



Technology Innovator

**Puya**

**AN1033**

**应用笔记**

PY32F071\_PY32F072 的应用

注意事项

## 前言

PY32F071\_PY32F072 系列微控制器采用高性能的 32 位 ARM® Cortex®-M0+ 内核，宽电压工作范围的 MCU。嵌入高达 128 Kbytes flash 和 16 Kbytes SRAM 存储器，最高工作频率 72 MHz。包含多种不同封装类型多款产品。

本应用笔记将帮助用户了解 PY32F071\_PY32072 各个模块应用的注意事项，并快速着手开发。

表 1. 适用产品

类型	产品系列
微型控制器系列	PY32F071、PY32F072

## 目录

1. PWR 使用注意事项 .....	3
2. ADC 上电校准 .....	3
3. ADC 使用注意事项 .....	4
4. TTCAN 系统时钟配置注意事项 .....	5
5. COMP 硬件设计 .....	5
6. CRC 使用注意事项 .....	5
7. DAC 使用注意事项 .....	6
8. DMA 使用注意事项 .....	6
9. GPIO 使用注意事项 .....	6
10. I2C 使用注意事项 .....	6
11. LCD 使用注意事项 .....	6
12. LPTIM 使用注意事项 .....	7
13. RCC 使用注意事项 .....	8
14. SPI 收发注意事项 .....	8
15. TIMER 使用注意事项 .....	8
16. USB 使用注意事项 .....	8
17. FLASH 使用注意事项 .....	8
18. Option 使用注意事项 .....	9
19. I2S 功能注意事项 .....	10
20. 版本历史 .....	11
附录 1 .....	12
1.1 PY32F071/PY32F072 低功耗模式下, 定时唤醒喂狗例程(LL 库) .....	12
1.2 PY32F071/PY32F072 低功耗模式下, 定时唤醒喂狗例程(HAL 库) .....	17
附录 2 .....	21
2.PY32F071 读取 information 区域中存放的 Vreferint 1.2 实测值(具体地址见 3.3) .....	21

## 1. PWR 使用注意事项

- 为了提供系统稳定性一定要使能看门狗功能；
- 推荐客户在 Option 中使能看门狗并根据实际情况软件设置看门狗溢出时间；
- 一旦使能看门狗，软件无法关闭，所以在低功耗模式下，需使用 LPTIM 定时唤醒，对看门狗进行喂狗；(例程参考附录 1)
- Sleep 模式下使用事件唤醒时，若 EXTI 模块时钟与 CPU 时钟为一个时钟源时，不得使用分频。

## 2. ADC 上电校准

### 2.1 注意事项

- 当 ADC 的工作条件发生改变时 (VCC 改变是 ADC offset 偏移的主要因素，温度改变次之)，推荐进行再次校准操作。
- 第一次使用 ADC 模块前，必须增加软件校准流程。

### 2.2 操作流程

- 使能 ADC 时钟，ADCEN=1；
- 初始化 ADC；
- ADC 校准。

### 2.3 代码示例

```
static void APP_AdcConfig()
{
  LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_ADC1);           //使能 ADC1 时钟

  if (LL_ADC_IsEnabled(ADC1) == 0)
  {
    LL_ADC_StartCalibration(ADC1);                                 //使能校准
    #if (USE_TIMEOUT == 1)
      Timeout = ADC_CALIBRATION_TIMEOUT_MS;
    #endif
    while (LL_ADC_IsCalibrationOnGoing(ADC1) != 0)
    {
      #if (USE_TIMEOUT == 1)                                       //检测校准是否超时
        if (LL_SYSTICK_IsActiveCounterFlag())
        {
          if(Timeout-- == 0)
          {
            }
          }
        }
      }
    }
  }
}
```

```
#endif
}
LL_mDelay(1);
}
}
```

### 3. ADC 使用注意事项

#### 3.1 ADC 软件配置

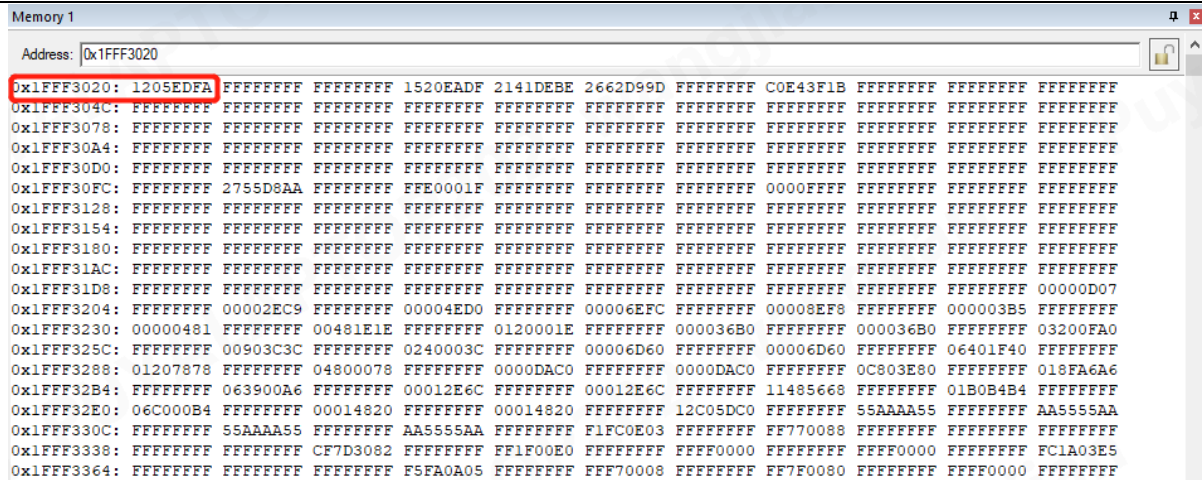
- 在采样内部通道(通道 16-通道 23)时, 需要设置当前使用通道的前一个通道采样周期, 而且需要设置采样周期一致, 例如使用通道 18 采样周期 239.5, 则通道 17 也需要设置采样周期 239.5; (C 版本已修复)
- 因为 ADC 在全部通道转换完成后才会一个 EOC 标志, 所以没办法使用非 DMA 方式的多通道采样; (可设置非连续模式使能, 一个通道转换即有一个 EOC 标志)
- ADC 参考电压 ( $V_{CC}$  或  $V_{REFBUF}$ ) 低于 2 V 时无法校准; (C 版本已修复)
- TIM\_CC 触发注入通道单次采样异常, 不建议使用; (C 版本已修复)
- 内部温度传感器通道 (Tsensor) 无法使用; (C 版本已修复)
- ADC->SR.Start=1 之前需确保 ADC->SR.EOC=0, 否则 ADC 使能失败;
- TIMER 触发 ADC 采样时, 需保证 ADC 工作时钟大于等于 TIMER 时钟; (C 版本前不支持 APB 分频, C 版增加 TIMER 倍频选择, APB 分频后关闭 TIMER 倍频)
- ADC 注入模式无法使用;
- 系统时钟 8 M 时且 ADC CLOCK 两分频时, ADC 无法校准;
- ADC 时钟需要配置到 16 MHz 以下, 确保 ADC 采样精度;
- 在配置通道 16 (OPA3\_IN) 的采样时间时, 需配置通道 0 与通道 16 为相同的采样时间;
- 使用 TIMER\_CC/TIMER\_TRGO 触发 ADC 转换, ADC 时钟不能 8 分频;
- 当时钟 AHBCLK/APBCLK $\geq$ 4 时, 不能设置 EXTI\_11 触发 ADC 规则转换, 不能使用 EXTI\_15 触发 ADC 注入转换。 (C 版本已修复)

#### 3.2 ADC 硬件配置

- ADC 通道电压不能高于  $V_{CC}+0.3$  V(即使 ADC 通道未配置为 AD 功能), 否则 ADC 采样不准。

#### 3.3 Vreferint 1.2 V

- 芯片的 Vreferint 1.2 V 实测值放置在 FLASH 中的 information 区域(0x1FFF3020)。(高 16 位是实际值, 低 16 位是反码), 读取 Vreferint 1.2 V 的程序见附录 2:



- 在采样内部参考电压 1.2 V 的时候，通过 ADC 采样时间转换公式算出来的结果至少需要 20 us，方法如下：

- 降低分辨率。
- 降低ADC的时钟频率。
- 提高ADC采样周期。

总转换时间计算如下：

$$t_{CONV} = \text{采样时间} + (\text{转换分辨率} + 0.5) \times \text{ADC 时钟周期}$$

例如：

当 ADC\_CLK = 12MHz，分辨率为 12 位，且采样时间为 239.5 个 ADC 时钟周期：

$$t_{CONV} = (239.5 + 12.5) \times \text{ADC 时钟周期} = 252 \times \text{ADC 时钟周期} = 21 \text{ us}$$

#### 4. TTCAN 系统时钟配置注意事项

- 系统时钟不得低于 CAN 模块工作时钟的 0.75 倍，CAN 的工作时钟是 20 MHz，所以系统时钟不得低于 15 MHz。

#### 5. COMP 硬件设计

- 当比较器的 COMP<sub>x</sub>\_INM 输入信号为内部的模拟电压源时(例如 VREFINT, TSVIN, VREF1P2)，外部输入通道 VINP 需要加一个电容(1 nF)到地。

#### 6. CRC 使用注意事项

- 禁止使用 DMA 模式。

- 禁止连续对 CRC\_DR 寄存器进行操作，建议使用库，库中有规避。

## 7. DAC 使用注意事项

- 在三角波波形生成和噪声波形生成模式下，DHR 会维持上一个周期的值。

## 8. DMA 使用注意事项

- 使用 DMA 传输搬运数据时，需等 DMA 传输完成后才能关闭 DMA，否则需强制 Reset 并重新初始化 DMA 才能正常使用。

## 9. GPIO 使用注意事项

- 不得使用 DMA GPIO 模式。(C 版本已修复)

## 10. I2C 使用注意事项

- 使用 DMA 进行 I2C 写数据搬运时，需配置 DMA 源地址和目标地址后再使能 I2C 的 DMA\_EN。

## 11. LCD 使用注意事项

- 向同一个 LCD\_RAMx 寄存器写数据时，需要在 2 个 lcd clk 周期内写完数据，之后等待 2 个 pclk+1 个 lcd clk 周期后才能继续写数据；(参考如下)

```
#define Delay 40*2
LCD_HandleTypeDef LcdHandle;
int main()
{
    RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;
    HAL_LCD_Write(&LcdHandle, LCD_RAM_REGISTER0, 0xf0f0f0f);
    APP_DelayNops(RatioNops);/*延迟 2 个 pclk+1 个 lcd clk 周期，约为 80us
    HAL_LCD_Write(&LcdHandle, LCD_RAM_REGISTER0, 0xf0f0f0f);
}
static void APP_DelayNops(uint32_t Nops)
{
    for(uint32_t i=0; i<Nops;i++)
    {
        __NOP();
    }
}
```

- 写 RAM 后需等待 2 个 lcd clk 周期才能进入 Stop 模式; (参考如下)

```
#define Delay 40*2
LCD_HandleTypeDef LcdHandle;

int main()
{
    RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;
    HAL_LCD_Write(&LcdHandle, LCD_RAM_REGISTER0, 0x0f0f0f);
    APP_DelayNops(RatioNops);/*延迟 2 个 lcd clk 周期, 约为 80us
}
while(1)
{
    /* Suspend SysTick interrupt */
    HAL_SuspendTick();
    /* Enter Stop Mode and Wakeup by WFI */
    HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,
PWR_STOPENTRY_WFI);
    /* Resume SysTick */
    HAL_ResumeTick();
}
static void APP_DelayNops(uint32_t Nops)
{
    for(uint32_t i=0; i<Nops;i++)
    {
        __NOP();
    }
}
```

## 12. LPTIM 使用注意事项

- 当 LPTIM 使用 RCC\_CCIPR->LPTIMSEL 来选择 PCLK 为时钟源时, 预分频不能设置为 1, 否则 LPTIM 有概率性运行异常;
- LPTIM 使用 RSTARE 功能时, 两次读取 CNT 寄存器的间隔要满足 4 个 LSI 时钟;

### 12.1 LPTIM 连续模式

- LPTIM 连续模式每次进入 Stop 前必须清 ARRMCF 并需等待 1 个 LSI 时钟周期\*PSC 系数。(约需 40 us\*PSC, 包含程序执行时间)
- 改 LPTIM 的重载值, 需等待 4 个 LSI 时钟周期\*PSC 系数。(约需 160 us\*PSC, 包含程序执行时间)

### 12.2 LPTIM 单次模式

- LPTIM 单次模式从 Stop 唤醒, 再次进入 Stop 前需等待 4 个 LSI 时钟周期。(约需 160 us, 包含程序执行时间)
- 改变重载值 LPTIM\_ARR 时, 需等待 4 个 LSI 时钟周期。(约需 160 us, 包含程序执行时间)

## 13. RCC 使用注意事项

- APB 分频系数大于 1 时，模块复位后需增加 (n+2) 个 \_\_nop() 空指令才能对模块寄存器进行读写操作，n 为 APB 分频系数；
- Stop 唤醒会将 HSIDIV 恢复到默认值，若实际使用 HSIDIV≠0 唤醒后需重新配置；
- HCLK 分频 HPRE 寄存器位不能从 8 切换到 1-7，否则会挂住时钟；
- 不支持通过 RCC\_CR->HSIDIV 进行分频。(仅有 C 版本和 E 版本存在此问题)

## 14. SPI 收发注意事项

- SPI 做从机发送时，需再每一帧数据发送前先将 SPI\_EN 清 0 再将 SPI\_EN 置 1；
- SPI 通信时，SPI->SR.BSY 位在最后一个时钟期间被清除，Polling 模式时，下一帧数据更新需确保上一帧数据传输完成。

## 15. TIMER 使用注意事项

- 使能 CC 中断时，对应的分频系数 PSC 不得高于 80；
- TIMER 预分频必须设置为 1，否则会导致 MCU 反复进中断；
- 禁止使用刹车功能；(C 版本已修复) (在 PWM 极性和空闲状态一致，且 PWMN 极性和空闲状态不一致时，禁止使用刹车功能)
- TIMER3 的 ETR 功能禁止使用。(C 版本已修复)

## 16. USB 使用注意事项

- 使能 USB 时钟门控前，需先开启 USB 48 MHz 时钟；
- USB 使用 DMA 通讯时，同时只能配置一个通道进行 DMA 通讯；
- USB 在使能端点中断前需要清端点 FIFO。

## 17. FLASH 使用注意事项

- FLASH 只支持 Page 擦和 Page 写，一个 Page 是 256 字节，起始地址只能 Page 对齐；(如起始地址 0x08005000, 0x08005100 等)
- 每次 Page 写之前必须先 Page 擦。



## 18. Option 使用注意事项

- 量产时，Option 操作需要在烧写器选项字节中配置，并把程序中操作 Option 的函数屏蔽；
- 建议客户程序使能写保护，写保护在 Option 中设置，具体步骤如图 18-1、图 18-2 所示；



图 18-1 轩微操作 Option 写保护

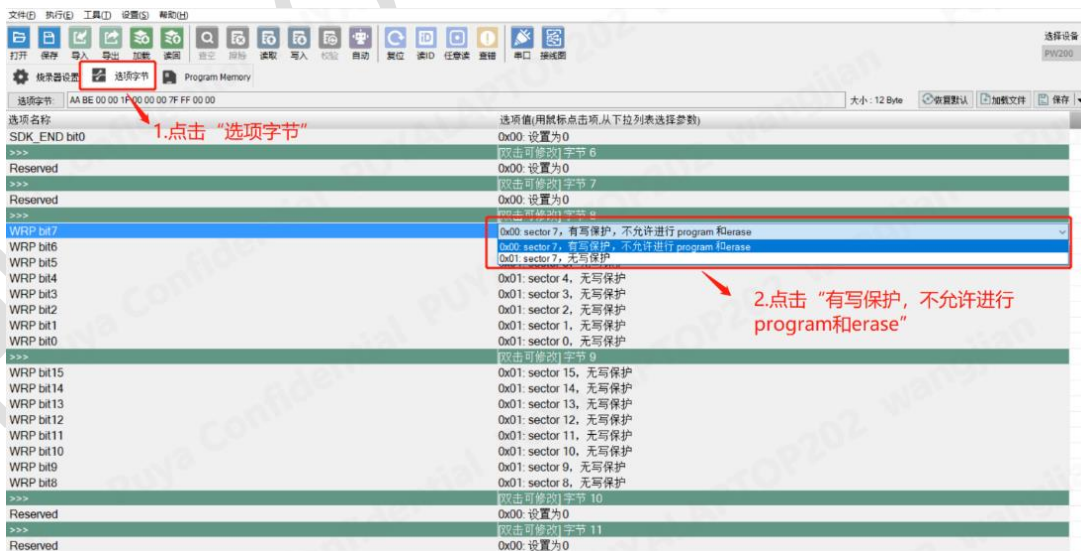


图 18-2 创芯工坊操作 Option 写保护

- 烧写器配置 Option 时，需勾选智能复位功能/编程后重启芯片(烧写器均有类似选项需要勾选)，操作步骤如图 18-3、图 18-4 所示。



图 18-3 轩微操作“智能复位”



图 18-4 创芯工坊操作勾选“编程后重启芯片”

## 19. I2S 功能注意事项

- 在 8 M 系统时钟下，Fs=32 KHz 和 22.05 KHz 无法使用。

## 20. 版本历史

版本	日期	更新记录
V1.0	2023.06.15	初版
V1.1	2024.07.01	1.添加 PWR、修改 4、6、14、15 章节，新增 5、8、9、11、12、13、16、17、18、19、20、附录 1 章节
V1.2	2024.08.25	合并 2、3、4 章节，合并第 11、12 章节 新增 15 章节，修改 2、8、11、13 章节
V1.3	2024.11.04	新增第 2 章 Vreferint 1.2V 内容 合并 2、3、4 章节，合并第 14、15 章节 修改第 7、17 章节内容
V1.4	2025.06.16	修改 LPTIM 章节内容
V1.5	2025.07.22	修改 ADC、RCC、LPTIM 章节内容
V1.6	2025.10.21	增加 ADC 上电校准、ADC、RCC、USB、SPI、I2S 模块内容



Puya Semiconductor Co., Ltd.

### 声 明

普冉半导体(上海)股份有限公司 (以下简称: "Puya" ) 保留更改、纠正、增强、修改 Puya 产品和/或本文档的权利, 恕不另行通知。用户可在下单前获取产品的最新相关信息。

Puya 产品是依据订单时的销售条款和条件进行销售的。

用户对 Puya 产品的选择和使用承担全责, 同时若用于其自己或指定第三方产品上的, Puya 不提供服务支持且不对此类产品承担任何责任。

Puya 在此不授予任何知识产权的明示或暗示方式许可。

Puya 产品的转售, 若其条款与此处规定不一致, Puya 对此类产品的任何保修承诺无效。

任何带有 Puya 或 Puya 标识的图形或字样是普冉的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代并替换先前版本中的信息。

普冉半导体(上海)股份有限公司 - 保留所有权利

## 附录 1

### 1.1 PY32F071/PY32F072低功耗模式下，定时唤醒喂狗例程(LL库)

```

#define Delay          40*4
int main(void)
{
    /* Configure system clock */
    APP_SystemClockConfig();

    /* Initialize LED */
    BSP_LED_Init(LED_GREEN);

    APP_GpioConfig();
    /* Initialize button */
    BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_GPIO);

    /* Configure EXTI Line29 corresponding to LPTIM as interrupt wake-up mode */
    LL_EXTI_EnableIT(LL_EXTI_LINE_29); /* Enable EXTI Line 29 interrupt wakeup */
    LL_EXTI_DisableEvent(LL_EXTI_LINE_29); /* Disable EXTI Line 29 event wakeup */

    /* Configure LPTIM clock source as LSI */
    APP_ConfigLptimClock();
    APP_LwdgConfig();
    /* Initialize LPTIM */
    LPTIM_InitStruct.Prescaler = LL_LPTIM_PRESCALER_DIV128; /* Prescaler: 128 */
    LPTIM_InitStruct.UpdateMode = LL_LPTIM_UPDATE_MODE_IMMEDIATE; /* Immediate
update mode */
    if (LL_LPTIM_Init(LPTIM, &LPTIM_InitStruct) != SUCCESS)
    {
        APP_ErrorHandler();
    }

    /* Turn on LED */
    BSP_LED_On(LED_GREEN);

    /* Wait for the button to be pressed */
    while (BSP_PB_GetState(BUTTON_USER) != 0)
    {
    }

    /* Turn off LED */
    BSP_LED_Off(LED_GREEN);

    /* Calculate the value required for a delay of macro-defined(Delay) */
    RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;

    /* Configure LPTIM and enable interrupt */
    APP_ConfigLptim();

    while (1)
    {
        /* LPTIM must be disabled to restore internal state before next time enter Stop mode */
        LL_LPTIM_Disable(LPTIM);

        /* Wait at least three LSI times for the completion of the disable operation */
    }
}

```

```

APP_DelayNops(RatioNops);

/* Enable LPTIM */
LL_LPTIM_Enable(LPTIM);

/* Set auto-reload value */
LL_LPTIM_SetAutoReload(LPTIM, 51);

/* Start in once mode */
LL_LPTIM_StartCounter(LPTIM, LL_LPTIM_OPERATING_MODE_ONESHOT);

/* Enable Stop mode */
APP_EnterStop();

/* PB1 toggle */
LL_GPIO_TogglePin(GPIOB, LL_GPIO_PIN_1);
}
}
/**
 * @brief Configure LPTIM clock
 * @param None
 * @retval None
 */
static void APP_ConfigLptimClock(void)
{
/* Enable LSI */
LL_RCC_LSI_Enable();
while(LL_RCC_LSI_IsReady() != 1)
{
}

/* Select LPTIM clock source as LSI */
LL_RCC_SetLPTIMClockSource(LL_RCC_LPTIM1_CLKSOURCE_LSI);

/* Enable LPTIM clock */
LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_LPTIM1);
}
/**
 * @brief Delayed by NOPS
 * @param None
 * @retval None
 */
static void APP_IwdgConfig(void)
{
/* Enable LSI */
LL_RCC_LSI_Enable();
while (LL_RCC_LSI_IsReady() == 0U) {}

/* Enable IWDG */
LL_IWDG_Enable(IWDG);

/* Enable write access */
LL_IWDG_EnableWriteAccess(IWDG);

/* Set IWDG prescaler */
LL_IWDG_SetPrescaler(IWDG, LL_IWDG_PRESCALER_32);

```

```

/* Set watchdog reload counter */
LL_IWDG_SetReloadCounter(IWDG, 1024); /* T*1024=1s */

/* IWDG initialization*/
while (LL_IWDG_IsReady(IWDG) == 0U) {}

/* Feed the watchdog */
LL_IWDG_ReloadCounter(IWDG);
}
static void APP_DelayNops(uint32_t Nops)
{
    for(uint32_t i=0; i<Nops;i++)
    {
        __NOP();
    }
}

/**
 * @brief GPIO configuration program
 * @param None
 * @retval None
 */
static void APP_GpioConfig(void)
{
    /* Enable GPIOB clock */
    LL_IOP_GRP1_EnableClock(LL_IOP_GRP1_PERIPH_GPIOB);

    /* Configure PB1 in output mode */
    LL_GPIO_SetPinMode(GPIOB, LL_GPIO_PIN_1, LL_GPIO_MODE_OUTPUT);
}

/**
 * @brief Configure LPTIM
 * @param None
 * @retval None
 */
static void APP_ConfigLptim(void)
{
    /* Enable LPTIM1 interrupt */
    NVIC_SetPriority(TIM6_LPTIM1_DAC_IRQn, 0);
    NVIC_EnableIRQ(TIM6_LPTIM1_DAC_IRQn);

    /* Enable LPTIM ARR match interrupt */
    LL_LPTIM_EnableIT_ARRM(LPTIM);
}

/**
 * @brief Enter Stop mode
 * @param None
 * @retval None
 */
static void APP_EnterStop(void)
{
    /* Enable PWR clock */
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);
}

```

```

/* VCORE = 0.8V when enter Stop mode */
LL_PWR_SetRegulVoltageScaling(LL_PWR_REGU_VOLTAGE_0P8V);

/* Enable Low Power Run mode */
LL_PWR_EnableLowPowerRunMode();

/* Enter DeepSleep mode */
LL_LPM_EnableDeepSleep();

/* Request Wait For interrupt */
__WFI();

LL_LPM_EnableSleep();
}

/**
 * @brief System clock configuration function
 * @param None
 * @retval None
 */
static void APP_SystemClockConfig(void)
{
    /* Enable HSI */
    LL_RCC_HSI_Enable();
    while(LL_RCC_HSI_IsReady() != 1)
    {
    }

    /* Set AHB prescaler */
    LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);

    /* Configure HSISYS as system clock source */
    LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSISYS);
    while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSISYS)
    {
    }

    /* Set APB1 prescaler*/
    LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
    LL_Init1msTick(8000000);

    /* Update system clock global variable SystemCoreClock (can also be updated by calling
    SystemCoreClockUpdate function) */
    LL_SetSystemCoreClock(8000000);
}

/**
 * @brief LPTIM interrupt callback program
 * @param None
 * @retval None
 */
void APP_LptimIRQCallback(void)
{
    if((LL_LPTIM_IsActiveFlag_ARRM(LPTIM) == 1) && (LL_LPTIM_IsEnabledIT_ARRM(LPTIM) ==
    1))
    {
        /* Clear autoreload match flag */
        LL_LPTIM_ClearFLAG_ARRM(LPTIM);

        LL_IWDG_ReloadCounter(IWDG);
    }
}

```

```
    /* LED Toggle */
    BSP_LED_Toggle(LED_GREEN);
}
}

/**
 * @brief This function is executed in case of error occurrence.
 * @param None
 * @retval None
 */
void APP_ErrorHandler(void)
{
    /* Infinite loop */
    while (1)
    {
    }
}
```



## 1.2 PY32F071/PY32F072低功耗模式下，定时唤醒喂狗例程(HAL库)

```

#define Delay          40*4
int main(void)
{
    EXTI_ConfigTypeDef    ExtiCfg = {0};

    /* Reset of all peripherals, Initializes the Systick */
    HAL_Init();

    /* Clock configuration */
    APP_RCCOscConfig();

    /* Initialize LED */
    BSP_LED_Init(LED_GREEN);

    APP_GpioConfig();

    /* Initialize button */
    BSP_PB_Init(BUTTON_USER,  BUTTON_MODE_GPIO);

    /* LPTIM initialization */
    LPTIMConf.Instance = LPTIM1; /* LPTIM1 */
    LPTIMConf.Init.Prescaler = LPTIM_PRESCALER_DIV128; /* Prescaler: 128 */
    LPTIMConf.Init.UpdateMode = LPTIM_UPDATE_IMMEDIATE; /* Immediate update mode */
    /* Initialize LPTIM */
    if (HAL_LPTIM_Init(&LPTIMConf) != HAL_OK)
    {
        APP_ErrorHandler();
    }

    /* Configure EXTI Line as interrupt wakeup mode for LPTIM */
    ExtiCfg.Line = EXTI_LINE_29;
    ExtiCfg.Mode = EXTI_MODE_INTERRUPT;
    /* The following parameters do not need to be configured */
    /* ExtiCfg.Trigger */
    /* ExtiCfg.GPIOSEL */
    HAL_EXTI_SetConfigLine(&ExtiHandle,  &ExtiCfg);

    /* Enable LPTIM1 interrupt */
    HAL_NVIC_SetPriority(TIM6_LPTIM1_DAC_IRQn,  0,  0);
    HAL_NVIC_EnableIRQ(TIM6_LPTIM1_DAC_IRQn);
    APP_IwdgConfig();
    /* Turn on LED */
    BSP_LED_On(LED_GREEN);

    /* Wait for the button to be pressed */
    while (BSP_PB_GetState(BUTTON_USER) != 0)
    {
    }

    /* Turn off LED */
    BSP_LED_Off(LED_GREEN);

    /* Calculate the value required for a delay of macro-defined(Delay) */
    RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;

```

```

while (1)
{
    /* LPTIM must be disabled to restore internal state before next time enter stop mode */
    __HAL_LPTIM_DISABLE(&LPTIMConf);

    /* Wait at least three LSI times for the completion of the disable operation */
    APP_DelayNops(RatioNops);

    /* Configure LPTIM for once mode and enable interrupt */
    HAL_LPTIM_SetOnce_Start_IT(&LPTIMConf, 51);

    /* Suspend SysTick interrupt */
    HAL_SuspendTick();

    /* VCORE = 0.8V when enter stop mode */
    PwrStopModeConf.LPVoltSelection = PWR_STOPMOD_LPR_VOLT_0P8V;
    PwrStopModeConf.FlashDelay = PWR_WAKEUP_HSIEN_AFTER_MR;
    PwrStopModeConf.WakeUpHsiEnableTime = PWR_WAKEUP_FLASH_DELAY_1US;
    HAL_PWR_ConfigStopMode(&PwrStopModeConf);

    /* Enter Stop Mode and Wakeup by WFI */
    HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,
        PWR_STOPEXIT_WFI);

    /* Resume SysTick */
    HAL_ResumeTick();
    if (HAL_IWDG_Refresh(&IwdgHandle) != HAL_OK)
    {
        APP_ErrorHandler();
    }
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_1);
}
}
static void APP_IwdgConfig()
{
    IwdgHandle.Instance = IWDG; /* Select IWDG */
    IwdgHandle.Init.Prescaler = IWDG_PRESCALER_32; /* Configure prescaler as 32 */
    IwdgHandle.Init.Reload = (1024); /* IWDG counter reload value is 1024, 1s */
    /* Initialize IWDG */
    if (HAL_IWDG_Init(&IwdgHandle) != HAL_OK)
    {
        APP_ErrorHandler();
    }
}
/**
 * @brief LPTIM AutoReloadMatchCallback
 * @param None
 * @retval None
 */
void HAL_LPTIM_AutoReloadMatchCallback(LPTIM_HandleTypeDef *LPTIMConf)
{
    BSP_LED_Toggle(LED_GREEN);
}
/**
 * @brief Clock configuration function

```

```

* @param None
* @retval None
*/
static void APP_RCCOscConfig(void)
{
    RCC_OscInitTypeDef OSCINIT = {0};
    RCC_PeriphCLKInitTypeDef LPTIM_RCC = {0};

    /* Oscillator configuration */
    OSCINIT.OscillatorType = RCC_OSCILLATORTYPE_LSI; /* Select oscillator LSI */
    OSCINIT.LSIState = RCC_LSI_ON; /* Enable LSI */
    OSCINIT.PLL.PLLState = RCC_PLL_NONE; /* PLL configuration
unchanged */
    /*OSCINIT.PLL.PLLSource = RCC_PLLSOURCE_HSI;*/
    /*OSCINIT.PLL.PLLMUL = RCC_PLL_MUL2;*/
    /* Configure oscillator */
    if (HAL_RCC_OscConfig(&OSCINIT) != HAL_OK)
    {
        APP_ErrorHandler();
    }

    /* LPTIM clock configuration */
    LPTIM_RCC.PeriphClockSelection = RCC_PERIPHCLK_LPTIM; /* Select peripheral clock:
LPTIM */
    LPTIM_RCC.LptimClockSelection = RCC_LPTIMCLKSOURCE_LSI; /* Select LPTIM clock
source: LSI */
    /* Peripheral clock initialization */
    if (HAL_RCCEx_PeriphCLKConfig(&LPTIM_RCC) != HAL_OK)
    {
        APP_ErrorHandler();
    }

    /* Enable LPTIM clock */
    __HAL_RCC_LPTIM_CLK_ENABLE();
}

/**
* @brief Configure GPIO
* @param None
* @retval None
*/
static void APP_GpioConfig(void)
{
    /* Configuration pins */
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    __HAL_RCC_GPIOB_CLK_ENABLE(); /* Enable the GPIO clock*/
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; /* GPIO mode is OutputPP */
    GPIO_InitStruct.Pull = GPIO_PULLUP; /* pull up */
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH; /* The speed is high */
    GPIO_InitStruct.Pin = GPIO_PIN_1;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

/**
* @brief Delayed by NOPS
* @param None
* @retval None
*/
static void APP_DelayNops(uint32_t Nops)

```

```
{
  for(uint32_t i=0; i<Nops;i++)
  {
    __NOP();
  }
}

/**
 * @brief This function is executed in case of error occurrence.
 * @param None
 * @retval None
 */
void APP_ErrorHandler(void)
{
  /* Infinite loop */
  while (1)
  {
  }
}
```

## 附录 2

### 2.PY32F071读取information区域中存放的Vreferint 1.2实测值(具体地址见3.3)

```
#define HAL_VREF_INT          (*(uint8_t*)(0x1fff3023))
#define HAL_VREF_DEC         (*(uint8_t*)(0x1fff3022))
#define vref_int             (*(uint8_t*)(HAL_VREF_INT))      //存放参考电压整数部分
#define vref_dec             (*(uint8_t*)(HAL_VREF_DEC))      //存放参考电压小数部分
float vref;              //参考电压值

static uint8_t Bcd2ToByte(uint8_t Value)
{
    uint32_t tmp = 0U;
    tmp = ((uint8_t)(Value & (uint8_t)0xF0) >> (uint8_t)0x4) * 10U;
    return (tmp + (Value & (uint8_t)0x0F));
}

float read_1_2V(void)
{
    uint8_t data_vref_int, data_vref_dec;
    data_vref_int = Bcd2ToByte(HAL_VREF_INT);
    data_vref_dec = Bcd2ToByte(HAL_VREF_DEC);

    //初始化所有外设, flash 接口, systick
    vref = data_vref_int/10;      //计算参考电压
    vref = vref + ((data_vref_int%10)*0.1 + data_vref_dec*0.001);
    return vref;
}
```