本历史

文档更新记录	文档名:	Lab11_AXI 总线接口设计（二）
	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 :	2019-10-23

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2019/10/23	贾凡	-	草稿。
2	2019/11/17	邢金璋	V0.1	初版。

文档信息反馈: xingjinzhang@loongson.cn

1 实验十一 AXI 总线接口设计（二）

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）。

(5) 掌握 CPU 流水线知识。

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 AXI 接口。

本次实验需要参考的文档包括但不限于：

(1) Lab11 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”。

(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 AXI 接口。

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：

1. 在 lab09 的实验环境的基础上，完成：
   a) 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。
   b) 集成到 SoC_AXI_Lite 系统中。
   c) 完成固定延迟和随机延迟的功能测试。
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

检查前需提交实验报告（每组一份）和调试好的 RTL 代码。本次实验在 2019 年 11 月 26 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 AXI 接口的交互过程。

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，每组只需要一人提交。

截止时间：2019 年 11 月 26 日 18:10。

提交内容：纸质档 lab11 实验报告。

(2) 电子档提交

提交方式：打包上传到 Sep 课程网站 lab11 作业下，每人都必须要有。

截止时间：2019 年 11 月 26 日 18:10。

提交内容：电子档为一压缩包，文件名是“lab11_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）。

|-lab11_箱子号/	目录，lab11 作品。
| |--lab11_箱子号.pdf/	Lab11 实验报告，实验报告模板参考“Lab03 实验报告模板_仅供参考.docx”
| **|--myCPU /**	**目录，myCPU 源码。目录请加一个 readme，简单描述下各文件。**

1.6 实验环境

本次实验的实验环境基于 ucas_CDE_axi，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。生成 golden_trace 的部分与原先相同，依然基于 SoC_Lite。测试程序对应 func_lab9，本次只有一块 RAM，只用加载 axi_ram.coe 即可。

本次实验环境中新的验证环境的组织如下表所示：

|-mycpu_axi_verify/	学生实现的 AXI 接口的 CPU 的验证开发环境。
| |--rtl/	SoC AXI Lite 设计代码目录
| | |--myCPU	基于 lab09 的 CPU 代码实现的具有 AXI 接口的 CPU
| | |--soc_axi_top.v	Soc_axi_top 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。
| | |--axi_wrap/	AXI 转接口，联接 CPU 和 crossbar。
| | |--xilinx_ip/	封装后的 Xilinx IP，包含 clk_pll，axi_ram 和 axi_crossbar_1x2。
| |--testbench/	功能仿真验证目录。

| |--mycpu_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。
| | |--mycpu_prj1/	创建的 Vivado 工程，名字就叫 mycpu。
| | | |--mycpu.xpr	Vivado 创建的工程文件。

本次实验的步骤是：

1) 准备好 lab11 的实验环境，UCAS_CDE_axi。
2) 将 lab9 发布的软件程序 func_lab9.zip，解压后，将 func_lab9/拷贝到 UCAS_CDE_axi/soft/目录里。
3) 打开 cpu132_gettrace 工程（UCAS_CDE/cpu132_gettrace/run_vivado/cpu132_gettrace/cpu132_gettrace.xpr）。
4) 对 cpu132_gettrace 工程中的 inst_ram 重新定制，此时选择加载 func_lab9 的 coe（UCAS_CDE/soft/func_lab9/obj/inst_ram.coe）。
5) 运行 cpu132_gettrace 工程的仿真（进入仿真界面后，直接点击 run all 等待仿真运行完成），生成新的参考 trace 文件 golden_trace.txt（UCAS_CDE/cpu132_gettrace/golden_trace.txt）。要等仿真运行完成，golden_trace.txt 才有完整的内容。
6) **编写 RTL 代码：将自己 lab9 完成的 myCPU 包装成 AXI 接口。**（推荐将 myCPU 的取指、数据访问先封装为类 sram 接口，将调用 lab10 完成的转换桥，一起将 myCPU 封装为 AXI 接口）
7) 将新完成的 myCPU 代码放在 UCAS_CDE/mycpu_axi_verify/rtl/myCPU/目录里。
8) 打开 myCPU 工程（UCAS_CDE/mycpu_axi_verify/run_vivado/mycpu_prj1/mycpu_prj1.xpr）。
9) 通过“Add Sources”将新的 myCPU 源码添加到工程中。
10) 对 myCPU 工程中的 axi_ram 重新定制，此时选择加载 func_lab9 的 coe（UCAS_CDE/soft/func_lab9/obj/inst_ram.coe）。
11) 运行 myCPU 工程的仿真（进入仿真界面后，直接点击 run all），开始 debug。
12) 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

1.7.1 仿真与上板效果

上板时要求“随意切换拨码开关按复位”，CPU 均通过 94 个功能点的测试。

“随意切换拨码开关按复位”是为了设定随机种子，重新开始运行功能测试功能。由于测试指令偏多，随机种子不同，CPU 行为轨迹大不相同，所以仿真时出现错误是很常见的。

请注意上板的具体操作：将 8 个拨码开关切换为随意状态，按复位键；松开复位键后，数码管开始累加，此时可以切换开关来控制 wait_1s 的累加速度。

1.7.2 拨码开关控制 wait_1s

上板时，拨码开关有两个作用，第一个作用是：**复位后**，拨码开关控制 wait_1s 的循环次数，也就是控制数码管累加的速度。

94 个功能点测试中，每两个功能点之间会穿插一个 wait_1s 函数，wait_1s 通过一段循环完成计时的功能：在上板时，wait_1s 循环次数由拨码开关控制，可设置循环次数为（0~0xaaaa）*2^9。请在复位后，通过拨码开关选择合理的 wait_1s 延时。

1.7.3 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： (1) 长延迟类型：随机种子低 8 位不为 8’hff，即 seed_init[7:0]!=8’hff (2) 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） (3) 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}

1.7.4 仿真通过、上板出错的调试方法

如果上板发现一个功能点都不显示，可能是以下问题之一导致的：

(1) 时序违约；

(2) 仿真时控制信号有“X”。仿真时，有“X”调“X”，有“Z”调“Z”。AXI 接口的 Valid 和 ready 信号千万不能出现“X”或“Z”。

(3) 模块里的控制路径上的信号未进行复位。

(4) 多驱动。

(5) 代码不规范，阻塞赋值乱用，always 语句随意使用。

(6) 时钟复位信号接错。

(7) 模块的 input/output 端口接入的信号方向不对。

如果上板时发现在某一些随机种子下测试通过，在另一些随机种子情况下出错，请按以下步骤进行调试：

(1) 确认出错的随机种子，修改 ucas_CDE_axi/rtl/CONFREG/confreg.v 里的 RANDOM_SEED 的定义，改为出错时的随机种子，随后进行仿真：如果有错，则调试；如果发现仿真没错，则在上板时找寻下一个出错的随机种子，同样设定好 RANDOM_SEED 后进行仿真，如果都没错则转(2)。

(2) 当碰到，相同环境下仿真无法复现上板的错误时，请都转到本步骤：反思设计；也可以使用 Vivado 的逻辑分析仪进行在线调试。

如果上板发现任意随机种子下，都只有部分功能点测试通过。则可能以上两种原因都有，请依次排查。如果排查后都无法解决问题，可以使用 Vivado 的逻辑分析仪进行在线调试。