B62007Y 2017-2018学年春季学期

计算机组成原理实验

处理器中的基本功能部件

主讲教师： 张 科

2018年3月23日

中国科学院大学
University of Chinese Academy of Sciences

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY,CAS

2017 ACM A.M. Turing Award

Association for Computing Machinery

Advancing Computing as a Science & Profession

Digital Library | CACM | Queue | TechNews | Lear

Join | Volunt

ABOUT ACM | MEMBERSHIP | PUBLICATIONS | SPECIAL INTEREST GROUPS | CONFERENCES | CHAPTERS | AWARDS | EDUCATION | PUBLIC POLICY

Home > Media Center > ACM A.M. Turing Award 2017

Pioneers of Modern Computer Architecture Receive ACM A.M. Turing Award

Hennessy and Patterson's Foundational Contributions to Today's Microprocessors Helped Usher in Mobile and IoT Revolutions

NEW YORK, NY, March 21, 2018 – ACM, the Association for Computing Machinery, today named John L. Hennessy, former President of Stanford University, and David A. Patterson, retired Professor of the University of California, Berkeley, recipients of the 2017 ACM A.M. Turing Award for pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry. Hennessy and Patterson created a systematic and quantitative approach to designing faster, lower power, and reduced instruction set computer (RISC) microprocessors. Their approach led to lasting and repeatable principles that generations of architects have used for many projects in academia and industry. Today, 99% of the more than 16 billion microprocessors produced annually are RISC processors, and are found in nearly all smartphones, tablets, and the billions of embedded devices that comprise the Internet of Things (IoT).

Hennessy and Patterson codified their insights in a very influential book, *Computer Architecture: A Quantitative Approach*, now in its sixth edition, reaching generations of engineers and scientists who have adopted and further developed their ideas. Their work underpins our ability to model and analyze the architectures of new processors, greatly accelerating advances in microprocessor design.

The ACM Turing Award, often referred to as the "Nobel Prize of Computing," carries a $1 million prize, with financial support provided by Google, Inc. It is named for Alan M. Turing, the British mathematician who articulated the mathematical foundation and limits of computing. Hennessy and Patterson will formally receive the 2017 ACM A.M. Turing Award at the ACM's annual awards banquet on Saturday, June 23, 2018 in San Francisco, California.

Contact:
Jim Ormond
212-626-0505
ormond@hq.acm.org

John L. Hennessy

David A. Patterson

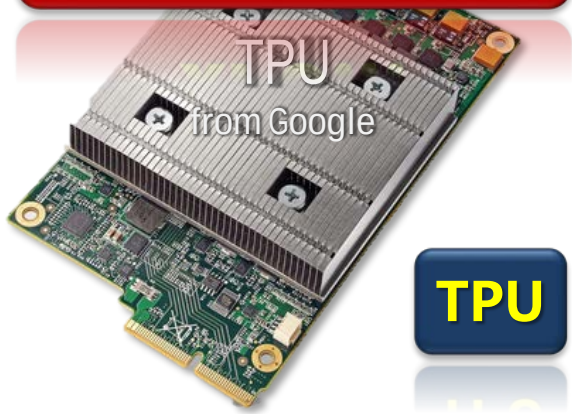
处理器

CPU

GPU

xPU...

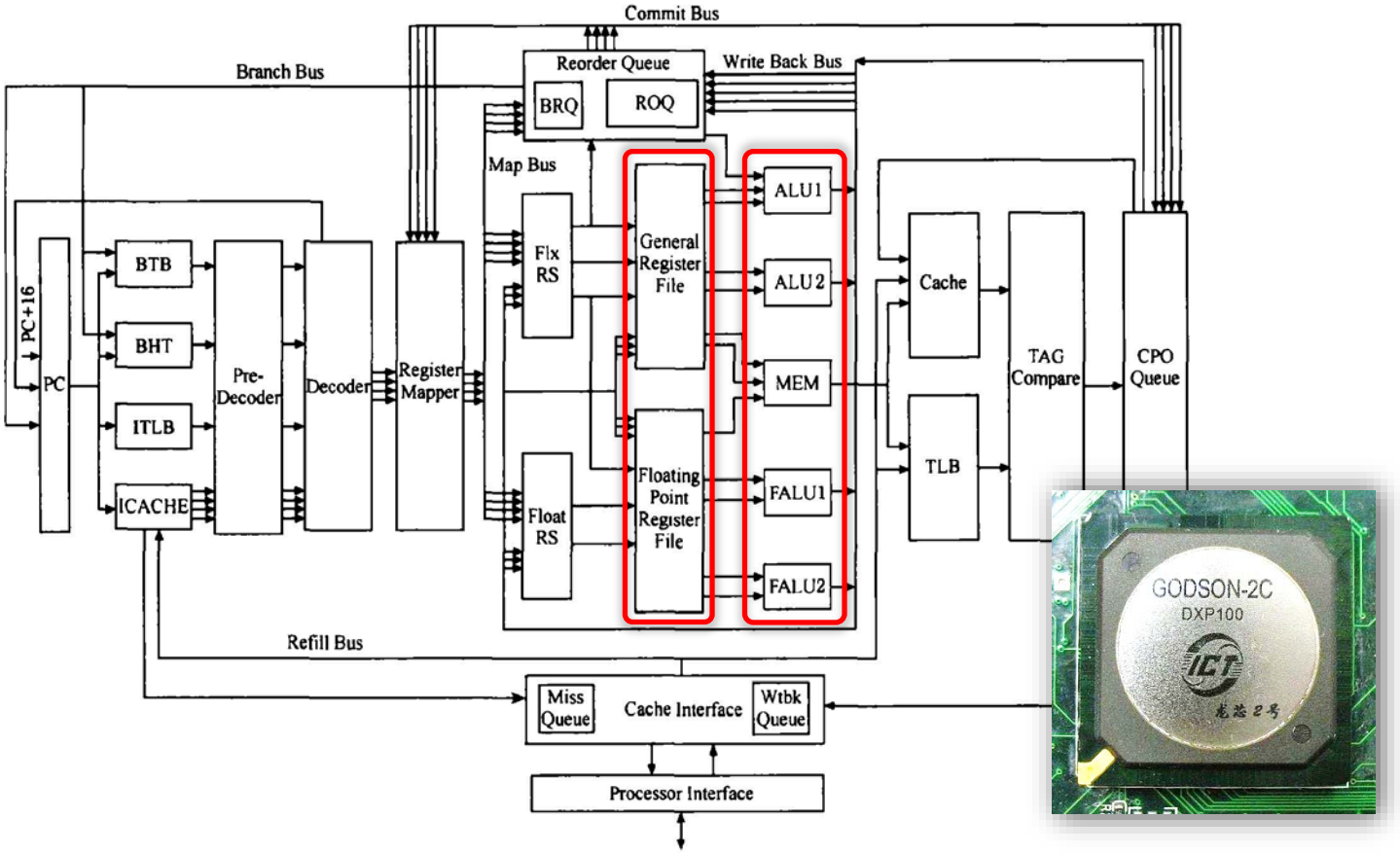
VPU



TPU

CPU(及其他众多xPU)的基本功能部件

- CPU中两个重要功能部件: **Register File** & **ALU**



龙芯2号（64位类MIPS指令集的通用RISC处理器）体系结构

CPU(及其他众多xPU)的基本功能部件

- CPU中两个重要功能部件: **Register File** & **ALU**

- CPU指令集架构（ISA）定义一组体系结构概念上的寄存器（architectural registers），用于在CPU运算部件与内存之间暂存数据

- 在简单的CPU中，这些架构寄存器与CPU芯片上物理存在的寄存器单元一一对应，一组物理寄存器（physical registers）构成一个寄存器堆（或称寄存器组，register file）

组合电路 vs. 时序电路

- **寄存器堆由多个寄存器和组合逻辑电路构成**
  - ALU是纯粹的组合逻辑电路
- **组合逻辑电路 Combinational logic**
  - 从电路本质上讲，组合逻辑电路的特点是任一时刻的输出信号只是当前时刻输入信号的函数，与其他时刻的输入状态（电路原来的状态）无关
  - 从电路行为上看，其特征就是输出信号的变化仅仅与输入信号的电平有关，不涉及对信号跳变沿的处理
  - Once the input arrives, the combinational logic starts processing it to make outputs
- **时序逻辑电路 Sequential logic**
  - 从电路特征上看来，其特点为任意时刻的输出不仅取决于该时刻的输入，而且还和电路原来的状态（或者说，之前的输入）有关
  - 从电路行为上讲，不管输入如何变化，仅当时钟的沿（上升沿或下降沿）到达时，才有可能使输出发生变化
  - Once the input arrives, you have to wait until the next positive or negative clock edge, before the input can be read in and processed

B62007Y 2017-2018学年春季学期

计算机组成原理实验

实验项目1 基本功能部件设计

——Register File & ALU

常轶松 赵然 王诲喆

2018年3月23日

中国科学院大学 University of Chinese Academy of Sciences
中科院计算所 INSTITUTE OF COMPUTING TECHNOLOGY,CAS

实验目的

- **设计MIPS处理器中使用的32-bit通用寄存器堆（Register File）和算术逻辑单元（ALU）**
  - 设计包含32个32-bit通用寄存器的寄存器堆
    - 支持2读1写端口
    - 理解“同步写、异步读”的基本原理
  - 设计基本算术逻辑单元ALU（下周讲解与布置）
  - 掌握实验项目的阶段提交方法
  - 为后续完整MIPS CPU设计实验准备好可用组件

实验项目发布与接收流程（1）

- 实验项目1参与邀请链接URL
  - **https://classroom.github.com/a/SFSWa1ba**
  - 请同学们在浏览器中输入上述URL
    - 如之前未使用该浏览器登录过GitHub，会提示输入GitHub用户名及密码登录
    - 点击绿色<Accept this assignment>按钮接受本次作业安排（如左图标出）
    - 在GitHub注册邮箱中会收到邮件，点击accept即可加入到本次作业中（如右图标出）

实验项目发布与接收流程（2）

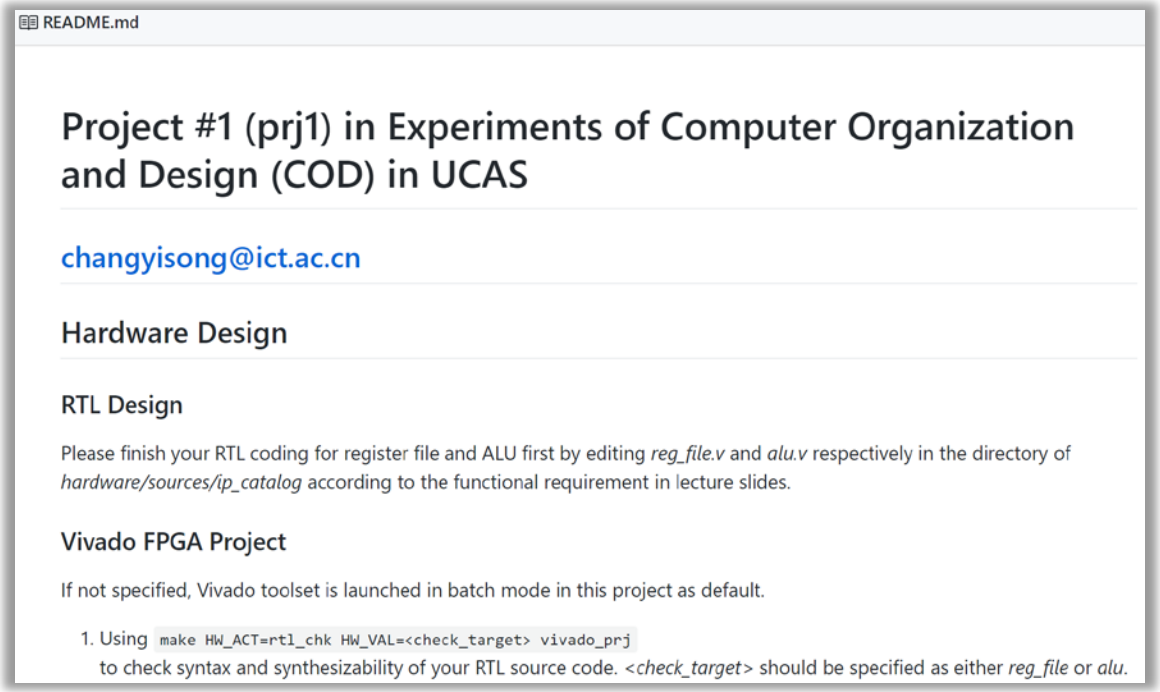
- **个人作业远程仓库**
  - prj1-YOUR_GitHub_USERNAME
    - 例如：某学生GitHub用户名为zhang_abc，其实验项目1的repo名即为prj1-zhang_abc
  - 远程仓库的URL地址https://github.com/ucas-cod-18sp/<repo名称>
    - 如上例： https://github.com/ucas-cod-18sp/prj1-zhang_abc
- **同学们可在浏览器中输入URL地址来访问自己的实验项目远程仓库**
- **请同学们在虚拟机shell终端中使用git clone命令克隆一个本地仓库**
  - 如上例：git clone https://github.com/ucas-cod-18sp/prj1-zhang_abc
  - 注意：虚拟机此时需要连通外网
  - 交互式输入GitHub用户名和密码
  - 如上例，命令完成后会在当前执行命令的路径下创建一个名为prj1-zhang_abc的目录，该目录即为本次实验项目的本地仓库
  - 实验代码编写、硬件工程创建及生成bitstream配置文件、使用远程和本地FPGA板卡等操作均在本地仓库中完成

实验框架更新同步

- **课程发布实验框架时会尽力保证无误，但在同学们使用过程中还是可能会发现一些新问题或者使用不便之处，因此我们将及时或适时对框架中的代码脚本进行更新**
  - 例如，同学们使用2018年3月23日发布的实验框架，在执行rtl_chk，sch_gen，bit_gen和bhv_sim时，Vivado会出现找不到top module的现象，导致脚本输出类似如下的错误：
    - 在rlt_chk和sch_gen阶段出现module ‘alu’ not found
    - 在bit_gen阶段出现module ‘mpsoc_wrapper’ not found
    - 因找不到reg_file_test.v顶层，在bhv_sim阶段出现Cannot find design unit xil_defaultlib.reg_file_testin library work located at xsim.dir/work.
  - 3月24日下午，已在prj1-student原始框架仓库（https://github.com/ucas-cod/prj1-student）中对脚本进行了更新，并调整了实验操作流程
- **框架发布更新后，需要同学们在自己的实验项目1本地仓库中进行同步**
  - 第一步，添加远程地址
    - **git remote add upstream https://github.com/ucas-cod/prj1-student**
  - 第二步，使用git status查看本地仓库没有“已修改但还未commit”的文件
    - 如存在未提交文件，需要**先完成对本地修改的commit**
  - 第三步，框架同步
    - **git pull upstream master**

更新后请阅读README.md文件

- 同学们在完成框架同步后，可在本地仓库中查看更新后的README.md帮助说明文件，或直接在网络浏览器中打开prj1-student远程仓库查看帮助说明文件
  - https://github.com/ucas-cod/prj1-student



README.md

Project #1 (prj1) in Experiments of Computer Organization and Design (COD) in UCAS

changyisong@ict.ac.cn

Hardware Design

RTL Design

Please finish your RTL coding for register file and ALU first by editing *reg_file.v* and *alu.v* respectively in the directory of *hardware/sources/ip_catalog* according to the functional requirement in lecture slides.

Vivado FPGA Project

If not specified, Vivado toolset is launched in batch mode in this project as default.

1. Using `make HW_ACT=rtl_chk HW_VAL=<check_target> vivado_prj`
   to check syntax and synthesizability of your RTL source code. *<check_target>* should be specified as either *reg_file* or *alu*.

内容大纲

❑通用寄存器堆（Register File）功能部件设计

- 通用寄存器堆基本功能定义及设计要点讲解
- 通用寄存器堆实验操作流程及提交要求

通用寄存器堆模块定义

❑ 支持32个32-bit寄存器堆输入输出端口定义

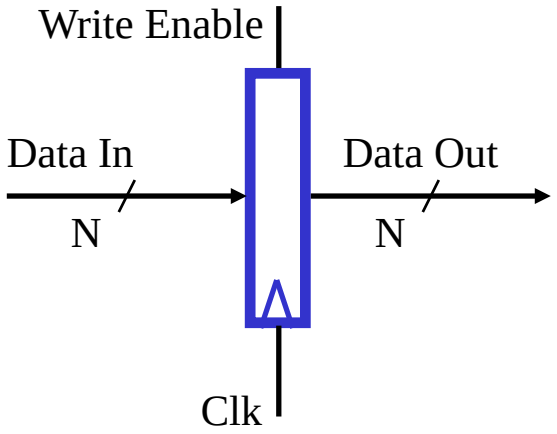
- 2读1写端口

信号名	I/O	说明
clk	Input	时钟
rst	Input	高电平同步复位（仅对0号寄存器有效）
waddr[4:0]	Input	写口地址
raddr1[4:0]	Input	读口地址1
raddr2[4:0]	Input	读口地址2
wen	Input	写使能
wdata[31:0]	Input	写数据
rdata1[31:0]	Output	读口1数据
rdata2[31:0]	Output	读口2数据

寄存器(Register)

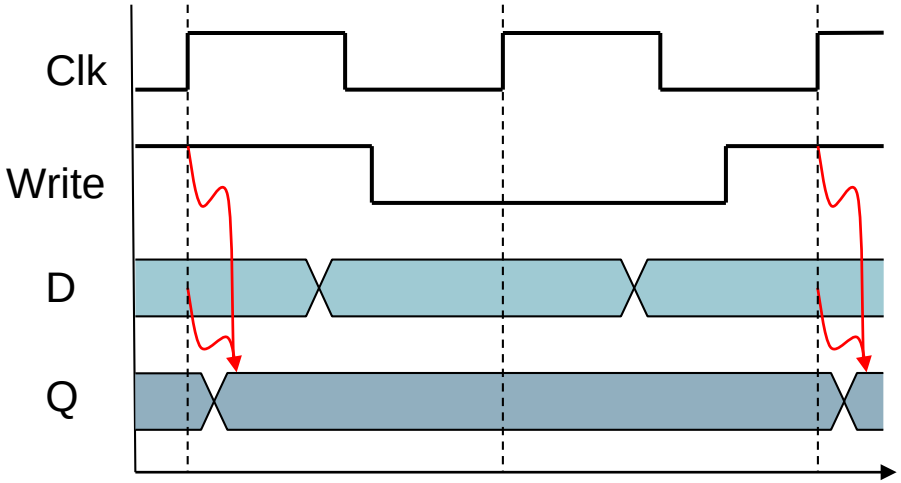
❑ Register

- Similar to the D Flip Flop except
  - N-bit input and output
  - Write Enable input



❑ Write Enable:

- =0: Data Out will not change
- =1: Data Out will become Data In (on the clock edge)
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later



一个1位寄存器

```
wire clk;
reg r;
wire data;

always @(posedge clk) begin
   r <= data;
end
```

寄存器组

- 多个存储长度相同的寄存器的组合
- 不能整体访问
- 地址 = 寄存器的编号
  - 读出5号寄存器的内容
  - 把数据写入17号寄存器

```
wire clk;                                    ???
reg r [7:0];  // reg [7:0] r;
wire rdata;
wire wdata;
wire [2:0] raddr;
wire [2:0] waddr;

always @(posedge clk) begin
    r[waddr] <= wdata;
end

assign rdata = r[raddr];
```

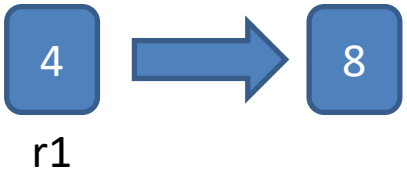
- 由8个1位寄存器组成的寄存器组
  - 通过二维寄存器引入”地址”的概念
  - 带有1个读口和1个写口
    - 异步读 = 读操作不受时钟约束 = 组合逻辑
    - 同步写 = 写操作受时钟约束 = 时序逻辑
  - 读写地址独立访问

MIPS中的通用寄存器堆

- 由32个32位寄存器组成的寄存器组
- 两个读口 + 一个写口
- 写使能信号 wen
  - 并不是所有指令都会修改通用寄存器, 如跳转指令
  - 寄存器堆需要提供写使能信号, 有效才写
- 0号寄存器（$zero）
  - 从0号地址读出的值总是常量32'b0

读写同一寄存器

- **add r1, r1, r1**
  - 新r1 = 旧r1 + 旧r1
- **raddr1 = raddr2 = waddr = 1**
  - 读出旧值
  - 算加法
  - 加法结果在下一个clk上升沿到来时写入
- **写入由时钟控制, 不会有问题**

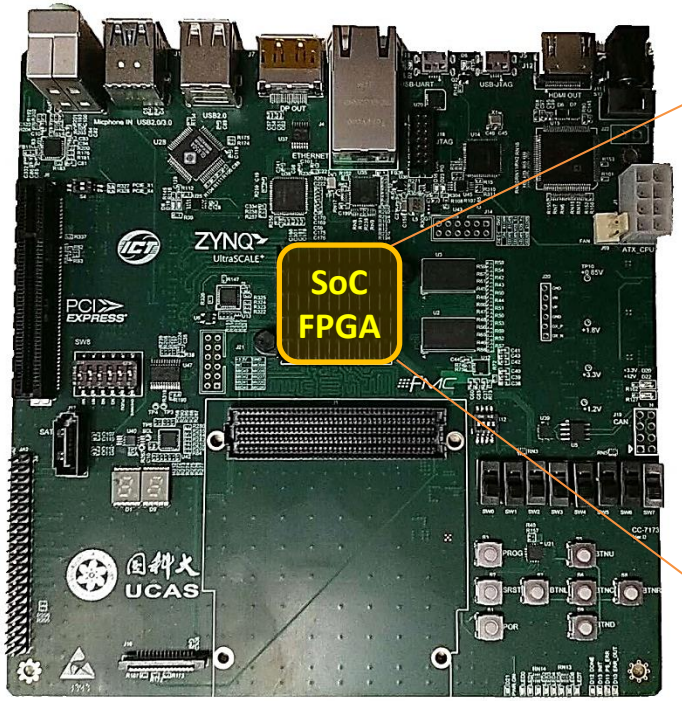


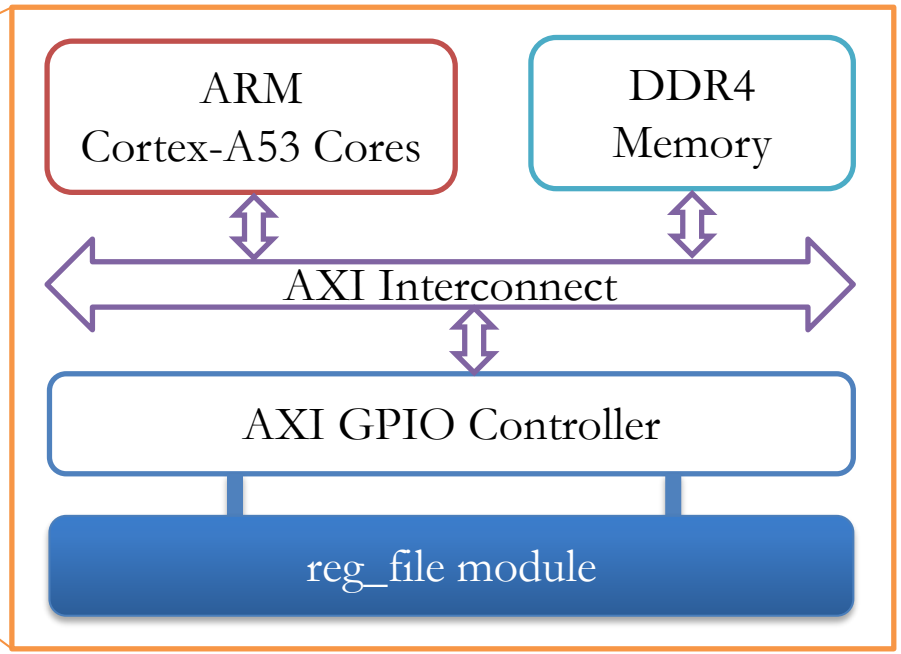
内容大纲

❑通用寄存器堆（Register File）功能部件设计

- 通用寄存器堆基本功能定义及设计要点讲解
- 通用寄存器堆实验操作流程及提交要求

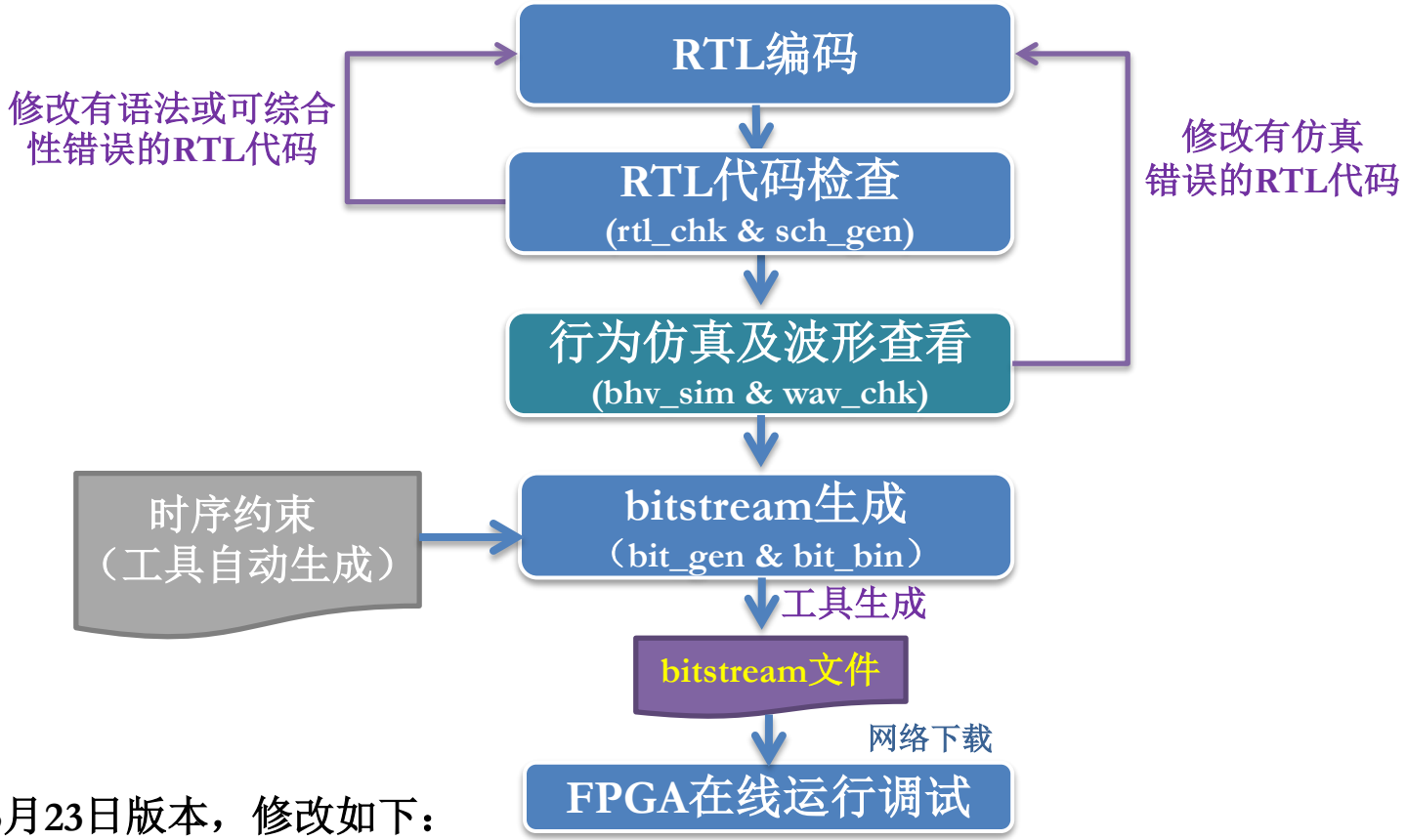
实验环境及工程框架





通过板卡ARM处理器上运行的软件，
控制寄存器堆读/写操作并自动检查比对执行结果

实验流程（3月24日更新）



- **相比3月23日版本，修改如下：**
  - 无需手动执行创建工程的动作prj_gen（脚本自动执行）
  - 代码检查命令变更为make HW_ACT=rtl_chk HW_VAL=reg_file vivado_prj
  - 原理图生成命令变更为make HW_ACT=sch_gen HW_VAL=reg_file vivado_prj
  - 其他命令格式不变

设计输入（编写RTL代码）

- **按照功能和接口定义完成reg_file.v模块的编写**
  - 已提供代码框架源文件，位于本地仓库hardware/sources/ip_catalog目录下

```
`timescale 10 ns / 1 ns

`define DATA_WIDTH 32
`define ADDR_WIDTH 5

module reg_file(
        input clk,
        input rst,
        input [`ADDR_WIDTH - 1:0] waddr,
        input [`ADDR_WIDTH - 1:0] raddr1,
        input [`ADDR_WIDTH - 1:0] raddr2,
        input wen,
        input [`DATA_WIDTH - 1:0] wdata,
        output [`DATA_WIDTH - 1:0] rdata1,
        output [`DATA_WIDTH - 1:0] rdata2
);

        // TODO: Please add your logic code here

endmodule
```

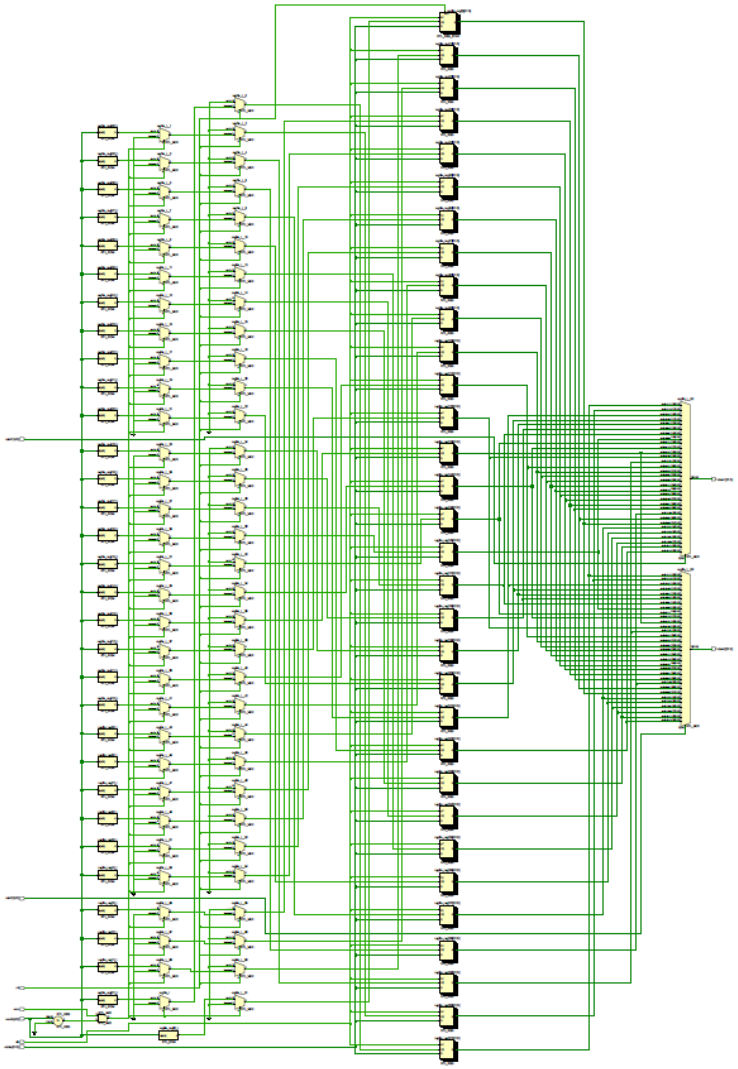
检查RTL代码及生成原理图

❑ RTL代码检查

- 在本地仓库中执行
  make HW_ACT=rtl_chk
  HW_VAL=reg_file vivado_prj
- 启动Vivado进行reg_file模块的RTL代码检查
- 请仔细核对出现的Warning和Error，并修改RTL代码，直至没有Warning和Error出现

❑ 原理图生成

- 在本地仓库中执行
  make HW_ACT=sch_gen
  HW_VAL=reg_file vivado_prj
- 启动Vivado图形界面再次进行reg_file模块的代码检查并生成RTL原理图（Schematics），生成后图形界面自动关闭
- 原理图（reg_file_sch.pdf）位于hardware/vivado_out/rtl_chk目录内



行为仿真

- **在本地仓库中执行**
  make HW_ACT=bhv_sim HW_VAL="reg_file <sim_time>" vivado_prj
  - <sim_time>表示仿真时间（时间单位为微秒μs），输入值必须是一个正实数（如<sim_time>=0.2，则对应仿真时间为200ns）
  - 举例： make HW_ACT=bhv_sim HW_VAL="reg_file 2" vivado_prj
    - 对reg_file模块进行仿真，仿真时间2μs
    - 注意：reg_file和2之间有空格，且双引号不能去掉
  - reg_file模块用到的仿真testbench为本地仓库 hardware/sources/testbench/reg_file_test.v，已提供框架代码（如右图），请同学们自行补充测试激励

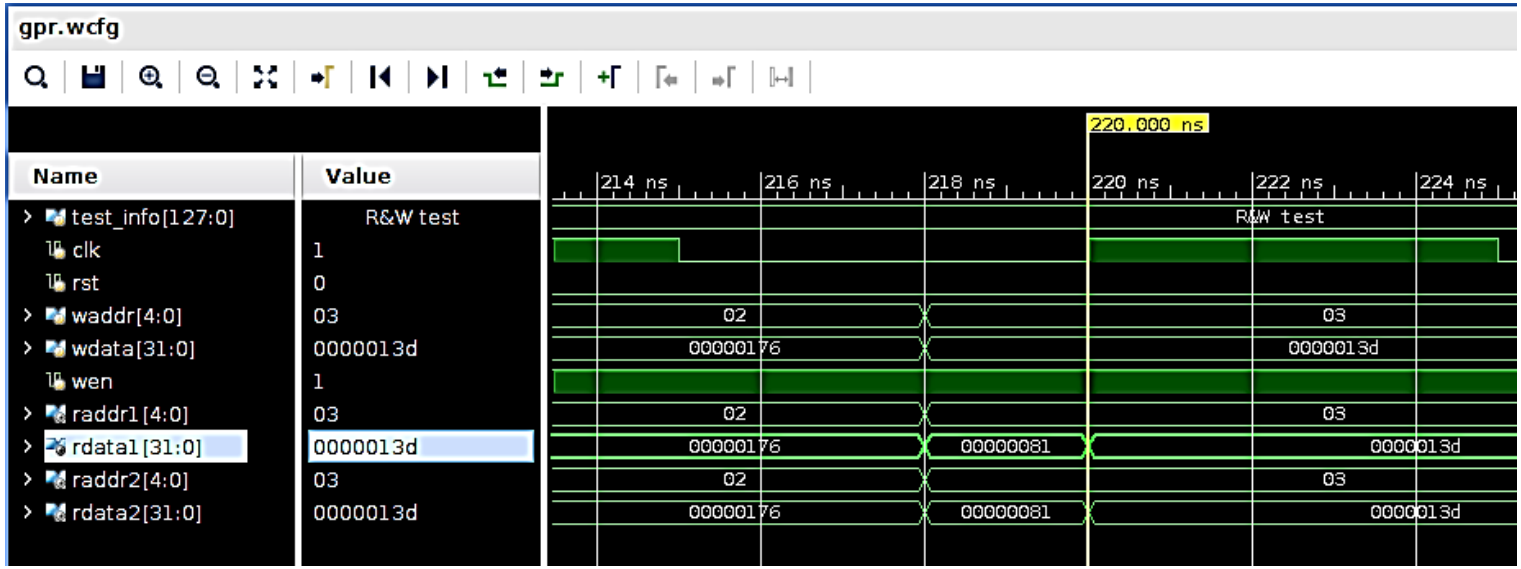
```
`timescale 1ns / 1ns

`define DATA_WIDTH 32
`define ADDR_WIDTH 5

module reg_file_test
();

        reg clk;
        reg rst;
        reg [`ADDR_WIDTH - 1:0] waddr;
        reg wen;
        reg [`DATA_WIDTH - 1:0] wdata;

        reg [`ADDR_WIDTH - 1:0] raddr1;
        reg [`ADDR_WIDTH - 1:0] raddr2;
        wire [`DATA_WIDTH - 1:0] rdata1;
        wire [`DATA_WIDTH - 1:0] rdata2;

        initial begin
                // TODO: Please add your testbench here
        end

        always begin
                #5 clk = ~clk;
        end

        reg_file u_reg_file(
                .clk(clk),
                .rst(rst),
                .waddr(waddr),
                .raddr1(raddr1),
                .raddr2(raddr2),
                .wen(wen),
                .wdata(wdata),
                .rdata1(rdata1),
                .rdata2(rdata2)
        );

endmodule
```

查看行为仿真波形

- 在本地仓库中执行

  make HW_ACT=wav_chk HW_VAL=reg_file vivado_prj

  - 在Vivado图形界面中打开仿真波形
  - 若在Vivado图形界面中对波形文件有所修改（添加/删除信号）并保存，需关闭Vivado图形界面重新执行仿真（HW_ACT=bhv_sim），再查看新波形（HW_ACT=wav_chk）



生成bitstream

- **在本地仓库中执行**
  make HW_ACT=bit_gen HW_VAL=reg_file vivado_prj
  - 自动完成综合、布局布线和bitstream生成（.bit后缀文件）
  - 生成的bitstream文件位于本地仓库顶层的hw_plat目录下

- **在本地仓库中执行make bit_bin**
  - 在本地仓库顶层的hw_plat目录下生成二进制bitstream文件（.bit.bin后缀文件）
  - 准备用于上板测试运行

本地与云端上板测试运行

- 本地上板测试时，请参考实验0讲义进行笔记本物理机和虚拟机网络的配置
- 在虚拟机的本地仓库中执行

  make BOARD_IP=<ip_address> local_run连接ZyForce本地板卡或

  make USER=<user_name> cloud_run连接ZyForce云端远程板卡

  - ip_address为本地板卡与笔记本电脑直连局域网内的IP地址
  - user_name为同学们姓名小写字母全拼（例如：张三同学对应user_name名为zhangsan）
    - 韩雨同学的usser_name值为hanyu1
    - 穆热迪力同学的usser_name值为muredili
  - 执行完成后将进入交互式命令行，可输入交互调试命令（严格区分命令大小写）
  - 注意：远程执行命令与实验项目0略有不同，不同点已用红色标出

交互调试命令	功能描述	执行效果
reg_file_eval	运行通用寄存器堆自动测试程序并比对输出结果	显示“Test Passed”表示通过测试；否则显示“Test Failed”及出错原因
help	输出交互命令格式	在交互式命令行显示
quit	退出交互式命令行	完成测试，退出运行

```
Please input command:
reg_file_eval
reg_file_eval
Test r/w on r0
Test r/w on r1~r31
⋮
writing 02901d82 to r26 wen=0
writing 3a95f874 to r27 wen=0
writing 08138641 to r28 wen=0
writing 1e7ff521 to r29 wen=0
writing 7c3dbd3d to r30 wen=0
writing 737b8ddc to r31 wen=0
reg_file_eval: Test passed (63 in total)
Please input command:
```

课堂验收要求

- 查看仿真波形，检查“异步读”和“读写同一寄存器”的功能实现效果
  - ARM软件程序串行控制寄存器堆读写端口，无法并行操作
- 检查在本地ZyForce板卡上的自动测试执行情况
- 课堂检查验收截止时间 - **2018/04/13 20:49:59**
  - 将与ALU一同进行课堂验收

实验1阶段提交

- **提交前请务必检查**
  - 已完成实验项目中关于通用寄存器堆的全部硬件设计流程
  - 已完成ZyForce云环境测试及ZyForce本地板卡环境下的通用寄存器堆测试
- **提交方法**
  - 在实验项目1本地仓库执行git push origin master
- **截止时间: 如无特殊原因, 迟交作业的同学将损失本次实验项目完整成绩的30%**
  - 代码提交截止时间 - 2018/03/30 20:49:59
  - 助教以最后一次提交的时间及内容作为评分依据

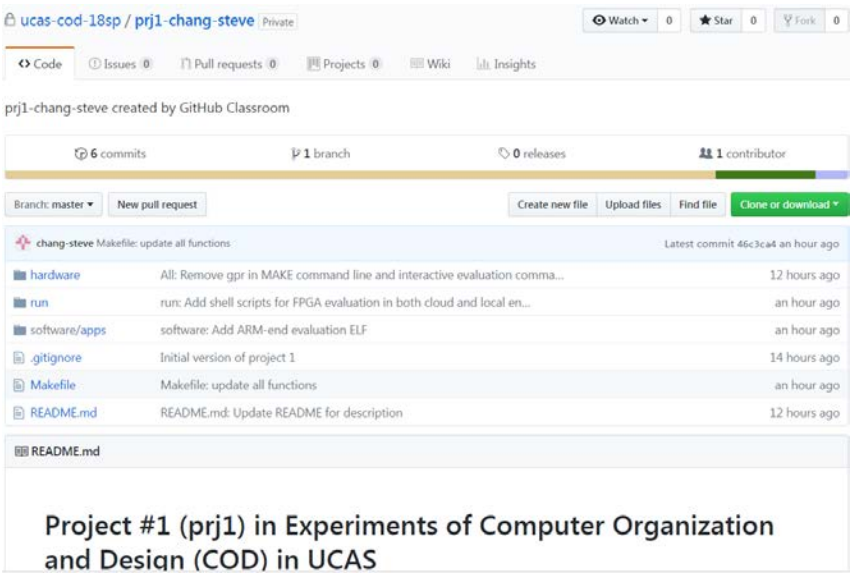
其他说明：查看开发日志

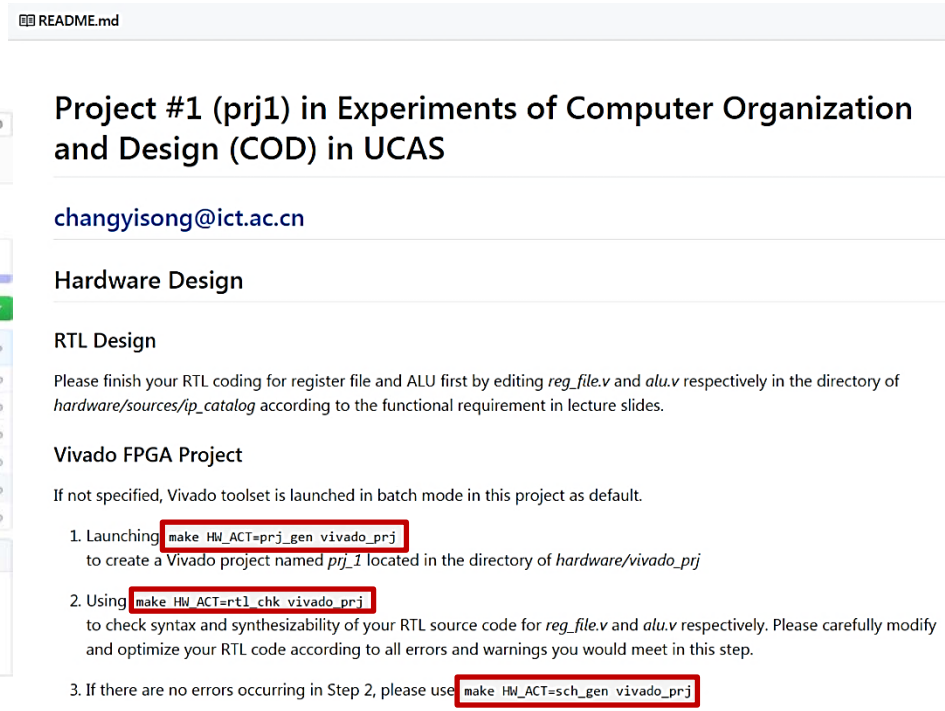
❑ 查看Vivado开发日志

- 日志所在目录：hardware/vivado_out/run_log/
- 日志命名规则：<HW_ACT>.log
  - HW_ACT=prj_gen/rtl_chk/bhv_sim/bit_gen
- 除sch_gen和wav_chk操作外，其余操作每次执行后都会把输出在终端屏幕上的信息记录在对应日志文件中，可追溯执行过程中遇到的Error、Warning等信息

Make命令经常敲错

- **实验项目0中遇到的问题**
  - 直接复制ppt里的文字，有可能会出现引号错误
  - ppt中的命令，空格显示不清楚，导致同学们出现少敲了一个空格的情况
- **建议解决方法**
  - 通过浏览器查看个人远程仓库中的README.md
  - 打卡个人远程仓库即可显示





Git自动提交错误补充说明

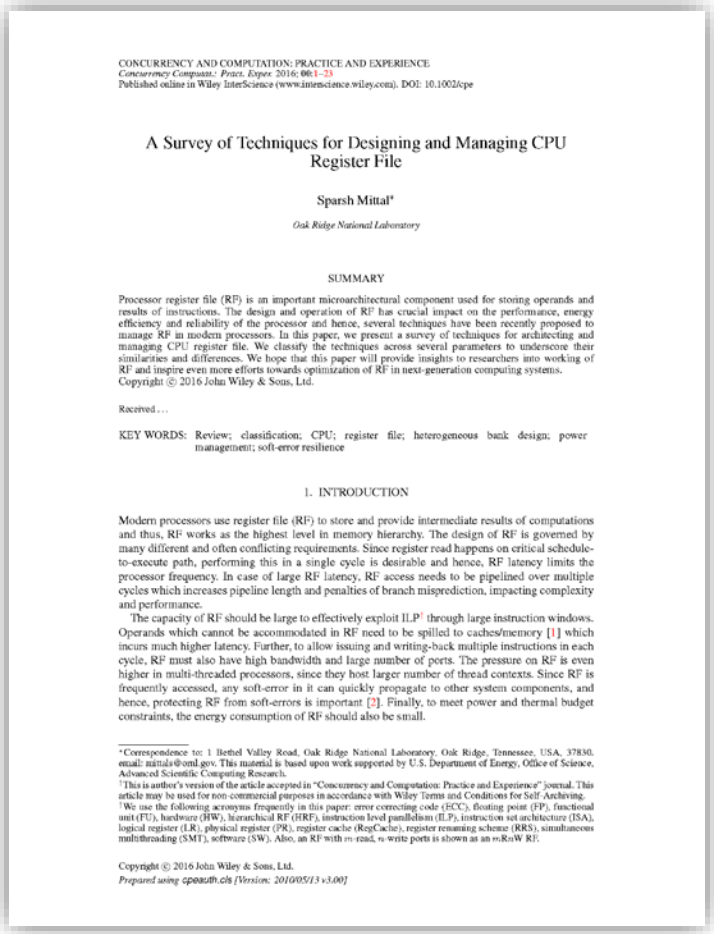
- 在执行Vivado开发流程时会因Git没有可提交的内容报错

```
nothing to commit, working directory clean
Makefile:53: recipe for target 'vivado_prj' failed
make: [vivado_prj] Error 1 (ignored)
```

- 不是由于Vivado执行错误，可以直接忽略

关于寄存器堆的高阶补充知识：

- **A Survey of Techniques for Designing and Managing CPU Register File**
  - Sparsh Mittal, Oak Ridge National Laboratory



CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; 00:1–23
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

A Survey of Techniques for Designing and Managing CPU Register File

Sparsh Mittal*

Oak Ridge National Laboratory

SUMMARY

Processor register file (RF) is an important microarchitectural component used for storing operands and results of instructions. The design and operation of RF has crucial impact on the performance, energy efficiency and reliability of the processor and hence, several techniques have been recently proposed to manage RF in modern processors. In this paper, we present a survey of techniques for architecting and managing CPU register file. We classify the techniques across several parameters to underscore their similarities and differences. We hope that this paper will provide insights to researchers into working of RF and inspire even more efforts towards optimization of RF in next-generation computing systems.

Received . . .

KEY WORDS: Review; classification; CPU; register file; heterogeneous bank design; power management; soft-error resilience

1. INTRODUCTION

Modern processors use register file (RF) to store and provide intermediate results of computations and thus, RF works as the highest level in memory hierarchy. The design of RF is governed by many different and often conflicting requirements. Since register read happens on critical schedule-to-execute path, performing this in a single cycle is desirable and hence, RF latency limits the processor frequency. In case of large RF latency, RF access needs to be pipelined over multiple cycles which increases pipeline length and penalties of branch misprediction, impacting complexity and performance.

The capacity of RF should be large to effectively exploit ILP[†] through large instruction windows. Operands which cannot be accommodated in RF need to be spilled to caches/memory [1] which incurs much higher latency. Further, to allow issuing and writing-back multiple instructions in each cycle, RF must also have high bandwidth and large number of ports. The pressure on RF is even higher in multi-threaded processors, since they host larger number of thread contexts. Since RF is frequently accessed, any soft-error in it can quickly propagate to other system components, and hence, protecting RF from soft-errors is important [2]. Finally, to meet power and thermal budget constraints, the energy consumption of RF should also be small.

*Correspondence to: 1 Bethel Valley Road, Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA, 37830. email: mittals@ornl.gov. This material is based upon work supported by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research.
†This is author's version of the article accepted in "Concurrency and Computation: Practice and Experience" journal. This article may be used for non-commercial purposes in accordance with Wiley Terms and Conditions for Self-Archiving.
‡We use the following acronyms frequently in this paper: error correcting code (ECC), floating point (FP), functional unit (FU), hardware (HW), hierarchical RF (HRF), instruction level parallelism (ILP), instruction set architecture (ISA), logical register (LR), physical register (PR), register cache (RegCache), register renaming scheme (RRS), simultaneous multithreading (SMT), software (SW). Also, an RF with n_r-read, n_w-write ports is shown as an n_rnRnW RF.

Prepared using cpeauth.cls *[Version: 2010/05/13 v3.00]*

Q & A ?


中国科学院大学
University of Chinese Academy of Sciences
ICT
中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY,CAS

B62007Y 2017-2018学年春季学期

计算机组成原理实验

实验项目1 基本功能部件设计

——Register File & ALU

常轶松 赵然 王诲喆

2018年3月30日

中国科学院大学
University of Chinese Academy of Sciences

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY,CAS

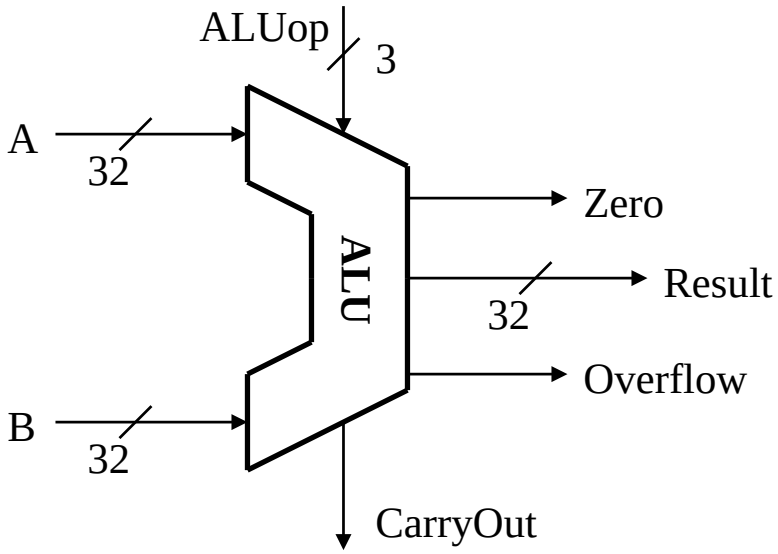
实验目的

- **设计MIPS处理器中使用的32-bit通用寄存器堆（Register File）和算术逻辑单元（ALU）**
  - 设计包含32个32-bit通用寄存器的寄存器堆（已完成）
  - 设计基本算术逻辑单元ALU
    - 支持逻辑按位“与”（AND）、逻辑按位“或”（OR）、算术加法（Add）、算术减法（Subtract）、有符号整数比较（Set on less than，Slt）等5种基本ALU运算功能
    - 理解数的表示方法及对应ALU计算标志位的含义
  - 掌握实验项目的阶段提交方法
  - 为后续完整MIPS CPU设计实验准备好可用组件

内容大纲

- 通用寄存器堆（Register File）功能部件设计
- ALU功能部件设计
  - ALU基本功能介绍
  - ALU实验操作流程及提交要求

32-bit ALU接口定义

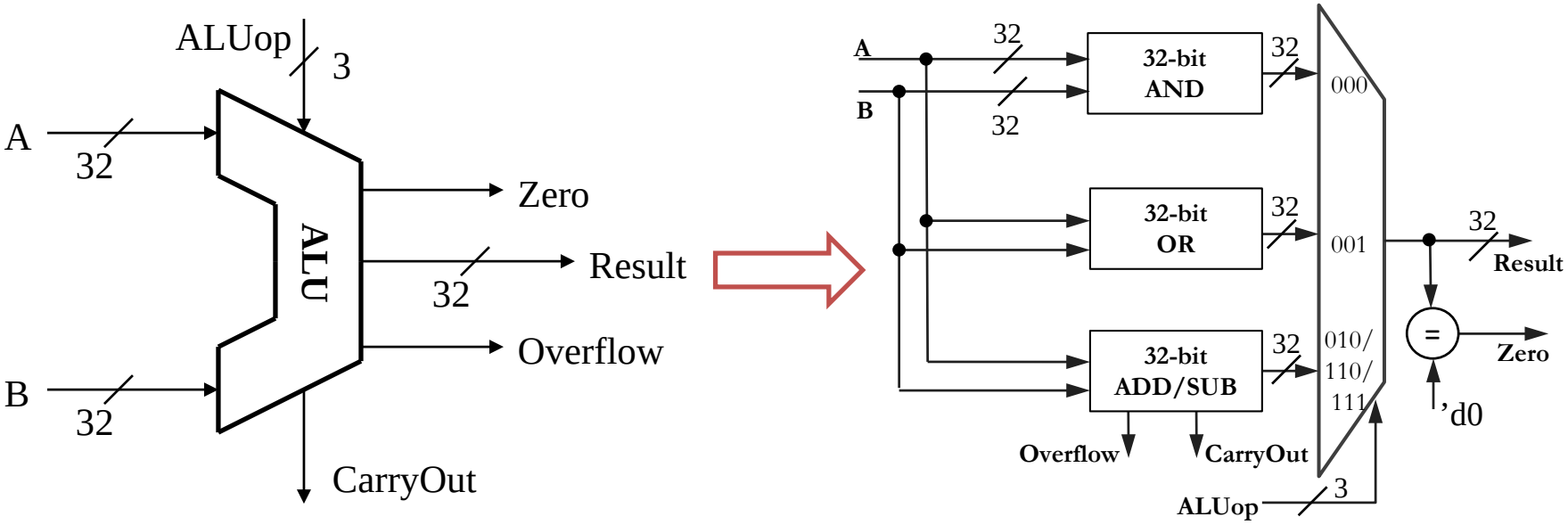


信号名	I/O	说明
A[31:0]	Input	操作数A
B[31:0]	Input	操作数B
ALUop[2:0]	Input	ALU部件控制端信号
Overflow	Output	指示有符号数的加减法运算结果是否溢出, 非此操作时结果未定义
CarryOut	Output	指示无符号数的加减法运算结果是否产生进位/借位, 非此操作时结果未定义
Zero	Output	指示运算结果是否为 0
Result[31:0]	Output	ALU的输出

纯组合逻辑电路实现！

需实现的五种ALU基本操作

ALUop	操作	使用相应操作的MIPS指令举例
000	逻辑按位与（AND）	and
001	逻辑按位或（OR）	or
010	算术加法（Add）	add, lw (load word), sw (store word)
110	算术减法（Subtract）	sub, beq (branch on equal)
111	有符号整数比较（Slt）	slt (set on less than)

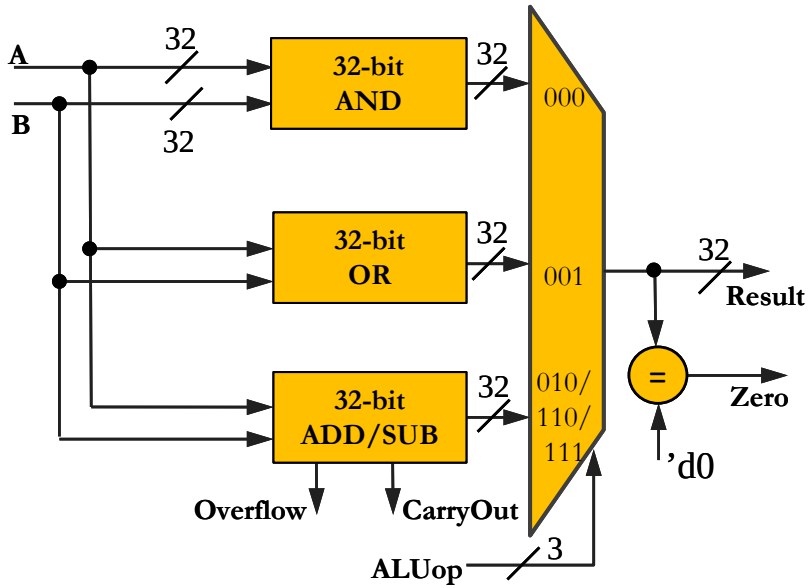


Overflow与CarryOut

- 溢出标志Overflow表示有符号数加减法操作结果是否超出范围，非此操作时结果未定义
  - 两个正数的加法结果是一个负数
  - 两个负数的加法结果是一个正数
- 进借位标志CarryOut表示无符号数加减法操作结果是否超出范围，非此操作时结果未定义
  - 加法计算结果比被加数小（有进位）
  - 减法计算结果比被减数大（有借位）
- https://en.wikipedia.org/wiki/Integer_overflow#Flags
- The overflow flag is set when the result of an operation on signed numbers does not have the sign that one would predict from the signs of the operands, e.g. a negative result when adding two positive numbers. This indicates that an overflow has occurred and the signed result represented in two's complement form would not fit in the given number of bits.
- The carry (carryout) flag is set when the result of an addition or subtraction, considering the operands and result as unsigned numbers, does not fit in the given number of bits. This indicates an overflow with a carry or borrow from the most significant bit. An immediately following add with carry or subtract with borrow operation would use the contents of this flag to modify a register or a memory location that contains the higher part of a multi-word value.

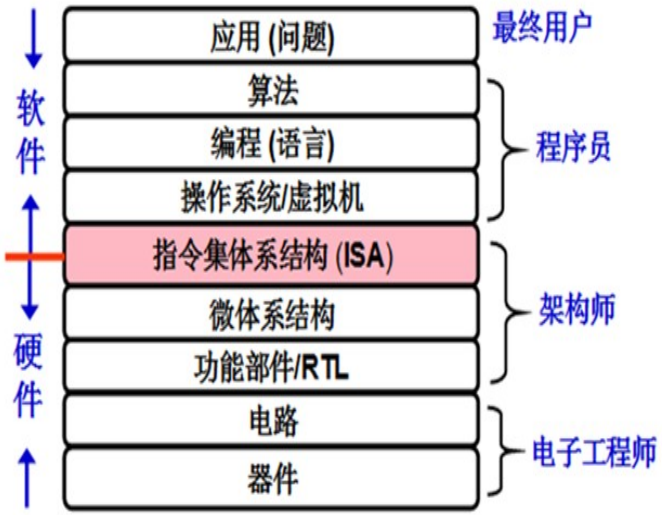
信号列表中的“未定义”

- 有些信号的输出在某些操作时无意义/未定义
  - 例如，ALU进行AND操作时Overflow无意义
  - 再如，ALU进行有符号数运算操作时CarryOut无意义
- 产生这些信号的电路一直都存在
  - 问题：当ALUop为AND操作时，哪部分电路在工作？
  -
- 如何解决?



解决方案 - 约定(specification)

- 微结构层次内的约定
  - 其它逻辑不要使用未定义信号
  - 不要使用 = 不要受影响
- ISA与程序员/编译器的约定
  - 程序员/编译器不要使用未定义结果
  - 例子 - 除0 (MIPS程序员手册)
- 这些层次之间是有联系的
- 基于约定的处理方法
  - 随便赋个确定的值
    - 反正没人用



Format: `DIVU rs, rt`

Purpose:

To divide a 32-bit unsigned integers

Description: `(LO, HI) ← rs / rt`

The 32-bit word value in GPR *rs* is divided by the 32-bit value in GPR *rt*
The 32-bit quotient is placed into special register *LO* and the 32-bit rema

No arithmetic exception occurs under any circumstances.

Restrictions:

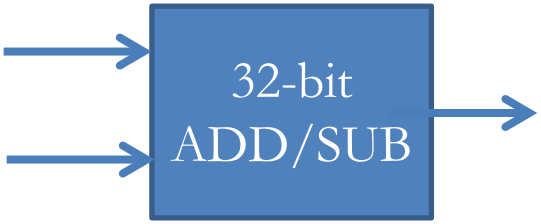
If the divisor in GPR *rt* is zero, the arithmetic result value is undefined.

对于A/B操作数和Result的理解

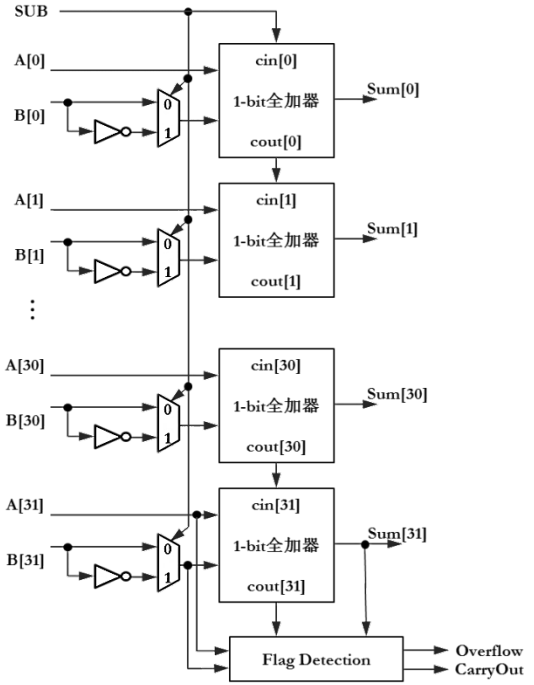
- 以下比特串(机器数) 对应的真值(数学值)是多少?
  - 1000 0000 0000 0000 0000 0000 0000 0000
- 不知道!!!
- 因为没有说明编码方式!
  - 补码: -2147483648
  - 无符号编码: 2147483648
  - 原码: -0
  - IEEE 754: -0.0
- 所以对一个比特串的解释(interpretation)有多种方式

加法操作的操作数和结果

- 这个加法器如何计算不同的加法表达式?
  - $-2^{31} + 1$（有符号加法）
  - $2^{31} + 1$（无符号加法）



- 两种表达式的计算过程完全相同
  - 加法器不区分无符号数加法和有符号数补码表示加法的输入操作数，也不区分不同的加法操作含义
  - 加法器通过相同的电路来完成运算输出二进制结果，并配合ALU产生所有的标志位

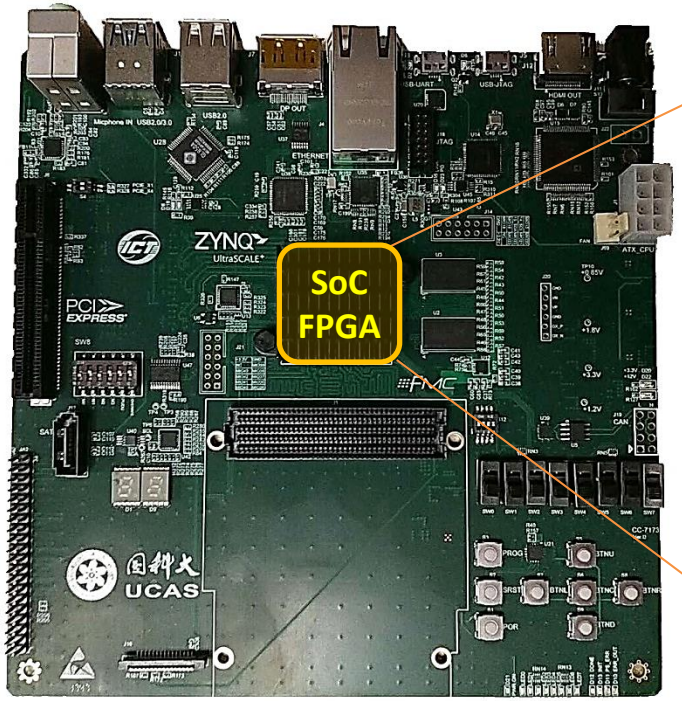


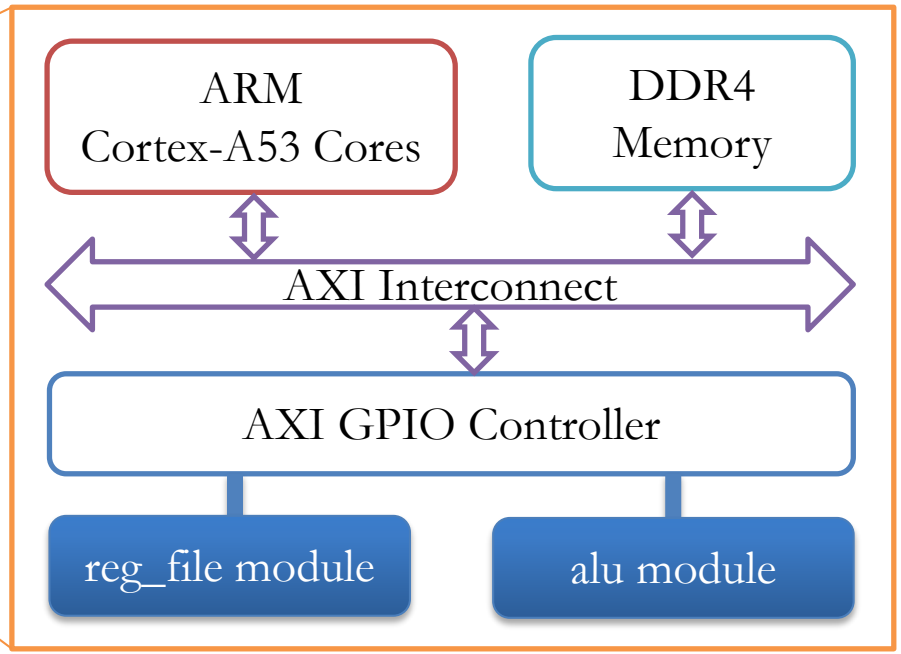
- 计算机如何表达不同的加法操作?
  - 编译器生成的不同加法运算指令
    - 负责向加法器传送输入操作数
    - 使用加法器的输出结果及对应的标志位
  - 还记得C语言里面的**int**和**unsigned int**吗?

内容大纲

- 通用寄存器堆（Register File）功能部件设计
- **ALU功能部件设计**
  - ALU基本功能介绍
  - ALU实验操作流程及提交要求

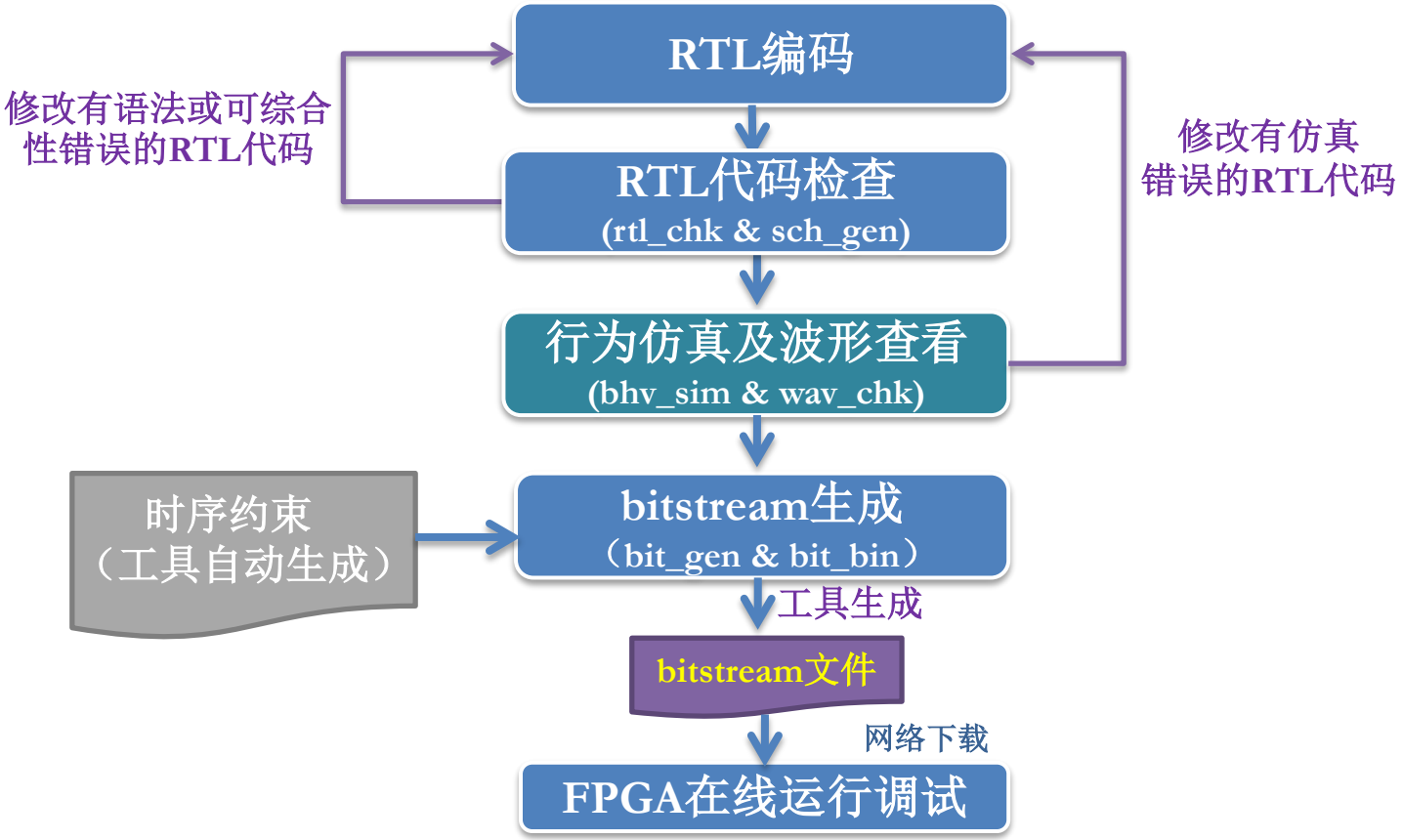
实验环境及工程框架





通过板卡ARM处理器上运行的软件，
控制ALU和寄存器堆操作，并自动检查比对执行结果

实验流程（3月24日更新）



- 请同学们使用上周实验创建的个人本地仓库，继续完成ALU部分的实验
- 实验操作命令及步骤可参考个人本地仓库中的README.md说明文档

设计输入（编写RTL代码）

- **按照功能和接口定义完成alu.v模块的编写**
  - 已提供代码框架源文件，位于本地仓库hardware/sources/ip_catalog目录下
  - 注意：无需用32个1-bit全加器搭建32-bit加法器，直接写“+”运算符对32-bit数据进行加法运算

```
`timescale 10 ns / 1 ns

`define DATA_WIDTH 32

module alu(
        input [`DATA_WIDTH - 1:0] A,
        input [`DATA_WIDTH - 1:0] B,
        input [2:0] ALUop,
        output Overflow,
        output CarryOut,
        output Zero,
        output [`DATA_WIDTH - 1:0] Result
);

        // TODO: Please add your logic code here

endmodule
```

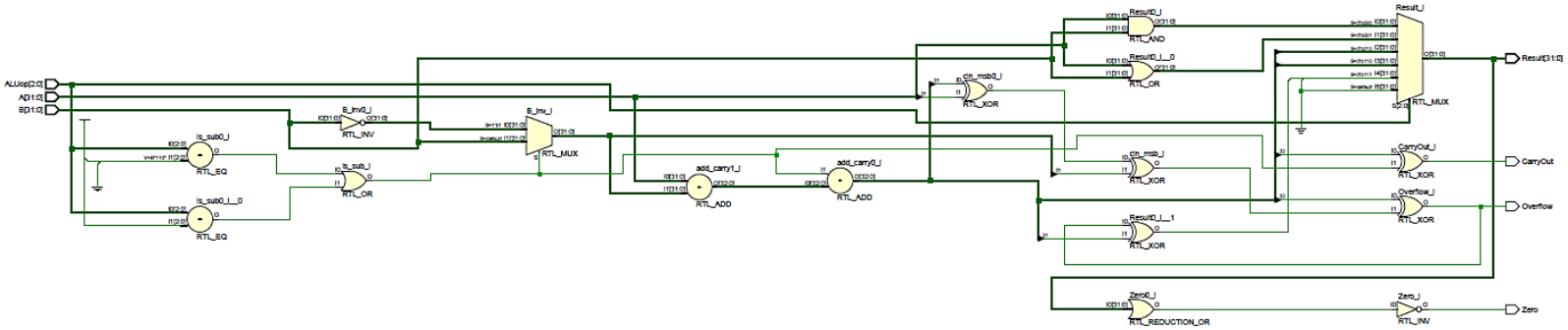
检查RTL代码及生成原理图

❑ RTL代码检查

- 在本地仓库中执行make HW_ACT=rtl_chk HW_VAL=alu vivado_prj
- 启动Vivado执行alu模块RTL代码检查
- 请仔细核对出现的Warning和Error，并修改RTL代码，直至没有Warning和Error出现

❑ 原理图生成

- 在本地仓库中执行make HW_ACT=sch_gen HW_VAL=alu vivado_prj
- 启动Vivado图形界面再次进行alu模块的代码检查并生成RTL原理图（Schematics），生成后图形界面自动关闭
- 原理图（alu_sch.pdf）位于hardware/vivado_out/rtl_chk目录内



行为仿真

- 在本地仓库中执行
  make HW_ACT=bhv_sim HW_VAL="alu <sim_time>" vivado_prj
  - <sim_time>表示仿真时间（时间单位为微秒μs），输入值必须是一个正实数（如<sim_time>=0.2，则对应仿真时间为200ns）
  - 举例： make HW_ACT=bhv_sim HW_VAL="alu 2" vivado_prj
    - 对alu模块进行仿真，仿真时间2μs
    - 注意：alu和2之间有空格，且双引号不能去掉
  - alu模块用到的仿真testbench为本地仓hardware/sources/testbench/alu_test.v，已提供框架代码（如右图），请同学们自行补充测试激励

```
`timescale 10ns / 1ns

`define DATA_WIDTH 32

module alu_test
();

        reg [`DATA_WIDTH - 1:0] A;
        reg [`DATA_WIDTH - 1:0] B;
        reg [2:0] ALUop;
        wire Overflow;
        wire CarryOut;
        wire Zero;
        wire [`DATA_WIDTH - 1:0] Result;

        initial
        begin
                // TODO: Please add your testbench here
        end

        alu u_alu(
                .A(A),
                .B(B),
                .ALUop(ALUop),
                .Overflow(Overflow),
                .CarryOut(CarryOut),
                .Zero(Zero),
                .Result(Result)
        );

endmodule
```

查看行为仿真波形

- 在本地仓库中执行

  make HW_ACT=wav_chk HW_VAL=alu vivado_prj

  - 在Vivado图形界面中打开仿真波形
  - 若在Vivado图形界面中对波形文件有所修改（添加/删除信号）并保存，需关闭Vivado图形界面重新执行仿真（HW_ACT=bhv_sim），再查看新波形（HW_ACT=wav_chk）

生成bitstream

- **在本地仓库中执行**
  make HW_ACT=bit_gen vivado_prj
  - 自动完成综合、布局布线和bitstream生成（.bit后缀文件）
  - 生成的bitstream文件位于本地仓库顶层的hw_plat目录下

- **在本地仓库中执行make bit_bin**
  - 在本地仓库顶层的hw_plat目录下生成二进制bitstream文件（.bit.bin后缀文件）
  - 准备用于上板测试运行

本地与云端上板测试运行

- 在虚拟机的本地仓库中执行

  make BOARD_IP=<ip_address> local_run连接ZyForce本地板卡或

  make USER=<user_name> cloud_run连接ZyForce云端远程板卡

  - 执行完成后将进入交互式命令行，可输入交互调试命令（严格区分命令大小写）

交互调试命令	功能描述	执行效果
reg_file_eval	运行通用寄存器堆自动测试程序并比对输出结果	显示“Test Passed”表示通过测试；否则显示“Test Failed”及出错原因
alu_eval	运行ALU自动测试程序并比对输出结果	
help	输出交互命令格式	在交互式命令行显示
quit	退出交互式命令行	完成测试，退出运行

```
Please input command:
alu_eval
alu_eval
1/320 passed
2/320 passed
3/320 passed
⋮
319/320 passed
320/320 passed
Test passed
Please input command:
```

```
66/320 failed
Error, SUB 0x00000000, 0x00000001 expects result 0xffffffff, overflow=0, carry=0, zero=0
                       but gets result 0xffffffff, overflow=0, carry=1, zero=0
67/320 failed
Error, SUB 0x00000000, 0x00000002 expects result 0xfffffffe, overflow=0, carry=0, zero=0
                       but gets result 0xfffffffe, overflow=0, carry=1, zero=0
```

实验项目1课堂验收要求

- **检查在本地ZyForce板卡上的自动测试执行情况**
  - 320个测试用例是否全部通过
- **课堂检查验收截止时间 - 2018/04/13 20:49:59**
  - reg_file和ALU一同进行课堂验收
    - 运行reg_file_eval和alu_eval

实验项目1网上提交内容

❑ 提交前请务必检查

- 已完成实验项目中关于通用寄存器堆和ALU的全部硬件设计流程
- 已完成ZyForce云环境测试及ZyForce本地板卡环境下的全部测试
  - 提交前使用ZyForce本地板卡或云环境时，请务必同时完成reg_file_eval和alu_eval两个测试，以保证评分系统能够一次性获取两个模块的评测结果

❑ 实验报告

- 请在实验项目1个人本地仓库中创建顶层目录doc（仅需执行一次）
  - mkdir –p doc
- 将实验报告的PDF版本放入doc目录，命名规则为“学号-prj1.pdf”，例如：
  - 2015K8009929000-prj1.pdf
- PDF文件大小应控制在5MB以内
- 实验报告内容中不必重复描述讲义中的实验流程, 但鼓励说明:
  - 你遇到的问题和对这些问题的思考
  - 或者你的其它想法, 例如实验心得, 对实验项目的建议, 对提供帮助的同学的感谢等
- 实验报告模板会发布在SEP网站上

实验项目1提交方法与截止时间

❑ 提交方法

- 确保doc目录下的实验报告PDF已添加到本地仓库中
  - 在实验项目1本地仓库执行git status –u查看PDF是否处于未提交状态

```
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       doc/2015K8009929000-prj1.pdf
```

  - 如文档还未添加到本地仓库（如上图），请执行
    git add --all && git commit –m “doc: Update project report”
- 在实验项目1本地仓库执行git push origin master

❑ 截止时间: 如无特殊原因, 迟交作业的同学将损失本次实验项目完整成绩的30%

- 代码提交截止时间 - 2018/04/20 18:09:59 （第七周上课前）
- 助教以最后一次提交的时间及内容作为评分依据

Q & A ?


中国科学院大学
University of Chinese Academy of Sciences
ICT
中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY,CAS