如何使用 RTC 工作在低功耗模式下

前言

KF32L/LS 提供多种模式供用户在不同工作场景下使用。包含两种运行模式、两种休眠模式及三种低功耗模式。

KF32L/LS 提供用户实时时间以及日历信息的高精度实时时钟。

本应用笔记将以 KF32L530 为例介绍如何配置进入低功耗停止模式和待机模式。

本应用笔记使用的 KF32 IDE 与 KF32Lxxx 外设固件库及代码例程可以从 ChipON 官方网站 www.chipon-ic.com 下载。

Github：https://github.com/ChipON-FAE-AE
Gitee：https://gitee.com/Cucoo/BSP

目录

1. RTC 特性

实时时钟 (Real Time Clock，RTC) 单元提供给用户实时时间以及日历信息。RTC 单元通过时间寄存器提供时间信息 (秒、分、时、星期、日、月、年)。数据信息由 BCD 码格式进行表示。修改计数器的值可以重新设置系统当前的时间和日期。

RTC 模块可以根据年、月份（闰年、大小月），自动补偿天数；还可以进行夏令时、冬令时补偿。

RTC 的时钟源可以通过软件选择外部低频晶振 EXTLF、内部低频时钟 INTLF 和外部高频晶振的 128 分频。RTC 模块自带高精度的数字时钟校准功能。

RTC 提供两个可编程的闹钟功能及中断，用户可预先在时间闹钟寄存器中设置闹钟日期进行闹铃设置。

2. 停止模式及待机模式的特性

KF32L/LS 系列提供两种停止模式：Stop0 和 Stop1，以及一种待机模式。在这几种休眠模式下，RTC 均能正常工作。停止模式与待机模式休眠特性请参考应用手册 ***AN32001*** 及 KF32L/LS 系列用户手册。

3. RTC 的使用

RTC 的属于备份域的外设，操作 RTC 的需要先打开备份域。请参考应用手册 ***AN32001 的备份域开启及关闭章节***。RTC 的读写控制有个专用配置标志位位于 **RTC_CR** 的 bit3。

RTC 时钟可在任何模式下工作，并且可触发中断将 CPU 从休眠模式唤醒。实时时钟可使用的专用振荡器频率为 32768Hz，通过 **BKP_CTL** 寄存器的 RTCCLKS<1:0>位可以选择实时时钟的时钟源。

RTC 带有时钟校正功能，校正因振荡频率的偏差而导致的时钟的提前或滞后的功能，校准方法请参考 KF32L/LS 系列用户手册。

4. 在休眠模式下 RTC 的保持运行

在休眠模式下，RTC 可以保持运行。RTC 位于备份域中，使能后，在 VBAT 维持供电条件下，即使 VDD 掉电，RTC 依然正常工作。在 VDD 和 VBAT 两者都掉电的时候，才会发生掉电复位。

配置 RTC 在休眠下运行，只需要配置对应的时钟源工作在休眠模式。RTC 的时钟来源为内部低频源或者外部低频时钟源。通过将 **PM_CTL0** 的 LSEEN 位或 LSIEN 位置“1”保持低频时钟。

5. 在休眠模式下通过 RTC 唤醒

在休眠模式下，RTC 有两个唤醒条件，分别为 RTC 的节拍中断及 RTC 的闹钟中断。支持从休眠模式状态下唤醒。RTC 的节拍中断以秒为基本单位进行分频处理。通过 **RTC_C** 的

RTCTT<2:0>配置节拍中间分频。休眠模式下的节拍中断的唤醒是依赖 RTC 节拍输出的上升沿进行唤醒，即一个节拍周期的上升沿唤醒一次。例如 RTC 节拍中断配置为 1/16S,则实际唤醒时间为 0.125S。RTC 的中断属于外部中断，在配置 RTC 的中断线时，需要将改中断线配置为上升沿触发。唤醒条件配置完毕后，通过配置 **PM_CTL2** 的 bit<12：15>使能 RTC 节拍唤醒功能。

6. RTC 低功耗配置相关寄存器

低功耗的寄存器都存放于备份域，操作之前需要打开备份域使能 **OSC_CTL0** 的 bit0 位，备份域退出复位 **PM_CTL0** 的 bit22 位，备份域数据区允许读写 **PM_CTL0** 的 bit7 位。

RTC 的节拍中断需要打开 RTC 配置寄存器节拍输出使能 **RTC_CR** 的 bit23 位，时间节拍设置 **RTC_CR** 的 RTCTT<2:0>。

OSC_CTL0 振荡器控制寄存器 0（偏移地址：0x000）

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
复位值	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W						R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W				R/W	R/W	R/W	R/W	R/W		R/W		R/W	R/W	R/W	R/W	R/W
位名						LFCKDIV <2:0>			HFCKDIV <3:0>					SCKDIV <2:0>						PLLCKS	HFCKS <2:0>			HFCKEN		LFCKS		LFCKEN		SCKS <2:0>		PMWREN

PM_CTL0 功耗模式控制寄存器 0（偏移地址：0x000）

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
复位值	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W		R/W	R/W	R/W		R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
位名	IOLATCH		LSIEN	LSEEN	LSEEXEN		MRBGEN	LDO18EN		BKPREGCLR	IWDTCLR	DPRAMASEL	LPRAMSEL	LP4MEN	LSECONF	IWDTRMSEL	PORDELAYSEL	BKPPORDELAYSEL	PHERIIOSEL					HSTEN	BKPWR	OCALLOCK	LPREN	NRSTEN	MEMSEL	LPMS2	LPMS1	LPMS0

PM_CTL2 功耗模式控制寄存器 2（偏移地址：0x014）

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
复位值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
位名	WKP5EN	WKP4EN	WKP3EN	WKP2EN	WKP1EN	WKP5P	WKP4P	WKP3P	WKP2P	WKP1P	VBKPOREN	VBKPORSHD	PMC1E	PMC1C		CT_RESET	RTCTWEN	RTCTPEN	RTCTSEL1	RTCTSEL0	VF12EN	VF121NF1	VF121NF0	POR18SHD	USART0CLKLPEN	LCDCLKLPEN	CCP0CLKLPEN	CAN0CLKLPEN	USART0LPEN	LCDLPEN	CCP0LPEN	CAN0LPEN

7. 快速唤醒如何降低功耗

从停止模式或待机模式状态唤醒，单片机需要从头开始执行，在从复位处运行期间，单片机默认的时钟为内部高速时钟 16M 的 128 分频，即 0.125M。此时代码执行速度极慢，若前面代码数据较多，则会占用大量的时间在启动过程。如图 5 所示，RTC 在每次唤醒时，有一段电流较低，但时间较长的时间区间。若唤醒频次较高则功耗将会大量损耗在此处。

单片机完整的上电流程如下图 1 所示。其中，Power12 及 Power18 为内部部件上电时间；ROM 启动为加载校准信息，如 Flash 校准信息，晶振校准信息等。

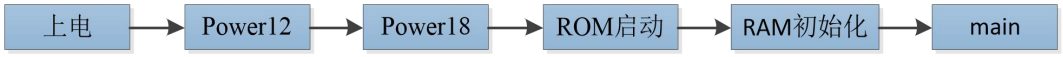

图 1：完整的上电流程

正常的停止模式或待机模式休眠唤醒流程如下图 2 所示。唤醒后单片机从 Power12 开始复位运行。工作电流测量图如图 5，RTC 定时唤醒休眠。电流波形上可以看出总共分为三段：

1.唤醒后有一段长时间的低频工作时间。此时时间端在 Power12 至 main 中的时钟初始化之前。

2.之后有一处脉冲电流为时间升频后的大电流，此时为时钟初始化之后，电流随着主频的提升而增加，执行速度也得到加快。

3.进入休眠，功耗降低。占用整个“唤醒-休眠”流程的绝大部分时间。

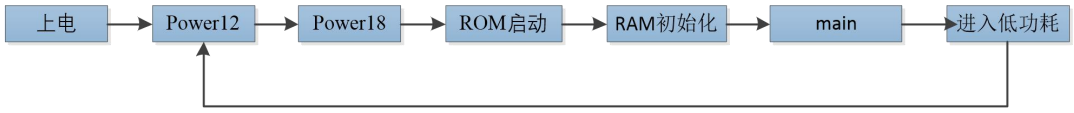

图 2：正常的休眠流程

优化后的停止模式或待机模式休眠唤醒流程如下图 3 所示。优化后的工作电流测量图如图 6 所示，RTC 定时唤醒并休眠，与上述正常模式的区别在于：

1.启动时，先执行到 main 函数，进行时钟的初始化，再执行 RAM 变量的初始化。RAM 变量初始化的时间与参数变量的数量成正比，变量越多初始化时间越长。

2.更改启动方式，从 Flash 启动。从 Flash 启动速度比 ROM 启动更快。

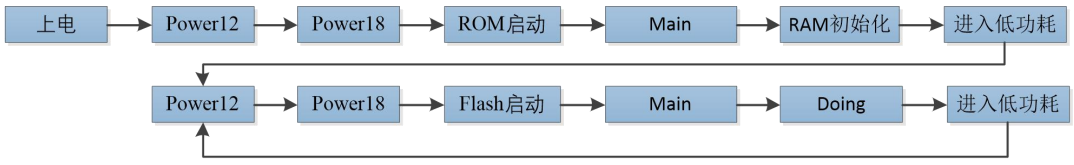

图 3：优化后的休眠流程

8. 低功耗优化操作

1.修改 vector.c 文件中“_start”函数中“**startup**”更改为“**main**”，上电启动后从“**main**”处开始执行。

```
298 const VectorEnter _start __attribute__((section(".text"))) =
299 {
300         &__initial_sp,
301         startup,
302         V2_0x00000008_VectorFunction,
303         V3_0x0000000C_VectorFunction,
304         V4_0x00000010_VectorFunction,
305         V5_0x00000014_VectorFunction,
306         V6_0x00000018_VectorFunction,
307         0,
308         V8_0x00000020_VectorFunction,
309         V9_0x00000024_VectorFunction,
310         V10_0x00000028_VectorFunction,
311         V11 0x0000002C VectorFunction,
```

2.删除 startup.c 文件中“**HWREG(0x40000000)=0;**”，不对主频做改变。

3.删除 startup.c 文件中“**main();**”，“**startup**”结束后不跳转。

```
19 void startup()
20 {
21     unsigned int *s,*begin,*end;
22 #ifdef  Project_Type__cplusplus
25 //##############         init work for the chip          ##############//
26 //  HWREG(0x40000000)=1;
27     HWREG(0x40000000)=0;
28 //############## init variable who have initialization   ##############//
29     s = (unsigned int*)&__text_end__;
30     begin = (unsigned int*)&__data_start__;
31     end = (unsigned int*)&__bss_start__;
32     while(begin < end)
33         *begin++ = *s++;
34 //############## init class who have initialization(C++) ##############//
35 #ifdef Project_Type__cplusplus
44 //############## init variable who have no initialization    ##############//
45 #if 1    //  0 not init this type variable
46     begin = (unsigned int*)&__bss_start__;
47     end = (unsigned int*)&__bss_end__;
48     while(begin < end)
49         *begin++ = 0;
50 #endif
51 //############## begin to run main function  ##############//
52     main();
53 }
```

9. 功耗测量图示

下图 4 为 STANDBY 休眠模式下保持 RTC 唤醒功能电流示意图：

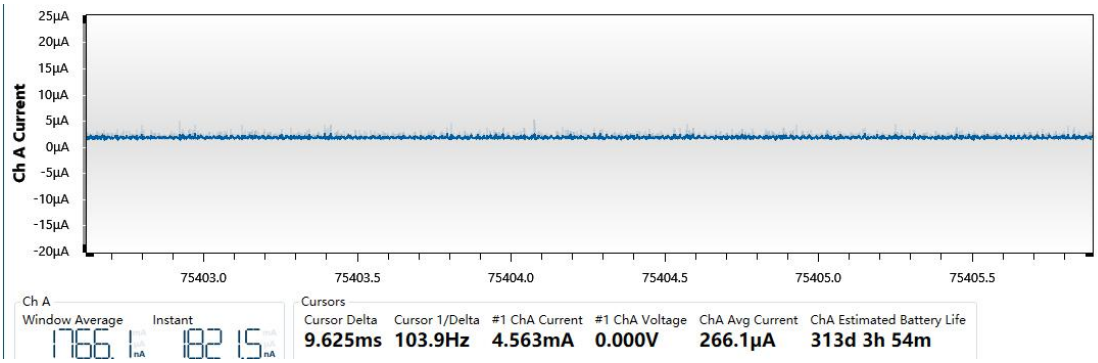

图 4：STANDBY 休眠模式电流

下图 5 为 STANDBY 休眠模式下使用 500msRTC 定时唤醒，唤醒后继续休眠

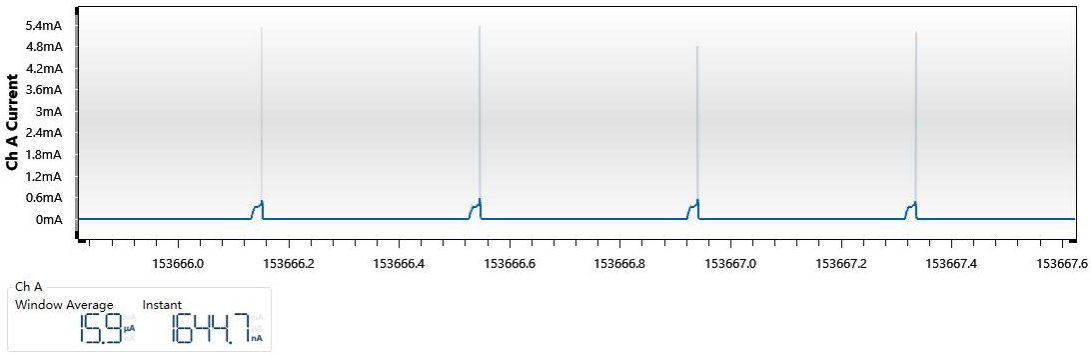

图 5：STANDBY 休眠模式 RTC 唤醒

下图 6 为 STANDBY 休眠模式下使用 500msRTC 定时唤醒，唤醒后继续休眠。与图 5 的区别在与对于启动流程的优化。可以看出，代码优化后电流消耗明显减少了很多。

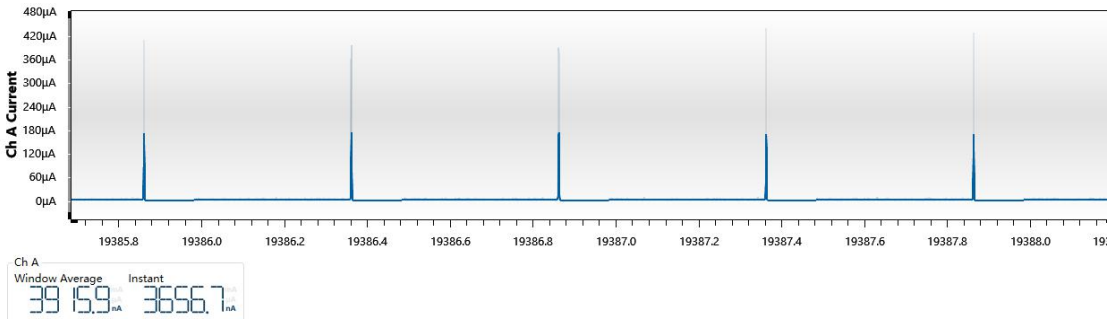

图 6：启动优化后 STANDBY 休眠模式 RTC 唤醒

10. 软件流程图

例程代码 KF32L530_StandbyMode_With_RTC 的软件流程图如下图 7。置于文章结束位置。

本应用笔记使用的 KF32 IDE 与 KF32Lxxx 外设固件库及代码例程可以从 ChipON 官方网

站 www.chipon-ic.com 下载。

11. 版本历史

文档版本历史

日期	版本	变更
2021 年 1 月 21 日	V1.0	初始版本。

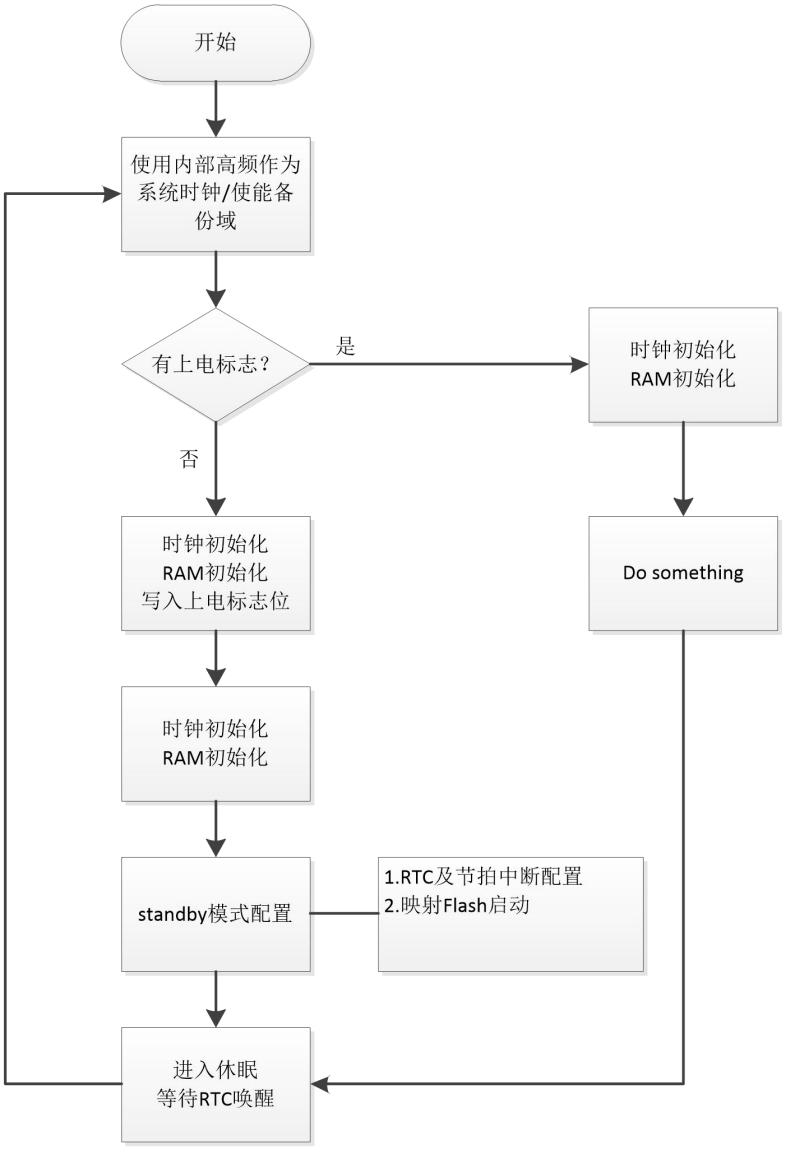

图 7：STANDBY 休眠模式下 RTC 快速唤醒软件流程图