浙江工业大学

《可编程逻辑器件及应用》
课程实验报告

学生姓名 凌智城

指导教师 龚树凤

专业班级 通信工程 1803 班

培养类别 全日制本科

所在学院 信息工程学院

提交日期 2021 年 1 月 4 日

（完成正文后，再根据实际页码对目录页码进行补正）

目　录

（目录内容可以根据个人实际情况，适当调整）

实验三：UART 串口发送/接收器的设计与仿真

3.1 设计任务

设计实现 UART 串口发送/接收器。按要求完成代码设计及验证工作

3.2 设计方案

由于 UART 是异步传输，没有传输同步时钟。为了能保证数据传输的正确性，UART 采用 16 倍数据波特率的时钟进行采样。每个数据有 16 个时钟采样，取中间的采样值，以保证采样不会滑码或误码。一般 UART 一帧的数据位数为 8，这样即使每个数据有一个时钟的误差，接收端也能正确地采样到数据。

整个 UART 的同步时钟频率为 16*f1，用系统时钟获取时，f1=9600 时，设计分频系数 N=f0/(16*f1)=325.52，取整为 326

UART 接收模块的功能：时时检测线路，当线路产生下降沿时，即认为线路有数据传输，启动接收数据进程进行接收，按从低位到高位接收数据。

UART 发送模块的功能：接收到发送指令后，把数据按 UART 协议输出，先输出一个低电平的起始位，然后从低到高输出 8 个数据位，接着是可选的奇偶校验位，最后是高电平的停止位。

UART 的接收数据时序为：当检测到数据的下降沿时，表明线路上有数据进行传输，这时计数器 CNT 开始计数，当计数器为 24=16+8 时，采样的值为第 0 位数据；当计数器的值为 40 时，采样的值为第 1 位数据，依此类推，进行后面 6 个数据的采样。如果需要进行奇偶校验，则当计数器的值为 152 时，采样的值即为奇偶位；当计数器的值为 168 时，采样的值为“1”表示停止位，一帧数据接收完成。本节章将按上面的算法进行 Verilog HDL 语言建模与仿真。

学号为 201806061211，末位奇数，采用奇校验的方式

3.3Verilog HDL 源代码

1) 波特率分频模块

a) uartclk.v

```
//整个 UART 的同步时钟频率为 16*f1，用系统时钟获取时，f1=9600 时，设计分频系数
N=f0/(16*f1)=325.52，取整为 326
//凌智城  201806061211  通信工程 1803
module clkdiv(clk, rst,clkout);
input clk;              //系统时钟
input rst;              //复位信号
output clkout;          //采样时钟输出
reg clkout;
reg [15:0] cnt;
parameter N=326;

always @(posedge clk)     //分频进程
      if(!rst)
```

```
            begin
                cnt<=1'b0;
                clkout<=1'b0;
            end
        else if(N%2==0)
            begin
                if(cnt<N/2-1)
                    begin
                        cnt<=cnt+1'b1;
                    end
                else
                    begin
                        cnt<=1'b0;
                        clkout<=~ clkout;
                    end
            end

endmodule
```

b) uartclk.vt

```
//凌智城 201806061211 通信工程 1803
`timescale 1 ps/ 1 ps
module clkdiv_vlg_tst();
reg clk;
reg rst;
wire clkout;

// assign statements (if any)
clkdiv i1 (
// port map - connection between master ports and signals/registers
    .clk(clk),
    .clkout(clkout),
    .rst(rst)
);
initial clk=0;
always
    begin
        #5 clk=1'b1;
        #5 clk=1'b0;
    end
initial
    begin
        rst=0;
        #20 rst=1;
        #8000 $finish;
```

```
        end
endmodule
```

2) 接收模块

a) uartrx.v

```
//凌智城 201806061211 通信工程 1803
`timescale 1ns / 1ps
module uartrx(clk, rst,rxd, rxd_data, rxd_en, dataerror, frameerror);
input clk;                  //采样时钟
input rst;                  //复位信号
input rxd;                   //UART 数据输入

output rxd_data;             //接收数据输出
output rxd_en;               //接收数据有效，高说明接收到一个字节
output dataerror;           //数据出错指示
output frameerror;          //帧出错指示

reg[7:0] rxd_data;
reg rxd_en, dataerror;
reg frameerror;
reg [7:0] cnt;
reg rxbuf, rxfall, receive;
parameter paritymode = 1'b1;
reg presult, idle;          //presult 奇偶校验位;idle 线路状态指示，高 1 为线路忙，低 0 为线路空闲

//当线路产生下降沿时，即认为线路有数据传输
always @(posedge clk)
begin
   rxbuf <= rxd;
   rxfall <= rxbuf & (~rxd);
end

//检测到线路的下降沿并且原先线路为空闲，启动接收数据进程
always @(posedge clk)
begin
   if (rxfall && (~idle))
       begin
            receive <= 1'b1;          //开始接收数据
       end
   else if(cnt == 8'd168)    //接收数据完成
       begin
            receive <= 1'b0;
       end
end
```

```
//使用 176 个时钟接收一个数据（起始位、8 位数据、奇偶校验位、停止位），每位占用 16 个时钟//
always @(posedge clk or negedge rst)
begin
  if (!rst)
    begin
      idle<=1'b0;
      cnt<=8'd0;
      rxd_en <= 1'b0;
      frameerror <= 1'b0;
      dataerror <= 1'b0;
      presult<=1'b0;
    end
  else if(receive == 1'b1)
    begin
    case (cnt)
    8'd0:                    //0~15 个时钟为接收第一个比特，起始位
      begin
        idle <= 1'b1;
        cnt <= cnt + 8'd1;
        rxd_en <= 1'b0;
      end
    8'd24:                   //16~31 个时钟为第 1 个 bit 数据，取中间第 24 个时钟的采样值
      begin
        idle <= 1'b1;
        rxd_data[0] <= rxd;
        presult <= paritymode^rxd;
        cnt <= cnt + 8'd1;
        rxd_en <= 1'b0;
      end
    8'd40:                   //47~32 个时钟为第 2 个 bit 数据，取中间第 40 个时钟的采样值
      begin
        idle <= 1'b1;
        rxd_data[1] <= rxd;
        presult <= presult^rxd;
        cnt <= cnt + 8'd1;
        rxd_en <= 1'b0;
      end
    8'd56:                   //63~48 个时钟为第 3 个 bit 数据，取中间第 56 个时钟的采样值
      begin
        idle <= 1'b1;
        rxd_data[2] <= rxd;
```

```
            presult <= presult^rxd;
            cnt <= cnt + 8'd1;
            rxd_en <= 1'b0;
        end
    8'd72:                  //79~64 个时钟为第 4 个 bit 数据，取中间第 72 个时钟的采样值
        begin
            idle <= 1'b1;
            rxd_data[3] <= rxd;
            presult <= presult^rxd;
            cnt <= cnt + 8'd1;
            rxd_en <= 1'b0;
        end
    8'd88:                  //95~80 个时钟为第 5 个 bit 数据，取中间第 88 个时钟的采样值
        begin
            idle <= 1'b1;
            rxd_data[4] <= rxd;
            presult <= presult^rxd;
            cnt <= cnt + 8'd1;
            rxd_en <= 1'b0;
        end
    8'd104:                 //111~96 个时钟为第 6 个 bit 数据，取中间第 104 个时钟的采样值
        begin
            idle <= 1'b1;
            rxd_data[5] <= rxd;
            presult <= presult^rxd;
            cnt <= cnt + 8'd1;
            rxd_en <= 1'b0;
        end
    8'd120:                 //127~112 个时钟为第 7 个 bit 数据，取中间第 120 个时钟的采样值
        begin
            idle <= 1'b1;
            rxd_data[6] <= rxd;
            presult <= presult^rxd;
            cnt <= cnt + 8'd1;
            rxd_en <= 1'b0;
        end
    8'd136:                 //143~128 个时钟为第 8 个 bit 数据，取中间第 136 个时钟的采样值
        begin
            idle <= 1'b1;
            rxd_data[7] <= rxd;
            presult <= presult^rxd;
            cnt <= cnt + 8'd1;
            rxd_en <= 1'b1;         //接收数据有效
        end
```

```
        8'd152:                 //159~144 个时钟为接收奇偶校验位，取中间第 152 个时钟的采样值
            begin
                idle <= 1'b1;
                    if(presult == rxd)
                        dataerror <= 1'b0;
                    else
                        dataerror <= 1'b1;          //如果奇偶校验位不对，表示数据出错
                cnt <= cnt + 8'd1;
                rxd_en <= 1'b1;
            end
        8'd168:                 //160~175 个时钟为接收停止位，取中间第 168 个时钟的采样值
            begin
            idle <= 1'b1;
            if(1'b1 == rxd)
                frameerror <= 1'b0;
            else
                frameerror <= 1'b1;         //如果没有接收到停止位，表示帧出错
            cnt <= cnt + 8'd1;
            rxd_en <= 1'b1;
            end
        default:
            begin
                cnt <= cnt + 8'd1;
            end
        endcase
        end
      else
        begin
            cnt <= 8'd0;
            idle <= 1'b0;
            rxd_en <= 1'b0;
        end
     end
    endmodule
```

b) uartrx.vt

```
//凌智城 201806061211 通信工程 1803
`timescale 1 ps/ 1 ps
module uartrx_vlg_tst();
// constants

// test vector input registers
reg clk;
reg rst;
reg rxd;
```

```
// wires
wire dataerror;
wire frameerror;
wire [7:0]    rxd_data;
wire rxd_en;

// assign statements (if any)
uartrx i1 (
// port map - connection between master ports and signals/registers
        .clk(clk),
        .dataerror(dataerror),
        .frameerror(frameerror),
        .rst(rst),
        .rxd(rxd),
        .rxd_data(rxd_data),
        .rxd_en(rxd_en)
);
initial clk=0;
always
        begin
                #5 clk=1'b1;
                #5 clk=1'b0;
        end
initial
        begin
                rst=0;
                rxd=1;
                #200 rst=1;
                        rxd=1'b1;

                #160 rxd=1'b0;   //起始位，从高到低电平

                #160 rxd=1'b1; //低位
                #160 rxd=1'b0;
                #160 rxd=1'b0;
                #160 rxd=1'b1;
                #160 rxd=1'b0;
                #160 rxd=1'b1;
                #160 rxd=1'b1;
                #160 rxd=1'b0;   //高位

                #160 rxd=1'b1;   //奇偶校验位，学号 11，奇校验

                #160 rxd=1'b1;   //停止位
```

```
            #8000 $finish;
        end

endmodule
```

3) 发送模块

a) uarttx.v

```
//凌智城 201806061211 通信工程 1803
`timescale 1ns / 1ps
module uarttx(clk, rst,txd_data, txd_en, idle, txd);
input clk;                      //UART 时钟
input [7:0] txd_data;          //需要发送的数据
input txd_en;                   //发送命令，上升沿有效
input rst;                      //系统复位

output idle;                    //线路状态指示，高 1 为线路忙，低 0 为线路空闲
output txd;                     //发送数据信号

reg idle, txd;
reg send;
reg txd_en_buf, txd_en_rise;
reg presult;                //奇偶校验位

reg[7:0] cnt;                   //计数器
parameter paritymode = 1'b1;//奇偶校验模式'高电平为奇校验，对每一位数据位进行异或，8 位
数据若有奇数个 1，则 presult 为 1

//检测发送命令是否有效，判断 txd_en 的上升沿
always @(posedge clk)
begin
    txd_en_buf <= txd_en;
    txd_en_rise <= (~txd_en_buf) & txd_en;
end

//当发送命令有效且线路为空闲时，启动新的数据发送进程
always @(posedge clk)
begin
  if (txd_en_rise &&   (~idle))
      begin
          send <= 1'b1;             //send=1 允许发送新的数据
      end
  else if(cnt == 8'd168)        //计数器的值为 168 时一帧资料发送结束
      begin
```

```
        send <= 1'b0;                    //新的数据发送完毕
        end
end


//使用 168 个时钟发送一个数据（起始位、8 位数据、奇偶校验位、停止位），每位占用 16 个时钟
//11*16-8=168
always @(posedge clk or negedge rst)       //异步复位
begin
   if (!rst)                               //如果有复位信号输入
      begin                                    //数据清零、计数器清零、预置结果清零、线路转为空闲状态
            txd <= 1'b0;
            cnt <= 8'd0;
            presult <= 1'b0;
            idle <= 1'b0;
      end
   else if(send == 1'b1)                   //如果允许发送
      begin
      case(cnt)                            //txd 变低电平产生起始位，0~15 个时钟为发送起始位
            8'd0:
                  begin
                        txd <= 1'b0;                  //串行发送
                        idle <= 1'b1;                 //占用忙状态
                        cnt <= cnt + 8'd1;        //计数器+1
                  end
            8'd16:
                  begin
                        txd <= txd_data[0];                        //发送数据位的低位 bit0,占用第 16~31 个时钟
                        presult <= txd_data[0]^paritymode;      //从 0 开始对每一个数据位作异或操作
                        idle <= 1'b1;                              //占用忙状态
                        cnt <= cnt + 8'd1;                        //计数器+1
                  end
            8'd32:
                  begin
                        txd <= txd_data[1];       //发送数据位的第 2 位 bit1,占用第 47~32 个时钟
                        presult <= txd_data[1]^presult;
                        idle <= 1'b1;
                        cnt <= cnt + 8'd1;
                  end
            8'd48:
```

```
        begin
            txd <= txd_data[2];      //发送数据位的第 3 位 bit2,占用第 63~48 个时钟
            presult <= txd_data[2]^presult;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
    8'd64:
        begin
            txd <= txd_data[3];      //发送数据位的第 4 位 bit3,占用第 79~64 个时钟
            presult <= txd_data[3]^presult;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
    8'd80:
        begin
            txd <= txd_data[4];     //发送数据位的第 5 位 bit4,占用第 95~80 个时钟
            presult <= txd_data[4]^presult;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
    8'd96:
        begin
            txd <= txd_data[5];      //发送数据位的第 6 位 bit5,占用第 111~96 个时钟
            presult <= txd_data[5]^presult;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
    8'd112:
        begin
            txd <= txd_data[6];      //发送数据位的第 7 位 bit6,占用第 127~112 个时钟
            presult <= txd_data[6]^presult;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
    8'd128:
        begin
            txd <= txd_data[7];      //发送数据位的第 8 位 bit7,占用第 143~128 个时钟
            presult <= txd_data[7]^presult;
            idle <= 1'b1;
            cnt <= cnt + 8'd1;
        end
    8'd144:
        begin
            txd <= presult;          //发送奇偶校验位，占用第 159~144 个时钟
```

```
                presult <= txd_data[0]^paritymode;
                idle <= 1'b1;
                cnt <= cnt + 8'd1;
            end
        8'd160:
            begin
                txd <= 1'b1;            //发送停止位，占用第 160~167 个时钟
                idle <= 1'b1;
                cnt <= cnt + 8'd1;
            end
        8'd168:
            begin
                txd <= 1'b1;
                idle <= 1'b0;           //一帧资料发送结束
                cnt <= cnt + 8'd1;
            end
        default: begin cnt <= cnt + 8'd1; end   //其他计数状态，未到数据位的中间，则计数器
+1
    endcase
    end
  else
    begin
        txd <= 1'b1;
        cnt <= 8'd0;
        idle <= 1'b0;
    end
end
endmodule
```

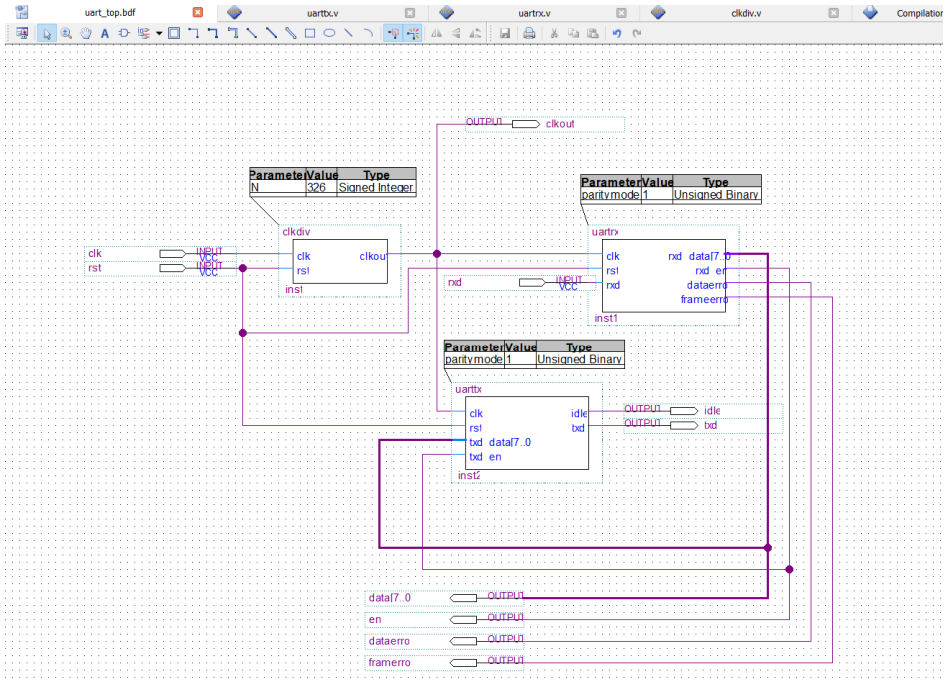
b) uarttx.vt

```
//凌智城 201806061211 通信工程 1803
`timescale 1 ps/ 1 ps
module uarttx_vlg_tst();
// constants
// test vector input registers
reg clk;
reg rst;
reg [7:0] txd_data;
reg txd_en;
// wires
wire idle;
wire txd;

// assign statements (if any)
uarttx i1 (
```

```
// port map - connection between master ports and signals/registers
        .clk(clk),
        .idle(idle),
        .rst(rst),
        .txd(txd),
        .txd_data(txd_data),
        .txd_en(txd_en)
);
initial clk=0;
always
        begin
                #5 clk=1'b1;
                #5 clk=1'b0;
        end
initial
        begin
                rst=0;
                txd_en=0;
                txd_data=8'b10100101;
                #200 rst=1;
                #200 txd_en =1;
                #8000 $finish;
        end

endmodule
```

4) 顶层合并

a) top_level.bdf



b) top_level.v

```
//凌智城  201806061211  通信工程 1803
module uart_top(
        clk,
        rst,
        rxd,
        clkout,
        en,
        dataerror,
        framerror,
        idle,
        txd,
        data
);


input wire  clk;
input wire  rst;
input wire  rxd;
output wire        clkout;
output wire        en;
output wire        dataerror;
output wire        framerror;
output wire        idle;
output wire        txd;
output wire        [7:0] data;

wire  SYNTHESIZED_WIRE_4;
wire  SYNTHESIZED_WIRE_2;
wire  [7:0] SYNTHESIZED_WIRE_3;

assign       clkout = SYNTHESIZED_WIRE_4;
assign       en = SYNTHESIZED_WIRE_2;
assign       data = SYNTHESIZED_WIRE_3;




clkdivb2v_inst(
        .clk(clk),
        .rst(rst),
        .clkout(SYNTHESIZED_WIRE_4));
        defparam   b2v_inst.N = 326;
```

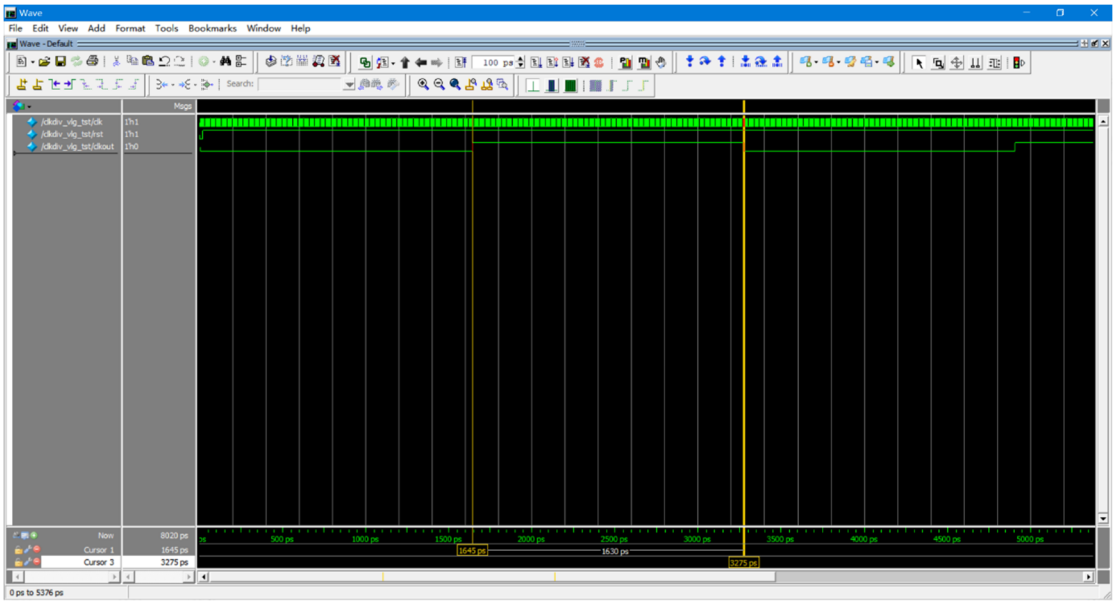
```
uartrx        b2v_inst1(
        .clk(SYNTHESIZED_WIRE_4),
        .rst(rst),
        .rxd(rxd),
        .rxd_en(SYNTHESIZED_WIRE_2),
        .dataerror(dataerror),
        .frameerror(framerror),
        .rxd_data(SYNTHESIZED_WIRE_3));
        defparam   b2v_inst1.paritymode = 1'b1;


uarttx        b2v_inst2(
        .clk(SYNTHESIZED_WIRE_4),
        .rst(rst),
        .txd_en(SYNTHESIZED_WIRE_2),
        .txd_data(SYNTHESIZED_WIRE_3),
        .idle(idle),
        .txd(txd));
        defparam   b2v_inst2.paritymode = 1'b1;

endmodule
```

c) top_level.vt

```
//凌智城  201806061211  通信工程 1803
`timescale 1 ps/ 1 ps
module uart_top_vlg_tst();
// constants
// test vector input registers
reg clk;
reg rst;
reg rxd;
// wires
wire clkout;
wire [7:0]    data;
wire dataerror;
wire en;
wire framerror;
wire idle;
wire txd;

// assign statements (if any)
uart_top i1 (
// port map - connection between master ports and signals/registers
```

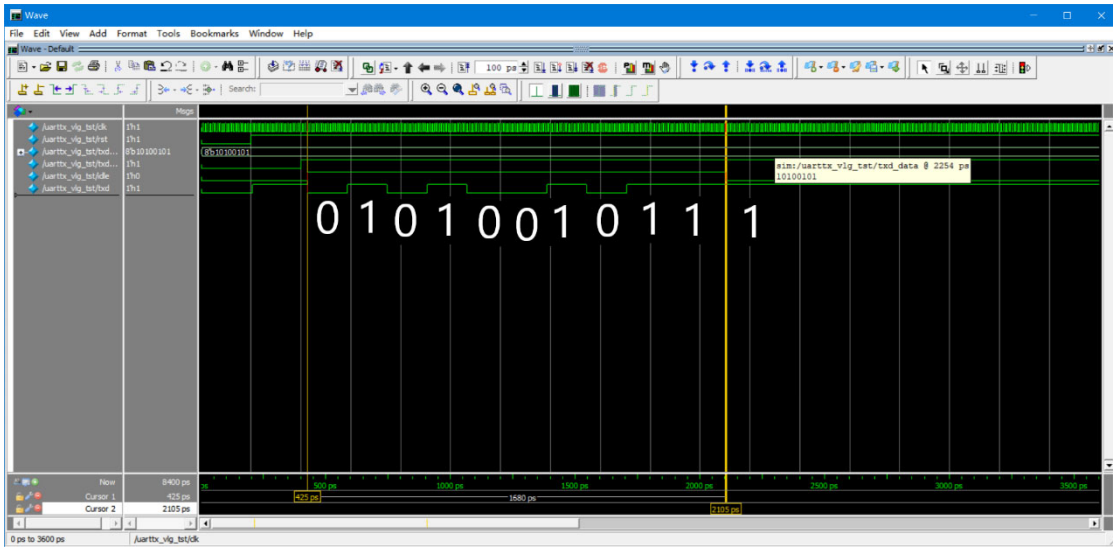
```
        .clk(clk),
        .clkout(clkout),
        .data(data),
        .dataerror(dataerror),
        .en(en),
        .framerror(framerror),
        .idle(idle),
        .rst(rst),
        .rxd(rxd),
        .txd(txd)
);
initial clk=0;
always
        begin
                #5 clk=1'b1;
                #5 clk=1'b0;
        end
initial
        begin
                rst=0;
                rxd=1;
                #200 rst=1;
                        rxd=1'b1;

                #52160 rxd=1'b0;   //起始位，从高到低电平

                #52160 rxd=1'b1; //低位
                #52160 rxd=1'b0;
                #52160 rxd=1'b0;
                #52160 rxd=1'b1;
                #52160 rxd=1'b0;
                #52160 rxd=1'b1;
                #52160 rxd=1'b1;
                #52160 rxd=1'b0;       //高位

                #52160 rxd=1'b1;       //奇偶校验位，学号 11，奇校验

                #52160 rxd=1'b1;   //停止位

                #2608000 $finish;
        end

endmodule
```

3.4 实验结果与分析



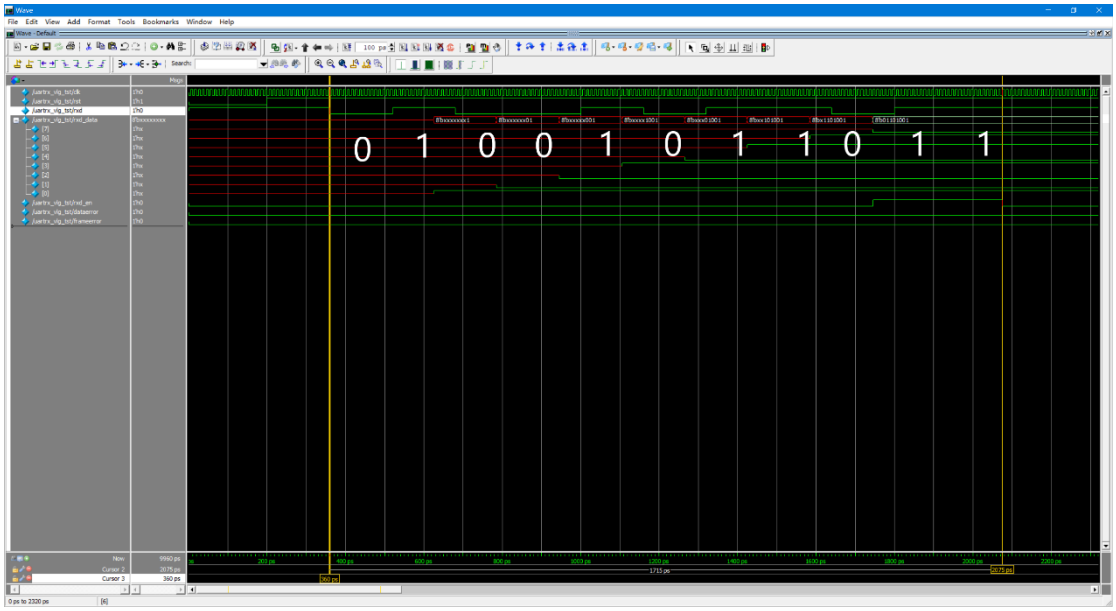
原始 clk 周期为 10ps，设置分频系数为 326，则 clkout 周期为 3260ps，图中截取了半个周期即 1630ps。

图 3-1 9600 波特率 326 分频器的仿真截图



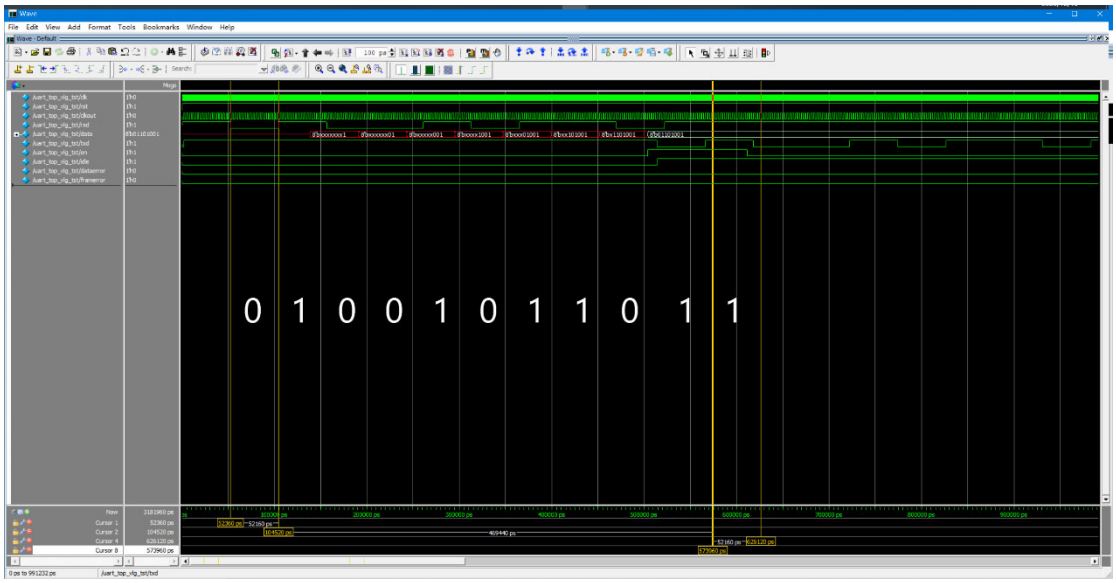
测试的发送并行数据 txd 为 8'b10100101，加上起始位、奇校验位、停止位，数据从低位到高位途中 从左到右串行为 0 10100101 1 1 ，425ps 位置为起始位初始，2105ps 位置为中止位初始。

图 3-2 发送模块（测试数据 10100101）的仿真截图



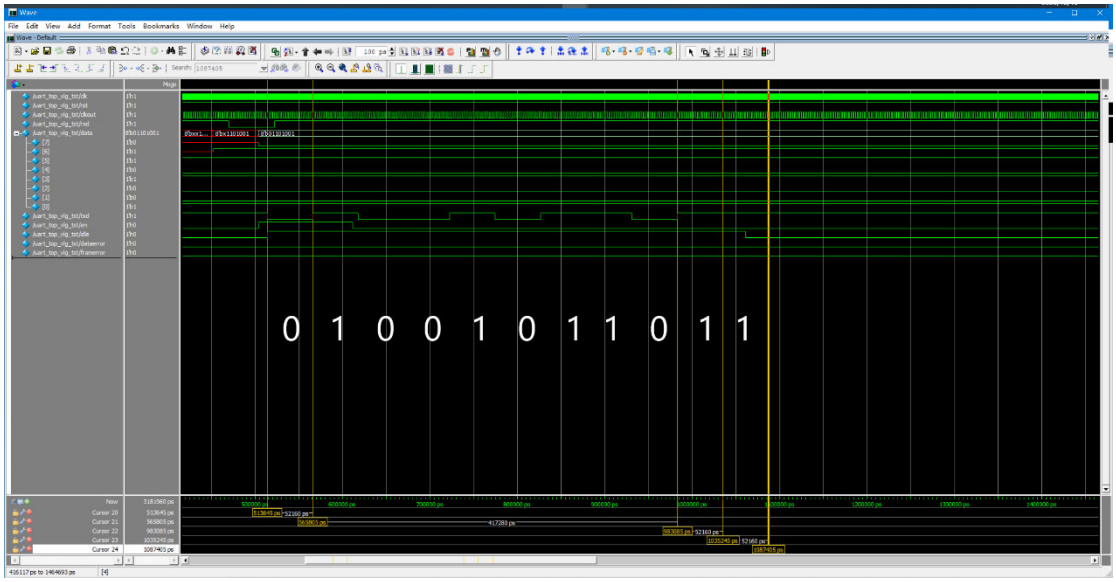
测试的接收数据 rxd 为 01101001，黄色区间依次接收的顺序为起始位低电平，1，0，0，1，0，1，1，0，奇校验位 1，终止位 1。接收完八位数据后 rxd_en 转为 1 表示数据接收完毕，dataerror 为 0 说明奇校验正确，framerror 为 0 说明接收到了终止位，一帧接收完毕。数据转存入 rxd 寄存器 8'b01101001，这里接收数据是从第一位开始发送，所以 8'b 高->低，这个顺序会反过来要注意。

图 3-3 接收模块（测试数据 01101001）的仿真截图



按照顶层模块设计，先由 uartrx 模块接收到串行数据 rxd，存入 data 寄存器，然后再由 uarttx 模块发送出串行数据 txd。顶层仿真图一中显示的是接收数据转存的部分，测试的串行数据 rxd 依次为 10010110，可以看到 rxd 按时序依次为起始位 0，10010110，奇校验位 1，终止位 1；由于采用了 9600 波特率的 326 分频器，clk 的周期为 10ps，此时 clkout 的周期为 3260ps，每一位数据/起始/校验/终止都是 3260ps*16=52160ps，如图可见。dataerror 为 0 说明奇校验正确，framerror 为 0 说明接收到了终止位，一帧接收完毕。en 在接收校验位和终止位时为 1，说明已经正确接收到八位有效数据。

图 3-4 顶层模块的仿真截图一



顶层仿真图二中显示的是从寄存器 data 中取出并行的八位数据以串行的方式发送出去 txd，并且携带起始位、奇校验位、终止位共十一位，每位占用时间位 326*10ps*16=52160ps，中间八位数据占用 52160ps*8=417280ps，如图可见。测试数据为 8'b01101001，故 txd 按照时序依次为起始位 0，1，0，0，1，0，1，1，0，奇校验 1，终止位 1。发送期间 idle 为高电平，表明发送通道忙。

图 3-5 顶层模块的仿真截图二

实验总结

1、 通过仿真和硬件实验的比较，你对哪些概念从不理解到理解了？

2、 相比其它方法（比如数字逻辑设计、单片机等），你认为 FPGA 对数字电路设计有哪些优势？

3、 你对设计一个比较复杂的工程项目有何一般性的方法？

实验改进建议

建议一：

建议二：