

**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
```

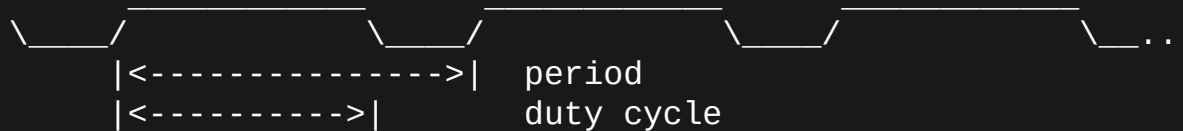
```
linux$ git lg --author=@pengutronix.de v6.2-rc5 | wc -l
7078
```

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



ABSTRACTION OF A PWM (CONT)

```
struct pwm_state {
    u64 period;          // [ns]
    u64 duty_cycle;     // [ns]
    enum pwm_polarity polarity;
    bool enabled;
    ...
};

struct pwm_ops {
    ...
    int (*apply)(struct pwm_device *pwm, const struct pwm_state *state);
    int (*get_state)(struct pwm_device *pwm, struct pwm_state *state);
    ...
};
```

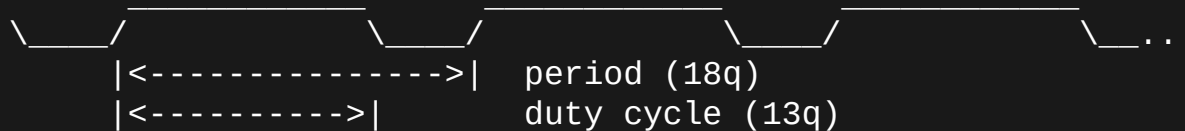
Goal "Idempotency":

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

doesn't modify hw state.

SIMPLE ABSTRACT PWM

- Input clk: 13333 kHz
- quantum $\approx 75.001875... \text{ ns}$
- duty_cycle and period $\in \{0 \text{ q}, 1 \text{ q}, \dots 1023 \text{ q}\}$



ISSUE: API HAS DIFFERENT ACCURACY THAN HARDWARE

- Input clk: 13333 kHz
- quantum $\approx 75.001875\dots$ ns
- duty_cycle and period $\in \{0\text{ q}, 1\text{ q}, \dots 1023\text{ q}\}$

Request:

- period = 30000 ns
- duty_cycle = 16000 ns

Pick period:

- 399 q ≈ 29925.748 ns ($\Delta \approx -74.252$ ns)
- 400 q ≈ 30000.750 ns ($\Delta \approx 0.750$ 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 $\approx 75.001875... \text{ ns}$
- duty_cycle and period $\in \{0 \text{ q}, 1 \text{ q}, \dots 1023 \text{ q}\}$

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

- pick period:
 - 11 q $\approx 825.021 \text{ ns}$ ($\Delta \approx -35.871 \text{ ns}$) ← better
 - 12 q $\approx 900.023 \text{ ns}$ ($\Delta \approx +39.131 \text{ ns}$)
- consider frequencies:
 - $1 / 11 \text{ q} \approx 1212090.909 \text{ Hz}$ ($\Delta \approx +50503.909 \text{ Hz}$)
 - $1 / 12 \text{ q} \approx 1111083.333 \text{ Hz}$ ($\Delta \approx -50503.667 \text{ Hz}$) ← better

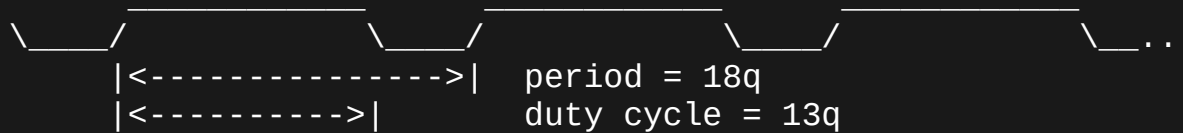
ISSUE: PRECISION OF CLK_GET_RATE()

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

Really: clk \in (32767, 32769) Hz

=> quantum \in (30516.646830846228, 30518.50947599719)

ISSUE: TRANSITIONS

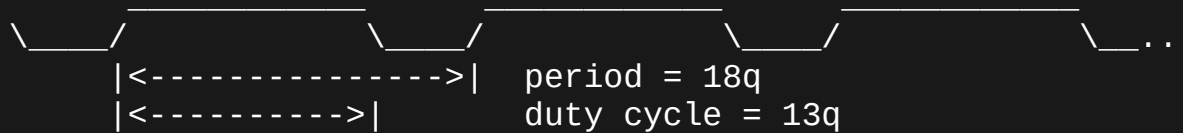


Reconfiguration request @14q to $\text{period} = 12q + \text{duty_cycle} = 5q$
might result in:

- Completes old period



ISSUE: TRANSITIONS (CONT)



Reconfiguration request @14q to $\text{period} = 12q + \text{duty_cycle} = 5q$
might result in:

- Immediate start of a new period:



ISSUE: TRANSITIONS (CONT)

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:

```
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

```
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: 🙄

ADVICE TO DRIVER AUTHORS

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