

# 数字电路与数字系统实验

## 实验四 触发器

姓名： 你猜

学号： 你猜

班级： 你猜

邮箱： 你猜

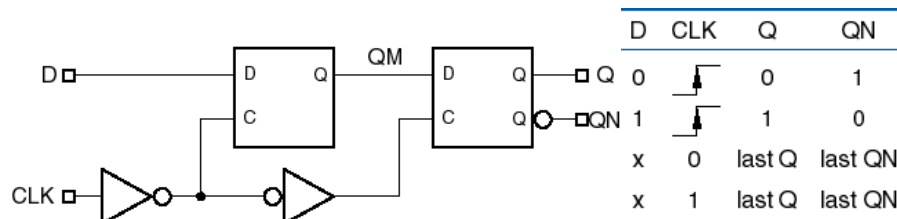
实验时间： 你猜

## 一、实验目的

学习锁存器和触发器的原理，通过 verilog 语言设计、实现一个异步 D 触发器和一个同步 D 触发器，了解 verilog 语言中阻塞赋值和非阻塞赋值的区别和用法。

## 二、实验原理

锁存器是通过激励输入的电平来控制元件的状态。触发器是时钟信号的边沿向触发器发命令，触发器根据激励信号改变状态。



异步清零，是指与时钟不同步，即清零信号有效时，无视触发脉冲，立即清零；同步是时钟触发条件满足时检测清零信号是否有效，有效则在下一个时间周期的触发条件下，执行清零。

## 三、实验环境

Quartus 18.1、FPGA 开发板

## 四、实验过程

设计思路：

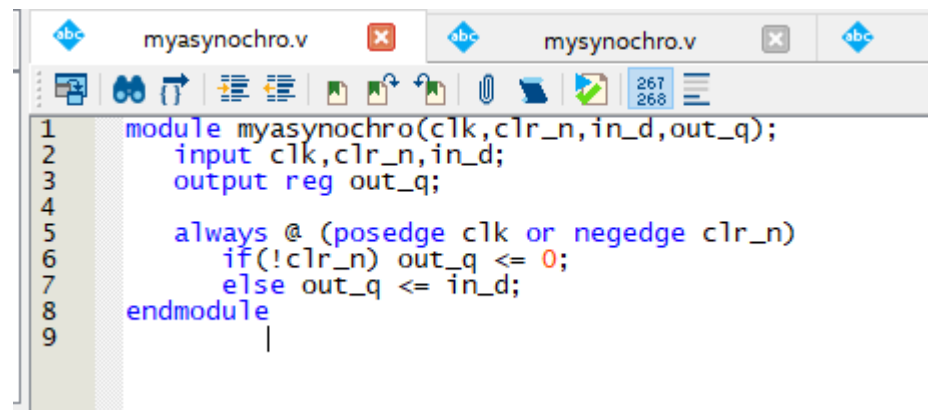
clk 作为控制输入端，clr\_n 作为清零端，in\_d 作为输入数据，out\_q 和 out\_q2 作为输出。先参考实验手册里的例子，设计一个同步清零的上升沿 D 触发器 mysynochro，若清零信号有效，则在下一个时钟周期清零。再修改代码，设计一个异步清零的触发器 myasynochro，这个触发器只要清零信号有效，且使能端为 1，则立即清零。

设计代码：

trigger.v

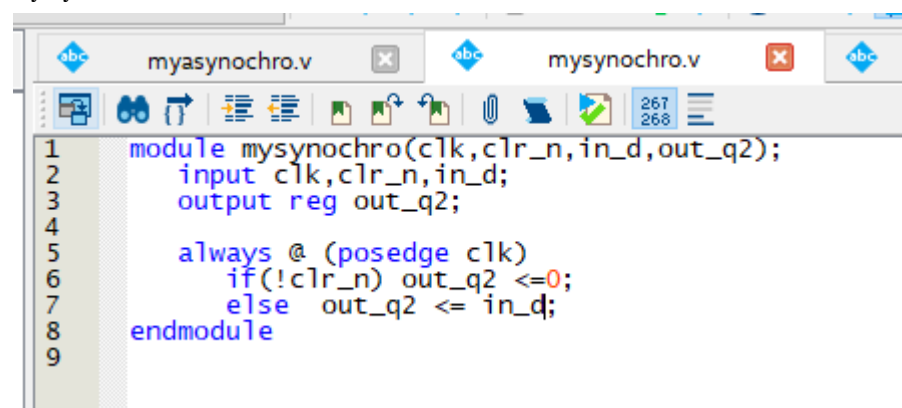
```
myasynochro.v  mysynochro.v  trigger.v
1 module trigger(clk,clr_n,in_d,out_q,out_q2);
2   input clk,clr_n,in_d;
3   output out_q,out_q2;
4
5   myasynochro asco(clk,clr_n,in_d,out_q);
6   mysynochro sco(clk,clr_n,in_d,out_q2);
7
8 endmodule
9
```

mysynochro.v



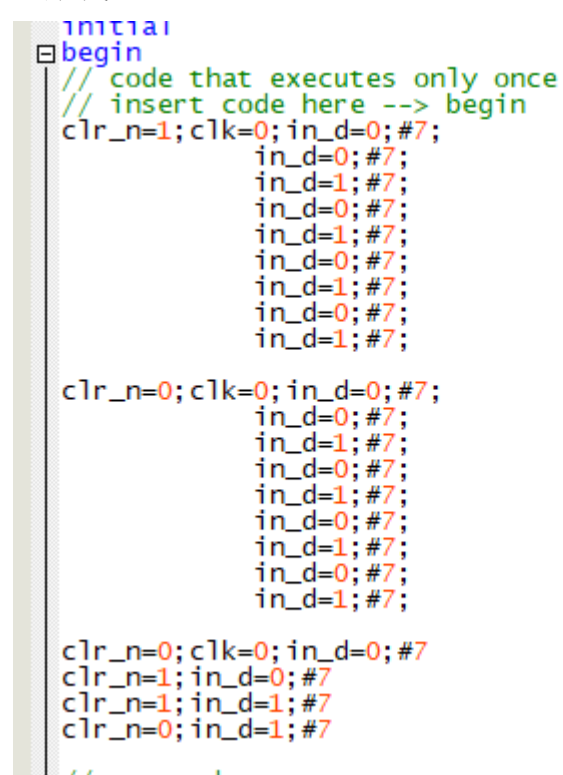
```
1 module mysynochro(clk,clr_n,in_d,out_q);
2     input clk,clr_n,in_d;
3     output reg out_q;
4
5     always @ (posedge clk or negedge clr_n)
6         if(!clr_n) out_q <= 0;
7         else out_q <= in_d;
8 endmodule
9
```

mysynochro.v



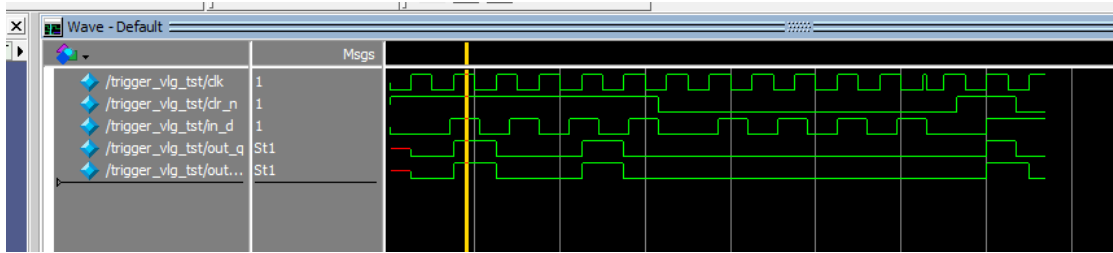
```
1 module mysynochro(clk,clr_n,in_d,out_q2);
2     input clk,clr_n,in_d;
3     output reg out_q2;
4
5     always @ (posedge clk)
6         if(!clr_n) out_q2 <=0;
7         else out_q2 <= in_d;
8 endmodule
9
```

激励代码:



```
1 initial
2 begin
3     // code that executes only once
4     // insert code here --> begin
5     clr_n=1;clk=0;in_d=0;#7;
6         in_d=0;#7;
7         in_d=1;#7;
8         in_d=0;#7;
9         in_d=1;#7;
10        in_d=0;#7;
11        in_d=1;#7;
12        in_d=0;#7;
13        in_d=1;#7;
14
15    clr_n=0;clk=0;in_d=0;#7;
16        in_d=0;#7;
17        in_d=1;#7;
18        in_d=0;#7;
19        in_d=1;#7;
20        in_d=0;#7;
21        in_d=1;#7;
22        in_d=0;#7;
23        in_d=1;#7;
24
25    clr_n=0;clk=0;in_d=0;#7
26    clr_n=1;in_d=0;#7
27    clr_n=1;in_d=1;#7
28    clr_n=0;in_d=1;#7
29
30    // < end
```

ModelSim 仿真：



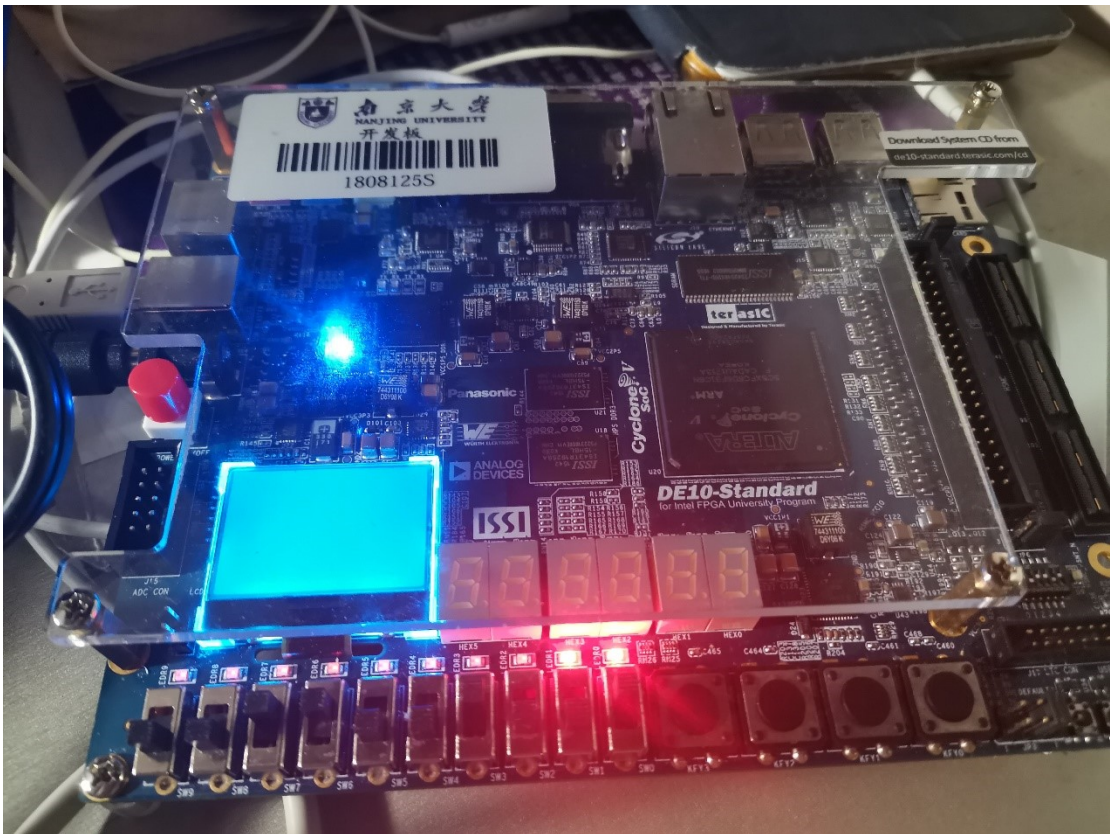
引脚分配：

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Res
in clk	Input	PIN_AJ4	3B	B3B_N0	2.5 V (default)	
in clr_n	Input	PIN_AK4	3B	B3B_N0	2.5 V (default)	
in in_d	Input	PIN_AB30	5B	B5B_N0	2.5 V (default)	
out out_q	Output	PIN_AA24	5A	B5A_N0	2.5 V (default)	
out out_q2	Output	PIN_AB23	5A	B5A_N0	2.5 V (default)	
<<new node>>						

实验结果：

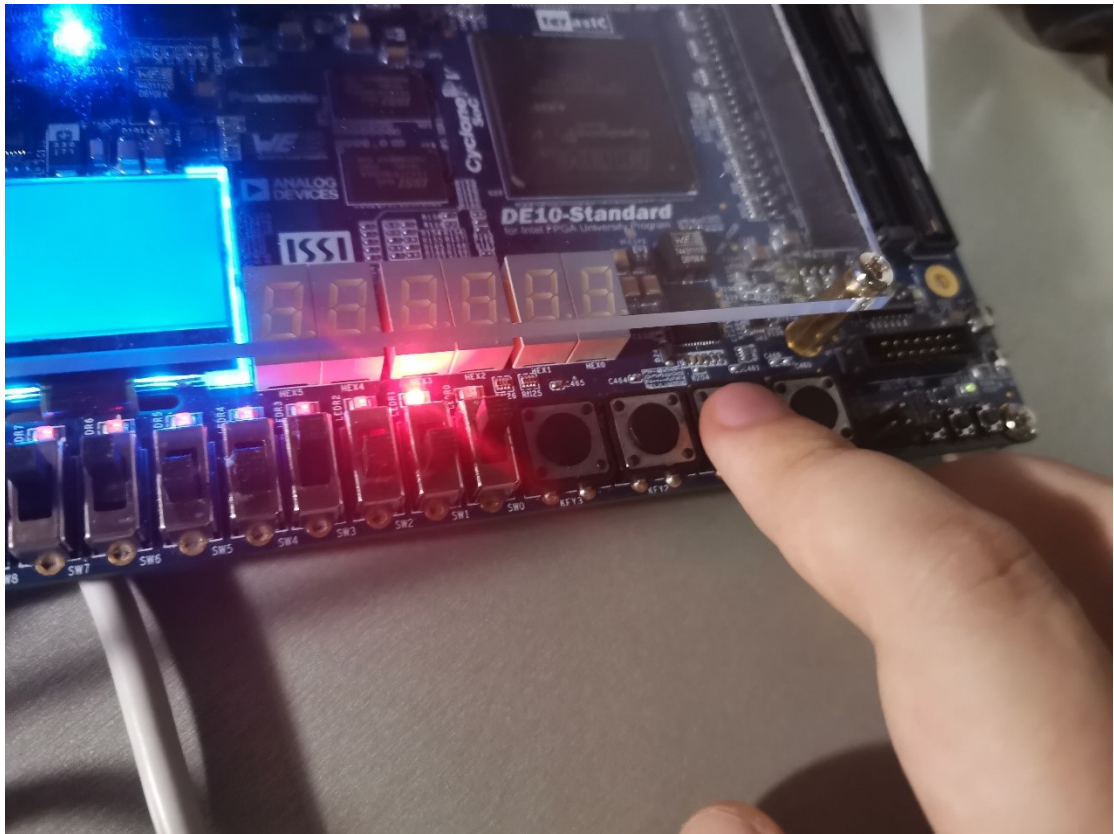
button0 是 clk, button1 是 clr\_n 即清零控制端, LED0 是异步触发器输出, LED1 是同步触发器输出, SW0 是 in\_d。

首先, 通过 button0, 产生一个上升沿, LED0 和 LED1 亮。

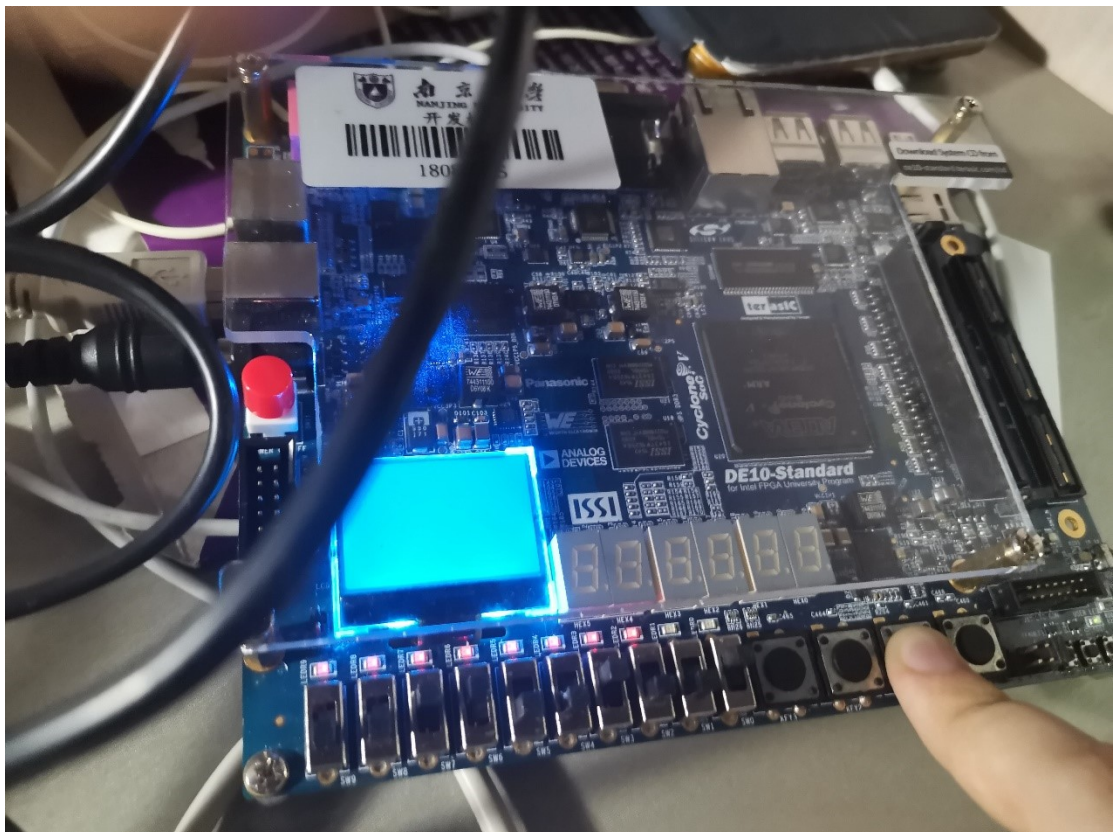




按住 button1, 检测到清零信号, 异步触发器立即清零, LED0 熄灭。



按住 button1 后, 按 button0 进行下一个时钟周期, 同步触发器在这时清零, LED1 熄灭。



## 五、 实验中遇到的问题及解决办法

在开始的设计中，没有注意到实验手册上的提示，

大家在设计异步清零的触发器的时候，如果触发器的清零信号（假设命名为 `clr_n`）是“0”有效，那么在敏感列表中应该是检测 `clr_n` 的下降沿，即代码应该这样：

```
1 always @(posedge clk or negedge clr_n)
2   if(!clr_n)
3     begin ... end
4   else ...
```

没有考虑到若清零信号是 0 有效的话，敏感列表中应该检测 `clr_n` 的下降沿，造成了不必要的错误。

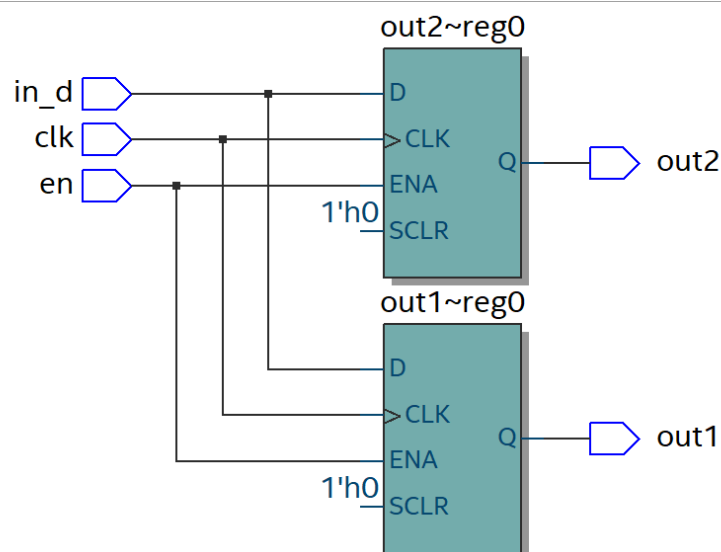
## 六、 启示

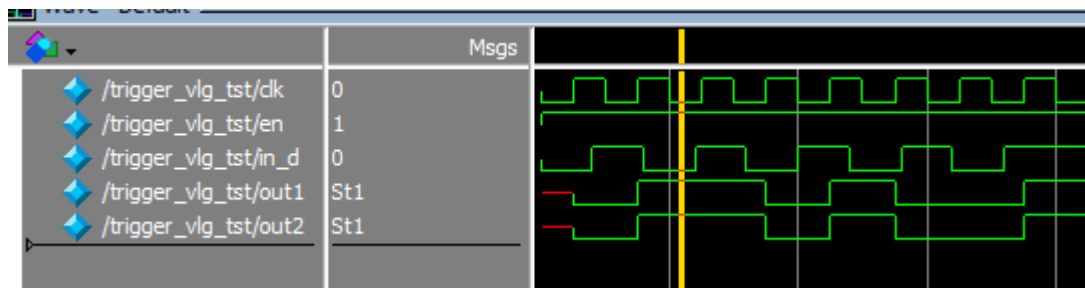
在时序逻辑电路的设计中，要使用非阻塞赋值语句，采用赋值符号 `<=`。在组合逻辑电路的设计中，要使用阻塞赋值语句，采用赋值符号 `=`。

阻塞赋值语句是立即赋值语句，其形式和作用都类似于其他任何过程语言（如 C 语言）的赋值语句。阻塞赋值语句在语句执行时，首先计算赋值语句右边的表达式的值，得到结果后立即将值赋给赋值语句左边的变量。

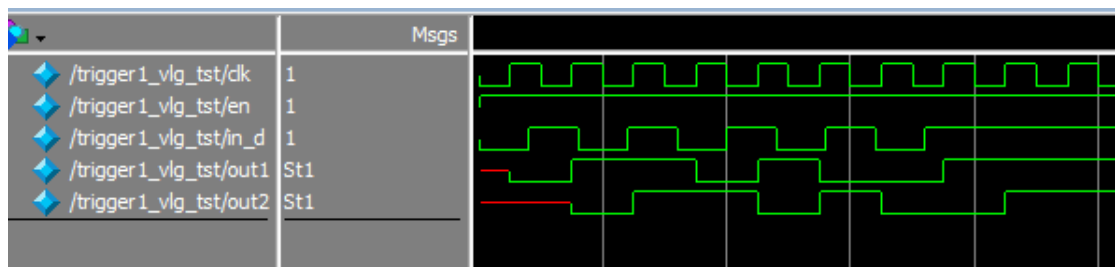
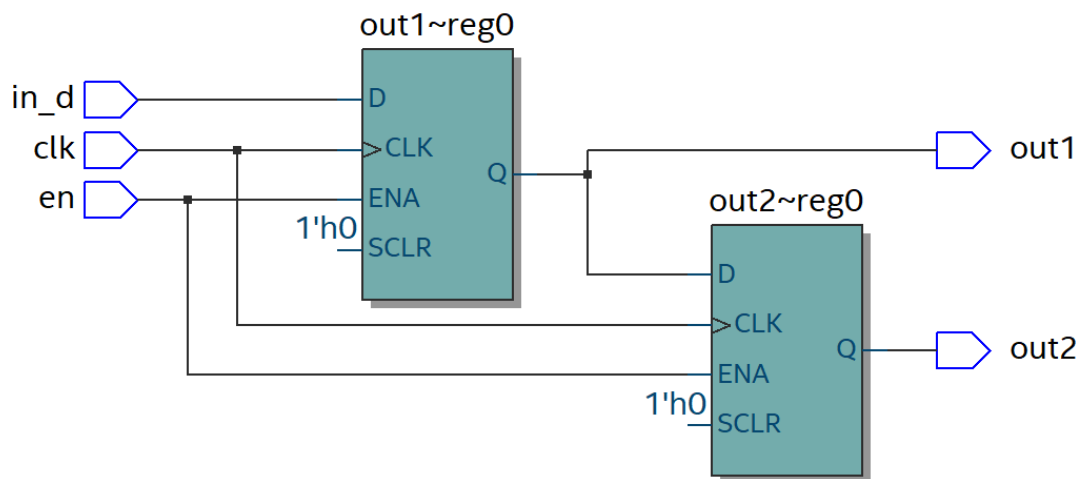
而非阻塞赋值语句却不同，非阻塞语句一般出现在 `always` 语句块中，非阻塞语句在执行时，虽然也是立即计算赋值语句右边的表达式的值，但却不将结果立即赋值给表达式左边，要等到整个 `always` 块执行完毕后，经过一个无穷小的延时才完成赋值。

分析阻塞与非阻塞 RTL 视图和仿真结果：  
阻塞赋值的两个触发器：





非阻塞赋值的两个触发器：



非阻塞赋值语句写的触发器会在 always 语块执行完之后，再将输入和原 out1 的值分别赋给 out1 和 out2；而阻塞赋值语句会在执行 always 语块的时候，将输入赋值给 out1，再立即将新 out1 的值赋值给 out2，而不是原来 out1 的值。

## 七、 意见与建议

虽然之前我并没有上过数字电路这门课，但是实验手册前面的讲解非常的清楚，由浅入深，帮助我学习和完成了这次实验。