

Samsung Electronics

Bridging the C/Rust Divide: Replicating the PWM Subsystem's Success

Michał Wilczyński

07.10.2026

1. The bottleneck, and the ask

The bottleneck

- Independent developers write new Rust bindings - and then stall at the subsystem gate
- The reviewer evaluates a Rust design through C paradigms: callbacks, structs, vtables, ownership by convention
- A maintainer who cannot review the Rust side has to take it on trust - so they reasonably don't
- And the hard part is not the merge. It is month two, when the abstraction needs a fix and nobody owns it

Two failure modes: the series never lands, or it lands and then rots.

What I want out of this session

7

other people have commits in
the abstractions

1

driver built on them - mine

0

requests to write a second one

- The abstractions merged in 6.19, and the code is alive - it gets fixed, refactored and carried along by treewide changes
- What has not happened is adoption: a year on, nobody has written a second PWM driver in Rust
- Bindings with a single user look like a science project, however good they are

The rest of this talk is the evidence. The question I need help with is why the second driver has not appeared.

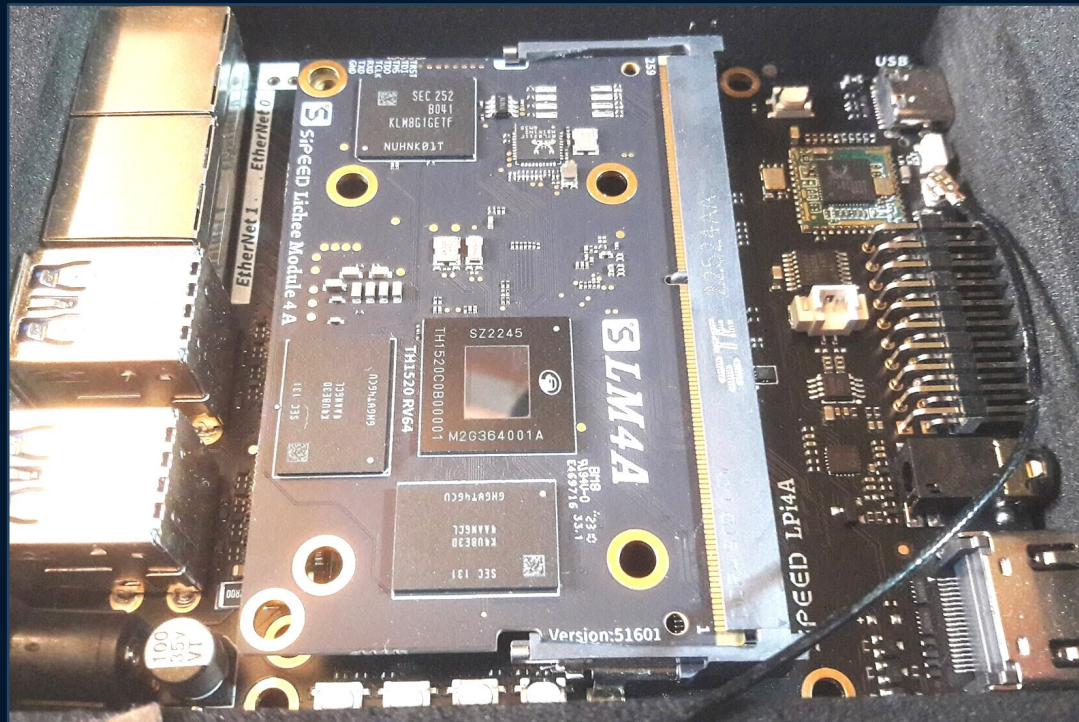
1. The bottleneck, and the ask

What happened with PWM

- rust/kernel/pwm.rs - safe abstractions for the PWM subsystem
- A real driver on real hardware: TH1520 PWM on the Lichee Pi 4A, driving a temperature-controlled fan
- Developed in public, with the C maintainer engaged throughout
- Merged for 6.19 - abstractions and driver together

1. The bottleneck, and the ask

The hardware, and the actual goal



Sipeed Lichee Pi 4A - T-HEAD TH1520

the LM4A module carries the SoC; the PWM block drives the fan header

What the driver does

six PWM channels, one of them wired to the CPU fan

Why it matters

the fan is under thermal control, so the SoC throttles without it

Why it is a good demo

a real driver on real hardware, not a synthetic example

2. What we did, and what it cost

Design in the maintainer's terms

- Do not ask a C maintainer to learn Rust to review your patch - map your design onto what they already own
- PWM's answer: build on the waveform API, the subsystem's own current best practice

A C pwm_ops vtable becomes a PwmOps trait - same four callbacks, same contract

- The reviewer checks the semantics they know; the language is an implementation detail

```
/* C: a vtable of function pointers */
struct pwm_ops {
    int (*round_waveform_tohw)(...);
    int (*round_waveform_fromhw)(...);
    int (*read_waveform)(...);
    int (*write_waveform)(...);
};
```

```
// Rust: the same contract as a trait
impl pwm::PwmOps for Th1520PwmDriverData {
    fn round_waveform_tohw(...) -> Result<..>
    fn round_waveform_fromhw(...) -> Result
    fn read_waveform(...) -> Result<..>
    fn write_waveform(...) -> Result
}
```

2. What we did, and what it cost

One of several reworks: a single allocation

v1 kept the Rust data in its own allocation with a pointer to it. Danilo Krummrich pointed at rust/kernel/drm/device.rs, where DRM puts the C struct and the driver data in one allocation.

v1 - two objects, two lifetimes

struct pwm_chip (C)

pointer

Rust driver data (Box)

v16 - one object, one lifetime

struct pwm_chip (C)

Th1520PwmDriverData (Rust)
constructed in place, pin_init

one pwmchip_alloc(dev, npwm, sizeof:<T>())

```
let sizeof_priv = core::mem::size_of::<T>();
let c_chip = bindings::pwmchip_alloc(parent.as_raw(), num_channels, sizeof_priv);
let drvdata = bindings::pwmchip_get_drvdata(c_chip); // the private area
pin_init::raw_try_init(drvdata.cast(), data)?; // build T in place
```

“This is exactly what we’re doing in DRM as well.” - Danilo Krummrich, v10 review. Subclassing is the shape a Rust binding should have; review is how I found that out.

2. What we did, and what it cost

Hide the unsafe, not the complexity

- The driver is 100% safe Rust; every unsafe block sits in the abstraction layer
- `Chip::new` returns an `UnregisteredChip`, and `register()` consumes it, so it cannot run twice
- `Chip::new` replaces four manual C steps with one call
- `PinnedDrop` disables the clock on unbind, automatically

```
// the whole of registration, in a driver that cannot leak it
let clk = Clk::get(dev, None)?;
clk.prepare_enable()?;

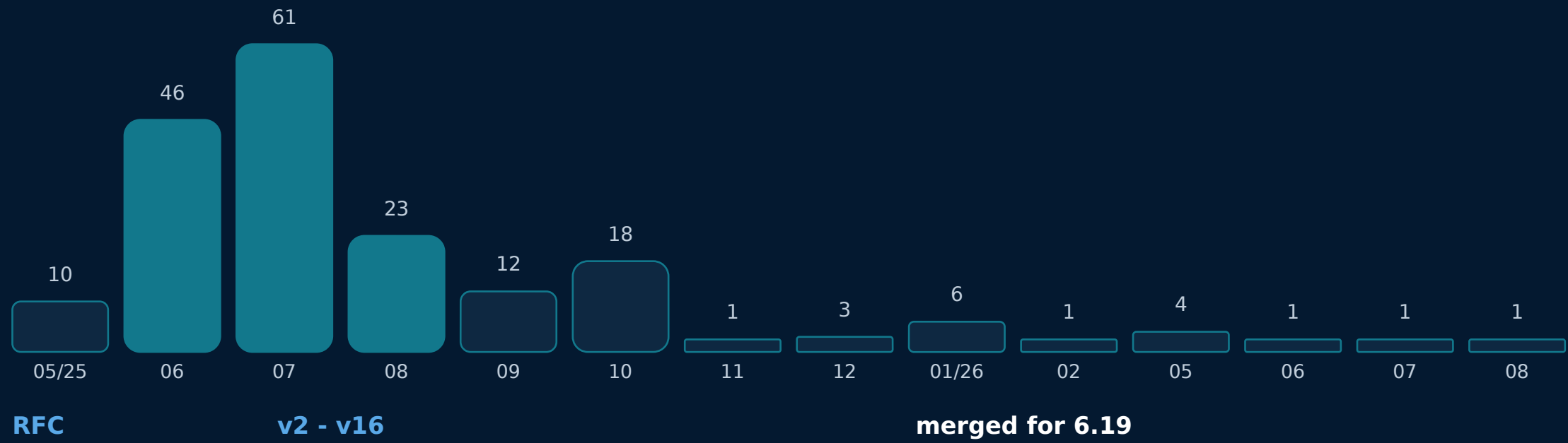
let chip = pwm::Chip::new(dev, TH1520_MAX_PWM_NUM,
    try_pin_init!(Th1520PwmDriverData {
        iomem <- request.iomap_sized::<TH1520_PWM_REG_SIZE>()?.into_devres(),
        clk    <- clk,
    })),
)?;

chip.register()?; // consumes the UnregisteredChip: cannot register twice
```

2. What we did, and what it cost

Fifteen months, in public

Every mail I posted with "pwm" in the subject, May 2025 to August 2026 - 188 in all.



Most of it is review. The small bars on the right are maintenance, still going.

3. After the merge

After the merge: the maintenance work starts

- Landing the bindings starts the maintenance work; it does not end it
- The driver shipped with two unsafe impls, Send and Sync - needed only because kernel Clk lacked those traits at the time
- Clk was fixed upstream, and Alice Ryhl deleted that block for me. It landed in 7.1, so the driver is now 100% safe

What the PWM maintainer did

IRC, directly

abstraction and driver fixes
coordinated in real time

patchwork.ozlabs.org

an account, set up for me by
Uwe Kleine-König, PWM
maintainer

CC'd on all Rust patches

every PWM Rust patch
reaches me before it lands

3. After the merge

Who has touched it since

13 people besides me have commits in `rust/kernel/pwm.rs` or the TH1520 driver; 19 have posted patches

Markus Probst

found a soundness hole: `register()` could be called twice on one chip. Added `UnregisteredChip`, which `register()` consumes - 7.0

Kari Argillander

memory leak on the init error path -> 6.19 stable, CVE-2026-45926

Maurice Hieronymus

narrowed `cycles_to_ns()` to `u32` so `div_ceil()` works; dropped a TODO - 7.2

Alice Ryhl

made `Clk Send+Sync` and removed the driver's two unsafe impls - 7.1

**Gary Guo, Francis Laniel,
Tamir Duberstein, Shankari Anand,
Guru Das Srinagesh**

`pin_init`, treewide renames, import style - the file is now part of the Rust kernel's ordinary maintenance surface

3. After the merge

So why has nobody written the second one?

- The abstraction is reviewed, maintained, and demonstrably fixable by someone who has never seen the hardware
- It is not a technical gap - a second driver is a few hundred lines of safe Rust on top of what is already merged

Three hypotheses - tell me which one is right, or that they are all wrong

Discovery

nothing tells a driver author that rust/kernel/pwm.rs exists

Incentive

C is still the default when someone writes a new driver

Confidence

one user is not yet evidence that the API has settled

This is a marketing problem, not a technical one. I cannot fix it by writing more code.

Discussion

1. Where should a driver author look to find out which subsystems already have Rust abstractions?
2. Uwe gave me a patchwork account and CCs me on every PWM Rust patch. Should that be the norm, and does it scale?
3. Who is on the hook when an abstraction breaks and the original author has moved on?

Backup - mapping C patterns onto Rust

Which C patterns map cleanly onto Rust traits, and which ones fight back?

How do we document an API boundary so a C maintainer can review it without learning the language?

Is 'build on the subsystem's newest C API' generalisable, or was PWM's waveform API just lucky timing?

Backup - adoption and the maintainer relationship

What makes a C maintainer comfortable reviewing the Rust side on an ongoing basis?

Where do we go looking for the second driver - which subsystem, which SoC, which person?

Does a working demo on cheap hardware - a spinning fan on a Lichee Pi 4A - do more than a design document?

Thank you