FPGA开发基础公益培训（第2讲）

掌握Verilog的设计利器

个人新浪微博：lucky_mao

主要内容

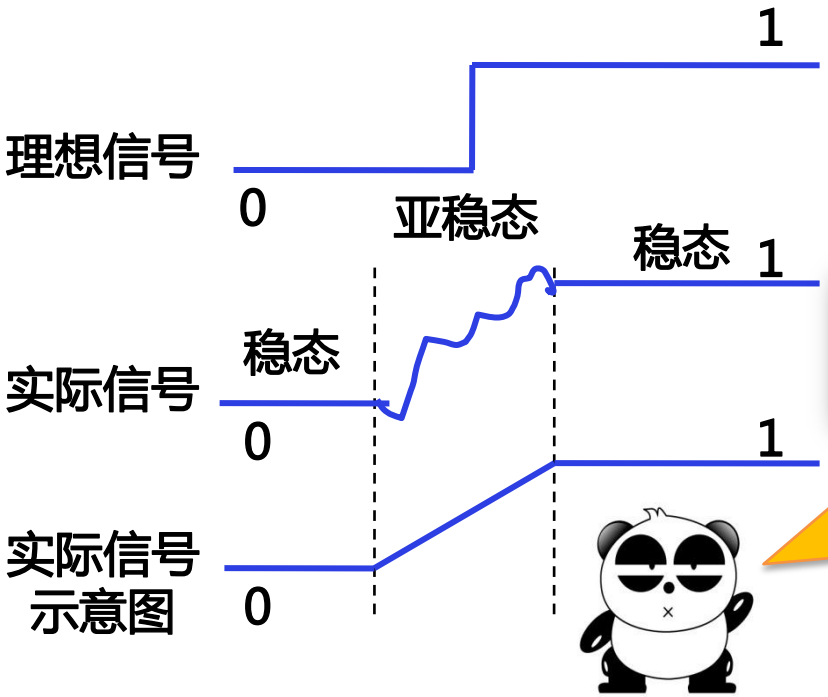
- Verilog基本语法与设计方法
- Verilog系统设计原则与技巧
- Verilog进行典型电路的设计

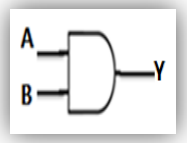
进一步认识数字电路（1）

数字电路：完成对电信号的**组合逻辑**和**时序逻辑**的**变换**。

- 可靠性
- 速度

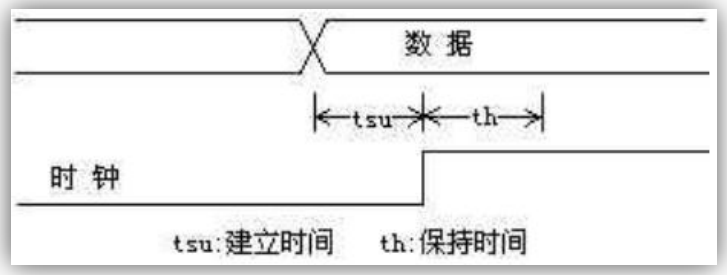
【单bit信号的变换过程】

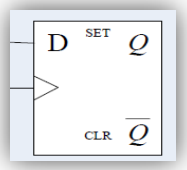




组合逻辑电路：

信号的变换是“立即”发生。





时序逻辑电路：

在沿变时才会发生变换。

所有的数字信号其实都是模拟信号！信号电平的翻转会经历稳态->亚稳态->稳态。信号经触发器输出需满足建立保持时间！

进一步认识数字电路（2）

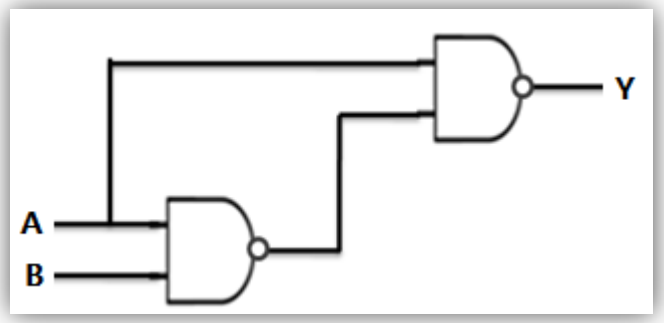
延时=门延时+线延时

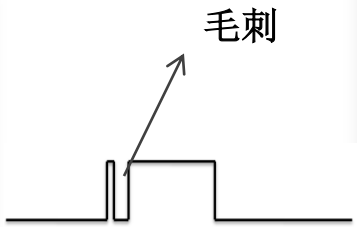
信号的变换在电路中是需要传输和建立时间的，要想获得稳定的输出信号，必须等待一定的**时间**。
控制延时的目的是为了获得更高的电路工作频率！

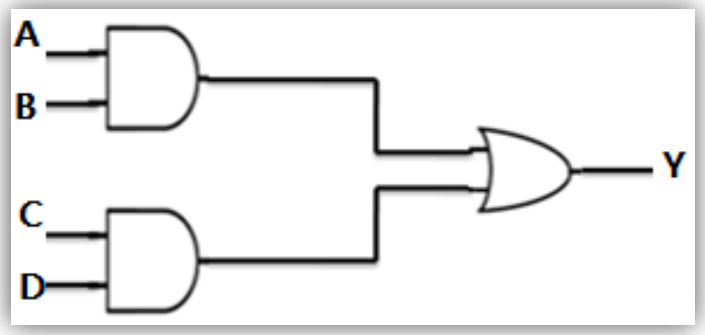
门延时：往往认为器件的门延时是固定的（实际上会受温度、电压等因素的影响）

线延时：布局布线不同，延时大小会不一样（通过设计约束和逻辑实现延时控制）

组合逻辑电路的冒险与竞争





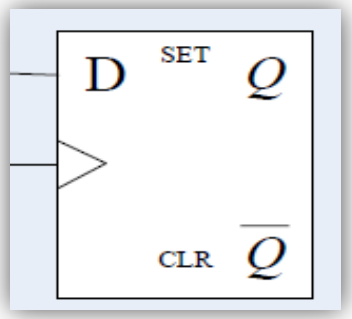


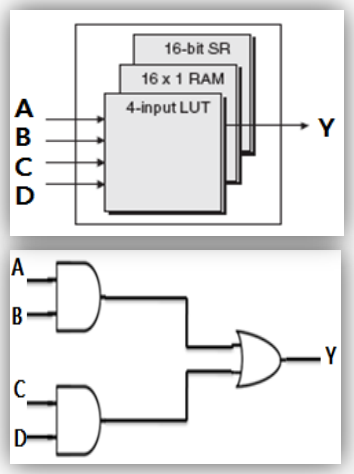
竞争：同一信号沿不同路径到达

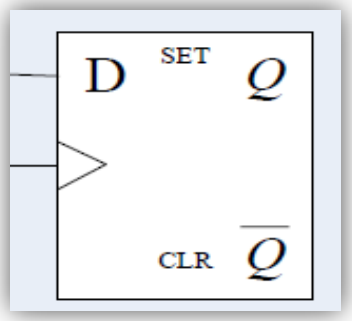
冒险：不同信号到达的时间不同

进一步认识数字电路（3）

输入信号



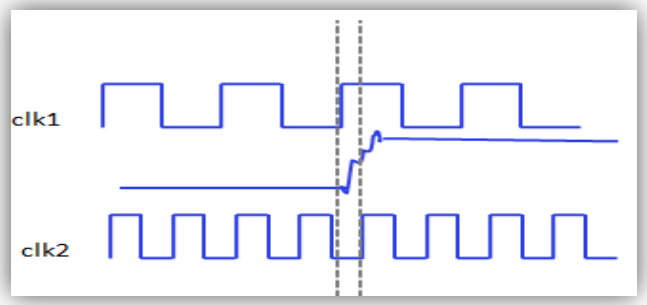




输出信号

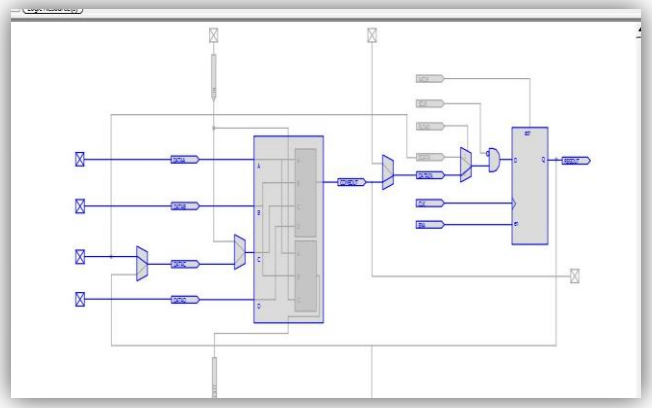
组合逻辑的大小会影响信号的稳定时间。

同步还是异步？——采用同步设计

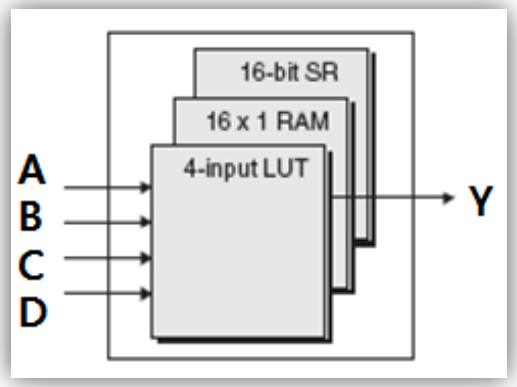


触发器能够滤除小的毛刺，可以防止其在电路中向后传播。如果存在一些**大的毛刺**或者存在**较多毛刺时**，就需要通过修改设计进行避免。

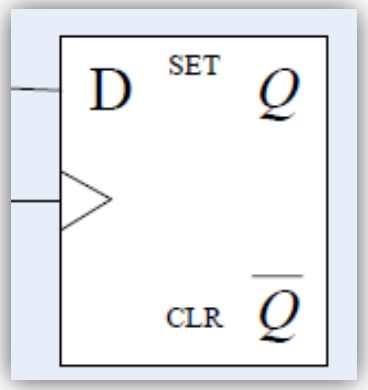
FPGA中实现组合逻辑是时序逻辑



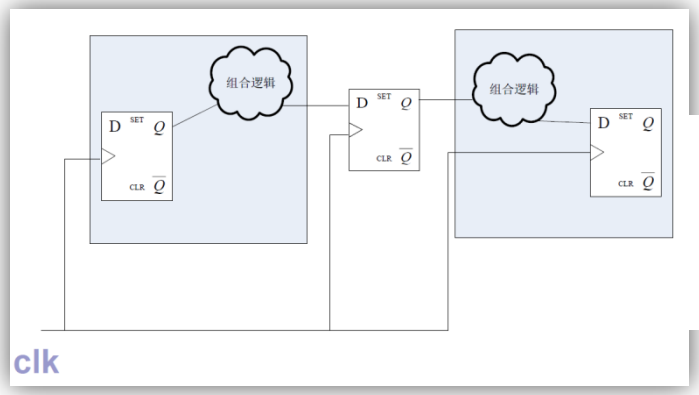
基本的逻辑单元



LUT实现组合逻辑



寄存器实现时序逻辑



当组合逻辑较大时，其输出信号到达稳定的时间会较长。通过在逻辑中插入一级寄存器，虽然增加了完成该过程所需的时钟数，却可以提高系统整体的工作频率——这就是流水设计的思想。

FPGA的ESL设计语言

ESL的建模层次

1. 系统级建模：C/C++/M语言，设计思想	——说明文
2. 算法级建模：C/C++/M语言，设计思想	——段落
3. 行为级建模：HDL，抽象功能描述	——语句
4. 门级建模：HDL，电路结构的具体描述	——字符
5. 开关级建模：MOS晶体管	——笔画

Verilog能够很方便地对电路系统进行行为级和门级的建模，结构简单、易综合、时序关系清晰，而且很容易被软件设计人员理解。

ESL设计的三要素

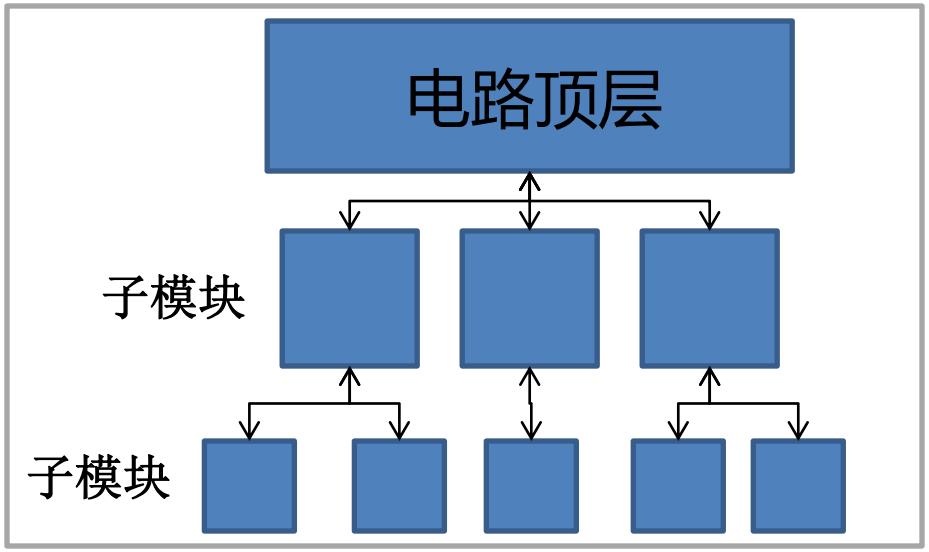
目标：能够设计出具有特定功能、稳定可靠的电路模块

需求：

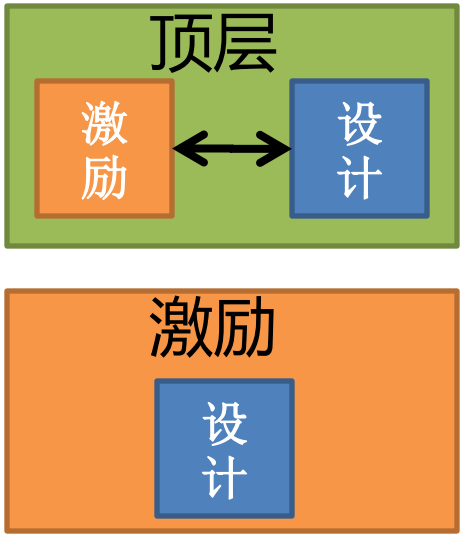
- 实现电路的目标功能（可综合描述语句：必须能综合成实际电路）
- 实现电路的仿真验证（不需要实际实现：丰富而灵活的描述方法）

方法：模块化的设计，自顶向下，自底向上

目标电路设计



目标电路验证



Verilog基本语法与设计方法

快速地进行电路模块的设计

1、定义一个顶层模块

2、设计内部电路逻辑

3、例化内部子模块

电路逻辑块：always

条件分支：if 、case

信号连续赋值：assign

运算符： = , <=, { }, &, &&

```
module <模块名>
        (<端口列表>);
        <定义>
        <内部逻辑实现>
endmodule
```

```
module_name  u0(
  .端口1信号名（连接端口1信号名),
  .端口2信号名（连接端口2信号名),
  .端口3信号名（连接端口3信号名)
);
```

```
always @ （敏感参数列表）
begin
        <内部逻辑实现>
end
```

Verilog基本语法与设计方法

快速地进行电路模块的仿真

1、定义一个顶层testbench

2、产生时钟等激励信号

3、例化设计模块

4、利用工具仿真testbench

电路逻辑块：initial、always

时间的单位和精度:`timescale

`timescale 1ns / 10ps

```
module  tb ( ) ;
reg       clk;
reg       asy_rst;
reg       sw ;
wire      led;
……
endmodule
```

```
initial
begin
asy_rst = 1’ b0;
# 10*duty   asy_rst = 1’ b1;
# 10*duty   asy_rst = 1’ b0;
end
```

```
always #50 clk=~clk
```

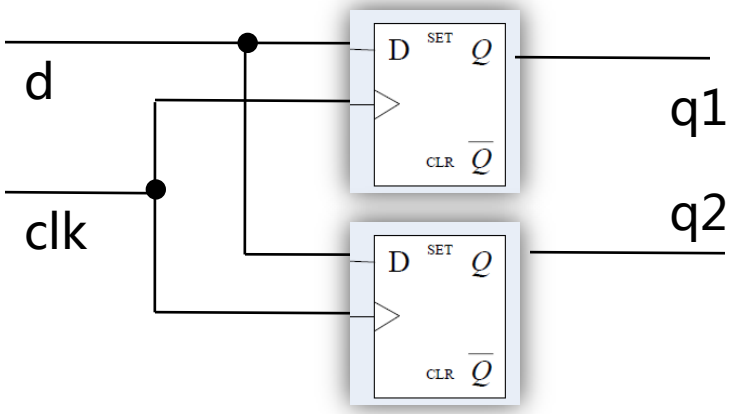
阻塞赋值与非阻塞赋值

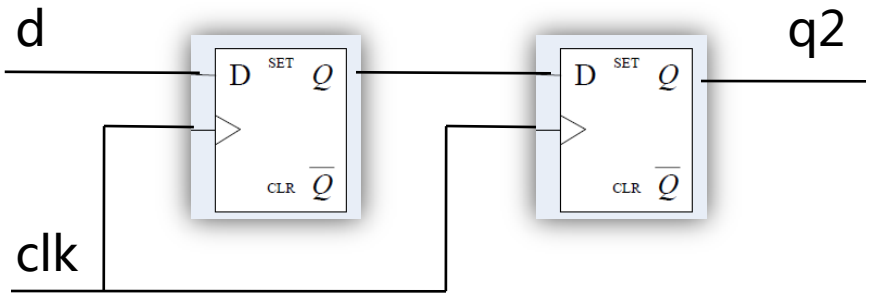
阻塞赋值 组合逻辑

```
module ex2(clk,d,q1,q2);
 input clk,d;
 output q1,q2;
 reg q1,q2;
 always @(posedge clk) begin
    q1 = d;
    q2 = q1;
  end
endmodule
```

```
module ex2(clk,d,q1,q2);
 input clk,d;
 output q1,q2;
 reg q1,q2;
 always @(posedge clk)
begin
    q1 <= d;
    q2 <= q1;
  end
endmodule
```

非阻塞赋值 时序逻辑

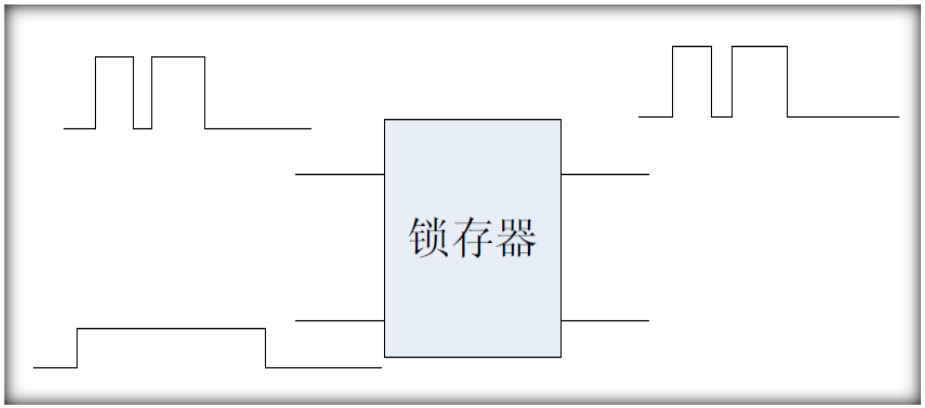




寄存器、触发器与锁存器

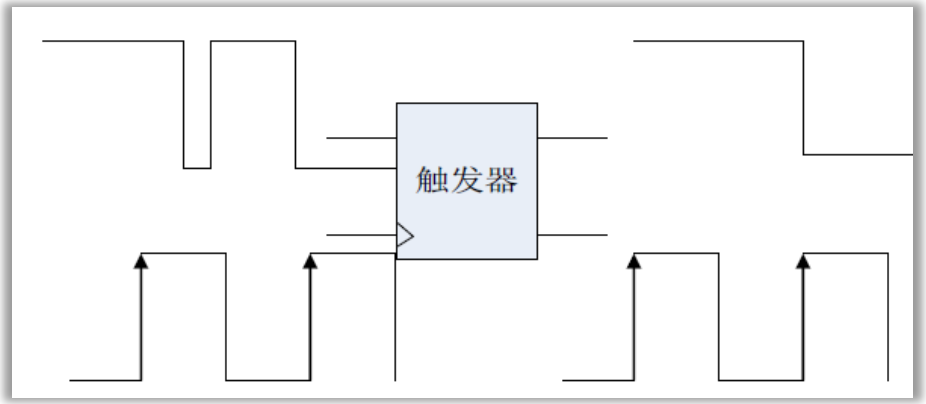
寄存器的定义： `reg data;`

寄存器的含义：带有存储功能电路结构（可以是RAM、触发器、锁存器。）



电平触发、异步工作模式

会透传出毛刺



沿变触发、同步工作模式

能滤除毛刺

电路设计实例

模块名与接口

```
module test(
          input     wire              asy_rst          , // reset signal high is active
          input     wire              clk              , // 100MHz
          input     wire      [7:0]   i_sw             , // switch signal
          output    wire      [31:0]  o_led              // LED
);
```

内部信号名与信号赋值

```
parameter LED_1MS = 28'd100_000_000; //100MHz时钟下，1ms的时钟个数

reg       [27:0]    led_cnt;            //计数器
reg       [7:0]     led_flow;           //流水灯
wire      [7:0]     led_sopc;           //子模块接口信号
reg                 led_block_a;        //阻塞赋值的灯a
reg                 led_block_b;        //阻塞赋值的灯b
reg                 led_non_block_a;    //非阻塞赋值的灯a
reg                 led_non_block_b;    //非阻塞赋值的灯b

assign o_led = {12'd0,led_block_a,led_block_b,led_non_block_a,led_non_block_b,led_sopc,led_flow};
```

电路设计实例

```
/*------------------------------------------------------------
gen the led_cnt signal
------------------------------------------------------------*/
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
        led_cnt <= 28'd0;
    else if ( ( i_sw[0]==1'b1 ) && ( led_cnt < ( LED_1MS -'d1 ) ) )
        led_cnt <= led_cnt + 28'd1;
    else
        led_cnt <= 28'd0;
end
```

计数器计时1ms电路

```
/*------------------------------------------------------------
gen the led_flow signal
------------------------------------------------------------*/
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
        led_flow <= 8'd0;
    else if ( led_cnt == 28'd1 )
        begin
            if ( led_flow == 8'd0 )
                led_flow <= 8'd1;
            else
                led_flow <= { led_flow[6:0],1'b1 };
        end
    else
        led_flow <= led_flow;
end
```

产生流水灯电路

电路设计实例

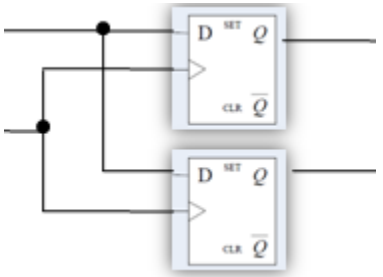
例化处理器电路子模块

```
//------------------------------------------------------------
// Instantiate the mb_soc module
//------------------------------------------------------------
mb_soc mb_soc_u
(
          .clk        ( clk_200  ),
          .rst        ( asy_rst  ),
          .o_mb_led   ( led_sopc )
          );
```

可以由处理器编写程序指令实现led灯的控制

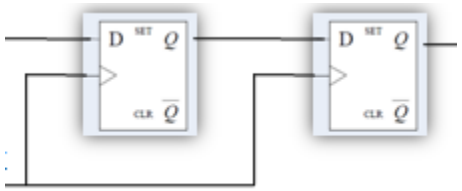
```
/*-------------------------------------------------------------
gen the led_block_a,led_block_b signals
-------------------------------------------------------------*/
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
    begin
        led_block_a = 1'b0;
        led_block_b = 1'b0;
    end
    else
    begin
        led_block_a = i_sw[1];
        led_block_b = led_block_a;
    end
end
```

阻塞赋值点亮
led_block_a
led_block_b

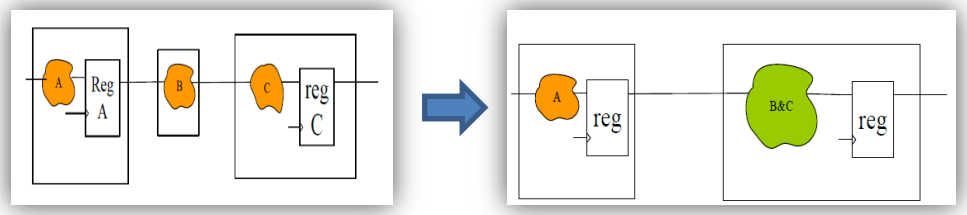



```
/*-------------------------------------------------------------
gen the led_non_block_a,led_non_block_b signals
-------------------------------------------------------------*/
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
    begin
        led_non_block_a <= 1'b0;
        led_non_block_b <= 1'b0;
    end
    else
    begin
        led_non_block_a <= i_sw[1];
        led_non_block_b <= led_non_block_a;
    end
end

endmodule
```

非阻塞赋值点亮
led_non_block_a
led_non_block_a



Verilog系统设计原则与技巧



【基本原则】

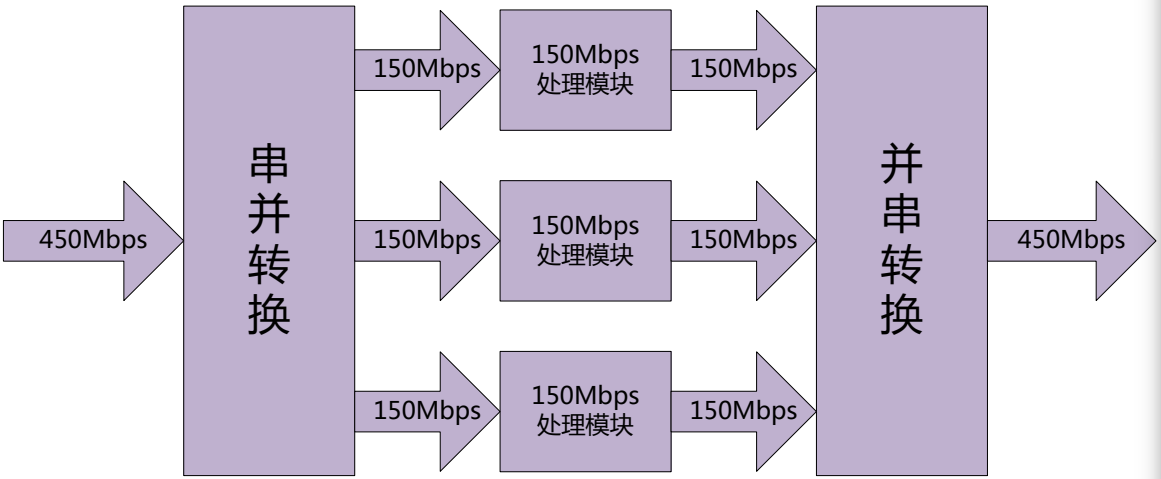
1. 模块化设计原则（模块接口清晰，功能划分合理）
2. 基于IP设计原则（尽量复用成熟的IP）
3. 硬件可实现原则（岁实现的电路做到心中有底）
4. 同步系统设计原则（系统稳定的关键）

【常用技巧】

1. 速度与面积的平衡
2. 乒乓操作
3. 流水线操作

小知识：FPGA中电路有四种状态：0、1、Z、X。Z是一个什么状态呢？

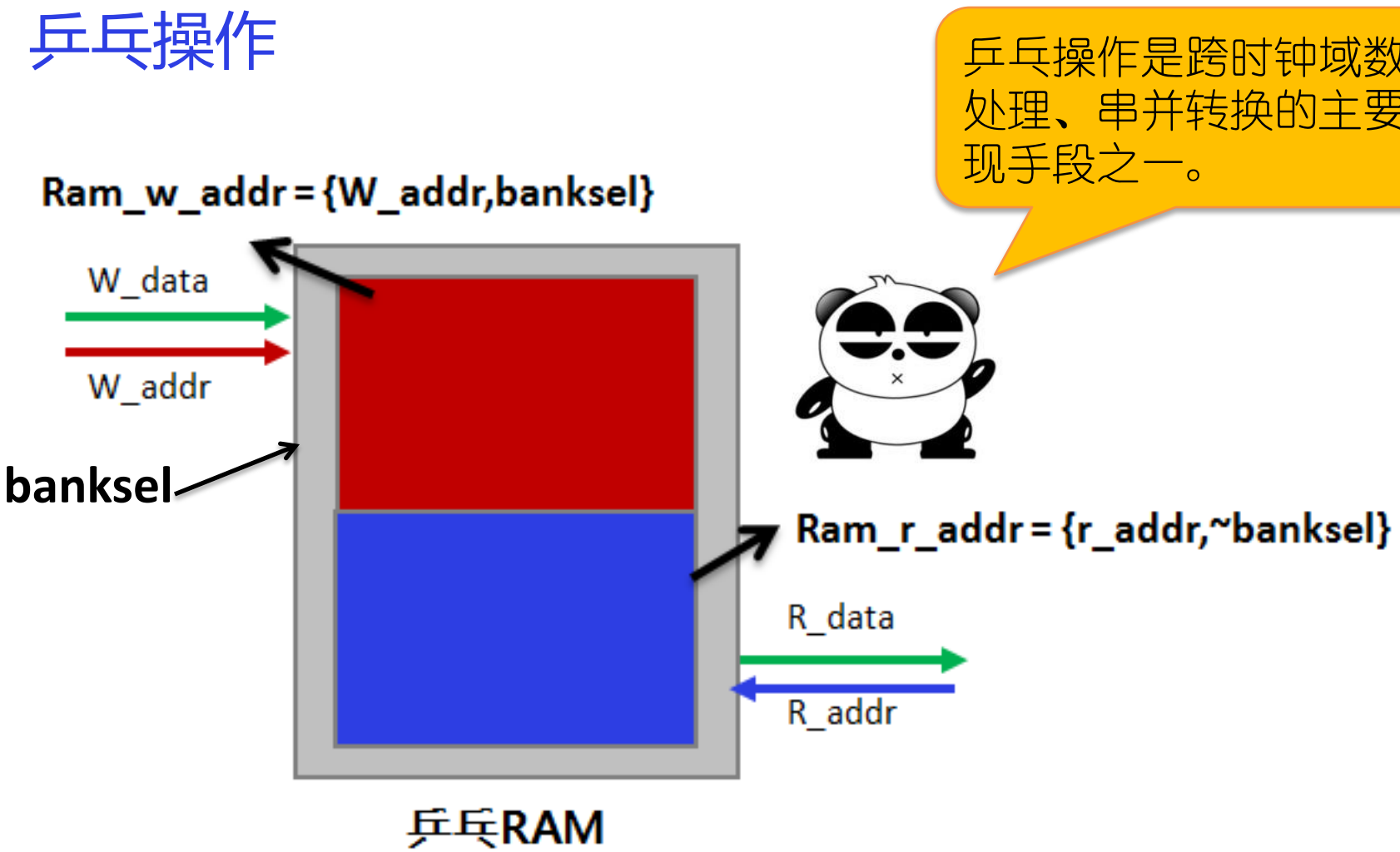
速度与面积的平衡



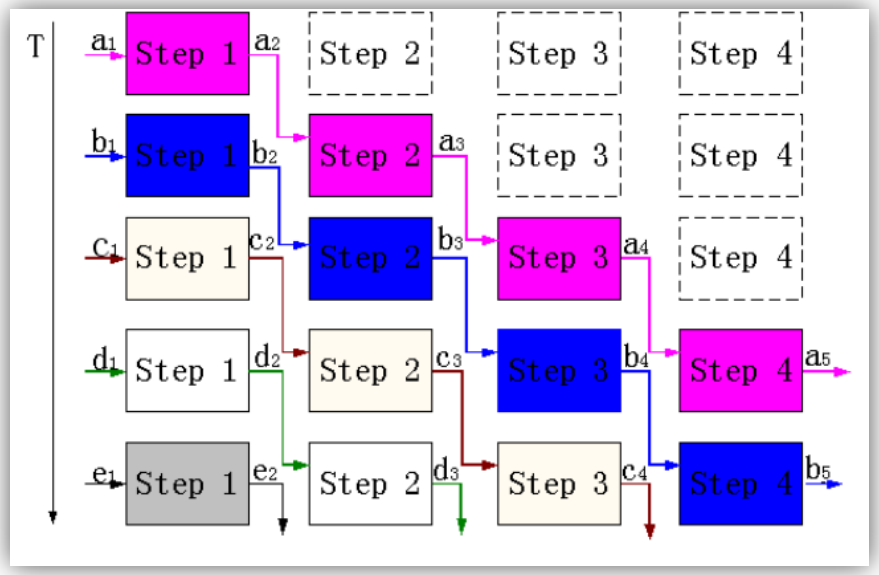

```
reg    [15:0]    data;
wire   [15:0]    result;

always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
        data <= 16'd0;
    else
        data <= {data[31:0],i_data}
end

data_process u1
(
          .clk        ( clk          ),
          .rst        ( asy_rst      ),
          .i_data     ( data[7:0]    ),
          .o_result   ( result[7:0]  )
          );

data_process u2
(
          .clk        ( clk          ),
          .rst        ( asy_rst      ),
          .i_data     ( data[15:8]   ),
          .o_result   ( result[15:8] )
          );
```

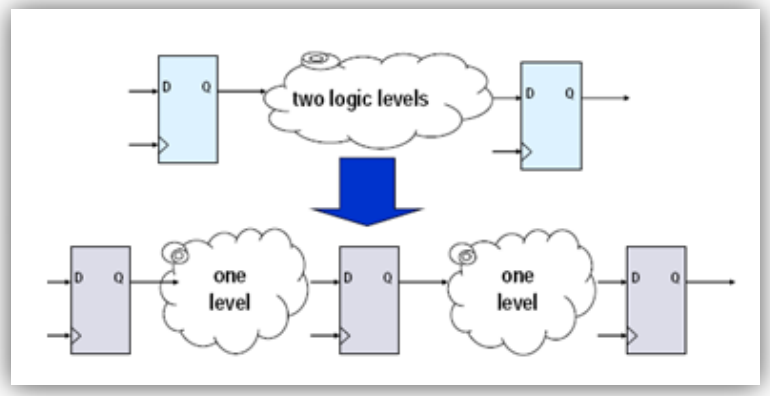
通过数据的串并转换完成速度与面积之间的灵活转换，可大大增加系统数据处理能力。

乒乓操作

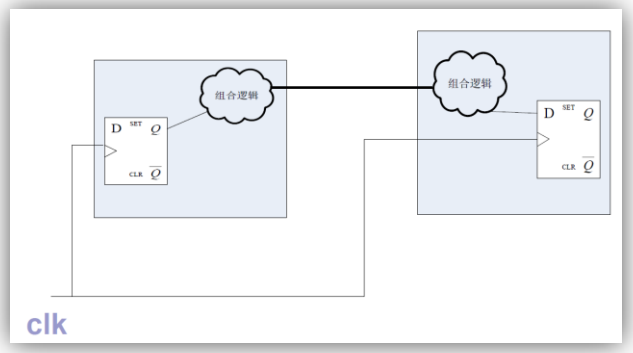

乒乓操作是跨时钟域数据处理、串并转换的主要实现手段之一。
Ram_w_addr = {W_addr,banksel}
W_data
W_addr
banksel
Ram_r_addr = {r_addr,~banksel}
R_data
R_addr
乒乓RAM

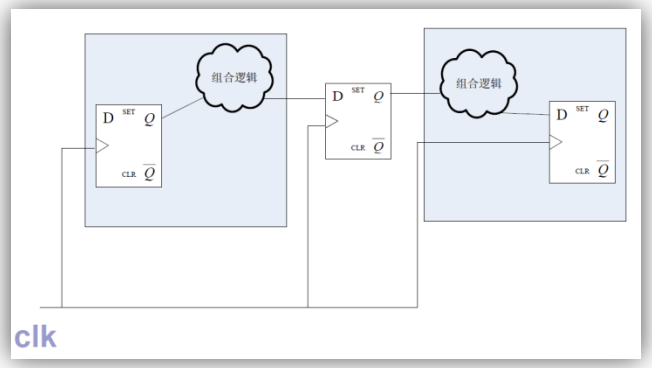
流水线操作





流水线操作的特点：各个电路在一个时钟周期内并行地完成自己的信号变换，是提高电路工作频率的主要方法之一。





典型电路：同步复位与异步复位

```
/*------------------------------------------------------------
gen the cnt signal
------------------------------------------------------------*/
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
        cnt <= 8'd0;
    else if ( flag )
        cnt <=  cnt + 8'd1;
    else
        cnt <= cnt;
end
```

信号的
异步复位

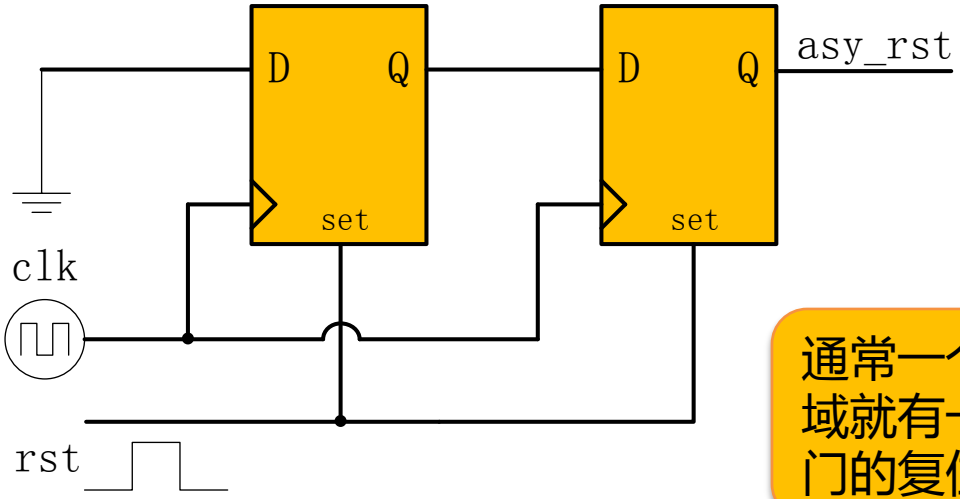
```
/*------------------------------------------------------------
gen the cnt signal
------------------------------------------------------------*/
always @ ( posedge clk )
begin
    if ( asy_rst )
        cnt <= 8'd0;
    else if ( flag )
        cnt <=  cnt + 8'd1;
    else
        cnt <= cnt;
end
```

信号的
同步复位

采用同步复位会占用一些额外的资源~

典型电路

异步复位同步释放



通常一个时钟域就有一个专门的复位电路~

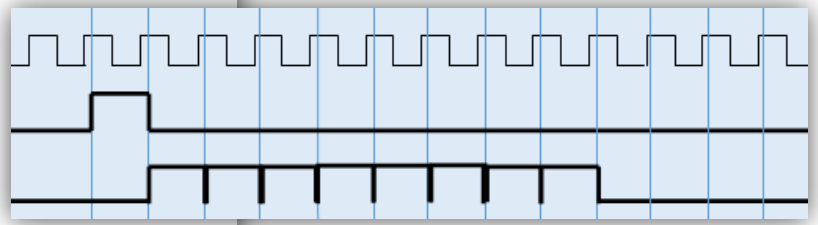
```
always @ (posedge clk or posedge rst)
begin
    if (rst)
        rst_shft[1:0]<= 2'b11;
    else
        rst_shft[1:0]<= {rst_shft[0] ,1'b0};
end
assign rst_syn = rst_shft[1];
```

典型电路：上升与下降沿的捕获

```
/*-------------------------------------------------------------
gen the sdi_posedge,sdi_negedge signal
-------------------------------------------------------------*/
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
    begin
        sdi_d1 <= 1'b0;
        sdi_d2 <= 1'b0;
    end
    else
    begin
        sdi_d1 <= i_sdi;
        sdi_d2 <= sdi_d1;
    end
end

assign sdi_posedge = ({sdi_d1,sdi_d2} == 2'b10) ? 1'b1: 1'b0;
assign sdi_negedge = ({sdi_d1,sdi_d2} == 2'b01) ? 1'b1: 1'b0;
```

典型电路：单bit信号的展宽与跨时钟传递

```
always @ (posedge asy_rst or posedge clk1)
begin
    if (asy_rst)
        hd_shift[7:0] <= 4'd0;
    else
        hd_shift[7:0] <= { hd_shift[6:0], hd_siganl};
end

assign hd_ext = |hd_shift;


always @ (posedge asy_rst or posedge clk2)
begin
    if (asy_rst)
        asy_samp[2:0] <= 3'd0;
    else
        asy_samp[2:0] <= { asy_samp[1:0], hd_ext};
end

assign hd_sigan2 = asy_samp[1] & (~asy_samp[2]);
```



典型电路：有限状态机FSM

【Mealy型】

输出取决于输入，也取决输出无关。

【Moore型】

输出取决于输入，与输出无关。

通常采用三段式，更加规范。

```
//------------------------------------------------------------
// gen the curr_state signal for state machine
//------------------------------------------------------------
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
        curr_state <= IDLE;
    else
        curr_state <= next_state;
end
```

```
//------------------------------------------------------------
// gen the next_state signal for state machine
//------------------------------------------------------------
always @ ( curr_state )
begin
    next_state = 8'hxx;
    begin
        case ( curr_state )
            {IDLE}    : next_state = STATE_1;
            {STATE_1} : next_state = STATE_2;
            {STATE_2} : next_state = STATE_3;
            {STATE_3} : next_state = STATE_4;
            {STATE_4} : next_state = IDLE;
            default   : next_state = curr_state;
        endcase
    end
end
```

```
//------------------------------------------------------------
// gen the jug_start signal
//------------------------------------------------------------
always @ ( posedge clk or posedge asy_rst )
begin
    if ( asy_rst )
        start <= 1'b0;
    else if ( ( curr_state == IDLE ) && i_recog_vld )
        start <= 1'b1;
    else
        start <= 1'b0;
end
assign o_start = start;
```

后续交流课程的一些说明

【第三讲】

◆**Modelsim/Questa Sim的使用**

◆**Altera开发工具的使用（Quartus12.0、SignalTap）**

◆**Xilinx开发工具的使用（ISE13.4、ChipScope）**

后续会上传部分软件操作的实验视频
敬请关注！