

数电实验EXP4

刘永鹏 191220070 计科 693901492@qq.com 2020.9.12

1. 实验目的

1. 请你建立两个工程，分别研究**阻塞赋值**和**非阻塞赋值**的 RTL 级视图和仿真 结果，步骤如下：
 1. 新建工程，用阻塞赋值语句设计两个触发器；保存 Verilog 语言文件。
 2. 在Tools 栏，点击Netlist Viewers 栏下的RTL Viewer 查看生成的RTL Schematic，看看在用阻塞赋值语句生成两个触发器的实际电路原理。
 3. 新建另一个工程，用非阻塞赋值语句实现两个触发器，重复上述步骤，比 较两种触发器实现方式在硬件电路实现上的异同。
2. 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。

2. 实验原理

1. **阻塞赋值与非阻塞赋值**：前者使用“=”，在执行完这一步赋值前，**不允许**后续代码执行，即程序阻塞。而后者使用“<=”，在一个always模块中，所有<=语句在模块结束时**同步执行**。
2. 上述的赋值方法，在逻辑电路中没有差别，而在**时序电路**中可能会导致不同，通常表现为两种不同赋值方法**结果相差一个时钟周期**。
3. D触发器在时钟正边沿触发，遵循激励方程：

```
1 | D_VAL = D_DATA
```

4. **同步清零和异步清零**：区别在于：同步清零端在生效后需等待下一个时钟上升沿才能真正清零，而异步是即刻清零。

3. 实验环境/器材

1. Quartus: 这次没有使用systembuilder, 而是自己设置引脚。
2. 使用开关代替时钟（开发板的时钟频率过高）。
3. 对于4-1，给出其RTL视图，采用EXP4_1顶层文件，调用block.v和unblock.v，分别查看阻塞赋值和非阻塞赋值的效果。

4. 程序代码

实验4_1

```

1 //顶层实体
2 module EXP4_1(
3     input clk,
4     input in_data,
5     input en,
6     output out_u1,
7     output out_u2,
8     output out_1,
9     output out_2
10 );
11 Block b1(.clk(clk), .in_data(in_data), .en(en), .out_1(out_1),
12         .out_2(out_2));
13 Unblock ub1(.clk(clk), .in_data(in_data), .en(en), .out_1(out_u1),
14             .out_2(out_u2));
15 endmodule

```

```

1 //非阻塞赋值模块
2 module Unblock(
3     input clk,
4     input in_data,
5     input en,
6     output reg out_1,
7     output reg out_2
8 );
9 always @ (posedge clk )
10
11 if(en)
12     begin
13         out_1 <= in_data;
14         out_2 <= out_1;
15     end
16 else
17     begin
18         out_1 <= out_1;
19         out_2 <= out_2;
20     end
21
22 endmodule
23
24 //阻塞赋值模块
25 module Block(
26     input clk,
27     input in_data,
28     input en,
29     output reg out_1,
30     output reg out_2
31 );
32 always @ (posedge clk )
33 if(en)
34     begin
35         out_1 = in_data;
36         out_2 = out_1;
37     end
38 else
39     begin
40         out_1 = out_1;

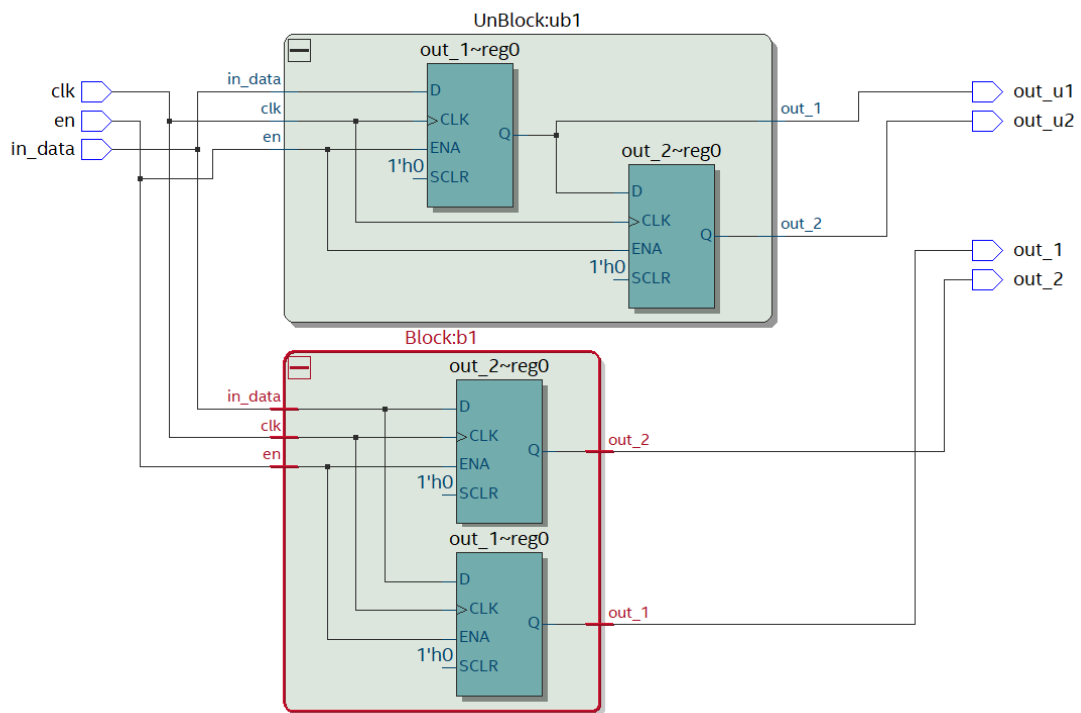
```

```

41         out_2 = out_2;
42     end
43 endmodule

```

RTL视图



实验4_2

```

1  //顶层实体
2  module D_trigger(
3      input en,
4      input clk,//SW[3]
5      input clr,
6      input in_data,
7      output out_sy,
8      output out_asy //清零: SW[2],使能, SW[1], 数据SW[0]
9  );
10
11  asyclr asy(.en(en), .clk(clk), .clr(clr), .in_data(in_data),
12             .out_asy(out_asy));
13  syclr sy(.en(en), .clk(clk), .clr(clr), .in_data(in_data),
14           .out_sy(out_sy));
15
16 endmodule

```

```

1  //异步清零
2  module asyclr(
3      input en,
4      input clk,
5      input clr,
6      input in_data,
7      output reg out_asy
8  );

```

```

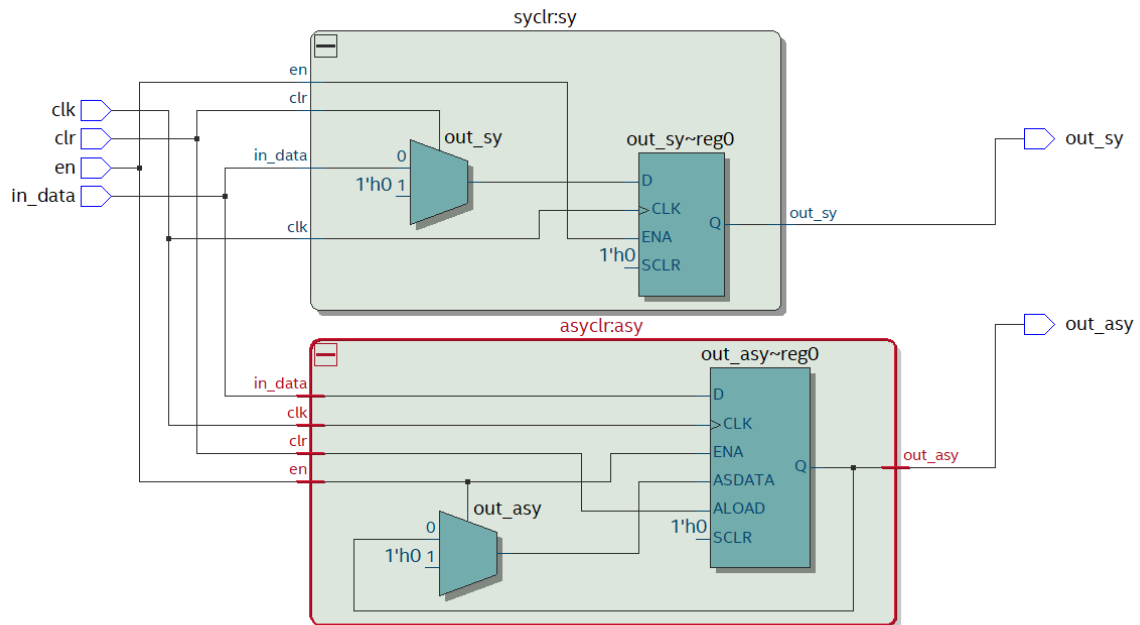
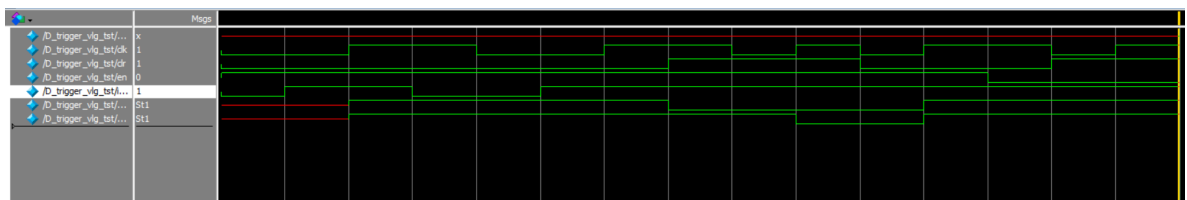
9  always @ (posedge clk or posedge clr)
10 begin
11     if(clr)
12         if(en)
13             out_asy <= 0;
14         else out_asy <= out_asy;
15     else
16     begin
17         if(en)
18             out_asy <= in_data;
19         else out_asy <= out_asy;
20     end
21 end
22
23 endmodule
24 //同步清零
25 module sysclr(
26     input en,
27     input clk,
28     input clr,
29     input in_data,
30     output reg out_sy
31 );
32 always @ (posedge clk)
33 begin
34     if(en)
35         if(clr)
36             out_sy <= 0;
37         else out_sy <= in_data;
38     else out_sy <= out_sy;
39 end
40
41 endmodule

```

```

1  //testbench
2  begin
3      // code executes for every event on sensitivity list
4      // insert code here --> begin
5      clk = 0; en = 1; clr = 0; in_data = 0; #10;
6      in_data = 1; #10;
7      clk = 1; in_data = 1; #10;
8      in_data = 0; #10;
9      clk = 0; #10;
10     in_data = 1; #10;
11     clk = 1; #10;
12     clr = 1; #10;
13     clk = 0; in_data = 1; #10;
14     clk = 1; #10;
15     clk = 0; clr = 0; #10;
16     clk = 1; in_data = 1; #10;
17     en = 0;
18     clk = 1; in_data = 1; #10;
19     clk = 0; in_data = 1; clr = 1; #10;
20     clk = 1; in_data = 1; #10;
21     @eachvec;
22     // --> end
23 end

```



5. 实验结果

结果符合预期，已验收，

6. 问题与解决方案/启示

1. 这期作业确实没什么问题，花了一个小时就写完了。
2. 大概要掌握多模块设计的技巧，这在后面的大项目中很有必要！

2. 大概要掌握多模块设计的技巧，这在后面的大项目中很有必要！

###