要点： 1、触发条件 同为电平或同为边沿;

2、一个else if 就是一个二选一；

3、对不同变量的赋值在不同always里进行，或者保证同一个always里的每一个变量在不同状态下都有赋值。

# 1. 二选一选择器MUX21

Module mux21 ( out, a, b, sel );//端口声明

//方向声明

input a, b, sel;

output out;

//端口数据类型声明

wire a, b, sel;

reg out; wire out;

//内部信号数据类型声明

//功能定义

always @ (a or b or sel) assign out = sel ? a:b;

begin

if (sel)

out = a;

else

out = b;

end

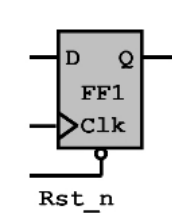
endmodule

# 2. 带异步复位的DFF

## 2.1 一位DFF

要点： 1、非阻塞赋值

2、触发条件 同为电平或同为边沿

module DFF ( clk, rst\_n, D, Q );

input clk,rst\_n;

input D;

output Q;

wire clk,rst\_n;

wire D;

reg Q;

always @ ( posedge clk or negedege rst\_n )

begin

if ( ~rst\_n )

Q <= 1’b0;

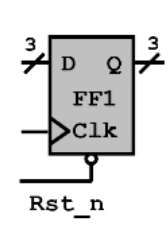
else

Q <= D;

end

endmodule

## 2.2 三位DFF

module DFF ( clk, rst\_n, D, Q );

input clk,rst\_n;

input [2:0] D;

output [2:0] Q;

wire clk,rst\_n;

wire [2:0] D;

reg [2:0] Q;

always @ ( posedge clk or negedege rst\_n )

begin

if (~rst\_n ) //低电平复位

Q <= 3’b0; //可以看出是3个DFF

else

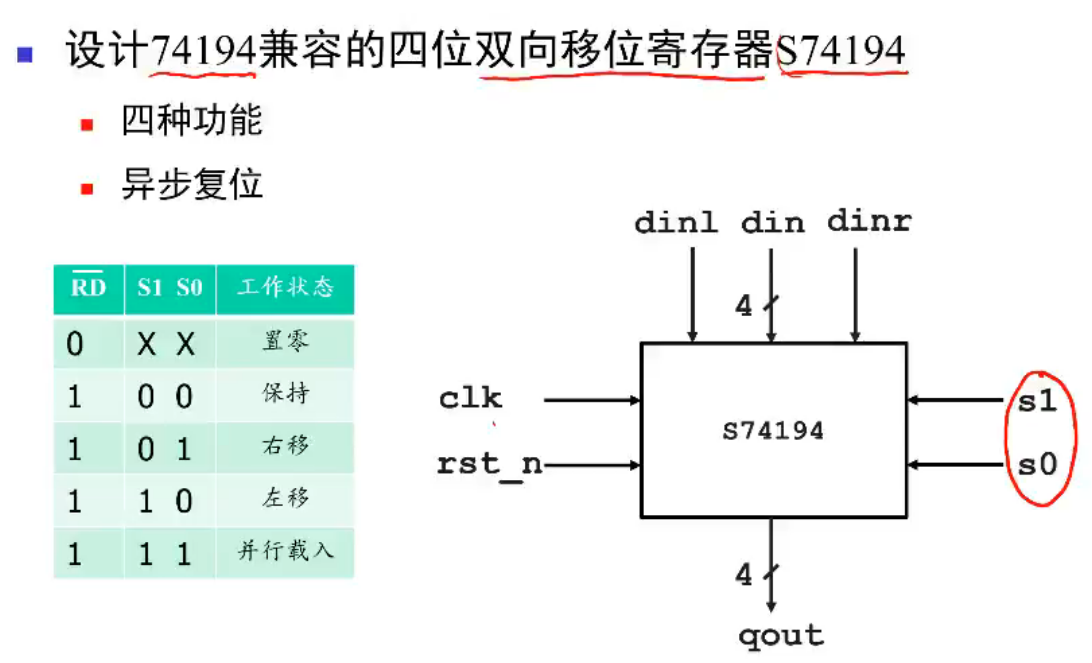
Q <= D;

end

endmodule

# 3.移位寄存器

## 3.1 四位双向移位寄存器



module shift4 ( rst\_n, clk, dinl, dinr, din, Q)

input rst\_n, clk, dinl, dinr;

input [3:0] din ;

ouput [3:0] Q;

wire rst\_n, clk, dinl, dinr;

wire [3:0] din ;

reg [3:0] Q;

always @ (posedge clk or negedge rst\_n)

begin

if(~rst\_n)

Q <= 4’b0000;

else

begin

case ( {s1,s0} )

2’b00: Q <= Q;

2’b01: Q <= { dinr, Q[3:1] };

2’b10: Q <= { Q[2:0], dinl };

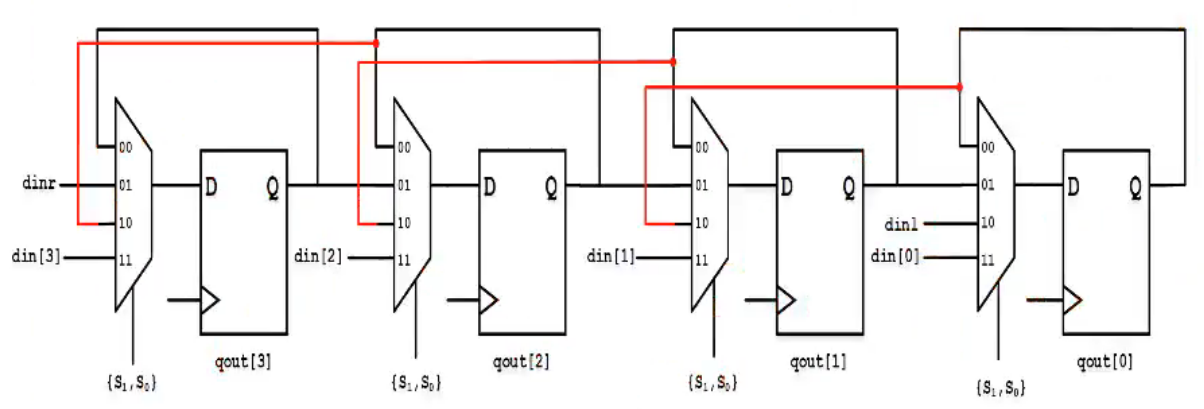
2’b11: Q <= din;

endcase

end

end

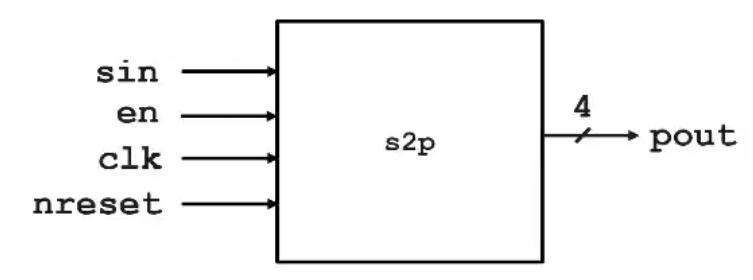
endmodule



## 3.2 四位串并转换器s2p

当en有效（高电平）时：转换进行

当en从有效切换回无效时：转换暂停



module shift (nrcset, clk, en, sin, out)

input nrcset, clk, en, sin;

output [3:0] out;

reg [3:0] out; //并行输出

reg [1:0] count; //移位计数，控制并行数据更新

reg [3:0] data; //用于移位

//移位

always @ ( posedge clk or negedge nreset )

begin

if( ~nreset )

data <= 4’b0000; //4个DFF

else if( en )

data <= { data[2:0], sin }

else

data <= data; //注意最好把状态机补完

end

//移位计数，移位4bits就并行输出一次

always @ ( posedge clk or negedge nreset )

begin

if( ~nreset ) //2个DFF

count <= 2’b00;

else if( en )

count <= count + 2’b01;

end

//并行输出

always @ ( posedge clk or negedge nreset )

begin

if( ~nreset ) //4个DFF

out <= 4’b0000;

else if( en && count == 2’b11 )

out <= data;

end

endmodule

# 4. 带使能端的寄存器设计（触发器+MUX21）

module （ Din, en, clk, rst\_n, Q ）

input Din, en, clk;

output Q;

reg Q;

always @ (posedge clk or negedge rst\_n)

begin

if( ~rst\_n )

Q <= 1’b0;

else if (en) //一个二选一

Q <= Din;

else

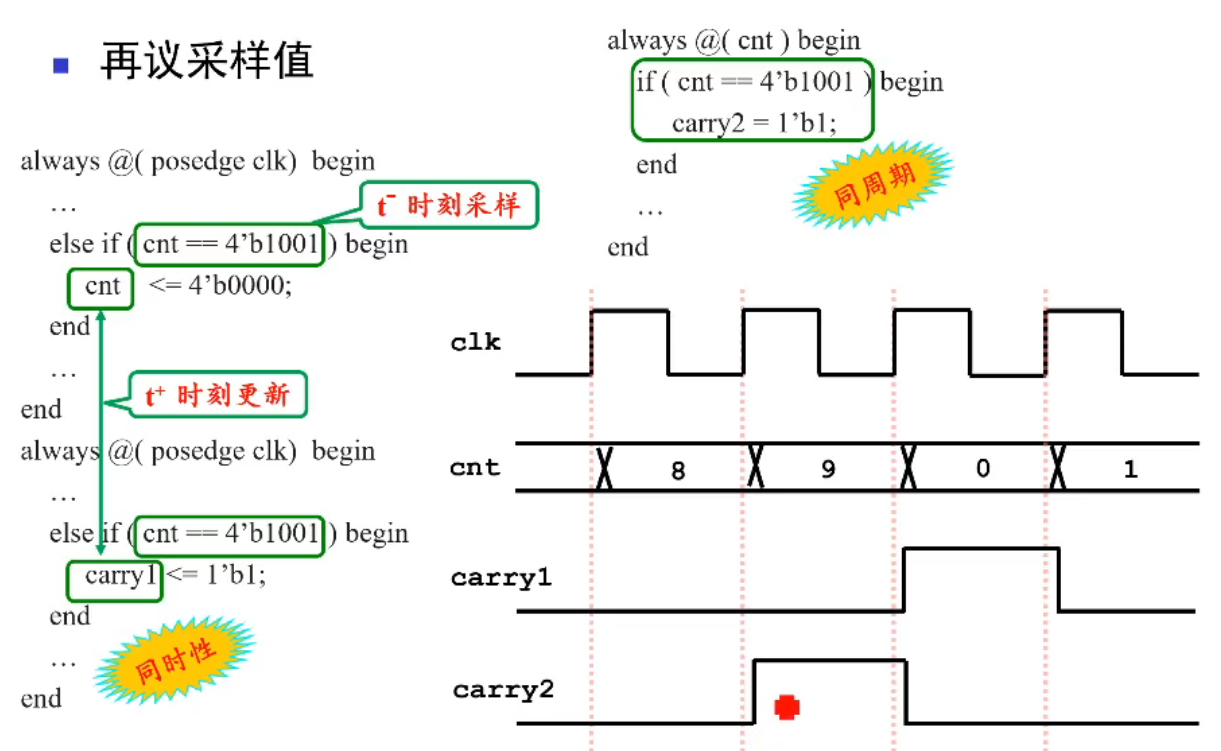
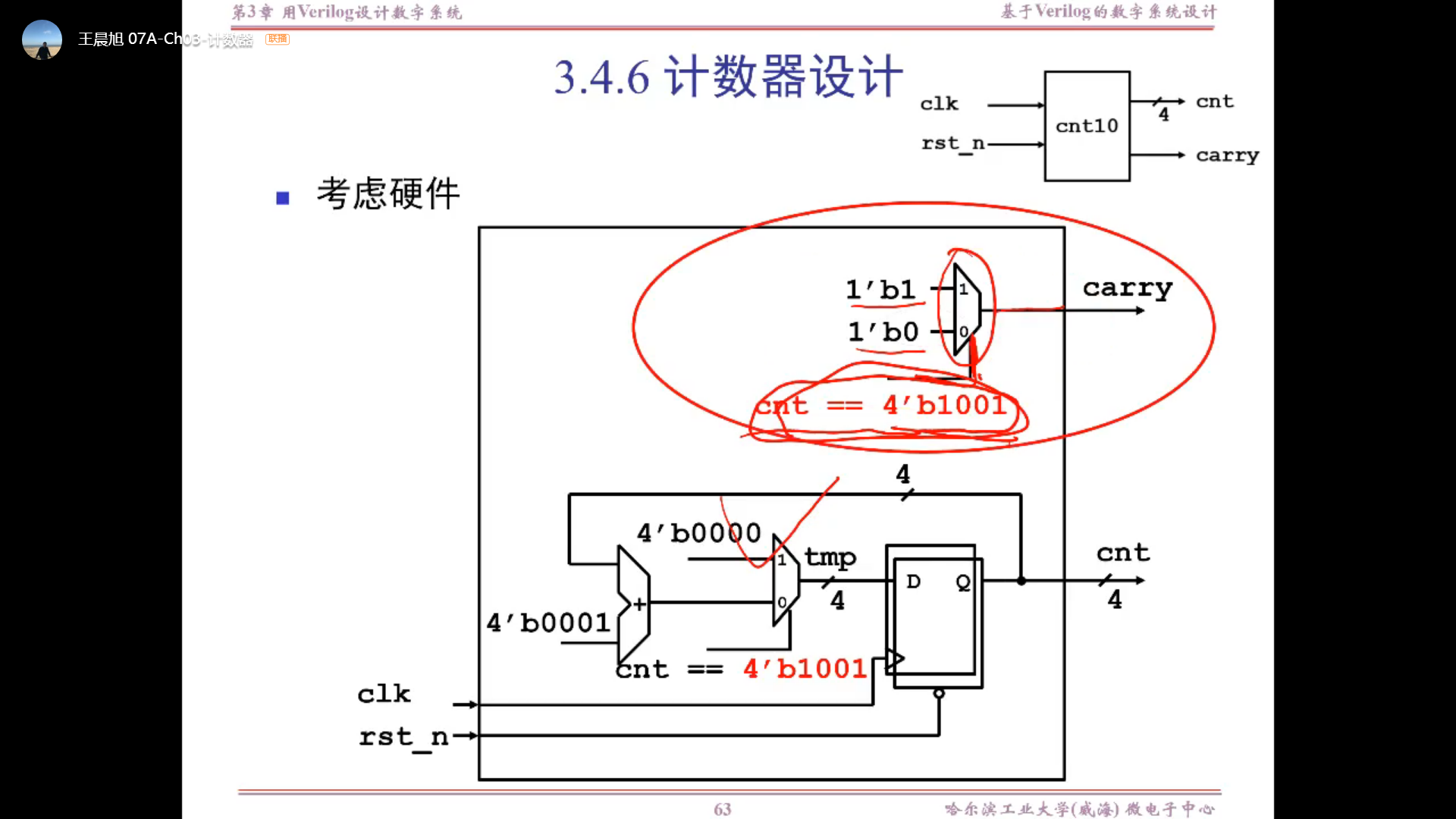
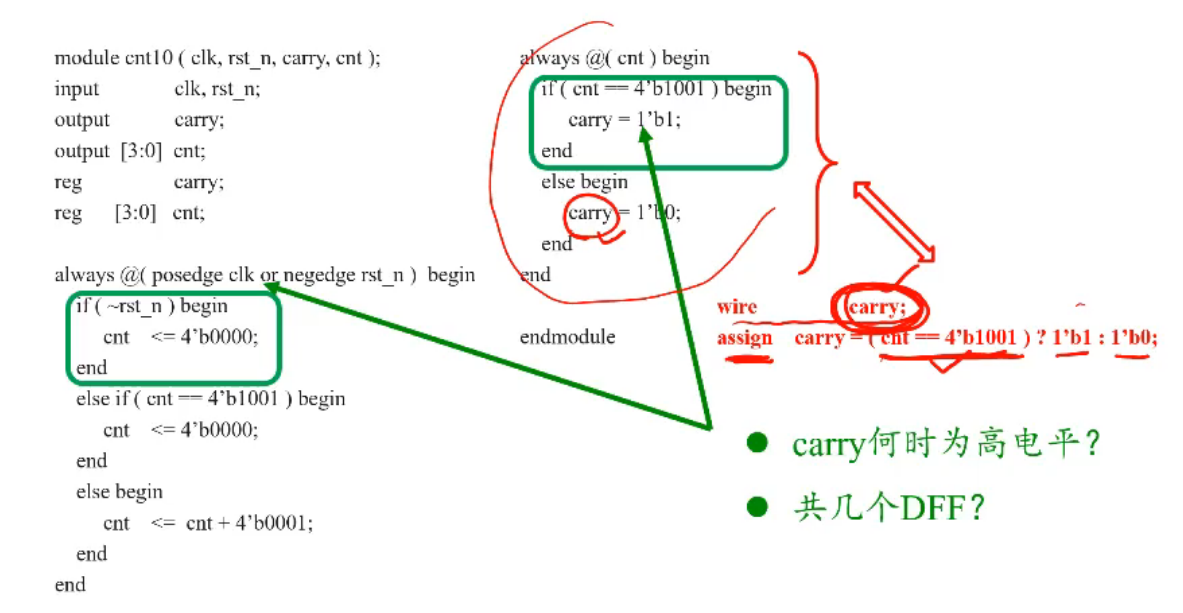
Q <= Q;

end

endmodule

# 5. 带进位的10进制计数器（进位与9同周期输出）

用cnt来做carry的触发条件是因为 触发器存在节拍延迟 而cntp判断的组合逻辑延迟较小，可以忽略

# 6. 分频器

例：将50MHz的时钟分频为1Hz，核心是计数器

使用分频器时，不建议直接使用分频后的1Hz时钟，可以用Fre\_50MHz作为触发条件，但是在always中添加Fre\_1Hz\_en作为判断条件，同时延后一个Fre\_50MHz进行下一步操作。

例：

reg [25:0] count;

always @ (posedge Fre\_50MHz )

begin

if( clr )

Fre\_1Hz\_en <= 1’b0;

else if( count == count\_width )

Fre\_1Hz\_en <= 1’b1;

else

Fre\_1Hz\_en <= 1’b0;

end

always @ (posedge Fre\_50MHz)

begin

if( clr )

…;

else if ( Fre\_1Hz\_en )

…;//要在1Hz下执行的操作

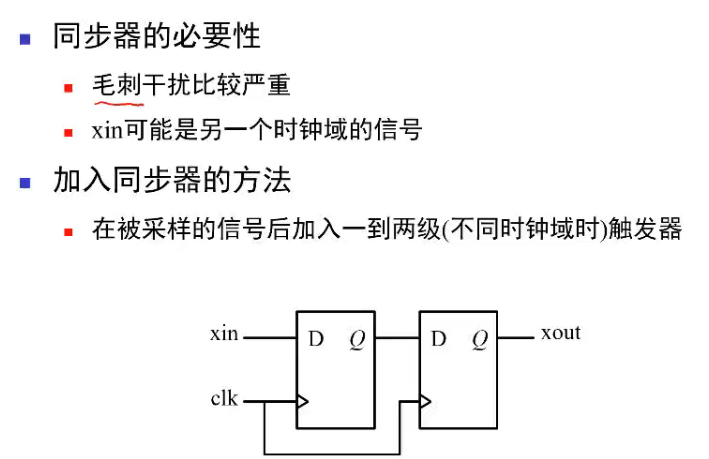
else

…;

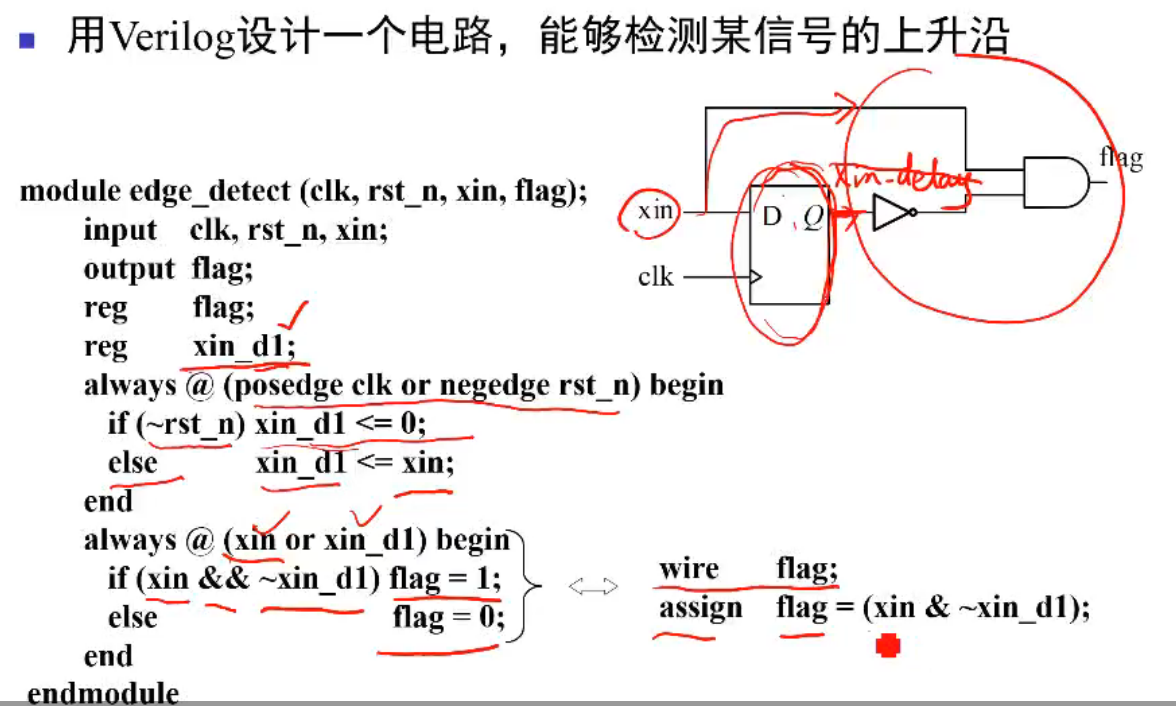
End

# 7.同步器设计

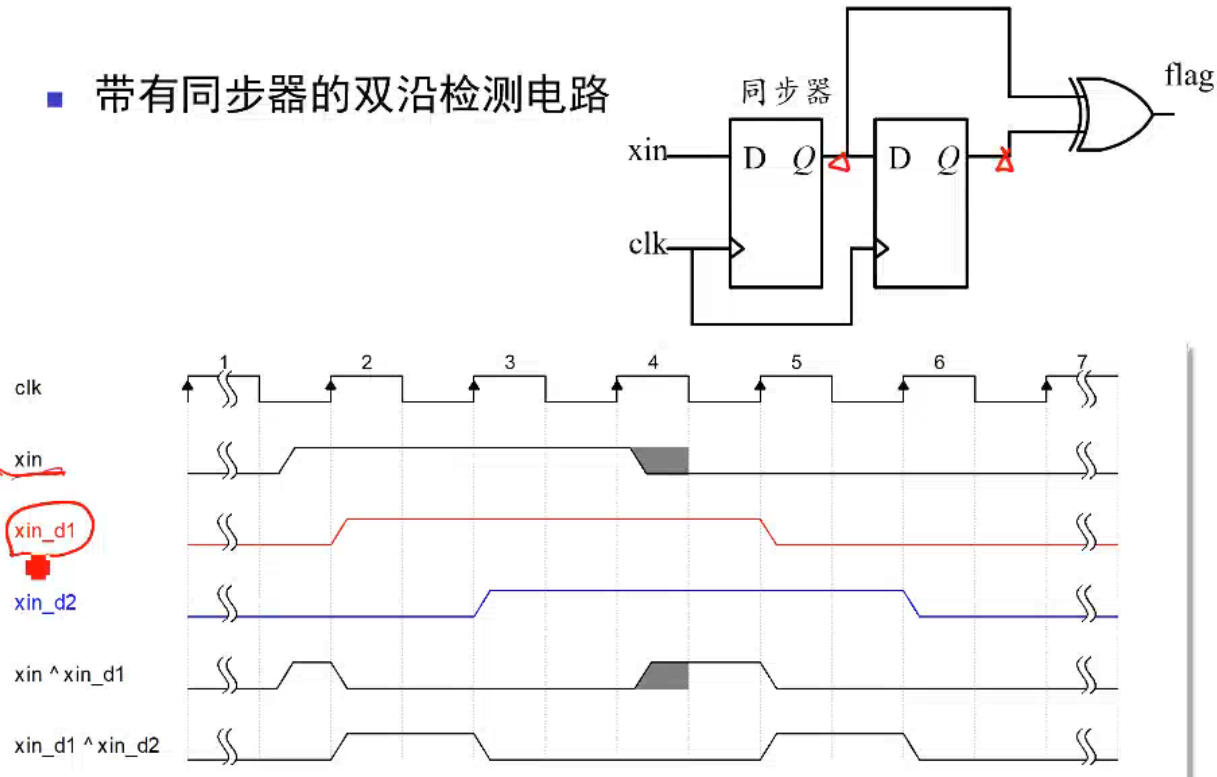
## 7.1 简单的同步器



## 7.2 带同步器的沿检测器



注意：不带同步器将导致flag不是一个完整的周期



# 8. 存储器

## 8.1 定义

reg [MSB:LSB]<memory\_name>[first\_addr:last\_addr];

例：

reg [7:0] mem [0:1023];//1K×8位的存储器

## 8.2 存取

利用暂存器 mem[1001][5]不合法

reg [7:0] mem [0:1023];

reg [7:0] mem\_word;

…

initial

begin

//取

mem\_word = mem[5];

$display( mem\_word[6] );

//存

mem\_word[6] = 1’b1;

mem[5] = mem\_word;

end

## 8.3 初始化

调用系统任务 $readmemb (二进制数据)/$readmemh（十六进制数据）

例：

$readmemb(“mem\_file”,mem);

## 8.4 ROM建模（只读）

从文件初始化，取地址读数据。

## 8.5 RAM建模

当RAM是设计中的一部分时，RAM建模必须可综合

分类：

按是否受时钟控制，分为①同步RAM，②异步RAM

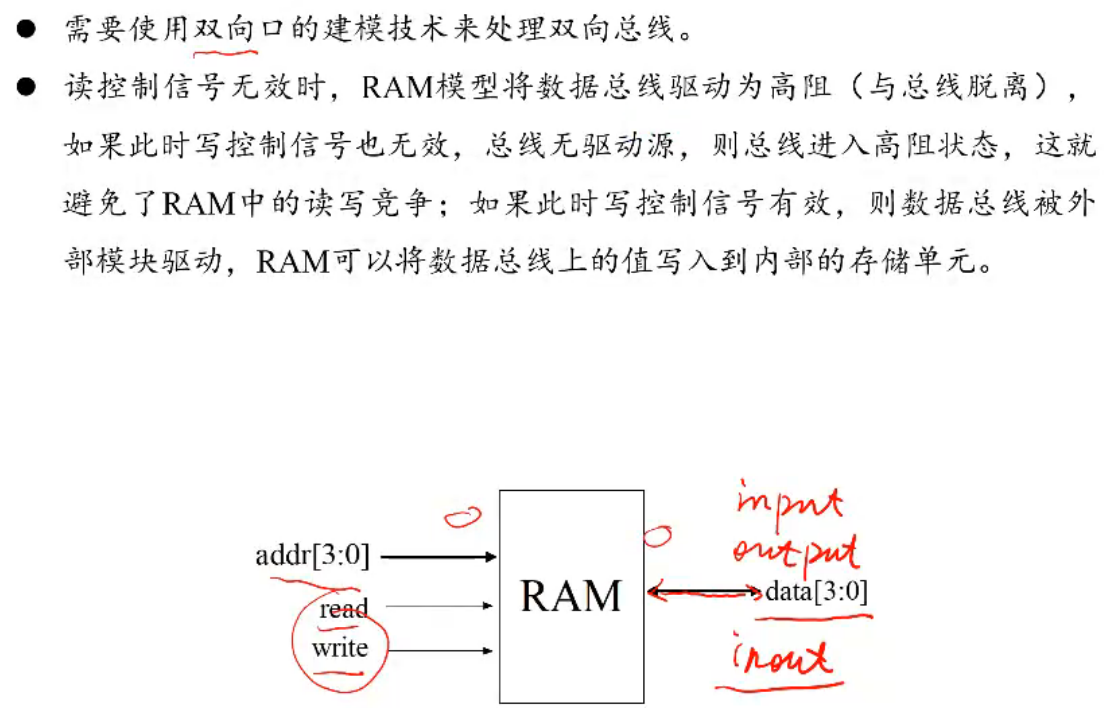
按RAM地址线套数，分为①单端口RAM②双端口RAM

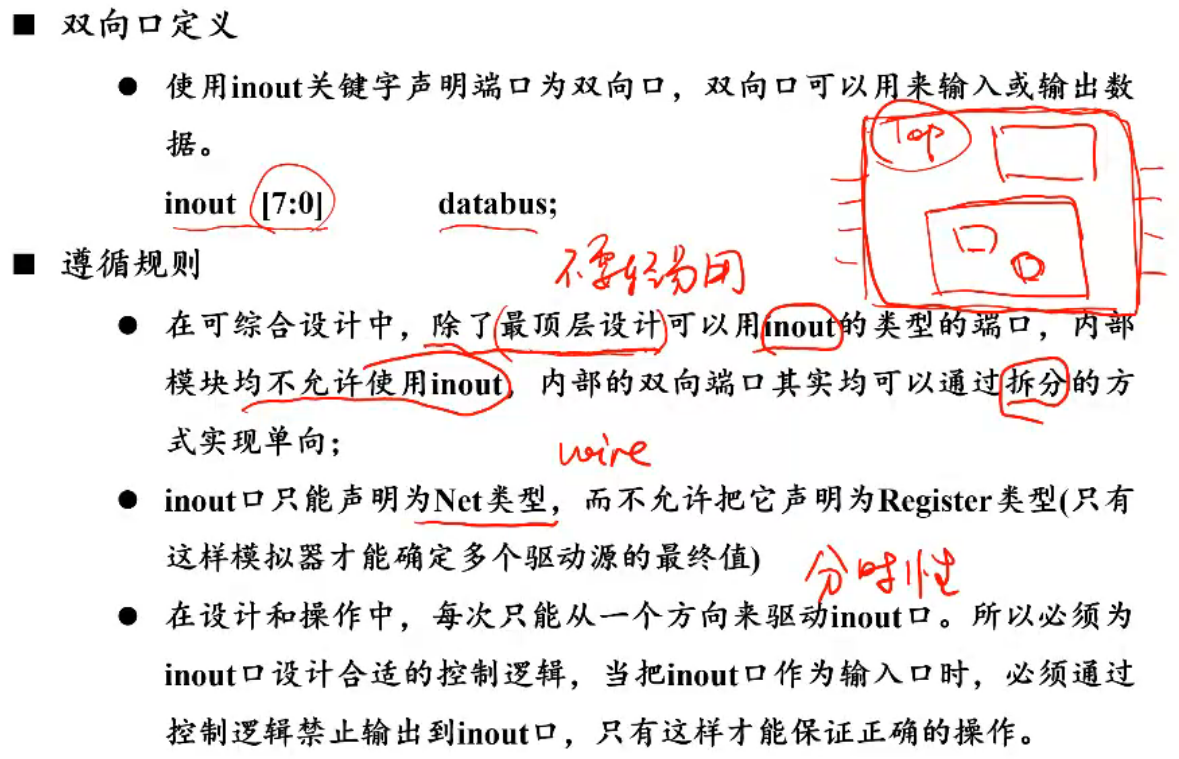
按读写时数据总线是否分开，分为

① 读写数据总线统一，此时数据端口为inout类型；

② 读写数据总线分离，数据端口分为datain（写）和dataout（读）。

时分复用的RAM建模





module myram ( data, addr, read, write )

inout [3:0] data;

input [3:0] addr;

input read, write;

reg [3:0] memory [0:15] //16×4bits

//从ram读数据到总线上

assign data = read ? memory[addr] : 4’bz;

//从总线上写数据到ram中

always @ ( posedge write )

memory[addr] = data;

endmodule

## 8.6 同步RAM的建模

分为：①同步写异步读 ②同步写同步读

常用读写总线分离的RAM

//同步写，异步读

module myram ( data, addr, clk, read, write )

inout [3:0] data;

input [3:0] addr;

input clk, read, write;

reg [3:0] memory [0:15] //16×4bits

//从ram读数据到总线上 //异步读

assign data = read ? memory[addr] : 4’b0000;

//从总线上写数据到ram中//同步写

always @ ( posedge clk )

begin

if( write )

memory[addr] = data;

end

endmodule

//同步写，同步读

module myram ( data, addr, clk, read, write )

inout [3:0] data;

input [3:0] addr;

input clk, read, write;

reg [3:0] memory [0:15] //16×4bits

//从ram读数据到总线上 //同步读

always @ ( posedge clk )

begin

if( read )

data = memory[addr];

end

//从总线上写数据到ram中//同步写

always @ ( posedge clk )

begin

if( write )

memory[addr] = data;

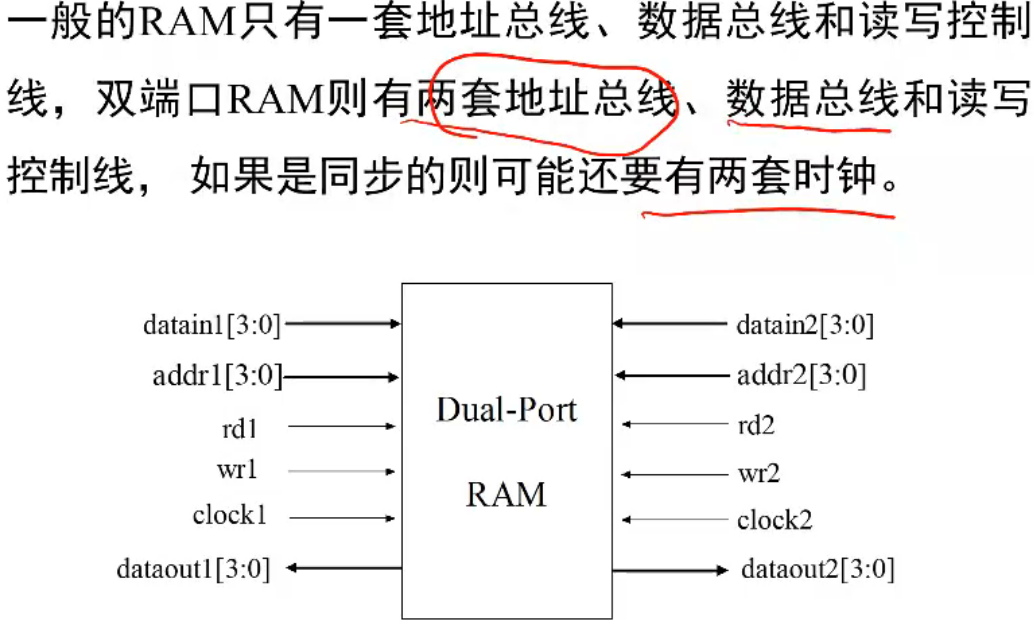
end

endmodule

## 8.7 双端口RAM建模

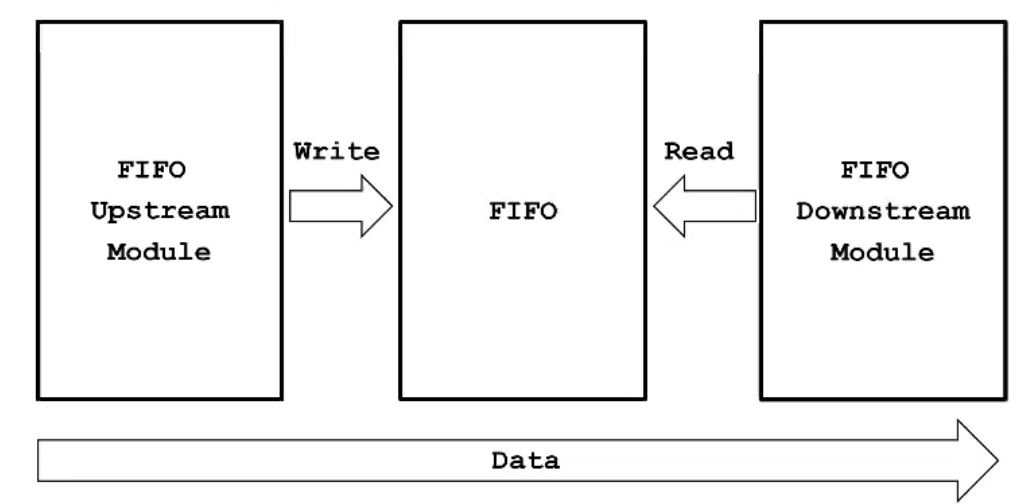
双端口有两套地址总线、数据总线和读写控制线(TURE),同步的可能需要两套时钟。

注意：多端口 主要看地址线



# 9. 同步FIFO设计

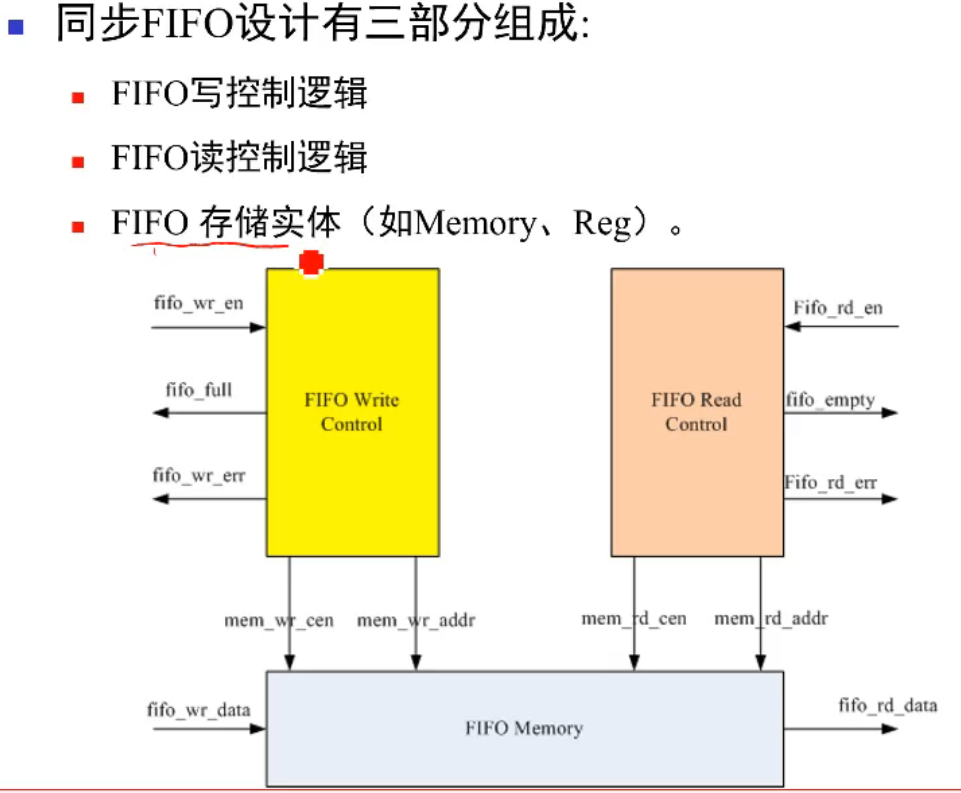
FIFO是一种先进先出的数据交互方式（计算机组成概念中的“队”）



FIFO按工作时钟域不同分为：①同步FIFO ②异步FIFO

1. 同步FIFO的 写时钟和读时钟为 同一时钟，FIFO内部逻辑均为同步逻辑，常用于 交互数据缓冲
2. 异步FIFO的 写时钟和读时钟为 异步时钟，FIFO内部写逻辑和读逻辑的交互需要异步处理，场用于 跨时钟域交互

## 9.1 同步FIFO设计



fifo\_data\_cnt: 一个fifo\_wr\_en加一，一个fifo\_rd\_en减一；

if( fifo\_data\_cnt ) == depth

fifo\_full = 1’b1;

if( fifo\_data\_cnt ) == d0

fifo\_empty = 1’b0;