版本历史

文档更新记录	文档名:	Lab06_CPU 总线接口支持
	版本号	V0.1
	创建人:	计算机体系结构研讨课教学组
	创建日期 :	2017-11-20

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2017/11/20	邢金璋	V0.1	初版。

文档信息反馈: xingjinzhang@loongson.cn

1 实验六 CPU 总线接口支持

在学习并尝试本章节前，你需要具有以下环境和能力：

(1) 较为熟练使用 Vivado 工具。

(2) 一定的 CPU 设计与实现能力。

通过本章节的学习，你将获得：

(1) AXI 协议的知识。

(2) CPU 总线接口设计的知识。

在本章节的学习过程中，你可能需要查阅：

(1) 参考资料“AMBA 总线协议”。

(2) 课本上总线知识。

在开展本次实验前，请确认自己知道以下知识：

(1) 总线协议的握手的概念。

(2) AXI 协议中握手信号 ready 和 valid 的概念。

(3) AXI 协议 5 个通道的概念。

(4) AXI 协议各控制信号的意思。

本次实验有以下几个坑，请在碰到问题时注意查看：

(1) 对 AXI 接口的 ar、aw、w 通道，如果 master 先置上了 valid，但没有收到对应的 ready，也就是握手失败了，但这时不能更改该通道的其他信号。比如 arvalid 置上时，未看到 arready，master 不能更改 araddr。**也就是 AXI 一旦发起请求，就不能撤销或更改请求，直至请求握手完成**。

(2) 但是第一阶段我们设定的类 SRAM 接口，却只保证 req&addr_ok 有效时，也就是握手成功时，addr 和 wdata 是有效的。其他时候，**即使 req 有效，但 addr_ok 为 0，addr 和 wdata 也是 x**。这样的设定虽然对第一阶段实验而言比较麻烦，却方便了第二阶段的实现。

(3) Lab6 上板运行时，可以根据拨码开关更改随机种子。所以有可能上板运行时，更改了随机种子，测试就跑不过了。这一情况在第一阶段可能比较少碰到，但在第二阶段应该会普遍碰到。一旦碰到，就需要仿真设定相同随机种子，来仿真复现该错误进行 debug。

以下几点，第一阶段可能碰不到，但第二阶段可能会碰到，最好可以提前考虑下：

(1) AXI 读写分类，对先写后读，可能读先完成的。也就是先写后读，且写读同一地址，如果在写为收到 b 通道时就发出了读的 ar 通道，那就有可能读出旧值，这是不正确的。类似的，先读后写，且读写同一地址，读未完成，写就发出了，那有可能读到写后的值，这也是不正确的。

1.1 实验目的

1. 掌握片上总线的一般性原理。
2. 掌握总线接口与 CPU 内部流水线之间的相互关系。

1.2 实验设备

1. 装有 Xilinx Vivado、MIPS 交叉编译环境的计算机一台。
2. 龙芯体系结构教学实验箱（Artix-7）一套。

1.3 实验任务

为 myCPU 增加 AXI 总线支持，完成功能测试，并支持运行一定的应用程序。

本次实验分两周完成，第一周（2017 年 11 月 28 日检查），需要完成：

(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。

第二周（2017 年 12 月 5 日检查），需要完成：

(1) CPU 顶层修改为 AXI 接口。CPU 对外只有一个 AXI 接口，需内部完成取指和数据访问的仲裁。

(2) 集成到 SoC_AXI_Lite 系统中。

(3) 完成功能测试。

(4) 在 myCPU 上运行 lab4 的电子表程序，要求能实现相同功能。

1.4 实验环境

本实验第一周的实验环境如下（黄色部分为实验内容），记为 cpu_axi_ifc_dev。

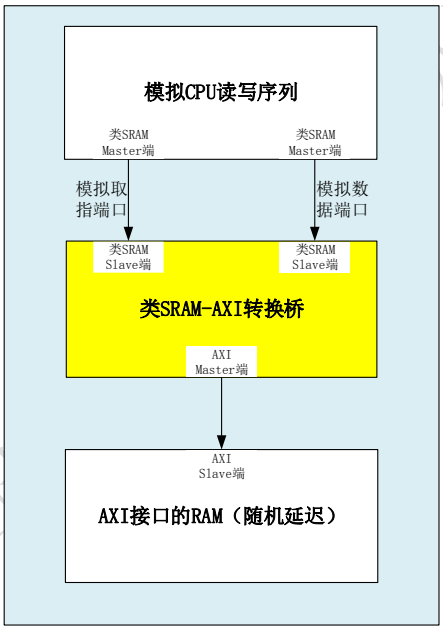


图 1-1 cpu_axi_ifc_dev 环境示意

本实验第二周的实验环境基于 ucas_CDE_v0.3，其中的 mycpu_verify 的环境不再使用 SoC_Lite 系统，而是使用 SoC_AXI_Lite 系统，该系统如下（黄色部分为实验内容）。生成 golden_trace 的部分与原先相同，依然基于 SoC_Lite。测试程序为 lab5_func_2 的更新，本次只有一块 RAM，只用加载 inst_ram.coe 即可（功能测试程序与数据段）。

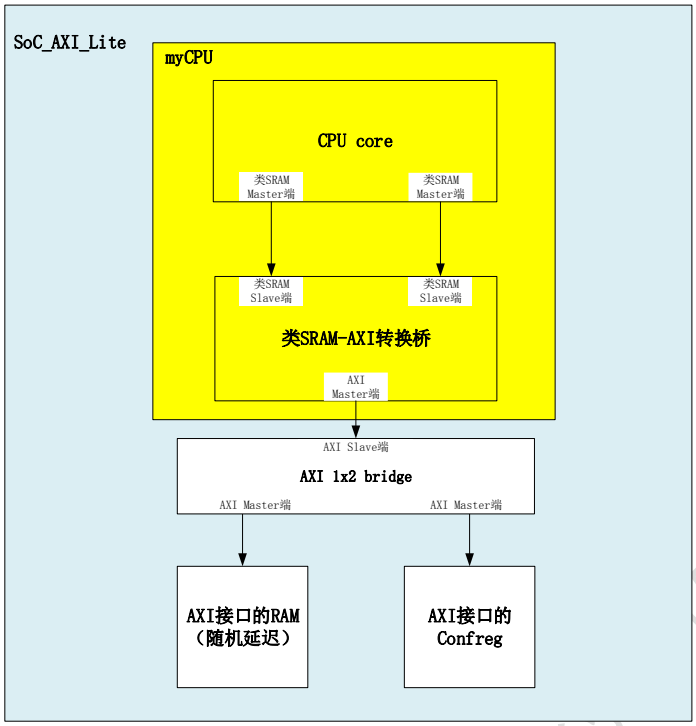


图 1-2 SoC_AXI_Lite 示意

1.5 实验检查

本次实验在 2017 年 11 月 28 日、2017 年 12 月 5 日分别进行检查。现场分仿真检查和上板检查。

第一阶段仿真和上板效果参见本章 1.7.3 节。

第二阶段仿真和上板效果参见本章 1.7.4 节。

1.6 实验提交

本次实验作品提交要求提交两次，第一次截止日期是 2017 年 11 月 28 日 18:00，第二次截止时间是 2017 年 12 月 5 日 18:00。

(1) 纸质档提交

提交方式：课上现场提交，每组都必须要有。

截止时间：2017 年 11 月 28 日 18:00、12 月 5 日 18:00。

提交内容：纸质档 lab6 各阶段实验报告，分别记为 lab6-1、lab6-2。

实验报告模板参考“A06_实验报告模板_v0.2”。

(2) 电子档提交

提交方式：打包上传到 Sep 课程网站 lab2 作业下，每组都必须要有。

截止时间：2017 年 11 月 28 日 18:00、12 月 5 日 18:00。

提交内容：电子档为一压缩包。

第一阶段提交目录层次如下（请将其中的“*组号*”，替换为本组组号）。

|-lab6-1_*组号*/	目录，lab6-1 作品。
| |--lab6-1_*组号*.pdf/	Lab6-1 实验报告，实验报告模板参考“A06_实验报告模板 v0.2
| |--**axi_ifc /**	**目录，自实现类 SRAM 接口到 AXI 接口的转换桥源码**

第二阶段提交目录层次如下（请将其中的“*组号*”，替换为本组组号）。

|-lab6-2_*组号*/	目录，lab6-2 作品。
	|--lab6-2_*组号*.pdf/	Lab6-2 实验验报告，实验报告模板参考“A06_实验报告模板 v0.2
	|--**myCPU /**	**目录，自实 myCPU 源码，最好有 readme 说明**

1.7 实验说明

1.7.1 类 SRAM 接口

之前实现的 myCPU 接口是 SRAM 接口，数据访问都是单周期返回的。接口简单，却也限制了 myCPU 的频率。myCPU 频率不能超过 RAM 的读写频率。

实际 CPU 频率是普遍高与存储器读写频率的，所以很多时候访问都是需要多个 CPU 周期才能返回的。为此为 SRAM 接口增加地址传输握手信号 addr_ok 和数据传输握手信号 data_ok，这样就可以实现任意周期返回数据了，本课程中称为类 SRAM 接口。

表 1-1 类 SRAM 接口信号

信号	位宽	方向	功能
clk	1	input	时钟
req	1	master—>slave	请求信号，为 1 时有读写请求，为 0 时无读写请求
wr	1	master—>slave	该次请求是写
size	[1:0]	master—>slave	该次请求传输的字节数，0: 1byte；1: 2bytes；2: 4bytes。
addr	[31:0]	master—>slave	该次请求的地址
wdata	[31:0]	master—>slave	该次请求的写数据
addr_ok	1	slave—>master	该次请求的地址传输 OK，读：地址被接收；写：地址和数据被接收
data_ok	1	slave—>master	该次请求的数据传输 OK，读：数据返回；写：数据写入完成。
rdata	[31:0]	slave—>master	该次请求返回的读数据。

需要注意，不同于 SRAM 接口，类 SRAM 的地址信号(addr)是字节寻址的，其指向读写数据的最低有效位。因而 addr 和 size 信号需配合使用，不支持 3 字节读写，有且只有以下类型组合。

1. addr[1:0]=2’b00 时，可能的组合：
   size=2’b00, size=2’b01, size=4’b10,
2. addr[1:0]=2’b01 时，可能的组合：
   size=2’b00
3. addr[1:0]=2’b10 时，可能的组合：
   size=2’b00, size=2’b01
4. addr[1:0]=2’b11 时，可能的组合：
   size=2’b00

wdata 和 rdata 有效数据字节也与 size 与 addr[1:0]信号对应，小尾端下，配合如下：

表 1-2 类 SRAM 接口数据有效情况

	data[31:24]	data[23:16]	data[15:8]	data[7:0]
size=2’b00,addr=2’b00	-	-	-	valid
size=2’b00,addr=2’b01	-	-	valid	-
size=2’b00,addr=2’b10	-	valid	-	-
size=2’b00,addr=2’b11	valid	-	-	-
size=2’b01,addr=2’b00	-	-	valid	valid
size=2’b01,addr=2’b10	valid	valid	-	-
size=2’b10,addr=2’b00	valid	valid	valid	valid

其读一个半字的时序逻辑如下：

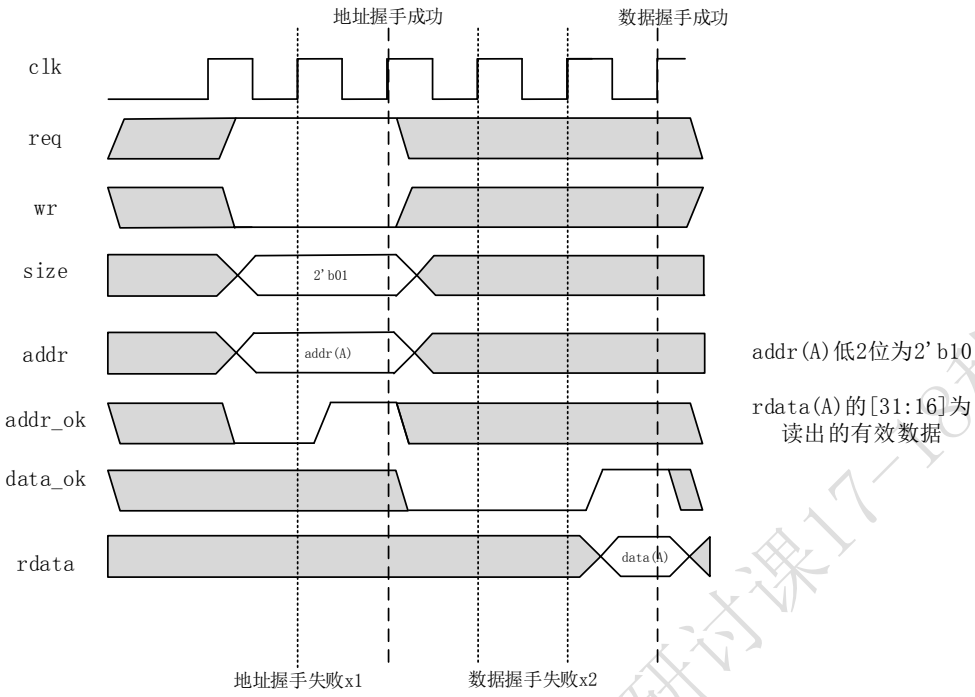


图 1-3 类 SRAM 接口一次读时序

其写一个字节的时序逻辑如下：

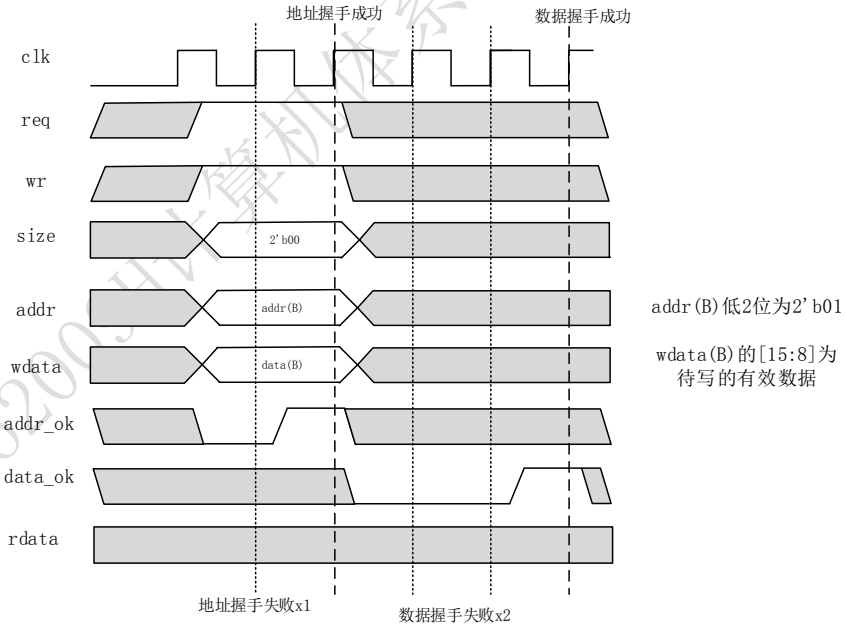


图 1-4 类 SRAM 接口一次写时序

其连续写读的的时序逻辑如下。可以看到连续写读时，slave 返回的 data_ok 应该顺序返回的，就是先写后读，必须先返回写的 data_ok，再返回读的 data_ok。另外，在一次传输 addr_ok 握手后 data_ok 握手前，是有可能出现好几次其他请求的 addr_ok 握手的，也就是可能出现握手序列 addr_ok->addr_ok->addr_ok->addr_ok->…->data_ok，master 端避免这一情况的出现可以通过拉低 req 信号，slave 避免这一情况的出现可以通过拉低 addr_ok 来避免。

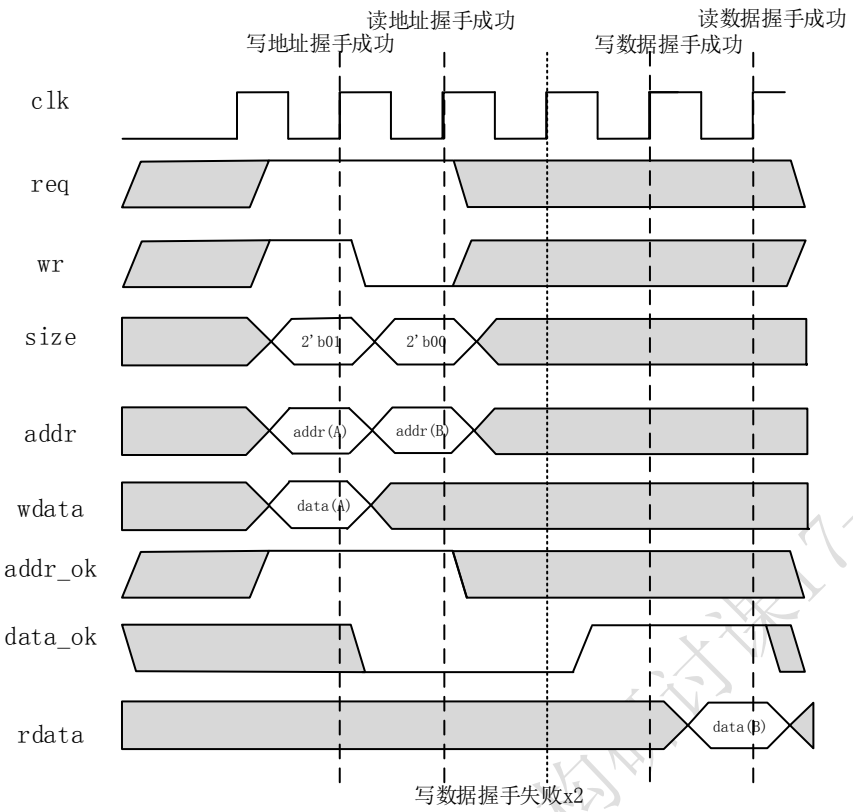


图 1-5 类 SRAM 接口连续写读时序

其连续读写的时序逻辑如下。从下图可以看到，当 addr_ok 和 data_ok 同时有效时，是针对不同请求的握手：addr_ok 是当前传输的地址握手成功，data_ok 是之前传输的数据握手成功。另外，图中读数据握手成功是有可能在写地址握手成功前完成的。

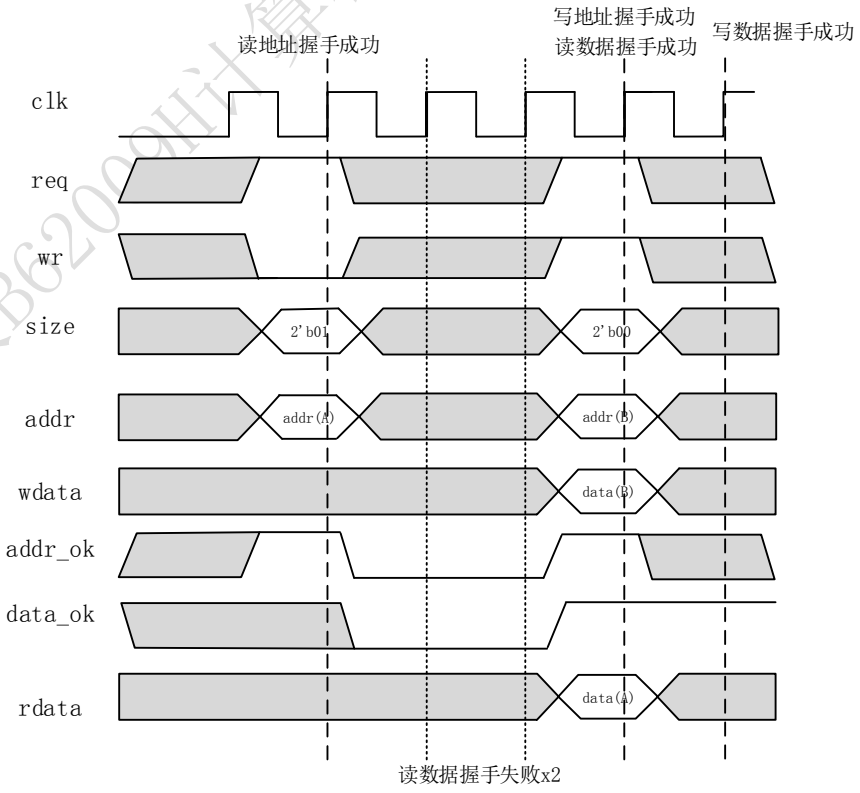


图 1-6 类 SRAM 接口连续读写时序

1.7.2 AXI 总线协议

AXI 总线协议是 AMBA 协议的一部分，面向高性能和高频率系统，它有如下特点：

a) 地址/控制信号和数据信号分离

b) 采用 VALID、READY 握手方式

在 AXI 协议中，发送读写请求的设备称为 master，另一方称为 slave。32 位 AXI 协议的接口如下表（红色为需要重点关注的）。关于 AXI 接口的时序图，请参考资料 AMBA 协议。

表 1-3 32 位 AXI 接口信号

信号	位宽	方向	功能	备注
AXI 时钟与复位信号				
aclk	1	input	AXI 时钟	
aresetn	1	input	AXI 复位，低电平有效	
读请求地址通道，（以 ar 开头）				
arid	[3:0]	master—>slave	读请求的 ID 号	固定为 0
araddr	[31:0]	master—>slave	读请求的地址	
arlen	[7:0]	master—>slave	读请求控制信号，请求传输的长度(数据传输拍数)	固定为 0
arsize	[2:0]	master—>slave	读请求控制信号，请求传输的大小(数据传输每拍的字节数)	
arburst	[1:0]	master—>slave	读请求控制信号，传输类型	固定为 2’b01
arlock	[1:0]	master—>slave	读请求控制信号，原子锁	固定为 0
arcache	[3:0]	master—>slave	读请求控制信号，CACHE 属性	固定为 0
arprot	[2:0]	master—>slave	读请求控制信号，保护属性	固定为 0
arvalid	1	master—>slave	读请求地址握手信号，读请求地址有效	
arready	1	slave—>master	读请求地址握手信号，slave 端准备好接受地址传输	
读请求数据通道，（以 r 开头）				
rid	[3:0]	slave—>master	读请求的 ID 号，同一请求的 rid 应和 arid 一致	可忽略
rdata	[31:0]	slave—>master	读请求的读回数据	
rresp	[1:0]	slave—>master	读请求控制信号，本次读请求是否成功完成	可忽略
rlast	1	slave—>master	读请求控制信号，本次读请求的最后一拍数据的指示信号	可忽略
rvalid	1	slave—>master	读请求数据握手信号，读请求数据有效	
rready	1	master—>slave	读请求数据握手信号，master 端准备好接受数据传输	
写请求地址通道，（以 aw 开头）				
awid	[3:0]	master—>slave	写请求的 ID 号	固定为 0
awaddr	[31:0]	master—>slave	写请求的地址	
awlen	[7:0]	master—>slave	写请求控制信号，请求传输的长度(数据传输拍数)	固定为 0
awsize	[2:0]	master—>slave	写请求控制信号，请求传输的大小(数据传输每拍的字节数)	
awburst	[1:0]	master—>slave	写请求控制信号，传输类型	固定为 2’b01
awlock	[1:0]	master—>slave	写请求控制信号，原子锁	固定为 0
awcache	[3:0]	master—>slave	写请求控制信号，CACHE 属性	固定为 0
awprot	[2:0]	master—>slave	写请求控制信号，保护属性	固定为 0
awvalid	1	master—>slave	写请求地址握手信号，写请求地址有效	
awready	1	slave—>master	写请求地址握手信号，slave 端准备好接受地址传输	
写请求数据通道，（以 w 开头）				
wid	[3:0]	master—>slave	写请求的 ID 号	固定为 0
wdata	[31:0]	master—>slave	写请求的写数据	
wstrb	[3:0]	master—>slave	写请求控制信号，字节选通位	
wlast	1	master—>slave	写请求控制信号，本次写请求的最后一拍数据的指示信号	固定为 1
wvalid	1	master—>slave	写请求数据握手信号，写请求数据有效	
wready	1	slave—>master	写请求数据握手信号，slave 端准备好接受数据传输	
写请求响应通道，（以 b 开头）				
bid	[3:0]	slave—>master	写请求的 ID 号，同一请求的 bid、wid 和 awid 应一致	可忽略
bresp	[1:0]	slave—>master	写请求控制信号，本次写请求是否成功完成	可忽略

bvalid	1	slave—>master	写请求响应握手信号，写请求响应有效	
bready	1	master—>slave	写请求响应握手信号，master 端准备好接受写响应	

1.7.3 第一阶段仿真与上板效果

第一阶段是针对类 SRAM 接口到 AXI 接口的 2x1 转换桥的验证。

验证环境中，有模拟 CPU（Virtual CPU）产生取指（从 inst sram-like 接口访问）和数据（从 data sram-like 接口访问）访问。该模块中设定了 5 次 inst 的读访问，设定了 5 次 data 的写访问和 5 次 data 的读访问。

(1) 仿真效果

在仿真时，控制台打印效果类似下图。也就是需要看到 10 次 OK 信号，其中 5 次是取指访问，5 次是读数据访问，打印顺序可能相互颠倒。总而言之，需要看到 10 个“OK”打印，并且最后打印“PASS”。

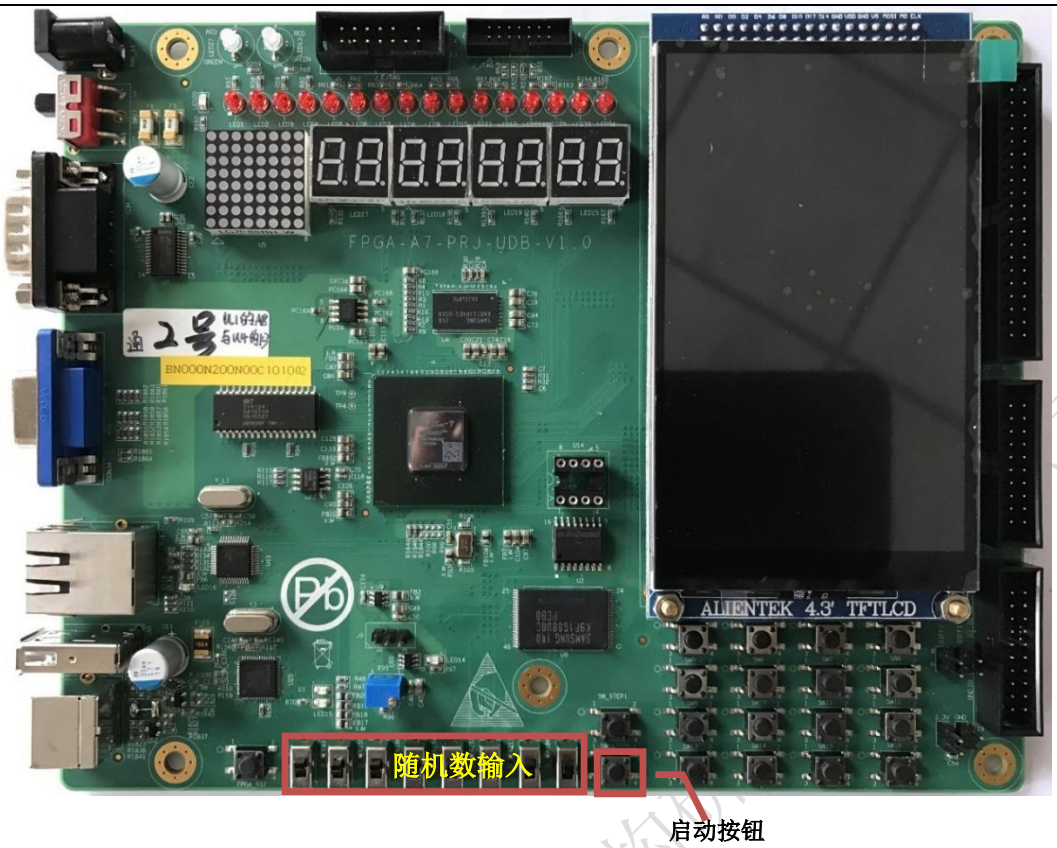
```
[   2705 ns] OK!!read data 0
[   2865 ns] OK!!!read data 1
[   2905 ns] OK!!!read data 2
[   3205 ns] OK!!!read data 3
[   3515 ns] OK!!!read data 4
[   3545 ns] OK!!!read inst 0
[   3655 ns] OK!!!read inst 1
[   3775 ns] OK!!!read inst 2
[   3935 ns] OK!!!read inst 3
[   4045 ns] OK!!!read inst 4
==============================================================
Test end!
----PASS!!!
```

如果仿真中发现错误，请进行 debug，需要观察 inst/data sram-like 接口的访问，明了该次请求的效果，然后查看 AXI 接口确认转换到 AXI 接口是否正确。

(2) 上板效果

第一阶段上板运行时，需要看到数码管如下变化 32’h1111_1111 -> 32’h2222_2222 -> 32’h3333_3333 -> 32’h4444_4444 -> 32’h5555_5555 -> 32’h6666_6666 -> 32’h7777_7777 -> 32’h8888_8888 -> 32’h9999_9999 -> 32’haaaa_aaaaa。

注意了，AXI 接口的 RAM 返回握手信号是随机返回的，其随机是使用伪随机算法产生的。我们上板时设定了一套机制：依据拨码开关输入一个随机种子，按下方脉冲开关，重新从头开始运行一次测试。操作按键如下：



既然 AXI 接口握手是随机的，那么就有可能出现这种情况：针对一个随机种子，运行测试通过了，但换另一个随机种子，运行测试就失败了。

在上板检查时，我们会按照以下流程检查：

(1) 下载 bit 文件，观察数码管是否从 32'h1111_1111 变化到 32'haaaa_aaaa；

(2) 随机拨动拨码开关，按启动按钮。再观察数码管是否从 32'h1111_1111 变化到 32'haaaa_aaaa；

(3) 重复（2）步骤多次。

所以大家应该课前尝试多组随机种子，确认代码无误。一旦发现在特定随机种子下，数码管展示异常时，就需要仿真进行 debug 了。这时候，首先需要在仿真时复现该次错误。这就需要确认出错时的随机种子，然后仿真设定随机种子为该值，则可以实现仿真复现错误。

仿真时的随机种子是在 confreg.v 的开头定义的，如下：

```
`define RANDOM_SEED {7'b1010101,16'h0000}
```

随机种子是 23bit。在上板时，拨码开关更改的是其低 16bit，该 16bit 也会展示到 LED 灯上。由于拨码开关只有 8 个，所以一个拨码开关控制 2bit。记拨码开关从左往右为 $a_{7\sim0}$，记拨码开关 a_i 为 1 表示拨上，为 0 表示拨下；级单色 led 灯从左往右为 $l_{15\sim0}$，记 led 灯 l_i 为 1 表示灯亮，为 0 表示灯灭。

那么拨动拨码开关，会看到 led 灯跟着变化，变化公式是：$l_{15\sim0}$={ a_7, a_7, a_6, a_6, a_5, a_5, a_4, a_4, a_3, a_3, a_2, a_2, a_1, a_1, a_0, a_0}，且 $l_{15\sim0}$ 对应的值即为随机种子的低 16bit。

比如，现在 8 个拨码开关的状态左侧第一个拨上，其余 7 个拨下，也就是 $a_{7\sim0}$=8'b1000_0000。那么会看到 LED 等左侧两个亮，其余 14 个灭，也就是 $l_{15\sim0}$=16'b1100_0000_0000_0000。那么该次上板时，对应的随机种子即是 RANDOM_SEED={7'b1010101, 16'b1100_0000_0000_0000}。如果该次上板，数码管行为不对，欲要仿真复现该错误，那请在 confreg.v 开头的随机种子定义处更改为 `define RANDOM_SEED {7'b1010101, 16'b1100_0000_0000_0000}，然后运行仿真即可。

1.7.4 第二阶段仿真与上板效果

第二阶段为 myCPU 添加 AXI 接口，并运行功能测试通过，功能测试程序时在 lab5_func_2 基础上的更新，将

在后续发布。另外，需要运行电子表。

所以，仿真效果同 lab5。

上板效果也类似 lab5。但第二阶段的上板，同样增加了拨码开关设定随机种子，按钮开关重新开始运行功能测试功能，具体实现方法参见上一节 1.7.3 小节。由于测试指令偏多，随机种子不同，CPU 行为必定大相径庭，所以仿真时出现错误是很普遍常见的。这时一定要掌握上一节 1.7.3 小节所述的更新仿真时随机种子的方法。