Technology Innovator

Puya

AN1065

应用笔记

PY32F030_PY32F003_PY32F002A 系列各模块的应用注意事项

前言

PY32F030_PY32F003_PY32F002A 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 64 Kbytes Flash 和 8 Kbytes SRAM 存储器，最高工作频率 48 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 PY32F030_PY32F003_PY32F002A 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	PY32F030、PY32F003、PY32F002A

目录

1 PWR 使用注意事项

- 为了提高系统稳定性必须使能看门狗功能；
- 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)
- MCU 进 Stop 之前需关闭 systick 中断(HAL_SuspendTick())；
- CPU 时钟分频，EXTI 模块的时钟和 CPU 时钟来自同一个时钟源但是为分频的条件下，在 Sleep 模式下使用事件唤醒 CPU，将会唤醒失败，需要使用中断唤醒。

2 GPIO 引脚使用注意事项

- 所有 GPIO 不能有超过-0.3 V 的负压，如对讲机市场，频道切换功能建议使用没有 AD 功能的 IO 口；
- 所有 GPIO 引脚的输入电压不得高于 V_{CC}+0.3 V；
- BOOT0 在任何复位产生的时候都不能为高电平(包括在被配置为普通 GPIO 状态后)，否则会进 BOOTLOADER；
- 初始化 GPIO 等其他的结构体都需要赋值为 0，避免初始值不固定。

3 ADC 使用注意事项

3.1 ADC 软件配置

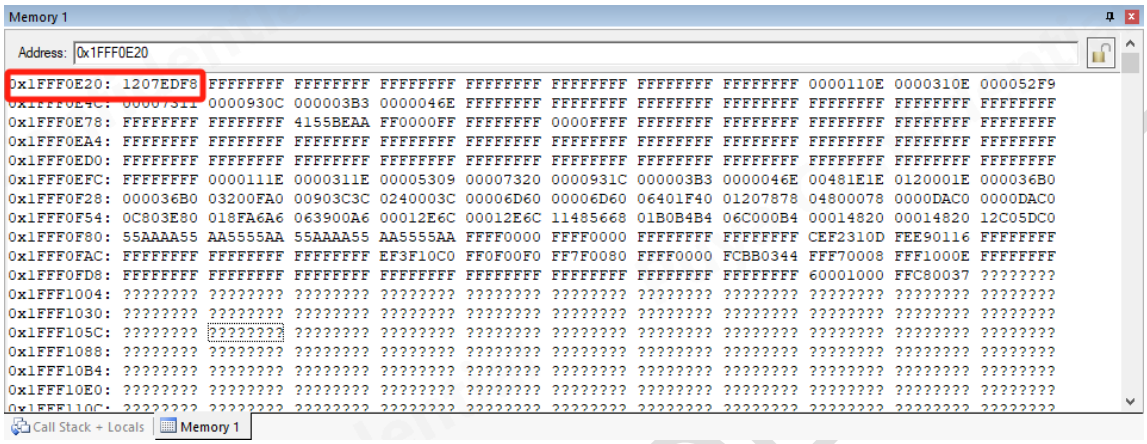
- ADC 初始化前添加 ADC_FORCE_RESET，确保初始化成功；
- ADC 需要在使能前配置通道，若在使能后配置则会失败；
- ADC 时钟需要配置到 16 MHz 以下，确保 ADC 采样精度；
- ADC 使能后需要增加 8 个 ADC 时钟的延时，才可以使能转换，否则会影响采样精度；
- GPIO 直接驱动大功耗器件会影响 ADC 采样结果(例如数码管显示，建议数码管显示的时候不采样 ADC，或者在数码管的各个数据线上面串入 10-100 Ω 电阻，可根据实际情况进行调整)；
- ADC 使能后软件不能禁能，需要复位 ADC 模块，然后重新初始化 ADC，最后启动 ADC；
- ADC 在连续模式或不连续模式下，仅使用通道 0 时，必须选择扫描序列向下；
- ADC 在单次模式下，转换结束后，需重新使能 ADC 模块（ADC_EN = 1），才能开始下一次转换。

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 采样异常。

3.3 Vreferint 1.2 V

- 芯片 Vreferint 1.2 V 实测值放置在 FLASH 中的 information 区域(0x1FFF0E20)。(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 Vreferint 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:

a) 降低分辨率；

b) 降低ADC的时钟频率；

c) 提高ADC采样周期。

总转换时间计算如下：

tCONV = 采样时间 + (转换分辨率+0.5) x 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：

tCONV = (239.5 + 12.5) x ADC 时钟周期= 252 x ADC 时钟周期 = 21 us

4 I2C 使用注意事项

- I2C 在初始化引脚 PF0、PF1 做 SCL、SDA 后，BUSY 位状态位受 IO 口影响置 1，影响 I2C 使用。软件必须在 IO 口初始化后复位一次 I2C 模块，使 BUSY 位清零；
- I2C 从机在发送一帧数据后，主机重新发地址后 buffer 指针会加 1，所以从机需在地址中断中重新初始化 buffer 指针；
- I2C 作从机，主机读取从机数据， 最后一个字节主机不回 ACK，从机进不了 Stop 中断，可使用 NACK 中断作 Stop 中断结束传输。(代码参考如下)

```
void I2C1_IRQHandler(void)
{
  HAL_I2C_EV_IRQHandler(&I2cHandle);
  HAL_I2C_ER_IRQHandler(&I2cHandle);
}
```

5 COMP 使用注意事项

- 使用比较器 2 需要同时使能比较器 1 的 SCALER_EN。(代码参考如下)

```
int main(void)
{
  /* Reset of all peripherals，  Initializes the Systick */
  HAL_Init();

  __HAL_RCC_COMP1_CLK_ENABLE();                    /* Enable COMP1 clock */
  __HAL_RCC_COMP2_CLK_ENABLE();                    /* Enable COMP2 clock */

  SET_BIT(COMP1->CSR，COMP_CSR_SCALER_EN);        /* Enable COMP1 SCALER_EN
*/
  while (1)
  {
  }

}
```

6 RCC 使用注意事项

- PLL 只能从 24 MHz 倍频到 48 MHz;
- 开启了 PLL，FLASH_LATENCY 需要设置为 1;
- PLL 在休眠前需要关闭，并且把时钟切换到 HSI;
- 48 MHz，IAP 跳转的时候关闭 PLL(例程参考附录 3);
- E 版本芯片 HSI 不支持分频;
- 在强电磁干扰环境下，不建议使用 HSE 和 LSE，如对讲机应用，建议在测频模式下使用 HSE，非测频模式下使用 HIS;(E 版本已修复)
- 不支持通过 RCC_CR->HSIDIV 进行分频。(仅有 E 版本存在此问题)

7 SPI 发送和接收

- SPI 作为主机接收一串数据会多一个字节，软件需要丢弃第一个字节；
- 使用 SPI 主机发送时不推荐使用硬件片选，推荐使用软件片选，释放硬件片选需要 disable SPI；
- SPI 做主机发送时每个字节前会多发一个 0x00，需要对 DR 寄存器做一下强制转换*((__IO uint8_t *) &SPI1->DR)) = byte，可避免这个问题；
- SPI 作为主机直接写 DR 寄存器发送数据的时候，需要在写 DR 后面添加四个 NOP()；确保发送成功；
- 建议客户直接使用库进行 SPI 操作；
- 当 SPI 使用 DMA 模式时，需先启动 SPI，然后开启 DMA。

8 SPI TFT 屏应用

- 建议 SPI 使用单工模式，TX 使用 polling 模式，RX 使用 DMA 模式。

9 IAP 升级

- PY32F030/PY32F003/PY32F002A 中，IAP 跳转到 APP 中需要中断重映射，APP 代码与中断矢量存于 (0x80000000+VECT_TAB_PFFSET) 的地址中，该地址为 0x80001000，故 VECT_TAB_OFFSET 为 0x1000，所以需定义 VECT_TAB_OFFSET 为 0x1000（#define VECT_TAB_OFFSET 0x1000）；
- BOOT 程序区域需要加写保护，避免 BOOT 被擦除；（写保护需要在烧写器上配置）
- 程序中有写 FLASH 操作的需在程序区域加写保护，避免误操作程序区。

10 LED 使用注意事项

- 在单独使用大电流脚时，需要打开 LED 模块时钟，置位 LED_CR_EHS = 1，并且配置 GPIO 速度为 VERY_HIGH。（参考代码如下）

```
int main(void)
{
  /* Reset of all peripherals，  Initializes the Systick */
  HAL_Init();
  SET_BIT(RCC->APBENR2，RCC_APBENR2_LEDEN);
  SET_BIT(LED->CR，LED_CR_EHS);
  APP_GpioConfig();
}
```

```
  /**
    * @brief  GPIO configuration
    * @param  None
    * @retval None
    */
  static void APP_GpioConfig(void)
  {
    GPIO_InitTypeDef  GPIO_InitStruct = {0};
    __HAL_RCC_GPIOB_CLK_ENABLE();                          /* Enable GPIOB clock
*/
    GPIO_InitStruct.Pin = GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;            /* Push-pull output */
    GPIO_InitStruct.Pull = GPIO_NOPULL;                    /* Enable pull-up */
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;     /* GPIO speed */
    /* GPIO Initialization */
    HAL_GPIO_Init(GPIOB,  &GPIO_InitStruct);
  }
```

11 DMA 使用注意事项

- DMA 必须确保在传输完成后才能关闭 DMA 使能，不能在传输过程中关闭，否则会导致 DMA 传输失败。

12 TIM1 使用注意事项

- TIM1_CH1、TIM1_CH2、TIM1_CH3 在禁能后重新使能，OCREF 不会清零，继续计数；TIM1_CH4 在禁能后重新使能，OCREF 会清零，重新计数；如果需要 TIM1_CH1/CH2/CH3 和 TIM1_CH4 信号同步，每次禁能后重新使能，需要禁能后手动清除所有通道 OCREF。

13 FLASH 使用注意事项

- FLASH 只支持 Page 擦和 Page 写，一个 Page 是 128 字节，起始地址只能 Page 对齐(如起始地址 0x08005000，0x08005080 等)；
- 每次 Page 写之前必须先 Page 擦。

14 Option 操作

- 量产时，Option 操作必须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 14-1、图 14-2 所示；

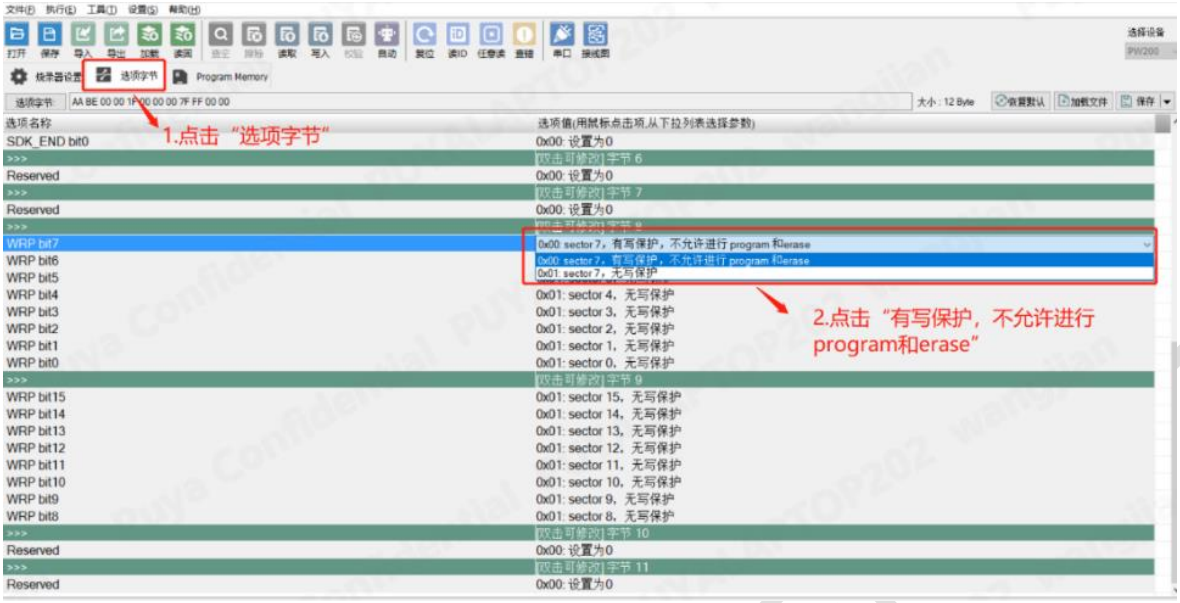


图 14-1 创芯工坊操作 Option 写保护

图 14-2 轩微操作 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)，具体步骤如图 14-3、图 14-4 所示。

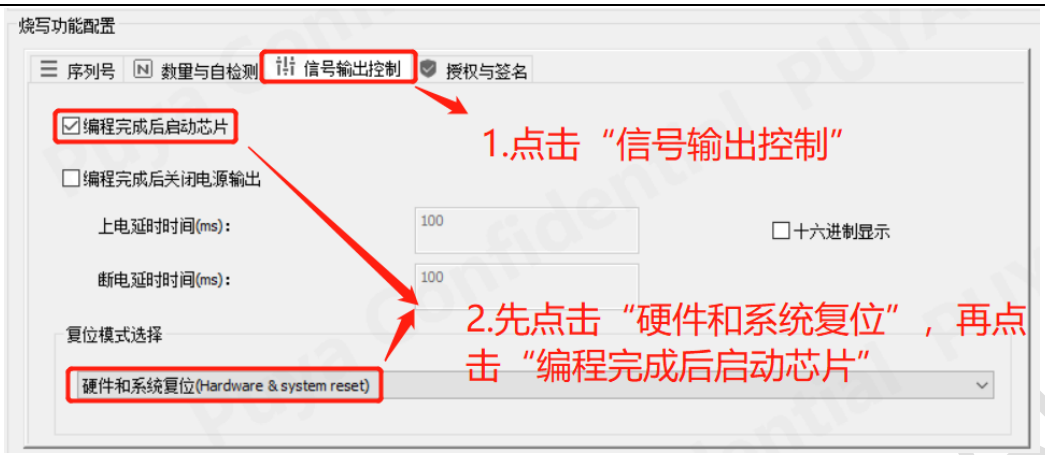


图 14-3 创芯工坊操作勾选“编程后重启芯片”

图 14-4 轩微操作“智能复位”

15 LPTIM 使用注意事项

- 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；
- LPTIM 时钟源为 LSE 时，使能 LSE 之前需要先使能 HSE；(例程参考附录 4，PY32F030-E/PY32F003-E/PY32F002A-E 版本不需要此操作)
- 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 单次模式每次进入 Stop 前必须清 ARRMCF 并需等待 4 个 LSI 时钟周期。(约需 160 us，包含程序执行时间)
- 改 LPTIM 的重载值，需等待 4 个 LSI 时钟周期。(约需 160 us，包含程序执行时间)

16 版本历史

版本	日期	更新记录
V1.0	2024.03.28	初版
V1.1	2024.06.06	修改 1、4、9、11 章节内容
V1.2	2024.10.15	新增 FLASH、LPTIM、TIM1 章节 合并第 3、4、5 章节，合并第 6、7 章节 修改第 1、2、3、4、5、6、7、9、10、12 章节 新增第 15 章节
V1.3	2025.06.16	修改 PWR、LPTIM、RCC 章节内容
V1.4	2025.07.22	修改 LPTIM、RCC 章节内容

Puya Semiconductor Co.， Ltd.

附录1

1.1 PY32F030/PY32F003/PY3F002A低功耗模式下，定时唤醒喂狗例程(LL库)

```
int main(void)
{
  /* Configure system clock */
  APP_SystemClockConfig();

  /* Enable LPTIM and PWR clock */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_LPTIM1);
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);

  /* Initialize LED and button */
  BSP_LED_Init(LED3);
  BSP_PB_Init(BUTTON_USER,BUTTON_MODE_GPIO);

  /* Configure LPTIM clock source as LSI */
  APP_LPTIMClockconf();
  /* Configure and enable LPTIM */
  APP_ConfigLPTIMOneShot();

  /* Turn on LED */
  BSP_LED_On(LED3);

  /* Wait for button press */
  while(BSP_PB_GetState(BUTTON_USER) != 0)
  {}
  APP_IwdgConfig();

  /* Turn off LED */
  BSP_LED_Off(LED3);

  while (1)
  {
    /* Enable low power run mode */
    LL_PWR_EnableLowPowerRunMode();
    /* Disable LPTIM */
    LL_LPTIM_Disable(LPTIM1);
    APP_uDelay(160);                    //必须在此处增加 160us 以上延迟
    /* Enable LPTIM */
    LL_LPTIM_Enable(LPTIM1);
```

```
        /* Set autoreload value */
        LL_LPTIM_SetAutoReload(LPTIM, 51);
        /* Start LPTIM in one-shot mode */
        LL_LPTIM_StartCounter(LPTIM1,LL_LPTIM_OPERATING_MODE_ONESHOT);

        /* Set SLEEPDEEP bit of Cortex System Control Register */
        LL_LPM_EnableDeepSLEEP();

        /* Request Wait For Interrupt */
        __WFI();
        LL_IWDG_ReloadCounter(IWDG);
        LL_GPIO_TogglePin(GPIOA,LL_GPIO_PIN_3);
    }
}
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);    /* T=1MS */

    /* Set watchdog reload counter */
    LL_IWDG_SetReloadCounter(IWDG, 1000);    /* 1ms*1000=1s */

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

    /* Feed watchdog */
    LL_IWDG_ReloadCounter(IWDG);
}
/**
    * @brief    LPTIM clock configuration
    * @param    None
    * @retval None
```

```
  */
static void APP_LPTIMClockconf(void)
{
  /* Enable LSI */
  LL_RCC_LSI_Enable();

  /* Wait for LSI to be ready */
  while(LL_RCC_LSI_IsReady() == 0)
  {}

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

/**
  * @brief  Configure LPTIM in one-shot mode
  * @param  None
  * @retval None
  */
static void APP_ConfigLPTIMOneShot(void)
{
  /* Configure LPTIM */
  /* LPTIM prescaler: divide by 128 */
  LL_LPTIM_SetPrescaler(LPTIM1,LL_LPTIM_PRESCALER_DIV128);

  /* Update ARR at the end of LPTIM counting period */
  LL_LPTIM_SetUpdateMode(LPTIM1,LL_LPTIM_UPDATE_MODE_ENDOFPERIOD);
  /* Enable ARR interrupt */
  LL_LPTIM_EnableIT_ARRM(LPTIM1);
  /* Enable NVIC interrupt request */
  NVIC_EnableIRQ(LPTIM1_IRQn);
  NVIC_SetPriority(LPTIM1_IRQn,0);
  /* Enable LPTIM */
  LL_LPTIM_Enable(LPTIM1);
  /* Configure auto-reload value: 51 */
  LL_LPTIM_SetAutoReload(LPTIM1,51);
}
void APP_LPTIMCallback(void)
{
  /*Toggle LED*/
  BSP_LED_Toggle(LED3);
}
static void APP_uDelay(uint32_t Delay)
{
```

```
    uint32_t temp;
    SysTick->LOAD=Delay*(SystemCoreClock/1000000);
    SysTick->VAL=0x00;
    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk;
    do
    {
      temp=SysTick->CTRL;
    }
    while((temp&0x01)&&!(temp&(1<<16)));
    SysTick->CTRL=0x00;
    SysTick->VAL =0x00;
}
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);
}
void APP_ErrorHandler(void)
{
    /* Infinite loop */
    while(1)
    {
    }
}
```

1.2 PY32F030/PY32F003/PY3F002A低功耗模式下，定时唤醒喂狗例程(HAL库)

```
int main(void)
{
  /* Reset of all peripherals, Initializes the Systick */
  HAL_Init();

  /* Clock configuration */
  APP_RCCOscConfig();

  /* Initialize LED */
  BSP_LED_Init(LED3);

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

  /* LPTIM initialization */
  APP_LPTIMInit();

  /* Enable PWR */
  __HAL_RCC_PWR_CLK_ENABLE();

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

  /* Wait for button press */
  while (BSP_PB_GetState(BUTTON_USER) != 0)
  {
  }

  /* IWDG initialization */
  APP_IWDGInit();

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

  while (1)
  {
    /* Disable LPTIM */
    __HAL_LPTIM_DISABLE(&LPTIMCONF);

    /* Enable LPTIM and interrupt, and start in single count mode */
    APP_LPTIMStart();

    /* Suspend Systick interrupt */
```

```
      HAL_SuspendTick();
      /* Enter STOP mode with interrupt wakeup */
      HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,PWR_STOPENTRY_WFI);
      /* Resume Systick interrupt */
          HAL_ResumeTick();
      if (HAL_IWDG_Refresh(&IwdgHandle) != HAL_OK)
      {
        Error_Handler();
      }
           /* LED Toggle */
      BSP_LED_Toggle(LED_GREEN);
   }
}

static void APP_RCCOscConfig(void)
{
   RCC_OscInitTypeDef OSCINIT;
   RCC_PeriphCLKInitTypeDef LPTIM_RCC;

   /* LSI clock configuration */
   OSCINIT.OscillatorType = RCC_OSCILLATORTYPE_LSI;      /* Set the oscillator type to LSI */
   OSCINIT.LSIState = RCC_LSI_ON;                        /* Enable LSI */
   /* Clock initialization */
   if (HAL_RCC_OscConfig(&OSCINIT) != HAL_OK)
   {
      Error_Handler();
   }

   /* 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)
   {
      Error_Handler();
   }

   /* Enable LPTIM clock */
   __HAL_RCC_LPTIM_CLK_ENABLE();
}
static void APP_LPTIMInit(void)
{
   /* LPTIM configuration */
```

```
  LPTIMCONF.Instance = LPTIM;                                    /* LPTIM */
  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)
  {
    Error_Handler();
  }
}

static void APP_LPTIMStart(void)
{
  /* Enable autoreload interrupt */
  __HAL_LPTIM_ENABLE_IT(&LPTIMCONF, LPTIM_IT_ARRM);

  __HAL_LPTIM_DISABLE(&LPTIMCONF);
  /* Delay 160us */
  APP_delay_us(160);                                    //必须在此处增加 160us 以上延迟

  /* Enable LPTIM */
  __HAL_LPTIM_ENABLE(&LPTIMCONF);

  /* Load autoreload value */
  __HAL_LPTIM_AUTORELOAD_SET(&LPTIMCONF, 51);

  /* Start single count mode */
  __HAL_LPTIM_START_SINGLE(&LPTIMCONF);
}
  static void APP_IWDGInit(void)
  {
    IwdgHandle.Instance = IWDG;                              /* Select IWDG */
    IwdgHandle.Init.Prescaler = IWDG_PRESCALER_32;     /* Configure prescaler to 32 */
    IwdgHandle.Init.Reload = (1000);                       /* Set IWDG counter reload value to 1000, 1s
*/
    /* Initialize IWDG */
    if (HAL_IWDG_Init(&IwdgHandle) != HAL_OK)
    {
      APP_ErrorHandler();
    }
  }

  static void APP_delay_us(int us)
 {
      unsigned t1, t2, count, delta, sysclk;    sysclk = 24  ;    //Modify this according to the system clock
```

```
        t1 = SysTick->VAL;
        while(1)
        {
            t2 = SysTick->VAL;
            delta = t2<t1?(t1-t2):(SysTick->LOAD - t2 + t1);
            if(delta >= us * sysclk)
            break;
        }
 }

 void Error_Handler(void)
{
   /* 无限循环 */
   while (1)
   {
   }
}
```

附录2

2. PY32F030/PY32F003/PY32F002A读取information区域中存放的Vreferint 1.2V实测值(具体地址见3.3)

```
#define HAL_VREF_INT                    (*(uint8_t *)(0x1fff0E23))
#define HAL_VREF_DEC                    (*(uint8_t *)(0x1fff0E22))
#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;
}
```

附录3

3. PLL48M做系统时钟时，IAP跳转关闭PLL

```
LL_UTILS_ClkInitTypeDef UTILS_ClkInitStruct;
static void SYSCLK(void);
/**
  * @brief  The application entry point.
  * @param  None
  * @retval None
  */
int main(void)
{
    SYSCLK();
#ifdef JUMP_TO_APP_BY_USER_BUTTON
  /* Configure user Button */
  BSP_PB_Init(BUTTON_USER, BUTTON_MODE_GPIO);

  /* Check if the USER Button is pressed */
  if (BSP_PB_GetState(BUTTON_USER) == 0x00)
  {
    JumpToAddress(APP_ADDR);
  }
#endif

  APP_SystemClockConfig(LL_RCC_HSICALIBRATION_24MHz, 24000000);

  Bootloader_Init();

  /* Infinite loop */
  while (1)
  {
    Bootloader_ProtocolDetection();
  }
}
static void SYSCLK(void)
{
  /* Enable and initialize HSI */
  LL_RCC_HSI_Enable();
  LL_RCC_HSI_SetCalibFreq(LL_RCC_HSICALIBRATION_24MHz);
  while(LL_RCC_HSI_IsReady() != 1)
  {
  }

  LL_PLL_ConfigSystemClock_HSI(&UTILS_ClkInitStruct);
```

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

  /* Configure HSISYS as system clock and initialize it */
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL)
  {
  }

  /* Set APB1 prescaler and initialize it */
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
  /* Update system clock global variable SystemCoreClock (can also be updated by calling
SystemCoreClockUpdate function) */
  LL_SetSystemCoreClock(48000000);
}
void APP_SystemClockConfig(uint32_t Value, uint32_t HCLKFrequency)
{
  /* HSI 使能及初始化 */
  LL_RCC_HSI_Enable();
  LL_RCC_HSI_SetCalibFreq(Value);
  while(LL_RCC_HSI_IsReady() != 1)
  {
  }

  /* 设置 AHB 分频 */
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
  /* 配置 HSISYS 为系统时钟及初始化 */
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSISYS);
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSISYS)
  {
  }
  LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
  /* 设置 APB1 分频及初始化 */
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
  /* 更新系统时钟全局变量 SystemCoreClock(也可以通过调用 SystemCoreClockUpdate 函数更新) */
  LL_SetSystemCoreClock(HCLKFrequency);
}
```

附录4

4.1 PY32F030/PY32F003/PY3F002A低功耗模式下，LSE做为LPTIM时钟源定时唤醒喂狗例程(LL库)

```
int main(void)
{
  /* Configure system clock */
    APP_SystemClockConfig();

    /* Enable LPTIM and PWR clock */
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_LPTIM1);
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);

    /* Initialize LED and button */
    BSP_LED_Init(LED3);
    BSP_PB_Init(BUTTON_USER,BUTTON_MODE_GPIO);
     /* Configure LPTIM clock source as LSI */
    APP_LPTIMClockconf();

    /* Configure and enable LPTIM */
    APP_ConfigLPTIMOneShot();

    /* Turn on LED */
    BSP_LED_On(LED3);

    /* Wait for button press */
    while(BSP_PB_GetState(BUTTON_USER) != 0)
    {}
    APP_IwdgConfig();
    /* Turn off LED */
    BSP_LED_Off(LED3);

    while (1)
    {
      /* Enable low power run mode */
      LL_PWR_EnableLowPowerRunMode();
      /* Disable LPTIM */
      LL_LPTIM_Disable(LPTIM1);
            APP_uDelay(160);       //必须在此处增加 160us 以上延迟
      /* Enable LPTIM */
      LL_LPTIM_Enable(LPTIM1);
        /* Set autoreload value */
```

```
    LL_LPTIM_SetAutoReload(LPTIM, 51);

    /* Start LPTIM in one-shot mode */
    LL_LPTIM_StartCounter(LPTIM1,LL_LPTIM_OPERATING_MODE_ONESHOT);

    /* Set SLEEPDEEP bit of Cortex System Control Register */
    LL_LPM_EnableDeepSLEEP();

    /* Request Wait For Interrupt */
    __WFI();
        LL_IWDG_ReloadCounter(IWDG);
        LL_GPIO_TogglePin(GPIOA,LL_GPIO_PIN_3);
  }
}
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);   /* T=1MS */

  /* Set watchdog reload counter */
  LL_IWDG_SetReloadCounter(IWDG, 1000);   /* 1ms*1000=1s */

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

  /* Feed watchdog */
  LL_IWDG_ReloadCounter(IWDG);
}
/**
  * @brief   LPTIM clock configuration
  * @param   None
  * @retval None
  */
static void APP_LPTIMClockconf(void)
```

```
{
  LL_PWR_EnableBkUpAccess();
  while(LL_PWR_IsEnabledBkUpAccess() == 0)
  {
  }
  LL_RCC_LSE_Enable();
  while (LL_RCC_LSE_IsReady() != 1)
  {
  }
  LL_PWR_DisableBkUpAccess();
  LL_APB1_GRP1_DisableClock(LL_APB1_GRP1_PERIPH_PWR);

  /* Configure LSI as LPTIM clock source */
  LL_RCC_SetLPTIMClockSource(LL_RCC_LPTIM1_CLKSOURCE_LSE);
}

/**
  * @brief  Configure LPTIM in one-shot mode
  * @param  None
  * @retval None
  */
static void APP_ConfigLPTIMOneShot(void)
{
  /* Configure LPTIM */
  /* LPTIM prescaler: divide by 128 */
  LL_LPTIM_SetPrescaler(LPTIM1,LL_LPTIM_PRESCALER_DIV128);

  /* Update ARR at the end of LPTIM counting period */
  LL_LPTIM_SetUpdateMode(LPTIM1,LL_LPTIM_UPDATE_MODE_ENDOFPERIOD);

  /* Enable ARR interrupt */
  LL_LPTIM_EnableIT_ARRM(LPTIM1);

  /* Enable NVIC interrupt request */
  NVIC_EnableIRQ(LPTIM1_IRQn);
  NVIC_SetPriority(LPTIM1_IRQn,0);

  /* Enable LPTIM */
  LL_LPTIM_Enable(LPTIM1);

  /* Configure auto-reload value: 51 */
  LL_LPTIM_SetAutoReload(LPTIM1,51);
}

/**
```

```
  * @brief  LPTIM ARR interrupt callback function
  * @param  None
  * @retval None
  */
void APP_LPTIMCallback(void)
{
  /*Toggle LED*/
  BSP_LED_Toggle(LED3);
}

/**
  * @brief  Microsecond delay function
  * @param  Delay: delay value
  * @retval None
  */
static void APP_uDelay(uint32_t Delay)
{
  uint32_t temp;
  SysTick->LOAD=Delay*(SystemCoreClock/1000000);
  SysTick->VAL=0x00;
  SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk;
  do
  {
    temp=SysTick->CTRL;
  }
  while((temp&0x01)&&!(temp&(1<<16)));
  SysTick->CTRL=0x00;
  SysTick->VAL =0x00;
}

/**
  * @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)
  {
  }
    LL_RCC_HSE_Enable();
    LL_RCC_HSE_SetFreqRegion(LL_RCC_HSE_16_32MHz);
```

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

4.2 PY32F030/PY32F003/PY3F002A低功耗模式下，LSE做为LPTIM时钟源定时唤醒喂狗例程(HAL库)

```
int main(void)
{
  /* Reset of all peripherals, Initializes the Systick */
  HAL_Init();

  APP_SystemClockConfig();

  /* Clock configuration */
  APP_RCCOscConfig();

  __HAL_RCC_LSI_ENABLE();

  /* Initialize LED */
  BSP_LED_Init(LED3);

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

  /* LPTIM initialization */
  APP_LPTIMInit();

  /* Enable PWR */

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

  /* Wait for button press */
  while (BSP_PB_GetState(BUTTON_USER) != 0)
  {
  }

  /* IWDG initialization */
  APP_IWDGInit();

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

  while (1)
  {
    /* Disable LPTIM */
    __HAL_LPTIM_DISABLE(&LPTIMCONF);
```

```
      /* Enable LPTIM and interrupt, and start in single count mode */
      APP_LPTIMStart();

      /* Suspend Systick interrupt */
      HAL_SuspendTick();
      /* Enter STOP mode with interrupt wakeup */
      HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,PWR_STOPENTRY_WFI);
      /* Resume Systick interrupt */
          HAL_ResumeTick();
      if (HAL_IWDG_Refresh(&IwdgHandle) != HAL_OK)
      {
        Error_Handler();
      }
           /* LED Toggle */
      BSP_LED_Toggle(LED_GREEN);
   }
}

static void APP_RCCOscConfig(void)
{
   RCC_OscInitTypeDef OSCINIT;
   RCC_PeriphCLKInitTypeDef LPTIM_RCC;

   /* LSI clock configuration */
   OSCINIT.OscillatorType = RCC_OSCILLATORTYPE_LSE;     /* Set the oscillator type to LSE*/
   OSCINIT.LSEState = RCC_LSE_ON;                       /* Enable LSE */
   /* Clock initialization */
   if (HAL_RCC_OscConfig(&OSCINIT) != HAL_OK)
   {
      Error_Handler();
   }

   /* LPTIM clock configuration */
   LPTIM_RCC.PeriphClockSelection = RCC_PERIPHCLK_LPTIM;          /* Select peripheral clock:
LPTIM */
   LPTIM_RCC.LptimClockSelection = RCC_LPTIMCLKSOURCE_LSE;      /* Select LPTIM clock
source: LSE */
   /* Peripheral clock initialization */
   if (HAL_RCCEx_PeriphCLKConfig(&LPTIM_RCC) != HAL_OK)
   {
      Error_Handler();
   }
static void APP_SystemClockConfig(void)
{
```

```
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /* Configure clock sources HSE/HSI/LSE/LSI */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE |
RCC_OSCILLATORTYPE_HSI | RCC_OSCILLATORTYPE_LSI | RCC_OSCILLATORTYPE_LSE;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
/* Enable HSI */
  RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV1;
/* HSI not divided */
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_24MHz;
/* Configure HSI output clock as 8MHz */
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
/* Enable HSE */
  RCC_OscInitStruct.HSEFreq = RCC_HSE_16_32MHz;
/* HSE frequency range 16MHz to 32MHz */
  RCC_OscInitStruct.LSIState = RCC_LSI_OFF;
/* Disable LSI */
  RCC_OscInitStruct.LSEState = RCC_LSE_OFF;
/* Enable LSE */
  RCC_OscInitStruct.LSEDriver = RCC_ECSCR_LSE_DRIVER_1;
/* LSE with default driving capability */
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_OFF;
/* Disable PLL */
  /* Initialize RCC oscillator */
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    APP_ErrorHandler();
  }

  /* Initialize CPU, AHB, and APB bus clocks */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK |
RCC_CLOCKTYPE_PCLK1;   /* RCC system clock types */
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
/*   SYSCLK source selection as LSE */
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
/* AHB clock not divided */
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
/* APB clock not divided */
  /* Initialize RCC system clock (FLASH_LATENCY_0=24M or below;   FLASH_LATENCY_1=48M) */
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    APP_ErrorHandler();
  }
}
```

```
  /* Enable LPTIM clock */
  __HAL_RCC_LPTIM_CLK_ENABLE();
}
static void APP_LPTIMInit(void)
{
  /* LPTIM configuration */
  LPTIMCONF.Instance = LPTIM;                                    /* LPTIM */
  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)
  {
    Error_Handler();
  }
}

static void APP_LPTIMStart(void)
{
  /* Enable autoreload interrupt */
  __HAL_LPTIM_ENABLE_IT(&LPTIMCONF, LPTIM_IT_ARRM);

  __HAL_LPTIM_DISABLE(&LPTIMCONF);
  /* Delay 160us */
  APP_delay_us(160);                                  //必须在此处增加 160us 以上延迟

  /* Enable LPTIM */
  __HAL_LPTIM_ENABLE(&LPTIMCONF);

  /* Load autoreload value */
  __HAL_LPTIM_AUTORELOAD_SET(&LPTIMCONF, 51);

  /* Start single count mode */
  __HAL_LPTIM_START_SINGLE(&LPTIMCONF);
}
  static void APP_IWDGInit(void)
  {
    IwdgHandle.Instance = IWDG;                                /* Select IWDG */
    IwdgHandle.Init.Prescaler = IWDG_PRESCALER_32;     /* Configure prescaler to 32 */
    IwdgHandle.Init.Reload = (1000);                        /* Set IWDG counter reload value to 1000, 1s
*/
    /* Initialize IWDG */
    if (HAL_IWDG_Init(&IwdgHandle) != HAL_OK)
    {
      APP_ErrorHandler();
```

```
    }
  }

  static void APP_delay_us(int us)
{
      unsigned t1, t2, count, delta, sysclk;   sysclk = 24 ;   //Modify this according to the system clock

      t1 = SysTick->VAL;
      while(1)
      {
          t2 = SysTick->VAL;
          delta = t2<t1?(t1-t2):(SysTick->LOAD - t2 + t1);
          if(delta >= us * sysclk)
          break;
      }
}

 void Error_Handler(void)
{
  /* 无限循环 */
  while (1)
  {
  }
}
```