Hard Real-Time is Easy!

Pawel Dzialo[12] Per Lindgren[12]

[1] Tampere University, Tampere, Finland

[2] Luleå University of Technology, Luleå, Sweden

Abstract

Ensuring the correctness and schedulability of real-time applications is key to implementing safe and robust real-time systems. However, formal verification implies additional effort and the need for specialized competence.

In this work we leverage on the Symex symbolic execution engine, and port the Symex General Assembly Language to support the RV32I instruction set to obtain cycle-accurate WCET estimates for the Hippomenes architecture.

We outline how this information can be utilized to determine specific task response time and overall schedulability of hard real-time systems expressed in the Stack-Resource Policy(SRP) based Rust RTIC framework.

Hard real-time using worst case **E**xecution time estim**A**tion by **SY**mbolic execution is EASY!

1 Background

For hard real-time systems, any unmet execution deadlines are unacceptable. Guaranteeing that performance requirements are met is a problem with many moving parts:

- threaded models with unbounded critical sections
- hardware wait states
- cache performance
- general difficulty of formally verifying code

In this work we attempt to solve these problems by simplification:

- The Stack Resource Policy[1] model-based RTIC framework[4] provides an outset for trivial schedulability analysis given worst-case execution time(WCET) of each task and critical section.
- The Hippomenes architecture bounds the WCET of any instruction to one cycle.
- The Symex[2][3] symbolic execution engine combined with Hippomenes provides cycle-accurate bounds to WCET for RTIC tasks/critical sections automatically.

Since Symex works directly on the compiled binary, the WCET analysis is toolchain agnostic, and does not assume toolchain correctness.

2 Methodology

Using the simple function displayed in Figure 1 as an outset, Symex performs a depth-first search over all of the possible execution paths, determining their path conditions. The results of this search are displayed in Figure 2

```
fn simple(t: u32) -> u32 {
    if t == 1 {
        return 2;
    } else if t == 2 {
        return 4;
    } else if t == 3 {
        panic!()
    } else {
        if t == 1 {
            return 13
        }
        return 42;
    }
    return 13;
}
```

Figure 1: The simple function under analysis

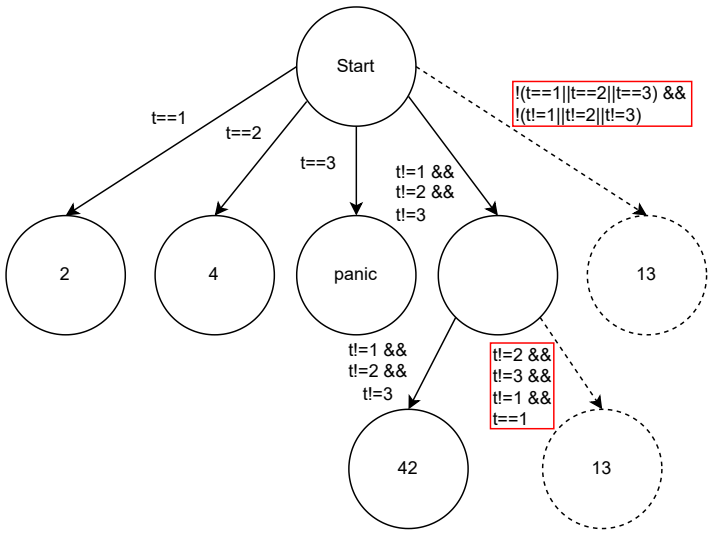

Figure 2: Symex search tree according to Figure 1. Generated path conditions are shown on edges. Unfeasible path conditions are highlighted in red.

The obtained execution time of each path is compared against actual execution time on Hippomenes, confirming the cycle-accurate WCET bounds.

Path ID	Return Value	Symex estimate (cycles)	Actual execution time (cycles)
1	2	4	4
2	4	6	6
3	panic	8	8
4	42	8	8

Figure 3: Comparison of Symex analysis against actual runtime of the code in Figure 1. Notice that paths leading to `return 13;` are proven unfeasible thus not taken into account. Although Symex provides execution time estimates for all feasible paths, for WCET, only the worst case (in this case Path ID 3 and 4) are of interest.

3 RTIC-based analysis

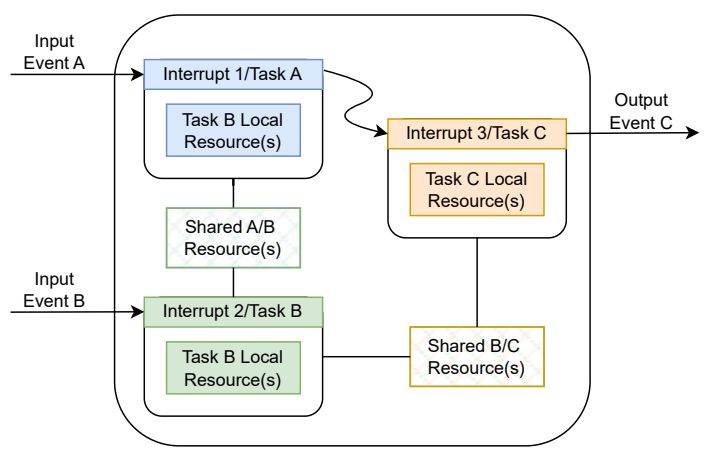

Figure 4: Based on the SRP Task/Resource model, the Rust RTIC framework generates a correct by construction binary with guaranteed memory safety and race-/dead-lock free execution.

The SRP Task/Resource model is amenable to static scheduling analysis. We obtain hard real-time guarantees by taking the Easy approach to determine WCETs for tasks and critical sections.

References

T. P. Baker. “A stack-based resource allocation policy for realtime processes”. In: *[1990] Proceedings 11th Real-Time Systems Symposium* (1990), pp. 191–200.

J. Norlén. “Architecture for a Symbolic Execution Environment”. MA thesis. Luleå University of Technology, Computer Science, 2022.

E. Serrander and J. Norlén. *Symex*. https://github.com/s7rul/symex. 2024.

R. Team. *The RTIC Book*. https://github.com/rtic-rs/. 2024.