东南大学

《数字逻辑与计算机体系结构（含实验）I》

实验报告

实验八　6位BCD计数/时钟显示FPGA系统

姓　　名： 李睿刚　**学　　号：** 61821326

同　　组： 无　**学　　号：** 无

专　　业： 工科试验班　**实 验 室：** 计算机硬件技术

实验时间： **2022** 年 **12** 月 **12** 日　**报告时间：** **2022** 年 **12** 月 **14** 日

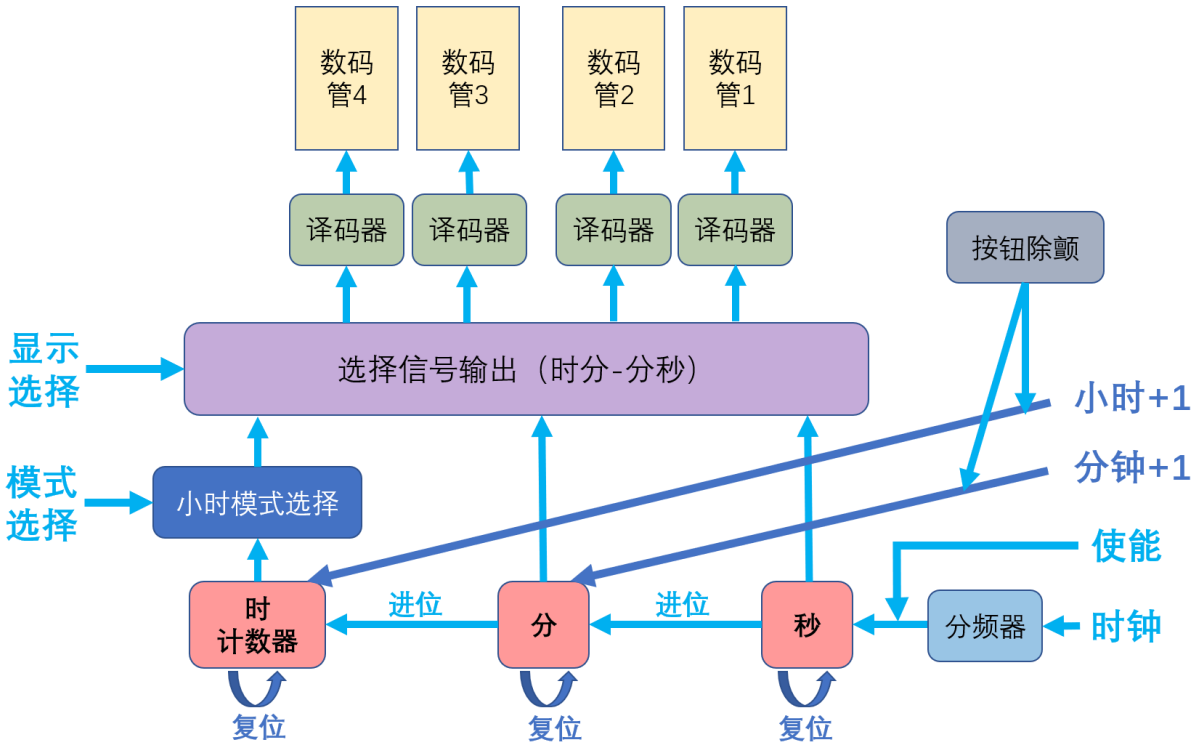
评定成绩：　**审阅教师：**

一. 实验目的与内容

1. 通过综合性 HDL 设计考察检查评估掌握典型组合逻辑与时序逻辑电路基础知识、FPGA 数字模块和应用基本知识、设计及实验方法的能力情况。
2. 参考电子时钟电路设计原理，结合实验 5-7 所掌握的 FPGA 编程知识和设计方法，搭建一 FPGA 控制的时分秒显示的 6 位（4 位）电子钟（4 位对应时-分或分-秒可切换），研究拓展利用（LPM 模块）库资源技术，完成基于 Verilog 模块或 IP（或 LPM 模块）的 FPGA 仿真和设计实现，掌握复杂数字系统 FPGA 的基本设计、仿真和调试方法。
3. 针对 2MHz 输入（或 32.768KHz，或分频后 1Hz）时钟，实现六位（4 位对应时-分或分-秒可切换）的 BCD 码（8-4-2-1）指示灯和 7 段数码管显示；有时分秒（暂停）设置功能；有必要的切换等其他功能（鼓励）；
4. 要求形成明确的原理图设计和完整的仿真测试方案和仿真测试模块；要求采用计数器模块实例化方法实现（不允许采用单一 HDL 模块编写）；至少完成实用有效的仿真（即测试模拟验证覆盖全部时分秒进位效果）*
5. 鼓励利用 IP（LPM）模块技术；
6. 鼓励把自行开发的模块 IP 化并应用验证（选做提高）。

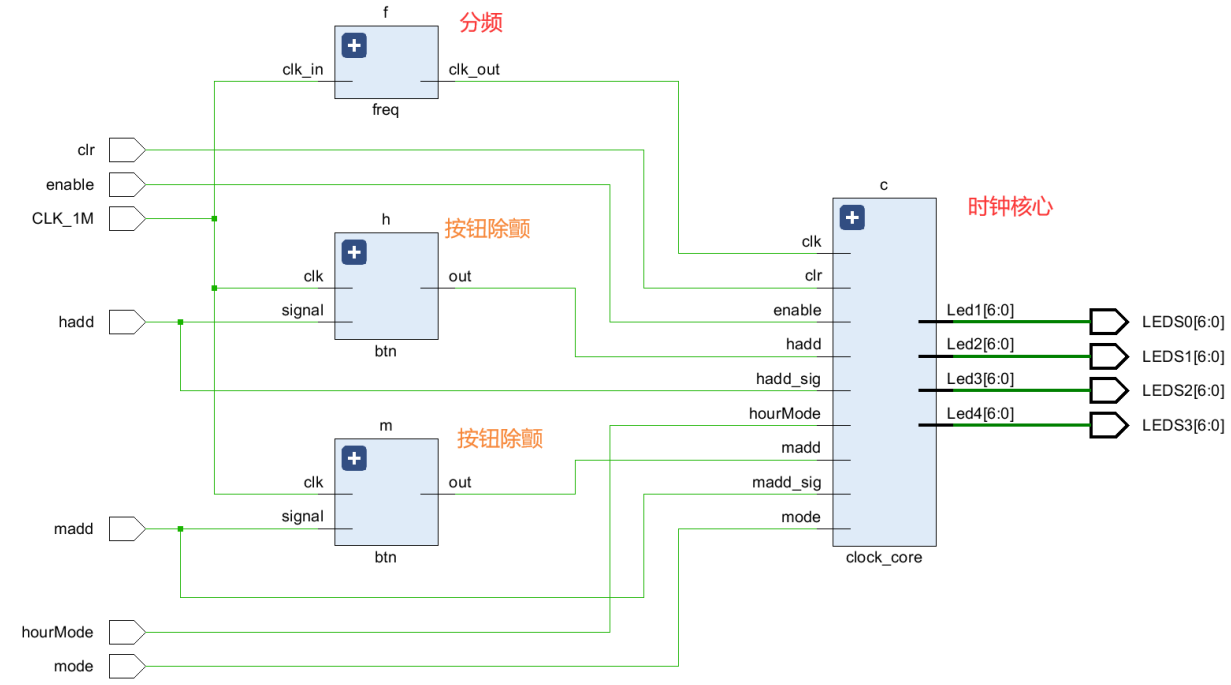
二. 方案实现

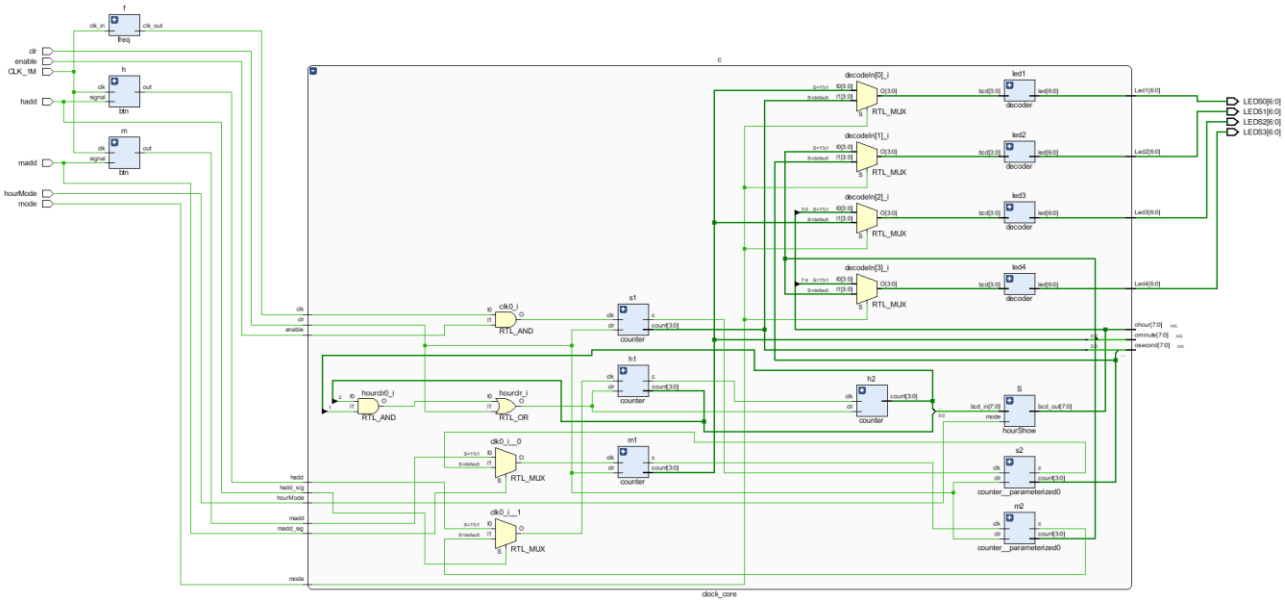
准备自己从头开始写，完成小时切换、四位显示模式、时分设置、按钮除颤四个额外功能。程序构架如下图所示：



需要编写“译码器”“计数器”“分频器”“小时模式选择”“按钮除颤”五个模块，而模块之间的关系以及信号选择输出，放在整合的文件中实现。

实际综合出的电路图如下图所示：





具体代码在“四、实验代码”中全部展示。

三．测试

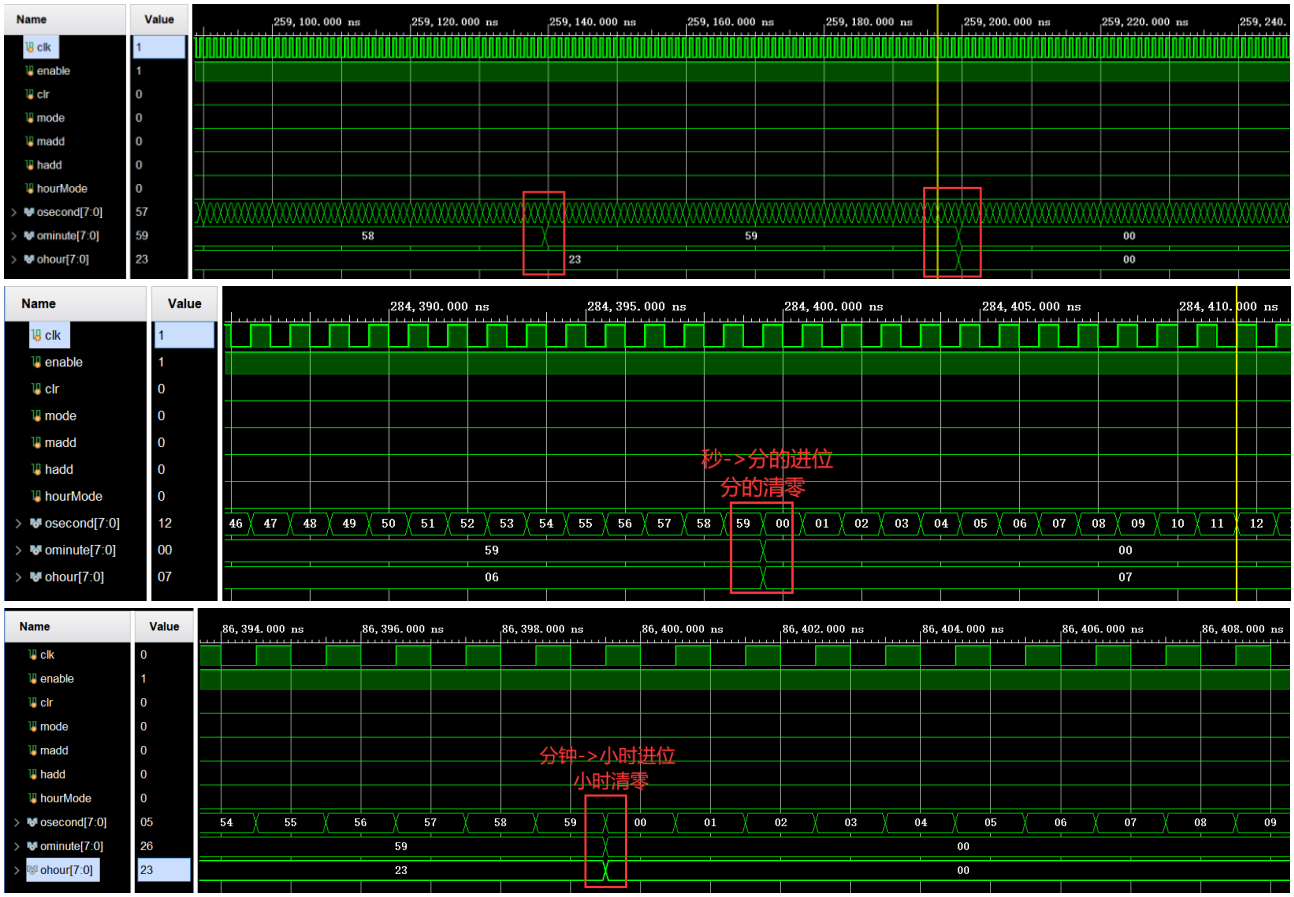
1. 测试计时功能

1) TestBench.v：一共仿真 865000ns，1ns 有一个 clk 上升沿

```
`timescale 1ns/ 1ps
module testbench();
    reg clk=0;   reg enable=1;
    reg clr=0;   reg mode=0;
```

```
    reg madd=0;  reg hadd=0;        // 分钟加 1，小时加 1
    reg hourMode=0;
    wire [7:0] osecond;             // 秒计数输出
    wire [7:0] ominute;             // 分钟计数输出
    wire [7:0] ohour;               // 小时计数输出
    // 没有引出 LED 的引脚。在测试 3 中进行相关检验。本次测试只检查计数
    clock_core
c(clk,enable,clr,mode,madd,madd,hadd,hadd,hourMode,osecond,ominute,oho
ur);
    always #0.5 clk<=~clk;          // 1ns 一周期
endmodule
```

2)　仿真结果及分析：



时分秒成功进位清零，计时功能测试成功。

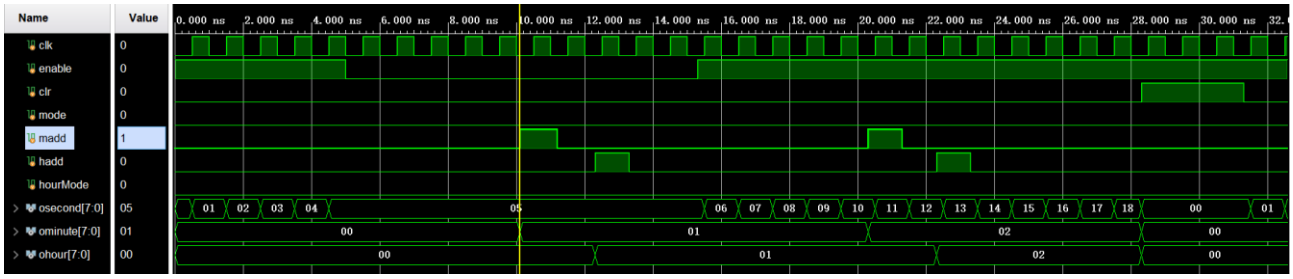
2.　测试 enable、minute_add、hour_add 功能

1)　TestBench.v：

```
`timescale 1ns/ 1ps
module testbench();
    reg clk=0;   reg enable=1;
    reg clr=0;   reg mode=0;
    reg madd=0;  reg hadd=0;        // 分钟加 1，小时加 1
    reg hourMode=0;
    wire [7:0] osecond;
```

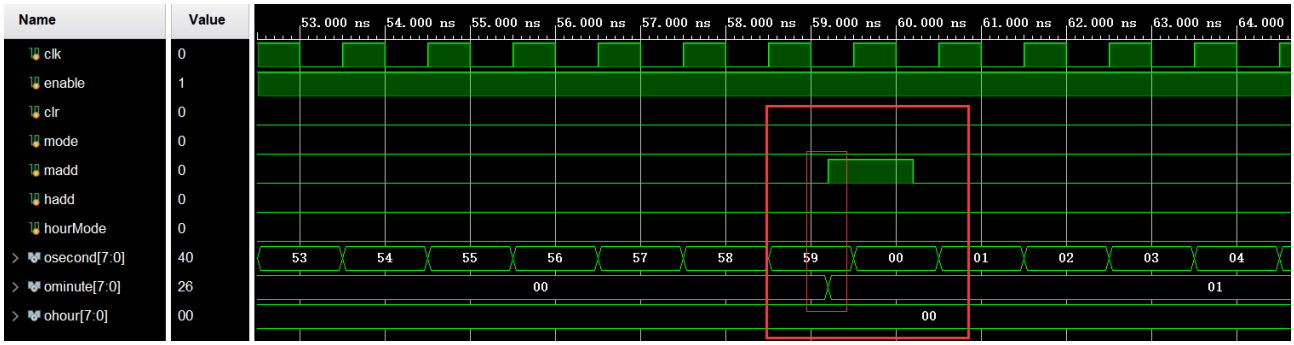
```
    wire [7:0] ominute;
    wire [7:0] ohour;
    clock_core
c(clk,enable,clr,mode,madd,madd,hadd,hadd,hourMode,osecond,ominute,oho
ur);
    initial begin
        // enable、调时间 功能测试
        #5 enable = 0;
        #5.1 madd = 1;
        #1.1 madd = 0;
        #1.1 hadd = 1;
        #1 hadd = 0;

        #2 enable = 1;
        #5 madd = 1;
        #1 madd = 0;
        #1 hadd = 1;
        #1 hadd = 0;
        // clear 功能测试
        #5 clr = 1;
        #3 clr = 0;
    end
    always #0.5 clk<=~clk;
endmodule
```

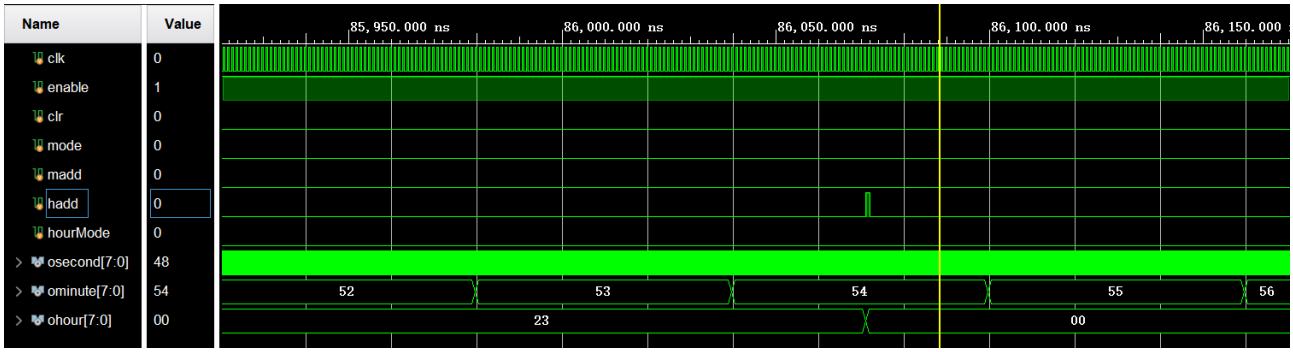
2)　仿真结果及分析：



enable=0 时，秒计数不再发生变化，时钟启动、暂停功能测试成功；

不管在 enable 为 0 还是 1 的情况下，分钟计数、小时计数都会在 madd、hadd 上升沿加一。为了验证加一过程中清零、进位是否正常，又进行了以下测试（testbench 文件略）：





如上两图，进位清零正常。在第一张图中，分钟 add 信号持续高电平时忽略了来自秒的进位信号。实际解释为：按钮没松开都视为人在操作，此时应该仅有人的行为可以改变被控制单位。相关说明见“六、分析与总结 4”。

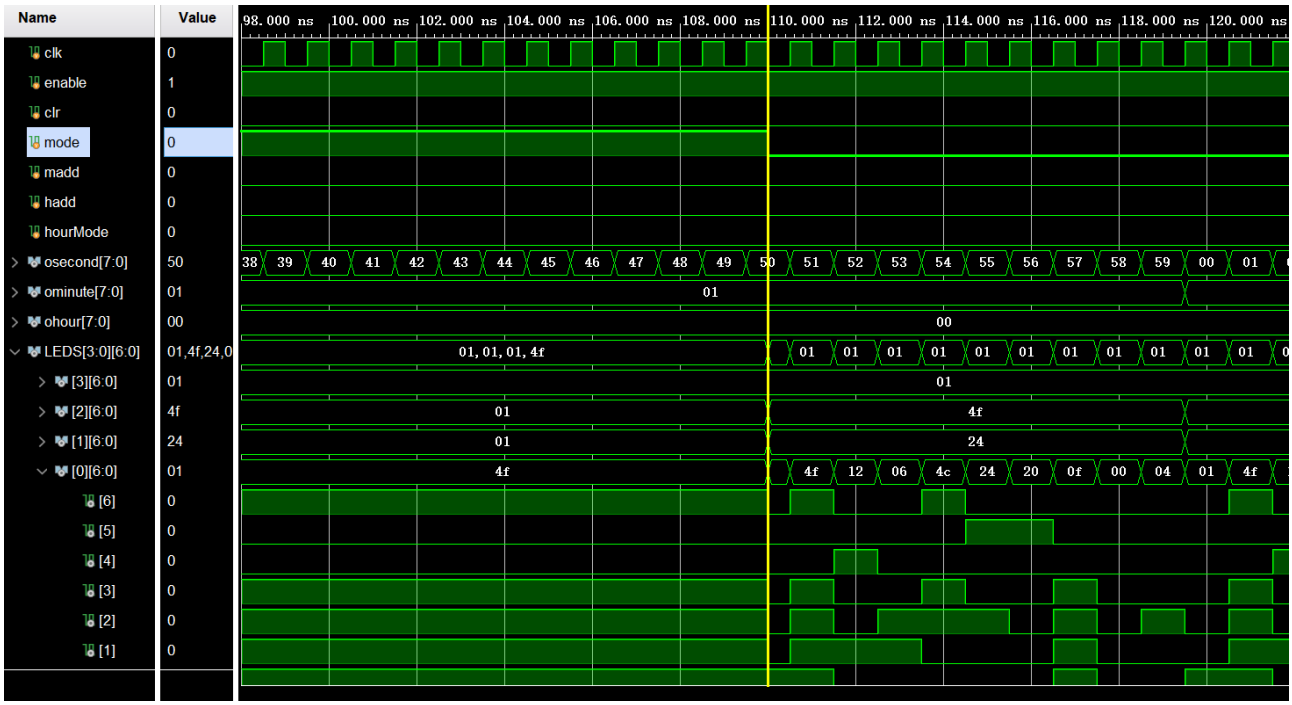
3. 测试时分、分秒、小时模式切换

1) TestBench.v：

```
`timescale 1ns/ 1ps
module testbench();
    reg clk=0;   reg enable=1;
    reg clr=0;   reg mode=0;
    reg madd=0;  reg hadd=0;
    reg hourMode=0;
    wire [7:0] osecond;
    wire [7:0] ominute;
    wire [7:0] ohour;
    wire [6:0] LEDS[3:0];
    clock_core
c(clk,enable,clr,mode,madd,madd,hadd,hadd,hourMode,osecond,ominute,oho
ur,LEDS[0],LEDS[1],LEDS[2],LEDS[3]);
    initial begin
          #10 mode = 1;                 // 数码管显示切换
          #100 mode = 0;
          #72000.1 hourMode = 1;        // 12<->24 小时切换
          #66000 hourMode = 0;
    end
    always #0.5 clk<=~clk;
endmodule
```

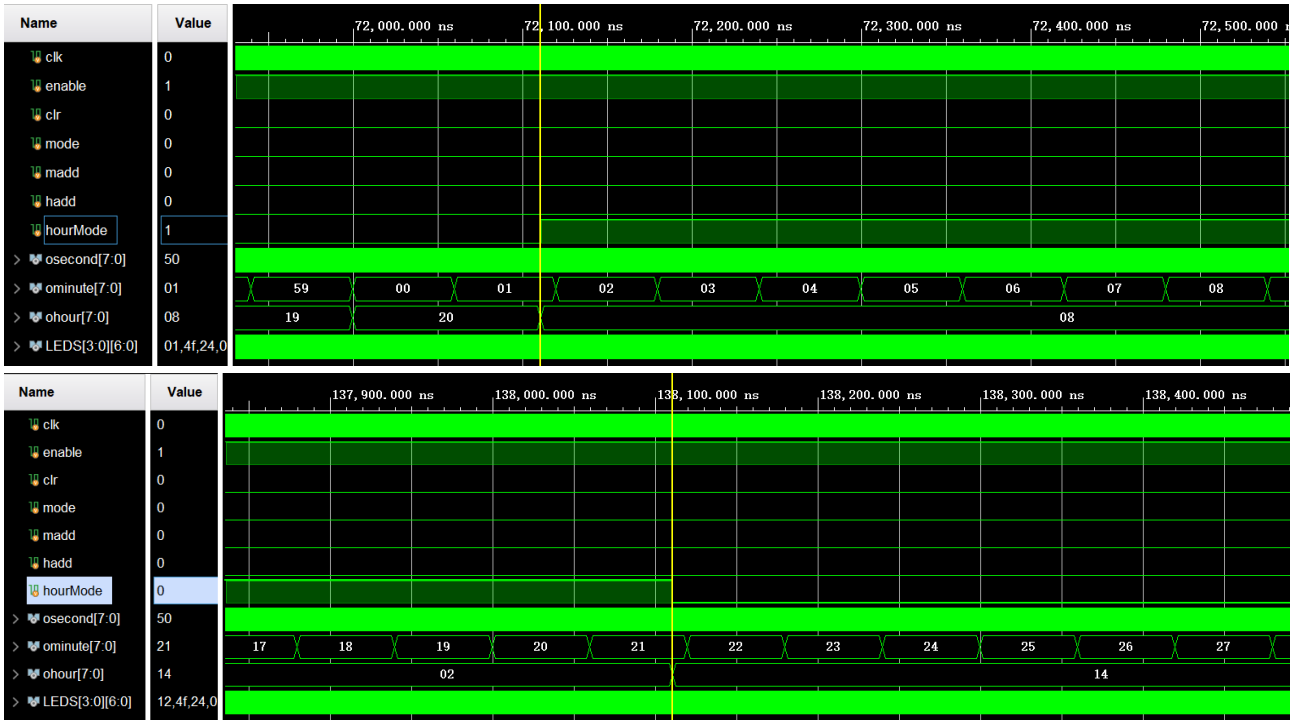
2) 仿真结果及分析：

先是“时分”↔“分秒”测试：



时、分切换如上两张图，显示了 mode 切换的两个过程。第一张图，mode 由 0 变 1，数码管从“分秒”变为“时分”，由图可知切换成功；第二张图 mode 恢复 0，同样成功。

接下来是 12↔24 小时测试：



以上两张图显示了 hourMode 切换的两个时刻的状态。关注“ohour”一行，确实在 hourMode=1 时变成了 12 进制的小时计数方式。

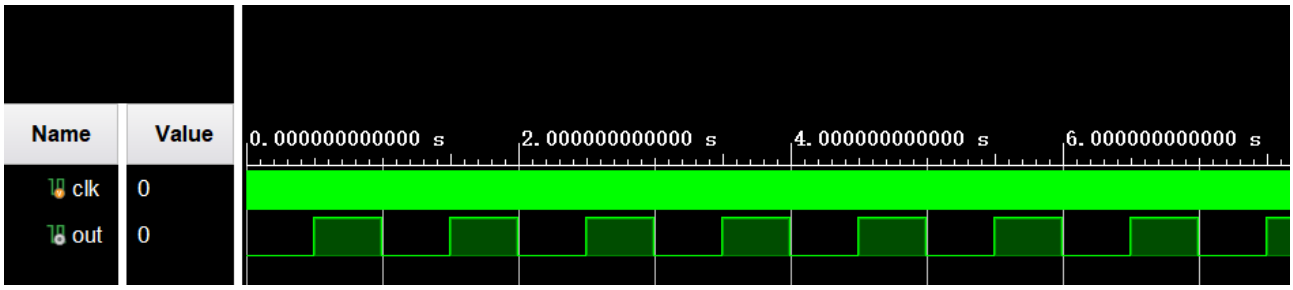
以上，时钟计数、修改小时分钟、停止时钟、4 位显示、小时显示方式的测试结果都符合预期。

4. 测试分频功能

1) TestBench.v：

```
`timescale 1us / 1ps
module testbench();
    reg clk=0;
    wire out;
    freq f(clk,out);
    always #0.5 clk=~clk;
endmodule
```

2) 仿真结果及分析：



一秒内 clk 反转一次，实现了一次计数。

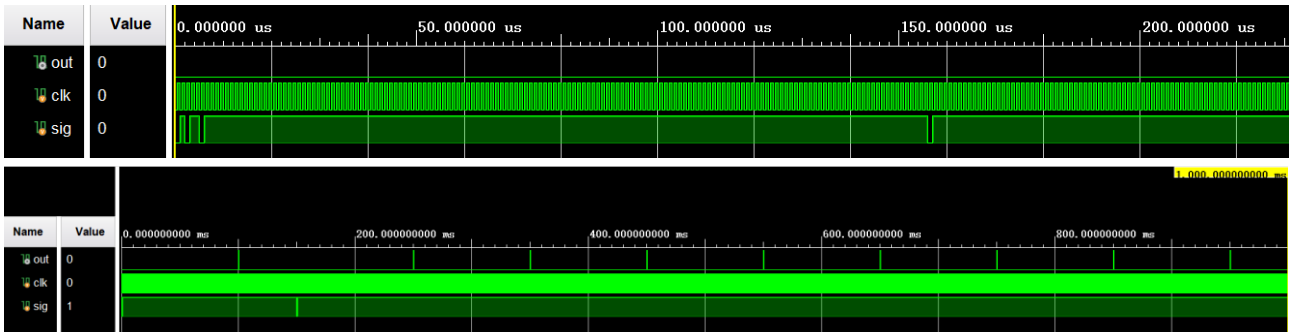
5. 测试按钮除颤功能

1) Testbench.v:

```
module testbench();
    wire out=0;
    reg clk = 0;
```

```
    reg sig=0;
    btn b(clk,sig,out);
    initial begin
        #1 sig=1;
        #1 sig=0;
        #1 sig=1;
        #2 sig=0;
        #1 sig=1;
        #150 sig = 0;
        #1 sig = 1;
        #150000 sig = 0;
        #1 sig = 1;
    end
    always #0.5 clk<=~clk;
endmodule
```

2)　仿真结果及分析：



按钮按下每满 0.1s 会发出一次信号。如果一直按着，也会以此频率发出信号。发出的信号宽度同 clk 信号宽度。前几微秒的扰动并没有使 out 为 1，除颤成功。约 150ms 时短暂“松开”按钮，成功模拟了一次触发。之后长按，成功实现自动循环触发。

四．实验代码

计数器：

```
`timescale 1ns/ 1ps
// bcd 计数器
module counter
    #(parameter max = 10)
    (
        input clk,
        input clr,
        output reg [3:0] count,
        output reg c        // 进位
    );

    initial begin   // 初始化置零
        count<=0;c<=0;
```

```
    end

    always @(posedge clk or posedge clr) begin
        if (clr) begin
            count<=0;c<=0;
        end
        else begin
            if (count >= max-1) begin
                count<=0;c<=1;
            end
            else begin
                count<=count+1;c<=0;
            end
        end
    end
endmodule
```

译码器：

```
`timescale 1ns / 1ps
// 7 段数码管译码器
module decoder (
        input [3:0] bcd,
        output reg [6:0] led
    );
    always @(bcd) begin
        case(bcd)
            0:  led=7'b0000001;
            1:  led=7'b1001111;
            2:  led=7'b0010010;
            3:  led=7'b0000110;
            4:  led=7'b1001100;
            5:  led=7'b0100100;
            6:  led=7'b0100000;
            7:  led=7'b0001111;
            8:  led=7'b0000000;
            9:  led=7'b0000100;
            default:    led=7'b0000001;
        endcase
    end
endmodule
```

24 小时转 12 小时：

```
`timescale 1ns/ 1ps
module hourShow(
        input mode,         //1 为 12 输出，0 为 24 输出
        input [7:0] bcd_in,
```

```
        output reg [7:0] bcd_out
    );
    always @(*) begin
        if(mode && bcd_in > 8'b00010010) begin
            if(bcd_in[3:0]>=4'b0010) bcd_out<=bcd_in-8'b00010010;
            else bcd_out=bcd_in-8'b00011000;
        end
        else bcd_out <= bcd_in;
    end
endmodule
```

时钟核心：

```
`timescale 1ns/ 1ps
// 时钟
module clock_core(
        input clk,
        input enable,   // 1 使能，0 禁用
        input clr,      // 1 清零
        input mode,     // 高电平显示时分，低电平显示分秒
        input madd,     // 分钟加 1
        input madd_sig, // 是否正在加一
        input hadd,     // 小时加 1
        input hadd_sig, // 是否正在加一
        input hourMode, // 小时输出模式 1 为 12 制
        // 仿真中展示每个计数器的输出 可删除
        output [7:0] osecond,
        output [7:0] ominute,
        output [7:0] ohour,
        // 四个 LED
        output [6:0] Led1,
        output [6:0] Led2,
        output [6:0] Led3,
        output [6:0] Led4
    );
    // 计数器维护
    wire [3:0] cout [5:0];  // 每个计数器的 bcd 输出
    wire carry [4:0];       // 计数器进位
    wire hourclr = (cout[4][2]&cout[5][1])|clr; // 小时清零(满 24 清零
    counter #(10)   s1(clk&enable, clr, cout[0], carry[0]);
    counter #(6)    s2(carry[0], clr, cout[1], carry[1]);
    counter #(10)   m1(madd_sig?madd:carry[1], clr, cout[2], carry[2]);
    counter #(6)    m2(carry[2], clr, cout[3], carry[3]);
    counter #(10)   h1(hadd_sig?hadd:carry[3], hourclr, cout[4],
carry[4]);
```

```
    counter #(10)   h2(carry[4], hourclr, cout[5]);

    // 小时模式
    wire [7:0] HOUR;
    hourShow S(hourMode,{cout[5],cout[4]},HOUR);

    // LED 输出
    wire [3:0] decodeIn [3:0];
    decoder led1(decodeIn[0], Led1);
    decoder led2(decodeIn[1], Led2);
    decoder led3(decodeIn[2], Led3);
    decoder led4(decodeIn[3], Led4);

    // 选择时-分 或 分-秒
    assign decodeIn[0] = mode?cout[2]:cout[0];
    assign decodeIn[1] = mode?cout[3]:cout[1];
    assign decodeIn[2] = mode?HOUR[3:0]:cout[2];
    assign decodeIn[3] = mode?HOUR[7:4]:cout[3];

    // 仿真用 可删除
    assign osecond = {cout[1],cout[0]};
    assign ominute = {cout[3],cout[2]};
    assign ohour = {HOUR[7:4],HOUR[3:0]};
endmodule
```

分频器：

```
`timescale 1ns / 1ps
// 分频器
// 1M Hz 一个上升沿，即计数 500000 次要翻转一次
module freq(
       clk_in,
       clk_out
    );
    input clk_in;
    output reg clk_out=0;
    reg [18:0] c = 0;   // 2^19 > 500000
    always @(posedge clk_in) begin
        if(c>=19'b1111010000100011111) begin    // 499999
            c<=0;
            clk_out<=~clk_out;
        end
        else c<=c+1;
    end
endmodule
```

按钮除颤：

```
`timescale 1ns/ 1ps
// 按钮除颤 按超过 0.1s 才发出信号 长按每 0.1s 发出信号
module btn(
        clk,
        signal,
        out
    );
    input clk,signal;
    output reg out;
    wire clr = ~signal;
    reg [16:0] count=0;
    initial begin
        count<=0;out<=0;
    end
    always @(posedge clk or posedge clr) begin
        if (clr) begin
            count<=0;out<=0;
        end
        else begin
            if (count >= 17'b11000011010011111) begin
                count<=0;out<=1;
            end
            else begin
                count<=count+1;out<=0;
            end
        end
    end
endmodule
```

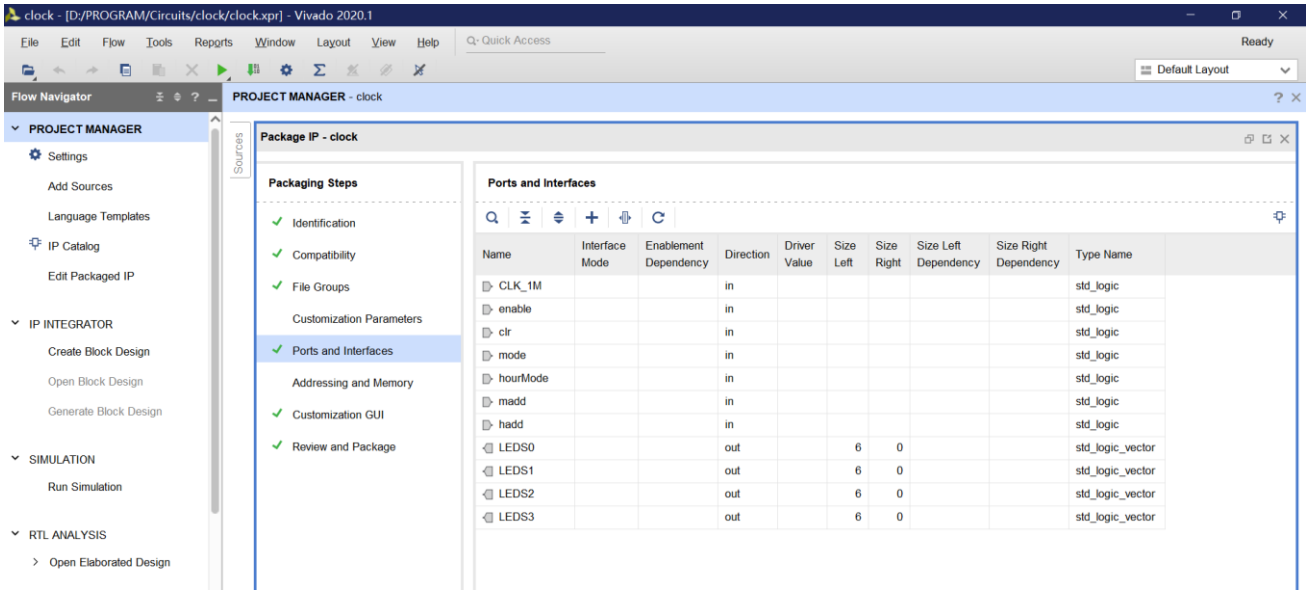
时钟（总）：

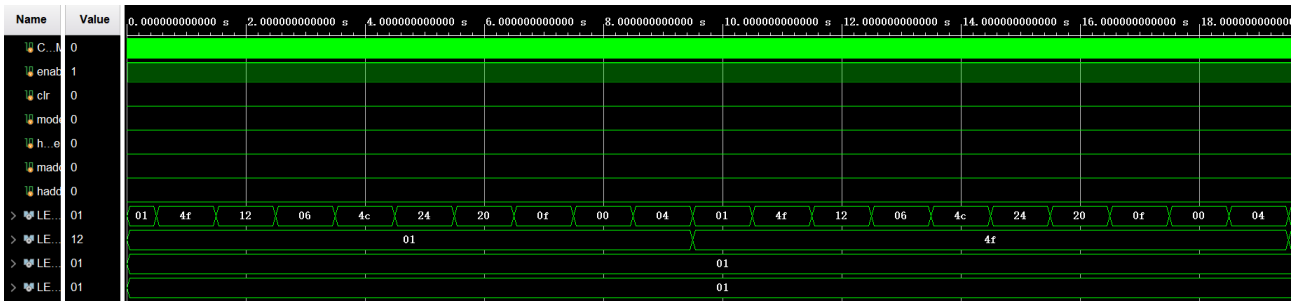
```
`timescale 1ns / 1ps
// 是加了分频器、按钮除颤的 clock_core。输入要 1M Hz
module clock(
        CLK_1M,enable,clr,mode,hourMode,madd,hadd,
        LEDS0,LEDS1,LEDS2,LEDS3
    );
    wire clk, minute_add, hour_add;
    input CLK_1M,enable,clr,mode,hourMode,madd,hadd;
    output [6:0] LEDS0,LEDS1,LEDS2,LEDS3;
    // 以下这行仅在仿真中测试用，可以删掉。同时应删掉 clock_core 中的相关内容
    wire [7:0] osecond;wire [7:0] ominute;wire [7:0] ohour;
    btn m(CLK_1M,madd,minute_add);
    btn h(CLK_1M,hadd,hour_add);
    freq f(CLK_1M,clk);
    clock_core
```

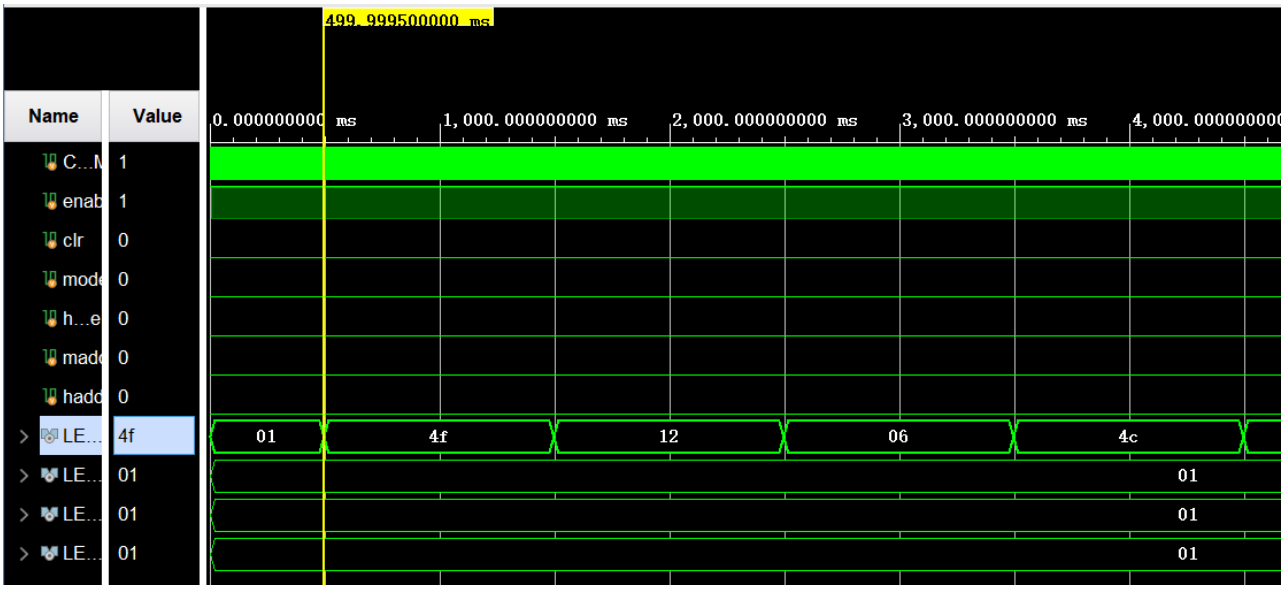
```
c(clk,enable,clr,mode,minute_add,madd,hour_add,hadd,hourMode,osecond,o
minute,ohour,LEDS0,LEDS1,LEDS2,LEDS3);
endmodule
```

五．提高与创新研究

将 clock IP 化，并测试。



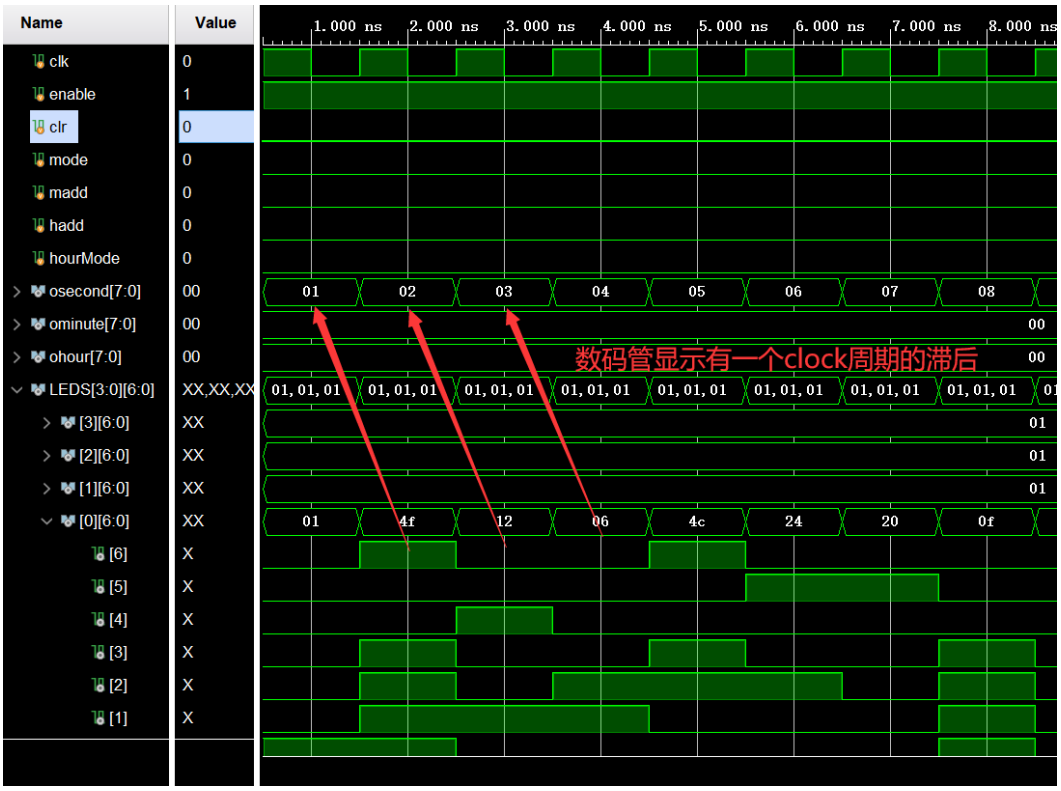




根据分频器仿真结果，在 500ms 处会触发一次计数。Ip 核构建、使用成功。

六．分析与总结

1. 在“时分-分秒”切换时，曾出现过下图的情况：数码管显示和计数有一个 clk 周期的滞后。原因是使用了以下代码：




```
clock_core 模块中：
reg [3:0] decodeIn [3:0];
always@(mode or posedge clk) begin
    decodeIn[0] <= mode?cout[2]:cout[0];
    decodeIn[1] <= mode?cout[3]:cout[1];
    decodeIn[2] <= mode?HOUR[3:0]:cout[2];
    decodeIn[3] <= mode?HOUR[7:4]:cout[3];
end
```

这个方案实际为时钟刷新数码管显示，刷新率为一个 clk 周期。本意是使用复用器选择，但写成了时序逻辑的形式，因为以为“?:”运算符必须用在 always 中。虽然整体滞后对于时钟使用并无大碍，但是这个设计颇为多余，花费了更多的 FPGA 资源。发现问题后改为 assign 的组合逻辑方法，使输出同步于计数器，解决了问题。

2. 在编写小时的 24->12 转换模块时，遇到了 BCD 码的减法问题。对于一般的 BCD 码加减法，算法颇为复杂；而这里被减数最大不过 23，减数也已经固定位 12，算法就简化了许多。需要注意，由两个 4 位 BCD 码拼起来的八位的 BCD 码不能直接相减，能直接相减当且仅当被减数的两个 4 位 BCD 码分别大于减数的两个 4 位 BCD 码。而此处被减数的范围为 13~23，因此需要分为：个位大于等于 2 和个位小于 2 两类。大于等于 2 时，被减数范围在 13~19，22~23，可以直接相减；小于 2 时，被减数可以取 20、21，对应输出为 8、9，写成 BCD 码后观察得到，此时输入的 8 位 BCD 减去 8'b00011000 即可。

3. 仿真时常常有输出没有任何变化的情况，排除 bug 无果，最后发现是使能端没有置 1。写 testbench 需谨慎。

4. 完善了时钟的一个不足：计数器进位高电平会持续 1s，而我将“加一”和“秒进位”用“或”传入分计数器的“clk”，会导致分在进位后的 1s 内无法对“加一”信号做出响应。想到了两个解决的办法：

①可以给计数器加一个专用的“add”端，但是“always @(posedge clk or posedge add or negedge clr)”在 RTL 综合时会报错：“ambiguous clock in event control”，原因是 always 中没有用到 add。解决这个问题可以给 add 单独开一个 always，但是综合出来的电路很丑陋，故抛弃。

②可以回归电路方法，用计数器特定的几位的逻辑运算控制。但是这样会面临进位信号宽度短，故抛弃。

“鱼和熊掌不可兼得”。于是想到了以下使用思路：当用户想要加的时候，忽略一切进位。于是，verilog 中有了“add_signal?add:clk”。当用户按下按钮，add_signal=1，表示想加，此时就变成了只有 add 信号可以使计时器加一。而 add 由 add_signal 除颤而得。就解决的这个不足。