实验序号及名称： 实验 一 基本逻辑门和组合电路的设计

一．实验目的

1. 掌握 Verilog 编程设计电路，熟悉 ModelSim 仿真工具。

2. 掌握基本逻辑门和加法器的功能。

二．实验工具

ModelSim

三．实验内容

1. 用 Verilog 完成三态门的设计，并用 ModelSim 进行仿真。
2. 用 Verilog 完成 4 位加法器的设计，并使用 ModelSim 进行仿真。

四．实验过程（包括源程序及仿真结果截图）

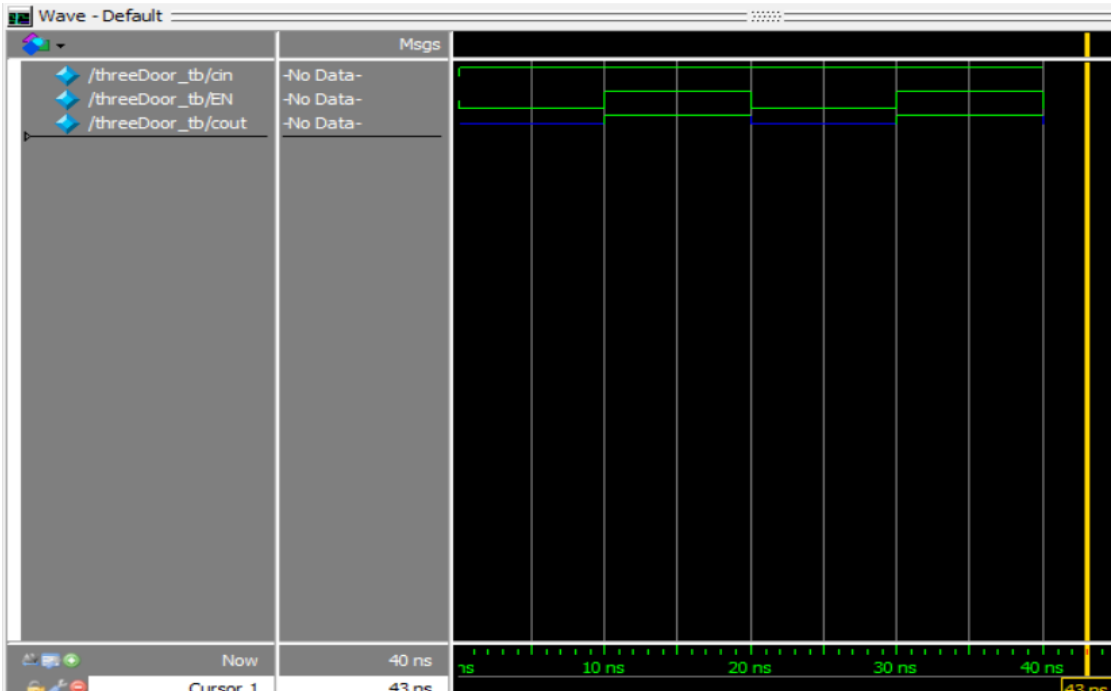
三态门源码：

```
module threeDoor(
cin,
cout,
EN);
input cin;
input EN;
output cout;
reg cout;
always @(cin,EN)
begin
    if(EN)cout=cin;
    else cout = 'bz;
end
endmodule
```

Testbench

```
module threeDoor_tb;

reg cin,EN;
wire cout;

initial begin
cin = 1'b1;
EN = 1'b0;
#10 EN = 1'b1;

#10 EN = 1'b0;

#10 EN = 1'b1;

#10 EN = 1'b0;

end
threeDoor uut(
.cin(cin), .cout(cout), .EN(EN)
);
endmodule
```

仿真结果：



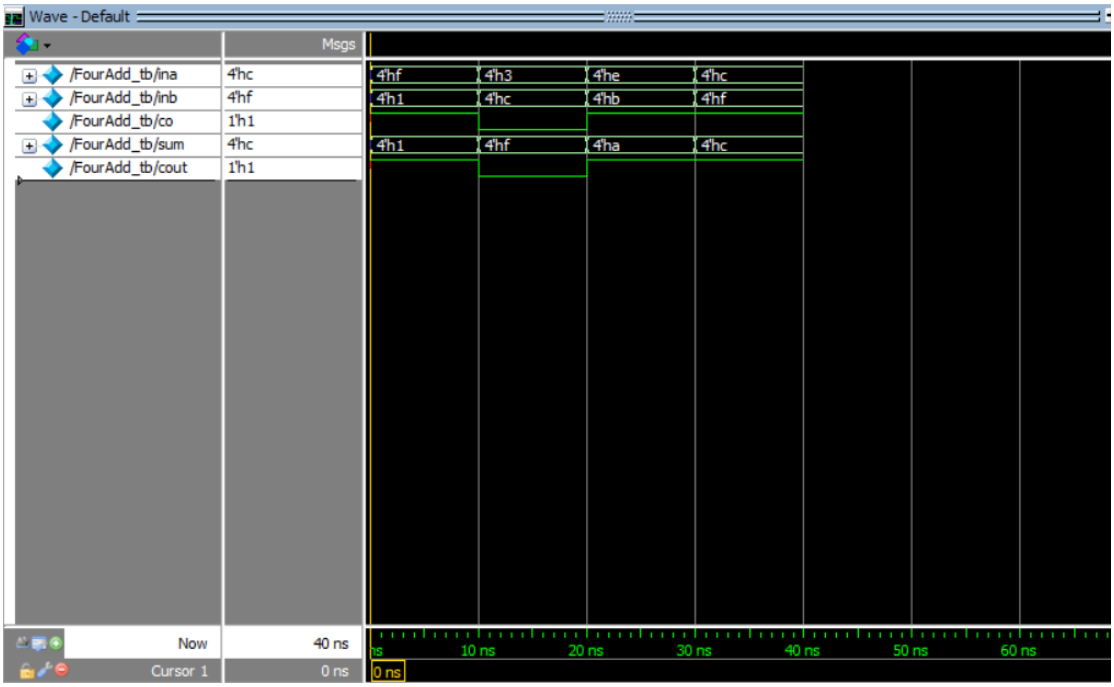
四位加法器源码：

```
module FourAdd(ina,inb,co,sum,cout);
    input[4:1]ina,inb;
    input co;
    output[4:1] sum;
    output cout;
    assign {cout,sum} = ina+inb+co;
endmodule
```

Testbench

```
module FourAdd_tb;
reg[4:1]ina,inb;
reg co;
wire[4:1] sum;
wire cout;
initial begin
ina = 4'b1111;
inb = 4'b0001;
co = 1;
#10;

ina = 4'b0011;
inb = 4'b1100;
co = 0;
#10;
ina = 4'b1110;
inb = 4'b1011;
co = 1;
#10;
ina = 4'b1100;
inb = 4'b1111;
co = 1;

#10 $stop;
end

FourAdd uut(
.ina(ina),.inb(inb),.co(co),.sum(sum),.cout(cout)
);

endmodule
```

仿真结果：



四．实验感想、体会

第一次使用 Verilog 语言，花了一个下午，看了老师发的华为教程还有一个 ppt 教程，然后取网上找了一个两小时快速入门 Verilog 的视频教程看，看完之后，对 Verilog 语言整体印象觉得还是挺简单的，就是对一些细节还是不清楚，看了 ppt 上的几个例子和老师的给的 mux 的例子后，掌握了 Verilog 的基本用法，所以写起来还是挺轻松的。但是 testbench 还是不太会写，这一个的 testbench 看了好多资料，感觉不太好写，最后还是参考老师那个 mux 的 testbench 来写的，出了很多次错后发现，testbench 的输入都是 reg 类型的，不是 wire 类型的，然后就完成了第一个实验，总体来说学的比较久，但是写起来觉得还是很简单。

实验序号及名称： 实验 二 组合电路的设计

一．实验目的

1. 掌握 Verilog 编程设计电路，熟悉 ModelSim 仿真工具。

2. 掌握译码器、数据比较器的功能。

二．实验工具

ModelSim

三．实验内容

1. 用 Verilog 完成 3-8 译码器的设计，并用 ModelSim 进行仿真。

2. 用 Verilog 完成两个 4 位二进制数据比较器的设计，并使用 ModelSim 进行仿真。

四．实验过程（包括源程序及仿真结果截图）

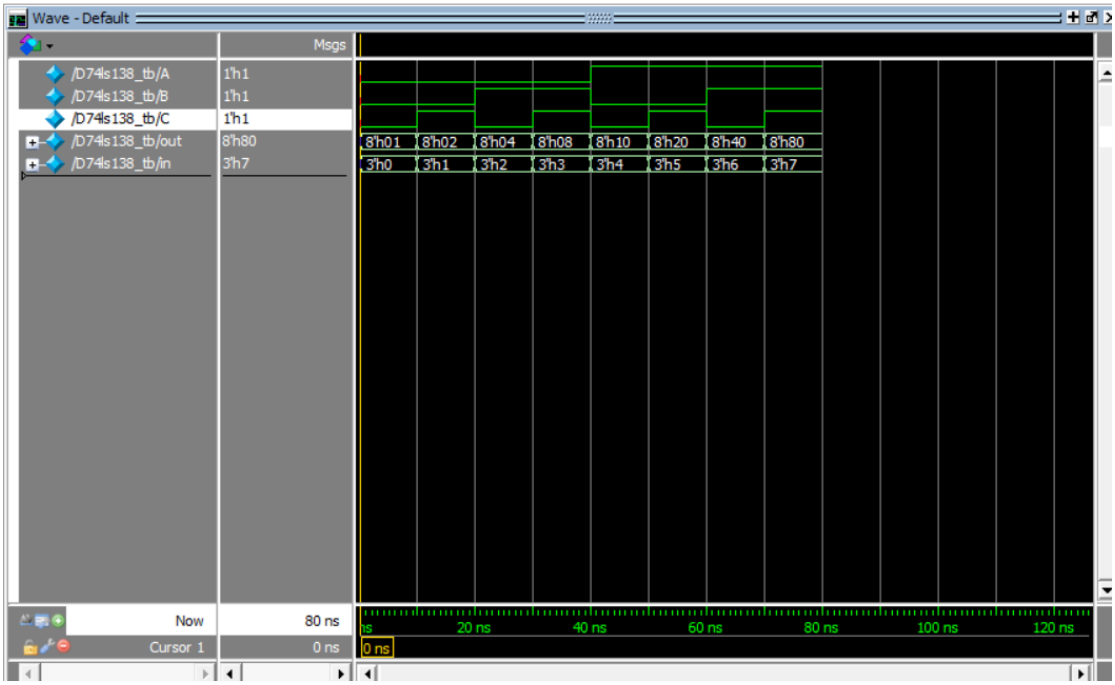
三八译码器源码：

```
module D74ls138(
    input wire[2:0] in,
    output reg[7:0] out
);
    integer i;
    always@(*)
        begin
            for(i = 0;i<8;i=i+1)
                if(i==in)out[i]<=1;
                else out[i]<=0;
        end
endmodule
```

Testbench：

```
module D74ls138_tb;
    reg A,B,C;
    wire [7:0] out;
    wire [2:0] in;
    assign in={A,B,C};
    D74ls138 uut(.in(in),.out(out));
    initial begin
        A=0;B=0;C=0;#10;
        A=0;B=0;C=1;#10;
        A=0;B=1;C=0;#10;
        A=0;B=1;C=1;#10;
        A=1;B=0;C=0;#10;
        A=1;B=0;C=1;#10;
        A=1;B=1;C=0;#10;
        A=1;B=1;C=1;#10;
    end
endmodule
```

仿真结果：



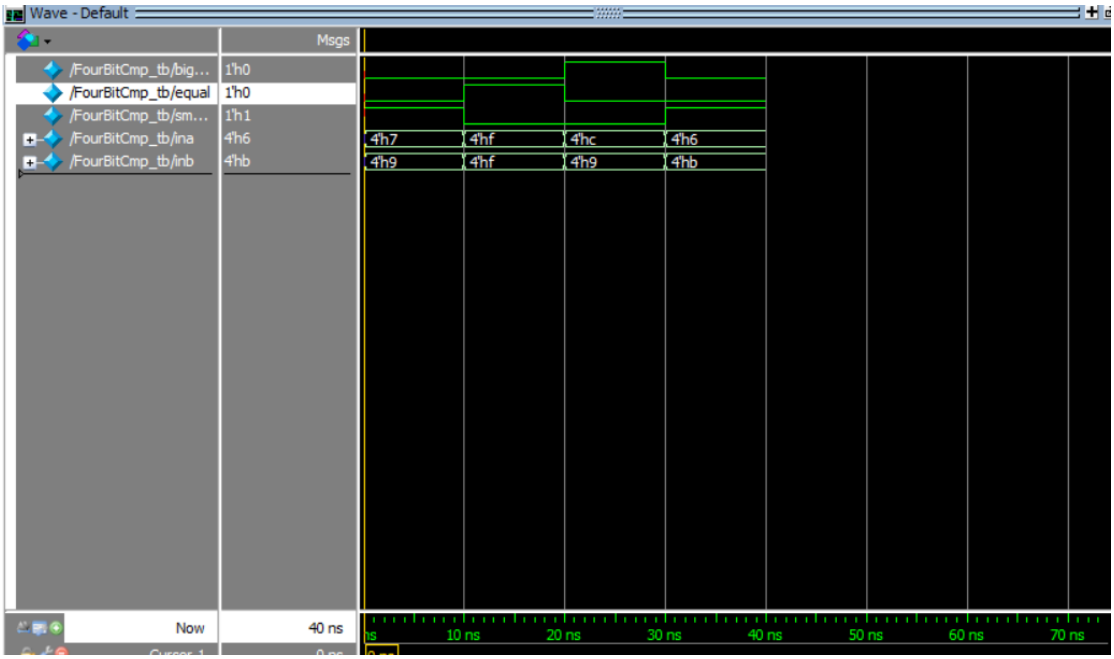
四位比较器源码：

```
module FourBitCmp(bigger,equal,smaller,ina,inb);
    parameter n=4;
    output bigger,equal,smaller;
    input[n:1] ina,inb;
    assign bigger=(ina>inb);
    assign equal=(ina==inb);
    assign  smaller=(ina<inb);
endmodule
```

Testbench:

```
module FourBitCmp_tb;

wire bigger,equal,smaller;
reg[4:1]ina,inb;

FourBitCmp uut(
.bigger(bigger),.equal(equal),.smaller(smaller),.ina(ina),.inb(inb)
);

initial begin

    ina = 4'b0111;
    inb = 4'b1001;
    #10;

    ina = 4'b1111;
    inb = 4'b1111;
    #10;

    ina = 4'b1100;
    inb = 4'b1001;
    #10;

    ina = 4'b0110;
    inb = 4'b1011;

    #10 $stop;


end

endmodule
```

仿真结果：



四．实验感想、体会

四位比较器的实现一开始觉得要一位一位进行比较，后来看了许多例子后发现直接将四位数字当作整体，当成一个数字就可以直接进行比较，非常简单。

三八译码器最初是想用 case 语句来列举所以的情况，但是如果位数多了以后，四位，五位，就需要很多行代码，但由于 Verilog 能直接将二进制的一位一位的数据与十进制整数进行比较，所以就能用一个 for 循环语句，将选中的输出位设为一，就能很简便的输出译码结果。但是对于 testbench 还是列举了八个信号，因为对于 testbench 还不熟悉，所以没有写简便一点的方法。

实验序号及名称： 实验 三 触发器

一．实验目的

1. 掌握 Verilog 编程设计电路，熟悉 ModelSim 仿真工具。

2. 掌握 D 触发器、JK 的功能。

二．实验工具

ModelSim

三．实验内容

1. 用 Verilog 完成具有异步清零和异步置 1 的 D 触发器的设计，并用 ModelSim 进行仿真。
2. 用 Verilog 完成具有同步清零和同步置 1 的 JK 触发器的设计，使用 ModelSim 进行仿真。

四．实验过程（包括源程序及仿真结果截图）

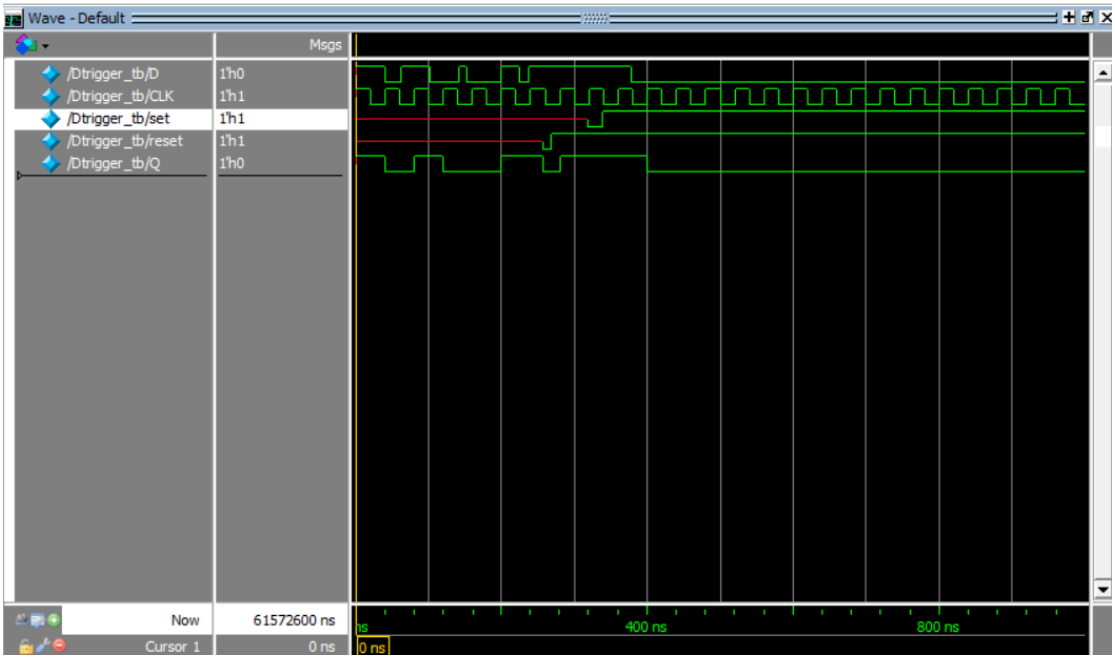
D 触发器源码（上升沿触发）：

```
module Dtrigger(D,CLK,Q,reset,set);
    input D,CLK,set,reset;
    output Q;
    reg Q;
    always@(posedge CLK or negedge reset or negedge set)
    begin
        if(!set)Q <= 1;
        else if(!reset)Q <= 0;
        else Q <= D;
    end
endmodule
```

Testbench:

```
module Dtrigger_tb;
reg D;
reg CLK;
reg set;
reg reset;
wire Q;
Dtrigger uut(
.D(D),.CLK(CLK),.set(set),.reset(reset),.Q(Q)
);
initial begin
CLK <= 1;
D<=1;
#40 D<=0;
#22 D<=1;
#40 D<=0;
#40 D<=1;
#10 D<=0;
#48 D<=1;
#24 D<=0;
#14 D<=1;
#20 reset<=0;
#10 reset<=1;
#30 D<=1;
#20 set<=0;
#20 set<=1;
#40 D<=0;
end
always
begin
#20 CLK<=~CLK;
end
endmodule
```

仿真结果：



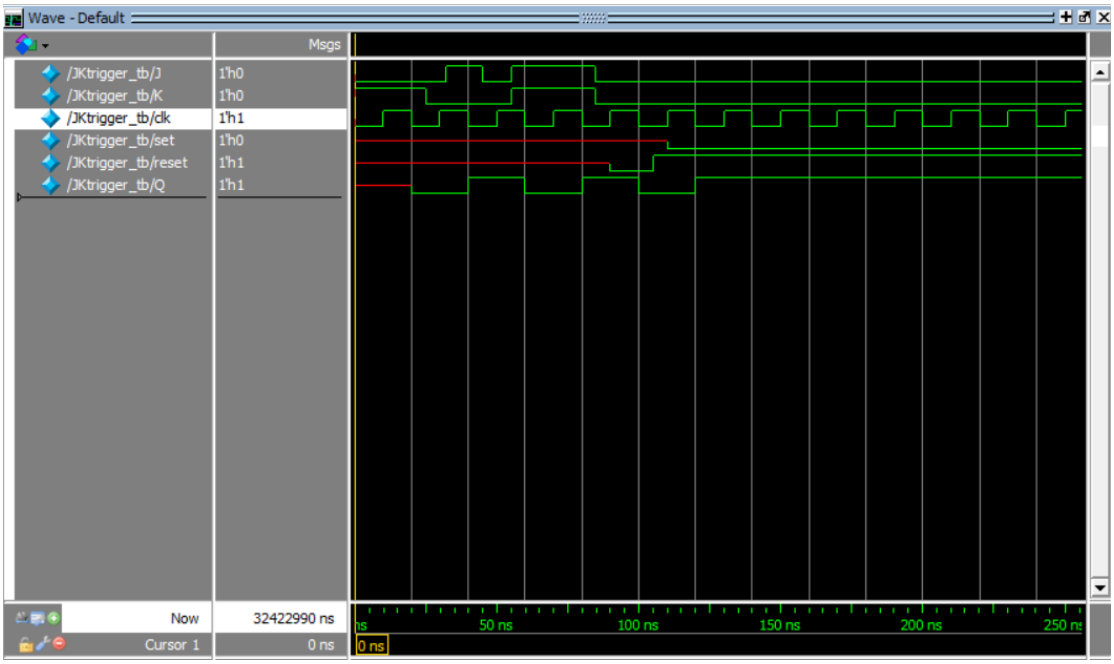
JK 触发器源码（下降沿触发）：

```
module JKtrigger(J,K,clk,Q,set,reset);
    input J,K,clk,set,reset;
    output Q;
    reg Q;
    always@(negedge clk)
    begin
        if(!set)Q <= 1;
        else if(!reset)Q <= 0;
        else
        begin
            case({J,K})
                2'b00:Q <= Q;
                2'b01:Q <= 0;
                2'b10:Q <= 1;
                default:Q<=~Q;
            endcase
        end
    end
endmodule
```

Testbench：

```
module JKtrigger_tb;
reg J,K,clk,set,reset;
wire Q;
JKtrigger uut(
.J(J),.K(K),.clk(clk),.Q(Q),.set(set),.reset(reset)
);
initial begin
clk = 0;
K<=1;
J<=0;
#25 K<=0;
#7 J<=1;
#13 J<=0;
#10;
K<=1;
J<=1;
#30;
K<=0;
J<=0;
#5 reset<=0;
#15 reset<=1;
#5 set <=0;
end

always #10 clk=~clk;
endmodule
```

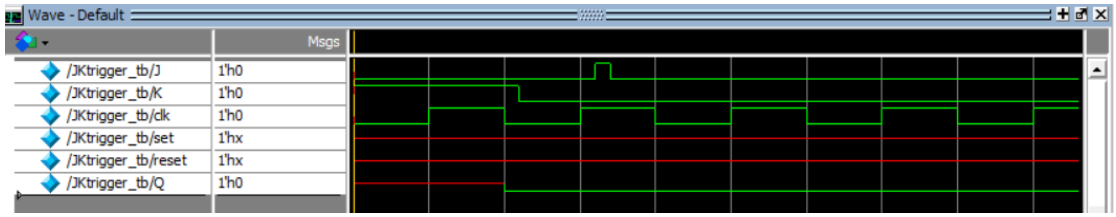
仿真结果：



四．实验感想、体会

D 触发器比较简单，用 **always** 接受三个敏感信号，然后对每个信号进行分析即可，但是由于 **if** 语句的先后顺序，在我的电路了会先检测 **set** 信号的变化，因为 **set** 信号在 **if** 语句里，最先执行。

JK 触发器我最先想到的是一次翻转现象，在 **cp=1** 期间，如果 **Qn=0**，若 **J** 变化，则 **Qn+1** 会变成 **1**，我进行了模拟仿真，发现进行仿真并不会出现一次变化，所以模拟仿真应该并不是真正把电路组成现实中的电路，也克服了像一次翻转这样的问题。



实验序号及名称： 实验 四 计数器

一．实验目的

1. 掌握 Verilog 编程设计电路，熟悉 ModelSim 仿真工具。

2. 掌握计数器的功能

二．实验工具

ModelSim

三．实验内容

1. 用 Verilog 完成具有异步清零和异步置 1 的模 10 计数器的设计，并用 ModelSim 进行仿真。
2. 用 Verilog 完成具有同步清零和同步置 1 的模 16 计数器的设计，使用 ModelSim 进行仿真。

四．实验过程（包括源程序及仿真结果截图）

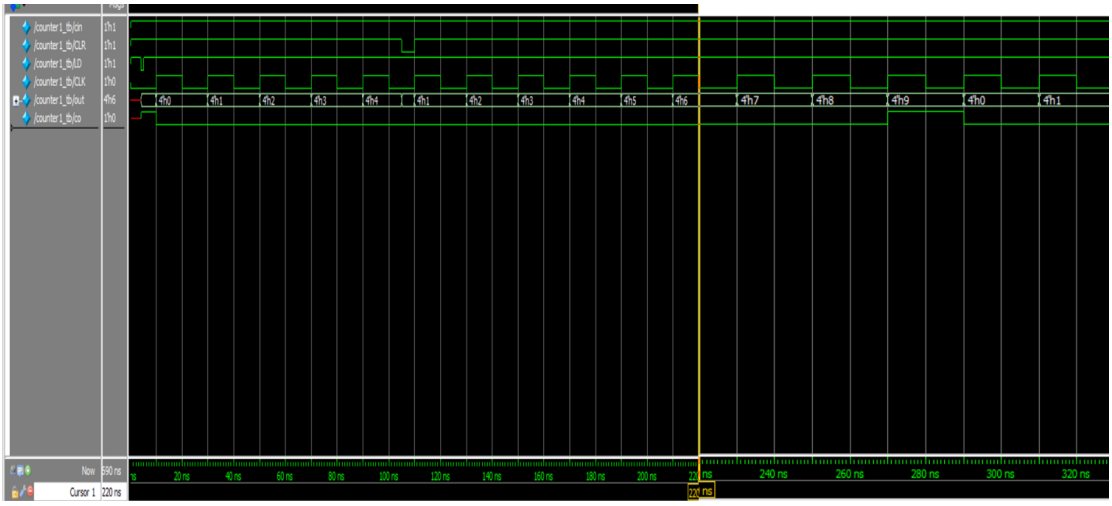
模 10 计数器源码：

```
module counter1(cin,CLR,LD,CLK,out,co);
    input cin,CLR,LD,CLK;
    output[3:0] out;
    output co;
    reg[3:0] out;

    always@(negedge CLR or posedge CLK or negedge LD)
    begin
        if(!LD)out<=4'b1111;
        else if(!CLR)out<=4'b0000;
        else if(cin)
        begin
            if(out==4'b1001)out<=4'b0000;
            else out=out+1'b1;
        end
        else out<=out;
    end

    assign co=out[3]&out[0];

endmodule
```

Testbench：

```
module counter1_tb;
reg cin,CLR,LD,CLK;
wire[3:0]out;
wire co;
counter1 uut(
.cin(cin),.CLR(CLR),.LD(LD),.CLK(CLK),.out(out),.co(co)
);

initial begin

CLK<=0;
cin<=1;
LD<=1;
CLR<=1;
#4 LD<=0;
#1 LD<=1;
#100 CLR = 0;
#5 CLR = 1;
end

always #10 CLK<=~CLK;

endmodule
```

仿真结果：



说明：

由于如果输出的计数结果 out 不初始化，就无法进行计数，所以在第一个上升沿前，将 LD 置为 0，异步把 out 置为 4'hf，也就是四个触发器都置为 1，然后在 4'h4 后异步置为 0（**图中 110nm 左右**）。随后计了十个数，在 4'h9 看到 co 变为一，下个周期 out 变回了 0，co 也变回 0（**图中 290nm 左右**）。

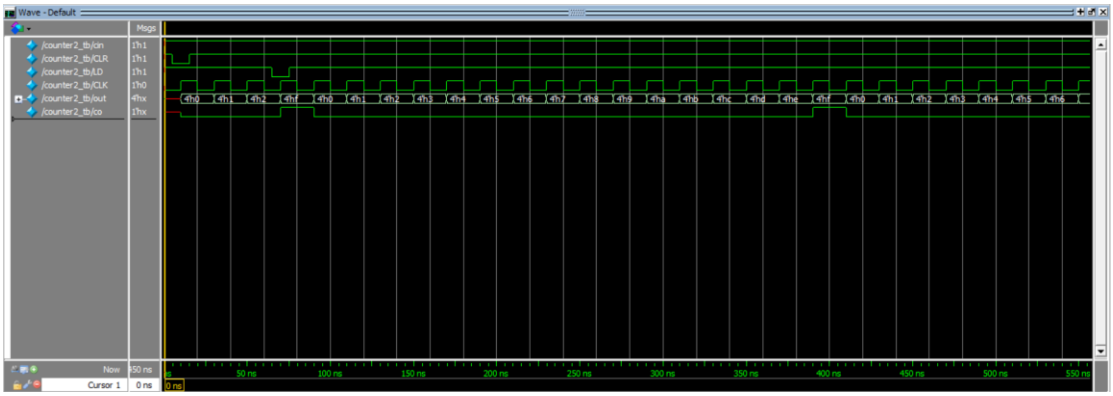
模十六计数器源码：

```
module counter2(cin,CLR,LD,CLK,out,co);
    input cin,CLR,LD,CLK;
    output[3:0] out;
    output co;
    reg[3:0] out;

    always@(posedge CLK)
    begin
        if(!CLR)out<=4'b0000;
        else if(!LD)out<=4'b1111;
        else if(cin)
            if(out==4'b1111)out<=4'b0000;
            else out<=out+1'b1;
        else out<=out;

    end

    assign co=out[0]&out[1]&out[2]&out[3];

endmodule
```

Testbench：

```
module counter2_tb;
reg cin,CLR,LD,CLK;
wire[3:0]out;
wire co;
counter2 uut(
.cin(cin),.CLR(CLR),.LD(LD),.CLK(CLK),.out(out),.co(co)
);

initial begin

CLK<=0;
cin<=1;
CLR<=1;
LD<=1;

#5 CLR=0;
#10 CLR=1;
#50 LD=0;
#10 LD=1;


end

always #10 CLK=~CLK;

endmodule;
```

仿真结果：



说明：在第一个时钟上升沿前，将 CLR 置为 0，在时钟上升沿时，输出 out 被初始化为 0，随后开始计数，cin 一直有效，也就是每个上升沿都计数，在 4’h2 后将 LD 置为 0（**图中 75nm 左右**），也就是四个触发器的输出值都置位 1，out 在下一个上升沿时就被置为了 4’hf，随后开始正常计数，在 4’hf 时进位 co 被置为 1（**图中 400ns 左右**），下一个周期输出 out 重新回到 0，co 回到 0。

四．实验感想、体会

在看实验要求的时候，不知道异步置 1 和同步置 1 是什么意思，问了老师才知道是把每个触发器的输出 Q 都置为 1，也就是把输出值置为 1111。通过这个实验也让我对计数器加深了印象，理解了细节。一开始不知道怎么做，以为需要通过置数和清零来实现模十和模十六，后来仔细想了一下才反应过来，这个清零端和置 1 端是为了让用该计数器组成其他模值的计数器才用的到的。所以只需要实现一个清零端，和一个置 1 端就好了，所以实现还是挺简单的。

通过这四个实验，我觉得用计算机语言来描述数字电路更方便，如果是用实际的物件来实现一个计数器，要列出状态转移方程，还需要编码，连接多个触发器才能实现一个计数器，而通过电脑模拟，就只需要写出电路的逻辑，就能用计算机自动模拟，非常方便。