实验二报告

学号 2018K8009922027
姓名 李国峰
箱子号 1

一、实验任务（10%）

Lab2 有三个子任务，第一个任务是对一组有 32 个 32 位寄存器、两读一写端口的寄存器堆进行仿真，结合代码观察波形并理解其工作原理。

第二个任务是调用 Xilinx 的库 IP 分别实例化同步 RAM 和异步 RAM，进行仿真，观察波形对比二者的异同，然后进行综合和实现，并且利用 VIVADO 自带的工具查看时序结果和资源利用率，借此预测设计上板的成功率。

第三个任务简而言之就是 debug 任务，通过阅读代码、观察波形、观察综合情况和上板调试来定位与预期功能不符的地方，并进行相应的修改。

二、实验设计（0%）

略。

三、实验过程（90%）

（一）实验流水账

2020 年 9 月 16 日 20:00-23:00 阅读讲义，基本掌握提到的 design 和 debug 技巧；

2020 年 9 月 17 日 15:00-17:00/ 18:00-21:00 阅读任务要求，完成子任务 1、2 的仿真和实现，阅读 debug 任务要求和源代码，没有读懂；

2020 年 9 月 18 日 10:30-11:30，基本读懂任务要求和代码的对应关系

2020 年 9 月 18 日 12:00-15:00，完成 debug，并完成仿真和最后的上板测试，测试通过。

（二）子任务一

新建工程后导入代码源文件和仿真激励文件，进行仿真，结合代码（图 1）观察并理解波形行为。

```
// WRITE
always @(posedge clk) begin
    if (we) rf[waddr]<= wdata;
end
// READ OUT 1
assign rdata1 = (raddr1==5'b0) ? 32'b0 : rf[raddr1];
// READ OUT 2
assign rdata2 = (raddr2==5'b0) ? 32'b0 : rf[raddr2];
endmodule
```

图 1

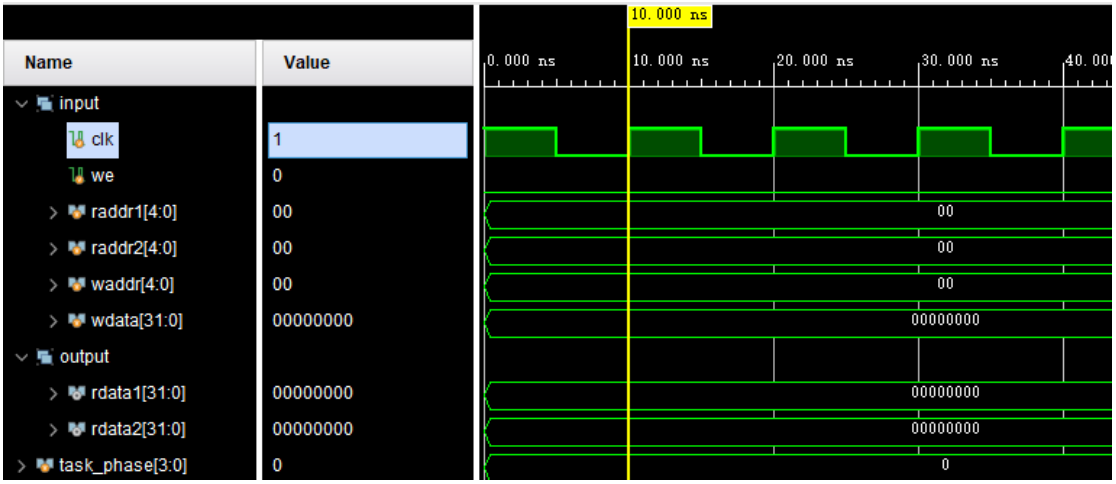

图 2

根据代码，我们可以得到以下信息：当且仅当时钟上升沿（posedge clk）且写使能信号（we）位于高电平时才能将数据写入指定地址的寄存器中。在图 2 所示时间段内，写使能信号都位于低电平，因此无论地址和数据如何变化，寄存器都不会被写入数据。

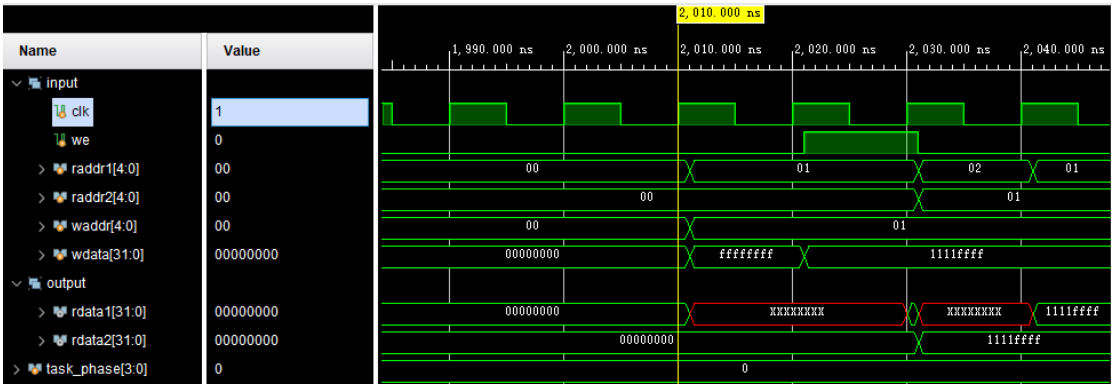

图 3

观察图 3，当时钟上升沿到来且写使能信号有效时（2030ns），写地址（waddr）和写数据（wdata）同时生效，数据 1111ffff 被写入 01 寄存器中，此后在新的数据写入之前，01 寄

存器读出的数据会一直保持 1111ffff。换句话说，数据的写入只有在时钟上升沿到来且写使能信号有效时才能进行，而数据的读取是一直在进行的，读出的数据只与读取地址有关，与时钟信号无关。

值得一提的是，在图 3 中出现的 X 波形是因为寄存器中事先没有存入数据产生的，并非设计缺陷。

（三）子任务二

1、仿真行为对比分析

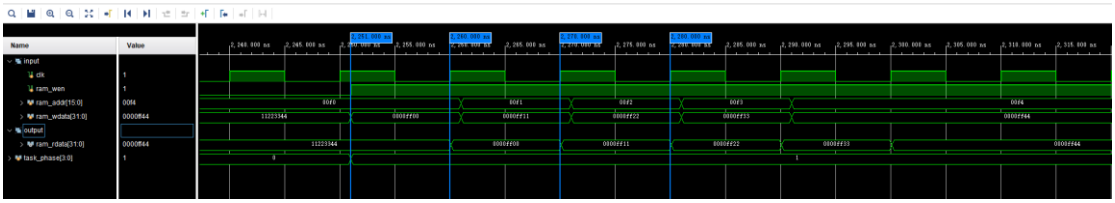

图 4

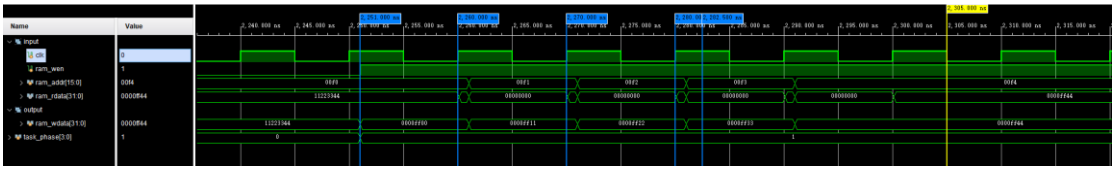

图 5

如图 4 和图 5 分别为同步 RAM 和异步 RAM 在同一时间段内的仿真波形。

对比图 4 和图 5 可知，在 ram_wen 信号有效时，同步 RAM 在某一时钟上升沿将数据写入，只有在下一个时钟上升沿到来时才可以读出数据；而异步 RAM 可以实时读取写入的数据。

2、时序、资源占用对比分析

由下图 6 和图 7 可知，同步 RAM 时序满足极好，异步 RAM 时序违约且十分糟糕，意味着异步 RAM 设计上板失败的可能性很大。

考虑资源利用率，同步 RAM 耗费主要资源远远少于异步 RAM，BRAM 占用率也远远小于 LUTRAM，同步 RAM 和异步 RAM 在 IO 和 BUFG 上占用资源相同，这意味着异步 RAM 电路的综合实现比同步 RAM 更加复杂，花费时间更多，可以大致判断异步 RAM 的电路结构比同步 RAM 复杂得多。

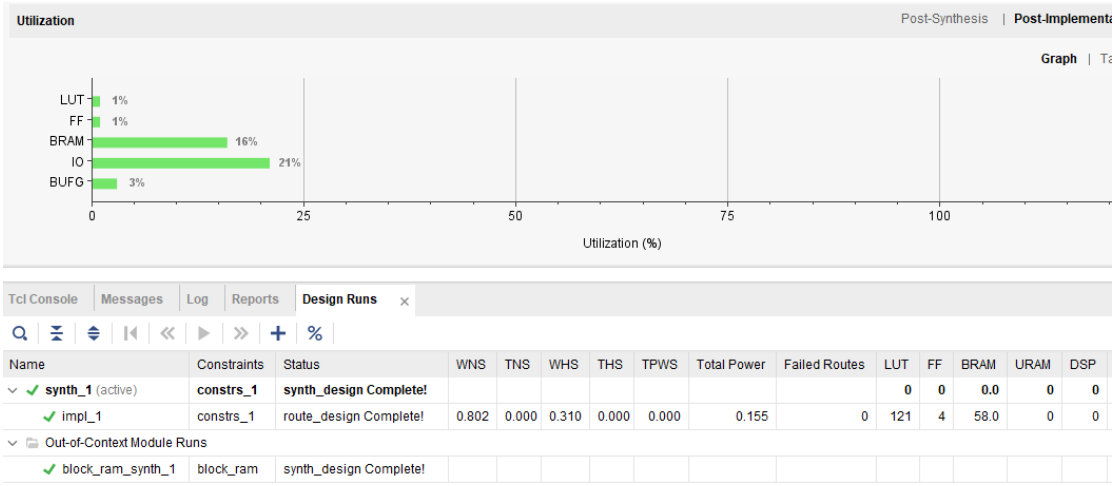

图 6

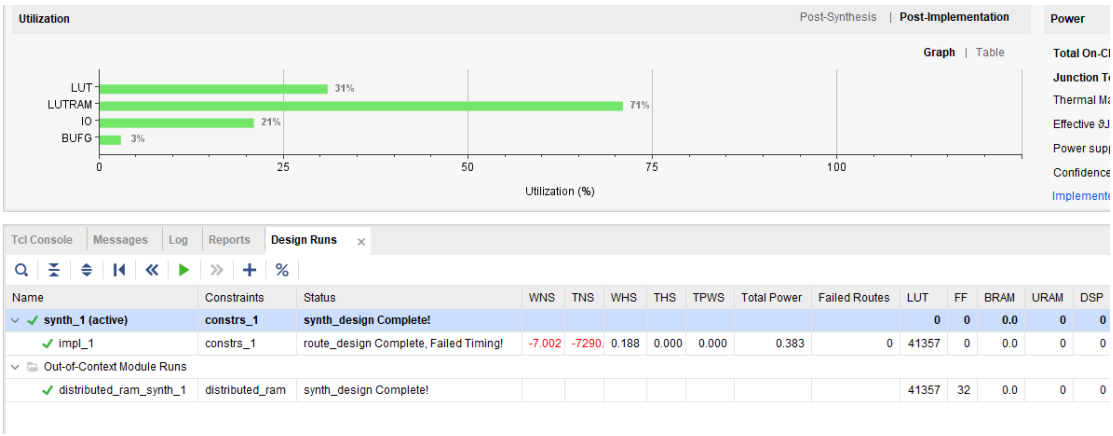

图 7

3、总结

（四）子任务三

1、错误 1：波形为 Z

（1）错误现象

在波形中存在蓝色的波线，读出来的值为 Z，如图 8。

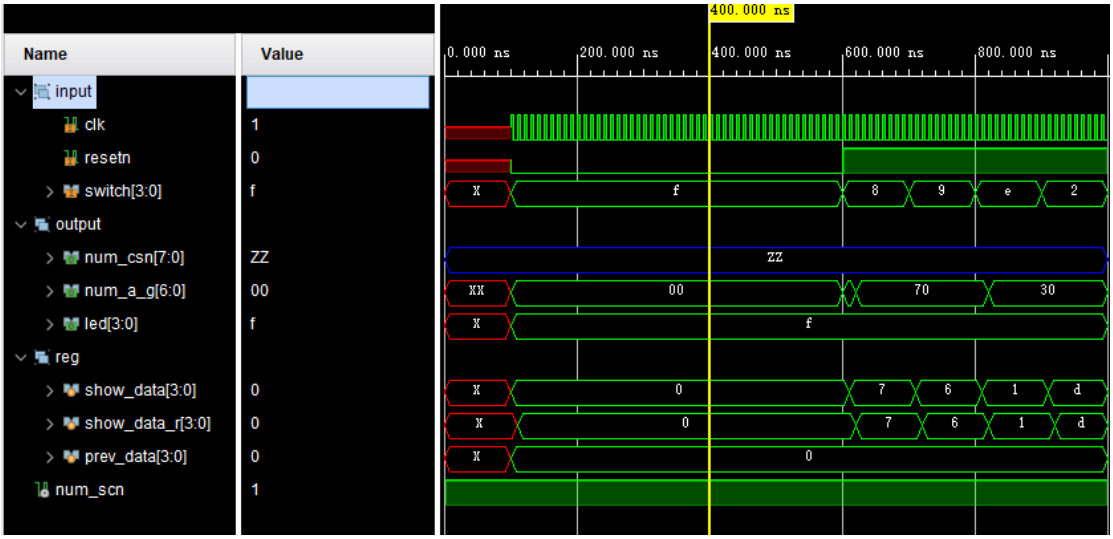

图 8

（2）分析定位过程

观察波形图，找到与出现的 Z 波形有关系的所有信号即 num_csn，回到代码中寻找这些波形之间的关系，再结合讲义中讲述的 Z 波形出现的原因可能是存在端口没有连接，有针对性地进行寻找。同时在波形图中发现突兀的 num_scn 信号。

（3）错误原因

调用模块时端口名输入错误（笔误）导致连接故障，正确的端口名 num_csn 被写成 num_scn。如图 9。

```
//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  )
);

endmodule
```

图 9

（4）修正效果

更正端口名即可，之后 Z 波形消失。

（5）归纳总结（可选）

输入笔误，可能是由于输入过快导致的字母顺序错误，更仔细一些即可，无需刻意关注。

2、错误 2：波形为 X

（1）错误现象

在波形中存在红色的波线，读出来的值为 X。如图 10 所示，show_data 和 show_data_r 的波形始终为 X。

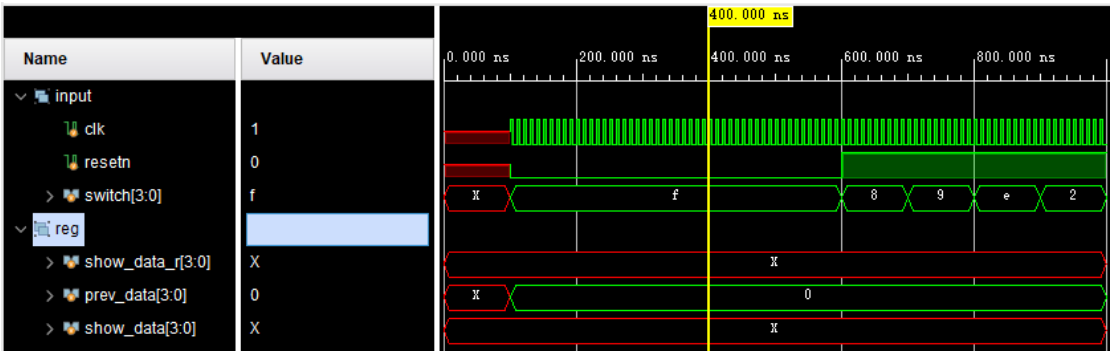

图 10

（2）分析定位过程

观察波形图，找到与出现的 X 波形有关系的所有信号，回到代码中寻找这些波形之间的关系，再结合讲义中讲述的 X 波形出现的原因可能是存在 reg 型变量没有被赋值或形成了多驱动，然后有针对性地进行寻找。如图 11 源代码片段所示，show_data_r 在时钟上升沿被赋给 show_data 的值，而 show_data 被赋给输入信号 switch 取反的值，但这一语句被注释掉了，所以 reg 型信号 show_data 没有被赋值。

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

always @(posedge clk)
begin
    show_data_r = show_data;
end
```

图 11

（3）错误原因

Reg 型变量 show_data 没有被赋值。

（4）修正效果

去掉//，让赋值生效，X 波形消失。

（5）归纳总结（可选）

Reg 型变量必须被赋值，在非组合逻辑中（always 语句中），能被赋值的变量一定是 reg 型。

3、错误 3：波形停止

（1）错误现象

进行仿真时出现了波形停止。

（2）分析定位过程

出现波形停止，认为是出现了组合环路。由于变量嵌套较少，很快找到了互相调用的变量。

（3）错误原因

```
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   ;
```

图 12

如图 12 所示，存在变量的互相调用，nxt_a_g 被赋予 keep_a_g 的值，keep_a_g 同时也被 nxt_a_g 的值影响，形成了组合环路。

（4）修正效果

根据功能需要删除多余的信号调用，只需将 keep_a_g 的赋值语句改为 keep_a_g=num_a_g;即可。组合环消失，仿真正常进行。

（5）归纳总结（可选）

深度较低的组合环可以根据观察得到，深度较深时可以先进行综合实现，根据 log 来判断出问题的信号。

4、错误 4：越延采样

（1）错误现象

波形中信号并没有随信号正确变化。如图 13 所示，pre_data 每一位的波形一直为 0，并没有如预期一样显示前一状态的波形。根据代码，只要在时钟上升沿来临，resetn 信号为高电平且 show_data 与 show_data_r 不相同时，prev_data 信号必然要获得 show_data_r 的值，但事实并非如此。

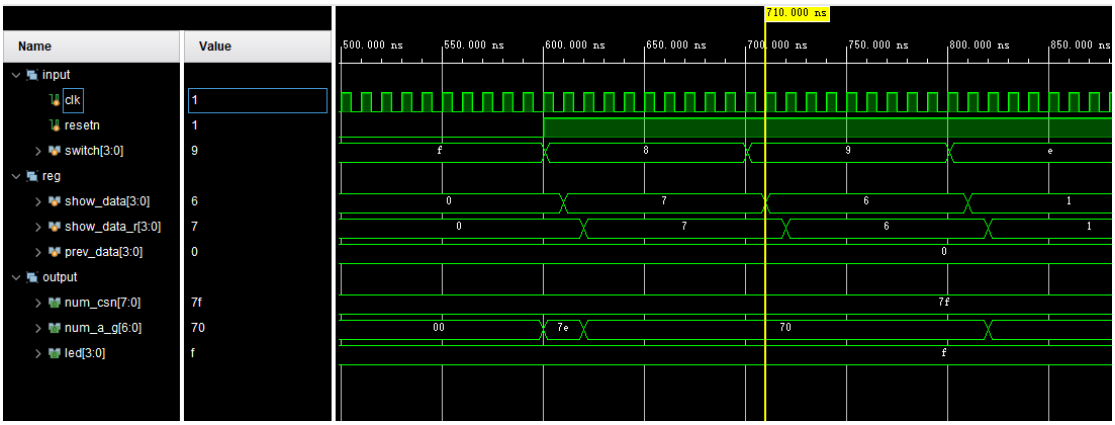

图 13

（2）分析定位过程

阅读代码的过程中发现在两个 always 模块中分别出现了阻塞赋值和非阻塞赋值，相关的信号恰好是出现波形异常的信号。

（3）错误原因

不合理使用阻塞赋值导致越延采样。

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

always @(posedge clk)
begin
    show_data_r = show_data;
end
```

图 14

（4）修正效果

使用非阻塞赋值，信号变化符合期望。

（5）归纳总结（可选）

需要深入理解阻塞和非阻塞赋值。阻塞赋值的过程是立即执行的，即阻塞赋值运算符右侧表达式求值完后会立刻更新至运算符左侧，且该执行过程不受其他语句执行的影响。非阻塞赋值在执行中，在当前仿真时间槽（time-slot）开始分析计算获得右侧表达式的值，在当前时间槽执行结束时更新左侧表达式的值，在右侧表达式分析计算和左侧表达式被更新之间，任何其他事件都可以执行，同时也有可能修改以及计算完成的右侧表达式的值，即左侧表达式获得的值不一定是最新的，非阻塞赋值的过程不影响其他语句的执行。

5、错误 5：功能 bug

（1）错误现象

上板时最左侧数码管无法显示数字 6。

（2）分析定位过程

阅读代码，发现缺失实现数字 6 的内容，如图 6 所示。

（3）错误原因

代码不完整导致功能缺失。

（4）修正效果

加上数字 6 的驱动代码，然后可以正常显示数字 6。值得一提的是，数码管的 7 段信号与 num_a_g 每一位的对应关系并不是显而易见的，需要根据其他的数字的 num_a_g 信号来推出该对应关系。

（5）归纳总结（可选）

略。

四、实验总结（可选）

实验手册的内容讲解十分详尽，特别适合我这样数字系统设计基础不牢固、经验不足的学生阅读和参考，我在阅读和上手敲代码的过程中学到了很多实用的设计、调试技巧。

做完试验后，回顾做实验的整个过程，觉得不是很难，很多时间花在对 debug 实验题目的理解上，不是很懂代码的各个端口和开发板的对应关系，因此一开始觉得无从下手。这种感觉十分玄学，可能和思维的状态好坏有关。

异步 RAM 的综合实现花了很长时间，一开始还以为 vivado 又卡住了。同步 RAM 和异步 RAM 的波形比较费了一番功夫，因为仿真的时间尺度较小，为了能够充分观察局部的细节进而方便比较，牺牲了同时对整体变化情况的把握。此外，将信息完整且清晰可见的波形图放到实验报告中也造成了一些麻烦。

希望以后的每一个项目都能按时高质量完成。