可靠设计与高性能设计

提纲

- 可靠设计
- 同步设计
- 异步设计
- 提升设计性能

提纲

- 可靠设计
- 同步设计
- 异步设计
- 提升设计性能

RTL描述

1.“X”状态的传递

2.敏感信号的问题

3.条件判断语句

4.多驱动与总线复用

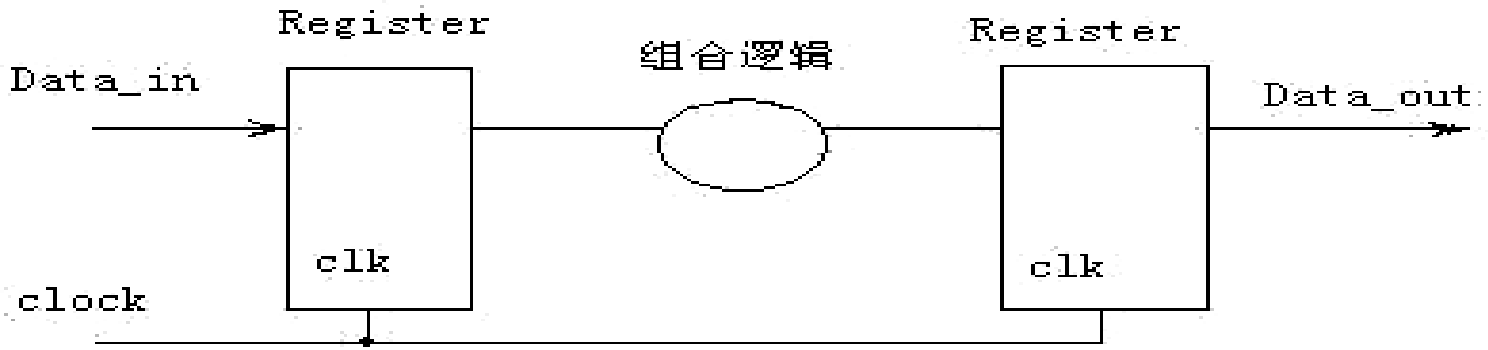
5.毛刺的消除

VHDL结构体描述

- 行为级描述
  - 可进行系统仿真，少数可用于综合。
- RTL(Register Transfer Level)级描述
  - 寄存器传输级描述，也称为数据流描述。可被综合器综合成门级电路。
- 结构级描述
  - 多用于顶层模块的连接和门级网表的描述。
  - 门级网表：用结构级直接描述硬件电路底层门的连接关系．一般为综合工具生成

RTL级描述

- 面向可综合的设计，处于抽象的行为级描述与门级描述之间的层次。
- 其对应的硬件层次为寄存器云图



RTL描述注意事项

1. “X”状态的传递
2. 敏感信号的问题
3. 条件判断语句
4. 多驱动与总线复用
5. 毛刺的消除

1. “X”状态的传递(1)

- 不确定态“X”经常在门级仿真中出现，为什么？
- 在RTL级的描述时，要做好“X”状态的处理。

1. “X”状态的传递(2)

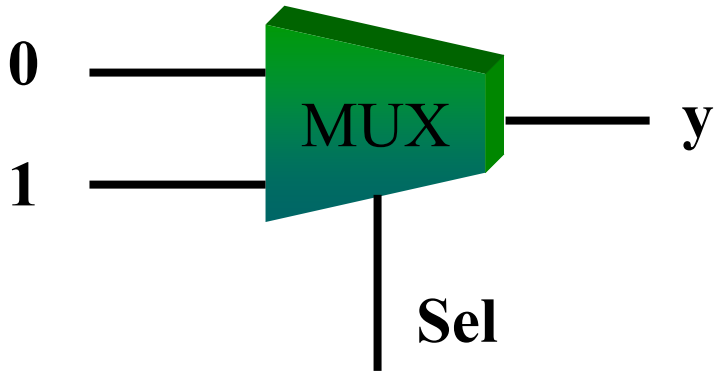
例子：

- if(sel = ‘1’) then
- y <= ‘0’;
- else
- y <= ‘1`;
- end if;

•从门级的观点看，该分支隐含的条件为：**If (sel = ‘0’)**。

•在**RTL**级仿真时，当**sel = ‘X’**时，**y**为**?**

•在综合后的门级仿真时，当**sel = ‘X’**时，**y**为**?**



1. “X”状态的传递(3)

MUX的模型：

- if(sel = ‘1’) then
- y <= ‘0’;
- elsif(sel = ‘0’) then
- y <= ‘1’;
- else
- y <= ‘X`;
- end if;

•建模时，加上了‘X’状态的处理。

1. “X”状态的传递(4)

- 要尽力避免不确定态的出现：
  - 电路初始化要完善，特别是时序电路中要有可靠的复位描述；
  - 尽量少用组合回环；

- 测试向量要完善，把可能的不确定态测试出来。

1. “X”状态的传递(5)

时序电路的复位例子:时钟处理模块(2分频)

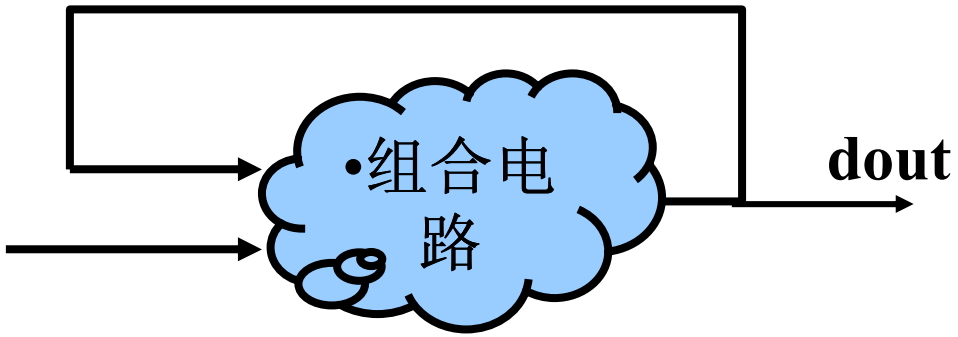
```
Process( clk )
Begin
  If( clk’event and clk = ‘1’ )
     then
       clk2d <= not din;
  End if;
End process;
```

```
Process( clk, reset_n )
Begin
  if( reset_n = ‘0’ ) then
       clk2d <= ‘0’;
  elsIf( clk’event and clk = ‘1’ )
     then
       clk2d <= not din;
  End if;
End process;
```

注：时序电路极力推荐使用复位．在某些情况下，复位电路会稍微降低系统的工作速率。是否取消复位电路，要看具体的目标芯片和系统的速率需求。

1. “X”状态的传递(6)

- 组合回环的避免



FPGA设计中不要使用使用组合回环.

2. 敏感信号的问题(1)

A. 敏感信号表只对仿真引擎起作用，而综合工具不会受到影响．综合后生成的门级网表是保证不遗漏敏感信号的，因此设计者不小心造成的敏感信号表的遗漏往往会导致前仿真和后仿真的不一致。

2. 敏感信号的问题(2)

B. 引起硬件动作的被读信号应该都放在敏感信号表中，纯组合电路描述中的所有被读的信号都必须放在敏感信号表中，这些信号包括：

1) 组合电路描述中，所有被读取的信号；
2) 时序电路中的时钟信号，异步控制信号。

3. 条件判断语句

- 锁存的避免
- 无关态的使用
- 优先级问题

锁存的避免

冗余锁存器的例子:

```
Process( sel, latch_in )
Begin
    if( sel = ‘1’ )then
       out <= latch_in;
    end if;
End process;
```

修正后:

```
Process( sel, in)
Begin
    if( sel = ‘1’ )then
       out <= in;
    else
       out <=‘0’;
    end if;
End process;
```

锁存的避免

状态机的例子:

```
Process( clk, reset )
Begin
    if( reset = ‘1’ )then
       Pst <= s0;
    elsif( clk’event and clk = ‘1’) then
       Pst <= Nst;
    end if;
End process;
```

■ (续)

```
Process( Pst, din )
Begin
    case Pst is
        when s0 =>
            if( din = '1' ) then
               Nst <=  s1;
            else
               Nst <= s0;
            end if;
        when s1 =>
….
End process;
```

•在din = '0'时，初学者往往认为状态机不翻转，则可不必再对Nst赋值，因为此时Nst和Pst的值一样，都为s0，再赋值一次显得多余。

•事实上，这个"多余"的赋值非常重要，少了这一句，电路将工作不正常。因为次态译码从纯组合电路变成了锁存电路。

锁存的避免 — 总结

- 组合电路描述中，条件判断语句必须指明所有条件分支情况下，被赋值信号的值。

- 分支不完整，意味着电路需要在某种电平状态下，让被赋值的信号“保持原值”，这只能使用锁存电路实现。

无关态的使用

代码段1：

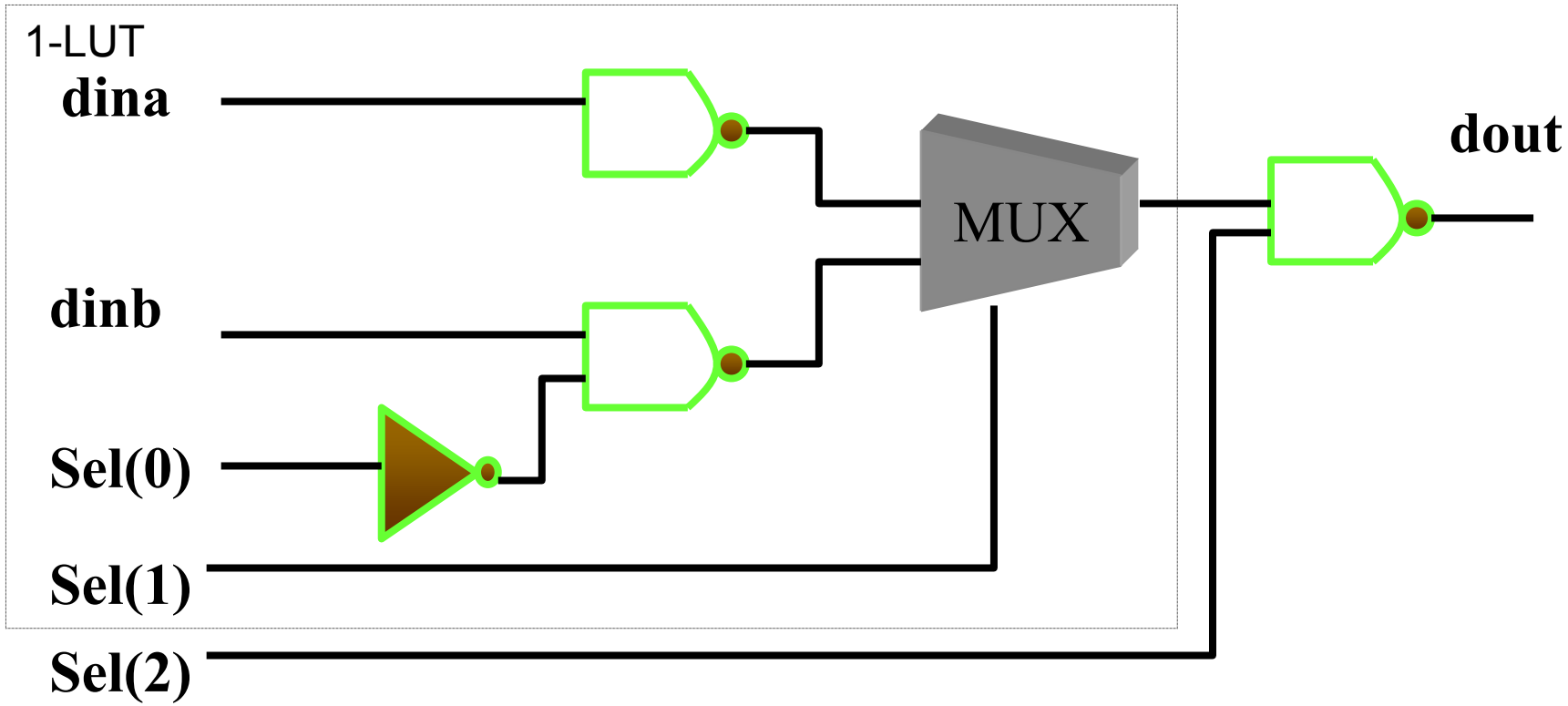
```
Case sel is
   when “000” =>  dout <= dina;
   when “010” => dout <= dinb;
   when others => dout <= ‘0’;
End case;
```

代码段2:

```
Case sel is
   when “000” =>  dout <= dina;
   when “010” => dout <= dinb;
   when others => dout <= ‘—’;
End case;
```

无关态的使用

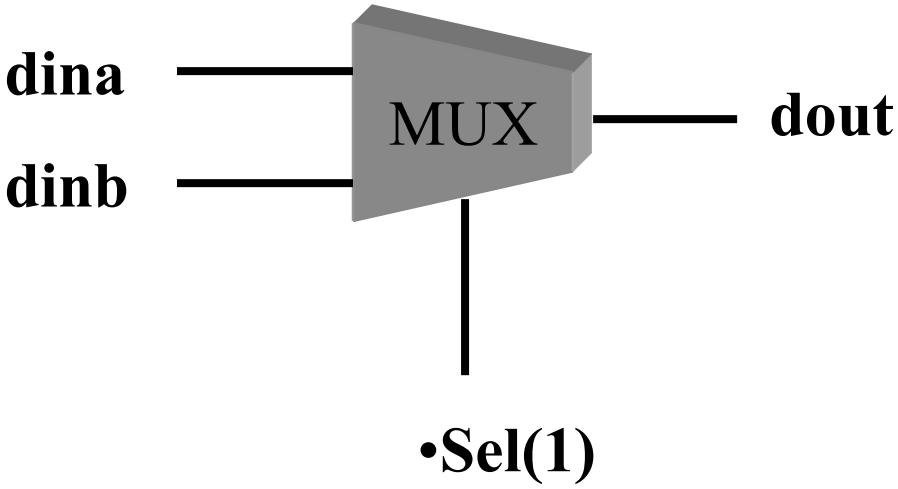
- 代码段1的综合结果：



需要两级4输入LUT实现

无关态的使用

- 代码段2的综合结果：



需要一级4输入LUT实现

无关态的使用 — 总结

- 对比以上两段代码的综合结果，可以发现，善于使用无关态‘-’来填补分支，可以引导综合工具生成优化的电路。
- 无关态‘-’在本质上时起了冗余电路删简的作用。

优先级问题

1. Case语句无优先级，if语句有优先级。
2. Multiple if statement和single if statement具有两种不同的优先级顺序。

Single if

```
Process( a )
Begin
    if      ( a(0) = ‘1’ ) then b <= “001”;
    elsif( a(1) = ‘1’ ) then b <= “010”;
    elsif( a(2) = ‘1’ ) then b <= “100”;
    else                                b <= “000”;
    end if;
End process;
```

Multiple if

```
Process( a )
Begin
    b <= “000”;
    if( a (0) = ‘1 ‘ ) then b <= “001”; end if;
    if( a (1) = ‘1 ‘ ) then b <= “010”; end if;
    if( a (2) = ‘1 ‘ ) then b <= “100”; end if;
End process;
```

4. 多驱动与总线复用

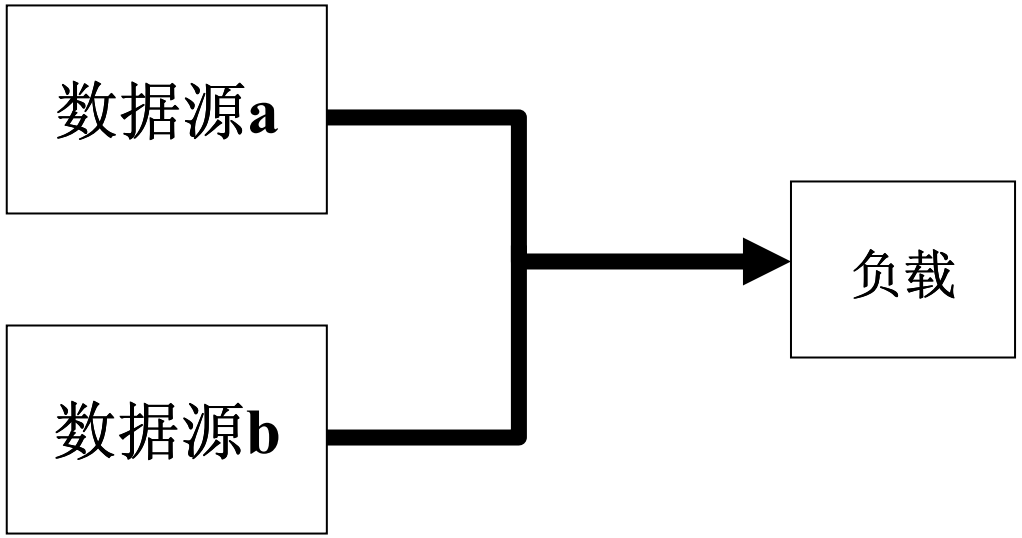
- 当多个信号源同时去驱动同一个负载，就会形成多驱动。
- 没有处理好多驱动，不但会造成逻辑混乱，而且容易损坏器件。

多驱动与总线复用

- 多驱动处理的要点：
  - 行为级思维 → 硬件思维；
  - 总线复用；
  - 线与；
  - 双向端口中的高阻态；

总线复用

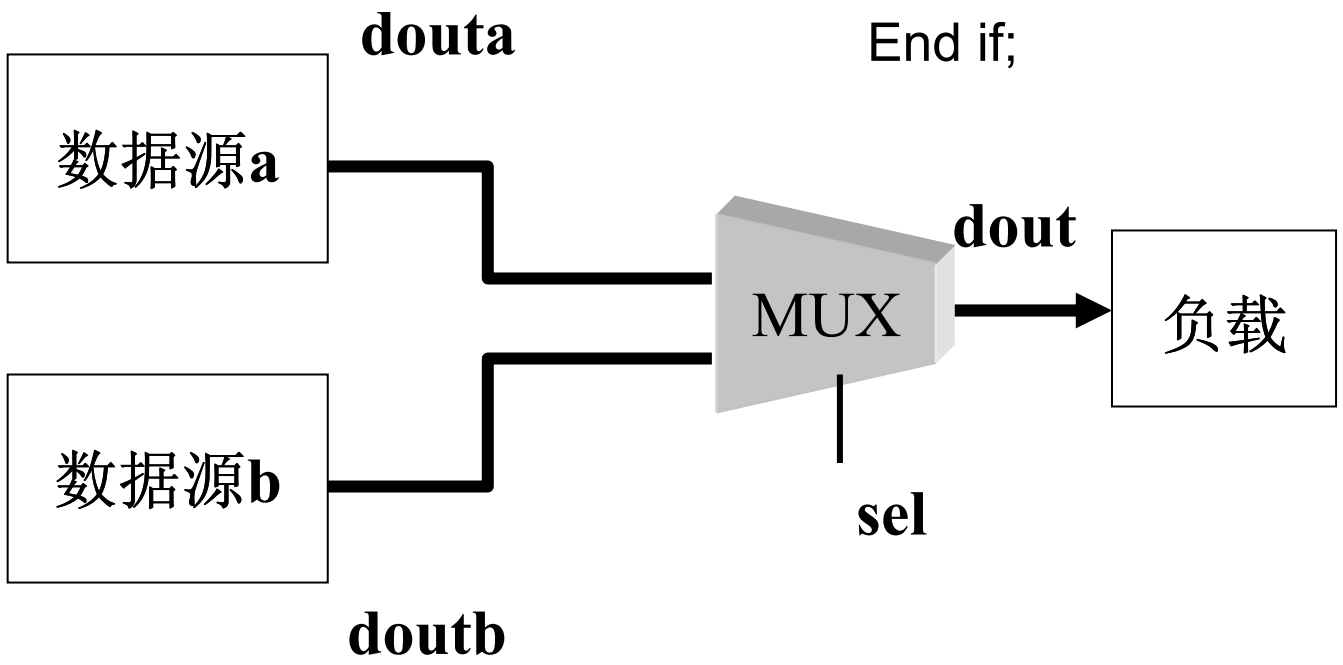
- 两个模块输出数据给同一个负载模块的设计中，必须处理好总线复用。



多路选择控制：

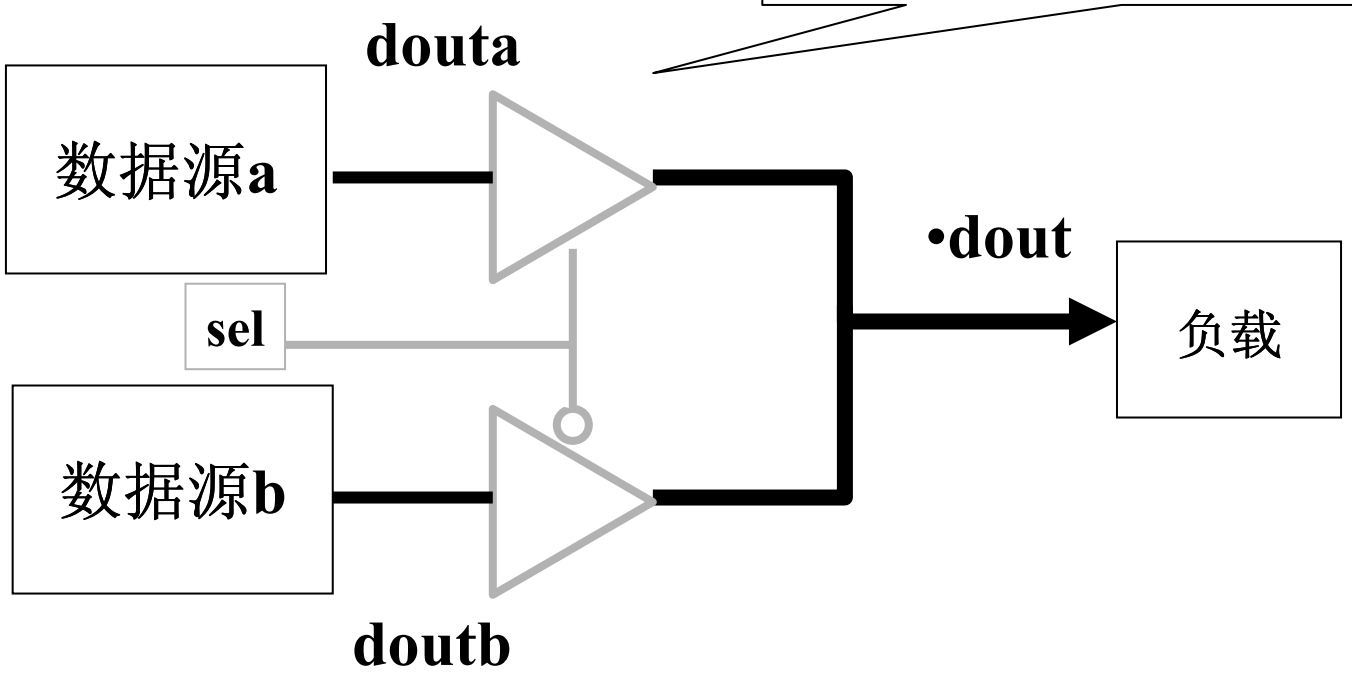
代码描述：

```
If ( sel = '1' ) then
    dout <= douta;
Else
    dout <= doutb
End if;
```



三态控制：

•在IO中经常使用
•尽量不要在芯片内部使用三态.
- 一些FPGA不支持这种特性
- 在ASIC设计中会带来一些测试上的问题。

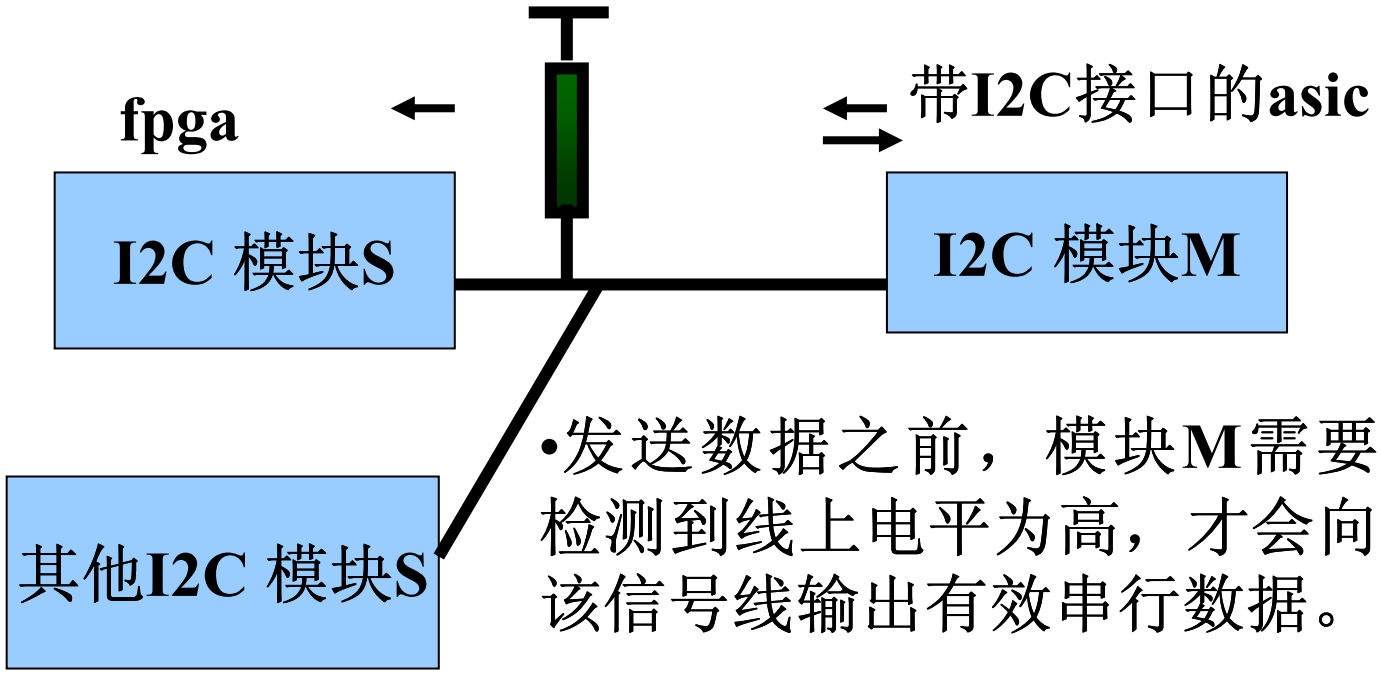


多驱动与总线复用

- 多驱动处理的要点：
  - 行为级思维 → 硬件思维；
  - 总线复用；
  - 线与；
  - 双向端口中的高阻态；

- CMOS工艺漏极开路输出(Open drain)的电路才能实现线与；
- 对应于TTL工艺，则为集电极开路(Open collector)。
- 部分FPGA可以将引脚设置为OD门输出，这个时候可以实现正常的线与逻辑；但是有部分FPGA没有这种输出逻辑，这时可以用高阻输出来代替OD模式，以实现安全的线与功能。当然这个时候要在芯片外部使用上下拉电阻来代替线与情况下的弱输出。

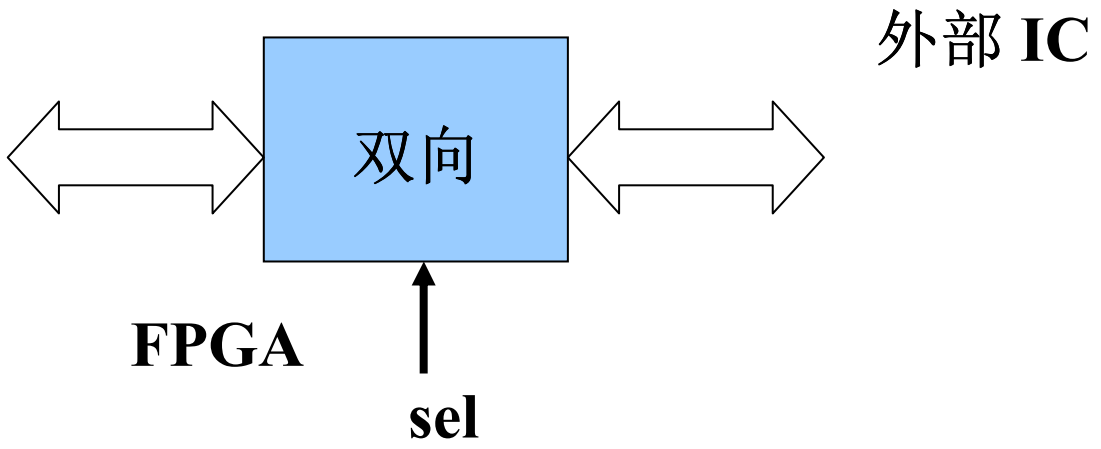
例子：I2C控制模块。


fpga
带I2C接口的asic
I2C 模块S
I2C 模块M
其他I2C 模块S
•发送数据之前，模块M需要检测到线上电平为高，才会向该信号线输出有效串行数据。

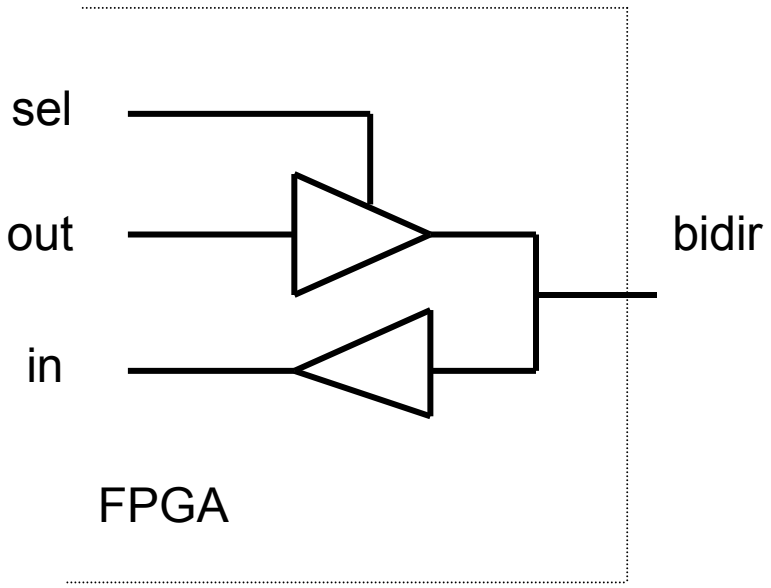
多驱动与总线复用

- 多驱动处理的要点：
  - 行为级思维 → 硬件思维；
  - 总线复用；
  - 线与；
  - 双向端口中的高阻态；

- 描述全双向端口，当sel = ‘1’时，数据从FPGA流向外部 **IC** ； sel = ‘0’时，数据从外部 **IC**流向FPGA 。



双向端口代码实现：分两个方向分别描述




```
-- Ext IC→ FPGA :
in <=    bidir;
-- FPGA → Ext IC:
if( sel = ‘1’ ) then
   bidir <= out;
else
   bidir <= “Z”;
end if;
```

双向端口设计

- 双向的设计时序规律为：可分为两个方向分别描述，对于作为被赋值者的任何一个端口，当它作为输出时，就被赋给输入的值；当它作为输入时，就被赋给高阻态。

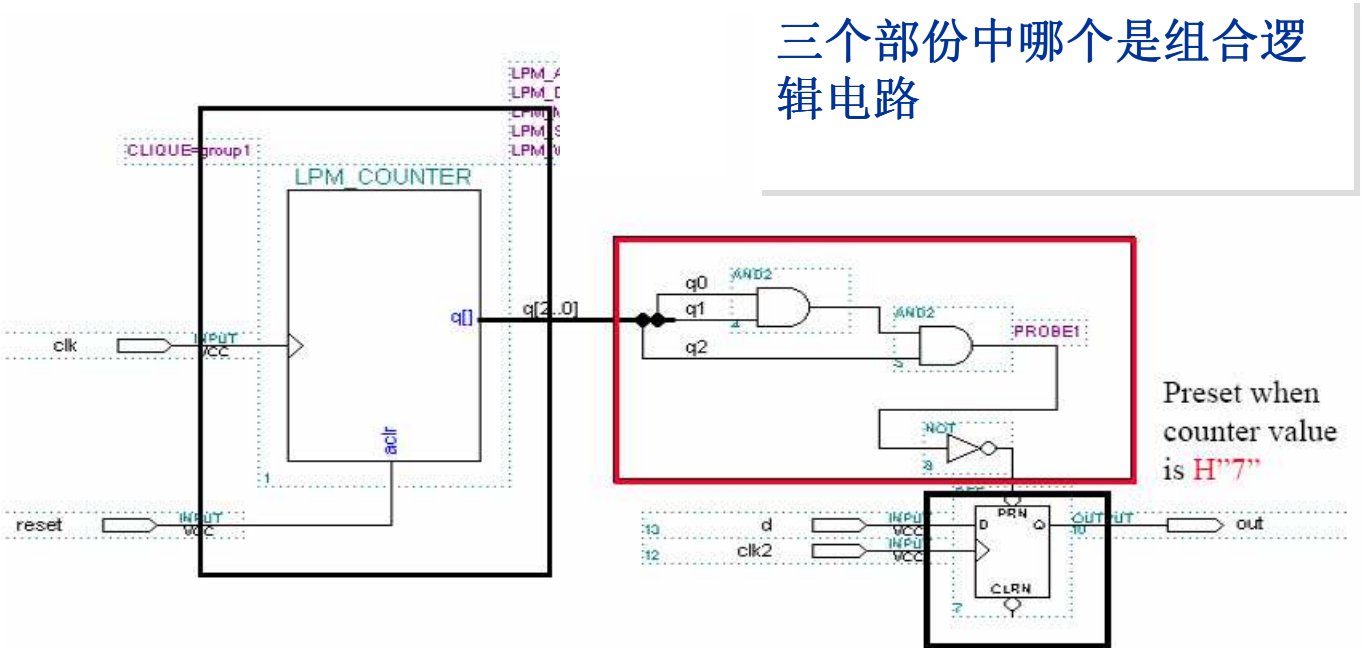
5. 毛刺的消除

- 毛刺产生的机理
  - 延时不平衡
  - 线间干扰
- 毛刺消除的方法
  - Gray编码
  - 寄存器吸收

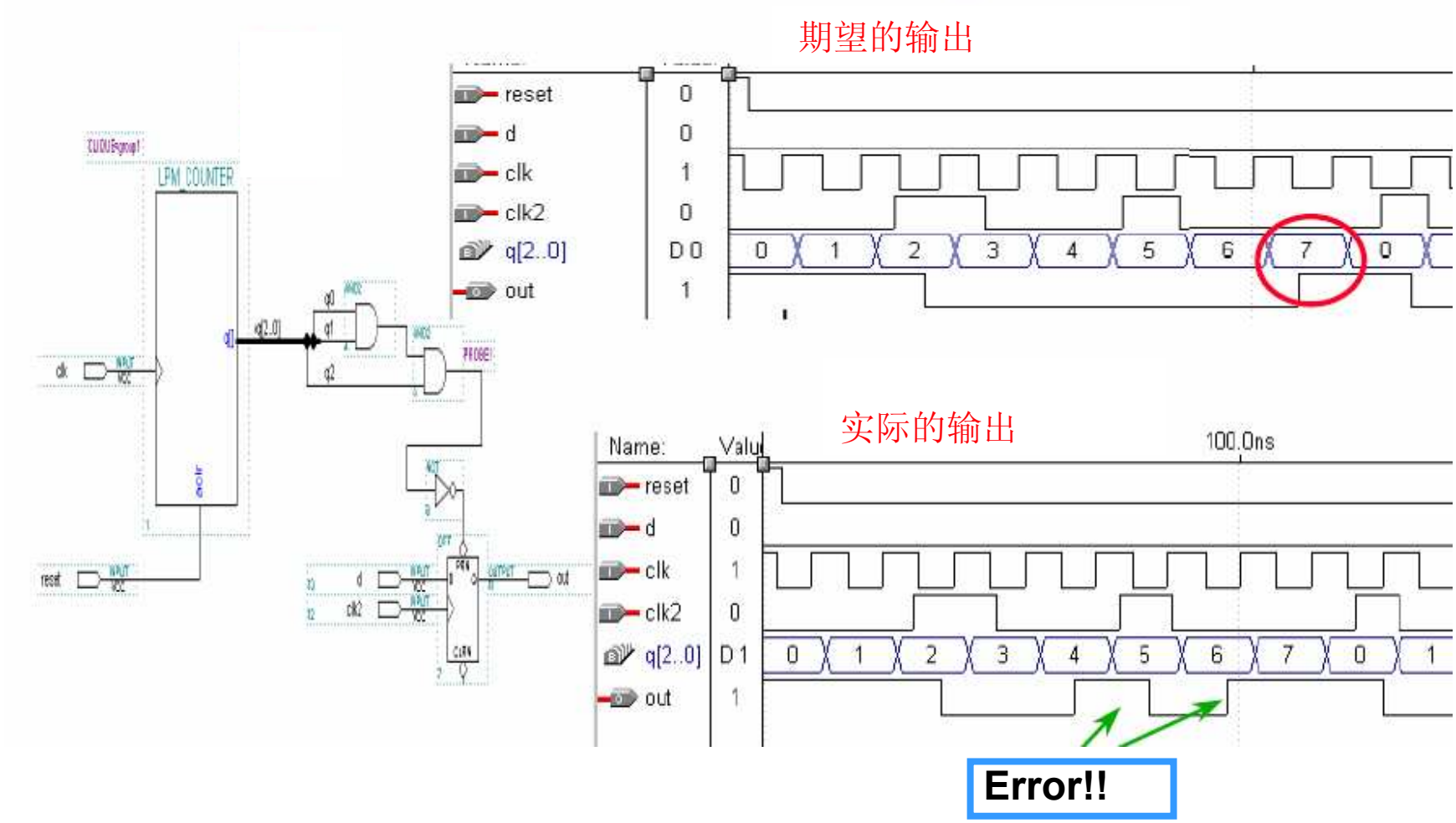
组合逻辑电路

- 如果一个电路满足下列条件,就是组合逻辑电路
  - 输出是同一时间所输入变量的函数(Outputs at a specified time are a function only of the inputs at that time).
- 比如:
  - Address decoder
  - Multiplexer
  - Adder
  - …

来看一个例子


三个部份中哪个是组合逻辑电路
CLIQUE=group1
LPM_COUNTER
clk
reset
aclr
q[]
q[2..0]
q0
q1
q2
AND2
AND2
PROBE1
NOT
Preset when counter value is H"7"
d
clk2
PRN
CLRN
out

仿真结果


期望的输出
reset
d
clk
clk2
q[2..0]
out
0
0
1
0
D 0
1
0 1 2 3 4 5 6 7 0
LPM_COUNTER
实际的输出
Name:
Valu
100.0ns
reset
d
clk
clk2
q[2..0]
out
0
0
1
0
D 1
1
0 1 2 3 4 5 6 7 0 1
Error!!

哦?

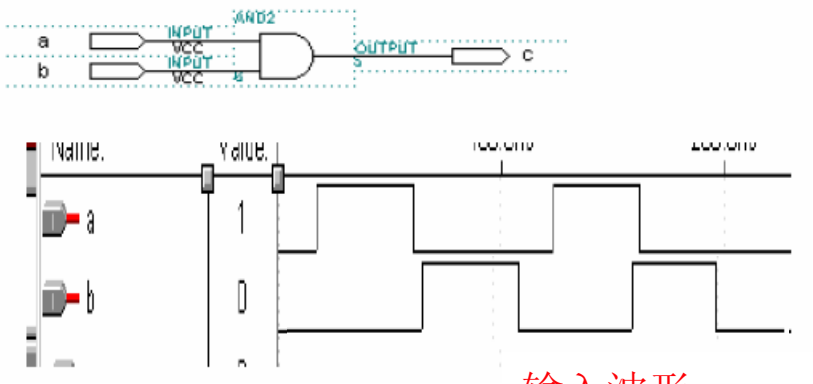
FPGA 质量有问题，我要用别家的！

怎么回事…？

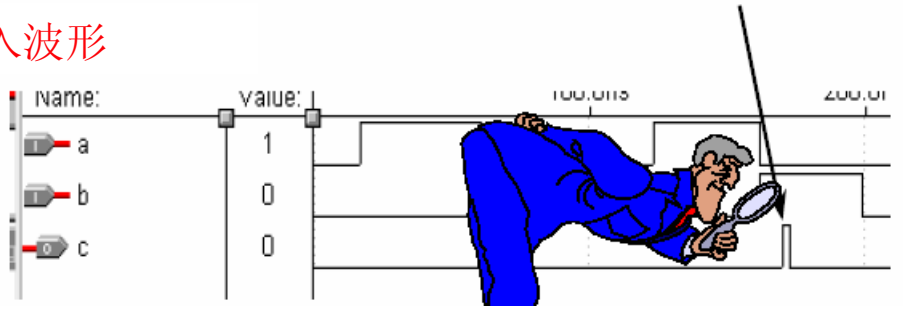
软件有 bug!
我再也不用你们的了！

最简单的组合逻辑电路

- 2 输入与门 / 2输入或门

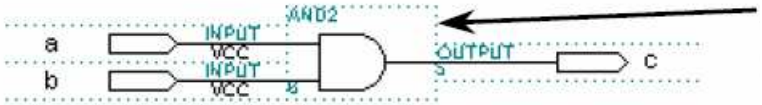


输入波形



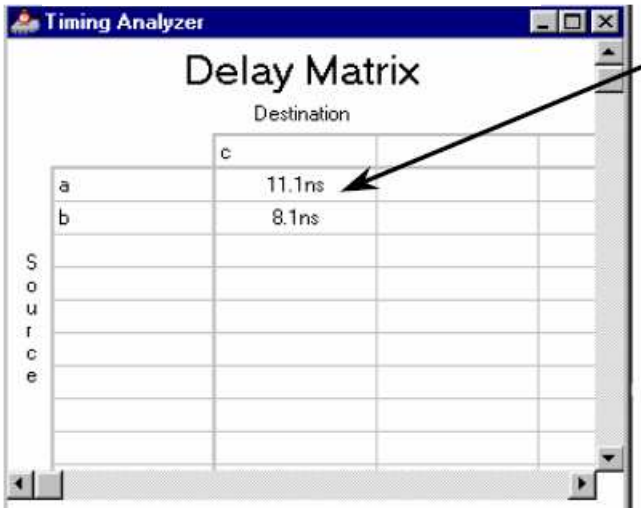
输出波形

仔细看看



AND门延迟为0.2ns

延迟

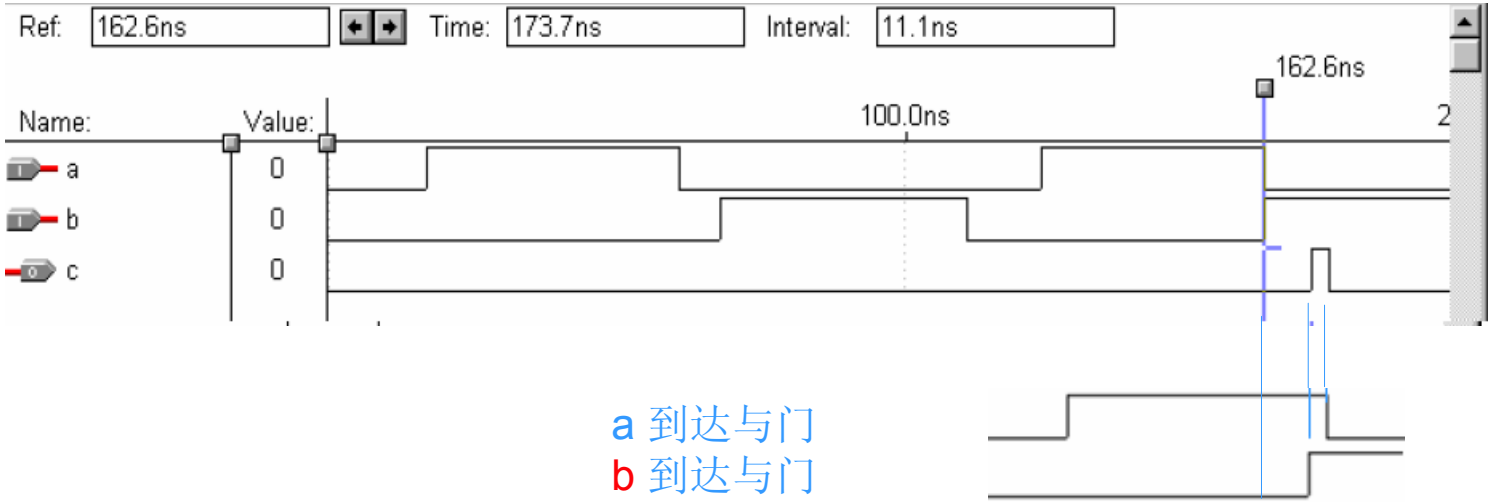


简单计算一下

For Signal b :
(Trace delay of b) + AND gate internal delay = 8.1ns
(Trace delay of b) + 0.2ns = 8.1ns
(Trace delay of b) = 7.9ns
For Signal a :
(Trace delay of a) + AND gate internal delay = 11.1ns
(Trace delay of a) + 0.2ns = 11.1ns
(Trace delay of a) = 10.9ns

毛刺分析

a 信号线的延迟: 10.9ns
b 信号线的延迟: 7.9ns



a 到达与门
b 到达与门

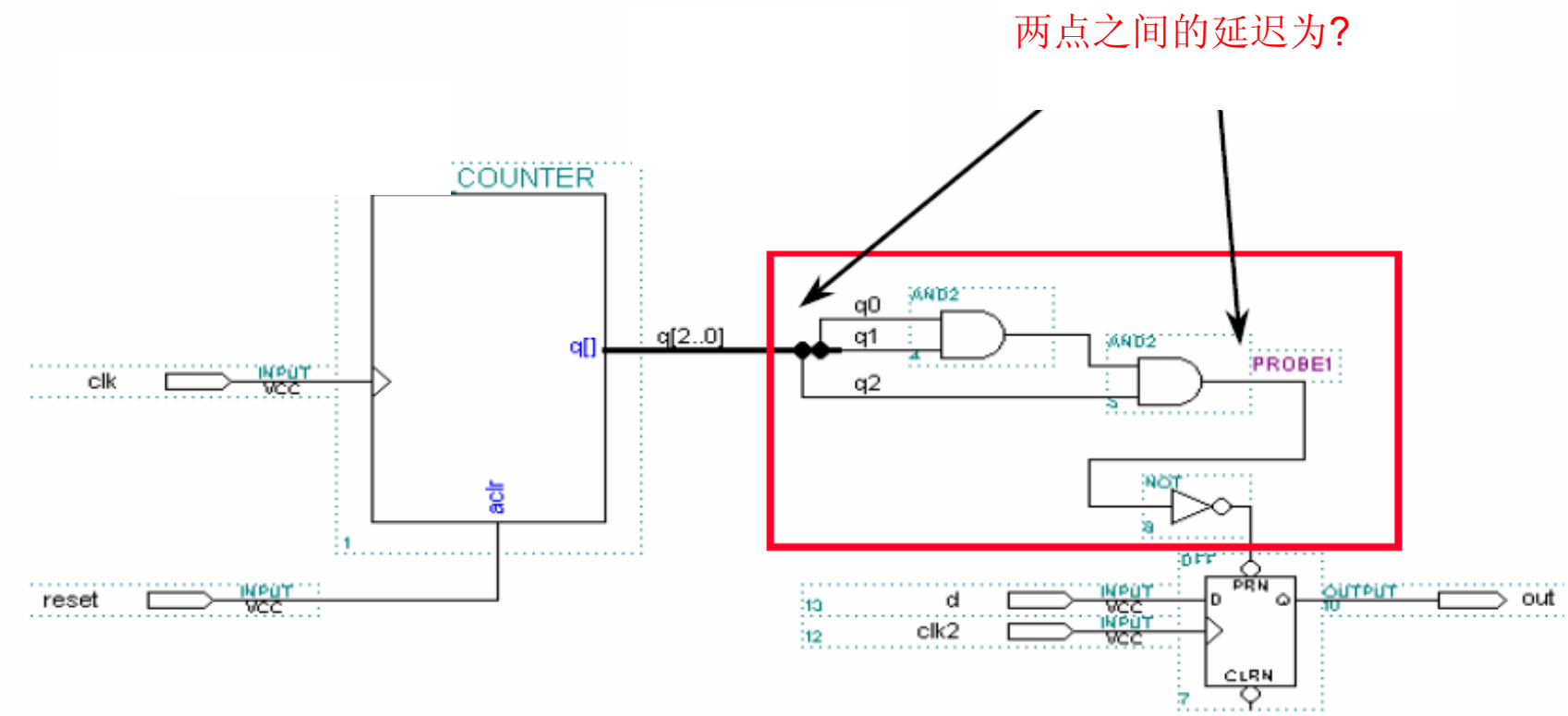
毛刺分析

- 功能仿真: 输出 C 应该为 “0”
- 时序仿真:输出 C 有一个宽度为3ns的脉冲
- 毛刺是由于连线的延迟引起
  - 一个设计不仅要 functional正确, 还要timing正确
  - 需要考虑门延迟和走线延迟

如果希望设计的电路稳定工作, 你需要考虑 TIMING 的因素.

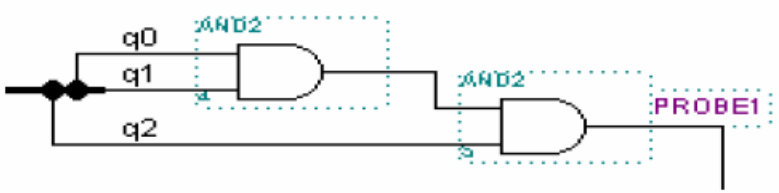
这不是器件和工具的问题, 问题出在设计上

回到前面的例子...


两点之间的延迟为?
COUNTER
clk
INPUT
VCC
reset
INPUT
VCC
q[]
aclr
q[2..0]
q0
q1
q2
AND2
AND2
PROBE1
NOT
DFF
PRN
D
Q
CLRN
d
clk2
INPUT
VCC
INPUT
VCC
OUTPUT
out

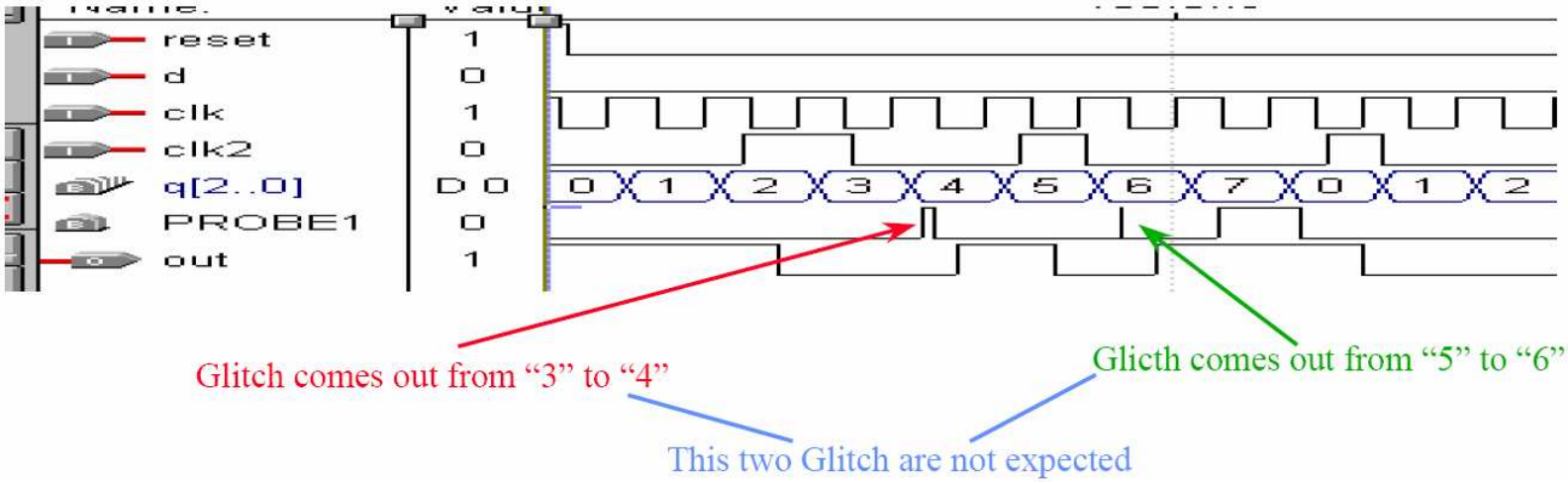
延迟表

	PROBE1
lpm_counter:1|dffs0.Q	3.9ns
lpm_counter:1|dffs1.Q	3.7ns
lpm_counter:1|dffs2.Q	2.0ns



Q2	Q1	Q0		
0	0	0	← Glitch ?	No
0	0	1	← Glitch ?	No
0	1	0	← Glitch ?	No
0	1	1	← Glitch ?	Yes
1	0	0	← Glitch ?	No
1	0	1	← Glitch ?	Yes
1	1	0	← Glitch ?	No
1	1	1		

在仿真中检查输出



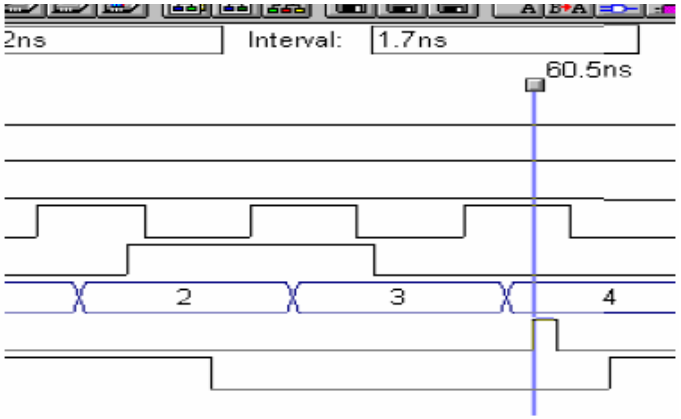
器件工作正常,仿真器工作也正常!

毛刺脉冲的宽度

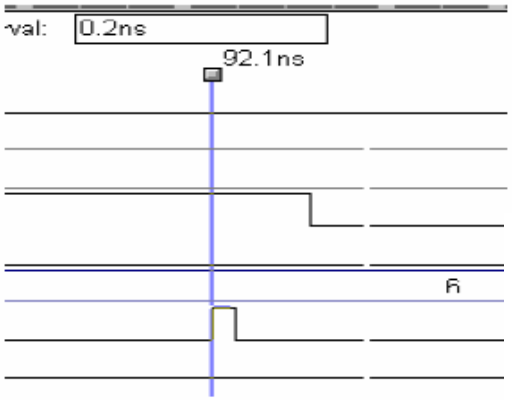
延迟表

	PROBE1
lpm_counter:1|dffs0.Q	3.9ns
lpm_counter:1|dffs1.Q	3.7ns
lpm_counter:1|dffs2.Q	2.0ns

“3” to “4”
011 -> 100



“5” to “6”
101 -> 110



毛刺的issue

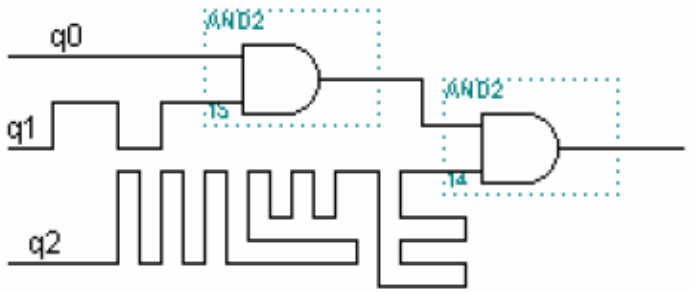
- 什么时候产生毛刺
  - 当组合逻辑电路一个以上的输入信号改变时
- 当设计组合逻辑电路时
  - 毛刺的产生不可避免
  - 如果碰巧没有, 只是运气
- 看起来很可怕,能解决吗?

方法一:

q0 = 3.9ns
q1 = 3.7ns
q2 = 2ns

有工程师用这样的方法

1.) extend the delay of q1 by +0.2ns
2.) extend the delay of q2 by +1.9ns
3.) then all q[2..0] delay is 3.9ns

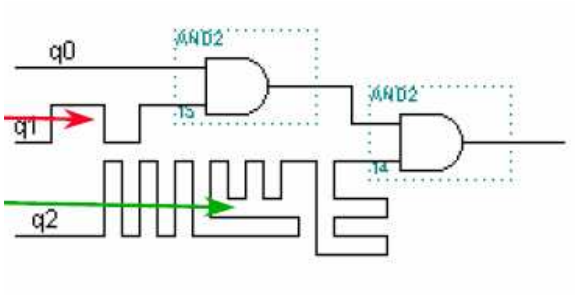


有工程师喜欢用这样的方法
并想一直用下去…

优点:

1) 问题解决的快
   (我们称之为 dirty solution)
2) 没有了…

苛刻的设计

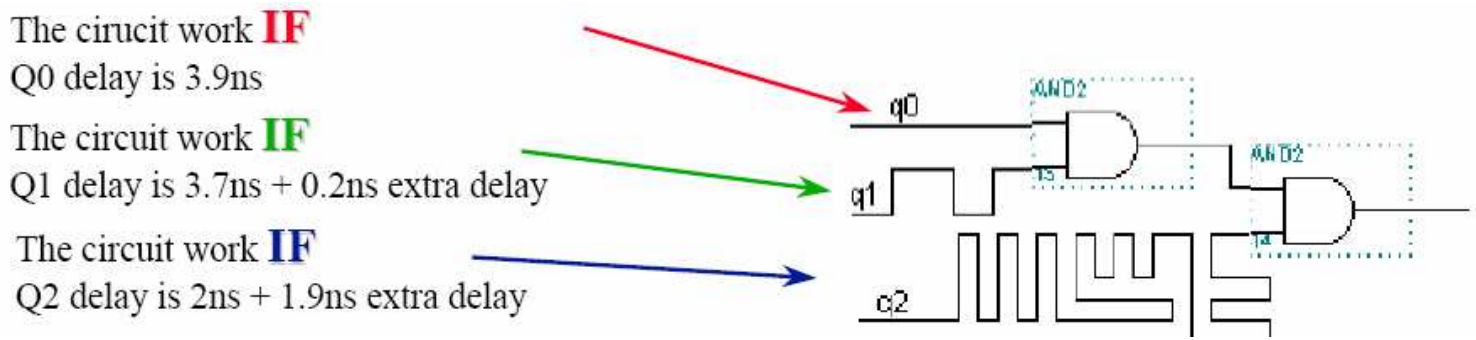


考虑下列因素了吗?

1) 不同的温度(0C,50C,70C)对延迟的影响
2) 不同的工作电压(3.3v,3.1v,3.5v)对延迟的影响
3) 芯片不同的工艺(.13um, 90ns)对延迟的影响
4) 其它部分设计修改对P&R 已及延迟的影响
5) 其它…

***Note*: Timing Analysis of tpd is similar**

假设的条件



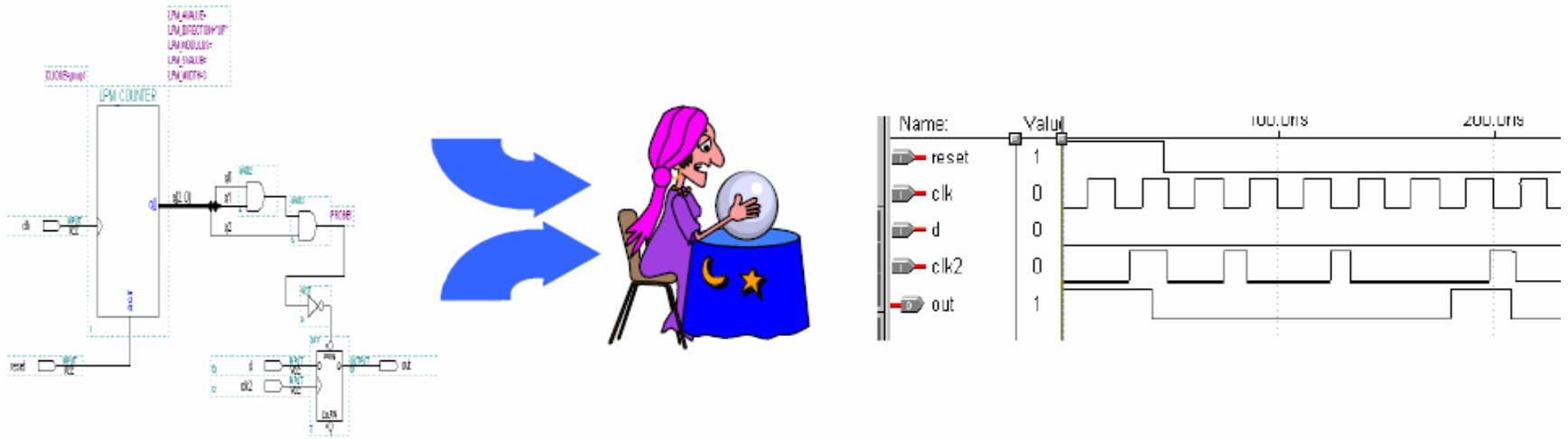
这样的电路工作在 If If If 下,一旦温度、工作电压、工艺改变?

一个 rigid的设计

电路现在工作正常,但愿
它明天也正常
祝你好运!
Bye Bye…

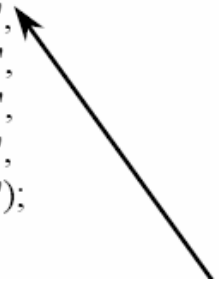
可靠的设计

重新设计电路,使其在各种条件下都能工作.



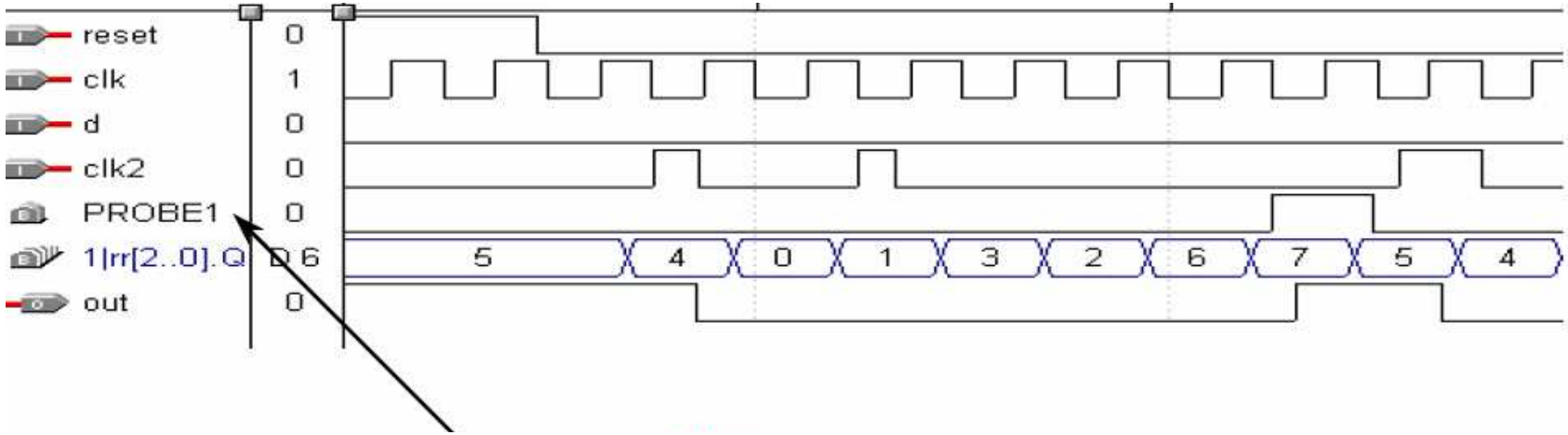
方法二:

```
subdesign modcount
( clk, reset : input;
  q[2..0] : output;
)
variable
counter3
    :MACHINE OF BITS (rr[2..0])
     WITH STATES(r0 = B"101",
                 r1 = B"100",
                 r2 = B"000",
                 r3 = B"001",
                 r4 = B"011",
                 r5 = B"010",
                 r6 = B"110",
                 r7 = B"111");
begin
q[] = rr[];
counter3.reset = reset;
counter3.clk = clk;
CASE counter3 IS
            when r0 => counter3 = r1;
    when r1 => counter3 = r2;
    when r2 => counter3 = r3;
    when r3 => counter3 = r4;
    when r4 => counter3 = r5;
    when r5 => counter3 = r6;
    when r6 => counter3 = r7;
    when r7 => counter3 = r0;
END CASE;
end;
```



Q0,Q1,Q2每次改变一位

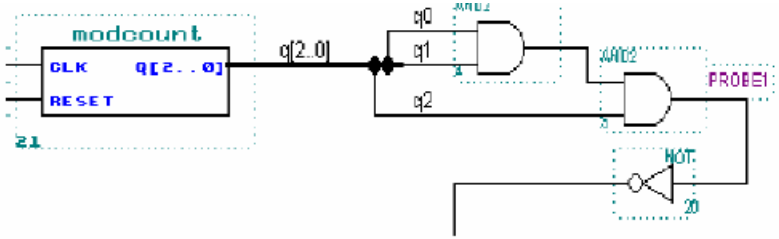
时序仿真结果



重新设计后，电路不再和连线与门的延迟有关连，不再有毛刺出现。

Robust设计的定义

The circuit work IF



There is no any assumption on this circuit, it work at any Vcc, Temp, and Process !!!!!! Great

重新规划设计的结构，使电路在各种条件下都能可靠的工作。

毛刺消除的方法(1)

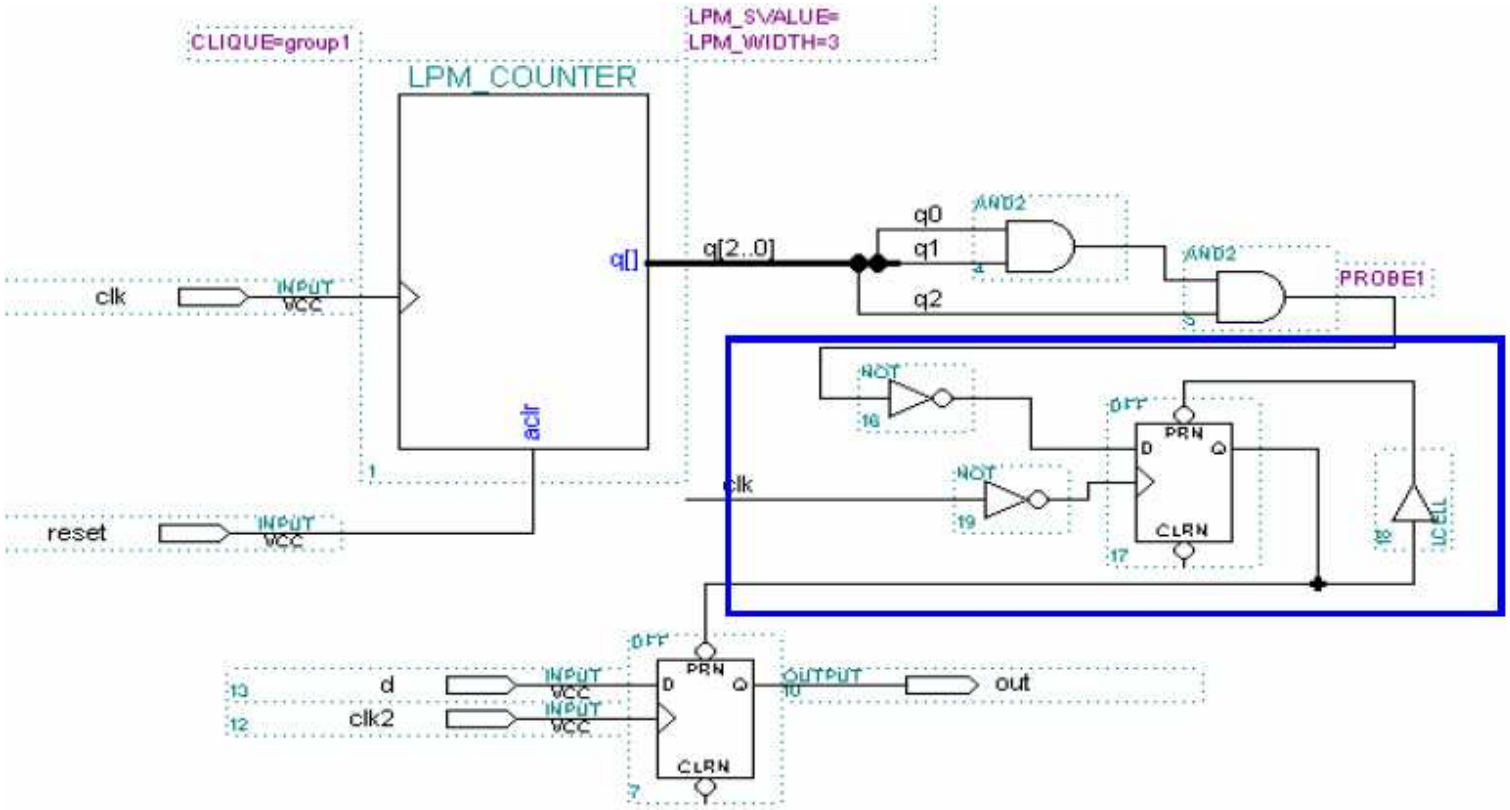
Gray编码方法

- 在任何相邻的两组代码中，仅有一位数码不同
- 计数器(状态机)电路中，采用Gray编码可以避免总线上的多个bit同时在一个时钟周期内翻转而导致毛刺。

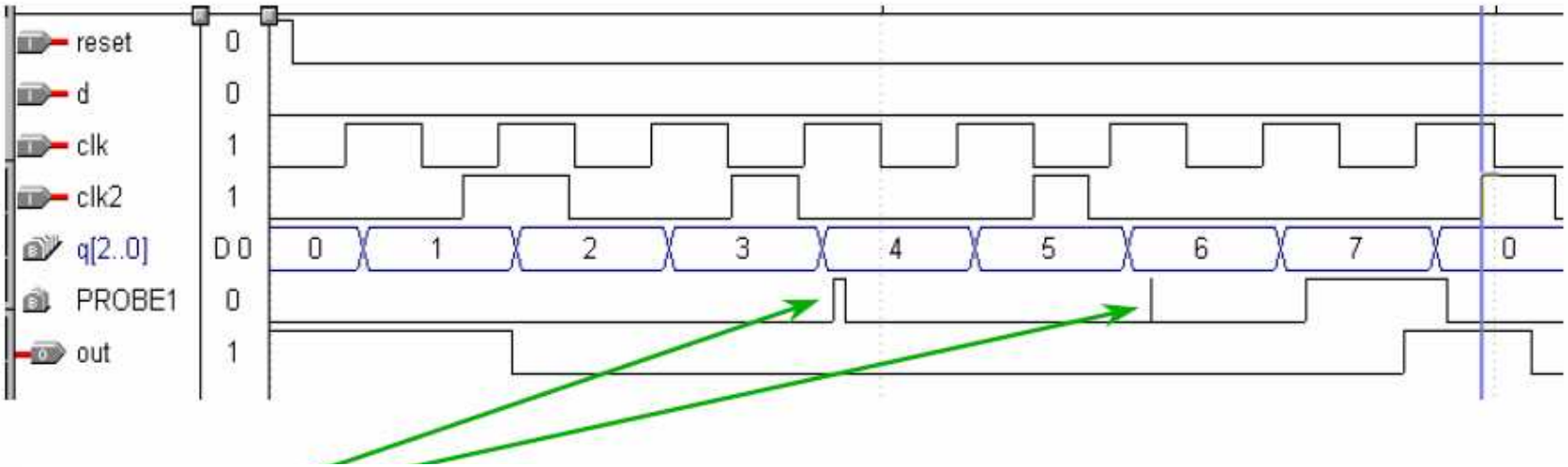
注：Gray码方法也是降低设计功耗的一个手段，因为它降低了寄存器的电平翻转率。

方法三:

修改电路如下:


CLIQUE=group1
LPM_SVALUE=
LPM_WIDTH=3
LPM_COUNTER
clk
INPUT
VCC
reset
INPUT
VCC
q[]
q[2..0]
aclr
q0
q1
q2
AND2
AND2
PROBE1
NOT
NOT
clk
DFF
PRN
D
Q
CLRN
LCELL
d
clk2
INPUT
OUTPUT
out

时序仿真结果

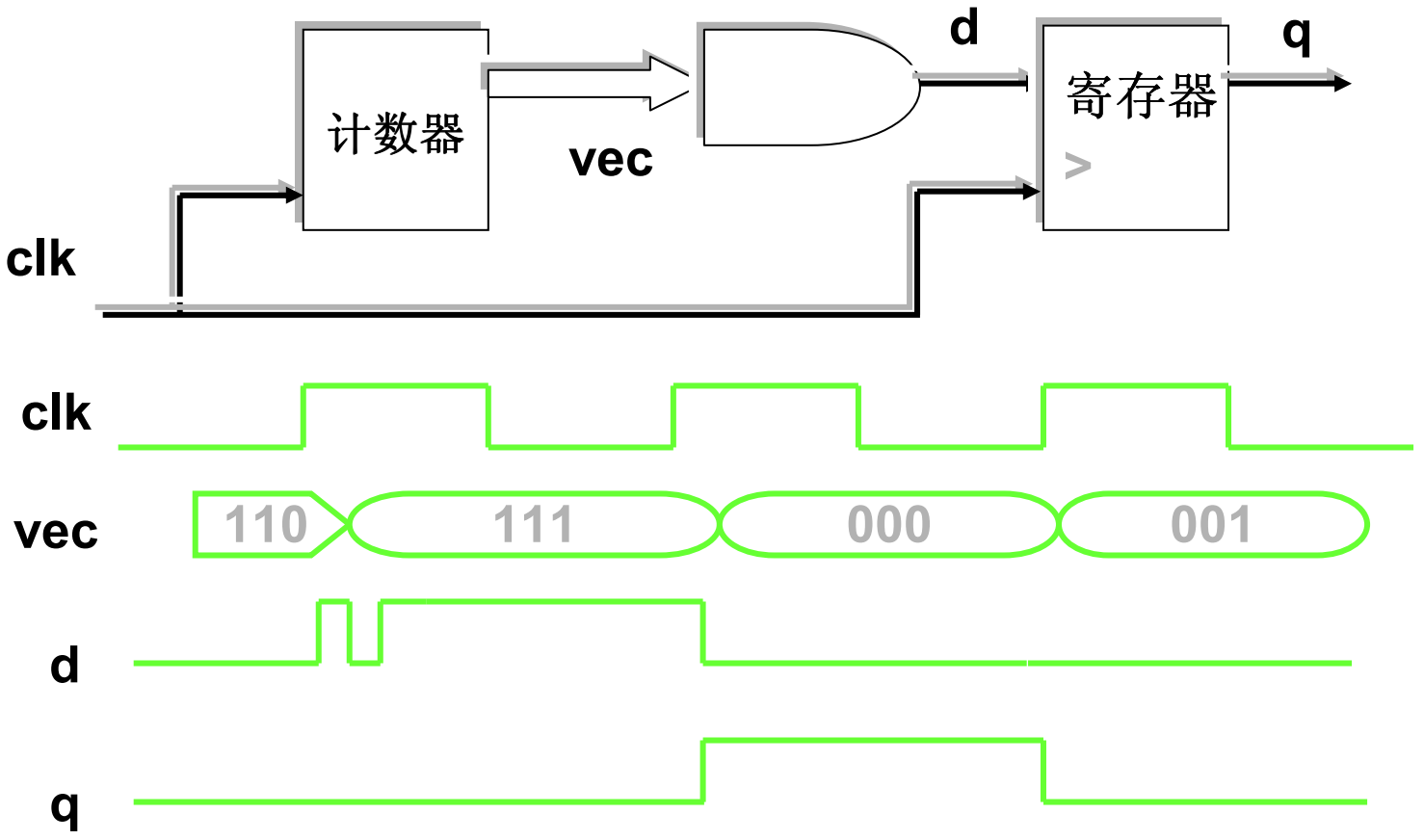


毛刺仍然存在，但新的设计对毛刺有免疫性!
时钟上升沿到来前数据信号已稳定.

毛刺消除的方法(2)

- 同步设计：
  - 寄存器的数据输入端和时钟使能端对毛刺不敏感，因此可以利用它们来“吸收”毛刺信号。

工作原理



请注意:

- 当组合逻辑的输出信号用作下列信号时:
  - FF异步清零
  - FF异步复位
  - FF的时钟
  - Latch的门控…

小结

- 组合逻辑电路是最常见的电路形态
- 毛刺是组合逻辑电路的固有属性
- 毛刺产生的机理
  - 组合延时，布线延时的不平衡，导致译码输出毛刺。
  - 线间的信号耦合，导致毛刺的产生。
- 如果考虑不周，组合逻辑产生的毛刺会引起系统的错误动作
  - 尤其是异步复位、时钟等输入端出现毛刺

提纲

- 可靠设计
- 同步设计
- 异步设计
- 提升设计性能

- 同步设计的概念
- 时钟质量的保证
- 路径延迟与系统最高工作频率

同步设计的概念

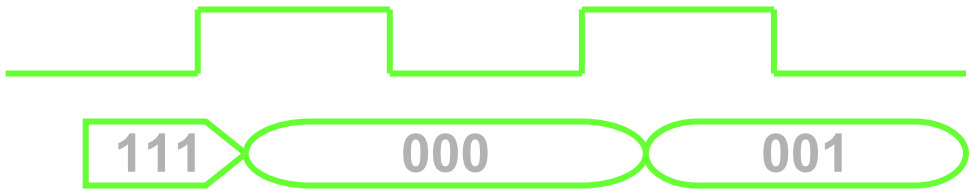
整个系统共用同一个时钟，或其派生(分频，倍频等，保证与源时钟有确定的相位关系) 时钟。

系统中的主要存储元件大都是时钟沿敏感的元件(即寄存器)，而不是电平敏感的元件(即锁存器)。

- 数字系统设计中，应该尽量地采用纯粹的同步系统(单时钟系统)设计。
- 异步设计会给电路带来很多不安全的因素。

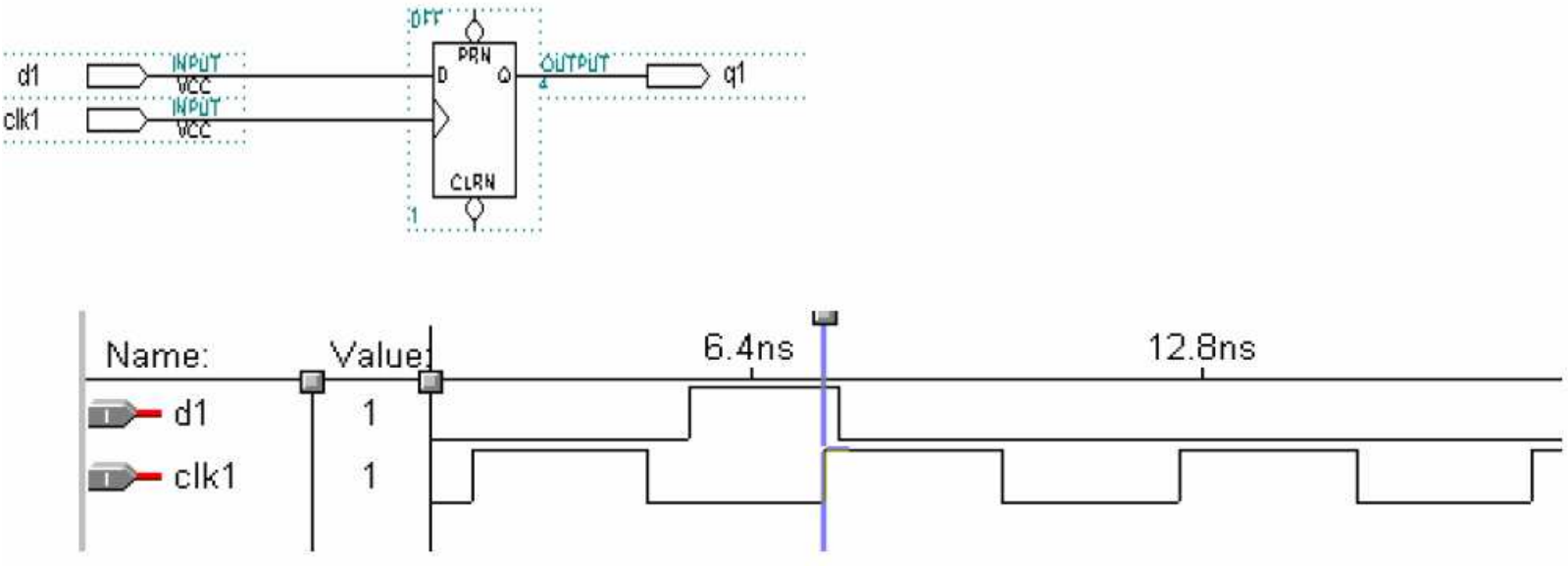
同步系统的时序特点

- 信号变化都是发生在时钟沿(之后的微小时间处)，即系统中的动作基本上都是“绑定”在时钟沿上。
- 信号在敏感的时钟沿之后可能会有一段不稳定时间，随后将保持一段时间的稳定，等待下一个敏感的时钟沿的采样。



建立/保持时间的定义

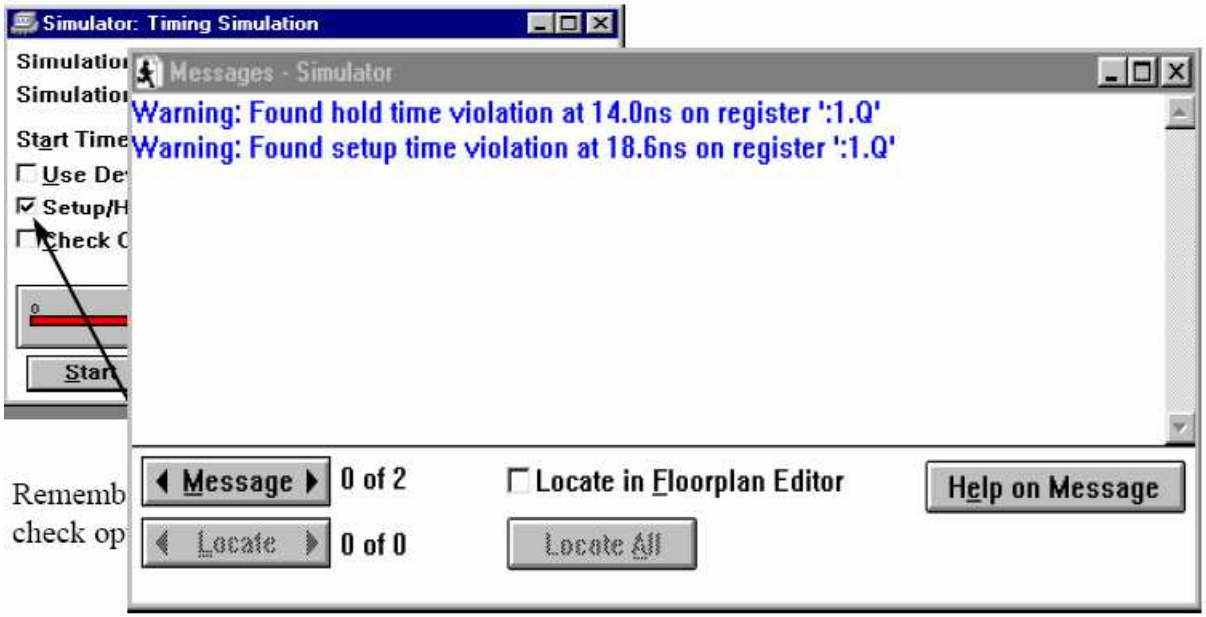
- Setup timing定义：输入数据信号在clock信号的上升沿到来前必须稳定的最小时间。
- Hold timing定义：输入数据信号在clock信号的上升沿过去后必须稳定的最小时间。



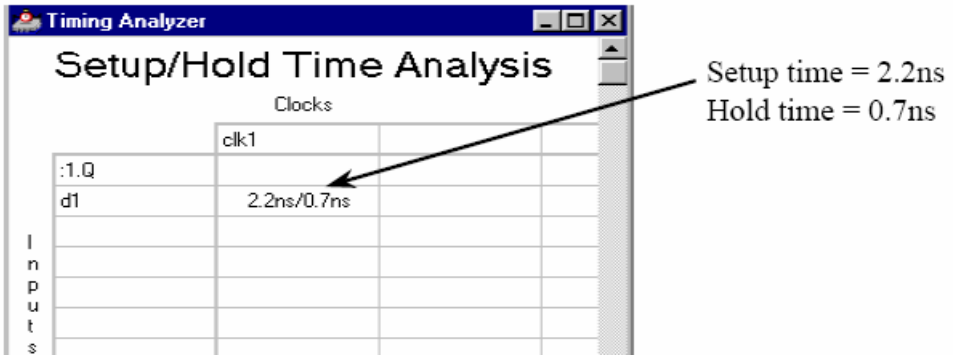
这么简单的设计会有问题?

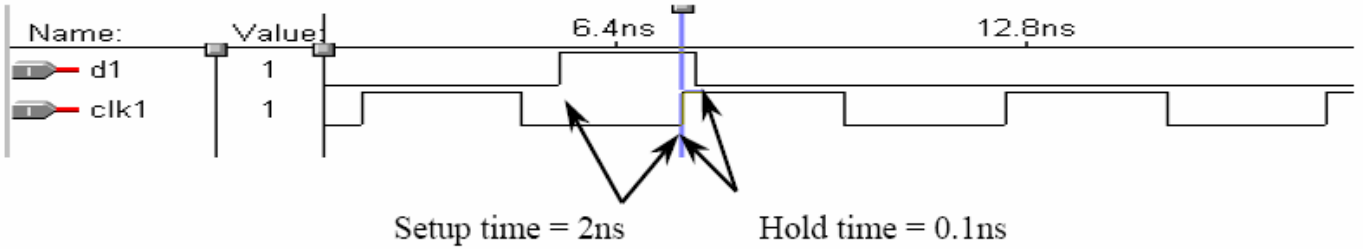
– take a look…

不能满足建立 / 保持时间的最小要求，触发器将不能正常反转（出现亚稳态）



输入波形





According to Setup/Hold Matrix
Setup time needs = 2.2ns
Hold time needs = 0.7ns

How to fix ?
It is easy. Extend the Setup time from 2ns to 2.2ns
Extend the Hold time from 0.1ns to 0.7ns

提纲

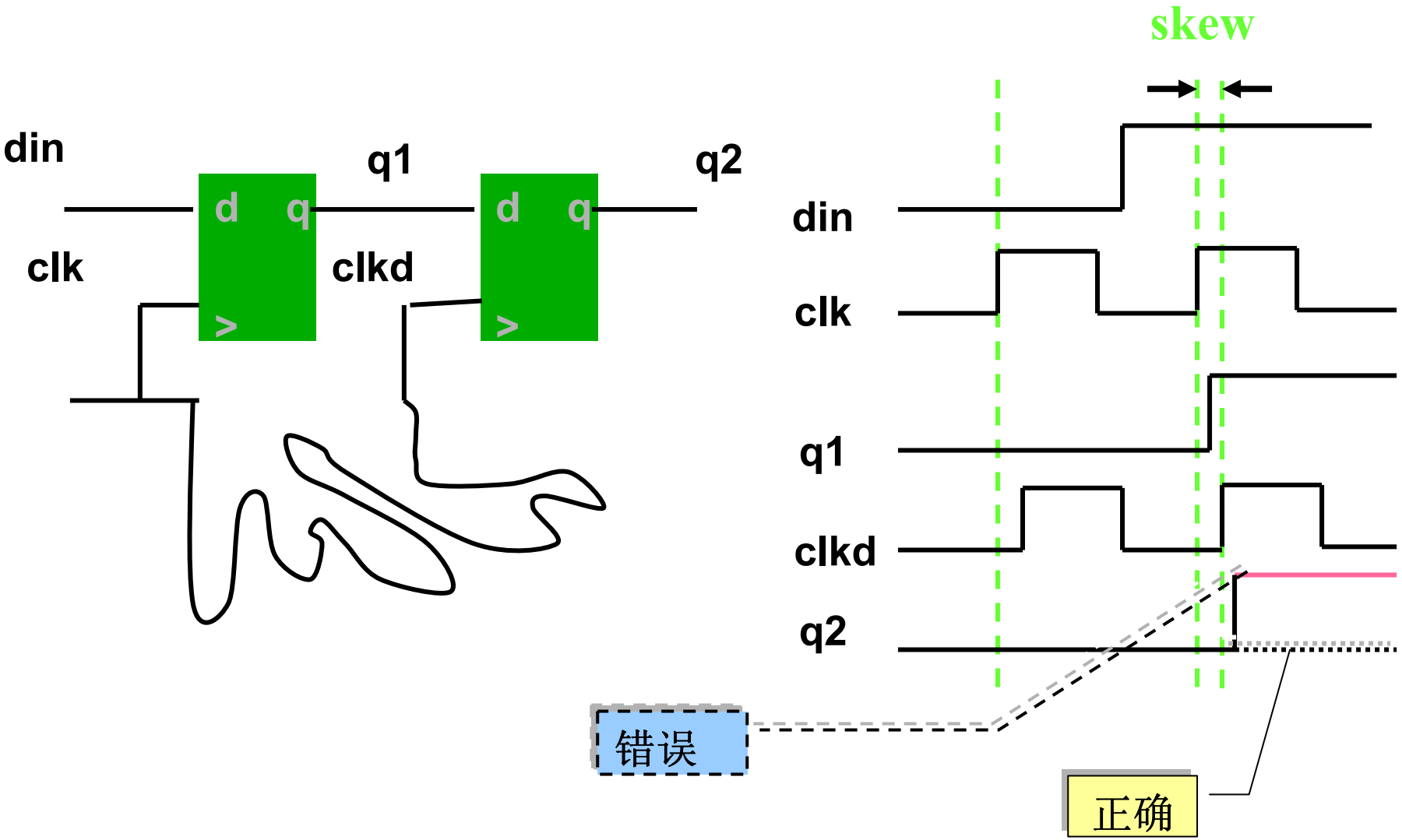
- 可靠设计
- 同步设计
- 异步设计
- 提升设计性能的技巧

- 同步设计的概念
- 时钟质量的保证
- 路径延迟与系统最高工作频率

时钟质量的保证

1. 减少时钟的Skew；
2. PLL的使用；
3. 门控时钟与时钟使能；
4. 派生时钟与派生使能；
5. 其他…

同步系统中的时钟skew (偏移)



同步系统设计中应使时钟的**skew**最小

-使用**FPGA**中快速的全局时钟走线来布时钟线**(**在**fpga**中，对全局走线有特殊处理，如采用全铜层工艺来实现全局通道**)**。

- 在前端设计上，应遵循一定的设计原则来避免时钟的过分偏移。使用时钟树**(ASIC)**

时钟质量的保证

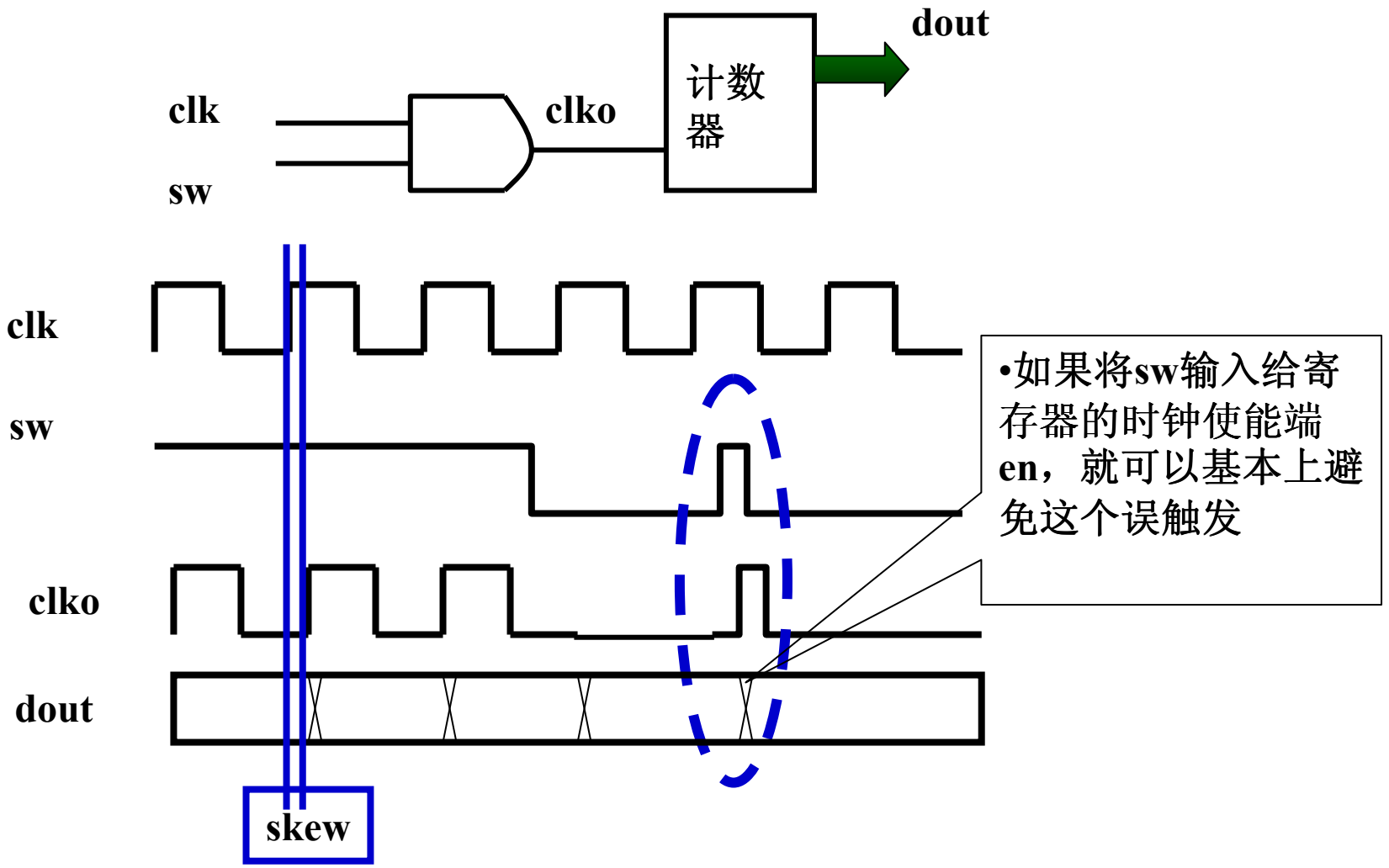
1. 减少时钟的Skew；
2. PLL的使用；
3. 门控时钟与时钟使能；
4. 派生时钟与派生使能；
5. 其他…

•目前所使用的中、高端**FPGA**内部均集成了**DPLL**，甚至模拟锁相环。
•这些锁相环提供对时钟的分频、倍频以及移相功能，并且保证**skew**最小。

时钟质量的保证

1. 减少时钟的Skew；
2. PLL的使用；
3. 门控时钟与时钟使能；
4. 派生时钟与派生使能；
5. 其他…

门控时钟(gated clk)



Gated clk的危害

1. 容易导致时序电路的误触发；
2. 增大了clk的skew。

Gated Clk也不是绝对不可用的，在某些情况下，它可以作为低功耗设计的手段。

- 因此在设计中，尽量避免时钟通过组合电路，避免使用组合电路来产生时钟。

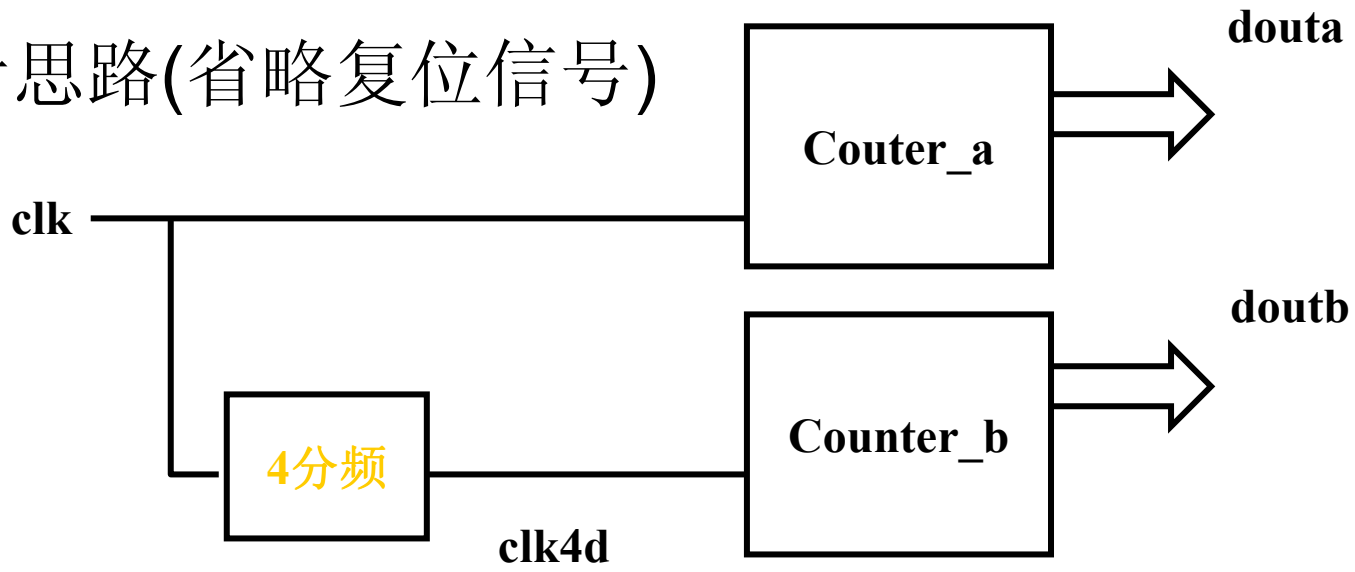
时钟质量的保证

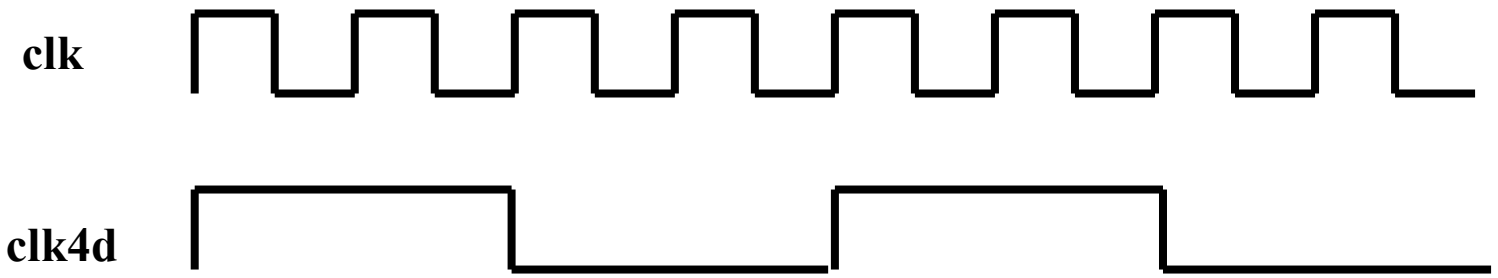
1. 减少时钟的Skew；
2. PLL的使用；
3. 门控时钟与时钟使能；
4. 派生时钟与派生使能；
5. 其他…

例子

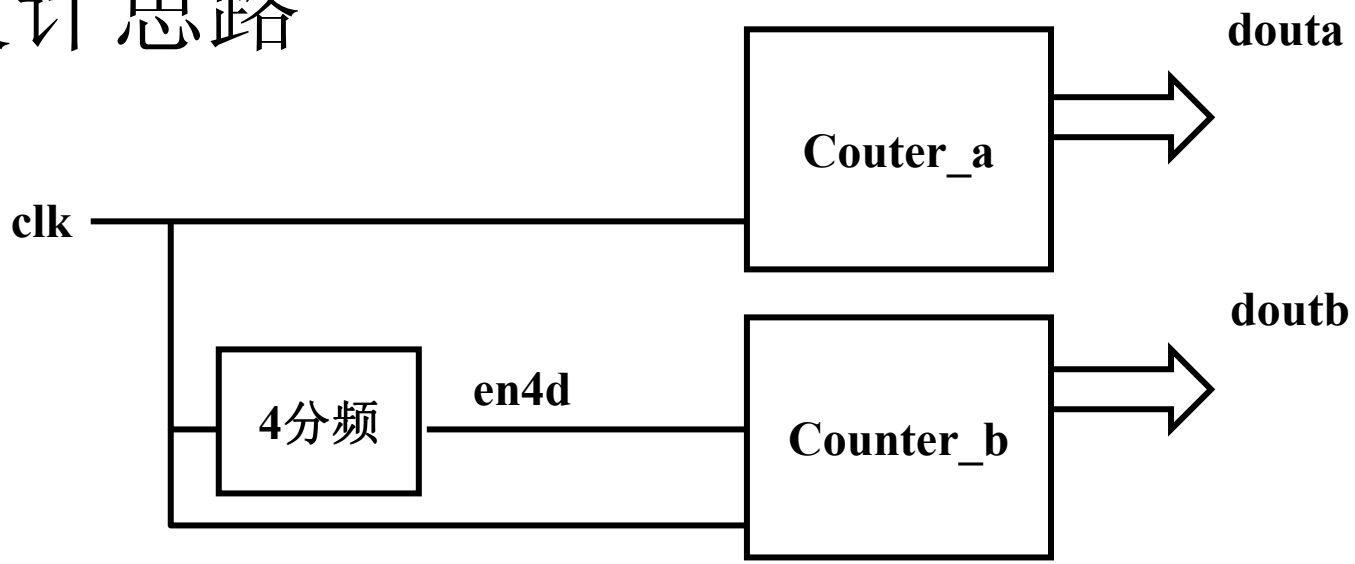
- 设计一个计数器系统，该系统包含两个计数器，其中一个计数器以系统时钟clk频率计数；另一个计数器以clk的1/4频率计数。

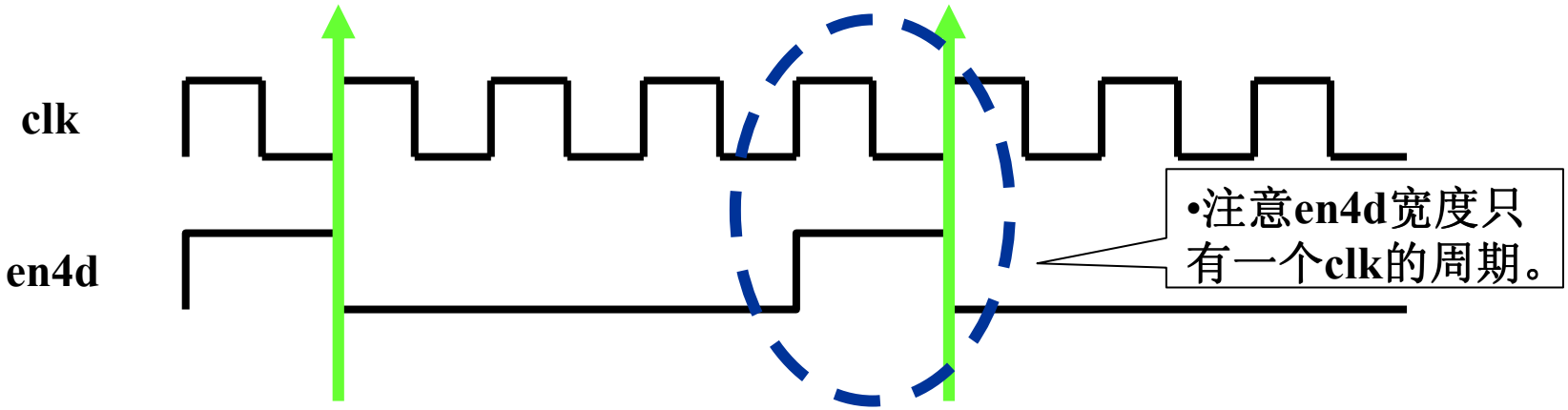
一种设计思路(省略复位信号)





优化设计思路


douta
Couter_a
clk
doutb
4分频
en4d
Counter_b


clk
en4d
•注意en4d宽度只有一个clk的周期。

选择派生使能方案的原因

- 派生时钟方案增加了全局通道的耗费， FPGA中全局通道比较稀缺。ASIC实现时导致了时钟树耗费增加。
- 派生使能不但可以消除以上的缺点，而且避免引入额外的时钟skew。

注意派生使能的产生

```
Process( clk, reset )
Begin
    if( reset = '1' ) then
      clk_vec <= ( others => '0' );
    elsif( clk'event and clk = '1' ) then
      clk_vec <= clk_vec + '1';
  end if;
End process;

Process( clk_vec )
Begin
    if( clk_vec = "00" ) then
      en4d <= '1';
    else
      en4d <= '0';
    end if;
End process;
```

时钟质量

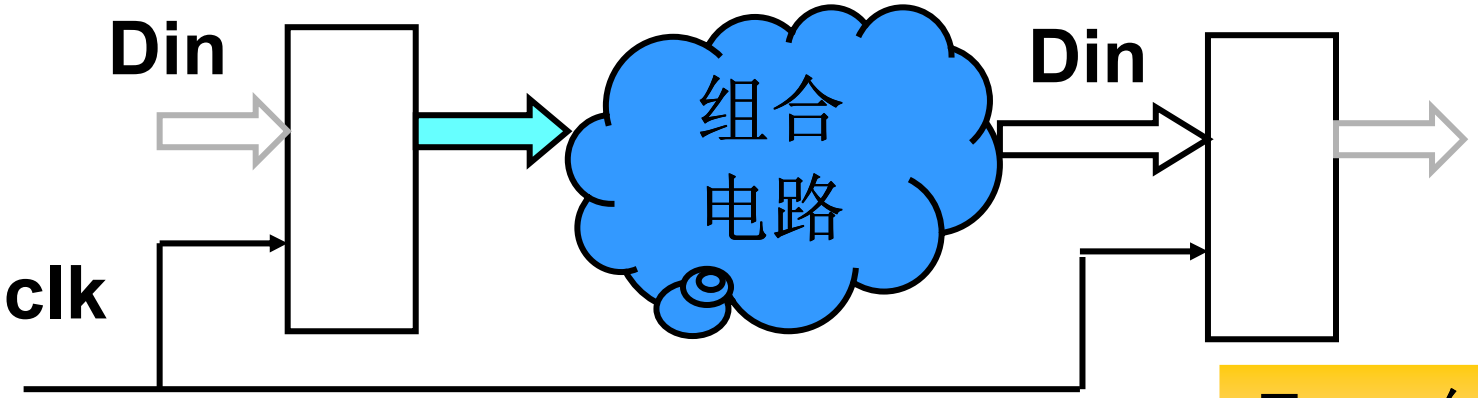
- 时钟的纯净：时钟最好不要通过任何组合电路，或者不要用组合电路产生时钟；
- 时钟的单一：数字系统设计时，应该尽量减少时钟的数目，最好整个系统只有一个时钟；当需要派生时钟时，尽量用派生使能去代替。

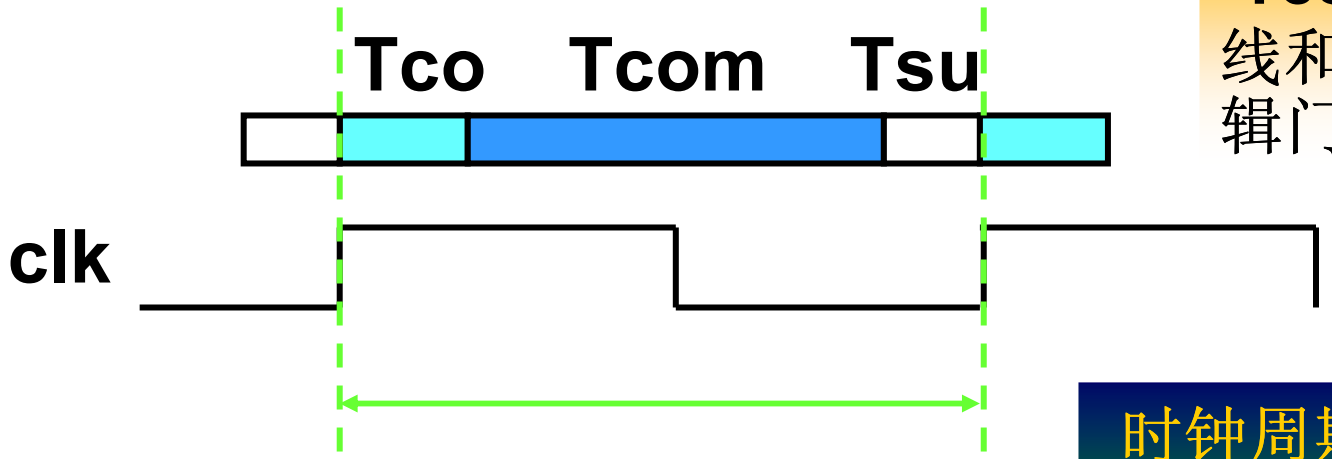
提纲

- 可靠设计
- 同步设计
- 异步设计
- 提升设计性能

- 同步设计的概念
- 时钟质量的保证
- 路径延迟与系统最高工作频率

路径延迟





Tcom包含了走线和组合电路逻辑门的延迟

时钟周期不能小于路径延迟

路径的定义

- 路径：是标识一条逻辑电路链的一系列元素的构成；可以由单个信号网络(net)或一组信号网络加上相关的组合逻辑元件构成；

- 路径从一个pad或者一个同步元件(如触发器,存储单元)的输出端开始，一直到遇到一个pad或一个同步元件的输入端时终止。

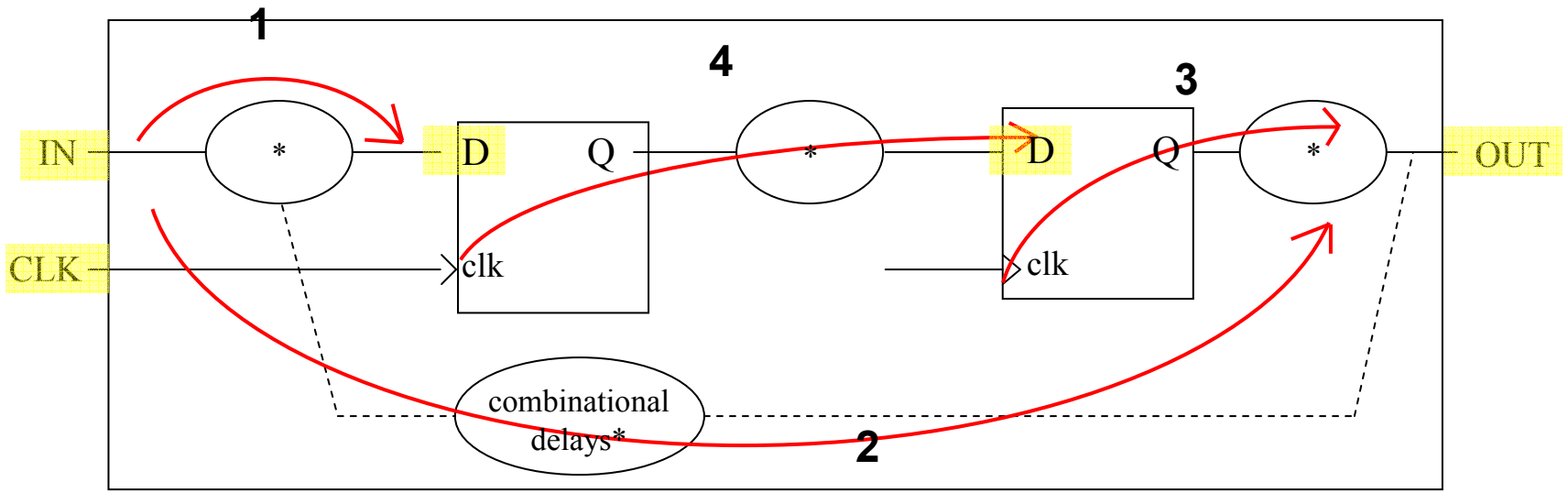
路径的图示

四种可能的路径:

Every path has a **start point** and an **end point**:

Start Points:
- Input ports
- Clock pins

End Points:
- Output ports
- Data input pins of sequential devices



关键路径决定系统最高工作频率

- 一个同步系统中的关键路径，就是它所有的路径中延迟最长的那一条。
- 显然，同步系统的最高工作频率，等于关键路径延迟的倒数。

提纲

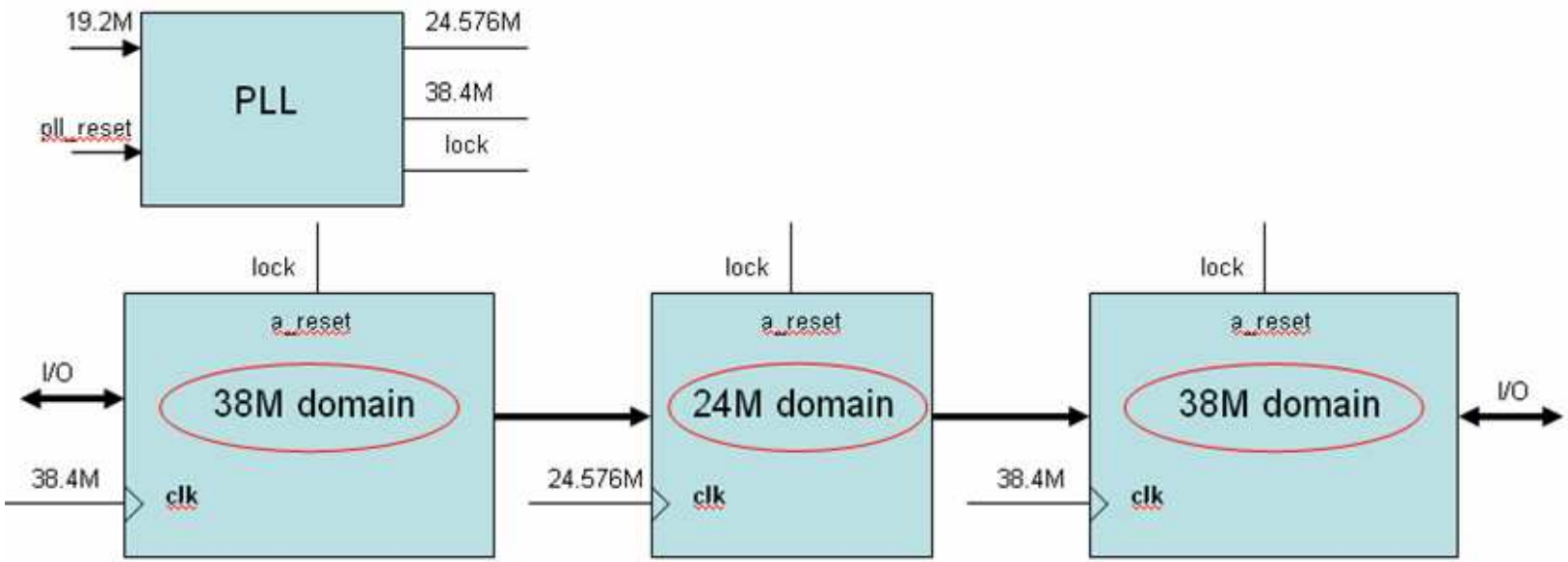
- 可靠设计
- 同步设计
- 异步设计
- 提升设计性能的技巧

异步系统

- 异步多时钟系统：
  - 一个系统中存在多个独立(异步)时钟，并且存在多时钟域(clock domain)之间的信号传输

例子:

数据穿过两个异步时钟域：38.4M和24.576M

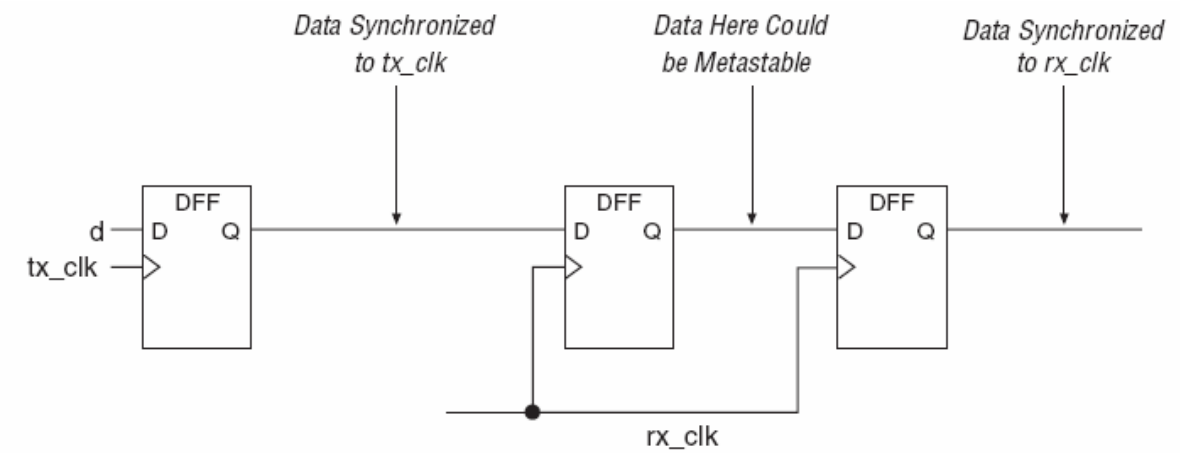


异步时钟域之间交换数据

- 如果两个异步时钟域之间需要交换数据，需要考虑数据交换的可靠性：
- 通常推荐的3个方法：
  - 使用双同步器
  - 使用异步 FIFO(first-in first-out) buffer
  - 使用握手协议 (handshake protocol)

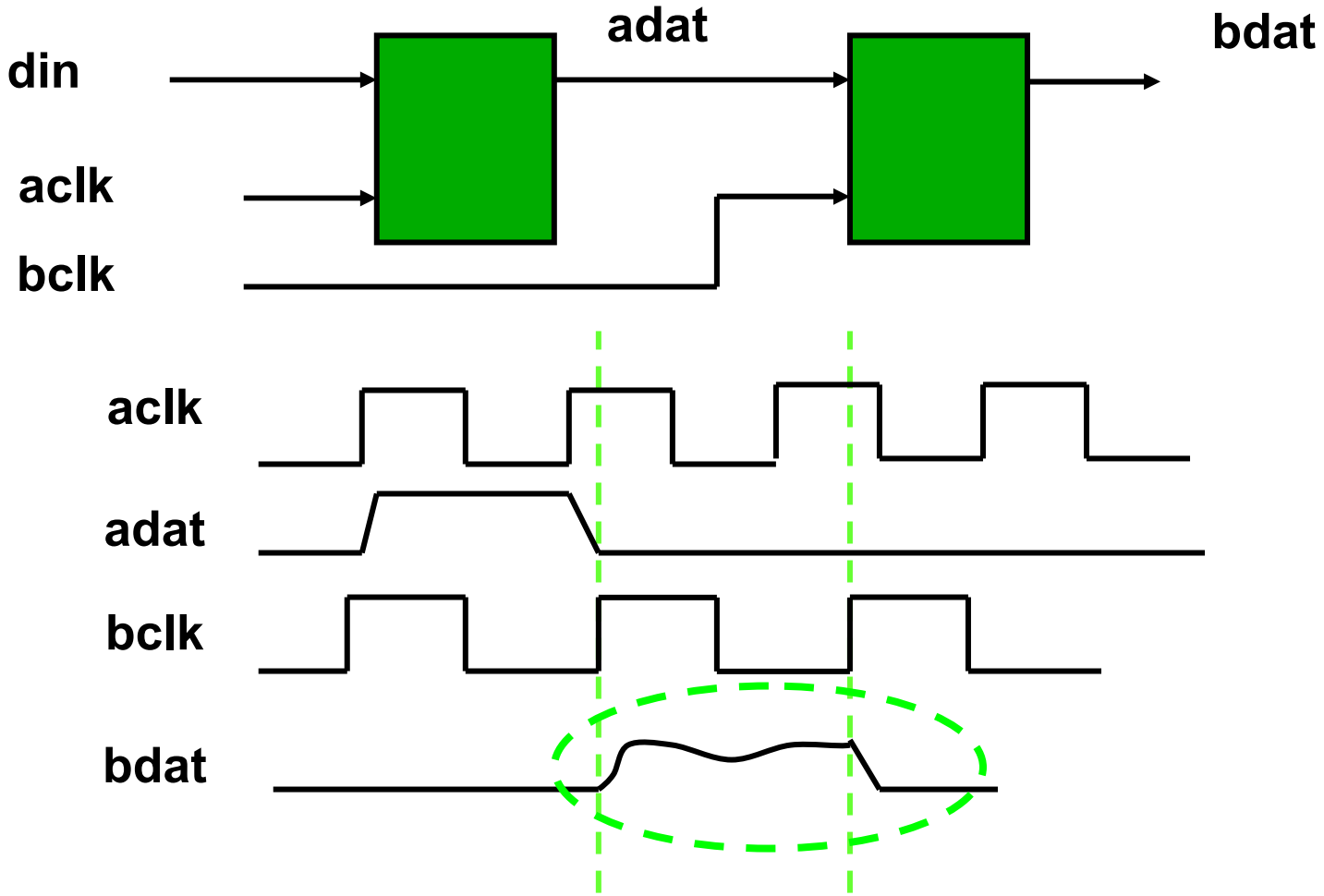
使用双同步器 dbl synchronizer（一）

- 每一根信号线都需要一个双同步器，它由2-bit rx_clk触发的移位寄存器组成.
- 注意tx侧的数据要直接从寄存器输出。
- 第二个移位寄存器 用于减少出现亚稳态(metastability) 的概率（unknown state, for 3.3v TTL:0.8v~2.4v）。 当数据从第一级寄存器输出并传输到第二级时，不定态的概率会大大减小.
- 推荐在单个信号线过时钟域时使用。而总线数据则不行，原因很简单：有些位在第一个时钟沿捕获，而另一些沿在下一级时钟沿才捕获到。
- 也可以使用多于两级的同步电路，更多级同步电路好处是每增加一级，MTBF (mean time between failures)也会相应增加，负面影响是增加了数据的latency..

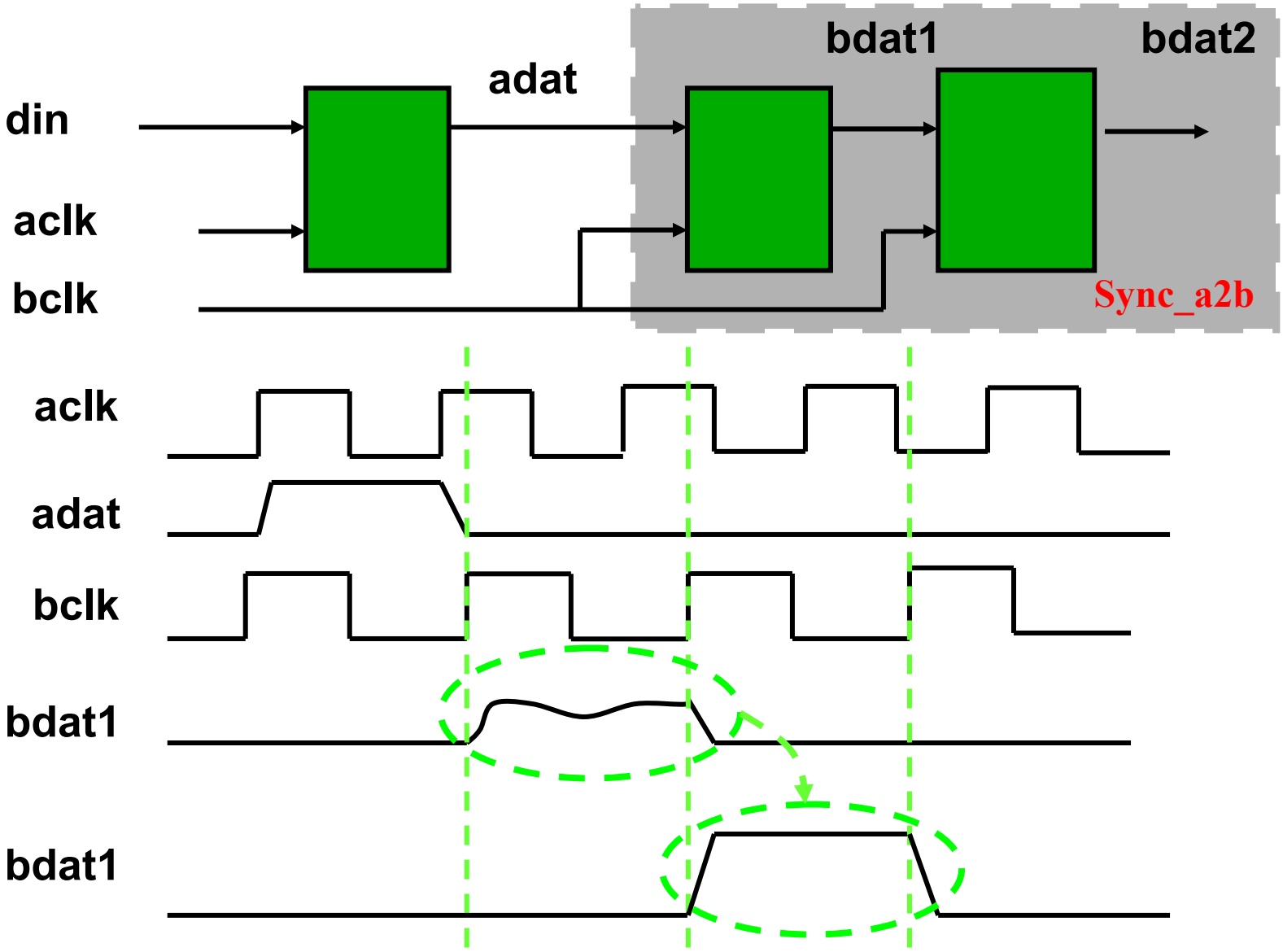


为什么要使用双同步器

如果直接跨时钟域(clock domain) 传输信号，那么电路会出现亚稳态。



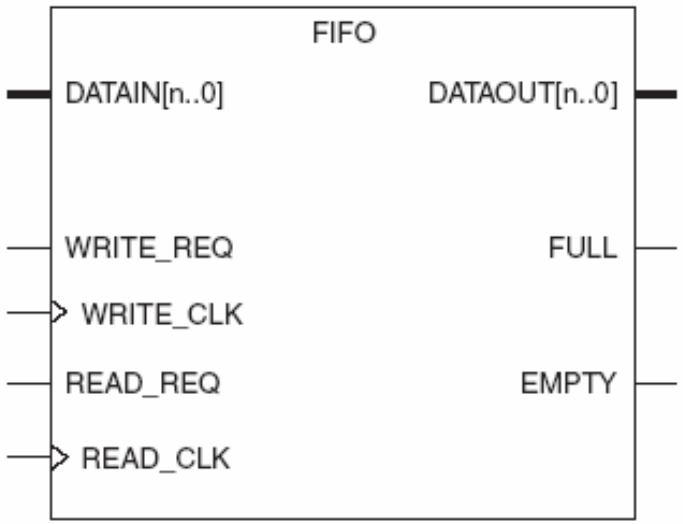
双同步消除亚稳态


bdat1
bdat2
adat
din
aclk
bclk
Sync_a2b
aclk
adat
bclk
bdat1
bdat1

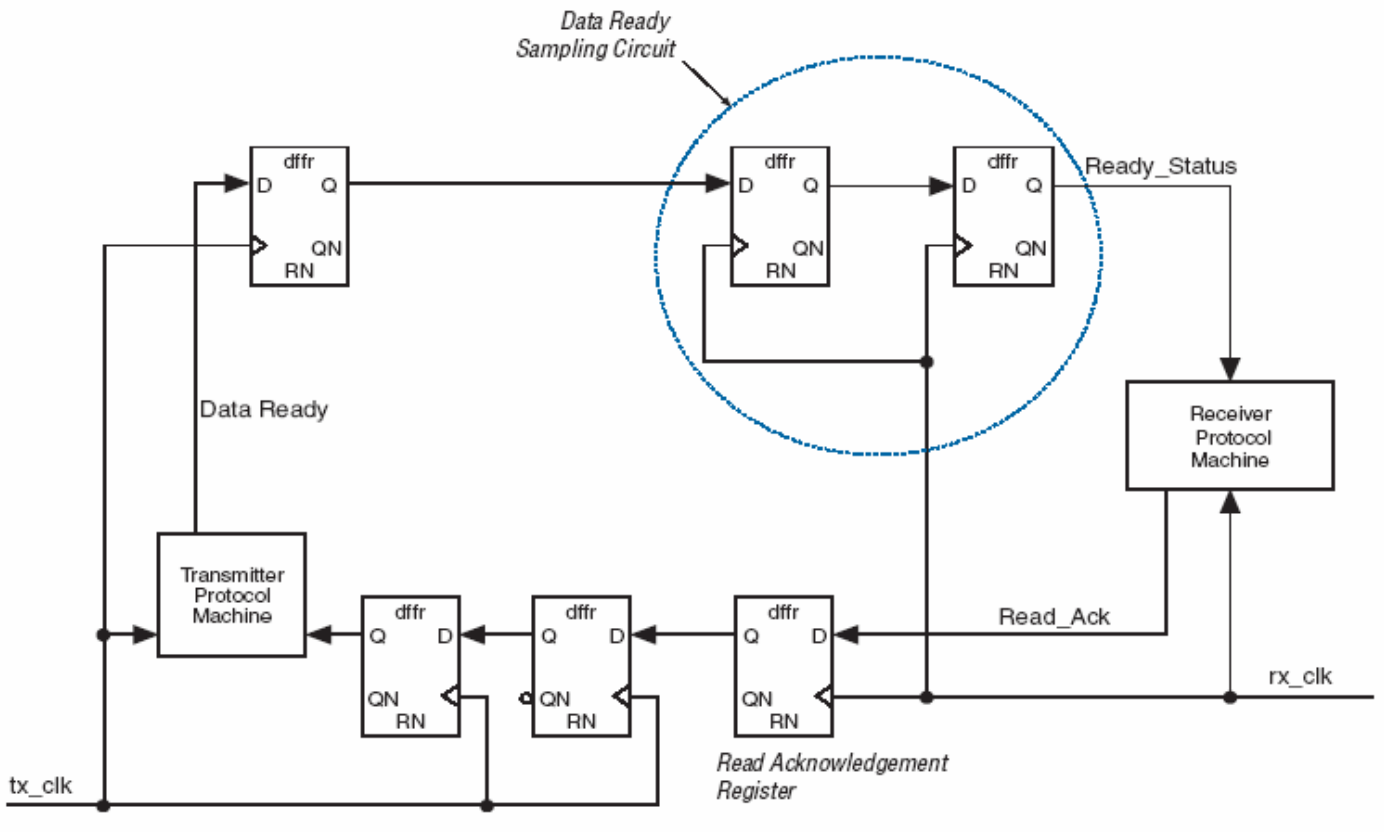
使用 FIFO Buffer(二)

- 当一组数据总线穿过异步时钟域时，采用FIFO buffer是一种有效的手段。
- 采用Altera's MegaWizard® Plug-In Manager可以很容易得设计一个 FIFO buffer.

A FIFO Buffer

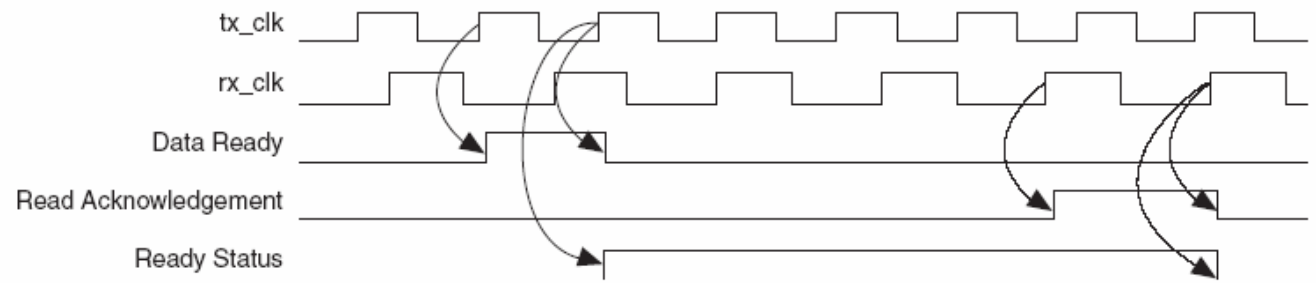


使用握手协议—Handshake Protocol（三）



使用握手协议－Handshake Protocol（续）

- 在没有memory可用做FIFO buffers，而一组数据总线又要穿过异步时钟域时非常有用。用少量的逻辑单元就可实现握手协议，并用来保证数据总线所有的位在穿过异步时钟域后能被同一个Rx_clk 时钟沿采集到。
- tx_clk 先将data ready 信号打为high，表示传送数据已经就绪； data ready采样电路（rx_clk域）采集到后将Ready_Status信号打为 high。
- Data Ready信号必须设计得足够宽以保证接收域采样得到，在 rx_clk 信号比 tx_clk信号慢时特别重要。
- 这时候， rx_clk时钟域就可以从 tx_clk时钟域读取数据. 读操作进行的时候， rx_clk 时钟域就产生一个同步Read_Ack 信号, 它被 read acknowledge register锁存，然后被 tx_clk 时钟域的Read_Ack 采样电路采集.
- Read_Ack信号必须设计得足够宽以保证发送域采样得到，在 tx_clk 信号比 rx_clk信号慢时特别重要。
- 这样一来就完成了2个异步时钟域的数据传诵。波形参见下图。



推荐：用不同时钟后缀对信号和模块命名

- 信号命名和模块划分的优点：
  - 有利于检查信号所通过的时钟域；
  - 有利于各模块进行单独的静态时序分析；
  - 有利于在静态时序分析中快速地设定false path；

异步设计小结

- 注意使用同步化电路来对异步信号进行同步；
- 进行科学的模块划分和信号命名；
- 尽量减少握手控制信号的数目，以避免同步化造成的信号拉伸而破坏控制信号之间的相位关系；
- 快时钟域信号进入慢时钟域时，要注意信号丢失的避免和检测；

提纲

- 可靠设计
- 同步设计
- 异步设计
- 提升设计性能

长路径的避免
串行运算转并行运算
优先级电路的延时优化
数据通路拷贝
数据运算式变换
变量运算优化
组合路径切割
资源共享
合理选择专有硬件资源
优化关键路径

延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

1. 长路径的避免

- 在设计中，时序能走短路径就尽量走短路径；组合电路能缩小就尽量缩小。以下举例来说明。

例子一：状态机

设计中状态编码采用Binary编码和One-hot编码对系统性能会造成什么样的影响？这两种编码对FPGA/CPLD的适用情况如何？

	Binary	One-hot
S0	00	0001
S1	01	0010
S2	10	0100
S3	11	1000

- one-hot编码方式只用一个bit来表示一个状态，缩小了状态译码的组合电路规模，使得路径延时更小，因此状态机的时钟可以运行在更高的频率上。
- One-hot因为寄存器消耗量比较大，所以往往用在寄存器资源比较丰富的FPGA中；
- 传统的CPLD中寄存器资源较少，使用时需要综合考虑。

减少组合逻辑的扇入

- 扇入系数与组合规模
  - 组合电路的规模往往受影响于其扇入系数：
  - 扇入↑ → 组合电路规模↑ → 从而路径延时 → ↑系统工作速率↓

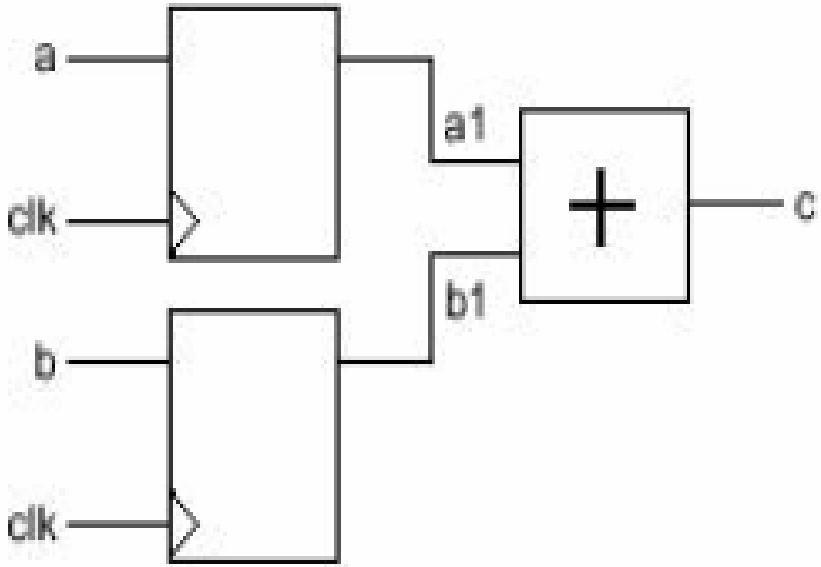
- 减少组合逻辑的扇入是提高系统工作速率的基本手段。

利用电路的等价性巧妙的分配延迟

- 在功能等价的情况下，我们可以根据时序需要安排组合逻辑电路在寄存器前后的位置，合理分配延时。
- 其实质为将关键路径中的部分延迟转移到其他非关键路径之上。

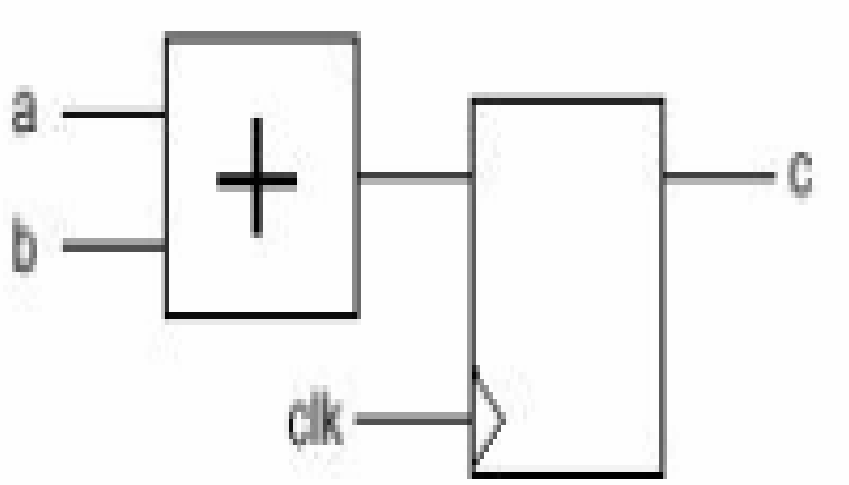
组合逻辑在寄存器之后

- 假定a b信号的延时非常大



组合逻辑在寄存器之前

- 假定a b信号的延时并不大，而寄存器c信号经过的逻辑比较多，延时大



延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

串行运算转并行运算

- 常用手段有：
  1. 使用Case语句代替IF语句
  2. cascade → Tree ；

IF和Case语句

- If语句指定了一个优先级编码逻辑，case语句生成的是并行逻辑，不具优先级；
- If语句包含不同的表达式，case语句比较的是一个公共的控制表达式；
- 一般情况下，case结构速度较快.
- 不正确使用嵌套的if语句会导致设计需要更大的延迟。

用if语句实现8选1多路选择器

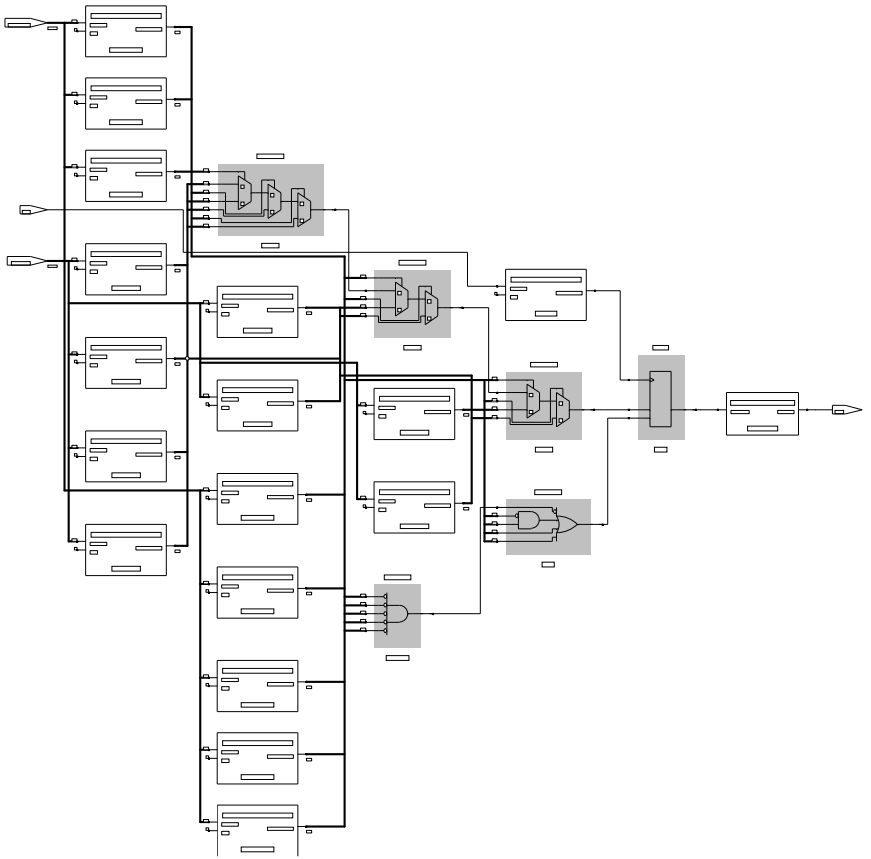
```
always @(sel or in)
begin
   if(sel == 3'b000)          out = in[0];
   else if(sel == 3'b001)     out = in[1];
   else if(sel == 3'b010)     out = in[2];
   else if(sel == 3'b011)     out = in[3];
   else if(sel == 3'b100)     out = in[4];
   else if(sel == 3'b101)     out = in[5];
   else if(sel == 3'b110)     out = in[6];
   else                       out = in[7];
end
```

用case语句实现8选1多路选择器

```
always @(sel or in) /* synthesis full_case */
begin
   case(sel)
     3'b000:              out = in[0];
     3'b001:              out = in[1];
     3'b010:              out = in[2];
     3'b011:              out = in[3];
     3'b100:              out = in[4];
     3'b101:              out = in[5];
     3'b110:              out = in[6];
     default:             out = in[7];
   endcase
end
```

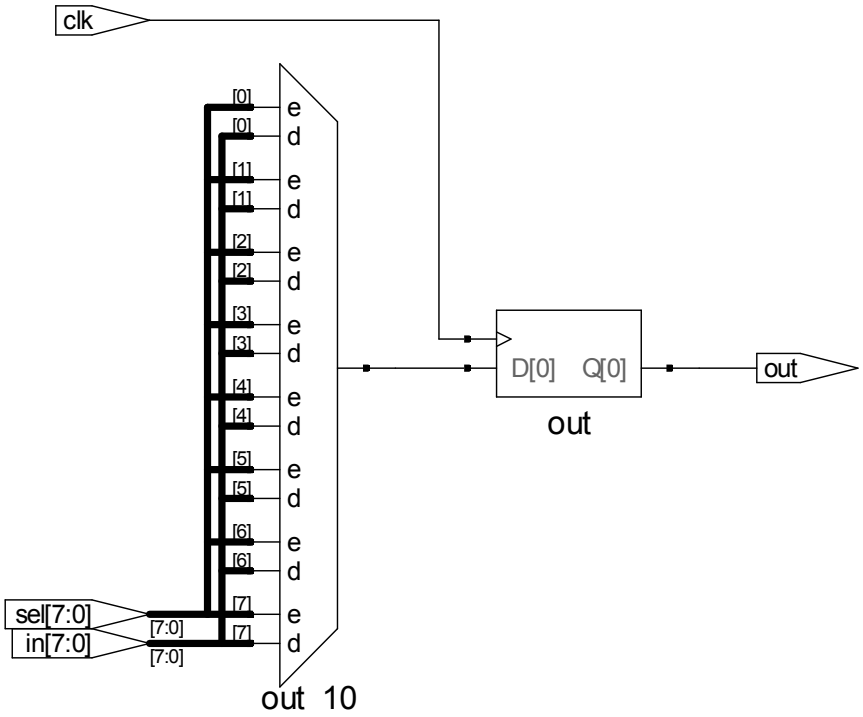
在EP2S15F484C4中得到的最终结果——if语句

- 关键路径延迟为4.830ns；
- 使用了1个7输入ALUT，和4个5输入ALUT；
- 共占用了4个ALM；



在EP2S15F484C4中得到的最终结果——case语句

- 关键路径延迟为4.462ns；
- 使用了1个6输入ALUT，1个5输入ALUT，2个4输入ALUT；
- 占用了3个ALM；



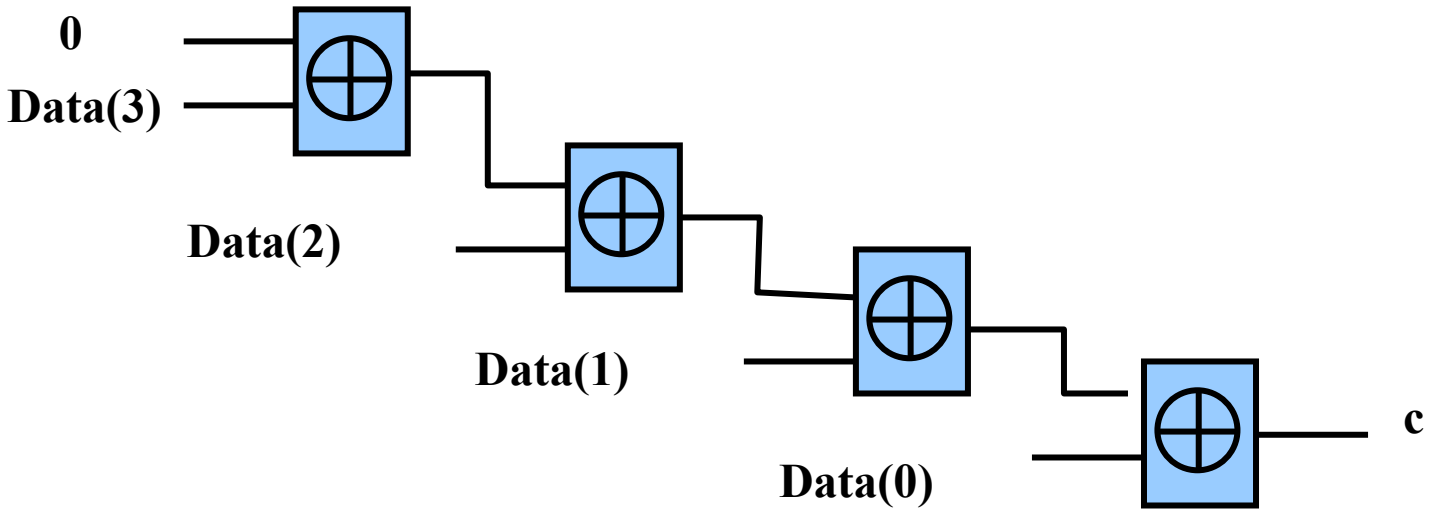
Cascade → Tree

例子一： for循环的优化

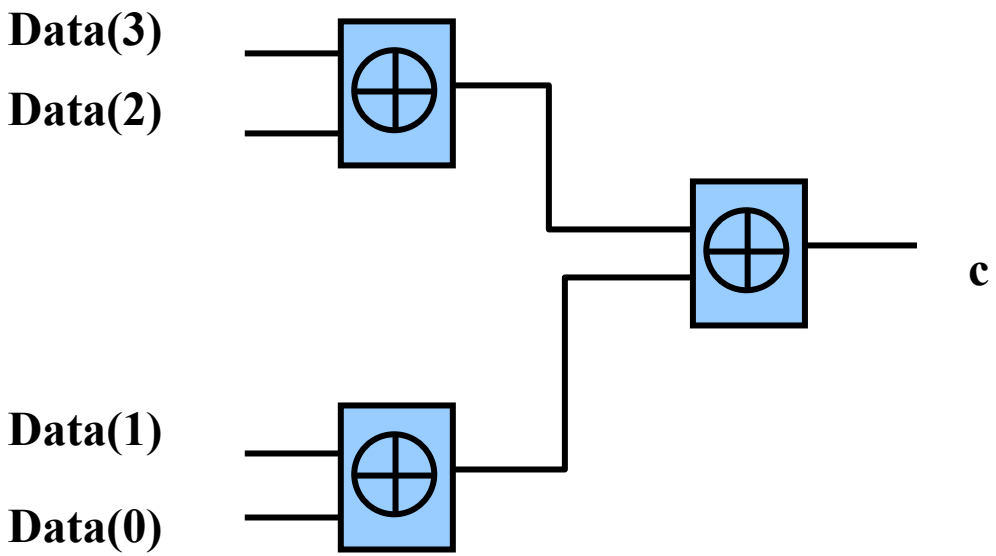
- 设计奇偶校验位生成电路：对数据data(3:0)生成校验位c。
- 一种思路如下：

```
result := '0';
   for I in data'RANGE loop
   result := result XOR data(I);
   end loop;
C <= result;
```

综合后的电路图



改进思路(目标电路图)



改进的描述方法

```
Sig1 <= Data(3) xor Data(2);
Sig2 <= Data(1) xor Data(0);
C <= sig1 xor sig2;
```

经验：在串转并中，适当地使用平衡的二叉树结构，不但但可以减少延时，而且有利于方便地引入流水线结构。

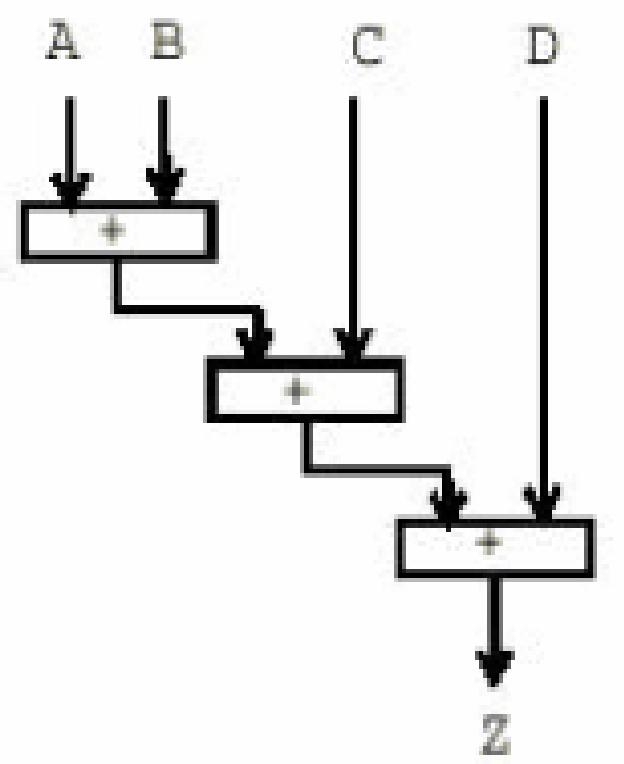
cascade → Tree

例子二：使用圆括号处理多个加法器

- 控制设计结构的另一种方式是使用圆括号来控制逻辑分组
- 下面的例子描述了一个4输入加法器分组及其实现结果

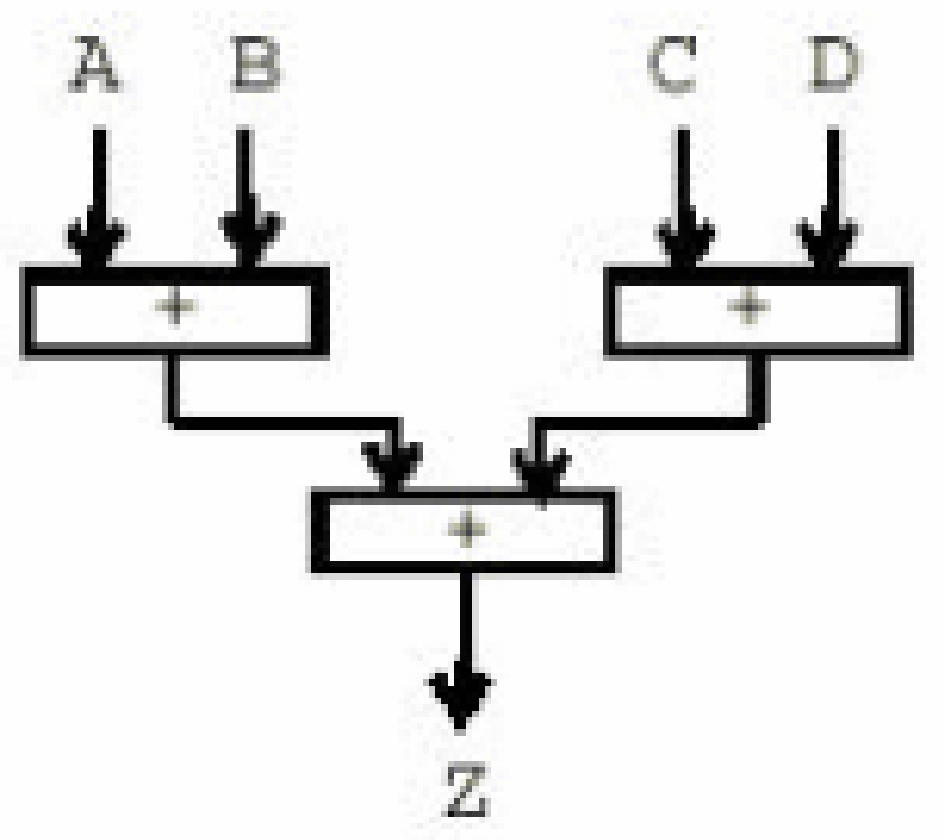
串行加法器

assign z = a + b + c + d;



用圆括号重新构建的加法器

```
assign z = (a + b) + (c + d);
```



结果分析

- 第一种方法面积小，但整体速度慢。但是，如果信号D是关键路径，其他信号是非关键路径；或者，设计中的关键路径与A,B,C,D无关则应采用这种方法；
- 第二种方法面积大，但整体速度快。如果对A,B,C,D的时序要求都比较苛刻，应当采用这种方法。

延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

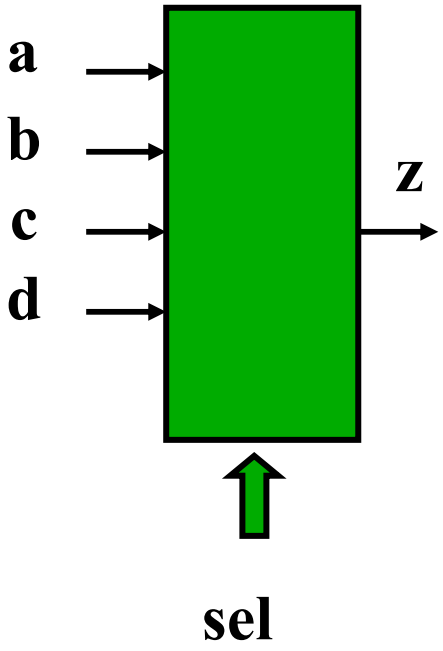
优化时序的设计调整

- 设计时，往往需要针对具体的情况来进行设计的调整，以使系统的时序得到优化。
- 时序优化的原则，最主要的还是在保证正确逻辑的基础上，尽量缩小关键路径的延迟。

例：带优先级的多路选择器

假设该选择器的真值表如下：

Sel(0)	Sel(1)	Sel(2)	Sel(3)	z
x	x	x	1	d
x	x	1	0	c
x	1	0	0	b
1	0	0	0	a
0	0	0	0	0



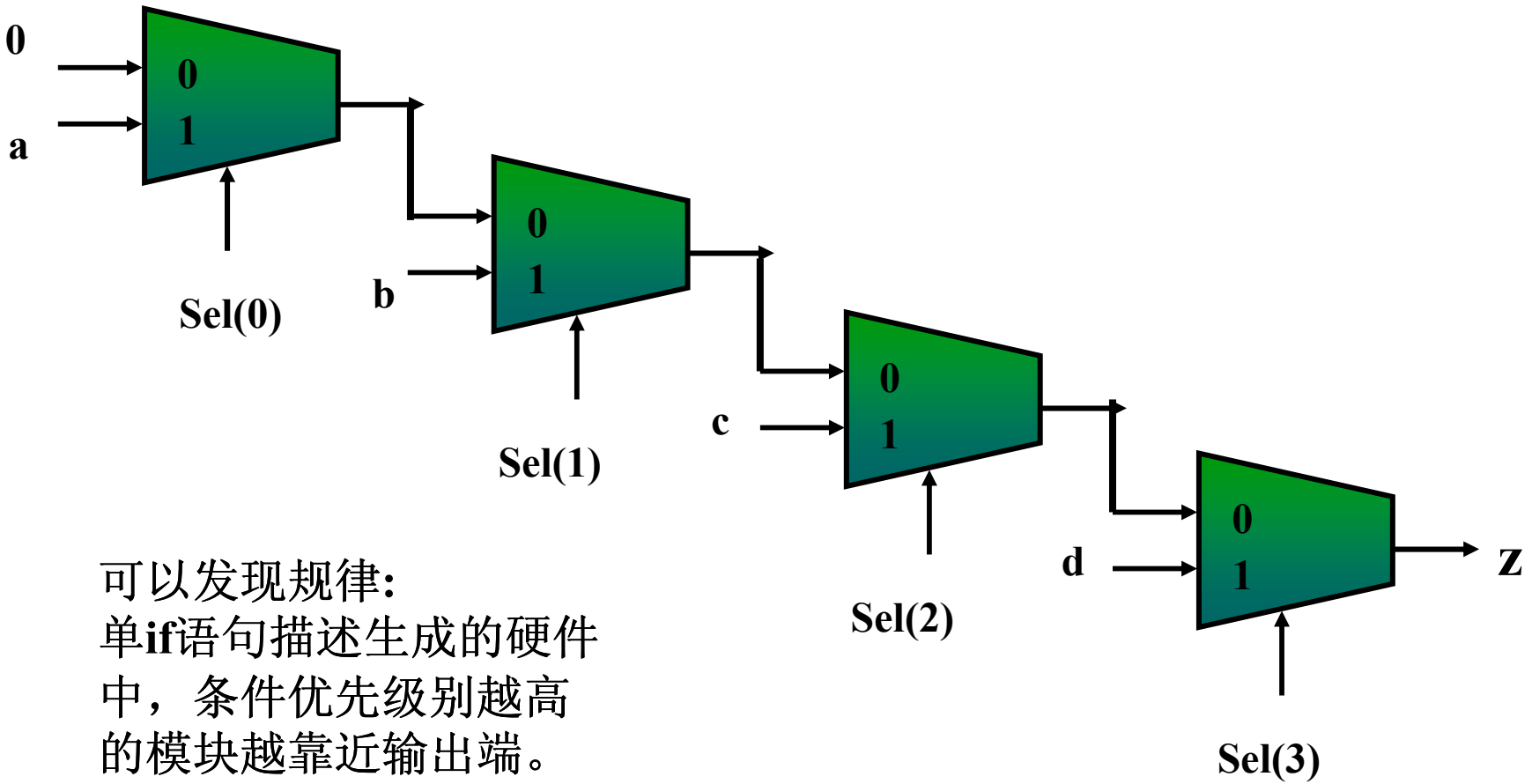
描述方法1：多if语句

```
Process(a, b, c, d, sel)
Begin
    z <= '0';  -- 初始化
    if( sel(0) = '1' ) then  z <= a; end if;
    if( sel(1) = '1' ) then  z <= b; end if;
    if( sel(2) = '1' ) then  z <= c; end if;
    if( sel(3) = '1' ) then  z <= d; end if;
End process;
```

描述方法2：单if语句

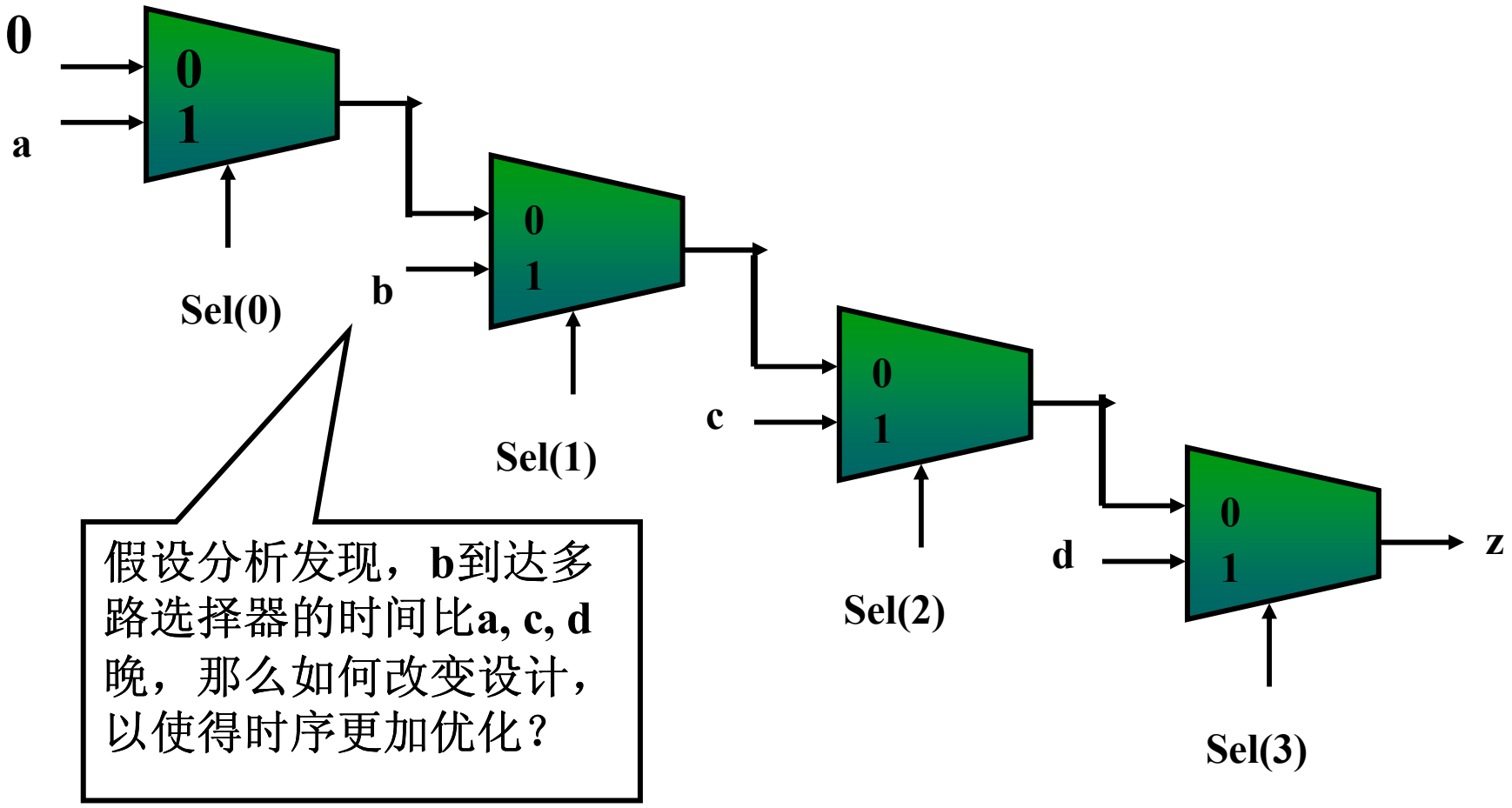
```
Process(a, b, c, d, sel)
Begin
    if    ( sel(3) = ‘1’ ) then  z <= d;
    elsif( sel(2) = ‘1’ ) then  z <= c;
    elsif( sel(1) = ‘1’ ) then  z <= b;
    elsif( sel(0) = ‘1’ ) then  z <= a;
    else                                z <= ‘0’;
    end if;
End process;
```

综合后的电路图

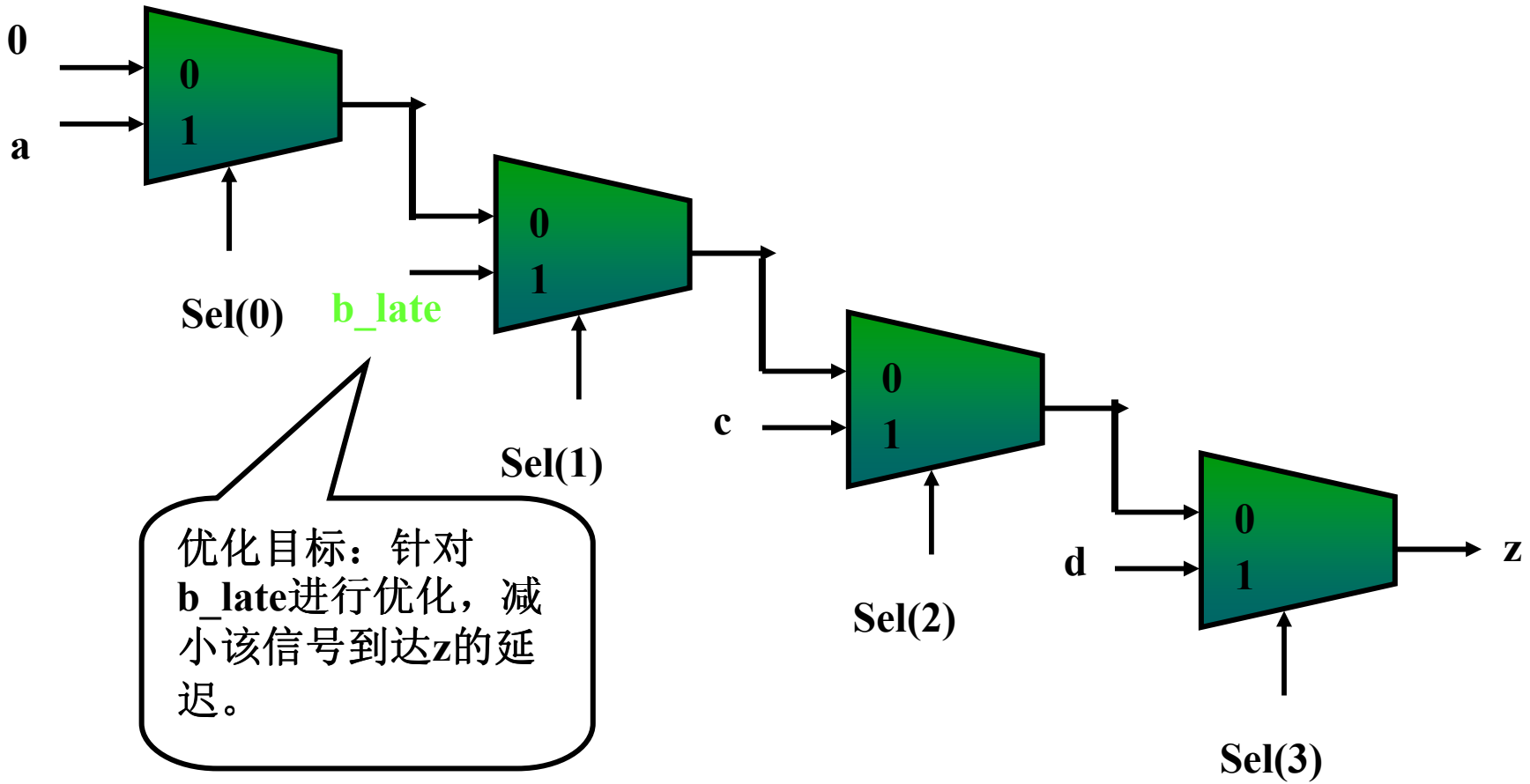


可以发现规律:
单if语句描述生成的硬件中，条件优先级别越高的模块越靠近输出端。

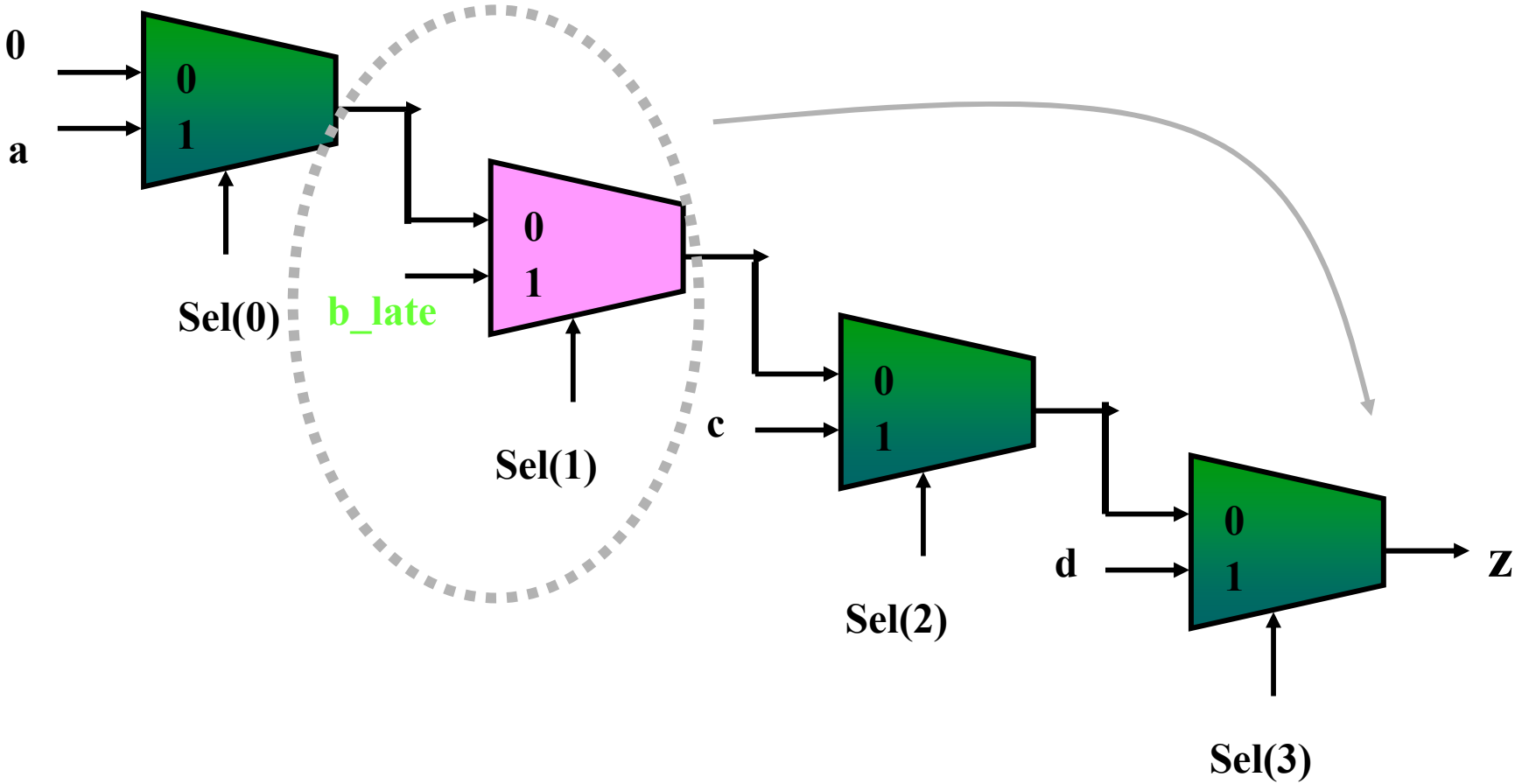
调整情况1: 数据到达延迟是(b ->b_late)



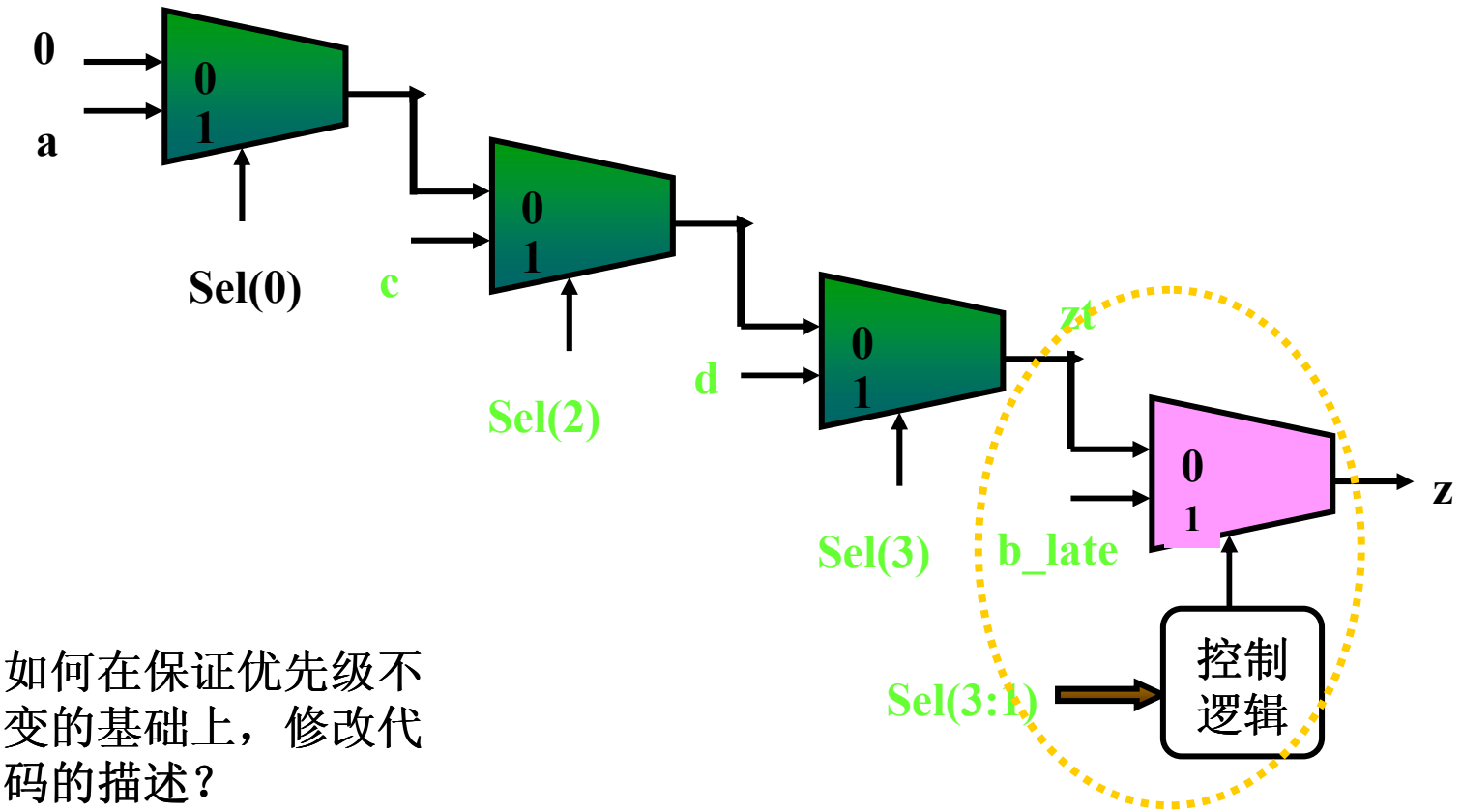
优化目标


0
a
0
1
Sel(0)
b_late
0
1
Sel(1)
c
0
1
Sel(2)
d
0
1
Sel(3)
z
优化目标：针对b_late进行优化，减小该信号到达z的延迟。

优化思路



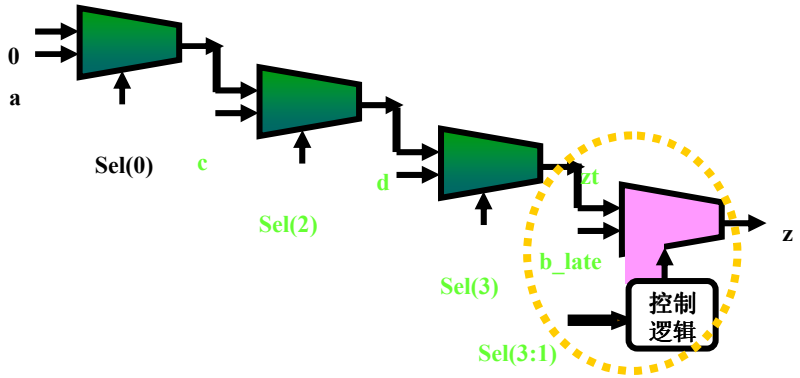
优化后的电路图



如何在保证优先级不变的基础上，修改代码的描述？

优化描述方法1：多if语句

```
Process( a, b, c, d, sel)
Begin
    zt <= '0';
    if( sel(0) = '1' ) then zt <=  a; end if;
    if( sel(2) = '1' ) then zt <=  c; end if;
    if( sel(3) = '1' ) then zt <=  d; end if;
    if( ( sel(1) = '1' ) and ( sel(2) = '0' ) and ( sel(3) = '0') )  then
          z <= b_late;
    else
          z <= zt;
    end if;
End process;
```



优化描述方法2：双if语句

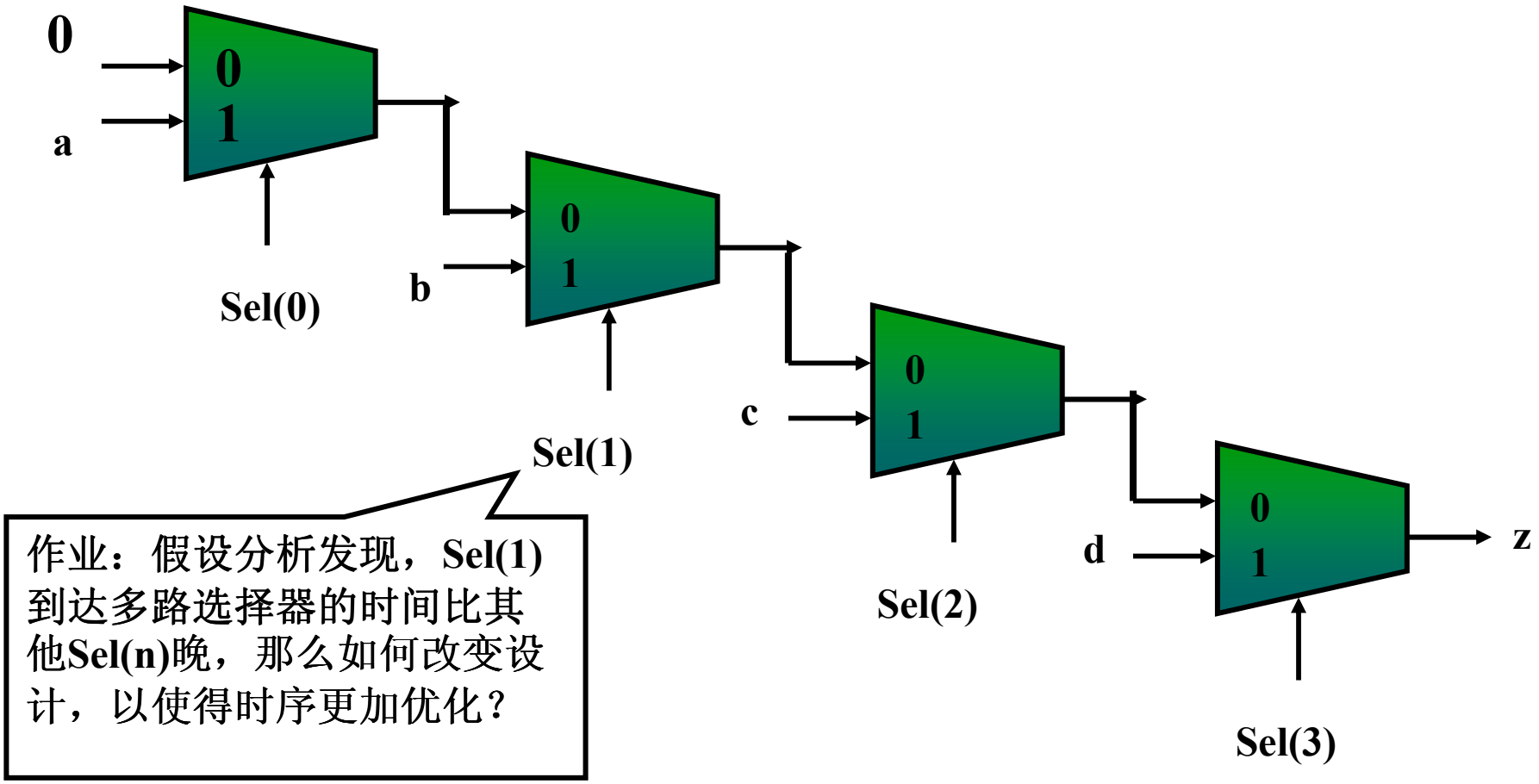
```
Process( a, b, c, d, sel)
Begin
   if    ( sel(3) = '1' ) then zt <= d;
   elsif( sel(2) = '1' ) then zt <= c;
   elsif( sel(0) = '1' ) then zt <= a;
   else                       zt <= 0;
   end if;

   if ( ( sel(1) = '1' ) and ( sel(2) = '0' ) and (sel(3) = '0' ) then
    z <= b_late;
   else                    z <= zt;
   end if;
End process;
```

方法分析

- 方法1和方法2实际上是等效的，都可以生成前述的优化电路。

- 一般来说，代码描述的风格不要太抽象化，应该遵循硬件与代码模块一一对应的模块化原则，这样有利于综合器生成与直观符合的硬件电路。

调整情况2：控制到达延迟
(Sel(1) -> Sel(1)_late)



作业：假设分析发现，Sel(1)到达多路选择器的时间比其他Sel(n)晚，那么如何改变设计，以使得时序更加优化？

带优先级的电路优化方法总结

- 对于单if语句描述(single if statement OR multiple if statement)，生成的硬件都是越高优先级条件判断对应的模块越靠近输出端。

- 如果到达比较晚(即延时较大)的信号(不管是data信号还是control信号)所对应的硬件模块，并不是处于最高优先级处时，那么就应该想办法将该模块向靠近输出端的位置搬移，以减小该信号所在的路径(很可能成为关键路径)的延迟。
- 可以通过搬移if条件模块的方法搬移需要移动的模块；
- 搬移之后，要调整if条件判断式，保证原来的优先级不变。

延时优化的几个要点

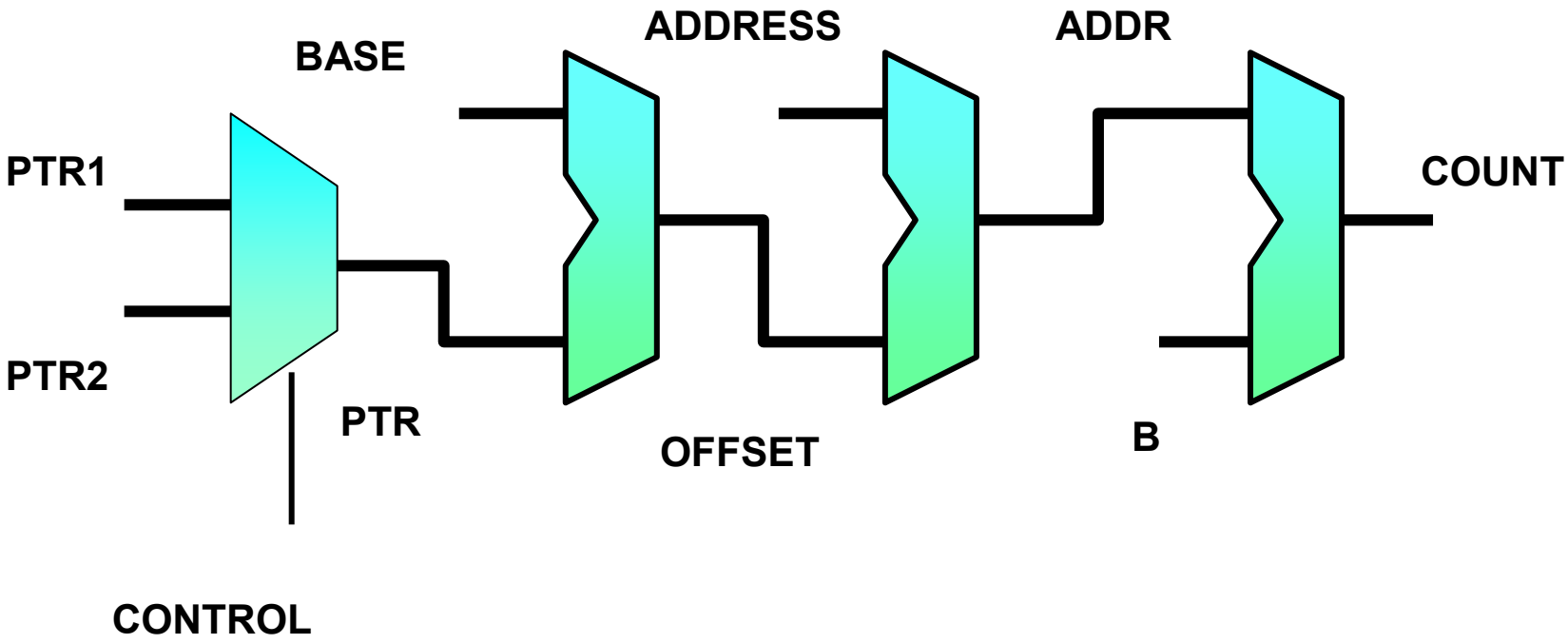
1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

例：数据通路复用

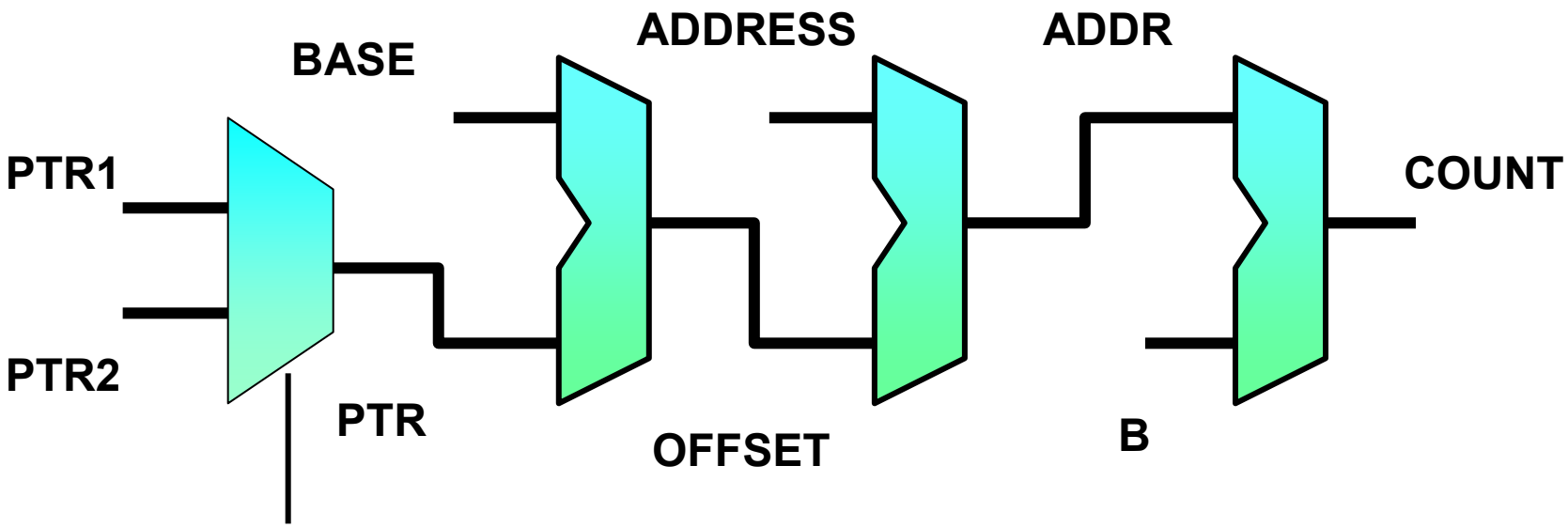
考虑下面的代码：

```
If( CONTROL = ’1’ )then
        PTR := PTR1;
else
        PTR := PTR2;
end if;
OFFSET := BASE - PTR;
ADDR := ADDRESS - OFFSET;
COUNT <= ADDR + B;
```

综合后的电路图

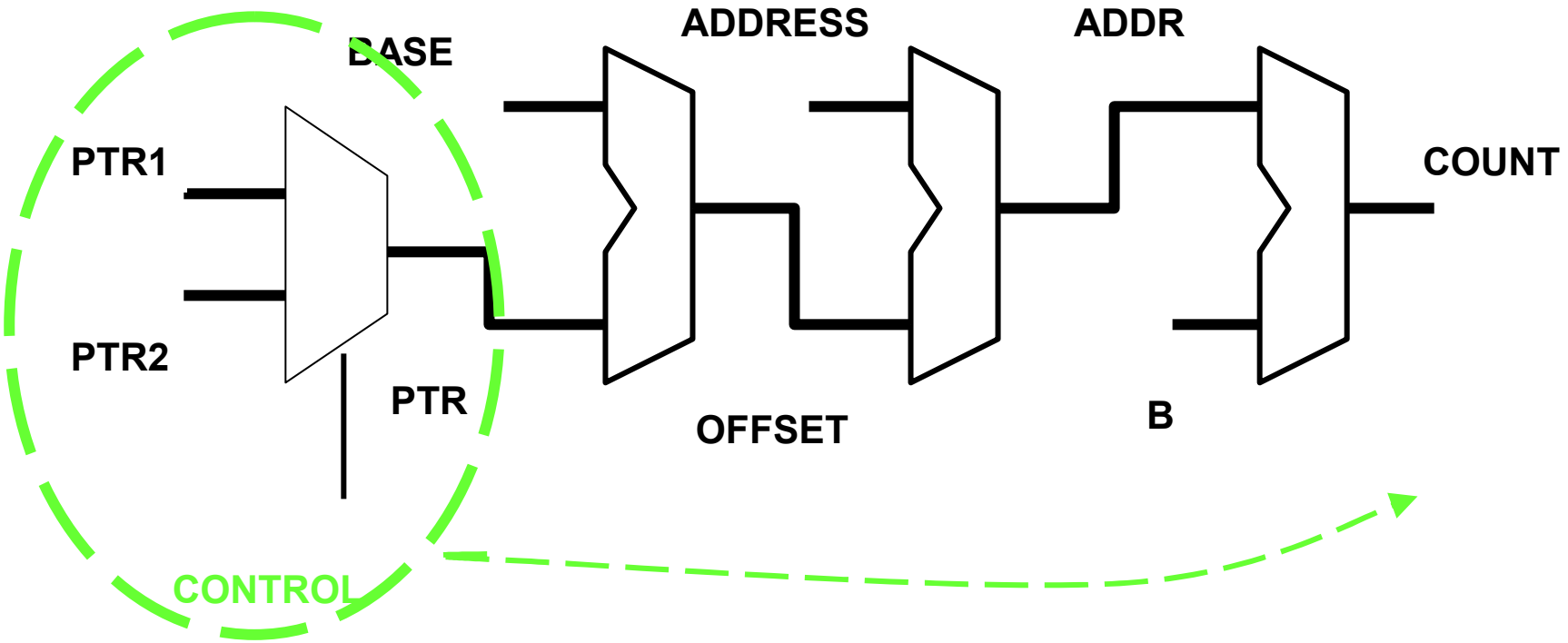


优化目标

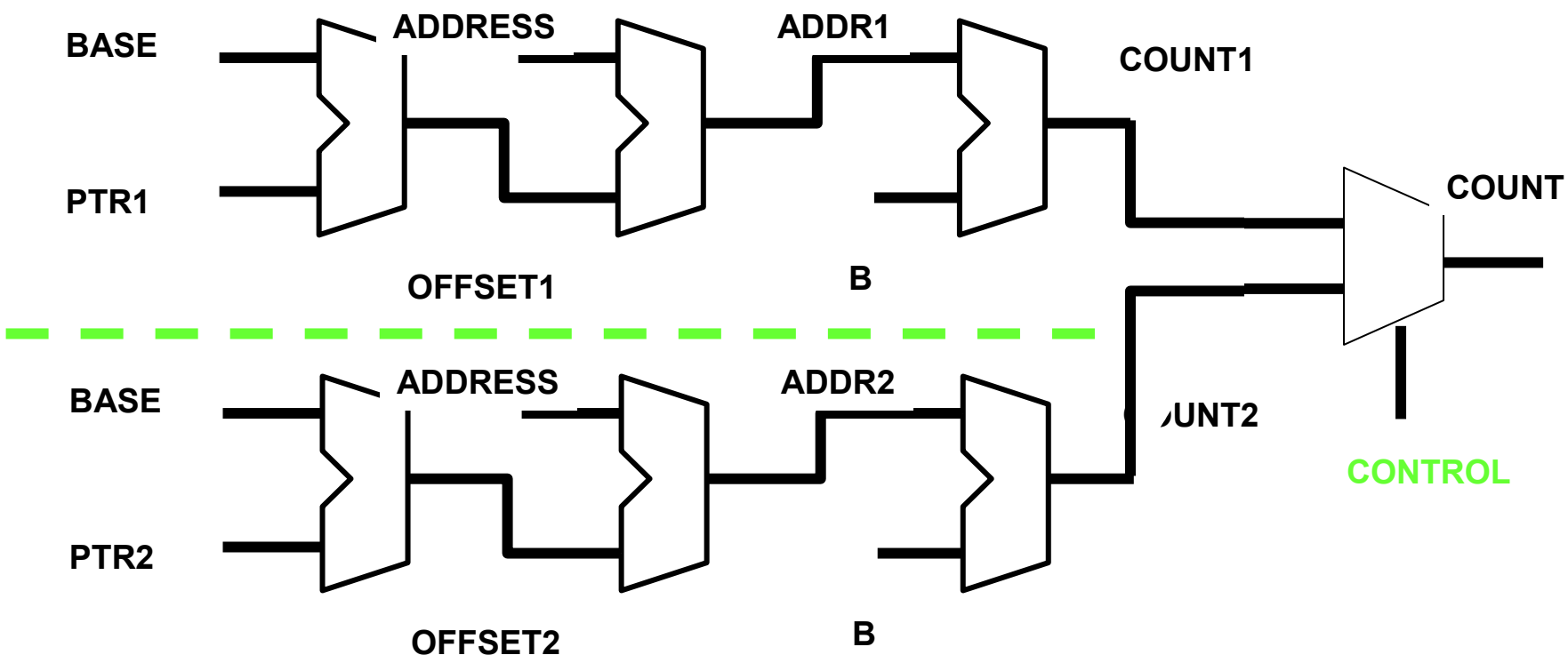


假设CONTROL是一个late arriving input signal。现在要对该信号进行优化，减小该信号到达COUNT的延时。

优化思路



优化后的电路图


BASE
ADDRESS
ADDR1
COUNT1
PTR1
COUNT
OFFSET1
B
BASE
ADDRESS
ADDR2
UNT2
CONTROL
PTR2
OFFSET2
B

性能对比

可见，逻辑复用方法在提高时序性能的同时，引起资源耗费增大。

	Data Arrival Time	Area
复用前：	5.23	1057
数据通路复用方法：	2.33	1622

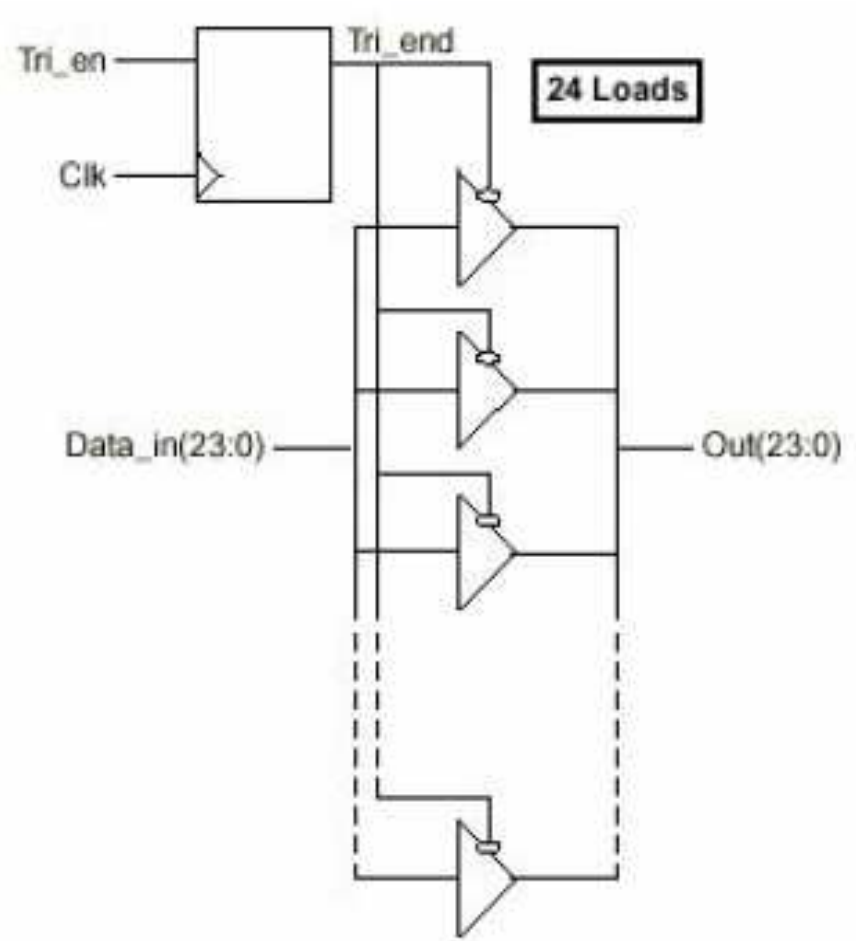
注：这里的Data Arrival Time指的是CONTROL到达输出端COUNT的延时。

复制电路减少扇出

- 提高关键路径速度的一个常用方法是复制电路，减少关键路径的扇出
- 当一个信号网络所带负载增加时，其路径延时也相应增加
- 扇出对关键路径延时的影响有时甚至超过了逻辑的级延时

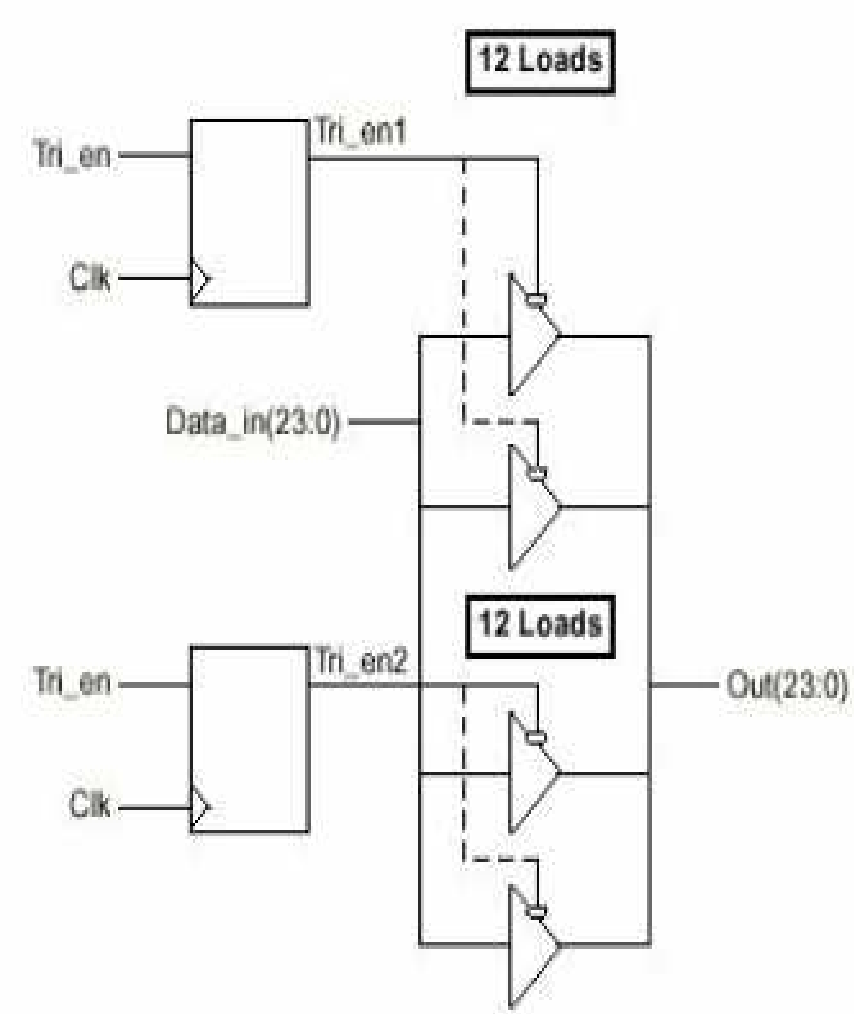
寄存器扇出为24

- 最大的Tco为tri_end到out之间，为9.989ns



每个寄存器扇出为12

- 最大Tco为tri_en2到out之间，为8.125ns



延时优化的几个要点

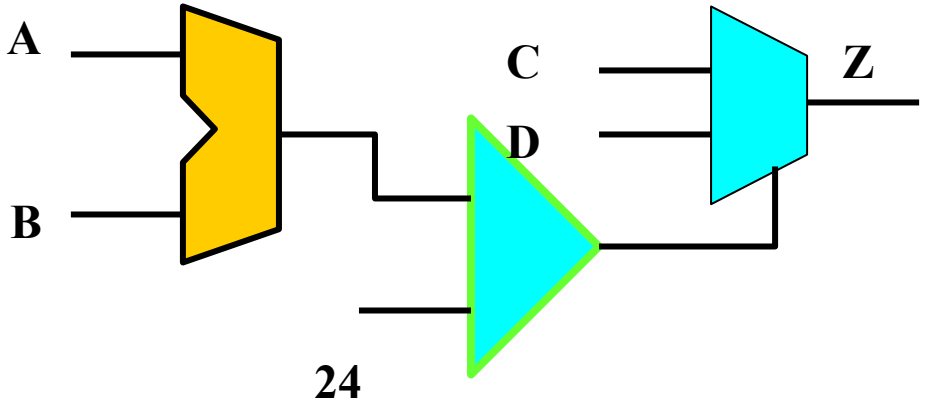
1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

例：if条件判断式中的数据运算

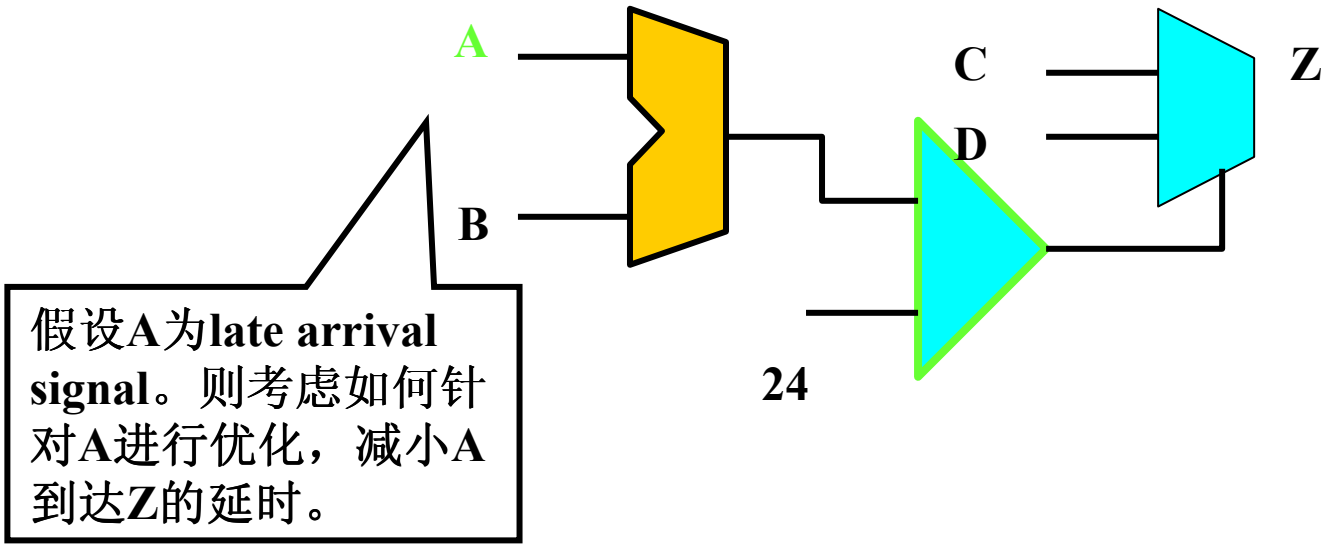
考虑下面的代码：

```
if (A + B < 24) then
      Z <= C;
else
      Z <= D;
end if;
```

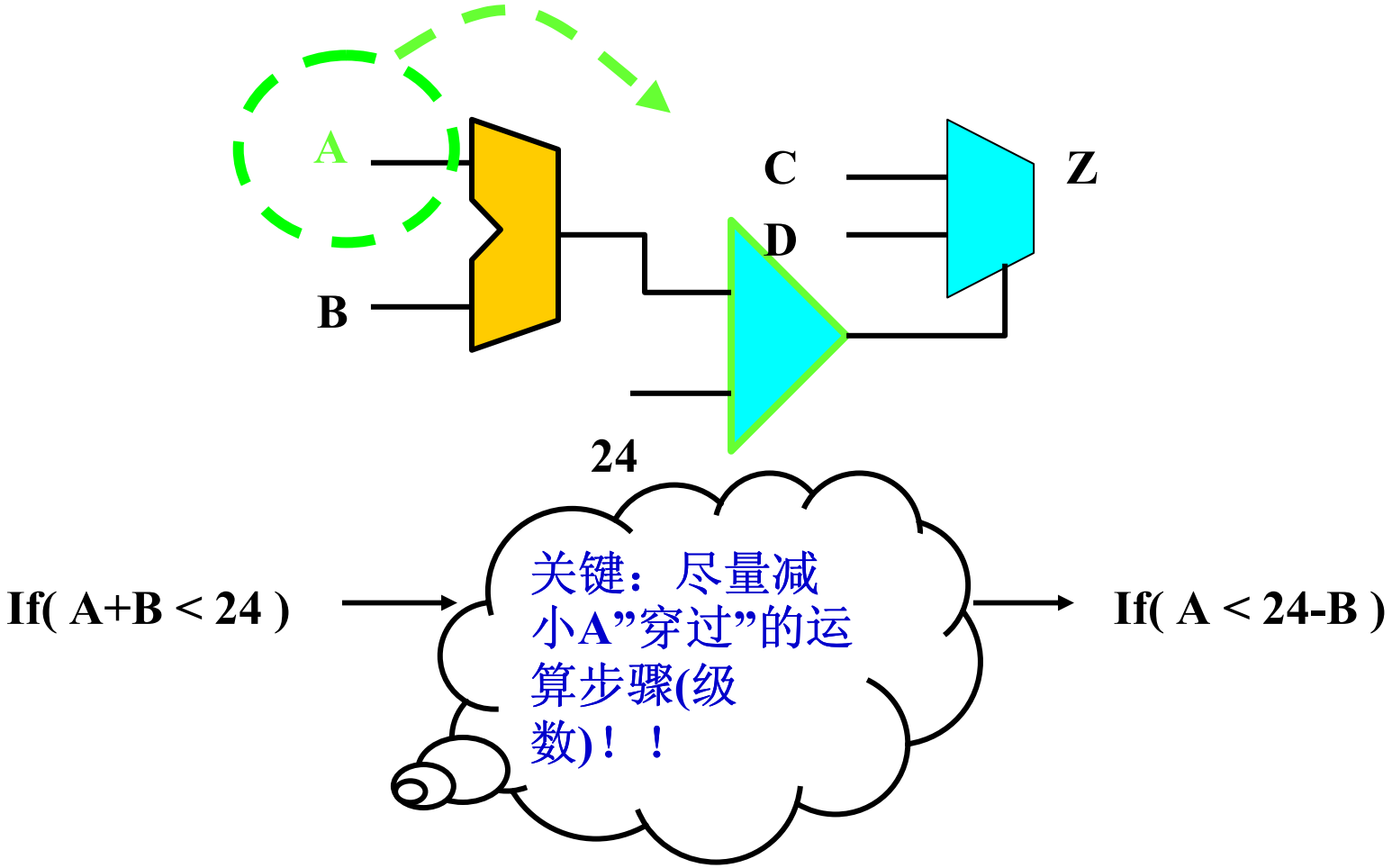
综合后的电路图如下：



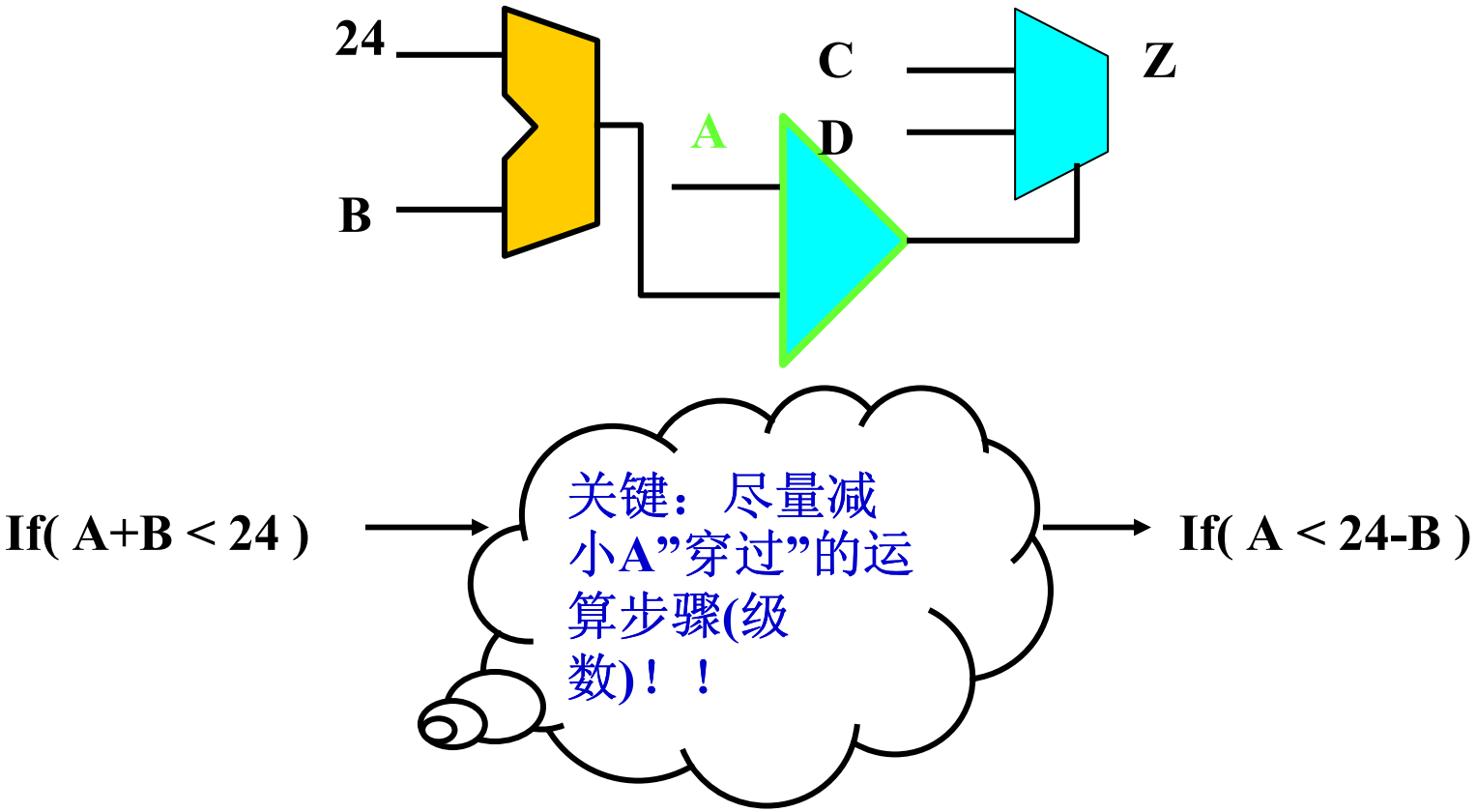
优化目标



优化思路



优化思路



优化后的代码

```
if (A < 24 - B) then
      Z <= C;
else
      Z <= D;
end if;
```

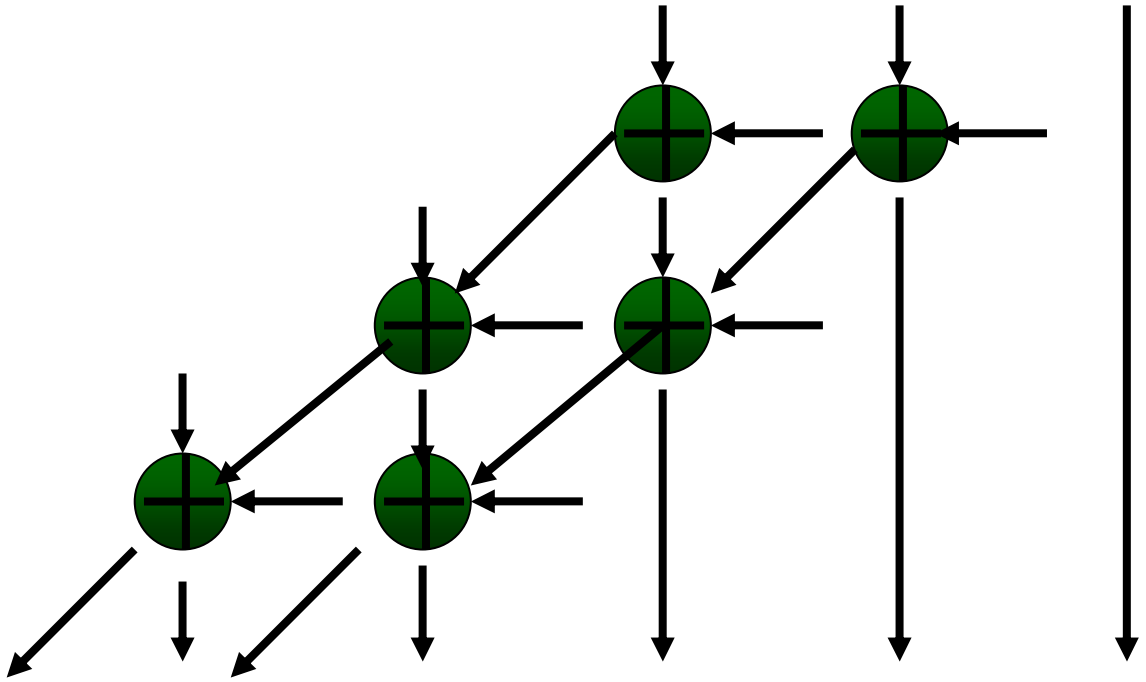
延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

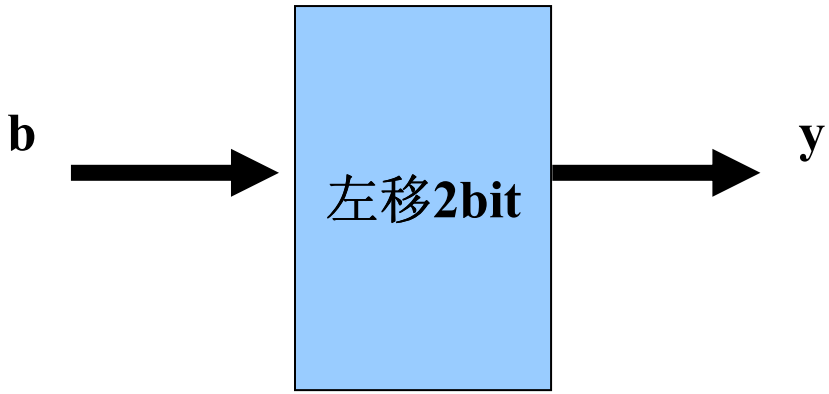
- 注意这里所说的“变量”指的是 “非常量”，其包含VHDL中的“变量”和“信号”。
- 变量计算和常量计算的综合结果在逻辑规模上差别极大。

例：变量乘法与常量乘法

- 假设a, b均为3bit信号(变量)，则y <= a * b的结果可能为(假设a, b，y均为无符号数):



- 假如在硬件运行的过程中，a始终为一个常数100，则可以将代码改成 y <= b * “100”。此时综合得到的电路为：



- 可见，如果能够用常量运算，就尽量用常量运算表达式。误用变量表达式虽然在逻辑上并没有错误，但是会导致组合电路规模过大，不但浪费芯片资源，而且降低系统工作速率。

延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

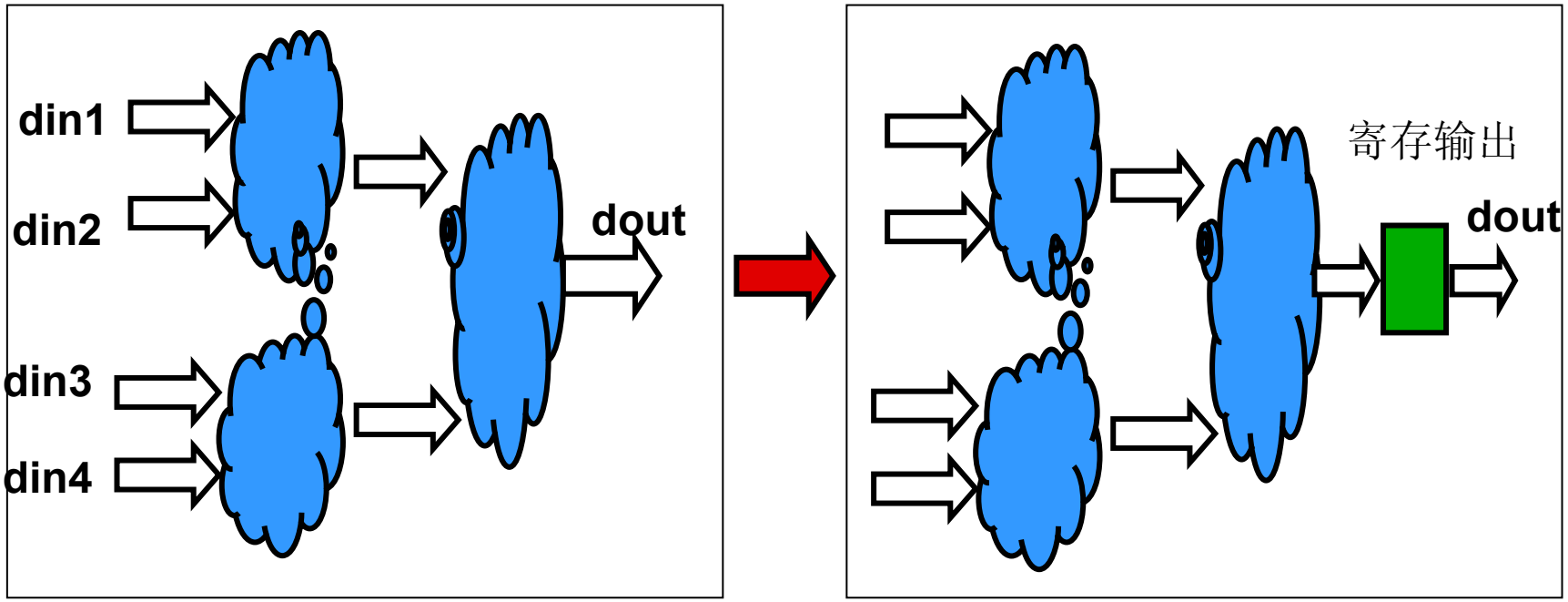
组合路径切割

- 常用手段有：
  1. 寄存输入和寄存输出
  2. 状态拆分；
  3. 流水线技术；

(1) 寄存输入和寄存输出

- 数字系统中，各模块应采取（寄存输入和）寄存输出，这样做有如下优点：
  1. 模块化清晰(特别是寄存输出)；
  2. 提高系统最高工作速率；
  3. 有利于整个系统和单个模块分别进行静态时序分析。

寄存输出设计例子



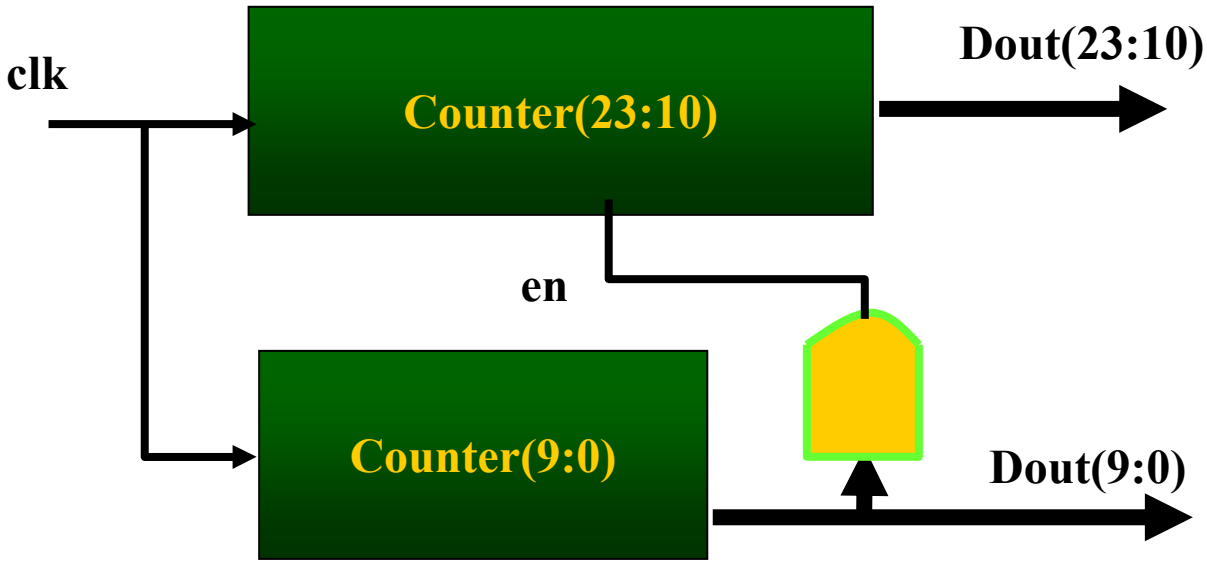
(2) 状态拆分

- 经典案例：预定标计数器。
- 设计一个32bit的计数器，采用binary编码。按照常规的设计：

```
If( reset = '1' ) then
    dout <= ( others => '0' );
elsif( clk'event and clk = '1' ) then
    dout <= dout + '1';
end if;
```

综合后的速度

- 用SCL05u ASIC库进行综合，得出关键路径延时为约14.07 ns，clk的最高工作频率约为69.6 M。
- 为了提高该设计的最高工作频率，修改方案：



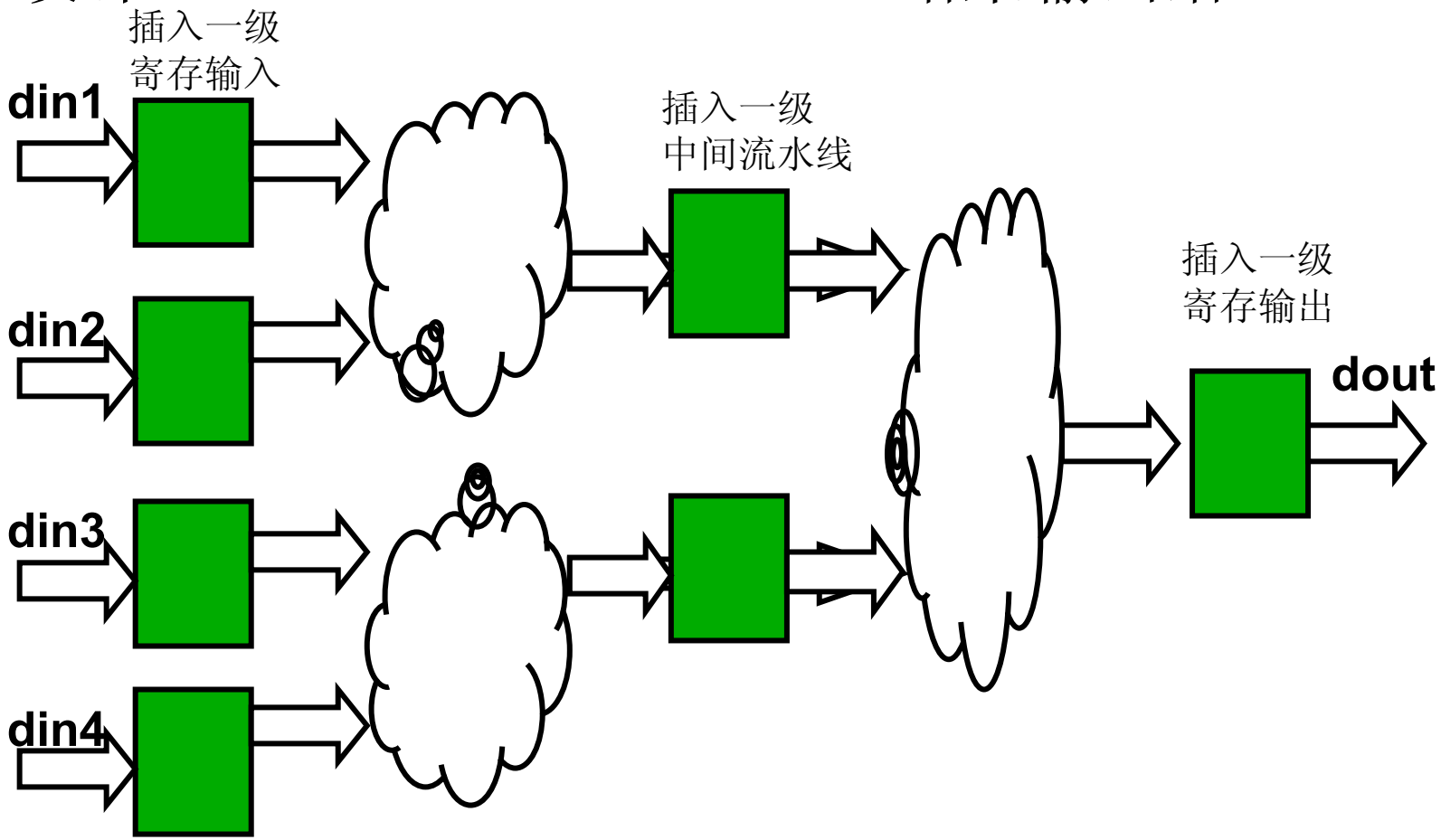
- 注意上图中en的高脉冲宽度为一个clk时钟周期。
- 综合后，其关键路径延时为6.40 ns，clk的最高工作频率可达140.8 MHz !!
- 可见，进行状态拆分后，各状态的译码的组合电路规模都比原来小，从而减小了关键路径的延时。

(3) 流水线技术

- 流水线技术几乎是最常用的提供系统工作速率的强有力手段。
- 流水线能动态的提升器件性能；

流水线设计例子

- 设计din1 + din2 + din3 + din4结果输出给dout。



流水线技术的原理

- 其基本思想是：利用寄存器将一条经过多级逻辑的长路径切分成几段小路径；把原来必须在一个时钟周期内完成的操作分成多个周期完成， 从而达到提高整体工作速率的作用。
- 假设原路径延时为t，加入2级流水线并且假设路径切割均匀，则路径延时可以减少到约t/3，从而系统速率可以提高到原来的3倍左右。
- 要注意的是：加入流水线后，输出信号和改动之前相比会往后推迟（上例中延迟为2t）。所以采用流水线技术时，要记得对相关的其它信号进行同步时序调整。

流水线与FPGA

- 由于FPGA内寄存器资源相当丰富，因此流水线结构通常为一种既能提高设计性能，又不需要增加额外的器件资源。在某些情况，还可以使布线更为容易。

延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

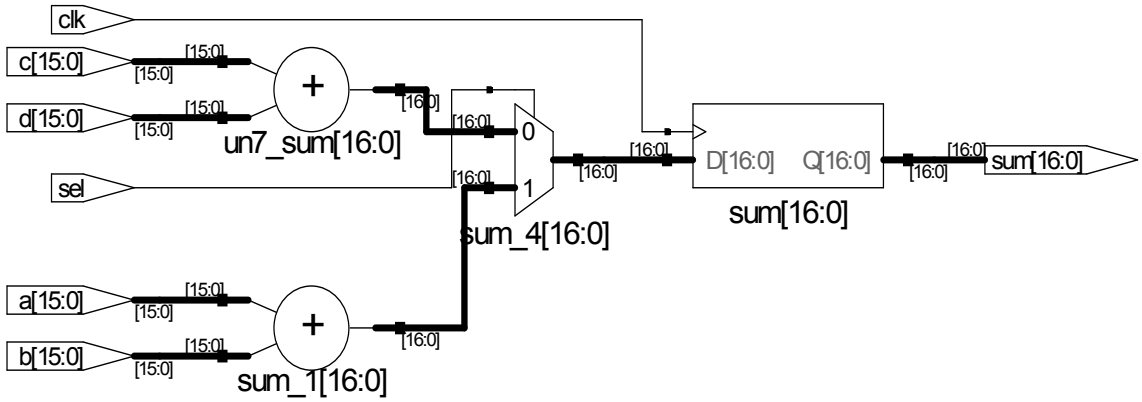
资源共享

- if语句情况的资源共享举例
- 循环语句情况的资源共享举例
- 子表达式共享举例

If语句的资源共享举例

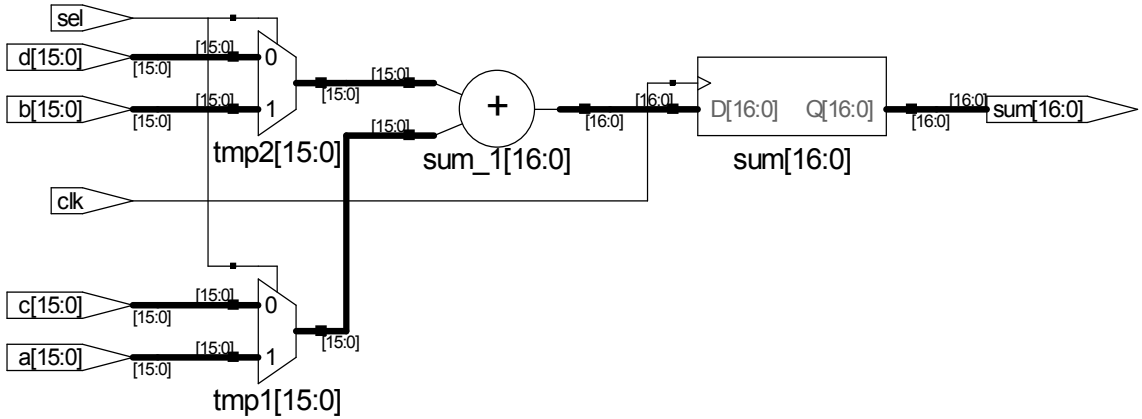
没有资源共享时用了2个加法器

```
If(sel)
  sum = a+b;
Else
  sum = c+d;
```



加法器资源宝贵，通过修改代码只使用2个选择器和1个加法器实现

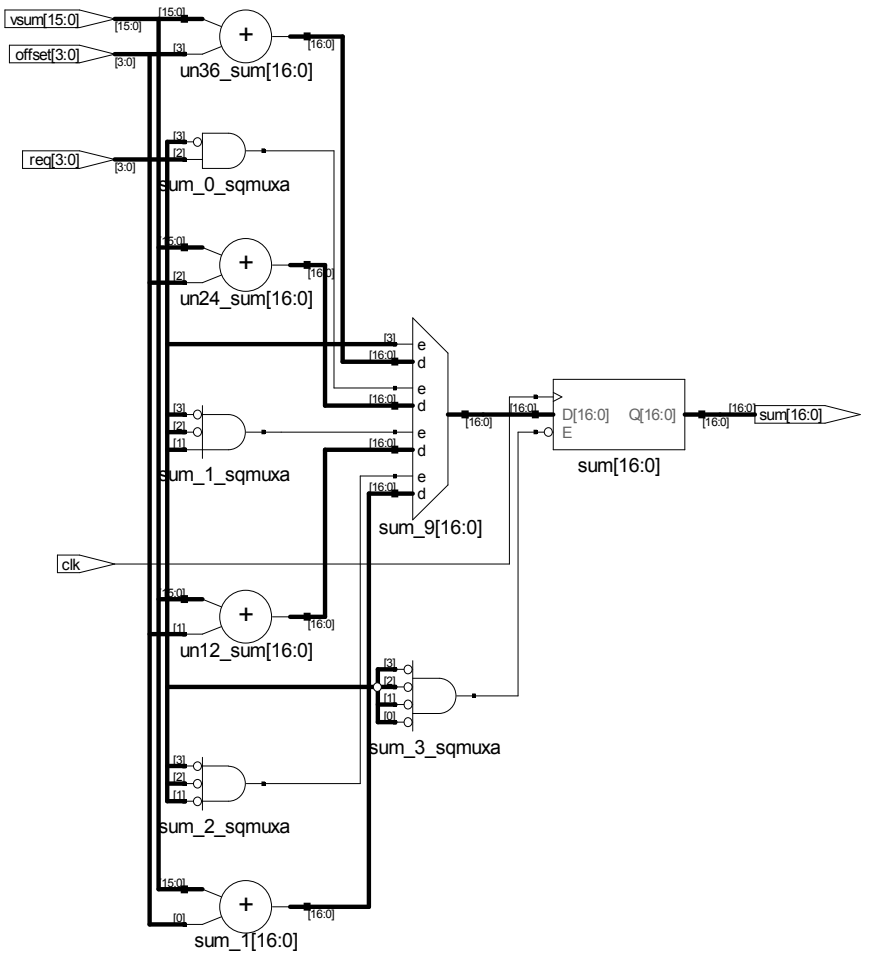
```
If(sel)
   tmp1 = a;
   tmp2 = b;
Else
   tmp1 = c;
   tmp2 = d;
Assign sum = tmp1 + tmp2;
```



循环语句情况的资源共享举例

- 与选择信号相比，运算符占用更多的逻辑资源，如果循环语句中有一个运算符，综合工具必须对所有的条件求值。
- 下面的设计，综合工具用4个加法器和一个选择器实现，只有当req信号为关键路径时才建议采用这种描述方式。

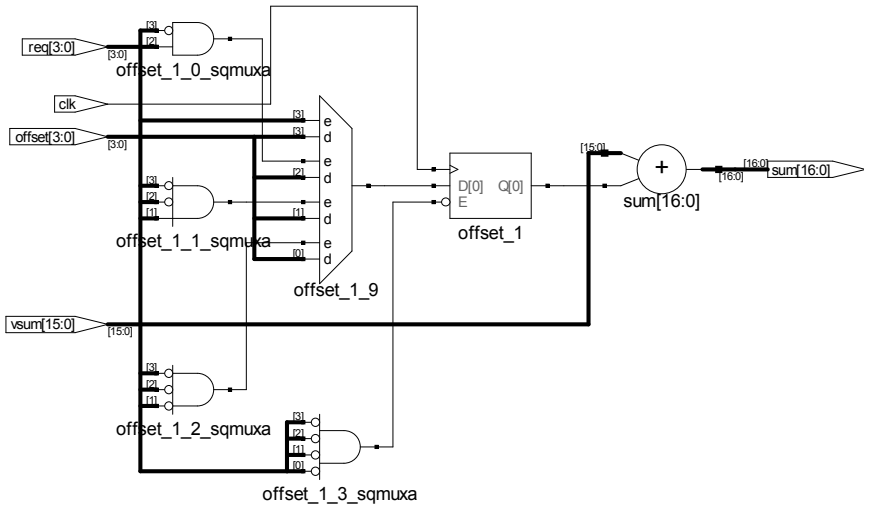
```
for(i=0;i<=3;i=i+1)
   begin
      if(req[i])
         sum <= vsum + offset[i];
   end
```



循环语句情况的资源共享举例(续)

- 运算符应该移到循环语句之外，这样在执行加法操作以前，对数据信号进行选择，用一个多路选择器和一个加法器就可实现上述功能。

```
always @(posedge clk)
begin
   for(i=0;i<=3;i=i+1)
   begin
       if(req[i])
           offset_1 = offset[i];
   end
end

assign   sum = vsum + offset_1;
```



子表达式共享举例

- 一个运算表达式包含了2个以上的变量，同时此表达式在多个等式中用到，应共享这些运算，减少电路的面积。
- 通过申明一个临时变量存储子表达式，然后在任何需要重复使用这个子表达式的地方用临时变量替代。如：

```
assign tmp = a+b;
assign x = tmp;
assign y = tmp + c;
```

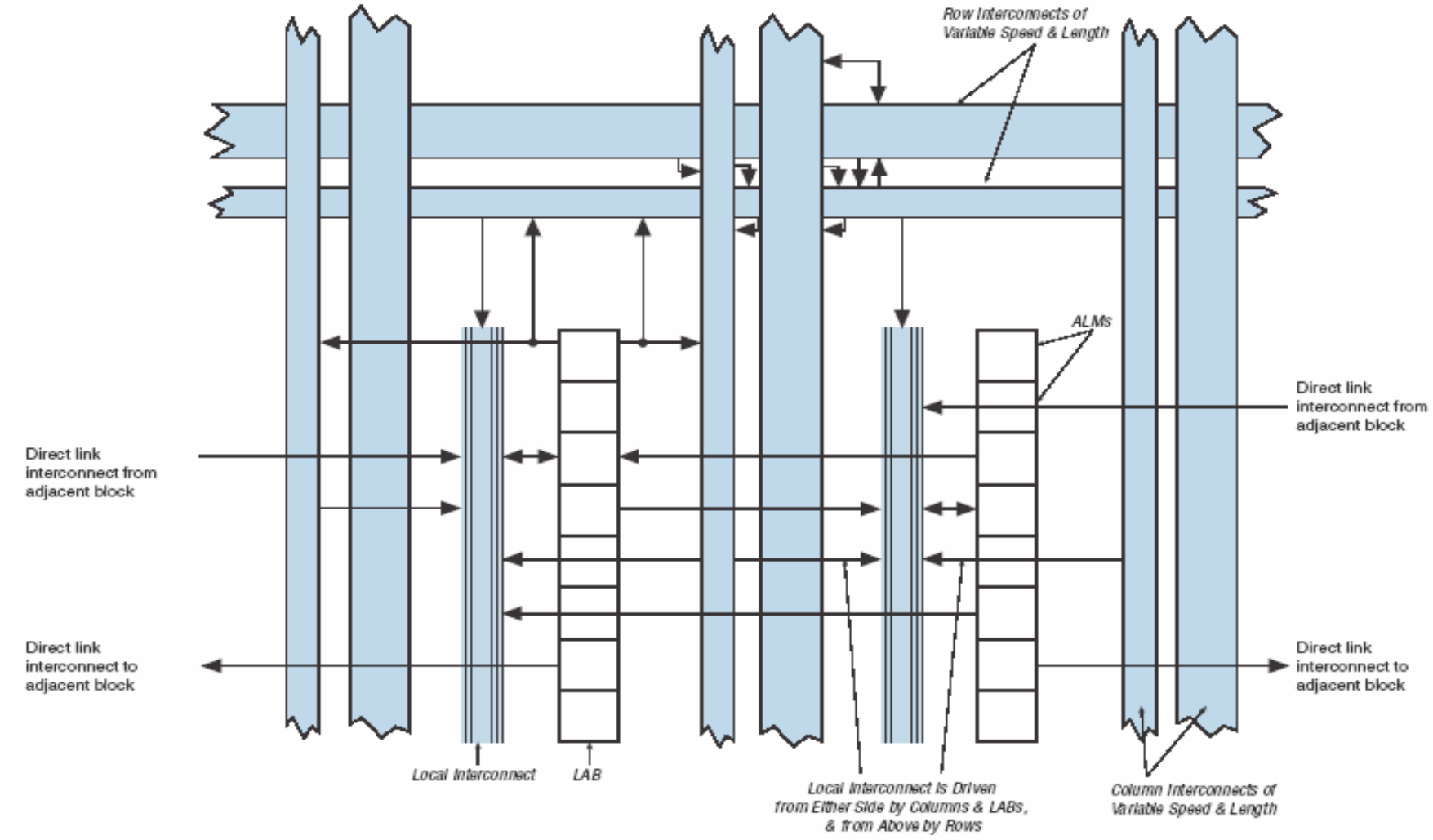
延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

专有资源的利用

- 针对LAB的结构进行设计调整
- 充分利用ALM特性进行设计
- 利用MultiTrack Interconnect的特性进行设计
- 高效利用IOE资源
- 利用TriMatrix Memory进行设计
- 合理选择硬件资源- DSP block

StratixII的LAB结构



充分利用fast local和direct link特性来提高设计性能

- FPGA内部线延迟，主要体现在Routing过程中经过的开关数量；
- 减少关键路径上使用的布线开关数量可以有效的降低线延迟；
- StratixII的每个ALM可以通过fast local和direct link直接驱动24个ALM；利用这个特性可指导我们进行流水线、寄存器复制、Retiming等优化工作。

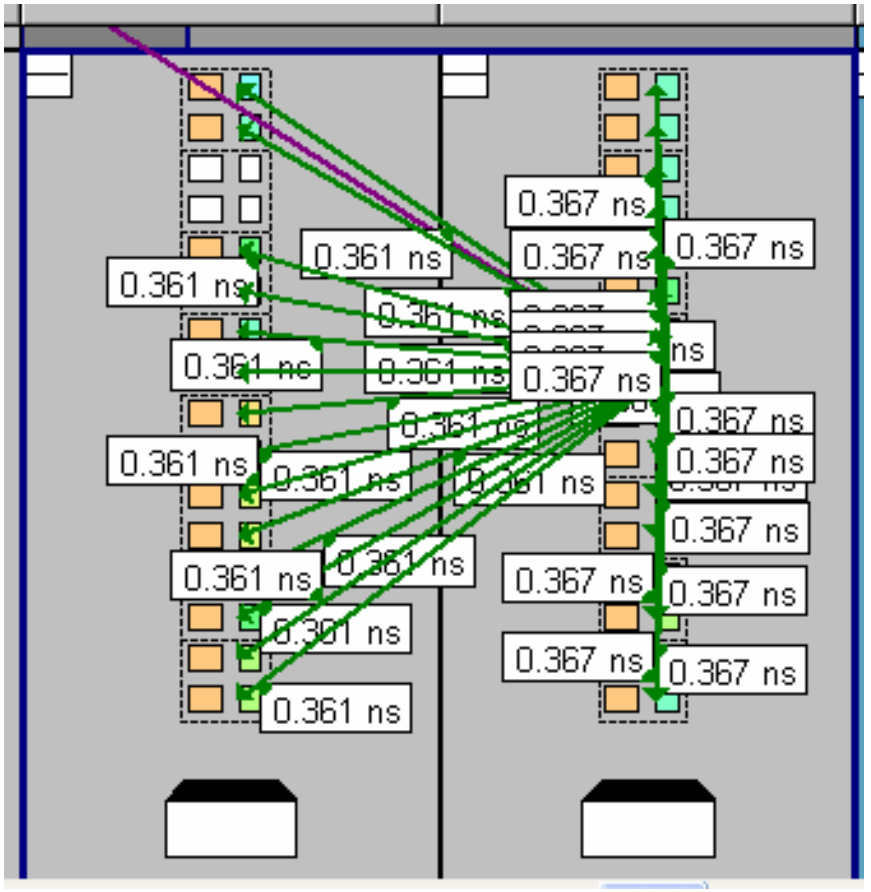
利用MultiTrack Interconnect的特性进行设计

- MultiTrack有着固定的速度和长度；
- 适用于模块之间的连接；
- 合理分配R4，C4，R24，C16的使用；

驱动12个ALM的情况

- 从en到其它ALM的最大走线延迟为0.367ns

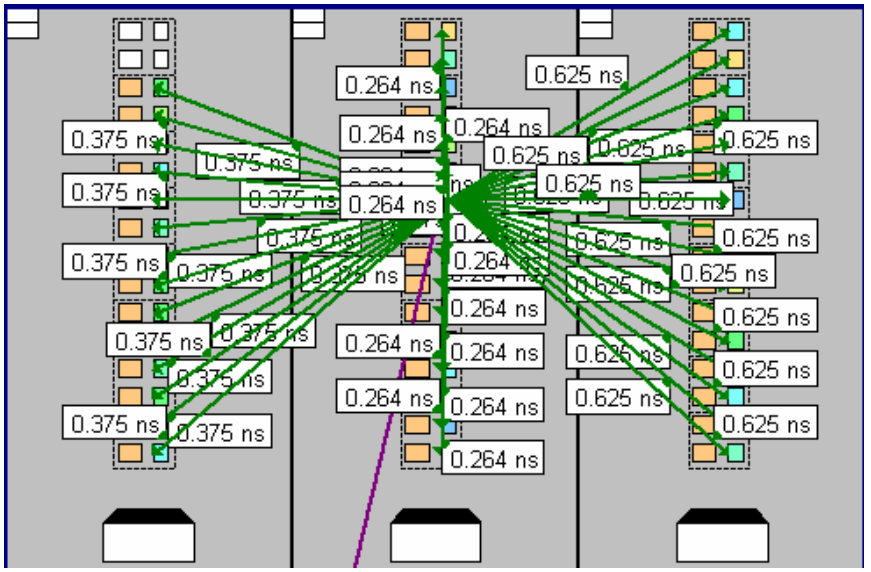
```
always @(posedge clk)
  en <= enpro_0 & enpro_1;

always @(posedge clk)
begin
  if(en)
  begin
     for(i=0;i<=28;i=i+1)
     begin
        if(i == 0)
           c[i] <= (a[i] & b[i]) | ((a[i] | b[i]) &
    cin);
        else
           c[i] <= (a[i] & b[i]) | ((a[i] | b[i]) &
    c[i-1]);
     end
  end
end
```



驱动23个ALM的情况

- 从en到其它ALM的最大走线延迟为0.625ns

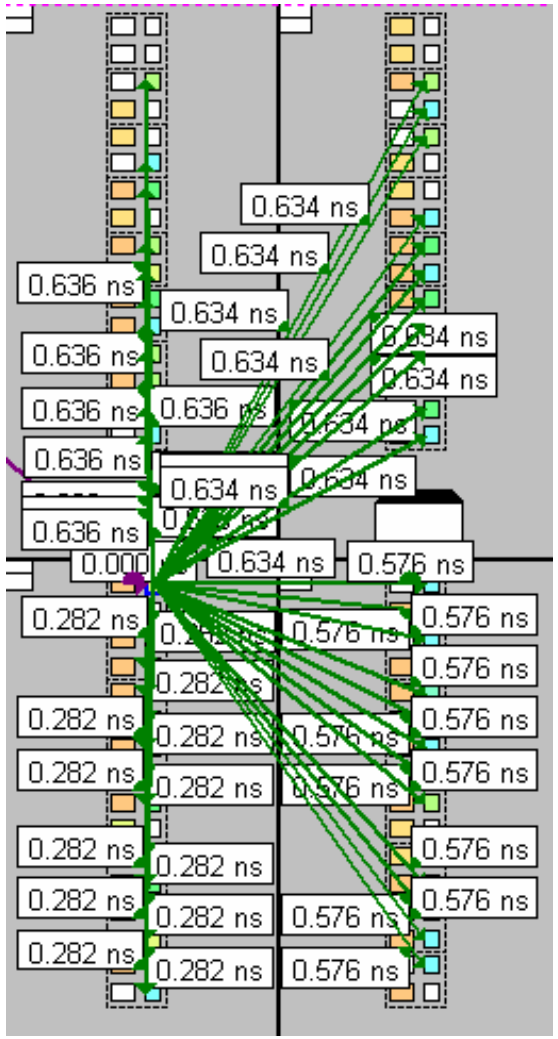
```
always @(posedge clk)
  en <= enpro_0 & enpro_1;

always @(posedge clk)
begin
  if(en)
  begin
    for(i=0;i<=44;i=i+1)
    begin
      if(i == 0)
        c[i] <= (a[i] & b[i]) | ((a[i] | b[i]) &
    cin);
      else
        c[i] <= (a[i] & b[i]) | ((a[i] | b[i]) &
    c[i-1]);
    end
  end
end
```



驱动25个ALM的情况

- 从en到其它ALM的最大走线延迟为0.636ns

```
always @(posedge clk)
  en <= enpro_0 & enpro_1;

always @(posedge clk)
begin
  if(en)
  begin
     for(i=0;i<=46;i=i+1)
     begin
        if(i == 0)
           c[i] <= (a[i] & b[i]) | ((a[i] | b[i]) &
    cin);
        else
           c[i] <= (a[i] & b[i]) | ((a[i] | b[i]) &
    c[i-1]);
     end
  end
end
```



充分利用ALM特性进行设计

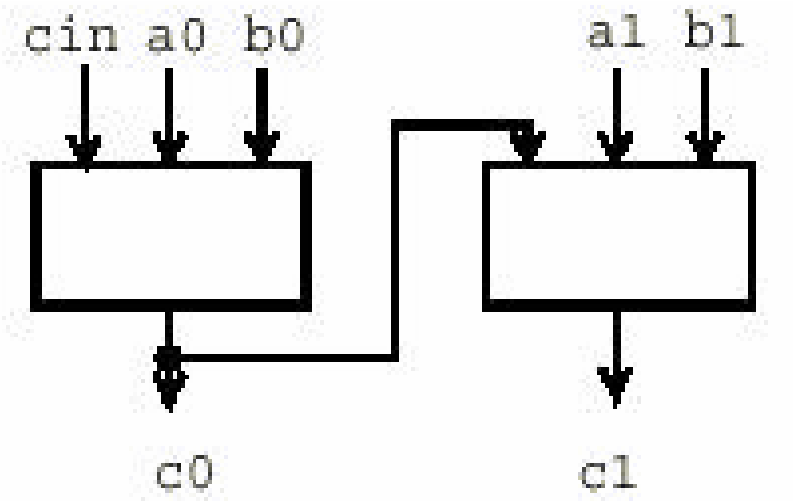
- 充分利用ALM的各种模式进行设计
- 充分利用ALM的各种特殊链路资源

串行进位链

- 串行进位与超前进位
- 每一个信号赋值、进程和元件的引用对应着特定的逻辑，通过语句的不同描述来控制具体的实现电路，可以达到实现不同结构的目的
- 加法器进位链电路的两种可能的描述

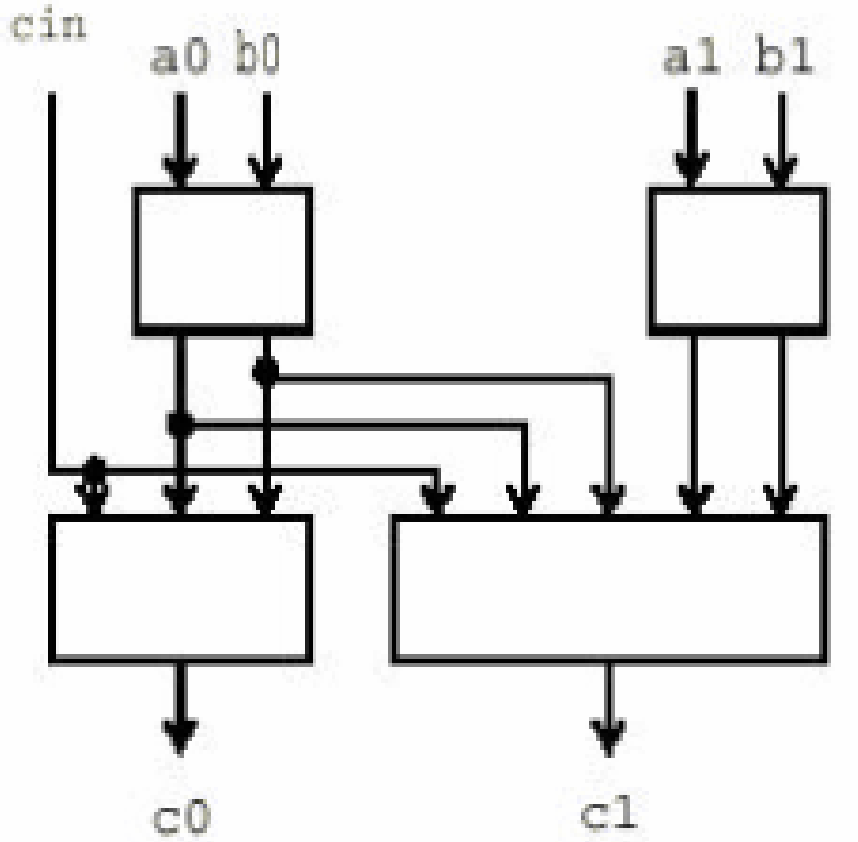
串行进位链

```
//a is the add end
//b is the aug end
//c is the the carry
//cin is the carry in
assign c0 = (a0 & b0) | ((a0 | b0) & cin);
assign c1 = (a1 & b1) | ((a1 | b1) & c0);
```



超前进位链（并行结构）

```
//ps is propagate
//gs is generate
Wire p0; /* synthesis syn_keep = 1*/
Wire p1; /* synthesis syn_keep = 1*/
Wire g0; /* synthesis syn_keep = 1*/
Wire g1; /* synthesis syn_keep = 1*/
assign p0 = a0 | b0;
assign g0 = a0 & b0;
assign p1 = a1 | b1;
assign g1 = a1 & b1;
assign c0 = g0 | (p0 & cin);
assign c1 = g1 | (p1 & g0)  | (p1 & p0
    & cin);
```

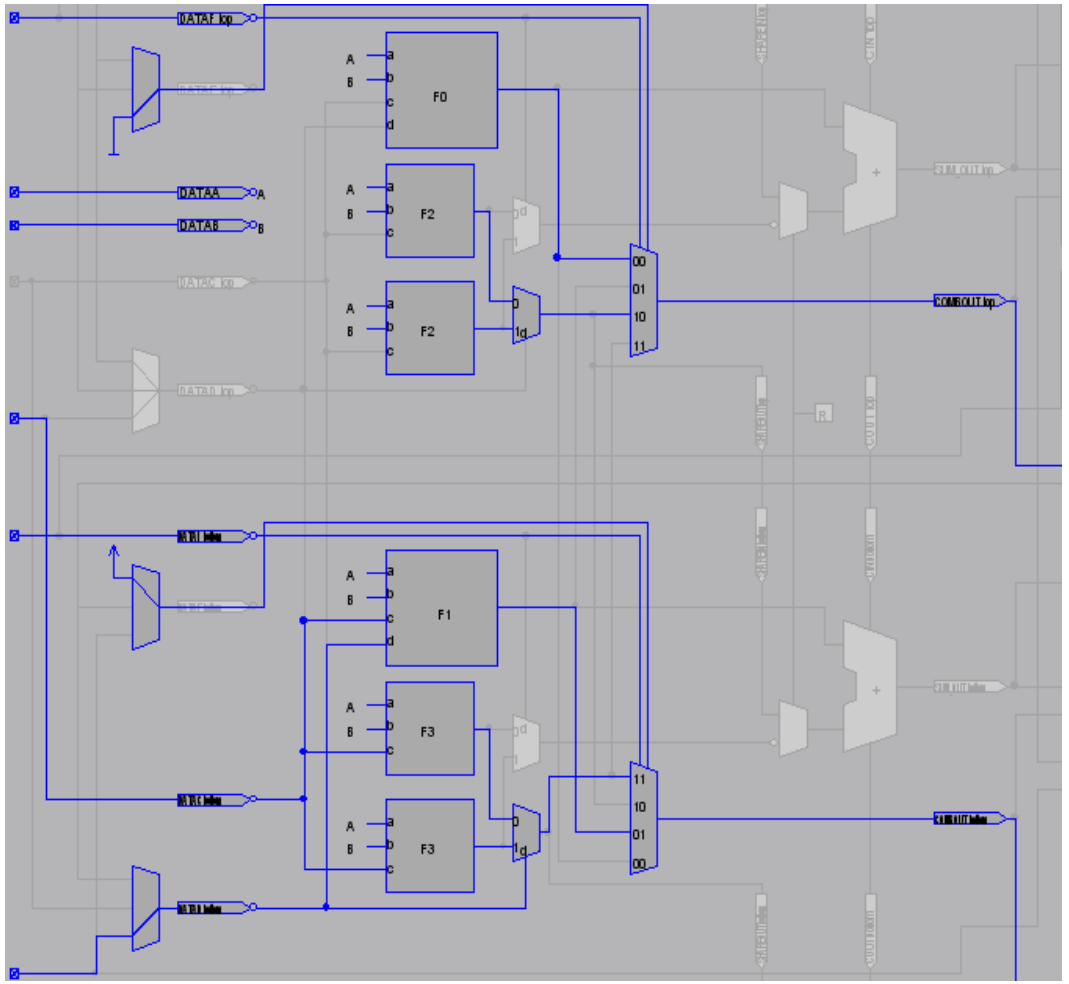


结果分析

- 如果我们是进行的ASIC设计，无疑的并行结构的加法器可以获得更快的速度，但是占用了更大的面积；
- 如果我们使用的FPGA进行设计，无疑第一种形式的串行加法器可以获得更快的速度，也占用最少的面积，因为它可以在同一个ALM中实现。

在EP2S15F484C4中得到的最终结果——串行加法器

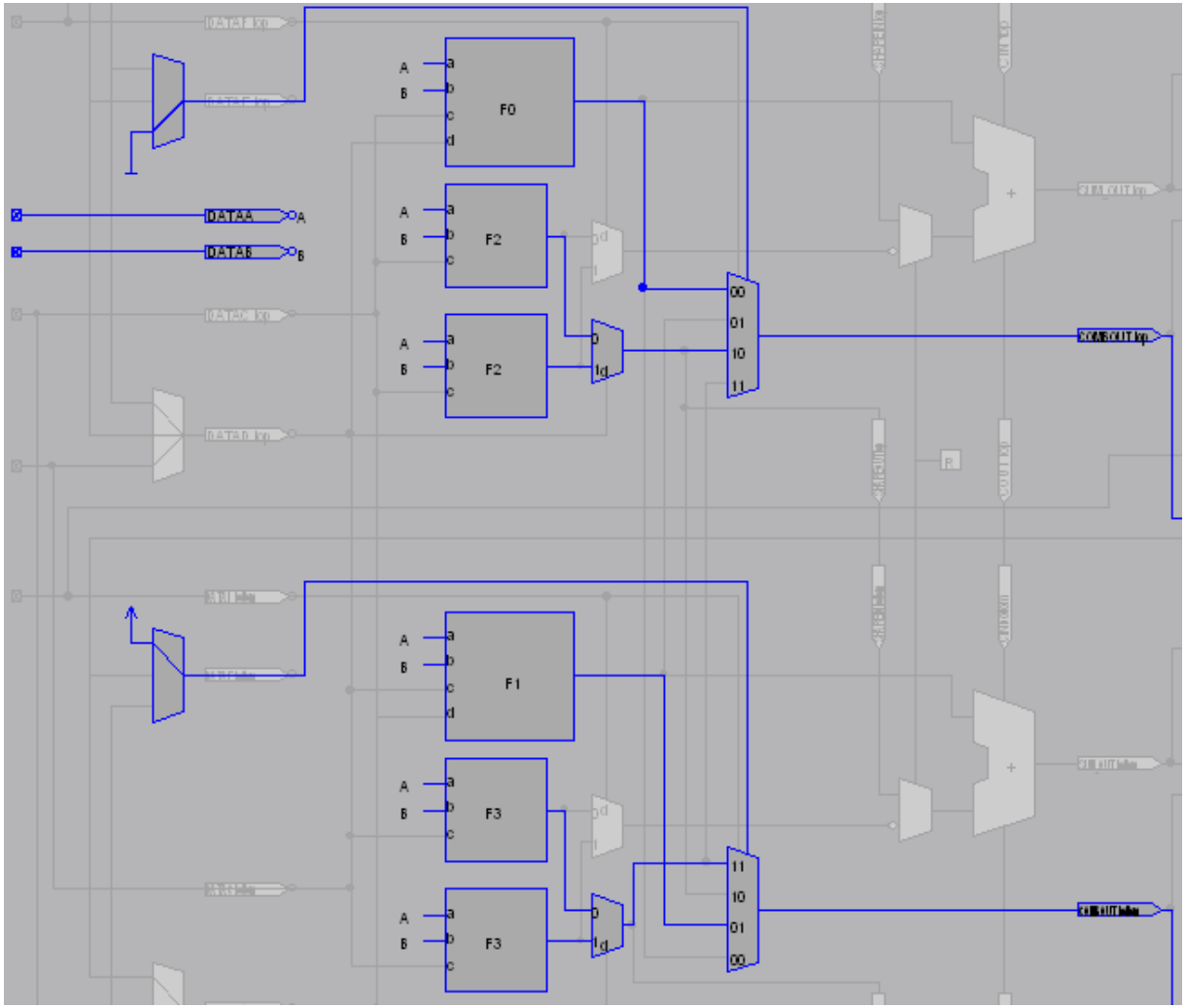
- 使用了一个ALM；
- 使用了2个4输入的LUT，4个3输入的LUT，2个或门，2个选择器
- 最大的Tpd为8.501ns



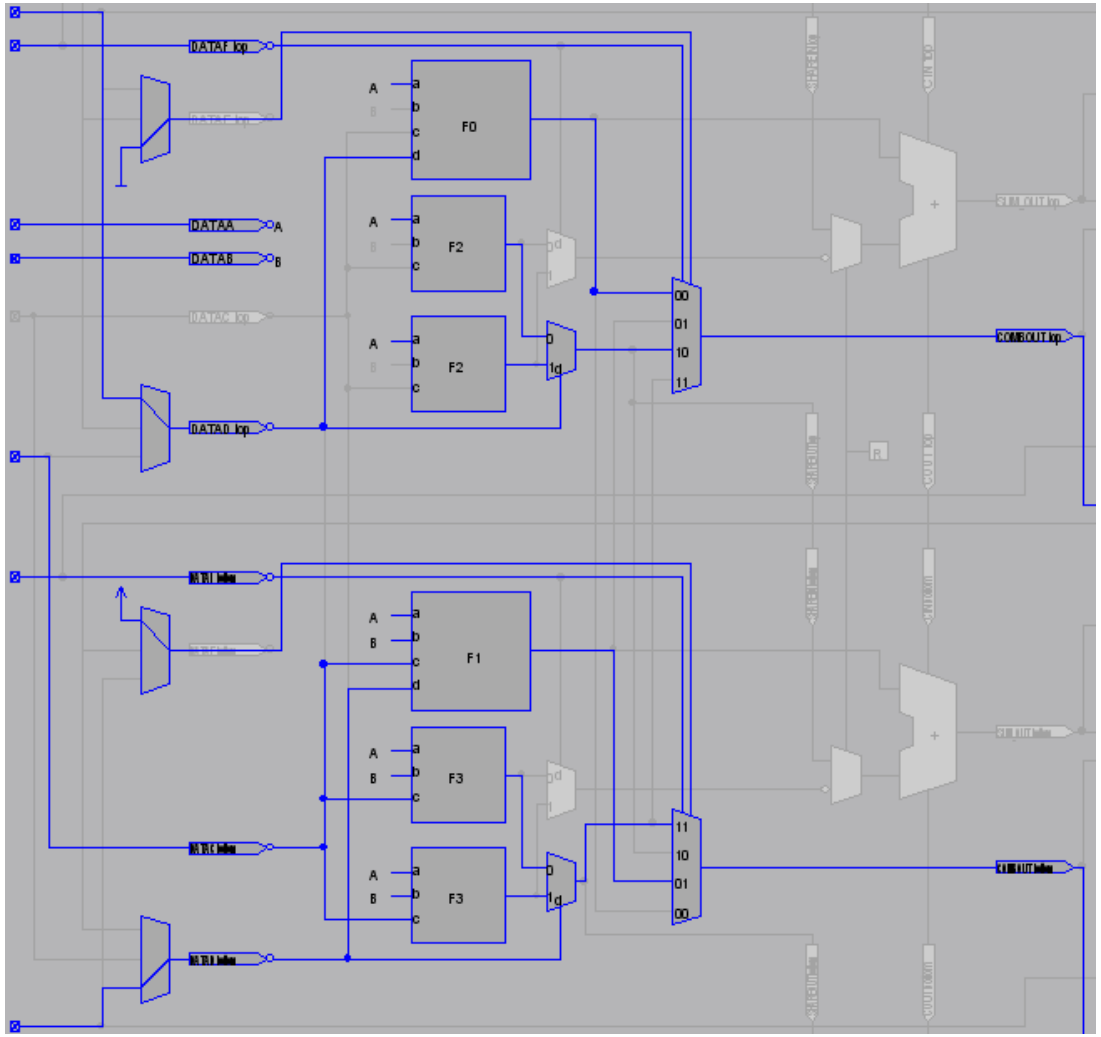
在EP2S15F484C4中得到的最终结果——并行加法器

- 使用了三个ALM；
- 使用了1个5输入的ALUT，5个3输入的ALUT
- 最大的Tpd为9.217ns

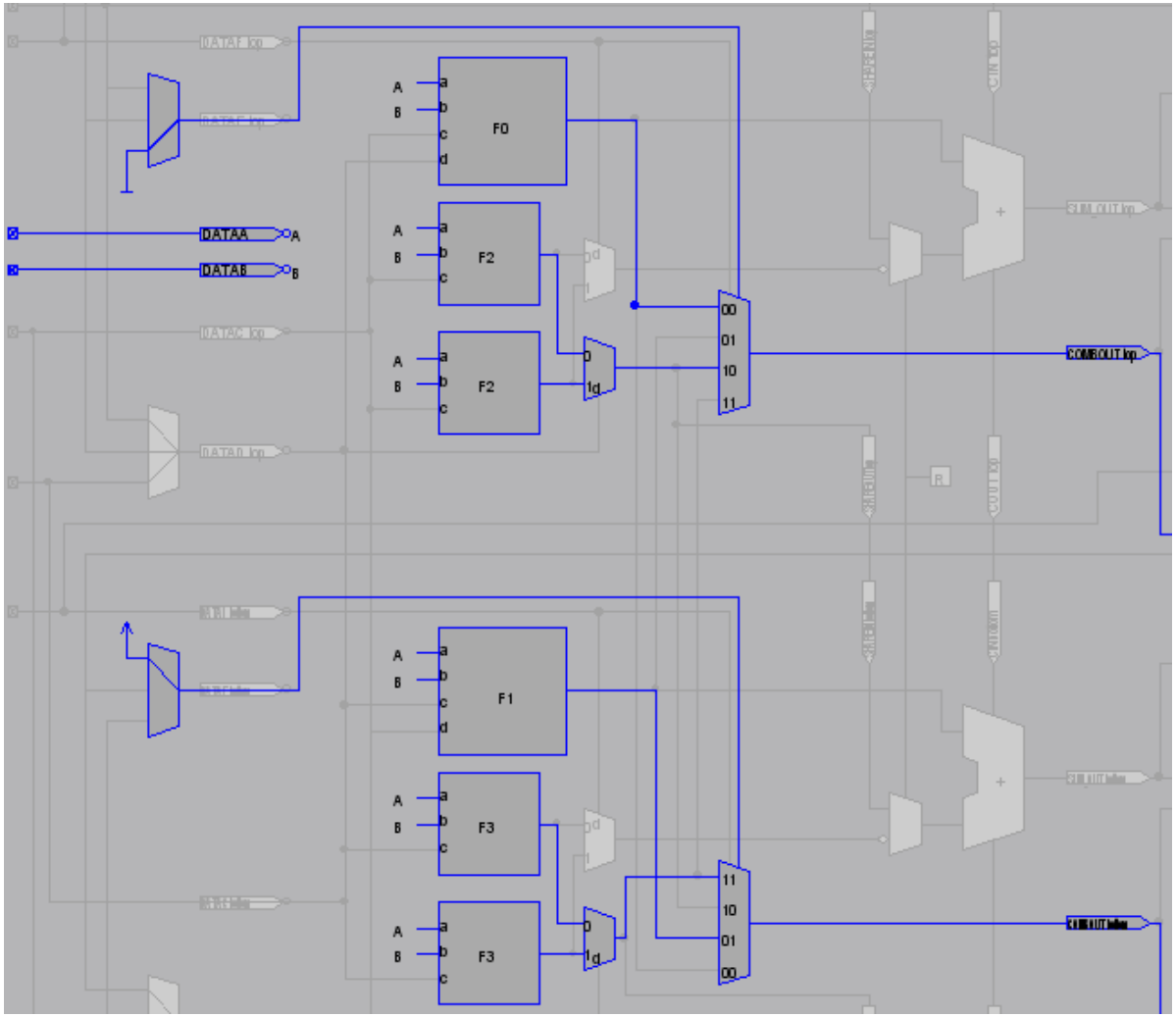
在EP2S15F484C4中得到的最终结果——并行加法器（续）



在EP2S15F484C4中得到的最终结果——并行加法器（续）

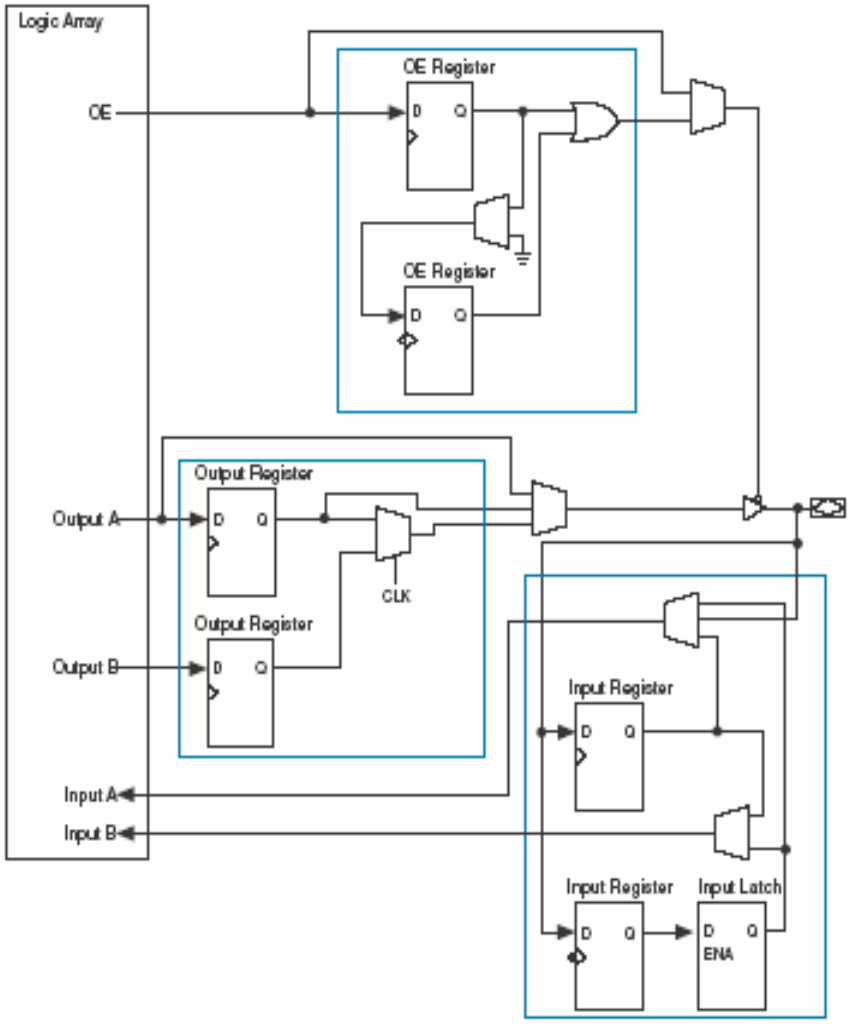


在EP2S15F484C4中得到的最终结果——并行加法器（续）



高效利用IOE资源

- 充分利用IOE中的寄存器资源，减少ALUT的使用；
- 进出FPGA的信号都尽量经过寄存器，这么做能够放宽对外部电路或其它芯片的时序要求，提高单板设计速度
- 处理双向口时，输入输出应采用同样的异步、同步置位或清零信号；每个输出信号都有独立的输出使能信号；三态控制低电平有效；



利用TriMatrix Memory进行设计

- 使用Memory实现移位寄存器；
- 充分利用奇偶校验位进行设计；
- 利用字节使能实现位宽变换；
- 合理选择使用的TriMatrix Memory类型;

合理选择硬件资源- DSP block

- null

延时优化的几个要点

1. 长路径的避免
2. 串行运算转并行运算
3. 优先级电路的延时优化
4. 数据通路拷贝
5. 数据运算式变换
6. 变量运算优化
7. 组合路径切割
8. 资源共享
9. 合理选择专有硬件资源
10. 优化关键路径

优化关键路径

- 优化关键路径的方法:
  - 关键路径位于同一个Module中，在综合时可以得到最佳效果；
  - 利用Logic Lock可以对关键路径模块采用不同的优化策略，可以单独进行综合和布局布线（关键路径模块采用速度优先策略，其他模块采用面积优先策略）；
  - 可以针对关键路径模块，进行位置约束，减少线延迟；也可以对关键信号进行约束，使用特殊的走线资源；
  - 采用迂回策略，降低非关键路径模块的面积，为关键路径腾挪空间和特殊走线资源；

小结

- 可靠设计：
  - RTL风格的描述中，要注意一个进程里只能判断一次时钟沿；
  - 敏感信号表不完整，往往会造成前仿真和后仿真结果不一致；
  - 条件判断语句中，要注意对所有分支进行输出信号的赋值，以避免锁存；
  - 巧妙地使用无关态，可以引导综合器综合出优化的电路；

小结(续)

- case无优先级；single if和multiple if语句的优先级顺序；
- 注意要正确地使用多路开关或者三态缓冲来进行总线复用，避免总线冲突；
- 双向端口描述中，注意高阻态的赋值；注意其测试矢量生成时，也要注意设置高阻态；
- 毛刺消除的要点：竞争和冒险的避免；Gray编码；寄存器消除方法(数据输入吸收，时钟使能吸收)。

小结(续)

- 同步设计
  - 尽量避免使用门控时钟，应以时钟使能来代替；
  - 尽量避免使用派生时钟，应以派生使能来代替；
  - 路径与路径延时的含义；路径延时与系统工作时钟的最高频率的关系；
  - 缩小组合电路的规模(扇入)，可以降低系统的路径延时从而提高最高工作速率；

小结(续)

- 异步设计
  - 尽量避免使用异步设计；
  - 异步多时钟系统中，注意做好时钟域之间的同步；
  - 三种异步时钟域的同步方法；
  - 引入时钟域同步后，容易造成信号相位拉伸；
  - 为避免矢量拉伸对数据矢量的破坏，可以考虑采用Gray码进行计数器编码；
  - 为了在静态时序分析中快速地进行false path的设置，最好根据时钟域来进行信号命名。

小结(续)

- 提高设计性能的技巧
  - 串转并不但可以降低系统的延时，而且可以大幅度提高系统的数据吞吐率。二叉树结构可以有助于引入平衡的流水线。
  - 避免长的路径。通过编码，降低扇入和利用电路的等价性巧妙的分配延迟，将长路径中的部分延迟转移到其他路径上。
  - 优先级电路中，对于到达比较晚(即延时较大)的信号，如果不是处于最高优先级处时，可想办法将该信号向靠近输出端的位置搬移，以减小所在的路径的延迟。
  - 可以通过搬移if条件模块的方法搬移需要移动的模块

小结(续)

- “信号搬移”进行延时优化的方法有：优先级电路中的模块搬移、数据通路拷贝、数据运算式变换等；
- 在优先级别电路中进行信号搬移时，要注意修正条件判断式，使得优先级别不变；
- 可能的情况下，应该尽量使用常量运算来代替变量(信号)运算；
- 可以使用状态机拆分、流水线等技术进行路径切割，达到延时优化的目的；

小结(续)

- (寄存器输入和)寄存输出不仅可以提高系统工作速率，而且有利于模块和整体进行分别的静态时序分析；
- 资源共享不仅可以节省面积，有时还可以方便布线和提升设计性能；
- 合理选择专有硬件资源，如进位链，IOE，TriMatrix Memory，DSP block等
- 用工具优化关键路径

END