基于LGTSDK Builder

LGT8F690A 快速开发系列教程

第七篇：动态功耗控制

概述

本篇为系列教程的第七篇。如果需要了解教程相关的软件硬件环境，请参考本系列教程的第一篇：《LGT8F690A快速开发系列教程第一篇_急速上手》

在一些电池供电的应用中， 对整机功耗的要求比较严格。系统功耗分为静态功耗和动态功耗两种。静态功耗对应系统处于休眠状态时的功耗；动态功耗为系统处于正常或者低速工作状态时的系统功耗。很多电池供电的消费类电子产品，大部分时间处于休眠状态，这种设备更多的关注系统的休眠功耗。而另外一些设备，以为应用的要求， 必须长期处于工作状态或者处于低速工作状态，这种设备，需要更低的动态功耗。

LGT8F690A支持最低1uA的深度休眠模式，可以满足大部分对静态功耗要求严格的应用。同时LGT8F690A也实现了一个细化的动态功耗调整机制， 可以根据系统的工作频率，调整动态功耗，达到合理控制动态功耗的目的。

本篇教程将分两部分， 分别介绍LGT8F690A的动态和静态功耗控制！

动态功耗控制

MCU的动态功耗取决与系统的工作电压，频率以及系统中处于工作状态的模块。由于应用环境的限制，工作电压一般都是处于比较固定的范围。 因此动态功耗的控制，更多是对系统的工作频率以及工作模块的功耗控制。

要达到理想的动态功耗，首先需要根据应用需要，选择合适的系统工作频率。在满足工作性能的前提下， 更低的工作频率能够获得更低的动态功耗。

LGT8F690A内部有高速、低速两种时钟源。在正常工作模式下， 内部16MHz的高速RC时钟，配合一个4位系统时钟分频器，可以实现灵活的系统时钟配置，满足系统正常的工作性能要求。内部32KHz低速时钟，主要用于系统不需要满负荷工作时，维持一个低速低功耗的工作状态， 同时又可以快速的影响系统状态的变化。

当系统处于高速时钟工作时， LGT8F690A还具有两种非常有效的功耗调整机制：

1. 指令周期模式的动态调整；
2. 电源驱动能力的动态调整；

指令周期模式用于调整内核执行指令的频率与系统主时钟直接的关系。LGT8F690A可以通过MCUCR寄存器，动态的将内核指令周期模式在1T/2T/4T直接调整。当我们需要较高的系统主频，而内核又不需要很高的性能时，可以将指令周期设置到4T模式， 降低运行功耗；

电源驱动能力的调整， 对于优化动态功耗， 效果非常理想。电源驱动能力的调整和系统指令运行的频率相关。当指令周期工作在较低的时候，我们可以通过设置合适的电源驱动能力， 降低系统中驱动模块的功耗级别， 获得更低的动态功耗。动态功耗的调整， 是通过MCUCR寄存器的PPLP控制。下面是PPLP控制的驱动与系统指令周期的对应关系：

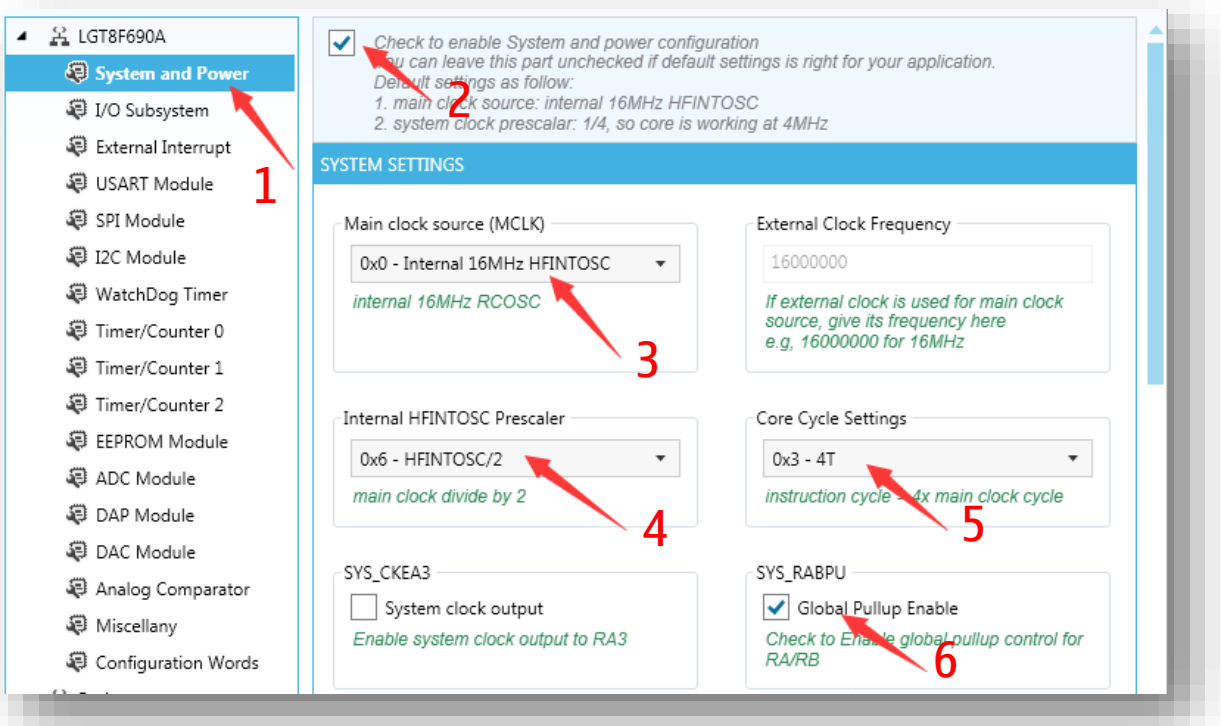
MCUCR:PPLP	指令周期频率	应用情况
11	8MHz ~ 20MHz	高速模式，要求工作电压不低压4.5V。高速模式，此时系统主频工作在最高级别，指令周期模式工作在1T，可以获得LGT8F690A最高性能，但也同时达到最大的功耗。
01	500KHz ~ 8MHz	低功耗模式， 工作电压不低于2.5V。这是LGT8F690A比较常用的工作模式。 我们也建议系统在处理常规任务是处于这个工作模式，在满足控制逻辑的同时， 也能获得较低的动态功耗。
00	低于500KHz	省电模式，工作电压不低于2.0V。主要用于低速低功耗模式。比如系统主时钟切换到内部32KHZ 低功耗RC振荡器。此模式下，系统可以在低速运行时，实现30uA@3.3V左右功耗。

以上表格中的指令周期频率， 并不是系统主时钟的频率。比如，如果系统工作在16MHz的内部RC频率下，但指令周期模式选择2T或者4T， 对应的是8MHz和4MHz的指令周期频率， 此时，我们仍然可以将电源驱动模式设置为低功耗驱动模式(PPLP = 01)。

下面我们将以LGTSDK Builder为例， 示例如何利用SDK中提供的功耗控制接口， 实现动态功耗控制。本实例需要电流表的协助， 以观察不同模式下， 芯片动态功耗的变化情况。另外也需要示波器或者协议分析仪， 监控一个I/O的输出，以确定程序的运行状态。

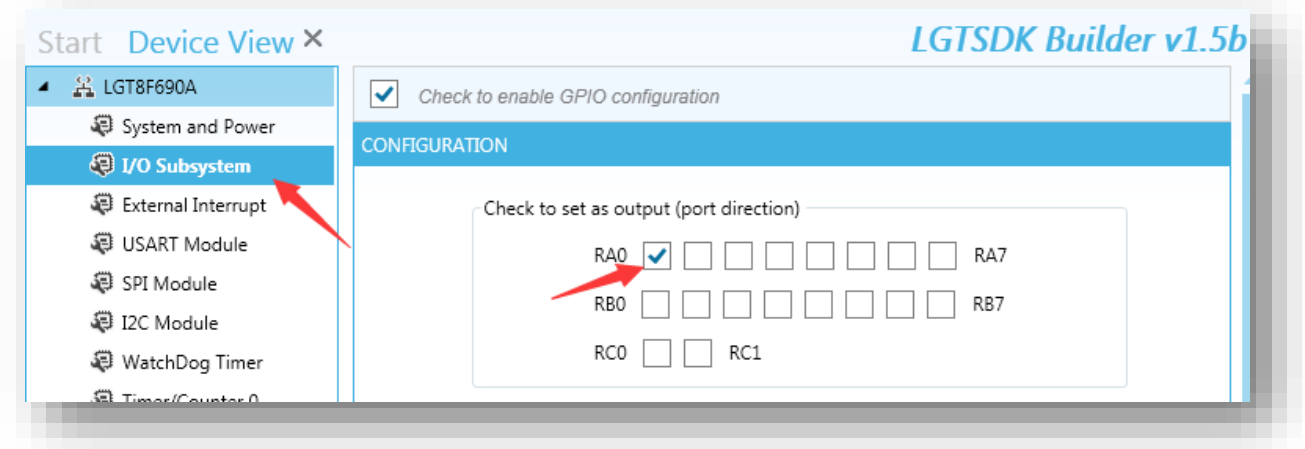
我们将要实现的程序逻辑如下：
首先， 程序启动后， 工作于一个我们配置好的高速工作模式，观察此时的输出状态和系统功耗；然后将一个输入的I/O，这里将使用RC0， 拉低。 触发软件进入到省电模式。观察此时的程序输状态和系统功耗。

为了获得最大程度的低功耗， 我们将其他没有使用的I/O全部设置为输入上拉。这是一种省电同时又相对安全的做法。动态功耗的控制在LGTSDK Builder的图形配置界面没有对应的选项。 我们需要通过SDK函数来实现对电源驱动模式的调制。



我们先是配置一些程序常规运行需要的设置， 这些配置都在[System and Power] 配置页中。这里选择默认的工作频率和指令周期模式：
3： 使用内部16MHz 高速RC振荡器作为主时钟源；
4： 系统预分频是2， 也就是系统工作频率为8MHz；
5： 指令周期模式选择4T模式， 这样可以进一步的降低动态功耗；
6： 使能全局上拉电阻。我们需要将没有使用到的I/O全部设置为输入上拉模式；

下面是配置下控制软件运行和输出运行状态的I/O， 我们用RA0的作为输出运行状态，RC0/1作为输入，实现通过I/O控制运行的模式：



所需外设已经配置完成， 下面是点击[Device View] 下的 [Build]按钮，生成SDK代码。
按照惯例， 我们先列出例程中将要使用到的SDK函数：

函数名称	功能描述	使用方法
sysClockINTRC()	设置系统时钟位内部时钟源 此函数在系列教程第三篇中有重点介绍	sysClockINTRC(SYS_INTRC_RC32K) 其他参数，比如： SYS_INTRC_DIV0: 内部16M，不分频 SYS_INTRC_DIV2: 内部16M，2分频
gppPullupEnable()	使能I/O的上拉电阻 前提需要使能全局上拉控制	gppPullupEnable(GPPA, P0|P1|P2) 将会使能RA0/1/2的上拉电阻
gppReadSingle()	读一个I/O的电平状态	Value = gppReadSingle(GPPC, P0)
gppToggle()	翻转一个I/O的输出状态	gppToggle(GPPA, P0) 翻转RA0的输出状态
sysPowerLevel()	设置电源驱动模式	sysPowerLevel(SYS_POWER_HL) 设置系统电源为高速驱动模式 sysPowerLevel(SYS_POWER_LL) 设置系统电源为最低功耗模式

```c
int main(void)
{
    // Device initialization
    init_modules();

    // pullup all unused I/O
    gppPullupEnable(GPPA, PALL);
    gppPullupEnable(GPPB, PALL);
    gppPullupEnable(GPPC, (P0|P1));

    // set power level (PPLP = 01)
    sysPowerLevel(SYS_POWER_ML);

    // Add your code from here
    while(1)
    {
        // toggle RA0
        gppToggle(GPPA, P0);

        if(gppReadSingle(GPPC, P0) == LOW)
        {
            // set system clock to LFINTOSC 32KHz
            sysClockINTRC(SYS_INTRC_RC32K);
            // set power level to lowest level
            sysPowerLevel(SYS_POWER_LL);
        }

        if(gppReadSingle(GPPC, P1) == LOW)
        {
            // firstly, restore power level settings
            sysPowerLevel(SYS_POWER_ML);
            // then, it's safe to restore system clock
            sysClockINTRC(SYS_INTRC_DIV2);
        }
    }
```

开启I/O的上拉电阻

设置电源驱动模式为低功耗模式

翻转RA0，工作指示

检测RC0是否为低电平

切换到低频32KHz运行
选择最低电源驱动模式

检测RC1是否为低电平

首先恢复电源驱动模式
然后切换到高速时钟运行

几点注意事项：

1. 浮空的输入模式的I/O会导致漏电，因此开启了所有I/O的内部上拉。RA0是输出的I/O，虽然也开了上拉，因为是输出I/O，上拉并不会打开。
2. 在降低电源驱动前，需要首先降低系统频率，确保系统功耗降低后，才能进入低驱动模式；
3. 开启高速时钟前，需要先设置到合适的电源驱动模式，然后才能切换到高速时钟；
4. 我们使用拉低RC0/1的方式进入对应的模式，因为RC0/1内部有上拉，所以当进入目标模式后，需要断开RC0/1的下拉，否则会带来多余的漏电。

代码编译通过，可以将HEX下载到最新板上测试了！
将系统电源串入电流表，示波器检测RA0的输出。然后通过下拉RC0/1分别进入不同的工作模式。