2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Automated Thread Evaluation of Various RISC-V Alternatives using Random Instruction Generation

Endri Kaja, Nicolas Gerlin, Dominik Stoffel, Wolfgang Kunz, Wolfgang Ecker

Infineon

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

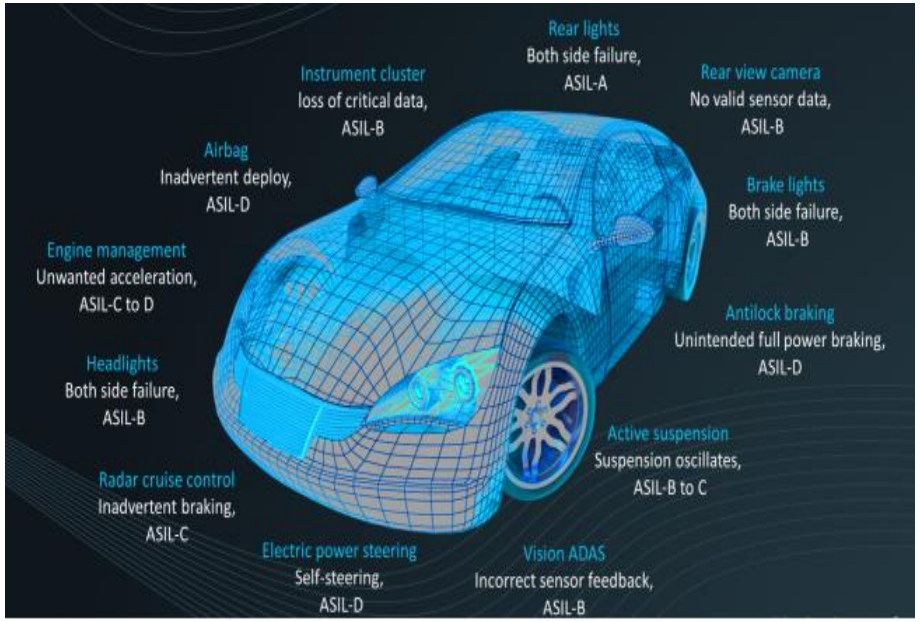
accellera SYSTEMS INITIATIVE

Outline

- Introduction and motivation
- Background
- RISC-V CPU generation
- Model-based fault simulation framework
- Random instruction generation
- Application and results
- Conclusion and future work

Introduction and motivation (1)

- Growth of RISC-V ecosystem into industries
  - AI
  - Machine learning
  - **Safety**
  - Security
- ISO26262 recommends fault injection to verify safety-critical designs
  - ASIL risk classification system

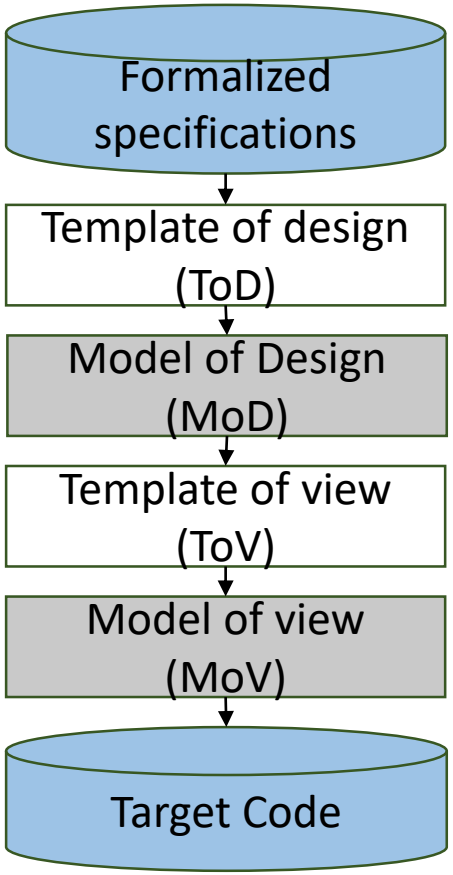


https://www.embitel.com/blog/embedded-blog/understanding-how-iso-26262-asil-is-determined-for-automotive-applications

Introduction and motivation (2)

- Safety-critical designs require rigorous, stringent and automated verification techniques
- Complex testing techniques
- Required to shorten Time-to-Market deadlines
  - ↗ Automation
  - ↗ Productivity
  - ↗ Performance

Background: RTL generation framework (MetaRTL)



Hardware specifications are captured formally

Microarchitecture blueprint

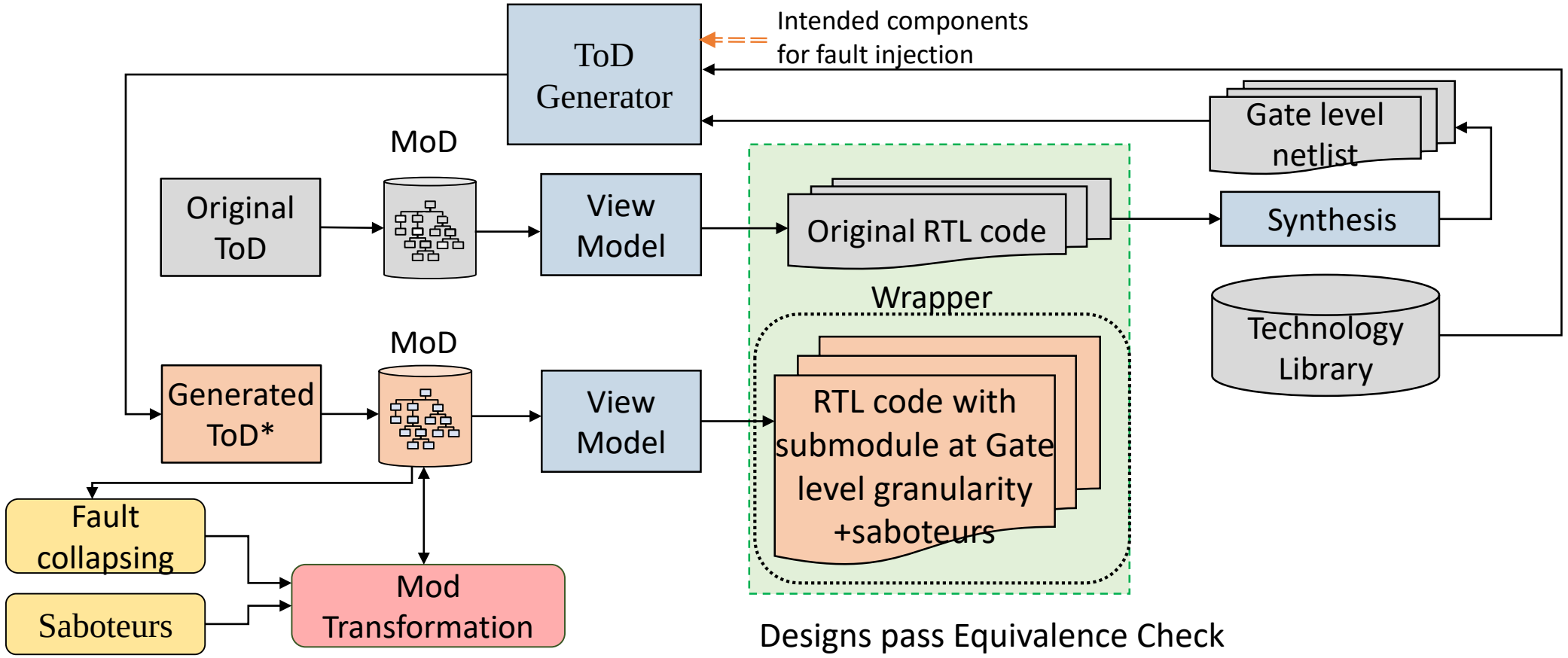
Platform and technology indipendent hardware description

Maps MoD to MoV elements

Tree based, target language specific

RTL HDL

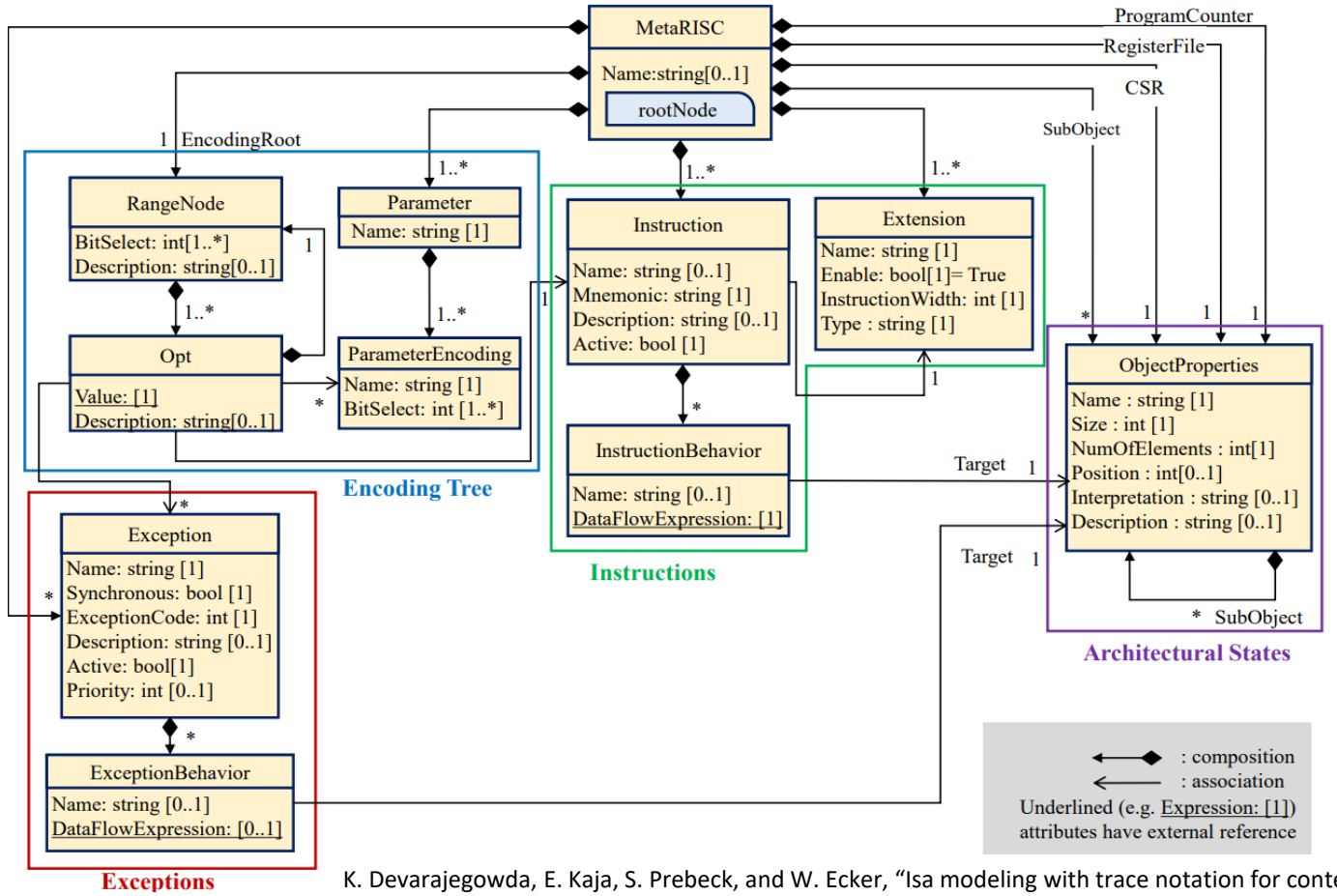
Background: Mixed register-transfer/gate level fault injector



RISC-V CPU generation

- Key element: RISC-V metamodel

- Four main components
  - Encoding tree
  - Instructions
  - Architectural states
  - Exceptions

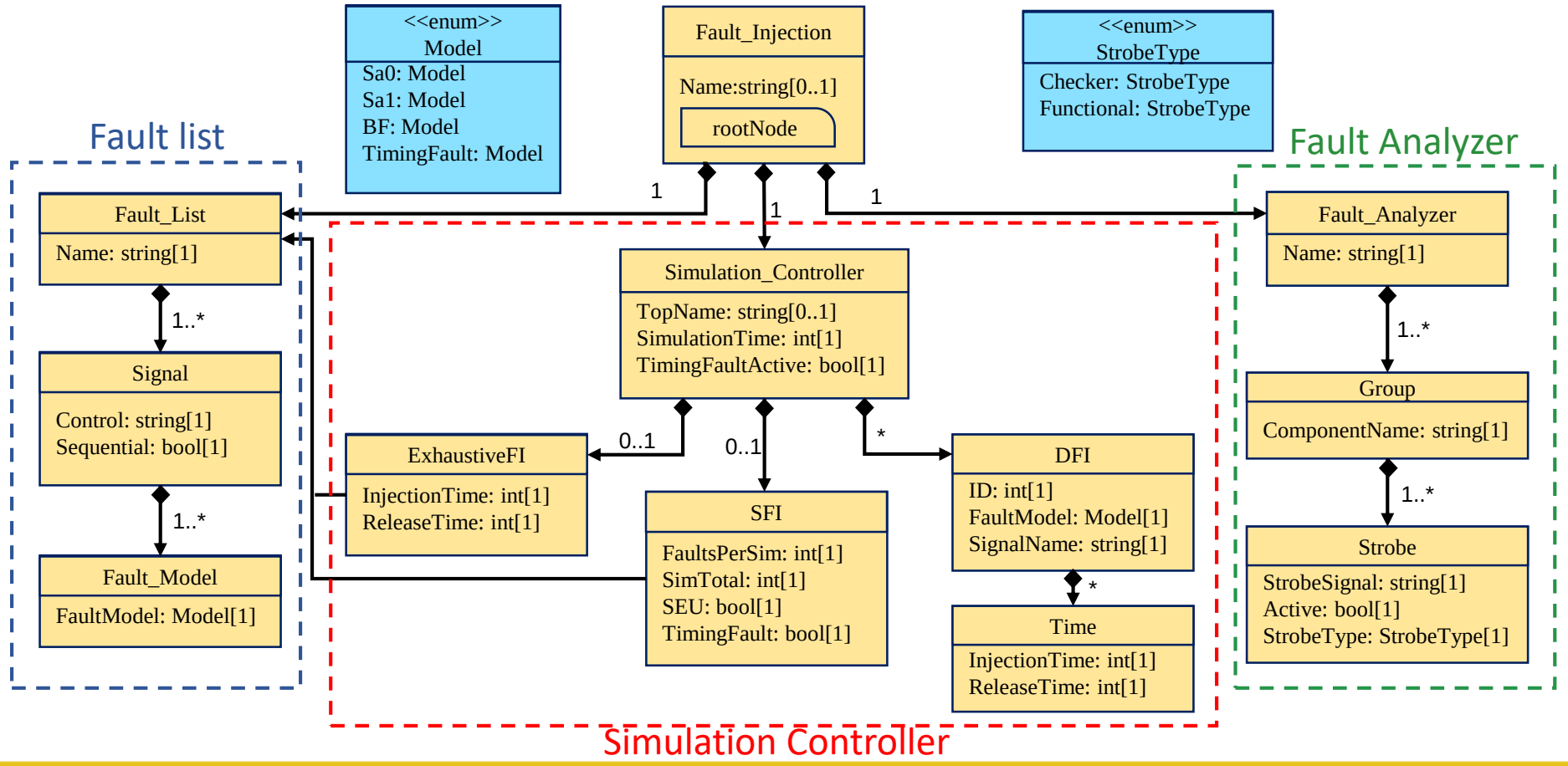


K. Devarajegowda, E. Kaja, S. Prebeck, and W. Ecker, “Isa modeling with trace notation for context free property generation,” in 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021, pp. 619–624

Model-based fault simulation framework(1)

- Fault simulation flow:
  - MetaRTL is utilized to generate the design
  - Fault injection framework transforms the design
  - A script extracts information from the design and generates the fault list
  - User configures the model accordingly
  - The framework generates SystemVerilog and C++ testbench in accordance with the model's data

Model-based fault simulation framework(2)



Model-based fault simulation framework(3)

Line 3 represents the simulation count

Line 4 defines the timepoints when the fault should be injected

Lines 10 determines the fault model to inject

```
1   always @(i, k) begin
2       if (!fault_injected) begin
3           if (i==1) begin
4               if (k >= 50 && k <= 52) begin
5                   if (!wr_flag) begin
6                       $fwrite(fault_sim_report, "%s%s", unify_char_nr("1", "post", 13),
7                       unify_char_nr("50", "post", 21));
8                       wr_flag = 1;
9                   end
10                  FI_Control_All_834 = 3'b101;
11                  bf_end = 1;
12              end
13              else if (bf_end) begin
14                  FI_Control_All_1 = 3'b100;
15                  fault_injected = 1;
16                  bf_end = 0;
17              end
18          end
```

Model-based fault simulation framework(4)

Fault_Sim	Fault_Inject_Time	Fault_Propagate_Time	Propagated_FStrobe	Fault_Detect_Time	Detected_CStrobe
1	50	101	comp_TopSoC.comp_MinimumSoc.comp_CoreGen.IB_addr_out	na	Fault not detected by checker strobe
2	50	177	comp_TopSoC.comp_MinimumSoc.comp_CoreGen.DB_wdata_out	na	Fault not detected by checker strobe
3	50	na	Fault did not propagate	na	Fault not detected by checker strobe
4	50	101	comp_TopSoC.comp_MinimumSoc.comp_CoreGen.IB_addr_out	na	Fault not detected by checker strobe
5	50	101	comp_TopSoC.comp_MinimumSoc.comp_CoreGen.IB_addr_out	na	Fault not detected by checker strobe

Safe Undetected Faults: 1
Safe Detected Faults: 0
Dangerous Undetected Faults: 4
Dangerous Detected Faults: 0
Total Faults: 5

- **Safe undetected faults**: did not propagate to any strobe
- **Safe detected faults**: detected and fixed by the safety mechanism
- **Dangerous undetected faults**: propagate to the functional strobes but are not detected by the safety mechanism
- **Dangerous detected faults**: propagate to functional strobes but are detected by the safety mechanism

Random instruction generation(1)

- Automatic Test Pattern Generation (ATPG)
  - Common technique to test safety-critical desings
  - Very high fault coverage but suffers from performance constraints
- ISO 26262 recommends various ASILs for different designs
- Combination of **model-driven fault simulation framework** with **model-driven random RISC-V instruction generation**

Random instruction generation(2)

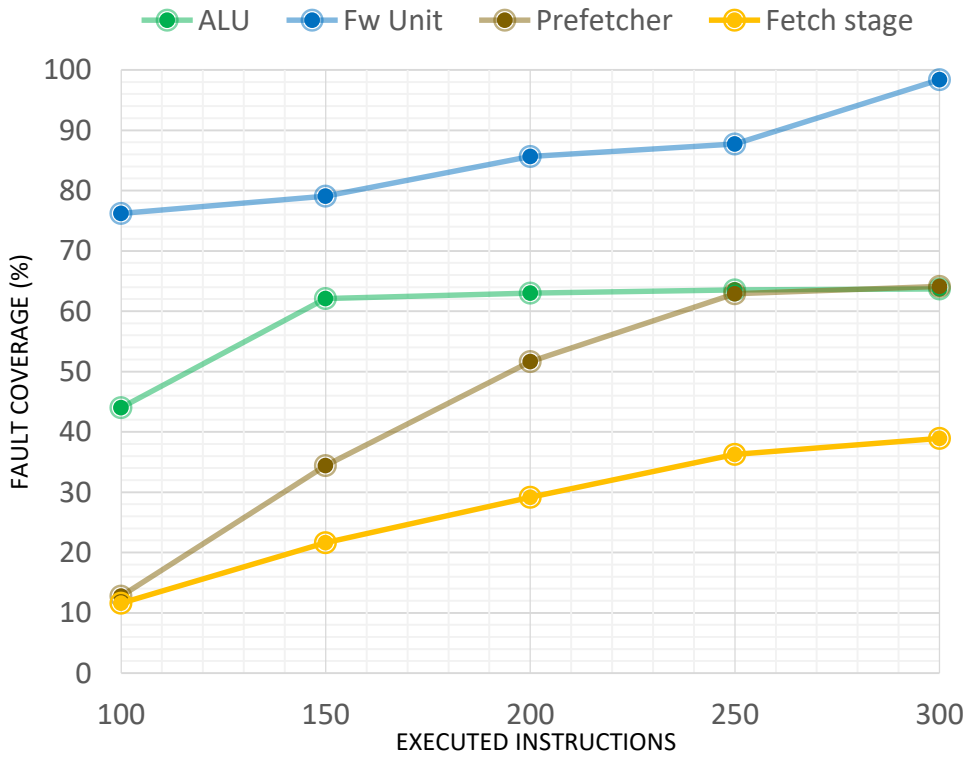
- Key element: RISC-V metamodel (slide 7)
- A python scripts reads the model data and generates random valid instructions
- Highly configurable
  - Constraints
  - Length
  - File Type
  - Memory Start Address

Application and results(1)

- The set of random instructions utilized as test input for the RISC-V CPU
- Fault simulation applied on a CPU subsystem with various RISC-V alternatives
- The tests are run on three similar subsystems with different RISC-V alternatives, i.e. only the CPU changes
- Faults injected on ALU, Forwarding Unit, Prefetcher, and Fetch stage

Application and results(2)

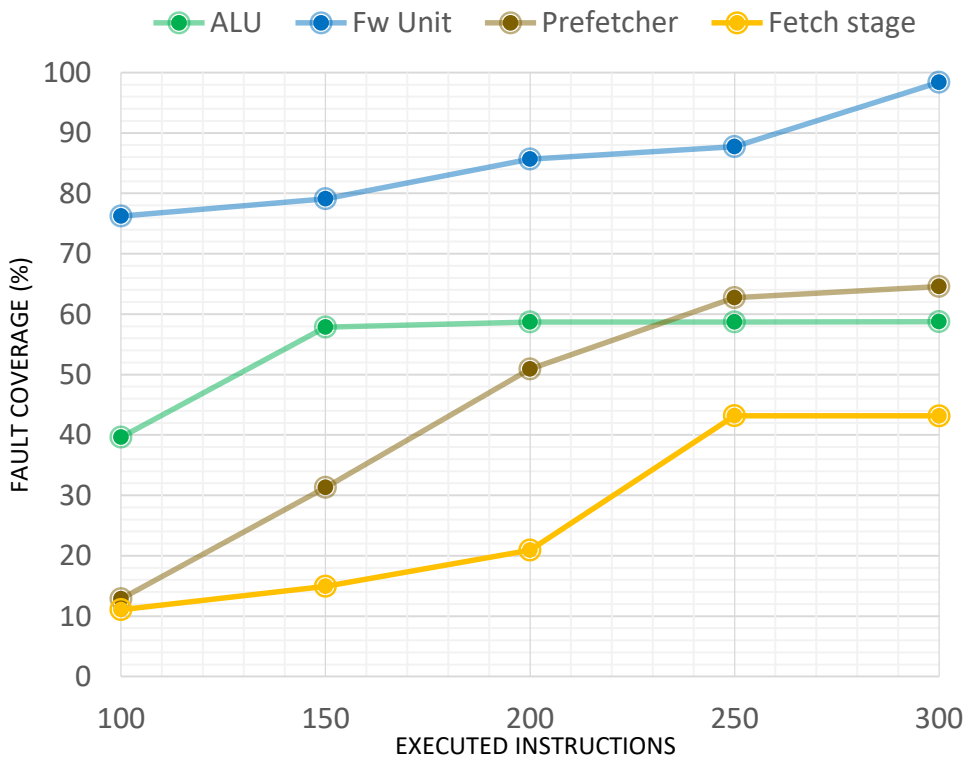
RV32IMC Exhaustive Fault Injection campaign



Component	Injected faults
ALU	2555
Fw Unit	871
Prefetcher	1873
Fetch stage	4351

Application and results(2)

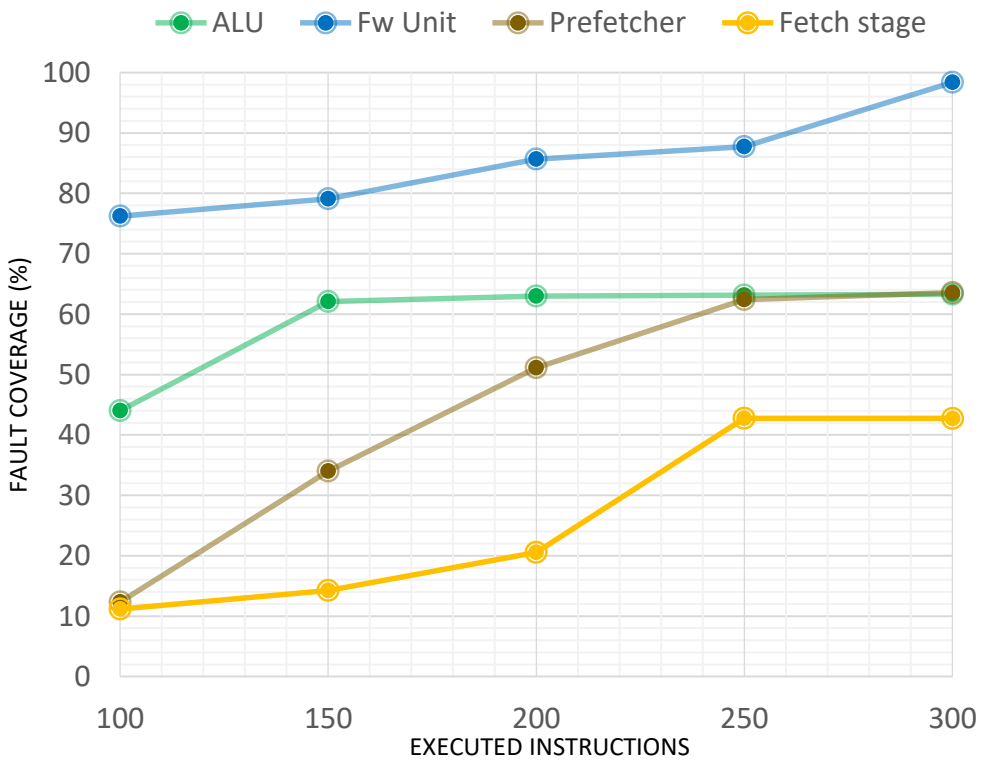
RV32IMC-Exceptions Exhaustive Fault Injection campaign



Component	Injected faults
ALU	2555
Fw Unit	871
Prefetcher	1873
Fetch stage	18936

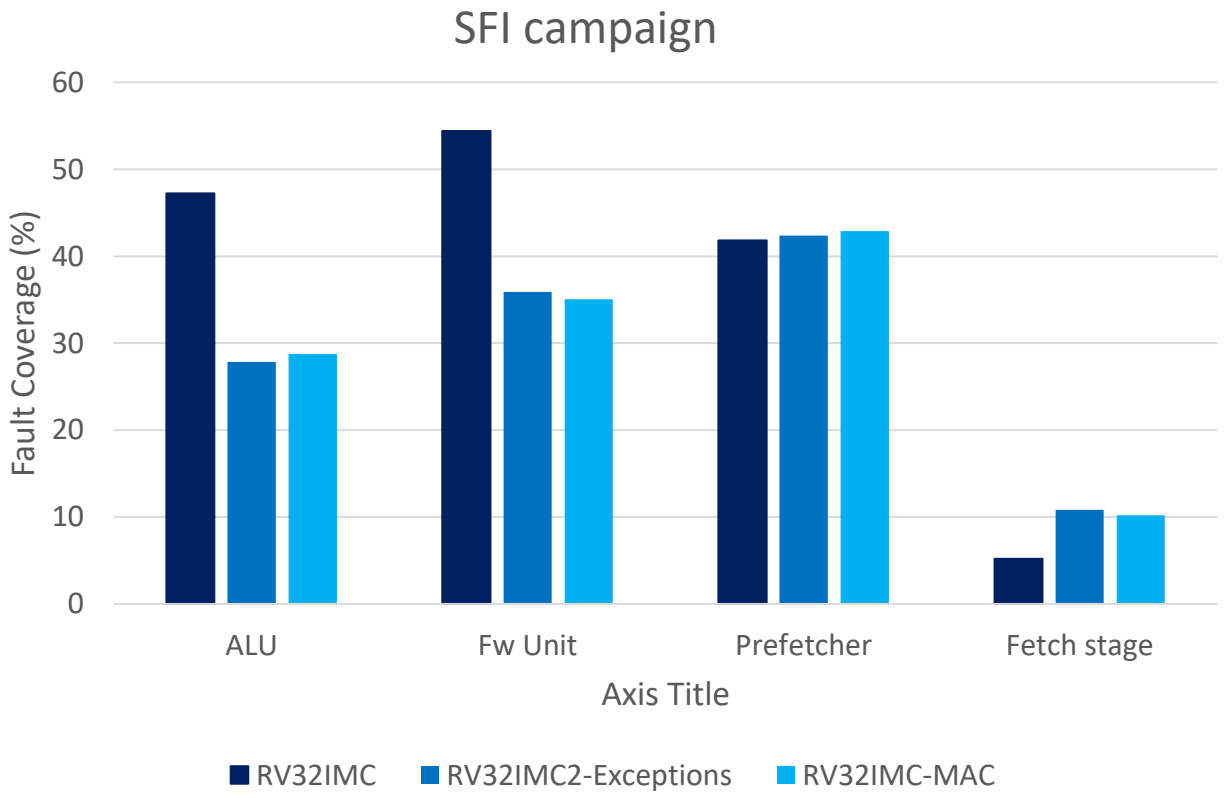
Application and results(3)

RV32IMC-MAC Exhaustive Fault Injection campaign



Component	Injected faults
ALU	2555
Fw Unit	871
Prefetcher	1873
Fetch stage	18936

Application and results(4)



- Statistical fault injection (SFI) was applied to the different CPU alternatives
- Different fault models injected:
  - Stuck-at
  - SEU
  - Timing faults
- 300 instructions as test length

Conclusion and future work

- Low-overhead automated safety evaluation of various RISC-V alternatives by combining Model-driven Fault Simulation and Model-driven Random Instruction Generation
- The EFI campaign resulted in an acceptable fault coverage range (40%-99%) for a length of input test of 300 instructions
- The total effort required to run all the campaigns was 1 person-day
- **Future work**: Extending the supported fault models, constraining jumps to in-bound memory locations, and further comparisons to other commercial and open-source fault simulation tools

Questions?

Thank you!