WT32L064_032
Peripheral Functions and Programs
User Guide

Rev. 1.0
May 2022

Disclaimers

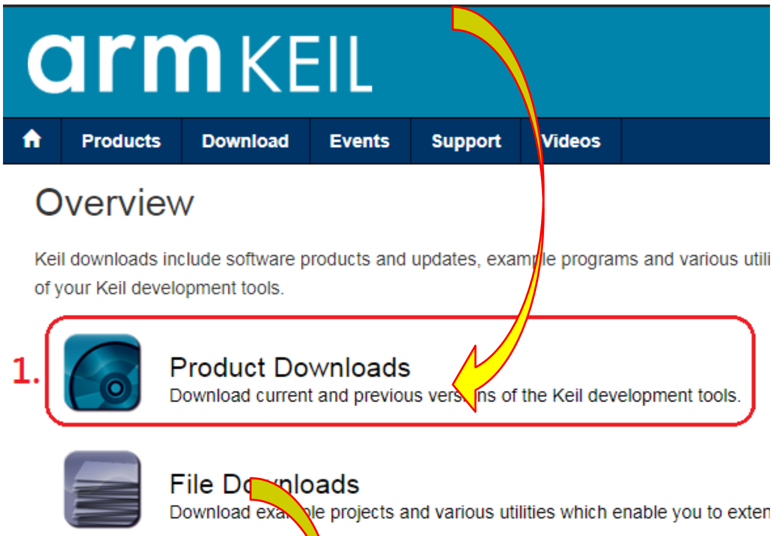
Right to make change –

This document provides technical information for user. Weltrend Semiconductor, Inc. reserves the right to make change without further notice to any products herein.

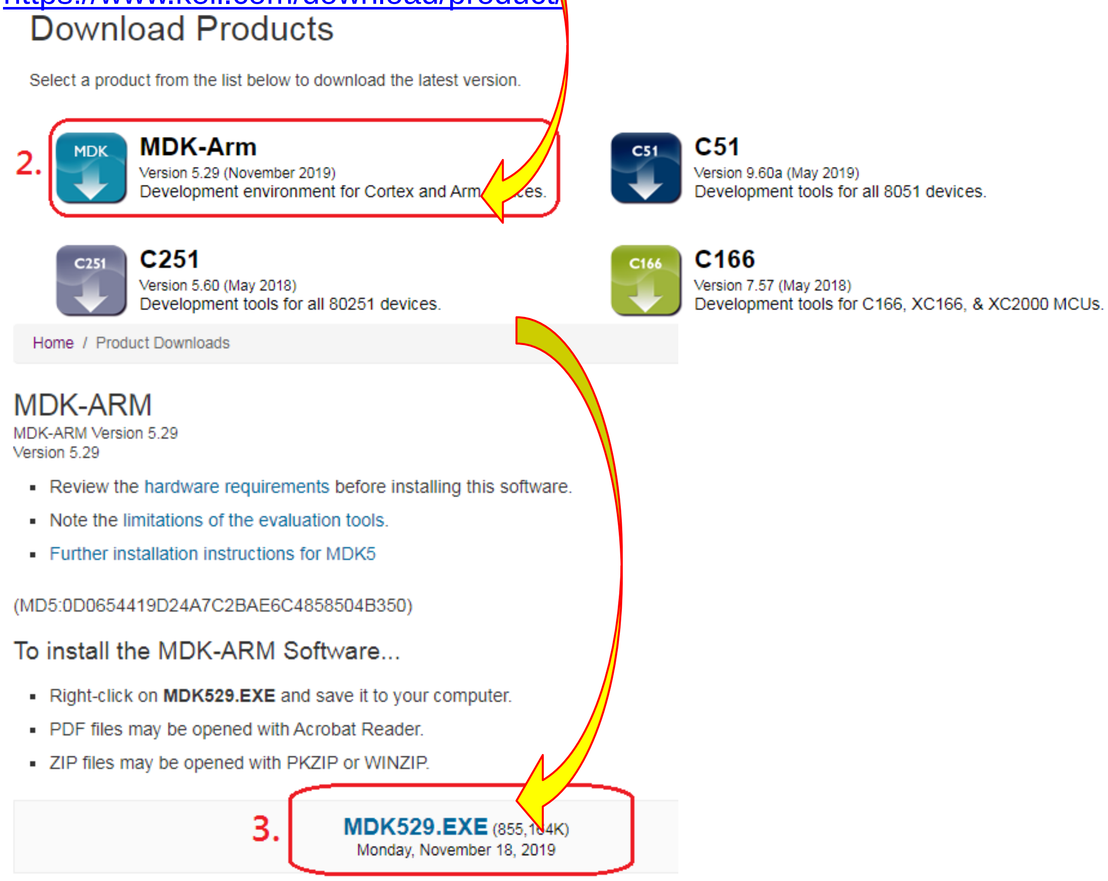
Table of Contents

1. ARM-MDK Installation & Environment setting

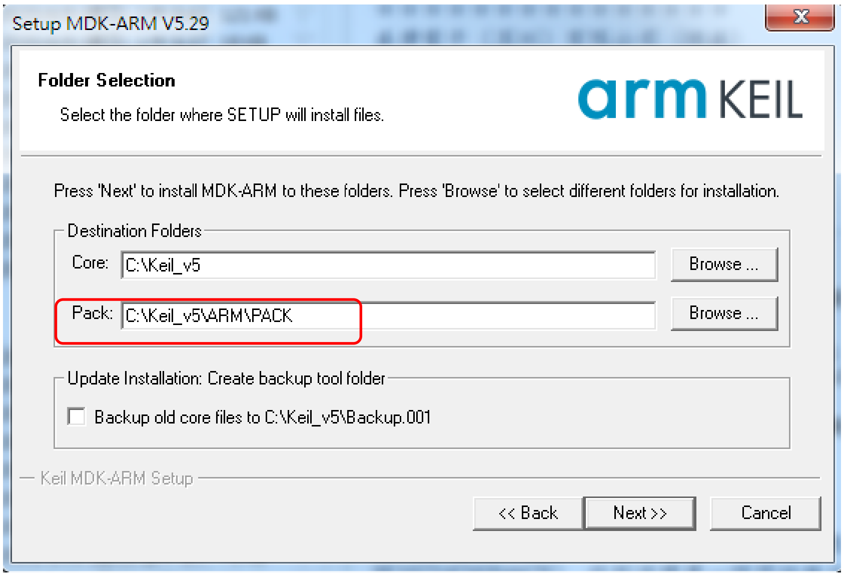
(Step 1) **Please download ARM-MDK** https://www.keil.com/download/



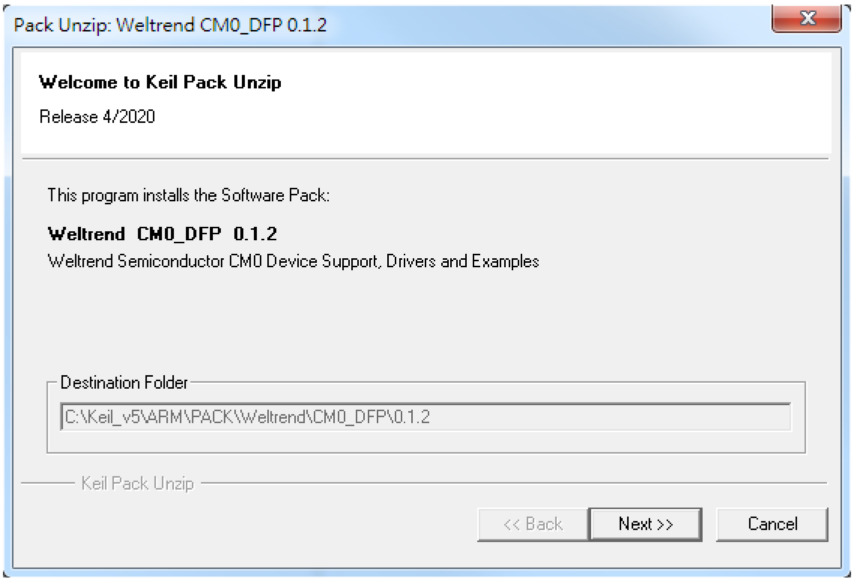
https://www.keil.com/download/product/



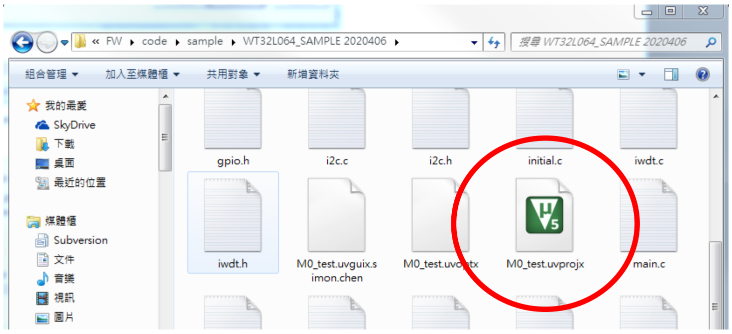
The default PACK path will be asked during installation, please specify **C:\Keil_v5\ARM\PACK** as follows to avoid subsequent PACK installation problems.



(Step 2) After downloading and installing MDK, please install Weltrend PACK file (Weltrend.CM0_DFP.0.1.x.pack) in your PC.



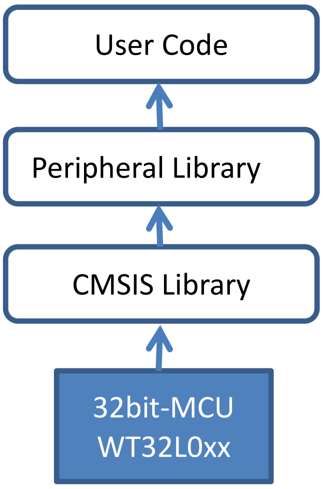
(Step 3) After installing the ARM-MDK, the basic 32KB is free for use, or you can purchase the software by yourself. After installation, please open the relevant WT32L064 project on your computer for compilation.



2. CMSIS Middleware Driver

2.1 Definition:

ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS) is a set of firmware libraries that can drive ARM processors. The firmware interface provides a standard function directly for the peripheral with the same name and is easy to use, which can be used repeatedly by software, reducing developing time for microcontroller developers. Based on this framework, the manufacturer provides a set of basic peripheral applications of sample programs or peripheral libraries, which can directly focus on the application side to speed up program operation and editing.



2.2 Application:

The purpose of CMSIS is to describe the function corresponding to the register control of the MCU. For users, standardized functions such as ADC_StartOfConversion() can be used, and the peripheral library (PL) provides the operation and example of the function.

EX: Program file main.c content

```
main() {
API_AverADCData()
}
```

Program file wt32l0xx_pl_adc.c for the peripheral application.

```
API_AverADCData() {
ADC_StartOfConversion();   // Using the CMSIS standard, call peripheral functions to average filter ADC
……
}
```

Contents of CMSIS-driven file wt32l064_adc.c

```
void ADC_StartOfConversion(void)
{
   ADC->ADCCR |= (uint32_t)ADC_START;   // Actually corresponds to the MCU register address
}
```

2.3 CMSIS Content:

After installing the WT32L064 PACK, the default CMSIS path is ***C:\Keil_v5\ARM\Packs\Weltrend\CM0_DFP\0.1.x \WT32L064\StdPeriph_Driver***. The header file is placed in the **Include** folder, the original file is placed in the **Source** folder, and which contents include the basic settings for all peripherals of WT32L064. The files list is as follows.

File Name	Description
wt32l064_adc	Analog detect ADC related function
wt32l064_crc32	CRC32 function
wt32l064_crs	Calibrated IC Internal Frequency related function
wt32l064_dac	Analog output DAC related function
wt32l064_dma	DMA related function
wt32l064_flash	EEPROM programming FLASH related function
wt32l064_gpio	GPIO related function
wt32l064_i2c	I2C related function
wt32l064_i2s	I2S related function
wt32l064_iwdt	IWDT Independent Watchdog related function
wt32l064_pmu	PMU Power Control Unit related function
wt32l064_pwm	PWM related function
wt32l064_rcc	RCC frequency control unit related function
wt32l064_rtc	RTC Timer related function
wt32l064_spi	SPI related function
wt32l064_timer	TIMER related function
wt32l064_usart	UART related function
wt32l064_usbd	USB related function
wt32l064_wwdt	WWDT Window Watchdog related function

At the beginning of each file, there is a brief description of the purpose and function of the file, and each function inside also describes the function and parameters. For example, in the file wt32l064_gpio.c, the content of the GPIO_SetBits() function is as follows.

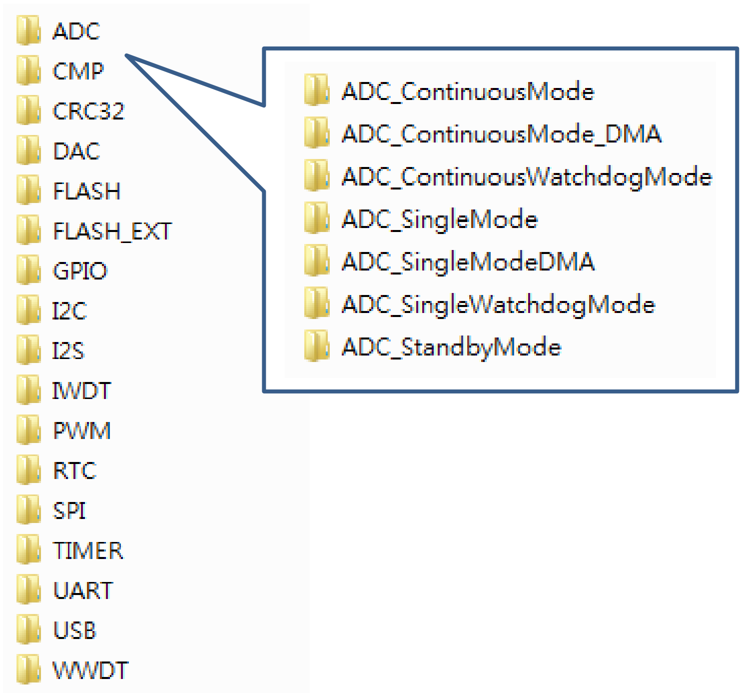
```
/**
  * @brief    Sets the selected data port bits.
  * @param    GPIOx: where x can be (A, B, C or D) to select the GPIO peripheral.
  * @param    GPIO_Pin: specifies the port bits to be written.
```

```
  * @note      This parameter can be GPIO_Pin_x where x can be 0 ~ 15 for GPIOA, GPIOB, GPIOC and
GPIOD.
  * @retval None
  */
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  /* Check the parameters */
  assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
  assert_param(IS_GPIO_PIN(GPIO_Pin));
  GPIOx->BT_SET = GPIO_Pin;
}
```

@Brief: The main function is Bit setting
@param GPIOx: target PORT
@param GPIO_Pin: target PIN
@note: Supplementary explanation PIN has 0~15 and has PortA~D

3. Structure of PACK Examples Program

There are basic example programs for various application units. After PACK is installed, please refer to the following path ***C:\...\Arm\Packs\Weltrend\ CM0_DFP\0.1.x \WT32L064\Examples***. There are sub-units in the folder that contain original files and Project. The following is an ADC sample program, which can be classified into single conversion and continuous conversion according to its functions, as shown in the figure below.



The folder contains the sample programs of the corresponding functions. For example, ADC->ADC_SingleMode is a single analog conversion function. There is readme.txt in this folder for simple instructions.

3.1 Examples folder

Root Folder	Folder Name	Description
ADC	ADC_ContinuousMode	Continuous ADC detect
	ADC_ContinuousMode_DMA	Use DAM as continuous ADC detect
	ADC_ContinuousWatchdogMode	Use consecutive ADC as WDT detect
	ADC_SingleMode	Single ADC detect
	ADC_SingleModeDMA	Using DAM as a single ADC detect

Root Folder	Folder Name	Description
	ADC_SingleWatchdogMode	Use single ADC as WDT detect
	ADC_StandbyMode	Using Low Power ADC Mode
CMP	CMP	Comparator Example
CR32	CRC32	CRC32 calculation example
DAC	DAC	DAC output example
	DAC_HighCurrent	DAC high thrust output example
FLASH	FLASH_PROGRAM	Example of programming area (EEPROM)
	FLASH_PROGRAM_INT	Program area (EEPROM) interrupt example
FLASH_EXT	FLASH_OB_EEPROM	Example of programming data area (EEPROM)
	FLASH_OB_LEVEL	Example of programming data area (EEPROM)
	FLASH_OB_READ_PROTECTION	Anti-read encryption in the data area (OB)
	FLASH_OB_WRITE_PROTECTION	Write-proof encryption in the data area (OB)
GPIO	GPIO	GPIO Basic Example
	GPIO_Bit_Set_Reset	Example of setting GPIO bits
	GPIO_Input	Example of setting GPIO input
	GPIO_Interrupt	Example of setting GPIO interrupt
	GPIO_Output	Example of setting GPIO output
	GPIO_Toggle	Example of setting GPIO output inversion
I2C	I2C_Master_Slave_DMA_FLAG	I2C Slave mode and DMA transfer
	I2C_Master_Slave_DMA_INT	I2C Slave mode and DMA interrupt
	I2C_Master_Slave_FLAG	I2C Slave Mode

Root Folder	Folder Name	Description
	I2C_Master_Slave_FLAG_EEPROM	I2C Slave mode and EEPROM programming
	I2C_Master_Slave_INT	Each group of I2C master and slave mode transfers to each other
I2S	I2S_DMA	I2S Slave mode and DMA transfer
	I2S_INT	I2S Slave mode and DMA interrupt
	I2S_POLLING	I2S Slave mode
IWDT	IWDT	Watchdog setting example
PWM	PWM	PWM pulse modulation example
RTC	RTC_1sec	RTC Timer setting 1 second example
	RTC_Alarm	RTC Timer setting Alarm example
SPI	MSPI_DMA_FLAG	SPI using DMA transfer example
	MSPI_DMA_INT	SPI using DMA transfer and interrupt example
	MSPI_FLAG	SPI transfer example
	MSPI_FLAG_FLASH_MX25L4006	SPI with MX25L4006 transmission example
	MSPI_INT	SPI transfer and interrupt example
TIMER	TMR_Capture_Mode	Timer capture mode example
	TMR_Compare_Mode	Timer compare mode example
	TMR_Counter_Mode	Timer counter mode example
	TMR_DMA_Mode	Example of using Timer with DMA
	TMR_PWM_MODE	Timer output PWM usage example
	TMR_Timer_Mode	Timer common timing example

Root Folder	Folder Name	Description
UART	UART_DMA	Serial transfer with DMA usage example
	UART_HalfDuplexMode	Serial transfers use the half-duplex example
	UART_InterruptAndFlagManage	Serial transfer using interrupt example
	UART_IrDA_Mode	Serial transfer using IRDA example
	UART_TxRx	Serial Transmission Simultaneous Transmit and Receive Example
USB	USB_HID	HID KEYBOARD Simple Example
	USB_HID_AUDIO_WM8731	HID with I2S to play WM8731 music
WWDT	WWDT	Windows Watchdog

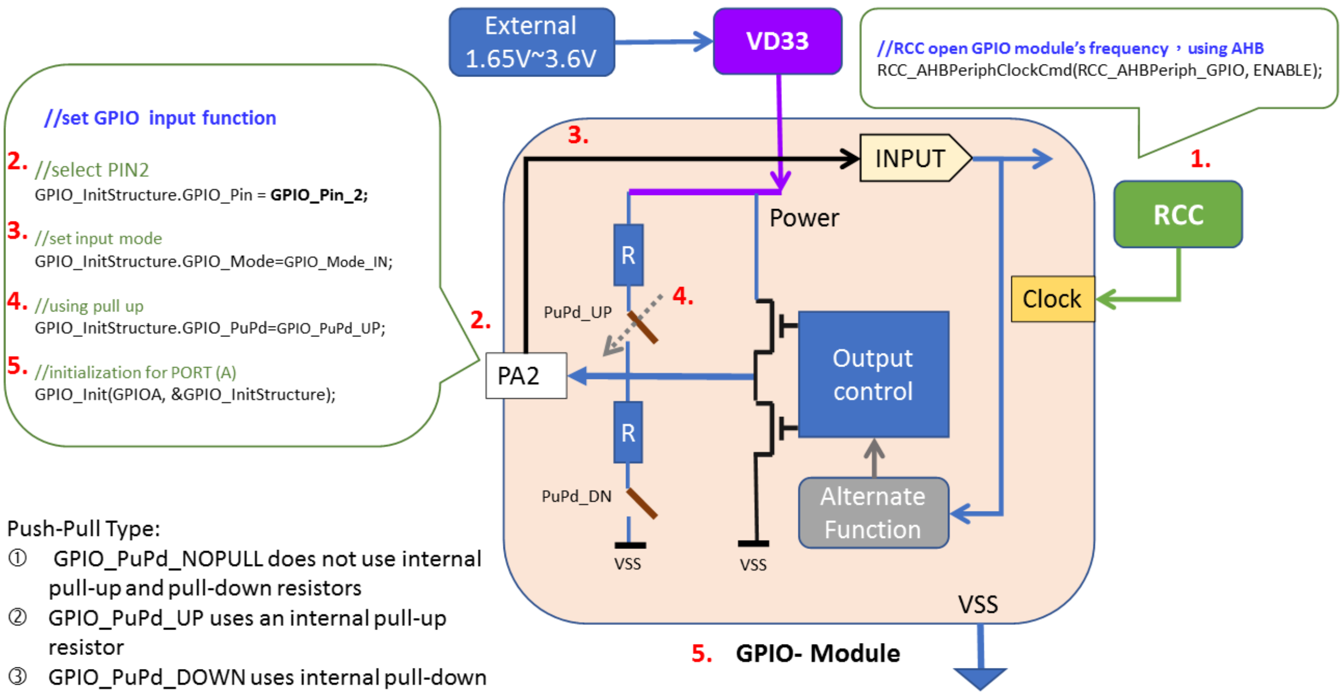
4. GPIO function description

Use the following illustration that PA2 of GPIO as input and PC4~PC7 as output. The action flow is as follows.

4.1 MCU performs GPIO initialization

The contents of using PA2 are as follows, please refer to the function InitialGpio ( ) of the peripheral library wt32l0xx_pl_gpio.c

(Step 1) Set the RCC (Reset and Clock Control) module to enable the clock to be provided to the GPIO, as shown in step 1 below.
(Step 2) Set GPIO, select PIN2 as an example here, as shown in step 2 below.
(Step 3) Set the input or output mode, the following example IO selects INPUT, as shown in step 3 below.
(Step 4) Set the pull-up or pull-down impedance. The following example IO selects the pull-up, as shown in step 4 in the following figure.
(Step 5) Set the Port-A module of GPIO to initialize and write to the register, as shown in step 5 in the following figure.



Push-Pull Type:
① GPIO_PuPd_NOPULL does not use internal pull-up and pull-down resistors
② GPIO_PuPd_UP uses an internal pull-up resistor
③ GPIO_PuPd_DOWN uses internal pull-down resistor

4.2 Read GPIO input value

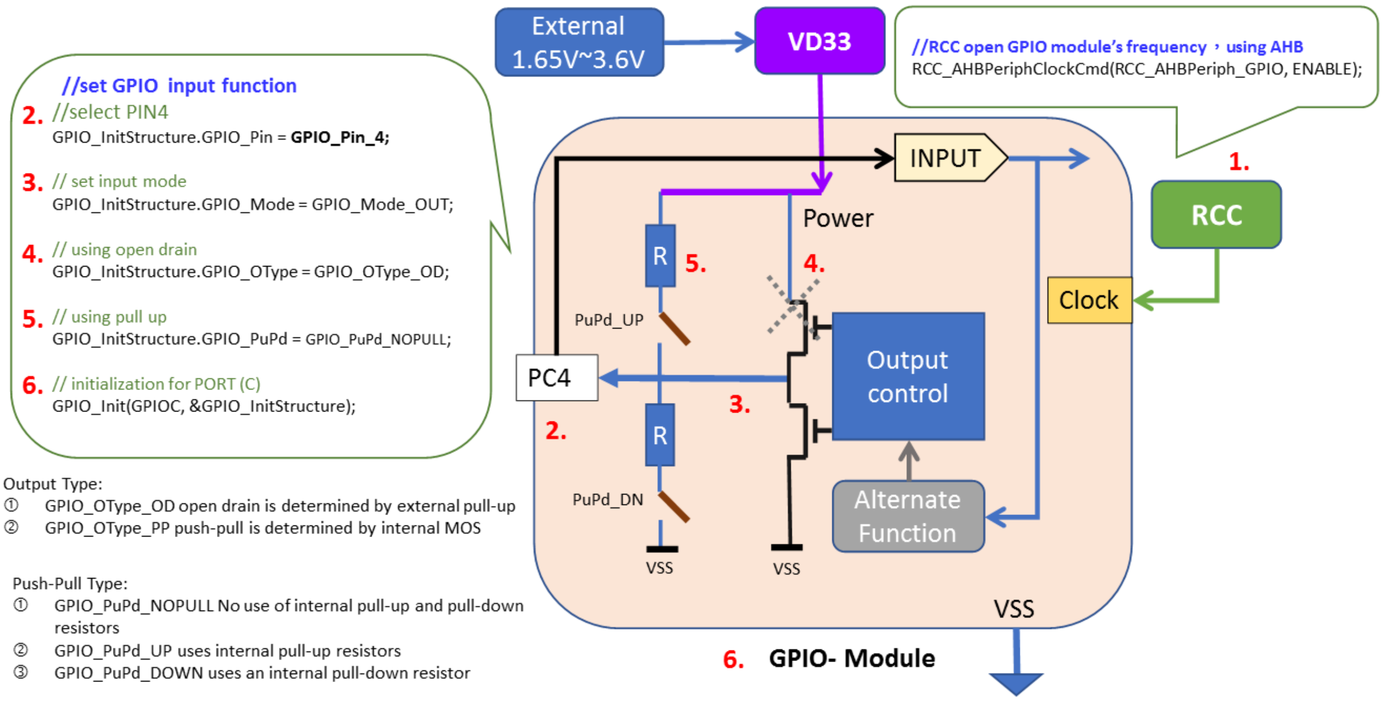
Use GPIO_ReadInputDataBit() to read the BIT data value. For example, when PA2=LO, the execution input is written as follows

```
if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_2) == 0) {//………Write the corresponding function
}
```

4.3 Set GPIO output value

After the MCU is powered on, initialize PC4~PC7. The content is as follows, or you can refer to the example function InitialGpio ( )

(Step 1) Set the RCC to enable the clock to be provided to GPIO, as shown in step 1 in the following figure.
(Step 2) Set GPIO, select PIN4, as shown in step 2 below.
(Step 3) Set the input or output mode, as shown in the following example, IO selects OUTPUT, as shown in step 3 in the following figure.
(Step 4) Set the output mode, there are push-pull and open-drain. In the example below, open-drain is selected, as shown in step 4 in the figure below.
(Step 5) Set the pull-up or pull-down impedance. In the following example, IO selects no pull-up, as shown in step 5 in the figure below.
(Step 6) Initialize the Port-C module of the GPIO, and write to the register, as shown in step 6 in the following figure.



4.4 Example program GPIO

Refer to the function InitialGpio( ) in wt32l0xx_pl_gpio.c. The following programs are executed in sequence with reference to the above steps 1~6

```
void InitialGpio(void)
{
    GPIO_InitTypeDef    GPIO_InitStructure;
```

GPIO_InitTypeDef
Refer to CMSIS definition

1 // RCC enables the working frequency of the GPIO module and uses AHB

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIO, ENABLE);

// set General GPIO pin INPUT
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;            // Set input mode
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;// Use input preset to pull up, but power saving mode
consumes power
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;               // Select PIN2
GPIO_Init(GPIOA, &GPIO_InitStructure);                  // Initialize for single PORT (A-D)

#if(ENABLE_LED_BLINK==ON)          // Determine whether to output the light signal
// set General GPIO pin PC4
2. GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;    // select PIN4
3. GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;        // Set output mode
4. GPIO_InitStructure.GPIO_OType = GPIO_OType_OD; // Set open drain type
5. GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;     // Set no pull up, no pull down

6. GPIO_Init(GPIOC, &GPIO_InitStructure);               // Initialize for single PORT-A
GPIO_SetBits(GPIOC, GPIO_Pin_4);                        // Set PORTC PIN4 output HI
 //........omit
#endif
```

GPIO_SetBitsOutput HI potential
GPIO_ResetBits Output LO potential

5. UART function description

Please refer to the following illustration to perform data transfer using UART0 or UART1.

5.1 Initialize UART after MCU is powered on

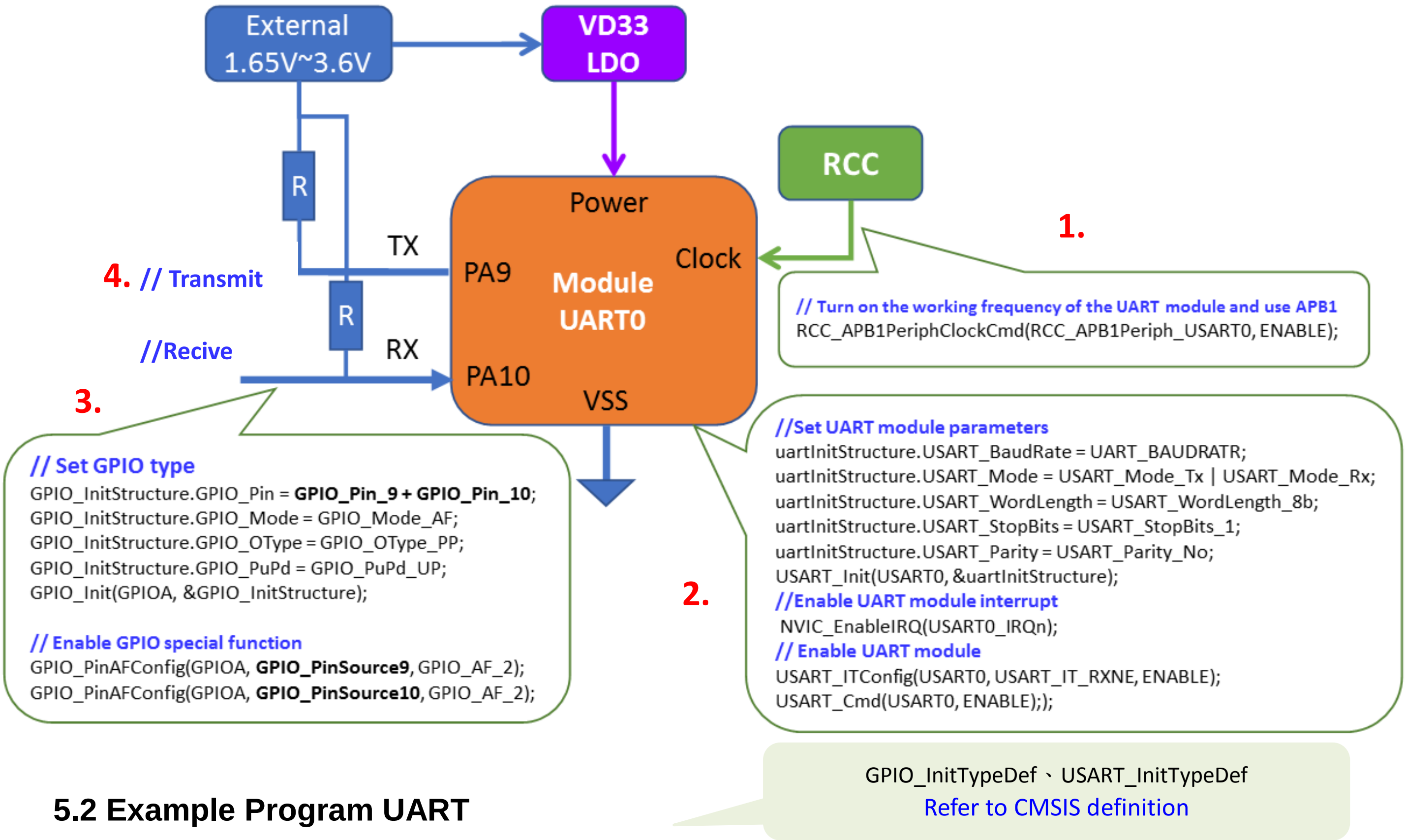
As shown in the following steps 1~4, you can refer to the peripheral library wt32l0xx_pl_uart.c to use the function InitialUart0( ) or InitialUart1( ).

(Step 1) Set the RCC to enable the clock to be used by the UART, as shown in step 1 in the following figure.

(Step 2) Set the parameters of the UART module, as shown in step 2 below.

(Step 3) Set the GPIO type (the IO is set last, to avoid the signal from being poured into the module whose status is not determined), as shown in step 3 in the following figure.

(Step 4) Transmit UART data, as shown in step 4 below.



5.2 Example Program UART

Please refer to the function InitialUart0 ( ) of wt32l0xx_pl_uart.c, the following programs are executed in sequence with reference to the above steps 1~4.

```
USART_InitTypeDef uartInitStructure;          // Initialization use, structure declaration
GPIO_InitTypeDef    GPIO_InitStructure;           // Initialization use, structure declaration
USART_DeInit(USART0);                             // Clear the initialization of UART0
```

1. // Turn on the working frequency of the UART module and use APB1

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART0, ENABLE);        // Input APB1 clock
```

2.

```
// Set UART module parameters
uartInitStructure.USART_BaudRate = UART_BAUDRATR;                 // Set Baud Rate
uartInitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;        // Set TX +RX function enable
uartInitStructure.USART_WordLength = USART_WordLength_8b;           // Setting Transmission length 8bit
uartInitStructure.USART_StopBits = USART_StopBits_1;              // Set 1 stop bit
uartInitStructure.USART_Parity = USART_Parity_No;              // Set whether to use the Parity bit
#if(ENABLE_OVER_SAMPLE8==ON)
USART_OverSampling8Cmd(USART0, ENABLE);                  // Whether to use Over Sampling acceleration
#endif
USART_Init(USART0, &uartInitStructure);                  // Do UART0 initialization

#if(ENABLE_INT_UART0==ON)
USART_ITConfig(USART0, USART_IT_RXNE, ENABLE);                 // Set UART0 interrupt type
NVIC_EnableIRQ(USART0_IRQn);                                   // Enable UART0 interrupt function
#endif
USART_Cmd(USART0, ENABLE);                                     // Enable UART0 module function
```

3.

```
// Set GPIO type
#if(SELECT_UART0_CH_A==ON)                                     // If you choose A channel
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 + GPIO_Pin_10;      // Select GPIO pin
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;                  // Use AF type
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;               // Selected GPIO push-pull or open-drain
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;                 // Selected GPIO pull-up or pull-down type
GPIO_Init(GPIOA, &GPIO_InitStructure);                       // Do GPIO initialization

// Enable GPIO special function
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_2); // Select the AF function, there are 0~5 types
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_2); // Select the AF function, there are 0~5 types
```

4.

```
// emission
printf("CH-A,Baud=%d,", UART_BAUDRATR);                      // Output UART0 data
#else                                                           // If B channel is selected
......... omit
#endif
```

5.3 UART for RX receiving data and TX transmitting data

The UART interrupt function is used in conjunction with UART0_Handler( ) to receive data in RX. The writing method is as follows.

```
void UART0_Handler(void)
{
    if (USART_GetITStatus(USART0, USART_IT_RXNE) != RESET) //get Rx Flag
    {
        unsigned char data = USART_ReceiveData(USART0); //get Rx Data
        //...........To Do
    }
  USART_ClearITPendingBit(USART0, USART_IT_RXNE);   //Clear UART RX-INT Flag
}
```

To transmit data, use ARM default printf( ) with fputc( ), or use the example DRV_Printf( ) to output data

```
EX:   printf("Hello World!");
      Drv_Printf("Baud=%d,", UART_BAUDRATR);
```

6. ADC function description

Please refer to the following illustration to perform analog signal input using the ADC, the action flow is as follows.

6.1 MCU performs ADC initialization

After the MCU is powered on, the initial content is as follows, you can refer to the peripheral library wt32l0xx_pl_adc.c to use the function InitialAdc( ).

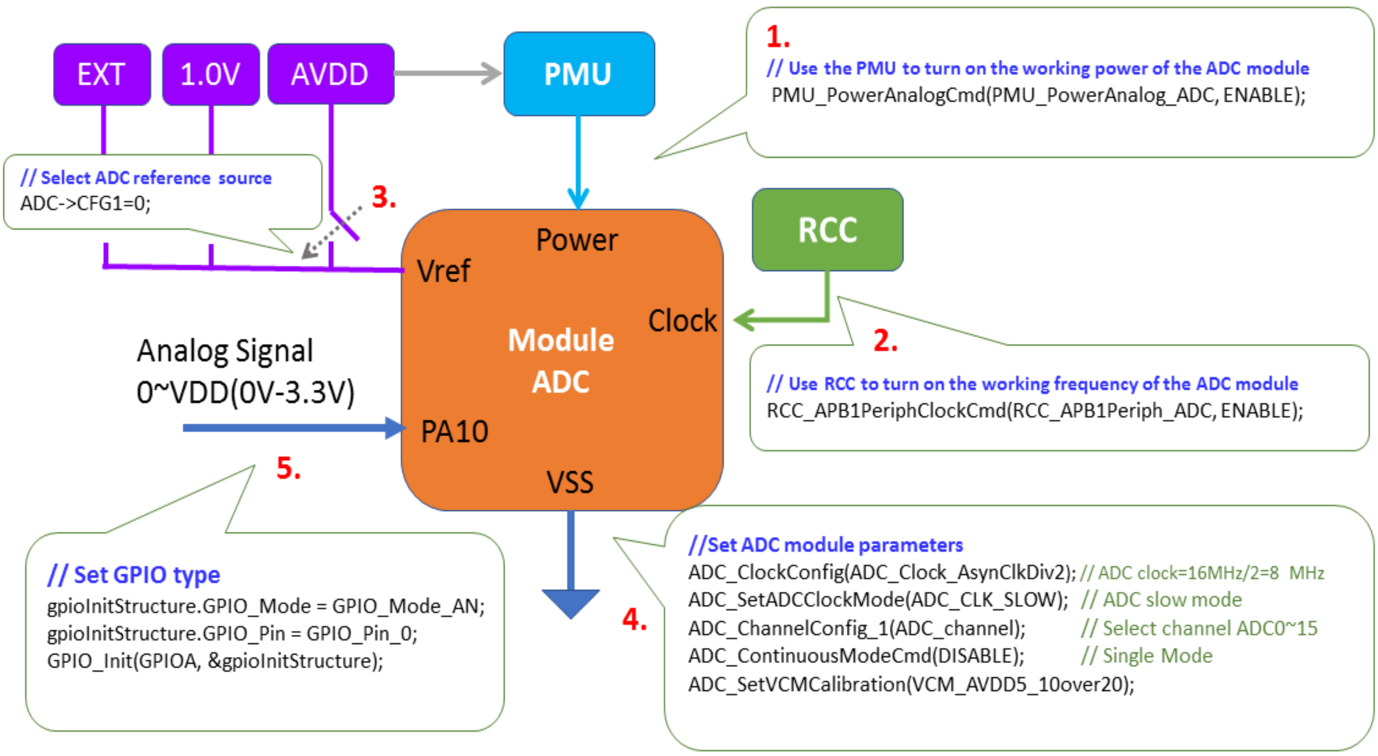
(Step 1) Set the PMU (Power Management Unit) to turn on the analog power supply for ADC use, as shown in step 1 below.

(Step 2) Set the RCC to enable the clock to be used by the ADC, as shown in step 2 in the figure below.

(Step 3) Select the reference potential source, there are AVDD, B-GAP 1.2V, External Pin input, as shown in step 3 below.

(Step 4) Set the ADC module parameters, set the conversion channel and speed, as shown in step 4 below.

(Step 5) Set the GPIO type (the IO is set last to prevent the signal from being poured into the module with undetermined status), as shown in step 5 in the figure below.



6.2 Example Program ADC

Refer to the function InitialAdc ( ) of wt32l0xx_pl_adc.c. The following programs are executed in sequence with reference to the steps 1~5 as mentioned above.

```
void InitialAdc(uint16_t ADC_channel)
{
   GPIO_InitTypeDef              gpioInitStructure;        // IO initialization use, variable structure

1. //---------- PMU -------------------------------------------
   // Use the PMU to turn on the working power of the ADC module
   PMU_PowerAnalogCmd(PMU_PowerAnalog_ADC, ENABLE);          // Powering the ADC

   //---------- RCC -------------------------------------------
2.    // Use RCC to turn on the working frequency of the ADC module and select APB1
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_ADC, ENABLE);

   //---------- ADC -------------------------------------------
   ADC_StopOfConversion_1();   // Stop ADC conversion first, if it was turned on before

3. #if(ADC_VREF_SEL_AVDD==ON)              // Select ADC reference potential source AVDD, 1.2V, EXT (external)
   ADC->CFG1 = 0;
#elif(ADC_VREF_SEL_1P2==ON)                // Vref= 1.2V
   ADC->CFG1 = 0x800;
#elif(ADC_VREF_SEL_EXT==ON)
   ADC->CFG1 = 0x18000;
   gpioInitStructure.GPIO_Pin = GPIO_Pin_0;              //VREF=PB0
   gpioInitStructure.GPIO_Mode = GPIO_Mode_AN;
   GPIO_Init(GPIOB, &gpioInitStructure);
#endif

4. ADC_ClockConfig(ADC_Clock_AsynClkDiv32);              // ADC clock = 16MHz / 32 = 500 kHz (4M,125K)

   ADC_SetADCClockMode(ADC_CLK_SLOW);                    // ADC slow mode
   ADC_ChannelConfig_1(ADC_channel);                     // Select channel ADC0~15
   ADC_ContinuousModeCmd(DISABLE);                       // Single Mode
   ADC_SetVCMCalibration(VCM_AVDD5_10over20);            // If AVDD is less than 1.8V, it needs to increase VCM
(Common-Mode)

#if(ADC_STANDBY_MODE==ON)
   ADC_StandbyCmd(ENABLE);                               // ADC standby mode, ADC clock must less 240KHz
#endif

#if(ENABLE_HW_ADC_AWD==ON)                 // If enabled, use AWD analog watchdog
   //...... omit
#endif

#if(ENABLE_FUNC_DMA==OFF)                  // If no DMA transfer is used, turn on the interrupt and judge that the
conversion is complete
   ADC->CFG1 |= 0x00000200;          // Enable ADC    interrupt
   ADC_ITConfig(ADC_IT_EOC, ENABLE);
   NVIC_EnableIRQ(ADC_IRQn);       // ADC    interrupt enable
#endif
```

5.

```
//----------------------------------------------------
// ADC channel setting, IO/Analog switching according to its PA~PC difference
    gpioInitStructure.GPIO_Mode = GPIO_Mode_AN;
    if (ADC_channel <= ADC_Channel_7)
    {
            switch (ADC_channel)
            {
            case ADC_Channel_0:   gpioInitStructure.GPIO_Pin = GPIO_Pin_0;   break;
            case ADC_Channel_1:   gpioInitStructure.GPIO_Pin = GPIO_Pin_1;   break;
            case ADC_Channel_2:   gpioInitStructure.GPIO_Pin = GPIO_Pin_2;   break;
            case ADC_Channel_3:   gpioInitStructure.GPIO_Pin = GPIO_Pin_3;   break;
            case ADC_Channel_4:   gpioInitStructure.GPIO_Pin = GPIO_Pin_4;   break;
            case ADC_Channel_5:   gpioInitStructure.GPIO_Pin = GPIO_Pin_5;   break;
            case ADC_Channel_6:   gpioInitStructure.GPIO_Pin = GPIO_Pin_6;   break;
            case ADC_Channel_7:   gpioInitStructure.GPIO_Pin = GPIO_Pin_7;   break;
            }
            GPIO_Init(GPIOA, &gpioInitStructure);            //Port-A its 1 channel, set ADC
    }
//.........The following IO settings are omitted
  }
```

6.3 Perform ADC detection and convert data

The sample program is as follows.

```
uint32_t ADC_Convert(uint16_t ADC_channel)
{
    uint32_t AD_buff;                            //12bit ADC buffer;

    ADC_StopOfConversion_1();                    // Stop ADC conversion first
    ADC_ChannelConfig_1(ADC_channel);            // Select ADC_channel, channel enable
    __nop();    __nop();    __nop();    __nop();
    gu16AdcFinish = 0;                           // clear transition flag

    ADC_StartOfConversion_1();          // Start ADC conversion

    while (gu16AdcFinish == 0);         // Wait for ADC conversion to complete Flag set
    AD_buff = ADC_GetConversionValue();    // Get the ADC converted value
    return AD_buff;
}
```

This line function is the write scratchpad command:
ADC->ADCCR |= (uint32_t)ADC_START;

7. DAC function description

Please refer to the following illustrations, use the DAC to perform analog signal input, the action flow is as follows.

7.1 MCU performs DAC initialization

The contents of initialization after power-on are as follows, you can refer to the peripheral library wt32l0xx_pl_dac.c to use the function InitialDac( ).

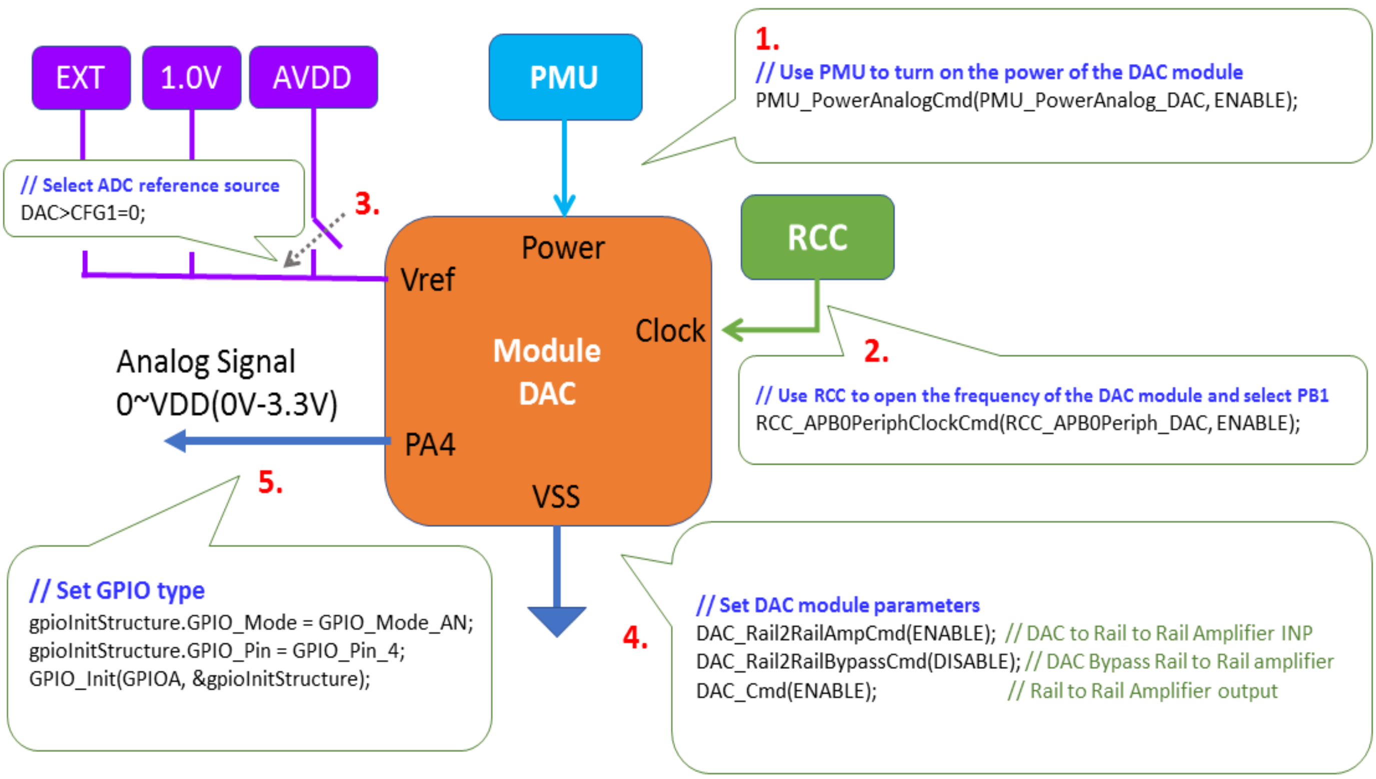
(Step 1) Set the PMU (Power Management Unit) to turn on the analog power supply for the DAC, as shown in step 1 below.

(Step 2) Set the RCC to enable the clock to be used by the DAC, as shown in step 2 in the following figure.

(Step 3) Select the reference potential source, there are AVDD, B-GAP 1.2V, External Pin input, as shown in step 3 below.

(Step 4) Set the parameters of the DAC module, set the conversion channel and speed, as shown in step 4 below.

(Step 5) Set the GPIO type (the IO is set last to prevent the signal from being poured into the module with undetermined status), as shown in step 5 in the figure below.



7.2 Example program dac

Please refer to the function InitialDac( ) in wt32l0xx_pl_dac.c, the following programs are executed in sequence refer to steps 1 to 5 as mentioned above.

```
void InitialDac(uint16_t DAC_channel)
{
 //----------- PMU ----------------------------------------
 // Use the PMU to turn on the working power of the DAC module
 PMU_PowerAnalogCmd(PMU_PowerAnalog_DAC, ENABLE);   // power supply

 //----------- RCC ----------------------------------------
 // Use RCC to open the working frequency of the DAC module and select APB1
 RCC_APB0PeriphClockCmd(RCC_APB0Periph_DAC, ENABLE);  // Supply frequency

 //----------- DAC ----------------------------------------
 DAC_DeInit();              // Clear old DAC settings first

#if(DAC_VREF_SEL_1P2==ON)
 DAC->CFG &= ~BIT2;             // Use BG1P0V as reference power

#elif(DAC_VREF_SEL_EXT==ON)
 DAC->CFG &= ~BIT3;             // Use external IO as reference power

 status = INW(0x4008c100);
 status = (status | BIT1 | BIT0);                //PB0 analog mode, Ext. Channel
 OUTW(0x4008c100, status);

#elif(DAC_VREF_SEL_AVDD==ON)
 DAC->CFG &= ~(BIT3 | BIT2); // Use AVDD as reference power
#endif

 DAC_Rail2RailAmpCmd(ENABLE);                    // DAC to Rail to Rail Amplifier INP
 DAC_Rail2RailBypassCmd(DISABLE);  // DAC Bypass Rail to Rail amplifier
 DAC_Cmd(ENABLE);                            // Turn on output, Rail to Rail Amplifier

 //----------    IO Setting ----------------------------------------
 GPIO_InitTypeDef          GPIO_InitStructure;
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
 GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

1. (PMU block) 2. (RCC block) 3. (reference power block) 4. (DAC amplifier/output block) 5. (IO Setting block)

7.3 DAC data conversion output

This line function is the write scratchpad command:
DAC->CVTD = Data;

The sample program is as follows.

```
uint32_t DAC_Convert(uint16_t DAC_channel,uint32_t u32DacOut)
{
 DAC_SetInputData(u32DacOut);        //analog output signal DAC->CVTD

 return          u32DacOut;
}
```

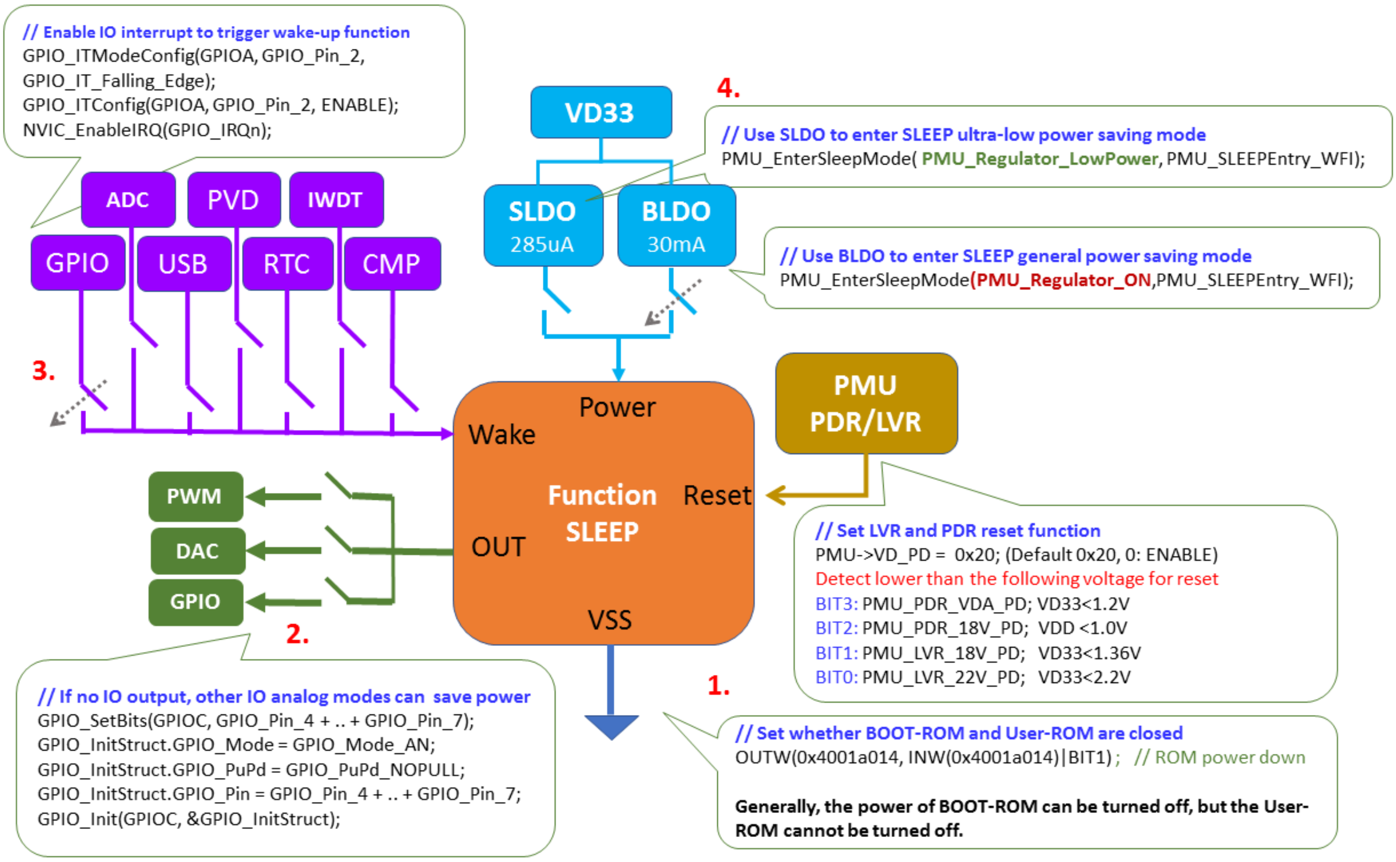
8. SLEEP function description

Use the following illustrations to enter the power saving mode using SLEEP. The action flow is as follows.

8.1 MCU performs SLEEP initialization

The contents of initialization after power-on are as follows, you can refer to the peripheral library wt32l0xx_pl_save.c to use the function save( ).

(Step 1) Set the Boot-ROM power off, and do not use the ISP function when entering SLEEP, as shown in step 1 below.
(Step 2) Set the GPIO type, and set it to the analog mode without using IO, as shown in step 2 in the following figure.
(Step 3) Set GPIO wakeup, all GPIOs can be set to trigger wakeup SLEEP, as shown in step 3 below.
(Step 4) Enter SLEEP mode, you can choose low power consumption and general power saving according to the power consumption, as shown in step 4 in the following figure.



8.2 Sample program save.c

Referring to the function save() of wt32l0xx_pl_save.c, the following programs are executed in sequence refer to the steps 1~4 as mentioned above.

```
void Save(uint16_t nMode)
{
    // -----------    ROM Power -------------------------
1.  OUTW(0x4001a014, INW(0x4001a014) | BIT1);                    // ROM power down

//---------------- Close IO Pull-up   ------------------------
#if(ENABLE_LED_BLINK==ON)
    GPIO_InitTypeDef    GPIO_InitStruct;
2.  GPIO_SetBits(GPIOC, GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7);       // Turn off the LED
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                      // Use analog mode to save power
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;          // Using pull-up in input mode will increase power
consumption
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;        //PIN select
    GPIO_Init(GPIOC, &GPIO_InitStruct);                                   // Perform IO initialization
#endif

//------------------        WAKE UP    Enable------------------------
#if((ENABLE_WAKEUP_CMP==ON)&&(ENABLE_FUNC_CMP==ON))
    NVIC_EnableIRQ(CMP0_VOUT_IRQn);      // COMP interrupt enable
#endif

#if((ENABLE_FUNC_GPIO==ON)&&(ENABLE_WAKE_GPIO==ON)&&(ENABLE_STANDBY_MODE==OFF) )
#if(STOP_WAKEUP_PA2==ON)   // Use PA2 for wake-up IO use
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
3.  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
    GPIO_ITModeConfig(GPIOA, GPIO_Pin_2, GPIO_IT_Falling_Edge); // GPIO interrupt
    GPIO_ITConfig(GPIOA, GPIO_Pin_2, ENABLE);
    NVIC_EnableIRQ(GPIO_IRQn);      // GPIO interrupt enable
#endif
#if(STOP_WAKEUP_PC9==ON)      // Use PC9 for wake-up IO usage
//......omit
#endif
#endif

#if((ENABLE_FUNC_RTC==ON)&&(ENABLE_WAKEUP_RTC==ON))
//......omit
#endif

#if(ENABLE_WAKEUP_IWDT==ON)              // IWDT is still on when power saving
    PMU_StopModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    PMU_StandbyModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    IWDT_ReloadCounter();
#endif

    //------------------ Sleep ------------------------------------
#if(ENABLE_SLEEP_MODE==ON)              // systick must be off or it will wake up
if (nMode == SAVE_MODE_SLEEP)
```

```
{

#if(ENABLE_FUNC_SYSTICK==ON)
        SysTick->CTRL = 0;
#endif

//entry SLEEP mode
  PMU_EnterSleepMode(PMU_Regulator_ON,PMU_SLEEPEntry_WFI);                    // BLDO=ON
  //PMU_EnterSleepMode(PMU_Regulator_ON,PMU_SLEEPEntry_WFE);                       // BLDO=ON
  //PMU_EnterSleepMode(PMU_Regulator_LowPower, PMU_SLEEPEntry_WFI);                // BLDO=OFF, canot run
HSI
  //PMU_EnterSleepMode(PMU_Regulator_LowPower,PMU_SLEEPEntry_WFE);                 // BLDO=OFF, canot run
HSI
}
#endif
```

4.

9. STOP function description

Use the following illustrations to enter the power saving mode using STOP. The action flow is as follows.

9.1 MCU performs STOP initialization

The contents of initialization after power-on are as follows, you can refer to the peripheral library wt32l0xx_pl_save.c to use the function save( ).

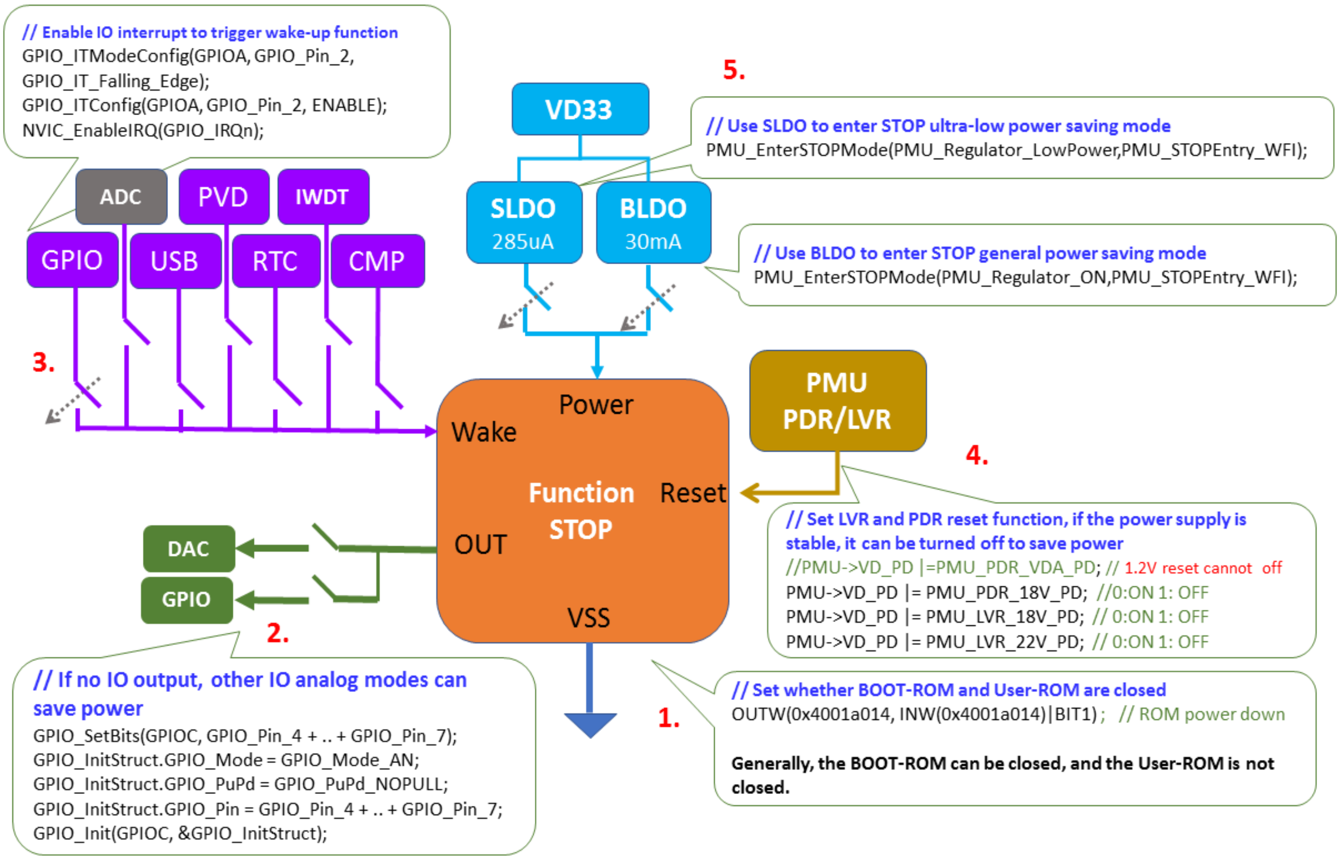
(Step 1) Set the Boot-ROM power off, and do not use the ISP function when entering STOP, as shown in step 1 in the following figure.

(Step 2) Set the GPIO type, and set it to the analog mode without using IO, as shown in step 2 in the following figure.

(Step 3) Set GPIO wake-up, all GPIO can be set to trigger wake-up STOP, as shown in step 3 below.

(Step 4) Set the PDR/LVR reset, if the power supply is stable, the LVR can be turned off to save power, the PDR is recommended to be turned on, as shown in step 4 below.

(Step 5) Enter STOP mode, you can choose low power consumption and general power saving according to the power consumption, as shown in step 5 in the figure below.



9.2 Sample program save

Referring to the function save() of wt32l0xx_pl_save.c, the following programs are executed in sequence refer to steps 1~5 as mentioned above.

```
void Save(uint16_t nMode)
{
    // -----------   ROM Power -------------------------
1.  OUTW(0x4001a014, INW(0x4001a014) | BIT1);                 // ROM power down


//---------------- Close IO Pull-up     ------------------------
#if(ENABLE_LED_BLINK==ON)
    GPIO_InitTypeDef    GPIO_InitStruct;
2.  GPIO_SetBits(GPIOC, GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7);        // Turn off the LED
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                  // Use analog mode to save power
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;          // Using pull-up in input mode will increase power
consumption
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;        //PIN select
    GPIO_Init(GPIOC, &GPIO_InitStruct);                                    // Perform IO initialization
#endif

//------------------ WAKE UP    Enable-----------------------
#if((ENABLE_WAKEUP_CMP==ON)&&(ENABLE_FUNC_CMP==ON))
    NVIC_EnableIRQ(CMP0_VOUT_IRQn);       // COMP interrupt enable
#endif

#if((ENABLE_FUNC_GPIO==ON)&&(ENABLE_WAKE_GPIO==ON)&&(ENABLE_STANDBY_MODE==OFF) )
#if(STOP_WAKEUP_PA2==ON)      // Use PA2 for wake-up IO use
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
3.  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
    GPIO_ITModeConfig(GPIOA, GPIO_Pin_2, GPIO_IT_Falling_Edge); // GPIO interrupt
    GPIO_ITConfig(GPIOA, GPIO_Pin_2, ENABLE);
    NVIC_EnableIRQ(GPIO_IRQn);      // GPIO interrupt enable
#endif
#if(STOP_WAKEUP_PC9==ON)        // Use PC9 for wake-up IO usage
//......omit
#endif
#endif

#if((ENABLE_FUNC_RTC==ON)&&(ENABLE_WAKEUP_RTC==ON))
//......omit
#endif

#if(ENABLE_WAKEUP_IWDT==ON)               // IWDT is still on when power saving
    PMU_StopModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    PMU_StandbyModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    IWDT_ReloadCounter();
#endif
```

```
//------------------ Sleep ------------------------------------
#if(ENABLE_SLEEP_MODE==ON)
    //......omit
#endif


//----------------- STOP -------------------------------------
    #if(ENABLE_STOP_MODE==ON)
    if (nMode == SAVE_MODE_STOP)
    {
        //PMU->VD_PD |=PMU_PDR_VDA_PD;   //PDR_VDA OFF
4.      PMU->VD_PD |= PMU_PDR_18V_PD;   //PDR_18V OFF
        PMU->VD_PD |= PMU_LVR_18V_PD;   //LVR_18V OFF
        PMU->VD_PD |= PMU_LVR_22V_PD;   //LVR_22V OFF

        PMU->ATPD_STOP |= 0x00000080U;//PMU_StopModeAutoPower_LVR22;   //OFF
        PMU->ATPD_STOP |= 0x00000040U;//PMU_StopModeAutoPower_LVR18;   //OFF
        //PMU->ATPD_STOP&=~0x00000001U;   //LSI; //ON

    // Select the Power-ON state in STOP mode
    #if(ENABLE_FUNC_DAC==ON)
        PMU->ATPD_STOP &= (~PMU_STOP_R2R_PD);
        PMU->ATPD_STOP &= (~PMU_StopModeAutoPower_DAC); // Automatically turn off the power
consumption of the DAC module
    #endif

    #if(ENABLE_FUNC_ADC==ON)
        PMU->ATPD_STOP &= (~PMU_StopModeAutoPower_ADC); // Automatically shut down ADC module
power consumption
    #endif

    #if(ENABLE_FUNC_LSI==ON)
        PMU->ATPD_STOP &= (~PMU_STOP_LSI_PD);  // Automatically turn off LSI module power consumption
    #endif

    #if(ENABLE_WAKEUP_CMP==ON)
        NVIC_EnableIRQ(CMP0_VOUT_IRQn);      // COMP interrupt enable
    #elif(ENABLE_FUNC_CMP==ON)
    PMU->ATPD_STOP &= (~PMU_StopModeAutoPower_CMP);      // Automatically turn off the power
consumption of the COMP module
    #endif

// enter STOP mode
    //PMU_EnterSTOPMode(PMU_Regulator_ON,PMU_STOPEntry_WFI);    //BLDO=ON
5.
    PMU_EnterTOPMode(PMU_Regulator_LowPower,PMU_STOPEntry_WFI);    //BLDO=OFF


    }
#endif
```

10. STANDBY function description

Use the following illustrations to enter the power saving mode using STANDBY. The action flow is as follows.

10.1 MCU performs STANDBY initialization

The contents of initialization after power-on are as follows, you can refer to the peripheral library wt32l0xx_pl_save.c to use the function save( ).

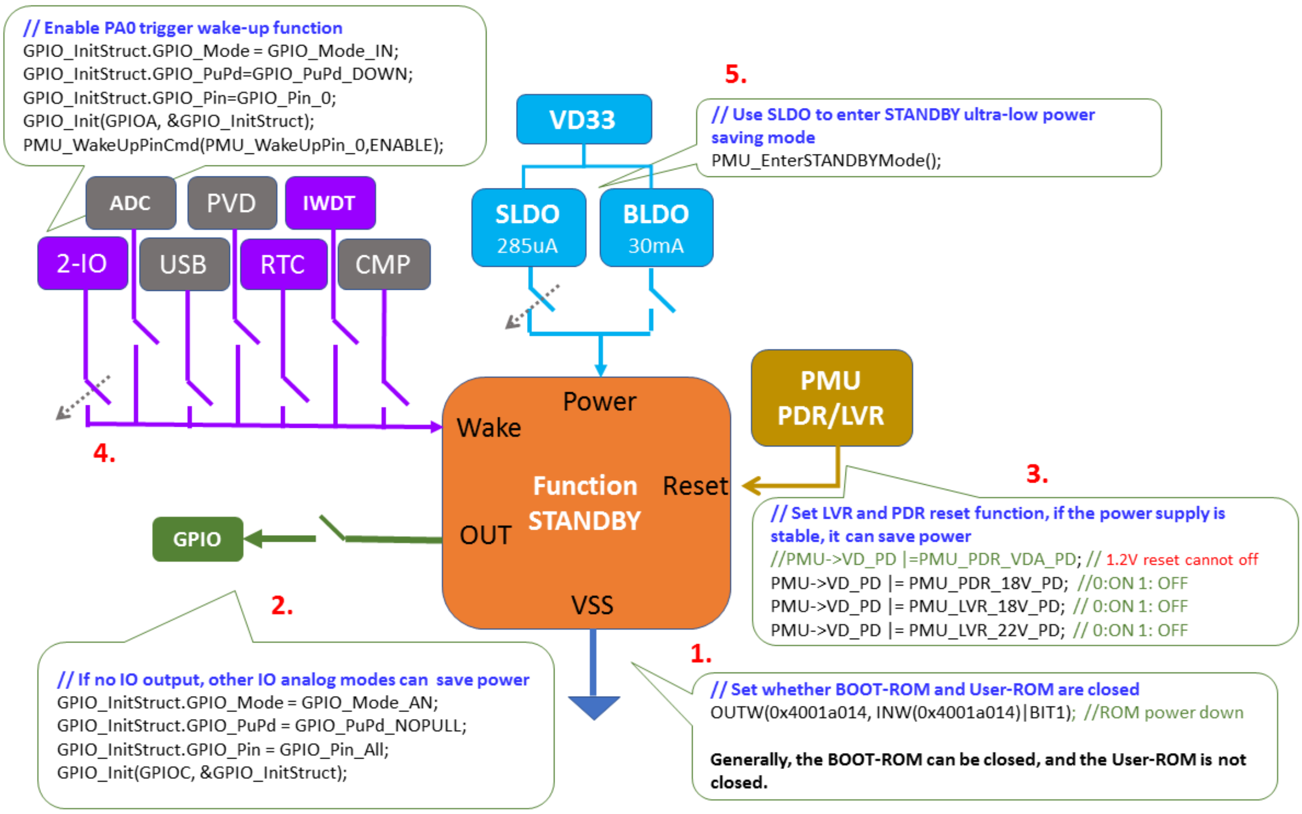
(Step 1) Set the Boot-ROM power off, and do not use the ISP function when entering STANDBY, as shown in step 1 as below.

(Step 2) Set the GPIO type, and set it to the analog mode without using IO, as shown in step 2 in the following figure.

(Step 3) Set the GPIO wake-up, there are two sets of GPIOs that can be set to trigger the wake-up STANDBY, as shown in step 3 as below.

(Step 4) Set the PDR/LVR reset, if the power supply is stable, the LVR can be turned off to save power, the PDR is recommended to be turned on, as shown in step 4 below.

(Step 5) Enter STANDBY mode, you can choose low power consumption and general power saving according to the power consumption, as shown in step 5 in the following figure.



10.2 Example Program Save

Please refer to the function save() of wt32l0xx_pl_save.c, the following programs are executed in sequence with reference to the above steps 1~5.

```
void Save(uint16_t nMode)
{
      // -----------    ROM Power -------------------------
1.    OUTW(0x4001a014, INW(0x4001a014) | BIT1);                // ROM power down


      //---------------- Close IO Pull-up        ------------------------
      #if(ENABLE_LED_BLINK==ON)
            GPIO_InitTypeDef    GPIO_InitStruct;
2.          GPIO_SetBits(GPIOC, GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7); // Turn off the LED
            GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                     // Use analog mode to save power
            GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;          // Using pull-up in input mode will increase
      power consumption
            GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7; //PIN select
            GPIO_Init(GPIOC, &GPIO_InitStruct);                                    // Perform IO initialization
      #endif

      //------------------    WAKE UP    Enable------------------------
      #if((ENABLE_WAKEUP_CMP==ON)&&(ENABLE_FUNC_CMP==ON))
            NVIC_EnableIRQ(CMP0_VOUT_IRQn);      // COMP interrupt enable
      #endif

      #if((ENABLE_FUNC_GPIO==ON)&&(ENABLE_WAKE_GPIO==ON)&&(ENABLE_STANDBY_MODE==OFF) )
         #if(STOP_WAKEUP_PA2==ON)            // Use PA2 for wake-up IO use
            GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
            GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
            GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
            GPIO_Init(GPIOA, &GPIO_InitStruct);
            GPIO_ITModeConfig(GPIOA, GPIO_Pin_2, GPIO_IT_Falling_Edge); // GPIO interrupt
            GPIO_ITConfig(GPIOA, GPIO_Pin_2, ENABLE);
            NVIC_EnableIRQ(GPIO_IRQn);      // GPIO interrupt enable
         #endif
         #if(STOP_WAKEUP_PC9==ON)    // Use PC9 for wake-up IO usage
         //……omit
         #endif
      #endif

      #if((ENABLE_FUNC_RTC==ON)&&(ENABLE_WAKEUP_RTC==ON))
      //……omit
      #endif

      #if(ENABLE_WAKEUP_IWDT==ON)            // IWDT is still on when power saving
            PMU_StopModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
            PMU_StandbyModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
            IWDT_ReloadCounter();
      #endif

      //------------------ Sleep ------------------------------------
      #if(ENABLE_SLEEP_MODE==ON)
```

```
        //……omit
    #endif

    //---------------- STOP -----------------------------------
    #if(ENABLE_STOP_MODE==ON)
        //……omit
    #endif

    //---------------- STANDBY -----------------------------------
    #if(ENABLE_STANDBY_MODE==ON)
    if(nMode==SAVE_MODE_STANDBY)
    {
3.      // -----------    Reset Power ------------------------
        //PMU->VD_PD |=PMU_PDR_VDA_PD;   //PDR_VDA OFF
        PMU->VD_PD |=PMU_PDR_18V_PD;     //PDR_18V OFF
        PMU->VD_PD |=PMU_LVR_18V_PD;     //LVR_18V OFF
        PMU->VD_PD |=PMU_LVR_22V_PD;     //LVR_22V OFF


        //PMU->ATPD_STBY |= (uint32_t)0x7FF;  //AUTO Close ALL,([0]LSI OFF)
        PMU->ATPD_STBY |= (uint32_t)0x7DF;   //[5]PDR-VDA=KEEP      , [6]PDR-V18=AUTO-OFF

        #if(ENABLE_FUNC_GPIO==ON)
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;
4.      GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;   //If pull-up will be lost power!
        GPIO_InitStruct.GPIO_Pin= GPIO_Pin_All ;
        GPIO_Init(GPIOA, &GPIO_InitStruct);
        GPIO_Init(GPIOB, &GPIO_InitStruct);
        GPIO_Init(GPIOC, &GPIO_InitStruct);
        GPIO_Init(GPIOD, &GPIO_InitStruct);
        #endif

        // PA0 & PC13    need set LO , USE External Pull-Up/Dn
        GPIO_InitTypeDef    GPIO_InitStruct;
        GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
        GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_DOWN;

        #if(STANDBY_WAKEUP_PA0==ON) // set PA0 to wakeup
        GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;;
        GPIO_Init(GPIOA, &GPIO_InitStruct);
        PMU_WakeUpPinCmd(PMU_WakeUpPin_0,ENABLE);
        #endif
        #if(STANDBY_WAKEUP_PC13==ON)
        GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13;        // set PC13 to wakeup
        GPIO_Init(GPIOC, &GPIO_InitStruct);
        PMU_WakeUpPinCmd(PMU_WakeUpPin_1,ENABLE);
        #endif

5.      // Enter STANDBY mode
        PMU_EnterSTANDBYMode();

    }
    #endif
```

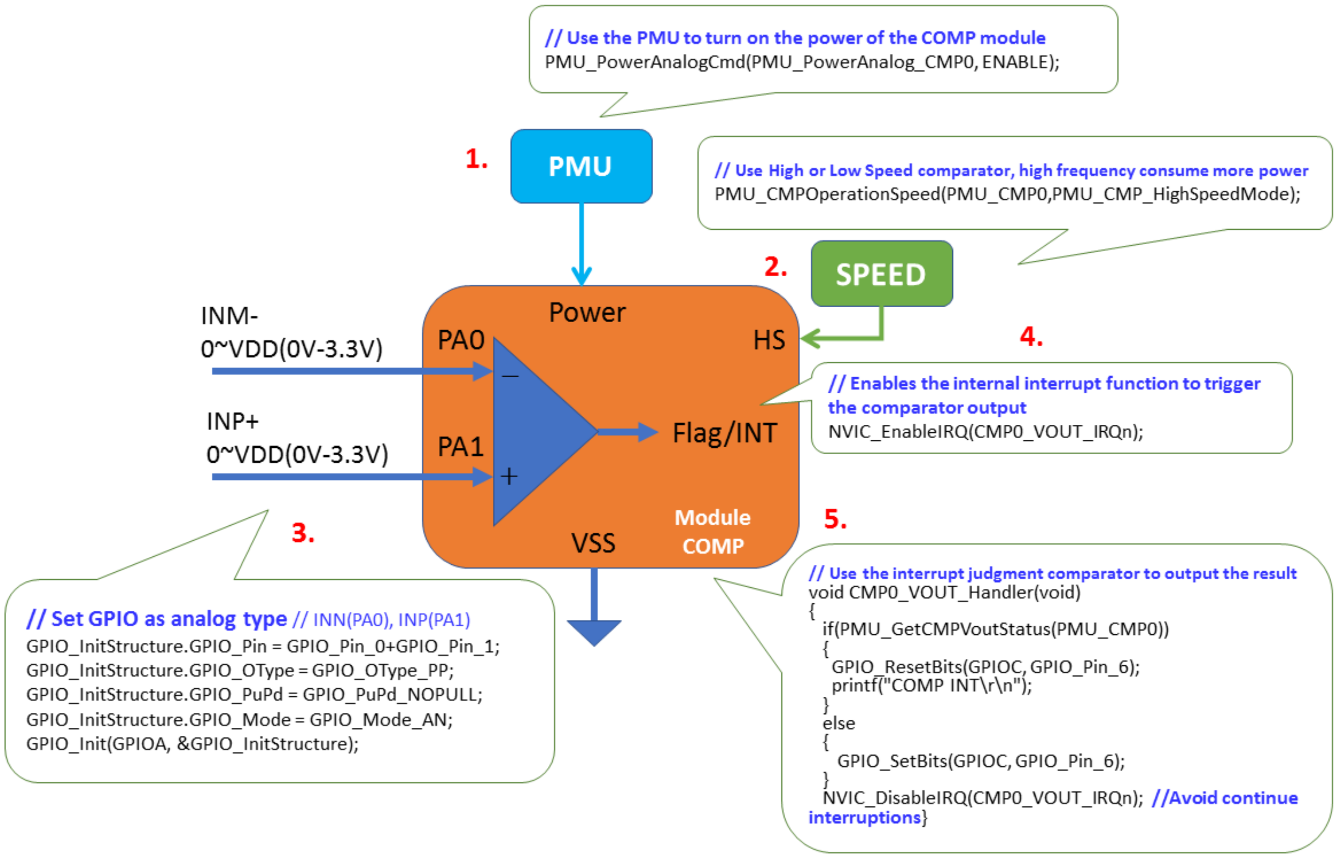
11. COMPARATOR function description

Please refer to the following illustrations, use the comparator (COMP) to perform analog signal input. The action flow is as follows.

11.1 MCU performs Comparator initialization

The contents of initialization after power-on are as follows, you can refer to the peripheral library wt32l0xx_pl_comp.c to use the function InitialComp ( ).

(Step 1) Set the PMU (Power Management Unit) to turn on the analog power supply for the COMP, as shown in step 1 as below.
(Step 2) Set the RCC to enable the clock to be used by the COMP, as shown in step 2 in the figure below.
(Step 3) Set the GPIO type (the IO is set last to avoid the signal being poured into the module whose status is not determined), as shown in step 3 in the following figure.
(Step 4) Set the interrupt function of the COMP module to trigger when the input potential INP>INM, as shown in step 4 in the following figure.
(Step 5) Trigger an interrupt when INP>INM, and the output result can also be read out with PMU_GetCMPVoutStatus().



11.2 Example program for Comparator

Please refer to the function InitialComp ( ) of wt32l0xx_pl_comp.c, and execute above steps 1~5 in sequence.

```
void InitialComp(void)
{
    GPIO_InitTypeDef    GPIO_InitStructure;

1.  #if(ENABLE_HW_CMP0==ON)        // Enable COMP_0
        // Use the PMU to turn on the working power of the COMP module
        PMU_PowerAnalogCmd(PMU_PowerAnalog_CMP0, ENABLE);

    // Use High or Low Speed to implement the comparator, high bandwidth increases power consumption
2.  #if(ENABLE_HW_CMP_SPEED_HI==ON)
        PMU_CMPOperationSpeed(PMU_CMP0, PMU_CMP_HighSpeedMode);
    #else
        PMU_CMPOperationSpeed(PMU_CMP0, PMU_CMP_LowSpeedMode);
    #endif

        // Set GPIO as analog type Analog function // INN(PA0), INP(PA1)
3.      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 + GPIO_Pin_1;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
4.
        NVIC_EnableIRQ(CMP0_VOUT_IRQn);      // COMP interrupt enable
    #endif

    #if(ENABLE_HW_CMP1==ON)        //open COMP_1
    //......omit
    #endif
}
```

11.3 Interrupt function of Comparator

The interrupt function CMP0_VOUT_Handler ( ) of the sample program comp.c can be compared with the e step as mentioned above.

```
void CMP0_VOUT_Handler(void)
{
5.      if (PMU_GetCMPVoutStatus(PMU_CMP0))
        {
            GPIO_ResetBits(GPIOC, GPIO_Pin_6);
            printf("COMP INT\r\n");
        }
        else
        {
            GPIO_SetBits(GPIOC, GPIO_Pin_6);
        }

        NVIC_DisableIRQ(CMP0_VOUT_IRQn);     // COMP INT disable，Avoid continue interruptions

}
```

After entering the interrupt, read the comparator result

12. FLASH read and write function description

Refer to the following diagrams, use the IC's internal FLASH to read and write data. A complete write and read operation process is as follows.

12.1 MCU performs FLASH initialization

To update the FLASH data after power-on, the data must be erased to 0xFF before the content can be written. You can refer to the peripheral library to use the function RunFlash ( ).

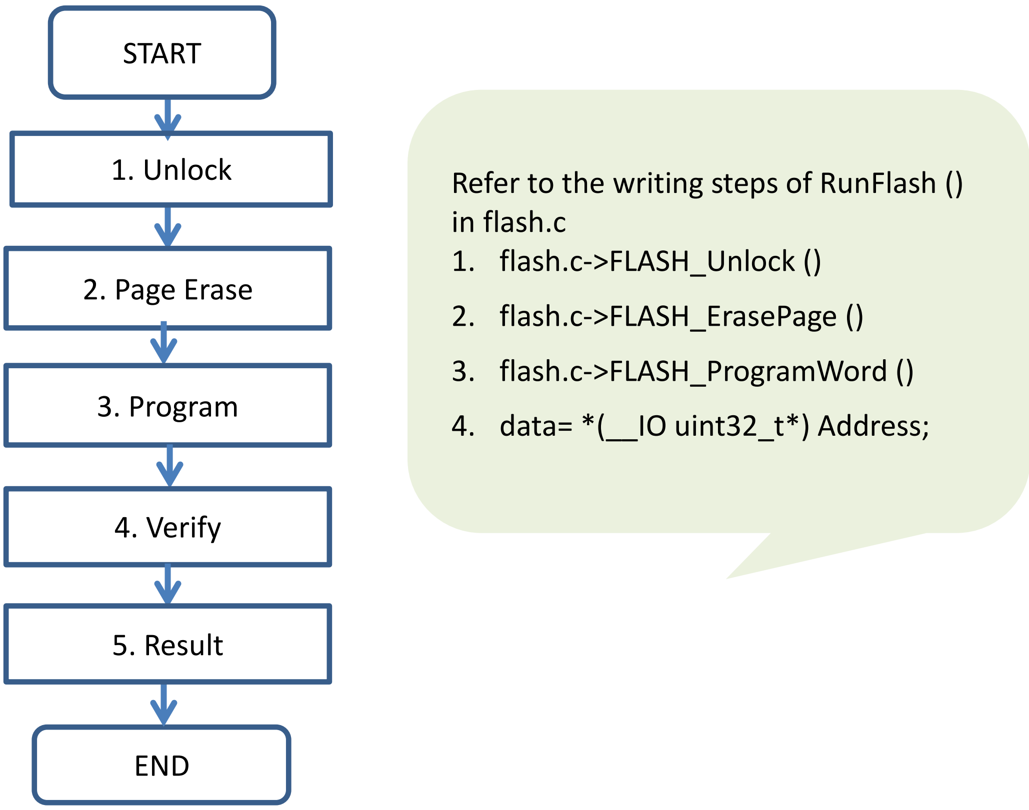
(Step 1) Unlock the FLASH protection lock, as shown in step 1 as below.

(Step 2) Set the address to be written, and first clear the page data size to 1KB, as shown in step 2 in the following figure.

(Step 3) Write WORD data, use CMSIS to call FLASH_ProgramWord(), this function uses __IO to address the ROM space, for example: *(__IO uint32_t *)Address = Data, as shown in step 3 below.

(Step 4) Check WORD data and directly use __IO to address ROM space, for example: Data= *(__IO uint32_t *)Address; as shown in step 4 below.

(Step 5) Use UART to output the result, as shown in step 5 below.



12.2 Sample Program Flash

Please refer to the function RunFlash ( ) of wt32l0xx_pl_flash.c, and execute above steps 1~5 in sequence.

```
void RunFlash(void)
{

#if(SYS_CLOCK_SEL!=CLK_MSI)
        FLASH_SetLatency(1);   // If system frequency >=16 Mhz     // Set latency
#endif
        FLASH_ClearFlag(FLASH_FLAG_EOP | /*FLASH_FLAG_PGERR |*/ FLASH_FLAG_WRPERR);
1.      //==================== Unlock    FLASH     ===============================//
        FLASH_Unlock();  // Release FLASH anti-write lock
                                                                1.  Unlock

        /* Define the number of page to be erased */
        TotalPages = (WRITE_END_ADDR - WRITE_START_ADDR + 1) / FLASH_PAGE_SIZE;
                                                                2. Page Erase
2.      //==================== Erase    FLASH     ===============================//
        for (EraseCounter = 0; (EraseCounter < TotalPages) && (FLASHStatus == FLASH_COMPLETE); EraseCounter++)
        {
        FLASHStatus = FLASH_ErasePage(WRITE_START_ADDR + (FLASH_PAGE_SIZE * EraseCounter));//Page erase
                if (FLASHStatus != FLASH_COMPLETE) // If the clear fails, output the value and terminate
                {
                        uint16_t readout = *(__IO uint16_t*)(WRITE_START_ADDR + (FLASH_PAGE_SIZE * EraseCounter));
                        printf("Page=0x%d,", START_ADDR_PAGE + EraseCounter); // read value and display
                        printf("Data=0x%x\r\n", readout);
                        break;
                }

        }
        if (FLASHStatus == FLASH_COMPLETE)      printf("Erase Done\r\n");
        else                                    printf("Erase Fail,Page=%d\r\n", START_ADDR_PAGE + EraseCounter);


        //==================== Program    FLASH     ===============================//
        uint32_t u32TargetStartAddr = 0;
        uint32_t u32TargetEndAddr = FLASH_PAGE_SIZE - 1;


        uint32_t Page = START_ADDR_PAGE, pos, PageCnt = 0;;
        while (((u32TargetEndAddr + WRITE_START_ADDR) <= WRITE_END_ADDR) && (FLASHStatus ==
FLASH_COMPLETE))
        {

                // Clear All pending flags
                FLASH_ClearFlag(FLASH_FLAG_EOP | /*FLASH_FLAG_PGERR |*/ FLASH_FLAG_WRPERR);

                //--------------- Program Flash Page--------------
3.              Address = WRITE_START_ADDR + u32TargetStartAddr;              3. Program
                //for(int i=0;i<(512);i++)          //512*32bit=2KB
                for (int i = 0; i < (FLASH_PAGE_SIZE / 4); i++)          //256*32bit=1KB
```

```
        {
            FLASHStatus = FLASH_ProgramWord(Address + 4 * i, i + Page);// Write WORD data
        }

        u32TargetStartAddr += FLASH_PAGE_SIZE;
        u32TargetEndAddr += FLASH_PAGE_SIZE;
        Page++;                 // page absolute address
        PageCnt++; //page count
    }
    if (FLASHStatus == FLASH_COMPLETE)          printf("Program Done\r\n");
    else                                        printf("Program Fail, Page=%d\r\n", Page - 1);

    //--------------- Test Lock ---------
    //......omit
```

4.

```
    //=================== Verify    FLASH    ===============================//
    u32TargetStartAddr = 0;
    u32TargetEndAddr = FLASH_PAGE_SIZE - 1;
```

4. Verify

```
    Page = START_ADDR_PAGE, PageCnt = 0;;
    while (((u32TargetEndAddr + WRITE_START_ADDR) <= WRITE_END_ADDR) && (FLASHStatus ==
FLASH_COMPLETE))
    {
        //--------------- Check Data ----------------------
        Address = WRITE_START_ADDR + u32TargetStartAddr;
        for (pos = 0; pos < 512; pos++)        // data: WORD
        {
            int readout = *(__IO uint32_t*) Address + pos;
            if (readout != (pos + Page))            // Check whether the value written before is correct?
            {
                MemoryProgramStatus = FAILED;
                printf("Page=%d,", Page);
                //......omit
                while (1);
            }
        }

        printf("Page=%d,", Page);
        printf("Offset=0x%x OK!\r\n", u32TargetStartAddr);

        u32TargetStartAddr += FLASH_PAGE_SIZE;
        u32TargetEndAddr += FLASH_PAGE_SIZE;
        Page++;                 //page absolute addres
        PageCnt++;    //page count
    }
```

5.

```
    if (FLASHStatus == FLASH_COMPLETE)
        printf("Total Page=%d, PASS!\r\n", PageCnt);
    else
        printf("Verify Fail\r\n");
```

5. Result

```
    while (1);    //End and stop here
}
```

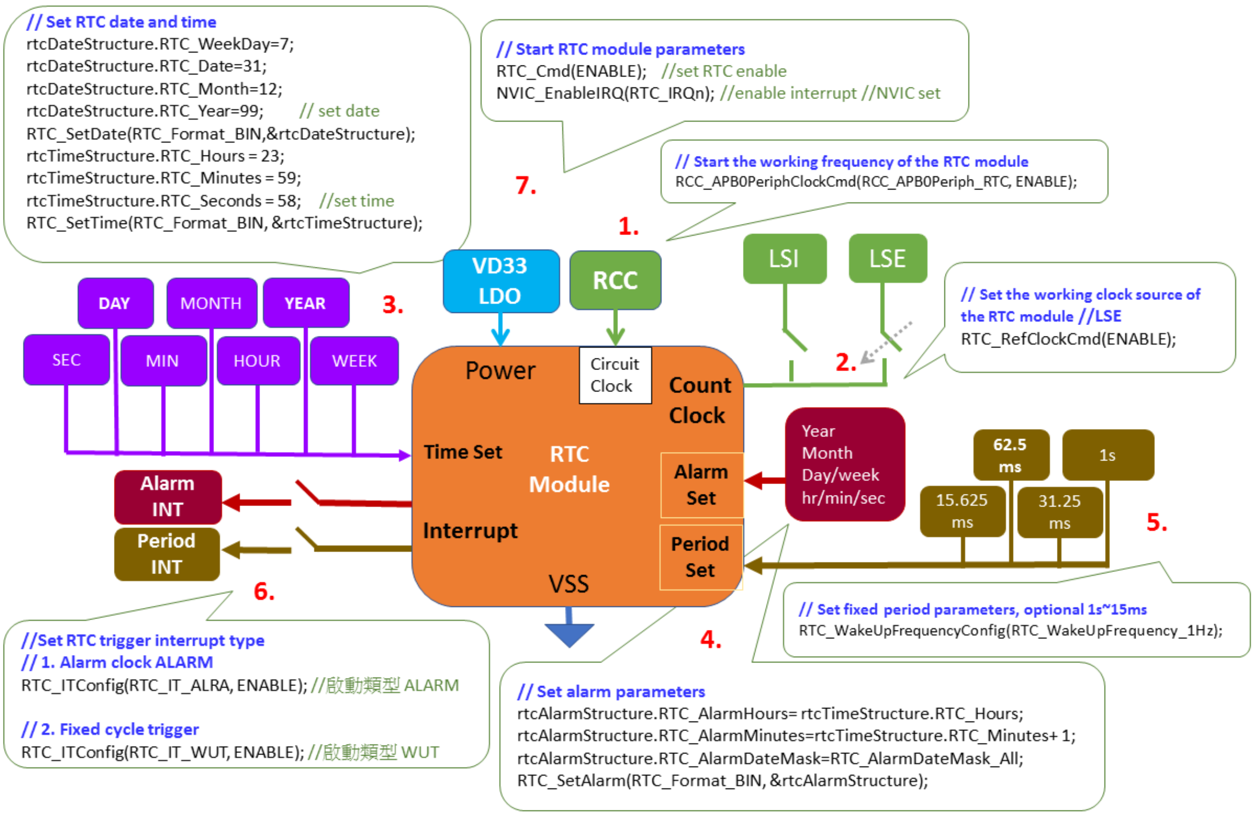
13. RTC function description

Refer to the following illustration to perform digital signal input using a real-time counter (RTC).

13.1 MCU performs RTC initialization

The contents of initialization after power-on are as follows, you can refer to the peripheral library wt32l0xx_pl_rtc.c to use the function InitialRtc ( ).

(Step 1) Set the RCC to enable the clock to be provided to the RTC, as shown in step 1 in the following figure.
(Step 2) Select the reference clock source LSI (37kHz) or LSE (32.768kHz), as shown in step 2 below.
(Step 3) Set the current date and time of RTC, as shown in step 3 below.
(Step 4) Set the alarm time to trigger the Alarm interrupt when it is the same as the current time, as shown in step 4 below.
(Step 5) Set the cycle time to trigger the WUT interrupt, there are 1sec~15msec options, as shown in step 5 below.
(Step 6) Set the interrupt switch, there are two types of interrupts, Alarm and Period, as shown in step 6 below.
(Step 7) Start the RTC function and turn on the NVIC interrupt master switch, as shown in step 7 below.



13.2 Example program rtc

Refer to the function InitialRtc ( ) of wt32l0xx_pl_rtc.c, and execute steps 1~6 as mentioned above in sequence.

```
void InitialRtc(void)
{
1.    RCC_APB0PeriphClockCmd(RCC_APB0Periph_RTC, ENABLE); //APB0 clock needs to be turned on before RTC setting
      RTC_WriteReadProtectionCmd(DISABLE);   //RTC protection switch, close before changing the setting
      RTC_DeInit();                                         // Clear RTC settings
2.    RTC_RefClockCmd(ENABLE);                     // Reference external clock source LSE: 32.768kHz

      rtcDateStructure.RTC_WeekDay = 7;
      rtcDateStructure.RTC_Date = 31;
      rtcDateStructure.RTC_Month = 12;
      rtcDateStructure.RTC_Year = 99;
      RTC_SetDate(RTC_Format_BIN, &rtcDateStructure);// Set date in RTC module

      rtcTimeStructure.RTC_Hours = 23;
      rtcTimeStructure.RTC_Minutes = 59;
      rtcTimeStructure.RTC_Seconds = 58;
3.    RTC_SetTime(RTC_Format_BIN, &rtcTimeStructure);      // Set time on RTC module

      rtcLastTime.RTC_Hours = 0;            // for test recording
      rtcLastTime.RTC_Minutes = 0;  // for test recording
      rtcLastTime.RTC_Seconds = 0;  // for test recording

       //-----------------  RTC ALARM -----------------------
       #if(ENABLE_FUNC_ALARM==ON)
4.          rtcAlarmStructure.RTC_AlarmHours = rtcTimeStructure.RTC_Hours;
            rtcAlarmStructure.RTC_AlarmMinutes = rtcTimeStructure.RTC_Minutes + 1;
            rtcAlarmStructure.RTC_AlarmDateMask = RTC_AlarmDateMask_All;
            RTC_SetAlarm(RTC_Format_BIN, &rtcAlarmStructure);
            RTC_ITConfig(RTC_IT_ALRA, ENABLE);         // Start interrupt subtype ALARM
       #else
5.          RTC_WakeUpFrequencyConfig(RTC_WakeUpFrequency_1Hz);
6.          RTC_ITConfig(RTC_IT_WUT, ENABLE); // Start interrupt subtype WUT, triggered every (seconds/ms) period
       #endif

7.          RTC_Cmd(ENABLE);                                 // Set RTC to start
            NVIC_EnableIRQ(RTC_IRQn);                 // Start interrupt function //NVIC set
  }
```

13.3 Set RTC time

When set time to trigger, the interrupt function RTC_Handler ( ) of the example program rtc.c.

```
void RTC_Handler(void)
{
      RTC_ClearITPendingBit(RTC_IT_ALRA + RTC_IT_WUT);   // clear hardware flags
      RTC_ITConfig(RTC_IT_ALRA, DISABLE); // Turn off interrupts if no interrupts are required
      gbRtcInt = 1;       // variable set 1

//............can be increased by itself
 }
```

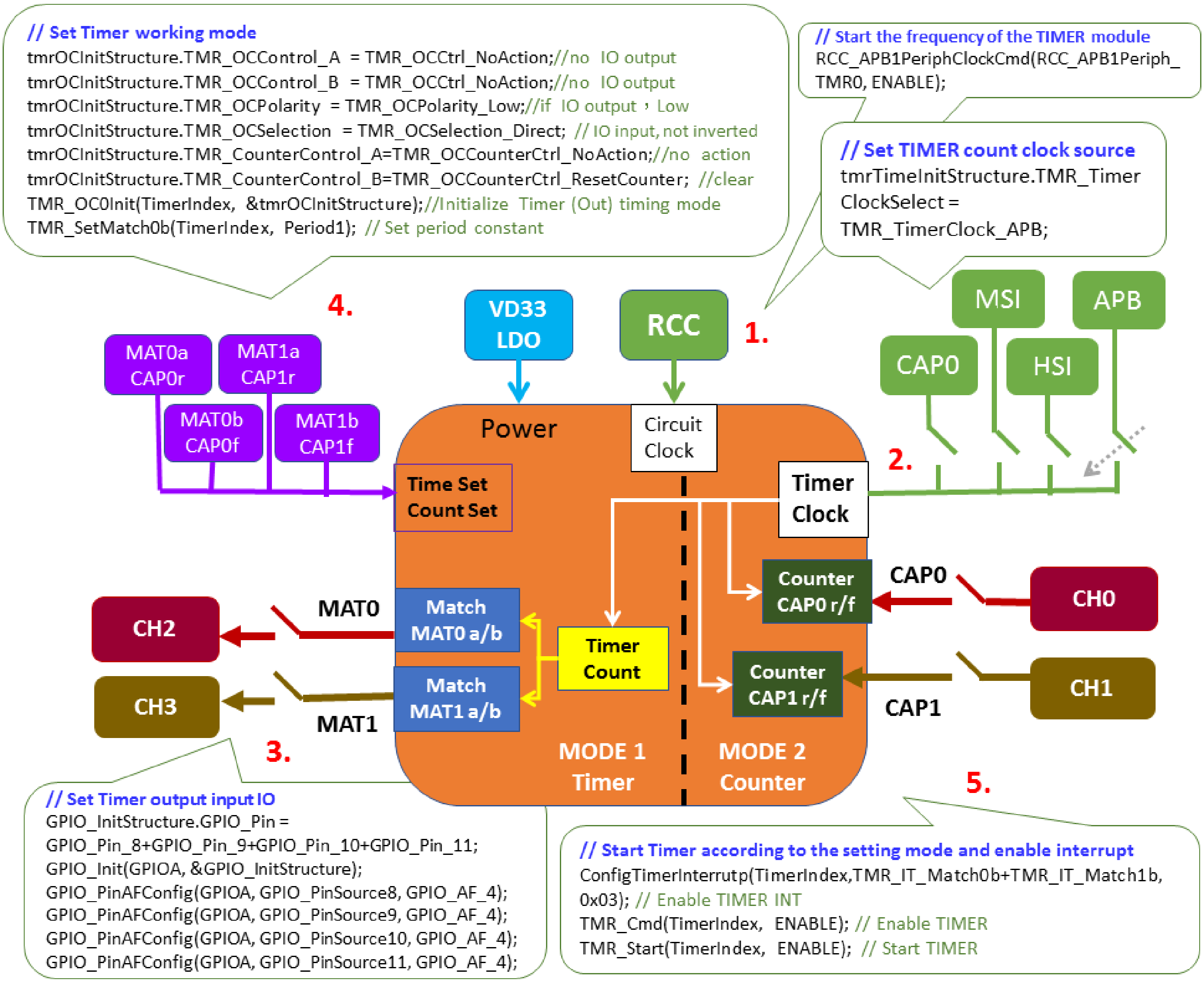
14. TIMER function description

Please refer to the following illustration, use count timer (TIMER) to perform digital signal input and output. The action flow is as follows.

14.1 MCU performs Timer initialization

The contents of initialization after power-on are as follows, you can refer to the peripheral library wt32l0xx_pl_timer.c to use the functions ConfigTimerClockGpio ( ), ConfigTimerTimeMode ( ).

(Step 1) Set the RCC to enable the clock to be used by the Timer circuit, as shown in step 1 in the figure below.
(Step 2) Set the input clock source and provide it for Timer calculation, as shown in step 2 in the following figure.
(Step 3) Set the GPIO type. One group of Timer has two outputs and two inputs, as shown in step 3 in the figure below.
(Step 4) Set the Timer cycle time parameter to trigger the interrupt and output signal, as shown in step 4 in the figure below.
(Step 5) Set the interrupt switch, and start the Timer according to the set parameters, timing or counting mode, as shown in step 5 in the figure below.



14.2 Sample Program Timer

Refer to the functions ConfigTimerClockGpio( ) and ConfigTimerTimeMode( ) in wt32l0xx_pl_timer.c, and execute steps 1~5 as mentioned above in sequence.

```
void ConfigTimerClockGpio(TMR_TypeDef* TimerIndex, uint32_t nPrescaler, uint16_t nChannelSetSel, uint16_t nSource)
{
    TMR_TimerInitTypeDef          tmrTimeInitStructure;
    TMR_DeInit(TimerIndex);       // clear settings
    tmrTimeInitStructure.TMR_TimerClockSelect = nSource; // Frequency source selection APB HSI MSI CAP0
    tmrTimeInitStructure.TMR_TimerPrescaler = nPrescaler;           // frequency division f'=1/ n+1

    //------------ Setting Timer/GPIO --------------------------------
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIO, ENABLE);    // Start GPIO working frequency
    GPIO_InitTypeDef             GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;        // GPIO use AF mode
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

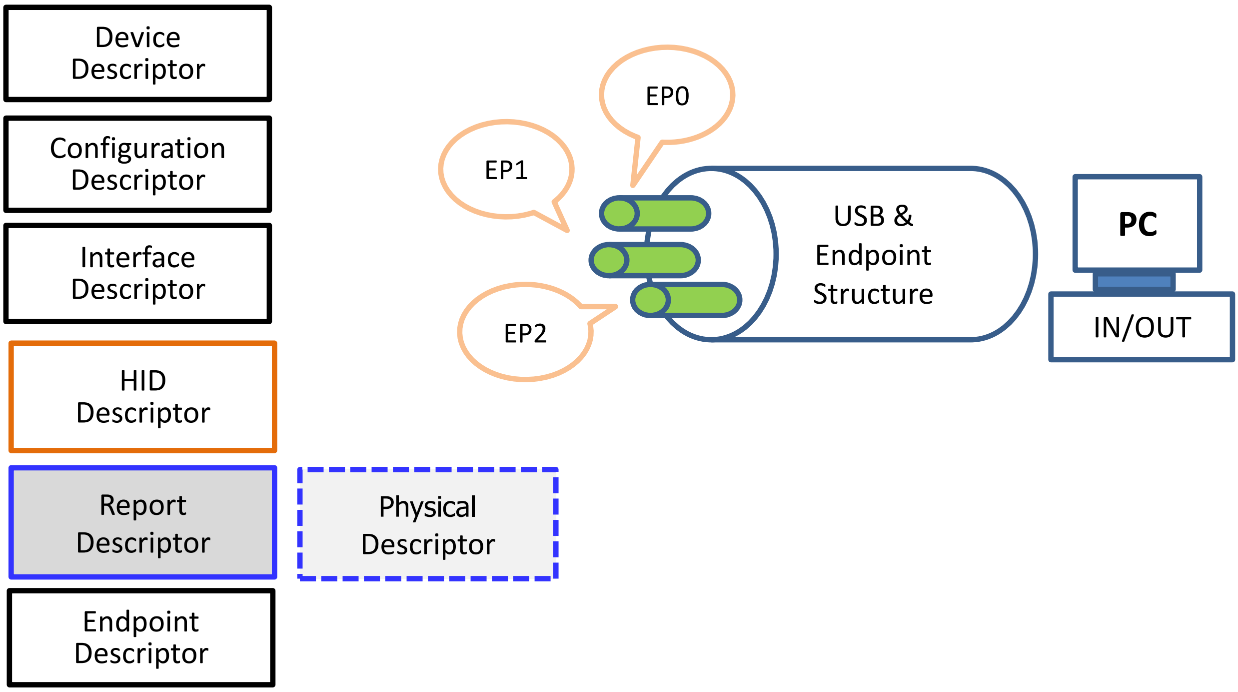
```
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;

        if (TimerIndex == TMR0)
        {
1.              RCC_APB1PeriphClockCmd(RCC_APB1Periph_TMR0, ENABLE);// Start RCC TIMER clock
                tmrTimeInitStructure.TMR_TimerClockSelect = TMR_TimerClock_APB;     //APB only

2.              TMR_TimerInit(TimerIndex, &tmrTimeInitStructure);// Initialize Timer clock source & frequency

                TMR_MatchInputSourceSwap(TMR0, DISABLE);   // don't swap IO

                if (nChannelSetSel == TMR_PIN_SET0)       // Configure IO with Group 1 Channels
                {
3.                      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 + GPIO_Pin_9 + GPIO_Pin_10 + GPIO_Pin_11;
                        GPIO_Init(GPIOA, &GPIO_InitStructure);

                        GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_4);
                        GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_4);
                        GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_4);
                        GPIO_PinAFConfig(GPIOA, GPIO_PinSource11, GPIO_AF_4);
                }
                else if (nChannelSetSel == TMR_PIN_SET1)  // Configure IO with Group 2 Channels
                {
                        //.....omit
                }
        }
        else if (TimerIndex == TMR1)
        {
                //.....omit
        }
        else if (TimerIndex == TMR2)
        {
                //.....omit
        }
        TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_NoFilter);       // No digital filtering is used
        //TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_2clks);  // Use digital filtering 2 clock
        //TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_4clks));// Use digital filtering 4 clock
}

void ConfigTimerTimeMode(TMR_TypeDef* TimerIndex, uint32_t Period1, uint32_t Period2)
{

        TMR_OCInitTypeDef                       tmrOCInitStructure;

        //------------- MATCH 0 --------------------------------------------
4.      tmrOCInitStructure.TMR_OCControl_A = TMR_OCCtrl_NoAction;// Do not do IO output
        tmrOCInitStructure.TMR_OCControl_B = TMR_OCCtrl_NoAction;// Do not do IO output

        tmrOCInitStructure.TMR_OCPolarity = TMR_OCPolarity_Low;  // If IO output, low potential
        tmrOCInitStructure.TMR_OCSelection = TMR_OCSelection_Direct; // IO input, not inverted
        tmrOCInitStructure.TMR_CounterControl_A = TMR_OCCounterCtrl_NoAction;   // No action after Match
        tmrOCInitStructure.TMR_CounterControl_B = TMR_OCCounterCtrl_ResetCounter;    // longest period
        TMR_OC0Init(TimerIndex, &tmrOCInitStructure); // Initialize Timer (Out) timing mode
```

```
    TMR_SetMatch0b(TimerIndex, Period1); // Set period constant


    //------------ MATCH 1 -------------------------------------------
    tmrOCInitStructure.TMR_OCControl_A = TMR_OCCtrl_NoAction;
    tmrOCInitStructure.TMR_OCControl_B = TMR_OCCtrl_NoAction;

    tmrOCInitStructure.TMR_OCPolarity = TMR_OCPolarity_Low;
    tmrOCInitStructure.TMR_OCSelection = TMR_OCSelection_Direct;
    tmrOCInitStructure.TMR_CounterControl_A = TMR_OCCounterCtrl_NoAction;
    tmrOCInitStructure.TMR_CounterControl_B = TMR_OCCounterCtrl_NoAction;    //2th period
    TMR_OC1Init(TimerIndex, &tmrOCInitStructure);

    TMR_SetMatch1b(TimerIndex, Period2);            // Set period constant


    //----------- Interrupt & Enable, use Match0b 、 Match1b ----------------
5.  ConfigTimerInterrutp(TimerIndex, TMR_IT_Match0b + TMR_IT_Match1b, 0x03);

    TMR_Cmd(TimerIndex, ENABLE);
    TMR_Start(TimerIndex, ENABLE);      // Enable TIMER

}
```

15. USB and HID function description

15.1 USB-HID Architecture Description

As shown in the USB Descriptor, the HID descriptor and Report descriptor can be added after the Interface descriptor. There are 3 types of HID report categories: Input, Output and Feature, which can be added as required. The picture shows the standard USB and HID device configuration settings.



In addition to the requirements of special physical devices, the General Physical Descriptor is not used, and the HID communication needs to set the Report Descriptor. The descriptor has the following types:

1. Input: The peripheral device transmits data to the computer, using the GET_REPORT command format
2. Output: The computer transmits data to the peripheral device, using the SET_REPORT command format
3. Feature: Data exchange between peripheral device and computer, using GET_REPORT and SET_REPORT command format

Use the correspondence between USB-Endpoint and HID-Report:

- HID communication using Feature format is to use Feature-Report to perform USB data exchange through EP0 (Endpoint 0), Feature and EP0 are bidirectional data transmission.

- HID communication using Input and Output format is to use Input-Report, Output-Report to choose USB data exchange through EP1~EP6, Report and EP1~EP6 are one-way data transmission, for example, Input-Report selects EP2 (IN single direction), while Output-Report selects EP1 (OUT unidirectional).

15.2 Description of USB-HID Devices and Configuration Descriptors

The sample program provides the following array parameters for users to modify. The parameters that usually need to be modified for self-defined HID transmission are EP0_Packet_Size, VENDOR_ID, PRODUCT_ID, and the number of endpoints. As shown in the red font, the example uses 3 endpoints, namely EP0 as Feature (bidirectional), EP1 as Report-IN, and EP2 as Report-OUT.

```
const unsigned char DEVICE_Descriptor[] = {
    DEVICE_DESCRIPTOR_LENGTH,          //Size of this descriptor in bytes.
    DEVICE_DESCRIPTOR_TYPE,            //Descriptor type.
BCD_USB_VERSION,                       //USB specification release number in
binary-coded-decimal.
    0x00,                              //Class code
    0x00,                              //Subclass code
    0x00,                              //Protocol code
    EP0_Packet_Size,                   //Maximum packet size for endpoint 0
    VENDOR_ID,                         //Vendor ID
    PRODUCT_ID,                        //Product ID
    BCD_DEVICE_NUMBER,                 //Device release number in binary-coded-decimal
 1,                            //Index of string descriptor describing manufacturer
 2,                            //Index of string descriptor describing product
0,                         //Index of string descriptor describing device's serial number
 1                             //Number of possible configurations
};

const unsigned char CONFIGURATION_Descriptor[] = {
//CONFIGURATION(9 bytes)
    CONFIGURATION_DESCRIPTOR_LENGTH,     //Size of this descriptor in bytes.
```

```
    CONFIGURATION_DESCRIPTOR_TYPE,          //Descriptor type.
TOTAL_LENGTH(0x29),         //Total length of byte returned for this configuration.
1,                          //Number of interfaces support by this configuration.
    0x01,                       //Value to use as an argument to the SetConfiguration() …
    0,                          //Index of string descriptor describing this configuration.
0xC0,                       //Configuration characteristics.
    MAX_POWER,                  //Maximum power consumption of the USB device ….

//-------- Interface ----------------------------------------
//INTERFACE(9 bytes)
INTERFACE_DESCRIPTOR_LENGTH,          //Size of this descriptor in bytes.
INTERFACE_DESCRIPTOR_TYPE,            //Descriptor type.
0,                                    //Number of this interface.
0x00,     //Value used to select alternate setting for the interface identified in the prior field.
2,                                    //Number of endpoints used by this interface.
3,                                    //Class code
0,                                    //Subclass code
0,                                    //Protocol code
0,                                  //Index of string descriptor describing this interface.

//HID(9 bytes)
HID_DESCRIPTOR_LENGTH,              //Size of this descriptor in bytes.
HID_DESCRIPTOR_TYPE,                //Descriptor type.
HID_VERSION,                  //HID specification release number in binary-coded-decimal.
0x00,                 //Numeric expression identifying country code of the localized hardware.
1,                        //Numeric expression identifying the number of class descriptor.
HID_REPORT_TYPE,                    //Constant name identifying type of class descriptor.
WORD(HID_ReportDescriptor0Length), //Numeric expression that is total size of the report …

//ENDPOINT(7 bytes)
ENDPOINT_DESCRIPTOR_LENGTH,           //Size of this descriptor in bytes.
```

```
 ENDPOINT_DESCRIPTOR_TYPE,                    //Descriptor type.
 IN_EP1,          //The address of the endpoint on the USB device described by this descriptor.
 INTERRUPT_TRANSFER,                 //This field describes the endpoint's attributes when it is
                                             //configured using the bConfigurationValue.
 WORD(0x21),                    //Maximum packet size this endpoint is capable of sending or
                                     //receiving when this configuration is selected.
 5,                                  //Interval for polling endpoint for data transfers(1ms/unit).

//ENDPOINT(7 bytes)
 ENDPOINT_DESCRIPTOR_LENGTH,                  //Size of this descriptor in bytes.
 ENDPOINT_DESCRIPTOR_TYPE,                    //Descriptor type.
 OUT_EP2,         //The address of the endpoint on the USB device described by this descriptor.
 INTERRUPT_TRANSFER,                 //This field describes the endpoint's attributes when it is
                                     //configured using the bConfigurationValue.
 WORD(0x21),                    //Maximum packet size this endpoint is capable of sending or
                                     //receiving when this configuration is selected.
 5,                                  //Interval for polling endpoint for data transfers(1ms/unit).
};

const unsigned char DeviceHidDescriptor0[]={
 HID_DESCRIPTOR_LENGTH,                       //[00]length of the descriptor
 HID_DESCRIPTOR_TYPE,                         //[01]HID descriptor type
        HID_VERSION,                                  //[02]HID class specification version
        0,                                            //[04]hardware target country
        1,                                            //[05]number of HID class descripters below
 HID_REPORT_TYPE,                             //[06]report descriptor type
 WORD(HID_ReportDescriptor0Length),           //[07]length of report descriptor
};
```

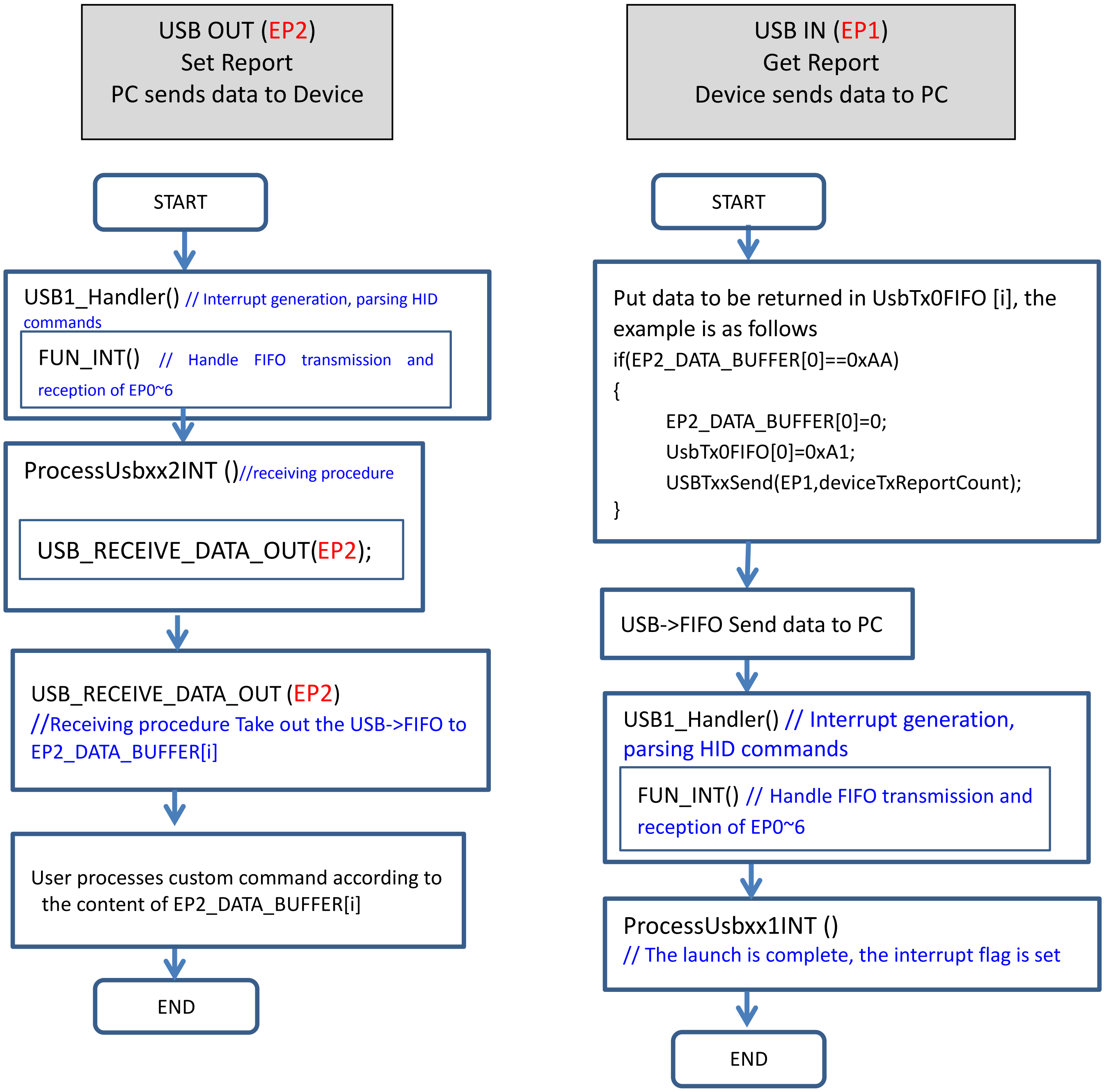
15.3 USB-HID report description element and purpose page description

The content of the report description element includes the USAGE PAGE, which mainly sets the custom transmission format, length and Report ID. Usually, one group of Interface is configured with one group of HID_ReportDescriptor. The parameters that need to be modified are deviceRxReportCount, FEATURE, REPORT OUTPUT and REPORT. INPUT can be added or deleted according to the needs, such as the following parts marked in red font.

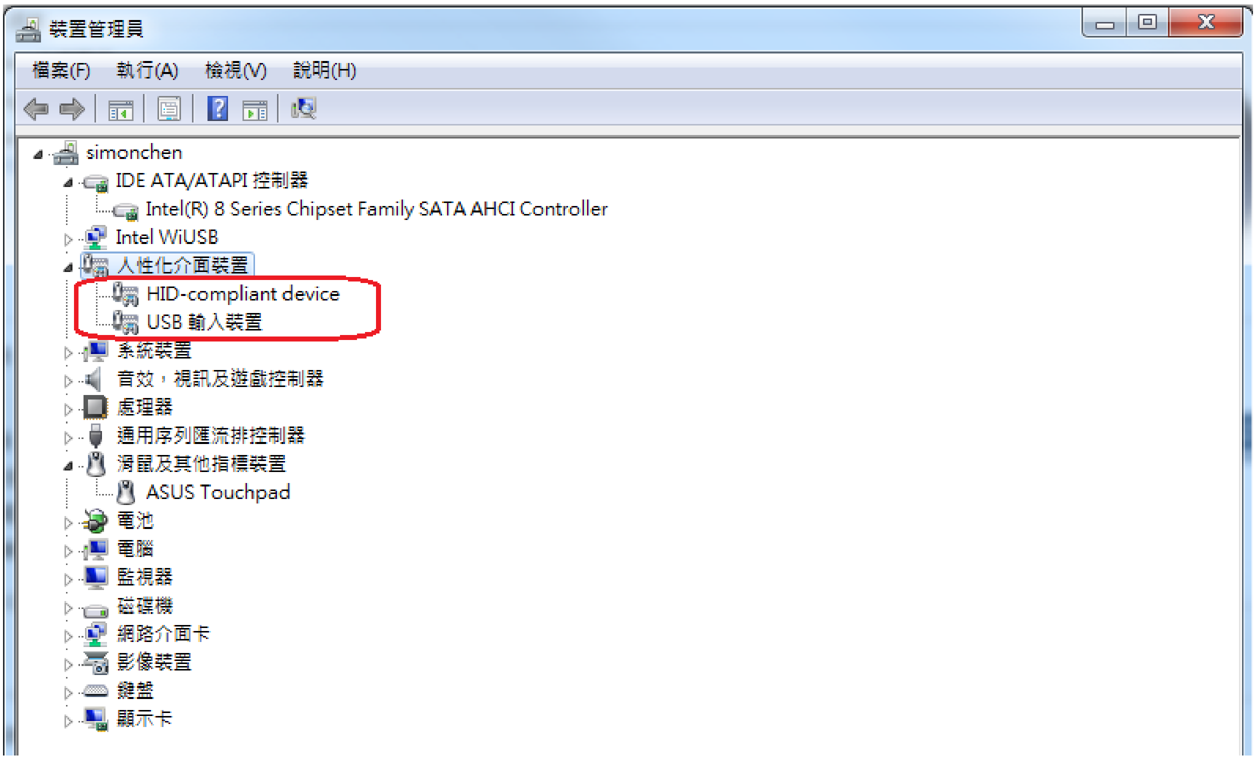
```
const unsigned char    HID_ReportDescriptor0[] = {
 /* USER CODE BEGIN 0 */
 0x06, 0xFF, 0x00,              /* USAGE_PAGE (Vendor Page: 0xFF00) */
 0x09, 0x01,                    /* USAGE (Demo Kit)                  */
 0xa1, 0x01,                    /* COLLECTION (Application)        */
 /* 6 */

 /* Rx_EP */
 0x85, deviceRxReportID,        /* RX_REPORT_ID(0x01)       */
 0x09, 0x01,                    /* USAGE, 0x09/0x?? for vendor-defined */
 0x15, 0x00,                    /* LOGICAL_MINIMUM(0) */
 0x26, 0xff, 0x00,              /* LOGICAL_MAXIMUM(255) */
 0x75, 0x08,                    /* REPORT_SIZE(8), unit of report = 8 bits ( or 16/32 bits) */
 0x95, deviceRxReportCount,     /* REPORT_COUNT(32), 32 bytes per packet, except ID */
 0xB1, 0x82,                    /* FEATURE (Data,Var,Abs,Vol) */

 0x85, deviceRxReportID,        /* RX_REPORT_ID(0x01) */
 0x09, 0x01,                    /* USAGE, 0x09/0x?? for vendor-defined */
 0x91, 0x82,                    /* REPORT OUTPUT (Data,Var,Abs,Vol)    */
 /* 27 */

 /* TX_EP */
 0x85, deviceTxReportID,        /* TX_REPORT_ID(0x02) */
 0x09, 0x07,                    /* USAGE, USAGE, 0x09/0x?? for vendor-defined */
 0x15, 0x00,                    /* LOGICAL_MINIMUM (0) */
 0x26, 0xff, 0x00,              /* LOGICAL_MAXIMUM (255) */
 0x75, 0x08,                    /* REPORT_SIZE(8), unit of report = 8 bits ( or 16/32 bits) */
 0x95, deviceTxReportCount,     /* REPORT_COUNT(32), 32 bytes per packet, except ID */
 0xB1, 0x82,                    /* FEATURE (Data,Var,Abs,Vol) */

 0x85, deviceTxReportID,        /* REPORT_ID(0x01) */
 0x09, 0x07,                    /* USAGE, EP name 0x0709 */
 0x81, 0x82,                    /* REPORT INPUT (Data,Var,Abs,Vol) */

 /* 48 */
 /* USER CODE END 0 */
 0xC0                               /* END_COLLECTION    */
};
```

15.4 HID Report transmission and reception process

The following describes the process of sending and receiving USB-HID data using Set Report and Get Report on the device side (WT32L064). The examples use EP2 and EP1 respectively.



As mentioned above, we use the device administrator to detect whether the device (WT32L064) has the USB-HID function. After inserting the device, the target USB device will be added, and the device will be listed in the human interface device, as shown in the figure below.



15.4.1 Example of sending and receiving HID Report on the host side

As listed in the table below, the device has 3 endpoints, EP0 as Feature (bidirectional), EP1 as Report-IN, EP2 as Report-OUT, and then we set the data to be transmitted by the host (PC) EP2 as 0xAA, 0x22 , 0x00….0x00.

Endpoint	Type	Direction	Class	Subclass	Protocol	Max. Packet
0	Control	IN/OUT	3	0	0	64(8)
2	Interrupt	OUT	3	0	0	33
1	Interrupt	IN	3	0	0	33

Use the USB software tool of the PC to detect the data flow of the USB in real time. When the host sends the USB data, the OUT of EP2 as shown in the figure below has sent 0xAA, 0x22, 0x00….0x00, and then the host EP1 will receive 32 Byte data is returned by the device as 0xA1, 0x00….0x00, the result here is the same as the program setting.

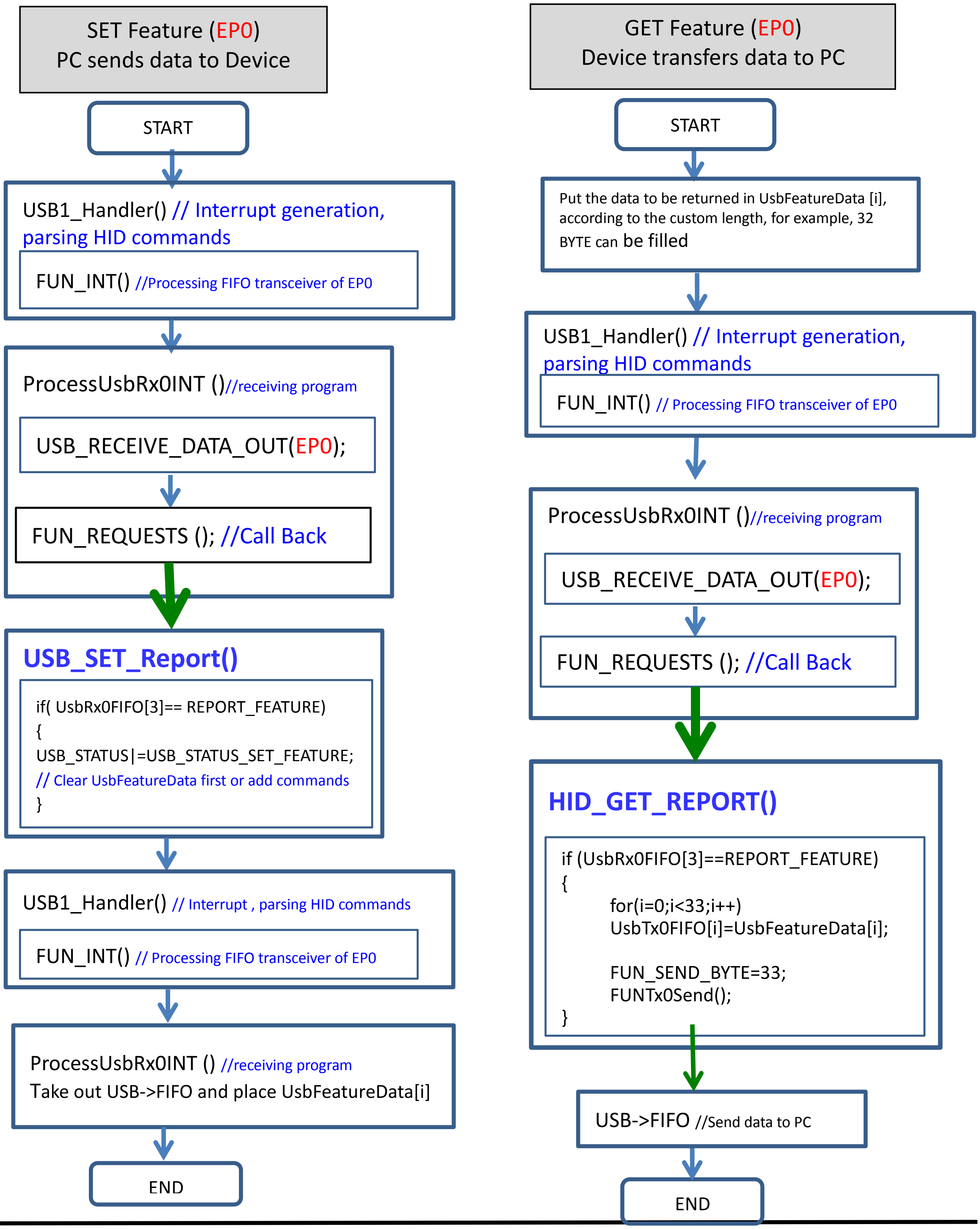
Endpoint	Direction	Data (hexadecimal)
2	OUT	AA 22 00 00_00 00 00 00_00 00 00 00_00 00 00 00 00 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00
1	IN	A1 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00 00 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00

The program is set to execute the return USB command 0xA1 when the USB command 0xAA is received, as shown in the following main.c program fragment:

```
if(EP2_DATA_BUFFER[0]==0xAA)
{
      EP2_DATA_BUFFER[0]=0;       //clear buffer
      UsbTx0FIFO[0]=0xA1;          // Set return 0xA1
      USBTxxSend(EP1, deviceTxReportCount);     // Execute USB EP1 transmit FIFO data
                                                 // deviceTxReportCount=32
}
```

15.5 HID Feature transmission and reception process

The following describes the process of sending and receiving USB-HID data by using SET Feature and GET Feature on the device side, and using EP0 respectively.



15.5.1 HID Feature Receiving Example

As listed in the table below, the device has 3 endpoints, EP0 can be used for feature two-way communication, EP1 is used as Report-IN, EP2 is used as Report-OUT, here USB communication uses EP0 as the control type setting, and the SETUP packet length The fixed value is 8 Bytes. We set the data to be transmitted by EP0 of the USB tool on the host side (PC) as 0xA1, 0x01, 0x00, 0x03, 0x00, 0x00, 0x08, 0x00, and then the data received from the device (WT32L064) is 0x01, 0x02 , 0x03….0x08.

Endpoint	Type	Direction	Class	Subclass	Protocol	Max. Packet
0	Control	IN/OUT	3	0	0	64(8)
2	Interrupt	OUT	3	0	0	33
1	Interrupt	IN	3	0	0	33

The HID command can refer to the following format, where 0xA1 and 0x01 are GET REPORT commands.

HID format	Request Type	bRequest	wValue		wIndex		wLength	
command string	0xA0	0x01	0x00(L)	0x03(H)	0x00(L)	0x00(H)	0x08(L)	0x00(H)
Command description	**Get_Report** (Feature Input,use EP0)		Report ID=0	Report Type =Feature	Interface No. =0		Length= 8 Bytes	

Use the USB software tool to actually detect the data flow of the USB. After pressing the execution, as shown in the table below, we send out 0xA1, 0x01, 0x00....0x00, and receive 8 bytes of data as 0x01, 0x02, 0x03....0x08 , the result here is the same as the program setting.

Endpoint	Direction	Data (hexadecimal)	Description
0	CTL	A1 01 00 03_00 00 08 00	GET REPORT
0	IN	01 02 03 04_05 06 07 08	----

15.5.2 HID Feature launch example

As shown in the figure below, set the data to be transmitted by EP0 of the USB tool on the PC side as 0x21, 0x09, 0x00, 0x03, 0x00, 0x00, 0x08, 0x00, and then transmit the data as 0x11, 0x22, 0x33….0x88.

Endpoint	Type	Direction	Class	Subclass	Protocol	Max. Packet
0	Control	IN/OUT	3	0	0	64(8)
2	Interrupt	OUT	3	0	0	33
1	Interrupt	IN	3	0	0	33

The HID command can refer to the following format, where 0x21 and 0x09 are SET REPORT commands.

HID format	Request Type	bRequest	wValue		wIndex		wLength	
command string	0x21	0x09	0x00(L)	0x03(H)	0x00(L)	0x00(H)	0x08(L)	0x00(H)
Command description	**Set_Report** (Feature output,use EP0)		Report ID=0	Report Type =Feature	Interface No. =0		Length= 8 Bytes	

We use the USB software tool to actually detect the data flow of the USB. After pressing the execution, as shown in the figure below, the host side transmits 0xA1, 0x01, 0x00....0x00, and then transmits 8 bytes of data as 0x11, 0x22, 0x33... .0x88, the result here is the same as the program setting.

Endpoint	Direction	Data (hexadecimal)	Description
0	CTL	21 09 00 03_00 00 08 00	SET REPORT
0	OUT	11 22 33 44_55 66 77 88	----

16. SPI function description

Use the following illustration to perform data transfer using SPI0 or SPI1. The action flow is as follows.

16.1 Initialize SPI after MCU is powered on

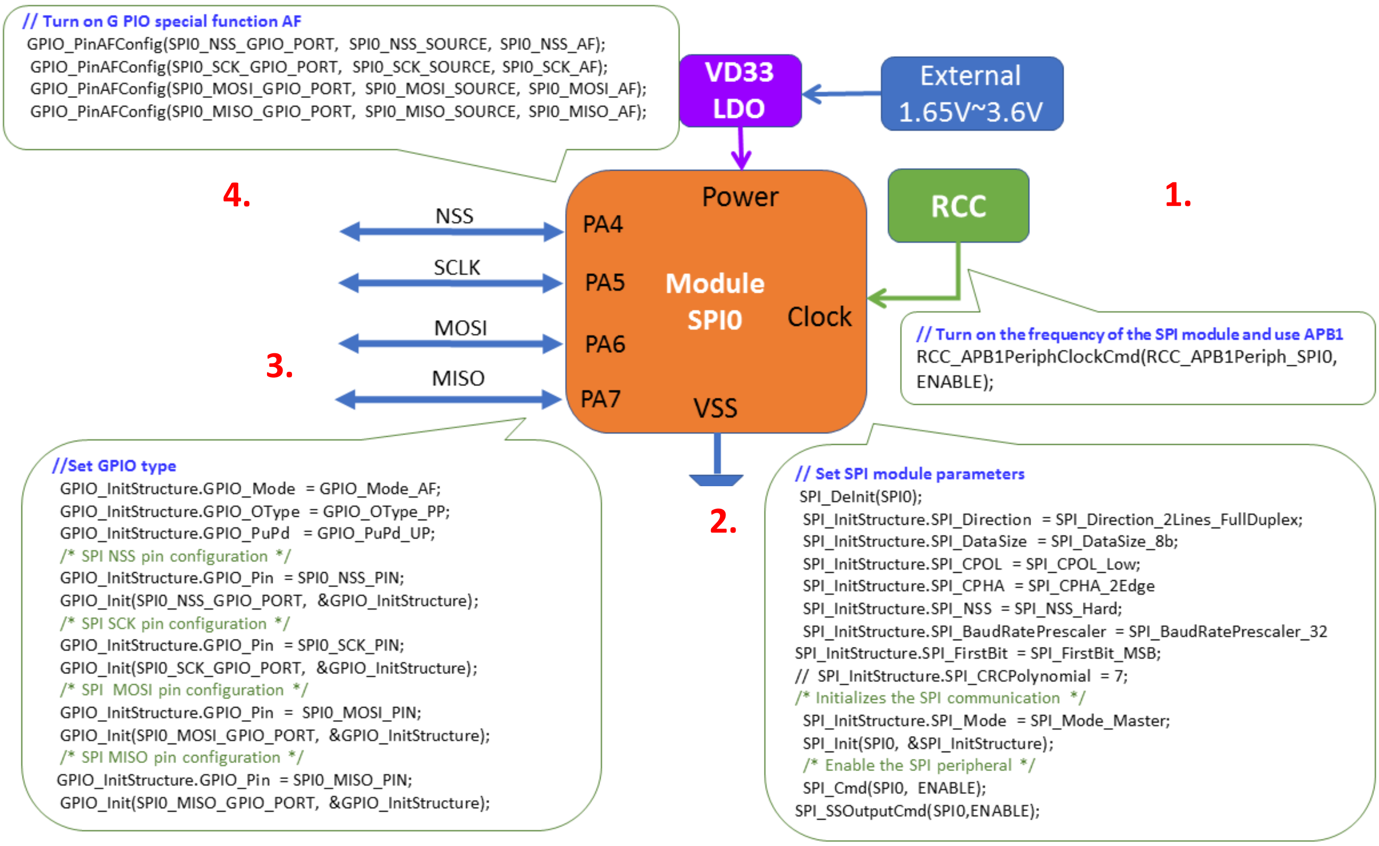
As shown in the following steps 1~4, you can refer to the peripheral library to use the function InitialSpi0( ) or InitialSpi1( ).

(Step 1) Set the RCC to enable the clock to be used by SPI, as shown in step 1 in the following figure.

(Step 2) Set the SPI module parameters, as shown in step 2 below.

(Step 3) Set the GPIO type, and set the push-pull and pull-up resistors as shown in step 3 in the figure below.

(Step 4) Set the GPIO type, set the AF3 type to make the IO have SPI function, as shown in step 4 below.



16.2 Sample Program

Refer to the function InitialSpi0 ( ) of wt32l0xx_pl_spi.c, the following programs are executed in sequence referring to the above steps 1~4.

```
void InitialSpi0(void)
{
```

1.
2.
3.
4.

```
    GPIO_InitTypeDef GPIO_InitStructure;
    /* Enable the SPI periph */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI0, ENABLE);

    /* SPI configuration -----------------------------------------------------*/
    SPI_DeInit(SPI0);
    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;//SPI_CPOL_Low;
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;//SPI_CPHA_1Edge;
    SPI_InitStructure.SPI_NSS = SPI_NSS_Hard;
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;//SPI_BaudRatePrescaler_4;
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
    //    SPI_InitStructure.SPI_CRCPolynomial = 7;

    SPI_InitStructure.SPI_Mode = SPI_Mode_Master; /* Initializes the SPI communication */
    SPI_Init(SPI0, &SPI_InitStructure);
    SPI_Cmd(SPI0, ENABLE); /* Enable the SPI peripheral */
    SPI_SSOutputCmd(SPI0, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;//GPIO_PuPd_DOWN;
    //    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;

    /* SPI NSS pin configuration */
    GPIO_InitStructure.GPIO_Pin = SPI0_NSS_PIN;
    GPIO_Init(SPI0_NSS_GPIO_PORT, &GPIO_InitStructure);

    /* SPI SCK pin configuration */
    GPIO_InitStructure.GPIO_Pin = SPI0_SCK_PIN;
    GPIO_Init(SPI0_SCK_GPIO_PORT, &GPIO_InitStructure);

    /* SPI    MOSI pin configuration */
    GPIO_InitStructure.GPIO_Pin = SPI0_MOSI_PIN;
    GPIO_Init(SPI0_MOSI_GPIO_PORT, &GPIO_InitStructure);

    /* SPI MISO pin configuration */
     // GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
    //GPIO_InitStructure.GPIO_PuPd    = GPIO_PuPd_NOPULL;     //GPIO_PuPd_DOWN;
    GPIO_InitStructure.GPIO_Pin = SPI0_MISO_PIN;
    GPIO_Init(SPI0_MISO_GPIO_PORT, &GPIO_InitStructure);

    GPIO_PinAFConfig(SPI0_NSS_GPIO_PORT, SPI0_NSS_SOURCE, SPI0_NSS_AF);
    GPIO_PinAFConfig(SPI0_SCK_GPIO_PORT, SPI0_SCK_SOURCE, SPI0_SCK_AF);
    GPIO_PinAFConfig(SPI0_MOSI_GPIO_PORT, SPI0_MOSI_SOURCE, SPI0_MOSI_AF);
    GPIO_PinAFConfig(SPI0_MISO_GPIO_PORT, SPI0_MISO_SOURCE, SPI0_MISO_AF);
}
```

17. I2C Function Description

Use the following illustration to perform data transfer using I2C0 or I2C1. The action flow is as follows.

17.1 Initialize I2C after MCU is powered on

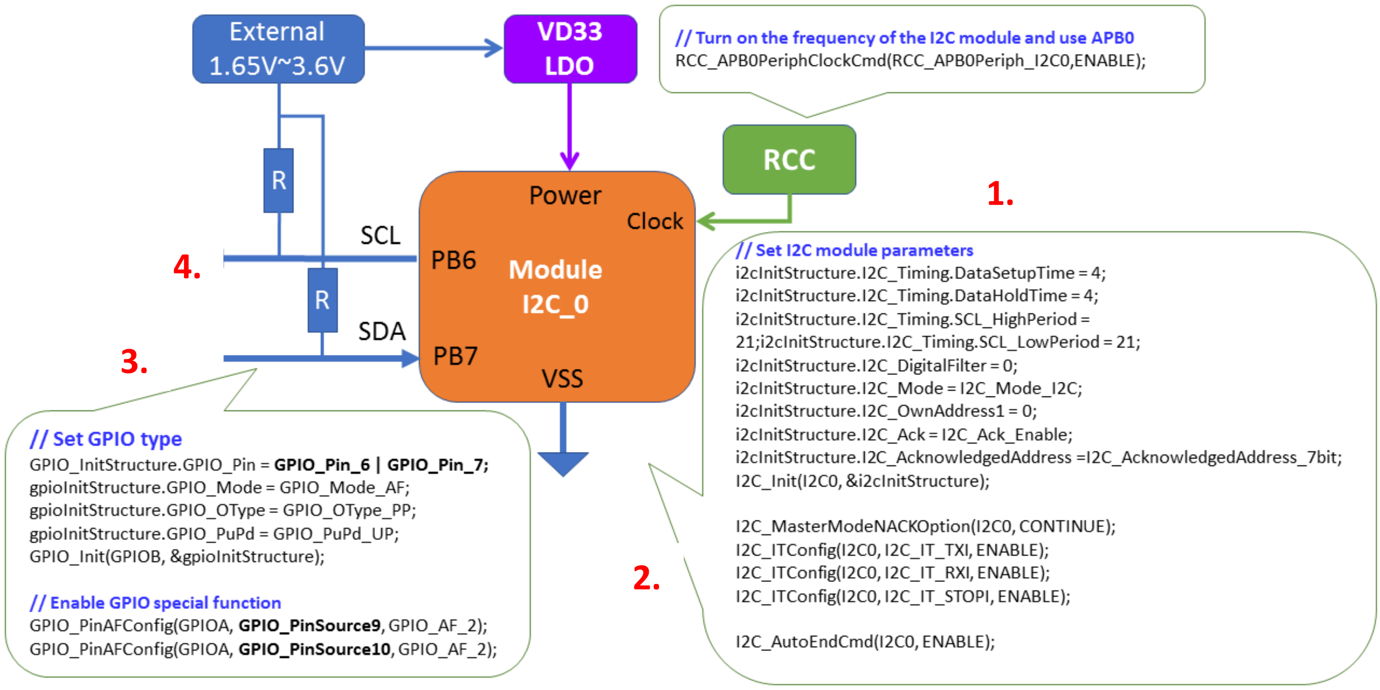
As shown in the following steps 1~4, you can refer to the peripheral library to use the function InitialI2c0( ) or InitialI2c1( ).

(Step 1) Set the RCC to enable the clock to be provided to I2C, as shown in step 1 in the following figure.

(Step 2) Set the I2C module parameters, as shown in step 2 below.

(Step 3) Set the GPIO type (the IO is set last, to avoid the signal from being poured into the module whose status is not determined), as shown in step 3 in the following figure.

(Step 4) Transmit I2C data, as shown in step 4 below.



17.2 Sample Program

```
void InitialI2c_0(uint8_t set, uint8_t mode)
{
    GPIO_InitTypeDef              gpioInitStructure;
    I2C_InitTypeDef               i2cInitStructure;

1.  RCC_APB0PeriphClockCmd(RCC_APB0Periph_I2C0, ENABLE);                    // enable clock for I2C0
```

2.

3

```
    if (mode == I2C_MASTER)
    {       //Master
            i2cInitStructure.I2C_Timing.DataSetupTime = 4;
            i2cInitStructure.I2C_Timing.DataHoldTime = 4;
            i2cInitStructure.I2C_Timing.SCL_HighPeriod = 234;    //(HSE=24MHz) 24:400K, 54:200K, 114:100K, 234:50K
            i2cInitStructure.I2C_Timing.SCL_LowPeriod = 234;
            i2cInitStructure.I2C_DigitalFilter = 0;
            i2cInitStructure.I2C_Mode = I2C_Mode_I2C;
            i2cInitStructure.I2C_OwnAddress1 = (0x00 >> 1);
            i2cInitStructure.I2C_Ack = I2C_Ack_Enable;
            i2cInitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
            I2C_Init(I2C0, &i2cInitStructure);
            I2C_MasterModeNACKOption(I2C0, CONTINUE);
    }
    else
    {
            //Slave
            i2cInitStructure.I2C_Timing.DataSetupTime = 0;
            i2cInitStructure.I2C_Timing.DataHoldTime = 0;
            i2cInitStructure.I2C_Timing.SCL_HighPeriod = 0;
            i2cInitStructure.I2C_Timing.SCL_LowPeriod = 0;
            i2cInitStructure.I2C_DigitalFilter = 0;
            i2cInitStructure.I2C_Mode = I2C_Mode_I2C;
            i2cInitStructure.I2C_OwnAddress1 = (0xA0 >> 1);
            i2cInitStructure.I2C_Ack = I2C_Ack_Enable;
            i2cInitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
            I2C_Init(I2C1, &i2cInitStructure);
            I2C_SlaveModeNACKOption(I2C1, CONTINUE);
    }

    if (set == 1)
            gpioInitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
    else if (set == 2)
            gpioInitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;

    gpioInitStructure.GPIO_Mode = GPIO_Mode_AF;
    gpioInitStructure.GPIO_OType = GPIO_OType_PP;                   //push-pull
    gpioInitStructure.GPIO_PuPd = GPIO_PuPd_UP;//GPIO_PuPd_NOPULL;//
    GPIO_Init(GPIOB, &gpioInitStructure);

    // connect I2C0 pins to I2C alternate function
    if (set == 1)
    {
            GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_1);
            GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_1);
    }
    else if (set == 2)
    {
            GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_1);
            GPIO_PinAFConfig(GPIOB, GPIO_PinSource9, GPIO_AF_1);
    }
}
```

17.3 I2C for RX receiving data and TX transmitting data

4.

```
void RunI2cTest(void)
{
 uint16_t     i;
 uint16_t TDATA_BUF[10];

 // --------------------------------
  // The host sends data to the slave
  // --------------------------------
 I2C_SlaveAddressConfig(I2C0, (0xA0 >> 1));
 I2C_MasterRequestConfig(I2C0, I2C_Direction_Transmitter);
 I2C_NumberOfBytesConfig(I2C0, 255);
 I2C_GenerateSTART(I2C0, ENABLE);
 while (!(I2C_GetFlagStatus(I2C1, I2C_FLAG_ADDR)));          // Slave Address match
 I2C_ClearITPendingBit(I2C1, I2C_IT_ADDR);

 I2C_SendData(I2C0, TDATA_BUF[i]);
 while (!(I2C_GetFlagStatus(I2C0, I2C_FLAG_TXE)));

 I2C_GenerateSTOP(I2C0, ENABLE);
 while ((I2C_GetFlagStatus(I2C0, I2C_FLAG_BUSY)));

 // --------------------------------
 // The host receives data from the slave
 // --------------------------------
 I2C_SlaveAddressConfig(I2C0, (0xA0 >> 1));
 I2C_MasterRequestConfig(I2C0, I2C_Direction_Receiver);
 I2C_NumberOfBytesConfig(I2C0, 255);
 I2C_GenerateSTART(I2C0, ENABLE);

 while (!(I2C_GetFlagStatus(I2C1, I2C_FLAG_ADDR)));          // Slave Address match
 I2C_ClearITPendingBit(I2C1, I2C_IT_ADDR);

 while (!(I2C_GetFlagStatus(I2C0, I2C_FLAG_RXNE)));          // Master RX Not Empty
 uint8_t temp = I2C_ReceiveData(I2C0);

 I2C_GenerateSTOP(I2C0, ENABLE);
 while ((I2C_GetFlagStatus(I2C0, I2C_FLAG_BUSY)));

 while (1);    //stop
}
```

18. I2S function description

Use the following illustration to perform data transfer using I2S0 or I2S1. The action flow is as follows.

18.1 Initialize I2S after MCU is powered on

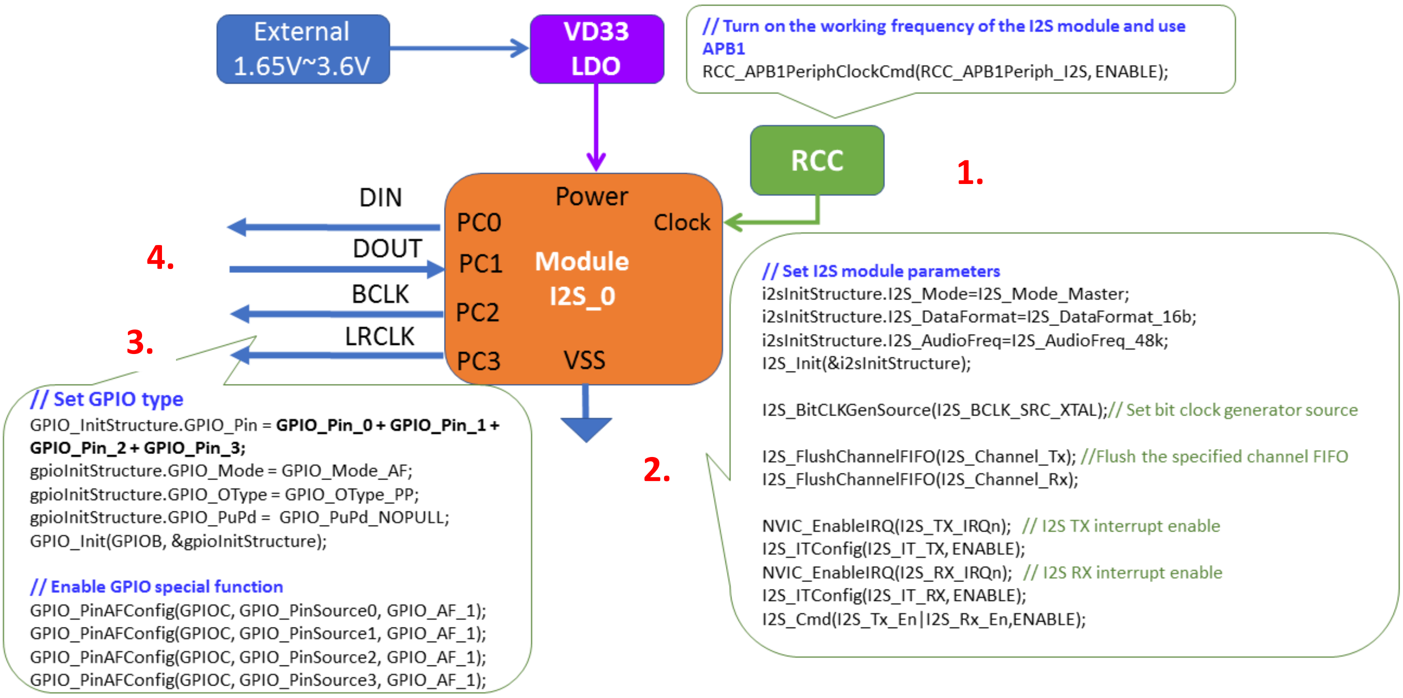
For steps 1~4 below, you can refer to the peripheral library to use the function InitialI2s0 ( ) or InitialI2s1 ( )

(Step 1) Set the RCC to enable the clock to be provided to I2S, as shown in step 1 in the following figure.

(Step 2) Set the I2S module parameters, as shown in step 2 below.

(Step 3) Set the GPIO type (IO last setting), as shown in step 3 below.

(Step 4) Transmit I2S data, as shown in step 4 below.



18.2 Sample Program

```
void InitialI2s_0(uint8_t set, uint8_t mode)
{
   GPIO_InitTypeDef   gpioInitStructure;    /* GPIO AF */
   I2S_InitTypeDef i2sInitStructure;
   /* reset I2S */
   I2S_DeInit();
   /* RCC Enable */
1  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2S, ENABLE);

   /* I2S initial */
2  i2sInitStructure.I2S_Mode = I2S_Mode_Master;
```

```
    i2sInitStructure.I2S_Standard = I2S_Standard_Phillips;

    i2sInitStructure.I2S_DataFormat = I2S_DataFormat_16b;
    i2sInitStructure.I2S_AudioFreq = I2S_AudioFreq_48k;
    I2S_Init(&i2sInitStructure);

    /* Set bit clock generator's clock source. */
    I2S_BitCLKGenSource(I2S_BCLK_SRC_XTAL);

    /* Flush the specified channel FIFO */
    I2S_FlushChannelFIFO(I2S_Channel_Tx);
    I2S_FlushChannelFIFO(I2S_Channel_Rx);

    /* I2S TX interrupt */
    NVIC_EnableIRQ(I2S_TX_IRQn);      // I2S TX interrupt enable
    I2S_ITConfig(I2S_IT_TX, ENABLE);

    /* I2S RX interrupt */
    NVIC_EnableIRQ(I2S_RX_IRQn);      // I2S RX interrupt enable
    I2S_ITConfig(I2S_IT_RX, ENABLE);
    I2S_Cmd(I2S_Tx_En | I2S_Rx_En, ENABLE);

    //Configure RCC
3.  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIO, ENABLE);

    //Configure GPIO C
    //PC0(I2S_DI), PC1(I2S_DO), PC2(I2S_BCLK), PC3(I2S_LRCK)
    gpioInitStructure.GPIO_Pin = GPIO_Pin_0;
    gpioInitStructure.GPIO_Mode = GPIO_Mode_AF;
    gpioInitStructure.GPIO_OType = GPIO_OType_PP;
    gpioInitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOC, &gpioInitStructure);
    gpioInitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOC, &gpioInitStructure);
    gpioInitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOC, &gpioInitStructure);
    gpioInitStructure.GPIO_Pin = GPIO_Pin_3;
    GPIO_Init(GPIOC, &gpioInitStructure);


    /* PC0(I2S_DI), PC1(I2S_DO), PC2(I2S_BCLK), PC3(I2S_LRCK)    */
    // Alt=1
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource0, GPIO_AF_1);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource1, GPIO_AF_1);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_1);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource3, GPIO_AF_1);

    while (1)
    {
4.      I2S_SendData(0x005500AA); // fill some data to TX0 FIFO
    }
}
```

19. PWM function description

Use the following illustration to implement width modulation output using PWM0A or PWM0B. The action flow is as follows.

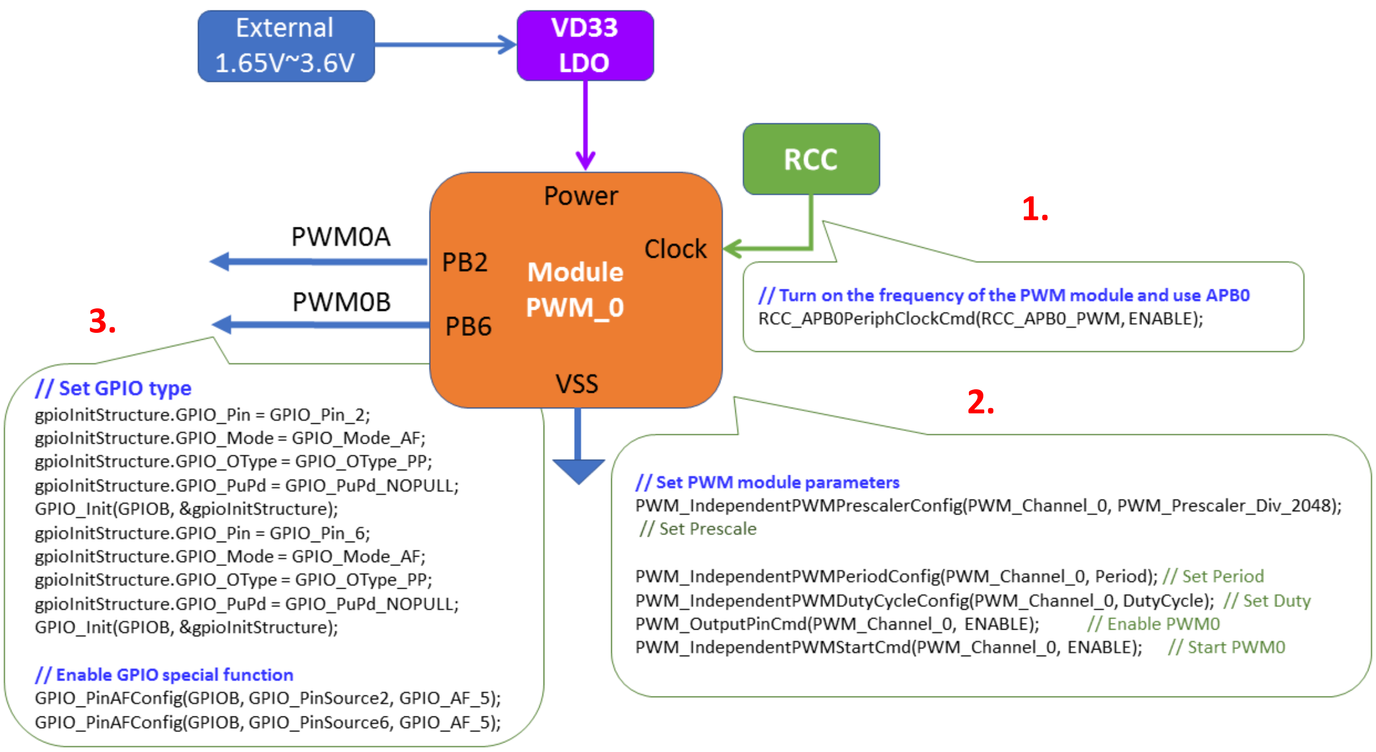
19.1 Initialize PWM after MCU is powered on

As shown in the following steps 1~4, you can refer to the peripheral library to use the function InitialPwm( )

(Step 1) Set the RCC to turn on the clock for PWM use, as shown in step 1 in the figure below.

(Step 2) Set the PWM module parameters, as shown in step 2 below.

(Step 3) Set the GPIO type (the IO is set last, to avoid the signal from being poured into the module whose status is not determined), as shown in step 3 in the following figure.



19.2 Sample Program

```
void InitialPwm(void)
{
    GPIO_InitTypeDef                gpioInitStructure;

1   PWM_DeInit();    // PWM clear
    RCC_APB0PeriphClockCmd(RCC_APB0_PWM, ENABLE);
```

2.

```
PWM_IndependentPWMPrescalerConfig(PWM_Channel_0, PWM_Prescaler_Div_2048); // Set Prescale
PWM_IndependentPWMPeriodConfig(PWM_Channel_0, Period); // Set Period
PWM_IndependentPWMDutyCycleConfig(PWM_Channel_0, DutyCycle);  // Set Duty
PWM_OutputPinCmd(PWM_Channel_0, ENABLE);         // Enable PWM0
PWM_IndependentPWMStartCmd(PWM_Channel_0, ENABLE);      // Start PWM0
```

3.

```
 // Set GPIO type
gpioInitStructure.GPIO_Pin = GPIO_Pin_2;
gpioInitStructure.GPIO_Mode = GPIO_Mode_AF;
gpioInitStructure.GPIO_OType = GPIO_OType_PP;
gpioInitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOB, &gpioInitStructure);
gpioInitStructure.GPIO_Pin = GPIO_Pin_6;
gpioInitStructure.GPIO_Mode = GPIO_Mode_AF;
gpioInitStructure.GPIO_OType = GPIO_OType_PP;
gpioInitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOB, &gpioInitStructure);
// Enable GPIO special function
GPIO_PinAFConfig(GPIOB, GPIO_PinSource2, GPIO_AF_5);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_5);

}
```

20. DMA function description

Please refer to the following illustration, use DMA0 and DMA1 channels to perform data transfer. The example is to read the ADC value, transfer the data to the Timer, and output the periodic waveform. The operation flow is as follows.

20.1 Initialize DMA after MCU is powered on

As shown in the following 1~2 steps, you can refer to the peripheral library to use the function InitiDma ( )

(Step 1) Set the DMA0 channel and transfer the ADC data to RAM address 0x300000000 through DMA0 as follows in step 1.

(Step 2) Set the DMA1 channel and transfer the RAM address 0x300000000 data to Timer2 via DMA1 as follows in step 2.

1.

```
// Set DAM 0 module parameters
DMA_DeInit(DMA_Channel0); //clear Register
DMA_StructInit(&DMA_0);    //Initialize Struct

DMA_0.DMA_SourceAddr=(uint32_t)ADC1_DR_ADDRESS; DMA_0.DMA_BufferSize = 2; // Data Length

DMA_0.DMA_SourceInc = DMA_SourceInc_Disable;
DMA_0.DMA_SourceDataSize =
DMA_SourceDataSize_Word;

DMA_0.DMA_DestinationAddr=(uint32_t) 0x30000000;
DMA_0.DMA_DestinationInc =
DMA_DestinationInc_Enable;
DMA_0.DMA_DestinationDataSize =
 DMA_DestinationDataSize_Word;

DMA_0.DMA_Mode = DMA_Mode_Normal;
DMA_0.DMA_Priority = DMA_Priority_High;
/ DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA_Channel0, &DMA_0);
/* Enable DMA1 Channel2 */
DMA_Cmd(DMA_Channel0, ENABLE);
DMA_ITConfig(DMA_Channel0,DMA_IT_TC,ENABLE);
NVIC_EnableIRQ(DMA0_IRQn);  // DMA  interrupt
```

2.

```
// Set DAM 1 module parameters
DMA_DeInit(DMA_Channel1);
// Initialize DMA Struct
DMA_StructInit(&DMA_InitStructure);

//DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize = 1;

DMA_InitStructure.DMA_SourceAddr=(uint32_t) 0x30000000;
DMA_InitStructure.DMA_SourceInc = DMA_SourceInc_Disable;
DMA_InitStructure.DMA_SourceDataSize = DMA_SourceDataSize_Word;

DMA_InitStructure.DMA_DestinationAddr = (uint32_t)TimerDmaAddr;
//TIM2_CCR1_ADDRESS;
DMA_InitStructure.DMA_DestinationInc =
DMA_DestinationInc_Disable;
DMA_InitStructure.DMA_DestinationDataSize =
DMA_DestinationDataSize_Word;

DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
// DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA_Channel1, &DMA_InitStructure);
/* Enable DMA1 Channel2 */
DMA_Cmd(DMA_Channel1, ENABLE);
```

20.2 Sample Program

```
void DMA_Config(uint32_t TimerDmaAddr)
{
 /* Enable DMA1 clock */
 RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA, ENABLE);
```

1.

```
//------ DMA 0 -------------------------------------------------------------------------
 // Initialize DMA hardware
DMA_DeInit(DMA_Channel0);
// Initialize DMA Struct
DMA_StructInit(&DMA_InitStructure);

//DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize = 2;
DMA_InitStructure.DMA_SourceAddr = (uint32_t)ADC1_DR_ADDRESS;
DMA_InitStructure.DMA_SourceInc = DMA_SourceInc_Disable;
DMA_InitStructure.DMA_SourceDataSize = DMA_SourceDataSize_Word;

//DMA_InitStructure.DMA_DestinationAddr = (uint32_t)TIM2_CCR1_ADDRESS;
DMA_InitStructure.DMA_DestinationAddr = (uint32_t)0x30000000;
DMA_InitStructure.DMA_DestinationInc = DMA_DestinationInc_Enable;
DMA_InitStructure.DMA_DestinationDataSize = DMA_DestinationDataSize_Word;

DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
// DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA_Channel0, &DMA_InitStructure);
/* Enable DMA1 Channel2 */
DMA_Cmd(DMA_Channel0, ENABLE);

DMA_ITConfig(DMA_Channel0, DMA_IT_TC, ENABLE);
NVIC_EnableIRQ(DMA0_IRQn);      // DMA    interrupt enable
```

2.

```
//------- DMA 1 -----------------------------------------------------------------------
 // Initialize DMA hardware
DMA_DeInit(DMA_Channel1);
// Initialize DMA Struct
DMA_StructInit(&DMA_InitStructure);

//DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize = 1;

DMA_InitStructure.DMA_SourceAddr = (uint32_t)0x30000000;
DMA_InitStructure.DMA_SourceInc = DMA_SourceInc_Disable;
DMA_InitStructure.DMA_SourceDataSize = DMA_SourceDataSize_Word;

DMA_InitStructure.DMA_DestinationAddr = (uint32_t)TimerDmaAddr;//      TIM2_CCR1_ADDRESS;
DMA_InitStructure.DMA_DestinationInc = DMA_DestinationInc_Disable;
DMA_InitStructure.DMA_DestinationDataSize = DMA_DestinationDataSize_Word;

DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
// DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
DMA_Init(DMA_Channel1, &DMA_InitStructure);
/* Enable DMA1 Channel2 */
DMA_Cmd(DMA_Channel1, ENABLE);
}
```

21. IWDT function description

Use the following illustrations to set the time using the IWDT. The action flow is as follows.

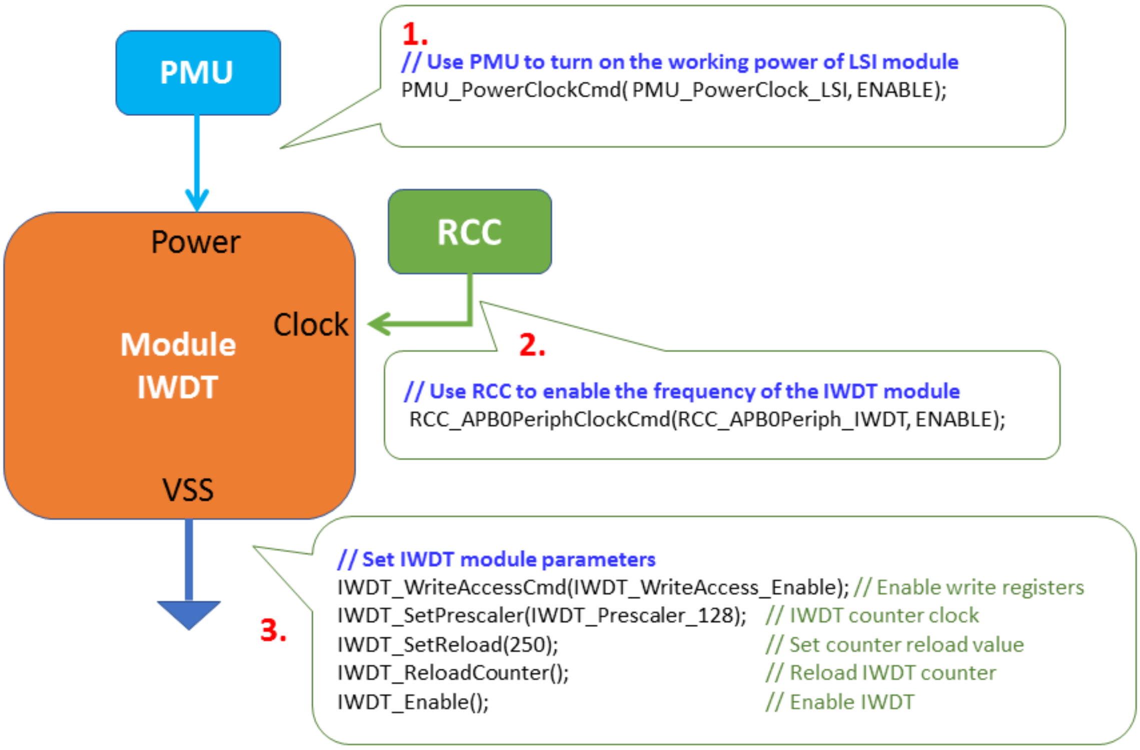
21.1 Initialize IWDT after MCU is powered on

As shown in the following steps 1~4, you can refer to the peripheral library to use the function InitialIwdt( )

(Step 1) Set the PMU (Power Management Unit) to turn on the analog power supply for the IWDT, as shown in step 1 below.

(Step 2) Set the RCC to enable the clock to be provided to the IWDT, as shown in step 2 in the following figure.

(Step 3) Set the parameters of the IWDT module, as shown in step 3 below.



21.2 Sample Program

```
void InitialIwdt(void)   {
1.  PMU_PowerClockCmd(PMU_PowerClock_LSI, ENABLE);
2.  RCC_APB0PeriphClockCmd(RCC_APB0Periph_IWDT, ENABLE);
    IWDT_WriteAccessCmd(IWDT_WriteAccess_Enable); // Enable access to IWDT_PR and IWDT_RLR registers
3.  IWDT_SetPrescaler(IWDT_Prescaler_128);// IWDT counter clock
    IWDT_SetReload(250);         // Set counter reload value   = 250ms / (LSI/32) = LsiFreq/128   =37K/128=250
    IWDT_ReloadCounter();// Reload IWDT counter
    IWDT_Enable();     // Enable IWDT
}
```

22. WWDT function description

Use the following illustration to set the time using WWDT. The action flow is as follows.

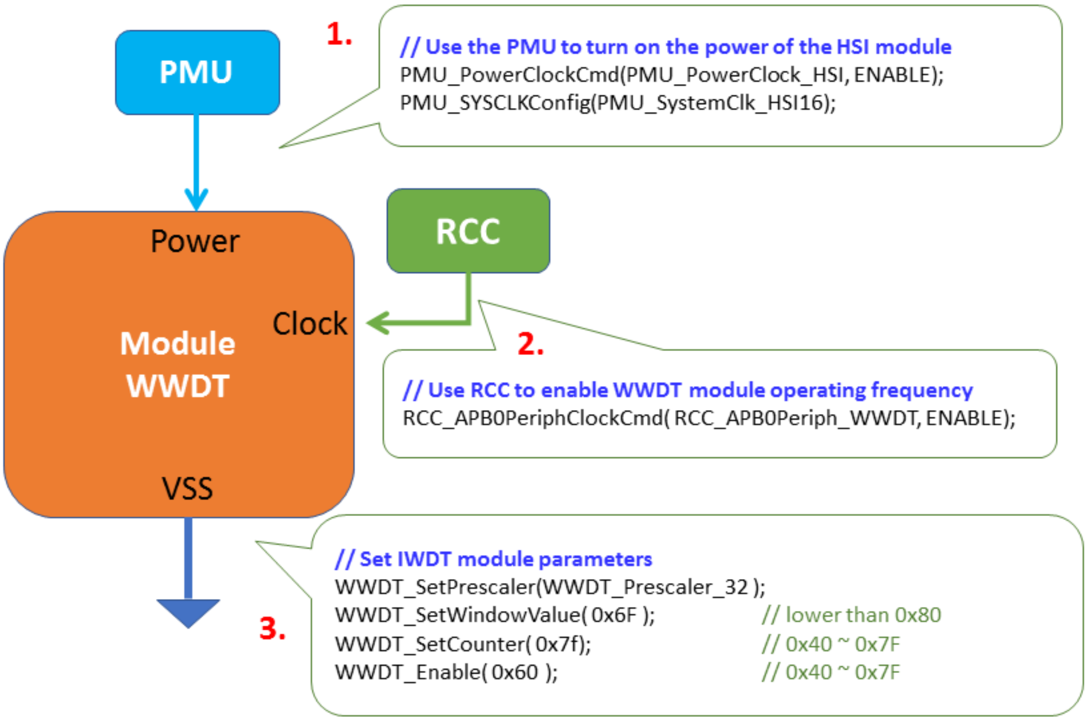
22.1 Initialize WWDT after MCU is powered on

As shown in the following steps 1~4, you can refer to the peripheral library to use the function InitialWwdt( )

(Step 1) Set the PMU (Power Management Unit) to turn on the analog power supply for the WWDT, as shown in step 1 below.

(Step 2) Set the RCC to enable the clock to be used by the WWDT, as shown in step 2 in the figure below.

(Step 3) Set the parameters of the WWDT module, as shown in step 3 below.

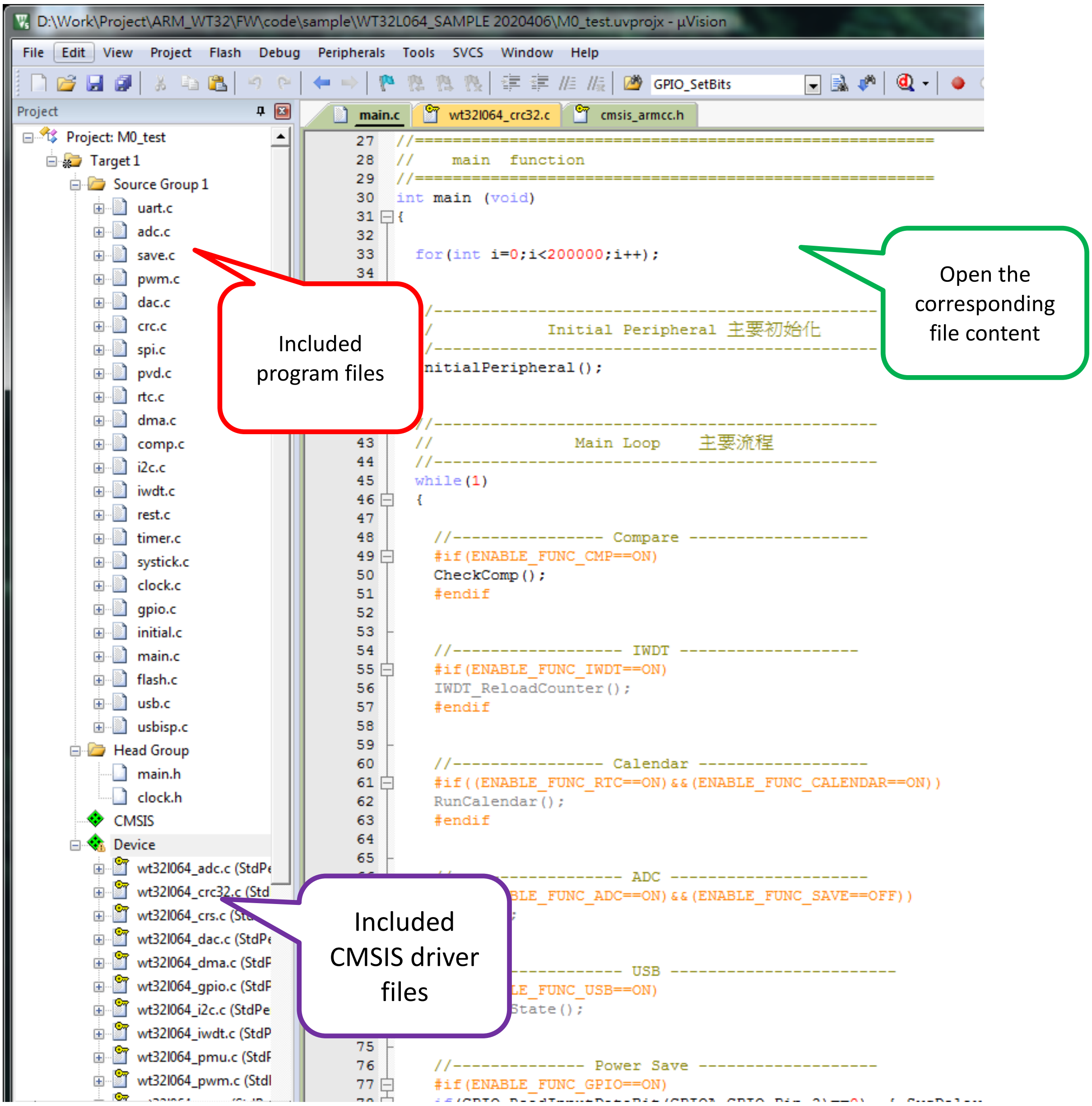


22.2 Sample Program

```
void InitialWwdt(void){
1.  PMU_PowerClockCmd(PMU_PowerClock_HSI, ENABLE);
    PMU_SYSCLKConfig(PMU_SystemClk_HSI16);
2.  RCC_APB0PeriphClockCmd(RCC_APB0Periph_WWDT, ENABLE);
    WWDT_DeInit();
3.  WWDT_SetPrescaler(WWDT_Prescaler_32);
    WWDT_SetWindowValue(0x6F);   // lower than 0x80
    WWDT_SetCounter(0x7f);       // 0x40 ~ 0x7F
    WWDT_Enable(0x60);           // 0x40 ~ 0x7F
}
```

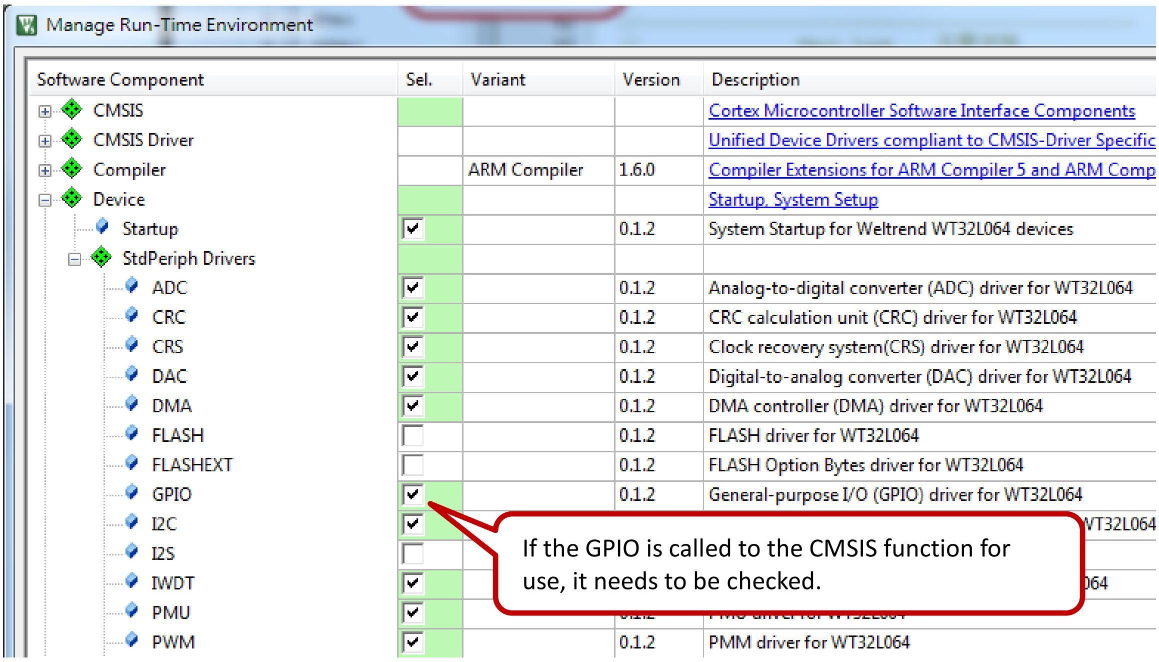
23. Example program operation instructions

The following illustrates how to use the reference example. Please refer to the example programs in the previous chapter, place the peripheral program library of the project into individual files according to the peripheral functions. After starting the project, the screen is as follows. It is divided into three parts: the project contains files, CMSIS drivers, and individual source file content.



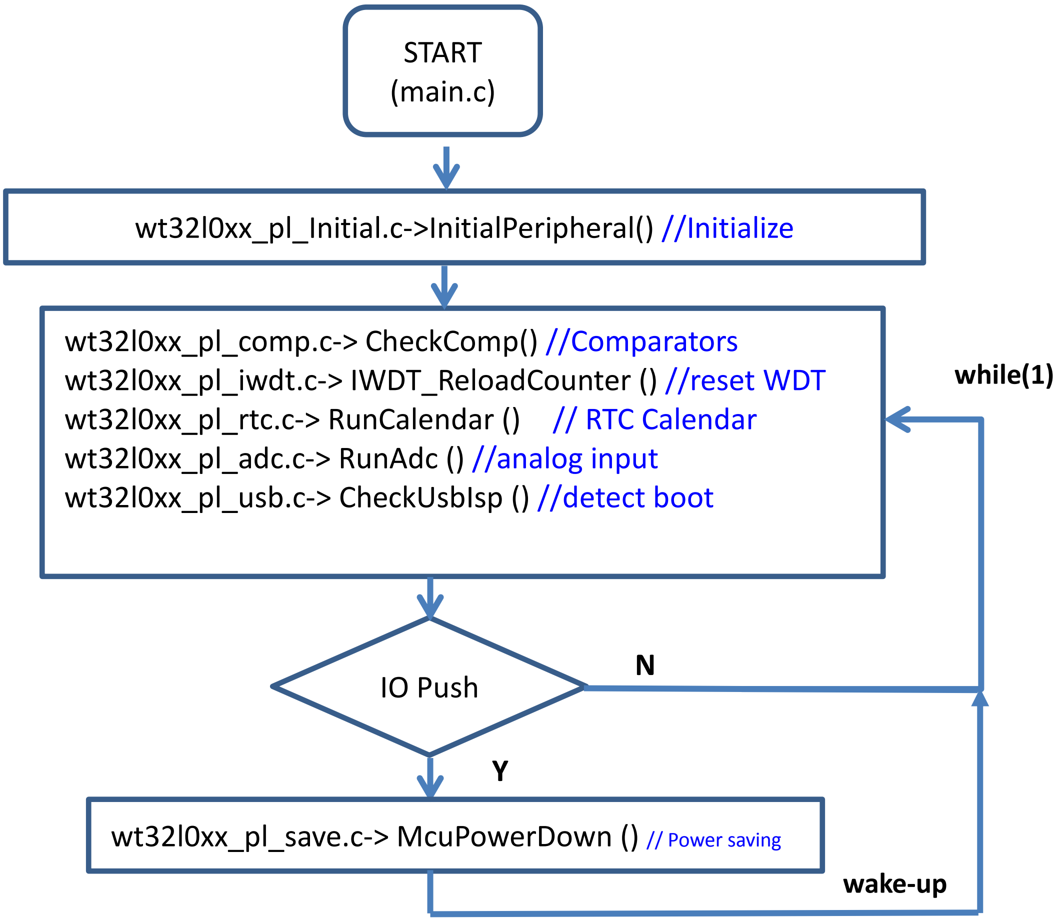
Add CMSIS driver layer functions for peripheral functions, click Manage Run-Time Environment on the upper menu of ARM-MDK, as shown in the figure below.

Click Device->StdPeriph Drivers in sequence as shown in the figure below, you can add the required functions according to the application requirements, e.g.: ADC, DAC, FLASH, GPIO, I2C...etc. The general example program has been added to the referenced CMSIS, if any missing parts or checked items can be re-selected.



23.1 Sample Flowchart of WT32L064_SAMPLE_2020xx

The following illustrates the flow chart of the sample program, and the main file contents and functions are as below.



According to the file name and function in the project, the description is as follows.

- **main.cThe main program flow, including the following functions**

1.) InitialPeripheral() -------------- Refer to initial.c, initialization of the surrounding
2.) CheckComp () ------------------ Refer to comp.c, the comparator output result
3.) IWDT_ReloadCounter() ------ Refer to iwdt.c to reset the watchdog counter
4.) RunCalendar() ------------------ Refer to rtc.c to detect the calendar value
5.) RunAdc() ------------------------- Refer to adc.c to perform ADC detection
6.) CheckUsbState() ---------------- Refer to usb.c to detect USB status
7.) McuPowerDown() -------------- Refer to save.c to perform power saving function

The main loop content of the program is as follows.

```
int main(void)
{
   for (int i = 0; i < 200000; i++); //Delay

   //----------------------------------------------
   //              Initial Peripheral main initialization
   //----------------------------------------------
   InitialPeripheral();

   //----------------------------------------------
   //                Main Loop                  Main process
   //----------------------------------------------
   while (1)     {
           //---------------- Compare -------------------
           #if(ENABLE_FUNC_CMP==ON)
           CheckComp();               // Check COMP comparator status
           #endif

           //------------------ IWDT -------------------
           #if(ENABLE_FUNC_IWDT==ON)
           IWDT_ReloadCounter();//watchdog overload
           #endif

           //---------------- Calendar ------------------
           #if((ENABLE_FUNC_RTC==ON)&&(ENABLE_FUNC_CALENDAR==ON))
           RunCalendar();             // Check RTC calendar data
           #endif

           //------------------ ADC ---------------------
           #if((ENABLE_FUNC_ADC==ON)&&(ENABLE_FUNC_SAVE==OFF))
           RunAdc();                  // Perform ADC detection
           #endif

           //------------------ USB ------------------------
           #if(ENABLE_FUNC_USB==ON)
           CheckUsbIsp();     // Check whether to enter Boot
           #endif

           //-------------- Power Save -------------------
           if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_2) == 0) {      SysDelay(100);
                   if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_2) == 0) {      //debounce

                   //-------- Sleep / Stop / Standby ----------
                   #if(ENABLE_FUNC_SAVE==ON)
                   McuPowerDown();              // enter power saving mode
                   #endif
                   }
           }
       };//while(1);
   }
```

- **wt32l0xx_pl_library.h Peripheral function switches, please enable or disable individual functions in sequence according to the requirements. The program content is as follows.**

```
//------------ Enable Function for Project ---------------------------
// Please enable the following functions in sequence, use ON to enable, OFF to disable

//------------- Core --------------------------
#define     SELECT_CORE_1p2V            OFF     // OFF:1.8V // ON: VCORE=1.2V
#define     ENABLE_FUNC_CLOCK           ON      // Set IRC 16M~32kHz
#define     ENABLE_FUNC_LSI               OF    t LSI 37kHz to enable or not
#define     ENABLE_USB_CLOCK            OFF      48MHz to enable or not
Switch off
#define     ENABLE_FUNC_SOFT_RST          OFF     // Set Soft Reset to enable or not

//------------ IO LED -------------------------
Switch on
#define     ENABLE_FUNC_GPIO            ON      function to enable or not
#if(ENABLE_FUNC_GPIO==ON)
#define ENABLE_GPIO_INT         OFF     // Set GPIO Interrupt to enable or not
#define     ENABLE_LED_BLINK        ON      // Set GPIO Port-C LED to enable or not
#define     ENABLE_LED_RESET        OFF     // Set GPIO test Reset to enable or not
#endif
#define     ENABLE_FUNC_SYSTICK         ON      // Set Systick to enable or not

//------------ Digtal ---------------
Digital function switch
#define     ENABLE_FUNC_UART                   功能to enable or not
#if(ENABLE_FUNC_UART==ON)
#define ENABLE_FUNC_UART0       ON      // Set UART0 to enable or not
#define ENABLE_FUNC_UART1       OFF     // Set UART1 to enable or not
#define ENABLE_HW_IRDA              OFF     // Set IRDAto enable or not 使用 UART0+1
#endif

#define     ENABLE_FUNC_PWM             OFF     // Set PWM to enable or not
#define     ENABLE_FUNC_IWDT        OFF     // Set IWDT to enable or not
#define     ENABLE_FUNC_WWDT        OFF     // Set WWDT to enable or not
#define     ENABLE_FUNC_FLASH       OFF     // Set Emulated EEPROM to enable or not
#define     ENABLE_FUNC_CRC             OFF     // Set CRC to enable or not
#define     ENABLE_FUNC_SPI             OFF     // Set SPI to enable or not
#define     ENABLE_FUNC_RESET       OFF     // Set Rest to enable or not        (test)
#define     ENABLE_FUNC_PVD             OFF     // Set Voltage detection to enable or not (test)
#define     ENABLE_FUNC_RESET       OFF     // Set Reset to enable or not       (test)
#define     ENABLE_FUNC_I2C             OFF     // Set I2C to enable or not
#define     ENABLE_FUNC_I2S             OFF     // Set I2S to enable or not
#define     ENABLE_FUNC_TIMER       OFF     // Set Timer to enable or not
#define     ENABLE_FUNC_DMA             OFF     // Set DMA to enable or not, use Timer+ADC
need enable both
#define     ENABLE_FUNC_USB             OFF     // Set USB to enable or not

analog function switch
//----------- Analog ----------------
#define     ENABLE_FUNC_CMP             ON      // Set COMPARE to enable or not
```

```
#define     ENABLE_HW_CMP_SPEED_HI      OFF     //HI:4.5uA  LO:5.5uA

#define     ENABLE_FUNC_ADC             OFF     // Set ADC to enable or not
#if(ENABLE_FUNC_ADC==ON)
#define ENABLE_HW_ADC_AWD               OFF
#define ENABLE_HW_ADC_ALL               OFF
#endif
#define     ENABLE_FUNC_DAC             OFF
```

RTC function switch

```
//-------- RTC -------
#define     ENABLE_FUNC_RTC                     OFF // Set RTC to enable or not
#if(ENABLE_FUNC_RTC==ON)
#define     ENABLE_FUNC_ALARM           OFF //RTC Enable first (59 sec)
#define     ENABLE_FUNC_CALENDAR    OFF         //RTC Enable first (not for sleep)
#define     ENABLE_RESET_RTC            OFF //ON: Test RTC keep RAM data
#endif
```

Power saving function switch

```
//------ Power Save ------
#define ENABLE_LPRUN_MODE               OFF     //GPIO canot change without BLDO

#if(ENABLE_LPRUN_MODE==OFF)
#define     ENABLE_FUNC_SAVE        OFF
#if(ENABLE_FUNC_SAVE==ON)
#define ENABLE_STANDBY_MODE         OFF
#define ENABLE_SLEEP_MODE               OFF     //ENABLE_FUNC_SYSTICK must OFF
#define ENABLE_STOP_MODE                ON
#endif
```

Wake-up function switch

```
//----- wake up -----
#if(ENABLE_FUNC_SAVE==ON)
#define     ENABLE_WAKE_GPIO        ON          //STADBY must OFF
#define     ENABLE_WAKEUP_CMP           OFF
#define     ENABLE_WAKEUP_ADC       OFF
#define     ENABLE_WAKEUP_DAC       OFF         //Only Output
#define     ENABLE_WAKEUP_RTC       OFF
#define     ENABLE_WAKEUP_IWDT          OFF
#endif
#endif
```

- **wt32l0xx_pl_initial.c The initialization of the surrounding, including the following functions**

1.) InitialPeripheral()-------------------Initialize peripheral functions EX: ADC, UART, PWM

Initialization sequence: InitialSysClock() -> InitialGpio() -> InitiSysTick() -> InitialUart0() ->... -> Initiallwdt() -> InitialAdc() -> InitialDac() -> SPI_Config0() -> InitialI2c() -> InitialPwm() -> InitialRtc()->...etc

- **wt32l0xx_pl_clock.h For the selection of operating frequency, four types of HIS, MSI, HSE and PLL can be selected. The program is as follows**

```
//--------- Use PLL for HSI 32MHz ----------------------------
#define CLOCK_PLL_HSI_X2_32MHZ    ON    //ON: Enable use of PLL multiplier HSI 16MHz to 32Hz for system use

//--------- Use PLL for USB 48MHz ----------------------------
#define USB_PLL       0         // 0:HSI48M, 1:PLL(From external crystal)

//--------- Select Frequency for MSI -------------------------
#define MSI_65K            PMU_MSIClock_Range0
#define MSI_131K        PMU_MSIClock_Range1
#define MSI_262K        PMU_MSIClock_Range2
#define MSI_524K            PMU_MSIClock_Range3
#define MSI_1M              PMU_MSIClock_Range4
#define MSI_2M              PMU_MSIClock_Range5
#define MSI_4M              PMU_MSIClock_Range6        //4.2MHz

#define MSI_CLOCK           MSI_4M        // When MSI is selected, the operating frequency selected by the system

//--------- Select MCU Clock Type ----------------------------
#defineCLK_HSI     0     //Internal OSC 16MHz
#defineCLK_MSI           1     //Internal OSC 65K~4M
#defineCLK_PLL           2     //Use Multiple X with HSI or HSE
#defineCLK_HSE           3 //External OSC 1~25MHz

#define       SYS_CLOCK_SEL    CLK_MSI           // The type of frequency selected by the system
```

choose speed

select type

- **wt32l0xx_pl_clock.c Working frequency setting function, including the following functions**

InitialSysClock () --------------- Perform system frequency selection, excerpted as follows

```
#if(SYS_CLOCK_SEL==CLK_HSI)          // Use HSI as system frequency
        PMU_PowerClockCmd(PMU_PowerClock_HSI, ENABLE);
        PMU_SYSCLKConfig(PMU_SystemClk_HSI16);
```

```
#elif(SYS_CLOCK_SEL==CLK_MSI)       // Use MSI as system frequency
        PMU_MSIConfig(MSI_CLOCK);                                   //Speed Setting
        PMU_PowerClockCmd(PMU_PowerClock_MSI, ENABLE);    //Power-On PLL
        PMU_SYSCLKConfig(PMU_SystemClk_MSI);        //Select System clock

#elif(SYS_CLOCK_SEL==CLK_PLL)       // Use PLL for system frequency
        //...omit
#elif(SYS_CLOCK_SEL==CLK_HSE)       // Use HSE as system frequency

        PMU_PowerClockCmd(PMU_PowerClock_HSE, ENABLE);
        PMU_SYSCLKConfig(PMU_SystemClk_HSE);
        #endif
```

1.) InitialUsbClock() --------------- Perform USB frequency selection

2.) Delayms() ----------------------- Execute delay function

3.) DelayCount() ------------------- Execute delay function

- **wt32l0xx_pl_gpio.c Peripheral IO type settings, including functions are as follows, please refer to Chapter 4**

1.) GPIO_Handler ()------------------Interrupt service GPIO function

2.) InitialGpio ()------------------------Initialize GPIO function

```
4 types of GPIO: GPIO_Mode_IN => basic input
                 GPIO_Mode_OUT => basic output
                 GPIO_Mode_AF => Composite use function, EX: UART, SPI, I2C ...
                 GPIO_Mode_AN => Analog input functions, EX: ADC, USB, COMP ...
```

- **wt32l0xx_pl_systick.c Built-in 24bit timer settings, including the following functions**

1.) SysTick_Handler ()-----------------Interrupt service systick function

2.) InitiSysTick ()----------------------Initialize the systick function

3.) SysDelay ()-------------------------Use the systick delay function

- **wt32l0xx_pl_flash.c Simulate EEPROM burning settings, including functions as follows**

1.) RunFlash ()------------------------Using the emulated EEPROM programming function

- **wt32l0xx_pl_uart.c Asynchronous transceiver transmission settings, including functions are as follows, please refer to Chapter 5**

1.) UART0_Handler ()------------Interrupt Service UART0 Function

2.) UART1_Handler()-------------Interrupt service UART1 function

3.) InitialUart0 ()-----------------Initialize UART0 function

4.) InitialUart1()-------------------Initialize UART1 function

5.) fputc ()----------------------------Use the function of sending serial data with printf()

6.) fgetc()-----------------------------Use the function of receiving serial data with printf()

7.) DRV_IntToStr()----------------Number to string

8.) Str2Num()---------------------- String to Number

9.) uart_send_str()---------------Use UART0/1 to transmit serial data

10.) uart_clear_str()---------------Clear the contents of the list

- **wt32l0xx_pl_adc.c analog detection settings, including the following functions**

1.) ADC_Handler ()----------------Interrupt service ADC function

2.) InitialAdc ()---------------------Initialize ADC function

3.) InitialAllAdc ()-------------------Initialize all ADC channel functions

4.) RunAdc()-------------------------Execute ADC target channel conversion function

5.) RunAllAdc ()---------------------Execute the conversion function of all channels of ADC

6.) RunAdcConvert()---------------Execute ADC channel single conversion function

7.) API_AverADCData ()------------Execute ADC channel conversion function, calculate average

8.) ADC_StartOfConversion_1() - Start ADC module conversion

9.) ADC_StopOfConversion_1() - Stop ADC module conversion

10.) HEX2BCD()-----------------------hexadecimal to decimal

- **wt32l0xx_pl_save.c power saving function settings, including the following functions**

1.) McuPowerDown ()---------------Perform the pre-operation of power saving function and call Save()

2.) Save()------------------------------- Perform power saving function according to the setting of SLEEP, STOP, STANDBY

- **wt32l0xx_pl_pwm.c Pulse period modulation function settings, including the following functions**

1.) InitialPwm() ---------------------- Perform PWM initialization and output function

- **wt32l0xx_pl_dac.c Analog output settings, including functions as follows**

1.) DAC_Convert () --------------------- Bring value to DAC and output function
2.) DAC_Handler()----------------------Execute DAC interrupt function
3.) InitialDac()--------------------------- Perform DAC initialization

- **wt32l0xx_pl_crc.c Check code CRC settings, including the following functions**

1.) DAC_Convert () --------------------- Bring value to DAC and output function
2.) DAC_Handler()----------------------Execute DAC interrupt function

- **wt32l0xx_pl_spi.c serial peripheral transmission settings, including the following functions**

1.) SPI_Config0 () ---------------------- Execute SPI 0 initialization
2.) SPI_Config1()-------------Execute SPI 1 initialization
3.) SPI1_Handler()----------------------Execute SPI interrupt function

- **wt32l0xx_pl_pvd.c voltage detection settings, including the following functions**

1.) InitPvd () ----------------------------- Perform PVD initialization
2.) PVD_Handler ()----------------------Execute PVD interrupt function

- **wt32l0xx_pl_rtc.c real-time counter settings, including the following functions**

1.) InitialRtc () ------------------------- Perform RTC initialization
2.) RTC_AlarmCmd ()------------------Execute DAC interrupt function
3.) RTC_Handler()----------------------Execute RTC interrupt function
4.) RunCalendar()----------------------Execute the RTC calendar function
5.) SetAlarm()--------------------------- Set the RTC alarm function

- **wt32l0xx_pl_dma.c direct memory access settings, including the following functions**

1.) ADC_Config () -----------Execute ADC initialization
2.) DMA_Config ()----------------------Execute DMA initialization
3.) DMA0_Handler()--------------------Execute DMA interrupt function
4.) InitiDma()---------------------------Initialize the DMA channel

5.) RunDma()--------------------------- Perform the above ADC move to DMA

- **wt32l0xx_pl_comp.c comparator settings, including the following functions**

1.) CheckComp() ---------------------Check COMP0 or COMP1
2.) CMP0_VOUT_Handler ()------------------Execute CPM0 interrupt function
3.) CMP1_VOUT_Handler()------------------Execute CMP1 interrupt function
4.) InitialComp()--------------------------------Initialize the COMP comparator
5.) RumComp() -------------------------------- Execute the COMP comparator

- **wt32l0xx_pl_i2c.c Standard I²C bus settings, including the following functions**

1.) InitialI2c () ----------------------- -Initialize I2C transfer
2.) RunI2cTest ()----------------------Execute I2C transfer

- **wt32l0xx_pl_iwdt.c watchdog settings, including the following functions**

1.) InitialIwdt () --------------------- Initialize the watchdog

- **wt32l0xx_pl_reset.c software reset settings, including the following functions**

1.) InitLowVoltReset () ----------------- Initialize low voltage reset
2.) RunReset ()--------------------------- Test low voltage reset

- **wt32l0xx_pl_timer.c count timer settings, including the following functions**

1.) ConfigTimerCapture () ----------- Configure Timer to execute capture mode
2.) ConfigTimerClockGpio ()------------------- Configure Timer to execute output mode
3.) ConfigTimerInterrutp()------------Configure Timer to execute interrupt mode
4.) ConfigTimerOutPWM()------------Configure Timer to execute PWM mode
5.) ConfigTimerTimeMode()------------------- Configure Timer to execute timer mode
6.) TIMER0_Handler()----------------------Execute TIMER0 interrupt function
7.) TIMER1_Handler()----------------------Execute TIMER1 interrupt function
8.) TIMER2_Handler()----------------------Execute TIMER2 interrupt function

- **wt32l0xx_pl_usb.c General serial bus settings, including the following functions**

1.) CLEAR_STALL () --------------------------- Clear EP endpoint STALL stall status
2.) ENDPOINT_DISABLE ()------------------- Disable EP endpoint function
3.) FUN_INIT()----------------------------------- Initialize USB endpoint EP0 or other endpoints

4.) FUN_INT()---------------------------- USB endpoint EP0~EPx interrupt service function

5.) FUN_INT2()------------------------- Handle terminal EP2 endpoint interrupt

6.) FUN_REQUESTS()------------------- After the PC sends the USB request command, it processes and parses the USB command

7.) FUNTx0Send()-----------------------device transfers data to USB FIFO

8.) HID_EP1()----------------------------USB endpoint EP1 transmits data

9.) HID_EP2()----------------------------USB endpoint EP2 transmits data

10.) HID_EP3()----------------------------USB endpoint EP3 transmits data

11.) HID_GET_IDLE()----------------------------USB-HID get IDLE time setting

12.) HID_GET_PROTOCOL()-------------------USB-HID get PROTOCOL setting

13.) HID_GET_REPORT()-------------USB-HID gets the value set by REPORT

14.) HID_SET_IDLE()----------------------------USB-HID set IDLE setting value

15.) HID_SET_PROTOCOL()-------------------USB-HID sets PROTOCOL format

16.) HID_SET_REPORT()-------------USB-HID set REPORT format

17.) IN_ENDPOINT_ENABLE()------------------- Start the IN function of the USB endpoint EP

18.) OUT_ENDPOINT_ENABLE()---------------Start the OUT function of the USB endpoint EP

19.) ProcessUsbResetINT()---------------------Reset and initialize USB endpoints EP0~EPx

20.) ProcessUsbRx0INT()-----------------------Process EP0 receive interrupt process

21.) ProcessUsbTx0INT()-----------------------Process EP0 transmit interrupt process

22.) ProcessUsbxx1INT()-----------------------Processing EP1 transceiver interrupt process

23.) ProcessUsbxx2INT()-----------------------Processing EP2 transceiver interrupt process

24.) ProcessUsbxx3INT()-----------------------Processing EP3 transceiver interrupt process

25.) SendFirstBuffer()----------------------------Send the first descriptor to the PC

26.) SendFirstBufferWithSize()---------------Sends the first descriptor to the PC with length

27.) SendNextBuffer()--------------send the second descriptor to the PC

28.) SET_STALL()------------------------------Set endpoint EP to stall

29.) USB_CLEAR_FEATURE()------------------Clear Feature configuration, process USB standard clear command

30.) USB_GET_CONFIG()-----------Get Config configuration
31.) USB_GET_DESCRIPTOR()---------------- Get Descriptor descriptor
32.) USB_GET_INTERFACE()-------------------Get Interface configuration
33.) USB_GET_STATUS()-------------Read STATUS status
34.) USB_NOT_SUPPORT()------------------- Response is not supported
35.) USB_RECEIVE_DATA()--------------------Read USB receive data
36.) USB_SET_ADDRESS()---------------------Sets the USB device address
37.) USB_SET_CONFIG()------------Set Config configuration
38.) USB_SET_FEATURE()-----------------------Set Feature configuration
39.) USB_SET_INTERFACE()---------------Set Interface configuration
40.) USB0_Handler()-------------USB signal interrupt vector service routine
41.) USB1_Handler()-------------USB endpoint EP interrupt vector service routine
42.) USBTxxINT()----------------------------USB transfer interrupt
43.) USBTxxSend()----------------------------USB transfer preload Buffer

- **wt32l0xx_pl_usbisp.c The general serial bus enters the Boot settings, including the following functions**

1.) CheckUsbIsp()-------------Check if the USB plug is connected to HOST
2.) enter_usbisp()--------------------------Execute USB burning ISP program, reset after setting
3.) go_usb_suspend()---------------------Enter Suspend standby power saving mode
4.) InitialUSB()-------------------------------performs USB initialization

24. Revidion History:

Version	History	Date
V1.0	Initial issue	2022/05/12