

Current Developments in the Common Clk Framework

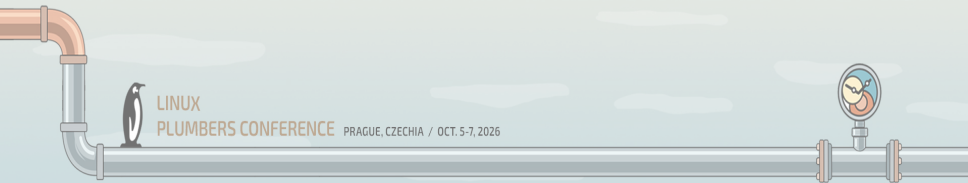
Brian Masney
CCF Co-Maintainer
bmasney@redhat.com

October 2026



Agenda

- Follow up from LPC 2025 in Japan about broken clock scaling issues. Walk through the problem, and how drivers are working around it today.
- Discussion about making the global clk prepare lock more fine grained.



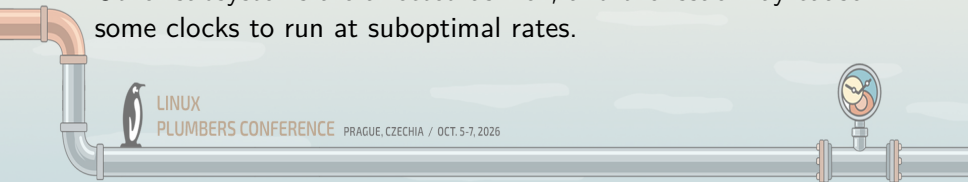
Broken Clock Scaling

The issue shows up when:

- Multiple sibling clocks divide down from a shared, configurable PLL.
- Each sibling needs its own precise rate via `CLK_SET_RATE_PARENT`.

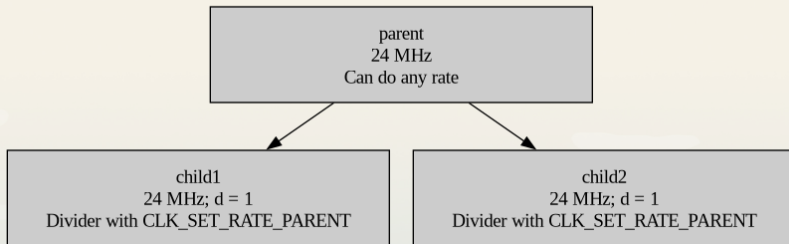
DRM/display and audio are the sensitive consumers that require exact rates. Issue has previously been reported on i.MX8M Plus and Allwinner A64.

Other subsystems are affected as well, and the issue may cause some clocks to run at suboptimal rates.



Problem 1: Divider Rate Change That Affects a Sibling (1 / 4)

Initial Clock Tree State



LINUX

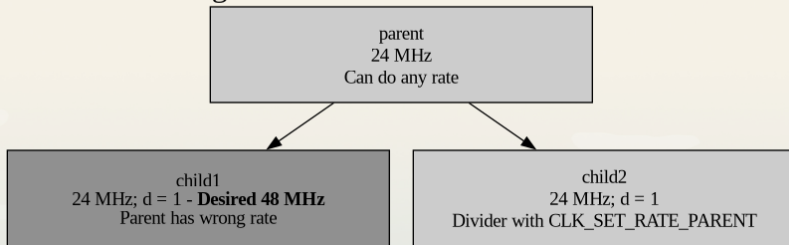
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem 1: Divider Rate Change That Affects a Sibling (2 / 4)

Change child1 from 24 MHz to 48 MHz



LINUX

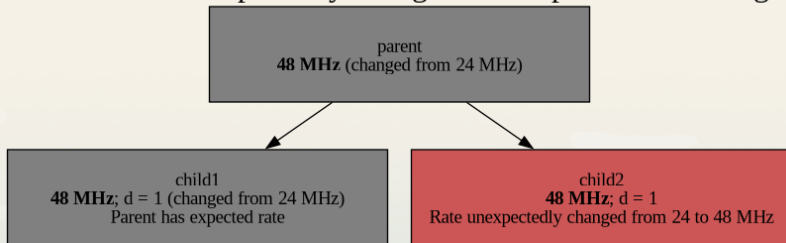
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem 1: Divider Rate Change That Affects a Sibling (3 / 4)

child2's rate unexpectedly changed due to parent rate change



LINUX

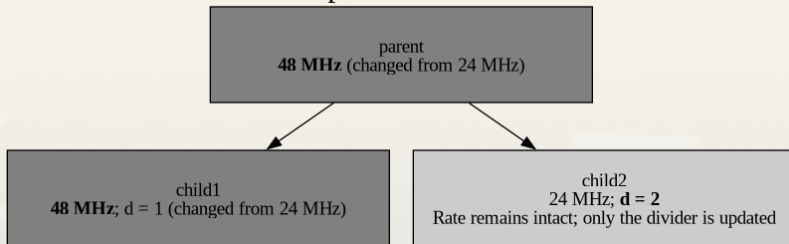
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem 1: Divider Rate Change That Affects a Sibling (4 / 4)

Expected Results



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Solution 1: Don't overwrite the sibling rate

`clk_calc_subtree()` always overwrites each sibling clock's rate when `CLK_SET_RATE_PARENT` is used. Siblings keep their rate (or as close as possible); the divider is derived again instead of clobbered.

I consider this to be a bug in the original code, and the few line fix is in this patch.

- [PATCH v8 5/8] clk: introduce new flag `CLK_V2_RATE_NEGOTIATION` for sensitive clocks
<https://lore.kernel.org/linux-clk/20260327-clk-scaling-v8-5-86cd0aba3c5f@redhat.com/>



LINUX

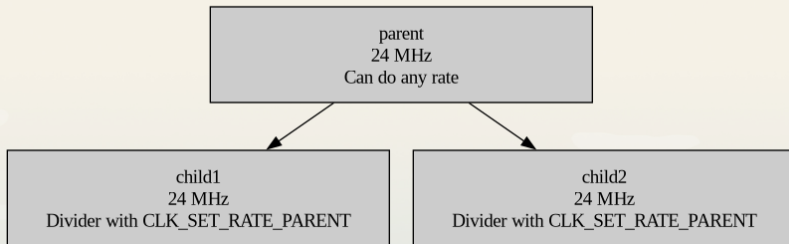
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem 2: No divider rate negotiation with siblings (1 / 4)

Initial Clock Tree State



LINUX

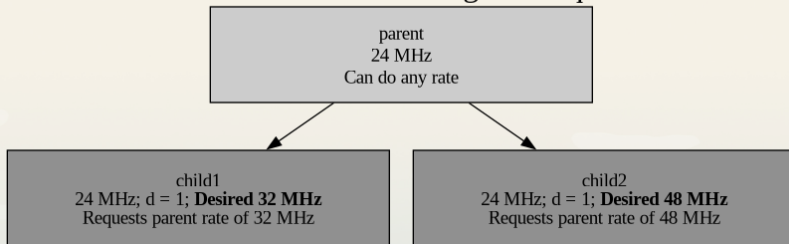
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem 2: No divider rate negotiation with siblings (2 / 4)

Children with conflicting rate requests



LINUX

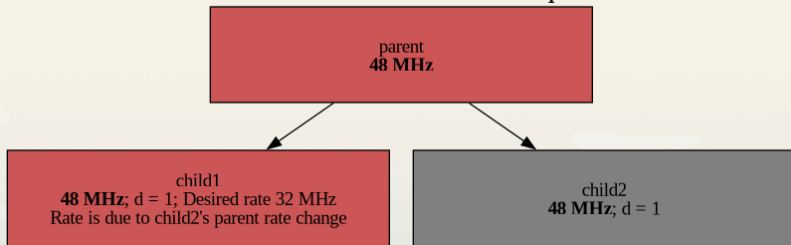
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem 2: No divider rate negotiation with siblings (3 / 4)

Current Behavior: Last Child Request Wins



LINUX

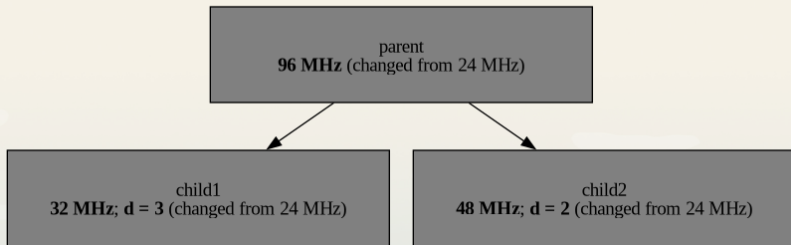
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Problem 2: No divider rate negotiation with siblings (4 / 4)

One Possible Correct Solution



LINUX

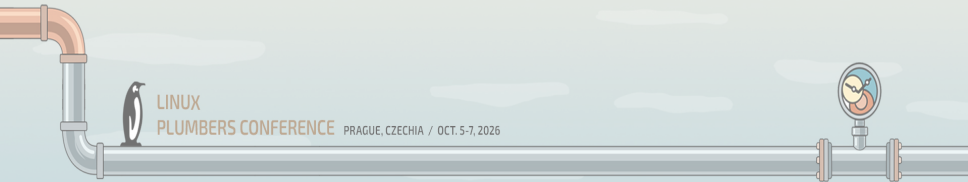
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



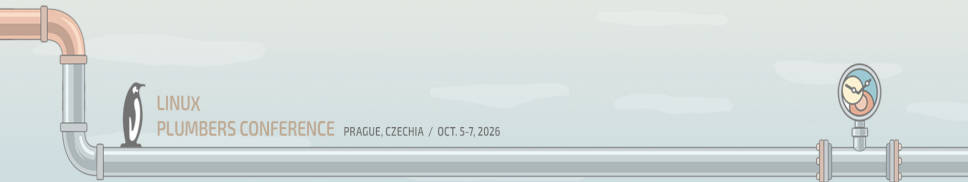
Solution 2: Add new helper to calculate LCM

Introduce functionality to `clk-divider.c` that recursively walks through all children and their descendants, calculating the lowest common multiple (LCM) of their rates.



Problem 3: Existing boards

Some existing boards are unknowingly dependent on the existing broken behavior. With over 1,100 files under `drivers/clock/` alone, and given the criticality of this subsystem, it's risky making big changes to the clock core.

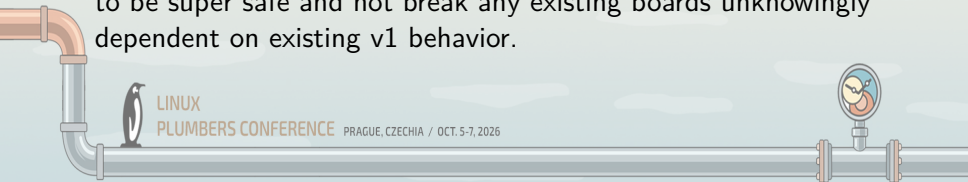


Solution 3: Add a CLK_V2_RATE_NEGOTIATION flag

Support both a v1 and v2 rate negotiation logic via a CLK_V2_RATE_NEGOTIATION clk flag.

- v1: unchanged existing broken behavior (clobbers sibling + parent rates), kept for compatibility.
- v2: new opt-in behavior for portions of the subtree that need this new behavior. Negotiation with the siblings for an optimal parent rate.

If we wrote the subsystem from scratch, this new flag wouldn't be there. Everyone would just get the v2 behavior. This is here to be super safe and not break any existing boards unknowingly dependent on existing v1 behavior.



Clk Scaling Latest Code

[https://git.kernel.org/pub/scm/linux/kernel/git/clk/linux.git/lo
g/?h=clk-divider-scaling-v9](https://git.kernel.org/pub/scm/linux/kernel/git/clk/linux.git/lo
g/?h=clk-divider-scaling-v9)

```
drivers/clk/.kunitconfig      |    1
drivers/clk/Kconfig           |    7 +++
drivers/clk/Makefile          |    1
drivers/clk/clk-divider.c     |   68 +++++
drivers/clk/clk-divider_test.c |  268 +++++
drivers/clk/clk.c             |  103 +++++
drivers/clk/clk_test.c        |    9 +---
include/kunit/clk.h           |   11 ++++
include/linux/clk-provider.h  |    6 +++
9 files changed, 453 insertions(+), 21 deletions(-)
```

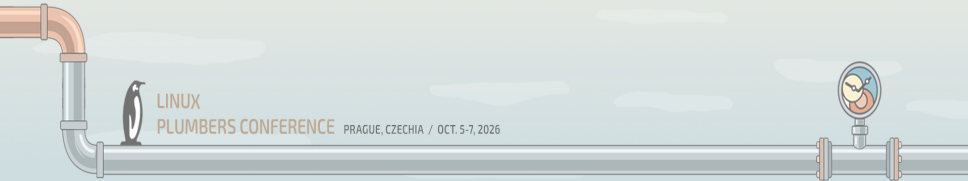


LINUX
PLUMBERS CONFERENCE PRAGUE, CZECHIA / OCT. 5-7, 2026



Existing Workarounds: Avoid Propagation

- Pin parent to a fixed rate, no `CLK_SET_RATE_PARENT`, just divide. Restricts available sibling rates. Sometimes the PLL is stuck high, which can waste power.
- `clk_set_rate_exclusive()` - locks parent, blocks negotiation, and limits possible child rates.



Existing Workarounds: Hand-rolled Coordination

- Manage the PLL in each platform driver. Re-invents core coordination. For example:
 - `tegra_audio_graph_update_pll()` in `sound/soc/tegra/tegra_audio_graph_card.c`
 - `pll-mipi` drives its parent `pll-video0` in `drivers/clk/sunxi-ng/ccu-sun50i-a64.c` via `ccu_nkm_find_best_with_parent_adj()`
- Rate change notifier. Driver fixes siblings when the shared parent moves. For example:
 - `rockchip_cpuclock_notifier_cb()` in `drivers/clk/rockchip/clk-cpu.c`
 - `exynos_cpuclock_notifier_cb()` in `drivers/clk/samsung/clk-cpu.c`



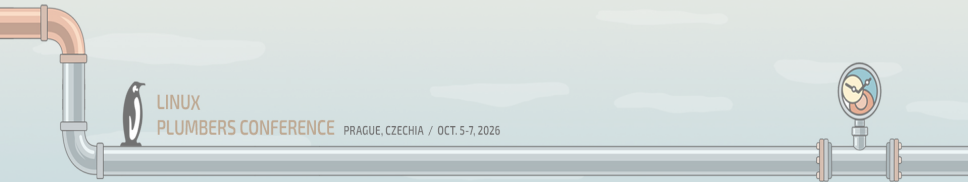
LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Clk Scaling Questions



Global clk prepare lock

The clk framework currently uses a single global, re-entrant mutex prepare lock:

```
static void clk_prepare_lock(void)
{
    if (!mutex_trylock(&prepare_lock)) {
        if (prepare_owner == current) {
            prepare_refcnt++;
            return;
        }
        mutex_lock(&prepare_lock);
    }
    ...
}
```



LINUX

PLUMBERS CONFERENCE

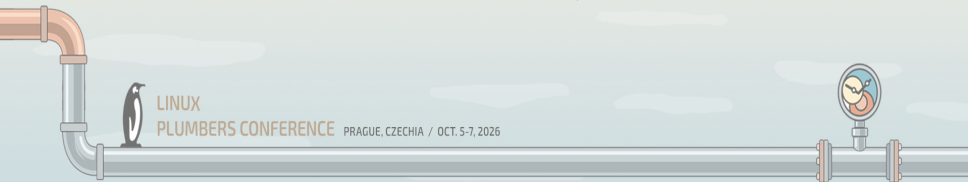
PRAGUE, CZECHIA / OCT. 5-7, 2026



Lockdep issues with the prepare lock

The `clk prepare_lock` sits in the middle of a large number of lock chains and lockdep sometimes reports possible circular locking dependencies involving it. Some recent examples:

- [RFC/PATCH] `clk: add lockdep_ignore_clk_prepare_lock` cmdline option
<https://lore.kernel.org/all/20260828131409.962825-1-m.szyprowski@samsung.com/>
- <https://lore.kernel.org/all/adqtn6iljcguv3m3ovhltusbbf4mljzlwg73yklaudvjjtazxv@vyf5fvuzic6x/>

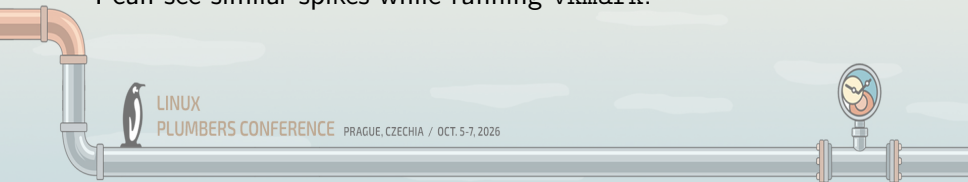


Contention on the prepare lock on boot up

On my Thinkpad x13s laptop (Qualcomm sc8280xp SoC), I gathered some statistics about how long it takes to acquire the global clk prepare lock on boot up:

- Samples: 122
- Min: 0 μ s
- Max: 10 361 μ s
- Mean: 243.04 μ s
- Median: 17 μ s

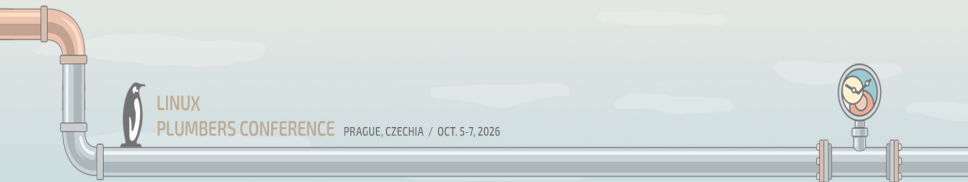
I can see similar spikes while running `vkmark`.



Dependencies

A more fine-grained locking scheme will increase the possibility of a deadlock.

- A clock in Controller A might rely on a power domain or voltage regulator controlled by Controller B, which itself depends on another clock.
- Parent locks need to be acquired, sometimes a new parent needs to be selected



Discussion

