测试用例

任务说明：编写 allwinner-hal 包 RCC 外设与 CPU 与外设时钟寄存器 的测试用例。

全志Allwinner公司的芯片具有复位与时钟控制器（Reset and Clock Controller, RCC）外设，可用于配置芯片级别各个外设与CPU的时钟开关、时钟源和频率，从而达到控制外设的效果。而 allwinner-hal 包中为 RCC 外设准备了以下的结构体，描述 RCC 外设中目前已支持的各个寄存器。在 `allwinner-hal/src/ccu.rs` 中，我们可以找到如下所示的时钟控制单元**寄存器**的结构体：

```
/// Clock Control Unit registers.
#[repr(C)]
pub struct RegisterBlock {
    /// 0x0 - CPU PLL Control register.
    pub pll_cpu_control: RW<PllCpuControl>,
    _reserved0: [u32; 3],
    /// 0x10 - DDR PLL Control register.
    pub pll_ddr_control: RW<PllDdrControl>,
    _reserved1: [u32; 3],
    /// 0x20 - Peripheral PLL Control register 0.
    pub pll_peri0_control: RW<PllPeri0Control>,
    _reserved2: [u32; 311],
    /// 0x500 - CPU AXI Configuration register.
    pub cpu_axi_config: RW<CpuAxiConfig>,
    _reserved3: [u32; 258],
    /// 0x90c - UART Bus Gating Reset register.
    pub uart_bgr: RW<UartBusGating>,
    _reserved4: [u32; 12],
    /// 0x940..=0x944 - SPI0 Clock Register and SPI1 Clock Register.
    pub spi_clk: [RW<SpiClock>; 2],
    _reserved5: [u32; 9],
    /// 0x96c - SPI Bus Gating Reset register.
    pub spi_bgr: RW<SpiBusGating>,
}
```

而对每个寄存器，芯片定义了不同的寄存器位域，代表对分管该外设的复位和时钟寄存器采取不同的操作。我们以CPU PLL 控制寄存器 `PllCpuControl` 为例：

```
pub struct PllCpuControl(u32);

impl PllCpuControl {
    const PLL_ENABLE: u32 = 1 << 31;
    const PLL_LDO_ENABLE: u32 = 1 << 30;
    const LOCK_ENABLE: u32 = 1 << 29;
    const LOCK: u32 = 1 << 28;
    const PLL_OUTPUT_GATE: u32 = 1 << 27;
    const PLL_N: u32 = 0xff << 8;
    const PLL_M: u32 = 0x3 << 0;

    /// Get if PLL is enabled.
    #[inline]
    pub const fn is_pll_enabled(self) -> bool {
        self.0 & Self::PLL_ENABLE != 0
    }
```

```rust
    /// Enable PLL.
    #[inline]
    pub const fn enable_pll(self) -> Self {
        Self(self.0 | Self::PLL_ENABLE)
    }
    /// Disable PLL.
    #[inline]
    pub const fn disable_pll(self) -> Self {
        Self(self.0 & !Self::PLL_ENABLE)
    }
    /// Get if PLL LDO is enabled.
    #[inline]
    pub const fn is_pll_ldo_enabled(self) -> bool {
        self.0 & Self::PLL_LDO_ENABLE != 0
    }
    /// Enable PLL LDO.
    #[inline]
    pub const fn enable_pll_ldo(self) -> Self {
        Self(self.0 | Self::PLL_LDO_ENABLE)
    }
    /// Disable PLL LDO.
    #[inline]
    pub const fn disable_pll_ldo(self) -> Self {
        Self(self.0 & !Self::PLL_LDO_ENABLE)
    }
    /// Get if PLL lock is enabled.
    #[inline]
    pub const fn is_lock_enabled(self) -> bool {
        self.0 & Self::LOCK_ENABLE != 0
    }
    /// Get if PLL is locked.
    #[inline]
    pub const fn is_locked(self) -> bool {
        self.0 & Self::LOCK != 0
    }
    /// Gate PLL output.
    #[inline]
    pub const fn gate_pll_output(self) -> Self {
        Self(self.0 | Self::PLL_OUTPUT_GATE)
    }
    /// Ungate PLL output.
    pub const fn ungate_pll_output(self) -> Self {
        Self(self.0 & !Self::PLL_OUTPUT_GATE)
    }
    /// Get if PLL output is gated.
    #[inline]
    pub const fn is_pll_output_gated(self) -> bool {
        self.0 & Self::PLL_OUTPUT_GATE != 0
    }
    /// Get PLL N factor.
    #[inline]
    pub const fn pll_n(self) -> u8 {
        ((self.0 & Self::PLL_N) >> 8) as u8
    }
    /// Set PLL N factor.
    #[inline]
    pub const fn set_pll_n(self, val: u8) -> Self {
        Self((self.0 & !Self::PLL_N) | ((val as u32) << 8))
```

```
    }
    /// Get PLL M factor.
    #[inline]
    pub const fn pll_m(self) -> u8 {
        (self.0 & Self::PLL_M) as u8
    }
    /// Set PLL M factor.
    #[inline]
    pub const fn set_pll_m(self, val: u8) -> Self {
        Self((self.0 & !Self::PLL_M) | val as u32)
    }
}
```

`PllCpuControl` 寄存器位域的定义如下：

位	名称	权限	说明
31	pll_enable	R/W	是否打开此CPU PLL的总开关。（注：写入函数不应当写成"set_pll_enable"，而是写成不带参数的"enable_pll"和"disable_pll"；读取函数命名为"is_pll_enabled"，返回bool类型）
30	pll_ldo_enable	R/W	是否打开此CPU PLL所需的电源开关。（注：写入函数和读取函数同上，不再重复）
29	lock_enable	R/W	是否锁定CPU PLL的频率。当软件重设PLL的频率时，重设的配置写入后，应当关闭锁定开关，等待lock位变为1，表示配置已被应用到硬件，此后再打开锁定开关。打开或关闭pll时，先打开pll总开关，再打开锁定开关；先关闭锁定开关，再关闭总开关。
28	lock	R（注：只读，无需编写写入函数）	锁定状态，表示PLL是否已稳定工作。
27	pll_output_gate	R/W	是否打开PLL的输出开关。必须等待PLL稳定工作后，才能打开输出开关。
15:8	pll_n	R/W	N因子，它和M因子共同构成PLL输出时钟的倍频参数。（注：请使用u8类型）
1:0	pll_m	R/W	M因子。（注：使用u8类型）

（关于PLL的一句话知识：PLL是一种电路，能够将输入信号的频率锁定并实现倍频，从而生成更高频率的输出信号。在本项目的芯片中，PLL可用于配置对应外设的时钟频率，如CPU PLL可用于控制CPU的工作频率。）

针对该寄存器编写测试用例，示例如下：

```
#[cfg(test)]
mod tests {
    use super::{PllCpuControl, ...};  // ...处应为其余寄存器结构体
```

```rust
    #[test]
    fn struct_pll_cpu_control_functions() {
        let mut val = PllCpuControl(0x0);

        val = val.enable_pll();
        assert_eq!(val.0, 0x80000000);   // 1 << 31
        assert!(val.is_pill_enabled());

        val = val.disable_pll();
        assert_eq!(val.0, 0x00000000);
        assert!(!val.is_pill_enabled());

        // TODO: 补全下面的寄存器操作函数的测试用例
```

本任务内容是，参照glb: 新增外设，寄存器以及对应的测试用例 · Pull Request !40 · RustSBI/bl-soc - Gitee.com，补充 PllCpuControl、PllDdrControl、PllPeri0Control 和 CpuAxiConfig 寄存器的测试用例。