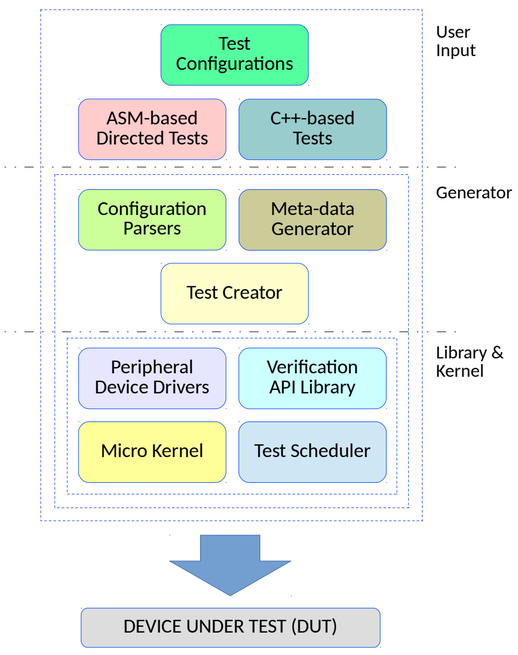
STING – A Complete RISC-V Functional Verification Solution

Introduction



Software stack of test generators, checkers, device drivers, API library and micro kernel; Can be flexibly configured into a portable bare-metal program

Custom DSL (configuration file based mechanism) and programming frameworks allows development of constrained random, directed, use-case or graph-based tests

Lightweight and deterministic kernel with a very small instruction and memory footprint

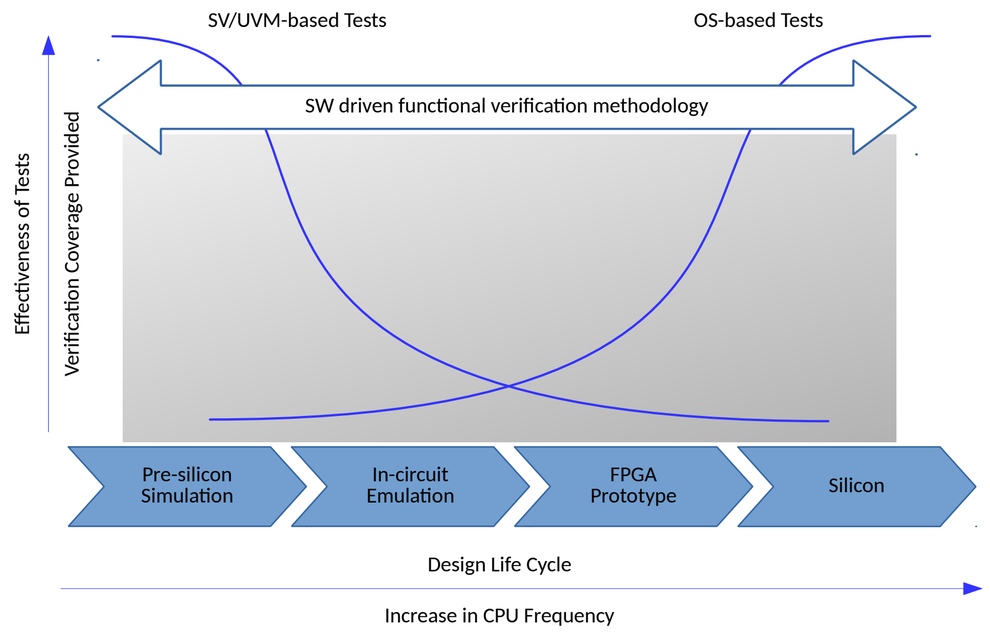
Modes of execution to make the most efficient use of cycles in any verification environment

Supports all the IPs present in the SoC

Can be used out-of-box for supported IP/SoC implementations or ported with minimal efforts for new ones

Generalized architecture agnostic solution for software-driven functional verification methodology

Software Driven Verification Methodology



Allow **consistent execution on all verification environments**

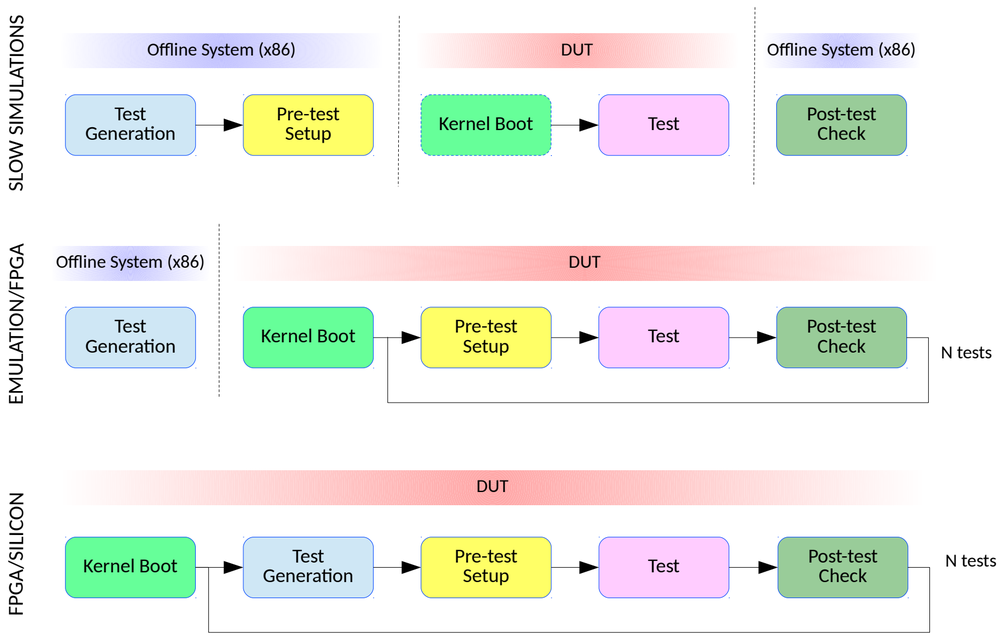
Test stimulus once developed can be reused easily across the design life cycle

Failures hit on silicon can be easily migrated to an earlier stage for faster debug

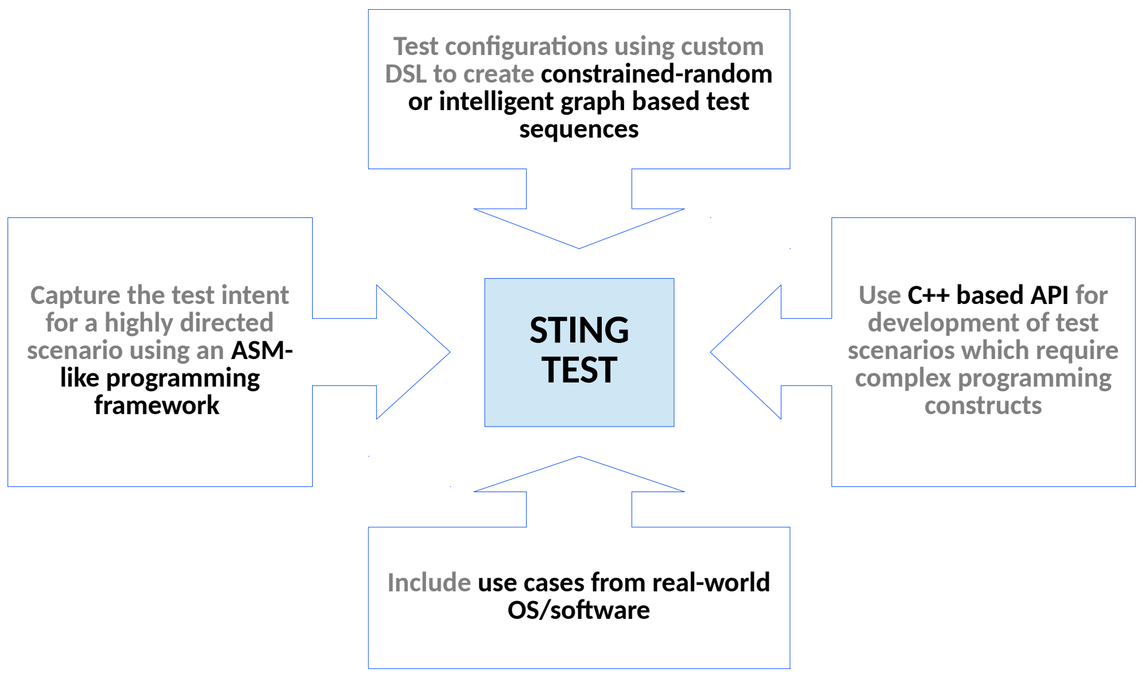
Early enabling of software based stimulus increases the chances of hitting complex bugs early

Save on duplicate efforts spent on verification and validation by acting as a bridge between the two methodologies

Diverse Modes of Execution



Support for Functional Verification Methodologies



Test Checking Mechanisms

COSIM based Checking: Run the test on simulator (ARM Fast Model, SPIKE, or QEMU) first and then on the target hardware followed by comparison of results

On-the-fly Simulation: Simulate the results of the test using APIs for each instruction classes and compare with the results from execution on target hardware

Pass1/Pass2 based Checking: Run the test two (or N) times on the target hardware followed by comparison of results; Crucial in hitting timing related bugs

Self-Checks: Directed tests have custom checks implemented to find if the results are as expected or not

Data Consistency/Corruption Checking: Results of all load and store operations are checked to ensure consistency of data in memory

System Error Checker: Error daemons monitoring occurence of system errors across multiple interfaces are run along with the test

Ease of Usage and Debug

Consistent failure reproduction guaranteed by seed based pseudo random test generation

Ability to start reruns of a test at a later point to enable fast failure reproduction in silicon based environments

Verbose reports and debug information generated with the test to enable quick triage

Helper scripts to identify possible failure points by analysis of failure logs and test reports

Programmable test start and end markers which can be traced during the debug

Directed macros to flag progress of the tests to zero in on a failure point faster

Allow users to **modify one/multiple instructions or a specific set** accessing a memory location in the test stream

Ability to pick and remove parts of test content which may contribute to the failure e.g. all downstream accesess to a MMIO region

RISC-V Verification Package

ARM AVS like package of directed tests and configurations for RISC-V ISA

Package consisting of directed tests, test generators, device drivers and configurations shipped with the tool

Covers all the important architectural and micro-architectural scenarios supported in the SoC

New scenarios can be easily created **using programming specifications and test specification mechanisms**

Brief overview of the verification package is given below -

<u>Test applications for verifying -</u>
Memory subsystem & cache hierarchy
Coverage and dependencies in ISA
Branch unit and branch prediction
Floating Point/SIMD compliance
Memory ordering and consistency

<u>Device drivers for -</u>
High speed I/O (PCI express, USB, SATA, etc.)
Low speed I/O (GPIO, I2C, SPI, etc.)
DMA Controllers
UART
Interrupt controllers (GIC, PLIC)
Memory controllers
Clock/PLL
Ethernet
Timer
SD/MMC

<u>Directed Tests for -</u>
Memory ordering scenarios
Self/cross modifying code
MMU aborts
Starvation scenarios
Register renaming stress
Pipeline hazards
Message passing

Forward progress throttling
Cache/TLB evictions
Atomic increments
Updates to MMU structures
Register access

<u>Configurations for -</u>
Coverage of memory attributes
Coverage of exception/privelege levels
Coverage of memory types (coh/incoh)
Power management testing
Coverage of MMU attributes
Cross product testing of CPU and IO
Different spreads of ISA

Concurrent SoC/CPU Testing