



# 南京大学

## 《数字电路与数字系统实验》实验报告

实验四： 触发器与锁存器

姓名： 毛彦杰

学号： 191220081

班级： 数字电路与数字系统实验 2 班

院系： 计算机科学与技术系

邮箱： 1363818182@qq.com

实验时间： 2020/9/26

预习报告：

锁存器和触发器是时序电路的基本构件。锁存器和触发器都是由独立的逻辑门电路和反馈电路构成的，锁存器在时钟信号为有效电平的整个时间段，不断监测其所有的输入端，此段时间内的任何满足输出改变条件的输入，都会改变输出；触发器只有在时钟信号变化的瞬间才改变输出值。

一、RS 锁存器

RS 锁存器中有一个  $Q$  和  $\bar{Q}$  同时为 1 的无效状态，这是  $R$  和  $S$  同时为 1 的缘故，如果强制  $R$  和  $S$  总是相反的逻辑，就可以避免这一现象产生。如图 4-3 所示，这个电路就是 D 锁存器电路，当时钟触发信号为 0 时，输出保持不变，当时钟触发信号为 1 时， $Q$  输出  $D$  的值，即  $Q$  随着  $D$  值的改变而改变。

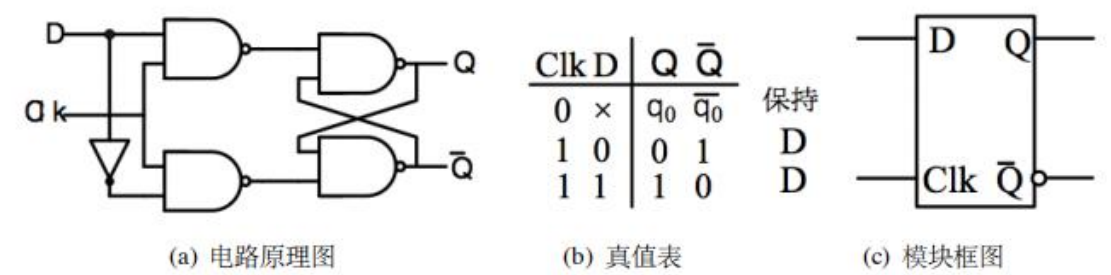


图 4-3: D 锁存器

表 4-1: D 锁存器代码

```
1 module D_latch(clk,in_d,en,out_q1);
2   input clk;
3   input in_d;
4   input en;
5   output reg out_q1;
6
7   always @ (*)
8     if ( en )
9       begin
10         if (clk) out_q1 <= in_d;
11       end
12     else
13       out_q1 <= out_q1;
14 endmodule
```

## 二、D 触发器

前面讨论的锁存器都是在时钟为高电平（也可以设计为低电平）时触发的，如果我们希望锁存器只在时钟的特定时刻（如上升沿或者下降沿）触发，锁存此时刻 D 的值，这样的锁存器通常称为时钟边沿触发的触发器。

用两个锁存器可以构成触发器，如图 4-5(a)所示。图中的两个 D 锁存器，前者为主锁存器，后者为从锁存器，当 Clk 信号为 1 时，主锁存器的  $Q_m$  随着 D 的变化而变化，从锁存器的状态保持不变。当时钟信号变为 0 后，主锁存器的状态不再变化，从锁存器  $Q_s$  的状态则跟随  $Q_m$  状态的变化而变化，由于当  $\text{Clk}=0$  时， $Q_m$  不会发生变化，因此对于外部的观察者而言，在一个时钟周期内  $Q$  只在 Clk 从 1 变为 0（即时钟的负跳变沿或下降沿）的时候发生一次变化。因此，我们也可以说输出信号  $Q$  是在时钟下降沿采集到的输入信号 D 的瞬间值。

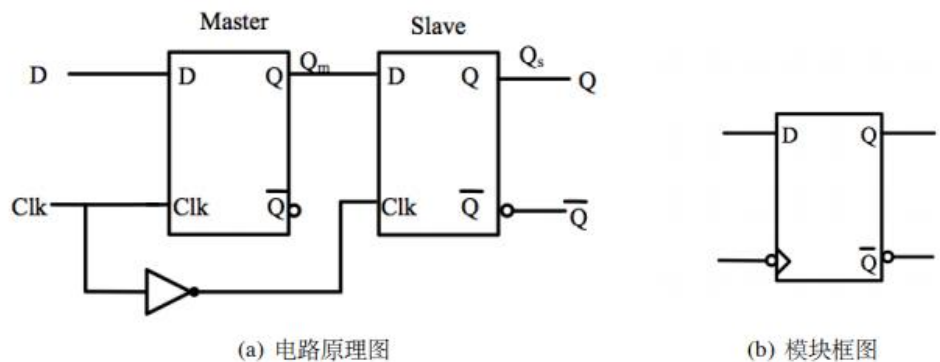


图 4-5: D 触发器

下面以一个上升沿触发的 D 触发器为例，说明触发器的设计方法。常见的触发器的 Verilog HDL 代码及其测试代码如表 4-3 和 4-4 所示。

表 4-3: D 触发器代码

```
1 module D_trigger(clk,in_d,en,out_gt);
2     input clk;
3     input in_d;
4     input en;
5     output reg out_gt;
6
7     always @ (posedge clk )
8         if(en)
9             out_gt <= in_d;
10        else
11            out_gt <= out_gt;
12 endmodule
```

表 4-4: D 触发器测试代码

```

1  `timescale 10 ns/ 1 ps
2  module D_latch_vlg_tst();
3      reg clk;
4      reg in_d;
5      reg en;
6
7      // wires
8      wire out_qt;
9
10     D_trigger i1 (
11         // port map - connection between master ports and signals/registers
12         .clk(clk),
13         .in_d(in_d),
14         .en(en),
15         .out_qt(out_qt)
16     );
17
18     initial
19     begin
20         // code that executes only once
21
22         clk = 0; in_d = 0; en = 0; #7;
23             in_d = 0; #7;
24             in_d = 1; #7;
25             in_d = 0; #7;
26         en = 1; #7;
27             in_d = 0; #7;
28             in_d = 1; #7;
29             in_d = 0; #7;
30             in_d = 1; #7;
31         en = 0; #7;
32             in_d = 0; #7;
33             in_d = 1; #7;
34             in_d = 0; #7;
35             in_d = 1; #7;
36
37         $stop;
38     end
39
40     always
41     begin
42         // code executes for every event on sensitivity list
43         #5 clk = ~clk;
44     end
45 endmodule

```



## \$stop

\$stop 指令是 Verilog HDL 的一个系统任务。在仿真的时候，默认的是一直仿真下去，如果在测试代码中加入 \$stop 指令的话，那么编译器仿真到 \$stop 指令时就在此处停止仿真了。\$stop 任务的作用是将 EDA 工具设置为暂停模式，在仿真环境下给出一个交互式的命令，将控制权交给用户。参数值越大，输出信息越多。

### 三、触发器设计中的非阻塞赋值语句

Verilog 语言中有两种赋值语句，之前我们使用的赋值语句采用赋值符号“=”，这种赋值被称为阻塞赋值语句；在设计触发器的时候我们使用另一种赋值语句，采用赋值符号“<=”，此赋值语句被称为非阻塞赋值语句。

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。

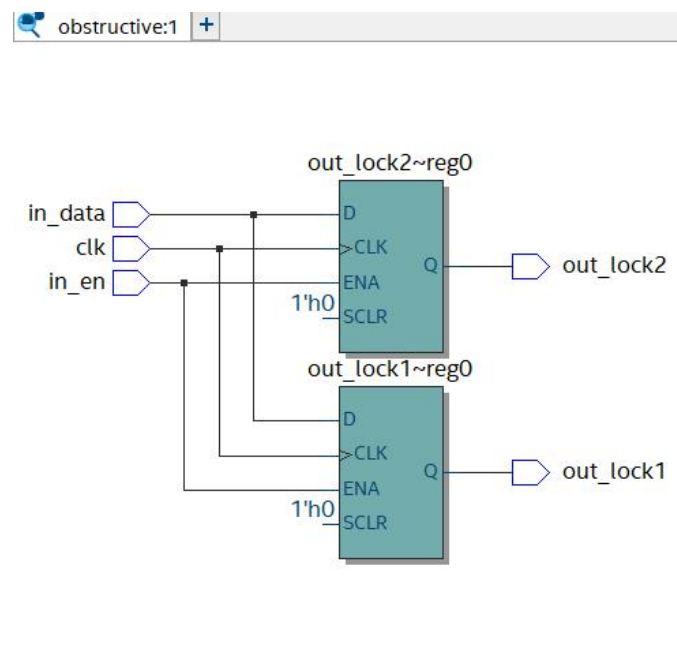
而非阻塞赋值语句却不同，非阻塞语句一般出现在 always 语句块中，非阻塞语句在执行时，虽然也是立即计算赋值语句右边的表达式的值，但却不将结果立即赋值给表达式左边，要等到整个 always 块执行完毕后，经过一个无穷小的延时才完成赋值。

#### Verilog 编码指导方针

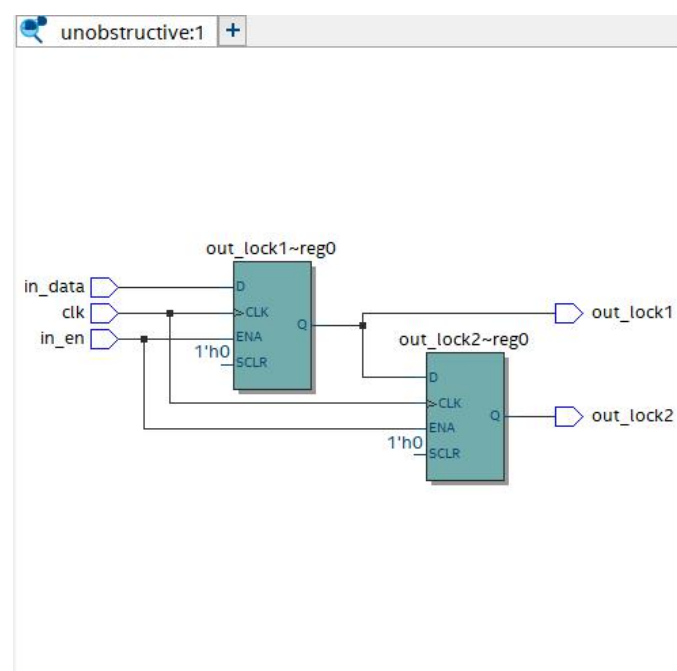
1. 当为时序逻辑电路建模时，使用“非阻塞赋值 ' <=' ”。
2. 当为锁存器（latch）建模时，使用“非阻塞赋值 ' <=' ”。
3. 当用 always 块为组合逻辑建模时，使用“阻塞赋值 ' =' ”。
4. 当一个 always 块里既有组合逻辑又有时序逻辑建模时，用“非阻塞赋值 ' <=' ”。
5. 不要在同一个 always 块里面混合使用“阻塞赋值 ' =' ”和“非阻塞赋值 ' <=' ”。
6. 不要在两个或两个以上 always 块里面对同一个变量进行赋值。
7. assign 语句使用“阻塞赋值 ' =' ”即可。

### 4.3.1 分析阻塞和非阻塞 RTL 视图和仿真结果

阻塞赋值语句生成 D 触发器的实际电路原理：



非阻塞赋值语句生成 D 触发器的实际电路原理：



由 RTL Schematic 可以看出，阻塞赋值语句生成 D 触发器由两个 D 触发器并联而成，而非阻塞赋值语句生成 D 触发器是由两个 D 触发器级联而成。

## 4.3.2 设计一个同步清零和一个异步清零的 D 触发器

### 一、实验目的

查阅资料，分析同步清零和异步清零的不同，并请在一个工程中设计两个触发器，一个是带有异步清零端的 D 触发器，而另一个是带有同步清零端的 D 触发器。

### 二、实验原理

**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；

**同步清零**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。同步清零就是把清零信号和时钟信号与或者与非处理后输入到清零端，同步清零可以保证状态在时钟的有效期内不会改变。

### 实例化要求

在利用多个模块进行设计的过程中，我们可以将所有模块放在一个文件中，也可以为每一个模块新建一个文件。**Verilog HDL 建议为每一个模块新建一个文件，文件名和模块名相同**，这样在一个工程中设计好的模块可以直接在另一个工程中被实例化。不仅提高了工程的可读性也提高了模块的可重用性。

### 三、实验设备环境

硬件器材：FPGA 开发板

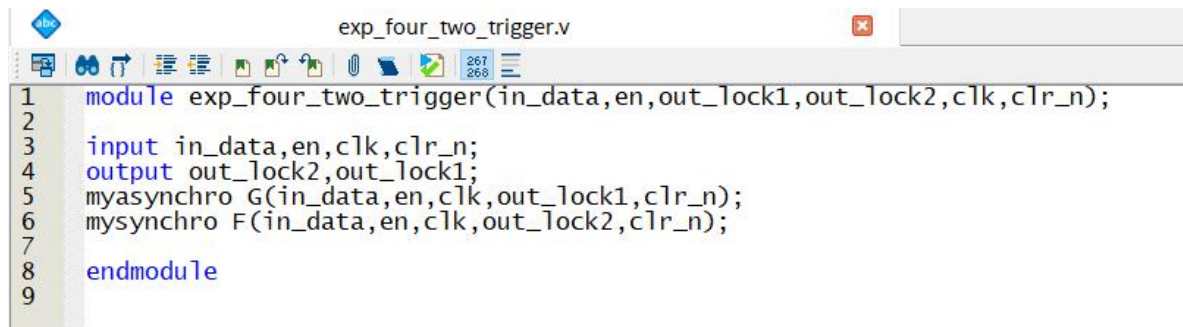
软件平台：Quartus 开发平台

## 四、实验步骤

### 设计思路：

模仿实验手册中 D 触发器的样例设计同步清零的触发器，然后通过仿真测试直到得到正确的结果。接下来再写一个异步清零的触发器，这个触发器和同步清零有所不同，只要清零端为 0，并且使能端为 1，就可以直接清零。接下来为了完成模块实例化，再添加一个顶层文件 Trigger.v。然后再一起仿真测试。

### 设计代码：



```
1 module exp_four_two_trigger(in_data,en,out_lock1,out_lock2,clk,clr_n);
2
3 input in_data,en,clk,clr_n;
4 output out_lock2,out_lock1;
5 myasynchro G(in_data,en,clk,out_lock1,clr_n);
6 mysynchro F(in_data,en,clk,out_lock2,clr_n);
7
8 endmodule
9
```



```

exp_four_two_trigger.v  mysynchro.v  myasynchro.v
1  module mysynchro(in_data,en,clk,out_lock2,clr_n);
2
3  input in_data,en,clk,clr_n;
4  output reg out_lock2;
5
6  initial out_lock2=0;
7
8  always @(posedge clk or negedge clr_n)
9  begin
10     if(!clr_n)
11     begin
12         if(en)
13             out_lock2<=0;
14         else
15             out_lock2<=out_lock2;
16     end
17     else
18     begin
19         if(en)
20             out_lock2<=in_data;
21         else
22             out_lock2<=out_lock2;
23     end
24 end
25
26 endmodule
27

```

```

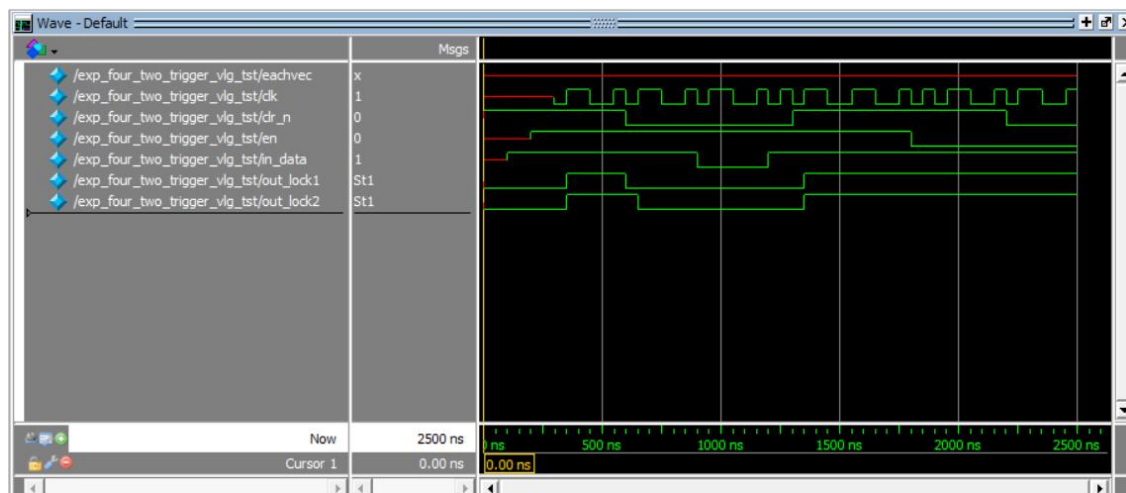
exp_four_two_trigger.v  mysynchro.v  myasynchro.v  Compilation Report - exp_fou
1  module myasynchro(in_data,en,clk,out_lock1,clr_n);
2  input in_data,en,clk,clr_n;
3  output reg out_lock1;
4
5  initial out_lock1=0;
6  always @(posedge clk)
7  begin
8      if(en && !clr_n)
9          out_lock1<=0;
10     else
11     begin
12         if(en)
13             out_lock1<=in_data;
14         else
15             out_lock1<=out_lock1;
16     end
17 end
18
19 endmodule
20
21

```

激励代码：

```
exp_four_two_trigger.v  mysynchro.v  myasynchro.v  Compilation Report - exp_four_two_trigger  exp_four_two_trigger.vt*
52 begin
53 // code that executes only once
54 // insert code here --> begin
55
56 clk=0;
57 in_data=1;
58 en=0;
59 clr_n=1; #5;
60 clr_n=0; #5;
61 clr_n=1; #5;
62 clr_n=0; #5;
63 en=1;
64 clr_n=1; #5;
65 clr_n=0; #5;
66 clr_n=1; #5;
67 clr_n=0; #5;
68
69 in_data=0;
70 en=0;
71 clr_n=1; #5;
72 clr_n=0; #5;
73 clr_n=1; #5;
74 clr_n=0; #5;
75 en=1;
76 clr_n=1; #5;
77 clr_n=0; #5;
78 clr_n=1; #5;
79 clr_n=0; #5;
80
81 $stop;
82 // --> end
83 // $display("Running testbench");
```

ModelSim 仿真波形：



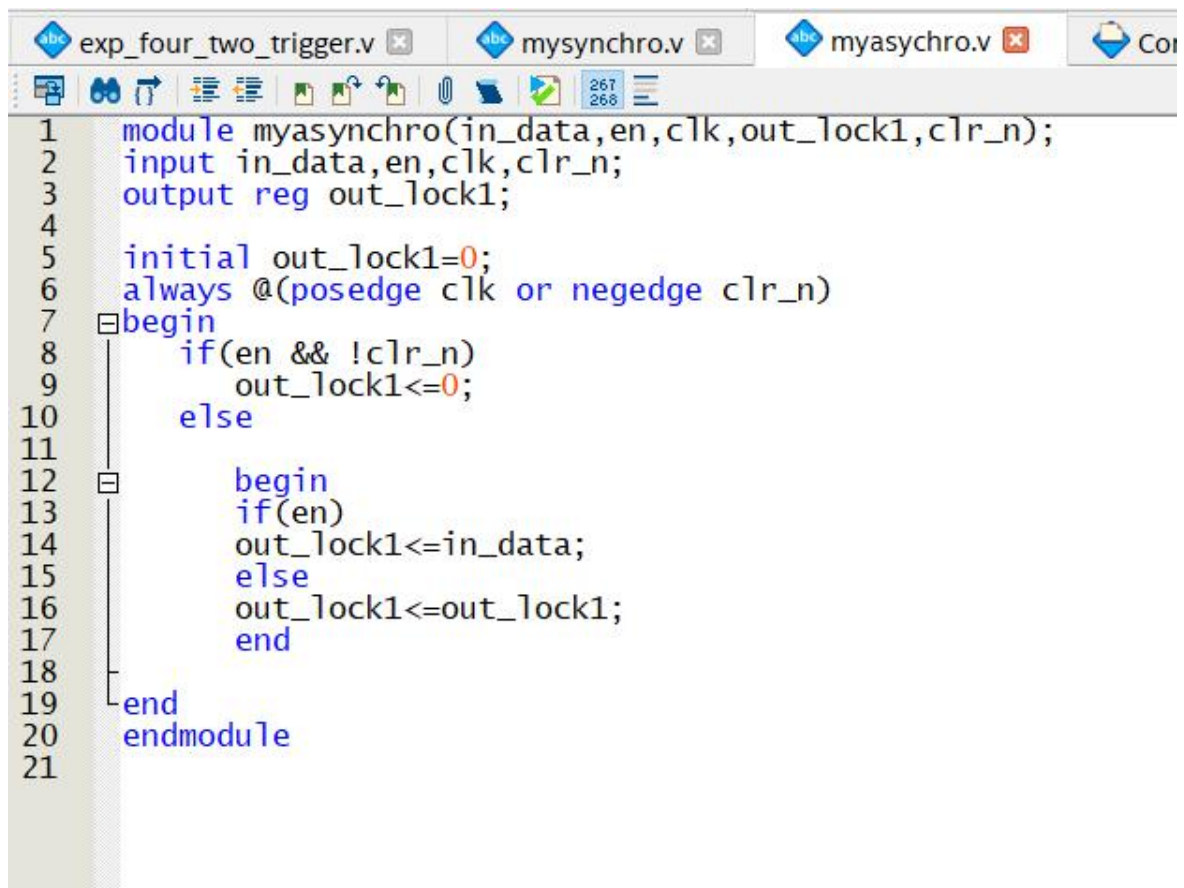
仿真结果和预期一样。

## 引脚分配:

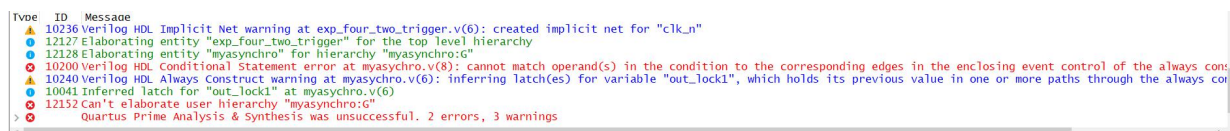
Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	ifft
in clk	Input	PIN_AJ4	3B	B3B_NO	2.5 V ...fault)		12mA ...ault)		
in clr_n	Input	PIN_AB30	5B	B5B_NO	2.5 V ...fault)		12mA ...ault)		
in en	Input	PIN_AA30	5B	B5B_NO	2.5 V ...fault)		12mA ...ault)		
in in_data	Input	PIN_Y27	5B	B5B_NO	2.5 V ...fault)		12mA ...ault)		
out out_lock1	Output	PIN_AA24	5A	B5A_NO	2.5 V ...fault)		12mA ...ault)	1 (default)	
out out_lock2	Output	PIN_AB23	5A	B5A_NO	2.5 V ...fault)		12mA ...ault)	1 (default)	
<<new node>>									

SW[9]:en  
SW[1]:in\_data  
SW[0]:clr\_n  
KEY[0]:clk  
LED[1]:out\_lock2, 表示同步计数器输出  
LED[0]:out\_lock1, 表示异步计数器输出

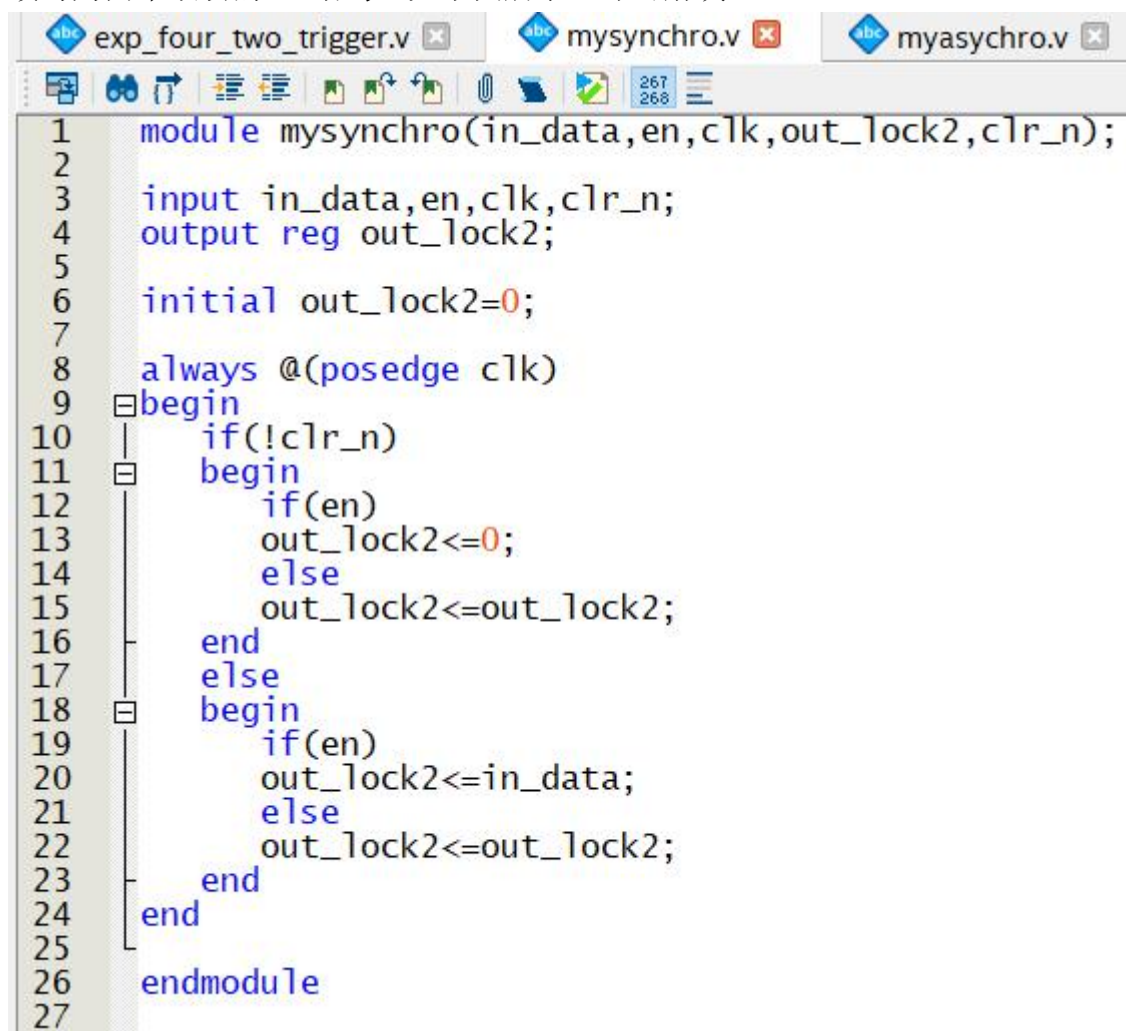
## 五：实验过程中遇到的问题及解决



```
1 module myasynchro(in_data,en,clk,out_lock1,clr_n);
2   input in_data,en,clk,clr_n;
3   output reg out_lock1;
4
5   initial out_lock1=0;
6   always @(posedge clk or negedge clr_n)
7   begin
8       if(en && !clr_n)
9           out_lock1<=0;
10      else
11
12          begin
13              if(en)
14                  out_lock1<=in_data;
15              else
16                  out_lock1<=out_lock1;
17          end
18      end
19  endmodule
20
21
```



在写异步清零计数端的时候在 if 语句中判断条件加入 clk 和 clr\_n 时编译报错。经筛查发现是 if 语句内不能把使能端和清零端同时作为判断条件，将该 if 语句改写为两个嵌套的 if 语句（如下图所示）即可解决。



## 六、实验得到的启示

与同学讨论发现，如果 always 语句内的敏感条件为（posedge clk or negedge clr\_n），则在 always 内的第一个 if 语句须以 !clr\_n 为判断条件，不能先以 en 为判断条件。

## 七、意见与建议

实验手册中的实例对学习 verilog 语句使用与 quartus 相关操作

有巨大帮助，希望能在学习新知识的同时接触更多相关例子来加深对新知的理解与应用能力。

看实验手册时对模块实例化方法不太理解，在网上查了好久才查明白。希望自己能加强对网上知识的搜索自学能力，这样可以迅速掌握新的知识。