

实验三、FPGA 串行通用异步收发器设计

实验目的： 1、掌握 QuartusII6.0 等 EDA 工具软件的基本使用；
2、熟悉 VHDL 硬件描述语言编程及其调试方法；
3、学习用 FPGA 实现接口电路设计。

实验内容：

本实验目标是利用 FPGA 逻辑资源，编程设计实现一个串行通用异步收发器。实验环境为 EDA 实验箱。电路设计采用 VHDL 硬件描述语言编程实现，开发软件为 QuartusII6.0。

1、UART 简介

UART (Universal Asynchronous Receiver Transmitter 通用异步收发器) 是一种应用广泛的短距离串行传输接口。常用于短距离、低速、低成本的通讯中。8250、8251、NS16450 等芯片都是常见的 UART 器件。

基本的 UART 通信只需要两条信号线 (RXD、TXD) 就可以完成数据的相互通信，接收与发送是全双工形式。TXD 是 UART 发送端，为输出；RXD 是 UART 接收端，为输入。

UART 的基本特点是：

(1) 在信号线上共有两种状态，可分别用逻辑 1 (高电平) 和逻辑 0 (低电平) 来区分。在发送器空闲时，数据线应该保持在逻辑高电平状态。

(2) 起始位 (Start Bit)：发送器是通过发送起始位而开始一个字符传送，起始位使数据线处于逻辑 0 状态，提示接受器数据传输即将开始。

(3) 数据位 (Data Bits)：起始位之后就是传送数据位。数据位一般为 8 位一个字节的数 (也有 6 位、7 位的情况)，低位 (LSB) 在前，高位 (MSB) 在后。

(4) 校验位 (parity Bit)：可以认为是一个特殊的数据位。校验位一般用来判断接收的数据位有无错误，一般是奇偶校验。在使用中，该位常常取消。

(5) 停止位：停止位在最后，用以标志一个字符传送的结束，它对应于逻辑 1 状态。

(6) 位时间：即每个位的时间宽度。起始位、数据位、校验位的位宽度是一致的，停止位有 0.5 位、1 位、1.5 位格式，一般为 1 位。

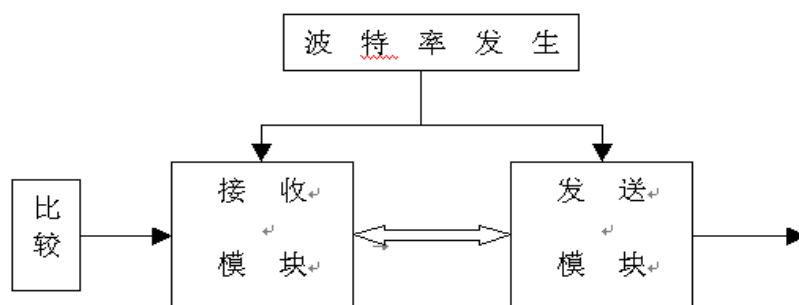
(7) 帧：从起始位开始到停止位结束的时间间隔称之为帧。

(8) 波特率：UART 的传送速率，用于说明数据传输的快慢。在串行通信中，数据是按位进行传送的，因此传送速率用每秒钟传送数据位的数目来表示，称之为波特率。如波特率 $9600=9600\text{bps}$ (位/秒)。

UART 的数据帧格式为：

START	D0	D1	D2	D3	D4	D5	D6	D7	P	STOP
起始位	数 据 位								校验位	停止位

FPGA UART 系统组成：如下图所示，FPGA UART 由三个子模块组成：波特率发生器；接收模块；发送模块；



2、模块设计：

系统由四部分组成：顶层模块；波特率发生器；UART 接收器； UART 发送器

1) 顶层模块

异步收发器的顶层模块由波特率发生器、UART 接收器和 UART 发送器构成。

UART 发送器的用途是将准备输出的并行数据按照基本 UART 帧格式转为 TXD 信号串行输出。

UART 接收器接收 RXD 串行信号，并将其转化为并行数据。

波特率发生器就是专门产生一个远远高于波特率的本地时钟信号对输入 RXD 不断采样，使接收器与发送器保持同步。

2) 波特率发生器

波特率发生器实际上就是一个分频器。

可以根据给定的系统时钟频率（晶振时钟）和要求的波特率算出波特率分频因子，算出的波特率分频因子作为分频器的分频数。

波特率分频因子可以根据不同的应用需要更改。

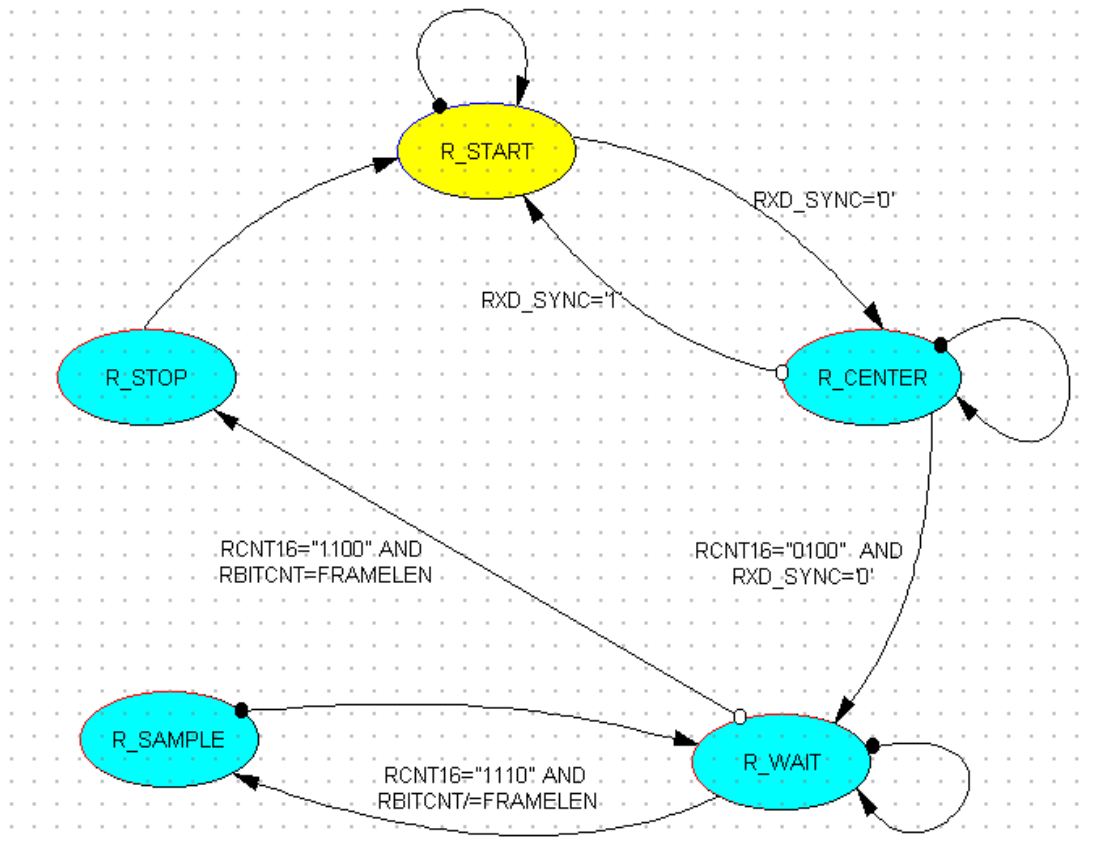
3) UART 接收器

由于串行数据帧和接收时钟是异步的，由逻辑 1 转为逻辑 0 可以被视为一个数据帧的起始位。

然而，为了避免毛刺影响，能够得到正确的起始位信号，必须要求接收到的起始位在波特率时钟采样的过程中至少有一半都是属于逻辑 0 才可认定接收到的是起始位。由于内部采样时钟 bclk 周期（由波特率发生器产生）是发送或接收波特率时钟频率的 16 倍，所以起始位需要至少 8 个连续 bclk 周期的逻辑 0 被接收到，才认为起始位接收到，接着数据位和奇偶校验位将每隔 16 个 bclk 周期被采样一次（即每一个波特率时钟被采样一次）。

如果起始位的确是 16 个 bclk 周期长，那么接下来的数据将在每个位的中点处被采样。

UART 接收器的接收状态机



接收状态机一共有 5 个状态： R_START（等待起始位）； R_CENTER（求中点）； R_WAIT（等待采样）； R_SAMPLE（采样）； R_STOP（停止位接收）。

R_START 状态

当 UART 接收器复位后，接收状态机将处于这一个状态。

在此状态，状态机一直在等待 RXD 的电平跳转，从逻辑 1 变为逻辑 0，即起始位，这意味着新的一帧 UART 数据帧的开始，一旦起始位被确定，状态机将转入 R_CENTER 状态。

状态图中的 RXD_SYNC 信号是 RXD 的同步信号，因为在进行逻辑 1 或逻辑 0 判断时，不希望检测的信号是不稳定的，所以不直接检测 RXD 信号，而是检测经过同步后的 RXD_SYNC 信号。

R_CENTER 状态

对于异步串行信号，为了使每一次都检测到正确的位信号，而且在较后的数据位检测时累计误差较小，显然在每位的中点检测是最为理想的。

在本状态，就是由起始位求出每位的中点，通过对 bclk 的个数进行计数（RCNT16），但计数值不是想当然的“1000”，要考虑经过一个状态，也即经过了一个 bclk 周期，所希望得到的是在采样时 1/2 位。

另外，可能在 R_START 状态检测到的起始位不是真正的起始位，可能是一个偶然出现的干扰尖脉冲（负脉冲）。这种干扰脉冲的周期是很短的，所以可以认为保持逻辑 0 超过 1/4 个位时间的信号一定是起始位。

R_WAIT 状态

当状态机处于这一状态，等待计满 15 个 bclk，在第 16 个 bclk 是进入 R_SAMPLE 状态进行数据位的采样检测，同时也判断是否采集的数据位长度已达到数据帧的长度（FRAMELEN），如果到来，就说明停止位来临了。

FRAMELEN 在设计时是可更改的（使用了 Generic），在本设计中默认为 8，即对应的 UART 工作在 8 位数据位、无校验位格式。

R_SAMPLE 状态

即数据位采样检测，完成后无条件状态机转入 R_WAIT 状态，等待下次数据位的到来。

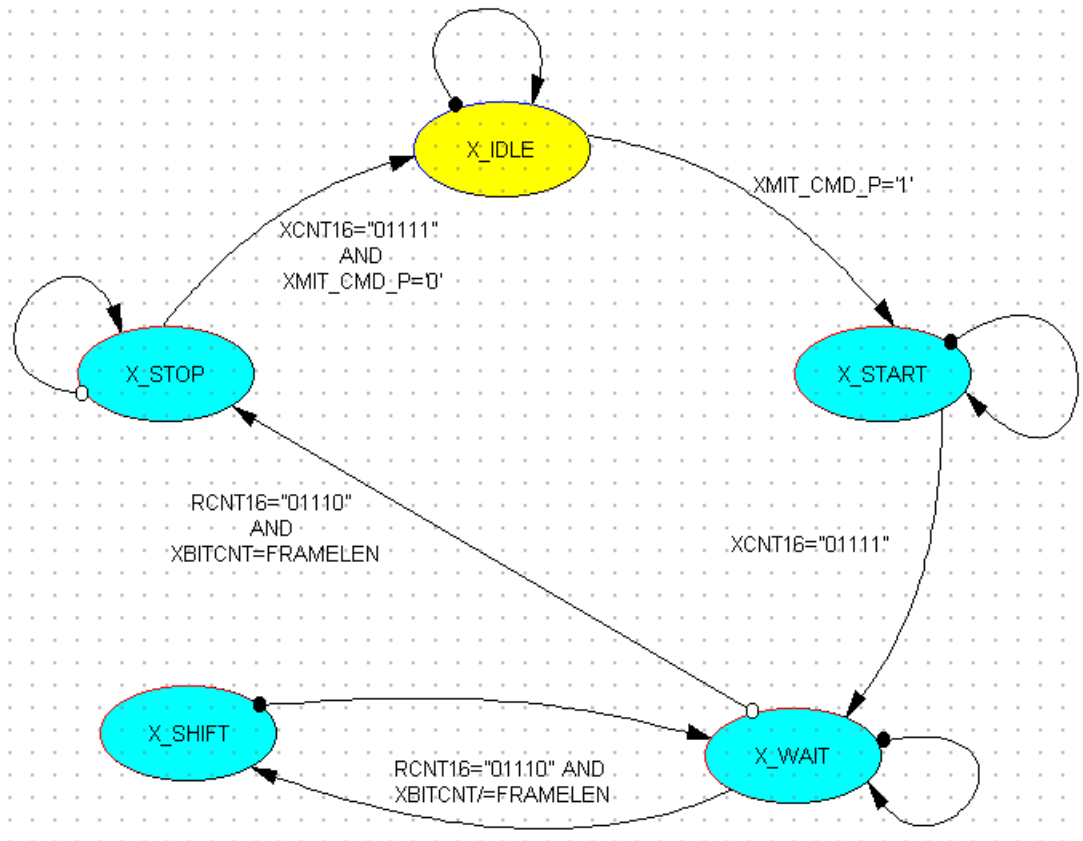
R_STOP 状态

无论停止位是 1 还是 1.5 位，或是 2 位，状态机在 R_STOP 不具体检测 RXD，只是输出帧接收完毕信号（REC_DONE<='1'），停止位后状态机转回到 R_START 状态，等待下一个帧的起始位。

4) UART 发送器

发送器只要每隔 16 个 bclk 周期输出 1 个数据即可，次序遵循第 1 位是起始位，第 8 位是停止位。在本设计中没有校验位，但只要改变 Generic 参数 FrameLen，也可以加入校验位，停止位是固定的 1 位格式。

发送状态机的状态图



发送状态机一共有 5 个状态：X_IDLE（空闲）；X_START（起始位）；X_WAIT（移位等待）；X_SHIFT（移位）；X_STOP（停止位）。

X_IDLE 状态：

当 UART 被复位信号复位后，状态机将立刻进入这一状态。

在这个状态下，UART 的发送器一直在等待一个数据帧发送命令 XMIT_CMD。

XMIT_CMD_P 信号是对 XMIT_CMD 的处理，XMIT_CMD_P 是一个短脉冲信号。这由于 XMIT_CMD 是一个外加信号，在 FPGA 之外，不可能对 XMIT_CMD 的脉冲宽度进行限制，如果 XMIT_CMD 有效在 UART 发完一个数据帧后仍然有效，那么就会错误地被认为，一个新的数据发送命令又到来了，UART 发送器就会再次启动 UART 帧的发送，显然该帧的发送是错误的。

在此对 XMIT_CMD 进行了脉冲宽度的限定，XMIT_CMD_P 就是一个处理后的信号。

当 XMIT_CMD_P= '1'，状态机转入 X_START，准备发送起始位。

X_START 状态：

在这个状态下，UART 的发送器一个位时间宽度的逻辑 0 信号至 TXD，即起始位。紧接着状态机转入 X_WAIT 状态。

XCNT16 是 bclk 的计数器

X_WAIT 状态

同 UART 接收状态机中的 R_WAIT 状态类似。

X_SHIFT 状态

当状态机处于这一状态时，实现待发数据的并串转换。转换完成立即回到 X_WAIT 状态。

X_STOP

停止位发送状态，当数据帧发送完毕，状态机转入该状态，并发送 16 个 bclk 周期的逻辑

辑 1 信号，即 1 位停止位。

状态机送完停止位后回到 X_IDLE 状态，并等待另一个数据帧的发送命令。

实验步骤:

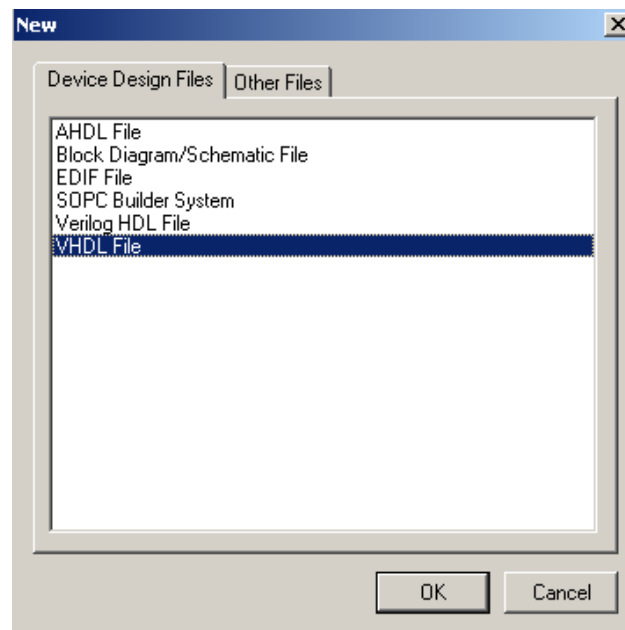
1、 创建工程文件

按照 Quartus 软件新建工程向导建立工程，工程名设为：uart_test（可自行命名）。

2、 子模块电路设计（包括各个模块的功能仿真）

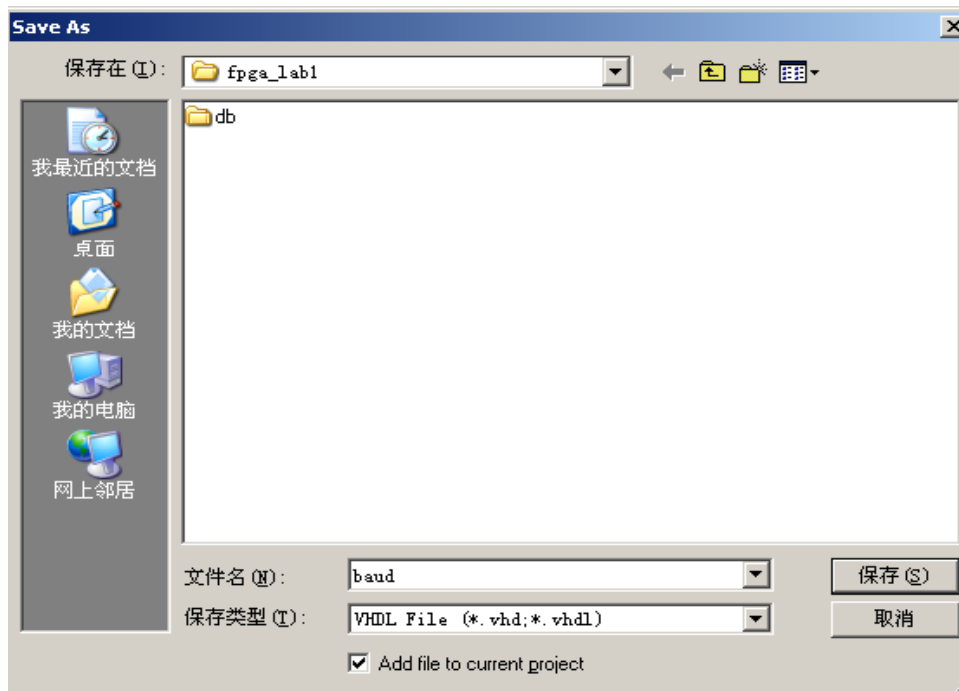
1) 波特率发生器

程序编写：在“文件”菜单下选择“New”，在弹出的窗口点击“VHDL File” 点击“OK”
打开 vhd1 编辑窗口。



编辑输入波特率发生器程序，编辑完毕后保存，文件名保存为“baud”

（注：文件名必须与程序中实体名一致）



选中“Add file to current project”选项，添加当前文件到项目。

-- 文件名：baud.vhd.

--功能：本实验想要实现的波特率为：9600，EDA 实验箱上晶振频率为： 4MHZ,

--波特率发生器的分频数计算如下式：

-- $4000000 / (16 * 9600) = 26$

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity baud is

 generic(framlenr:integer:=26);

 Port (clk,resetb:in std_logic;

 bclk :out std_logic);

end baud;

architecture Behavioral of baud is

 begin

 process(clk,resetb)

 variable cnt:integer;

 begin

 if resetb='1' then cnt:=0; bclk<='0'; --复位

 elsif rising_edge(clk) then

 if cnt>=framlenr then cnt:=0; bclk<='1';--设置分频系数

 else cnt:=cnt+1; bclk<='0';

 end if;

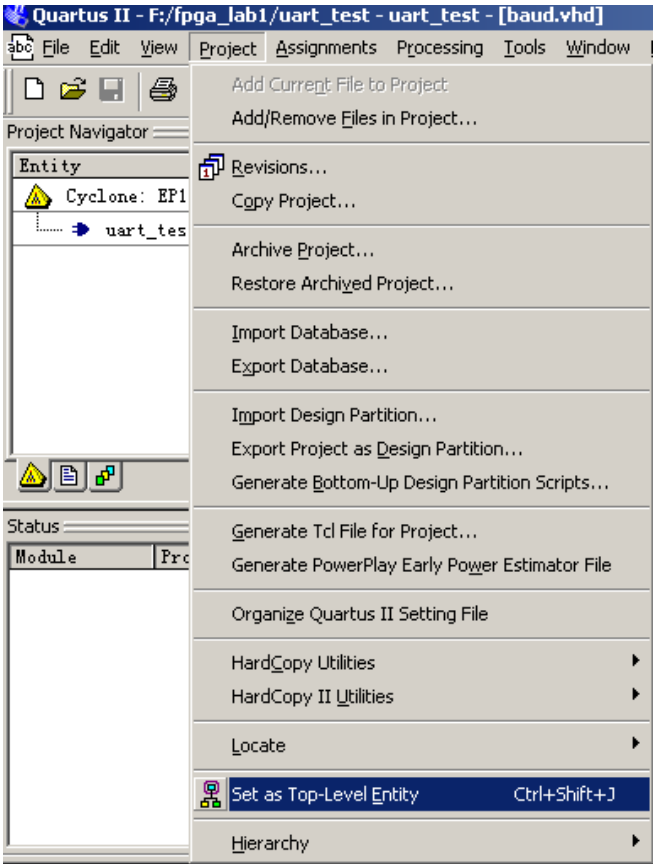
 end if;

 end process;

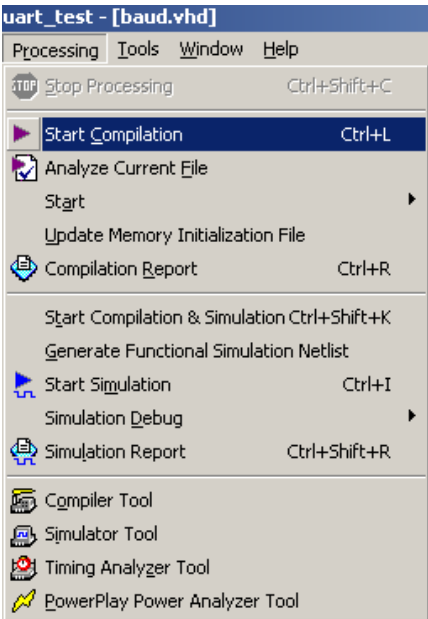
end Behavioral;

文件编译：保存文件后，选择“Project”菜单，点击“Set as Top-Level Entity”项，把当前文件设置为顶层实体。

（注：Quartus 环境下所有操作（综合、编译、仿真、下载等）都只对顶层实体进行，所以编译任何程序前，必须先设置该选项，把当前要编译的文件设置为顶层实体后，才能对该文件进行编译等操作）



打开“Processing”菜单，点击“Start Compilation”执行完全编译



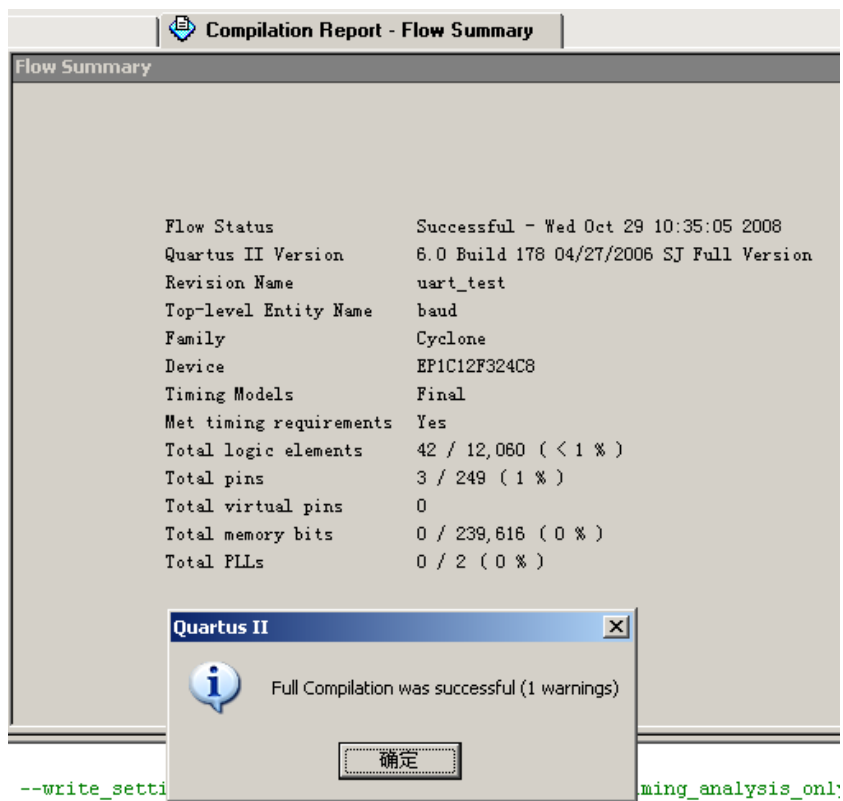
状态窗口显示编译过程进度信息

Module	Progress %	Time
Full Compilation	51 %	00:00:00
Analysis & Synthesis	100 %	00:00:00
Fitter	100 %	00:00:00
Assembler	4 %	00:00:00
Timing Analyzer	0 %	00:00:00

编译结束，系统会弹出编译结束窗口，报告错误与警告数，点击“确定”。

编译报告给出所有编译结果信息，包括硬件信息、资源占用率等。

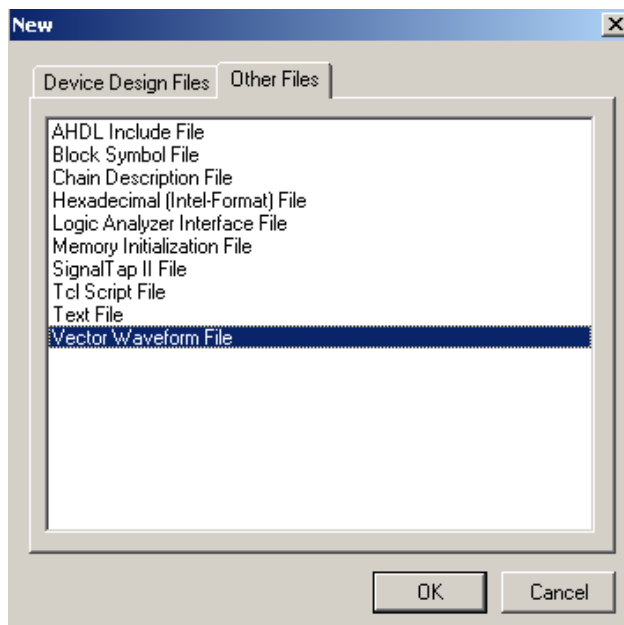
错误修改：如果程序中有错误，需要根据“Messages”消息栏给出的错误提示修改程序，保存后须再次编译，直至所有错误均改正后，方可执行下一步操作。警告信息可以忽略。



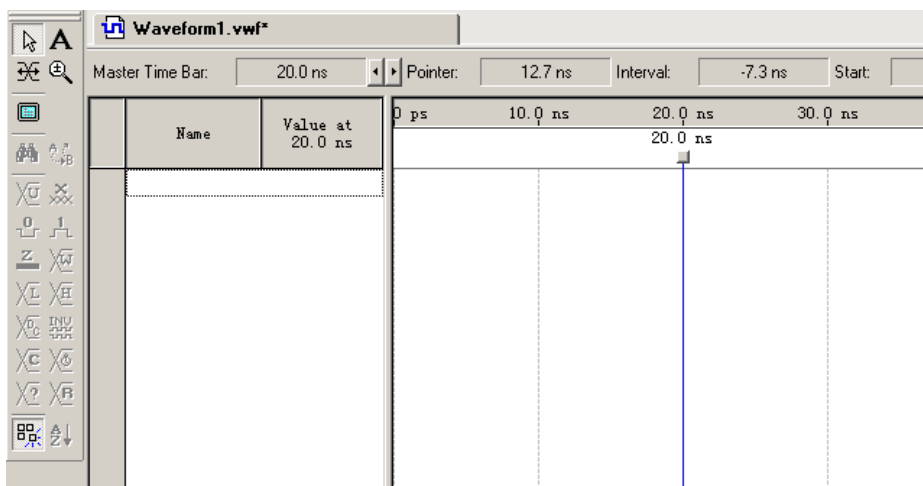
波形仿真：

1. 建立波形图文件

关闭编译报告窗口，在“文件”菜单下选择“New”，选中“other files”标签页，在弹出的窗口点击“Vector Waveform File”点击“OK” 打开波形编辑窗口。

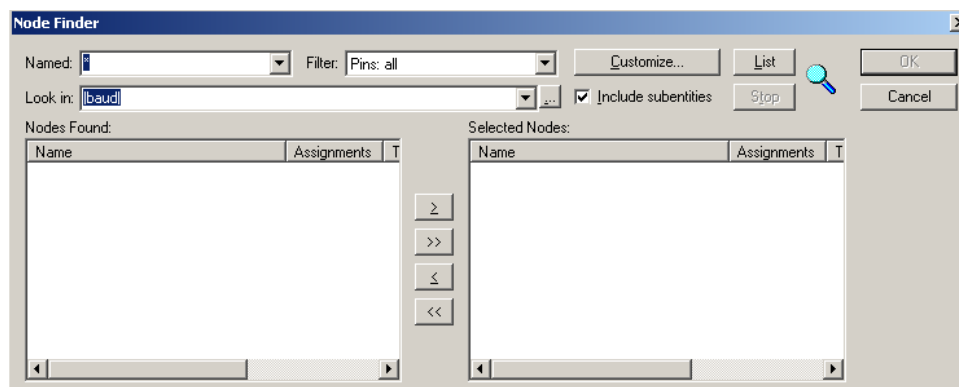


波形图编辑窗口

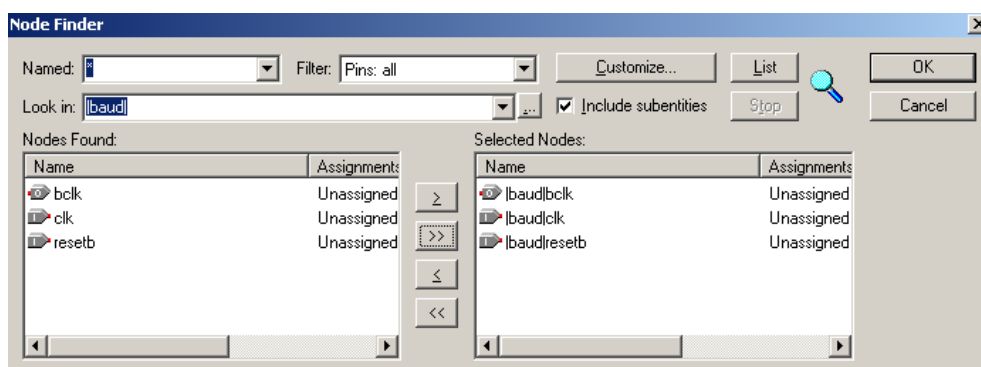


2. 定义仿真观测的输入输出节点

在波形编辑窗口左侧栏内单击鼠标右键，出现浮动菜单，选择“Insert Note or Bus...”出现“Insert Note or Bus...”对话框，点击“Node Finder...”按钮，出现“Node Finder”对话框，如下图所示。

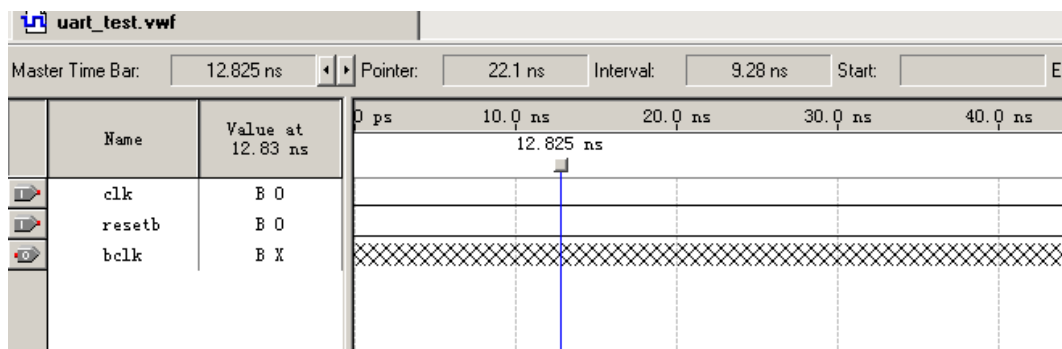


在图中“Filter:”选项下选择管脚类型为“Pins: all”，然后单击 List 按钮，可在左下侧区域看到设计项目中的输入输出信号，单击按钮“= >”，将这些信号选择到“Selected Nodes”区，表示对这些信号进行观测，单击 OK



此时的波形编辑窗口如下图所示。


保存波形文件，文件名为 uart_test.vwf（注：扩展名默认不填，文件名与项目名同名）

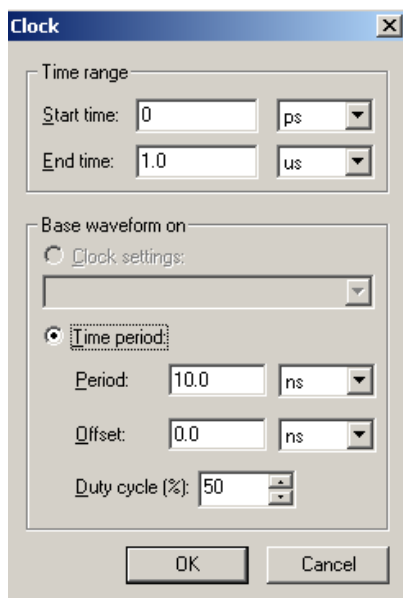


3. 为输入信号赋值

波形编辑器窗口左侧为信号赋值工具条，根据实际要求点选工具按钮对输入信号赋值。

1) 为时钟信号 clk 赋值：单击 clk，使其呈蓝色即选中了 clk，单击为时钟信号赋值工

具按钮 ，弹出 Clock 对话框，在 Period 框中输入合适的时钟周期数，其它值按默认即可，点击 OK。

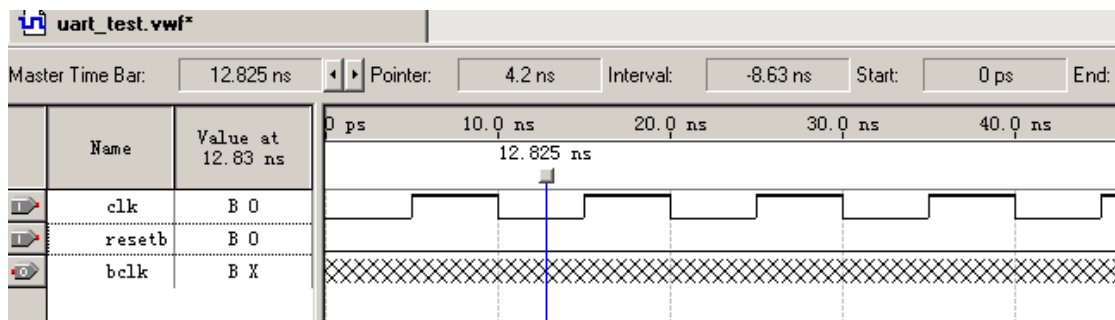


2) 为复位信号 resetb 赋值：选中 resetb，单击赋值 0 工具条按钮，为 resetb 赋值逻辑 0。

3) 设置仿真时间

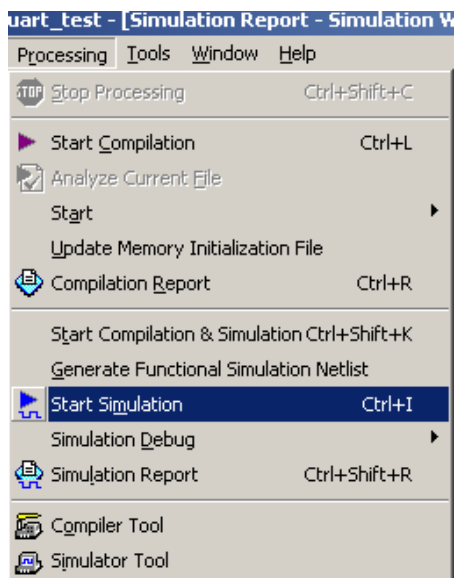
选择 Edit 菜单下的 End Time... 选项，打开 End Time 对话框，在 time 框内输入 100 单位为 us。

再次保存波形文件，窗口如下

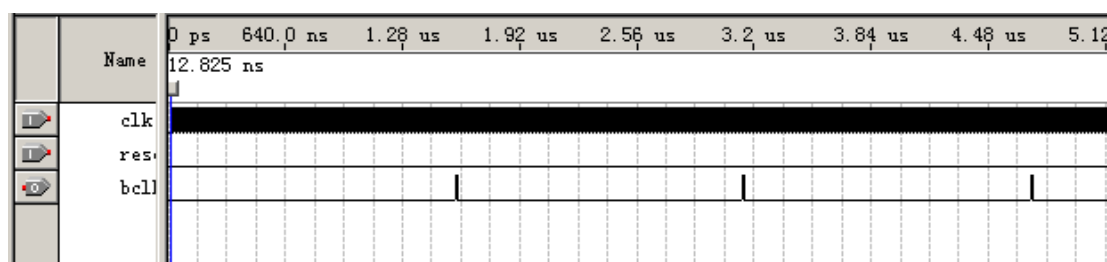


4. 时序仿真

选择 Processing 菜单下的 Begin Simulation 选项，即开始波形仿真。状态窗口会显示出仿真进程，



仿真结束后可以看到仿真结果波形，如下图所示。观察波形可用工具条上的放大缩小按钮放大缩小波形图，下图即为已经缩小了很多倍后的波形图。

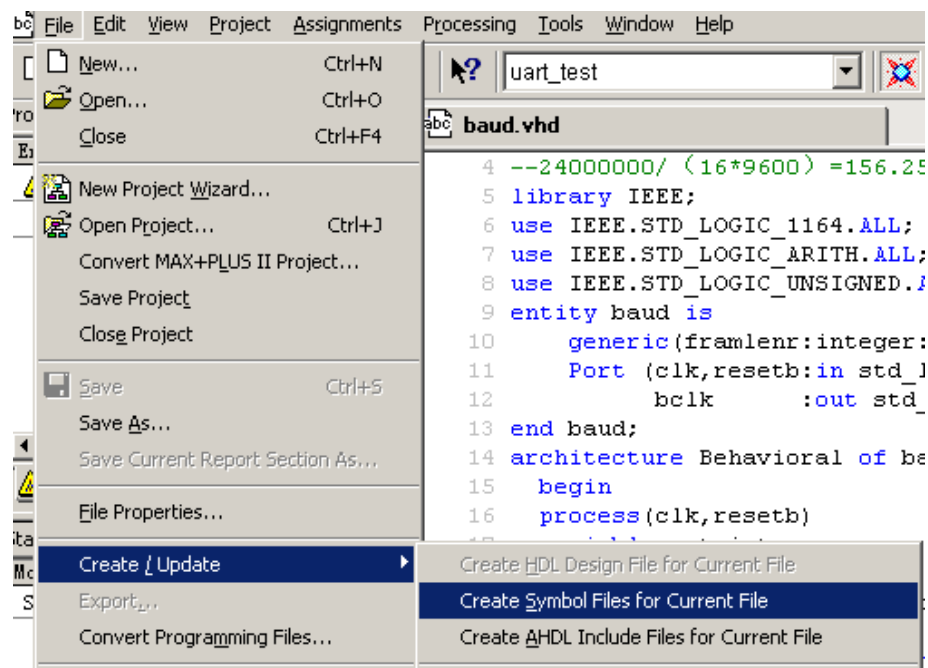


结果分析：图上可以观察到输出端 bckl 等间隔的有脉冲信号输出，通过标尺可以计算出它的脉冲输出频率及其与 clk 输入信号的关系。

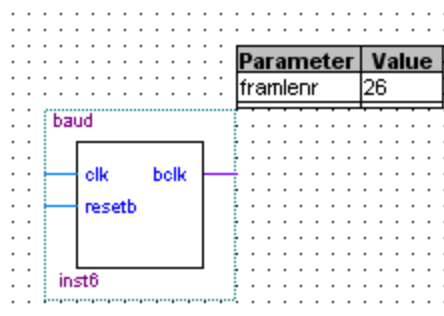
该程序实现的是波特率发生器，输出频率为时钟输入频率的 1/26。

生成符号文件：

通过波形仿真可以确定程序功能是否正确后，就可以把该程序生成符号文件，以便在后面的程序中调用。具体操作是：打开 File 文件菜单，选择 Creat/Updata 菜单项，右侧弹出子菜单再选择 Creat Symbol files for Current file 把当前文件创建成符号文件。状态窗口有进度信息显示。



生成的符号文件可以通过打开原理图窗口调入，进行验证。下图为生成的波特率模块。



按照生成波特率模块的步骤完成以下其它模块的程序编写，并进行仿真波形、功能验证，仿真通过后生成各自的符号文件。这里只给出仿真得到的波形图。

2) UART 发送器

UART 发送器程序：

-- 文件名：transfer.vhd

-- 功能：UART 发送器

-- 说明：系统由五个状态（x_idle, x_start, x_wait, x_shift, x_stop）和一个进程构成。

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity transfer is
```

```
    generic(framlent:integer:=8); --类属说明
```

```
    Port (bclkt,reset,xmit_cmd_p:in std_logic; --定义输入输出信号
```

```
          txdbuf                                :in std_logic_vector(7 downto 0);
```

```
          txd ,txd_done                          :out std_logic);
```

```
end transfer;
```

```

architecture Behavioral of transfer is
type states is (x_idle,x_start,x_wait,x_shift,x_stop);    --定义 5 个子状态
signal state:states:=x_idle;
signal tcnt:integer:=0;
begin
    process(bclkt,reset,xmit_cmd_p,txdbuf)  --主控时序进程
        variable xcnt16:std_logic_vector(4 downto 0):="00000"; --定义中间变量
        variable xbitcnt:integer:=0;
        variable txds:std_logic;
    begin
        if reset='1' then state<=x_idle;    --复位，txd 输出保持 1
            txd_done<='0';
            txds:='1';
        elsif rising_edge(bclkt) then
        case state is
            when x_idle=>    --状态 1，等待数据帧发送命令
                if xmit_cmd_p='1' then state<=x_start;txd_done<='0';
                else state<=x_idle;
                end if;
            when x_start=>    --状态 2，发送信号至起始位
                if xcnt16="01111" then state<=x_shift;xcnt16:="00000";
                else xcnt16:=xcnt16+1;txds:='0';state<=x_start;    --输出开始位，'0'
                end if;
            when x_wait=>    --状态 3，等待状态
                if xcnt16>="01110" then
                    if xbitcnt=framlent then state<=x_stop;xbitcnt:=0;xcnt16:="00000";
                    else state<=x_shift;
                    end if;
                    xcnt16:="00000";
                else xcnt16:=xcnt16+1;state<=x_wait;
                end if;
            when x_shift=>    --状态 4，将待发数据进行并串转换
                txds:=txdbuf(xbitcnt);
                xbitcnt:=xbitcnt+1;
                state<=x_wait;
            when x_stop=>    --状态 5，停止位发送状态
                if xcnt16>="01111" then
                    if xmit_cmd_p='0' then state<=x_idle;--高电平保持时间应低于一个帧发送的时间
                    xcnt16:="00000";
                    else xcnt16:=xcnt16;state<=x_stop;
                    end if;
                    txd_done<='1';
                else xcnt16:=xcnt16+1;txds:='1';state<=x_stop;
                end if;
        end case;
    end process;
end architecture;

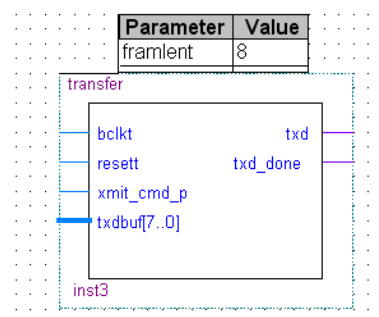
```

```

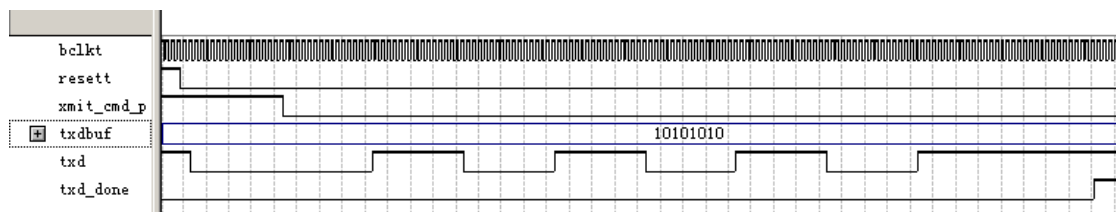
when others=>state<=x_idle;
end case;
end if;
txd<=txds;
end process;
end Behavioral;

```

生成的模块如下图所示：



仿真波形：



- 说明：1) resett: 为复位信号，高电平复位，初始时刻需要手动设置一段时间的高电平
2) Bclkt 为波特率发生信号，用时钟信号定义，周期设为 10ns。
3) 为了得到完整的仿真结果，仿真时间要设的长一些（100us）
4) xmit_cmd_p: 为发送控制信号，开始时刻设置一段高电平
5) txdbuf: 为要发送的八位数据，点击“+”后，可对每一个位信号分别赋值，
本波形设置为 txdbuf[7]、[5]、[3]、[1]位置“1”，[6]、[4]、[2]、[0]位置“0”
6) txd: 为发送器发出的串行信号
7) txd_done: 为发送结束信号，高电平有效

3) UART 接收器

UART 接收器程序：

```

--文件名: reciever.vhd.
--功能:UART 接受器。
--系统由五个状态 (r_start,r_center,r_wait,r_sample,r_stop) 和两个进程构成
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity reciever is
generic(framlenr:integer:=8);          --传送的数据位为 8 位
Port ( bclkr,resetr,rxdr:in std_logic; --定义输入输出信号
       r_ready      :out std_logic;

```

```

        rbuf      :out std_logic_vector(7 downto 0) );
end reciever;
architecture Behavioral of reciever is
    type states is (r_start,r_center,r_wait,r_sample,r_stop);--定义各子状态
    signal state:states:=r_start;
    signal rxd_sync:std_logic;    -- rxd_sync 内部信号，接受 rxd 输入
begin
    pro1:process(rxdr)
    begin
        if rxdr='0' then rxd_sync<='0';
        else rxd_sync<='1';
        end if;
    end process;
    pro2:process(bclk,resetr,rxd_sync)    --主控时序、组合进程
    variable count:std_logic_vector(3 downto 0); --定义中间变量
    variable rcnt:integer:=0;            -- rcnt 为接收的数据位数计数
    variable rbufs:std_logic_vector(7 downto 0);
    begin
        if resetr='1' then state<=r_start; count:="0000"; --复位
        elsif rising_edge(bclk) then    --波特率信号的上升沿
            --状态机
            case state is
                when r_start=>            --状态 1，等待起始位
                    if rxd_sync='0' then state<=r_center;
                    r_ready<='0';
                    rcnt:=0;
                    else state<=r_start; r_ready<='0';
                    end if;
                when r_center=>            --状态 2，求出每位的中点
                    if rxd_sync='0' then    --每个数据位被分为 16 等分，中点为 8
                        if count="0100" then state<=r_wait; count:="0000";
                        else count:=count+1; state<=r_center;
                        end if;
                    else state<=r_start;
                    end if;
                when r_wait=>            --状态 3，等待状态
                    if count>="1110" then
                        if rcnt=framenr then state<=r_stop;
                        -- rcnt=framenr 表示数据接收够 8 位
                        else state<=r_sample;
                        end if;
                    count:="0000";
                    else count:=count+1; state<=r_wait;
                    end if;
            end case;
        end if;
    end process;
end architecture;

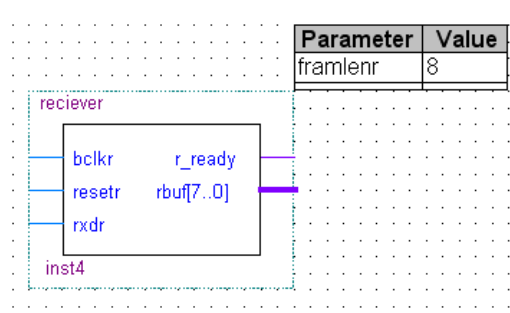
```

```

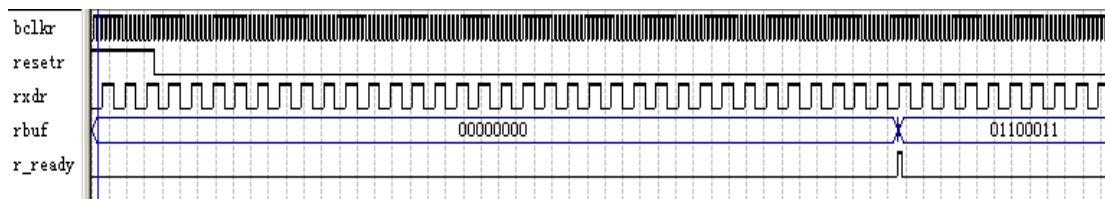
when r_sample=>rbufs(rcnt):=rxd_sync; --状态 4，数据位采样检测
rcnt:=rcnt+1;
state<=r_wait;
when r_stop=>r_ready<='1'; rbuf<=rbufs; --状态 5，输出帧接收完毕信号
state<=r_start;
when others=>state<=r_start;
end case;
end if;
end process;
end Behavioral;

```

生成的模块：



仿真波形：



- 说明：1) resett: 为复位信号，高电平复位，初始时刻需要手动设置一段时间的高电平
2) Bclkt 为波特率发生信号，用时钟信号定义，周期设为 10ns。
3) 为了得到完整的仿真结果，仿真时间要设的长一些（100us）
4) rxdr: 为预接收的串行信号，这里用周期信号模拟，周期设为 50ns
5) rbuf: 为接收到的八位数据
6) rxd_ready: 为接收器接收满 8 位数据后发出的结束信号，高电平有效

4) 其它功能模块

为了更直观的反应串口通信的运行结果，本实验还另外设计了两个模块，使得程序在做硬件功能验证时更加直观。

A、分频器模块

实验箱提供的晶振频率为：4Mhz，实验中需要较低的频率信号驱动计数器计数和提供给发送器触发信号（xmit_cmd_p），本实验分频器设置分频数为 1000000，得到 4hz 脉冲输出。

分频器程序：

```

--文件名: clock_div.vhd.
--功能: 实验开发板晶振频率为: 4MHZ, 经过分频后得到 4HZ 脉冲频率
--分频数计算: 4000000/4 = 1000000
library IEEE;

```

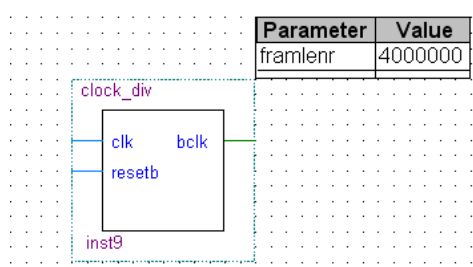


```

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity clock_div is
    generic(framlenr:integer:=1000000);
    Port (clk,resetb:in std_logic;
          bclk:      inout std_logic);
end clock_div;
architecture Behavioral of clock_div is
    begin
        process(clk,resetb)
            variable cnt:integer;
        begin
            if resetb='1' then cnt:=0; bclk<='0';
            elsif rising_edge(clk) then
                if cnt>=framlenr then cnt:=0; bclk<='0';
                elsif cnt>=framlenr/2 then cnt:=cnt+1;bclk<='1';
                else cnt:=cnt+1; bclk<='0';
                end if;
            end if;
        end process;
    end Behavioral;

```

生成的模块：



B、译码显示功能模块

说明：为了使接收器接收到的数据直观的在实验箱上显示出来，增加了译码显示程序，把 8 位接收数据在 LED 显示器（即 L29）上译码显示。

a: 输入信号，接收器接收到的 8 位数据。

q: 输出信号，八位数据经译码后输出。

程序：

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity decoder_8_8 is

```

```

    Port (
        a :in std_logic_vector(7 downto 0);

```

```

        q :out std_logic_vector(7 downto 0) );
end decoder_8_8;

```

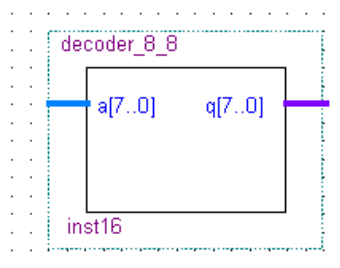
architecture Behavioral of decoder_8_8 is

```

begin
  process(a)
  begin
    case a is
      when x"00" => q <= "11000000";
      when x"01" => q <= "11111001";
      when x"02" => q <= "10100100";
      when x"03" => q <= "10110000";
      when x"04" => q <= "10011001";
      when x"05" => q <= "10010010";
      when x"06" => q <= "10000010";
      when x"07" => q <= "11111000";
      when x"08" => q <= "10000000";
      when x"09" => q <= "10010000";
      when x"0a" => q <= "10001000";
      when x"0b" => q <= "10000011";
      when x"0c" => q <= "11000110";
      when x"0d" => q <= "10100001";
      when x"0e" => q <= "10000110";
      when x"0f" => q <= "10001110";
      when others => q <= "11111111";
    end case;
  end process;
end Behavioral;

```

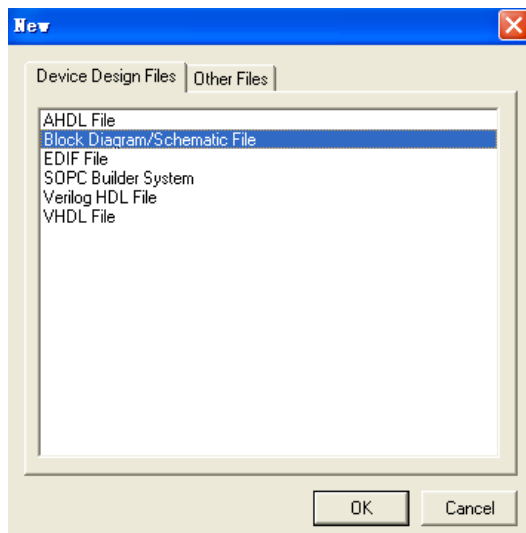
生成的模块：



3、顶层文件设计

新建原理图文件

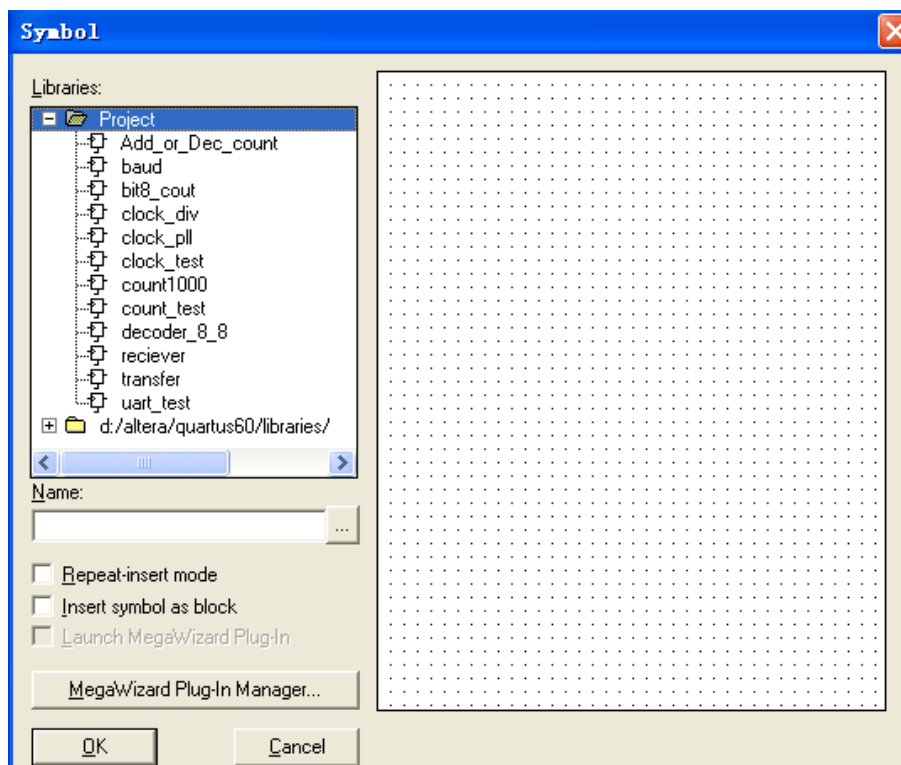
在“文件”菜单下选择“New”，在弹出的窗口点击“Block Diagram/Schematic File”，点击“OK”打开原理图编辑窗口。



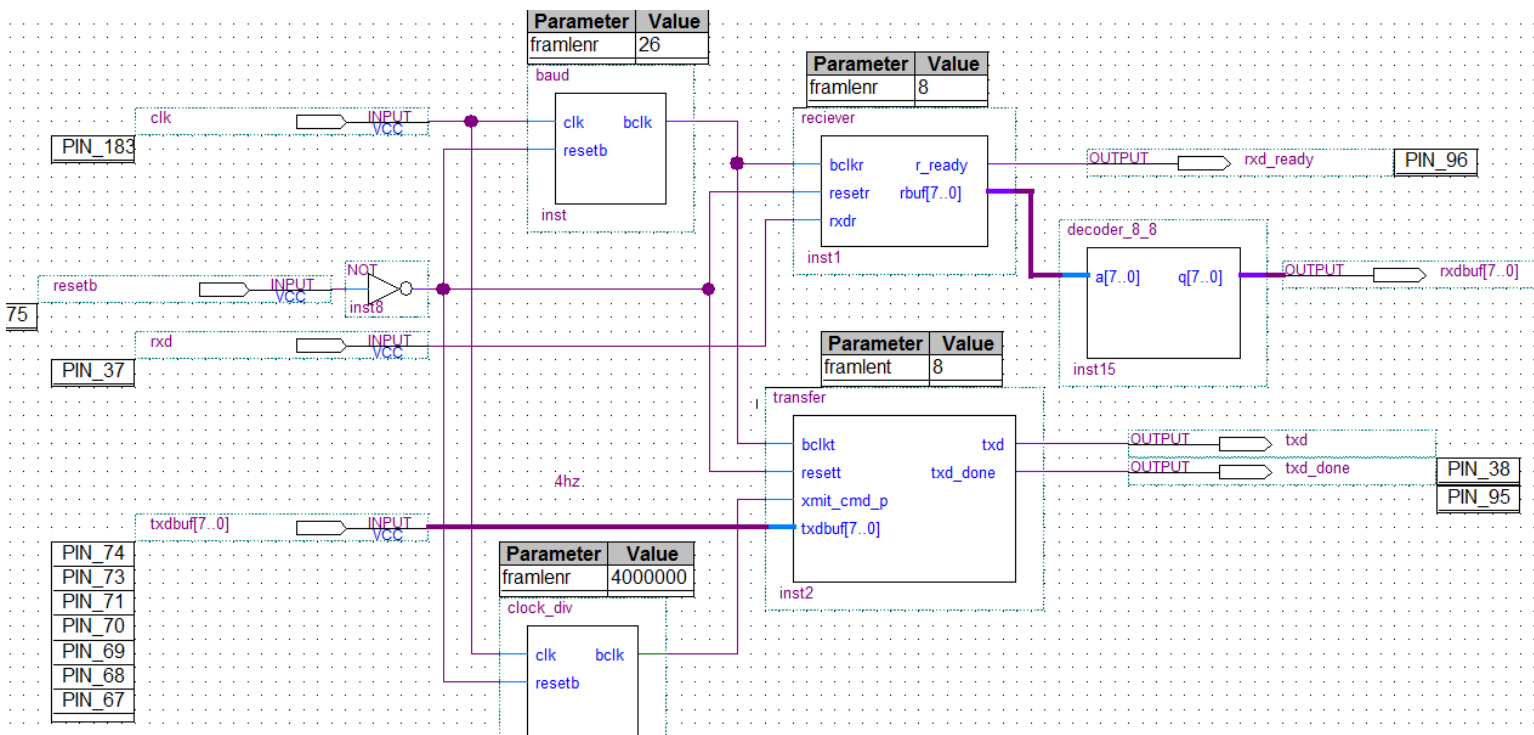
在窗口中双击鼠标左键，弹出 symbol 器件添加窗口，选择 Libraries（器件库目录）

Project: 为用户自己生成的符号库，如图所示，本实验已经建立的 baud、transfer、reciever 等符号都已经在目录中列出。

Altera/.../libraries/: 为系统自带的器件库目录



单击需要添加的模块，点击 OK 即可通过鼠标拖动到图中相应位置，按照下图添加器件并连接相应管脚。

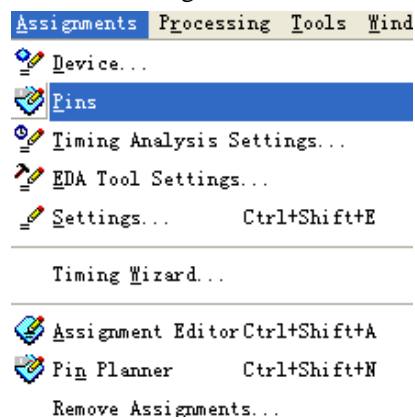


添加输入和输出管脚，符号库中输入为 input，输出为 output，可在 Symbol 窗口的 Name 栏内直接输入已知器件的名称来添加器件。

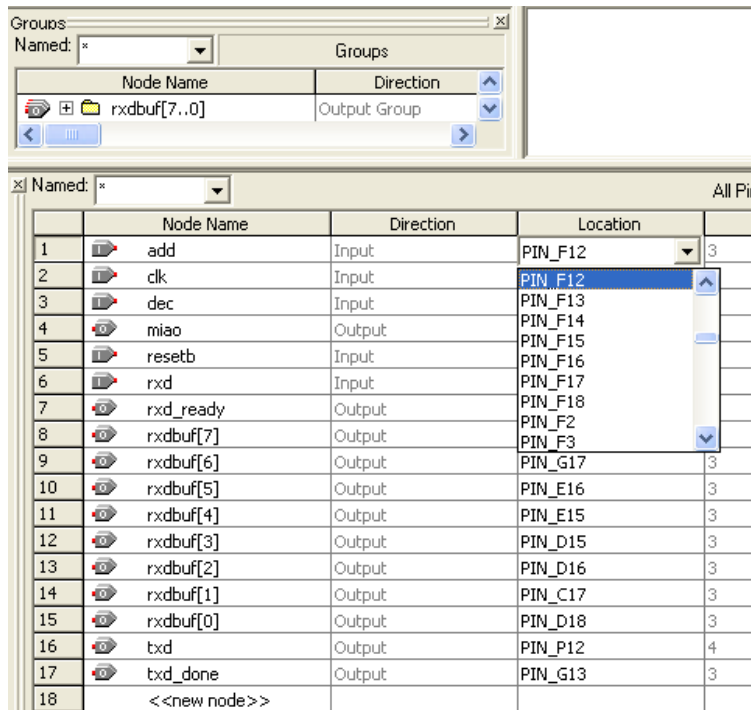
完成整个原理图文件的编辑后，保存文件名为 `uart_test`。设置该文件为顶层文件后，执行全编译。若发现错误，进行修改，再次编译直至成功。

4、器件管脚分配

打开 Assignments 菜单，选择 Pins 选项



弹出管脚分配窗口，在 Location 位置点击下拉按钮选择相应管脚



管脚分配表

信号名	对应器件名称	管脚号
clk	石英晶振	183
resetb	开关 S1	75
rx_d	RS232 接口电路发送端	37
rx_d_ready	发光二极管 L24	96
rxdbuf[7..0]	数码管 L29	168, 179, 177, 176, 175, 174, 173, 172
tx_d	RS232 接口电路接收端	38
tx_d_done	发光二极管 L23	95G
txdbuf[7..1]	开关 S8~S2	67, 68, 69, 70, 71, 73, 74
注: txdbuf[0]未分配		

管脚分配完成后，需要重新编译使之生效。

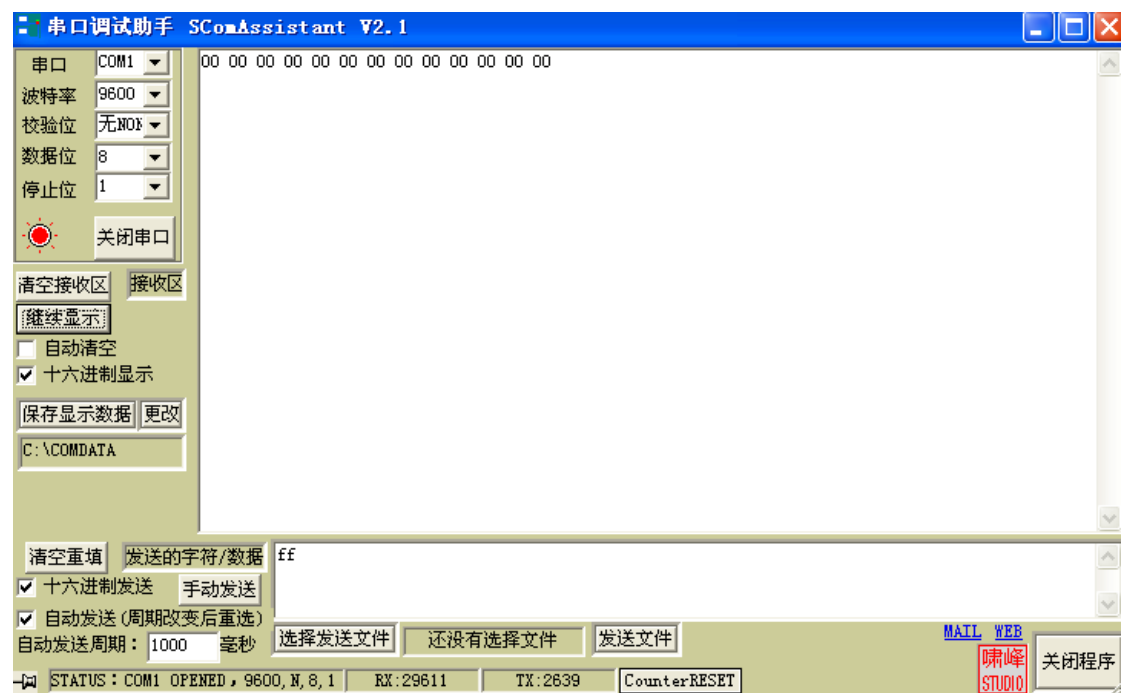
5、 硬件下载及调试

把程序下载到 FPGA 上，步骤略。

程序硬件调试

把实验箱串口插头与串口线（已经与 pc 机串口相连）联接

- 1) 检查 led 灯 L23,L24 是否闪烁，若闪烁则证明程序已经正确下载到 fpga 上
- 2) 点击复位按钮 S1，L23 灯开始闪烁，表示 fpga 已通过串口向外发出数据
- 3) 在 Pc 机上打开“串口调试助手”软件，设置如下图所示



串口选 COM1，波特率选 9600，无校验位，数据位 8，停止位 1，显示和发送都选为十六进制。

观察是否能接收到数据，发送数据，看实验箱数码管 L29 是否有变化。