实验 2 报告

学号　2019K8009929019

姓名　桂庭辉

箱子号　44

1　实验任务

根据讲义内容完成三个实践任务，通过实践认识掌握寄存器堆、同步 RAM、异步 RAM 的读写行为与时序特征，进一步熟悉 Vivado 的使用与从设计到上板完整的 FPGA 开发流程，在实践中认识开发过程中的各种 bug 与解决方式。前两任务通过对波形图买哦书读写行为检验，后一任务可通过检查代码或上板行为进行检验。

2　实验过程

2.1　实验流水账

2021.09.05 20:30 ~ 2021.09.05 21:15 ： 完成实践任务一与实验报告相关部分撰写

2021.09.05 21:40 ~ 2021.09.05 23:30 ： 完成实践任务二与实验报告相关部分撰写

2021.09.06 18:45 ~ 2021.09.06 22:30 ： 完成实践任务三与实验报告整体撰写排版

2.2　寄存器堆仿真

寄存器堆的读写时序特征可概括为*同步写*、*异步读*，此外还包括对零号寄存器特殊读写要求的处理，以下将对这三个部分分别进行说明。

2.2.1　零号寄存器

零号寄存器的特征在于无论其中存储内容为何，在对其进行读操作时的结果始终为 0，即从用户层面看来零号寄存器中的值始终为 0。

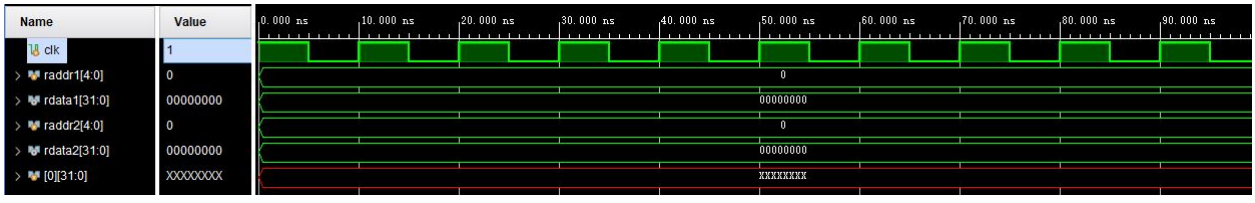

图 1: 寄存器堆零号寄存器与其读取结果

2.2.2　同步写

从以下的源码片段不难看出，向寄存器堆写入数据即修改寄存器堆内容始终发生在时钟信号 `clk` 的上升沿处，即同步时序过程。

```
// WRITE
always @(posedge clk) begin
    if (we) rf[waddr]<= wdata;
end
```

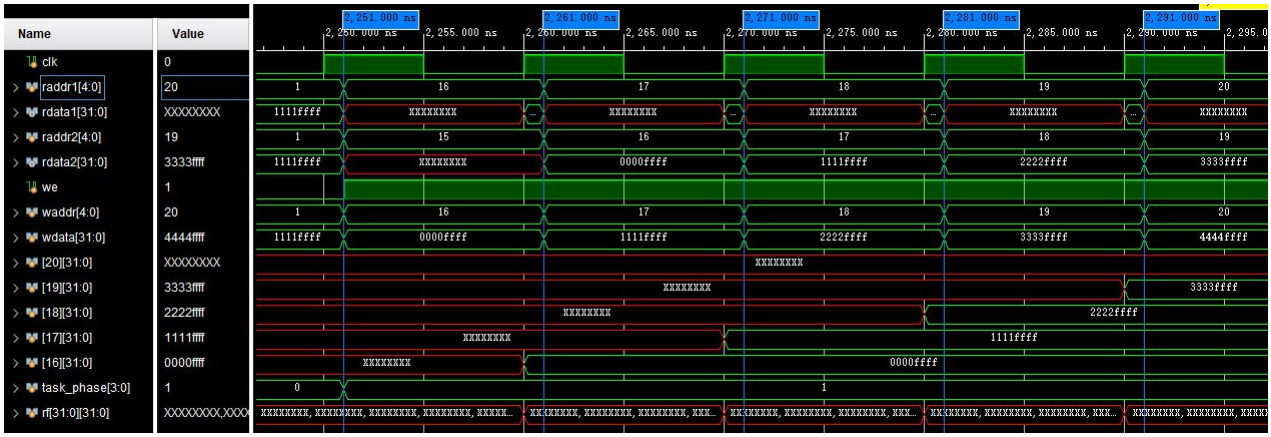

图 2: 寄存器堆部分仿真波形

根据图 2[①]，Marker 标记处写使能信号 `we` 拉高，`waddr`、`wdata` 发生改变，但寄存器堆内容 `rf` 并未立刻发生改变，而是在此后的第一个时钟信号上升沿发生改变（见 `[16]-[19]` 信号的变化），即写数据写入寄存器堆，符合同步时序过程的特征。

2.2.3 异步读

从源码可以看出读端口逻辑为组合逻辑，即不依赖 `clk` 信号，为异步过程。

```
// READ OUT 1
assign rdata1 = (raddr1==5'b0) ? 32'b0 : rf[raddr1];
// READ OUT 2
assign rdata2 = (raddr2==5'b0) ? 32'b0 : rf[raddr2];
```

同样观察图 2，读地址信号 `raddr1` 与 `raddr2` 在 Marker 处改变，读数据信号随之发生改变而不会等待至时钟上升沿。此外在时钟上升沿处，若写过程导致读端口对应的某寄存器内容改变则读数据信号也随之立刻改变[②]。

2.3 同步 RAM 和异步 RAM 仿真、综合与实现

2.3.1 仿真行为对比分析

通过行为仿真可以观察到“同步 RAM”、“异步 RAM”中的“同步”、“异步”是对读过程的描述，即 Block RAM（以下简称 BRAM）同步读数据，而 Distributed RAM（以下简称 DRAM）异步读数据。

由图 3、4 可见，BRAM 读数据在每个时钟周期上升沿到来后改变而非如 DRAM 那样随读地址的改变而改变，即读取过程中 BRAM 与时钟信号同步进行数据读取，读得结果相比访问地址改变延后了一个周期，而 DRAM 读取结果随访问地址异步变化，在访问地址变化的当拍获得读取数据。

[①]其中地址信号修改为十进制形式，其他信号数据进制为默认。
[②]其间存在由组合逻辑产生的延迟。

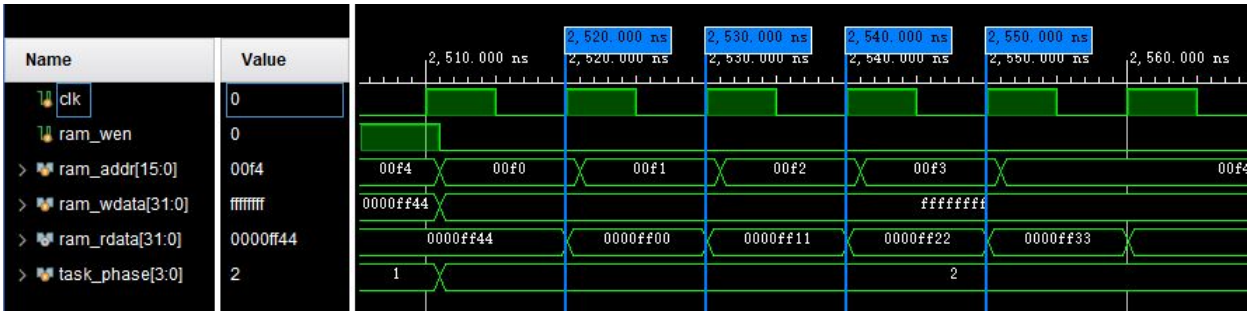

图 3: BRAM 读过程波形

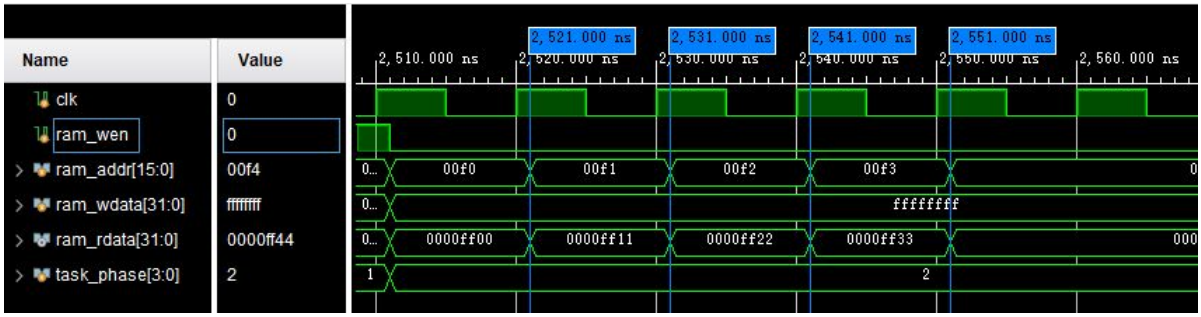

图 4: DRAM 读过程波形

在明确二者读过程时序特征后考察图 5、6 中二者写过程的波形变化。可以注意到由于同步读的特征，BRAM 在某一时间的写入信息需要在下一拍才能反映在读数据端口上。而对于 DRAM，在明确其异步读的基础上可以其写入时序为同步写，即在时钟上升沿到来（且 `wen` 使能为写使能）时，写数据写入 RAM，通过异步读端口反映在读数据端口上。

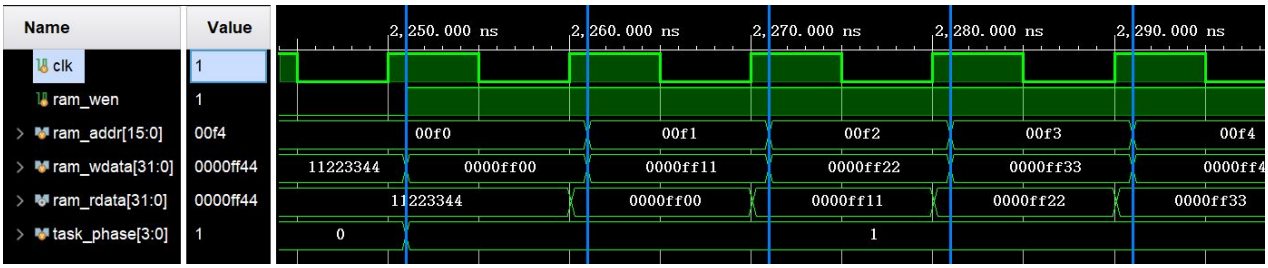

图 5: BRAM 写过程波形

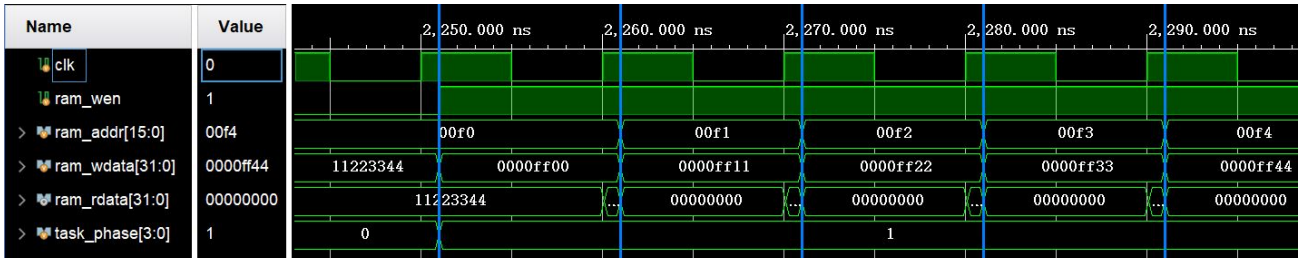

图 6: DRAM 写过程波形

2.3.2 时序、资源占用对比分析

在时序上，BRAM 的 WNS 为正值，时序满足极好，没有发生时序违约；而 DRAM 的 WNS 来到了 $-6.672\,\text{ns}$，违约值超过 $300\,\text{ps}$，根据课程讲义内容时序很糟糕，可能上板无法运行，Vivado 工具也给出了

有关时序的 Critical Warning。

在资源上，BRAM 主要使用 FPGA 内部集成的 RAM 资源，而 DRAM 在 LUT 资源上开销远多于 BRAM，即其主要使用 LUT 资源搭建逻辑电路。

由于块 RAM 一旦使用了一部分，其他部分不再能被其他模块利用，故而一般来说在构建较大的存储或时序要求较高时应用 BRAM，在小 RAM 与时序要求不高时使用 DRAM。

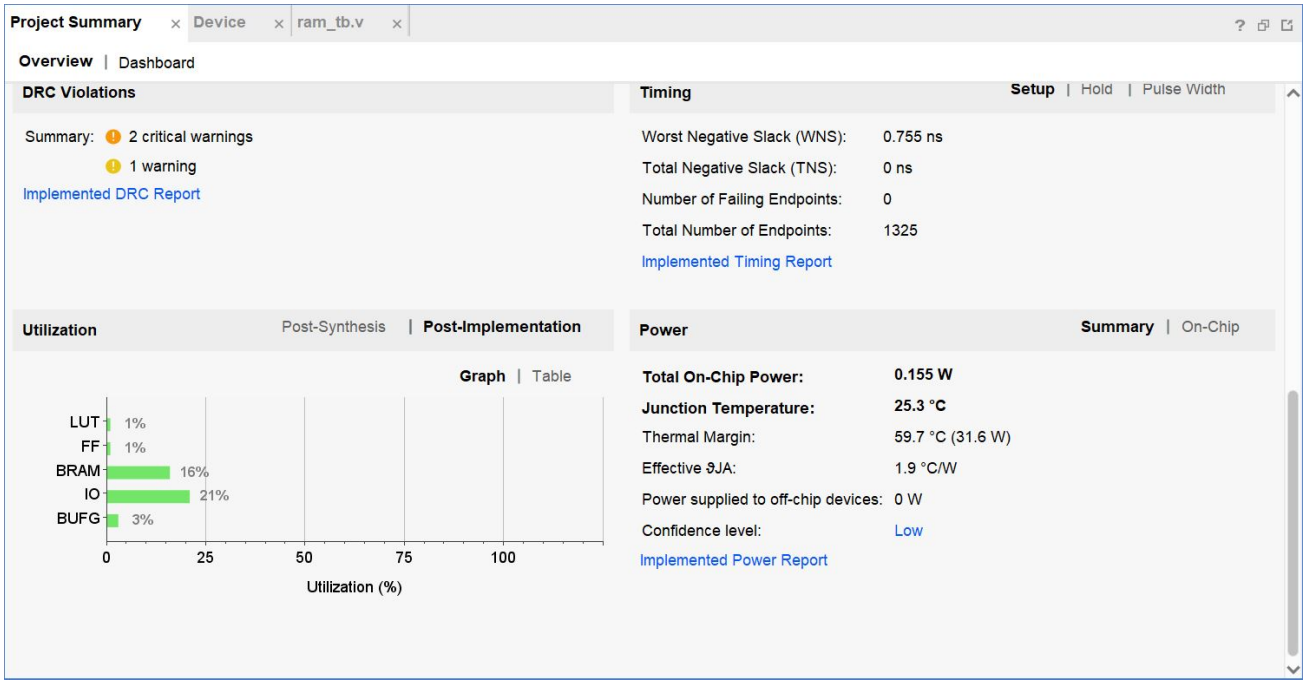

图 7: BRAM 实现报告总结

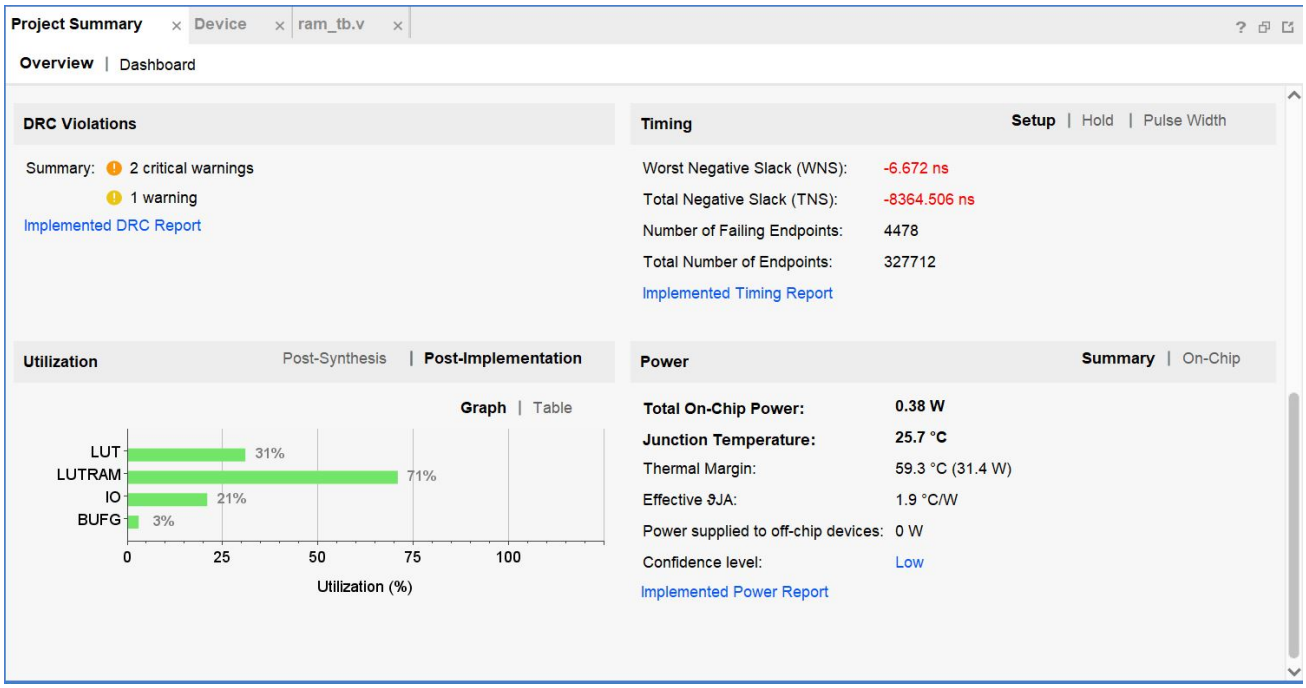

图 8: DRAM 实现报告总结

2.3.3 总结

综合考虑前两小节，在实际应用过程中需要综合考虑读写时序行为与资源利用在二者之间合理取舍，同时在相关模块的设计上协同调整。

2.4 数字逻辑电路的设计与调试

2.4.1 错误 1：信号为“Z”

2.4.1.1 错误现象

如下图所示，模块 show_sw 中信号 num_csn 始终为 Z，而子模块 show_num 中信号 num_csn 信号正常。

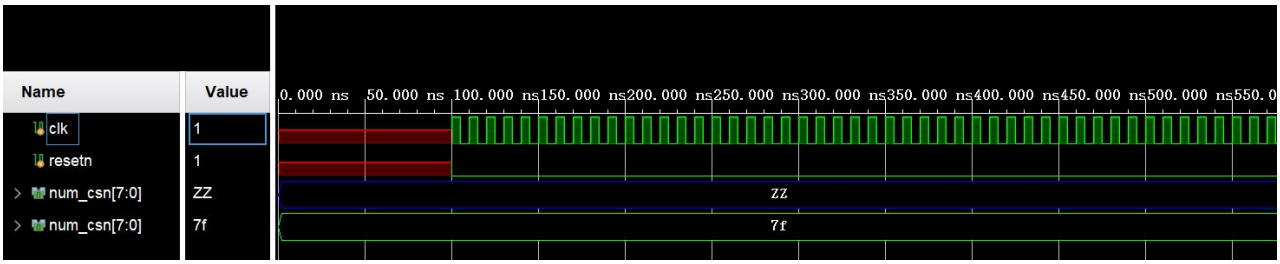

图 9: num_csn 信号为 Z

2.4.1.2 分析定位过程

根据讲义内容，可能是 wire 类型变量未被赋值或模块调用的信号未连接导致信号悬空，进而产生仿真波形中信号为 Z。考察与信号 num_csn 有关的逻辑，可注意到如下代码：

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

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

其中 num_scn 为未声明信号，取代了本该在此处的 num_csn 导致该信号悬空。

2.4.1.3 错误原因

拼写错误导致模块调用间连接出错，num_csn 信号悬空。

2.4.1.4 修正效果

修改 num_scn → num_csn，将两模块该信号连接，模块 show_sw 中 num_csn 恢复正常，与子模块中同名信号数据相同。

2.4.1.5 归纳总结

该错误来自编写代码时的拼写错误，避免该错误的方式除编写时更加专注外，还可以使用 verilator 作为 Linter，可得到错误拼写信号未声明的相关 warning 从而注意到该错误。

2.4.2 错误 2：信号为“X”

2.4.2.1 错误现象

如下图所示，多个信号为 X。

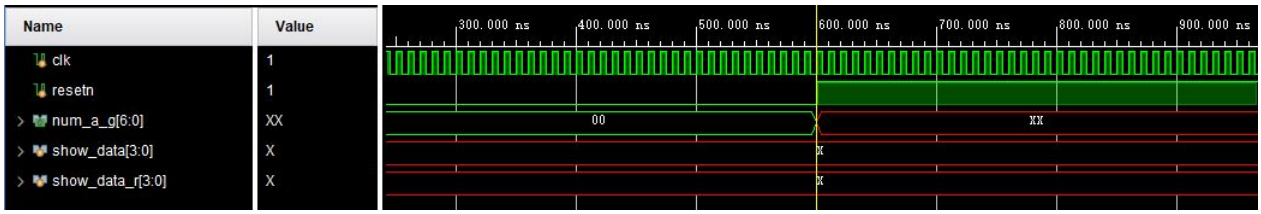

图 10: 多信号为 X 错误波形

2.4.2.2 分析定位过程

根据讲义内容，该类型错误来自 reg 类型未被赋值或者可能的多驱动代码，考察上图中为 X 的三个变量与其间逻辑关系，可注意到 `num_a_g` 与 `show_data_r` 依赖 `show_data`，而该寄存器有关的代码如下。

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

2.4.2.3 错误原因

`show_data` 寄存器未能正常赋值进而影响其他寄存器的值。

2.4.2.4 修正效果

取消该行注释，重新仿真，上述三个寄存器值正常获取，但遇到下一个错误。

2.4.2.5 归纳总结

本次实验中为了取消设计难度而通过注释的方式设置这样的错误，实际开发过程中可能会漏写了处理某个寄存器的 `always` 块进而导致该寄存器未被赋值，进而导致这种问题。故而实际开发过程中应牢记预先构建的数据通路，对电路各个元件有充分的认识。

2.4.3 错误 3：波形停止

2.4.3.1 错误现象

图 11: 波形停止

如上图所示，波形在 710 ns 处停止，取消仿真后停止在代码第 90 行。

2.4.3.2 分析定位过程

根据讲义内容，该类型错误来源为组合逻辑环，结合 Vivado 取消仿真的停止位置可注意到源代码 90 ~ 101 行存在组合逻辑环：

```
assign keep_a_g = num_a_g + nxt_a_g;

assign nxt_a_g = show_data==4'd0 ? 7'b1111110 :    //0
                 show_data==4'd1 ? 7'b0110000 :    //1
                    ...
                 show_data==4'd9 ? 7'b1111011 :    //9
                                   keep_a_g   ;
```

2.4.3.3 错误原因

nxt_a_g 与 keep_a_g 之间互相依赖，形成了组合环路。

2.4.3.4 修正效果

考察该处设计目的，可发现引入 keep_a_g 是为了在 show_data ⩾ 10 时传递当前值给 nxt_a_g，从而使得 show_data ⩾ 10 时显示上一次的数值，故而修改方式可为删去 keep_a_g 信号，将 nxt_a_g 在 show_data ⩾ 10 时缺省值改为 num_a_g 即可。

2.4.3.5 归纳总结

组合逻辑环的产生主要因为编写代码前未对电路具有充分的认识与设计，导致编写出混乱的环路代码，实际开发过程中应先对目标电路形成充分的认识之后再进行代码的编写。

2.4.4 错误 4：越沿采样

2.4.4.1 错误现象

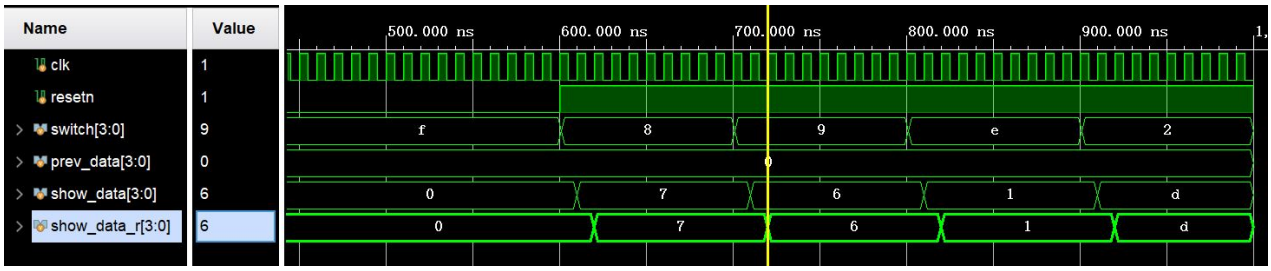

图 12: 越沿采样错误波形

如上图所示，prev_data 需要更新时始终认为 show_data 与 show_data_r 相等而不进行更新，始终保持值为 0。

2.4.4.2 分析定位过程

根据讲义内容该错误来自于非阻塞赋值与阻塞赋值的混用，检查各寄存器的赋值语句后可注意到 show_data_r 使用了阻塞赋值。

2.4.4.3 错误原因

混用非阻塞与阻塞赋值导致数据的时序关系混乱，对于在上升沿处需求赋值前处理数据的情况会导致错误。

2.4.4.4 修正效果

将 `show_data_r` 处赋值方式修改为非阻塞赋值即可。

2.4.4.5 归纳总结

遵循编写 RTL 代码的代码规范，在 `always` 块中的时序逻辑只使用非阻塞赋值。

2.4.5 错误 5：功能错误

2.4.5.1 错误现象

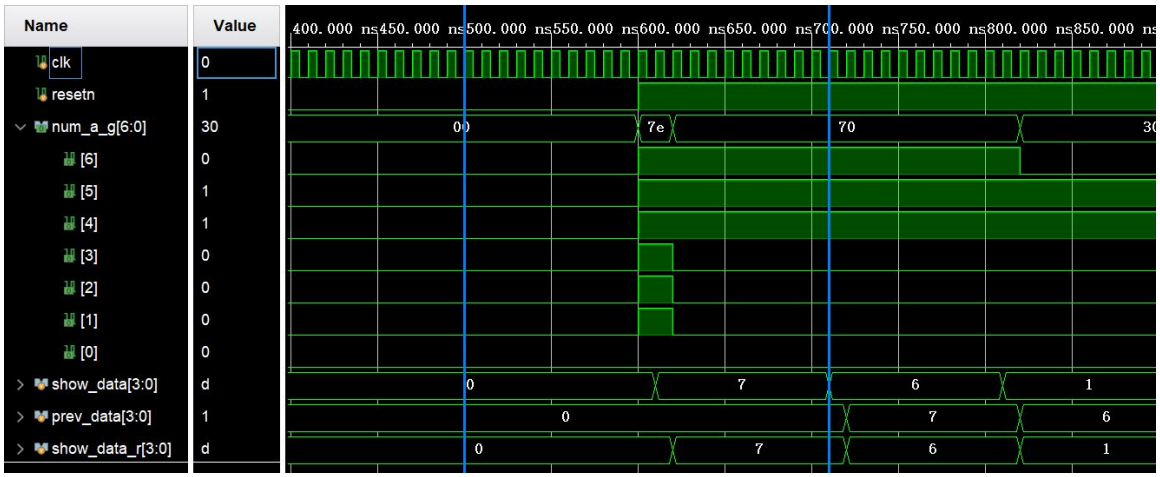

图 13: 功能错误

如图所示前一个 Marker 处复位状态下数码管不显示，需求显示 0；第二个 Marker 处 `show_data` 为 6 但 `num_a_g` 仍为此前 7 的显示。

2.4.5.2 分析定位过程

该处功能错误来自于 `num_a_g` 相关逻辑的错误，考察代码中相关逻辑的实现即可注意到 `num_a_g` 的复位值错误且 `nxt_a_g` 中遗漏了 `show_data` 为 6 时的情况。

```
always @(posedge clk)
begin
    if ( !resetn )
    begin
        num_a_g <= 7'b0000000;  // no display
    end
    else
    begin
        num_a_g <= nxt_a_g;
    end
end

//keep unchange if show_dtaa>=10

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
                    // missing 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;
```

2.4.5.3 错误原因

设计过程中对需求未有充分的认识，在细节把控上未有充足的构思。

2.4.5.4 修正效果

将 `num_a_g` 的复位值修改为 `7'b1111110`,补充 `show_data` 为 6 时的数码管阵列的赋值 `7'b1011111`。修改后综合上板，验证需求即可。

3 实验总结

本次实验主要对 Vivado 与常规的数字逻辑设计过程进行了进一步的了解，在此前数字电路与组成原理课程的基础上更系统完善地认识了数字逻辑设计的流程、常见 bug 与调试方式。