**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

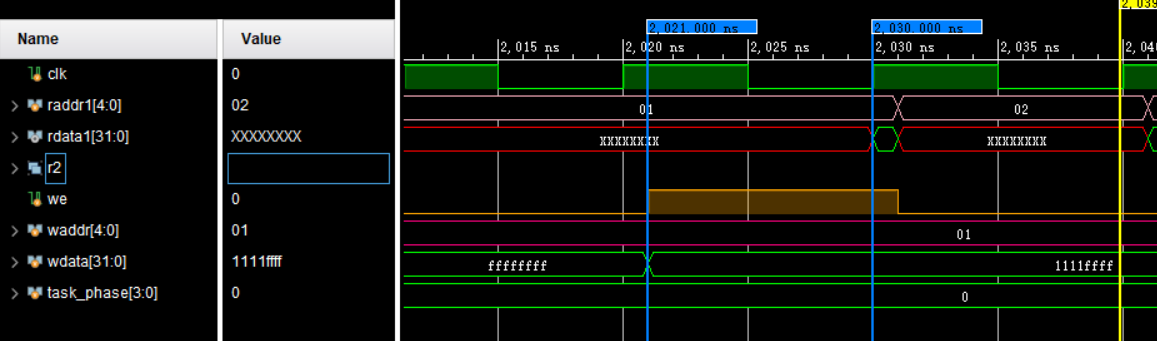
2019/9/6 13:00 – 19:00 任务一、任务二调试，书写实验报告

（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。

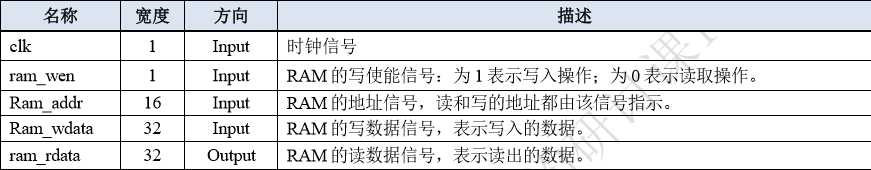
以上便是对寄存器堆“同步写，异步读”的实例分析。

（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。

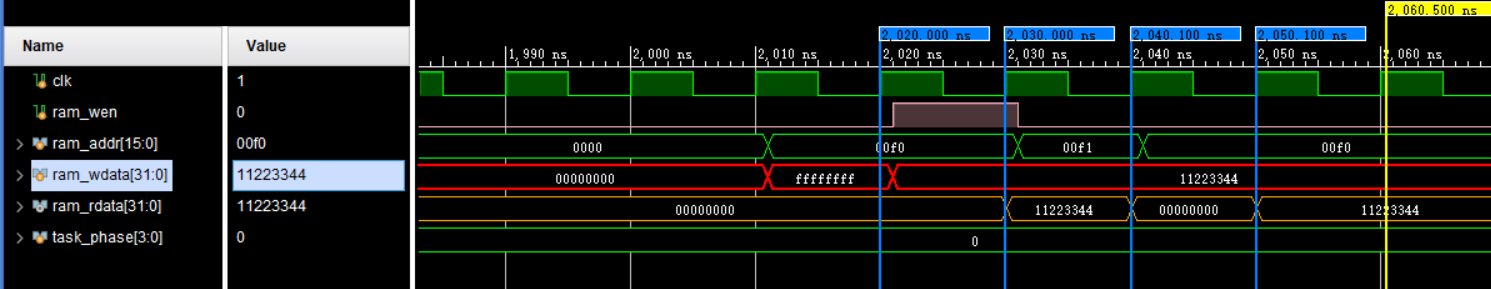
实验声明：



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；

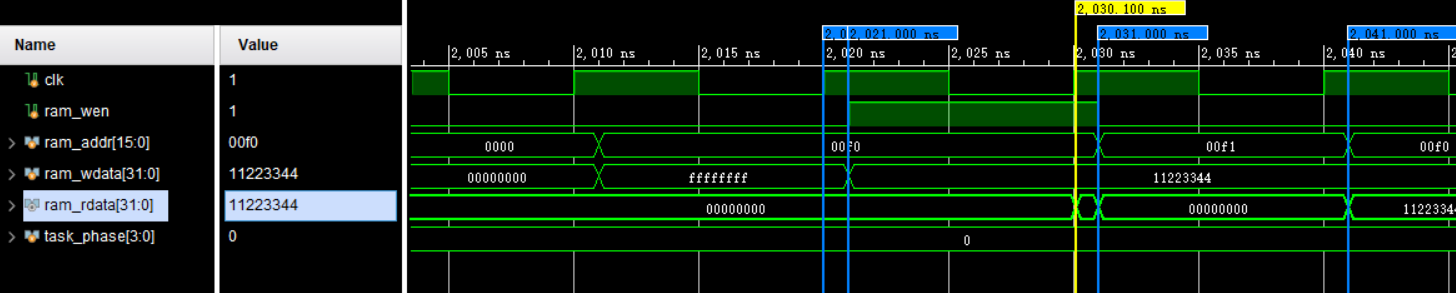
在2031ns时，地址更换为0x00f1，但由于未遇见上升沿，故数据无变化；

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。

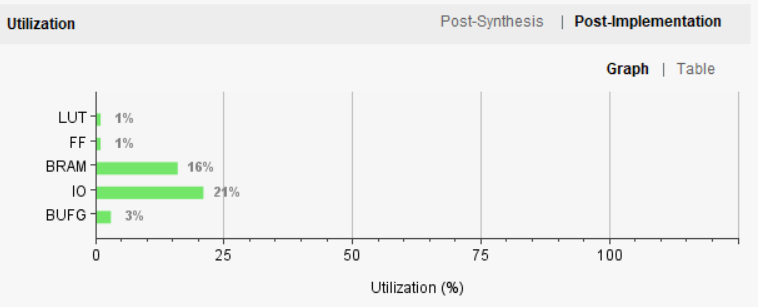
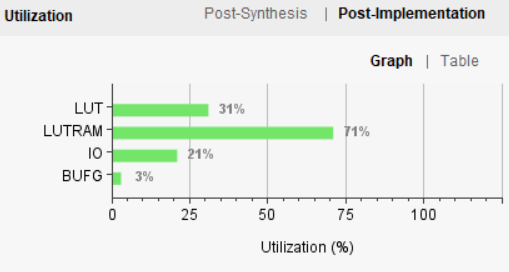
在2030ns时（黄线），时钟出现上升沿，wen信号为拉高，数据写入0x00f0。读头读出数据为0x11223344。

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。

1. 时序、资源占用对比分析

（1）fpga使用率



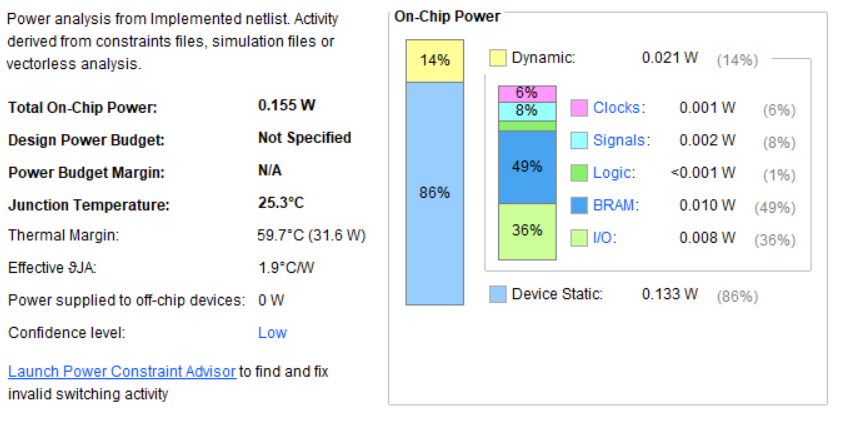
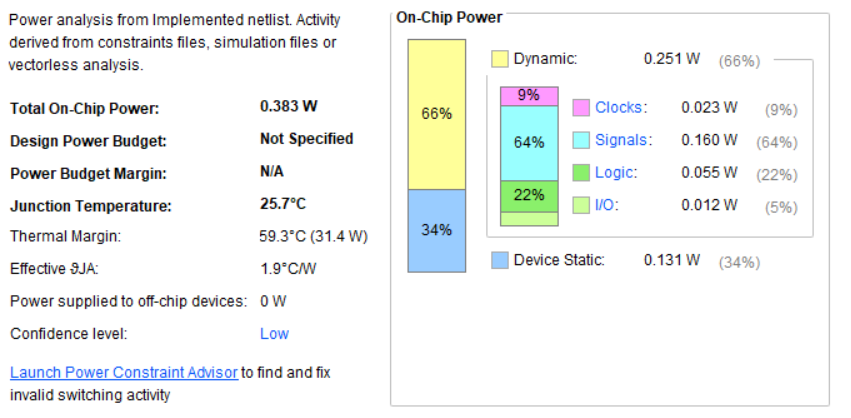
图五 Block\_ram 图六 Distributed\_ram

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%。

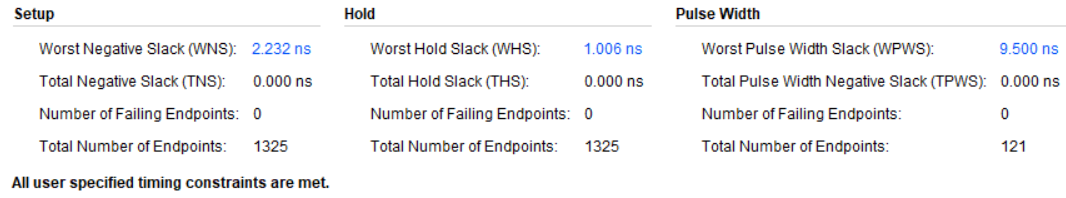
此外，由于存在同步读写，同步式中用到了FF触发器。

（2）能耗

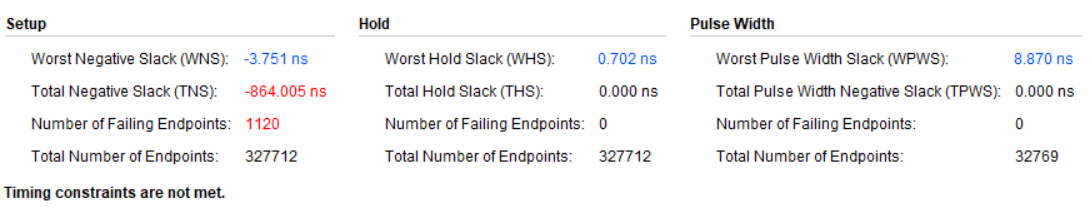
图七 Block\_ram 图八 Distributed\_ram

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。

1. 总结

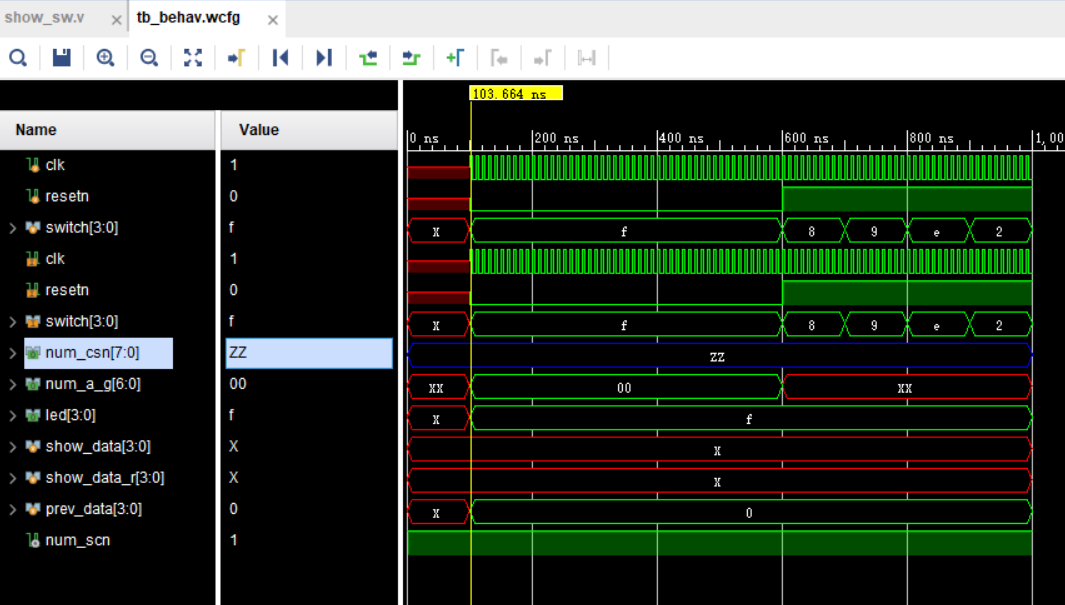
同步式和异步式都有自己的特点，但在总体评测下：

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。

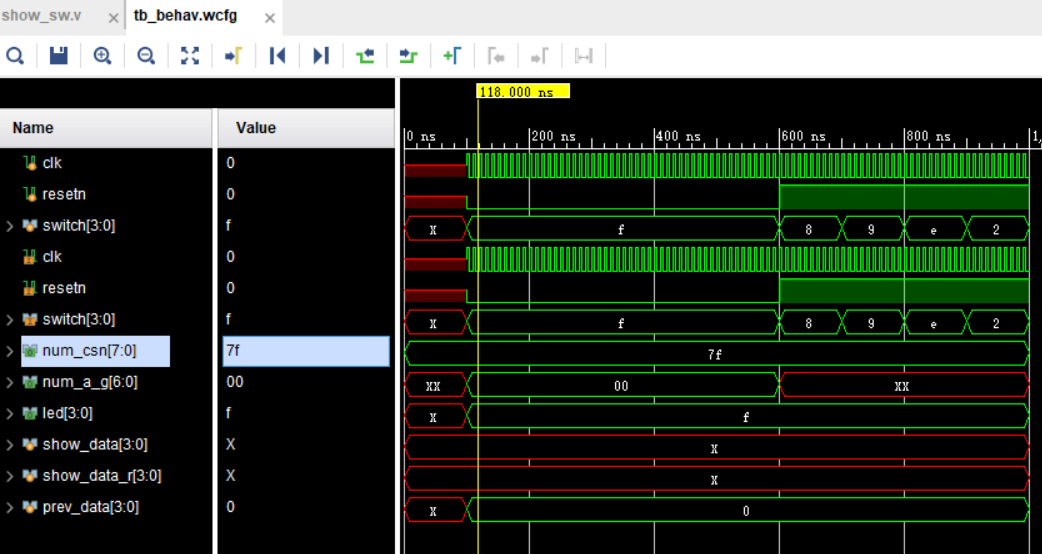
针对情况二进行检查，发现在主模块调用模块show\_num时，端口 *.num\_csn (num\_scn ),*处出现未知信号*num\_scn*，怀疑此处拼写错误导致问题发生。

（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。



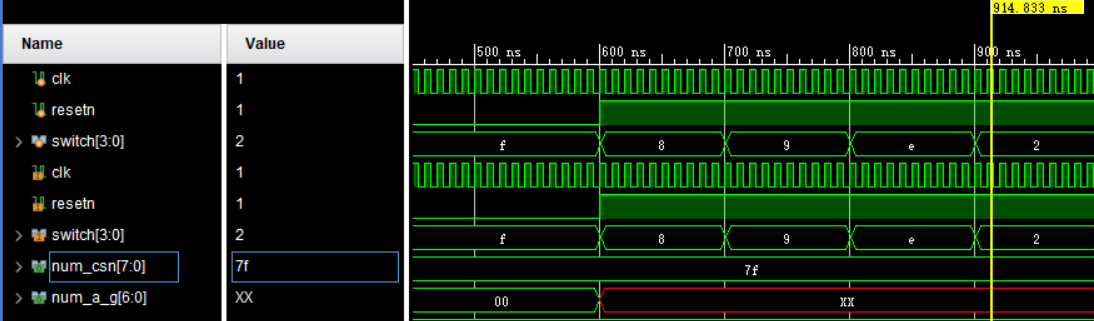
图十二 信号为z得到修正

（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

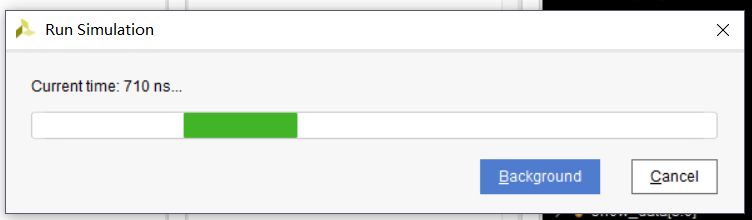
将注释号删除后所有的X信号消失。

（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

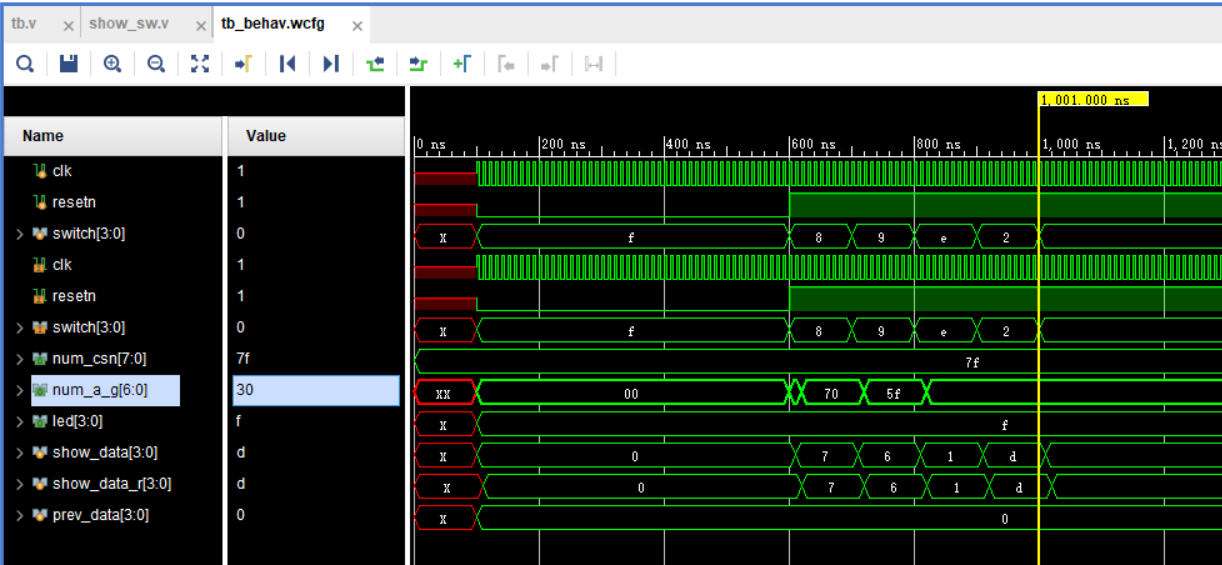
波形停止的原因可能是因为RTL代码中存在组合环路。运行综合，在warning报错中发现有如下错误：[Synth 8-326] inferred exception to break timing loop: 'set\_false\_path -through \u\_show\_num/keep\_a\_g\_carry /。于是定位到错误源自于信号keep\_a\_g。其赋值语句为*assign keep\_a\_g = num\_a\_g + nxt\_a\_g;* 推测此处存在组合环。在nxt\_a\_g的赋值语句中，我们发现在show\_data == 4’d6以及大于等于4’d10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。



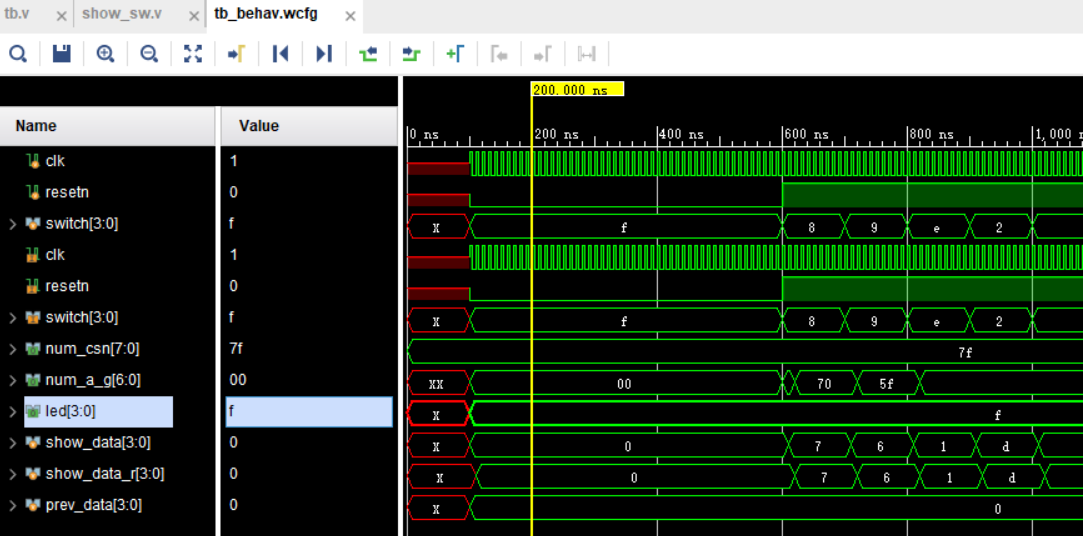
图十五 波形停止修正后

（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。

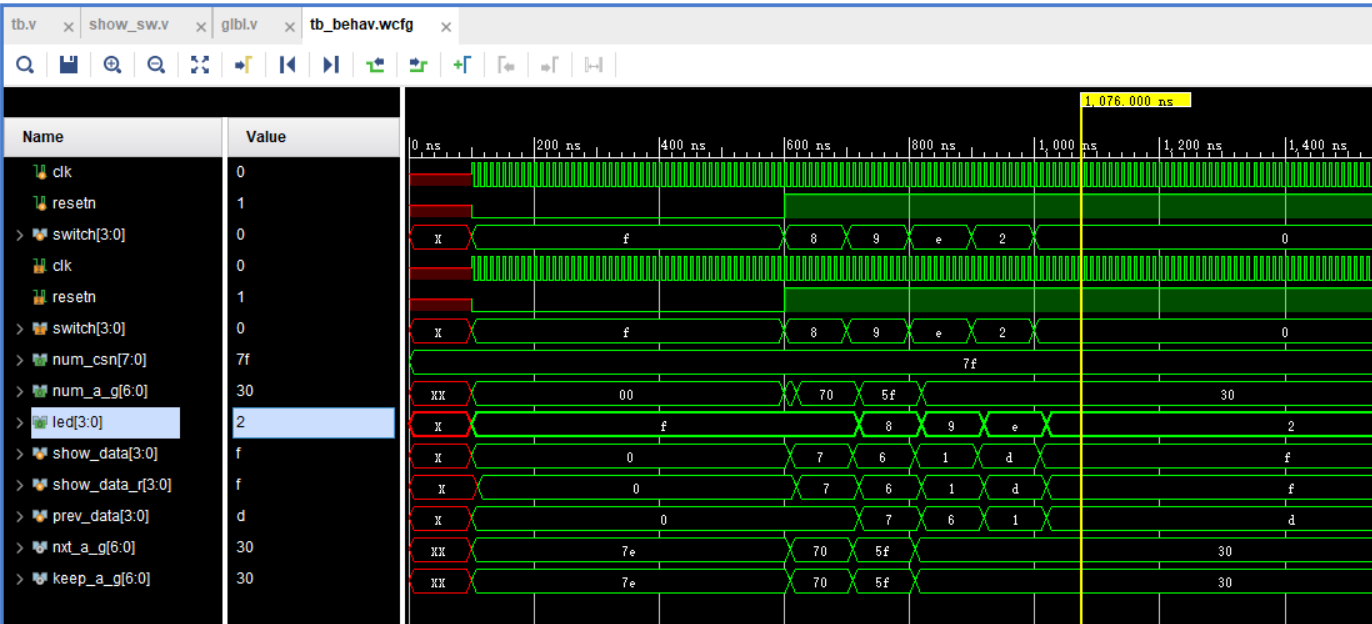
（2）分析定位过程

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