实验 1 报告

学号 2017K8009922027
姓名 张磊
箱子号 39

一、实验任务（10%）

任务一：通过对寄存器写入和读取内容的分析，理解波形的意义，学会利用波形分析仿真情况；

任务二：通过对同步 RAM 和异步 RAM 仿真波形的分析，理解同步 RAM，异步 RAM，寄存器堆的读写区别，并通过对同步 RAM 和异步 RAM 的时序分析，资源占用分析，进一步了解同步 RAM 和异步 RAM 的区别；

任务三：通过对设计代码的仿真调试和上板调试，熟悉 vivado 的操作，掌握 debugg 的能力；

二、实验设计（0%）

三、实验过程（90%）

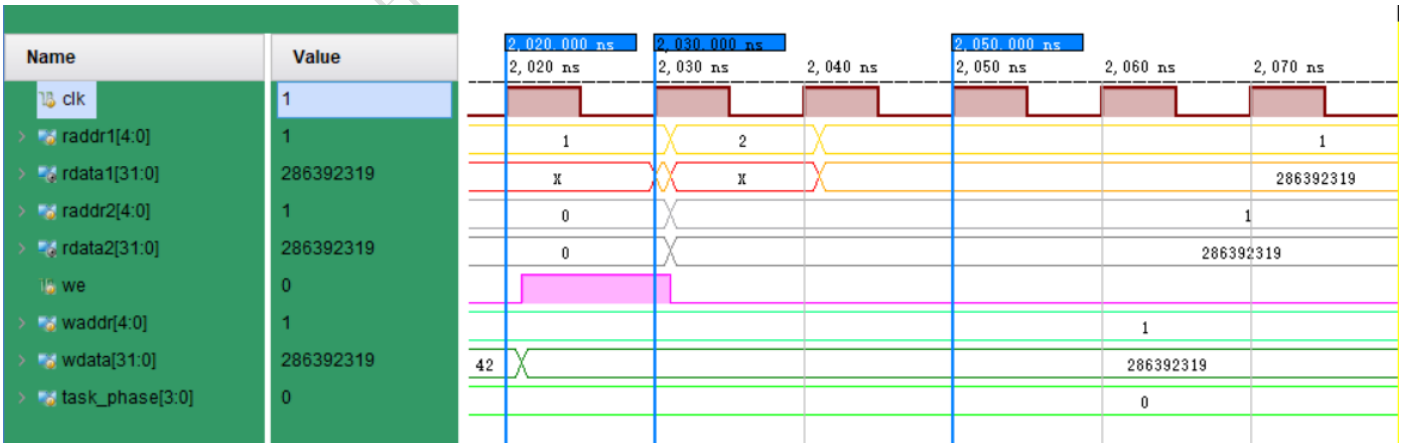
（一）实验流水账

任务一：2019 年 9 月 8 日，下午 14 点 30 到 15 点 40，完成了 regfile_task 任务；

任务二：2019 年 9 月 8 日，下午 16 点到 19 点，完成 ram_task 任务；

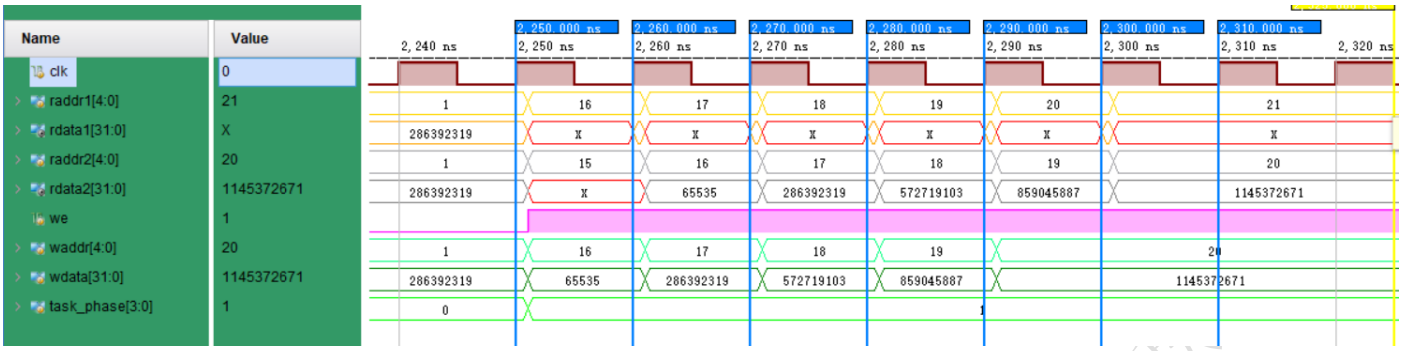
任务三：2019 年 9 月 8 日，晚上 19 点到晚上 21 点，完成 show_switch 任务；

（二）子任务一



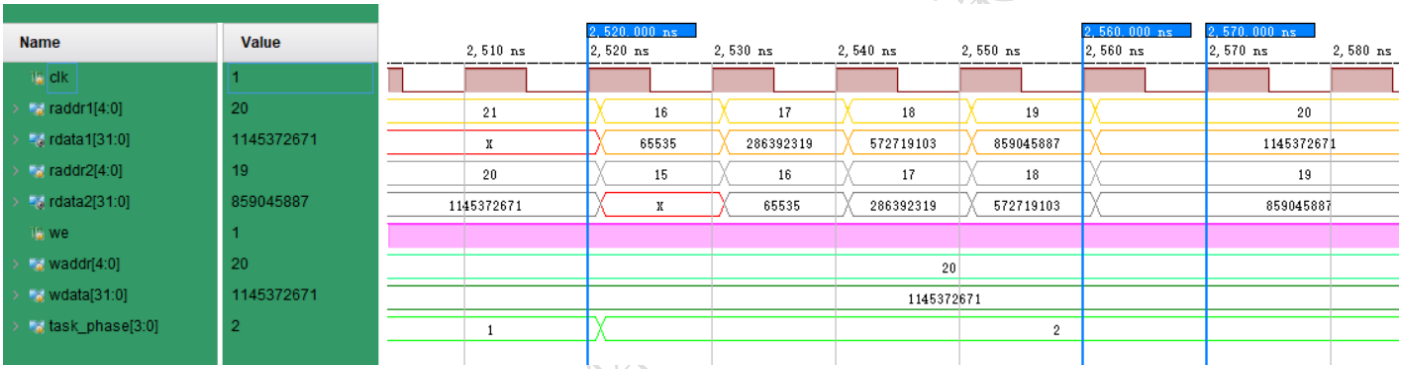
1. 第一个 Marker 时，时钟上升沿，寄存器读取 1 号寄存器，但由于之前没有写入，且没有复位为 0，所以为 X，此时虽然 wdata 有数据，但由于写使能 we 为 0 所以写入无效；
2. 第二个 Marker 时，时钟上升沿，此时写使能有效，因此，寄存器在 1 号寄存器内写入 wdata；
3. 第三个 Marker 时，时钟上升沿，此时写使能无效，可以看到，从 1 号寄存器中读出了刚才写入的数据，

证明写入成功；



4. 由于从上图的第 3 个 Marker 到此图的第 1 个 Marker 写使能一直为 0，所以 1 号寄存器内容并未改变，从此图的第 2 个 Marker 开始，写使能一直为 1，所以，开始陆续向 16-20 号寄存器中写入相应的数据；

5. 从第 2 个 Marker 开始，读数据为 X 的原因同 1 号寄存器，由于之前没有写入数据，也没有进行复位操作；

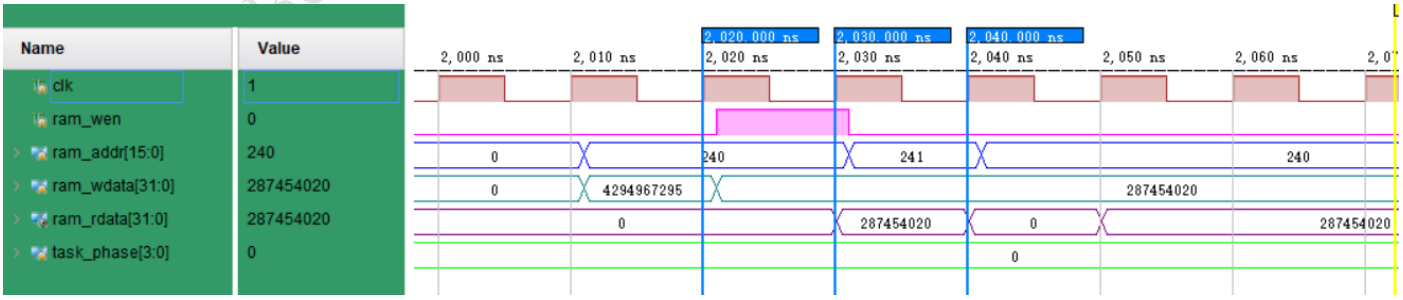


6. 从上图第 1 个 Marker 开始，到第 2 个 Marker，寄存器陆续读出我们刚才写入到 16-19 号寄存器内的数据，并向 20 号寄存器中写入新的数据，在第 3 个 Marker 处，我们看到，写入 20 号寄存器的内容已经改变为我们刚才写入的值。

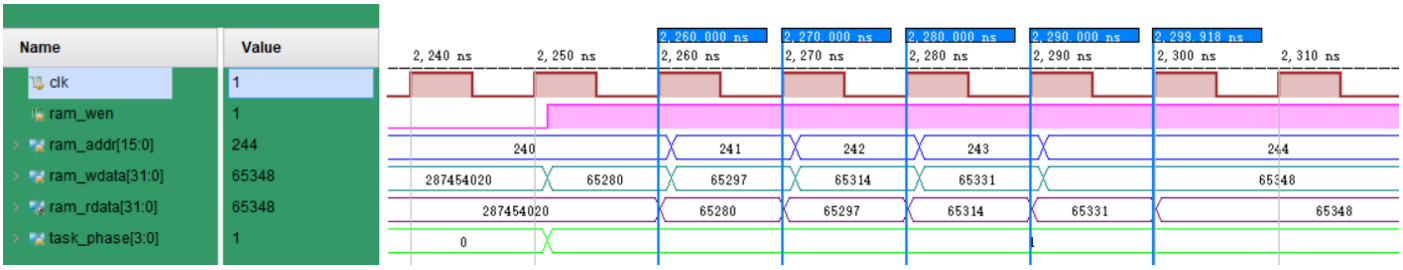
（三）子任务二

1、仿真行为对比分析

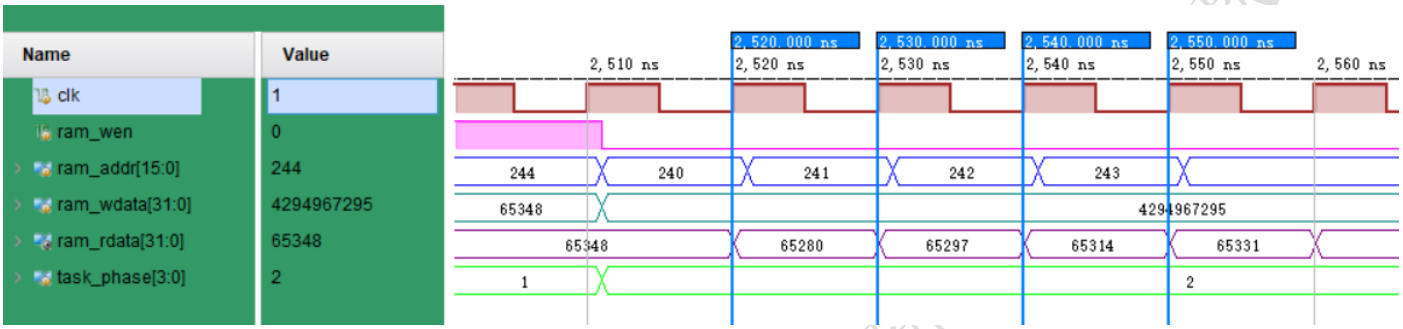
1. 同步 RAM



1. 第 1 个 Marker 时，读取出地址为 240 的内存的数据，第 2 个 Marker 时，在时钟上升沿，同步向 240 地址处写入数据，并读出 240 地址处的数据，第 3 个 Marker 时，同步读出 241 地址处的数据。

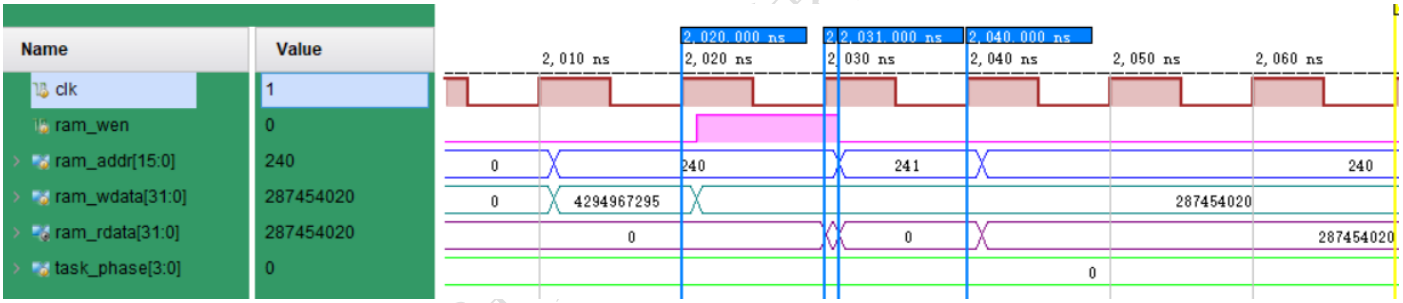


2. 从第 1 个 Marker 开始，到第 5 个 Marker，时钟上升沿时同步读写，同时向对应地址写入数据，并读出对应地址的数据，可以看到，写入应该是要比读出稍快一些，即，若写使能为 1，时钟上升沿同步读出的数据即为同步写入的数据。

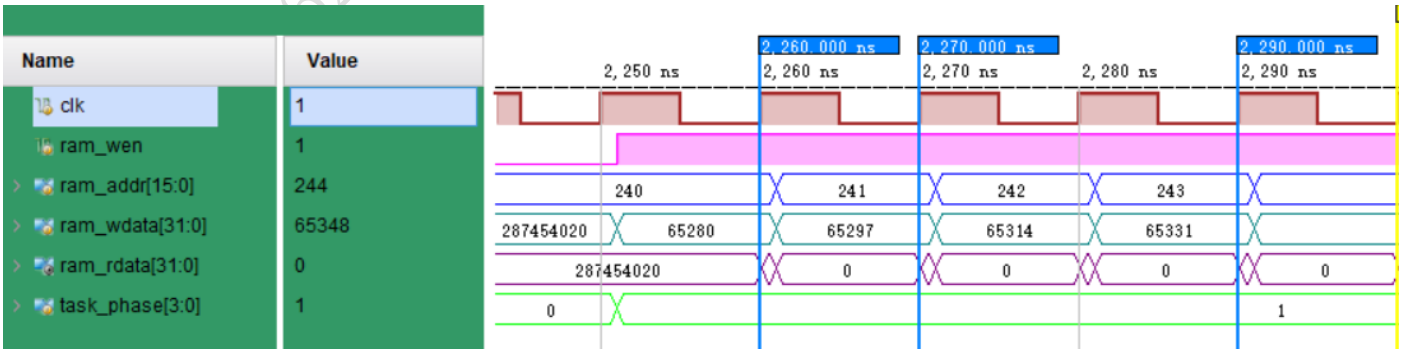


3. 可以看到，同步 RAM 读写都是与时钟同步的；

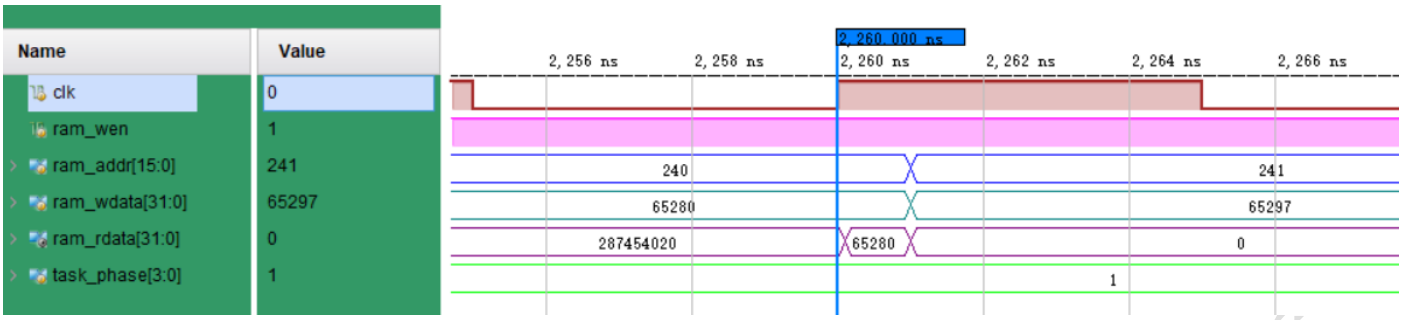
2. 异步 RAM



1. 第 1 个 Marker 时，读取出地址为 240 的内存的数据，第 2 个 Marker 时，在时钟上升沿，同步向 240 地址处写入数据，并读出 240 地址处的数据，第 3 个 Marker 时，读出 241 地址处的数据。
2. 我们看到，和同步 RAM 不同的是，在第 2 个 Marker 后迅速出现了 3 个 Marker 的变化，而且，变化并不与时钟信号同步，反而是和地址信号的变化同步；



3. 可以看到，每次同步写入数据后，读数据都会产生上述变化。



4. 放大后查看，确实，读数据的变化与地址变化同步，与时钟信号异步。

2、时序、资源占用对比分析

1. 时序对比

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.00	0	0
impl_1	constrs_1	route_design Complete!	0.805	0.0...	0.538	0.0...	0.000	0.155	0	129	4	58.00	0	0
Out-of-Context Module Runs														
block_ram_synth_1	block_ram	synth_design Complete!								124	4	58.00	0	0

（同步 RAM）

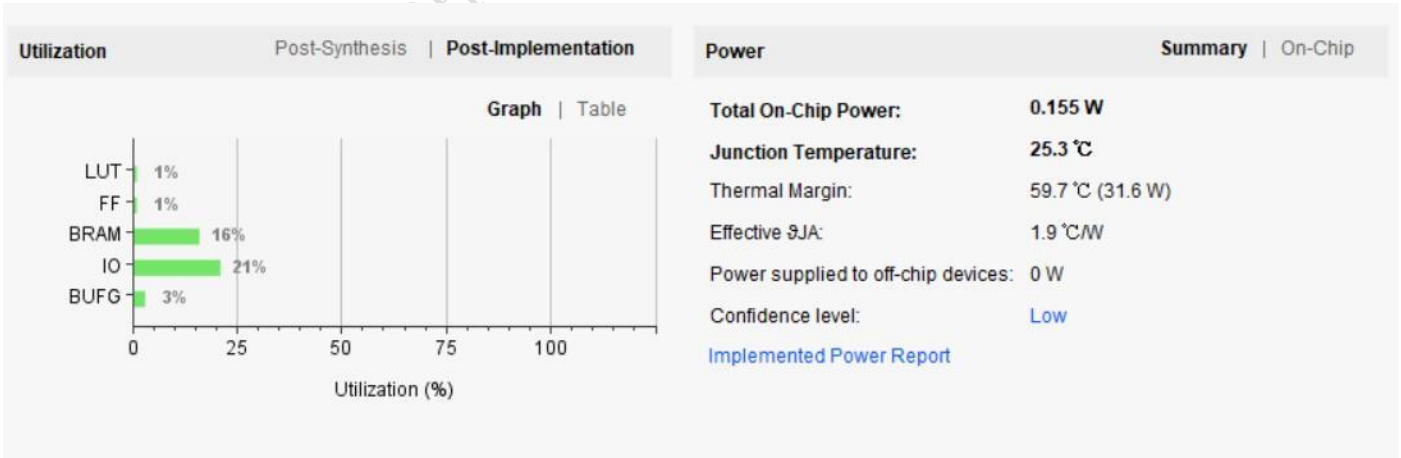
Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.00	0	0
impl_1	constrs_1	route_design Complete, Failed Timing!	-6.5...	-64...	0.153	0.0...	0.000	0.380	0	41357	0	0.00	0	0
Out-of-Context Module Runs														
distributed_ram_synth_1	distributed_ram	synth_design Complete!								41357	32	0.00	0	0

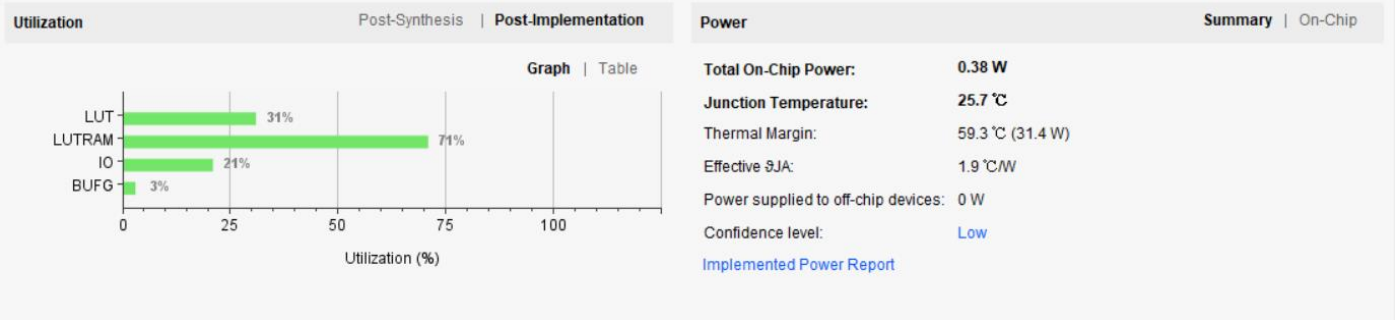
（异步 RAM）

（1） 在同步 RAM 中，WNS 为正值，说明同步 RAM 最长时序路径的 setup 没有违约；而在异步 RAM 中，WNS 值变为了负值，证明在当前的输入输出频率已经超过了异步 RAM 能达到的速度，因此，可以判断，同步 RAM 的输入输出频率比异步 RAM 要快；

2. 时序对比



（同步 RAM）



（异步 RAM）

（2） 同步 RAM 和异步 RAM 占用的 BUFG 和 IO 的比率相同，但是异步 RAM 的 LUT 占用比例比同步 RAM 多了 30%，LUTRAM 更达到了 71%，可见，异步 RAM 要比同步 RAM 占用更多的资源。

3、总结

1. 寄存器：同步写，异步读； 同步 RAM：同步写，同步读； 异步 RAM：同步写，异步读；
2. 通过时序分析：同步 RAM 的读取速度比异步 RAM 快；
3. 通过资源占用分析：异步 RAM 占用资源比同步 RAM 多；

（四）子任务三

1、错误 1：模块调用信号未连接

（1）错误现象

仿真波形信号为“Z”：

（2）分析定位过程

仿真波形显示 num_scn 信号值为 Z，要么是 wire 型的 num_csn 未被赋值，要么就是模块调用时信号未连接，

于是我回到顶层代码查看，发现 num_csn 已被赋值，所以又来到模块调用模块，发现 bug；

（3）错误原因

模块调用信号未连接导致的信号悬空：模块调用时，信号名称拼写错误，num_csn 写成了 num_scn。

（4）修正效果

修正方法：将 num_scn 改为 num_csn，成功。

```
show_num u_show_num(
        .clk        (clk      ),
        .resetn     (resetn   ),

        .show_data  (show_data),
        .num_csn    (num_scn  ),
        .num_a_g    (num_a_g  )
);
```

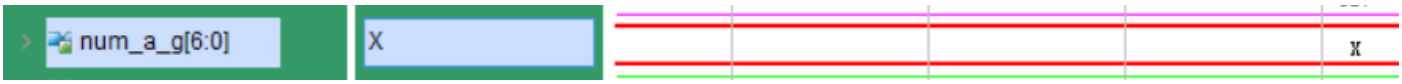
（5）归纳总结（可选）

可归类为信号为“Z”，模块调用时，细心检查信号名是否正确。

2、错误 2：信号为“X”

（1）错误现象

仿真波形信号为“X”：



（2）分析定位过程

仿真后查看波形，发现信号为 X，要么是多驱动赋值，要么是没有被赋值，或者，和没有赋值的变量进行运算，也会导致自身变为 X；

（3）错误原因

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

与从未被赋值的 Reg 型变量参与运算；

（4）修正效果

去掉注释：

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

有效：

（5）归纳总结（可选）

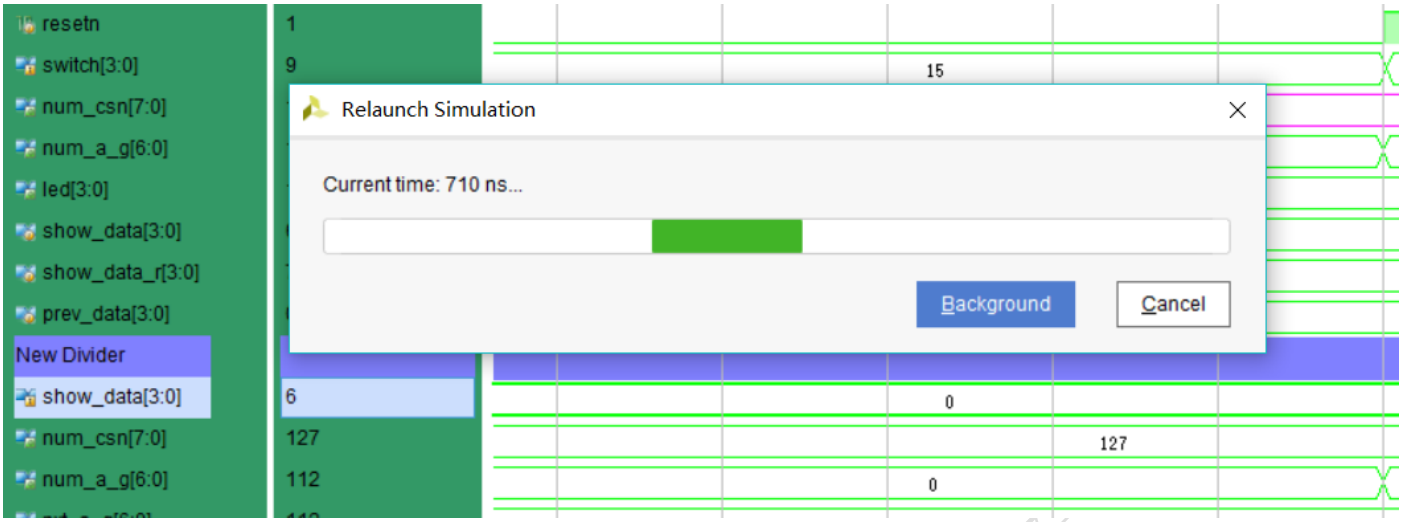
可归类为信号为 X，一旦发现某个信号为 X，则向前追踪产生该因子的信号，看看是哪个为 X，一直追踪下去，找到未赋值信号。

如果所有信号都有没有赋值的，则进行综合排查 error 和 crital warning。

3、错误 3：波形停止

（1）错误现象

仿真停止在某一个时刻，但却显示在不停的运行：



（2）分析定位过程

波形停止，很可能是产生了组合环，先综合，再查看 critical warning，定位到产生组合换的代码处，再吃检查；

（3）错误原因

组合逻辑的 nxt_a_g 产生组合环:

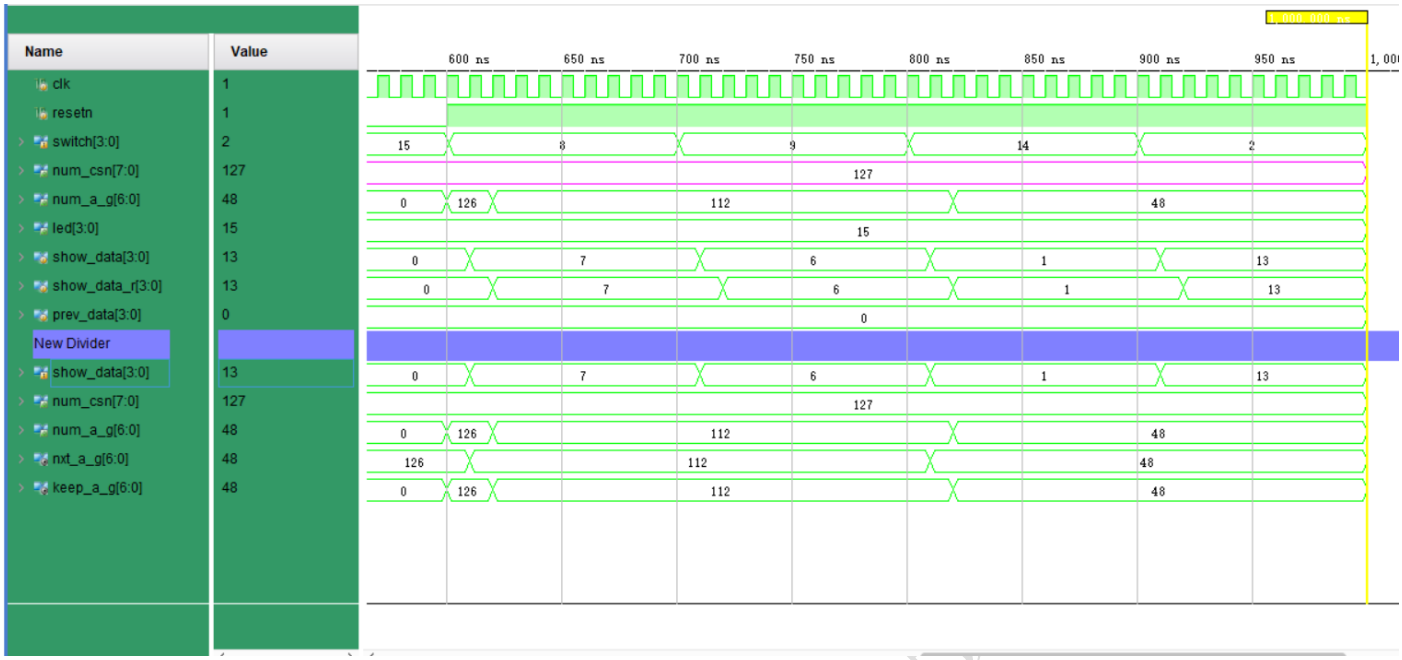
```
wire [6:0] keep_a_g;
assign     keep_a_g = num_a_g + nxt_a_g;

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   ;
```

（4）修正效果

修正方法：删除 nxt_a_g 变量；

```
assign     keep_a_g = num_a_g;
```

有效，成功跑完仿真：



（5）归纳总结（可选）

可以归类为波形停止，写组合逻辑时，注意变量的值是否受自身影响；

4、错误 4：越沿采样

（1）错误现象

变量 prev_data 始终为 0：

（2）分析定位过程

按照源码，prev_data 是不可能一直为零的，一定是赋值出了问题，果然在判断条件里，show_data_r 不应该和 show_data 相等，再往上找，就发现了在时序逻辑里采用非阻塞赋值的错误。

```
always @(posedge clk)
begin
    if(!resetn)
    begin
        prev_data <= 4'd0;
    end
    else if(show_data_r != show_data)
    begin
        prev_data <= show_data_r;
    end
end
```

（3）错误原因

在时序逻辑中采用非阻塞赋值：

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

（4）修正效果

将非阻塞赋值改为阻塞赋值：

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

有效，prev_data 的值产生变化：

（5）归纳总结（可选）

越沿采样，在时序逻辑中，绝对不能出现非阻塞赋值。

5、错误 5：功能缺失

（1）错误现象

没有办法显示数字 6；

（2）分析定位过程

在上板测试时，发现无法显示出数字 6，回来排查，发现缺少显示数字 6 的代码；

（3）错误原因

漏写显示出数字 6 的代码：（只有 1，2，3，4，5，7，8，9）

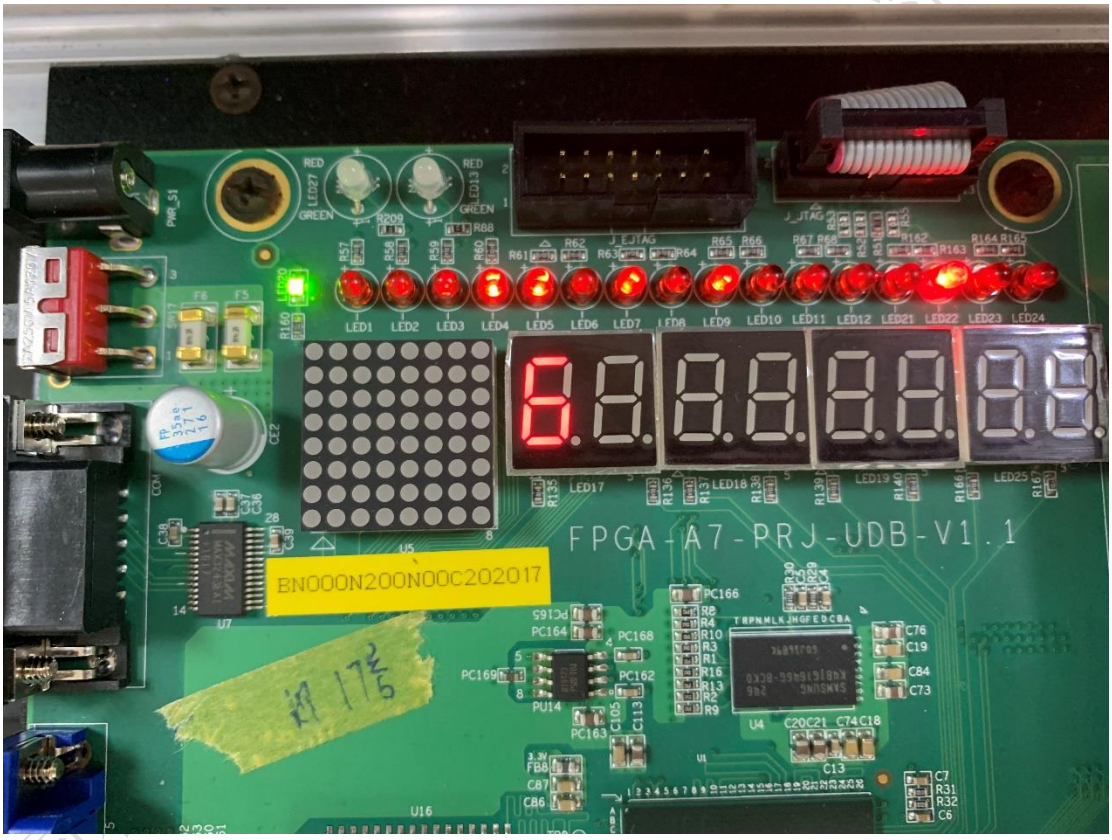
```
assign nxt_a_g = show_data==4'd0 ? 7'b1111110 :    //0
                 show_data==4'd1 ? 7'b0110000 :    //1
                 show_data==4'd2 ? 7'b1101101 :    //2
                 show_data==4'd3 ? 7'b1111001 :    //3
                 show_data==4'd4 ? 7'b0110011 :    //4
                 show_data==4'd5 ? 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   ;
```

（4）修正效果

修正方法：对比显示 1,2 的代码，补上显示 6 的代码：

```
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'd6 ? 7'b1111101 :    //6
                 show_data==4'd7 ? 7'b1110000 :    //7
                 show_data==4'd8 ? 7'b1111111 :    //8
                 show_data==4'd9 ? 7'b1111011 :    //9
                                   keep_a_g   ;
```

有效：



（5）归纳总结（可选）

粗心漏写代码，尽量细心。

四、实验总结（可选）

1.这次试验，加深了我对寄存器，同步 RAM，异步 RAM 读写方式的理解，对波形的变化有了更加深入的理解；

2.在 debugg 实验中，我熟悉了多种常见的 verilog 编程错误的现象，并初步掌握了 debugg 的能力，也学会了如何在写代码的时候尽量避免出现错误。