```
#include    "app_pwm.h"
#include    "nrf_drv_timer.h"
#include    "nrf_drv_ppi.h"
#include    "nrf_drv_common.h"
#include    "nrf_drv_gpiote.h"
#include    "nrf_gpiote.h"
#include    "nrf_gpio.h"
#include    "app_util.h"
#include    "app_util_platform.h"
#include    "nrf_assert.h"
#include    <stdio.h>


#define   APP_PWM_CHANNEL_INITIALIZED                     1
#define   APP_PWM_CHANNEL_UNINITIALIZED                   0

#define   APP_PWM_CHANNEL_ENABLED                         1
#define   APP_PWM_CHANNEL_DISABLED                        0

#define   TIMER_PRESCALER_MAX                             9
#define   TIMER_MAX_PULSEWIDTH_US_ON_16M                  4095

#define   APP_PWM_REQUIRED_PPI_CHANNELS_PER_INSTANCE      2
#define   APP_PWM_REQUIRED_PPI_CHANNELS_PER_CHANNEL       2

#define   UNALLOCATED                                     0xFFFFFFFFUL

#define   PWM_MAIN_CC_CHANNEL                             2
#define   PWM_SECONDARY_CC_CHANNEL                        3

volatile    uint8_t    m_pwm_ready_counter[TIMER_COUNT];

typedef   struct
{
        uint8_t                 gpio_pin;
        uint32_t                pulsewidth;
        nrf_ppi_channel_t       ppi_channels[   2 ];
        app_pwm_polarity_t      polarity;
        uint8_t                 initialized;
}    app_pwm_channel_cb_t;

typedef   struct
{
        app_pwm_channel_cb_t        channels_cb[APP_PWM_CHANNELS_PER_INSTANCE];
        uint32_t                    period;
        app_pwm_callback_t          p_ready_callback;
        nrf_ppi_channel_t           ppi_channels[   2 ];
        nrf_ppi_channel_group_t     ppi_group;
        nrf_drv_state_t             state;
}    app_pwm_cb_t;

static    const    app_pwm_t    *   m_instances[TIMER_COUNT];

//  获取给定实例或者通道极性的宏
#define   POLARITY_ACTIVE(INST,CH)       ((   ((app_pwm_cb_t*)(INST)->p_cb)->channels_cb[(CH)].polarity                ==
                    APP_PWM_POLARITY_ACTIVE_LOW)?(   0):(   1 ))
#define   POLARITY_INACTIVE(INST,CH)       ((   ((app_pwm_cb_t*)(INST)->p_cb)->channels_cb[(CH)].polarity                ==
                    APP_PWM_POLARITY_ACTIVE_LOW)?(   1):(   0 ))

/**   @brief   Workaround for PAN-73.
 *
 *    参数  [in] timer              定时器
 *    参数  [in] enable             使能或者除能
 */
static    void    pan73_workaround(NRF_TIMER_Type       *  p_timer,     bool    enable)
{
        if   (p_timer     ==   NRF_TIMER0)
        {
            *(uint32_t     *)  0x40008C0C    =  (enable     ?  1  :   0 );
        }
        else   if   (p_timer     ==   NRF_TIMER1)
        {
            *(uint32_t     *)  0x40009C0C    =  (enable     ?  1  :   0 );
        }
        else   if   (p_timer     ==   NRF_TIMER2)
        {
            *(uint32_t     *)  0x4000AC0C    =  (enable     ?  1  :   0 );
        }
        return    ;
}


  /**  函数描述：使能指定 PWM实例 IRQ
   *
   *  参数 [in] p_instance  PWM 实例
   */
__STATIC_INLINE void pwm_irq_enable(app_pwm_t const * const p_instance)
{
        nrf_drv_timer_compare_int_enable(p_instance->p_timer, PWM_MAIN_CC_CHANNEL);
}


  /**  函数描述：除能指定 PWM实例 IRQ 中断使能标识
   *
   *  参数 [in] p_instance  PWM 实例
   */
__STATIC_INLINE void pwm_irq_disable(app_pwm_t const * const p_instance)
{
        nrf_drv_timer_compare_int_disable(p_instance->p_timer, PWM_MAIN_CC_CHANNEL);
}


  /**  函数描述：除能 PWM通道 PPI
   *
   *  参数 [in] p_instance  PWM 实例
   */
__STATIC_INLINE void pwm_channel_ppi_disable(app_pwm_t const * const p_instance, uint8_t channel)
{
        app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;

        nrf_drv_ppi_channel_disable(p_cb->channels_cb[channel].ppi_channels[0]);
        nrf_drv_ppi_channel_disable(p_cb->channels_cb[channel].ppi_channels[1]);
}


  /**  函数描述：除能 PWM PPI
   *
   *  参数 [in] p_instance  PWM 实例
   */
__STATIC_INLINE void pwm_ppi_disable(app_pwm_t const * const p_instance)
{
        app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;

        nrf_drv_ppi_channel_disable(p_cb->ppi_channels[0 ]);
        nrf_drv_ppi_channel_disable(p_cb->ppi_channels[1 ]);
}


  /**  函数描述：在 duty 设置后，这个函数将会被中断函数调用
   *
   *  参数 [in] timer              PWM 模块使用的定时器
   *  参数 [in] timer_instance_id  定时器实例
   */
  void pwm_ready_tick(nrf_timer_event_t event_type, void * p_context)
{
        uint32_t timer_instance_id = (uint32_t)p_context;

        if (m_pwm_ready_counter[timer_instance_id])
        {
            --m_pwm_ready_counter[timer_instance_id];
            if (m_pwm_ready_counter[timer_instance_id])
            {
                return ;
            }

            // 禁止定时器中断
            nrf_drv_timer_compare_int_disable(m_instances[timer_instance_id]->p_timer,
  PWM_MAIN_CC_CHANNEL);
            app_pwm_cb_t * p_cb = (app_pwm_cb_t *)m_instances[timer_instance_id]->p_cb;

            // 调用 PWM完整周期结束回调函数 用于设置标志位 通知应用程序
            p_cb->p_ready_callback(timer_instance_id);
        }
}


```

```
/**   函数描述：资源重新分配
 *
 *    参数  [in] p_instance       PWM                实例
 */
static      void     pwm_dealloc(app_pwm_t          const     *    const     p_instance)
{
        app_pwm_cb_t      *   p_cb    =   (app_pwm_cb_t       *)p_instance->p_cb;

        for    (uint8_t        i   =   0;   i   <  APP_PWM_REQUIRED_PPI_CHANNELS_PER_INSTANCE;      ++i)
        {
                if    (p_cb->ppi_channels[i]               !=    (nrf_ppi_channel_t)(uint8_t)(UNALLOCATED))
                {
                        nrf_drv_ppi_channel_free(p_cb->ppi_channels[i]);
                }
        }
        if    (p_cb->ppi_group           !=    (nrf_ppi_channel_group_t)UNALLOCATED)
        {
                nrf_drv_ppi_group_free(p_cb->ppi_group);
        }

        for    (uint8_t        ch   =   0;   ch   <  APP_PWM_CHANNELS_PER_INSTANCE;    ++ch)
        {
                for    (uint8_t        i   =   0;   i   <  APP_PWM_REQUIRED_PPI_CHANNELS_PER_CHANNEL;     ++i)
                {
                        if    (p_cb->channels_cb[ch].ppi_channels[i]                        !=    (nrf_ppi_channel_t)UNALLOCATED)
                        {
                                nrf_drv_ppi_channel_free(p_cb->channels_cb[ch].ppi_channels[i]);
                                p_cb->channels_cb[ch].ppi_channels[i]                        =   (nrf_ppi_channel_t)UNALLOCATED;
                        }
                }
                if    (p_cb->channels_cb[ch].gpio_pin                    !=    UNALLOCATED)
                {
                        nrf_drv_gpiote_out_uninit(p_cb->channels_cb[ch].gpio_pin);
                        p_cb->channels_cb[ch].gpio_pin                    =   UNALLOCATED;
                }
                p_cb->channels_cb[ch].initialized                       =   APP_PWM_CHANNEL_UNINITIALIZED;
        }
        nrf_drv_timer_uninit(p_instance->p_timer);
        return    ;
}


/**PWM  状态转换      从 0%~100%   转换到   %0或者  100%
 *
 *    参数  [in] p_instance       PWM                实例
 *    参数  [in] channel          PWM                通道号
 *    参数  [in] ticks                               时钟数
 */
static      void     pwm_transition_n_to_0or100(app_pwm_t                 const     *   const      p_instance,
                                                          uint8_t       channel,      uint16_t       ticks)
{
        app_pwm_cb_t                          *   p_cb           =   (app_pwm_cb_t       *)p_instance->p_cb;
        app_pwm_channel_cb_t                  *   p_ch_cb        =   &p_cb->channels_cb[channel];
        nrf_ppi_channel_group_t               *   p_ppigrp       =   &p_cb->ppi_group;

        pwm_ppi_disable(p_instance);                                                                                     //  除能   PWM
PPI
        nrf_drv_ppi_group_clear(*p_ppigrp);                                                                              //  清组通道
        nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[                              0],     *p_ppigrp);
//  将 channel[0]      加入组通道
        nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[                              1],     *p_ppigrp);
//  将 channel[1]      加入组通道

        if    (!ticks)
//  设置时钟数为      0
        {
                nrf_drv_ppi_channel_assign(p_cb->ppi_channels[                          0],
                                         nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
channel),      //   通道  channel   比较事件
                                         nrf_drv_ppi_task_addr_group_disable_get(*p_ppigrp));
//  获取组通道除能寄存器地址
        }
        else
//  设置时钟数非      0
        {
                ticks      =   p_cb->period;
                nrf_drv_ppi_channel_assign(p_cb->ppi_channels[                          0],
                                         nrf_drv_timer_compare_event_address_get(p_instance->p_timer,                             PWM_MAIN_CC_CHANNEL),
                                nrf_drv_ppi_task_addr_group_disable_get(*p_ppigrp));
//  获取组通道除能寄存器地址
        }

        nrf_drv_ppi_channel_enable(p_cb->ppi_channels[                    0]);
//  使能  PPI 通道
        p_ch_cb->pulsewidth         =   ticks;
//  将 tick   赋值给   PWM脉宽
}


/**PWM  状态转换      从 0%~100%   转换到   %0~100%
 *
 *    参数  [in] p_instance       PWM                 实例
 *    参数  [in] channel          PWM                 通道号
 *    参数  [in] ticks                                时钟数
 */
static      void     pwm_transition_n_to_m(app_pwm_t              const     *   const     p_instance,
                                                        uint8_t        channel,        uint16_t         ticks)
{
        app_pwm_cb_t                      *   p_cb           =   (app_pwm_cb_t        *)p_instance->p_cb;
        app_pwm_channel_cb_t              *   p_ch_cb        =   &p_cb->channels_cb[channel];
        nrf_ppi_channel_group_t           *   p_ppigrp       =   &p_cb->ppi_group;

        nrf_drv_ppi_group_clear(*p_ppigrp);
        nrf_drv_ppi_channel_include_in_group(p_cb->ppi_channels[                        0],     *p_ppigrp);
        nrf_drv_ppi_channel_include_in_group(p_cb->ppi_channels[                        1],     *p_ppigrp);
        nrf_drv_ppi_group_disable(*p_ppigrp);

        /*  注释说明
       *
       *    这个函数主要是更新脉宽值              通过  PWM_SECONDARY_CC_CHANNEL通道事件       触发捕获事件
       *    将定时器计数值（其值为           PWM_SECONDARY_CC_CHANNEL）中的值更新到通道比较寄存器中
       *    达到脉宽更新的效果
       *    这个地方研究很久才看出来              官网数据手册就一句带过          ( 没有引起重视，导致半天没有看懂              )
       */
        nrf_drv_ppi_channel_assign(p_cb->ppi_channels[                    0],
                        nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
PWM_SECONDARY_CC_CHANNEL),
                        (uint32_t)&p_instance->p_timer->p_reg->TASKS_CAPTURE[channel]                              );

        if     (ticks        <   p_cb->channels_cb[channel].pulsewidth)
        {
                // For lower value we need one more transition.
                //  添加通道     1控制脉宽
                nrf_drv_ppi_channel_assign(p_cb->ppi_channels[                    1],
                        nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
PWM_SECONDARY_CC_CHANNEL),
                        nrf_drv_gpiote_out_task_addr_get(p_ch_cb->gpio_pin));
        }
        else
        {
                //  将通道    1移出
                nrf_drv_ppi_channel_remove_from_group(p_cb->ppi_channels[                    1],     *p_ppigrp);
        }

        //  注意     这里只使能了组通道
        nrf_drv_ppi_group_enable(*p_ppigrp);
        p_ch_cb->pulsewidth         =   ticks;

        //  这个地方采用第三通道比较数据产生              比较事件        然后将数据更新到第一第二通道              CC寄存器
        nrf_drv_timer_compare(p_instance->p_timer,                    (nrf_timer_cc_channel_t)                PWM_SECONDARY_CC_CHANNEL,
ticks,        false    );
}


/**PWM  状态转换      从 0%或者   100%   转换到其它值      n
 *
 *    参数  [in] p_instance       PWM                 实例
 *    参数  [in] channel          PWM                 通道号
 *    参数  [in] ticks                                时钟数
 */
static      void     pwm_transition_0or100_to_n(app_pwm_t              const     *   const     p_instance,
                                                          uint8_t        channel,        uint16_t         ticks)
{
        app_pwm_cb_t                      *   p_cb             =   (app_pwm_cb_t        *)p_instance->p_cb;
        app_pwm_channel_cb_t              *   p_ch_cb          =   &p_cb->channels_cb[channel];
        nrf_ppi_channel_group_t           *   p_ppigrp         =   &p_cb->ppi_group;
        nrf_timer_cc_channel_t       pwm_ch_cc   = (nrf_timer_cc_channel_t)(channel);

        pwm_ppi_disable(p_instance);
        pwm_channel_ppi_disable(p_instance,       channel);

        nrf_drv_timer_compare(p_instance->p_timer,        pwm_ch_cc,    ticks,     false   );
        nrf_drv_ppi_group_clear(*p_ppigrp);
        nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[          0],    *p_ppigrp);
        nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[          1],    *p_ppigrp);

        if   (!p_ch_cb->pulsewidth)
        {
                //  当前通道脉宽占空比为     0%。强制输出为非激活状态并且在下次转换调度
                nrf_drv_ppi_channel_assign(p_cb->ppi_channels[         0],
//PPI  通道  0 分配事件地址和任务地址
                     nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
channel),    //  获取通道   channel   通道比较事件地址
                     nrf_drv_ppi_task_addr_group_enable_get(*p_ppigrp));
//  获取组任务使能寄存器地址
        }
        else
        {
                //  当前通道脉宽占空比为     100%。强制输出为激活状态并且在下次转换调度
                nrf_drv_ppi_channel_assign(p_cb->ppi_channels[         0],
                     nrf_drv_timer_compare_event_address_get(p_instance->p_timer,         PWM_MAIN_CC_CHANNEL),
                     nrf_drv_ppi_task_addr_group_enable_get(*p_ppigrp));
        }
        nrf_drv_ppi_channel_enable(p_cb->ppi_channels[         0]);
//  使能  PPI 通道
        p_ch_cb->pulsewidth      =  ticks;
//  设置脉宽时钟数
}


/**PWM  状态转换    从 0%或者   100%   转换到   %0或者   100%
 *
 *   参数  [in] p_instance    PWM         实例
 *   参数  [in] channel      PWM          通道号
 *   参数  [in] ticks                    时钟数
 */
static    void   pwm_transition_0or100_to_0or100(app_pwm_t         const   *  const   p_instance,
                                                     uint8_t     channel,     uint16_t     ticks)
{
        app_pwm_cb_t             *  p_cb          = (app_pwm_cb_t     *)p_instance->p_cb;
        app_pwm_channel_cb_t     *  p_ch_cb       = &p_cb->channels_cb[channel];
        nrf_timer_cc_channel_t      pwm_ch_cc     = (nrf_timer_cc_channel_t)(channel);          //  获取通道

        pwm_ppi_disable(p_instance);
        pwm_channel_ppi_disable(p_instance,       channel);

        if   (!ticks)
        {
                //  设置为   0%     引脚强制输出为非激活态
                nrf_drv_gpiote_out_task_force(p_ch_cb->gpio_pin,         POLARITY_INACTIVE(p_instance,       channel));
        }
        else   if   (ticks     >=  p_cb->period)
        {
                //  设置为   100%  引脚强制输出为激活态
                ticks    =  p_cb->period;         //  将周期值付给     ticks
                nrf_drv_gpiote_out_task_force(p_ch_cb->gpio_pin,         POLARITY_ACTIVE(p_instance,       channel));
        }

        nrf_drv_timer_compare(p_instance->p_timer,        pwm_ch_cc,    ticks,     false   );
        p_ch_cb->pulsewidth      =  ticks;     //  将 ticks    赋值给     脉冲宽度
        return   ;
}


/**Function for setting PWM channel duty cycle in clock ticks.
 *
 *   参数  [in] p_instance    PWM         实例
 *   参数  [in] channel      PWM          通道号
 *   参数  [in] ticks                    时钟数
 *
 *   返回值        操作成功返回           NRF_SUCCESS
 *   返回值     PWM   模块没有准备好返回         NRF_ERROR_BUSY
 *   返回值        如果给定的实例没有初始化返回          NRF_ERROR_INVALID_STATE
 *
 *  注意事项：
 *  一个完成的 PWM完整时钟周期结束才能改变占空比
 *  在 PWM完整时钟周期结束前，任何尝试修改占空比操作将返回错误代码 NRF_ERROR_BUSY
 */
uint32_t app_pwm_channel_pulsewidth_ticks_set(app_pwm_t const * const p_instance,
                                              uint8_t channel, uint16_t ticks)
{
    app_pwm_cb_t        * p_cb     = (app_pwm_cb_t *)p_instance->p_cb;
    app_pwm_channel_cb_t * p_ch_cb = &p_cb->channels_cb[channel];

    ASSERT(channel < APP_PWM_CHANNELS_PER_INSTANCE);
    ASSERT(p_ch_cb->initialized == APP_PWM_CHANNEL_INITIALIZED);

    if (p_cb->state != NRF_DRV_STATE_POWERED_ON)
    {
        return NRF_ERROR_INVALID_STATE;     // 初始化执行到此处返回 提示 PWM电源模块没有打开
    }
    if (ticks == p_ch_cb->pulsewidth)
    {
        p_cb->p_ready_callback(p_instance->p_timer->instance_id);
        return NRF_SUCCESS;                 // No action required.
首次执行到此处返回（ ticks 和 pulsewidth 都为 0）
    }
    if (m_pwm_ready_counter[p_instance->p_timer->instance_id])
    {
        return NRF_ERROR_BUSY;              //
这个值在中断时会自动减一，只有减到 0后这个函数才能成功
    }
    m_pwm_ready_counter[p_instance->p_timer->instance_id] = 2; // PWM will be ready to next change
on 2nd CC0 event (minimum 1 full period)

    // 脉宽改变序列
    if (!p_ch_cb->pulsewidth || p_ch_cb->pulsewidth >= p_cb->period)
    {
        // 脉宽为 0%或者 100% 通道处于除能状态
        if (!ticks || ticks >= p_cb->period)
        {
            // 设置脉宽为 0%或者 100%
            pwm_transition_0or100_to_0or100(p_instance, channel, ticks);
        }
        else
        {
            // 设置脉宽在 0%~100%之间（不包含 0%和 100%）
            pwm_transition_0or100_to_n(p_instance, channel, ticks);
        }
    }
    else
    {
        // 脉宽在 0%~100%之间（不包含 0%和 100%） 通道处于使能状态
        if (!ticks || ticks >= p_cb->period)
        {
            // 设置脉宽为 0%或者 100% 关闭通道
            pwm_transition_n_to_0or100(p_instance, channel, ticks);
        }
        else
        {
            // 设置脉宽在 0%~100%之间（不包含 0%和 100%）
            pwm_transition_n_to_m(p_instance, channel, ticks);
        }
    }
    pwm_irq_enable(p_instance);   // 使能定时器 PWM_MAIN_CC_CHANNEL通道比较中断
使能定时器中断 (定时器中断函数每两次中断会关闭中断一次 -- 一个 PWM周期 )
    return NRF_SUCCESS;
}

  /** 功能描述：设置 PWM 占空比
   *  参数 []:PWM 实例
   *  参数 []:PWM 通道
   *  参数 []:PWM 占空比值
   */
uint32_t app_pwm_channel_duty_set(app_pwm_t const * const p_instance,
                                  uint8_t channel, app_pwm_duty_t duty)
{
  // 获取设置的周期值 按照百分比转换成 TICK
    uint32_t ticks = (nrf_drv_timer_capture_get(p_instance->p_timer,
                     (nrf_timer_cc_channel_t) PWM_MAIN_CC_CHANNEL) * (uint32_t)duty) / 100UL ;
  // 将 TICK 值
    return app_pwm_channel_pulsewidth_ticks_set(p_instance, channel, ticks);
```

```
}


app_pwm_duty_t app_pwm_channel_duty_get(app_pwm_t const * const p_instance, uint8_t channel)
{
    uint32_t value = (nrf_drv_timer_capture_get(p_instance->p_timer,
                     (nrf_timer_cc_channel_t)(channel)) * 100UL )
                   / nrf_drv_timer_capture_get(p_instance->p_timer,
                     (nrf_timer_cc_channel_t) PWM_MAIN_CC_CHANNEL);

    return (app_pwm_duty_t)value;
}


/** 函数描述：初始化 PWM通道
 * 参数 [in]:PWM 实例
 * 参数 [in]: 通道号
 * 参数 [in]:GPIO 脚号
 *
 * 返回值：成功返回 NRF_SUCCESS
 * 返回值：资源不足返回 NRF_ERROR_NO_MEM
 * 返回值：定时器已经使用或者初始化失败返回 NRF_ERROR_INVALID_STATE
 */
static uint32_t app_pwm_channel_init(app_pwm_t const * const p_instance, uint8_t channel,
                                     uint32_t pin, app_pwm_polarity_t polarity)
{
    ASSERT(channel < APP_PWM_CHANNELS_PER_INSTANCE);
    app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
    app_pwm_channel_cb_t * p_channel_cb = &p_cb->channels_cb[channel];

    if (p_cb->state != NRF_DRV_STATE_UNINITIALIZED)
    {
        return NRF_ERROR_INVALID_STATE;
    }

    p_channel_cb->pulsewidth = 0;
    ret_code_t err_code;

    //GPIOTE 初始化
    nrf_drv_gpiote_out_config_t out_cfg = GPIOTE_CONFIG_OUT_TASK_TOGGLE(
POLARITY_INACTIVE(p_instance, channel) );
    err_code = nrf_drv_gpiote_out_init((nrf_drv_gpiote_pin_t)pin,&out_cfg);
    if (err_code != NRF_SUCCESS)
    {
        return NRF_ERROR_NO_MEM;
    }
    p_cb->channels_cb[channel].gpio_pin = pin;
    nrf_drv_gpiote_out_task_enable(pin);

    //PPI 初始化
    for (uint8_t i = 0; i < APP_PWM_REQUIRED_PPI_CHANNELS_PER_CHANNEL; ++i)
    {
        if (nrf_drv_ppi_channel_alloc(&p_channel_cb->ppi_channels[i]) != NRF_SUCCESS)
        {
            return NRF_ERROR_NO_MEM; // Resource deallocation is done by callee.
        }
    }

    nrf_drv_ppi_channel_disable(p_channel_cb->ppi_channels[0]);
    nrf_drv_ppi_channel_disable(p_channel_cb->ppi_channels[1]);
    nrf_drv_ppi_channel_assign(p_channel_cb->ppi_channels[0],
                               nrf_drv_timer_compare_event_address_get(p_instance->p_timer, channel),
                               nrf_drv_gpiote_out_task_addr_get(p_channel_cb->gpio_pin));

   nrf_drv_ppi_channel_assign(p_channel_cb->ppi_channels[1],
                               nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
PWM_MAIN_CC_CHANNEL),
                               nrf_drv_gpiote_out_task_addr_get(p_channel_cb->gpio_pin));

    if (polarity == APP_PWM_POLARITY_ACTIVE_HIGH)
    {
        nrf_drv_gpiote_out_task_trigger(p_channel_cb->gpio_pin);
    }

    p_channel_cb->polarity = polarity;
    p_channel_cb->initialized = APP_PWM_CHANNEL_INITIALIZED;

    return NRF_SUCCESS;
```

```
}


/**brief Function for calculating target timer frequency, which will allow to set given period length.
 *    函数描述：函数用来计算目标定时器频率,
 *
 *    参数  [in] period_us          单位  us
 *
 *    返回值       时钟频率    .
 */
inline    nrf_timer_frequency_t        pwm_calculate_timer_frequency(uint32_t            period_us)
{
        uint32_t     f     =  (uint32_t)NRF_TIMER_FREQ_16MHz;
        uint32_t     min   =  (uint32_t)NRF_TIMER_FREQ_31250Hz;

        //  以 16M HZ  频率计数    16  位宽度定时器最大计时       us 值为  (1us=16tick 16      位 counter    最大值   =65536 转换成   max_us=65536/16=4096)
        while    ((period_us      >   TIMER_MAX_PULSEWIDTH_US_ON_16M)    &&  (f    <   min))
        {
            period_us      >>=   1;
            ++f;
        }
        return     (nrf_timer_frequency_t)f;
}

/**   功能描述：初始化        PWM模块
 *
 *    输入参数：     [in] p_instance      PWM              实例
 *    输入参数：     [in] p_config        PWM              配置
 *    输入参数：     [in] p_ready_callback               回调函数
 */
ret_code_t          app_pwm_init(app_pwm_t         const    *   const     p_instance,        app_pwm_config_t        const     *   const     p_config,
                                          app_pwm_callback_t          p_ready_callback)
{
        ASSERT(p_instance);
        ASSERT(p_ready_callback);

        if    (!p_config)
        {
            return      NRF_ERROR_INVALID_DATA;
        }

        app_pwm_cb_t     *   p_cb    =  (app_pwm_cb_t      *)p_instance->p_cb;

        if    (p_cb->state         !=    NRF_DRV_STATE_UNINITIALIZED)
        {
            return      NRF_ERROR_INVALID_STATE;
        }

        uint32_t      err_code      =  nrf_drv_ppi_init();
        if    ((err_code       !=    NRF_SUCCESS)   &&  (err_code       !=    MODULE_ALREADY_INITIALIZED))
        {
            return      NRF_ERROR_NO_MEM;
        }

        if    (!nrf_drv_gpiote_is_init())
        {
            err_code      =  nrf_drv_gpiote_init();
            if    (err_code       !=    NRF_SUCCESS)
            {
                return      NRF_ERROR_INTERNAL;
            }
        }

        //  初始化资源状态
        p_cb->ppi_channels[         0]    =  (nrf_ppi_channel_t)UNALLOCATED;
        p_cb->ppi_channels[         1]    =  (nrf_ppi_channel_t)UNALLOCATED;
        p_cb->ppi_group                   =  (nrf_ppi_channel_group_t)UNALLOCATED;

        for    (uint8_t      i   =  0;   i   <  APP_PWM_CHANNELS_PER_INSTANCE;    ++i)
        {
            p_cb->channels_cb[i].initialized                      =  APP_PWM_CHANNEL_UNINITIALIZED;
            p_cb->channels_cb[i].ppi_channels[              0]    =  (nrf_ppi_channel_t)UNALLOCATED;
            p_cb->channels_cb[i].ppi_channels[              1]    =  (nrf_ppi_channel_t)UNALLOCATED;
            p_cb->channels_cb[i].gpio_pin                         =  UNALLOCATED;
        }

        //  分配  PPI  通道和组通道
```

```
    for (uint8_t i = 0; i < APP_PWM_REQUIRED_PPI_CHANNELS_PER_INSTANCE; ++i)
    {
        if (nrf_drv_ppi_channel_alloc(&p_cb->ppi_channels[i]) != NRF_SUCCESS)
        {
            pwm_dealloc(p_instance);
            return NRF_ERROR_NO_MEM;
        }
    }

    if (nrf_drv_ppi_group_alloc(&p_cb->ppi_group) != NRF_SUCCESS)
    {
        pwm_dealloc(p_instance);
        return NRF_ERROR_NO_MEM;
    }

    // 初始化通道
    for (uint8_t i = 0; i < APP_PWM_CHANNELS_PER_INSTANCE; ++i)
    {
        if (p_config->pins[i] != APP_PWM_NOPIN)
        {
            err_code = app_pwm_channel_init(p_instance, i, p_config->pins[i],
p_config->pin_polarity[i]);
            if (err_code != NRF_SUCCESS)
            {
                pwm_dealloc(p_instance);
                return err_code;
            }
            // 注意 现在调用下面的函数是没有意义的 直接返回错误代码 指示 PWM电源未打开 可以注释掉
//          app_pwm_channel_pulsewidth_ticks_set(p_instance, i, 0);
        }
    }

    // 定时器初始化
    nrf_timer_frequency_t timer_freq = pwm_calculate_timer_frequency(p_config->period_us);
    nrf_drv_timer_config_t timer_cfg = {
        .frequency          = timer_freq,
        .mode               = NRF_TIMER_MODE_TIMER,
        .bit_width          = NRF_TIMER_BIT_WIDTH_16,
        .interrupt_priority = APP_IRQ_PRIORITY_LOW,
        .p_context          = (void *) (uint32_t) p_instance->p_timer->instance_id
    };

    // 传入事件回调函数 pwm_ready_tick
    err_code = nrf_drv_timer_init(p_instance->p_timer, &timer_cfg, pwm_ready_tick);
    if (err_code != NRF_SUCCESS)
    {
        pwm_dealloc(p_instance);
        return err_code;
    }

    // 这个地方调用库函数将 us 数转换成 TICK 库没理解他的意思 我的算法为先计算 1us 的 tick
    //1us 的 tick= 频率 /1000000; 所以要先获取频率值
    uint32_t ticks = nrf_drv_timer_us_to_ticks(p_instance->p_timer, p_config->period_us);
    p_cb->period = ticks;                                   // 脉宽调制周期设置 值为 : 0x9c40
    nrf_drv_timer_clear(p_instance->p_timer);
    nrf_drv_timer_extended_compare(p_instance->p_timer, (nrf_timer_cc_channel_t) PWM_MAIN_CC_CHANNEL,
                                   ticks, NRF_TIMER_SHORT_COMPARE2_CLEAR_MASK, true );
    nrf_drv_timer_compare_int_disable(p_instance->p_timer, PWM_MAIN_CC_CHANNEL);

    p_cb->p_ready_callback = p_ready_callback;              //PWM 回调函数赋给 PWM模块变量
    m_pwm_ready_counter[p_instance->p_timer->instance_id] = 0; // 一个完整的 PWM波需要两次比较事件
初始化为清零状态
    pan73_workaround(p_instance->p_timer->p_reg, true );    // 这个官网手册没有查到
    m_instances[p_instance->p_timer->instance_id] = p_instance;
    p_cb->state = NRF_DRV_STATE_INITIALIZED;

    return NRF_SUCCESS;
}


void app_pwm_enable(app_pwm_t const * const p_instance)
{
    app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;

    ASSERT(p_cb->state != NRF_DRV_STATE_UNINITIALIZED);   // 判断 PWM是否初始化
    nrf_drv_timer_enable(p_instance->p_timer);            // 使能 PWM定时器
    p_cb->state = NRF_DRV_STATE_POWERED_ON;               // 设置 PWM电源状态标识
    return ;
```

```
}


    void    app_pwm_disable(app_pwm_t          const     *   const     p_instance)
{
            app_pwm_cb_t      *   p_cb    =  (app_pwm_cb_t       *)p_instance->p_cb;

            ASSERT(p_cb->state          !=    NRF_DRV_STATE_UNINITIALIZED);          //  判断   PWM是否初始化
            nrf_drv_timer_disable(p_instance->p_timer);                              //  除能   PWM定时器
            pwm_irq_disable(p_instance);                                             //PWM 中断除能
            p_cb->state        =   NRF_DRV_STATE_INITIALIZED;                        //  设置   PWM电源状态标识为初始化状态
            return    ;
}


uint32_t           app_pwm_uninit(app_pwm_t           const     *   const     p_instance)
{
            app_pwm_cb_t      *   p_cb    =  (app_pwm_cb_t       *)p_instance->p_cb;

            if    (p_cb->state          ==   NRF_DRV_STATE_POWERED_ON)
            {
                   app_pwm_disable(p_instance);
            }
            else    if    (p_cb->state          ==   NRF_DRV_STATE_UNINITIALIZED)
            {
                   return      NRF_ERROR_INVALID_STATE;
            }
            pan73_workaround(p_instance->p_timer->p_reg,                      false    );
            pwm_dealloc(p_instance);

            p_cb->state        =   NRF_DRV_STATE_UNINITIALIZED;
            return      NRF_SUCCESS;
}

    /*PWM 工作说明
     *
     *  捕获通道    0、 1用于更新       脉宽比例参数至          比较寄存器     1、 2
     *
     *  比较寄存器     0、 1用于控制脉宽         比较寄存器      2用于控制周期
     *  比较寄存器     3产生事件触发捕获寄存器            0、 1的的捕获事件      （最终达到更新比较寄存器           0、 1内的值，更新占空比         ）
     *
     *  每次设置脉宽占空比时都会产生一次中断，最后是无需                      CPU干涉，由    GPIOTE 、 PPI 和 timer   的事件和任务
     *  配合产生    PWM波
     *
     *  关键点：    timer   的捕获事件会自动复制定时器             counter   寄存器值到相应的比较匹配寄存器中，间接更新咯                    CC寄存器值
     *
     *PPI   通道  0、 1赋值打开    / 关闭通道组（      2,3,4,5    ），配合定时器比较寄存器             3调节脉宽
     *
     *PPI   通道  2、 3为 PWM0通道（通道     3控制周期，通道       2控制脉宽）
     *                      |----                                            由 PWM组通道控制使能       / 除能
     *PPI   通道  4、 5位 PWM1通道（通道     5控制周期，通道       4控制脉宽）     /
     *
     */


```