QEMU Based Fault Effect Analysis for RISC-V

Peer Adelt, Bastian Koppelmann, Wolfgang Müller, Christoph Scheytt

Heinz Nixdorf Institute – Paderborn, Germany
Hardware Simulation

- **HW**: Model of the HW under test
- **Stimuli**: Test vectors for the HW
- **Response**: Any externally observable HW reaction during simulation

➢ Response is compared with expected results after simulation to validate correct behaviour of the HW
**Mutant Killed:** Response differs from golden response

- test vector can detect the fault → useful test vector → keep it
**HW Fault Simulation**

**Test Vectors**

- Golden HW
- HW Mutant

**Response**

- ok
- ok

**Mutant Not Killed:** Same output as golden response

- test vector cannot detect the fault → useless test vector → discard
RISC-V QEMU Fault Simulation

- **Golden RISC-V CPU**
- **RISC-V CPU Mutant:**
  Copy of the Golden RISC-V CPU model an inject fault
- **Exit code:** different signature, exception, …
Our Fault Model: Bit Faults in Microprocessors
Faults in RISC-V CPU registers (GPR, CSR, Instructions) and Memory

Bit-level fault analysis for:

- **Instructions** (Opcode, Operands)
  - Pipeline-related faults
  - Instruction cache faults
  - Instruction memory faults

- **General Purpose Registers (GPR)**

- **Control and Status Registers (CSR)**
  - Core CSRs
  - MM CSRs (Device registers; memory-mapped)

- **Memory (incl. MMIO)**
Permanent Stuck-Ats-0/1 in RISC-V 32 Bit Microprocessors

Fault Models  Permanent faults: Stuck-at-1/0 and Bit-Flip

Instruction Faults (Opcode and Operand):
- 32 Bit Instructions
- #Permanent 1bit Faults: $32 \times \langle\#\text{Instruction Instances}\rangle$

GPR Faults:
- 31 Registers á 32 Bit
- #Permanent 1bit Faults: $32 \times \langle\#\text{GPRs}\rangle = 992$

CSR Faults (Core CSRs, MM CSRs):
- Register Count is implementation specific, Word Length: 32 Bit
- #Permanent 1bit Faults: $32 \times \langle\#\text{CSRs}\rangle$

Memory Faults (incl. MMIO):
- Memory divided into Bytes
- #Permanent 1bit Faults: $8 \times \langle\#\text{Memory Bytes}\rangle$

Based on static memory content initialization
Transient Bit-Flips in RISC-V 32 Bit Microprocessors

Fault Models  
Transient faults: Bit-Flip

Instruction Faults (Opcode and Operand):
- 32 Bit Instructions
- #Transient 1bit Faults: \(32 \times \#\text{Instruction Executions}\) → Loops, Jumps, Branches, etc.

GPR Faults:
- 31 Registers à 32 Bit
- #Transient 1bit Faults: \(32 \times \#\text{GPR Executions}\)

CSR Faults (Core CSRs, MM CSRs):
- Register Count is implementation specific, Word Length: 32 Bit
- #Transient 1bit Faults: \(32 \times \#\text{CSR Executions}\)

Memory Faults (incl. MMIO):
- Memory divided into Bytes
- #Transient 1bit Faults: \(8 \times \#\text{Memory Bytes} \times \#\text{Executions}\)

Issue: #Transient Faults explode, because each execution creates a separate mutant
Scalable Fault Effect Analysis for RISC-V with QEMU

Framework Overview

Phases of Fault Effect Analysis for RISC-V (FEAR-V):

1. Software Generation & Compilation
2. Golden Run w/ HW Execution Analysis
3. Mutant Generation with Fault Injection
4. Fault Simulation & Analysis & Optimization

Optimization:
increase coverage & decrease test vectors

Configurable: RV ISA subset, iterations, n-bit faults, transient/permanent, fault model, …
Test Software Generation
For different RISC-V ISA Subsets

1. **Software Generation & Compilation:**

   SW Generators start with basic parameters

   - Csmith SW Generator * (University of Utah)
   - Torture SW Generators * (University of California, Berkeley)

   SW Test Libraries

   - RISC-V – Tests (University of California, Berkeley)
   - RISC-V Architecture Tests * (RISC-V Foundation Architecture Test SIG)
   - UPB CSR Tests (Paderborn University)

   Compiler: RISC-V GCC.

* with signatures
ISA and Register Execution Analysis
For different RISC-V ISA Subsets

2. Golden Run w/ HW Execution Analysis:
   • Which and how many (%) Instructions are executed?
   • Which and how many (%) GPRs and CSRs are executed?
   • Additional properties/statistics:
     • #Instructions (types, executions)
     • Lines of code
     • Program Execution time
     • Memory Execution
     • …

Example:
• ISA: RV32GC
• 7 Csmith Programs

<table>
<thead>
<tr>
<th>Analysis/ SWProgram</th>
<th>P01</th>
<th>P02</th>
<th>P03</th>
<th>P04</th>
<th>P05</th>
<th>P06</th>
<th>P07</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr. Type (#)</td>
<td>59</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>59</td>
<td>62</td>
<td>58</td>
</tr>
<tr>
<td>Instr. Type (%)</td>
<td>38,1</td>
<td>38,1</td>
<td>37,4</td>
<td>36,8</td>
<td>38,1</td>
<td>40</td>
<td>37,4</td>
</tr>
<tr>
<td>GPR Cov (#)</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>27</td>
<td>27</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>GPR Cov (%)</td>
<td>96,8</td>
<td>96,8</td>
<td>96,8</td>
<td>87,1</td>
<td>87,1</td>
<td>96,8</td>
<td>87,1</td>
</tr>
<tr>
<td>CSR Cov (#)</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total LoC (#)</td>
<td>4071</td>
<td>3385</td>
<td>2455</td>
<td>2436</td>
<td>2428</td>
<td>3575</td>
<td>3735</td>
</tr>
</tbody>
</table>
Mutant Generation & Fault Injection
With automatic Mutant Reduction

3. Mutant Generation w/ Fault Injection:
   • Estimate fault simulation time
   • Inject permanent / transient n-bit faults:
     Instructions (Opcode & Operands), GPRs, CSRs, Memory (incl. MMIO)

Mutant reduction → reduce #simulation runs/time
   • Discard mutants for non-executed Opcodes & Registers
   • Discard mutants accessing invalid memory when memory protection (MMU/MPU) is available

![Diagram showing the process of Mutant Generation & Fault Injection]

- **Base (GPR)**: 0x000001000
- **Offset (Immediate)**: 0x0000003B
- **RAM**:
  - **valid**
  - **invalid**
Mutant Generation & Fault Injection
With automatic Mutant Reduction

Mutant Reduction

- Discard mutants for non-executed Opcodes & Registers
  - Example: Only 30 of 31 GPRs are executed, → 960 instead of 992 permanent GPR mutants

- Discard mutants executing invalid memory when memory protection is available
  - Example: SW has 1.943Mio GPR executions, from which 1.607M are Load/Store instructions for invalid memory.
  → Only 0.335Mio GPR mutants (~17%) need to be simulated.
4. Fault Simulation & Analysis:
   • Simulate the reduced set of mutants from previous step
   • exit code for each mutant

Exit codes:
   • **Timeout**: Endless loops, etc. \(\rightarrow\) Watchdog
   • **Killed**:
     • RISC-V Standard Exceptions (unaligned memory access, illegal instruction, misaligned instruction, access fault, load misaligned, store address misaligned, etc.)
     • Different Signature (Golden output \(!=\) mutant output)
   • **Not Killed**: no observable faulty behavior
     • to be discarded by setcover optimization
     \(\rightarrow\) Improve HW/SW safety measures
# Fault Effect Analysis for RISC-V (FEAR)

**Example: Analysis of 24 Test Programs from RISCV-Torture for Freedom E300**

### Golden Run/Programs

<table>
<thead>
<tr>
<th>Golden Run/Programs</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Executions (#)</td>
<td>344366</td>
<td>342190</td>
<td>343317</td>
<td>343433</td>
<td>343936</td>
<td>... 343565</td>
</tr>
<tr>
<td>GPR Cov (#)</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>... 31</td>
</tr>
<tr>
<td>GPR Cov (%)</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>... 100.00</td>
</tr>
<tr>
<td>GPR Exe (#)</td>
<td>1509760</td>
<td>1506573</td>
<td>1507902</td>
<td>1508039</td>
<td>1506600</td>
<td>... 1503688</td>
</tr>
<tr>
<td>CSR Cov (#)</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>... 9</td>
</tr>
<tr>
<td>Instr. Type Cov (#)</td>
<td>76</td>
<td>75</td>
<td>76</td>
<td>75</td>
<td>77</td>
<td>... 76</td>
</tr>
<tr>
<td>Instr. Type Cov (%)</td>
<td>84.4%</td>
<td>83.3%</td>
<td>84.4%</td>
<td>83.3%</td>
<td>85.5%</td>
<td>... 84.4%</td>
</tr>
<tr>
<td>Instr. Type Total (#)</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>... 90</td>
</tr>
<tr>
<td>Instr. LoC (#)</td>
<td>1526</td>
<td>1479</td>
<td>1493</td>
<td>1500</td>
<td>1480</td>
<td>... 1480</td>
</tr>
<tr>
<td>Instr. Executions (#)</td>
<td>807061</td>
<td>802448</td>
<td>804558</td>
<td>804508</td>
<td>804066</td>
<td>... 802807</td>
</tr>
<tr>
<td>Prg Execution Time (us)</td>
<td>8168</td>
<td>6399</td>
<td>6863</td>
<td>10375</td>
<td>7312</td>
<td>... 6577</td>
</tr>
</tbody>
</table>

### Exit Codes/Programs

<table>
<thead>
<tr>
<th>Exit Codes/Programs</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Kill. (#)</td>
<td>19676</td>
<td>18729</td>
<td>19574</td>
<td>19505</td>
<td>18168</td>
<td>... 19101</td>
</tr>
<tr>
<td>Timeout (#)</td>
<td>594</td>
<td>772</td>
<td>799</td>
<td>414</td>
<td>25694</td>
<td>... 1120</td>
</tr>
<tr>
<td>Killed (#)</td>
<td>25948</td>
<td>25421</td>
<td>24901</td>
<td>25995</td>
<td>26317</td>
<td>... 24877</td>
</tr>
<tr>
<td>Total (#)</td>
<td>46218</td>
<td>44922</td>
<td>45274</td>
<td>45914</td>
<td>45162</td>
<td>... 45098</td>
</tr>
</tbody>
</table>

**Total #Mutants:** 1,081,200

**Total #Instr. Executions:** 19,279,825

→ **Fast Fault Simulation:** 3776.54 MIPS (on 24 Threads *)

(Total Sim. Time: 229 seconds)

* AMD Ryzen Threadripper PRO 3945WX System
Summary & Next Step

Designed and implemented a mutation testing framework for Fault Effect Analysis for RISC-V (FEAR-V):

- Based on QEMU
- SW Library and Generation frontend
- Highly Configurable with YAML (ISA subsets, 1/2/..n-bit faults, bit-flip, SA-1/0, transient/permanent, …)
- Fast Fault Simulation Time (> 3000 MIPS)
  fast enough for up to 2-bit faults for RISC-V processors
  fast enough for transient faults
  fast enough to run multiple iterations for test vector optimization

Next Step:

- Chip Layout dependent fault generation
  Based on observation:
  1:1 relationship between QEMU bit variable and flipflop in chip layout
Thank you for your attention

Peer Adelt, Bastian Koppelmann, Wolfgang Müller, Christoph Scheytt

Heinz Nixdorf Institute – Paderborn, Germany
Backup: TCG with Fault Injection

QEMU Fault Injection