本历史

文档更新记录	文档名:	Lab10_AXI 总线接口设计（一）
	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 :	2019-10-22

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2019/10/22	贾凡	-	草稿。
2	2019/11/08	邢金璋	-	V0.1。

文档信息反馈: xingjinzhang@loongson.cn

1 实验十 AXI 总线接口设计（一）

在学习并尝试本章节前，你需要具有以下环境和能力：

(1) 装有 Vivado 的电脑一台。

(2) 熟悉 Vivado，并能初步使用。

如果对 Vivado 不熟悉，请参考课程讲义中的第一讲内容。

(3) 初步掌握 Verilog 的简单语法。

(4) 熟悉龙芯体系结构实验箱（Artix-7）。

通过本章节的学习，你将获得：

(1) 掌握 AXI 协议的知识。

(2) 学会设计类 SRAM 接口到 AXI 接口的转换转换桥。

本次实验需要参考的文档包括但不限于：

(1) Lab10 任务书（本文档）。

(2) 体系结构研讨课总讲义之第八章“AXI 总线接口设计”。（此为重点资料）

(3) 《AMBA AXI Protocol Specification v1.0》。

1.1 实验目的

1. 掌握 AXI 协议的知识。
2. 学会设计类 SRAM 接口到 AXI 接口的转换转换桥。

1.2 实验设备

1. 装有 Xilinx Vivado 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

本次实验只有一个任务（本次实验请重点参考总讲义第八章内容）：

1. 在 lab10 的实验环境中，完成：
   a) 带握手的类 SRAM 接口到 AXI 接口的转换桥 RTL 代码编写。
   b) 通过简单的读写测试。
2. 本次实验要求以组为单位提交实验报告和 RTL 代码，以报告评分和现场检查评分作为最后的实验得分：
   (1) 报告评分：描述自己的设计方案，记录调试过程（错误记录应该配截图）。实验报告模板请使用 lab3 的模板。
   (2) 现场检查评分：检查包含仿真检查和上板检查。

1.4 实验检查

检查前需提交以组为单位提交实验报告和调试好的 RTL 代码。本次实验在 2019 年 11 月 12 日或 2019 年 11 月 19 日进行检查。

现场检查，分仿真检查和上板检查：

1) 仿真检查：对照波形进行描述类 SRAM 接口与 AXI 接口转换的过程。

2) 上板检查：查看上板行为。

现场检查要求能正确应对检查者的提问，并根据要求进行正确的操作演示

1.5 实验提交

提交的作品包括纸质档和电子档。

(1) 纸质档提交

提交方式：课上现场提交，每组提交一份。

截止时间：2019 年 11 月 19 日 18:10。

提交内容：纸质档 lab10 实验报告。

(2) 电子档提交

提交方式：打包上传到 Sep 课程网站 lab10 作业下，每人都必须要有。

截止时间：2019 年 11 月 19 日 18:10。

提交内容：电子档为一压缩包，文件名是“lab10_箱子号.zip” ，目录层次如下（请将其中的“箱子号”替换为本组箱子号）。

|-lab10_箱子号/	目录，lab10 作品。
| |--lab10_箱子号.pdf/	Lab10 实验报告，实验报告模板参考“Lab03 实验报告模板_仅供参考.docx”
| |--axi_ifc /	**目录，自实现类 SRAM 接口到 AXI 接口的转换桥源码**
| |--axi_ifc.bit	**功能测试 bit 文件**

1.6 实验环境

本次实验的环境与之前的环境不相同。本次实验的环境是针对类 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。

本次实验环境的目录如下表所示：

|-cpu_axi_ifc_dev/	实验环境目录。
| |--axi_ram.coe	axi_ram 的初始化文件。
| |--rtl/	包含模拟 CPU、转换桥以及 RAM 的设计代码目录。
| | |--axi_ifc	目录，其中应当包含转换桥，需要同学们自己完成。
| | |--axi_ifc_top.v	Axi_ifc 的顶层文件。
| | |--CONFREG/	confreg 模块，用于访问 CPU 与开发板上数码管、拨码开关等外设。
| | |--ram_wrap/	以类 SRAM 接口封闭的 RAM 模块。
| | |--virtual_cpu/	模拟 CPU，用于产生取指和数据访问请求。
| | |--xilinx_ip/	封装后的 Xilinx IP，包含 axi_ram。

| |--testbench/	功能仿真验证目录。
| | |--axi_ifc_tb.v	仿真顶层。
| |--run_vivado/	Vivado 工程的运行目录。
| | |--soc_lite.xdc	Vivado 工程设计的约束文件。
| | |--cpu_axi_ifc/	创建的 Vivado 工程，名字就叫 cpu_axi_ifc。
| | | |--cpu_axi_ifc.xpr	Vivado 创建的工程文件。

本次实验的步骤是：

1) 准备好 lab10 的实验环境，cpu_axi_ifc_dev。
2) 阅读讲义和任务书，开始编写“类 SRAM 到 AXI 的转换桥”的代码，该模块名需要为“cpu_axi_interface”。
3) 将编写后的转换桥代码放到 cpu_axi_ifc_dev/rtl/axi_ifc/目录里，注意模块名需为“cpu_axi_interface”。
4) 打开 cpu_axi_ifc 工程（cpu_axi_ifc_dev/run_vivado/cpu_axi_ifc/cpu_axi_ifc.xpr），可能需要对 axi_ram 进行 upgrade IP 操作。
5) 在 Vivado 工程中通过“Add Sources”添加 cpu_axi_ifc_dev/rtl/axi_ifc/里的转换桥代码。
6) 运行 cpu_axi_ifc 工程的仿真（进入仿真界面后，直接点击 run all），开始 debug。
7) 仿真通过后，进行综合、布局布线和生成 bit 流文件，并进行上板验证。

1.7 实验说明

(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”。

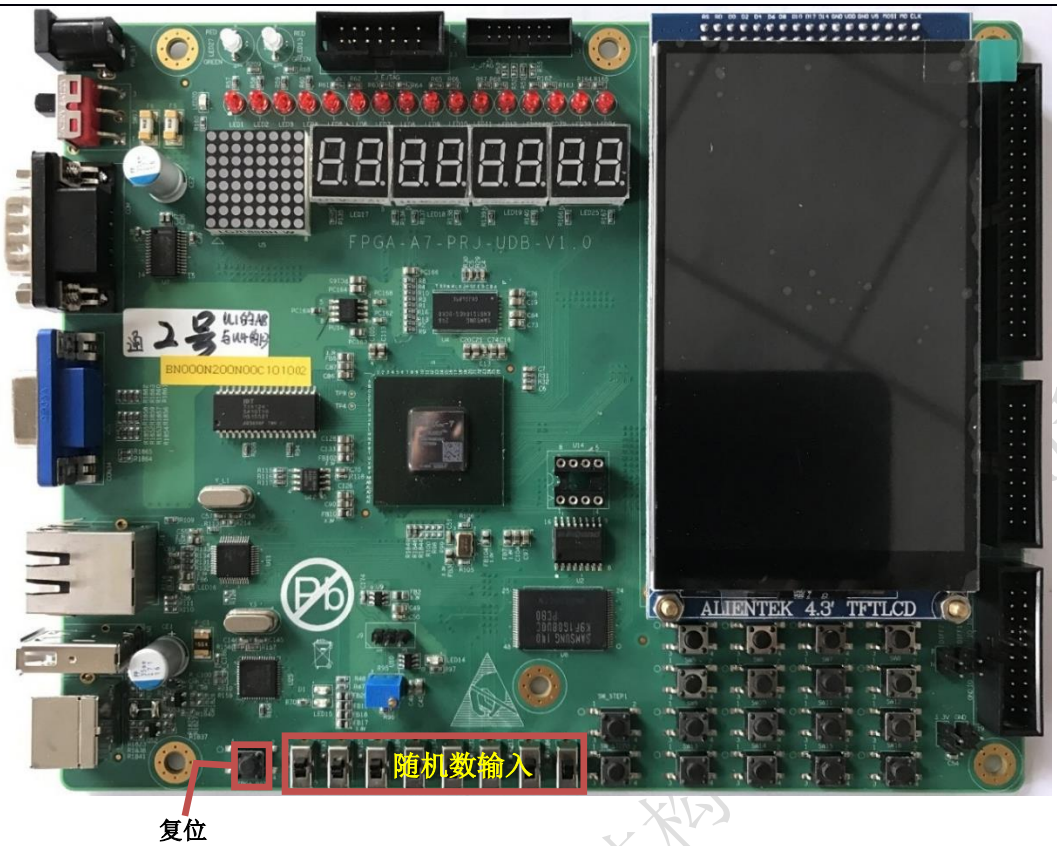
```
[   2705 ns] OK!!read data 0
[   2865 ns] OK!!!read data 1
[   2905 ns] OK!!!read data 2
[   3205 ns] OK!!!read data 3
[   3515 ns] OK!!!read data 4
[   3545 ns] OK!!!read inst 0
[   3655 ns] OK!!!read inst 1
[   3775 ns] OK!!!read inst 2
[   3935 ns] OK!!!read inst 3
[   4045 ns] OK!!!read inst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

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，按下方复位键，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。关于拨码开关控制随机种子和上板出错后的调试方法参见(3)小节。

(3) 拨码开关控制随机种子

上板时，拨码开关有两个作用，第二个作用是：**复位期间，拨码开关控制随机种子**，也就是 axi ram 访问随机延迟的初始种子。

为尽可能验证 myCPU 的功能，axi_ram 的访问具有随机延时，随机延时是通过一个 23 位的伪随机数生成：在仿真时，初始随机种子由 confreg.v 里的 RANDOM_SEED 宏定义；在上板时，初始随机种子由拨码开关控制。
实验箱上共有 8 个拨码开关，实际电平是：拨上为 0，拨下为 1；但为便于以下描述，我们记作：拨上为 1，拨下为 0。16 个 LED 单色灯，实际电平是：驱动 0 亮，驱动 1 灭；但为便于以下描述，我们记作：驱动 1 亮，驱动 0 灭。

上板时，**按下复位键**，会自动采样 8 个拨码开关的值，传为初始随机种子，且会显示初始随机种子低 16 位到单色 LED 灯上。上板时随机种子与拨码开关对应关系如下表，需要注意的时延迟类型依据拨码开关的值分为三大类：长延迟、短延迟和无延迟类型。在上板运行时都应当覆盖到这三类延迟类型。

表 1-1 上板时随机种子设定

拨码开关状态	LED 灯显示	实际初始种子 seed_init
约定，8 个拨码开关：拨上为 1，拨下为 0，记作 switch[7:0] 约定，16 个单色 LED 灯：驱动 1 亮，驱动 0 灭，记作 led[15:0]； 对应关系：led[15:0]={{2{switch[7]}}, {2{switch[6]}}, {2{switch[5]}}, {2{switch[4]}}, {2{switch[3]}}, {2{switch[2]}}, {2{switch[1]}}, {2{switch[0]}}}		

随机延迟类型分为 3 中类型： 长延迟类型：随机种子低 8 位不为 8’hff，即 seed_init[7:0]!=8’hff 短延迟类型：随机种子低 8 位为 8’hff，即 seed_init[7:0]==8’hff，（排除无延迟类型） 无延迟类型：随机种子低 16 位为 16’h00ff，即 seed_init[15:0]==16’h00ff		
8’h00	16’h0000	{7’b1010101, 16’h0000}
8’h01	16’h0003	{7’b1010101, 16’h0003}
8’h02	16’h000c	{7’b1010101, 16’h000c}
8’h03	16’h000f	{7’b1010101, 16’h000f}
…	…	…
8’hff	16’hffff	{7’b1010101, 16’hffff}