Chair of Electronic Design Automation
School of Computation, Information and Technology
Technical University of Munich

Towards Coverage Analysis for Translating Instruction Set Simulators

Karsten Emrich, Daniel Mueller-Gritschneder, Ulf Schlichtmann

Motivation

- VPs and ISSs are pivotal for early design verification and architecture exploration
- DSLs and modeling tools such as nML, CODASIP and CoreDSL2 allow fast & easy definition of custom ASIPs
- Functional and formal verification is a crucial aspect of deploying new VPs and ISSs, and increases in complexity once model-based extensible ISAs are used
- Coverage analysis provides a good overview of how much input source is targeted by a given test suite
- Monolithic ISSs are easily tested for coverage with standard tools
- Modern ISSs utilizing DSL-based target ISA specification and dynamic binary translation cannot be analyzed by traditional tools

Goal: Derive and implement a coverage analysis methodology for DSL-based, DBT accelerated simulators

Test Setup

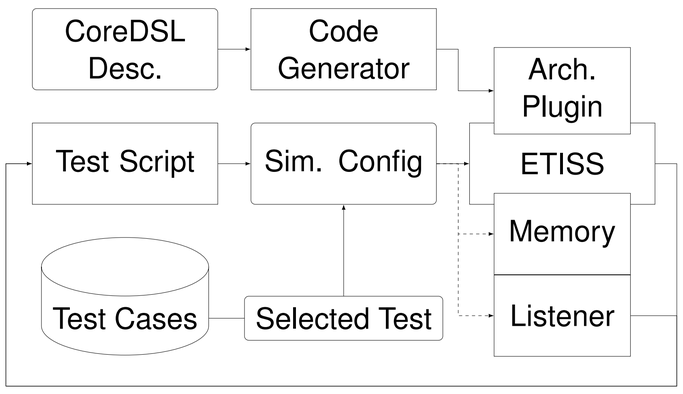

Figure 1 ISS Test Setup

- riscv-tests instruction-level test suite as basic initial plausibility check to show importance of coverage analysis
- Automated test script to simplify targeted, parallel execution

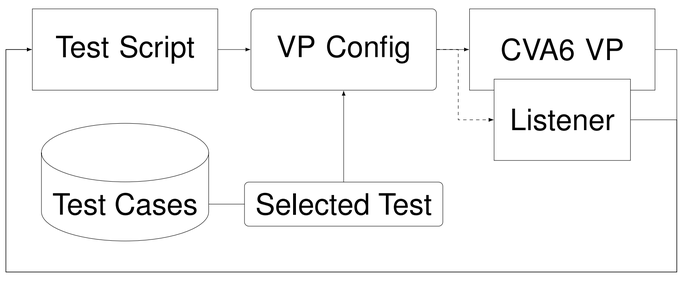

Figure 2 CVA6 Test Setup

- Additional comparison against OpenHwGroup CVA6 processor to check improved test cases

Application in a DBT ISS

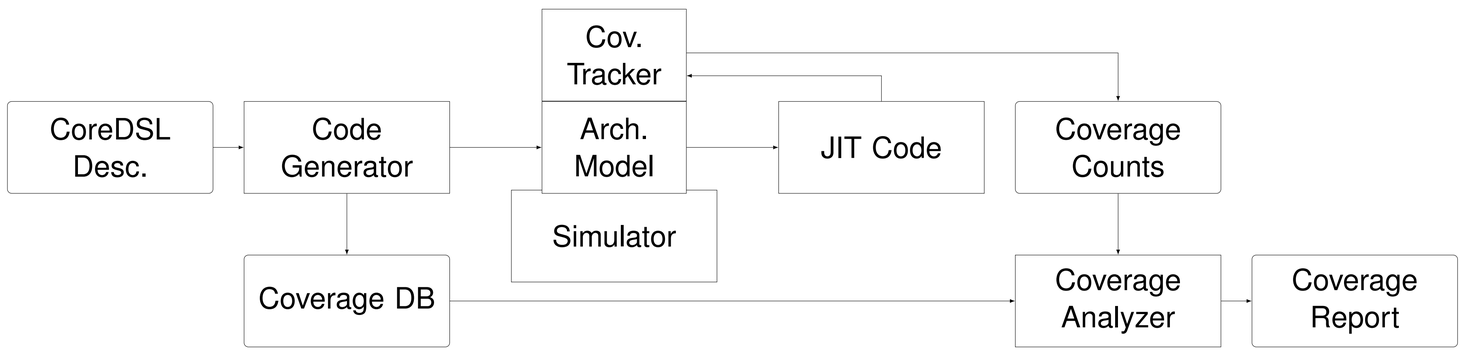

Figure 3 DBT ISS Coverage Analysis Dataflow

- Coverage tracking addition to existing DBT ISS ETISS
- ETISS is built on a CoreDSL2 model description, code generator toolchain M2-ISA-R parses model and generates architecture simulation model
- M2-ISA-R is extended to embed coverage tracking information into simulation model
- Analysis tool consolidates generated coverage data, matches against metadata from architecture model and outputs LCOV compatible report files

Case Study

LCOV - code coverage report

		Coverage	Total	Hit
Current view:	top level - etiss_arch_riscv/rv_base - RVI.core_desc (source / functions)			
Test:	RV64IMACFD.rv.info			
Test Date:	2024-05-08 18:27:20			
	Lines:	94.2 %	156	147
	Functions:	98.2 %	55	54
	Branches:	82.1 %	112	92

```
                 Branch data     Line data    Source code
   1                    :               :  import "RISCVBase.core_desc"
   2                    :               :
   3                    :               :  InstructionSet RV32I extends RISCVBase {
   4                    :               :      architectural_state {
   5                    :               :          XLEN = 32;
   6                    :               :      }
   7                    :               :
   8                    :               :      instructions {
   9                    :               :          LUI {
  10                    :               :              encoding: imm[31:12] :: rd[4:0] :: 7'b0110111;
  11                    :               :              assembly: "{name(rd)}, {imm:#05x}";
  12          [ +  + ]:            2080 :              behavior: if ((rd % RFS) != 0) X[rd % RFS] = (unsigned<XLEN>) ((signed) imm);
  13                    :               :          }
  14                    :               :
  15                    :               :          AUIPC {
  16                    :               :              encoding: imm[31:12] :: rd[4:0] :: 7'b0010111;
  17                    :               :              assembly: "{name(rd)}, {imm:#08x}";
  18          [ +  - ]:            1694 :              behavior: if ((rd % RFS) != 0) X[rd % RFS] = PC + (signed)imm;
  19                    :               :          }
  20                    :               :
  21                    :               :          JAL [[no_cont]] {
  22                    :               :              encoding: imm[20:20] :: imm[10:1] :: imm[11:11] :: imm[19:12] :: rd[4:0] :: 7'b1101111;
  23                    :               :              assembly: "{name(rd)}, {imm:#0x}";
  24                    :             207 :              behavior: {
  25          [ -  + ]:             207 :                  if (imm % INSTR_ALIGNMENT) {
  26                    :               0 :                      raise(0, RV_CAUSE_MISALIGNED_FETCH);
  27                    :             207 :                  } else {
  28          [ +  + ]:             207 :                      if ((rd % RFS) != 0) X[rd % RFS] = PC + 4;
  29                    :             207 :                      PC = PC + (signed)imm;
```

Figure 4 riscv-tests Integer Instructions Coverage Example

- Overall: 89.7% line coverage, 92.5% function coverage, 68.8% branch coverage for riscv-tests
- Critical coverage misses in shift instructions and rounding mode for floating-point instructions reveal bugs in CoreDSL2 and CVA6 cores
- Targeted fixes implemented for coverage misses possible
- **Future Additions:** Analysis of edge cases, register usage, automatic test case generation and extension, performance improvements (~75 MIPS default vs 2.5 with coverage analysis active)

This project is supported by the KDT-JU project TRISTAN under the grant nr. 16MEE0272.

The TRISTAN project, nr. 101095947 is supported by Chips Joint Undertaking (CHIPS-JU) and its members Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey.

Contact:
daniel.mueller@tum.de
karsten.emrich@tum.de

Open Source:
https://github.com/tum-ei-eda