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

Sören Tempel[1], Vladimir Herdt[1,2], Rolf Drechsler[1,2]

tempel@uni-bremen.de

[1]Group of Computer Architecture, University of Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany

SPONSORED BY THE

Federal Ministry of Education and Research

Scale4Edge
16ME0127

VerSys
01IW19001

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, ...

$\Rightarrow$ Alternative: Employ accurate peripheral models

Background: Peripheral Modelling

- 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

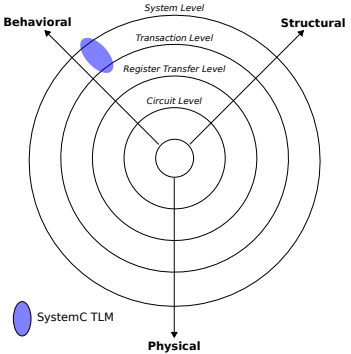


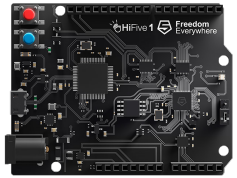
Figure: Gajski-Kuhn Chart

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 for Embedded Software

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

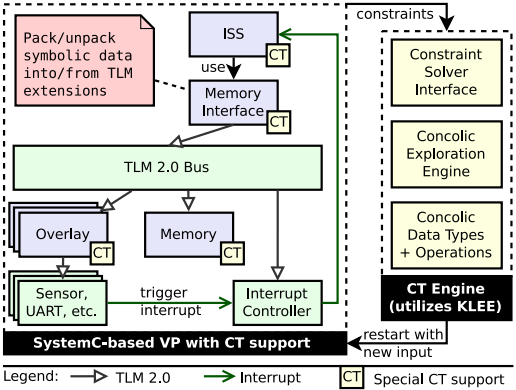


Figure: Architectural overview of SymEx-VP

Connection with KLEE

Focus: Addressing integration challenges in the embedded domain
$\Rightarrow$ 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.

$\Rightarrow$ SW instrumentation to propagate pointer bounds

```
1  static char buf[BUFFER_SIZE];
2
3  int add_to_buffer(char c) {
4    static size_t index = 0;
5    if (index >= BUFFER_SIZE)
6      return -1;
7
8    // --- [[ Original Code ]] --
9    char *ptr = &buf[0];
10   *(ptr + index) = c;
11   // --- [[ Instrumented ]] ---
12   char *ptr = &buf[0];
13   setbound(&ptr, ptr, sizeof(buf));
14   *(ptr + index) = c;
15   // ------- END --------------
16
17   index++;
18   return 0;
19 }
```

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`

$\Rightarrow$ Used as the basis of various existing scientific publications

More Information: Sören Tempel, Vladimir Herdt, and Rolf Drechsler. *SymEx-VP: An Open Source Virtual Prototype for OS-agnostic Concolic Testing of IoT Firmware*. Journal of Systems Architecture (JSA), 2022[1].

[1] `https://doi.org/10.1016/j.sysarc.2022.102456`