青风带你玩蓝牙 nRF51822 系列教程

----------作者：青风

出品论坛：　www.qfv8.com　青风电子社区

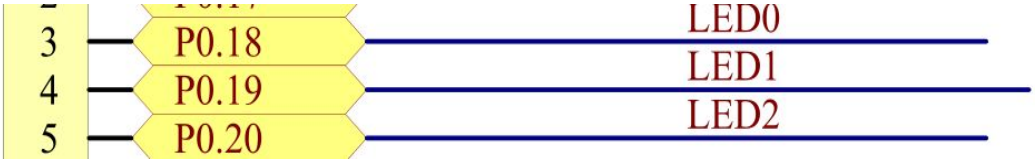
作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：346518370
硬件平台：青云 QY-nRF51822 开发板

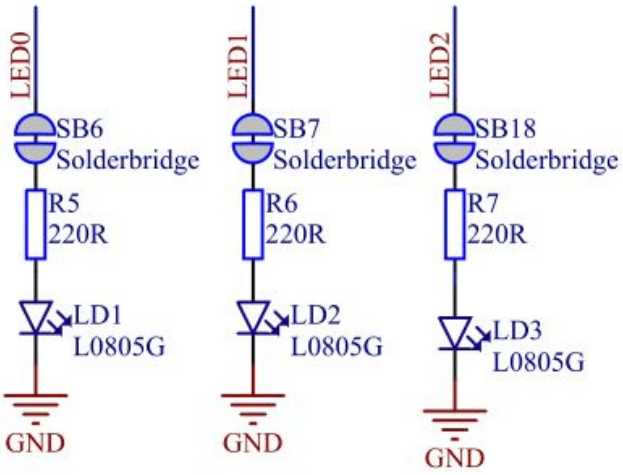
2.9 PPI 模块的使用

nRF51822 是 cortex m0 内核，内部设置了 PPI 方式，PPI 和 DMA 功能有些类似，也是用于不同外设之间进行互连，而不需要 CPU 进行参与。PPI 主要的连接对象是任务和事件。下面将详细进行讨论：

2.9.1 硬件准备：

本例要用到的外设为定时器，通过定时器 1 和定时器 2 来控制器定时器 0，而通过定时器 0 来控制 LED 灯的亮灭。因此要使用的外设就是 3 个 LED 灯：



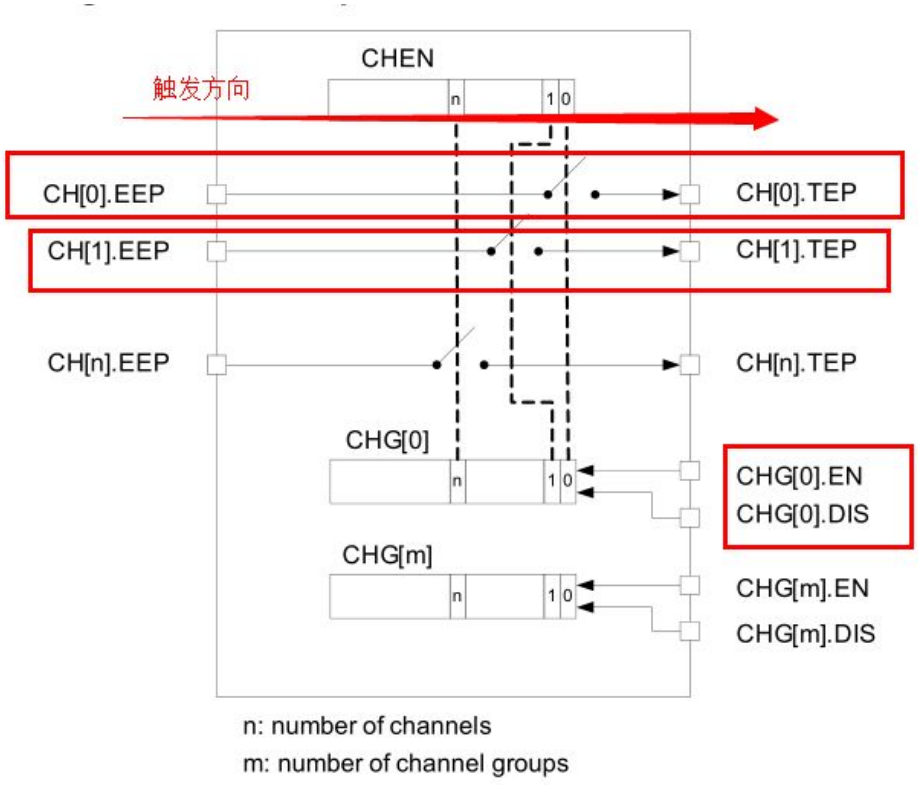


2.9.2 软件准备：

在代码文件中，实验 9 建立了一个演示历程，我们还是采用分层思想，通过官方提供的库文件。打开 user 文件夹中的工程：

如上图所示：我们只需要编写 **time.c** 驱动和 **ppi.c** 驱动，主函数 **main.c** 的三个文件就 **OK** 了,现在我们就来讨论下如何编写 **ppi.c** 在这个驱动子文件中的内容。

首先我们来了解一下 **PPI** 的原理，首先看一下其结构图：



其结构还是非常简单的。如图所示，PPI 实际上提供了一种直连的机制，这种机制可以把一个外设发生的事件（event）来触发另一个外设的任务（task），整个过程不需要 CPU 进行参与。

因此一个任务（task）通过 PPI 通道和事件（event）进行互连。PPI 通道由两个终点寄存器组成，分别为：事件终点寄存器 (EEP) 和任务终点寄存器 (TEP)。

可以把外设任务（task）通过任务寄存器的地址与任务终点寄存器(TEP)进行赋值。同理，也可以把外设事件通过事件（event）寄存器的地址与事件终点寄存器(EEP) 进行赋值 。

按照上面的分享，我们来配置 PPI，设置代码如下：

```
01.    void ppi_init(void)
02. {
03.      // 通道 0 的 EFP 和 TEP 设置
04.     //把定时器 1 的比较事件作为事件，定时器 0 的停止作为任务
05.     //通过定时器 1 比较事件来触发定时器 0 停止
06.      NRF_PPI->CH[0].EEP = (uint32_t)(&NRF_TIMER1->EVENTS_COMPARE[0]);
07.      NRF_PPI->CH[0].TEP = (uint32_t)(&NRF_TIMER0->TASKS_STOP);
08.
09.      // 通道 1 的 EFP 和 TEP 设置
10.      //把定时器 2 的比较事件作为事件，定时器 0 的开始作为任务
11.     //通过定时器 2 比较事件来触发定时器 0 开始
12.      NRF_PPI->CH[1].EEP = (uint32_t)(&NRF_TIMER2->EVENTS_COMPARE[0]);
13.      NRF_PPI->CH[1].TEP = (uint32_t)(&NRF_TIMER0->TASKS_START);
14.
15.      // 使能 PPI 通道 1 和通道 0
16.      NRF_PPI->CHEN = (PPI_CHEN_CH0_Enabled << PPI_CHEN_CH0_Pos) |
    (PPI_CHEN_CH1_Enabled << PPI_CHEN_CH1_Pos);
17. }
```

这个代码实际上完成了 2 个事情：

有两种方式来使能或者关掉 PPI 频道:通过定时器 2 比较事件来触发定时器 0 开始计数。通过定时器 1 比较事件来触发定时器 0 停止计数。整个过程不需要 CPU 参与，和中断触发类型有点相似。

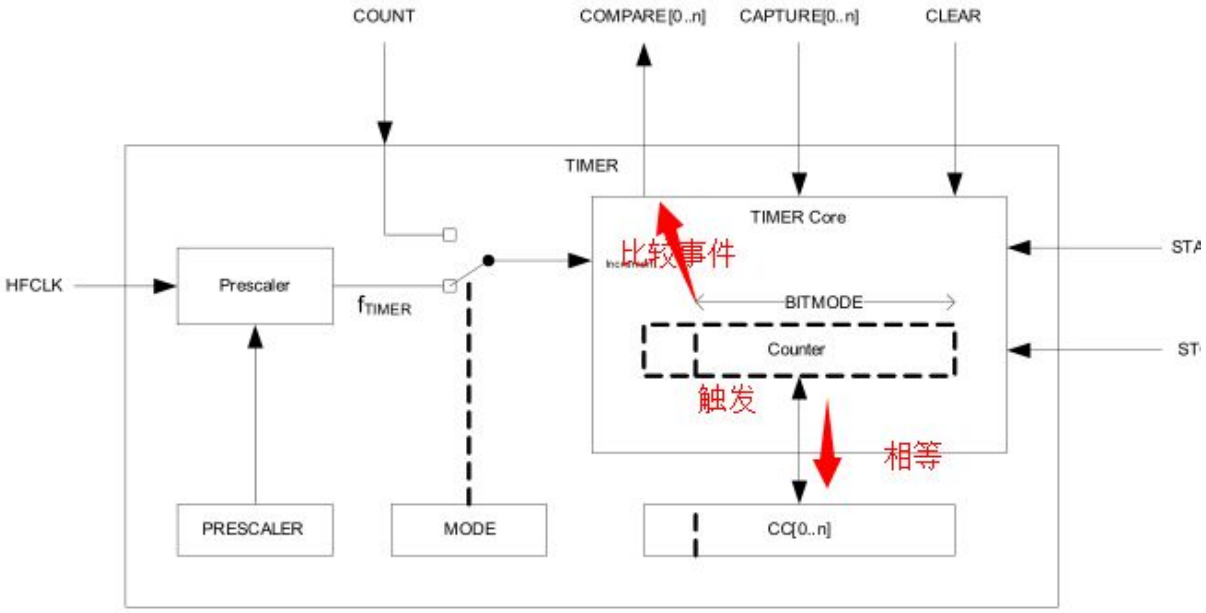
最后是使能通道，实际上开通道和关通道有两种方式：

•方法 1：通过独立设置 CHEN, CHENSET, and CHENCLR 寄存器。

•方法 2 ：通过 PPI 频道组的使能和关断任务。

下面来继续讨论 TIME 定时器的配置，这里使用了三个定时器 time0,time1 和 time2。上面设置 PPI 的启动时，谈到了定时器的比较事件，这个是我们分析的重点。

如下图定时器结构图，定时器的比较事件的发生是当定时器计数器 counter 和捕获比较寄存器 CC 相等的时候，这时就可以触发比较事件，那么我们就需要在定时器 1 和定时器 2 的 CC 寄存器中预设值，当定时器计数器计数到预设值的时候就启动比较事件。



分析到这里，我们下面来设置代码：

```
18.  void timer0_init(void)
19. {
20.      NRF_TIMER0->MODE      = TIMER_MODE_MODE_Counter;  // 设置定时器位计数模式.
21.      NRF_TIMER0->BITMODE = TIMER_BITMODE_BITMODE_24Bit;  // 24-bit 模式.
22. }
23.
```

定时器 0 的设置比较简单了设置为普通的计数模式，依次计数。BITMODE 设置为 24bit.定时器 1 和定时器 2 的设置类似，区别就是为了错开比较事件的时间，也就是通过 PPI 设置的定时器 0 的打开计数和关闭时间，因此 CC 寄存器的预设值有区别：

```
24.   void timer1_init(void)
25. {
26.      // 配置定时器每 2 seconds 溢出.
27.      // SysClk = 16 Mhz
28.      // BITMODE = 16 bit
29.      // PRESCALER = 9
30.      // The overflow occurs every 0xFFFF/(SysClk/2^PRESCALER).
31.      // = 65535/31250 = 2.097 sec
32.      NRF_TIMER1->BITMODE       = (TIMER_BITMODE_BITMODE_16Bit <<
    TIMER_BITMODE_BITMODE_Pos);
33.      NRF_TIMER1->PRESCALER     = 9;
34.      NRF_TIMER1->SHORTS        = (TIMER_SHORTS_COMPARE0_CLEAR_Enabled <<
    TIMER_SHORTS_COMPARE0_CLEAR_Pos);
35.
36.      // Trigger interrupt for compare[0] event.
37.      NRF_TIMER1->MODE          = TIMER_MODE_MODE_Timer;
38.      NRF_TIMER1->CC[0]         = 0xFFFFUL;  // Match at even number of seconds
```

```
39. }
40.
41.     void timer2_init(void)
42. {
43.     // Generate interrupt/event when half of time before the timer overflows has past, that is at
    1,3,5,7... seconds from start.
44.     // SysClk = 16Mhz
45.     // BITMODE = 16 bit
46.     // PRESCALER = 9
47.     // now the overflow occurs every 0xFFFF/(SysClk/2^PRESCALER)
48.     // = 65535/31250 = 2.097 sec */
49.     NRF_TIMER2->BITMODE       = (TIMER_BITMODE_BITMODE_16Bit <<
    TIMER_BITMODE_BITMODE_Pos);
50.     NRF_TIMER2->PRESCALER     = 9;
51.     NRF_TIMER2->SHORTS        = (TIMER_SHORTS_COMPARE0_CLEAR_Enabled <<
    TIMER_SHORTS_COMPARE0_CLEAR_Pos);
52.
53.     // Trigger interrupt for compare[0] event.
54.     NRF_TIMER2->MODE          = TIMER_MODE_MODE_Timer;
55.     NRF_TIMER2->CC[0]         = 0x7FFFUL;   // Match at odd number of seconds.
56. }
57.
```

定时器 2 的 CC 寄存器预设值设为了 0x7FFF，通过之前所讲的定时器设置，这个比较触发的时间大概是 1s，定时器 1 的 CC 寄存器预设值设为了 0xFFFF，这个比较触发的时间大概是 2s，因此定时器 0 的开始任务与结束任务时间大概是 1S。

那么主函数就是十分的简单了，直接调用我们写好的驱动函数，初始化 PPI,定时器 1，定时器 2，定时器 0，然后打开定时器 1 和定时器 2 等待触发比较事件的发生，当定时器 0 被触发后，把捕获寄存器内的值给定时器 0 的 CC 寄存器，来控制 LED 的亮灭：

```
///******************** (C) COPYRIGHT 2014 青风电子 ********************
 * 文件名   ：main
 * 描述     ：
 * 实验平台：青云 nRF5188 开发板
 * 描述     ：串口中断输出
 * 作者     ：青风
 * 店铺     ：qfv5.taobao.com
*********************************************************************************/
#include "nrf51.h"
#include   "led.h"
#include   "time.h"
#include   "ppi.h"
#include "nrf_delay.h"
```

```
#include "nrf_gpio.h"

int main(void)
{
    timer0_init();   // Timer0 用于 LED 灯的亮灭
    timer1_init();   // Timer1  定时器产生的偶数秒的事件
    timer2_init();   // Timer2 定时器产生的奇数秒事件
    ppi_init();      // PPI 事件重定向到计时器启动/停止任务

    NRF_POWER->TASKS_CONSTLAT = 1;//启用恒定延迟模式

    // Start clock.
    NRF_TIMER1->TASKS_START = 1;//开定时器 1
    NRF_TIMER2->TASKS_START = 1;//开定时器 2

    nrf_gpio_range_cfg_output(LED_START, LED_STOP);//设置 LED 灯的范围

    while (1)
    {
        NRF_TIMER0->TASKS_COUNT          = 1;//设置定时器为递增计数模式
        NRF_TIMER0->TASKS_CAPTURE[0]     = 1;//捕捉定时器值寄存器 CC0
        nrf_gpio_port_write(NRF_GPIO_PORT_SELECT_PORT2, (uint8_t)NRF_TIMER0->CC[0]);//
用 CCO 的值来通知 LED 灯
        nrf_delay_ms(100);
    }
}
```

实验下载到青云 nRF51822 开发板后 led 灯以二进制计数方式闪亮循环，亮点顺序如下：

000-->100--->010-->110-->001-->101-->111-->000