tx_flag <=      1'b0;
            else if(tx_trig == 1'b1)
                    tx_flag <=      1'b1;
            else if(bit_cnt == BIT_END && bit_flag == 1'b1)
                    tx_flag <=      1'b0;
    end

    //baud_cnt
    always  @(posedge sclk or negedge s_rst_n) begin
            if(s_rst_n == 1'b0)
                    baud_cnt        <=      'd0;
            else if(baud_cnt == BAUD_END)
                    baud_cnt        <=      'd0;
            else if(tx_flag == 1'b1)
                    baud_cnt        <=      baud_cnt + 1'b1;
            else
                    baud_cnt        <=      'd0;
    end

    // bit_flag
    always  @(posedge sclk or negedge s_rst_n) begin
            if(s_rst_n == 1'b0)
```

```
                bit_flag        <=      1'b0;
        else if(baud_cnt == BAUD_END)
                bit_flag        <=      1'b1;
        else
                bit_flag        <=      1'b0;
end

//bit_cnt
always  @(posedge sclk or negedge s_rst_n) begin
        if(s_rst_n == 1'b0)
                bit_cnt <=      'd0;
        else if(bit_flag == 1'b1 && bit_cnt == BIT_END)
                bit_cnt <=      'd0;
        else if(bit_flag == 1'b1)
                bit_cnt <=      bit_cnt + 1'b1;
end

// rs232_tx
always  @(posedge sclk or negedge s_rst_n) begin
        if(s_rst_n == 1'b0)
                rs232_tx        <=      1'b1;
        else if(tx_flag == 1'b1)
                case(bit_cnt)
                        0:      rs232_tx        <=      1'b0;      // start bit
                        1:      rs232_tx        <=      tx_data_r[0];
                        2:      rs232_tx        <=      tx_data_r[1];
                        3:      rs232_tx        <=      tx_data_r[2];
                        4:      rs232_tx        <=      tx_data_r[3];
                        5:      rs232_tx        <=      tx_data_r[4];
                        6:      rs232_tx        <=      tx_data_r[5];
                        7:      rs232_tx        <=      tx_data_r[6];
                        8:      rs232_tx        <=      tx_data_r[7];
                        default:rs232_tx        <=      1'b1;
                endcase
        else
                rs232_tx        <=      1'b1;
end
```

```
endmodule
```

三、串口发送模块仿真验证

对于串口发送模块的仿真，要比对串口接收模块的仿真要简单得多，下边是对串口发送模块进行仿真的 TestBench 代码。

TestBench 文件源代码如下：

```
`timescale      1ns/1ns

module  tb_uart_tx;

reg             sclk;
reg             s_rst_n;
reg             tx_trig;
reg     [ 7:0]  tx_data;

wire            rs232_tx;

//------------- generate system signals -----------------------------------
initial begin
        sclk    =       1;
        s_rst_n <=      0;
        #100
        s_rst_n <=      1;
end

always  #5      sclk    =       ~sclk;

//-------------------------------------------------------------------------
initial begin
        tx_data <=      8'd0;
        tx_trig <=      0;
        #200
        tx_trig <=      1'b1;
        tx_data <=      8'h55;
```

```
        #10
        tx_trig <=      1'b0;
end

uart_tx         uart_tx_inst(
        // system signals
        .sclk                   (sclk                   ),
        .s_rst_n                (s_rst_n                ),
        // UART Interface
        .rs232_tx               (rs232_tx               ),
        // others
        .tx_trig                (tx_trig                ),
        .tx_data                (tx_data                )
);

endmodule
```

下图是对串口发送模块进行仿真的波形，待发送的数据位 8'h55，大家可以看到 rs232_tx 这个信号也是发送的 8'h55，这说明串口发送模块是设计正确的。

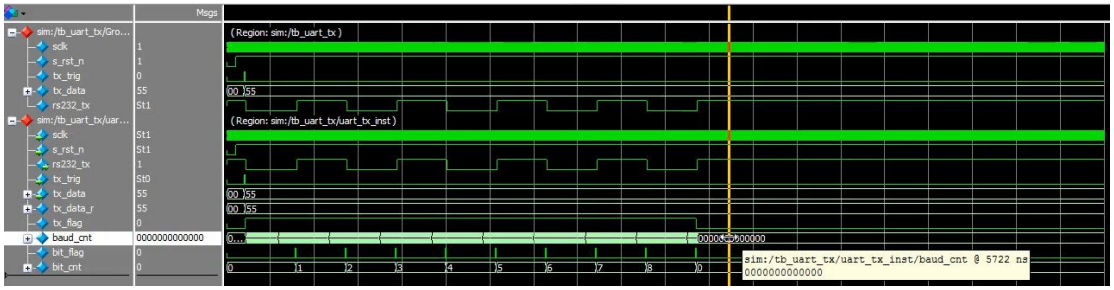

图 2 串口发送模块仿真波形

四、串口收发模块整合及板级验证

现在，串口的发送模块和接收模块都已经弄好了，下边我们再来写一个顶层文件，并将已设计好的这两个模块在顶层中例化，来形成一个串口的回环工程。

顶层文件代码如下：

```
// ********************************************************************************
// Project Name :
// Author       : Kevin
// Email        : deng.kanwen@163.com
// Creat Time   : 2016/8/29 14:18:53
```

```
// File Name    : .v
// Module Name  :
// Called By    :
// Abstract     :
//
// CopyRight(c) 2016, OpenSoc Studio..
// All Rights Reserved
//
// ***************************************************************************
// Modification History:
// 1. initial
// ***************************************************************************/
// *************************
// MODULE DEFINITION
// *************************

module  top(
        // system signals
        input                   sclk                    ,
        input                   s_rst_n                 ,
        // UART Interface
        input                   rs232_rx                ,
        output  wire            rs232_tx
);

//=============================================================\
// ********** Define Parameter and Internal Signals *************
//=============================================================/
wire    [ 7:0]                  rx_data                         ;
wire                            tx_trig                         ;
//=============================================================================
// ***************      Main    Code    ****************
//=============================================================================
uart_rx         uart_rx_inst(
        // system signals
        .sclk                    (sclk                  ),
        .s_rst_n                 (s_rst_n               ),
        // UART Interface
```

```
        .rs232_rx                   (rs232_rx           ),
        // others
        .rx_data                    (rx_data            ),
        .po_flag                    (tx_trig            )
);

uart_tx         uart_tx_inst(
        // system signals
        .sclk                       (sclk               ),
        .s_rst_n                    (s_rst_n            ),
        // UART Interface
        .rs232_tx                   (rs232_tx           ),
        // others
        .tx_trig                    (tx_trig            ),
        .tx_data                    (rx_data            )
);

endmodule
```

好了，这样我们就在顶层中把串口的发送和接收模块都例化好了，大家快点新建一个工程把这 3 个文件添加进去吧，马上就可以享受到上位机与 FPGA 进行串口通信的乐趣了！！！

对于串口上位机的使用，大家可以观看视频教程进行学习，Kevin 就不在此赘述。

好了，第三讲到这就结束了，从第四讲开始，我们就会进入到 SDRAM 相关内容的部分了，第四讲的内容一定会让你对【开源骚客】的持续关注得到更高的回报哦，敬请期待！！！

特别提醒 #1

a. 发烧友课堂在线观看地址： http://t.elecfans.com/1418.html

b. 百度网盘下载地址： http://pan.baidu.com/s/1o88h0Ps 密码： t9qg

c. 开源骚客 FPGA 交流群：312109973

特别提醒 #2

【开源骚客（微信号：OpenSoc）】公众号致力免费分享 FPGA 相关项目知

识，既包括文章分享，也会包括视频教学。

扫一扫，关注开源骚客公众号

特别提醒 #3

若发现该项目代码有任何 **BUG**，请联系下方微信；

当然，若对该项目代码有任何不理解的地方，也可以加下方微信进行讨论。

【开源骚客公众号】创始人微信号