PULSE-WIDTH-MODULATION (PWM) IS EASY, ISN'T IT?
(TURNING IT OFF AND ON AGAIN)
ABOUT ME & PENGUTRONIX

Uwe:

- kernel engineer @ Pengutronix since 2008
- PWM reviewer
- contributor to various kernel subsystems
- ukleinek on libera and OFTC
- PGP: E2DCDD9132669BD6

Pengutronix:

- Embedded Linux consulting & support since 2001

```
linux$ grep -c @pengutronix.de MAINTAINERS
40

linux$ git lg --author=@pengutronix.de v6.2-rc5 | wc -l
7078
```
WHAT IS A PWM?

- periodic square wave signal
- used to
  - blink or dim LEDs
  - drive display backlights
  - motor control (e.g. fan)
  - remote controls
ABSTRACTION OF A PWM

- period + duty cycle [ns]
- polarity (normal or inverted)
- enable & disable
Goal "Idempotency":

ops->get_state(mypwm, &state);
op->apply(mypwm, &state);

doesn't modify hw state.
SIMPLE ABSTRACT PWM

- Input clk: 13333 kHz
- quantum ≈ 75.001875... ns
- duty_cycle and period ∈ \{ 0 q, 1 q, ... 1023 q \}
ISSUE: API HAS DIFFERENT ACCURACY THAN HARDWARE

- Input clk: 13333 kHz
- quantum ≈ 75.001875... ns
- duty_cycle and period ∈ { 0 q, 1 q, ... 1023 q }

Request:

- period = 30000 ns
- duty_cycle = 16000 ns

Pick period:

- 399 q ≈ 29925.748 ns (Δ ≈ -74.252 ns)
- 400 q ≈ 30000.750 ns (Δ ≈ 0.750 ns)
ISSUE: PRECISION OF INTEGER MATH (DIVISION)

Request: period = 30000 ns

period_steps = clkrate / NSEC_PER_SEC * period

Always divide in the last step and only once.
ISSUE: TIME VS. FREQUENCY

- Input clk: 13333 kHz
- quantum ≃ 75.001875... ns
- duty_cycle and period ∈ { 0 q, 1 q, ... 1023 q }

Request: frequency = 1161587 Hz (period = 860.891 ns)

- pick period:
  - 11 q ≃ 825.021 ns (Δ ≃ -35.871 ns) ← better
  - 12 q ≃ 900.023 ns (Δ ≃ +39.131 ns)

- consider frequencies:
  - 1 / 11 q ≃ 1212090.909 Hz (Δ ≃ +50503.909 Hz)
  - 1 / 12 q ≃ 1111083.333 Hz (Δ ≃ -50503.667 Hz) ← better
ISSUE: PRECISION OF CLK_GET_RATE()

- Input clk: 32768 Hz
- quantum $\approx 30517.578125$ ns

Really: $\text{clk} \in (32767, 32769)$ Hz

$\Rightarrow$ quantum $\in (30516.646830846228, 30518.50947599719)$
ISSUE: TRANSITIONS

Reconfiguration request @14q to period = 12q + duty_cycle = 5q might result in:

- Completes old period

old period (18q)
old duty cycle (13q)
new period (12q)
new duty cycle (5q)
Reconfiguration request @14q to period = 12q + duty_cycle = 5q might result in:

- Immediate start of a new period:

```plaintext
\________________________|  period = 18q
|<-----------------------------|  duty cycle = 13q
```
Several more possible issues:

- mixed settings (e.g. a cycle with new period but the old duty cycle)
- hardware must be disabled for reconfiguration

Depending on hardware glitches cannot be prevented reliably.
ISSUE: BEHAVIOUR ON DISABLE

Typical (wrong) expectation:

```c
pwm_get_state(mypwm, &state);
state.enabled = false; // <--- Wrong!
state.duty_cycle = 0;
pwm_apply_state(mypwm, &state);
```

Usual behaviours:

- inactive level
- freeze
- high-Z

If you want constant inactive output, use

```c
state.enabled = true;
state.duty_cycle = 0;
```
FURTHER COMMON HARDWARE LIMITATIONS

- duty_cycle != 0
- duty_cycle != period
- shared or fixed period
- no .get_state() possible

```
sed -rn '/Limitations:.,/\*
\*/?$/p' drivers/pwm/*.c
```
ROUNDING STRATEGY (CONSUMER SIDE)

There is no "best" rounding strategy.

So pick an easy one: Always round down.

Consumers should know the result beforehand to determine "best" request.

Idea: new callback .round_state() that determines the state actually implemented for a given request (always rounding down).
API POLICY: ROUND DOWN PERIOD AND DUTY_CYCLE

- consistent .apply() <-> .get_state()
- time vs frequency
- simple to implement
- simple to work with (.round_state())

Status quo: 😞
Enabling PWM_DEBUG during tests

Compares HW state before and after a call to .apply(). Wails if the old state is a better match for the request than the new state or the new state is determined using unexpected rounding.

Tests idempotency.
ADVICE TO DRIVER AUTHORS (CONT)

- document hardware properties
- link to manual