Laurentiu Mihalcea

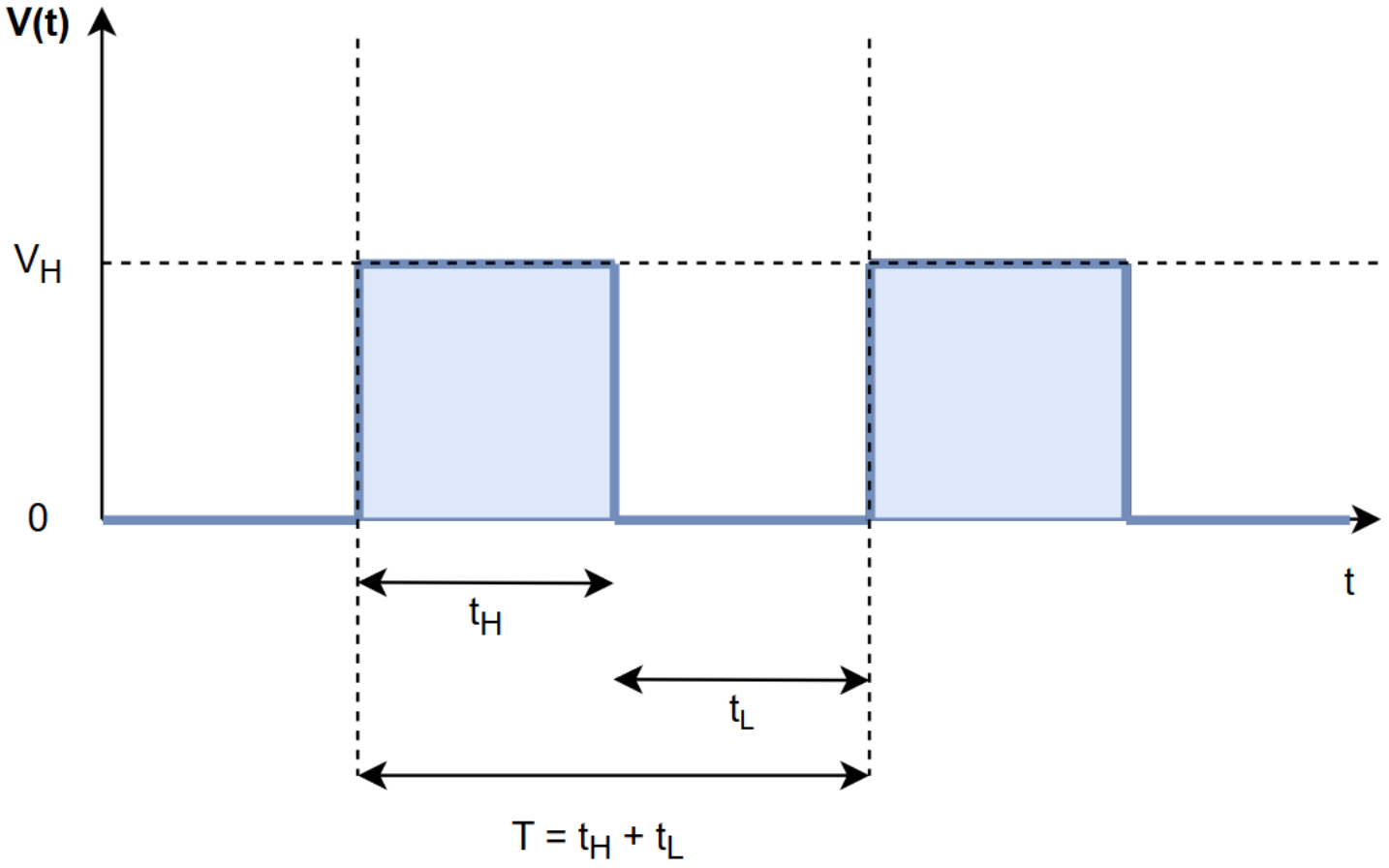
Iuliana Prodan

Daniel Baluta

| **Public** | NXP, and the NXP logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2024 NXP B.V.

Day 4: Pulse Width Modulation (PWM)

Recap – digital signals



T = period

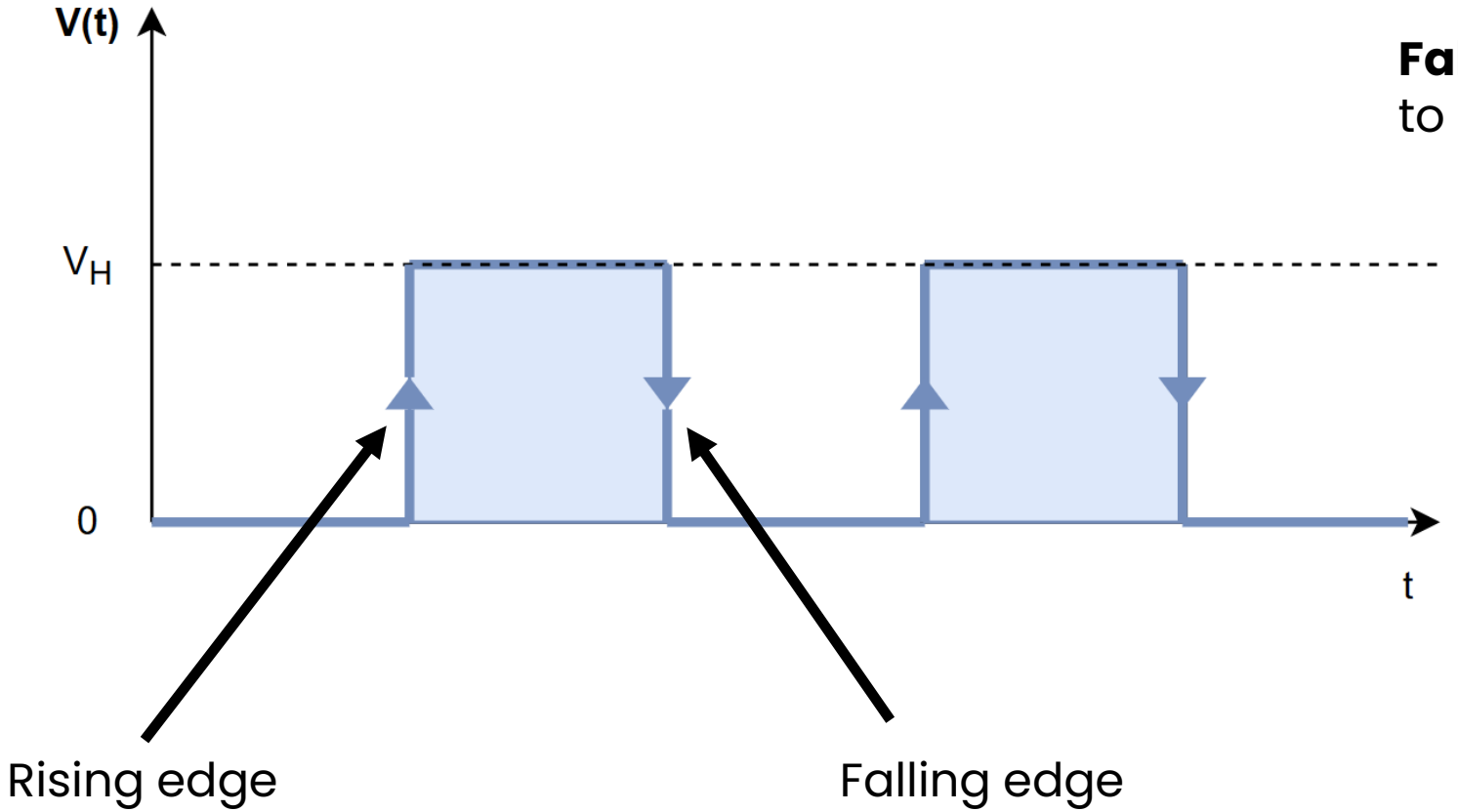
V_H = high level output voltage

Notes and sources section

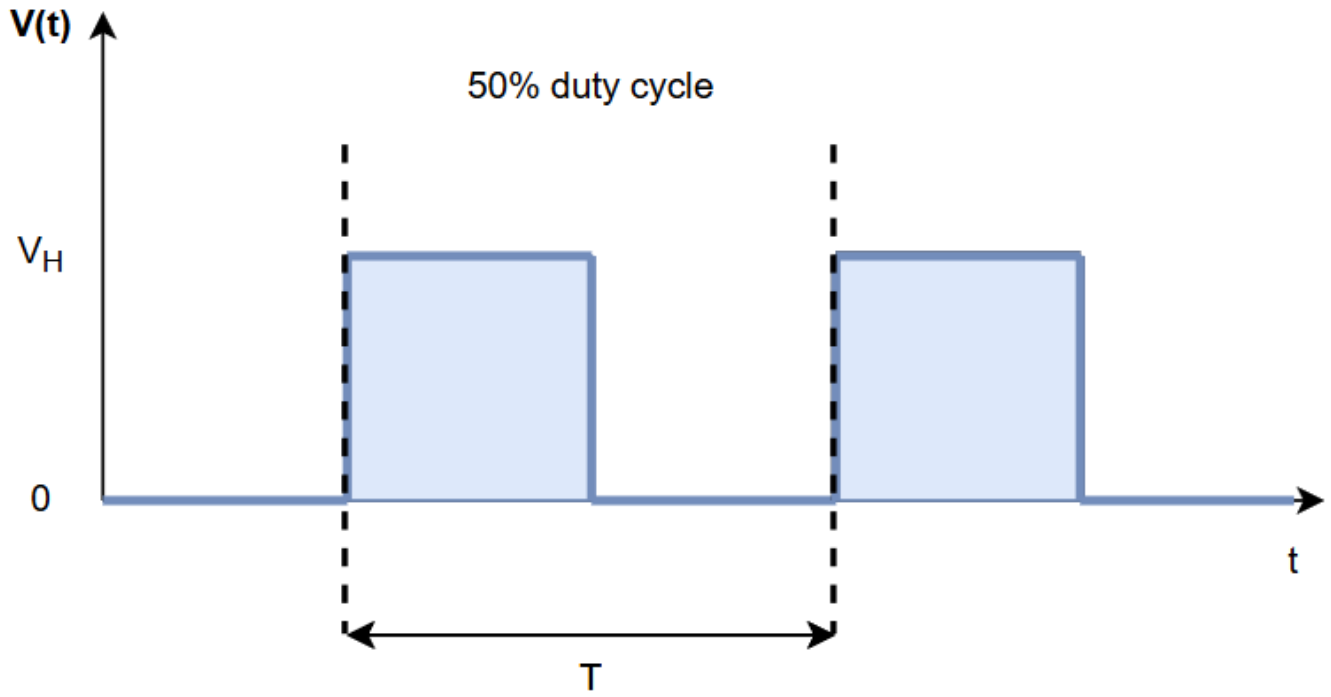
Recap – digital signals

Rising edge – transition from logic LOW to logic HIGH

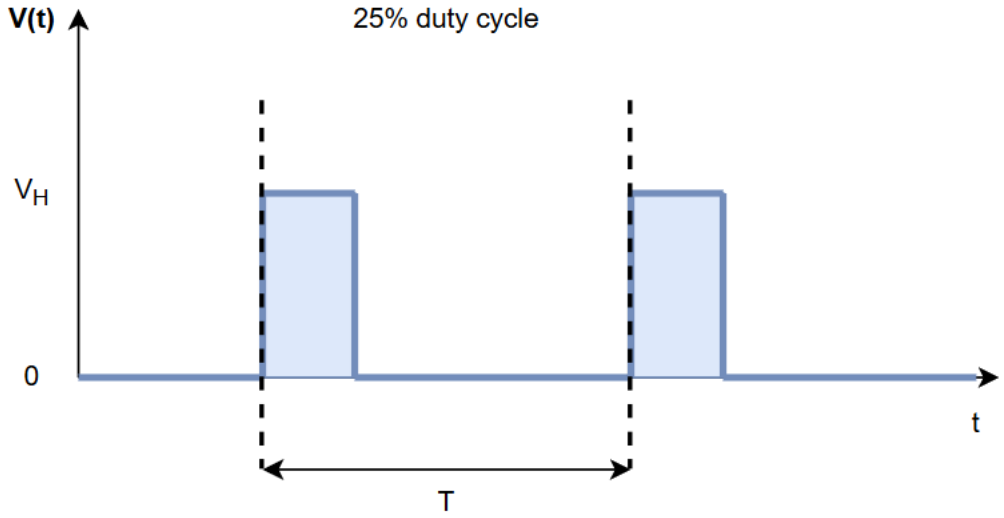
Falling edge – transition from logic HIGH to logic LOW

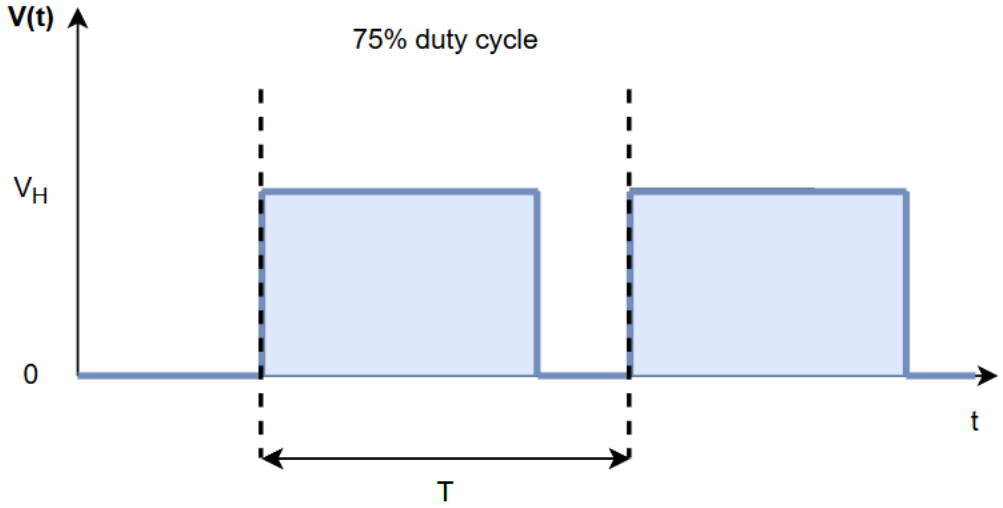


Recap – digital signals



Recap – digital signals

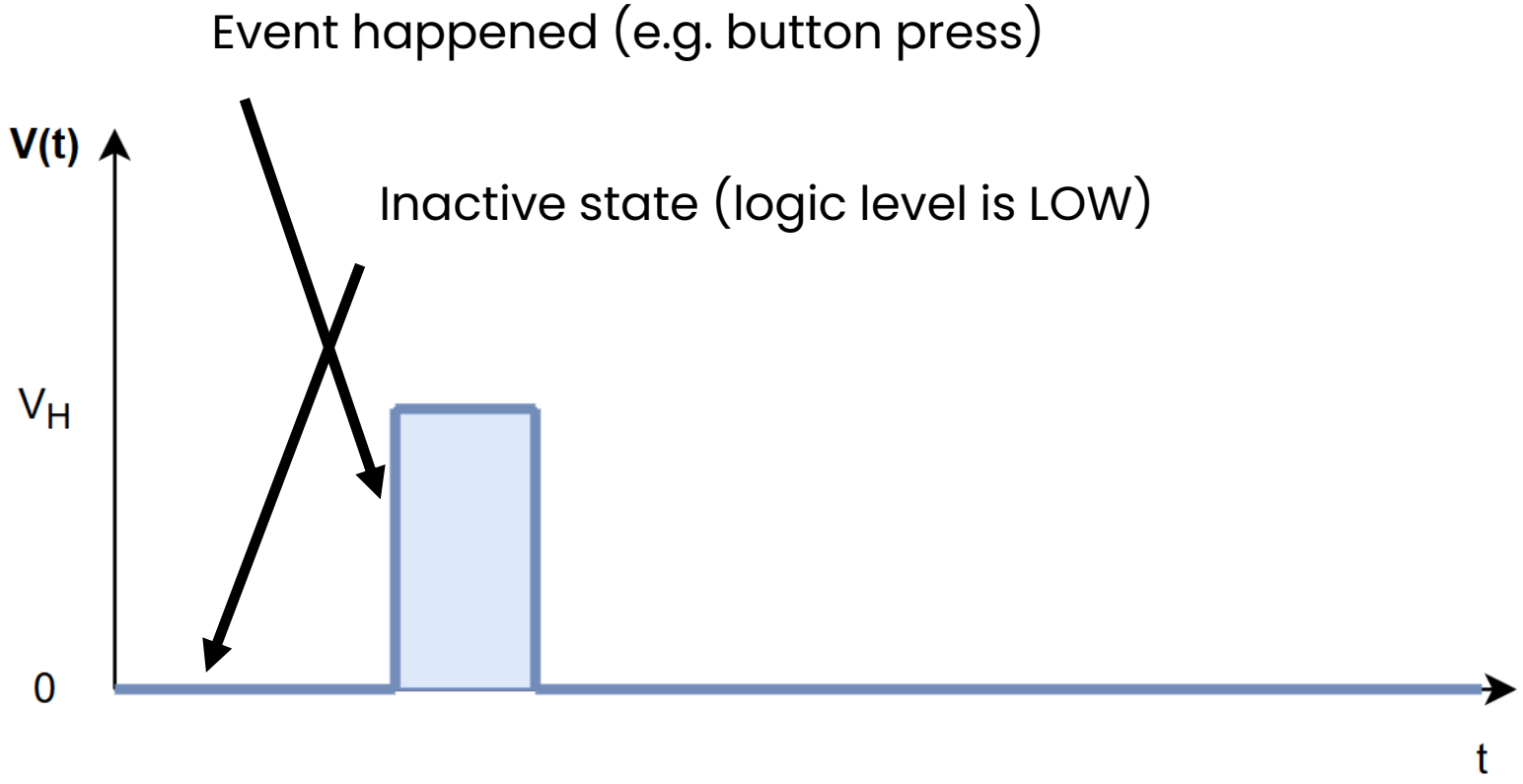




Recap – digital signals

$$D = \frac{t_H}{T} * 100 = \text{duty cycle } (\%)$$

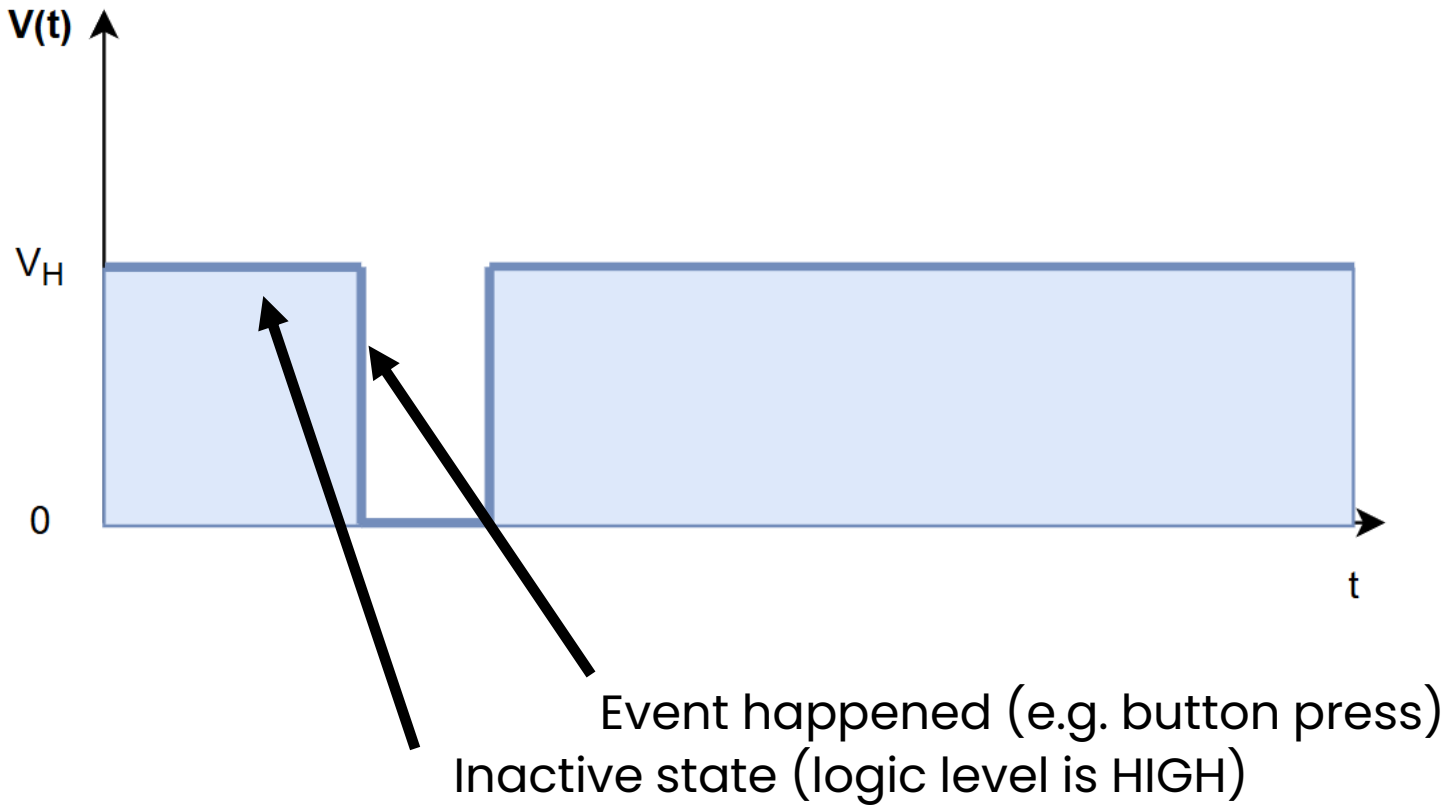
Recap – digital signals



Polarity - ACTIVE HIGH

Recap – digital signals

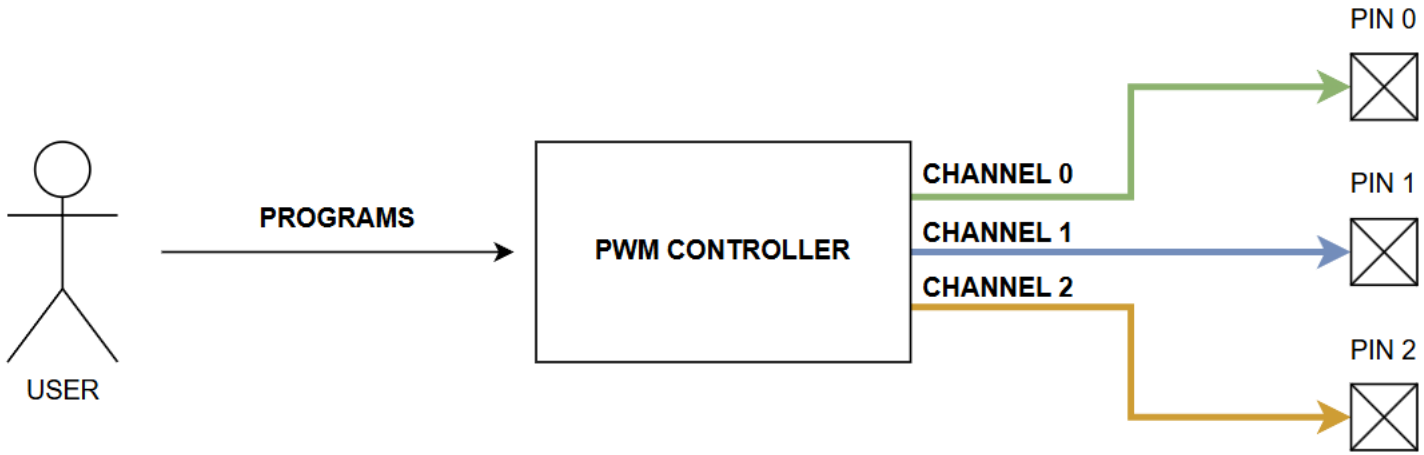
Polarity - ACTIVE LOW



Pulse Width Modulation (PWM)

- change the signal’s duty cycle to control the average power
- applications: LED dimming, motor control, etc.

PWM in embedded devices



PWM using Linux

- PWM controller = PWM chip
- PWM channel -> identified by a reference to a struct pwm_device

```
struct pwm_device {
        const char *label;
        unsigned long flags;
        unsigned int hwpwm;
        struct pwm_chip *chip;

        struct pwm_args args;
        struct pwm_state state;
        struct pwm_state last;
};
```

- **label** - channel string identifier
- **flags** – used by the PWM core
- **hwpwm** – channel identifier
- **chip –** reference to PWM controller (chip)
- **args** – DTB arguments

- PWM channels are requested using **pwm_get():**

```
struct pwm_device *pwm_get(struct device *dev, const char *con_id);
```

```
buzzer {
        compatible = "lkss,buzzer";
        pwms = <&tpm3 0 100000 0>;
        status = "disabled";
};
```

```
pwm-rgb-led {
        compatible = "lkss,pwm-rgb-led";
        pwms = <&tpm3 0 100000 0>,
               <&tpm3 1 100000 0>,
               <&tpm3 2 100000 0>;
        pwm-names = "red", "blue", "green";
        status = "disabled";
};
```

- when done with a PWM channel, you can release it using **pwm_put():**

```
void pwm_put(struct pwm_device *pwm);
```

PWM states

- each PWM channel has a state
- information provided by the state:
  - period
  - duty cycle
  - enabled/disabled
  - duty cycle
  - polarity
- identified by a **struct pwm_state**

```
struct pwm_state {
        u64 period;
        u64 duty_cycle;
        enum pwm_polarity polarity;
        bool enabled;
        bool usage_power;
};
```

- state can be initialized with **pwm_init_state():**

```
static inline void pwm_init_state(const struct pwm_device *pwm,
                                  struct pwm_state *state)
```

- state can be applied with **pwm_apply_might_sleep():**

```
int pwm_apply_might_sleep(struct pwm_device *pwm, const struct pwm_state *state);
```

Enabling/disabling a PWM channel

- enable using **pwm_enable():**

```
static inline int pwm_enable(struct pwm_device *pwm)
```

- disable using **pwm_disable():**

```
static inline void pwm_disable(struct pwm_device *pwm)
```

PWM channel configuration flow

```c
/* Step 1: request the PWM channel */
struct pwm_device *pwm_dev = pwm_get(dev, NULL);

/* Step 2: initialize the PWM state */
struct pwm_state state;
pwm_init_state(pwm_dev, &state);

/* Step 3: modify the state to meet your requirements */
state.period = 100000;
state.duty_cycle = state.period / 2;
state.polarity = PWM_POLARITY_INVERSED;

/* Step 4: apply the new state */
pwm_apply_might_sleep(pwm_dev, &state);

/* Step 5: enable the PWM */
pwm_enable(pwm_dev);
```

Setting the duty cycle (alternative)

```c
/* Step 1: request the PWM channel */
struct pwm_device *pwm_dev = pwm_get(dev, NULL);

/* Step 2: initialize the PWM state */
struct pwm_state state;
pwm_init_state(pwm_dev, &state);

/* Step 3: modify the state to meet your requirements */
state.period = 100000;
state.polarity = PWM_POLARITY_INVERSED;

pwm_set_relative_duty_cycle(&state, 50, 100);

/* Step 4: apply the new state */
pwm_apply_might_sleep(pwm_dev, &state);

/* Step 5: enable the PWM */
pwm_enable(pwm_dev);
```