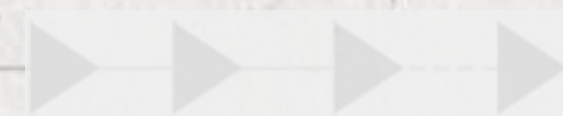


芯动力——硬件加速设计方法

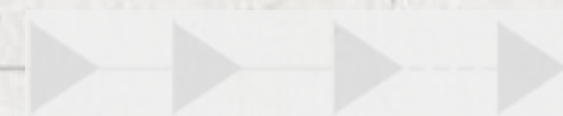
第二章：高质量VerilogHDL描述方法(2)

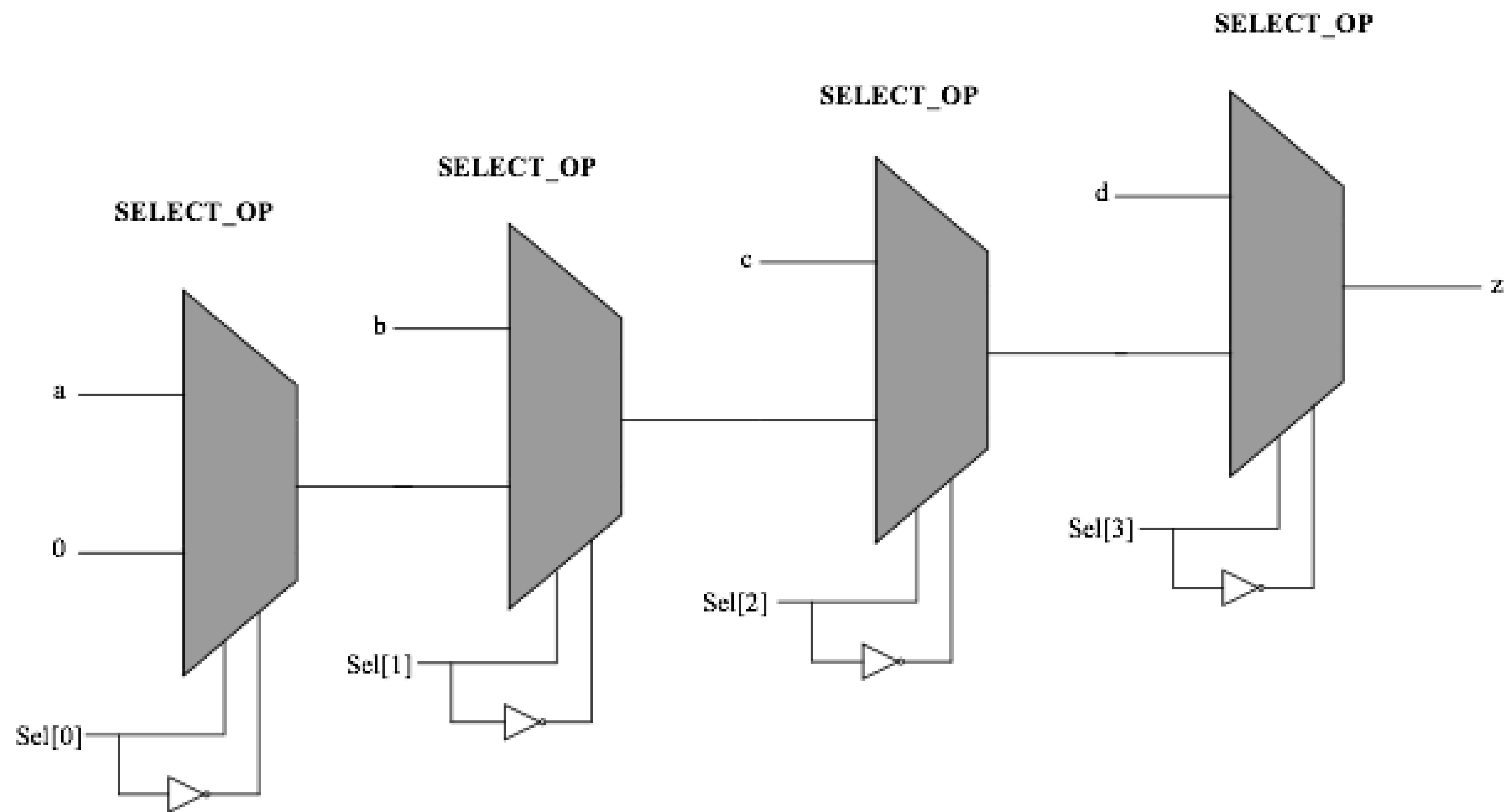
邸志雄@西南交通大学
zxdi@home.swjtu.edu.cn



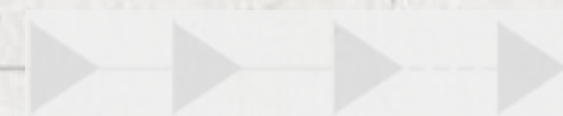
一、在RTL编码中考虑时延

```
module mult_if (a,b,c,d,sel,z)
input      a,b,c,d ;
input [3:0] sel  ;
output     z      ;
reg        z      ;
always @ ( * )
begin
    z = 1' b0      ;
    if (sel[0]) z = a  ;
    if (sel[1]) z = b  ;
    if (sel[2]) z = c  ;
    if (sel[3]) z = d  ;
end
endmodule
```





假设b信号的延迟较大，到来较晚，那么，电路应该如何修改？




```
module mult_if (a,b,c,d,sel,z)
input      a,b,c,d ;
input [3:0] sel ;
output     z      ;
reg        z      ;
always @ ( * )
begin
    z = 1' b0      ;
    if (sel[0]) z = a ;
    if (sel[1]) z = b ;
    if (sel[2]) z = c ;
    if (sel[3]) z = d ;
end
endmodule
```

sel[0]=0

sel[1]=1

sel[2]=0

sel[3]=0

z = b

不等于0

z ≠ b

if (sel[1] & ~(sel[2] | sel[3]))

module mult_if (a,b,c,d,sel,z)

input a,b,c,d ;

input [3:0] sel ;

output z ;

reg z ;

always @ (*)

begin

z = 1'b0 ;

if (sel[0]) z = a ;

if (sel[1]) z = b ;

if (sel[2]) z = c ;

if (sel[3]) z = d ;

end

endmodule

修改后:

always @ (*)

begin

z = 1'b0 ;

if (sel[0]) z = a ;

if (sel[2]) z = c ;

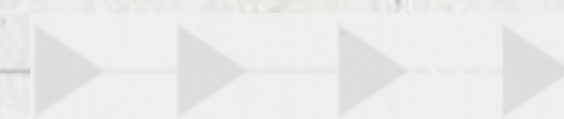
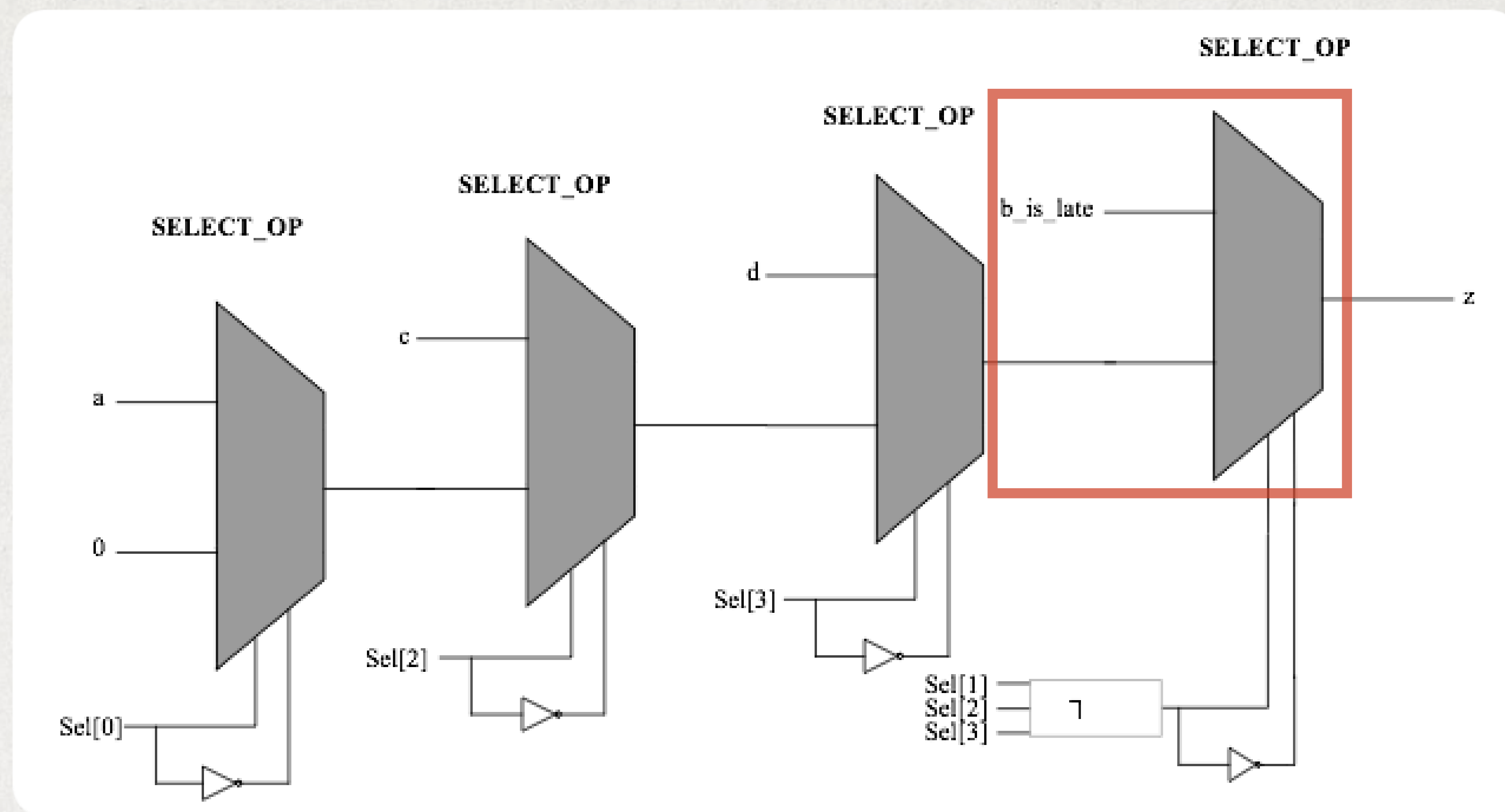
if (sel[3]) z = d ;


if (sel[1] & ~(sel[2] | sel[3]))

z = b_is_late ;

end

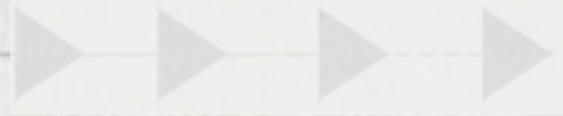
修改后:



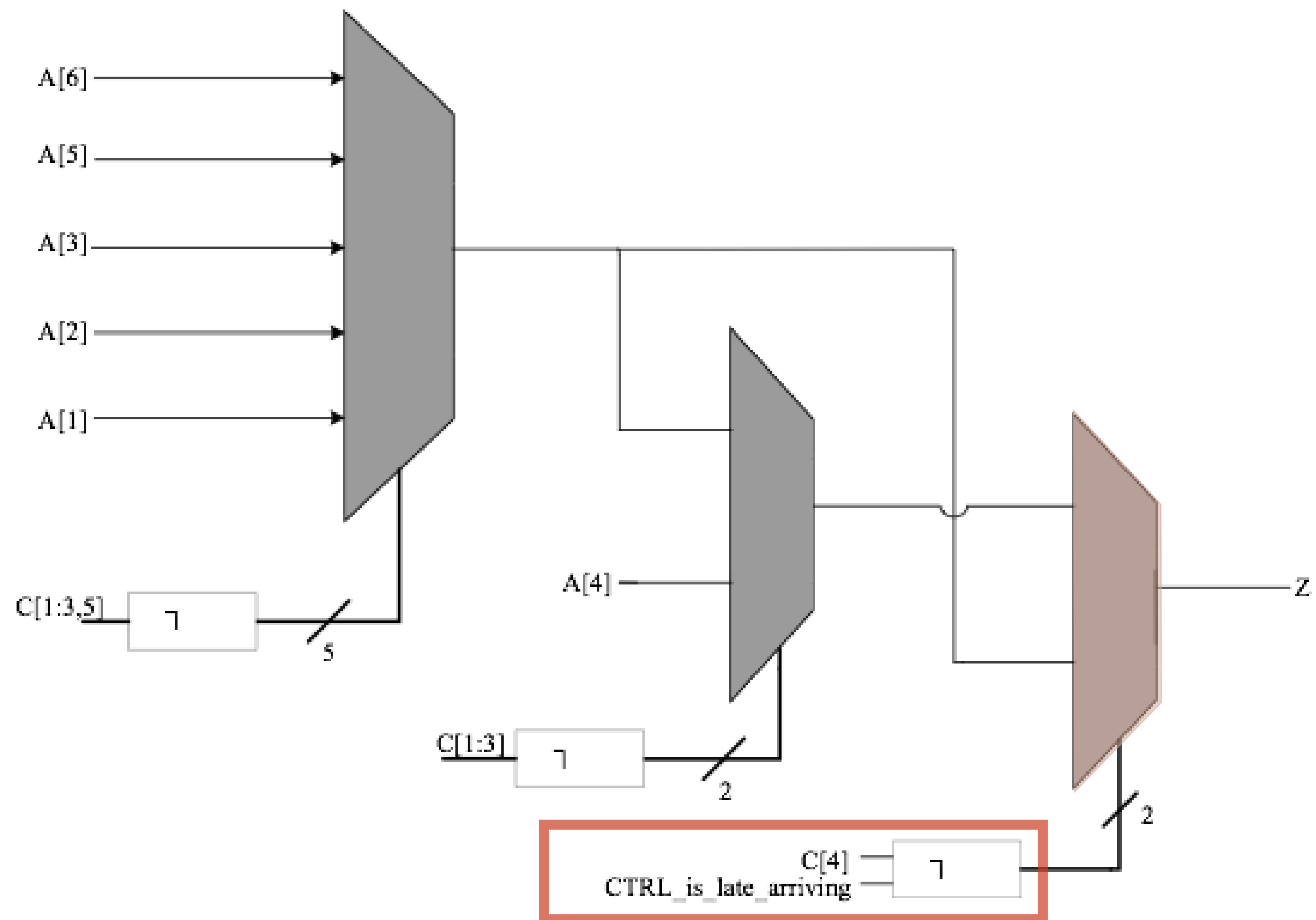


```
module single_if_late (A,C,  
    CTRL_is_late_arriving,Z);  
input [6:1] A;  
input [5:1] C;  
input CTRL_is_late_arriving;  
output Z;  
reg Z;  
always @ (A or C or CTRL_is_late_arriving)  
begin  
    if (C[1]==1'b1)  
        Z=A[1];  
    else if (C[2]==1'b0)
```

```
        Z=A[2];  
    else if (C[3]==1'b1)  
        Z=A[3];  
    else if (C[4]==1'b1  
        && CTRL_is_late_arriving==1'b0)  
        Z=A[4];  
    else if (C[5]==1'b0)  
        Z=A[5];  
    else  
        Z=A[6];  
end  
endmodule
```



修改后:




```

module single_if_improved (A,C,
                           CTRL_is_late_arriving,Z);
input [6:1] A;
input [5:1] C;
input CTRL_is_late_arriving;
output Z;
reg Z,Z1;
wire Z2, prev_cond ;
always @ (A or C)
begin
    if (C[1]==1'b1)
        Z1=A[1];
    else if (C[2]==1'b0)
        Z1=A[2];
    else if (C[3]==1'b1)
        Z1=A[3];
    else if (C[5]==1'b0)//removed the brain with the
late_arriving control signal

```

修改后:

```

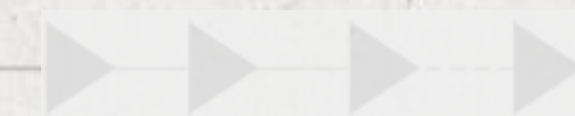
        Z1=A[5];
    else
        Z1=A[6];
    end
    assign Z2=A[4];
    assign
prev_cond=(C[1]==1'b1)|| (C[2]==1'b0)
           ||(C[3]==1'b1);
    always @ (C or prev_cond or CTRL_is_late_arriving or Z1 or Z2)
    begin
        if((C[4]==1'b1)&&(CTRL_is_late_arriving==1'b0))
            if(prev_cond) Z=Z1;
        else Z=Z2;
        else Z=Z1 ;
    end
endmodule

```

prev_cond = 0

A[4]的条件为真

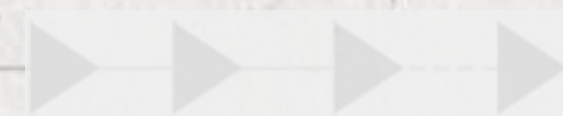
Z=A[4]



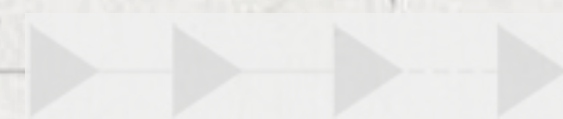
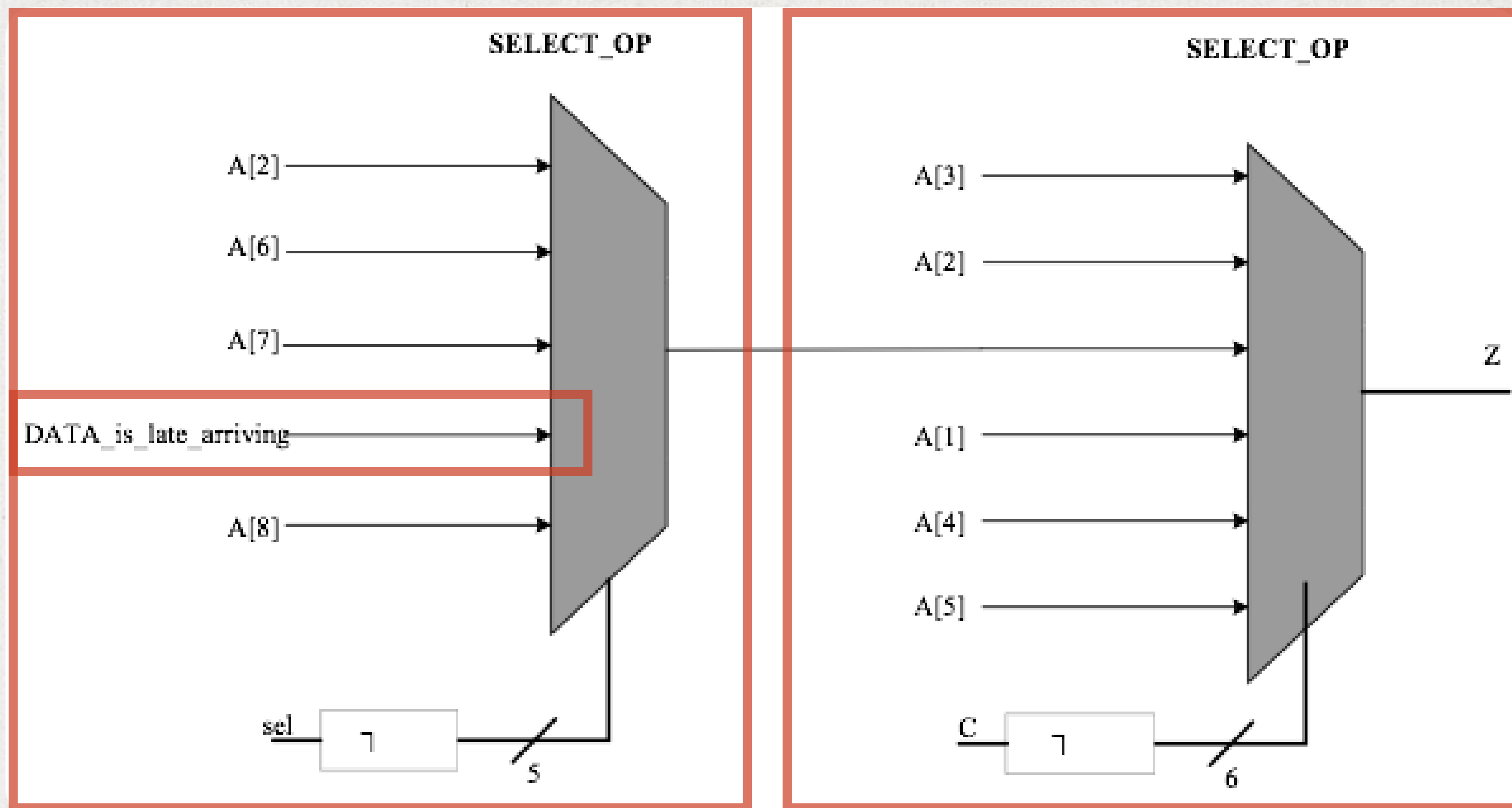
对应的电路结构是什么?

```
module case_in_if_01
(A,DATA_is_late_arriving,C,sel,Z);
input [8:1] A;
input DATA_is_late_arriving;
input [2:0] sel;
input [5:!] C;
output Z;
reg Z;
always @ (sel or C or A or
DATA_is_late_arriving)
begin
    if (C[1])
        Z=A[5] ;
    else if (C[2]==1'b0)
        Z=A[4] ;
```

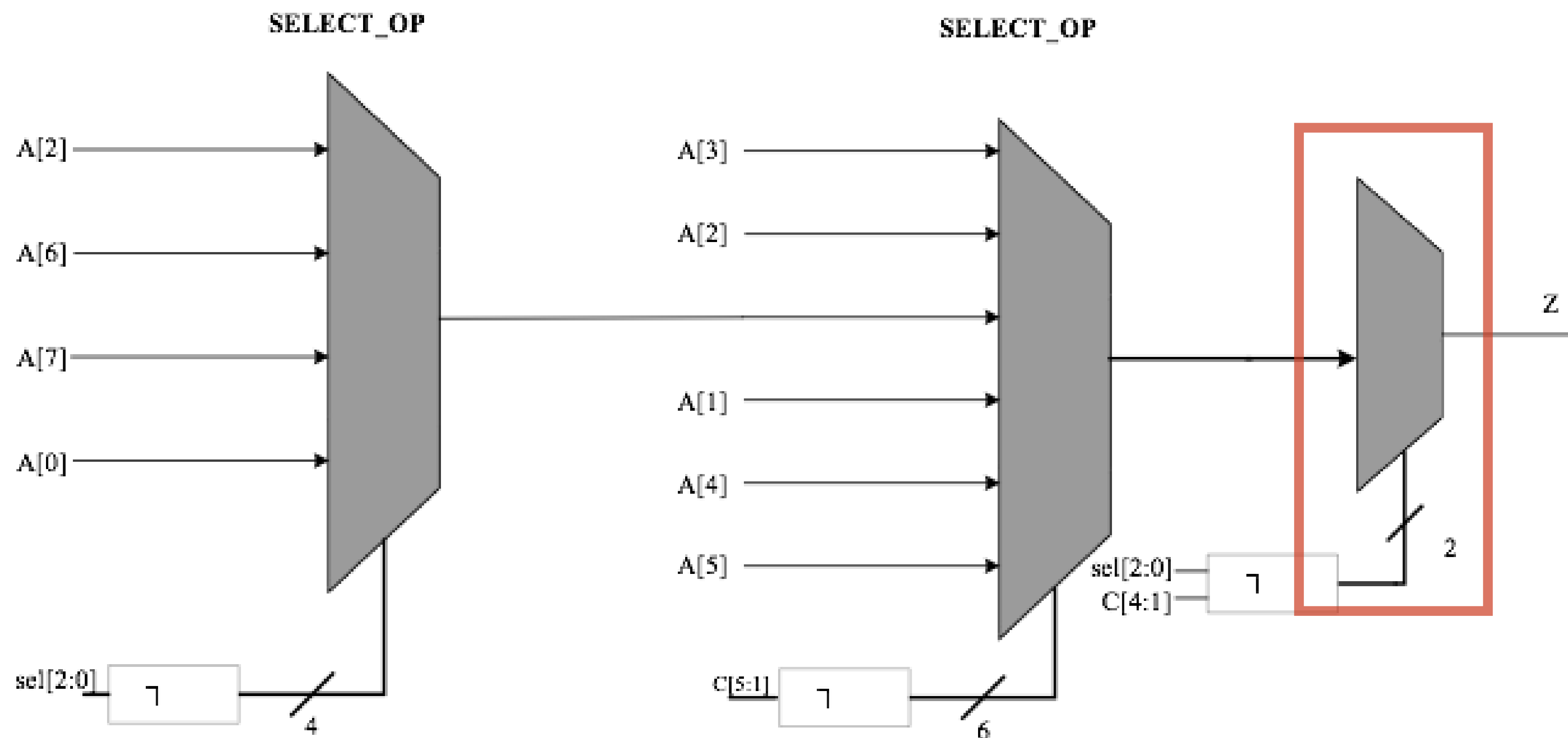
```
    else if (C[3])
        Z=A[1] ;
    else if (C[4])
        case (sel)
            3'b010:Z=A[8];
            3'b011:Z=DATA_is_late_arriving;
            3'b101:Z=A[7];
            3'b110:Z=A[6];
            default:Z=A[2];
        endcase
    else if (C[5]==1'b0)
        Z=A[2];
    else Z=A[3];
end
endmodule
```



对应的电路结构：



修改后:

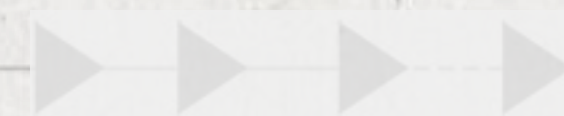


修改后:

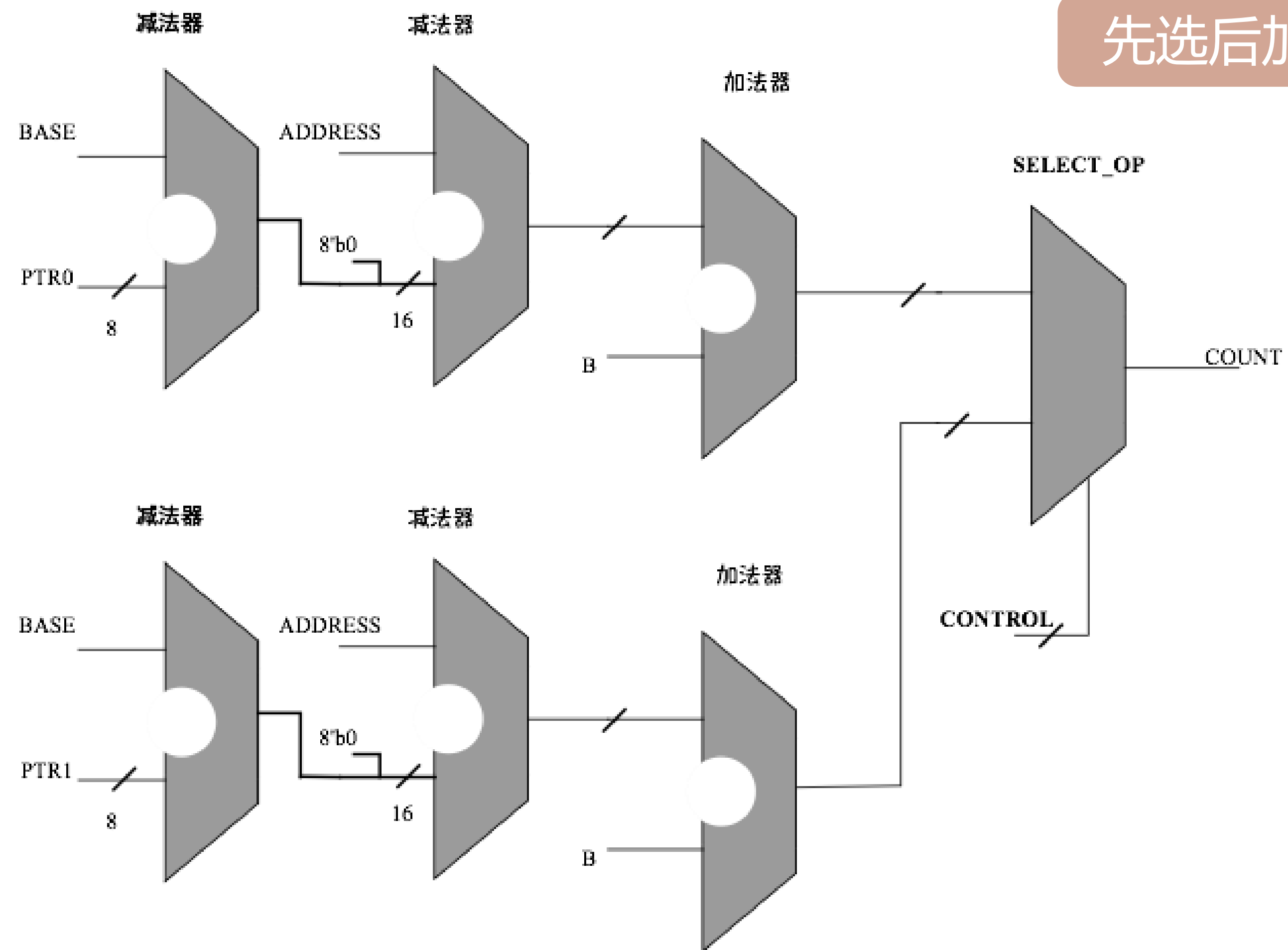
```
always @ (sel or C or A or DATA_is_late_arriving)
begin
    if (C[1])
        Z1=A[5] ;
    else if (C[2]==1'b0)
        Z1=A[4] ;
    else if (C[3])
        Z1=A[1] ;
    else if (C[4])
        case (sel)
            3'b010:Z1=A[8];
            //3'b011:Z1=DATA_is_late_arriving;
            3'b101:Z1=A[7];
            3'b110:Z1=A[6];
            default:Z1=A[2];
        endcase
    else if (C[5]==1'b0)
        Z1=A[2];
    else Z=A[3];
    FIRST_IF=(C[1]==1'b1) || (C[2]==1'b0) || (C[3]==1'b1);
    if (!FIRST_IF&&C[4]&&(sel==3'b011))
        Z=DATA_is_late_arriving;
    else Z=Z1 ;
end
```


信号CONTROL到达时间较晚，如何修改能够提高电路性能？

```
module BEFORE (ADDRESS, PTR1, PTR2, B, CONTROL, COUNT);  
input [7:0] PTR1, PTR2;  
input CONTROL; //CONTROL is late arriving  
output [15:0] COUNT;  
parameter [7:0] BASE=8'b10000000;  
wire [7:0] PRT,OFFSET;  
wire [15:0] ADDR;  
assign PTR = (CONTROL ==1'b1)?PTR1:PTR2;  
assign OFFSET=BASE-PTR;  
assign ADDR = ADDRESS-{8'h00-OFFSET};  
assign COUNT=ADDR+B;  
endmodule
```



- 复制数据路径，将CONTROL信号放到最后



- 复制数据路径，将CONTROL信号放到最后

```
assign OFFSET1=BASE-PTR1; //could be f(BASE,PTR)
```

```
assign OFFSET2=BASE-PTR2; //could be f(BASE,PTR)
```

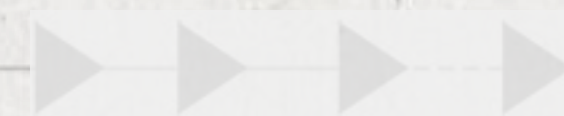
```
assign ADDR1 = ADDRESS-{8'h00-OFFSET1};
```

```
assign ADDR2 = ADDRESS-{8'h00-OFFSET2};
```

```
assign COUNT1=ADDR1+B;
```

```
assign COUNT2=ADDR2+B
```

```
assign COUNT=(CONTROL==1'b1)?COUNT1:COUNT2;
```

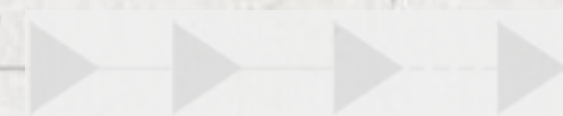


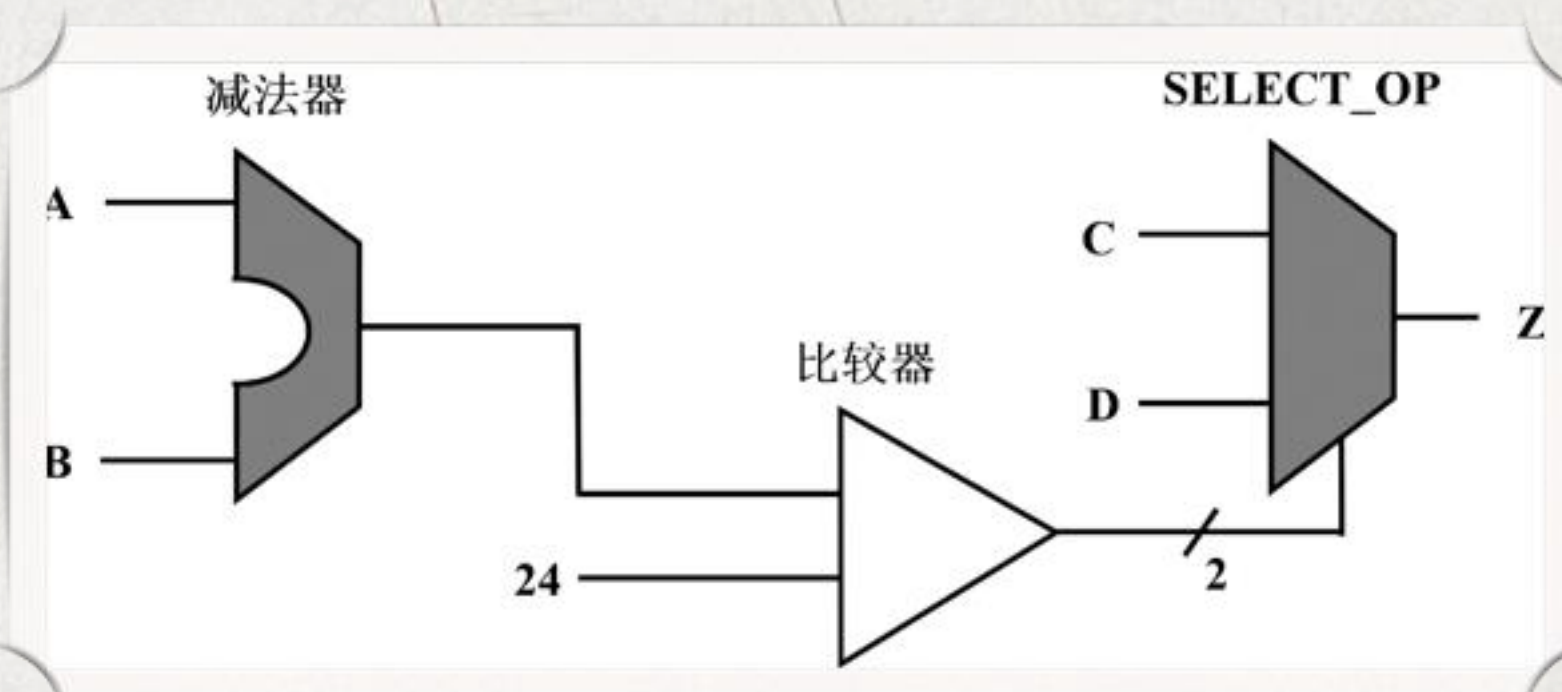
假设A信号到来较晚，如何修改能够提高电路性能？

```
module cond_oper(A,B,C,D,Z);  
parameter N=8;  
input [N-1:0] A,B,C,D; //A is late arriving  
output [N-1:0] Z;  
reg [N-1:0] Z;  
always @ (A,B,C,D)  
begin  
    if (A+B<24) Z<=C;  
    else Z<=D;  
end  
endmodule
```

一个加法器

一个比较器



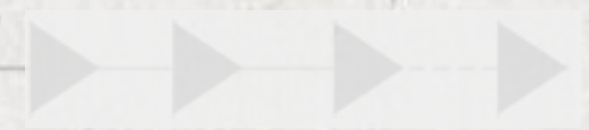
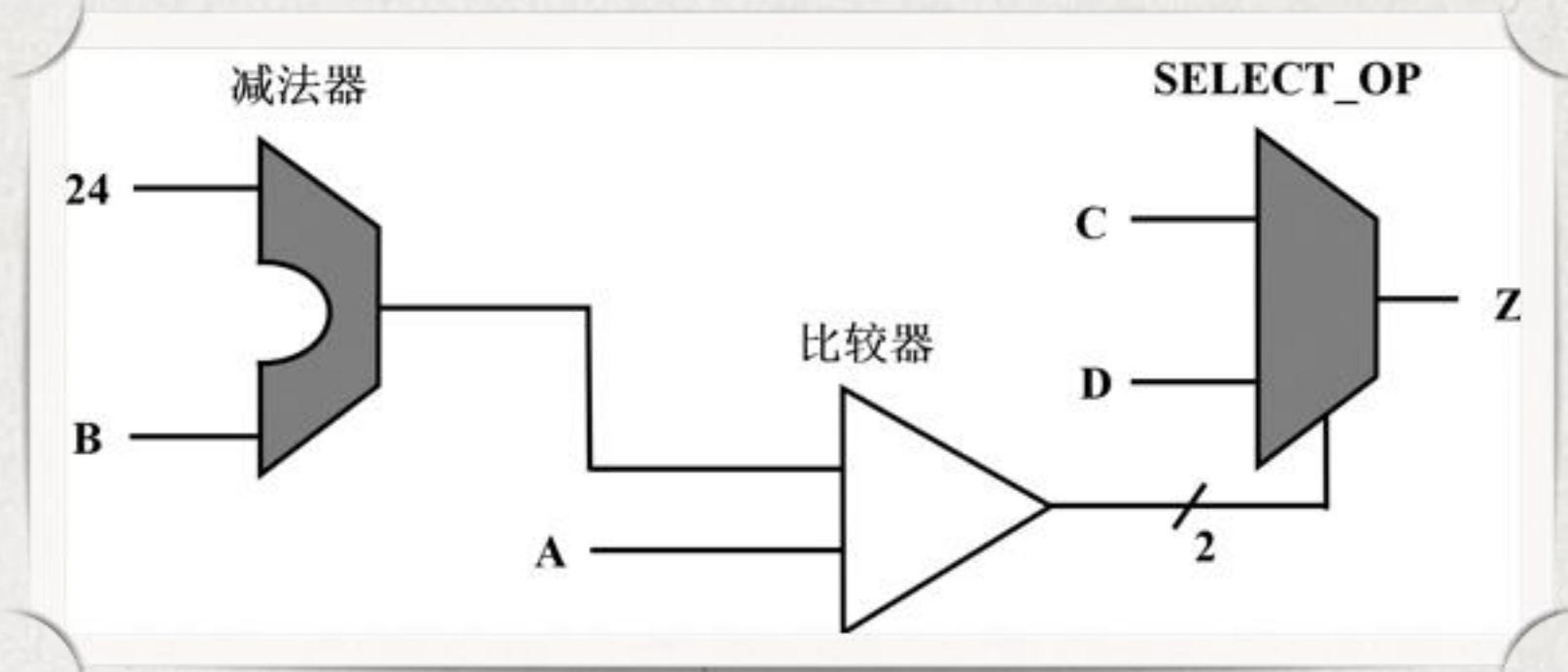


```

always @ (A,B,C,D)
begin
    if (A<24-B) Z<=C;
    else Z<=D;
end

```

调整A信号的计算顺序:



二、在RTL编码中考虑面积

- 随着芯片工艺的进步和生产成本的降低，面积显得没有时序问题重要。

减少设计面积:

- 成本降低、功耗降低
- 特别是对于FPGA的设计
- 直接决定着FPGA的选型

- 一般综合过程中可以对面积进行优化，但在RTL编码中如果注意节约设计的面积，往往可以达到事半功倍的效果。

减少设计面积:

- 估计设计使用资源的数量
- 知道设计中哪些部分占用了较大的面积

触发器

加法器

乘法器

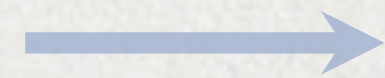
触发器的数量

- 由功能决定，很难减少

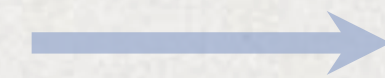
触发器的面积

- 比较好估计

组合逻辑



RTL代码



各种操作符

- RTL代码中的一个 “+ ” 可能对应着一个64位的加法器

[+]

[-]

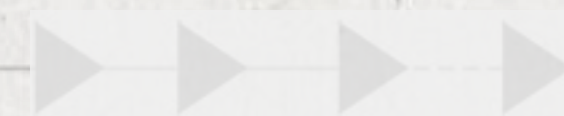
[×]

[÷]

条件语句中的比较运算

- 判断其必要性

- 是否能用更简单的运算代替



6' 10_0000

<32

A[5]=0

If (A<32)

B<=0;

A<=A+1;

Else

B<=1;

If (A[5]=1 'b1)

B<=1;

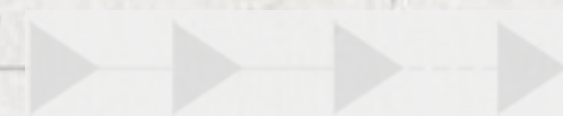
Else


B<=1;

A<=A+1;

"<" 修改为 "=", 可以节省一个6bit的比较器

- 如果，必须使用复杂的运算符，则应考虑是否可以资源共享。尽管电路逻辑综合工具也会在综合的过程中采用资源共享的方法进行优化，但是，综合器的策略是有限的，因此，在编写RTL的时候，应该尽量考虑共享，而不是把这项工作完全留给综合工具。

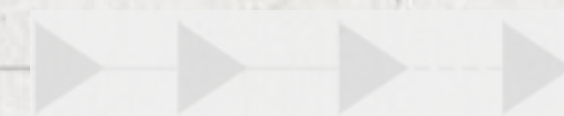




```
if(y1>a+b+q)
    statement1 ;
if(y2>a+b+r)
    statement2 ;
if(y3>a+b+s)
```

```
sum<=a+b;
if(y1>sum+q)
    statement1 ;
if(y2>sum+r)
    statement2 ;
if(y3>sum+s)
```

这样可以减少两个不必要的加法器，实际的设计中资源共享可能不会像这样明显，因此，平时应该按照这样的思路多多练习。



操作符

多比特

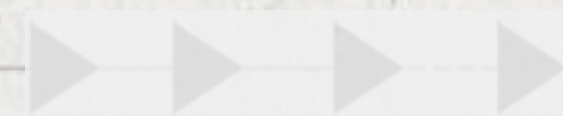
成倍使用资源

- 应该看一看这个信号的所有比特是否都需要参与操作，如果不是，则可以只对需要的部分比特进行操作。

访问一RAM的地址有8比特，而写入操作时从0开始，每隔32个地址写入一个值，地址的产生可以有两种写法。

```
addr<=addr+32;
```

```
addr[7:5]<=addr[7:5]+1;  
addr[4:0]<=addr[4:0]+0;
```





针对不同的设计

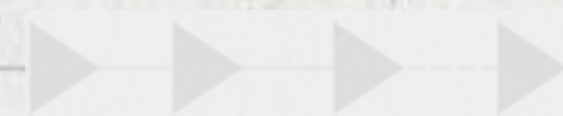
- 可能有各种各样的优化和改进的方法

编写代码

- 应对操作符有足够的重视
- 对有可能简化的地方尽量简化

逻辑简化

- 减少面积的同时也减少了延迟



三、在RTL编码中考虑功耗

$$P_d = \sum afCV^2$$

Pd - 电路割点的功耗总和

负载电容

工作电压

RTL设计无法改变的因素

a - 该点电路的翻转次数

f - 电路的工作频率

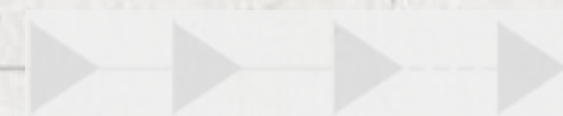
C - 该点的电容

V - 电压值

在RTL级主要考虑尽量降低电路的翻转频率

主要措施包括如下：

- 门控时钟
- 增加使能信号，使得部分电路只有在需要工作时才工作；
- 对芯片各个模块进行控制，在需要工作时才工作；
- 除了有用信号和时钟的翻转会消耗功耗，组合逻辑产生的毛刺也会大量消耗功耗。但是，毛刺在设计中无法避免，因此，只有尽量减少毛刺在电路中的传播，才可以减少功耗。即，在设计中，尽量把产生毛刺的电路放在传播路径的最后。另外，可以使用一些减少毛刺的技术。



主要措施包括如下：

- 对于有限状态机，可以通过低功耗编码来减少电路的翻转。
4比特状态编码

状态A	0101
状态B	0100

每次状态转移使得4个比特信号发生翻转

- 考虑全局的功耗控制；
- 在RTL编码中，注意消耗功率较多的电路；
- 在综合中，使用门控时钟和其他减少功耗的优化技术。

状态机
译码器
多路选择器

四、在RTL编码中考虑布线问题

布线 (routing)

最后的阶段

功能

- 根据门级网表 的描述实现各个单元的连接

布局 (placement)

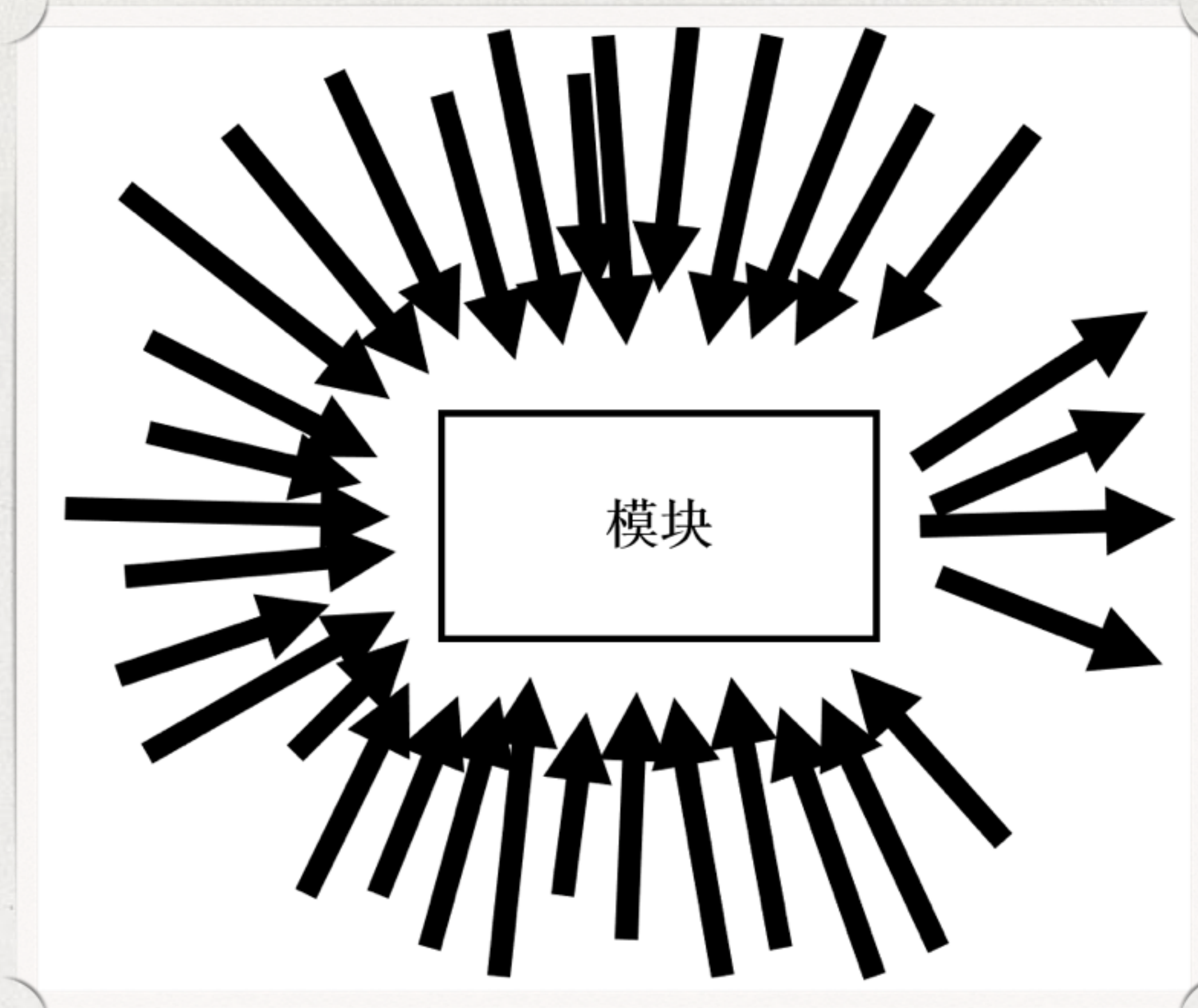
最关键的因素

修改RTL级设计

- 但即使使用最好的布局工具，还是可能出现无法布通的情况。

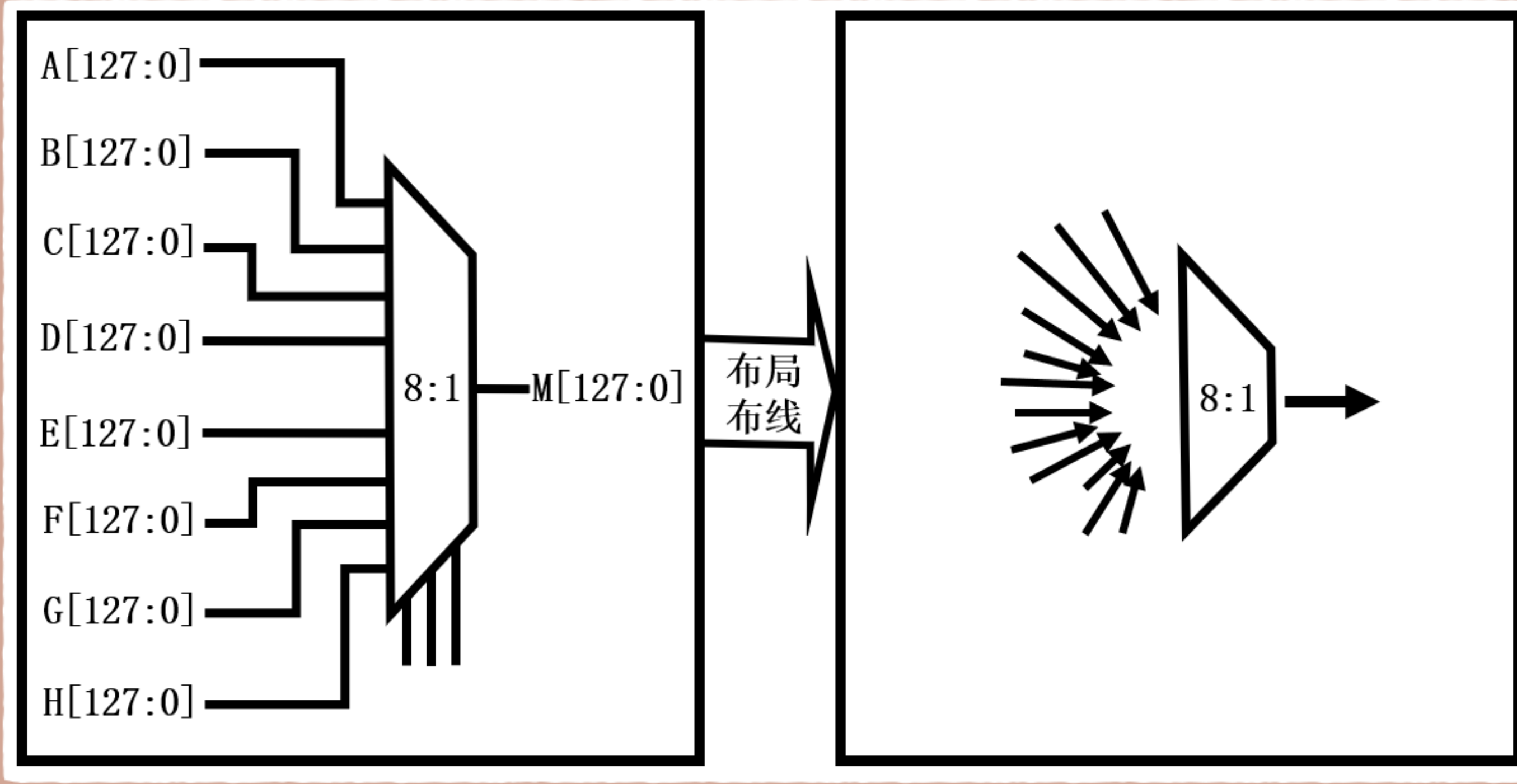
如果可以在RTL编码阶段考虑代码可能对布线产生的影响
就可能避免最后出现无法布通的情况

- 热点是指设计的功能需要在一个面积内占用大量的布线资源



- 热点产生原因：RTL编码时使用了特定的结构，如很大的MUX

```
always @ (A or B or C or D or E or F or G or H or SEL)
begin
  case(SEL)
    3'b000:M = A;
    3'b001:M = B;
    3'b010:M = C;
    3'b011:M = D;
    3'b100:M = E;
    3'b101:M = F;
    3'b110:M = G;
    3'b111:M = H;
  end
```



- 这种结构产生的热点，在综合的时候，导致的延迟是看不出来的，只有到了布线阶段才能给看到它的负面影响。因此，我们在RTL阶段应该重视这种电路，及早发现可能在布线阶段产生的问题。
- 如果设计的功能中确实需要采用大的mux，可以通过其他方式改变他的结构。其基本的思路是将一个大的mux分解为多级较小的mux。

