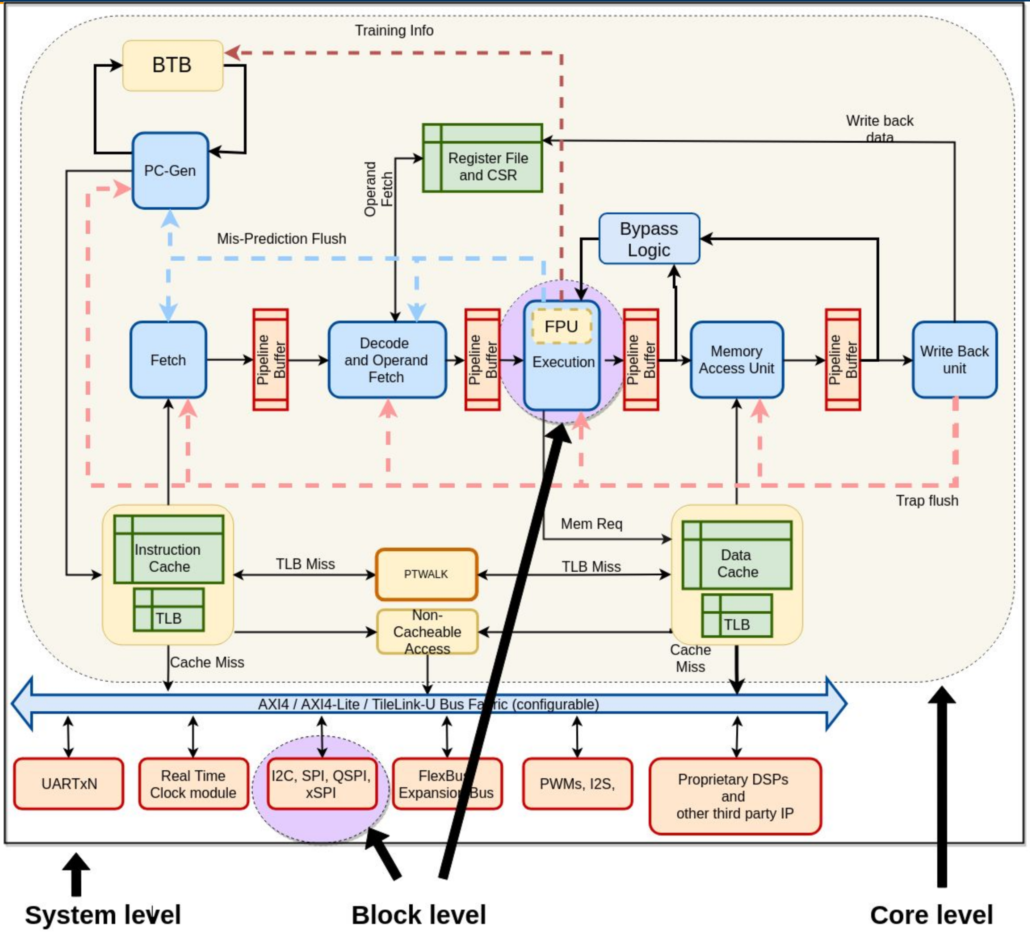
SHAKTI C-Class SoC Verification

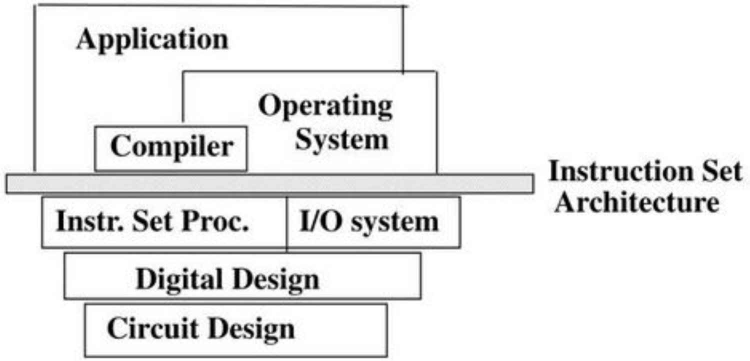
Lavanya Jagadeeswaran

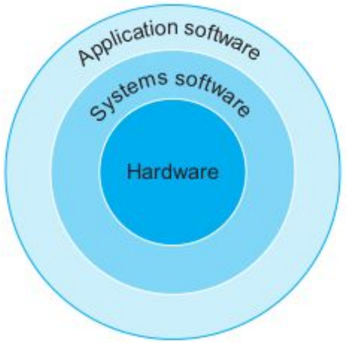
Project Manager | SHAKTI Lab | RISE Group

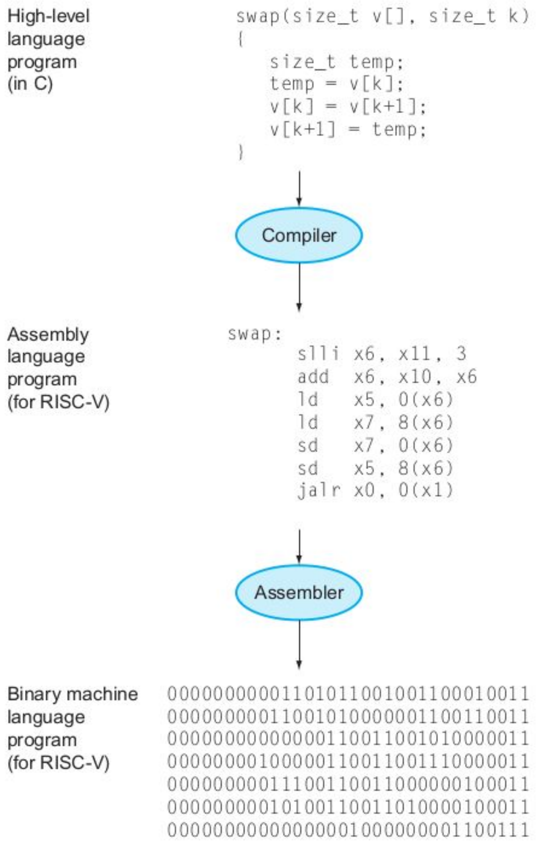
This Session



What is an ISA ?







High-level language program (in C)

```
swap(size_t v[], size_t k)
{
   size_t temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Assembly language program (for RISC-V)

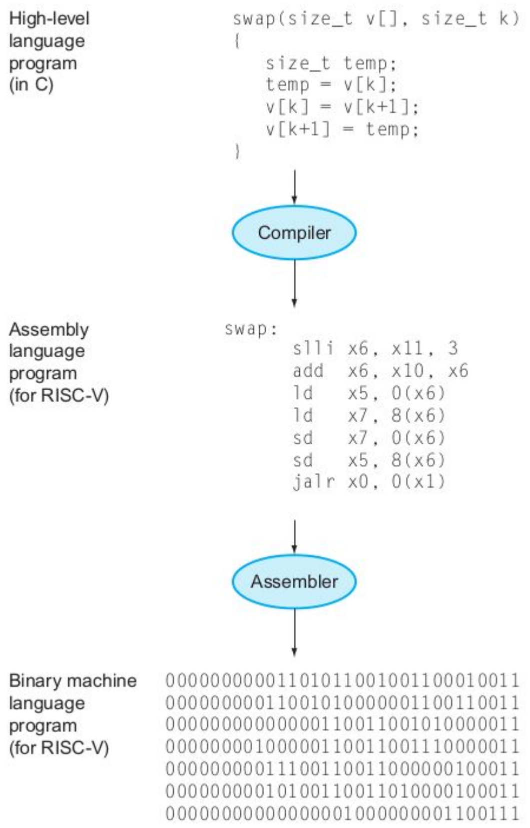
```
swap:
      slli x6, x11, 3
      add  x6, x10, x6
      ld   x5, 0(x6)
      ld   x7, 8(x6)
      sd   x7, 0(x6)
      sd   x5, 8(x6)
      jalr x0, 0(x1)
```

Binary machine language program (for RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000001100111
```

What does any ISA Specification define

- Instruction Encodings
  - Eg. instruction addi -> ‘b0010011
- Register files
  - Eg. x0-x31
  - Control and Status Registers
- Modes of operation


High-level language program (in C)
Compiler
Assembly language program (for RISC-V)
Assembler
Binary machine language program (for RISC-V)

```
swap(size_t v[], size_t k)
{
   size_t temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
      slli x6, x11, 3
      add  x6, x10, x6
      ld   x5, 0(x6)
      ld   x7, 8(x6)
      sd   x7, 0(x6)
      sd   x5, 8(x6)
      jalr x0, 0(x1)
```

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000001100111
```

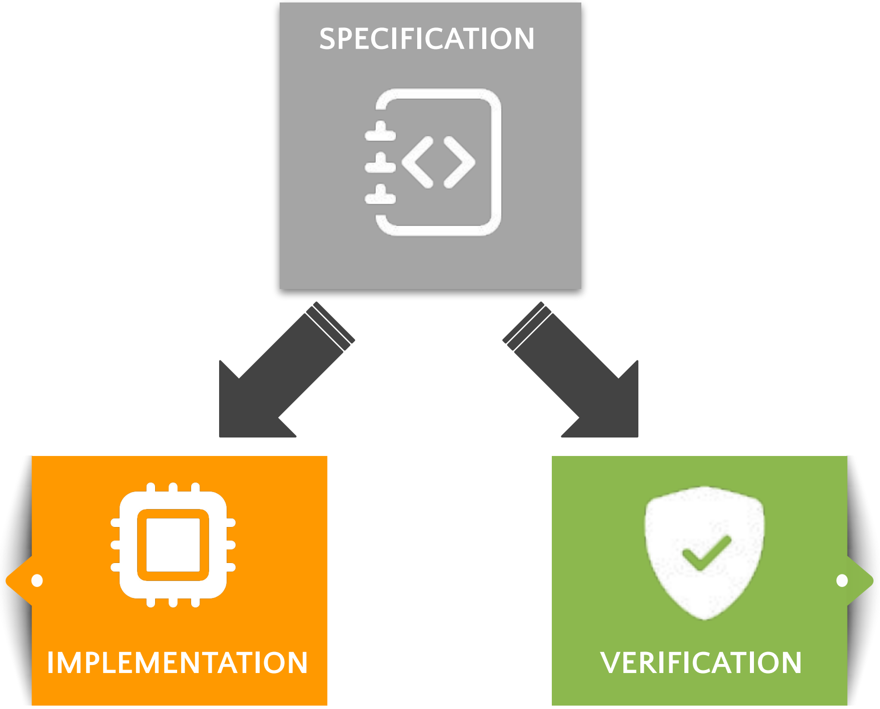
RISC-V Implementation & Why we need it ?

- Numerous processor implementations based on RISC-V
  - IITM's SHAKTI Class of processors, CDAC's VEGA, SiFive's Rocket, Freedom, OpenHW's CV32E40P, CVA6 and many more
- The specification defined in terms of hardware design is known as its implementation
- Isolation of architecture from implementation
- In order to bring out a bug free implementation, verification engineers are expected understand the architecture and basic underlying principles governing the implementation
  - Pipelining: Hazards, Memory hierarchy, Branch predictions, In order, Out-of-order processors

How to verify them ?

Verification - Catch bugs!

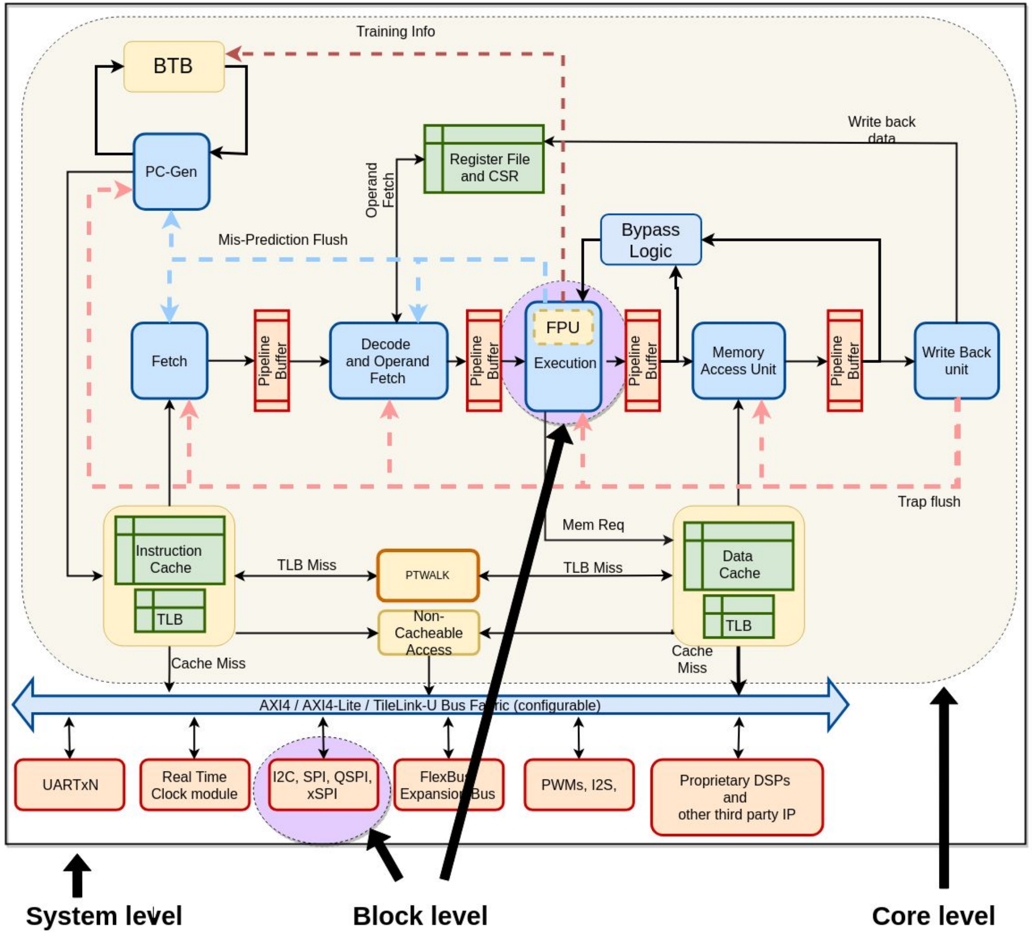
- Demonstrates functional correctness
- Based on the same specification
- Bug escapes to silicon is costly
- More than 50% of resource (time, money, manpower) spent on verification



SHAKTI Verification

- SHAKTI Verification is based on open-source tools and the framework is maintained commonly for various classes of RISC-V cores like C-Class, E-Class, I-Class maximizing reuse.
- Comprehensive suites of directed and random assembly tests are simulated on the Bluespec generated verilog design using Verilator
- Processor verification incorporates ISA level state checking at every instruction execution along with end of test memory check.
- Self-checking frameworks are developed to aid simulation and FPGA level verification

SHAKTI Verification Levels



- **Block level**
  - UVM methodology
  - Interaction with DUT using CoCoTb libraries
- **Core level**
  - RISC-V core verification
  - Framework generating and simulating directed/random tests
- **System level**
  - Simulation & FPGA based verification

Generic Verification Methodology

Test Plan Preparation

- Feature extraction
- Test scenarios
- Checking / Coverage of test feature

Test Bench Development

- Test Bench Components
- Constraint random test sequences
- Coverage definition

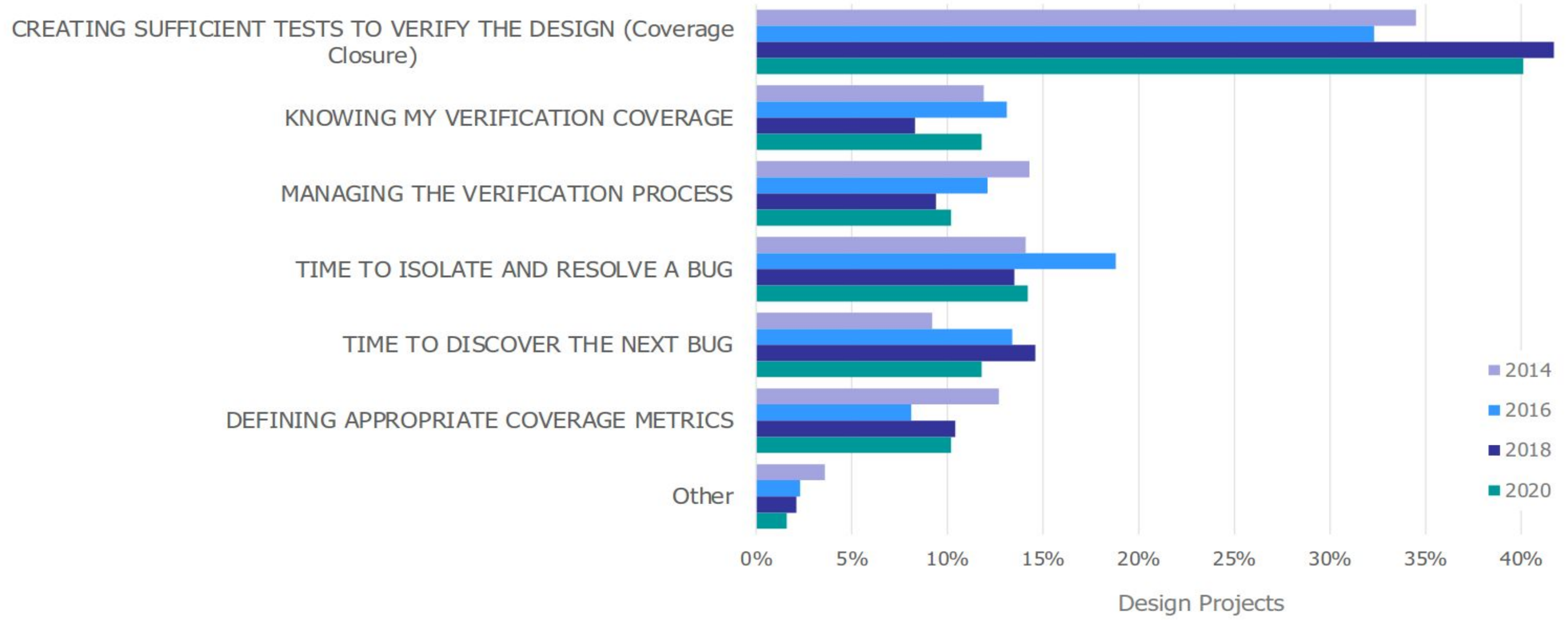
Regression Setup

- Shift left approach
- Continuous Integration (CI)

Sign-off

- Coverage closure
- Code and functional coverage
- 100% holes explained

Verification Challenges



Ref: Wilson Research Group and Mentor, A Siemens Business, 2020 Functional Verification Study

Why Verification is Important ?!

Bug	Year	Loss
Intel's FDIV Bug	1994	**$475 million** for replacements ~ $752 million in 2020.
Intel's Cougar Point chipset problem	2011	$300 million in lost sales and **$700 million** in repairs
Spectre and Meltdown Secutiry Flaws Affect Intel, ARM, and AMD	2018	~ **$18 billion**

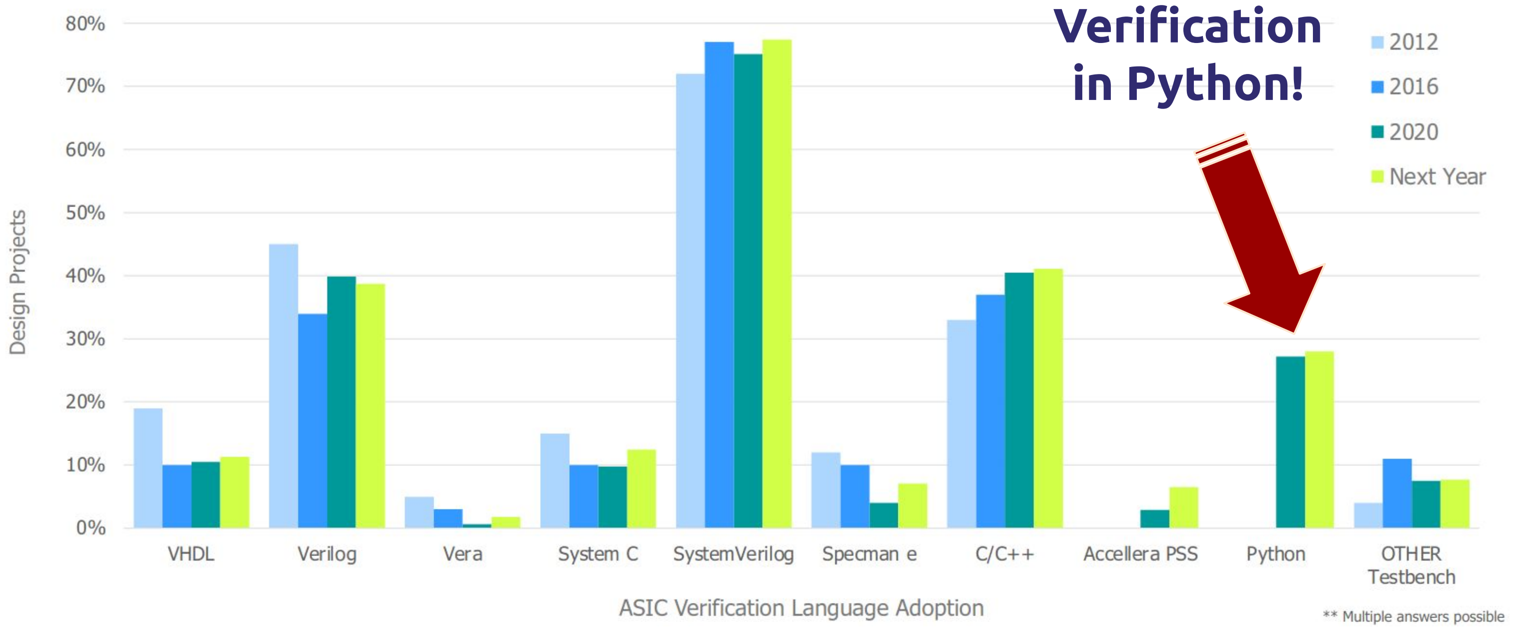
Refs:

https://en.wikipedia.org/wiki/Pentium_FDIV_bug

https://www.anandtech.com/show/4143/the-source-of-intels-cougar-point-sata-bug

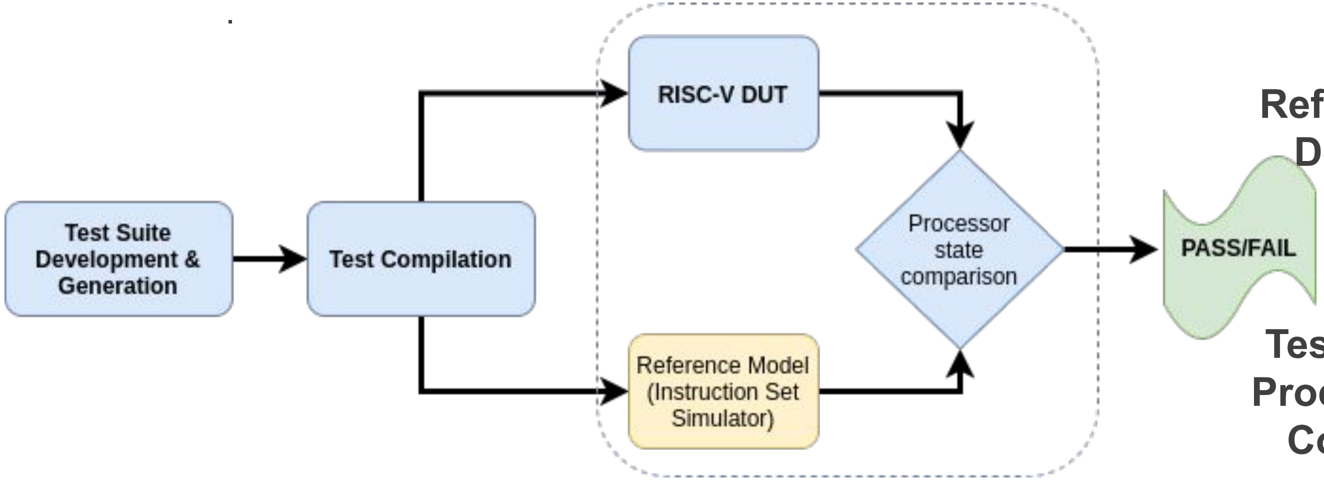
https://www.statista.com/statistics/800258/worldwide-meltdown-spectre-potential-mitigation-cost-by-device-type/

Emerging Trends



Ref: Wilson Research Group and Mentor, A Siemens Business, 2020 Functional Verification Study

RISC-V Verification Components


Test Suite Development & Generation
Test Compilation
RISC-V DUT
Reference Model (Instruction Set Simulator)
Processor state comparison
PASS/FAIL

Test Suite Development & Generation

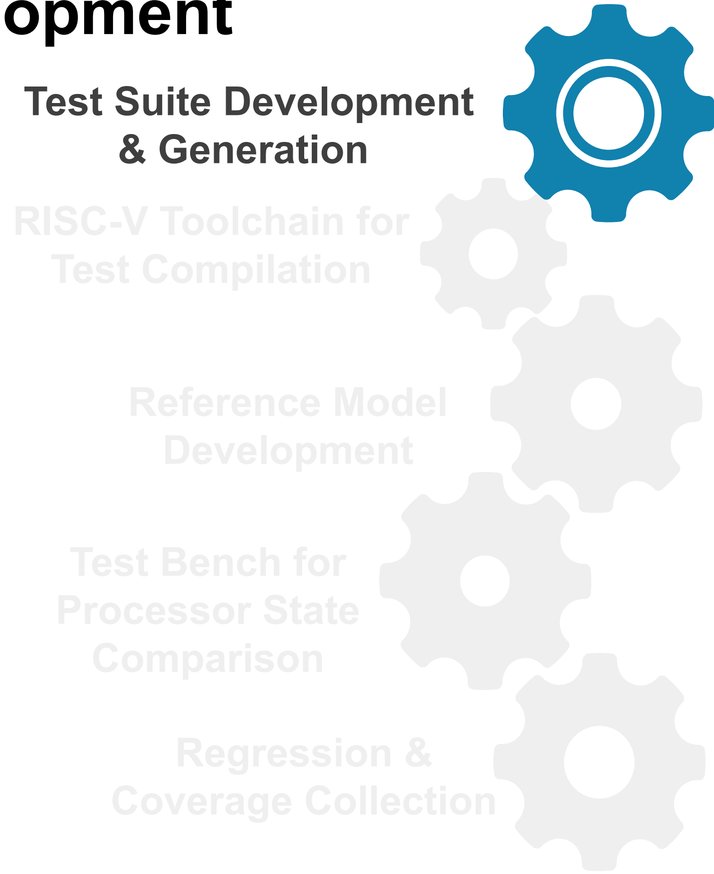
RISC-V Toolchain for Test Compilation

Reference Model Development

Test Bench for Processor State Comparison

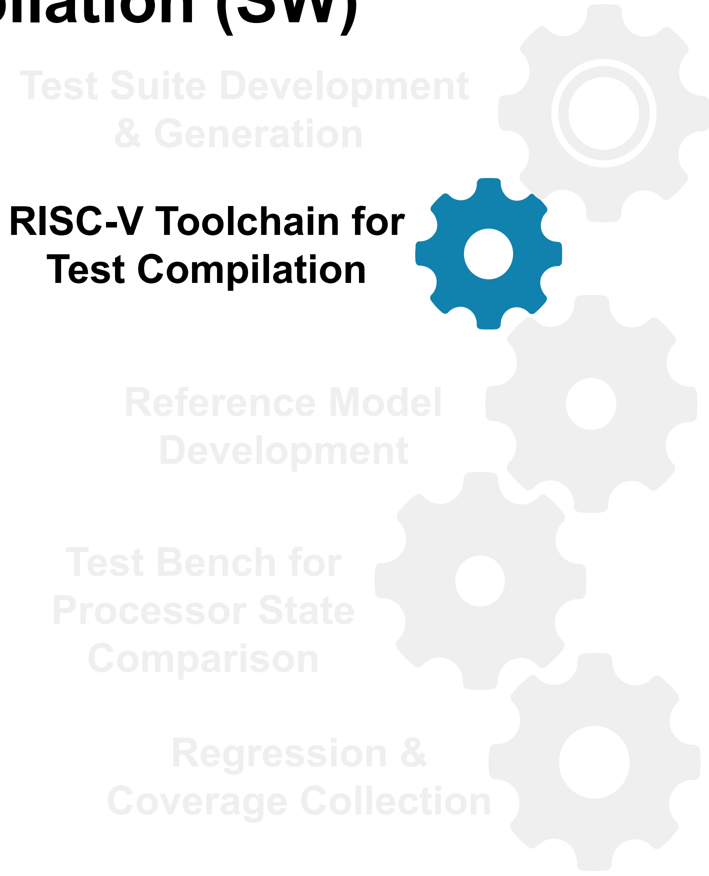
Regression & Coverage Collection

Verification Components - Test Development


Test Suite Development & Generation
RISC-V Toolchain for Test Compilation
Reference Model Development
Test Bench for Processor State Comparison
Regression & Coverage Collection

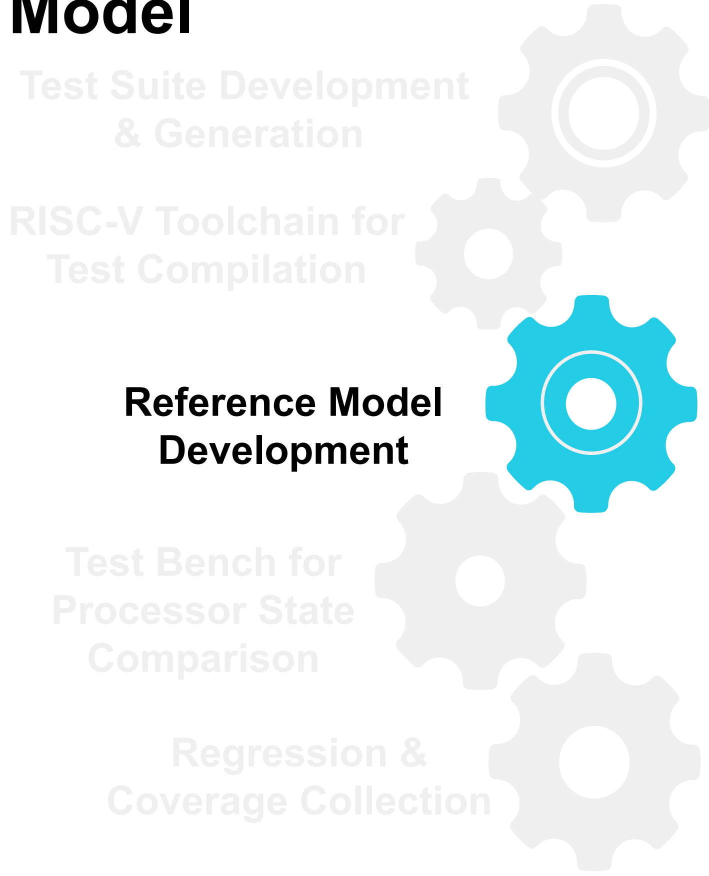
- For RISC-V core verification purposes, self checking assembly tests are used
  - riscv-tests
  - riscv-arch-tests
  - Implementation specific tests
- RISC-V Random test generation
  - Shakti's AAPG
  - RISCV-Torture
  - Google's RISCV-DV
  - MicroTesk
  - Force-RISCV

Verification Components - Test Compilation (SW)


Test Suite Development & Generation
RISC-V Toolchain for Test Compilation
Reference Model Development
Test Bench for Processor State Comparison
Regression & Coverage Collection

- riscv-gnu-toolchain
- For custom extensions, toolchain support has to be added

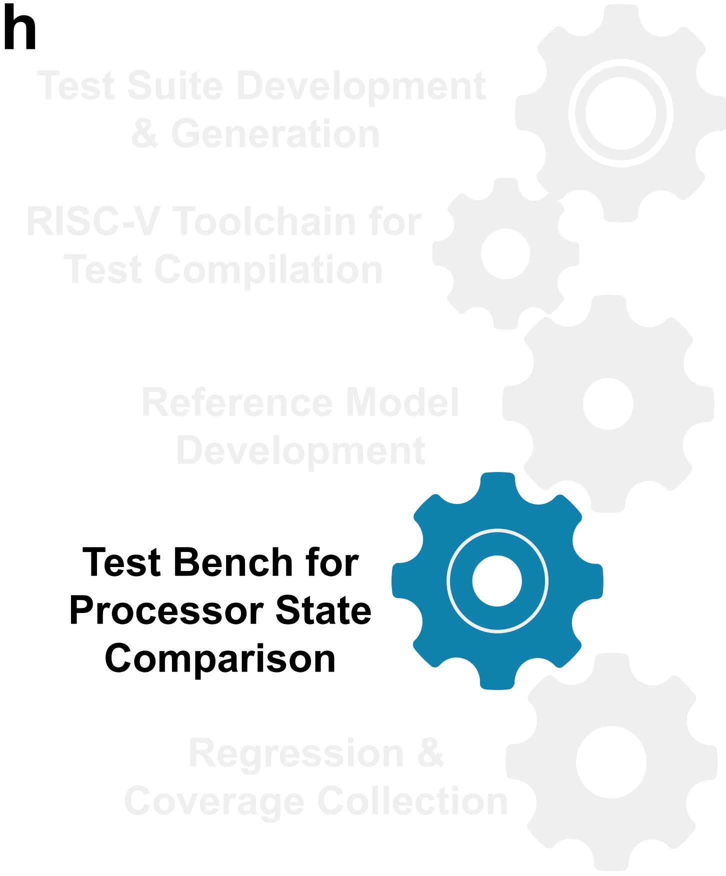
Verification Components - Reference Model


Test Suite Development & Generation
RISC-V Toolchain for Test Compilation
Reference Model Development
Test Bench for Processor State Comparison
Regression & Coverage Collection

- Spike, the Instruction Set Simulator is used as the reference model
- Alternate commercial support: riscvOVPSim

Verification Components - Test Bench

- Shakti's C-Class employs log based processor state comparison and UVM based verification environment
- Processor State:
  - XPR, FPR and CSR. mem on loads/stores

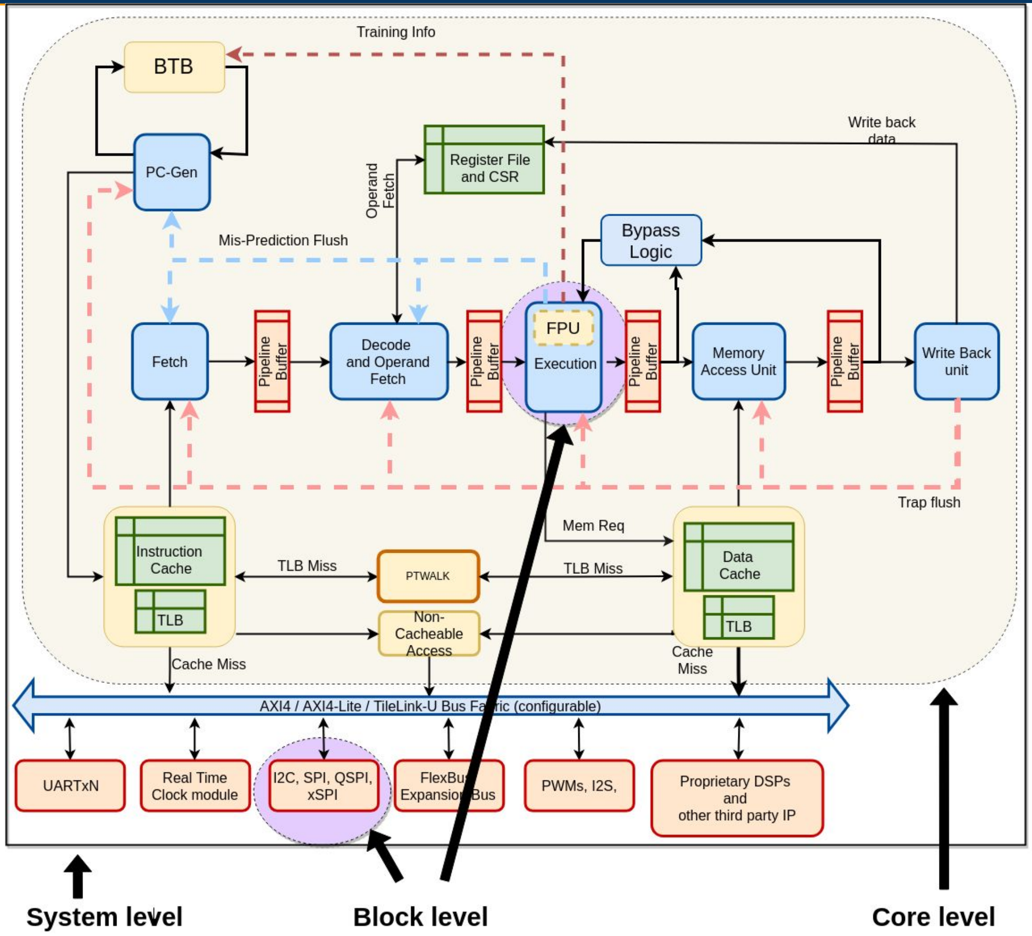

Test Suite Development & Generation
RISC-V Toolchain for Test Compilation
Reference Model Development
Test Bench for Processor State Comparison
Regression & Coverage Collection

Verification Components - Regression

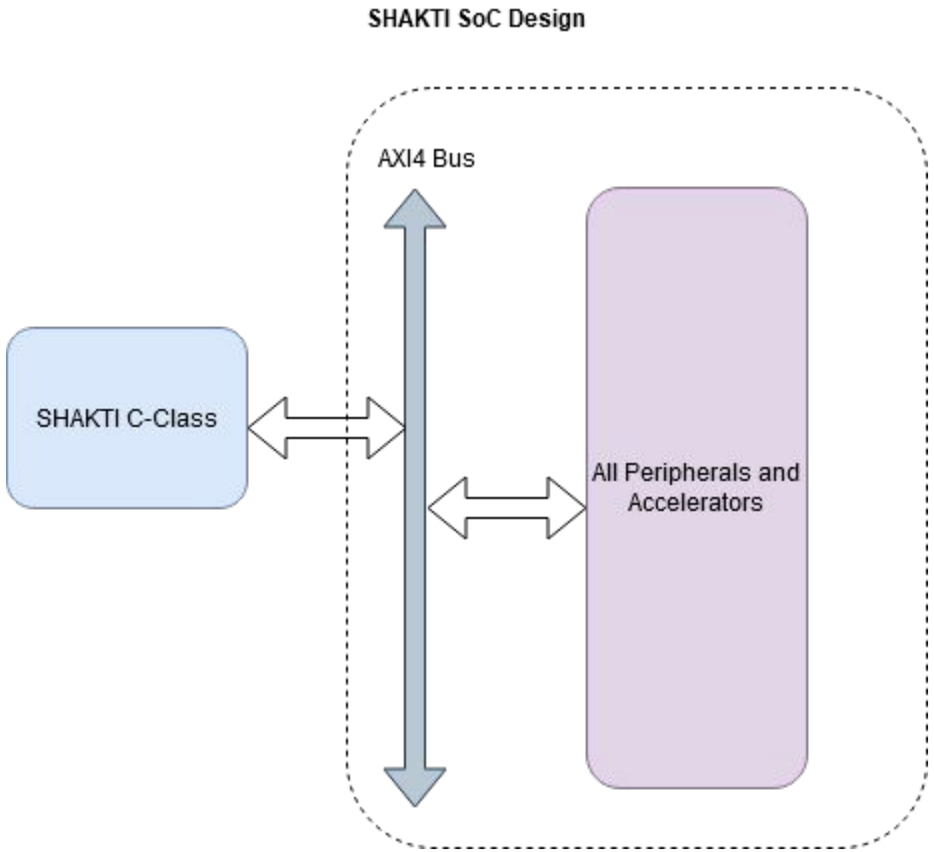
- Shift-Left Approach
- Design and verification starts in parallel
- Verification support incrementally added based on the design feature addition
- Design merge triggers smoke regression
- Nightly regression using Continuous Integration

Test Suite Development & Generation
RISC-V Toolchain for Test Compilation
Reference Model Development
Test Bench for Processor State Comparison
Regression & Coverage Collection

SHAKTI SoC Verification Summary

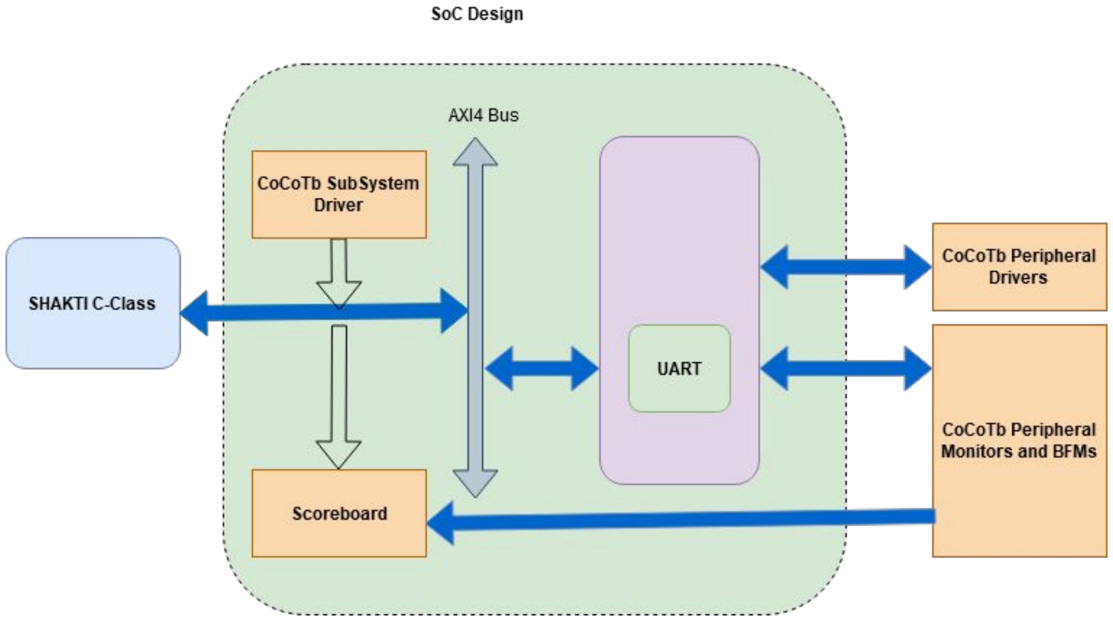


SHAKTI C-Class System Level Verification



- SHAKTI C-Class is connected to the SoC subsystem using the AXI4 interface
- SHAKTI C-Class core Verfication with Assembly test cases
- Memory Mapped Peripherals and accelerators

SHAKTI SoC Sub-System Level Verification



- AXI4 interface as a common driver component
- By generating various AXI transactions to the sub-system, the SoC is being verified.
- The transactions can be towards verifying a Single IP or the whole interactions with the subsystem
- CoCoTb VIPs used for Verification

Python based SHAKTI SoC Verification

Constraint random tests ✓

Test bench components ✓

Reference model development ✓

Coverage definition ✓

Continuous integration ✓

References

- RISC-V History: https://riscv.org/about/history/

- RISC-V Specifications https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154769/RISC-V+Technical+Specifications

- Computer Organization and Design: The hardware / software interface - By David A. Patterson and John L. Hennessey

- The RISC-V Reader: An Open Architecture Atlas authored by David Patterson, Andrew Waterman

- SHAKTI AAPG: https://gitlab.com/shaktiproject/tools/aapg

- riscv-dv: https://github.com/google/riscv-dv

- riscv-tests: https://github.com/riscv/riscv-tests

- riscv-torture: https://github.com/ucb-bar/riscv-torture

- spike: https://github.com/riscv/riscv-isa-sim

- CoCoTb: https://www.cocotb.org/

- UART CoCoTb Ext: https://github.com/alexforencich/cocotbext-uart

Thank You