实验 2 报告

2017K8009908018
蔡润泽
实验箱：45

一、实验任务（10%）

本次实验主要是通过让我们利用仿真和上板，熟悉硬件设计与调试的流程，并了解 Reg_File 寄存器堆, 同步 RAM、异步 RAM 的特性，其中：

实验一：利用给出的寄存器堆相关代码，通过仿真调试，得出寄存器堆的特性。

实验二：利用同步 RAM、异步 RAM 的相关代码，通过仿真、综合，查看两者的波形和资源利用率，学习同步和异步 RAM 的读写行为和两者的异同，并于寄存器堆的行为进行比较。

实验三：对于给出的代码通过仿真和上板进行调试修改，找出存在的 5 处 bug。

二、实验设计（0%）

三、实验过程（90%）

（一）实验流水账

9 月 6 日，19：00-20：00，实验一代码仿真，添加测试信号和 marker，并保存相关文件

9 月 7 日，10：00-11：30，实验二、实验三代码仿真、综合，添加测试信号和 marker，并保存相关文件

9 月 7 日，20：00-22：00，实验三代码调试、仿真、综合、上板，开始撰写实验报告

9 月 8 日， 9：30-11：30，继续撰写实验报告

（二）子任务一

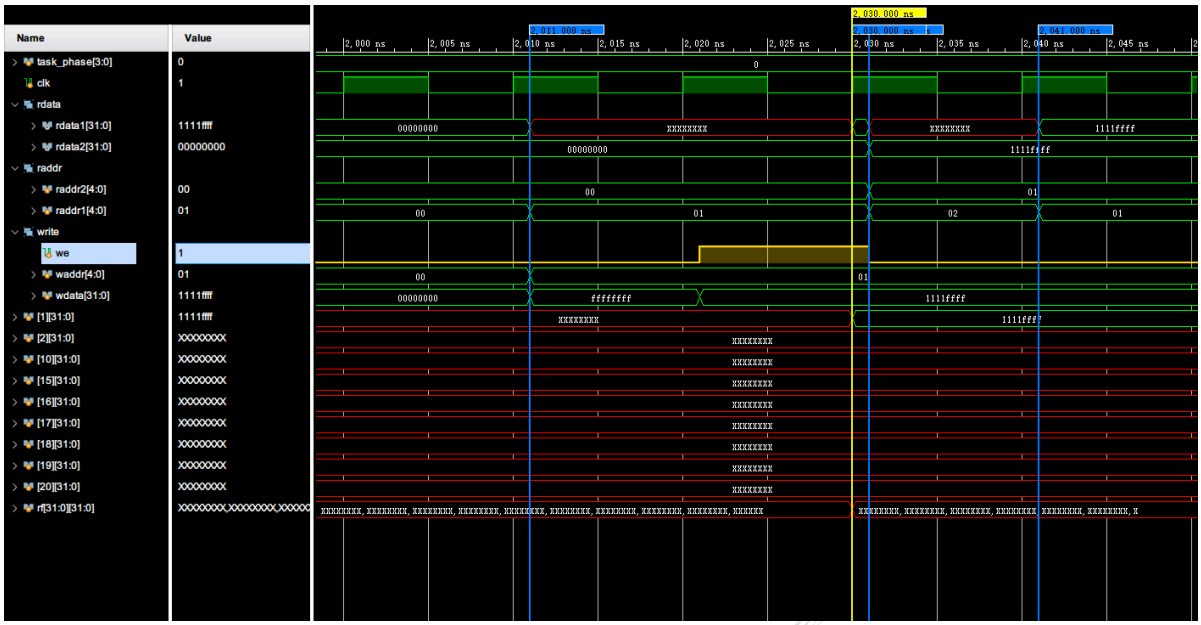


图 1.1 寄存器堆写使能为 1，时钟上升沿时的寄存器堆波形图

从此图中可以看出，第一根蓝线对应的时刻是 we 写使能信号变为 1 的时候。但此时时钟信号 clk 未处于上升沿，因此，此时 waddr 对应的 1 号寄存器的值依然没有被赋予 wdata 的值：1111ffff；而图中黄线对应的时刻，即当信号 we 变为 1，即写使能，且时钟信号处在上升沿时，wdata 的值 1111ffff 存入 waddr 指向的 1 号寄存器中。这体现了寄存器堆同步写的特点。

图中第二根蓝线是两个 raddr 读地址改变的时刻，此时的 rdata1 和 rdata2 的值也立马随之改变，分别输出 1 号寄存器的值和 2 号寄存器的值。这体现了寄存器堆异步读的特点，即不受时钟上升沿控制。

（三）子任务二

1、仿真行为对比分析

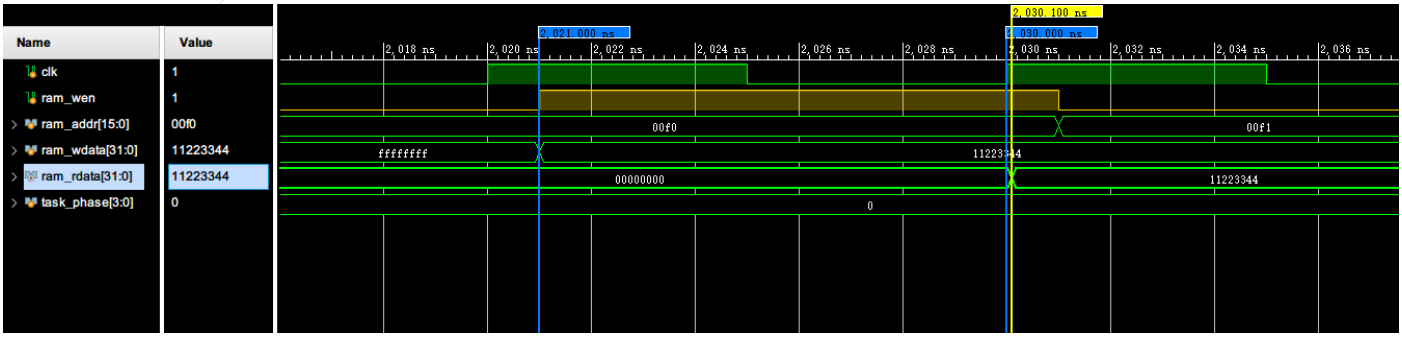


图 2.1 同步 RAM 仿真波形图一

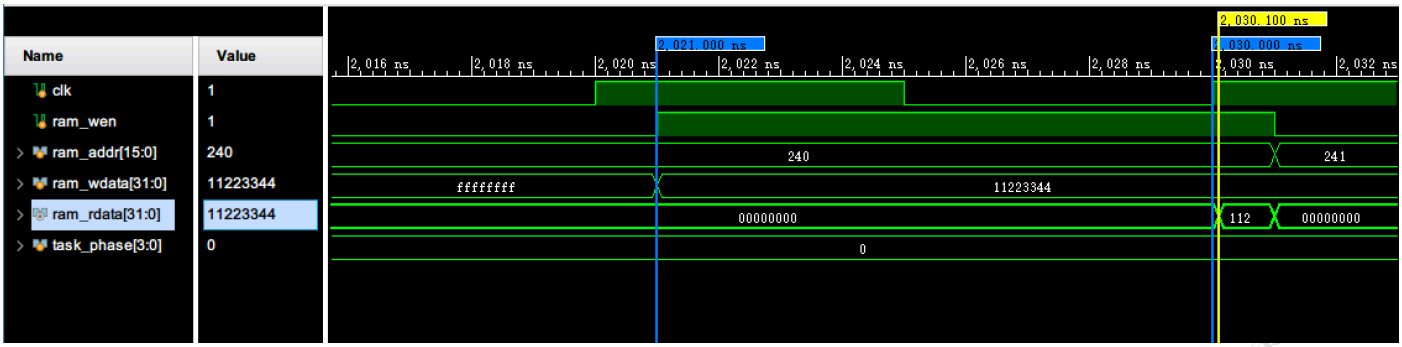


图 2.2 异步 RAM 仿真波形图一

（1） 两种 RAM 的相同点

从图 2.1 和图 2.2 中可以看出，第一根蓝线对应的时刻是 ram_wen 写使能信号变为 1 的时候，但此时 clk 并未处于上升沿，因此，此时 wdata 的值还未发生改变；而图中第二根黄线对应的时刻，即当 wen 信号为 1，即写使能，且时钟处在上升沿时，wdata 的值 11223344 存入 ram_addr 指向的内存地址中，这从黄线时指向同一地址的 ram_rdata 读数也变成 11223344 里体现出来。这体现了两个 RAM 均为同步输入的特点。

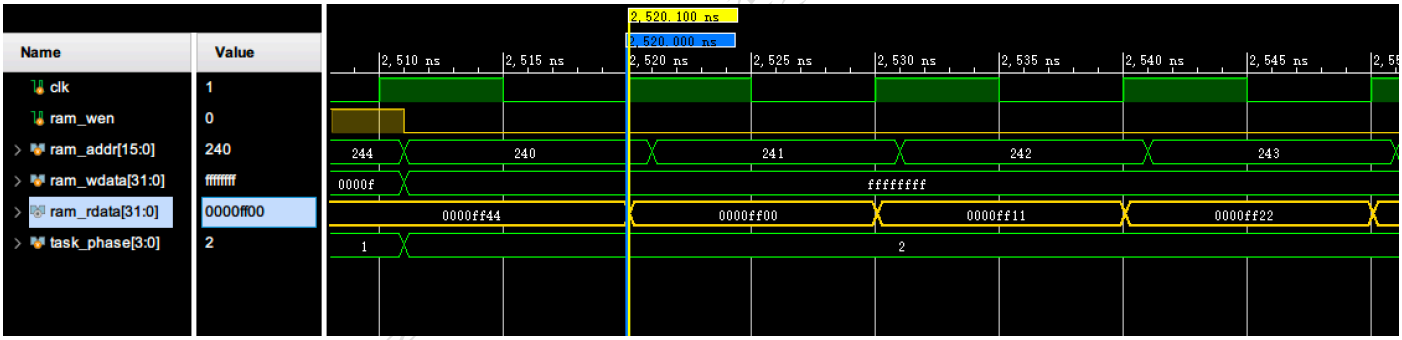


图 2.3 同步 RAM 仿真波形图二

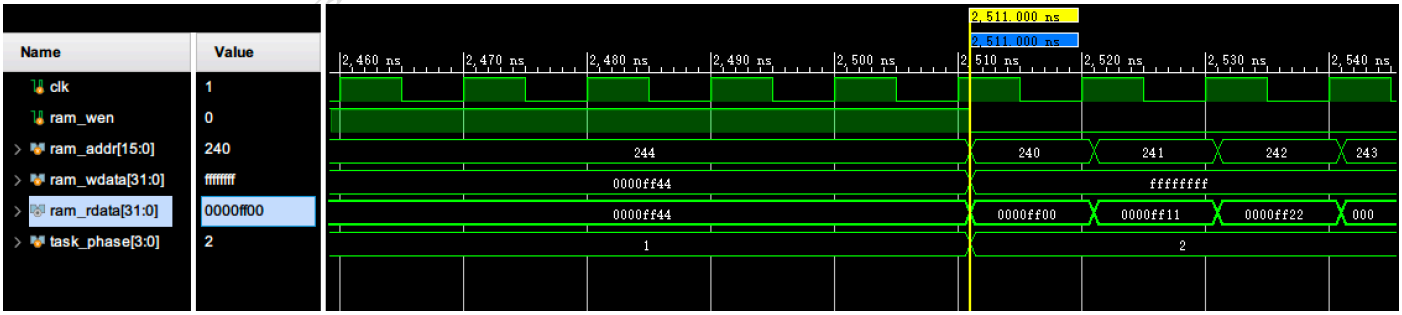


图 2.4 异步 RAM 仿真波形图二

（2） 两种 RAM 的不同点

从图 2.3 和图 2.4 中可以看出，在内存输出地址在 240 位置的值时，同步 RAM 是在时钟的上升沿才进行输出；而异步 RAM 是在读地址信号(ram_rdata)发生改变时就异步输出了。这体现出同步 RAM 同步输出，异步

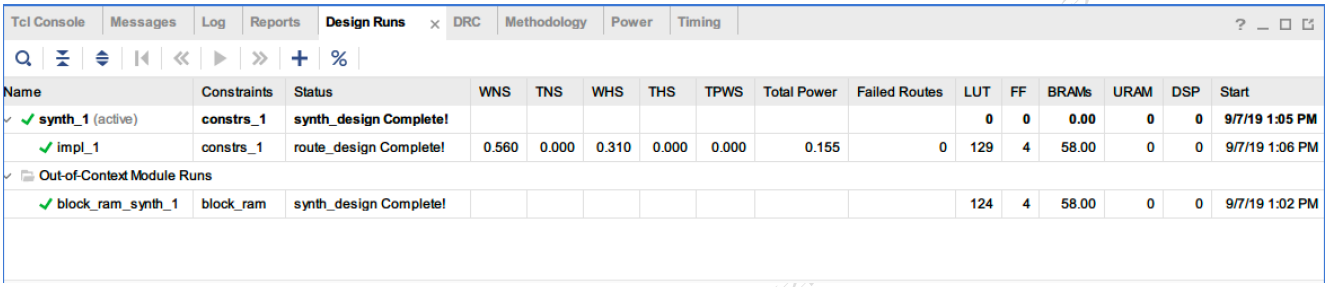
RAM 异步输出的特点。

（3） 与寄存器堆进行比较

将两种 RAM 的读写行为与寄存器堆的读写行为进行比较，可以发现，两种 RAM 的写行为都与寄存器堆相同，均为同步写。而对于读行为，只有异步 RAM 与寄存器堆相同，为异步读；而同步 RAM 为同步写。

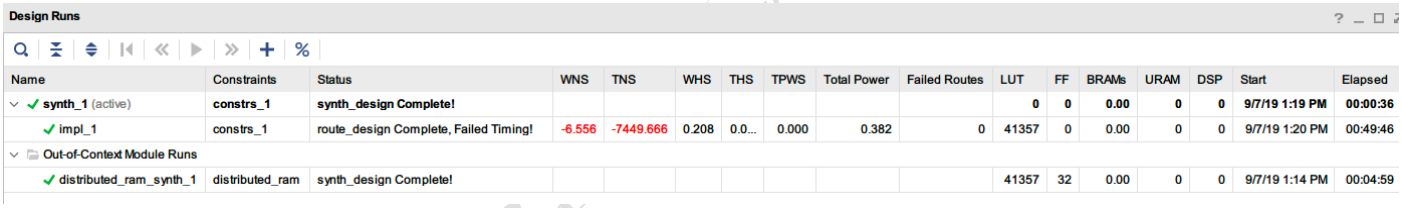
2、时序、资源占用对比分析

（1）时序对比分析



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.00	0	0	9/7/19 1:05 PM
impl_1	constrs_1	route_design Complete!	0.560	0.000	0.310	0.000	0.000	0.155	0	129	4	58.00	0	0	9/7/19 1:06 PM
Out-of-Context Module Runs															
block_ram_synth_1	block_ram	synth_design Complete!								124	4	58.00	0	0	9/7/19 1:02 PM

图 2.5 同步 RAM Design Runs



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.00	0	0	9/7/19 1:19 PM	00:00:36
impl_1	constrs_1	route_design Complete, Failed Timing!	-6.556	-7449.666	0.208	0.0...	0.000	0.382	0	41357	0	0.00	0	0	9/7/19 1:20 PM	00:49:46
Out-of-Context Module Runs																
distributed_ram_synth_1	distributed_ram	synth_design Complete!								41357	32	0.00	0	0	9/7/19 1:14 PM	00:04:59

图 2.6 异步 RAM Design Runs

通过图 2.5 和图 2.6 可以看出，同步 RAM 的最差负时序裕量 (WNS)为正 0.56，而异步 RAM 的最差负时序裕量 (WNS)为负 6.556；同步 RAM 的总负时序裕量 (TNS)为 0，而异步 RAM 的总负时序裕量 (TNS)为负 7449.666。对于时序裕量，应尽量做到 TNS=THS。

而同步 RAM 对于 LUT 的占用率和 FF 的占用率极低；异步 RAM 对于 LUT 和 LUTRAM 的占用率比较高，特别是对于 LUTRAM 的占用率达到了 71%。异步 RAM 的功耗也相对较高，是同步 RAM 的 2.5 倍。

（2）资源占用对比分析

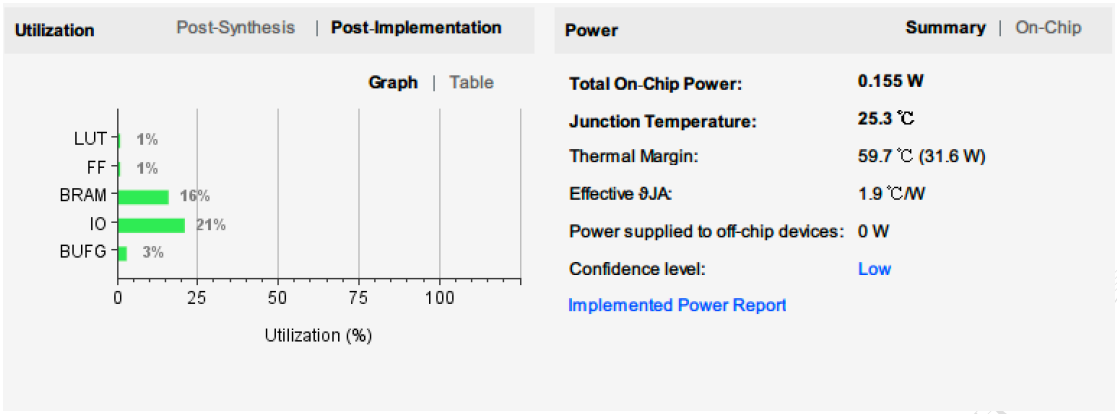


图 2.7 同步 RAM 资源利用率

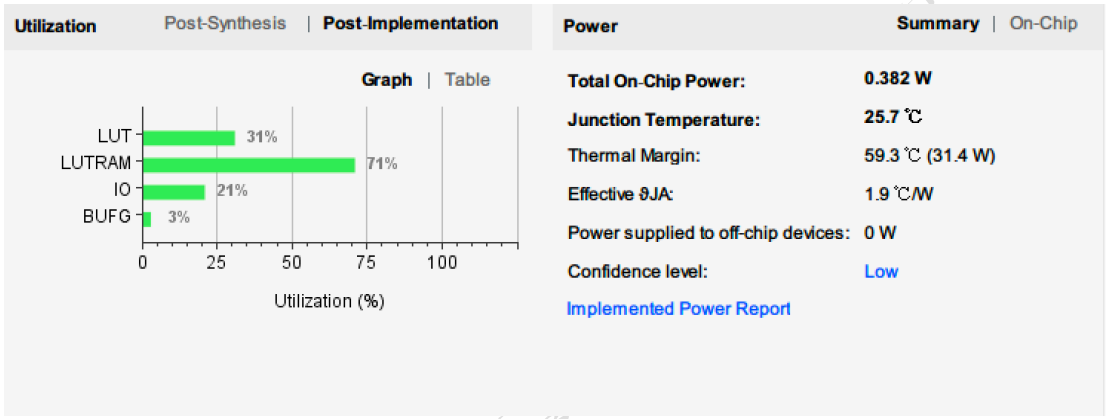


图 2.8 异步 RAM 资源利用率

通过图 2.7 和图 2.8 可以看出，两种 RAM 的 I/O 和 BUFG 占用率相当。

而同步 RAM 对于 LUT 的占用率和 FF 的占用率极低；异步 RAM 对于 LUT 和 LUTRAM 的占用率比较高，特别是对于 LUTRAM 的占用率达到了 71%。同时异步 RAM 的功耗也相对较高，是同步 RAM 功耗的 2.5 倍。

3、总结

通过上述对比分析，在生成大块 RAM 时，选择同步 RAM 来实现的做法更优。虽然异步 RAM 灵活方便，但同步 RAM 的稳定性更高，资源占用率更低，因此更适应于大块 RAM 的生成。

（三）子任务三

1、错误 1：变量命名错误导致波形为 Z

（1）错误现象

Vivado 软件有该错误变量命名行出现高亮，同时 num_csn 信号为 ZZ，如图 3.1。

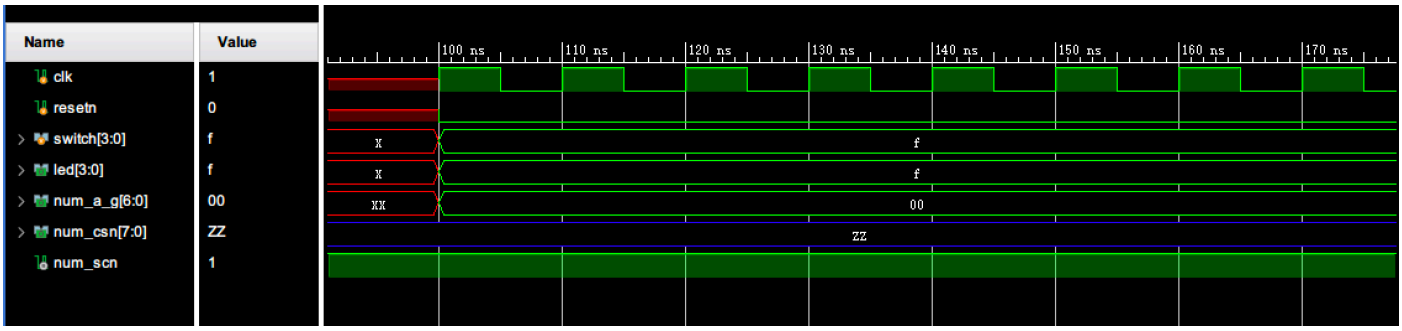


图 3.1 错误一导致的波形

（2）分析定位过程

num_csn 的波形为“ZZ”，同时 Vivado 软件对存在错误变量命名的代码行显示出高亮，通过高亮部分定位到错误变量名 num_scn，通过与前部分代码中的“num_csn”变量名比对发现，错误变量名“num_scn”应该修改为“num_csn”。

（3）错误原因

由于粗心导致的同一变量的变量名前后不一致。

（4）修正效果

修改变量名为“num_csn”高亮消失，num_csn 波形恢复正常，如图 3.2。

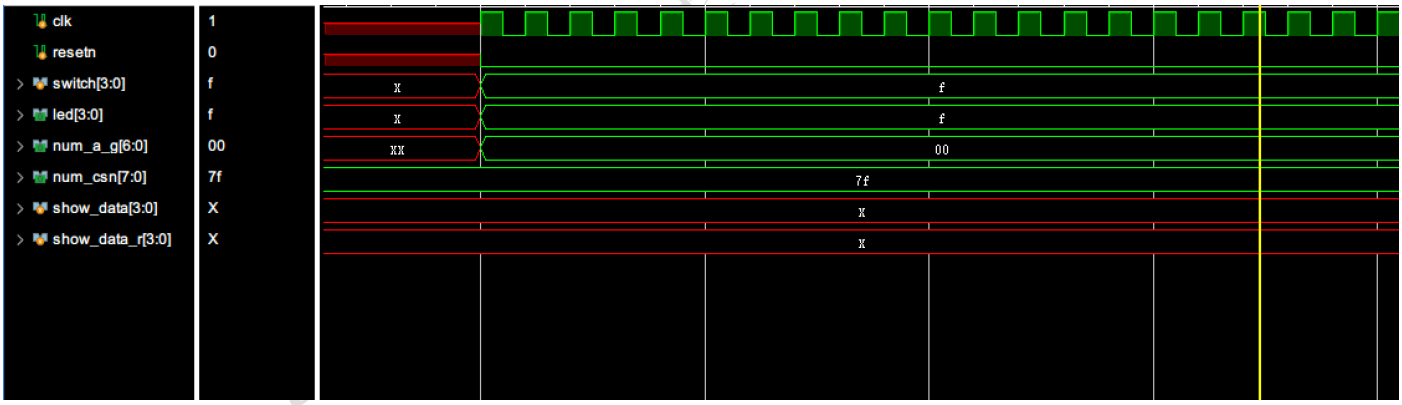


图 3.2 错误一修复后的波形、错误二出现“X”

（5）归纳总结（可选）

可以用一些 IDE 工具，通过变量补全的插件，自动补全变量。写代码时应该要时刻保持细心。

2、错误 2：赋值语句被注释导致波形为 X

（1）错误现象

如图 3.2, 波形中的 show_data 和 show_data_r 为“X”。

（2）分析定位过程

通过找 show_data 和 show_data_r 的赋值语句，发现“// show_data <= ~ switch;”，即赋值语句被注释掉。

（3）错误原因

由于失误，“//”将 show_data 的赋值语句给注释掉，导致未给 show_data 赋值。

（4）修正效果

删除注释“//”。但当修改完这一步，会发现一个新的问题，如图 3.3, 因此修正效果需要将新问题解决后才能查看。

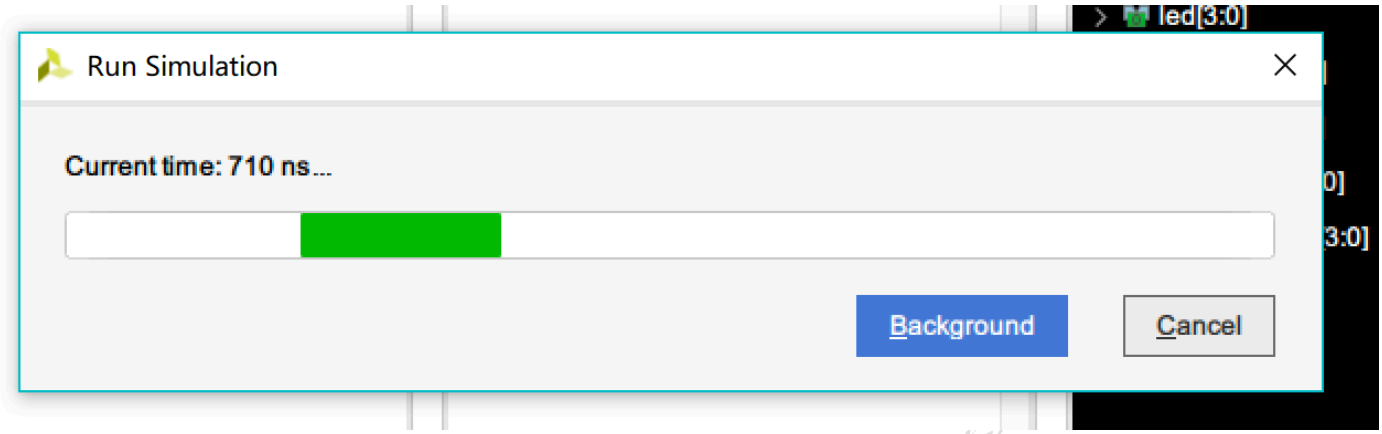


图 3.3 错误二修复后的仿真时出现新问题

（5）归纳总结（可选）

由于粗心导致的问题，编写代码时还需要更加细心。

3、错误 3：由于组合环导致波形停止

（1）错误现象

仿真卡在 710ns，不能顺利进行，如图 3.3。

（2）分析定位过程

Vivado 产生高亮，如图 3.4。

```
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   ;
endmodule
```

图 3.4 错误三仿真失败后的代码部分产生高亮

通过分析发现，“assign keep_a_g = num_a_g + nxt_a_g;”和”assign nxt_a_g = show_data==…=keep_a_g ;”中 nxt_a_g 参与了 keep_a_g 的赋值，而 keep_a_g 参与了 nxt_a_g 的赋值，因此形成了变量赋值的组合环。通过

对代码的语义分析，可以将“assign keep_a_g = num_a_g + nxt_a_g；”改为“assign keep_a_g = num_a_g；”。

（3）错误原因

nxt_a_g 参与了 keep_a_g 的赋值，而 keep_a_g 参与了 nxt_a_g，导致形成组合环，从而导致了波形停止。

（4）修正效果

如图 3.5，将“assign keep_a_g = num_a_g + nxt_a_g；”改为“assign keep_a_g = num_a_g；”后，仿真正常执行，错误 2 的修改也得到了验证。

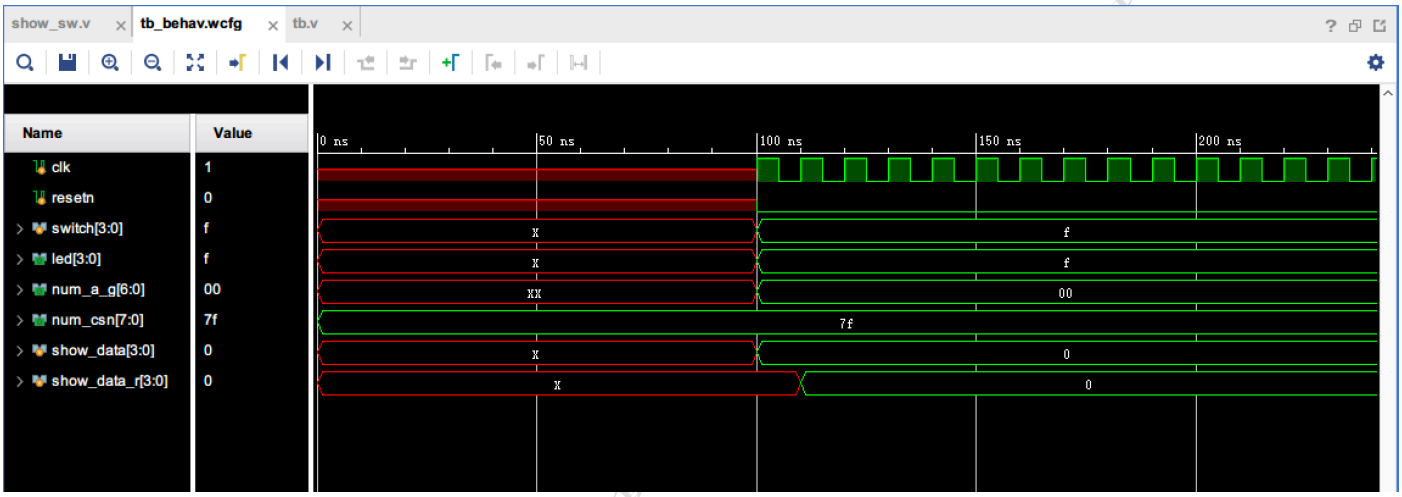


图 3.5 错误三修改后的波形图

4、错误 4：always 语句中未用非阻塞赋值形成越沿采样

（1）错误现象

通过观察代码，发现 show_data_r 的赋值语句在 always 块中，但采取的是阻塞赋值方式

（2）分析定位过程

通过找 show_data_r 的赋值语句，发现“show_data_r = show_data;”，即 show_data_r 没用使用非阻塞赋值，而在时序逻辑里，需要使用非阻塞复制，因此，我将“show_data_r = show_data;”改为“show_data_r <= show_data;”

（3）错误原因

show_data_r 的赋值语句在 always 块中，但采取的是阻塞赋值方式。而在时序逻辑里，需要使用非阻塞复制。

（4）修正效果

将“show_data_r = show_data;”改为“show_data_r <= show_data;”

5、错误 5：功能 BUG

（1）错误现象

上板后，如图 3.6，从右至左拨第 2、第 3 个开关，数字“6”无法显示。

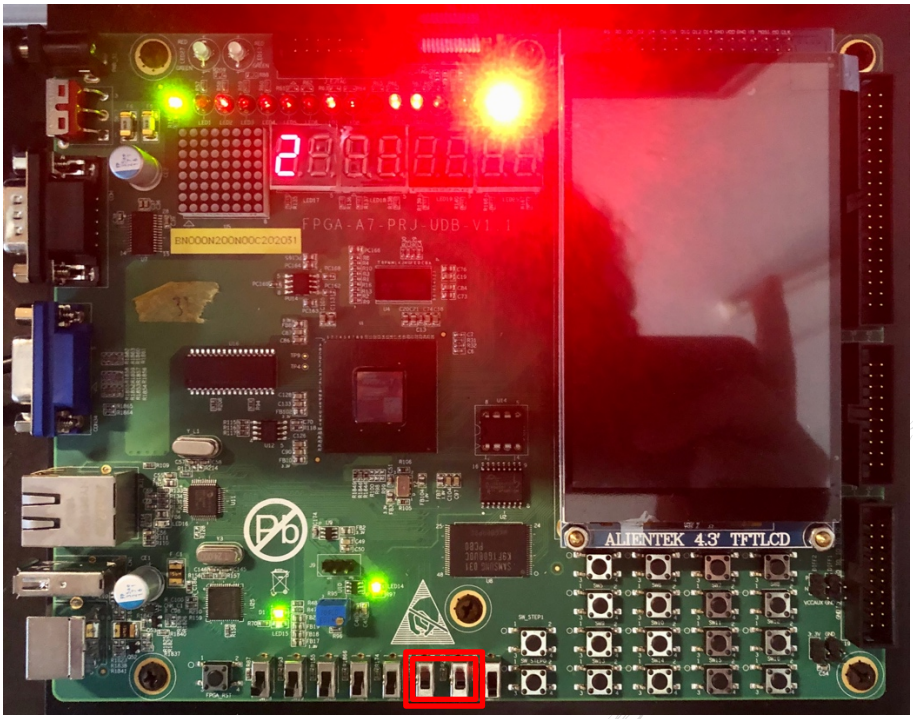


图 3.6 错误五无法显示数字“6”的上板图

（2）分析定位过程

通过找变量“nxt_a_g”的条件赋值语句，发现缺少了“show_data==4'd6？”这一个选择条件，因此，代码加上了“show_data==4'd6？7'b 1011111：”。

（3）错误原因

变量“nxt_a_g”的条件赋值语句，缺少了“show_data==4'd6？”这一个选择条件，导致板上数字“6”无法显示。

（4）修正效果

代码加上了“show_data==4'd6？7'b 1011111：”后，如图 3.7 上板数字“6”可以显示。

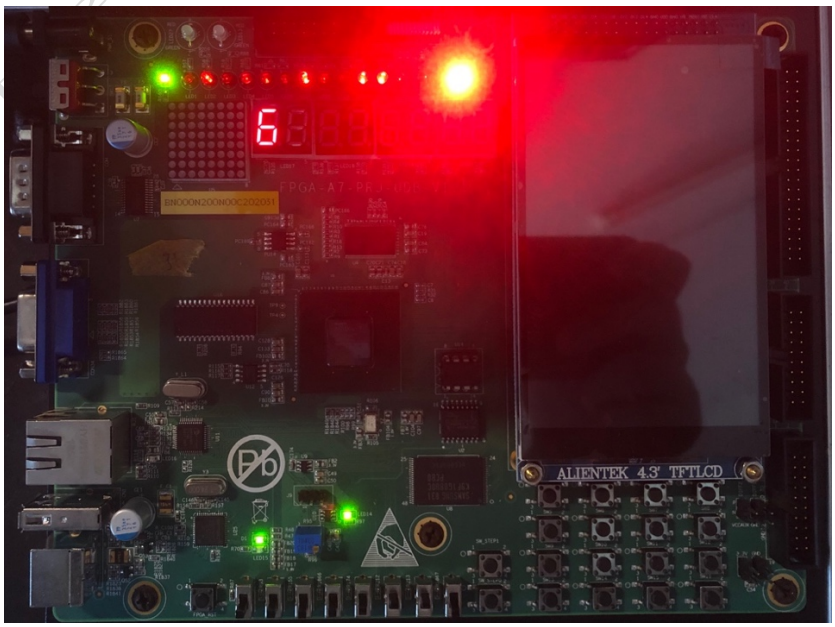


图 3.7 错误五修正后显示数字“6”的上板图

四、实验总结（可选）

通过完成这次实验，在硬件部分，我较之前增加了对同步 RAM 和异步 RAM 的了解。发现同步、异步 RAM 只是在读数据时有区别。异步 RAM 的异步只指读异步。通过两者的对比，也了解到生成大块 RAM 时，同步 RAM 更具有优势。

而子任务三对于测试代码的调试，让我了解到保持细致在编程时的重要性。做实验之前，本以为这 5 个 bug 会藏得很深，但发现这 5 处 bug 都只是非常常见的错误，甚至我第一次接触测试代码时，在还没有经过仿真和上板的测试的情况下，就直接找出了这 5 处 bug。当然，“当局者迷”，找别人的 bug 可能相对容易，但自己写代码的时候，可能一不小心也是会犯很多自己并不容易察觉的小错误，所以今后写代码的确需要更加细致。

另外，这次调试虽然没有测试自己编写的代码，但上板调试的过程能看到代码的变动能反应在机器的变化上，还是非常具有体验感。这也让我对今后的实验充满了期待。当然了，还是希望自己今后的实验能少写一些 bug，争取少调几次就能通过。