

```
1  #include "app_pwm.h"
2  #include "nrf_drv_timer.h"
3  #include "nrf_drv_ppi.h"
4  #include "nrf_drv_common.h"
5  #include "nrf_drv_gpiote.h"
6  #include "nrf_gpiote.h"
7  #include "nrf_gpio.h"
8  #include "app_util.h"
9  #include "app_util_platform.h"
10 #include "nrf_assert.h"
11 #include <stdio.h>
12
13
14 #define APP_PWM_CHANNEL_INITIALIZED 1
15 #define APP_PWM_CHANNEL_UNINITIALIZED 0
16
17 #define APP_PWM_CHANNEL_ENABLED 1
18 #define APP_PWM_CHANNEL_DISABLED 0
19
20 #define TIMER_PRESCALER_MAX 9
21 #define TIMER_MAX_PULSEWIDTH_US_ON_16M 4095
22
23 #define APP_PWM_REQUIRED_PPI_CHANNELS_PER_INSTANCE 2
24 #define APP_PWM_REQUIRED_PPI_CHANNELS_PER_CHANNEL 2
25
26 #define UNALLOCATED 0xFFFFFFFFUL
27
28 #define PWM_MAIN_CC_CHANNEL 2
29 #define PWM_SECONDARY_CC_CHANNEL 3
30
31 volatile uint8_t m_pwm_ready_counter[TIMER_COUNT];
32
33 typedef struct
34 {
35     uint32_t gpio_pin;
36     uint32_t pulsewidth;
37     nrf_ppi_channel_t ppi_channels[ 2];
38     app_pwm_polarity_t polarity;
39     uint8_t initialized;
40 } app_pwm_channel_cb_t;
41
42 typedef struct
43 {
44     app_pwm_channel_cb_t channels_cb[APP_PWM_CHANNELS_PER_INSTANCE];
45     uint32_t period;
46     app_pwm_callback_t p_ready_callback;
47     nrf_ppi_channel_t ppi_channels[ 2];
48     nrf_ppi_channel_group_t ppi_group;
49     nrf_drv_state_t state;
50 } app_pwm_cb_t;
51
52 static const app_pwm_t * m_instances[TIMER_COUNT];
53
54 // 获取给定实例或者通道极性的宏
55 #define POLARITY_ACTIVE(INST,CH) (( ((app_pwm_cb_t*)(INST)->p_cb)->channels_cb[(CH)].polarity ==
56     APP_PWM_POLARITY_ACTIVE_LOW)?( 0):( 1 ))
57 #define POLARITY_INACTIVE(INST,CH) (( ((app_pwm_cb_t*)(INST)->p_cb)->channels_cb[(CH)].polarity ==
58     APP_PWM_POLARITY_ACTIVE_LOW)?( 1):( 0 ))
59
60 /** @brief Workaround for PAN-73.
61  *
62  * 参数 [in] timer 定时器
63  * 参数 [in] enable 使能或者除能
64  */
65 static void pan73_workaround(NRF_TIMER_Type * p_timer, bool enable)
66 {
67     if (p_timer == NRF_TIMER0)
68     {
69         *(uint32_t *) 0x40008C0C = (enable ? 1 : 0);
70     }
71     else if (p_timer == NRF_TIMER1)
72     {
73         *(uint32_t *) 0x40009C0C = (enable ? 1 : 0);
74     }
75     else if (p_timer == NRF_TIMER2)
76     {
77         *(uint32_t *) 0x4000AC0C = (enable ? 1 : 0);
78     }
```

```
79     return    ;
80 }
81
82
83 /** 函数描述：使能指定      PWM实例  IRQ
84  *
85  * 参数 [in] p_instance PWM      实例
86  */
87 __STATIC_INLINE void pwm_irq_enable(app_pwm_t const * const p_instance)
88 {
89     nrf_drv_timer_compare_int_enable(p_instance->p_timer, PWM_MAIN_CC_CHANNEL);
90 }
91
92
93 /** 函数描述：除能指定      PWM实例  IRQ 中断使能标识
94  *
95  * 参数 [in] p_instance PWM      实例
96  */
97 __STATIC_INLINE void pwm_irq_disable(app_pwm_t const * const p_instance)
98 {
99     nrf_drv_timer_compare_int_disable(p_instance->p_timer, PWM_MAIN_CC_CHANNEL);
100 }
101
102
103 /** 函数描述：除能      PWM通道 PPI
104  *
105  * 参数 [in] p_instance PWM      实例
106  */
107 __STATIC_INLINE void pwm_channel_ppi_disable(app_pwm_t const * const p_instance, uint8_t channel)
108 {
109     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
110
111     nrf_drv_ppi_channel_disable(p_cb->channels_cb[channel].ppi_channels[0]);
112     nrf_drv_ppi_channel_disable(p_cb->channels_cb[channel].ppi_channels[1]);
113 }
114
115
116 /** 函数描述：除能      PWM PPI
117  *
118  * 参数 [in] p_instance PWM      实例
119  */
120 __STATIC_INLINE void pwm_ppi_disable(app_pwm_t const * const p_instance)
121 {
122     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
123
124     nrf_drv_ppi_channel_disable(p_cb->ppi_channels[0]);
125     nrf_drv_ppi_channel_disable(p_cb->ppi_channels[1]);
126 }
127
128
129 /** 函数描述：在      duty 设置后，这个函数将会被中断函数调用
130  *
131  * 参数 [in] timer      PWM      模块使用的定时器
132  * 参数 [in] timer_instance_id      定时器实例
133  */
134 void pwm_ready_tick(nrf_timer_event_t event_type, void * p_context)
135 {
136     uint32_t timer_instance_id = (uint32_t)p_context;
137
138     if (m_pwm_ready_counter[timer_instance_id])
139     {
140         --m_pwm_ready_counter[timer_instance_id];
141         if (m_pwm_ready_counter[timer_instance_id])
142         {
143             return ;
144         }
145
146         // 禁止定时器中断
147         nrf_drv_timer_compare_int_disable(m_instances[timer_instance_id]->p_timer, PWM_MAIN_CC_CHANNEL);
148         app_pwm_cb_t * p_cb = (app_pwm_cb_t *)m_instances[timer_instance_id]->p_cb;
149
150         // 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
151         p_cb->p_ready_callback(timer_instance_id);
152     }
153 }
154
155
```

```
156  /** 函数描述：资源重新分配
157  *
158  * 参数 [in] p_instance    PWM          实例
159  */
160  static void pwm_dealloc(app_pwm_t const * const p_instance)
161  {
162      app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
163
164      for (uint8_t i = 0; i < APP_PWM_REQUIRED_PPI_CHANNELS_PER_INSTANCE; ++i)
165      {
166          if (p_cb->ppi_channels[i] != (nrf_ppi_channel_t)(uint8_t)(UNALLOCATED))
167          {
168              nrf_drv_ppi_channel_free(p_cb->ppi_channels[i]);
169          }
170      }
171      if (p_cb->ppi_group != (nrf_ppi_channel_group_t)UNALLOCATED)
172      {
173          nrf_drv_ppi_group_free(p_cb->ppi_group);
174      }
175
176      for (uint8_t ch = 0; ch < APP_PWM_CHANNELS_PER_INSTANCE; ++ch)
177      {
178          for (uint8_t i = 0; i < APP_PWM_REQUIRED_PPI_CHANNELS_PER_CHANNEL; ++i)
179          {
180              if (p_cb->channels_cb[ch].ppi_channels[i] != (nrf_ppi_channel_t)UNALLOCATED)
181              {
182                  nrf_drv_ppi_channel_free(p_cb->channels_cb[ch].ppi_channels[i]);
183                  p_cb->channels_cb[ch].ppi_channels[i] = (nrf_ppi_channel_t)UNALLOCATED;
184              }
185          }
186          if (p_cb->channels_cb[ch].gpio_pin != UNALLOCATED)
187          {
188              nrf_drv_gpiote_out_uninit(p_cb->channels_cb[ch].gpio_pin);
189              p_cb->channels_cb[ch].gpio_pin = UNALLOCATED;
190          }
191          p_cb->channels_cb[ch].initialized = APP_PWM_CHANNEL_UNINITIALIZED;
192      }
193      nrf_drv_timer_uninit(p_instance->p_timer);
194      return ;
195  }
196
197
198  /**PWM 状态转换 从 0%~100% 转换到 %0或者 100%
199  *
200  * 参数 [in] p_instance    PWM          实例
201  * 参数 [in] channel      PWM          通道号
202  * 参数 [in] ticks        时钟数
203  */
204  static void pwm_transition_n_to_0or100(app_pwm_t const * const p_instance,
205                                          uint8_t channel, uint16_t ticks)
206  {
207      app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
208      app_pwm_channel_cb_t * p_ch_cb = &p_cb->channels_cb[channel];
209      nrf_ppi_channel_group_t * p_ppigrp = &p_cb->ppi_group;
210
211      pwm_ppi_disable(p_instance); // 除能 PWM
212
213      nrf_drv_ppi_group_clear(*p_ppigrp); // 清组通道
214      nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[0], *p_ppigrp);
215      // 将 channel[0] 加入组通道
216      nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[1], *p_ppigrp);
217      // 将 channel[1] 加入组通道
218
219      if (!ticks)
220      // 设置时钟数为 0
221      {
222          nrf_drv_ppi_channel_assign(p_cb->ppi_channels[0],
223                                     nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
224                                     channel), // 通道 channel 比较事件
225                                     nrf_drv_ppi_task_addr_group_disable_get(*p_ppigrp));
226          // 获取组通道除能寄存器地址
227      }
228      else
229      // 设置时钟数非 0
230      {
231          ticks = p_cb->period;
232          nrf_drv_ppi_channel_assign(p_cb->ppi_channels[0],
233                                     PWM_MAIN_CC_CHANNEL,
```

```
227         nrf_drv_ppi_task_addr_group_disable_get(*p_ppigrp));
228     // 获取组通道除能寄存器地址
229     }
230     nrf_drv_ppi_channel_enable(p_cb->ppi_channels[0]);
231     // 使能 PPI 通道
232     p_ch_cb->pulsewidth = ticks;
233     // 将 tick 赋值给 PWM脉宽
234 }
235
236 /**PWM 状态转换 从 0%~100% 转换到 %0~100%
237 *
238 * 参数 [in] p_instance PWM 实例
239 * 参数 [in] channel PWM 通道号
240 * 参数 [in] ticks 时钟数
241 */
242 static void pwm_transition_n_to_m(app_pwm_t const * const p_instance,
243                                   uint8_t channel, uint16_t ticks)
244 {
245     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
246     app_pwm_channel_cb_t * p_ch_cb = &p_cb->channels_cb[channel];
247     nrf_ppi_channel_group_t * p_ppigrp = &p_cb->ppi_group;
248
249     nrf_drv_ppi_group_clear(*p_ppigrp);
250     nrf_drv_ppi_channel_include_in_group(p_cb->ppi_channels[0], *p_ppigrp);
251     nrf_drv_ppi_channel_include_in_group(p_cb->ppi_channels[1], *p_ppigrp);
252     nrf_drv_ppi_group_disable(*p_ppigrp);
253
254     /* 注释说明
255     *
256     * 这个函数主要是更新脉宽值 通过 PWM_SECONDARY_CC_CHANNE通道事件 触发捕获事件
257     * 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
258     * 达到脉宽更新的效果
259     * 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，导致半天没看懂）
260     */
261     nrf_drv_ppi_channel_assign(p_cb->ppi_channels[0],
262                               nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
263                               PWM_SECONDARY_CC_CHANNEL),
264                               (uint32_t)&p_instance->p_timer->p_reg->TASKS_CAPTURE[channel]);
265
266     if (ticks < p_cb->channels_cb[channel].pulsewidth)
267     {
268         // For lower value we need one more transition.
269         // 添加通道 1控制脉宽
270         nrf_drv_ppi_channel_assign(p_cb->ppi_channels[1],
271                                   nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
272                                   PWM_SECONDARY_CC_CHANNEL),
273                                   nrf_drv_gpiote_out_task_addr_get(p_ch_cb->gpio_pin));
274     }
275     else
276     {
277         // 将通道 1移出
278         nrf_drv_ppi_channel_remove_from_group(p_cb->ppi_channels[1], *p_ppigrp);
279     }
280
281     // 注意 这里只使能了组通道
282     nrf_drv_ppi_group_enable(*p_ppigrp);
283     p_ch_cb->pulsewidth = ticks;
284
285     // 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
286     nrf_drv_timer_compare(p_instance->p_timer,
287                           (nrf_timer_cc_channel_t) PWM_SECONDARY_CC_CHANNEL,
288                           ticks, false );
289 }
290
291 /**PWM 状态转换 从 0%或者 100% 转换到其它值 n
292 *
293 * 参数 [in] p_instance PWM 实例
294 * 参数 [in] channel PWM 通道号
295 * 参数 [in] ticks 时钟数
296 */
297 static void pwm_transition_0or100_to_n(app_pwm_t const * const p_instance,
298                                         uint8_t channel, uint16_t ticks)
299 {
300     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
301     app_pwm_channel_cb_t * p_ch_cb = &p_cb->channels_cb[channel];
302     nrf_ppi_channel_group_t * p_ppigrp = &p_cb->ppi_group;
```

```
299     nrf_timer_cc_channel_t      pwm_ch_cc  = (nrf_timer_cc_channel_t)(channel);
300
301     pwm_ppi_disable(p_instance);
302     pwm_channel_ppi_disable(p_instance,      channel);
303
304     nrf_drv_timer_compare(p_instance->p_timer,      pwm_ch_cc,      ticks,      false  );
305     nrf_drv_ppi_group_clear(*p_ppigrp);
306     nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[      0],      *p_ppigrp);
307     nrf_drv_ppi_channel_include_in_group(p_ch_cb->ppi_channels[      1],      *p_ppigrp);
308
309     if  (!p_ch_cb->pulsewidth)
310     {
311         // 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
312         nrf_drv_ppi_channel_assign(p_cb->ppi_channels[      0],
313 //PPI 通道 0分配事件地址和任务地址
314         nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
315 channel),      // 获取通道  channel  通道比较事件地址
316         nrf_drv_ppi_task_addr_group_enable_get(*p_ppigrp));
317 // 获取组任务使能寄存器地址
318     }
319     else
320     {
321         // 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
322         nrf_drv_ppi_channel_assign(p_cb->ppi_channels[      0],
323         nrf_drv_timer_compare_event_address_get(p_instance->p_timer,      PWM_MAIN_CC_CHANNEL),
324         nrf_drv_ppi_task_addr_group_enable_get(*p_ppigrp));
325     }
326     nrf_drv_ppi_channel_enable(p_cb->ppi_channels[      0]);
327 // 使能 PPI 通道
328     p_ch_cb->pulsewidth      = ticks;
329 // 设置脉宽时钟数
330 }
331
332 /**PWM 状态转换  从 0%或者 100% 转换到  %0或者 100%
333 *
334 * 参数 [in] p_instance  PWM      实例
335 * 参数 [in] channel      PWM      通道号
336 * 参数 [in] ticks      时钟数
337 */
338 static      void      pwm_transition_0or100_to_0or100(app_pwm_t      const      *      const      p_instance,
339      uint8_t      channel,      uint16_t      ticks)
340 {
341     app_pwm_cb_t      *      p_cb      = (app_pwm_cb_t      *)p_instance->p_cb;
342     app_pwm_channel_cb_t      *      p_ch_cb      = &p_cb->channels_cb[channel];
343     nrf_timer_cc_channel_t      pwm_ch_cc      = (nrf_timer_cc_channel_t)(channel);      // 获取通道
344
345     pwm_ppi_disable(p_instance);
346     pwm_channel_ppi_disable(p_instance,      channel);
347
348     if  (!ticks)
349     {
350         // 设置为  0%  引脚强制输出为非激活态
351         nrf_drv_gpiote_out_task_force(p_ch_cb->gpio_pin,      POLARITY_INACTIVE(p_instance,      channel));
352     }
353     else  if  (ticks      >= p_cb->period)
354     {
355         // 设置为  100%  引脚强制输出为激活态
356         ticks      = p_cb->period;      // 将周期值付给  ticks
357         nrf_drv_gpiote_out_task_force(p_ch_cb->gpio_pin,      POLARITY_ACTIVE(p_instance,      channel));
358     }
359
360     nrf_drv_timer_compare(p_instance->p_timer,      pwm_ch_cc,      ticks,      false  );
361     p_ch_cb->pulsewidth      = ticks;      // 将 ticks  赋值给  脉冲宽度
362     return  ;
363 }
364
365 /**Function for setting PWM channel duty cycle in clock ticks.
366 *
367 * 参数 [in] p_instance  PWM      实例
368 * 参数 [in] channel      PWM      通道号
369 * 参数 [in] ticks      时钟数
370 *
371 * 返回值      操作成功返回      NRF_SUCCESS
372 * 返回值      PWM  模块没有准备好返回      NRF_ERROR_BUSY
373 * 返回值      如果给定的实例没有初始化返回      NRF_ERROR_INVALID_STATE
374 *
```

```
372  * 注意事项：
373  * 一个完成的    PWM完整时钟周期结束才能改变占空比
374  * 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
375  */
376 uint32_t      app_pwm_channel_pulsewidth_ticks_set(app_pwm_t      const * const p_instance,
377                                                     uint8_t      channel,      uint16_t      ticks)
378 {
379     app_pwm_cb_t      * p_cb      = (app_pwm_cb_t      *)p_instance->p_cb;
380     app_pwm_channel_cb_t      * p_ch_cb      = &p_cb->channels_cb[channel];
381
382     ASSERT(channel      < APP_PWM_CHANNELS_PER_INSTANCE);
383     ASSERT(p_ch_cb->initialized      == APP_PWM_CHANNEL_INITIALIZED);
384
385     if (p_cb->state      != NRF_DRV_STATE_POWERED_ON)
386     {
387         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
388     }
389     if (ticks      == p_ch_cb->pulsewidth)
390     {
391         p_cb->p_ready_callback(p_instance->p_timer->instance_id);
392         return      NRF_SUCCESS;          // No action required.
393     }
394     首次执行到此处返回 (      ticks  和 pulsewidth  都为 0 )
395     if (m_pwm_ready_counter[p_instance->p_timer->instance_id])
396     {
397         return      NRF_ERROR_BUSY;          //
398         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
399     }
400     m_pwm_ready_counter[p_instance->p_timer->instance_id]      = 2;  // PWM will be ready to next change
401     on 2nd CC0 event (minimum 1 full period)
402
403     // 脉宽改变序列
404     if (!p_ch_cb->pulsewidth      || p_ch_cb->pulsewidth      >= p_cb->period)
405     {
406         // 脉宽为 0%或者 100% 通道处于除能状态
407         if (!ticks      || ticks      >= p_cb->period)
408         {
409             // 设置脉宽为 0%或者 100%
410             pwm_transition_0or100_to_0or100(p_instance,      channel,      ticks);
411         }
412         else
413         {
414             // 设置脉宽在 0%~100%之间 ( 不包含 0%和 100% )
415             pwm_transition_0or100_to_n(p_instance,      channel,      ticks);
416         }
417     }
418     else
419     {
420         // 脉宽在 0%~100%之间 ( 不包含 0%和 100% ) 通道处于使能状态
421         if (!ticks      || ticks      >= p_cb->period)
422         {
423             // 设置脉宽为 0%或者 100% 关闭通道
424             pwm_transition_n_to_0or100(p_instance,      channel,      ticks);
425         }
426         else
427         {
428             // 设置脉宽在 0%~100%之间 ( 不包含 0%和 100% )
429             pwm_transition_n_to_m(p_instance,      channel,      ticks);
430         }
431     }
432     pwm_irq_enable(p_instance);          // 使能定时器 PWM_MAIN_CC_CHANNEL通道比较中断
433     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 )
434     return      NRF_SUCCESS;
435 }
436
437 /** 功能描述：设置 PWM 占空比
438  * 参数 []:PWM 实例
439  * 参数 []:PWM 通道
440  * 参数 []:PWM 占空比值
441  */
442 uint32_t      app_pwm_channel_duty_set(app_pwm_t      const * const p_instance,
443                                       uint8_t      channel,      app_pwm_duty_t      duty)
444 {
445     // 获取设置的周期值      按照百分比转换成      TICK
446     uint32_t      ticks      = (nrf_drv_timer_capture_get(p_instance->p_timer,
447                                                           (nrf_timer_cc_channel_t)      PWM_MAIN_CC_CHANNEL) * (uint32_t)duty)      / 100UL ;
448     // 将 TICK 值
449     return      app_pwm_channel_pulsewidth_ticks_set(p_instance,      channel,      ticks);
450 }
```

```
446 }
447
448
449 app_pwm_duty_t app_pwm_channel_duty_get(app_pwm_t const * const p_instance, uint8_t channel)
450 {
451     uint32_t value = (nrf_drv_timer_capture_get(p_instance->p_timer,
452                                                  (nrf_timer_cc_channel_t)(channel)) * 100UL )
453                     / nrf_drv_timer_capture_get(p_instance->p_timer,
454                                                  (nrf_timer_cc_channel_t) PWM_MAIN_CC_CHANNEL);
455
456     return (app_pwm_duty_t)value;
457 }
458
459
460 /** 函数描述：初始化 PWM通道
461 * 参数 [in]:PWM 实例
462 * 参数 [in]: 通道号
463 * 参数 [in]:GPIO 脚号
464 *
465 * 返回值：成功返回 NRF_SUCCESS
466 * 返回值：资源不足返回 NRF_ERROR_NO_MEM
467 * 返回值：定时器已经使用或者初始化失败返回 NRF_ERROR_INVALID_STATE
468 */
469 static uint32_t app_pwm_channel_init(app_pwm_t const * const p_instance, uint8_t channel,
470                                     uint32_t pin, app_pwm_polarity_t polarity)
471 {
472     ASSERT(channel < APP_PWM_CHANNELS_PER_INSTANCE);
473     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
474     app_pwm_channel_cb_t * p_channel_cb = &p_cb->channels_cb[channel];
475
476     if (p_cb->state != NRF_DRV_STATE_UNINITIALIZED)
477     {
478         return NRF_ERROR_INVALID_STATE;
479     }
480
481     p_channel_cb->pulsewidth = 0;
482     ret_code_t err_code;
483
484     //GPiOTE 初始化
485     nrf_drv_gpiote_out_config_t out_cfg = GPIOTE_CONFIG_OUT_TASK_TOGGLE(
486     POLARITY_INACTIVE(p_instance, channel) );
487     err_code = nrf_drv_gpiote_out_init((nrf_drv_gpiote_pin_t)pin,&out_cfg);
488     if (err_code != NRF_SUCCESS)
489     {
490         return NRF_ERROR_NO_MEM;
491     }
492     p_cb->channels_cb[channel].gpio_pin = pin;
493     nrf_drv_gpiote_out_task_enable(pin);
494
495     //PPI 初始化
496     for (uint8_t i = 0; i < APP_PWM_REQUIRED_PPI_CHANNELS_PER_CHANNEL; ++i)
497     {
498         if (nrf_drv_ppi_channel_alloc(&p_channel_cb->ppi_channels[i]) != NRF_SUCCESS)
499         {
500             return NRF_ERROR_NO_MEM; // Resource deallocation is done by callee.
501         }
502
503         nrf_drv_ppi_channel_disable(p_channel_cb->ppi_channels[0]);
504         nrf_drv_ppi_channel_disable(p_channel_cb->ppi_channels[1]);
505         nrf_drv_ppi_channel_assign(p_channel_cb->ppi_channels[0],
506                                   nrf_drv_timer_compare_event_address_get(p_instance->p_timer, channel),
507                                   nrf_drv_gpiote_out_task_addr_get(p_channel_cb->gpio_pin));
508
509         nrf_drv_ppi_channel_assign(p_channel_cb->ppi_channels[1],
510                                   nrf_drv_timer_compare_event_address_get(p_instance->p_timer,
511                                   PWM_MAIN_CC_CHANNEL),
512                                   nrf_drv_gpiote_out_task_addr_get(p_channel_cb->gpio_pin));
513
514         if (polarity == APP_PWM_POLARITY_ACTIVE_HIGH)
515         {
516             nrf_drv_gpiote_out_task_trigger(p_channel_cb->gpio_pin);
517         }
518
519         p_channel_cb->polarity = polarity;
520         p_channel_cb->initialized = APP_PWM_CHANNEL_INITIALIZED;
521
522         return NRF_SUCCESS;
523     }
```

```
522 }
523
524
525 /**brief Function for calculating target timer frequency, which will allow to set given period length.
526 * 函数描述：函数用来计算目标定时器频率，
527 *
528 * 参数 [in] period_us          单位 us
529 *
530 * 返回值      时钟频率      .
531 */
532 inline      nrf_timer_frequency_t          pwm_calculate_timer_frequency(uint32_t          period_us)
533 {
534     uint32_t      f      = (uint32_t)NRF_TIMER_FREQ_16MHz;
535     uint32_t      min     = (uint32_t)NRF_TIMER_FREQ_31250Hz;
536
537     // 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)
538     while ((period_us      >  TIMER_MAX_PULSEWIDTH_US_ON_16M)    && (f      < min))
539     {
540         period_us      >=> 1;
541         ++f;
542     }
543     return      (nrf_timer_frequency_t)f;
544 }
545
546 /** 功能描述：初始化      PWM模块
547 *
548 * 输入参数：      [in] p_instance      PWM      实例
549 * 输入参数：      [in] p_config      PWM      配置
550 * 输入参数：      [in] p_ready_callback      回调函数
551 */
552 ret_code_t      app_pwm_init(app_pwm_t      const *      const p_instance,      app_pwm_config_t      const *      const p_config,
553                               app_pwm_callback_t      p_ready_callback)
554 {
555     ASSERT(p_instance);
556     ASSERT(p_ready_callback);
557
558     if (!p_config)
559     {
560         return      NRF_ERROR_INVALID_DATA;
561     }
562
563     app_pwm_cb_t      * p_cb      = (app_pwm_cb_t      *)p_instance->p_cb;
564
565     if (p_cb->state      !=  NRF_DRV_STATE_UNINITIALIZED)
566     {
567         return      NRF_ERROR_INVALID_STATE;
568     }
569
570     uint32_t      err_code      = nrf_drv_ppi_init();
571     if ((err_code      !=  NRF_SUCCESS) && (err_code      !=  MODULE_ALREADY_INITIALIZED))
572     {
573         return      NRF_ERROR_NO_MEM;
574     }
575
576     if (!nrf_drv_gpiote_is_init())
577     {
578         err_code      = nrf_drv_gpiote_init();
579         if (err_code      !=  NRF_SUCCESS)
580         {
581             return      NRF_ERROR_INTERNAL;
582         }
583     }
584
585     // 初始化资源状态
586     p_cb->ppi_channels[      0] = (nrf_ppi_channel_t)UNALLOCATED;
587     p_cb->ppi_channels[      1] = (nrf_ppi_channel_t)UNALLOCATED;
588     p_cb->ppi_group      = (nrf_ppi_channel_group_t)UNALLOCATED;
589
590     for (uint8_t      i      = 0; i      < APP_PWM_CHANNELS_PER_INSTANCE; ++i)
591     {
592         p_cb->channels_cb[i].initialized      = APP_PWM_CHANNEL_UNINITIALIZED;
593         p_cb->channels_cb[i].ppi_channels[      0] = (nrf_ppi_channel_t)UNALLOCATED;
594         p_cb->channels_cb[i].ppi_channels[      1] = (nrf_ppi_channel_t)UNALLOCATED;
595         p_cb->channels_cb[i].gpio_pin      = UNALLOCATED;
596     }
597
598     // 分配 PPI 通道和组通道
```



```
599     for (uint8_t i = 0; i < APP_PWM_REQUIRED_PPI_CHANNELS_PER_INSTANCE; ++i)
600     {
601         if (nrf_drv_ppi_channel_alloc(&p_cb->ppi_channels[i]) != NRF_SUCCESS)
602         {
603             pwm_dealloc(p_instance);
604             return NRF_ERROR_NO_MEM;
605         }
606     }
607
608     if (nrf_drv_ppi_group_alloc(&p_cb->ppi_group) != NRF_SUCCESS)
609     {
610         pwm_dealloc(p_instance);
611         return NRF_ERROR_NO_MEM;
612     }
613
614     // 初始化通道
615     for (uint8_t i = 0; i < APP_PWM_CHANNELS_PER_INSTANCE; ++i)
616     {
617         if (p_config->pins[i] != APP_PWM_NOPIN)
618         {
619             err_code = app_pwm_channel_init(p_instance, i, p_config->pins[i],
620             p_config->pin_polarity[i]);
621             if (err_code != NRF_SUCCESS)
622             {
623                 pwm_dealloc(p_instance);
624                 return err_code;
625             }
626             // 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
627             // app_pwm_channel_pulsewidth_ticks_set(p_instance, i, 0);
628         }
629     }
630
631     // 定时器初始化
632     nrf_timer_frequency_t timer_freq = pwm_calculate_timer_frequency(p_config->period_us);
633     nrf_drv_timer_config_t timer_cfg = {
634         .frequency = timer_freq,
635         .mode = NRF_TIMER_MODE_TIMER,
636         .bit_width = NRF_TIMER_BIT_WIDTH_16,
637         .interrupt_priority = APP_IRQ_PRIORITY_LOW,
638         .p_context = (void *) (uint32_t) p_instance->p_timer->instance_id
639     };
640
641     // 传入事件回调函数 pwm_ready_tick
642     err_code = nrf_drv_timer_init(p_instance->p_timer, &timer_cfg, pwm_ready_tick);
643     if (err_code != NRF_SUCCESS)
644     {
645         pwm_dealloc(p_instance);
646         return err_code;
647     }
648
649     // 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
650     // 1us 的 tick= 频率 /1000000; 所以要先获取频率值
651     uint32_t ticks = nrf_drv_timer_us_to_ticks(p_instance->p_timer, p_config->period_us);
652     p_cb->period = ticks; // 脉宽调制周期设置 值为 0x9c40
653     nrf_drv_timer_clear(p_instance->p_timer);
654     nrf_drv_timer_extended_compare(p_instance->p_timer, (nrf_timer_cc_channel_t) PWM_MAIN_CC_CHANNEL,
655     ticks, NRF_TIMER_SHORT_COMPARE2_CLEAR_MASK, true );
656     nrf_drv_timer_compare_int_disable(p_instance->p_timer, PWM_MAIN_CC_CHANNEL);
657
658     p_cb->p_ready_callback = p_ready_callback; //PWM 回调函数赋给 PWM模块变量
659     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
660     // 初始化为清零状态
661     pan73_workaround(p_instance->p_timer->p_reg, true ); // 这个官网手册没有查到
662     m_instances[p_instance->p_timer->instance_id] = p_instance;
663     p_cb->state = NRF_DRV_STATE_INITIALIZED;
664
665     return NRF_SUCCESS;
666 }
667
668 void app_pwm_enable(app_pwm_t const * const p_instance)
669 {
670     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
671
672     ASSERT(p_cb->state != NRF_DRV_STATE_UNINITIALIZED); // 判断 PWM是否初始化
673     nrf_drv_timer_enable(p_instance->p_timer); // 使能 PWM定时器
674     p_cb->state = NRF_DRV_STATE_POWERED_ON; // 设置 PWM电源状态标识
675     return ;
```

```
675 }
676
677
678 void app_pwm_disable(app_pwm_t const * const p_instance)
679 {
680     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
681
682     ASSERT(p_cb->state != NRF_DRV_STATE_UNINITIALIZED); // 判断 PWM是否初始化
683     nrf_drv_timer_disable(p_instance->p_timer); // 除能 PWM定时器
684     pwm_irq_disable(p_instance); //PWM 中断除能
685     p_cb->state = NRF_DRV_STATE_INITIALIZED; // 设置 PWM电源状态标识为初始化状态
686     return ;
687 }
688
689
690 uint32_t app_pwm_uninit(app_pwm_t const * const p_instance)
691 {
692     app_pwm_cb_t * p_cb = (app_pwm_cb_t *)p_instance->p_cb;
693
694     if (p_cb->state == NRF_DRV_STATE_POWERED_ON)
695     {
696         app_pwm_disable(p_instance);
697     }
698     else if (p_cb->state == NRF_DRV_STATE_UNINITIALIZED)
699     {
700         return NRF_ERROR_INVALID_STATE;
701     }
702     pan73_workaround(p_instance->p_timer->p_reg, false );
703     pwm_dealloc(p_instance);
704
705     p_cb->state = NRF_DRV_STATE_UNINITIALIZED;
706     return NRF_SUCCESS;
707 }
708
709 /*PWM 工作说明
710 *
711 * 捕获通道 0、 1用于更新 脉宽比例参数至 比较寄存器 1、 2
712 *
713 * 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
714 * 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 ）
715 *
716 * 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
717 * 配合产生 PWM波
718 *
719 * 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
720 *
721 *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
722 *
723 *PPI 通道 2、 3为 PWM0通道（通道 3控制周期，通道 2控制脉宽）
724 * |--- 由 PWM0通道控制使能 / 除能
725 *PPI 通道 4、 5位 PWM1通道（通道 5控制周期，通道 4控制脉宽） /
726 *
727 */
728
729
```