计算机组成原理Lab2

一、实验流程

1. 首先在lab_top.sv中加入如下代码，其分别绑定了计数器、按键检测和译码器模块的相应接口。

```
// 内部信号声明
logic trigger;
logic [3:0] count;

// 计数器模块
// TODO: 在 lab2 目录中新建 counter.sv，实现该模块
counter u_counter (
  .clk    (clk_10M),
  .reset  (reset_of_clk10M),
  .trigger(trigger),
  .count  (count)
);

// 按键检测模块，在按键上升沿（按下）后输出高电平脉冲
// TODO: 同上，实现 trigger 模块，并例化
debouncer u_debouncer (
  .clk      (clk_10M),
  .reset    (reset_of_clk10M),
  .push     (push_btn),
  .debounced(trigger)
);

// 低位数码管译码器
// TODO: 例化模板中的 SEG7_LUT 模块
SEG7_LUT u_seg (
  .oSEG1(dpy0),
  .iDIG (count)
);

endmodule
```

2. 计数器模块当 `reset` 时输出置零，否则按照约定把输出值累加1，直到计数器达到0xF为止。

```
module counter(
```

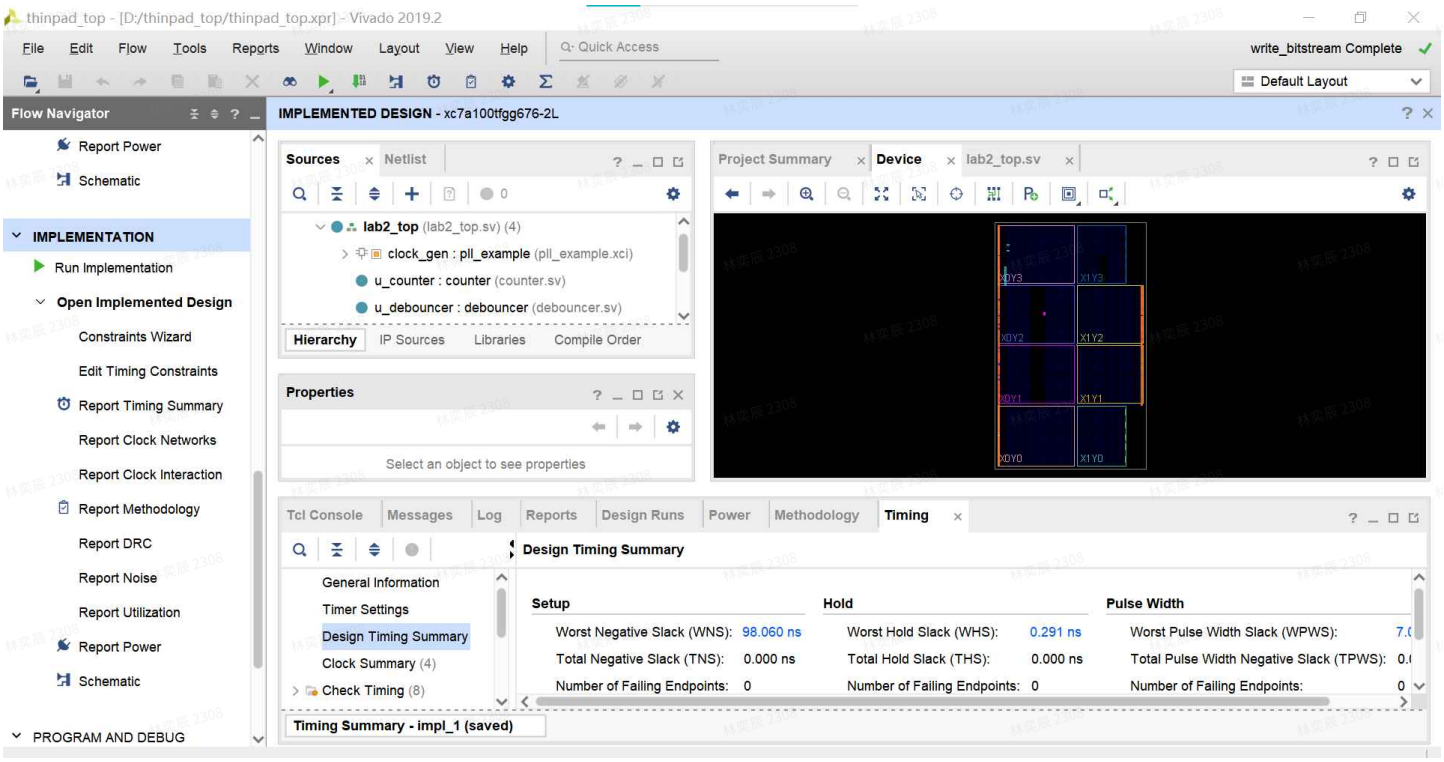
```
    // 时钟与复位信号，每个时序模块都必须包含
    input wire clk,
    input wire reset,
    // 计数触发信号
    input wire trigger,
    // 当前计数值
    output wire [3:0] count
    );

    reg [3:0] count_reg;
    always_ff @ (posedge clk or posedge reset) begin
        if (reset) begin
            count_reg <= 4'd0;
        end else begin
            if (trigger && count_reg != 4'hf)   // 增加此处
                count_reg <= count_reg + 4'd1;  // 暂时忽略计数溢出
        end
    end
    assign count = count_reg;

endmodule
```

3. 按键检测模块则需要记忆按键的在上一刻时候的值，以此来确保每次按键时数值仅增加1。

```
module debouncer(
    input wire clk,
    input wire reset,
    input wire push,
    output wire debounced
);

reg last_push_1;
reg last_push_2;

always_ff @ (posedge clk or posedge reset) begin
    if (reset) begin
        last_push_1 <= 1'b0;
        last_push_2 <= 1'b0;
    end else begin
        last_push_1 <= push;
        last_push_2 <= last_push_1;
    end
end

assign debounced = last_push_1 && (!last_push_2);
```

```
22
23 endmodule
```

4. 之后在**Vivado**平台进行测试，将`lab2_top`置为最顶层，并且将`counter.sv`和`debouncer.sv`作为source文件也加入`lab2_top`模块中，通过综合、生成.bit文件。



5. 进行在线测试后，按下CLK，开始累加

6. 按下RST，值归零

7. 最大累加到0xF为止。

二、思考题

> **计数器模块中提到的异步逻辑与同步逻辑有何不同？可以通过观察 Vivado 综合后的电路原理图，并且查阅相关资料回答本题。**

异步逻辑与同步逻辑在硬件设计中有明显的区别，主要体现在时序性和时钟控制上：

1. 时序性：
    - 异步逻辑：异步逻辑是指在信号改变时立即响应的逻辑，不受全局时钟的控制。异步逻辑的执行时间取决于输入信号的变化，因此不具备明确的时序关系。在异步逻辑中，信号的改变可以立即影响输出，而且不受时钟的限制。
    - 同步逻辑：同步逻辑是受全局时钟信号控制的逻辑。它在时钟信号的上升沿或下降沿触发时才执行操作，具有明确的时序关系。同步逻辑通常用于确保在时序系统中准确地控制信号的变化和传输。
2. 时钟控制：
    - 异步逻辑：异步逻辑没有全局时钟，因此不受时钟同步限制。它可以随时响应输入信号的变化，但容易引入时序问题，如冲突、竞争条件等。
    - 同步逻辑：同步逻辑受全局时钟信号的驱动，只在时钟的上升沿或下降沿执行操作。这种时钟控制有助于避免时序问题，确保系统稳定性。

总体而言，同步逻辑的时钟之间有固定的因果关系，这是由于同步逻辑电路利用一个全局时钟脉冲信号，当此信号同时到达元件时才 会引起状态改变。而异步逻辑没有全局统一的时钟驱动（即时钟之间无固定的因果关系），任意输入信号都可以引起电路状态改变，因此电路中状态变化的时刻是不稳定的。

观察综合后的电路原理图可以明显看出异步逻辑和同步逻辑的不同。异步逻辑通常在原理图中没有与时钟相关的元件，而同步逻辑会显示与时钟信号相关的触发器（如D触发器或JK触发器）等元件，以确保时序正确性。同时，综合工具通常会进行时序分析，以检查同步逻辑中的时序约束是否满足，而不会对异步逻辑执行类似的分析。这是因为同步逻辑的时序行为是可预测和可控的，而异步逻辑则更容易引入时序问题。