Cross-Level Verification of Hardware Peripherals

Sallar Ahmadi-Pour[1], Muhammed Hassan[1,2], Rolf Drechsler[1,2]
[1] Institute of Computer Science, Universität Bremen, Germany
[2] Cyber-Physical Systems, DFKI GmbH, Germany
{sallar, hassan, drechsler}@uni-bremen.de

Introduction

- Modern embedded systems rely on complex System on Chips (SoCs)
- Virtual Prototypes (VPs) are used in SoC development to handle complexity
- VPs enable early and parallelized Hardware (HW) / Software (SW) development and verification
- Verification methods for Central Processing Units (CPUs) with VPs available
- Verification of SoC peripherals is comparably neglected

Problem: Peripherals are different than CPUs

Solution: VP-based verification methods of hardware peripherals

State of the Art

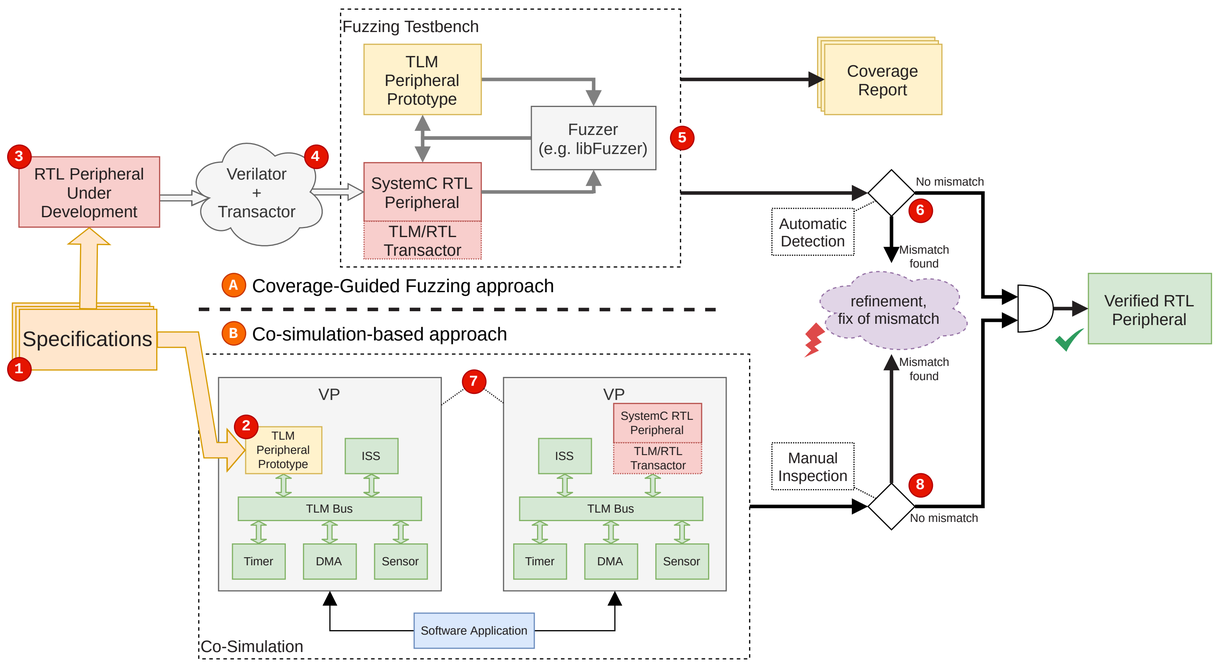
- Existing approaches are similar to constrained random methods, recent HW verification utilizes fuzzing
- Often faster than methods that rely on Register-Transfer Level (RTL) simulation
- Full system interaction (including interrupts) is usually not considered
- Current VP-based methods mostly look into CPU verification

Challenge: Prior work lacks full system consideration

Approach: VP makes full system interaction available

Verification Methodology

- Utilize available Transaction Level Modeling (TLM) based peripherals
- Employ state-of-the-art fuzzer for powerful Coverage-Guided Fuzzing (CFG)
- Benefit from VP-based full platform simulation



Detailed overview of our verification methodology for hardware peripherals

Two approaches acting in synergy
Ⓐ CFG on the Unit-Level
Ⓑ Application-driven Co-Simulation on the System-Level

❶ Specifications that describe behavior
❷ TLM implementation inside VP, Golden Reference
❸ RTL Peripheral, the Design under Verification (DUV)
❹ Translation of RTL from HDL to C++ with transactor
❺ Execution of Fuzzing testbench in differential setup
❻ Automatic detection of mismatches allows refining and retesting
❼ Utilize VP to execute applications on system with golden model and RTL DUV
❽ Detection of mismatch allows refinement

Evaluation

- Evaluate RISC-V Platform Level Interrupt Controller (PLIC) as peripheral with broad functionality:
  - Handle bus transactions
  - Timing-specific behavior (e.g., maximum delay to inform the CPU about interrupt)
  - Various I/O and internal configuration registers
- Utilize open-source RISC-V VP for platform simulation
- Enable CFG through LLVM libFuzzer

Coverage Guided Fuzzing

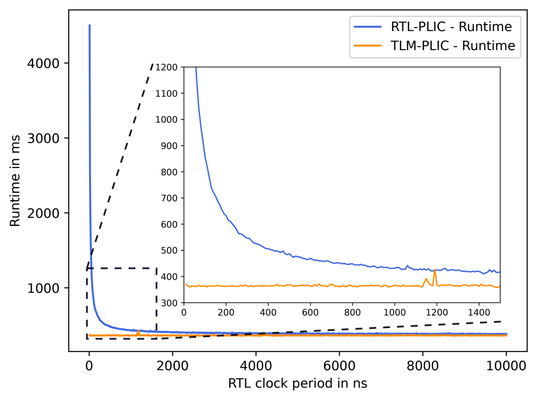
- Executed CFG for 1 hour
- Generated 39 445 input sequences in total
- Detected and fixed three bugs
- Achieve high coverage through fuzzing

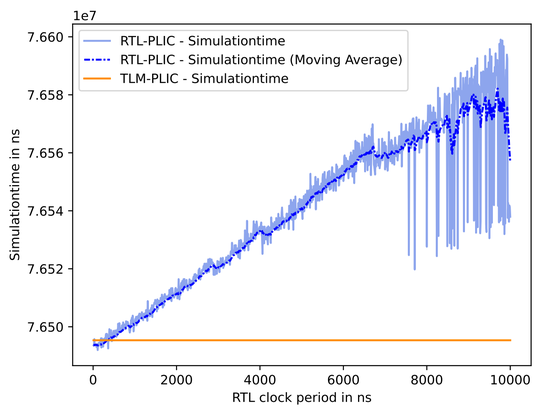
Coverage Metric	TLM PLIC			RTL PLIC		
	Hit	Available	Coverage	Hit	Available	Coverage
Line coverage	119	121	98.3%	3212	3721	86.3%
Function coverage	13	13	100%	20	24	83.3%
Branch coverage	72	118	61.0%	1056	1432	73.7%

Obtained coverage metrics after 1 h of Coverage-Guided Fuzzing

Application-driven Co-Simulation

- Compare functionality of peripherals across full system VP simulations
- Use FreeRTOS-based applications, actively utilizing the peripheral
- Identify performance cost of co-simulating RTL and TLM inside VP
- Trace and compare timestamps for various RTL clock speeds
- Identify non-functional mismatches quickly





Effect of RTL clock period of DUV on 1) VP performance (left), 2) simulation accuracy(right)

Timestamp / µs			Actor	Event
TLM	RTL (10 ns)	RTL (100 000 ns)		
74	74	80	Task 1	Context switch on CPU 0 to Task 1
165	165	175	WFR	Context switch on CPU 0 to WFR
165	165	175	WFR	xSemaphoreGiveFromISR(0x800545DC)
165	165	175	WFR	Actor Ready: Task 2
165	165	177	Task 2	Context switch on CPU 0 to Task 2
165	165	177	Task 2	xSemaphoreTake(0x800545DC, 100)
167	167	179	Task 2	xSemaphoreTake(0x800545DC, 100) blocks
167	167	-	Task 1	Context switch on CPU 0 to Task 1
173	173	179	Task 1	Actor Ready: TzCtrl
173	173	180	Task 1	Context switch on CPU 0 to Task 1
265	265	275	WFR	Context switch on CPU 0 to WFR

Excerpt of FreeRTOS trace highlighting non-functional mismatch for different clock periods

Future Work

- Compare different state-of-the-art fuzzers (e.g., libFuzzer, AFL/AFL++) to highlight differences in input generation and coverage
- Fill coverage gap through additional software-based verification methods (e.g. symbolic execution)
- Explore more peripherals to study scalability and efficiency

Funded by:
Bundesministerium für Bildung und Forschung
Grant number 16ME0127 and 01IW22002
Scale4Edge ECXL

Find more about our RISC-V related work at agra.informatik.uni-bremen.de/projects/risc-v/ or scan the QR code

RISC-V® related research by AGRA

Universität Bremen

dfki ai Deutsches Forschungszentrum für Künstliche Intelligenz *German Research Center for Artificial Intelligence*