WT32L064_032
周边功能与程式说明
应用文件

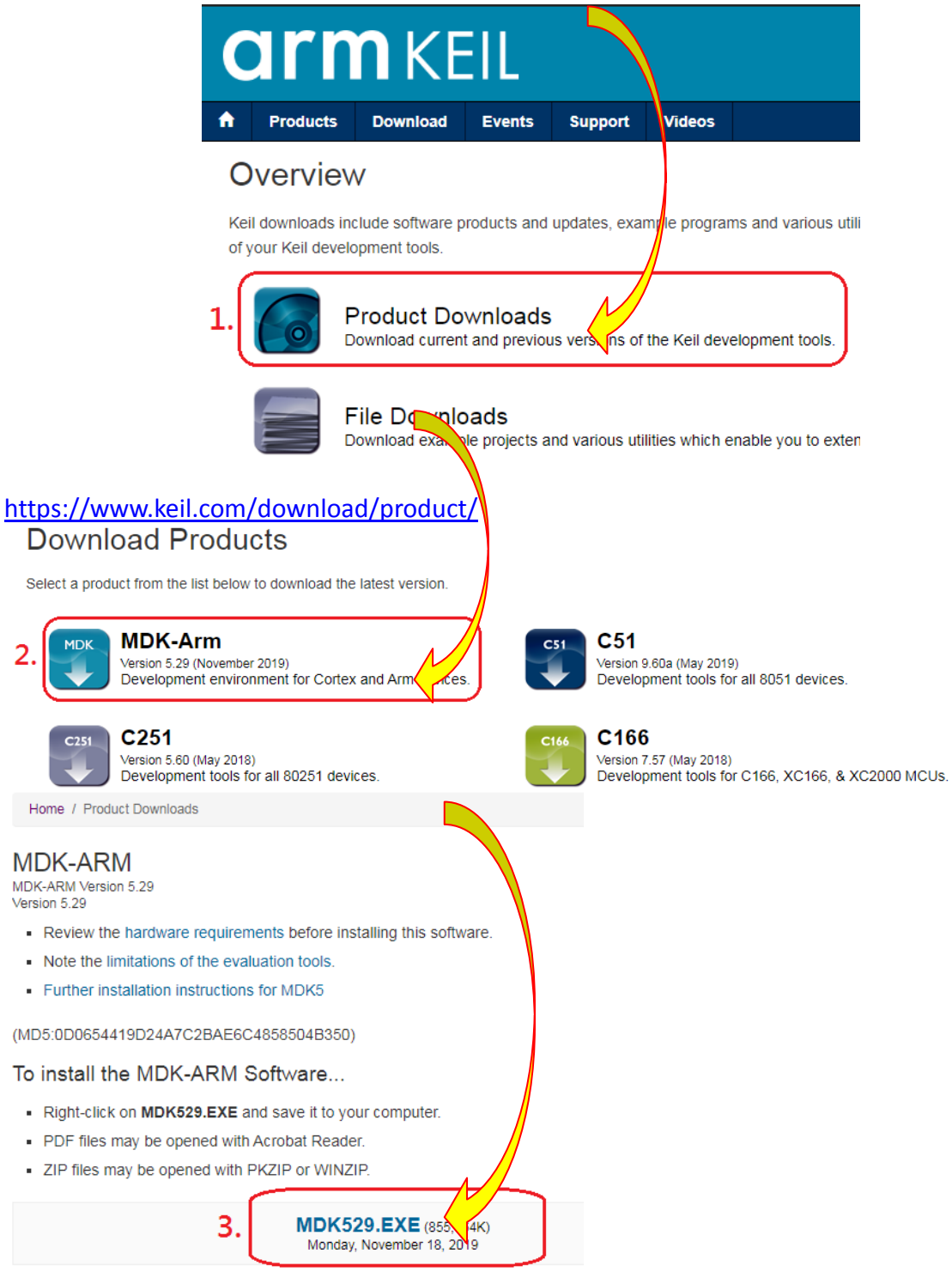
(简体版)

Rev. 1.0
September 2020

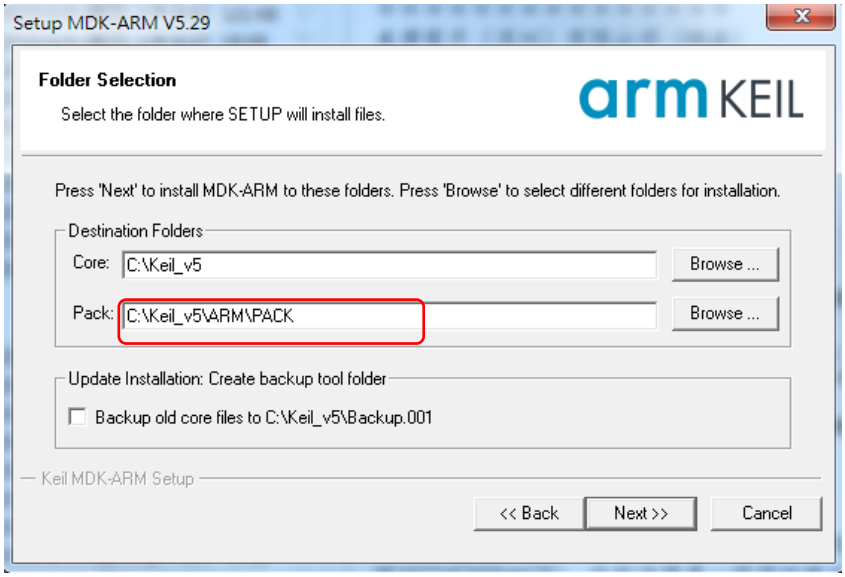
目 录

1. ARM-MDK 安装与环境设定

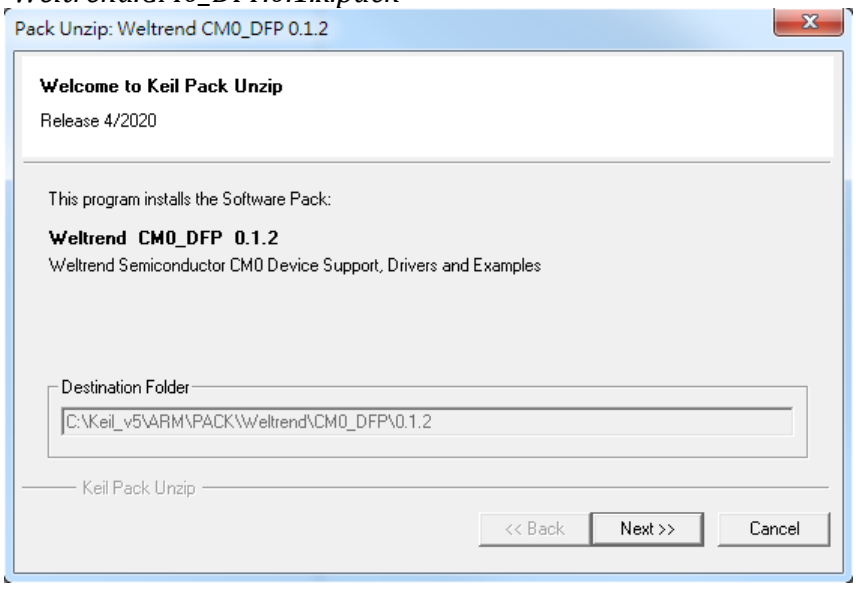
(Step 1) 请先上网下载 **ARM-MDK**，https://www.keil.com/download/



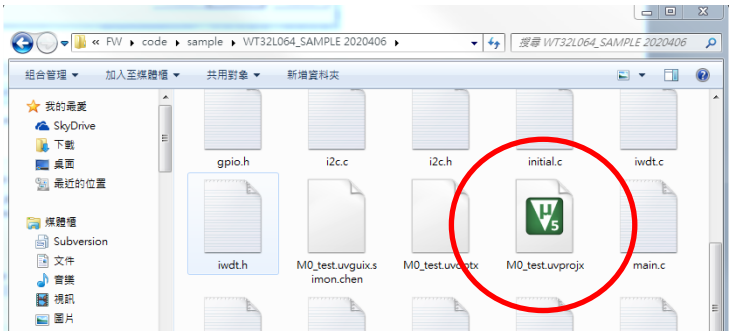
安装过程中会询问预设 PACK 路径，请指定 **C:\Keil_v5\ARM\PACK** 如下，避免后续 PACK 安装问题



(Step 2)下载并安装 MDK 后，请于 PC 端再安装伟诠 PACK 档案 *Weltrend.CM0_DFP.0.1.x.pack*



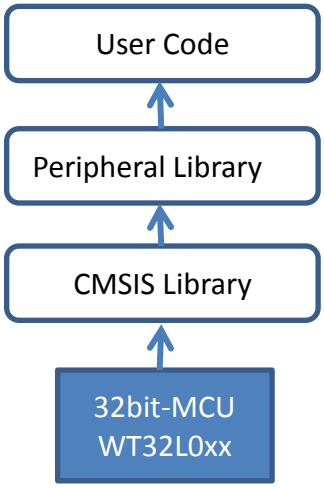
(Step 3)　安装 ARM-MDK 后有基础 32KB 可免费使用，或可自行采购软体，安装后请在电脑上开启相关 WT32L064 专案进行编译工作。



2. CMSIS 中间层驱动说明

2.1 定义:

ARM® Cortex™ 微控制器软体介面标准 (CMSIS)是一组韧体库可驱动 ARM 处理器，该韧体介面提供一标准函式直接面向周边且名称一致使用简单，可利软体的重复呼叫使用，缩短微控制器开发人员开发时间。与此架构上厂商再提供一层周边程式库(Peripheral Library)直接面向基本应用，提供一组基本初始化与操作范例程式，直接面向应用端可加快程式操作与编写。



2.2 应用说明:

CMSIS 目的是将函式对应到 MCU 的暂存器控制描写出来，对于使用者可使用标准化函式例如 ADC_StartOfConversion() ，而周边程式库(PL)则是提供该函式的操作与范例。

EX: 程式档案 main.c 内容

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

周边应用的档案 wt32l0xx_pl_adc.c 内容

```
API_AverADCData() {
ADC_StartOfConversion();   //使用 CMSIS 标准，呼叫周边函式进行平均滤波 ADC
……
}
```

CMSIS 驱动的档案 wt32l064_adc.c 内容

```
void ADC_StartOfConversion(void)
{
   ADC->ADCCR |= (uint32_t)ADC_START;   // 实际对应到 MCU 暂存器位址
}
```

2.3 CMSIS 内容说明:

安装完 WT32L064 PACK 后，预设 CMSIS 的路径为 C:\Keil_v5\ARM\Packs\Weltrend\CM0_DFP\0.1.2 \WT32L064\StdPeriph_Driver，标头档放置 Include 资料夹，原始档放置 Source，其内容有对 WT32L064 所有的周边做基础设定，档案清单如下。

档案名称	功能说明
wt32l064_adc	类比侦测 ADC 相关函式
wt32l064_crc32	CRC32 计算关函式
wt32l064_crs	校正 IC 内部频率相关函式
wt32l064_dac	类比输出 DAC 相关函式
wt32l064_dma	直接记忆体存取 DMA 相关函式
wt32l064_flash	仿真式 EEPROM 烧录 FLASH 相关函式
wt32l064_gpio	GPIO 相关函式
wt32l064_i2c	I2C 相关函式
wt32l064_i2s	I2S 相关函式
wt32l064_iwdt	IWDT 独立看门狗相关函式
wt32l064_pmu	PMU 电源控制单元相关函式
wt32l064_pwm	PWM 相关函式
wt32l064_rcc	RCC 频率控制单元相关函式
wt32l064_rtc	RTC 计时器相关函式
wt32l064_spi	SPI 相关函式
wt32l064_timer	TIMER 相关函式
wt32l064_usart	UART 相关函式
wt32l064_usbd	USB 相关函式
wt32l064_wwdt	WWDT 视窗型看门狗相关函式

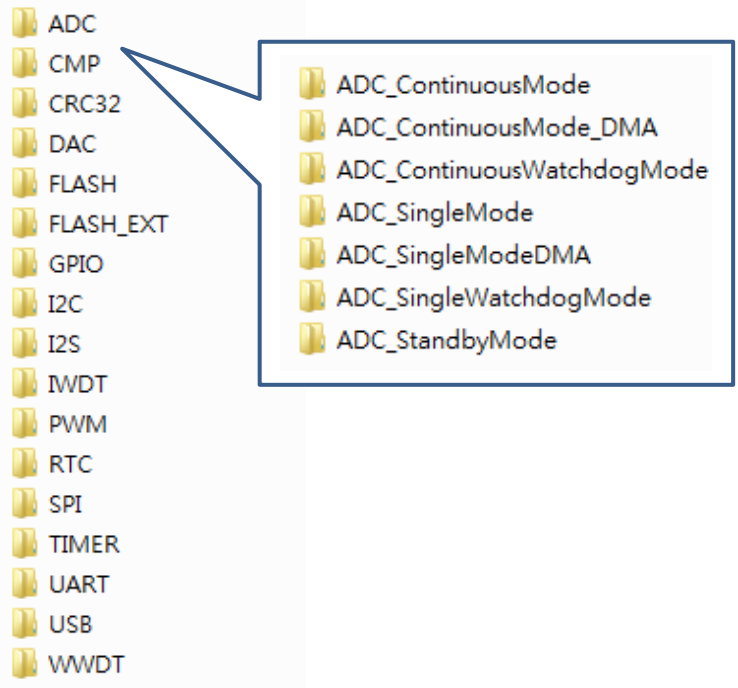
每个档案开头都有简易说明该档案的目的功能为何，内部每个函式亦有针对该功能与参数做说明，举例说明档案 wt32l064_gpio.c 内，其中 GPIO_SetBits()函数的内容如下

```
/**
   * @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: 主要功能为 Bit 设定
@param GPIOx: 目标 PORT
@param GPIO_Pin: 目标 PIN
@note: 补充说明 PIN 有 0~15 且有 PortA~D

3. PACK 范例程式架构说明

针对各种应用单元有基本范例程式，当 PACK 安装后参考下列路径 C:\...\Arm\Packs\Weltrend\CM0_DFP\0.1.2 \WT32L064\Examples，资料夹内有子单元内含原始档与专案，下列为 ADC 范例程式，依其功能有单一次转换与连续转换与其分类，如下图示所示。



资料夹内含有相对应功能的范例程式，例如 ADC->ADC_SingleMode 为单次类比转换功能，该资料夹内有 readme.txt 可参阅简易说明

3.1 Examples 资料夹内功能说明

根资料夹	资料夹名称	功能说明
ADC	ADC_ContinuousMode	连续 ADC 侦测
	ADC_ContinuousMode_DMA	使用 DAM 作连续 ADC 侦测
	ADC_ContinuousWatchdogMode	使用连续 ADC 做边界侦测
	ADC_SingleMode	单一 ADC 侦测
	ADC_SingleModeDMA	使用 DAM 作单一 ADC 侦测
	ADC_SingleWatchdogMode	使用单一 ADC 做边界侦测
	ADC_StandbyMode	使用低耗电 ADC 模式
CMP	CMP	比较器范例
CR32	CRC32	CRC32 计算范例
DAC	DAC	DAC 输出范例
	DAC_HighCurrent	DAC 高推力输出范例
FLASH	FLASH_PROGRAM	烧录程式区(EEPROM)范例
	FLASH_PROGRAM_INT	烧录程式区(EEPROM)中断范例
FLASH_EXT	FLASH_OB_EEPROM	烧录资料区(EEPROM)范例
	FLASH_OB_LEVEL	于资料区(OB)作加密等级

根资料夹	资料夹名称	功能说明
	FLASH_OB_READ_PROTECTION	于资料区(OB)作防读加密
	FLASH_OB_WRITE_PROTECTION	于资料区(OB)作防写加密
GPIO	GPIO	GPIO 基本范例
	GPIO_Bit_Set_Reset	设定 GPIO 位元的范例
	GPIO_Input	设定 GPIO 输入的范例
	GPIO_Interrupt	设定 GPIO 中断的范例
	GPIO_Output	设定 GPIO 输出的范例
	GPIO_Toggle	设定 GPIO 输出反向的范例
I2C	I2C_Master_Slave_DMA_FLAG	I2C 从端模式与 DMA 搬移
	I2C_Master_Slave_DMA_INT	I2C 从端模式与 DMA 中断
	I2C_Master_Slave_FLAG	I2C 从端模式
	I2C_Master_Slave_FLAG_EEPROM	I2C 从端模式与 EEPROM 烧录
	I2C_Master_Slave_INT	I2C 主端与从端模式各一组互传
I2S	I2S_DMA	I2S 从端模式与 DMA 搬移
	I2S_INT	I2S 从端模式与 DMA 中断
	I2S_POLLING	I2S 从端模式
IWDT	IWDT	看门狗设定范例
PWM	PWM	PWM 脉波调变范例
RTC	RTC_1sec	RTC 计时器设定 1 秒范例
	RTC_Alarm	RTC 计时器设定闹钟范例
SPI	MSPI_DMA_FLAG	SPI 使用 DMA 传输范例
	MSPI_DMA_INT	SPI 使用 DMA 传输与中断范例
	MSPI_FLAG	SPI 传输范例
	MSPI_FLAG_FLASH_MX25L4006	SPI 搭配 MX25L4006 传输范例
	MSPI_INT	SPI 传输与中断范例
TIMER	TMR_Capture_Mode	Timer 捕捉模式范例
	TMR_Compare_Mode	Timer 比较模式范例
	TMR_Counter_Mode	Timer 计数模式范例
	TMR_DMA_Mode	Timer 搭配 DMA 使用范例
	TMR_PWM_MODE	Timer 输出 PWM 使用范例
	TMR_Timer_Mode	Timer 普通计时范例
UART	UART_DMA	串列传输搭配 DMA 使用范例
	UART_HalfDuplexMode	串列传输使用半双工范例
	UART_InterruptAndFlagManage	串列传输使用中断范例
	UART_IrDA_Mode	串列传输使用 IRDA 范例
	UART_TxRx	串列传输同时发射与接收范例
USB	USB_HID	HID KEYBOARD 简易范例
	USB_HID_AUDIO_WM8731	HID 搭配 I2S 拨放 WM8731 音乐
WWDT	WWDT	视窗型看门狗

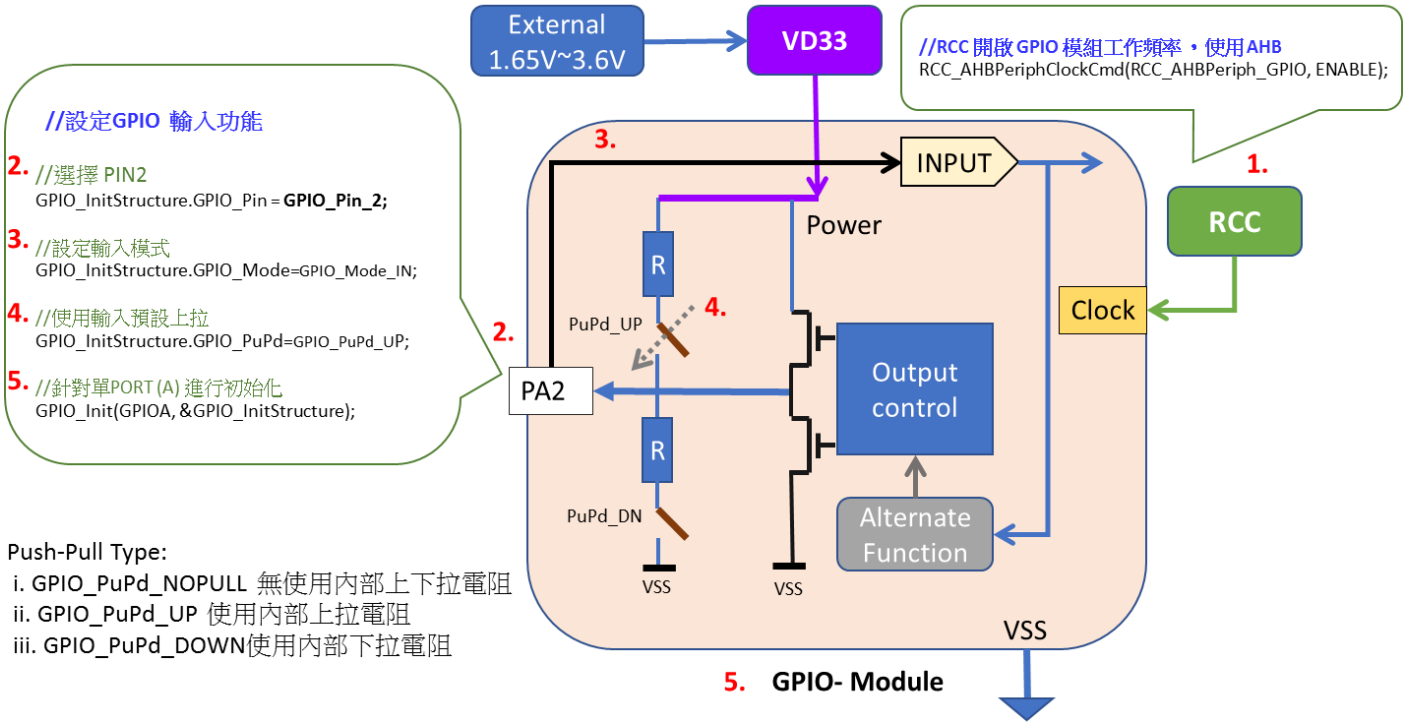
4. GPIO 功能说明

使用下列图示说明，GPIO 使用 PA2 做输入，使用 PC4~PC7 做输出，动作流程如下:

4.1 MCU 进行 GPIO 初始化

使用 PA2 内容如下，可参考周边程式库 wt32l0xx_pl_gpio.c 之函式 InitialGpio ( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 GPIO 使用，如下图步骤 1.
(Step 2) 设定 GPIO，此处范例选择 PIN2，如下图步骤 2.
(Step 3) 设定输入或输出模式，如下范例 IO 选择 INPUT，如下图步骤 3.
(Step 4) 设定上拉或下拉阻抗，如下范例 IO 选择上拉，如下图步骤 4.
(Step 5)设定 GPIO 之 Port-A 模组初始化，并写入暂存器，如下图步骤 5.



4.2 读取 GPIO 输入值

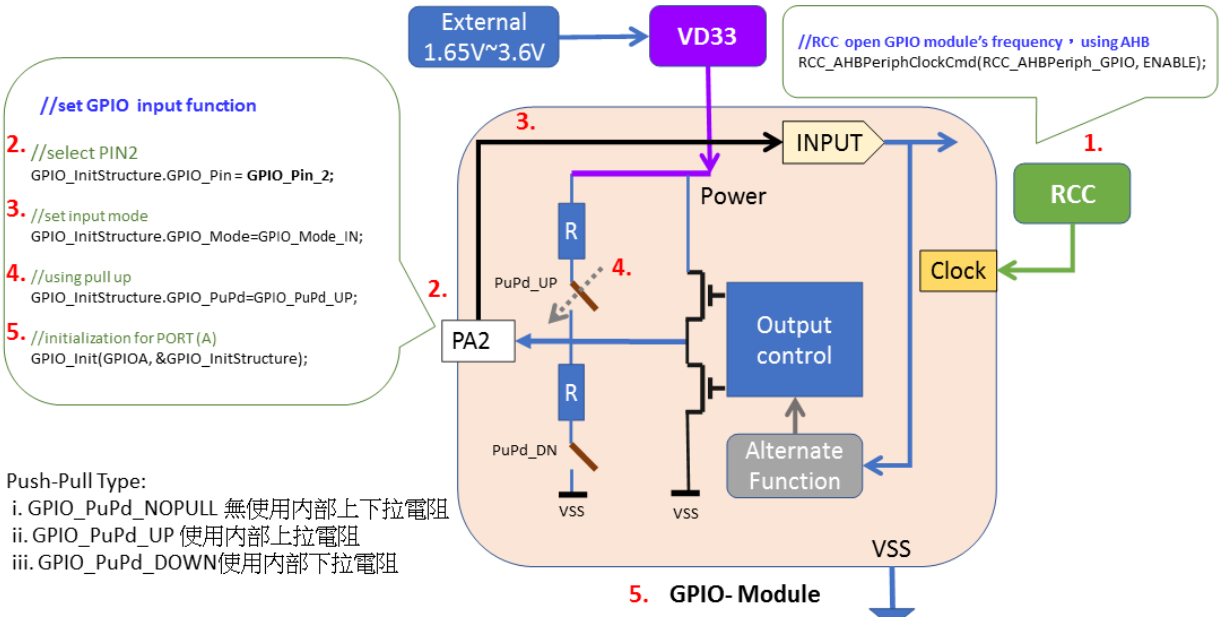
使用 GPIO_ReadInputDataBit()读取 BIT 资料值，例如当 PA2=LO 时，执行输入的写法如下

```
if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_2) == 0) {
//…… 写入对应功能
}
```

4.3 设定 GPIO 输出值

MCU 上电后初始化 PC4~PC7，其做法内容如下，或可参考范例函式 InitialGpio ( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 GPIO 使用，如下图步骤 1.
(Step 2) 设定 GPIO，可选择 PIN4，如下图步骤 2.
(Step 3) 设定输入或输出模式，如下范例 IO 选择 OUTPUT，如下图步骤 3.
(Step 4) 设定输出模式，有推挽式与开汲极，下例选择开汲极，如下图步骤 4.
(Step 5) 设定上拉或下拉阻抗，如下范例 IO 选择无上拉，如下图步骤 5.
(Step 6) 设定 GPIO 之 Port-C 模组初始化，并写入暂存器，如下图步骤 6.



4.4 范例程式 gpio

参考 wt32l0xx_pl_gpio.c 之函式 InitialGpio( )下列程式为参照上述 1~6 步骤依序执行

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

  //RCC 开启 GPIO 模组工作频率 ，使用 AHB
1 RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIO, ENABLE);

  // set General GPIO pin INPUT
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;        //设定输入模式
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;        //使用输入预设上拉，但省电模式会耗电
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;           //选择 PIN2
  GPIO_Init(GPIOA, &GPIO_InitStructure);              //针对单PORT (A-D) 进行初始化

#if(ENABLE_LED_BLINK==ON)       //判断是否需输出灯号
  // set General GPIO pin PC4
```

GPIO_InitTypeDef
参考 CMSIS 定义

```
2.  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;      //选择PIN4
3.  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;       //设定输出模式
4.  GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;      //设定开汲极类型
5.  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;    //设定无上、下拉

6.  GPIO_Init(GPIOC, &GPIO_InitStructure);              //针对单PORT-A 进行初始化
    GPIO_SetBits(GPIOC, GPIO_Pin_4);                    //设定 PORTC PIN4 输出 HI
     //.....省略
  #endif
```

GPIO_SetBits 输出 HI 电位
GPIO_ResetBits 输出 LO 电位

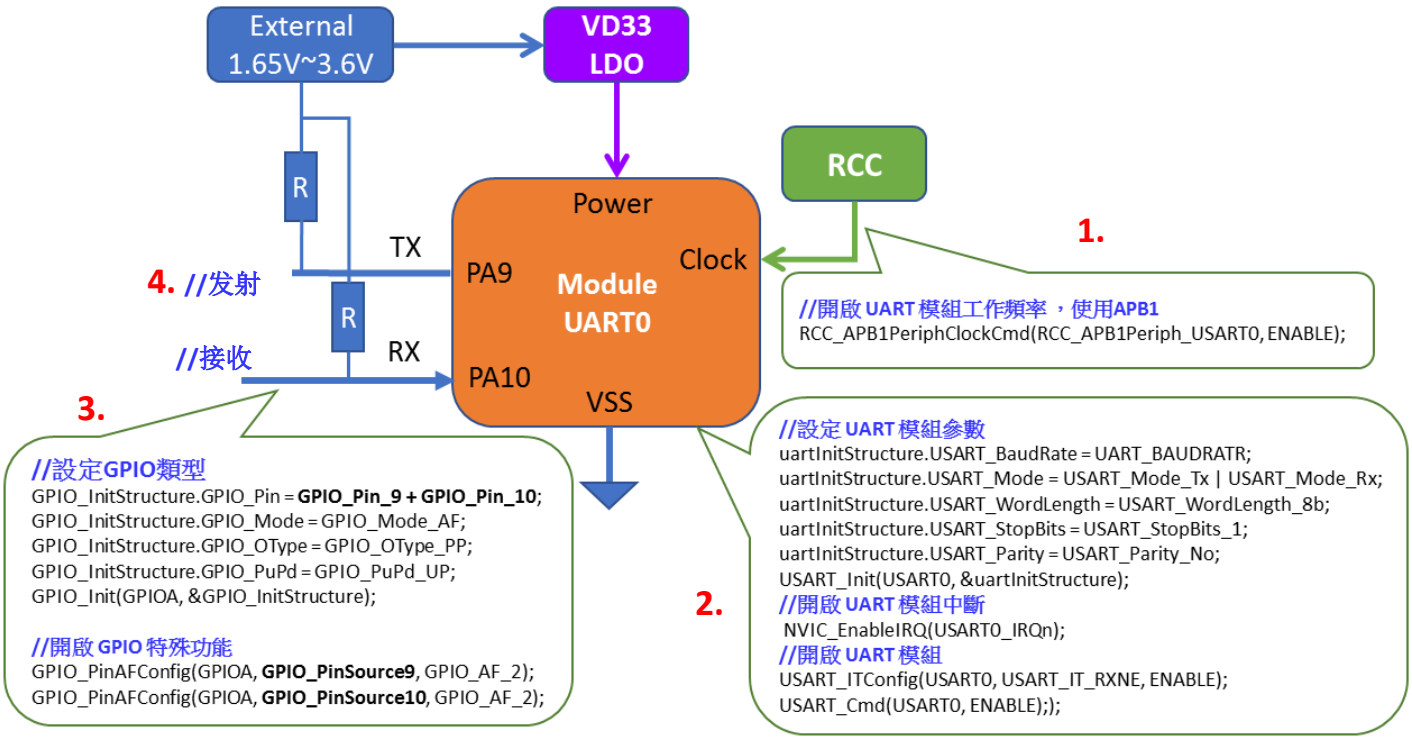
5. UART 功能说明

使用下列图示说明，使用 UART0 或 UART1 执行资料传输，动作流程如下:

5.1 MCU 上电后初始化 UART

如下列 1~4 步骤,可参考周边程式库 wt32l0xx_pl_uart.c 使用函式 InitialUart0( )或 InitialUart1( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 UART 使用，如下图步骤 1.
(Step 2) 设定 UART 模组参数，如下图步骤 2.
(Step 3) 设定 GPIO 类型 (IO 最后设定，避免信号灌入状态未定的模组) ，如下图步骤 3.
(Step 4) 发射 UART 资料，如下图步骤 4.



5.2 范例程式 uart

参考 wt32l0xx_pl_uart.c 之函式 InitialUart0 ( )，下列程式为参照上述 1~4 步骤依序执行

```
USART_InitTypeDef uartInitStructure;          //初始化使用，结构宣告
GPIO_InitTypeDef  GPIO_InitStructure;         //初始化使用，结构宣告
USART_DeInit(USART0);                         //清除UART0 的初始化使
```

GPIO_InitTypeDef、USART_InitTypeDef 参考 CMSIS 定义

1.
```
//开启 UART 模组工作频率 ，使用APB1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART0, ENABLE);        //输入 APB1 时脉
```

2.

```
//设定 UART 模组参数
uartInitStructure.USART_BaudRate = UART_BAUDRATR;                //设定 Baud Rate
uartInitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;    //设定TX +RX 功能启用
uartInitStructure.USART_WordLength = USART_WordLength_8b;        //设定 传输长度 8bit
uartInitStructure.USART_StopBits = USART_StopBits_1;             //设定1个停止位元
uartInitStructure.USART_Parity = USART_Parity_No;                //设定是否使用Parity位元
#if(ENABLE_OVER_SAMPLE8==ON)
USART_OverSampling8Cmd(USART0, ENABLE);                  //是否使用Over Sampling 加速
#endif
USART_Init(USART0, &uartInitStructure);                  //进行 UART0 初始化

#if(ENABLE_INT_UART0==ON)
USART_ITConfig(USART0, USART_IT_RXNE, ENABLE);           //设定 UART0 中断类型
NVIC_EnableIRQ(USART0_IRQn);                             //启动 UART0 中断功能
#endif
USART_Cmd(USART0, ENABLE);                               //启动 UART0 模组功能
```

3.

```
//设定GPIO类型
#if(SELECT_UART0_CH_A==ON)                               //若选择 A 通道
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 + GPIO_Pin_10;  //选定 GPIO 脚位
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;             //使用 AF 类型
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;           //选定 GPIO 推挽式或 开汲极
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;             //选定 GPIO 上拉或下拉类型
GPIO_Init(GPIOA, &GPIO_InitStructure);                   //进行 GPIO 初始化

//开启 GPIO 特殊功能
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_2);     //选定AF对应功能，有0~5种类
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_2);    //选定AF对应功能，有0~5种类
```

4.

```
//发射
printf("CH-A,Baud=%d,", UART_BAUDRATR);                  //输出 UART0 资料
#else                                                    //若选择 B 通道
..... 省略
#endif
```

5.3 UART 进行 RX 接收资料与 TX 发射资料

UART 中断功能搭配使用 UART0_Handler( ) 进行 RX 接收资料，写法如下

```
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
}
```

发射资料可搭配 fputc( )使用 ARM 预设 printf( ) ，或是使用范例 DRV_Printf( )输出资料

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

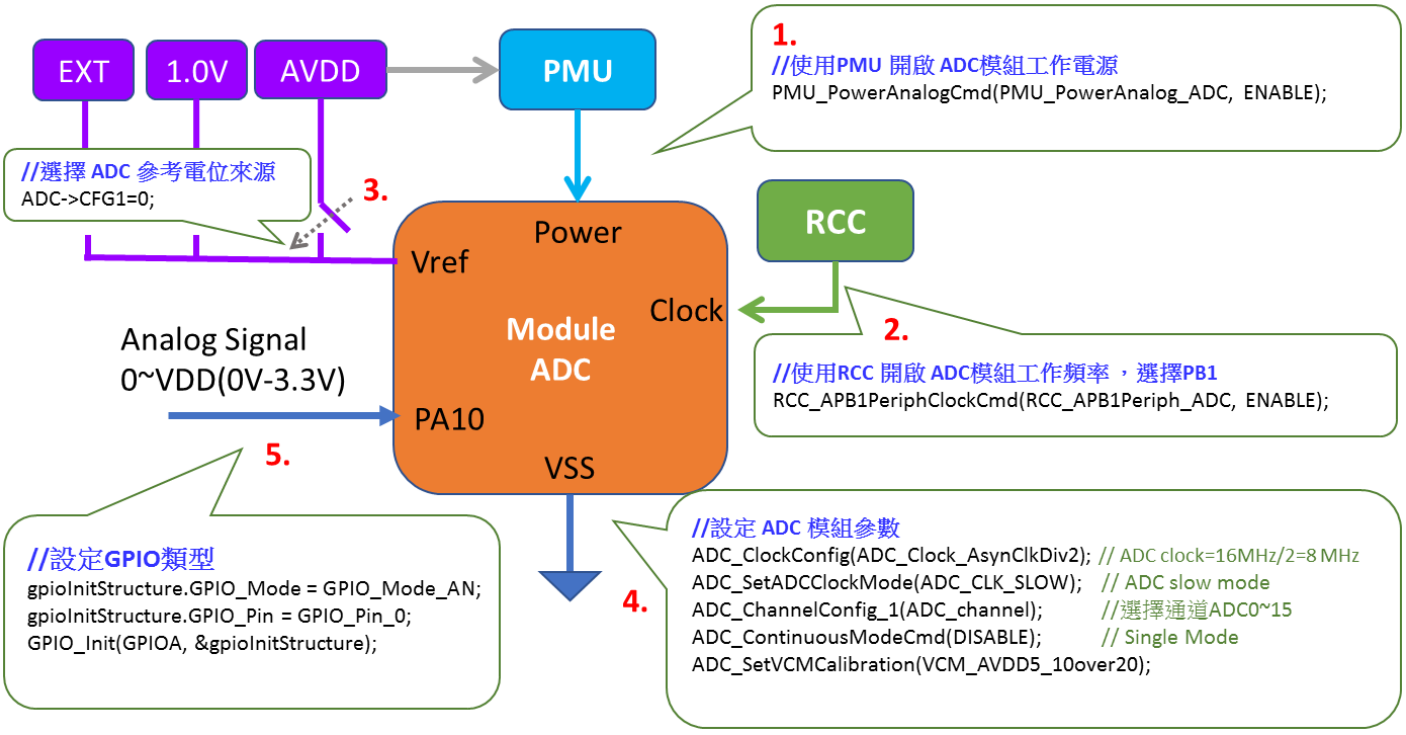
6. ADC 功能说明

使用下列图示说明，使用 ADC 执行类比信号输入，动作流程如下:

6.1 MCU 进行 ADC 初始化

MCU 上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_adc.c 使用函式 InitialAdc( )

(Step 1) 设定 PMU(电源管理单元) 开启类比电源提供给 ADC 使用，如下图步骤 1.
(Step 2) 设定 RCC (时钟控制模组) 开启时脉提供给 ADC 使用，如下图步骤 2.
(Step 3) 选择参考电位来源，有 AVDD、B-GAP 1.2V、External Pin 输入，如下图步骤 3.
(Step 4) 设定 ADC 模组参数，设定转换通道、速度，如下图步骤 4.
(Step 5) 设定 GPIO 类型 (IO 最后设定，避免信号灌入状态未定的模组)，如下图步骤 5.



6.2 范例程式 adc

参考 wt32l0xx_pl_adc.c 之函式 InitialAdc ( )下列程式为参照上述 1.~5.步骤依序执行

```
void InitialAdc(uint16_t ADC_channel)
{
  GPIO_InitTypeDef        gpioInitStructure;       // IO 初始化使用，变数结构

1.  //---------- PMU -----------------------------------------
  // 使用PMU 开启 ADC模组工作电源
  PMU_PowerAnalogCmd(PMU_PowerAnalog_ADC, ENABLE);        // 供电给 ADC
```

2.
```
//---------- RCC ----------------------------------------
  // 使用RCC 开启 ADC 模组工作频率 ，选择APB1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_ADC, ENABLE);

//---------- ADC ----------------------------------------
ADC_StopOfConversion_1();//先停止ADC转换，若之前有开启
```

3.
```
#if(ADC_VREF_SEL_AVDD==ON)            //选择 ADC 参考电位来源 AVDD、1.2V、EXT(外部)
  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);                 // 选择通道 ADC0~15
  ADC_ContinuousModeCmd(DISABLE);                   // Single Mode
  ADC_SetVCMCalibration(VCM_AVDD5_10over20);        // 若AVDD小于1.8V需调高 VCM(Common-Mode
Voltage)

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

#if(ENABLE_HW_ADC_AWD==ON)      //若开启使用AWD 类比看门狗
  //…… 省略
#endif

#if(ENABLE_FUNC_DMA==OFF)             //若无使用DMA搬运，则开启中断并判断转换完成
  ADC->CFG1 |= 0x00000200;      // Enable ADC  interrupt
  ADC_ITConfig(ADC_IT_EOC, ENABLE);
  NVIC_EnableIRQ(ADC_IRQn);   // ADC  interrupt enable
#endif
```

5.
```
//-------------------------------------------------
// ADC 通道设定，依其PA~PC分不同进行 IO/Analog 切换
  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 其1通道，设定 ADC
    }
  //·····以下IO设定省略
  }
```

6.3 进行 ADC 侦测与转换资料

范例程式如下:

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

    ADC_StopOfConversion_1();           //先停止ADC 转换
    ADC_ChannelConfig_1(ADC_channel);   // 选择 ADC_通道，channel enable
    __nop();  __nop();  __nop();  __nop();
    gu16AdcFinish = 0;                  //清除转换旗标

    ADC_StartOfConversion_1();          //启动 ADC 转换

    while (gu16AdcFinish == 0);         //等待ADC 转换完成 旗标 设立
    AD_buff = ADC_GetConversionValue(); // 取出 ADC 转换 数值
    return AD_buff;
}
```

此行函式为写入暂存器命令:
ADC->ADCCR |= (uint32_t)ADC_START;

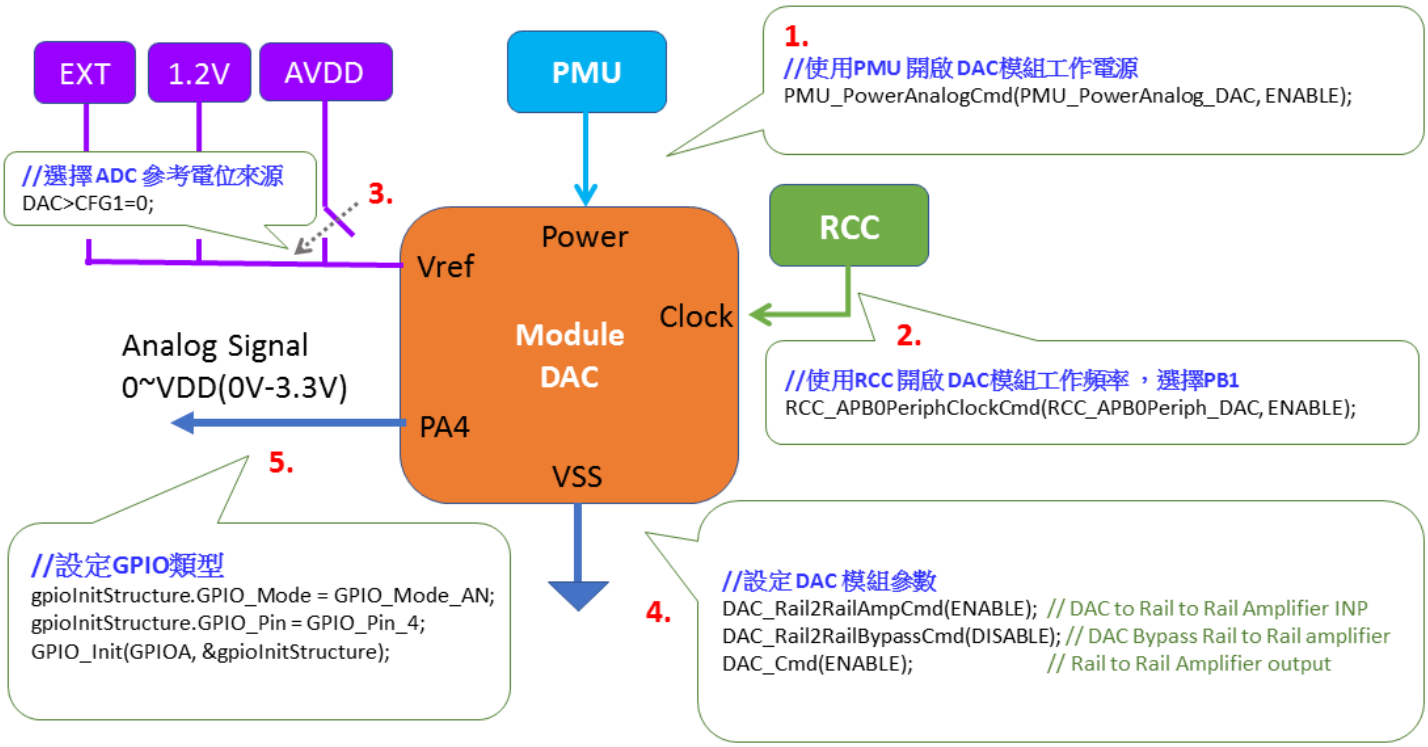
7. DAC 功能说明

使用下列图示说明，使用 DAC 执行类比信号输入，动作流程如下:

7.1 MCU 进行 DAC 初始化

上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_dac.c 使用函式 InitialDac( )

(Step 1) 设定 PMU(电源管理单元) 开启类比电源提供给 DAC 使用，如下图步骤 1.
(Step 2) 设定 RCC (时钟控制模组) 开启时脉提供给 DAC 使用，如下图步骤 2.
(Step 3) 选择参考电位来源，有 AVDD、B-GAP 1.2V、External Pin 输入，如下图步骤 3.
(Step 4) 设定 DAC 模组参数，设定转换通道、速度，如下图步骤 4.
(Step 5) 设定 GPIO 类型 (IO 最后设定，避免信号灌入状态未定的模组)，如下图步骤 5.



7.2 范例程式 dac

参考 wt32l0xx_pl_dac.c 之函式 InitialDac( )，下列程式为参照上述 1~5 步骤依序执行:

```
void InitialDac(uint16_t DAC_channel)
{
 //----------- PMU ----------------------------------------
 // 使用PMU 开启 DAC 模组工作电源
 PMU_PowerAnalogCmd(PMU_PowerAnalog_DAC, ENABLE);   //供给电源
```

1.

```
//----------- RCC ----------------------------------------
// 使用RCC 开启 DAC 模组工作频率 , 选择APB1
RCC_APB0PeriphClockCmd(RCC_APB0Periph_DAC, ENABLE);     //供给频率
```

2.

```
//----------- DAC ----------------------------------------
DAC_DeInit();       //先清除 DAC 旧设定
```

3.

```
#if(DAC_VREF_SEL_1P2==ON)
 DAC->CFG &= ~BIT2;        //使用 BG1P0V 作为 参考电源

#elif(DAC_VREF_SEL_EXT==ON)
 DAC->CFG &= ~BIT3;        //使用 外部IO 作为 参考电源

 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); //使用 AVDD 作为 参考电源
#endif
```

4.

```
DAC_Rail2RailAmpCmd(ENABLE);            // DAC to Rail to Rail Amplifier INP
DAC_Rail2RailBypassCmd(DISABLE);    // DAC Bypass Rail to Rail amplifier
DAC_Cmd(ENABLE);                    // 开启输出, Rail to Rail Amplifier
```

5.

```
//----------  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);
}
```

7.3 进行 DAC 资料转换输出

范例程式如下:

```
uint32_t DAC_Convert(uint16_t DAC_channel,uint32_t u32DacOut)
{
 DAC_SetInputData(u32DacOut);   //输出类比信号 DAC->CVTD

 return    u32DacOut;
}
```

此行函式为写入暂存器命令:
DAC->CVTD = Data;

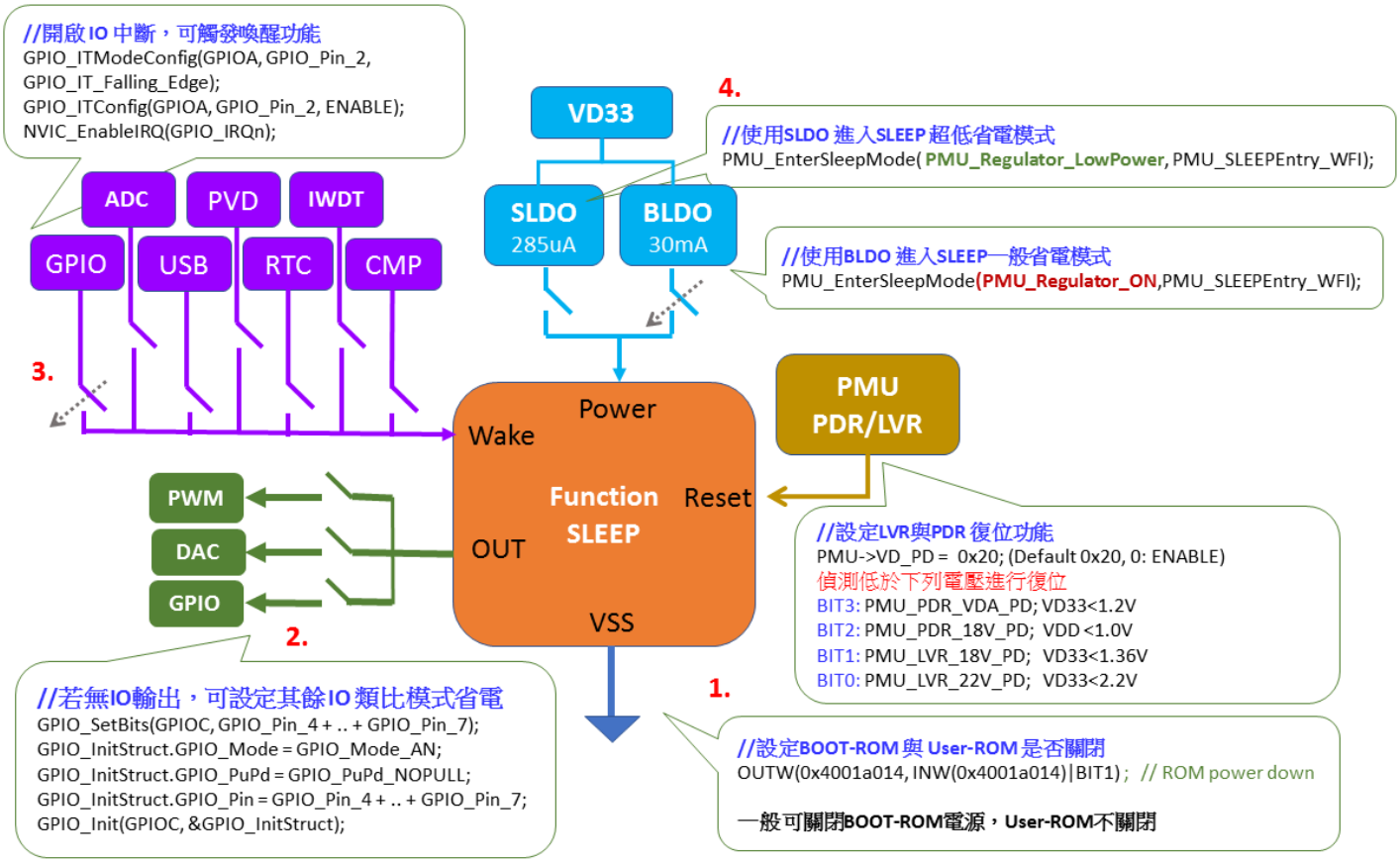
8. SLEEP 功能说明

使用下列图示说明，使用 SLEEP 进入省电模式，动作流程如下:

8.1 MCU 进行 SLEEP 初始化

上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_save.c 使用函式 save( )

(Step 1) 设定 Boot-ROM 电源关闭，进入 SLEEP 时无使用 ISP 功能，如下图步骤 1.
(Step 2) 设定 GPIO 类型，无使用 IO 设成类比型态(Analog Mode)，如下图步骤 2.
(Step 3) 设定 GPIO 唤醒，所有的 GPIO 都可设定触发唤醒 SLEEP，如下图步骤 3.
(Step 4) 进入 SLEEP 模式，可依耗电情况选用低功耗与一般省电，如下图步骤 4.



8.2 范例程式 save.c

参考 wt32l0xx_pl_save.c 之函式 save()，下列程式为参照上述 1~4 步骤依序执行

```
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)
2.  GPIO_InitTypeDef  GPIO_InitStruct;
  GPIO_SetBits(GPIOC, GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7);      //将 LED 熄灭
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                 //使用类比模式 可以省电
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;       //输入模式下使用 pull-up 会增加耗电
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;     //PIN 选用
  GPIO_Init(GPIOC, &GPIO_InitStruct);                                          //进行IO 初始
#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)    //使用 PA2 做唤醒 IO 使用
3.  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)     //使用 PC9 做唤醒 IO 使用
//……省略
#endif
#endif

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

#if(ENABLE_WAKEUP_IWDT==ON)      //省电时仍开启IWDT
  PMU_StopModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
  PMU_StandbyModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
  IWDT_ReloadCounter();
#endif

  //----------------- Sleep ----------------------------------
#if(ENABLE_SLEEP_MODE==ON)       //systick 必须关闭否则会唤醒
if (nMode == SAVE_MODE_SLEEP)
{

#if(ENABLE_FUNC_SYSTICK==ON)
```

```
        SysTick->CTRL = 0;
#endif

//进入 SLEEP 模式
  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.

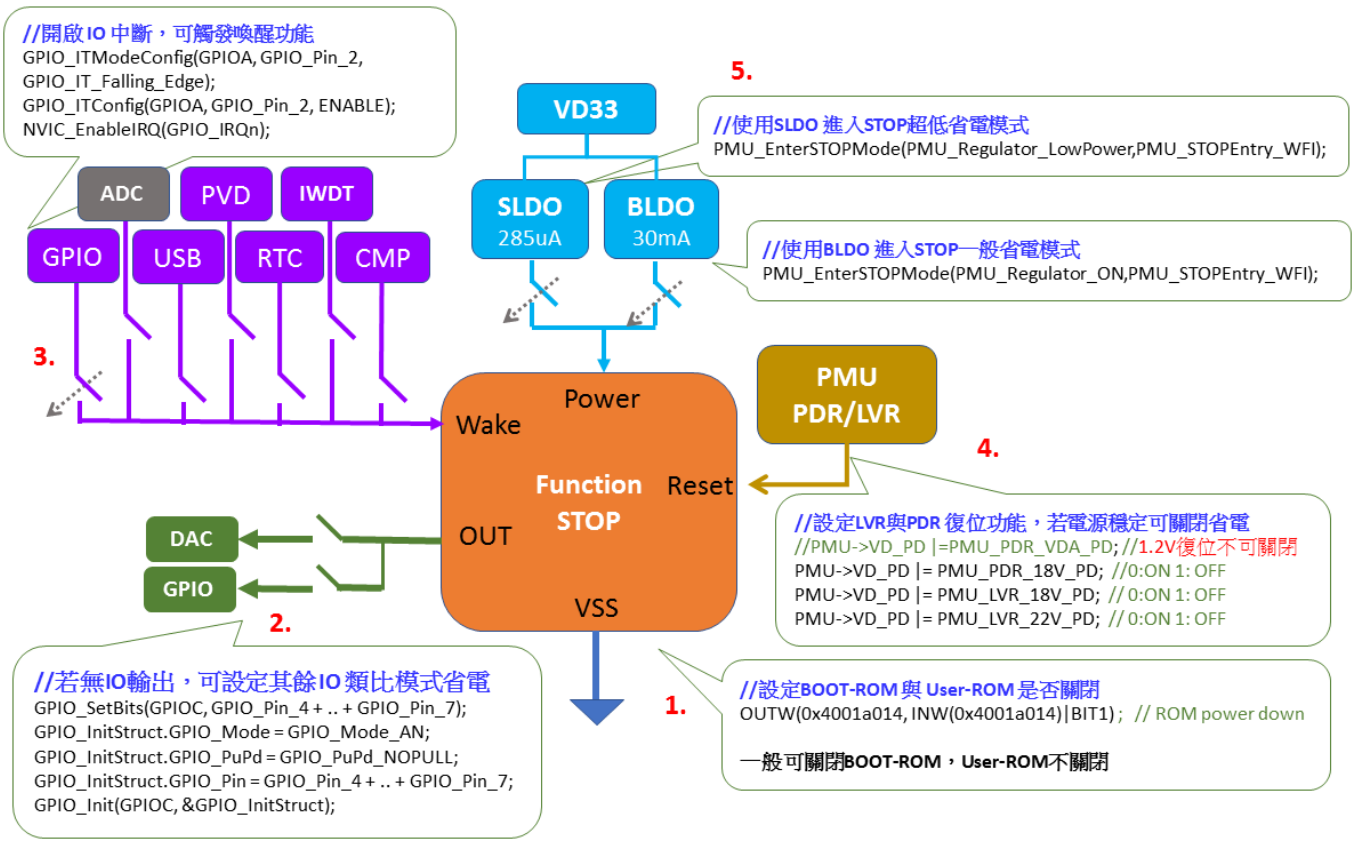
9. STOP 功能说明

使用下列图示说明，使用 STOP 进入省电模式，动作流程如下:

9.1 MCU 进行 STOP 初始化

上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_save.c 使用函式 save( )

(Step 1) 设定 Boot-ROM 电源关闭，进入 STOP 时无使用 ISP 功能，如下图步骤 1.
(Step 2) 设定 GPIO 类型，无使用 IO 设成类比型态(Analog Mode)，如下图步骤 2.
(Step 3) 设定 GPIO 唤醒，所有的 GPIO 都可设定触发唤醒 STOP，如下图步骤 3.
(Step 4) 设定 PDR/LVR 复位，若电源稳定可将 LVR 关闭省电，PDR 建议开启，如下图步骤 4.
(Step 5) 进入 STOP 模式，可依耗电情况选用低功耗与一般省电，如下图步骤 5.



9.2 范例程式 save

参考 wt32l0xx_pl_save.c 之函式 save()，下列程式为参照上述 1.~5.步骤依序执行

```
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)
2.  GPIO_InitTypeDef  GPIO_InitStruct;
    GPIO_SetBits(GPIOC, GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7);     //将 LED 熄灭
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                 //使用类比模式 可以省电
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;      //输入模式下使用 pull-up 会增加耗电
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;    //PIN 选用
    GPIO_Init(GPIOC, &GPIO_InitStruct);                                          //进行IO 初始
#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)     //使用 PA2 做唤醒 IO 使用
3.  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)       //使用 PC9 做唤醒 IO 使用
//……省略
#endif
#endif

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

#if(ENABLE_WAKEUP_IWDT==ON)        //省电时仍开启IWDT
    PMU_StopModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    PMU_StandbyModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    IWDT_ReloadCounter();
#endif


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

```
    //……省略
#endif


//----------------- STOP -------------------------------------
     #if(ENABLE_STOP_MODE==ON)
    if (nMode == SAVE_MODE_STOP)
    {
        //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_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);          //自动关闭 DAC 模组耗电
     #endif

     #if(ENABLE_FUNC_ADC==ON)
        PMU->ATPD_STOP &= (~PMU_StopModeAutoPower_ADC);          //自动关闭 ADC 模组耗电
     #endif

     #if(ENABLE_FUNC_LSI==ON)
        PMU->ATPD_STOP &= (~PMU_STOP_LSI_PD);                    //自动关闭 LSI 模组耗电
     #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);    //自动关闭 COMP 模组耗电
     #endif

//进入 STOP 模式
    //PMU_EnterSTOPMode(PMU_Regulator_ON,PMU_STOPEntry_WFI);      //BLDO=ON
     PMU_EnterTOPMode(PMU_Regulator_LowPower,PMU_STOPEntry_WFI); //BLDO=OFF

    }
#endif
```

4.

5.

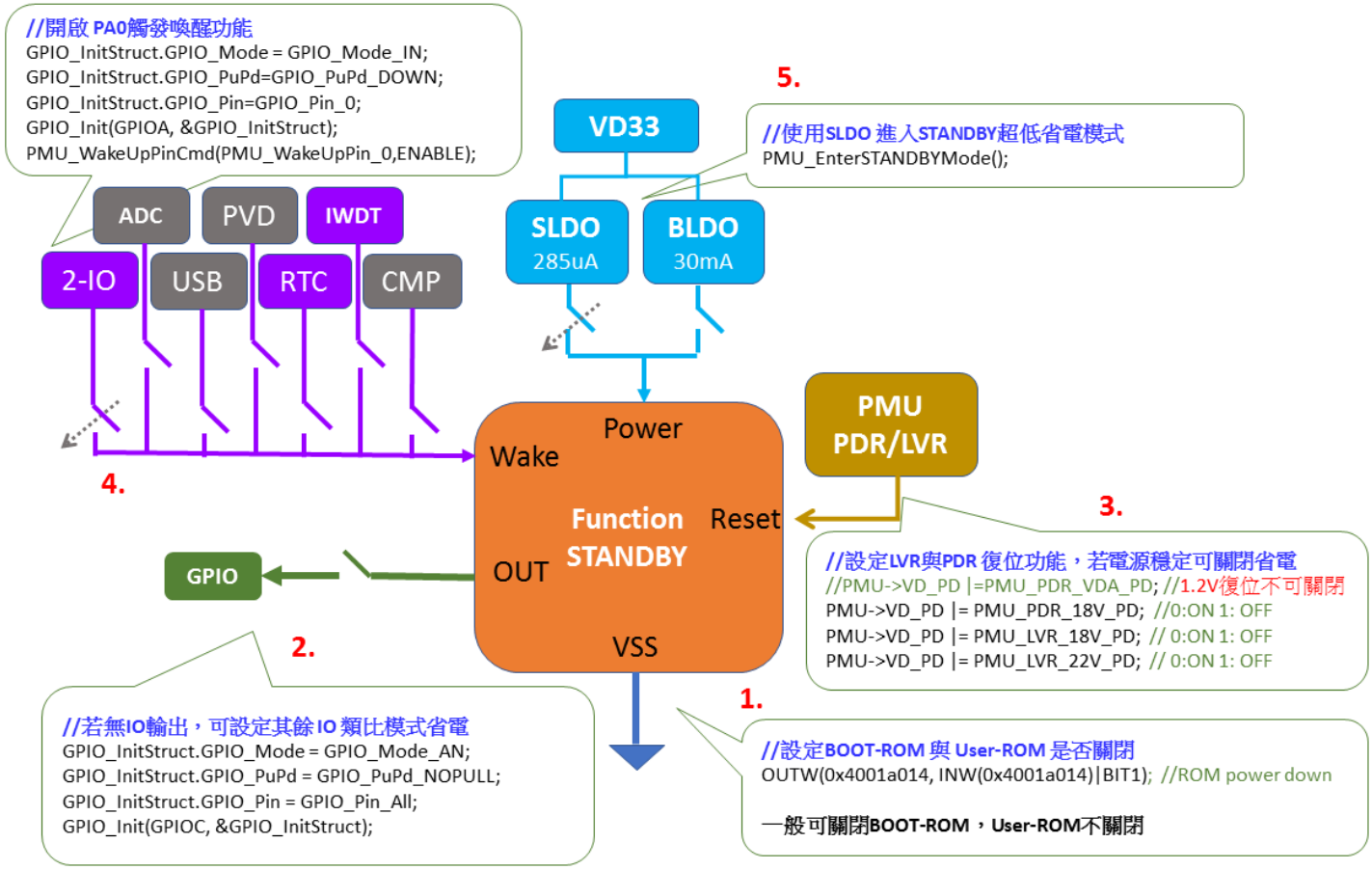
10. STANDBY 功能说明

使用下列图示说明，使用 STANDBY 进入省电模式，动作流程如下:

10.1 MCU 进行 STANDBY 初始化

上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_save.c 使用函式 save( )

(Step 1) 设定 Boot-ROM 电源关闭，进入 STANDBY 时无使用 ISP 功能，如下图步骤 1.
(Step 2) 设定 GPIO 类型，无使用 IO 设成类比型态(Analog Mode)，如下图步骤 2.
(Step 3) 设定 GPIO 唤醒，有两组 GPIO 都可设定触发唤醒 STANDBY，如下图步骤 3.
(Step 4) 设定 PDR/LVR 复位，若电源稳定可将 LVR 关闭省电，PDR 建议开启，如下图步骤 4.
(Step 5) 进入 STANDBY 模式，可依耗电情况选用低功耗与一般省电，如下图步骤 5.



10.2 范例程式 save

参考 wt32l0xx_pl_save.c 之函式 save()，下列程式为参照上述 1.~5.步骤依序执行

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

2.

```
//---------------- Close IO Pull-up-----------------------
#if(ENABLE_LED_BLINK==ON)
    GPIO_InitTypeDef  GPIO_InitStruct;
    GPIO_SetBits(GPIOC, GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7);      //将 LED 熄灭
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                 //使用类比模式 可以省电
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;       //输入模式下使用 pull-up 会增加耗电
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;     //PIN 选用
    GPIO_Init(GPIOC, &GPIO_InitStruct);                                              //进行IO 初始
#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)          //使用 PA2 做唤醒 IO 使用
    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) //使用 PC9 做唤醒 IO 使用
  //……省略
  #endif
#endif

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

#if(ENABLE_WAKEUP_IWDT==ON)          //省电时仍开启IWDT
    PMU_StopModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    PMU_StandbyModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
    IWDT_ReloadCounter();
#endif


//----------------- Sleep ----------------------------------
#if(ENABLE_SLEEP_MODE==ON)
    //……省略
#endif

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

```
//----------------- STANDBY -----------------------------------
#if(ENABLE_STANDBY_MODE==ON)
if(nMode==SAVE_MODE_STANDBY)
{
    // -----------  Reset Power ------------------------
3.  //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)
4.  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;
    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.  //进入 STANDBY 模式
    PMU_EnterSTANDBYMode();

}
#endif
```

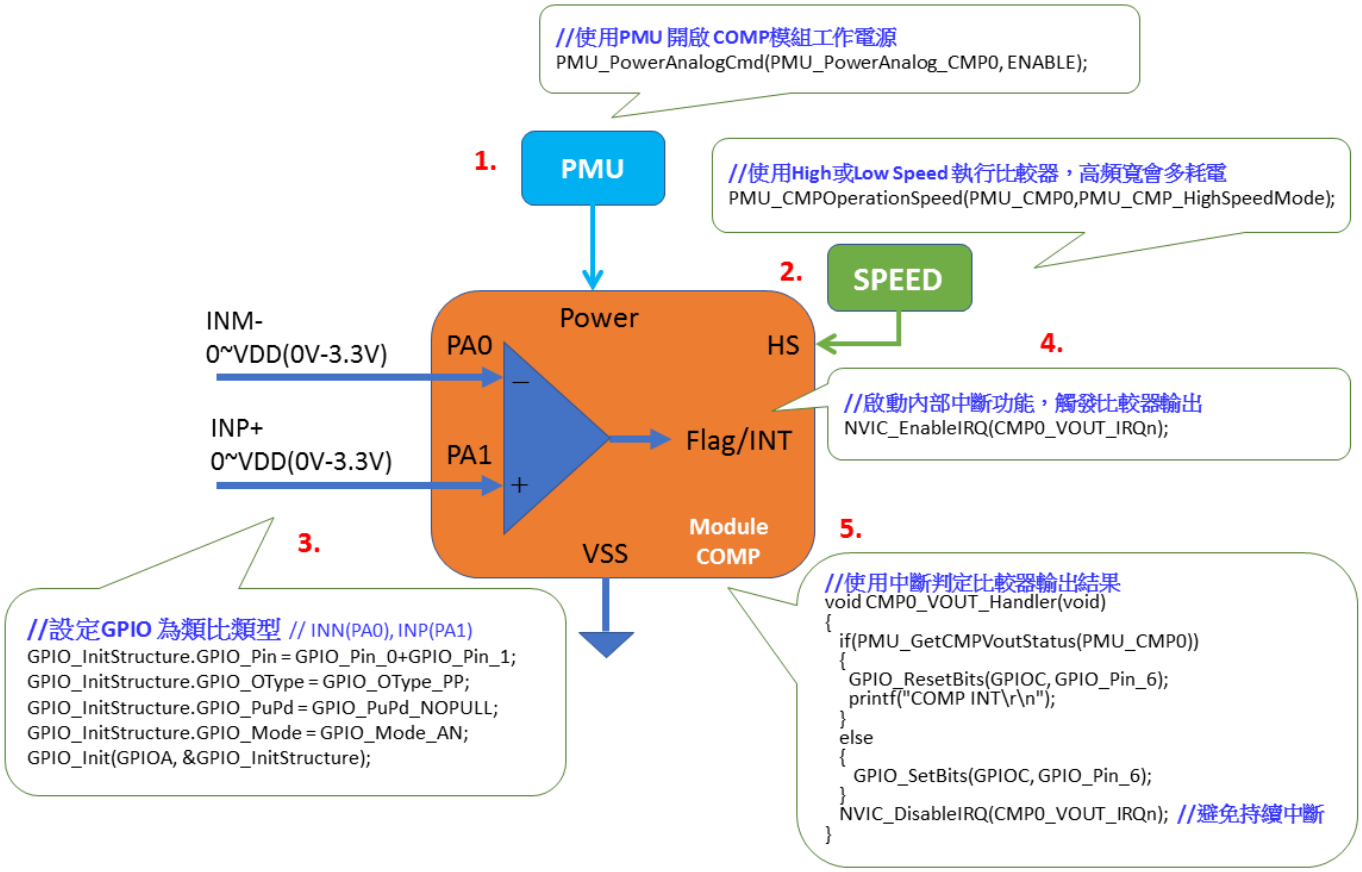
11. COMPARATOR 功能说明

使用下列图示说明，使用比较器(COMP)执行类比信号输入，动作流程如下:

11.1 MCU 进行 Comparator 初始化

上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_comp.c 使用函式 InitialComp ( )

(Step 1) 设定 PMU(电源管理单元) 开启类比电源提供给 COMP 使用，如下图步骤 1.
(Step 2) 设定 RCC (时钟控制模组) 开启时脉提供给 COMP 使用，如下图步骤 2.
(Step 3) 设定 GPIO 类型 (IO 最后设定，避免信号灌入状态未定的模组)，如下图步骤 3.
(Step 4) 设定 COMP 模组中断功能，当输入电位 INP>INM 时触发，如下图步骤 4.
(Step 5) 当 INP>INM 时触发中断，输出的结果也可用 PMU_GetCMPVoutStatus()读出



11.2 范例程式 comp

参考 wt32l0xx_pl_comp.c 之函式 InitialComp ( )，参照上述 1.~5.步骤依序执行

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

1.  #if(ENABLE_HW_CMP0==ON)  //开启COMP_0
      //使用PMU 开启 COMP 模组工作电源
```

```
        PMU_PowerAnalogCmd(PMU_PowerAnalog_CMP0, ENABLE);


        //使用High或Low Speed 执行比较器，高频宽增加耗电
2.  #if(ENABLE_HW_CMP_SPEED_HI==ON)
        PMU_CMPOperationSpeed(PMU_CMP0, PMU_CMP_HighSpeedMode);
    #else
        PMU_CMPOperationSpeed(PMU_CMP0, PMU_CMP_LowSpeedMode);
    #endif

        // 设定GPIO 为类比类型  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)  //开启COMP_1
    //……省略
    #endif
}
```

11.3 Comparator 之中断功能

范例程式 comp.c 之中断函式 CMP0_VOUT_Handler ( )，可对照上述 e.描述

```
    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，避免持续中断

    }
```

进入中断后，再读取比较器结果

12. FLASH 读写功能说明

使用下列图示说明，使用 IC 内部 FLASH 执行读写资料，一次完整写入与读出动作流程如下:

12.1 MCU 进行 FLASH 初始化

上电后要更新 FLASH 资料，须将资料抹除成 0xFF 后才能对其内容写入，可参考周边程式库使用函式 RunFlash ( )。

(Step 1) 解开 FLASH 保护锁，如下图步骤 1.
(Step 2) 设定预计写入的位址，并先清除该页资料大小为 1KB.，如下图步骤 2.
(Step 3) 写入 WORD 资料 ，使用 CMSIS 呼叫 FLASH_ProgramWord()，该函式使用 __IO 定址 ROM 空间，例如: *(__IO uint32_t *)Address = Data，如下图步骤 3.
(Step 4) 检查 WORD 资料 ，直接使用 __IO 定址 ROM 空间，例如: Data= *(__IO uint32_t *)Address ;，如下图步骤 4.
(Step 5) 将结果使用 UART 输出，如下图步骤 5.

参考 flash.c 内 RunFlash ()的写法步骤
1. flash.c->FLASH_Unlock ()
2. flash.c->FLASH_ErasePage ()
3. flash.c->FLASH_ProgramWord ()
4. data= *(__IO uint32_t*) Address;

12.2 范例程式 flash

参考 wt32l0xx_pl_flash.c 之函式 RunFlash ( )，参照上述 1.~5.步骤依序执行

```
void RunFlash(void)
{
```

```
#if(SYS_CLOCK_SEL!=CLK_MSI)
    FLASH_SetLatency(1);//若系统频率 >=16 Mhz     // Set latency
#endif
    FLASH_ClearFlag(FLASH_FLAG_EOP | /*FLASH_FLAG_PGERR |*/ FLASH_FLAG_WRPERR);
1.  //==================== Unlock  FLASH   ============
    FLASH_Unlock();//解除 FLASH 防写锁

    /* Define the number of page to be erased */
    TotalPages = (WRITE_END_ADDR - WRITE_START_ADDR + 1) / FLASH_PAGE_SIZE;

2.  //==================== Erase  FLASH   ============================//
    for (EraseCounter = 0; (EraseCounter < TotalPages) && (FLASHStatus == FLASH_COMPLETE);
EraseCounter++)
    {
        FLASHStatus = FLASH_ErasePage(WRITE_START_ADDR + (FLASH_PAGE_SIZE * EraseCounter));//页清除
        if (FLASHStatus != FLASH_COMPLETE) //若清除失败，输出数值并终止
        {
            uint16_t readout = *(__IO uint16_t*)(WRITE_START_ADDR + (FLASH_PAGE_SIZE * EraseCounter));
                printf("Page=0x%d,", START_ADDR_PAGE + EraseCounter);   //读值并显示
            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);


3.      //--------------- Program Flash Page-------------
        Address = WRITE_START_ADDR + u32TargetStartAddr;


        //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);//写入 WORD 资料
        }
```

1.Unlock 解锁

2. Page Erase 抹除

3. Program 写入

```
        u32TargetStartAddr += FLASH_PAGE_SIZE;
        u32TargetEndAddr += FLASH_PAGE_SIZE;
        Page++;              //页绝对位址
        PageCnt++;//页计数
    }
    if (FLASHStatus == FLASH_COMPLETE)       printf("Program Done\r\n");
    else                                     printf("Program Fail, Page=%d\r\n", Page - 1);


    //--------------- Test Lock (测试防写)---------
    //……省略


    //=================== Verify  FLASH  ============================//
4.  u32TargetStartAddr = 0;
    u32TargetEndAddr = FLASH_PAGE_SIZE - 1;

    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))             //检测之前写的数值 正确否 ?
            {
                MemoryProgramStatus = FAILED;
                printf("Page=%d,", Page);
                //……省略
                while (1);
            }
        }

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

        u32TargetStartAddr += FLASH_PAGE_SIZE;
        u32TargetEndAddr += FLASH_PAGE_SIZE;
        Page++;              //页绝对位址
        PageCnt++;  //页计数
    }


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

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

4. Verify 检查

5. Result输出结果

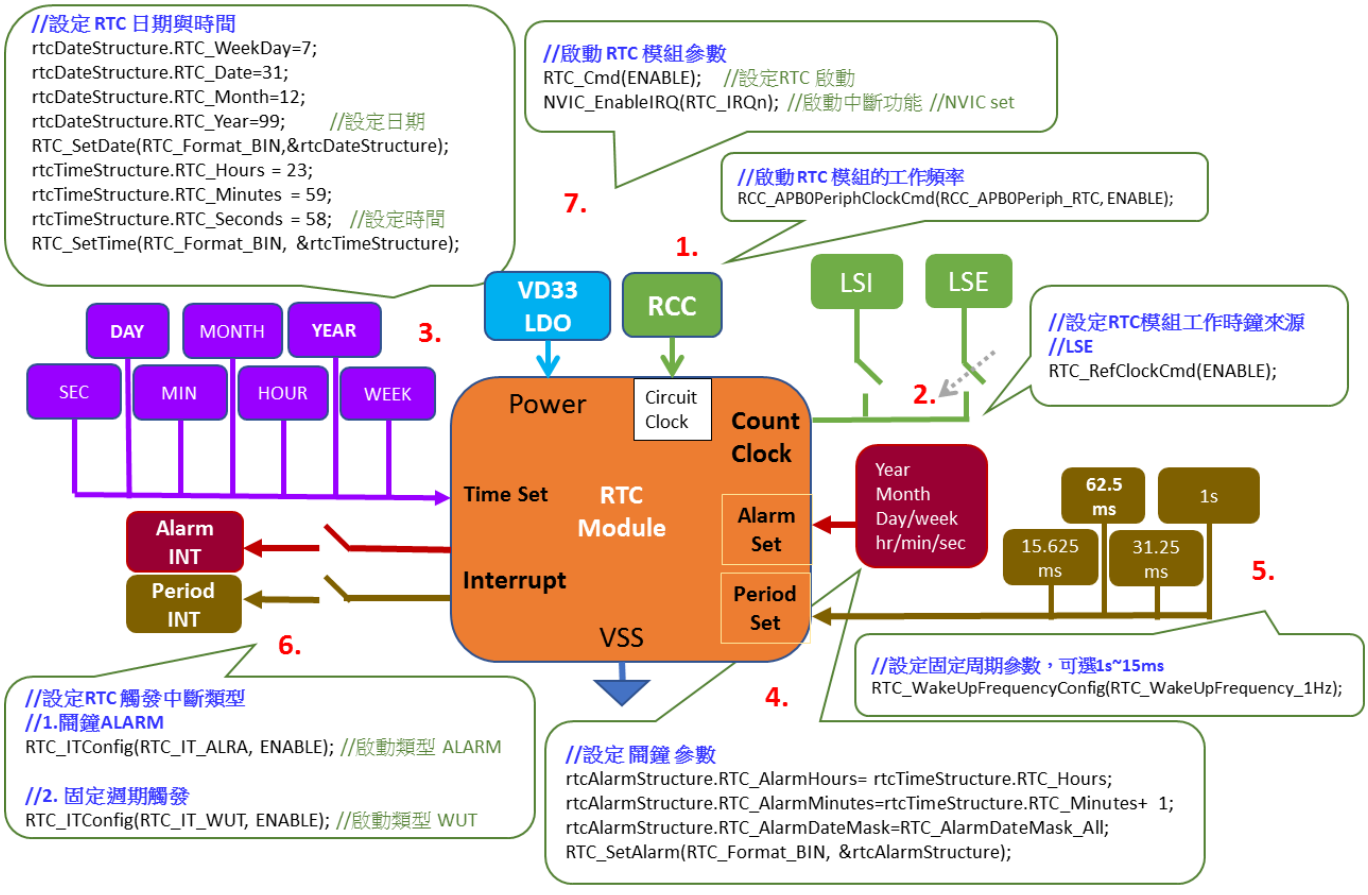
13. RTC 功能说明

使用下列图示说明，使用实时计数器(RTC)执行数位信号输入，动作流程如下:

13.1MCU 进行 RTC 初始化

上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_rtc.c 使用函式 InitialRtc ( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 RTC 使用，如下图步骤 1.
(Step 2) 选用参考钟源 LSI (37KHz)或 LSE (32.768KHz)，如下图步骤 2.
(Step 3) 设定 RTC 目前日期与时间，如下图步骤 3.
(Step 4) 设定闹钟时间当与目前时间相同时可触发 Alarm 中断，如下图步骤 4.
(Step 5) 设定周期时间可触发 WUT 中断，有 1sec~15msec 可以选，如下图步骤 5.
(Step 6) 设定中断开关，有 Alarm 与周期两种中断类型可选择，如下图步骤 6.
(Step 7) 启动 RTC 功能并开启 NVIC 中断总开关，如下图步骤 7.



13.2 范例程式 rtc

参考 wt32l0xx_pl_rtc.c 之函式 InitialRtc ( )，参照上述 1.~6.步骤依序执行

```
void InitialRtc(void)
{   RCC_APB0PeriphClockCmd(RCC_APB0Periph_RTC, ENABLE);     // 需先开启 APB0 clock 才可设定 RTC
    RTC_WriteReadProtectionCmd(DISABLE);  //RTC 保护开关，更改设定前要关闭
    RTC_DeInit();                                                          //清除 RTC 设定
    RTC_RefClockCmd(ENABLE);                                  //参考外部时钟源 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);//设定日期于 RTC 模组

    rtcTimeStructure.RTC_Hours = 23;
    rtcTimeStructure.RTC_Minutes = 59;
    rtcTimeStructure.RTC_Seconds = 58;
    RTC_SetTime(RTC_Format_BIN, &rtcTimeStructure);    //设定时间于 RTC 模组

    rtcLastTime.RTC_Hours = 0;          //测试记录用
    rtcLastTime.RTC_Minutes = 0;   //测试记录用
    rtcLastTime.RTC_Seconds = 0;   //测试记录用


    //------------------    RTC 闹钟 (ALARM)    ------------------------
    #if(ENABLE_FUNC_ALARM==ON)
        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); //启动中断子类型 ALARM
    #else
        RTC_WakeUpFrequencyConfig(RTC_WakeUpFrequency_1Hz);
        RTC_ITConfig(RTC_IT_WUT, ENABLE); //启动中断子类型 WUT，每(秒/ms)周期触发
    #endif

        RTC_Cmd(ENABLE);                                  //设定RTC 启动
        NVIC_EnableIRQ(RTC_IRQn);           //启动中断功能 //NVIC set

    }
```

13.3 设定 RTC 时间

当设定时间触发动作，范例程式 rtc.c 之中断函式 RTC_Handler ( )

```
void RTC_Handler(void)
{
    RTC_ClearITPendingBit(RTC_IT_ALRA + RTC_IT_WUT);   //清除硬体旗标
    RTC_ITConfig(RTC_IT_ALRA, DISABLE);// // 若无中断需求可关闭中断
    gbRtcInt = 1;   //变数设 1

//……可自行增加
}
```

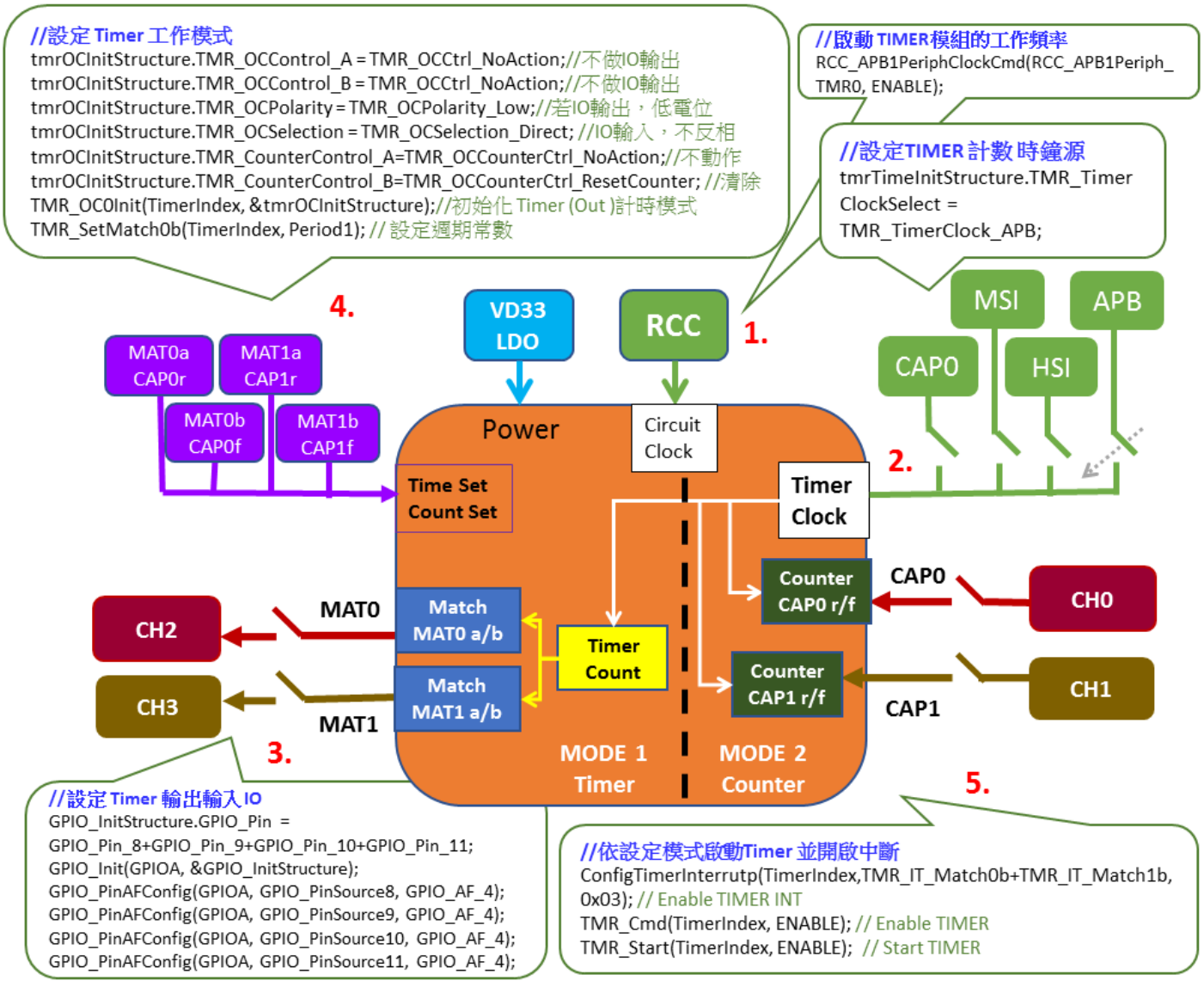
14. TIMER 功能说明

使用下列图示说明，使用计数计时器(TIMER)执行数位信号输入与输出，动作流程如下:

14.1 MCU 进行 Timer 初始化

上电后初始化其内容如下，可参考周边程式库 wt32l0xx_pl_timer.c 使用函式 ConfigTimerClockGpio ( )、 ConfigTimerTimeMode ( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 Timer 电路使用，如下图步骤 1.
(Step 2) 设定输入时钟来源，提供给 Timer 计算使用，如下图步骤 2.
(Step 3) 设定 GPIO 类型，1 组 Timer 输出有两路，输入有两路，如下图步骤 3
(Step 4) 设定 Timer 周期时间参数可触发中断与输出信号，如下图步骤 4.
(Step 5) 设定中断开关，并依设定参数、计时或计数模式开启 Timer，如下图步骤 5.



14.2 范例程式 timer

参考 wt32l0xx_pl_timer.c 之函式 ConfigTimerClockGpio ( )、ConfigTimerTimeMode( )，参照上述 1.~5.步骤依序执行

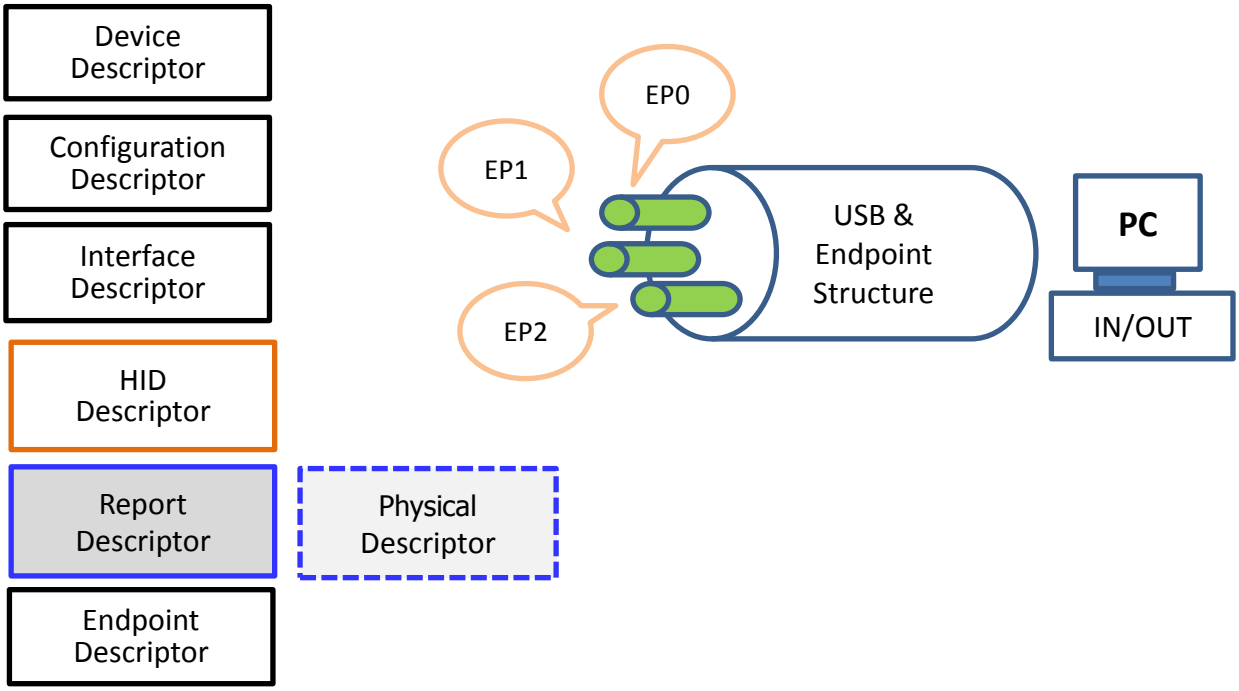
```
void ConfigTimerClockGpio(TMR_TypeDef* TimerIndex, uint32_t nPrescaler, uint16_t nChannelSetSel,
uint16_t nSource)
{
    TMR_TimerInitTypeDef          tmrTimeInitStructure;
    TMR_DeInit(TimerIndex);   //清除设定
    tmrTimeInitStructure.TMR_TimerClockSelect = nSource;    //频率来源选择 APB HSI MSI CAP0
    tmrTimeInitStructure.TMR_TimerPrescaler = nPrescaler;        // 除频 f'=1/ n+1

    //------------ 设定 Timer/GPIO -------------------------------
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIO, ENABLE); // 启动 GPIO 工作频率
    GPIO_InitTypeDef        GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;  //GPIO使用 AF 模式
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;

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

2.      TMR_TimerInit(TimerIndex, &tmrTimeInitStructure);//初始化Timer 时钟源、除频

        TMR_MatchInputSourceSwap(TMR0, DISABLE); //不交换IO

        if (nChannelSetSel == TMR_PIN_SET0)//使用第1组通道配置IO
        {
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)//使用第2 组通道配置IO
        {
            //.....省略
        }
    }
    else if (TimerIndex == TMR1)
    {
        //.....省略
    }
    else if (TimerIndex == TMR2)
    {
        //.....省略
    }
    TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_NoFilter); // 无使用数位滤波
    //TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_2clks);  // 使用数位滤波 2 clock
    //TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_4clks)); // 使用数位滤波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;//不做IO输出
    tmrOCInitStructure.TMR_OCControl_B = TMR_OCCtrl_NoAction;//不做IO输出

    tmrOCInitStructure.TMR_OCPolarity = TMR_OCPolarity_Low; //若IO输出，低电位
    tmrOCInitStructure.TMR_OCSelection = TMR_OCSelection_Direct; //IO输入，不反相
    tmrOCInitStructure.TMR_CounterControl_A = TMR_OCCounterCtrl_NoAction;  //Match后不动作
    tmrOCInitStructure.TMR_CounterControl_B = TMR_OCCounterCtrl_ResetCounter;   //最长 周期
    TMR_OC0Init(TimerIndex, &tmrOCInitStructure); //初始化 Timer (Out )计时模式
    TMR_SetMatch0b(TimerIndex, Period1); // 设定周期常数


    //------------- 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 周期
    TMR_OC1Init(TimerIndex, &tmrOCInitStructure);

    TMR_SetMatch1b(TimerIndex, Period2);          // 设定周期常数


    //----------- 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 与 HID 功能说明

15.1 USB-HID 架构说明

如同下图 USB 的阶层描述元(Descriptor)，其中于 Interface 描述元之后可增加 HID 的描述元与报告(Report)描述元，HID 的报告分类共有 3 种 Input、Output 与 Feature 可依需求增加，下图为标准 USB 与 HID 装置组态设定。



除了特殊物理装置需求否则一般 Physical Descriptor 不使用，而 HID 通讯需设定 Report Descriptor，该描述元有下列类型:

1. Input: 周边装置传输资料至电脑，使用 GET_REPORT 命令格式
2. Output: 电脑传输资料至周边装置，使用 SET_REPORT 命令格式
3. Feature: 周边装置与电脑进行资料交换，使用 GET_REPORT 与 SET_REPORT 命令格式

使用 USB-Endpoint 与 HID- Report 之间的对应关系:

- 使用 Feature 格式进行 HID 通讯是使用 Feature-Report 通过 EP0(Endpoint 0)进行 USB 资料交换，Feature 与 EP0 都是双向资料传输。
- 使用 Input、Output 格式进行 HID 通讯是使用 Input-Report、Output-Report 可选择通过 EP1~EP6 进行 USB 资料交换，Report 与 EP1~EP6 都是单向资料传输，例如 Input-Report 选定 EP2(IN 单向)，而 Output-Report 选定 EP1(OUT 单向)。

15.2 USB-HID 装置与组态描述元说明

范例程序提供有下列阵列参数供使用者可修改设定，自行定义 HID 传输通常需要修改的参数有 EP0_Packet_Size 、VENDOR_ID、PRODUCT_ID、与 endpoints 数量，如下列标示为红色字体部分，该范例使用 3 个 endpoint，分别为 EP0 作 Feature (双向)，EP1 作 Report-IN，EP2 作 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 the 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 the 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 报告描述元与用途页说明

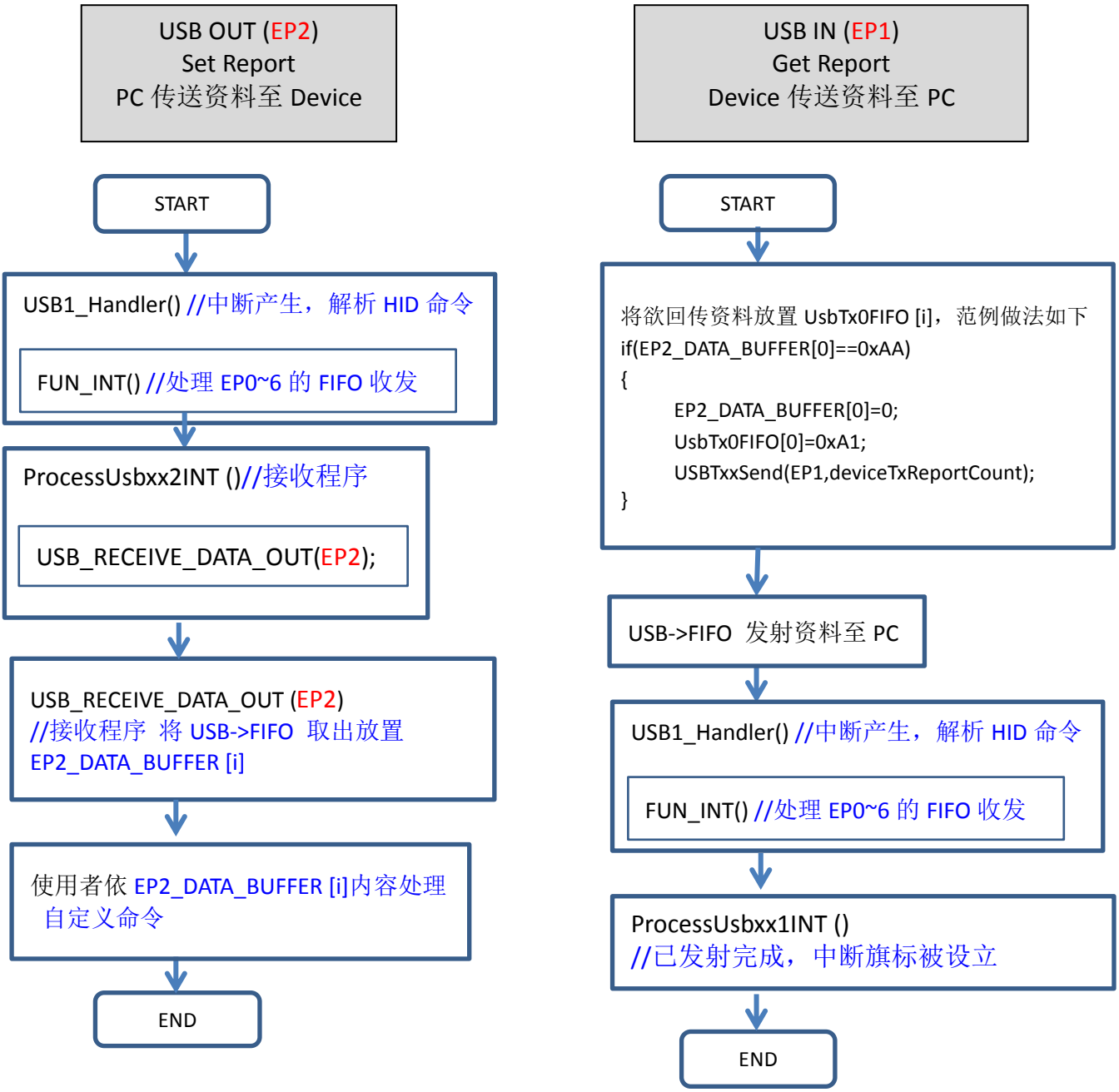
报告描述元其内容包含用途页(USAGE PAGE)主要设定自订的传输格式、长度与 Report ID，通常 1 组 Interface 配置 1 组 HID_ReportDescriptor，自行定义通常需要修改的参数有 deviceRxReportCount 、FEATURE、REPORT OUTPUT 与 REPORT INPUT 可依需求增列或删除，如下列标示为红色字体部分。

```
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 */
```

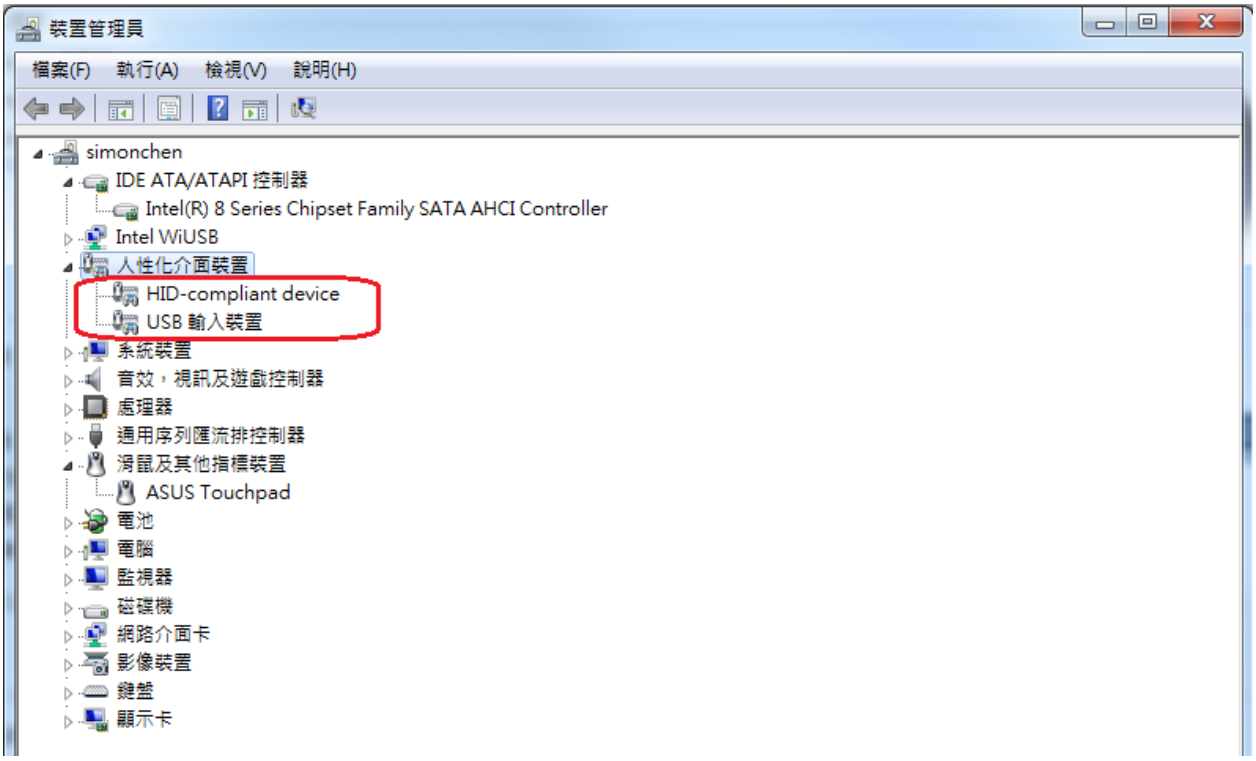
```
    0x91, 0x82,                 /* REPORT OUTPUT (Data,Var,Abs,Vol)    */
    /* 27 */

    /* 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 发射与接收流程

下列描述装置端(WT32L064) 于 Set Report、Get Report 作为收发 USB-HID 资料的流程，范例分别使用 EP2 与 EP1。



承上我们使用装置管理员侦测装置(WT32L064) 是否具备 USB-HID 功能，插入装置后会新增目标 USB 装置，该装置会列举在人性介面装置，如下图所示。



15.4.1 主机端发射与接收 HID Report 范例

如下表所列，该装置具有 3 个 endpoint，分别为 EP0 作 Feature (双向)，EP1 作 Report-IN，EP2 作 Report-OUT，接着我们设定主机端(PC) EP2 将发射的资料为 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

使用 PC 的 USB 软体工具，即时侦测 USB 的资料流向，当主机端发出 USB 资料后如下图所示 EP2 之 OUT，已传出 0xAA、0x22、0x00….0x00，然后主机端 EP1 会收取到 32 个 Byte 资料为装置回传 0xA1、0x00….0x00，此处结果与程式设定相同。

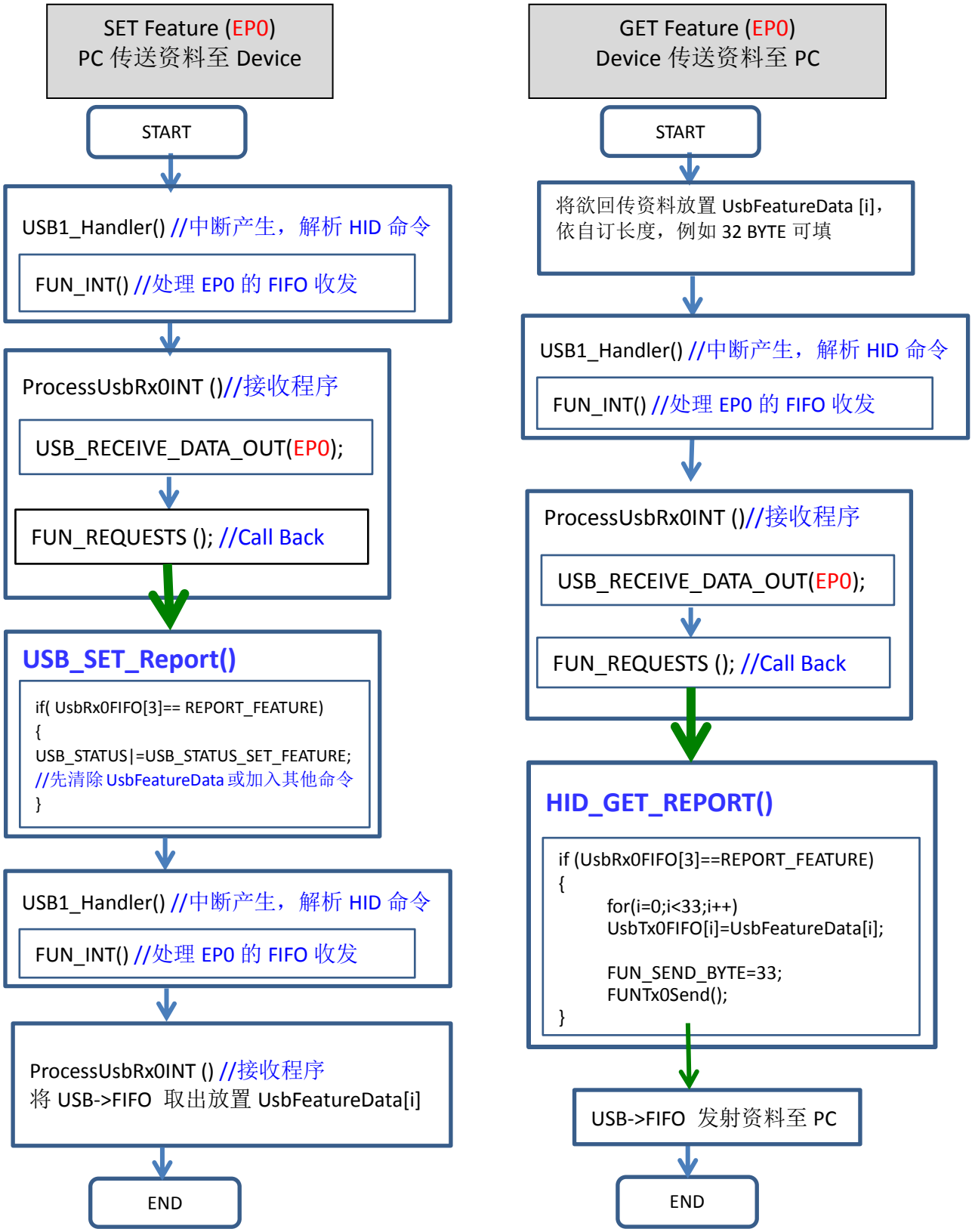
Endpoint	Direction	Data (16 进制)
2	OUT	AA 22 00 00_00 00 00 00_00 00 00 00_00 00 00 00 00 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00
1	IN	A1 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00 00 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00

程式设定当收取到 USB 指令 0xAA 时执行回传 USB 命令 0xA1，如下 main.c 程式片段:

```
if(EP2_DATA_BUFFER[0]==0xAA)
{
    EP2_DATA_BUFFER[0]=0;       //清除 Buffer
    UsbTx0FIFO[0]=0xA1;         //设定回传 0xA1
    USBTxxSend(EP1, deviceTxReportCount);     //执行 USB 的 EP1 发射 FIFO 资料
                                              // deviceTxReportCount=32
}
```

15.5 HID Feature 发射与接收流程

下列描述装置端于 SET Feature、GET Feature 作为收发 USB-HID 资料的流程，分别使用到 EP0。



15.5.1 HID Feature 接收范例

如下表所列，该装置具有 3 个 endpoint，分别为 EP0 可作 Feature 双向沟通，EP1 作 Report-IN，EP2 作 Report-OUT，此处 USB 通讯将 EP0 固定为控制型设定使用，而 SETUP 封包长度固定是 8 Bytes，我们设定主机端(PC) USB 工具之 EP0 将发射的资料为 0xA1、0x01、0x00、0x03、0x00、0x00、0x08、0x00，接着收到装置(WT32L064)的资料为 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

HID 命令可参考下列格式，其中 0xA1、0x01 为 GET REPORT 命令。

HID 格式	Request Type	bRequest	wValue		wIndex		wLength	
命令串	0xA0	0x01	0x00(L)	0x03(H)	0x00(L)	0x00(H)	0x08(L)	0x00(H)
命令 说明	**Get_Report** (Feature Input,使用 EP0)		Report ID=0	Report Type =Feature	Interface No. =0		长度= 8 Bytes	

使用 USB 软体工具，实际侦测 USB 的资料流向，当按下执行后如下表所示，我们传出 0xA1、0x01、0x00….0x00，收到了 8 个 Byte 资料为 0x01、0x02、0x03….0x08，此处结果与程式设定相同。

Endpoint	Direction	Data (16 进制)	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 发射范例

如下图所示，设定 PC 端 USB 工具之 EP0 将发射的资料为 0x21、0x09、0x00、0x03、0x00、0x00、0x08、0x00，接着发射资料为 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

HID 命令可参考下列格式，其中 0x21、0x09 为 SET REPORT 命令。

HID格式	Request Type	bRequest	wValue		wIndex		wLength	
命令串	0x21	0x09	0x00(L)	0x03(H)	0x00(L)	0x00(H)	0x08(L)	0x00(H)
命令说明	**Set_Report** (Feature output,使用EP0)		Report ID=0	Report Type =Feature	Interface No. =0		长度= 8 Bytes	

我们使用 USB 软体工具，实际侦测 USB 的资料流向，当按下执行后如下图所示，主机端传出 0xA1、0x01、0x00….0x00，接续发射 8 个 Byte 资料为 0x11、0x22、0x33….0x88，此处结果与程式设定相同。

Endpoint	Direction	Data (16 进制)	Description
0	CTL	21 09 00 03_00 00 08 00	SET REPORT
0	OUT	11 22 33 44_55 66 77 88	----

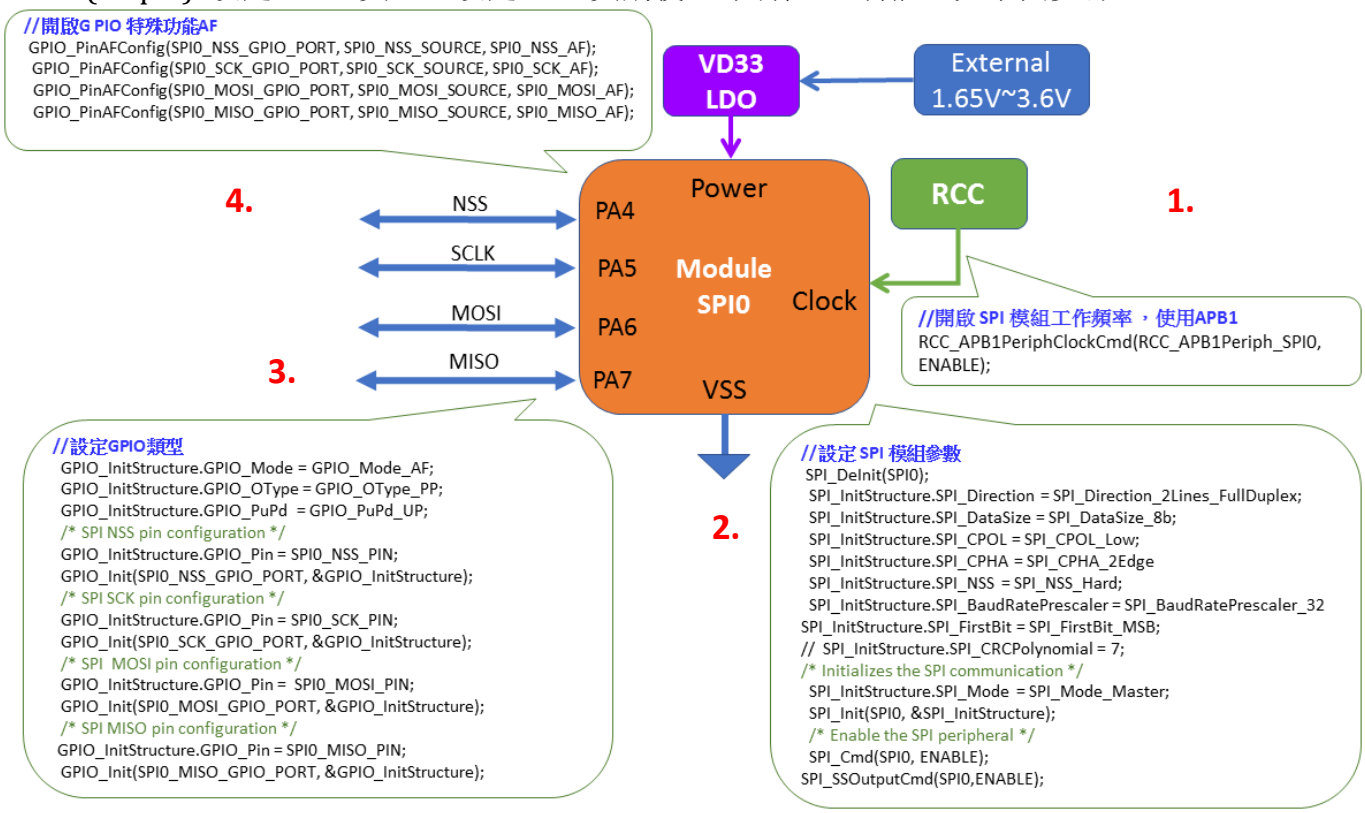
16. SPI 功能说明

使用下列图示说明，使用 SPI0 或 SPI1 执行资料传输，动作流程如下:

16.1 MCU 上电后初始化 SPI

如下列 1~4 步骤，可参考周边程式库使用函式 InitialSpi0( )或 InitialSpi1( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 SPI 使用，如下图步骤 1.
(Step 2) 设定 SPI 模组参数，如下图步骤 2.
(Step 3) 设定 GPIO 类型，设定推挽式与上拉电阻如下图步骤 3.
(Step 4) 设定 GPIO 类型，设定 AF3 类别使 IO 具有 SPI 功能，如下图步骤 4.



16.2 范例程式

参考 wt32l0xx_pl_spi.c 之函式 InitialSpi0 ( )，下列程式为参照上述 1~4 步骤依序执行

```
void InitialSpi0(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  /* Enable the SPI periph */
1.RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI0, ENABLE);

  /* SPI configuration -------------------------------------------------------*/
2.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);
```

3.

```
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);
```

4.

```
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);


}
```

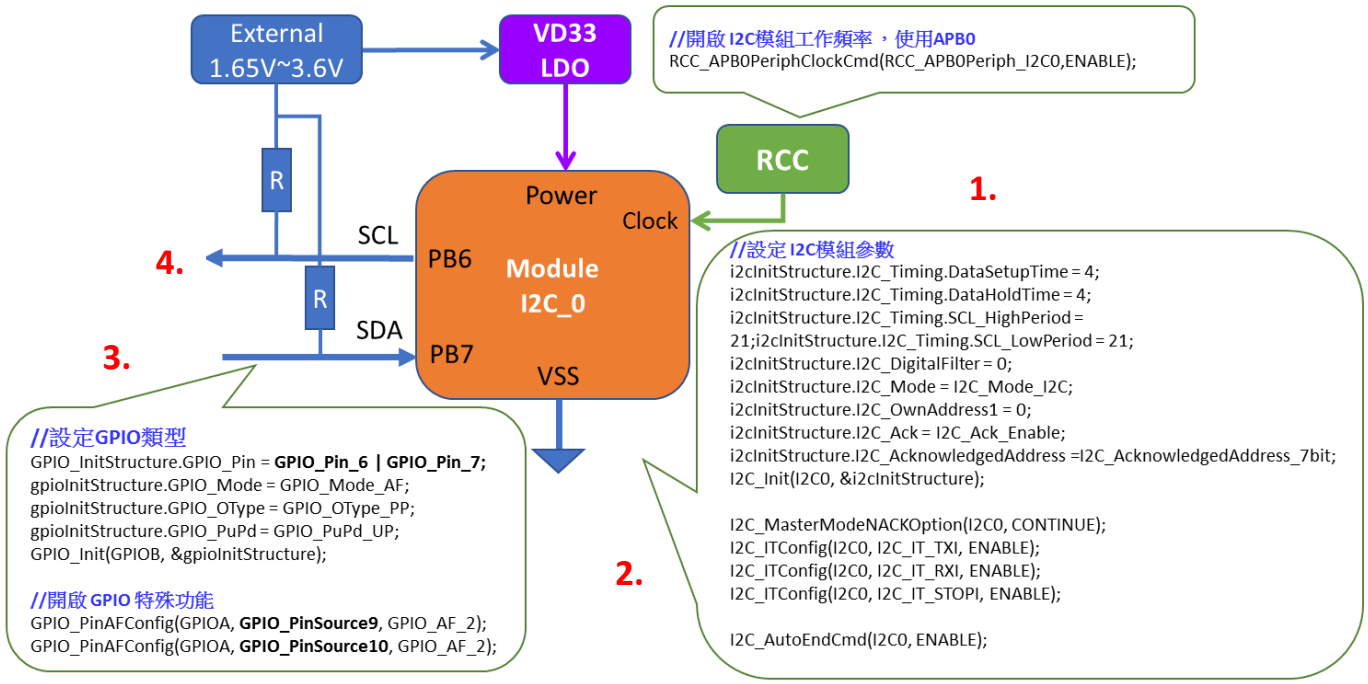
17. I2C 功能说明

使用下列图示说明，使用 I2C0 或 I2C1 执行资料传输，动作流程如下:

17.1 MCU 上电后初始化 I2C

如下列 1~4 步骤，可参考周边程式库使用函式 InitialI2c0( )或 InitialI2c1( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 I2C 使用，如下图步骤 1.
(Step 2) 设定 I2C 模组参数，如下图步骤 2.
(Step 3) 设定 GPIO 类型 (IO 最后设定，避免信号灌入状态未定的模组) ，如下图步骤 3.
(Step 4) 发射 I2C 资料，如下图步骤 4.



17.2 范例程式

```
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

    if (mode == I2C_MASTER)
    {    //Master
2.      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);
    }

3   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 进行 RX 接收资料 与 TX 发射资料

4.

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

  // --------------------------------
  // 主机发送资料给子机
  // --------------------------------
  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)));

  // --------------------------------
  // 主机接收资料于子机
  // --------------------------------
  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
}
```

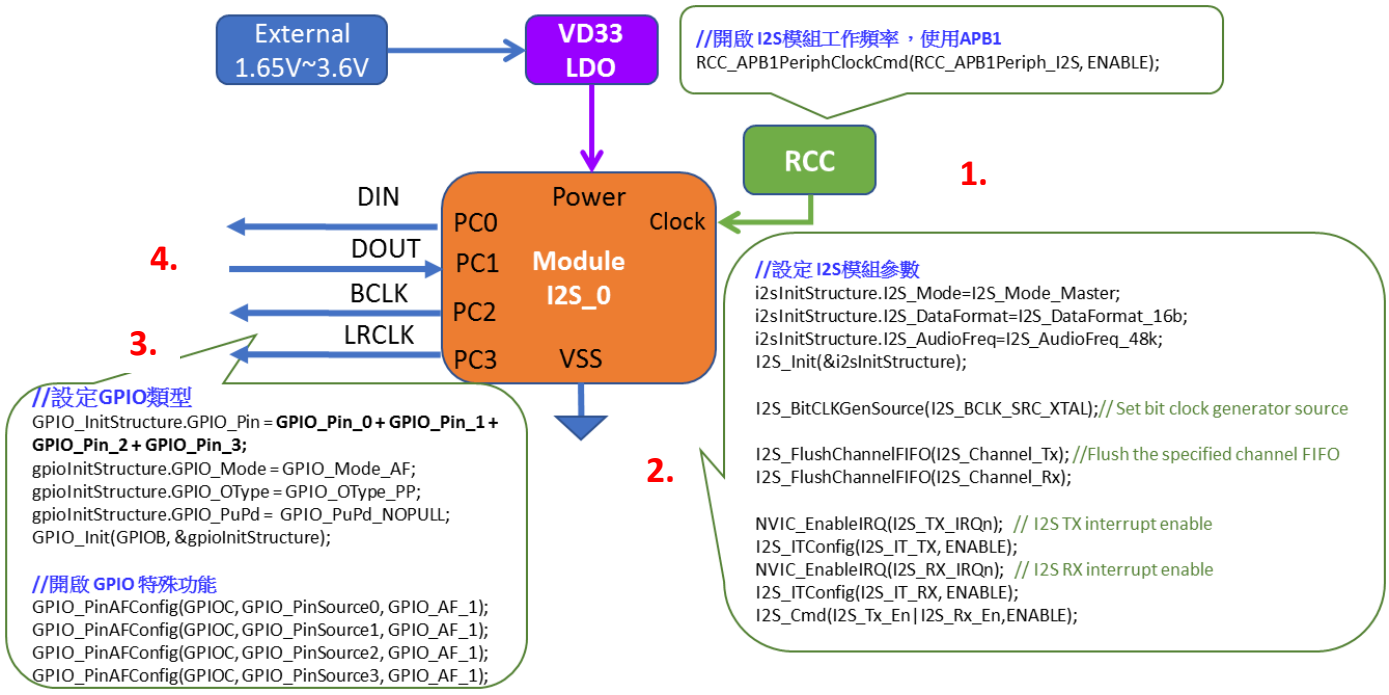
18. I2S 功能说明

使用下列图示说明，使用 I2S0 或 I2S1 执行资料传输，动作流程如下:

18.1 MCU 上电后初始化 I2S

如下列 1~4 步骤，可参考周边程式库使用函式 InitialI2s0 ( )或 InitialI2s1 ( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 I2S 使用，如下图步骤 1.
(Step 2) 设定 I2S 模组参数，如下图步骤 2.
(Step 3) 设定 GPIO 类型 (IO 最后设定) ，如下图步骤 3.
(Step 4) 发射 I2S 资料，如下图步骤 4.



18.2 范例程式

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

3.

4.

19. PWM 功能说明

使用下列图示说明，使用 PWM0A 或 PWM0B 执行宽度调变输出，动作流程如下:

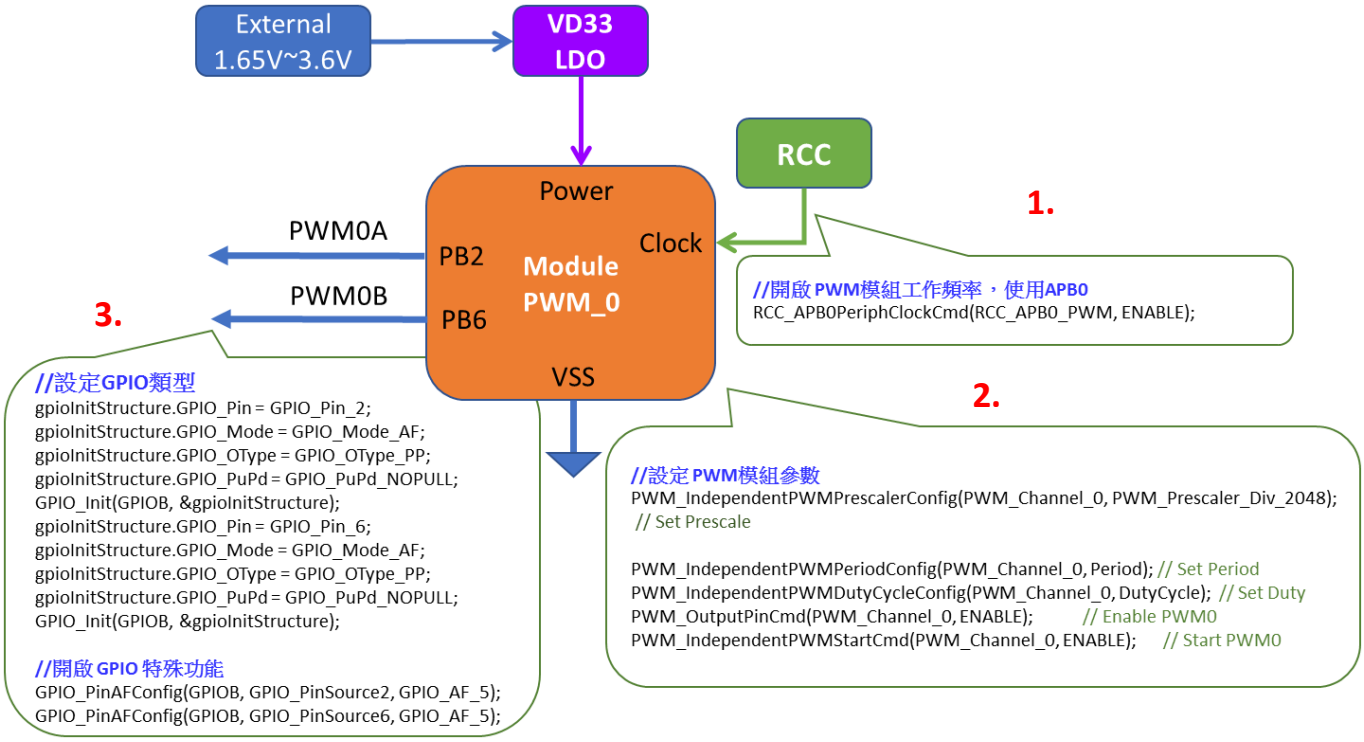
19.1 MCU 上电后初始化 PWM

如下列 1~4 步骤，可参考周边程式库使用函式 InitialPwm( )

(Step 1) 设定 RCC (时钟控制模组) 开启时脉提供给 PWM 使用，如下图步骤 1.

(Step 2) 设定 PWM 模组参数，如下图步骤 2.

(Step 3) 设定 GPIO 类型 (IO 最后设定，避免信号灌入状态未定的模组) ，如下图步骤 3.



19.2 范例程式

```
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.

```
    //设定GPIO类型
    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);
    //开启 GPIO 特殊功能
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource2, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_5);

}
```

20. DMA 功能说明

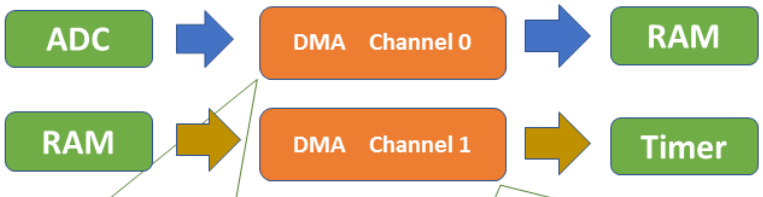
使用下列图示说明，使用 DMA 0 与 DMA1 通道执行资料传输，范例为读取 ADC 数值将资料搬运至 Timer 并输出周期波形，动作流程如下:

20.1 MCU 上电后初始化 DMA

如下列 1~2 步骤，可参考周边程式库使用函式 InitiDma ( )

(Step 1) 设定 DMA0 通道，将 ADC 资料透过 DMA0 传至 RAM 地址 **0x300000000** 如下步骤 1.

(Step 2) 设定 DMA1 通道，将 RAM 地址 **0x300000000** 资料透过 DMA1 传至 Timer2 如下步骤 2.



1.

```
//設定 DAM 0模組參數
DMA_DeInit(DMA_Channel0); //clear Register
DMA_StructInit(&DMA_0);    //Initialize Struct

DMA_0.DMA_SourceAddr=(uint32_t)ADC1_DR_ADDRESS; D
MA_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.

```
//設定 DAM 1 模組參數
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 范例程式

```
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);
}
```

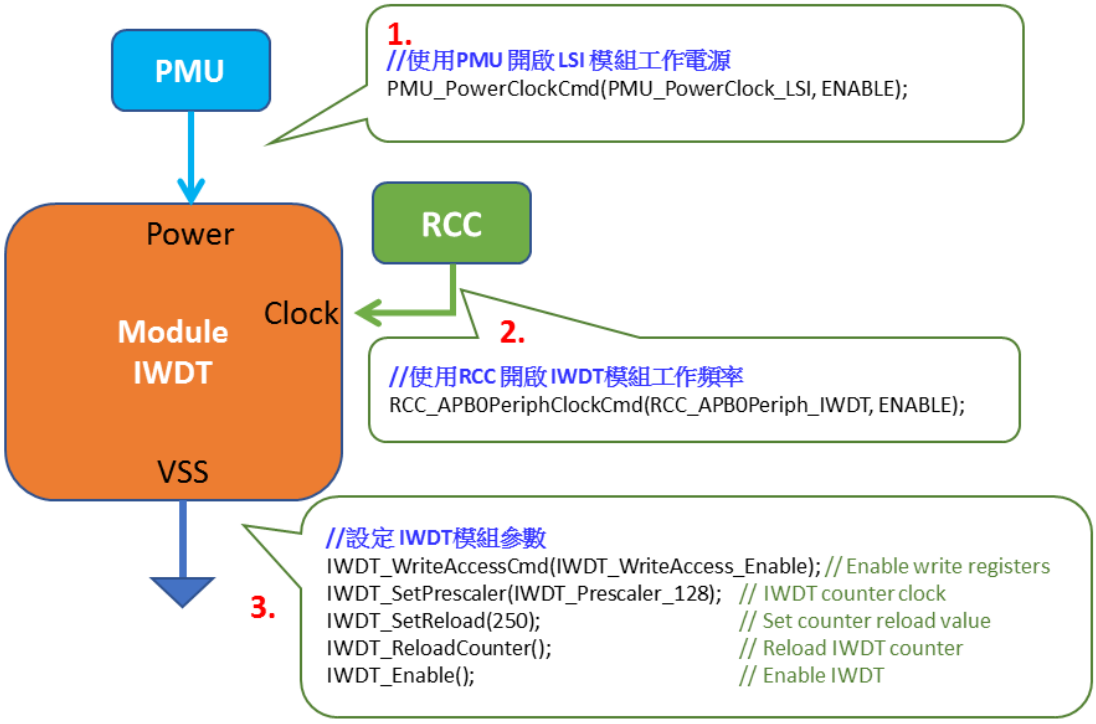
21. IWDT 功能说明

使用下列图示说明，使用 IWDT 设定时间，动作流程如下:

21.1 MCU 上电后初始化 IWDT

如下列 1~4 步骤，可参考周边程式库使用函式 InitialIwdt( )

(Step 1) 设定 PMU(电源管理单元) 开启类比电源提供给 IWDT 使用，如下图步骤 1.
(Step 2) 设定 RCC (时钟控制模组) 开启时脉提供给 IWDT 使用，如下图步骤 2.
(Step 3) 设定 IWDT 模组参数，如下图步骤 3.



21.2 范例程式

```
void InitialIwdt(void)  {
1.  PMU_PowerClockCmd(PMU_PowerClock_LSI, ENABLE);
2.  RCC_APB0PeriphClockCmd(RCC_APB0Periph_IWDT, ENABLE);
    IWDT_WriteAccessCmd(IWDT_WriteAccess_Enable); // Enable write 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
}
```

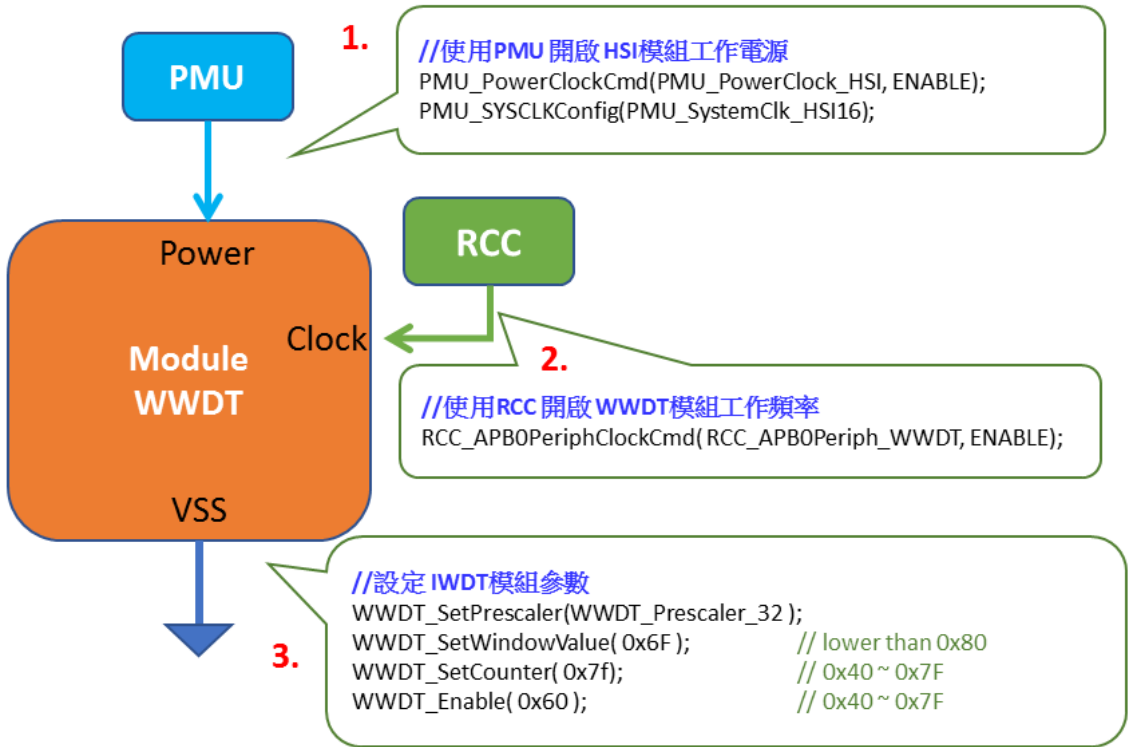
22. WWDT 功能说明

使用下列图示说明，使用 WWDT 设定时间，动作流程如下:

22.1 MCU 上电后初始化 WWDT

如下列 1~4 步骤，可参考周边程式库使用函式 InitialWwdt( )

(Step 1) 设定 PMU(电源管理单元) 开启类比电源提供给 WWDT 使用，如下图步骤 1.
(Step 2) 设定 RCC (时钟控制模组) 开启时脉提供给 WWDT 使用，如下图步骤 2.
(Step 3) 设定 WWDT 模组参数，如下图步骤 3.

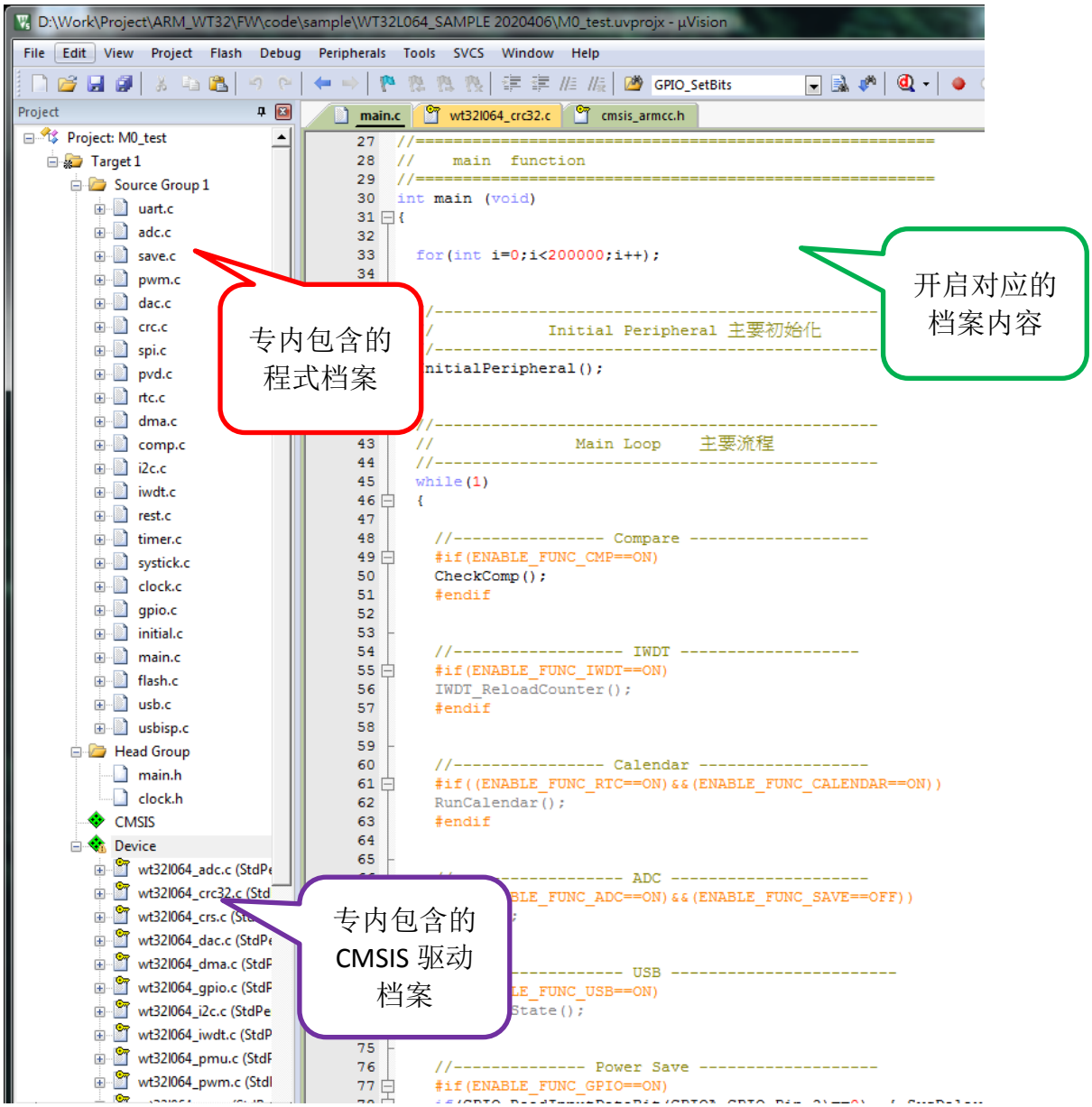


22.2 范例程式

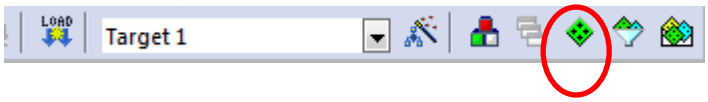
```
void InitialWwdt(void){
1.  PMU_PowerClockCmd(PMU_PowerClock_HSI, ENABLE);
    PMU_SYSCLKConfig(PMU_SystemClk_HSI16);
2.  RCC_APB0PeriphClockCmd(RCC_APB0Periph_WWDT, ENABLE);
    WWDT_DeInit();
3.  WWDT_SetPrescaler(WWDT_Prescaler_32);
    WWDT_SetWindowValue(0x6F);     // lower than 0x80
    WWDT_SetCounter(0x7f);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

23. 实例程式操作说明

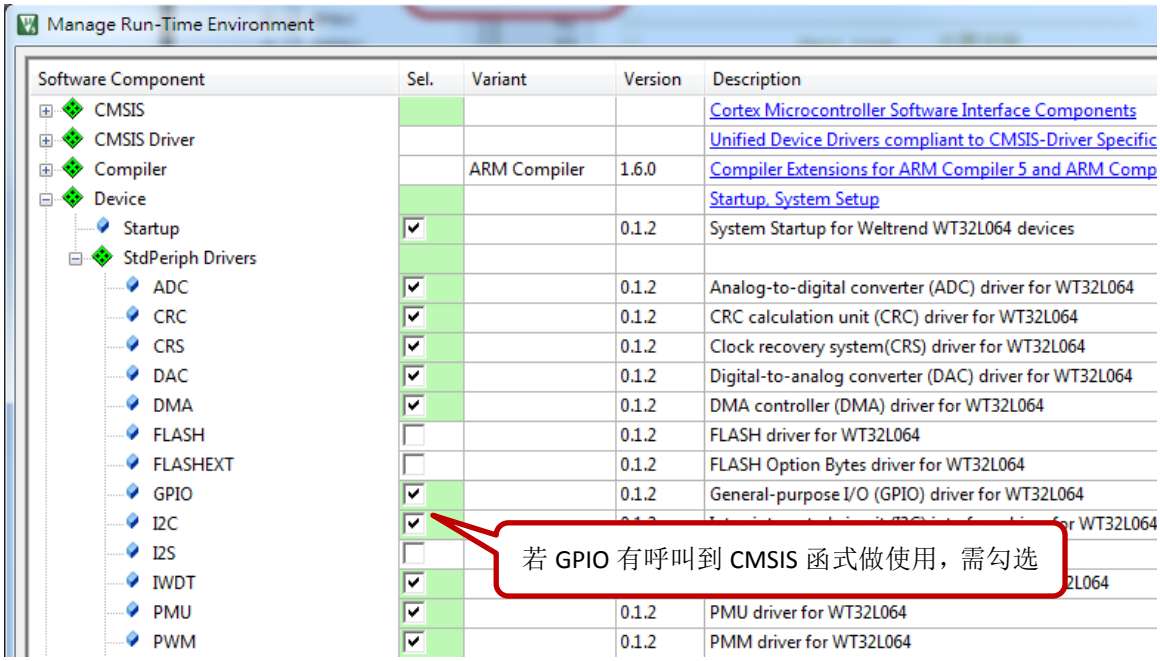
下列为如何使用参考范例的说明，专案名称周边程式库参考前章节范例程式，依周边功能放置个别档案，开启专案后的画面如下，主要分三部分：专案包含档案、CMSIS 驱动、各源始档内容



针对周边功能新增 CMSIS 驱动层函式，于 ARM-MDK 上方选单上点选 Manage Run-Time Environment 如下图示。

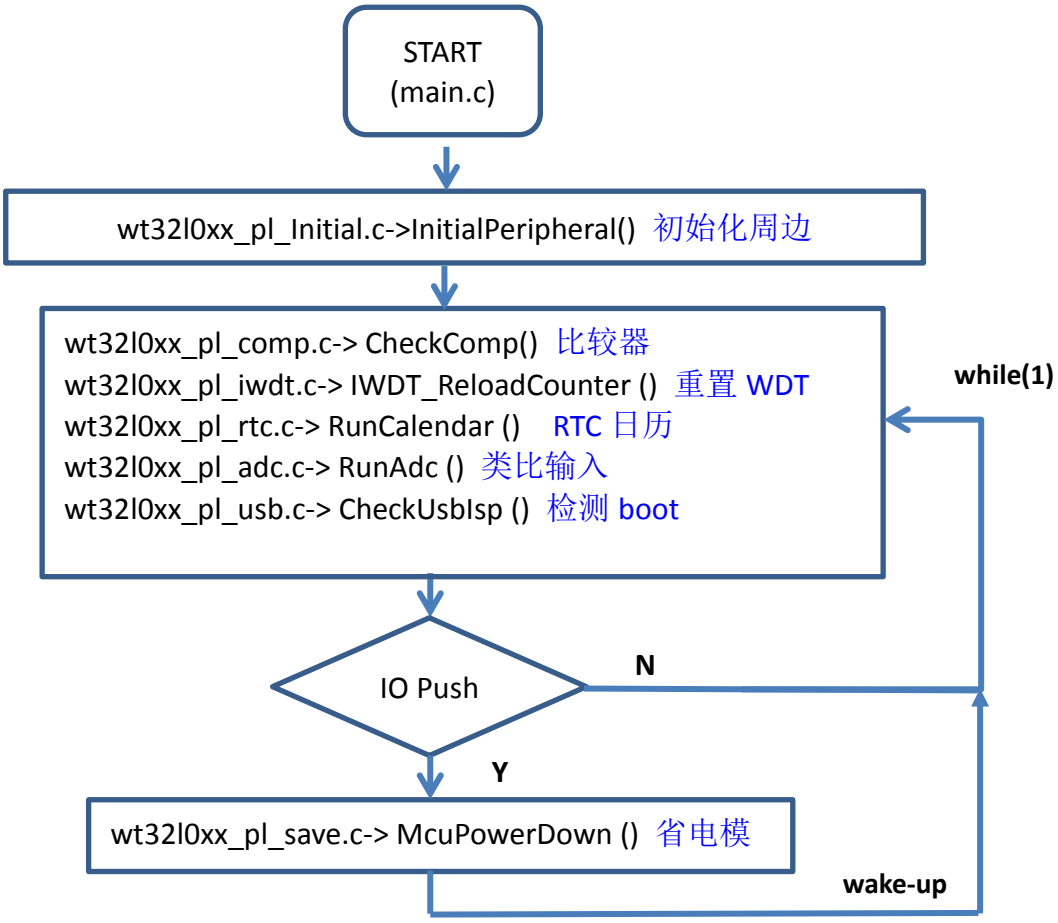


依序点选 Device->StdPeriph Drivers 如下图所示，可依应用需求加入所需功能，EX: ADC、DAC、FLASH、GPIO、I2C…etc，一般范例程式都已加入参考到的 CMSIS，若有缺少部分或反黄项可再补加选



23.1 范例 WT32L064_SAMPLE_2020xx 流程图

下列说明 WT32L064_SAMPLE_2020xx 流程图、主要档案内容与功能如下



依专案内档案名称与函式进行说明如下:

- **main.c 主程式流程，包括函式如下**

1.) InitialPeripheral() ----------参考到 initial.c，对周边的初始化
2.) CheckComp () ----------参考到 comp.c，比较器输出结果
3.) IWDT_ReloadCounter() ------参考到 iwdt.c，重置看门狗计数器
4.) RunCalendar() ----------参考到 rtc.c，检测日历数值
5.) RunAdc() ----------参考到 adc.c，执行 ADC 侦测
6.) CheckUsbState() ----------参考到 usb.c，检测 USB 状况
7.) McuPowerDown() ----------参考到 save.c，执行省电功能

程式主回圈内容如下:

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

  //-----------------------------------------------
  //        Initial Peripheral 主要初始化
  //-----------------------------------------------
  InitialPeripheral();

  //-----------------------------------------------
  //         Main Loop          主要流程
  //-----------------------------------------------
  while (1) {
       //---------------- Compare -------------------
       #if(ENABLE_FUNC_CMP==ON)
       CheckComp();         //检查COMP比较器状态
       #endif

       //------------------ IWDT -------------------
       #if(ENABLE_FUNC_IWDT==ON)
       IWDT_ReloadCounter();//看门狗重载
       #endif

       //---------------- Calendar ------------------
       #if((ENABLE_FUNC_RTC==ON)&&(ENABLE_FUNC_CALENDAR==ON))
       RunCalendar();       //检查RTC 日历资料
       #endif

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

       //------------------ USB ---------------------
       #if(ENABLE_FUNC_USB==ON)
       CheckUsbIsp(); //检查是否进入 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();      //进入省电模式
            #endif
            }
       }
     };//while(1);
  }
```

- **wt32l0xx_pl_library.h** 周边功能的开关，请依需求依序开启或关闭个别功能，程式内容如下。

```
//------------ Enable Function for Project --------------------------
// 请依序开启下列功能，  使用ON为致能，  OFF则为关闭

//------------ Core -------------------------
#define     SELECT_CORE_1p2V                OFF       // OFF:1.8V     // ON: VCORE=1.2V
#define     ENABLE_FUNC_CLOCK               ON        // 设定 IRC 16M~32KHz
#define     ENABLE_FUNC_LSI                 OFF       // 设定 LSI 37KHz 是否启动
#define     ENABLE_USB_CLOCK                OFF       // 设定 USB 48MHz 是否启动

#define     ENABLE_FUNC_SOFT_RST            OFF       // 设定 Soft Reset 是否启动

//------------ IO LED -------------------------
#define     ENABLE_FUNC_GPIO                ON        // 设定 GPIO 功能是否启动
#if(ENABLE_FUNC_GPIO==ON)
#define ENABLE_GPIO_INT             OFF       // 设定 GPIO Interrupt 是否启动
#define     ENABLE_LED_BLINK            ON        // 设定 GPIO Port-C LED 是否启动
#define     ENABLE_LED_RESET            OFF       // 设定 GPIO 测试 Reset 是否启动
#endif
#define     ENABLE_FUNC_SYSTICK         ON        // 设定 Systick 是否启动

//------------ Digtal ---------------
#define     ENABLE_FUNC_UART                      ON        // 设定 UART 功能是否启动
#if(ENABLE_FUNC_UART==ON)
#define ENABLE_FUNC_UART0       ON        // 设定 UART0 是否启动
#define ENABLE_FUNC_UART1       OFF       // 设定 UART1 是否启动
#define ENABLE_HW_IRDA              OFF       // 设定 IRDA是否启动 使用 UART0+1
#endif

#define     ENABLE_FUNC_PWM             OFF       // 设定 PWM 是否启动
#define     ENABLE_FUNC_IWDT            OFF       // 设定 IWDT 是否启动
#define     ENABLE_FUNC_WWDT            OFF       // 设定 WWDT 是否启动
#define     ENABLE_FUNC_FLASH           OFF       // 设定 仿真式 EEPROM 是否启动
#define     ENABLE_FUNC_CRC             OFF       // 设定 CRC 是否启动
#define     ENABLE_FUNC_SPI             OFF       // 设定 SPI 是否启动
#define     ENABLE_FUNC_RESET           OFF       // 设定 Rest 是否启动          (测试用)
#define     ENABLE_FUNC_PVD             OFF       // 设定 电压检测 是否启动(测试用)
#define     ENABLE_FUNC_RESET           OFF       // 设定 Reset 是否启动         (测试用)
#define     ENABLE_FUNC_I2C             OFF       // 设定 I2C 是否启动
#define     ENABLE_FUNC_I2S             OFF       // 设定 I2S 是否启动
#define     ENABLE_FUNC_TIMER           OFF       // 设定 Timer 是否启动
#define     ENABLE_FUNC_DMA             OFF       // 设定 DMA 是否启动, 使用 Timer+ADC 但须都启
动
#define     ENABLE_FUNC_USB             OFF       // 设定 USB 是否启动


//----------- Analog -----------------
#define     ENABLE_FUNC_CMP             ON        // 设定 COMPARE 是否启动
#define     ENABLE_HW_CMP_SPEED_HI      OFF       //HI:4.5uALO:5.5uA
```

OFF: 关闭

ON: 开启

数位功能类 开关

类比功能类 开关

```
#define     ENABLE_FUNC_ADC          OFF        // 设定 ADC 是否启动
#if(ENABLE_FUNC_ADC==ON)
#define ENABLE_HW_ADC_AWD            OFF
#define ENABLE_HW_ADC_ALL            OFF
#endif
#define     ENABLE_FUNC_DAC          OFF


//-------- RTC -------
#define     ENABLE_FUNC_RTC                OFF  // 设定 RTC 是否启动
#if(ENABLE_FUNC_RTC==ON)
#define     ENABLE_FUNC_ALARM              OFF  //RTC Enable first (59 sec)
#define     ENABLE_FUNC_CALENDAROFF             //RTC Enable first (not for sleep)
#define     ENABLE_RESET_RTC               OFF  //ON: Test RTC keep RAM data
#endif


//------ 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 -----
#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
```

RTC 功能开关

省电功能开关

唤醒功能开关

- **wt32l0xx_pl_initial.c 周边的初始化，包括函式如下**

1.) InitialPeripheral()----------初始化周边功能 EX: ADC、UART、PWM

初始化顺序: **InitialSysClock() -> InitialGpio() -> InitiSysTick() -> InitialUart0() ->… -> InitialIwdt() -> InitialAdc() -> InitialDac() -> SPI_Config0() -> InitialI2c() -> InitialPwm() -> InitialRtc()->…etc**

- **wt32l0xx_pl_clock.h 工作频率的选择，可选择 HIS、MSI、HSE、PLL 四种类型，程式如下**

```
//--------- Use PLL for HSI 32MHz ----------------------------
#define CLOCK_PLL_HSI_X2_32MHZ  ON    //ON:开启使用PLL倍频HSI 16MHz至32Hz给系统使用

//--------- 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          //当选择MSI时，系统选择的工作频率

//--------- Select MCU Clock Type ----------------------------
#define     CLK_HSI    0    //Internal OSC 16MHz
#define     CLK_MSI         1     //Internal OSC 65K~4M
#define     CLK_PLL         2     //Use Multiple X with HSI or HSE
#define     CLK_HSE         3 //External OSC 1~25MHz

#define     SYS_CLOCK_SEL   CLK_MSI         //系统选择频率的类型
```

选择速度

选择类型

- **wt32l0xx_pl_clock.c 工作频率设置函式，包括函式如下**

1.) InitialSysClock () ----------执行系统频率选择，节录内容如下

```
#if(SYS_CLOCK_SEL==CLK_HSI)         //使用 HSI 作系统频率
      PMU_PowerClockCmd(PMU_PowerClock_HSI, ENABLE);
      PMU_SYSCLKConfig(PMU_SystemClk_HSI16);

#elif(SYS_CLOCK_SEL==CLK_MSI)       //使用 MSI 作系统频率
      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)       //使用 PLL 作系统频率
      //…省略
#elif(SYS_CLOCK_SEL==CLK_HSE)       //使用 HSE 作系统频率
```

2.) InitialUsbClock() ----------执行 USB 频率选择
3.) Delayms() ----------执行延迟功能
4.) DelayCount() ----------执行延迟功能

- **wt32l0xx_pl_gpio.c 外设 IO 类型设置，包括函式如下，可参考章节 4**

1.) GPIO_Handler ()----------中断服务 GPIO 功能
2.) InitialGpio ()----------初始化 GPIO 功能

```
GPIO的4种类型: GPIO_Mode_IN => 基本输入
               GPIO_Mode_OUT =>基本输出
               GPIO_Mode_AF =>复合使用功能，EX:UART、SPI、I2C …
               GPIO_Mode_AN =>类比输入功能，EX:ADC、USB、COMP …
```

- **wt32l0xx_pl_systick.c 内建 24bit 计时器设置，包括函式如下**

1.) SysTick_Handler ()----------中断服务 systick 功能
2.) InitiSysTick ()----------初始化 systick 功能
3.) SysDelay ()----------使用 systick 延迟功能

- **wt32l0xx_pl_flash.c 模拟 EEPROM 烧录设置，包括函式如下**

1.) RunFlash ()----------使用模拟 EEPROM 烧录功能

- **wt32l0xx_pl_uart.c 异步收发器传输设置，包括函式如下，可参考章节 5**

1.) UART0_Handler ()----------中断服务 UART0 功能
2.) UART1_Handler()----------中断服务 UART1 功能
3.) InitialUart0 ()----------初始化 UART0 功能
4.) InitialUart1()----------初始化 UART1 功能
5.) fputc ()----------搭配 printf()使用发射串列资料功能
6.) fgetc()----------搭配 printf()使用接收串列资料功能
7.) DRV_IntToStr()----------数字转字串
8.) Str2Num()----------字串转数字
9.) uart_send_str()----------使用 UART0/1 发射串列资料
10.) uart_clear_str()----------清除串列内容

- **wt32l0xx_pl_adc.c 类比侦测设置，包括函式如下**

1.) ADC_Handler ()----------中断服务 ADC 功能
2.) InitialAdc ()----------初始化 ADC 功能
3.) InitialAllAdc ()----------初始化 ADC 所有通道功能
4.) RunAdc()----------执行 ADC 目标通道转换功能
5.) RunAllAdc ()----------执行 ADC 所有通道转换功能
6.) RunAdcConvert()----------执行 ADC 通道单次转换功能
7.) API_AverADCData ()----------执行 ADC 通道转换功能，计算平均
8.) ADC_StartOfConversion_1() -启动 ADC 模组转换
9.) ADC_StopOfConversion_1() -停止 ADC 模组转换
10.) HEX2BCD()----------16 进制转 10 进制

- **wt32l0xx_pl_save.c 省电功能设置，包括函式如下**

1.) McuPowerDown ()----------执行省电功能前置作业并呼叫 Save()
2.) Save()----------依设定 SLEEP、STOP、STANDBY 执行省电功能

- **wt32l0xx_pl_pwm.c 脉冲周期调变功能设置，包括函式如下**

1.) InitialPwm() ----------执行 PWM 初始化并输出功能

- **wt32l0xx_pl_dac.c 类比输出设置，包括函式如下**

1.) DAC_Convert () ----------带入数值至 DAC 并输出功能
2.) DAC_Handler()----------执行 DAC 中断功能
3.) InitialDac()----------执行 DAC 初始化

- **wt32l0xx_pl_crc.c 检查码 CRC 设置，包括函式如下**

1.) DAC_Convert () ----------带入数值至 DAC 并输出功能
2.) DAC_Handler()----------执行 DAC 中断功能

- **wt32l0xx_pl_spi.c 串列周边传输设置，包括函式如下**

1.) SPI_Config0 () ----------执行 SPI 0 初始化
2.) SPI_Config1()----------执行 SPI 1 初始化
3.) SPI1_Handler()----------执行 SPI 中断功能

- **wt32l0xx_pl_pvd.c 电压侦测设置，包括函式如下**

1.) InitPvd () ----------执行 PVD 初始化
2.) PVD_Handler ()----------执行 PVD 中断功能

- **wt32l0xx_pl_rtc.c 实时计数器设置，包括函式如下**

1.) InitialRtc () ----------执行 RTC 初始化
2.) RTC_AlarmCmd ()----------执行 DAC 中断功能
3.) RTC_Handler()----------执行 RTC 中断功能
4.) RunCalendar()----------执行 RTC 日历功能
5.) SetAlarm()----------设定 RTC 闹钟功能

- **wt32l0xx_pl_dma.c 直接记忆体存取设置，包括函式如下**

1.) ADC_Config () ----------执行 ADC 初始化
2.) DMA_Config ()----------执行 DMA 初始化
3.) DMA0_Handler()----------执行 DMA 中断功能
4.) InitiDma()----------初始化 DMA 通道
5.) RunDma()----------执行上述 ADC 搬移至 DMA

- **wt32l0xx_pl_comp.c 比较器设置，包括函式如下**

1.) CheckComp () ----------
2.) CMP0_VOUT_Handler ()----------执行 CPM0 中断功能
3.) CMP1_VOUT_Handler()----------执行 CMP1 中断功能
4.) InitialComp()----------初始化 COMP 比较器
5.) RumComp()----------执行 COMP 比较器

- **wt32l0xx_pl_i2c.c 标准 I²C 汇流排设置，包括函式如下**

1.) InitialI2c () ---------- -初始化 I2C 传输
2.) RunI2cTest ()----------执行 I2C 传输

- **wt32l0xx_pl_iwdt.c 看门狗设置，包括函式如下**

1.) InitialIwdt () ---------- 初始化看门狗

- **wt32l0xx_pl_reset.c 软体复位设置，包括函式如下**

1.) InitLowVoltReset () ---------- 初始化低电压复位
2.) RunReset ()----------测试低电压复位

- **wt32l0xx_pl_timer.c 计数计时器设置，包括函式如下**

1.) ConfigTimerCapture () ----------配置 Timer 执行捕捉模式
2.) ConfigTimerClockGpio ()----------配置 Timer 执行输出模式
3.) ConfigTimerInterrutp()----------配置 Timer 执行中断模式
4.) ConfigTimerOutPWM()----------配置 Timer 执行 PWM 模式
5.) ConfigTimerTimeMode()----------配置 Timer 执行计时器模式
6.) TIMER0_Handler()----------执行 TIMER0 中断功能
7.) TIMER1_Handler()----------执行 TIMER1 中断功能
8.) TIMER2_Handler()----------执行 TIMER2 中断功能

- **wt32l0xx_pl_usb.c 通用序列汇流排设置，包括函式如下**

1.) CLEAR_STALL () ---------- 清除 EP 端点 STALL 停滞状态
2.) ENDPOINT_DISABLE ()---------- 关闭 EP 端点功能
3.) FUN_INIT()---------- 初始化 USB 端点 EP0 或其余端点
4.) FUN_INT()---------- USB 端点 EP0~EPx 中断服务函式
5.) FUN_INT2()---------- 处理终端 EP2 端点中断
6.) FUN_REQUESTS()----------PC 端发送 USB 请求命令后，处理解析 USB 命令

7.) FUNTx0Send()----------装置传输资料至 USB FIFO
8.) HID_EP1()----------USB 端点 EP1 传送资料
9.) HID_EP2()----------USB 端点 EP2 传送资料
10.) HID_EP3()----------USB 端点 EP3 传送资料
11.) HID_GET_IDLE()----------USB-HID 取得 IDLE 时间设置
12.) HID_GET_PROTOCOL()----------USB-HID 取得 PROTOCOL 设置
13.) HID_GET_REPORT()----------USB-HID 取得 REPORT 设置的值
14.) HID_SET_IDLE()----------USB-HID 设定 IDLE 设置值
15.) HID_SET_PROTOCOL()----------USB-HID 设定 PROTOCOL 格式
16.) HID_SET_REPORT()----------USB-HID 设定 REPORT 格式
17.) IN_ENDPOINT_ENABLE()----------启动 USB 端点 EP 的 IN 功能
18.) OUT_ENDPOINT_ENABLE()----------启动 USB 端点 EP 的 OUT 功能
19.) ProcessUsbResetINT()----------重置并初始化 USB 端点 EP0~EPx
20.) ProcessUsbRx0INT()----------处理 EP0 接收中断行程
21.) ProcessUsbTx0INT()----------处理 EP0 发射中断行程
22.) ProcessUsbxx1INT()----------处理 EP1 收发中断行程
23.) ProcessUsbxx2INT()----------处理 EP2 收发中断行程
24.) ProcessUsbxx3INT()----------处理 EP3 收发中断行程
25.) SendFirstBuffer()----------发送第一笔描述元至 PC 端
26.) SendFirstBufferWithSize()----------发射第一笔描述元至 PC 端，带长度
27.) SendNextBuffer()----------发送第二笔描述元至 PC 端
28.) SET_STALL()----------设定端点 EP 停滞
29.) USB_CLEAR_FEATURE()----------清除 Feature 配置， 处理 USB 标准清除命令
30.) USB_GET_CONFIG()----------取得 Config 配置
31.) USB_GET_DESCRIPTOR()----------取得 Descriptor 描述元
32.) USB_GET_INTERFACE()----------取得 Interface 配置
33.) USB_GET_STATUS()----------读取 STATUS 状态
34.) USB_NOT_SUPPORT()----------不支持回应
35.) USB_RECEIVE_DATA()----------读取 USB 接收资料
36.) USB_SET_ADDRESS()----------设定 USB 装置地址
37.) USB_SET_CONFIG()----------设定 Config 配置
38.) USB_SET_FEATURE()----------设定 Feature 配置

39.) USB_SET_INTERFACE()----------设定 Interface 配置
40.) USB0_Handler()----------USB 信号中断向量服务常式
41.) USB1_Handler()----------USB 端点 EP 中断向量服务常式
42.) USBTxxINT()----------USB 传输中断
43.) USBTxxSend()----------USB 传输预载入 Buffer

- **wt32l0xx_pl_usbisp.c 通用序列汇流排进入 Boot 设置，包括函式如下**

1.) CheckUsbIsp()----------检查 USB 插头是否接入 HOST
2.) enter_usbisp()----------执行 USB 烧录 ISP 程式，设定后复位
3.) go_usb_suspend()----------进入 Suspend 待机省电模式
4.) InitialUSB()----------执行 USB 初始化

24. 版本更改纪录:

版 本	纪 录	日 期
V1.0	初稿	2020/9/18