WT32L064_032
周邊功能與程式說明
應用文件

(中文版)

Rev. 1.0
September 2020

目 錄

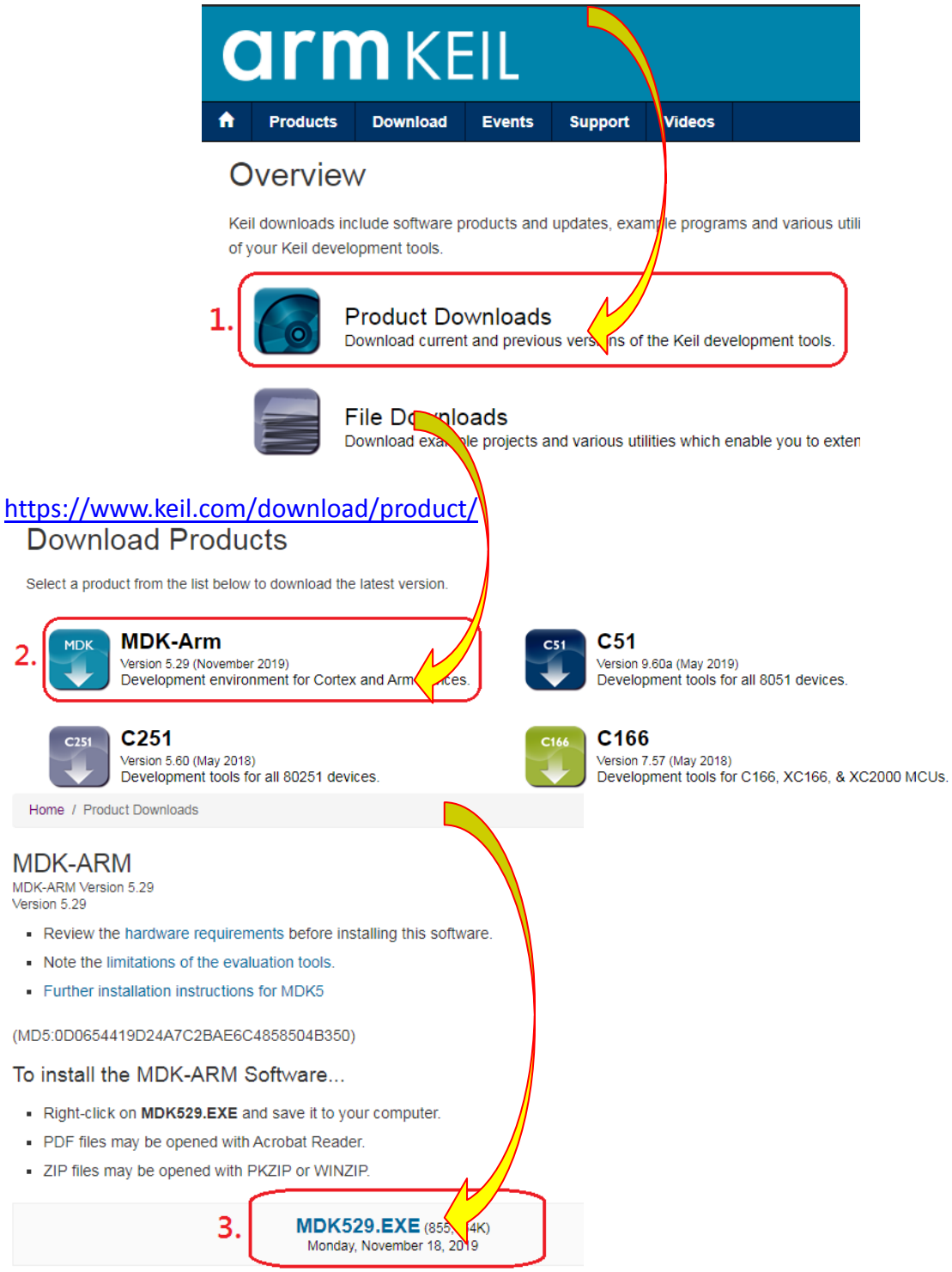
1. ARM-MDK 安裝與環境設定 5
2. CMSIS 中間層驅動說明 8
2.1 定義: 8
2.2 應用說明: 8
2.3 CMSIS 內容說明: 9
3. PACK 範例程式架構說明 10
3.1 EXAMPLES 資料夾內功能說明 10
4. GPIO 功能說明 12
4.1 MCU 進行 GPIO 初始化 12
4.2 讀取 GPIO 輸入值 12
4.3 設定 GPIO 輸出值 13
4.4 範例程式 GPIO 13
5. UART 功能說明 15
5.1 MCU 上電後初始化 UART 15
5.2 範例程式 UART 15
5.3 UART 進行 RX 接收資料與 TX 發射資料 16
6. ADC 功能說明 17
6.1 MCU 進行 ADC 初始化 17
6.2 範例程式 ADC 17
6.3 進行 ADC 偵測與轉換資料 19
7. DAC 功能說明 20
7.1 MCU 進行 DAC 初始化 20
7.2 範例程式 DAC 20
7.3 進行 DAC 資料轉換輸出 21
8. SLEEP 功能說明 22
8.1 MCU 進行 SLEEP 初始化 22
8.2 範例程式 SAVE.C 23
9. STOP 功能說明 25
9.1 MCU 進行 STOP 初始化 25
9.2 範例程式 SAVE 26

10. STANDBY 功能說明 28
10.1 MCU 進行 STANDBY 初始化 28
10.2 範例程式 SAVE 28
11. COMPARATOR 功能說明 31
11.1 MCU 進行 COMPARATOR 初始化 31
11.2 範例程式 COMP 31
11.3 COMPARATOR 之中斷功能 32
12. FLASH 讀寫功能說明 33
12.1 MCU 進行 FLASH 初始化 33
12.2 範例程式 FLASH 33
13. RTC 功能說明 36
13.1MCU 進行 RTC 初始化 36
13.2 範例程式 RTC 37
13.3 設定 RTC 時間 37
14. TIMER 功能說明 38
14.1 MCU 進行 TIMER 初始化 38
14.2 範例程式 TIMER 39
15. USB 與 HID 功能說明 41
15.1 USB-HID 架構說明 41
15.2 USB-HID 裝置與組態描述元說明 42
15.3 USB-HID 報告描述元與用途頁說明 44
15.4 HID REPORT 發射與接收流程 46
15.5 HID FEATURE 發射與接收流程 49
16. SPI 功能說明 52
16.1 MCU 上電後初始化 SPI 52
16.2 範例程式 52
17. I2C 功能說明 54
17.1 MCU 上電後初始化 I2C 54
17.2 範例程式 54
17.3 I2C 進行 RX 接收資料 與 TX 發射資料 56
18. I2S 功能說明 57
18.1 MCU 上電後初始化 I2S 57
18.2 範例程式 57

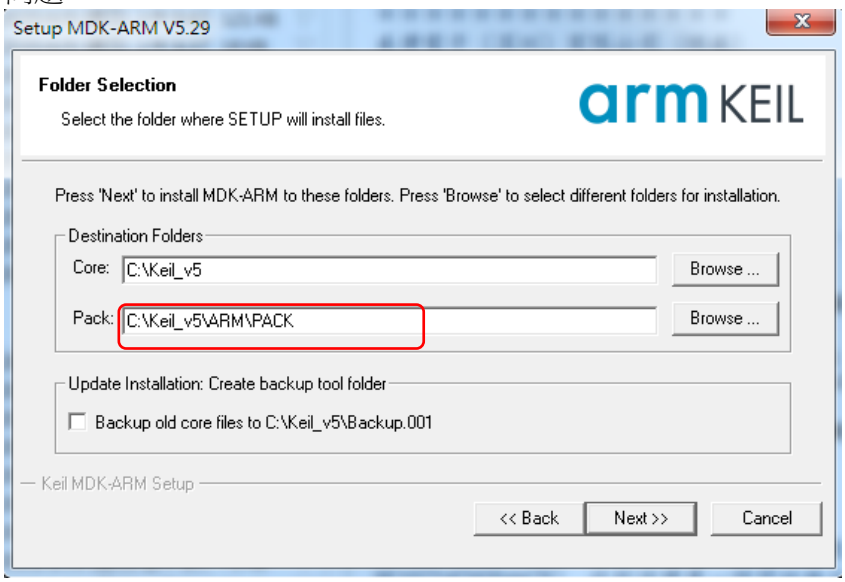
19. PWM 功能說明 59
19.1 MCU 上電後初始化 PWM 59
19.2 範例程式 59
20. DMA 功能說明 61
20.1 MCU 上電後初始化 DMA 61
20.2 範例程式 61
21. IWDT 功能說明 63
21.1 MCU 上電後初始化 IWDT 63
21.2 範例程式 63
22. WWDT 功能說明 64
22.1 MCU 上電後初始化 WWDT 64
22.2 範例程式 64
23. 實例程式操作說明 65
23.1 範例 WT32L064_SAMPLE_2020xx 流程圖 67
24. 版本更改紀錄: 78

1. ARM-MDK 安裝與環境設定

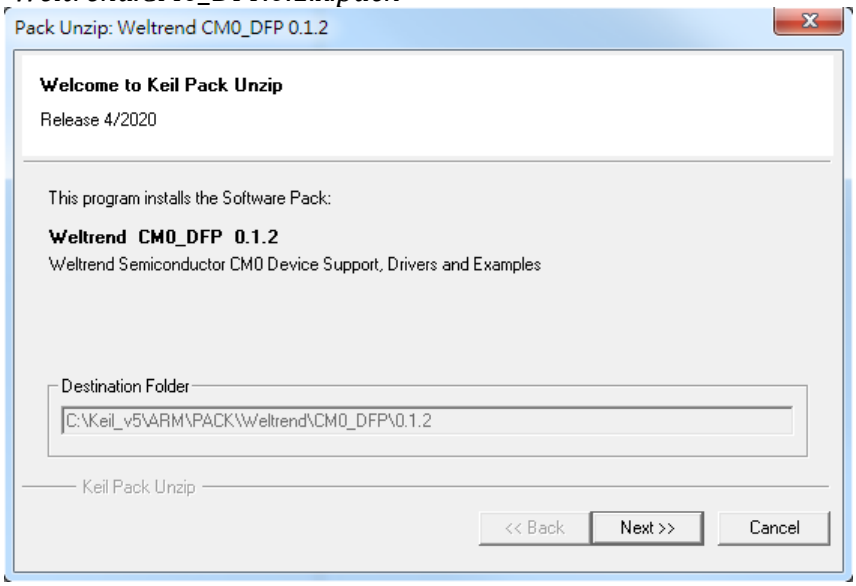
(Step 1)　請先上網下載 **ARM-MDK**，https://www.keil.com/download/



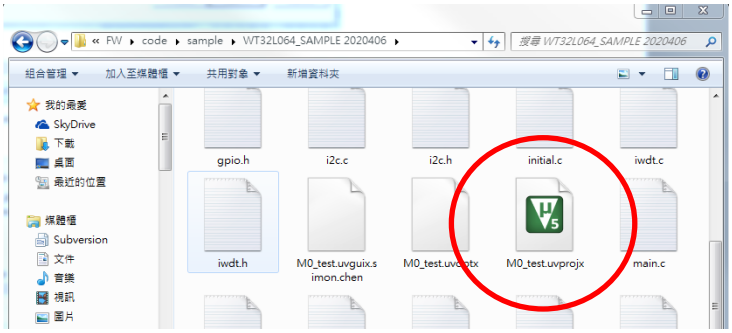
安裝過程中會詢問預設 PACK 路徑，請指定 **C:\Keil_v5\ARM\PACK** 如下，避免後續 PACK 安裝問題



(Step 2)下載並安裝 MDK 後，請於 PC 端再安裝偉詮 PACK 檔案 *Weltrend.CM0_DFP.0.1.x.pack*



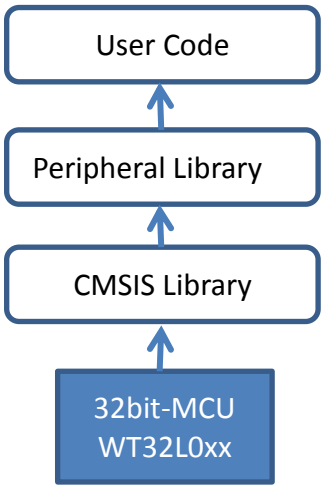
(Step 3)　安裝 ARM-MDK 後有基礎 32KB 可免費使用，或可自行採購軟體，安裝後請在電腦上開啟相關 WT32L064 專案進行編譯工作。



2. CMSIS 中間層驅動說明

2.1 定義:

ARM® Cortex™ 微控制器軟體介面標準 (CMSIS)是一組韌體庫可驅動 ARM 處理器，該韌體介面提供一標準函式直接面向周邊且名稱一致使用簡單，可利軟體的重複呼叫使用，縮短微控制器開發人員開發時間。與此架構上廠商再提供一層周邊程式庫(Peripheral Library)直接面向基本應用，提供一組基本初始化與操作範例程式，直接面向應用端可加快程式操作與編寫。



2.2 應用說明:

CMSIS 目的是將函式對應到 MCU 的暫存器控制描寫出來，對於使用者可使用標準化函式例如 ADC_StartOfConversion() ，而周邊程式庫(PL)則是提供該函式的操作與範例。

EX: 程式檔案 main.c 內容

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

周邊應用的檔案 wt32l0xx_pl_adc.c 內容

```
API_AverADCData() {
ADC_StartOfConversion();   //使用 CMSIS 標準，呼叫周邊函式進行平均濾波 ADC
……
}
```

CMSIS 驅動的檔案 wt32l064_adc.c 內容

```
void ADC_StartOfConversion(void)
{
   ADC->ADCCR |= (uint32_t)ADC_START;   // 實際對應到 MCU 暫存器位址
}
```

2.3 CMSIS 內容說明:

安裝完 WT32L064 PACK 後，預設 CMSIS 的路徑為 C:\Keil_v5\ARM\Packs\Weltrend\CM0_DFP\0.1.2 \WT32L064\StdPeriph_Driver，標頭檔放置 Include 資料夾，原始檔放置 Source，其內容有對 WT32L064 所有的周邊做基礎設定，檔案清單如下。

檔案名稱	功能說明
wt32l064_adc	類比偵測 ADC 相關函式
wt32l064_crc32	CRC32 計算關函式
wt32l064_crs	校正 IC 內部頻率相關函式
wt32l064_dac	類比輸出 DAC 相關函式
wt32l064_dma	直接記憶體存取 DMA 相關函式
wt32l064_flash	仿真式 EEPROM 燒錄 FLASH 相關函式
wt32l064_gpio	GPIO 相關函式
wt32l064_i2c	I2C 相關函式
wt32l064_i2s	I2S 相關函式
wt32l064_iwdt	IWDT 獨立看門狗相關函式
wt32l064_pmu	PMU 電源控制單元相關函式
wt32l064_pwm	PWM 相關函式
wt32l064_rcc	RCC 頻率控制單元相關函式
wt32l064_rtc	RTC 計時器相關函式
wt32l064_spi	SPI 相關函式
wt32l064_timer	TIMER 相關函式
wt32l064_usart	UART 相關函式
wt32l064_usbd	USB 相關函式
wt32l064_wwdt	WWDT 視窗型看門狗相關函式

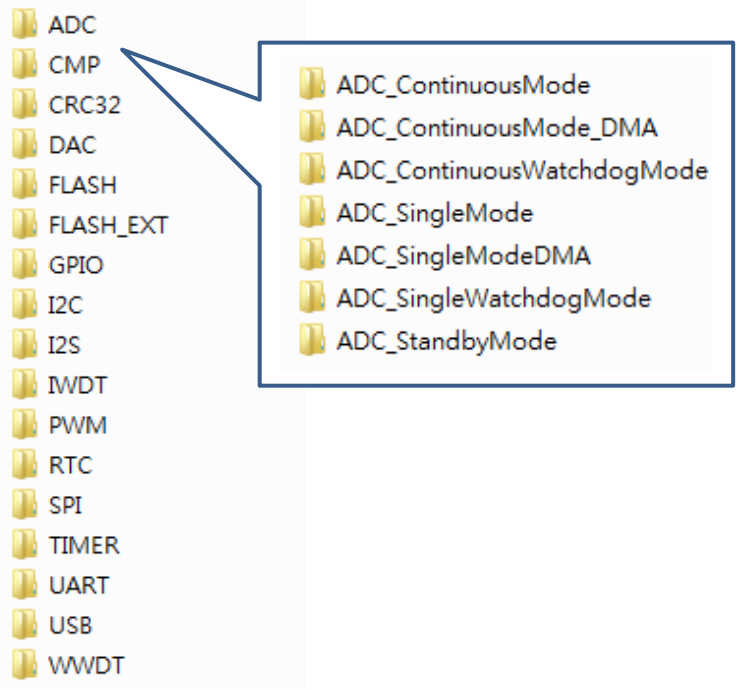
每個檔案開頭都有簡易說明該檔案的目的功能為何，內部每個函式亦有針對該功能與參數做說明，舉例說明檔案 wt32l064_gpio.c 內，其中 GPIO_SetBits()函數的內容如下

```
/**
   * @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: 主要功能為 Bit 設定
@param GPIOx: 目標 PORT
@param GPIO_Pin: 目標 PIN
@note: 補充說明 PIN 有 0~15 且有 PortA~D

3. PACK 範例程式架構說明

針對各種應用單元有基本範例程式，當 PACK 安裝後參考下列路徑 C:\...\Arm\Packs\Weltrend\CM0_DFP\0.1.2 \WT32L064\Examples，資料夾內有子單元內含原始檔與專案，下列為 ADC 範例程式，依其功能有單一次轉換與連續轉換與其分類，如下圖示所示。



資料夾內含有相對應功能的範例程式，例如 ADC->ADC_SingleMode 為單次類比轉換功能，該資料夾內有 readme.txt 可參閱簡易說明

3.1 Examples 資料夾內功能說明

根資料夾	資料夾名稱	功能說明
ADC	ADC_ContinuousMode	連續 ADC 偵測
	ADC_ContinuousMode_DMA	使用 DAM 作連續 ADC 偵測
	ADC_ContinuousWatchdogMode	使用連續 ADC 做邊界偵測
	ADC_SingleMode	單一 ADC 偵測
	ADC_SingleModeDMA	使用 DAM 作單一 ADC 偵測
	ADC_SingleWatchdogMode	使用單一 ADC 做邊界偵測
	ADC_StandbyMode	使用低耗電 ADC 模式
CMP	CMP	比較器範例
CR32	CRC32	CRC32 計算範例
DAC	DAC	DAC 輸出範例
	DAC_HighCurrent	DAC 高推力輸出範例
FLASH	FLASH_PROGRAM	燒錄程式區(EEPROM)範例
	FLASH_PROGRAM_INT	燒錄程式區(EEPROM)中斷範例
FLASH_EXT	FLASH_OB_EEPROM	燒錄資料區(EEPROM)範例
	FLASH_OB_LEVEL	於資料區(OB)作加密等級

根資料夾	資料夾名稱	功能說明
	FLASH_OB_READ_PROTECTION	於資料區(OB)作防讀加密
	FLASH_OB_WRITE_PROTECTION	於資料區(OB)作防寫加密
GPIO	GPIO	GPIO 基本範例
	GPIO_Bit_Set_Reset	設定 GPIO 位元的範例
	GPIO_Input	設定 GPIO 輸入的範例
	GPIO_Interrupt	設定 GPIO 中斷的範例
	GPIO_Output	設定 GPIO 輸出的範例
	GPIO_Toggle	設定 GPIO 輸出反向的範例
I2C	I2C_Master_Slave_DMA_FLAG	I2C 從端模式與 DMA 搬移
	I2C_Master_Slave_DMA_INT	I2C 從端模式與 DMA 中斷
	I2C_Master_Slave_FLAG	I2C 從端模式
	I2C_Master_Slave_FLAG_EEPROM	I2C 從端模式與 EEPROM 燒錄
	I2C_Master_Slave_INT	I2C 主端與從端模式各一組互傳
I2S	I2S_DMA	I2S 從端模式與 DMA 搬移
	I2S_INT	I2S 從端模式與 DMA 中斷
	I2S_POLLING	I2S 從端模式
IWDT	IWDT	看門狗設定範例
PWM	PWM	PWM 脈波調變範例
RTC	RTC_1sec	RTC 計時器設定 1 秒範例
	RTC_Alarm	RTC 計時器設定鬧鐘範例
SPI	MSPI_DMA_FLAG	SPI 使用 DMA 傳輸範例
	MSPI_DMA_INT	SPI 使用 DMA 傳輸與中斷範例
	MSPI_FLAG	SPI 傳輸範例
	MSPI_FLAG_FLASH_MX25L4006	SPI 搭配 MX25L4006 傳輸範例
	MSPI_INT	SPI 傳輸與中斷範例
TIMER	TMR_Capture_Mode	Timer 捕捉模式範例
	TMR_Compare_Mode	Timer 比較模式範例
	TMR_Counter_Mode	Timer 計數模式範例
	TMR_DMA_Mode	Timer 搭配 DMA 使用範例
	TMR_PWM_MODE	Timer 輸出 PWM 使用範例
	TMR_Timer_Mode	Timer 普通計時範例
UART	UART_DMA	串列傳輸搭配 DMA 使用範例
	UART_HalfDuplexMode	串列傳輸使用半雙工範例
	UART_InterruptAndFlagManage	串列傳輸使用中斷範例
	UART_IrDA_Mode	串列傳輸使用 IRDA 範例
	UART_TxRx	串列傳輸同時發射與接收範例
USB	USB_HID	HID KEYBOARD 簡易範例
	USB_HID_AUDIO_WM8731	HID 搭配 I2S 撥放 WM8731 音樂
WWDT	WWDT	視窗型看門狗

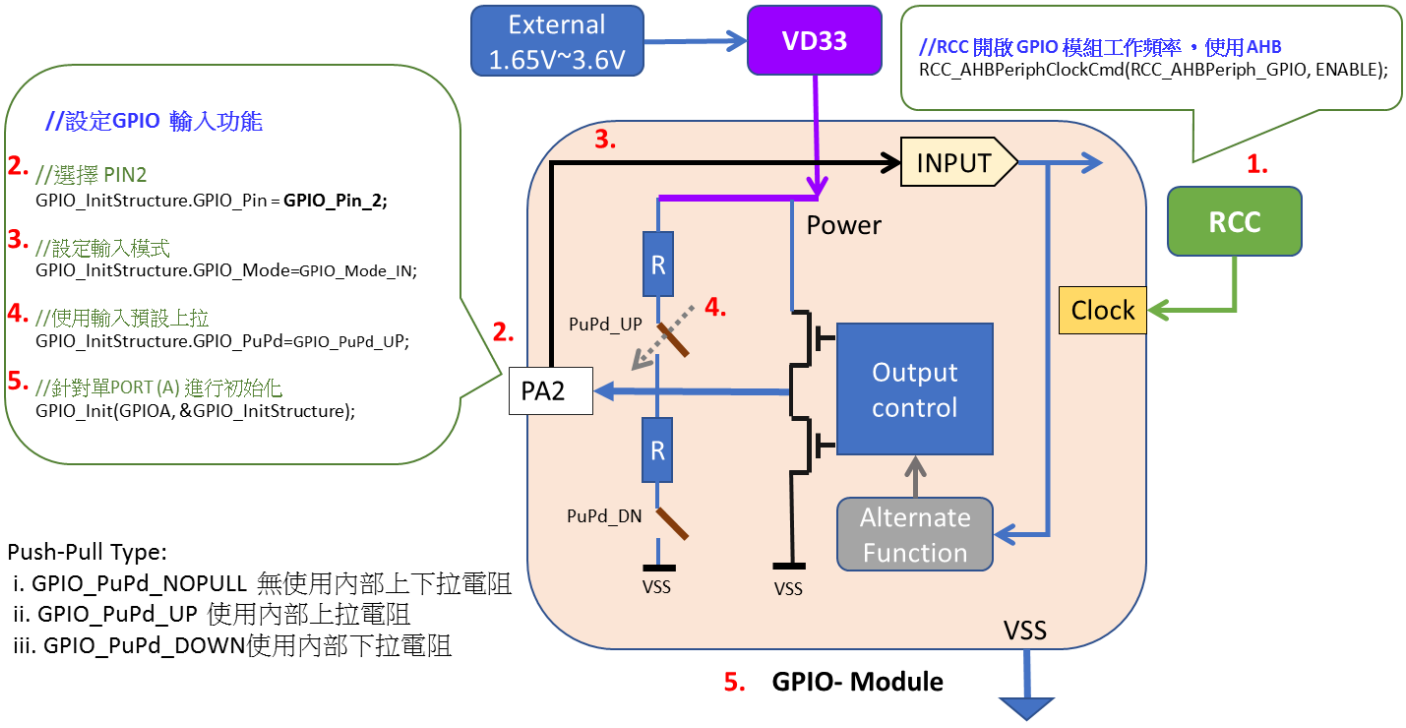
4. GPIO 功能說明

使用下列圖示說明，GPIO 使用 PA2 做輸入，使用 PC4~PC7 做輸出，動作流程如下:

4.1 MCU 進行 GPIO 初始化

使用 PA2 內容如下，可參考周邊程式庫 wt32l0xx_pl_gpio.c 之函式 InitialGpio ( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 GPIO 使用，如下圖步驟 1.
(Step 2) 設定 GPIO，此處範例選擇 PIN2，如下圖步驟 2.
(Step 3) 設定輸入或輸出模式，如下範例 IO 選擇 INPUT，如下圖步驟 3.
(Step 4) 設定上拉或下拉阻抗，如下範例 IO 選擇上拉，如下圖步驟 4.
(Step 5)設定 GPIO 之 Port-A 模組初始化，並寫入暫存器，如下圖步驟 5.



4.2 讀取 GPIO 輸入值

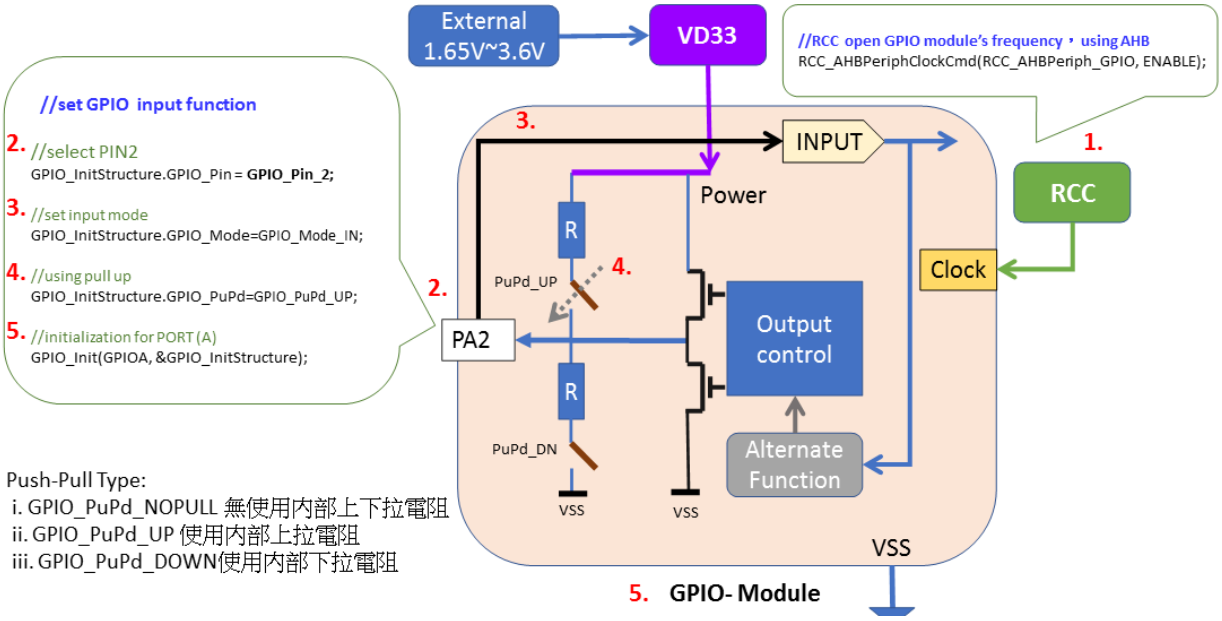
使用 GPIO_ReadInputDataBit()讀取 BIT 資料值，例如當 PA2=LO 時，執行輸入的寫法如下

```
if (GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_2) == 0) {
//…… 寫入對應功能
}
```

4.3 設定 GPIO 輸出值

MCU 上電後初始化 PC4~PC7，其做法內容如下，或可參考範例函式 InitialGpio ( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 GPIO 使用，如下圖步驟 1.
(Step 2) 設定 GPIO，可選擇 PIN4，如下圖步驟 2.
(Step 3) 設定輸入或輸出模式，如下範例 IO 選擇 OUTPUT，如下圖步驟 3.
(Step 4) 設定輸出模式，有推挽式與開汲極，下例選擇開汲極，如下圖步驟 4.
(Step 5) 設定上拉或下拉阻抗，如下範例 IO 選擇無上拉，如下圖步驟 5.
(Step 6) 設定 GPIO 之 Port-C 模組初始化，並寫入暫存器，如下圖步驟 6.



4.4 範例程式 gpio

參考 wt32l0xx_pl_gpio.c 之函式 InitialGpio( )下列程式為參照上述 1~6 步驟依序執行

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

  //RCC 開啟 GPIO 模組工作頻率 ，使用 AHB
1 RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIO, ENABLE);

  // set General GPIO pin INPUT
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;        //設定輸入模式
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;        //使用輸入預設上拉，但省電模式會耗電
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;           //選擇 PIN2
  GPIO_Init(GPIOA, &GPIO_InitStructure);              //針對單PORT (A-D) 進行初始化

#if(ENABLE_LED_BLINK==ON)       //判斷是否需輸出燈號
  // set General GPIO pin PC4
```

GPIO_InitTypeDef 參考 CMSIS 定義

```
2.  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;      //選擇PIN4
3.  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;       //設定輸出模式
4.  GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;      //設定開汲極類型
5.  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;    //設定無上、下拉

6.  GPIO_Init(GPIOC, &GPIO_InitStructure);              //針對單PORT-A 進行初始化
    GPIO_SetBits(GPIOC, GPIO_Pin_4);                    //設定 PORTC PIN4 輸出 HI
     //.....省略
  #endif
```

GPIO_SetBits 輸出 HI 電位
GPIO_ResetBits 輸出 LO 電位

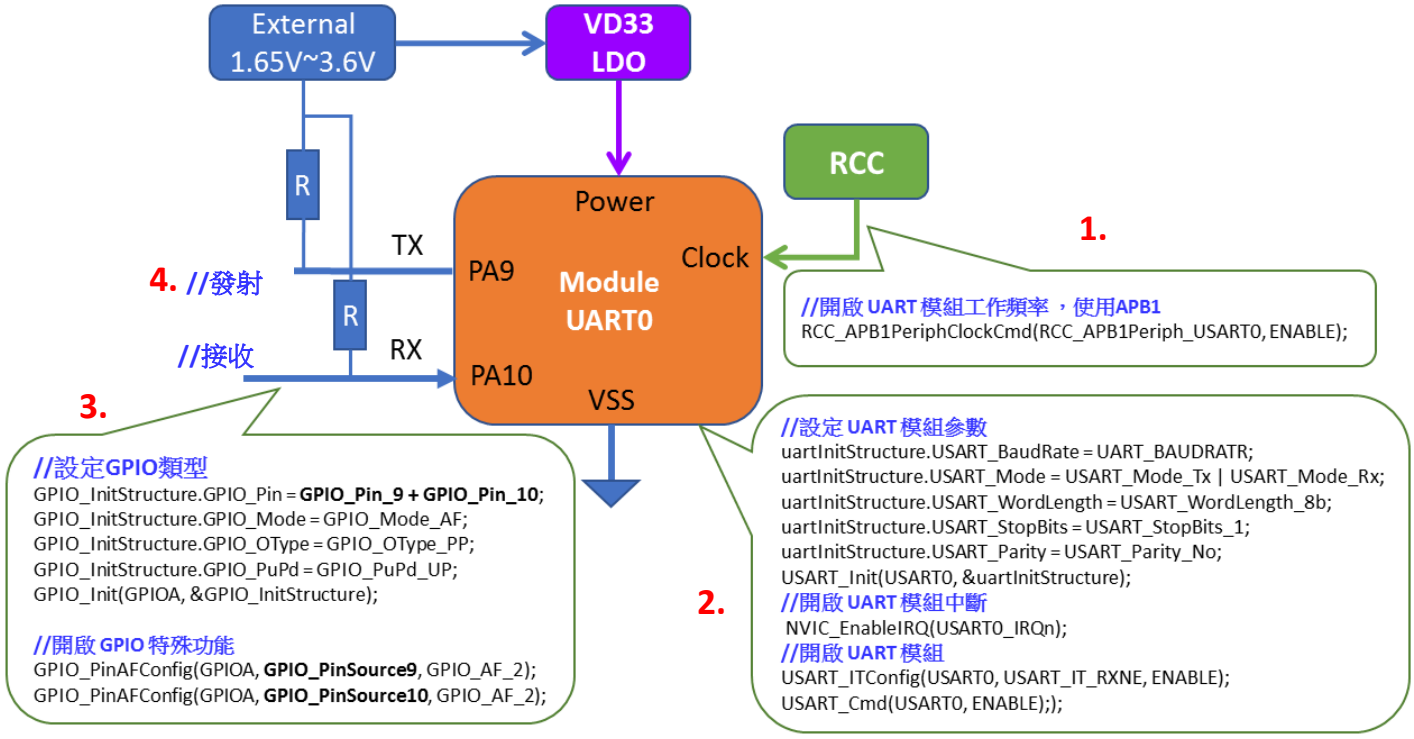
5. UART 功能說明

使用下列圖示說明，使用 UART0 或 UART1 執行資料傳輸，動作流程如下:

5.1 MCU 上電後初始化 UART

如下列 1~4 步驟,可參考周邊程式庫 wt32l0xx_pl_uart.c 使用函式 InitialUart0( )或 InitialUart1( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 UART 使用，如下圖步驟 1.
(Step 2) 設定 UART 模組參數，如下圖步驟 2.
(Step 3) 設定 GPIO 類型 (IO 最後設定，避免信號灌入狀態未定的模組) ，如下圖步驟 3.
(Step 4) 發射 UART 資料，如下圖步驟 4.



5.2 範例程式 uart

參考 wt32l0xx_pl_uart.c 之函式 InitialUart0 ( )，下列程式為參照上述 1~4 步驟依序執行

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

GPIO_InitTypeDef、USART_InitTypeDef 參考 CMSIS 定義

1.
```
//開啟 UART 模組工作頻率 ，使用APB1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART0, ENABLE);        //輸入 APB1 時脈
```

2.

```
//設定 UART 模組參數
uartInitStructure.USART_BaudRate = UART_BAUDRATR;                //設定 Baud Rate
uartInitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;    //設定TX +RX 功能啟用
uartInitStructure.USART_WordLength = USART_WordLength_8b;        //設定 傳輸長度 8bit
uartInitStructure.USART_StopBits = USART_StopBits_1;             //設定1個停止位元
uartInitStructure.USART_Parity = USART_Parity_No;                //設定是否使用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);                  //進行 UART0 初始化

#if(ENABLE_INT_UART0==ON)
USART_ITConfig(USART0, USART_IT_RXNE, ENABLE);           //設定 UART0 中斷類型
NVIC_EnableIRQ(USART0_IRQn);                             //啟動 UART0 中斷功能
#endif
USART_Cmd(USART0, ENABLE);                               //啟動 UART0 模組功能
```

3.

```
//設定GPIO類型
#if(SELECT_UART0_CH_A==ON)                               //若選擇 A 通道
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 + GPIO_Pin_10;  //選定 GPIO 腳位
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;             //使用 AF 類型
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;           //選定 GPIO 推挽式或 開汲極
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;             //選定 GPIO 上拉或下拉類型
GPIO_Init(GPIOA, &GPIO_InitStructure);                   //進行 GPIO 初始化

//開啟 GPIO 特殊功能
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_2);     //選定AF對應功能，有0~5種類
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_2);    //選定AF對應功能，有0~5種類
```

4.

```
//發射
printf("CH-A,Baud=%d,", UART_BAUDRATR);                  //輸出 UART0 資料
#else                                                    //若選擇 B 通道
…… 省略
#endif
```

5.3 UART 進行 RX 接收資料與 TX 發射資料

UART 中斷功能搭配使用 UART0_Handler( ) 進行 RX 接收資料，寫法如下

```
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
}
```

發射資料可搭配 fputc( )使用 ARM 預設 printf( ) ，或是使用範例 DRV_Printf( )輸出資料

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

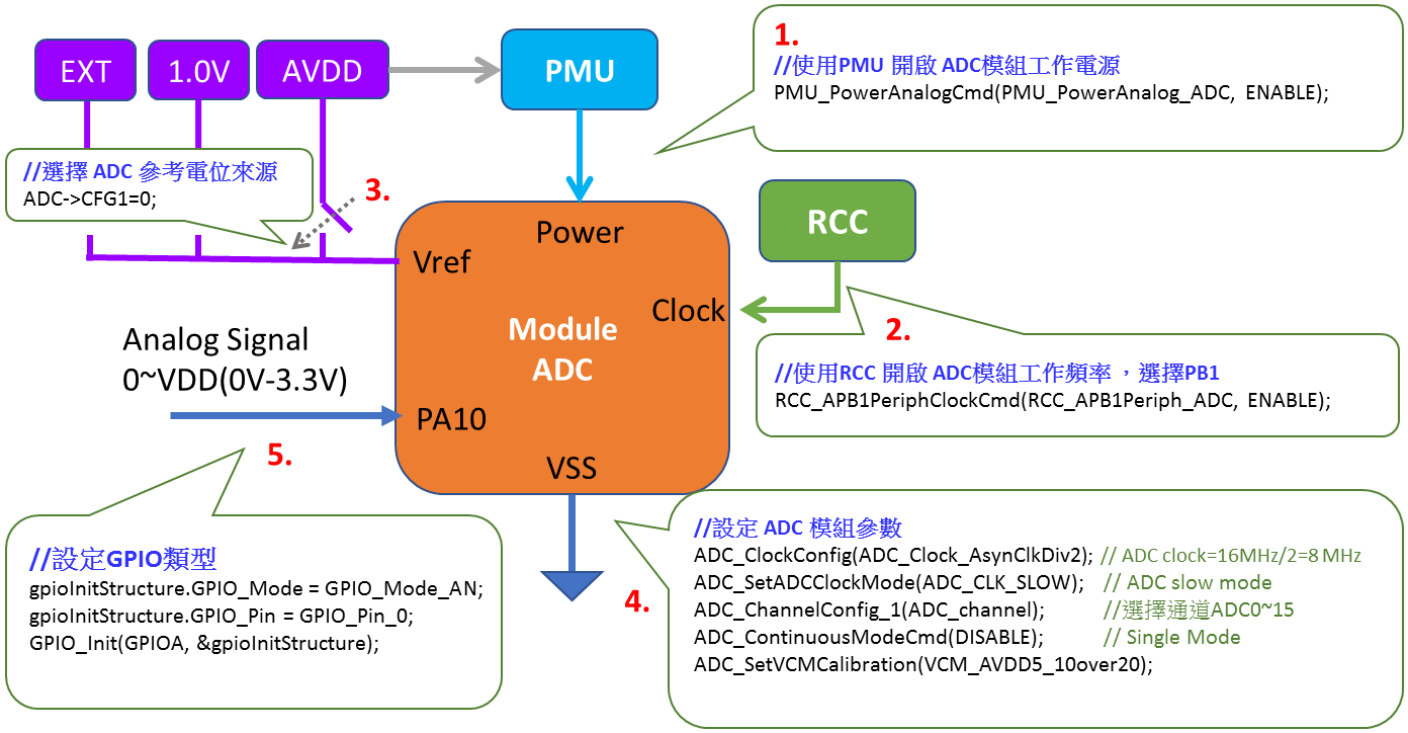
6. ADC 功能說明

使用下列圖示說明，使用 ADC 執行類比信號輸入，動作流程如下:

6.1 MCU 進行 ADC 初始化

MCU 上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_adc.c 使用函式 InitialAdc( )

(Step 1) 設定 PMU(電源管理單元) 開啟類比電源提供給 ADC 使用，如下圖步驟 1.
(Step 2) 設定 RCC (時鐘控制模組) 開啟時脈提供給 ADC 使用，如下圖步驟 2.
(Step 3) 選擇參考電位來源，有 AVDD、B-GAP 1.2V、External Pin 輸入，如下圖步驟 3.
(Step 4) 設定 ADC 模組參數，設定轉換通道、速度，如下圖步驟 4.
(Step 5) 設定 GPIO 類型 (IO 最後設定，避免信號灌入狀態未定的模組)，如下圖步驟 5.



6.2 範例程式 adc

參考 wt32l0xx_pl_adc.c 之函式 InitialAdc ( )下列程式為參照上述 1.~5.步驟依序執行

```
void InitialAdc(uint16_t ADC_channel)
{
  GPIO_InitTypeDef        gpioInitStructure;       // IO 初始化使用，變數結構

1.
  //---------- PMU -----------------------------------------
  // 使用PMU 開啟 ADC模組工作電源
  PMU_PowerAnalogCmd(PMU_PowerAnalog_ADC, ENABLE);        // 供電給 ADC
```

2.

```
//---------- RCC ----------------------------------------
  // 使用RCC 開啟 ADC 模組工作頻率 ，選擇APB1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_ADC, ENABLE);

//---------- ADC ----------------------------------------
ADC_StopOfConversion_1();//先停止ADC轉換，若之前有開啟
```

3.

```
#if(ADC_VREF_SEL_AVDD==ON)            //選擇 ADC 參考電位來源 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);                 // 選擇通道 ADC0~15
ADC_ContinuousModeCmd(DISABLE);                   // Single Mode
ADC_SetVCMCalibration(VCM_AVDD5_10over20);        // 若AVDD小於1.8V需調高 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)       //若開啟使用AWD 類比看門狗
  //…… 省略
#endif

#if(ENABLE_FUNC_DMA==OFF)             //若無使用DMA搬運，則開啟中斷並判斷轉換完成
  ADC->CFG1 |= 0x00000200;       // Enable ADC  interrupt
  ADC_ITConfig(ADC_IT_EOC, ENABLE);
  NVIC_EnableIRQ(ADC_IRQn);   // ADC  interrupt enable
#endif
```

5.

```
//-------------------------------------------------
// ADC 通道設定，依其PA~PC分不同進行 IO/Analog 切換
  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，設定 ADC
    }
  //·····以下IO設定省略
  }
```

6.3 進行 ADC 偵測與轉換資料

範例程式如下:

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

    ADC_StopOfConversion_1();           //先停止ADC 轉換
    ADC_ChannelConfig_1(ADC_channel);   // 選擇 ADC_通道，channel enable
    __nop();  __nop();  __nop();  __nop();
    gu16AdcFinish = 0;                  //清除轉換旗標

    ADC_StartOfConversion_1();          //啟動 ADC 轉換

    while (gu16AdcFinish == 0);         //等待ADC 轉換完成 旗標 設立
    AD_buff = ADC_GetConversionValue(); // 取出 ADC 轉換 數值
    return AD_buff;
}
```

此行函式為寫入暫存器命令:
ADC->ADCCR |= (uint32_t)ADC_START;

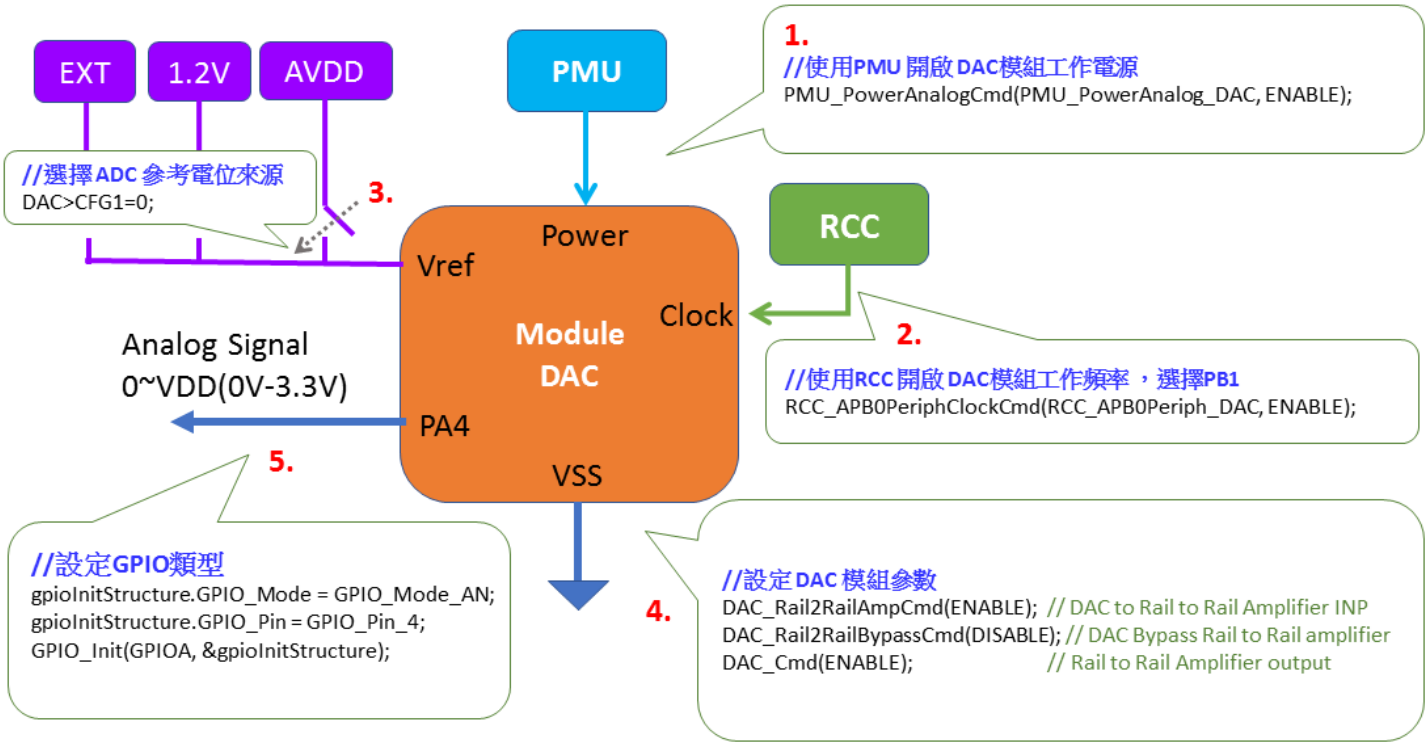
7. DAC 功能說明

使用下列圖示說明，使用 DAC 執行類比信號輸入，動作流程如下:

7.1 MCU 進行 DAC 初始化

上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_dac.c 使用函式 InitialDac( )

(Step 1) 設定 PMU(電源管理單元) 開啟類比電源提供給 DAC 使用，如下圖步驟 1.
(Step 2) 設定 RCC (時鐘控制模組) 開啟時脈提供給 DAC 使用，如下圖步驟 2.
(Step 3) 選擇參考電位來源，有 AVDD、B-GAP 1.2V、External Pin 輸入，如下圖步驟 3.
(Step 4) 設定 DAC 模組參數，設定轉換通道、速度，如下圖步驟 4.
(Step 5) 設定 GPIO 類型 (IO 最後設定，避免信號灌入狀態未定的模組)，如下圖步驟 5.



7.2 範例程式 dac

參考 wt32l0xx_pl_dac.c 之函式 InitialDac( )，下列程式為參照上述 1~5 步驟依序執行:

```
void InitialDac(uint16_t DAC_channel)
{
 //----------- PMU ----------------------------------------
 // 使用PMU 開啟 DAC 模組工作電源
 PMU_PowerAnalogCmd(PMU_PowerAnalog_DAC, ENABLE);   //供給電源
```

1.

```
//----------- RCC ----------------------------------------
// 使用RCC 開啟 DAC 模組工作頻率 ，選擇APB1
RCC_APB0PeriphClockCmd(RCC_APB0Periph_DAC, ENABLE);     //供給頻率

//----------- DAC ----------------------------------------
DAC_DeInit();       //先清除 DAC 舊設定

#if(DAC_VREF_SEL_1P2==ON)
 DAC->CFG &= ~BIT2;        //使用 BG1P0V 作為 參考電源

#elif(DAC_VREF_SEL_EXT==ON)
 DAC->CFG &= ~BIT3;        //使用 外部IO 作為 參考電源

 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 作為 參考電源
#endif

 DAC_Rail2RailAmpCmd(ENABLE);            // DAC to Rail to Rail Amplifier INP
 DAC_Rail2RailBypassCmd(DISABLE);    // DAC Bypass Rail to Rail amplifier
 DAC_Cmd(ENABLE);                    // 開啟輸出, 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);
}
```

7.3 進行 DAC 資料轉換輸出

範例程式如下:

```
uint32_t DAC_Convert(uint16_t DAC_channel,uint32_t u32DacOut)
{
 DAC_SetInputData(u32DacOut);   //輸出類比信號 DAC->CVTD

 return    u32DacOut;
}
```

此行函式為寫入暫存器命令:
DAC->CVTD = Data;

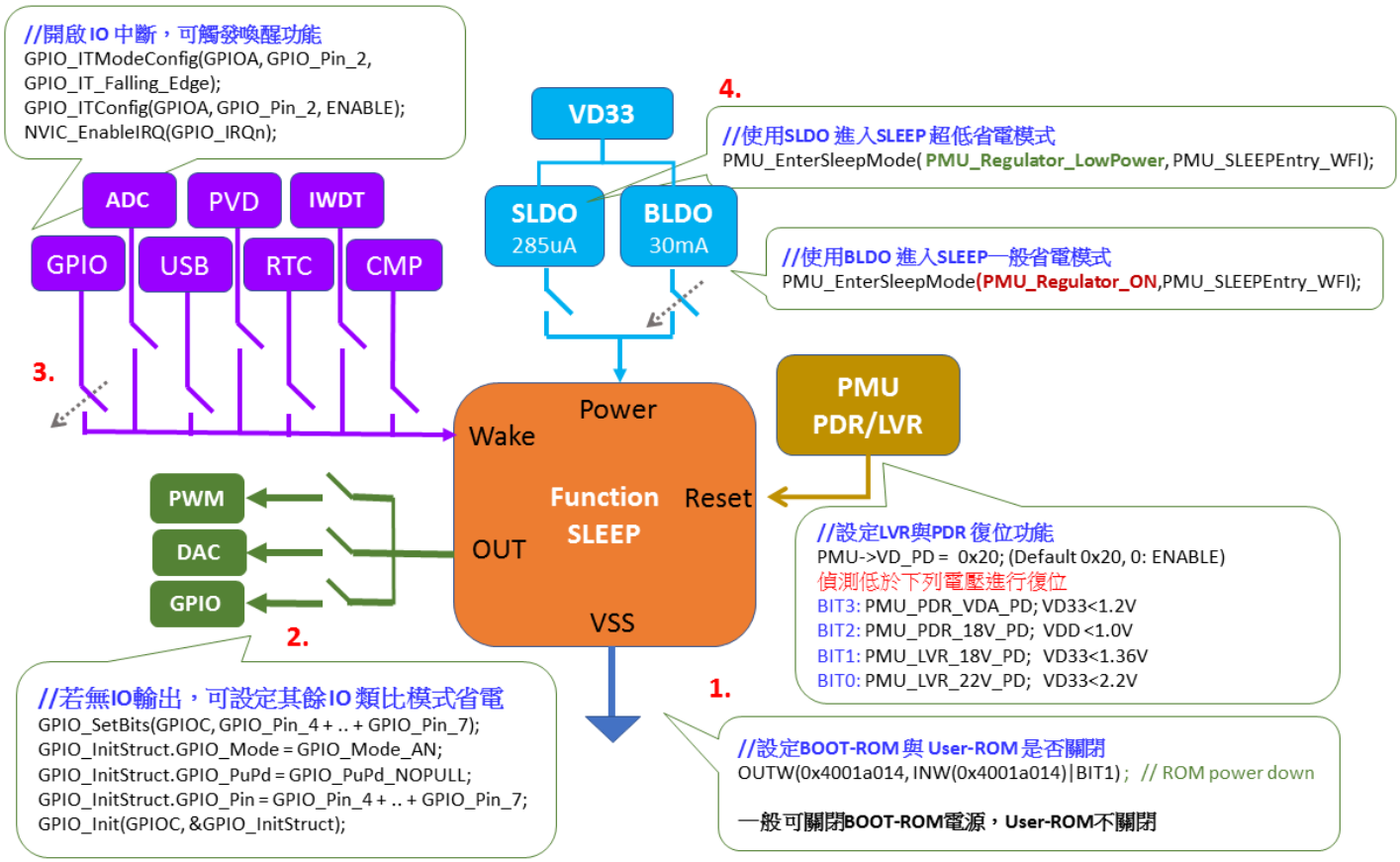
8. SLEEP 功能說明

使用下列圖示說明，使用 SLEEP 進入省電模式，動作流程如下:

8.1 MCU 進行 SLEEP 初始化

上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_save.c 使用函式 save( )

(Step 1) 設定 Boot-ROM 電源關閉，進入 SLEEP 時無使用 ISP 功能，如下圖步驟 1.
(Step 2) 設定 GPIO 類型，無使用 IO 設成類比型態(Analog Mode)，如下圖步驟 2.
(Step 3) 設定 GPIO 喚醒，所有的 GPIO 都可設定觸發喚醒 SLEEP，如下圖步驟 3.
(Step 4) 進入 SLEEP 模式，可依耗電情況選用低功耗與一般省電，如下圖步驟 4.



8.2 範例程式 save.c

參考 wt32l0xx_pl_save.c 之函式 save()，下列程式為參照上述 1~4 步驟依序執行

```
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);      //將 LED 熄滅
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                 //使用類比模式 可以省電
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;       //輸入模式下使用 pull-up 會增加耗電
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;     //PIN 選用
  GPIO_Init(GPIOC, &GPIO_InitStruct);                                          //進行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 做喚醒 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 做喚醒 IO 使用
//……省略
#endif
#endif

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

#if(ENABLE_WAKEUP_IWDT==ON)      //省電時仍開啟IWDT
  PMU_StopModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
  PMU_StandbyModeAutoPowerCmd(PMU_StandbyAutoPower_LSI, ENABLE);
  IWDT_ReloadCounter();
#endif

  //----------------- Sleep ----------------------------------
#if(ENABLE_SLEEP_MODE==ON)       //systick 必須關閉否則會喚醒
if (nMode == SAVE_MODE_SLEEP)
{

#if(ENABLE_FUNC_SYSTICK==ON)
```

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

//進入 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.

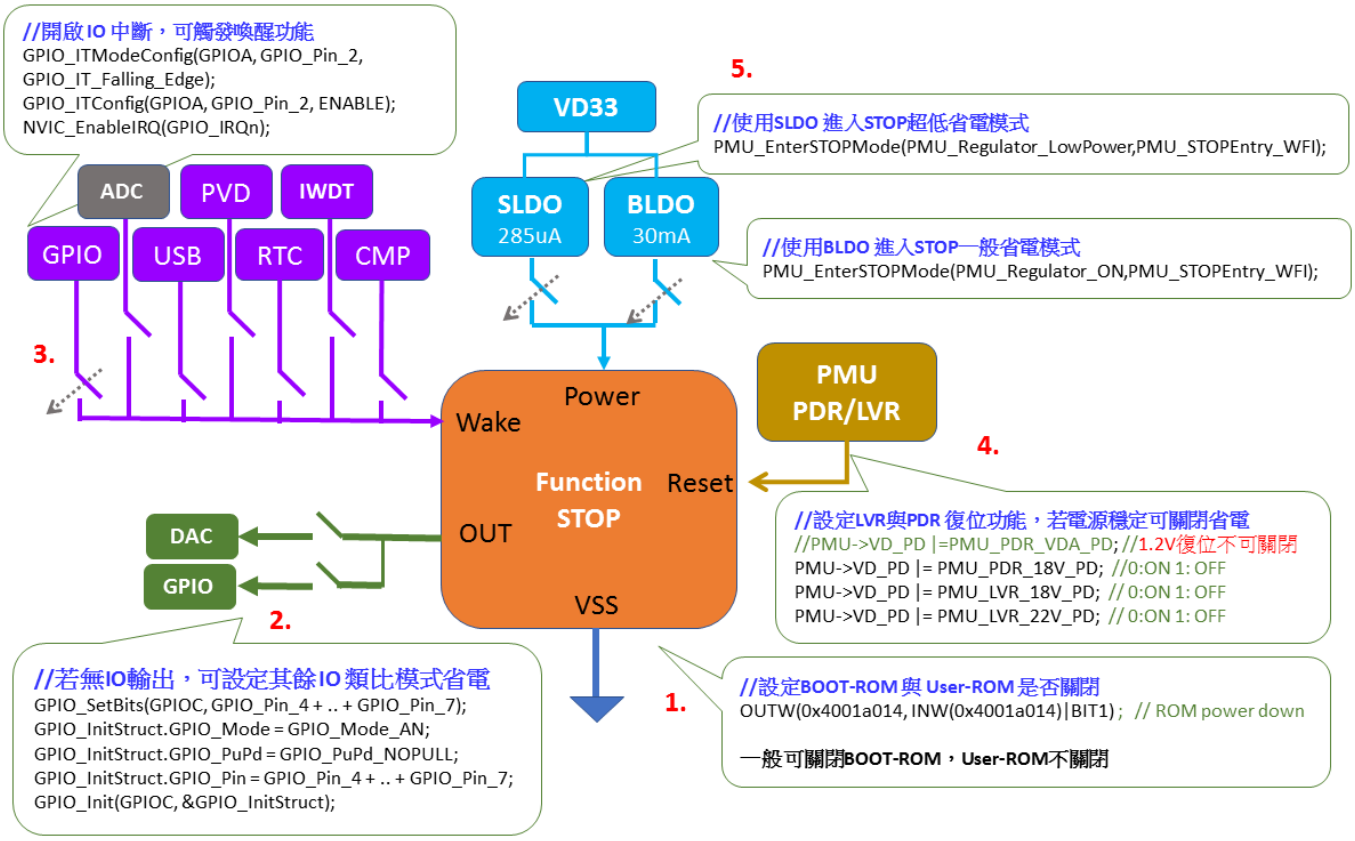
9. STOP 功能說明

使用下列圖示說明，使用 STOP 進入省電模式，動作流程如下:

9.1 MCU 進行 STOP 初始化

上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_save.c 使用函式 save( )

(Step 1) 設定 Boot-ROM 電源關閉，進入 STOP 時無使用 ISP 功能，如下圖步驟 1.
(Step 2) 設定 GPIO 類型，無使用 IO 設成類比型態(Analog Mode)，如下圖步驟 2.
(Step 3) 設定 GPIO 喚醒，所有的 GPIO 都可設定觸發喚醒 STOP，如下圖步驟 3.
(Step 4) 設定 PDR/LVR 復位，若電源穩定可將 LVR 關閉省電，PDR 建議開啟，如下圖步驟 4.
(Step 5) 進入 STOP 模式，可依耗電情況選用低功耗與一般省電，如下圖步驟 5.



9.2 範例程式 save

參考 wt32l0xx_pl_save.c 之函式 save()，下列程式為參照上述 1.~5.步驟依序執行

```
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);      //將 LED 熄滅
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                 //使用類比模式 可以省電
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;       //輸入模式下使用 pull-up 會增加耗電
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;     //PIN 選用
    GPIO_Init(GPIOC, &GPIO_InitStruct);                                               //進行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 做喚醒 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 做喚醒 IO 使用
//……省略
#endif
#endif

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

#if(ENABLE_WAKEUP_IWDT==ON)        //省電時仍開啟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);          //自動關閉 DAC 模組耗電
     #endif

     #if(ENABLE_FUNC_ADC==ON)
        PMU->ATPD_STOP &= (~PMU_StopModeAutoPower_ADC);          //自動關閉 ADC 模組耗電
     #endif

     #if(ENABLE_FUNC_LSI==ON)
        PMU->ATPD_STOP &= (~PMU_STOP_LSI_PD);                    //自動關閉 LSI 模組耗電
     #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);    //自動關閉 COMP 模組耗電
     #endif

//進入 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.

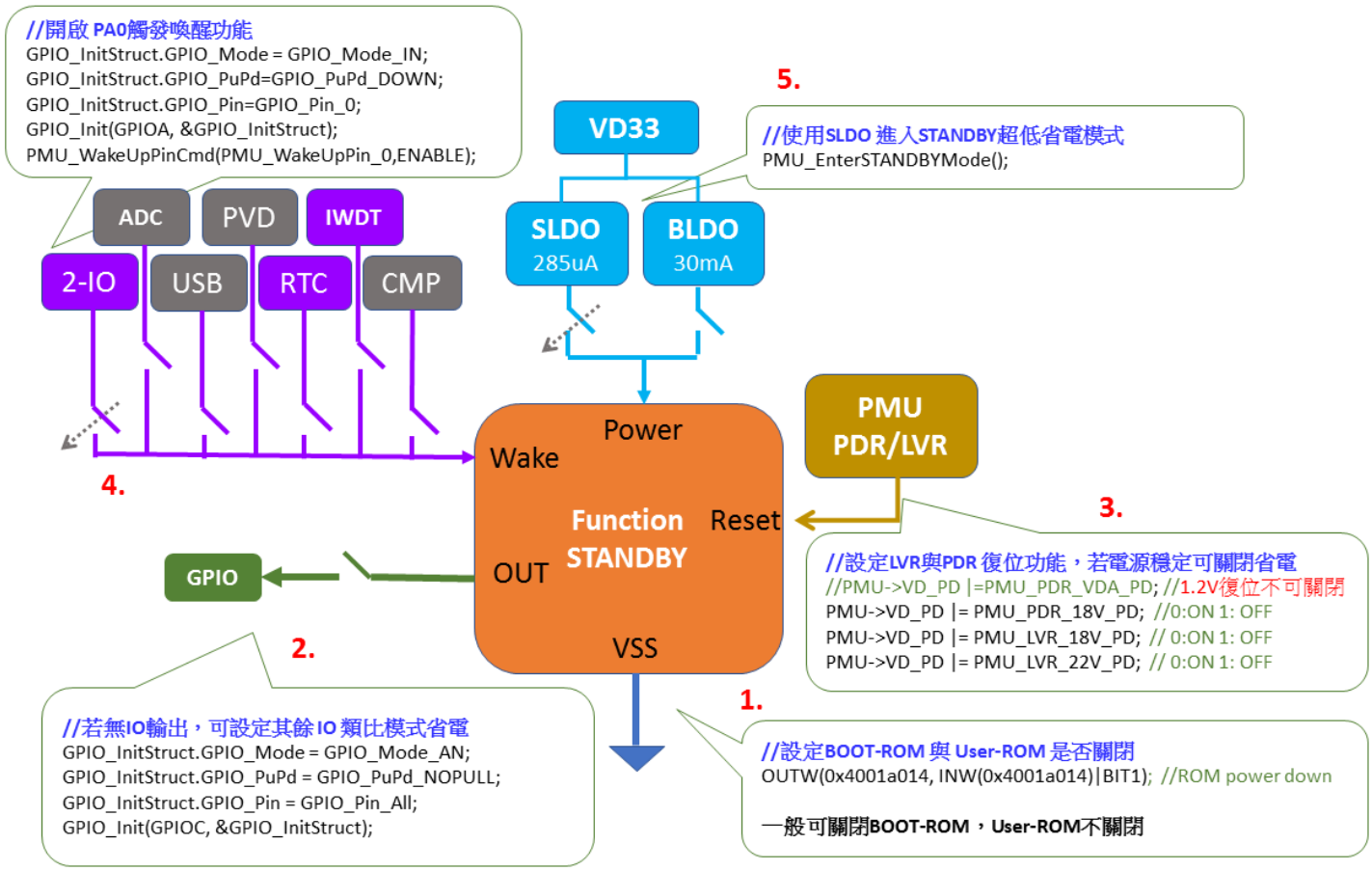
10. STANDBY 功能說明

使用下列圖示說明，使用 STANDBY 進入省電模式，動作流程如下:

10.1 MCU 進行 STANDBY 初始化

上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_save.c 使用函式 save( )

(Step 1) 設定 Boot-ROM 電源關閉，進入 STANDBY 時無使用 ISP 功能，如下圖步驟 1.
(Step 2) 設定 GPIO 類型，無使用 IO 設成類比型態(Analog Mode)，如下圖步驟 2.
(Step 3) 設定 GPIO 喚醒，有兩組 GPIO 都可設定觸發喚醒 STANDBY，如下圖步驟 3.
(Step 4) 設定 PDR/LVR 復位，若電源穩定可將 LVR 關閉省電，PDR 建議開啟，如下圖步驟 4.
(Step 5) 進入 STANDBY 模式，可依耗電情況選用低功耗與一般省電，如下圖步驟 5.



10.2 範例程式 save

參考 wt32l0xx_pl_save.c 之函式 save()，下列程式為參照上述 1.~5.步驟依序執行

```
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);      //將 LED 熄滅
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AN;                 //使用類比模式 可以省電
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;       //輸入模式下使用 pull-up 會增加耗電
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 + GPIO_Pin_5 + GPIO_Pin_6 + GPIO_Pin_7;     //PIN 選用
    GPIO_Init(GPIOC, &GPIO_InitStruct);                                              //進行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 做喚醒 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 做喚醒 IO 使用
  //……省略
  #endif
#endif

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

#if(ENABLE_WAKEUP_IWDT==ON)          //省電時仍開啟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.  //進入 STANDBY 模式
    PMU_EnterSTANDBYMode();

}
#endif
```

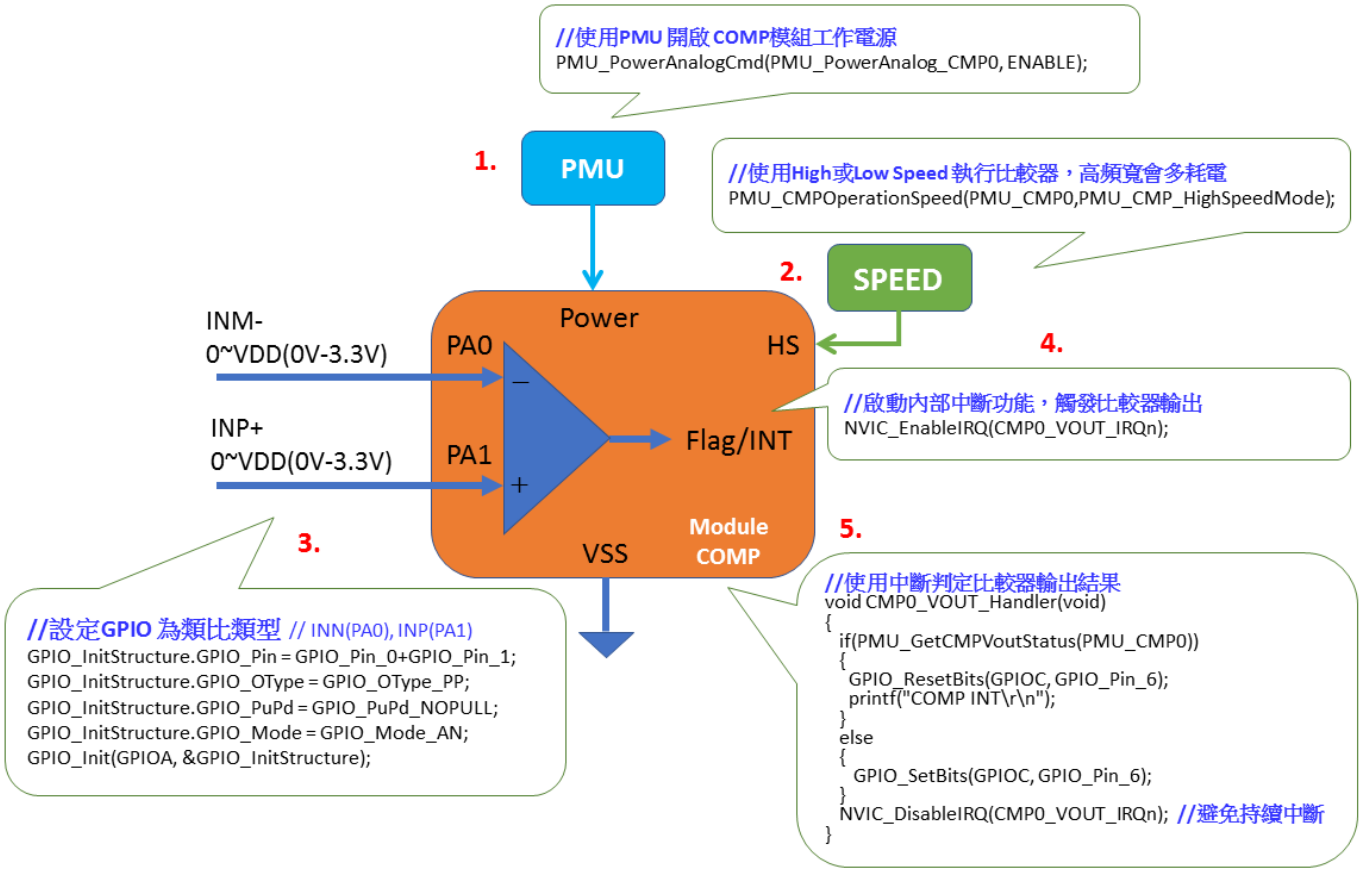
11. COMPARATOR 功能說明

使用下列圖示說明，使用比較器(COMP)執行類比信號輸入，動作流程如下:

11.1 MCU 進行 Comparator 初始化

上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_comp.c 使用函式 InitialComp ( )

(Step 1) 設定 PMU(電源管理單元) 開啟類比電源提供給 COMP 使用，如下圖步驟 1.
(Step 2) 設定 RCC (時鐘控制模組) 開啟時脈提供給 COMP 使用，如下圖步驟 2.
(Step 3) 設定 GPIO 類型 (IO 最後設定，避免信號灌入狀態未定的模組)，如下圖步驟 3.
(Step 4) 設定 COMP 模組中斷功能，當輸入電位 INP>INM 時觸發，如下圖步驟 4.
(Step 5) 當 INP>INM 時觸發中斷，輸出的結果也可用 PMU_GetCMPVoutStatus()讀出



11.2 範例程式 comp

參考 wt32l0xx_pl_comp.c 之函式 InitialComp ( )，參照上述 1.~5.步驟依序執行

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

1.  #if(ENABLE_HW_CMP0==ON)  //開啟COMP_0
      //使用PMU 開啟 COMP 模組工作電源
```

```
        PMU_PowerAnalogCmd(PMU_PowerAnalog_CMP0, ENABLE);


        //使用High或Low Speed 執行比較器，高頻寬增加耗電
2.  #if(ENABLE_HW_CMP_SPEED_HI==ON)
        PMU_CMPOperationSpeed(PMU_CMP0, PMU_CMP_HighSpeedMode);
    #else
        PMU_CMPOperationSpeed(PMU_CMP0, PMU_CMP_LowSpeedMode);
    #endif

        // 設定GPIO 為類比類型  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)  //開啟COMP_1
    //……省略
    #endif
}
```

11.3 Comparator 之中斷功能

範例程式 comp.c 之中斷函式 CMP0_VOUT_Handler ( )，可對照上述 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，避免持續中斷

    }
```

進入中斷後，再讀取比較器結果

12. FLASH 讀寫功能說明

使用下列圖示說明，使用 IC 內部 FLASH 執行讀寫資料，一次完整寫入與讀出動作流程如下:

12.1 MCU 進行 FLASH 初始化

上電後要更新 FLASH 資料，須將資料抹除成 0xFF 後才能對其內容寫入，可參考周邊程式庫使用函式 RunFlash ( )。

(Step 1) 解開 FLASH 保護鎖，如下圖步驟 1.
(Step 2) 設定預計寫入的位址，並先清除該頁資料大小為 1KB.，如下圖步驟 2.
(Step 3) 寫入 WORD 資料 ，使用 CMSIS 呼叫 FLASH_ProgramWord()，該函式使用 __IO 定址 ROM 空間，例如: *(__IO uint32_t *)Address = Data，如下圖步驟 3.
(Step 4) 檢查 WORD 資料 ，直接使用 __IO 定址 ROM 空間，例如: Data= *(__IO uint32_t *)Address ;，如下圖步驟 4.
(Step 5) 將結果使用 UART 輸出，如下圖步驟 5.

參考 flash.c 內 RunFlash ()的寫法步驟
1. flash.c->FLASH_Unlock ()
2. flash.c->FLASH_ErasePage ()
3. flash.c->FLASH_ProgramWord ()
4. data= *(__IO uint32_t*) Address;

12.2 範例程式 flash

參考 wt32l0xx_pl_flash.c 之函式 RunFlash ( )，參照上述 1.~5.步驟依序執行

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

```
#if(SYS_CLOCK_SEL!=CLK_MSI)
    FLASH_SetLatency(1);//若系統頻率 >=16 Mhz     // Set latency
#endif
    FLASH_ClearFlag(FLASH_FLAG_EOP | /*FLASH_FLAG_PGERR |*/ FLASH_FLAG_WRPERR);
1.  //==================== Unlock  FLASH   ============================//
    FLASH_Unlock();//解除 FLASH 防寫鎖

    /* 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));//頁清除
        if (FLASHStatus != FLASH_COMPLETE) //若清除失敗，輸出數值並終止
        {
            uint16_t readout = *(__IO uint16_t*)(WRITE_START_ADDR + (FLASH_PAGE_SIZE * EraseCounter));
                printf("Page=0x%d,", START_ADDR_PAGE + EraseCounter);   //讀值並顯示
            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);//寫入 WORD 資料
        }
```

1.Unlock 解鎖

2. Page Erase 抹除

3. Program 寫入

```
        u32TargetStartAddr += FLASH_PAGE_SIZE;
        u32TargetEndAddr += FLASH_PAGE_SIZE;
        Page++;                 //頁絕對位址
        PageCnt++;//頁計數
    }
    if (FLASHStatus == FLASH_COMPLETE)      printf("Program Done\r\n");
    else                                    printf("Program Fail, Page=%d\r\n", Page - 1);


    //--------------- Test Lock (測試防寫)---------
    //……省略


    //=================== Verify  FLASH  ============================//
    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))            //檢測之前寫的數值 正確否 ?
            {
                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++;                 //頁絕對位址
        PageCnt++;  //頁計數
    }


    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.

4. Verify 檢查

5.

5. Result 輸出結果

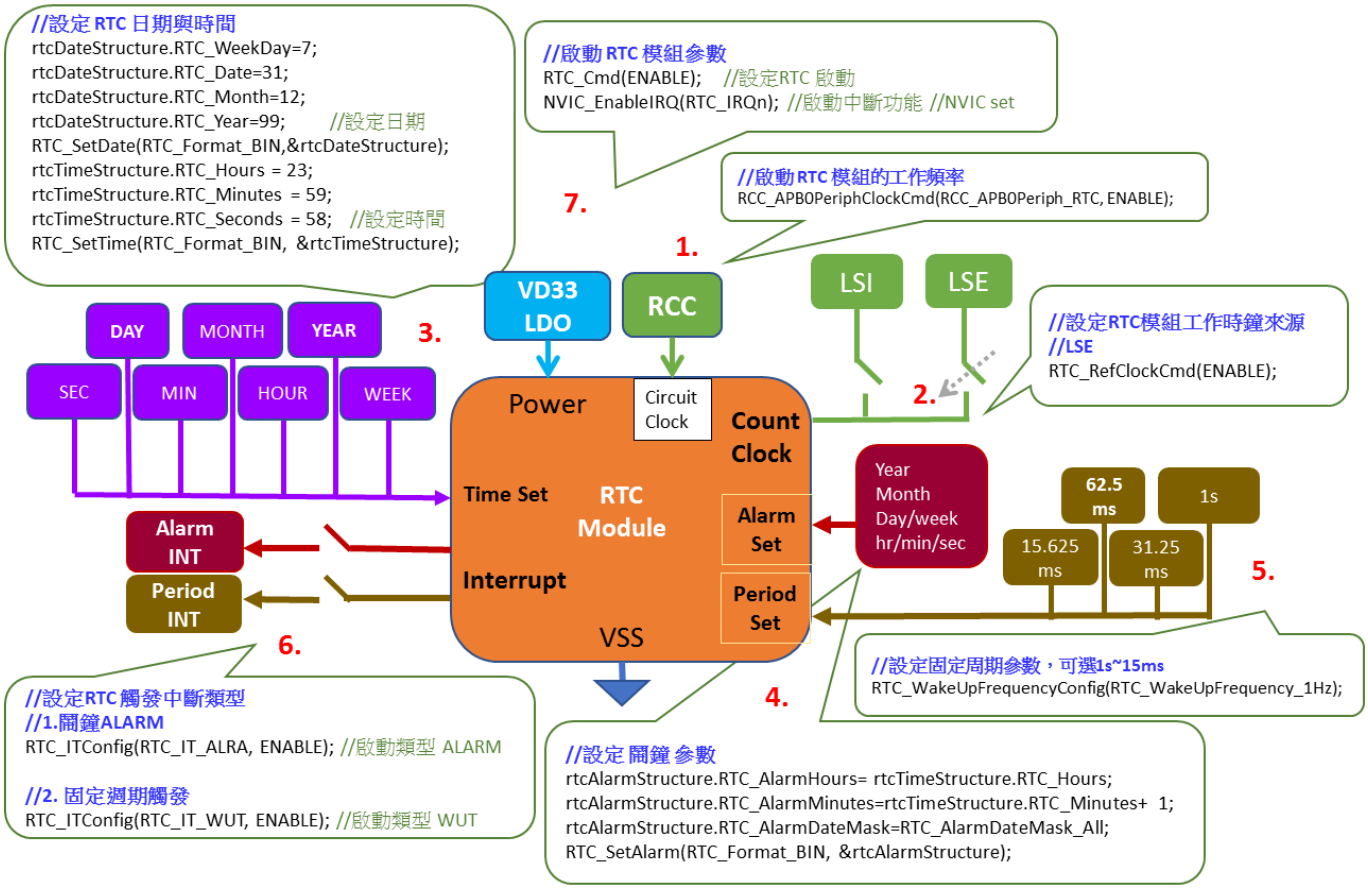
13. RTC 功能說明

使用下列圖示說明，使用實時計數器(RTC)執行數位信號輸入，動作流程如下:

13.1MCU 進行 RTC 初始化

上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_rtc.c 使用函式 InitialRtc ( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 RTC 使用，如下圖步驟 1.
(Step 2) 選用參考鐘源 LSI (37KHz)或 LSE (32.768KHz)，如下圖步驟 2.
(Step 3) 設定 RTC 目前日期與時間，如下圖步驟 3.
(Step 4) 設定鬧鐘時間當與目前時間相同時可觸發 Alarm 中斷，如下圖步驟 4.
(Step 5) 設定週期時間可觸發 WUT 中斷，有 1sec~15msec 可以選，如下圖步驟 5.
(Step 6) 設定中斷開關，有 Alarm 與周期兩種中斷類型可選擇，如下圖步驟 6.
(Step 7) 啟動 RTC 功能並開啟 NVIC 中斷總開關，如下圖步驟 7.



13.2 範例程式 rtc

參考 wt32l0xx_pl_rtc.c 之函式 InitialRtc ( )，參照上述 1.~6.步驟依序執行

```
void InitialRtc(void)
{   RCC_APB0PeriphClockCmd(RCC_APB0Periph_RTC, ENABLE);     // 需先開啟 APB0 clock 才可設定 RTC
1.  RTC_WriteReadProtectionCmd(DISABLE);  //RTC 保護開關，更改設定前要關閉
    RTC_DeInit();                                                          //清除 RTC 設定
2.  RTC_RefClockCmd(ENABLE);                                  //參考外部時鐘源 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);//設定日期於 RTC 模組

    rtcTimeStructure.RTC_Hours = 23;
    rtcTimeStructure.RTC_Minutes = 59;
    rtcTimeStructure.RTC_Seconds = 58;
3.  RTC_SetTime(RTC_Format_BIN, &rtcTimeStructure);    //設定時間於 RTC 模組

    rtcLastTime.RTC_Hours = 0;          //測試記錄用
    rtcLastTime.RTC_Minutes = 0;   //測試記錄用
    rtcLastTime.RTC_Seconds = 0;   //測試記錄用


    //------------------    RTC 鬧鐘 (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); //啟動中斷子類型 ALARM
    #else
5.      RTC_WakeUpFrequencyConfig(RTC_WakeUpFrequency_1Hz);
6.      RTC_ITConfig(RTC_IT_WUT, ENABLE); //啟動中斷子類型 WUT，每(秒/ms)週期觸發
    #endif

7.      RTC_Cmd(ENABLE);                                  //設定RTC 啟動
        NVIC_EnableIRQ(RTC_IRQn);           //啟動中斷功能 //NVIC set

    }
```

13.3 設定 RTC 時間

當設定時間觸發動作，範例程式 rtc.c 之中斷函式 RTC_Handler ( )

```
void RTC_Handler(void)
{
    RTC_ClearITPendingBit(RTC_IT_ALRA + RTC_IT_WUT);   //清除硬體旗標
    RTC_ITConfig(RTC_IT_ALRA, DISABLE);// // 若無中斷需求可關閉中斷
    gbRtcInt = 1;   //變數設 1

//……可自行增加
}
```

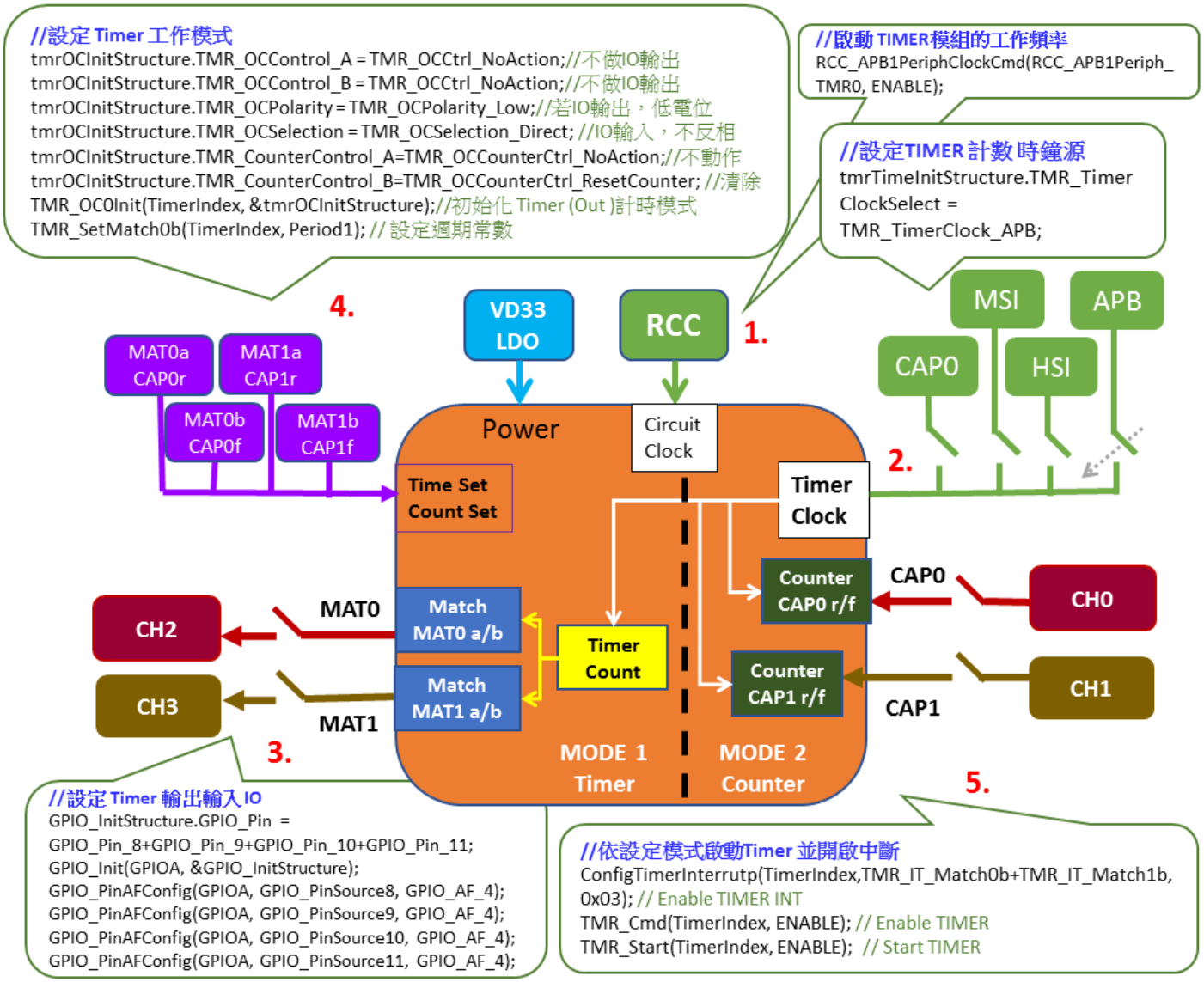
14. TIMER 功能說明

使用下列圖示說明，使用計數計時器(TIMER)執行數位信號輸入與輸出，動作流程如下:

14.1 MCU 進行 Timer 初始化

上電後初始化其內容如下，可參考周邊程式庫 wt32l0xx_pl_timer.c 使用函式 ConfigTimerClockGpio ( )、 ConfigTimerTimeMode ( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 Timer 電路使用，如下圖步驟 1.
(Step 2) 設定輸入時鐘來源，提供給 Timer 計算使用，如下圖步驟 2.
(Step 3) 設定 GPIO 類型，1 組 Timer 輸出有兩路，輸入有兩路，如下圖步驟 3
(Step 4) 設定 Timer 週期時間參數可觸發中斷與輸出信號，如下圖步驟 4.
(Step 5) 設定中斷開關，並依設定參數、計時或計數模式開啟 Timer，如下圖步驟 5.



14.2 範例程式 timer

參考 wt32l0xx_pl_timer.c 之函式 ConfigTimerClockGpio ( )、ConfigTimerTimeMode( )，參照上述 1.~5.步驟依序執行

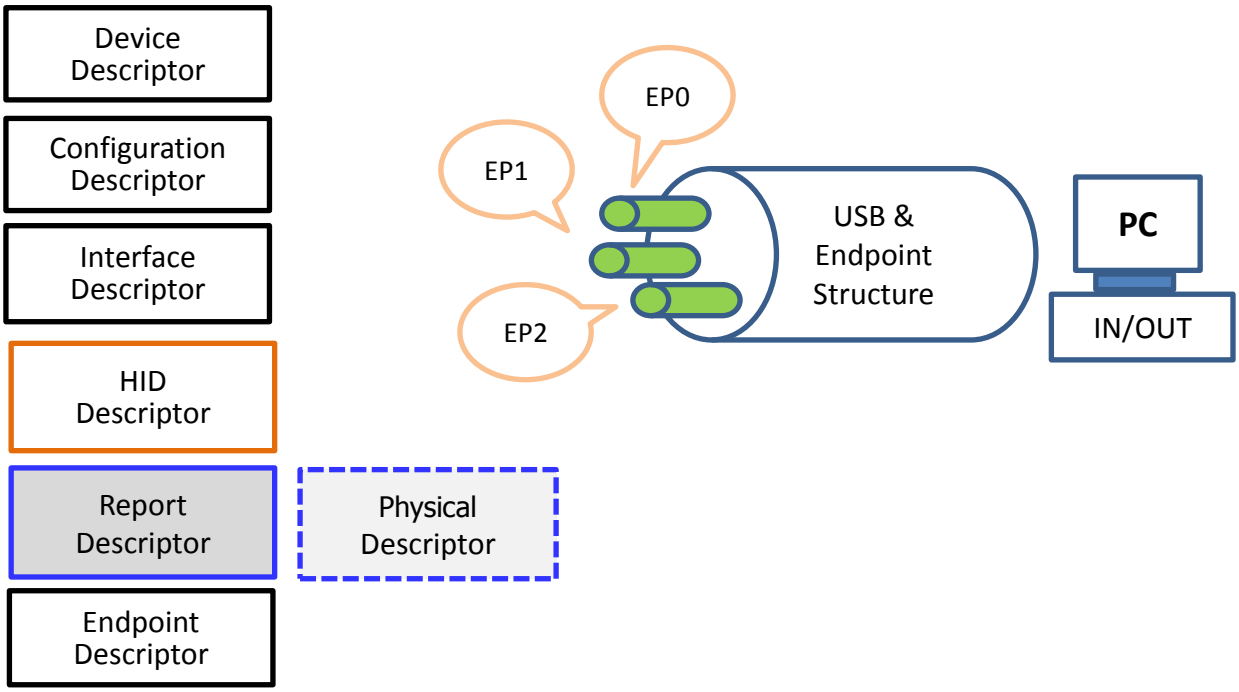
```
void ConfigTimerClockGpio(TMR_TypeDef* TimerIndex, uint32_t nPrescaler, uint16_t nChannelSetSel,
uint16_t nSource)
{
    TMR_TimerInitTypeDef          tmrTimeInitStructure;
    TMR_DeInit(TimerIndex);   //清除設定
    tmrTimeInitStructure.TMR_TimerClockSelect = nSource;    //頻率來源選擇 APB HSI MSI CAP0
    tmrTimeInitStructure.TMR_TimerPrescaler = nPrescaler;        // 除頻 f'=1/ n+1

    //------------ 設定 Timer/GPIO ------------------------------
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIO, ENABLE); // 啟動 GPIO 工作頻率
    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);// 啟動 RCC TIMER clock
        tmrTimeInitStructure.TMR_TimerClockSelect = TMR_TimerClock_APB;   //APB only

2.      TMR_TimerInit(TimerIndex, &tmrTimeInitStructure);//初始化Timer 時鐘源、除頻

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

        if (nChannelSetSel == TMR_PIN_SET0)//使用第1組通道配置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 組通道配置IO
        {
            //.....省略
        }
    }
    else if (TimerIndex == TMR1)
    {
        //.....省略
    }
    else if (TimerIndex == TMR2)
    {
        //.....省略
    }
    TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_NoFilter); // 無使用數位濾波
    //TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_2clks);  // 使用數位濾波 2 clock
    //TMR_ICDigitalFilter(TimerIndex, TMR_ICFilter_4clks)); // 使用數位濾波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輸出
    tmrOCInitStructure.TMR_OCControl_B = TMR_OCCtrl_NoAction;//不做IO輸出

    tmrOCInitStructure.TMR_OCPolarity = TMR_OCPolarity_Low; //若IO輸出，低電位
    tmrOCInitStructure.TMR_OCSelection = TMR_OCSelection_Direct; //IO輸入，不反相
    tmrOCInitStructure.TMR_CounterControl_A = TMR_OCCounterCtrl_NoAction;  //Match後不動作
    tmrOCInitStructure.TMR_CounterControl_B = TMR_OCCounterCtrl_ResetCounter;   //最長 周期
    TMR_OC0Init(TimerIndex, &tmrOCInitStructure);//初始化 Timer (Out )計時模式
    TMR_SetMatch0b(TimerIndex, Period1); // 設定週期常數


    //------------- 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);          // 設定週期常數


    //----------- 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 與 HID 功能說明

15.1 USB-HID 架構說明

如同下圖 USB 的階層描述元(Descriptor)，其中於 Interface 描述元之後可增加 HID 的描述元與報告(Report)描述元，HID 的報告分類共有 3 種 Input、Output 與 Feature 可依需求增加，下圖為標準 USB 與 HID 裝置組態設定。



除了特殊物理裝置需求否則一般 Physical Descriptor 不使用，而 HID 通訊需設定 Report Descriptor，該描述元有下列類型:

1. Input: 周邊裝置傳輸資料至電腦，使用 GET_REPORT 命令格式
2. Output: 電腦傳輸資料至周邊裝置，使用 SET_REPORT 命令格式
3. Feature: 周邊裝置與電腦進行資料交換，使用 GET_REPORT 與 SET_REPORT 命令格式

使用 USB-Endpoint 與 HID- Report 之間的對應關係:

- 使用 Feature 格式進行 HID 通訊是使用 Feature-Report 通過 EP0(Endpoint 0)進行 USB 資料交換，Feature 與 EP0 都是雙向資料傳輸。
- 使用 Input、Output 格式進行 HID 通訊是使用 Input-Report、Output-Report 可選擇通過 EP1~EP6 進行 USB 資料交換，Report 與 EP1~EP6 都是單向資料傳輸，例如 Input-Report 選定 EP2(IN 單向)，而 Output-Report 選定 EP1(OUT 單向)。

15.2 USB-HID 裝置與組態描述元說明

範例程序提供有下列陣列參數供使用者可修改設定，自行定義 HID 傳輸通常需要修改的參數有 EP0_Packet_Size 、VENDOR_ID、PRODUCT_ID、與 endpoints 數量，如下列標示為紅色字體部分，該範例使用 3 個 endpoint，分別為 EP0 作 Feature (雙向)，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 報告描述元與用途頁說明

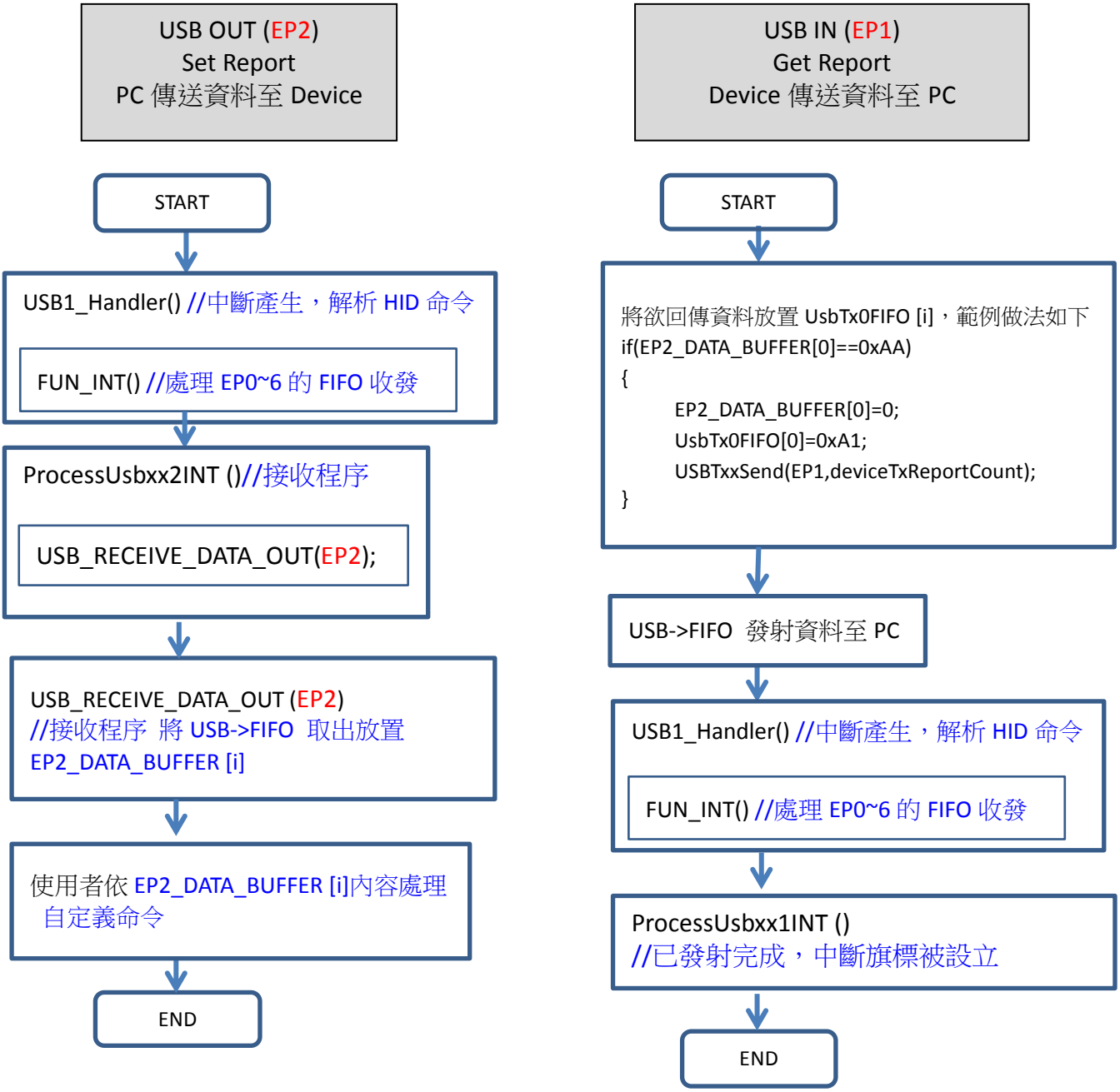
報告描述元其內容包含用途頁(USAGE PAGE)主要設定自訂的傳輸格式、長度與 Report ID，通常 1 組 Interface 配置 1 組 HID_ReportDescriptor，自行定義通常需要修改的參數有 deviceRxReportCount 、FEATURE、REPORT OUTPUT 與 REPORT INPUT 可依需求增列或刪除，如下列標示為紅色字體部分。

```
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) */
```

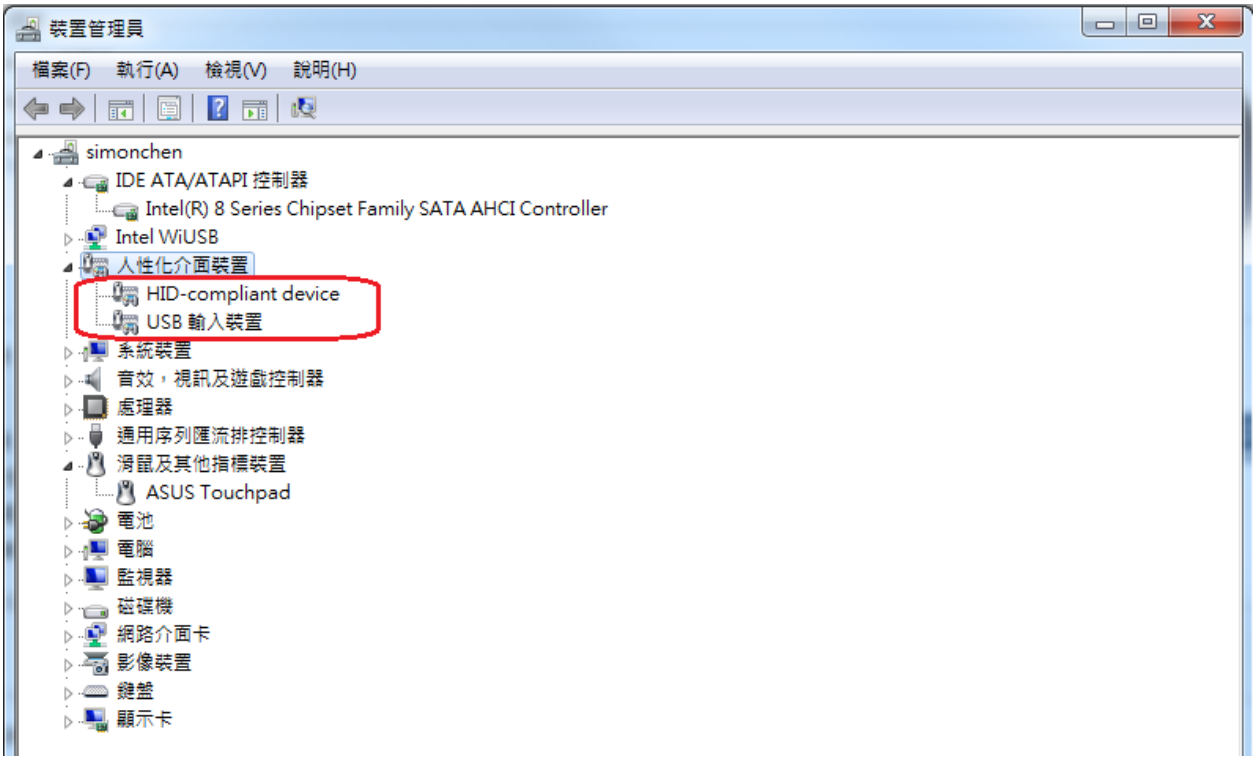
```
    0x09, 0x01,                 /* USAGE, 0x09/0x?? for vendor-defined */
    0x91, 0x82,                 /* REPORT OUTPUT (Data,Var,Abs,Vol)    */
    /* 27 */

    /* 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 發射與接收流程

下列描述裝置端(WT32L064) 於 Set Report、Get Report 作為收發 USB-HID 資料的流程，範例分別使用 EP2 與 EP1。



承上我們使用裝置管理員偵測裝置(WT32L064) 是否具備 USB-HID 功能，插入裝置後會新增目標 USB 裝置，該裝置會列舉在人性介面裝置，如下圖所示。



15.4.1 主機端發射與接收 HID Report 範例

如下表所列，該裝置具有 3 個 endpoint，分別為 EP0 作 Feature (雙向)，EP1 作 Report-IN，EP2 作 Report-OUT，接著我們設定主機端(PC) EP2 將發射的資料為 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 軟體工具，即時偵測 USB 的資料流向，當主機端發出 USB 資料後如下圖所示 EP2 之 OUT，已傳出 0xAA、0x22、0x00….0x00，然後主機端 EP1 會收取到 32 個 Byte 資料為裝置回傳 0xA1、0x00….0x00，此處結果與程式設定相同。

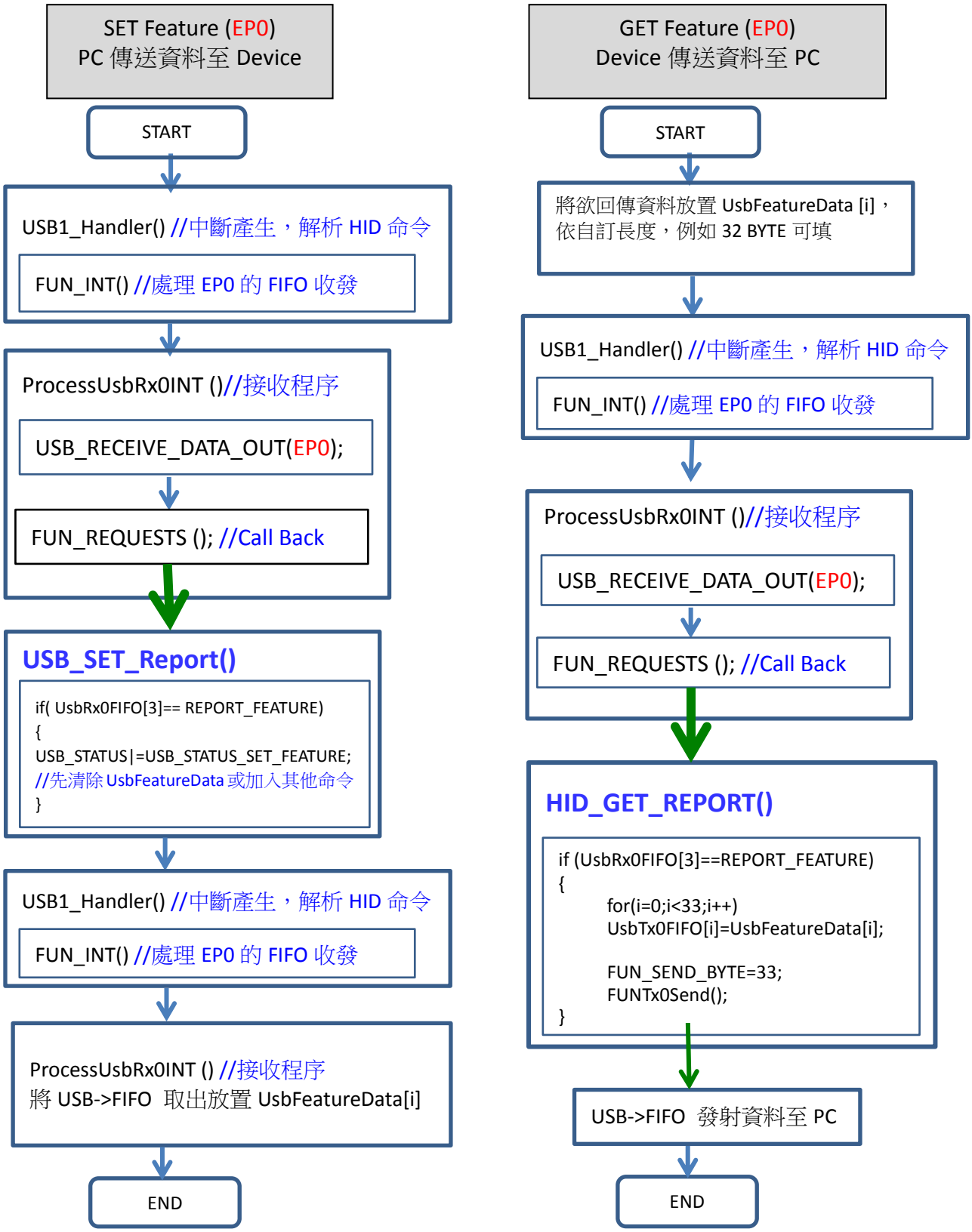
Endpoint	Direction	Data (16 進制)
2	OUT	AA 22 00 00_00 00 00 00_00 00 00 00_00 00 00 00 00 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00
1	IN	A1 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00 00 00 00 00_00 00 00 00_00 00 00 00_00 00 00 00

程式設定當收取到 USB 指令 0xAA 時執行回傳 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;         //設定回傳  0xA1
      USBTxxSend(EP1, deviceTxReportCount);     //執行 USB 的 EP1 發射 FIFO 資料
                                                // deviceTxReportCount=32
}
```

15.5 HID Feature 發射與接收流程

下列描述裝置端於 SET Feature、GET Feature 作為收發 USB-HID 資料的流程，分別使用到 EP0。



15.5.1 HID Feature 接收範例

如下表所列,該裝置具有 3 個 endpoint,分別為 EP0 可作 Feature 雙向溝通,EP1 作 Report-IN，EP2 作 Report-OUT，此處 USB 通訊將 EP0 固定為控制型設定使用，而 SETUP 封包長度固定是 8 Bytes,我們設定主機端(PC) USB 工具之 EP0 將發射的資料為 0xA1、0x01、0x00、0x03、0x00、0x00、0x08、0x00，接著收到裝置(WT32L064)的資料為 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 命令可參考下列格式，其中 0xA1、0x01 為 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)
命令說明	**Get_Report** (Feature Input,使用 EP0)		Report ID=0	Report Type =Feature	Interface No. =0		長度= 8 Bytes	

使用 USB 軟體工具，實際偵測 USB 的資料流向，當按下執行後如下表所示，我們傳出 0xA1、0x01、0x00….0x00，收到了 8 個 Byte 資料為 0x01、0x02、0x03….0x08，此處結果與程式設定相同。

Endpoint	Direction	Data (16 進制)	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 發射範例

如下圖所示，設定 PC 端 USB 工具之 EP0 將發射的資料為 0x21、0x09、0x00、0x03、0x00、0x00、0x08、0x00，接著發射資料為 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 命令可參考下列格式，其中 0x21、0x09 為 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)
命令說明	**Set_Report** (Feature output,使用 EP0)		Report ID=0	Report Type =Feature	Interface No. =0		長度= 8 Bytes	

我們使用 USB 軟體工具，實際偵測 USB 的資料流向，當按下執行後如下圖所示，主機端傳出 0xA1、0x01、0x00….0x00，接續發射 8 個 Byte 資料為 0x11、0x22、0x33….0x88，此處結果與程式設定相同。

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

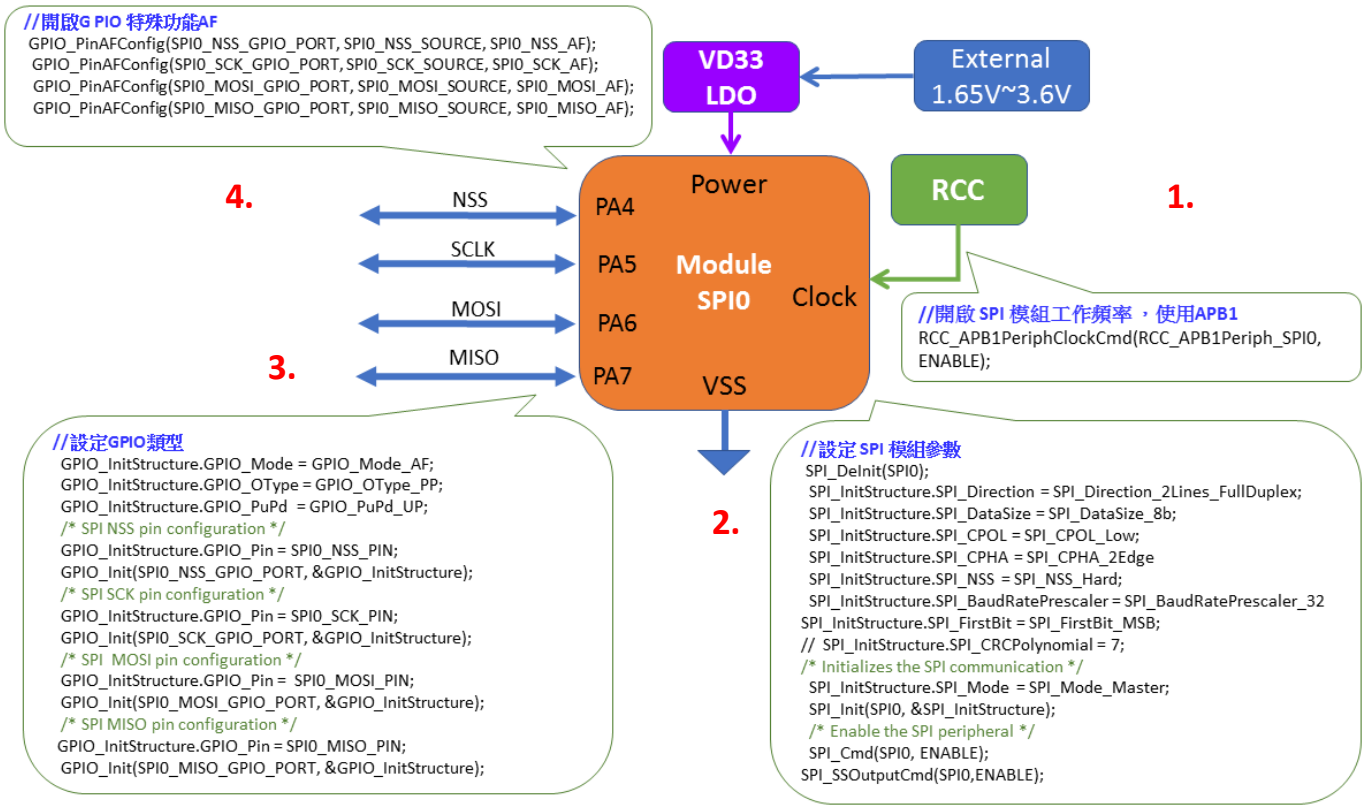
16. SPI 功能說明

使用下列圖示說明，使用 SPI0 或 SPI1 執行資料傳輸，動作流程如下:

16.1 MCU 上電後初始化 SPI

如下列 1~4 步驟，可參考周邊程式庫使用函式 InitialSpi0( )或 InitialSpi1( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 SPI 使用，如下圖步驟 1.
(Step 2) 設定 SPI 模組參數，如下圖步驟 2.
(Step 3) 設定 GPIO 類型，設定推挽式與上拉電阻如下圖步驟 3.
(Step 4) 設定 GPIO 類型，設定 AF3 類別使 IO 具有 SPI 功能，如下圖步驟 4.



16.2 範例程式

參考 wt32l0xx_pl_spi.c 之函式 InitialSpi0 ( )，下列程式為參照上述 1~4 步驟依序執行

```
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 功能說明

使用下列圖示說明，使用 I2C0 或 I2C1 執行資料傳輸，動作流程如下:

17.1 MCU 上電後初始化 I2C

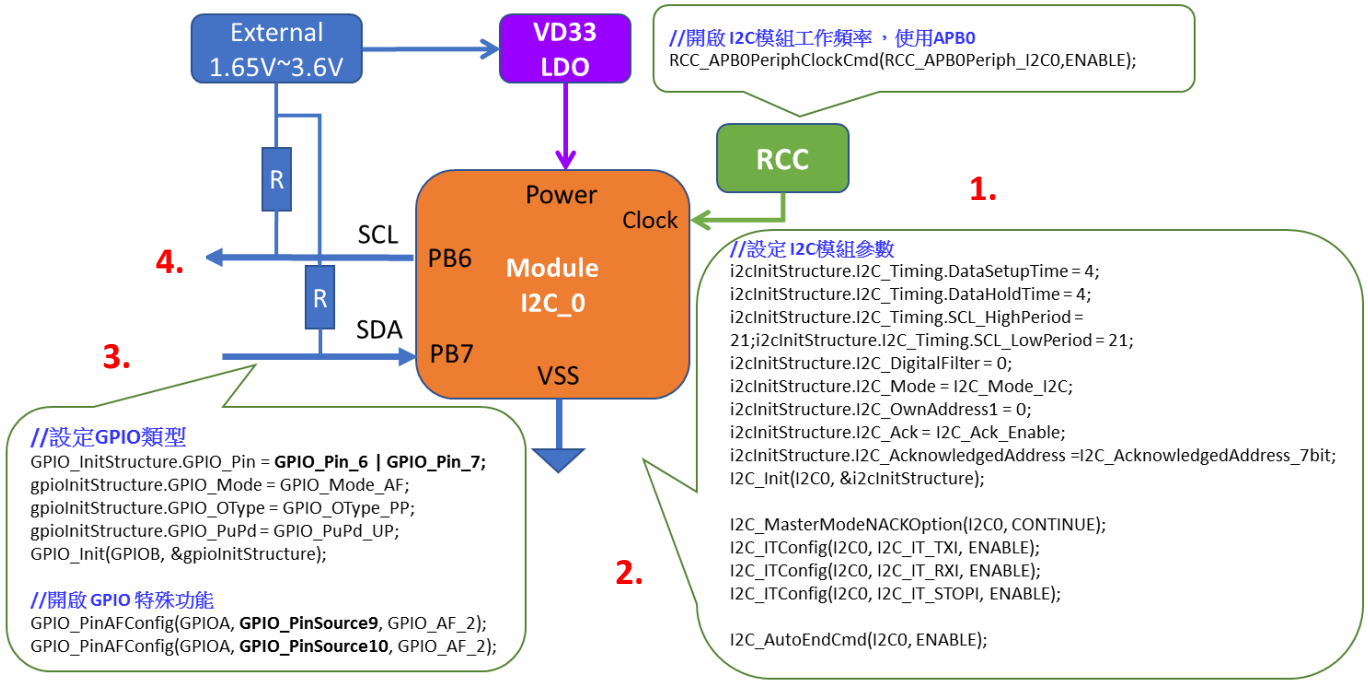
如下列 1~4 步驟，可參考周邊程式庫使用函式 InitialI2c0( )或 InitialI2c1( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 I2C 使用，如下圖步驟 1.

(Step 2) 設定 I2C 模組參數，如下圖步驟 2.

(Step 3) 設定 GPIO 類型 (IO 最後設定，避免信號灌入狀態未定的模組) ，如下圖步驟 3.

(Step 4) 發射 I2C 資料，如下圖步驟 4.



17.2 範例程式

```
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 進行 RX 接收資料 與 TX 發射資料

4.

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

  // --------------------------------
  // 主機發送資料給子機
  // --------------------------------
  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)));

  // --------------------------------
  // 主機接收資料於子機
  // --------------------------------
  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
}
```

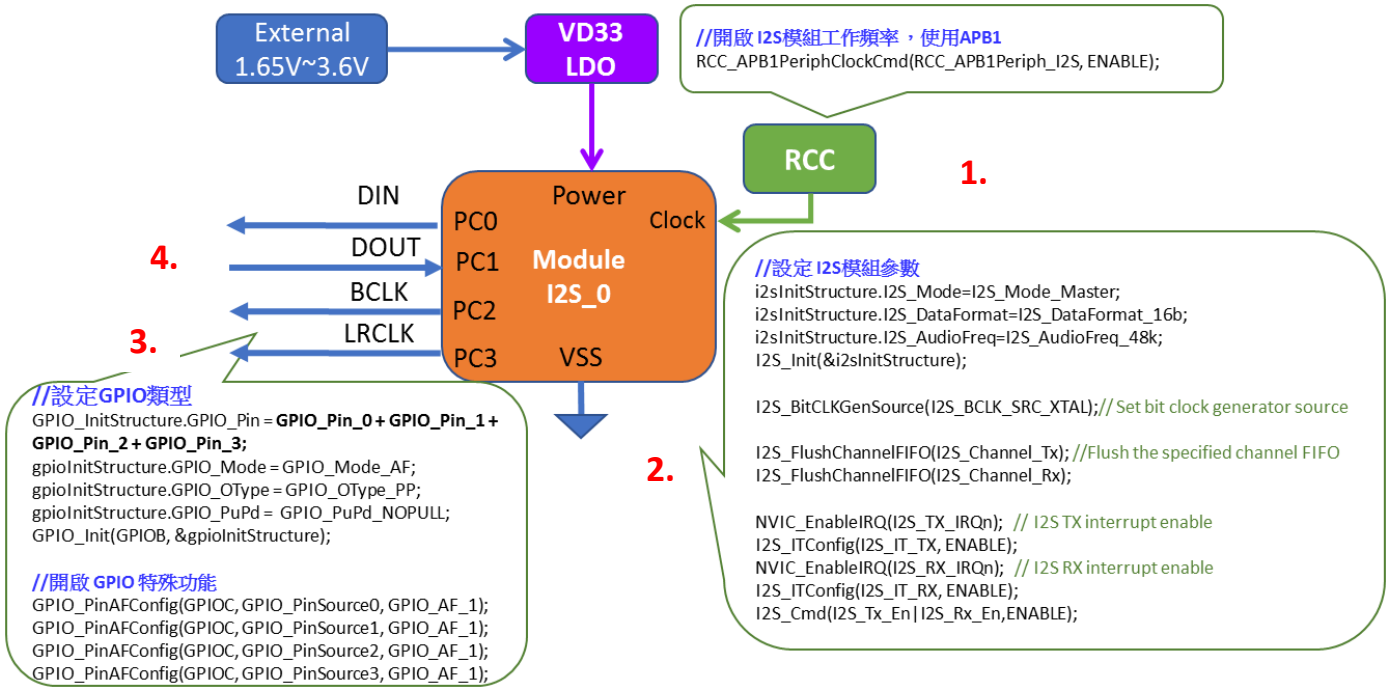
18. I2S 功能說明

使用下列圖示說明，使用 I2S0 或 I2S1 執行資料傳輸，動作流程如下:

18.1 MCU 上電後初始化 I2S

如下列 1~4 步驟，可參考周邊程式庫使用函式 InitialI2s0 ( )或 InitialI2s1 ( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 I2S 使用，如下圖步驟 1.
(Step 2) 設定 I2S 模組參數，如下圖步驟 2.
(Step 3) 設定 GPIO 類型 (IO 最後設定) ，如下圖步驟 3.
(Step 4) 發射 I2S 資料，如下圖步驟 4.



18.2 範例程式

```
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
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 功能說明

使用下列圖示說明，使用 PWM0A 或 PWM0B 執行寬度調變輸出，動作流程如下:

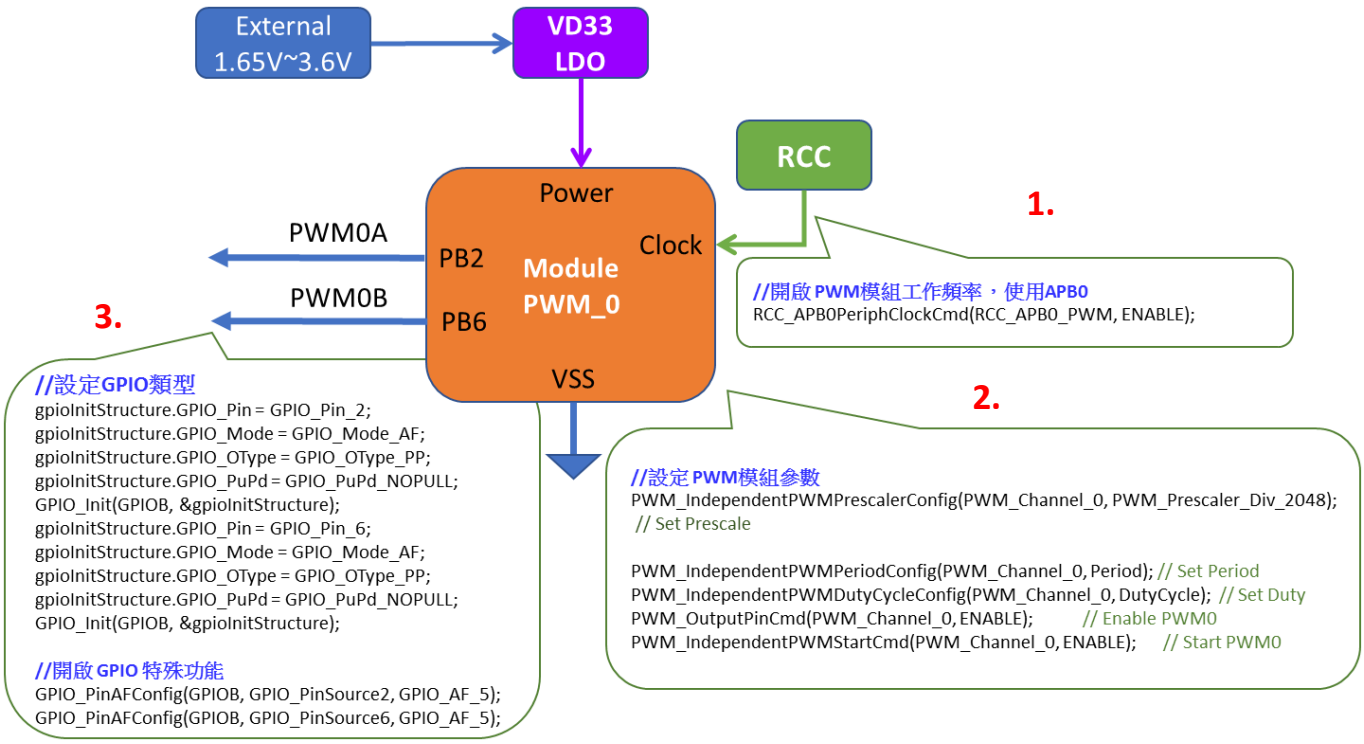
19.1 MCU 上電後初始化 PWM

如下列 1~4 步驟，可參考周邊程式庫使用函式 InitialPwm( )

(Step 1) 設定 RCC (時鐘控制模組) 開啟時脈提供給 PWM 使用，如下圖步驟 1.

(Step 2) 設定 PWM 模組參數，如下圖步驟 2.

(Step 3) 設定 GPIO 類型 (IO 最後設定，避免信號灌入狀態未定的模組) ，如下圖步驟 3.



19.2 範例程式

```
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

    //設定GPIO類型
3.  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);
    //開啟 GPIO 特殊功能
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource2, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_5);

}
```

20. DMA 功能說明

使用下列圖示說明，使用 DMA 0 與 DMA1 通道執行資料傳輸，範例為讀取 ADC 數值將資料搬運至 Timer 並輸出週期波形，動作流程如下:

20.1 MCU 上電後初始化 DMA

如下列 1~2 步驟，可參考周邊程式庫使用函式 InitiDma ( )

(Step 1) 設定 DMA0 通道，將 ADC 資料透過 DMA0 傳至 RAM 地址 **0x300000000** 如下步驟 1.

(Step 2) 設定 DMA1 通道，將 RAM 地址 **0x300000000** 資料透過 DMA1 傳至 Timer2 如下步驟 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 範例程式

```
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);
}
```

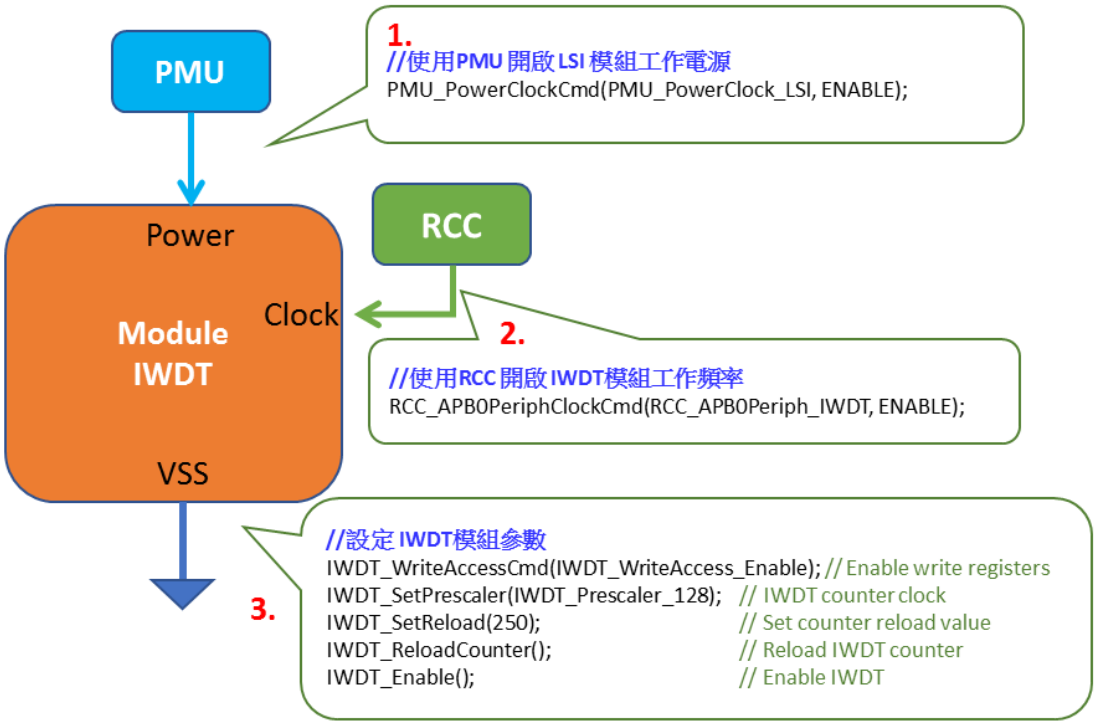
21. IWDT 功能說明

使用下列圖示說明，使用 IWDT 設定時間，動作流程如下:

21.1 MCU 上電後初始化 IWDT

如下列 1~4 步驟，可參考周邊程式庫使用函式 InitialIwdt( )

(Step 1) 設定 PMU(電源管理單元) 開啟類比電源提供給 IWDT 使用，如下圖步驟 1.
(Step 2) 設定 RCC (時鐘控制模組) 開啟時脈提供給 IWDT 使用，如下圖步驟 2.
(Step 3) 設定 IWDT 模組參數，如下圖步驟 3.



21.2 範例程式

```
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
}
```

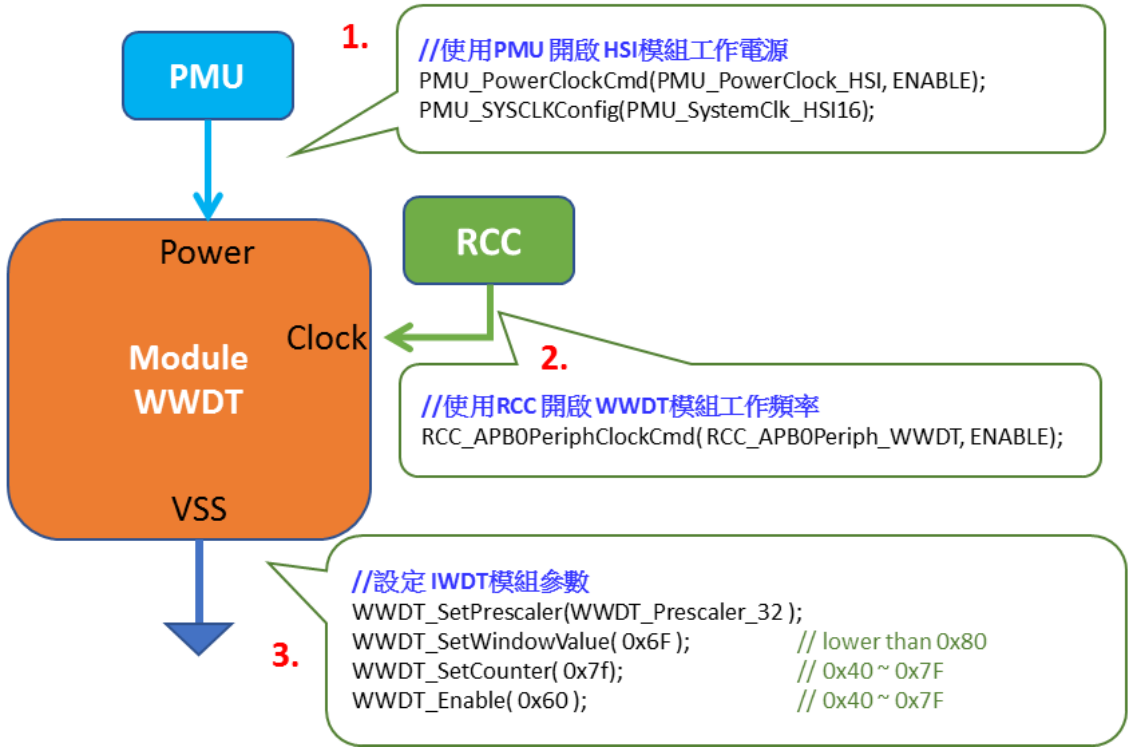
22. WWDT 功能說明

使用下列圖示說明，使用 WWDT 設定時間，動作流程如下:

22.1 MCU 上電後初始化 WWDT

如下列 1~4 步驟，可參考周邊程式庫使用函式 InitialWwdt( )

(Step 1) 設定 PMU(電源管理單元) 開啟類比電源提供給 WWDT 使用，如下圖步驟 1.
(Step 2) 設定 RCC (時鐘控制模組) 開啟時脈提供給 WWDT 使用，如下圖步驟 2.
(Step 3) 設定 WWDT 模組參數，如下圖步驟 3.

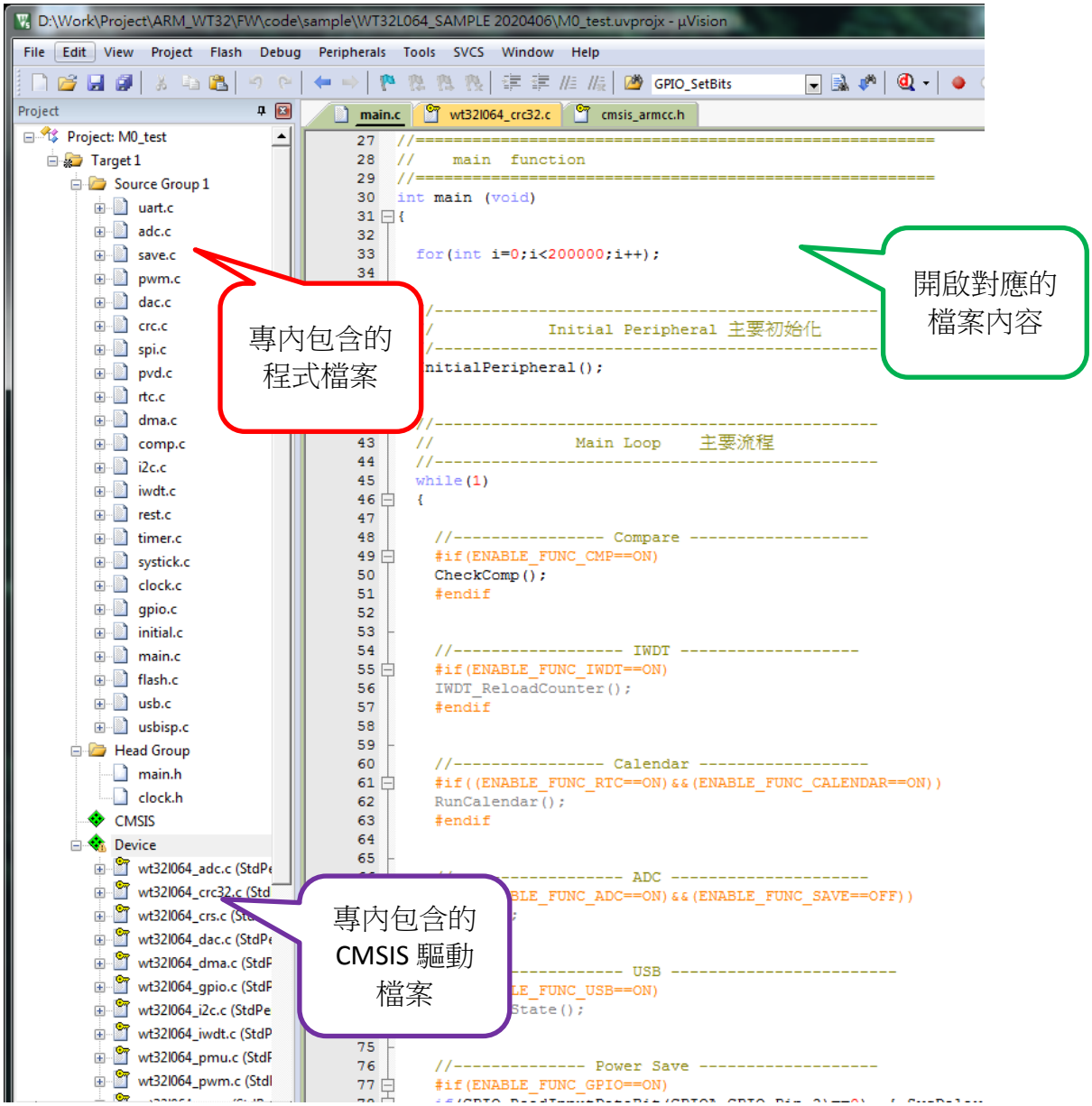


22.2 範例程式

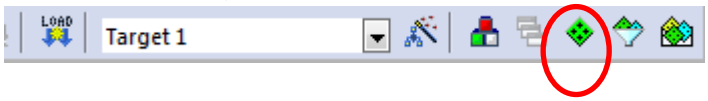
```
void InitialWwdt(void){
1.  PMU_PowerClockCmd(PMU_PowerClock_HSI, ENABLE);
    PMU_SYSCLKConfig(PMU_SystemClk_HSI16);
2.  RCC_APB0PeriphClockCmd(RCC_APB0Periph_WWDT, ENABLE);
    WWDT_DeInit();
3.  WWDT_SetPrescaler(WWDT_Prescaler_32);
    WWDT_SetWindowValue(0x6F);     // lower than 0x80
    WWDT_SetCounter(0x7f);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

23. 實例程式操作說明

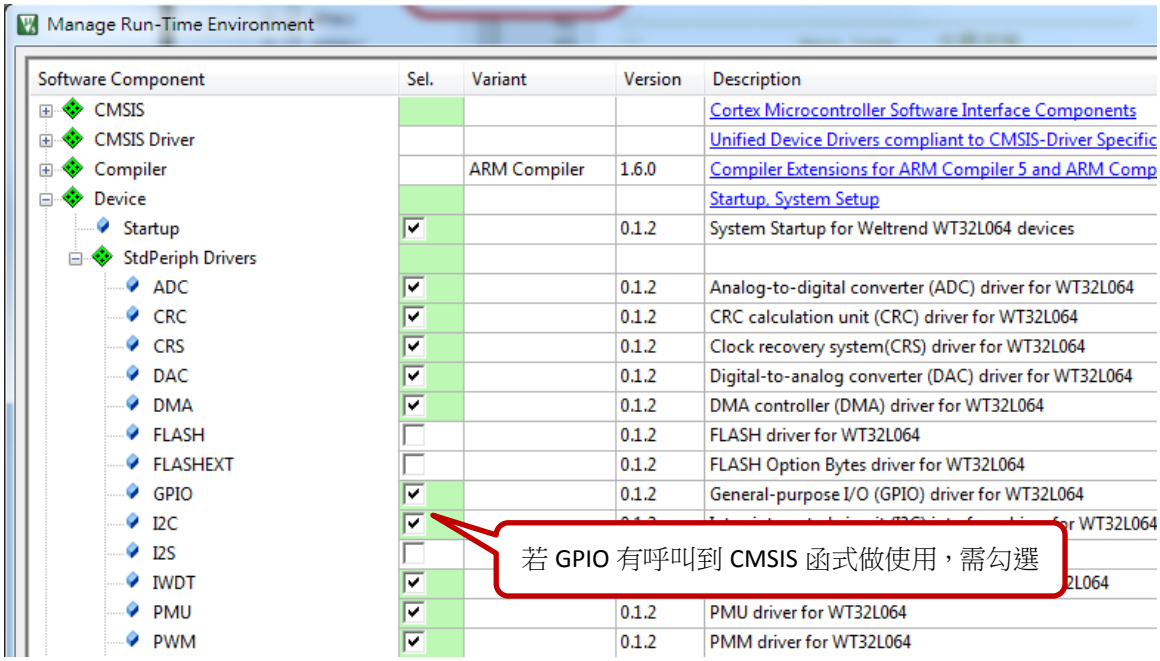
下列為如何使用參考範例的說明，專案名稱周邊程式庫參考前章節範例程式，依周邊功能放置個別檔案，開啟專案後的畫面如下，主要分三部分: 專案包含檔案、CMSIS 驅動、各源始檔內容



針對周邊功能新增 CMSIS 驅動層函式，於 ARM-MDK 上方選單上點選 Manage Run-Time Environment 如下圖示。

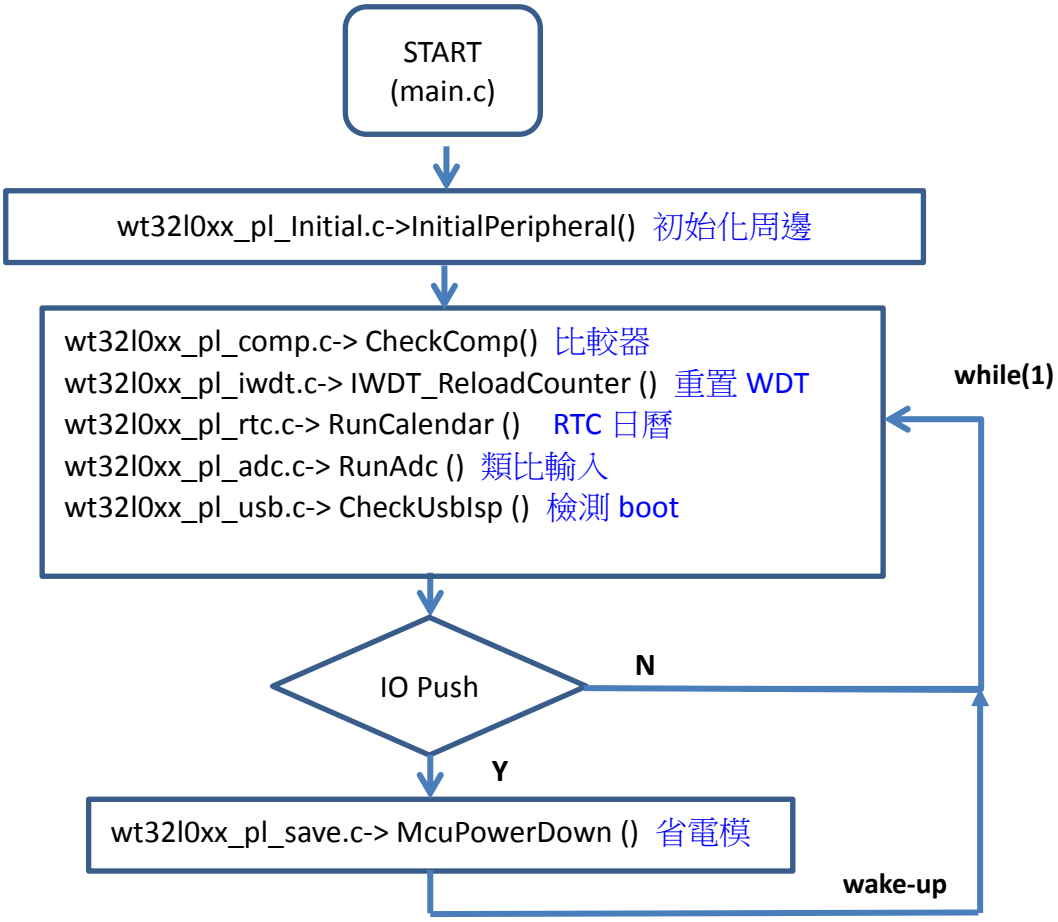


依序點選 Device->StdPeriph Drivers 如下圖所示，可依應用需求加入所需功能，EX: ADC、DAC、FLASH、GPIO、I2C…etc，一般範例程式都已加入參考到的 CMSIS，若有缺少部分或反黃項可再補加選



23.1 範例 WT32L064_SAMPLE_2020xx 流程圖

下列說明 WT32L064_SAMPLE_2020xx 流程圖、主要檔案內容與功能如下



依專案內檔案名稱與函式進行說明如下:

- **main.c 主程式流程，包括函式如下**

1.) InitialPeripheral() ----------參考到 initial.c，對周邊的初始化
2.) CheckComp () ----------參考到 comp.c，比較器輸出結果
3.) IWDT_ReloadCounter() ------參考到 iwdt.c，重置看門狗計數器
4.) RunCalendar() ----------參考到 rtc.c，檢測日曆數值
5.) RunAdc() ----------參考到 adc.c，執行 ADC 偵測
6.) CheckUsbState() ----------參考到 usb.c，檢測 USB 狀況
7.) McuPowerDown() ----------參考到 save.c，執行省電功能

程式主迴圈內容如下:

```
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();         //檢查COMP比較器狀態
       #endif

       //------------------ IWDT -------------------
       #if(ENABLE_FUNC_IWDT==ON)
       IWDT_ReloadCounter();//看門狗重載
       #endif

       //---------------- Calendar ------------------
       #if((ENABLE_FUNC_RTC==ON)&&(ENABLE_FUNC_CALENDAR==ON))
       RunCalendar();       //檢查RTC 日曆資料
       #endif

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

       //------------------ USB ---------------------
       #if(ENABLE_FUNC_USB==ON)
       CheckUsbIsp(); //檢查是否進入 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();      //進入省電模式
            #endif
            }
       }
     };//while(1);
  }
```

- **wt32l0xx_pl_library.h** 周邊功能的開關，請依需求依序開啟或關閉個別功能，程式內容如下。

```
//------------ Enable Function for Project -------------------------
// 請依序開啟下列功能， 使用ON為致能， OFF則為關閉

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

#define     ENABLE_FUNC_SOFT_RST            OFF         // 設定 Soft Reset 是否啟動

//------------ IO LED -------------------------
#define     ENABLE_FUNC_GPIO                ON          // 設定 GPIO 功能是否啟動
#if(ENABLE_FUNC_GPIO==ON)
#define ENABLE_GPIO_INT             OFF         // 設定 GPIO Interrupt 是否啟動
#define     ENABLE_LED_BLINK            ON          // 設定 GPIO Port-C LED 是否啟動
#define     ENABLE_LED_RESET            OFF         // 設定 GPIO 測試 Reset 是否啟動
#endif
#define     ENABLE_FUNC_SYSTICK         ON          // 設定 Systick 是否啟動

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

#define     ENABLE_FUNC_PWM             OFF         // 設定 PWM 是否啟動
#define     ENABLE_FUNC_IWDT            OFF         // 設定 IWDT 是否啟動
#define     ENABLE_FUNC_WWDT            OFF         // 設定 WWDT 是否啟動
#define     ENABLE_FUNC_FLASH           OFF         // 設定 仿真式 EEPROM 是否啟動
#define     ENABLE_FUNC_CRC             OFF         // 設定 CRC 是否啟動
#define     ENABLE_FUNC_SPI             OFF         // 設定 SPI 是否啟動
#define     ENABLE_FUNC_RESET           OFF         // 設定 Rest 是否啟動          (測試用)
#define     ENABLE_FUNC_PVD             OFF         // 設定 電壓檢測 是否啟動(測試用)
#define     ENABLE_FUNC_RESET           OFF         // 設定 Reset 是否啟動         (測試用)
#define     ENABLE_FUNC_I2C             OFF         // 設定 I2C 是否啟動
#define     ENABLE_FUNC_I2S             OFF         // 設定 I2S 是否啟動
#define     ENABLE_FUNC_TIMER           OFF         // 設定 Timer 是否啟動
#define     ENABLE_FUNC_DMA             OFF         // 設定 DMA 是否啟動, 使用 Timer+ADC 但須都啟
動
#define     ENABLE_FUNC_USB             OFF         // 設定 USB 是否啟動


//----------- Analog ----------------
#define     ENABLE_FUNC_CMP             ON          // 設定 COMPARE 是否啟動
#define     ENABLE_HW_CMP_SPEED_HI      OFF         //HI:4.5uALO:5.5uA
```

OFF: 關閉

ON: 開啟

數位功能類 開關

類比功能類 開關

```
#define     ENABLE_FUNC_ADC           OFF       // 設定 ADC 是否啟動
#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  // 設定 RTC 是否啟動
#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 功能開關

省電功能開關

喚醒功能開關

- **wt32l0xx_pl_initial.c** 周邊的初始化，包括函式如下

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

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

- **wt32l0xx_pl_clock.h** 工作頻率的選擇，可選擇 **HIS、MSI、HSE、PLL** 四種類型，程式如下

```
//--------- Use PLL for HSI 32MHz ----------------------------
#define CLOCK_PLL_HSI_X2_32MHZ  ON    //ON:開啟使用PLL倍頻HSI 16MHz至32Hz給系統使用

//--------- 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          //當選擇MSI時，系統選擇的工作頻率

//--------- 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         //系統選擇頻率的類型
```

選擇速度

選擇類型

- **wt32l0xx_pl_clock.c** 工作頻率設置函式，包括函式如下

1.) InitialSysClock () ----------執行系統頻率選擇，節錄內容如下

```
#if(SYS_CLOCK_SEL==CLK_HSI)         //使用 HSI 作系統頻率
      PMU_PowerClockCmd(PMU_PowerClock_HSI, ENABLE);
      PMU_SYSCLKConfig(PMU_SystemClk_HSI16);

#elif(SYS_CLOCK_SEL==CLK_MSI)       //使用 MSI 作系統頻率
      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 作系統頻率
      //…省略
#elif(SYS_CLOCK_SEL==CLK_HSE)       //使用 HSE 作系統頻率
```

2.) InitialUsbClock() ----------執行 USB 頻率選擇
3.) Delayms() ----------執行延遲功能
4.) DelayCount() ----------執行延遲功能

- **wt32l0xx_pl_gpio.c 外設 IO 類型設置，包括函式如下，可參考章節 4**

1.) GPIO_Handler ()----------中斷服務 GPIO 功能
2.) InitialGpio ()----------初始化 GPIO 功能

```
GPIO的4種類型: GPIO_Mode_IN => 基本輸入
               GPIO_Mode_OUT =>基本輸出
               GPIO_Mode_AF =>複合使用功能，EX:UART、SPI、I2C …
               GPIO_Mode_AN =>類比輸入功能，EX:ADC、USB、COMP …
```

- **wt32l0xx_pl_systick.c 內建 24bit 計時器設置，包括函式如下**

1.) SysTick_Handler ()----------中斷服務 systick 功能
2.) InitiSysTick ()----------初始化 systick 功能
3.) SysDelay ()----------使用 systick 延遲功能

- **wt32l0xx_pl_flash.c 模擬 EEPROM 燒錄設置，包括函式如下**

1.) RunFlash ()----------使用模擬 EEPROM 燒錄功能

- **wt32l0xx_pl_uart.c 異步收發器傳輸設置，包括函式如下，可參考章節 5**

1.) UART0_Handler ()----------中斷服務 UART0 功能
2.) UART1_Handler()----------中斷服務 UART1 功能
3.) InitialUart0 ()----------初始化 UART0 功能
4.) InitialUart1()----------初始化 UART1 功能
5.) fputc ()----------搭配 printf()使用發射串列資料功能
6.) fgetc()----------搭配 printf()使用接收串列資料功能
7.) DRV_IntToStr()----------數字轉字串
8.) Str2Num()----------字串轉數字
9.) uart_send_str()----------使用 UART0/1 發射串列資料
10.) uart_clear_str()----------清除串列內容

- **wt32l0xx_pl_adc.c 類比偵測設置，包括函式如下**

1.) ADC_Handler ()----------中斷服務 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()----------執行 ADC 目標通道轉換功能
5.) RunAllAdc ()----------執行 ADC 所有通道轉換功能
6.) RunAdcConvert()----------執行 ADC 通道單次轉換功能
7.) API_AverADCData ()----------執行 ADC 通道轉換功能，計算平均
8.) ADC_StartOfConversion_1() -啟動 ADC 模組轉換
9.) ADC_StopOfConversion_1() -停止 ADC 模組轉換
10.) HEX2BCD()----------16 進制轉 10 進制

- **wt32l0xx_pl_save.c 省電功能設置，包括函式如下**

1.) McuPowerDown ()----------執行省電功能前置作業並呼叫 Save()
2.) Save()----------依設定 SLEEP、STOP、STANDBY 執行省電功能

- **wt32l0xx_pl_pwm.c 脈衝週期調變功能設置，包括函式如下**

1.) InitialPwm() ----------執行 PWM 初始化並輸出功能

- **wt32l0xx_pl_dac.c 類比輸出設置，包括函式如下**

1.) DAC_Convert () ----------帶入數值至 DAC 並輸出功能
2.) DAC_Handler()----------執行 DAC 中斷功能
3.) InitialDac()----------執行 DAC 初始化

- **wt32l0xx_pl_crc.c 檢查碼 CRC 設置，包括函式如下**

1.) DAC_Convert () ----------帶入數值至 DAC 並輸出功能
2.) DAC_Handler()----------執行 DAC 中斷功能

- **wt32l0xx_pl_spi.c 串列周邊傳輸設置，包括函式如下**

1.) SPI_Config0 () ----------執行 SPI 0 初始化
2.) SPI_Config1()----------執行 SPI 1 初始化
3.) SPI1_Handler()----------執行 SPI 中斷功能

- **wt32l0xx_pl_pvd.c 電壓偵測設置，包括函式如下**

1.) InitPvd () ----------執行 PVD 初始化
2.) PVD_Handler ()----------執行 PVD 中斷功能

- **wt32l0xx_pl_rtc.c 實時計數器設置，包括函式如下**

1.) InitialRtc () ----------執行 RTC 初始化
2.) RTC_AlarmCmd ()----------執行 DAC 中斷功能
3.) RTC_Handler()----------執行 RTC 中斷功能
4.) RunCalendar()----------執行 RTC 日曆功能
5.) SetAlarm()----------設定 RTC 鬧鐘功能

- **wt32l0xx_pl_dma.c 直接記憶體存取設置，包括函式如下**

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

- **wt32l0xx_pl_comp.c 比較器設置，包括函式如下**

1.) CheckComp () ----------
2.) CMP0_VOUT_Handler ()----------執行 CPM0 中斷功能
3.) CMP1_VOUT_Handler()----------執行 CMP1 中斷功能
4.) InitialComp()----------初始化 COMP 比較器
5.) RumComp()----------執行 COMP 比較器

- **wt32l0xx_pl_i2c.c 標準 I²C 匯流排設置，包括函式如下**

1.) InitialI2c () ---------- -初始化 I2C 傳輸
2.) RunI2cTest ()----------執行 I2C 傳輸

- **wt32l0xx_pl_iwdt.c 看門狗設置，包括函式如下**

1.) InitialIwdt () ---------- 初始化看門狗

- **wt32l0xx_pl_reset.c 軟體復位設置，包括函式如下**

1.) InitLowVoltReset () ---------- 初始化低電壓復位
2.) RunReset ()----------測試低電壓復位

- **wt32l0xx_pl_timer.c 計數計時器設置，包括函式如下**

1.) ConfigTimerCapture () ----------配置 Timer 執行捕捉模式
2.) ConfigTimerClockGpio ()----------配置 Timer 執行輸出模式
3.) ConfigTimerInterrutp()----------配置 Timer 執行中斷模式
4.) ConfigTimerOutPWM()----------配置 Timer 執行 PWM 模式
5.) ConfigTimerTimeMode()----------配置 Timer 執行計時器模式
6.) TIMER0_Handler()----------執行 TIMER0 中斷功能
7.) TIMER1_Handler()----------執行 TIMER1 中斷功能
8.) TIMER2_Handler()----------執行 TIMER2 中斷功能

- **wt32l0xx_pl_usb.c 通用序列匯流排設置，包括函式如下**

1.) CLEAR_STALL () ---------- 清除 EP 端點 STALL 停滯狀態
2.) ENDPOINT_DISABLE ()---------- 關閉 EP 端點功能
3.) FUN_INIT()---------- 初始化 USB 端點 EP0 或其餘端點
4.) FUN_INT()---------- USB 端點 EP0~EPx 中斷服務函式
5.) FUN_INT2()---------- 處理終端 EP2 端點中斷

6.) FUN_REQUESTS()----------PC 端發送 USB 請求命令後，處理解析 USB 命令
7.) FUNTx0Send()----------裝置傳輸資料至 USB FIFO
8.) HID_EP1()----------USB 端點 EP1 傳送資料
9.) HID_EP2()----------USB 端點 EP2 傳送資料
10.) HID_EP3()----------USB 端點 EP3 傳送資料
11.) HID_GET_IDLE()----------USB-HID 取得 IDLE 時間設置
12.) HID_GET_PROTOCOL()----------USB-HID 取得 PROTOCOL 設置
13.) HID_GET_REPORT()----------USB-HID 取得 REPORT 設置的值
14.) HID_SET_IDLE()----------USB-HID 設定 IDLE 設置值
15.) HID_SET_PROTOCOL()----------USB-HID 設定 PROTOCOL 格式
16.) HID_SET_REPORT()----------USB-HID 設定 REPORT 格式
17.) IN_ENDPOINT_ENABLE()----------啟動 USB 端點 EP 的 IN 功能
18.) OUT_ENDPOINT_ENABLE()----------啟動 USB 端點 EP 的 OUT 功能
19.) ProcessUsbResetINT()----------重置並初始化 USB 端點 EP0~EPx
20.) ProcessUsbRx0INT()----------處理 EP0 接收中斷行程
21.) ProcessUsbTx0INT()----------處理 EP0 發射中斷行程
22.) ProcessUsbxx1INT()----------處理 EP1 收發中斷行程
23.) ProcessUsbxx2INT()----------處理 EP2 收發中斷行程
24.) ProcessUsbxx3INT()----------處理 EP3 收發中斷行程
25.) SendFirstBuffer()----------發送第一筆描述元至 PC 端
26.) SendFirstBufferWithSize()----------發射第一筆描述元至 PC 端，帶長度
27.) SendNextBuffer()----------發送第二筆描述元至 PC 端
28.) SET_STALL()----------設定端點 EP 停滯
29.) USB_CLEAR_FEATURE()----------清除 Feature 配置， 處理 USB 標準清除命令
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()----------讀取 STATUS 狀態
34.) USB_NOT_SUPPORT()----------不支持回應
35.) USB_RECEIVE_DATA()----------讀取 USB 接收資料
36.) USB_SET_ADDRESS()----------設定 USB 裝置地址
37.) USB_SET_CONFIG()----------設定 Config 配置

38.) USB_SET_FEATURE()----------設定 Feature 配置
39.) USB_SET_INTERFACE()----------設定 Interface 配置
40.) USB0_Handler()----------USB 信號中斷向量服務常式
41.) USB1_Handler()----------USB 端點 EP 中斷向量服務常式
42.) USBTxxINT()----------USB 傳輸中斷
43.) USBTxxSend()----------USB 傳輸預載入 Buffer

- **wt32l0xx_pl_usbisp.c** 通用序列匯流排進入 **Boot** 設置，包括函式如下

1.) CheckUsbIsp()----------檢查 USB 插頭是否接入 HOST
2.) enter_usbisp()----------執行 USB 燒錄 ISP 程式，設定後復位
3.) go_usb_suspend()----------進入 Suspend 待機省電模式
4.) InitialUSB()----------執行 USB 初始化

24. 版本更改紀錄:

版 本	紀 錄	日 期
V1.0	初稿	2020/9/18