微处理器与设计实验说明

2022年10月

一、实验环境

1. 虚拟机VMware

   > 安装VMware 15.5及以上版本虚拟机，建议安装提供的16.2版本，许可证问题请自行解决

2. 操作系统Ubuntu

   > 使用提供的Ubuntu 18.04操作系统，相关工具已在其中安装；
   > 用户名： `mirco` ，密码： `123456` ；
   > 样例工程位置： `/home/micro/Documents/examples` 。

3. 交叉编译工具链RISC-V GNU Compiler Toolchain

   > RISC-V和C++的交叉编译器，已正确安装；
   > 支持 `Newlib` 和 `glibc` 工具链，支持 `32` 位和 `64` 位交叉编译；
   > `Newlib` 交叉编译器相关命令为： `riscv64-unknown-elf-*` ，主要面向裸机程序；
   > `Glibc` 交叉编译器相关命令为： `riscv64-unknown-linux-gnu-*` ，支持Linux标准库；
   > 工具链源码位置： `/home/micro/tools` ，其下载、编译和安装可参考RISC-V GNU Compiler Toolchain。

4. 指令模拟器riscv-isa-sim： `Spike`

   > `Spike` 实现了一个或多个hart的功能模型，已正确安装；
   > 支持 `RV32GC` 、 `Debug` 等几乎所有架构相关的功能；
   > `Spike` 一般在代理内核 `pk` 上执行，具体可参考riscv-isa-sim。

5. 代理内核riscv-pk： `pk`

   > `pk` 是RISC-V Proxy Kernel和Boot Loader，已正确安装；
   > 支持静态链接的RISC-V ELF二进制文件的轻量级应用执行环境，借助 `spike` 运行；
   > 其核心是支持了代码执行所需要的一些IO操作，具体可参考riscv-pk。

6. 硬件仿真器 `Verilator`

   > `Verilator` 是开源的 `Verilog` / `System Verilog` 硬件仿真器，已正确安装；
   > `Verilator` 将 `Verilog` / `System Verilog` 代码翻译成可编译的 `C++` / `SystemC` 代码，之后再编写合适的驱动代码，一同使用 `C++` 编译器进行编译，由此得到可执行的程序，来对 `Verilog` / `System Verilog` 代码进行仿真；

> `Verilator` 安装使用可参考Verilator User's Guide；
> 也可根据个人习惯选用其他工具。

7. 波形查看器 `GTKWave`
> `GTKWave` 是开源的波形查看器，已正确安装；
> `GTKWave` 安装使用可参考gtkwave;
> 也可根据个人习惯选用其他工具。

8. 其他工具
> 其他工具根据需要自行解决。

二、实验示例

1. CPU选择
> 主要采用2021年 ***Jiadong Wu*** 同学的作业，该项目可参考AdamRiscv；
> 该CPU实现了 `M` 态，指令集为 `RV32I` ，五级流水线单发射顺序执行，哈佛架构；
> 通过停顿和数据旁路解决数据冲突（ `load` 指令后数据冲突需停顿1拍，其他指令数据相关不需停顿），通过清空流水线解决控制冲突（跳转指令2拍空泡）。

2. 样例 `RVDesign` 基本情况
> 样例位置： `/home/Documents/examples/RVDesign` ；
> 样例包含文件夹和文件：

- `AdamRiscv` 文件夹——存放CPU的完整RTL代码和设计文档，指令 `HEX` 文件在 `/home/Documents/examples/RVDesign/AdamRiscv/rom` 中；
- `logs` 文件夹——程序运行生成的log文件夹，波形图在里面；
- `obj_dir` 文件夹——程序运行生成的编译文件夹，RTL代码编译后的文件都在里面；
- `test_program` 文件夹——测试程序文件夹，包含从汇编和C程序生成Verilog中RAM使用的 `HEX` 格式文件；
- `adam_riscv_main.cpp` 文件——verilator封装文件，作用为将verilator编译后的RTL代码封装到该文件中运行，给予RTL顶层文件激励等；
- `Makefile` 和 `Makefile_obj` 文件——编译和运行文件的脚本，使用 `make` 命令执行。

> 样例使用：

- 在 `RVDesign` 文件夹中打开终端，输入 `make` 命令回车执行，程序运行完成后打开GTKWave，即可查看波形图。

3. 样例 `RVDesign` 重要说明
> RTL代码：

- 重新编写或添加的RTL代码建议放在原文件夹 `/home/Documents/examples/RVDesign/AdamRiscv` ；

- 更改RTL代码后需要重新编译，可以使用 make 命令运行 Makefile ，需要注意的是，如果有增删RTL文件，相应地应在 Makefile 文件中更改；
- CPU的指令存储器初始化文件存放在 /home/Documents/examples/RVDesign/AdamRiscv/rom 文件夹，命名为 test_program.hex ，更改指令文件可直接替换该文件，需要注意的是，程序大小应适应指令存储器大小；
- test_program.hex 文件可由 test_program 文件夹下的程序产生。

> adam_riscv_main.cpp 封装文件：

- Verilator编译过程会将RTL代码编译成 C++ 或 SystemC 文件，输出 .cpp 和 .h 文件（在 obj_dir 文件夹中），之后再编写合适的驱动代码（也就是 adam_riscv_main.cpp ），最后一起使用 C++ 编译器进行编译，得到可执行程序，来对原RTL代码进行仿真，得到波形图（也就是 logs 文件夹中的 wave.vcd 文件）等仿真结果；
- 封装文件的编写主要注意：
  - 包含顶层 .h 文件，如RTL代码的顶层模块为 adam_riscv ，则编译后会生成 Vadam_riscv 类，需要将其头文件 Vadam_riscv.h 包含进来；
  - 包含 verilated.h 文件，这是verilator里面最常用的类；
  - 包含 verilated_vcd_c.h 文件，tracing波形图必备的类；
  - 在每个仿真时间 main_time 生成时钟 clk 和置位 rst 信号值；
  - 给顶层模块输入赋值后一定要调用 eval() 函数，得到运行结果，即各信号的稳定输出；
  - 波形每个仿真时间 main_time 记录，调用 VerilatedVcdC 类的 dump 函数，否则无法查看波形图；
  - adam_riscv_main.cpp 封装文件使用仿真时间 main_time 作为仿真结束条件，可根据自己需求更改；
  - 更多细节请参考Verilator User's Guide和 /usr/share/doc/verilator/examples 文件夹中的示例。

> Makefile 和 Makefile_obj 编译文件

- Makefile 和 Makefile_obj 文件将编译等命令集中到一个文件中，执行时只需 make 一下即可，提供编译效率；
- 编译多个文件时，需要包括所有文件，并使用参数 --top-module 顶层模块名 并指定顶层模块；
- 其他参数请查看 Makefile 文件注释或参考Verilator User's Guide。

> test_program.hex 指令生成

- test_program 文件夹有两种生成 test_program.hex 文件的示例， assembleProgram2hex 文件夹是从汇编语言生成 test_program.hex 文件， CProgram2hex 文件夹是从C语言生成 test_program.hex 文件；
- assembleProgram2hex 文件夹包括 S2hex.sh 和 test.s 文件， S2hex.sh 文件是编译命令shell脚本， test.s 文件是汇编语言文件，使用 riscv 汇编语言编写，为使用 RV32I 指令集计算两个无

符号数的乘法结果（***Jiadong Wu*** 同学编写），运行 `S2hex.sh` 文件即在 `/AdamRiscv/rom/` 文件夹中生成 `test_program.hex` 文件；

- `CProgram2hex` 文件夹包括 `C2hex.sh` 、 `init.s` 、 `link.ld` 和 `main.c` 文件， `C2hex.sh` 文件是编译命令shell脚本， `init.s` 是为初始化 `sp` 和 `s0` 寄存器而特意编写的汇编文件， `link.ld` 文件指定了输出文件架构、程序入口和指令地址等， `main.c` 文件是用户编写的C程序（使用riscv基本指令集），运行 `C2hex.sh` 文件即在 `/AdamRiscv/rom/` 文件夹中生成 `test_program.hex` 文件；

4. 其他样例 `test_asm` 和 `test_verilator`

> `test_asm` 文件夹是 `riscv` 程序交叉编译和运行的示例，包括 `asm_add` 和 `c-hello` 示例：

- `asm_add` 示例是使用 `riscv` 汇编语言编写程序并在 `spike + pk` 上运行，可用来检验所编写的CPU运行结果是否正确；
- `c-hello` 示例是使用C语言编写程序并在 `spike + pk` 上运行，可用来检验所编写的CPU运行结果是否正确。

> `test_verilator` 文件夹是RTL代码使用verilator编译仿真的示例，包括 `cpp-test` 和 `tracing_c` 示例：

- `cpp-test` 是一个简单的verilator使用示例；
- `tracing_c` 是一个复杂一些的verilator使用示例；
- 更多示例可参考 `/usr/share/doc/verilator/examples` 文件夹和verilator examples。