北京郵電大學
Beijing University of Posts and Telecommunications

硬件编程语言思想

北京邮电大学 集成电路学院
赵康

参考书目

1. **《Verilog HDL 数字设计与综合》**，Sanir Palnitkar著，夏宇闻，胡燕祥，刁岚松译，电子工业出版社，2009.
2. **《Verilog HDL高级数字设计（第二版）》**，西勒提（Michael D.Ciletti）著，李文军，林水生，阎波译，电子工业出版社，2014.

硬件编程语言思想

1. HDL概述（以Verilog为例）
2. 电路描述基础
3. 流水线与存储器
4. 高质量硬件描述方法

HDL概述

关于 HDL的认知

对Verilog/VHDL的误解

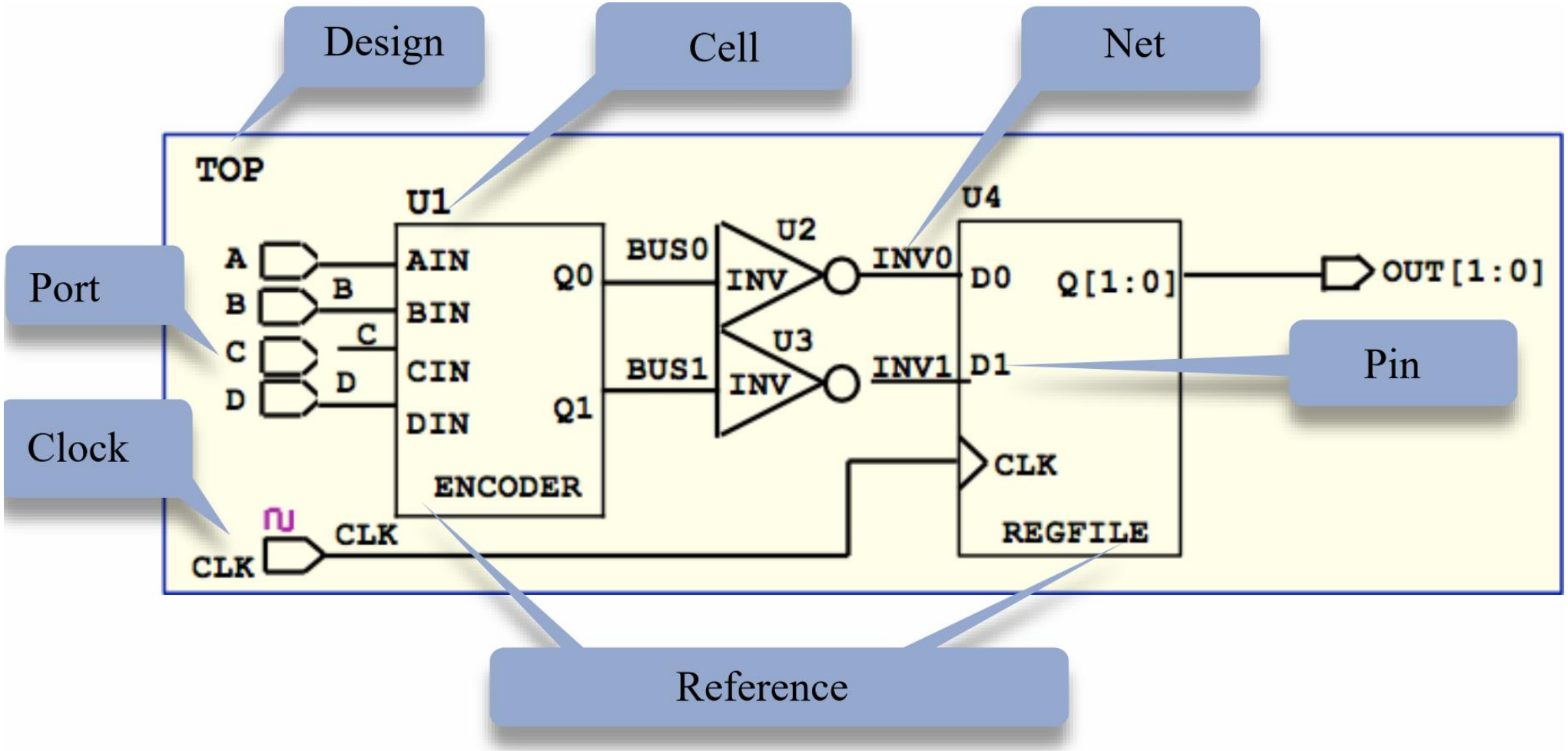
- 很多语法规则与C语言相似，书写时可参考C语言
- 追求代码的整洁、简短
- 着眼于代码书写，性能优化由综合器实现
- 把Verilog/VHDL代码当做了程序，把电路设计当成了编程

对Verilog/VHDL的正确认知

- Verilog/VHDL是一门描述语言：已知硬件电路的文本表示形式
- 硬件架构需要人工设计及优化

HDL概述

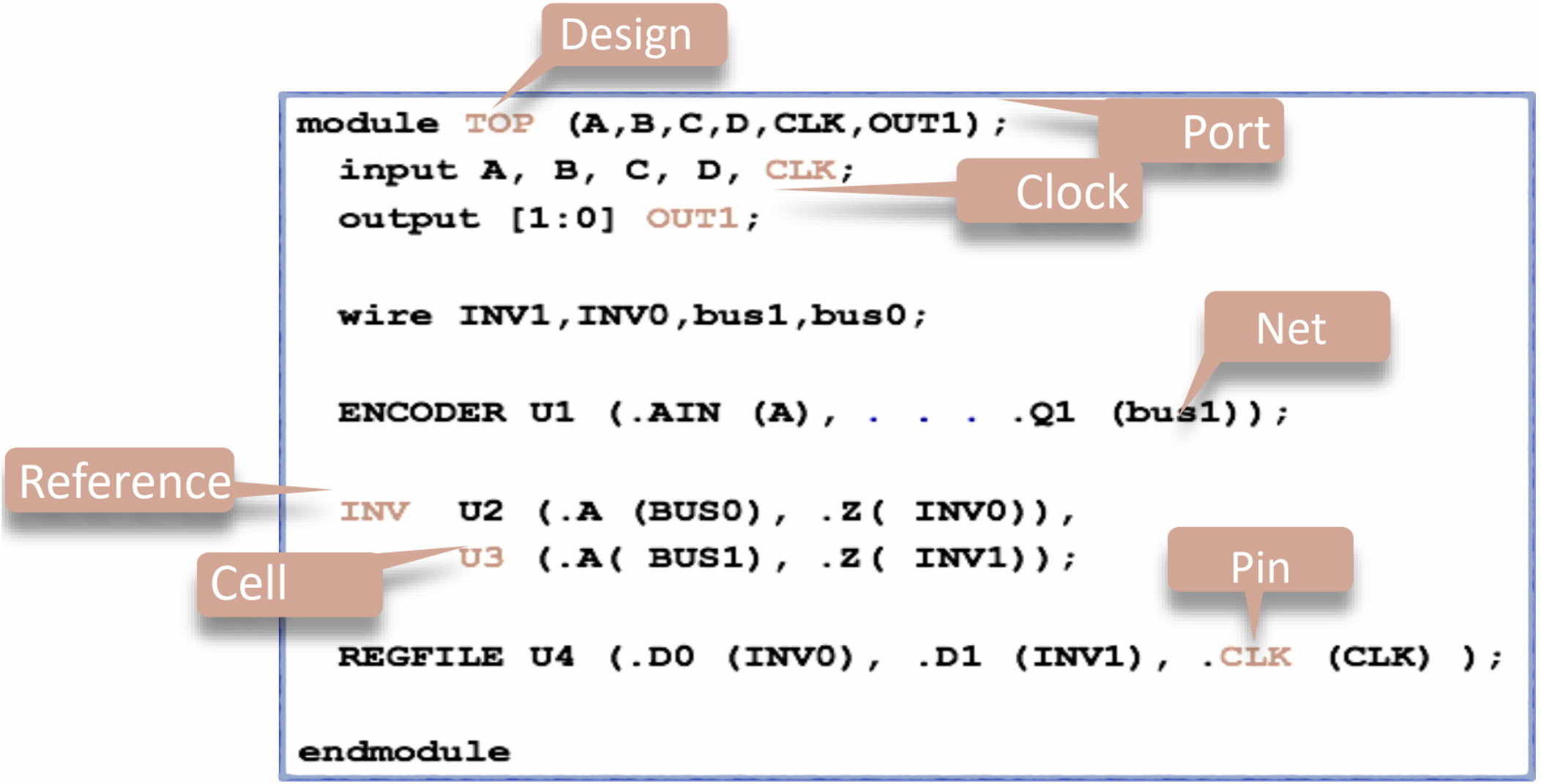
关于Verilog/VHDL HDL的认知



Verilog不是编程语言；用来“描述”电路，先确定（想好）电路设计再用Verilog表达出来

HDL概述

关于Verilog HDL的认知



HDL概述

硬件设计的基本概念

VerilogHDL具备从“抽象表达”到“门级连接”的多层次表征的能力

Behavioral

```
module cter (
       input  rst, clock,
       output reg  [1:0] count
             );
always@(posedge clock)
begin
 if (rst) count = 0;
 else     count = count +1;
end
endmodule
```

Structural

```
module cter ( rst, clock, count );
  output [1:0] count;
  input rst, clock;
  wire   N5, n1, n4, n5, n6;
  FFD U0  (.D(N5), .CP(clock),
           .Q(count[0]), .QN(n6));
  FFD U1  (.D(n1), .CP(clock),
           .Q(count[1]), .QN(n5));
  MUX21 U2 (.A(N5), .B(n4),
            .S(n5), .Z(n1) );
  NR U3 (.A(n6), .B(rst), .Z(n4));
  NR U4 (.A(count[0]), .B(rst),
          .Z(N5));
endmodule
```

HDL概述

硬件设计的基本概念

HDL具备从“抽象表达”到“门级连接”的多层次表征的能力

互联（connectivity）

wire型变量描述各个模块之间的端口与网线连接关系

并发（concurrency）

可以有效地描述并行的硬件系统

时序（timing）

定义了绝对和相对的时间度量，可综合操作符具有物理延迟

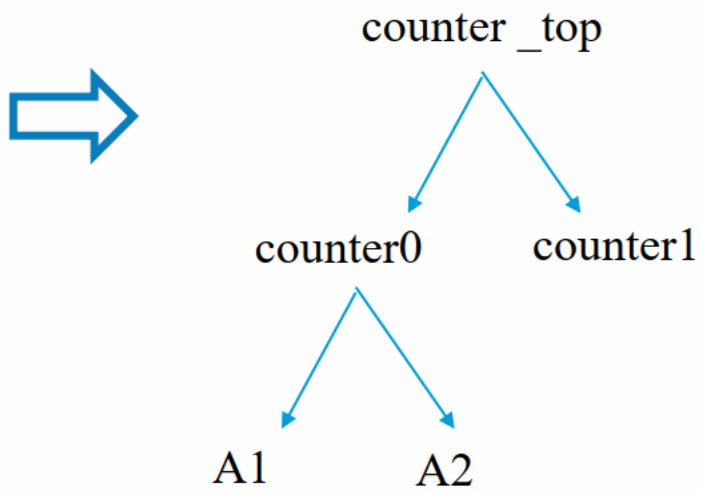
组合逻辑的延迟、时序逻辑的时钟

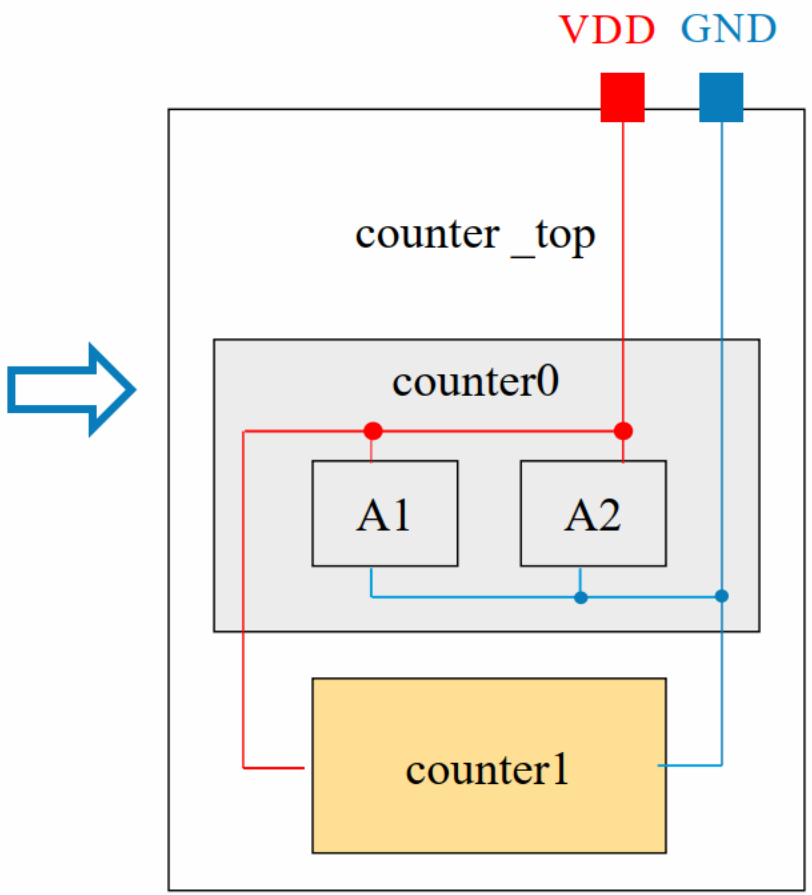
HDL概述

互联特性

```
module counter_top(cin,clk,cout,q);
……
counter    counter0( .cin(cin),  .clock(clk),  .cout(cout0),  .q(q[3:0]));
counter    counter1( .cin(cout0), .clock(clk), .cout(cout),  .q(q[7:4]) );
……
endmodule
```

```
module counter(xxx, xxx, xxx);
……
and A1 (a,b,c);
and A2 (a,b,c);
……
endmodule
```





并发执行与顺序执行

- **module**中，所有描述语句（包括连续赋值语句、行为语句块always、initial以及实例化等）都是并行发生的
- **begin ... end**中存在的语句是顺序执行的

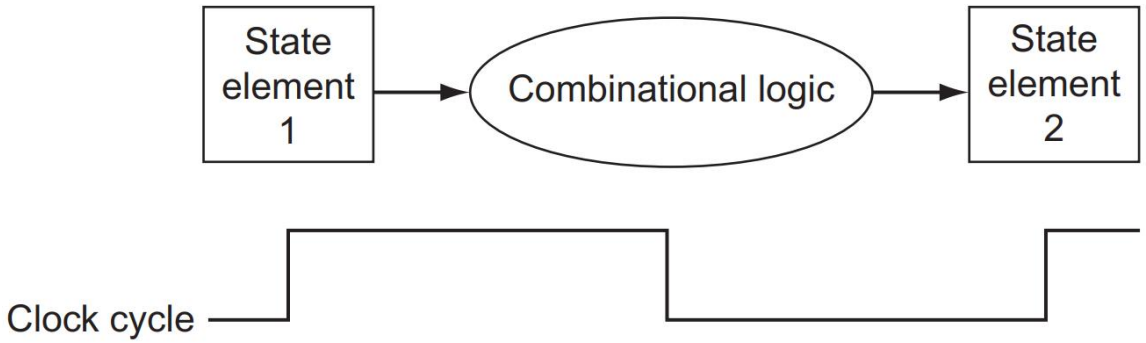
```
module  TOP ;

always  @  (*)
begin
   if   ()
   if   ()
end

assign  a = b ;

endmodule
```

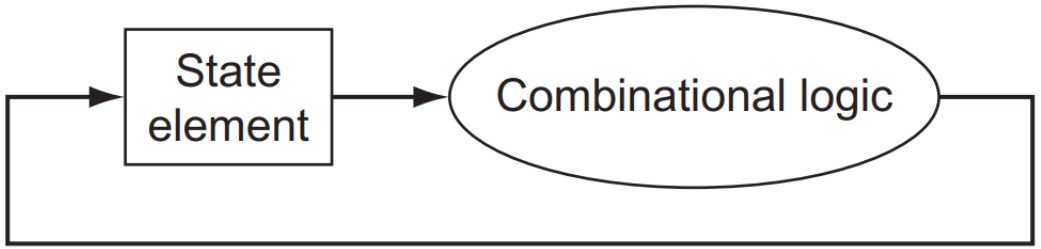
- always, assign 之间并行
- begin…end 内部串行

时钟与时序

组合逻辑在时钟周期中进行数据变换

- 在时钟边沿到来时进行输入数据更新
- 从状态单元输入， 输出结果存入状态单元
- 组合逻辑的最长延迟决定了时钟周期





HDL概述

可综合描述的语句（以verilog为例）

可综合“四大法宝”

- always
- if-else
- case
- assign

不可或不利于综合的“迷惑行为”

- function
- for
- fork-join
- while

testbench

硬件编程语言思想

1. HDL概述
2. 电路描述基础
3. 流水线与存储器
4. 高质量硬件描述方法

电路描述基础

变量（以verilog为例）

- **wire**型：表示电路模块中的物理连线
- **reg**型：占用仿真环境中的物理内存

注意事项

- 凡是在always、initial语句中赋值的变量，一定是reg型
- 凡是在assign语句中赋值的变量，一定是wire型
- reg变量仅仅是语法定义， 不等于电路中寄存器
- 只有时序电路中的reg变量才会被逻辑综合工具认为是寄存器

电路描述基础

assign

- 连续驱动赋值
- 赋值的对象应该是wire类型
- “=”右边的任何变化都会被立即计算并驱动给等号左边
- 用于对信号连接，重命名，简单组合逻辑

```
assign a = b | c;
assign a = b ? c : d;
assign a = d;
assign a = e[3:2];
assign a = (b | c) & (d ^ e);
```

电路描述基础

always

- 后接敏感列表，用@表示
- always@(a or b or c)表示只要a,b,c中有一个产生变换，则执行该always块
- 一般包含begin…end语句组
- always@(*)表示自动将该always块中所有引用的信号都自动添加到敏感列表中
- always@(posedge clk or negedge rst_n)代表只在clk上升沿或rst_n下降沿上执行该always块 ——寄存器，时序逻辑
- 千万不要混合组合逻辑和时序逻辑： always@(posedge clk or a or b)

always@(posedge clk or posedge rst)

always@(posedge clk)

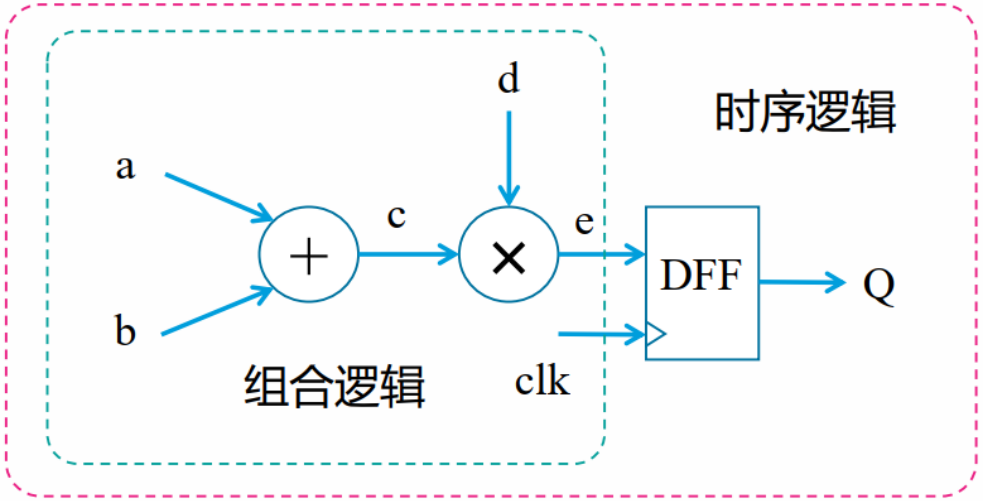
always@(a or b or c)

时序逻辑

组合逻辑：虽然使用reg型，但无保存/锁存数据的功能

电路描述基础

阻塞赋值（=）与非阻塞赋值（<=）



- 组合逻辑 必须使用 “=”
- 时序逻辑 必须使用 “<=”

- 时序逻辑必须要复位
- 每加一个DFF，输出就推迟1个时钟周期

```
always  @ ( * )
begin
   c  = a+b  ;
   e  = c * d ;
end
```

```
always  @ (posedge clk  or  negedge rst_n )
begin
   if (rst_n == 1'b 0)
      Q  <= 0 ;
   else
      Q  <= e  ;
end
```

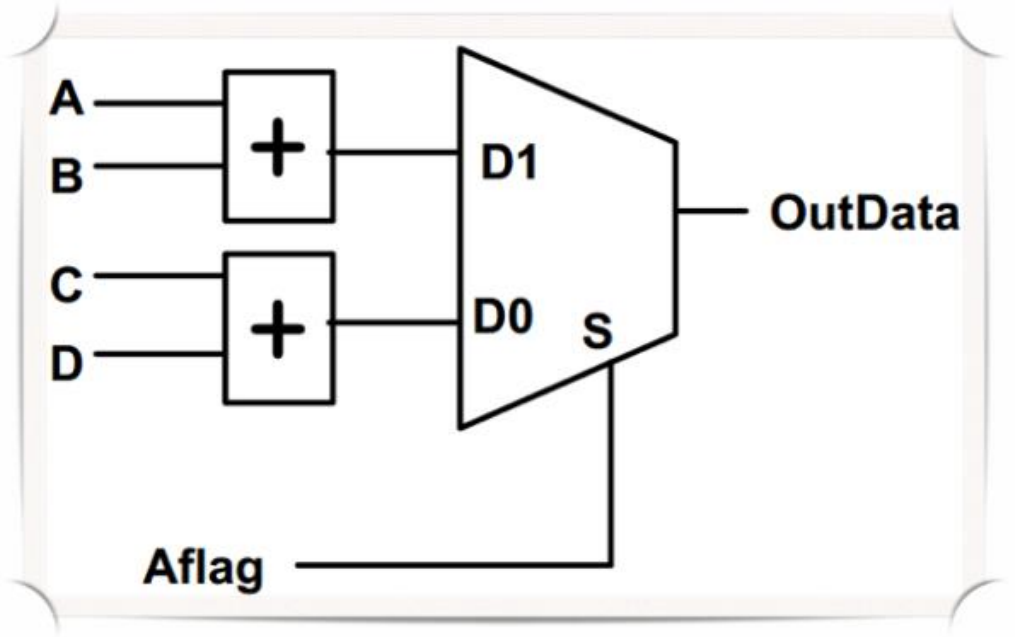
等价

```
always  @ (posedge clk  or  negedge rst_n )
begin
   if (rst_n == 1'b 0)
      Q  <= 0 ;
   else
      Q  <= (a+b) * d   ;
end
```

电路描述基础

多路选择器（MUX）

```
always @ ( * )
begin
    if (aflag == 1'b 1)
        outdata = A+B;
    else
        outdata = C+D;
end
```

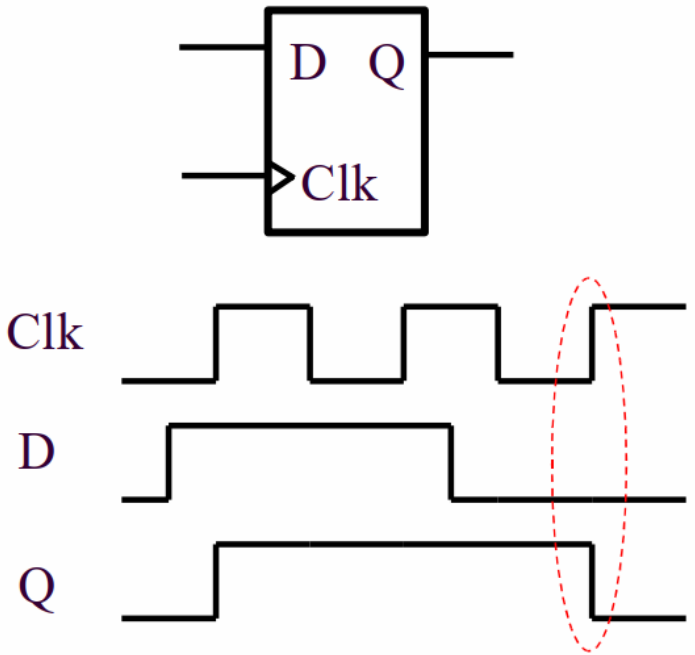


电路描述基础

寄存器 & 锁存器

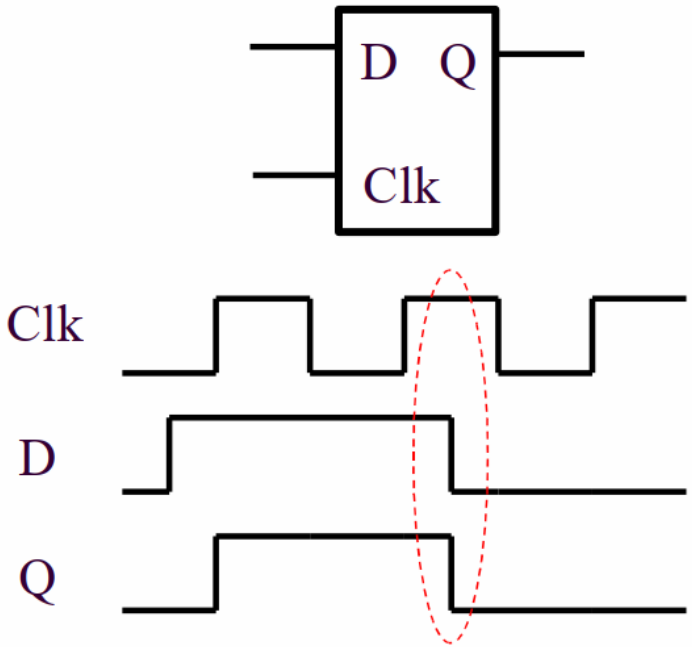
寄存器

- Register、flip-flop（FF）
- 边沿触发
- 输入 - 输出不透明



锁存器

- Latch
- 电平敏感
- 输入 - 输出透明



电路描述基础

Latch出现：组合逻辑中有不完备的条件判断语句

缺少else

```
always @(cond1 or data in)
begin
   if(cond1==1 )
      data_out=data_in;
end
```

缺少default

```
casex ({sel0,  sel1,  se12})
   2'd0: z=d;
   2'd1: z=c;
   2'd2: z=b;
endcase
```

防止产生非目的性latch的措施

- 使用完备的if…else语句
- 为case语句设置default操作
- always敏感列表写全
- 综合器中，latch会以warning的形式报告

电路描述基础

Latch出现：组合逻辑中有不完备的条件判断语句

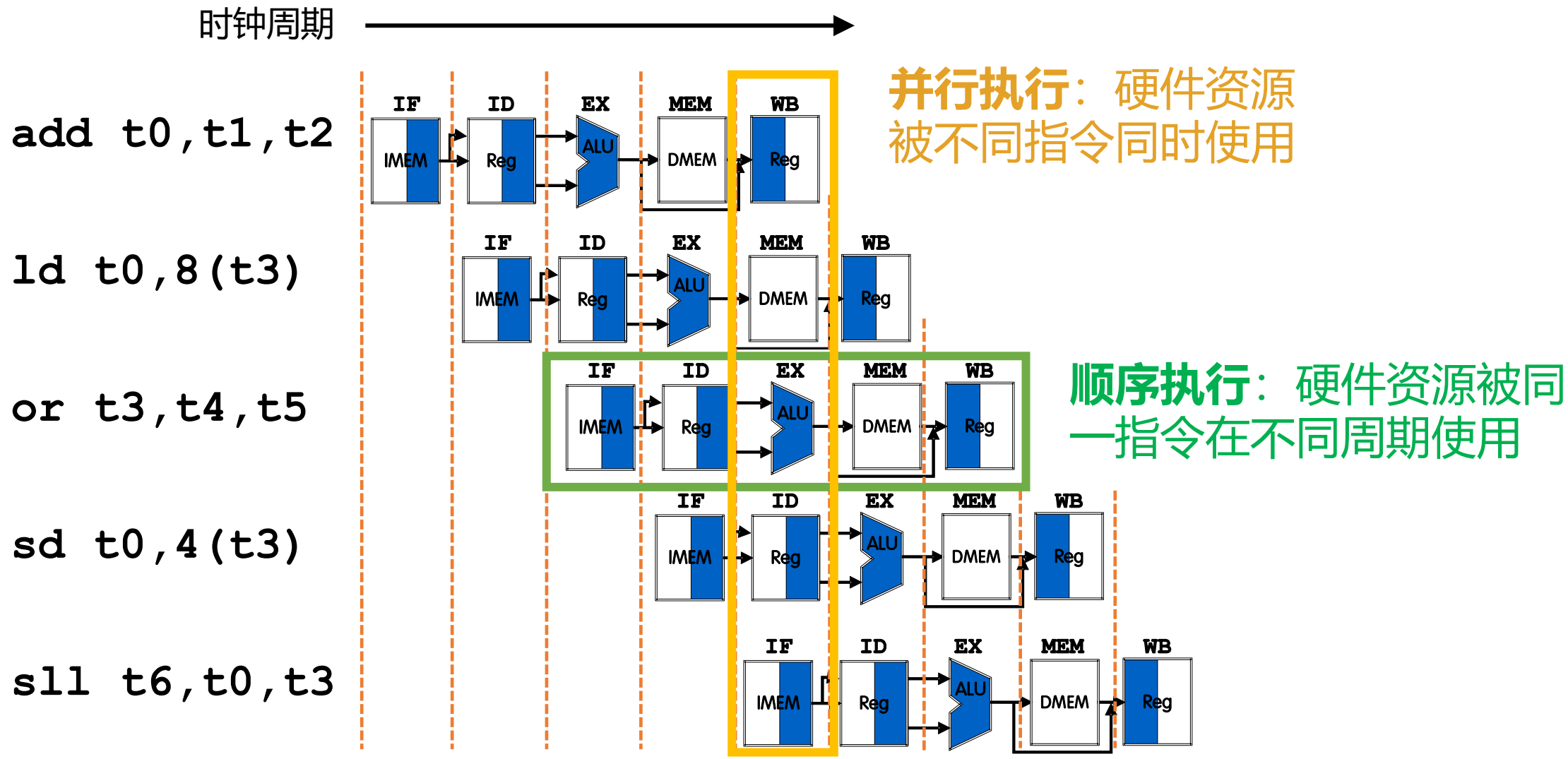
- Latch由电平触发，非同步控制
- Register由边沿触发，同步控制
- Latch容易产生毛刺，register不容易产生毛刺（采样只有一瞬间）
- 一般设计规则：尽量使用register，避免使用latch，因为不能过滤毛刺，带到下一级电路是危险行为

硬件编程语言思想

1. HDL概述
2. 电路描述基础
3. 流水线与存储器
4. 高质量硬件描述方法

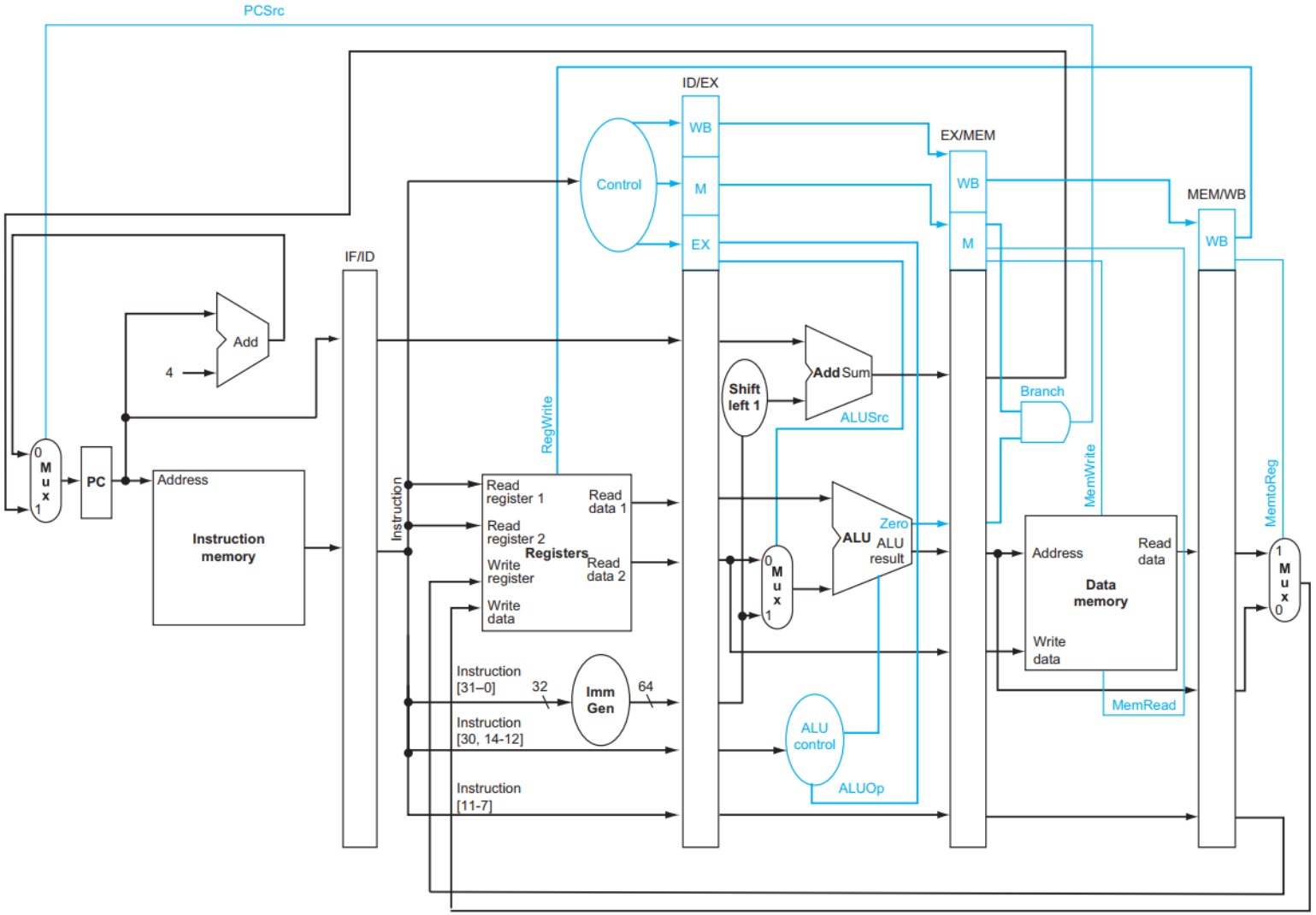
流水线与存储器

RISC-V处理器流水线



流水线与存储器

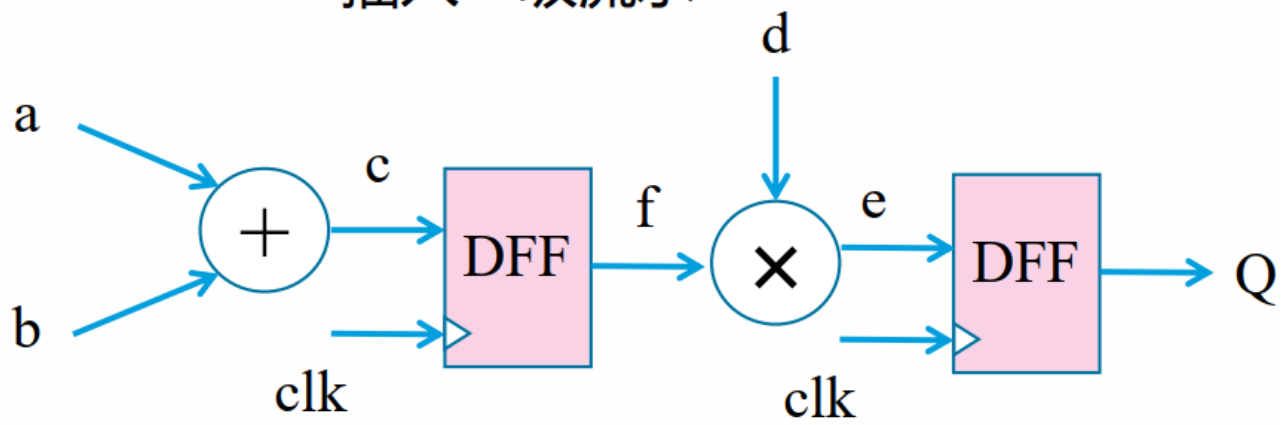
RISC-V处理器流水线



流水线与存储器

数字电路的流水线设计

插入一级流水




```
always  @ (posedge clk  or  negedge rst_n )
begin
   if (rst_n == 1'b 0)
      f <= 0 ;
   else
      f  <= a + b ;
end
```

```
always  @ (posedge clk  or  negedge rst_n )
begin
   if (rst_n == 1'b 0)
      Q  <= 0 ;
   else
      Q  <= f * d   ;
end
```

流水线与存储器

存储器

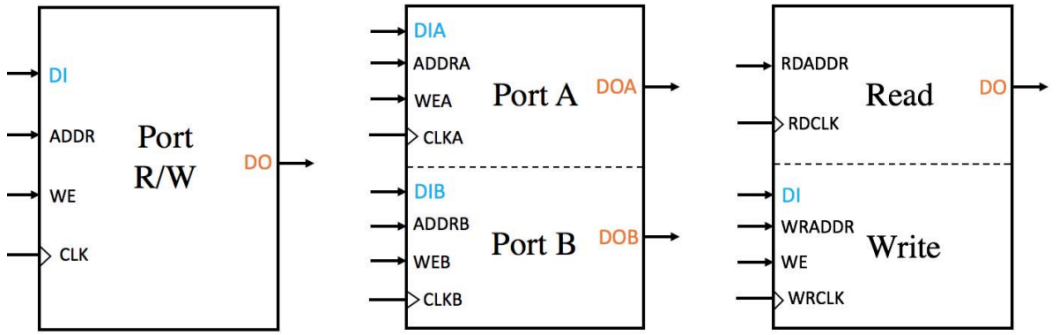
- 逻辑电路设计会经常使用一些单口RAM、双口RAM、ROM、FIFO等类型的存储器
- Verilog 语法中基本的存储单元定义格式如下：

  reg [datawidth-1:0] MemoryName [addresswidth-1:0];

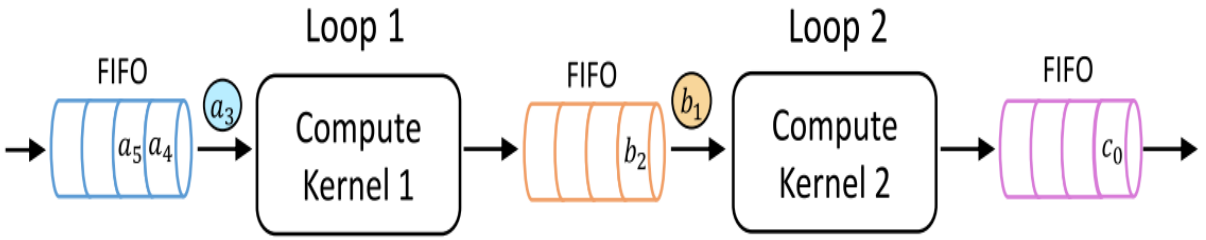
- 但是一般情况下不采用自定义的RAM/FIFO等存储器，而应使用EDA厂商提供的IP进行实例化

Storage of block RAM	SP mode	TDP mode	SDP mode	
18 K-bit	36	18	36	最大数据位宽
36 K-bit	72	36	72	

RAM



FIFO



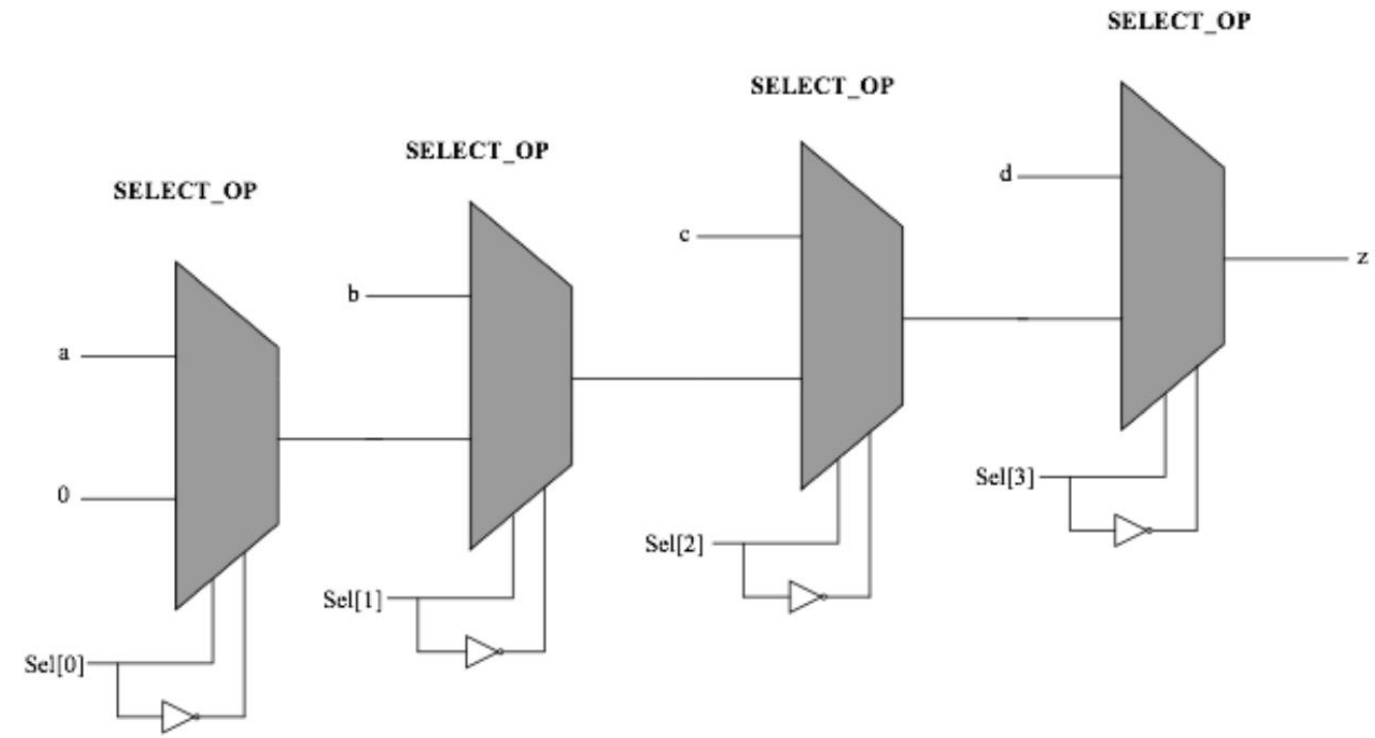
硬件编程语言思想

1. HDL概述
2. 电路描述基础语法
3. 流水线与存储器
4. 高质量硬件描述方法

高质量硬件描述方法

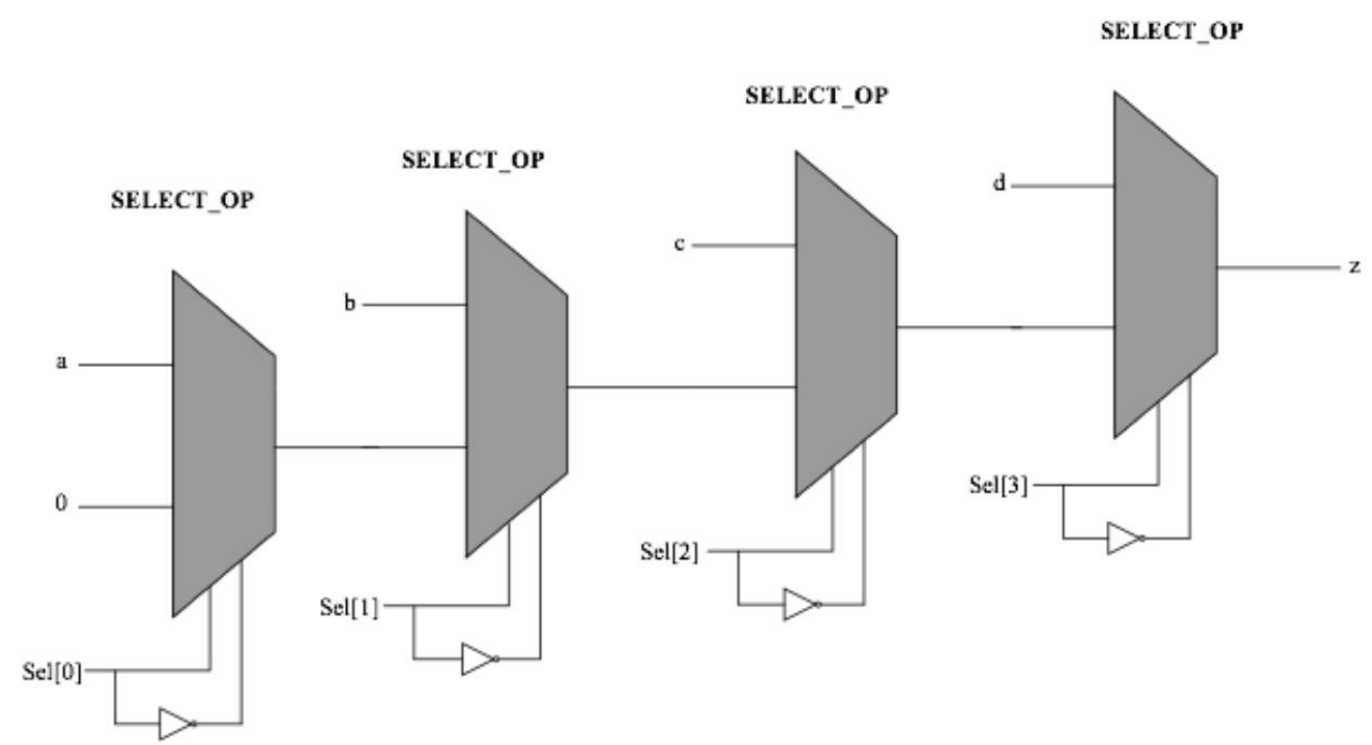
① 在RTL代码编写过程中考虑时延

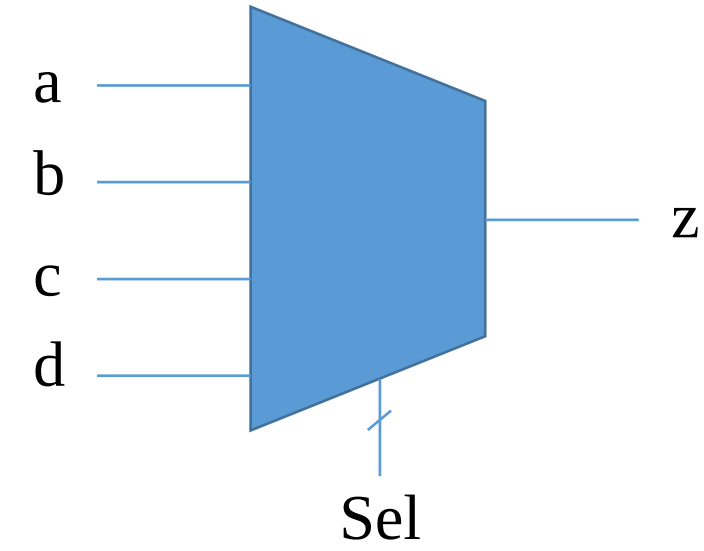
```
module mult_if (a,b,c,d,sel,z)
input          a,b,c,d  ;
input   [3:0]   sel     ;
output          z       ;
reg             z       ;
always @ ( * )
begin
   z = 1'  b0           ;
   if (sel[0]) z = a    ;
   if (sel[1]) z = b    ;
   if (sel[2]) z = c    ;
   if (sel[3]) z = d    ;
end
endmodule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



高质量硬件描述方法

① 在RTL代码编写过程中考虑时延：平衡各路径的延迟

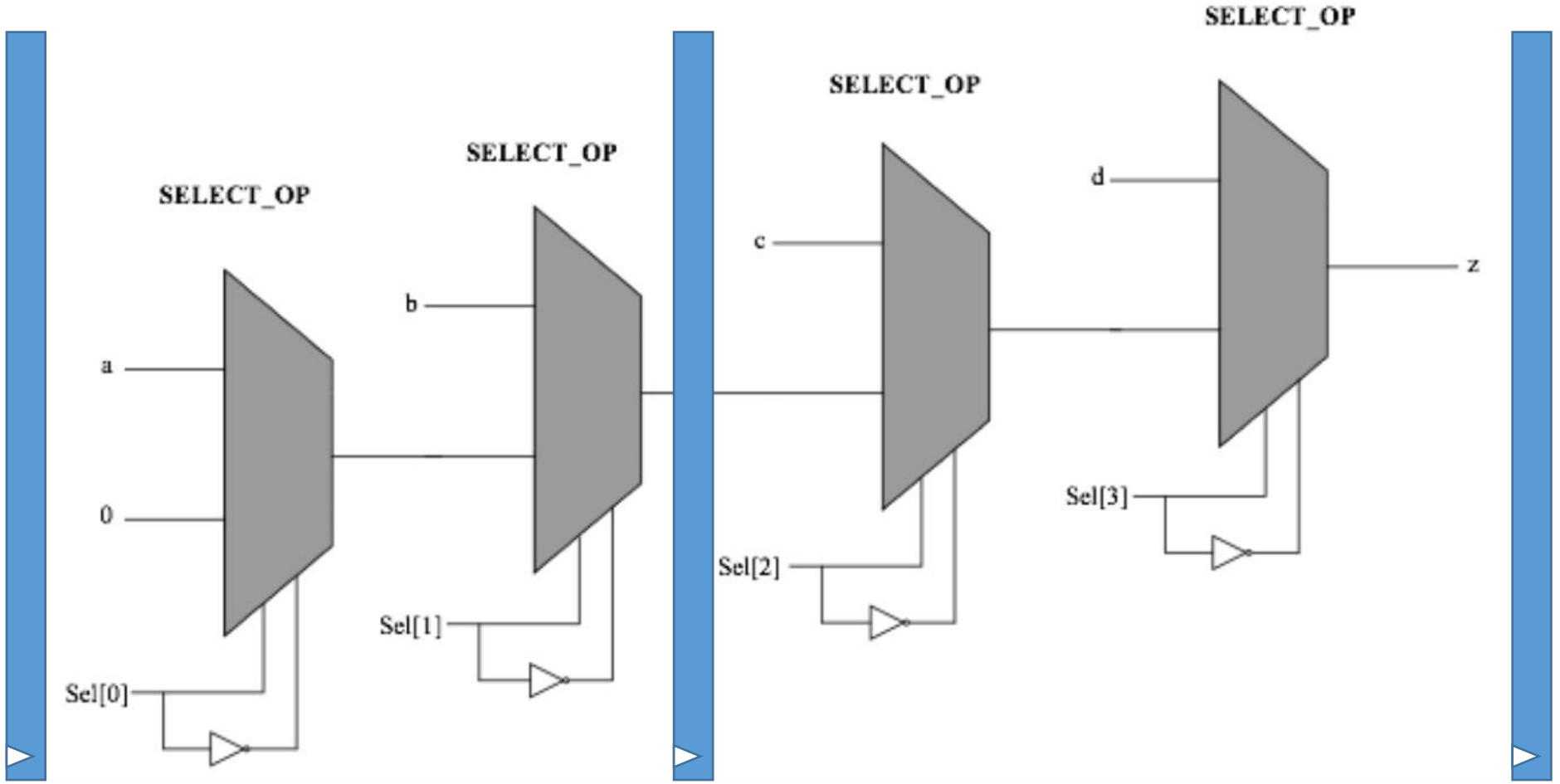




if … else if … else …

高质量硬件描述方法

① 在RTL代码编写过程中考虑时延：减少关键路径延迟



高质量硬件描述方法

② 信号延迟较大，将其使用时机延后：控制信号

```
module  BEFORE (ADRESS, 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 = ADRESS-{8'h00-OFFSET};
assign COUNT=ADDR+B;
endmodule
```

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 = ADRESS-{8'h00-OFFSET1};
assign ADDR2 = ADRESS-{8'h00-OFFSET2};
assign COUNT1=ADDR1+B;
assign COUNT2=ADDR2+B
assign COUNT=(CONTROL==1'b1)?COUNT1:COUNT2;
```

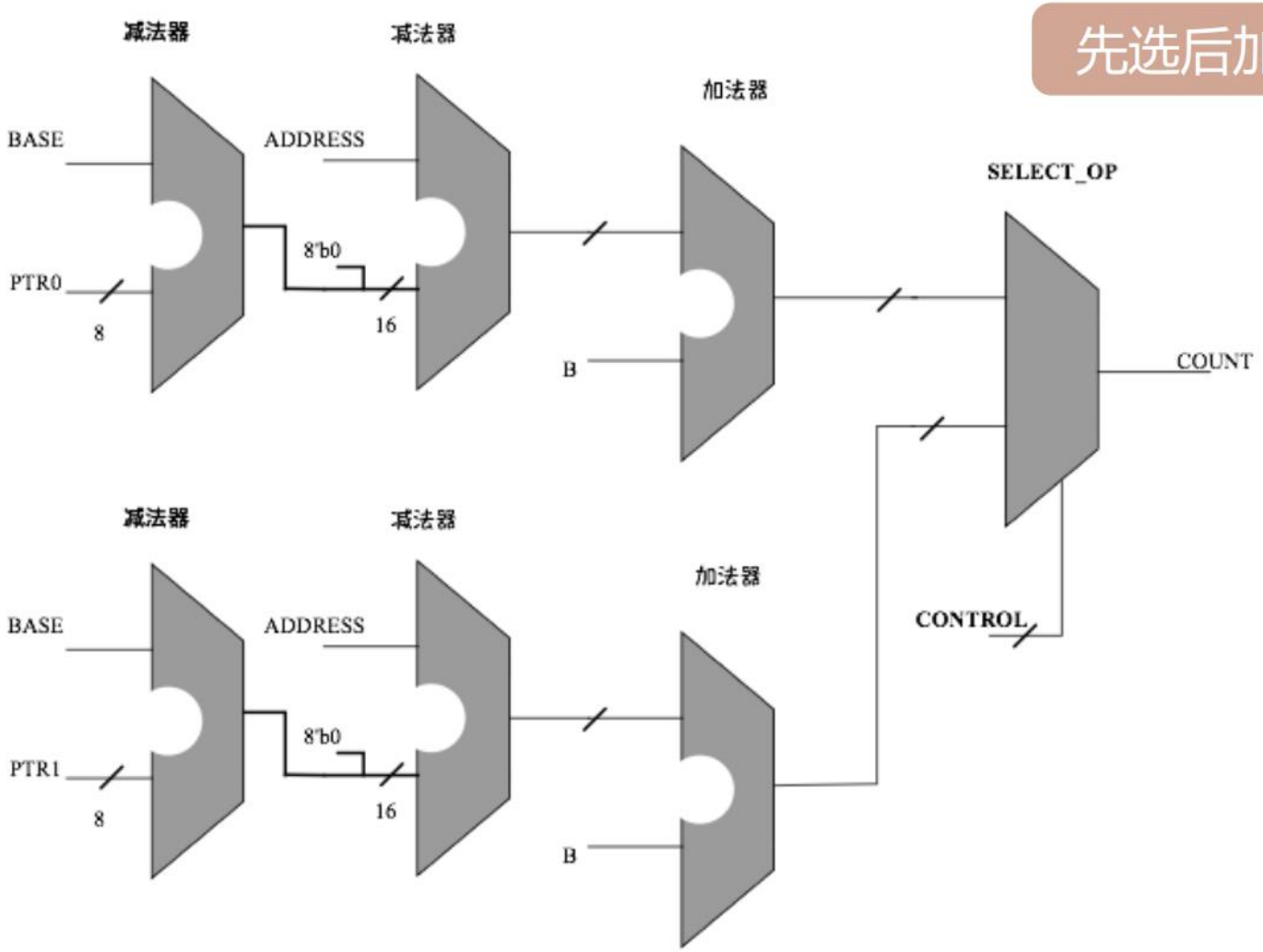
高质量硬件描述方法

② 信号延迟较大，将其使用时机延后：控制信号

```
assign PTR = (CONTROL ==1'b1)?PTR1:PTR2;
assign OFFSET=BASE-PTR;
assign ADDR = ADRESS-{8'h00-OFFSET};
assign COUNT=ADDR+B;
```

```
assign OFFSET1=BASE-PTR1; //could be f(BASE,PTR)
assign OFFSET2=BASE-PTR2; //could be f(BASE,PTR)
assign ADDR1 = ADRESS-{8'h00-OFFSET1};
assign ADDR2 = ADRESS-{8'h00-OFFSET2};
assign COUNT1=ADDR1+B;
assign COUNT2=ADDR2+B
assign COUNT=(CONTROL==1'b1)?COUNT1:COUNT2;
```

先选后加



高质量硬件描述方法

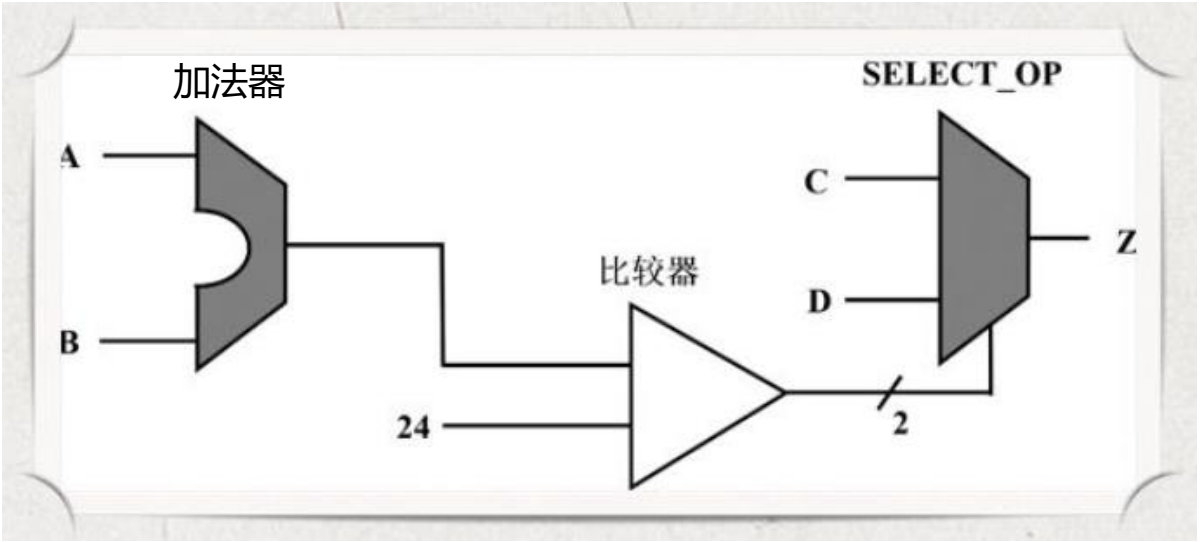
② 信号延迟较大，将其使用时机延后：数据信号

假设信号A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

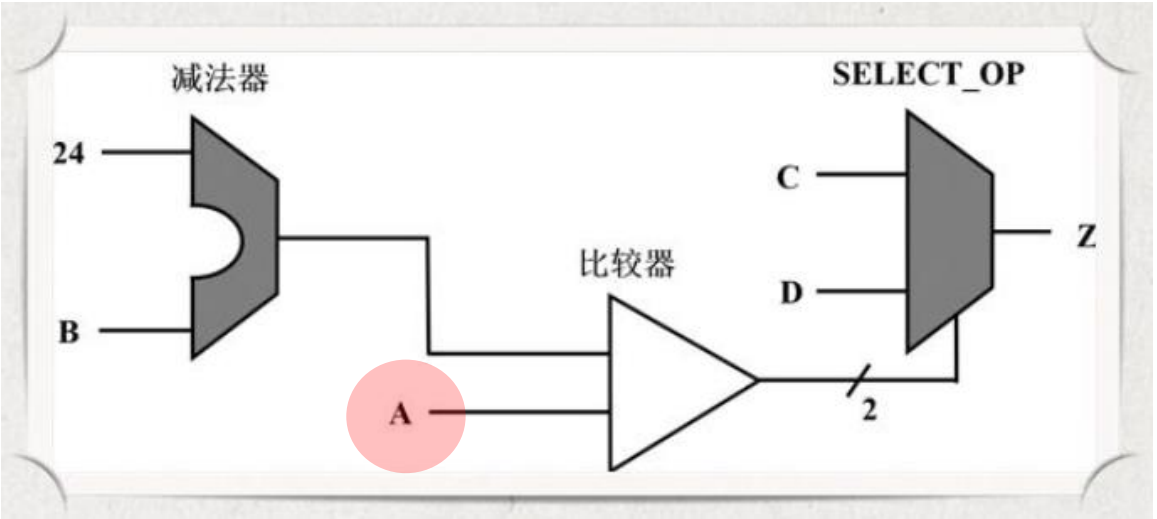


高质量硬件描述方法

② 信号延迟较大，将其使用时机延后：数据信号

假设信号A到来较晚，如何修改能够提高电路性能?

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

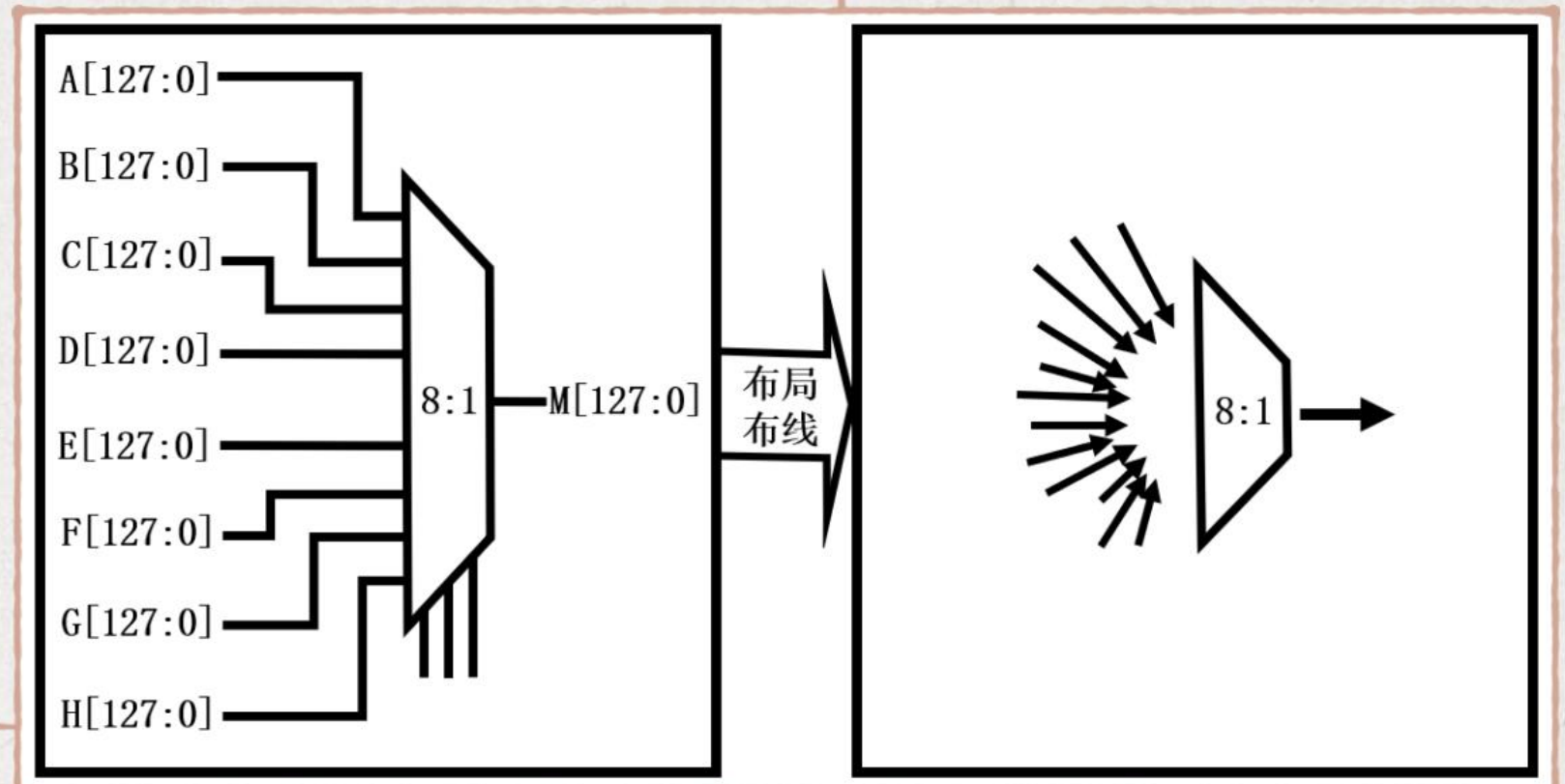


高质量硬件描述方法

③ 减少大的扇入/扇出

扇入/扇出：一个单元的输入/输出信号数量

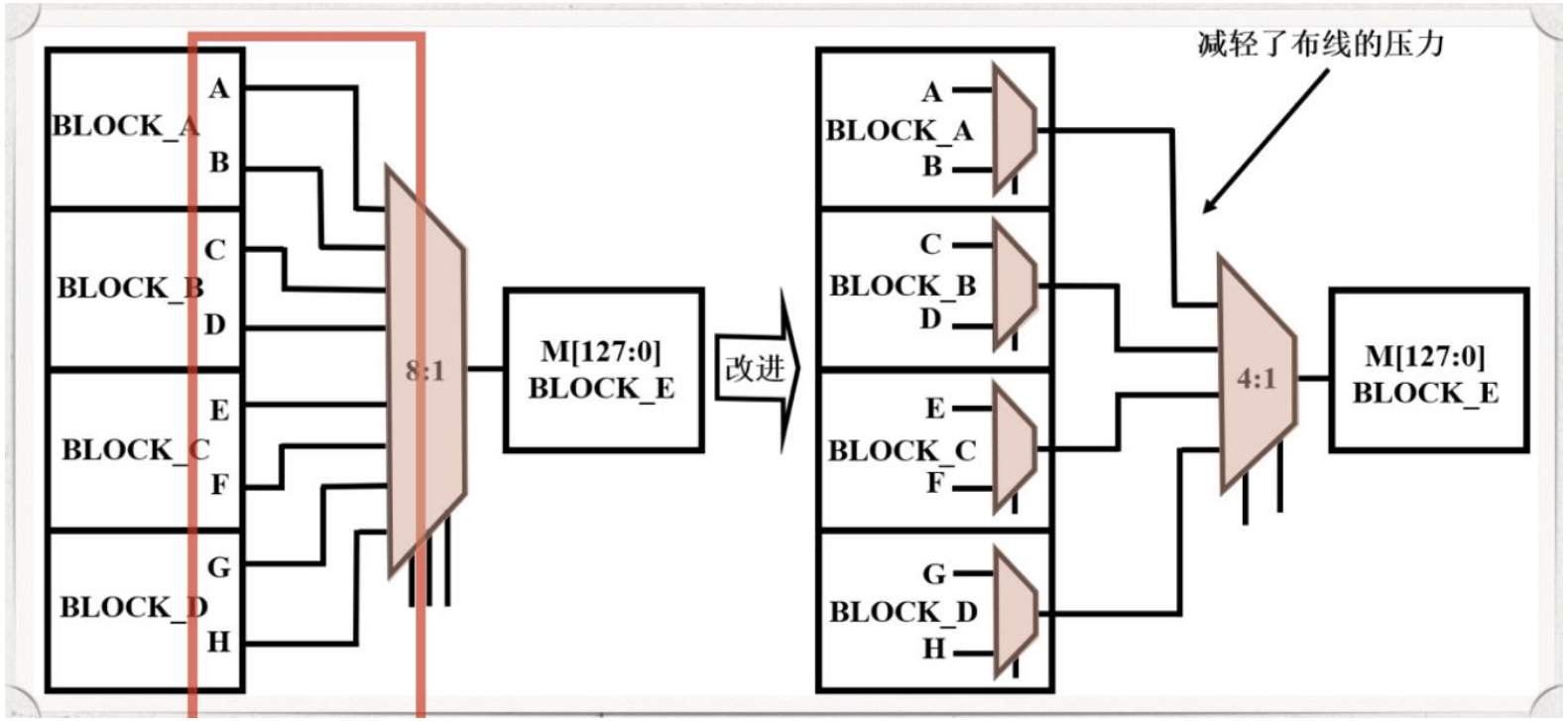
```
always @ (A or B or C or D or E or F or G or H or SEL)
begin
    case(SEL)
        3'b000:M = A;
        3'b001:M = B;
        3'b010:M = C;
        3'b011:M = D;
        3'b100:M = E;
        3'b101:M = F;
        3'b110:M = G;
        3'b111:M = H;
end
```



高质量硬件描述方法

③ 减少大的扇入/扇出

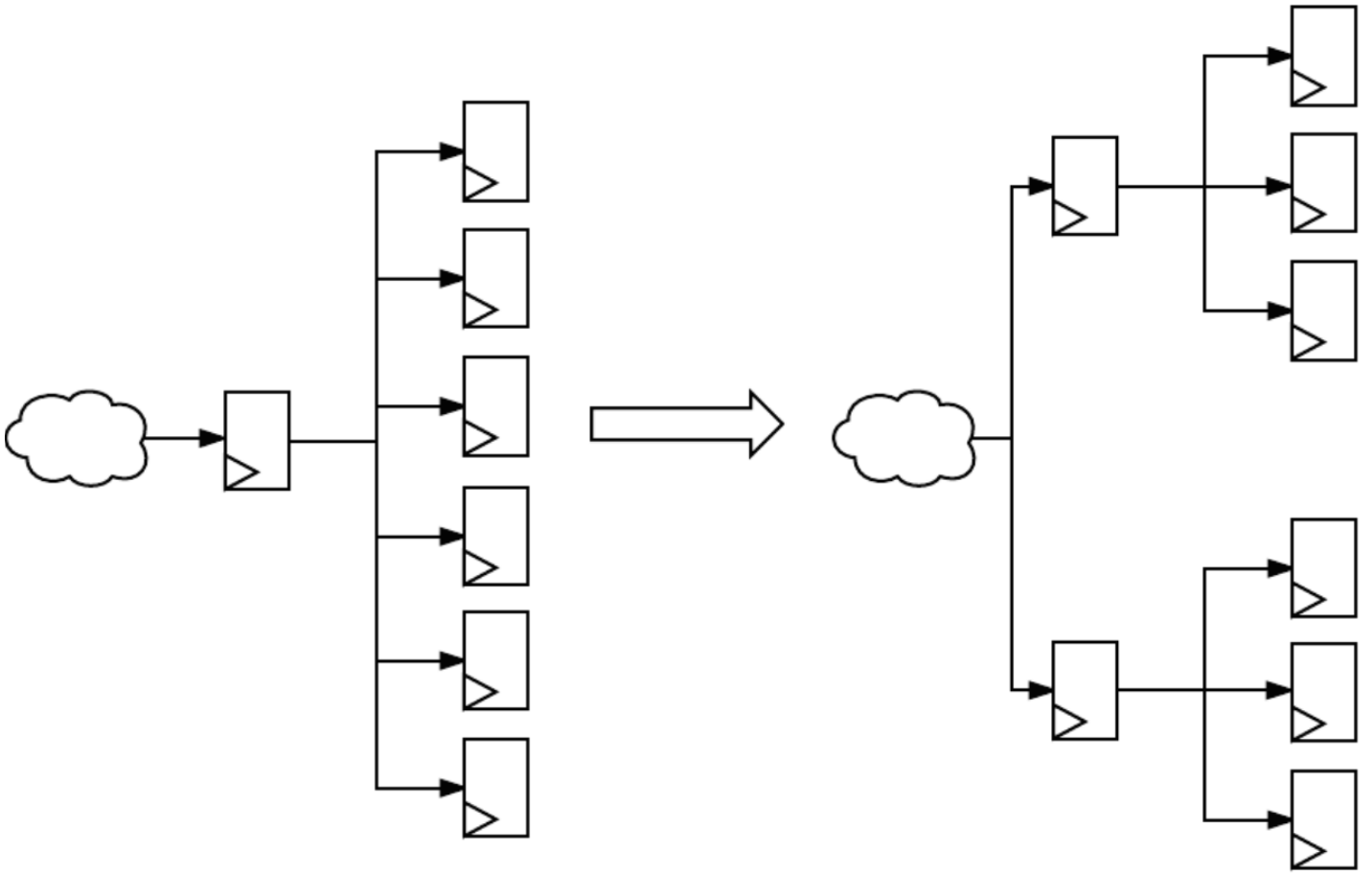
- 在综合的时候，扇入/扇出的延迟是看不出来的，只有到了布线阶段才能看到它的负面影响
- 减少扇入/扇出的方法：1. 将一个大的模块分解为多级较小的模块



高质量硬件描述方法

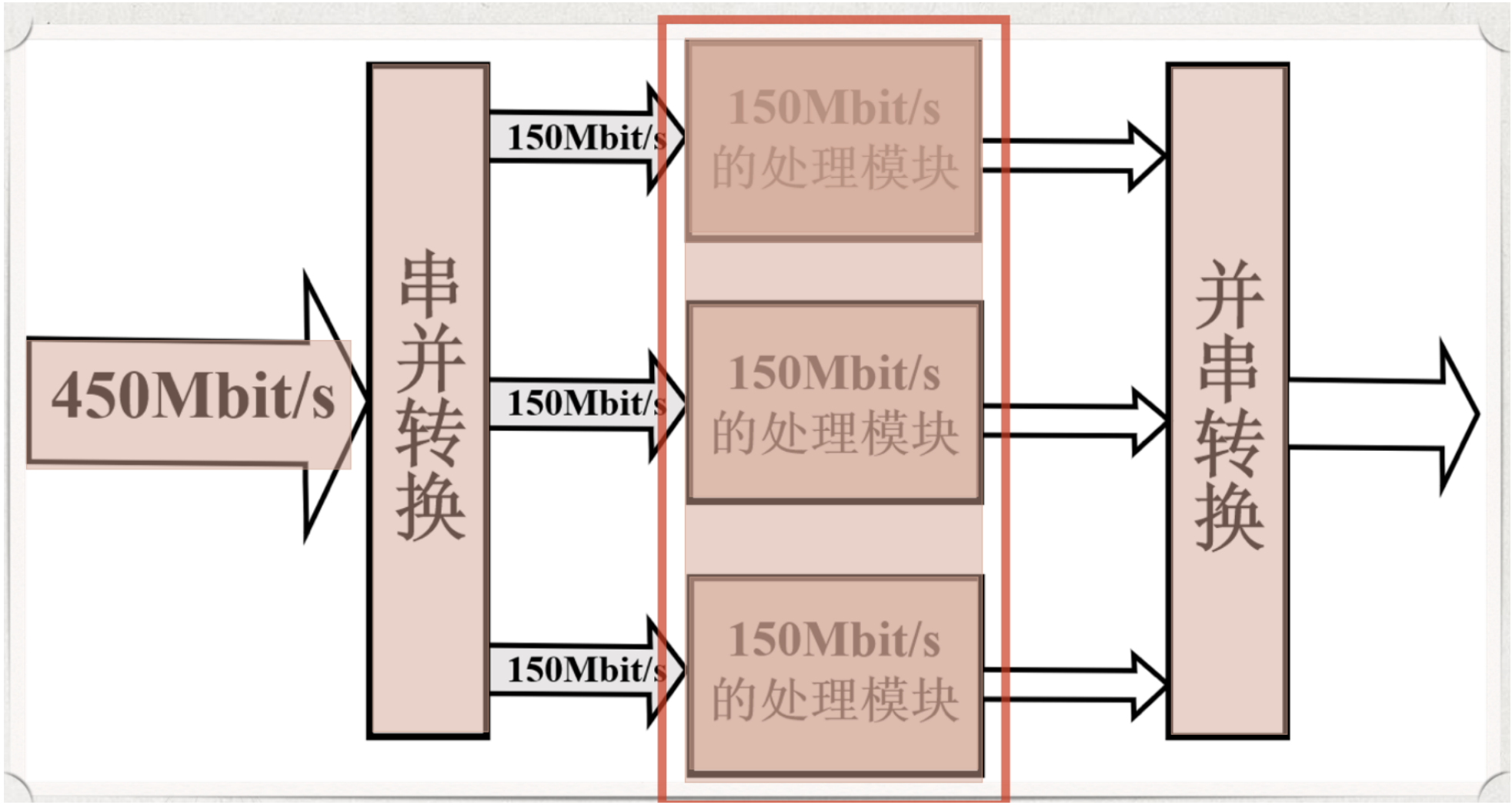
③ 减少大的扇入/扇出

- 减少扇入/扇出的方法：2. 逻辑复制 & 负载均分



高质量硬件描述方法

④ 面积换速度：串/并转换



高质量硬件描述方法

④ 面积换速度：乒乓缓存

- 普通流程（串行）：数据读取至缓冲模块 → 数据流运算
- 乒乓缓存（读取&运算并行）：缓冲模块1/2进行读取
  缓冲模块2/1 → 数据流运算