北京开源芯片研究院
BEIJING INSTITUTE OF OPEN SOURCE CHIP

XSV

面向复杂 RISC-V SoC 的统一验证平台实践

Practices of a Unified Verification Platform for Complex RISC-V SoCs

马久跃 / majiuyue@bosc.ac.cn

BOSC · 香山验证团队 · RVSC2026

RISC-V SoC 验证的碎片化困局与统一环境需求

验证环境碎片化：UT/IT/ST 各建各的环境，仿真/EMU/原型各用各的流程，SoC 配置一变验证环境即重建。

现状 · 碎片化

层级割裂 · 激励不垂直复用

UT 环境 | IT 环境 | ST 环境

工具割裂 · 流程与调试接口各异

软件仿真 | EMU 原型 | FPGA 原型

配置割裂 · 信号路径随配置重建

SoC 配置变化 → 验证环境重建

复用率低 · 集成成本高 · Sign-Off 标准难统一

目标 · 统一开源 SoC 集成与验证环境



01 生态扩张与验证瓶颈

RISC-V开放模块化催生碎片化，指令组合爆炸使验证成本失控，成为其向高性能高可靠场景跃迁的核心壁垒。

02 层级 · 工具 · 配置三重割裂

UT/IT/ST 环境不互通、激励无法垂直复用；仿真/EMU/原型流程与调试接口各异；SoC 配置一改信号路径即重建。

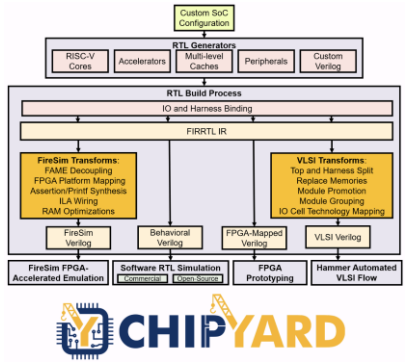
03 从项目内生转向生态复用

将RISC-V核与已验证IP标准化复用，以开源生态分工替代重复劳动，缩短周期、降低风险，依赖开源平台的汇聚与支撑。

开源社区解决方案

Chipyard

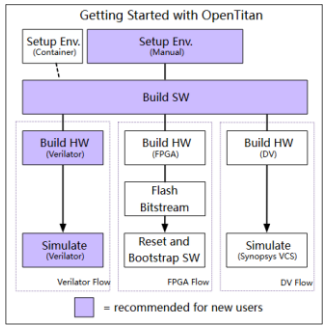
Berkeley · SoC 生成框架



- 配置驱动的 **SoC 生成框架**
- **仿真→原型→后端**全流程统一
- 多核/加速器/NoC 生成器生态

OpenTitan

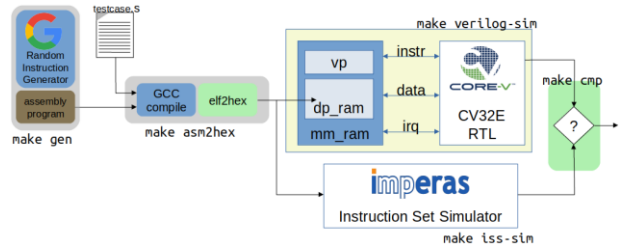
lowRISC · 安全 SoC 参考设计



- Comportable IP 规范与**可复用 IP 库**
- 工业级验证与安全设计方法论
- 完整 monorepo：**HW+SW+工具链一体化**

OpenHW

CORE-V Verif 验证方案



- UVM 验证积木库，**可组装复用**
- 多仿真器归一 + **CI 工程化闭环**
- 验证环境快速搭建

共识：SoC 集成与验证能力正从项目脚本走向生态复用 —— 框架生成 • IP 复用 • 积木组装

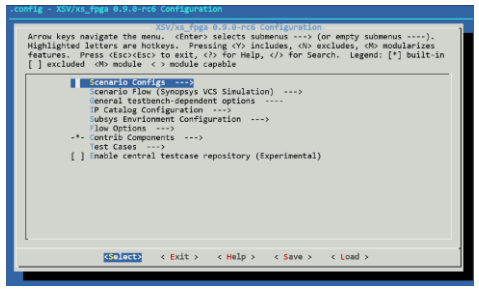
一条命令配置，一条命令构建，一条命令运行，一条命令回归

kmh_ut / ddrc / iommu / kmhv2 / ...

dual_chi / quad_pcie / chi_zj / ...

v3lt / uvs / glx / pldm / uvhs / haps / ...

```
make SC=xs_fpga quad_chi_vcs_defconfig
```



make menuconfig

```
make -j
```



Kconfig & Kbuild
from Linux Kernel

```
make run APP=xxx
```

```
make pytest tc=xxx
```

配置是唯一需要选型的环节——选对 SC 和 defconfig，后续构建、运行、回归全部由 make 一条命令驱动。

Kconfig 描述配置，Kbuild 消费配置，.config 固化配置

01

关注点分离

Kconfig 管「选什么」、Kbuild 管「做什么」、Makefile 管「怎么做」，三层各管一件事。

02

配置依赖管理

`depends on` 设上限、`select` 设下限，配置项自动联动，不写命令式脚本。

03

递归目录组织

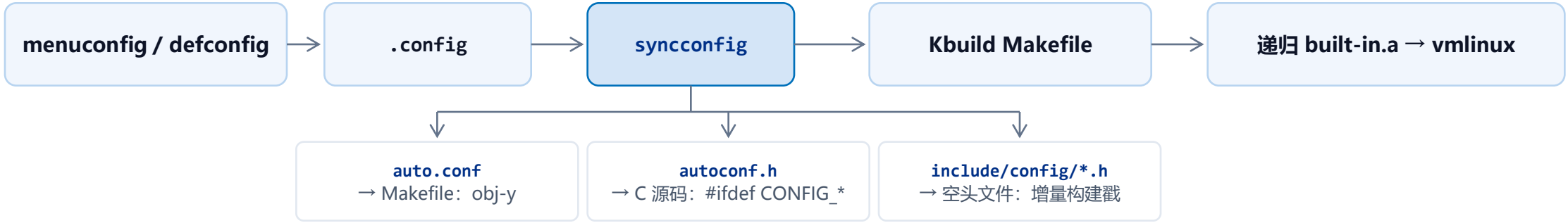
每子目录一份 Kconfig / Kbuild，`obj-$(CONFIG_XX) += dir/` 按配置递归；新子系统就近添加，顶层 Makefile 不动。

04

配置即数据

defconfig 入版本库，构建可复现、可 code review、可 git bisect 二分检索。

HOW IT WORKS · 一次构建的数据流




```
config ISDN_I4L
    bool "ISDN 4 Linux"
    depends on ISDN
    select ISDN_I4L_NET
```

```
# drivers/isdn/Kbuild
obj-$(CONFIG_ISDN_I4L) += isdn.o
obj-$(CONFIG_ISDN) += i4l/
ccflags-$(CONFIG_ACPI_DEBUG) += -D...
```

```
/* C 源码：autoconf.h 已 #define */
#ifdef CONFIG_ISDN_I4L
isdn_i4l_init(); /* y：内建 */
#endif
```

来源：kernel.org · Kconfig Language & kbuild documentation

统一配置入口，分层实现

用户侧：一个稳定入口

```
make SC=<scenario> <design>_<flow>_defconfig
```

面对的是稳定的 scenario、defconfig入口，不必为每个设计/流程重新理解一套脚本。

配置固化为 .config，向全局导出参数选择：




```
#
# Scenario Configs
#
CONFIG_USE_XS_FPGA=y
CONFIG_USE_GENERIC_SOC=y
CONFIG_USE_XSCORE_QUAD=y
CONFIG_HAVE_4_CORE=y
```

```
rtl-$(CONFIG_USE_MIG) += jtag_ddr_subsys_wrapper.v
rtl-$(CONFIG_USE_SNPS_UMCTL) += umctl_subsys.v umctl_subsys_wrapper.v \
                                umctl_apb_mux.v
sim-$(CONFIG_USE_AXI4MEM) += axi4mem_subsys_wrapper.v
rtl-$(CONFIG_USE_MMP_MEMSS) += mmp_subsys_wrapper.v
rtl-$(CONFIG_USE_HAPS_DDR4) += haps_ddr_subsys_wrapper.sv
```

平台内部：

配置层 · Kconfig / defconfig

固化 Flow、平台、设计选择

Tool flow 层 · 工具后端

实现 VCS / PLDM / Vivado / HAPS / UVHS / UVS 等编译运行流程

公共库层 · 可复用能力

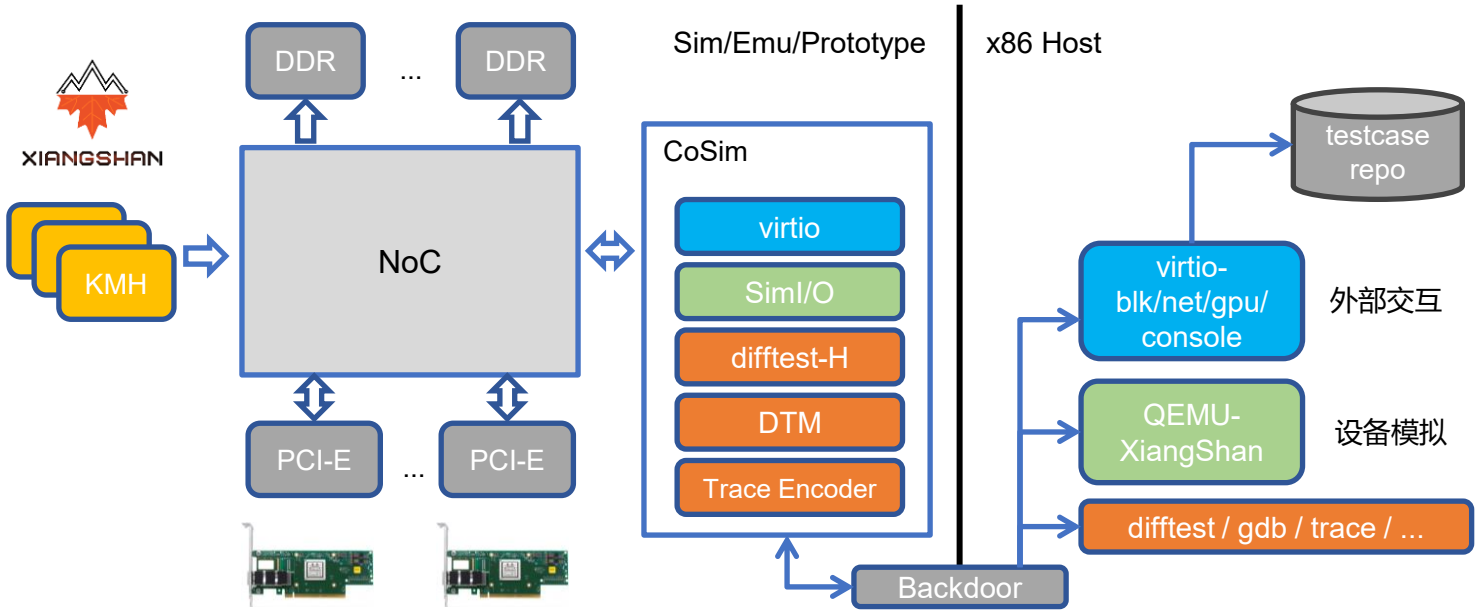
ipcatalog、subsystem、VIP、UVM环境、testcases、...

scenario• 定义验证对象与场景

组装DUT顶层设计（xs_fpga / kmhv2 / kmh_ut / iommu / ...）

用户侧一条 make 入口，操作面稳定，配置决策落到 .config 配置全局可见。

XiangShan ST 验证场景（SC: xs_fpga）



验证平台选择

软仿 / EMU / FPGA原型

选择结果 → 决定 TOP & flow

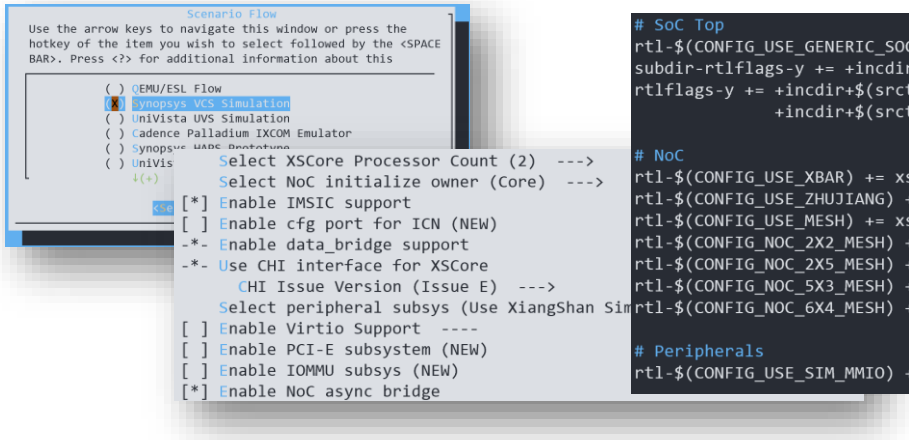
子系统选择

Core · NoC · DDR · PCIe · 外设

选择结果 → 确定SoC框架

特性选择

IMSIC, DTM, virtio, iommu, trace, ...




```
# SoC Top
rtl-$(CONFIG_USE_GENERIC_SOC) += xs_core_def.v
subdir-rtlflags-y += +incdir+$(srctree)/$(src)
rtlflags-y += +incdir+$(srctree)/lib/fpga \
              +incdir+$(srctree)/lib/xscore

# NoC
rtl-$(CONFIG_USE_XBAR) += xs_icn_direct.v
rtl-$(CONFIG_USE_ZHUJIANG) += xs_icn_zj_single.v
rtl-$(CONFIG_USE_MESH) += xs_icn_mesh.v
rtl-$(CONFIG_NOC_2X2_MESH) += mesh_2x2.v
rtl-$(CONFIG_NOC_2X5_MESH) += mesh_2x5.v
rtl-$(CONFIG_NOC_5X3_MESH) += mesh_5x3.v
rtl-$(CONFIG_NOC_6X4_MESH) += mesh_6x4.v

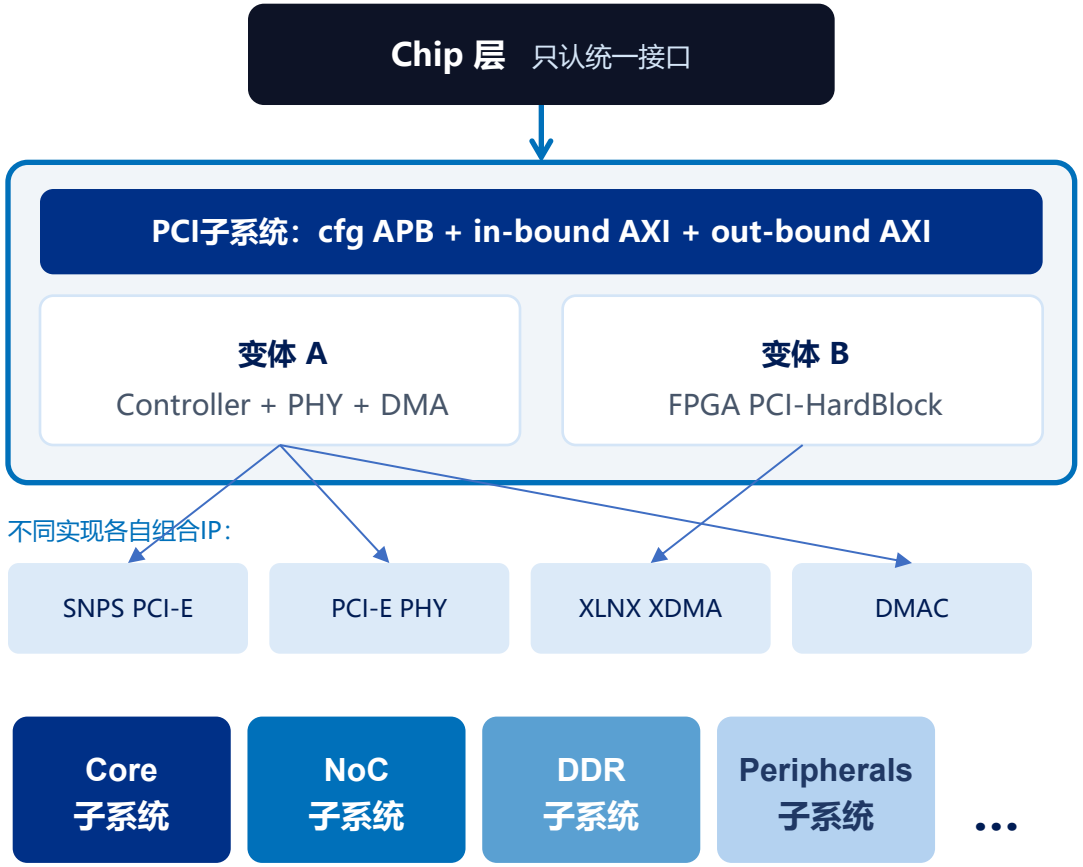
# Peripherals
rtl-$(CONFIG_USE_SIM_MMIO) += xs_sim_peri.v
```

```
xs_sys_icn u_icn(
    ...
`ifdef CONFIG_USE_XSCORE_AXI
    .core_axi_awready       (xstile_s2m_awready),
    .core_axi_awvalid       (xstile_m2s_awvalid),
    ...
`elsif CONFIG_USE_XSCORE_CHI
    .core_chi_nodeID        (xstile_chi_nodeID),
    .core_chi_tx_req_flitv  (xstile_chi_tx_req_flitv),
    .core_chi_tx_req_flit   (xstile_chi_tx_req_flit ),
    ...
`endif
`ifdef CONFIG_USE_PCIE
    .pcie_s_awaddr   (pcie2noc_awaddr ),
    .pcie_m_araddr   (noc2pcie_araddr ),
    ...
`endif /* CONFIG_USE_PCIE */
```

```
XSTop_wrapper xs_cores(
    ...
`ifdef CONFIG_USE_XSCORE_AXI
    .io_core_awready        (xstile_axi_awready),
    .io_core_awvalid        (xstile_axi_awvalid),
    ...
`elsif CONFIG_USE_XSCORE_CHI
    .io_chi_nodeID          (xstile_chi_nodeID),
    .io_chi_tx_req_flitv    (xstile_chi_tx_req_flitv),
    .io_chi_tx_req_flit     (xstile_chi_tx_req_flit),
    ...
`endif
);
```

```
`ifdef CONFIG_USE_PCIE
pcie_wrapper u_pcie_wrapper(
    .pcie_s_araddr      (noc2pcie_araddr ),
    .pcie_m_awaddr      (pcie2noc_awaddr ),
);
`endif /* CONFIG_USE_PCIE */
```

“可配置”子系统




```
config USE_PCI_SUBSYS
  bool “Enable PCI”
  default n

choice PCI_SUBSYS
  depends on USE_PCI_SUBSYS
  default PCI_XLNX

  config PCI_SUBSYS_SNPS
    bool “SNPS”
    depends on VCS
    select PCI_SNPS
    select PCI_SNPS_PHY
    select HAVE_DMAC
  config PCI_SUBSYS_XLNX
    bool “XDMA”
    depends on XLNX_FPGA
    select XLNX_XDMA
endchoice

config USE_PCI_DMA
  bool “Enable DMA in PCI”
  depends on HAVE_DMAC
  default n
```

```
chip.sv
------------------------------
module chip(…);
  …
`ifdef CONFIG_USE_PCI_SUBSYS
  pci_subsys u_pci (…)
`endif
endmodule

pci_subsys_snps.sv
------------------------------
module pci_subsys(…);
  snps_pcie_top U_PCI_INST(…);
  snps_pcie_phy U_PHY_INST(…);
`ifdef CONFIG_USE_PCIE_DMA
  DW_dmac U_DMAC_INST ( ..)
`endif
Endmodule

pci_subsys_xlnx.sv
------------------------------
module pci_subsys(…);
  xlnx_xdma U_XDMA_INST(…);
endmodule
```

子系统 把「接口契约」与「IP 实现」解耦：必选接口恒在，可选能力通过 choice / depends on / select 联动；
对 TOP 提供统一界面，换 IP 组合不波及上层。

IP 复用：分类登记、依赖显式、按参数生成

定义层 • 结构化进 Kconfig

IP 分类

Core / NoC / PCI / DDR / PHY / VIP / iommu / intc / mcu / …

平台 & IP 依赖

depends on 前置条件，select 关联依赖，不满足不出现

VIP depends on EDA flow / Xilinx IP 依赖 Vivado / CDNS-IP依赖PLDM / …

IP个性化配置 公共打底，个性化配置

统一使用Kbuild引入IP，根据IP需求提供生成参数、配置选项

实现层 • Makefile定义流程，Kbuild引用IP例化

```
$(patsubst %,$(obj)/%.fir,$(xscore-y)): \
$(obj)/%.fir: $(GENERATOR) $(CONFIG_XSCORE_CFGYAML_FILE) FORCE
        $(call if_changed,xs_gen_core, \
                          $(GENERATOR_FLAGS) \
                          $(if $(CONFIG_XSCORE_CFGYAML), \
                               --yaml-config $(CONFIG_XSCORE_CFGYAML_FILE)) \
                          $(if $(CONFIG_XSCORE_CUSTOMIZE),$(if $(CONFIG_XSCORE_CFGYAML),, \
                               --l2-cache-size $(CONFIG_XSCORE_L2CACHE_SIZE) \
                               --l3-cache-size $(CONFIG_XSCORE_L3CACHE_SIZE))))
$(foreach m,$(patsubst %,$(obj)/%.fir,$(xscore-y)),$(eval $(m): MAINCLASS=))
```

```
#
# Generate XS Core
#
xscore-$(CONFIG_XSCORE_AP) += xscore_ap
PREFIX_xscore_ap := $(if $(CONFIG_XSCORE_AP_HAVE_PREFIX),$(CONFIG_XSCORE_AP_PREFIX))
CFG_xscore_ap := $(or $(CONFIG_XSCORE_AP_CFG),DefaultConfig)
FLAGS_xscore_ap := $(if $(CONFIG_XSCORE_CFGYAML),, \
                        $(if $(CONFIG_XSCORE_AP_CHI_ISSUE_B), --issue B) \
                        $(if $(CONFIG_XSCORE_AP_CHI_ISSUE_C), --issue C --chi-addr-width 44) \
                        $(if $(CONFIG_XSCORE_AP_CHI_ISSUE_E), --issue E.b) \
                        $(if $(CONFIG_HAVE_XSCORE_CUSTOMIZE), \
                            --yaml-config $(srctree)/$(src)/cfgs/PMA.yml \
                            $(if $(CONFIG_XSCORE_AP_CHI), \
                                --yaml-config $(srctree)/$(src)/cfgs/IMSIC.yml) \
                            $(if $(CONFIG_XSCORE_AP_LOWPOWER), \
                                --yaml-config $(srctree)/$(src)/cfgs/LowPower.yml)))
```

可复用

IP一次定义，跨芯片跨 flow 复用

可组合

IP依赖(depends)/组合(select)显式搭配

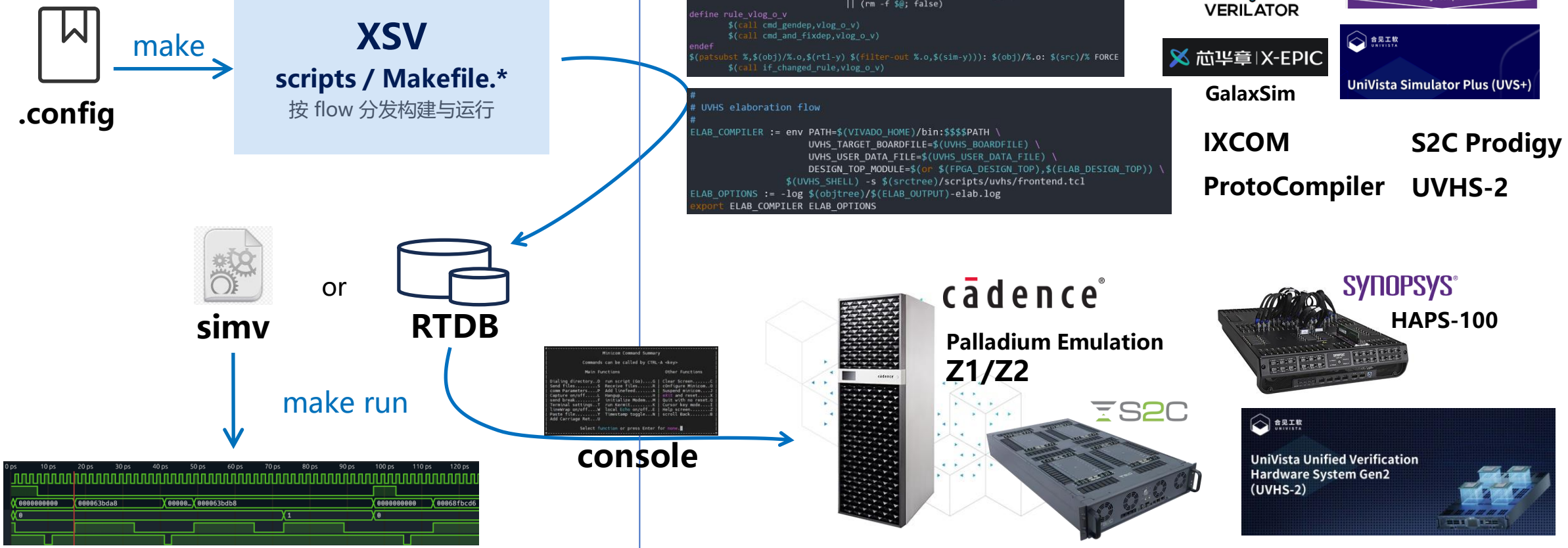
可复现

defconfig 固化全部选择，一键复现

可扩展

新增 IP 只加配置项，不动骨架

统一构建入口


.config
make
XSV
scripts / Makefile.*
按 flow 分发构建与运行
simv
or
RTDB
make run
console
VERILATOR
SYNOPSYS VCS
Design
Testbench
Assertions
Single compiler
芯华章 X-EPIC
GalaxSim
UniVista Simulator Plus (UVS+)
IXCOM
S2C Prodigy
ProtoCompiler
UVHS-2
cadence
Palladium Emulation
Z1/Z2
SYNOPSYS
HAPS-100
S2C
UniVista Unified Verification
Hardware System Gen2
(UVHS-2)

测试用例

nexus-am

Github
https://github.com › OpenXiangShan › nexus-am 翻译此结果
GitHub - OpenXiangShan/nexus-am
In this section, we demonstrate how to build memory images (base_address = 0x80000000) for simulation. Here we use the DiffTest RTL-simulation framewor…

riscv-hyp-tests

Github
https://github.com › josecm › riscv-hyp-tests 翻译此结果
GitHub - josecm/riscv-hyp-tests: A bare-metal application to test ...

FOUNDATION OPENHW PROVEN PROCESSOR IP

FORCE RISC-V

[1]



SYNOPSYS®

STING

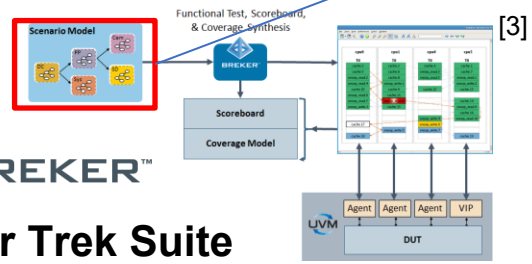
[2]

```
NAME
    mkcosim - Make a STING cosim image

SYNOPSIS
    mkcosim helper=<file.py> conf=<file.conf> [seed=<seed_no>]
    [bd=<build_dir>] [<KEY1.*>=<VALUE>]
    mkcosim -h

DESCRIPTON
    Build a STING cosim image for a configuration file passed as an
    input parameter to the script.
```

[3]



BREKER™

Breker Trek Suite

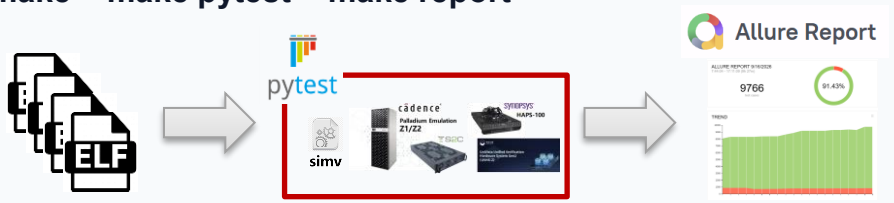
目标 SoC / Core 配置
.config · Kconfig 输出

读取 SoC/Core 配置，写入各自配置文件

FORCE cfg | STING cfg | Trek cfg | misc cfg

XSV 用例构建/回归/报告

make + make pytest + make report



[1] https://github.com/openhwfoundation/force-riscv/blob/master/doc/FORCE-RISCV_brief_instroduction_from_user_view.pdf
[2] STING man-pages
[3] https://brekersystems.com/products/trekuvm/

测试用例：随 defconfig 自动生成，本地与 CI 同入口

把用例组织成与defconfig配置对齐的可回归集合

① Kconfig 选用例

自动选择依赖的运行时：
nexus-am, hyper-test
force-riscv, STING,
TrekSVIP

→

② Kbuild 编译

编译依赖的runtime
编译用例，生成索引
根据Flow配置生成运行参数

→

③ pytest adaptor

按 flow 自动选 backend：
eda / pldm / fpga

→

④ make pytest

-k 过滤 · xdist 并行 ·
timeout · FAIL regex

关键：用例索引由 Makefile 根据当前 defconfig 生成，执行参数/编译选项随flow选择自动处理，所有平台使用pytest界面统一回归。

```
# riscv-hyp-tests based privileged tests
config TC_RV_PRIVILEGED
        bool "Enable RISC-V Privileged Testcases"
        select CONTRIB_HYPTEST
        default n
```

```
obj-$(CONFIG_TC_RV_PRIVILEGED) += tc_privileged/
```

```
# bare-metal test with hyptest library
rvprogs-always-y += rvh_smoke
rvh_smoke-deflib := hyptest
rvh_smoke-objs := rvh_smoke_wfi.o rvh_smoke_ili.o
```

```
$ make pytest tc=smoke
Found 5 tests in “local:./scenario/xs_fpga/tc.yaml
================================== test session starts ==================================
4 workers [2 items]
scheduling tests via LoadScheduling

../scripts/pytest/vcs/test_uvm.py::test_uvm[rvh_smoke (seed=1234)]
../scripts/pytest/vcs/test_uvm.py::test_uvm[friscv_smoke (seed=1234)]
[gw1] [ 50%] PASSED ../scripts/pytest/vcs/test_uvm.py::test_uvm[rvh_smoke (seed=1234)]
[gw0] [100%] PASSED ../scripts/pytest/vcs/test_uvm.py::test_uvm[friscv_smoke (seed=1234)]

------------- generated xml file: /nfs/home/majiuyue/XSV/build/junit.xml ---------------
============================== 2 passed in 162.02s (0:02:42) ==============================
```

测试用例：随 defconfig 自动生成，本地与 CI 同入口

把用例组织成与defconfig配置对齐的可回归集合

① Kconfig 选用例

自动选择依赖的运行时：
nexus-am, hyper-test
force-riscv, STING,
TrekSVIP

→

② Kbuild 编译

编译依赖的runtime
编译用例，生成索引
根据Flow配置生成运行参数

→

③ pytest adaptor

按 flow 自动选 backend：
eda / pldm / fpga

→

④ make pytest

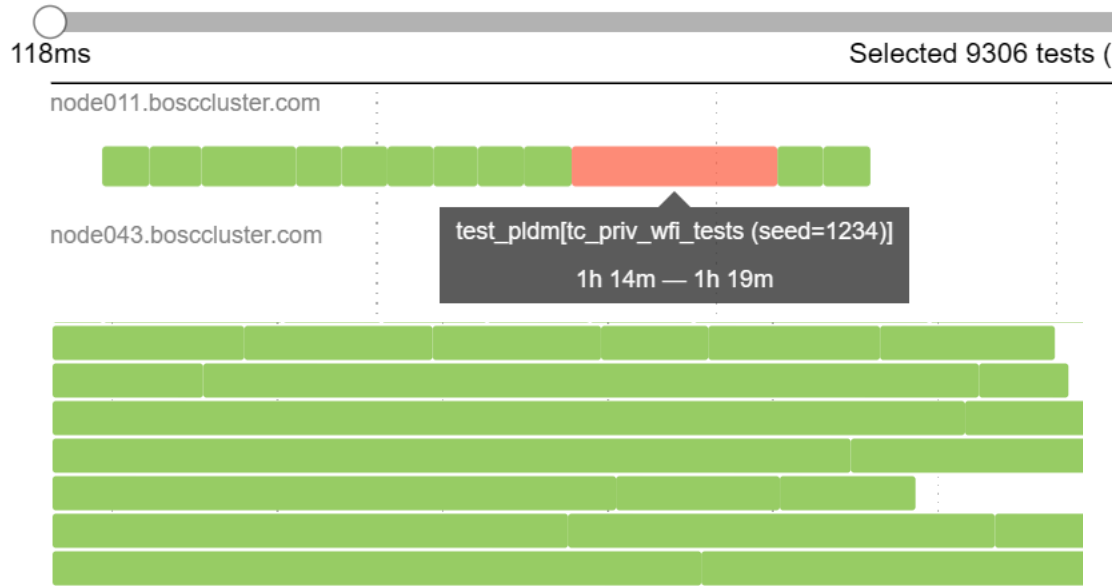
-k 过滤 · xdist 并行 ·
timeout · FAIL regex

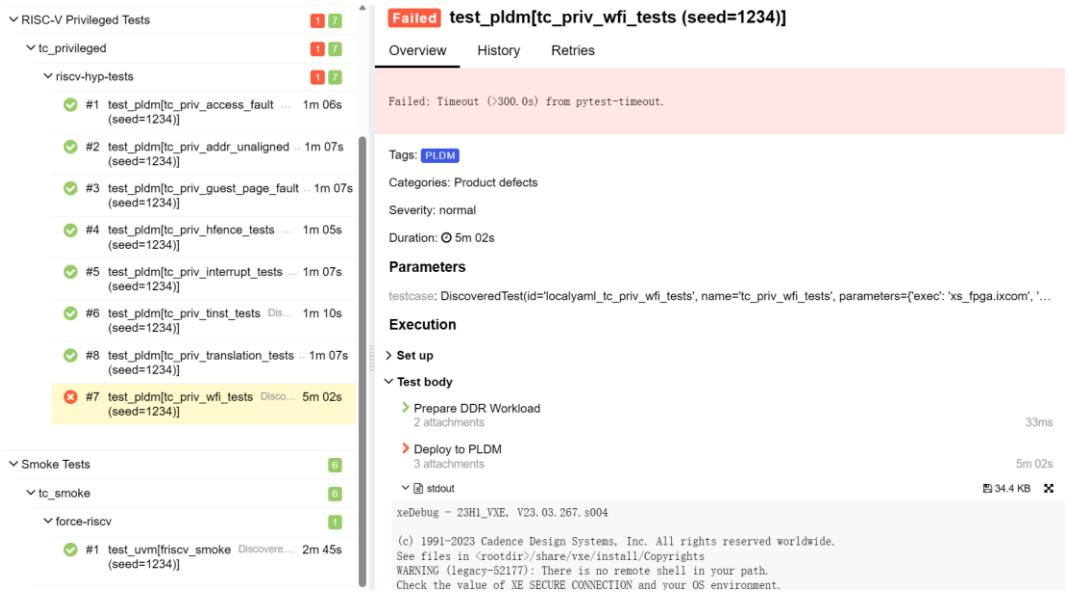
→

⑤ make report

Allure / JUnit 统一报告

关键：报告可视化，结果分类组织，信息完备。





存量 UVM 迁移入库，UCAgent/UT 接入，回归与覆盖率统一汇总

① 存量 UVM/UT 迁移入库

复用XSV中仿真与回归流程支持

原样迁入香山子模块UVM/UT环境

新增SC=KMH_UT场景

② 对接 UCAgent[1]/UT 用例

UCAgent 自动生成 UT环境与用例

按 XSV Kconfig/Kbuild 集成

UCAgent用例与人工用例同索引

Kconfig/Kbuild 底座

Kconfig 选：SC=kmh_ut

defconfig选择 UVM环境 或 UCAgent环境

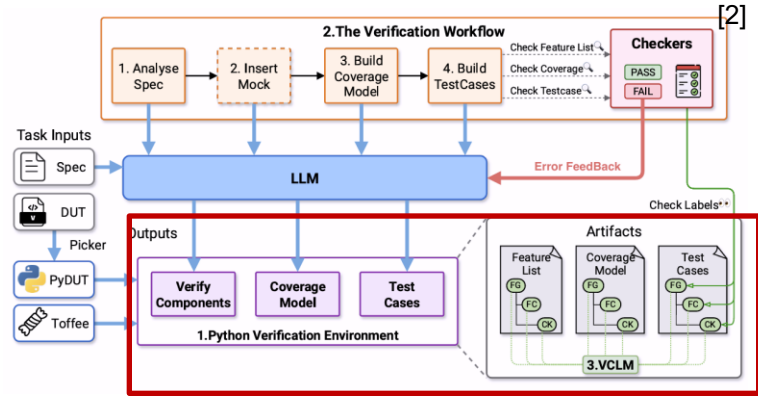
统一编译 RTL，产出测试索引、运行参数

统一回归 + 覆盖率汇总

pytest 按 flow 自动选仿真器后端

UVM用例 与 UCAgent用例 同跑一轮回归

覆盖率数据跨来源汇总，报告归一



[2]

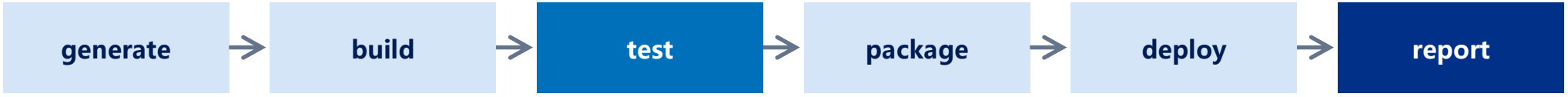
```
#
# UCAgent UT Tests
#
obj-$(CONFIG_UCAGENT_TESTS) += ucagent/
```

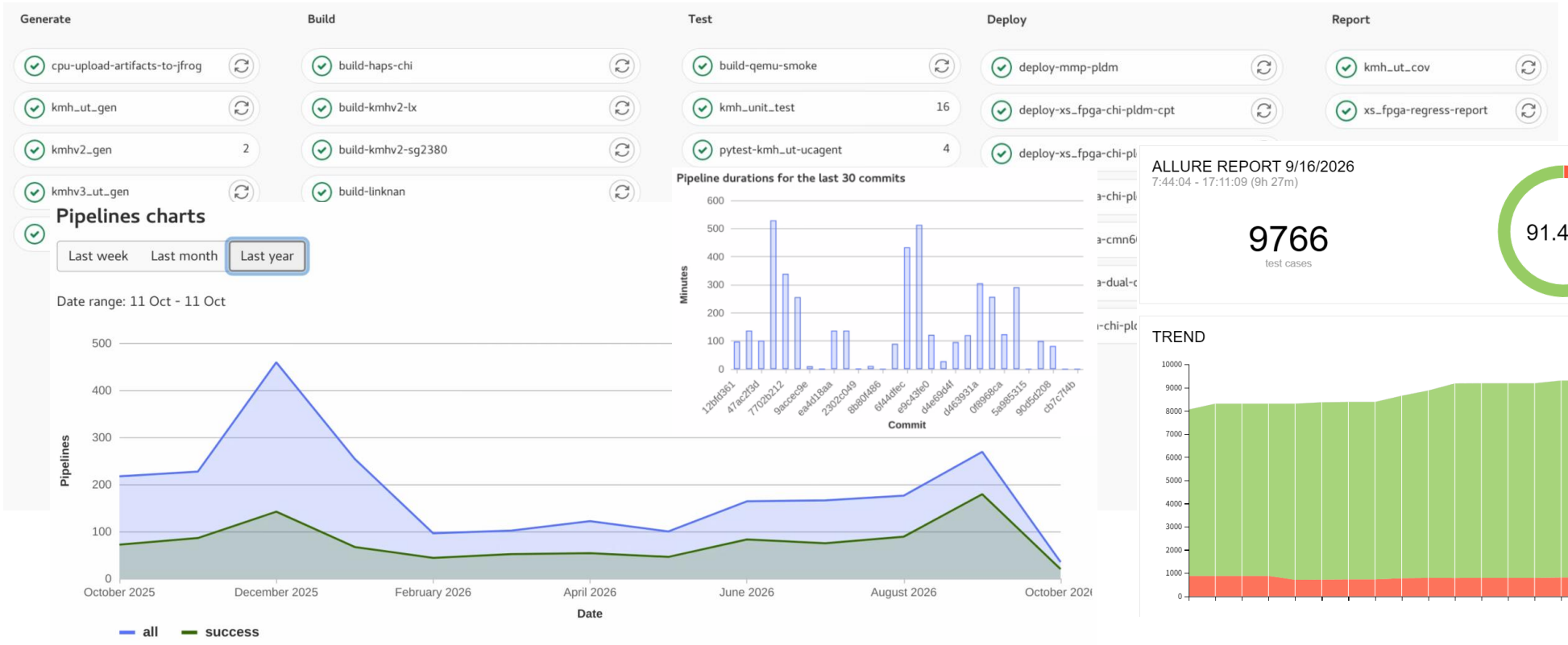
bosc_AtomicsUnit	bosc_LinkMonitor	bosc_PTW	bosc_Tage_SC
bosc_BypassNetwork	bosc_LLPTW	bosc_PtwCache	bosc_TLBNonBlock
bosc_CtrlUnit	bosc_LoadMisalignBuffer	bosc_PTWNewFilter	bosc_TLBNonBlock_1
bosc_DataPath	bosc_LoadPipe	bosc_PTWRepeaterNB	bosc_TLBNonBlock_2
bosc_DataStorage	bosc_LoadQueueRAR	bosc_RAS	bosc_TXDAT
bosc_DecodeUnit	bosc_LoadQueueRAW	bosc_Rename	bosc_TXREQ
bosc_DecodeUnitComp	bosc_LoadQueueReplay	bosc_RenameTableWrapper	bosc_TXRSP
bosc_Directory	bosc_LoadQueueUncache	bosc_RequestArb	bosc_Uncache
bosc_ExuBlock	bosc_LoadUnit	bosc_RequestBuffer	bosc_VfofBuffer
bosc_ExuBlock_1	bosc_MainPipe	bosc_Rob	bosc_VirtualLoadQueue
bosc_ExuBlock_2	bosc_MainPipe_1	bosc_RXDAT	bosc_VLMergeBufferImp
bosc_FauFTB	bosc_MissQueue	bosc_RXRSP	bosc_VLSplitImp
bosc_FTB	bosc_MMIOBridge	bosc_RXSNP	bosc_VSegmentUnit
bosc_Ftq	bosc_MSHRCtl	bosc_Sbuffer	bosc_VSMergeBufferImp
bosc_FusionDecoder	bosc_NewDispatch	bosc_Scheduler	bosc_VSSplitImp
bosc_GrantBuffer	bosc_NewIFU	bosc_Scheduler_1	bosc_WbDataPath
bosc_IBuffer	bosc_Op2ForVector	bosc_Scheduler_2	bosc_WritebackQueue
bosc_ICacheMainPipe	bosc_PMP	bosc_SinkA	conftest.py
bosc_InstrUncache	bosc_PMPChecker	bosc_SinkC	Kbuild
bosc_ITTage	bosc_PMPChecker_10	bosc_SMSPrefetcher	Kconfig
bosc_L1Prefetcher	bosc_PreDecode	bosc_StoreMisalignBuffer	
bosc_L2TlbMissQueue	bosc_Prefetcher	bosc_StoreQueue	
bosc_L2TlbPrefetch	bosc_ProbeQueue	bosc_StoreUnit	

存量 UVM用例与 UCAgent/UT用例 都登记进同一套 Kconfig/Kbuild——**配置显式、编译统一、测试索引自动生成。**

[1] UCAgent (UnityChip Verification Agent) Project. https://github.com/XS-MLVP/UCAgent
[2] UCAgent: An End-to-End Agent for Block-Level Functional Verification. https://arxiv.org/pdf/2603.25768

Gitlab-CI + pytest + Allure Report





UCAgent生成用例

四层架构：验证对象、公共能力、测试用例、工具流程各司其职

scenario

验证对象与场景策略

xs_fpga Generic SoC
Kmh-v2 pre-release
kmh_ut, iommu, ddrc, pci

common

公共能力与生成组件

ipcatalog
core/noc/ddr/io subsys
Testbench, VIP, UVM环境

tests

用例生成与归档

testcases and runtime
Kconfig dependency
testcases index
nexus-am，riscv-hyp-tests,
force-riscv, STING

scripts

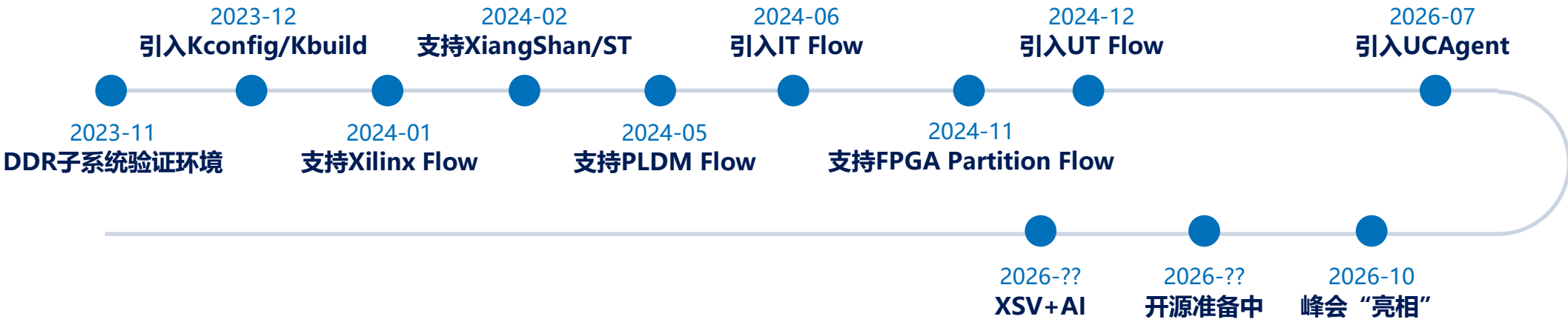
工具 flow 与运行、回归

VCS, PLDM, Vivado
HAPS, UVHS, UVS, ...
rtl-y, sim-y, filelist-y, dpi-y
pytest, Allure Report

新增 scenario：复用已有 common / tests / scripts，只补新设计差异；
新增 flow / 用例：接入 Makefile.* 或 testcase index，即可被多个 scenario 共享——各类扩展互不污染。

过去与未来

XSV不是一个实验场景，而是把SoC集成、UT/IT/ST/原型平台串成连续链路的公共基座。



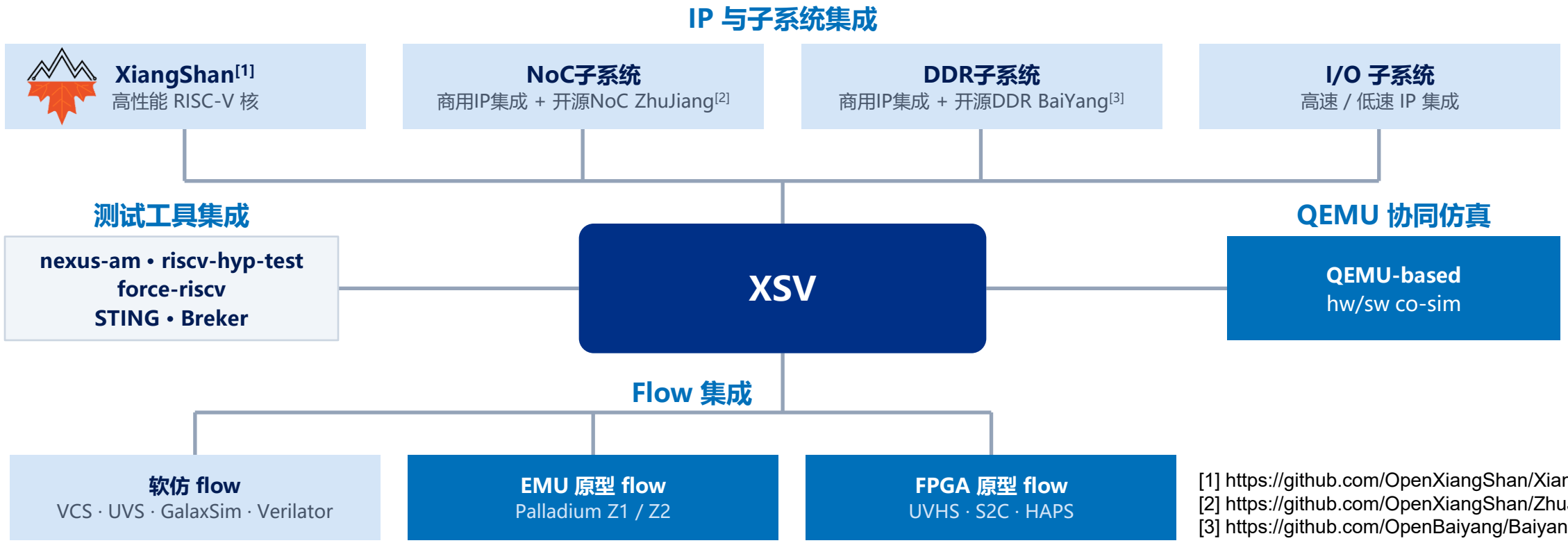
目标愿景

- 为OpenXiangShan社区提供完整的**验证基础设施**与**SoC解决方案**
- 协同RISC-V社区，构建开源芯片技术体系



XSV：一个开箱即用的验证平台

让 RISC-V SoC 的验证从"脚本工程"变成"配置工程"。



[1] https://github.com/OpenXiangShan/XiangShan
[2] https://github.com/OpenXiangShan/ZhuJiang
[3] https://github.com/OpenBaiyang/Baiyang

统一入口

一个 scenario + defconfig 覆盖 UT、ST、IT、EMU与FPGA原型，不再为每个流程学一套命令。

配置即文档

版本、平台、IP、用例全部固化进 defconfig，构建可复现、可 code review、可追溯。

双向可扩展

新增设计复用 flow，新增 flow 服务多设计；验证基础设施随产品线一起生长。