第二讲 串口接收模块设计及仿真验证

最近更新日期：2016/10/29

相信大家看完第一讲的内容之后，想必对咱们这套教程中的这个 SDRAM 控制器的设计方法还会是有那么一丁点兴趣的 o((≧▽≦o)。

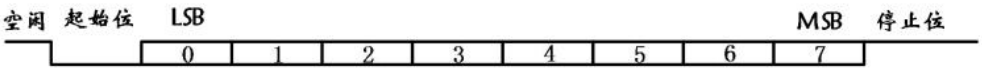
这一讲内容和第三讲内容，主要都是给大家介绍这个项目中有关于串口的内容，而本讲内容主要是讲解串口的接收模块，在这一讲中我们需要完成串口接收模块的所有工作，下一讲则是讲解串口的发送。

本讲主要内容如下：

1) **RS-232 协议介绍；**
2) **串口接收模块时序设计；**
3) **串口接收模块代码编写；**
4) **串口接收模块仿真验证。**

一、RS-232 协议介绍

可能之前有些朋友有接触过 RS-232 协议，为了照顾尚未接触过串口协议的朋友，Kevin 还是打算先花点篇幅介绍下 RS-232 协议，为本讲的后续内容先打下基础。



图一 RS-232 协议示意图

图一是一个不“完整”RS232 协议时序图，我们以第一讲中的那个演示为例。

当 PC 机未给 FPGA 通过串口发送数据时，串口的发送端处于空闲状态，而串口发送端是以高电平来表示空闲状态，也就是说，当串口未发送数据时，其发送端是一直处于高电平。

当 PC 端需要开始发送数据时，首先需要给一个起始位来表示从空闲状态转变为忙碌状态。起始位发送完后，接着就是发送一个字节数据的最低位，然后依次发送直至发送完最高位。

一个完整的 RS232 协议，在发送完数据的最高位之后，还有 1bit 的校验位。但在本套教程中，我们未使用校验位，所以在发送完数据最高位之后，直接给了停止位。

好了，RS232 协议还是比较简单的，我们就先介绍到这。

二、串口接收模块时序设计

在讲解设计思路之前，先说一下几个参数，串口的波特率为 9600，FPGA 使用的系统时钟为 50MHz.

在了解了 RS232 的协议之后，怎么来设计我们的这个串口接收模块呢？首先第一点大家要明确，我们来检测串口上的数据，肯定是在每一位的最中间进行的，这样才能保证检测到的串口数据是在稳定之后的。

既然这样，那我们是不是可以产生一个计数器，这个计数器的计数范围就是串口数据 1bit 所占用的时间，然后在该计数器计到最大值的一半时，进行检测串口上的数据。OK 了，想到这一点接下来就好办了。

对于串口接收模块的时序，如图 2 所示。

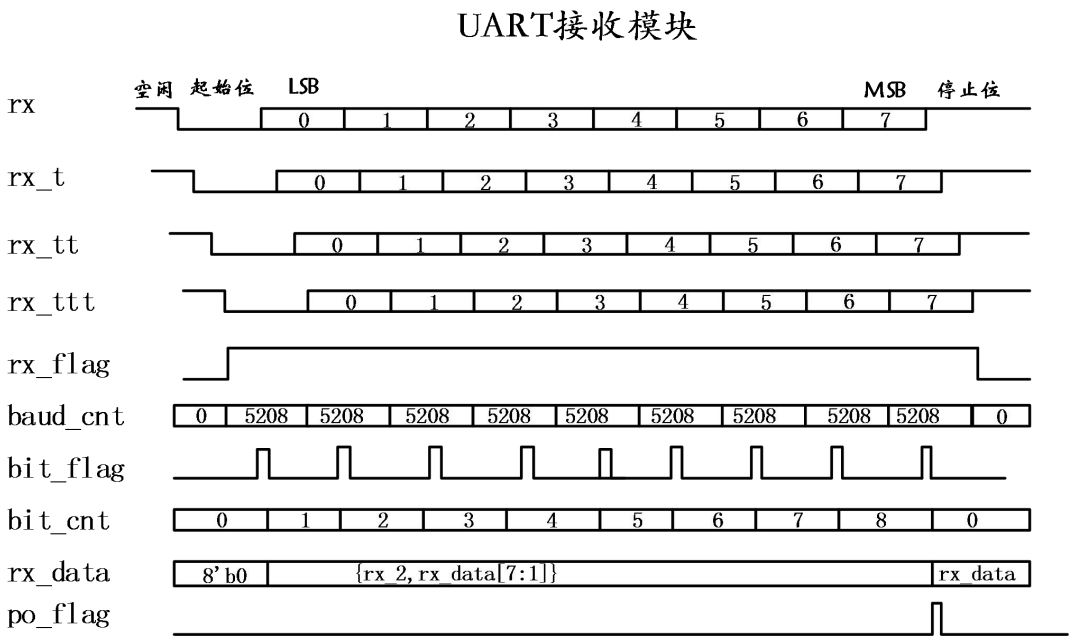


图 2　串口接收模块时序图

对于串口的收发模块设计，Kevin 以前也写过一篇文章专门对其进行介绍，只是当时的那篇文章只是单独介绍串口的，不像现在这样弄成了一个系列教程中的一部分。

好了，言归正传，我们还是进入到介绍串口接收模块时序的这个话题上来。

① rx 表示 PC 端的串口发送端（对于 FPGA 端，为串口的接收端），在串口空闲状态时，rx 一直处于高电平。若 PC 端需要通过串口发送数据，则需要将 rx 从高拉低，表示串口发送的起始位，接着依次发送 8bit 的数据（最低位优先），发送完最后 1bit 数据后，rx 保持高电平（无校验位）；

② 由于 PC 端与 FPGA 端处于两个时钟域，所以在 FPGA 端需要进行跨时钟域处理，即对 rx 进行打两拍处理（rx_t，rx_tt），rx_ttt 是 rx 的 3 级寄存器；

③ rx_flag 是串口处于接收状态的标志信号，rx_flag 拉高的条件是 rx 从空闲状态发送起始位的下降沿，拉低条件是接收完一帧串口数据（8bit）；

④ baud_cnt 为波特率计数器，FPGA 使用时钟为 50MHz，所以串口发送 1bit 的数据占用的时钟周期为：（ 1/9600）x 109÷20≈5208，该计数器自加的条件为 rx_flag 为高，当计数满 5207 时，自动清零；

⑤ bit_flag 为检测 rx 上串口数据的标志信号，只有当 baud_cnt 计数到 2603 时，bit_flag 才会拉高。选在 baud_cnt 计数器的中间值来检测串口数据，是为提高检测数据的可靠性；

⑥ bit_cnt 为已接收一帧串口数据的 bit 数，其自加条件为 bit_flag，当计数到 8，即表示接收完一帧数据后清零；

⑦ rx_data 是接收串口数据的寄存器，位宽为 8，该寄存器使用了位拼接操作，其目的是对接收到的串口数据进行移位，将接收到的 1bit 数据构造成 8bit 的数据，也可以理解成串转并的一个操作；

⑧ po_flag 为接收完一帧串口数据的标志信号，用于提供给外部模块检测接收到的串口数据，即 rx_data 寄存器的值。

三、串口接收模块代码编写

好了，串口接收模块的思路讲清楚了，那我们就可以根据设计好的时序图来编写代码了。

根据时序图，写出来的代码如下（建议大家先尝试自己把代码写出来，对于代码，在这不作解释，可以对照着配套视频教程进行学习）：

```
// ********************************************************************************
// Project Name :
// Author       : Kevin
// Email        : deng.kanwen@163.com
// Creat Time   : 2016/8/29 14:18:53
// File Name    :
// Module Name  : uart_rx
// Called By    :
// Abstract     :
//
// CopyRight(c) 2016, OpenSoc Studio..
// All Rights Reserved
//
// ********************************************************************************
```

```
// Modification History:
// 1. initial
//*****************************************************************************/
// *************************
// MODULE DEFINITION
// *************************

//`define SIM

module  uart_rx(
        // system signals
        input                   sclk                ,
        input                   s_rst_n             ,
        // UART Interface
        input                   rs232_rx            ,
        // others
        output  reg     [ 7:0]  rx_data             ,
        output  reg             po_flag
);

//===================================================================\
// ********** 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                             rx_r1                       ;
reg                             rx_r2                       ;
reg                             rx_r3                       ;
reg                             rx_flag                     ;
reg     [12:0]                  baud_cnt                    ;
reg                             bit_flag                    ;
reg     [ 3:0]                  bit_cnt                     ;
```

```
    wire                            rx_neg                          ;
//=============================================================================
// ***************      Main    Code     ****************
//=============================================================================

    assign  rx_neg  =       ~rx_r2 & rx_r3;

    always  @(posedge sclk) begin
            rx_r1   <=      rs232_rx;
            rx_r2   <=      rx_r1;
            rx_r3   <=      rx_r2;
    end

    always  @(posedge sclk or negedge s_rst_n) begin
            if(s_rst_n == 1'b0)
                    rx_flag <=      1'b0;
            else if(rx_neg == 1'b1)
                    rx_flag <=      1'b1;
            else if(bit_cnt == 'd0 && baud_cnt == BAUD_END)
                    rx_flag <=      1'b0;
    end

    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(rx_flag == 1'b1)
                    baud_cnt        <=      baud_cnt + 1'b1;
            else
                    baud_cnt        <=      'd0;
    end

    always  @(posedge sclk or negedge s_rst_n) begin
            if(s_rst_n == 1'b0)
                    bit_flag        <=      1'b0;
            else if(baud_cnt == BAUD_M)
```

```
                bit_flag        <=      1'b1;
        else
                bit_flag        <=      1'b0;
end

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

always  @(posedge sclk or negedge s_rst_n) begin
        if(s_rst_n == 1'b0)
                rx_data <=      'd0;
        else if(bit_flag == 1'b1 && bit_cnt >= 'd1)
                rx_data <=      {rx_r2, rx_data[7:1]};
end

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

endmodule
```

四、串口接收模块仿真验证

好了，串口接收模块的代码已经写好了，那如何利用 Modelsim 对这个模块进行仿真呢，如何来编写 TestBench 文件呢？

大家需要想清楚一点，我们仿真的意义是？仿真就是需要来验证设计模块的正确性，而在 TestBench 文件中就需要产生被仿真模块所需要的信号。**既然这是一个串口的接收模块，那我们就必须要产生一个串口发送的时序来验证这个模块。**

TestBench 文件源代码如下：

```
`timescale      1ns/1ns

module  tb_uart_rx;

reg             sclk;
reg             s_rst_n;
reg             rs232_tx;
wire            po_flag;
wire    [ 7:0]  rx_data;
reg     [ 7:0]  mem_a[3:0];

initial begin
        sclk    =       1;
        s_rst_n <=      0;
        rs232_tx<=      1;
        #100
        s_rst_n <=      1;
        #100
        tx_byte();
end

always  #5      sclk    =       ~sclk;

initial $readmemh("./tx_data.txt", mem_a);

task    tx_byte();
        integer i;
```

```
        for(i=0; i<4; i=i+1) begin
                tx_bit(mem_a[i]);
        end
endtask

task    tx_bit(
                input   [ 7:0]  data
                );
        integer i;
        for(i=0; i<10; i=i+1) begin
                case(i)
                        0:      rs232_tx        <=      1'b0;
                        1:      rs232_tx        <=      data[0];
                        2:      rs232_tx        <=      data[1];
                        3:      rs232_tx        <=      data[2];
                        4:      rs232_tx        <=      data[3];
                        5:      rs232_tx        <=      data[4];
                        6:      rs232_tx        <=      data[5];
                        7:      rs232_tx        <=      data[6];
                        8:      rs232_tx        <=      data[7];
                        9:      rs232_tx        <=      1'b1;
                endcase
                #560;
        end
endtask

uart_rx         uart_rx_inst(
        // system signals
        .sclk                   (sclk               ),
        .s_rst_n                (s_rst_n            ),
        // UART Interface
        .rs232_rx               (rs232_tx           ),
        // others
        .rx_data                (rx_data            ),
        .po_flag                (po_flag            )
);

endmodule
```

当然，对于在 TB 文件中产生串口发送的时序，还有很多种方法，Kevin 在这只是给大家提供这样的一种思路而已。

```
initial $readmemh("./tx_data.txt", mem_a);
```

可能代码中这一句大家见得比较少，这是读取外部 txt 文件中的十六进制数据，并将其作为存储器 mem_a（数组）的值。

对于仿真模块的详细解释，建议大家观看由【开源骚客】录制的本套教程的第二讲内容。

好了，仿真文件写好之后，大家就赶紧启动 Modelsim 进行仿真吧。

Kevin 就不对仿真的操作进行详细的介绍了。

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