Scalability of Post-Silicon Test Generation for Multi-core RISC-V SOC Validation

SihPin Tan

YungIt Ho

StarFive Technology International, Malaysia

A Quick Primer On ISG

- A modern SoC design typically incorporates a CPU subsystem that contains a multitude of processor cores executing software code compiled for its Instruction Set Architecture (ISA), e.g. ARM, RISC-V, x86

- To validate the functional correctness of the CPU vs. the ISA specifications, various verification tests may be enlisted to verify the different ISA aspects – Instruction Stream Generators (ISG) are an important category of ISA test tools

- An ISG generates randomized sequences of assembly instructions (ASM) to target architectural corner cases that are harder to achieve with conventional compiler-generated software written in higher-abstraction languages like C/C++

- Generating such random ASM sequences requires an ISG to accurately model the target ISA's behavior on every attempted instruction and determine the next step given the possibility space, making ISG test generation a highly compute-intensive process

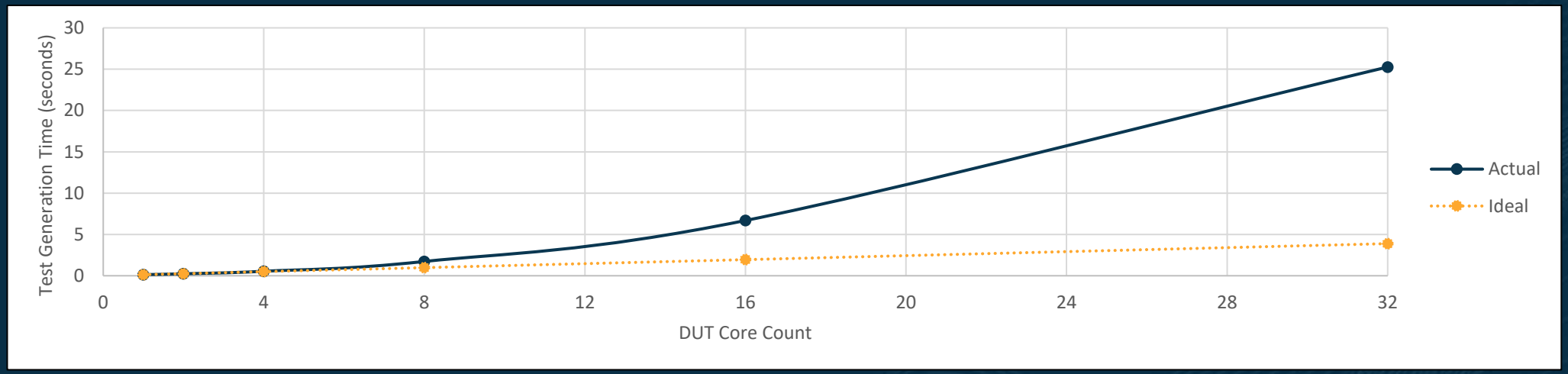
The ISG Scalability Problem

- A modern ISG often runs on a separate system from the DUT, known under the “off-platform” test generation scheme
- De-coupling from the DUT allows the use of much more powerful “host” computers to achieve a higher throughput via parallelism
- Increasing ISA complexity and CPU core count with successive generations of SoC designs therefore translates to ballooning demands on hosts’ compute capacity
- This study demonstrates that ISG compute needs increase in a worse-than-linear fashion vs. the DUT’s CPU core count and output test length (ASM count)
- Futurewei’s open-source FORCE-RISCV ISG was chosen as the testbed for this study as it natively supports symmetric multiprocessing (SMP) CPU targets, allowing simulation of a server-grade DUT that scales to many-core designs

Characterizing ISG Performance

Test Generation Throughput vs. DUT Core Count

- Time taken by the ISG to generate a single test, against number of DUT cores, with other parameters unchanged:

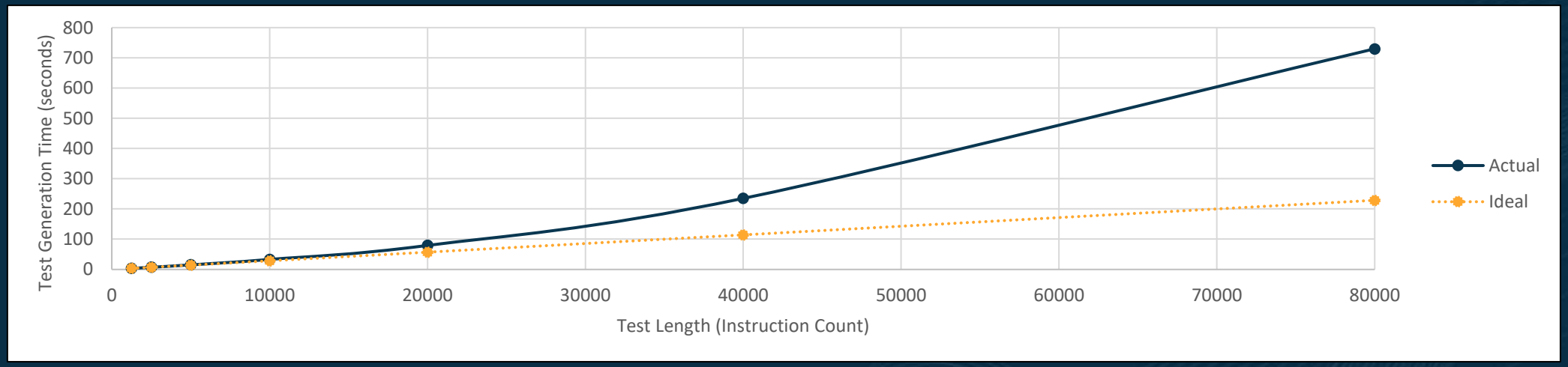


- The test generation time increases exponentially once core count exceeds 4. At the far end, generating a test for a 32-core target would incur 6.5 times the compute costs vs. a simple linear extrapolation from the 1-core baseline
- This results in either drastically higher acquisition costs for more generator compute capacity to maintain the desired test generation throughput, or drastically lower test generation throughput (fewer tests, lower validation quality) to maintain the compute budget

Characterizing ISG Performance

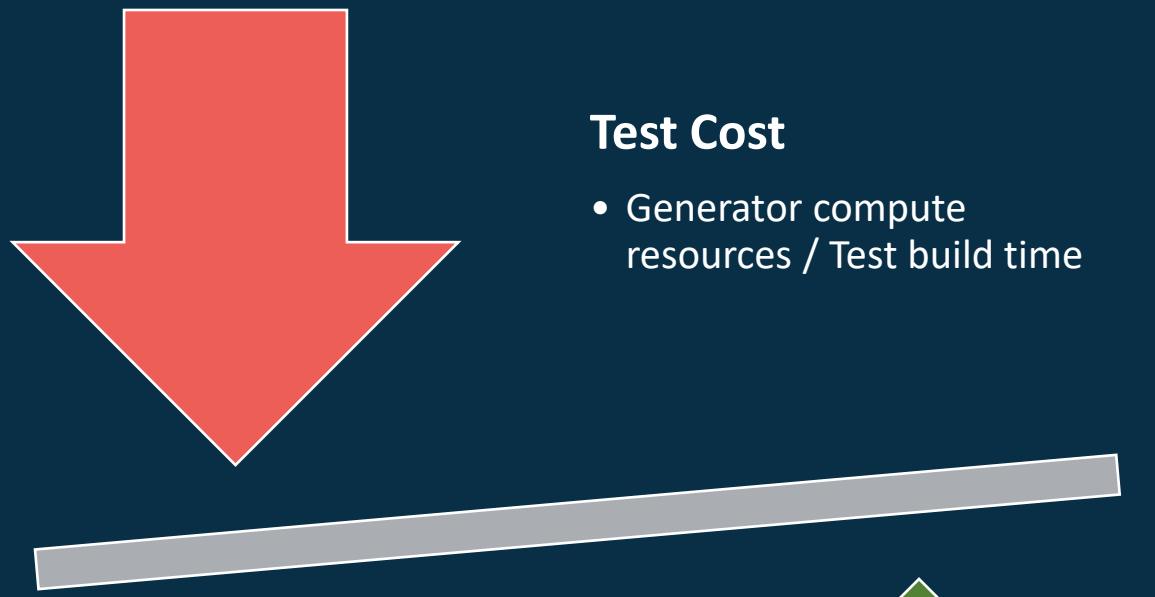
Test Generation Throughput vs. Test Length

- Time taken by the ISG to generate a single test, against test length (number of unique RISC-V assembly instructions per test), with other parameters unchanged:



- In this example, 10,000 instructions would be the sweet spot for maximizing test generation throughput vs. compute acquisition costs
- Going beyond this sweet spot would mean a disproportionate increase in test generation time, resulting in either higher compute costs to maintain the same number of tests overall, or lowered validation quality by sacrificing the total volume of tests

Balancing Test Cost and Validation Quality



Test Cost

- Generator compute resources / Test build time

Validation Quality

- Validate all available CPU cores on the DUT
- Maintain per-core instruction count at a level sufficient to provide the desired architectural interactions and permutations

- **Less motivation to increase per-core instruction count beyond the optimum point** It is more desirable to spread the validation goal of total instruction count across a higher number of optimally sized tests, as opposed to a lower number of larger tests – higher level of test permutations and therefore broader validation coverage

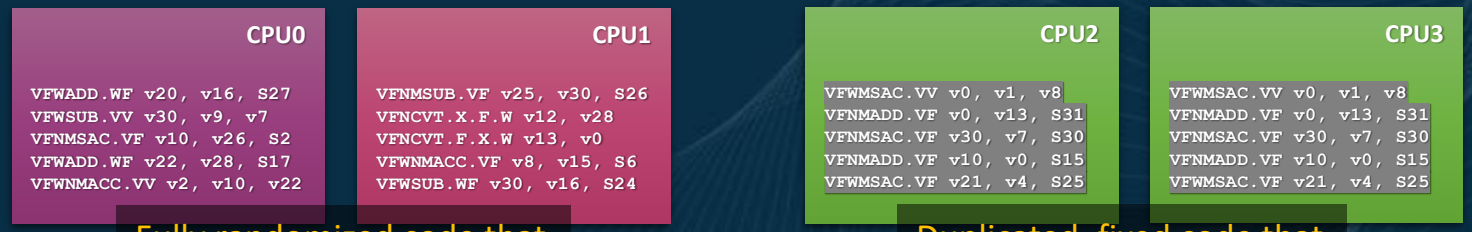
- **Reducing per-core instruction count below the optimum level** means the resulting tests may not be long enough to adequately fill up or “warm” the core’s internal caches. The tests also may not create sufficient data reuse and address aliasing vital for validating cache and translation look-aside buffer (TLB) functionality

- **Reducing the output test’s core count** (while maintaining optimum per-core instruction count), some DUT cores will be left idle and not executing any meaningful code. Loss of coverage on SMP inter-core interactions, including cache coherency, interrupts, power management flows

Proposed Solution

Use Of Fixed Code Sequences To Conserve Compute

- An ISG's compute intensity is driven by the need to accurately model the target's architectural state, as well as resolving the next instruction's randomization space
- A potential solution: selectively replace random instructions with fixed routines that perform pre-determined operations and therefore require less compute
- One option is to replace the entire code section for certain DUT cores with fixed routines, effectively reducing the total number of DUT cores to be processed
- But: insertion of fixed routines must not diminish the value of the ISG or the original test template. Example: template intends to validate Vector instructions, fixed routines of RVV instructions with pre-determined results in some of the cores can still exercise their VPU

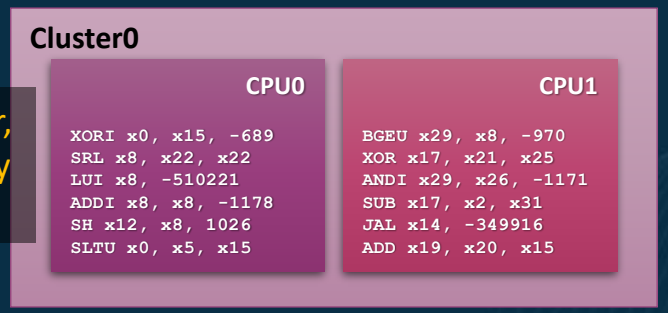


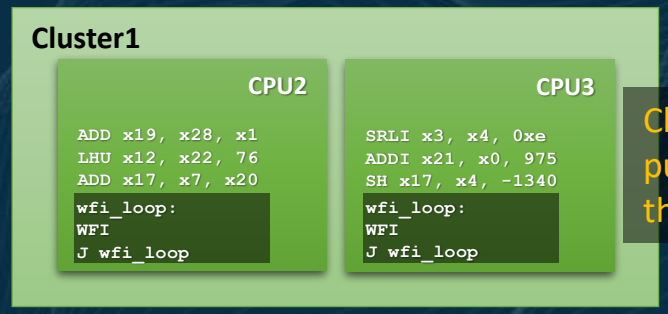
Proposed Solution

Use Of Fixed Sequences To Validate Specific Conditions

- Replacing fully randomized instruction streams with fixed routines may make the test more “static” and less random, BUT this directedness can be harnessed to achieve specific validation goals
- Example: Hierarchical CPU power management with core- and cluster-level idle states
  - RISC-V cores execute the WFI instruction to enter a low-power state
  - After all cores in a cluster have entered sleep, the entire cluster can be taken off-line
- Validation of such hierarchical sleep states traditionally rely on directed tests or driver-type software to ensure coordinated handling of the flow
- Insertion of fixed routines for low-power mode can give ISG a directedness that makes it a viable option for validating CPU power management

Cluster0 cores run regular, randomized code and stay active throughout





Cluster1 cores share fixed WFI block, putting both cores and subsequently the entire cluster to sleep

Proposed Solution

Effective Mixture Of Randomized And Fixed Code Routines

- The key to adding fixed routines to an ISG is balancing the inherent value of the ISG's randomness and the goodness from the added directedness
- If randomness is preserved for much of the test, a combination of randomized and fixed code sequences can help achieve better overall validation by creating conditions that are otherwise impossible with random code or fixed code alone
- In the previous CPU power management example, although cores in Cluster1 share the same WFI low-power code, it's preceded by random code unique to each core
  - This preceding random code serves as a "preamble" that sets up the initial conditions for the WFI code, including internal cache/TLB states, architectural states, and internal micro-arch states
  - Essentially, two permutations of WFI are executed, thereby achieving broader validation coverage than using directed tests with the same preamble code
- This illustrates how adding well-defined fixed routines to an ISG can improve efficiency AND add value and broaden its application scope beyond the conventional ISA-centric validation usage

Conclusion

- ISG are an important component in validating modern SOC designs
  - Yet, the increasing design complexity with each generation of SOC results in rapidly deteriorating ISG test generation throughput
  - A brute-force approach of adding more compute to keep up is not economically viable!
- First step: characterize ISG's performance against DUT core count and test length
  - We saw how compute costs scale against the DUT configurations in a non-linear fashion, and showed how to identify the configuration sweet spots for optimal compute investment
  - But shortening tests without understanding the impact is bad – we need a smarter approach!
- Our alternative: Insert fixed routines into ISG to lessen compute burden
  - We saw how selective replacement of randomized code with fixed routines can reduce costs yet still preserve meaningful validation, despite a reduction in overall randomization
- Going further: How to exploit this loss of randomness to our advantage
  - We saw how fixed routines can help validate conditions that require SMP coordination
  - This hybrid method combines the merits of directed tests and the qualities of an ISG

Thank You!