实验三　芯片的封装与应用

李雨轩 2204112913 计算机 2205

2024 年 6 月

1　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。

2　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。
3. 将芯片及相关模块封装为 IP 核，通过原理图设计实现 N 进制计数器，记录、分析仿真波形和电路图。

3　实验要求

1. 说明电路功能，分析设计、仿真代码和电路图。
2. 分析仿真波形，观察输入输出是否与预期电路功能相符（测试要全面，关注特殊情况的测试）。
3. 记录设计和调试过程。

4 实验过程及结果分析

4.1 6 进制计数器的设计

4.1.1 74LS161 芯片

1. 设计文件及仿真文件 ：

```
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
```

模块 Counter_74LS161 模拟了使用 74LS161 同步计数器 IC 实现的 4 位计数器功能。它接收多个输入信号，包括时钟 (CP)、异步复位 (CR_n)、数据输入 (D0 到 D3)、加载使能 (LD_n) 和控制信号 (EP, ET)。Data_in 信号由输入数据线派生而来。

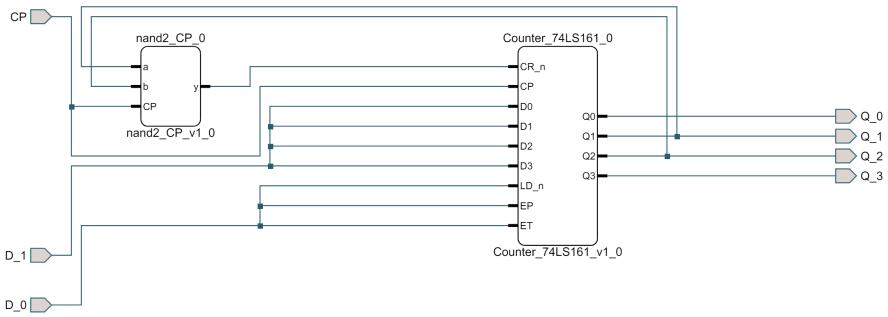
always 块在时钟上升沿 (CP) 或复位下降沿 (CR_n) 触发。根据控制信号 (LD_n, EP, ET)，它更新 Data_out 寄存器：如果复位 (CR_n) 激活，则将 Data_out 复位为 0；如果加载使能 (LD_n) 激活，则使用 Data_in 加载 Data_out；根据 EP 和 ET 的组合，保持 Data_out 或将其增加 1。

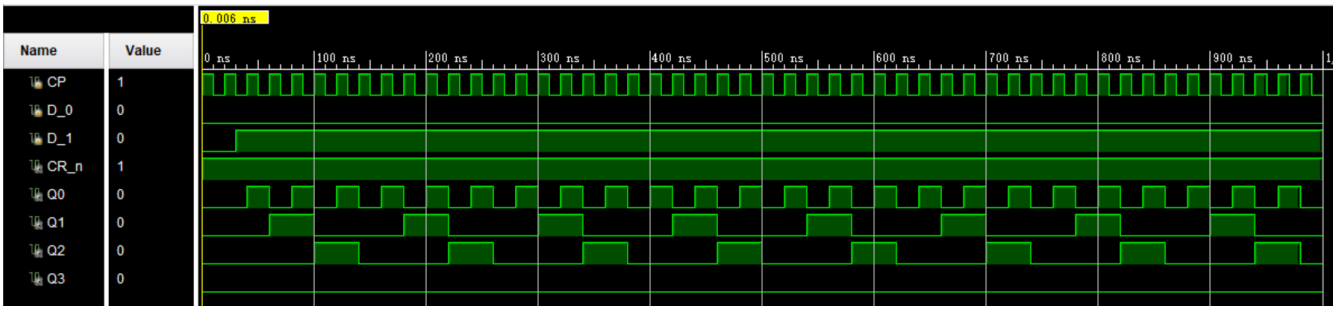
该模块输出存储在 Data_out 中的 4 位计数值 (Q0 到 Q3)，供外部使用。

模块 Counter_74LS161 的仿真文件如下：

```
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 T(CR_n,CP,D_0,D_0,D_0,D_0,D_1,D_1,D_1,Q0,Q1,Q2,Q3);
endmodule
```

2. RTL 分析及仿真结果： 通过 RTL 分析生成如下 Schematic，并使用上述仿真文件验证模块 Counter_74LS161 的正确性。





4.1.2 时钟分频及 EGo1 数码管显示模块

```
module hex8seg_EGo1(
    input wire x0, x1, x2, x3,
    output wire [7:0] seg_cs_pin,
    output reg [7:0] seg_data_0_pin);

    wire [3:0] x;
    assign x = {x3, x2, x1, x0};
    assign seg_cs_pin = 8'b0000_0001;

    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;
    endcase
endmodule
```

模块 hex8seg_EGo1 实现了一个将四位输入信号转换为七段数码管显示的功能。根据输入信号 x0 到 x3 组成的四位数，通过组合逻辑将其转换为对应的七段数码管显示数据。模块中定义了 seg_cs_pin 和 seg_data_0_pin 作为输出端口，其中 seg_cs_pin 被赋值为固定的 8 位二进制数 0000_0001，用于控制数码管选通信号。seg_data_0_pin 则根据输入信号 x 的不同取值，使用 case 语句来分别设置为对应的七段数码管数据值。如果输入的 x 值不在定义的范围内（0 到 9 和 A 到 F），则默认将 seg_data_0_pin 设置为全灭状态 0111_1111，即数码管不显示任何内容。

```
module clk_div(
    input clk_100M,
    output clk_100,clk_50,clk_20,clk_10,clk_5,clk_1);
    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 begin
        cnt_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
    end
    always @(posedge clk_100M) begin
        cnt_khz <= cnt_khz+1;
        if(cnt_khz == 16'hC34F)
            clk_1k <= ~clk_1k;
    end
    always @(posedge clk_1k) begin
        cnt_hz <= cnt_hz+1;
    end
    always @(posedge clk_1k) begin
        if(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;
    end
    assign 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 ‘clk_div‘ 实现了一个时钟分频器，将输入的 100MHz 时钟信号 ‘clk_100M‘ 分频为多个不同频率的时钟信号输出。模块中使用了多个计数器 ‘cnt_khz‘ 和 ‘cnt_hz‘ 来实现不同的分频比例。具体而言，‘cnt_khz‘ 计数到 16’hC34F（49999）时，产生 1kHz 的时钟信号 ‘clk_1k‘，作为进一步分频的基准。随后，‘cnt_hz‘ 计数的结果用于分别控制生成的时钟信号 ‘clk_100‘、‘clk_50‘、‘clk_20‘、‘clk_10‘、‘clk_5‘、‘clk_1‘ 的频率。这些时钟信号根据不同的计数条件在相应的 ‘always‘ 块内被翻转和赋值，最终通过 ‘assign‘ 语句输出到模块的端口上，供外部电路使用。

4.1.3 基于 EGo1 开发板的 6 进制计数器

```
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 N(Q2,Q1,CR_n);
endmodule
```

模块 Counter_74LS161_EGo1 组合了多个子模块，实现了一个综合性的数字电路功能。模块中包含了输入时钟信号 CP 和数据输入信号 D_0、D_1，以及输出的七段数码管控制信号 seg_cs_pin 和数据信号 seg_data_0_pin。在内部，它实例化了以下几个模块：首先是 clk_div 模块，用来分频输入时钟信号 CP，并产生多个分频后的时钟信号 clk_100、clk_40、clk_20、clk_10、clk_5、clk_1；接着是 Counter_74LS161 模块 T，用来计数并输出四位计数值 Q0、Q1、Q2、Q3；然后是 hex8seg_EGo1 模块 H，将四位计数值转换为七段数码管显示的控制信号；最后是 nand2_CP 模块 N，它使用 Q2 和 Q1 的值来控制异步复位信号 CR_n。

通过这样的组合，Counter_74LS161_EGo1 模块实现了从时钟分频、计数、数码管显示到复位控制的完整功能，每个子模块分别负责其特定的任务，最终合并在一起实现复杂的数字逻辑功能。

同时还需要如下的配置文件：

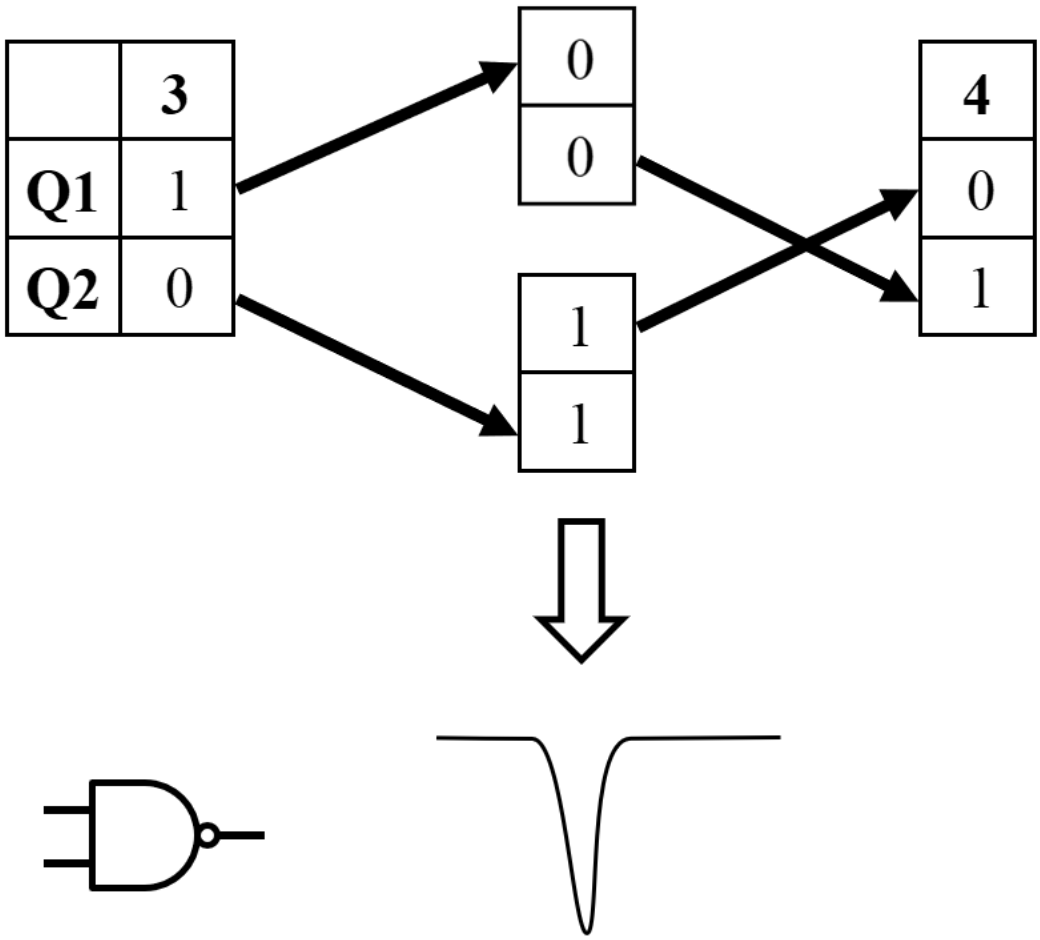
```
set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports CP]

set_property -dict {PACKAGE_PIN P5 IOSTANDARD LVCMOS33} [get_ports D_0]
set_property -dict {PACKAGE_PIN P4 IOSTANDARD LVCMOS33} [get_ports D_1]

set_property -dict {PACKAGE_PIN G2 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[0] } ]
set_property -dict {PACKAGE_PIN C2 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[1] } ]
set_property -dict {PACKAGE_PIN C1 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[2] } ]
set_property -dict {PACKAGE_PIN H1 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[3] } ]
set_property -dict {PACKAGE_PIN G1 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[4] } ]
set_property -dict {PACKAGE_PIN F1 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[5] } ]
set_property -dict {PACKAGE_PIN E1 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[6] } ]
set_property -dict {PACKAGE_PIN G6 IOSTANDARD LVCMOS33} [get_ports {seg_cs_pin[7] } ]

set_property -dict {PACKAGE_PIN B4 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[0] } ]
set_property -dict {PACKAGE_PIN A4 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[1] } ]
set_property -dict {PACKAGE_PIN A3 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[2] } ]
set_property -dict {PACKAGE_PIN B1 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[3] } ]
set_property -dict {PACKAGE_PIN A1 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[4] } ]
set_property -dict {PACKAGE_PIN B3 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[5] } ]
set_property -dict {PACKAGE_PIN B2 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[6] } ]
set_property -dict {PACKAGE_PIN D5 IOSTANDARD LVCMOS33} [get_ports {seg_data_0_pin[7] } ]
```

4.1.4 将带时钟分频和数码管显示的 6 进制计数器烧录到开发板进行验证

RTL 分析、SIMULATION 仿真分析验证功能实现无误后，通过 SYNTHESIS、IMPLEMENTATION、Generate Bitstream 以及 Program Device 步骤，成功在 EGo1 开发板上实现计数器功能。但是发现实际运行时 EGo1 开发板上实现的是一个四进制的计数器，原因在于不

同路径输入信号变化传输到同一点门级电路，时间上有先后，也就是存在竞争现象导致冒险产生了错误输出，如下图展示的那样

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Q0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Q1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Q2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Q3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

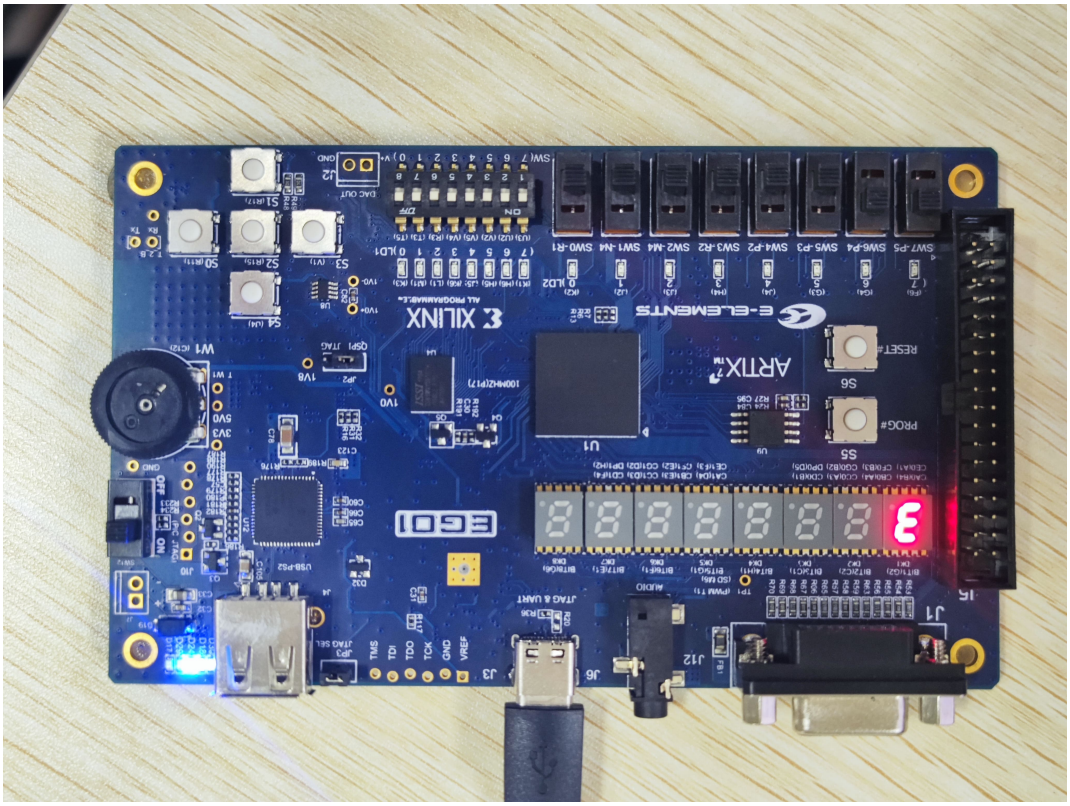


因此修改 nand2 模块如下，同时将模块 `Counter_74LS161_EGo1` 中对于 nand2 的调用修改为 `nand2_CP N(Q2,Q1,CR_n);`

```
module nand2_CP(
    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
```

模块 nand2_CP 实现了一个带有时钟控制的 NAND 门。它的设计思路是利用两个时钟边沿敏感的触发器（always 块），一个在时钟下降沿触发，另一个在时钟上升沿触发。这种设计与普通的 NAND 门不同之处在于，它确保了输出 y 在时钟信号 CP 的稳定时刻更新，避免了在输入变化时输出 y 不稳定的情况发生。这种时钟控制的 NAND 门适用于需要同步输入和输出的电路设计，保证了在时钟信号的不同相位上，输入信号的变化不会导致竞争条件的出现，确保了信号在时序上的正确性和稳定性。

修改后再次由 RTL 分析、SIMULATION 仿真分析验证功能实现无误后，通过 SYNTHESIS、IMPLEMENTATION、Generate Bitstream 以及 Program Device 步骤，成功在 EGo1 开发板上实现计数器功能。如下图所示：



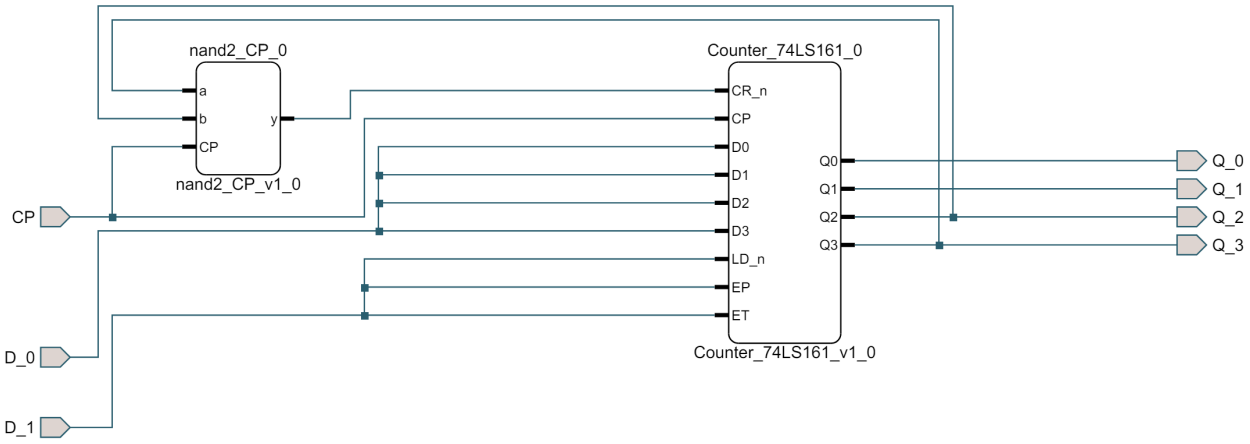
4.2 原理图设计 13 进制计数器（0-12）

由于开发板验证电路正确后，可以将各子模块封装为 IP，采用原理图设计方式实现 N 进制计数器，每个模块单独工程封装。

所以在完成对二输入与非门（nand2）模块和 71LS161 芯片模块的 IP 封装后，在原理图设计-新建工程-设置-添加 IP 目录后，本人采用原理图设计实现一个 13 进制计数器（学号后两位）

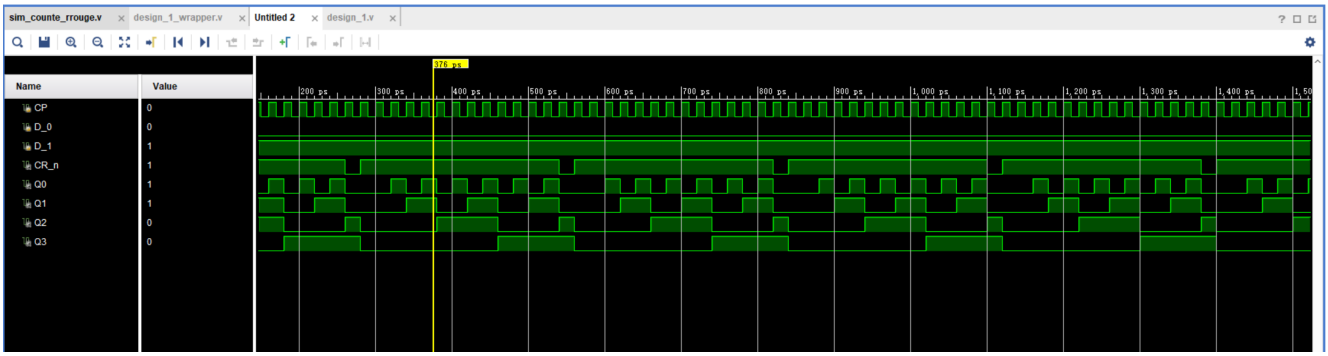
通过鼠标拖动连接各模块接口连线；选中输入/输出端口，右键选择“Create Port/Make External”创建输入输出接口。

下面是 13 进制计数器原理图:



实现的逻辑就是将计数值为 12 时作为起跳状态，检测到 2 时向同步置零信号发送 1，这样下一时刻就会进入 0。

同时编写仿真文件，产生波形图如下:



5 思考题: 如何利用多片 74LS161 芯片实现如 24 进制计数器

为实现模 24 的计数器（计数范围从 00000 到 10111），通过级联多个 74LS161 芯片来实现所需的计数范围。在这之前要为 74LS161 芯片添加进位输出 (RCO) 。

5.1 设计思路

需要计数从 0 到 23（十进制），即从 00000 到 10111（二进制）。对于这个一个 5 位二进制计数器，可以分解为两个部分：

- 一个 4 位计数器，实现低 4 位的计数。
- 一个 2 位计数器，实现高 1 位的计数。

具体来说，我们可以用以下方式实现：

1. 使用两个 74LS161 芯片级联来实现 5 位计数器。
2. 第一个 74LS161 芯片（低位计数器）计数从 0000 到 1111。
3. 第二个 74LS161 芯片（高位计数器）实现对低位计数器的溢出计数，即在低位计数器从 1111 变为 0000 时，高位计数器计数加 1。

5.2 具体电路连接

5.2.1 低位计数器（74LS161_1）

- 时钟输入 (CP) 接系统时钟。
- 异步清零 (CR_n) 接地（低电平有效）。
- 使能输入 (ET 和 EP) 接高电平。
- 进位输出 (RCO) 连接到高位计数器的使能输入 (ET)。
- 数据输入 (D0-D3) 接地。
- 加载使能 (LD_n) 接高电平。

5.2.2 高位计数器（74LS161_2）

- 时钟输入 (CP) 接低位计数器的进位输出 (RCO)。
- 异步清零 (CR_n) 接地（低电平有效）。
- 使能输入 (EP) 接高电平，(ET) 接低位计数器的进位输出 (RCO)。
- 数据输入 (D0-D3) 仅需要 D0 和 D1，接地。
- 加载使能 (LD_n) 接高电平。

5.2.3 实现模 24 的逻辑

- 高位计数器的输出 Q1 和 Q0 以及低位计数器的输出 Q3 和 Q2 构成 5 位计数器的前 5 位。
- 当计数值达到 24（11000）时，需要复位两个计数器。
- 可以通过组合逻辑来检测高位计数器输出 Q1 为 1、Q0 为 0、低位计数器 Q3 为 1 和 Q2 为 0，并通过一个与门控制两个计数器的 CR_n 输入。

5.3 verilog 代码实现

```
module mod24_counter (
    input wire clk,
    input wire reset,
    output wire [4:0] count
);
    wire [3:0] low_count;
    wire [1:0] high_count;
    wire rco_low, reset_all;

    // 低位计数器
    Counter_74LS161 low_counter (
        .CP(clk),
        .CR_n(reset_all),
        .EP(1'b1),
        .ET(1'b1),
        .LOAD(1'b1),
        .D(4'b0000),
        .Q(low_count),
        .RCO(rco_low)
    );

    // 高位计数器
    Counter_74LS161 high_counter (
        .CP(rco_low),
        .CR_n(reset_all),
        .EP(1'b1),
        .ET(rco_low),
        .LOAD(1'b1),
        .D(4'b0000),
        .Q({2'b00, high_count}),
        .RCO()
    );

    // 检测模24的复位逻辑
    assign reset_all = reset | (high_count == 2'b10 && low_count[3:2] == 2'b11);
    assign count = {high_count, low_count};

endmodule
```

通过上述步骤，可以成功实现一个模 24 的计数器。

6 调试和心得体会

6.1 调试过程

在初步在 EGo1 开发板上验证时，发现实际运行时计数器显示的是一个四进制计数器，而不是预期的六进制计数器。经过仔细观察和分析，发现是由于竞争和冒险现象导致了错误的计数输出。所以我进行了如下的调试过程：

- 修改 NAND 门模块的 Verilog 代码，增加时钟控制逻辑，确保同步更新输出。
- 使用 RTL 分析和仿真工具验证修改后的功能正确性。
- 重新生成 Bitstream 并在 EGo1 开发板上烧录，验证计数器功能成功按预期工作

6.2 心得体会

在本次实验中，我深入理解了 Verilog 语言的基本语法和模块化设计原则。通过对 74LS161 计数器芯片的实现，我不仅学会了如何编写和调试 Verilog 代码，还掌握了如何利用仿真工具验证电路功能的正确性。

通过本次实验，我也意识到了实验设计和文档记录的重要性。详细的实验设计和清晰的文档记录帮助我更好地理解和复习实验内容，确保了实验的顺利进行和成果的有效展示。

在今后的学习和工作中，我将继续深入学习数字电路设计和 FPGA 开发技术，不断提升自己的实践能力和解决问题的能力。通过更多实验和项目，我希望能够在硬件设计领域取得更多的学习。