面向RISC-V指令集扩展的软硬件协同敏捷验证方法

蒋子健，郑恪然，包云岗，石侃

中国科学院计算技术研究所 INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

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

Imperial College London

芯片开发的瓶颈——验证

- **芯片验证极其重要:**
  - 验证过程占据了整个芯片开发周期高达70%的时间

> The SoC must be verified for validation before being manufactured in a foundry, in a process known as functional verification. The verification process accounts for a large portion of the chip design life cycle —up to 70%. Once the chip is fully verified to work as intended, it is sent to the foundry for mass production.

—— Synopsys

芯片开发的瓶颈——验证

- **芯片验证极其重要:**
  - 验证过程占据了整个芯片开发周期高达70%的时间

> The SoC must be verified for validation before being manufactured in a foundry, in a process known as functional verification. The verification process accounts for a large portion of the chip design life cycle —up to 70%. Once the chip is fully verified to work as intended, it is sent to the foundry for mass production.

—— Synopsys

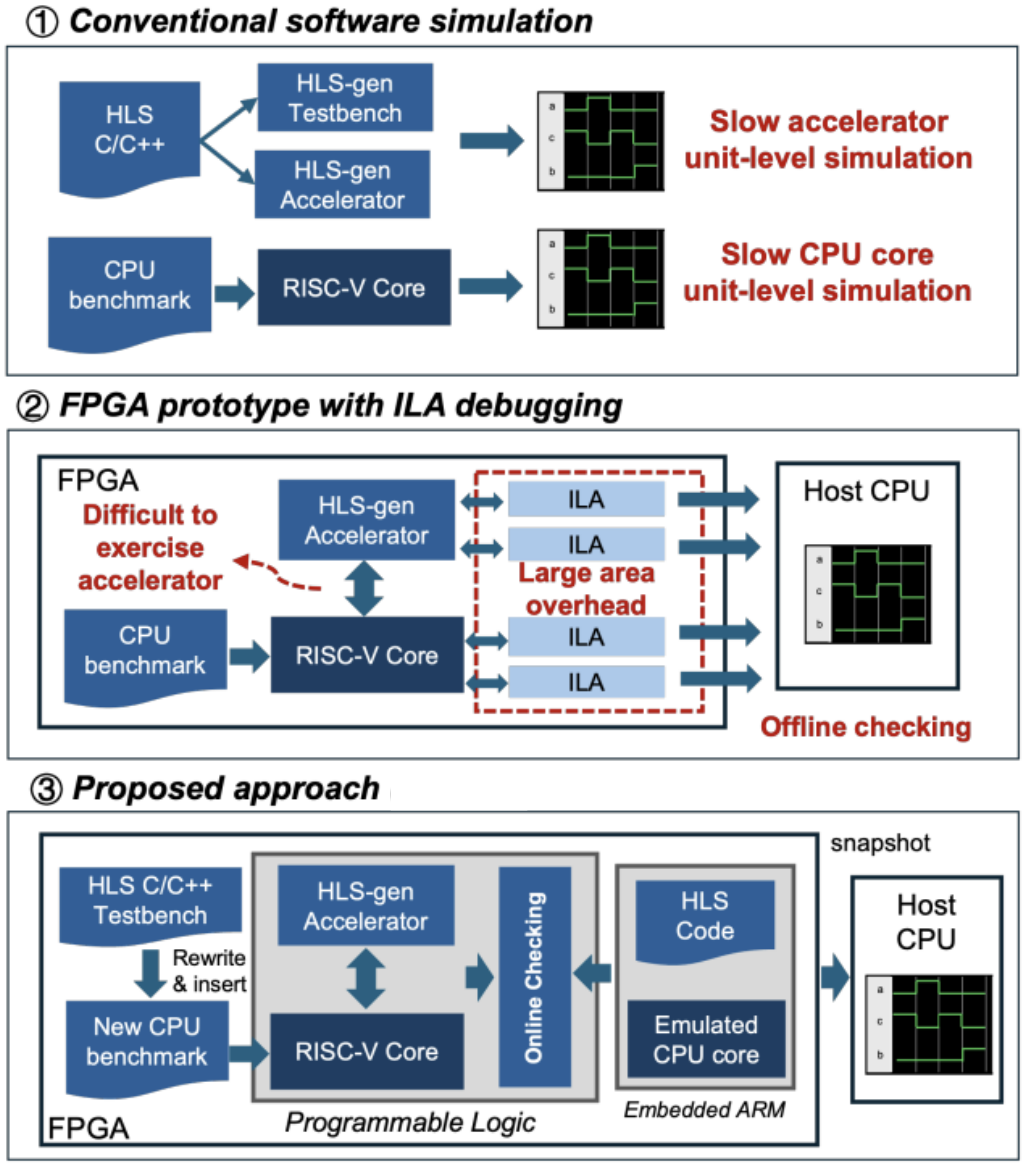
- **芯片验证非常困难:**
  - 现有的验证方法难以覆盖全部功能验证目标
  - 随机约束测试,通用验证方法学UVM, 形式化验证, FPGA原型验证, 仿真加速器…

验证RISC-V指令集扩展

- **RISC-V指令集扩展需求日益增加:**
  - RISC-V指令集架构（ISA）的重要特点之一是支持自定义扩展

- **如何验证包含RISC-V核以及扩展加速器的系统?**
  - 从unit-level verification到system-level verification

验证包含RISC-V核与HLS扩展加速器的系统



传统软件仿真方法

- 低仿真性能
- 单元级仿真

FPGA原型验证（使用ILA调试）

- 难以测试加速器
- ILA探针带来较大面积开销
- 离线检查

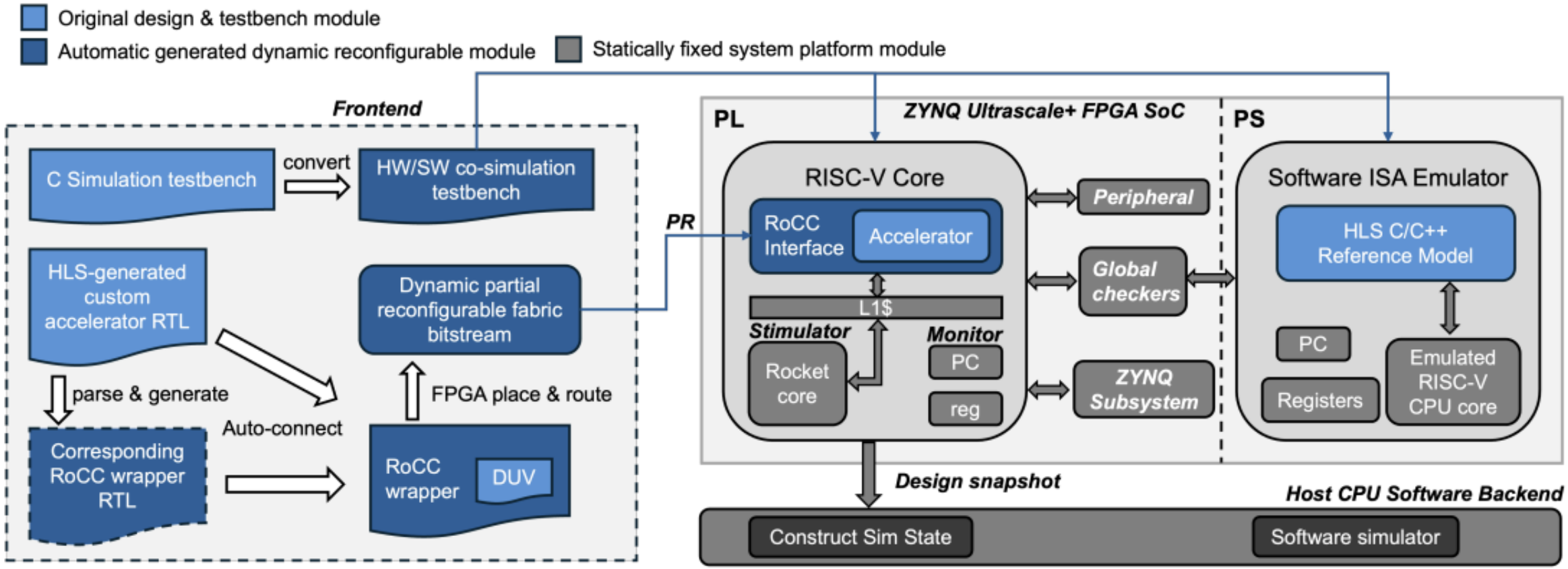
我们的方法

- RISC-V核连接加速器—系统级测试
- 运行时检查机制
- FPGA加速
- 较低的面积开销

主要贡献

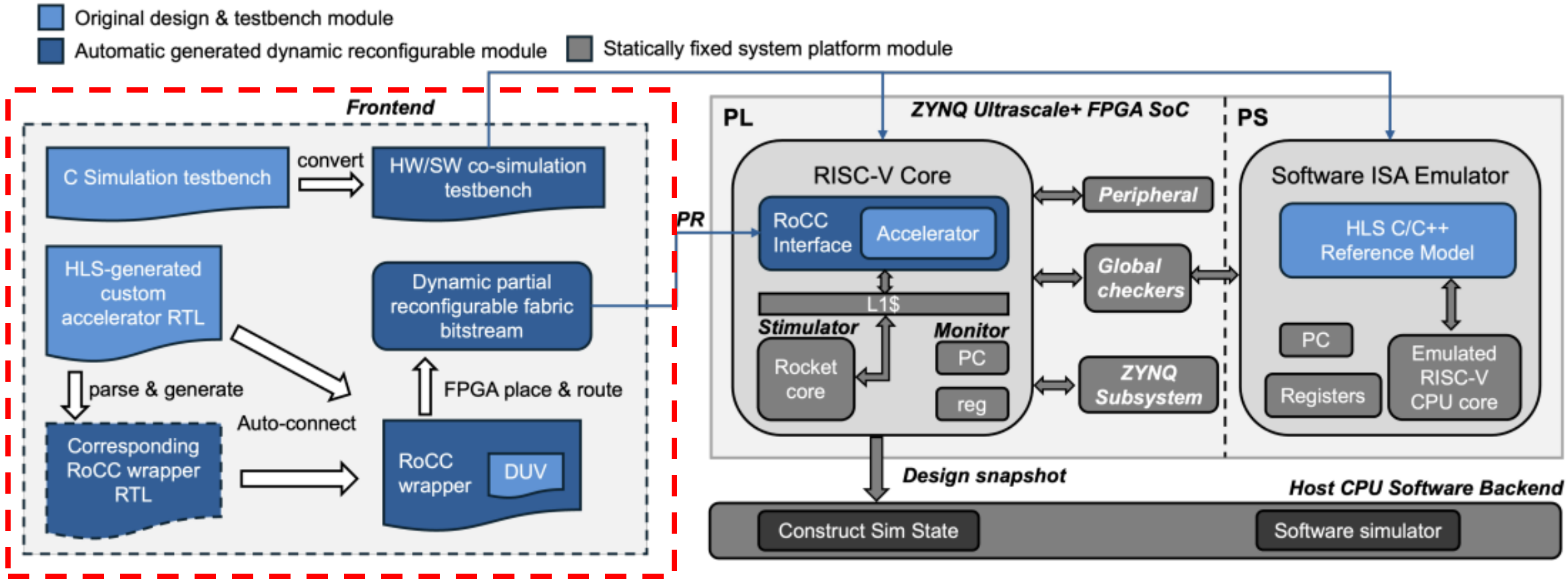
- **面向RISC-V指令集扩展的新验证方法**
  - 首个面向 RISC-V 处理器核及 HLS 扩展加速器的**系统级**验证框架
  - 基于ENCORE验证框架
- **自动化集成待测设计和测试程序的工具流**
  - 解析 HLS 扩展加速器，以较低的面积开销集成待测设计到系统中
  - 工具流将软件仿真 Testbench 自动转换成调用扩展指令的 HW/SW co-simulation 测试程序
- **运行时：自动进行系统级检查，软件自动收集覆盖率信息**
  - 多种覆盖率收集机制，确认各种功能验证目标完整覆盖
  - 运行时对硬件端待测设计以及软件参考模型实时检查
  - 一旦遇到潜在错误，硬件快照机制进行波形调试
- **在 AMD UltraScale+ FPGA 上使用真实的 Benchmark 相比软件仿真达到4423× 到16626x 的性能提升，并引入较小的面积开销**

总体架构



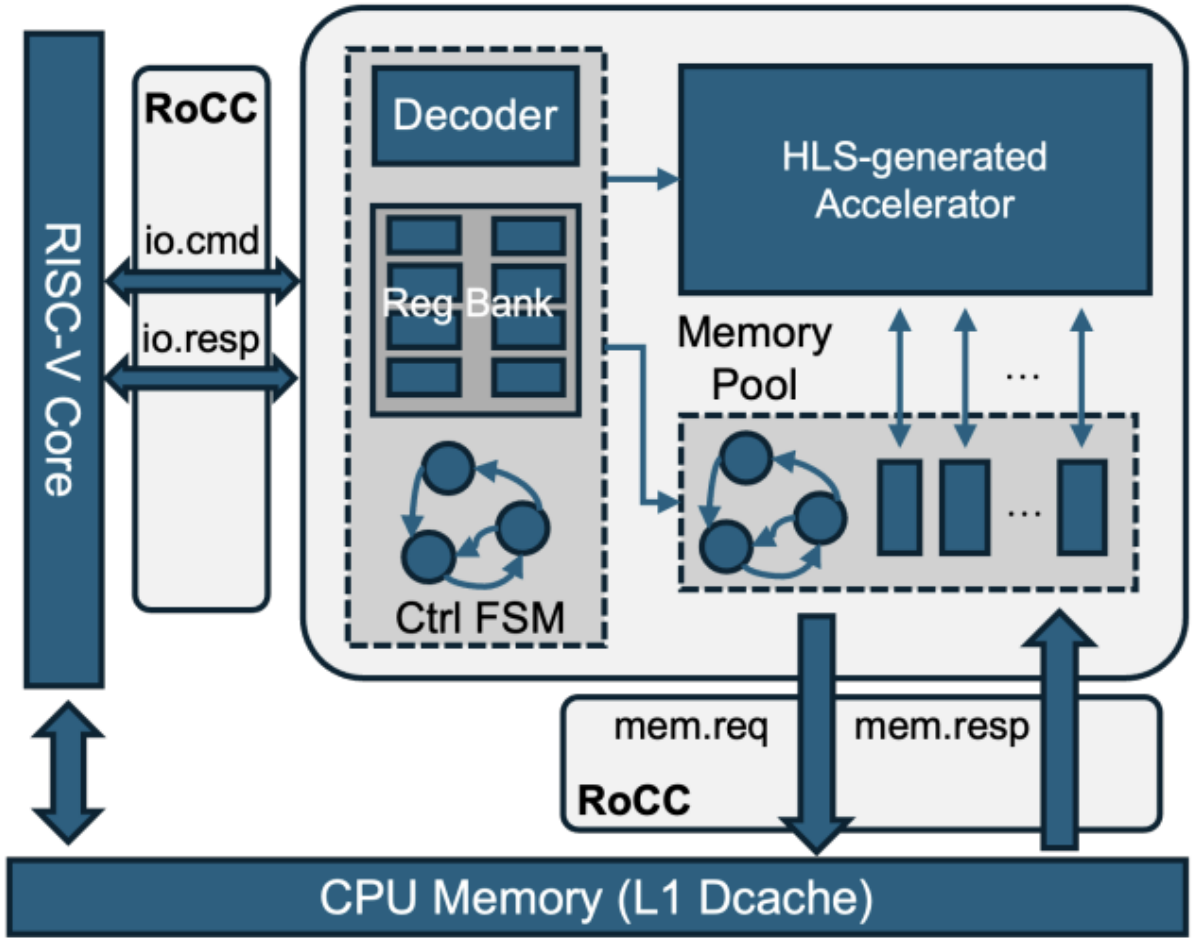
- **Frontend**——集成待测设计到系统中，转换测试用例为软硬件协同测试程序
- **FPGA architecture**——验证运行时架构，包括检查器，硬件待测设计，以及软件参考模型
- **Host CPU Software backend**——硬件快照机制，用于保存FPGA状态以及波形重建

总体架构



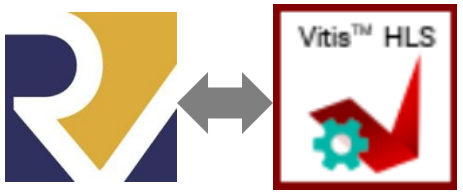
- **Frontend**——集成待测设计到系统中，转换测试用例为软硬件协同测试程序
- **FPGA architecture**——验证运行时架构，包括检查器，硬件待测设计，以及软件参考模型
- **Host CPU Software backend**——硬件快照机制，用于保存FPGA状态以及波形重建

前端架构



自动生成硬件Wrapper架构

- 用于对扩展指令执行的调度
- 硬件包括译码器、缓存池以及状态机
- 通过RoCC协议RISC-V Core与HLS加速器连接
- RISC-V Core与HLS加速器共享Dcache



前端架构

```c
int main(int argc, char **argv)
{
  struct bench_args_t data;
  int i, fd;
  struct prng_rand_t state;

  // Fill data structure
  prng_srand(1,&state);
  for(i=0; i<row_size*col_size; i++)
    data.orig[i] = prng_rand(&state)%(MAX-MIN) + MIN;
  for(i=0; i<f_size; i++)
    data.filter[i] = prng_rand(&state)%(MAX-MIN) + MIN;

  // Open and write
  fd = open("input.data", O_WRONLY|O_CREAT|O_TRUNC,
  ↪ S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH);
  assert( fd>0 && "Couldn't open input data file" );
  data_to_input(fd, (void *)(&data));

  return 0;
}

void run_benchmark( void *vargs ) {
  struct bench_args_t *args = (struct bench_args_t
  ↪ *)vargs;
  stencil( args->orig, args->sol, args->filter );
}
```

Code Snippet 1: Original HLS code samples for Stencil computation.

```c
int main() {
    uint64_t checksum_inst = 0;
    _ioe_init();

    for(uint32_t i = 0; i < ITERATION; i++) {
        bench_srand(i);
        //generate test vector
        generate();
    #if       TEST
        //call hardware accelerator
        //reset RoCC output memory
            LOAD_EXTEND(sol_base, sol, 13, WD);
            LOAD_EXTEND(orig_base, orig, 13, WD);
            LOAD(filter_base, filter, f_size, WD);
            EXEC(checksum_inst);
            STORE_EXTEND(sol_base, sol, 13, WD);
        asm volatile("fence":::"memory");
    #else
        //run on RISC-V core without acceleration
        stencil_functional(orig, sol, filter);
    #endif
        //check if the result is correct
        for (int j=0; j<8192; j++) sol[j];
    }
    return 0;
}
```

Code Snippet 2: Automatically generated test programs for the Stencil computation algorithm using proposed custom instructions to call accelerator.

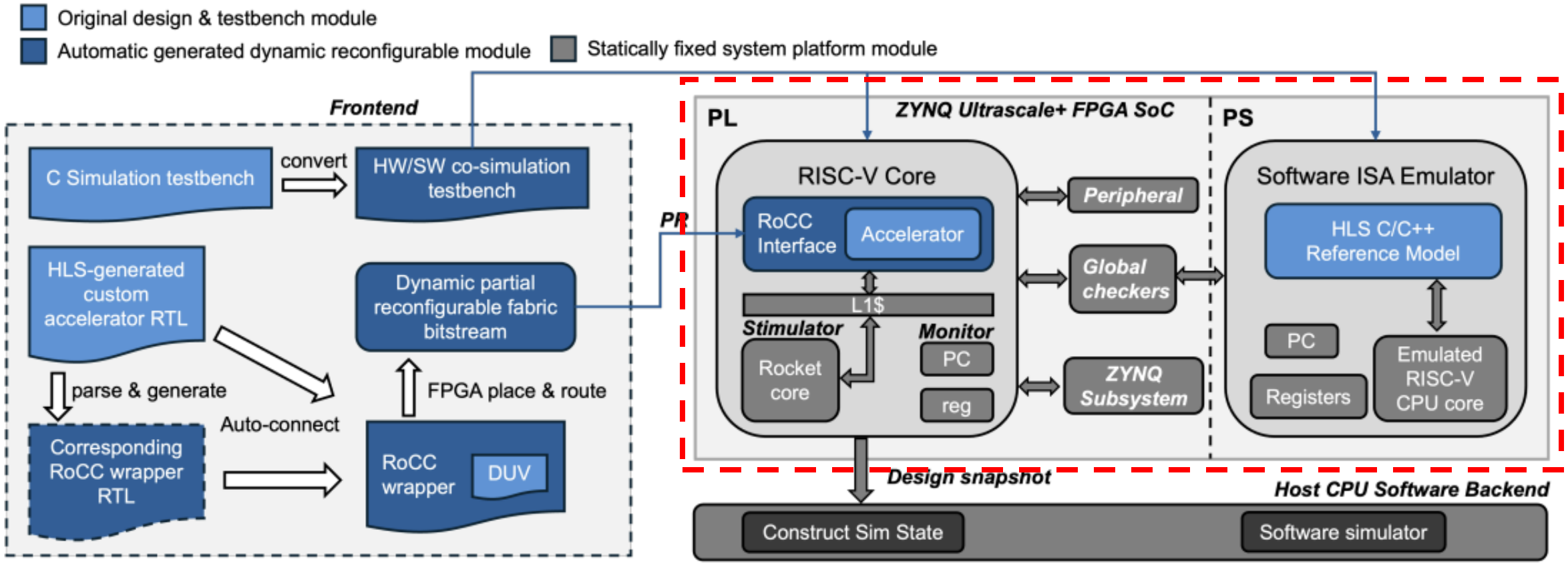
CPU与加速器间通信

- 定义load、store、execute三类通信指令

重写测试用例

- 在CPU Benchmark中插入扩展指令，调用加速器功能
- 待测设计与参考模型同步运行重写后的测试用例

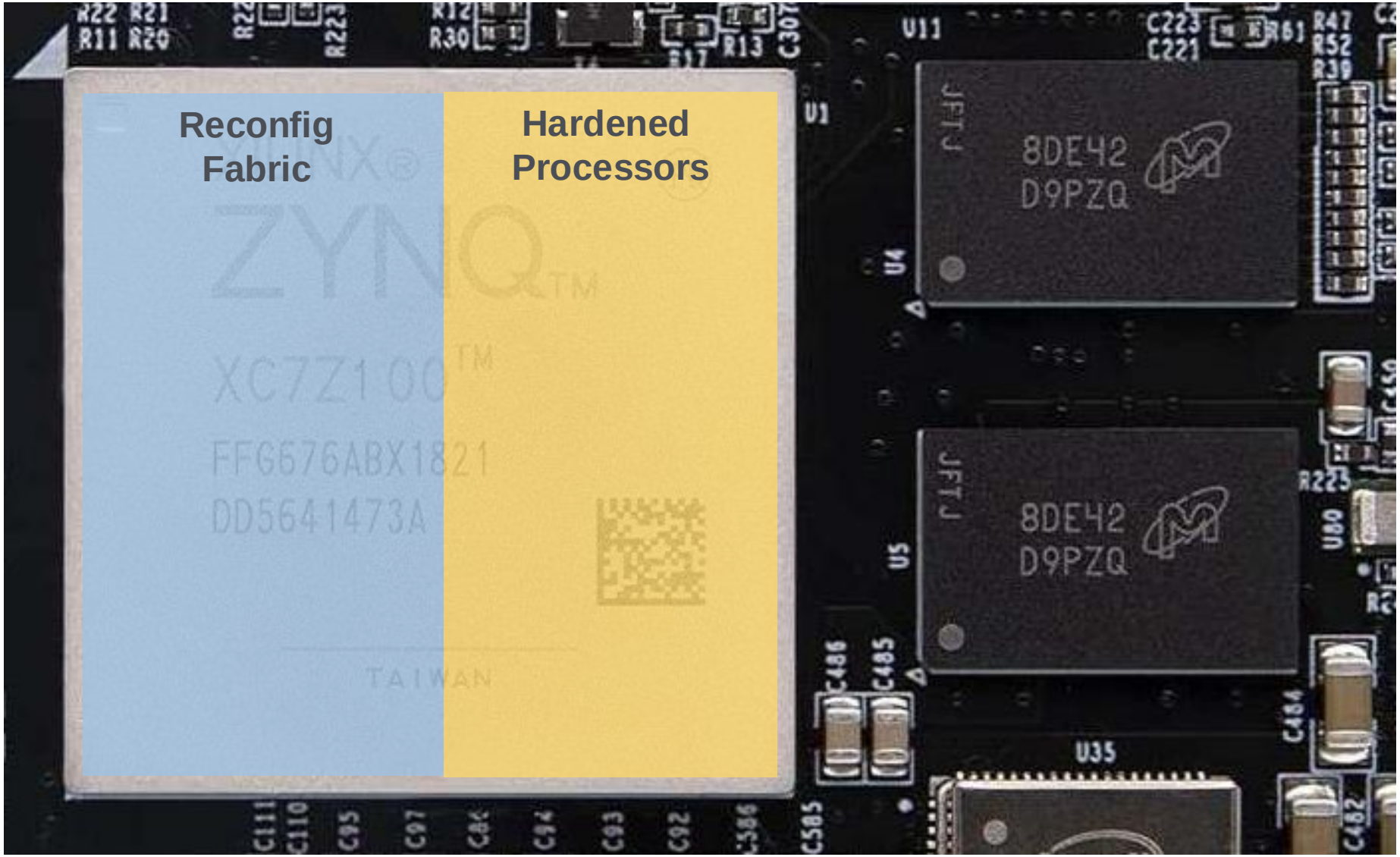
总体架构



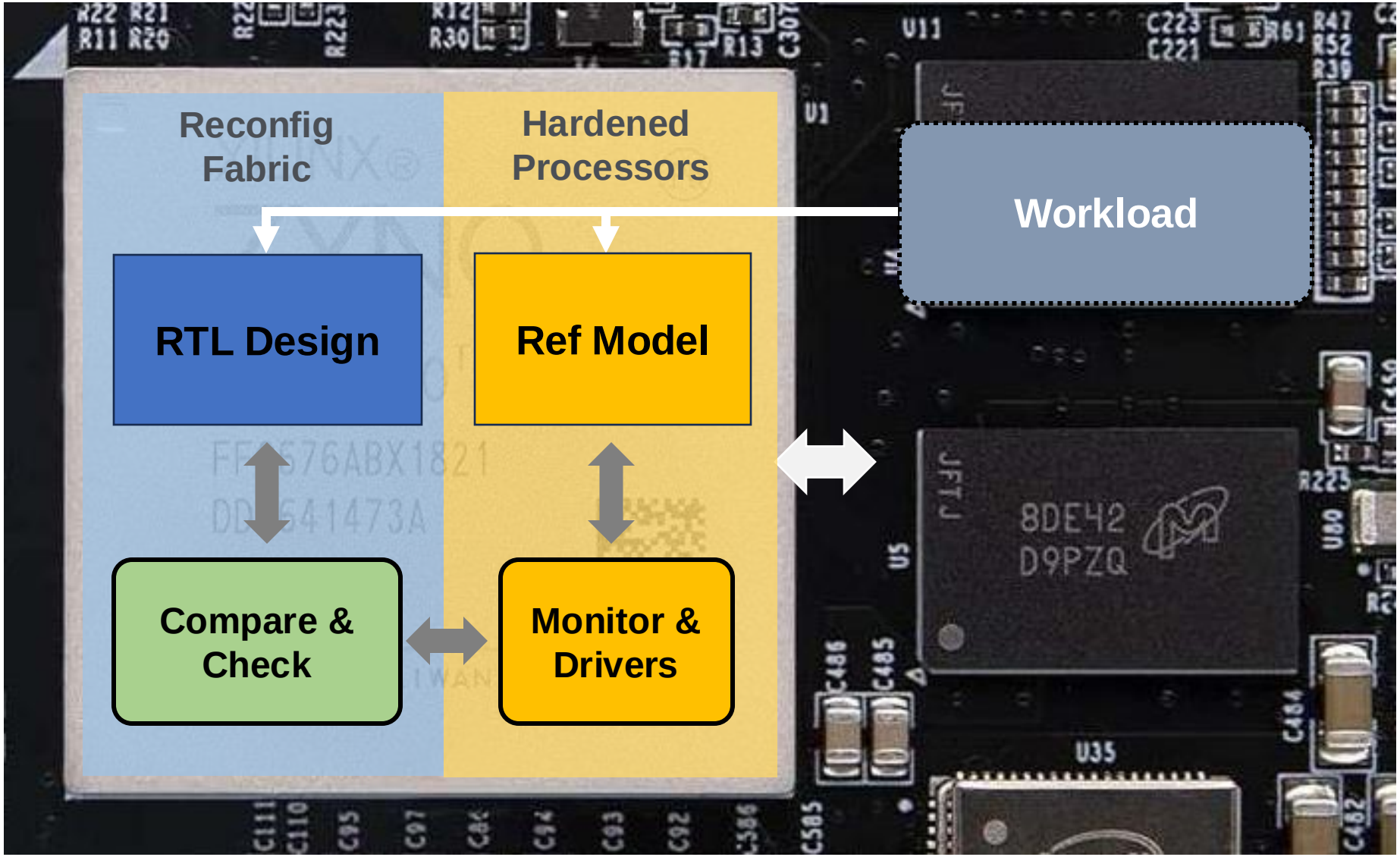
- **Frontend**——集成待测设计到系统中，转换测试用例为软硬件协同测试程序
- **FPGA architecture**——验证运行时架构，包括检查器，硬件待测设计，以及软件参考模型
- **Host CPU Software backend**——硬件快照机制，用于保存FPGA状态以及波形重建

硬件架构：基于ENCORE

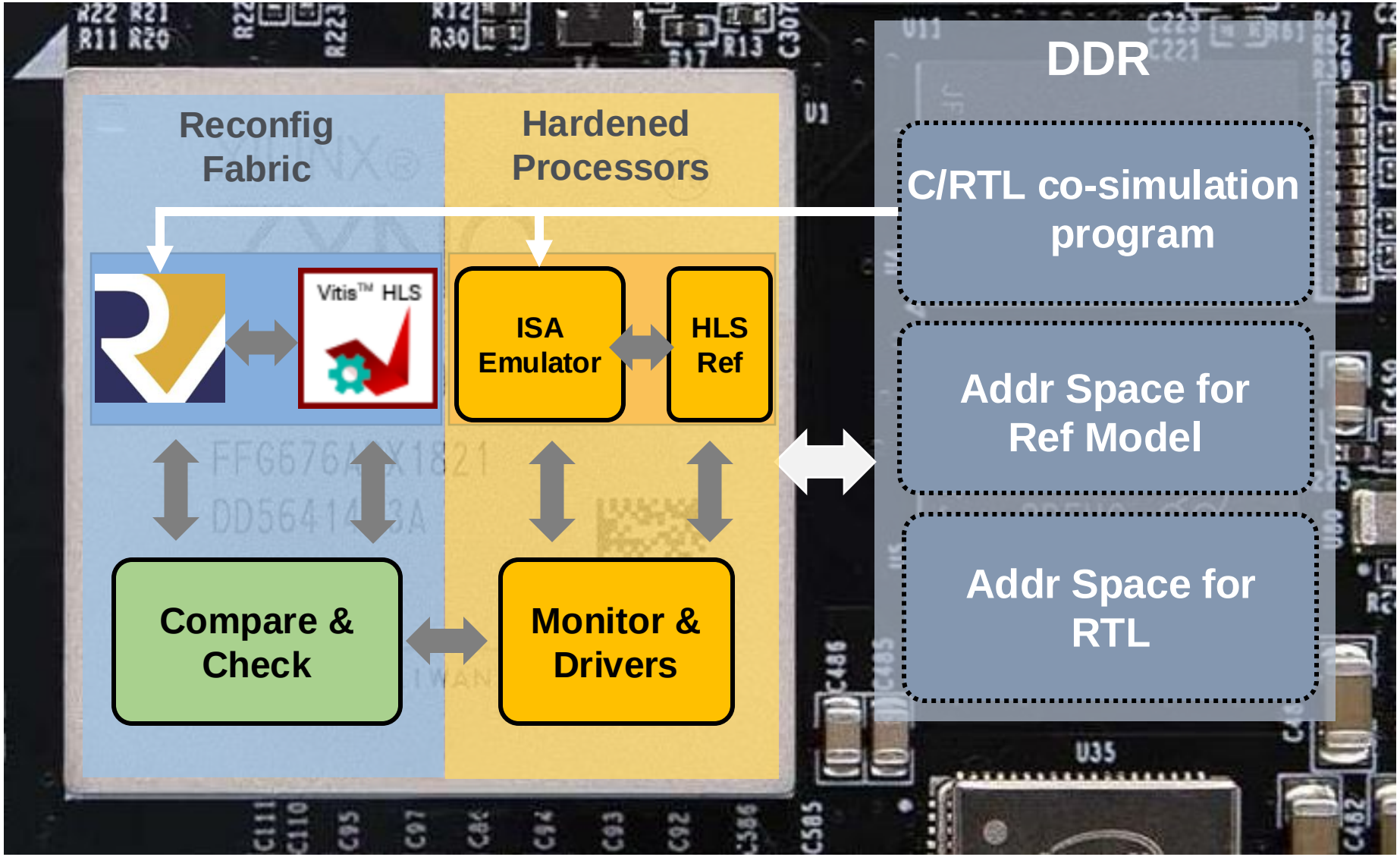
硬件架构：基于ENCORE



硬件架构：基于ENCORE



硬件架构：基于ENCORE



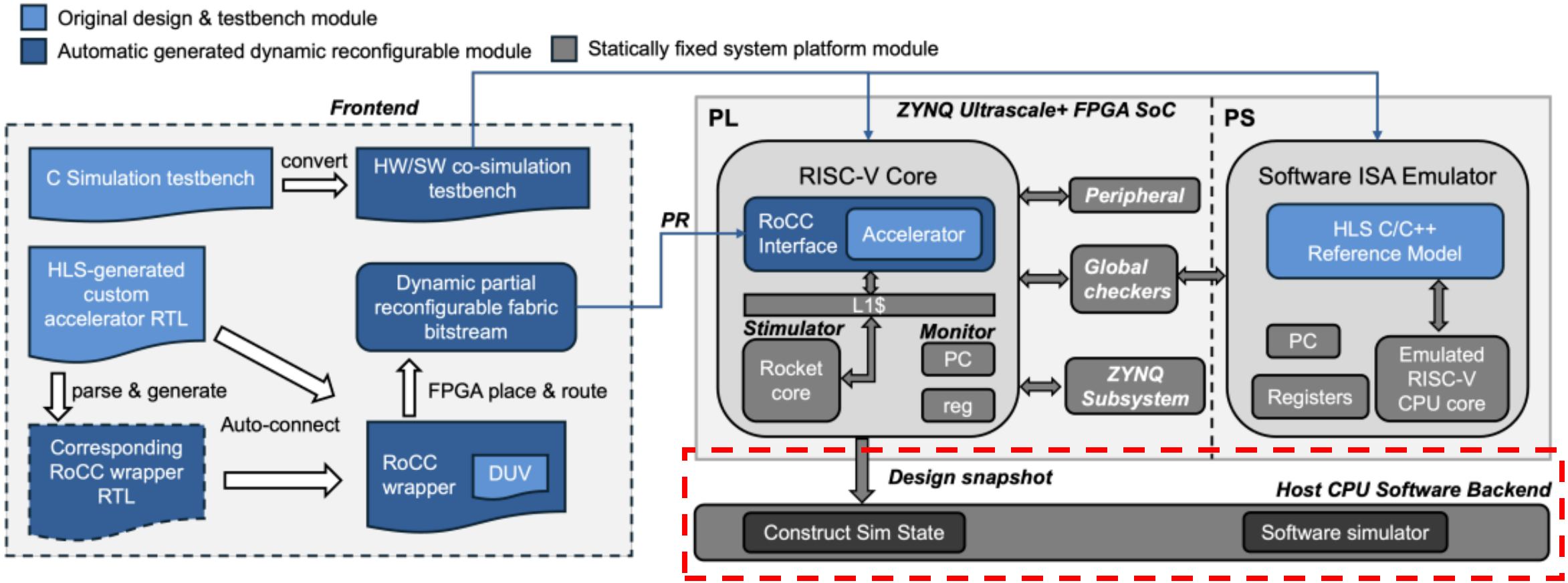
自动化检查&错误定位

- 软硬件对比RISC-V处理器核指令集架构（ISA）级信号。包括**PC、GPR、CSR**
- 可选择对比HLS加速器内部的关键信号

运行时架构

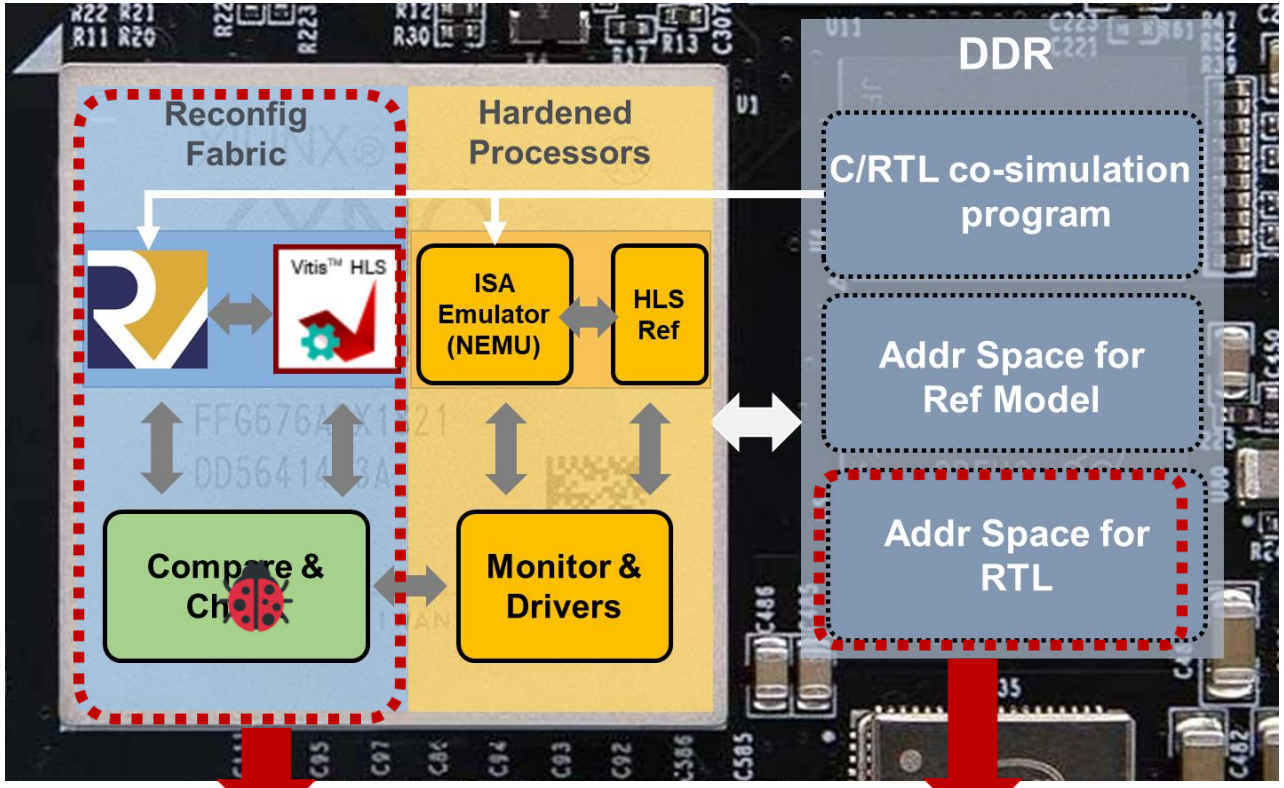
- 可编程区域运行：RISC-V core，HLS accelerator
- SoC硬核上运行：RISC-V ISA模拟器，HLS软件参考模型

总体架构



- **Frontend**——集成待测加速器设计到系统中，转换测试用例为软硬件协同测试程序
- **FPGA architecture**——验证运行时架构，包括检查器，硬件待测设计，以及软件参考模型
- **Host CPU Software backend**——硬件快照机制，用于保存FPGA状态以及波形重建

硬件快照&波形重建：基于ENCORE

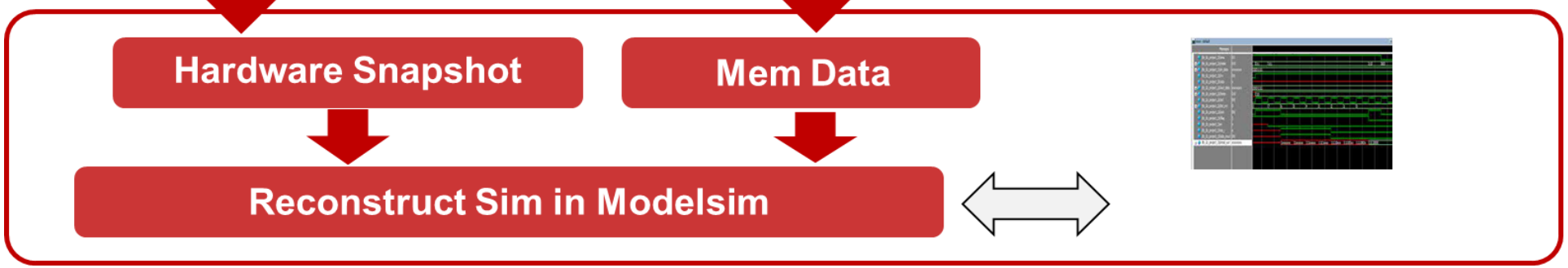


硬件快照

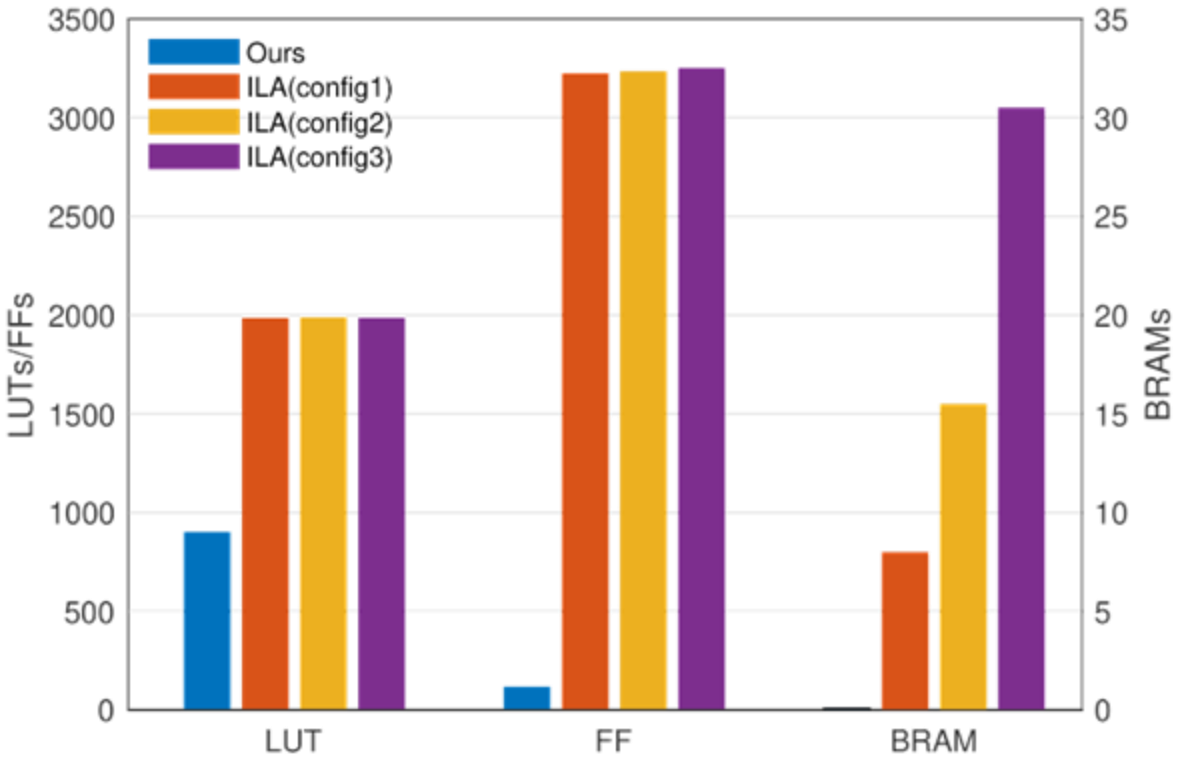
- 一旦对比出错，系统立刻中断PL上运行的全部硬件
- 保留加速器部分的硬件状态，包括FF、LUT、BRAM、DDR

波形重建

- 在Host CPU上在软件仿真器Modelsim中重建仿真，对出错的原因进行细粒度波形调试



资源开销评估



资源开销评估

- 使用真实的HLS加速器设计Benchmark（MachSuite）作为待测扩展设计进行资源评估

与传统FPGA调试方法比较

- 基于ILA:集成逻辑分析器
  - 探针监视更多信号需要**额外逻辑资源**
  - 采样深度有限，只能观测一部分波形，额外的采样深度**消耗大量BRAM资源**

我们的方法

- 运行时自动对比检查RISC-V core的PC、GPR、CSR等关键寄存器，以及HLS加速器IP端口级信号
- 硬件快照+仿真重建机制，能够观测**所有信号的全部波形**
- 我们的方法的资源开销小于ILA，保留了波形可见度，提供自动检查机制

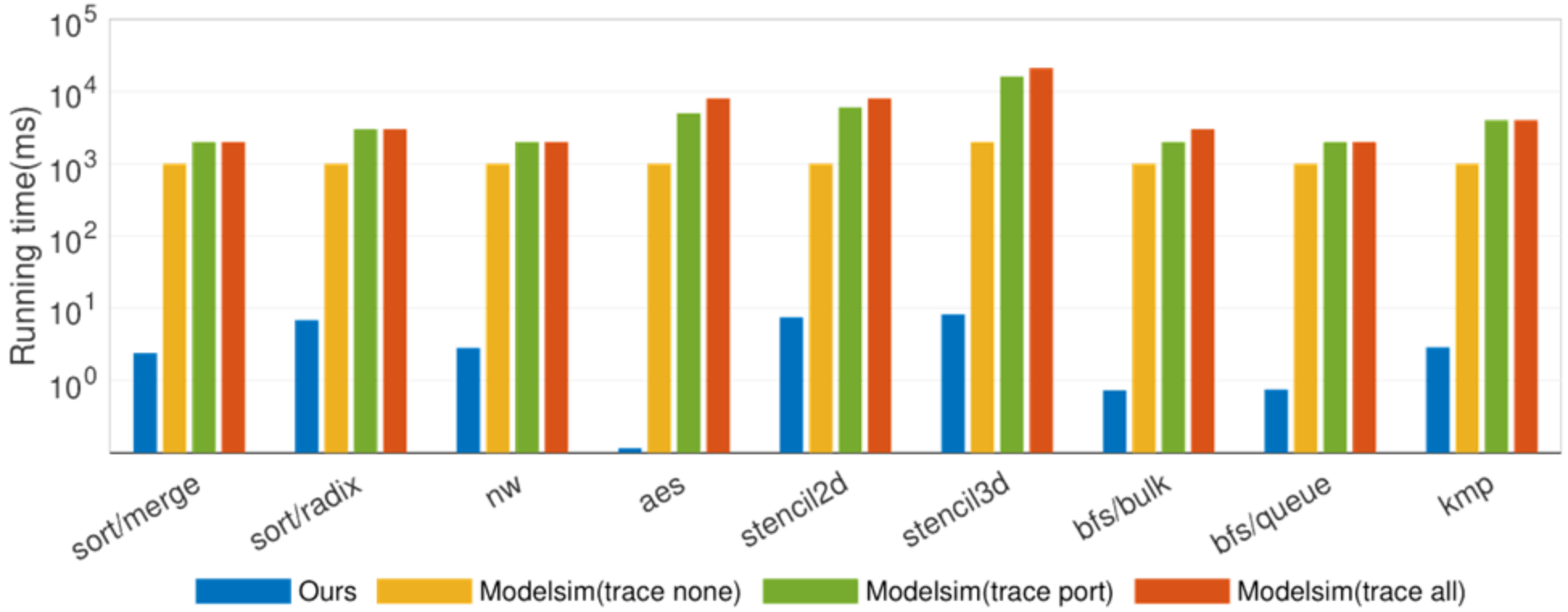
性能评估

实验环境设置

- 使用真实的HLS加速器设计Benchmark （MachSuite）作为待测扩展设计进行性能评估
- Vitis HLS co-simulation软件仿真方法作为baseline，Modelsim作为软件仿真器
- 实验运行在AMD UltraScale+ XCZU19EG FPGA上，通过JTAG连接到AMD Ryzen 5950x 16核服务器

与软件仿真方法性能比较

- **我们的方法相对 Modelsim 加速: 4423× to 16626×**



结论

相关工作已在ISEDA'24会议发表
欢迎合作交流!
我的邮箱:
jiangzijian24@mails.ucas.ac.cn
石侃老师邮箱:
shikan@ict.ac.cn

- **面向RISC-V指令集扩展的新验证方法**
  - 首个面向 RISC-V 处理器核及 HLS 扩展加速器的**系统级**验证框架
  - 基于ENCORE验证框架
- **自动化集成待测设计和测试程序的工具流**
  - 解析 HLS 扩展加速器，以较低的面积开销集成待测设计到系统中
  - 工具流将软件仿真 Testbench 自动转换成调用扩展指令的 HW/SW co-simulation 测试程序
- **运行时：自动进行系统级检查，软件自动收集覆盖率信息**
  - 多种覆盖率收集机制，确认各种功能验证目标完整覆盖
  - 运行时对硬件端待测设计以及软件参考模型实时检查
  - 一旦遇到潜在错误，硬件快照机制进行波形调试
- **在 AMD UltraScale+ FPGA 上使用真实的 Benchmark 相比软件仿真达到4423× 到16626x 的性能提升，并引入较小的面积开销**