

## 第四讲

### 仿真步骤

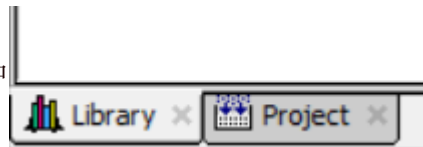
编译完成后仿真验证

从最顶层文件开始

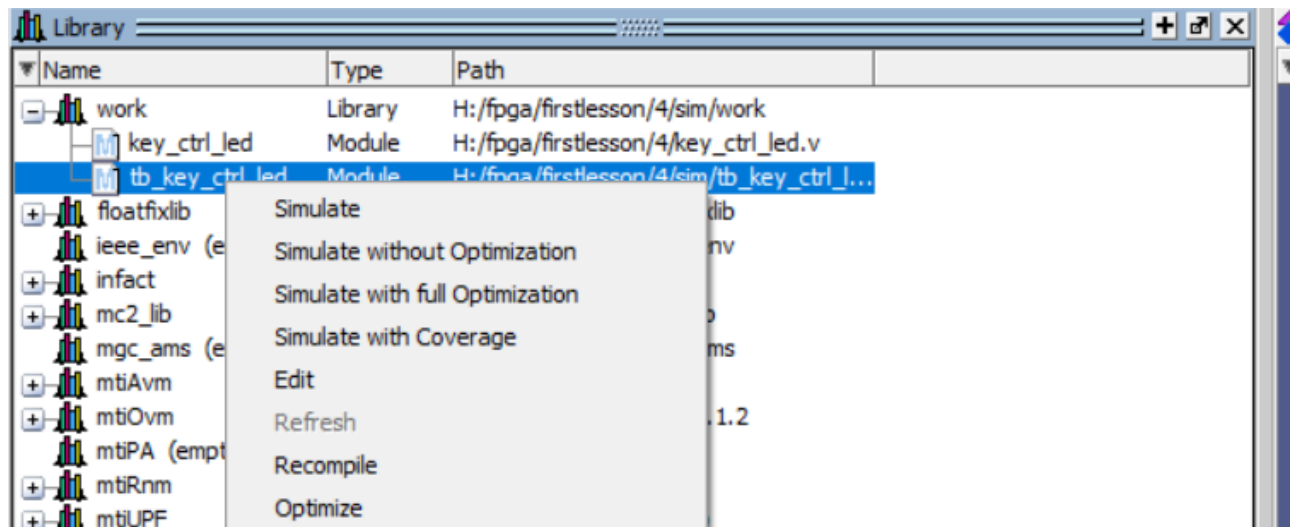
`tb_key_ctrl_led`

测试激励

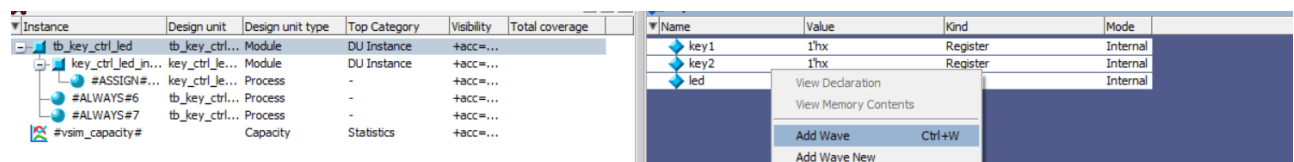
从 `project` 切换到 `library` 中



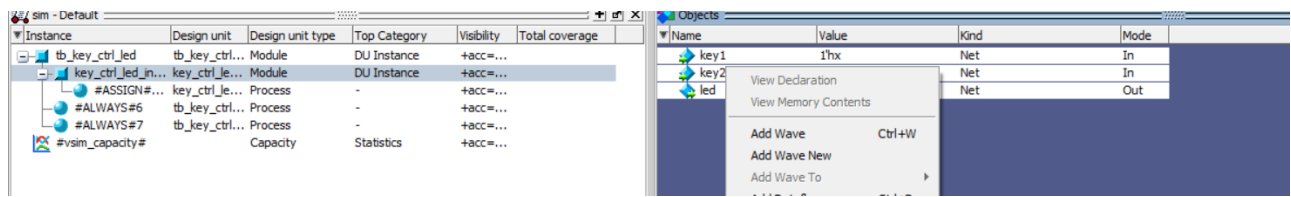
对 `tb_key_ctrl_led` 仿真



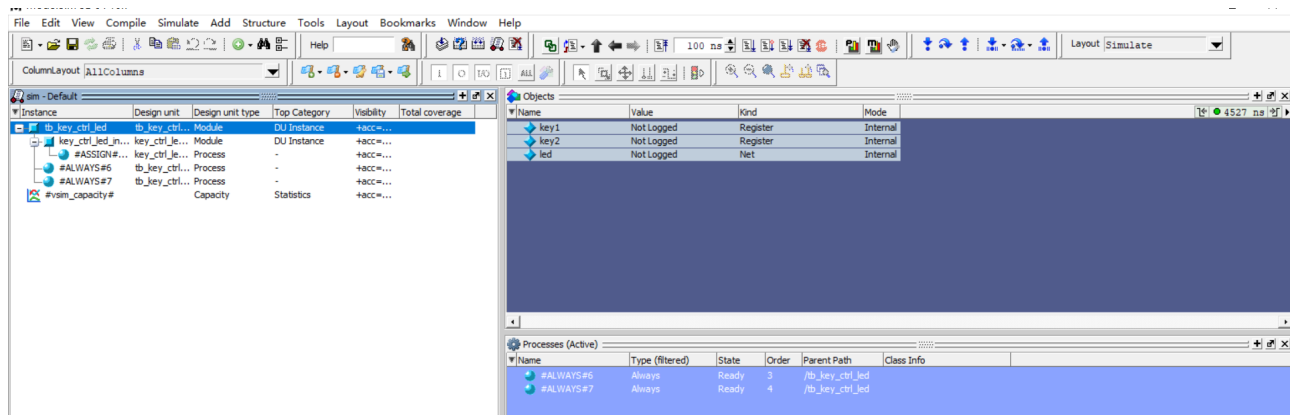
对 `tb_key_ctrl_led` 添加波形



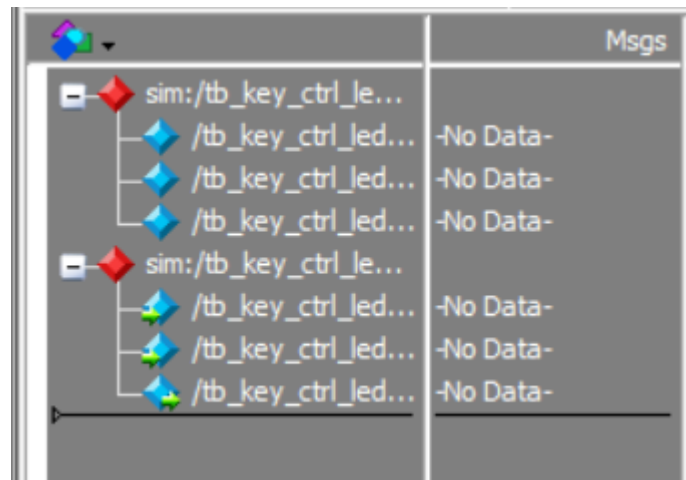
对 `key_ctrl_led` 添加波形



view可以打开误操作关闭的窗口



ctrl+g 分组



从左往右指向模块里面是输入，指向模块外面是输出

run

## Wave放大缩小操作



放大

缩小

变为初始状态

放大时标的位置（黄线）

只显示信号名称不显示路径

Display

Grid & Timeline

⏪ ⏩

Display Signal Path

1 (# elements)  
Use 0 for full path

Snap Distance

10 (pixels)

Show Radix Base

☐ Waveforms

Row Margin

4 (pixels)

Justify Value

☒ Left ☐ Right

Child Row Margin

2 (pixels)

Enable/Disable

☒ Waveform popup showing data value  
☐ Waveform selection highlighting  
☒ Scroll to end when run completes  
☐ On close, ask about saving window contents  
☒ On close, ask about saving editable wave commands  
☐ PA waveform highlighting

Double-click will: Find Active Driver

Dataset Prefix Display

☐ Always show  
☒ Show if 2 or more  
☐ Never show

Display Signal Array Size

☐ Show vector and array indices

OK

Cancel

Apply

	Msgs
sim:/tb_key_ctrl_led/Group1	
/tb_key_ctrl_led/key1	1'hx
/tb_key_ctrl_led/key2	1'hx
/tb_key_ctrl_led/led	1'hx
sim:/tb_key_ctrl_led/key_ctrl_led_inst/Group1	
/tb_key_ctrl_led/key_ctrl_led_inst/key1	1'hx
/tb_key_ctrl_led/key_ctrl_led_inst/key2	1'hx
/tb_key_ctrl_led/key_ctrl_led_inst/led	1'hx

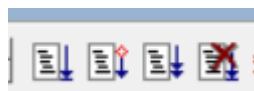
	Msgs
sim:/tb_key_ctrl_led/Group1	
key1	1'hx
key2	1'hx
led	1'hx
sim:/tb_key_ctrl_led/key_ctrl_led_inst/Group1	
key1	1'hx
key2	1'hx
led	1'hx

代码有错误重新wave

代码有错误重新显示波形wave的话

直接编译，编译完了在wave页面restart

run的使用



run

一直run

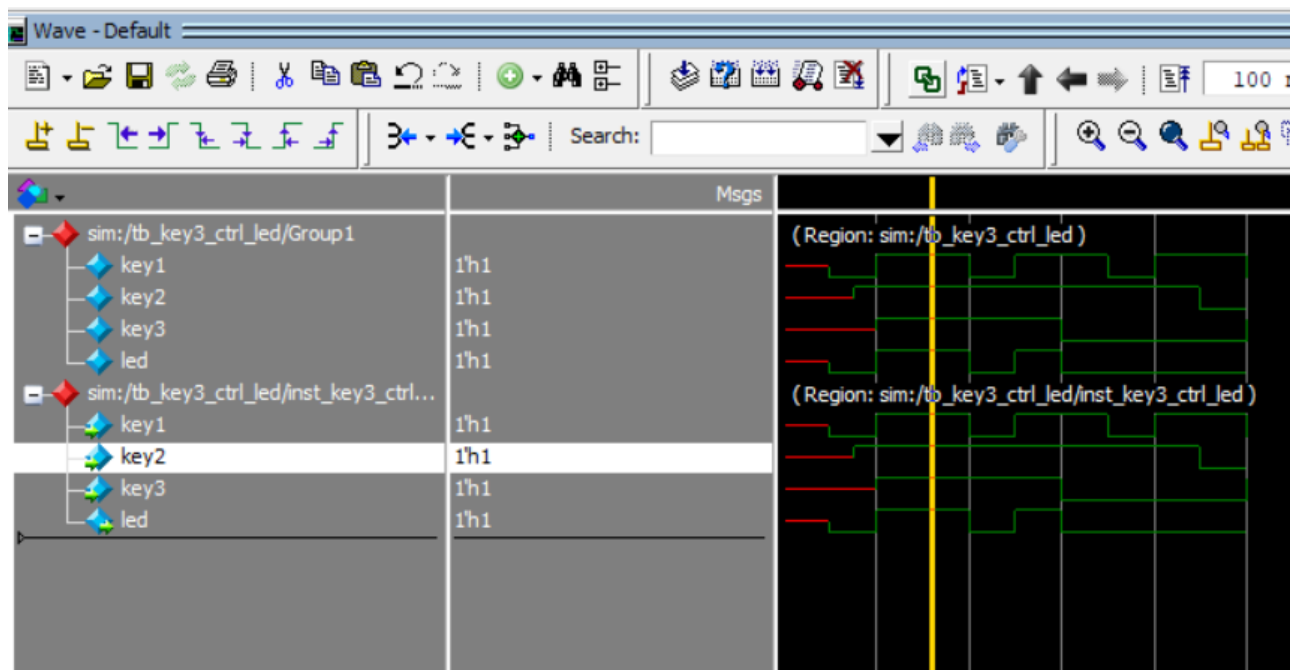
run到stop位置

run到中断位置

## homework

```
module key3_ctrl_led(  
    input wire key1,  
    input wire key2,  
    input wire key3,  
    output wire led  
);  
assign led = key1&key2&key3;  
endmodule
```

```
`timescale 1ns/1ns  
module tb_key3_ctrl_led();  
    reg key11;  
    reg key21;  
    reg key31;  
    wire led1;  
    always #10 key11={$random};  
    always #15 key21={$random};  
    always #20 key31={$random};  
    key3_ctrl_led inst_key3_ctrl_led (  
        .key1(key11),  
        .key2(key21),  
        .key3(key31),  
        .led(led1)  
    );  
endmodule
```



tb文件的信号

内部测试的信号

两者通过接口相连。通过.()相连

## 第五讲

```
always @(posedge clk) begin
    po_c1 <= pi_a&pi_b;
end
assign po_c2 = pi_a&pi_b;
```

时序一样吗？

不一样，`po_c1`是时序逻辑，随着时钟的上升沿触发，非阻塞赋值（没有延迟）；`po_c2`是组合逻辑，阻塞赋值（有延迟）。

## 时序+组合电路

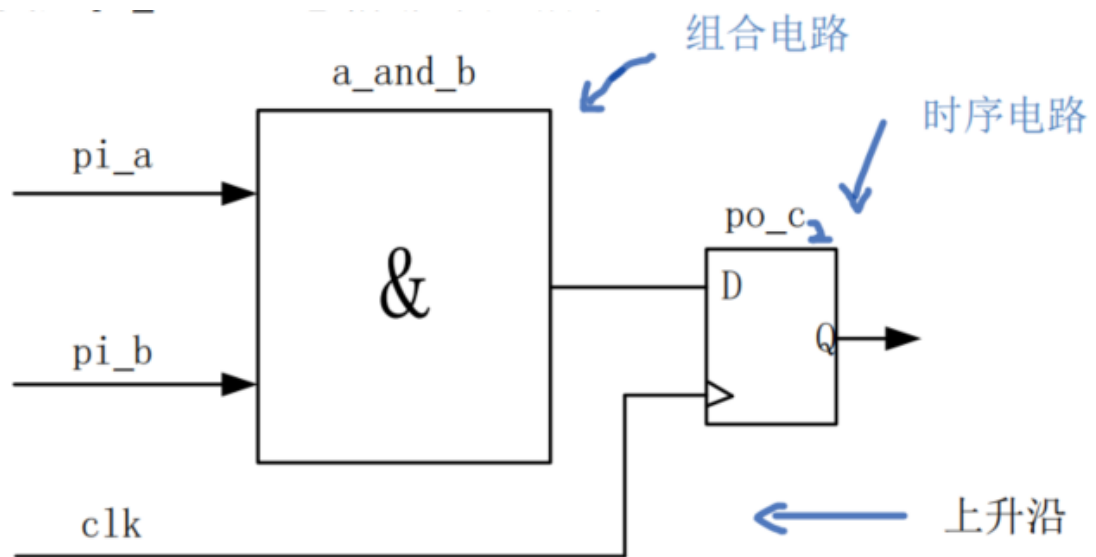


图 3 RTL 电路图

使用 Verilog HDL 描述出图 3 给出的 RTL 电路图

```
module a_and_b(  
    input wire pi_a,  
    input wire pi_b,  
    input wire clk,  
    output wire po_c1, // 组合逻辑  
    output reg po_c2 // 时序逻辑  
);  
    assign po_c1 = pi_a & pi_b; // wire 类型的是 assign  
  
    always @(posedge clk) begin // reg 类型的是 always  
        po_c2 <= pi_a & pi_b; // 将 pi_a 和 pi_b 以时钟的形式赋值给 po_c2  
    end  
  
endmodule
```

测试模块

```
`timescale 1ns/1ns  
module tb_a_and_b();  
    // 输入
```



```

reg pi_a,pi_b,inc1k;
//输出
wire c1,c2;

//给输入激励
always #15 pi_a={$random};
always #20 pi_b={$random};

initial begin
    inc1k=0;
end
always #10 inc1k=~inc1k;

//实例化
a_and_b a_and_b_inst(
    .pi_a(pi_a),
    .pi_b(pi_b),
    .clk(inc1k),
    .po_c1(c1),
    .po_c2(c2)
);
endmodule

```

## 变化位宽

```

module test_bit(
input wire clk,
input wire [1:0] pi_a,
input wire [2:0] pi_b,
output reg [3:0] po_c
);

parameter w=15; //不能在接口里面使用parameter
reg [7:0] counter;
reg [7:0] counter2=8'd255;//8'hff //8'b1111_1111
//十进制 8'd81 十六进制 8'h51 二进制 8'b0101_0001
//255默认是32bit

//reg [3:0] 代表低四位 counter2=255;

```

```
//4'b1111 4'd15 4'hf
```

```
//reg [3:0] 代表低四位 counter2=81
```

```
//counter2=4'b0001; 4'd1 4'h1
```

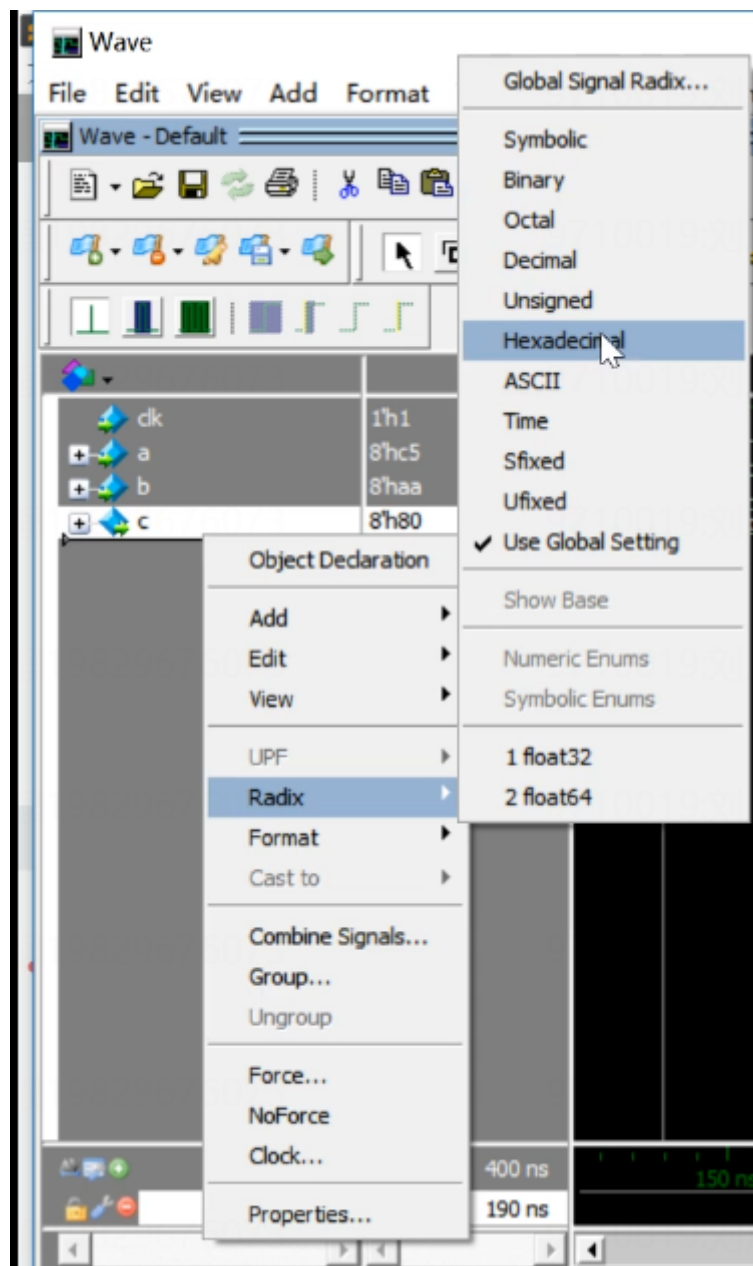
```
wire [w:0] interface_a;
```

```
always @(posedge clk) begin
```

```
    po_c <= pi_a & pi_b;
```

```
end
```

```
endmodule
```



## 课后题

按照表 1 中定义的信号和模块名称编写功能模块代码，其中 c 变量是 a 和 b 的‘按位与’的结果。

编写测试激励代码，a 和 b 的激励为随机 8 位 0~255 的数据。

注释：8 位 0~255 激励数据生成方法为  $\{\$random\} \% 256$

这里使用了%是求模运算，返回值是除以 256 后的余数。

test_a4_and_b4	clk	input	wire	1
test_a4_and_b4	a	input	wire	8
test_a4_and_b4	b	input	wire	8
test_a4_and_b4	c	output	reg	8

```
module a_and_b_bit(  
    input wire clk,  
    input wire [7:0] a,  
    input wire [7:0] b,  
    output reg [7:0] c  
);  
    //按位与  
    //8'b1001_0010  
    //8'b1001_1101  
    //8'b1001_0000  
    always @(posedge clk) begin  
        c <= a&b;  
    end  
  
endmodule
```

```
`timescale 1ns/1ns  
module tb_a_and_b_bit();  
    reg [7:0] a,b;
```

```

reg inc1k;
wire c; //一位 只显示最低的一位 外部的
        //wire [7:0] c; 波形就不是一条线了，而是七位的，这个代表外部的测试波形

initial begin
    a=0;
    b=0;
    inc1k=0;
end

always #20 a={$random}%256;
always #20 b={$random}%256;

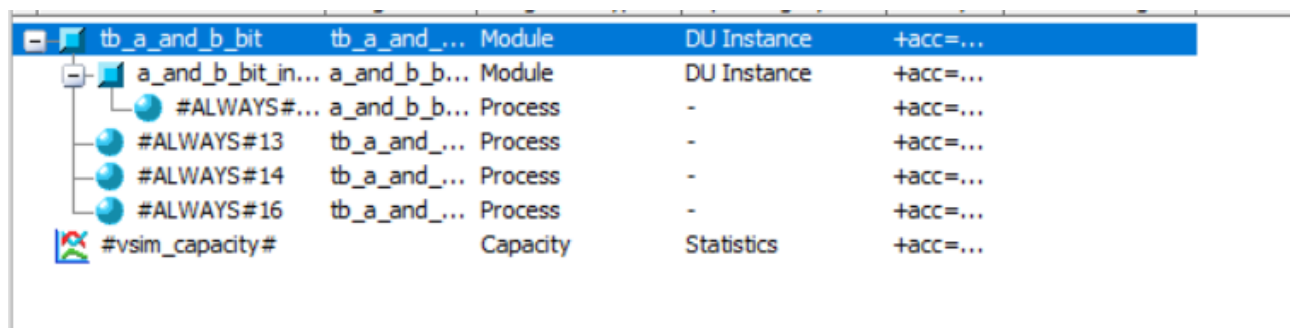
always #10 inc1k=~inc1k;

// 内部的接口模型 根据第一个代码中定义的，是八位
a_and_b_bit a_and_b_bit_inst(
    .clk(inc1k),
    .a(a),
    .b(b),
    .c(c)
);

endmodule

```

对应上下两个波形



	tb_a_and_b_bit	tb_a_and_...	Module	DU Instance	+acc=...
	a_and_b_bit_in...	a_and_b_b...	Module	DU Instance	+acc=...
	#ALWAYS#...	a_and_b_b...	Process	-	+acc=...
	#ALWAYS#13	tb_a_and_...	Process	-	+acc=...
	#ALWAYS#14	tb_a_and_...	Process	-	+acc=...
	#ALWAYS#16	tb_a_and_...	Process	-	+acc=...
	#vsim_capacity#		Capacity	Statistics	+acc=...

