

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、PY32F003、PY32F030 PY32F002B PY32F040、PY32F071 PY32F403

目录

1	PY32F030/PY32F003/PY32F002A应用注意事项	4
1.1	PWR	4
1.1.1	看门狗	4
1.2	GPIO	4
1.2.1	引脚设计注意事项	4
1.3	ADC	4
1.3.1	ADC配置	4
1.3.2	内部参考电压1.2V	4
1.4	I2C	5
1.4.1	PF0,PF1作为I2C引脚使用流程	5
1.4.2	I2C从机通讯	5
1.5	COMP	6
1.5.1	COMP2	6
1.6	RCC	6
1.6.1	PLL	6
1.7	SPI	6
1.7.1	SPI 使用DMA	6
1.7.2	SPI 发送和接收	6
1.7.3	TFT屏应用	6
1.8	IAP升级	6
1.9	LED	6
1.9.1	LED使用	6
2	PY32F002B应用注意事项	7
2.1	ADC	7
2.1.1	ADC 内部1.2V	7
2.1.2	ADC配置	7
2.2	PWR	8
2.2.1	看门狗	8
2.3	COMP	8
2.3.1	COMP使用注意事项	8
2.4	GPIO	8
2.4.1	引脚设计注意事项	8
2.5	I2C	8
2.5.1	I2C从机通讯	8
3	PY32F040/PY32F071应用注意事项	9

目录

3.1	ADC	9
3.1.1	ADC配置	9
4	PY32F403应用注意事项	10
4.1	PWR	10
4.1.1	PLL	10
4.2	SPI	10
4.2.1	SPI配置	10
5	版本历史	11
附录1		12
1.1	PY32F030/PY32F003/PY3F002A低功耗模式下, 定时唤醒喂狗例程(LL库)	12
1.2	PY32F030/PY32F003/PY3F002A低功耗模式下, 定时唤醒喂狗例程(HAL库)	14
附录2		17
2.1	PY32F002B 低功耗模式下, 定时唤醒喂狗例程(LL库)	17
2.2	PY32F002B 低功耗模式下, 定时唤醒喂狗例程(HAL库)	19
附录3		21
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1.3.2)	21
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)	21

1 PY32F030/PY32F003/PY32F002A应用注意事项

1.1 PWR

1.1.1 看门狗

- 推荐客户使用硬件看门狗替代软件看门狗，提高程序可靠性。
- 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)

1.2 GPIO

1.2.1 引脚设计注意事项

- PB7不能有小于-0.3V负压。
- 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。

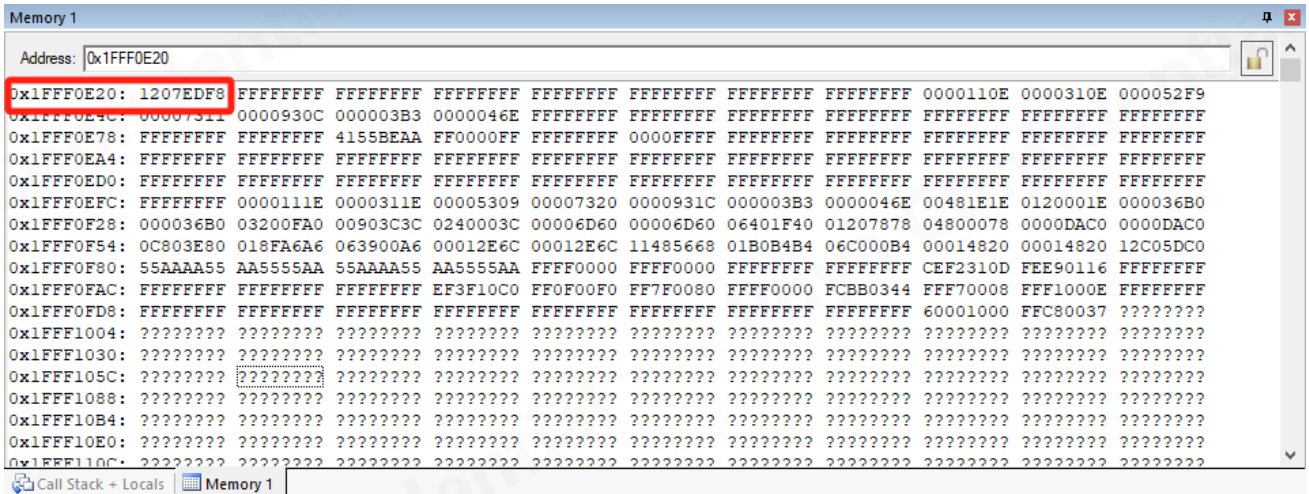
1.3 ADC

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16MHz以下，确保ADC采样精度。
- 有AD功能的通道的输入不能超过VCC电压，要不然AD采样值异常。
- 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。
- 数码管显示会影响ADC采样结果(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串入10-100欧姆电阻，可根据实际情况进行调整)。

1.3.2 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(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10us，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：

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

例如：

当 ADC_CLK = 16MHz，分辨率为12位，且采样时间为 3.5个ADC 时钟周期：

$$t_{CONV} = (3.5 + 12.5) \times \text{ADC 时钟周期} = 16 \times \text{ADC 时钟周期} = 1 \mu\text{s}$$

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，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。

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。
- 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。

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。

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。

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。
- 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。
- 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。

1.7.3 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。

1.8 IAP升级

- APP程序必须修改VECT_TAB_OFFSET, 例如`#define VECT_TAB_OFFSET 0x1000`。

1.9 LED

1.9.1 LED使用

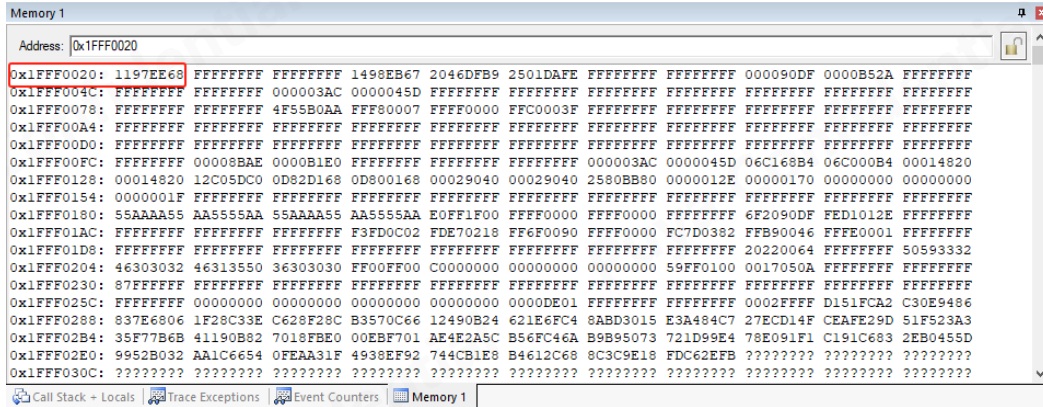
- 在单独使用大电流脚时, 需要打开LED模块时钟, 置位LED_CR_EHS = 1, 并且配置GPIO速度为VERY_HIGH。

2 PY32F002B应用注意事项

2.1 ADC

2.1.1 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10us。
 - 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：

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

例如：

当 $ADC_CLK = 16\text{MHz}$ ，分辨率为12位，且采样时间为 3.5个ADC 时钟周期：

$$t_{CONV} = (3.5 + 12.5) \times \text{ADC 时钟周期} = 16 \times \text{ADC 时钟周期} = 1 \mu\text{s}$$

2.1.2 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。
- 有AD功能的通道的输入不能超过VCC电压，要不然AD采样值异常。
- 进休眠模式前，需要复位ADC模块。

2.2 PWR

2.2.1 看门狗

- 推荐客户使用硬件看门狗替代软件看门狗，提高程序可靠性。
- 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)

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等其他结构体都需要赋值为0，避免初始值不固定。

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。

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)

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。

5 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



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

1.1 PY32F030/PY32F003/PY3F002A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);
    /* Configure Systemclock */

    APP_IwdgConfig();
    /* 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 ON */
    BSP_LED_On(LED_GREEN);

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

    /* LED off */
    BSP_LED_Off(LED_GREEN);

    while (1)
    {
        //Reconfigure LPTIM
        APP_ConfigLptim();
        //Clear ARRM
        LL_LPTIM_ClearFLAG_ARRM(LPTIM1);
        //wait for 3*Isi CLK
        APP_delay_us(120);
        //Close all non-wake-up interrupts, include systick.
        LL_SYSTICK_DisableIT();
        /* 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();
//Resume interrupts
LL_SYSTICK_EnableIT();
LL_IWDG_ReloadCounter(IWDG);
/* LED toggle */
BSP_LED_Toggle(LED_GREEN);
}
}

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 */
    LL_LPTIM_EnableIT_ARRM(LPTIM);

    /* 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(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;
    }
}
```

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)
    {
    }

    /* 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)
        {
            APP_ErrorHandler();
        }
        /* 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)
{
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();
}
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)
{
APP_ErrorHandler();
}
}

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

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

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

/* Delay 65us */
APP_delay_us(75);

/* Start single count mode */
__HAL_LPTIM_START_SINGLE(&LPTIMConf);
}
static void APP_delay_us(int us)
{
unsigned t1, t2, count, delta, sysclk; sysclk = 24 ; //Modify this according to the system clock

```

附录1

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

PUYA CONFIDENTIAL

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);
    /* Configure Systemclock */

    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 ON */
    BSP_LED_On(LED_GREEN);

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

    /* LED off */
    BSP_LED_Off(LED_GREEN);

    while (1)
    {
        //Reconfigure LPTIM
        APP_ConfigLptim();
        //Clear ARRM
        LL_LPTIM_ClearFLAG_ARRM(LPTIM1);
        //wait for 3*Isi CLK
        APP_delay_us(120);
        //Close all non-wake-up interrupts, include systick.
        LL_SYSTICK_DisableIT();
        /* 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();
//Resume interrupts
LL_SYSTICK_EnableIT();
LL_IWDG_ReloadCounter(IWDG);
/* LED toggle */
BSP_LED_Toggle(LED_GREEN);
}
}

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 */
LL_LPTIM_EnableIT_ARRM(LPTIM);

/* 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(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;
}
}

```

}

2.2 PY32F002B 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();
    APP_IwdgConfig();
    /* Configure RCCOSC */
    APP_RCCOscConfig();

    /* Initialize LED */
    BSP_LED_Init(LED_GREEN);

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

    /* Initialize LPTIM */
    LPTIMConf.Instance = LPTIM1; /* LPTIM1 */
    LPTIMConf.Init.Prescaler = LPTIM_PRESCALER_DIV128; /* DIV 128 */
    LPTIMConf.Init.UpdateMode = LPTIM_UPDATE_IMMEDIATE; /* UPDATE IMMEDIATE */
    /* Initialize LPTIM */
    if (HAL_LPTIM_Init(&LPTIMConf) != HAL_OK)
    {
        APP_ErrorHandler();
    }

    /* Configure Event Wakeup */
    ExtiCfg.Line = EXTI_LINE_29;
    ExtiCfg.Mode = EXTI_MODE_EVENT;
    HAL_EXTI_SetConfigLine(&ExtiHandle, &ExtiCfg);

    /* LED ON*/
    BSP_LED_On(LED_GREEN);

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

    /* LED OFF */
    BSP_LED_Off(LED_GREEN);

    while (1)
    {
        //LPTIM Once Start
        HAL_LPTIM_SetOnce_Start_IT(&LPTIMConf, 51);
        //Clear ARRM
        __HAL_LPTIM_CLEAR_FLAG(&LPTIMConf, LPTIM_FLAG_ARRM);
        //wait for 3*Isi CLK
        APP_delay_us(120);
        //Close all non-wake-up interrupts, include systick.
        HAL_SuspendTick();
        /* Enter Stop Mode and Wakeup by WFI */

        HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON,PWR_STOPENTRY_WFI
        );
        //Resume interrupts
    }
}
```

```
HAL_ResumeTick();

if (HAL_IWDG_Refresh(&IwdgHandle) != HAL_OK)
{
    APP_ErrorHandler();
}
/* LED Toggle */
BSP_LED_Toggle(LED_GREEN);
}
}

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;
    }
}

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

    /* LSI Clock Configure */
    OSCINIT.OscillatorType = RCC_OSCILLATORTYPE_LSI; /* LSI */
    OSCINIT.LSIState = RCC_LSI_ON; /* LSI ON */
    /* 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();
}
}
```

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1.3.2)

```

#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;

```

附录3

```
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;
}
```