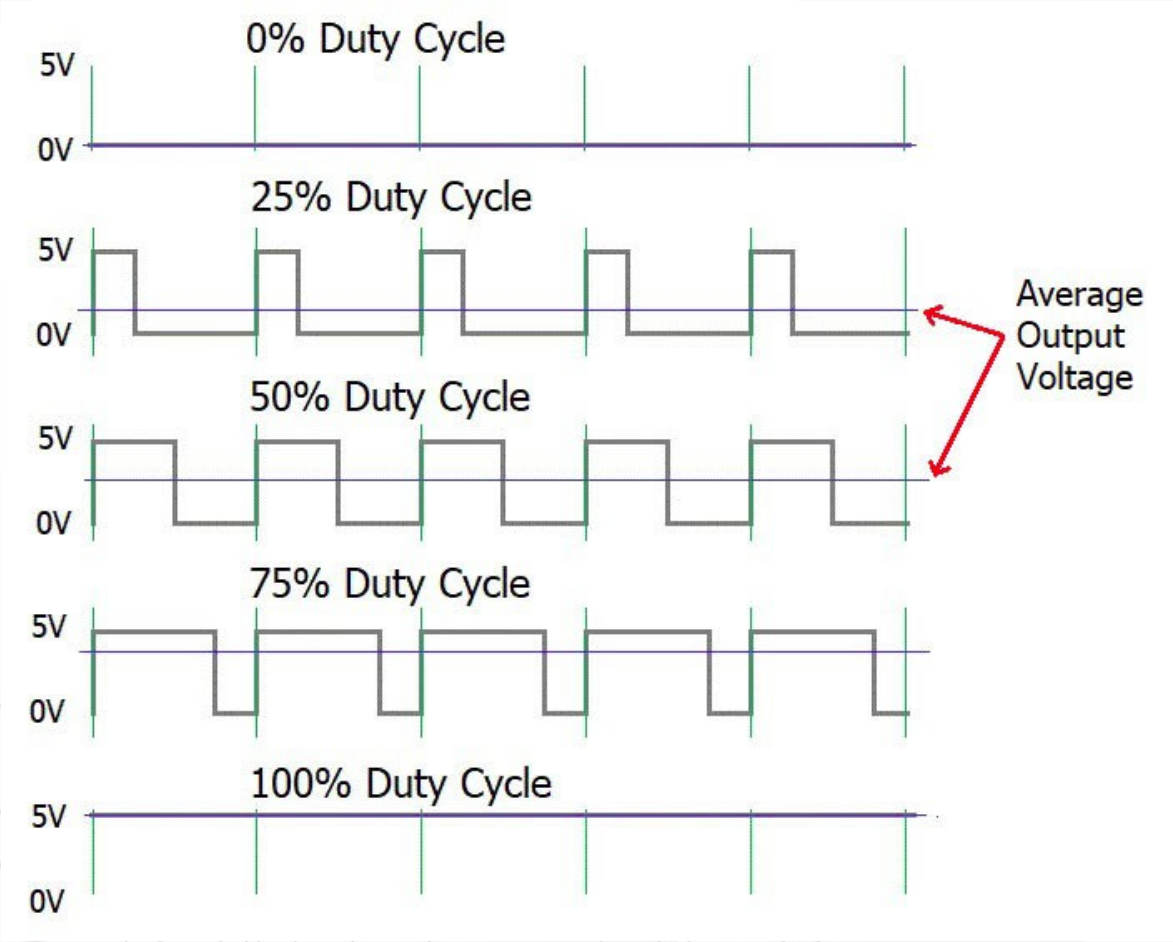
PWM

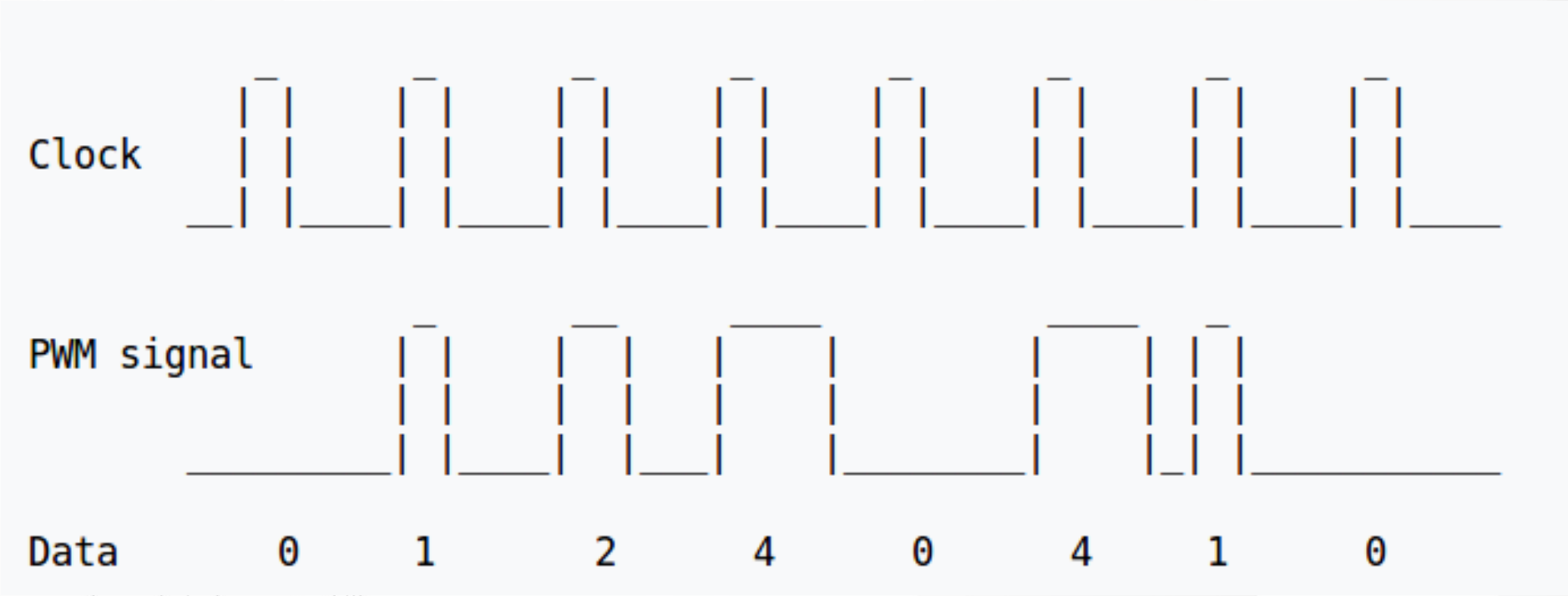
PWM

- PWM : Pulse Width Modulation



https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation

PWM



https://en.wikipedia.org/wiki/Pulse-width_modulation

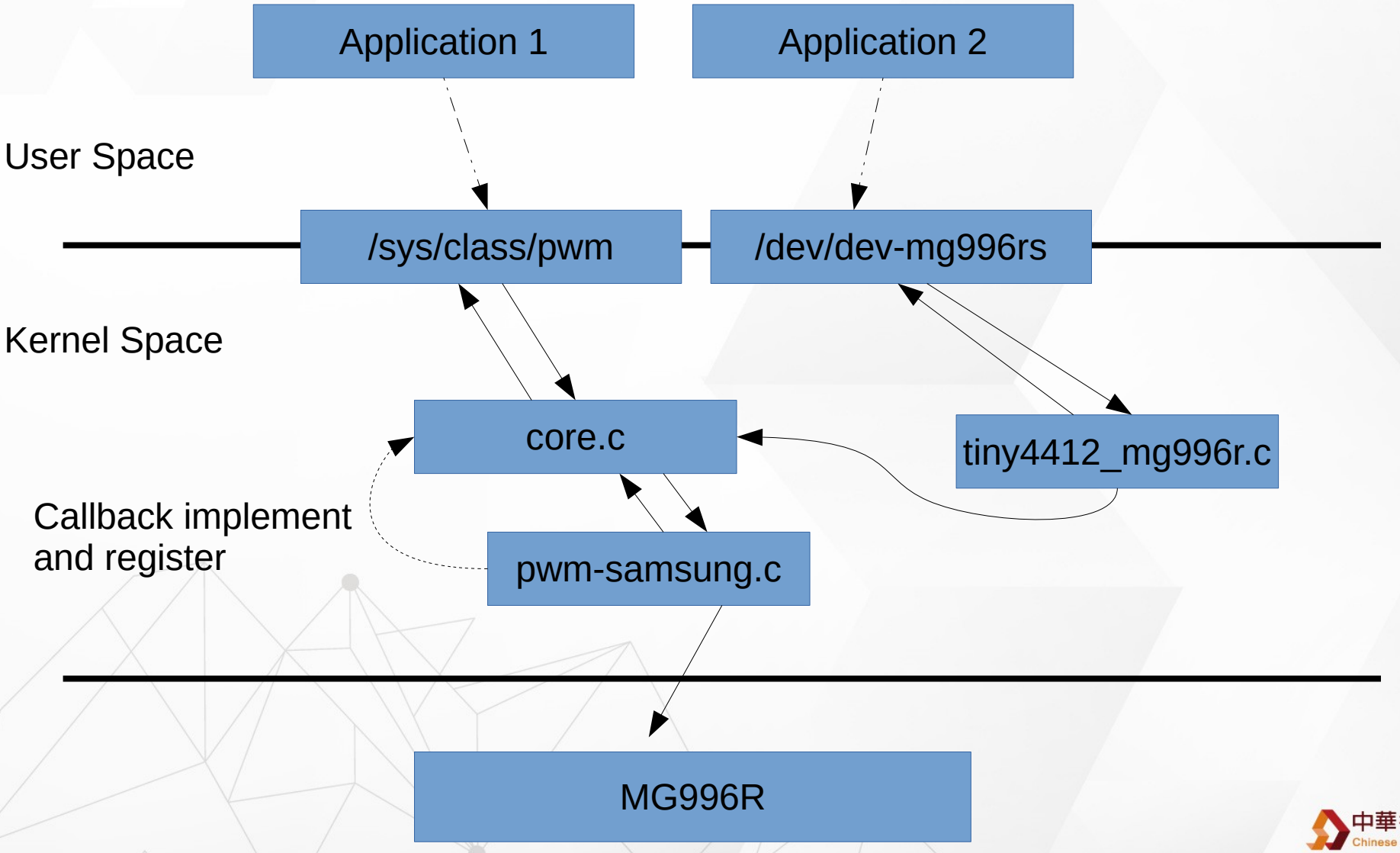
PWM Parameter in Linux

- Period
  - The total period of the PWM signal
  - Value is in nanoseconds
  - sum of the active and inactive time of the PWM
- duty_cycle
  - The active time of the PWM signal
  - Value is in nanoseconds
  - must be less than the period.
- Polarity
  - The polarity of the PWM signal
- Enable

PWM Driver

- $(KERNEL_SRC)/Documentation/pwm.txt
- Platform Driver
  - drivers/pwm/
  - drivers/pwm/core.c
  - drivers/pwm/pwm-samsung.c

PWM Subsystem



PWM DeviceTree

exynos4.dtsi

```
pwm: pwm@139d0000 {
        compatible = "samsung,exynos4210-pwm";
        reg = <0x139D0000 0x1000>;
        interrupts = <GIC_SPI 37 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 38 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 39 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 40 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 41 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&clock CLK_PWM>;
        clock-names = "timers";
        #pwm-cells = <3>;
        status = "disabled";
};
```

PWM DeviceTree

exynos4412-tiny4412.dts

```
&pwm {
	/* PWM 0 -- For MG996R */
	pinctrl-0 = <&pwm0_out>;
	pinctrl-names = "default";
	samsung,pwm-outputs = <0>;
	status = "okay";
};
```

PWM SYSFS

```
/sys/class/pwm/pwmchip0

device    export    npwm    power    subsystem  uevent    unexport
```

```
echo 0 > export

capture    enable    polarity    uevent    duty_cycle  period    power
```

```
echo “20000000” > period          //20ms, 50 Hz

echo "2000000" > duty_cycle       //2ms

echo 1 > enable                   //Enable
```

PWM - MG996G

```
mg996r {
    compatible = "tiny4412,mg996r-sample";
    pwms = <&pwm 0 20000000 100>;
    /* PWM 0 -- For MG996R */
    status = "okay";
};
```

PWM - API

```
/**
 * devm_of_pwm_get() - resource managed of_pwm_get()
 * @dev: device for PWM consumer
 * @np: device node to get the PWM from
 * @con_id: consumer name
 *
 * This function performs like of_pwm_get() but the acquired PWM device will
 * automatically be released on driver detach.
 *
 * Returns: A pointer to the requested PWM device or an ERR_PTR()-
encoded
 * error code on failure.
 */
struct pwm_device *devm_of_pwm_get(struct device *dev, struct device_node *np,
                                   const char *con_id)
```

PWM - API

```
/**
 * pwm_config() - change a PWM device configuration
 * @pwm: PWM device
 * @duty_ns: "on" time (in nanoseconds)
 * @period_ns: duration (in nanoseconds) of one cycle
 *
 * Returns: 0 on success or a negative error code on failure.
 */
```

```
static inline int pwm_config(struct pwm_device *pwm, int duty_ns,
                             int period_ns)
```

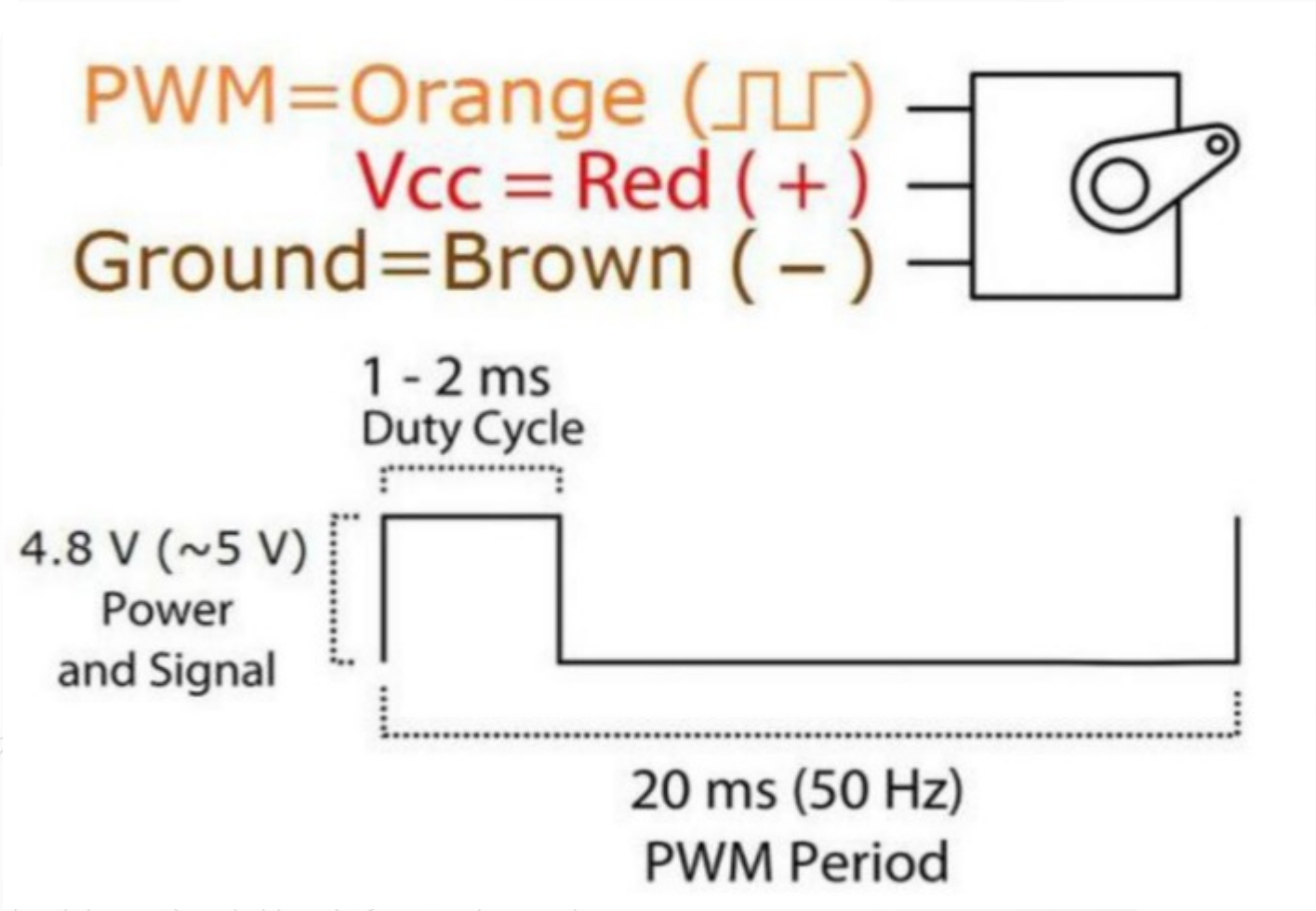
PWM - API

```
/**
 * pwm_enable() - start a PWM output toggling
 * @pwm: PWM device
 *
 * Returns: 0 on success or a negative error code on failure.
 */
static inline int pwm_enable(struct pwm_device *pwm)
```

```
/**
 * pwm_disable() - stop a PWM output toggling
 * @pwm: PWM device
 */
static inline void pwm_disable(struct pwm_device *pwm)
```

MG966R

MG966R



Tiny4412-mg966r.c

Tiny4412-mg966r.c

```
static int tiny4412_mg996r_probe(struct platform_device *pdev)
{
	int ret;
	struct device_node	*node = pdev->dev.of_node;

	mg996r.pwm_device = devm_of_pwm_get(&pdev->dev, node, NULL);
	pwm_config(mg996r.pwm_device, MG996R_TURN_RIGHT, MG996R_PERIOD);

	pr_err("period:%d, duty:%d\r\n",
		pwm_get_period(mg996r.pwm_device),
		pwm_get_duty_cycle(mg996r.pwm_device));

	ret = misc_register(&tiny4412_mg996r_dev);
	if (ret) {
		dev_err(&pdev->dev, "misc_register fail\r\n");
		return ret;
	}

	return 0;
}
```

Tiny4412-mg966r.c

Tiny4412-mg966r.c

```
static long tiny4412_mg996rs_ioctl(struct file *filp, unsigned int cmd,
        unsigned long arg)
{
    pr_err("%s: %lu 0x%x\n", __func__, arg, cmd);

    switch (cmd) {
        case IOCTL_TURN_ON:
            pwm_enable(mg996r.pwm_device);
            break;
        case IOCTL_TURN_OFF:
            pwm_disable(mg996r.pwm_device);
            break;
        case IOCTL_TURN_RIGHT:
            pr_err("%s: IOCTL_TURN_RIGHT\n", __func__);
            pwm_config(mg996r.pwm_device, MG996R_TURN_RIGHT, MG996R_PERIOD);
            break;

        case IOCTL_TURN_LEFT:
            pr_err("%s: IOCTL_TURN_LEFT\n", __func__);
            pwm_config(mg996r.pwm_device, MG996R_TURN_LEFT, MG996R_PERIOD);
            break;

        default:
            return -EINVAL;
    }

    pr_err("period:%d, duty:%d\r\n",
        pwm_get_period(mg996r.pwm_device),
        pwm_get_duty_cycle(mg996r.pwm_device));

    return 0;
}
```