501 的省電管理

V1.0

Publication Release Date: May. 2009

Table of Contents

1. 前言

NUC501 是以 ARM7TDMI 爲核心的 32 位單晶片微處理器，提供高達 108MHz 的運作頻率，滿足高速運算需求的工作。

然而在一般簡單控制或系統負載較低時，CPU 並不需要執行在最高的運行時脈下即可應付所需要的計算量，這時候，便可以利用系統時脈的控制來降低時脈，以達到省電的目的。

另一種情況是 CPU 因爲等待下一筆工作或只需要久久工作一次時，就可以先將 CPU 進入 Idle 或 Power Down 模式等到一定的時間之後或是有特定的事件發生時，才醒過來進行相關事件的處理，同時，如果系統內沒有用到的系統模組，也可以將其關閉以節省不必要的耗電。

接下來的章節將會介紹各種不同的省電方法的原理，以及提供相關的範例說明。

2. 開關系統模組

2.1. 爲什麼要關閉系統模組

在 NUC501 中，爲了省電的需求，當某個系統模組不使用時，可以將其輸入時脈關閉，使其進入關閉的模式，依此來達到省電的效果。

在 501 中，幾乎所有的系統模組都可以各別被關閉，這些模組包括了：

APU, SPIM, USB, ADC, SPIMS, I2C, PWM, UART, RTC, Watch Dog Timer and TIMER.

各模組的相關耗電如下表所示：

IP Enable/ Disable	Current Consumption(mA)
APU	1.24
SPIM	1.8
USB	1.17
ADC	0.48
SPIMS	0.40
I2C	0.24
PWM	0.47
UART	0.07
RTC	0.22
Watch Dog Timer	0.11
TIMER	0.11

Note: 以上數據由 3.3Volt 的情況下測得，由於環境及製程因素，可能會有些許變動。

2.2. 如何開關各模組

501 藉由關閉各模組時脈的方式來達到關閉各模組的目的，所以要關閉模組就必須設定相關的時脈控制暫存器，分別爲 AHBCLK Register 與 APBCLK Register，這兩個暫存器的功能說明如下表所示：

- **AHB 各模組時脈控制暫存器 (AHBCLK)**

在 AHBCLK 暫存器內的各位元，被用來作爲 AMBA clock, AHB Bus 上的模組以及 APB Bus 的時脈開關。

Register	Address	R/W	Description	Reset Value
AHBCLK	0xB1000200 + 04	R/W	AHB 模組時脈控制暫存器	0x0000_0083

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved							APU_CK_EN
7	6	5	4	3	2	1	0
SPIM_CK_EN	USBD_CK_EN	Reserved				APB_CK_EN	CPU_CK_EN

Bits	Descriptions	
[31:9]	Reserved	Reserved
[8]	APU_CK_EN	APU 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[7]	SPIM_CK_EN	SPIM 模組的時脈控制(SPIM0 & SPIM1) 0 = 關閉時脈 1 = 開啓時脈
[6]	USBD_CK_EN	USB 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[5:2]	Reserved	Reserved
[1]	APB_CK_EN	APB Bus 的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[0]	CPU_CK_EN	CPU 的時脈控制 0 = 關閉時脈 1 = 開啓時脈

- **APB 各模組時脈控制暫存器（APBCLK）**

APBCLK 暫存器內的各個位元被用來控制 APB Bus 上各模組的時脈。

Register	Address	R/W	Description	Reset Value
APBCLK	0xB1000200 + 08	R/W	APB Devices Clock Enable Control Register	0x0000_0007

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved						ADC_CK_EN	SPIMS_CK_EN
7	6	5	4	3	2	1	0
Reserved	I2C_CK_EN	PWM_CK_EN	UART1_CK_EN	UART0_CK_EN	RTC_CK_EN	WD_CK_EN	TIMER_CK_EN

Bits	Descriptions	
[31:10]	Reserved	Reserved
[9]	ADC_CK_EN	Analog-Digital-Converter (ADC)模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[8]	SPIMS_CK_EN	SPIMS 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[7]	Reserved	Reserved
[6]	I2C_CK_EN	I2C 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[5]	PWM_CK_EN	PWM 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[4]	UART1_CK_EN	UART1 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[3]	UART0_CK_EN	UART0 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[2]	RTC_CK_EN	Real-Time-Clock (RTC) 模組的 APB 介面時脈控制。這個位元只能用來控制 RTC 模組與 APB 連接介面的時脈，而不能控制由外部所提供的 32.768kHz crystal 時脈。 0 = 關閉時脈 1 = 開啓時脈
[1]	WD_CK_EN	Watch Dog Timer (WDT) 模組的時脈控制 0 = 關閉時脈 1 = 開啓時脈
[0]	TIMER_CK_EN	Timer 模組的時脈控制。 0 = 關閉時脈 1 = 開啓時脈

想要控制特定模組的開關，只需要找到相對應的控制位元，並將其設置爲 0 或 1 即可。一般可以透過 501BSP 內的 DrvSYS Driver 來完成。例如要關閉 UART1 模組時，可以使用下面的代碼來完成：

DrvSYS_SetIPClock(E_DRVSYS_UART1_CLK, 0);

如果想要直接通過暫存器的設定來開關各個模態的時脈，則可以使用底層的 I/O 存取，下面的代碼便演示了如何直接關閉 UART1 的時脈：

```
outp32(APBCLK, inp32(APBCLK) & (~(1 << 4)));
```

上述的代碼，直接使用了 outp32/inp32 直接對 APBCLK 暫存器作設置，因爲在這個例子中我們只想要將 UART1 關閉，所以需要先將 APBCLK 的設置讀回來，改動要設定的位元後，再寫回 APBCLK 暫存器中，以免誤關到別的模組。

3. 系統時脈設定

NUC501 本身的耗電量和它的運行時脈有很有的關係，如果運行的時脈高，則耗電高，反之則比較省電，但同時 CPU 就可能無法負荷大量的運算。因此如果要同時兼顧大量運算跟省電的話，就必須能夠根據實際上的需求來調整 CPU 的工作時脈，以求達到最佳的運作效率。

NUC501 內建了 PLL，能使用 2MHz ~ 20MHz 的 Crystal 產生系統所需的時脈，由 PLL 所產生的時脈再經過適當除頻，即可作爲 CPU 的工作頻率，因此 501 對於工作時脈的設定，有相當大的彈性，除了快到可以運行到 108MHz，必要時甚至可以直接使用外部的 Crystal 時脈作爲系統的時脈，例如使用 12MHz 的 Crystal 等等，使 CPU 運行在極低的時脈下。

想要設定 501 的工作頻率，可直接在程序裡調用 DrvSYS 庫裡函數來完成，主要的相關函數如下：

- *DrvSYS_Open(UINT32 u32ExtClockKHz, UINT32 u32PllClockKHz)*
- *DrvSYS_SetCPUClock(UINT32 u32CpuClockKHz)*

DrvSYS_Open 是用來設定 PLL 輸出頻率的函數，它的輸入參數有兩個，第一個是要輸入系統外部所使用的 Crystal 震盪頻率，另一個參數則指定了 PLL 的輸出頻率，另外還必須注意這兩個參數是以 1000Hz 爲單位。例如當系統所使用的 Crystal 震盪頻率是 12MHz，而需要 PLL 輸出 216MHz 的話，那可調用如下的程序來進行設定：

```
DrvSYS_Open(12000, 216000);
```

其中第一個參數 12000 表示現在系統外部所使用的 Crystal 震盪頻率是 12MHz，第二個參數表示要將 PLL 的輸出設定爲 216MHz。

當 PLL 的輸出頻率決定了之後，接下在就是要決定 CPU 的頻率，這個設定可通過呼叫 DrvSYS_SetCPUClock( ) 來完成，此函數的參數同樣是以 1000Hz 爲單位，而且必須注意由於硬體限制，CPU 的頻率至少要小於或等於 PLL 頻率的二分之一，也就是說如果 CPU 要使用 108Mhz 的頻率，那麼 PLL 最少就要有 216MHz 的輸出頻率才行。

```
DrvSYS_SetCPUClock(108000);
```

需要特別注意的是，調用這些頻率的設定函數並不一定真的能將相關時脈設定成想要的樣子，主要是因爲硬體有它一定的限制，因此如果有需要確認目前系統實際上的運作時脈的話，可通過 DrvSYS_GetPLLClock()及 DrvSYS_GetCPUClock()來取得。

501 允許使用者在程序中重複呼叫 DrvSYS_Open/DrvSYS_SetCPUClock 來重新設定運作時脈，不過由於重新設定 PLL 輸出時脈需要有一定的程序，將會造成一定程度的延遲，所以如果有動態改變系統運作時脈的需要時，建議利用改變 CPU 時脈的方式來完成即可，如此可將切換運作時脈的延遲減到最低。

4. IDLE 省電模式

501 除了可利用設定最適合的工作時脈來達到省電的目的外，如果在系統完全不需要工作時，還可以將 CPU 及大部分的硬體關閉，以達到最大的省電效果，這樣子搭配關閉 CPU 及大部分硬體的模式，我們稱之爲省電模式，其中又包括了 Idle 省電模式及 Power Down 省電模式兩種，下面我們將先就 Idle 省電模式進行說明。

4.1. 什麼是 IDLE 省電模式

藉由關閉大部分硬體時脈以達到最大省電效率的模式主要有兩種，一種是 Idle 省電模式，另一種是 Power Down 省電模式，這兩種模式最大的不同點是，當系統進入 Idle 省電模式下時，任何的中斷事件(interrupt)，都可以重新喚醒 CPU，好讓系統可以處理新進的事件，但是如果系統是處於 Power Down 省電模式的話，就只有少數特定的中斷事件能夠喚醒系統。

4.2. 如何進入 IDLE 省電模式

由於所謂的 Idle 模式，實際上是把 CPU 的時脈關閉，使得在沒有工作時，讓 CPU 能夠處在最省電的模式，而且因爲在這個模式下，只是將 CPU 的時脈關閉，一旦有任何中斷事件發生，馬上就可以打開 CPU 的時脈來處理相關的事件，因此有不會造成處理事件的延遲，又可以達到省電的效果，是兼顧速度與省電的一種模式。

要進入 Idle 模式，可以透過呼叫 DrvSYS 庫的 DrvSYS_EnableIdle()來達成，如下面的程序所示：

```
DrvSYS_EnableIdle();
```

一旦進入 Idle 模式，CPU 將立刻停止運作，因此在 DrvSYS_EnableIdle()函數之後得程序都必須等到 CPU 被喚醒後才會被執行。

一旦進入 Idle 省電模式，501 的耗電量將降到約 360uA。

4.3. 如何離開 IDLE 省電模式

進入 Idle 省電模式之後，如果要喚醒 CPU，回到一般的工作模式，就必須先產生中斷訊號，這中斷訊號可以是 501 中的任何模組所產生的中斷，也可以透有外部中斷的方式來喚醒 CPU，而由 Idle 省電模式喚醒 CPU 後，再進入中斷事件處理函數的延遲時間約 1.7us。要注意的是，當決定 CPU 要由某個中斷來喚醒時，就必須在進入 Idle 省電模式前，將該中斷設定完成，才能夠使其產生中斷事件來喚醒 CPU。至於中斷事件的設定，可根據不同的模組，利用 501BSP 中相對應的驅動庫來完成。

5. Power Down 省電模式

5.1. 什麼是 Power Down 省電模式

Power Down 是 501 最省電的一種模式，因爲一旦進入 Power Down 省電模式，Crystal 時脈會被關閉，整顆 501 呈現靜止的狀態，這時的耗電量將減到最小的程度，而在此模式下，也只有少數的特定中斷事件能夠喚醒 501，使其恢復工作狀態。

5.2. 如何進入 Power Down 省電模式

要進入 Power Down 省電模式，可以透過呼叫 DrvSYS 庫的 DrvSYS_EnablePowerDown()來達成，如下面的程序所示：

DrvSYS_EnablePowerDown();

DrvSYS_EanblePowerDown()將會設定 501 的 Power Down 控制暫存器，501 將在暫存器設定完成後再延遲 256 個外部 Crystal 時脈才真正將外部 Crystal 輸入關閉，此時才算是真正進入 Power Down 省電模式，因此在程序中要注意 DrvSYS_EnablePowerDown()之後的程序是有可能會在進入 Power Down 省電模式之前被執行到的，通常可以在呼叫 DrvSYS_EnablePowerDown()後，加入一段延遲的迴圈，來確保 CPU 在進入 Power Down 省電模式前，不會再進行額外的動作。

由於在 Power Down 省電模式下，整個時脈都停止了，所以數位電路的部分耗電量會最小，但因爲 PLL 是屬於類比模組，因此如果沒有特別將 PLL 關閉的話，即使關閉它的輸入時脈，它仍會運作並輸出無法預期的時脈訊號，因而消耗約 200uA 的電流，所以要真正將 501 Power Down，就必須先把 PLL 關閉，而關閉 PLL 前必須先將系統使用的時脈切換成使用外部 Crystal，才能讓 CPU 進行接下來真正進入 Power Down 省電模式的動作。

另一點要注意的是 501 在進入 Power Down 之後，由於是系統時脈關閉，所以幾乎所有的模組及 GPIO 都會保持 Power Down 以前的狀態，但是其中 GPIOA[11:8]將被強迫切換成輸入狀態且 GPIO[11:8]內部的 Pull Up 電阻此時也會無作用。

在進入 Power Down 之後，如果有 I/O 腳位是再輸入狀態，但卻沒有真正的信號輸入，可能會造成浮接，也就是輸入的信號無法確定是 0 或 1，這將會造成漏電的問題，因此在 Power Down 時，如果有任何腳位是處於輸入狀態的話，我們必須確保其不會有浮接的狀況，如果確認相關腳位會有浮接的狀況，可以通過加入 Pull Up 電阻來解決。如果是 GPIO 腳位，那我們可以使用內部的 Pull Up 電路即可，但是因爲 GPIOA[11:8]是由內部電路強迫其切換成輸入模式並使得內部 Pull Up 電阻失效，所以只能使用外部的 Pull Up 或 Pull Down 電路來避免浮接情況發生。

綜合以上各個要注意的事項，我們可以簡單的將整個進入 Power Down 省電模式的完整程序敘述如下：

1. 先確認進 Power Down 後，不會有浮接的輸入 I/O，包括 GPIOA[11:8]也要注意
2. 將系統時脈設定成外部 Crystal 時脈
3. 設定 MPLLCON 暫存器，將 PLL Power Down
4. 呼叫 DrvSYS_EnablePowerDown()，開始進入 Power Down

下面列出上述流程的代碼以供參考：

```
/* The GPIO status should be checked before power down to avoid I/O leakage caused by floating pin. */
    /* outp32(GPIOA_PUEN, 0xFFFF); */
    /* outp32(GPIOB_PUEN, 0xFFFF); */
    /* outp32(GPIOC_PUEN, 0xFFFF); */

    /* Use external clock */
    DrvSYS_SetSystemClockSource(E_DRVSYS_SYS_EXT);

    /* Power down the PLL */
    outp32(MPLLCON, inp32(MPLLCON) | (1 << 16));

    /* Disable crystal to enter power down */
    DrvSYS_EnablePowerDown();

    /* Wait for more than 256 external crystal cycles for entering power down */
    delay = 0x1000;
    while(delay--);
```

一旦進入 Power Down 省電模式，501 的耗電量將降到約 25uA，如果進入 Power Down 省電模式下，卻量到比 25uA 高很多的耗電量，就很有可能是 PLL 沒有關閉或是有 I/O 漏電的問題產生。

5.3. 如何脫離 Power Down 省電模式

一旦 501 進入 Power Down 省電模式，由於大部分的邏輯電路都進入停止狀態，要想喚醒 CPU，和 Idle 省電模式下，只要有任何中斷即可喚醒 CPU 不同，不過相同的是，兩者都必須要在進入省電模式前，先設定好將來要用來喚醒 CPU 的模組，這包括其中斷或喚醒功能，這些爲了喚醒 CPU 的準備都完成之後，才能進入省電模式，否則就沒有任何方式能夠再喚醒 CPU 了。

由於所謂的 Power Down 省電模式，實際上是把 Crystal 的時脈關閉，讓整個 501 呈現靜止的狀態，而且因爲在這個模式下，Crystal 也被關閉了，一旦有 Wakeup 中斷事件發生，需要等到 Crystal 穩定下來後，CPU 才能繼續運作，所以由 Power Down 進入一般的工作模式，會需要延遲一點時間，這一段延遲時間的長短是可以設定的，需根據 Crystal 的穩定性來調整，由實際測量的結果顯示，當 Wakeup 事件發生，到執行中斷處理函數，約需要 360us，如果爲了 Crystal 的穩定，另外加上了延遲設定，則必須根據延遲設定值再加上 360us。

這裡所謂的延遲時間設定值，是由 PWRCON[8:23]來決定，最小值爲 0，相當於關閉延遲設定(PWRCON[1])時的延遲時間，延遲設定值是以外部 Crystal 時脈乘上 256 爲單位，如果外部以 12MHz

Crystal 爲時脈輸入源的話，而且 PWRCON[8:23]設置爲 1，則其延遲設定約爲 1/12MHz * 256 約等於 21.33us，再加上系統原本就有延遲 360us，系統由 Wakeup 事件發生到進入中斷處理函數的延遲將變成約 381.33us。

下表是 PWRCON 暫存器的詳細說明：

Register	Address	R/W	Description	Reset Value
PWRCON	0xB1000200 + 00	R/W	Power Down 控制暫存器	0x00FF_FF03

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Pre-Scale[15:8]							
15	14	13	12	11	10	9	8
Pre-Scale[7:0]							
7	6	5	4	3	2	1	0
Reserved				INT_EN	INTSTS	XIN_CTL	XTAL_EN

Bits	Descriptions	
[31:24]	Reserved	Reserved
[23:8]	Pre-Scale	Pre-Scale counter 當系統由 Power Down(Crystal 關閉)狀態下，重新進入正常工作模式時，用來設定等待外部 Crystal 穩定的延遲值，等時脈穩定後才讓其進入系統，此設定值的單位爲外部 Crystal 時脈乘上 256。
[7:4]	Reserved	Reserved
[3]	INTSTS	Power Down 中斷狀態 0 = 一般狀態 1 = 系統由 Power Down 恢復正常運作，這可能是由 GPIO、UART 或 RTC 來喚醒系統的 要清除本狀態，要對 INTSTS 寫 1
[2]	INT_EN	Power Down 中斷控制開關 0 = 關閉 Power Down 中斷 1 = 開啓 Power Down 中斷 當打開 Power Down 中斷開關之後，每當 XTAL_EN 由 0 變 1 時，就會產生中斷。
[1]	XIN_CTL	此位元是用來控制 Pre-Scale 的設定值是否生效 1 = 啓用 Pre-Scale 設定值 0 = 關閉 Pre-Scale 設定值
[0]	XTAL_EN	Crystal Oscillator (Power Down) 控制開關 1: 啓用 Crystal oscillation(一般工作模式) 0: 關閉 Crystal oscillation(Power Down 省電模式)

要脫離 Power Down 省電模式，並不是單純的打開模組的中斷功能即可，而是要特別去打開該模組的喚醒功能，目前有提供由 Power Down 省電模式喚醒功能的模組有：

- 外部訊號 (GPIO interrupt)
- UART
- RTC

，除此之外，如果在 Power Down 前，我們有先將 PLL 關閉，那麼喚醒系統之後的第一件事，通常就是先將 PLL 打開，以確保接下來的程序運作正常。下面我們將一一說明各個喚醒方式如何實現。

5.3.1. 由外部訊號喚醒

要由外部訊號喚醒 CPU，可以使用外部中斷的方式，設定方式跟一般外部中斷相同，但有兩點要特別注意，一個是需要另外打開該中斷的喚醒功能，也就是要呼叫 DrvGPIO_EnableWakeupInt()來打開該中斷的喚醒功能，另外就是中斷的觸發方式要選擇上下緣都產生中斷，也就是呼叫 DrvGPIO_EnableInt()時要設定成 IO_BOTH_EDGE。

下面是以使用 GPIOA[3]來由 Power Down 省電模式喚醒 CPU 的範例：

```
/* Enable Key A interrupt to wakeup the CPU */
DrvGPIO_Open(GPIOA, 3, IO_INPUT, IO_NO_PULL_UP, IO_LOWDRV);
DrvGPIO_EnableInt(GPIOA, 3, IO_INT0, IO_BOTH_EDGE, (DRVGPIO_CALLBACK)IoHandler, 0);
DrvGPIO_EnableWakeupInt(IO_INT0);

/* Set I bit of ARM */
DrvAIC_SetCPSR(AIC_ENABLE_IRQ);
```

其中 IoHandler()是外部中斷時的處理函數，其至少要包含將中斷旗標清除的動作，否則會造成一直發出中斷訊號，且一直喚醒 CPU 導致無法再進入 Power Down 省電模式，簡單的 IoHandler()範例如下：

```
VOID IoHandler(UINT32 u32UserData)
{
    UINT32 *pu32Reg;

    pu32Reg = DrvGPIO_GetIntStatus();
    pu32Reg[0] = (1 << 3); /* Clear the interrupt flag */

}
```

請注意因爲我們在設定外部中斷時，使用了上下緣觸發，因此這個 IoHandler()將在外部有任何變化時都會被呼叫到。

5.3.2. 由 UART 訊號喚醒

UART 模組是利用 Modem 的中斷事件訊號，來對 Power Down 下的系統做喚醒的動作，在 501 中，這個訊號就是 CTS。

因此如過要利用 UART 來喚醒 CPU，首先必須要讓 UART 的 CTS/RTC 腳位有連接到外部，這可透過呼叫 DrvGPIO_SetPadReg1()來完成，例如我們要使用 UART0 來喚醒 CPU，就要使用以下的程序先將相關的 GPIO 切換成 UART 加上 CTR/RTS 的功能：

```
DrvGPIO_SetPadReg1(FUNC_UART0, 3);
```

I/O 腳位設定好了之後，接下來就是要打開 UART 的 Modem 中斷事件，然後再啓動由 Modem 中斷事件來喚醒 CPU 的功能，其相關程序範例下：

```
/* Enable the modem interrupt and install its callback */
DrvUART_EnableInt(UART_PORT0, DRVUART_MOSINT, (PFN_DRVUART_CALLBACK *)UartHandler);

/* Enable UART0 CTS of modem interrupt to wakeup the CPU */
outp32(UA_IER, inp32(UA_IER) | (1 << 7));
```

除了設置好 Modem 的中斷外，爲了要打開 UART 的 Wakeup 功能，我們必須將 UA_IER 暫存器的 bit 7 設置爲 1，如此當 Modem 中斷發生時，才能喚醒系統。

請注意上述的範例省略了 UART 的初始化程序，而相關的代碼可以在 UART 的 Driver Sample 中找到。

範例中所使用的 UartHandler 則是用來處理 UART 中斷的函數，使用者可以利用它來清除 UART 中斷旗標或做其他功能的處理。

依上述的說明，設定好了 UART 之後，便可以利用 CTS 訊號的變化來由 Power Down 省電模式來喚醒 CPU 了。

5.3.3. 由 RTC 訊號喚醒

進入 Power Down 省電模式之後， 可以利用 RTC 鬧鐘(Alarm)功能來喚醒系統，由於 RTC 的運作時脈爲 32.768kHz，跟系統的 Crystal 時脈是不一樣的，所以即使進入 Power Down，RTC 仍可以正常工作，此時我們就可以利用 Alarm 功能，在特定的時間喚醒系統，當然也可以透過設定好下一次 Alarm 的時間，達到固定週期性的喚醒系統，以進行必要的工作。

下面的程序是一個週期性 Alarm 的範例，搭配之前提到進入 Power Down 省電模式的方法，即可達到固定時間喚醒，然後再進入 Power Down 的效果，在下面程序中，我們先初始化 RTC 之後，便進入無限迴圈中，等待 Alarm 事件的發生，在這無限迴圈內，可以加入 Alarm 時所要處理的工作，並在工作完成之後，再進入 Power Down 省電模式，並等待下一次的 Alarm 來喚醒系統。

```
S_DRVRTC_TIME_DATA_T g_sCurTime;

/* Time Setting */
g_sCurTime.u32Year = 2009;
g_sCurTime.u32cMonth = 1;
g_sCurTime.u32cDay = 19;
g_sCurTime.u32cHour = 13;
g_sCurTime.u32cMinute = 20;
g_sCurTime.u32cSecond = 0;
g_sCurTime.u32cDayOfWeek = DRVRTC_MONDAY;
g_sCurTime.u8cClockDisplay = DRVRTC_CLOCK_24;

DrvRTC_Init();
```

```
    /* Initialization the RTC timer */
    if(DrvRTC_Open(&g_sCurTime) !=E_SUCCESS)
            DrvSIO_printf("Open Fail!!\n");

    /* Trigger the Alarm start */
    RtcHandler();

    DrvAIC_SetCPSR(AIC_ENABLE_IRQ);

    while(1)
    {
            /* The code to handle the task after Alarm and enter power down mode again. */
    }
```

上述範例中的 RtcHandler 內容可參考下面的程序：

```
  VOID RtcHandler(VOID)
  {


    /* PLL should work before we can print message */

    /* Set PLL to normal mode */
    outp32(MPLLCON, inp32(MPLLCON) & (~(1 << 16)));

    /* Use PLL clock */
    DrvSYS_SetSystemClockSource(E_DRVSYS_SYS_PLL);


    /* Get the currnet time */
    DrvRTC_Read(DRVRTC_CURRENT_TIME, &g_sCurTime);

    /* Set Alarm call back function */
    g_sCurTime.pfnAlarmCallBack = RtcHandler;

    DrvSIO_printf("Wakeup by RTC Alarm. Current Time:%d/%02d/%02d %02d:%02d:%02d\n",
    g_sCurTime.u32Year,g_sCurTime.u32cMonth,g_sCurTime.u32cDay,g_sCurTime.u32cHour,g_sCurTime.u32cMi
  nute,g_sCurTime.u32cSecond);

    /* The alarm time setting: every 3 seconds */
    g_sCurTime.u32cSecond = (g_sCurTime.u32cSecond + 3);
    if( g_sCurTime.u32cSecond >= 60)
    {
          g_sCurTime.u32cSecond -= 60;
          g_sCurTime.u32cMinute ++;
    }

    /* Set the alarm time (Install the call back function and enable the alarm interrupt)*/
    DrvRTC_Write(DRVRTC_ALARM_TIME,&g_sCurTime);


  }
```

RtcHandler()主要的功能是讀出目前的 RTC 時間，並在該時間加上 3 秒鐘，來表示三秒後要再發一次 Alarm 事件，用以喚醒系統。請注意因爲 RtcHandler()有使用 DrvSIO_printf()來透過 UART 輸出一些提示訊息，由於 UART 需要 PLL 時脈來產生 baud rate， 爲了確保訊息列印沒問題，程序中會在一進入 RtcHandler 便將 PLL 打開，使訊息的列印能夠正常工作。

6. 總結

在本文件中，我們介紹了 501 所提供的各種省電方式，包括了關閉沒用到的模組、調整 CPU 運作的時脈、Idle 省電模式與 Power Down 省電模式，使用者可以根據自身的應用，選擇適合的方式，甚至組合不同的省電方法以達到最佳的省電效率。在 501 BSP 也提供了相關的範例程序，以供使用者參考，此範例可在 501BSP\NuvotonPlatform_ADS\Sample\Diagnostic\Smpl_PowerManagement 目錄下找到。

7. Revision History

Version	Date	Description
V1.0	July. 17, 2009	● Created

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.

Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.