AN1053

应用笔记

PY32Fxxxx 系列电子烟应用指南

前言

本应用笔记将帮助客户了解PY32Fxxxx系列芯片，并且快速进行电子烟开发。

表1. 适用产品

类型	产品系列
微型控制器系列	PY32F002A、PY32F002A –E、PY32F003、PY32F003-E、 PY32F030、PY32F030-E PY32F002B PY32F040、PY32F071 PY32F403 PY32F031

目录

目录

目录

1 PY32F030/PY32F003/PY32F002A应用注意事项

1.1 PWR

1.1.1 看门狗

- 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1.1、1.2)
- MCU进STOP之前需关闭systick中断(HAL_SuspendTick())

1.2 GPIO

1.2.1 引脚设计注意事项

- 所有GPIO不能有超过-0.3V的负压
- 所有GPIO引脚的输入电压不得高于VCC+0.3V
- 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。

1.3 ADC

1.3.1 ADC硬件设计注意事项

- ADC所有通道的电压不能高于VCC (即使ADC通道未配置为AD功能),否则ADC采样异常。

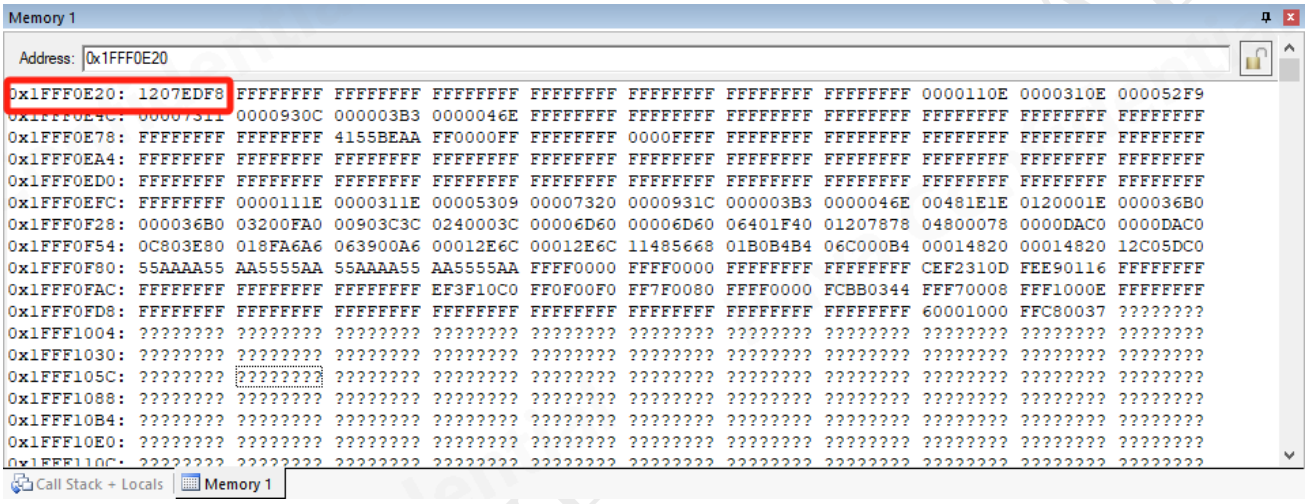
1.3.2 ADC配置

- 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（ADC_EN 置 1 到 ADSTART 置 1，时间间隔应大于 8 个 ADC 时钟）。
- 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16MHz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。
- 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欧姆电阻，可根据实际情况进行调整)。

1.3.3 内部参考电压1.2V

- 芯片的内部参考电压1.2V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) ，读取内部参考电压1.2V的程序见附录3:



- 在采样内部参考电压1.2V的时候，通过ADC采样时间转换公式算出来的结果要至少20us，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

1.4 I2C

1.4.1 PF0,PF1作为I2C引脚使用流程

- I2C 在初始化引脚 PF0、PF1 做 SCL、SDA 后，IO 口拉低状态下会使 BUSY 位置 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。

1.4.2 I2C从机通讯

- 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作从机接收每一个字节都需要时钟延长的情况下，地址匹配成功后，主机发送的前两个字节无法时钟延长。
- 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);
}
```

1.5 COMP

1.5.1 COMP2

- 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)
  {
  }
}
```

1.6 RCC

1.6.1 PLL

- PLL只能从24MHz倍频到48MHz。
- 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。
- 48MHz，IAP跳转的时候关闭PLL。(例程参考附录1.5)

1.6.2 HSI

- PY32F030-E、PY32F003-E、PY32F002A-E版本芯片的HSI不支持分频

1.7 SPI

1.7.1 SPI 使用DMA

- 先启动SPI，然后开启DMA。

1.7.2 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，或者使用全双工替代半双工。
- 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。

1.8 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。

1.9 LED

1.9.1 LED使用

- 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）

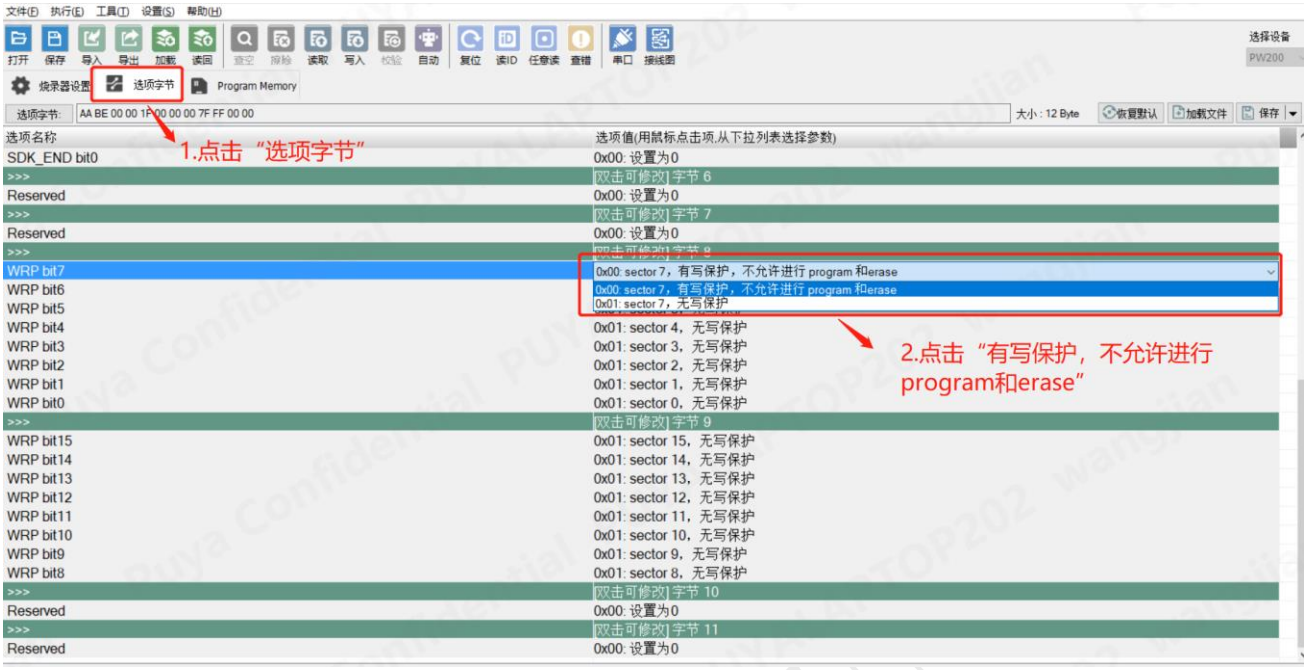
```
int main(void)
{
  /* Reset of all peripherals, Initializes the Systick */
  HAL_Init();
  SET_BIT(RCC->APBENR2,RCC_APBENR2_LEDEN);
  SET_BIT(LED->CR,LED_CR_EHS);
  APP_GpioConfig();
}

/**
  * @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);
}
```

1.10 OPTION操作

- 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。
- 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)。
- 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。

创芯工坊操作勾选“编程后重启芯片”



创芯工坊操作OPTION写保护

轩微操作“智能复位”

轩微操作OPTION写保护

1.11 LPTIM 使用注意事项

- LPTIM时钟源为LSE时，使能LSE之前需要先使能HSE。(例程参考附录1.3、1.4，PY32F030-E/PY32F003-E/PY32F002A-E版本不需要此操作)
- 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；
- 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。

1.11.1 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)

1.11.2 LPTIM单次模式

- 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)

2 PY32F002B应用注意事项

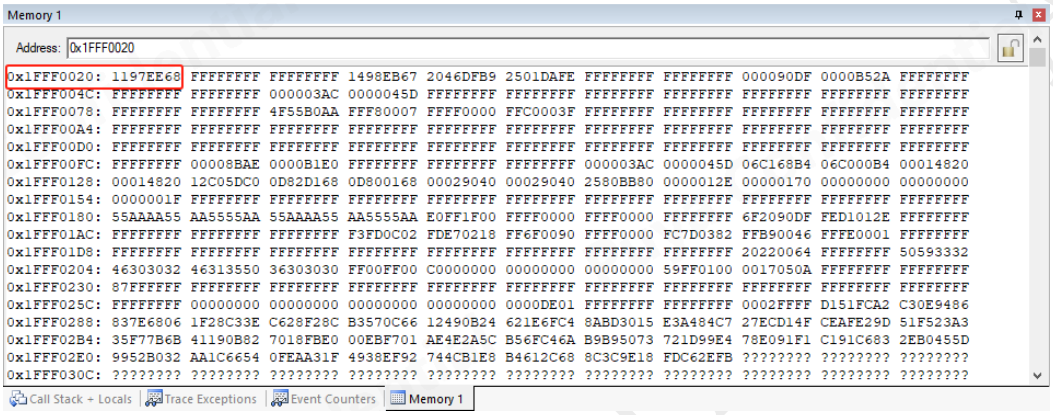
2.1 ADC

2.1.1 ADC硬件设计注意事项

- ADC所有通道的电压不能高于VCC (即使ADC通道未配置为AD功能),否则ADC采样异常。

2.1.2 ADC 内部1.2V

- 芯片的内部参考电压1.2V实测值放置在FLASH中的information区域(0x1FFF0020)。(高16位是实际值，低16位是反码)，读取内部参考电压1.2V的程序见附录3:



- 使用内部参考电压1.5V时，需要使能内部参考电压1.2V。
- 在采样内部参考电压1.2V的时候，通过ADC采样时间转换公式算出来的结果要至少20us。
  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 3.5个ADC 时钟周期:

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

2.1.3 ADC配置

- 切换ADC通道 ，需要关闭ADC使能。
- 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。
- 进休眠模式前，需要复位ADC模块。

2.2 PWR

2.2.1 看门狗

- 为了提高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2.1、2.2)
- MCU进STOP之前需关闭systick中断(HAL_SuspendTick())

2.3 COMP

2.3.1 COMP使用注意事项

- 比较器在低功耗模式下不支持唤醒。

2.4 GPIO

2.4.1 引脚设计注意事项

- 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。
- 所有GPIO不能有超过-0.3V的负压

2.5 I2C

2.5.1 I2C从机通讯

- 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。
- 在I2C从机接收到每一个字节都需要时钟延长时，I2C主机发地址到从机的前两个字节无法时钟延长。

2.6 OPTION操作

- 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。
- 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，具体操作参考1.10图片)。
- 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。

2.7 LPTIM

- 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；
- 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；

2.7.1 LPTIM 连续模式注意事项

- 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40us*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160us*PSC，包含程序执行时间）。

2.7.2 LPTIM 单次模式注意事项

- 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160us，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160us，包含程序执行时间）。

3 PY32F040/PY32F071应用注意事项

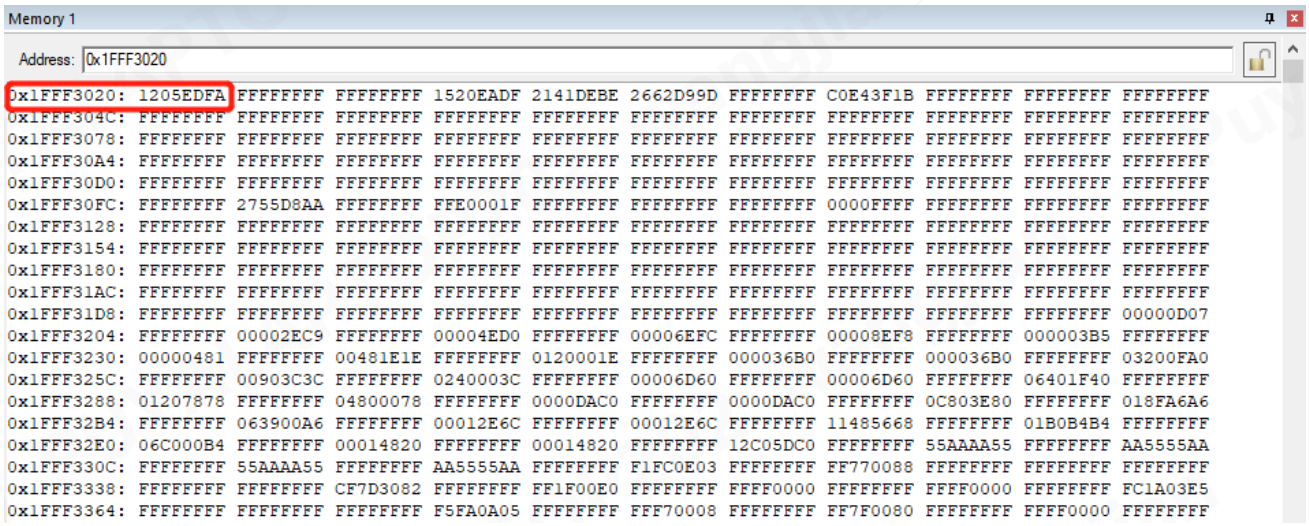
3.1 ADC

3.1.1 ADC配置

- 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)。
- 在使用ADC DMA连续采样内部通道(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。(使用通道16时需要配置通道0的采样周期，且两个通道的采样周期需一致)

3.1.2 内部参考电压1.2V

- 芯片的内部参考电压1.2V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) ，读取内部参考电压1.2V的程序见附录3:



- 在采样内部参考电压1.2V的时候，通过ADC采样时间转换公式算出来的结果要至少20us，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

3.2 LPTIM

- 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；

3.2.1 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)

3.2.2 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)

4 PY32F403应用注意事项

4.1 PWR

4.1.1 PLL

- PLL 倍频后不支持读取OTP FLASH，需要在系统时钟为HSI 8MHz时进行读取。

4.2 SPI

4.2.1 SPI配置

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

4.3 内部参考电压1.2V

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

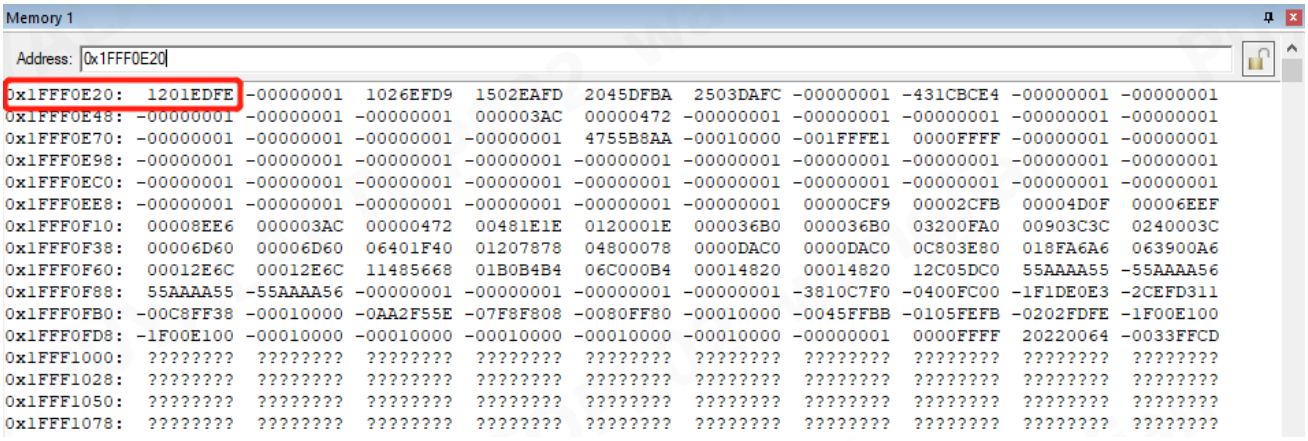
  当 ADC_CLK = 12MHz，分辨率为12位，且采样时间为 239.5个ADC 时钟周期:

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

5 PY32F031应用注意事项

5.1 内部参考电压1.2V

- 芯片的内部参考电压1.2V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) ，读取内部参考电压1.2V的程序见附录3:



- 在采样内部参考电压1.2V的时候，通过ADC采样时间转换公式算出来的结果要至少20us，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

5.2 LPTIM

- 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;
- 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;

5.2.1 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)

5.2.2 LPTIM单次模式

- 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)

6 版本历史

版本	日期	更新记录
V1.0	2024.01.31	初版
V1.1	2024.02.21	更新I2C,SPI
V1.2	2024.02.27	更新文件格式，1.1/1.2/1.3/1.4/1.7/2.1/2.2/2.4/2.5/6.3描述
V1.3	2024.3.22	更新2.2描述
V1.4	2024.03.25	更新1.1/1.2/1.3/1.10/2.1/2.2/2.4/2.6内容
V1.5	2024.05.14	更新1.1/1.3/1.7/2.1/2.2内容
V1.6	2024.07.04	修改1.1/1.2/1.3/1.4/1.5/1.6/1.7/1.8/1.9/1.10,新增1.11，修改2.1/2.2/2.6/2.7，修改附录例程
V1.7	2024.08.07	修改1.1.1/1.1.3/2.1.1/2.1.2/3.1.2,增加5.1/4.3 增加附录3内容
V1.8	2025.06.04	修改1.11，增加2.7/5.2内容
V1.9	2025.06.16	修改1.11，增加2.7/5.2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);
```

附录1

```
    /* 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);
  }
}
```

附录1

```
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);
}
```

附录1

```
  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)
  {
  }
}
```

1.3 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);
      /* 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 */
```

附录1

```
  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);
```

附录1

```
    /* 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)
    {
    }
```

附录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)
  {
  }
}
```

1.4 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)
   {
```

附录1

```
    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)
```

附录1

```
    {
      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)
  {
  }
}
```

1.5 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)
```

附录1

```
{
  /* 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2

2.1 PY32F002B 低功耗模式下，定时唤醒喂狗例程(LL库)

```
int main(void)
{
      APP_SystemClockConfig();
      BSP_LED_Init(LED3);
      BSP_PB_Init(BUTTON_USER, BUTTON_MODE_GPIO);

   BSP_LED_On(LED_GREEN);

   /* Wait the button be pressed */
   while (BSP_PB_GetState(BUTTON_USER) != 0)
   {
   }
      APP_IwdgConfig();
   /* Set wake-up mode of the LPTIM(EXTI Line29) to event request */
   LL_EXTI_DisableIT(LL_EXTI_LINE_29);     /* Disable interrupt request for EXTI Line29 */
   LL_EXTI_EnableEvent(LL_EXTI_LINE_29); /* Enable event request for EXTI Line29 */
   /* Set LSI as LPTIM clcok source */
   APP_ConfigLptimClock();

   /* Initialize LPTIM */
   LPTIM_InitStruct.Prescaler = LL_LPTIM_PRESCALER_DIV128;             /* prescaler: 128 */
   LPTIM_InitStruct.UpdateMode  =  LL_LPTIM_UPDATE_MODE_IMMEDIATE;    /*  registers  are
updated after each APB bus write access */
   if (LL_LPTIM_Init(LPTIM, &LPTIM_InitStruct) != SUCCESS)
   {
      APP_ErrorHandler();
   }
   /* LED off */
   BSP_LED_Off(LED_GREEN);

   /* Set LPTIM to continus mode Enable autoreload match interrupt */
//    APP_ConfigLptim();

   while (1)
   {
      APP_ConfigLptim();
      LL_LPTIM_ClearFLAG_ARRM(LPTIM1);

      /* Enable STOP mode */
      APP_EnterStop();
      LL_IWDG_ReloadCounter(IWDG);
      /* LED toggle */
      BSP_LED_Toggle(LED_GREEN);
   }
}
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 to IWDG_PR, IWDG_RLR and IWDG_WINR registers */
```

```
    LL_IWDG_EnableWriteAccess(IWDG);

    /* Set IWDG prescaler */
    LL_IWDG_SetPrescaler(IWDG, LL_IWDG_PRESCALER_32); /* T=1MS */

    /* Set IWDG reload value */
    LL_IWDG_SetReloadCounter(IWDG, 1000); /* 1ms*1000=3s */

    /* Check if all flags Prescaler, Reload & Window Value Update are reset or not */
    while (LL_IWDG_IsReady(IWDG) == 0U) {;}

    /* Reloads IWDG counter with value defined in the reload register */
    LL_IWDG_ReloadCounter(IWDG);
}
static void APP_SystemClockConfig(void)
{
    /* Enable HSI */
    LL_RCC_HSI_Enable();
    while(LL_RCC_HSI_IsReady() != 1)
    {
    }

    /* Set AHB divider: HCLK = SYSCLK */
    LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);

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

    /* Set APB1 divider */
    LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
    LL_Init1msTick(24000000);

    /* Update CMSIS variable (which can be updated also through SystemCoreClockUpdate function)
*/
    LL_SetSystemCoreClock(24000000);
}
static void APP_ConfigLptimClock(void)
{
    /* Enabel LSI */
    LL_RCC_LSI_Enable();
    while(LL_RCC_LSI_IsReady() != 1)
    {
    }

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

    /* Enable LPTIM clock */
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_LPTIM1);
}
static void APP_ConfigLptim(void)
{
    /* Enable LPTIM1 interrupt */
    NVIC_SetPriority(LPTIM1_IRQn, 0);
    NVIC_EnableIRQ(LPTIM1_IRQn);

    /* Enable LPTIM autoreload match interrupt    */
```

附录2

```
    LL_LPTIM_EnableIT_ARRM(LPTIM);

    LL_LPTIM_Disable(LPTIM);
    APP_delay_us(160);                                  //必须在此处增加160us以上延迟
    /* Enable LPTIM */
    LL_LPTIM_Enable(LPTIM);

    /* Set autoreload value */
    LL_LPTIM_SetAutoReload(LPTIM, 51);
    /* LPTIM starts in continuous mode */
    LL_LPTIM_StartCounter(LPTIM, LL_LPTIM_OPERATING_MODE_ONESHOT);

}
static void APP_delay_us(uint32_t nus)
 {
    uint32_t temp;
    SysTick->LOAD=nus*(SystemCoreClock/1000000);
    SysTick->VAL=0x00;
    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk;
    do
    {
       temp=SysTick->CTRL;
    }
    while((temp&0x01)&&!(temp&(1<<16)));
    SysTick->CTRL=SysTick_CTRL_ENABLE_Msk;
    SysTick->VAL =0x00;
 }
static void APP_EnterStop(void)
{
    /* Enable PWR clock */
    LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);
    /* STOP mode with low power regulator ON */
    LL_PWR_SetLprMode(LL_PWR_LPR_MODE_LPR);
    /* SRAM retention voltage aligned with digital LDO output */
    LL_PWR_SetStopModeSramVoltCtrl(LL_PWR_SRAM_RETENTION_VOLT_CTRL_LDO);
    /* Enter DeepSleep mode */
    LL_LPM_EnableDeepSleep();
    /* Request Wait For event */
     __SEV();
     __WFE();
     __WFE();
     LL_LPM_EnableSleep();
}
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);
    }
}
void APP_ErrorHandler(void)
{
    /* Infinite loop */
    while (1)
    {
    }
}
```

2.2 PY32F002B 低功耗模式下，定时唤醒喂狗例程(HAL库)

```
int main(void)
{
  EXTI_ConfigTypeDef        ExtiCfg;

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

    APP_IWDGConfig();

  /* Configure RCCOSC */
  APP_RCCOscConfig();

  /* Initialize LED */
  BSP_LED_Init(LED_GREEN);

  /* Initialize PA3 */
  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;
  HAL_EXTI_SetConfigLine(&ExtiHandle, &ExtiCfg);

  /* Enable LPTIM1 interrupt */
  HAL_NVIC_SetPriority(LPTIM1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(LPTIM1_IRQn);

  /* Suspend Systick */
  HAL_SuspendTick();

  /* LED ON*/
  BSP_LED_On(LED_GREEN);

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

  /* LED OFF */
  BSP_LED_Off(LED_GREEN);

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

  while (1)
```

附录2

```
    {
      /* 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_delay_us(160);                    //必须在此处增加160us以上延迟
      /* Configure LPTIM for once mode and enable interrupt */
      HAL_LPTIM_SetOnce_Start_IT(&LPTIMConf, 51);

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

            if (HAL_IWDG_Refresh(&IwdgHandle) != HAL_OK)
      {
        APP_ErrorHandler();
      }

      HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_3);
    }
}

void APP_IWDGConfig(void)
{
      IwdgHandle.Instance = IWDG;                                  /* IWDG */
    IwdgHandle.Init.Prescaler = IWDG_PRESCALER_32;    /* Prescaler DIV 32 */
    IwdgHandle.Init.Reload = (1000);                          /* IWDG Reload value 1000 */

    if (HAL_IWDG_Init(&IwdgHandle) != HAL_OK)          /* Initialize the IWDG */
    {
      APP_ErrorHandler();
    }
}

/**
    * @brief     LPTIM AutoReloadMatchCallback
    * @param     None
    * @retval    None
    */
void HAL_LPTIM_AutoReloadMatchCallback(LPTIM_HandleTypeDef *LPTIMConf)
{
    BSP_LED_Toggle(LED_GREEN);
}

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

    /* LSI Clock Configure */
    OSCINIT.OscillatorType = RCC_OSCILLATORTYPE_LSI;    /* LSI */
    OSCINIT.LSIState = RCC_LSI_ON;                                  /* LSI ON */
    OSCINIT.LSICalibrationValue = RCC_LSICALIBRATION_32768Hz;       /* LSI Set 32768Hz */
    /* RCC Configure */
    if (HAL_RCC_OscConfig(&OSCINIT) != HAL_OK)
```

```
  {
    APP_ErrorHandler();
  }

  LPTIM_RCC.PeriphClockSelection = RCC_PERIPHCLK_LPTIM;          /* Clock Configure
Selection：LPTIM */
  LPTIM_RCC.LptimClockSelection = RCC_LPTIMCLKSOURCE_LSI;    /* Select LPTIM Clock
Source：LSI */
  /* Peripherals Configure */
  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;
  __HAL_RCC_GPIOA_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_3;
  HAL_GPIO_Init(GPIOA, &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)
{
  while (1)
  {
  }
}
```

附录3

3.1 PY32F030/PY32F003/PY32F002A读取information区域中存放的内部参考电压1.2V实测值(具体地址见1.3.3)

```
#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;
}
```

3.2 PY32F002B读取information区域中存放的内部参考电压1.2V实测值(具体地址见2.1.1)

```
#define HAL_VREF_INT                    (*(uint8_t *)(0x1fff0023))
#define HAL_VREF_DEC                    (*(uint8_t *)(0x1fff0022))
#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;
}
```

3.3 PY32F040/PY32F071读取information区域中存放的内部参考电压1.2V实测值(具体地址见3.1.2)

```
#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;
}
```

3.3 PY32F031读取information区域中存放的内部参考电压1.2V实测值(具体地址见5.1)

```
#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;
```

附录3

```
        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4

4 PF0,PF1作为I2C引脚使用流程

```
void HAL_I2CInit(I2C_HandleTypeDef *hi2c)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_I2C_CLK_ENABLE();                                  /* Enable I2C clock */
  __HAL_RCC_GPIOF_CLK_ENABLE();                                /* Enable GPIOF clock */
  GPIO_InitStruct.Pin = GPIO_PIN_0 | GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;                      /* Open-drain mode */
  GPIO_InitStruct.Pull = GPIO_PULLUP;                          /* Pull-up */
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF12_I2C;                   /* Alternate as I2C */
  HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);                      /* Initialize GPIO */
  /* Reset I2C */
  __HAL_RCC_I2C_FORCE_RESET();
  __HAL_RCC_I2C_RELEASE_RESET();
}
```