AN1033
应用笔记

PY32F071_PY32F072 的应用注意事项

前言

PY32F071/PY32F072 系列微控制器采用高性能的 32 位 ARM® Cortex®-M0+ 内核，宽电压工作范围的 MCU。嵌入高 达 128 Kbytes flash 和 16 Kbytes SRAM 存储器，最高工作频率 72 MHz。包含多种不同封装类型多款产品。

本应用笔记将帮助用户了解 PY32F071/PY32072 各个模块应用的注意事项，并快速着手开发。

表 1. 适用产品

类型	产品系列
微型控制器系列	PY32F071、PY32F072

目录

1. PWR

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

2. ADC 硬件设计注意事项

- ADC通道电压不能高于VCC(即使ADC通道未配置为AD功能),否则ADC采样不准

3. ADC 使用 TIMER_CC/TRGO 触发注意事项

- 使用TIMER_CC/TIMER_TRGO触发ADC转换，ADC时钟不能8分频

4. ADC 外部中断(EXTI)触发注意事项

- 当时钟AHBCLK/APBCLK≥4时，不能设置EXTI_11触发ADC规则转换，不能使用EXTI_15触发ADC注入转换

5. ADC 配置

- 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) （C版本已修复）
- 因为ADC在全部通道转换完成后才会一个EOC标志，所以没办法使用非DMA方式的多通道采样(可设置非连续模式使能，一个通道转换即有一个EOC标志)
- ADC参考电压（VCC或VREFBUF）低于2V时无法校准（C版本已修复）
- TIM_CC触发注入通道单次采样异常，不建议使用（C版本已修复）
- 内部温度传感器通道（Tsensor）无法使用（C版本已修复）
- 软件判断EOC拉低后在开始Start,避免EOC跟Start信号同时产生
- TIMER触发ADC采样时，需保证ADC工作时钟大于等于TIMER时钟
- ADC注入模式无法使用（C版本已修复）

- 系统时钟8M时且ADC CLOCK两分频时，ADC无法校准（C版本已修复）
- 在配置通道16（OPA3_IN）的采样时间时，需配置通道0与通道16为相同的采样时间

6. TTCAN 系统时钟配置注意事项

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

7. COMP 硬件设计

- 当比较器的VINM输入信号为内部的模拟电压源时（例如VREFINT，TSVIN，VREF1P2），外部输入通道VINP需要加一个电容(1nF)到地

8. CRC 注意事项

- 禁止使用DMA模式
- 禁止连续对CRC_DR寄存器进行操作，建议使用库，库中有规避

9. DAC 注意事项

- 在三角波波形生成和噪声波形生成模式下，DHR会维持上一个周期的值

10. DMA 应用注意事项

- 外设使用DMA搬移数据时，需要等待DMA搬移完数据才能关闭外设的DMA使能

11. GPIO 注意事项

- 不得使用DMA GPIO模式（C版本已修复）

12. I2C 应用注意事项

- 使用DMA进行I2C的写数据搬运时，需在传输完地址之后再使能I2C的DMA_EN

13. LCD 应用注意事项

- 向同一个LCD_RAMx寄存器写数据时，需要在2个lcd clk周期内写完数据，之后等待2个pclk+1个lcd

clk周期后才能继续写数据(参考如下)

```
  #define   Delay   40*2

  LCD_HandleTypeDef LcdHandle;

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

  HAL_LCD_Write(&LcdHandle, LCD_RAM_REGISTER0, 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;

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();
  }
}
```

14. 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）

15. LPTIM 单次模式注意事项

- 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160us，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160us，包含程序执行时间）

16. RCC 注意事项

- APB分频系数大于1时，模块复位后需增加（n+2）个__nop()空指令才能对模块寄存器进行读写操作，n为APB分频系数
- stop唤醒会将HSIDIV恢复到默认值，若实际使用HSIDIV≠0唤醒后需重新配置

17. SPI 收发注意事项

- SPI做从机发送时，需再每一帧数据发送前先将SPI_EN清0再将SPI_EN置1

18. TIMER 使用注意事项

- 使能CC中断时，对应的分频系数PSC不得高于80
- TIMER预分频必须设置为1，否则会导致MCU反复进中断
- 禁止使用刹车功能（C版本已修复）
- TIMER3的ETR功能禁止使用（C版本已修复）

19. USB 使用注意事项

- 使能USB时钟门控前，需先开启USB 48MHz时钟
- USB在使能端点中断前需要清端点FIFO

20. OPTION & FLASH 操作注意事项

- 量产时，OPTION操作需要在烧写器选项字节中配置，并把程序中操作OPTION的函数屏蔽
- 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256字节，起始地址只能Page对齐(如起始地址0x08005000，0x08005100等)
- 每次Page写之前必须先Page擦

21. 版本历史

版本	日期	更新记录
V1.0	2023.6.15	初版
V1.1	2024.7.1	1.添加 PWR、修改 4、6、14、15 章节，新增 5、8、9、11、12、13、16、17、18、19、20、附录 1 章节
V1.2	2024.8.25	修改 5、11、16、18 章节

Puya Semiconductor Co., Ltd.

附录 1

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

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

  /* Initialize LED */
  BSP_LED_Init(LED_GREEN);

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

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

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

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

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

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

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

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

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

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

```
        /* Enable LPTIM */
        LL_LPTIM_Enable(LPTIM);

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

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

        /* Enable STOP mode */
        APP_EnterStop();

        /* PB1 toggle */
        LL_GPIO_TogglePin(GPIOB, LL_GPIO_PIN_1);
    }
}

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

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

  /* Enable LPTIM clock */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_LPTIM1);
}

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

  /* Enable IWDG */
  LL_IWDG_Enable(IWDG);

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

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

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

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

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

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

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

}

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

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

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

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

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

```
    /* Enter DeepSleep mode */
    LL_LPM_EnableDeepSleep();

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

    LL_LPM_EnableSleep();
}

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

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

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

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

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

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

        LL_IWDG_ReloadCounter(IWDG);

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

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

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

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

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

  /* Clock configuration */
  APP_RCCOscConfig();

  /* Initialize LED */
  BSP_LED_Init(LED_GREEN);

  APP_GpioConfig();

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

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

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

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

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

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

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

  while (1)
  {
```

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

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

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

      /* Suspend SysTick interrupt */
      HAL_SuspendTick();

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

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

      /* Resume Systick */
      HAL_ResumeTick();
      if (HAL_IWDG_Refresh(&IwdgHandle) != HAL_OK)
      {
        APP_ErrorHandler();
      }
      HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_1);
   }
}
static void APP_IwdgConfig()
{
   IwdgHandle.Instance = IWDG;                              /* Select IWDG */
   IwdgHandle.Init.Prescaler = IWDG_PRESCALER_32;   /* Configure prescaler as 32 */
   IwdgHandle.Init.Reload = (1024);                         /* IWDG counter reload value is 1024, 1s */
   /* Initialize IWDG */
   if (HAL_IWDG_Init(&IwdgHandle) != HAL_OK)
   {
      APP_ErrorHandler();
   }

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

/**
   * @brief     Clock configuration function
   * @param     None
   * @retval    None
   */
static void APP_RCCOscConfig(void)
{
```

```
    RCC_OscInitTypeDef OSCINIT = {0};
    RCC_PeriphCLKInitTypeDef LPTIM_RCC = {0};

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

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

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

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

/**
    * @brief    Delayed by NOPS
    * @param    None
    * @retval None
    */
static void APP_DelayNops(uint32_t Nops)
{
    for(uint32_t i=0; i<Nops;i++)
    {
        __NOP();
    }
}
```

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