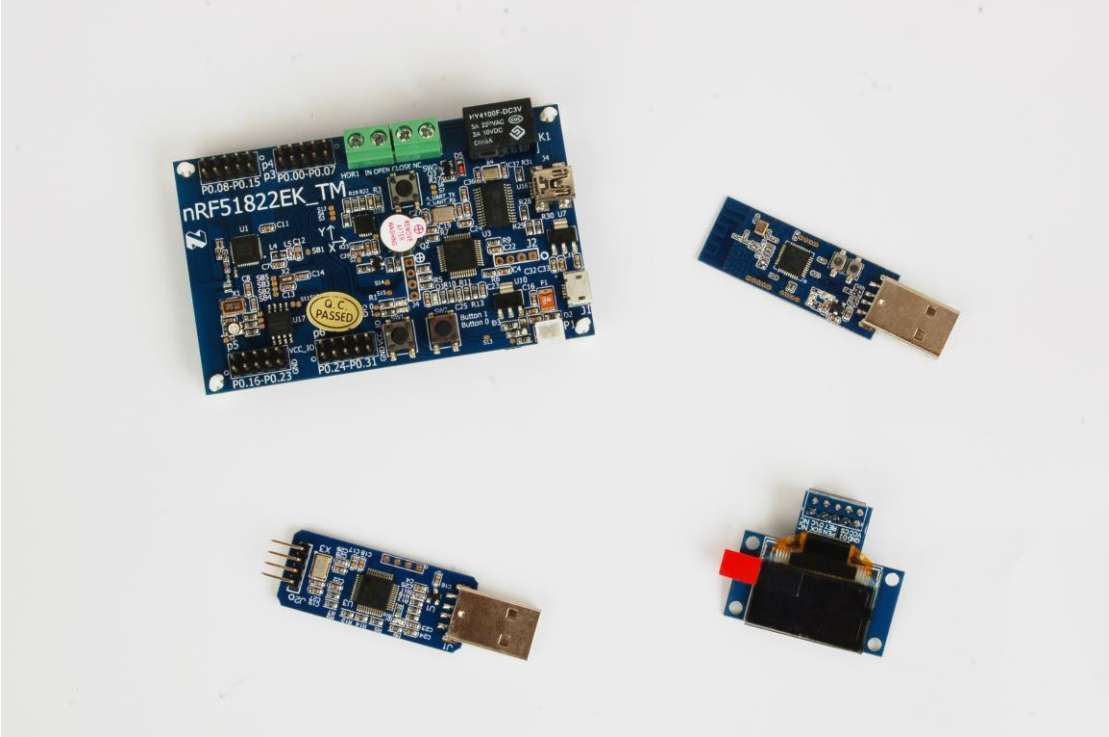
讯联电子nRF51822蓝牙4.0开发实战

PWM

V:1.0

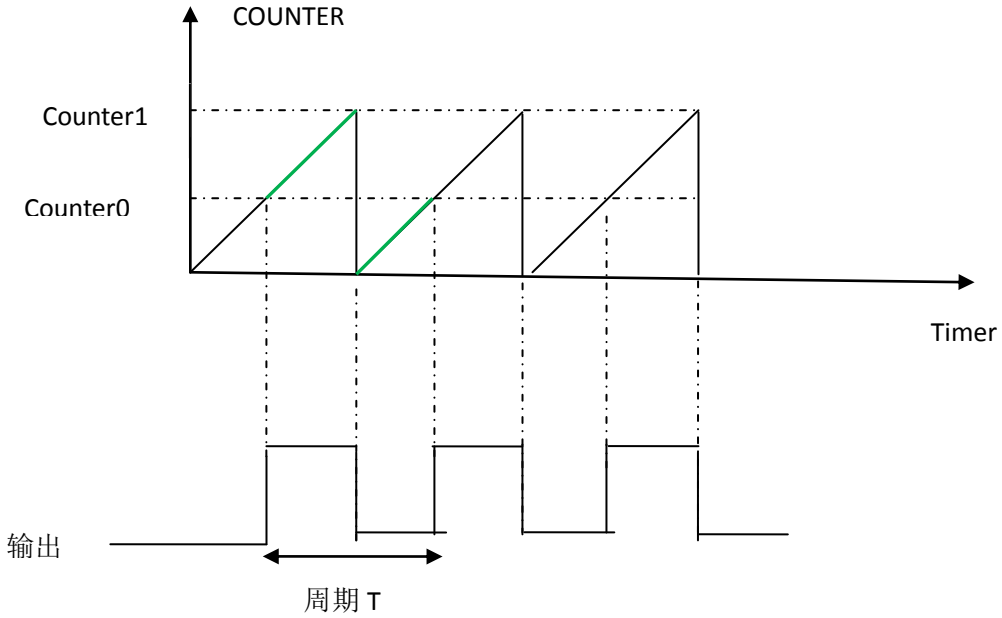
作者：不离不弃　qq 574912883

申明：本教程版权归讯联电子所有。本教程仅供内部客户交流之用。如需引用，请注明出处。由于工程师水平有限，文档难免有所疏漏和错误，由此造成的损失，讯联电子不承担任何责任。



作者：不离不弃　qq 574912883

先简单介绍一下 PWM 的原理。



原理很简单。 假设 COUNTER 是个从 0 开始递增的计数器。 我们设置两个值 counter0 和 counter1 在 COUNTER 计数到 counter0 的值时候翻转输出的电平，然后 COUNTER 继续计数，在计数到 counter1 的值的时候再翻转输出电平。 同时清零 COUNTER 计数器。让其从 0 开始重新计数，这样就可以产生一个方波。

从上面的图可以看出这个方波的一个周期 T 的时间是由 counter1 来决定的。所以周期的调节就是通过 counter1 的值来调节。 而 counter0 的值则影响着方波的占空比。

综上，PWM 的实现就是通过调节 counter1 和 counter0 的两个值来实现周期和占空比可调。

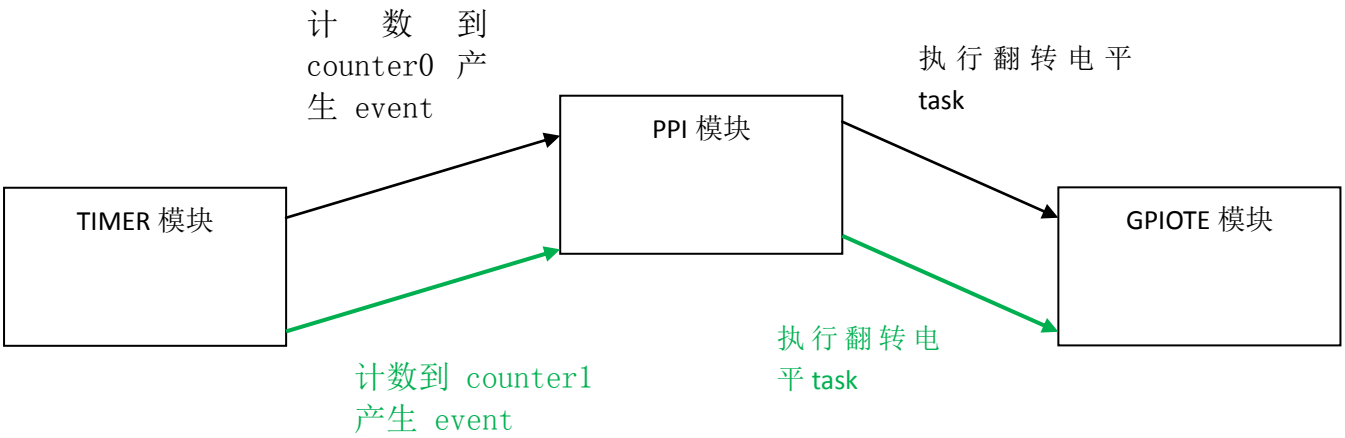
51822 硬件没有 PWM 模块，所以如果需要使用 PWM，从上面的原理介绍可以知道使用 timer 定时器就可以实现上述功能。

我们可以使用 timer 定时器中的寄存器 cc[1],和 cc[0]来设置上面说的 counter1 值和 counter0 值。并分别设置当计数器计数到指定值是产生中断。 在中断里面 将电平翻转就可以了。
但是这中方法因为中断的处理需要 CPU 参数，会影响 PWM 的周期和占空比。更多的影响是如果 timer 会频繁产生中断。导致正常的程序执行流程会被频繁打断。

作者：不离不弃 qq 574912883

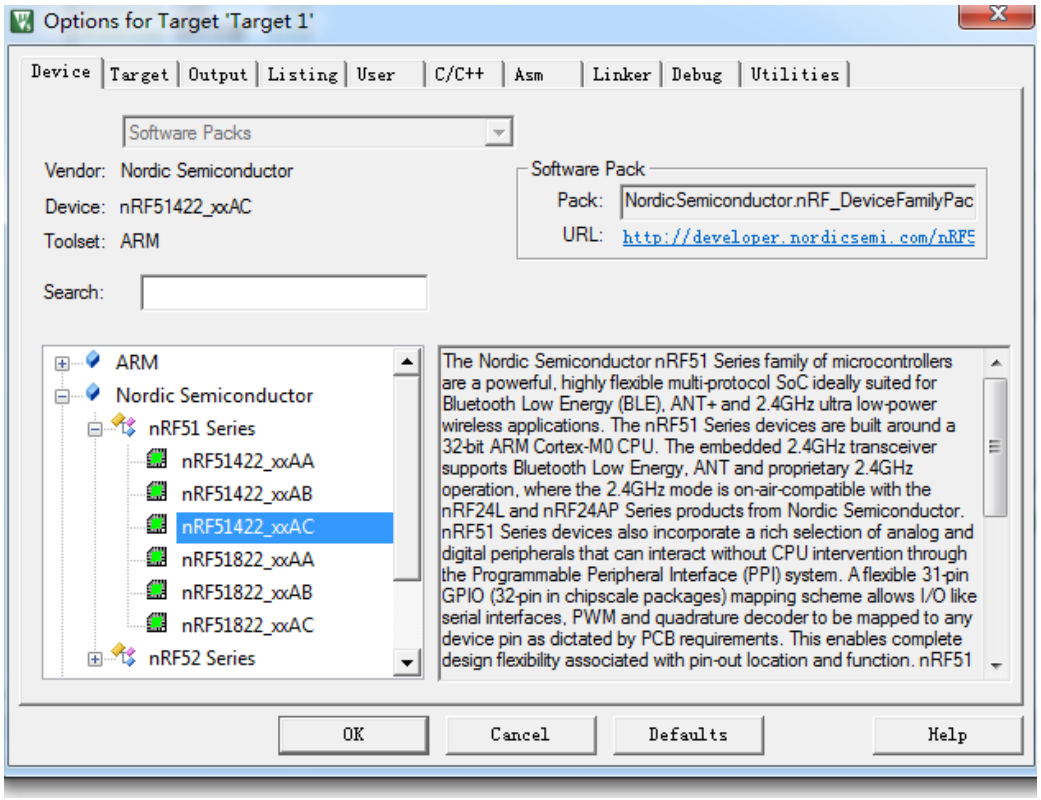
所以这里需要用到 51822 的 可编程外围互联系统(PPI), 该系统可以使 51822 的外围模块在无 CPU 参与的情况下相互协作。(详见 PPI 教程)

同时因为使用 PPI 让 timer 模块和 GPIO 模块来协作产生 PWM，所以这里不能使用普通的 GPIO，而需要使用针对 PPI 的 GPIOTE 模块。(详见 GPIOTE 教程)

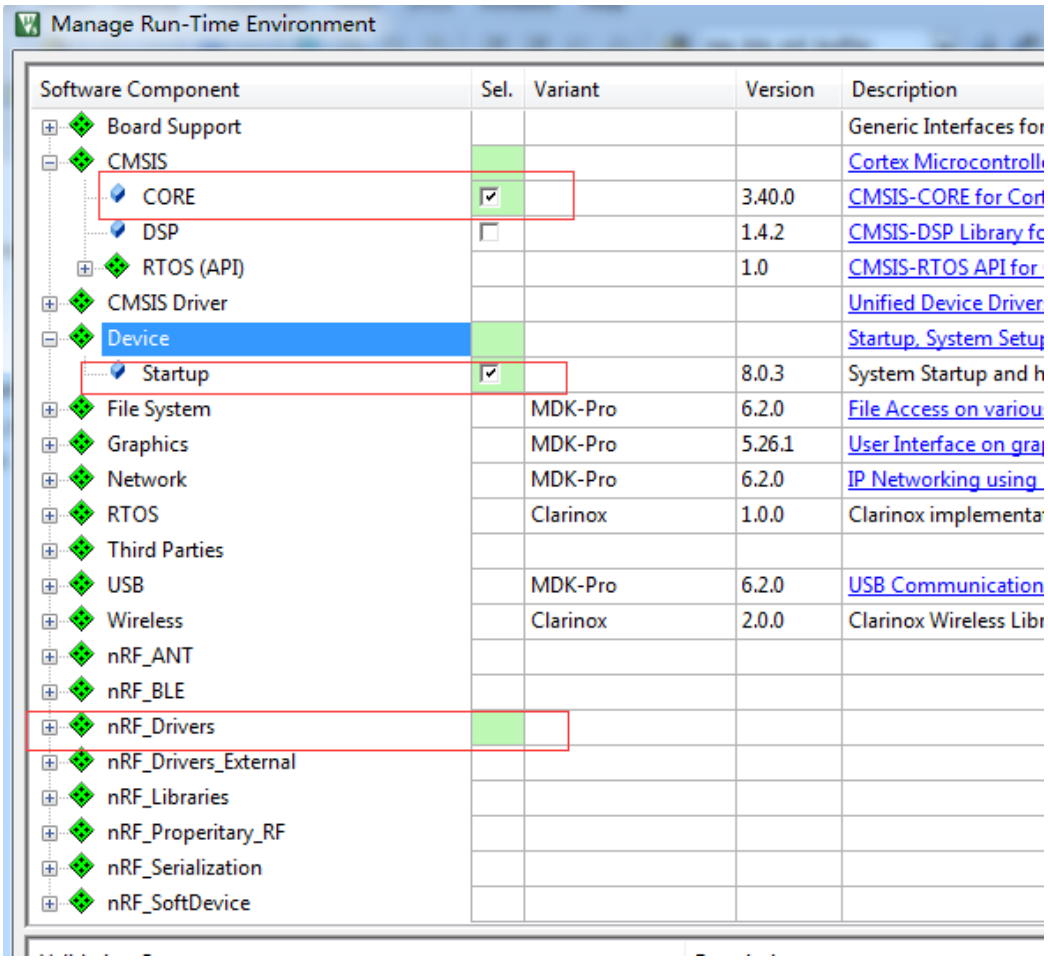


如上图所示。我们使用 timer 模块 让其 计数到 counter0 和 counter1 时分别产生 event0，和 event1。这两个 event 通过 PPI 然后触发同一个 task，这个 task 就是翻转电平。

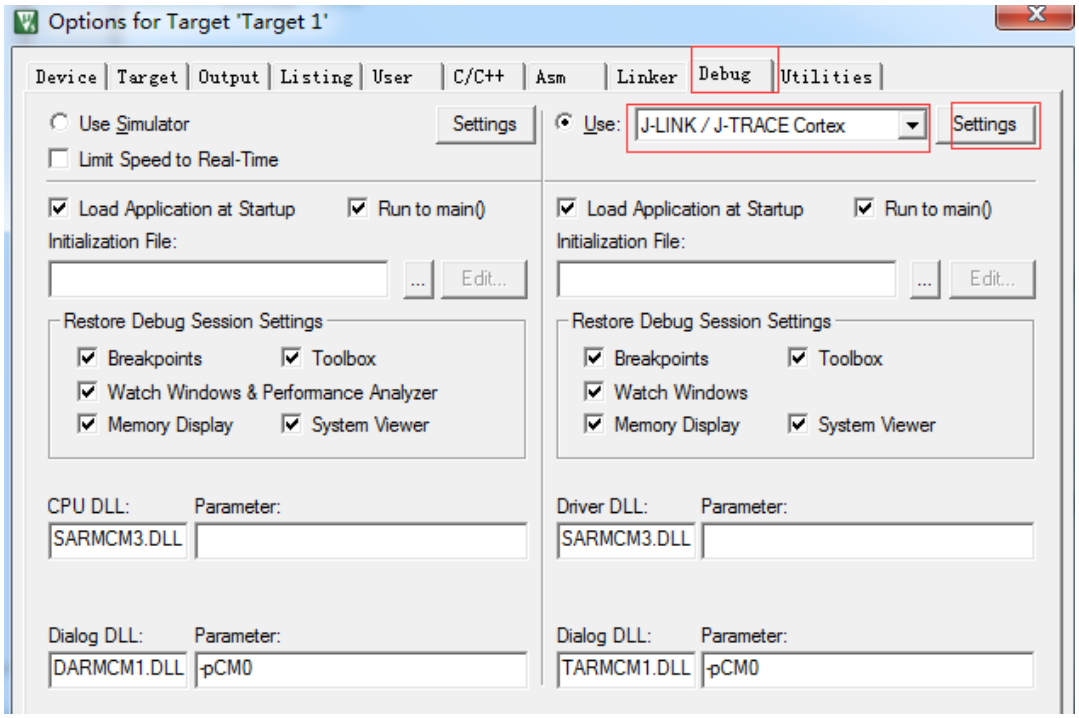
下面新建工程选择自己板子的芯片型号

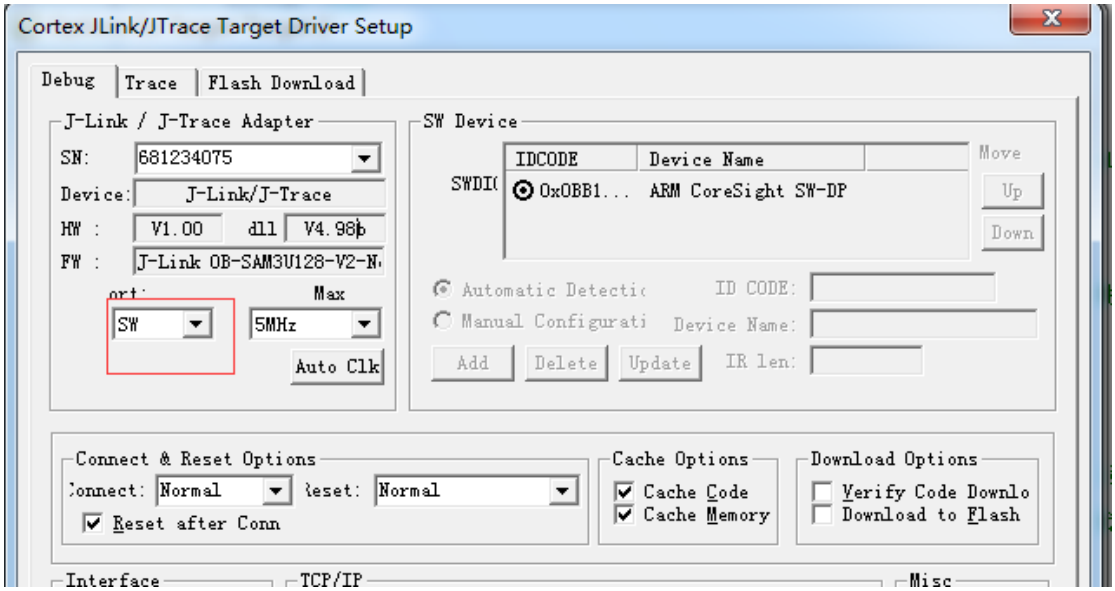


同样也是不适用 sdk 中提供的现成库，而是直接操作个模块寄存器来实现 PWM，所以运行时环境勾选下必要的 CMSIS 下的 CORE，Device 下的 Startup。因为用了 gpio 的函数 勾选一下 nRF_Drivers 下的 nrf_gpio 就可以了。

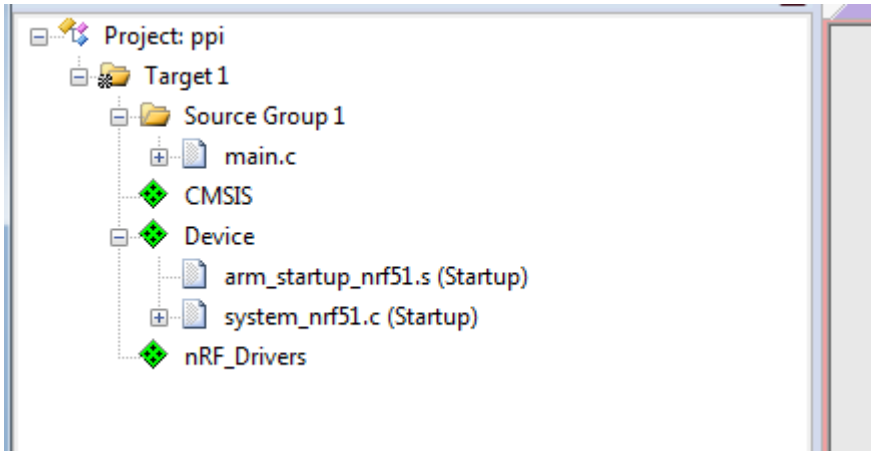


然后配置 jlink 的设置(我的板子使用的是 jlink 的 sw 方式下载程序)。





创建 main.c 文件，然后添加到工程中



下面是 main.c 代码细节。

```
#include "nrf51.h"
#include "stdio.h"
#include "nrf_gpio.h"

#define PWM_OUT          22

void timer0_init(void){
    NRF_TIMER0->PRESCALER  = 4;      //2^4    16 分频成 1M 时钟源
    NRF_TIMER0->MODE = 0;            //timer 模式
    NRF_TIMER0->BITMODE = 3;     //32bit

    NRF_TIMER0->CC[1] = 1000000;     //cc[1]的值等于是 1s，这里相当于方
波的周期为 1s
    NRF_TIMER0->CC[0] =500000;       //调节占空比，这里设置为 0.5

    NRF_TIMER0->SHORTS = 1<<1;       //设置到计数到 cc1 中的值时  自动清 0
重新开始计数

    NRF_TIMER0->TASKS_START = 1;     //启动 timer


}
```

```
void gpiote_init(void){

    NRF_GPIOTE->CONFIG[0] = ( 3 << 0 )          //作为 task 模式
                          | ( PWM_OUT << 8) //设置 PWM 输出引脚
                          | ( 3 << 16 )     //设置 task 为翻转 PWM 引脚的
电平
                          | ( 1 << 20);     //初始输出电平为高
}



//使用了两个 PPI 通道。 通道 0 用来将 timer 的 event0 (计数到 cc0 的值产
生的事件) 与 上面设置的 GPIOTE task 绑定在一起
//通道 1 用来将 timer 的 event1 (计数到 cc1 的值产生的事件) 也与上面的
GPIOTE task 事件绑定在一起。
//这样到计数到 cc0 和 cc1 时都会自动翻转 PWM_OUT 引脚的电平。
void ppi_set(void){
    NRF_PPI->CH[0].EEP  =  (uint32_t)(&NRF_TIMER0->EVENTS_COMPARE[0]);
    //注意，这里赋值要取地址
    NRF_PPI->CH[0].TEP = (uint32_t)(&NRF_GPIOTE->TASKS_OUT[0]);


    NRF_PPI->CH[1].EEP = (uint32_t)(&NRF_TIMER0->EVENTS_COMPARE[1]);
    NRF_PPI->CH[1].TEP = (uint32_t)(&NRF_GPIOTE->TASKS_OUT[0]);

    //两个通道的 task 端绑定的都是翻转电平的 task
    //使能 PPI 通道 0 和 通道 1
    NRF_PPI->CHENSET = 0x03;
}

int main(void){

    gpiote_init();
    ppi_set();
    timer0_init();
    while(1);
return 0;
}
```

通过调节 cc0 和 cc1 的值就可以分别控制占空比和周期了。这里只是实例。实际使用简单封装下就可以当做自己的 PWM 来使用了

作者：不离不弃　qq 574912883