实验三 芯片的封装及应用

一、实验目的

1. 掌握Verilog语言框架，编程及调试的方法；

2. 熟悉Verilog的基本语法；

3. 掌握Vivado开发平台及FPGA开发板的使用。

二、实验内容

1. 完成74LS161计数器芯片的实现、测试及6进制计数器的实现，将程序下载到FPGA开发板进行验证；

2. 分析电路中的竞争与冒险，给出解决方案并验证；

3. 将芯片及相关模块封装为IP核，通过原理图设计实现10进制计数器，观察、分析仿真波形和电路图；

三、实验要求

1. 画出模块的电路图；

2. 分析电路的仿真波形和板卡验证结果；

3. 记录设计和调试过程。

四、实验过程及结果分析

**对实验代码、仿真结果等进行分析说明，不能只贴截图。**

**3.3.1 74LS161芯片**

1.74LS161芯片计数器

设计文件：

module Counter\_74LS161(

input CR\_n, CP, D0, D1, D2, D3, LD\_n, EP, ET,

output Q0, Q1, Q2, Q3);

wire [3:0] Data\_in;

reg [3:0] Data\_out;

assign Data\_in = {D3, D2, D1, D0};

always @(posedge CP or negedge CR\_n) begin

if(CR\_n == 0) Data\_out <= 0;

else if(LD\_n == 0) Data\_out <= Data\_in;

else if(LD\_n == 1 && EP == 0 && ET == 0) Data\_out < = Data\_out;

else if(LD\_n == 1 && EP == 0 && ET == 1) Data\_out < = Data\_out;

else if(LD\_n == 1 && EP == 1 && ET == 0) Data\_out < = Data\_out;

else if(LD\_n == 1 && EP == 1 && ET == 1) Data\_out < = Data\_out + 1;

end

assign Q0 = Data\_out[0];

assign Q1 = Data\_out[1];

assign Q2 = Data\_out[2];

assign Q3 = Data\_out[3];

endmodule

仿真文件：

module sim\_74ls161;

reg CP, D\_0, D\_1;

wire CR\_n;

initial begin

CP = 1; D\_0 = 0; D\_1 = 0;

#30 D\_1 = 1;

end

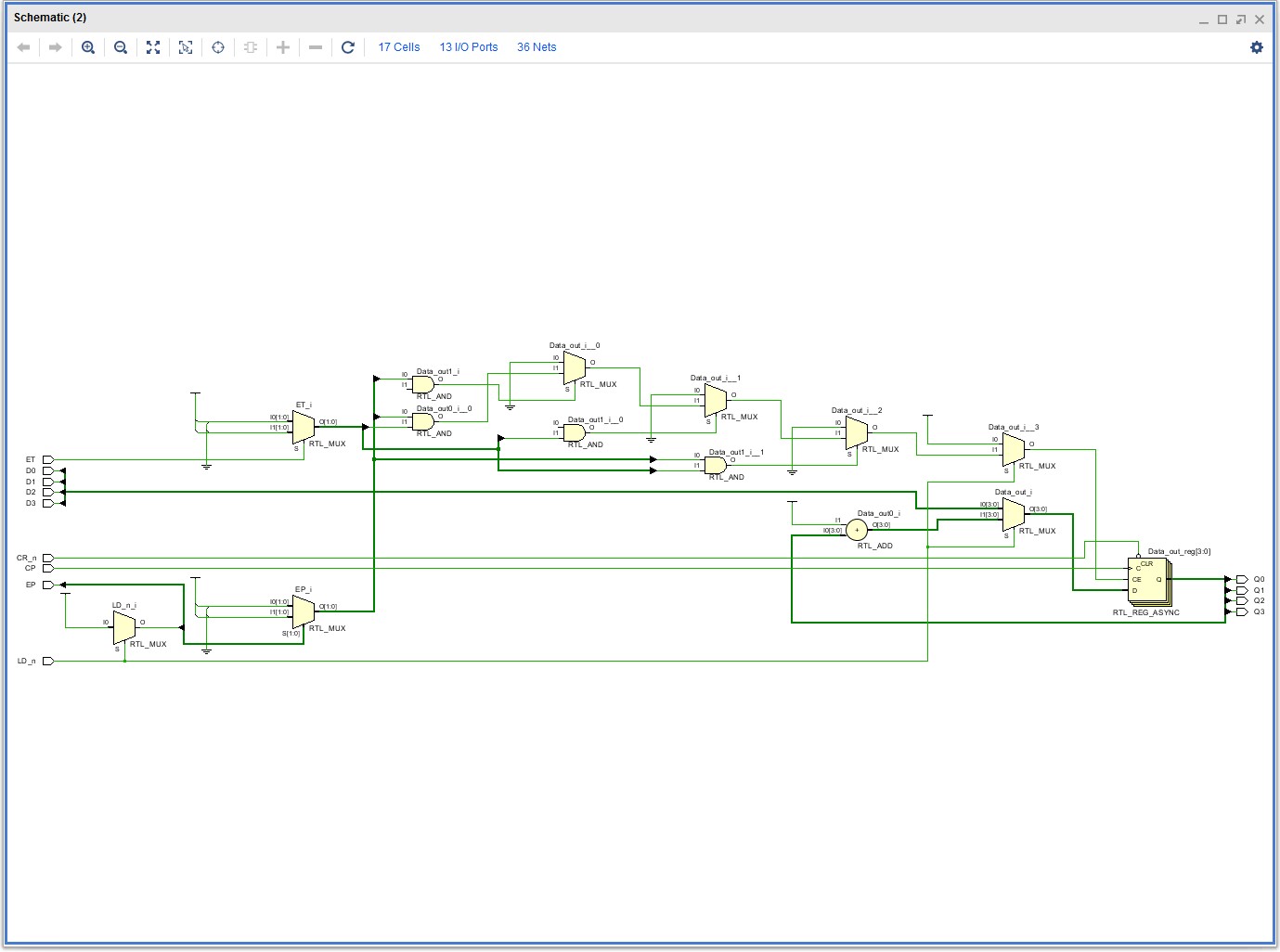
always #10 CP = ~CP;

assign CR\_n = ~(Q1 & Q2);

Counter\_74LS161(CR\_n, CP, D\_0, D\_0, D\_0, D\_0, D\_1, D\_1, D\_1, Q0, Q1, Q2, Q3);

endmodule

RTL仿真：



波形仿真图：



代码分析：

Design模块：

定义名为Counter\_74LS161模块

声明输入端口CR\_n,CP,D0,D1,D2,D3,LD\_n,EP,ET

声明输出端口Q0,Q1,Q2,Q3

声明内部Data\_in线，宽度为4，表示输入数据，由D3,D2,D1,D0组成

声明Data\_out寄存器，宽度为4，表示输出数据

使用always语句，上升沿或者CR\_n的下沿触发，根据不同条件对Data\_out进行赋值，赋值规则如下：

CR\_n为低，Data\_out为0

LD\_n为低，Data\_in数据赋值给Data\_out

否则根据EP和ET的组合，进行操作

使用assign将Data\_out赋给Q0,Q1,Q2,Q3

Sim模块

定义名为sim\_74ls161的模块

声明寄存器变量CP，D\_0,D\_1,控制计数器输入信号

声明线变量CR\_n,连接计数器清零信号

Initial对CP,D\_0,D\_1初始化

Always语句每10个时间单位翻转CP

Assign对Q1,Q2取反，并赋值给CR\_n

最后实例化Counter\_74LS161模块，对输入输出信号进行连接。

这段代码实现了一个74LS161型计数器的行为。它根据输入信号 CP 控制计数器的时钟，通过输入信号 D0、D1 控制计数器的加载数据，根据输入信号 EP 和 ET 控制计数器的操作模式。输出信号 Q0、Q1、Q2 和 Q3 表示计数器的当前值。

2、时钟分频模块

代码：

module clk\_div(input clk\_100M,//输入时钟信号output clk\_100,clk\_50,clk\_20,clk\_10, clk\_5, clk\_1);//输出时钟信号reg [15:0] cnt\_khz;reg [11:0] cnt\_hz;reg clk\_1k, clk\_100\_reg ,clk\_50\_reg ,clk\_20\_reg, clk\_10\_reg, clk\_5\_reg ,clk\_1\_reg;initial begincnt\_khz =0; cnt\_hz = 0; clk\_1k= 0;clk\_100\_reg =0; clk\_50\_reg =0; clk\_20\_reg = 0; clk\_10\_reg = 0; clk\_5\_reg=0;clk\_1\_reg=0;endalways @(posedge clk\_100M) begincnt\_khz <= cnt\_khz + 1;if(cnt\_khz == 16'hC34F)clk\_1k <=~clk\_1k;endalways @(posedge clk\_1k) begincnt\_hz <= cnt\_hz + 1;endalways @(posedge clk\_1k) beginif(cnt\_hz[2:0] ==3'b100) clk\_100\_reg <= ~clk\_100\_reg;else if(cnt\_hz[3:0] == 4'b1001) clk\_50\_reg <= ~clk\_50\_reg;else if(cnt\_hz[4:0]==5'b11000) clk\_20\_reg <=~clk\_20\_reg;else if(cnt\_hz[5:0] ==6'b110001) clk\_10\_reg <=~clk\_10\_reg;else if(cnt\_hz[6:0] == 7'b1100011) clk\_5\_reg <= ~clk\_5\_reg;else if(cnt\_hz[8:0]== 9'b11\_1110\_011) clk\_1\_reg <=~clk\_1\_reg;endassign clk\_100 = clk\_100\_reg;assign clk\_50 = clk\_50\_reg;assign clk\_20 = clk\_20\_reg;assign clk\_10 = clk\_10\_reg;assign clk\_5 = clk\_5\_reg;assign clk\_1 = clk\_1\_reg;endmodule

模块定义了输入和输出端口：

clk\_100M 是输入的主时钟信号

clk\_100、clk\_50、clk\_20、clk\_10、clk\_5 和 clk\_1 是分频后的输出时钟信号

声明了一些寄存器变量和计数器变量来进行时钟分频计数

initial 初始化所有的寄存器和计数器变量

第一个 always 块使用输入主时钟信号 clk\_100M 的上升沿作为触发条件。在每个主时钟周期中，cnt\_khz 计数器自增 1。当 cnt\_khz 达到阈值 16'hC34F 时，clk\_1k 取反。

第二个 always 块使用 clk\_1k 的上升沿作为触发条件。在每个 clk\_1k 周期中，cnt\_hz 计数器自增 1

第三个 always 块也使用 clk\_1k 的上升沿作为触发条件。根据 cnt\_hz 的不同值，选择性地将分频后的时钟信号 clk\_100\_reg、clk\_50\_reg、clk\_20\_reg、clk\_10\_reg、clk\_5\_reg 和 clk\_1\_reg 取反

使用 assign 语句将分频后的时钟信号连接到输出端口。

3.数码管显示模块

代码：

module Hex8seg\_EGo1(input wire x0, x1, x2,x3,output wire [7:0] seg\_cs\_pin,//位选信号output reg [7:0] seg\_data\_0\_pin);//段选信号wire [3:0] x;assign x = {x3, x2, x1,x0};assign seg\_cs\_pin = 8'b0000\_0001;//位选，1有效//数码管显示:1位点+7位数码always @(\*) case(x)0: seg\_data\_0\_pin = 8'b0011\_1111;1:seg\_data\_0\_pin = 8'b0000\_0110;2: seg\_data\_0\_pin = 8'b0101\_1011;3: seg\_data\_0\_pin = 8'b0100\_1111;4: seg\_data\_0\_pin = 8'b0110\_0110;5: seg\_data\_0\_pin = 8'b0110\_1101;6: seg\_data\_0\_pin = 8'b0111\_1101;7: seg\_data\_0\_pin = 8'b0000\_0111; 8: seg\_data\_0\_pin = 8'b0111\_1111;9: seg\_data\_0\_pin = 8'b0110\_1111;'hA: seg\_data\_0\_pin = 8'b0111\_0111;'hB: seg\_data\_0\_pin = 8'b0111\_1100;'hC:seg\_data\_0\_pin = 8'b0011\_1001;'hD: seg\_data\_0\_pin = 8'b0101\_1110;'hE: seg\_data\_0\_pin = 8'b0111\_1000;'hF: seg\_data\_0\_pin = 8'b0111\_0001;default: seg\_data\_0\_pin = 8'b0111\_1111;endcaseendmodule

使用 case 语句来根据 x 的值选择性地给 seg\_data\_0\_pin 赋予不同的值，从而实现对每段数码管亮灭的控制

当 x 的值为 0 时，seg\_data\_0\_pin 被赋值为 8 位二进制数 8'b0011\_1111。

当 x 的值为 1 时，seg\_data\_0\_pin 被赋值为 8 位二进制数 8'b0000\_0110。

当 x 的值为 2 时，seg\_data\_0\_pin 被赋值为 8 位二进制数 8'b0101\_1011。

以此类推

此外，当 x 的值不匹配以上任何一个条件时，使用 default 条件，将 seg\_data\_0\_pin 赋值为 8 位二进制数 8'b0111\_1111。

4.六进制计数器实现模块

注：需自定义二输入与非门nand2

module nand2(  
input a,b,  
output y,  
input CP);  
reg yi,yt;  
assign y=yi;  
always @(posedge CP) begin  
yt<=~(a&b);  
end  
always @(posedge CP) begin  
yi<=yt;  
end  
endmodule

nand2代码

module Counter\_74LS161\_EGo1(

input CP, D\_0, D\_1,

output wire[7:0] seg\_cs\_pin,

output wire[7:0] seg\_data\_0\_pin);

wire clk\_100M, clk\_100, clk\_40, clk\_20, clk\_10, clk\_5, clk\_1;

clk\_div(CP, clk\_100, clk\_40, clk\_20, clk\_10, clk\_5, clk\_1);

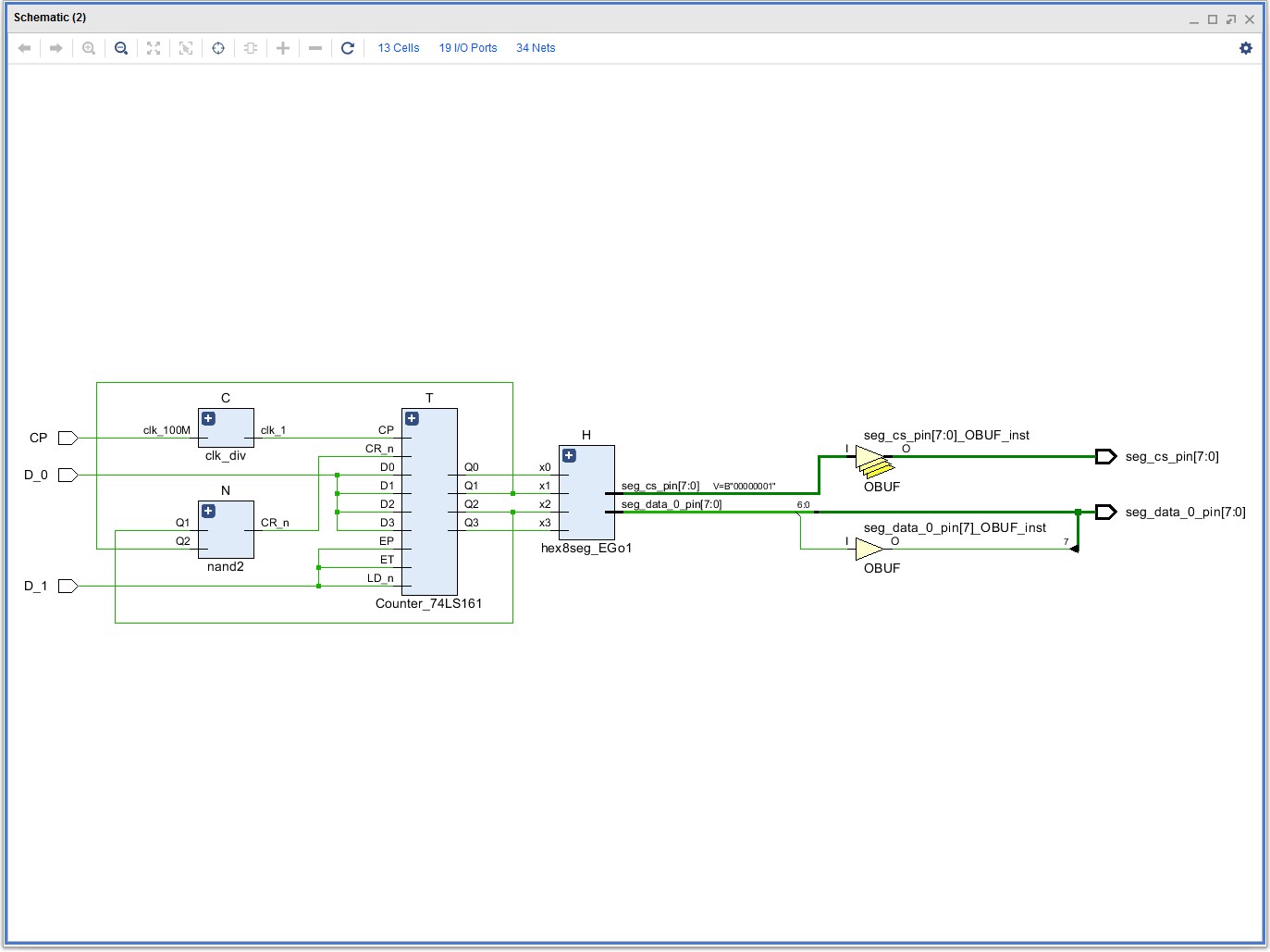
Counter\_74LS161 T(CR\_n, clk\_1, D\_0, D\_0, D\_0, D\_0, D\_1, D\_1, D\_1, Q0, Q1, Q2, Q3);

hex8seg\_EGol H(Q0, Q1, Q2, Q3, seg\_cs\_pin, seg\_data\_0\_pin)

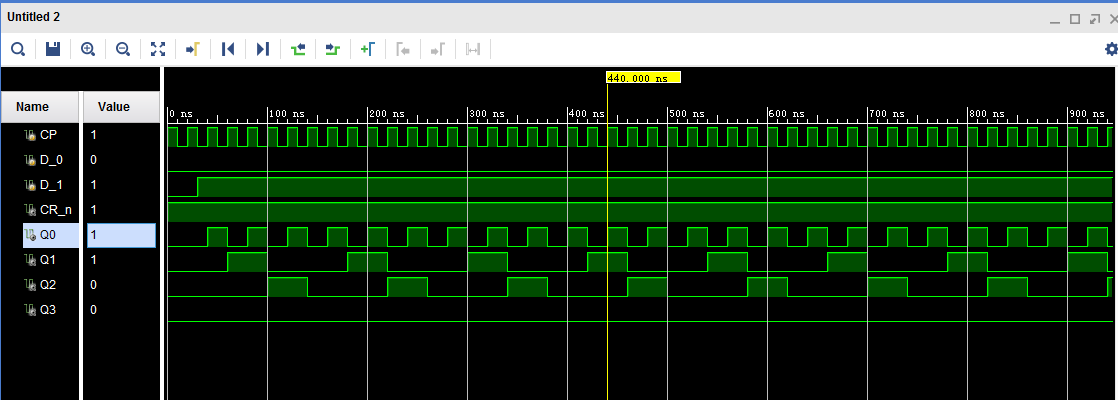
nand2 N(Q2, Q1, CR\_n);

endmodule

封装后的RTL电路：



波形仿真：



代码分析：

实现了一个计数器模块 Counter\_74LS161\_EGo1，并包含了其他两个模块 clk\_div 和 hex8seg\_EGol，以及一个 NAND 门模块 nand2。输入信号：

CP：时钟脉冲输入。

D\_0、D\_1：用作计数器的输入信号。

输出信号：

seg\_cs\_pin：七段显示器的控制信号引脚。

seg\_data\_0\_pin：七段显示器的数据引脚。

模块实例化：

clk\_div 模块：将输入的时钟信号 CP 分频得到不同频率的时钟信号。

Counter\_74LS161 模块：使用计数器 74LS161，将输入信号 D\_0、D\_1 进行计数，并输出计数结果。

hex8seg\_EGol 模块：将计数器的输出结果转换为七段显示器的控制信号和数据信号。

nand2 模块：实现了一个与非门。

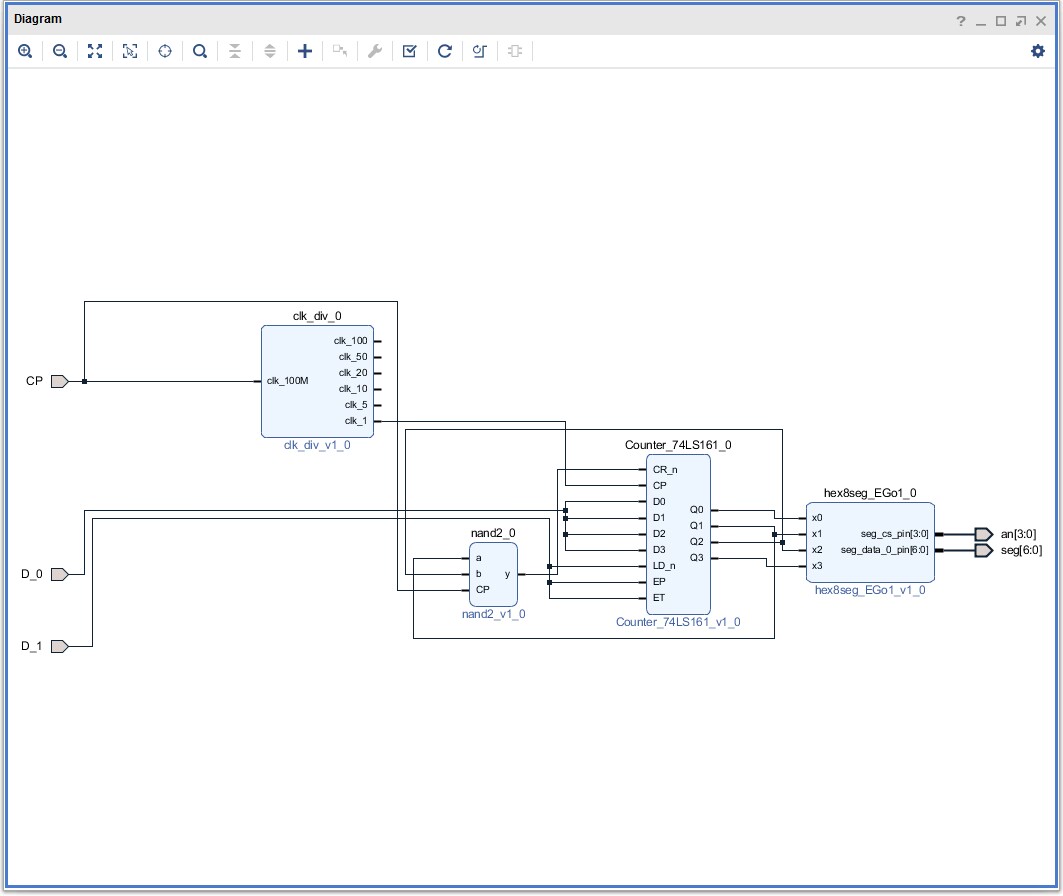
1. 约束文件



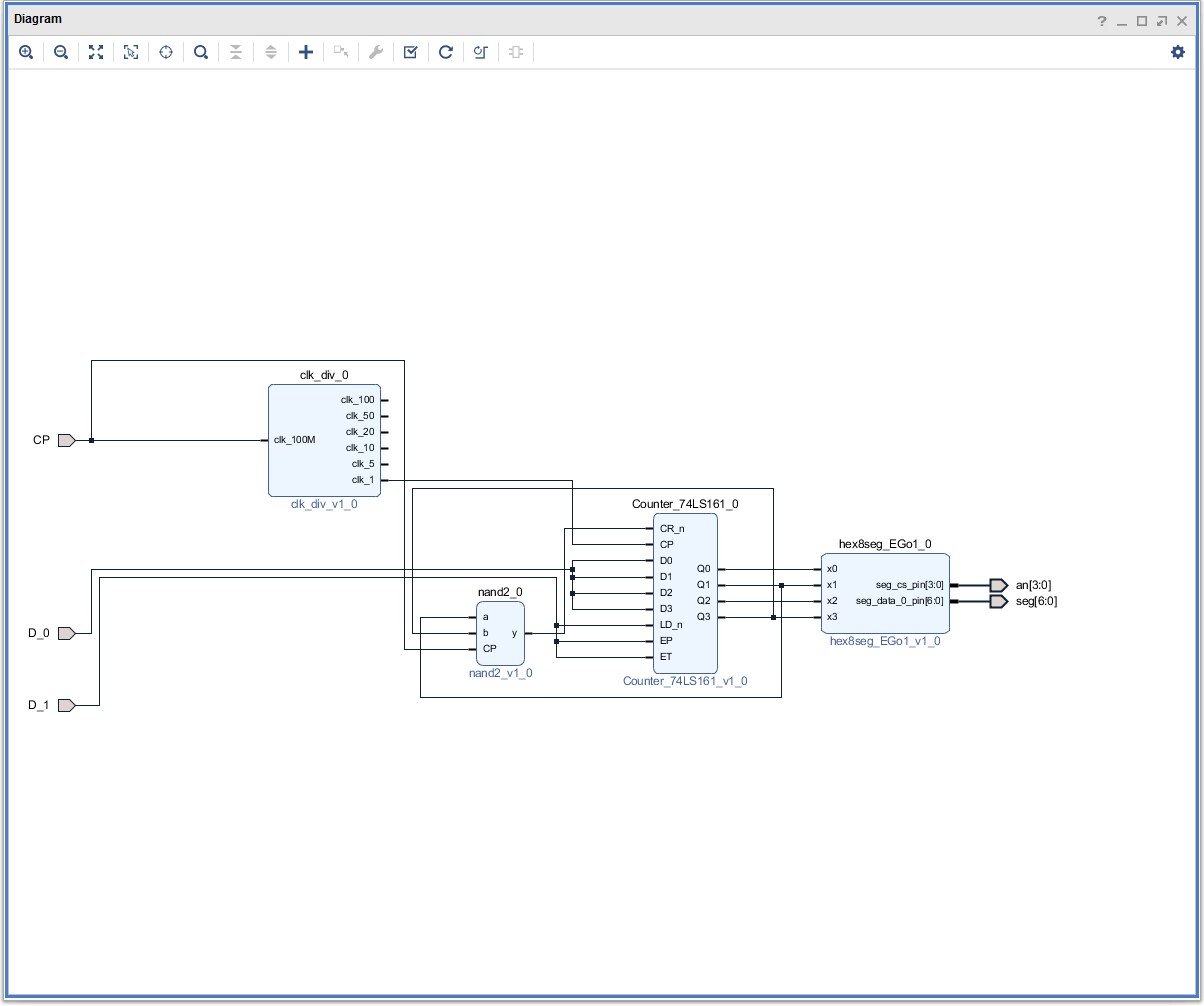
**3.3.2竞争与险象**

程序烧入开发板，表现出来4进制计数器，需要进行改进，解决竞争与险象。

**3.3.3 IP封装及原理图设计（设计10进制计数器）**

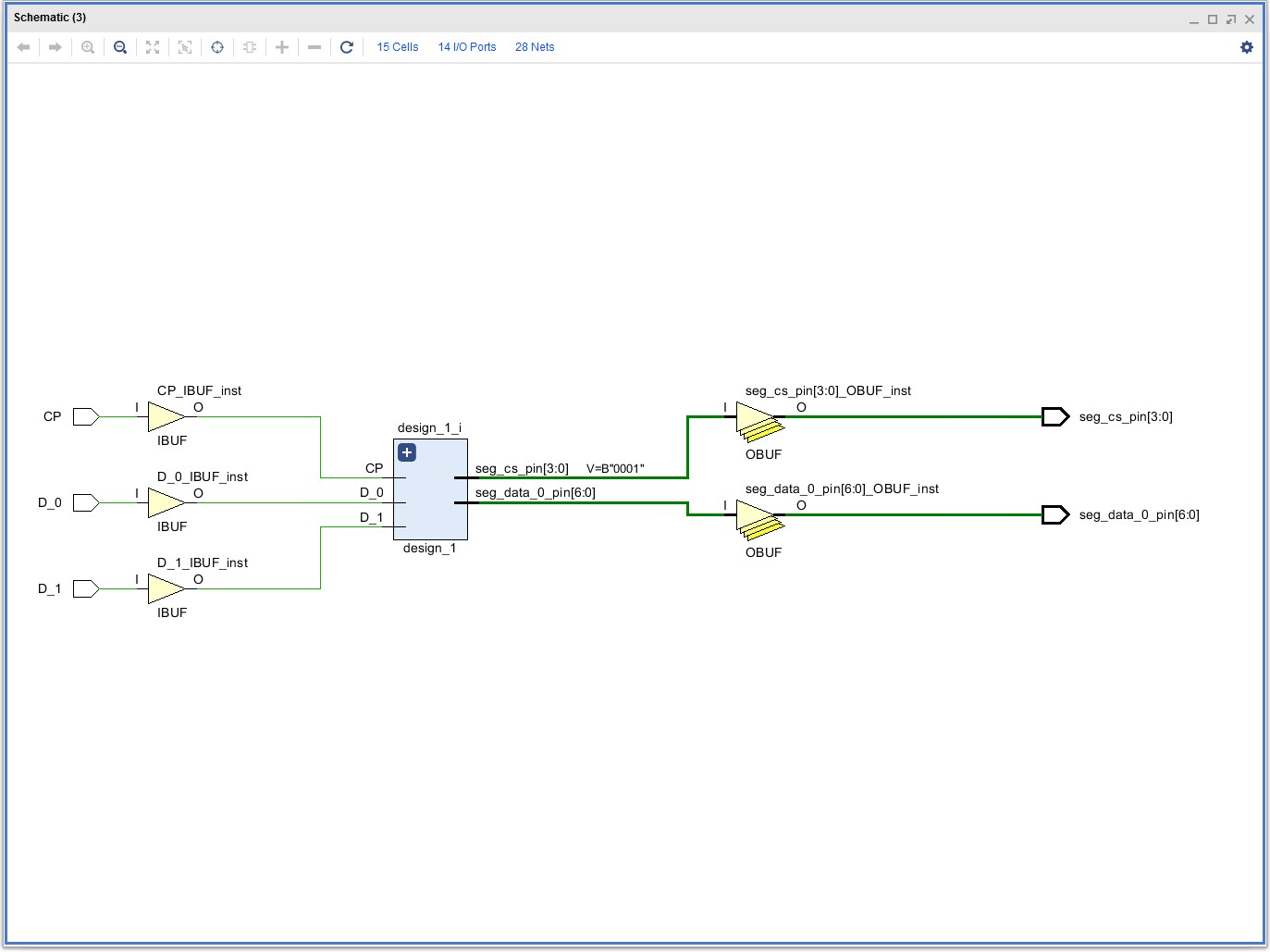


原6进制design图

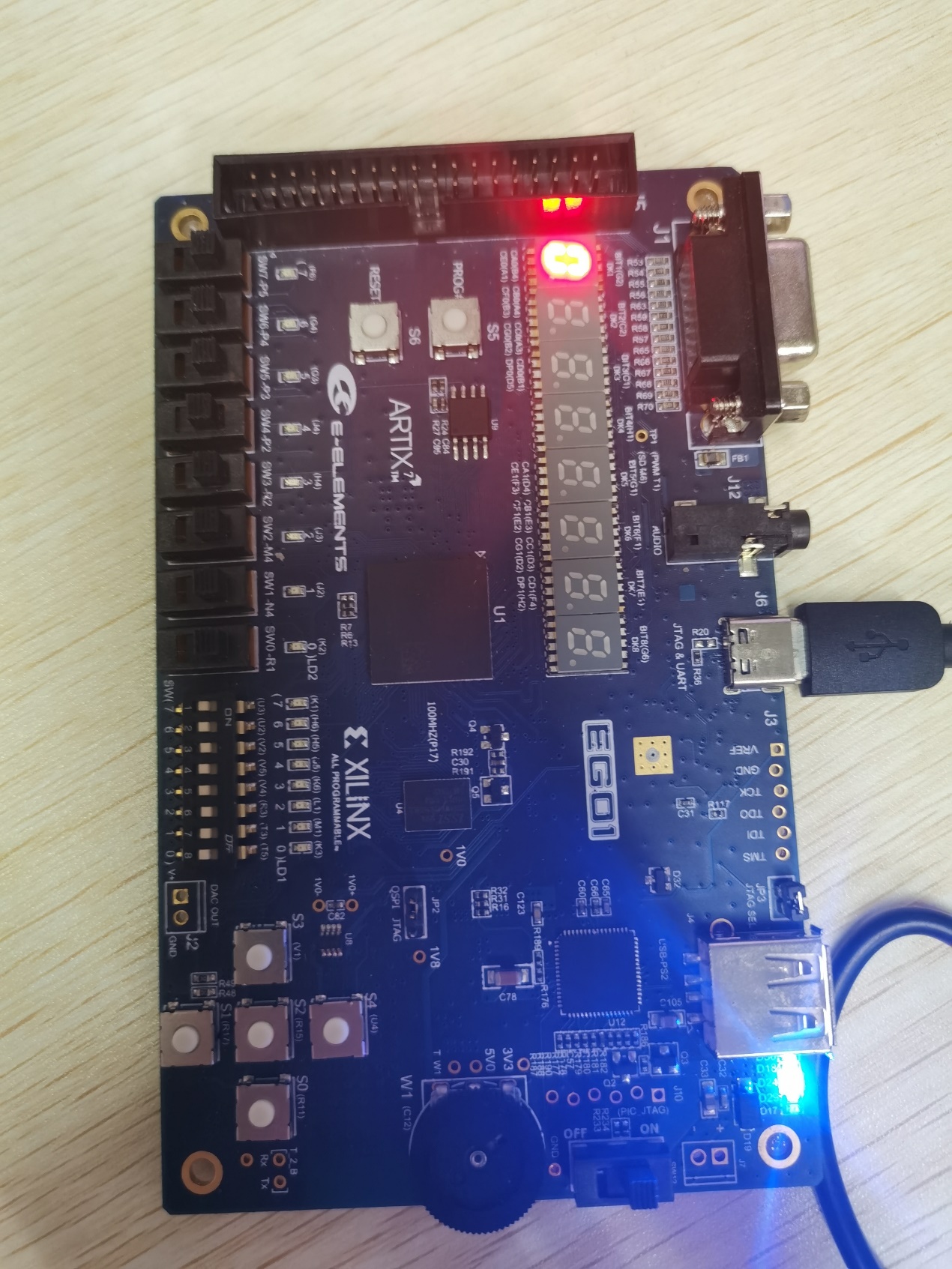


修改后的10进制design图

原理与6进制类似，将D1，D3接到nand2上，因为当D1、D3第一次同时为高时，为1010，即10



烧录至开发板验证：

。

五、调试和心得体会

**记录实验过程中遇到的问题和调试解决过程，心得体会等。**

遇到的问题主要是设计6进制计数器时，未出现逾期情况，而是变成4进制计数器。

原因：

为什么是4？

k进制计数器实际上是当计数器出现表示数字为k的状态时，直接跳过当前状态并且回到原始状态，4用四位二进制数表示为0100，上一个逻辑状态为0011，由于时序逻辑电路传输电信号速度不同，可能会出现最低位发生了翻转，次高位发生了翻转，而次低位没有发生改变，于是现了0110的逻辑状态，电路误判直接回到初始状态。

我使用了实验采用带时钟和缓存的与非门（nand2）来消除险象：

修改后的代码如下：

Module nand2(

Input a,b,

Output y,

input CP)

reg yi,yt;

assign y=yi;

always @(negedge CP)begin

yt<=~(a&b);

end

always @(posedge CP)begin

yi<-yt;

end

endmodule

修改后增加了时钟输入信号，yi与输出y连接，下降沿时将正常执行nand2功能赋值给yt，上升沿将yt赋给yi实现对输出y的控制并且解决了险象

module Counter\_74LS161\_EGo1(

input CP,D\_0,D\_1,

output wire[7:0] seg\_cs\_pin,

output wire[7:0] seg\_data\_0\_pin);

wire clk\_100M,clk\_100,clk\_40,clk\_20,clk\_10,clk\_5,clk\_1;

clk\_div C(CP,clk\_100,clk\_40,clk\_20,clk\_10,clk\_5,clk\_1);

Counter\_74LS161 T(CR\_n,clk\_1,D\_0,D\_0,D\_0,D\_0,D\_1,D\_1,D\_1,Q0,Q1,Q2,Q3);

hex8seg\_EGo1 H(Q0,Q1,Q2,Q3,seg\_cs\_pin,seg\_data\_0\_pin);

nand2 H(Q2,Q1,CR\_n,CP);

endmodule

由于对nand2进行了修改，在实例化声明时需要对H的参数重新定义，以上时修改后的代码。

运行结果如下图

