Technology Innovator

Puya

AN1038

应用笔记

PY32F031 系列各模块的应用注意事项

前言

PY32F031 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 72 MHz。包含多种不同封装类型多款产品。芯片集成多路 I2C、SPI、USART 等通讯外设，1 路 12 位 ADC，4 个 16 位定时器，1 个 32 位定时器，以及 2 路比较器，2 路运算放大器，1 路 LCD 驱动器。

本应用笔记将帮助用户了解 PY32F031 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	PY32F031

目录

1 使用 SPI DMA 与 FLASH 进行通信

1.1 注意事项

- 在使用 PY32 设备的 SPI 模块的时钟 2 分频与 flash 进行通信时，为了保证数据传输的可靠性，需要考虑 SPI 时钟分频和 Datasize 之间的关系，由于 DMA 在 SPI 在时钟 2 分频的情况下，使用 8BIT Datasize 来不及搬运数据，建议使用 16BIT 的 Datasize,表 1-1 给出了 SPI 在不同模式下时钟与 Datasize 配置的建议。

表 1-1 SPI 不同模式下频率限制

模式	描述
Master	主模式下 SCK 频率最大为 PCLK/2
	Datasize 设置为 16BIT

1.2 操作流程

- 定义 SPI_HandleTypeDef 类型变量 SpiHandle，并初始化 SpiHandle 各个成员变量；
- 对于 SpiHandle 中的 BaudRatePrescaler 成员变量，设置为 SPI_BAUDRATEPRESCALER_2（2 分频）；
- 对于 SpiHandle 中的 DataSize 成员变量，设置为 SPI_DATASIZE_16BIT（16BIT）；
- 初始化 SPI 模块；
- 使用 SPI 进行通信。

1.3 代码示例

```
  SPI_HandleTypeDef SpiHandle;
  /* De-Initialize the SPI peripheral */
  Spi1Handle.Instance                = SPI1;                                  /* SPI1 */
  Spi1Handle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2; /* prescaler :2 */
  Spi1Handle.Init.Direction          = SPI_DIRECTION_2LINES;          /* full duplex */
  Spi1Handle.Init.CLKPolarity        = SPI_POLARITY_LOW;     /* SPI Clock Polarity: low */
  Spi1Handle.Init.CLKPhase           = SPI_PHASE_1EDGE ;      /* Data sampling starts at the
first clock edge */
  Spi1Handle.Init.DataSize           = SPI_DATASIZE_16BIT;   /* SPI Data Size is 16 bit */
  Spi1Handle.Init.FirstBit           = SPI_FIRSTBIT_MSB;      /* SPI MSB Transmission */
  Spi1Handle.Init.NSS                = SPI_NSS_SOFT;         /* NSS Software Mode    */
  Spi1Handle.Init.Mode = SPI_MODE_MASTER;                     /* Configure as host */
  Spi1Handle.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;      /* The CRC check is
disabled */

    /* Initialize SPI peripheral */
    if (HAL_SPI_Init(&Spi1Handle) != HAL_OK)
    {
      APP_ErrorHandler();
    }
```

2 PWR 使用注意事项

- 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；
- 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；
- 进入 stop 模式时，HSE 的时钟频率低于系统时钟，会导致 HSE 时钟不能正常关闭；
- VCC 下降速率须慢于 30 us/V，否则程序会出现异常。

3 I2C 使用注意事项

- 在 IIC 的 FM+模式下，如果要使通讯速率达到 1 MHz，则需要外接 1 KΩ的上拉电阻。

4 LCD 相关注意事项

- 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;
  __IO uint32_t      RatioNops = 0;

int main()
{
  RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;

  HAL_LCD_Write(&LcdHandle，  LCD_RAM_REGISTER0，  0x0f0f0f0f);

  APP_DelayNops(RatioNops);/*延迟 2 个 pclk+1 个 lcd clk 周期，约为 80us

  HAL_LCD_Write(&LcdHandle，  LCD_RAM_REGISTER0，  0xf0f0f0f0);
}

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;
  __IO uint32_t      RatioNops = 0;

int main()
{
  RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;

  HAL_LCD_Write(&LcdHandle，  LCD_RAM_REGISTER0，  0x0f0f0f0f);

  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();
  }
}
```

5 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；
- 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。

6 ADC 上电校准

6.1 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。

6.2 操作流程

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

6.3 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);
   }
}
```

7 ADC 使用注意事项

7.1 ADC 软件配置

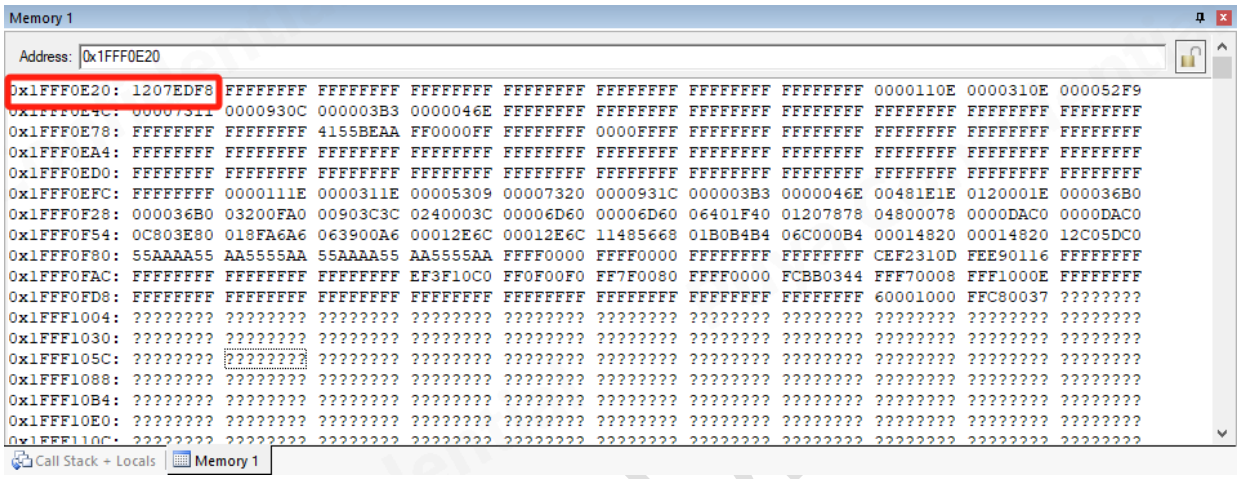
- ADC 初始化前添加 ADC_FORCE_RESET，确保初始化成功;
- ADC 需要在使能前配置通道，若在使能后配置则会失败;
- ADC 时钟需要配置到 16 MHz 以下，确保 ADC 采样精度;
- ADC 使能后需要增加 8 个 ADC 时钟的延时，才可以使能转换，否则会影响采样精度;
- 在切换 ADC 从连续模式到非连续模式之前，需要先关闭 ADC，然后再将其设置为非连续模式;
- GPIO 直接驱动大功耗器件会影响 ADC 采样结果(例如数码管显示，建议数码管显示的时候不采样 ADC，或者在数码管的各个数据线上面串入 10-100 Ω 电阻，可根据实际情况进行调整)。

7.2 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。

7.3 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 1：



- 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

8 RCC 使用注意事项

- 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；
- 以 PLL 作为系统时钟时，IAP 跳转的时候关闭 PLL(以 48MHz 为例，其例程参考附录 2)；
- HCLK 分频 HPRE 寄存器位不能从 8 切换到 1-7，否则会挂住时钟；

- APB 分频系数大时，执行（APB 总线上的）模块复位后，不能立马对模块寄存器进行读写操作，如需操作，则需要增加__NOP()空指令，空指令数量要大于 APB 分频数，例如 APB 8 分频，建议增加 10 个以上的__NOP 指令。

9 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 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；
- SPI 做从机发送模式，SPI->CR1.CPHA=0，发送数据时，从第二帧开始，会数据错误，表现为第一个发送的数据是前一帧最后一个数据，除非在每一帧发送之前对 SPI->CR1.SPE 先写 0 再写 1 操作；
- 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；
- 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。

10 IAP 升级

- PY32F031 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。

11 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 0x08000000，0x08000080 等)。
- 每次 Page 写之前必须先 Page 擦。

12 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 12-1、图 12-2 所示；

图 12-1 创芯工坊操作 Option 写保护

图 12-2 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 12-3、图 12-4 所示。

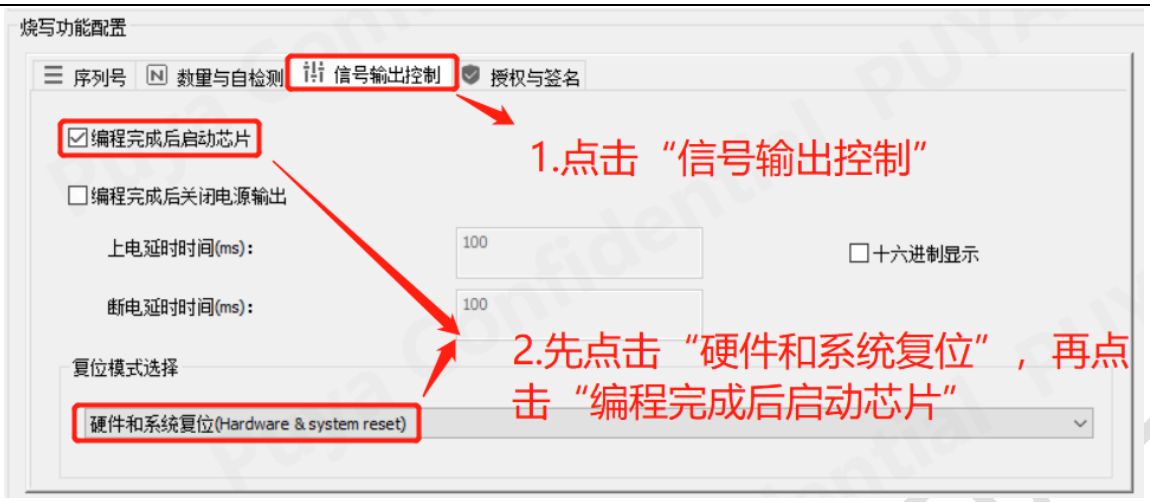


图 12-3 创芯工坊操作勾选“编程后重启芯片”

图 12-4 轩微操作“智能复位”

13 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；
- 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；

13.1 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。(约需 40 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*PSC 系数。(约需 160 us*PSC，包含程序执行时间)

13.2 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)

14 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。

15 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。

16 版本历史

版本	日期	更新记录
V1.0	2023.10.09	初版
V1.1	2025.01.19	增加 PWR、I2C、LCD、GPIO、ADC、RCC、 SPI、IAP、FLASH、Option、LPTIM、DMA 模块内容
V1.2/V1.3	2025.07.22	增加 LPTIM 模块内容
V1.4	2025.10.21	增加 ADC 上电校准，修改 ADC、SPI 模块内容
V1.5	2026.03.05	修改 LPTIM 模块内容

Puya Semiconductor Co.， Ltd.

附录1

PY32F031读取information区域中存放的Vreferint 1.2V实测值(具体地址见7.3)

```
#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2

48MHzPLL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);
}
```