Symbolic Execution for RISC-V Embedded Software Using SystemC Peripheral Models

Sören Tempel\textsuperscript{1}, Vladimir Herdt\textsuperscript{1,2}, Rolf Drechsler\textsuperscript{1,2}

tempel@uni-bremen.de

\textsuperscript{1}Group of Computer Architecture, University of Bremen, Germany
\textsuperscript{2}Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
Motivation

**Goal:** Employing symbolic execution for testing embedded SW

**Challenges:**
- Heterogeneous software ecosystem
- Utilization of different peripherals
- Interaction with low-level machine details

**Prior Work:** Firmware rehosting, hybrid emulation, ...

⇒ Alternative: Employ accurate peripheral models
Hardware can be modeled on different abstraction levels
- Examples: Register transfer level, transaction level, ...
- Trade-off between accuracy and simulation performance
- Desired for symbolic execution: High simulation speed
- We utilize SystemC TLM for hardware modelling
  - Modelling language based on C++
  - Operates primarily on the transaction level
  - Models peripherals based on a bus abstraction

Figure: Gajski-Kuhn Chart
Approach: Virtual Prototyping

**Virtual Prototypes:** Executable SW model of a HW platform
⇒ Including provided peripherals in SystemC TLM

▶ Our work is based on the open source riscv-vp
▶ Support different RISC-V based hardware platforms
▶ Provides an executable model for the SiFive HiFive1

**GitHub:** https://github.com/agra-uni-bremen/riscv-vp
**SymEx-VP**: Symbolic execution meets VPs

- Integrates existing riscv-vp with KLEE
- Symbolically executes RISC-V machine code
- Provides TLM extension for symbolic values

⇒ SW is explored based on peripheral inputs

---

**Figure**: Architectural overview of SymEx-VP
Connection with KLEE

**Focus:** Addressing integration challenges in the embedded domain
⇒ Uses stripped-down version of KLEE without LLVM-specific code

- Presently offers a simple *Concolic Testing* implementation
- Offline executor implementing *Dynamic Symbolic Execution*
  - SystemC simulation is restarted for each new input
  - *Address Concretization* is used as a memory model
- Primarily uses KLEE’s solver interface and SMT abstractions

**GitHub:** https://github.com/agra-uni-bremen/clover
Error Detection

Problem: Low abstraction level complicates error detection

- Majority of embedded software is written in C/C++
- Limited protections against C pitfalls (memory safety, ...)
- Goal: Finding spatial violations in embedded C software
- Leverage prior work on HardBound by Devietti et al.

⇒ SW instrumentation to propagate pointer bounds

```
static char buf[BUFFER_SIZE];
int add_to_buffer(char c) {
    static size_t index = 0;
    if (index >= BUFFER_SIZE)
        return -1;
    // --- [[ Original Code ]] ---
    char *ptr = &buf[0];
    *(ptr + index) = c;
    // --- [[ Instrumented ]] ---
    char *ptr = &buf[0];
    setbound(&ptr, ptr, sizeof(buf));
    *(ptr + index) = c;
    // ------- END --------------
    index++;
    return 0;
}
```

Figure: HardBound compiler pass
Experiments with IoT Operating Systems

**Evaluation**: Performed several tests with RIOT, Zephyr, Apache NuttX, ...

- Tested different modules of the network stack
- Found 13 previously unknown bugs in RIOT
- Tested modules: DHCP, DNS, RPL, URI parsers, ...
Conclusion

Key Insights:
▶ Accurate peripheral models reduce integration effort for testing SW
▶ Modularity of RISC-V eases integration with symbolic execution

Source Code: https://github.com/agra-uni-bremen/symex-vp
⇒ Used as the basis of various existing scientific publications


¹https://doi.org/10.1016/j.sysarc.2022.102456