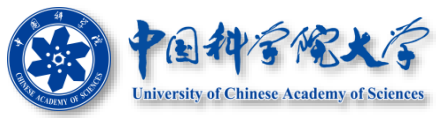
❑请在“国科大在线”APP签到

B0911007Y-01/02　2021-2022学年春季学期

计算机组成原理（研讨课）

实验项目4　定制RISC-V功能型处理器设计

2022年5月6日



0.1 实验内容与实验目的

- **基于实验项目3已实现的微结构，设计支持RISC-V 32-bit 整型指令集（RV32I）的功能型处理器**
  - 通过阅读手册，初步了解新兴RISC-V指令集
  - 进一步理解理论课讲授的精简指令集架构
  - **通过功能和性能评估，**对比RISC-V/MIPS指令集译码器的实现开销，理解RISC-V指令格式的设计思想

0.2 实验项目进度安排

课程内容	2022 4.15	4.22	4.29	5.6	5.13	5.20	5.27	5.30 (周一)
实验项目内容	实验发布			课堂讲解				
课堂验收							截止时间 18:59:59	
最终推送								截止时间 23:59:59

- 本次实验不设阶段提交
- 自5月13日上课时间起开始进行课堂验收
- 课堂验收截止：5月27日下课时（18:59:59）
- 最终提交截止时间前（5月30日周一23:59:59）需提交完整Verilog HDL代码及实验报告

1.1 RISC-V 32-bit整型指令集（RV32I）

The RISC-V Instruction Set Manual
Volume I: User-Level ISA
Document Version 2.2

Editors: Andrew Waterman[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu
May 7, 2017

RISC-V指令集手册（已上传至SEP网站）

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

与MIPS相比，RISC-V具有更加规整的指令格式

指令格式及指令集详见指令手册第104页

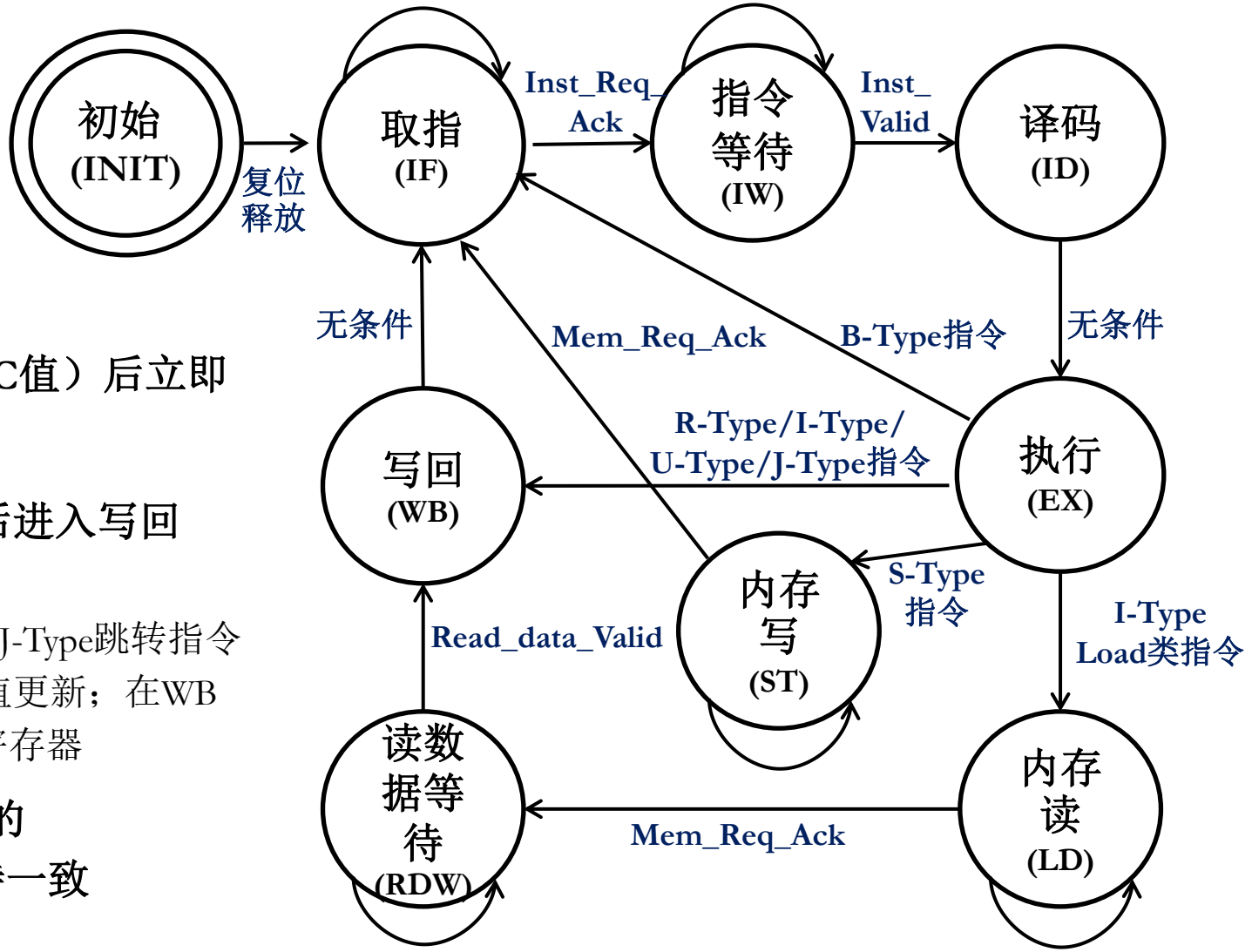
具体指令语义请查看指令手册第2章

本次实验要实现的37条基本指令

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000	pred | succ	00000	000	00000	0001111	FENCE
0000	0000 0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

1.2 处理器状态机



- 分支指令在执行阶段（更新PC值）后立即进入下一条指令的取指阶段
- R-Type类型指令在执行阶段后进入写回（WB）阶段
  - I-Type跳转指令（JALR）和J-Type跳转指令（JAL）在EX阶段完成PC值更新；在WB阶段将返回地址写入目的寄存器
- 其他状态与实验项目3中实现的多周期处理器状态机要求保持一致

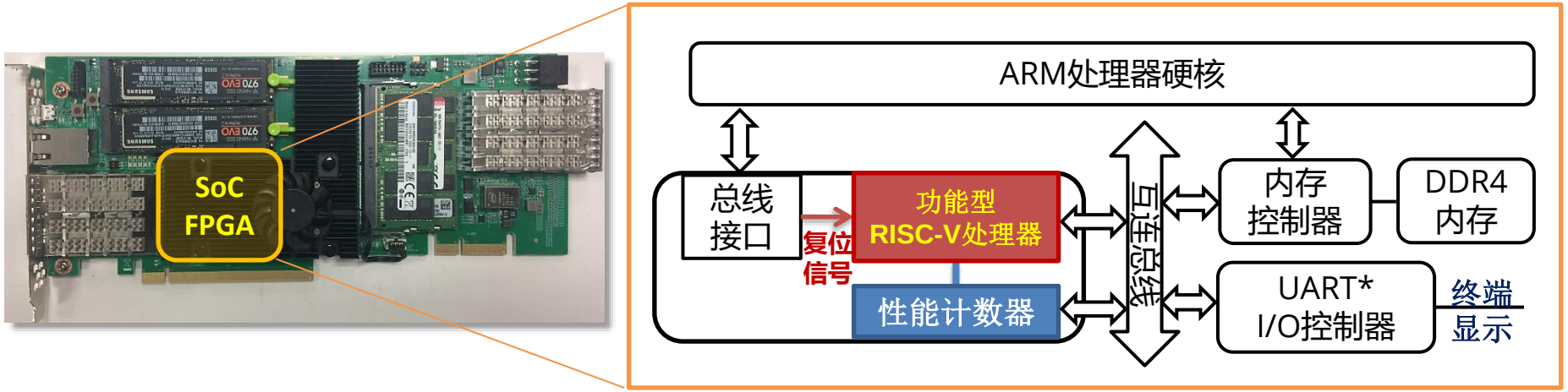
2.1 FPGA运行测试方法

❑在平台板卡的SoC-FPGA芯片中

a) 由ARM处理器向DDR4内存加载RISC-V处理器的各benchmark测试用例

b) ARM处理器释放RISC-V处理器核的复位信号，使RISC-V处理器开始运行

c) ARM处理器读取DDR4内存的0x0C地址，检查RISC-V是否正确运行测试用例



2.2.0* 设计输入——设置实验流程的目标模块

- 同步框架更新

```
cd ~/COD-Lab && git pull upstream master
```

- 编辑脚本文件

```
cd ~/COD-Lab && vim lab_env.yml
```

```
variables:
  # TARGET_DESIGN can be "example", "alu", "reg_file",
  TARGET_DESIGN: "custom_cpu"

  # CPU_ISA must be either "mips" or "riscv32"
  CPU_ISA: "mips"
```

将mips改为riscv32

```
  # single_cycle or multi_cycle
  SIM_DUT_TYPE: "multi_cycle"
```

- 确认TARGET_DESIGN设置为custom_cpu
- 将CPU_ISA设置为riscv32（如右图）
- 确认SIM_DUT_TYPE设置为multi_cycle
- 将修改提交到代码仓库

```
cd ~/COD-Lab && git add lab_env.yml && git commit -m “lab_env: set design flow for RISC-V custom_cpu”
```

2.2.1* 设计输入 —— RTL代码编写

- 添加多周期定制处理器数据及控制通路代码

```
cd ~/COD-Lab && vim fpga/design/ucas-cod/hardware/sources/custom_cpu/riscv32/custom_cpu.v
```

- 其他软硬件代码无需修改

```
module custom_cpu(
    input         clk,
    input         rst,

    //Instruction request channel
    output [31:0] PC,
    output        Inst_Req_Valid,
    input         Inst_Req_Ready,

    //Instruction response channel
    input  [31:0] Instruction,
    input         Inst_Valid,
    output        Inst_Ready,

    //Memory request channel
    output [31:0] Address,
    output        MemWrite,
    output [31:0] Write_data,
    output [ 3:0] Write_strb,
    output        MemRead,
    input         Mem_Req_Ready,

    //Memory data response channel
    input  [31:0] Read_data,
    input         Read_data_Valid,
    output        Read_data_Ready,

    input         intr,

    output [31:0] cpu_perf_cnt_0,
    output [31:0] cpu_perf_cnt_1,
    output [31:0] cpu_perf_cnt_2,
    output [31:0] cpu_perf_cnt_3,
    output [31:0] cpu_perf_cnt_4,
    output [31:0] cpu_perf_cnt_5,
    output [31:0] cpu_perf_cnt_6,
    output [31:0] cpu_perf_cnt_7,
    output [31:0] cpu_perf_cnt_8,
    output [31:0] cpu_perf_cnt_9,
    output [31:0] cpu_perf_cnt_10,
    output [31:0] cpu_perf_cnt_11,
    output [31:0] cpu_perf_cnt_12,
    output [31:0] cpu_perf_cnt_13,
    output [31:0] cpu_perf_cnt_14,
    output [31:0] cpu_perf_cnt_15
);
```

2.2.2* 设计输入—— 将代码修改提交到本地仓库

❑ 每次修改代码后，都需要手动把修改提交（commit）到本地仓库

`cd ~/COD-Lab && git add` 已修改的文件路径

`cd ~/COD-Lab && git commit`（自行添加提交说明）

❑ 提交说明的编写要求请自学

- 尽量使用英文，并充分描述本次代码修改的目的、内容、预期达到的效果等信息
- 如何写好提交说明
  https://www.ruanyifeng.com/blog/2016/01/commit_message_change_log.html （中文参考）

2.2.2* 设计输入—— 触发自动化开发流程

- 在完成一次或多次代码修改及提交后，可推送代码到SERVE个人远程仓库

```
cd ~/COD-Lab && git push origin master
```

- 推送之后，自动化开发及部署流程就会在云平台开始运行
  - 可在浏览器中打开GitLab上的个人远程仓库，查看开发任务的执行情况
  - 行为仿真（bhv_sim）只针对basic、medium、advanced和hello四组共31个测试用例
  - FPGA测试运行（fpga_eval）针对上述四组+microbench共40个测试用例

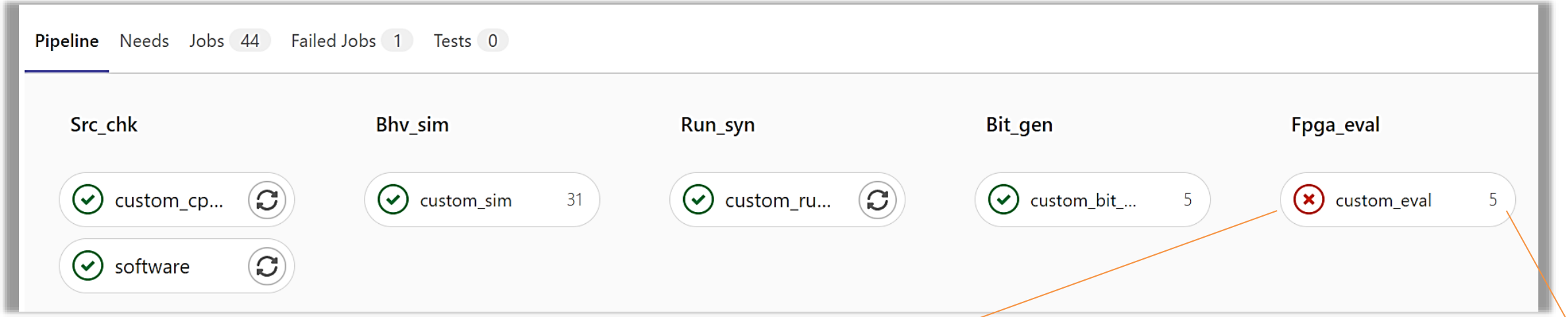
2.3* 查看错误的行为仿真波形

在虚拟机中执行（注意：下面的命令是一条完整的命令，一行写不下所以出现换行）

```
cd ~/COD-Lab && make FPGA_PRJ=ucas-cod FPGA_BD=nf SIM_TARGET=custom_cpu
SIM_DUT=riscv32:multi_cycle WORKLOAD=simple_test:benchmark组名:benchmark名称 wav_chk
```

- 需要保证虚机可以上网
- 注意：命令第一行的最后有一个空格；不要按回车，直接输入第二行
- benchmark组名为：basic、medium、advanced、hello四个中的一个
- benchmark名称可根据SERVE网站上正在调试的仿真Job名称查看
- 波形文件的最后是出错指令的情况（与上页PPT输出信息对应）
- 从波形中找到出错点，结合测试用例的汇编指令序列，向前找到出错点
- 可按需在波形中添加要观察的信号

同学们遇到的实验调试困境



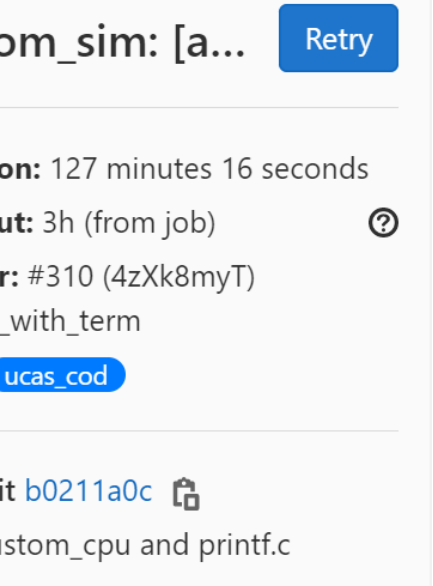
- 使用软件仿真工具（iverilog）对custom_cpu进行逻辑行为仿真（bhv_sim阶段）完全正确
- 但custom_cpu的真实逻辑电路进行FPGA硬件测试运行（fpga_eval阶段）出现错误
- **问题：如何定位custom_cpu的Verilog HDL代码中的设计缺陷？**

```
[queen] Queen placement: time 2894.98ms
custom cpu running time out
wait_for_finish: AXI firewall status: 00000001
reset: before MMIO access...
reset: MMIO accessed
./software/workload/ucas-cod/benchmark/simple_test/microbench/mips/elf/queen failed
Hit bad trap
 pass 8 / 9
```

挑战1：bhv_sim阶段速度慢且测试覆盖率低

- custom_cpu与之前设计相比，测试复杂度提升
  - 使用带有随机延时的内存模型，加剧仿真变慢
  - advanced:shuixianhua的仿真时间>120分钟

- 未对microbench仿真，测试覆盖不完整
  - 内存模型需模拟256MB内存大小，对服务器内存需求提高
  - 程序迭代次数多，仿真时间进一步加长

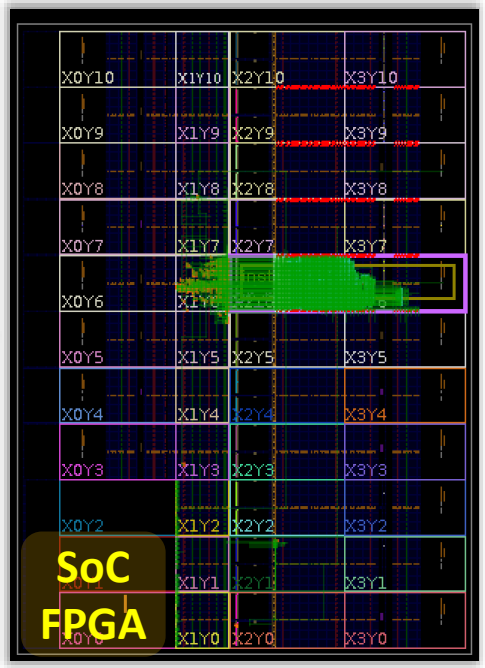


挑战2：fpga_eval阶段执行速度快但缺乏硬件电路调试手段

```
module cvc_enet_four_port
  #(parameter DATA_WIDTH = 64,
              COUNT_WIDTH = 4,
              NUM_PORTS = 4)
  (input clk,
         rst_n,
   input [NUM_PORTS-1 : 0] in_port_data_valid,
   input [NUM_PORTS*DATA_WIDTH-1 : 0] in_port_data,
   output reg [NUM_PORTS-1 : 0] out_port_data_sop,
                                out_port_data_eop,
   output [NUM_PORTS*DATA_WIDTH-1 : 0] out_port_data);
```

custom_cpu处理器设计
Verilog HDL代码

代码转换成
FPGA上的
硬件逻辑电路
(100MHz
工作频率)



在处理器
硬件电路上
运行测试用例

只需1-2分钟即可看到microbench运行结果

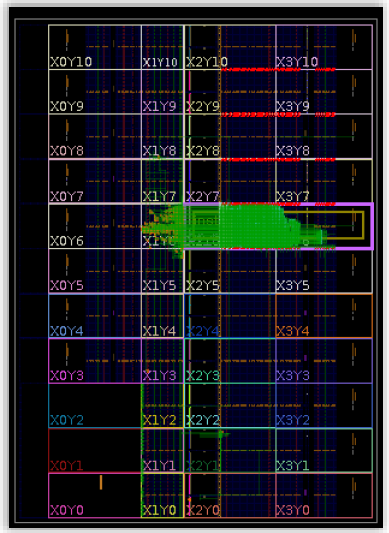
```
[15pz] A* 15-puzzle search: X Failed.
[bf] Brainf**k interpreter: X Failed.
pass 7 / 9
```

FPGA运行结果只能展示对错信息，不能在FPGA上观察真实处理器电路内部信号行为，无法定位处理器代码设计缺陷。真实芯片（如“香山”开源RISC-V处理器芯片）开发效率也受此影响

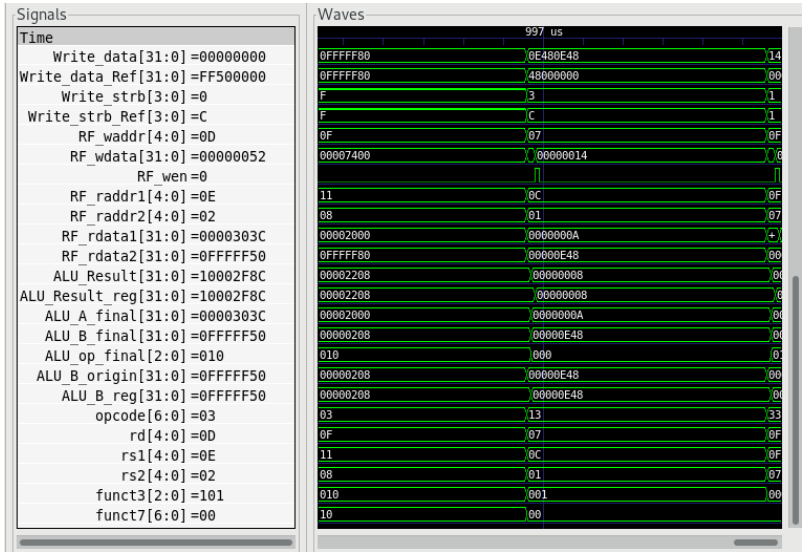
解决方案：FPGA仿真加速（Emulation）

❑ 一种真实硬件电路运行时的调试工具

- 精细控制custom_cpu真实硬件逻辑电路的运行和暂停， 快速定位出错点
- 观测custom_cpu真实硬件逻辑电路的全部逻辑信号，并借助软件仿真工具恢复波形

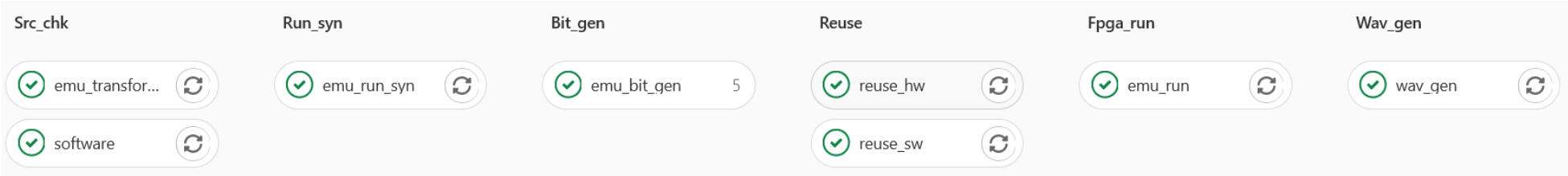


获取真实硬件电路的全部逻辑信号并重建波形

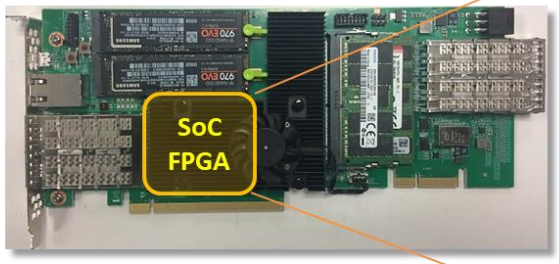


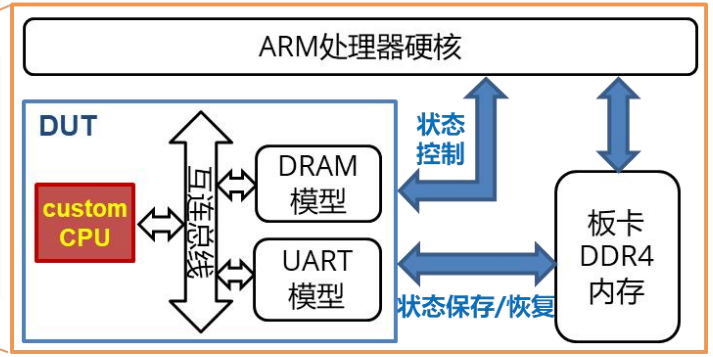
在fpga_eval出错时，可使用FPGA仿真加速
快速暴露custom_cpu逻辑电路及代码的设计缺陷

SERVE平台提供的方法：云原生的FPGA仿真加速框架

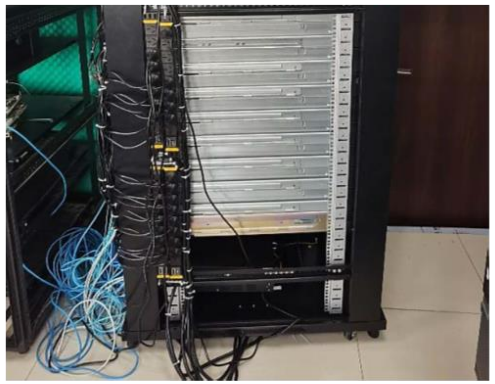


SERVE平台FPGA节点

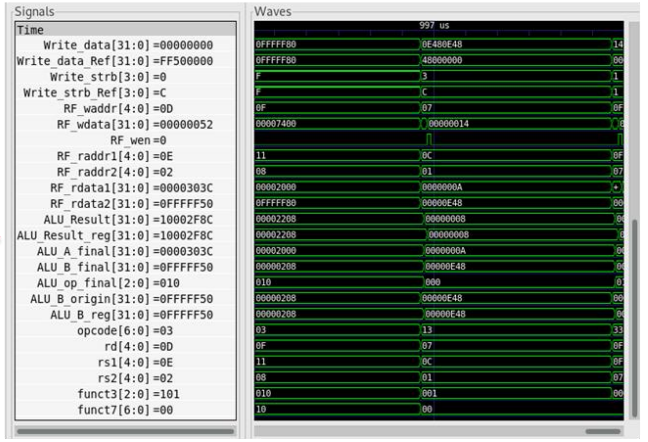




自动抓取custom_cpu
运行时的电路状态



恢复
并保存
波形



SERVE平台服务器定制仿真工具软件

3.1 实验课的仿真加速流程配置文件

❑ 实验流程修改（lab_env.yml）

- 目标设计为custom_cpu_emu
  - 软硬件框架与custom_cpu相同
- 仿真DUT类型为multi_cycle
- CPU_ISA在实验4中设置为riscv32


```yaml
variables:
  # TARGET_DESIGN can be "example", "alu", "reg_file", "simple_cpu", "custom_cpu"...
  TARGET_DESIGN: "example"

  # CPU_ISA must be either "mips" or "riscv32"
  CPU_ISA: "mips"

  SIM_DUT_TYPE: "single_cycle"
```

引号中内容替换为custom_cpu_emu

引号中内容替换为riscv32

引号中内容替换为multi_cycle

❑ 仿真加速配置（emu_settings.yml）

- 指定benchmark组名与名称
  - 每次只能调试一个测试用例
- 指定保存波形的时钟周期区间（§3.2.1或§3.2.2两种方法二选一）


```yaml
variables:
  ### Custom CPU Emulation Flow (TARGET_DESIGN is "custom_cpu_emu") ###

  # Specify benchmark suite & name to run
  EMU_BENCH_SUITE: "microbench"
  EMU_BENCH_NAME: "15pz"

  # Specify whether a manual replay is enabled
  # A replayed waveform file will be generated:
  #   for EMU_REPLAY_WINDOW cycles starting from EMU_MANUAL_REPLAY_BEGIN, if set to "y
  #   for the last EMU_REPLAY_WINDOW cycles before an activated trigger or timeout, if
  EMU_MANUAL_REPLAY_ENABLE: "no"

  # Specify the beginning cycle of a manual replay window
  EMU_MANUAL_REPLAY_BEGIN: 0

  # Specify the count of last cycles to replay
  EMU_REPLAY_WINDOW: 10000

  # Timeout (in cycles)
  EMU_TIMEOUT: 100000000000
```

修改为待调试的benchmark组名与名称

波形时钟周期区间参数

超时周期数（一般无需更改）

3.2.1 自动确定保存波形的时钟周期区间

- EMU_MANUAL_REPLAY_ENABLE设置为no（emu_settings.yml，右图蓝色横线处）
- 检查custom_cpu的执行结果，以出错时刻T为区间终点
  - 比对custom_cpu内存访问端口信号
- 指定区间长度（ EMU_REPLAY_WINDOW 右图橙色横线处）
  - 默认为10000个周期，可按需修改
  - 注意：此变量为整数值，前后没有引号

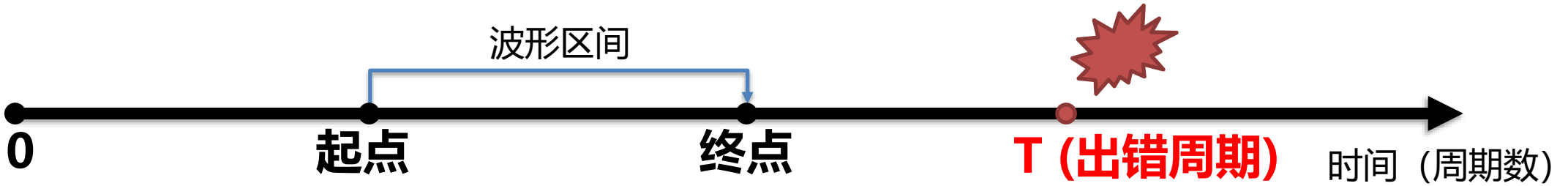
```
variables:
  ### Custom CPU Emulation Flow (TARGET_DESIGN is "custom_c

  # Specify benchmark suite & name to run
  EMU_BENCH_SUITE: "microbench"
  EMU_BENCH_NAME: "15pz"

  # Specify whether a manual replay is enabled
  # A replayed waveform file will be generated:
  #   for EMU_REPLAY_WINDOW cycles starting from EMU_MANUAL
  #   for the last EMU_REPLAY_WINDOW cycles before an activ
  EMU_MANUAL_REPLAY_ENABLE: "no"

  # Specify the beginning cycle of a manual replay window
  EMU_MANUAL_REPLAY_BEGIN: 0

  # Specify the count of last cycles to replay
  EMU_REPLAY_WINDOW: 10000

  # Timeout (in cycles)
  EMU_TIMEOUT: 100000000000
```

3.2.2 手动指定保存波形的时钟周期区间

- EMU_MANUAL_REPLAY_ENABLE设置为yes（emu_settings.yml，右图蓝色横线处）
- 指定保存波形的起点时钟周期（EMU_MANUAL_REPLAY_BEGIN，右图绿色横线处）
  - 可查看距出错点较远位置的波形
  - 注意：此变量为整数值，前后没有引号
- 指定区间长度（ EMU_REPLAY_WINDOW 右图橙色横线处）
  - 默认为10000个周期，可按需修改
  - 注意：此变量为整数值，前后没有引号

```
1  variables:
2    ### Custom CPU Emulation Flow (TARGET_DESIGN is "custom_c
3
4    # Specify benchmark suite & name to run
5    EMU_BENCH_SUITE: "microbench"
6    EMU_BENCH_NAME: "15pz"
7
8    # Specify whether a manual replay is enabled
9    # A replayed waveform file will be generated:
10   #   for EMU_REPLAY_WINDOW cycles starting from EMU_MANUAL
11   #   for the last EMU_REPLAY_WINDOW cycles before an activ
12   EMU_MANUAL_REPLAY_ENABLE: "no"
13
14   # Specify the beginning cycle of a manual replay window
15   EMU_MANUAL_REPLAY_BEGIN: 0
16
17   # Specify the count of last cycles to replay
18   EMU_REPLAY_WINDOW: 10000
19
20   # Timeout (in cycles)
21   EMU_TIMEOUT: 100000000000
```

改为"yes"



3.3 推送代码触发仿真加速流程

- 修改配置文件（ lab_env.yml 和emu_settings.yml ）后，提交并推送到GitLab远程仓库，触发自动化流程（如下图）
  - git add和git commit等命令不再赘述
- 无论benchmark是否运行成功，所有步骤都会正常结束
  - **需通过仿真加速输出结果和波形确定custom_cpu中的设计缺陷**

3.3.1 运行FPGA仿真加速（emu_run）

```
26 RUNNER_CNT = 4
27 Completed FPGA configuration
28 [firewall] status: 0x0
29 [firewall] waiting for firewall to be idle ...
30 [firewall] unblock firewall ok
31 [EMU] Memory region: 4a00000000 - 4a10000000 (emu_top.u_rammodel.host_axi)
32 [EMU] Memory region: 4a10000000 - 4a10010000 (emu_top.u_rammodel.lsu_axi)
33 [EMU] Memory region: 4a10010000 - 4a10020000 (scanchain)
34 [EMU] Run from initial state
35 [EMU] Initialize emu_top.u_rammodel.host_axi with file software/workload/ucas-cod/benchmark/simple_test/microbench/riscv32/bin/bf.bin
36 [EMU] Cycle 0: reset=1
37 [EMU] Cycle 0: start emulation
38 [EMU] Cycle 0: save checkpoint begin
39 [EMU] Cycle 0: save checkpoint end
40 [EMU] Cycle 10: reset=0
41 [bf] Brainf**k interpreter: [EMU] Cycle 28344150: user trigger 0 activated at previous cycle
42 [EMU] Rewind to cycle 28244150
43 [EMU] Cycle 0: reset=1
44 [EMU] Cycle 0: load checkpoint begin
45 [EMU] Cycle 0: load checkpoint end
46 [EMU] Cycle 0: start emulation
47 [EMU] Cycle 10: reset=0
48 [bf] Brainf**k interpreter: [EMU] Cycle 28244150: stop emulation
49 [firewall] status: 0x0
```

custom_cpu运行的测试负载从UART打印内容

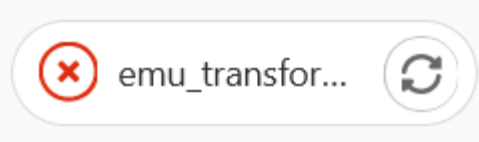
custom_cpu因设计缺陷停止运行

仿真运行过程

第二遍运行
定位到区间起点
（仅自动模式下）

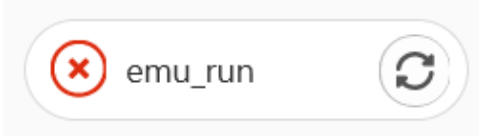
3.3.2 FPGA仿真加速流程中可能出现的错误

- emu_transform（插入控制电路）步骤出现“Asynchronous logic not supported”错误
  - **同学们的Verilog HDL代码有设计缺陷，**请检查对应变量是否产生锁存器或使用异步复位




```
3751  Checking module alu...
3752  Checking module custom_cpu...
3753  === ERROR Traceback ===
3754  Object declared at fpga/design/ucas-cod/hardware/emu/scripts/../../sources/custom_cpu/riscv32/custom_cpu.v:84.14-84.24
3755  ERROR: Asynchronous logic not supported: custom_cpu.next_state
3756  make: *** [fpga/design/ucas-cod/scripts/hardware.mk:42: emu_transform] Error 1
```

- emu_run（运行FPGA仿真加速）步骤出现“Bus error”
  - 可能为平台的偶发性错误，请重试（Retry）该步骤



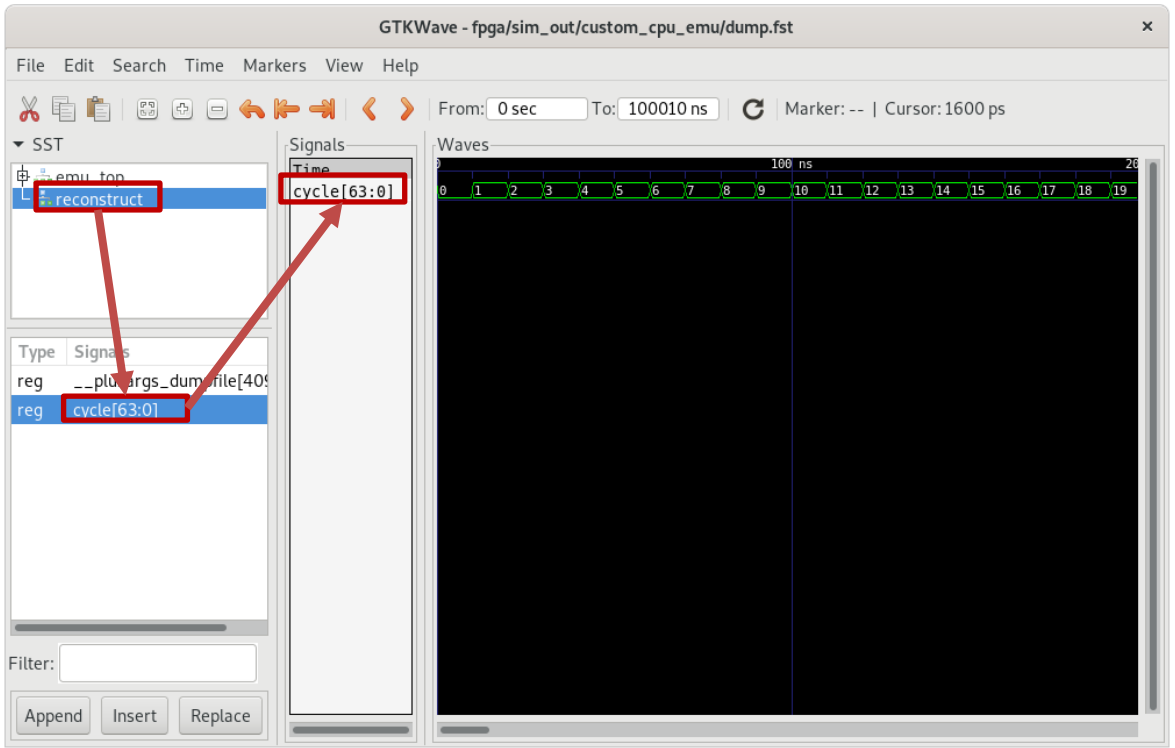

```
26  RUNNER_CNT = 1
27  Completed FPGA configuration
28  [firewall] status: 0x0
29  [firewall] waiting for firewall to be idle ...
30  [firewall] unblock firewall ok
31  [EMU] Memory region: 4880000000 - 4890000000 (emu_top.u_rammodel.host_axi)
32  [EMU] Memory region: 4890000000 - 4890010000 (emu_top.u_rammodel.lsu_axi)
33  [EMU] Memory region: 4890010000 - 4890020000 (scanchain)
34  ./fpga/design/ucas-cod/run/emu/fpga_run.sh: line 108: 29918 Bus error               python3 -m monitor --initmem emu_to
    _WINDOW --dump $EMU_DUMP_PATH $EMU_CONFIG $EMU_CKPT_PATH
35  [firewall] status: 0x1
```

3.4* 查看反映custom_cpu设计缺陷的波形

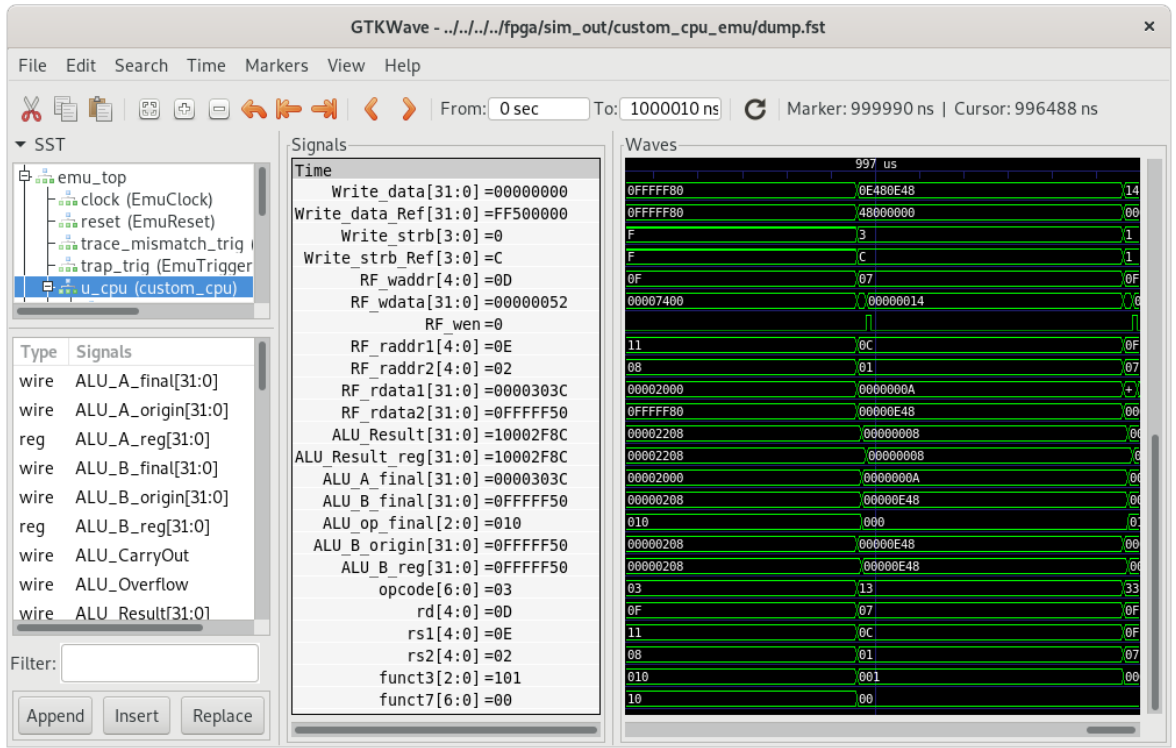
- 在虚拟机中执行如下命令：

```
make FPGA_PRJ=ucas-cod FPGA_BD=nf SIM_TARGET=custom_cpu_emu wav_chk
```

- 注意：无需关注波形上方的时间（Time），需添加周期数信号（cycle）以确定当前的仿真周期



添加周期数信号（位于reconstruct模块）



添加CPU内信号

3.6 调试结束返回正常流程

- 修改脚本文件

```
cd ~/COD-Lab && vim lab_env.yml
```

- 将TARGET_DESIGN后边

  引号中的内容替换为custom_cpu（全部为小写字母）

- 确保CPU_ISA设置为riscv32（全部为小写字母）

- 确保SIM_DUT_TYPE后边引号中的内容为multi_cycle（全部为小写字母）

- 提交修改到代码仓库，并在commit message中描述遇到的bug（具体问题、多少个cycle出现）

```
cd ~/COD-Lab && git add lab_env.yml && git commit -m “lab_env: return from emu flow” -m “bug进行说明”
```

- 最终评分以正常流程的Fpga_eval执行情况作为依据

```
variables:
  # TARGET_DESIGN can be "example", "alu", "reg
  TARGET_DESIGN: "simple_cpu"
  # CPU_ISA must be either "mips" or "riscv32"
  CPU_ISA: "mips"

  # single_cycle or multi_cycle
  SIM_DUT_TYPE: "multi_cycle"
```

设置为 custom_cpu

设置为riscv32

确认此处为 multi_cycle

FPGA仿真加速小结

- 当fpga_eval测试运行出现错误时，定位硬件电路设计缺陷的调试工具
- 通过硬件信号比对、波形抓取保存、波形文件恢复等手段，快速展示出现设计缺陷的电路行为
- 同学们需根据波形文件，修改对应的Verilog HDL代码
- 最终目标：测试修改后的custom_cpu Verilog HDL代码在fpga_eval阶段是否可以正确执行所有的测试用例

❑提交及验收说明

1* 在个人仓库中添加标签（十分重要）

- 在完成所有设计和云上仿真、FPGA测试流程后，需在仓库中添加标签
- 便于助教老师后续进行代码评分
- 请执行如下命令（需要严格一致，否则影响成绩评分）

```
cd ~/COD-Lab && git tag -a custom_cpu-riscv32_multi_cycle -m “Release RISC-V multi_cycle custom CPU design”

cd ~/COD-Lab && git push origin master --tags
```

2* 实验报告撰写要求

- 描述定制RISC-V处理器的设计情况
  - 基本要求与实验项目3一致
  - 使用FPGA仿真加速（如有）解决的bug情况及各bug的详细描述
  - RISC-V指令集与MIPS指令集的对比
- 实验报告大小尽量控制在5MB以内
- 实验报告请命名为“**prj4.pdf**”

3* 实验报告提交及远程推送方法

- **cd ~/COD-Lab && mkdir -p reports**
- **将prj4.pdf这个文件拷贝到~/COD-Lab/reports**
  - 本地虚拟机：通过图形界面拷贝
  - 云端虚拟机：通过SFTP上传（具体需查看自己使用terminal终端的SFTP使用方法）
- **提交到个人本地仓库**

  cd ~/COD-Lab && git add reports/ && git commit -m “docs: add prj4 report”
- **推送到个人远程仓库**

  cd ~/COD-Lab && git push origin master
- **请确保提交报告前，实验编写的Verilog HDL代码已完成云端测试**

4. 课堂验收要求（5月13日开始）

1. **实验态度端正，完成代码编写和上板测试，可得基准成绩50分**
   - 查看SERVE平台上五组benchmark的运行结果
2. **其他检查要点 （满分40分）**
   - 状态机编码用one-hot，并按照三段式正确表述（+10分）
   - 除状态机第二段外的组合逻辑必须用assign语句描述（+10分）
   - RISC-V/MIPS指令集性能分析对比（+20分）
3. **同学对代码及实验内容的思考（满分10分，助教根据沟通情况打分）**
4. **同学需根据助教建议修改代码，并在实验报告中进行说明。助教会根据课堂验收记录，检查最终提交代码修改情况**
5. **如果明显抄袭 0分记录（对代码完全说不清楚的，疑似抄袭）**

5*. 填写在线问卷调查

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