实验一报告

学号 2017K8009929044
姓名 李昊宸
箱子号 33

一、实验任务（10%）

1. 了解寄存器堆的原理。
2. 了解同步 RAM 和异步 RAM 的原理。
3. 理解同步 RAM 和异步 RAM 的区别。
4. 掌握调用 Xilinx 库 IP 实例化 RAM 的设计方法。
5. 学会识别常见波形异常，理解其产生原因，并能修正。
6. 熟悉并运用 verilog 语言进行电路设计。

1 二、实验设计（0%）

针对 Lab2 实验，无实验设计内容，该部分不需要描写。

三、实验过程（90%）

（一）实验流水账

2019/9/5 11:00 – 2:00 任务三调试 debug，书写实验报告

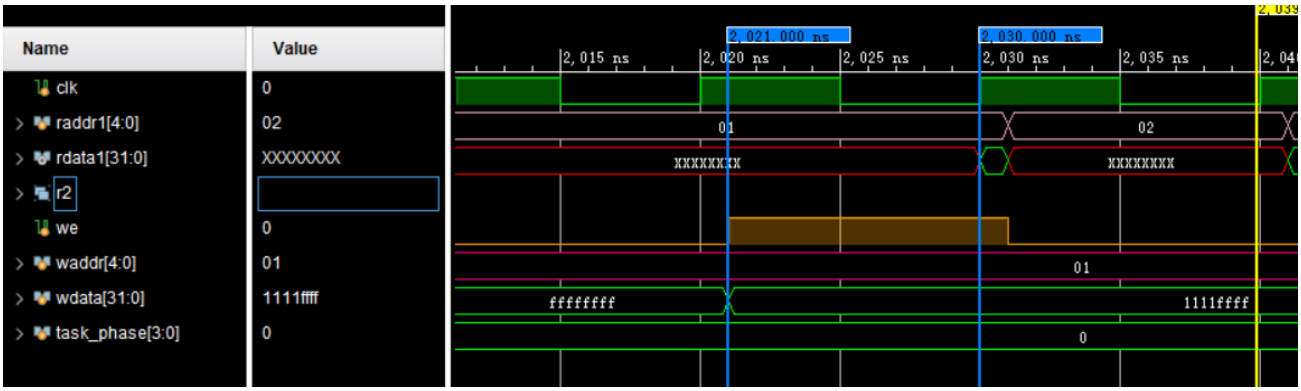
2019/9/6 13:00 – 19:00 任务一、任务二调试，书写实验报告

（二）子任务一

实验目标：寄存器堆仿真。

实验方式：建立寄存器堆工程，加入我们提供的寄存器堆的源码和仿真文件，建立工程， 并进行仿真，得到正确的波形，根据波形描述寄存器堆的读写行为。

实验效果：



图一　寄存器堆仿真

这张波形图很典型的反映了寄存器堆异步写同步读的特性。下面先给出信号表

时钟：　　clk 1 input 时钟信号

读端口一：raddr1 5 input 寄存器堆读地址 1

　　rdata1 32 output 寄存器堆读返回数据 1

读端口二：raddr2 5 input 寄存器堆读地址 2。

　　rdata2 32 output 寄存器堆读返回数据 2

写端口：　we 1 input 寄存器堆写使能

　　waddr 5 input 寄存器堆写地址

　　wdata 32 input 寄存器堆写数据

方便起见，图中采用分组（Division）方式将读端口二的信号设为不可见，只分析读端口一的行为。

在 2021ns 之前，写使能信号 we 始终处于低电平状态，故无数据写入寄存器堆，并且从寄存器中读取的数据也一直为 X（因为信号没有初值）；

2021ns 时（左侧第一条蓝色标线），we 拉高，数据可写入。此时写数据 wdata 为 0x1111ffff，waddr 为 01，但是此时时钟并非为上升沿，故数据一直不可被写入，读返回数据也始终为 X；

2030ns 时（右侧第一条蓝色标线），时钟迎来上升沿。此时写数据 wdata 为 0x1111ffff，waddr 为 01，数据被写入 01 号寄存器。此时读地址 raddr 为 01 号寄存器，读返回数据 rdata 为 0x1111ffff。

2031ns 时，raddr 更改为 02 号寄存器，因为 02 号寄存器从未被写过，于是读返回数据 rdata 为 X。

以上便是对寄存器堆“同步写，异步读”的实例分析。

（三）子任务二

实验目标：同步、异步 RAM 仿真、综合实现

实验方式：本实验要求对同步、异步 RAM 各自建立一个工程，调用 Xilinx 库 IP 实例化同步、异步 RAM，但是都会提供 一个设计的顶层文件，将他们封装成相同的模块名和接口。

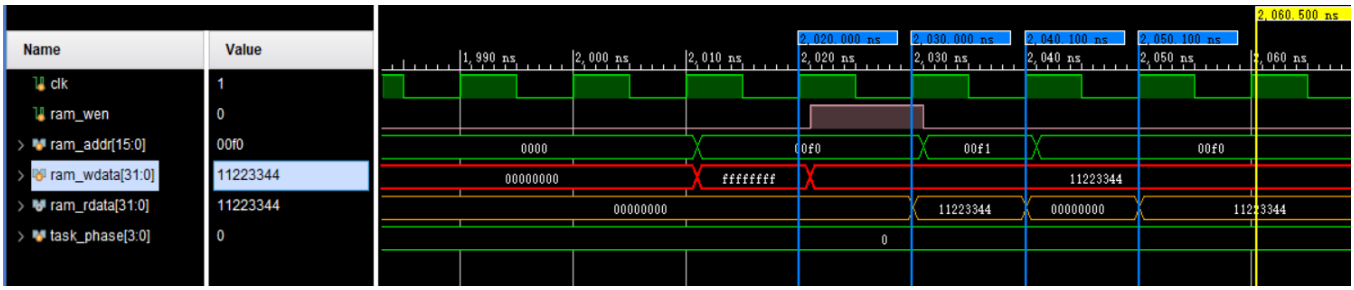
实验声明：

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 的写数据信号，表示写入的数据。
ram_rdata	32	Output	RAM 的读数据信号，表示读出的数据。

图二 ram 信号示例

1、仿真行为对比分析

同步 ram：



图三 Block_ram 仿真波形图

同步 ram 的主要特点为其读写都依赖于时钟的上升沿。

在 2020ns 时（左侧第一条蓝色线），迎来时钟上升沿，此时使能信号 wen 还没有拉高，故此时处于读状态。地址 ram_addr 此时为 0x00f0，读出的是空值 0x00000000；

在 2021ns 时迎来 wen 拉高，但是此时还没有遇见下一次上升沿，故信号无变化；

在 2030ns 时（左侧第二条蓝色线），迎来时钟上升沿。此时 wen 信号拉高，处于写读状态。向 0x00f1 地址写入数据 0x11223344，此时读头读出数据 0x11223344；

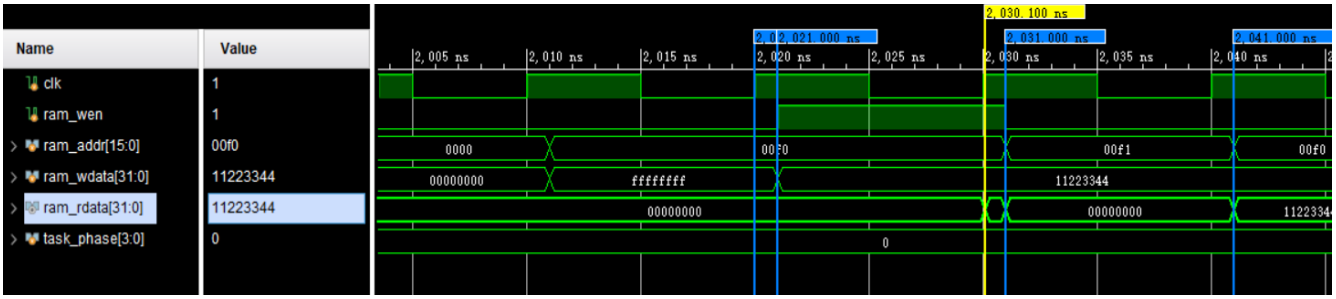
在 2031ns 时，地址更换为 0x00f1，但由于未遇见上升沿，故数据无变化；

在 2040ns 时（右侧第二条蓝色线），迎来上升沿，wen 信号拉低，为读状态。读出空值数据 0x0000000；

在 2041ns 时，地址更换为 0x00f0，但由于未遇见上升沿，故数据无变化；

在 2050ns 时（右侧第一条蓝色线），迎来上升沿，wen 信号拉低，为读状态。读出数据 0x11223344。

异步 ram：



图四 Distributed_ram 仿真波形图

分布式异步 ram 的主要特征为同步写，异步读。

在 2020ns 时（左侧第一条蓝线），时钟出现上升沿。使能信号 wen 拉低，为读状态。没有什么事情发生。

在 2021ns 时（左侧第二条蓝线），使能信号 wen 拉高，状态为写状态。地址 ram_addr 指向 0x00f0，写数据为 0x11223344。但是此时不是上升沿，故不能写入。

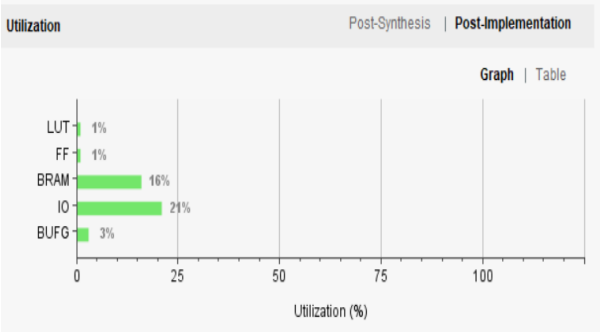
在 2030ns 时（黄线），时钟出现上升沿，wen 信号为拉高，数据写入 0x00f0。读头读出数据为 0x11223344。

在 2031ns 时（右侧第二条蓝线），地址转变为 0x00f1，读头数据立即更改为空值 0x00000000。

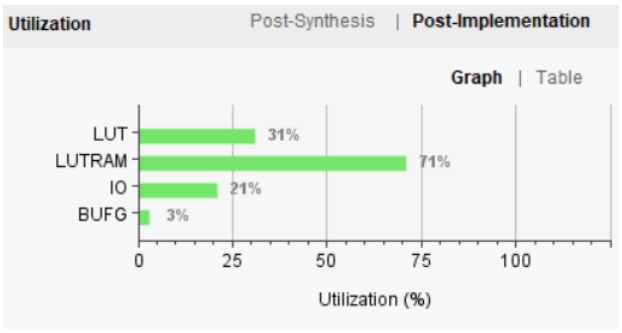
在 2031ns 时（右侧第二条蓝线），地址转变为 0x00f0，读头数据立即更改为数据 0x11223344。

2、时序、资源占用对比分析

（1）fpga 使用率



图五 Block_ram



图六 Distributed_ram

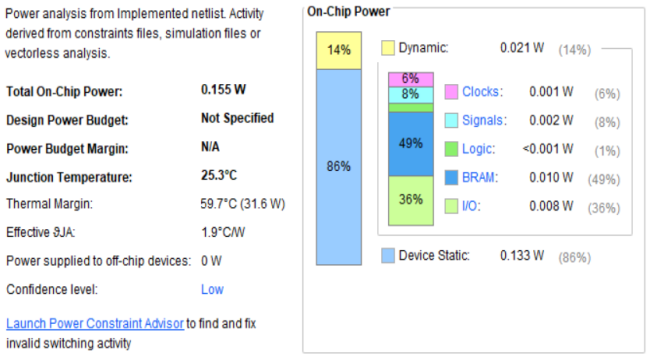
无论是在同步式还是异步式 ram 中，fpga 板上 IO 设备的使用率均为 21%，BUFG 全局时钟缓冲器的使用率均为 3%。

同步式中，有 16%的 BRAM 被用来做 Block_ram 的存储片；异步式中，在 SLICEM 中有 71%的 LUTRAM 被用来做 Distributed_ram 的存储片。

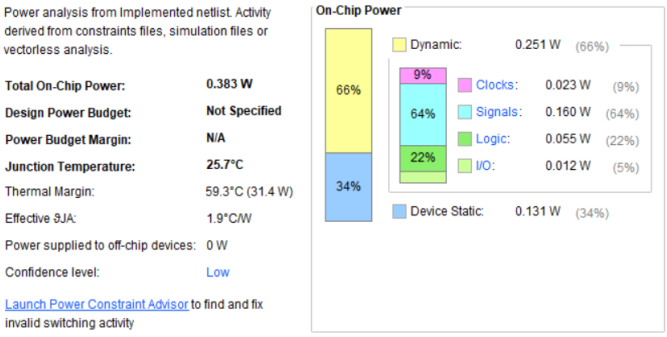
对于 LUT 查找表的消耗，同步式只使用了 1%，而异步式使用了 31%。

此外，由于存在同步读写，同步式中用到了 FF 触发器。

（2）能耗



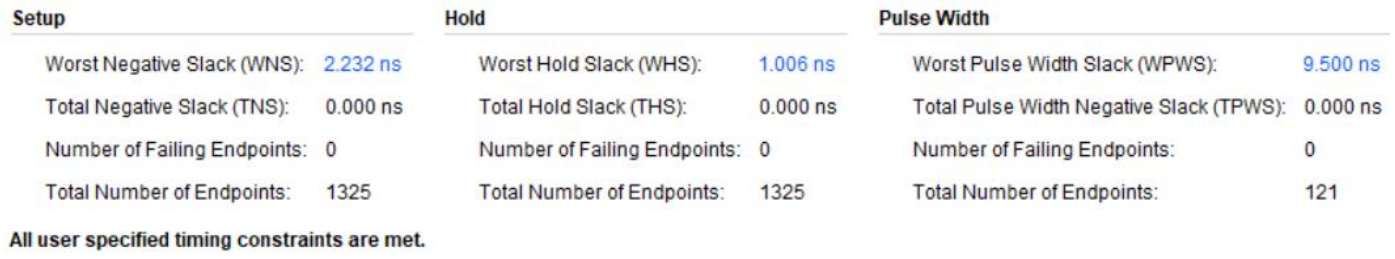
图七 Block_ram



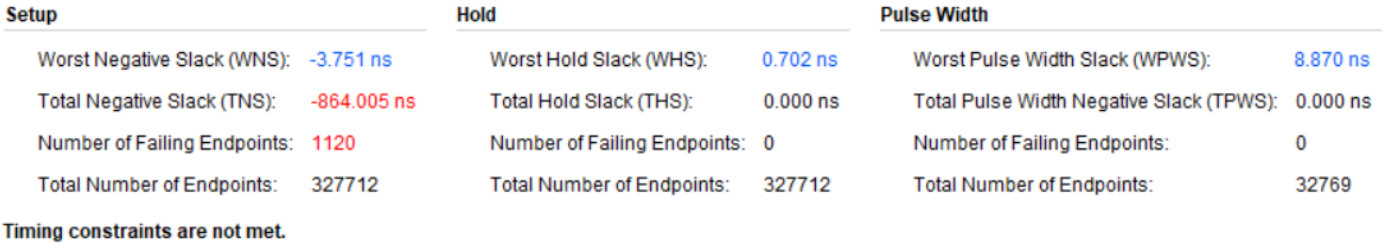
图八 Distributed_ram

看上去同步式的功率比异步式的功率小，大概只有不到一半。在同步式中，耗电来源主要来自于静态损耗，动态损耗很低，这可能与同步式的信息转换仅在时钟上升沿触发时发生有关。在异步式中，动态损耗达到 66%，其他的耗电与同步式相近，但是各种信号（Signals）的损耗很高，这可能与异步中异步读信号变化需要始终检测有关。

（3）时序分析



图九 Block_ram



图十 Distributed_ram

查看 Worst Negative Slack（WNS）（违反时序限制最大的路径延迟），看到同步式的最长时序路径提前周期 2.232ns 完成，而异步式最长会超时 3.751ns，并且超时比率达到了 0.3%，如果不修改会导致严重的滞后。这导致真实制作芯片时，在同样的配置下，异步式的主频将被迫降低，效率总的来说也将降低。其他几项（WHS、THS、WPWS 等）二者差距不大，但都是同步式略胜一筹。

3、总结

同步式和异步式都有自己的特点，但在总体评测下：

（1） 同步式在芯片占用率上水平较低，所用器件成本较低

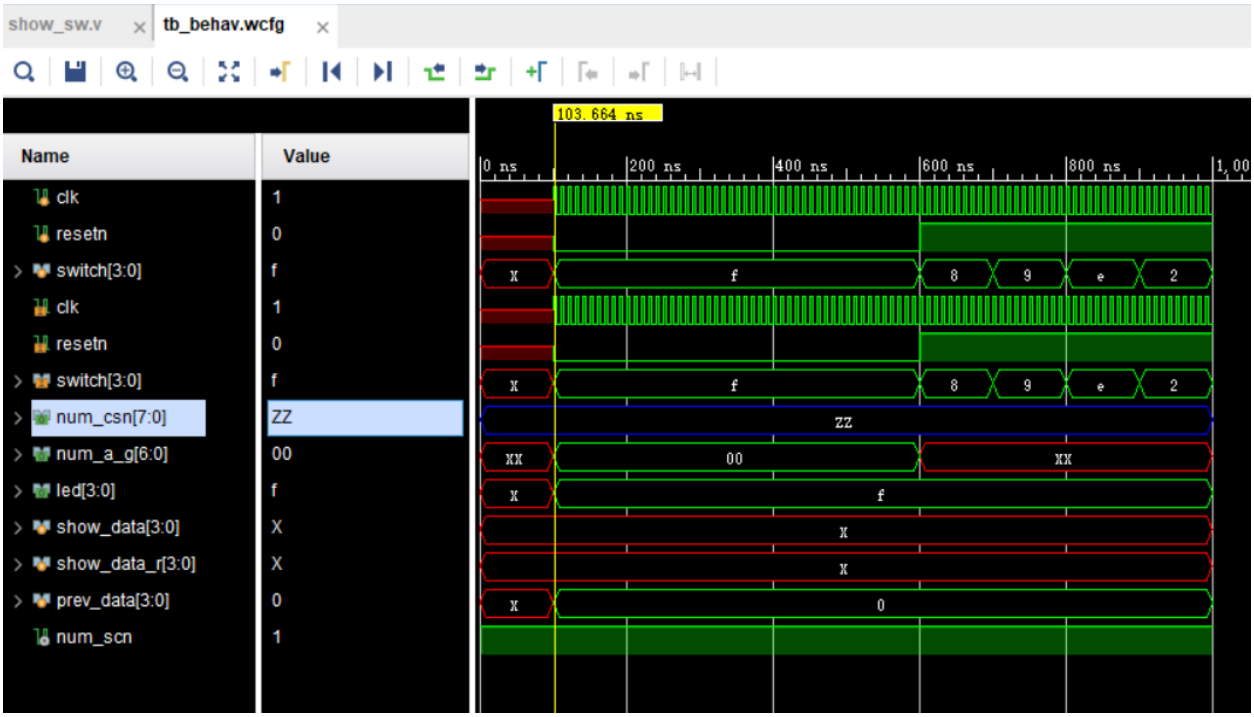
（2） 同步式在能耗上有优势

（3） 同步式在效率上更有利，制作 cpu 时主频可以制作的更高

（四）子任务三

1、错误 1：信号为 Z

（1）错误现象



图十一 信号为 z

num_csn 信号出现 ZZ 波形。

（2）分析定位过程

Z 信号表示出现高阻。

num_csn 信号为 8 位宽信号，类型为 wire 类型。正常情况下 wire 类型出现 Z 信号有两种可能：1）未用 assign 语句赋初值 2）调用模块时没有连接到端口。

针对情况一进行检查，发现在子模块 show_num 中已有赋值语句 *assign num_csn = 8'b0111_1111*，于是排除情况一。

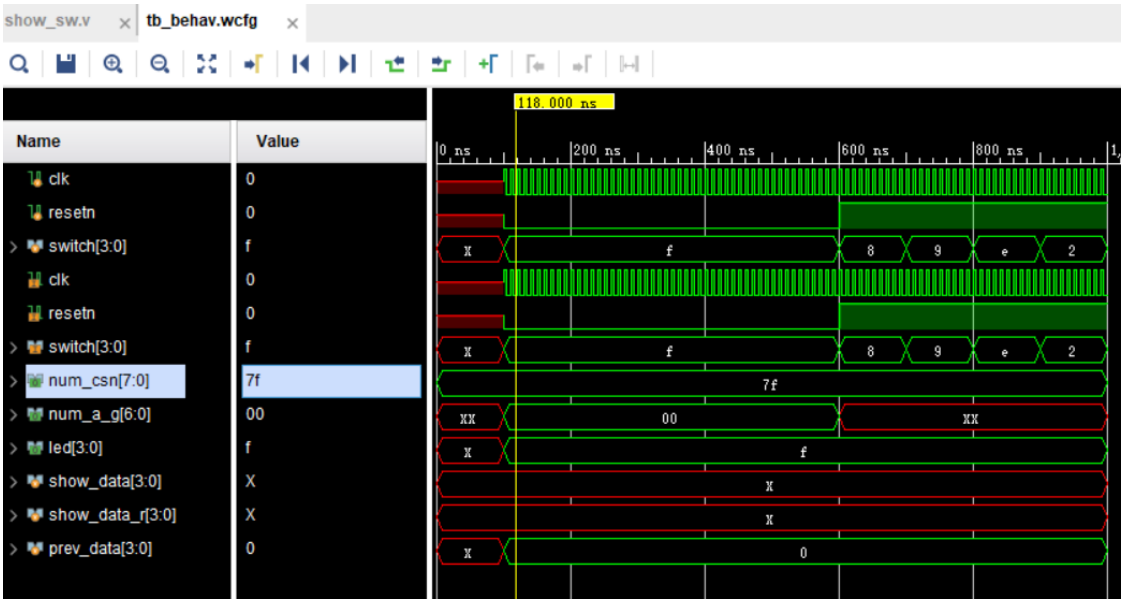
针对情况二进行检查，发现在主模块调用模块 show_num 时，端口 *.num_csn (num_scn),*处出现未知信号 *num_scn*，怀疑此处拼写错误导致问题发生。

（3）错误原因

端口变量连接错误。

（4）修正效果

将 *num_scn* 修改为 *num_csn* 后问题得到解决。



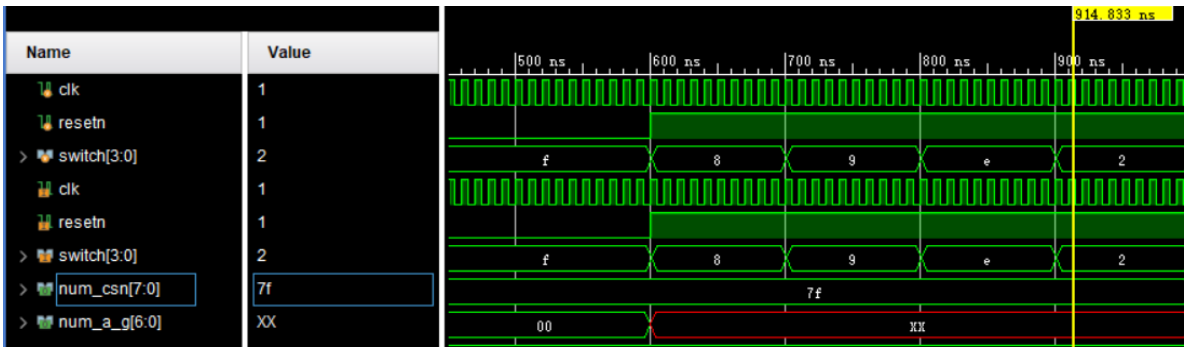
图十二 信号为 z 得到修正

（5）归纳总结（可选）

以后写完程序后先用 ctrl+f 检查一下有没有只出现过一次的变量，排除书写错误的情况。

2、错误 2：信号为 X

（1）错误现象



图十三 信号为 X

num_a_g 信号在 600ns 后出现 XX 波形

（2）分析定位过程

X 信号表示出现不定值，这种问题的出现主要有两种情况：1）声明为 wire 型的变量从未被赋值 2）写成了多驱动的代码。

num_a_g 信号为 reg 型输出变量。检查其在子模块 show_num 的赋值逻辑，发现它在起初 reset 信号为 0 时有着正常的值 7‘b0000000，但在 reset 信号拉高之后出现 X 信号。继续寻找，发现赋值语句 *num_a_g <= nxt_a_g；* 继续寻找上一级赋值，在对 nxt_a_g 的赋值中，找到同样为 X 信号的变量 show_data。该变量为 wire 型，故推断为该变量未赋初值。随后发现被注释掉的赋值语句 *show_data <= ~switch；*

（3）错误原因

wire 型变量从未被赋值。

（4）修正效果

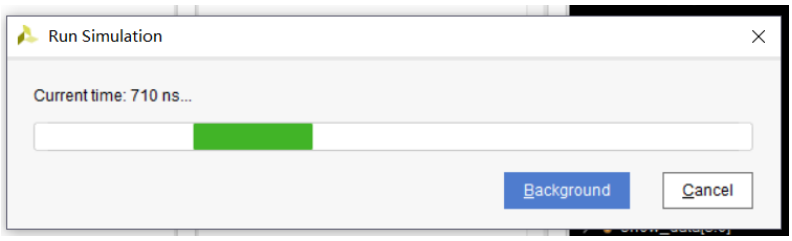
将注释号删除后所有的 X 信号消失。

（5）归纳总结（可选）

在每次定义完变量之后，立刻检查是否对其有赋值或者是否多变量对其赋值。

3、错误 3：波形停止

（1）错误现象



图十四 波形停止

仿真波形停止在 710ns。

（2）分析定位过程

波形停止的原因可能是因为 RTL 代码中存在组合环路。运行综合，在 warning 报错中发现有如下错误：[Synth 8-326] inferred exception to break timing loop: 'set_false_path -through \u_show_num/keep_a_g_carry /。于是定位到错误源自于信号 keep_a_g。其赋值语句为 *assign keep_a_g = num_a_g + nxt_a_g;* 推测此处存在组合环。在 nxt_a_g 的赋值语句中，我们发现在 show_data == 4’d6 以及大于等于 4’d10 时，nxt_a_g 被赋值为 keep_a_g，这样是不正确的，会出现组合环。

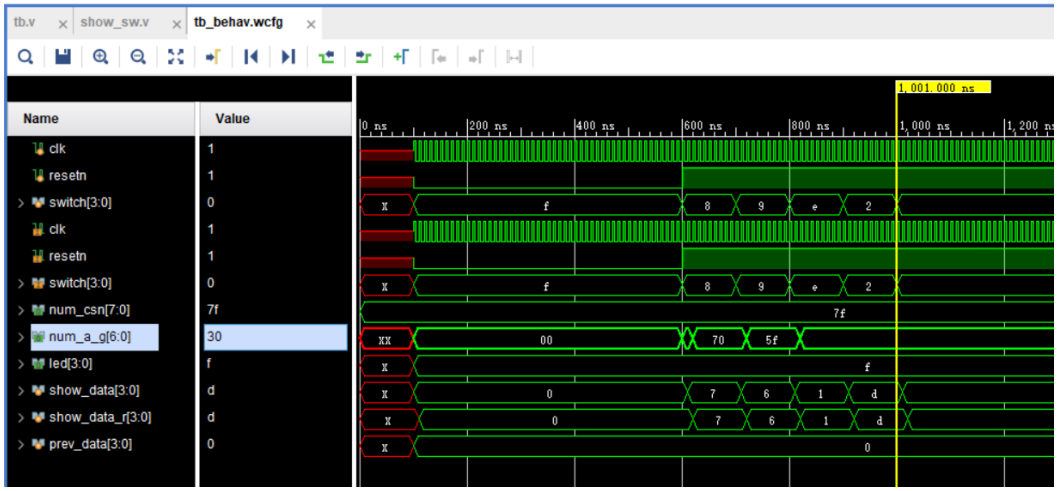
（3）错误原因

RTL 代码中存在组合环路。

（4）修正效果

首先，修改 keep_a_g 的赋值语句为 *assign keep_a_g = nxt_a_g;*另外，在 show_data == 4’d6 时，补充 nxt_a_g

的赋值为对应数字 6 的 7’b1011111 。随后波形正常。



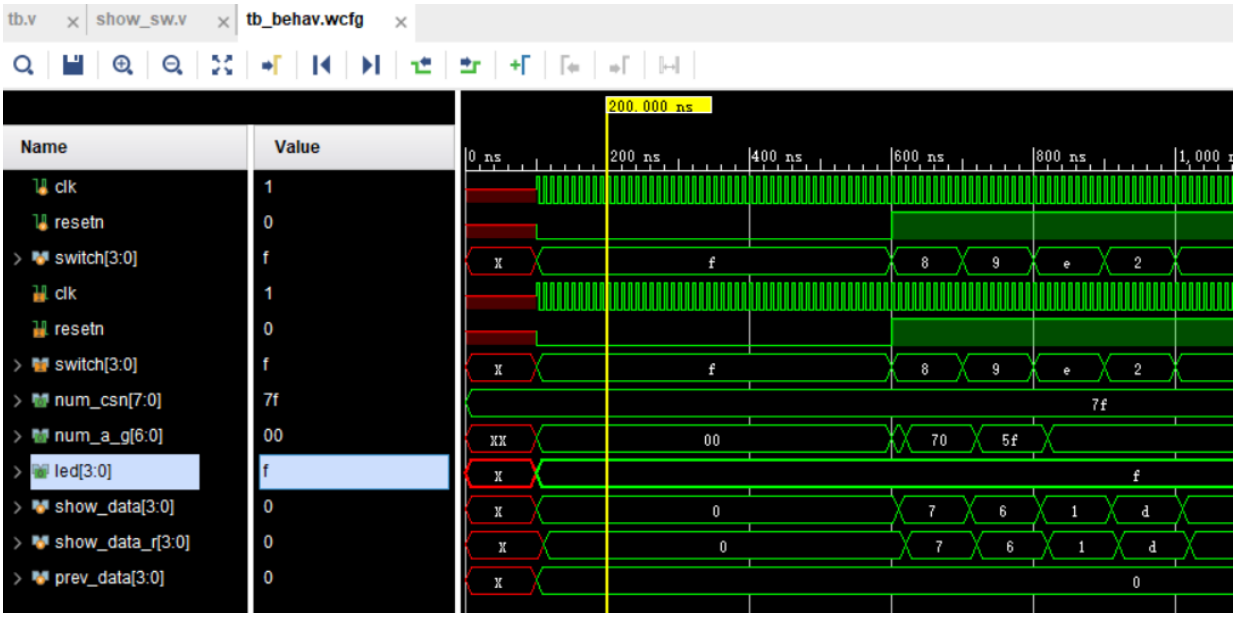
图十五 波形停止修正后

（5）归纳总结（可选）

嗯。。。组合环是最难检查的一个部分，在书写代码的时候注意如果出现随时间变化有时维持自身不变，有时由其他信号驱动的变量，尽量用 reg 型定义，用 always 块书写。

4、错误 4：越沿采用

（1）错误现象



图十六 越沿采用

led 信号一直停在 f。

（2）分析定位过程

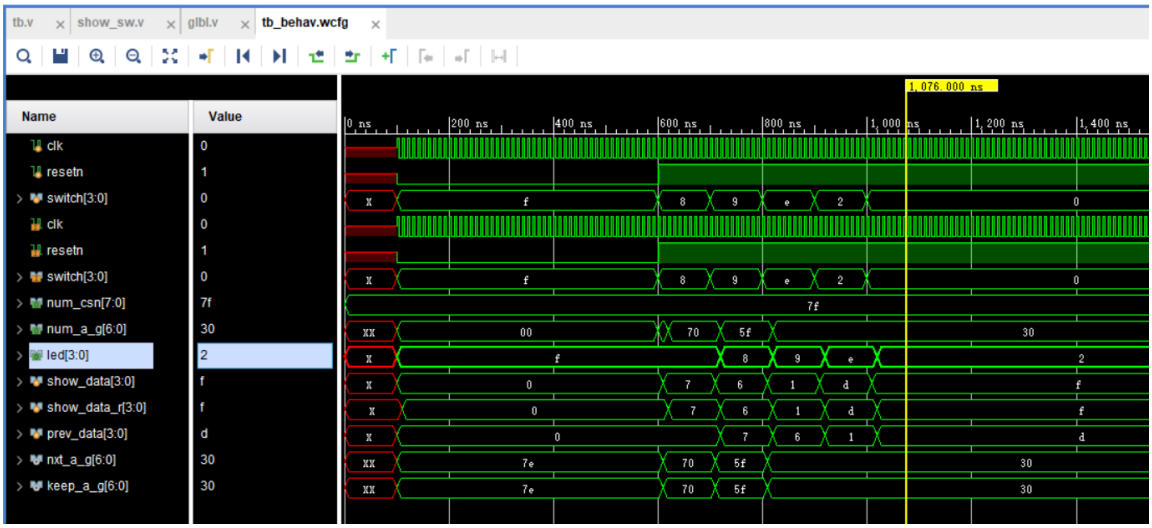
仔细观察，led 信号的作用是将上一次操作的结果表示出来，但是目前波形是始终维持在开机时的 f 状态。推测为在进行延时转换时出现了越沿采用。观察与 led 赋值有关的变量，依次找到 *assign led = ~prev_data; prev_data*

<= show_data_r; show_data_r = show_data; 找到问题所在：阻塞赋值。

（3）错误原因

在 always 块中采用了阻塞赋值。

（4）修正效果



图十七 越沿采用修正后

修正之后，出现正常波形。

（5）归纳总结（可选）

牢牢记住 always 块中只能出现<=。

5、错误 5：逻辑错误

（1）错误现象

在实际测试中，出现了从别的信号拨到数字 6 时，led 灯与数码管不正常工作。

（2）分析定位过程

问题出在对信号 keep_a_g 赋值时未考虑到等于 6 的情况。该 bug 实际上在错误 3 时顺便一起解决，具体修正详见错误 3 部分。

（3）错误原因

略

（4）修正效果

略

（5）归纳总结（可选）

略

6、错误 6：不可生成比特流文件

（1）错误现象

在运行 genbitstream 命令时，出现 write-bitstream error，查看错误日志和综合文件后发现出现 timing loop。

（2）分析定位过程

进一步查看 timing loop 问题，引导至 show_num 子模块中的 reg 类变量 num_a_g 出现环路。考虑到语句 *num_a_g <= nxt_a_g;*可以确定问题大概率在 nxt_a_g 的赋值上。

（3）错误原因

观察赋值，我们发现了这样的组合环路结构：*assign keep_a_g = nxt_a_g; 以及 assign nxt_a_g = (……) : keep_a_g ;*于是我将 keep_a_g 更改为 wire 信号，并采用下面的方式赋值：

```
reg [6:0] keep_a_g;
always @(posedge clk)
begin
   keep_a_g <= nxt_a_g;
end
```

（4）修正效果

成功消除组合环，上板测试结果也正确。

（5）归纳总结（可选）

略

四、实验总结（可选）

实验做起来不是很难。

但是实验报告写起来好麻烦啊！

老师你看它有 4000 个字！

我好难.jpg