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

鹏城实验室 Peng Cheng Laboratory

中国开放指令生态（RISC-V）联盟 China RISC-V Alliance

香山：用敏捷开发方法实现开源高性能RISC-V处理器

香山团队

2021/11/4@海思OpenNetChip技术论坛

开源芯片包含三个层次

① 指令集 + ② 处理器微架构设计/实现 + ③ 设计流程/工具

处理器设计新方法、新流程、新平台

开源 微架构设计 工程开发 工具

EDA工具

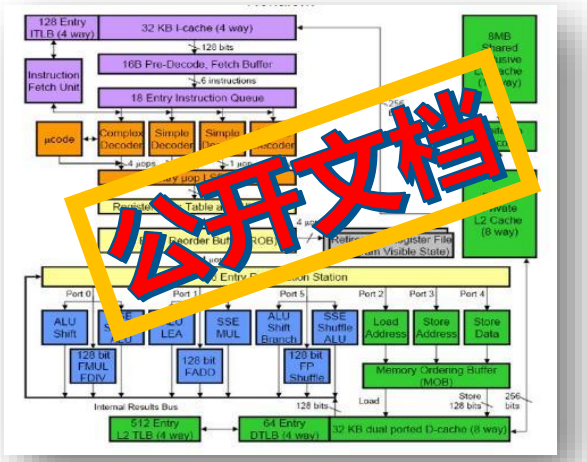
指令集手册

设计文档

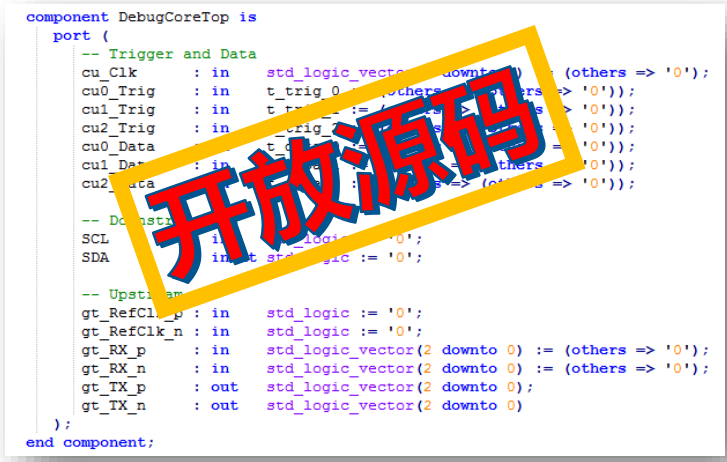
RTL代码

芯片版图

开放免费
共同制定



公开文档

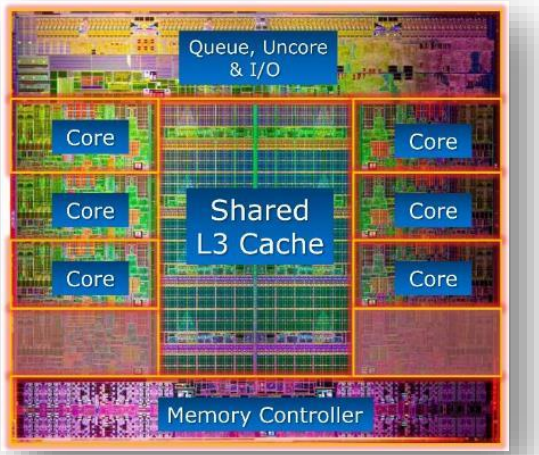


```
component DebugCoreTop is
  port (
    -- Trigger and Data
    cu_Clk      : in    std_logic_vecto     downto 0)     (others => '0');
    cu0_Trig    : in    t_trig_0     others     => '0'));
    cu1_Trig    : in    t_trig_1     others     => '0'));
    cu2_Trig    : in    t_trig_2     others     => '0'));
    cu0_Data    : in    t_data_0     others     => '0'));
    cu1_Data    : in    t_data_1     others     => '0'));
    cu2_Data    : in    t_data_2     others     => '0'));

    -- Debug Hub
    SCL         : inout std_logic := '0';
    SDA         : inout std_logic := '0';

    -- Upstream
    gt_RefClk_p : in    std_logic := '0';
    gt_RefClk_n : in    std_logic := '0';
    gt_RX_p     : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_RX_n     : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_TX_p     : out   std_logic_vector(2 downto 0);
    gt_TX_n     : out   std_logic_vector(2 downto 0)
  );
end component;
```

开放源码



为什么做开源高性能 RISC-V 核?

- **RISC-V**的一大优势——形成**“竞争前合作”**，实现各界**联合开发开源CPU架构**
- 国内外现有开源RISC-V项目**尚不满足业界对高性能处理器的需求**
- 建立像**Linux**的被工业界广泛应用的**体系结构创新开源平台——存活30年**!
- 用于研究和验证芯片**敏捷开发方法、流程与工具**

微架构设计 / 指令集	① 开放免费的设计	② 需授权的设计	③ 封闭的设计
开放免费的指令集 (RISC-V)	Berkeley的Rocket Chip/剑桥lowRISC/芯来科技蜂鸟E203	平头哥/SiFive/晶心科技Andes的RISC-V处理器核	Google和NVIDIA的自研RISC-V处理器
需授权的指令集 (ARM)		ARM的处理器设计，如Cortex-A76等	基于ARM架构的Apple处理器
封闭的指令集 (x86)			Intel和AMD的处理器

产品可选的设计 (对应各指令集)
① ② ③
② ③
③

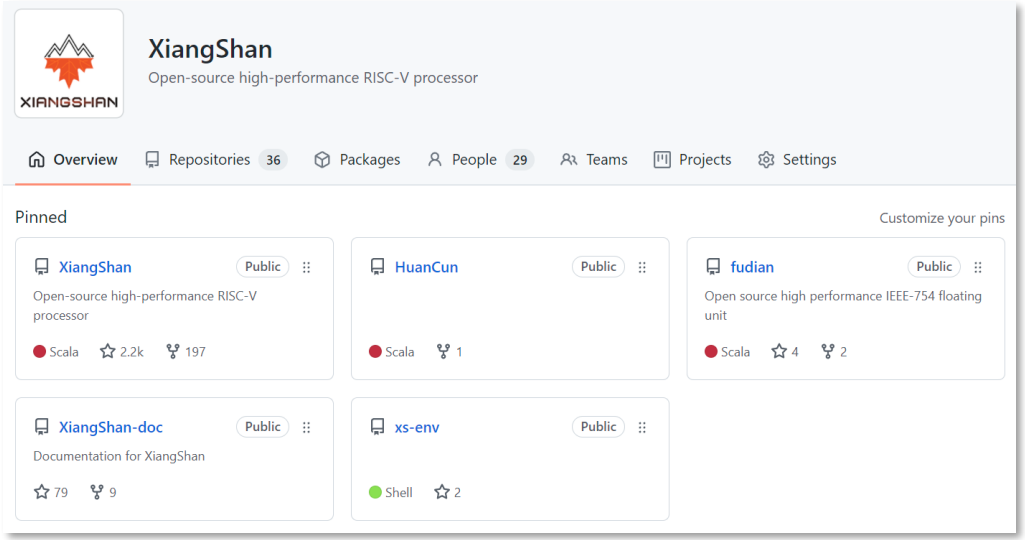
香山：开源高性能 RISC-V 处理器

- **第一版：雁栖湖架构**
  - 2020/6：代码仓库建立
  - 2021/4：RTL完成
  - 2021/6：在RISC-V中国峰会正式发布
  - 频率：1.3GHz@28nm
  - 性能：预估SPEC CPU2006 ~7分/GHz
- **第二版：南湖架构**
  - 2021/5：开始RTL实现工作，同时持续进行设计讨论
  - 2021/10：功能特性确定，进入时序收敛阶段
  - 计划：11月完成RTL freeze，22年初投片，**目标SPEC CPU2006 20 分，2GHz@14nm**
- **开源情况**
  - 开源协议：**MulanPSLv2协议**（兼容Apache v2.0）
  - 代码托管：GitHub (https://github.com/OpenXiangShan/XiangShan)；镜像：Gitee/Trustie/iHub
  - 报告/文档/进展：微信公众号、知乎、微博 香山开源处理器
  - 邮件列表：xiangshan-all@ict.ac.cn

香山源码仓库

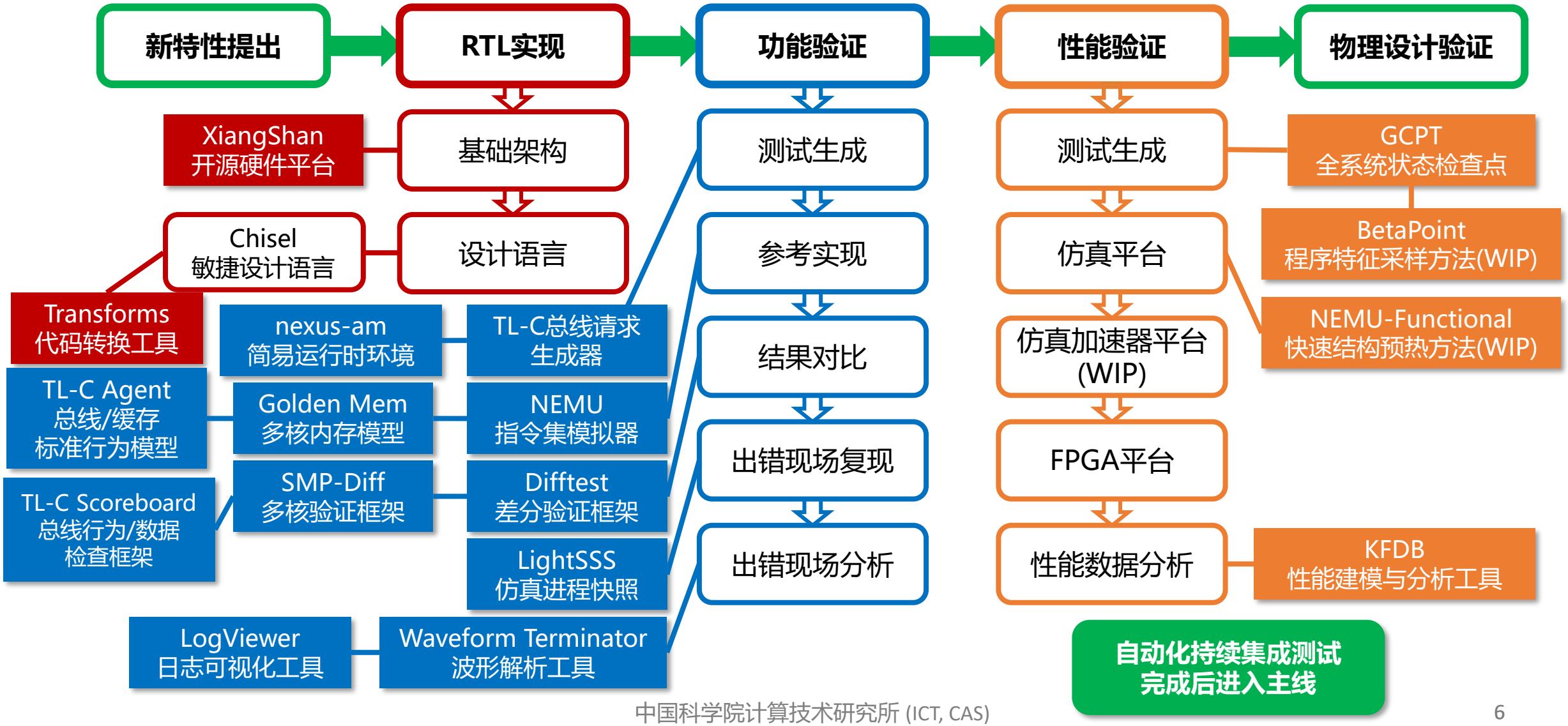
南湖：香山下一代微结构

- **改进方向①：取指与分支预测**
  - 采用解耦的前端取指与分支预测架构
  - 更高的吞吐率、更高的预测准确率
- **改进方向②：运算单元**
  - 支持 RISC-V 位操作 (B)、标量加密运算 (K) 扩展指令集
  - 开源高性能浮点运算部件 FuDian
- **改进方向③：功能支持**
  - 支持 RISC-V 物理内存保护机制 (PMP) 等安全特性
  - 支持自定义的可配置物理内存属性 (PMA)
- **改进方向④：Load Store Unit**
  - 提高 TLB、L1 Cache 等结构容量
  - 重构 L1 DCache 流水线设计，降低访问冲突，提高效率
- **改进方向⑤：L2/L3 Cache**
  - 支持 inclusive / non-inclusive 的开源高性能 L2/LLC 模块 HuanCun



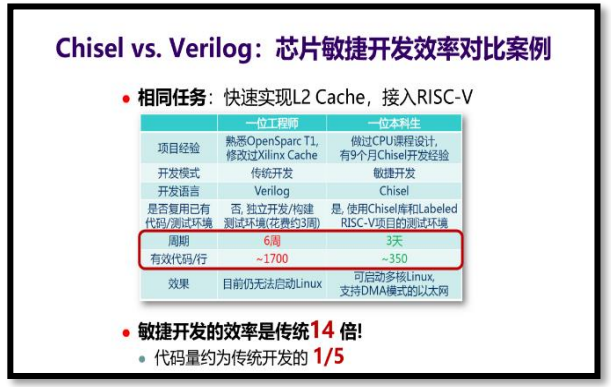
所有修改全部开源

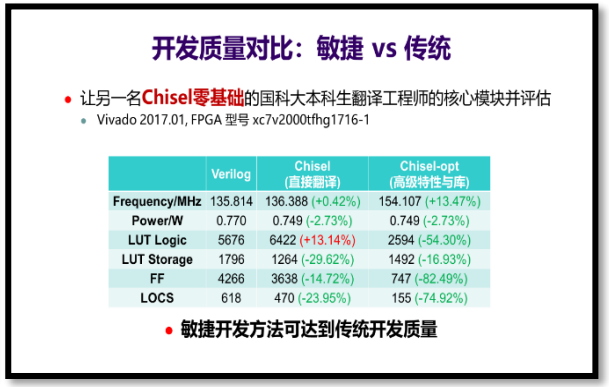
香山：高性能处理器敏捷开发实践



新型硬件设计语言带来编码效率的大幅提升

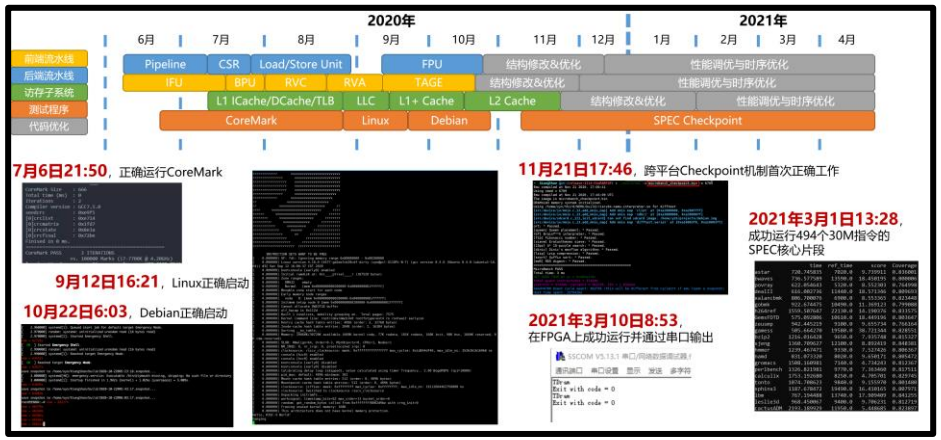
- **2018年：设计定量实验对比Chisel与Verilog，代码量降低80%**

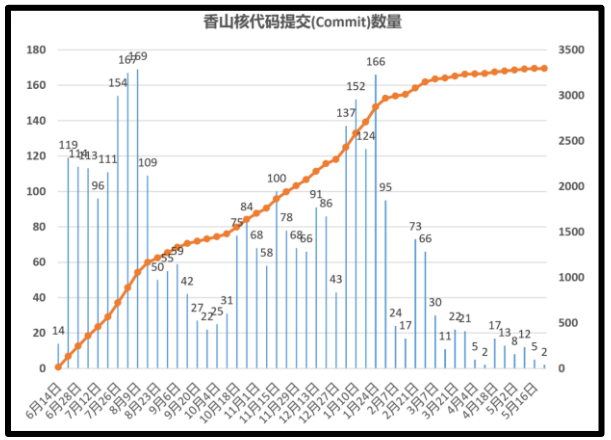




余子濠,刘志刚,李一苇,黄博文,王卅,孙凝晖,包云岗. **芯片敏捷开发实践：标签化RISC-V**. 计算机研究与发展, 2019

- **2020年：采用Chisel语言完成香山高性能处理器实现，三个月启动Linux**

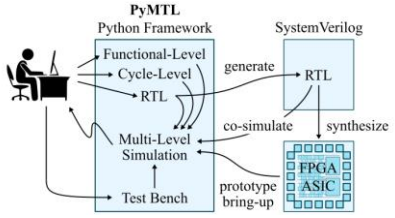


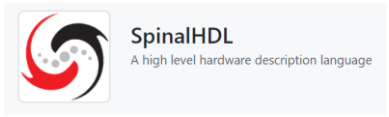


香山：开源高性能RISC-V处理器. RISC-V中国峰会, 2021

新型硬件设计语言驱动敏捷硬件设计范式变化

CHISEL





bluespec

cucapra/latte21

Languages, Tools, and Techniques for Accelerator Design

Scala　python　OCaml

Verilog　SystemVerilog

敏捷硬件设计语言

新型硬件设计范式 — **抽象层**

高抽象层次的硬件描述语言

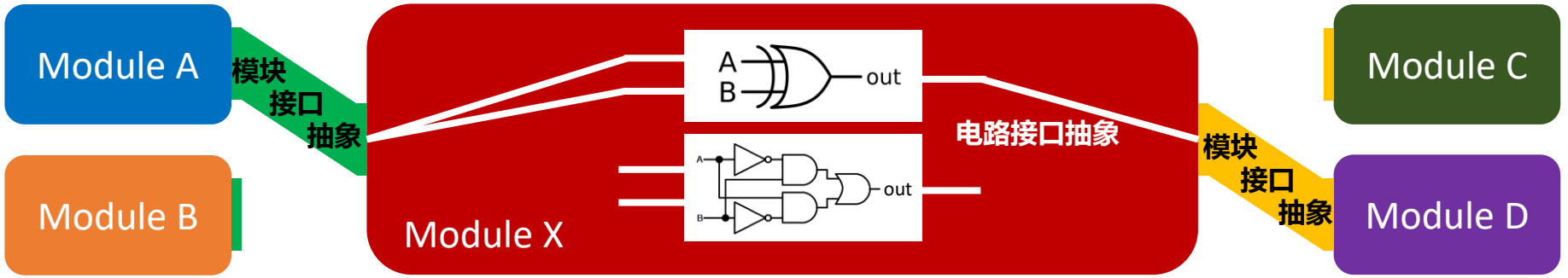
硬件描述语言 — **描述层**

处理器体系结构 — **结构层**

逻辑电路 — **电路层**

敏捷硬件设计方法

抽象层次①：模块/电路接口



- 示例：替换算法、数据校验算法

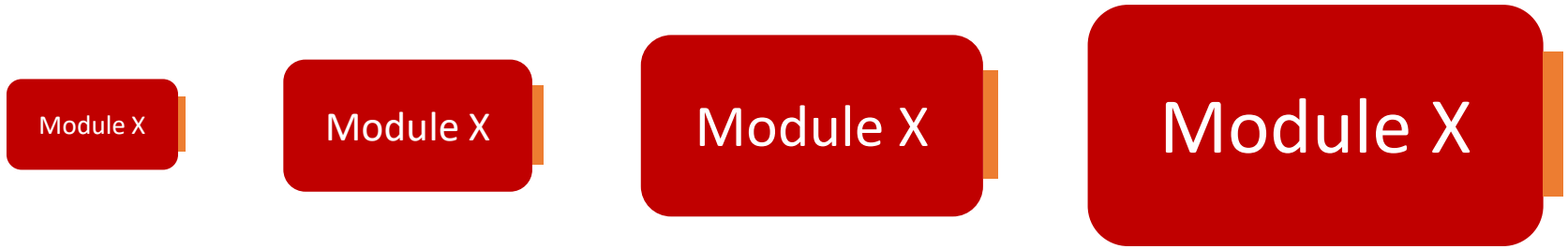
```scala
object ReplacementPolicy {
  //for fully associative mapping
  def fromString(s: Option[String],n_ways: Int): ReplacementPolicy = fromString(s.getOrEl
  def fromString(s: String, n_ways: Int): ReplacementPolicy = s.toLowerCase match {
    case "random" => new RandomReplacement(n_ways)
    case "lru"    => new TrueLRU(n_ways)
    case "plru"   => new PseudoLRU(n_ways)
    case t => throw new IllegalArgumentException(s"unknown Replacement Policy type $t")
  }
  //for set associative mapping
  def fromString(s: Option[String], n_ways: Int, n_sets: Int): SetAssocReplacementPolicy
  def fromString(s: String, n_ways: Int, n_sets: Int): SetAssocReplacementPolicy = s.toLo
    case "random"    => new SetAssocRandom(n_sets, n_ways)
    case "setlru"    => new SetAssocLRU(n_sets, n_ways, "lru")
    case "setplru"   => new SetAssocLRU(n_sets, n_ways, "plru")
    case t => throw new IllegalArgumentException(s"unknown Replacement Policy type $t")
  }
}
```

(1) 实现不同的替换算法

```scala
icacheParameters: ICacheParameters = ICacheParameters(
  tagECC = Some("parity"),
  dataECC = Some("parity"),
  replacer = Some("setplru"),
  nMissEntries = 2
),
l1plusCacheParameters: L1plusCacheParameters = L1plusCacheParameters(
  tagECC = Some("secded"),
  dataECC = Some("secded"),
  replacer = Some("setplru"),
  nMissEntries = 8
),
dcacheParameters: DCacheParameters = DCacheParameters(
  tagECC = Some("secded"),
  dataECC = Some("secded"),
  replacer = Some("setplru"),
  nMissEntries = 16,
  nProbeEntries = 16,
  nReleaseEntries = 16,
  nStoreReplayEntries = 16
),
```

(2) 基于Scala语言完成算法的实例化

抽象层次②：基础硬件模块/数据结构



- 示例：不同大小的硬件队列（包括队列指针、数据存储空间等）

```scala
class CircularQueuePtr[T <: CircularQueuePtr[T]](val entries: Int) extends Bundle {

  def this(f: Parameters => Int)(implicit p: Parameters) = this(f(p))

  val PTR_WIDTH = log2Up(entries)
  val flag = Bool()
  val value = UInt(PTR_WIDTH.W)
```

(1) 定义一个参数化的队列指针类型

```scala
class FtqPtr(implicit p: Parameters) extends CircularQueuePtr[FtqPtr](
  p => p(XSCoreParamsKey).FtqSize
){
  override def cloneType = (new FtqPtr).asInstanceOf[this.type]
}

object FtqPtr {
  def apply(f: Bool, v: UInt)(implicit p: Parameters): FtqPtr = {
    val ptr = Wire(new FtqPtr)
    ptr.flag := f
    ptr.value := v
    ptr
  }
}
```

(2) 基于队列大小参数继承
得到对应队列的指针类型

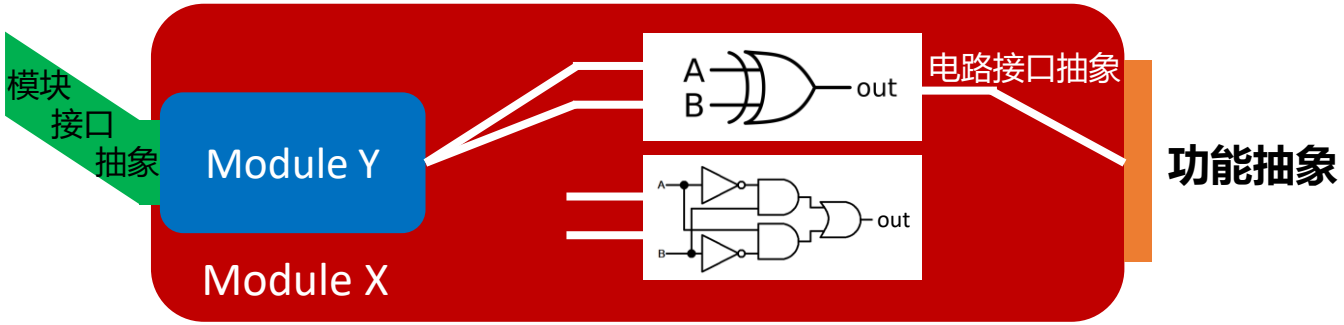
```scala
// multi-write
val update_target = Reg(Vec(FtqSize, UInt(VAddrBits.W)))
val cfiIndex_vec = Reg(Vec(FtqSize, ValidUndirectioned(UInt(log2Up(PredictWidth).W))))
val cfiIsCall, cfiIsRet, cfiIsJalr, cfiIsRVC = Reg(Vec(FtqSize, Bool()))
val mispredict_vec = Reg(Vec(FtqSize, Vec(PredictWidth, Bool())))
```

(3) 基于队列大小参数来例化数据存储

```scala
when(real_fire) {
  val enqIdx = tailPtr.value
  commitStateQueue(enqIdx) := VecInit(io.enq.bits.valids.map(v => Mux(v, s_valid, s_invalid)))
  cfiIndex_vec(enqIdx) := io.enq.bits.cfiIndex
  cfiIsCall(enqIdx) := io.enq.bits.cfiIsCall
  cfiIsRet(enqIdx) := io.enq.bits.cfiIsRet
  cfiIsJalr(enqIdx) := io.enq.bits.cfiIsJalr
  cfiIsRVC(enqIdx) := io.enq.bits.cfiIsRVC
  mispredict_vec(enqIdx) := WireInit(VecInit(Seq.fill(PredictWidth)(false.B)))
  update_target(enqIdx) := io.enq.bits.target
}
```

(4) 使用参数化的队列指针完成
对数据存储的操作

抽象层次③：具有功能抽象的参数化模块设计



- 示例：一个参数化的发射队列

```
case class RSParams
(
  var numEntries: Int = 0,
  var numEnq: Int = 0,
  var numDeq: Int = 0,
  var numSrc: Int = 0,
  var dataBits: Int = 0,
  var dataIdBits: Int = 0,
  var numFastWakeup: Int = 0,
  var numWakeup: Int = 0,
  var hasFeedback: Boolean = false,
  var delayedRf: Boolean = false,
  var fixedLatency: Int = -1,
  var checkWaitBit: Boolean = false,
  var optBuf: Boolean = false,
  // special cases
  var isJump: Boolean = false,
  var isAlu: Boolean = false,
  var isStore: Boolean = false,
  var isMul: Boolean = false,
  var isLoad: Boolean = false,
  var exuCfg: Option[ExuConfig] = None
){
  def allWakeup: Int = numFastWakeup + numWakeup
  def indexWidth: Int = log2Up(numEntries)
  // oldestFirst: (Enable_or_not, Need_balance, Victim_index)
  def oldestFirst: (Boolean, Boolean, Int) = (true, !isLoad, if (isLoad) 0
  def hasMidState: Boolean = exuCfg.get == FmacExeUnitCfg
  def needScheduledBit: Boolean = hasFeedback || delayedRf || hasMidState
  def needBalance: Boolean = exuCfg.get.needLoadBalance
```

(1) 确定发射队列的参数需求
并定义对应的参数类型

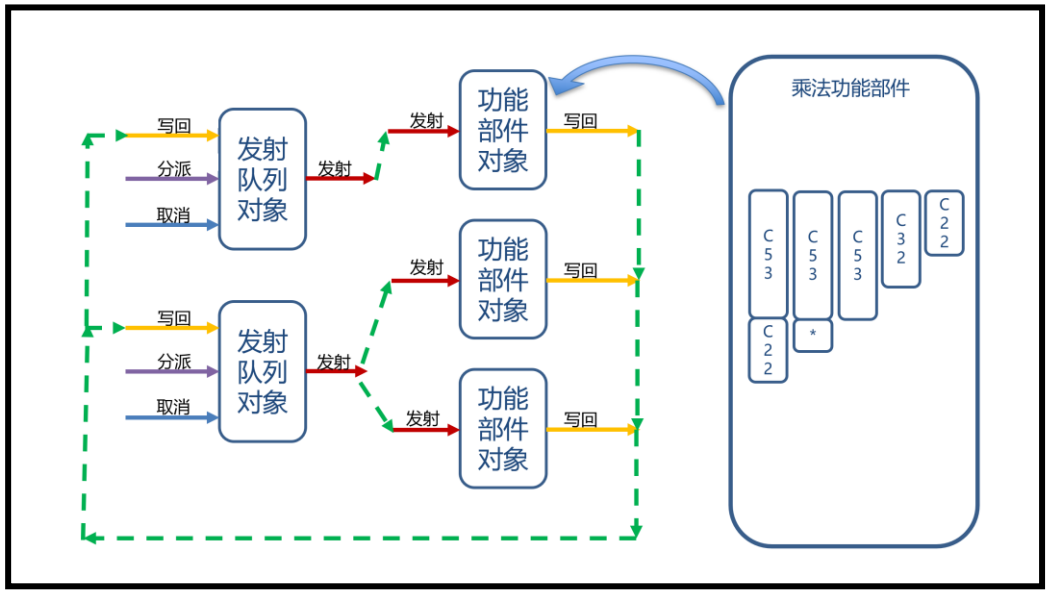
```
class StatusArray(params: RSParams)(implicit p: Parameters) extends XSModule
  with HasCircularQueuePtrHelper {
  val io = IO(new Bundle {
    val redirect = Flipped(ValidIO(new Redirect))
    // current status
    val isValid = Output(UInt(params.numEntries.W))
    val canIssue = Output(UInt(params.numEntries.W))
    val flushed = Output(UInt(params.numEntries.W))
    // enqueue, dequeue, wakeup, flush
    val update = Vec(params.numEnq, new StatusArrayUpdateIO(params))
    val wakeup = Vec(params.allWakeup, Flipped(ValidIO(new MicroOp)))
    val wakeupMatch = Vec(params.numEntries, Vec(params.numSrc, Output(UInt(params.allWakeup.W))))
    val issueGranted = Vec(params.numDeq, Flipped(ValidIO(UInt(params.numEntries.W))))
    val isFirstIssue = Vec(params.numDeq, Output(Bool()))
    val allSrcReady = Vec(params.numDeq, Output(Bool()))
    val updateMidState = Input(UInt(params.numEntries.W))
    val deqRespWidth = if (params.hasFeedback) params.numDeq * 2 else params.numDeq
    val deqResp = Vec(deqRespWidth, Flipped(ValidIO(new Bundle {
      val rsMask = UInt(params.numEntries.W)
      val success = Bool()
      val resptype = RSFeedbackType() // update credit if needs replay
      val dataInvalidSqIdx = new SqPtr
    })))
    val stIssuePtr = if (params.checkWaitBit) Input(new SqPtr()) else null
    val memWaitUpdateReq = if (params.checkWaitBit) Flipped(new MemWaitUpdateReq) else null
  })

  val statusArray = Reg(Vec(params.numEntries, new StatusEntry(params)))
```

(2) 完成模块接口、基础硬件结构、
内部逻辑的参数化实现

抽象层次④：模块对象化

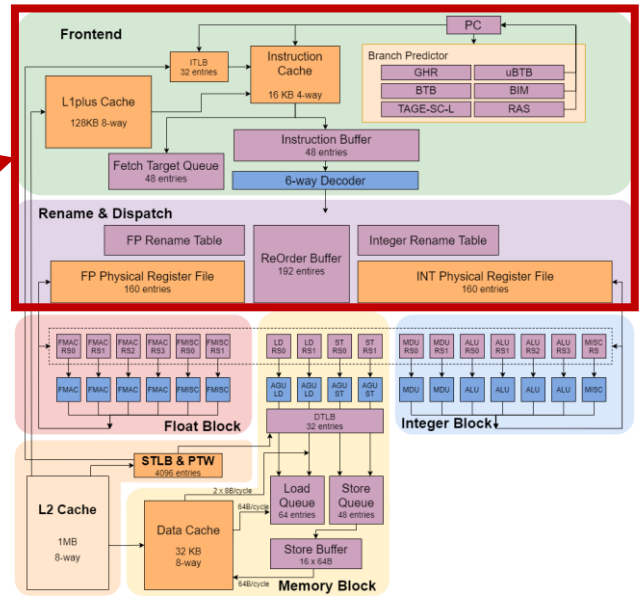
- 确定对象间交互方式之后，使用对象化的方式完成模块间组织



示例：发射队列、功能部件的对象化组织

```
// Synthesizable minimal XiangShan
// * It is still an out-of-order, super-scalaer arch
// * L1 cache included
// * L2 cache NOT included
// * L3 cache included
class MinimalConfig(n: Int = 1) extends Config(
  new DefaultConfig(n).alter((site, here, up) => {
    case SoCParamsKey => up(SoCParamsKey).copy(
      cores = up(SoCParamsKey).cores.map(_.copy(
        DecodeWidth = 2,
        RenameWidth = 2,
        FetchWidth = 4,
        IssQueSize = 8,
        NRPhyRegs = 80,
        LoadQueueSize = 16,
        StoreQueueSize = 16,
        RoqSize = 32,
        BrqSize = 8,
        FtqSize = 16,
        IBufSize = 16,
        StoreBufferSize = 4,
        StoreBufferThreshold = 3,
        dpParams = DispatchParameters(
          IntDqSize = 8,
          FpDqSize = 8,
          LsDqSize = 8,
```

All affected
But it works

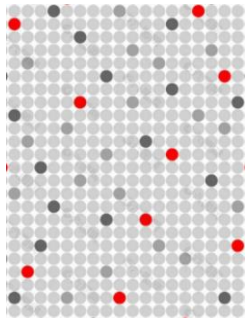


示例：将香山从六发射改为双发射

处理器敏捷设计范式

处理器

面向电路的
处理器设计



像素
(门电路)

面向电路

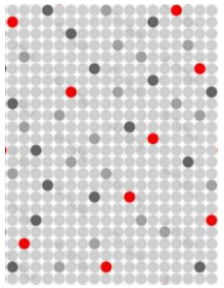
- 以门电路作为基本设计单元
- 可模块化但接口混乱，不易复用

处理器

面向对象的
处理器设计

对象
(可复用的模块)

复杂度
降低



面向对象

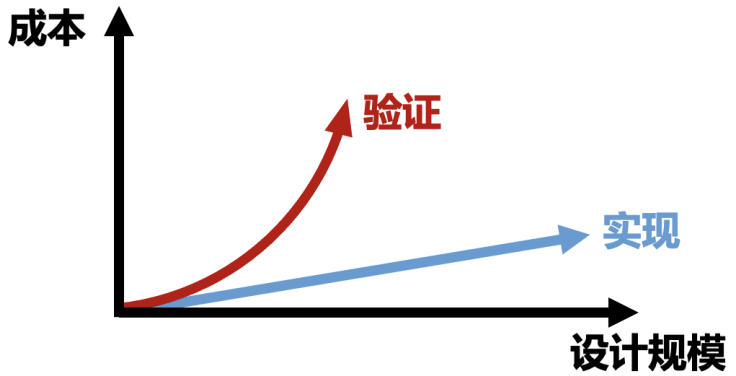
- 以对象作为基本设计单元
- 可模块化且接口标准，易复用

高性能处理器的验证方法

- **验证是处理器开发流程中耗时最长的一个阶段**
  - 香山第一版9月启动Linux，次年4月基本完成验证工作，7月流片

- **随着设计效率的提高，验证将成为处理器开发的瓶颈**

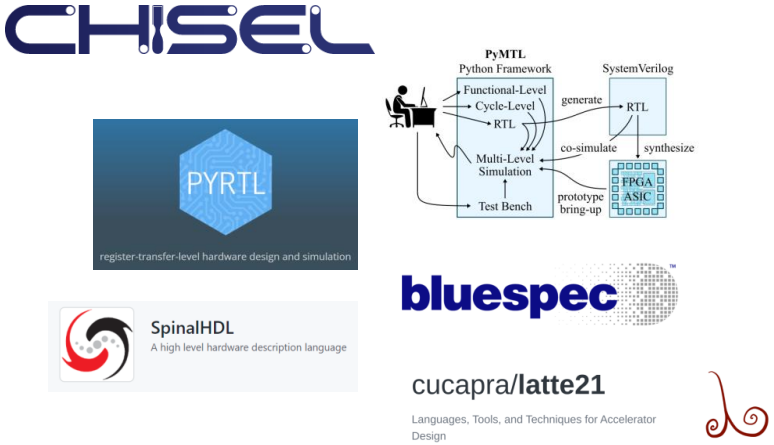
Amdahl's Law：$S_{latency}(s) = \frac{1}{(1-p)+\frac{p}{s}} < \frac{1}{1-p}$

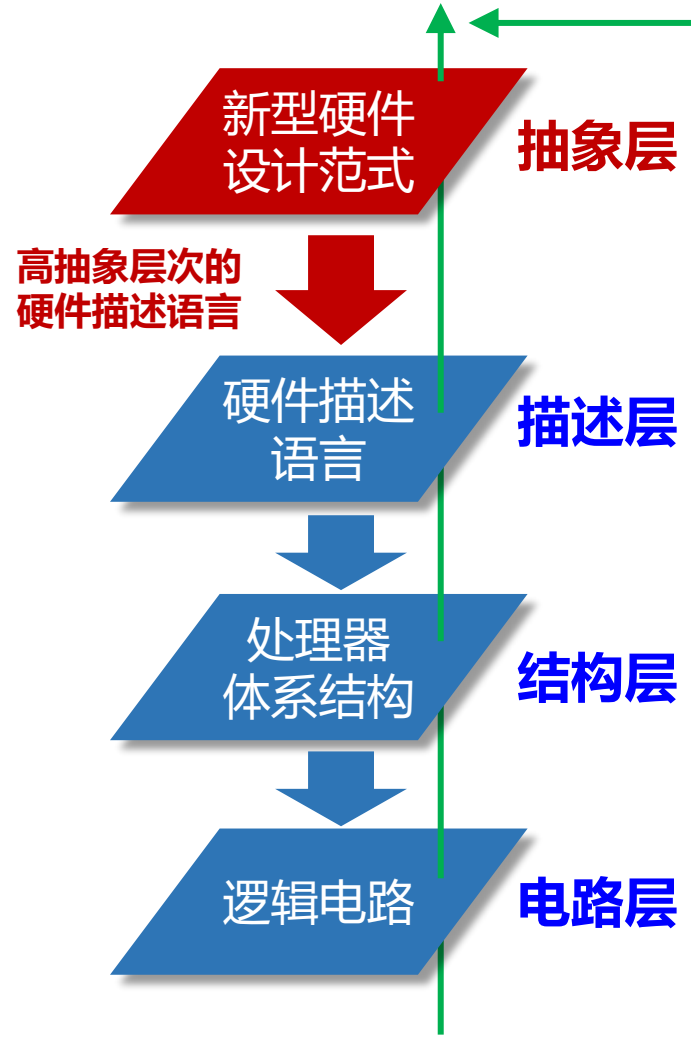


- **敏捷验证：使用更低的（人力、金钱、时间等）成本完成验证工作**

敏捷设计范式背景下的敏捷验证

敏捷工具支持下的
敏捷开发范式





敏捷硬件设计语言

敏捷硬件设计方法

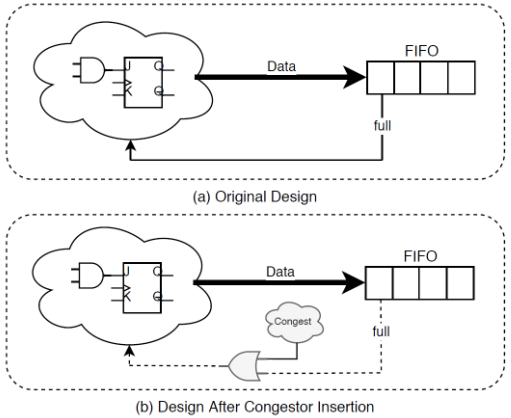
敏捷硬件验证方法

处理器敏捷验证方法

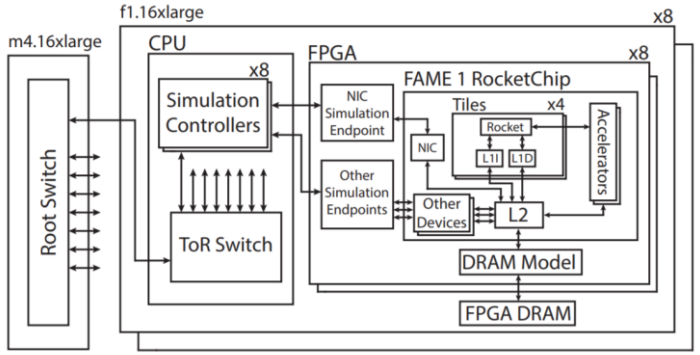
- 处理器的验证至少包括以下两个方面：
  - 功能验证：处理器的功能是否满足ISA定义的规范要求
  - 性能验证：处理器的性能是否达到了预期的性能指标

目前香山前端团队主要关注的内容

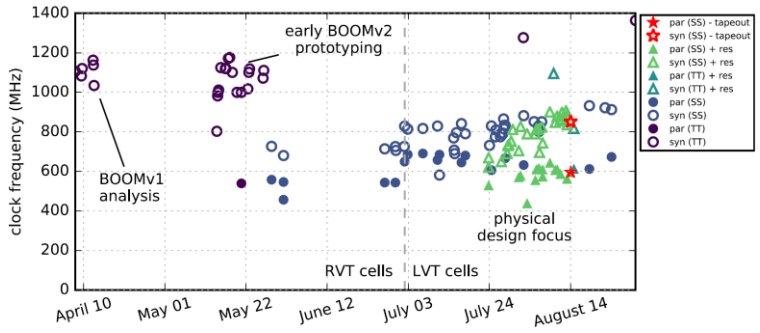
- 已有一些工作关注处理器敏捷验证方法



Dromajo[MICRO'21]：通过Fuzzy Logic设置随机事件以提高验证覆盖率

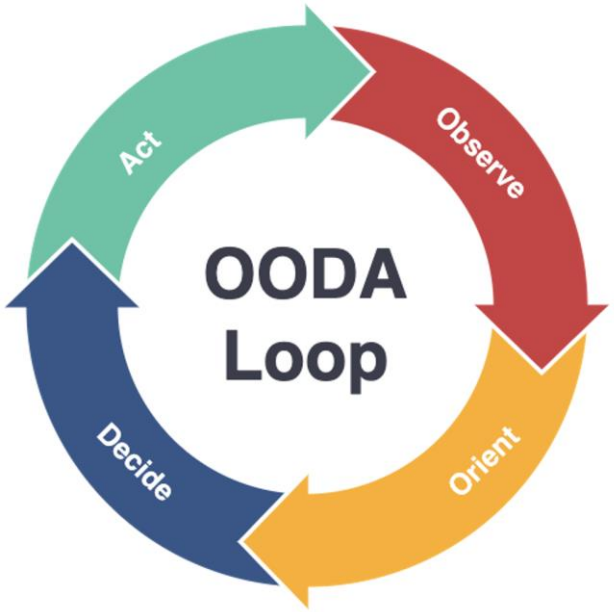


FireSim[ISCA'18]：基于公有FPGA云实现互联总线级大规模设计验证



BROOM[HotChips'18]：通过参数化缩小结构参数完成关键路径评估

敏捷验证：快速完成验证迭代的循环



观察 Observe

判断目前的设计是否存在问题

定位 Orient

确定目前的设计存在什么样的问题

决策 Decide

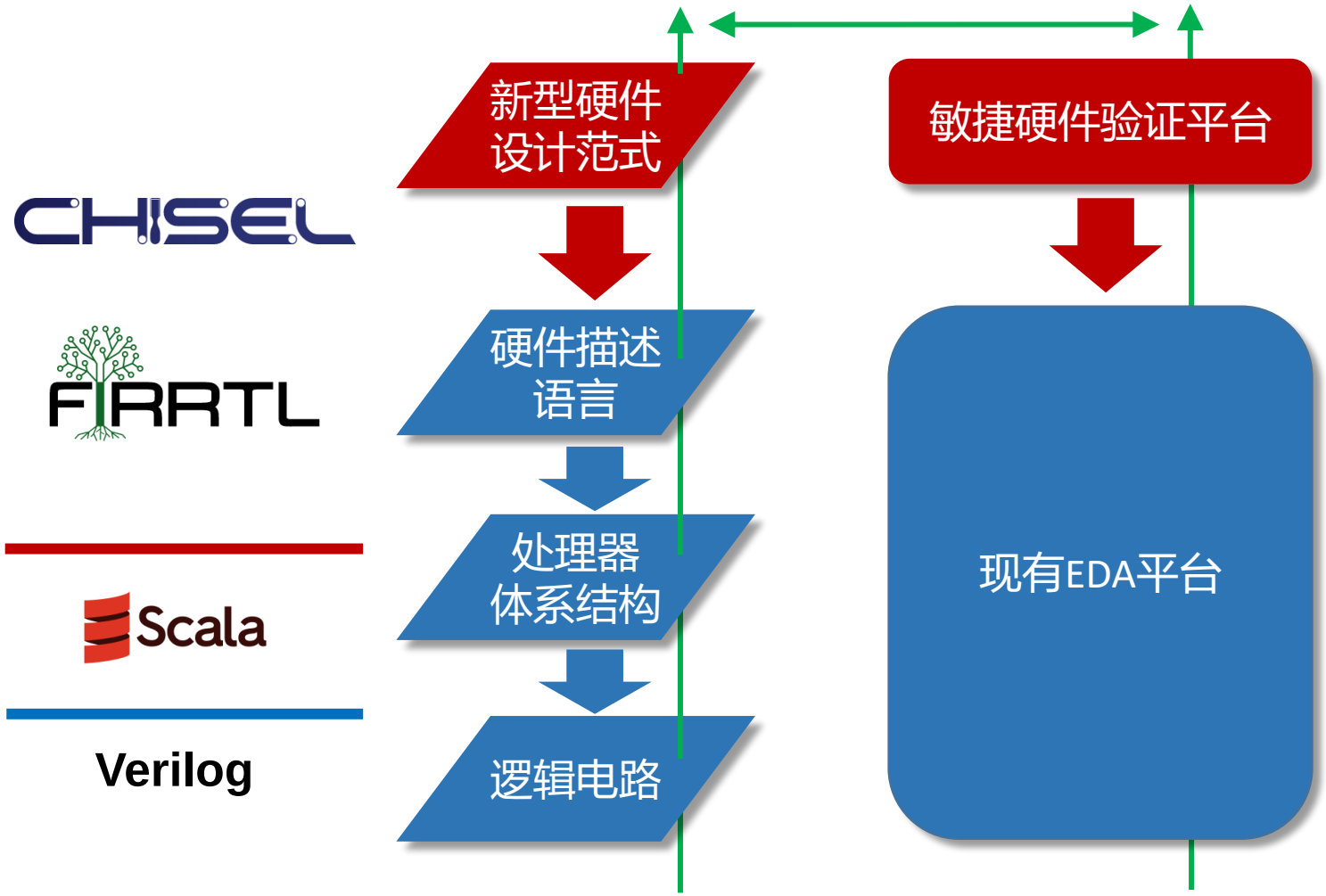
决定如何解决目前的设计存在的问题

行动 Act

应用已知的解决方案至目前的设计中

图：https://online.visual-paradigm.com/cn/knowledge/decision-analysis/what-is-ooda-loop/

建立匹配敏捷设计工具的敏捷验证平台



敏捷硬件设计语言 **敏捷硬件设计方法** **敏捷硬件验证方法** **敏捷硬件验证工具**

①敏捷功能验证方法

difftest

VERILATOR

发现错误

保存出错现场

运行回归测试

定位并解决问题

Snapshot

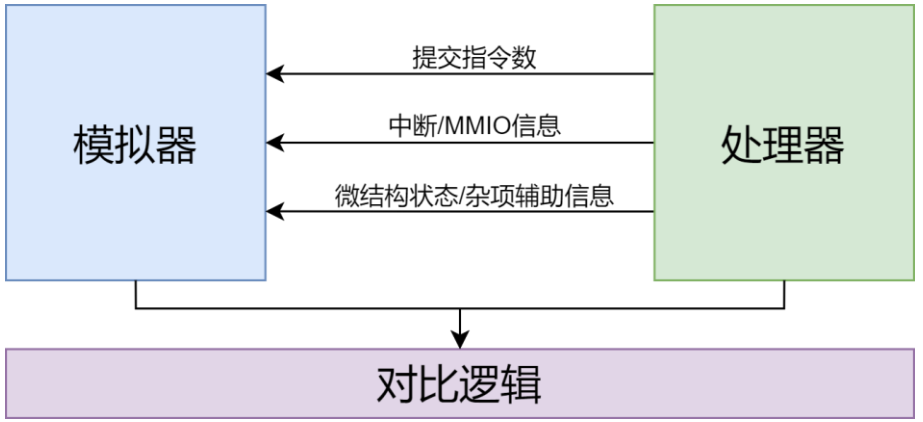
LightSSS

ArchDB

WT

Difftest：指令级在线差分验证框架

- **基本流程**
  - 处理器仿真产生指令提交/其他状态更新
  - 模拟器执行相同的指令
  - 比较两者状态
  - 单步结果：报错或继续



基本验证框架

- **提供Chisel和Verilog API**
  - **支持Verilator/VCS等RTL仿真器、NEMU模拟器**

- **SMP-Difftest：支持 SMP 结构的全系统仿真**
  - 支持**多线程**程序、**SMP Linux 内核**等负载
  - 支持检测**Cache一致性、内存一致性**方面的软硬件问题

```
while (1) {
    icnt = cpu_step();
    nemu_step(icnt);
    r1s = cpu _getregs();
    r2s = nemu_getregs();
    if (r1s != r2s) { abort(); }
}
```

在线对比机制

如何更快地获得出错现场信息

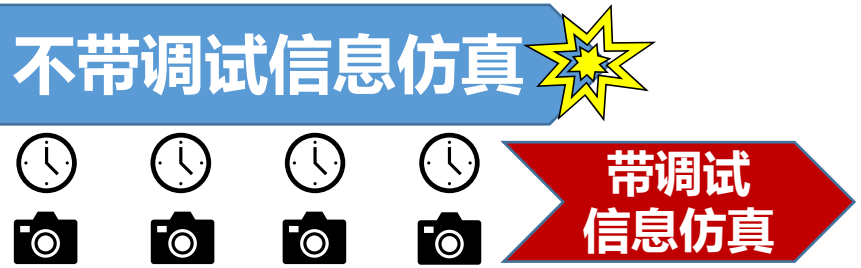
方法1 带调试信息仿真

问题①：无法预知是否会出错——产生大量无用的调试信息

方法2 不带调试信息仿真 不带调试信息仿真 带调试信息仿真

问题②：无法预知什么时候出错——出错位置越晚，复现成本越高

软件容错技术：快照+恢复

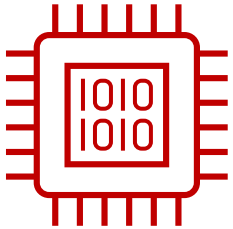
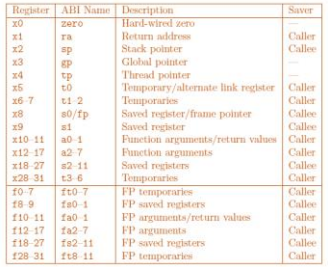
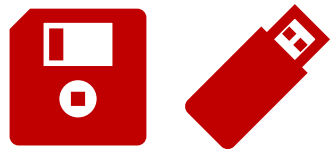


新方法 **定期对仿真进行快照，并在出错时恢复相应快照进行调试**

实现方案：应用级快照；系统级快照

快照方案1：应用级快照

- **SSS**：由程序代码直接**指定快照操作需要保存的变量和数据**

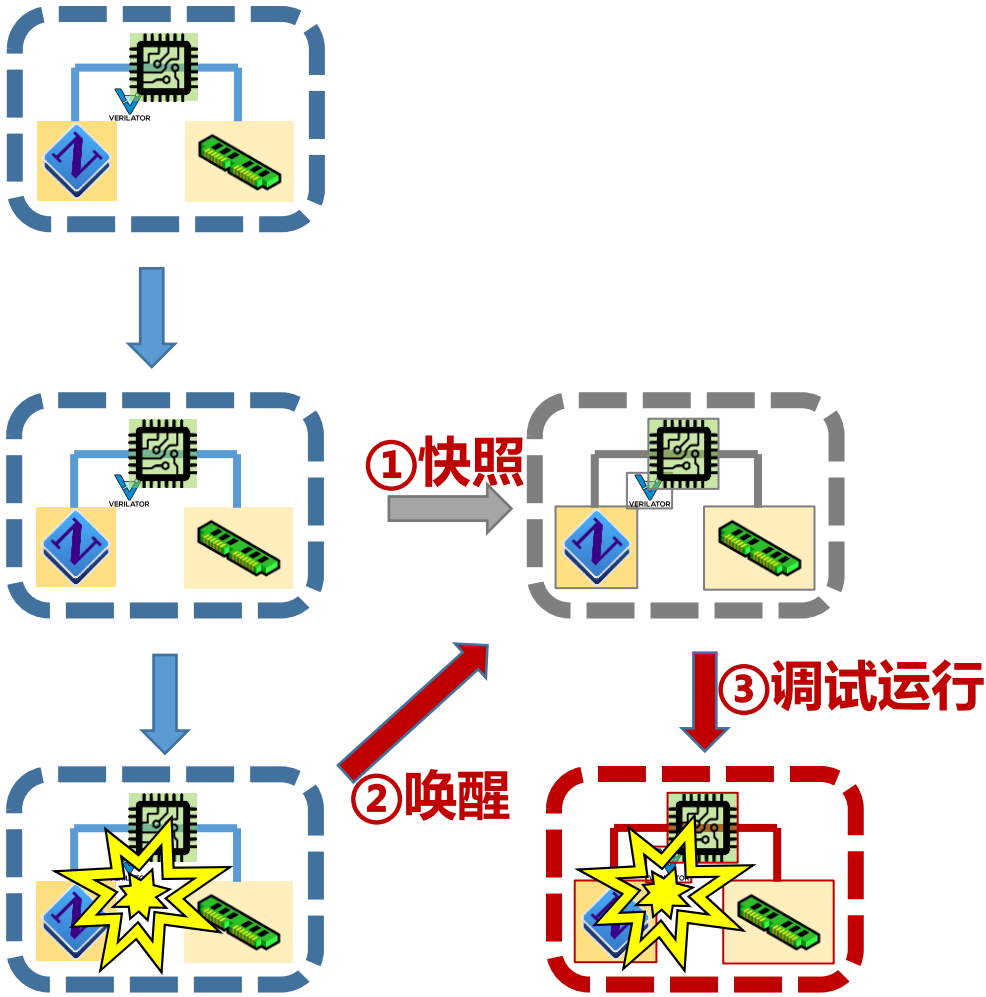
	RTL仿真电路状态	NEMU状态	非RTL外设状态
数据内容			
	所有电路信号值	**体系结构寄存器、内存等**	**状态、数据等**
实现方式		**逐个保存**	**逐个保存**

快照方案2：系统级快照

- **LightSSS: Light-weight Simulation SnapShot**
  - 一种轻量级仿真快照方法

- **关键技术：用fork对进程状态做快照**
  - **由OS提供写时复制技术（Copy On Write, COW）**

- **优势1：可扩展性好，**支持对任意外部模型的快照
  - 无需理解外部模型的内部状态
- **优势2：快照效率高，LightSSS v.s. SSS**
  - Verilator编译：**3.8X**
  - g++编译：**2.8X**
  - 单次快照：**6951.4X**



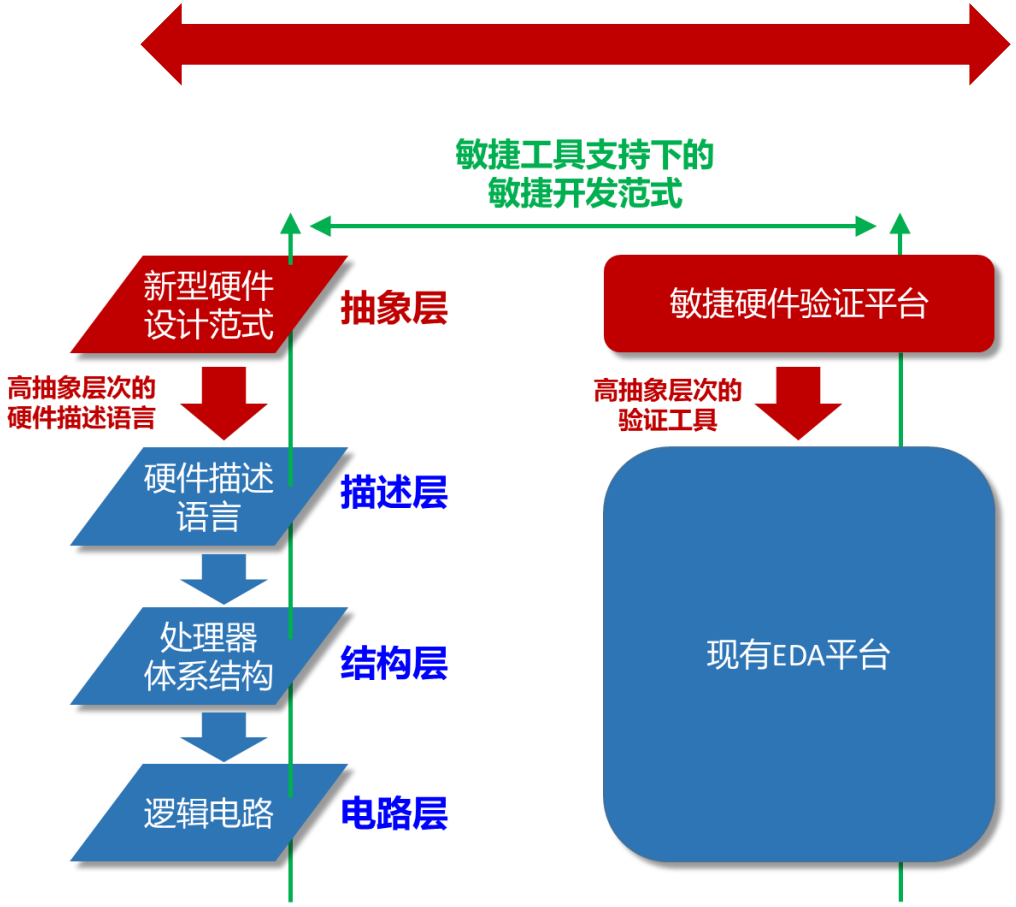
构建敏捷设计与验证工具的闭环

CHISEL

FIRRTL

Scala

Verilog



支持硬件敏捷设计范式的敏捷调试手段：Verilog与Chisel之间的桥梁

错误现场

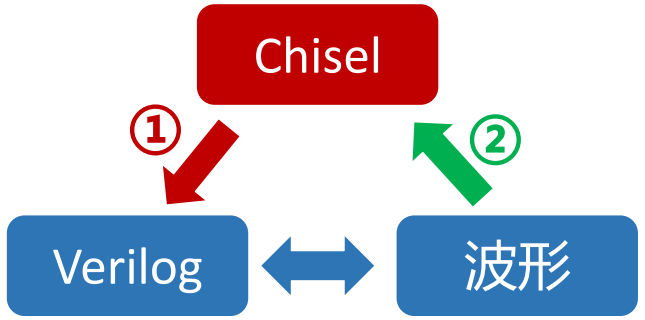
错误检查

VERILATOR

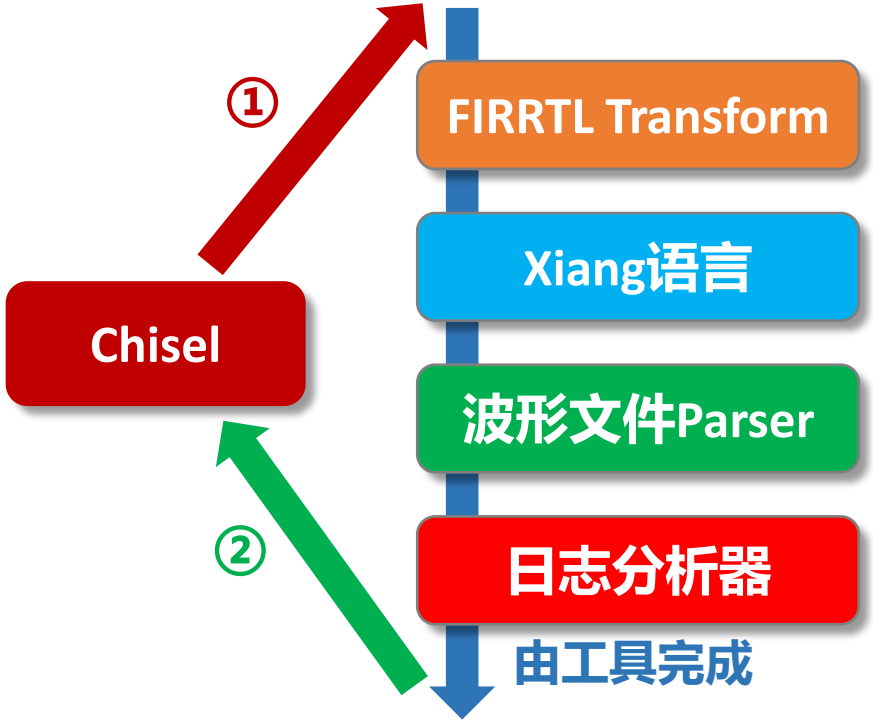
仿真波形文件

Waveform Terminator：敏捷硬件调试栈

- 核心目标：从波形中恢复被丢失的抽象语义信息



- 案例：基于高层次事件的调试方案



将Chisel中开发者指定的事件自动转换成 ”Xiang” 语言描述

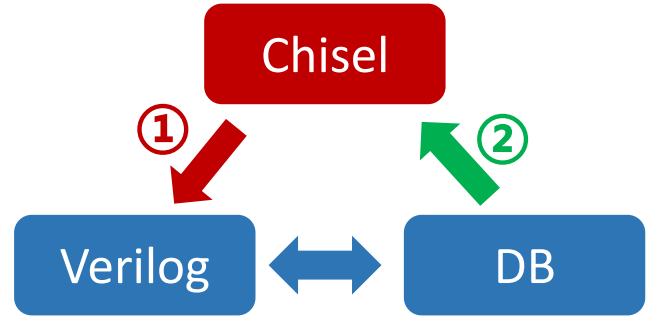
自定义 “Xiang” 语言，用于描述从波形信息到事件日志信息的转换规则

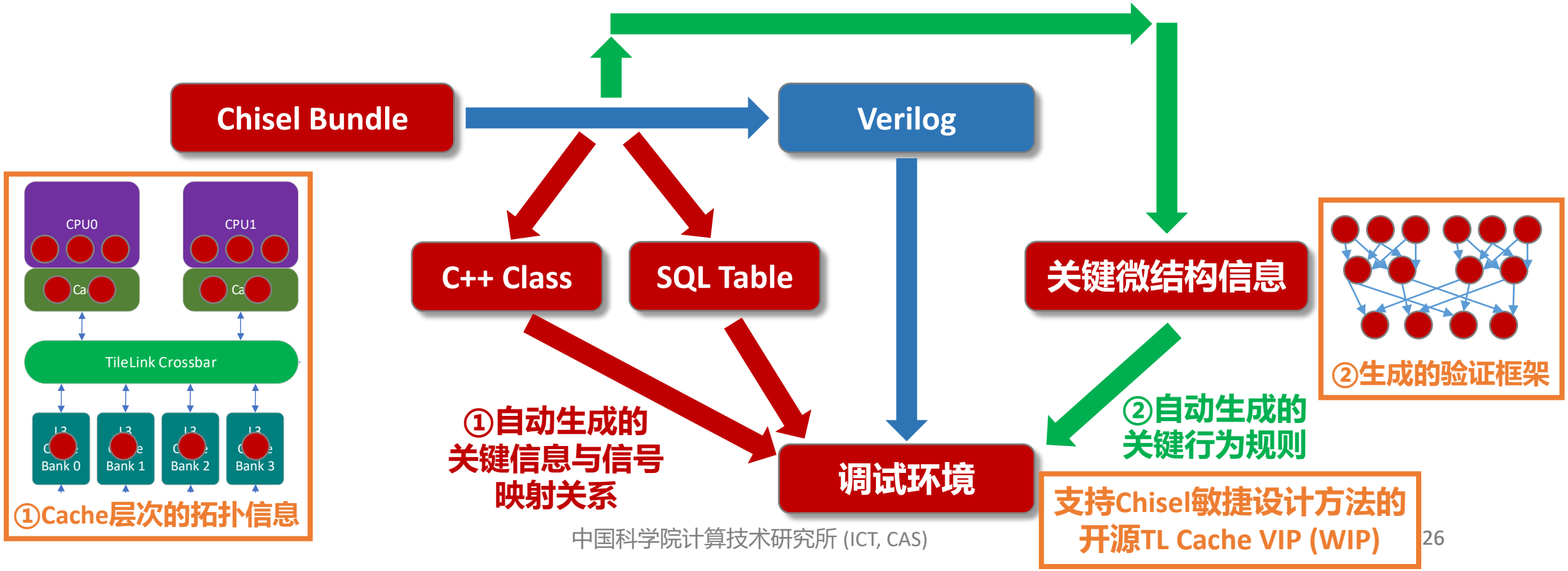
解析波形文件，获取其中的关键信息

对得到的关键信息日志进行高层次的语义分析和检查

ArchDB：敏捷硬件调试工具

- 关键：使用软件工程的方法，提高①②的自动化程度
- 案例：针对关键微结构信息的调试方案（WIP）





②敏捷性能验证方法

Chisel-based prototypes

功能点实现

运行性能测试

VERILATOR

umd-memsys/
DRAMsim3
DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator

XSPerf

Top-down analysis

性能分析

性能评估

RISC-V Checkpoint

Functional Warmup

SimPoint

基于Chisel的设计空间探索方法

- **解决了准确度的问题**：基于RTL代码的性能验证，与真实硬件表现完全一致
- **解决了重复工作的问题**：不再需要完成模拟器与硬件代码的对齐工作

- **带来了另一个问题：如何保证性能验证的效率?**
  - 敏捷设计方法保证了实现的效率，但仍然有大量的验证工作需要完成

	Verilog/SV	Cycle-approximate Simulator	Cycle-accurate Simulator	Chisel
准确程度	高	较高	低	**高**
实现速度	慢	较快	快	**快**
评估速度	慢	较快	快	**快? 慢?**
分析难度	高	较低	低	**高? 低?**
对齐成本	低	较高	高	**低**

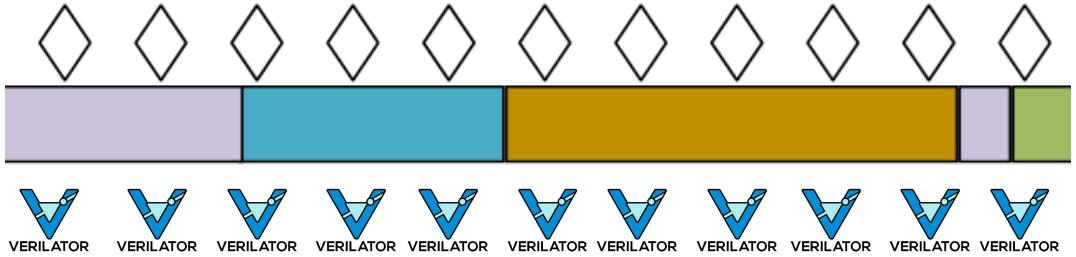
基于仿真的敏捷性能验证方法

- **敏捷验证**：使用**更低的（人力、金钱、时间等）成本**完成验证工作

- 然而，**传统的性能验证方式在高性能处理器场景下成本极高**

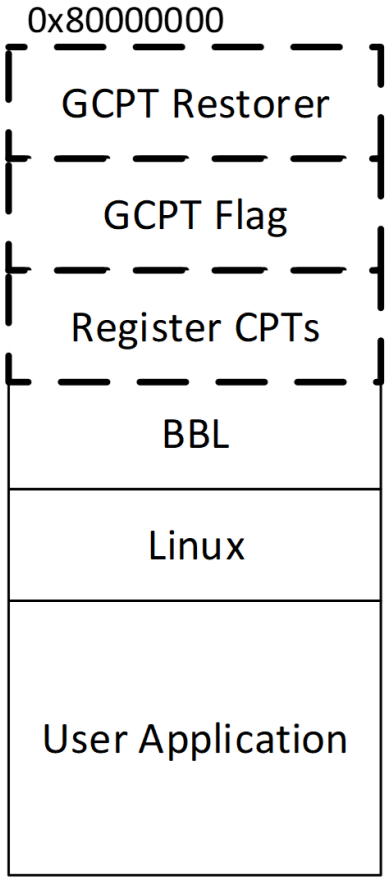
通过RISC-V Checkpoint提高仿真并行度

- **出发点：将程序切成小片段，片段间可以实现并行仿真**



- **难点①：对程序在某一时刻的状态进行快照**
  - 基于指令集模拟器完成，与Snapshot-NEMU类似

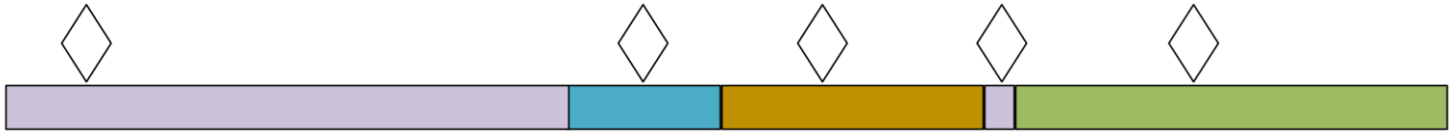
- **难点②：在香山仿真时完成快照的恢复**
  - 关键问题：RTL并不确定，需要一种普适的方法完成状态恢复
  - **原理：通过特权指令实现各寄存器的初始化**
  - 已有ARM等指令集上的类似工作，需要针对RISC-V指令集进行设计



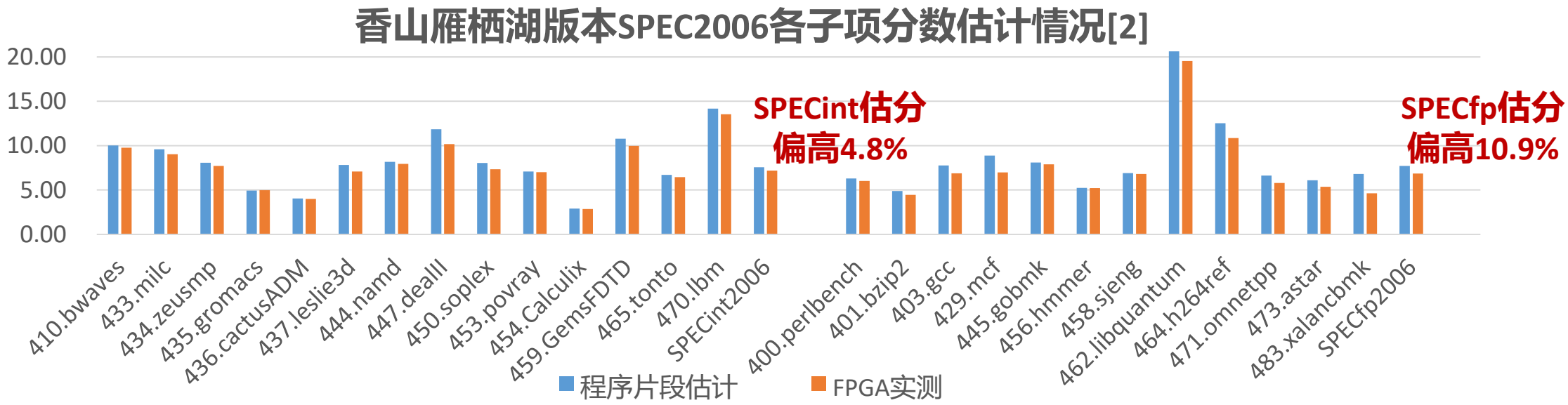
RISC-V Checkpoint 基本格式

特征采样：进一步选出具有代表性的程序片段

- **Simpoint[1]**：针对程序基本块的聚类，**从程序中选出具有代表性的片段**



- **案例：结合片段IPC与聚类权重完成程序总体执行时间的估计**



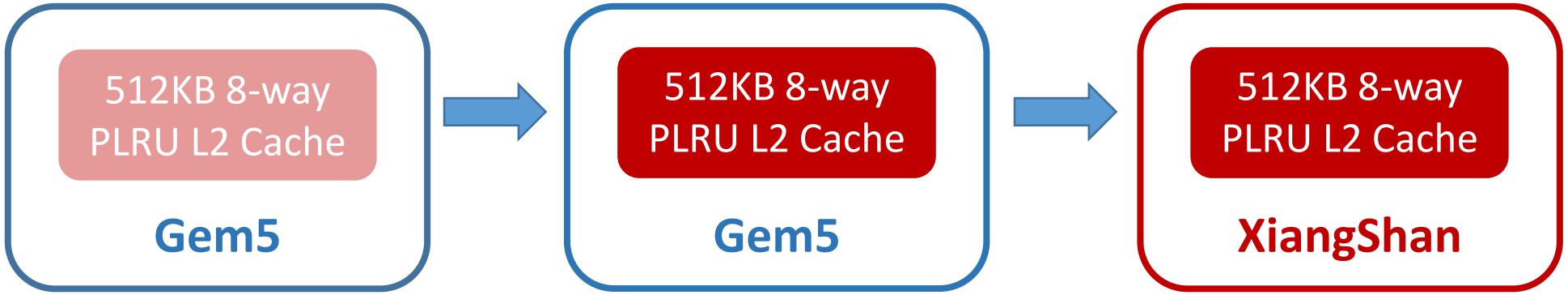
[1] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder, SimPoint 3.0: Faster and More Flexible Program Analysis , Journal of Instruction Level Parallel, September 2005.

[2] 仿真和FPGA平台均使用90周期固定访存延迟，Gamess与wrf因当时存在的编译问题未包含在内。Simpoint设置Coverage 80%, maxK=30

功能预热：进一步提高仿真效率

- 程序切片仿真存在需要**预热问题：分支预测、Cache等结构初始值是无效的**
- **效率低**：需要跑足够长的一个预热片段后，才能进入性能评估片段
  - 往往预热片段长度与性能评估片段长度在一个数量级
  - 导致**实际可达到的性能评估仿真速度只有仿真器理论速度的一半**

- **解决方案：基于模拟器完成结构的功能预热（WIP）**



①对齐目标结构的关键参数

②使用模拟器执行一定长度的预热片段

③将模拟器运行结果导入仿真RTL作为初始值

支持敏捷设计语言的敏捷性能分析框架

- RTL仿真带来的优势：方便地获取**覆盖广泛的细节性能数据**

- **XSPerf：基于Chisel/Scala的性能计数器配置框架**
  - 支持参数化，匹配硬件敏捷设计范式需求
  - 自动化完成性能计数器的实例化、运行时开关等，提高代码复用程度
- **Top-down analysis：细粒度的性能分析工具**
  - 可视化展示处理器的性能瓶颈位置，指导设计空间探索

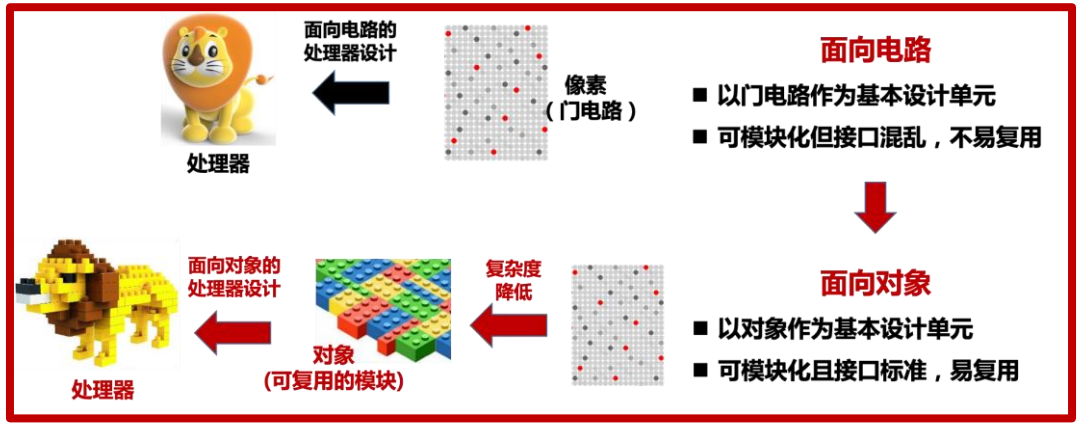
我们的目标：在24小时内完成一个功能特性的性能验证

敏捷验证：进一步工作

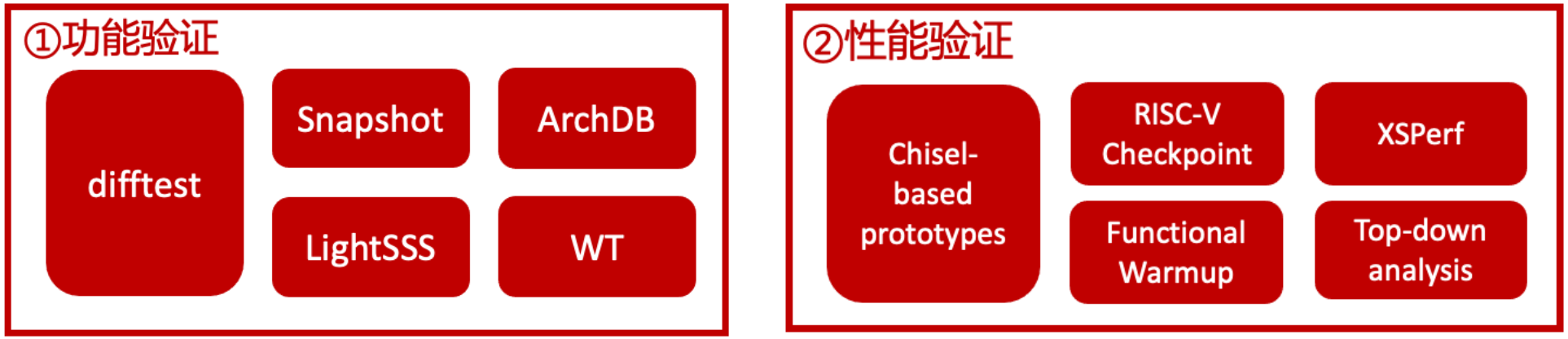
- **问题①**：是否有可能通过EDA工具的优化进一步提高测试的运行速度？
  - 是否有可能采用测试之外的手段完成验证，如形式化方法？
- **问题②**：单次验证循环的效率得到了提升，如何进一步减少验证的循环次数？
  - 是否可以使用更少的测试集，达到相同的测试覆盖率？或达到相同的性能评估准确度？
- **问题③**：在敏捷设计的背景下，如何进一步优化敏捷物理验证流程？
  - 是否有可能进行敏捷频率、功耗、面积评估？

香山：面向高性能处理器的敏捷开发实践

- **敏捷设计**：以Chisel为例实现对模块的功能抽象，探索面向对象的处理器设计



- **敏捷验证**：针对敏捷设计范式搭建验证工具与平台，探索处理器敏捷验证方法



欢迎加入香山开源社区

- GitHub/Gitee/trustie/ihub 等开源代码托管平台
- 邮件列表：xiangshan-all@ict.ac.cn
- Gitter: OpenXiangShan/community
- 微信技术讨论群
- 微信公众号/知乎/微博技术研讨平台：**香山开源处理器**
  - ①技术类文章/点评：Hot Chips 等会议解读、香山技术分析
  - ②香山项目进展：每双周香山开发进展同步
  - ③观点文章

谢谢！
请各位批评指教！