B62007Y 2017-2018学年春季学期

计算机组成原理实验

实验项目4 性能计数器设计

常轶松 王诲喆

2018年6月1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

内容大纲

- 实验内容简介及要求
- 实验要点讲解
- 实验操作流程
- 注意事项

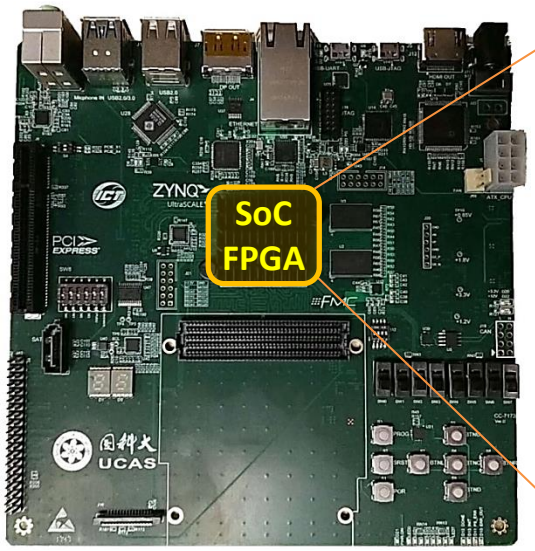
实验目的

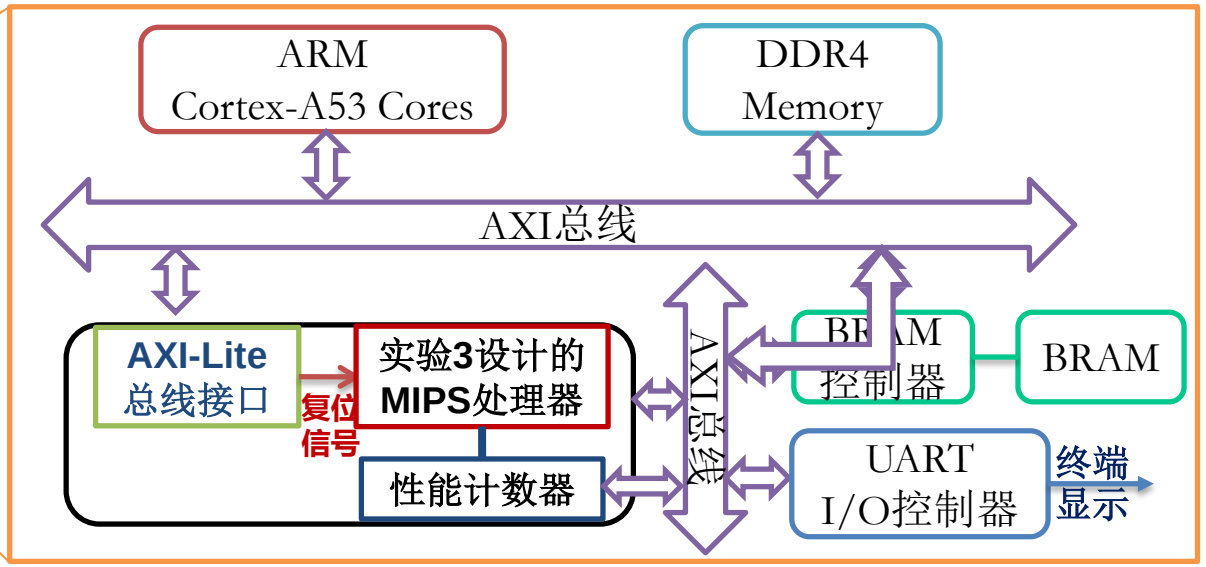
❑ 基于实验项目3改进的MIPS处理器

- 添加性能计数器
  - 统计处理器运行周期、完成执行的指令数、访存指令数、访存延时、跳转发生/不发生指令数等性能指标

- 利用程序计数器对实验项目提供的microbench进行性能评测
  - 修改microbench源码，在核心算法执行完成后访问性能计数器
  - 基于实验项目3的终端打印软件栈，打印microbench的性能统计结果

实验环境及工程框架





- **MIPS处理器访问大容量（1GB）真实DDR（DRAM）内存**
  - **已在顶层框架做好，同学们无需改动MIPS处理器RTL代码即可访问DDR**
- **添加性能计数器逻辑**
  - **MIPS处理器按照I/O外设访问方式读取性能计数器内容**

实验项目发布与接收流程（1）

- **实验项目3参与邀请链接URL**
  - **https://classroom.github.com/a/PHVgRjuR**
  - 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

实验项目发布与接收流程（2）

- **个人作业远程仓库**
  - prj4-YOUR_GitHub_USERNAME
    - 例如：某学生GitHub用户名为zhang_abc，其实验项目4的repo名即为prj4-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/prj4-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/prj4-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prj4-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、软-硬件协同仿真、bitstream生成、使用远程和本地FPGA板卡等操作均在本地仓库中完成

实验项目发布与接收流程（3）

- 在实验项目4的个人本地仓库中添加本次实验的原始框架仓库地址
  - git remote add upstream https://github.com/ucas-cod/prj4-student
- 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原始框架中的代码脚本进行更新
- 框架发布更新后，助教会及时发布通知，并请需要同学们在自己的实验项目4本地仓库中进行同步
  - 第一步，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
  - 第二步，框架同步
    - git pull upstream master

实验项目进度安排

❑ 课堂验收

- 截止日：2018年06月15日20:49:59
- 完成性能计数器设计，并基于microbench进行性能评测

❑ 验收及提交要求

- 全部RTL源码、修改的microbench软件程序源码、运行日志文件及实验报告
- 截止日：2018年06月22日18:09:59

内容大纲

- 实验内容简介及要求
- 实验要点讲解
  - 性能计数器
  - microbench软件代码修改
- 实验操作流程
- 注意事项

性能计数器（Performance Counter）

- 处理器内部硬件实现的特殊功能寄存器（计数器）
  - 统计指令执行的硬件行为
    - 指令执行、内存访问、缓存（Cache）性能等
  - 这些计数器一般是只读的，仅在复位时清0
  - 用于CPU或计算机系统的底层性能在线统计、分析和调优
  - https://en.wikipedia.org/wiki/Hardware_performance_counter

具体实现

- **基于实验项目3的处理器添加Performance Counter**
  - 已在mips_cpu.v模块顶层预留16个32-bit性能计数器接口，请同学们自行添加对应计数器功能
  - 同学们可实现对应的性能计数器核心逻辑，并使用assign语句将计数器输出连接到对应接口
    - 例如：将mips_perf_cnt_0作为周期计数器

```
reg [31:0] cycle_cnt;
always @ (posedge clk)
begin
        if (rst == 1'b1)
                cycle_cnt <= 32'd0;

        else
                cycle_cnt <= cycle_cnt + 32'd1;
end
assign mips_perf_cnt_0 = cycle_cnt;
```

  - 可根据实际情况选择要实现的性能计数器个数，无需将16个计数器预留接口全部用满
    - 但至少需要实现一个周期计数器

```
module mips_cpu(
        input  rst,
        input  clk,

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

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

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

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

        output [31:0]       mips_perf_cnt_0,
        output [31:0]       mips_perf_cnt_1,
        output [31:0]       mips_perf_cnt_2,
        output [31:0]       mips_perf_cnt_3,
        output [31:0]       mips_perf_cnt_4,
        output [31:0]       mips_perf_cnt_5,
        output [31:0]       mips_perf_cnt_6,
        output [31:0]       mips_perf_cnt_7,
        output [31:0]       mips_perf_cnt_8,
        output [31:0]       mips_perf_cnt_9,
        output [31:0]       mips_perf_cnt_10,
        output [31:0]       mips_perf_cnt_11,
        output [31:0]       mips_perf_cnt_12,
        output [31:0]       mips_perf_cnt_13,
        output [31:0]       mips_perf_cnt_14,
        output [31:0]       mips_perf_cnt_15
);
```

MIPS处理器如何访问性能计数器

- **本次实验项目将性能计数器作为I/O外设，允许MIPS处理器使用Load-Store指令访问**
  - 各性能计数器的物理地址如右图

- **上述方式与MIPS处理器架构规范要求不同，仅为了让同学们通过较为简单的方式，理解性能计数器的功能和用途**
  - 如对MIPS架构规范给出的性能计数器访问方式感兴趣，请参考：
    MIPS32 Architecture For Programmers
    Volume III: The MIPS32™ Privileged Resource Architecture

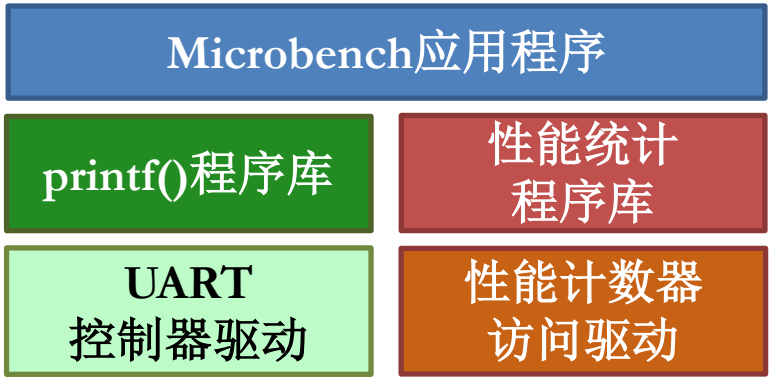
计数器名称	MIPS可见物理地址
mips_perf_cnt_0	0x4002_0000
mips_perf_cnt_1	0x4002_0008
mips_perf_cnt_2	0x4002_1000
mips_perf_cnt_3	0x4002_1008
mips_perf_cnt_4	0x4002_2000
mips_perf_cnt_5	0x4002_2008
mips_perf_cnt_6	0x4002_3000
mips_perf_cnt_7	0x4002_3008
mips_perf_cnt_8	0x4002_4000
mips_perf_cnt_9	0x4002_4008
mips_perf_cnt_10	0x4002_5000
mips_perf_cnt_11	0x4002_5008
mips_perf_cnt_12	0x4002_6000
mips_perf_cnt_13	0x4002_6008
mips_perf_cnt_14	0x4002_7000
mips_perf_cnt_15	0x4002_7008

内容大纲

- 实验内容简介及要求
- 实验要点讲解
  - 性能计数器
  - microbench软件代码修改
- 实验操作流程
- 注意事项

microbench软件栈

- 全部软件源码位于benchmark/microbench目录下
  - common子目录中的printf.c
    - 实验项目要修改的源码文件，包含printf()函数库和UART控制器驱动
    - 可复用实验项目3中已修改的源码
    - 注意：UART控制器的物理地址调整为0x40010000（与实验项目3不同）
  - src子目录
    - bench.c：microbench主程序，同时包含性能计数器访问接口，实验项目需修改该文件，添加性能计数器访问功能
    - 其他子目录：核心算法程序



microbench运行时间统计（bench.c）

- 同学们在硬件设计阶段需要至少实现一个运行周期计数器，用于在microbench统计算法执行周期
- 算法执行周期统计
  - 算法开始运行前，调用bench_prepare()函数，获取计数器初值，并保存在Result结构体对象的msec变量中
  - 算法运行结束后，调用bench_done()函数，获取计数器当前值，与计数器初值相减，即可得到执行周期数，并保存在msec变量中
- 计数器访问函数_uptime()
  - 需要同学们进行修改，访问硬件上的周期计数器
- 执行周期结果显示
  - 在main()的末尾处添加周期统计结果打印（使用printk()函数，其参数格式与printf()一致）

```
typedef struct Result {
  int pass;
  unsigned long msec;
} Result;

unsigned long _uptime() {
  // TODO [COD]
  //   You can use this function to access
  return 0;
}

static void bench_prepare(Result *res) {
  // TODO [COD]
  //   Add preprocess code, record performa
  //   You can communicate between bench_pr
  //   static variables or add additional f
  res->msec = _uptime();
}

static void bench_done(Result *res) {
  // TODO [COD]
  //  Add postprocess code, record performa
  res->msec = _uptime() - res->msec;
}
```

```
int main() {
  int pass = 1;

  _Static_assert(ARR_SIZE(benchmarks) > 0, "non benchmark");

  for (int i = 0; i < ARR_SIZE(benchmarks); i ++) {
    Benchmark *bench = &benchmarks[i];
    current = bench;
    setting = &bench->settings[SETTING];
    const char *msg = bench_check(bench);
    printk("[%s] %s: ", bench->name, bench->desc);
    if (msg != NULL) {
      printk("Ignored %s\n", msg);
    } else {
      unsigned long msec = ULONG_MAX;
      int succ = 1;
      for (int i = 0; i < REPEAT; i ++) {
        Result res;
        run_once(bench, &res);
        printk(res.pass ? "*" : "X");
        succ &= res.pass;
        if (res.msec < msec) msec = res.msec;
      }

      if (succ) printk(" Passed.\n");
      else printk(" Failed.\n");

      pass &= succ;

      // TODO [COD]
      //   A benchmark is finished here, you can use printk to output some informantion.
      //   `msec' is intended indicate the time (or cycle),
      //   you can ignore according to your performance counters semantics.
    }
  }
```

在代码中访问其他性能计数器

- **Step #1：修改Result结构体定义，添加统计变量**
  - 例如：添加unsigned long mem_cycle，统计内存访问周期
- **Step #2：参考_uptime()，添加其他性能计数器的访问函数**
  - 例如：添加_read_mem_cycle()函数
- **Step #3：在bench_prepare()和bench_done()两个函数中，添加对计数器访问函数（如_read_mem_cycle()函数）的调用**
- **Step #4：在main()函数结尾，添加相应的结果显示打印功能**

内容大纲

- 实验内容简介及要求
- 实验要点讲解
- 实验操作流程
  - 硬件实验流程
  - 软件编译流程
  - 上板运行流程
  - 上板调试流程
- 注意事项

RTL代码编写

- **请在本地仓库hardware/sources/ip_catalog/mips_core目录下实现处理器代码**
  - 初始目录下仅放置顶层模块mips_cpu.v，该模块包含了支持多周期访存通路MIPS处理器的完整输入输出端口列表（如第11页所述）
    - 注意：顶层的输入输出端口信号位宽及命名不允许修改，但可修改信号类型
  - 请同学们将自己实验项目3中已设计好的单周期MIPS处理器所有源码拷贝到上述目录，并开始修改

硬件操作命令

- **本次实验项目使用的Vivado硬件流程命令与实验项目3基本一致，在此不再赘述。请参考本次实验项目仓库中的README**
  - 注意：DDR内存无法在仿真环境提供，所以功能仿真和时序仿真只能使用BRAM作为MIPS访问的真实内存
    - 仿真阶段可使用实验项目3中使用的30个benchmark进行测试，并通过波形判断性能计数器硬件逻辑是否工作正常
    - 本次实验项目提供的microbench不能用于仿真
- **讲义重点介绍如何添加硬件信号探针（probing）方法**
  - 可根据实验需要选择执行，不是必须完成的操作
  - 添加信号探针会占用逻辑资源并改变逻辑电路结构
    - 需要重新综合并生成bitstream
  - 如何使用硬件探针观察硬件行为的方法将在后边“上板调试方法”中介绍

硬件信号探针添加（1）

- **在个人本地仓库hardware/scripts/dbg目录下编辑要探测的硬件信号列表（例如：编辑mips_cpu_dbg.cfg ）**
  - 信号列表必须以.cfg为扩展名，且必须放在hardware/scripts/dbg目录下
  - 信号列表格式（如下图）
    - 第一行设置采样深度，有效值是1024，2048，4096或8192
    - 第二行是要探测的信号个数，1-256
    - 第三行开始的每一行是要探测的信号名及信号位宽（二者中间有空格），行数要和第二行给定的信号数目一致
      - 添加的信号中至少要有一个作为探针采样触发条件的信号
      - 信号名与信号位宽必须与RTL模块定义一致
      - 如信号位宽为1，可默认不写，仅在该行保留信号名即可

```
<ila_sampling_depth>
<probing_signal_num>
<probe_signal_0_name> <probe_signal_0_bit_width>
<probe_signal_1_name> <probe_signal_1_bit_width>
<probe_signal_2_name> <probe_signal_2_bit_width>
...
```

  - 注意：添加的信号必须来自同一个RTL模块，且这个模块必须有时钟。如需探测来自不同模块的硬件信号，需编写多个不同的.cfg文件

硬件信号探针添加（2）

❑ 在指定模块中实例化探针模块

- 例如：假设我们编辑了mips_cpu_dbg.cfg信号列表，且添加的信号来自mips_cpu.v
  - 脚本会在执行bit_gen时，自动创建一个名为mips_cpu_dbg的模块
    - 该模块包含一个时钟端口（端口名为ila_clk）和多个探针端口
    - 探针端口的数量及顺序与mips_cpu_dbg.cfg信号列表中的定义一致
    - 每个探针端口的名称及位宽与mips_cpu_dbg.cfg中的对应行的定义一致
  - 需要在执行bit_gen之前，在mips_cpu.v中实例化mips_cpu_dbg模块
    - 时钟端口连接被探测模块的时钟（如本例即使用mips_cpu.v模块的时钟）
    - 将被探测的信号依次与mips_cpu_dbg的探针端口连接

硬件信号探针添加（3）

❑ 生成带探针的bitstream配置文件

- 在本地仓库目录中执行
  make HW_ACT=bit_gen HW_VAL="<探针信号文件列表>" vivado_prj
  - 其中探针信号文件列表为hardware/scripts/dbg目录下已存在且在对应RTL模块内已实例化探针模块的.cfg文件名（不包括.cfg扩展名）
  - 例如：如果想在硬件生成时观察使用mips_cpu_dbg.cfg中列出的信号，可执行make HW_ACT=bit_gen HW_VAL="mips_cpu_dbg" vivado_prj
  - bit_gen执行完毕，会在仓库顶层的hw_plat目录中生成.bit文件（已包含硬件探针）和debug_nets.ltx文件（Vivado硬件调试信号列表）

内容大纲

- 实验内容简介及要求
- 实验要点讲解
- 实验操作流程
  - 硬件实验流程
  - 软件编译流程
  - 上板运行流程
  - 上板调试流程
- 注意事项

Microbench软件程序编译

❑ 软件程序交叉编译

- 在个人本地仓库中执行make microbench进行编译，生成的可执行文件位于benchmark/microbench/bin下
- 如编译后再次修改C语言源码文件，需重新执行上述命令，重新生成可执行程序
- 执行make microbench_clean，将删除编译生成的二进制可执行文件

内容大纲

- 实验内容简介及要求
- 实验要点讲解
- 实验操作流程
  - 硬件实验流程
  - 软件编译流程
  - 上板运行流程
  - 上板调试流程
- 注意事项

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> HW_VAL="<benchmark列表>" local_run

  连接ZyForce本地板卡或

  make USER=<user_name> HW_VAL="<benchmark列表>" cloud_run

  连接ZyForce云端远程板卡

  - 注意：上述命令中的引号不能去掉
  - <benchmark列表> = <benchmark组名>[:<benchmark组内序列号列表>]
    - []内的内容是可选项，不是必须填写。[]在实际命令中不出现
  - <benchmark组名>：basic/medium/advanced/microbench四选一
  - 如果<benchmark组内序列号列表>为空，表示要运行某一组内的全部测试程序
    - 例如：make BOARD_IP=192.168.100.20 HW_VAL="medium" local_run
  - 如果<benchmark组内序列号列表>不为空，请在冒号后添加要运行的测试程序序列号（可按任意顺序且可重复），序列号之间用空格分隔
    - 例如：make USER=zhangsan HW_VAL="advanced:01 09 05 12" cloud_run
    - 组内序列号有效值：basic组01；medium组01-12；advanced组01-17；microbench组01-09
    - 注意：不要忘记组名和序号列表之前的冒号（英文符号）
    - 注意：个位数的序列号前请务必加0（如序列号5务必输入为05）

内容大纲

- 实验内容简介及要求
- 实验要点讲解
- 实验操作流程
  - 硬件实验流程
  - 软件编译流程
  - 上板运行流程
  - 上板调试流程
- 注意事项

上板调试运行（1）

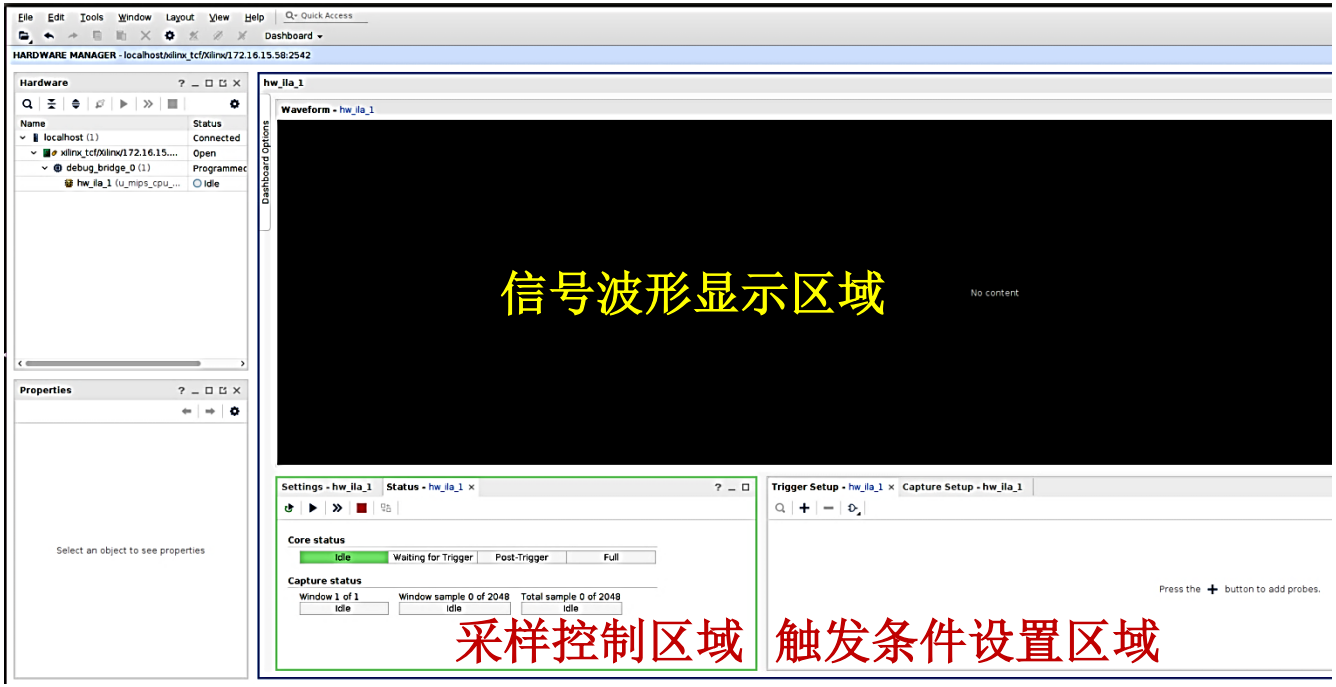
❑ 开启调试运行

- 在本地仓库目录中执行

  make USER=xxx HW_VAL=basic:01 HW_DBG=y cloud_run

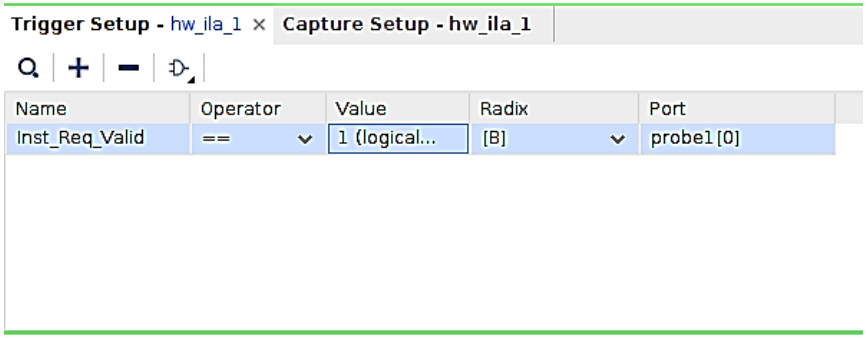
  – 一般情况下上板调试运行一个benchmark即可

  – 执行上述命令后，会在虚拟机中自动打开Vivado图形界面，并进入硬件调试窗口

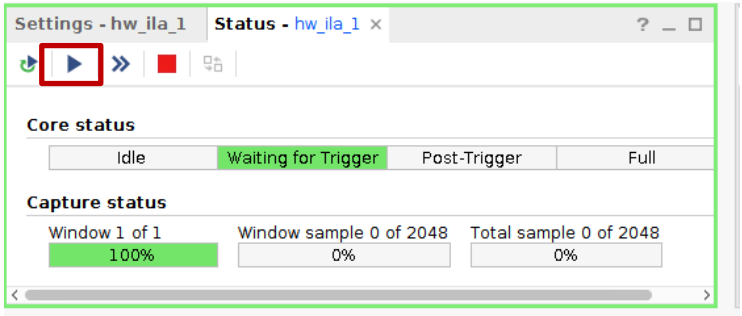


上板调试运行（2）

- **设置信号采样触发条件并开启采样**
  - 点击触发条件设置区域的“+”按钮，添加触发条件
    - 在触发条件满足时，所有硬件探针才开始连续采样硬件信号值
    - 可设置触发条件为触发信号==或!=某一逻辑值（如下图所示）



  - 点击采样控制区域的三角形按钮，开启探针采样，点击后会看到硬件探针的状态切换为“Waiting for trigger”（如下图所示）

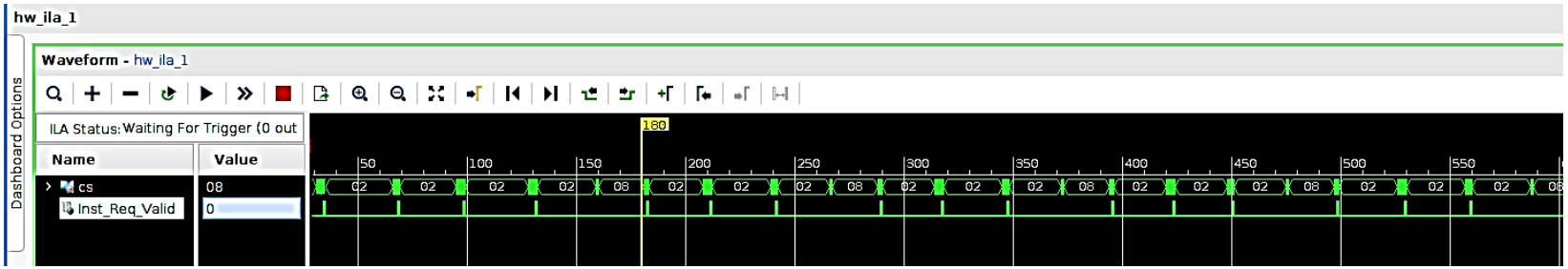


上板调试运行（3）

- 启动软件运行
  - 回到终端窗口，可以看到MIPS端的软件还未加载运行
  - 输入y并回车，将启动软件加载执行

```
cod@cod-VirtualBox:~/ucas-cod/prj4-teacher$ make USER=changyisong HW_VAL=basic:01 HW_DBG=y cloud_run
Starting xl2tpd (via systemctl): xl2tpd.service.
Remote target: root@172.16.15.58
Try to reboot 172.16.15.58
Warning: Permanently added '172.16.15.58' (ECDSA) to the list of known hosts.
Connection to 172.16.15.58 closed by remote host.
Waiting for target reboot...
Completed FPGA configuration
/home/cod/ucas-cod/prj4-teacher/run
Evaluating basic benchmark suite...
If Vivado preparition is finished, please press y to continue...
y
```

- 如触发条件满足，即可看到对应的信号波形图
  - 可在MIPS软件程序运行完成后，继续观察波形



上板调试运行（4）

❑ 调试退出

- 手动关闭Vivado图形界面，并等待cloud_run命令行自动退出

❑ 补充说明

- 如为多个模块添加硬件探针，在Vivado调试界面中都会出现各个模块的探针控制及波形显示界面
  - 注意：各模块使用的探针需独立设置触发信号与触发条件，不能混用
- 如一次调试没有找到问题，就需要重新添加被探测硬件信号、修改探针模块实例化、生成bitstream、上板调试
- 实验项目3还没完全调通的同学，可基于实验项目4的整体框架进行硬件Debug
- 更多硬件探针使用细节请参考Xilinx UG908的第11章（https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug908-vivado-programming-debugging.pdf）

内容大纲

- 实验内容简介及要求
- 实验要点讲解
- 实验操作流程
- 注意事项

实验项目4完整提交内容

❑ 提交前请务必检查

- 使用最后提交的MIPS处理器RTL设计，基于云环境或本地板卡，完成benchmark性能评测，并将评测结果进行终端打印

❑ 实验报告

- 请在实验项目4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-prj4.pdf”，例如：
  - 2015K8009929000-prj4.pdf
- PDF文件大小尽量控制在5MB以内
- 实验报告内容中不必详细描述实验过程, 但我们鼓励你在报告中描述如下内容:
  - 你遇到的问题和对这些问题的思考
  - 你的其它想法, 例如实验心得, 对提供帮助的同学的感谢等

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