Design and Implementation of Universal Asynchronous Transceiver (UART)

通用异步收发器 UART 的设计与实现

姓名：刘北北

学号：2011019060027

班级：英才 2011 级 1 班

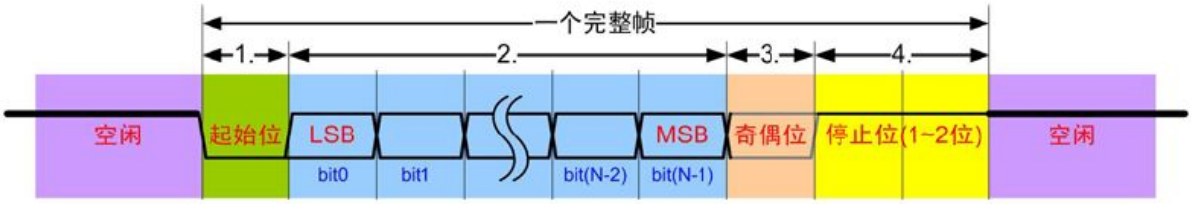
指导老师：林水生老师　周亮老师

【实验原理】：

一、UART 简介：

UART 的全称是通用异步收发器（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 用来主机与辅助设备通信，如汽车音响与外接 AP 之间的通信，与 PC 机通信包括与监控调试器和其它器件，如 EEPROM 通信。

二、帧格式：



起始位：先发出一个逻辑”0”的信号，表示传输字符的开始。

资料位：紧接着起始位之后。资料位的个数可以是 4、5、6、7、8 等，构成一个字符。通常采用 ASCII 码。从最低位开始传送，靠时钟定位。

奇偶校验位：资料位加上这一位后，使得“1”的位数应为偶数(偶校验)或奇数(奇校验)，以此来校验资料传送的正确性。

停止位：它是一个字符数据的结束标志。可以是 1 位、1.5 位、2 位的高电平。 由于数据是在传输线上定时的，并且每一个设备有其自己的时钟，很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束，并且提供计算机校正时钟同步的机会。适用于停止位的位数越多，不同时钟同步的容忍程度越大，但是数据传输率同时也越慢。[3]

空闲位：处于逻辑“1”状态，表示当前线路上没有资料传送。

波特率：是衡量资料传送速率的指标。表示每秒钟传送的二进制位数。例如资料传送速率为 120 字符/秒，而每一个字符为 10 位，则其传送的波特率为 10×120=1200 位/秒=1200 波特。

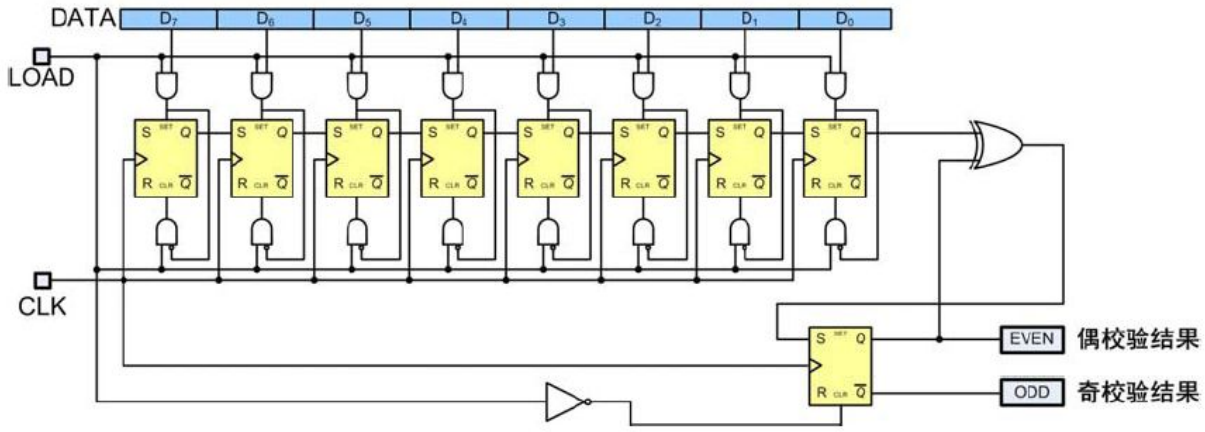
三、奇偶校验：

奇偶校验用于检验数据传输过程中是否出错，但其能力有限，可配合更高层次的校验保证数据的可靠性。

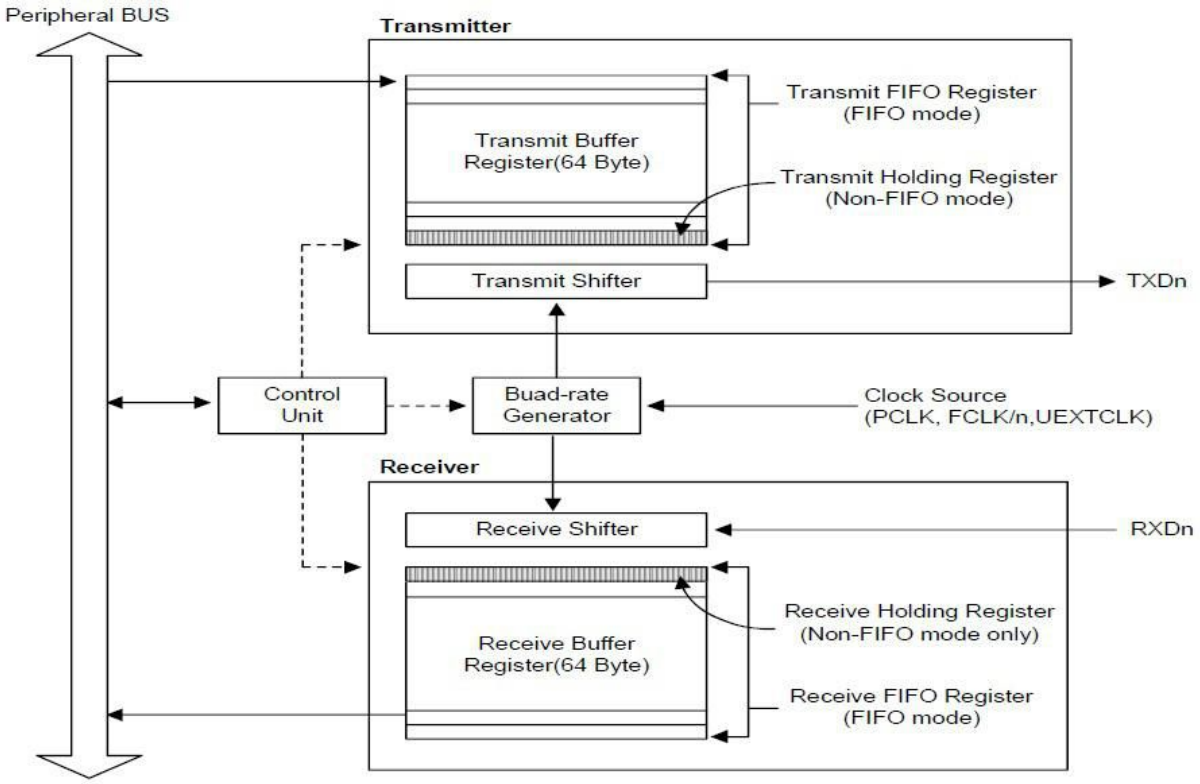
对数据进行逐位同或 / 异或运算

DEVEN ＝ D7⊕D6⊕D5⊕D4⊕D3⊕D2⊕D1⊕D0

DODD ＝ ~DEVEN



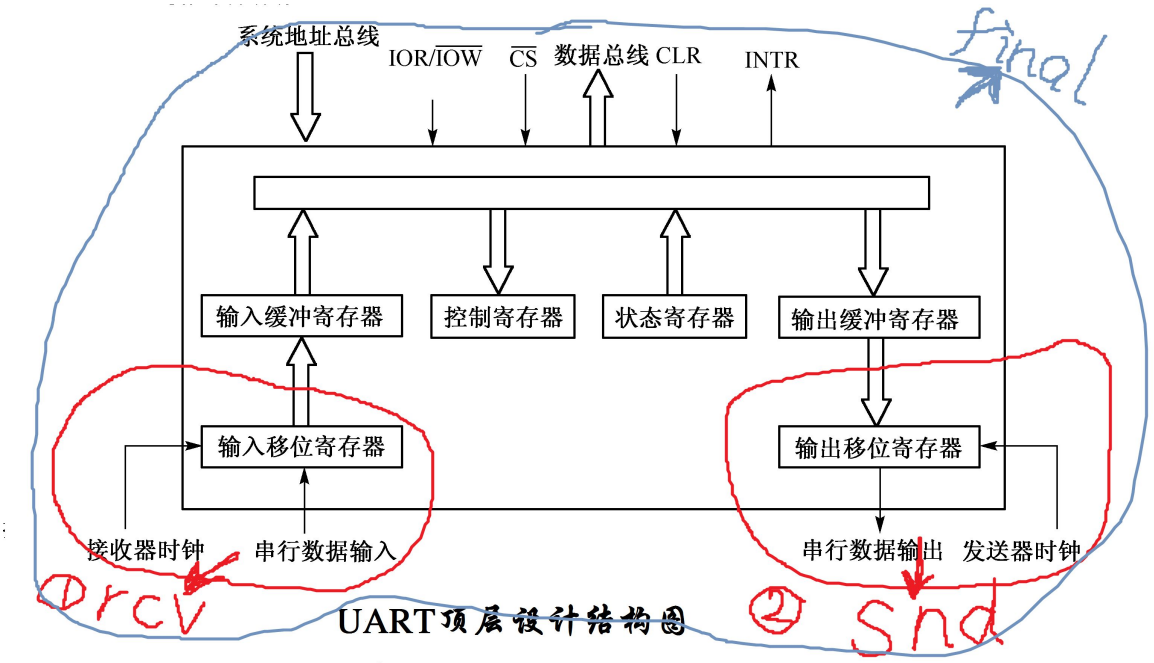
四、总体结构：



【设计思路】：

采用自顶向下设计方法：

一、顶层设计：



二、接收模块——rcv：

示意图见彩页 1

接收机的设计思路，是以状态机为框架（虚线框内），具体功能模块为附属零件（外部蓝色圈）。

状态机采用三段式。（在网上查到有一段式、两段式和三段式，三段式功能要分立一些，更清楚，移植性也更好）

process1:同步时序进程描述状态寄存器：

根据时钟的节拍，每来一个上升沿，就把“当前状态”更新为刚才的“下一状态”。

```
63
64  ----------------1、同步时序进程描述状态寄存器-------------------------
65  process(rst,clk_baud)
66  begin
67      if rst='1' then
68           current_state<=RX_IDLE; --复位时，停在空闲状态
69      elsif clk_baud'event and clk_baud='1' then
70          current_state<=next_state; --当前状态变为刚才的"下一状态"
71      end if;
72  end process;
73
```

process2：组合逻辑进程描述次态逻辑：

描述各个状态之间的转换关系，如图，黑色圈表示状态，黑色箭头表示条件转移，每个时钟上升沿来的时候默认动作是保持当前状态不变。

这个进程只负责根据状态的指示，或是周围模块的执行结果来进行状态转移，并不负责具体功能的执行。

分为 7 个状态：

1、RX_IDLE：空闲状态

2、RX_SYNC：等待接收起始位状态

3、RX_DATA：接收数据状态

4、RX_PARITY：计算校验和状态

5、RX_STOP：等待接收停止位状态

6、RX_ENDING：接收结束状态

7、RX_DONE：一帧数据接收完成状态

```
73
74  ----------------2、组合逻辑进程描述次态逻辑----------------------------
75
76  process(current_state,rst,RXD,RcvStart,FinishRcv,NeedCheck,RcvStop,CheckResult,FinishProcess)
77  begin
78      next_state<=current_state;    --初始化：先让next_state和current_state一样
79      if rst='1' then
80           next_state<=RX_IDLE; --复位时，停在空闲状态
81      else
82          case current_state is
83              when RX_IDLE => if RXD='0' then              --RX_IDLE：空闲状态
84                                  next_state<=RX_SYNC;
85                              else
86                                  next_state<=current_state;
87                              end if;
88              when RX_SYNC => if RcvStart='0' then          --RX_SYNC：等待接收起始位状态
89                                  next_state<=RX_IDLE;
90                              elsif RcvStart='1' then
91                                  next_state<=RX_DATA;
92                              else
93                                  next_state<=current_state;
94                              end if;
95              when RX_DATA => if FinishRcv='1' then            --RX_DATA：接收数据状态
96                                    if NeedCheck='1' then
97                                       next_state<=RX_PARITY;
98                                    elsif NeedCheck='0' then
99                                         next_state<=RX_STOP;
100                                   end if;
101                             else
102                                 next_state<=current_state;
103                             end if;
```

```
104             when RX_PARITY=>if CheckResult='1' then         --RX_PARITY：计算校验和状态
105                                 next_state<=RX_STOP;
106                             else
107                                 next_state<=current_state;
108                             end if;
109             when RX_STOP => if RcvStop='1' then             --RX_STOP：等待接收停止位状态
110                                 next_state<=RX_ENDING;
111                             else
112                                 next_state<=current_state;
113                             end if;
114             when RX_ENDING => if FinishProcess='1' then      --RX_ENDING：接收结束状态
115                                 next_state<=RX_DONE;
116                             else
117                                 next_state<=current_state;
118                             end if;
119             when RX_DONE => next_state<=RX_IDLE;       --RX_DONE：一帧数据接收完成状态
120             when others =>  next_state<=RX_IDLE;       --默认状态为空闲RX_IDLE
121         end case;
122     end if;
123 end process;
124
```

process3：输出逻辑进程：

对应每一个状态，它负责产生状态指示，指挥周围的具体功能模块的启动。

```
124
125   ------------------3、输出逻辑进程---------------------------------
126   process(current_state)
127   begin
128       case current_state is
129                   when RX_IDLE => IsIdle<='1';
130                   when RX_SYNC => ListentoStart<='1';
131                   when RX_DATA => DataRcv<='1';
132                   when RX_PARITY=> DataCheck<='1';
133                   when RX_STOP => ListentoStop<='1';
134                   when RX_ENDING => StartProcess<='1';
135                   when RX_DONE => success <='1';
136                   when others => IsIdle<='1';
137           end case;
138    end process;
139
140   end Behavioral;
141
```

外围的具体功能模块：

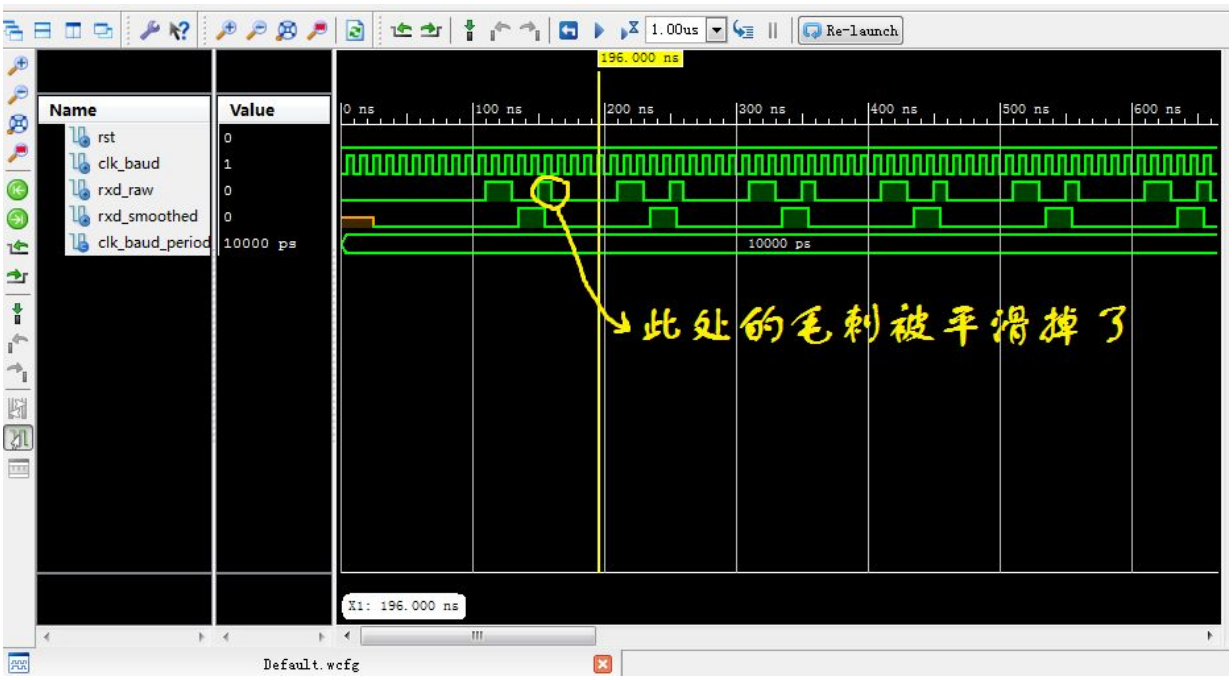
1、RXD_smooth 模块：

采用多数判决平滑 RXD 输入。用移位寄存器保存当前（RXD_raw）和前面的一共三个 RXD 输入（a,b,c),当两个有效,或是三个同时有效时，输出（RXD_smoothed）才判为有效（RXD 为高电平）。

这样可以平滑输入的毛刺，确保输入的稳定。

```
--------RXD_smooth模块：采用多数判决平滑RXD输入-----------------

entity RXD_smooth is
    Port ( rst:in  STD_LOGIC;
           clk_baud:in  STD_LOGIC;
           RXD_raw : in  STD_LOGIC;
           RXD_smoothed : out  STD_LOGIC);
end RXD_smooth;

architecture Behavioral of RXD_smooth is
signal a,b,c:STD_LOGIC;
begin

process(RXD_raw,clk_baud,a,b,c,rst)
begin
   if rst='1' then
        a<='0';
        b<='0';
        c<='0';
        RXD_smoothed<='0';
   elsif clk_baud'event and clk_baud='1' then
       c<=b;
       b<=a;
       a<=RXD_raw;
       RXD_smoothed<=(a and b)or (a and c) or (b and c);
    end if;
end process;
end Behavioral;
```

模块仿真结果：



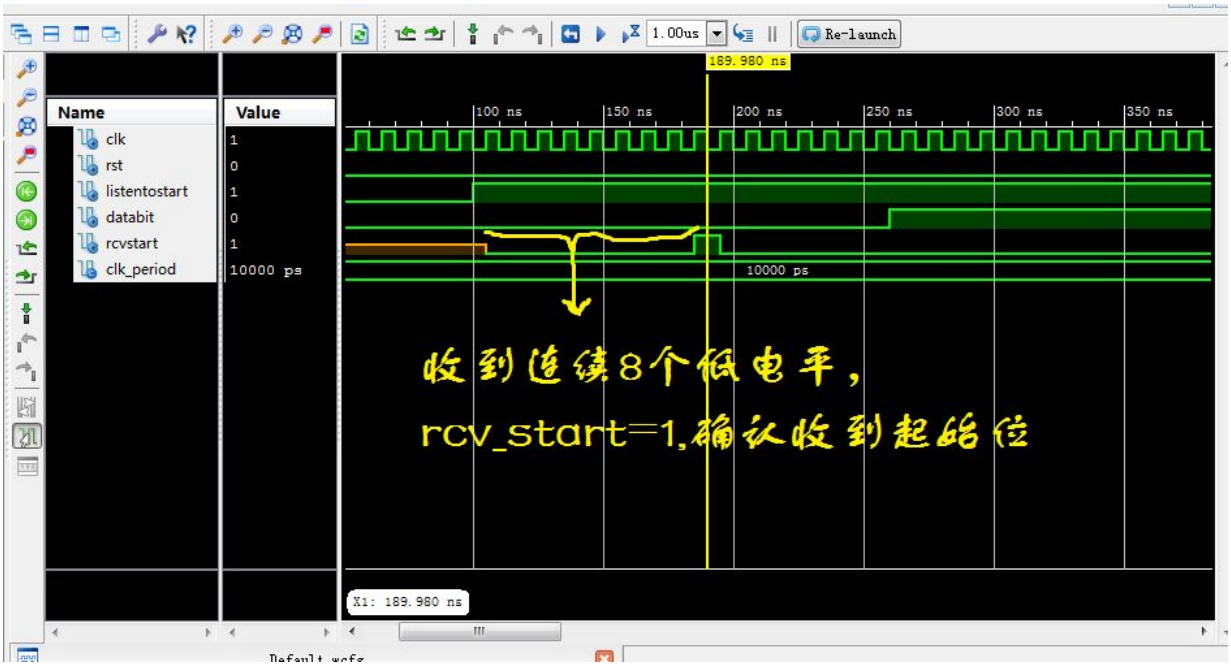
2、rcv_start 模块：

接收起始位。按时钟（clk）的节拍，在每个上升沿进行数据线（databit)的电平采

样。如果检测到连续的 8 个低电平，则认为收到了起始位（rcy_start=1）。因为波特率时钟是 clk 的 16 分频，所以收到起始位后每隔 16 个时钟脉冲 T 对数据线采样 1 次，以确保可以在稳定状态接收到该 bit 数据。

```
------------ rcv_start模块：接收起始位---------------------

entity rcv_start is
    Port ( clk: in  STD_LOGIC;
           rst : in  STD_LOGIC;
           ListentoStart: in  STD_LOGIC;
           databit: in  STD_LOGIC;
           RcvStart : out  STD_LOGIC);
end rcv_start;

architecture Behavioral of rcv_start is

begin
process(clk,rst,databit,ListentoStart)

variable count:integer range 0 to 8;

begin
    if rst='1' then
        count:=0;
        RcvStart<='0';
   elsif clk'event and clk='1' and ListentoStart='1' then
        if count=8 then
            RcvStart<='1';
            count:=0;
        elsif databit='0' then
            count:=count+1;
        elsif databit='1' then
            count:=0;
        end if;
   end if;
end process;

end Behavioral;
```

模块仿真结果：

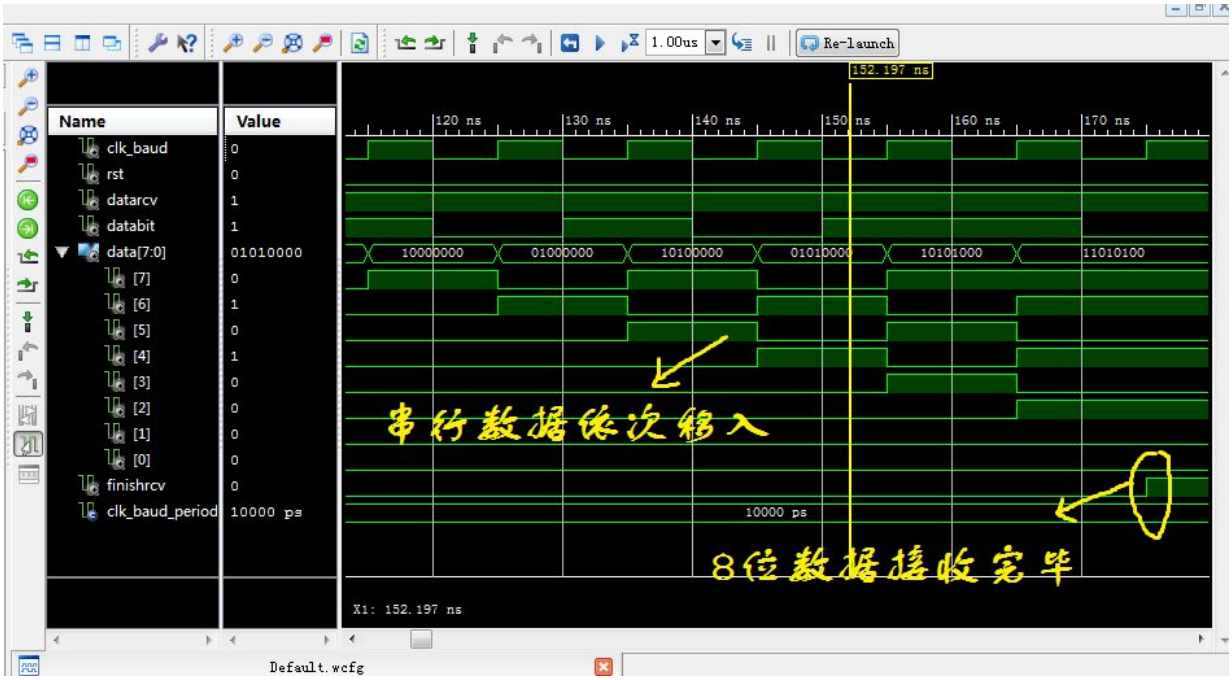


3、shift_reg 模块：

移位寄存器。将接收到的串行数据（databit)一位一位移入移位寄存器里，然后 8 位并行输出（data)。计数，每移进一位加一，当 8 位全部移入后返回数据接收完毕的状态指示（finish_recieve)。

```
-------shift_reg模块：移位寄存器-----------------------
entity shift_reg is
    Port ( clk_baud :  in  STD_LOGIC;
           rst :  in  STD_LOGIC;
           DataRcv : in  STD_LOGIC;
           databit : in  STD_LOGIC;
           data: out  STD_LOGIC_vector(7 downto 0);
           FinishRcv : out  STD_LOGIC);
end shift_reg;

architecture Behavioral of shift_reg is

signal data_temp:STD_LOGIC_vector(7 downto 0):="00000000";
begin
process(rst,DataRcv,databit,clk_baud,data_temp)
variable count: integer range 0 to 8;
begin
        if rst='1' then
                count:=0;
                data_temp<="00000000";
                FinishRcv<='0';
        elsif clk_baud'event and clk_baud='1' and DataRcv='1' then
                if ( count=7) then
                    FinishRcv<='1';
                    count:=0;
                else
                    FinishRcv<='0';
                    count:=count+1;
                    data_temp(6 downto 0)<=data_temp(7 downto 1);
                    data_temp(7)<=databit;
                end if;
        end if;
end process;
data<=data_temp;
end Behavioral;
```

模块仿真结果：



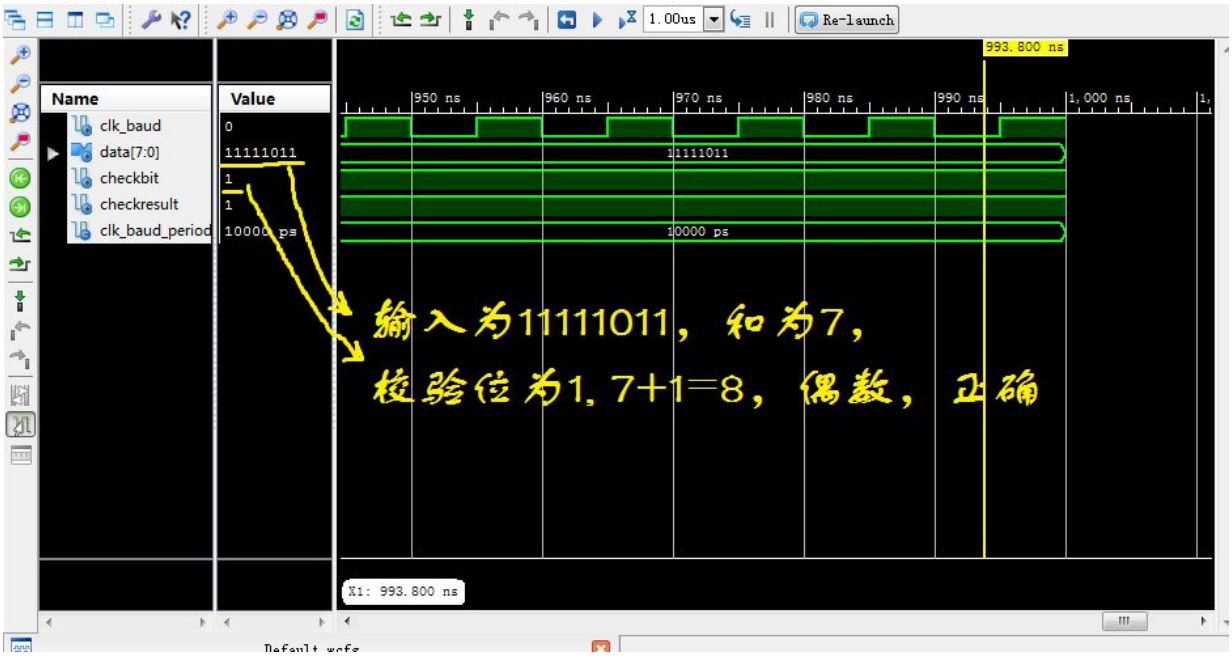
4、check 模块：

检测校验和是否正确。将 8 位数据位求和，然后模 2（程序中是各个比特做异或运算），结果与校验位比较，如果一致，则认为校验结果为正确，否则为错误。

本串口协议采用偶校验，即 8 位数据加上校验位的和为偶数，即模 2 后为 0 时正确。

```
32  ----------check模块：检测校验和是否正确--------------------
33
34  entity check is
35      Port (
36             clk_baud: in  STD_LOGIC;
37             data : in  STD_LOGIC_VECTOR (7 downto 0);
38             CheckBit : in  STD_LOGIC;
39             CheckResult: out STD_LOGIC);
40  end check;
41
42  architecture Behavioral of check is
43
44  signal sum: STD_LOGIC;
45
46  begin
47  process(clk_baud,data,CheckBit,sum)
48  begin
49      if clk_baud'event and clk_baud='1'  then
50        sum<=data(7) xor data(6)xor data(5)xor data(4)xor data(3)xor data(2)xor data(1)xor data(0);
51      end if;
52      if sum=CheckBit then
53          CheckResult<='1';
54      else
55          CheckResult<='0';
56      end if;
57  end process;
58  end Behavioral;
59
```

模块仿真结果：

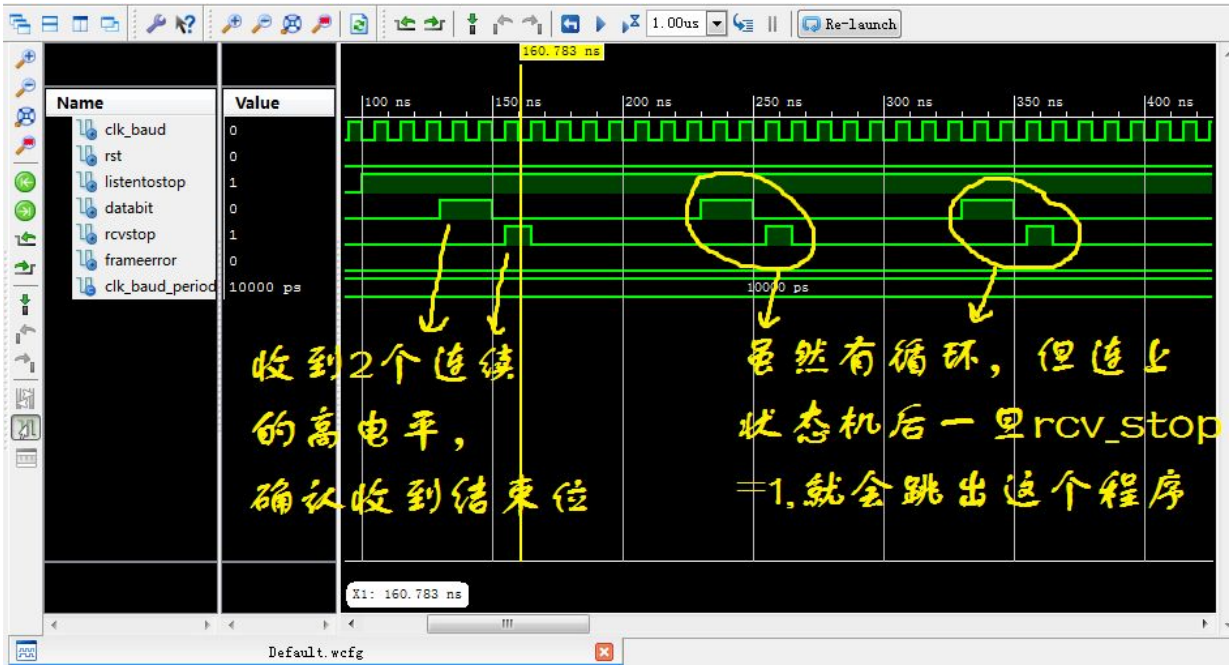


5、rcv_stop 模块：

接收停止位。

```
------------rcv_stop模块：接收停止位-----------------
entity rcv_stop is
    Port ( clk_baud : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           ListentoStop : in  STD_LOGIC;
           databit : in  STD_LOGIC;
           RcvStop : out  STD_LOGIC;
           FrameError: out  STD_LOGIC);
end rcv_stop;

architecture Behavioral of rcv_stop is

begin
process(clk_baud,rst,databit,ListentoStop)

variable count:integer range 0 to 2;
begin
    if rst='1' and ListentoStop='1' then
        count:=0;
        RcvStop<='0';
   elsif clk_baud'event and clk_baud='1' then
       if count=2 then
            RcvStop<='1';
            count:=0;
            FrameError<='0';
        elsif databit='1' then
            count:=count+1;
            FrameError<='0';
            RcvStop<='0';
        elsif databit='0' then
            FrameError<='0';
            RcvStop<='0';
        end if;
   end if;
end process;
```

模块仿真结果：

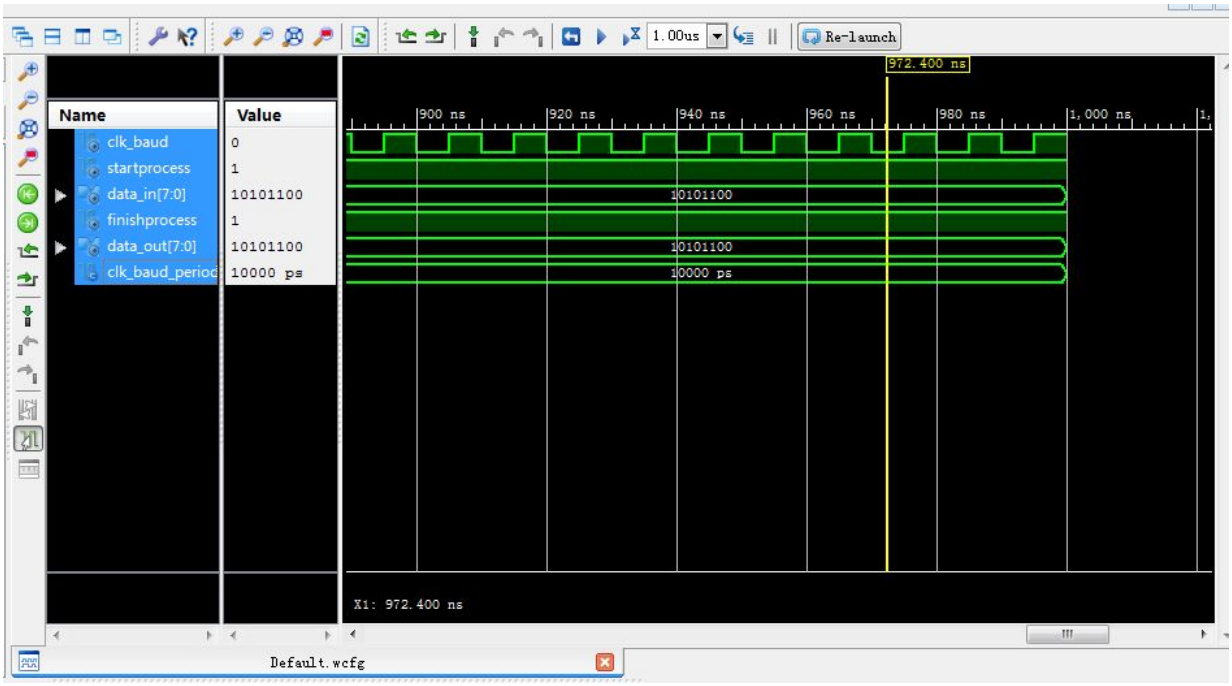


6、data_process 模块：

将移位寄存器里的 8 位并行数据存入接收缓冲寄存器。

```
-----data_process模块：将移位寄存器里的8位并行数据存入接收缓冲寄存器---------

entity data_process is
    Port ( clk_baud:in  STD_LOGIC;
           StartProcess : in  STD_LOGIC;
           FinishProcess : out  STD_LOGIC;
           data_in : in  STD_LOGIC_VECTOR (7 downto 0);
           data_out : out  STD_LOGIC_VECTOR (7 downto 0));
end data_process;

architecture Behavioral of data_process is

begin
process(StartProcess,data_in,clk_baud)
begin
    if StartProcess='1' and clk_baud'event and clk_baud='1' then
        data_out<=data_in;
        FinishProcess<='1';
    end if;
end process;

end Behavioral;

```

模块仿真结果：

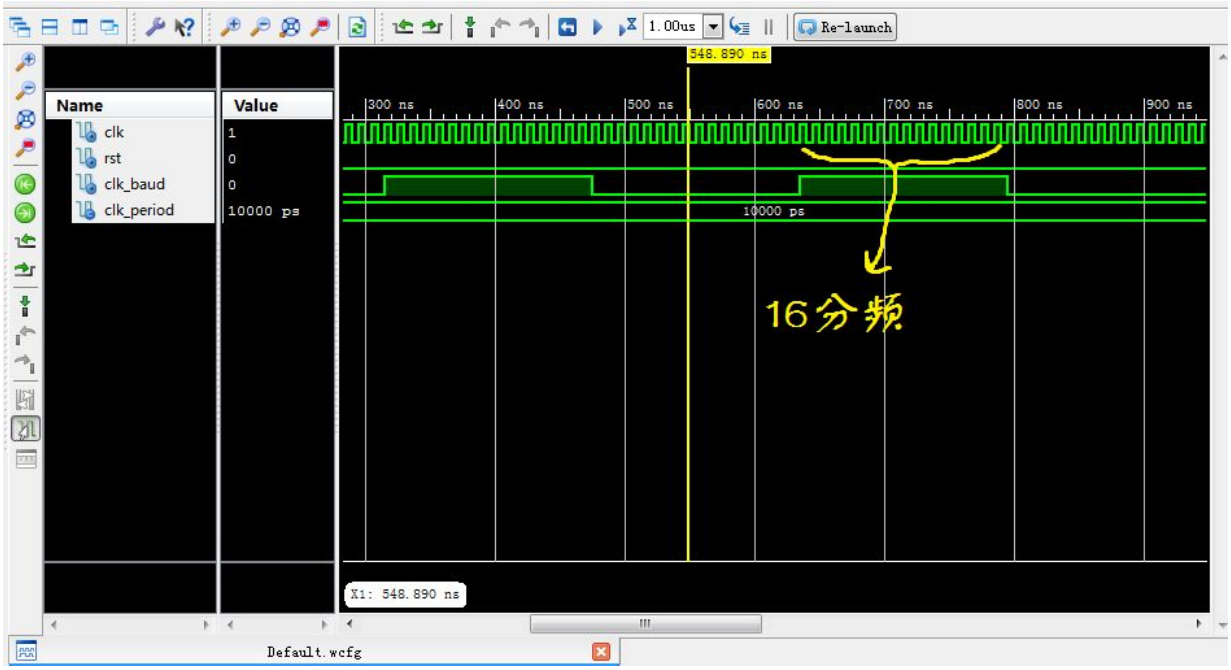


7、baud_make 模块：

波特率时钟发生。

```
36  ---------------- baud_make模块：波特率时钟发生----------------
37  entity baud_make is
38     Port (    clk : in  STD_LOGIC;
39               rst : in  STD_LOGIC;
40               clk_baud : out  STD_LOGIC
41           );
42  end baud_make;
43
44  architecture Behavioral of baud_make is
45
46  signal temp:std_logic:='1'; --初始化
47
48  begin
49  process(clk,rst,temp)
50  variable count: integer range 0 to 15;
51  begin
52      if (rst='1') then
53          count:=0;
54          temp<='0';
55      elsif (clk'event and clk='1') then
56          --flag<=conv_std_logic_vector(count,4);
57          if (count=15) then
58              --temp<='1';
59              temp <= not temp ;
60              count:=0;
61          else
62             count:=count+1;
63          end if;
64      end if;
65  end process;
66  clk_baud<=temp;
67  end Behavioral;
```

模块仿真结果：



三、发送模块——snd:

示意图见彩页 2

发送机的设计思路，和接收机类似，是以状态机为框架（虚线框内），具体功能模块为附属零件（外部蓝色圈）。

状态机仍然采用三段式。

process1:同步时序进程描述状态寄存器：

根据时钟的节拍，每来一个上升沿，就把“当前状态”更新为刚才的“下一状态”。

```
60
61  ----------------1、同步时序进程描述状态寄存器----------------------------
62  process(rst,clk_baud)
63  begin
64      if rst='1' then
65           current_state<=TX_IDLE; --复位时，停在空闲状态
66      elsif clk_baud'event and clk_baud='1' then
67          current_state<=next_state; --当前状态变为刚才的"下一状态"
68      end if;
69  end process;
70
```

process2：组合逻辑进程描述次态逻辑：

描述各个状态之间的转换关系。原理和接收部分类似，不再赘述。但由于过程不同，状态的设置与接收机不同。

分为 7 个状态：

1、TX_DATA:加载数据状态

2、TX_STARTSIGN：添加起始位状态

3、TX_CALCU：计算校验和状态

4、TX_SEND：发送数据状态

5、TX_STOPSIGN：添加结束位状态

6、TX_DELAY：延时（加空闲位）状态，数据未发完

7、TX_IDLE：空闲状态，数据发完了

```
----------------2、组合逻辑进程描述次态逻辑----------------------------

process(current_state,rst,IsIdle,FinishSend,AllFinished,DelayOver,send_en,NeedCheck)
begin
    next_state<=current_state;    --初始化：先让next_state和current_state一样
    if rst='1' then
         next_state<=TX_IDLE; --复位时，停在空闲状态
    else
        case current_state is
            when TX_DATA => if IsIdle='1' then           --TX_DATA:加载数据状态
                                next_state<=TX_STARTSIGN;
                            else
                                next_state<=current_state;
                            end if;
            when TX_STARTSIGN=>if NeedCheck='1' then     --TX_STARTSIGN: 添加起始位状态
                                next_state<=TX_CALCU;
                            else
                                next_state<=TX_SEND;
                            end if;
            when TX_CALCU => next_state<=TX_SEND;        --TX_CALCU: 计算校验和状态
            when TX_SEND => if FinishSend='1' then       --TX_SEND: 发送数据状态
                                next_state<=TX_STOPSIGN;
                            else
                                next_state<=current_state;
                            end if;
            when TX_STOPSIGN => if AllFinished='0' then --TX_STOPSIGN: 添加结束位状态
                                next_state<=TX_DELAY;
                            elsif AllFinished='1' then
                                next_state<=TX_IDLE;
                            end if;
            when TX_DELAY => if DelayOver='1' then    --TX_DELAY: 延时（加空闲位）状态
                               next_state<=TX_DATA;   --（数据未发完）
                            else
                                next_state<=current_state;
                            end if;
```

```
             when TX_IDLE =>if send_en='1' then       --TX_IDLE: 空闲状态（数据发完了）
                               next_state<=TX_DATA;
                            else
                                next_state<=current_state;
                            end if;
             when others =>  next_state<=TX_IDLE;     --默认状态: 空闲状态
         end case;
     end if;
end process;
```

process3：输出逻辑进程：

对应每一个状态，它负责产生状态指示，指挥周围的具体功能模块的启动。

```
------------------3、输出逻辑进程--------------------------------
process(current_state)
begin
    case current_state is
            when TX_DATA => StartLoad<='1';
                            TXD<='1';
            when TX_STARTSIGN=> AddStartSign<='1';
            when TX_CALCU => CalcuSum<='1';
            when TX_SEND => DataSend<='1';
            when TX_STOPSIGN => AddStopSign<='1';
            when TX_DELAY => GoDelay<='1';
            when TX_IDLE => TXD<='0';
            when others => TXD<='0';
        end case;
 end process;
end Behavioral;

```

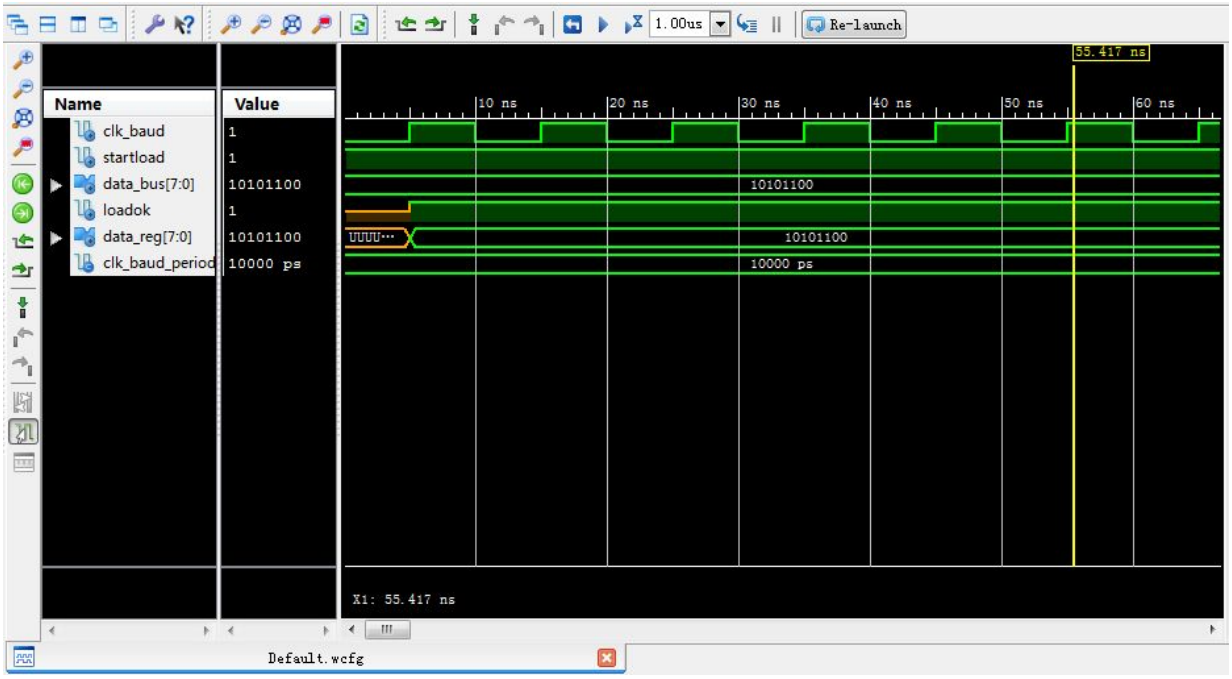
具体功能模块：

1、load 模块：

将发送数据缓冲寄存器中的数据加载到移位寄存器。

```
----------load模块：加载数据到移位寄存器---------------

entity load is
    Port ( StartLoad : in  STD_LOGIC;
           LoadOk : out  STD_LOGIC;
           data_bus : in  STD_LOGIC_VECTOR (7 downto 0);
           data_reg : out  STD_LOGIC_VECTOR (7 downto 0));
end load;

architecture Behavioral of load is

begin
process(StartLoad,data_bus)
begin
    if StartLoad='1' then
        data_reg<=data_bus;
        LoadOk<='1';
    end if;
end process;

end Behavioral;

```

模块仿真结果：

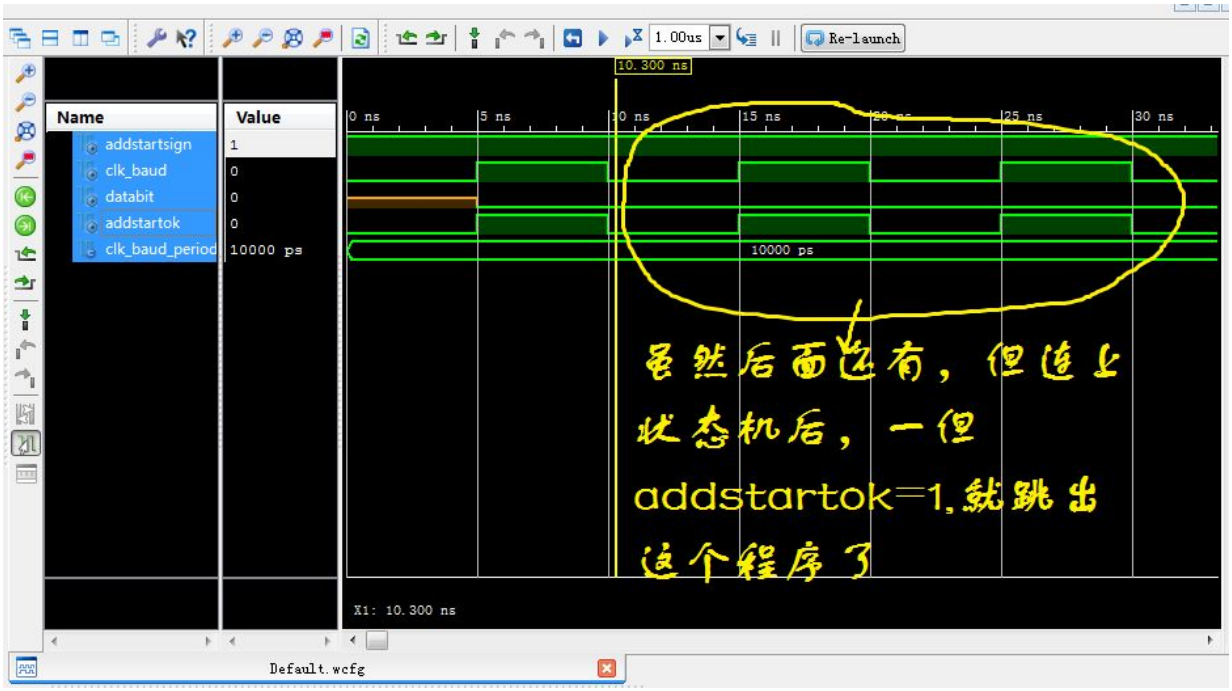


2、add_start 模块：

添加起始位。

```
-------------add_start模块：添加起始位-----------------------

entity add_start is
    Port ( AddStartSign : in  STD_LOGIC;
           databit : out  STD_LOGIC;
           clk_baud : in  STD_LOGIC;
           AddStartOk: out  STD_LOGIC);
end add_start;

architecture Behavioral of add_start is

begin
process(clk_baud,AddStartSign)
begin
if clk_baud'event and clk_baud='1' and AddStartSign='1' then
    databit<='0';
    AddStartOk<='1';
else
    AddStartOk<='0';
end if;
end process;
end Behavioral;

```

模块仿真结果：

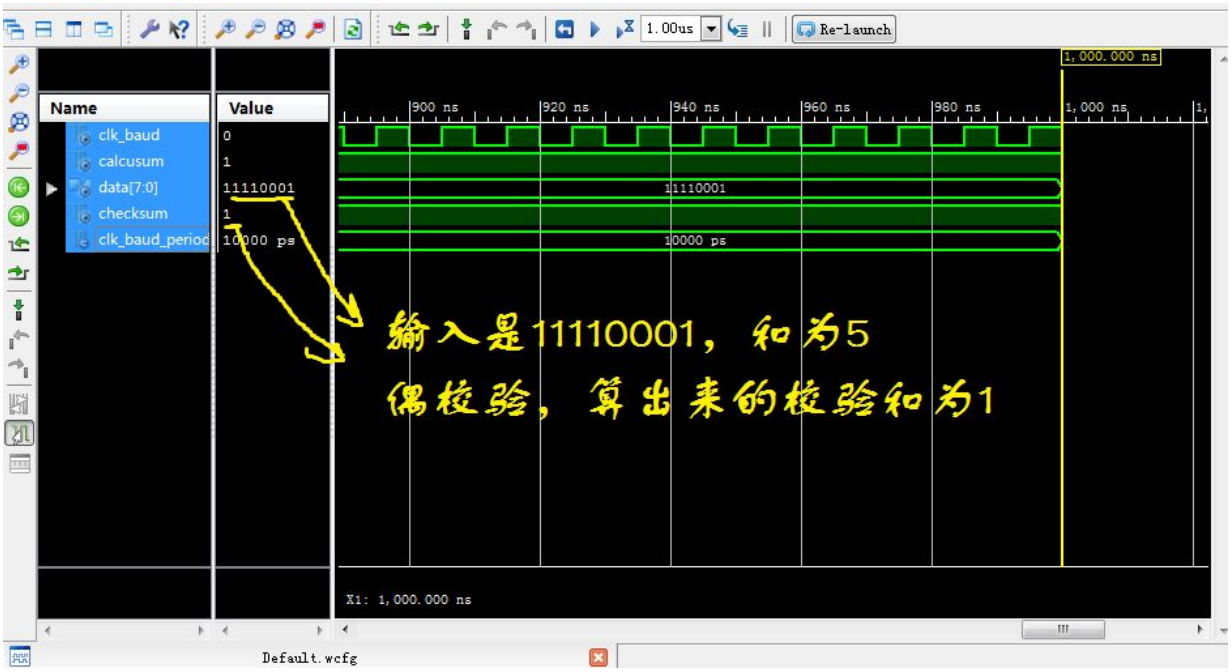


3、calculate 模块：

计算校验和。

```
---------calculate模块：计算校验和----------------------
entity calculate is
    Port ( clk_baud: in  STD_LOGIC;
           CalcuSum : in  STD_LOGIC;
           CheckSum : out  STD_LOGIC;
           data : in  STD_LOGIC_VECTOR (7 downto 0));
end calculate;

architecture Behavioral of calculate is

begin
process(clk_baud,data,CalcuSum)
begin
    if clk_baud'event and clk_baud='1' and CalcuSum='1' then
CheckSum<=data(7) xor data(6)xor data(5)xor data(4)xor data(3)xor data(2)xor data(1)xor data(0);
    end if;
end process;
end Behavioral;
```

模块仿真结果：

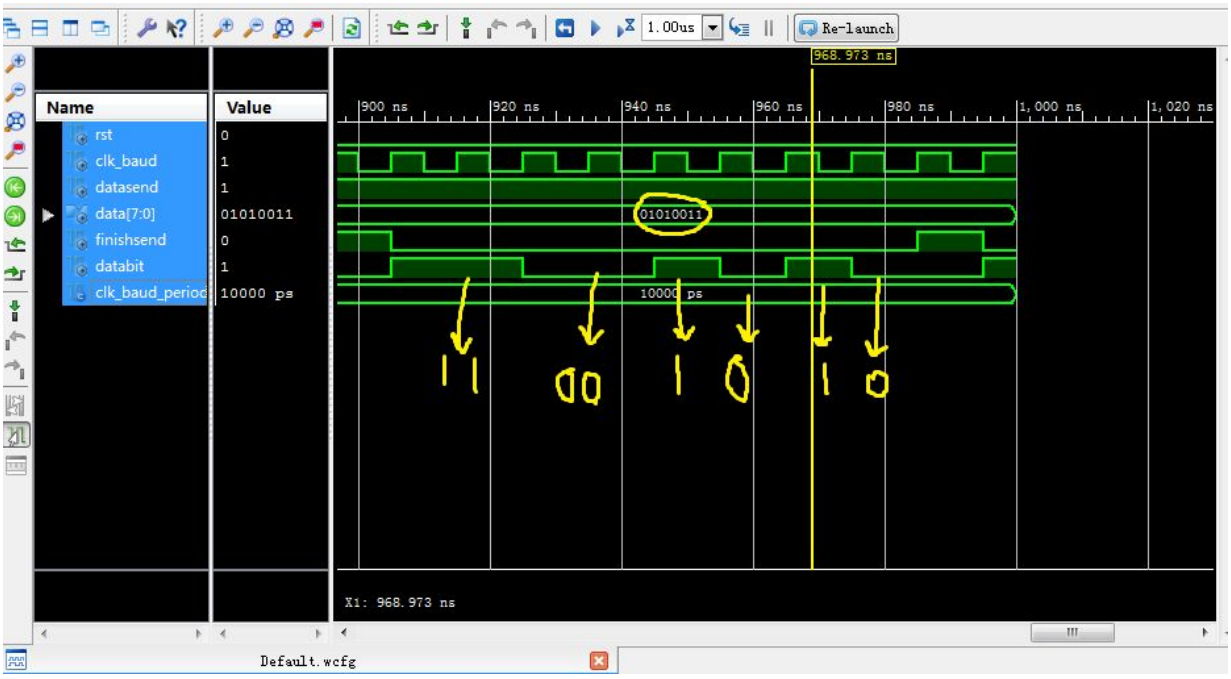


4、shift_out 模块：

并行数据串行输出。

```
-------shift_out模块：并行数据串行输出----------------------
entity shift_out is
    Port ( rst:in  STD_LOGIC;
           clk_baud:in  STD_LOGIC;
           DataSend : in  STD_LOGIC;
           FinishSend : out  STD_LOGIC;
           data : in  STD_LOGIC_VECTOR (7 downto 0);
           databit : out  STD_LOGIC
          );
end shift_out;

architecture Behavioral of shift_out is

begin
process(rst,clk_baud,DataSend)
variable count: integer range 0 to 8;
begin
    if rst='1' then
        count:=0;
        FinishSend<='0';
   elsif clk_baud'event and clk_baud='1' and DataSend='1' then
        if count=8 then
            FinishSend<='1';
            count:=0;
        else
            FinishSend<='0';
            databit<=data(count);
            count:=count+1;
        end if;
   end if;
end process;
end Behavioral;

```

模块仿真结果：

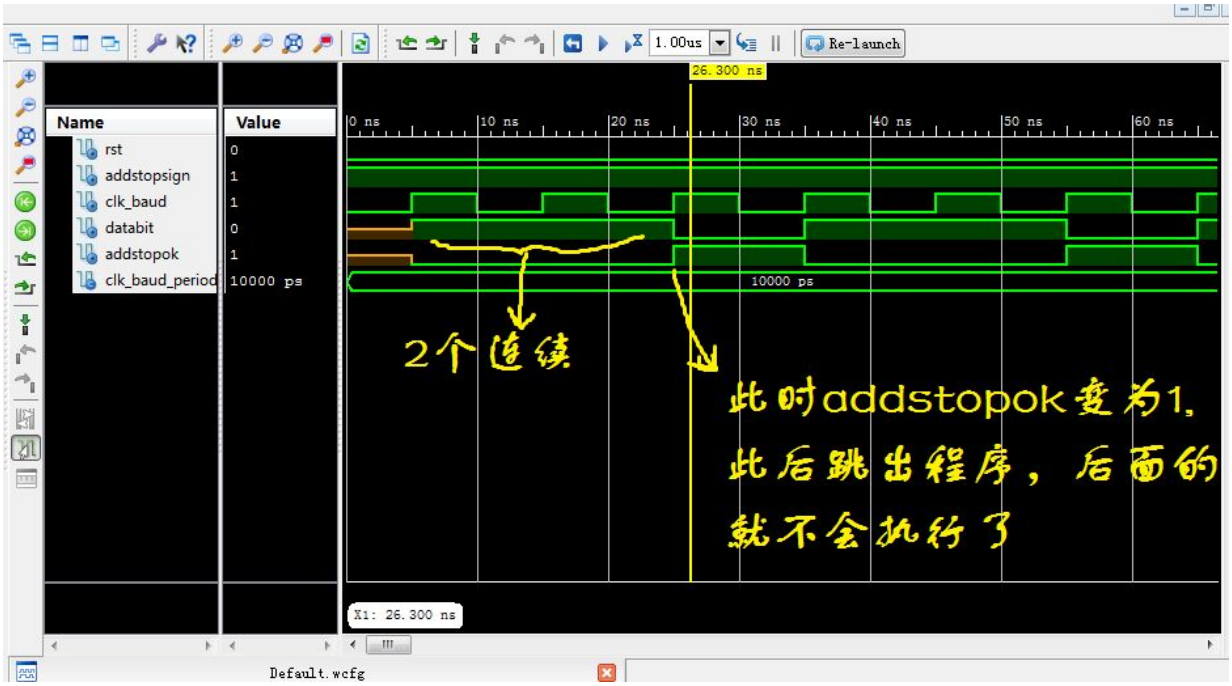


5、add_stop 模块：

添加结束位

```
--------add_stop模块：添加结束位----------------
entity add_stop is
    Port ( rst:in  STD_LOGIC;
           AddStopSign : in  STD_LOGIC;
           databit : out  STD_LOGIC;
           clk_baud : in  STD_LOGIC;
           AddStopOk : out  STD_LOGIC);
end add_stop;

architecture Behavioral of add_stop is

begin
process(clk_baud,AddStopSign,rst)
variable count: integer range 0 to 3;
begin
    if rst='1' then
        count:=0;
        databit<='1';
        AddStopOk<='0';
    elsif clk_baud'event and clk_baud='1' and AddStopSign='1' then
        databit<='1';
        count:=count+1;
        AddStopOk<='0';
        if count=3 then
            count:=0;
            AddStopOk<='1';
            databit<='0';
        end if;
    end if;
end process;
end Behavioral;
```

模块仿真结果：

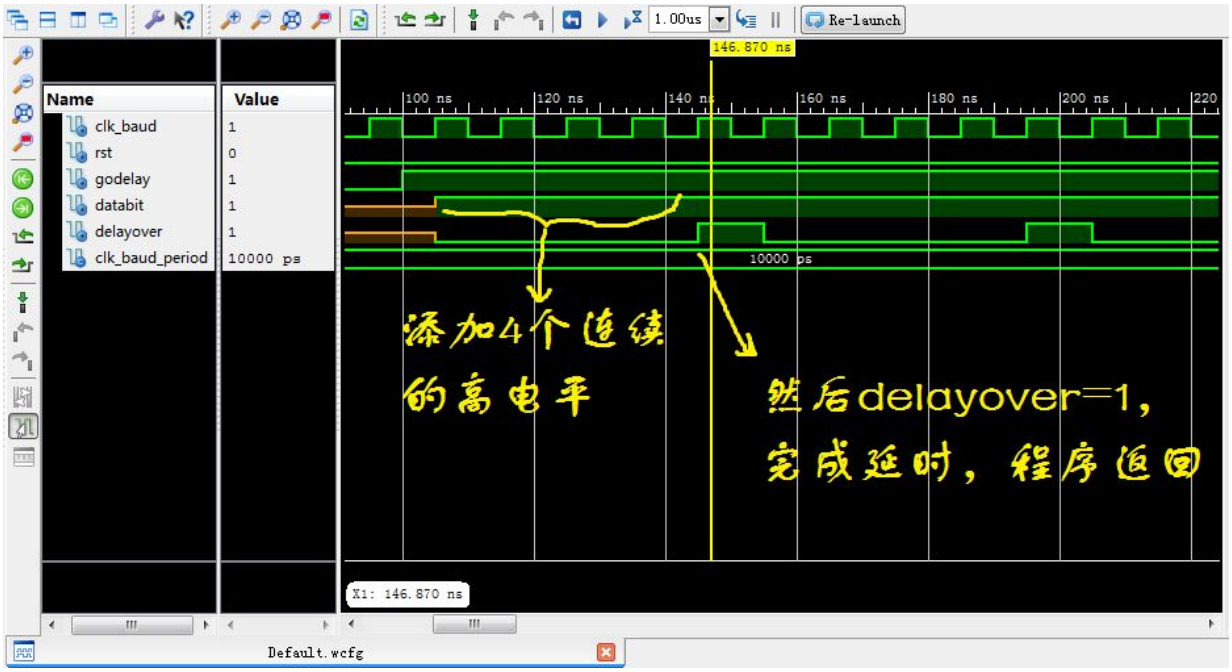


6、delay 模块：

延时，即添加空闲位

```
32
33  ------------delay模块：延时，即添加空闲位-----------------------
34  entity delay is
35      Port ( clk_baud:in  STD_LOGIC;
36             rst :in  STD_LOGIC;
37             databit:out  STD_LOGIC;
38             GoDelay : in  STD_LOGIC;
39             DelayOver : out  STD_LOGIC);
40  end delay;
41
42  architecture Behavioral of delay is
43
44  begin
45
46  process(clk_baud,GoDelay,rst,GoDelay)
47  variable count: integer range 0 to 4;
48
49  begin
50      if rst='1' then
51              count:=0;
52      elsif clk_baud'event and clk_baud='1' and GoDelay='1' then
53              if count=4 then       --4个连续的高电平
54                  DelayOver<='1';
55                  count:=0;
56              else
57                  databit<='1';
58                  DelayOver<='0';
59                  count:=count+1;
60              end if;
61      end if;
62  end process;
63
64  end Behavioral;
```

模块仿真结果：


1.00us
Re-launch
146.870 ns
Name
Value
clk_baud
1
rst
0
godelay
1
databit
1
delayover
1
clk_baud_period
10000 ps
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220
10000 ps
添加4个连续
的高电平
然后delayover=1,
完成延时，程序返回
X1: 146.870 ns
Default.wcfg

7、