****

**《计算组成原理实验》**

**实验报告**

**（实验四）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 计算机学院 | | | | | |
| **专业（班级）** | **：** | 21级行政1班 | | | | | |
| **学生姓名** | **：** | 黄鑫 | | | | | |
| **学 号** | **：** | 21307008 | | | | | |
| **时间** | **：** | 2023 | 年 | 1 | 月 | 22 | 日 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | **成绩** | **:** |  |
| **实验四** | **：** | **总线（YstCPU Risc-V）** | | | |

**目录**

[第一章 实验目的 3](#_Toc125244406)

[第二章 实验器材 3](#_Toc125244407)

[第三章 实验任务 3](#_Toc125244408)

[第四章 实验原理与分析 3](#_Toc125244409)

[1. 总线 3](#_Toc125244410)

[2. 写操作的流程 3](#_Toc125244411)

[第五章 实验过程与结果 5](#_Toc125244412)

[1. 代码实现过程 5](#_Toc125244413)

[2. 标准测试全部通过 9](#_Toc125244414)

[第六章 实验心得 10](#_Toc125244415)

# 实验目的

1. AXI4-Lite 总线协议原理

2. 使用状态机实现总线协议

# 实验器材

电脑一台，Intellij IDEA 软件一套。

# 实验任务

lab4中的实验任务如下所示：

主从设备的代码位于 src/main/scala/bus/AXI4Lite.scala，请在标有 //lab4 (BUS) 的注释处，实现状态机的状态切换，以通过 BusTest，和 CPUTest。

# 实验原理与分析

本实验采用AXI4-Lite 总线协议原理，并用状态机实现。

## 总线

总线通道共5条，写操作有3条，分别为：地址通路、数据通路、回复通路；读操作有2条，相比于写操作少的那条是回复通路——由于数据传输方向的特点，直接省去了。

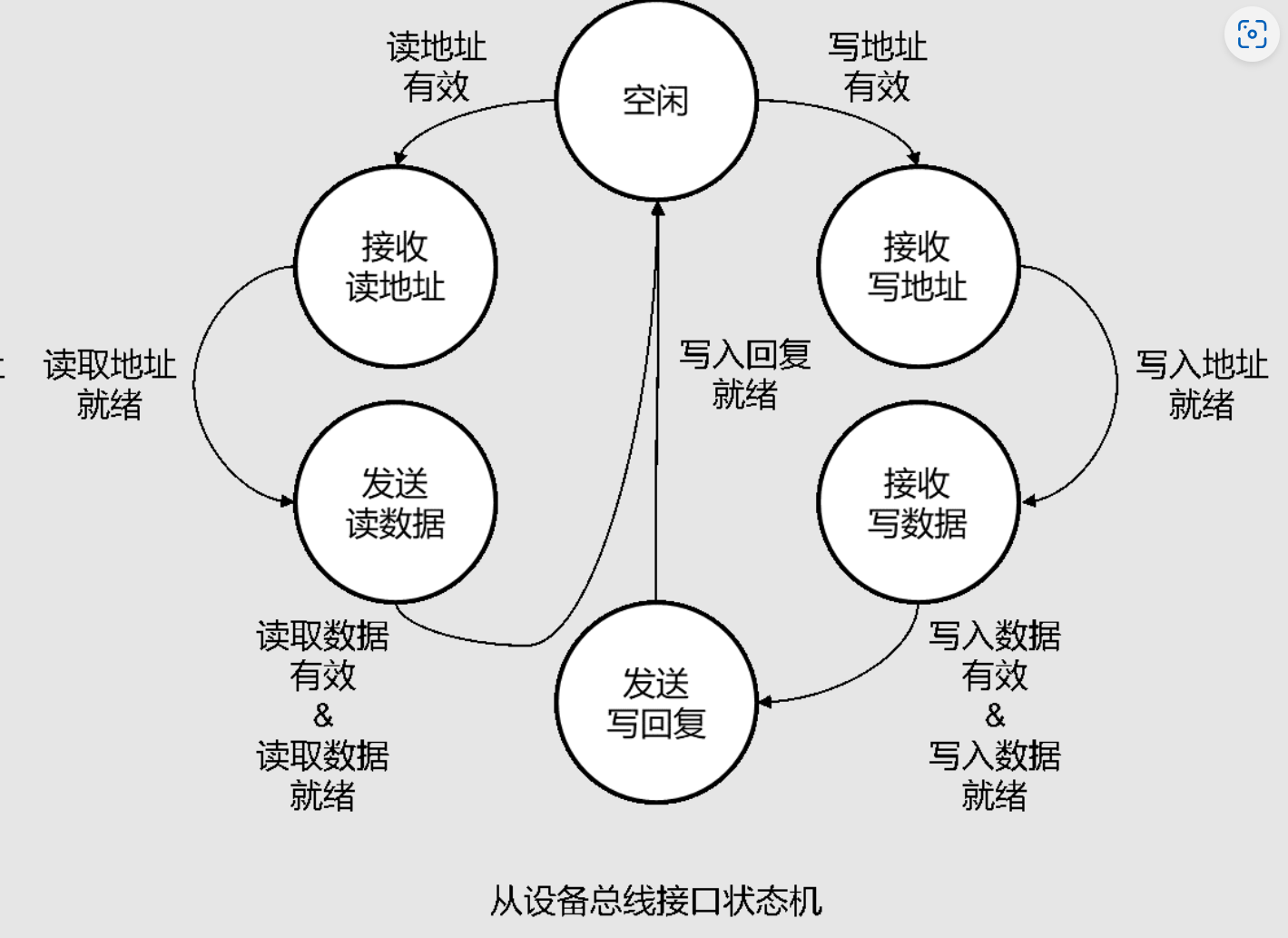
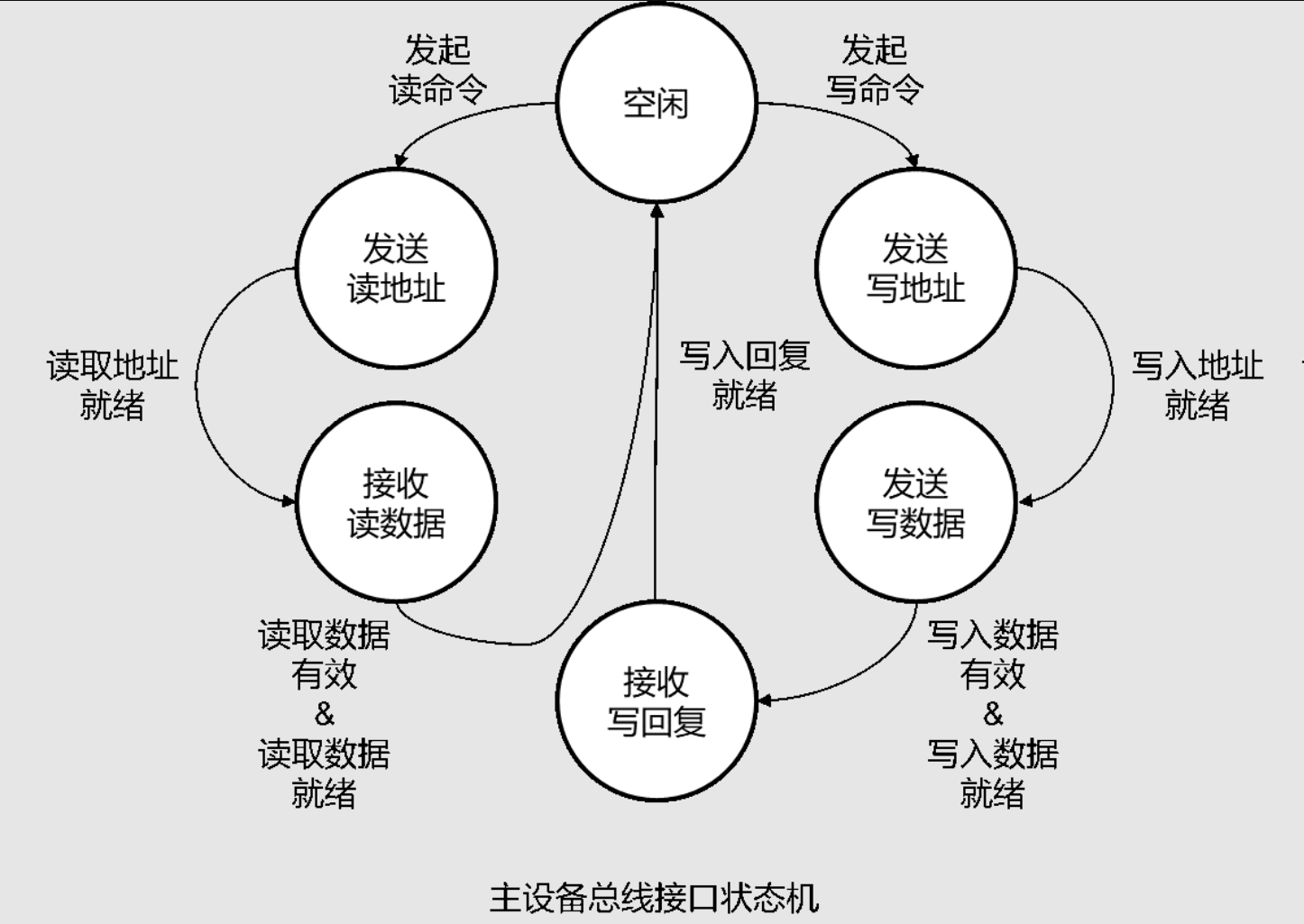
AXI4-Lite的握手方式是VALID/READY 机制，通信双方分别扮演发送方和接收方，两者的操作并不相同。发送方置高 VALID 信号表示发送方已经将数据，地址或者控制信息已经就绪，并保持于消息总线上；接收方置高 READY 信号表示接收方已经做好接收的准备。我认为实际上，两者的到达时机以同时为佳。

实验手册写的流程不明不白的，我再捋一捋。

由于读操作只是写操作的简化版本，下面就只分析写操作。

## 写操作的流程

首先研究读操作时，主从两设备的状态转换流程。 



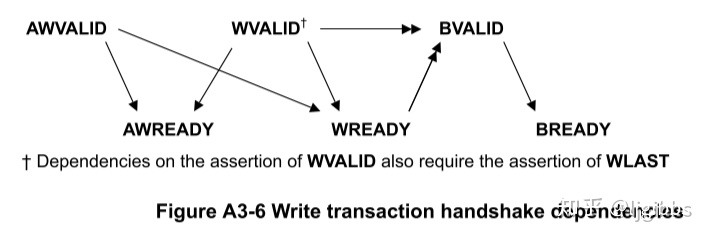
初始时，主从机都是Idle状态，这时主机收到来自CPU的写内存请求，此时主机记录下来自CPU的write\_data/write\_strobe/addr等写入数据信息，并进入Write\_Addr状态。

在WA状态中，主机向从机发出AWVALID和AWADDR，从机接收到AWVALID后便进入WA状态并向主机发出AWREADY。主机的AWVALID&&AWREADY有效时，便进入Write\_Data状态并关AWVALID；从机的AWVALID&&AWREADY有效时，便接收AWADDR并进入Write\_Data状态并关AWREADY。

在WD状态中，主机向从机发送WVALID和WDATA，从机向主机发送WREADY。主机WVALID&&WREADY有效时，便向从机发送write\_data并进入Write\_Resp状态并关WVALID；从机WVALID&&WREADY有效时，便向外设发送写请求write与相关信息（地址、数据、strobe）并进入Write\_Resp状态并关WREADY。

在WRESP状态中，主机向从机发送BREADY，从机向主机发送BVALID。主机BVALID&&BREADY有效时，便向CPU发送写成功write\_valid信号并进入Idle状态并关BREADY；从机BVALID&&BREADY有效时，便关闭外设写请求write信号与相关信息（地址、数据、strobe）并进入Ilde状态并关BVALID。

下面是状态机控制信号之间的官方依赖关系，仅作参考，与我的实现不符：



注：图中的单头箭头表示：其指向的信号可以在箭头起始信号置起之前或之后置起（无依赖）

图中的双头箭头表示：其指向的信号必须在箭头起始信号置起之后置起（指向信号依赖起始信号）

# 实验过程与结果

## 代码实现过程

值得注意的地方有：

1. 我在研究的过程中，将AXI4-Lite.scala文件弄得规整并填加大量注释，值得一看
2. 在具体实现中，我在注释中将外设与内存统称“外设”。
3. 在具体实现中，助教大大已经实现了大量连线。其中为了保证信号在上升沿发出，状态机中创建了它们的寄存器，并统一在下一周期初发送到CPU/通道/外设。

现列出在程序实现使用到的端口和寄存器：

Slave

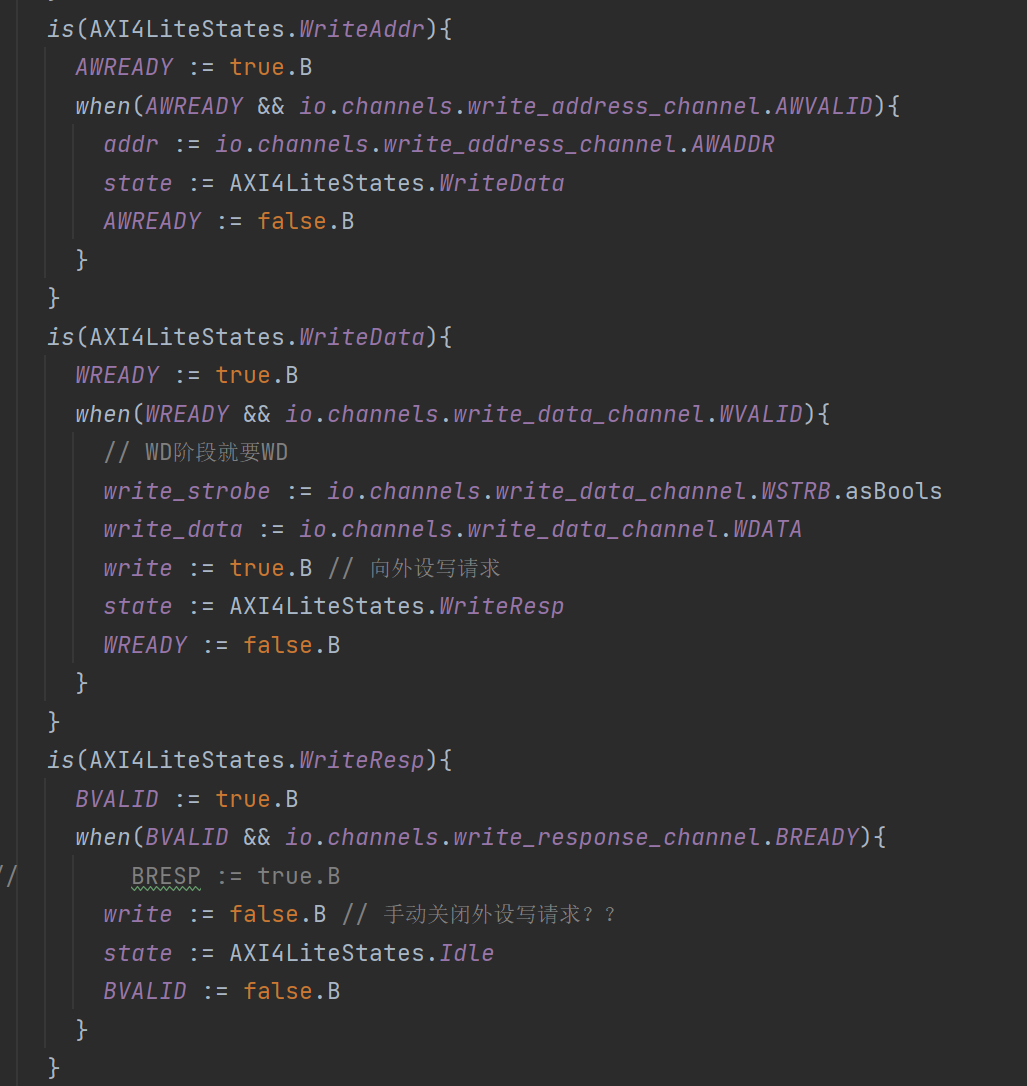
|  |  |
| --- | --- |
| 端口或寄存器名 | 用途 |
| state | Slave状态寄存器 |
| ARREADY | 置高则向主机发送ARREADY |
| addr | 输入则向外设发出读写地址 |
| read | 置高则向外设发出读请求 |
| RVALID | 置高则向主机发送RVALID |
| AWREADY | 置高则向主机发送AWREADY |
| WREADY | 置高则向主机发送WREADY |
| write | 置高则向外设的写请求 |
| write\_data | 输入则向外设发送写数据 |
| write\_strobe | 输入则向外设发送strobe |
| BVALID | 置高则向主机发送BVALID |

Master

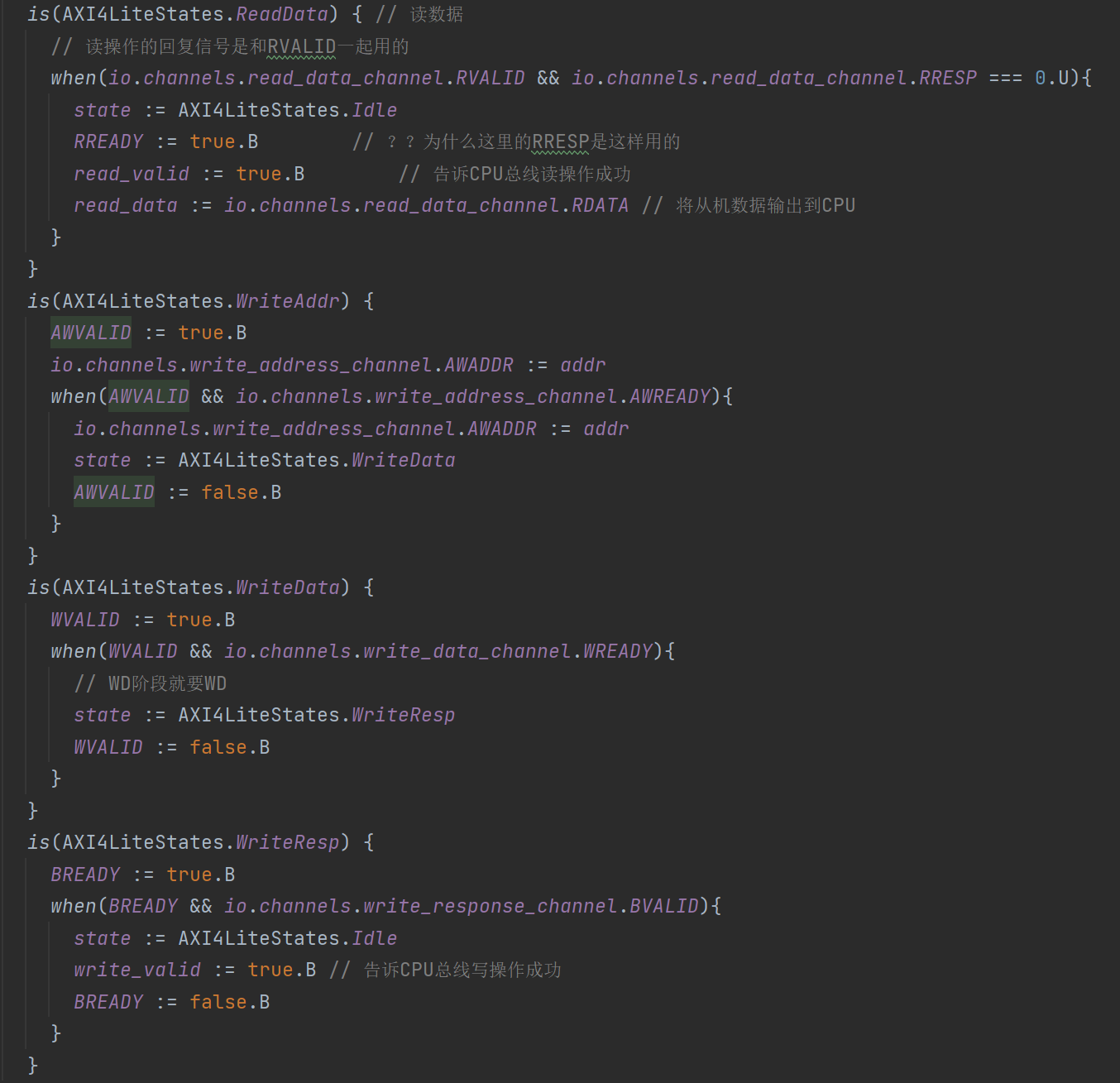
|  |  |
| --- | --- |
| 端口或寄存器名 | 用途 |
| state | Master状态寄存器 |
| read | 来自CPU的读请求 |
| write | 来自CPU的写请求 |
| addr | 保存读写地址（需手动设置从CPU获取/发送到从机的时机） |
| write\_data | 输入则向从机发送写数据 |
| write\_strobe | 输入则向从机发送strobe |
| ARVALID | 置高则向从机发送ARVALID |
| RREADY | 置高则向从机发送RREADY |
| read\_valid | 置高则向CPU发送读成功信号 |
| read\_data | 输入则向CPU输送读数据 |
| AWVALID | 置高则向从机发送AWVALID |
| BREADY | 置高则向从机发送BREADY |
| write\_valid | 输入则向CPU发送写成功信号 |

捋清楚执行流程与端口或寄存器名后，我们就能动手实现状态机了：  
Slave：





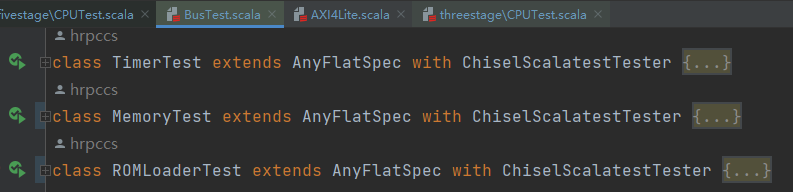
Master：

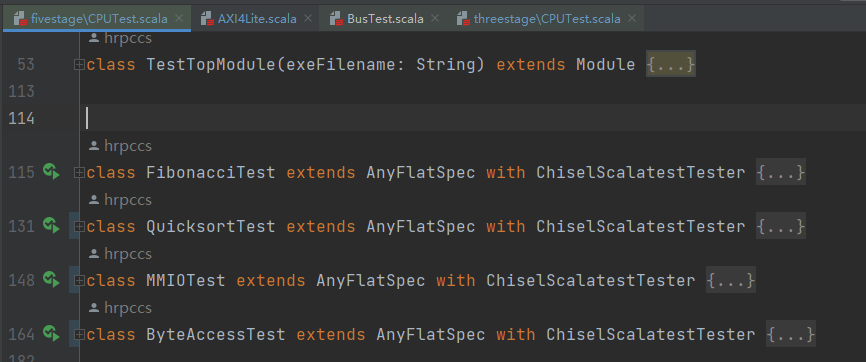
## 标准测试全部通过

注：运行符左边的√就是测试通过的意思。

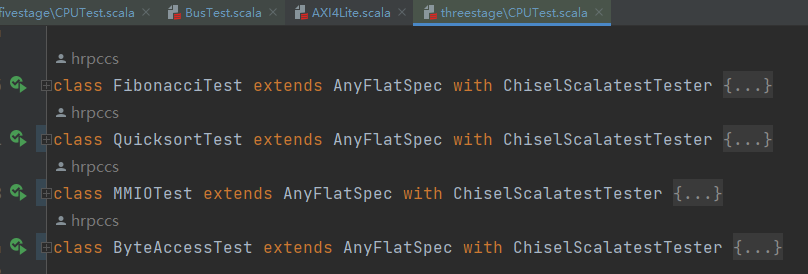
BusTest



FiveStage CPU



ThreeStageCPU



# 实验心得

这个实验的一个难点在于自学AXI4Lite总线协议并实现，因为实验手册的描述十分简略，而且AXI4Lite协议存在一定的自由发挥空间。

而另一个难点在于理解助教大大[hrpccs](https://github.com/hrpccs)程序中的各种接口与寄存器的用途。真的不梳理一下就不会做呀……

剩下的问题：我还是不很理解回复信号resp的用法……

查了资料才知道，resp不仅要让主机知道从机操作成功没有，如果错误还要报告错误的原因：

操作回复信号BRESP\RRESP：

1. OKEY 0 正常访问成功
2. SLVERR 2 从机错误
3. DECERR 3 解码错误，比如没有从机的地址。

由于只要一切正常，就不用修改回复信号的值。而用得到回复信号的场合，大概则是实现总线异常处理的时候了（会不会与OS有关呢？）。