LAB2 报告

学号：2020K8009929013
姓名：史文轩
箱子号：69

一、实验任务（10%）

该实验是本学期体系结构研讨课的第一次实验，由三个小项目组成，分别是寄存器堆仿真、同步和异步 RAM 的仿真，综合与实现、以及指定数字逻辑电路的设计与调试。

由于第一个实验的目的更多是让我们熟悉 Vivado 的实验环境、复习基础的 Verilog 语法以及稍微锻炼通过仿真波形 DEBUG 的能力，且前几个小项目都很基础，因而实验的代码都是直接给出的。

实践任务 2 和 3 给出了现成的正确代码，目的是在熟悉 Vivado 的调试过程之后，仔细观察波形了解寄存器堆和同步、异步 RAM 的读写时序特征。而实践任务 4 则是给出了有数个 BUG 的代码，每个 BUG 都是讲义中提及的典型错误，我们要通过观察波形反推找出所有错误，并完成上板实验，使得板上表现出的特性符合预期。通过对这些经典 DEBUG 的过程，可以快速熟悉以后 Verilog 编程 DEBUG 的方法。

二、实验设计（0%）

三、实验过程（90%）

（一）实验流水账

周二晚上根据讲义教程在官网下载了 Vivado 2019.2 版本的压缩包，并下载安装到了本地。

周三课余时间略读了提供的讲义前三章以及附录 C。

周四又读了一遍第三章和附录 C，D 强化一下记忆。

周五做了实践任务 2,3,4，由于实验箱还没有轮换到我这里，故未能上板。

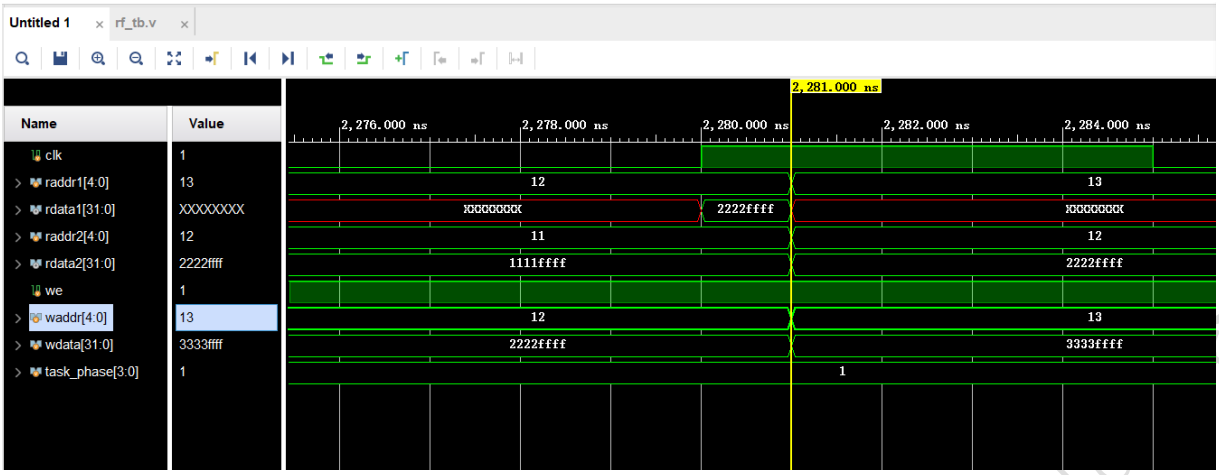
周六上板成功，写实验报告。

（二）实践任务 2：寄存器堆仿真

按照讲义上的步骤创建项目并添加了提供的文件并进行仿真后，得到了仿真波形。

通过生成的波形图可以发现寄存器堆的读写特点：在写操作上，寄存器堆在时钟上升沿同步更新；在读操作上，寄存器堆异步随时读出新的数据，即具有同步写、异步读的特点。

我们以 task_phase 信号为 1 时的一段波形为例：



由 raddr1 的波形在读十六进制下的第 12 号和第 13 号寄存器的前大半段时显示为 X 可知，没有进行过写操作的寄存器是读不出有效数据的。但是在第 2280ns，随着一个新的时钟上升沿到来，同步的写操作开始执行，第 12 号寄存器中被写入 2222ffff 这一数据。当寄存器中存在数据后，由于读操作是异步的随时更新有效数据，因而此时正在读第 12 号寄存器的第一个读接口波形数据就从 X 更新为有效数据 2222ffff，这与同步读异步写的寄存器堆读写特点符合得很好。（ps：该寄存器堆为两读一写，两个读接口的功能和特点是完全等价的）

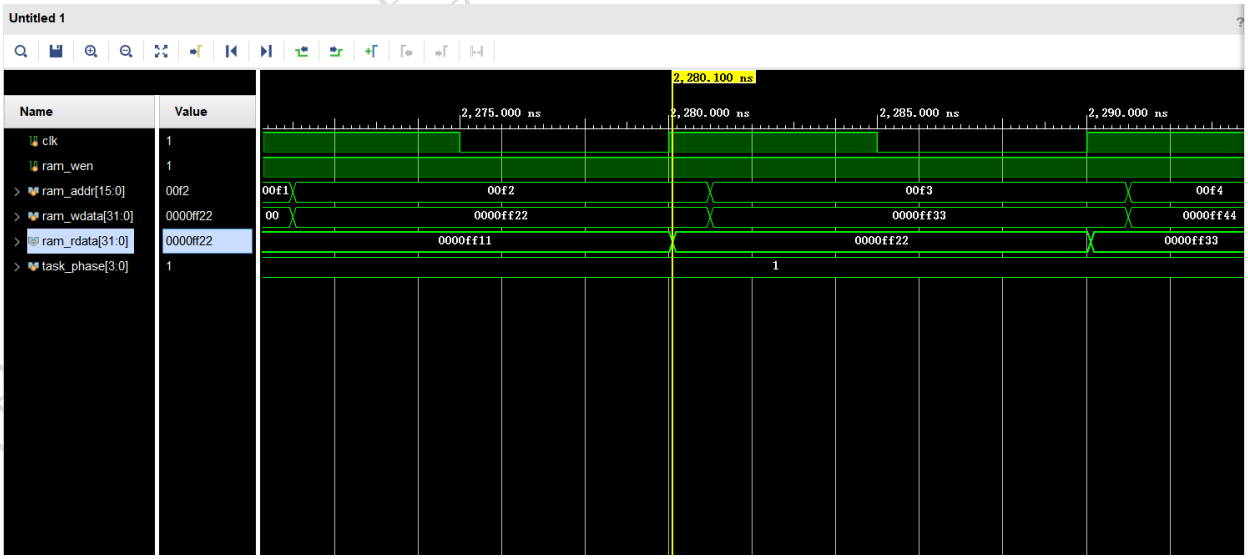
（三）实践任务 3：同步 RAM 和异步 RAM 仿真、综合与实现

1、仿真行为对比分析

（1）同步 RAM

通过观察波形，可以发现同步 RAM 的写、读操作在时钟信号的上升沿同步更新。

我们以 task_phase 信号为 1 时的一段波形为例：



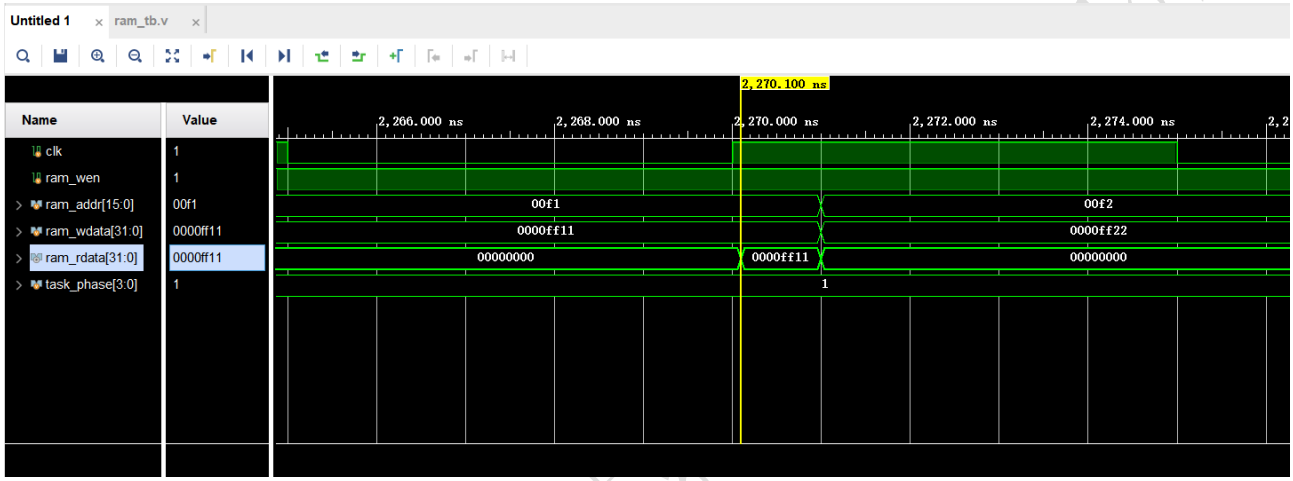
在黄线标记处左侧 0.1ns 位置的时钟上升沿，由于 ram_wen 信号为 1，因此可以向 RAM 中地址为 00f2 的位置写入有效数据 0000ff22。由于同步 RAM 的读操作也是在时钟上升沿更新读出数据，此时在 00f2 地址处读出的数据也随之更新为 0000ff22。后面读地址虽然改变，但由于没到达时钟上升沿，同步读出的数据不会发生变化。值得注

意的是，从第一个时钟上升沿到读出数据发生变化，二者间存在 0.1ns 的延迟。有关这个问题，我在 piazza 平台上的问答中找到了答案：延迟是由于 ram_ip 核中的仿真模型里有一个#1 的逻辑，这样设计是为了模拟现实中读写的延迟。

（2）异步 RAM

通过观察波形，可以发现异步 RAM 的写操作在时钟信号的上升沿同步更新，而读操作是异步地随时更新读出数据。

我们以 task_phase 信号为 1 时的一段波形为例：



在黄线标记处左侧 0.1ns 位置的时钟上升沿，由于 ram_wen 信号为 1，因此可以向 RAM 中地址为 00f1 的位置写入有效数据 0000ff11。由于读操作是异步地随时更新读出数据，故这里应该随之更新为 0000ff11。由于（1）中提及的 ram_ip 核的问题，出现了 0.1ns 的延迟。随后地址变为 0000ff22，由于这里的读取操作是异步的，因而读出数据随着地址的改变立刻改变为 0。

2、时序、资源占用对比分析

（1）同步 RAM

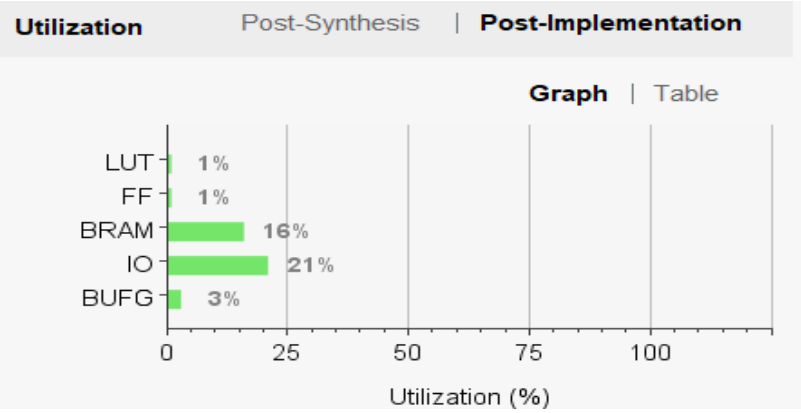
综合实现后，通过查看 Vivado 给出的报告可以得出：同步 RAM 的时序性很好，资源占用率较低。

Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.0	0	0	8/26/22, 1:44 PM
impl_1	constrs_1	route_design Complete!	0.802	0.000	0.310	0.000	0.000	0.155	0	121	4	58.0	0	0	8/26/22, 1:45 PM
Out-of-Context Module Runs															
block_ram_synth_1	block_ram	synth_design Complete!								124	4	58.0	0	0	8/26/22, 9:22 AM

讲义中对时序性有说明：WNS 为非负时，时序性极好；WNS 小于 300ps 时，时序性较好；WNS 大于 300ps 时，时序性较差。而同步 RAM 的 WNS 非负，故时序性很好。

由上图看出，同步 RAM 的时序性很好。



上图为同步 RAM 的资源占用率，可看出各项资源占用都比较低。

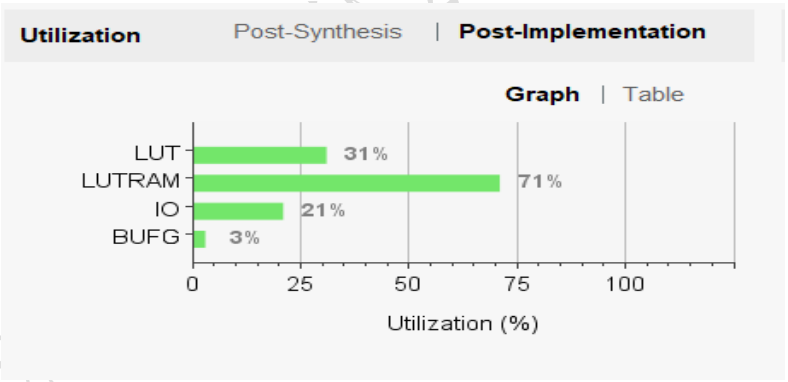
（2）异步 RAM

综合实现后，通过查看 Vivado 给出的报告可以得出：异步 RAM 的时序性一般，资源占用率较高。

Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.0
impl_1	constrs_1	route_design Complete, Failed Timing!	-7.002	-7290.607	0.188	0.000	0.000	0.383	0	41357	0	0.0
Out-of-Context Module Runs												
distributed_ram_synth_1	distributed_ram	synth_design Complete!								41357	32	0.0

异步 RAM 的 WNS 为负，但绝对值并不大，故时序性一般，比之同步 RAM 时序性差了不少。



上图为异步 RAM 的资源占用率，可看出各项资源占用都比较高，只有 BUFG 一项与同步 RAM 持平。

3、总结

同步 RAM 的写、读操作在时钟信号的上升沿同步更新；异步 RAM 的写操作在时钟信号的上升沿同步更新，而读操作是异步地随时更新读出数据。

同步 RAM 的时序性很好，资源占用率较低。而相比之下，异步 RAM 的时序性一般，资源占用率较高。

（四）实践任务 4：数字逻辑电路的设计与调试

1、错误 1：num_csn 信号为“Z”

（1）错误现象

仿真波形中 num_csn 信号始终为“Z”。

（2）分析定位过程

一般出现“Z”这个高阻信号，要么是由于 RTL 里声明为 wire 型的变量从未被赋值，要么是由于模块调用的信号未连接导致信号悬空。通过在源码中定位与 num_csn 信号相关的接口即可找到出错位置。

（3）错误原因

在调用 show_num 模块时由于将“num_csn”写作“num_scn”导致信号悬空。

（4）修正效果

将错误拼写改正过来，发现高阻信号“Z”消失。

（5）归纳总结（可选）

典型的拼写错误，但是这个拼写错误不违反语法导致比较隐蔽，但是看到高阻信号“Z”就会去看是否是 wire 变量没赋值或者是否信号悬空，较容易定位。

2、错误 2：多个信号为“X”

（1）错误现象

仿真波形中多个信号始终为“X”。

（2）分析定位过程

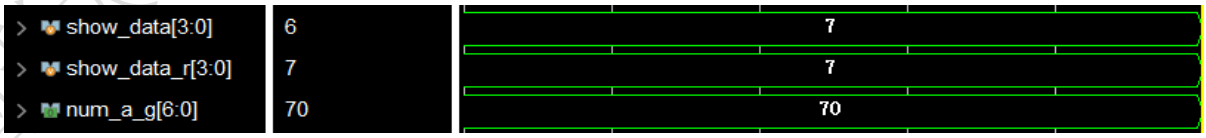
出现“X”信号一般是 reg 型变量没赋值或者是代码中存在多驱动问题。因而，我就去寻找这几个始终为“X”信号在代码中是否有 reg 型变量没赋值的现象，果然发现了信号 show_data 信号未赋值。

（3）错误原因

误把赋值语句注释掉了导致 reg 型变量 show_data 没赋值。

（4）修正效果

将错误改正过来，发现高阻信号“X”消失。



3、错误 3：波形停止在 1000ps

（1）错误现象

仿真波形在 1000ps 处异常停止。

（2）分析定位过程

波形停止问题一般是由于 RTL 代码中存在组合环路。一般来说需要进行综合靠 Vivado 的检查功能定位，但由于这次的代码量很少又比较容易理解，我在抱着找组合环路的想法仔细阅读代码的过程中很容易就发现了组合环

路出现的位置如下。

```
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'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'd7 ? 7'b1110000 :    //7
                 show_data==4'd8 ? 7'b1111111 :    //8
                 show_data==4'd9 ? 7'b1111011 :    //9
                                   keep_a_g   ;
```

nxt_a_g 和 keep_a_g 信号互相引用出现了组合循环。

（3）错误原因

根据代码要达成的逻辑，直接将 keep_a_g 的赋值语句中把+ nxt_a_g 去掉即可。

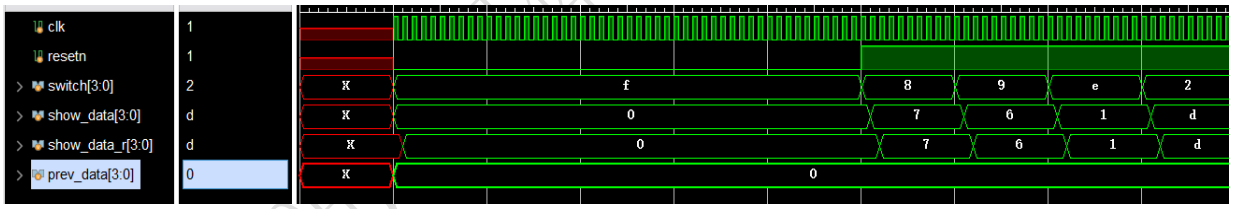
（4）修正效果

将错误改正后，发现波形停止现象消失。

4、错误 4：越沿采样

（1）错误现象

show_data_r 信号出现越沿采样问题，导致 prev_data 信号错误。



（2）分析定位过程

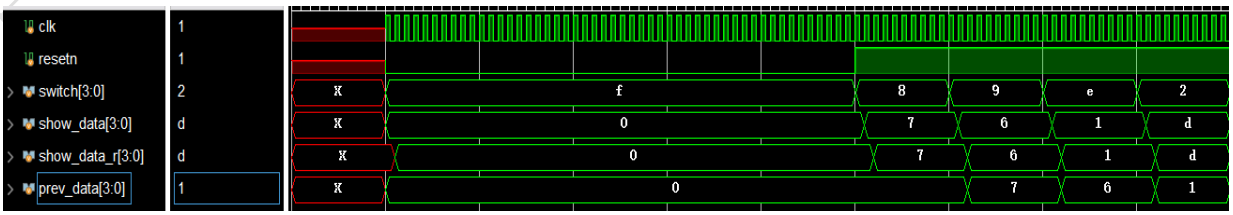
其实这个错误出现的比较明显，在代码中的 always 块里，只有 show_data_r 的赋值语句采用了阻塞赋值，而我们要实现的内容很明显用不到特意设计阻塞赋值。

（3）错误原因

把 show_data_r 的赋值语句改为非阻塞赋值语句即可。

（4）修正效果

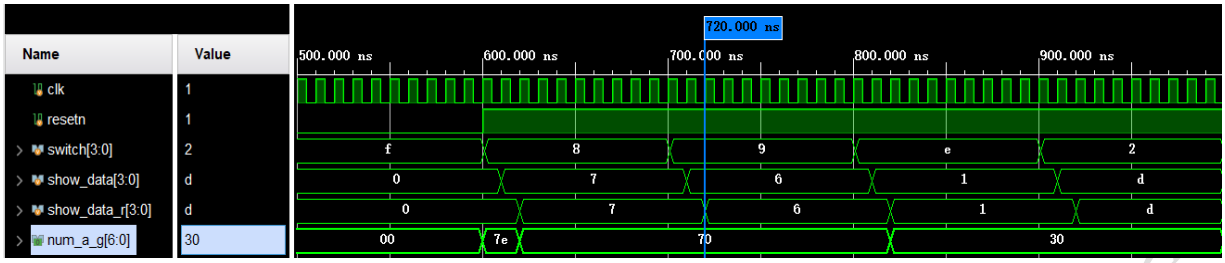
将错误改正过来，发现越沿采样现象消失，prev_data 信号恢复正常。



5、错误 5：功能未合理实现

（1）错误现象

num_a_g 信号应该更新数值的时刻未更新。（下图 730ns 处）



（2）分析定位过程

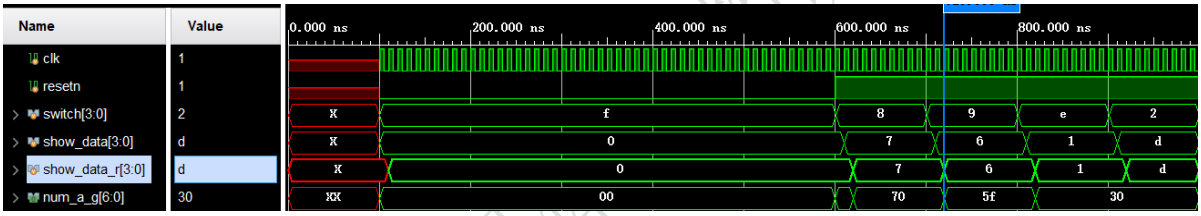
我们从逻辑上反推考虑，当 show_data 为 7 时按照数码管的灯管位置 num_a_g 信号应该为 70，但是 show_data 更新为 6 时 num_a_g 信号没有随之更新，我们只能去 RTL 代码中查看 num_a_g 在 show_data 为 6 时应该更新为什么，结果发现直接没有更新的对象，发现错误。

（3）错误原因

show_data 为 6 时 num_a_g 信号没有可更新的值。

（4）修正效果

将错误改正过来，发现 num_a_g 信号波形恢复正常。（如下图）



四、实验总结（可选）

总的来说，本次实验并不困难，主要是用于熟悉实验环境和开发流程。但是还是提供的几种 DEBUG 方法我是很受用的。之前的 DEBUG 感觉只知道这错了就去通篇乱找，好像无头苍蝇一样。这次讲义中针对几种最常见的现象给出了错误的可能性分析和操作方法，总的来看十分系统有逻辑，给我提供了一套很方便的方法！