State Of Tock

TockWorld 2022

July 19th-20th, 2022

Outline

A Brief History of Tock

Project Mission

Introductions

The state of Tock since the previous TockWorld

Progress towards reducing code size

Progress towards building Tock with stable Rust

Agenda

Focus of this TockWorld

A Brief History of Tock

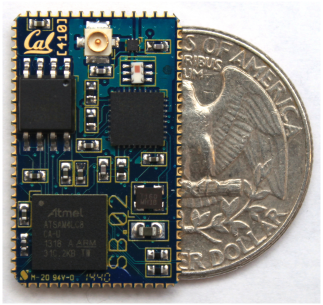
6 nerds walk into a mailing list. . .

```
From: Philip Levis
Subject: [helena-project] SenSys poster/demo
To: helena-project@lists.stanford.edu
Date: Wed, 09 Jul 2014 13:15:12 -0700
```

> *Each of us has made some interesting progress on a next generation embedded platform. Michael @Berkeley has storm (an M4 + RF233), Tom and Amit @Stanford have a TinyOS nRF8001 stack and some interesting measurements for BLE, Pat and Brad @Michigan have a few Cortex M platforms, with the CC2420.*
>
> *Operating system: what should an operating system for such a device look like? Can we achieve something like the efficiency and dependability of TinyOS without being so difficult to extend and program?*

Before Tock, there was `storm.rs` (October 2014)

Storm is a 16mm x 26mm (1/2“ by 1”) solder on module that combines a 32 bit Cortex M4 microcontroller with an 802.15.4 radio, and 64 Mbit of flash memory. This ultra low power module exports 80 pins via castellated edges, and is designed to be soldered into a larger PCB that contains additional sensors or actuators.



Reboot with First Tock commit

```
commit a14379b850bf47e89cd2945226cbf9bcbab5f43f
Author: Amit Aryeh Levy <amit@amitlevy.com>
Date:   Tue May 19 15:29:44 2015

    Initial commit

    Barebones build system and boot to Rust on Storm
```

Milestones since

- 2016: Dynamic userland code loading
- 2017: Tock training at RustConf, first deployment (Signpost)
- 2018: 1.0 release

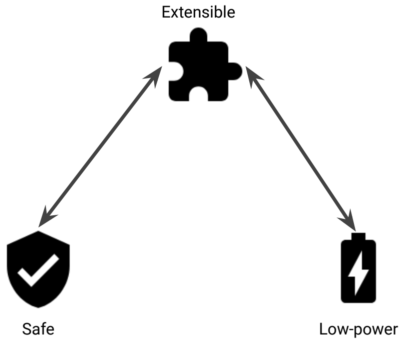
Project Mission

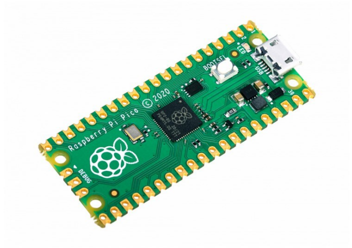
We believe in

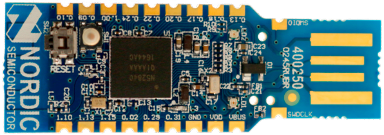
Building embedded systems that

- are safe
- people can program
- not resource intensive

A modest desire, for modest devices, with big implications.




Raspberry Pi Pico
© 2020
BOOTSEL
LED
USB
DEBUG


NORDIC
SEMICONDUCTOR
RESET
SWDIO
SWDCLK
1.10
1.13
1.15
0.02
0.29
0.31
GND
VDD
VBUS

We believe that the best way to accomplish this includes

- Rust
  - Type safety
  - Careful and pragmatic use of formal tools
- Co-development with
  - Hardware
  - Applications
- Open source collaboration between
  - Practitioners
  - Researchers
  - Educators

Introductions

The state of Tock since the previous TockWorld

The Project Today

TockWorld 4.0 (November 2019) -> Today (July 2022)

- Contributors: 226(ish) overall
  - tock: 76 -> 196
  - libtock-c: 28 -> 56
  - libtock-rs: 18-43
  - tockloader: 22
- Over 10,000 commits, 2,497 closed PRs (so close to 2,500!)
  - over 5000 since TockWorld 4.0!
- Releases since TockWorld 4.0
  - 2.0 in August 2021
  - 1.6 in October 2020
  - 1.5 in April 2020

Major Milestones

- Tock 2.0
- Security Model

Community

- Core WG: added Johnathan Van Why in March 2020, Leon Schuermann in April 2021
- Mailing list: 192 subscribed (pretty inactive)
- Slack: 372 members, more active, pasta-monster save us

Progress towards reducing code size

Background

- Tock binaries seem to be somewhat (~50-100%) larger than comparable projects using C-based RTOS's
- Idiomatic use of Rust contributes to this issue
- I worked on this at Google last summer, using Ti50 as an artifact for evaluation
- `https://github.com/tock/tock/blob/master/doc/CodeSize.md` details several mechanisms for writing Rust to avoid large constructions

Summary

- >20 PRs merged into upstream Tock kernel + libtock-rs towards reducing size
- These + other Ti50 specific changes generated ~19% savings in Ti50 (76 kB from a 400 kB binary)
- Applying similar changes to the upstream nrf52dk kernel binary reduced its size by 26.5% (23 kB)
- Most significant benefits:
  1. Configuration options to remove uncalled panic / debug code, necessary because Rust could not remove uncalled virtual methods.
  2. Reducing monomorphization in grant code
  3. Removing individual panics by using non-panicking methods
  4. RISC-V linker relaxation (RISC-V only): ~6% size reduction
- Published a paper at LCTES 2022 talking about this: `https://dl.acm.org/doi/abs/10.1145/3519941.3535075`

Upstream Response

- Significant reductions of the size overhead of `#[derive(Debug)]` relative to handwritten alternatives: `https://github.com/rust-lang/rust/pull/98190`
- Implementation of `-Z virtual-function-elimination` option for `rustc`, which enables automatic removal of uncalled virtual functions without relying on compile-time feature flags.
  - This provides 6 kB of savings for the Tock Imix binary today, and should allow us to remove `debug_panics` Config option from the kernel.
- Ongoing efforts to add non-panicking alternatives to functions in `core`.
- Renewed efforts towards RISC-V linker relaxation support in `lld`: LLVM-15 will have significant improvements `https://maskray.me/blog/2022-07-10-riscv-linker-relaxation-in-lld`

Future Work to improve code size

- Continue removing panics in the kernel/capsules
- Once it is better tested, enable `-Z virtual-function-elimination` by default, and remove redundant kernel config options
- Packed system calls: At a small cost in size to the kernel, can enable smaller applications.
- Investigate the use of alternative formatting libraries (ufmt or defmt) within the kernel based on experiences in libtock-rs
- Improved visibility of per-PR size tracking

Threats to code size

- Continued tradeoff between kernel features, kernel code maintainability, and size.
  - example: `load_and_check_processes` adds a valuable feature, but costs size even for boards that do not use it. Removing this overhead would require maintaining two different process loading mechanisms, one synchronous and one asynchronous, and making sure their behavior stays consistent across future updates.

Progress towards building Tock with stable Rust

Background

- Tock has only compiled using nightly Rust since its inception
- Given the greater dependability of the stable compiler, it has been a goal of Tock to build using stable Rust since 2020

Remaining features used in `kernel/` / `chips/` / `arch/` / `capsules/` /

`libraries/`

- Features with outstanding PRs for their removal:
  - `#[feature(core_intrinsics)]`
  - `#[feature(const_mut_refs)]`
- `#[feature(asm_const]`: Could be removed by going back to a macro-based approach for defining RISC-V CSRs, instead of relying on const generics.
- `#[feature(naked_functions)]`: Approved for stabilization upstream
- `#[feature(asm_sym)]`: Nominated for stabilization upstream, waiting on 2 more approvals

Remaining features used in `boards/` / nightly-only compiler flags

- `#[feature(custom_test_frameworks)]`: Very useful for enabling software-only kernel tests using QEMU
- `-Z build-std`: Important for code size
- `-Z virtual-function-elimination`: Important for code size
- `-Z emit-stack-sizes`: Useful for reducing RAM use
- rustdoc: `--document-hidden-items` + `-D warnings`

Proposed path forward

1. Merge outstanding PRs
2. Wait for `#[feature(naked_functions)]` and `#[feature(asm_sym)]` to stabilize upstream
3. Assuming `#[asm_const]` remains not on track for near-term stabilization, use macros for RISC-V CSRs (`https://github.com/tock/tock/pull/2470`)
4. Wait until a stable Rust version has been released containing `#[feature(naked_functions)]` and `#[feature(asm_sym)]` (xx weeks after they are stabilized in nightly).
5. Continue to use a nightly Rust version in `rust-toolchain`, but add a CI check that verifies all boards compile using a stable compiler.

Why keep using nightly upstream?

- Significant size savings for users of upstream board definitions (11.1 kB on Imix)
- Simplest approach to continued use of `#[feature(custom_test_frameworks)]` for CI
- Still enables out-of-tree boards to use a stable compiler, or to use a "stable compiler" with the `RUSTC_BOOTSTRAP` flag toggled to allow nightly features anyway.

Alternative

- Use stable compiler in rust-toolchain, but preserve `#[feature(custom_test_frameworks)]` using `#[cfg]` approach that removes the feature unless a nightly compiler is being used, and overwrites rust-toolchain only when compiling tests.

Agenda

Today

9:30	State of Tock
10:45	Break
11:00	OpenTitan
11:30	Teaching Tock
12:00	Lunch
13:00	Execution Bounds
13:30	Ti50
14:00	Repurposable Devices
14:30	Break
15:00	Development Focus Areas
16:00	Break
16:30	Community Development

Tomorrow

8:00	Breakfast
9:00	Who is Tock for?
11:00	Project Governance
12:00	Lunch

Focus of this TockWorld

Common development environments

- Strong shift towards RISC-V
  - Userland not as well supported
  - Commercial hardware platform availability
- Varried use cases
  - Secure elements
  - Sensor networks
  - Experimental OS / networking

Long-term community viability/stability

- Governance
- Outreach

Supporting research, education, *and* production

- Big kid concerns, such as:
  - copyright
  - security auditing
  - API stability
- Engagement in non-academic settings
- Fairness and efficacy in decision making