Verilog设计初步

2.1 Verilog简介

2.2 Verilog模块的结构

2.3 Verilog语言的基本要素

2.4 Verilog基本描述语句

2.5 Verilog基本组合电路设计

2.6 Verilog基本时序电路设计

2.1 Verilog简介

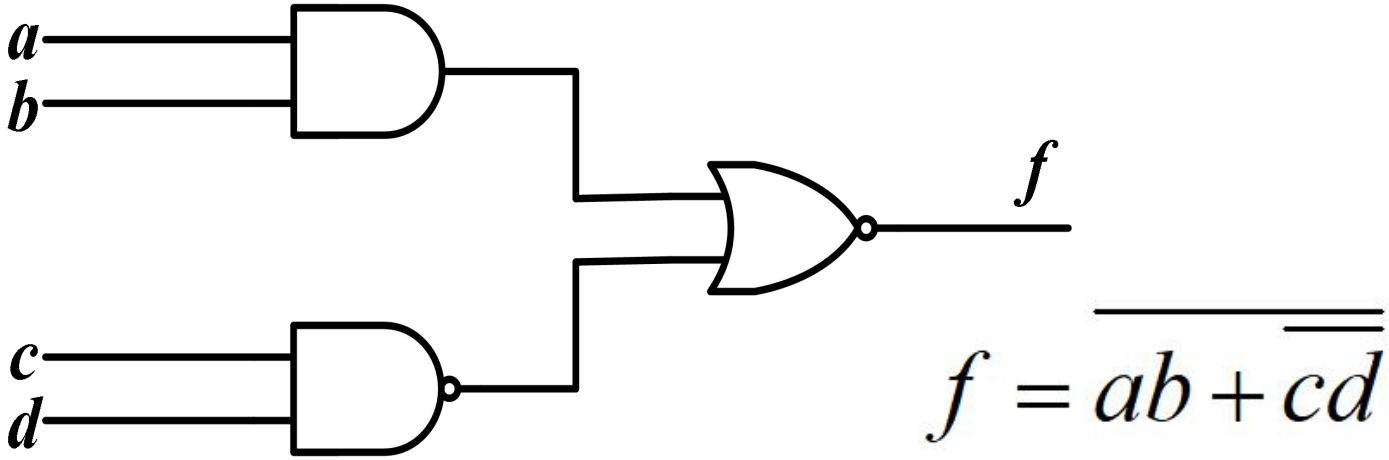
1.Verilog的历史

- Verilog语言是1983年由GDA（Gateway Design Automation）公司的Phil Moorby首创
- Verilog于1995年成为IEEE标准Verilog-1995
- IEEE标准Verilog-2001获得通过

2.Verilog语言的特点

- 既适于可综合的电路设计，也可胜任电路与系统的仿真。
- 能在多个层次上对所设计的系统加以描述，从开关级、门级、寄存器传输级（RTL）到行为级。
- 内置各种基本逻辑门，如and、or、xor等。
- Verilog的行为描述语句类似于C语言，如if-else、case等。
- 提供了算术、逻辑、位操作等运算符。
- 易学易用，功能强，可满足各个层次设计人员的需要。

2.2 Verilog模块的结构




```
module aoi(a,b,c,d,f);
input a,b,c,d;
output f;

assign f=~((a&b)|(~(c&d)));
endmodule
```

描述端口

描述逻辑功能

```
module aoi(a,b,c,d,f);
/* 模块名为aoi，端口列表a，b，c，d，f */
input a,b,c,d;                    //模块的输入端口为a，b，c，d
output f;                         //模块的输出端口为f

assign f=~((a&b)|(~(c&d)));       //逻辑功能描述
endmodule
```

- **Verilog程序是由模块构成的。每个模块的内容都嵌在module和endmodule两个关键字之间；每个模块实现特定的功能。**
- **模块名一般跟文件名一致。**
- **每个模块首先要进行端口定义，并说明输入和输出口（input、output或inout），然后对模块的功能进行定义。**
- **Verilog程序书写格式自由，一行可以写几个语句，一个语句也可以分多行写。**
- **除了endmodule等少数语句外，每个语句的最后必须有分号。**
- **可用 /*……*/ 和 //……对Verilog程序作注释。**

1．模块声明

模块声明包括模块名字，模块输入、输出端口列表。

模块定义格式如下：

```
module aoi(a,b,c,d,f);
  input a,b,c,d;
  output f;          //Verilog1995标准


module aoi(
  input a,b,c,d,
  output f
);                   //Verilog2001标准
```

2．端口（Port）定义

说明模块的输入输出端口类型

其格式为：

input 端口名1，端口名2，…… 端口名n；
//输入端口

output 端口名1，端口名2，…… 端口名n；
//输出端口

inout 端口名1，端口名2，…… 端口名n；
//输入输出端口

多位端口定义：

output [width-1:0] 端口名1，端口名2，…… 端口名n；

3．信号类型声明

说明端口或内部用到的信号类型

格式：

```
module sample( q, a, b, sel, clk, resetn );
    input     a, b, sel, clk, resetn;
    output    q;

    reg       q;   //  输出端口类型定义
    wire      c;   //  模块内信号线的定义

    ……
endmodule
```

output reg q;

- 常用的信号类型有线网型（wire）、寄存器型（reg）、整数型（integer）等。
- 在设计文件中，输出端口可以声明为reg型，但输入和双向端口信号不能声明为reg型。（在仿真文件中，输入必须设成reg型）
- 如果信号的数据类型没有定义，则综合器将其默认为wire型。

4. 逻辑功能定义

Verilog模块中最核心的部分是逻辑功能定义。

定义逻辑功能的几种基本方法:

- 数据流描述：用assign连续赋值语句定义
- 行为描述：用always过程块定义
- 结构描述：调用元件（元件例化）

（1）数据流描述（用assign连续赋值语句定义）

格式：

```
assign  变量名=表达式;
```

- 多用于组合逻辑电路的描述

（2）行为描述（用always过程块定义）

格式：

```
always @ (敏感信号表达式)
  begin
    //过程赋值
    //if-else，case语句
    //while，repeat，for循环语句
    //task，function调用
  end
```

- 既可以用来描述组合电路，也可以描述时序电路

（3）结构描述（调用元件（元件例化））

格式：

```
调用模块名 例化模块名 (
  端口列表port_list
);
```

eg.

```
div u0(
   .clk(clk0),
   .clk1k(clk1),
   .clk1hz(clk2)
);
```

- 调用已有的电路单元或模块
- 类似于在电路图输入方式下调入图符号来完成设计
- 侧重于电路的结构描述

Verilog 模块的模板

```
module  <顶层模块名> (<输入输出端口列表>);
  input   输入端口列表;                          //输入端口声明
  output  输出端口列表;                          //输出端口声明
/*定义数据，信号的类型，函数声明*/
  wire  信号名;
  reg  信号名;
//逻辑功能定义
assign <结果信号名>=<表达式>;        //使用assign语句定义逻辑功能
//用always块描述逻辑功能
always @ (<敏感信号表达式>)
   begin
      //过程赋值
       //if-else，case语句
       //while，repeat，for循环语句
       //task，function调用
    end
//调用其他模块
 <调用模块名module_name > <例化模块名> (<端口列表port_list >);
//门元件例化
 门元件关键字 <例化门元件名> (<端口列表port_list>);
endmodule
```

2.3 Verilog语言的基本要素

1. 词法约定

(1)关键字

- 必须用小写

如：module input wire always if else ……

(2)标识符

- 可以由一组字母、数字、下划线“_”及符号“$”组成；
- 第一个字符必须是字母或是“_”；
- 必须在英文输入法下输入，且区分大小写。

(3)标识符注释

- 单行注释：以“//”开始到本行结束；
- 多行注释：以“/*”开始，到“*/”结束。

2. 数据类型

(1)常量

格式：parameter　N=8；　　//整型常量定义

- 程序运行中其值不变的量
- 三种类型：整型常量、实型常量、字符型常量

(2)变量

a、线网型变量

格式：wire [3:0] a；

- 输出值紧随输入变化而变化；

b、寄存器型变量

格式：reg [3:0] q；

- 表示抽象的数据存储单元；
- 只能在always和initial块语句中赋值，若未被初始化，值为未知值X。

在设计文件中给reg型变量赋初值方法：

reg [3:0] q=4’b0000；

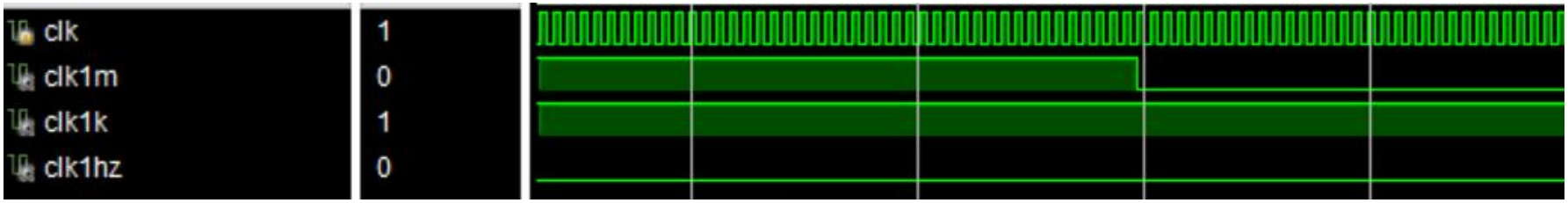
或 reg [3:0] q=4’d0；

注：若仿真波形中信号值为X（红色的线），一般是未给reg型变量赋初值导致


```
module div(
  input clk,
  output clk1m,clk1k,clk1hz,clk05
    );
  reg clk1m_r,clk1k_r=0,clk05_r=0,clk1hz_r=0;
```

解决办法：在设计文件中给相应reg型变量赋个初值

```
  reg clk1m_r=0,clk1k_r=0,clk05_r=0,clk1hz_r=0;
```



3. 常用运算符

(1)算术运算符 +加，-减，*乘，/除，%取模

(2)赋值运算符 =阻塞赋值，<= 非阻塞赋值

(3)关系运算符> ,<,>=,<=,==等于,!=不等于

(4)逻辑运算符 &&逻辑与, ||逻辑或, !逻辑非

(5)条件运算符 ？：(eg. max=(a>b)?a:b)

(6)位运算符 ~非, | 或, &与 ,^异或 ,^~同或

(7)移位运算符 <<左移 ,>>右移

(8)拼接运算符 {} (eg. c={a,b})

2.4 Verilog语言基本描述语句

1. 赋值语句

(1)连续赋值语句

格式：

assign 变量名=表达式;

- 给线网型（wire）变量赋值，不能给寄存器型（reg）变量赋值；
- 多用于组合逻辑电路的赋值；
- 只要右边表达式的值有变化，就给左边变量重新赋值。

(2)过程赋值语句

- 分为阻塞赋值和非阻塞赋值两类：

阻塞赋值：“=”

例如：

```
begin
    a=5;
    b=a;
    c=b;
end
```

顺序执行语句

结果：a=b=c=5

非阻塞赋值：“<=”

例如：

```
begin
    a<=5;
    b<=a;
    c<=b;
end
```

并行执行语句

结果：a=5，b=a原来的值

c=b原来的值

- 只能给寄存器型（reg）变量赋值；
- 用于always和initial语句内赋值；
- 既可以用来描述组合电路，也可以描述时序电路。

2．结构说明语句

(1)always语句

格式：

```
always @（敏感事件列表） 或 always @（*）
  begin
    //过程赋值
    //if-else，case语句
    //while，repeat，for循环语句
    //task，function调用
  end
```

- 既可以描述组合电路，也可以描述时序电路；
- always块中的输出必须是reg类型；
- 一个模块可以有多个always说明语句；
- 不能在多个always语句中给同一个信号赋值；

- 满足 always @后的敏感事件列表条件时执行；

- 敏感事件可以是电平触发，也可以是边沿触发；
（上升沿：posedge　　下降沿：negedge）
eg.　always @ (sel,a,b)　　always @ (posedge clk)

- 一个always语句中不可以同时包含电平触发和边沿触发；
eg.　always @ (rst or posedge clk)　错误！！！

- 敏感事件不可以是同一个时钟信号的两个边沿触发；
eg.　always @ (posedge clk or negedge clk)　错误！！！

- always语句在设计文件中不能没有敏感事件，如果任何情况都运行的话，就加@（*）；
在仿真文件中always后面可以没有敏感事件列表。

(2)initial语句

格式：

```
initial
    begin
        ……
    end
```

- 程序一开始就执行，只执行一次；
- 主要用于初始化和波形生成。

3. 条件语句

根据某个条件来确定是否执行其后的语句

(1)if-else语句

格式：

```
if（条件表达式1）
    语句1;
else if（条件表达式2 ）
    语句2;
……
else
    语句m;
```

(2)case语句

格式：

case（条件表达式）

分支1：语句1;

分支2：语句2;

……

default：语句m;

endcase

default：包含前面分支1、分支2……未指定的所有值

4. 循环语句

用来控制执行语句的执行次数

(1)for语句

格式：

```
for（i=0； i<=3;i++）
    语句;
```

(2)while语句

格式：

```
while（条件表达式）
    语句;
```

(3)repeat语句

格式：

repeat（循环次数）

语句;

(4)forever语句

格式：

forever

语句;

- 常用于产生周期性波形，不能独立写在程序中，必须写在initial块中。

2.5 Verilog基本组合电路设计

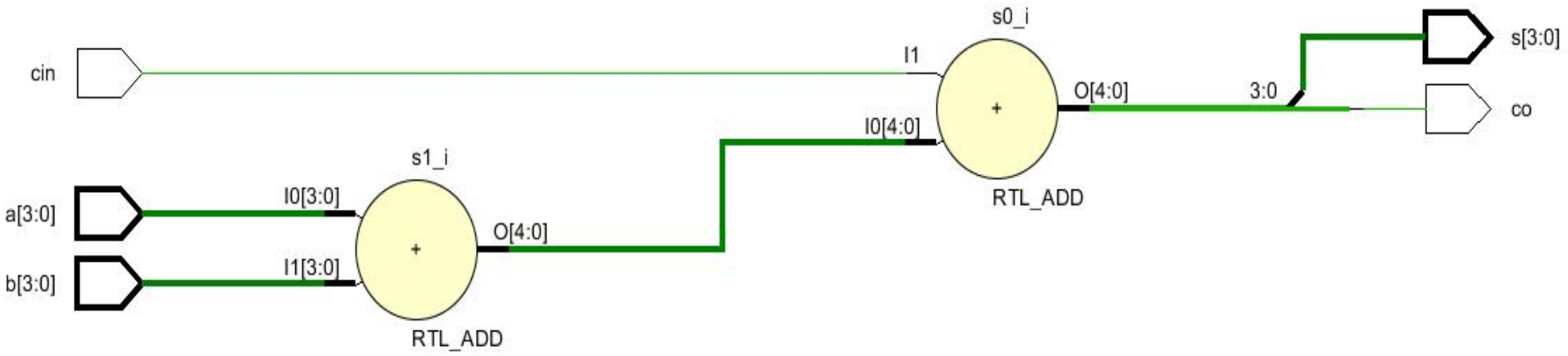
【例1】 三人表决电路的Verilog描述

```
module vote(a,b,c,f);                  //模块名与端口列表
  input a,b,c;                         //模块的输入端口
  output f;                            //模块的输出端口
  wire a,b,c,f;                         //定义信号的数据类型

 assign f=(a&b)|(a&c)|(b&c);     //逻辑功能描述
endmodule
```

【例2】4位二进制加法器的Verilog描述

```
module adder4(a,b,cin,s,co);
  input[3:0] a,b;
  input cin;
  output[3:0] s;
  output co;

 assign {co,s}=a+b+cin;          //逻辑功能定义
endmodule
```

综合 （RTL级）

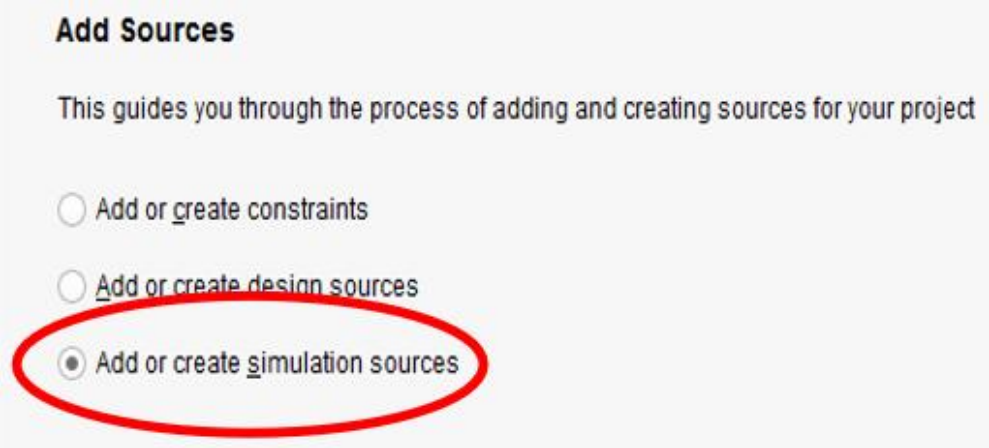


4位二进制加法器RTL级综合结果

模块的测试仿真

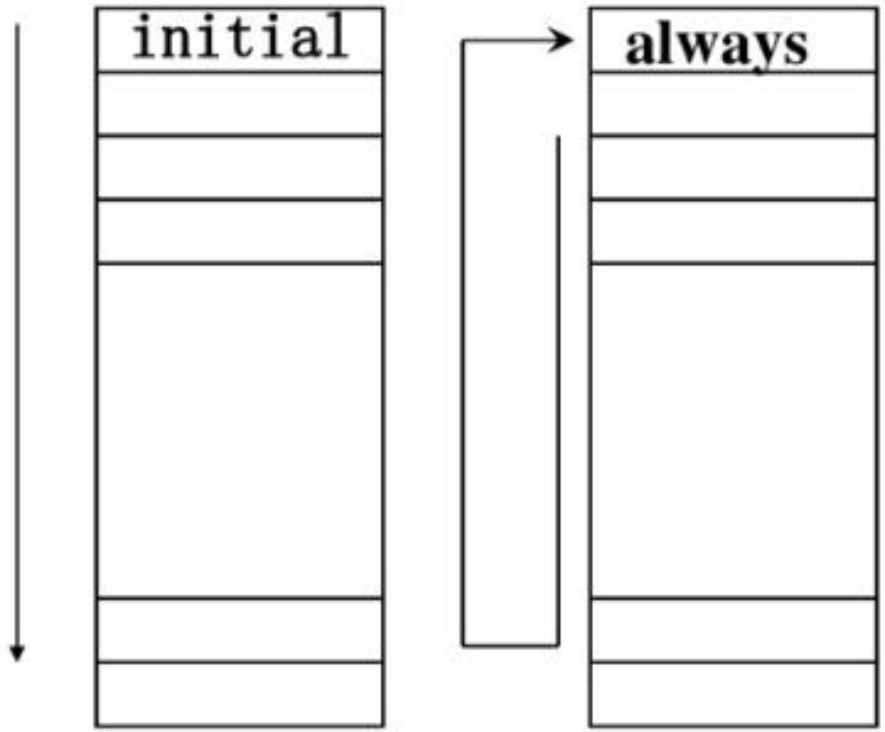
1.编写测试仿真文件

测试仿真文件也称Test Bench程序，用来定义待仿真模块中输入信号的值



2.运行仿真，产生仿真波形

测试仿真文件中的过程块



所有的过程块都在0时刻同时启动；它们是并行的，在模块中不分前后。

- initial块只执行一次。
- always块只要符合触发条件可以循环执行。

4位二进制加法器的Test Bench测试仿真文件

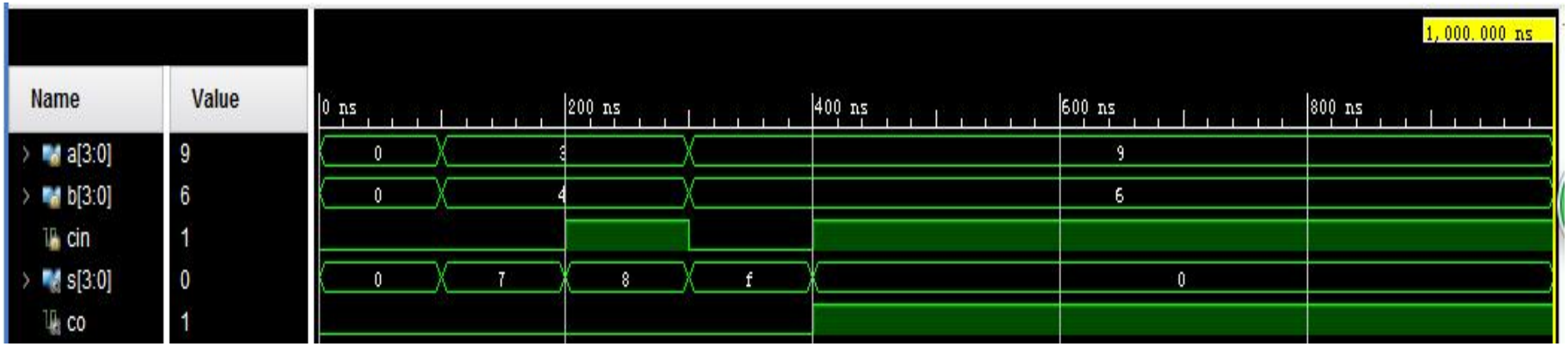
```
`timescale 1ns / 1ps
module adder4_simu;
  reg [3:0] a,b;
  reg cin;
  wire [3:0] s;
  wire co;
adder4 u0(            //例化被测试模块
    .a(a), .b(b), .cin(cin), .s(s), .co(co)
    );
initial
begin          //激励波形描述
    a=4'b0000; b=4'b0000; cin=1'b0;
    #100 a=4'b0011; b=4'b0100; cin=1'b0;
       //或   #100 a=4'd3; b=4'd4; cin=1'b0;
    #100 a=4'b0011; b=4'b0100; cin=1'b1;
    #100 a=4'b1001; b=4'b0110; cin=1'b0;
    #100 a=4'b1001; b=4'b0110; cin=1'b1;
end
endmodule
```

测试仿真模块说明：

- ▲ `timescale 编译器指令在模块说明外部出现；
- ▲ 测试模块只有模块名字，没有端口列表；
- ▲ 输入信号（激励信号）必须定义为reg型，以保持信号值；
- ▲ 输出信号（显示信号）必须定义为wire型；
- ▲ 一般用initial、always过程块定义激励信号波形。

仿真波形

```
initial
begin          //激励波形描述
   a=4'b0000; b=4'b0000; cin=1'b0;
   #100 a=4'b0011; b=4'b0100; cin=1'b0;
     //或   #100 a=4'd3; b=4'd4; cin=1'b0;
   #100 a=4'b0011; b=4'b0100; cin=1'b1;
   #100 a=4'b1001; b=4'b0110; cin=1'b0;
   #100 a=4'b1001; b=4'b0110; cin=1'b1;
end
```



2.6 Verilog基本时序电路设计

【例3】 基本D触发器的Verilog描述

```
module dff(clk,d,q);
  input clk,d;
  output reg q;

always @(posedge clk)
   begin
      q<=d;
   end
endmodule
```

【例4】带同步清0/同步置1（低电平有效）的D触发器

```
module dff_syn(clk,d,prn,clrn,q,qn);
  input clk,d,prn,clrn;
  output reg q,qn;
always @(posedge clk)
  begin
    if(!clrn)                          //同步清0，低电平有效
      begin
        q<=1'b0;
        qn<=1'b1;
      end
    else if(!prn)                      //同步置1，低电平有效
        begin
          q<=1'b1;
          qn<=1'b0;
        end
        else
        begin
          q<=d;
         qn<=~d;
        end
  end
endmodule
```

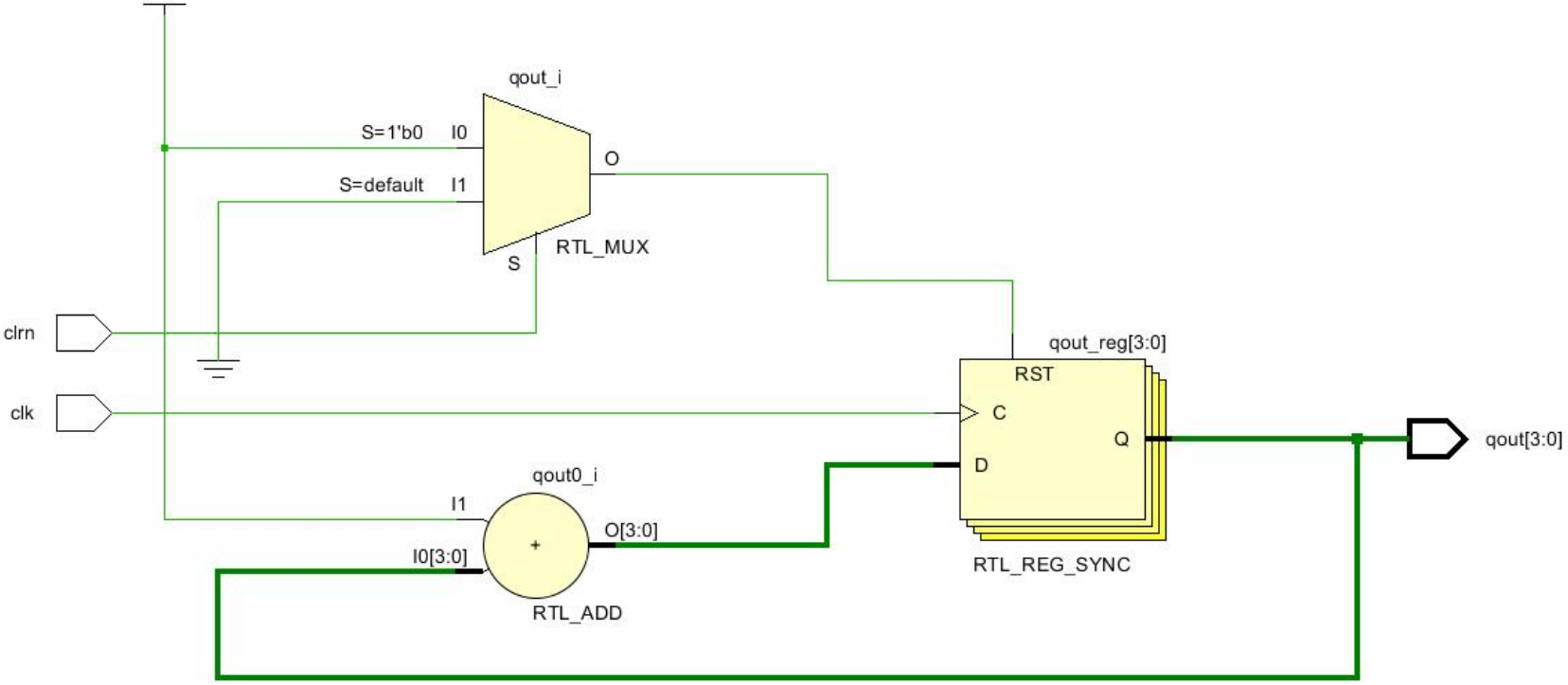
【例5】 带异步清0/异步置1（低电平有效）的D触发器

```
module dff_asyn(clk,d,prn,clrn,q,qn);
  input clk,d,prn,clrn;
  output reg q,qn;
always @(posedge clk or negedge prn or negedge clrn)
   begin
     if(!clrn)              //异步清0，低电平有效
       begin
         q<=1'b0;
         qn<=1'b1;
       end
    else if(!prn)          //异步置1，低电平有效
       begin
         q<=1'b1;
         qn<=1'b0;
       end
       else
       begin
         q<=d;
         qn<=~d;
       end
   end
endmodule
```

【例6】 4位二进制计数器

```
module count4(clk,clrn,qout);
  input clk,clrn;
  output reg[3:0] qout;

always @(posedge clk)
  begin
    if(!clrn)
      qout<=0;                  //同步复位
    else
      qout<=qout+1;             //计数
  end
endmodule
```

综合 （RTL级）



4位二进制加法计数器RTL级综合视图

4位二进制计数器的Test Bench测试仿真文件

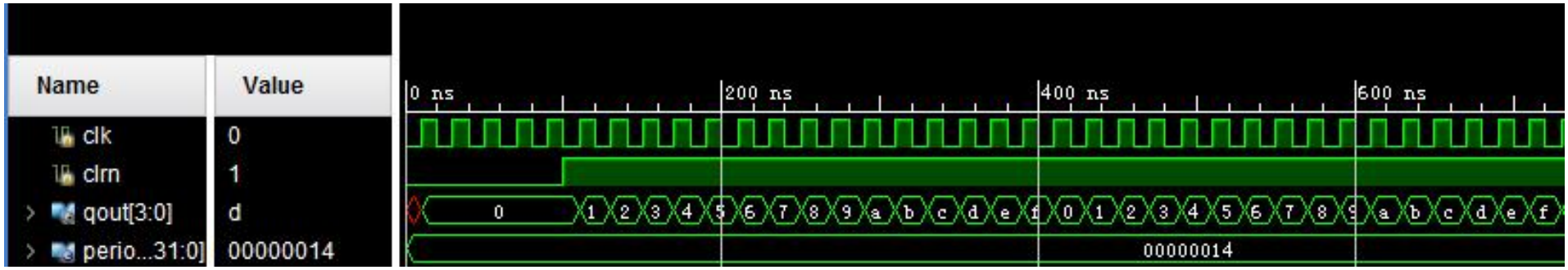
```
`timescale 1ns / 1ps
module count4_simu;
   reg clk,clrn;
   wire [3:0] qout;
count4 u0 (                    //例化被测试模块
  .clk(clk), .clrn(clrn), .qout(qout)
   );
initial
  begin
     clk=1'b0;
     clrn=1'b0;
     #100 clrn=1'b1;
  end
parameter period_clk=20;    //定义时钟周期为20ns
always
  begin
    #(period_clk/2) clk=~clk;
  end
endmodule
```

测试仿真模块说明：

- ▲ `timescale 编译器指令在模块说明外部出现；
- ▲ 测试模块只有模块名字，没有端口列表；
- ▲ 输入信号（激励信号）必须定义为reg型，以保持信号值；
- ▲ 输出信号（显示信号）必须定义为wire型；
- ▲ 用always过程块描述所需的时钟波形。

仿真波形

```
initial
  begin
     clk=1'b0;
     clrn=1'b0;
     #100 clrn=1'b1;
  end
parameter period_clk=20;    //定义时钟周期为20ns
always
  begin
    #(period_clk/2) clk=~clk;
  end
```



练习：

1.将自己班号的后4位数字显示在EGO1板左边的4个数码管上；

2.将自己学号的后4位数字显示在EGO1板右边的4个数码管上。

（注：数码管采用动态显示原理，且左边4个和右边4个独立工作）

（1）固定显示学号；

（2）开关控制输入任意学号；

（3）每次输入一位学号，输入后学号的其他位依次左移。

……