

设计报告

课程名称：EDA 原理及应用

设计内容：二进制数转换为 BCD 码

学 院：信息科学与技术学院

专业班级：信工 1702 班

姓 名：黄玥

学 号：2017040481

2019 年 11 月 28 日

北京化工大学

一. 设计要求

（信工专业）通过不同的算法，实现 N 位二进制数到 BCD 码的转换并通过行为级和时序仿真对设计进行验证，要求如下：

- （1）数据经过寄存器后，执行 9 位二进制数到 BCD 码的转换。
- （2）设计和调用函数来实现 9 位二进制数到 BCD 码的转换。
- （3）在行为仿真时，调用系统任务 \$display，在 Tcl Console 窗口中显示结果。

（4）拓展设计，实现 16 位二进制数到 BCD 码的转换。

二. 设计思路（九位二进制数）

1. 转换原理

由二进制数转换为 BCD 码实际上也是一种由二进制数向十进制数的转换。这种转换有利于将未来设计使数字能够在 LED 灯或数码管上进行逐位显示。

思路一：基于数学分析

对于 9 位二进制数，最大值为 512，因此需要 3 个四位 BCD 码，分别代表数据的百位、十位、个位。考虑使用数学方法实现，即 9 位二进制数除以 100 所得的整数应为百位，即可得最高位 BCD 码，对 100 取余再除以 10 所得的整数应为十位，即可得次高位 BCD 码，对 10 取余所得的整数应为个位，即可得最低位 BCD 码。这种方法较为简单，但考虑到这种方法没有使用寄存器，与题目中要求不符合，所

以放弃该思路。

思路二：基于左移移位加三法

我们需要转换的是 9 位二进制数，由于 9 位二进制数的最大值为 512，所以使用 3 个四位 BCD 码即可转换。需要计算二进制数和 BCD 码之间的转换关系。考虑使用左移的方法实现转换，将 9 位二进制数逐渐左移进 BCD 码中，则需要 9 次移位，并对移位数据进行校准。校准过程如下：由于每一组 BCD 码对应的十进制数都应小于等于 10（因为这三组四位 BCD 码分别代表十进制数的百位、十位、个位），即不应大于等于 1010，即进行左移一位的操作前不应大于等于 0101，即进行左移操作前 BCD 码对应的十进制数不应大于 5。如果数据大于等于 5，则加上 3 后再进行左移。（我们的每一组 BCD 码对应的十进制数都不能大于等于 10，即逢 10 进位，而正常的 4 位二进制是满 16 才进一位。这时候我们就需要将数值中的 10 变成 16，可以将该数值加 6 即 0110。对左移一位前的上一个值来说，就是加 0011 即加 3）。

举例验证方法正确性：例如十进制数 377，它的二进制码应为 1 0111 1001，而对应的三个四位 BCD 码应为 0011 0111 0111，分别对应十进制数的百位、十位、个位。我们可以分析它的移位和校准功能如下表：

	BCD (百位)	BCD (十位)	BCD (个位)	原二进制码 (串行输入)
未移位	0 0 0 0	0 0 0 0	0 0 0 0	1 0111 1001
左移 1 位	0 0 0 0	0 0 0 0	0 0 0 1	0111 1001
左移 2 位	0 0 0 0	0 0 0 0	0 0 1 0	111 1001
左移 3 位	0 0 0 0	0 0 0 0	0 1 0 1	11 1001
判断已大于等于 5，进行校准			1000 (加 3)	
左移 4 位	0 0 0 0	0 0 0 1	0 0 0 1	1 1001
左移 5 位	0 0 0 0	0 0 1 0	0 0 1 1	1001
左移 6 位	0 0 0 0	0 1 0 0	0 1 1 1	001
判断已大于等于 5，校准			1010 (加 3)	
左移 7 位	0 0 0 0	1 0 0 1	0 1 0 0	01
判断已大于等于 5，校准		1100(加 3)		
左移 8 位	0 0 0 1	1 0 0 0	1 0 0 0	1
判断已大于等于 5，校准		1011(加 3)	1011 (加 3)	
左移 9 位	0 0 1 1	0 1 1 1	0 1 1 1	

左移并校准后的 BCD 码为 001101110111，即百位为 3，十位为 7，个位为 7，与我们手动计算得到的十进制数 377 相等。

验证可知，这种方法是正确的。

2. 框架设计

输入：时钟沿、清零信号、9 位二进制数

输出：三组四位 BCD 码

设计过程：

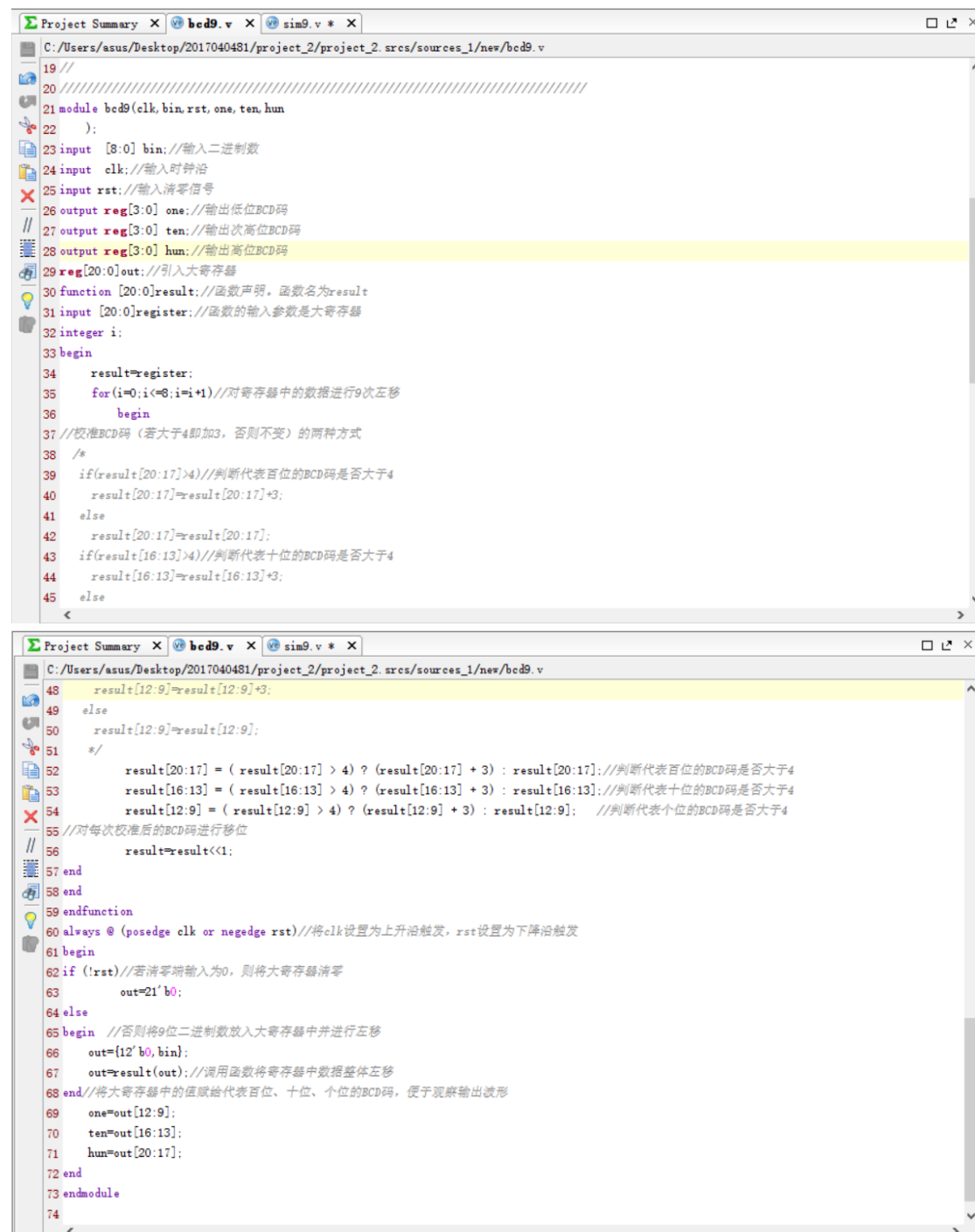
将三组四位 BCD 码和 9 位二进制数放入一个大寄存器类型的数组中，9 位二进制数放在最右边，BCD 码按照百位、十位、个位的顺序放在左边。每次将寄存器数组内数据整体左移一位，并判断 3 组 4 位 BCD 码是否大于等于 5（即大于 4），若是，执行加 3 操作后再左移，不是，直接左移。

时钟沿作用：时钟上升沿来时，读取新的 9 位二进制码并送入寄存器数组执行转换操作。

清零端作用：清零信号下降沿有效，下降沿来时，不管输入的二进制码为多少，输出的三组四位 BCD 码都清零。

三. Verilog 语言实现

Design Sources 设计（相关注释已写在设计文件中）



```
19 //
20 ///////////////////////////////////////////////////////////////////
21 module bcd9(clk, bin, rst, one, ten, hun
22 );
23 input [8:0] bin; //输入二进制数
24 input clk; //输入时钟沿
25 input rst; //输入清零信号
26 output reg[3:0] one; //输出低位BCD码
27 output reg[3:0] ten; //输出次高位BCD码
28 output reg[3:0] hun; //输出高位BCD码
29 reg[20:0] out; //引入大寄存器
30 function [20:0] result; //函数声明，函数名为result
31 input [20:0] register; //函数的输入参数是大寄存器
32 integer i;
33 begin
34     result=register;
35     for(i=0;i<=8;i=i+1) //对寄存器中的数据进行9次左移
36     begin
37 //校正BCD码（若大于4即加3，否则不变）的两种方式
38 /*
39 if(result[20:17]>4) //判断代表百位的BCD码是否大于4
40     result[20:17]=result[20:17]+3;
41 else
42     result[20:17]=result[20:17];
43 if(result[16:13]>4) //判断代表十位的BCD码是否大于4
44     result[16:13]=result[16:13]+3;
45 else
46     result[16:13]=result[16:13];
47 if(result[12:9]>4) //判断代表个位的BCD码是否大于4
48     result[12:9]=result[12:9]+3;
49 else
50     result[12:9]=result[12:9];
51 */
52     result[20:17] = ( result[20:17] > 4 ) ? ( result[20:17] + 3 ) : result[20:17]; //判断代表百位的BCD码是否大于4
53     result[16:13] = ( result[16:13] > 4 ) ? ( result[16:13] + 3 ) : result[16:13]; //判断代表十位的BCD码是否大于4
54     result[12:9] = ( result[12:9] > 4 ) ? ( result[12:9] + 3 ) : result[12:9]; //判断代表个位的BCD码是否大于4
55 //对每次校正后的BCD码进行移位
56     result=result<<1;
57 end
58 endfunction
59 always @ (posedge clk or negedge rst) //将clk设置为上升沿触发，rst设置为下降沿触发
60 begin
61 if (!rst) //若清零端输入为0，则大寄存器清零
62     out=21'b0;
63 else
64 begin //否则将9位二进制数放入大寄存器中并进行左移
65     out={12'b0, bin};
66     out=result(out); //调用函数将寄存器中数据整体左移
67 end //将大寄存器中的值赋给代表百位、十位、个位的BCD码，便于观察输出波形
68     one=out[12:9];
69     ten=out[16:13];
70     hun=out[20:17];
71 end
72 endmodule
73
74
```

要求一.寄存器使用：将二进制数和 BCD 码都放进大寄存器数组中进行整体移位。由于寄存器数组的低 9 位存放 9 位二进制数，即

out[8:0], 根据左移移位的规则, 应在第 13 位到第 10 位存放表示个位的 BCD 码, 即 out[12:9], 在第 17 位到第 14 位存放表示十位的 BCD 码, 即 out[16:13], 在第 21 位到第 18 位存放表示百位的二进制码, 即 out[20:17]。最后将此数组中的值输出到百位、十位和个位。

要求二.函数调用: 将大寄存器的移位和校准功能写在函数中, 输入的是只包含二进制数, 未移位形成 BCD 码的寄存器数组内容, 输出的是已完成移位校准、已产生 BCD 码的寄存器数组内容, 函数名为 result。

Simulation Sources 设计 (相关注释已写在仿真文件中)

(文件中所写的赋值给二进制数的循环和单独赋值选用一个即可)。



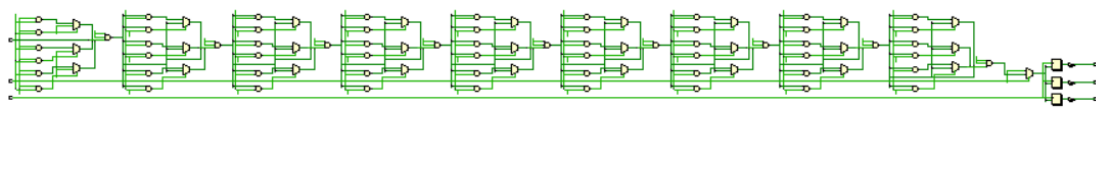
```
20 ///////////////////////////////////////////////////////////////////
21 module sim9;
22 reg [8:0] bin; //输入用reg类型
23 reg clk;
24 reg rst;
25 wire [3:0] hun; //输出用wire类型
26 wire [3:0] ten;
27 wire [3:0] one;
28 integer i;
29 bcd9 uut(//将sim9中的变量与bcd9中的变量对应起来
30 .clk(clk),
31 .rst(rst),
32 .bin(bin),
33 .hun(hun),
34 .ten(ten),
35 .one(one)
36);
37 initial
38 begin//设置初始的输入脉冲和清零信号
39 rst=0;
40 clk=1;
41 #100;
42 rst=1;
43 end
44 always//设置输入的脉冲信号每100ns翻转一次
45 begin
46 clk=1;
```

```
Project Summary X bcd9.v X sim9.v * X
C:/Users/asus/Desktop/2017040481/project_2/project_2_srcs/sim_1/new/sim9.v
46 clk=1;
47 #100;
48 clk=0;
49 #100;
50 end
51 always//输入9位二进制数, 方便起见使用一个循环, 后一个数比前一个数大1
52 begin
53   for(i=20;i<57;i=i+3)
54     begin
55       bin=i;
56       #200;
57       //调用系统函数在Icl Console中显示结果
58       $display("the initial number is:%b",bin);//显示初始9位二进制码
59       $display("the initial number(decimal) is:%d",bin);//显示初始9位二进制码对应的十进制数
60       $display("the bcd number is:%b %b %b",hun,ten,one);//显示3组四位BCD码
61       $display("the bcd number(decimal) is:%d %d %d",hun,ten,one);//显示BCD码所代表的十进制数
62     end
63   /*
64   bin=9'b000001110;//也可单独输入9位二进制数
65   #2000;
66   $display("the initial number is:%b",bin);
67   $display("the initial number(decimal) is:%d",bin);
68   $display("the bcd number is:%b %b %b",hun,ten,one);
69   $display("the bcd number(decimal) is:%d %d %d",hun,ten,one);
70   */
71 end
72 endmodule
```

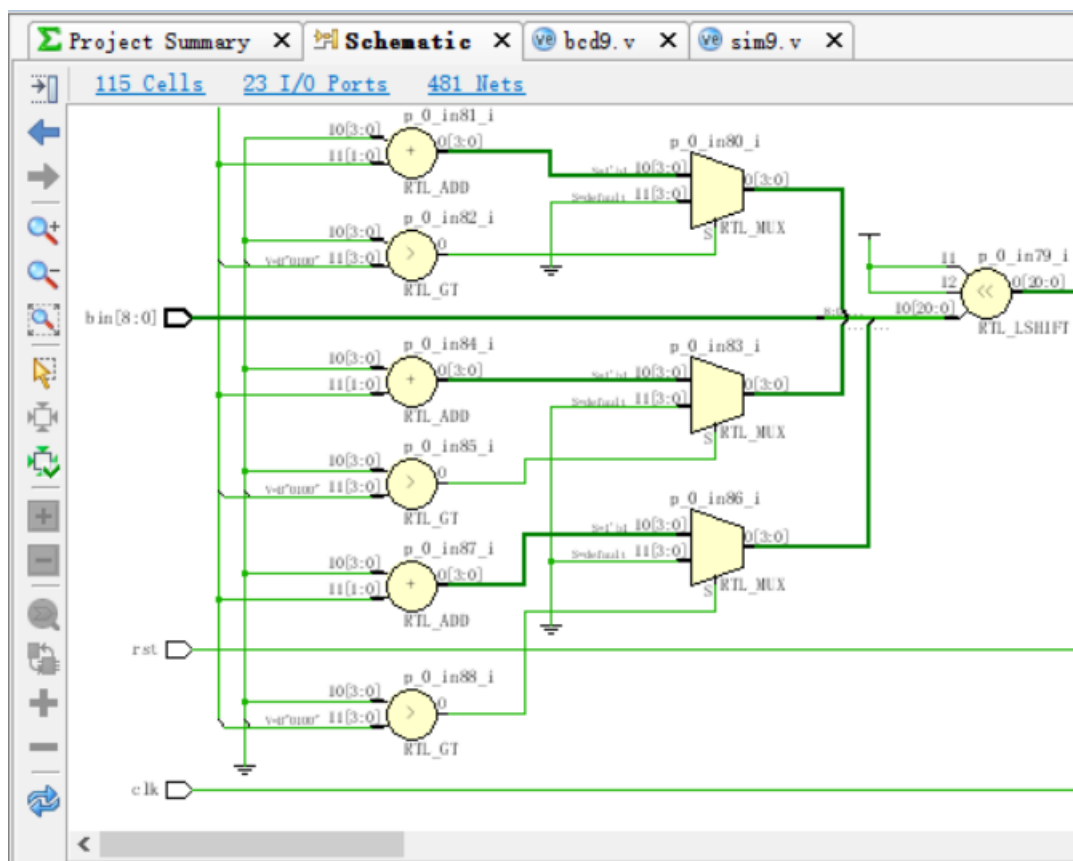
三. 运行结果

RTL 详细描述

使用 RTL 分析中的详细描述功能, 观察生成的原理图:

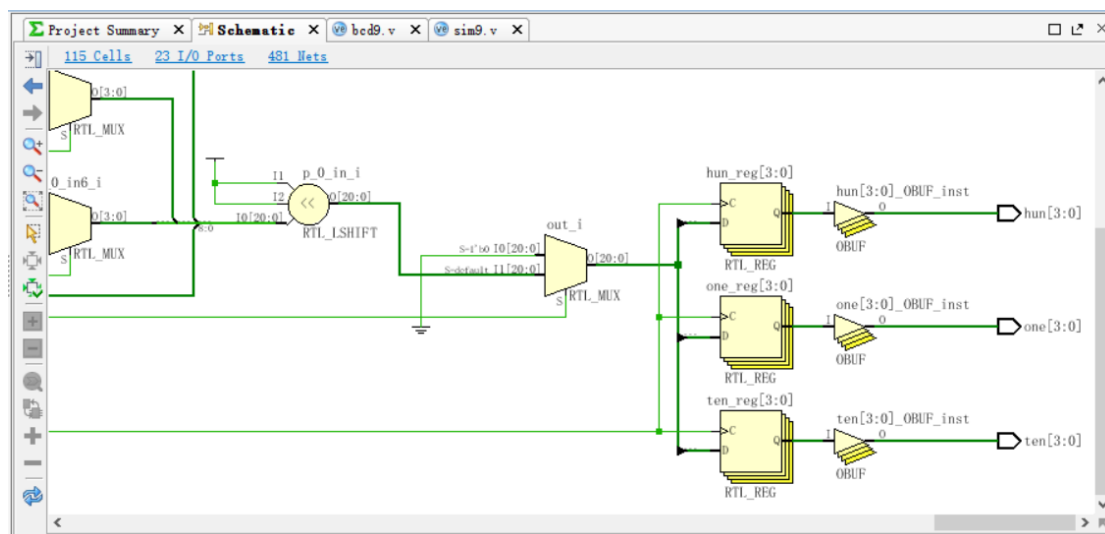


将它放大, 观察输入和输出:

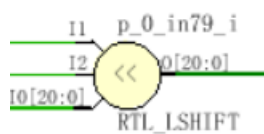


可以看到，输入管脚确实为设置的三个输入参数，即 9 位二进制数、时钟输入和清零端。

而后面的器件实现的即为若四位 BCD 码大于 4，则执行+3 操作的过程。由于有百位、十位、个位三位上的 BCD 码，所以每次移位后都有共 3 组这样的器件对 3 组 BCD 码进行校准工作。

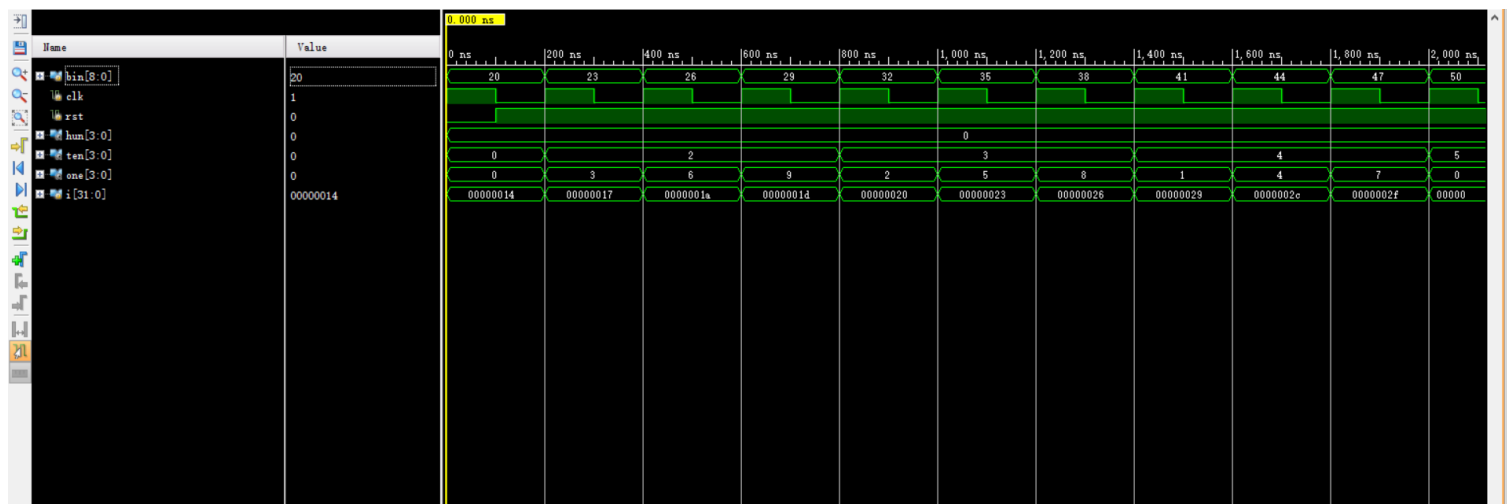


可以看到，输出的三个管脚为设置的 BCD 码的百位、十位、个位。由于每个 BCD 码都为四位，因此每组 BCD 的输出端都有四个 D 触发器相组合。C 端口都接在输入端的时钟脉冲上，为上升沿触发。由总的器件原理图我们可以看见，由于需要将 9 位二进制数转化为 BCD 码需要 9 次移位，整个图中共有 9 个如图所示的实现移位功能的器件。



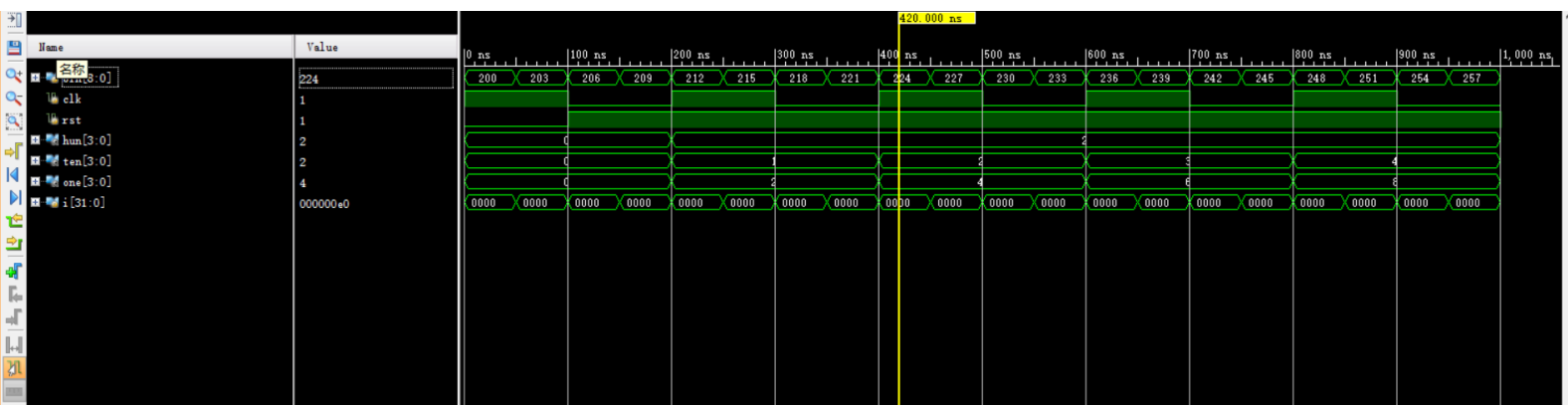
所以根据原理图，可以这样描述它的功能：将二进制数、时钟、清零信号输入，对代表百位、十位、个位的四位 BCD 码分别进行判断，若大于 4，则执行加 3 操作，判断完毕后，将它们和二进制数放在一起，并进行左移一位的操作。重复上述操作九次，由此分析，原理图和所写的 verilog 语言所描述的功能是相同的。

行为级仿真



此处我设置的时钟为每 100ns 翻转一次，即以 200ns 为周期。由于初始状态，我将清零信号 `rst` 设置为了 0，即清零，所以一开始的 BCD 码的百位、十位、个位都为 0，而后面清零信号 `rst` 置 1，能够正常输出，所以每来一个时钟脉冲的上升沿，进行一次由二进制码向 BCD 码的转换。由于此处我设置的二进制数的变化周期与时钟周期相等，不能很好的观察这一特性，因此将输入二进制数的变化周期设置为小于时钟周期，再运行行为级仿真。

如图所示，在一个时钟周期内，虽然二进制数变化不止一次，但 BCD 码的值总是等到时钟上升沿时才变化，即时钟上升沿来时，进行由二进制数到 BCD 码的转换；时钟上升沿没有来时，尽管输入的二进制数发生了改变，输出也不会改变，即不执行将二进制数转化为 BCD 码的操作。



在图中光标处，输入的二进制数转化为十进制数为 224，因此输出的 BCD 码的百位、十位、个位分别为 2 2 4，即结果正确。

将二进制数的变化周期重新设置为与时钟周期相等，便于观察调用系统函数 display 在 Tcl Console 中显示的结果是否正确。

```
Tcl Console
the initial number is:011001011
the initial number(decimal) is:203
the bcd number is:0010 0000 0011
the bcd number(decimal) is: 2 0 3
the initial number is:011001110
the initial number(decimal) is:206
the bcd number is:0010 0000 0110
the bcd number(decimal) is: 2 0 6
the initial number is:011010001
the initial number(decimal) is:209
the bcd number is:0010 0000 1001
the bcd number(decimal) is: 2 0 9
```

截取其中一组输出结果：

```
the initial number is:011001110
the initial number(decimal) is:206
the bcd number is:0010 0000 0110
the bcd number(decimal) is: 2 0 6
```

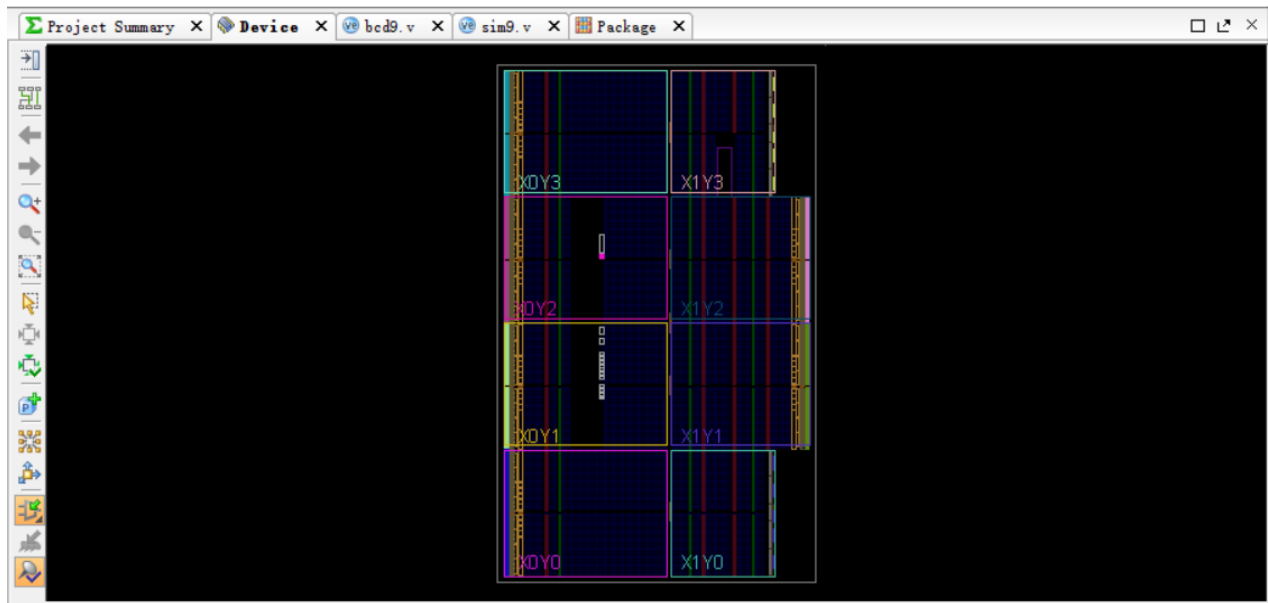
即输入的九位二进制码为 0 1100 1110，二进制码对应的十进制数为 206，输出的 BCD 码为 0010 0000 0110，输出的 BCD 码分别对应的百位、十位、个位数为 2 0 6，即结果正确。

设计综合

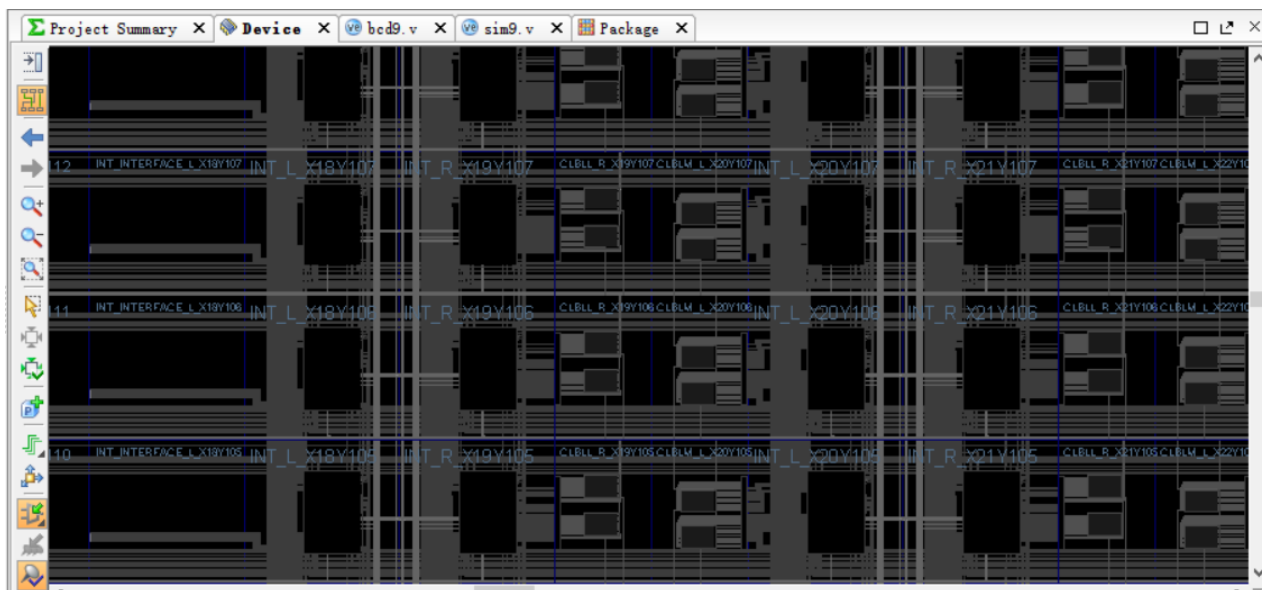
综合就是将 RTL 级的设计描述转换成门级的描述。（按照何老师

的慕课视频中的过程进行分析)

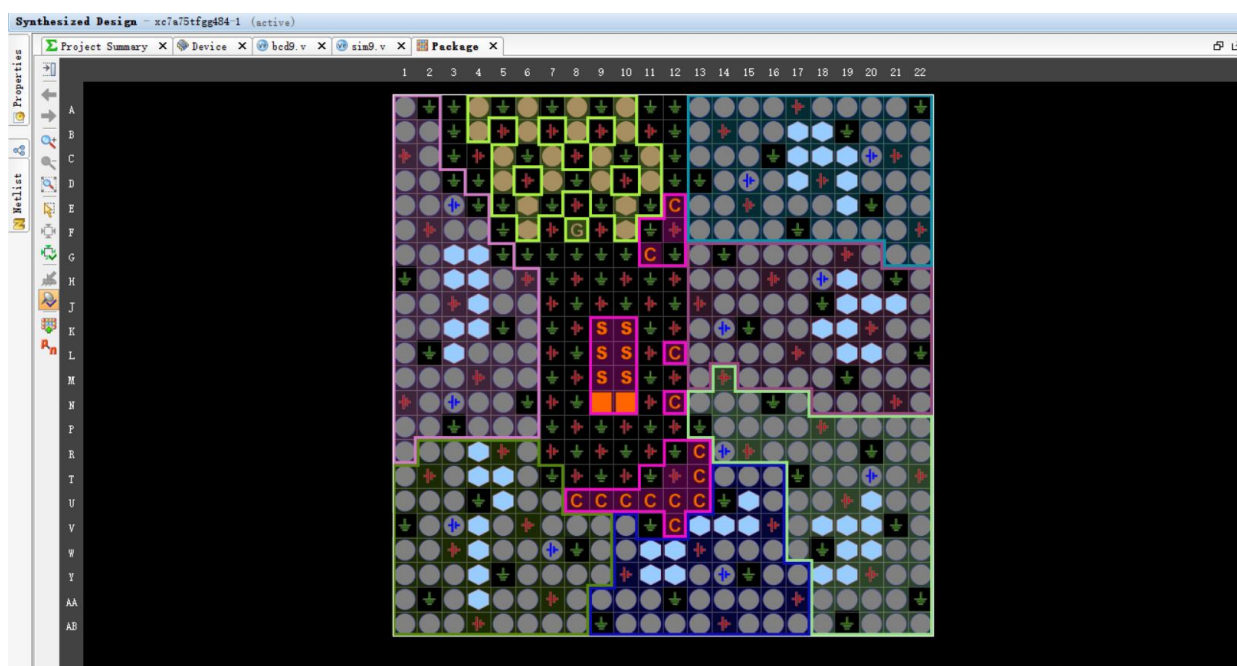
(1) Device 视图



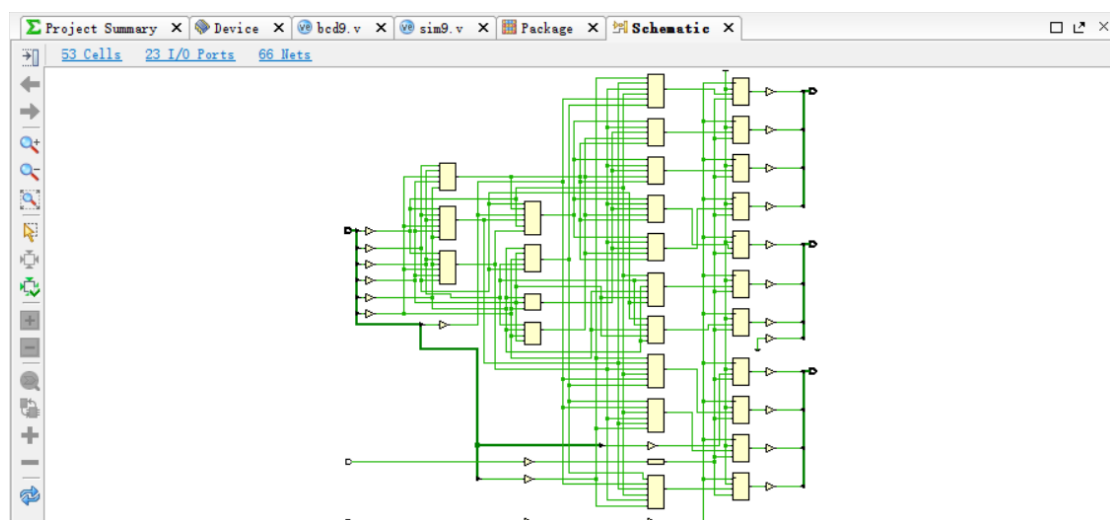
点击布线按钮并放大后可查看内部逻辑资源情况



(2) Package 观察引脚封装视图



(2) 综合后的原理图

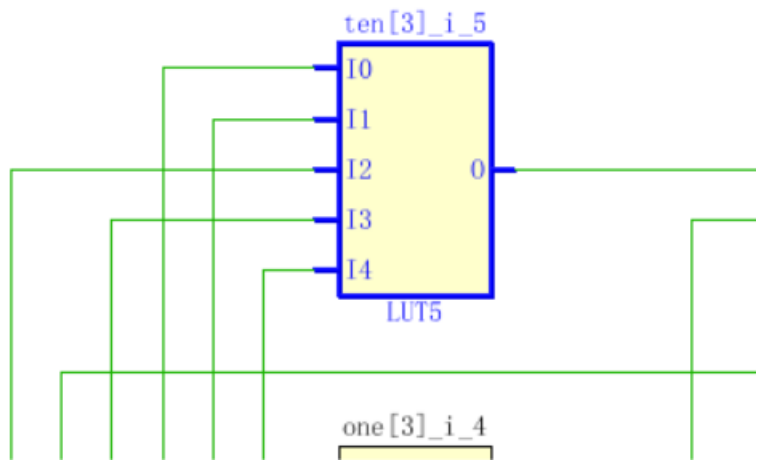


与前面 RTL 分析时的原理图有所不同，这是因为 FPGA 是基于查找表的，在综合后，将前面的逻辑运算转化为了查找表结构，输入输出都需要经过缓冲区。

从图中可以看到，共有 23 个 I/O 端口，我们进行简单的计算，9 位二进制数，一个时钟输入，一个清零端输入，3 组 4 位 BCD 码，共

23 位，与设计相符合。

选中其中一个元件，观察它的真值表，如选中 LUT5



查找表的真值表为

Properties

ten[3]_i_5

$0 = !I2 \& !I3 \& !I4 + I2 \& I3 \& !I4 + !I0 \& !I1 \& I2 \& !I3 \& I4 + !I1 \& !I2 \& I3 \dots$

I4	I3	I2	I1	I0	
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	1
1	1	1	1	1	1

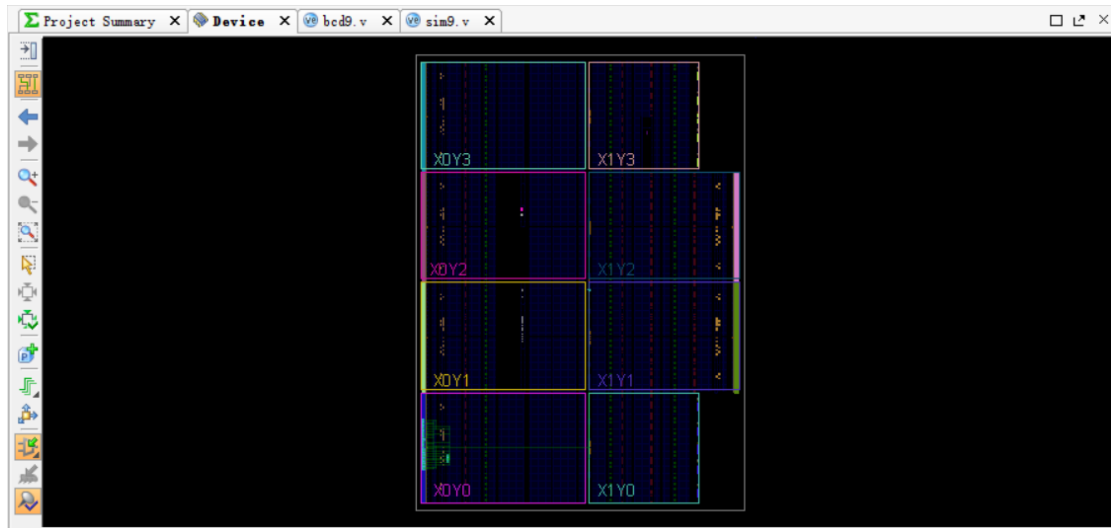
Edit LUT Equation...

General | Properties | Power | Nets | Cell Pins | **Truth Table**

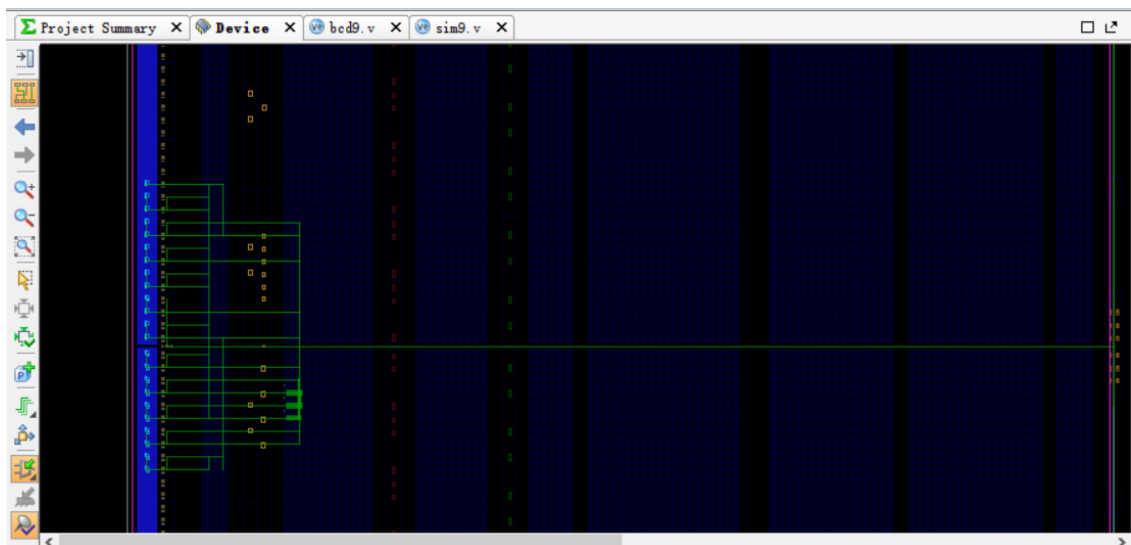
即可方便我们分析逻辑功能。

设计实现

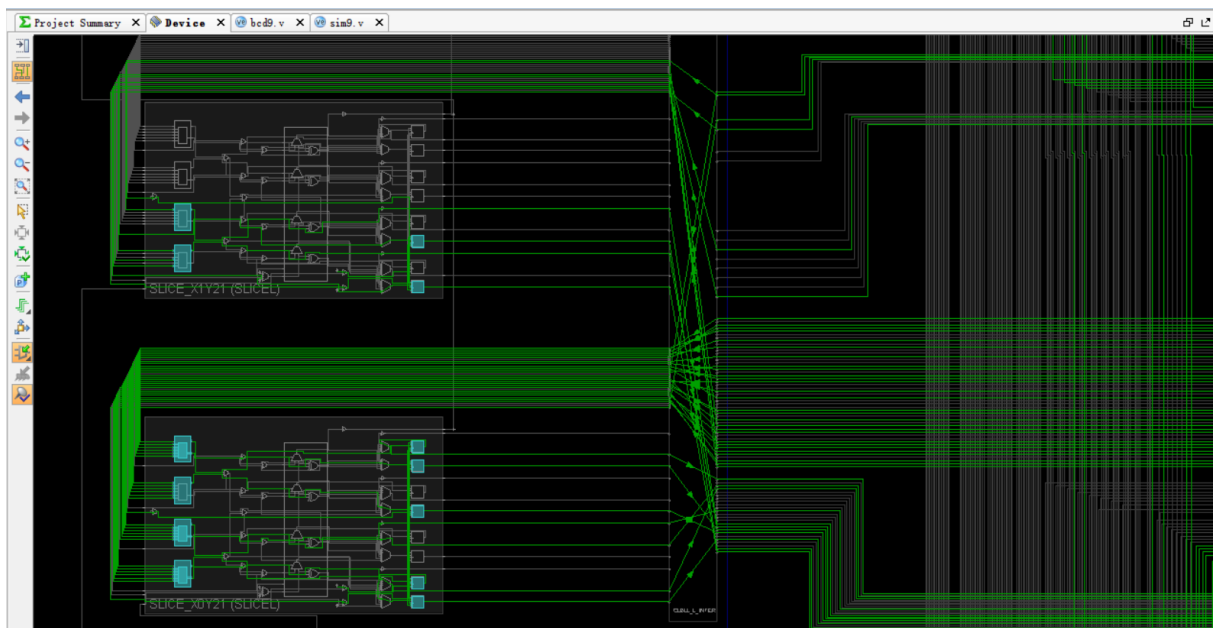
(1) Device 视图（显示布局布线）



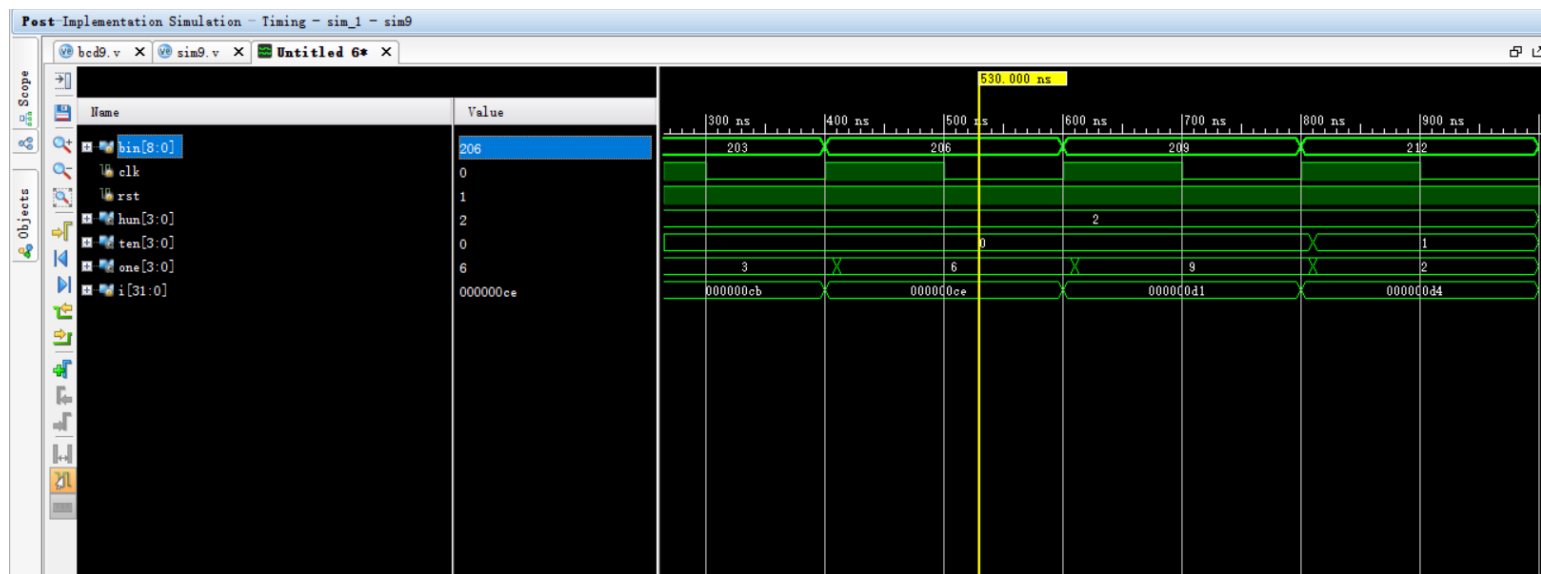
放大后观察已使用的 I/O 资源



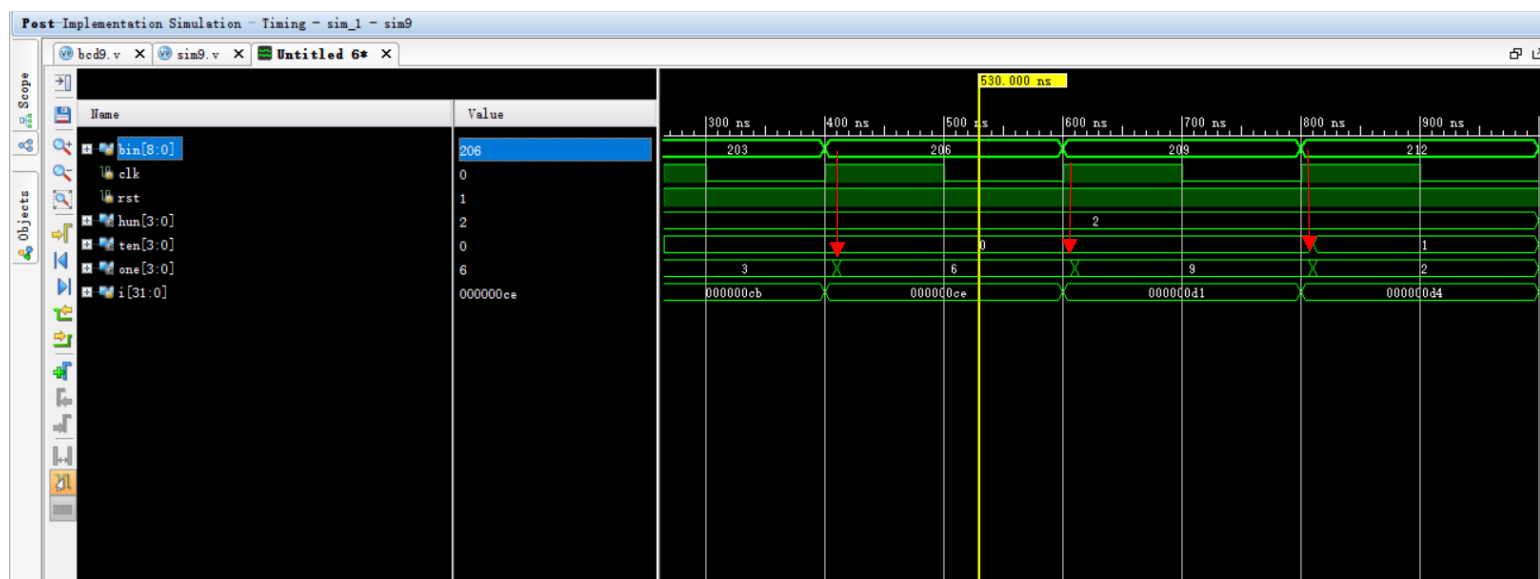
放大即可看到 SLICE 内部单元



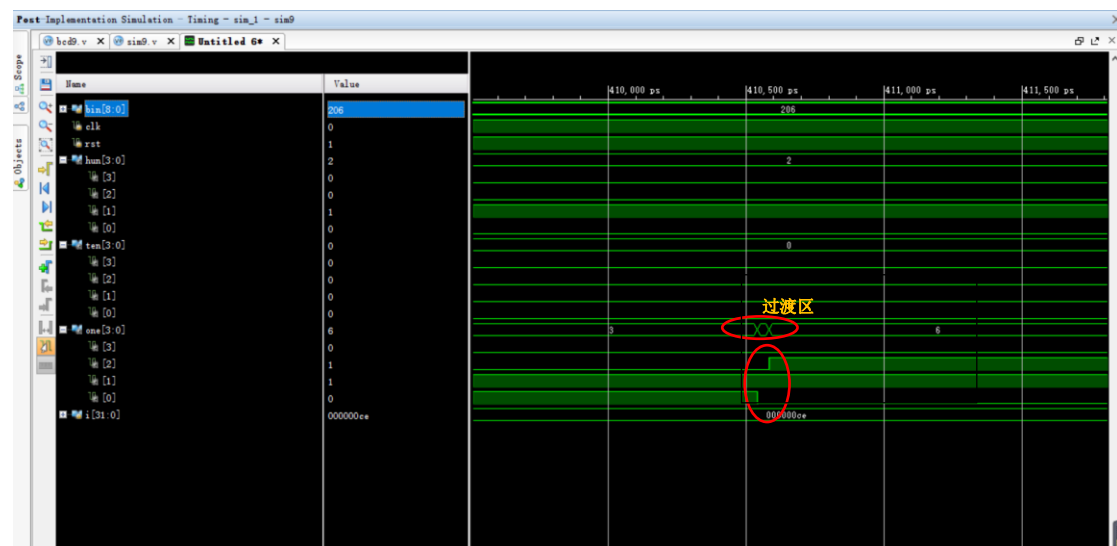
实现后的时序仿真（运行布局布线后的仿真）



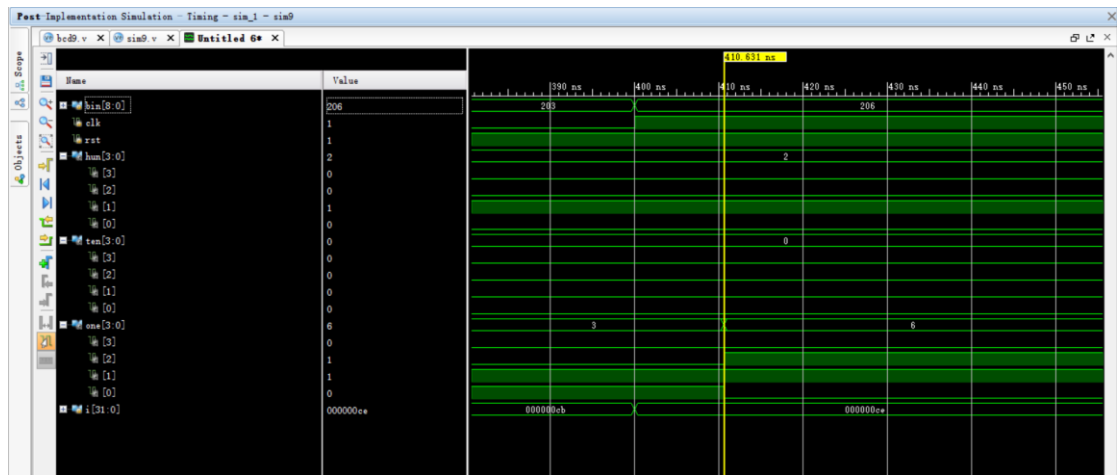
可以看到图中标红处有明显的类似毛刺的现象且与时钟上升沿相比，存在明显的延时。



放大毛刺所在处，并且显示出四位 BCD 码的每一位二进制数的变化，可以发现，同一个 BCD 码内部的二进制数的变化时间其实也是不同的，存在 ps 级别的差异，如图所示，因此，造成了 BCD 码处于一个过渡区，是一个不确定状态，缩小后则会形成近似毛刺的图形。

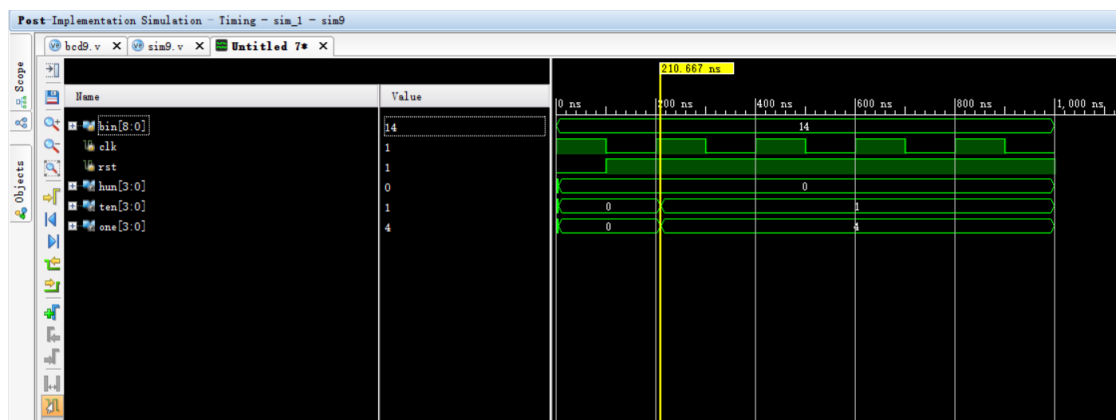


测量延时：



即大约 10.6ns，即注意时钟输入的周期，如果比 10.6ns 更快，D 触发器会产生错误的结果，可以看出，延时对 FPGA 整个系统的工作速度产生重要影响。

上面是在循环中观察输入与输出的关系，也可以单独输入一个数，进行测试和观察：



可见，结果也存在延时，且也在 10.6ns 左右。但延时结束后的输出仍然是正确的。

四．拓展设计

实现 16 位二进制数到 BCD 码的转换

原理和实现方法都与 9 位二进制数转换为 BCD 码类似，但由于

16 位二进制数的最大值为 65536，所以需要五组四位 BCD 码，分别代表万位、千位、百位、十位、个位，即共 20 位 BCD 码。所以可知：

输入：时钟沿、清零信号、16 位二进制数

输出：五组四位 BCD 码

由于需要将 16 位二进制数和 20 位 BCD 码都放入一个大寄存器数组中，所以需要有一个 36 位的寄存器类型数组。

Verilog 语言描述

Design Source 设计

```
bed16.v
C:/Users/asus/Desktop/2017040481/project_1/project_1.srcs/sources_1/new/bed16.v
20 //////////////////////////////////////////////////
21 module bed16(clk, bin, rst_n, one, ten, hun, tho, tth
22 );
23 input  [15:0] bin;
24 input  clk;
25 input  rst_n;
26 output reg[3:0] one;
27 output reg[3:0] ten;
28 output reg[3:0] hun;
29 output reg[3:0] tho;
30 output reg[3:0] tth;
31 reg[35:0] out;
32 function [35:0] result;
33 input  [35:0] register;
34 integer i;
35 begin
36     result=register;
37     for(i=0;i<=15;i=i+1)
38         begin
39 /*
40 if(result[35:32]>4)
41     result[35:32]=result[35:32]*3;
42 else
43     result[35:32]=result[35:32];
44 if(result[31:28]>4)
45     result[31:28]=result[31:28]*3;
46 else
47     result[31:28]=result[31:28];
48 if(result[27:24]>4)
49     result[27:24]=result[27:24]*3;
50 else
51     result[27:24]=result[27:24];
52 if(result[23:20]>4)
53     result[23:20]=result[23:20]*3;
54 else
55     result[23:20]=result[23:20];
56 if(result[19:16]>4)
57     result[19:16]=result[19:16]*3;
58 else
59     result[19:16]=result[19:16];

```

Elaborated Design - xc7a75tfgg484-1 (active)

Project Summary X Schematic X sim16.v X bcd16.v X

C:/Users/asus/Desktop/2017040481/project_1/project_1.srcs/sources_1/new/bcd16.v

```
56 if(result[19:16]>4)
57   result[19:16]=result[19:16]*3;
58 else
59   result[19:16]=result[19:16];
60   */
61   result[35:32] = ( result[35:32] > 4) ? (result[35:32] + 3) : result[35:32];
62   result[31:28] = ( result[31:28] > 4) ? (result[31:28] + 3) : result[31:28];
63   result[27:24] = ( result[27:24] > 4) ? (result[27:24] + 3) : result[27:24];
64   result[23:20] = ( result[23:20] > 4) ? (result[23:20] + 3) : result[23:20];
65   result[19:16] = ( result[19:16] > 4) ? (result[19:16] + 3) : result[19:16];
66   result=result<<1;
67 end
68 end
69 endfunction
70 always @ (posedge clk or negedge rst_n)
71 begin
72 if (!rst_n)
73   out=36'b0;
74 else
75 begin
76   out={20'b0,bin};
77   out=result(out);
78 end
79   one=out[19:16];
80   ten=out[23:20];
81   hun=out[27:24];
82   tho=out[31:28];
83   tth=out[35:32];
84 end
85 endmodule
86
```

Design Runs

Simulation Source 设计

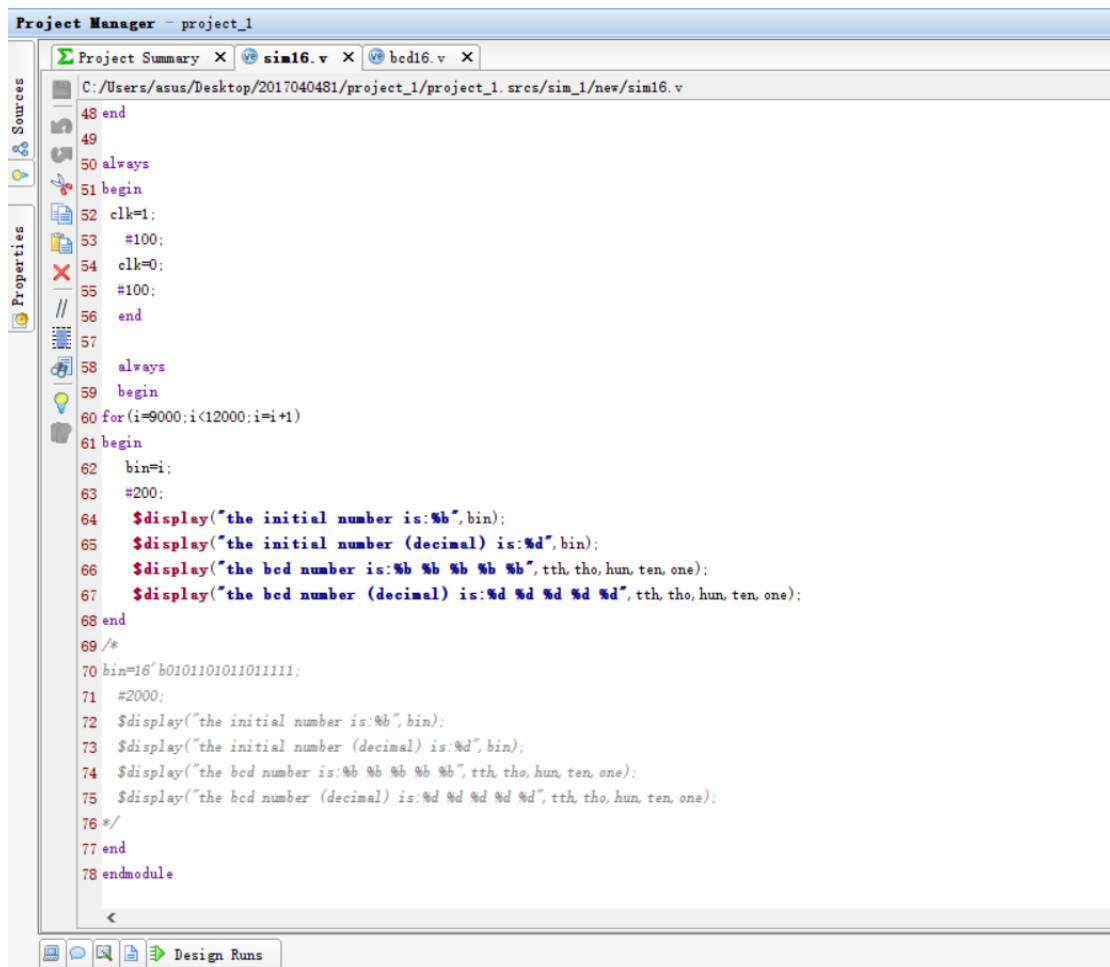
Project Manager - project_1

Project Summary X sim16.v X bcd16.v X

C:/Users/asus/Desktop/2017040481/project_1/project_1.srcs/sim_1/new/sim16.v

```
20 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
21 module sim16;
22 reg [15:0] bin;
23 reg clk;
24 reg rst_n;
25 wire [3:0] tth;
26 wire [3:0] tho;
27 wire [3:0] hun;
28 wire [3:0] ten;
29 wire [3:0] one;
30 integer i;
31 bcd16 uut(
32   .clk(clk),
33   .rst_n(rst_n),
34   .bin(bin),
35   .tho(tho),
36   .tth(tth),
37   .hun(hun),
38   .ten(ten),
39   .one(one)
40 );
41
42 initial
43 begin
44   rst_n=0;
45   clk=1;
46   #10;
47   rst_n=1;
48 end
49
50 always
51 begin
52   <
```

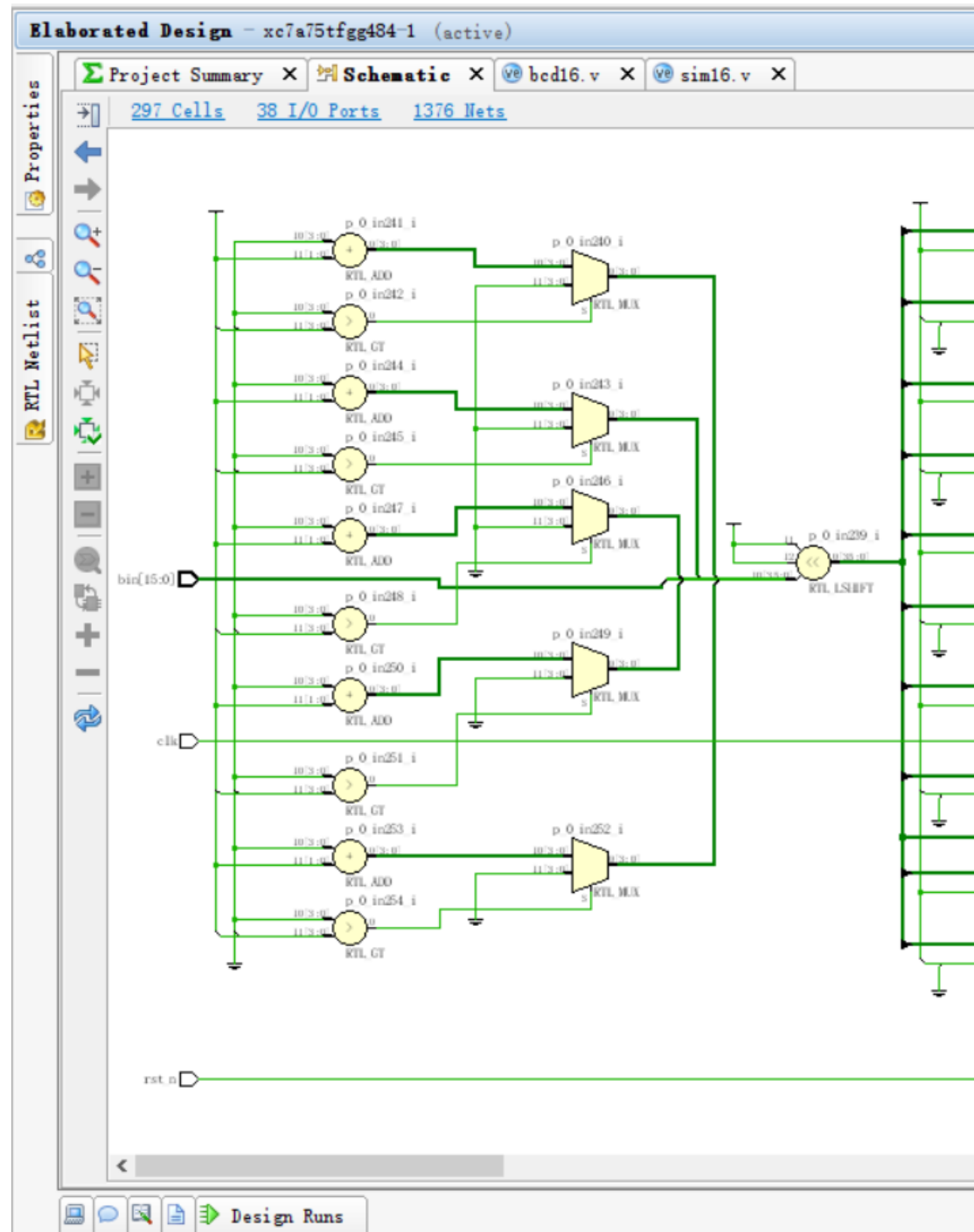
Design Runs

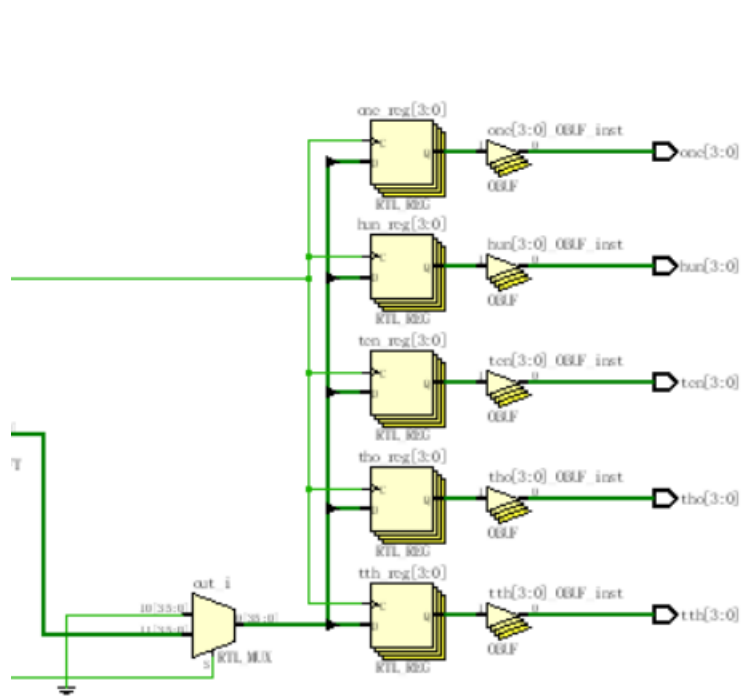


RTL 详细描述



放大后可看输入输出端口：





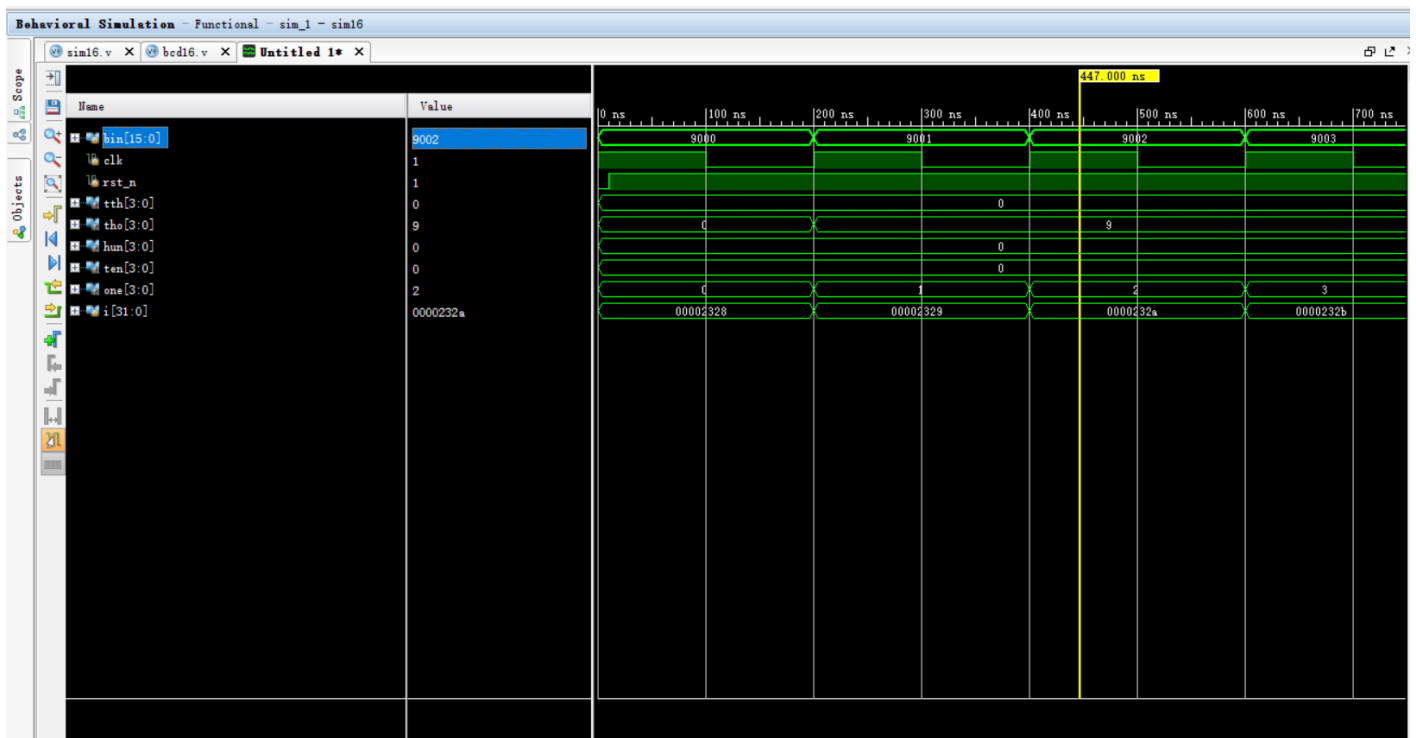
输入端口为时钟和清零端，以及一个 16 位的二进制数。

输出端口为五组四位 BCD 码。

显示共有 38 个输入输出端口，进行简单计算：时钟和清零端以及 16 位二进制数和 5 组四位的 BCD 码即 20 位 BCD 码，一共 38 个输入输出端口。

从中间的结构中的“+”“>”可以看出，它的功能为每次移位前分别对代表万位、千位、百位、十位、个位的 BCD 码进行校准。所以根据原理图，可以这样描述它的功能：将二进制数、时钟、清零信号输入，对代表万位、千位、百位、十位、个位的四位 BCD 码分别进行判断，若大于 4，则执行加 3 操作，判断完毕后，将它们和二进制数放在一起，并进行左移一位的操作。重复上述操作十六次，由此分析，原理图和所写的 verilog 语言所描述的功能是相同的。因此设计的逻辑是正确的。

行为级仿真



如图，16 位二进制数的变化周期是 200ns，而时钟以 100ns 翻转一次，即 200ns 时钟周期。每遇到一个时钟上升沿，进行一次从 16 位二进制数到 BCD 码的转化。

在 Tcl Console 中可以看到

```
# run 1000ns
the initial number is:0010001100101000
the initial number (decimal) is: 9000
the bcd number is:0000 0000 0000 0000 0000
the bcd number (decimal) is: 0 0 0 0 0
the initial number is:0010001100101001
the initial number (decimal) is: 9001
the bcd number is:0000 1001 0000 0000 0001
the bcd number (decimal) is: 0 9 0 0 1
the initial number is:0010001100101010
the initial number (decimal) is: 9002
the bcd number is:0000 1001 0000 0000 0010
the bcd number (decimal) is: 0 9 0 0 2
the initial number is:0010001100101011
the initial number (decimal) is: 9003
the bcd number is:0000 1001 0000 0000 0011
the bcd number (decimal) is: 0 9 0 0 3
the initial number is:0010001100101100
the initial number (decimal) is: 9004
the bcd number is:0000 1001 0000 0000 0100
the bcd number (decimal) is: 0 9 0 0 4
```

取其中一个数据

```
the initial number is:0010001100101001
the initial number (decimal) is: 9001
the bcd number is:0000 1001 0000 0000 0001
the bcd number (decimal) is: 0 9 0 0 1
```

即输入的 16 位二进制数为 0010 0011 0010 1001

对应的十进制数为 9001

输出的 5 组 4 位 BCD 码为 0000 1001 0000 0000 0001

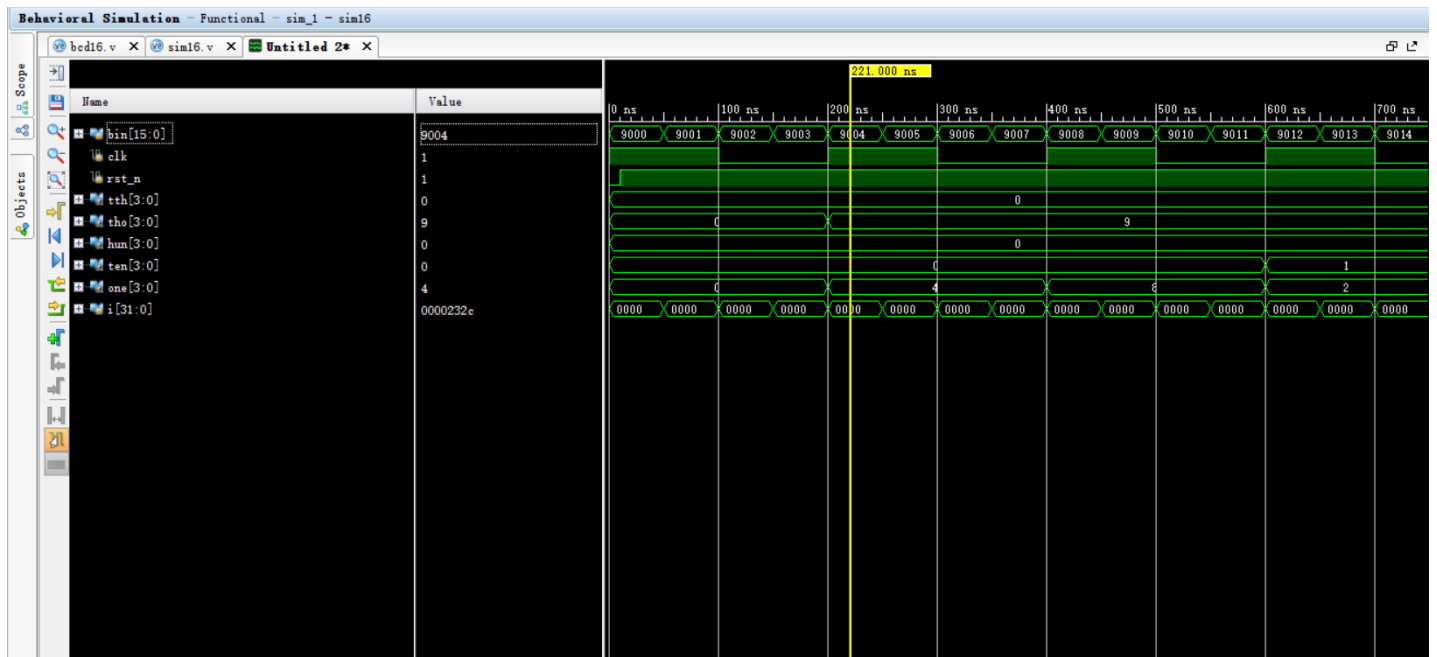
转化为十进制数，对应的万位、千位、百位、十位、个位分别为

0 9 0 0 1

所以这种转化是正确的。

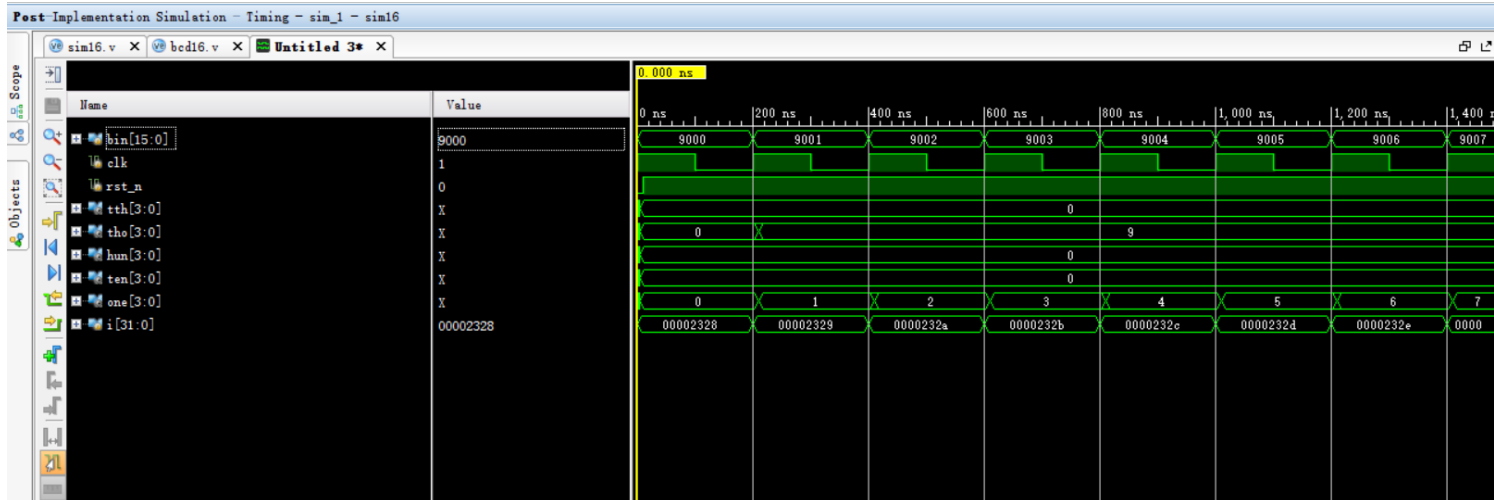
为了观察时钟的上升沿对这种转化的影响，将数据的变化周期设置为

小于时钟周期的数

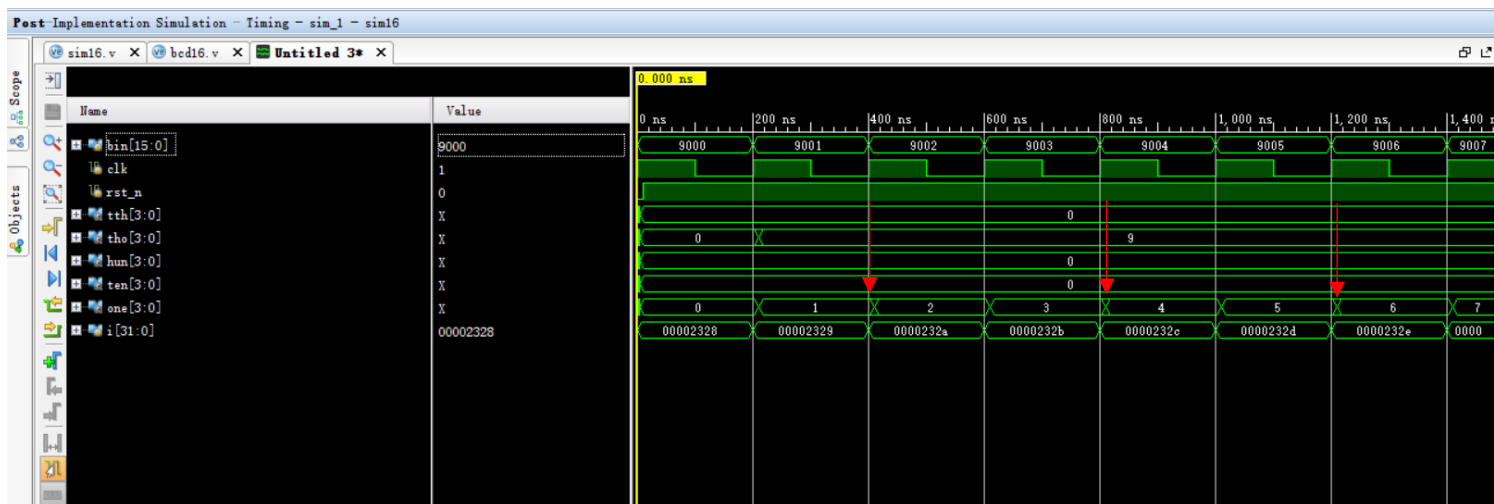


可以看见，当时钟上升沿没有到时，即使二进制数发生了改变，输出也不会发生改变。在图中光标处，可以看到，输入的二进制数转化为十进制数等于 9004，而输出的万位、千位、百位、十位、个位分别为 0 9 0 0 4；所以这个过程是正确的。

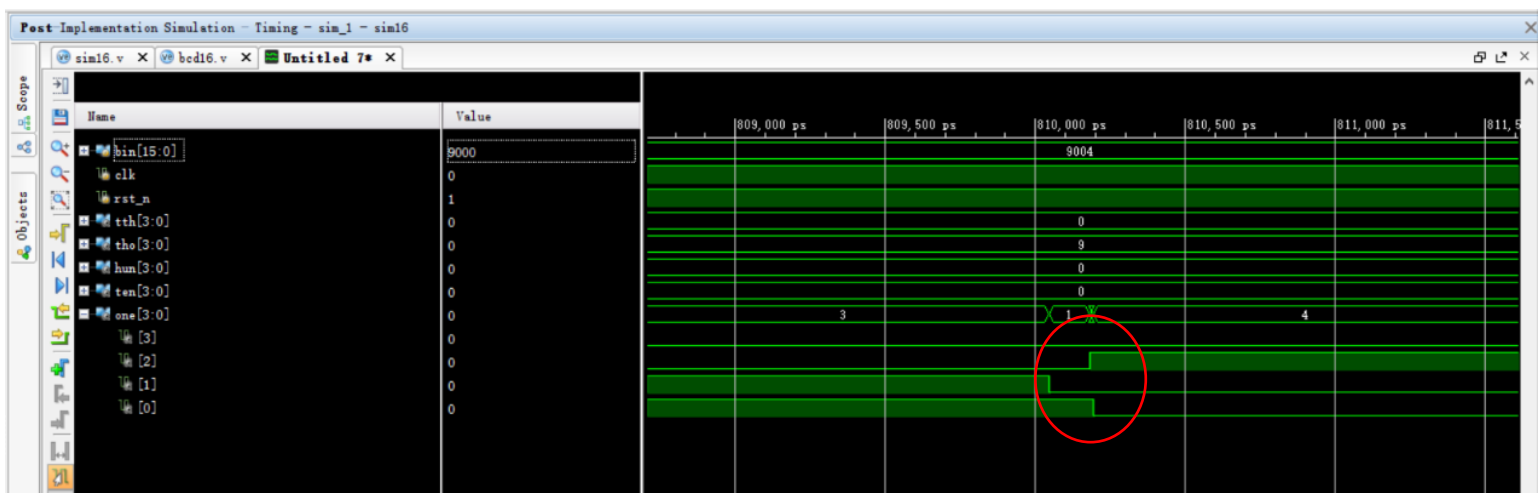
时序仿真



可以明显看到，图中标红处存在类似毛刺

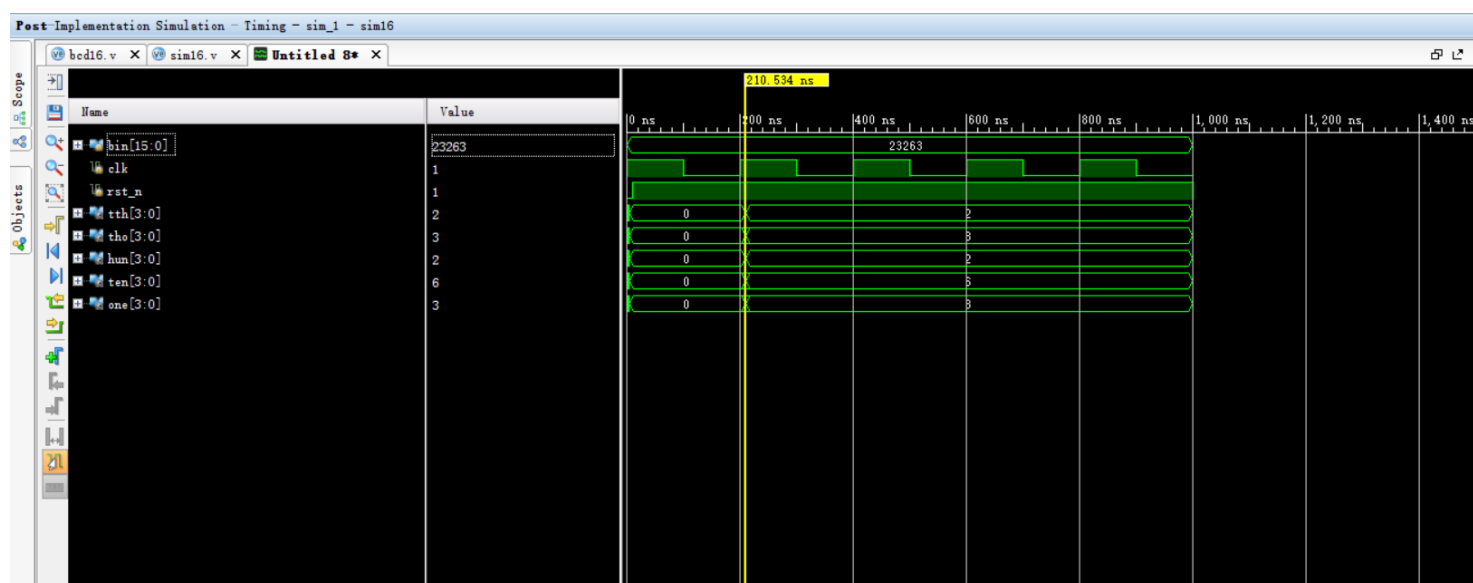


放大后可以看到



每一个 BCD 码内部的四个二进制数的翻转时间有所差异，差异在 ps 级，这也导致了 BCD 码的过渡带，即为不确定状态。如图所示，这个状态所得到的 BCD 码是错误的，缩小后即形成类似毛刺的图案。

单独测试一个 16 位二进制数：



第一个时钟的上升沿来时，由于清零端为 0，输出为 0，所以必须等到第二个时钟脉冲来时，才能进行从 16 位二进制数向 BCD 码的转换。也存在延时，由图中光标可测量，与 9 位二进制数的延时时间基本相同。

五. 总结和反思

在这次设计中，了解了二进制数和 BCD 码之间的转换关系，实践了使用 verilog 语言编写 design source 和 simulation source 文件，也实践了函数的声明和调用。还按照何老师的 MOOC 视频讲解，进行了一遍从 RTL 详细描述到行为级仿真，到设计综合，再到设计实现

和实现后的时序仿真的完整过程，了解了设计的流程。同时，也体会到了延时对于 FPGA 的影响，了解了时序问题的重要性。