1 引言

由于微电子学和计算机科学的迅速发展，给 EDA(电子设计自动化)行业带来了巨大的变化。特别是进入 20 世纪 90 年代后，电子系统已经从电路板级系统集成发展成为包括 ASIC、FPGA/CPLD 和嵌入系统的多种模式。可以说 EDA 产业已经成为电子信息类产品的支柱产业。EDA 之所以能蓬勃发展的关键因素之一就是采用了硬件描述语言(HDL)描述电路系统。就 FPGA 和 CPLD 开发而言，比较流行的 HDL 主要有 Verilog HDL、VHDL、ABEL-HDL 和 AHDL 等，其中 VHDL 和 Verilog HDL 因适合标准化的发展方向而最终成为 IEEE 标准。

下面的设计就是用 VHDL 来完成实现的。

2 UART 设计实例

通常设计数字电路大都采用自顶向下将系统按功能逐层分割的层次化设计方法，这比传统自下向上的 EDA 设计方法有更明显的优势(当时的主要设计文件是电路图)。因为由自顶向下的设计过程可以看出，从总体行为设计开始到最终逻辑综合，形成网络表为止。每一步都要进行仿真检查，这样有利于尽早发现系统设计中存在的问题，从而可以大大缩短系统硬件的设计周期。

下面以 UART 的设计为例具体说明：（本设计只对本设计的总模块做各种基于 MAX+PLUS II 环境下的各种分析，对于各分模块只是作些必要的说明。）

UART（即 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。UART 允许在串行链路上进行全双工的通信。串行外设用到的 RS232-C 异步串行接口，一般采用专用的集成电路即 UART 实现。如 8250、8251、NS16450 等芯片都是常见的 UART 器件，这类芯片已经相当复杂，有的含有许多辅助的模块（如 FIFO），有时我们不需要使用完整的 UART 的功能和这些辅助功能。或者设计上用到了 FPGA/CPLD 器件，那么我们就可以将所需要的 UART 功能集成到 FPGA 内部。使用 VHDL 将 UART 的核心功能集成，从而使整个设计更加紧凑、稳定且可靠。本文应用 EDA 技术，基于 FPGA/CPLD 器件设计与实现 UART。

2.1 UART 简介

2.1.1 UART 结构

UART 主要有由数据总线接口、控制逻辑、波特率发生器、发送部分和接收部分等组成。本设计主要设计 UART 中最重要的发送部分和接收部分，其他部分设计不在赘述。

功能包括发送缓冲器（tbr）、发送移位寄存器（tsr）、帧产生、奇偶校验、并转串、数据接收缓冲器（rbr）、接收移位寄存器（rsr）、帧产生、奇偶校验、串转并。图 1 是 UART 的典型应用。

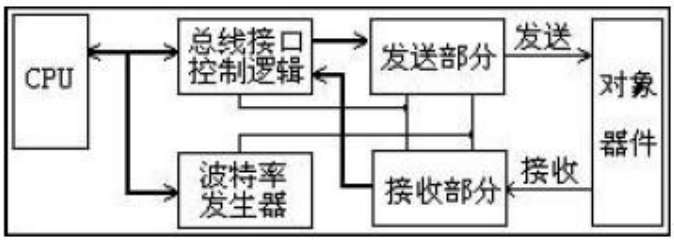


图 1

2.1.2 UART 的帧格式

UART 的帧格式如图 2 所示。

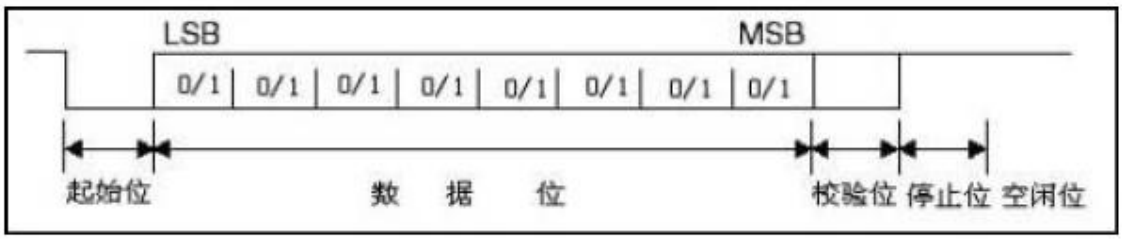


图 2

包括线路空闲状态（idle，高电平）、起始位(start bit，低电平)、5～8 位数据位(data bits)、校验位(parity bit，可选)和停止位(stop bit，位数可为 1、1.5、2 位)。

这种格式是由起始位和停止位来实现字符的同步 。

UART 内部一般有配置寄存器，可以配置数据位数（5～8 位）、是否有校验位和校验的类型、停止位的位数（1，1.5，2）等设置。

本设计没有奇偶校验位，所设置的奇偶校验只是检验数据中是否有奇数或偶数个 1。数据位为 8 位，停止位为 1 位。

2.2 UART 的设计与实现

2.2.1 UART 发送器

发送器每隔 16 个 CLK16 时钟周期输出 1 位，次序遵循 1 位起始位、8 位数据位、1 位停止位。

CPU 何时可以往发送缓冲器 tbr 写入数据，也就是说 CPU 要写数据到 tbr 时必须判断当前是否可写，如果不判这个条件，发送的数据会出错。本设计由 wrn 控制。

数据的发送是由微处理器控制，微处理器给出 wrn 信号，发送器根据此信号将并行数据 din[7..0]锁存进发送缓冲器 tbr[7..0]，并通过发送移位寄存器 tsr[7..0]发送串行数据至串行数据输出端 sdo。在数据发送过程中用输出信号 tbre、tsre 作为标志信号，当一帧数据由发送缓冲器 tbr[7..0]送到发送发送移位

寄存器 tsr[7..0]时，tbre 信号为 1，而数据由发送移位寄存器 tsr[7..0]串行发送完毕时，tsre 信号为 1，通知 CPU 在下个时钟装入新数据。

发送器端口信号如图 3 所示。

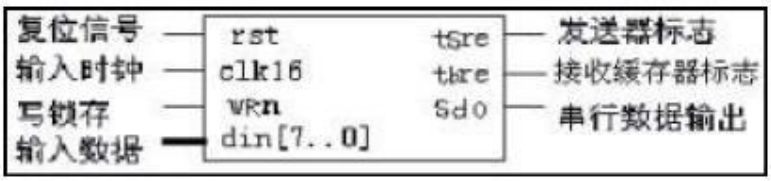


图 3

引入发送字符长度和发送次序计数器 no_bits_sent，实现设计的源程序如下。

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
entity txmit is
port (rst,clk16x,wrn : in std_logic ;
din : in std_logic_vector(7 downto 0) ;
tbre : out std_logic ;
tsre : out std_logic ;
sdo : out std_logic ) ;
end txmit ;

architecture v1 of txmit is
signal clk1x_enable : std_logic ;
signal tsr : std_logic_vector (7 downto 0) ;
signal tbr : std_logic_vector (7 downto 0) ;
signal parity : std_logic ;
signal clkdiv : unsigned (3 downto 0) ; --用来控制数据采样时钟的
signal clk1x : std_logic ;
signal no_bits_sent : unsigned (3 downto 0) ;
signal wrn1 : std_logic ;
signal wrn2 : std_logic ;
begin

process (rst,clk16x) --对 wrn 进行脉宽处理，以防接收数据错误
```

```
begin
if rst = '1' then
wrn1 <= '1' ;
wrn2 <= '1' ;
elsif clk16x'event and clk16x = '1' then
wrn2 <= wrn1 ;
wrn1 <= wrn ;
end if ;
end process ;

process (rst,clk16x) --对 clk1x_enable 进行控制
begin
if rst = '1' then
clk1x_enable <= '0' ;
tbre <= '1' ;
elsif clk16x'event and clk16x = '1' then
if wrn1 = '0' and wrn2 = '1' then
tbre <= '0' ;
clk1x_enable <= '1' ;
elsif std_logic_vector(no_bits_sent) = "0010" then
tbre <= '1' ;
elsif std_logic_vector(no_bits_sent) = "1101" then
clk1x_enable <= '0' ;
end if ;
end if ;
end process ;

process (rst,wrn) --接收数据至 tbr
begin
if rst = '1' then
tbr <= (others => '0') ;
elsif wrn'event and wrn = '0' then
tbr <= din ;
end if ;
end process ;
```

```
process (rst,clk16x,clk1x_enable)
begin
if rst = '1' then
clkdiv <= "0000" ;
elsif clk16x'event and clk16x = '1' then
if clk1x_enable = '1' then
clkdiv <= clkdiv + "0001" ;
end if ;
end if ;
end process ;

clk1x <= clkdiv(3) ; --产生 clk1x 时钟

process (rst,clk1x,no_bits_sent,tbr)
begin
if rst = '1' then
sdo <= '1' ;
tsre <= '1' ;
tsr <= "00000000" ;
parity <= '1' ;
elsif clk1x'event and clk1x = '1' then
if std_logic_vector(no_bits_sent) = "0001" then
tsr <= tbr ; --发送缓冲器 tbr 数据进入发送移位寄存器 tsr
tsre <= '0' ; --发送移位寄存器空标志置“0”
elsif std_logic_vector(no_bits_sent) = "0010" then
sdo <= '0' ; --发送起始位信号“0”
elsif std_logic_vector(no_bits_sent) >= "0011" and
std_logic_vector(no_bits_sent) <= "1010" then
tsr <= tsr(6 downto 0) & '0' ; --从低位到高位进行移位输出至串行
输出端 sdo
sdo <= tsr(7) ;
parity <= parity xor tsr(7) ; --数据位中的 1 校验
```

```
end if ;
end if ;
end process ;

process (rst,clk1x,clk1x_enable) --产生发送字符长度和发送次序计
数器
begin
if rst = '1' or clk1x_enable = '0' then
no_bits_sent <= "0000" ;
elsif clk1x'event and clk1x = '1' then
if clk1x_enable = '1' then
no_bits_sent <= no_bits_sent + "0001" ;
end if ;
end if ;
end process ;
end ;
```

2.2.2 UART 接收器

串行数据帧和接收时钟是异步的，发送来的数据由逻辑 1 变为逻辑 0 可以视为一个数据帧的开始。接收器先要捕捉起始位，确定 rxd 输入由 1 到 0，逻辑 0 要 8 个 CLK16 时钟周期，才是正常的起始位，然后在每隔 16 个 CLK16 时钟周期采样接收数据，移位输入接收移位寄存器 rsr，最后输出数据 dout。还要输出一个数据接收标志信号标志数据接收完 。

接收器的端口信号如图 4 所示。

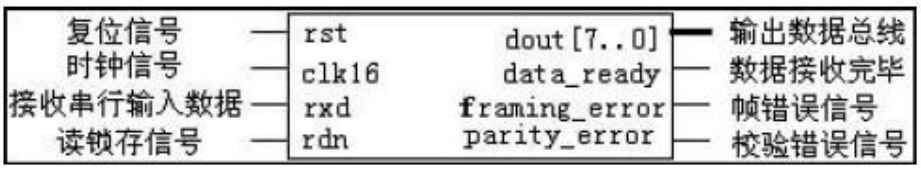


图 4

引入接收字符长度和接收次序计数器 no_bits_rcvd，实现设计的源程序如下。由于与发送器的一些说明相似，这里就不再重复。

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
entity rcvr is
```

```
port (rst,clk16x,rxd,rdn : in std_logic ;
dout : out std_logic_vector (7 downto 0) ;
data_ready : out std_logic ;
framing_error : out std_logic ;
parity_error : out std_logic ) ;
end rcvr ;
architecture v1 of rcvr is
signal rxd1 : std_logic ;
signal rxd2 : std_logic ;
signal clk1x_enable : std_logic ;
signal clkdiv : unsigned (3 downto 0) ;
signal rsr : unsigned (7 downto 0) ;
signal rbr : unsigned (7 downto 0) ;
signal no_bits_rcvd : unsigned (3 downto 0) ;
signal parity : std_logic ;
signal clk1x : std_logic ;
begin

process (rst,clk16x)
begin
if rst = '1' then
rxd1 <= '1' ;
rxd2 <= '1' ;
elsif clk16x'event and clk16x = '1' then
rxd2 <= rxd1 ;
rxd1 <= rxd ;
end if ;
end process ;

process (rst,clk16x,rxd1,rxd2,no_bits_rcvd)
begin
if rst = '1' or std_logic_vector(no_bits_rcvd) = "1100" then
clk1x_enable <= '0' ;
elsif clk16x'event and clk16x = '1' then
if rxd1 = '0' and rxd2 = '1' then
clk1x_enable <= '1' ;
end if ;
```

```
end if ;
end process ;

process (rst,clk16x,rdn,no_bits_rcvd)
begin
if rst = '1' or rdn = '0' then
data_ready <= '0' ;
elsif clk16x'event and clk16x = '1' then
if std_logic_vector(no_bits_rcvd) = "1100" then
data_ready <= '1' ;
end if ;
end if ;
end process ;

process (rst,clk16x,clk1x_enable)
begin
if rst = '1' then
clkdiv <= "0000" ;
elsif clk16x'event and clk16x = '1' then
if clk1x_enable = '1' then
clkdiv <= clkdiv + "0001" ;
end if ;
end if ;
end process ;

clk1x <= clkdiv(3) ;

process (clk1x,rst)
begin
if rst = '1' then
rsr <= "00000000" ;
rbr <= "00000000" ;
parity <= '1' ;
framing_error <= '0' ;
parity_error <= '0' ;
elsif clk1x'event and clk1x = '1' then
if std_logic_vector(no_bits_rcvd) >= "0001" and
```

```
std_logic_vector(no_bits_rcvd) < "1001" then  --- 数据帧数据
由接收串行数据端移位入接收移位寄存器
rsr(0) <= rxd2 ;
rsr(7 downto 1) <= rsr(6 downto 0) ;
parity <= parity xor rsr(0) ;
elsif std_logic_vector(no_bits_rcvd) = "1010" then
rbr <= rsr ; --接收移位寄存器数据进入接收缓冲器
elsif parity = '0' then
parity_error <= '1' ;
elsif std_logic_vector(no_bits_rcvd) = "1001" and rxd2 = '0'
then
framing_error <= '1' ;
end if ;
end if ;
end process ;

process (rst,clk1x,clk1x_enable,no_bits_rcvd)
begin
if rst = '1' or (std_logic_vector(no_bits_rcvd) = "1100" and
clk1x_enable = '0') then
no_bits_rcvd <= "0000" ;
elsif clk1x'event and clk1x = '1' then
if clk1x_enable = '1' then
no_bits_rcvd <= no_bits_rcvd + "0001" ;
end if ;
end if ;
end process ;
dout <= std_logic_vector(rbr) when rdn = '0' else "ZZZZZZZZ" ;
end ;
```

2.2.3 UART 设计总模块

将发送器和接收器模块组装起来，就能较容易地实现通用异步收发器总模块，而且硬件实现不需要很多资源，尤其能较灵活地嵌入到 FPGA/CPLD 的开发中。以下给出从模块源代码：

```
library ieee;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;
entity uart is
PORT (rst,clk16x,rxd,rdn,wrn : in std_logic;
din : in std_logic_vector(7 downto 0);
dout : out std_logic_vector(7 downto 0);
data_ready : out std_logic;
parity_error : out std_logic;
framing_error : out std_logic;
tbre : out std_logic;
tsre : out std_logic;
sdo : out std_logic);
end uart;
architecture v1 of uart is
component txmit
port (rst,clk16x,wrn : in std_logic;
din : in std_logic_vector(7 downto 0);
tbre,tsre,sdo: out std_logic);
end component ;
component rcvr
port (rst,clk16x,rxd,rdn : in std_logic;
data_ready, parity_error, framing_error : out std_logic;
dout : out std_logic_vector(7 downto 0));
end component ;
begin
u1 : txmit PORT MAP (rst => rst,clk16x => clk16x,din => din,tbre
=> tbre,tsre => tsre,
sdo => sdo);
u2 : rcvr PORT MAP (rst => rst,clk16x => clk16x,rxd => rxd,rdn
=> rdn,data_ready => data_ready,framing_error =>
framing_error,parity_error => parity_error,dout => dout) ;
end v1 ;
```

总模块 RTL 图如图 5

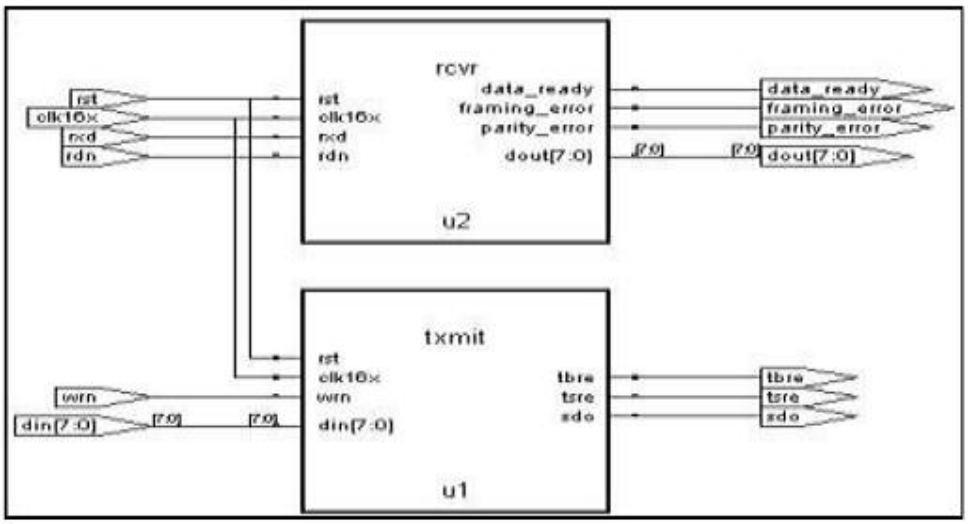


图 5

2.3 程序在 MAX+PLUS II 环境下的分析

在此设计中我所采用的编程目标芯片是 ALTERA 公司的 EPF10K10LC84-3.

2.3.1 波形仿真

波形仿真图如图 6 所示：

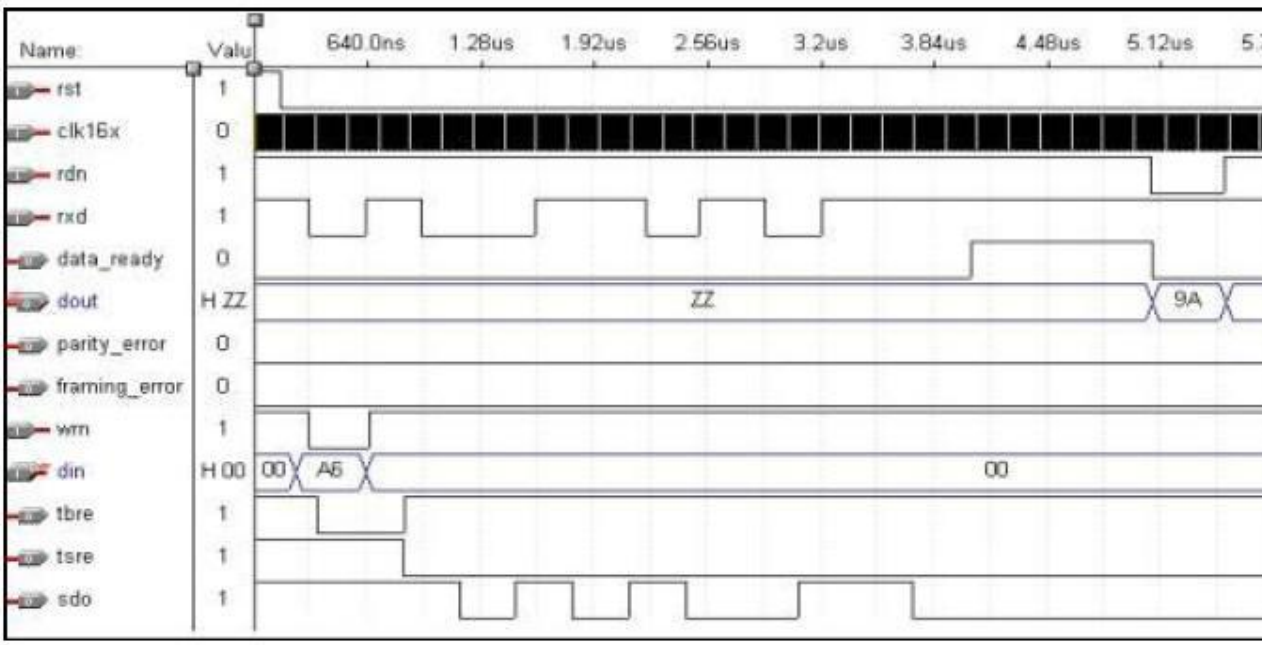


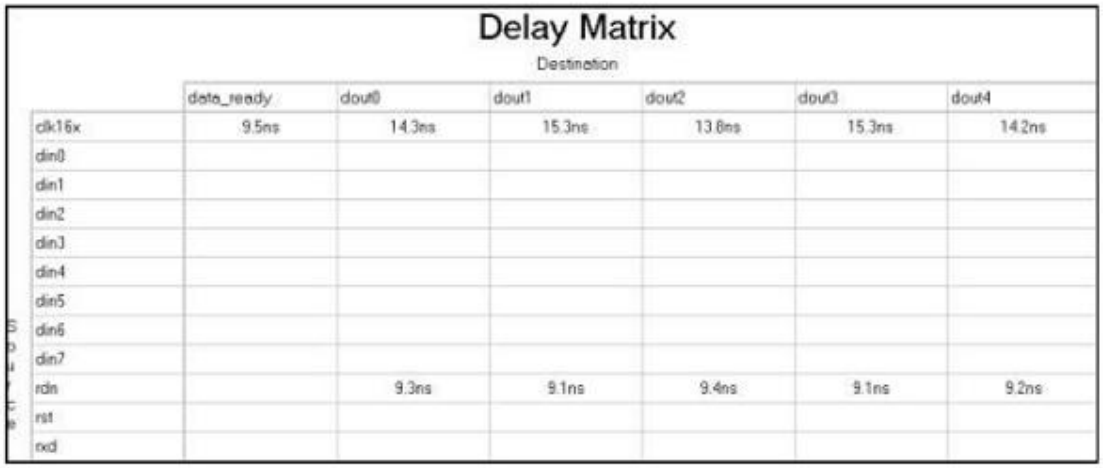
图 6

由于条件限制，数据给的太多，从图 6 上是看不出来的，所以，为了说明设计的正确性，只给出了一个数据。通过波形仿真图我们可以清楚的看到 UART 的工作原理。

2.3.2 时序分析

我对设计在没有优化的情况下做了分析，首要的目的是验证用 FPGA/CPLD 设计的正确性，次要的目的主要是与优化后的分析进行比较，以说明用 FPGA/CPLD 设计的优越性。

优化前的时序分析图如下：

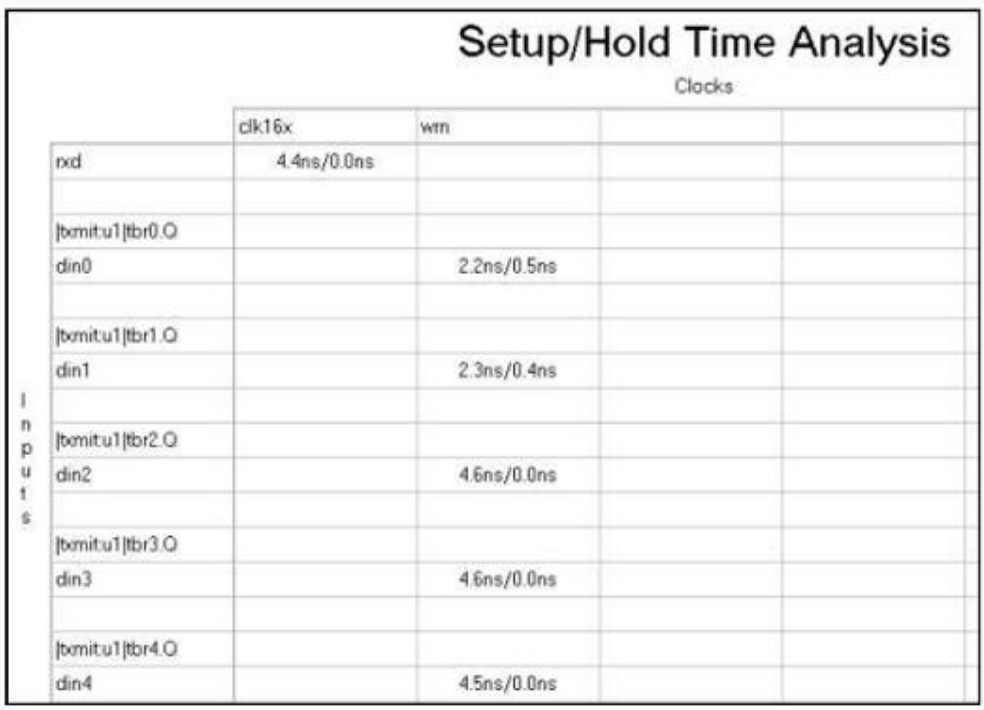


Delay Matrix

Destination

Source	data_ready	dout0	dout1	dout2	dout3	dout4
clk16x	9.5ns	14.3ns	15.3ns	13.8ns	15.3ns	14.2ns
din0						
din1						
din2						
din3						
din4						
din5						
din6						
din7						
rdn		9.3ns	9.1ns	9.4ns	9.1ns	9.2ns
rst						
rxd						

图 7、延时矩阵图



Setup/Hold Time Analysis

Clocks

Inputs	clk16x	wrn
rxd	4.4ns/0.0ns	
|transmit:u1|tbr0.Q		
din0		2.2ns/0.5ns
|transmit:u1|tbr1.Q		
din1		2.3ns/0.4ns
|transmit:u1|tbr2.Q		
din2		4.6ns/0.0ns
|transmit:u1|tbr3.Q		
din3		4.6ns/0.0ns
|transmit:u1|tbr4.Q		
din4		4.5ns/0.0ns

图 8、建立/保持时序分析图

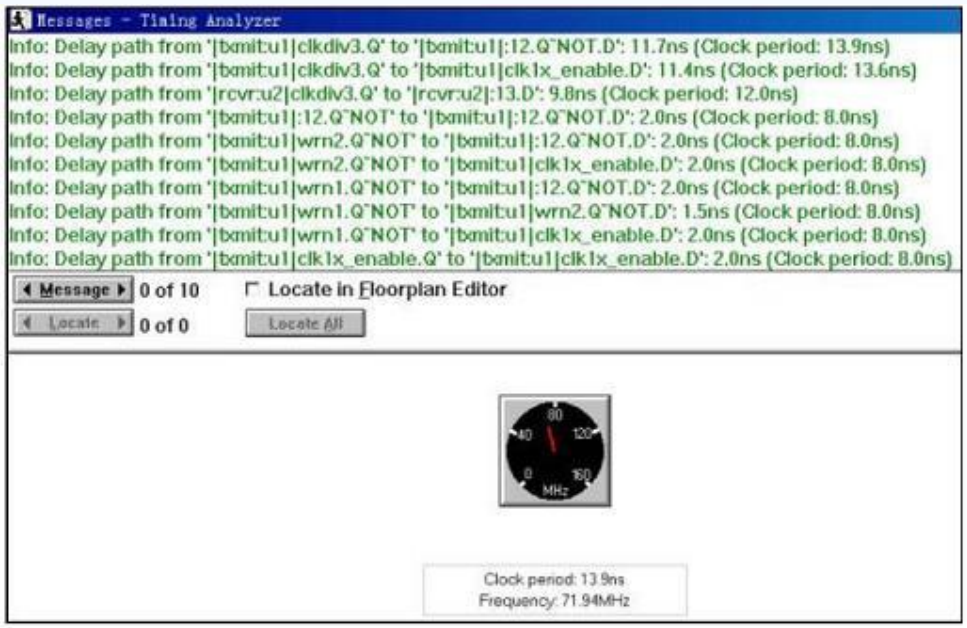


图 9、延时路径及寄存器时钟特性图

对本设计采用 MAX+PLUS II 优化设计中的专门针对其 ALTERA 器件的优化。在逻辑综合类型中选择快速类型（Fast）。这种类型是对大部分的器件特殊结构进行优化，是对性能要求较高的设计采用的综合类型。但是编译适配要比普通类型（Normal）慢许多。对于适配 FLEX/ACEX 等系列器件，综合器会自动使用 FLEX/ACEX 中的级链、进位链结构（对于 Normal 是忽略的）。
优化后的时序分析图如下：

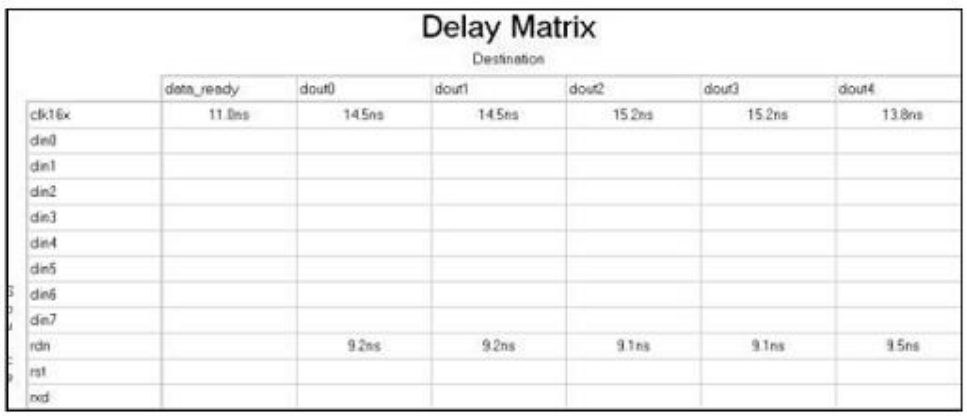


图 10、优化后的延时矩阵图

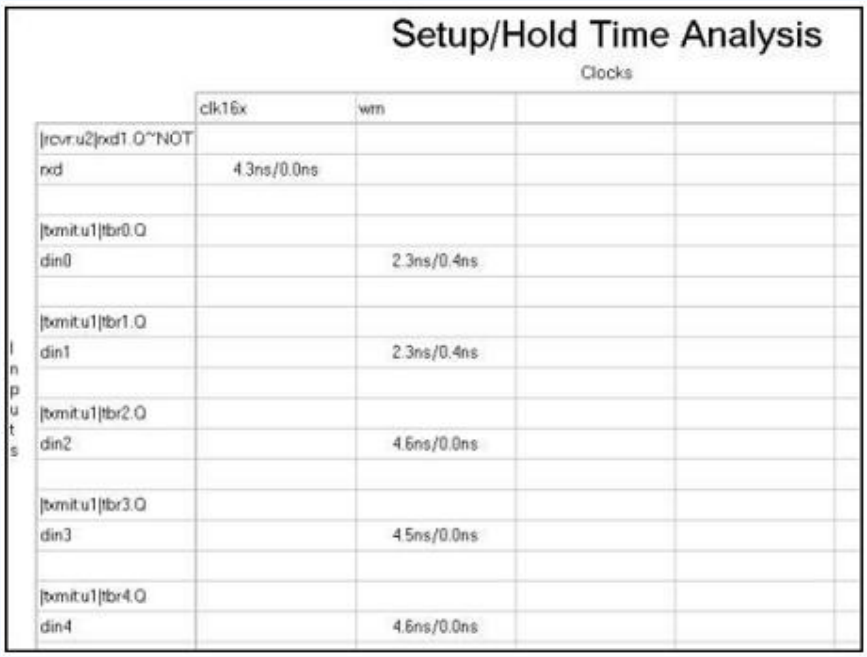


图 11、优化后的建立/保持时序分析图

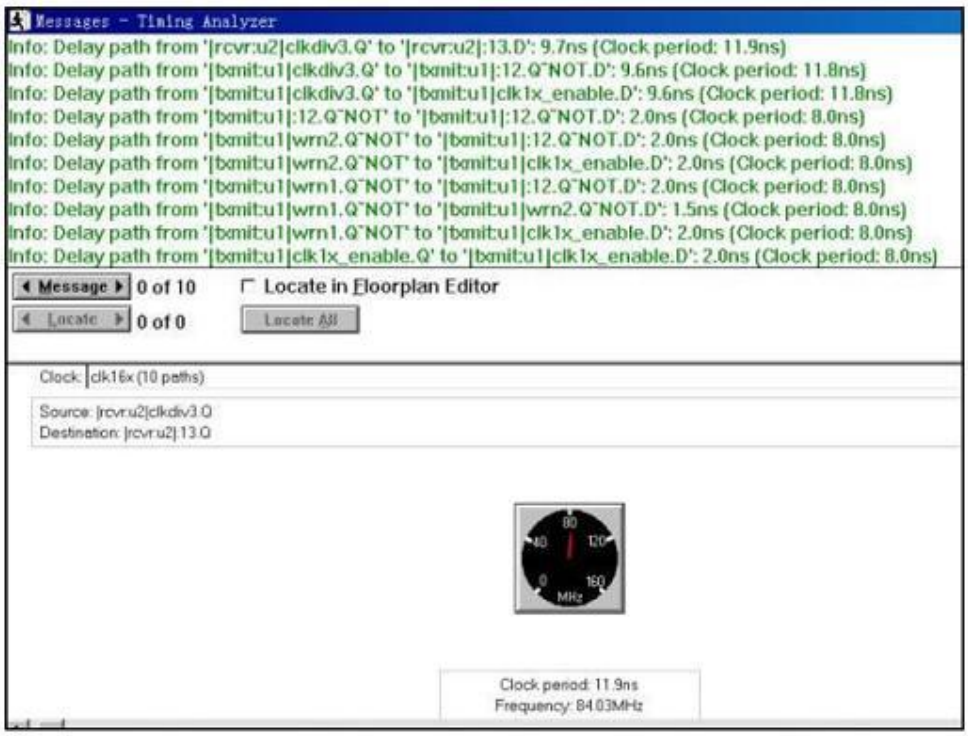


图 12、延时路径及寄存器时钟特性图

从时序分析我们可看到本设计基本满足设计要求。（上面的时序分析图中，某些图中只包含了部分数据）

对于优化前与优化后的数据比较来看，虽然在某些数据上优化后反而不如优化前的，这是正常的。因为优化不可能照顾到每一个信号。

2.3.3 资源利用

对于资源利用的情况，也分优化前和优化后进行比较。

优化前的资源分布图如下：

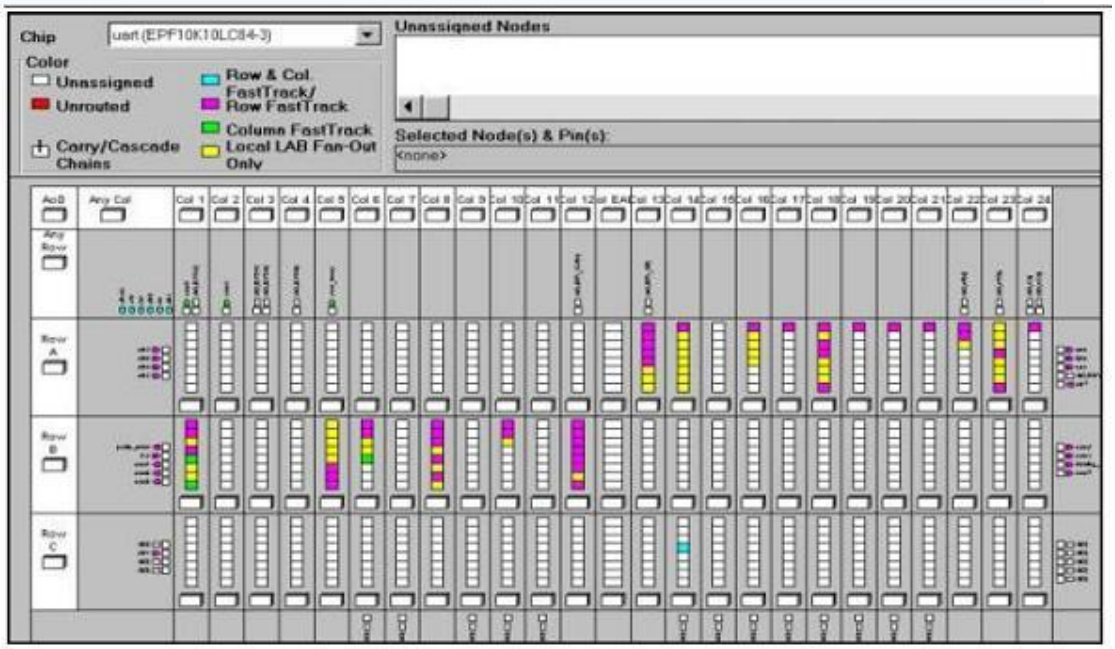


图 13、资源利用分布图

由于 FPGA/PLD 的阵列结构，如果在设计中，将某些关联模块在适配的时候集中分配在一块，其性能会大大提高。因此，对本设计进行打包（Clique）（又称分组）。

优化后的资源分布图如下

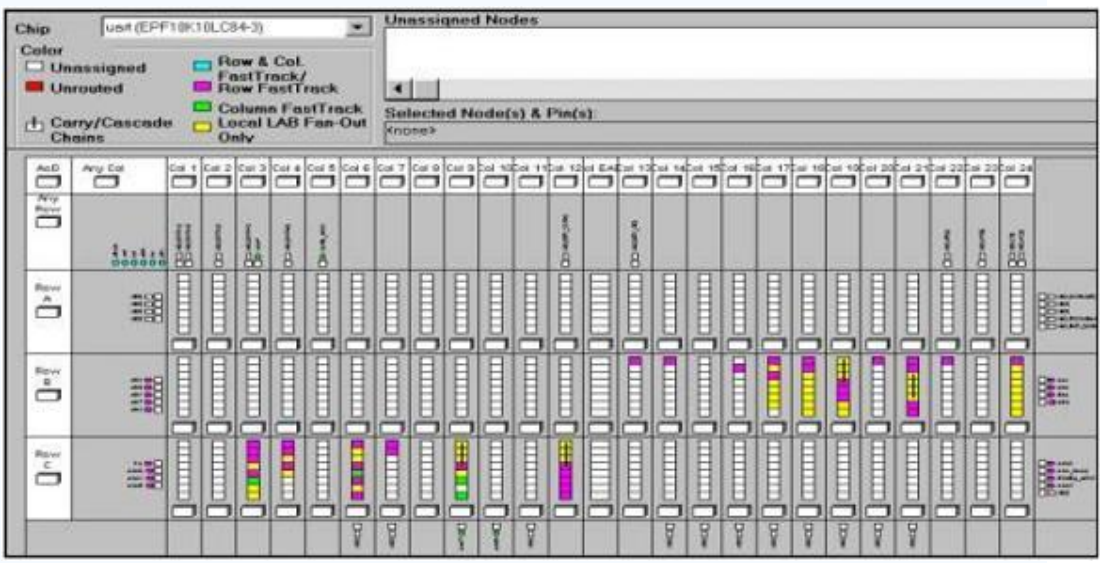


图 14、优化后资源利用分布图

从上面的比较中，我们很清楚的看到，优化后的资源利用率有了很大的提高。

3 总结

本设计由于采用了 VHDL 语言作为输入方式并结合 FPGA/CPLD，大大缩短了设计周期，提高了设计的可靠性、灵活性，使用户可根据自己的需求，方便、高效地设计出适合的串行接口电路。