龙芯杯个人赛经验分享

“一生一芯”分享会

杨骐嘉

2024-08-25

ysyx

Outline

Outline

1.1 个人赛是什么

每年可能不太一样，请参考 www.nscscc.com 官方发布的技术方案为准

- 赛程而言，包括初赛和决赛；
- 空间而言，包括线上和线下；
- 成绩而言，包括基准测试程序运行成绩(70%)和现场编程任务成绩(30%)；
  - 前者：基于在线竞赛平台，在截止时间前，提交 Vivado 比特流文件，在线计时测评
  - 后者：线下现场发布题目，限时编程解题，可选修改硬件设计
- 赛道而言，分为 LoongArch 指令集和 MIPS 指令集两个赛道；
- 获奖名额而言，各赛道一等奖一位，二等奖二位，三等奖若干；

1.2 今年的情况

~~若有出入，请以官方统计为准~~

初赛

仅仅通过所有三级功能测试和性能测试已经不够了，需要达到一定性能才能进入决赛（e.g. 有一位 50MHz 流水线+ICache 的选手最终未进入决赛）

决赛

性能测试而言，延续去年路线，频率很重要，今年最高的已经超到了 267MHz (LA 赛道 Rk1：单发射，性能测试 0.222s；MIPS 赛道 Rk1：双发射，180MHz，性能测试 0.198s)

现场答题而言，延续变简单的趋势，比去年的题目更简单，但是竞争同样激烈。~~出现了 0.000s，逃。~~

Outline

1. Intro
   - 个人赛是什么
   - 今年的情况
2. Difftest
   - 痛点与需求分析
   - 基于统一抽象模型的 difftest
   - 细粒度、基于事件跟踪的 difftest
   - 在线测试平台的仿真细节
3. Tricks
   - 超频
   - 多周期访存
   - 协处理器

2.1 痛点与需求分析

现有的 Debug 方案存在的问题：

方法	查错效率	速度	完整性
Vivado 仿真+手工对波形	低	慢	较完整
CPU 设计实战的 gettrace	较高	较快	?
Vivado ILA 远程调试	极低	极慢	完整

同时，测试程序数量多（三个等级测试+三个性能测试程序+监控程序）、与外设交互复杂（在线测试平台通过串口输入命令，进行程序的烧录、计时、验证等等）。

并且，决赛现场时间短、压力大，CPU 有一定概率会翻车，需要更快、更准的调试框架。

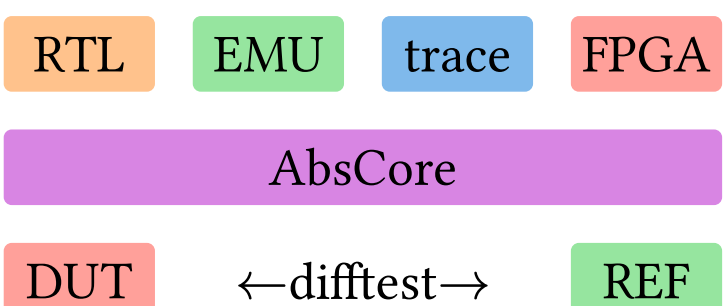
而且，最好能够与一生一芯项目(RISC-V)的 difftest 系统复用，要求具有较强的复用性。

需要更快、更准、能够在多种微架构、多个工程之间进行复用的 **difftest** 框架

2.2 基于统一抽象模型的 difftest

RTL、模拟器、trace log、FPGA 通过抽象都视为 Core，自由作为 ref 或者 dut，运行 difftest

- 每新接入一个 Core，只实现 AbsCore 虚基类所需要的虚函数接口并继承，即可完成接入
- difftest 主函数的实现只与 AbsCore 基类有关，“实现一次，difftest 所有”
- 各种目标都是 Core，地位等价，自由组合
- ref 和 dut 都是 Core，地位等价，自由互换（有时候 ref 和 dut 需要同步开发）



AbsCore 虚基类

```
template<class addr_t, class word_t, uint8_t GPR_NUM>
class Core{
public:
    virtual void init()=0;
    virtual bool step(int step)=0;
    virtual bool get_trace(trace_t& trace)=0;
    virtual std::string get_perf()=0;
    // ...
```

difftest 主函数

```
template<typename ADDR_T, typename DATA_T, uint8_t GPR_NUM>
void difftest(Core<ADDR_T, DATA_T, GPR_NUM>& core, Core<ADDR_T, DATA_T, GPR_NUM>&
ref, ...){
  // ...
}
```

2.3 细粒度、基于事件跟踪的 difftest

细粒度、基于事件（而非状态的检查）的跟踪：

- 跟踪所有 GPR 的读写事件
- 跟踪所有跳转时的 PC 改变事件
- 跟踪所有访存的读写事件
- ...

好处：

- 利于 difftest 与具体的微架构、实现方式解耦
- 只检查改变量，减少额外开销
- 用户可以根据目标（可能无法跟踪一部分事件）或需求（只需要快速验证对错，不关心准确出错位置）的不同，调整细粒度的大小

但同时也增大了匹配的难度：

- 超标量提交、模拟器、DPIC 等目标提交事件的相对顺序可能不同

2.4 在线测试平台的仿真细节

通过分析在线竞赛平台每个 Testcase 的 Python 测评脚本源代码来建立自己的仿真环境，下面以性能测试中的 MATRIX 程序为例：

```
#                name, offset in kernel.bin, length
UTEST_ENTRY = [('STREAM', 0x3008, 0x28),
               ('MATRIX', 0x3030, 0x84),
               ('CRYPTONIGHT', 0x30b4, 0x94)]
# ...
def initialize(self):
  # ...
  kernel_bin = base64.b64decode(RESOURCES['kernel_bin'])
  self.utest = UTEST_ENTRY[IBOARD]
  off, size = self.utest[1], self.utest[2]
  self.test_bin = kernel_bin[off : off+size]
```

```
self.preloadTestData()
BaseRAM[:] = kernel_bin
```

```
def preloadTestData(self):
    # ...
    elif self.utest[0] == 'MATRIX':
        self.testdata = base64.b64decode(RESOURCES['matrix_bin'])
        ExtRAM[:0x30000:False] = self.testdata[:0x30000]
def verifyTestData(self) -> bool:
    # ...
    elif self.utest[0] == 'MATRIX':
        return ExtRAM[0x20000:0x30000:False] == self.testdata[0x30000:]
```

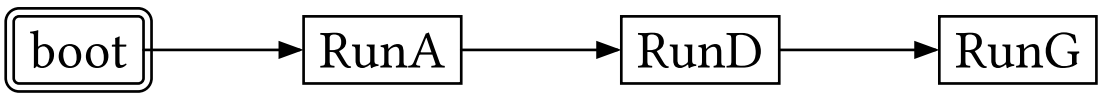
通过阅读以上 Python 代码，其实容易看出 MATRIX 程序的初始化相关细节：

- MATRIX 程序位于 kernel_bin 的 0x3030 处，长度为 0x84
- MATRIX 程序的测试输入数据被写入到了 ExtRAM 起始(即 0x80400000) ,长度为 0x30000
- MATRIX 程序的输出在 ExtRAM 的 0x20000 处(即 0x80420000) ，长度为 0x10000

有关运行的相关细节，同样可以参考 Python 测评脚本源代码中的 stateChange 相关函数，以及 RTFSC 监控程序源代码 https://github.com/z4yx/supervisor-mips32

Figure 1: 点击 ID 即可查看测评脚本

大致的启动流程：



- boot: 监控程序启动后，串口会输出启动信息 `MONITOR for Loongarch32 - initialized.`
- RunA: A 命令，会向 0x80100000 地址开始写入用户程序
- RunD: D 命令，会读取并向串口打印 0x80100000 地址开始的内存空间，以检查用户程序
- RunG: G 命令，会开始运行 0x80100000 处的用户程序，当用户程序开始时，会通过串口输出 `0x06` 提示开始计时，输出 `0x07` 提示计时结束，输出 `0x80` 提示出现错误

一些 tricks:

- 可以绕过麻烦的串口控制器，在总线接入 difftest
- 性能计数器可以在串口收到 `0x06` 时重置以获取更准确的性能信息
- 可以跳过 A、D 命令步骤，简化启动流程

Outline

1. Intro
   - 个人赛是什么
   - 今年的情况
2. Difftest
   - 痛点与需求分析
   - 基于统一抽象模型的 difftest
   - 细粒度、基于事件跟踪的 difftest
   - 在线测试平台的仿真细节
3. Tricks
   - 超频
   - 多周期访存
   - 协处理器

3.1 超频

- 个人赛中，允许一定的时序违例：即便 WNS 为负红了，在竞赛平台上也能跑通
- 但是时序违例后，会由于测评时刻的不同或者测评 FPGA 板素质的不同（~~超频制裁者~~），运行结果会变得不稳定（~~比较玄学~~）
- 这是个人赛中比较奇怪的部分，有一点非技术性的成分在里面，但是由于超频带来的收益较大，而且大家都超，你不超就寄了（
- 希望官方能出台规则，让 WNS 必须为正

3.2 多周期访存

- 个人赛的 BaseRAM 和 ExtRAM 只能工作在最高大概 60MHz
- CPU 可以工作在更高的频率，但如果不做跨频，那么 CPU 的工作频率将会被 RAM 的最高工作频率限制住
- 跨时钟频率有两种方式
  - 多周期访存：即对 RAM 读写时手动停住 N 个周期，对 RAM 而言就工作在 $\frac{\mathrm{clk}_{\mathrm{cpu}}}{N}$ 频率
  - AXI4 CDC：比较优雅，但是每次 sync 会多打两拍，性能开销比较大

```
when(rs === rs_r){
    when(io.axi.r.fire){
        rrcycs := rExtCycs.U
    }.otherwise{
        rrcycs := Mux(rrcycs === 0.U, 0.U, rrcycs - 1.U)
    }
}
io.axi.r.valid := rs === rs_r && rrcycs === 0.U
```

3.3 协处理器

- 决赛时，软件层面的优化（如循环展开）效果比较有限
- 当问题比较简单，或者问题可以分解成一个硬件相对容易解决的问题时，可以考虑编写专用的加速硬件（比如今年的求数组最大值）
- 具体来讲，可以把这个协处理器挂在总线上，通过 MMIO 的形式与 CPU 互相通信，协处理器可以自主管理访存（类似于 DMA 完全绕过 CPU），总线之间的通信可以使用 AXI4（仲裁等处理比较好做，但是握手开销比较大），CPU 向协处理器发出命令后，只需要轮询协处理器 `ready` 状态即可。
- 说起来容易，做起来难，在压力较大、时间紧迫的情况下，非常考验基础设施

祝大家能在龙芯杯玩的开心