数字逻辑电路专题实验报告

2021K8009929033 马迪峰 5 号箱

任务与实践

实践任务 2：寄存器堆仿真

要求：对一个寄存器堆设计进行功能仿真，通过观察其仿真波形了解行为特征。

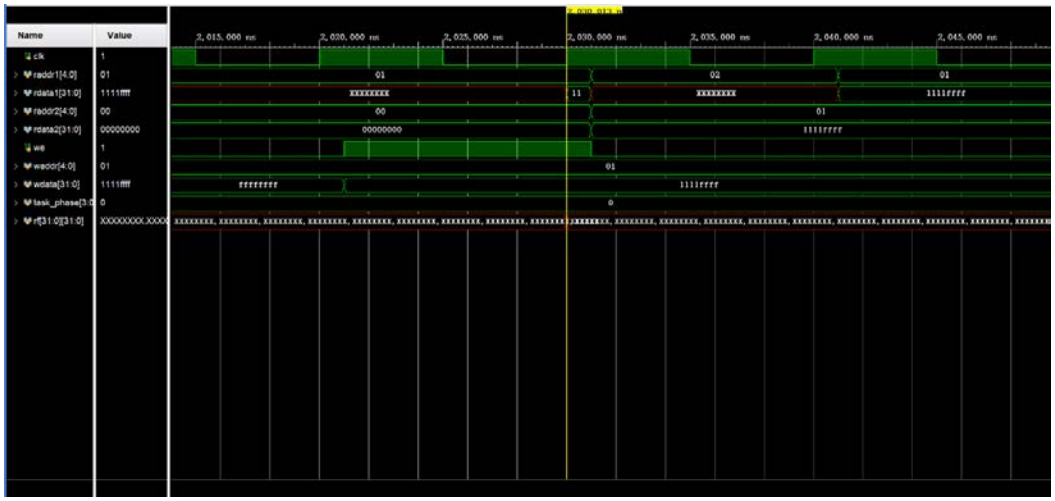
代码分析 ：

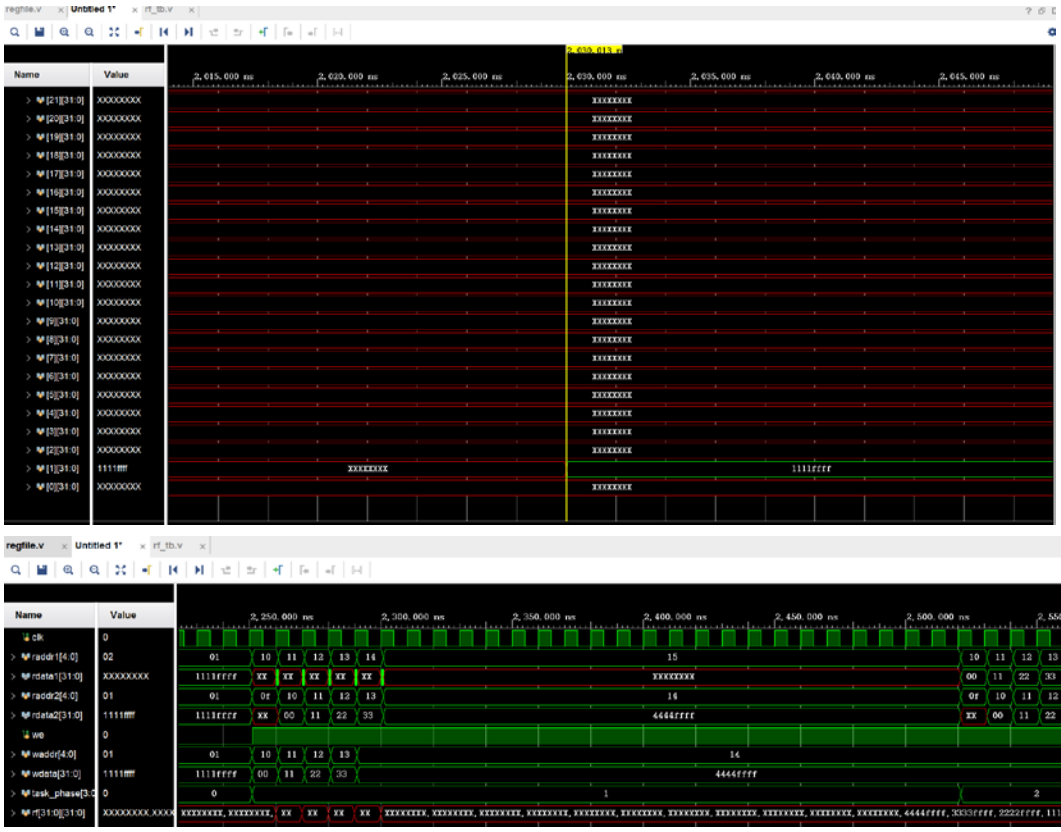
本任务所提供的源码中，寄存器堆是“两读一写”的结构，拥有两个读端口和一个写端口，其中读端口没有读使能信号，代表读信号永远使能，而写端口则由写使能信号控制，当且仅当时钟上升沿且写使能信号为真时才能写入数据。值得注意的是，规定 0 号位置寄存器读数据始终为 0.

仿真测试：

仿真激励文件中共进行了三组测试，2011ns 时开始进行读写操作。在此之前对各信号进行了初始化步骤，如果寄存器堆设计正确，寄存器堆最开始将会在 01 地址写入 32’bffffffff 数据，同时读出 1 号位置的数据，但是此时 1 号位置尚未存在数据，因此读出的数据应为不定态，在下一个时钟的上升沿阶段 1 号位置才会写入数据。以此类推可以分析其他部分的数据是否正确。

其波形样例如下：





可以看出当写使能信号为真时，写数据写入相应寄存器，在之后某段时刻当读地址访问该寄存器时会读出写入的数据，同时如果是 0 号寄存器，则会读出 0.这可以说明寄存器堆的功能正确。

●注意：VIVADO 默认仿真测试时间是 10us,但是仿真测试开始的时间是 20us，因此最开始出现的数据全为 0，再执行一次才会出现测试结果。

实践任务 3：同步 RAM 与异步 RAM 的仿真、综合与实现

要求：

1. 调用 Xilinx 库 IP 实例化一个同步 RAM，进行仿真以观察行为，进行综合和实现后查看时序结果和资源利用率。
2. 调用 Xilinx 库 IP 实例化一个异步 RAM，进行仿真以观察行为，进行综合和实现后查看时序结果和资源利用率。

同步 RAM 和异步 RAM 的区别：

●同步 RAM：同步 RAM 使用一个外部时钟信号来协调内存操作。这个时钟信号确保了数据的读取和写入都在特定的时钟周期内完成，以便与其他系统组件同步。这可以提高内存访问的可靠性和性能。

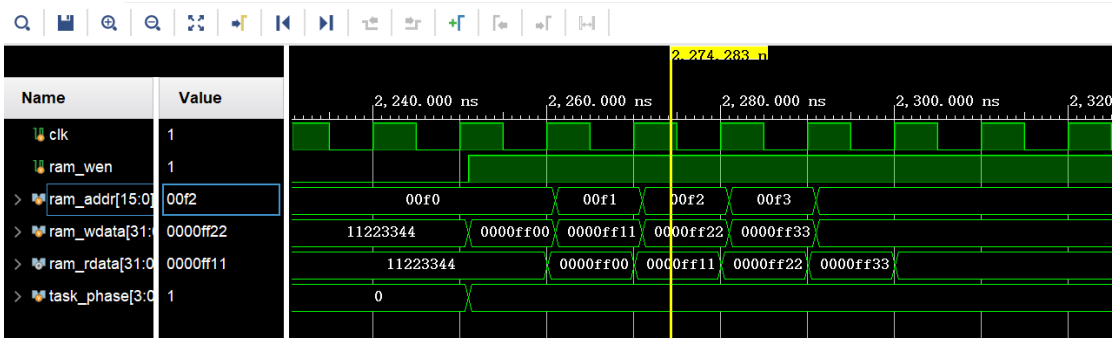
●异步 RAM：异步 RAM 不依赖于外部时钟信号来同步内存操作。它的读写操作是根据控制信号和时序信号来执行的，没有严格的时钟要求。这使得异步 RAM 设计相对简单，但可能会导致性能不稳定。

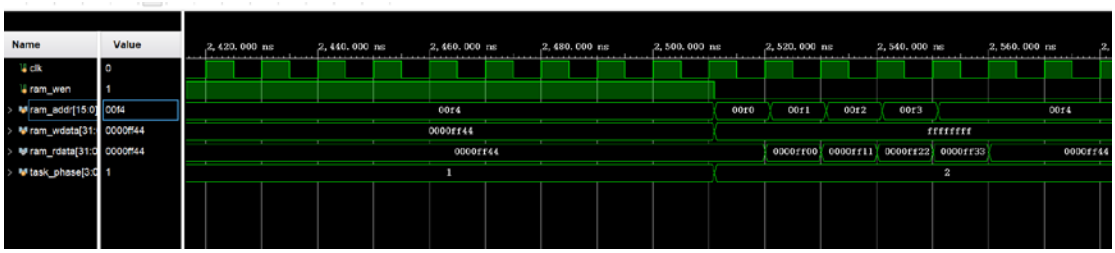
同步、异步 RAM 端口如下：

名称	宽度	方向	描述
clk	1	Input	时钟信号
ram_wen	1	Input	RAM 的写使能信号：为 1 表示写入操作，为 0 表示读取操作
ram_addr	16	Input	RAM 的地址信号，读和写的地址都由该信号指示
ram_wdata	32	Input	RAM 写入的数据
ram_rdata	32	Output	RAM 读出的数据

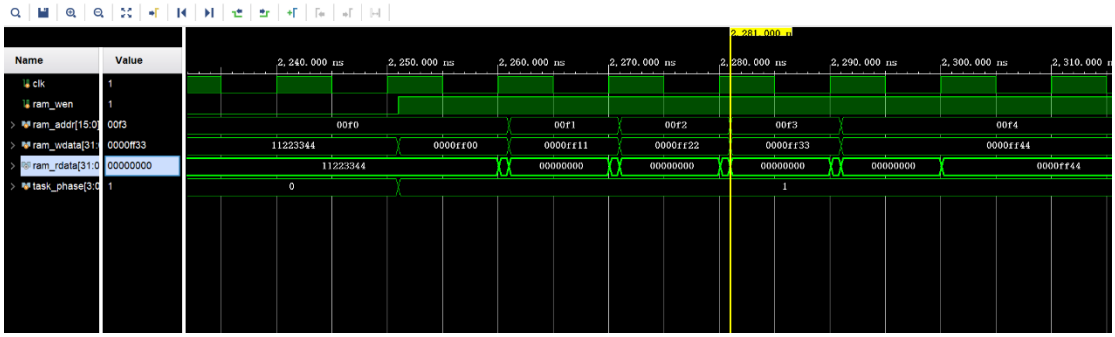
仿真波形：

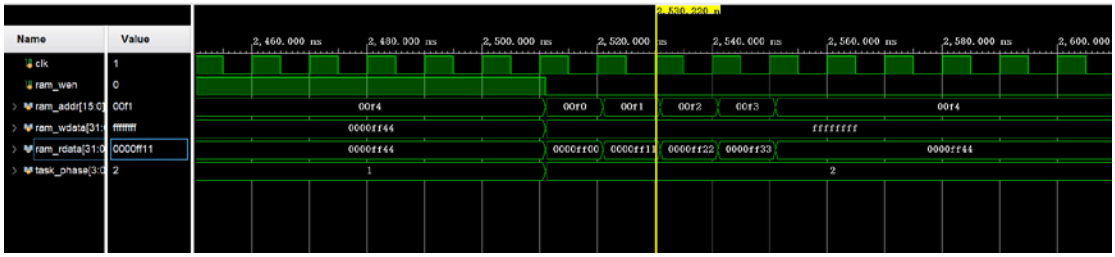
（1）同步 RAM





（2）异步 RAM

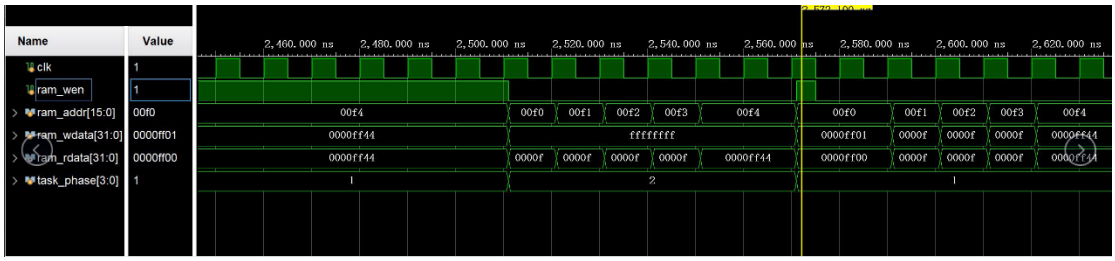




同步异步读写行为的异同：

很容易可以看出，同步异步 RAM 最大的不同表现在读行为上，异步 RAM 读数据并不依赖于外部时钟，也就是说它并没有与时钟同步，而是完全依赖于地址线的变化，即读行为有单独的控制线。在同步 RAM 中，读行为只会在时钟上升沿中发生变化，以同步 RAM 示意图为例，2260 ns 时为写操作，在 00f0 地址上写入了数据 0000ff00 ,同时读数据立即变为 0000ff00，表明读行为是与时钟同步的，否则应该在地址线发生变化时才会发生数据的变化。

注意到，由波形并不能判断出写是否是同步的（一般理应为同步），但是我们可以通过改变激励文件，让 wen 信号很短不跨过时钟上升沿，通过读数据看看是不是写进去了，就知道是不是同步的了



如图，这个 0000ff01 根本就没有写进 00f0，读出来的还是之前的 0000ff00。说明并没有写进去，这表明是同步写异步读。

时序结果和资源利用率：

●TNS（Total Negative Slack）：TNS 代表总负时序余量，它是一种度量，用于衡量数字电路的时序性能。TNS 表示了设计中所有时序路径中最紧迫的负余量之和。如果 TNS 为正，表示设计满足了所有时序要求；如果 TNS 为负，表示设计未能满足某些时序要求，需要进一步的优化。

●WNS（Worst Negative Slack）：WNS 代表最差负时序余量，它表示设计中最紧迫的负余量，即距离时序要求最远的路径的负余量。WNS 是一种关键性能指标，因为它反映了设计中最需要改进的时序路径。通过减少 WNS，可以提高整体的时序性能

时序余量表示在一个特定时序路径上的信号传播所剩余的时间，即信号在达到目的地之前的时间余量。

时序余量 = 到达时间（Arrival Time） - 需求时间（Required Time）

●到达时间（Arrival Time）：表示信号从源端到达目的端的时间。它考虑了信号的

传播延迟以及时钟等因素，通常由时序分析工具计算得出。

●需求时间（Required Time）：表示信号在目的端需要达到的时间。这是由设计规范或时序约束规定的，它告诉我们信号需要在特定时钟周期内到达目的地。

因此可以作为评估时序性能的一个关键指标

（1）正时序余量：时序余量为正表示信号到达目的地比所需的时间早，即信号到达得足够快。这通常是一个良好的情况，因为它意味着设计有一些时序余量，可以容纳一些不确定性。

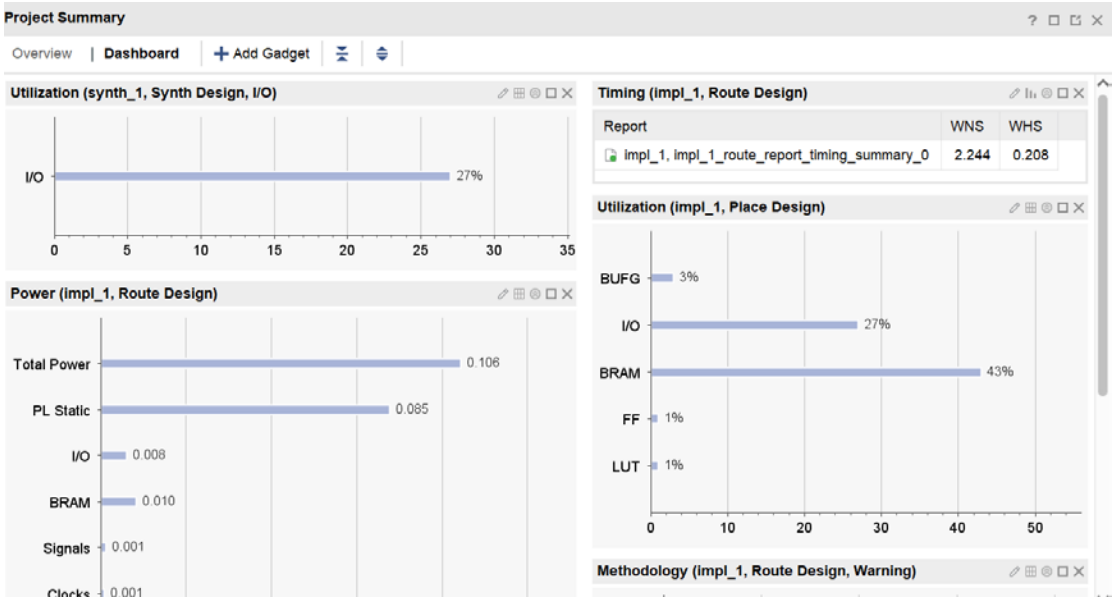
（2）零时序余量：时序余量为零表示信号到达目的地的时间与所需的时间完全匹配。这在大多数情况下是可接受的，但需要非常精确的时序控制。

（3）负时序余量：时序余量为负表示信号到达目的地比所需的时间晚，即信号到达较慢。这通常是一个问题，因为它表示设计未能满足时序要求，可能需要进一步的优化。

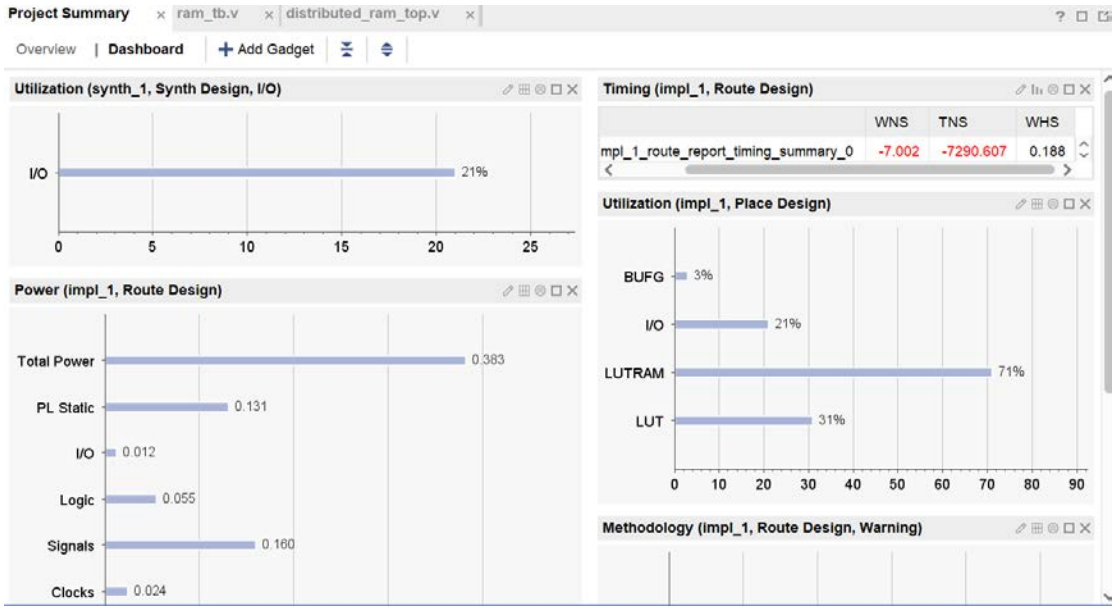
可以看到，同步 RAM 的 WNS 和 TNS 都是正值，这表明设计很好，已满足预期效果。而在异步 RAM 的设计中 WNS 和 TNS 的值均为负值，表明设计存在违约，与约束文件中所需求的时间不符并且远远超过 300ps，这说明时序远远不符合要求，需要继续优化。

原因就在于，异步 RAM 没有时钟同步，性能相对不稳定，内存访问时间会受到访问模式和外部环境等的影响。同步 RAM 中 BRAM 使用率很高，BRAM 的主要特点就是高带宽和低延迟，可以很好满足时序要求，而在异步 RAM 中则主要是使用了 LUTRAM，LUTRAM 的操作通常是时序独立的，不受外部时钟信号的限制，也是为什么异步 RAM 不同步的主要原因。

同步 RAM 结果如下：



异步 RAM 结果如下：



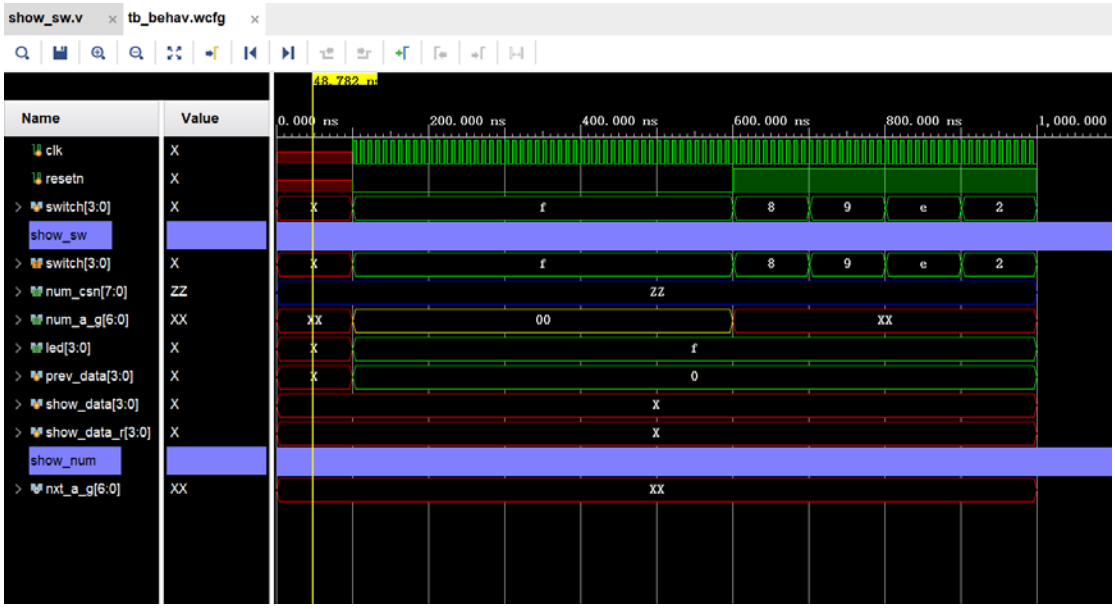
实践任务 4 数字逻辑电路的设计与调试

要求：调试并修正一个给定数字逻辑电路设计中的功能错误，上板验证功能正确。

正确功能如下：

1. 获取开发板最右侧 4 个拨码开关的状态（记为“拨上为 1，拨下为 0”，实际开发板上拨码开关的电平是“拨上为低电平，拨下为高电平”），共有 16 个状态（数字编号是 0~15）。
2. 最左侧数码管实时显示 4 个拨码开关的状态。数码管只支持显示 0~9，如果拨码开关状态是 10~15，则数码管的显示状态不更改（显示上一次的显示值）。
3. 最右侧的 4 个单色 LED 灯会显示上一次的拨码开关的状态，支持显示 0~15（拨码开关拨上，对应 LED 灯亮）。

错误的波形分析：



显然这样的波形并不符合设计，num_csn 信号悬空导致为高阻态，同时 nxt_a_g 数码管信号也始终为不定态，表明其未经过初始化。同样的情况也发生在 show_data 和

show_data_r 信号上，逐步分析可以依次找到问题所在。

●num_csn 信号悬空，分析源码发现在例化时，该信号未被正确连接，导致 num_csn 信号出错，仅需将 num_scn 改为 num_csn 即可。

```
show_num u_show_num(
        .clk        (clk      ),
        .resetn     (resetn   ),

        .show_data  (show_data),
        .num_csn    (num_scn  ),
        .num_a_g    (num_a_g  )
);
```

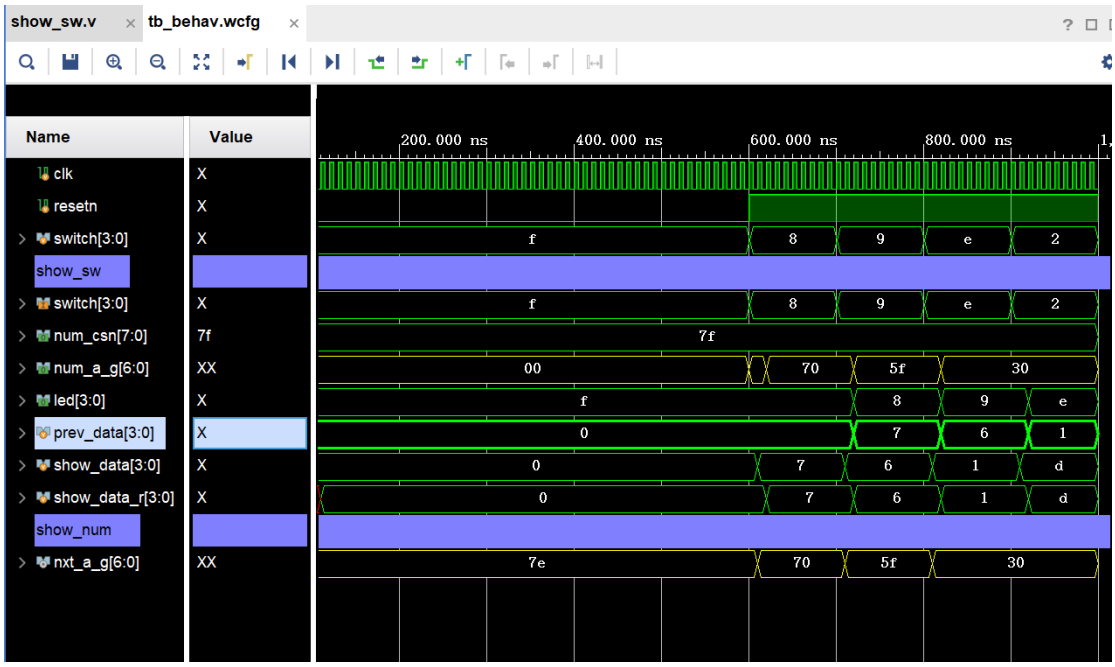
●show_data 信号为不定态，检查后发现 show_sw 部件中并未给其初始化，这是因为 show_data=~ switch 这条语句被注释所导致的结果，删除注释符号即可。

●同样，show_data_r=show_data 因为采用了阻塞赋值，导致两个信号的变化在上升沿时始终相等，最后 pre_data 信号不会更新出现错误，将其改为非阻塞赋值即获得正确结果。

●进一步发现，nxt_a_g 在 show_data==6 时并未置为正确的数码管信号，检查 nxt_a_a 选择器发现，show_data==6 并未考虑在其中，因此增上这条语句即可。

●改完上述错误后进行仿真发现波形不会停止，推测原因可能是设计中存在组合环路，检查代码发现 keep_a_g 信号和 nxt_a_g 信号互相产生，出现组合环路。根据设计分析，正确的设计应该是 keep_a_g 信号保持为 num_a_g，因此删去 nxt_a_g 信号即可。

改正后正确的波形如下：



正确源码如下：

```
module show_sw (
  input         clk,
  input         resetn,

  input    [3 :0] switch,   //input
```

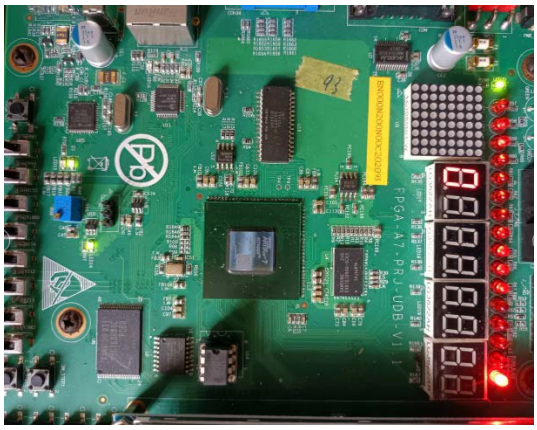
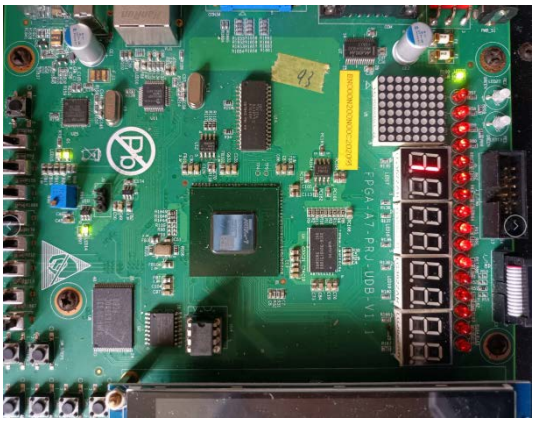
```
    output     [7 :0] num_csn,   //new value
    output     [6 :0] num_a_g,

    output     [3 :0] led        //previous value
);
//1. get switch data
//2. show switch data in digital number:
//   only show 0~9
//   if >=10, digital number keep old data.
//3. show previous switch data in led.
//   can show any switch data.

reg [3:0] show_data;
reg [3:0] show_data_r;
reg [3:0] prev_data;

//new value
always @(posedge clk)
begin
    show_data   <= ~switch;
end

always @(posedge clk)
begin
    show_data_r <= show_data;
end
//previous value
always @(posedge clk)
begin
    if(!resetn)
    begin
        prev_data <= 4'd0;
    end
    else if(show_data_r != show_data)
    begin
        prev_data <= show_data_r;
    end
end

//show led: previous value
assign led = ~prev_data;

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

        .show_data  (show_data),
        .num_csn    (num_csn  ),
        .num_a_g    (num_a_g  )
);

endmodule

//----------------------------{digital number}begin----------------------//
module show_num (
    input             clk,
    input             resetn,

    input      [3 :0] show_data,
    output     [7 :0] num_csn,
    output reg [6 :0] num_a_g
);
//digital number display
assign num_csn = 8'b0111_1111;

wire [6:0] nxt_a_g;

always @(posedge clk)
```

```
begin
   if ( !resetn )
   begin
      num_a_g <= 7'b0000000;
   end
   else
   begin
      num_a_g <= nxt_a_g;
   end
end

//keep unchange if show_data>=10
wire [6:0] keep_a_g;
assign     keep_a_g = num_a_g;

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 :          //6
                 show_data==4'd7 ? 7'b1110000 :   //7
                 show_data==4'd8 ? 7'b1111111 :   //8
                 show_data==4'd9 ? 7'b1111011 :   //9
                                   keep_a_g   ;
endmodule
//----------------------------{digital number}end-----------------------//
```

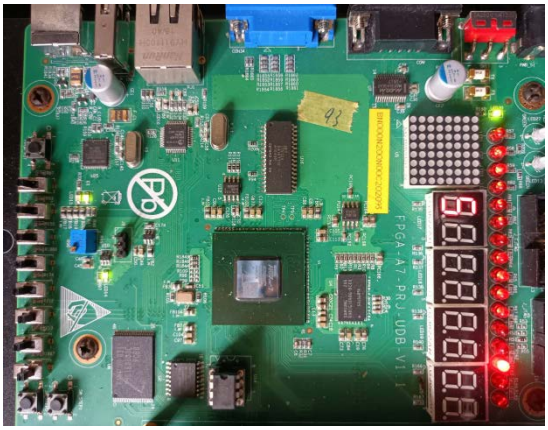
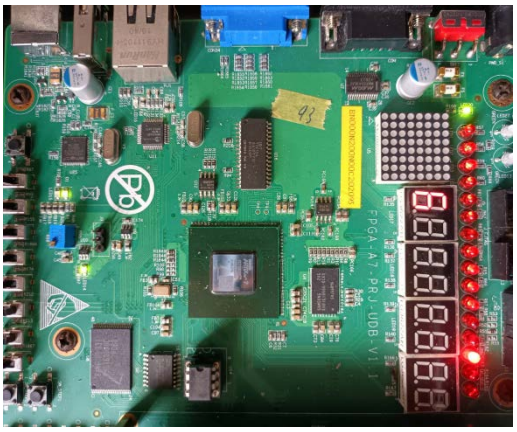
上板测试结果：

（1） 如果没有添加数码管的 6, 则将会停留在上次显示的值上（即为 2）

（2）0 拨到 1

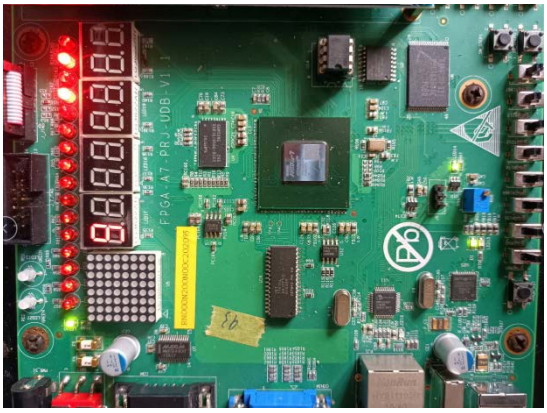
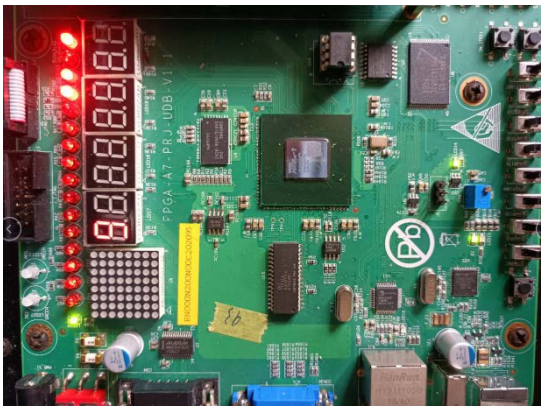



（3）5 拨到 6

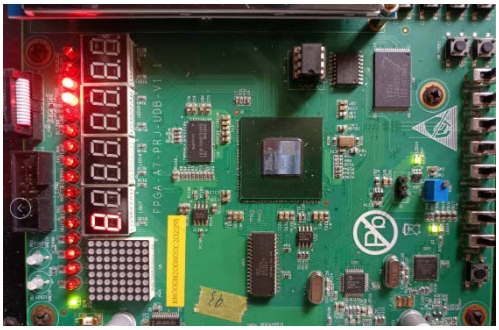
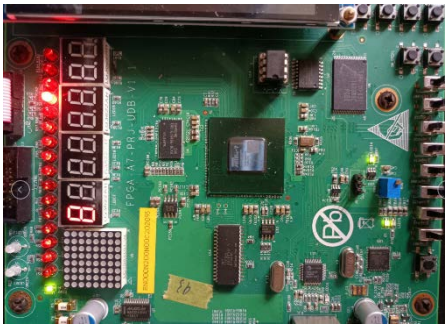



（4）大于十的情况

①9->10

②8->12

实验总结：

本次实验难度并不高，非常基础，与以往数字电路和计算机组成原理的实验相比，最大的区别是可以使用 FPGA 开发板，真正体验到了动手的感觉。基本上没有什么难度，但是仍然包括了一些新的知识点和概念，大体上对电路设计有了一个更细致的理解。

非要吐槽的话，莫过于没有参考的实验模板，实验内容虽然简单，但根本不知道写些什么，如何下手，比较耗费时间和精力。

更正：原来是模板发晚了，更痛苦了，写了好久发现原来有模板，颇有一种回炉重造的痛苦，希望下次可以提前发。