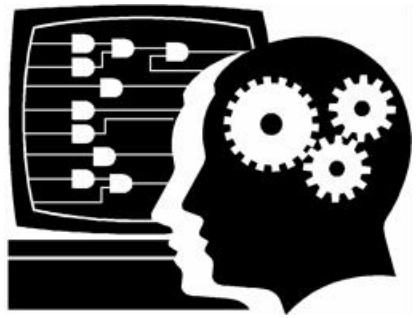
# 数字系统设计与Verilog HDL

## （第6版）



# 第10章 Verilog设计的优化

10.1 设计的可综合性

10.2 流水线设计（Pipeline Design）技术

10.3 资源共享（Resource Sharing）

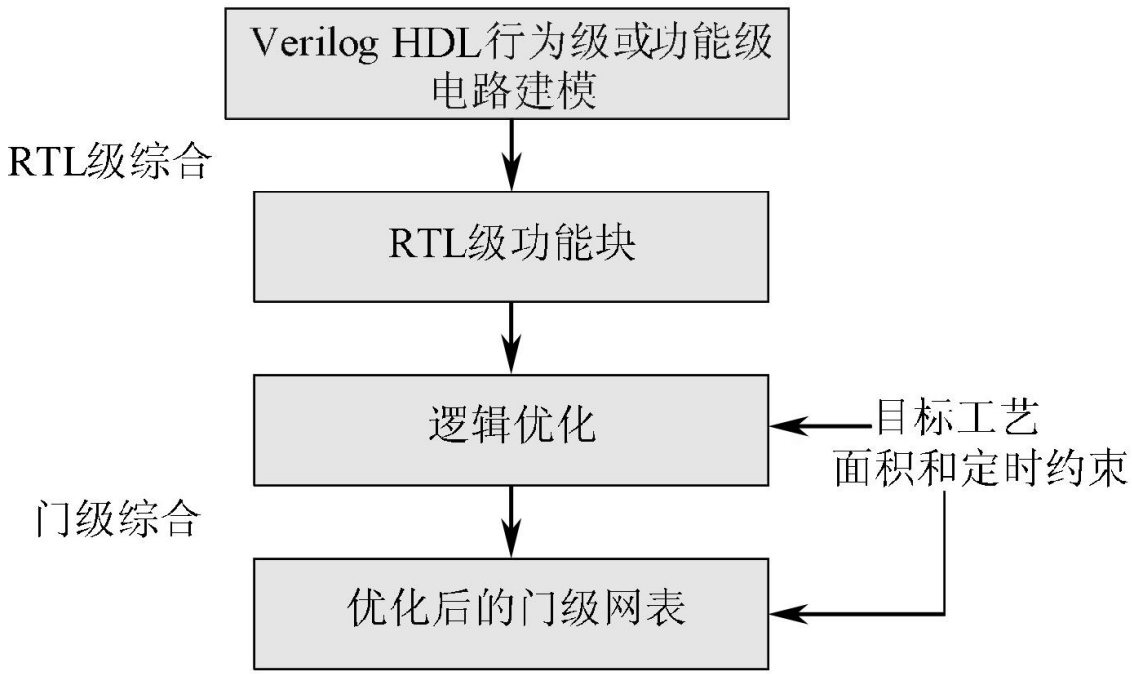
10.4 过程

10.5 阻塞赋值与非阻塞赋值

# 10.1 设计的可综合性

- 用FPGA/CPLD器件实现的设计中，综合就是将Verilog或VHDL语言描述的行为级或功能级电路模型转化为RTL级功能块或门级电路网表的过程

## 综合过程



# 可综合的设计中应注意

- 不使用初始化语句；不使用带有延时的描述；不使用循环次数不确定的循环语句，如forever，while等。
- 应尽量采用同步方式设计电路。除非是关键路径的设计，一般不采用调用门级元件来描述设计的方法，建议采用行为语句来完成设计。
- 用always过程块描述组合逻辑，应在敏感信号列表中列出块中出现的所有输入信号。

# 可综合的设计中应注意

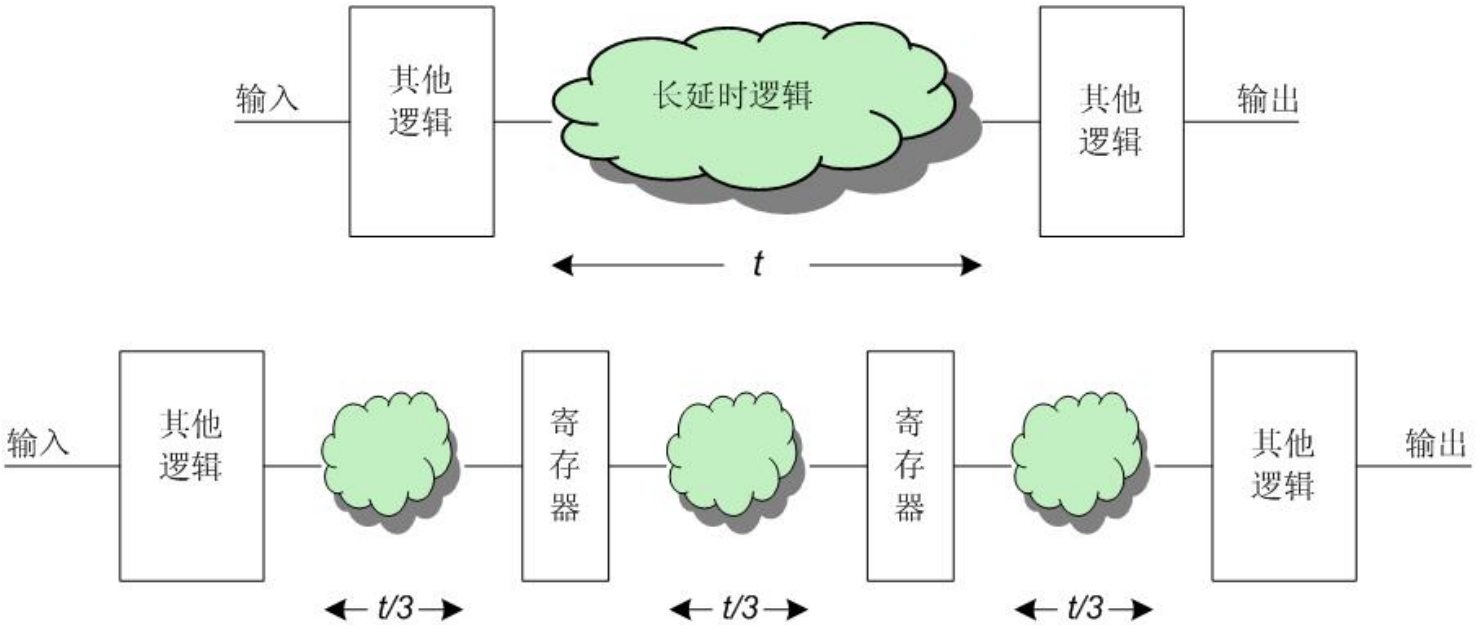
- 所有的内部寄存器都应该能够被复位，在使用FPGA实现设计时，应尽量使用器件的全局复位端作为系统总的复位，用器件的全局时钟端作为系统外部时钟输入端。
- 在Verilog模块中，任务（task）通常被综合成组合逻辑的形式；每个函数（function）在调用时通常也被综合为一个独立的组合电路模块。

# 10.2 流水线设计技术

- 流水线设计是经常用于提高所设计系统运行速度的一种有效的方法。为了保障数据的快速传输，必须使系统运行在尽可能高的频率上，但如果某些复杂逻辑功能的完成需要较长的延时，就会使系统难以运行在高的频率上，在这种情况下，可使用流水线技术，即在长延时的逻辑功能块中插入触发器，使复杂的逻辑操作分步完成，减小每个部分的延时，从而使系统的运行频率得以提高。流水线设计的代价是增加了寄存器逻辑，增加了芯片资源的耗用。

# 流水线操作的概念

- 如某个复杂逻辑功能的实现需较长的延时，可将其分解为几个（如3个）步骤来实现，每一步的延时变小，在各步间加入寄存器，以暂存中间结果，这样可大大提高整个系统的最高工作频率。




流水线操作的示意图

# 非流水线方式8位全加器

```
module adder8(cout,sum,ina,inb,cin,clk);
input[7:0] ina,inb; input cin,clk;
output[7:0] sum; output cout;
reg[7:0] tempa,tempb,sum; reg cout,tempc;
always @(posedge clk)
begin   tempa=ina;tempb=inb;tempc=cin; end
  //输入数据锁存
always @(posedge clk)
begin    {cout,sum}=tempa+tempb+tempc; end
endmodule
```

# 两级流水实现的8位加法器

```
module adder_pipe2(cout,sum,ina,inb,cin,clk);
input[7:0] ina,inb; input cin,clk;
output reg[7:0] sum;
output reg cout;
reg[3:0] tempa,tempb,firsts;
reg firstc;
always @(posedge clk)
begin  {firstc,firsts}=ina[3:0]+inb[3:0]+cin;
tempa=ina[7:4];  tempb=inb[7:4];
end
always @(posedge clk)
begin  {cout,sum[7:4]}=tempa+tempb+firstc;
sum[3:0]=firsts;
end
endmodule
```

# 四级流水线实现的8位加法器

```
module  pipeline(cout,sum,ina,inb,cin,clk);
output[7:0]  sum;output  cout;
input[7:0] ina,inb;input  cin,clk; reg[7:0] tempa,tempb,sum;
reg  tempci,firstco,secondco,thirdco, cout;
reg[1:0]  firsts, thirda,thirdb;
reg[3:0]  seconda, secondb, seconds; reg[5:0]  firsta, firstb, thirds;

always @(posedge clk)
begin tempa=ina;  tempb=inb;  tempci=cin; end //输入数据缓存
always @(posedge clk)
begin {firstco,firsts}=tempa[1:0]+tempb[1:0]+tempci;  //第一级加（低2位）
firsta=tempa[7:2]; firstb=tempb[7:2];          //未参加计算的数据缓存
end
always @(posedge clk)
begin {secondco,seconds}={firsta[1:0]+firstb[1:0]+firstco,firsts};
seconda=firsta[5:2]; secondb=firstb[5:2];     //数据缓存
end
always @(posedge clk)
begin {thirdco,thirds}={seconda[1:0]+secondb[1:0]+secondco,seconds};
thirda=seconda[3:2];thirdb=secondb[3:2];      //数据缓存
end
always @(posedge clk)
begin
{cout,sum}={thirda[1:0]+thirdb[1:0]+thirdco,thirds}; //第四级加
end  endmodule
```

# 设计综合到不同器件的最高工作频率

| 器　件 | 非流水线设计/MHz | 2级流水线设计/MHz | 4级流水线设计/MHz |
|---|---|---|---|
| EP1K10TC100-3（FPGA） | 138.89 | 158.73 | 166.67 |
| EPM3064ALC44-4（CPLD） | 105.26 | 149.25 | 158.73 |

# 10.3 资源共享（Resource Sharing）

```
module resource1(sum,a,b,c,d,sel);
parameter SIZE=4; input sel;
input[SIZE-1:0] a,b,c,d;
output reg[SIZE:0] sum;
always @(*)                    //使用通配符
begin if(sel)       sum=a+b;
else          sum=c+d;
end   endmodule
```

**2个加法器和1个选择器的实现方式**

# 资源共享（Resource Sharing）

```
module resource2(sum,a,b,c,d,sel);
parameter SIZE=4; input sel; input[SIZE-1:0] a,b,c,d;
output reg[SIZE:0] sum; reg[SIZE-1:0] atemp,btemp;
always @(*)                    //使用通配符
begin if(sel) begin atemp=a;btemp=b;end
else  begin atemp=c;btemp=d;end
sum=atemp+btemp;
end
endmodule
```

**2个选择器和1个加法器的实现方式**

# 资源共享（Resource Sharing）

| | |
|---|---|
| Top-level Entity Name | resource1 |
| Family | ACEX1K |
| Device | EP1K10TC100-3 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 17 / 576 ( 3 % ) |
| Total pins | 22 / 66 ( 33 % ) |
| Total memory bits | 0 / 12,288 ( 0 % ) |
| Total PLLs | 0 |

| | |
|---|---|
| Top-level Entity Name | resource2 |
| Family | ACEX1K |
| Device | EP1K10TC100-3 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Total logic elements | 13 / 576 ( 2 % ) |
| Total pins | 22 / 66 ( 33 % ) |
| Total memory bits | 0 / 12,288 ( 0 % ) |
| Total PLLs | 0 |

器件资源的消耗对比

# 资源共享（Resource Sharing）

器件资源消耗的比较

| 设计规模（size） | 方式 1（resource1） | 方式 2（resource2） |
| --- | --- | --- |
| 4 位 | 17 个 LE | 13 个 LE |
| 8 位 | 33 个 LE | 25 个 LE |
| 10 位 | 41 个 LE | 31 个 LE |

## 结　论

方式一需要2个加法器，而方式二通过增加一个MUX，共享一个加法器，由于加法器耗用的资源比MUX更多，因此方式二更节省资源。所以在电路设计中，应尽可能使硬件代价高的功能模块资源共享，从而降低整个系统的成本。

# 10.4 过程

- 在Verilog语言中，过程语句包括always和initial。always过程反复执行其中的块语句，而initial过程中的语句块只执行一次。always过程可综合，initial语句只能用于仿真。
- always过程语句与VHDL语言的进程语句Process非常相像，它既可以用来描述时序电路，也可以用来描述组合电路。一个Verilog模块中的不同always过程语句是并行运行的。
- assign赋值语句、实例元件的调用也都是并行运行的，我们可以这样理解：这些语句最终都综合或翻译成为具体的电路结构，而这些电路结构是同时在运行或动作的。

# 在进行数字系统设计的时候应注意

- 将组合逻辑实现的电路和用时序逻辑实现的电路应尽量分配到不同的always过程中。
- 一个always过程中只允许描述对应于一个时钟信号的同步时序逻辑。
- always过程必须由敏感信号的变化来启动，因此应精心选择进程敏感表达式中的敏感变量。
- 多个always过程间可通过信号线进行通信和协调。

# 10.5 阻塞赋值与非阻塞赋值

- 在可综合的硬件设计中，使用阻塞和非阻塞赋值语句时，应注意以下原则

（1）当用“always”块来描述组合逻辑时，既可以用阻塞赋值，也可以采用非阻塞赋值，应尽量使用阻塞赋值。

（2）对时序逻辑描述和建模，使用非阻塞赋值方式。

（3）为锁存器（Latch）建模，应使用非阻塞赋值。

（4）若在同一个“always”过程块中既为组合逻辑建模，又为时序逻辑建模，最好使用非阻塞赋值方式。

# 10.5 阻塞赋值与非阻塞赋值

- 在可综合的硬件设计中，使用阻塞和非阻塞赋值语句时，应注意以下原则

（5）在一个“always”过程块中，最好不要混合使用阻塞赋值和非阻塞赋值，虽然同时使用这两种赋值方式在综合时并不一定会出错，但对同一个变量不能既进行阻塞赋值，又进行非阻塞赋值，这样在综合时会报错。

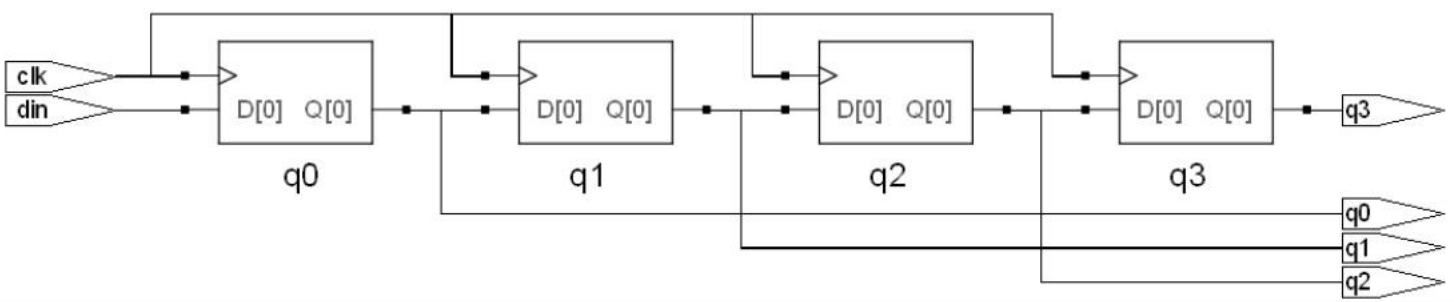
（6）不能在两个个或两个以上的“always”过程块中对同一个变量赋值，这样会引发冲突，在综合时会报错。

（7）仿真时使用$strobe显示非阻塞赋值的变量。

# 时序逻辑建模应尽量使用非阻塞赋值方式

【例10.9】　阻塞赋值方式描述的移位寄存器1

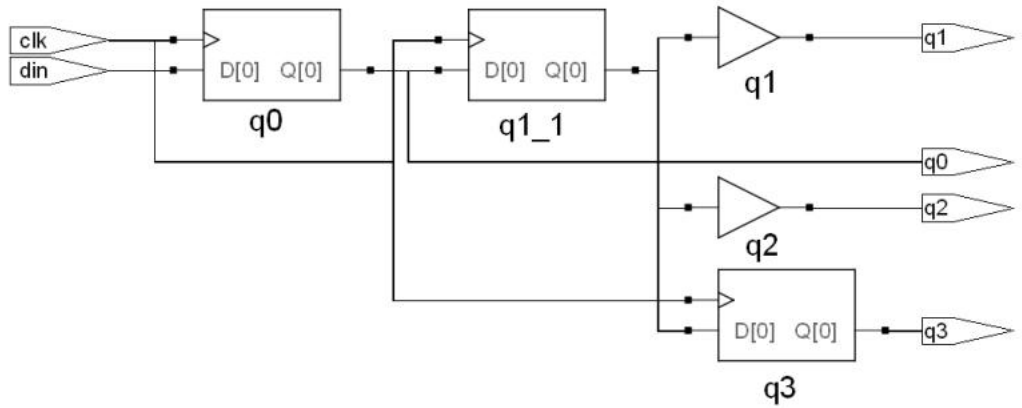
```
module block1(q0,q1,q2,q3,din,clk);
input clk,din; output reg q0,q1,q2,q3;
always @(posedge clk)
begin          q3=q2;          //注意赋值语句的顺序
               q2=q1;
       q1=q0;
       q0=din;
end  endmodule
```




例10.9的RTL综合结果

【例10.10】　阻塞赋值方式描述的移位寄存器2。

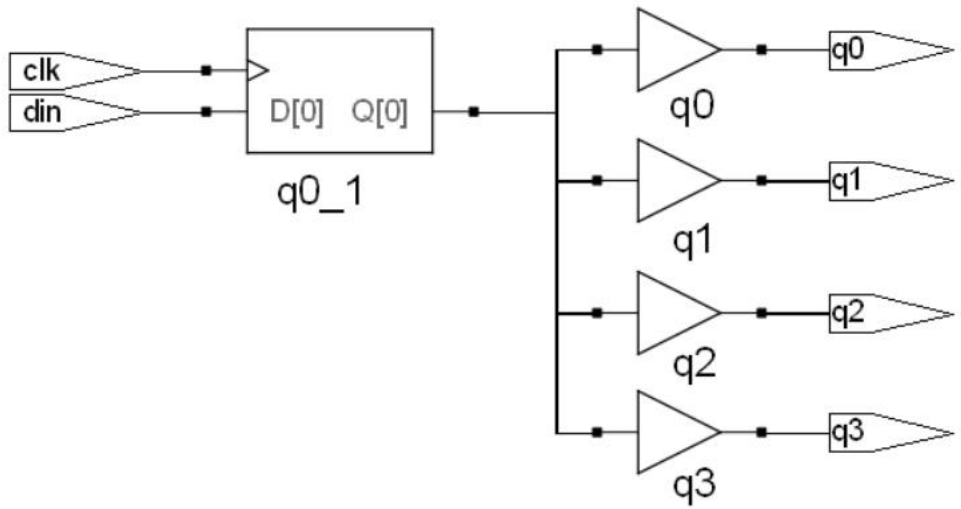
```
module block2(q0,q1,q2,q3,din,clk);
input clk,din; output reg q0,q1,q2,q3;
always @(posedge clk)
begin        q3=q2;
       q1=q0;
             //该句与下句的顺序与例10.9颠倒
       q2=q1;
       q0=din;
end  endmodule
```




**例10.10的RTL综合结果**

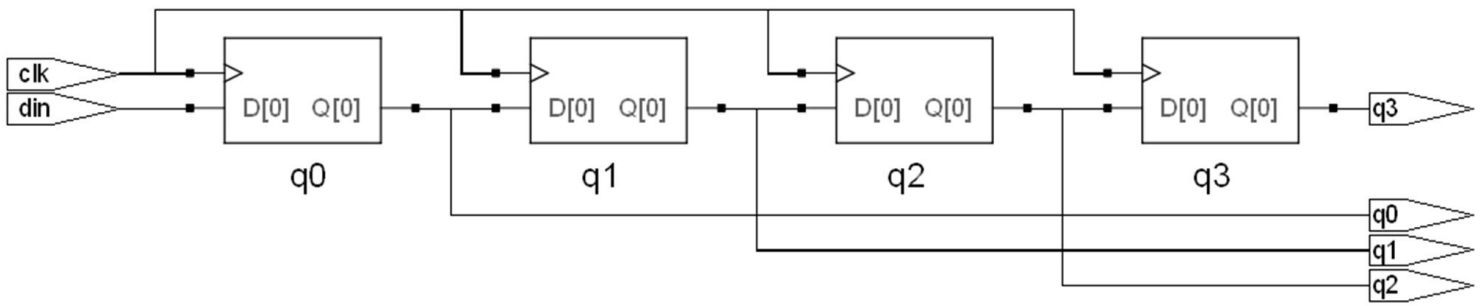
【例10.11】　阻塞赋值方式描述的移位寄存器3。

```
module block3(q0,q1,q2,q3,din,clk);
input clk,din; output reg q0,q1,q2,q3;
always @(posedge clk)
begin        q0=din;
   //4条赋值语句的顺序与例10.9中完全颠倒
       q1=q0;
       q2=q1;
       q3=q2;
end  endmodule
```



【例10.12】　非阻塞赋值方式描述的移位寄存器。

```
module block4(q0,q1,q2,q3,din,clk);
input clk,din; output reg q0,q1,q2,q3;
always @(posedge clk)
begin  q3<=q2;
       q1<=q0;
       q2<=q1;
       q0<=din;
end  endmodule
```



# 结　论

对于阻塞赋值来说，赋值语句的顺序对最后的综合结果有着直接的影响。而如果采用非阻塞赋值方式来描述的话，则可以不考虑赋值语句的排列顺序，只需将其连接关系描述清楚即可。

# 习 题 10

10.1 阻塞赋值与非阻塞赋值有什么本质的区别，在使用中应注意哪些方面，结合自己的设计实践进行总结。

10.2 流水线设计技术为什么能提高数字系统的工作频率？

10.3 设计一个加法器，实现sum=a0+a1+a2+a3，a0、a1、a2、a3宽度都是8位。如用下面两种方法实现，哪种方法更好一些。

（1）sum=((a0+a1)+a2)+a3

（2）sum=(a0+a1)+(a2+a3)

10.4 用流水线技术对上题中的sum=((a0+a1)+a2)+a3的实现方式进行优化，对比最高工作频率。

10.5 在FPGA设计开发中，还有哪些方法可提高设计性能？

# 实验与设计

## 10-1 小数分频

实验要求：用Verilog HDL实现小（分）数分频，假定源时钟为60MHz，先从60MHz经小数分频得到50.4MHz的时钟信号，进而从50.4MHz时钟分频得到10KHz、20KHz、…、100KH等10个时钟频率。

# 10-2 如何在FPGA设计中消除毛刺

实验要求：探讨在FPGA设计中消除毛刺的方法并进行仿真。

信号在FPGA器件内部通过连线和逻辑门时，都有一定的延时。因此多路信号的电平值发生变化时，在信号变化的瞬间，组合逻辑的输出有先后顺序，往往会出现一些不正确的“毛刺”（Glitch），称为“冒险”（Hazard）现象。

这些毛刺在电路板的设计中由于PCB走线时，存在分布电感和分布电容，所以许多毛刺能够被自然滤除，而在PLD内部没有分布电感和电容，这些毛刺将被完整地保留并向下一级传递，所以在FPGA设计中，如何消除毛刺就变得很重要。

# 10-2 如何在FPGA设计中消除毛刺

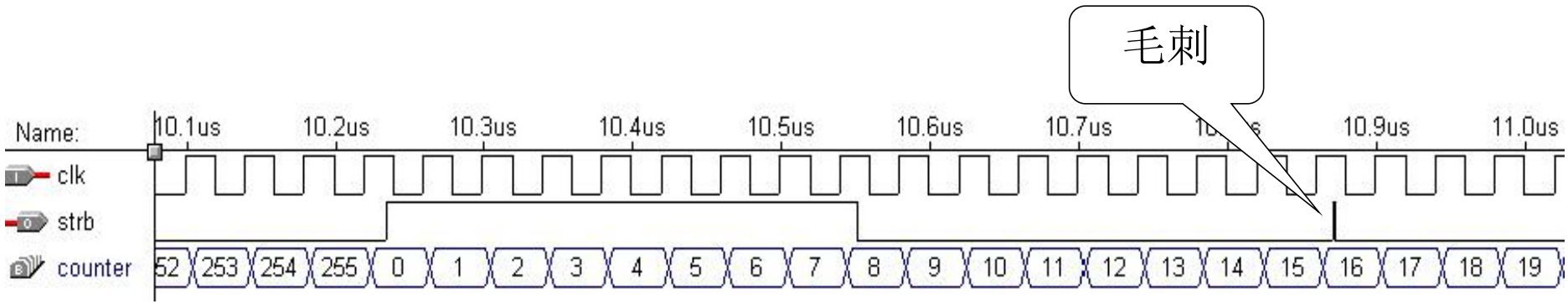
可通过改变设计，破坏毛刺产生的条件，来减少毛刺的发生。

例如，在数字电路设计中，常常采用格雷码计数器取代普通的二进制计数器。

还可根据D触发器的D输入端对毛刺不敏感的特点而消除毛刺，举例说明

【例10.23】 长帧同步时钟的产生

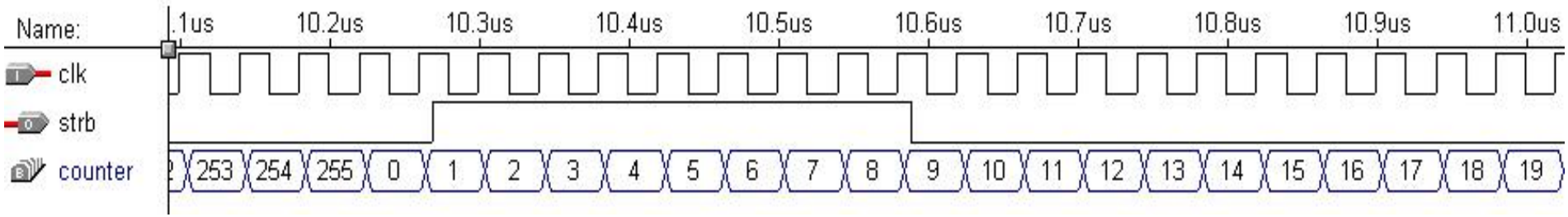
```
module longframe1(clk,strb);
parameter DELAY=8; input clk;
output reg strb; reg[7:0] counter;
always@(posedge clk)
begin if(counter==255) counter<=0;
else counter<=counter+1;  end
always@(counter)
begin if(counter<=(DELAY-1)) strb<=1; else strb<=0;
  end
endmodule
```


毛刺
Name:
10.1us
10.2us
10.3us
10.4us
10.5us
10.6us
10.7us
10.9us
11.0us
clk
strb
counter
252
253
254
255
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

# 时序仿真输出波形

## 引入了D触发器的长帧同步时钟的产生

```
module longframe2(clk,strb);
parameter DELAY=8; input clk;
output strb; reg[7:0] counter; reg temp,strb;
always@(posedge clk)
begin if(counter==255) counter<=0;
else counter<=counter+1;
end
always@(posedge clk)
begin strb<=temp;end    //引入触发器
always@(counter)
begin if(counter<=(DELAY-1)) temp<=1; else temp<=0; end
endmodule
```




# 消除毛刺后的时序仿真输出波形