串口收发实验

1 文档简介

本文主要讲解如何编写 FPGA 串口通信的收发程序，在程序中使用了状态机，是学习状态机的重要实验。

2 实验环境

- 黑金 FPGA 开发板（AXKU040开发板）
- 串口调试助手

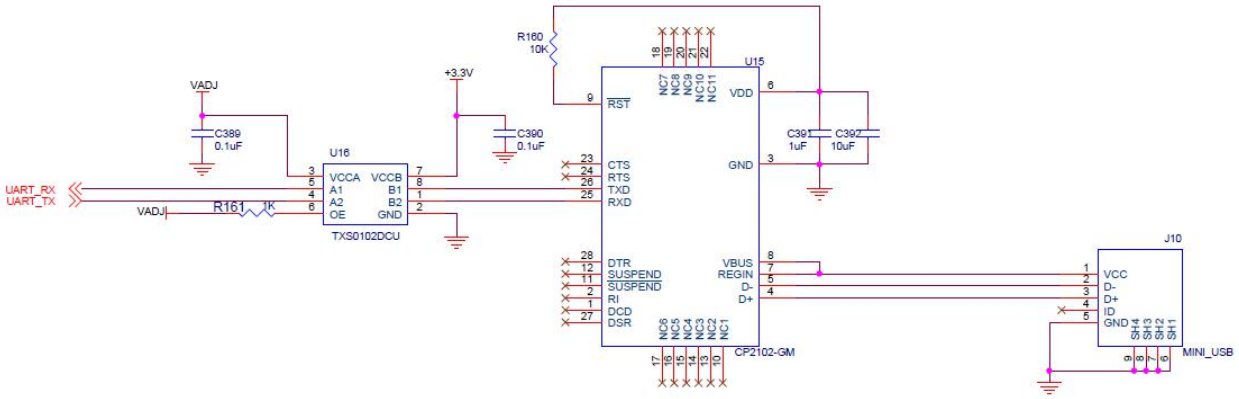
3 实验原理

3.1 串口通信简介

本文所述的串口指异步串行通信，异步串行是指 UART （Universal Asynchronous Receiver/Transmitter），通用异步接收/发送。UART 是一个并行输入成为串行输出的芯片，通常集成在主板上。UART 包含 TTL 电平的串口和 RS232 电平的串口。 TTL 电平是 3.3V 的，而 RS232 是负逻辑电平，它定义+5~+12V 为低电平，而-12~-5V 为高电平， MDS2710、 MDS SD4、 EL805 等是 RS232 接口，EL806 有 TTL 接口。

串行接口按电气标准及协议来分包括 RS-232-C、RS-422、RS485 等。RS-232-C、RS-422 与 RS-485 标准只对接口的电气特性做出规定，不涉及接插件、电缆或协议。

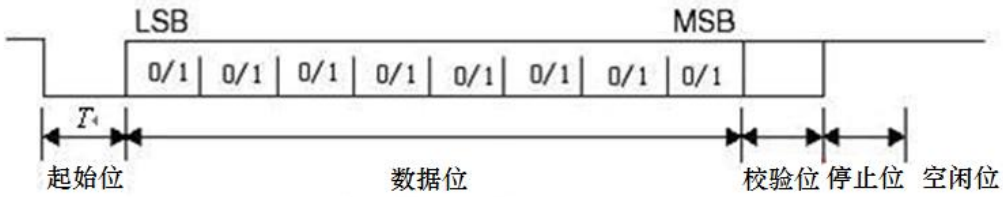
黑金 FPGA 开发板的串口通信通过 USB 转串口方式，主要是解决很多人电脑不带串口接口的问题，所以这里不涉及到电气协议标准，用法和 TTL 电平串口类似。FPGA 芯片使用 2 个 IO 口和 USB 转串口芯片 CP2102 相连。



AXKU040 开发板的 USB 转串口部分

3.2 异步串口通信协议

消息帧从一个低位起始位开始，后面是 7 个或 8 个数据位，一个可用的奇偶位和一个或几个高位停止位。接收器发现开始位时它就知道数据准备发送，并尝试与发送器时钟频率同步。如果选择了奇偶校验，UART 就在数据位后面加上奇偶位。奇偶位可用来帮助错误校验。在接收过程中，UART 从消息帧中去掉起始位和结束位，对进来的字节进行奇偶校验，并将数据字节从串行转换成并行。UART 传输时序如下图所示：



从波形上可以看出起始位是低电平，停止位和空闲位都是高电平，也就是说没有数据传输时是高电平，利用这个特点我们可以准确接收数据，当一个下降沿事件发生时，我们认为将进行一次数据传输。

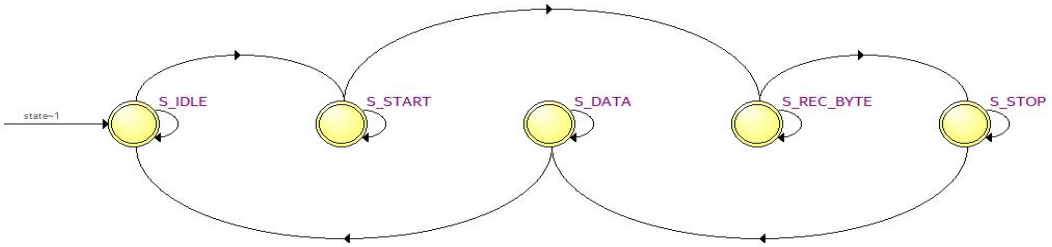
3.3 关于波特率

常见的串口通信波特率有 2400 、9600、115200 等，发送和接收波特率必须保持一致才能正确通信。波特率是指 1 秒最大传输的数据位数，包括起始位、数据位、校验位、停止位。假如通信波特率设定为 9600，那么一个数据位的时间长度是 1/9600 秒。

4 程序设计

4.1 接收模块设计

串口接收模块是个参数化可配置模块，参数“CLK_FRE”定义接收模块的系统时钟频率，单位是 Mhz，参数“BAUD_RATE”是波特率。接收状态机状态转换图如下：



“S_IDLE”状态为空闲状态，上电后进入“S_IDLE”，如果信号“rx_pin”有下降沿，我们认为是串口的起始位，进入状态“S_START”,等一个 BIT 时间起始位结束后进入数据位接收状态“S_REC_BYTE”,本实验中数据位设计是 8 位，接收完成以后进入“S_STOP”状态，在“S_STOP”没有等待一个 BIT 周期，***只等待了半个 BIT 时间***，这是因为如果等待了一个周期，有可能会错过下一个数据的起始位判断，最后进入“S_DATA”状态，将接收到的数据送到其他模块。在这个模块我们提一点：为了满足采样定理，在接受数据时每个数据都在波特率计数器的时间中点进行采样，以避免数据出错的情况：

```
//receive serial data bit data
always@(posedge clk or negedge rst_n)
begin
        if(rst_n == 1'b0)
                rx_bits <= 8'd0;
        else if(state == S_REC_BYTE && cycle_cnt == CYCLE/2 - 1)
                rx_bits[bit_cnt] <= rx_pin;
        else
                rx_bits <= rx_bits;
end
```

注意：***本实验没有设计奇偶校验位***。

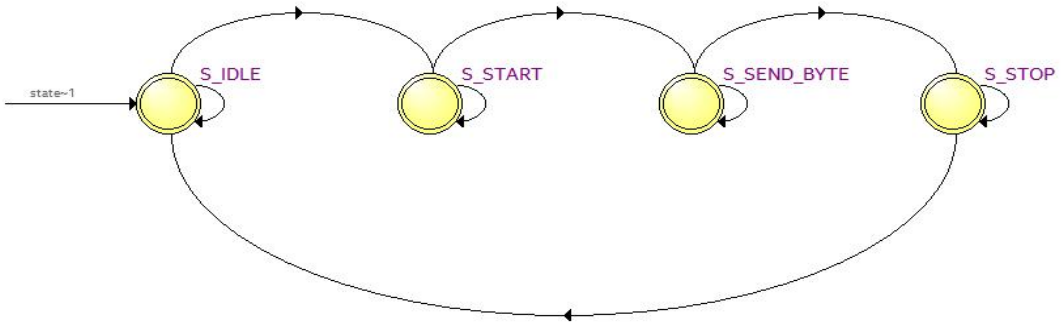
信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟

rst_n	in	1	异步复位，低电平复位
rx_data	out	8	接收到的串口数据（8 位数据）
rx_data_valid	out	1	接收到的串口数据有效（高有效）
rx_data_ready	in	1	可以接收数据，当 rx_data_ready 和 rx_data_valid 都为高时数据送出
rx_pin	in	1	串口接收数据输入

串口接收模块端口

4.2 发送模块设计

发送模式设计和接收模块相似，也是使用状态机，状态转换图如下：



上电后进入“S_IDLE”空闲状态，如果有发送请求，进入发送起始位状态“S_START”,起始位发送完成后进入发送数据位状态“S_SEND_BYTE”,数据位发送完成后进入发送停止位状态“S_STOP”,停止位发送完成后又进入空闲状态。在数据发送模块中，从顶层模块写入的数据直接传递给寄存器‘tx_reg’，并通过‘tx_reg’寄存器模拟串口传输协议在状态机的条件转换下进行数据传送：

```
always@(posedge clk or negedge rst_n)
begin
        if(rst_n == 1'b0)
                tx_reg <= 1'b1;
        else
                case(state)
                        S_IDLE,S_STOP:
                                tx_reg <= 1'b1;
                        S_START:
                                tx_reg <= 1'b0;
                        S_SEND_BYTE:
                                tx_reg <= tx_data_latch[bit_cnt];
                        default:
                                tx_reg <= 1'b1;
                endcase
end
```

信号名称	方向	宽度 (bit)	说明
clk	in	1	系统时钟
rst_n	in	1	异步复位，低电平复位
tx_data	in	8	要发送的串口数据(8 位数据)
tx_data_valid	in	1	发送的串口数据有效（高有效）
tx_data_ready	out	1	可以发送数据，当 tx_data_ready 和 tx_data_valid 都为高时数据被发送
tx_pin	out	1	串口发送数据发送

串口发送模块端口

4.3 测试程序

测试程序设计 FPGA 为 1 秒向串口发送一次“HELLO ALINX\r\n”,不发送期间，如果接受到串口数据，直接把接收到的数据送到发送模块再返回。 “\r\n”,在这里和 C 语言中表示一致，都是回车换行。

测试程序分别例化了发送模块和接收模块，同时将参数传递进去，波特率设置为 115200。

```
always@(posedge sys_clk or negedge rst_n)
begin
    if(rst_n == 1'b0)
    begin
        wait_cnt <= 32'd0;
        tx_data <= 8'd0;
        state <= IDLE;
        tx_cnt <= 8'd0;
        tx_data_valid <= 1'b0;
```

```
    end
    else
    case(state)
        IDLE:
            state <= SEND;
        SEND:
        begin
            wait_cnt <= 32'd0;
            tx_data <= tx_str;

            if(tx_data_valid == 1'b1 && tx_data_ready == 1'b1 && tx_cnt
< 8'd12)//Send 12 bytes data
            begin
                tx_cnt <= tx_cnt + 8'd1; //Send data counter
            end
            else if(tx_data_valid && tx_data_ready)//last byte sent is complete
            begin
                tx_cnt <= 8'd0;
                tx_data_valid <= 1'b0;
                state <= WAIT;
            end
            else if(~tx_data_valid)
            begin
                tx_data_valid <= 1'b1;
            end
        end
        WAIT:
        begin
            wait_cnt <= wait_cnt + 32'd1;

            if(rx_data_valid == 1'b1)
            begin
                tx_data_valid <= 1'b1;
                tx_data <= rx_data;    // send uart received data
            end
            else if(tx_data_valid && tx_data_ready)
            begin
                tx_data_valid <= 1'b0;
            end
            else if(wait_cnt >= CLK_FRE * 1000000)  // wait for 1 second
                state <= SEND;
        end
        default:
            state <= IDLE;
    endcase
end

//combinational logic
//Send "HELLO ALINX\r\n"
always@(*)
begin
    case(tx_cnt)
        8'd0 :  tx_str <= "H";
        8'd1 :  tx_str <= "E";
        8'd2 :  tx_str <= "L";
        8'd3 :  tx_str <= "L";
        8'd4 :  tx_str <= "O";
        8'd5 :  tx_str <= " ";
        8'd6 :  tx_str <= "A";
```

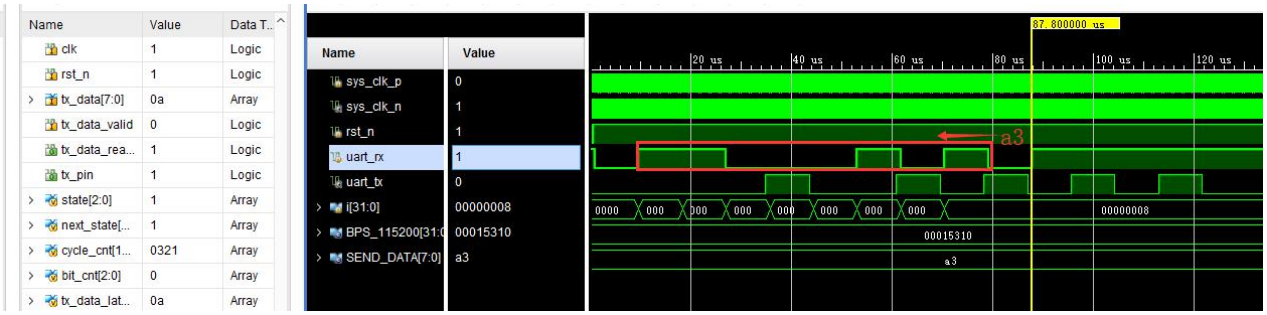
```
            8'd7 :  tx_str <= "L";
            8'd8 :  tx_str <= "I";
            8'd9 :  tx_str <= "N";
            8'd10:  tx_str <= "X";
            8'd11:  tx_str <= "\r";
            8'd12:  tx_str <= "\n";
            default:tx_str <= 8'd0;
        endcase
end

uart_rx#
(
    .CLK_FRE(CLK_FRE),
    .BAUD_RATE(115200)
) uart_rx_inst
(
    .clk                        (sys_clk                    ),
    .rst_n                      (rst_n                      ),
    .rx_data                    (rx_data                    ),
    .rx_data_valid              (rx_data_valid              ),
    .rx_data_ready              (rx_data_ready              ),
    .rx_pin                     (uart_rx                    )
);

uart_tx#
(
    .CLK_FRE(CLK_FRE),
    .BAUD_RATE(115200)
) uart_tx_inst
(
    .clk                        (sys_clk                    ),
    .rst_n                      (rst_n                      ),
    .tx_data                    (tx_data                    ),
    .tx_data_valid              (tx_data_valid              ),
    .tx_data_ready              (tx_data_ready              ),
    .tx_pin                     (uart_tx                    )
);
```

5 仿真

这里我们添加了一个串口接收的激励程序 vtf_uart_test.v 文件，用来仿真 uart 串口接收。这里向串口模块的 uart_rx 发送 0xa3 的数据, 每位的数据按 115200 的波特率发送，1 位起始位，8 位数据位和 1 位停止位。

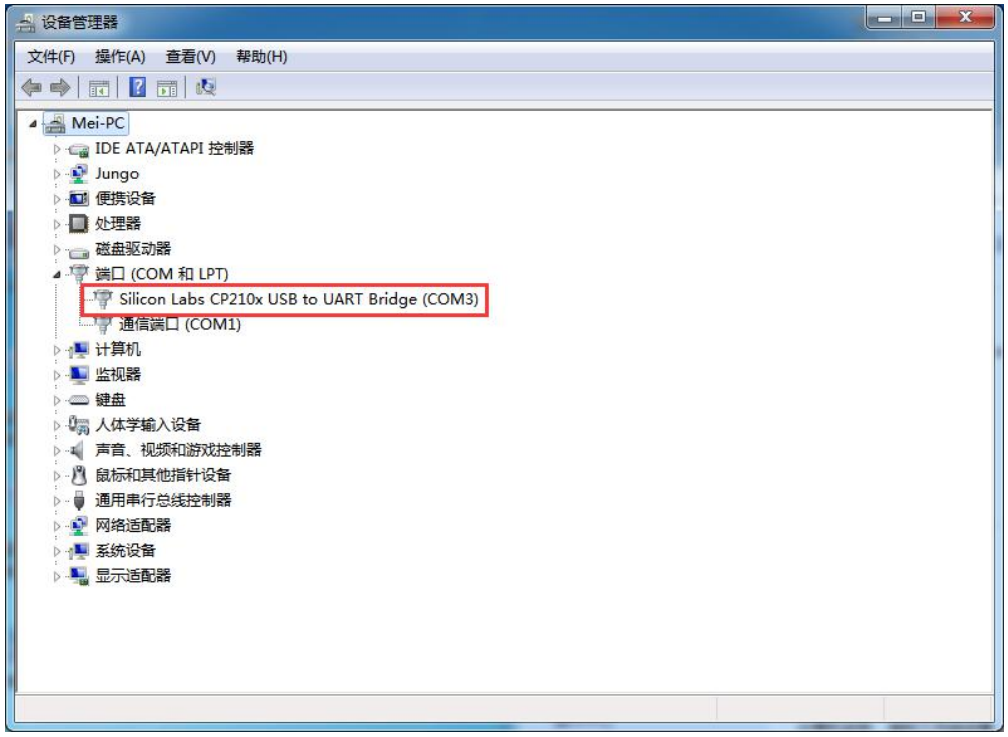
```
    // Wait 1000 ns for global reset to finish
    #1000;
      rst_n = 1;
    // Add stimulus here
    #2000000;
  //  $stop;
end

always #25 sys_clk_p = ~ sys_clk_p;    //5ns一个周期，产生200MHz时钟源
assign sys_clk_n=~sys_clk_p;

parameter                       BPS_115200 = 86800;//每个比特的时间
parameter                       SEND_DATA = 8'b1010_0011;//

integer i = 0;

 initial begin
   uart_rx = 1'b1;    //bus idle
   #10000 uart_rx = 1'b0;      //stranmit start bit

   for (i=0;i<8;i=i+1)
   #BPS_115200 uart_rx = SEND_DATA[i];      //stranmit data bit

   #BPS_115200 uart_rx = 1'b0;      //stranmit stop bit
   #BPS_115200 uart_rx = 1'b1;      //bus idle

  end
```

仿真的结果如下，当程序接收到 8 位数据的时候，rx_data[7:0]的数据位 a3,在接收到数据后，uart_tx 会不断地发出接收到的数据。



6 实验测试

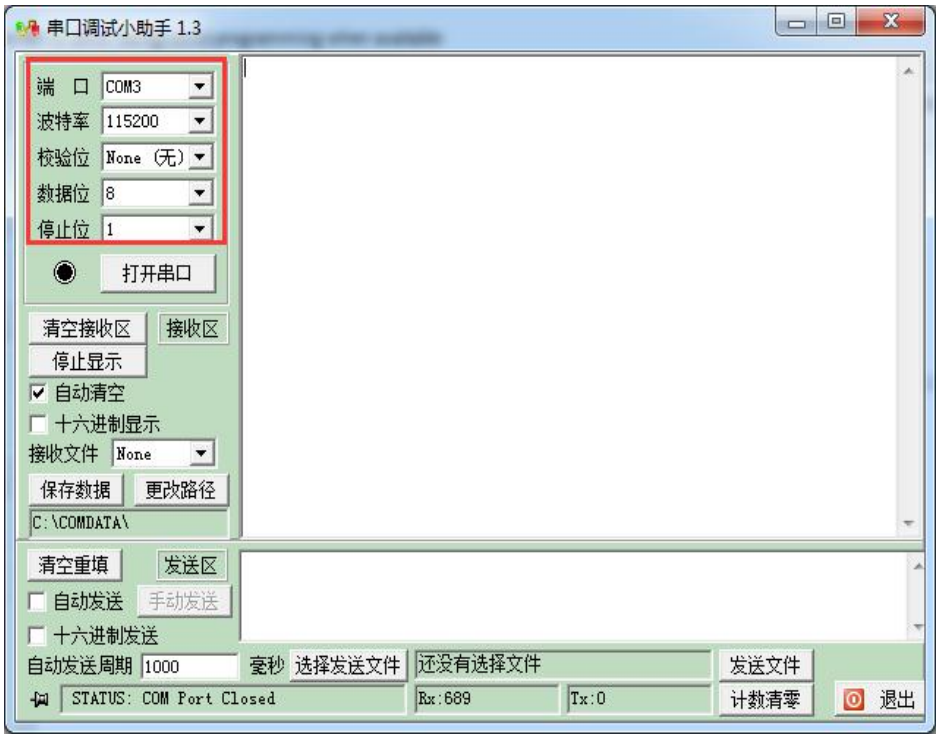
由于开发板的串口使用 USB 转串口芯片，首先要安装串口驱动程序，正确安装驱动状态如下图所示（当然要连接串口的 USB 到电脑)。如果没有正确连接请参考本文附录“串口驱动的安装”。



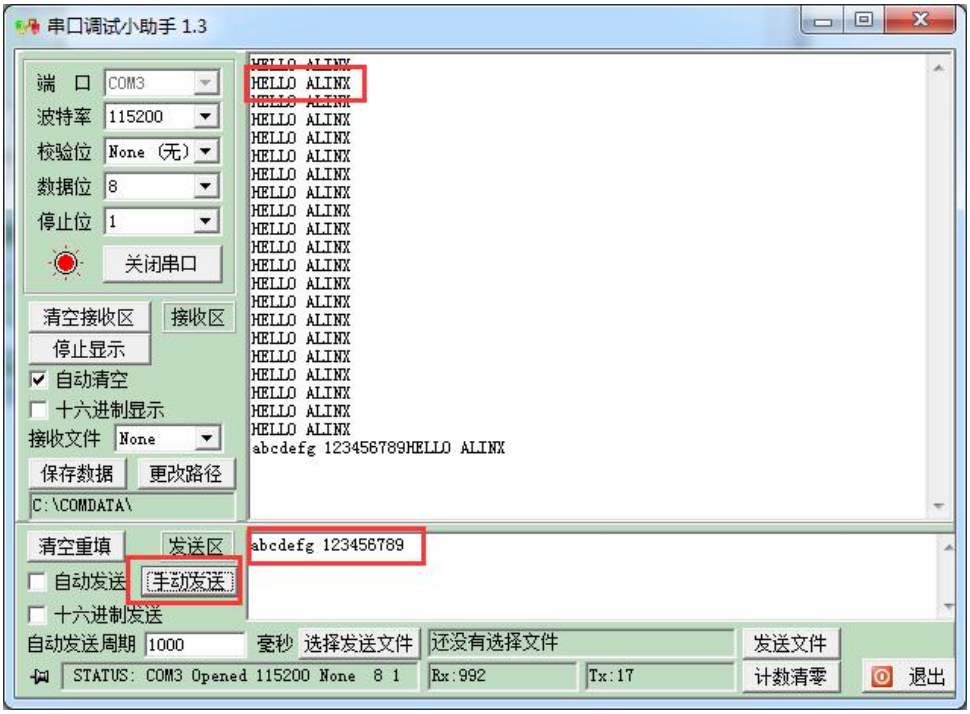
串口驱动正常的状态

从图中可以看出系统给串口分配的串口号是“COM3”，串口号的分配是系统完成的，自动分配情况下每台电脑可能会有差异，笔者这里是“COM3”，使用串口号时要根据自己的分配情况选择。

打开串口调试，端口选择“COM3”（根据自己情况选择），波特率设置 115200，检验位选 None，数据位选 8，停止位选 1，然后点击“打开串口”。如果找不到这个小软件使用 windows 搜索功能，在黑金给的资料文件夹里搜索“串口调试”。



打开串口以后，每秒可收到“HELLO ALINX”，在发送区输入框输入要发送的文字，点击“手动发送”，可以看到接收到自己发送的字符。



7 附录

串口驱动安装

没有安装驱动插入 usb 转串口以后设备管理器下会出现如下情况：

驱动程序的安装文件可以在我们提供的资料里的“软件工具及驱动\USB 转串口驱动”目录下找到，如果操作系统是 32 位的用户双击 CP210x_VCPInstaller_x86.exe 开始安装; 如果操作系统是 64 位的用户双击 CP210x_VCPInstaller_x64.exe 开始安装;

驱动安装成功后，再打开“设备管理器”， 打开“端口(COM 和 LPT)”，会出现对应的 COM Number。分配的编号由系统决定。