Preface

The series of PY32L020 microcontrollers features a high-performance 32-bit ARM® Cortex®-M0+ core, a wide voltage range MCU. They are embedded with up to 24 Kbytes of flash memory and 3 Kbytes of SRAM memory, operating at a maximum frequency of 24 MHz. A variety of products with different packaging types are included.

This application note will help you understand the attentions for using the PY32L020 modules and start quick development.

Table 1 Applicable Products

Type	Product Series
Microcontroller Series	PY32L020

Contents

1. PWR Configuration

- In order to improve the stability of the system, the watchdog function must be enabled.
- It is recommended to enable the watchdog in the Option and set the watchdog overflow time according to actual conditions through software.
- Once the watchdog is enabled, it cannot be turned off by software. Therefore, in low power consumption modes, an LPTIM timer should be used to wake up and feed the watchdog. (refer to example routines in Appendix 1)
- Before entering Stop mode, the MCU needs to disable the SysTick interrupt (HAL_SuspendTick()).
- In sleep mode, the CPU cannot be awakened by an event if the EXTI module clock and the CPU clock are sourced from the same clock but are divided. Interrupts must be used to wake up the CPU.

2. ADC Power-On Calibration

2.1 Attentions

- It is recommended to recalibrate when the working conditions of the ADC change. (changes in V_{CC} are the main factors affecting ADC offset, with temperature changes being secondary.)
- Before using the ADC module for the first time, a software calibration process must be added.

2.2 Operation Process

- Enable the ADC clock, ADCEN=1.
- Initialize the ADC.
- Calibrate the ADC.

2.3 Code Example

```
static void APP_AdcConfig()
{
LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_ADC1);          // Enable the ADC1
clock
  if (LL_ADC_IsEnabled(ADC1) == 0)
  {
    LL_ADC_StartCalibration(ADC1);                                  //Enable Calibration
#if (USE_TIMEOUT == 1)
    Timeout = ADC_CALIBRATION_TIMEOUT_MS;
```

```
#endif
while (LL_ADC_IsCalibrationOnGoing(ADC1) != 0)
      {
#if (USE_TIMEOUT == 1)                                  // Check if Calibration has Timed Out
         if (LL_SYSTICK_IsActiveCounterFlag())
         {
            if(Timeout-- == 0)
            {
            }
         }
#endif
      }
LL_mDelay(1);
   }
}
```

3. ADC Configuration

3.1 ADC Software Configuration

- Switching ADC channels requires to disable ADC.
- After enabling ADC, 8 ADC clock cycles delay is required before enabling conversion, otherwise it will affect the sampling accuracy.
- Before entering sleep mode, the ADC module needs to be reset.
- When the analog watchdog only monitors single channel 0, it does not work.
- When the ADC CLK division is greater than 1 and then the end of conversion (eoc) flag is detected, the eoc flag should be cleared after 1 ADC CLK.
- Directly driving high-power devices with GPIO can affect ADC sampling results.(e.g., digital tube display; it is not recommended to sample ADC when the digital tube is displaying, or to insert a 10-100 ohm resistor in series on each data line of the digital tube, which can be adjusted according to actual conditions.)
- Before switching ADC from continuous mode to non-continuous mode, the ADC must be disabled first, and then set to non-continuous mode.
- To enable the internal 1.5V VerfBuffer, the 1.2V VREFEN must also be enabled in software. (ADC->CCR. VREFEN=1)

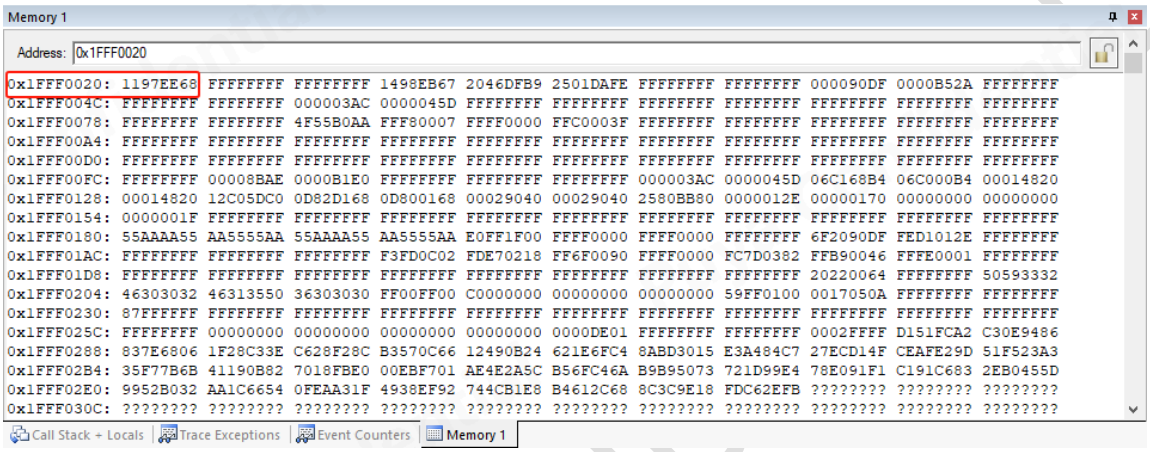
3.2 ADC Hardware Configuration

- The ADC does not support sampling V_{CC}/3.(Internal channel 10)
- The voltage on ADC channels must not exceed V_{CC}+0.3V (even if the ADC channel is

not configured for AD function), otherwise the ADC sampling will be abnormal.

3.3 Vreferint 1.2V

- The actual value of the Vreferint 1.2V which the chip uses is plasced in the information area (0x1FFF0020) in FLASH. (The high 16 bits are the actual value and the low 16 bits are the inverse code.) The procedure for reading Vreferint 1.2V is shown in Appendix 2:



- When using the internal reference voltage of 1.5V, Vreferint 1.2V must be enabled.
- When sampling the internal reference voltage of 1.2V, the result calculated through the ADC sampling time conversion formula should be at least 20 μs The methods are as follows:

a) Reduce the resolution.

b) Decrease the ADC clock frequency.

c) Increase the ADC sampling period.

The total conversion time is calculated as follows:

tCONV = Sampling Time + (Conversion Resolution+0.5) x ADC Clock Cycle

For example:

When ADC_CLK = 12MHz, the resolution is 12 bits, and the sampling time is 239.5 ADC clock cycles:

tCONV = (239.5 + 12.5) x ADC Clock Cycle=252 x ADC Clock Cycle= 21 μs

4. SPI Configuration

SPI Mode	Receive/Transmit mode	SPI Fastest Speed

Slave Full-Duplex	Receive	PCKL/16
Slave Full-Duplex	Transmit	PCKL/16
Master Full-Duple	Receive	PCKL/2
Master Full-Duple	Transmit	PCKL/2

- When the SPI operates in slave transmit mode, setting SPI->CR1.CPHA=0, data errors occur starting from the second frame, where the first data sent is the last data of the previous frame. This can be resolved by writing 0 and then 1 to SPI->CR1.SPE before each frame transmission.
- The SPI->SR.BSY bit is cleared during the last clock cycle during SPI communication. In Polling mode, ensure that the transmission of the previous frame is complete before updating the next frame data.

5. TIMER Configuration

- In the timer interrupt function, clearing the CC interrupt flag must wait for TIM_PSC*PCLK; otherwise, it may lead to failure in clearing the interrupt flag.
- When automatic output enable is used, PWM outputs with dead-time will periodically insert dead-time after braking. (The use of the braking function is not recommended.)

6. LPTIM Configuration

- When LPTIM uses RCC_CCIPR->LPTIMSEL to select PCLK as the clock source, the pre-scaler cannot be set to 1, otherwise LPTIM has a probability of running abnormally.
- When LPTIM uses the RSTARE function, the interval between two reads of the CNT registers should satisfy 4 LSI clocks.

6.1 LPTIM Continuous Mode

- LPTIM continuous mode must clear ARRMCF each time before entering Stop and wait for 1 LSI clock cycle*PSC factor. (Approx.40μs*PSC including programme execution time.)
- When changing the reload value of LPTIM, it is necessary to wait for 4 LSI clock cycles multiplied by the PSC coefficient. (Approx.160 μs*PSC, including the time for program execution.)

6.2 LPTIM Once Mode

- LPTIM once mode must clear ARRMCF and wait 3 LSI clock cycles before entering Stop each time. (Approx.120us, including programme execution time.)

- When changing the reload value of LPTIM, it is necessary to wait for 4 LSI clock cycles. (Approx.160 μs, including the time for program execution.)

7. COMP Configuration

7.1 COMP Hardware Configuration

- When the comparator's VINM input signal is an internal analog voltage source, an external capacitor (1nF) should be added to the VINP input channel to ground.

7.2 COMP Software Configuration

- The comparator only supports waking up the MCU from Sleep mode.

8. IO Backflow Current Driving MCU Operation

8.1 Attentions

- The backflow current on the IO which will make the MCU work when the VCC of the MCU is not powered up can be avoided by software configuration.

8.2 Operation Procedure

- Hardware: Corresponding IO ports need to be strung with 100Ω~1KΩ resistors.
- The corresponding IO outputs need to be set to open-drain mode before power-on initialization.
- Delay for 5ms.
- Normal program initialization.

8.3 Code Example

```
int main(void)
{
    LL_IOP_GRP1_EnableClock(LL_IOP_GRP1_PERIPH_GPIOA);     /*Enable the GPIOA clock*/
    LL_GPIO_SetPinMode(GPIOA,  LL_GPIO_PIN_1,  LL_GPIO_OUTPUT_OPENDRAIN);
                                                          /*Configure pin PA1 as open-drain output */
    LL_mDelay(5);                                         /*Delay for 5ms*/
}
```

9. IWDG Does Not Support Freeze Function

9.1 Attentions

- IWDG cannot be turned off after it is enabled and the freeze function is not effective. When using IWDG in conjunction with Stop mode, LPTIM should be used for timed wake-ups to feed the watchdog. The following table shows the impact of timed wake-ups on power consumption.

Note: The following operating power consumption is calculated with the system clock at 24MHz				
Low-power Run Time(ms)	Stop Mode Current (μA)	Wake-up Run Time (ms)	Operating Mode Current (μA)	Average Power Consumption (μA)
500	1.7	1	1100	3.892215569
1000		1		2.797202797
2000		1		2.248875562
3000		1		2.065978007

9.2 Operation Procedure

- Configure all wake-up sources to event wake-up.
- Before entering Stop mode, disable all interrupts and clear the corresponding flags.
- Enable LPTIM, enable the timed wake-up function, and ensure that the LPTIM timing is less than the IWDG overflow time.

10. Option Configuration

- When mass production, Option operation needs to be configured in the Option Byte configuration of the porgammer, and the function that operates Option in the program should be blocked.
- It is recommended that the customer program enables write protection, which is set in Option, as shown in Figure 10-1 and Figure 10-2.

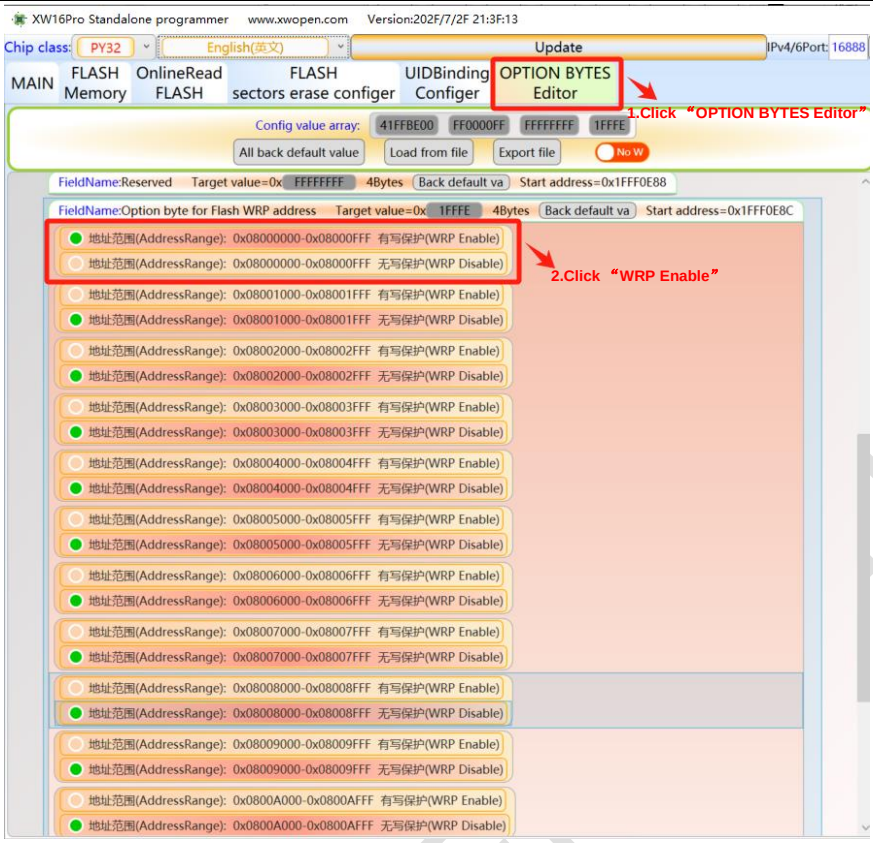


Figure 10-1 XW Operation Write Protection

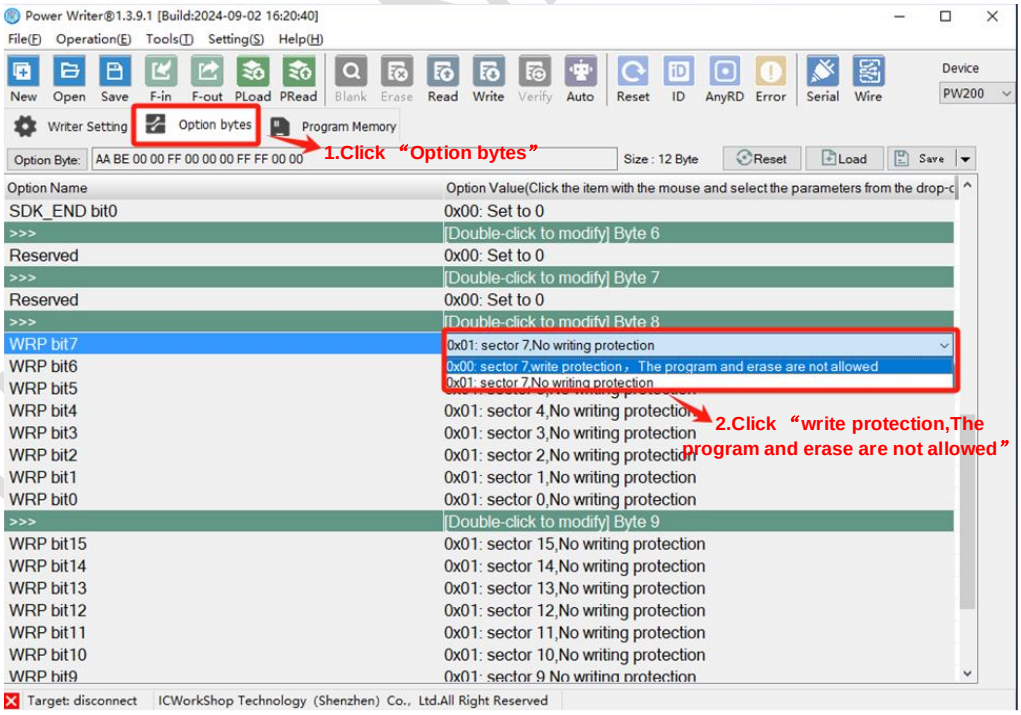


Figure 10-2 Power Write Operation Write Protection

- When configuring the Option of the programmer, you need to check the "Smart Reset" function or "Restart the chip after programming" (programmers typically have similar

options that need to be selected). The specific steps are shown in Figures 10-3 and 10-4.

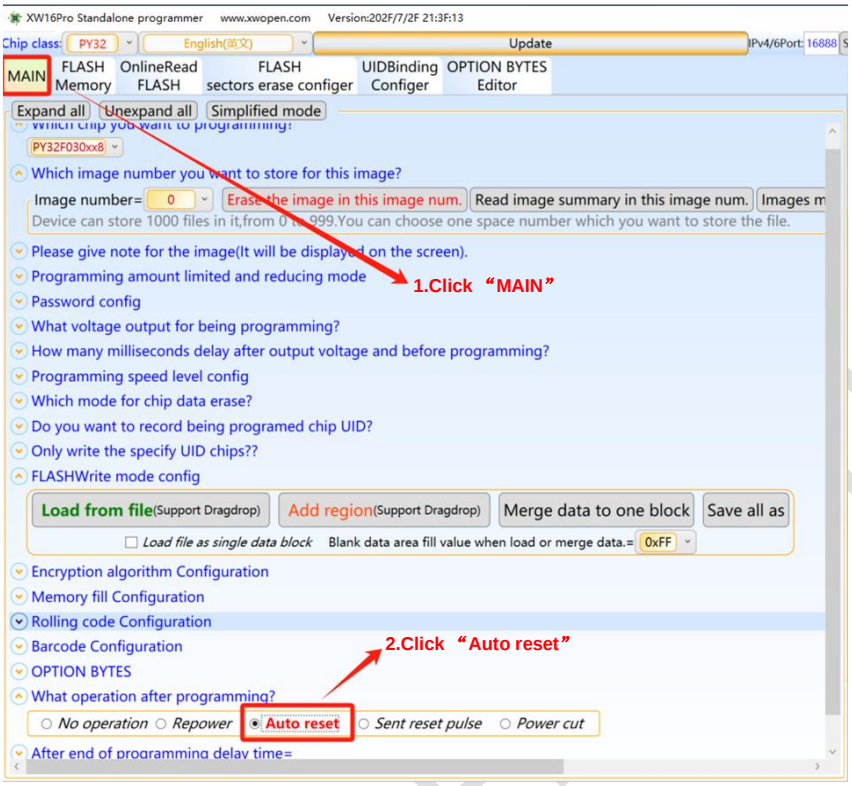


Figure 10-3 XW Operation Click “Auto reset”

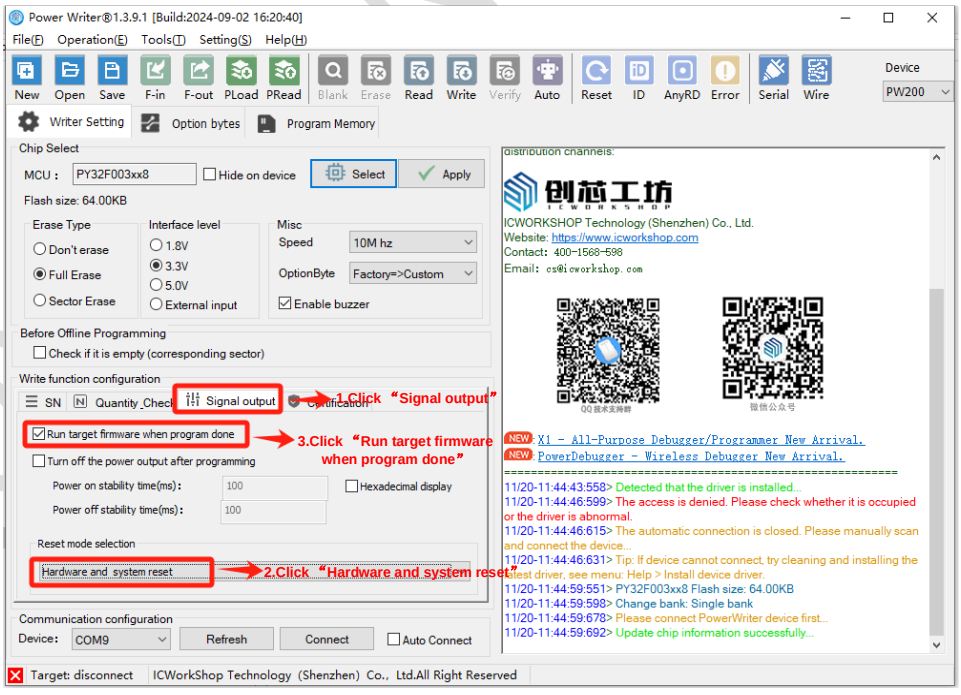


Figure 10-4 Power Write Operation Click “Restart the chip after programming”

11. GPIO Configuration

- Initializing other structures such as GPIOs needs to be assigned a value of 0 to avoid having an unfixed initial value.
- All GPIOs must not have a negative voltage of more than -0.3V on them.

12. I2C Slave Communication Precautions

- After sending a frame of data, the I2C slave will add 1 to the buffer pointer after the host re-addresses it, so the slave needs to re-initialise the buffer pointer in the address interrupt.
- When the I2C slave needs to clock-stretch after receiving each byte, the first two bytes from the I2C master to the slave cannot be clock-stretched.

13. Version History

Version	Date	Update Records
V1.0	2023.06.15	Initial release
V1.1	2023.07.15	Update CMP Hardware Design
V1.2	2023.11.10	Add the average power consumption after LPTIM wake-up in Chapter 11 Add that LPTIM using event wake-up does not require waiting time in Chapters 7 and 8
V1.3	2023.11.17	Modify the content of Chapter 11
V1.4	2023.11.22	Modify the LPTIM-related content in Chapters 7, 8, and 11
V1.5	2024.03.28	Add content for PWR, ADC, COMP, GPIO, I2C, and Option chapters
V1.6	2024.06.06	Modify the content of Chapters 1 and 6
V1.7	2024.08.30	Merge Chapters 3, 4, 5, and 6; merge Chapters 9 and 10; merge Chapters 11 and 12 Modify the content of Chapters 1, 3, 4, 5, and 10
V1.8	2025.06.18	Modify COMP module contents

Puya Semiconductor Co., Ltd.

Appendix 1

1.1 The routine of using timed wake-up to feed dog in the low power mode of PY32L002B (LL Library)

```
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);                    // A delay of more than 120 microseconds must be added here
/* 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 The routine of using timed wake-up to feed dog in the low power mode of PY32L002B (HAL Library)

```
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);
```

```
  /* 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);              // A delay of more than 120 microseconds must be added here
   /* Suspend SysTick interrupt */
    HAL_SuspendTick();
  /* 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();
    }
    /* Resume Systick */
    HAL_ResumeTick();
    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)
    {
    }
}
```

Appendix 2

2.PY32L020 reads the Vreferint 1.2V actual value stored in the information area (see 3.3 for specific address).

```
#define HAL_VREF_INT                    (*(uint8_t *)(0x1fff3023))
#define HAL_VREF_DEC                    (*(uint8_t *)(0x1fff3022))
#define vref_int        (*(uint8_t *)(HAL_VREF_INT))          //Store the integer part of the reference voltage
#define vref_dec        (*(uint8_t *)(HAL_VREF_DEC))          // Store the fractional part of the reference voltage
float vref;                 //Reference voltage value

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);


  //Initialize all peripherals, flash interface，systick
    vref = data_vref_int/10;        //Calculate the reference voltage
    vref = vref + ((data_vref_int%10)*0.1 + data_vref_dec*0.001);
    return vref;
}
```