实验四：串口收发器设计

漆耘含

无 63

2016011058

一、 实验目的

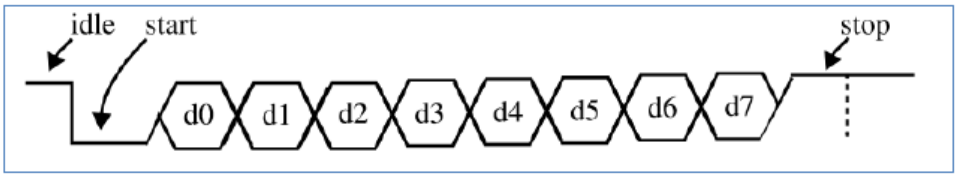
了解和掌握 UART 的工作原理

二、 实验原理

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。



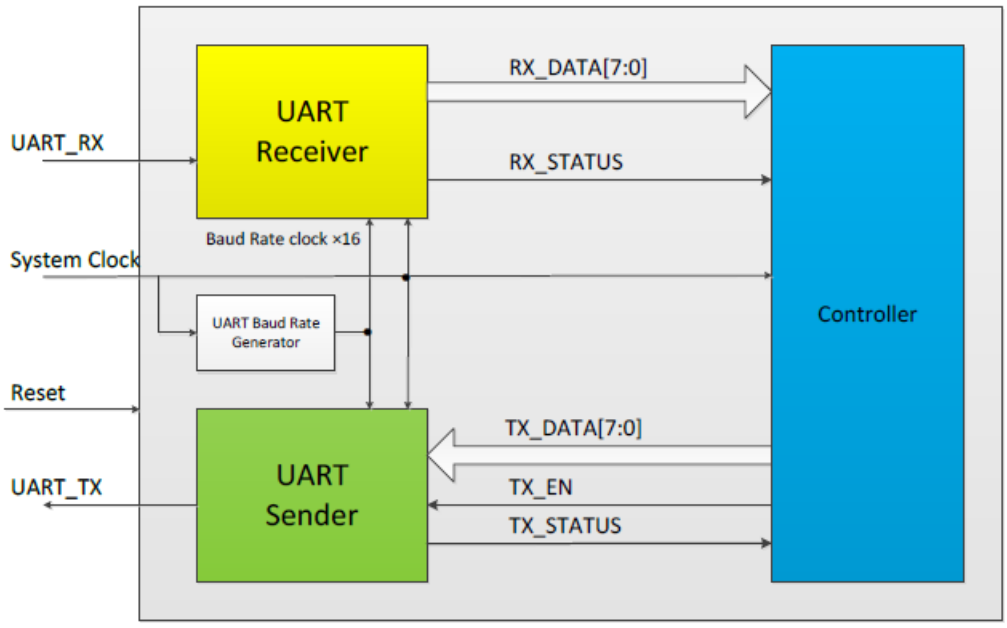
上图表明在异步传送中串行发送一个数据字节的位定时关系(图中没有包括奇偶校验位)。发送一个完整的字节信息，首先是一个作为起始位的逻辑“0”位，接着是8个数据位，然后是1个、1＋1/2个或2个停止位逻辑“1”位，数据线空闲时呈现为高或“1”状态。在字符的8位数据部分，先发送数据的最低位，最后发送最高位。每位持续的时间是

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。

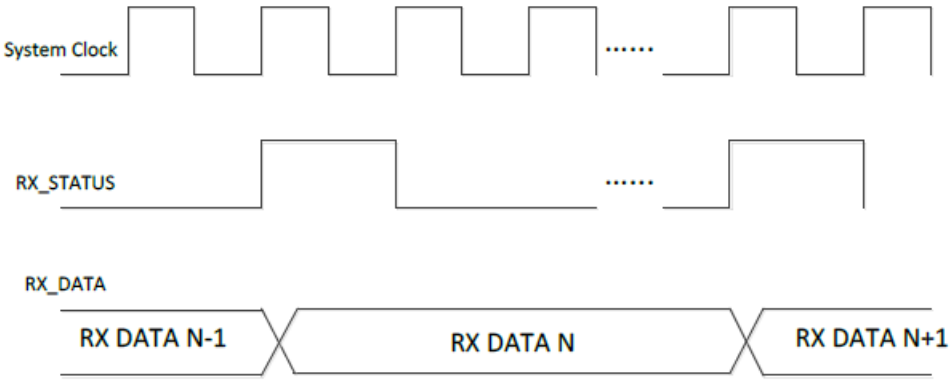
2. 实验设计原理

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。



a. 串口接收模块（receiver）

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 上的接收数据一致有效到下一个 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：



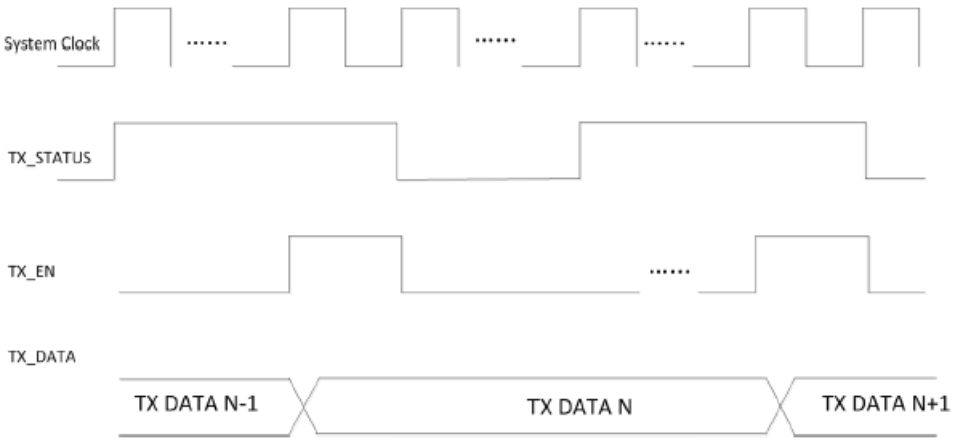
由于串行数据帧与接收时钟是异步的，所以接收器功能实现中的关键是接收器时钟与每个接收字符的同步。一个有效的方法是接收器采用高速率时钟对串行数据进行采样，通常采样频率是位时钟频率的整数倍。理论上倍数越高接收数据各位的分辨率越高，实际中，一般最大选择16 倍。波特率发生器(Baud Rate Generator)模块负责根据System clock 时钟产生所需的16 倍（或者其他倍数）波特率的接收时钟。

接收器应该尽可能地在靠近位周期的中心处对每位采样。如果接收器能很好地预测起始位的开始，那么他可在起始位的下降沿到来之后，等待半个位周期再采样数据位。此后，接收器每等待一个位周期采样一个数据位，直至收到最后一位为止。倘若接收时钟的频率足够接近发送时钟，使得最后位能在离该位的精确中心位置半个周期内对他采样，以上方案就能正确地工作。这意味着接收时钟相对于发送时钟在 10～11 个时钟周期内，其增加和减少应小于半个位的时间间隔。因此，要求收发双方 2 个时钟的误差容限在 5%以内。

我的接收模块的时钟是系统 50M 的时钟，在模块里面进行分频，产生 16 倍波特率的时钟，有计数寄存器 count 来计数，在开始判断位来的时候开始计数，当计数到 16 的时候，说明发来了一个序列需要接收，这是 receive_enable=1，count 归 0 重新计数，当计数到 128+16（停止位）时，receive_enable=0，接收结束。采样信号生成是在判断位 count 计数到 7 时，采样信号计数从 0 开始，每计数 16，产生一个采样信号，之后就可根据采样信号对输入信号进行采样。

b. 发送模块（sender）

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：



我的发送模块比较简单，输入的时钟是系统时钟 50MHz，在模块内实现分频，产生波特率为 9600Hz 的时钟，在可以发送的时候，每过 9600Hz 的一个周期就输出一位。

c. 控制模块

控制模块是接收到输入进来的序列，产生需要发送的序列和相应的控制信号，如果输入的最高位是 1，则将序列按位取反，如果最高位是 0，则直接发送。

三、 关键代码及文件清单

1. 文件清单：

chuankou.v

receiver.v

sender.v

controller.v

2. 关键代码及仿真：

a. Chuankou.v:

```
module chuankou(sys_clk,UART_RX,reset,UART_TX);
  input sys_clk,UART_RX,reset;
  output UART_TX;


  wire [7:0]RX_DATA;
  wire RX_STATUS;
  wire [7:0]TX_DATA;
  wire TX_EN;
  wire TX_STATUS;


  receiver r(.sys_clk(sys_clk),.UART_Rx(UART_RX),.RX_data(RX_DATA),.RX_status(RX_STATUS));
  controller c(.sys_clk(sys_clk),.RX_data(RX_DATA),.RX_status(RX_STATUS),.TX_data(TX_DATA),
               .TX_en(TX_EN),.TX_status(TX_STATUS));
  sender s(.sys_clk(sys_clk),.reset(reset),.TX_data(TX_DATA),.TX_en(TX_EN),
           .TX_status(TX_STATUS),.UART_tx(UART_TX));

endmodule // chuankou
```

模块分析：

这个模块是串口的顶层模块，将下属的三个模块连接起来。

仿真：

由上图仿真可见，输入到输出是相同的，最高位是 0，满足要求，同时采样信号也是正确的，在每一个的中点进行采样。

b. Receiver.v

```
1   module receiver(sys_clk,UART_Rx,RX_data,RX_status);
2     input sys_clk;
3     input UART_Rx;
4     output [7:0]RX_data;
5     output RX_status;
6
7     reg [7:0]count;
8     reg receive_enable;
9     reg [10:0]signal_count;
10    reg [10:0]resignal_count;
11    reg cai_signal;
12    reg enable;
13    reg [4:0]digital;
14    reg [7:0]RX_data_b;
15    reg RX_status_b;
16    reg [15:0]cnt;
17
18    initial
19    begin
20      count<=0;
21      receive_enable<=0;
22      signal_count<=0;
23      resignal_count<=0-10'd14;
24      cai_signal<=0;
25      enable<=0;
26      digital<=0;
27      RX_data_b<=0;
28      RX_status_b<=0;
29      cnt<=0;
30    end
```

```
33    assign RX_data=RX_data_b;
34
35    always@(posedge sys_clk)
36    begin
37      if(cnt==16'd324)
38      begin
39        cnt<=0;
40        count<=count+1;
41
42
43        if(~receive_enable)
44          begin
45            if(~UART_Rx)
46              begin
47                if(count==8'd15)
48                begin
49                  receive_enable<=1'b1;
50                  count<=0;
51                  cnt<=0;
52                end
53              end
54            else
55              begin
56                enable<=0;
57                count<=0;
58                cnt<=0;
59              end
60
61            if(count==8'd7)
62              enable<=1;
63          end
```

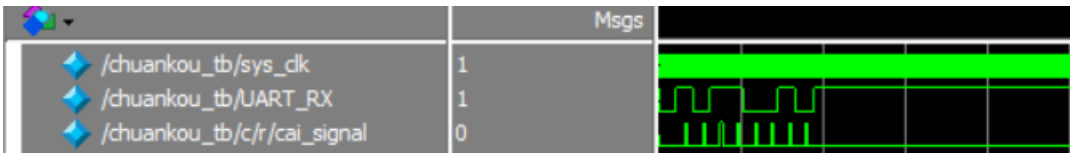
```
64        else
65          begin
66            if(enable==1)
67              begin
68                if(signal_count==resignal_count+11'd15)
69                  begin
70                    cai_signal<=1'b1;
71                  end
72                if(signal_count==resignal_count+11'd17)
73                  begin
74                    cai_signal<=1'b0;
75                    resignal_count<=signal_count-11'd1;
76                  end
77
78                if(count<=8'd7)
79                  signal_count<=0;
80                else
81                  signal_count<=count-11'd7;
82              end
83            if(count==8'd128)
84              begin
85                enable<=0;
86                RX_status_b<=1;
87              end
88            else
89              begin
90                if(cai_signal==1)
91                  begin
92                  RX_data_b[digital]<=UART_Rx;
93                  digital<=signal_count/8'd16+1;
94                end
95              end
```

```
96              if(count>=8'd128)
97                begin
98                  if(count==8'd128+8'd16)
99                    begin
100                     count<=0;
101                     receive_enable<=0;
102                     signal_count<=0;
103                     resignal_count<=0-10'd14;
104                     cai_signal<=0;
105                     enable<=0;
106                     digital<=0;
107                     RX_data_b<=0;
108                     RX_status_b<=0;
109                   end
110               end
111           end
112       end
113       else
114         cnt<=cnt+1;
115     end
116
117
118 endmodule
```

模块分析：

在判断位的时候，当 count 计数到 7，采样计数信号就开始计数，每过 16 输出一个采样脉冲信号之后可根据采样信号进行采样。在 count 第一次计数到 16，即判断位 0 持续一个周期，则表明有序列发过来，则 count 归 0，重新计数到 128+16（停止位）。在没有序列发过来的时候，不允许计数。所有的 count 为 0。

仿真：



由仿真可以看出，在第一个判断位为 0 的时候是没有采样脉冲的，在后面连续 8 位有采样脉冲，最后的停止位也没有采样脉冲。符合理论分析和设计要求。

c. Sender.v

d.

```
module sender(sys_clk,reset,TX_data,TX_en,TX_status,UART_tx);
  input sys_clk,TX_en,reset;
  input [7:0]TX_data;
  output TX_status;
  output UART_tx;


  reg [3:0]digital;
  reg TX_status_b;
  reg [7:0]save_data;
  reg TX_send;
  reg UART_tx_b;
  reg [15:0]cnt1;

  assign UART_tx=UART_tx_b;
  assign TX_status=TX_status_b;

  initial
  begin
    digital<=0;
    UART_tx_b<=1;
    TX_status_b<=0;
    save_data<=0;
    TX_send<=0;
    cnt1<=0;
  end
```

```
  always@(posedge sys_clk or negedge reset)
  begin
    if(~reset)
      begin
        digital<=0;
        UART_tx_b<=1;
        save_data<=0;
        TX_send<=0;
      end
    else
      begin
        if(TX_en&&(~TX_send))
          begin
            TX_send<=1;
            save_data [7:0]<=TX_data[7:0];
          end
        else if (TX_send)
          begin
            TX_status_b<=1;
            if(cnt1==15'd5208)
            begin
              cnt1<=0;
              digital<=digital+1;
```

```
52                  case(digital)
53                  4'd0:begin
54                      UART_tx_b<=0;
55                      TX_status_b<=0;
56                    end
57                  4'd1:UART_tx_b<=save_data[0];
58                  4'd2:UART_tx_b<=save_data[1];
59                  4'd3:UART_tx_b<=save_data[2];
60                  4'd4:UART_tx_b<=save_data[3];
61                  4'd5:UART_tx_b<=save_data[4];
62                  4'd6:UART_tx_b<=save_data[5];
63                  4'd7:UART_tx_b<=save_data[6];
64                  4'd8:UART_tx_b<=save_data[7];
65                  4'd9:UART_tx_b<=1;
66                  4'd10:begin
67                      digital<=0;
68                      TX_status_b<=0;
69                      TX_send<=0;
70                      end
71                  endcase // digital
72                end
73                else
74                  cnt1<=cnt1+1;
75              end
76          end
77      end
78  endmodule // sender
```

模块分析：

在模块内产生了频率为 9600Hz 的时钟，在允许发送的时候，每隔一个周期发送一个数据。

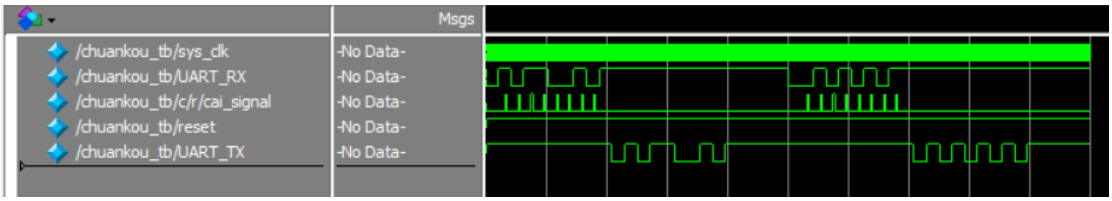
e. Controller.v

```
module controller(sys_clk,RX_data,RX_status,TX_data,TX_en,TX_status);
  input [7:0]RX_data;
  input RX_status,sys_clk;
  output [7:0]TX_data;
  output TX_en;
  input TX_status;

  reg [7:0]TX_d;
  reg TX_en_b;

  initial
    begin
      TX_en_b<=0;
      TX_d<=0;
    end

  assign TX_data=TX_d;
  assign TX_en=TX_en_b;

  always@(posedge sys_clk)
  begin
    if(RX_status)
      begin
        if(RX_data[7])
          TX_d<=~RX_data;
        else
          TX_d<=RX_data;
      end

    if(RX_status)
      TX_en_b<=1;
    else if(~TX_status)
      TX_en_b<=0;
  end

endmodule // controller
```

模块分析：

在拿到接收模块接收到的信息，对序列进行处理，如果最高位为 1，则序列反转，如果最高位为 0，则送到发射模块进行输出。

四、 顶层模块仿真结果



分析：

第一个序列最高位是 0，则将原序列发送回去，第二个序列最高位是 1，则将序列反转之后再发送，在接收模块中产生了 8 个采样脉冲，满足设计要求，仿真结果正确。

五、 综合情况

Flow Summary	
Flow Status	Successful - Fri Jun 01 15:37:04 2018
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	chuankou
Top-level Entity Name	chuankou
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	215 / 33,216 (< 1 %)
Total combinational functions	212 / 33,216 (< 1 %)
Dedicated logic registers	98 / 33,216 (< 1 %)
Total registers	98
Total pins	4 / 475 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

TimeQuest Timing Analyzer Summary	
Quartus II Version	Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Web Edition
Revision Name	chuankou
Device Family	Cyclone II
Device Name	EP2C35F672C6
Timing Models	Final
Delay Model	Combined
Rise/Fall Delays	Unavailable

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	170.59 MHz	170.59 MHz	CLK_50MHz	

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	UART_RX	CLK_50MHz	4.713	4.713	Rise	CLK_50MHz
2	reset	CLK_50MHz	4.802	4.802	Rise	CLK_50MHz

六、 实验总结

本次实验实现的是串口收发器，模块关系相对简单。在编写代码的时候，我添加了一个产生两种频率的模块，分别产生 9600Hz 和 9600*16Hz 的时钟，分别输入到 sender 和 receiver 模块中，因为我编写程序问题，仿真没有问题，在 Quartus 里面显示使用了多驱动，后来把两个模块融合在一起写才没有多驱动。我觉得本实验最难的部分就是接收模块，关键是采信号要生成好，以及要和接收的序列同步，并且接收完之后要都清零同步，否则在后面输入较多序列的时候，会出现采样错误的问题。

通过这次实验，我对串口通信有了更深刻的认识，对 UART 的机制进行初步的了解，收获很多！