实验 6 报告 1

第 32 小组
袁峥

一、实验任务（10%）

1、完全带握手的类 SRAM 接口到 AXI 接口的转换桥 RTL 代码编写。

2、通过简单的读写测试。

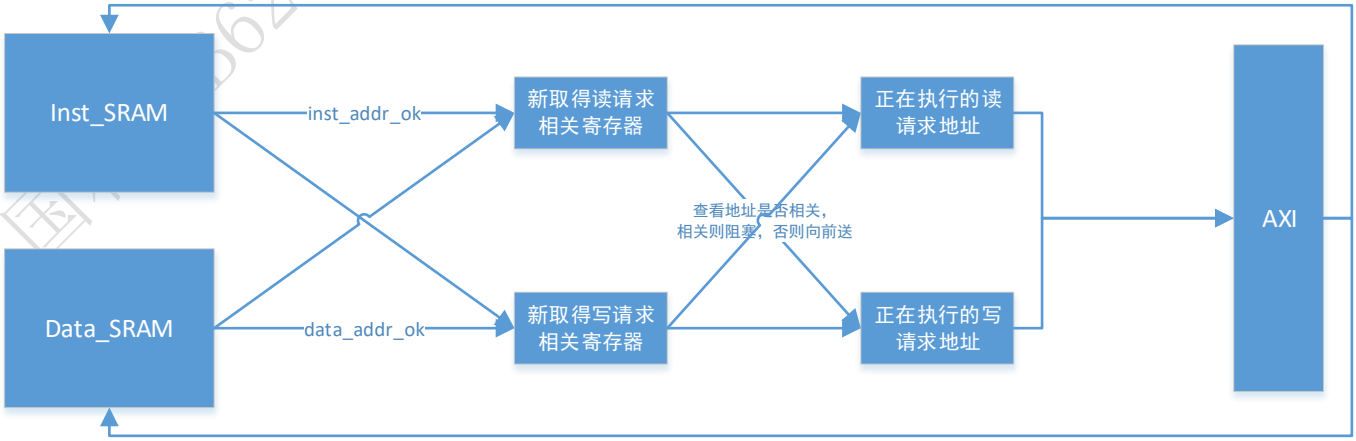
二、实验设计（30%）

此次实验需要完成两个类 SRAM 接口到一个 AXI 接口的转换，整体设计将读和写的内容分开，可以同时进行处理。

首先先将指令和数据存储器发出的请求按照读和写分类，使用 inst_addr_ok 和 data_addr_ok 信号作为读写操作请求的准入信号，如果指令和数据存储器同时发出读请求，先满足数据存储器。并将取得的读写请求相关信号（addr、size、wdata）存入一级寄存器，再将传入的地址进行判断，如果新发出的读请求的地址与正在执行的写地址的信号相等或新发出的写请求的地址与正在执行的读地址的信号相等，那么将新传入的请求阻塞，直到地址不相关再往前送，否则直接将新的请求存取后一级寄存器，进行真正的读写操作。

将真正需要执行的读写请求的信号送入 AXI 接口，等到操作完成的信号（rvalid&&rready、bvalid&&bready）有效，则再将相应的 inst_addr_ok 或 data_addr_ok 置 1，取得新的读写操作请求。同时每次完成读写请求后，将相应寄存器中的信号清空。

整个实验设计的流程图如下：



三、实验过程（60%）

（一）实验流水账

11 月 26 日下午 3 点至 10 点，阅读文档 L06_CPU 总线接口支持，并查看 AXI 协议。

11 月 27 日下午 3 点至 11 点，编写代码，完成调试、仿真及上板。

11 月 28 日上午 9 点至 11 点，完成实验报告。

（二）错误记录

1、错误 1

（1）错误现象

仿真一开始就卡住，没有任何输出内容。

（2）分析定位过程

查看波形后发现是由于连第一条读写操作请求都没有取到。

（3）错误原因

再次仔细阅读了任务书及课件后，发现自己对 addr_ok 信号的理解有问题，inst_addr_ok 和 data_addr_ok 信号为 1 时，inst_addr/data_addr、data_wdata 信号才会有效，而不是我之前所理解的先发送 inst_addr/data_addr、data_wdata，等到 addr_ok 为 1 时再采样，因此没有能取进第一个请求。

（4）修正效果

正确理解 addr_ok 信号的含义后，发现其可以作为请求的准入信号，即 addr_ok 有效后，将相关请求先取得并放入缓冲存储器等待。修改代码后仿真中尽管仍旧有错，但跟踪波形后发现第一个请求已经取得。

（5）归纳总结

这个错误主要是由于自己对 addr_ok 信号的理解不够深刻，一方面可以通过在写代码之前反复阅读文档，并思考信号间相关逻辑来加深理解，另一方面，在写代码后的调试中发现错误并改正本身也可以加深对该信号的理解。

2、错误 2

（1）错误现象

仿真时通过指令存储器有关的 5 个测试点，但是在数据存储器的第二个测试数据上发生错误，取得的数据与正确情况不一致。

（2）分析定位过程

查看了仿真波形和 vitual_cpu.v 的代码，发现在读写同一个地址的操作同时分别进入读通道和写通道，产生的请求是先写再读，但实际读出的数据是完成写操作之前的数据。

```
//1 :write 2 bytes
assign data_wr_v         [1] = 1'b1;
assign data_size_v       [1] = 2'd1;
assign data_addr_v       [1] = 32'h0000_8042;
assign data_wdata_v      [1] = 32'h2222_2222; //mem[8'h40] = 32'h2222_0000
//2 :read 4 bytes
assign data_wr_v         [2] = 1'b0;
assign data_size_v       [2] = 2'd2;
assign data_addr_v       [2] = 32'h0000_8040;
assign data_rdata_ref_v  [2] = 32'h2222_0000; //mem[8'h40][31:0]
assign data_rdata_ref_m_v[2] = 32'hffff_ffff;
```

（3）错误原因

在设计上存在缺陷，并没有考虑到同时读写同一个存储器地址的情况。

（4）修正效果

考虑到在将 addr_ok 信号置 1 前，无法获得操作地址，也就无法提前判断地址是否相关，因此需要增加一级缓冲寄存器，先将读写操作的信号存入缓冲寄存器，然后判断新取得的操作与正在执行的操作的地址是否相同，如果相同，那么将新取得的操作暂时阻塞，等到正在执行的地址相关请求完成后再执行，否则直接将缓冲寄存器中的信号往后送，进行真正的读写请求处理。代码完善后再次进行仿真，仍然在此处发生错误，原因见错误 3。

（5）归纳总结

这个错误比较严重，属于整体设计上的缺陷，需要对代码进行大范围调整，十分浪费时间。在进行代码编写前阅读文档时遗漏了这个问题，没有进行细致考虑。

3、错误 3

（1）错误现象

仿真时仍旧无法通过数据存储器的第 2 个测试点。

（2）分析定位过程

再次查看波形，发现在发生数据读写请求同时分别进入读和写通道，且地址相关时，后来的请求并没有如预期进行阻塞，而是直接继续执行。

（3）错误原因

由于在解决错误 2 时已经加入了判断地址是否相关的逻辑，但实际效果来看并没有生效，因此考虑是判断地址相同的地方出现了错误。再次查看 vitual_cpu.v 后知道了错误原因，先发出的写操作的地址是 32’h00008042，后发出的读操作的地址是 32’h00008040，这两个地址尽管指向同一个 32 位的地址，但是最后 2 位不同，而之前的代码中判断地址相同是用的 32 位地址相同来比较的。

（4）修正效果

将判断地址相关的代码从 read_addr_temp != write_addr 改为 read_addr_temp[31:2] != write_addr[31:2]。

（5）归纳总结

这个问题主要还是由于粗心造成的，其实在看到测试数据后一下就应该想到，而且在经过实验 5 的地址错例外相关处理后，对于地址没有对齐的情况应该格外关注。

4、错误 4

（1）错误现象

仿真时在数据存储器的第 4 个测试点结束后卡死，第 5 个测试点的结果没有打印。

（2）分析定位过程

查看波形后发现，在读取第 5 个测试点的请求后，arvalid 信号一直没有置 1，从而倒是请求无法继续完成，被卡住。

（3）错误原因

在完成数据存储器的同时读写请求的地址相关处理后，增加了缓冲寄存器，因此并非一取得操作信号后，寄存器中的值就是有效的，需要进行地址相关判断后再送入后一级寄存器来真正执行请求，因此 arvalid 对应的应该是 read_addr 中的信号而非缓冲寄存器 read_addr_temp 中的信号，而当时 arvalid 信号中使用了错误地使用了 read_addr_go 这一个表示允许 read_addr_temp 寄存器传到 read_addr 寄存器的信号，导致由于地址相关被阻塞后 arvalid 一直无法置 1，从而无法完成后续的操作。

（4）修正效果

更新了 arvalid 信号的逻辑，加入了判断 read_addr == read_addr_temp，即表示缓冲存储器中的值已经被传入真正执行操作的地址寄存器。对应的，也更新了 awvalid 信号，加入了判断 write_addr == write_addr_temp。

（5）归纳总结

这个错误是由于在设计上的不严谨，考虑问题不全面，从而在信号的逻辑上出现了错误，是的流水级出现了阻塞。特别是 wire 和 reg 型信号混用的地方，十分容易出现差一拍而导致与预想的情况不一样，以后在处理这些地方的时候可以先仔细理清每一拍后各级寄存器中的内容，看是否与设想一致。

四、实验总结

（一）组员：袁峥

这次的实验主要花费的时间是在阅读任务书和查看 AXI 协议手册来理解信号含义上，这部分工作花了大约一般的时间，尽管如此在编写代码后的仿真阶段还是发现了信号理解有偏差的情况。总体感觉上来说这次的实验和之前的感觉不太一样，之前的实验基本上逻辑比较清晰，就是具体的代码实现上有许多细节需要考虑全面，而总线的实现上，对于各信号之间的逻辑的理解上十分不容易，理清逻辑花费了很长的时间，在完全搞懂各信号含义后实际编写的代码量不是很大。

在实验的整体设计上，简要的说是采取了读写分开并行完成，同时针对每一个读操作，直到完成了这个读操作，再取入下一个读操作请求，对于写操作也是相同。但这样的设计其实会比较慢，在完成了这次实验后我又进行了思考，发现其实还可以进一步改进。

对于每一个读操作，在读地址被接收后，即 arvalid && arready 时，便可以接着读取下一个读操作，这样可以

加快执行速度，但是在设计上便需要在模块中加入 buffer，来记录当前未完成的读请求的地址、数据或指令请求等信息，以便进行地址相关和判断和返回读数据时的通道选择。在代码实现上需要使用一个 FIFO 的 buffer，增加 buffer 计数器并考虑 buffer 满的情况，对于写操作同样的也需要加入类似的 buffer。

由于是在完成本次实验后才想到的改进方法，因此没有连夜熬夜进行修改。考虑到本次实验的内容相对于 CPU 独立，因此后续有时间进行修改后可以直接拼接使用，争取下周完成这部分的优化。