组成原理课程第三次实验报告

实验名称： 设计实战Lab2

学号:2310764 姓名:王亦辉 班次:计科一班

1 寄存器堆仿真

1.1 实验目的

了解寄存器堆的“两读一写”结构。

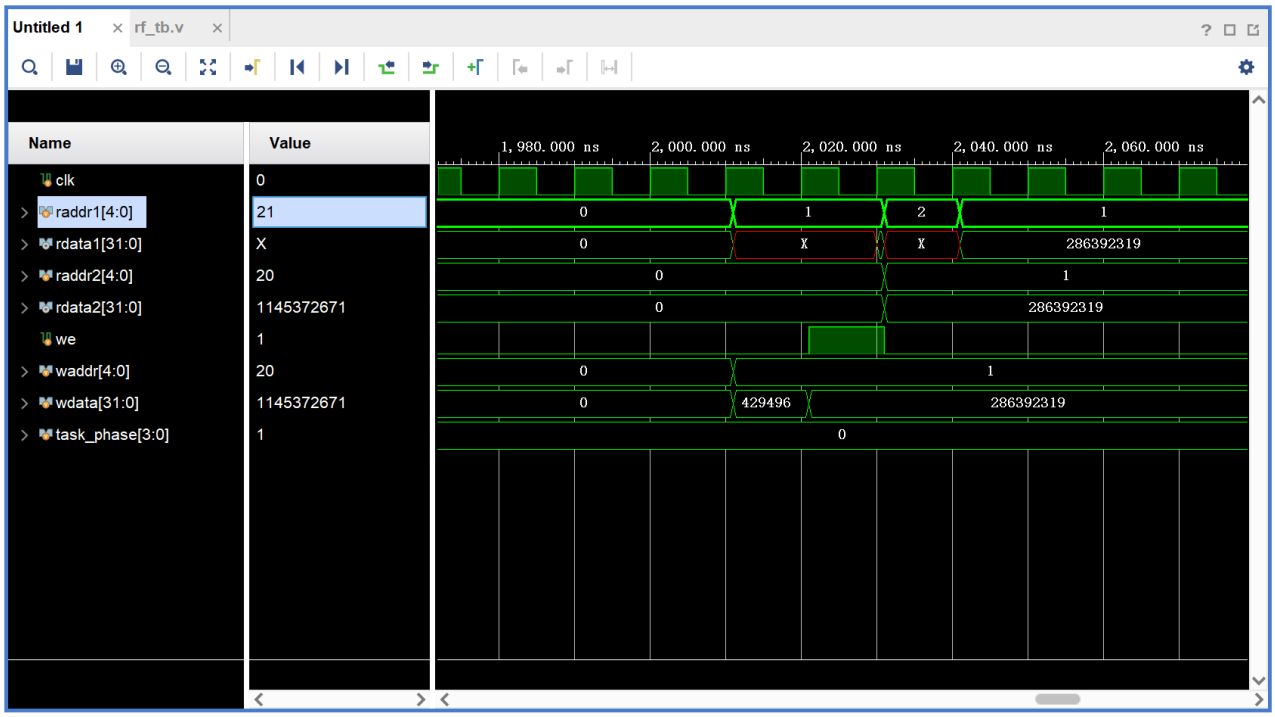
1.2 实验内容说明

1. 学习 3.1 和 3.3 节的内容。
2. 使用 Vivado 新建一个工程。
3. 点击“Add Sources”,选择添加设计源码 (design sources), 加入 lab2. zip 里的 regfile_task/regfile. v。
4. 点击“Add Sources”，选择添加仿真源码 (simulation sources), 加入 lab2. zip 里的 regfile_task/rf tb. v。
5. 对工程进行仿真测试，结合波形观察寄存器堆的读写行为。

1.3 实验步骤

新建工程，添加相应文件，进行仿真。

1.4 实验结果分析



可以看到，两个读是随其地址随时变化的，并不需要等待 clk 上升沿；而写入是需要 we 使能且遇到 clk 上升沿时，才会将数据写入寄存器。

注意到有部分时间，rdata1 是 X，这无关紧要，只是因为没有对相应寄存器赋值，且在 testbench 开始的时候没有初始化这些寄存器。

1.5 总结感想

1. 寄存器堆的写入是通过时钟来同步的。计算机里需要改变状态的任务，大都需要同步以确保稳定的状态。
2. 为什么寄存器堆要设计成“两读一写”?
   大多数指令需要读取两个寄存器，设计成一个周期能够读两个寄存器，算是加速大概率事件，提高指令的运行速度。如果同时可以多个写，可能会引入写入权限冲突的问题，制造不必要的麻烦，并且 RISC 中没有需要向两个寄存器分别写数据的指令。

2 同步 RAM 和异步 RAM 仿真、综合与实现

2.1 实验目的

了解同步 RAM 和异步 RAM 的特性与不同之处。

2.2 实验内容说明

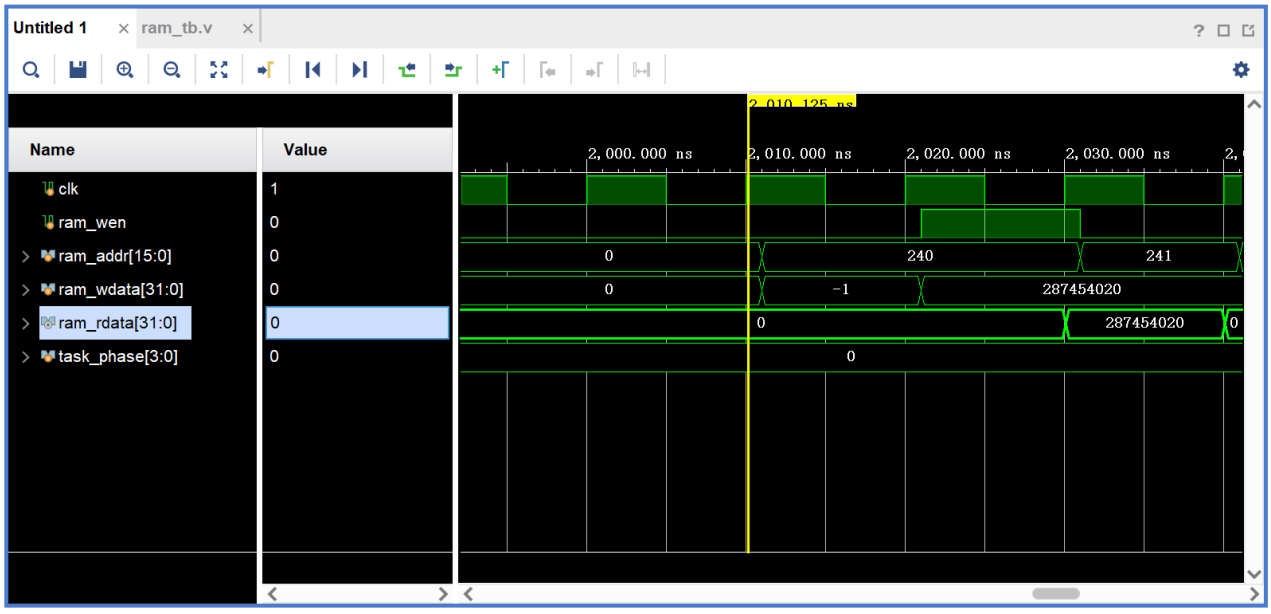
为同步 RAM、异步 RAM 各建立一个工程，调用 Xilinx 库 IP 实例化同步 RAM、异步 RAM，会提供一个设计的顶层文件，将它们封装成相同的模块名和接口。注意：封装后的 RAM 接口没有使能（或者称为片选）信号，表示永远使能（使能信号在 RAM 内部恒为 1）。

2.3 实验步骤

将相关文件添加进项目。
按照书上流程添加同步/异步 IP 核
进行仿真，观察仿真结果。

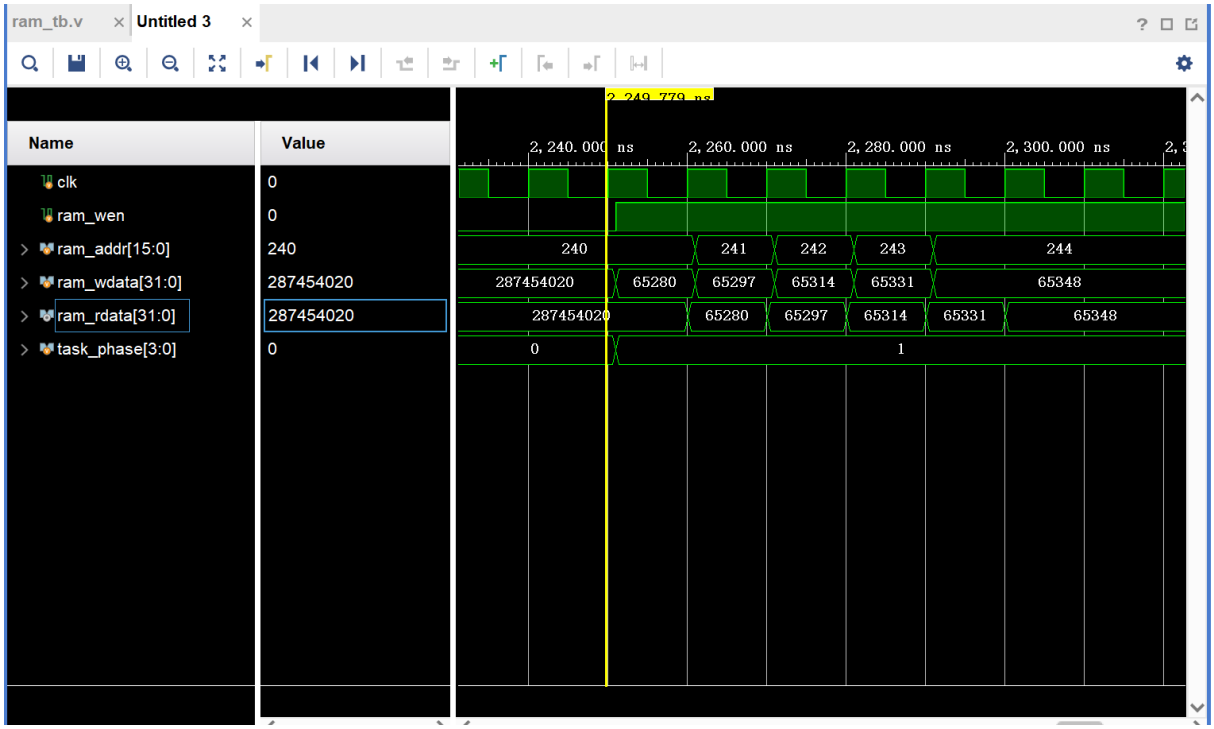
2.4 实验结果分析

2.4.1 同步 RAM

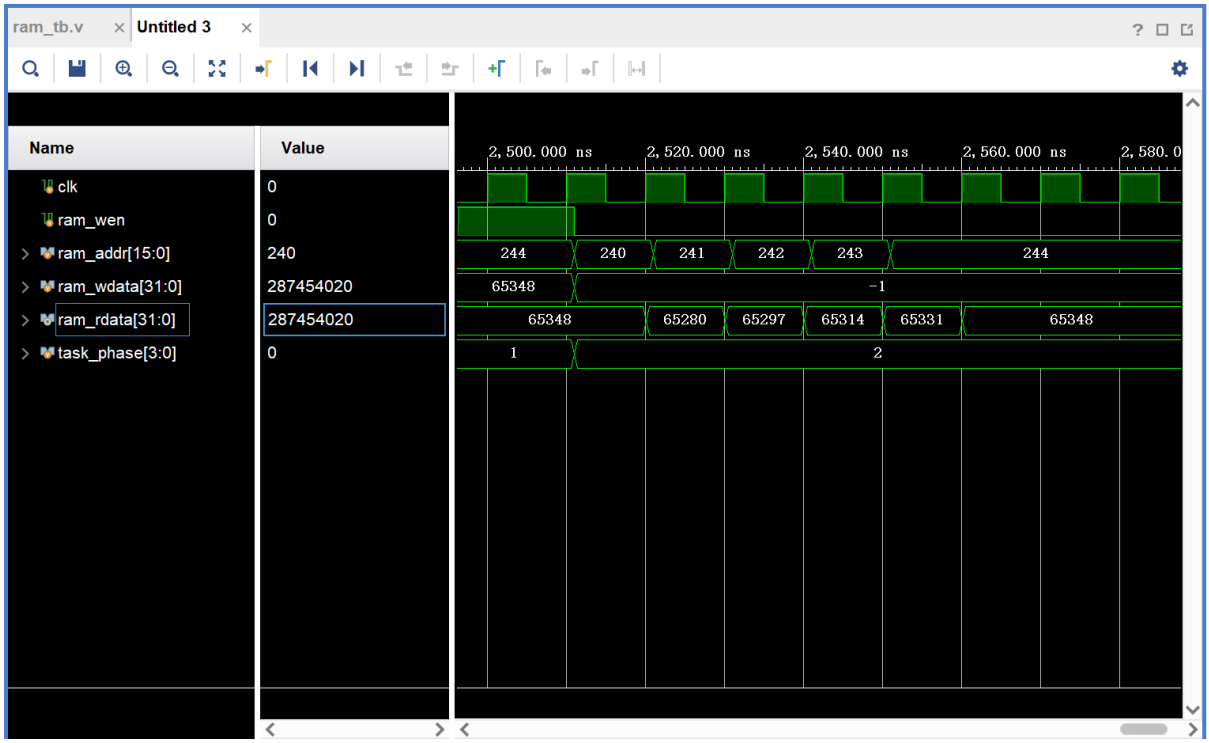


- 从上图可以发现，同步 RAM 只在 clk 上升沿进行数据的写入和读取，且写入时额外需要 ram_wen 为 1，即写入使能。
- 认为黄色竖线处为第一个 clk 上升沿，则将 wdata 改为-1 后，虽然遇到了第二个 clk 上升沿，但是没有使能，因此未写入；而将 wdata 改为 28745020 之后，虽然有写入使能，但仍要到第三个 clk 上升沿才会写入。
- 第三个上升沿处，rdata 被更新为 28745020，虽然之后 addr 改为了 241，但是直到第四个上升沿，rdata 才更新为 0，因此读取也是在 clk 上升沿进行。

- 所以说同步 RAM 的同步是指写入和读取都依赖时钟，只在时钟上升沿时更新。

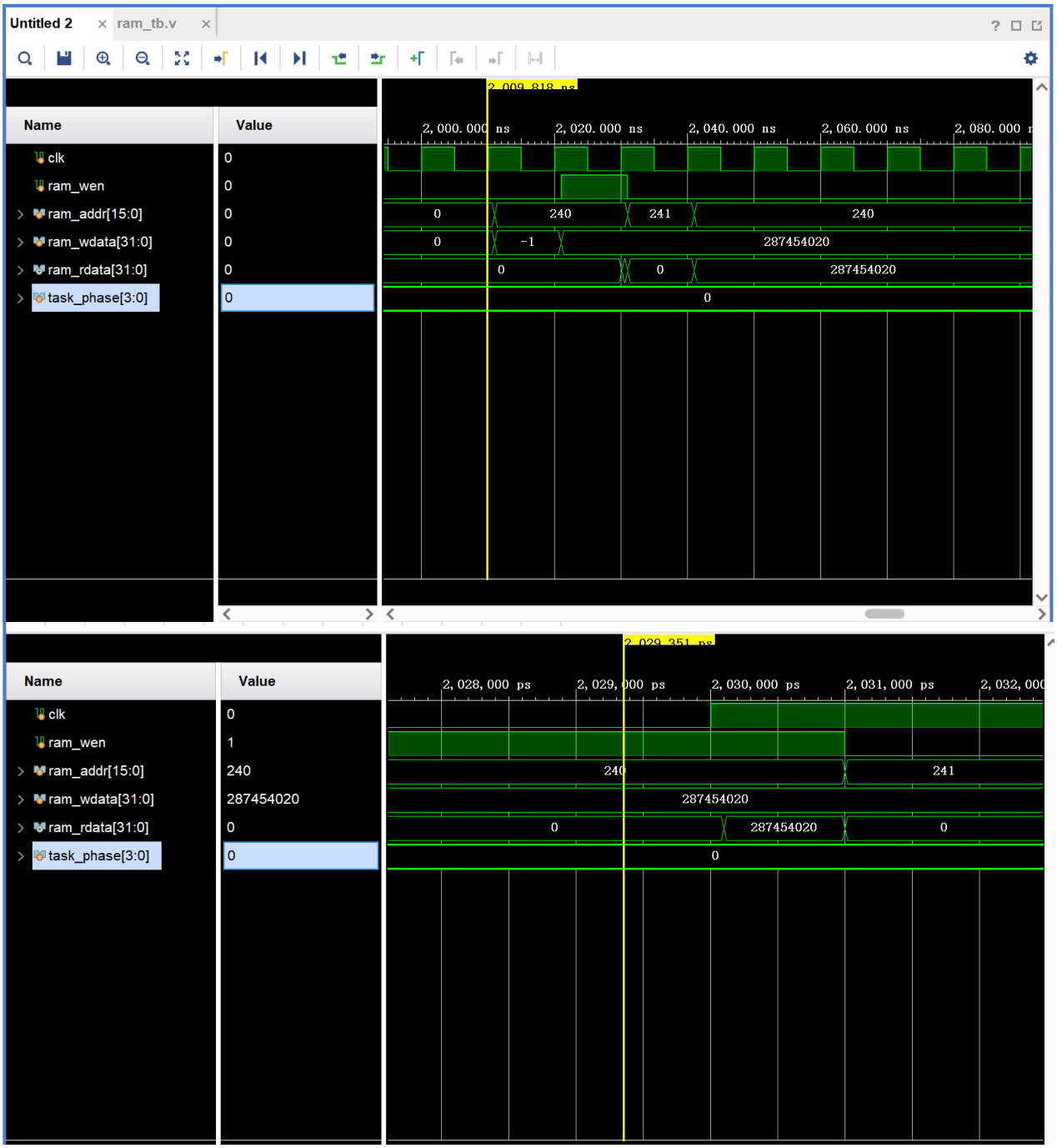


task_phase 1 处同样是展示写入和读取的同步性，可以看到，每次在 clk 上升沿处，rdata 才会发生改变，即使 addr 和 wdata 已经改变了很久。



task_phase2 则是展示刚才写入的所有数据

2.4.2 异步 RAM



- 观察上图一，异步 RAM 只在 clk 上升沿且使能时写入，而读取是异步的，会实时读取当前地址的数据。
- 以上图一黄线为第一个 clk 上升沿。此上升沿之后，wdata 改为 -1，但观察发现即使在第二个 clk 上升沿，rdata 仍为 0，因为 wdata 为-1 的全部时期都没有使能信号。而在第三个 clk 上升沿，此时有使能，因此写入，图二为放大后的结果，可以看到 rdata 为 28745020。因此
- 对于读入，可以看到图一的第三个上升沿之后，addr 改变，rdata 随即改变为 0，这是 241 位置处的默认值，因此异步 RAM 中，读取是实时的。

- task_phase 1、2 同样是展示异步 RAM 的写入是同步且需要使能信号，读取是异步，随时随 addr 改变的特性，笔者不再赘述。

2.5 总结感想

1. 同步 ram 和异步 ram 各自的特点和区别：
   同步 RAM ，读写都与时钟同步；异步 RAM，读取是异步的，而写入仍然是同步的。写入都使用同步，可能是为了确保数据的稳定性，防止寄存器变化过快 (变化时间小于一个时钟周期)导致竞争使某些地方产生不可预料的结果。

3 数字逻辑电路的设计与调试

3.1 实验目的

熟悉使用 vivado 进行 Debug 的方法。

3.2 实验内容说明

本实践任务提供了一个有5个 bug 的数字逻辑电路设计源码。进行仿真，并充分利用仿真的辅助小技巧（分割、分组、颜色变化、标志等）进行调试，找出所有的 bug。

3.3 实验步骤

一共有 5 个 BUG

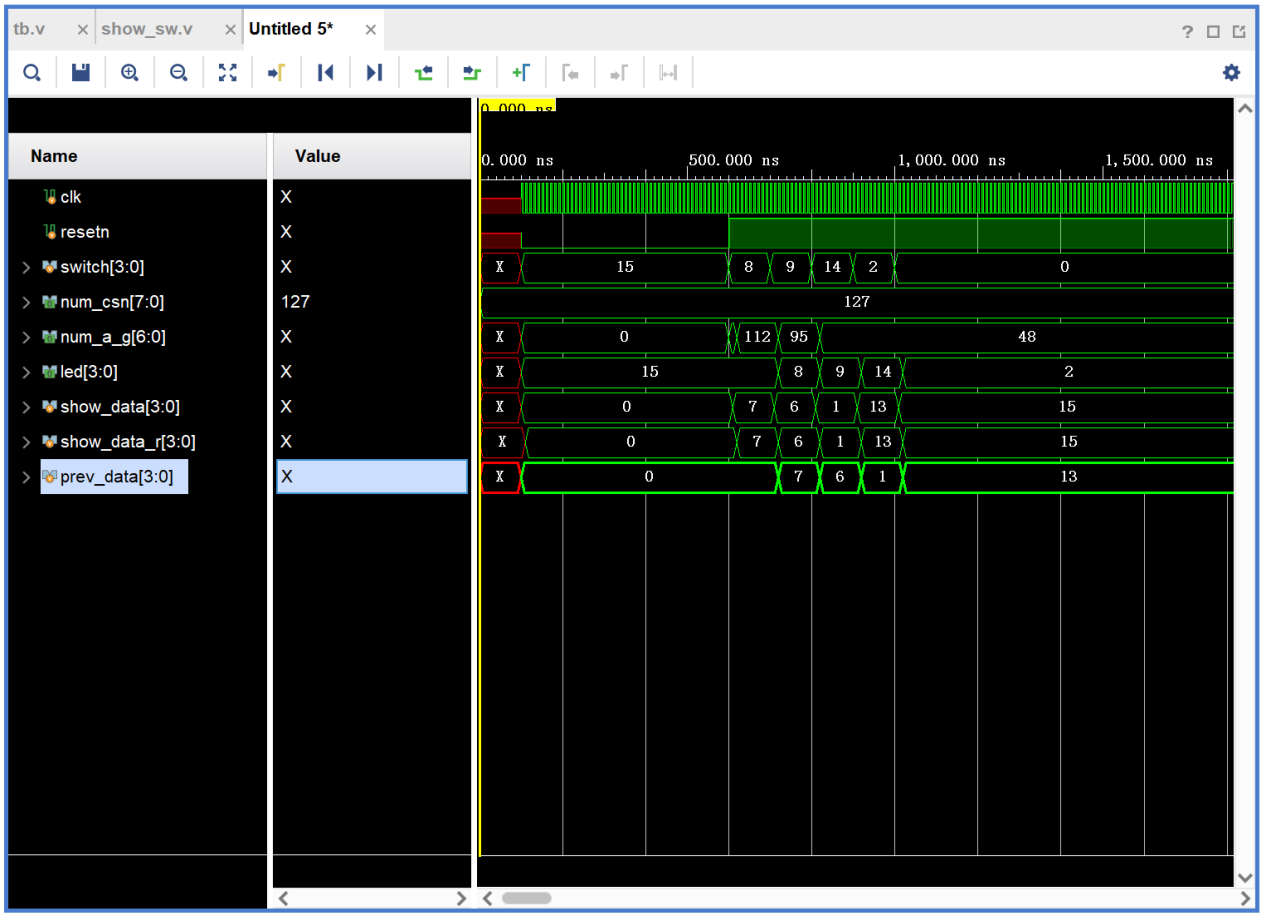
```
    //new value
    always @(posedge clk)
    begin
        show_data   <= ~switch;// BUG1 导致X未确定 type2 。 reg未赋值  注释掉了?此地无银三百两
    end

    always @(posedge clk)
    begin
        show_data_r <= show_data; //BUG2  导致越沿采样  。 "=" -> "<="
    end
    //previous value
    always @(posedge clk)

    //show number: new value
    show_num u_show_num(
            .clk        (clk      ),
            .resetn     (resetn   ),

            .show_data  (show_data),
            .num_csn    (num_csn  ), //BUG3 导致Z高阻 type1  。 拼写问题num_scn -> num_csn
            .num_a_g    (num_a_g  )
    );

    //keep unchange if show_dtaa>=10
    // wire [6:0] keep_a_g; 一整个没用。。。。
    // assign     keep_a_g = num_a_g ; //BUG4 不应该 “+ nxt_a_g;”。 会导致组合环路，且不符合功能

    assign nxt_a_g = show_data==4'd0 ? 7'b1111110 :   //0
                     show_data==4'd1 ? 7'b0110000 :   //1
                     show_data==4'd2 ? 7'b1101101 :   //2
                     show_data==4'd3 ? 7'b1111001 :   //3
                     show_data==4'd4 ? 7'b0110011 :   //4
                     show_data==4'd5 ? 7'b1011011 :   //5
                     show_data==4'd6 ? 7'b1011111 : //BUG5 。功能错误,没写6
                     show_data==4'd7 ? 7'b1110000 :   //7
                     show_data==4'd8 ? 7'b1111111 :   //8
                     show_data==4'd9 ? 7'b1111011 :   //9
                                       num_a_g   ;
    endmodule
```

寻找 BUG 的详细过程如下：

1. 阅读代码，发现 always 块里的代码居然被注释了，检查后得知这不是我做的修改。观察上下文，发现这里是需要这个赋值的。
2. 阅读代码，看到 always 块里有阻塞赋值，感觉不对劲结合指导书得知会发生越沿采样，可能会导致另外一个用到 `show_data_r` 的变量提前一个 clk 上升沿发生变化。
3. 在仿真波形中，发现这个变量一直是 X，于是去源文件检查。在源文件中，利用双击高亮相同字符串的功能，看出是拼写错误导致的 `num_csn` 未被赋值。

4. 修正过一些 BUG 后，再进行仿真，遇到了跟实验指导书上一样的波形停止问题，我的是 `current time:710ns...` 猜测是组合环路，点击 cancel 之后，自动跳转到 `keep_a_g=num_a_g+nxt_a_g;` 这一行。阅读源代码，发现 `num_a_g+next_a_g` 没有意义，检查前后文，发现这里的功能是在新输入大于等于 10 的时候保持原值，思考后笔者得出并不需要 `keep_a_g` 这个变量的结论。
5. 阅读代码，发现 0~9 中少了 6 的情况。

3.4 实验结果分析



修改完 BUG 之后的波形。注意到最左边的 X 是在仿真文件中，开始的一段时间未对模块复位造成的，并非 BUG 。

3.5 总结感想

1. 找 BUG 是比较困难的，特别是当不理解变量命名的时候，比如后缀 `_r` 代表一定延时前或单纯就是上一次的某个变量。
2. 通过仿真得到的波形图可以较轻松地找到一些典型的 BUG，如 X 可能对应 reg 未赋值或者多驱动，Z 可能对应 wire 变量未被赋值等等。
3. 养成良好的代码编写习惯是减少 BUG 的关键。比如 always 里不应该使用阻塞赋值 `=` 而应用 `<=`；使用有意义的命名且尽量使用编辑器提供的代码补全而不是全部自己敲。