AN1032

应用笔记

PY32L020 的应用注意事项

前言

PY32L020 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 24Kbytes Flash 和 3Kbytes SRAM 存储器，最高工作频率 48MHz。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 PY32L020 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	PY32L020

目录

1 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
- 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)

2 ADC 上电校准

2.1 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。

2.2 操作流程

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

2.3 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);
  }
}
```

3 ADC 硬件设计注意事项

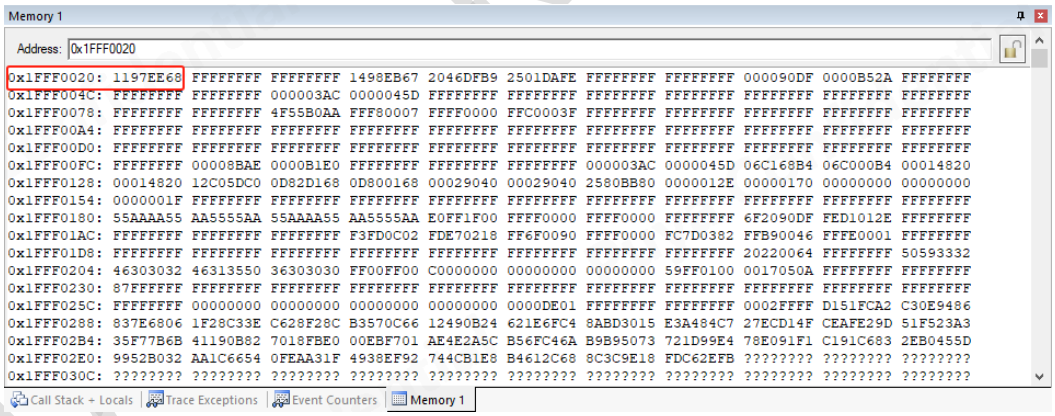
- ADC不支持采样VCC/3(内部通道10)
- ADC通道电压不能高于（VCC+0.3V）(即使ADC通道未配置为AD功能),否则ADC采样异常

4 ADC VerfBuffer 注意事项

- 若要使能内部1.5V VerfBuffer，软件上必须同时使能1.2V VREFEN(ADC->CCR. VREFEN=1)

5 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2：



- 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：

tCONV = 采样时间 + (转换分辨率+0.5) x 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：

tCONV = （3.5 + 12.5） x ADC 时钟周期= 16 x ADC 时钟周期 = 1 μs

6 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。

7 SPI 最快传输速度

SPI 模式	收/发模式	SPI 最快速度
从机全双工	收	PCKL/16
从机全双工	发	PCKL/16
主机全双工	收	PCKL/2
主机全双工	发	PCKL/2

8 TIMER 使用 CC 中断注意事项

- Timer中断函数中，清CC中断标志，必须等待TIM_PSC*PCLK，否则会导致清中断标志失败

9 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）

10 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3个LSI时钟周期（约需120us，包含程序执行时间）

11 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，外部输入通道VINP需要加一个电容(1nF)到地

12 COMP 使用注意事项

- 比较器在低功耗模式下不支持唤醒。

13 GPIO 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

14 I2C 从机通讯注意事项

- 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

15 IO 倒灌电流使 MCU 工作

15.1 注意事项

- VCC 未供电的情况下，IO 倒灌电流使 MCU 工作，可通过软件配置规避

15.2 操作流程

- 硬件：对应 IO 口需串 100Ω~1KΩ 电阻
- 上电初始化前需设置对应 IO 输出为开漏模式
- 延迟 5ms
- 程序正常初始化

15.3 代码示例

```
 int main(void)
{
    LL_IOP_GRP1_EnableClock(LL_IOP_GRP1_PERIPH_GPIOA);    /*使能 GPIOA 时钟*/
    LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_1, LL_GPIO_OUTPUT_OPENDRAIN);
                                                          /*将 PA1 引脚配置为开漏输出*/
    LL_mDelay(5);                                         /*延迟 5ms*/
}
```

16 IWDG 不支持冻结功能

16.1 注意事项

- IWDG 冻结功能无效，使能以后无法关闭。IWDG 和 STOP 模式同时使用时，需要使用 LPTIM 定时唤醒喂狗。下表是定时唤醒对功耗的影响：

注意：以下运行功耗是在系统时钟为 24M 情况下计算				
低功耗运行时间(ms)	stop 模式电流(uA)	唤醒运行时间(ms)	运行模式电流(uA)	平均功耗(uA)
500	1.7	1	1100	3.892215569
1000		1		2.797202797
2000		1		2.248875562
3000		1		2.065978007

16.2 操作流程

- 将所有的唤醒源配置为事件唤醒；
- 进入 STOP 前关闭所有中断并清除相应的标志；
- 开启 LPTIM，使能定时唤醒功能，并确保 LPTIM 定时时间小于 IWDG 溢出时间；(参考第 7、8 章进行配置)

17 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

18 版本历史

版本	日期	更新记录
V1.0	2023.6.15	初版
V1.1	2023.7.15	更新 CMP 硬件设计
V1.2	2023.11.10	第 11 章增加 LPTIM 唤醒后的平均功耗 第 7、8 章增加 LPTIM 使用事件唤醒不需要等待时间
V1.3	2023.11.17	修改第 11 章内容
V1.4	2023.11.22	修改第 7、8、11 章 LPTIM 相关内容
V1.5	2024.3.28	增加 PWR、ADC、COMP、GPIO、I2C、OPTION 章内容，
V1.6	2024.6.6	修改 1、6 章内容

Puya Semiconductor Co., Ltd.

附录 1

1.1 PY32L020 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    */
    LL_LPTIM_EnableIT_ARRM(LPTIM);

LL_LPTIM_Disable(LPTIM);
```

```
APP_delay_us(120);                                    //必须在此处增加120us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)
   {
   }
}
```

1.2 PY32L020 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)
  {
    /* 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(120);                    //必须在此处增加120us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 2

PY32L020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)

```
#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;
}
```