**团队编号： A**

**2017年西安电子科技大学网络与信息安全学院**

**FPGA实验报告**

**团队名称： 霸王集团**

**队长\副队长： 陈启源**

**团队成员： 李艺扬，刘晋，朱晓宇**

**团队成员： 王莹，李丽青，刘靖山**

**团队成员： 文浩斌， 费伊扬，罗泽宇**

**Github账号： https://github.com/skyel1u/fpga**

**队长联系电话： 18392987838**

**队长电子邮箱： nem9ma0@outlook.com**

**指导教师： 傅延增**

**2017年 12 月 8 日**

**网络与信息安全学院**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **基本信息表** | | | | |
| **实验内容：RISC ALU代码补全计划** | | | | |
| **实验摘要：**  **基于老师所给alu.vhd与risc\_i\_16pack.vhd文件中的代码，补全alu.vhd中所需的变量声明，函数等。其中老师已经给出加法器的代码，所以我们需要补全基于给出的除加法器外的其他逻辑函数，并最后对该ALU进行测试。** | | | | |
| **团队任务分工：** | | | | |
|  | **姓名** | **学号** | **分工** | **签字** |
|  | **陈启源** | **16069110001** | **设计** |  |
|  | **刘晋** | **15180110082** | **设计** |  |
|  | **朱晓宇** | **15180110011** | **设计** |  |
|  | **李丽青** | **16180120015** | **设计** |  |
|  | **李艺扬** | **16180110103** | **设计** |  |
|  | **王莹** | **16180120003** | **测试** |  |
|  | **刘靖山** | **16010410024** | **测试** |  |
|  | **文浩斌** | **16180120041** | **测试** |  |
|  | **罗泽宇** | **16180120025** | **文案** |  |
|  | **费伊扬** | **16180120024** | **文案** |  |

**一、RISC ALU补全计划：**

|  |
| --- |
| **题目描述：**  **基于老师所给alu.vhd与risc\_i\_16pack.vhd文件中的代码，补全alu.vhd中所需的变量声明，函数等。其中老师已经给出加法器的代码，所以我们需要补全基于给出的除加法器外的其他逻辑函数，并最后对该ALU进行测试。** |
| **背景与原理：**  **题目背景：基于已给代码补全ALU中其他功能。**  **理论原理：了解并掌握Verilog语言的语法及运行方式。了解16位RISC运算器（ALU）的运作原理，明白需要补全的指令集的功能。了解测试需要的技巧。** |
| **步骤：**   1. **先将指令集需要的函数划分出来，我们组将24个指令集所实现的功能划分为了8个函数（算上原来所给的adder函数），分别为adder（基础运算），ander（且），orer（或），xorer（异或），noter（非），mover（赋值），slcer（循环左移），srcer（循环右移）。在函数中声明并初始化它们的变量。**   **代码如下：**  **architecture rtl of alu is**  **-- helper signals for adder**  **signal add\_b : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal add\_cin : std\_ulogic;**  **signal add\_res : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal add\_cout : std\_ulogic;**  **-- helper signal for ander**  **signal and\_res : std\_ulogic\_vector(bit\_width\_g-1 downto 0);**  **--helper signals for orer**  **signal or\_res : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal or\_zout : std\_ulogic;**    **-- helper signals for xorer**  **signal xor\_res : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal xor\_zout : std\_ulogic;**  **-- helper signals for noter**  **signal not\_res : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal not\_zout : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **-- helper signals for mover**  **signal move\_b : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal move\_res : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **-- helper signals for slcer**  **signal slc\_res : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal slc\_count : std\_ulogic;**  **-- helper signals for srcer**  **signal src\_res : std\_ulogic\_vector(bit\_width\_g - 1 downto 0);**  **signal src\_count : std\_ulogic;**   1. **再根据划分的函数，将它们的功能逐一实现** 2. **. 基本上orer, xor, ander, noter运算可以由Verilog中自带的运算符来实现**   **-- xor**  **xorer\_inst: process (side\_a\_i, side\_b\_i)**  **begin**  **xor\_res <= std\_ulogic\_vector(unsigned(side\_a\_i) XOR unsigned(side\_b\_i));**  **end process xorer\_inst;**  **--not**  **noter\_inst:process(side\_a\_i)**  **begin**  **not\_res <= std\_ulogic\_vector(NOT(unsigned(side\_a\_i)));**  **end process noter\_inst;**  **-- or**  **or\_inst: process (side\_a\_i, side\_b\_i)**  **begin -- process or\_inst**  **add\_res <= std\_ulogic\_vector( unsigned(side\_a\_i) OR unsigned(side\_b\_i) );**  **end process or\_inst;**  **-- and**  **and\_inst: process(side\_a\_i,side\_b\_i)**  **begin**  **and\_res <= std\_ulogic\_vector(unsigned(side\_a\_i) and unsigned(side\_b\_i));**  **end process and\_inst;**   1. **mover则是需要判断传过来的是从a端还是b端传入的值之后再将判断后的值赋值之后输出：**   **-- move**  **move\_inst: process (side\_a\_i,side\_b\_i)**  **begin**  **move\_res <= std\_ulogic\_vector(unsigned(move\_b));**  **end process move\_inst;**  **with alu\_func\_i select**  **move\_b <=**  **side\_a\_i when alu\_pass\_a\_c,**  **side\_b\_i when alu\_pass\_b\_c;**   1. **循环左移/循环右移则是需要获取最左端/右端的数并将其移到该值的最右端/左端。所以在这里我们使用位连接符号 & 来完成函数。**   **-- slc**  **slc\_inst: process(side\_a\_i)**  **begin**  **slc\_count <= side\_a\_i(side\_a\_i'high);**  **slc\_res <= side\_a\_i(side\_a\_i'high - 1 downto 0) & slc\_count;**  **end process slc\_inst;**  **-- src**  **src\_inst: process(side\_a\_i)**  **begin**  **src\_count <= side\_a\_i(0);**  **src\_res <= src\_count & side\_a\_i(side\_a\_i'high downto 1);**  **end process src\_inst;**   1. **最后用一个case选择来根据传入指令输出函数命令：**   **process(alu\_func\_i)**  **begin**  **CASE alu\_func\_i IS**  **--when alu\_add\_c => result\_o <= add\_res;**  **--when alu\_inc\_c => result\_o <= add\_res;**  **--when alu\_sub\_c => result\_o <= add\_res;**  **--when alu\_dec\_c => result\_o <= add\_res;**  **when alu\_pass\_a\_c => result\_o <= move\_res;**  **when alu\_pass\_b\_c => result\_o <= move\_res;**  **when alu\_and\_c => result\_o <= and\_res;**  **when alu\_or\_c => result\_o <= or\_res;**  **when alu\_xor\_c => result\_o <= xor\_res;**  **when alu\_not\_c => result\_o <= not\_res;**  **when alu\_slc\_c => result\_o <= slc\_res;**  **when alu\_src\_c => result\_o <= src\_res;**  **when others => NULL;**  **END CASE;**  **end process;**  **以上代码均是由测试小组的排查测试之后，解决了bug的最终代码。** |
| **讨论与结论**  **1.设计ALU时我们应该把指令集按照功能划分为不同的模块，然后再对每个模块分别进行设计和实现。这样不仅有利于功能的例化和复用，也方便了后期的测试与调试。**  **2.很多基本的运算功能Verilog已有相应的运算符，我们只需要合理的使用它们即可。** |
| **完成心得与其它**  **通过这次的RISC ALU的设计，我们收获很大。了解到设计一个ALU是比较麻烦的，里面可能包含有很多的函数，如果一个函数没处理好，就会导致设计的失败。本次的RISC ALU的设计，我们深度了解到一个ALU的工作原理和FPGA开发的流程。这对于以后我们学习FPGA必定有很大的帮助。** |
| **参考文献**   * **[16位RISC微处理器在FPGA上的设计与实现](http://xueshu.baidu.com/s?wd=paperuri%3A%2882f9639df309bb80f6fe5c9c6e0fb69c%29&filter=sc_long_sign&tn=SE_xueshusource_2kduw22v&sc_vurl=http%3A%2F%2Fd.wanfangdata.com.cn%2FThesis%2FD561086&ie=utf-8&sc_us=16903667839675474657" \t "_blank)** * **[16位RISC处理器的设计和FPGA实现](http://xueshu.baidu.com/s?wd=paperuri%3A%28fa548406c0c9aaf2c52f7714ceacc895%29&filter=sc_long_sign&tn=SE_xueshusource_2kduw22v&sc_vurl=http%3A%2F%2Fcdmd.cnki.com.cn%2FArticle%2FCDMD-10730-1015352544.htm&ie=utf-8&sc_us=7452721847219559344" \t "_blank)** |