MVR8X

LGT8F0XA 系列 8 位微控制器系列

LGT8F08A

应用指南

LGT8F08A 应用开发指南

V1.0

2013/12/23

LGT8F08A 应用开发指南

内容概述

LGT8F08A 是 LogicGreen 设计第一代 8 位高性能 RISC 微控制器，定位于低端消费类电子市场，目前已经被越来越多的用户接受，广泛用于各种控制类电子产品。

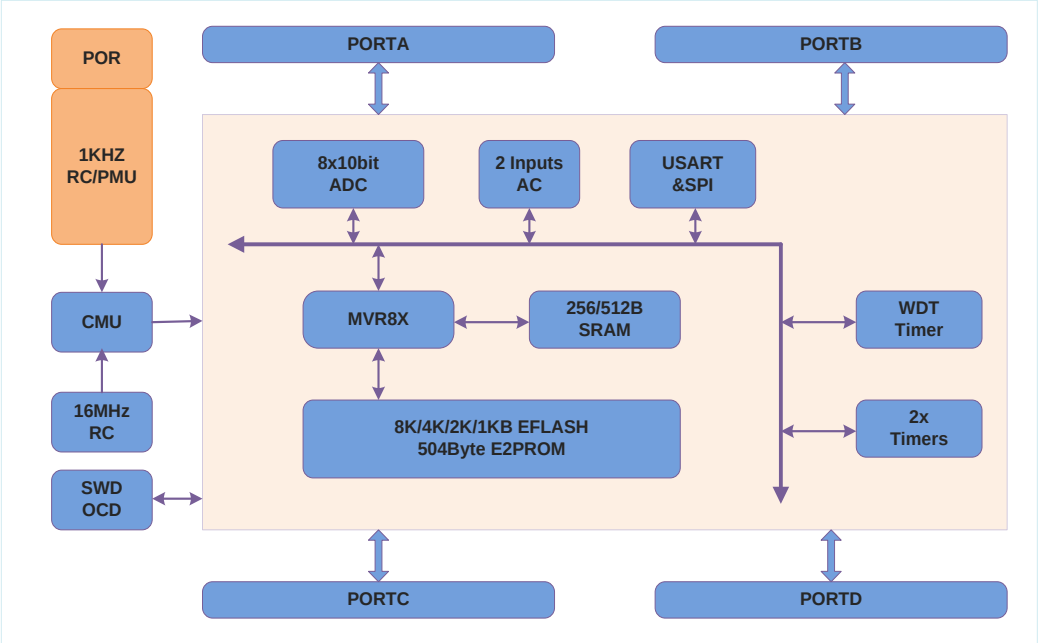
LGT8F08A 设计之初没有考虑到对目前产品的兼容，因此给第一次使用该产品的客户带来一些不便，不同客户的不同应用环境，也让 LGT8F08A 的缺陷充分的暴露。在这个过程中，LGT 和大家共同努力，基本上使得 LGT8F08A 产品的应用方案趋于稳定。LGT 非常感谢各位愿意尝试使用我们产品和对我们产品提出各种意见和反馈的每一位客户，没有大家的积极参与，我们也无法推出更加完善，更有竞争力的产品。

我们把与 LGT8F08A 所有在使用过程中遇到的问题和解决的办法总结归纳，提供给各位参考，希望有助于加快产品开发的周期。

文档涉及的内容如下：

1. LGT8F08A 功能介绍
2. LGT8F08A 的开发平台搭建
3. LGT8F08A 已知 BUG 和解决的方法
4. LGT8F08A ADC 的问题和解决方法
5. LGT8F08A 时钟和功耗管理
6. LGT8F08A –SOP8 封装的应用开发

功能介绍



[LGT8F08A 系统框架图]

LGT8F08A 基于 LGT 开发的第一代 MVR8X 8 位 RISC 内核，兼容 AVR8L 指令集，内核经过大量的优化，使得 LGT8F08A 在指令执行效率上高于 AVR 原有的构架。
下面是 LGT8F08A 功能列表，具体的细节，请参考 LGT8F0XA 系列编程手册。

- ✓ 高性能，低功耗 8 位 RISC 内核
- ✓ 8K 字节可编程 FLASH 程序存储器
- ✓ 512 字节内部 SRAM
- ✓ 504 字节可字节访问的数据 FLASH
- ✓ 系统配置位以及程序空间访问 LOCK 控制位
- ✓ SWD 双线调试接口
- ✓ 8 通道 10 位模数转换器（ADC）
- ✓ 2 输入模拟比较器
- ✓ 内部上电和掉电复位功能的 POR
- ✓ 1%可校准 16MHz 内部 RC 振荡器
- ✓ 1KHz 低功耗内部 RC
- ✓ 封装：SSOP28L/SSOP24L/SOP8L， QFP32L
- ✓ 一路具有独立预分频器的 8 位定时计数器
- ✓ 一路具有独立预分频器的 16 位定时计数器
  - 支持比较输出和输入复活功能
- ✓ 最多支持 3 路 PWM 输出
- ✓ 可工作与主从模式的 SPI 控制器
- ✓ 可编程同步/异步 USART 控制器
- ✓ 可编程看门狗定时器
- ✓ 最多 25 个可编程 I/O
- ✓ 丰富的内部和外部中断源
- ✓ 工作电压： 1.8V～3.6V
- ✓ 速度等级： 0～16MHz@1.8V~3.6V
  0～25MHz@3.0V~3.6V

开发平台

概述

LGT8F08A 内核兼容 AVR8L 指令，I/O 寄存器地址定义基本与 ATMega162/164A 一致，因此在开发环境的选择上，也非常的灵活。所有兼容 AVR 的开发环境，都可以用于开发 LGT8F08A。
下面是一些常用 AVR 开发环境：
AVR Studio 4.18 (支持在线调试， 一般是配合 winavr 使用)
AVR Studio 6.0 build 1843 （支持在线调试）
AVR Studio 6.0 build 1996 （支持在线调试）
IAR Workbench for AVR （支持在线调试）

ImageCraft C for AVR
CodeVisionAVR
WinAVR (通过 GDB 在线调试)
AVRGCC Toolchain （支持 LINUX 系统）
CrossPack for AVR （支持 Mac OSX 系统）

以上系统都可以用于开发 LGT8F08A，可以根据你的实际情况选择。
对于初学者或刚刚接触到 AVR 的用户，我们推荐使用：
AVR Studio 4.18 (with SP3)
IAR Workbench for AVR

推荐这两款的原因主要是，他们都支持在线调试功能，支持烧写程序，因此在开发阶段可以不依赖与其他辅助软件，就可以完成所有的流程。
AVR Studio 4.18 的下载地址：AVR Studio 4.18 + SP3
IAR Workbench for AVR 是收费软件，推荐的版本是 5.51.1，网上可以找到破解版本。

有了开发 IDE，下面就是需要调试器和下载器了。
调试器连接开发环境和目标板，可以实现程序下载以及调试功能。
支持 LGT8F08A 调试，需要使用 LGT 提供的 SWDICE_mkII 调试器，SWDICE_mkII 兼容 AVR Studio 以及 IAR Workbench for AVR， 在功能上等价于 ATMEL 的 JTAGICE_mkII 调试器，但只能用于调试 LGT 的 MCU。

SWDICE_mkII 虽然能够实现程序的下载，但是对于 LGT8F08A 来说，还不能算一个完整的下载器，因为 SWDICE_mkII 只能读写程序空间，如果需要烧写 E2PROM 区域以及配置位，就需要 LGT8F08A 专用的 ISP 下载器, LGT_USBASP。

与 LGT_USBASP 配套使用的下载工具是 LGTMix_ISP，用于烧写所有 LGT 推出的 MCU 产品，目前而言，可以支持 LGT8F08A 系列以及 LGT8F88A。

安装 AVR Studio 4.18

我们下面就以使用 AVR Studio 4.18 为例，描述如何安装和配置开发环境。
AVR Studio 4.18 本身没有编译器，需要配合其他的 AVR 编译器使用，目前支持的编译器有:
WinAVR， 以及 ATMEL 的 avr-toolchain。比较常用的是 WinAVR.
在安装 AVR Studio 4.18 之前，首先需要下载安装 WinAVR ： WinAVR-20100110

下载到的是一个可执行的安装文件，执行点击就可以启动安装，中间可能需要指定安装位置，这个根据你自己的情况设置。安装程序也会把 WinAVR 安装路径添加到系统环境变量中，这样我们就可以在全局环境下使用 WinAVR。

安装完成后，可以打开一个 cmd 终端，输入 avr-gcc，如何提示没有这个命令，说明系统环境变量无效，你需要手工把 WinAVR 安装目录下的 bin 的路径指定到系统环境变量 PATH 中。

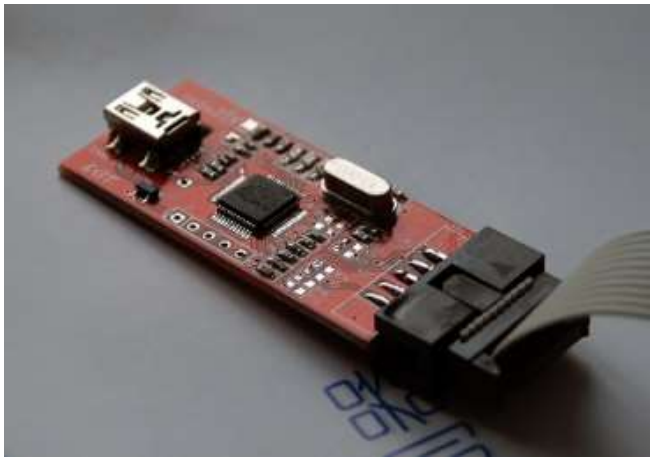
下面就可以安装 AVR Studio 了，从上一节给的地址中，找到并下载 AVR Studio 4.18 以及 AVR Studio 4.18 SP3，首先安装 AVR Studio 4.18，然后再安装 sp3 补丁文件。

安装过程也非常的简单，没有复杂的设置。安装过程中也会同时安装 AVRJungoUSB 驱动，这个就是我们调试器的驱动程序。

AVR Studio 安装过程中，会根据系统的环境变量，找到我们之前安装的 WinAVR，对自己的编译器环境进行正确的配置，这个过程不需要我们完成。

AVR Studio 安装完成，下面是准备调试器，测试开发平台。

SWDICE_mkII



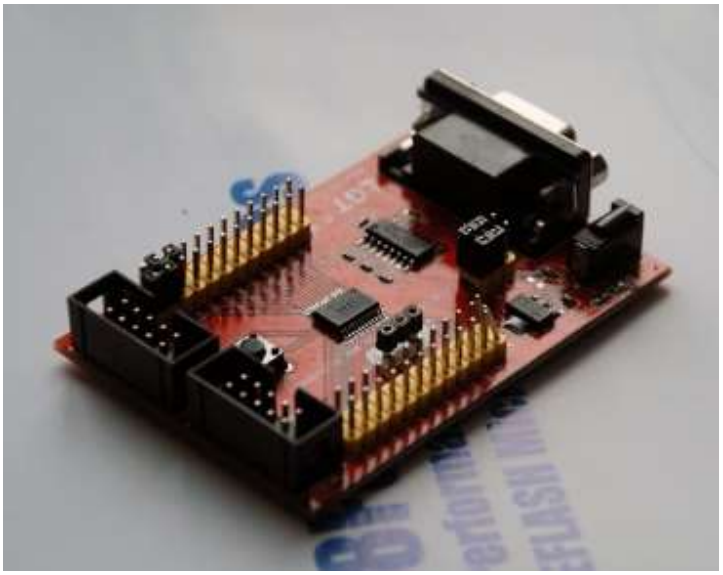
SWDICE_mkII 调试器外形如上图所示，在使用之前，首先需要确认 SWDICE_mkII 运行的固件是否支持当前的开发环境。SWDICE_mkII 支持 Studio 4.18 和 Studio 6.0 需要不同的固件，因此如果你不确定当前 SWDICE_mkII 固件的版本，建议重新为其更新固件版本，SWDICE_mkII 更新固件的方法非常简单，下面是步骤：

1. 首先在 LGT 的官方网站上下载到 Studio 4.18 的对应固件，解压待用；
2. 将 SWDICE_mkII 背面的[Update]跳线短路
3. 将 SWDICE_mkII 与电脑连接，等待片刻就会有一个 U 盘弹出

4. 将下载固件目录下的 bin 文件复制到这个 U 盘中
5. 退出 U 盘，移除[Update]上的短路跳线，升级完成！

开发环境的连接测试

调试器已经准备就绪，下面是连接开发板，这里我们以 LGT 的官方 SSOP24L 封装的 LGT0F08A 最小评估板为例：



LGT8F08A_SSOP24L 最小评估板

LGT8F08A_SSOP24L 评估板可以在官方网站上申请，也可以在 LGT 的官方淘宝店购买。

由于 SWDICE_mkII 调试器无法给目标板供电，需要为目标板单独供电，所需供电线一般会与最小评估板一同发发货。电源输入口靠近 DB232 插座。

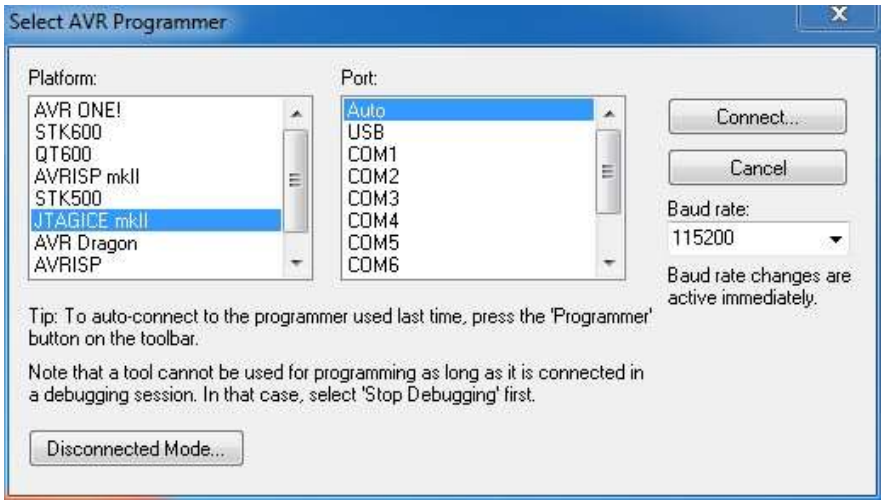
首先，将 SWDICE_mkII 调试器通过 10 针扁平线与评估板上的 SWD 插座相连接，注意评估板上有两个 10 针插座，一个为 ISP 烧写插座，一个是 SWD 调试插座，请根据板上的标识识别。

SWDICE_mkII 连接后，给评估板上电。然后将 SWDICE_mkII 调试器与 PC 连接。

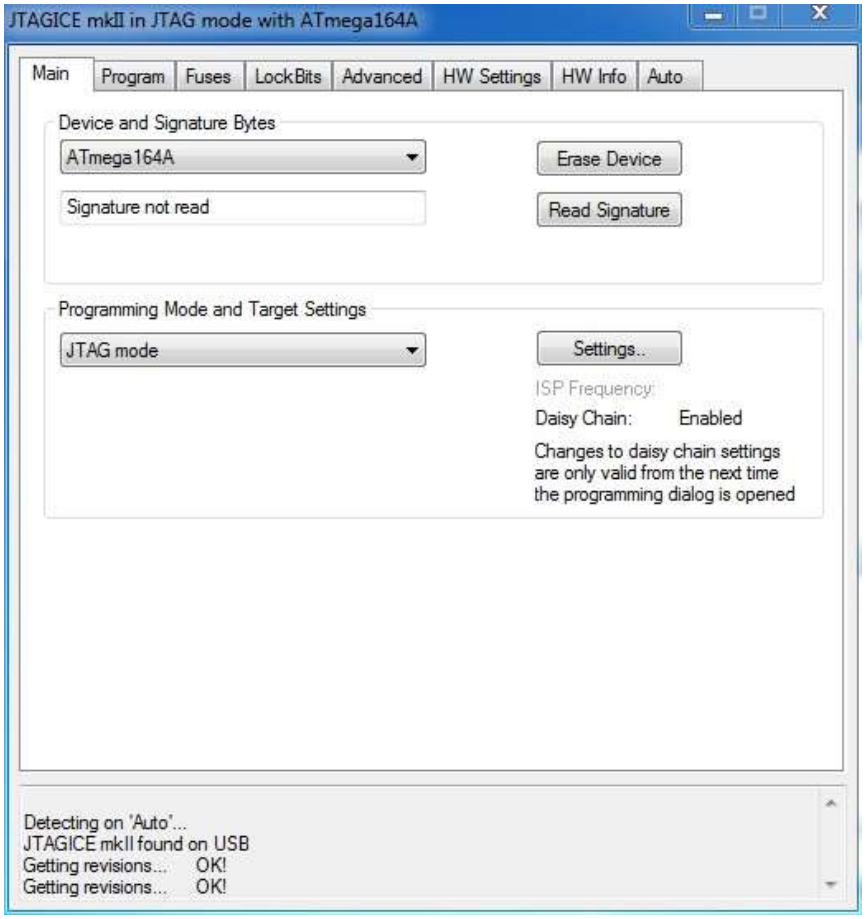
在电脑上通过开始菜单，启动 AVR Studio 4.18，启动后，默认将弹出一个建立工程的对话框，我们这里先进行硬件连接测试，所以先将其关闭。

在工具栏中，找到如下图标的按钮，启动连接对话框：

连接对话框如下图所示：



按照上图所示，在[Platform]中选中[JTAGICE mkII]，在 Port 里选择[Auto]，然后点右上角的[Connect…]按钮，如果一起顺利，片刻之后，会有一个新的对话框弹出，如下：



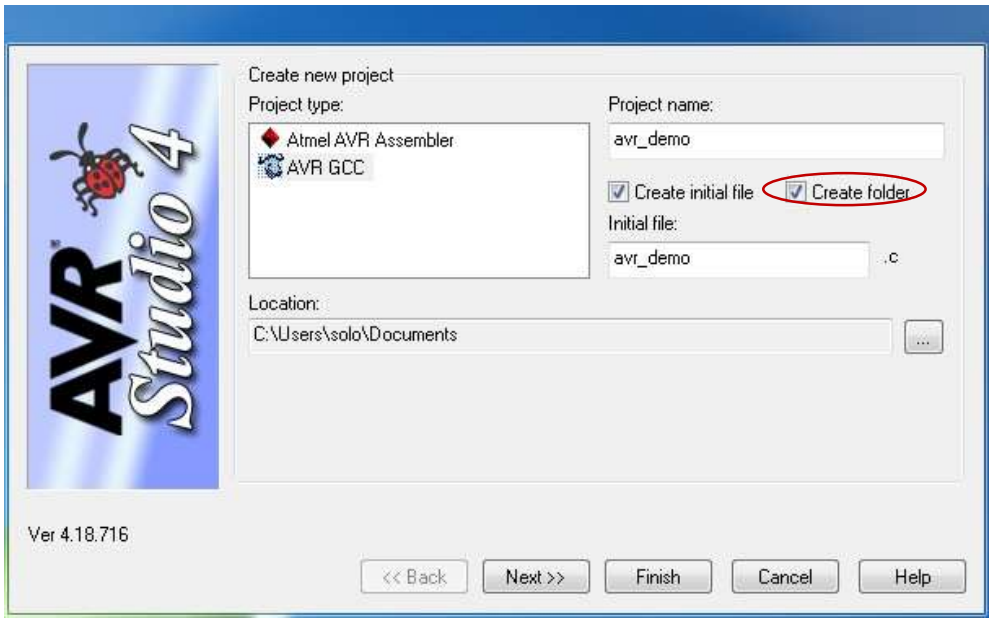
出现上图信息，说明硬件连接一起正常，开发平台基本搭建完毕，可以使用。
如果弹出其他出错信息，基本上和目标板供电以及连接线有关，请仔细检查重试。

有时，会出现上面的对话框，但没有正确的识别到目标设备。这时可以手工在[Main]表项的[Device and Signature Gytes]分组下面，指定设备为：ATMega164A。然后点傍边的

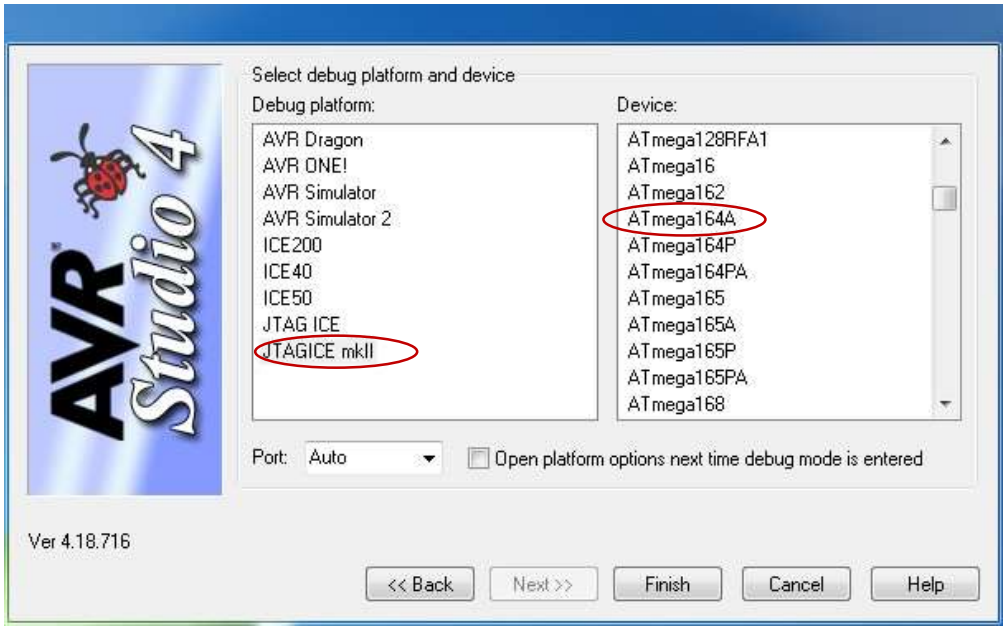
[Read Signature]按钮，重新读取，如果读到了返回值，说明硬件连接正常，否则需要仔细检查目标板是否正常工作。

创建和配置新项目

确认硬件连接正常后，下面就可以创建一个新工程，编写测试代码。
下面是创建新工程和设置的简单介绍：
首先，从 Studio 的主菜单 [Project] -> [New Project]启动新工程：



在这个对话框里填写项目的名称，如果你想同时也创建一个空的 C 文件，勾选[Create initial file]，建议也勾选上 [Create folder]，为工程建立独立的目录。然后点击[Next >>]:



这一步是要选择项目的目标设备和调试平台。首先，在[Debug Platform]下选择[JTAGICE mkII]， 然后在[Device]下面，选择[ATMega164A]。

这里，调试平台我们选择了[JTAGICE mkII]，也就是当你编译完成，启动调试时，代码会通过调试器下载到目标板上，运行在线仿真。如果你想用软仿真，可以在这里选择[AVR Simulator]，或者选择[AVR Simulator 2]。后面还可以通过主菜单[Debug] -> [Select Platform and Device...]重新选择其他配置。

LGT8F0XA BSP/SDK

为了让用户更加方便快速的使用 LGT8F08A，我们提供了一个经过测试的开发包，包括了 LGT8F08A 所有模块的驱动源代码和外设使用的例程；用户可以充分利用这里的代码，直接运用到自己的项目中，

LGT8F0XA BSP/SDK 可以从 LGT 官方网站上下载。

下面我们就以 BSP/SDK 中的例程，简单介绍下如何使用 AVR Studio 编译，下载和在线调试。

首先将下载到的 BSP/SDK 解压到一个专用的目录下。

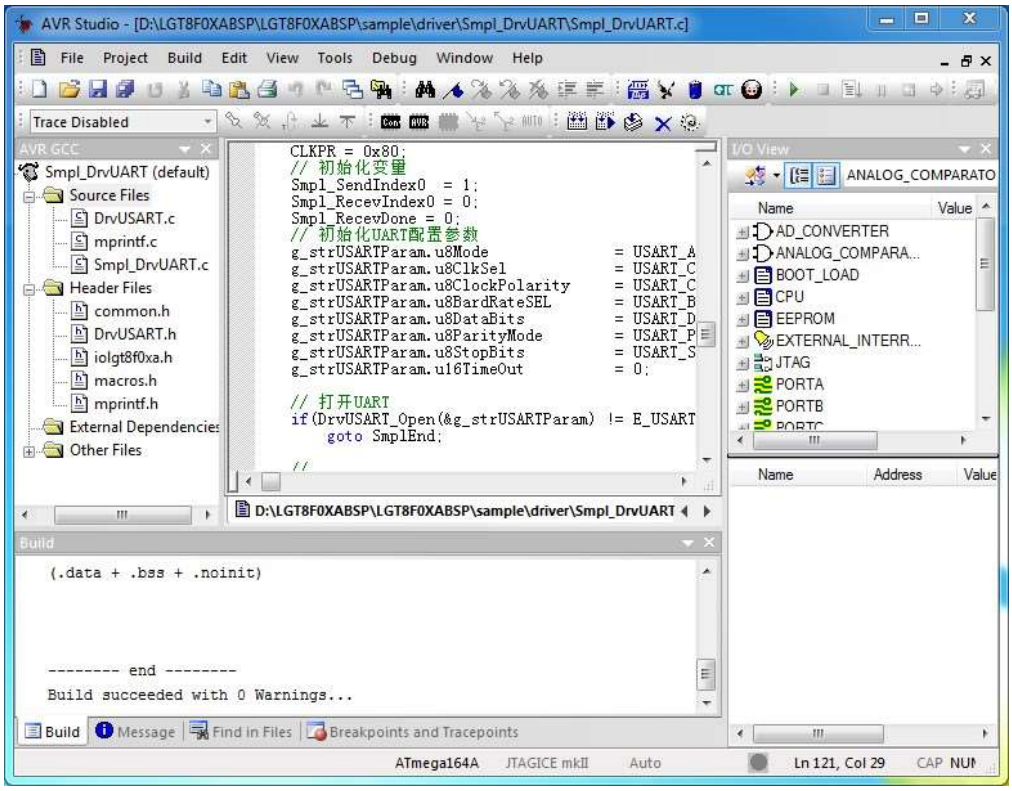
然后，在使用 AVR Studio 打开之前，我们需要先根据我们系统中 WinAVR 的安装路径和版本更改下 Makefile 文件， 这里我们就以 LGT8F0XABSP\sample\driver\Smpl_DrvUART 为例：

进入这个目录后，用记事本或者其他文本编辑器打开目录下的 Makefile 文件，然后找到下面一行：

```
# include folder
INCLUDES = -I "$(INCDIR)/driver" -I "$(INCDIR)/system" -I"c:/WinAVR/avr/include/avr"
```

将”WinAVR”路径更改为操作系统中 WinAVR 的实际路径，保持文件即可。

更改完成后，双击这个目录下的 Smpl_DrvUART.aps 文件即可打开这个项目。



打开项目后，可以通过主菜单[Build] -> [Build]编译整个项目，查看输出窗口，如果编译失败，请检查刚才指定的 WinAVR 目录是否正确。

除了通过菜单命令，使用工具栏的图标也是非常快捷的：

编译通过之后，可以下载到目标板上运行，调试了。

首先，先通过[Debug] ->[Select Platform and Device]选择正确的调试器和目标芯片名称。

然后，通过[Debug]菜单下的[start debugging]或工具栏的[Start Debugging]按钮启动下载调试。程序将首先通过调试期下载到目标芯片，然后代码中断在 main 函数的入口处。

调试相关的操作，可以在主菜单[Debug]下面找到。

LGT8F0XA BSP/SDK 里面有非常多的例程和驱动，具体的使用细节，请参考相关文档。

已知 BUG 和解决方法

BUG[1]：SBIC/SBIS 指令相关的 BUG

[BUG 产生的原因]

LGT8F0XA 的 PIN 寄存器设计为异步时序，直接反映了端口上的变化。这种异步使得 SBIC/SBIS 这样直接根据端口状态执行的操作可能会发生不稳定状态，而导致程序执行错误。

[解决的办法]

解决的方法就是避免使用 SBIC/SBIS 指令。

对于汇编语言，直接使用 IN 指令将端口值读入到通用工作寄存器，然后在使用 SBRC/SBRS 指令根据通用工作寄存器的值进行其他操作。

示例代码如下：

汇编代码如何避免使用 SBIS/SBIC

```
Label:
in r16, PINA
; Skip if PINA[1] is clear, avoid to use sbic directly
sbrc r16, 1
; Skip if PINA[1] is set, avoid to use sbis directly
sbrs r16, 1
```

对应 C 语言的设计，也要避免直接根据 PIN 寄存器的值做为判断的条件，如果需要根据 PIN 的状态产生跳转，也需要先将其读入到一个变量，然后把变量的值作为跳转条件，这里需要注意的是，为了保证编译优化后的代码和我们预期一致，需要对中间变量进行限定：

示例代码如下：

C 语言代码如何避免使用 SBIS/SBIC

```
volatile unsigned char pv; /*volatile to disable optimize on 'pv' */
/* wait until PINA[1] is set */
do {
    pv = PINA;
} while ((pv&0x2) != 0x02); /* avoid to use while(PINA&0x2) */
```

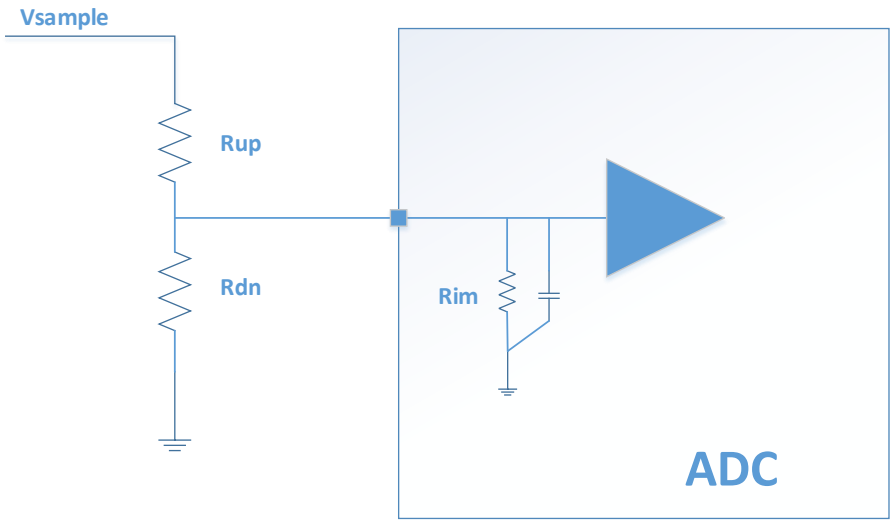
ADC 的相关问题和解决办法

LGT8F08A 的 ADC 相关问题主要有两个方面：

1. ADC 的输入阻抗小，约为 20KOhm 左右
2. ADC 的内部 1.25V 参考精度 5%

首先介绍输入阻抗小带来的问题，和相应的解决方法：

对于一般的分压测试电路，电路的结构如下：

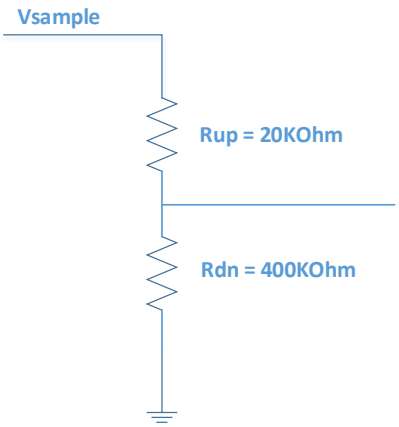


外部为一个分压电路，电压输入到 ADC 的输入通道，其中 Rim 为 ADC 输入通道对应的输入阻抗。

从上图可以看出，当 Rim 的值小于或接近外部的分压电阻 Rdn 时，内部阻抗对外部的硬件就不能忽略，内部的阻抗相当于与外部的下拉分压电路并联，结果分压点的电压要远小于我们要求的范围。

这个时候，设计分压电路时，就需要考虑到这个内部阻抗的影响。但由于考虑到分压电路本身的功耗，一般我们不会使用太小的分压电阻，这样我们就没有办法可以做到直接忽略内部阻抗，只有将内部阻抗(20KOhm)当作分压电路的一部分考虑。

下面是一种推荐的 1/2 分压电路，分压电路 Rup 为 20KOhm，接近 ADC 的输入阻抗；下拉的分压电阻 Rdn 可以考虑尽量大，这样既可以减小对内部阻抗的影响，也可以降低分压电路的功耗。



如果需要其他比例的分压值，可以通过变化 Rup 分压电阻实现。 如果不考虑内部阻抗的一致性问题，这样的分压电路基本上解决了 ADC 输入阻抗较小的问题。

但实际上，ADC 的输入阻抗有一定的不一致性，但经过我们测试，同一批次的芯片，基本上都在一个可以接受的范围内变化，可以满足大部分 ADC 采样精度需求。

下面介绍 ADC 的内部 1.25V 参考电压和相关的问题。

首先，ADC 的内部 1.25V 参考电压的设计精度保证在 5%以内。对于精度要求比较高的应用，建议使用 VCC 作为参考电压（VCC 需要由 LDO 或 DC/DC 提供）。

在一些低成本的应用中，可能也无法避免的使用到内部参考。内部参考可以保证在 5%以内，但是如果考虑到 ADC 的输入阻抗本身也具有一定的误差，总体的一致性可能就不能保证在 5%以内。

为了能够提高 ADC 的测量精度和一致性，我们给对 ADC 使用内部参考要求相对高的产品应用，给出了一种比较可靠的解决方法，经过我们实际测试，效果非常的好：

我们在测试芯片时，用一个精度很高的 VCC 供电，测试出每颗芯片的内部参考的实际值，把这个值保存到 GUID 的最低位置。这样，我们使用内部参考的值就是非常精确。

因为 LGT8F08A 有大量的芯片已经完成了测试，所以这些芯片中没有记录精确测量的内部参考电压值。**如果客户的应用使用了内部参考，并且有比较高的精度和一致性要求，可以提前告知我们，我们将会单独提供含有精确测量的内部参考值的芯片。**

调试器使用相关注意事项

在使用 LGT8F08A 的客户中，有一些客户遇到了调试器使用一次后就无法连接的问题。这个问题主要是因为两种情况导致：

1. 由于 SWD 调试接口是分别其他数字功能复用，当应用中有把 SWD 相对应的引脚用作其他功能时，将会影响到调试功能。
2. SWD 数据线是与 ADC 的输入通道 6 复用的。在使用 ADC 功能时，可能会配置 DIDR0 寄存器，这个寄存器是把 ADC 输入通道对应的数字功能关闭。因此需要注意，当关闭了 ADC6 的数字功能后，也会关闭 SWD 的功能。因此在配置 DIDR0 时请特别注意，不能简单的将 DIDR0 寄存器设置为 0xFF, 这样会关闭掉 SWD 功能。

   第二种情况也是发生可能性最多的，请遇到类似问题时，特别注意。

时钟管理和低功耗设计

系统时钟配置

LGT8F08A 正常工作时，支持三种时钟源，分别为：

1. 内部低频 RC（128Hz）
2. 内部高频 RC（16MHz）
3. 外部晶振/陶振/时钟输入

软件通过 PMCR 寄存器，可以实时在这三种时钟之间切换。时钟切换完成之后，为了降低功耗，可以将没有使用的时钟源关闭。PMCR 寄存器定义请参考 LGT8F0XA 系列编程手册。
下面是一段完整的子程序代码，用于切换系统时钟，并设置预分频。

代码示例：如何切换系统时钟到外部晶振

```
// css: system clock source select
//    0x0 : internal 16MHz RC
//    0x1 : external crystal
//    0x2 : internal 128Hz RC
// cps: clock pre-scalar select
//    0x0 : no scalr
//    0x1 :  clock/2
//    0x2 : clock/4
//    0x3 : clock/8
//    ……
//    0x7 : clock/128
void init_clock(unsigned char css, unsigned char cps)
{
    volatile unsigned char ctmp = 0;

   if(css == 0x1) {
      PMCR &= 0xfb;  // enable crystal i/o
   } else if(css == 0x0) {
     PMCR |= 0x01;   // enable 16Mhz RC source
   } else if(css == 0x2) {
     PMCR |= 0x02;  // enable 128Hz RC
   }

   delay_ms(1);     // waiting for clock stable

   ctmp = PMCR;
   ctmp &= 0x9f;
   ctmp |= css << 5;
   PMCR = ctmp;    // switch to desired clock source
```

```
    // clock pre-scalar
    CLKPR = 0x80;          // pre-scalar change enable
    CLKPR = cps & 0x7;     // set clock pre-scalar
}
```

低功耗设计

LGT8F08A 支持四种低功耗模式，其中三种为带电低功耗模式：

1. **空闲模式（IDLE）**
2. **ADC 噪声抑制模式**
3. **掉电模式**
4. **断电模式**

软件可以通过配置 SMCR 寄存器并配合使用 SLEEP 指令进入这三种模式；第四种为断电模式，在这种模式下，系统将获得最低的功耗。断电模式只能通过 PC6 引脚或 RTC 唤醒，并且只支持高电平唤醒。软件需要通过配置 RTCSR 寄存器才能进入到断电模式。RTCSR 寄存器的定义，请参考 LGT8F0XA 系列编程手册。

除了定义的四种低功耗模式，软件可以根据具体需求，关闭没有使用的模块；比如没有使用的时钟源，模拟模块以及通过 PRR 寄存器关闭没有使用的数字外设等等。

下面，我们通过一张表格，通过了解系统在掉电模式下的功耗情况，分析那些因素会影响到系统的功耗，从而在实际应用中可以有效的控制。

#	16M RCOSC	128Hz RCOSC	16M Crystal	系统时钟	外部晶振引脚	平均值 (mA)
1	**使能**	**禁止**	**禁止**	**RC8M**	**悬空**	**0.708**
2	**使能**	**禁止**	**禁止**	**RC8M**	**上拉**	**0.708**
3	使能	禁止	使能	RC8M	悬空	1.811
4	使能	禁止	使能	RC8M	上拉	0.792
5	使能	使能	禁止	RC8M	悬空	0.723
6	使能	使能	禁止	RC8M	上拉	0.724
7	使能	使能	使能	RC8M	悬空	1.826
8	使能	使能	使能	RC8M	上拉	0.808
9	**禁止**	**使能**	**禁止**	**RC128H**	**悬空**	**0.263**
10	**禁止**	**使能**	**禁止**	**RC128H**	**上拉**	**0.263**
11	禁止	使能	使能	RC125H	悬空	1.412
12	禁止	使能	使能	RC125H	上拉	0.350
13	禁止	使能	使能	OSC8M	16M	0.866
14	禁止	禁止	使能	OSC8M	16M	0.845

从上述列表中可以得出以下几个特点：

1. *系统工作频率越低，功耗越低。*
   因此，我们可以考虑在必要的时候切换到更低的时钟源并关闭高频时钟源
2. *当引脚悬空时，也会增加额外的功耗*
   因此，应该避免系统上有浮空的引脚，不用的引脚建议使用上拉或下拉的方式。

3. *在大部分使用内部 RC 的应用中，关闭外部晶振将会节省 1mA 左右功耗*

需要注意外部晶振默认是打开的，可以通过 PMCR 寄存器关闭。

LGT8F0XA 系列芯片 I/O 内部均有上拉电阻，外部浮空的 I/O，建议打开内部上拉电阻，内部上拉电路的打开方式与 AVR 一致，通过将 I/O 设置为输入模式，并设置相应的 PORTX 寄存器位。

下面的代码示例如何进入空闲，ADC 噪声抑制以及掉电模式：

代码示例：系统工作模式切换

```
// mode:
//  0x0 : 空闲模式
//  0x1 : ADC 噪声抑制模式
//  0x2 : 掉电模式
void sys_mode(unsigned char mode)
{
   PMCR = 0x01;            // sleep enable
   PMCR |= mode << 1;      // set sleep mode
   asm("sleep");           // enter sleep mode
   asm("nop");             // optional nop
   asm("nop");             // optional nop
}
```

系统进入 sleep 模式后，可以通过有效的唤醒源唤醒。不同待机模式支持的唤醒源也不相同，下面是不同待机模式下支持的唤醒源列表：

睡眠模式	工作时钟					唤醒源				
	clkcpu	clkflash	clkio	clkadc	RTC	INTn PCINTn	WDT	ADC	USART USPI TC	PC6 RTC
空闲模式			√	√		√	√	√	√	
ADC 噪声抑制模式				√		√	√	√		
掉电模式						√	√			
断电模式					√					√

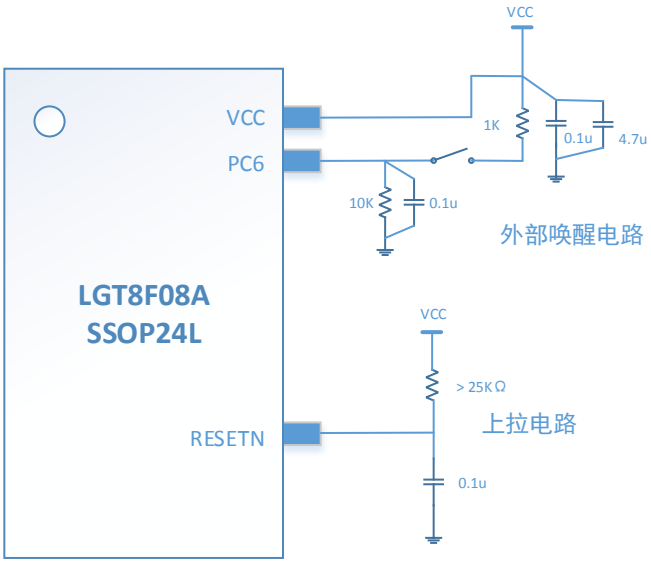
断电模式系统设计

断电模式是系统能够获得最低功耗（**35uA**）的模式，进入断电模式后，系统内部电路都处于断电状态，寄存器和 RAM 的内容也将丢失。从断电模式唤醒的过程，与系统重新上电的过程相同。

由于 LGT8F0XA 系列芯片采用低压低功耗设计，当芯片的任何引脚上有一个足够大的上拉驱动，都会导致整个驱动通过引脚内部的 ESD 二极管反灌到内部的供电电路上，使系统进入工作状态。因此，当我们设计支持断电模式系统时，需要尽量保证在 LGT8F0XA 进入到断电模式后，所有引脚上不要有过于强的上拉或者驱动。对于一些无法避免的上拉电路，建议上拉电阻大于 20KΩ。

当使用调试器与 LGT8F0XA 相连接时，调试器数据线的上拉电阻也会影响系统进入断电模式，因此在使用调试器时，在进入断电模式前，需要断开与调试器的连接。

下面以 LGT8F08A-SSOP24L 封装芯片为例的掉电模式设计电路示例：



与前面介绍的三种待机模式不同，进入断电模式需要通过设置 RTCSR 寄存器实现。

断电模式支持 2 中唤醒源，PC6 高电平唤醒和 RTC 定时唤醒。唤醒源的配置也是通过 RTCSR 寄存器实现。有关 RTC 的配置以及 RTCSR 寄存器的定义，请参考 LGT8F0XA 系列编程手册。

需要特别注意的是，如果从断电模式唤醒后，PC6 需要用作普通 I/O 功能，也需要通过 RTCSR 寄存器关闭 PC6 的唤醒功能。

代码示例：进入断电模式

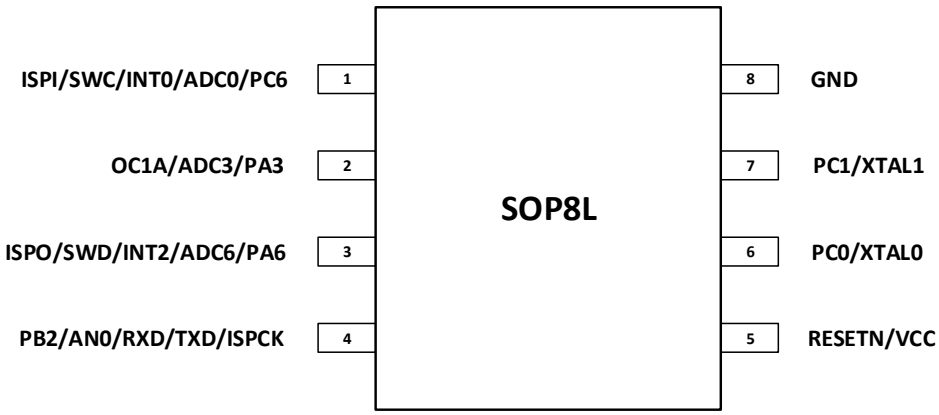
```
// wss : 唤醒源选择
//  0x7 : PC6 或者 RTC 都可以唤醒
//  0x4 : PC6 引脚高电平唤醒
//  0x3 : RTC 定时器唤醒
void power_off(unsigned char wss)
{
   // wait for RTCSR ready
   while((RTCSR & 0x80) != 0x80);
   // clear power/off flag
   RTCSR &= 0xfe;
   // wait for RTCSR ready
   while((RTCSR & 0x80) != 0x80);
   // set wake up sources
   RTCSR |= wss << 1;
   // wait for RTCSR ready
   while((RTCSR & 0x80) != 0x80);
   // set to power/off mode
   RTCSR |= 0x01;
   // optional nops
   asm("nop"); asm("nop");
}
```

LGT8F08A-SOP8 封装的应用开发

SOP8 封装概述

LGT8F08A-SOP8 封装是一种特殊应用封装，利用了 LGT8F0XA 系列芯片低压低功耗的特点，通过普通的 I/O 实现为芯片供电。

LGT8F08A 封装是基于 LGT8F08A-SSOP24 封装的基础上定制，SOP8 的某些引脚，是 SSOP24 封装多个引脚绑定到一起的结果，因此在配置引脚的方向和输出时，需要特别注意，避免对其他引脚产生影响，本章节我们也将用具体的代码示例，给出建议的端口使用方式。



引脚功能说明：

管脚	名称	功能描述
PIN1	ISPI/SWC/INT0/ADC0/PC6	ISPI – ISP 编程数据输入 SWC – SWD 调试接口时钟 INT0 – 外部中断 0 ADC0 – ADC 通道 0 外部输入 PC6 – 可编程 I/O，可用于外部唤醒断电模式
PIN2	OC1A/ADC3/PA3	OC1A – Timer1 比较输出/PWM 输出 ADC3 – ADC 通道 3 外部输入 PA3 – 可编程 I/O
PIN3	ISPO/SWD/INT2/ADC6/PA6	ISPO – ISP 编程数据输出 SWD – SWD 调试接口数据线 INT2 – 外部中断 2 ADC6 – ADC 通道 6 外部输入 PA6 – 可编程 I/O
PIN4	PB2/AN0/RXD/TXD/ISPCK	ISPCK – ISP 编程时钟输入 AN0 – 比较器正端输入(负端可用 1.25V 内部参考) RXD/TXD – UART 数据线，只支持单线 UART
PIN5	RESETN/VCC	RESETN/VCC 外部异步复位输入，此引脚也同时为芯片供电 RESETN 通过内部 ESD 二极管向 VCC 供电，因此芯片的内部

		工作电压与外部供电之间有一个二极管的压降(0.6V)
PIN6	PC0/XTAL0	XTAL0 – 外部晶振输入 PC0 – 可编程 I/O
PIN7	PC1/XTAL1	XTAL1 – 内部振荡器输出 PC1 – 可编程 I/O
PIN8	GND	数字、模拟共用地

SOP8L 在以 SSOP24L 为基础封装，功能配置和 SSOP24 一致。其中 SOP8L 的一些引脚是 SSOP24L 中几个引脚封装到一起的结果。下面表格说明 SOP8L 和 SSOP24L 引脚直接的对应关系，请在使用时特别注意：

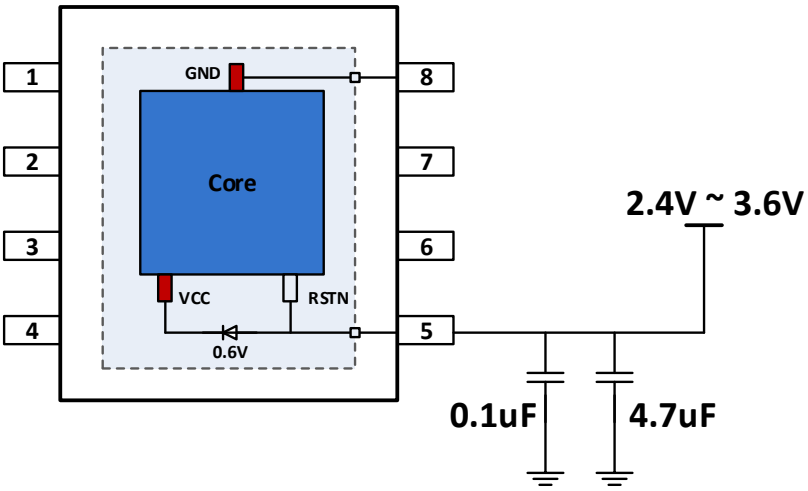
SOP8L	*SSOP24L*
PIN1 *- PA0/INT0/ADC0*	*PIN1 + PIN23* *PIN23 – PC6/WAKUP/SWC/ISPI*
PIN3 *-ISPO/SWD/INT2/ADC6/PA6*	*PIN5 + PIN7* *PIN5 – INT2/ADC5/PA5* *PIN7 – ISPO/SWD/ADC6/PA6*
PIN4 *-PB2/AN0/RXD/TXD/ISPCK*	*PIN12 + PIN15 + PIN16* *PIN12 – ISPCK/AN0/PB2* *PIN15 – PD0/RXD* *PN16 – PD1/TXD*
PIN5 *-RESETN*	*PIN17* *PIN17 – RESETN*

引脚功能复用注意事项：

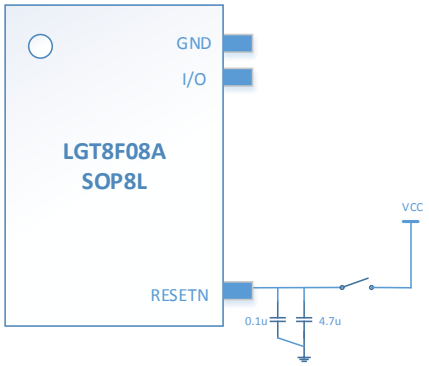
1. SOP8L 本身并没有单独的 VCC 供电，而是使用 RESETN 引脚供电，同时提供了复位和供电功能。因此使用复位功能时需要注意，如果系统中没有其他 I/O 可以供电，复位引脚为低电平时，芯片是断电状态。释放复位信号，芯片将重复一个上电过程。
2. RXD/TXD 引脚被同时复用到 SOP8/PIN4 上，因此 UART 只能工作与单工模式。当系统发送数据时，发送的数据也会同时回环到接收上，软件处理时需要特别注意。
3. 当引脚是多个 I/O 的组合时，配置引脚方向时需要特别注意，避免相互影响。比如 SOP8L 封装的 PIN4，这个引脚其实是 SSOP24 封装的 PIN12/15/16 三个引脚共同组合的，当我们使用 SOP8/PIN4 的 UART 功能是，就必须保证 PB2 为输入端口，否则 PB2 的驱动状态就会影响到 RXD/TXD 正常驱动。
4. SOP8L 封装具备完整的 ISP 接口和 SWD 调试接口，但这些接口都复用了其他的功能，对于 SOP8L 封装，建议在设计 PCB 是留出 ISP 接口的针点（ISP 接口同时也包含了 SWD 调试接口），这样可以确保芯片焊接以后，仍有机会使用 ISP 接口升级程序。另外一种更加建议的方式是 FLASH 驻留自升级的 bootloader，可以通过串口或 VUSB 升级。如果用户有类似的需要，可以直接和我们联系咨询 bootloader 的设计方案。

SOP8 供电与低功耗设计

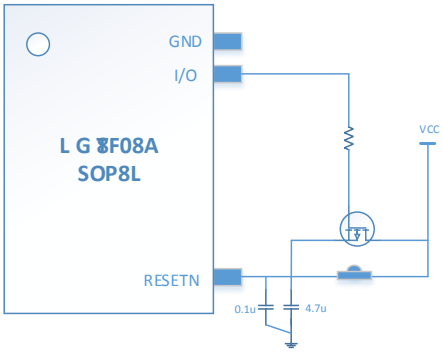
如下图所示，大部分情况下，可以通过 SOP8/PIN5 引脚供电。PIN5 会通过 I/O 内部的 ESD 二极管倒灌的方式给内部的 VCC 供电。因此芯片的实际供电电压和外部供电直接会有一个大约 0.6V 左右的压降。考虑到 LGT8F0XA 系列工作电压范围为 1.8V~3.3V，因此建议外部供电为 2.4V～3.6V 左右。



这种通过 I/O 倒灌供电的方式，使得芯片无法进入到断电模式，因此 SOP8L 的低功耗设计需要另外的方案，下面给出两种比较建议的低功耗设计电路：



被动开关方案



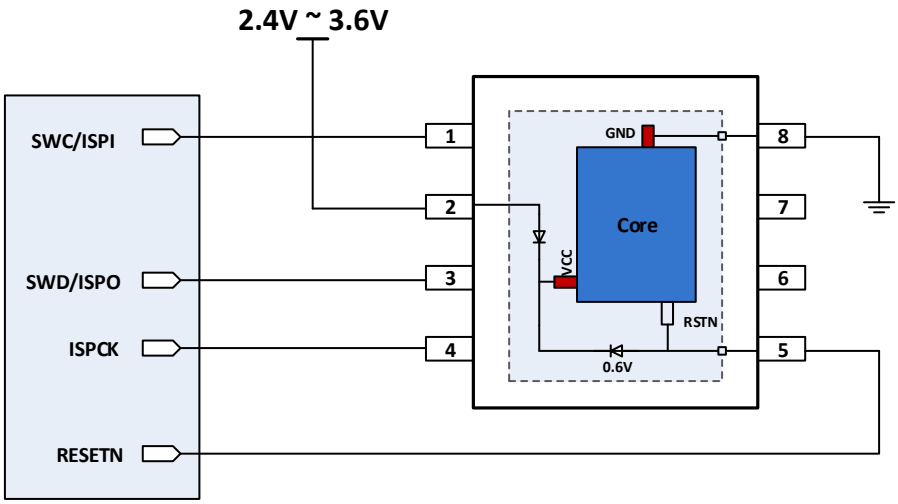
主动开关方案

被动开关方案： 这是一种直接使用开关控制供电的方式，被动的由外部控制工作模式，这种方式简单，对于某些系统还是比较合适；

主动开关方案： 这种方式更加灵活，首先系统有一个触发开关启动，系统启动后，由软件通过 I/O 控制 MOS 管持续供电。当需要进入到断电模式时，可以通过 I/O 口将 MOS 管关闭，在经过电容的延时后，系统进入断电状态。

SOP8 ISP/SWD 接口设计

如下图所示，由于 ISP 接口协议需要控制 RESETN 引脚，因此我们需要使用另外的空闲的 I/O 给系统供电，下图中使用了 SOP8/PIN2。ISP 接口同时包含了 SWD 调试接口。



SOP8 I/O 配置示例

由于 SOP8 引脚的复用特性，强烈建议在配置 I/O 的功能时，配置代码仅仅作用到需要使用的 I/O 上，不要产生任何多余的影响。比如对于一个端口方向的配置，要使用”或”, “与”等操作方式，不能简单的使用赋值操作，当使用复制操作时，要明确每一位对端口的影响。

代码示例：SOP8L 封装芯片的引脚功能配置

```
// 下面是一个简单的例子，配置如下：
// PIN1(PC6)为输入 I/O
// PIN3(PA6)为输出 I/O
void init_port(void)
{
   // 仅仅改变DDRC[6]
   DDRC &= 0xbf;  // DDRC[6] = 0 : INPUT

   // 仅仅改变DDRA[6]
   DDRA |= 0x40;  // DDRA[6] = 1 : OUTPUT
}
```

版本历史

版本	作者	日期	版本日志
1.0.	LGT	2013/12/26	初始版本