北京航空航天大學
BEIHANG UNIVERSITY

北航龙架构处理器芯片敏捷设计框架

周振源，王哲，郭鸿宇，苏阳，王钧石

北京航空航天大学 计算机学院

2024年7月25日

技术交流微信群

培训提纲

一 项目背景

二 总体规划

三 实践案例

四 总结与展望

1.1 系统能力培养目标

- **系统能力培养的三个挑战**
  - **难度大**：无法像开发软件一样开发硬件
  - **效率低**：对工程能力要求高
  - **错误多**：无稳定的系统测试平台（LoongArch）
- **已有课程只提供有限的支持**

课程	开发难度	开发效率	开发正确性
无实验环节的计算机组成课程	大	低	无法保证
北航计算机组成课程	中	中	基本保证
国科大一生一芯项目	大	高	保证
伯克利计算机体系结构及工程课程	中	高	未保证

- **解决方案**
  - **保持工作精简**
  - **尽早且持续地交付**
  - **重视可制造的原型**
  - **重视工具和生成器**

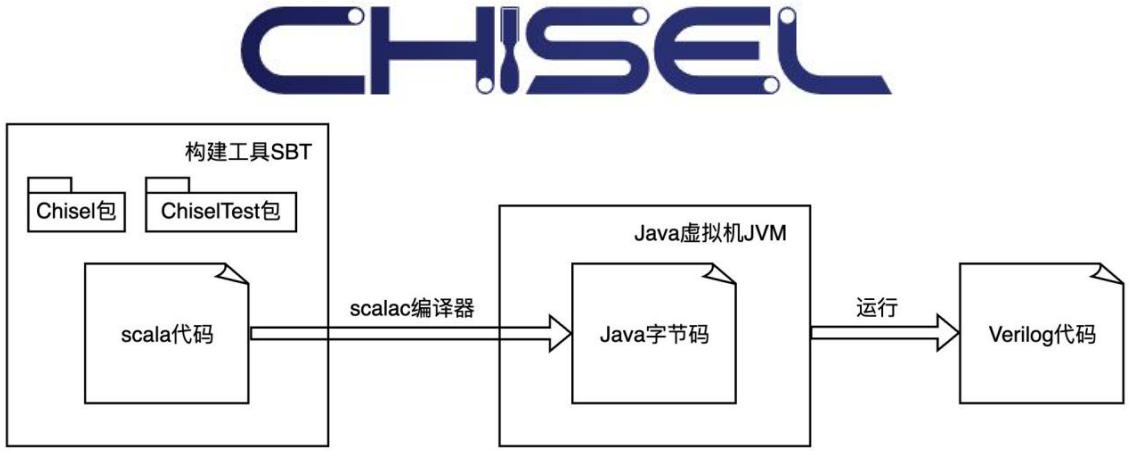
——软件/硬件敏捷开发宣言

- **项目目标**
  - **集成敏捷开发工具**
  - **促进系统能力培养**

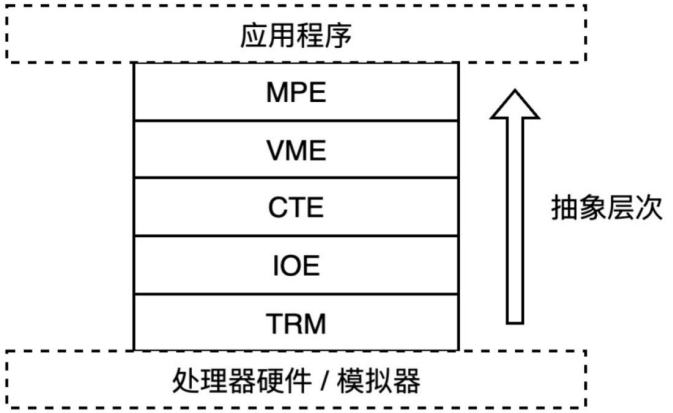
充分利用高效开发工具，让本科生能驾驭一定工程规模的硬件开发

1.2 现有敏捷开发工具

- **硬件构建语言 Chisel**
  - Scala 内嵌的开源高级硬件描述语言
  - 支持编写复杂、可参数化的电路生成器
  - 面向对象/函数式编程
  - 丰富开源工具库

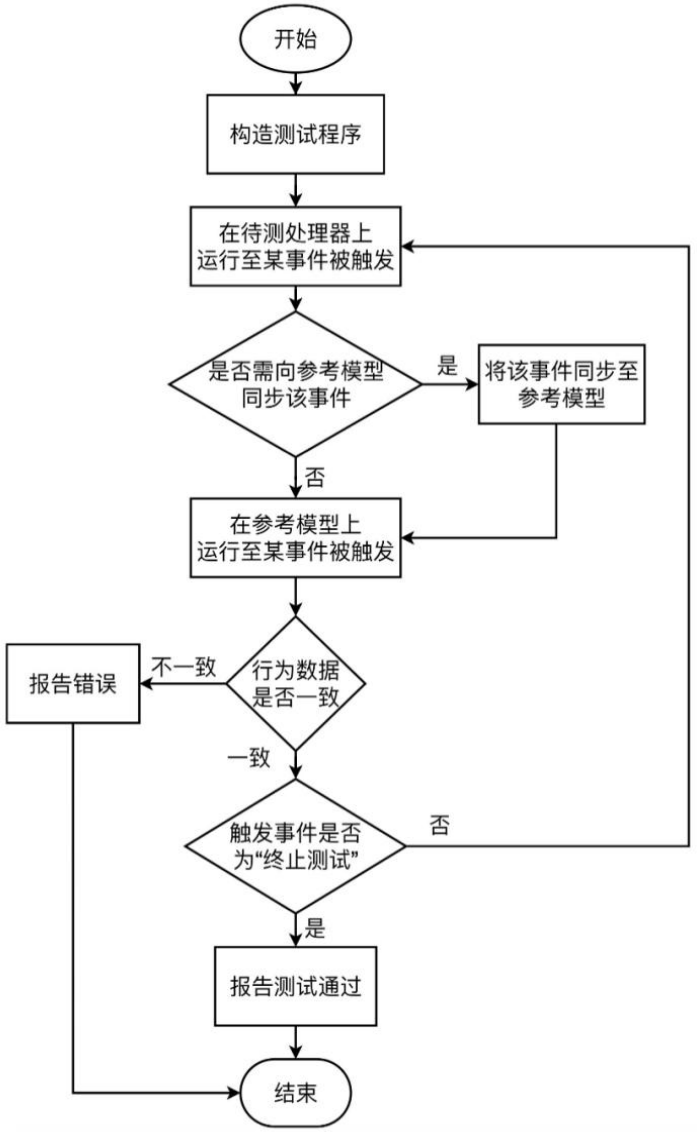


- **裸机运行时环境 AM**
  - 应用程序的运行都需要运行时环境的支持
  - 只进行纯粹计算任务的程序在TRM上就可以运行
    - 把程序放在正确的内存位置
    - 让PC指向第一条指令
    - 计算机自动执行程序



1.2 现有敏捷开发工具

- **基于GitLab的持续集成**
  - 覆盖软件开发周期的开放一体化 DevOps 平台
  - 编写自动化测试脚本持续进行 CI/CD 回归测试
- **差分测试框架 DiffTest**
  - 现代协同仿真框架， DPI-C 实现跨语言交互
  - 在线指令集行为验证方法，运行/验证同步进行
- **仿真模拟器 NEMU**
  - 单周期处理器行为仿真模型，全系统模拟
  - 灵活拓展，支持不同指令集架构



1.3 Chiplab 项目

- **chip：两套顶层 SoC 代码，分别用于仿真及综合**
- **IP：搭建 SoC 所需的 IP（串口、网卡、CPU 等)**
- **软件**
  - 功能测试集：基本功能测试、Linux，RTThread 等
  - 性能测试集：Dhrystone、Coremark 、Fireeye 等
  - 随机指令序列
- **工具链**
  - la32r-nemu：difftest 金标准
  - newlib：C库
- **仿真环境：基于 difftest，Verilator / IVerilog**

Chiplab 致力于构建基于 LoongArch 32 Reduced 的 SoC 敏捷开发平台

1.4 Chiplab 局限

- **高耦合度**
  - 配置参数过多，且部分参数互相依赖，心智负担重
  - Makefile 复杂，难以修改，参数分散在各个文件中
  - 一次运行需要走完仿真环境/RTL/测试软件的全部 make 流程，链条过长
  - git 提交记录混淆 IP 核修改 / 测试软件修改 / 仿真框架修改
- **难以调试**
  - 仿真框架中访存及串口均由 C++ 模拟，无法查看波形
  - newlib 没有源代码、难以查看测试程序的完整反汇编
- **难以扩充**
  - 难以运行外部测试程序
  - 难以修改 SoC 结构，添加外设

Chiplab 不符合高内聚低耦合原则，难以适应复杂多变的 CPU 开发需求

汇报提纲

一 项目背景

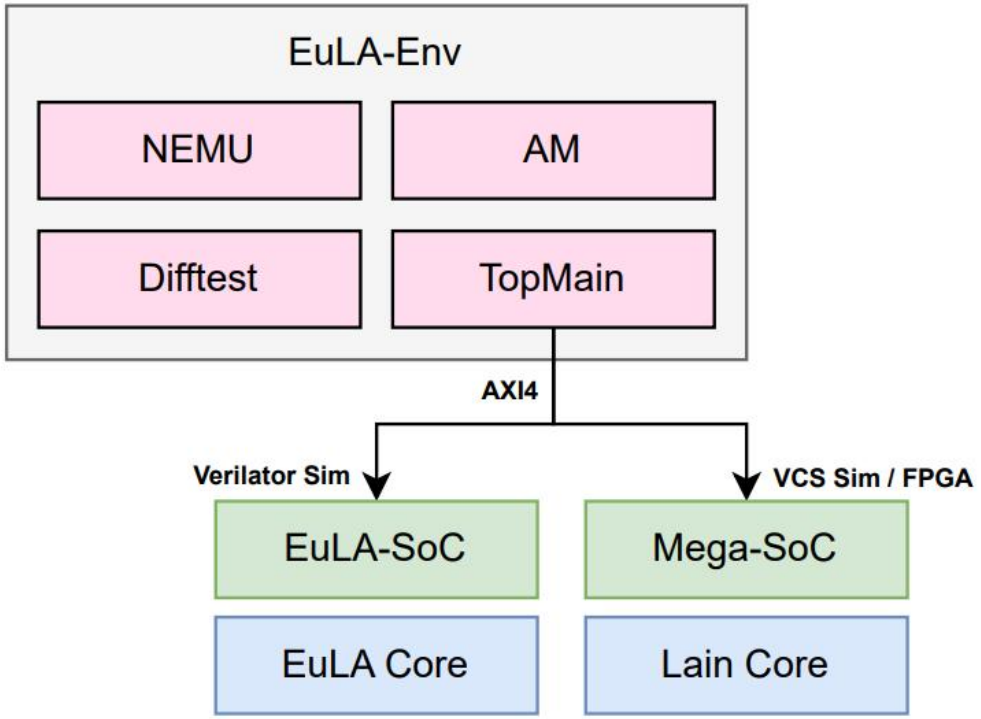
二 总体规划

三 实践案例

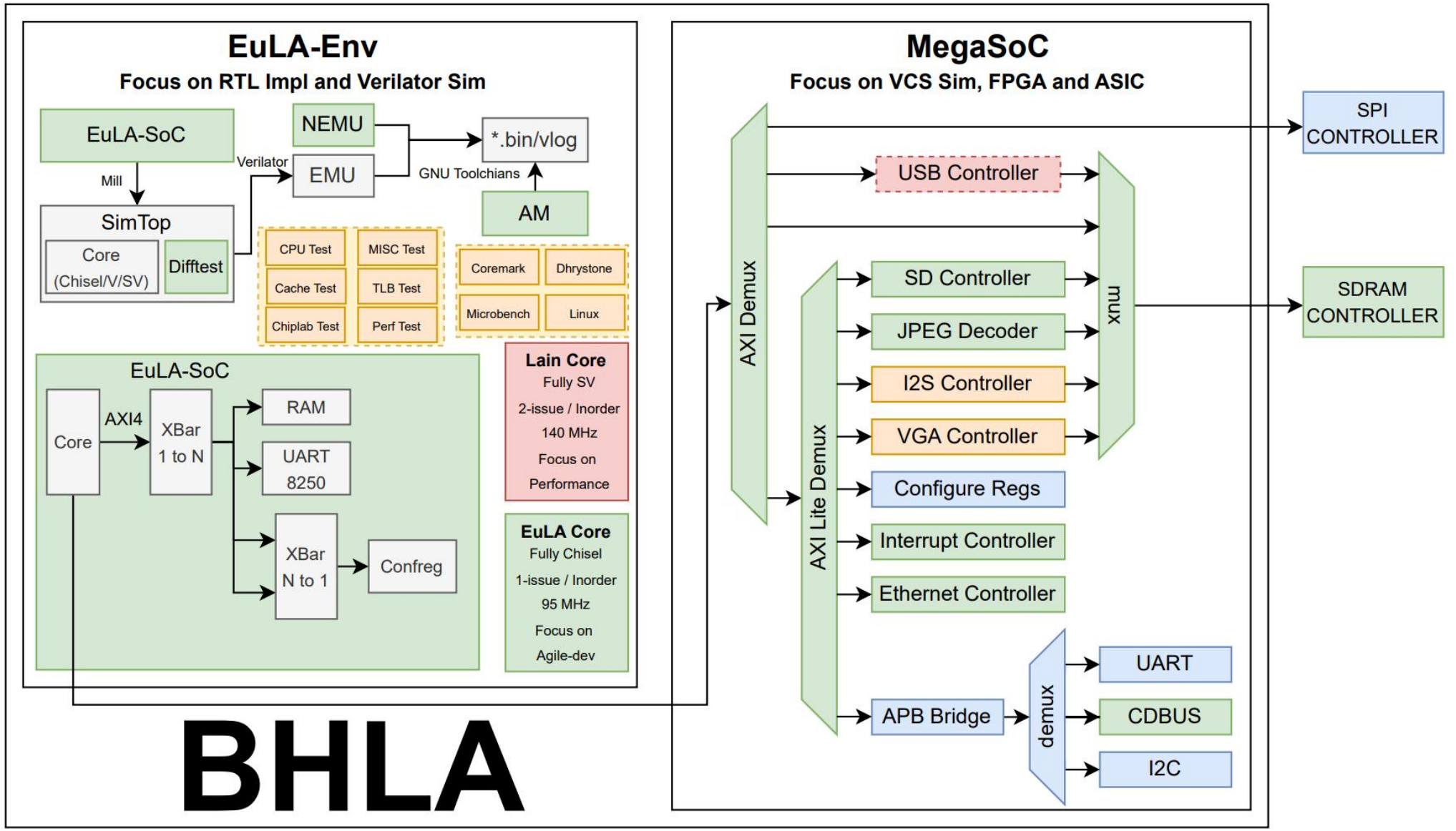
四 总结与展望

2.1 BHLA 总体框架

- **EuLA-Env：处理器敏捷设计集成平台**
  - NEMU：单周期仿真模拟器
  - AM：裸机运行时环境
  - Difftest：差分测试框架
  - TopMain：处理器模块顶层
- **EuLA-SoC：Verilator 仿真 SoC**
  - EuLA Core：EuLA-SoC / Difftest 接入样例 CPU
- **Mega-SoC：VCS 仿真 / FPGA 功能 SoC**
  - Lain Core：Mega-SoC 接入样例 CPU

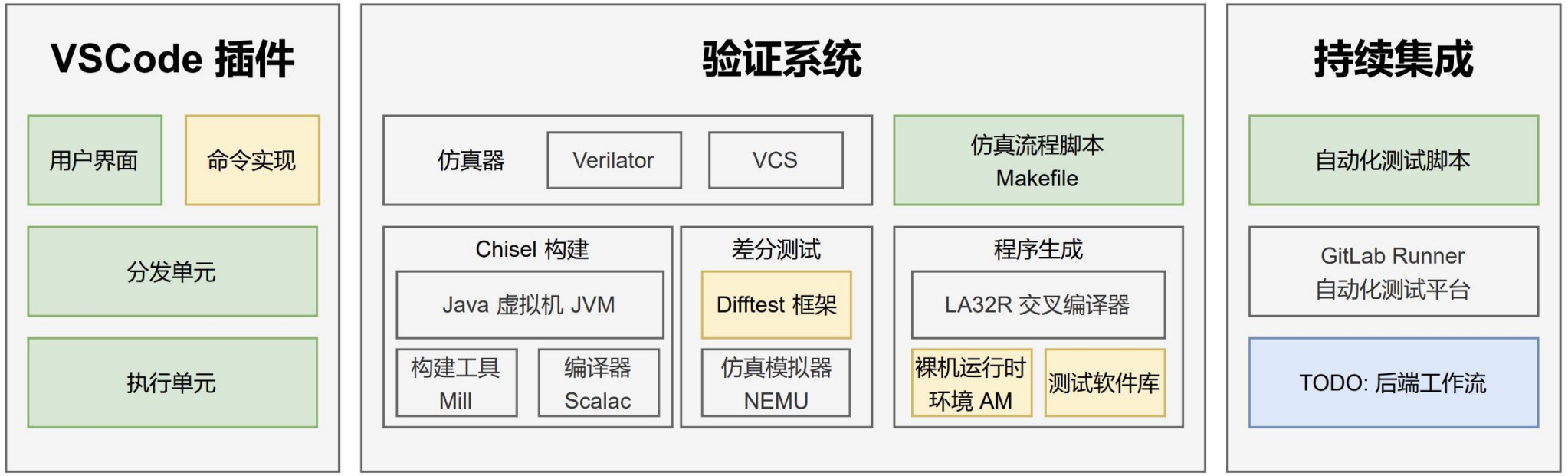


2.1 BHLA 总体框架


EuLA-Env
Focus on RTL Impl and Verilator Sim
EuLA-SoC
Mill
SimTop
Core
(Chisel/V/SV)
Difftest
Verilator
NEMU
EMU
*.bin/vlog
GNU Toolchians
AM
CPU Test
MISC Test
Cache Test
TLB Test
Chiplab Test
Perf Test
Coremark
Dhrystone
Microbench
Linux
EuLA-SoC
Core
AXI4
XBar
1 to N
RAM
UART
8250
XBar
N to 1
Confreg
Lain Core
Fully SV
2-issue / Inorder
140 MHz
Focus on
Performance
EuLA Core
Fully Chisel
1-issue / Inorder
95 MHz
Focus on
Agile-dev
BHLA
MegaSoC
Focus on VCS Sim, FPGA and ASIC
AXI Demux
AXI Lite Demux
USB Controller
SD Controller
JPEG Decoder
I2S Controller
VGA Controller
Configure Regs
Interrupt Controller
Ethernet Controller
APB Bridge
demux
UART
CDBUS
I2C
mux
SPI
CONTROLLER
SDRAM
CONTROLLER

2.2 EuLA-Env

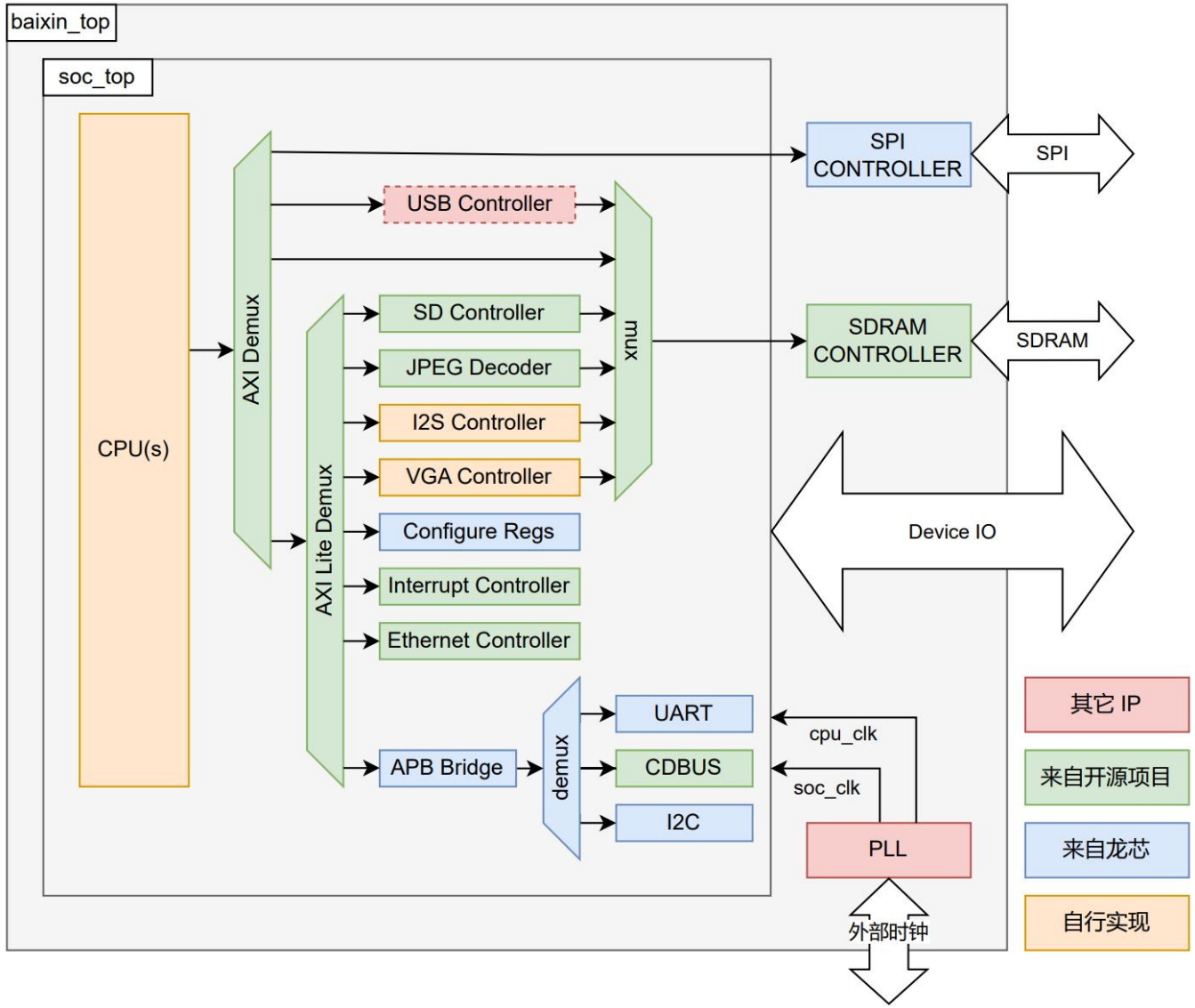
- **集成敏捷开发平台**：便捷配置，易于拓展
- **完备前端工作流**：Verilator / VCS 仿真，无缝上板
- **特色片上系统**：结构清晰，外设丰富，流片验证
- **全流程自动化**：持续回归测试，设计/测试同步推进



EuLA-Env 致力于构建 LoongArch 32 Reduced 集成芯片敏捷开发平台

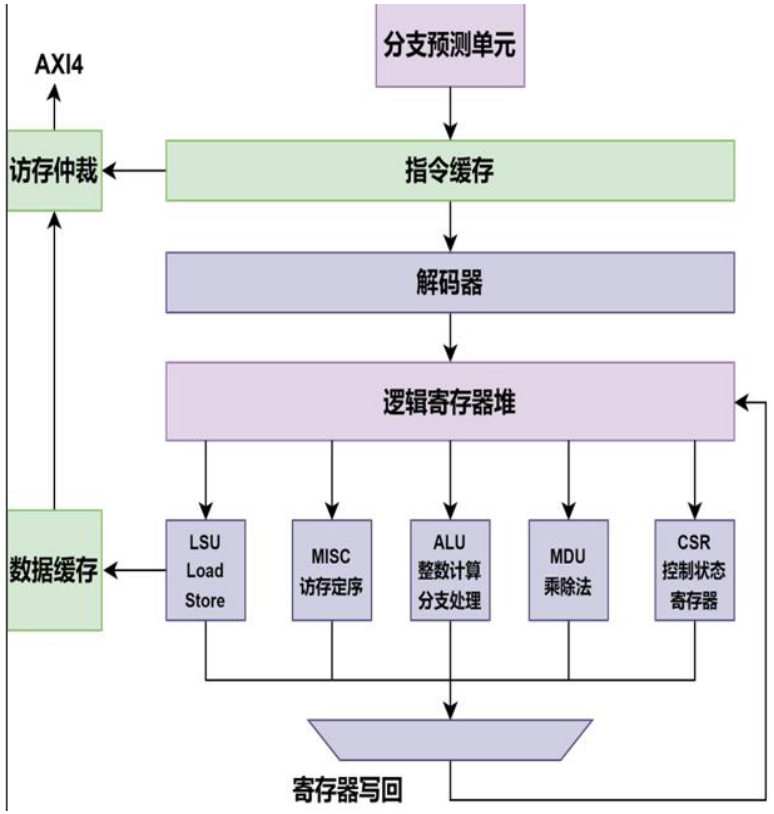
2.3 MegaSoC

- **总线支持参数化配置**：易于拓展丰富外设，适配处理器核需求
- **支持丰富外设接口**：UART、SPI、RMII、SDIO、VGA、I2S、I2C
- **多媒体应用特化**：集成开源 JPEG 硬件解码器 - 支持视频播放
- **纯 RTL 交付**：不依赖 Xilinx Primitive，灵活支持不同平台



2.4 芯片敏捷开发成果展示

EULA九级单发射处理器



Chisel 硬件描述语言开发
完整验证芯片敏捷开发全流程

共同技术特点

支持 LoongArch
32位精简指令集整数部分

实现混合分支预测机制
有效减少流水线空泡产生

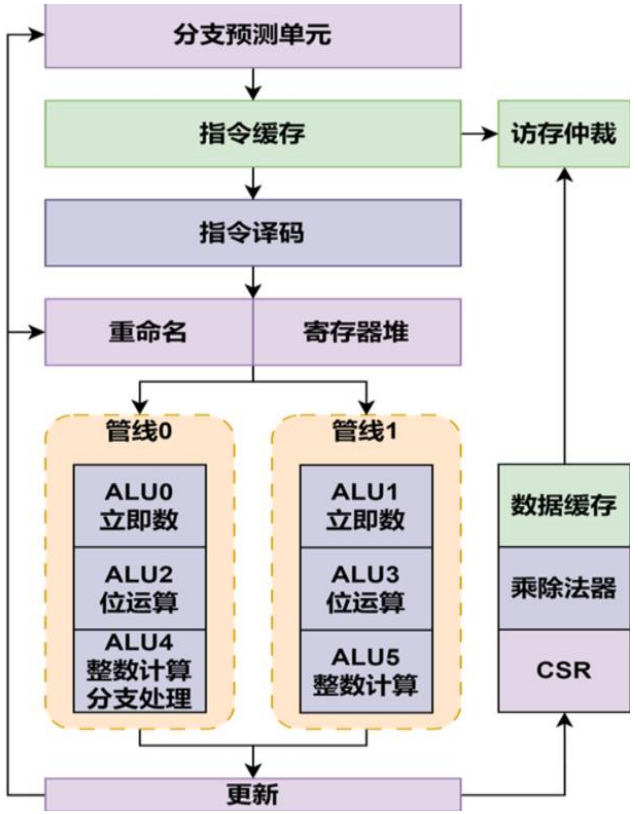
实现 AXI4 总线协议
利用 Burst 传输提高访存速度

8KB 指令缓存
8KB 数据缓存

支持虚拟地址翻译
及TLB快表机制

支持运行 Linux 5.19 内核、
U-Boot 等多种系统软件

Lain顺序双发射处理器

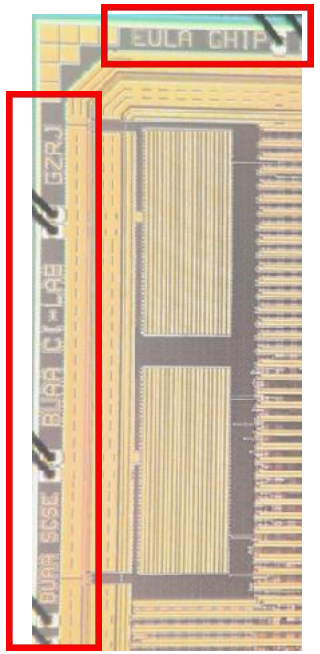


重命名机制优化执行管线
挖掘顺序核性能极限

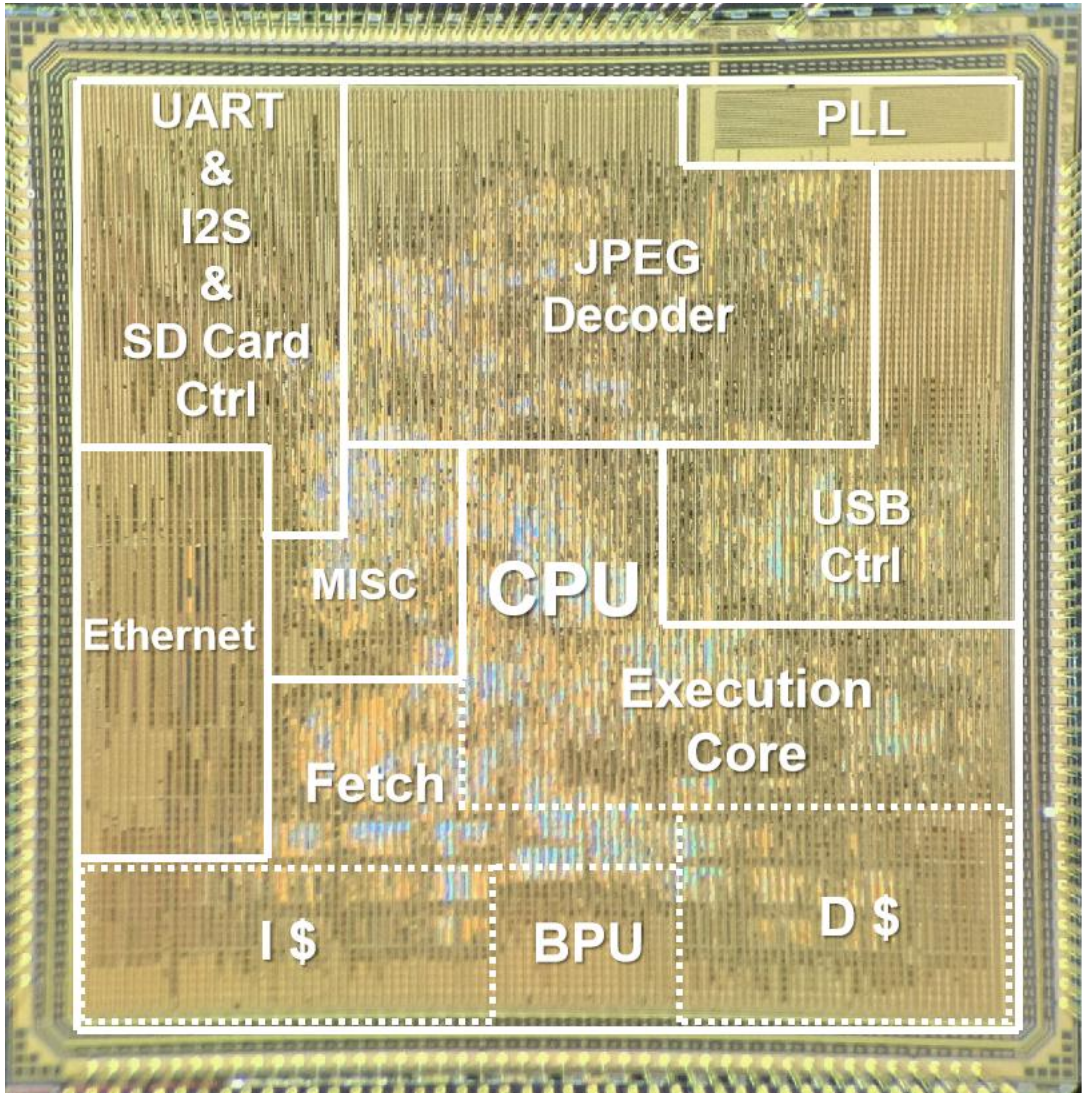
2.4 芯片敏捷开发成果展示

- **EULA 处理器芯片**
  - 基于**自主可控 180nm** 工艺流片
  - 芯片面积 5mm × 5mm
  - 板级典型功耗 **2000mw@150Mhz**
  - LQFP 176 封装

芯片封装视图



芯片标记

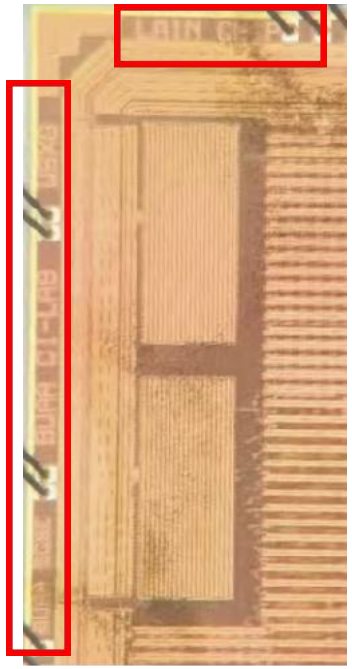


芯片内部模块划分

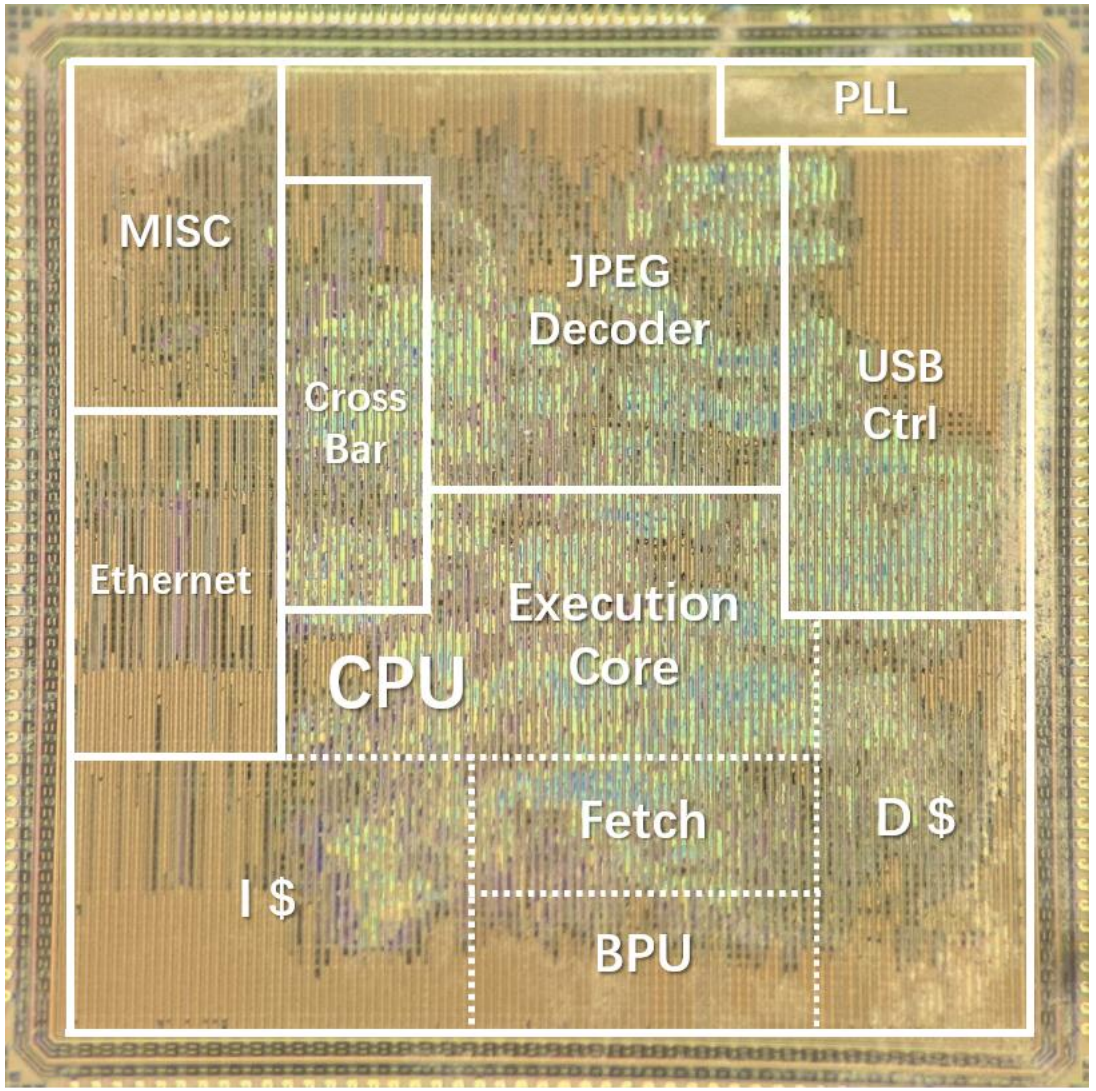
2.4 芯片敏捷开发成果展示

- **Lain 处理器芯片**
  - 基于自主可控 180nm 工艺流片
  - 芯片面积 5mm × 5mm
  - 板级典型功耗 2350mw@150Mhz
  - LQFP 176 封装

芯片封装视图



芯片标记



芯片内部模块划分

汇报提纲

一 项目背景

二 总体规划

三 实践案例

四 总结与展望

3.1 EuLA-Env 项目结构

- **开发环境集成**
  - AM
  - Difftest
  - GNU 工具链
  - NEMU
  - EuLA Core
- **一键配置**
  - sudo bash setup-tools.sh
  - source setup.sh

```
eula-env/
├── am
├── env.sh
├── eulacore
│   ├── core
│   └── difftest
├── install-verilator.sh
├── la32r-toolchains
├── nemu
├── setup-tools.sh
├── setup.sh
└── smic-sram
```

```bash
#!/bin/bash

# This script will setup eula develop environment automatically

# Init submodules
git submodule update --init --recursive

# Setup eula-env environment variables
source env.sh
# OPTIONAL: export them to .bashrc

echo PROJECT_ROOT: ${PROJECT_ROOT}
echo NEMU_HOME: ${NEMU_HOME}
echo AM_HOME: ${AM_HOME}
echo NOOP_HOME: ${NOOP_HOME}
echo LA32RTC_HOME: ${LA32RTC_HOME}


cd ${NEMU_HOME}
make la32-reduced-ref_defconfig
make -j

cd ${AM_HOME}/apps/coremark
make ARCH=la32r-eula

cd ${PROJECT_ROOT}
```

3.1 EuLA-Env 快速配置

- **快速配置**
  - **1. 克隆仓库**
    - git clone <path> --recursive
  - **2. 安装工具链（eula-env 目录，可选）**
    - source setup-tools.sh
  - **3. 加载环境变量（env-env 目录）**
    - source env.sh
  - **4. 编译 NEMU（eula-env/nemu/NEMU 目录）**
    - make la32-reduced-ref_defconfig
    - make
  - **5. 编译 AM（以 eula-env/am/apps/linux-hello 目录为例）**
    - make ARCH=la32r-eula
  - **6. 编译仿真顶层**
    - make verilog # 默认 SOC=eulasoc PROD=top

AXI4 接口

```
mycpu_mega_top core (
  .awready(core_awready),
  .awvalid(core_awvalid),
  .awaddr(core_awaddr),
  .awprot(core_awprot),
  .awid(core_awid),
  .awuser(core_awuser),
  .awlen(core_awlen),
  .awsize(core_awsize),
  .awburst(core_awburst),
  .awlock(core_awlock),
  .awcache(core_awcache),
  .awqos(core_awqos),
  .wready(core_wready),
  .wvalid(core_wvalid),
  .wdata(core_wdata),
  .wstrb(core_wstrb),
  .wlast(core_wlast),
  .bready(core_bready),
  .bvalid(core_bvalid),
  .bresp(core_bresp),
  .bid(core_bid),
  .buser(core_buser),
  .arready(core_arready),
  .arvalid(core_arvalid),
  .araddr(core_araddr),
  .arprot(core_arprot),
  .arid(core_arid),
  .aruser(core_aruser),
  .arlen(core_arlen),
  .arsize(core_arsize),
  .arburst(core_arburst),
  .arlock(core_arlock),
  .arcache(core_arcache),
  .arqos(core_arqos),
  .rready(core_rready),
  .rvalid(core_rvalid),
  .rresp(core_rresp),
  .rdata(core_rdata),
  .rlast(core_rlast),
  .rid(core_rid),
  .ruser(core_ruser),
  .intrpt(core_intrpt),
  .aclk(core_aclk),
  .aresetn(core_aresetn),
  .global_reset(core_global_reset)
);
```

3.1 EuLA-Env 快速配置

```
[    2.884000] mousedev: PS/2 mouse device common for all mice
[    3.004000] IR MCE Keyboard/mouse protocol handler initialized
[    3.108000] hid: raw HID events driver (C) Jiri Kosina
[    3.292000] NET: Registered PF_INET6 protocol family
[    3.468000] Segment Routing with IPv6
[    3.572000] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[   16.536000] Warning: unable to open an initial console.
[   17.336000] Freeing unused kernel image (initmem) memory: 2800K
[   17.388000] This architecture does not have kernel memory protection.
[   17.436000] Run /init as init process
[   17.484000]   with arguments:
[   17.532000]     /init
[   17.580000]   with environment:
[   17.628000]     HOME=/
[   17.676000]     TERM=linux
Core 0: HIT GOOD TRAP at pc = 0x10680
total guest instructions = 304,907,395
instrCnt = 304,907,395, cycleCnt = 852,005,261, IPC = 0.357870
Seed=366 Guest cycle spent: 852,005,263 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 1,463,336ms
Job succeeded
```

7. 适配核心

- 核心对外为 AXI4 接口，具体定义参见 SimTop
- 在核心中实例化 Difftest 相应模块并连线
- 在 SimTop 文件 include 核心顶层模块对应文件

8. 编译 emu（eula-env/eulacore 目录）

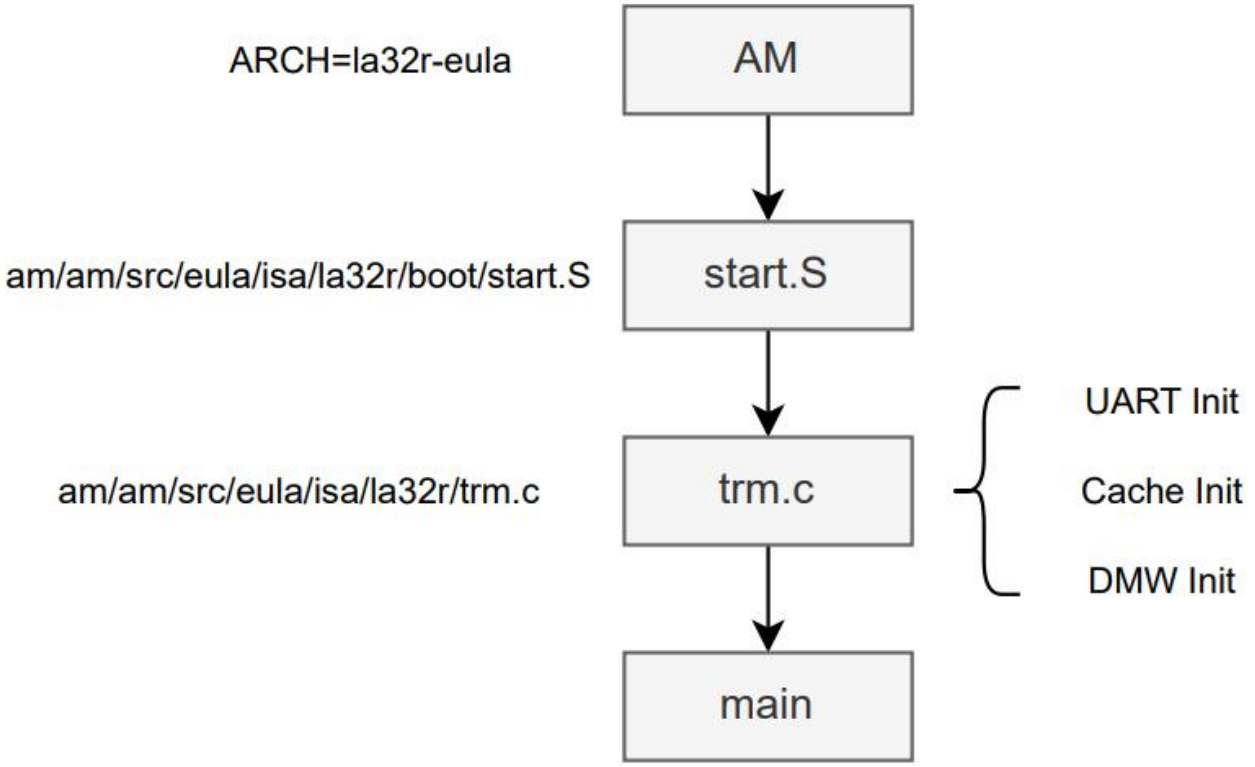
- make emu EMU_TRACE = 1

9. 仿真测试运行（eula-env/eulacore/buid 目录）

- ./emu -i $(AM_HOME)/apps/linux-hello/build/linux.vlog

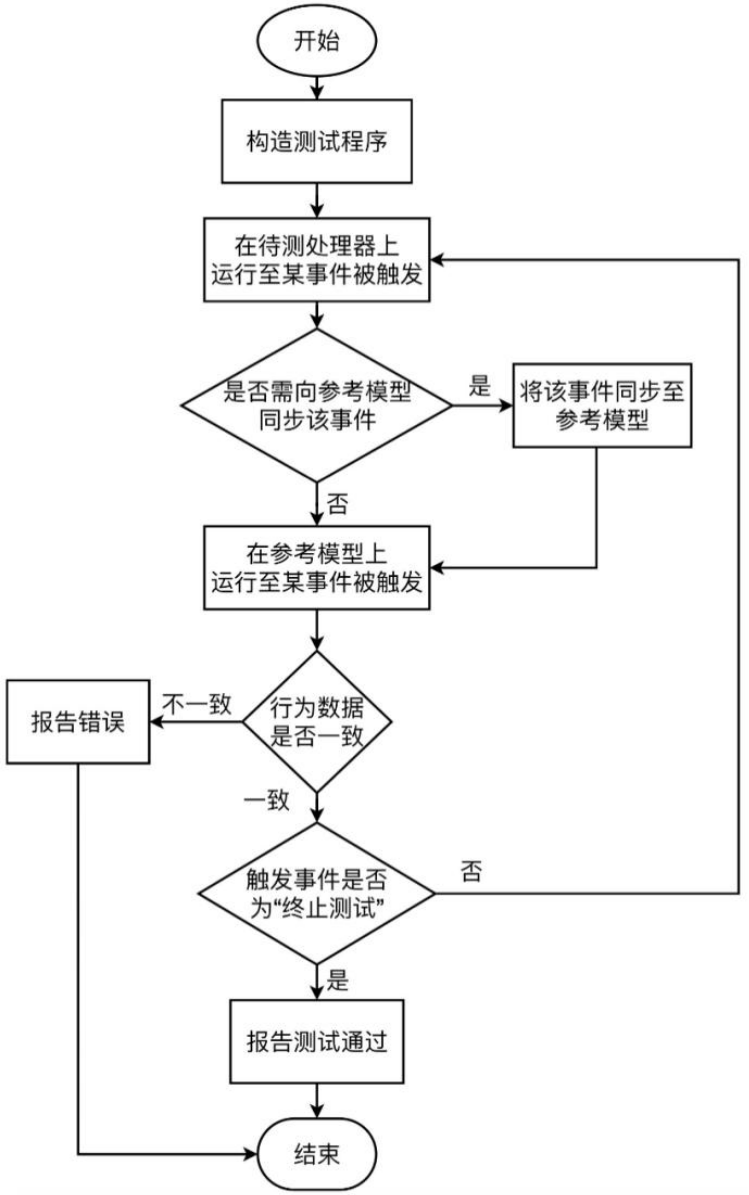
3.2 EuLA-Env 测试添加

- **编写测试（以 am/tests/cachetest/tests/*.c 为例）**
  - **编写主程序**
    - 添加 *.c 文件并编写 main 函数
    - 引用头文件
      - #include <am.h>
      - #include <klib.h>
      - #include <klib-macros.h>
  - **编译测试（am/tests/cachetest/tests 目录）**
    - make ARCH=la32r-eula



3.3 DiffTest验证平台：简介

- **处理器验证功能**
  - 体系结构模拟器与待测模型的逐拍仿真验证
  - 移植自香山团队面向RISC-V的差分测试框架
  - 增加TLB表项比对等关键功能
- **关键思想：解耦测试平台与微架构设计**
  - 只检查指令集架构指定的信息是否出错
  - 为不同的微架构设计提供统一测试平台
  - 符合指令集规范的处理器可即接即用
  - 无需自行开发验证工具，只需进行适配



3.3 DiffTest验证平台：使用方法

- **验证工程的目录组织结构**
  - ./build：Chisel构建得到的Verilog代码，该目录自动生成
  - ./difftest：使用者将DiffTest仓库克隆至此处
  - ./src：存放待测处理器的Chisel代码

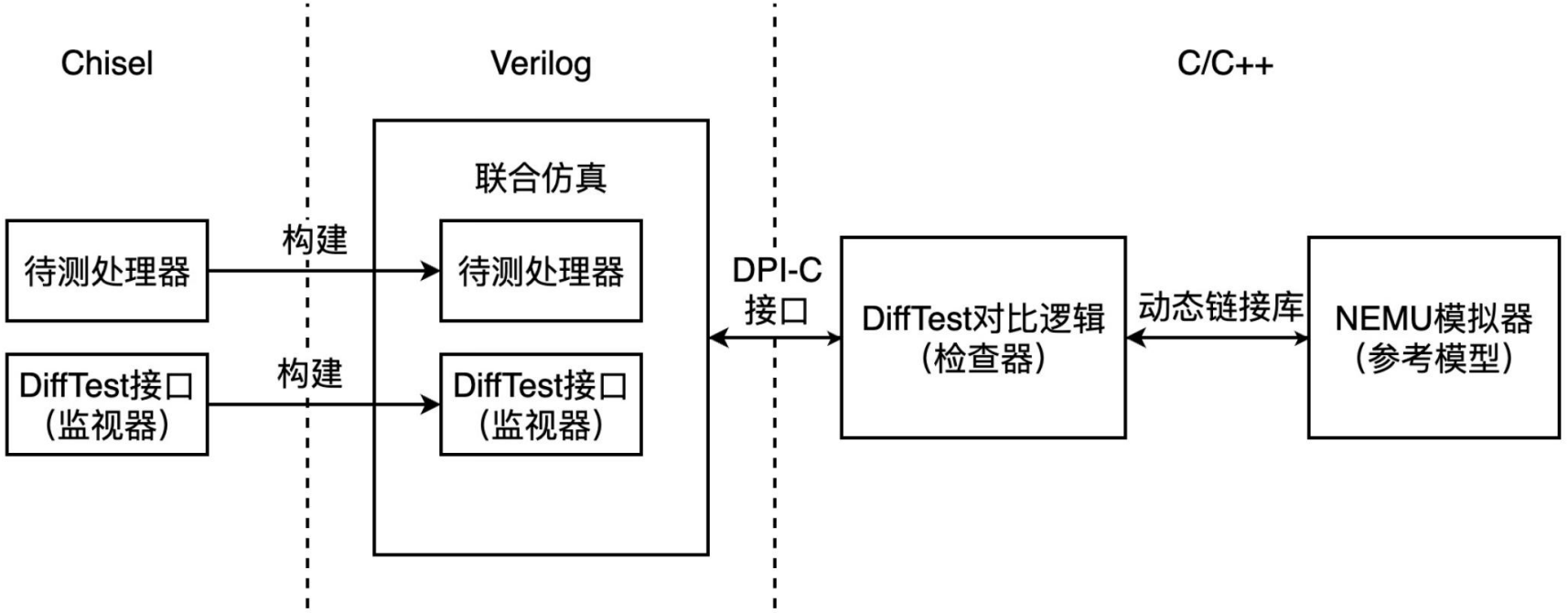
```
eulacore/
├── build
├── difftest
|   ├── config
|   ├── doc
|   └── src
└── src
    └── main
```

- **待测处理器接入方法**
  - 实例化并连接名为Difftest*的模块
  - 可使用Chisel提供的“<>”运算符，将Bundle进行整体连接
  - 需要保证传入DiffTest框架的信号，在指令提交时其产生的影响恰好生效
  - 必要时需使用RegNext延迟信号

```
// eulacore
import difftest._
if (!p.FPGAPlatform) {
  val difftest_commit = Module(new DifftestInstrCommit)
  difftest_commit.io.clock := clock
  difftest_commit.io.valid := RegNext(io.in.valid)
  difftest_commit.io.pc    := RegNext(o.in.bits.decode.cf.pc)
  // ...
}
```

3.3 DiffTest验证平台：组成部分

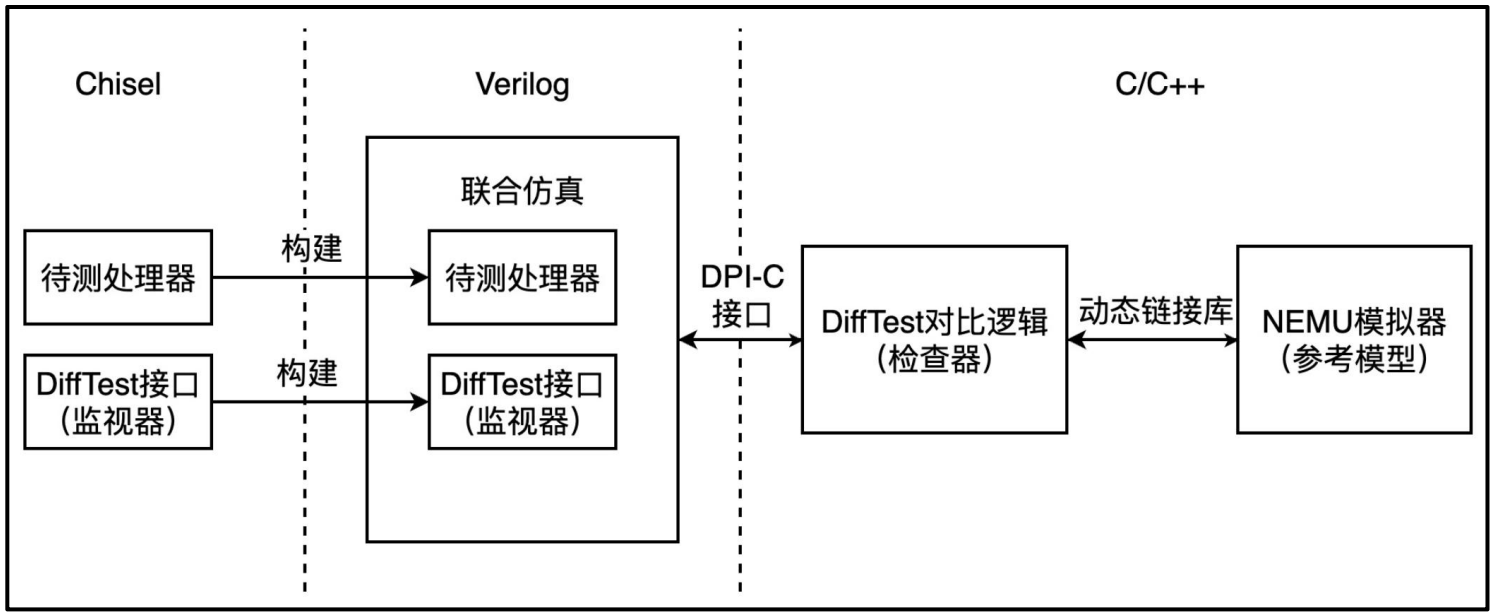
- **DiffTest平台的组成部分**
  - 待测处理器接口：连接待测处理器与DiffTest提供的Chisel模块接口
  - Verilog中间代码：待测处理器和DiffTest接口构建为用于仿真的Verilog代码
  - 对比逻辑：Verilog编写的逐周期仿真控制及体系结构对比模块
  - 参考模型：将NEMU模拟器以动态链接库形式封装，作为金标准



3.3 DiffTest验证平台：迭代开发

- 何时需对DiffTest进行迭代开发
  - **禁用或新增处理器状态的对比**
    - 访存写操作测试（store）
    - 打开或关闭CSR中某些特定寄存器的测试
    - 增加测试项目
  - **更改待测处理器和模拟器的对比逻辑**
    - 待测处理器传入信号与模拟器的时序同步
    - 例外响应的适配
    - 自定义比对逻辑
  - **修改输出信息**
    - 增加或减少日志信息
    - 适配持续集成环境
  - **支持其他指令集架构**

- 案例演示：增加TLB表项测试功能
  - **DUT->DiffTest状态汇报**
    - 增加待测处理器Chisel接口
    - 提供收集TLB表项信息的DPI-C接口
    - 在DiffTest中储存TLB相关数据
  - **DiffTest内进行DUT-REF状态对比**
  - **REF<>DiffTest双向信息同步**



3.3 DiffTest验证平台：待测处理器接口

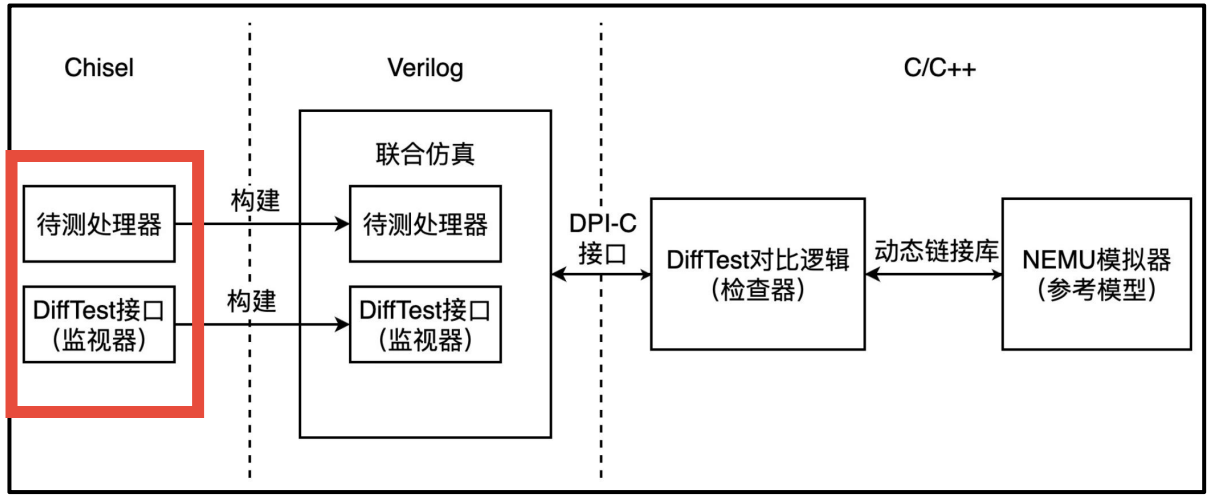
- 待测处理器接口
  - 通过继承抽象类DifftestBundle，实现待测处理器与DiffTest的接口
  - 使用继承关系，可对接口进行模块化扩展
- 增加TLB表项接口
  - 待测处理器需向DiffTest报告表项 (hi, lo0, lo1)
  - 混入DifftestWithIndex特质，以传递所有表项

```scala
// src/main/scala/Difftest.scala
trait DifftestWithIndex { val index = Input(UInt(8.W)) }
abstract class DifftestBundle extends Bundle
  with DifftestParameter
  with DifftestWithClock
  with DifftestWithCoreid

class DiffLa32rTlbEntryIO extends DifftestBundle with DifftestWithIndex {
  val entryhi = Input(UInt(64.W))
  val entrylo0 = Input(UInt(32.W))
  val entrylo1 = Input(UInt(32.W))
}
```

- 待测处理器连接新增端口
  - DifftestBundle是Bundle的子类
  - 可用foreach对所有表项进行连接

```scala
// eulacore
(0 until tlbEntryNum).foreach{ i =>
  val diffTlbEntry = Module(new DifftestLa32rTlbEntry)
  diffTlbEntry.io.clock := clock
  diffTlbEntry.io.index := i.U
  diffTlbEntry.io.coreid := 0.U
  diffTlbEntry.io.entryhi := tlbEntrys(i).tlbhi.asUInt()
  diffTlbEntry.io.entrylo0 := tlbEntrys(i).tlblo0.asUInt()
  diffTlbEntry.io.entrylo1 := tlbEntrys(i).tlblo1.asUInt()
}
```



3.3 DiffTest验证平台：Verilog中间代码

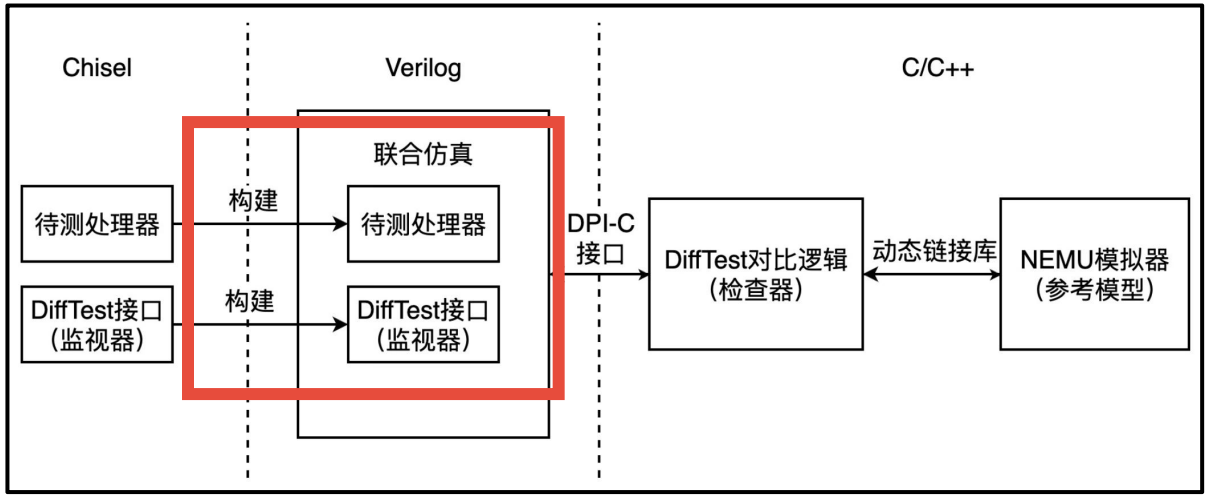
- 接口的构建
  - 使用Chisel自动生成接口的Verilog代码
  - 代码见src/main/scala/Difftest.scala

```
module DifftestLa32rEstatState(
    input io_clock,
    input [ 7:0] io_coreid,
    input [31:0] io_estat ,
    input [31:0] io_wmask );
`ifndef SYNTHESIS
`ifdef DIFFTEST

import "DPI-C" function void v_difftest_La32rEstatState (
    input byte io_coreid,
    input int io_estat,
    input int io_wmask);
    always @(posedge io_clock) begin
        v_difftest_La32rEstatState (io_coreid, io_estat, io_wmask);
    end

`endif
`endif
endmodule
```

- 参与仿真的Verilog代码
  - 待测处理器（构建而成）
  - 含有DPI-C调用的DiffTest接口（构建而成）
    - 该接口与待测处理器直接相连
    - 每个时钟周期通过DPI-C接口调用相应DiffTest函数，汇报待测处理器的状态
  - DiffTest提供的仿真顶层



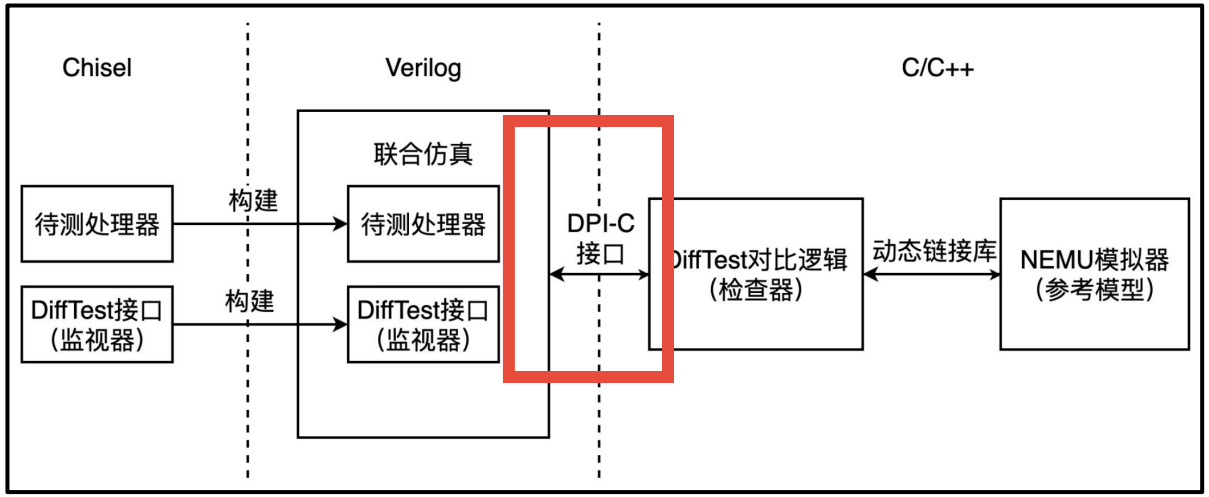
3.3 DiffTest验证平台：DUT->DiffTest信息传递

- DiffTest框架提供的DPI-C接口函数
  - 供生成的接口Verilog代码仿真时调用
  - 收集待测处理器状态信息

```
// src/src/test/csrc/difftest/interface.h
#define INTERFACE_TLBENTRY                          \
  DIFFTEST_DPIC_FUNC_DECL(La32rTlbEntry) (          \
    DPIC_ARG_BYTE coreid,                           \
    DPIC_ARG_BYTE index,                            \
    DPIC_ARG_LONG entryhi,                          \
    DPIC_ARG_INT  entrylo0,                         \
    DPIC_ARG_INT  entrylo1                          \
  )
// src/src/test/csrc/difftest/interface.cpp
INTERFACE_TLBENTRY {
  RETURN_NO_NULL;
  auto packet = difftest[coreid]->get_la32r_tlb_entry(index);
  packet->entryhi = entryhi;
  packet->entrylo0 = entrylo0;
  packet->entrylo1 = entrylo1;
}
```

- 增加TLB表项传递函数
  - 仿照interface.h增加函数
  - 在Difftest类中返回dut内相关状态域的引用
  - 函数中更新dut内部状态

```
// src/test/csrc/difftest/difftest.h
class Difftest {
  inline la32r_tlb_entry_t *get_la32r_tlb_entry(uint8_t index) {
    return &(dut.la32r_tlb_entrys[index]);
  }
  // ...
}
```



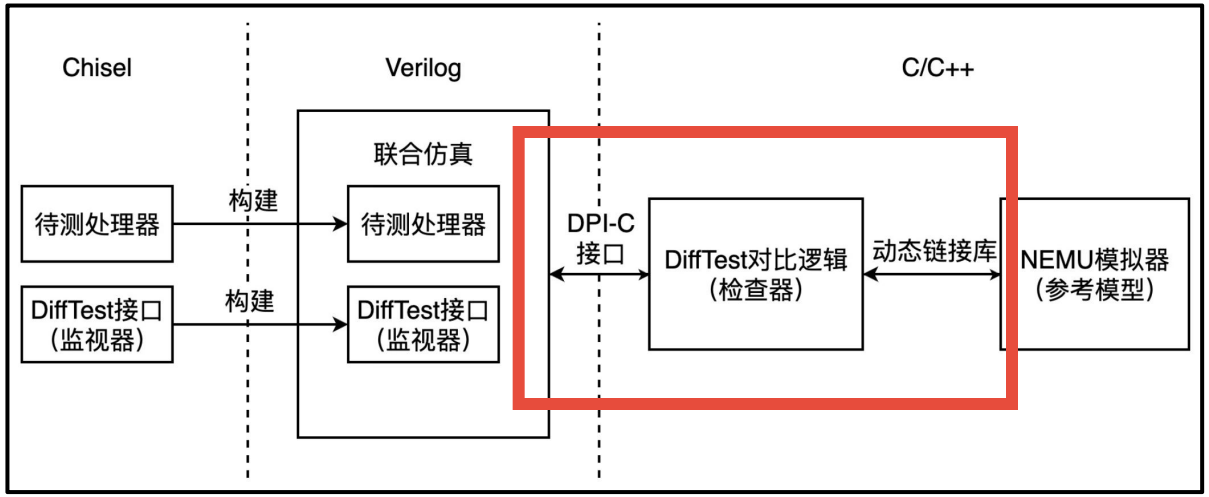
3.3 DiffTest验证平台：DUT-REF状态对比

- REF–DUT体系结构状态对比
  - DiffTest框架对比逻辑核心：Difftest类
  - 类中储存dut和ref状态
  - 通过step()比对状态一致性

```
// src/test/csrc/difftest/difftest.h
class Difftest {
  difftest_core_state_t dut;
  difftest_core_state_t ref;
  DIFF_PROXY *proxy = NULL;
  virtual int step();
  // ...
}
```

- 状态对比的迭代开发
  - 修改step()方法
  - 通过掩码打开或关闭寄存器比对

```
// src/test/csrc/difftest/difftest.cpp
static const char compare_mask[DIFFTEST_NR_CSRREG] = {
    1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
    1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
    0,  1,  1,  1,  1,  0,  0 // dose not diff TVAL & ESTAT & PC
};
```



3.3 DiffTest验证平台：DUT-REF状态对比

- **Difftest::step()函数**

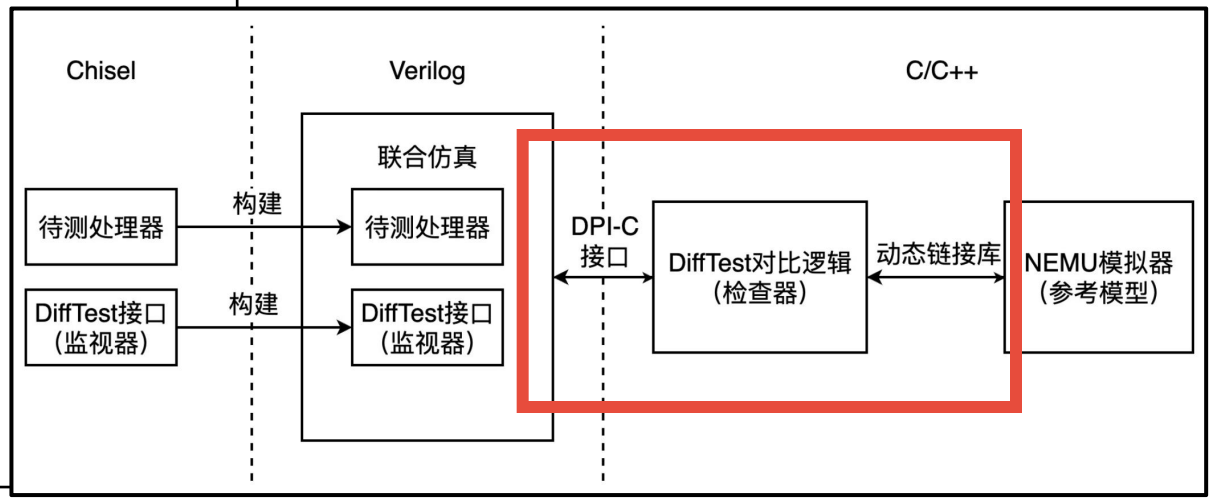
```
// src/test/csrc/difftest/difftest.cpp
int Difftest::step() {
  do_first_instr_commit();
  // handle exceptions
  do_instr_commit();

  proxy->regcpy(ref_regs_ptr, REF_TO_DUT, true); // get regs from REF
  if (memcmp(dut_regs_ptr, ref_regs_ptr, /* ... */)) {} // compare regs

  if (ENABLE_CSR_DIFF) { /* ... */ } // compare CSR

  // compare TLB entries
  if (ENABLE_TLB_DIFF && tlbModify) {
    for (int i = 0; i < LA32R_TLB_ENTRY_NUM; i++) {
      proxy->tlbcpy(i, &(ref.la32r_tlb_entrys[i]));
    }
    bool consistent = !memcmp(/* ... */)
  }
  // ...
}
```

- **do_first_commit：执行第一条指令时调用，可初始化模拟器**
- **do_instr_commit：记录指令流**
- **proxy：模拟器的动态链接库代理**
- **从模拟器拷贝数据，用memcmp对比模拟器与待测处理器的状态一致性**

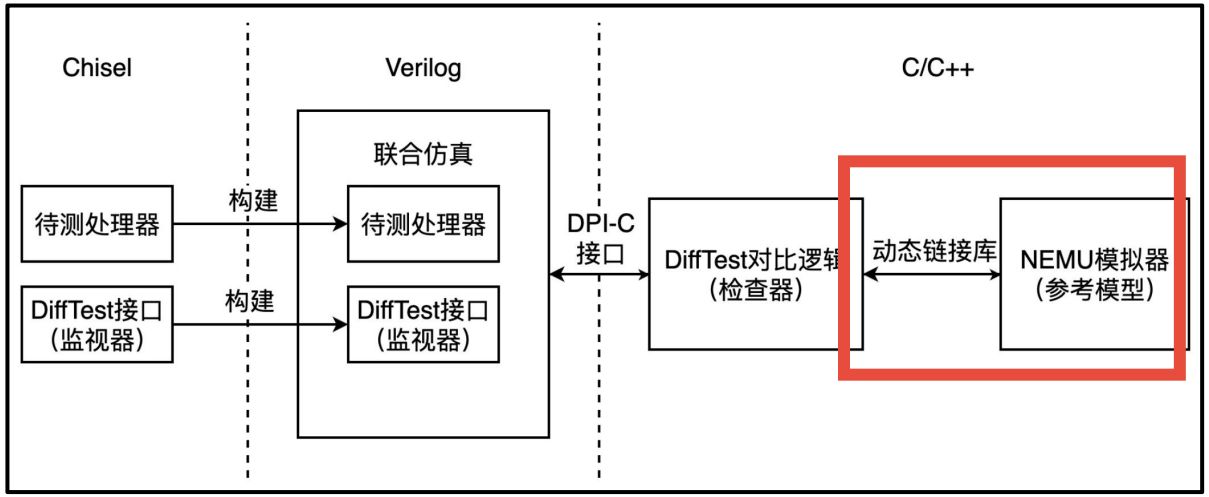


3.3 DiffTest验证平台：REF->DiffTest信息传递

- **REF–DiffTest双向信息同步**
  - 在模拟器中添加函数，返回或设置相关状态值
  - DiffTest中使用dlsym加载动态链接库
  - step()调用模拟器函数进行状态获取和状态同步
- **获取模拟器的TLB表项数据**
  - step()中调用tlbcpy方法从模拟器拷贝TLB表项
  - tlbcpy是函数指针，指向模拟器中的difftest_tlbcpy_to_difftest函数
  - 该函数返回模拟器中的TLB表项数据的引用
  - step()函数后续需对比REF和DUT中表项的一致性

```cpp
// src/test/csrc/difftest/difftest.cpp
int Difftest::step() {
  // ...
    for (int i = 0; i < LA32R_TLB_ENTRY_NUM; i++) {
      proxy->tlbcpy(i, &(ref.la32r_tlb_entrys[i]));
    }
  // ...
}
```

```cpp
// src/test/csrc/difftest/refproxy.cpp
tlbcpy = (void (*)(uint32_t, void*))dlsym(
    handle, "difftest_tlbcpy_to_difftest");
check_and_assert(tlbcpy);
```



3.4 MegaSoC

- **快速配置**
  - **1. 克隆仓库**
    - https://github.com/BUAA-CI-LAB/MegaSoC
  - **2. 使用 Vivado 打开工程**
  - **3. 导入自行设计的 CPU 核**
  - **4. 构建比特流**

AXI4 Peripheral Address Mapping:

Address Range	Device	Select Value
`0x00000000 - 0x0FFFFFFF`	Memory Controller	0
`0x1C000000 - 0x1C0FFFFF` `0x1D000000 - 0x1D00FFFF`	SPI Device	1
`0x1D000000 - 0x1D0FFFFF`	AXI-Lite Device	2
`0x1D100000 - 0x1D1FFFFF`	USB Controller	3
Other Addresses	AXI-Lite Device (Default)	2

AXI4-Lite Peripheral Address Mapping:

Address Range	Device	Select Value
`0x1D100000 - 0x1D3FFFFF`	APB Devices (UART, I2C, CDBUS)	1
`0x1D400000 - 0x1D4FFFFF`	Configuration Registers	2
`0x1D500000 - 0x1D5FFFFF`	Ethernet Controller	3
`0x1D600000 - 0x1D6FFFFF`	Interrupt Controller	4
`0x1D700000 - 0x1D7FFFFF`	SD Controller	5
`0x1DA00000 - 0x1DAFFFFF`	JPEG Controller	6
`0x1DB00000 - 0x1DBFFFFF`	I2S Controller-0	7
`0x1DC00000 - 0x1DCFFFFF`	I2S Controller-1	8
`0x1DD00000 - 0x1DDFFFFF`	VGA Controller	9
Other Addresses	SRAM Controller (Default)	0

3.5 系统软件编译流程

- **快速配置**
  - **1. 克隆仓库**
    - git clone git@github.com:BUAA-CI-LAB/u-boot.git
    - git clone git@github.com:BUAA-CI-LAB/linux.git
  - **2. 安装并引入编译工具链**
    - source ${PROJECT_HOME}/setup-tools.sh
    - source ${PROJECT_HOME}/env.sh
  - **3. 编译 u-boot**
    - export ARCH=la32r; export CROSS_COMPILE=loongarch32r-linux-gnusf-
    - make la32rmega_defconfig
    - make -j`nproc`
  - **4. 编译 Linux**
    - export ARCH=loongarch; export CROSS_COMPILE=loongarch32r-linux-gnusf-
    - make la32_defconfig（or make la32rmega_defconfig）
    - make -j`nproc`

汇报提纲

一 项目背景

二 总体规划

三 实践案例

四 总结与展望

4.1 总结与展望

- **解决的问题**
  - **开发环境配置缓慢**
    - EuLA-SoC 一键配置环境，处理器可通过 AXI4 接口快速接入仿真 SoC
  - **测试添加受限**
    - AM 支持自由编写测试，并内置针对 CSR/TLB/Cache 的功能测试以及 Coremark/Linux 等软件系统测试
  - **片上系统拓展困难**
    - MegaSoC 结构清晰，外设丰富，并通过流片验证，配套相应 Linux 配置
  - **展示环节千篇一律**
    - 跳出往年 Chiplab SoC 通过 PMON 启动 Linux 的固化展示范式

这是经过几年的积累，欢迎大家使用并反馈，我们将持续迭代完善

4.1 总结与展望

- **可改进之处**
  - **支持多核处理器的差分测试**
    - 适配北航首款 LA32R 四核乱序双发射处理器 Wired
  - **仿真支持更多外设**
    - 实现 UART / 网口等设备的仿真模型，模拟 MegaSoC 中对应设备的行为
  - **添加后端工作流**
    - 支持将 RTL 代码转化为门级网表，优化逻辑结构以提高性能并降低功耗
    - 基于布局基础进行布线，保证信号传输正确性与可靠性
    - 支持通过数学方法进行形式化验证以保证电路设计正确性
    - 支持进行 DRC 和 LVS 等物理验证，确保设计符合工艺要求

欢迎大家一起开发与完善，共同为龙芯技术社区贡献技术力量！

4.2 开源仓库

- **BHLA：北航龙架构处理器芯片敏捷开发项目**
  - **https://github.com/BUAA-CI-LAB/BHLA.README**
  - **EuLA-Env：https://github.com/BUAA-CI-LAB/eula-env**
    - AM：https://github.com/BUAA-CI-LAB/am
    - DiffTest：https://github.com/BUAA-CI-LAB/difftest
    - EuLA Core：https://github.com/BUAA-CI-LAB/eulacore
    - Lain Core：https://github.com/LainChip/LainCore
    - GNU 工具链：https://github.com/BUAA-CI-LAB/la32r-toolchains
    - NEMU：https://github.com/BUAA-CI-LAB/nemu
  - **MegaSoC：https://github.com/BUAA-CI-LAB/MegaSoC**
  - **U-Boot：https://github.com/BUAA-CI-LAB/u-boot**
  - **Linux：https://github.com/BUAA-CI-LAB/linux**
  - **Chiplab：https://github.com/BUAA-CI-LAB/chiplab**

致 谢

带着自己设计的芯片毕业

毕设团队合影

致 谢

- 龙芯中科
- Nutshell 团队
- OpenXiangShan 团队
- 南京大学 ProjectN 团队
- 其他开源贡献者（IP、系统软件等）

致 谢

衷心感谢各位老师同学！

敬请批评指正！

BHLA

技术交流微信群

相关报道

芯片工作展示视频

https://www.bilibili.com/video/BV12Ybxe7Eih

https://mp.weixin.qq.com/s/Vn9fkeU_I68M4mcQs-yg5A