NUC980 Family Programming Guide

Document Information

Abstract	This document introduces the control sequence of NUC980 family peripherals.
Apply to	NUC980 family.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 概述

NUC980 系列32-bit 微處理器基於Arm926EJ-S™ 處理器核心，帶有 16 KB I-cache, 16 KB D-cache 及 MMU 可執行到 300 MHz。自帶的 SDRAM 介面支援 SDR/DDR/DDR2/LPDDR 種類的 SDRAM 時脈最高至 150 MHz。NUC980 系列帶有16 KB SRAM 及 16.5 KB IBR (Internal Boot ROM) 用於從 USB，NAND，SD/eMMC 及 SPI 閃存開機，支持供規操作溫度從 -40℃ 到 85℃。 NUC980 提供內建 DDR 的 LQFP 包裝，減少 PCB 設計難度及降低 BOM 成本。

NUC980 系列提供了多個高速數字週邊，例如兩組 10/100 Mbps 乙太網的 RMII介面，一個 USB 2.0 HS host/device 及一個 USB 2.0 HS host 控制器，最多六個 USB 2.0 FS host lite 介面，兩個 CMOS 介面可支持 CCIR601 及 CCIR656 的 sensor，兩個 SD 介面支持 SD/SDHC/SDIO 卡，一個 NAND 閃存介面可支持 SLC 及 MLC 的 NAND 閃存，一個 I2S 介面支持 I2S 及 PCM 協議。另外NUC980 系列提供了硬件加解密加速器支援 RSA，ECC，AES，SHA，HMAC 以及亂數產生器(RNG) 。

NUC980 系列提供了最多十組 UART 介面, 兩組 ISO-7816-3 介面, 一組 Quad-SPI 介面, 兩組 SPI 介面, 最多四組 I2C 介面，四組CAN 2.0B 介面，八個 PWM 輸出通道，八組 12-bit SAR ADC，六個 32-bit 時鐘, WDT (看門狗) ，WWDT (窗口看門狗) ，32.768 kHz 晶振接口及 RTC (實時時鐘) 。同時NUC980 系列也支持了兩個十通道的 PDMA 供週邊使用於對記憶體讀寫資料。

本文件由軟體工程師的角度描述了系統設定，時鐘控制，中斷處理，及各週邊的控制方式。

2 系統管理器 (System Manager)

2.1 概述

系統管理介紹了以下信息和功能。

- 系統復位
- 系統內存映射
- 系統管理寄存器用於產品標識（PDID），上電設置，系統喚醒，復位控制芯片控制器/外部設備，以及多功能引腳控制。
- 系統控制寄存器

2.2 功能描述

2.2.1 暫存器解鎖/鎖定

有些系統控制暫存器需要被保護起來，以防止誤操作而影響芯片運行，而這些被保護的寄存器在上電復位到使用者解鎖之前是鎖定的狀態。使用者必須連續依次寫入"59h"，"16h"，"88h"到暫存器SYS_REGWPCTL即可將這些暫存器解鎖。任何不同的值、不同順序或者在寫入這三個數據期間有任何其他的操作都會破壞整個的解鎖過程。然而用戶不需考慮寫入這三筆資料之間的時間，僅需依序寫入即可。

解鎖後，使用者可以檢查REGPRTDIS(SYS_REGWPCTL[0])狀態，1表示已經解鎖，0表示鎖定。使用者可以在更新目標暫存器後，再向暫存器SYS_REGWPCTL寫入任何值，就可以重新鎖定保護暫存器。

2.2.2 端口多功能控制 (Multiple Function Control)

使用各個模組之前要先切換對應的控制功能，以SPI0為例，就是將GPD2～5設定成SPI0所使用的 pin 腳 。 GPD3 是 SPI0_SS0 ， GPD2 是 SPI0_CLK ， GPD4 是 SPI0_DATAO ， GPD5 是 SPI0_DATAI，因此，SYS_GPD_MFPL要填入0x00111100。。(每根 pin 的設定，請參閱 NUC980 Technical Reference Manual) 。

2.2.3 系統復位

系統重置可以由以下列其中一個的事件發出，其中系統重置之前要先設置CHIPRST(SYS_AHBIPRST[0])、 CPU復位設置CPURST(SYS_AHBIPRST[2])才會打開功能。讀取SYS_RSTSTS暫存器可以得知系統重置事件，並向該位元寫1即可清除事件。

- 上電復位，PORRSTS (SYS_RSTSTS[0])
- 低電位 RESET 引腳，PINRSTS (SYS_RSTSTS[1])
- 低電壓復位，LVRRSTS (SYS_RSTSTS[2])，請參考 1.3.4
- 系統重置，CHIPRSTS (SYS_RSTSTS[3])
- CPU 復位，CPURSTS (SYS_RSTSTS[4])

- 看門狗超時復位，WDTRSTS (SYS_RSTSTS[5])

2.2.4 低電壓偵測/復位

當電壓低於2.6伏特或是2.8伏特，SYS_MISCISR寄存器的LVD_IS位會設置，如果中斷有始能，就會產生中斷。當電壓從2.3伏特上升超過2.4伏特，系統就會復位。

低電壓復位或檢測的順序如下：

1. 設定SYS_LVRDCR寄存器LVD_SEL位，選擇檢測電壓為2.6伏特還是2.8伏特。
2. 檢測，始能SYS_LVRDCR寄存器LVD_EN位。
3. 檢測和復位，始能SYS_LVRDCR寄存器LVD_EN位和LVR_EN位。
4. 啟動中斷，始能SYS_MISCIER寄存器LVD_EN位。
5. 確認中斷狀態，檢查SYS_MISCISR寄存器LVD_IS位。
6. 清除SYS_MISCISR寄存器LVD_IS位。
7. 重複步驟2～6。

2.2.5 周邊設備復位

在SYS_AHBIPRST、SYS_APBIPRST0或SYS_APBIPRST1這三個暫存器中依照每個周邊設備設定對應的位元寫1再寫0就可以讓周邊設備復位。以TIMER0為例，使用者在TIMER0RST(SYS_APBIPRST0[8])先寫1再寫0即可讓TIMER0重置。

2.2.6 電源模式與喚醒來源

- 正常模式：CPU 正常工作且所有時鐘 ON。
- 掉電模式：除了 LXT 之外，所有時鐘停止。

芯片進入掉電模式後，系統等待喚醒來源(wake-up source)發生後回到正常模式繼續執行程式，使用者可以透過讀取暫存器SYS_WKUPSSR0或SYS_WKUPSSR1來得知喚醒來源的狀態。掉電模式的喚醒來源包括：WDT、GPIO、EINT、Timer、UART、I^2C、RTC、CAN、LVD、EMAC、USBH、USBD、SDH和ADC，使用這些喚醒來源之前必須始能暫存器SYS_WKUPSER0或SYS_WKUPSER1，並且需要在喚醒來源對應周邊設備的暫存器裡再去各自設定。

在掉電模式下利用TIMER0來喚醒系統，步驟如下：

1. 打開TIMER0時鐘並選擇時鐘來源為LXT
2. 打開TIMER0 time-out中斷，設定time-out時間為100個LXT clocks。
3. TMR0WKEN(SYS_WKUPSER0[8])設定為1
4. 設定WKEN(TIMER0_CTL[2])為1，打開TIMER0的喚醒功能
5. TIMER0開始計數
6. 執行__wfi後進入掉電模式，程式碼如下

```
__asm void __wfi(void)
{
    MCR p15, 0, r1, c7, c0, 4
    BX  lr
}
```

7. 等待TIMER0 time-out中斷就會把系統喚醒。
8. 讀取TMR0WK(SYS_WKUPSSR0[8])來確定喚醒來源為TIMER0。

2.2.7 USB ID 偵測

USB Host和USB Device有一個共用埠，當接上A類型（Host）接頭，軟體可以檢測出來並執行Host程序；反之，接上B類型（Device）接頭，軟體就可以執行Device程序。

使用方式如下：

9. 始能SYS_MISCIER寄存器USBIDC_IEN位。
10. 插上接頭，產生中斷，確認SYS_MISCISR寄存器USBIDC_IS位是否設置。
11. 判斷SYS_MISCISR寄存器USB0_IDS位，1為接上USB Host；0為接上USB Device。
12. 清除SYS_MISCISR寄存器USBIDC_IS位。
13. 重複步驟2～4。

2.3 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
SYS Base Address: **SYS_BA = 0xB000_0000**				
SYS_PDID	SYS_BA+0x000	R	Product Identifier Register	0x1030_D016[1]
SYS_PWRON	SYS_BA+0x004	R/W	Power-On Setting Register	0xXXXX_XXXX[2]
SYS_LVRDCR	SYS_BA+0x020	R/W	Low Voltage Reset & Detect Control Register	0x0000_0001
SYS_MISCFCR	SYS_BA+0x030	R/W	Miscellaneous Function Control Register	0x0000_0200
SYS_MISCIER	SYS_BA+0x040	R/W	Miscellaneous Interrupt Enable Register	0x0000_0000
SYS_MISCISR	SYS_BA+0x044	R/W	Miscellaneous Interrupt Status Register	0x0001_0000
SYS_WKUPSER0	SYS_BA+0x050	R/W	System Wakeup Source Enable Register0	0x0000_0000
SYS_WKUPSER1	SYS_BA+0x054	R/W	System Wakeup Source Enable Register1	0x0000_0000
SYS_WKUPSSR0	SYS_BA+0x058	R/W	System Wakeup Source Status Register 0	0x0000_0000

SYS_WKUPSSR1	SYS_BA+0x05C	R/W	System Wakeup Source Status Register 1	0x0000_0000
SYS_AHBIPRST	SYS_BA+0x060	R/W	AHB IP Reset Control Register	0x0000_0000
SYS_APBIPRST0	SYS_BA+0x064	R/W	APB IP Reset Control Register 0	0x0000_0000
SYS_APBIPRST1	SYS_BA+0x068	R/W	APB IP Reset Control Register 1	0x0000_0000
SYS_RSTSTS	SYS_BA+0x06C	R/W	Reset Source Active Status Register	0x0000_00XX
SYS_GPA_MFPL	SYS_BA+0x070	R/W	GPIOA Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPA_MFPH	SYS_BA+0x074	R/W	GPIOA High Byte Multiple Function Control Register	0x0000_0000
SYS_GPB_MFPL	SYS_BA+0x078	R/W	GPIOB Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPB_MFPH	SYS_BA+0x07C	R/W	GPIOB High Byte Multiple Function Control Register	0x0000_0000
SYS_GPC_MFPL	SYS_BA+0x080	R/W	GPIOC Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPC_MFPH	SYS_BA+0x084	R/W	GPIOC High Byte Multiple Function Control Register	0x0000_0000
SYS_GPD_MFPL	SYS_BA+0x088	R/W	GPIOD Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPD_MFPH	SYS_BA+0x08C	R/W	GPIOD High Byte Multiple Function Control Register	0x0000_0000
SYS_GPE_MFPL	SYS_BA+0x090	R/W	GPIOE Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPE_MFPH	SYS_BA+0x094	R/W	GPIOE High Byte Multiple Function Control Register	0x0000_0000
SYS_GPF_MFPL	SYS_BA+0x098	R/W	GPIOF Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPF_MFPH	SYS_BA+0x09C	R/W	GPIOF High Byte Multiple Function Control Register	0x0000_0000
SYS_GPG_MFPL	SYS_BA+0x0A0	R/W	GPIOG Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPG_MFPH	SYS_BA+0x0A4	R/W	GPIOG High Byte Multiple Function Control Register	0x7777_7000
SYS_DDR_DSCTL	SYS_BA+0x0F0	R/W	DDR I/O Driving Strength Control Register	0x0000_0000
SYS_PORDISCR	SYS_BA+0x100	R/W	Power-On-Reset Disable Control Register	0x0000_00XX
SYS_RSTDEBCTL	SYS_BA+0x10C	R/W	Reset Pin De-bounce Control Register	0x0000_04B0
SYS_REGWPCTL	SYS_BA+0x1FC	R/W	Register Write-Protection Control Register	0x0000_0000

Note: [1] Dependents on part number.

Note: [2] Dependents on power-on setting.

3 時鐘控制器 (Clock Controller)

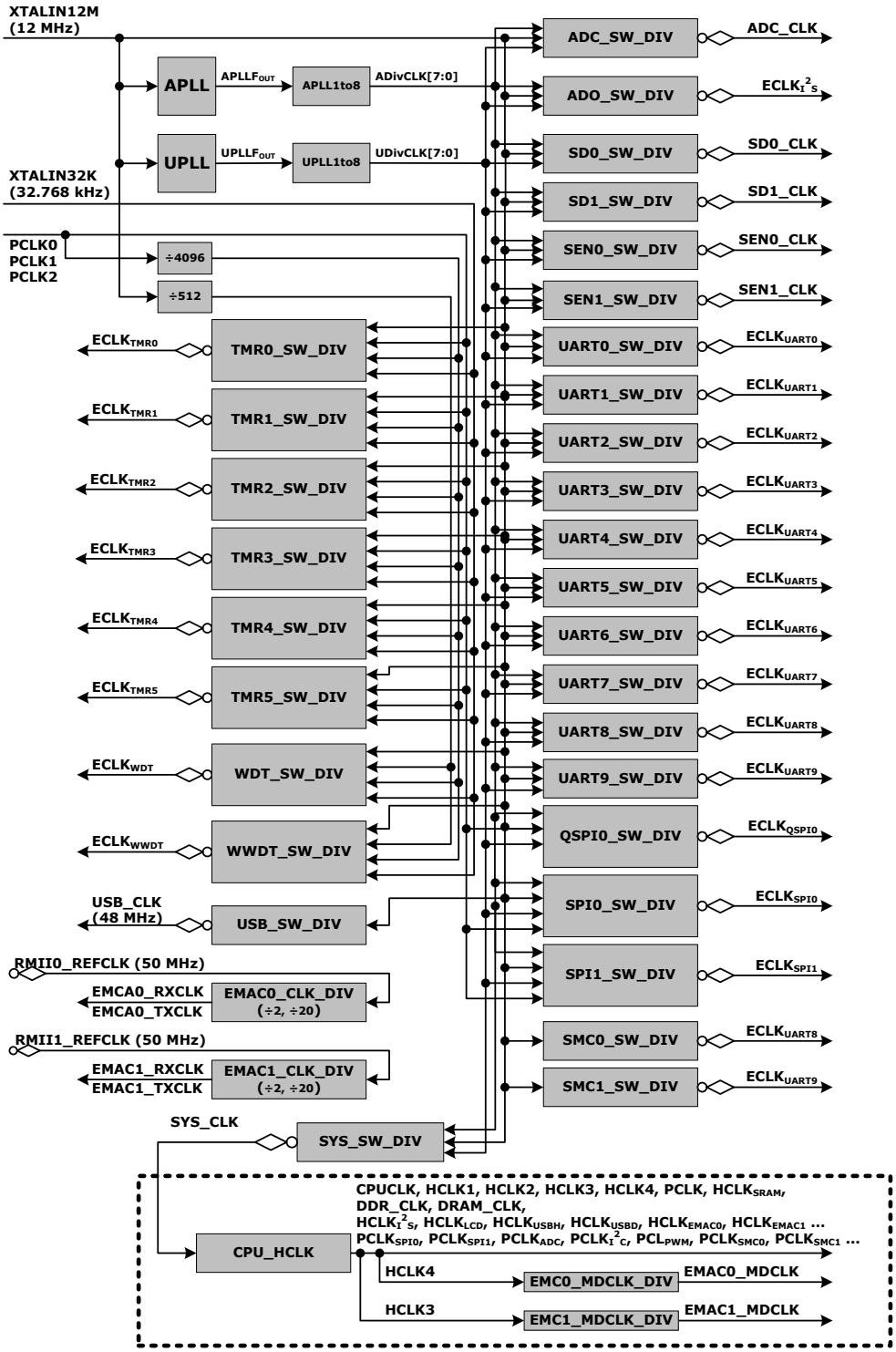
3.1 概述

時鐘控制器產生所有的時鐘提供給視頻，音頻，CPU，AMBA和所有模塊。該芯片包括兩個PLL模塊。每個模塊的時鐘源來自PLL，或直接從外部晶振輸入。對於每一個時鐘在CLKEN寄存器裡可以單獨控制時鐘的開啟或關閉，並且在CLK_DIVCTL寄存器裡有設置除法器。

3.2 特性

- 支持兩個PLL，高達500 MHz，用於高性能的系統運行
- 外部12MHz的高速晶振輸入用於精確定時操作
- 外部32.768 kHz低速晶振輸入用於RTC功能和低速時鐘源

3.3 方塊圖



3.4 功能描述

3.4.1 Pre-Scalar 計數器

為了避免系統使用不穩定的時鐘，時鐘控制器實現 Pre-Scalar 計數器 PRESCALE(CLK_PMCON[23:8])，等待計數器計數至PRESCALE×256晶振週期後，晶振時鐘穩

定，此時時鐘控制器才會有時鐘的輸出。

3.4.2 模組時鐘開關

NUC980各個模組都有獨立的時鐘開關，讀寫寄存器之前要先始能，模組才能啟動。時鐘開關存在於三個寄存器，CLK_HCLKEN，CLK_PCLKEN0，CLK_PCLKEN1。AHB總線上的模組要設定CLK_HCLKEN，而APB總線上的模組，則設定CLK_PCLKEN0或CLK_PCLKEN1。

以NAND為例，NAND是由AHB總線上的閃存接口（FMI）模組控制，因此要啟動NAND必需始能FMI和NAND，也就是要設置CLK_HCLKEN寄存器裡的FMI和NAND位。以UART0為例，要利用UART0來打印信息，必須先始能UART0的時鐘，也就是要設置CLK_PCLKEN0寄存器裡的UART0位。

3.4.3 時鐘除頻器

可外接裝置的模組都有各自的時鐘除頻器，用以提供正確的時鐘輸出。每個除頻器除了設定除數之外，還可以選擇時鐘來源。以外接SD卡為例：時鐘來源選擇UPLL，初始化時要輸出300KHz，傳輸時要輸出50MHz，假設UPLL的頻率是300MHz，除頻器設定如下：

1. SD1 時鐘除頻寄存器為 CLKDIV9，需要控制 SD1_N，SD1_S。
2. 設定 SD1 時鐘來源是 UPLL，SD1_S 位要填入 11b。
3. 初始化頻率為 300KHz，在 SD1_N 填入 999，即可得到 300KHz 輸出。
4. 數據傳輸頻率為 50MHz，設定 SDH_N 為 5，代表 300MHz 除 6，就能輸出 50MHz。

3.4.4 PLL 設置

NUC980 PLL默認值是設定300MHz，調整PLL頻率需要符合下列的算式所計算出來的值才是合法的。

$$F_{pllout} = 12\ \text{MHz} \times \frac{N}{M \times P}$$

$$F_{vco} = 12\ \text{MHz} \times \frac{N}{M}$$

$$200\ \text{MHz} < F_{vco} < 500\ MHz$$

$$F_{pfd} = \frac{12\ MHz}{M} = \frac{F_{vco}}{N}$$

N	F_{pfd} Range
1	$11.0 \le F_{pfd} \le 80$
2	$7.0 \le F_{pfd} \le 80$
3	$5.0 \le F_{pfd} \le 80$
4	$4.0 \le F_{pfd} \le 80$

5	$3.5 \leq F_{pfd} \leq 80$
6	$3.0 \leq F_{pfd} \leq 80$
7 ~ 8	$2.5 \leq F_{pfd} \leq 80$
9 ~ 10	$3.5 \leq F_{pfd} \leq 80$
11 ~ 40	$3.0 \leq F_{pfd} \leq 80$
41 ~ 128	$2.5 \leq F_{pfd} \leq 80$

3.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
CLK Base Address: **CLK_BA = 0xB000_0200**				
CLK_PMCON	CLK_BA+0x000	R/W	Power Management Control Register	0xFFFF_FF03
CLK_HCLKEN	CLK_BA+0x010	R/W	AHB Devices Clock Enable Control Register	0x0000_0527
CLK_PCLKEN0	CLK_BA+0x018	R/W	APB Devices Clock Enable Control Register 0	0x0000_0000
CLK_PCLKEN1	CLK_BA+0x01C	R/W	APB Devices Clock Enable Control Register 1	0x0000_0000
CLK_DIVCTL0	CLK_BA+0x020	R/W	Clock Divider Control Register 0	0x0000_00XX
CLK_DIVCTL1	CLK_BA+0x024	R/W	Clock Divider Control Register 1	0x0000_0000
CLK_DIVCTL2	CLK_BA+0x028	R/W	Clock Divider Control Register 2	0x0000_1500
CLK_DIVCTL3	CLK_BA+0x02C	R/W	Clock Divider Control Register 3	0x0000_0000
CLK_DIVCTL4	CLK_BA+0x030	R/W	Clock Divider Control Register 4	0x0000_0000
CLK_DIVCTL5	CLK_BA+0x034	R/W	Clock Divider Control Register 5	0x0000_0000
CLK_DIVCTL6	CLK_BA+0x038	R/W	Clock Divider Control Register 6	0x0000_0000
CLK_DIVCTL7	CLK_BA+0x03C	R/W	Clock Divider Control Register 7	0x0000_0000
CLK_DIVCTL8	CLK_BA+0x040	R/W	Clock Divider Control Register 8	0x0000_0500
CLK_DIVCTL9	CLK_BA+0x044	R/W	Clock Divider Control Register 9	0x0000_0000
CLK_APLLCON	CLK_BA+0x060	R/W	APLL Control Register	0x1000_0015
CLK_UPLLCON	CLK_BA+0x064	R/W	UPLL Control Register	0xX000_0015
CLK_PLLSTBCNTR	CLK_BA+0x080	R/W	PLL Stable Counter and Test Clock Control Register	0x0000_1800

4 中斷控制器 (AIC)

4.1 概述

中斷是臨時改變程式執行的順序時所進行的反應，以一個特定的事件，例如電源故障、看門狗定時器超時、MAC控制器的發送/接收請求等的特定事件作出反應。CPU處理器提供兩種中斷模式：快速中斷(FIQ)模式和中斷(IRQ)模式。當nIRQ輸入發生IRQ請求，類似地，當nFIQ輸入發生FIQ請求。而FIQ有特權可以搶占正在進行的IRQ。當前程式狀態寄存器(CPSR)的F和I bit時，它可以忽略FIQ和IRQ通過設置。

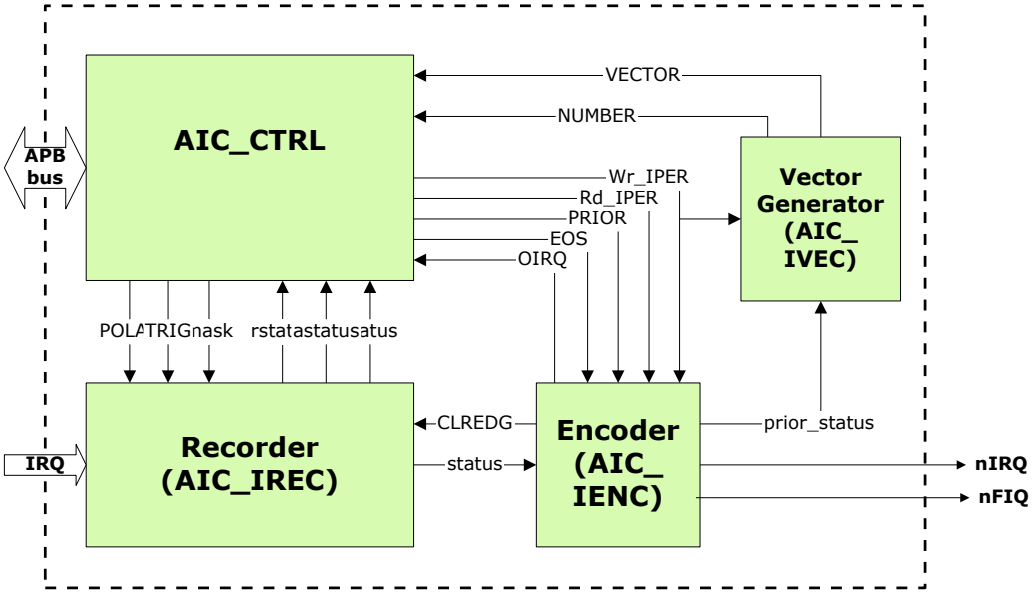
高級中斷控制器(AIC)能夠處理多達64個不同來源的中斷請求。目前，63個中斷源被定義。每個中斷來源被分配給一個中斷通道。例如，看門狗定時器中斷被分配到通道1。AIC實現了63個中斷來源和8個優先級。優先級0中的中斷來源為最高和優先級7的中斷來源是最低的。為了使優先級正常工作，優先級必須指定給每個中斷來源在上電時初始化；否則，系統會產生意外行為。每個優先級在一個較低的中斷通道具有更高的優先級。中斷來源為最高優先級0會被晉升為FIQ。中斷來源除0之外的優先級都會是IRQ。而IRQ可由FIQ的發生時被搶占。

雖然中斷來源來自晶片本身本質上是高電平敏感，對AIC可以配置每個中斷來源為低電平敏感、高電平敏感、負邊沿觸發，或者正邊沿觸發。

4.2 特性

- AMBA APB 總線介面
- 外部中斷可以被編程為邊沿觸發或電平檢測
- 外部中斷可以被編程為無論是低電位動作和高電位動作
- 利用標誌反映每個中斷來源的狀態
- 每個中斷來源都有獨立的遮罩
- 支持專有的 8 級優先級中斷。
- 優先級的方法是採用允許中斷菊花鏈(daisy-chaining)
- 自動遮罩了在中斷嵌套低優先級的中斷
- 自動清除中斷標誌當外部中斷來源被編程為邊沿觸發

4.3 方塊圖



4.4 功能描述

4.4.1 中斷通道配置

每個中斷通道都有獨立的來源代碼控制寄存器來設定其類型和優先級。所有NUC980系列內部中斷類型都是正電平觸發，這些不應該在正常操作期間被改變。設備驅動必須根據外部設備所設定的中斷類型來決定中斷來源的觸發，每個中斷裝置完全決定優先級。上電或復位後，所有的通道由AIC分配為優先級0～7。下圖所示來源控制寄存器(AIC_SRCCTLx)的內容。

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

Type [7:6]		Interrupt Type
0	0	Low Level Sensitive
0	1	High-Level Sensitive
1	0	Negitive-Edge Triggered
1	1	Positive-Edge Triggered

4.4.2 中斷遮罩

NUC980系列 AIC提供一組寄存器來遮蔽每個中斷通道。遮罩啟用命令寄存器(AIC_INTEN0和AIC_INTEN1)。寫1到AIC_INTEN0或AIC_INTEN1將使相對應的中斷通道啟用中斷。相反，遮罩禁用命令寄存器(AIC_INTDIS0和AIC_INTDIS1)。寫1到AIC_INTDIS0或AIC_INTDIS1會禁止相對應的中斷通道禁用中斷。寫0到AIC_INTENx或AIC_INTDISx則沒有效果，因此設備驅動器可以任意改變這兩個寄存器不保持其原始值。如果有必要，設備驅動器可以讀取中斷遮罩寄存器(AIC_INTMSK0或AIC_INTMSK1)知道中斷通道是否被啟用或禁用。如果中斷通道被啟用，則對應的位元能讀取到1，否則為0。

4.4.3 中斷處理操作流程

AIC實現了一個專有的8個優先級方案。使用這種機制，中斷通道發生之前需要正確設定AIC_SRCCTLx。AIC個別提供AIC_FIQNUM與AIC_IRQNUM中斷來源編號暫存器，用來識別是FIQ還是IRQ的中斷來源。FIQ或IRQ的中斷處理程式可以分別通過讀取AIC_FIQNUM寄存器及AIC_IRQNUM寄存器得到各自的中斷源。一般情況下有一個函數表，以保持內部設備和外部設備的中斷服務程式。當中斷是由CPU內核的認可，FIQ或IRQ異常處理程式首先執行，然後它會根據AIC_FIQNUM 或AIC_IRQNUM內容調用適當的中斷服務程式。異常處理程式和中斷服務程式應遵循一定的規則，讓這個機制工作正常。控制流程如下：

1. 配置中斷來源(AIC_SRCCTLx)
2. 啟用中斷AIC_INTENx
3. 等待中斷發生
4. 進入對應的中斷服務程式(IRQ service routine)。
5. 讀取AIC_FIQNUM/AIC_IRQNUM來獲得向量，是當前編碼中斷通道號碼，這可以防止AIC一個較低的優先級中斷請求
6. 寫任何值到AIC_EOIS寄存器，來完成中斷。

```
__irq void sysIrqHandler()
{
    UINT32 volatile num;

    num = inpw(REG_AIC_IRQNUM);
    if (num != 0)
        (*sysIrqHandlerTable[num])();
    outpw(REG_AIC_EOIS, 1);
}
```

4.4.4 EINT 使用方式

NUC980系列提供4個外部中斷(EINT0~EINT4)，下表列出可配置成EINT功能的GPIO，使用外部中斷需要配置GPIO寄存器。

EINT	Pin Name	MFP
EINT0	PA.0	5
	PA.13	8
EINT1	PA.1	5
	PA.14	8
EINT2	PD.0	4
	PB.3	3
EINT3	PD.1	4
	PG.15	4

EINT0使用步驟如下：

1. 設定GPIO(CLK_HCLKEN[11])為1，打開GPIO的時鐘。
2. PA.0多功能引腳設定成EINT0功能。
3. 設定PA0為Input Mode。
4. 設定PA->INTTYPE為 1(Level trigger interrupt)。
5. 設定PA->INTEN為1(Low level)。
6. 註冊EINT0的中斷服務程式(IRQ service routine)
7. 設定中斷類型為IRQ
8. 啟用中斷AIC_INTENx

除了上述步驟，在中斷服務程式中還需要清除相對應的中斷源寄存器PA->INTSRC。

```
void UserEINT0Init(void)
{
    /* Enable GPIO engine clock */
    outpw(REG_CLK_HCLKEN,inpw(REG_CLK_HCLKEN) | (1<<11));
    /* Configure PA.0 as EINT0 pin and enable interrupt by low level trigger */
    outpw(REG_SYS_GPA_MFPL, 0x5);
```

```
    PA->MODE = 0x0; //input mode
    PA->INTTYPE = (1 << 0); //LEVEL
    PA->INTEN = (1 << 0); // LOW LEVEL


    /* EINT0 AIC setting */
    sysInstallISR(HIGH_LEVEL_SENSITIVE | IRQ_LEVEL_1, EINT0_IRQn,
                  (PVOID)EINT0_IRQHandler);
    sysSetLocalInterrupt(ENABLE_IRQ);
    sysEnableInterrupt(EINT0_IRQn);
}


INT32 EINT0_IRQHandler(void)
{
    PA->INTSRC = PA->INTSRC;
    sysprintf("IRQ Num=%d\n", inpw(AIC_IRQNUM));
}
```

4.4.5 中斷來源

以下為 NUC980 所有的中斷源列表.

Priority	Name	Mode	Source
1 (Highest)	WDT_INT	Positive Level	Watch Dog Timer Interrupt
2	WWDT_INT	Positive Level	Windowed-WDT Interrupt
3	LVD_INT	Positive Level	Low Voltage Detect Interrupt
4	EXT_INT0	Positive Level	External Interrupt 0
5	EXT_INT1	Positive Level	External Interrupt 1
6	EXT_INT2	Positive Level	External Interrupt 2
7	EXT_INT3	Positive Level	External Interrupt 3
8	GPA_INT	Positive Level	GPIO A Interrupt
9	GPB_INT	Positive Level	GPIO B Interrupt
10	GPC_INT	Positive Level	GPIO C Interrupt
11	GPD_INT	Positive Level	GPIO D Interrupt
12	I2S_INT	Positive Level	I2S Interrupt
13	(Reserved)	Positive Level	(Reserved)
14	VCAP0_INT	Positive Level	VCAP 0 Interrupt
15	RTC_INT	Positive Level	RTC Interrupt

16	TIMER0_INT	Positive Level	Timer 0 Interrupt
17	TIMER1_INT	Positive Level	Timer 1 Interrupt
18	ADC_INT	Positive Level	ADC Interrupt
19	EMC0_RX_INT	Positive Level	EMC 0 RX Interrupt
20	EMC1_RX_INT	Positive Level	EMC 1 RX Interrupt
21	EMC0_TX_INT	Positive Level	EMC 0 TX Interrupt
22	EMC1_TX_INT	Positive Level	EMC 1 TX Interrupt
23	EHCI_INT	Positive Level	USB 2.0 Host Controller Interrupt
24	OHCI_INT	Positive Level	USB 1.1 Host Controller Interrupt
25	PDMA0_INT	Positive Level	PDMA Channel 0 Interrupt
26	PDMA1_INT	Positive Level	PDMA Channel 1 Interrupt
27	SDH_INT	Positive Level	SD/SDIO Host Interrupt
28	FMI_INT	Positive Level	FMI Interrupt
29	UDC_INT	Positive Level	USB Device Controller Interrupt
30	TIMER2_INT	Positive Level	Timer 2 Interrupt
31	TIMER3_INT	Positive Level	Timer 3 Interrupt
32	TIMER4_INT	Positive Level	Timer 4 Interrupt
33	VCAP1_INT	Positive Level	VCAP 1 Interrupt
34	TIMER5_INT	Positive Level	Timer 5 Interrupt
35	CRYPTO_INT	Positive Level	CRYPTO Engine Interrupt
36	UART0_INT	Positive Level	UART 0 Interrupt
37	UART1_INT	Positive Level	UART 1 Interrupt
38	UART2_INT	Positive Level	UART 2 Interrupt
39	UART4_INT	Positive Level	UART 4 Interrupt
40	UART6_INT	Positive Level	UART 6 Interrupt
41	UART8_INT	Positive Level	UART 8 Interrupt
42	CAN3_INT	Positive Level	CAN 3 Interrupt
43	UART3_INT	Positive Level	UART 3 Interrupt
44	UART5_INT	Positive Level	UART 5 Interrupt
45	UART7_INT	Positive Level	UART 7 Interrupt
46	UART9_INT	Positive Level	UART 9 Interrupt
47	I2C2_INT	Positive Level	I2C 2 Interrupt
48	I2C3_INT	Positive Level	I2C 3 Interrupt
49	GPE_INT	Positive Level	GPIO E Interrupt
50	SPI1_INT	Positive Level	SPI 1 Interrupt

51	QSPI0_INT	Positive Level	QSPI 0 Interrupt
52	SPI0_INT	Positive Level	SPI 0 Interrupt
53	I2C0_INT	Positive Level	I2C 0 Interrupt
54	I2C1_INT	Positive Level	I2C 1 Interrupt
55	SMC0_INT	Positive Level	SmartCard 0 Interrupt
56	SMC1_INT	Positive Level	SmartCard 1 Interrupt
57	GPF_INT	Positive Level	GPIO F Interrupt
58	CAN0_INT	Positive Level	CAN 0 Interrupt
59	CAN1_INT	Positive Level	CAN 1 Interrupt
60	PWM0_INT	Positive Level	PWM 0 Interrupt
61	PWM1_INT	Positive Level	PWM 1 interrupt
62	CAN2_INT	Positive Level	CAN 2 Interrupt
63	GPG_INT	Positive Level	GPIO G Interrupt

4.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Address	R/W	Description	Reset Value
AIC_BA = 0xB800_2000				
AIC_SRCCTL0	AIC_BA+0x000	R/W	Source Control Register 0	0x4747_4747
AIC_SRCCTL1	AIC_BA+0x000	R/W	Source Control Register 1	0x4747_4747
AIC_SRCCTL2	AIC_BA+0x004	R/W	Source Control Register 2	0x4747_4747
AIC_SRCCTL3	AIC_BA+0x008	R/W	Source Control Register 3	0x4747_4747
AIC_SRCCTL4	AIC_BA+0x00C	R/W	Source Control Register 4	0x4747_4747
AIC_SRCCTL5	AIC_BA+0x010	R/W	Source Control Register 5	0x4747_4747
AIC_SRCCTL6	AIC_BA+0x014	R/W	Source Control Register 6	0x4747_4747
AIC_SRCCTL7	AIC_BA+0x018	R/W	Source Control Register 7	0x4747_4747
AIC_SRCCTL8	AIC_BA+0x01C	R/W	Source Control Register 8	0x4747_4747
AIC_SRCCTL9	AIC_BA+0x020	R/W	Source Control Register 9	0x4747_4747
AIC_SRCCTL10	AIC_BA+0x024	R/W	Source Control Register 10	0x4747_4747
AIC_SRCCTL11	AIC_BA+0x028	R/W	Source Control Register 11	0x4747_4747
AIC_SRCCTL12	AIC_BA+0x02C	R/W	Source Control Register 12	0x4747_4747
AIC_SRCCTL13	AIC_BA+0x030	R/W	Source Control Register 13	0x4747_4747
AIC_SRCCTL14	AIC_BA+0x034	R/W	Source Control Register 14	0x4747_4747
AIC_SRCCTL15	AIC_BA+0x038	R/W	Source Control Register 15	0x4747_4747
AIC_RAWSTS0	AIC_BA+0x100	R	Interrupt Raw Status Register 0	0x0000_0000

AIC_RAWSTS1	AIC_BA+0x104	R	Interrupt Raw Status Register 1	0x0000_0000
AIC_ACTSTS0	AIC_BA+0x108	R	Interrupt Active Status Register 0	0x0000_0000
AIC_ACTSTS1	AIC_BA+0x10C	R	Interrupt Active Status Register 1	0x0000_0000
AIC_INTSTS0	AIC_BA+0x110	R	Interrupt Status Register 0	0x0000_0000
AIC_INTSTS1	AIC_BA+0x114	R	Interrupt Status Register 1	0x0000_0000
AIC_IRQNUM	AIC_BA+0x120	R	IRQ Source Number Register	0x0000_0000
AIC_FIQNUM	AIC_BA+0x124	R	FIQ Source Number Register	0x0000_0000
AIC_INTMSK0	AIC_BA+0x128	R	Interrupt Mask Register 0	0x0000_0000
AIC_INTMSK1	AIC_BA+0x12C	R	Interrupt Mask Register 1	0x0000_0000
AIC_INTEN0	AIC_BA+0x130	W	Interrupt Mask Enable Command Register 0	Undefined
AIC_INTEN1	AIC_BA+0x134	W	Interrupt Mask Enable Command Register 1	Undefined
AIC_INTDIS0	AIC_BA+0x138	W	Interrupt Mask Disable Command Register 0	Undefined
AIC_INTDIS1	AIC_BA+0x13C	W	Interrupt Mask Disable Command Register 1	Undefined
AIC_EOIS	AIC_BA+0x150	W	End of IRQ Service Command Register	Undefined
AIC_EOFS	AIC_BA+0x154	W	End of FIQ Service Command Register	Undefined

5 外部總線 (External Bus Interface)

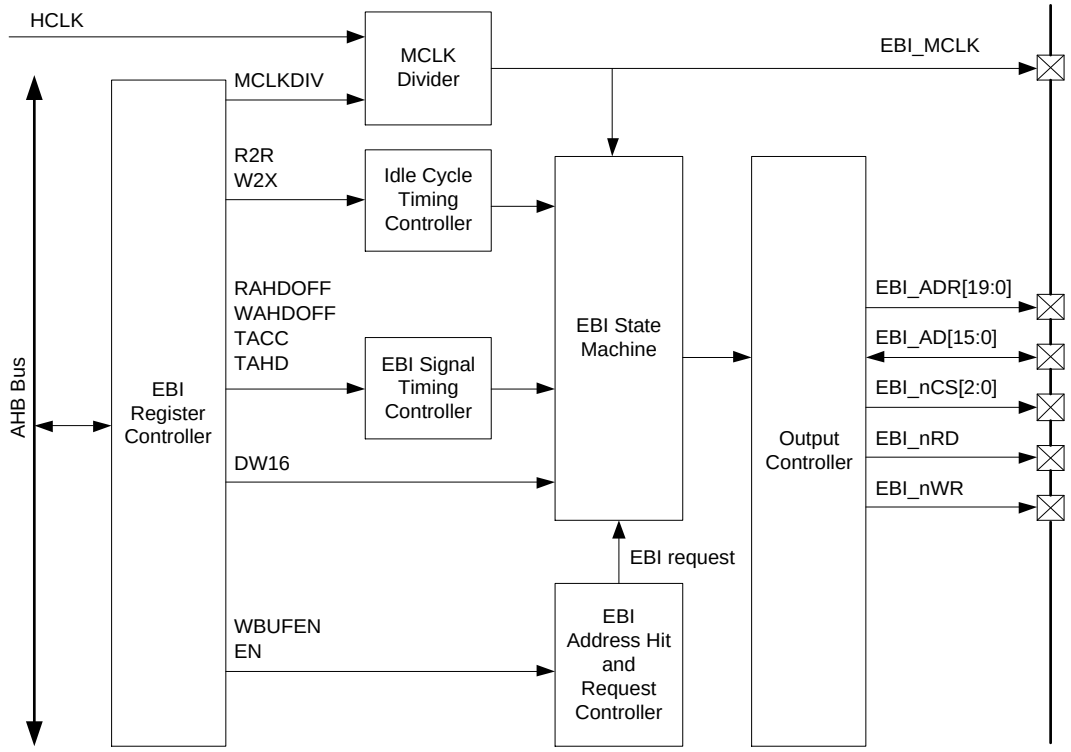
5.1 概述

NUC980系列外部總線介面 (EBI)控制訪問外部存儲器(SRAM)和外部I/O設備。EBI可連接三個各自有不同時序的20位元定址空間的外部設備。為了映射整個EBI記憶體空間，8位元資料寬度設備需20位元位址寬度，16位元資料寬度設備需19位元位址寬度。

5.2 特性

- 支援帶極性控制的兩種片選方式
- 每個片選信號控制的設備提供最高1M的位址空間
- 支援基於HCLK所產生的可設定不同頻率的外部匯流排基本時鐘(MCLK)
- 支援8位元或16位元資料寬度
- 支援可變的資料存取時間(tACC)
- 支援可配置的空閒週期以用於不同訪問條件：空閒寫命令完成(W2X)，空閒連續讀(R2R)
- 支持PDMA模式
- 支持LCD介面i80模式

5.3 方塊圖



5.4 功能描述

5.4.1 基本配置

EBI使用之前，必須將EBI(CLK_HCLKEN[9])設置為1，並將多功能腳位相關腳位設定給EBI使用。EBI腳位設定如下表：

Group	Pin Name	GPIO	MFP
EBI	EBI_AD0	PG. 10, PC. 0	MFP1
	EBI_AD1	PC. 1	MFP1
		PD. 12	MFP8
	BI_AD2	PC. 2	MFP1
		PD. 13	MFP8
	EBI_AD3	PC. 3	MFP1
		PD. 14	MFP8
	EBI_AD4	PC. 4	MFP1
		PD. 15	MFP8
	EBI_AD5	PC. 5	MFP1
		PF. 0	MFP8
	EBI_AD6	PC. 6	MFP1
		PF. 1	MFP8
	EBI_AD7	PC. 7	MFP1
		PF. 2	MFP8
	EBI_AD8	PC. 8	MFP1
		PF. 3	MFP8
	EBI_AD9	PC. 9	MFP1
		PF. 4	MFP8
	EBI_AD10	PC. 10	MFP1
		PF. 5	MFP8
	EBI_AD11	PC. 11	MFP1
		PF. 6	MFP8
	EBI_AD12	PC. 12	MFP1
		PF. 7	MFP8
	EBI_AD13	PC. 13	MFP1
		PF. 8	MFP8
	EBI_AD14	PC. 14	MFP1
		PF. 9	MFP8
	EBI_AD15	PC. 15	MFP1

		PF. 10	MFP8
	EBI_ADR0	PG. 0	MFP1
	EBI_ADR1	PG. 1	MFP1
	EBI_ADR2	PG. 2	MFP1
	EBI_ADR3	PG. 3	MFP1
	EBI_ADR4	PG. 6	MFP1
	EBI_ADR5	PG. 7	MFP1
	EBI_ADR6	PG. 8	MFP1
	EBI_ADR7	PG. 9	MFP1
	EBI_ADR8	PA. 12	MFP1
	EBI_ADR9	PA. 11	MFP1
	EBI_ADR10	PA. 10	MFP1
	EBI_ADR11	PB. 8	MFP1
	EBI_ADR12	PB. 0, PG. 5	MFP1
	EBI_ADR13	PA. 13, PB. 6	MFP1
	EBI_ADR14	PA. 14, PB. 4	MFP1
	EBI_ADR15	PB. 7	MFP1
	EBI_ADR16	PB. 5	MFP1
	EBI_ADR17	PB. 1	MFP1
	EBI_ADR18	PB. 3, PG. 4	MFP1
	EBI_ADR19	PA. 5, PB. 2	MFP1
	EBI_MCLK	PA. 1	MFP2
		PB. 2	MFP3
	EBI_nCS0	PA. 9	MFP1
	EBI_nCS1	PA. 6	MFP1
	EBI_nCS2	PA. 1	MFP1
	EBI_nRD	PA. 8	MFP1
	EBI_nWR	PA. 7	MFP1

5.4.2 操作與存取時間控制

EBI映射位址位於0x6000_0000～0x602F_FFFF，總內存空間為3MB。當系統請求地址到達EBI的存儲空間時，使對應的EBI_nCSx信號生效，啟動EBI狀態機工作。

在EBI開始執行時，EBI_nCS0、EBI_nCS1和EBI_nCS2為低電平，並等待一個EBI_MCLK用於

地址建立時間(tASU)以保持地址穩定。當EBI_nRD在讀取資料時或在寫入資料時EBI_nWR置為低電平，等待EBI_nRD或EBI_nWR在資料讀取或寫入結束的處理時間(tACC)後置為高電平。EBI信號在資料訪問保持時間(tAHD)之後，EBI_nCS0、EBI_nCS1和EBI_nCS2被置為高電平，位址由當前訪問控制釋放。

EBI控制器為不同的外部設備提供靈活的時序控制。在EBI時序控制中，tASU固定為1個EBI_MCLK。tACC可以設定為1~32個EBI_MCLK，透過寄存器設定TACC(EBI_TCTLx[7:3])，tAHD可以在1~8個EBI_MCLK，透過寄存器設定TAHD (EBI_TCTLx[10:8])。某些外部設備可以支援資料處理的保持時間為0，此時EBI控制器可以忽略tAHD，透過暫存器設定WAHDOFF(EBI_TCTLx[23])和RAHDOFF(EBI_TCTLx[22])來提高資料處理的速度。

對於每個EBI的外部設備的資料寬度、EBI_nCSx的電壓極性可以由EBI_CTLx控制；另外，暫存器EBI_TCTLx則提供各自的時序控制，時序控制週期如下表：

Parameter	Value	Unit	Description
tASU	1	MCLK	Address Latch Setup Time.
tACC	1 ~ 32	MCLK	Data Access Time. Controlled by TACC (EBI_TCTLx[7:3]).
tAHD	1 ~ 8	MCLK	Data Access Hold Time. Controlled by TAHD (EBI_TCTLx[10:8]).
IDLE	0 ~ 15	MCLK	Idle Cycle. Controlled by R2R (EBI_TCTLx[27:24]) and W2X (EBI_TCTLx[15:12]).

5.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
EBI Base Address: **EBI_BA = 0xB001_0000**				
EBI_CTL0	EBI_BA+0x00	R/W	External Bus Interface Bank0 Control Register	0x0000_0008
EBI_TCTL0	EBI_BA+0x04	R/W	External Bus Interface Bank0 Timing Control Register	0x0000_0000
EBI_CTL1	EBI_BA+0x10	R/W	External Bus Interface Bank1 Control Register	0x0000_0000
EBI_TCTL1	EBI_BA+0x14	R/W	External Bus Interface Bank1 Timing Control Register	0x0000_0000
EBI_CTL2	EBI_BA+0x20	R/W	External Bus Interface Bank2 Control Register	0x0000_0000
EBI_TCTL2	EBI_BA+0x24	R/W	External Bus Interface Bank2 Timing Control Register	0x0000_0000

6 通用 I/O 控制器 (GPIO)

6.1 概述

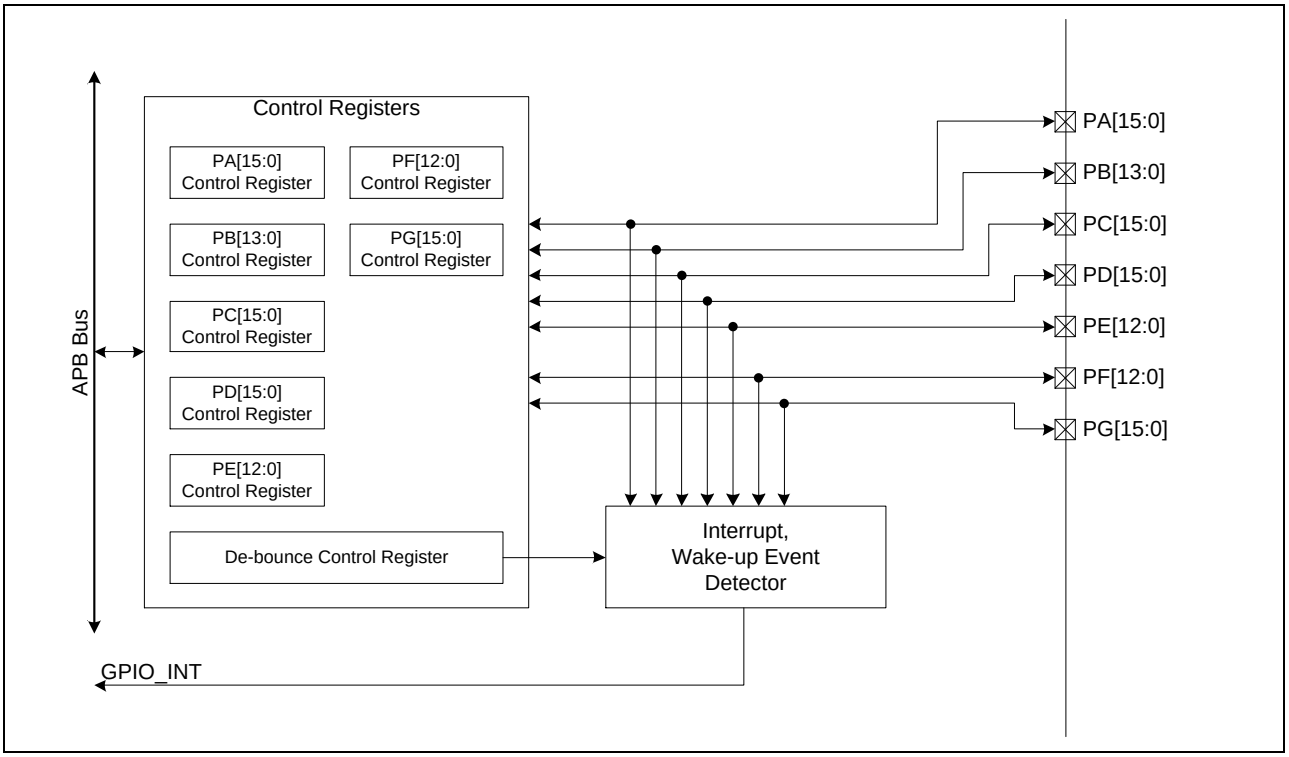
NUC980系列擁有多達104個通用 I/O 引腳，這些管腳可以和其他功能管腳共享。這些104個通用 I/O 引腳分布在PA, PB, PC, PD, PE, PF 和 PG。PA, PC, PD, 和PG各擁有16個通用 I/O 引腳，PB 擁有14個通用 I/O 引腳,PE和PF擁有13個通用 I/O 引腳。每個I/O 引腳是獨立的，可以很容易地由用戶配置的，以滿足不同的系統配置和設計要求。復位後，所有的I/O 引腳配置為通用I/O輸入模式。

當所有的I/O引腳用作通用I/O，其I/O類型可由用戶單獨配置為輸入或輸出模式。在輸入模式下，輸入緩衝區類型可以選擇為CMOS輸入緩衝器或Schmitt Trigger輸入緩衝器。每個I/ O引腳還配備一個上拉電阻(45 kΩ ~ 82 kΩ)和下拉電阻(37 kΩ ~ 91 kΩ)。上拉使能/下拉電阻是可控的。

6.2 特性

- 四種 I/O 模式:
  - 准雙向模式
  - 推挽輸出
  - 開漏輸出
  - 高阻態輸入
- 可選TTL/Schmitt 觸發輸入
- I/O可以配置為邊沿/電平觸發的中斷源
- 支援高驅動力及高翻轉速率的I/O模式。
- I/O管腳僅在准雙向模式，內部上拉電阻才使能。
- 使能管腳中斷功能同時也使能了GPIO喚醒功能。

6.3 方塊圖



6.4 功能描述

6.4.1 多功能引腳設置

GPIO配置引腳Px.n為通用I/O，需要設定相對應的多功能引腳設置，SYS_GPA_MFPL, SYS_GPA_MFPH, SYS_GPB_MFPL, SYS_GPB_MFPH, SYS_GPC_MFPL, SYS_GPC_MFPH, SYS_GPD_MFPL, SYS_GPD_MFPH, SYS_GPE_MFPL, SYS_GPE_MFPH, SYS_GPF_MFPL, SYS_GPF_MFPH, SYS_GPG_MFPL 和 SYS_GPG_MFPH 設定為 0，例如，如果希望配置引脚PA.1作为通用I/O，有必要设置MFP_GPA1(SYS_GPA_MFPL[4：7])為0。範例如下：

```
int value;
// Read SYS_GPA_MFPL register value
value = inpw(SYS_GPA_MFPL);
// Set PA.1 as I/O pin
value = value & (~0x000000F0);
// Save the setting to SYS_GPA_MFPL register
outpw(SYS_GPA_MFPL, value);
```

6.4.1.1 輸入模式

設置 MODEn (Px_MODE[2n+1:2n]) 為00, Px.n 管腳為輸入模式，I/O管腳為三態（高阻），沒有輸出驅動能力。管腳 (Px_PIN[n]) 的值反映相應埠的狀態。

6.4.1.2 推挽輸出模式

設置 MODEn (Px_MODE[2n+1:2n]) 為 01， Px.n 管腳為推挽輸出模式，I/O支援數位輸出功能，

有拉/灌電流能力。DOUT (Px_DOUT[n]) 相應位bit[n]的值被送到相應管腳上。

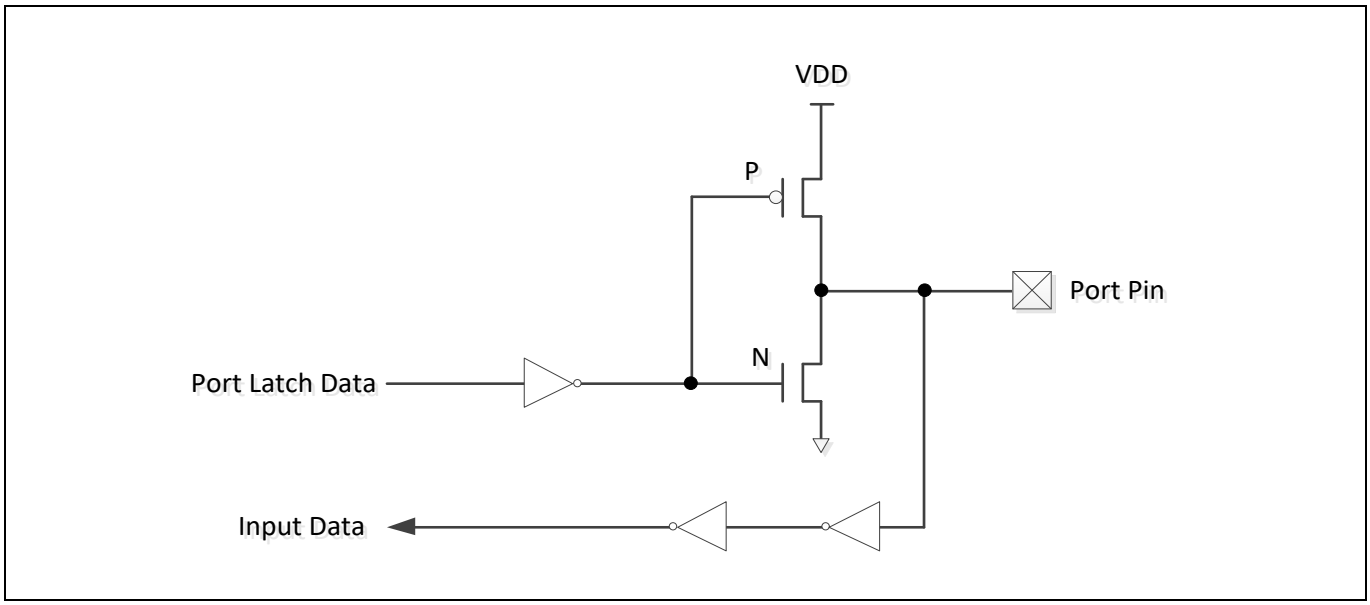


圖 6.4-1 推挽輸出

6.4.1.3 開漏輸出模式

設置 MODEn (Px_MODE[2n+1:2n]) 為 10, Px.n 管腳為開漏模式且I/O管腳數位輸出功能僅支援灌電流, 驅動到高電平需要一個外加上拉電阻。如果DOUT (Px_DOUT[n]) 相應位為‘0’, 管腳上輸出低. 如果DOUT (Px_DOUT[n]) 相應位為 ‘1’, 該管腳輸出為高阻，可以由外部上拉電阻控制。

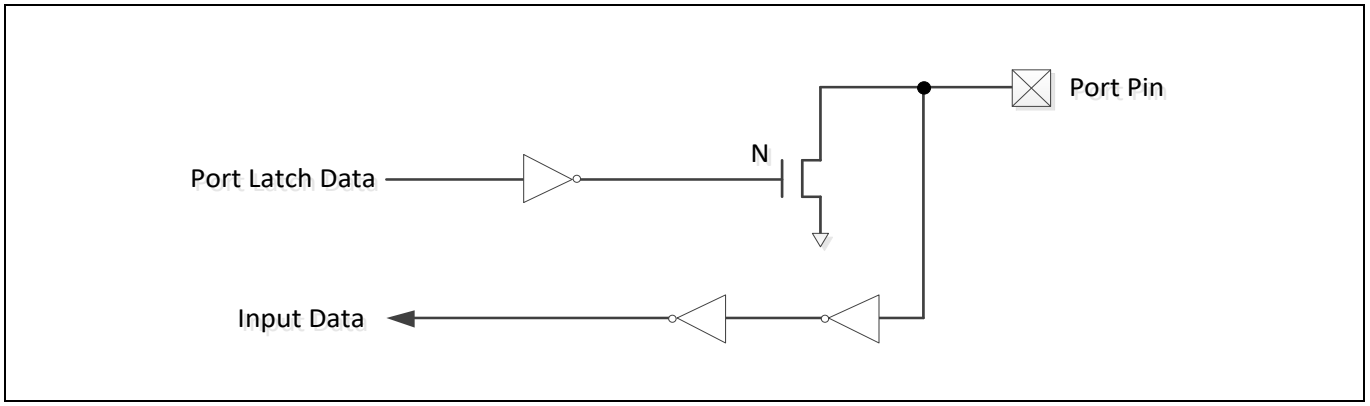


圖 6.4-2 開漏輸出

6.4.1.4 准雙向模式

設置MODEn (Px_MODE[2n+1:2n])為11，Px.n 管腳為准雙向模式，I/O同時支援數位輸出和輸入功能，但拉電流能力僅達數百uA。要實現數位輸入，需要先將DOUT (Px_DOUT[n])相應位置1。准雙向輸出是80C51 及其派生產品常見的模式。若DOUT (Px_DOUT[n])相應位bit[n]為‘0’ ，管腳上輸出為“低”。若DOUT (Px_DOUT[n])相應位bit[n]為‘1’，該管腳將檢測管腳值。若管腳值

為高，沒有任何動作，若管腳值為低，在該管腳上將有強輸出驅動2個時鐘週期的高電平，然後禁止強輸出驅動，其後管腳狀態由內部上拉電阻控制。注意: 准雙向模式source電流的大小僅有200 uA到30 uA(相應VDD的電壓從5.0 V到2.5 V) 。

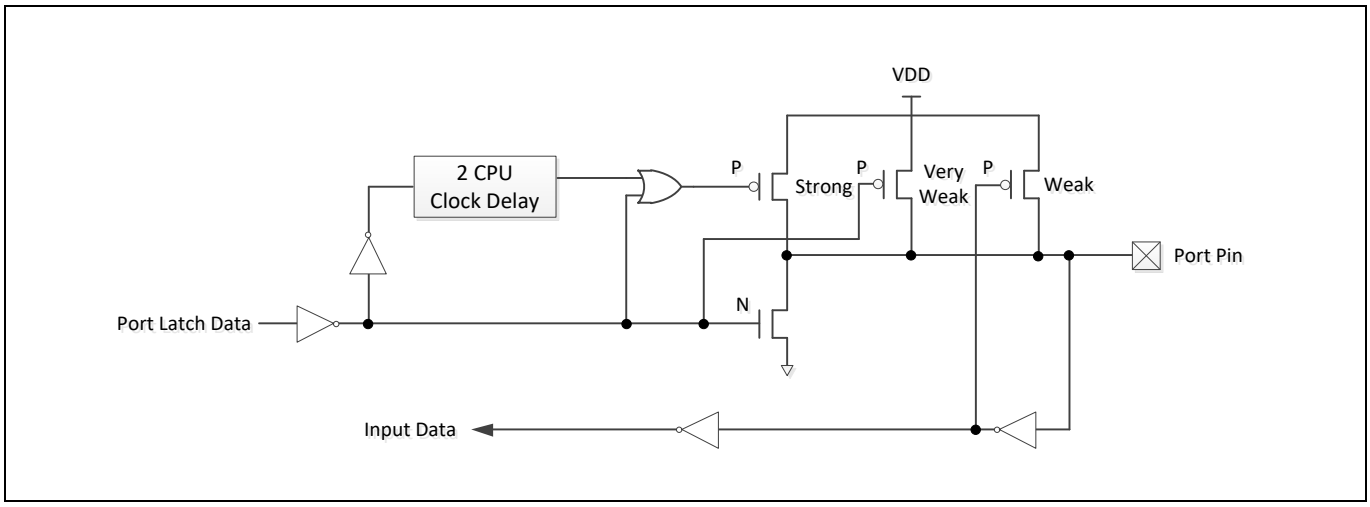


圖 6.4-3 准雙向 I/O 模式

6.4.1.5 施密特觸發(Schmitt Trigger)

對於標準施密特觸發器，當輸入電壓高於順向閾值電壓，輸出為高；當輸入電壓低於負向閾值電壓，輸出為低。當輸入訊號有雜訊的時候，施密特是可以有效過濾雜訊。

6.4.1.6 上拉/下拉模式

使用上拉/下拉模式可以確保每個管腳的輸入狀態。上拉模式只有在MODE (Px_MODE[n]))設定為推挽輸出模式或開漏輸出模式才有效。下拉模式只有在MODE (Px_MODE[n]))設定為 推挽輸出模式才有效。

6.4.1.7 禁止數位輸入路徑

使用者可以透過設置DINOFF (Px_DINOFF[n])來禁止GPIO數位輸入路徑去避免漏電的情況發生。當GPIO數位輸入路徑禁止時，GPIO輸入管腳值 PIN (Px_PIN[n])連接到低電平。

6.4.1.8 GPIO中斷和喚醒功能

每個GPIO管腳都可以通過RHIEN (Px_INTEN[n+16])/FLIEN (Px_INTEN[n]) 位和TYPE (Px_INTTYPE[n])設置成晶片的中斷源。有五種中斷觸發條件可以設置：低電平觸發、高電平觸發、下降沿觸發和上升沿觸發以及上升與下降沿同時觸發。在邊沿觸發中使用者可以通過使能輸入信號去抖功能來阻止由雜訊引起的意外中斷。去抖時鐘源和採樣週期可以通過DBCLKSRC (GPIO_DBCTL[4]) 和DBCLKSEL (GPIO_DBCTL[3:0]) 寄存器來設置。

當晶片進入Idle/Power-down模式時，GPIO也可以喚醒系統。設置GPIO為喚醒觸發的條件與GPIO中斷觸發的條件相同。

6.5 寄存器

R: read only, **W**: write only, **R/W**: both read and write.

Register	Offset	R/W	Description	Reset Value
GPIO Base Address: **GPIO_BA = 0xB000_4000**				
PA_MODE	GPIO_BA+0x000	R/W	PA I/O Mode Control	0xXXXX_XXXX
PA_DINOFF	GPIO_BA+0x004	R/W	PA Digital Input Path Disable Control	0x0000_0000
PA_DOUT	GPIO_BA+0x008	R/W	PA Data Output Value	0x0000_FFFF
PA_DATMSK	GPIO_BA+0x00C	R/W	PA Data Output Write Mask	0x0000_0000
PA_PIN	GPIO_BA+0x010	R	PA Pin Value	0x0000_XXXX
PA_DBEN	GPIO_BA+0x014	R/W	PA De-Bounce Enable Control Register	0x0000_0000
PA_INTTYPE	GPIO_BA+0x018	R/W	PA Interrupt Trigger Type Control	0x0000_0000
PA_INTEN	GPIO_BA+0x01C	R/W	PA Interrupt Enable Control Register	0x0000_0000
PA_INTSRC	GPIO_BA+0x020	R/W	PA Interrupt Source Flag	0x0000_XXXX
PA_SMTEN	GPIO_BA+0x024	R/W	PA Input Schmitt Trigger Enable Register	0x0000_0000
PA_SLEWCTL	GPIO_BA+0x028	R/W	PA High Slew Rate Control Register	0x0000_0000
PB_MODE	GPIO_BA+0x040	R/W	PB I/O Mode Control	0xXXXX_XXXX
PB_DINOFF	GPIO_BA+0x044	R/W	PB Digital Input Path Disable Control	0x0000_0000
PB_DOUT	GPIO_BA+0x048	R/W	PB Data Output Value	0x0000_FFFF
PB_DATMSK	GPIO_BA+0x04C	R/W	PB Data Output Write Mask	0x0000_0000
PB_PIN	GPIO_BA+0x050	R	PB Pin Value	0x0000_XXXX
PB_DBEN	GPIO_BA+0x054	R/W	PB De-Bounce Enable Control Register	0x0000_0000
PB_INTTYPE	GPIO_BA+0x058	R/W	PB Interrupt Trigger Type Control	0x0000_0000
PB_INTEN	GPIO_BA+0x05C	R/W	PB Interrupt Enable Control Register	0x0000_0000
PB_INTSRC	GPIO_BA+0x060	R/W	PB Interrupt Source Flag	0x0000_XXXX
PB_SMTEN	GPIO_BA+0x064	R/W	PB Input Schmitt Trigger Enable Register	0x0000_0000
PB_SLEWCTL	GPIO_BA+0x068	R/W	PB High Slew Rate Control Register	0x0000_0000
PC_MODE	GPIO_BA+0x080	R/W	PC I/O Mode Control	0xXXXX_XXXX
PC_DINOFF	GPIO_BA+0x084	R/W	PC Digital Input Path Disable Control	0x0000_0000
PC_DOUT	GPIO_BA+0x088	R/W	PC Data Output Value	0x0000_FFFF
PC_DATMSK	GPIO_BA+0x08C	R/W	PC Data Output Write Mask	0x0000_0000

PC_PIN	GPIO_BA+0x090	R	PC Pin Value	0x0000_XXXX
PC_DBEN	GPIO_BA+0x094	R/W	PC De-Bounce Enable Control Register	0x0000_0000
PC_INTTYPE	GPIO_BA+0x098	R/W	PC Interrupt Trigger Type Control	0x0000_0000
PC_INTEN	GPIO_BA+0x09C	R/W	PC Interrupt Enable Control Register	0x0000_0000
PC_INTSRC	GPIO_BA+0x0A0	R/W	PC Interrupt Source Flag	0x0000_XXXX
PC_SMTEN	GPIO_BA+0x0A4	R/W	PC Input Schmitt Trigger Enable Register	0x0000_0000
PC_SLEWCTL	GPIO_BA+0x0A8	R/W	PC High Slew Rate Control Register	0x0000_0000
PD_MODE	GPIO_BA+0x0C0	R/W	PD I/O Mode Control	0xXXXX_XXXX
PD_DINOFF	GPIO_BA+0x0C4	R/W	PD Digital Input Path Disable Control	0x0000_0000
PD_DOUT	GPIO_BA+0x0C8	R/W	PD Data Output Value	0x0000_FFFF
PD_DATMSK	GPIO_BA+0x0CC	R/W	PD Data Output Write Mask	0x0000_0000
PD_PIN	GPIO_BA+0x0D0	R	PD Pin Value	0x0000_XXXX
PD_DBEN	GPIO_BA+0x0D4	R/W	PD De-Bounce Enable Control Register	0x0000_0000
PD_INTTYPE	GPIO_BA+0x0D8	R/W	PD Interrupt Trigger Type Control	0x0000_0000
PD_INTEN	GPIO_BA+0x0DC	R/W	PD Interrupt Enable Control Register	0x0000_0000
PD_INTSRC	GPIO_BA+0x0E0	R/W	PD Interrupt Source Flag	0x0000_XXXX
PD_SMTEN	GPIO_BA+0x0E4	R/W	PD Input Schmitt Trigger Enable Register	0x0000_0000
PD_SLEWCTL	GPIO_BA+0x0E8	R/W	PD High Slew Rate Control Register	0x0000_0000
PE_MODE	GPIO_BA+0x100	R/W	PE I/O Mode Control	0xXXXX_XXXX
PE_DINOFF	GPIO_BA+0x104	R/W	PE Digital Input Path Disable Control	0x0000_0000
PE_DOUT	GPIO_BA+0x108	R/W	PE Data Output Value	0x0000_FFFF
PE_DATMSK	GPIO_BA+0x10C	R/W	PE Data Output Write Mask	0x0000_0000
PE_PIN	GPIO_BA+0x110	R	PE Pin Value	0x0000_XXXX
PE_DBEN	GPIO_BA+0x114	R/W	PE De-Bounce Enable Control Register	0x0000_0000
PE_INTTYPE	GPIO_BA+0x118	R/W	PE Interrupt Trigger Type Control	0x0000_0000
PE_INTEN	GPIO_BA+0x11C	R/W	PE Interrupt Enable Control Register	0x0000_0000
PE_INTSRC	GPIO_BA+0x120	R/W	PE Interrupt Source Flag	0x0000_XXXX
PE_SMTEN	GPIO_BA+0x124	R/W	PE Input Schmitt Trigger Enable Register	0x0000_0000
PE_SLEWCTL	GPIO_BA+0x128	R/W	PE High Slew Rate Control Register	0x0000_0000
PE_DRVCTL	GPIO_BA+0x12C	R/W	PE High Drive Strength Control Register	0x0000_0000
PF_MODE	GPIO_BA+0x140	R/W	PF I/O Mode Control	0x0000_XXXX

PF_DINOFF	GPIO_BA+0x144	R/W	PF Digital Input Path Disable Control	0x0000_0000
PF_DOUT	GPIO_BA+0x148	R/W	PF Data Output Value	0x0000_00FF
PF_DATMSK	GPIO_BA+0x14C	R/W	PF Data Output Write Mask	0x0000_0000
PF_PIN	GPIO_BA+0x150	R	PF Pin Value	0x0000_00XX
PF_DBEN	GPIO_BA+0x154	R/W	PF De-Bounce Enable Control Register	0x0000_0000
PF_INTTYPE	GPIO_BA+0x158	R/W	PF Interrupt Trigger Type Control	0x0000_0000
PF_INTEN	GPIO_BA+0x15C	R/W	PF Interrupt Enable Control Register	0x0000_0000
PF_INTSRC	GPIO_BA+0x160	R/W	PF Interrupt Source Flag	0x0000_00XX
PF_SMTEN	GPIO_BA+0x164	R/W	PF Input Schmitt Trigger Enable Register	0x0000_0000
PF_SLEWCTL	GPIO_BA+0x168	R/W	PF High Slew Rate Control Register	0x0000_0000
PG_MODE	GPIO_BA+0x180	R/W	PG I/O Mode Control	0x0000_XXXX
PG_DINOFF	GPIO_BA+0x184	R/W	PG Digital Input Path Disable Control	0x0000_0000
PG_DOUT	GPIO_BA+0x188	R/W	PG Data Output Value	0x0000_00FF
PG_DATMSK	GPIO_BA+0x18C	R/W	PG Data Output Write Mask	0x0000_0000
PG_PIN	GPIO_BA+0x190	R	PG Pin Value	0x0000_00XX
PG_DBEN	GPIO_BA+0x194	R/W	PG De-Bounce Enable Control Register	0x0000_0000
PG_INTTYPE	GPIO_BA+0x198	R/W	PG Interrupt Trigger Type Control	0x0000_0000
PG_INTEN	GPIO_BA+0x19C	R/W	PG Interrupt Enable Control Register	0x0000_0000
PG_INTSRC	GPIO_BA+0x1A0	R/W	PG Interrupt Source Flag	0x0000_00XX
PG_SMTEN	GPIO_BA+0x1A4	R/W	PG Input Schmitt Trigger Enable Register	0x0000_0000
PG_SLEWCTL	GPIO_BA+0x1A8	R/W	PG High Slew Rate Control Register	0x0000_0000
GPIO_DBCTL	GPIO_BA+0x440	R/W	Interrupt De-bounce Control Register	0x0000_0020
PAn_PDIO n=0,1..15	GPIO_BA+0x800+(0x04 * n)	R/W	GPIO PA.n Pin Data Input/Output Register	0x0000_000X
PBn_PDIO n=0,1..15	GPIO_BA+0x840+(0x04 * n)	R/W	GPIO PB.n Pin Data Input/Output Register	0x0000_000X
PCn_PDIO n=0,1..15	GPIO_BA+0x880+(0x04 * n)	R/W	GPIO PC.n Pin Data Input/Output Register	0x0000_000X
PDn_PDIO n=0,1..15	GPIO_BA+0x8C0+(0x04 * n)	R/W	GPIO PD.n Pin Data Input/Output Register	0x0000_000X
PEn_PDIO n=0,1..14	GPIO_BA+0x900+(0x04 * n)	R/W	GPIO PE.n Pin Data Input/Output Register	0x0000_000X
PFn_PDIO	GPIO_BA+0x940+(0x04 * n)	R/W	GPIO PF.n Pin Data Input/Output Register	0x0000_000X

n=0,1..7				

7 外設 DMA 控制器 (PDMA)

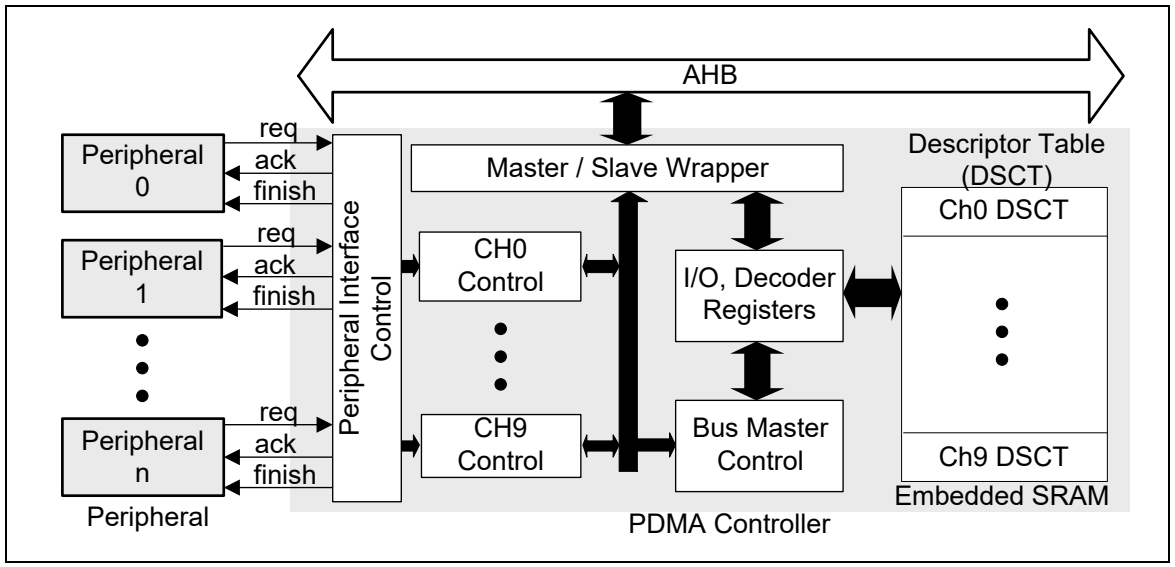
7.1 概述

NUC980系列有兩個PDMA控制器，PDMA控制器可以從一個位址到另一個位址傳輸資料，無需CPU介入。這樣做的好處是減少CPU的工作量，把節省下的CPU資源做其他應用。PDMA控制器包含10個通道，每個通道支援記憶體和外設之間的資料傳輸和記憶體與記憶體之間的資料傳輸。。

7.2 特性

- 支援2個PDMA控制器, PDMA0 和 PDMA1。
- 支援10個可獨立配置的通道。
- 支持2種優先順序選擇(固定優先順序(fixed priority)或輪循調度優先順序(round-robin priority)) 。
- 傳輸資料寬度支援8位元，16位，32位。
- 支援來源位址與目的地址方向遞增或固定，資料寬度支援位元組，半字，字。
- 支援軟體，SPI, UART, ADC 和 TIMER等請求。
- 支援Scatter-Gather模式，通過描述符鏈表執行靈活的資料傳輸。
- 支援單一和批量傳輸傳輸類型。
- 支援超時偵測。
- 0~5通道支援stride 模式。

7.3 方塊圖



7.4 功能描述

直接記憶體存取(PDMA)控制器可以從一個位址到另一個位址傳輸資料，無需CPU介入。PDMA有10個獨立通道，因為每次只有一個通道工作，因此，PDMA 控制器支援兩種通道優先順序：固定優先順序和調度優先順序(round-robin priority)，PDMA控制器通道執行的優先順序是從高到低的。PDMA 控制器支援兩種運行模式：基本模式和Scatter-gather模式。基本模式用於按照一個傳輸描述符表格傳輸資料。Scatter-gather模式對於每個PDMA都有多個傳輸描述符表格，所以PDMA控制器通過這些表格，實現靈活的資料傳輸，傳輸描述符表資料結構包含了傳送資訊，例如：傳輸源位址，傳輸定義位址，傳輸計數，批量傳輸資料大小，傳輸類型和操作模式。下圖是描述符表(DSCT)資料結構。

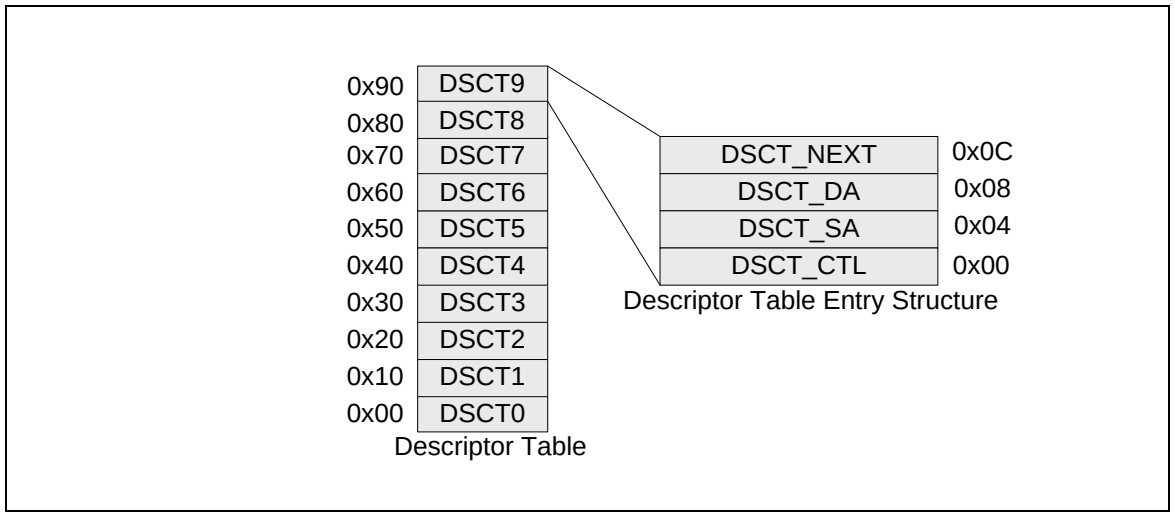


圖 7.4-1 描述符表入口結構

PDMA 控制器也支援單一和成組資料的傳輸類型，請求源可以是軟體請求，介面請求，記憶體之間的資料傳輸是使用軟體請求。單一傳輸的意思是軟體或介面準備好傳輸一個資料(每個資料需要一次請求)，批量傳輸的意思是軟體或介面將傳輸多個資料(多個資料僅需一次請求)

7.4.1 通道優先順序

PDMA控制器支援兩級通道優先等級，包括固定優先順序和輪循調度優先順序(round-robin priority)。固定優先順序比輪循調度優先順序(round-robin priority)的優先順序別更高。如果多路通道設定為固定優先順序或調度優先順序(round-robin priority)，高通道的優先順序別也高。優先順序原則如下表。

PDMA_PRISET	通道數	優先順序設定	優先順序遞減順序
1	11	Channel11, 固定優先順序(Fixed Priority)	最高(Highest)
1	10	Channel10, 固定優先順序(Fixed Priority)	---
---	---	---	---
1	0	Channel0, 固定優先順序(Fixed Priority)	---
0	11	Channel11, 輪循調度優先順序(Round-Robin Priority)	---
0	10	Channel10, 輪循調度優先順序(Round-Robin Priority)	---
---	---	---	---
0	0	Channel0, 輪循調度優先順序(Round-Robin Priority)	最低(Lowest)

表 7.4-1 通道優先順序表

7.4.2 PDMA 操作模式

PDMA支援兩種操作模式，包括: 基本模式和Scatter-Gather模式。

基本模式

基本模式用於執行一個描述符表格的傳輸模式。該模式用於記憶體與記憶體之間的資料傳輸或介面與記憶體之間的資料傳輸。PDMA 控制器操作模式可以通過寄存器OPMODE (PDMA_DSCTn_CTL[1:0], n 代表PDMA 通道數) 設定, 預設設置是在空閒狀態(OPMODE (PDMA_DSCTn_CTL[1:0]) = 0x0)，推薦用戶設定描述符表為空閒狀態。如果操作模式不是空閒狀態，使用者重新配置通道設定可能會引起操作錯誤。

用戶必須填寫傳輸計數寄存器TXCNT (PDMA_DSCTn_CTL[29:16])和傳輸寬度選擇寄存器TXWIDTH (PDMA_DSCTn_CTL[13:12]), 目的地址遞增大小寄存器DAINC (PDMA_DSCTn_CTL[11:10])，源位址遞增大小寄存器SAINC (PDMA_DSCTn_CTL[9:8]),批量大小寄存器BURSIZE (PDMA_DSCTn_CTL[6:4]) 和傳輸類型寄存器TXTYPE(PDMA_DSCTn_CTL[2])，那麼PDMA控制器將在接收到請求信號後執行傳輸操作。如果相應的PDMA 中斷位INTENn (PDMA_INTEN[11:0])使能，傳輸完成後將給CPU產生一個中斷，操作模式將被更新為空閒模式，如下圖。如果軟體配置操作模式為空閒狀態，PDMA控制器不會執行任何傳輸，並清除這個操作請求。如果相應的PDMA中斷位被使能，完成這個任務後也會給CPU產生中斷。

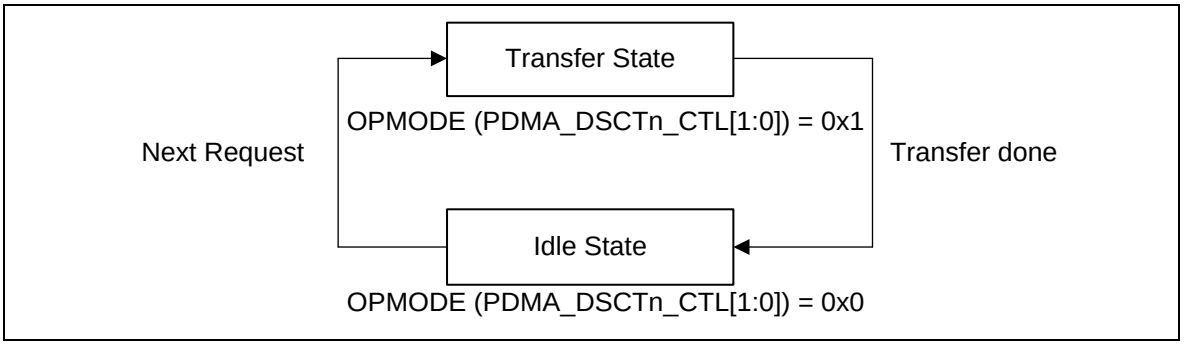


圖 7.4-2 基本模式的有限狀態機

Scatter-Gather 模式

Scatter-Gather模式是一個綜合模式，通過如下圖描述符鏈表設定，可以實現靈活的資料傳輸。通過該模式，使用者可以實現外設的回環訪問。用於多路PDMA任務或用於系統記憶體的不同位置(非相鄰位置)的資料搬移。

在Scatter-Gather模式，這個表用於跳轉到下一個表的入口。第一個任務不會做資料搬移操作。如果相應的PDMA 中斷位使能，寄存器TBINTDIS (PDMA_DSCTn_CTL[7])=0，完成每個任務後，將給CPU產生一個中斷，(任務完成，TBINTDIS位為“0”，會插入相應的寄存器TDIFn (PDMA_TDSTS[11:0])標誌，TBINTDIS位元為“1”，禁用TDIFn)

如果觸發了通道11，在Scatter-Gather(OPMODE (PDMA_DSCTn_CTL[1:0]) = 0x10x2)模式，硬體將把寄存器PDMA_DSCTn_NEXT (link address)和PDMA_SCATBA (基底位址)相加來獲取真正的PDMA任務資訊。例如，基底位址是0x2000_0000(PDMA_SCATBA寄存器中僅MSB 16位元有效)，目前連結位址是0x0000_0100(在寄存器PDMA_DSCTn_NEXT中，僅LSB 16位(不包括[1:0])有效)，那麼，下一個DSCT的起始位址是0x2000_0100.

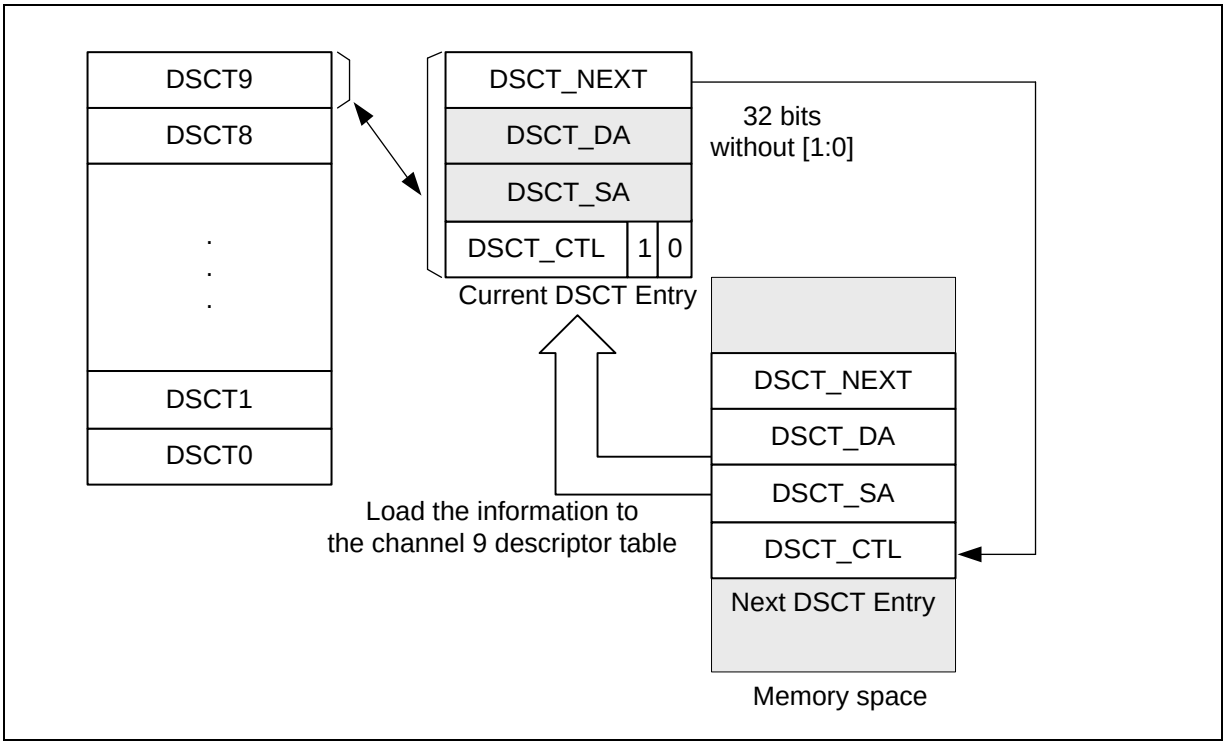


圖 7.4-3 描述符表格連結單結構

上述連結清單操作是Scatter-Gather 模式的DSCT狀態。當載入資訊結束後，將自動進入搬移資料狀態。然而，如果下一個PDMA資訊也是Scatter-Gather模式，當前任務結束後，硬體將抓下一個PDMA塊資訊。Scatter-Gather模式只有在PDMA控制器操作模式切換到基本模式並完成最後一次傳輸或者直接切換到空閒狀態後才會結束。

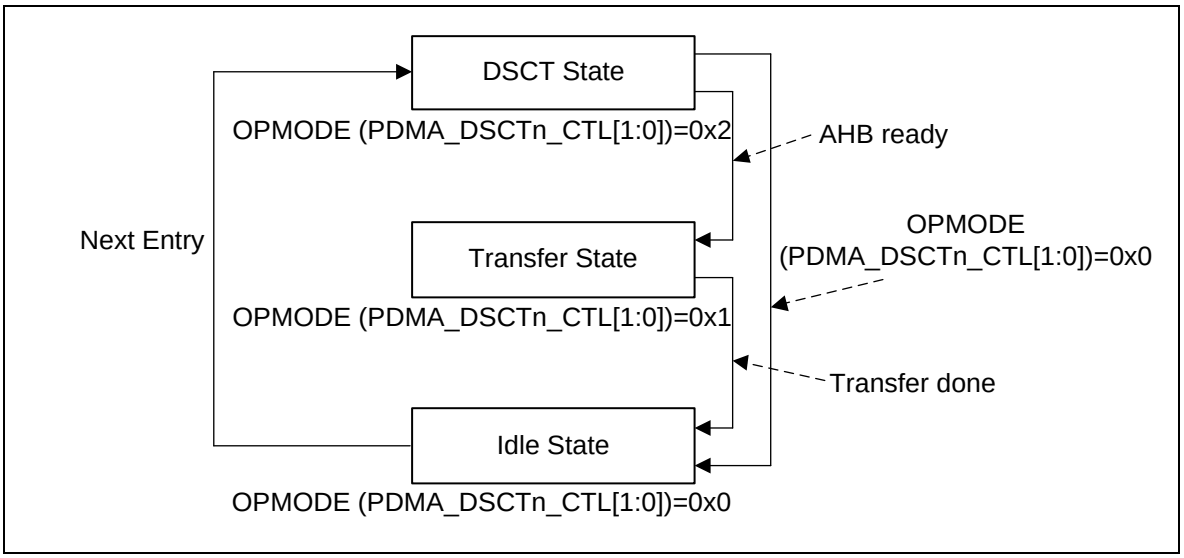


圖 7.4-4 Scatter-Gather 模式的有限狀態機

7.4.3 批量傳輸 Transfer Type 傳輸類型

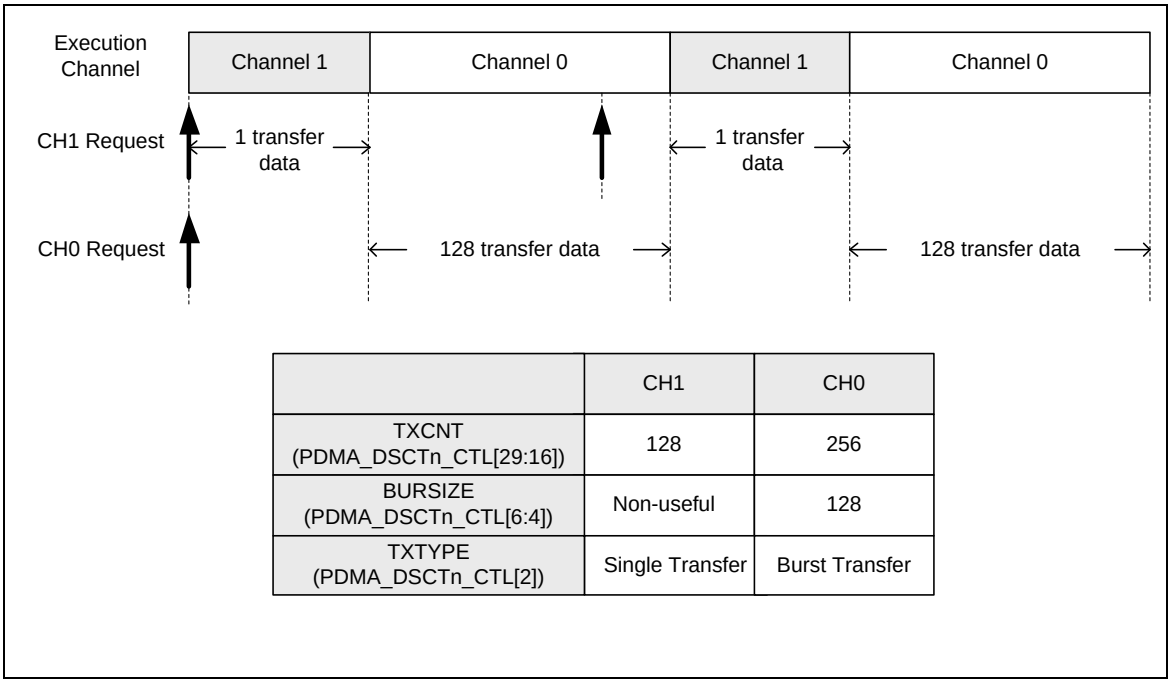
PDMA 控制器支援兩種傳輸模式：單一傳輸和批量傳輸模式，通過寄存器TXTYPE (PDMA_DSCTn_CTL[2])設定。

當PDMA控制器運行在單一傳輸模式，每搬移一個資料需要一次請求，當搬移一次資料，寄存器TXCNT (PDMA_DSCTn_CTL[29:16])會減1，直到寄存器TXCNT (PDMA_DSCTn_CTL[29:16])為0，搬移才會完成。在該模式，寄存器BURSIZE (PDMA_DSCTn_CTL[6:4])不用於控制搬移資料量大小。BURSIZE (PDMA_DSCTn_CTL[6:4])數值是固定的。

在批量搬移模式，PDMA控制器搬移TXCNT (PDMA_DSCTn_CTL[29:16])個資料，僅需一次請求。當搬移BURSIZE (PDMA_DSCTn_CTL[6:4]) 個資料後，寄存器TXCNT (PDMA_DSCTn_CTL[29:16]) 中的數目會減去BURSIZE。直到TXCNT (PDMA_DSCTn_CTL[29:16])遞減為0，搬移資料才完成。該模式用於PDMA控制器做批量資料的搬移，但用戶需確認外設是否支援批量資料傳輸。

圖7.4-5是基本傳輸模式的單一傳輸和批量傳輸模式舉例。在這個例子，通道1使用單一傳輸模式，TXCNT (PDMA_DSCTn_CTL[29:16]) = 128. 通道0使用批量傳輸模式，BURSIZE (PDMA_DSCTn_CTL[6:4]) = 128 並且TXCNT (PDMA_DSCTn_CTL[29:16]) = 256. 操作順序如下：

1. 通道 0 與通道 1 同時接收到觸發信號。
2. 默認通道 1 的優先順序高於通道 0；PDMA 控制器將先載入通道 1 的描述符表並執行。但通道 1 是單一傳輸模式，所以 PDMA 控制器僅傳輸一個資料。
3. 然後，PDMA 控制器切換到通道 0 並載入通道 0 的描述符表。通道 0 是批量傳輸模式，大小是 128 個.所以，PDMA 控制器將傳輸 128 個資料。
4. 當通道 0 搬運 128 個資料，通道 1 得到另一個請求信號，所以當通道 0 搬運完 128 個資料，PDMA 控制器將切回通道 1，來搬運通道 1 的下一個資料。
5. 當通道 1 搬運完資料， PDMA 控制器切換到低優先順序的通道 0 來繼續搬運下一組 128 個資料。
6. 當通道 0 完成 256 次數據搬移，通道 1 完成 128 次數據搬移，PDMA 將完成資料搬移。



	CH1	CH0
TXCNT (PDMA_DSCTn_CTL[29:16])	128	256
BURSIZE (PDMA_DSCTn_CTL[6:4])	Non-useful	128
TXTYPE (PDMA_DSCTn_CTL[2])	Single Transfer	Burst Transfer

圖7.4-5 基本模式的單字傳輸和批量傳輸舉例

7.4.4 通道 Time-Out

當通道被選擇傳輸到指定的外設中並使能，相應的通道 Time-Out TOUTENn (PDMA_TOUTEN[n], n=0,1…9)也被使能，那麼，在通道從外設收到觸發信號後，通道的相應Time-Out向上計數器就開始從0開始計數。

Time-Out 計數器的時鐘由 HCLK 分頻獲得，通過設置相應通道的 TOUTPSCn (PDMA_TOUTPSC[2+4n:4n], n=0,1…9)寄存器。當通道相應的TOUTIENn (PDMA_TOUTIEN[n], n=0,1…9)寄存器被使能，如果Time-Out計數器從0向上計數到寄存器TOCn (PDMA_TOC0_1[16(n+1)-1):16n], n=0,1…9)中設定的值，PDMA控制器將產生一個中斷信號。當Time-Out發生時，相應通道的REQTOFn (PDMA_INTSTS[n+8], n=0,1…9) 寄存器將置位，用以表明通道發生了Time-Out。

在計數器計數到TOCn (PDMA_TOC0_1[16(n+1)-1):16n]寄存器中設定的值或者收到觸發信號，Timer-out功能被禁止或者晶片進入到掉電模式，Time-Out計數器就重定到0。

圖 7.4-6是一個關於Time-out計數器操作的例子，操作順序如下描述：

1. 通過設置 TOUTEN0(PDMA_TOUTEN[0])為 1，使能通道 0 的 time-out 功能，但此時 time-out 計數器還沒有開始計數。
2. 當收到第一個外設請求後，time-out 計數器開始從 0 向上計數到 TOC0(PDMA_TOC0_1[15:0]) 中設定的值。
3. 當收到外設第二個請求後，time-out 計數器復位到 0。

4. 當 time-out 計數器計數到 5 時，通道 0 的 time-out 標誌(REQTOF0(PDMA_INTSTS[8]))將會被置高。通道 0 計數器將繼續從 0 到 5 迴圈計數。使用者可以清掉 REQTOF0 標誌，然後輪循 REQTOF0 標誌來檢查下一個 time-out 是否發生。
5. 當 time-out 功能被禁用時，time-out 計數器復位到 0。

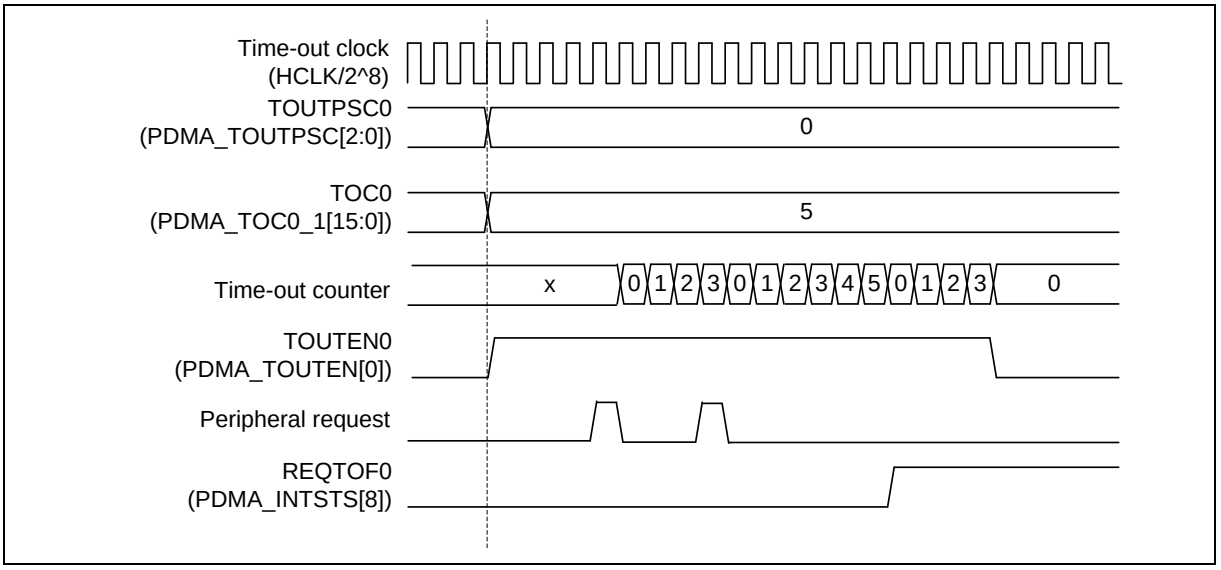


圖 7.4-6 PDMA 通道0 Time-out 計數器操作舉例

7.4.5 通道 Stride

只有通道0到通道5支持Stride功能。Stride功能可以支持位址傳輸到多個位址也可以支持區塊的傳送。使用Stride功能時，傳輸位址可以設定成固定位址或遞增模式。當通道STRIDEEN (PDMA_DSCTn_CTL[15])被使能，那麼，可以透過SASOL (PDMA_ASOCRn[15:0]) 設定來源位址的位移量,DASOL (PDMA_ASOCRn[31:16]) 設定目的位址的位移量，和 TXCNT (PDMA_DSCTn_CTL) 設定每次搬移的長度。Stride 功能可以支持軟體請求或介面請求。

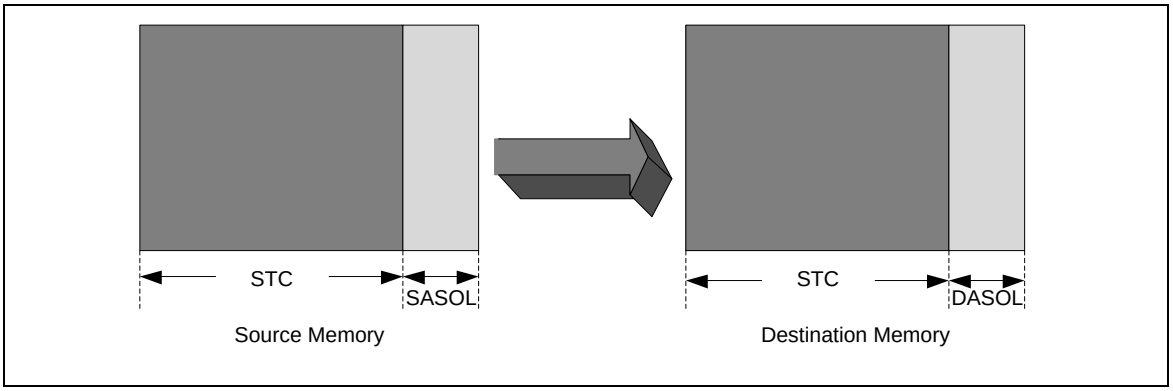


Figure 7.4-7 區塊傳送

7.5 寄存器

R: read only, **W**: write only, **R/W**: both read and write.

Register	Offset	R/W	Description	Reset Value
PDMA Base Address: **PDMAx_BA = 0xB000_8000 + (0x1000*x)** **x=0, 1** **DSCT_CTL_BA = PDMAx_BA** **DSCT_SA_BA = PDMAx_BA+4** **DSCT_DA_BA = PDMAx_BA+8** **DSCT_NEXT_BA = PDMAx_BA+c** **CURSCAT_BA = PDMAx_BA+80**				
PDMAx_DSCTn_CTL **n = 0,1..9**	DSCT_CTL_BA + 0x10 * n	R/W	Descriptor Table Control Register of PDMA Channel n	0xXXXX_XXXX
PDMAx_DSCTn_SA **n = 0,1..9**	DSCT_SA_BA + 0x10 * n	R/W	Source Address Register of PDMA Channel n	0xXXXX_XXXX
PDMAx_DSCTn_DA **n = 0,1..9**	DSCT_DA_BA + 0x10 * n	R/W	Destination Address Register of PDMA Channel n	0xXXXX_XXXX
PDMAx_DSCTn_NEXT **n = 0,1..9**	DSCT_NEXT_BA + 0x10 * n	R/W	Next Scatter-Gather Descriptor Table Offset Address of PDMA Channel n	0xXXXX_XXXX
PDMAx_CURSCATn **n = 0,1..9**	CURSCAT_BA + 0x004 * n	R	Current Scatter-Gather Descriptor Table Address of PDMA Channel n	0xXXXX_XXXX
PDMAx_CHCTL	PDMAx_BA + 0x400	R/W	PDMA Channel Control Register	0x0000_0000
PDMAx_PAUSE	PDMAx_BA + 0x404	W	PDMA Transfer Pause Control Register	0x0000_0000
PDMAx_SWREQ	PDMAx_BA + 0x408	W	PDMA Software Request Register	0x0000_0000
PDMAx_TRGSTS	PDMAx_BA + 0x40C	R	PDMA Channel Request Status Register	0x0000_0000
PDMAx_PRISET	PDMAx_BA + 0x410	R/W	PDMA Fixed Priority Setting Register	0x0000_0000
PDMAx_PRICLR	PDMAx_BA + 0x414	W	PDMA Fixed Priority Clear Register	0x0000_0000
PDMAx_INTEN	PDMAx_BA + 0x418	R/W	PDMA Interrupt Enable Register	0x0000_0000
PDMAx_INTSTS	PDMAx_BA + 0x41C	R/W	PDMA Interrupt Status Register	0x0000_0000
PDMAx_ABTSTS	PDMAx_BA + 0x420	R/W	PDMA Channel Read/Write Target Abort Flag Register	0x0000_0000
PDMAx_TDSTS	PDMAx_BA + 0x424	R/W	PDMA Channel Transfer Done Flag Register	0x0000_0000
PDMAx_ALIGN	PDMAx_BA + 0x428	R/W	PDMA Transfer Alignment Status Register	0x0000_0000
PDMAx_TACTSTS	PDMAx_BA + 0x42C	R	PDMA Transfer Active Flag Register	0x0000_0000
PDMAx_TOUTPSC	PDMAx_BA + 0x430	R/W	PDMA Time-out Prescaler Register	0x0000_0000
PDMAx_TOUTEN	PDMAx_BA + 0x434	R/W	PDMA Time-out Enable Register	0x0000_0000

PDMAx_TOUTIEN	PDMAx_BA + 0x438	R/W	PDMA Time-out Interrupt Enable Register	0x0000_0000
PDMAx_SCATBA	PDMAx_BA + 0x43C	R/W	PDMA Scatter-Gather Descriptor Table Base Address Register	0x2000_0000
PDMAx_TOC0_1	PDMAx_BA + 0x440	R/W	PDMA Time-out Counter Ch1 and Ch0 Register	0x0000_0000
PDMAx_CHRST	PDMAx_BA + 0x460	R/W	PDMA Channel Reset Register	0x0000_0000
PDMAx_REQSEL0_3	PDMAx_BA + 0x480	R/W	PDMA Request Source Select Register 0	0x0000_0000
PDMAx_REQSEL4_7	PDMAx_BA + 0x484	R/W	PDMA Request Source Select Register 1	0x0000_0000
PDMAx_REQSEL8_11	PDMAx_BA + 0x488	R/W	PDMA Request Source Select Register 2	0x0000_0000
PDMAx_STCR0	PDMAx_BA + 0x500	R/W	Stride Transfer Count Register of PDMA Channel 0	0x0000_0000
PDMAx_ASOCR0	PDMAx_BA + 0x504	R/W	Address Stride Offset Register of PDMA Channel 0	0x0000_0000
PDMAx_STCR1	PDMAx_BA + 0x508	R/W	Stride Transfer Count Register of PDMA Channel 1	0x0000_0000
PDMAx_ASOCR1	PDMAx_BA + 0x50C	R/W	Address Stride Offset Register of PDMA Channel 1	0x0000_0000
PDMAx_STCR2	PDMAx_BA + 0x510	R/W	Stride Transfer Count Register of PDMA Channel 2	0x0000_0000
PDMAx_ASOCR2	PDMAx_BA + 0x514	R/W	Address Stride Offset Register of PDMA Channel 2	0x0000_0000
PDMAx_STCR3	PDMAx_BA + 0x518	R/W	Stride Transfer Count Register of PDMA Channel 3	0x0000_0000
PDMAx_ASOCR3	PDMAx_BA + 0x51C	R/W	Address Stride Offset Register of PDMA Channel 3	0x0000_0000
PDMAx_STCR4	PDMAx_BA + 0x520	R/W	Stride Transfer Count Register of PDMA Channel 4	0x0000_0000
PDMAx_ASOCR4	PDMAx_BA + 0x524	R/W	Address Stride Offset Register of PDMA Channel 4	0x0000_0000
PDMAx_STCR5	PDMAx_BA + 0x528	R/W	Stride Transfer Count Register of PDMA Channel 5	0x0000_0000
PDMAx_ASOCR5	PDMAx_BA + 0x52C	R/W	Address Stride Offset Register of PDMA Channel 5	0x0000_0000

8 計時器控制器 (TMR)

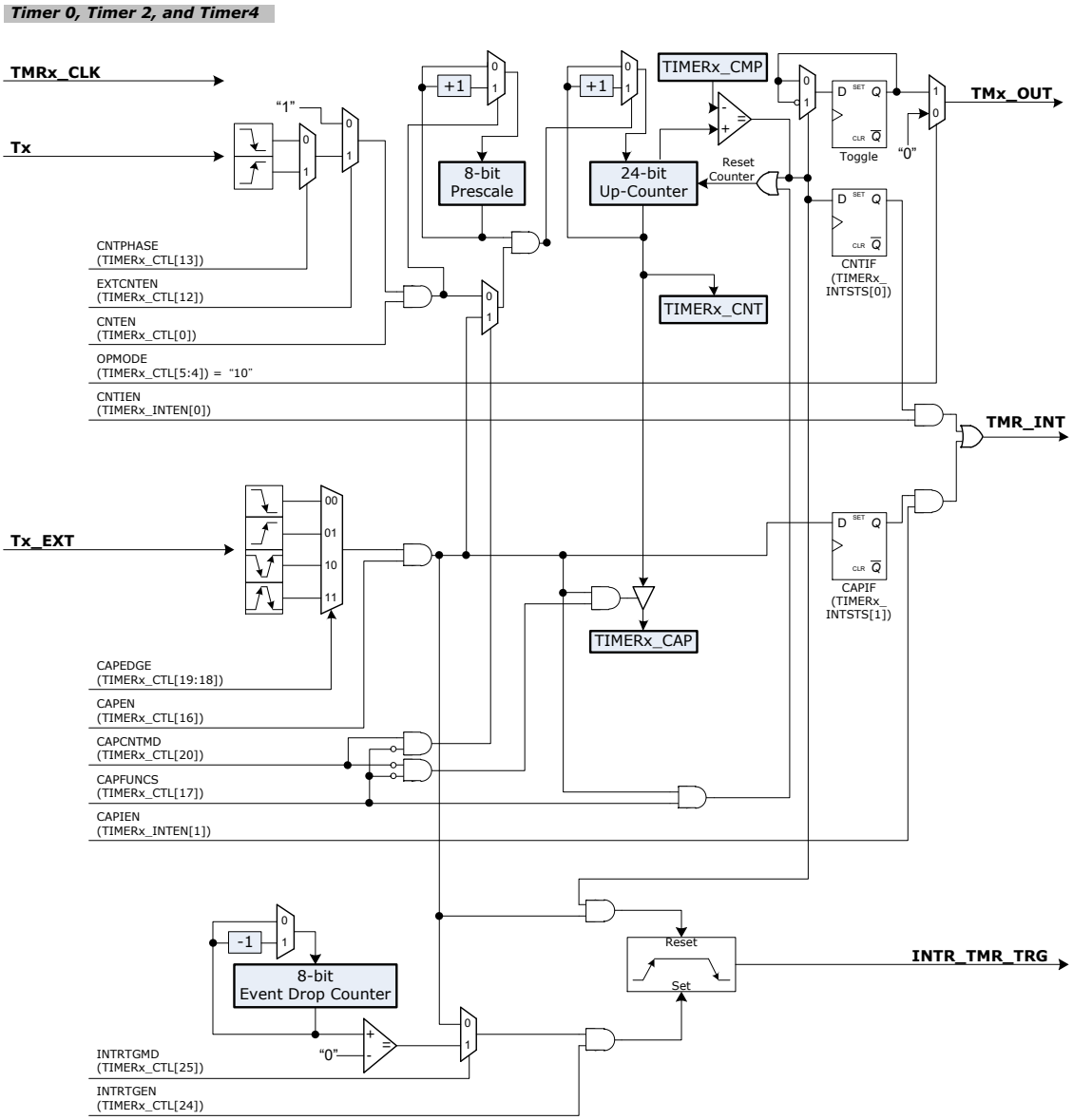
8.1 概述

該晶片帶有六個計時器模組TMR0, TMR1, TMR2, TMR3, TMR4與 TMR5, 使用者可以很方便的使用這些計時器執行計數或時間控制機制. 計時器模組可執行像頻率測量, 間隔時間測量, 時鐘產生, 延遲時間, 外部輸入引腳事件計數, 外部捕捉引腳間隔測量等功能. 計時器可在計時溢出時產生中斷信號, 也可在操作過程中提供計數的當前值.

8.2 特性

- 共6組32位計時器，每組都有一個24位的向上計數器和一個8位的預分頻計數器
- 每個計時器都有獨立的時鐘源 (TMRx_CLK, x=0,1,2,3,4,5)
- 超時週期 = (輸入給計時器的時鐘週期) * (8 位預分頻計數器 + 1) * (24 位CMP)
- 計數週期 = (1 / TMRx_CLK) * (2^8) * (2^{24})
- 內部 24 位上數型計數器通過 CNT (TMRx_CNT[23:0])可讀上數計數器的值
- 讀CAPDAT (TMRx_CAP[23:0])可得24位的捕捉值
- 支援單次、週期、輸出翻轉和連續計數操作模式
- 支援事件計數功能來計算管腳TMx_CNT (x = 0~5)上的輸入事件的個數
- 支援外部引腳輸入捕捉功能用於時間間隔測量
- 支援外部引腳輸入捕捉功能用於計時器計數器復位
- 支援用計時器中斷信號把晶片從空閒/掉電模式喚醒
- 支持用超時中斷或者捕捉中斷來觸發PDMA
- 支援計時器互觸發功能，TMR0可以觸發TMR1、TMR2可以觸發TMR3及TMR4可以觸發TMR5

8.3 方塊圖



8.4 功能描述

定時器控制器提供單次、週期、翻轉輸出和連續計數等操作模式。也提供外部引腳捕捉功能用於時間間隔測量或者復位定時器計數器。另外，定時器控制器提供定時器相互觸發模式用於精確測量輸入頻率。每個操作功能模式如下.

8.4.1 計時器計時初始化

計時器可以依以下流程初始化, 並開始計時:

1. 將 CNTEN (TMRx_CTL[0]) 清0, 停止計時器.
2. 設置操作模式 OPMODE (TMRx_CTL[5:4])

3. 若要使能中斷, 將 CNTIEN (TMRx_INTEN [0]) 置 1, 否則清 0.
4. 設置預分頻 PSC (TMRx_PRECNT[7:0])
5. 設置比較值 CMPDAT (TMRx_CMP [24:0])
6. 將CNTEN (TMRx_CTL[0]) 置 1, 計時器開始上數計數.

8.4.2 計時器捕獲初始化

計時器可以依以下流程初始化, 並開始計時:

1. 將 CNTEN (TMRx_CTL[0]) 以及 CAPEN (TMRx_CTL[16]) 清0, 停止計時器.
2. 設置操作模式 OPMODE (TMRx_CTL[5:4])
3. 若要使能中斷, 將CAPIEN (TMRx_INTEN[1]) 置 1, 否則清 0.
4. 設置 CAPCNTMD (TMRx_CTL[20]), 以及 CAPFUNCS (TMRx_CTL[17]) 選擇捕獲模式.
5. 設置 CAPEDGE (TMRx_CTL[19:18]), 選擇捕獲的觸發條件
6. 若需要去抖操作, 將 CAPDBEN (TMRx_CTL[22]) 置1, 否則清 0.
7. 設置預分頻PSC (TMRx_PRECNT[7:0])
8. 設置比較值 CMPDAT (TMRx_CMP[24:0])
9. 將CAPEN (TMRx_CTL[16]) 置1, 使能捕獲模式.
10. 將CNTEN (TMRx_CTL[0]) 置 1, 使能計時器.

注意: 在觸發計數模式下, OPMODE 的設置無意義, 只有在自由計數模式計數器重定模式下, OPMODE 的設置才可影響計數器的操作模式.

8.4.3 中斷處理

每個計時器都有自己的中斷源, 而中斷可以是溢出中斷, 或是捕獲中斷. 當觸發溢出中斷時, CNTIF (TMRx_INTSTS[0]) 會置 1. 當觸發捕獲中斷時, CAPIF (TMRx_INTSTS[1]) 會置 1. 這兩個中斷狀態標誌位可以使用寫 1 的方式清 0.

在捕獲模式下, 若是在清除 CAPIF 之前, 又有新的捕獲事件發生, 則 CAPDATOF (TMRx_INTSTS[5]) 會被置 1. 此標誌位會在清除 CAPIF 時, 一併清 0.

8.4.4 計時器頻率

每個計時計都有獨立的中斷觸發源, 觸發中斷的頻率時間可以用以下公式計算:

頻率 = TMRx_CLK / ((PRESCALE + 1) * CMP)

其中 TMRx_CLK 為計時器時鐘源, 可以是HXT (12MHz 外部晶振), PCLK, PCLK/4096, 或是LXT

(32.768kHz 外部晶振). PRESCALE 預分頻定義在 PRECNT (TMRx_PRECNT[7:0]), CMP 比較值定義在 CMPDAT (TMRx_CMP[24:0]). 以下表格列出了幾組使用設定頻率為 1Hz, 10Hz, 100Hz, 1000Hz 的方式.

輸出頻率	時鐘源	PRECNT (TMRx_PRECNT[7:0])	TMR_CMP (TMRx_CMP[24:0])
1Hz	LXT	0	0x8000
10Hz	HXT	0	0x124F80
10Hz	HXT	9	0x1D4C0
100Hz	HXT	9	0x2EE0
100Hz	HXT	19	0x1770
1000Hz	PCLK (75MHz)	4	0x3A98
1000Hz	HXT	9	0x4B0

8.4.5 單次模式

如果計時器控制器工作在單次模式 OPMODE (TMRx_CTL[5:4] 為 0x0) 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 被設置為 1, 計時器控制器開始計數. 一旦計時器計數器的值 (TMRx_CNT 值) 達到計時器比較器 (TMRx_CMP) 的值，CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1. 如果CNTIEN (TMRx_INTEN[0] 計時器中斷使能位) 被設置為 1, 且 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 為 1, 中斷信號將會被產生並被發送到 AIC 來通知 CPU 計時器計數溢出發生. 如果 CNTIEN (TMRx_INTEN[0] 計時器中斷使能位) 被設置為 0, 則不會有中斷信號產生.

在這種操作模式下, 一旦計時器計數器的值 (TMRx_CNT 的值) 達到計時器比較器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1, 計時器計數操作停止, 計時器計數器的值 (TMRx_CNT 的值) 恢復到計數初始值, 然後CNTEN (TMRx_CTL[0] 計時器計數器使能位) 被計時器控制器自動清除為 0. 也即是說, 在程式設計計時器比較寄存器 (TMRx_CMP) 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 之後, 計時器計數和與 TMRx_CMP 的值比較大操作只進行一次. 所以, 這個操作模式叫做單次模式.

8.4.6 週期模式

如果計時器工作在週期模式OPMODE (TMRx_CTL[5:4] 為 0x1) 且 CNTEN (TMRx_CTL[0] 計時

器計數器使能位) 被設置為 1, 計時器計數器開始計數. 一旦計數值 (TMRx_CNT 的值) 計時器達到比較寄存器(TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1. 如果 CNTIEN (TMRx_INTEN[0] 計時器中斷使能位) 被設置為 1, 且 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 為 1, 中斷信號產生並被發送到 AIC 來通知 CPU 計時器計數溢出發生. 如果 CNTIEN (TMRx_INTEN[0]計時器中斷使能位) 被設置為 0, 則不會有中斷信號產生.

在這種操作模式下, 一旦計時器計數器的值 (TMRx_CNT 的值) 達到計時器比較寄存器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1, 計時器計數器的值 (TMRx_CNT 的值) 恢復到計數初值, 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 保持為 1 (繼續計數使能), 計時器計數器再一次開始向上計數. 如果 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 被軟體清除, 一旦計時器計數值 (TMRx_CNT 的值) 再一次達到計時器比較寄存器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被再次設置為 1. 也即是說, 計時器計數和與 TMRx_CMP 的值比較的功能是週期性的. 計時器計數操作不會停止, 直到 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 被設置為 0. 中斷信號也週期性的被產生. 所以, 這個操作模式叫做週期模式.

8.4.7 翻轉模式

如果計時器工作在翻轉模式 OPMODE (TMRx_CTL[5:4] 為 0x2) 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 被設置為 1, 計時器計數器開始向上計數. 一旦計時器計數器的值 (TMRx_CNT 的值) 達到計時器比較寄存器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1. 如果 CNTIEN (TMRx_INTEN[0] 計時器中斷使能位) 被設置為 1, 且 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 為 1, 中斷信號產生並被發送到 AIC 來通知 CPU 計時器計數溢出發生. 如果 CNTIEN (TMRx_INTEN[0] 計時器中斷使能位) 被設置為 0, 無中斷信號產生.

在本操作模式下, 一旦計時器計數器的值 (TMRx_CNT 的值) 達到計時器比較寄存器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1, 翻轉輸出信號被設置為 1, 計時器計數器的值 (TMRx_CNT 的值) 恢復到計數初值, 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位)保持為 1 (繼續計數使能), 計時器計數器再一次開始向上計數. 如果 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 被軟體清除為 0, 一旦計時器計數器的值 (TMRx_CNT 的值) 再一次達到計時器比較寄存器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將再一次被設置為 1, 翻轉輸出信號被設置為 0. 計時器計數操作不會停止, 直到 CNTEN (TMRx_CTL[0] 計時器計數器使能位)被設置為 0. 這樣, 翻轉輸出信號輸出 50% 占空比的週期信號. 所以, 本作模式叫做翻轉模式.

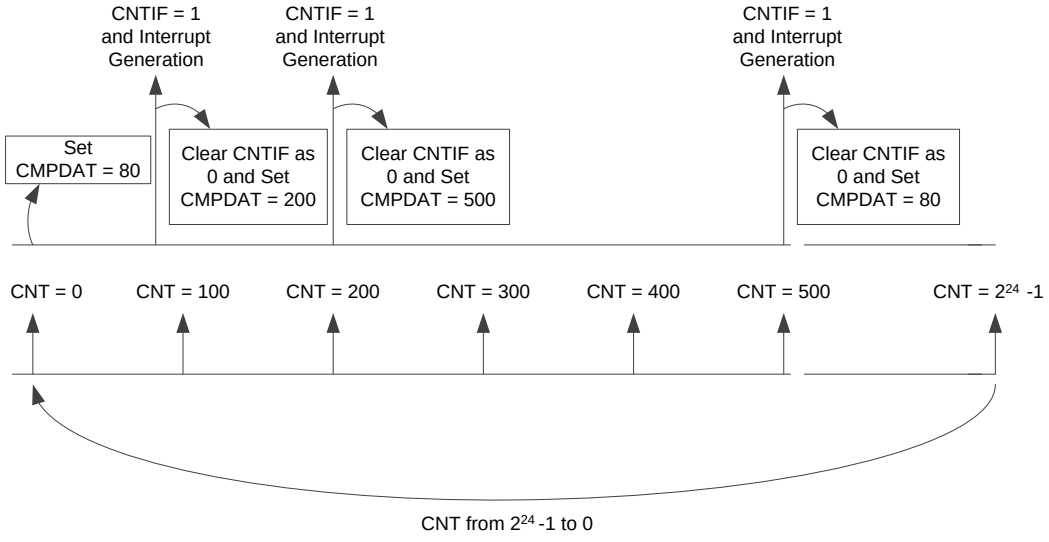
8.4.8 連續計數模式

如果計時器工作在連續計數模式 OPMODE (TMRx_CTL[5:4] 為 0x3) 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 被設置為 1, 計時器計數器開始向上計數. 一旦計時器計數器的值 (TMRx_CNT 的值) 達到計時器比較寄存器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1. 如果 CNTIEN (TMRx_INTEN[0] 計時器中斷使能位) 被設置為 1, 且 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 為 1, 中斷信號產生並被發送到 AIC 來通知 CPU 計時器計數溢出發生. 如果 CNTIEN (TMRx_INTEN[0] 計時器中斷使能位元) 被設置為 0, 則不會有中斷信號產生.

在本操作模式下, 一旦計時器計數器的值 (TMRx_CNT 的值) 達到計時器比較寄存器 (TMRx_CMP) 的值, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1, 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 保持為 1 (繼續計數使能), 計時器計數器繼續計數, 而不用重載計時器計數器的值 (TMRx_CNT 的值) 到計數初值. 用戶立即可以改變不同的計時器比較寄存器 (TMRx_CMP) 的值, 而不用禁止計時器計數器和重啟計時器計數器.

例如, 計時器比較寄存器 (TMRx_CMP) 被設置為 80 (計時器比較寄存器 (TMRx_CMP) 的值必須大於 1, 且小於 2^{24}). 一旦計時器計數器的值 (TMRx_CNT 的值) 達到 80, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會被設置為 1, 且 CNTEN (TMRx_CTL[0] 計時器計數器使能位) 保持為 1 (繼續計數使能), 計時器計數器的值 (TMRx_CNT 的值) 將不會恢復到 0, 而是繼續計數, 81, 82, 83…直到 (2^{24}-1)，然後再一次從0 開始計數 0, 1, 2, 3… 到 2^{24}-1, 如此往復. 接下來, 如果用戶程式設計計時器比較寄存器 (TMRx_CMP) 的值為 200, 且 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 被清除為 0, 當計時器比較寄存器的值 (TMRx_CNT 的值) 達到 200, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會再一次被設置為 1. 最後, 用戶程式設計計時器比較寄存器 (TMRx_CMP) 的值為 500, 並清除 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 為 0, 當計時器計數器的值 (TMRx_CNT 的值) 達到 500, CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 將會再一次被設置為 1. 在這種模式下, 計時器計數器的值(TMRx_CNT 的值) 總是保持持續向上計數, 即便 CNTIF (TMRx_INTSTS[0] 計時器中斷狀態) 為 1. 所以, 本操作模式叫做連續計數模式.

下圖即為一個連續計數模式的使用範例.



8.4.9 事件計數模式

計時器控制器也提供這樣的應用，它能對輸入事件(來自腳位TMx x=0~5)計數並將事件的次數反應到CNT的值，稱為事件計數功能。要使用該功能，EXTCNTEN(TMRx_CTL[12])設定為1。

軟體可以通過CNTDBEN(TMRx_CTL[12])來使能或關閉TMx腳位消抖電路，並通過設置CNTPHASE(TMRx_CTL[13])來選擇邊沿檢測TMx腳位的相位。

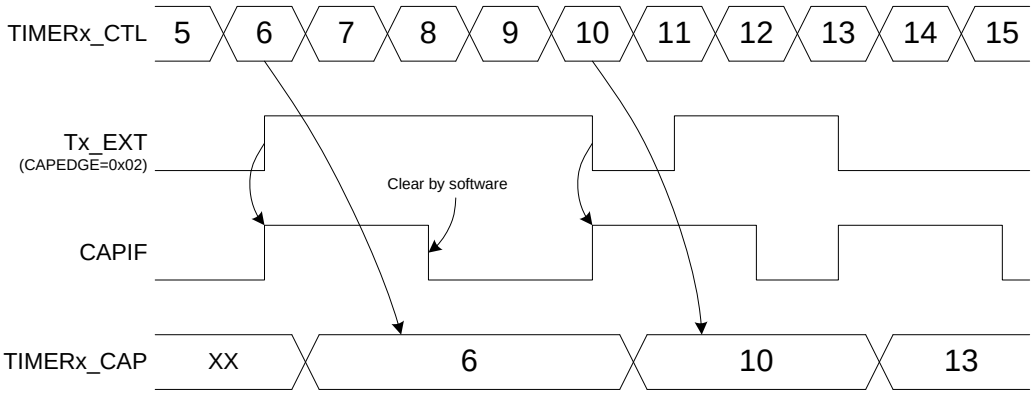
在事件計數模式，計時器計數操作模式可以設置為單次、週期和連續計數模式來計算來自TMx腳位的輸入事件CNT的值。

8.4.10 自由計數模式

在本模式下，計時器會監視外部引腳上的電平翻轉，來保存24位元計數器的值。

如果 CAPFUNCS (TMRx_CTL[17]) 為 0, 且 CAPCNTMD (TMRX_CTL[20])為 0, 則計時器計數器工作在自由計數模式. 24 位元上數型計數器將會保持連續不停的計數, 當外部引腳上的電平翻轉與在 CAPEDGE (TMRX_CTL[19:18]) 位中設定的翻轉條件相匹配時, 24 位上數型計數器的值會被保存到 TMRx_CAP 寄存器. 若是 CAPIEN (TMRx_INTEN[1]) 為 1, 在此同時 CAPIF(TMRx_INTSTS [1]) 會置 1 並觸發中斷.

在自由計數模式下, 當CAPEDGE 為 0, TMx_CAP 引腳上的下降緣 (從 1 到 0 翻轉) 是有效邊緣. 當CAPEDGE 為 1, TMx_CAP 引腳上的上升緣 (從 0 到 1 翻轉) 是有效邊緣. 若CAPEDGE 為 2 或 3, TMx_CAP 引腳上的下降緣 (從 1 到 0 翻轉) 以及上升緣 (從 0 到 1 翻轉) 都是有效邊緣. 下圖是自由計數模式的範例.

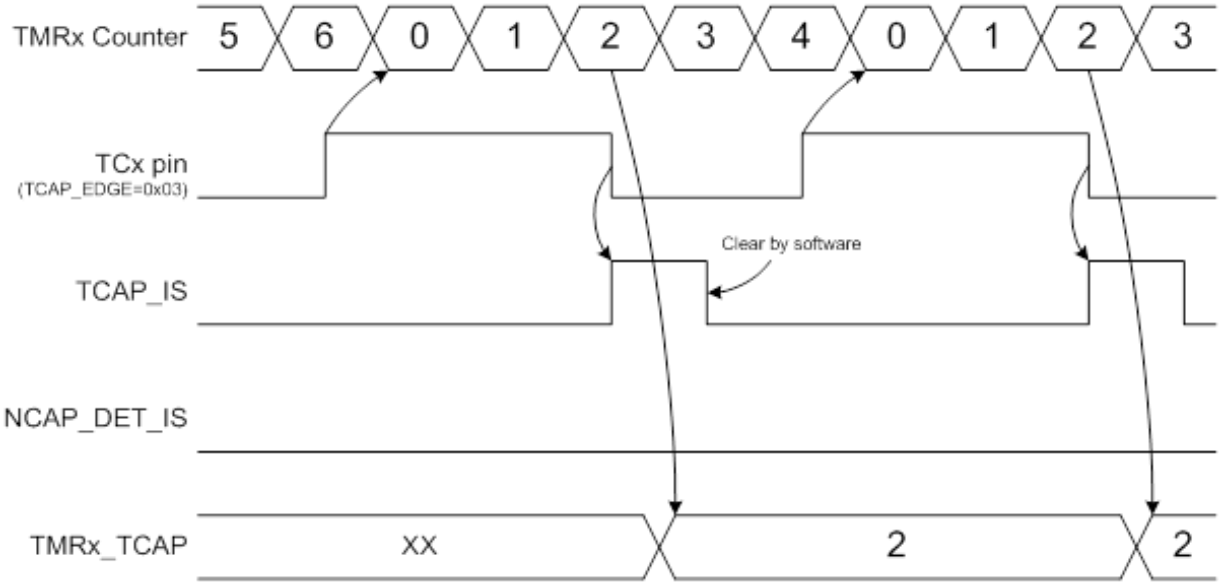


8.4.11 觸發計數模式

在本模式下, 計時器會監視外部引腳上的電平翻轉, 來保存 24 位元計數器的值.

如果 CAPFUNCS (TMRx_CTL[17]) 為 0, 且 CAPCNTMD (TMRx_CTL[20]) 為 1, 計時器計數器工作在觸發計數模式. 24 位元上數型計數器的值會保持為 0, 一旦外部引腳上的電平翻轉與 CAPEDGE (TMRx_CTL[19:18]) 位域中設定的第一個翻轉條件相匹配的話, 24 位上數型計數器將會開始計數. 當外部引腳上再一次電平翻轉與在 CAPEDGE 位域中設定的第二個翻轉條件相匹配的話, 24 位上數型計數器將停止計數, 而計數器的當前值也將會被保存到 TMRx_CAP 寄存器中. 若是CAPIEN (TMRx_INTEN[1]) 為 1, 在此同時 CAPIF(TMRx_INTSTS[1]) 會置 1 並觸發中斷.

在觸發計數模式下, 當CAPEDGE 為 0, TMx_CAP 引腳上的第一個下降緣觸發 24 位元計數器開始計數, 第二個下降緣觸發 24 位元計數器停止計數. 當CAPEDGE 為 1, TMx_CAP 引腳上的第一個上升緣觸發 24 位元計數器開始計數，第二個上升緣觸發 24 位元計數器停止計數. 若CAPEDGE 為 2, TMx_CAP 引腳上的下降緣觸發 24 位元計數器開始計數, 上升緣觸發 24 位元計數器停止計數. 若CAPEDGE 為 3, TMx_CAP 引腳上的上升緣觸發 24 位元計數器開始計數, 下降緣觸發 24 位元計數器停止計數. 下圖是觸發計數模式的範例.

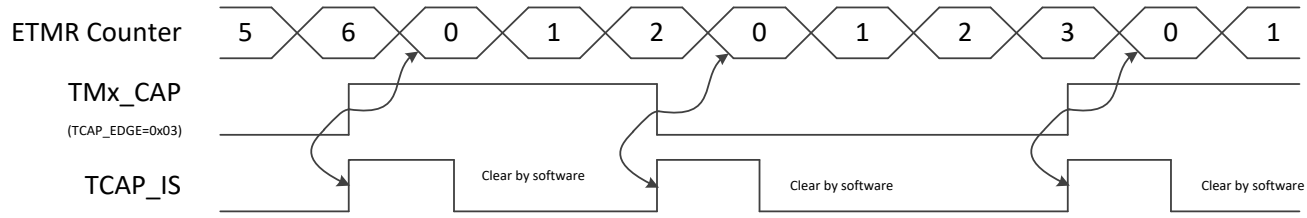


8.4.12 計數器重定模式

在本模式下, 計時器會監視外部引腳上的電平翻轉, 來復位 24 位元計數器的值. 復位前的計數器值並不會被保留下來.

如果 CAPFUNCS (TMRx_CTL[17]) 為 1, 外部引腳上的電平翻轉被用於復位計時器計數器. 在這種模式下, 一旦外部引腳上的電平翻轉與在 CAPEDGE (TMRx_CTL[19:18]) 位域中設定的條件相匹配時, 24 位上數型計數器將會被復位. 若是CAPIEN (TMRx_INTEN[1]) 為 1, 在此同時 CAPIF(TMRx_INTSTS[1]) 會置 1 並觸發中斷.

在計數器重定模式下, 當CAPEDGE 為 0, TMx_CAP 引腳上的下降緣 (從 1 到 0 翻轉) 是有效邊緣. 當CAPEDGE 為 1, TMx_CAP 引腳上的上升緣 (從 0 到 1 翻轉) 是有效邊緣. 若CAPEDGE 為 2 或 3, TMx_CAP 引腳上的下降緣 (從 1 到 0 翻轉) 以及上升緣 (從 0 到 1 翻轉) 都是有效邊緣. 下圖是計數器重定模式的範例.



8.4.13 捕獲模式去抖動

為了檢測外部引腳的電平翻轉, 計時器電路對外部引腳執行去抖操作. 基於去抖電路的結果, 外部引腳上的上升沿或下降沿可以被檢測到. 去抖電路的重定值為 0, 且僅當 CAPDBEN (TMRx_CTL[22]) 和 CAPEN (TMRx_CTL[16]) 都被使能的情況下, 去抖功能才是有效的. 所以, 如果外部引腳電平為 1, 且 CAPEDGE (TMRx_CTL[19:18]) 被設置為檢測外部引腳的上升沿, 然後設置去抖電路有效 (CAPDBEN 和 CAPEN 均設置為 1), 一個偽上升沿將會被檢測到, 這會導致在第一次 CAPIF(TMRx_INTSTS[1]) 被置位元時, 捕捉的資料不正確 (TMRx_CAP), 為了避免這種不正確的捕捉資料影響捕捉應用, 建議忽略第一次捕捉到的資料. 以免使用了不正確的捕獲值做運算.

8.4.14 定時器互觸發模式

計時器控制器提供計時器互觸發功能來精確測量頻率。內部計時器功能下，TMR0可以觸發TMR1、TMR2可以觸發TMR3及TMR4可以觸發TMR5。

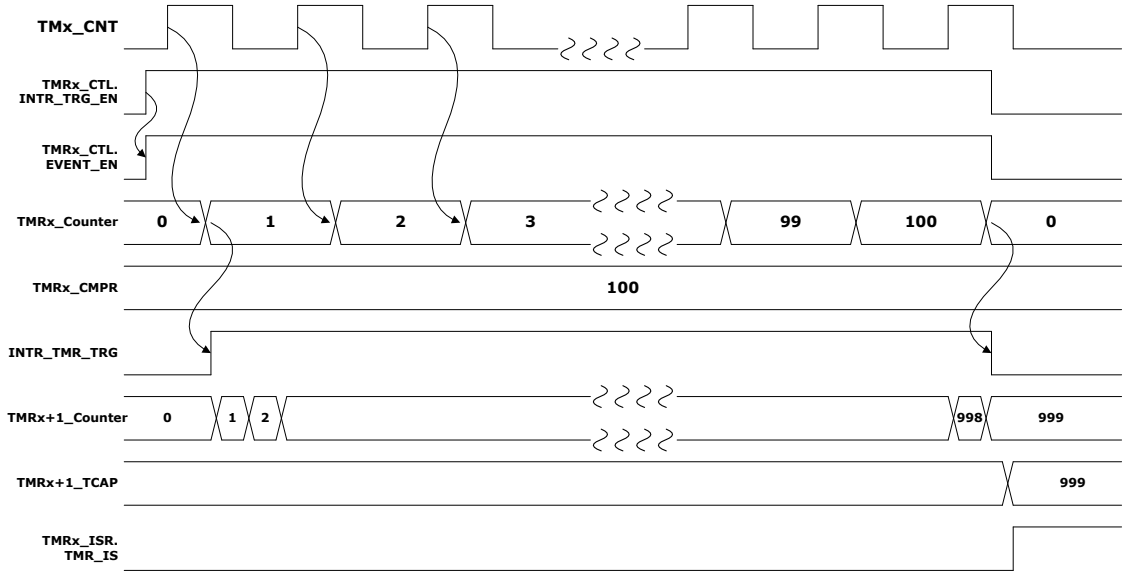
如果配置了TMR0和TMR1為計時器互觸發模式 (INTRTGEN (TMR0_CTL[24]) 為 1)，TMR0工作在事件捕捉模式來計數TM0_CNT腳位上的輸入事件並產生內部信號(INTR_TMR_TRG) 到TMR1。在檢測到TM0_CNT腳位上的第一個輸入事件時，TMR0翻轉內部信號INTR_TMR_TRG從低到高。接著，如果CNT (TMR0_CNT[23:0])到達CMPDAT (TMR0_CMP[23:0])的值， TMR0再翻轉內部信號(INTR_TMR_TRG)由高到低。TMR1則工作在外部捕捉觸發模式，在檢測到INTR_TMR_TRG的上升沿時，計時器開始向上計數；在檢測到INTR_TMR_TRG的下降沿時，載入CNT (TMR1_CNT[23:0])的值到CAPDAT (TMR1_CAP[23:0])。

如果INTRTGEN (TMR2_CTL[24])為1，TMR2和TMR3處於計時器互觸發模式。此時，TMR2和TMR3的行為與上面TMR0和TMR1的行為一致。如果INTRTGEN (TMR4_CTL[24])為1，TMR4和TMR5處於計時器互觸發模式。此時，TMR4和TMR5的行為與上面TMR0和TMR1的行為一致。

下圖展示計時器互觸發功能下TMR0和TMR0的工作模式。計時器互觸發功能結束時， CNTIF (TMR0_INTSTS[0])會置1，INTRTGEN (TMR0_CTL[24])會自動清0。與此同時，如果CNTIEN (TMR0_INTEN[0])為1，會產生計時器中斷並送到AIC通知CPU。

使用計時器互觸發功能可以精確地測量管腳TM0_CNT的輸入事件頻率。如圖 6.7-8，當TMR0計數了100個輸入事件，TMR1計數到999。如果TMR1的時鐘頻率為10MHz，即100個事件的時長

是99900ns。因此，一個輸入事件的時長是999ns，則輸入事件的頻率是1.001MHz。



8.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
TMR Base Address: **TMR0_BA = 0xB005_0000** **TMR1_BA = 0xB005_0100** **TMR2_BA = 0xB005_1000** **TMR3_BA = 0xB005_1100** **TMR4_BA = 0xB005_2000** **TMR5_BA = 0xB005_2100**				
TMRn_CTL **n = 0,1,2,3,4,5**	TMRn_BA+0x000	R/W	Enhance Timer n Control and Status Register	0x0000_0000
TMRn _PRECNT **n = 0,1,2,3,4,5**	TMRn_BA+0x004	R/W	Enhance Timer n Pre-Scale Counter Register	0x0000_0000
TMRn _CMP **n = 0,1,2,3,4,5**	TMRn_BA+0x008	R/W	Enhance Timer n Compare Register	0x0000_0000
TMRn _INTEN **n = 0,1,2,3,4,5**	TMRn_BA+0x00C	R/W	Enhance Timer n Interrupt Enable Register	0x0000_0000
TMRn _INTSTS **n = 0,1,2,3,4,5**	TMRn_BA +0x010	R/W	Enhance Timer n Interrupt Status Register	0x0000_0000
TMRn _CNT **n = 0,1,2,3,4,5**	TMRn_BA+0x014	R/W	Enhance Timer n Counter Data Register	0x0000_0000
TMRn _CAP **n = 0,1,2,3,4,5**	TMRn_BA+0x018	R	Enhance Timer n Capture Data Register	0x0000_0000
TMRn _ECTL **n = 0,1,2,3,4,5**	TMRn_BA+0x020	R/W	Enhance Timer n Extended Control Register	0x0000_0000

9 脈寬調製 (PWM)

9.1 概述

NUC980 有 兩組 PWM，分別為 PWM0 和 PWM1，每個 PWM 有4 個獨立的 PWM 輸出, CH0~CH3, 或者作為兩個帶有可編程死區發生器的互補的PWM對, (CH0, CH1), (CH2, CH3).

每兩個PWM輸出, (CH0, CH1), (CH2, CH3), 共用同一個8-位預分頻, 同一個提供5級分頻 (1, 1/2, 1/4, 1/8, 1/16) 的時鐘分頻器. 每個 PWM 輸出有一個用於 PWM 週期控制的獨立的 16 位 PWM 下數型計數器, 和一個用於 PWM 占空比控制的 16 位比較器. 每一個死區發生器有兩個輸出, 第一個死區發生器的輸出是 CH0 和 CH1, 而第二個死區發生器的輸出是 CH2 和 CH3. PWM 控制器一共提供 4 個獨立的PWM中斷標誌, 當某一個 PWM 週期下數計數器計數到 0 時, 相應的中斷標誌由硬體置位元. 當 PWM 中斷源和相應的中斷使能位有效時, PWM 中斷將會被觸發. 每一個 PWM 輸出都可以被配置為單次模式來產生僅一個 PWM 週期的信號, 或者連續模式來連續輸出 PWM 波形.

當 DZEN01 (PWM_PCR[4]) 位被設置為 1 時, CH0 與 CH1 執行互補的 PWM 對功能, 這一對 PWM 的時序, 週期, 占空比, 和死區時間由 PWM 通道 0 計時器和死區發生器 0 決定. 同樣, 當 DZEN23 (PWM_PCR[5]) 位被設置為 1 時, 互補的 PWM 對 (CH2, CH3) 由 PWM 通道 2控制.

為防止 PWM 驅動輸出引腳輸出不穩定波形, 16 位週期下數型計數器和 16 位比較器均採用雙緩存, 當用戶向計數器/比較器緩存寄存器內寫入值時, 只有當下數型計數器的值計數到 0 時, 更新的值才會被重新載入下數型計數器/比較器. 該雙緩存特性可以避免 PWM 輸出時產生干擾波形.

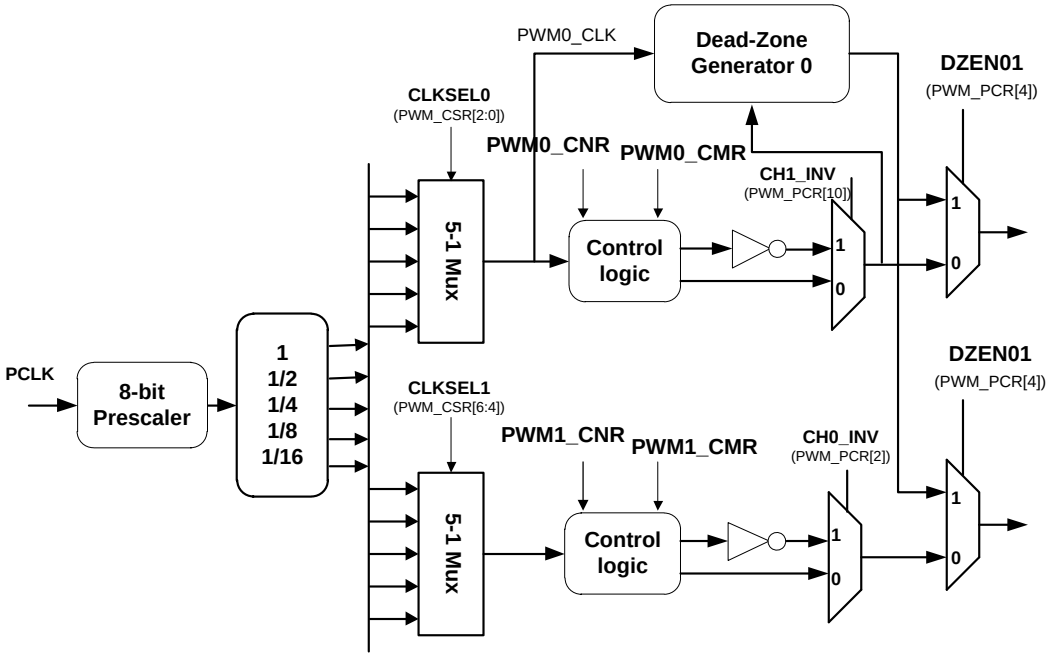
當 16 位下數型計數器達到 0 時, 中斷要求產生. 如果 PWM 輸出被設置為連續模式, 當下數計數器計數到 0 時, 下數計數器會重複自動重新裝載 PWMx_CNR 寄存器中 CNR 的值, 並開始減計數. 如果 PWM 輸出被設置為單次模式, 當下數計數器計數到 0 時, 停止計數, 並產生一個插斷要求.

PWM 比較器用於脈衝寬度調製, 當下數計數器的值與比較寄存器的值匹配時, 計數器控制邏輯會改變 PWM 輸出電平.

9.2 特性

- 4 個獨立的PWM 輸出, 每個通道均帶有中斷
- 互補的PWM 對, (CH0, CH1) 及 (CH2, CH3), 支援程式設計死區發生器
- 每對 PWM 內部帶有 8 位預分頻, 以及除頻器
- 高達 16 位的 PWM計數器以及比較器寬度
- 每個通道均可設置獨立的時鐘源
- 支援單次或連續模式

9.3 方塊圖



9.4 功能描述

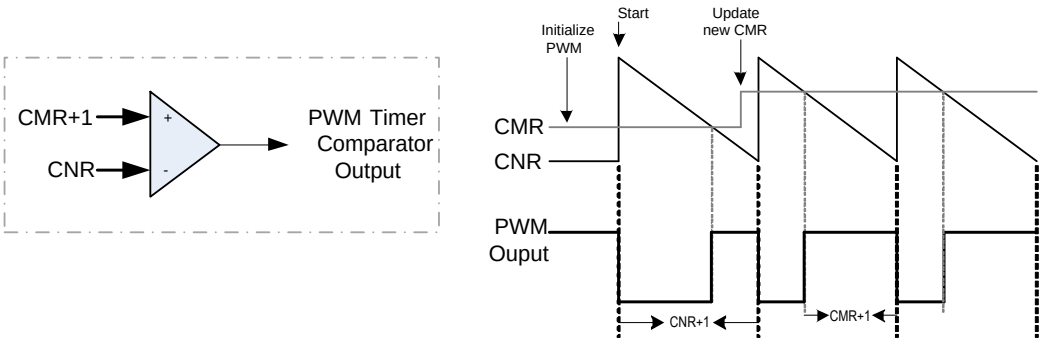
9.4.1 PWM 計時器操作

PWM 週期和占空比控制由 PWMx_CNR 寄存器和 PWMx_CMR 寄存器決定. PWM 計時器工作時序如下圖. 脈寬調製的公式如下:

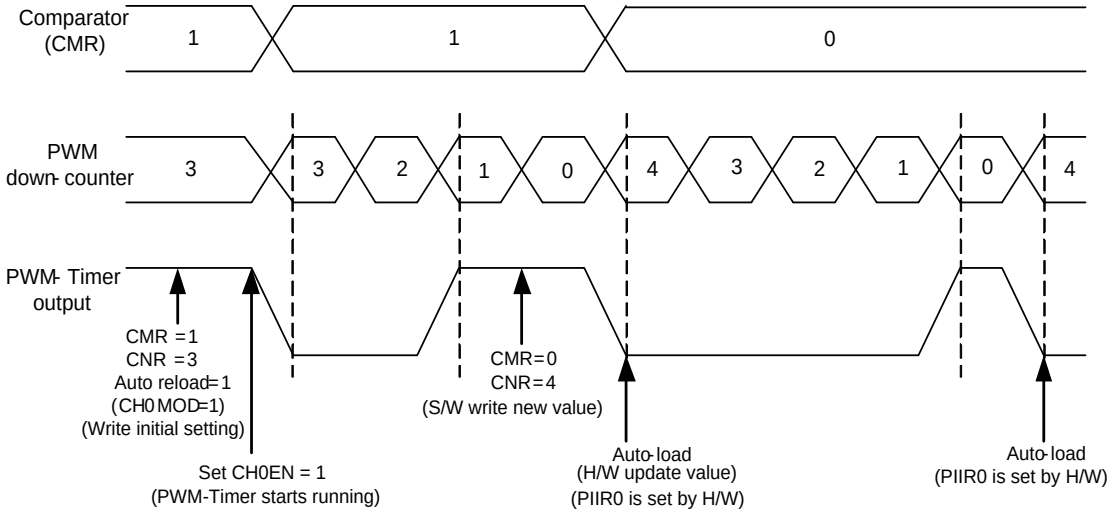
PWM 頻率 = PWM_CLK / ((prescale + 1) * (clock divider)) / (CNR + 1)

PWM 占空比 = (CMR + 1) / (CNR + 1)

由輸出波形來看, 當 CMR >= CNR: PWM 輸出總是為高. 當 CMR < CNR: PWM 低脈寬 = (CNR - CMR) 個 PWM 時鐘, PWM 高脈寬 = (CMR + 1) 個 PWM 時鐘. 而當 CMR = 0: PWM 低脈寬 = CNR個 PWM 時鐘; PWM 高脈寬 = 1個 PWM 時鐘.

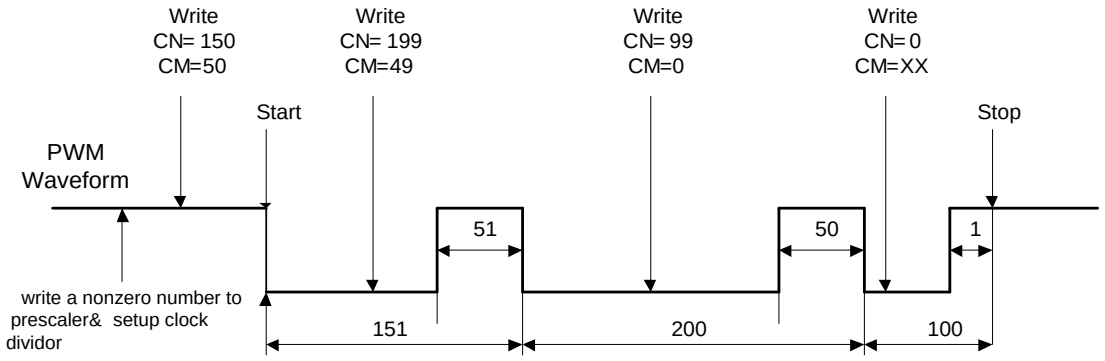


下圖顯示了 PWM計時器的比較器運作原理. 當下數計數器的值與比較寄存器的值匹配時，計數器控制邏輯會改變 PWM 輸出電平.

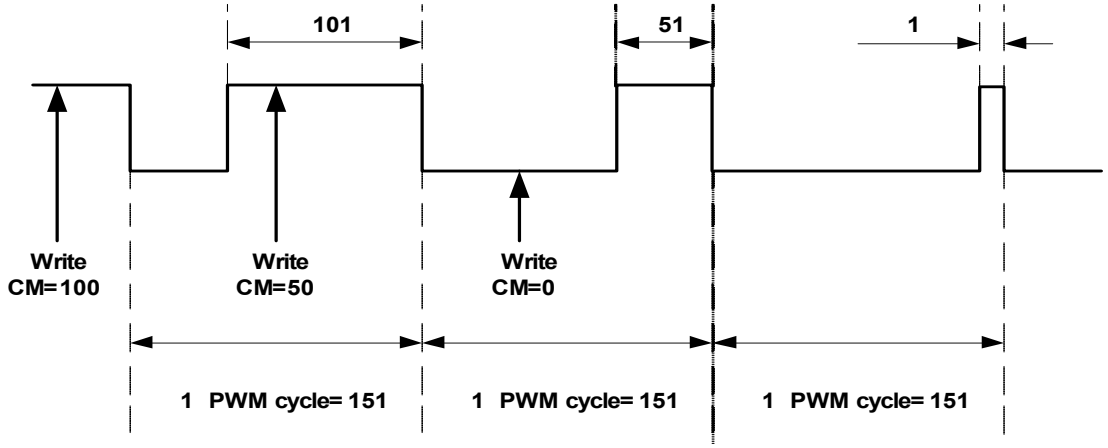


9.4.2 PWM 雙緩存功能

PWM 具有雙緩存功能, 重載值只在下一個週期開始時被更新, 而不會影響當前計時器工作. PWM 計數器的值是在 PWM_CNR 的 [15:0] 位.



雙緩存功能也允許比較器在當前週期的任意時刻被寫入, 寫入值會在下一個週期生效. PWM 比較器的值是在 PWM_CMR 的 [15:0] 位.



Modulate PWM controller ouput duty ratio (CN = 150)

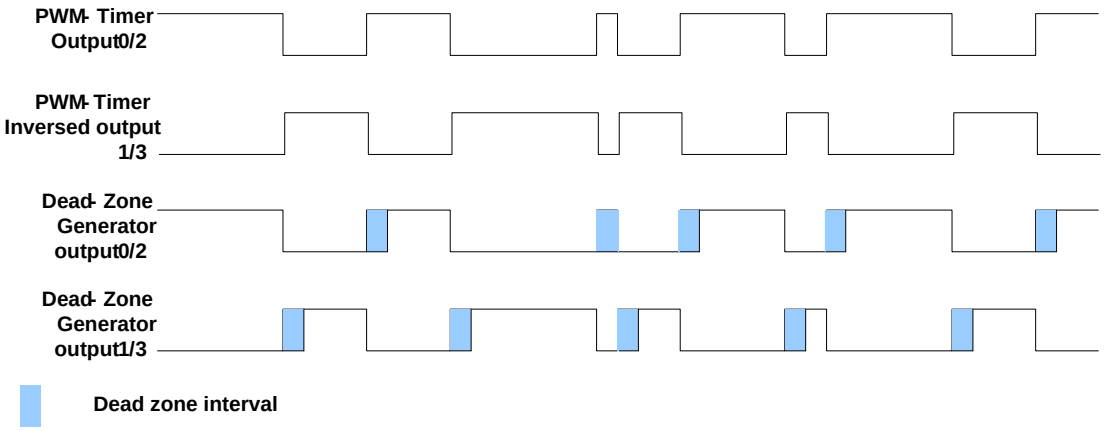
9.4.3 連續以及單次操作

PWM 控制寄存器 (PWM_PCR) 中 CHxMOD 位定義 PWMx (x = 0~3) 是連續模式或是單次模式。舉例來說, 如果 CH0MOD (PWM_PCR[3]) 被設為 1, 則 PWM 0 在連續模式下工作, 當 PWM 計數器計數到 0, PWM 控制器會載入CNR 的值到 PWM 計數器, 並繼續計數. 但如果 CN 被設為 0, PWM 計數器在計數到 0 後, 將停止運行.

在單次模式下 (CH0MOD=0), 相應的通道將僅僅輸出一次占空波形, 並且如果沒有進一步的相應占空寄存器的更新, 計數器將停止. 當 PWM 計數器正在運行時, 更新相應的占空寄存器將會進行下一次的占空波形輸出.

9.4.4 死區發生器

PWM 控制器提供死區發生器, 用於功率器件保護. 該功能在 PWM 上升沿輸出時產生可程式設計的時隙來延遲 PWM 上升沿輸出. 用戶可通過程式設計死區計數器來確定死區間隔. 下圖是死區的演示範例.



9.4.5 PWM 計時器開啟過程

以下以設置 PWM 通道0 當例子, 說明啟動 PWM 的步驟.

1. 設置時鐘選擇器 CLKSEL0 (PWM_CSR[2:0])
2. 設置預分頻器 PRESCALE (PWM_PPR[7:0])
3. 設置CH0INV (PWM_PCR[2]) , 控制輸出反轉打開或是關閉關閉
4. 設置 DZEN01 (PWM_PCR[4]) 控制死區發生器打開/關閉, 若是死區功能開啟, 設置死區間隔 DZL01 (PWM_PPR[23:16])
5. 設置CH0MOD (PWM_PCR[3]), 選擇工作模式是自動重載或是單次模式.
6. 設置中斷使能位 PIER0 (PWM_PIER[0])
7. 設置相應管腳為 PWM 功能
8. 將 CH0EN((PWM_PCR[0])) 置 1. 使能 PWM 下數型計數器開始運行.
9. 設置 PWMx_CMR寄存器的和 PWMx_CNR 寄存器位域來設定 PWM 占空比

上述步驟中的 1~8 可以不按照上述的順序設置，這對 PWM 計時器的正常工作沒有影響.

以下是設置 PWM0 輸出 1000Hz 週期, 佔空比 40% 的範例程式

```
// Assume PWM clock source, PCLK, is 75 MHz.
PWM->PPR = 74;        // so now PWM clock is 75MHz / (74 + 1) = 1MHz
PWM->CSR = 4;     // Prescale output divide by 1
PWM->PCR = 9;     // Enable PWM0 in periodic mode
```

```
// 1M / 1000 = 1000
// 1000 * 40% = 400
PWM->CMR = (400 - 1);
PWM->CNR = (1000 - 1);
```

9.4.6 PWM 計時器停止過程

有兩種方式可以停止 PWM 計時器, 以通道 0 為例說明:

方式 1:

設置 PWMx_CNR 寄存器為 0，並等待 PWM 超時中斷 (如果PIER0 (PWM_PIER[0]) 被置位) 發生, 或者輪詢相應的超時標誌 PIIR0 (PWM_PIIR[0]). 當 PWM 超時中斷發生, 或者超時標誌被置位元, 將 CH0EN((PWM_PCR[0])) 清為 0. (推薦)

方式 2:

直接將 CH0EN((PWM_PCR[0])) 清為 0 . (不推薦)

不推薦方式 2 是因為禁止 CH0EN 將立即停止 PWM 輸出信號, 會導致 PWM 輸出的占空比改變, 這可能引起電機控制電路的損壞.

9.5 寄存器

Register	Offset	R/W	Description	Reset Value
PWM Base Address: **PWM0_BA = 0xB005_8000** **PWM1_BA = 0xB005_9000**				
PWM_PPR	PWM_BA+0x000	R/W	PWM Pre-scale Register	0000_0000
PWM_CSR	PWM_BA+0x004	R/W	PWM Clock Select Register	0000_0000
PWM_PCR	PWM_BA+0x008	R/W	PWM Control Register	0000_0000
PWM0_CNR	PWM_BA+0x00C	R/W	PWM Counter Register 0	0000_0000
PWM0_CMR	PWM_BA+0x010	R/W	PWM Comparator Register 0	0000_0000
PWM0_PDR	PWM_BA+0x014	R	PWM Data Register 0	0000_0000
PWM1_CNR	PWM_BA+0x018	R/W	PWM Counter Register 1	0000_0000
PWM1_CMR	PWM_BA+0x01C	R/W	PWM Comparator Register 1	0000_0000
PWM1_PDR	PWM_BA+0x020	R	PWM Data Register 1	0000_0000
PWM2_CNR	PWM_BA+0x024	R/W	PWM Counter Register 2	0000_0000
PWM2_CMR	PWM_BA+0x028	R/W	PWM Comparator Register 2	0000_0000
PWM2_PDR	PWM_BA+0x02C	R	PWM Data Register 2	0000_0000

PWM3_CNR	PWM_BA+0x030	R/W	PWM Counter Register 3	0000_0000
PWM3_CMR	PWM_BA+0x034	R/W	PWM Comparator Register 3	0000_0000
PWM3_PDR	PWM_BA+0x038	R	PWM Data Register 3	0000_0000
PWM_PIER	PWM_BA+0x03C	R/W	PWM Timer Interrupt Enable Register	0000_0000
PWM_PIIR	PWM_BA+0x040	R/W	PWM Timer Interrupt Indication Register	0000_0000

10 看門狗計時器控制器 (WDT)

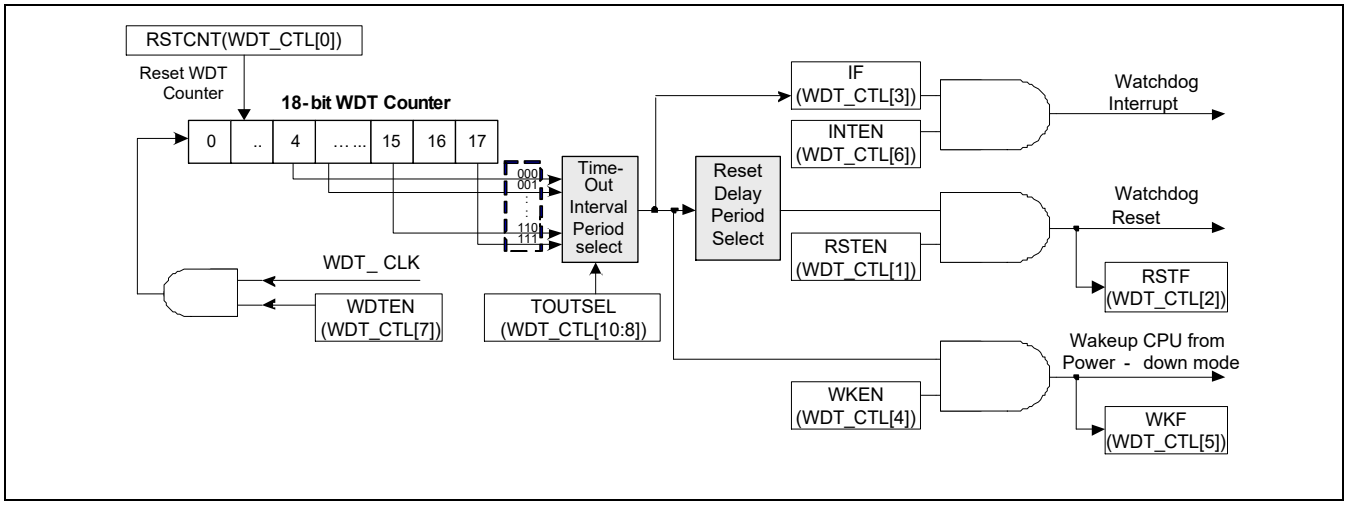
10.1 概述

看門狗計時器的用途是在軟體出問題時執行系統重定功能，這可以防止系統無限期地掛起. 除此之外, 看門狗計時器還支援將 CPU 從休眠模式喚醒的功能. 看門狗計時器包含一個 18 位的自由運行計數器, 定時溢出間隔可程式設計.

10.2 特性

- 18-位自由運行 WDT 計數器用於看門狗計時器超時間
- 可選擇的超時間隔 (2^4 ~ 2^{18}), 如果 WDT_CLK = 32.768kHz, 則超時間隔為 0.49 mS ~ 8S.
- 復位週期 = (1 / WDT_CLK) * 63
- 復位延時可調整為 1026, 130, 18 或 3 個 WDT_CLK
- 支持由開機設置使能 WDT
- 當 WDT 時鐘源選擇 32.768 kHz 晶振時, 可將 CPU 從休眠模式中喚醒

10.3 方塊圖



10.4 功能描述

10.4.1 WDT 設置

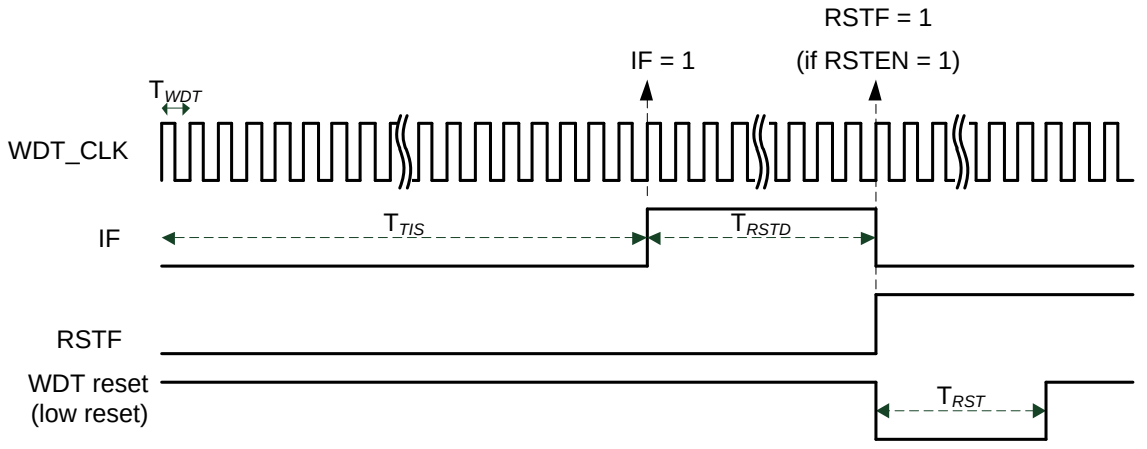
看門狗計時器的用途是在軟體出問題時執行系統復位功能，這可以防止系統無限期地掛起. 除此之外, 看門狗計時器還支援將 CPU 從掉電模式喚醒的功能, 此外當 CPU 進入掉電模式時, WDT 計數器會自動重置. 看門狗計時器包含一個 18 位的自由運行計數器, 定時溢出間隔可程式設計.

接下來的表格給出了看門狗超時間隔選擇, 表格內的 T_{WDT} 依選擇的時鐘源而定, 透過 WDT_S(CLK_DIVCTL8[9:8])可設置為 HXT (12MHz 晶振), 12MHz/512, PCLK/4096, 或是 LXT (32KHz 晶振).

TOUTSEL (WDT_CTL[10:8])	超時週期	WDT_CLK=HXT 超時間隔	WDT_CLK=LXT超時間隔
000	$2^4 * T_{WDT}$	1.33 uS	488.28 uS
001	$2^6 * T_{WDT}$	5.33 uS	1.95 mS
010	$2^8 * T_{WDT}$	21.3 uS	7.81 mS
011	$2^{10} * T_{WDT}$	85.3 uS	31.25 mS
100	$2^{12} * T_{WDT}$	341.3 uS	125 mS
101	$2^{14} * T_{WDT}$	1.37 mS	0.5 S
110	$2^{16} * T_{WDT}$	5.46 mS	2.0 S
111	$2^{18} * T_{WDT}$	21.8 mS	8.0 S

設置 WDTEN (WDT_CTL[7]) 使能看門狗計時器和 WDT 計數器開始計數. 當計數器達到選擇的超時間隔, 看門狗計時器中斷標誌 IF (WDT_CTL[3]) 將被立即被置位, 如果看門狗計時器中斷使能位 INTEN(WDT_CTL[6]) 置位, 則會並請求 WDT 中斷, 同時緊接著超時事件會有一個指定的延時可透過 RSTDSEL (WDT_ALTCTL[1:0]) 設置, 用戶必須在該指定延時過期前設置 RSTCNT (WDT_CTL[0]) (看門狗計時器復位) 為高, 或是將 WDT_RSTCNT 寫入 0x00005AA5 來復位18 位 WDT 計數器, 以防止晶片復位. RSTCNT 位元在 WDT 計數器重定後由硬體自動清零. 通過設置 TOUTSEL (WDT_CTL[10:8]), 有 8 種帶指定延時時間的超時間隔可供選擇. 如果 RSTEN (WDT_CTL[1]) 為 1, 且在指定延遲時間過期後, WDT 計數器沒有被清零, 看門狗計時器將置位元看門狗計時器重定標誌 WDTRSTS (SYS_RSTSTS[5]), 並復位 CPU. 這個重定將持續 63 個 WDT 時鐘 (T_{RST}), 然後晶片重啟. WDTRSTS 不會被看門狗復位清零. 使用者可用軟體輪詢 WDTRSTS 來識別復位源是否 WDT.

接下來的圖片給出了看門狗中斷信號和重定信號的時序.



- T_{WDT} : *Watchdog Clock Time Period*
- T_{TIS} : *Watchdog Time-out Interval Period ((2^4 ~ 2^{18}) * T_{WDT})*
- T_{RSTD} : *Watchdog Reset Delay Period*
  *- Selectable 3/18/130/1026 * T_{WDT} delay period controlled by RSTDSEL(WDT_ALTCTL [1:0])*
- T_{RST} : *Watchdog Reset Period (63 * T_{WDT})*

注意: 若是開機設置沒有使能 WDT, 而是上電之後由軟件使能, WDT 將無法將系統復位.

10.4.2 WDT 喚醒功能

若 WDT 時鐘源設置為 LXT, 則 WDT 可觸發計時器超時中斷, 將系統自休眠模式中喚醒. 若要使能喚醒功能, INTEN (WDT_CTL[6]), WKEN (WDT_CTL[4]) 以及 WDT(SYS_WKUPSER[28]) 在 WDT 啟動時, 均須置 1. 超時中斷喚醒系統後, WKF (WDT_CTL[5]) 以及 WDT (SYS_WKUPSSR[28]) 均會被置 1. 使用者可用軟體輪詢這兩位來識別喚醒源是否 WDT. 軟體可以將這兩位寫 1, 清除標誌位.

10.5 寄存器

Register	Offset	R/W	Description	Reset Value
WDT Base Address: **WDT_BA = 0xB004_0000**				
WDT_CTL	WDT_BA+0x00	R/W	WDT Control Register	0x0000_0700
WDT_ALTCTL	WDT_BA+0x04	R/W	WDT Alternative Control Register	0x0000_0000
WDT_RSTCNT	WDT_BA+0x08	W	WDT Reset Counter Register	0x0000_0000

11 窗口看門狗定時控制器 (WWDT)

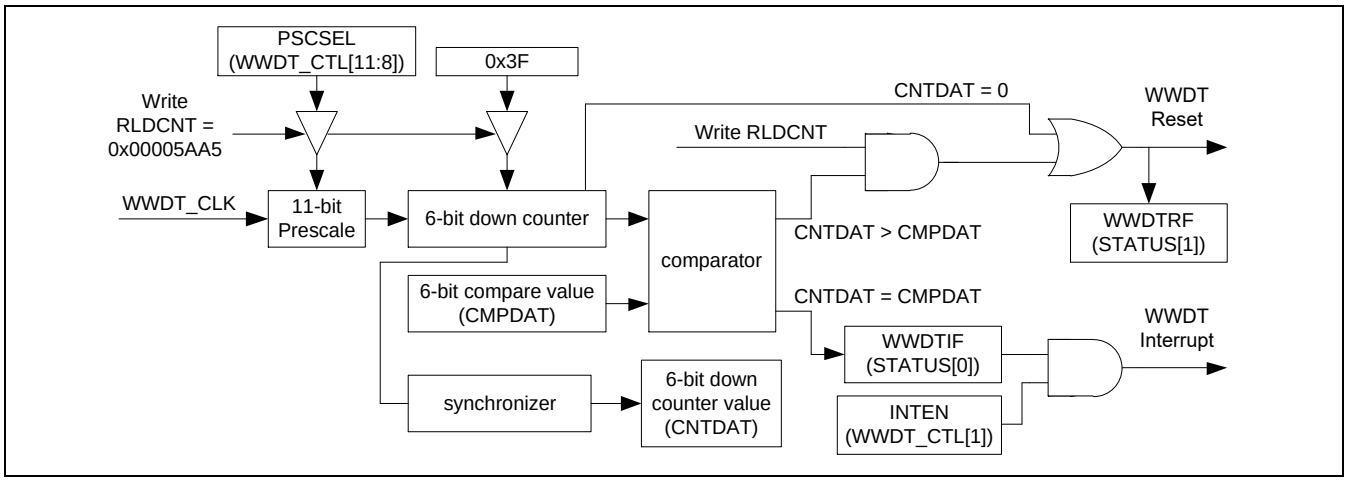
11.1 概述

窗口看門狗計時器的目的是在一個指定的視窗週期中執行系統重定, 防止軟體在任何不可預知的條件下進入不可控制的狀態.

11.2 特性

- 一個6-bit 下數計數器和一個6-bit比較值使視窗週期可調
- 可選的 WWDT 時鐘預分頻計數器使WWDT溢出間隔可變

11.3 方塊圖



11.4 功能描述

11.4.1 超時設置

視窗看門狗計時器包括一個6-bit 下數計數器和用來定義不同超時間隔的可程式設計的預分頻器。

6-bit 視窗看門狗計時器的時鐘源可透過 WWDT_S(CLK_DIVCTL8[11:10]) 設置為 HXT (12MHz 晶振), 12MHz/512, PCLK/4096, 或是 LXT (32KHz 晶振). 另有一個可程式設計的11-bit的預分頻器。可程式設計的11-bit的預分頻器通過寄存器 PSCSEL (WWDT_CTL[11:8])來控制. PSCSEL 和預分頻器的值的關係如下表所示：

PSCSEL	預分頻值	超時週期	超時間隔 WWDT_CLK=HXT	超時間隔 WWDT_CLK=LXT
0000	1	1 * 64 * T_{WWDT}	5.33 uS	1.95 mS
0001	2	2 * 64 * T_{WWDT}	10.66 uS	3.91 mS
0010	4	4 * 64 * T_{WWDT}	21.33 uS	7.81 mS
0011	8	8 * 64 * T_{WWDT}	42.67 uS	15.63 mS
0100	16	16 * 64 * T_{WWDT}	85.33 uS	31.25 mS
0101	32	32 * 64 * T_{WWDT}	170.67 uS	62.50 mS
0110	64	64 * 64 * T_{WWDT}	341.33 uS	125.00 mS
0111	128	128 * 64 * T_{WWDT}	682.67 uS	250.00 mS
1000	192	192 * 64 * T_{WWDT}	1.02 mS	375.00 mS
1001	256	256 * 64 * T_{WWDT}	1.37 mS	500.00 mS
1010	384	384 * 64 * T_{WWDT}	2.05 mS	750.00 mS
1011	512	512 * 64 * T_{WWDT}	2.73 mS	1.00 S
1100	768	768 * 64 * T_{WWDT}	4.10 mS	1.50 S
1101	1024	1024 * 64 * T_{WWDT}	5.46 mS	2.00 S
1110	1536	1536 * 64 * T_{WWDT}	8.19 mS	3.00 S
1111	2048	2048 * 64 * T_{WWDT}	10.09 mS	4.00 S

視窗看門狗計時器可以通過設置WWDTEN (WWDT_CTL[0]) 為1使能. 當視窗看門狗計時器使能, 下數計數器開始從0x3F下數, 且不能被軟件停止. 當前的計數器值可透過讀取 WWDT_CNT 寄存器來獲得.

如果WWDT計數器值到0, WWDT發生復位. 在WWDT下數到0之前, 軟體可以寫特定值 0x00005AA5 到寄存器WWDTRLD來重新載入 0x3F 到 WWDT 計數器防止 WWDT 復位發生. 這個重新載入的動作僅在 WWDT 計數器值小於或等於 WINCMP 時有效。如果軟體在WWDT計數器大於WINCMP期間寫WWDTRLD, 會導致晶片復位.

為防止程式跑到非預期代碼禁止視窗看門狗，控制寄存器WWDTCR 和 WWDT_IER在晶片上電或重定之後僅能寫一次. 一旦視窗看門狗被軟體使能, 軟體不能禁止視窗看門狗, 改變預分頻週期或改變視窗比較值, 除非晶片重定.

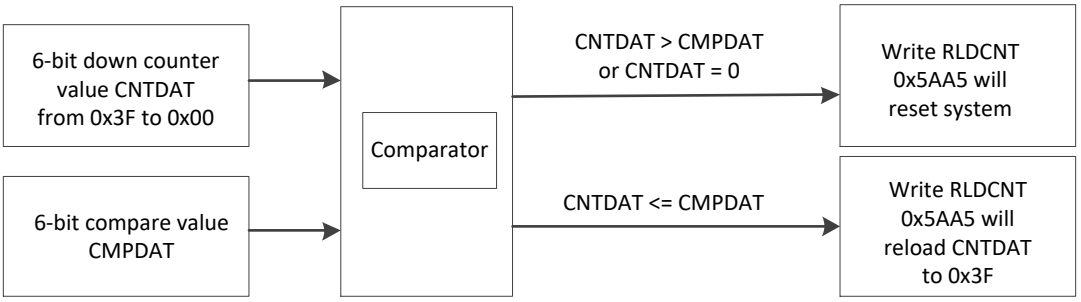
11.4.2 WWDT 中斷

在 WWDT 下數期間, 如果計數器值等於視窗看門狗計時器比較值 CMPDAT

(WWDT_CTL[21:16]) 且 INTEN (WWDT_CTL[1]) 置 1, 會發生中斷. 且 WWDTIF (WWDT_STATUS[0]) 會被置 1. 軟體可以寫 1 清除此標誌位.

11.4.3 系統復位

當WWDTIF (WWDT_STATUS[0]) 會置 1 後, 軟體必須在計數器下數到 0 之前寫特定值 0x00005AA5 到寄存器 WWDTRLD 來防止系統復位. 但若是軟體在當前計數器的值仍然大於 CMPDAT 之前就來填寫 0x00005AA5 進 WWDTRLD, 會馬上造成系統復位. 軟件可透過讀取 WWDTRF (WWDT_STATUS[1]) 是否被置 1 來判斷是否 WWDT 造成系統復位. 軟體可以寫 1 清除此標誌位.



11.4.4 使用限制

當軟體寫特定值 0x00005AA5 到寄存器 WWDTRLD重載WWDT計數器, 需要3個視窗看門狗時鐘來同步重載命令實現真正執行重載操作. 這意味著如果軟體設置視窗時鐘分頻器為1, 比較值 WINCMP (WWDTCR[21:16])應該大於2或軟體不能在WWDT重定之前重載WWDT計數器. 下表列出預分頻以及比較器值配合上的限制:

PSCSEL (WWDT_CTL[11:8])	預分頻	可用的 CMPDAT (WWDT_CTL[21:16])值
0000	1	0x3 ~ 0x3F
0001	2	0x2 ~ 0x3F
Others	Others	0x0 ~ 0x3F

另外, WWDT 在系統進入休眠模式後, 會暫停計數, 所以無法透過WWDT 中斷喚醒系統.

11.5 寄存器

Register	Offset	R/W	Description	Reset Value
WWDT Base Address: **WWDT_BA = 0xB004_0100**				
WWDT_RLDCNT	WWDT_BA+0x00	W	WWDT Reload Counter Register	0x0000_0000
WWDT_CTL	WWDT_BA+0x04	R/W	WWDT Control Register	0x003F_0800
WWDT_STATUS	WWDT_BA+0x08	R/W	WWDT Status Register	0x0000_0000
WWDT_CNT	WWDT_BA+0x0C	R	WWDT Counter Value Register	0x0000_003F

12 實時時鐘 (RTC)

12.1 概述

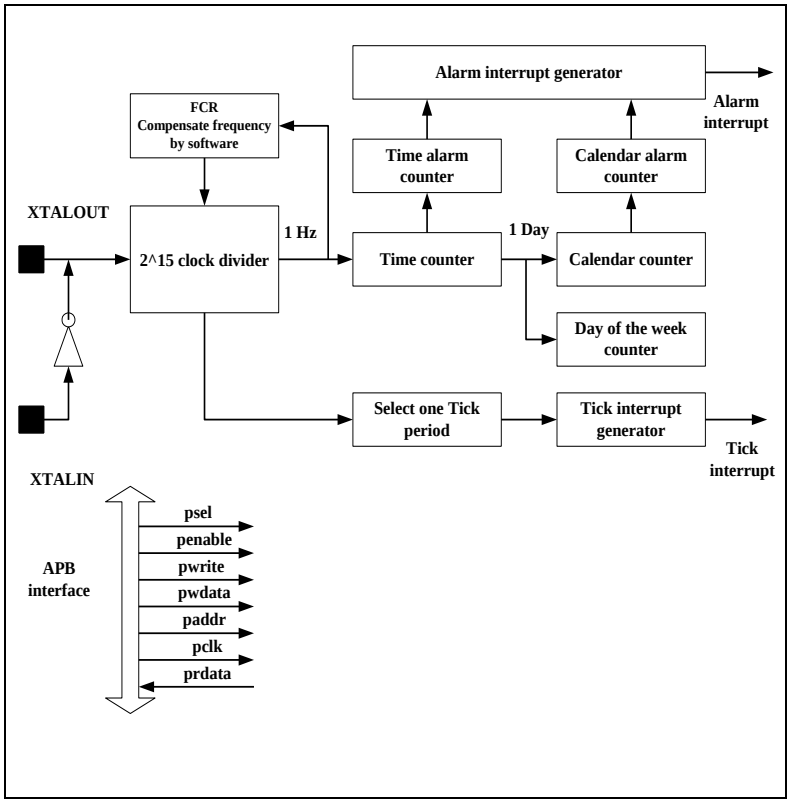
實時時鐘（RTC）模塊當系統電源關閉可以由獨立的電源進行供電。該RTC模塊採用外部晶振產生32.768 kHz時鐘。 RTC可以將數據用BCD碼傳輸到CPU。這些數據包括時間訊息(秒，分，時)和日曆訊息(日，月，年)。此外，為了達到更好的頻率精度，RTC計數器可以通過軟件進行調整。

12.2 特性

- 時間計數器(秒，分，時)和日曆計數器(日，月，年)
- 鬧鐘寄存器(秒，分，時，日，月，年)
- 12 小時或 24 小時模式可選澤
- 閏年自動識別
- 一週天數計數器
- 頻率補償寄存器(FCR)
- 所有時間日期由 BCD 碼表示
- 支持周期時間節拍中斷，提供 8 個周期選項供選擇 1/128 秒，1/64 秒，1/32 秒，1/16 秒，1/8 秒，1/4 秒，1/2 秒及 1 秒
- 支持從掉電模式下喚醒芯片
- 支持電源開/關控制機制，控制系統的電源
- 支持 64 字節的備用寄存器來存儲用戶的重要信息

12.3 方塊圖

實時時鐘(RTC)的模塊圖如下所示:



12.4 功能描述

12.4.1 初始化

當RTC模塊上電後，RTC模塊處於復位狀態，用戶需寫入(0xa5eb1357)至INIR寄存器讓RTC模塊離開復位狀態，一旦RTC_INIT寄存器被寫入(0xa5eb1357)後，將繼存器INIR的位0讀出,當INIR[0]為1時,表示RTC已被重置完成. 用戶再寫入其他值到RTC_INIT寄存器時，不會影響RTC正常運行。初始化只需要在RTC模塊第一次上電時做就可以了.

12.4.2 訪問(讀/寫)限制

RTC_RWEN寄存器的[15:0]位可控制RTC模塊部份寄存器的讀/寫功能，用于避免系統掉電時對RTC模塊的誤寫。用戶在寫入除了RTC_INIT所有寄存器之前，必須通過寫0xa965到RTC_RWEN寄存器的[15:0]位，使得RWENF被設成1時，用戶才可將數據寫入寄存器，RWENF將在一個短周期內(約24ms)保持為1，在這一個短周期(約24ms)之後，RWENF會自動由內部狀態機設成0。 用戶可以關閉 RTC clock (CLK_PCLKEN0[2]),用以降低芯片的功耗.

RTC_WEEKDAY寄存器提供一周日期，RTC_WEEKDAY寄存器的值0~6分別表示週日至週六

RTC_TALM, RTC_CALM, RTC_TIME, RTC_CAL均是使用BCD格式,但RTC_FREQADJ不使用BCD格式. 使用者必需了解，RTC模塊本身無法檢查使用者設定的時間是否合理，例如：把

RTC_CAL設定成201a(年),13(月),00(日)或是CLR設定的日期和星期幾的設定不符合.

復位狀態：

Register	Value	Description
RTC_RWEN	0	RTC 寄存器讀寫關閉
RTC_CAL	05 , 1 ,1	2005-1-1
RTC_TIME	00 00 00	00時, 00分, 00秒
RTC_CALM	00,00,00	2000-0-0
RTC_TALM	00,00,00	00時, 00分, 00秒
RTC_TIMEFMT	1	24小時模式
RTC_WEEKDAY	6	星期六
RTC_INTEN	0	時鐘節拍中斷以及計數器不使能 RTC鬧鐘中斷不使能
RTC_INTSTS	0	時鐘節拍中斷沒發生 鬧鐘中斷沒發生
RTC_LEAPYEAR	0	表示今年不是閏年
RTC_TICK	0	時鐘節拍使能

12.4.3 12/24 小時格式顯示切換

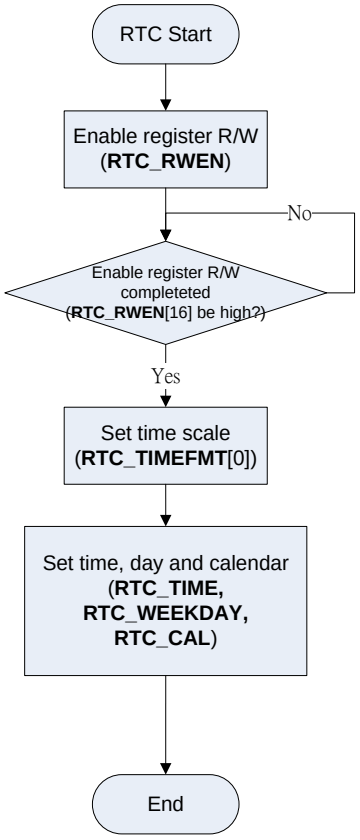
設置RTC_TIMEFMT寄存器的[0]位為0/1，用於切換顯示格式為12/24小時制.

24-小時制	12-小時制	24-小時制	12-小時制
00	12(AM12)	12	32(PM12)
01	01(AM01)	13	21(PM01)
02	02(AM02)	14	22(PM02)

03	03(AM03)	15	23(PM03)
04	04(AM04)	16	24(PM04)
05	05(AM05)	17	25(PM05)
06	06(AM06)	18	26(PM06)
07	07(AM07)	19	27(PM07)
08	08(AM08)	20	28(PM08)
09	09(AM09)	21	29(PM09)
10	10(AM10)	22	30(PM10)
11	11(AM11)	23	31(PM11)

12.4.4 設定日期和時間

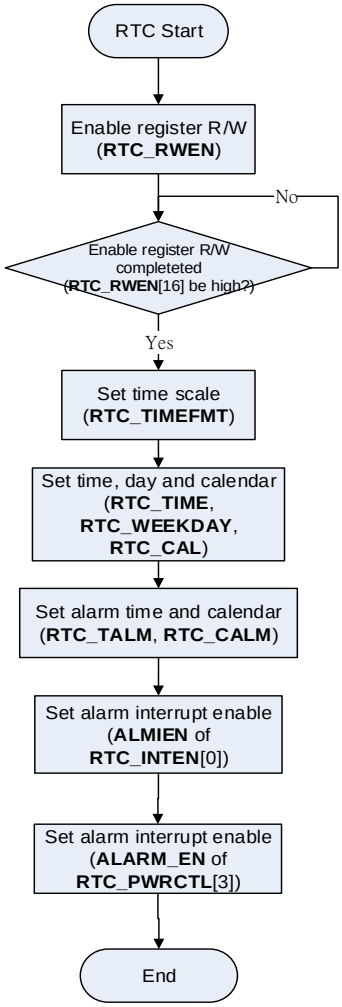
1. 將繼存器 RTC_RWEN 寫入 0xA965,使能 RTC 繼存器訪問(讀/寫)功能
2. 確認 RWENF(RTC_RWEN[16])變成 1,確認 RTC 繼存器訪問功能已被使能.
3. RWENF(RTC_RWEN[16])會在 1024 個 RTC clock 之後自動被清除成 0
4. 設置 24HEN(RTC_TIMEFMT[0]),決定時間格式(12 小時/24 小時)
5. 將年,月,日寫入繼存器 RTC_CAL
6. 將星期一~日寫入繼存器 RTC_WEEKDAY
7. 將時,分,秒寫入繼存器 RTC_TIME



12.4.5 絕對時間鬧鐘設定

1. 將 ALMINT(RTC_INTSTS[0])寫入 1,用來清除鬧鐘中斷
2. 設定目前的時間和日期(參考上述步驟 1~6)
3. 設定鬧鐘日期(年,月,日)到繼存器 RTC_CALM, 另外可以設置日期遮罩.
4. 設定鬧鐘時間(時,分,秒)到繼存器 RTC_TALM, 另外可以設置時間遮罩
5. 設定 ALMIEN(RTC_INTEN[0])使能鬧鐘中斷
6. 設定 ALARM_EN(RTC_PWRCTL[3]),將鬧鐘功能使能

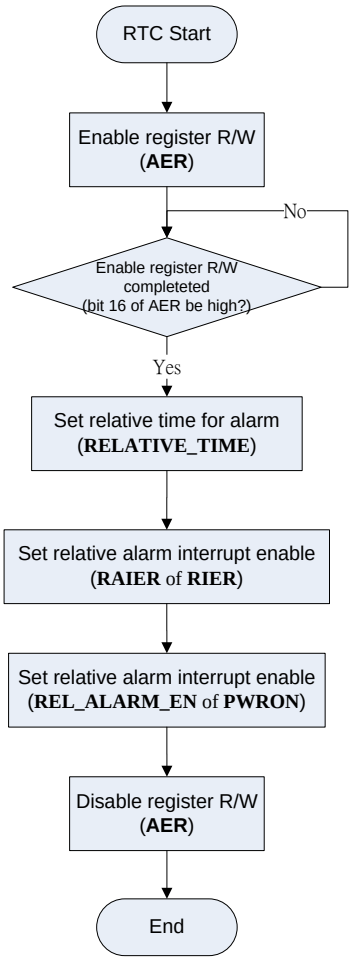
注意: 星期幾的設定也包含在鬧鐘的設定之中



12.4.6 相對時間鬧鐘設定

1. 將 RELALMINT(RTC_INTSTS[4])寫入 1,用來清除鬧鐘中斷
2. 將繼存器 RTC_RWEN 寫入 0xA965,使能 RTC 繼存器訪問(讀/寫)功能
3. 確認 RWENF(RTC_RWEN[16])變成 1,確認 RTC 繼存器訪問功能已被使能.
4. RWENF(RTC_RWEN[16])會在 1024 個 RTC clock 之後自動被清除成 0
5. 設定相對時間 RELALM_TIME(RTC_PWRCTL[27:16])
6. 最大相對時間為 1800(大約 30 分鐘)
7. 設定 RELALMIEN(RTC_INTEN[4])使能相對時間鬧鐘中斷
8. 設定 REL_ALARM_EN(RTC_PWRCTL[4])使能相對時間鬧鐘功能

注意: 當鬧鐘中斷產生是,必須將相對時間鬧鐘中斷RELALMIEN(RTC_INTEN[4])關閉.否則鬧鐘中斷將會在30分鐘之後再度發生

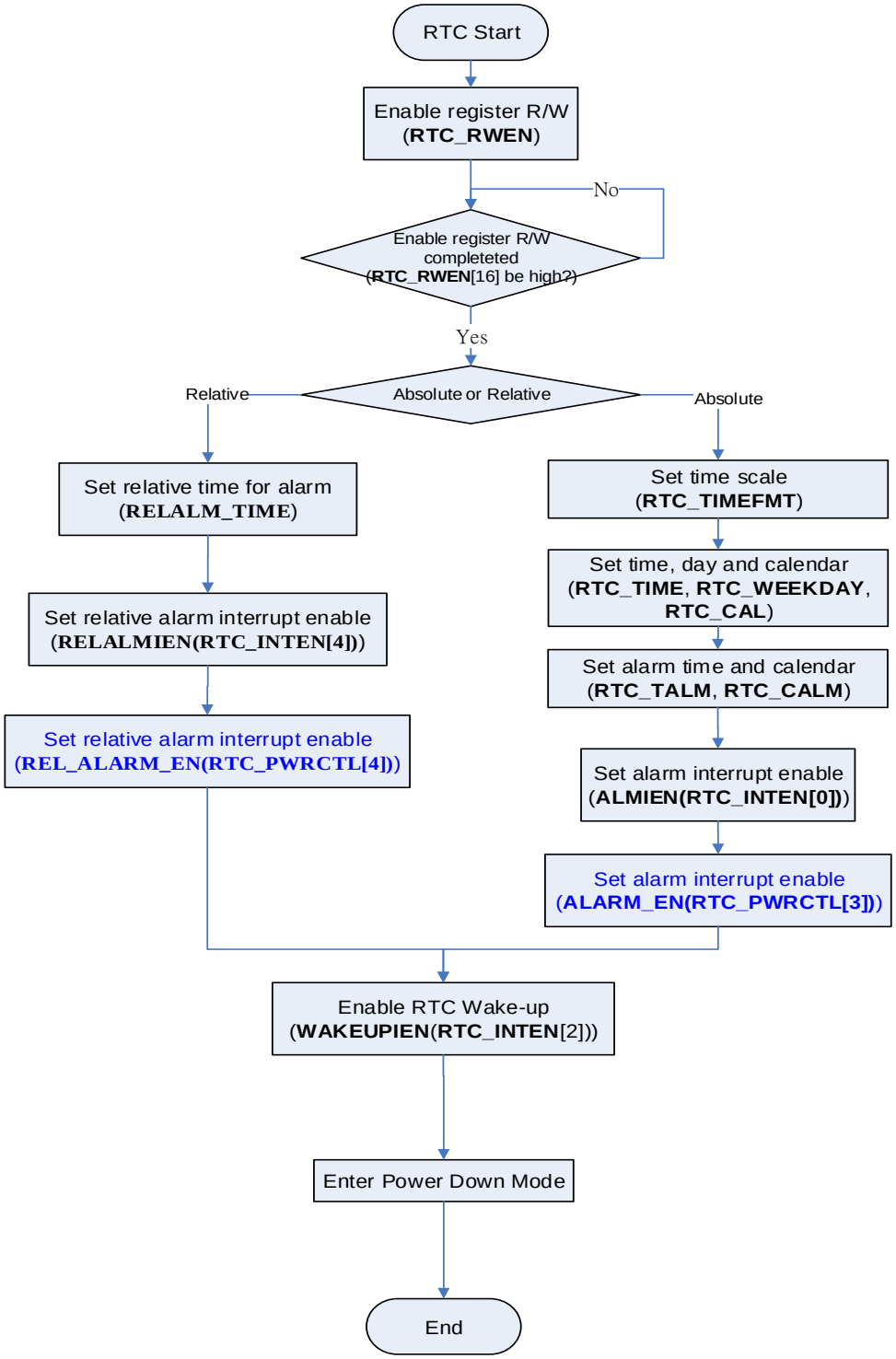


12.4.7 喚醒功能

設置RTC_INTEN寄存器的WAKEUPIEN位可以啟用喚醒功能。當芯片在掉電模式，利用鬧鐘功能將芯片喚醒. 設置程序如下:

1. 將 WAKEUPINT(RTC_INTSTS[2])寫入 1,用來清除喚醒中斷
2. 設定鬧鐘的相對/絕對時間
3. 設定 WAKEUPIEN(RTC_INTEN[2]),使能鬧鐘喚醒功能
4. 使系統進入掉電(power down)模式
5. 當系統時間到達所設定的鬧鐘時間,就會將芯片喚醒

如果用戶不希望使用喚醒功能時,就要不使能鬧鐘喚醒功能WAKEUPIEN(RTC_INTEN[2]). 當鬧鐘中斷產生是,必須將相對時間鬧鐘中斷RELALMIEN(RTC_INTEN[4])關閉.否則鬧鐘中斷將會在30分鐘之後再度發生.



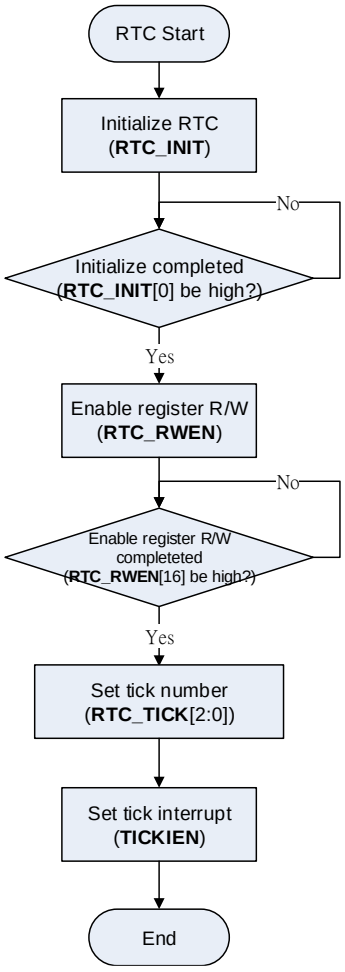
12.4.8 時鐘節拍

設置RTC_TICK [2:0]位，可選擇時鐘節拍的週期為1/128秒，1/64秒，1/32秒，1/16秒，1/8秒，

1/4秒，1/2秒及1秒。

時鐘節拍的中斷處理如下:

1. 將 TICKINT(RTC_INTSTS[1])寫入 1,用來清除節拍中斷
2. 將繼存器 RTC_RWEN 寫入 0xA965,使能 RTC 繼存器訪問(讀/寫)功能
3. 確認 RWENF(RTC_RWEN[16])變成 1,確認 RTC 繼存器訪問功能已被使能.
4. RWENF(RTC_RWEN[16])會在 1024 個 RTC clock 之後自動被清除成 0
5. 設置 RTC_TICK[2:0]位，選擇時鐘節拍的週期
6. 設定 TICKIEN(RTC_INTEN[1])使能節拍中斷



12.4.9 頻率補償

RTC_FREQADJ 允許軟件對32.768 kHz晶振做數位補償。在製造過程中，用戶可以使能SRCSEL(TIMERx_ECTL[16])來測量1Hz的頻率，並且將值存在閃存存儲器中，當成產品第一次上電時計算頻率補償值的參考。時鐘輸入的頻率入須在32776Hz到32761Hz範圍內 才可正確補償.

例如：

計頻器量測數值:32773.65

整數部份：32773

參照下表

Integer part of detected value	RTC_FREQADJ[11:8]	Integer part of detected value	RTC_FREQADJ[11:8]
32776	1111	32768	0111
32775	1110	32767	0110
32774	1101	32766	0101
32773	1100	32765	0100
32772	1011	32764	0011
32771	1010	32763	0010
32770	1001	32762	0001
32769	1000	32761	0000

RTC_FREQADJ的Integer就要選擇設定0xC

小數部份：0.65 * 60 = 39 = 0x27, RTC_FREQADJ的Fraction就需要填入0x27

所以 RTC_FREQADJ 寄存器需填入 0xC27

12.5 寄存器

R : Read only, **W** : Write only, **R/W** : Both read and write, **C** : Only value 0 can be written

Register	Address	R/W	Description	Reset Value
RTC_BA = 0xB004_1000				
RTC_INIT	RTC_BA+0x000	R/W	RTC Initiation Register	Undefined
RTC_RWEN	RTC_BA+0x004	R/W	RTC Access Enable Register	0x0000_0000
RTC_FREQADJ	RTC_BA+0x008	R/W	RTC Frequency Compensation Register	0x0000_0700
RTC_TIME	RTC_BA+0x00C	R/W	RTC Time Counter Register	0x0000_0000
RTC_CAL	RTC_BA+0x010	R/W	RTC Calendar Counter Register	0x0005_0101
RTC_TIMEFMT	RTC_BA+0x014	R/W	RTC Time Format Selection Register	0x0000_0001
RTC_WEEKDAY	RTC_BA+0x018	R/W	RTC Day of the Week Register	0x0000_0006

RTC_TALM	RTC_BA+0x01C	R/W	RTC Time Alarm Register	0x0000_0000
RTC_CALM	RTC_BA+0x020	R/W	RTC Calendar Alarm Register	0x0000_0000
RTC_LEAPYEAR	RTC_BA+0x024	R	RTC Leap year Indicator Register	0x0000_0000
RTC_INTEN	RTC_BA+0x028	R/W	RTC Interrupt Enable Register	0x0000_0000
RTC_INTSTS	RTC_BA+0x02C	R/C	RTC Interrupt Status Register	0x0000_0000
RTC_TICK	RTC_BA+0x030	R/W	RTC Time Tick Register	0x0000_0000
RTC_PWRCTL	RTC_BA+0x034	R/W	RTC Power Control Register	0x0000_7000
RTC_PWRCNT	RTC_BA+0x038	R	RTC Power Control Counter Register	0x0000_0000
RTC_SPR0 ~ RTC_SPR15	RTC_BA+0x040 ~ RTC_BA+0x07C	R/W	RTC Spare Register 0 ~ 15	0x0000_0000

13 通用異步收發器 (UART)

13.1概述

通用非同步收發器(UART)在從外設接收資料時執行串列到並行的轉換，從CPU發送資料時執行並行到串列的轉換。該UART控制器同時支援IrDA(SIR)功能，LIN功能和RS-485功能模式。每個UART通道支援9種類型的中斷，包括接收閾值到達中斷(RDAINT)，發送FIFO空中斷(THREINT)，線狀態中斷(break錯誤，校驗錯誤，格式錯誤或者RS-485中斷)(RLSINT)，超時中斷(RXTOINT)，MODEM狀態中斷(MODEMINT)，緩存錯誤中斷(BUFERRINT)，喚醒中斷(WKINT)，自動串列傳輸速率檢測或自動串列傳輸速率計數器溢出標誌中斷(ABRINT)和LIN功能中斷(LININT)。

UART0~9內嵌一個16位發送FIFO (TX_FIFO)和一個16位接收FIFO (RX_FIFO) 。在操作過程中CPU可以隨時讀UART的狀態。報告的狀態資訊包括已經被UART執行的傳輸操作的類型和條件，也包括當接收資料可能發生的3種錯誤條件(校驗錯誤，格式錯誤和break中斷)。UART控制器支援自動串列傳輸速率檢測，自動串列傳輸速率檢測控制為串列傳輸速率發生器所進行的對傳入的時鐘/資料速率的測量進程，並可按照使用者的意願被讀寫。UART控制器也支援CTSn喚醒功能，當系統處於掉電模式時，一個傳入的CTSn信號將會把CPU從掉電模式喚醒。UART包括一個可程式設計的串列傳輸速率發生器，它可以將輸入晶振除以一個除數來得到收發器需要的串列時鐘。串列傳輸速率公式為串列傳輸速率 = UART_CLK /M* [BRD + 2]，其中 BRD 在串列傳輸速率分頻寄存器 (UARTx_BAUD) 中定義。下表列舉了不同條件下的等式和 UART 串列傳輸速率設置表。

Mode	DIV_X_EN	DIV_X_ONE	DIVIDER X	BRD	Baud Rate Equation
0	Disable	0	Don't Care	A	UART_CLK / [16 * (A+2)]
1	Enable	0	B	A	UART_CLK / [(B+1) * (A+2)] , B must >= 8
2	Enable	1	Don't care	A	UART_CLK / (A+2), A must >=9

System Clock = Internal 22.1184 MHz high-speed oscillator						
Baud Rate	Mode0		Mode1		Mode2	
	Parameter	Register	Parameter	Register	Parameter	Register
921600	x	x	A=0,B=11	0x2B00_0000	A=22	0x3000_0016
460800	A=1	0x0000_0001	A=1,B=15 A=2,B=11	0x2F00_0001 0x2B00_0002	A=46	0x3000_002E
230400	A=4	0x0000_0004	A=4,B=15 A=6,B=11	0x2F00_0004 0x2B00_0006	A=94	0x3000_005E
115200	A=10	0x0000_000A	A=10,B=15 A=14,B=11	0x2F00_000A 0x2B00_000E	A=190	0x3000_00BE

57600	A=22	0x0000_0016	A=22,B=15 A=30,B=11	0x2F00_0016 0x2B00_001E	A=382	0x3000_017E
38400	A=34	0x0000_0022	A=62,B=8 A=46,B=11 A=34,B=15	0x2800_003E 0x2B00_002E 0x2F00_0022	A=574	0x3000_023E
19200	A=70	0x0000_0046	A=126,B=8 A=94,B=11 A=70,B=15	0x2800_007E 0x2B00_005E 0x2F00_0046	A=1150	0x3000_047E
9600	A=142	0x0000_008E	A=254,B=8 A=190,B=11 A=142,B=15	0x2800_00FE 0x2B00_00BE 0x2F00_008E	A=2302	0x3000_08FE
4800	A=286	0x0000_011E	A=510,B=8 A=382,B=11 A=286,B=15	0x2800_01FE 0x2B00_017E 0x2F00_011E	A=4606	0x3000_11FE

UART控制器用2種低電平信號，CTSn (clear-to-send)和RTSn(request-to-send)，來支援自動流控制功能，用來控制UART和外部設備(如：Modem)之間的資料流程傳輸。當自動流控被使能時，UART將不允許接收資料直到UART向外部設備置RTSn(RTSn為高)為有效，當RX FIFO內的位元組數等於RTSTRGLV(UA_FIFO[19:16])的值時，RTSn信號被置為無效。當UART控制器從外部設備偵測到 CTSn有效信號(CTSn為高)時，UART控制器向外發送資料。如果有效的CTSn信號未被探測到，UART 控制器將不會向外發送資料。

UART控制器支援喚醒系統功能。當系統處於掉電模式時，UART可以通過CTSn引腳來喚醒系統。

UART控制器提供串列IrDA(SIR,串列紅外)功能(用戶需設定(FUNCSEL(UART_FUNCSEL[2:0]) = 010)使能 IrDA 功能)，SIR定義短程紅外非同步序列傳輸模式，該模式有1個起始位元元，8個資料位元,和1個停止位. 最大資料速率為115.2 Kbps (半雙工). IrDA SIR包括一個IrDA SIR協定編碼/解碼器. IrDA SIR協定只是半雙工協定，不能同時傳輸和接收資料。IrDA SIR實體層規定在傳輸和接收之間至少10ms傳輸延時，該特性必須由軟體執行

UART控制器的另一個可選的功能是RS-485 9位元元元模式，由RTS腳控制收/發方向或由軟體程式設計執行該功能。RS-485模式通過設置寄存器UA_FUN_SEL來選擇。RS-485驅動器的使能由RTS控制信號實現控制。在 RS-485模式，RX和TX的許多特性與UART相同.

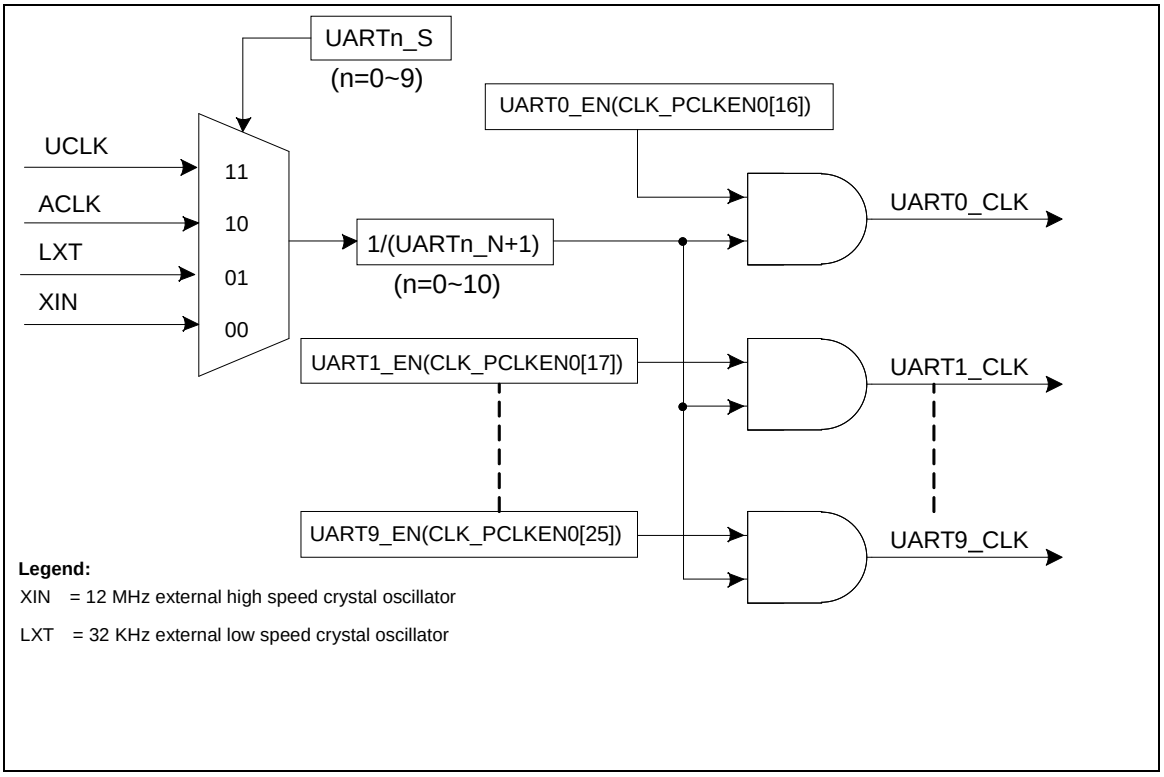
LIN模式可以通過設定UART_FUN_SEL寄存器中的LIN_EN位來選擇。在LIN模式下，依照LIN標準，要求資料格式為1個起始位元，8個資料位元和1個停止位

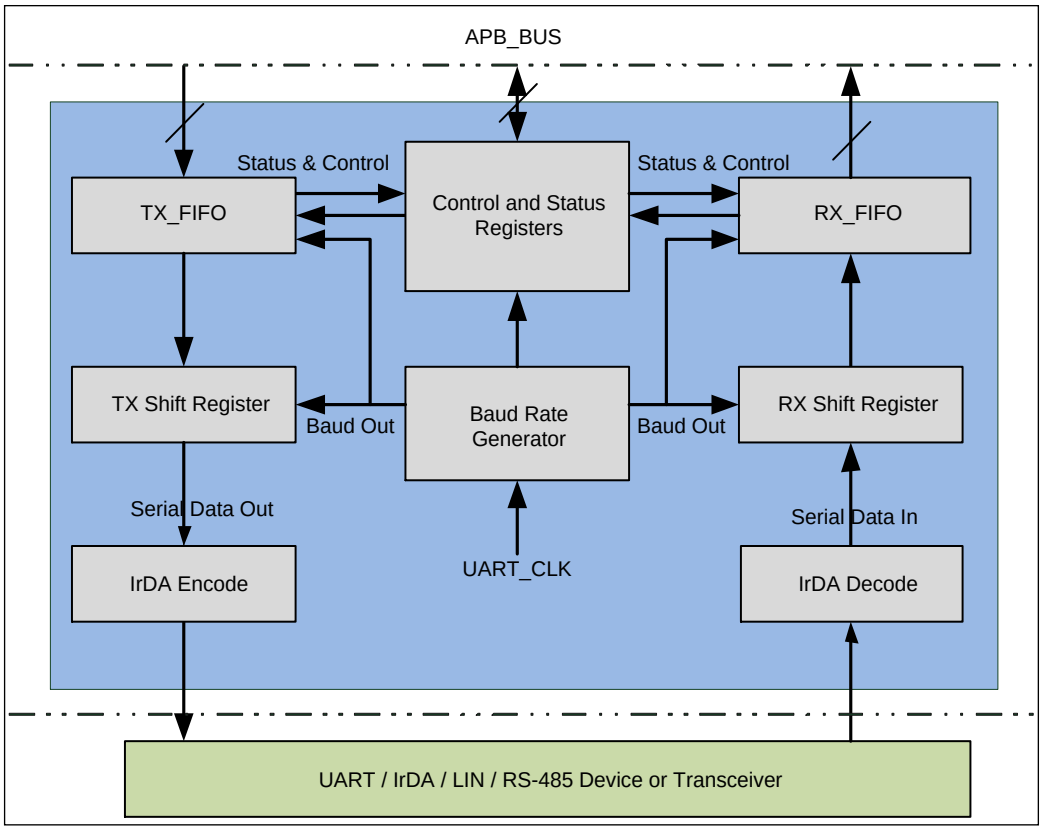
13.2特性

- 全雙工，非同步通信
- 獨立的接收/發送64/16位元組FIFO，用於資料裝載
- 支援硬體自動流控制/流控制功能(CTSn, RTSn)和可程式設計的(CTSn, RTSn)流控制觸發電平
- 對於每個通道，支援可程式設計的串列傳輸速率發生器

- 支援可程式設計的接收緩存觸發極限值
- 支援CTSn喚醒功能
- 支援8位接收緩存定時溢出檢測功能
- 通過設置寄存器DLY(UART_TOUT[15:8])可程式設計設定在上一次資料傳輸的停止位與下一次資料傳輸的開始位元之間發送資料的延遲時間
- 支援 break error, frame error, parity error 和接收/發送緩存溢出檢測功能
- 完全可程式設計的串口特性
  - 可程式設計為 5-, 6-, 7-, 8-位元的數據位元
  - 可程式設計的校驗位, even, odd, no parity 或 stick parity bit 產生和偵測
  - 可程式設計為 1, 1.5, 或 2 位的停止位
- 支援IrDA SIR功能模式
- 支援LIN功能模式(僅UART1/2)
- 支援RS485功能模式

13.3方塊圖

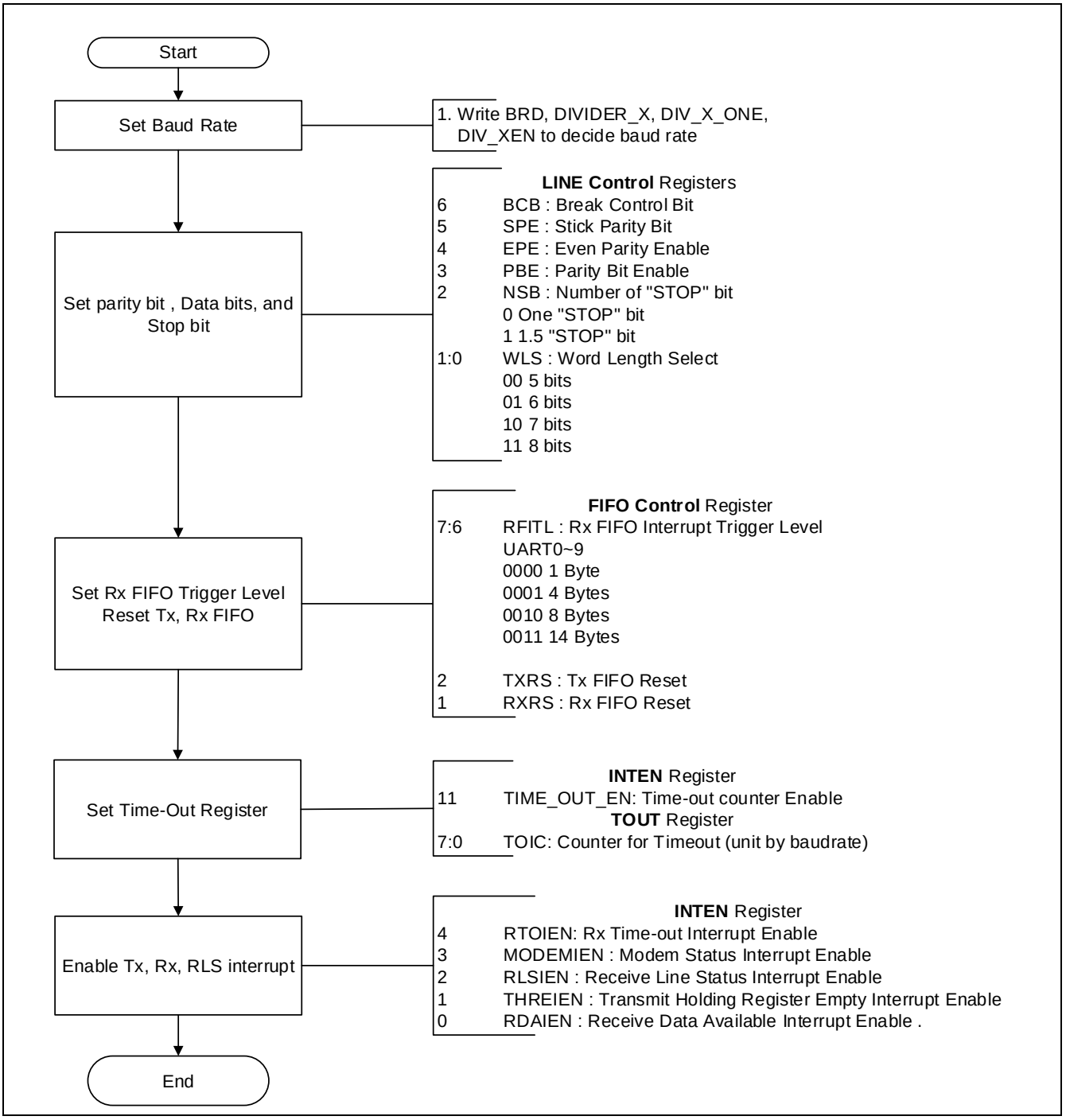




13.4功能描述

13.4.1 初始化

在開始使用UART傳輸之前，必須要對UART做初始設置，包含串列傳輸速率，奇偶檢驗位，數據位，停止位的設置。另外，如果將TX，RX，RLS的中斷使能，則在每一次的傳輸結束時，就可以觸發相對的的中斷。



13.4.2 IrDA 功能模式

UART 支援 IrDA SIR (串列紅外) 發送編碼器和接收解碼器，IrDA模式可通過設定UART_FUNCSEL寄存器中的FUNCSEL位來選擇，當UART控制器工作在IrDA模式時，必須通

過設定RFITL(UART_FIFO[7:4]) = 000 來把接收FIFO觸發閾值設置為1。

IrDA模式下，BAUDM1(UART_BAUD[29])位不能被使能。

Baud Rate = Clock / (16 * (BRD + 1))，其中BRD是在串列傳輸速率分頻寄存器UART_BAUD中定義的串列傳輸速率除數。

IrDA SIR編碼/解碼器提供UART資料流程和半雙工串列SIR之間的轉換。

程式設計流程示例：

1. 程式設計 UART_FUNCSEL 寄存器中的 FUNCSEL 位來選擇 IrDA 功能.
2. 設定 TXINV(UART_IRDA[5]) = 0 選擇 TX 訊號不反相.
3. 設定 RXINV(UART1_IRDA[6]) = 1 選擇 RX 訊號反相.
4. 通過設定 TXEN(UART_IRDA[1])選擇半雙工串列為 TX 或 RX。當 TXEN(UART_IRDA[1]) = 1 時，是選擇 TX。當 TXEN(UART_IRDA[1]) = 0 時，是選擇 RX。

13.4.3 RS485 功能模式

UART支援RS-485 9位元元元模式功能。RS-485模式可以通過設置寄存器UART_FUNCSEL來選擇。RS-485 驅動器控制可以通過使用來自一個非同步串列口的RTSn控制信號來實現。在RS-485模式下，RX和TX的很多特性跟在UART模式下一樣。

在RS-485功能模式下，第9位元元元將會被配置為位元元址位元元元，對於資料字元，第9位要被設置為0，軟體可以程式設計UART_LINE寄存器來控制第9位(當PBE，EPE和SPE被置位，第9位作為0被發送；而當PBE和SPE被置位，EPE被清零時，第9位作為1被發送)。該模式下，UART控制器支援三種操作模式，分別是RS-485普通多點操作模式(RS-485 NMM模式)，RS-485自動位元址檢測操作模式(RS-485 AAD模式)和RS-485自動方向控制操作模式(RS-485 AUD模式)，可通過程式設計UART_ALTCTL寄存器選擇這三種模式中的其中一種，並且軟體可以通過設置DLY(UART_TOUT[15:8])寄存器來在上一次資料傳輸的停止位與下一次資料傳輸的開始位之間插入一個發送延時。

13.4.3.1 RS-485 普通多點操作模式 (NMM)

在RS-485普通多點操作模式下，在檢測到位址位元組之前(bit 9 = “1”)，軟體可以決定接收器是否忽略資料。當位址位元組被硬體檢測到(bit 9 = “1”)，位址位元組資料將會被存儲到RX-FIFO，軟體可以通過設定RXOFF(UART_FIFO[8])位來決定是使能還是禁止接收器接受接下來的資料位元組。如果接收器被使能(RXOFF(UART_FIFO[8])位為0)，所有接收到的資料都將會被接受並存儲到RX-FIFO；如果接收器被禁止(RXOFF(UART_FIFO[8])位為1)，所有接收到的位元組資料都會被忽略，直到下一個位址位元組被檢測到。如果軟體通過設定RXOFF(UART_FIFO[8])位為1來禁止接收器，當下一個位址位元組被檢測到，UART控制器會清零RXOFF(UART_FIFO[8])位元，位址位元組資料將會被存儲到RX-FIFO。

程式設計流程示例：

1. 程式設計 UART_FUNCSEL 寄存器的 FUNCSEL 位來選擇 RS-485 功能。

2. 程式設計 UART_FIFO 寄存器的 RXOFF 位元來決定在位址位元組被檢測到之前(bit 9 = “1”)，是否存儲接收到的資料。
3. 通過設定 RS485NMM(UART_ALTCTL[8])寄存器來程式設計設定 RS-485_NMM。
4. 當一個位址位元組被檢測到(bit 9 = “1”)，硬體會置位元 RLSIF(UART_INTSTS[2])和 ADDRDETF(UA_FIFOSTS[3])標誌。
5. 體可以通過設定 RXOFF(UA_FIFO[8])來決定是否接受接下來的資料。
6. 重複步驟 4 和步驟 5。

13.4.3.2 RS-485 自動位元址識別操作模式 (AAD)

在 RS-485 自動位元址識別操作模式下，接收器在檢測到位址位元組 (bit9 = “1”)，並且位址位元組資料與ADDRMV(UART_ALTCTL[31:24])的值相匹配之前，將忽略所有資料。位址位元組資料將存儲在 RX-FIFO，接下來的所有資料將被接受並存儲於RX-FIFO，直到位址位元組與ADDRMV(UART_ALTCTL[31:24])的值不匹配。當處於RS-485 AAD模式時，不要寫任何值到RXOFF(UART_FIFO[8])位。

程式設計流程示例：

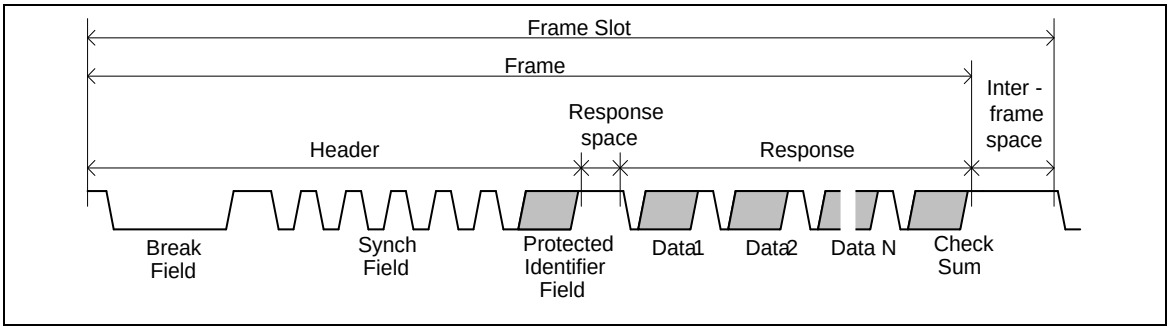
1. 程式設計 UART_FUNCSEL 寄存器中的 FUNCSEL 位來選擇 RS-485 功能.
2. 通過設定 UART_ALTCTL[9]寄存器來程式設計設定 RS485_AAD.
3. 當位址位元組被檢測到(bit9 = “1”)，硬體會比較位址位元組與 ADDRMV(UART_ALTCTL[31:24]) 的值。
4. 如果位址位元組與 ADDRMV(UART_ALTCTL[31:24])的值相匹配，硬體會置位元 RLSIF(UART_INTSTS[2])和 ADDRDETF(UART_FIFOSTS[3])標誌，接收器會存儲位址位元組到 RX-FIFO，接受接下來的資料，並把它們存儲到 RX-FIFO，直到下一個位址位元組被檢測到。 然而，如果位址位元組與 ADDRMV(UART_ALTCTL[31:24])的值不匹配，硬體將忽略位址位元組，並忽略接下來的資料傳輸。
5. 重複步驟 3 和步驟 4。

13.4.3.3 RS-485 自動方向模式 (AUD)

RS-485控制器的另一個功能是RS-485自動方向控制。RS-485驅動器控制可以通過使用來自一個非同步串列口的RTSn控制信號來實現。RTSn線被連接到RS-485驅動器使能引腳，設置RTSn線為高(邏輯1)將使能RS-485驅動器；設置RTSn線為低(邏輯0)，將會使驅動器進入高阻態。用戶可以通過設置寄存器 RTSACTLV(UART_MODEM[9])位來改變RTSn驅動電平

13.4.4 LIN 功能模式

UART支援LIN功能，LIN模式可以通過設定UART_FUNCSEL寄存器的來選擇。在LIN模式下，依照LIN的標準，每個位元組域初始由一個值為0的開始位(顯性)，後跟8個資料位元(LSB優先)，最後是值為1的1個停止位(隱性)。.



LIN總綫發送傳輸(TX)編程流程：

1. 設置寄存器 UART_FUNCSEL 的 LIN_EN 位使能 LIN 總綫模式
2. 寫 BKFL(UART_LINCTL[19:16])選擇 break 域長度(break 域長度是 UART_LIN_BKFL+2)
3. 設置寄存器 SENDH(UA_ALTCTL[8])位開始傳輸(當 break 域操作完成，SENDH 將被自動清除)
4. 寫 0x55 到 UART_DAT 請求同步域傳輸
5. 寫保護標識符到 UART_DAT，請求標識符域傳輸
6. 當最後字節 DAT 的停止位被發到總綫後，硬體將置 UART_FIFOSTS 寄存器中的 TXEMPTY 為 1
7. 寫 N 字節數據和檢驗和(checksum)到 UART_DAT，然後重復步驟 5 和步驟 6 傳輸數據

LIN總綫接收傳輸(RX)編程流程：

1. 設置寄存器 UART_FUSCSEL 的 LIN_EN 位使能 LIN 總綫模式
2. 設置寄存器 SENDH(UART_ALTCTL[8]) = 1，使能 LIN RX 模式
3. 等待 UART_LINSTS 中的標誌位 BKDETF，用來檢查 RX 有無 break 域接收
4. 等待 UART_INTSTS 中的標誌位 RDAIF 和讀寄存器 UART_DAT

13.4.5 PDMA 傳輸功能

通過配置PDMA參數並將UART_DAT設置為PDMA目標地址。當TXPDMAEN(UART_INTEN [14])設置為1時，控制器將向PDMA控制器發出請求，以自動啟動PDMA傳輸過程。

通過配置PDMA參數並將UART_DAT設置為PDMA源地址。當RXPDMAEN(UART_INTEN [15])設置為1時，控制器將啟動PDMA接收過程。當RX FIFO緩衝區中有數據時，UART控制器將自動向PDMA控制器發出請求。

Note：如果STOPn(PDMA_STOP [n])被設置為停止UART RXPDMA任務，並且UART接收未完成。 UART控制器將完成傳輸並將當前接收數據存儲在接收緩衝區中。通過讀取RXEMPTY(UART_FIFOSTS [14])來檢查接收緩衝區中是否有有效數據。

13.4.6 UART 控制器喚醒功能

UART控制器支援喚醒系統功能。喚醒功能包括nCTS引腳，輸入數據喚醒，接收數據FIFO達到閥值喚醒，RS-485位址匹配(AAD模式)喚醒和接收數據FIFO閥值超時喚醒功能。CTSWKF(UART_WKSTS [0])，DATWKF(UART_WKSTS [1])，RFRTWKF(UART_WKSTS [2])，RS485WKF(UART_WKSTS [3])或TOUTWKF(UART_WKSTS [4])會導致喚醒中斷標誌WKIF(UART_INTSTS [6])被生成。 如果WKIEN(UART_INTEN [6])使能，則喚醒中斷標誌WKIF(UART_INTSTS [6])會導致喚醒中斷WKINT(UART_INTSTS [14])產生。

nCTS引腳喚醒：

當系統處於掉電模式且WKCTSEN（UART_WKCTL [0]）置1時，nCTS引腳的切換可以喚醒系統。 如果WKCTSEN（UART_WKCTL [0]）被使能，nCTS引腳的切換會導致產生nCTS喚醒標誌CTSWKF（UART_WKSTS [0]）。 nCTS喚醒如圖13.4-1和圖13.4-2所示。

nCTS 喚醒狀況1 (nCTS從低變高)

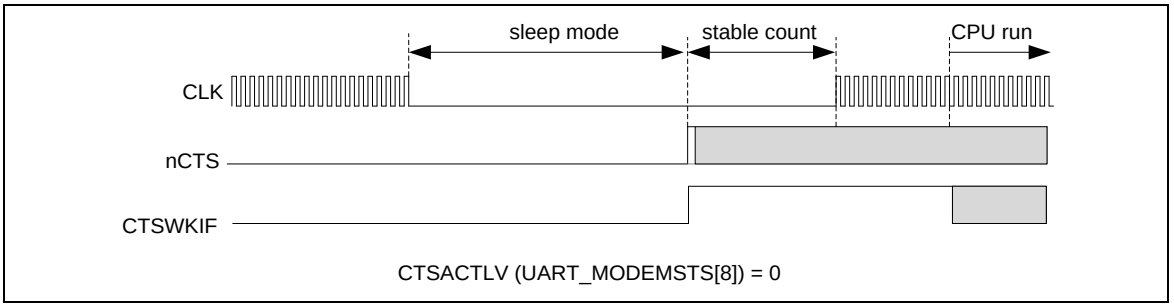


圖 13.4-3 UART nCTS 喚醒狀況 1

nCTS 喚醒狀況2 (nCTS從高變低)

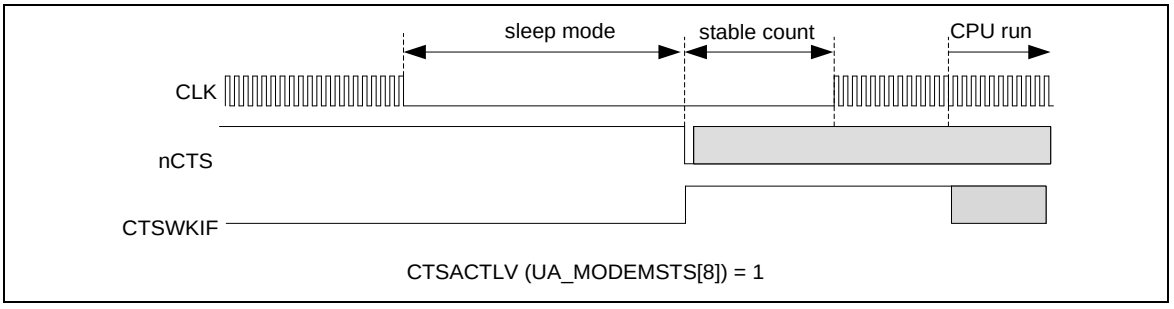


圖 13.4-4 UART nCTS 喚醒狀況 2

RX數據喚醒

當系統處於掉電模式且WKDATEN(UART_WKCTL [1])置1時，輸入數據(UART_RXD)可喚醒系統。為了在系統喚醒後接收輸入數據，應設置STCOMP(UART_DWKCOMP [15：0])。STCOMP的這些位字段指示在系統從掉電模式喚醒時，UART_CLK選擇了多少個時鐘週期，

UART控制器可以獲得第一位(起始位)。

當輸入數據喚醒系統時，輸入數據將被接收並存儲在FIFO中。如果使能了WKDATEN(UART_WKCTL [1])，則輸入數據(UART_RXD)會導致傳入數據喚醒標誌DATWKF(UART_WKSTS [1])生成。圖13.4-5中顯示了數據喚醒。

Note 1：應選擇UART控制器時鐘源作為HIRC，起始位的補償時間約為10.865us。這意味著STCOMP(UART_DWKCOMP [15：0])的值可以設置為0x207。

Note 2：BRD的值(UART_BAUD [15：0])應該大於STCOMP（UART_DWKCOMP [15：0]）。

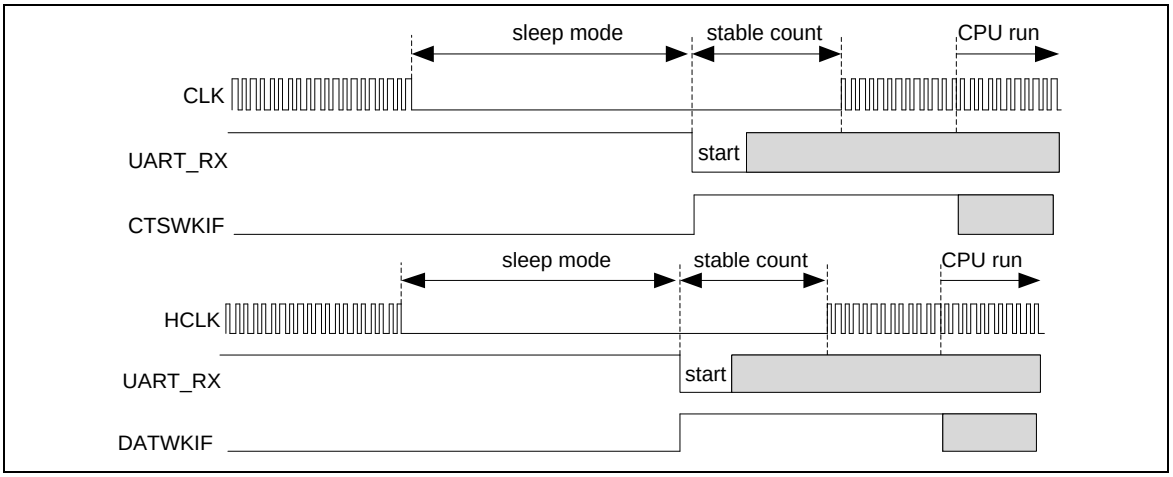


圖 13.4-6 UART RX 數據喚醒

RX FIFO達到閥值喚醒

通過設置WKRFRTEN(UART_WKCTL [2])，達到RX FIFO閥值喚醒功能。在掉電模式下，當RX FIFO中的接收數據達到閥值RFITL(UART_FIFO [7：4])時，可以喚醒系統。如果WKRFRTEN(UART_WKCTL [2])被使能，RX FIFO中接收數據的數量達到閥值RFITL(UART_FIFO [7：4])，使RX FIFO達到閥值喚醒標誌RFRTWKF(UART_WKSTS [2])被生成。RX FIFO達到閥值喚醒如圖13.4-7所示。

Note：在掉電模式下，應選擇UART控制器時鐘源作為LXT來接收數據。

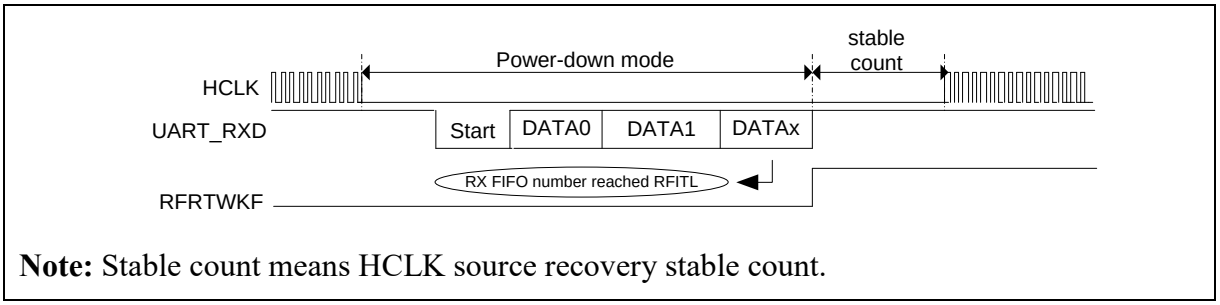
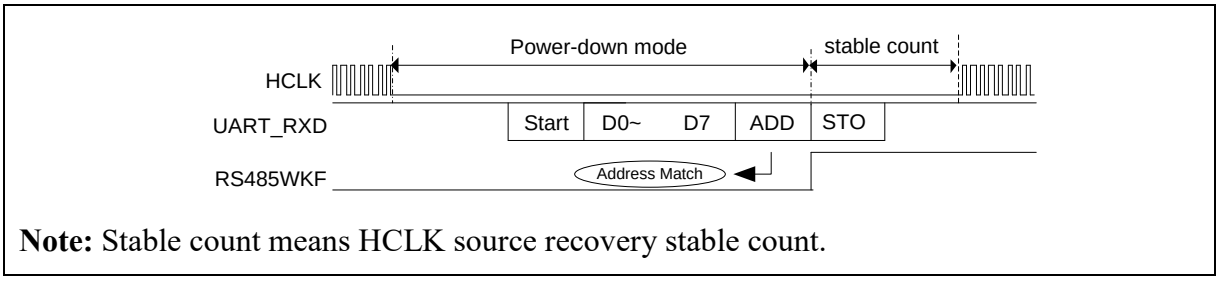


圖 13.4-8 RX FIFO 達到閥值喚醒

RS-485位址匹配(AAD模式)喚醒

通過設置WKRFRTEN(UART_WKCTL [2])和WKRS485EN(UART_WKCTL [3])使能RS-485位址匹配喚醒功能。 該功能用於RS-485功能模式下的RS-485自動位址檢測(AAD)模式，並且ADDRDEN(UART_ALTCTL [15])被設置為1在掉電模式下，當檢測到位址字節並與ADDRMV(UART_ALTCTL [31:24])或接收FIFO中的接收數據達到閥值RFITL(UART_FIFO [7：4])時，它會喚醒系統。 如果WKRS485EN(UART_WKCTL [3])被使能，當檢測到一個位址字節並且與引起 RS485 位址匹配 (AAD 模式) 喚醒標誌 RS485WKF(UART_WKSTS [3]) 的ADDRMV(UART_ALTCTL [31:24])被生成。 RS-485位址匹配(AAD模式)喚醒如圖13.4-9所示。

Note：在掉電模式下，應選擇UART控制器時鐘源作為LXT來接收數據。



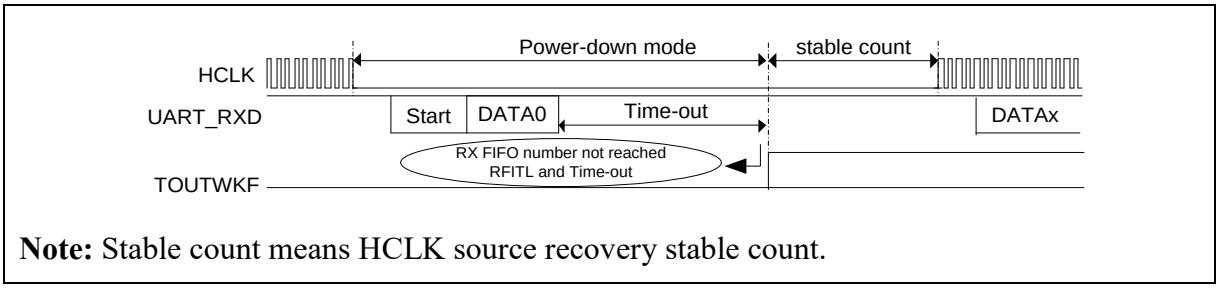
Note: Stable count means HCLK source recovery stable count.

圖 13.4-10 RS-485 AAD 模式下位址匹配喚醒

RX FIFO閥值超時喚醒

RX FIFO 閥值超時喚醒功能通過設置 WKRFRTEN(UART_WKCTL [2]) 和WKTOUTEN(UART_WKCTL [4])使能。設置TOCNTEN(UART_INTEN [11])使能接收緩衝區超時計數器。在掉電模式下，當RX FIFO中的接收數據數量未達到閥值RFITL(UART_FIFO [7：4])且超時計數器等於超時值TOIC(UART_TOUT [7：0])，它可以喚醒系統。如果WKTOUTEN(UART_WKCTL [4])被使能，當超時計數器等於RX FIFO閥值超時喚醒的超時值TOIC(UART_TOUT [7：0])時，將生成標誌TOUTWKF(UART_WKSTS [4])。RX FIFO閥值超時喚醒如圖13.4-11所示。

Note：在掉電模式下，應選擇UART控制器時鐘源作為LXT來接收數據。



Note: Stable count means HCLK source recovery stable count.

圖 13.4-12 RX FIFO 閥值超時喚醒

13.5寄存器

R: read only, W: write only, R/W: both read and write.

Register	Offset	R/W	Description	Reset Value
UART Base Address: **UARTx_BA = 0x4007_0000 + (0x1000 * x)** **x=0~9**				
UART_DAT	UARTx_BA+0x00	R/W	UART Receive/Transmit Buffer Register	Undefined
UART_INTEN	UARTx_BA+0x04	R/W	UART Interrupt Enable Register	0x0000_0000
UART_FIFO	UARTx_BA+0x08	R/W	UART FIFO Control Register	0x0000_0101
UART_LINE	UARTx_BA+0x0C	R/W	UART Line Control Register	0x0000_0000
UART_MODEM	UARTx_BA+0x10	R/W	UART Modem Control Register	0x0000_0200
UART_MODEMSTS	UARTx_BA+0x14	R/W	UART Modem Status Register	0x0000_0110
UART_FIFOSTS	UARTx_BA+0x18	R/W	UART FIFO Status Register	0xB040_4000
UART_INTSTS	UARTx_BA+0x1C	R/W	UART Interrupt Status Register	0x0040_0002
UART_TOUT	UARTx_BA+0x20	R/W	UART Time-out Register	0x0000_0000
UART_BAUD	UARTx_BA+0x24	R/W	UART Baud Rate Divider Register	0x0F00_0000
UART_IRDA	UARTx_BA+0x28	R/W	UART IrDA Control Register	0x0000_0040
UART_ALTCTL	UARTx_BA+0x2C	R/W	UART Alternate Control/Status Register	0x0000_000C
UART_FUNCSEL	UARTx_BA+0x30	R/W	UART Function Select Register	0x0000_0000
UART_LINCTL	UARTx_BA+0x34	R/W	UART LIN Control Register	0x000C_0000
UART_LINSTS	UARTx_BA+0x38	R/W	UART LIN Status Register	0x0000_0000

UART_BRCOMP	UARTx_BA+0x3C	R/W	UART Baud Rate Compensation Register	0x0000_0000
UART_WKCTL	UARTx_BA+0x40	R/W	UART Wake-up Control Register	0x0000_0000
UART_WKSTS	UARTx_BA+0x44	R/W	UART Wake-up Status Register	0x0000_0000
UART_DWKCOMP	UARTx_BA+0x48	R/W	UART Incoming Data Wake-up Compensation Register	0x0000_0000

14 智能卡接口 (SC)

14.1 概述

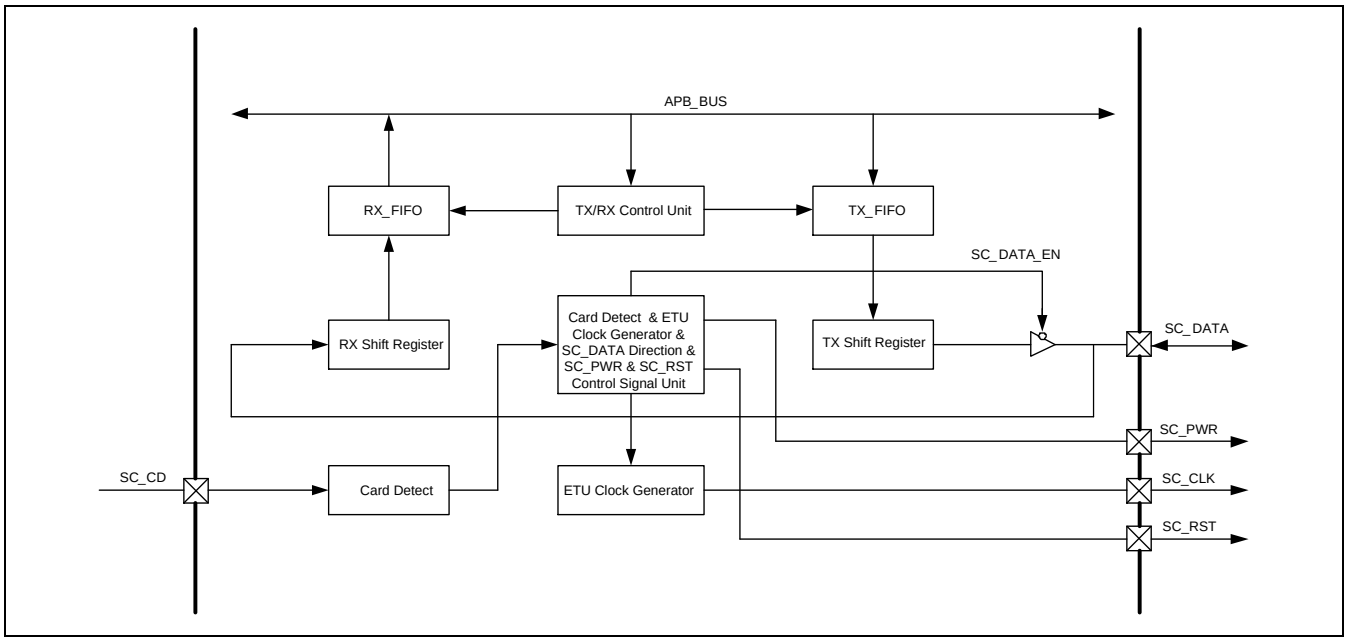
智能卡介面控制器 (SC controller) 是基於 ISO/IEC 7816-3 標準並完全相容 PC/SC 規格. 它也提供卡插入/移除的狀態

14.2 特性

- 與ISO-7816-3 T = 0, T = 1 相容.
- 與EMV2000 相容
- 支援 2 個 ISO-7816-3 埠
- 用於資料負載的獨立的接收/發送 4 位元組入口緩存
- 可程式設計的發送時鐘頻率
- 可程式設計的接收器緩存觸發水準.
- 可程式設計的保護時間選擇 (11 ETU ~ 267 ETU).
- 一個 24-位和兩個8-位元數目器用於請求應答 (Answer to Reset (ATR)) 和等待時間處理
- 支援自動反向約定功能
- 支援傳送器和接收器錯誤重試和錯誤數目限制功能
- 支援硬體啟動序列處理
- 支援硬體暖重定序列處理
- 支援硬體釋放序列處理.
- 支援當檢測到卡移除時，硬體自動釋放序列
- 支援 UART 模式
  - 支持全雙工, 異步傳輸
  - 用於資料負載的獨立的接收/發送 4 位元組入口緩存
  - 在每個通道上支援可程式設計的串列傳輸速率發生器
  - 支持可程式設計的接收器緩存觸發水準
  - 從最後一個停止位被從 TX-FIFO 發出到釋放的發送資料延遲時間通過設定 SCx_EGTR[EGT] 可程式設計

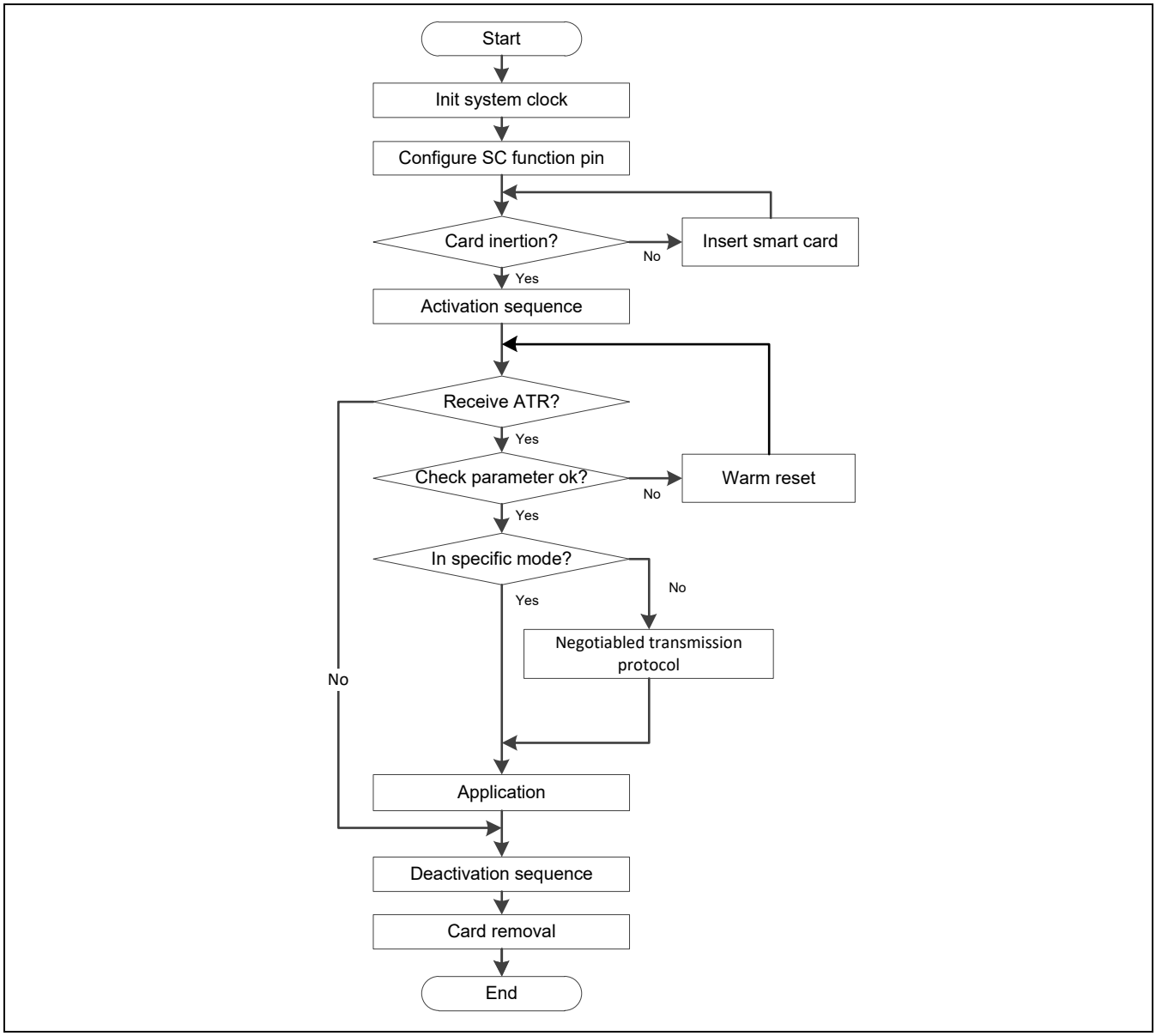
- 可程式設計的偶, 奇或者無校驗位生成和檢測.
- 可程式設計的停止位, 1 或 2 停止位生成

14.3 方塊圖



14.4 功能描述

本節會描述智能卡接口的操作模式, 但是 ISO 7816 或是 EMV 規範則不在介紹範圍之內. 若是要編寫驅動程式操控智能卡, 建議須對 ISO 7816, 或是 EMV 規範有基礎了解. 基本的智能卡控制流程如下圖:



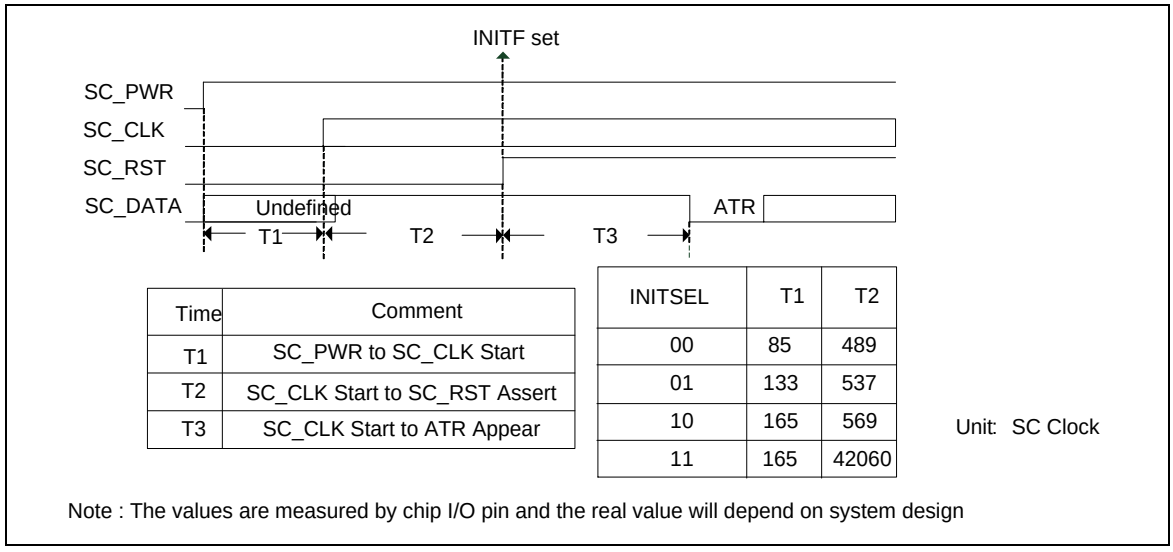
14.4.1 啟動 (Cold Reset)

智慧卡介面控制器支援硬體啟動, 暖重定和釋放序列. 啟動序列能夠被軟體或硬體控制. 如果軟體想控制, 軟體能夠控制 SC_PINCTL寄存器去處理啟動序列, 透過軟件控制啟動序列的方式如下:

- 將 RSTSTS (SC_PINCTL[18]) 清為0, 設置 SC_RST 為低
- 將 PWRSTS (SC_PINCTL[18]) 置 1, 設置 SC_PWR 在高電平.
- 將 DATSTS (SC_PINCTL[16]) 置 1, 將 SC_DATA 設定在接收模式.
- 將 CLKKEEP (SC_PINCTL[6]) 置 1, 使能 SC_CLK 時鐘.
- 將 RSTSTS (SC_PINCTL[18]) 置 1, 釋放 SC_RST 到高電平

如下是硬體啟動模式下的啟動控制序列, 比較容易使用.

- 通過設置INITSEL (SC_ALTCTL[9:8]) 設置啟動時序.
- 當 TMRSEL (SC_CTL[14:13])為 01, 10 或or 11時，TMR0 可以被選擇.
- 設置操作模式 OPMODE (SC_TMRCTL0[27:24]) 為 0011，通過設置 CNT (SC_TMRCTL0[23:0] 寄存器設定請求應答的值.
- 將ACTEN (SC_ALTCTL[3]) 置 1. 硬體將開始自動執行啟動控制.
- 當硬體釋放 SC_RST 到高，硬體將同時產生一個中斷到 CPU, 並將INTIF (SC_INTSTS[8]) 置 1.
- 如果從 SC_RST拉高開始TMR0 減小計數值到 0, 而且在此之前卡沒有回應 ATR, 硬體將產生中斷到 CPU, 並將 TMR0IF (SC_INTSTS[3]) 置 1.
- 若是需要特加長T1 的時間, 可以透過控制 T1EXT(SC_ACTCTL[4:0]) 來達成.



Time	Comment
T1	SC_PWR to SC_CLK Start
T2	SC_CLK Start to SC_RST Assert
T3	SC_CLK Start to ATR Appear

INITSEL	T1	T2
00	85	489
01	133	537
10	165	569
11	165	42060

Unit: SC Clock

Note : The values are measured by chip I/O pin and the real value will depend on system design

圖 14-1 SC 啟動序列

14.4.2 暖復位 (Warm Reset)

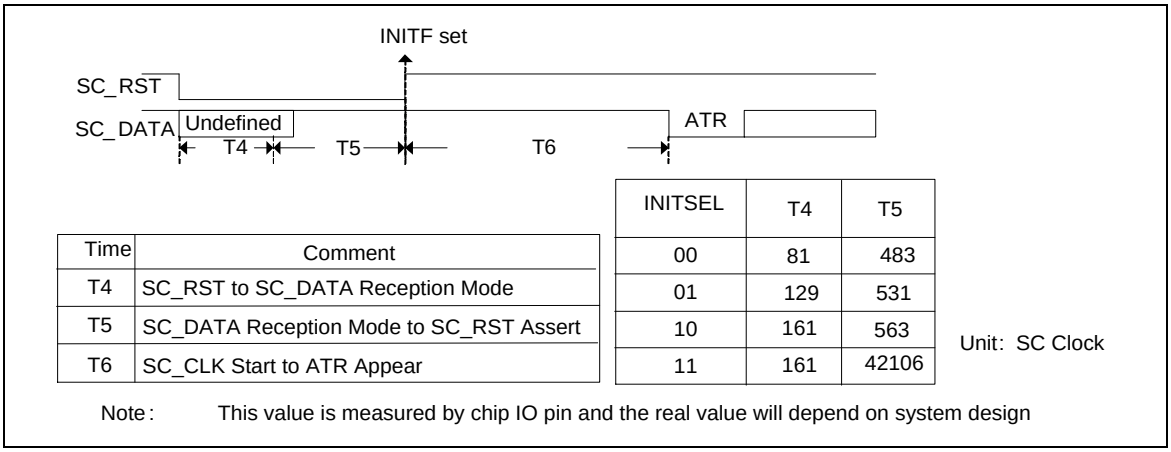
暖重定序列能夠被軟體和硬體控制, 如果軟體想控制該順序, 軟體能夠控制 SC_PINCTL寄存器去處理暖復位序列或者設置 WARSTEN (SC_ALTCTL[4]) 寄存器, 然後介面將執行硬體暖重定序列.

若是透過軟體模式, 操作流程如下:

- 將 RSTSTS (SC_PINCTL[18]) 清為0, 設置 SC_RST 為低
- 將 DATSTS (SC_PINCTL[16]) 置 1, 將 SC_DATA 設定在接收模式.
- 將 RSTSTS (SC_PINCTL[18]) 置 1, 釋放 SC_RST 到高電平

如下是硬體暖重定模式下的暖重定控制序列。

- 通過設置 INITSEL (SC_ALTCTL[9:8])設定暖復位時序
- 通過設置 TMRSEL (SC_CTL[14:13]) 寄存器（TMR_SEL 可以是 01, 10, 或者 11)選擇 TMR0
- 設定操作模式 OPMODE (SC_TMRCTL0[27:24]) 為 011，通過設置 CNT (SC_TMRCTL0[23:0]) 寄存器設定請求應答的值.
- 通過設置 SC_ALTCTL 寄存器設置 CNTEN0 (SC_ALTCTL[5]) 和 WARSTEN (SC_ALTCTL[4]) 開始計數
- 當硬體釋放 SC_RST 到高，硬體將同時產生中斷到 CPU INTIF (SC_INTSTS[8]) 並將 INTIF (SC_INTSTS[8]) 置 1.
- 如果從 SC_RST拉高開始, TMR0減小計數器到 0, 而且在此之前卡沒有回應 ATR, 硬體將產生中斷到 CPU, 並將 TMR0IF (SC_INTSTS[3]) 置 1.



Time	Comment
T4	SC_RST to SC_DATA Reception Mode
T5	SC_DATA Reception Mode to SC_RST Assert
T6	SC_CLK Start to ATR Appear

INITSEL	T4	T5
00	81	483
01	129	531
10	161	563
11	161	42106

Unit: SC Clock

Note: This value is measured by chip IO pin and the real value will depend on system design

圖 14-2 SC 暖復位序列

14.4.3 釋放 (Deactivation)

釋放序列可以由軟體一根腳位一根腳位控制, 或者硬體自動控制控制. 如果軟體想控制的話, 軟體能夠控制 SC_PINCSR 處理釋放序列, 釋放序列如下:

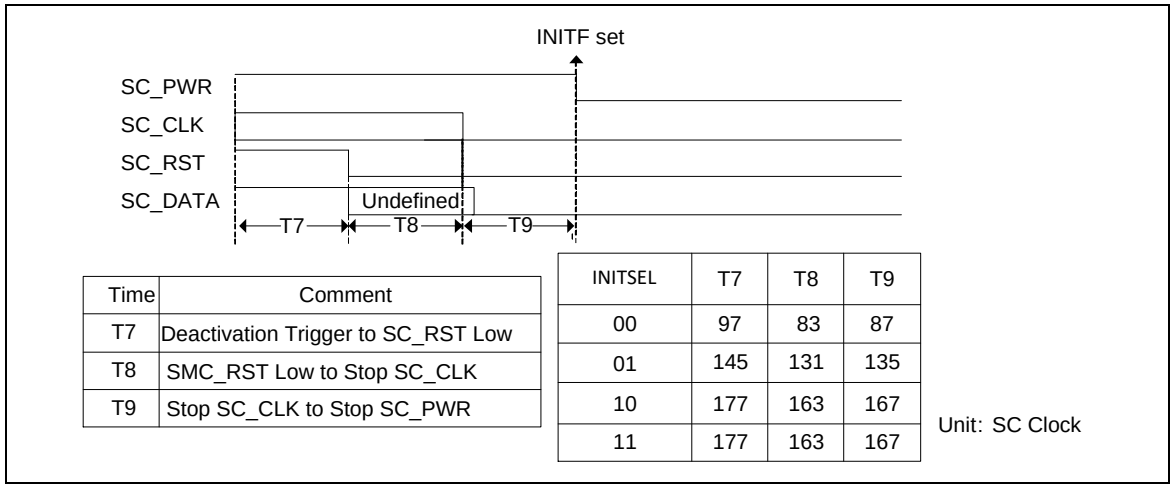
- 將 RSTSTS (SC_PINCTL[18]) 清為0, 設置 SC_RST 為低
- 將 CLKKEEP (SC_PINCTL[6]) 清為0, 停止 SC_CLK 時鐘輸出.
- 將 DATSTS (SC_PINCTL[16]) 清為0, 將 SC_DATA 設定在低電平.
- 將 PWRSTS (SC_PINCTL[18]) 清為0, 設置 SC_PWR 在低電平

如下是硬體釋放模式下的釋放控制序列:

- 通過設置 INITSEL (SC_ALTCTL[9:8])設置釋放時序

- 通過設置 DACTEN (SC_ALTCTL[2]) 為 1 釋放時序
- 當硬體釋放 SC_PWR 為低，控制器將同時產生一個中斷到 CPU, 並將 INTIF (SC_INTSTS[8]) 置 1.

當設置了卡移除檢測ADACEN (SC_ALTCTL[11])時, SC 控制器也支援偵測到卡移除時, 自動執行釋放序列.



Time	Comment
T7	Deactivation Trigger to SC_RST Low
T8	SMC_RST Low to Stop SC_CLK
T9	Stop SC_CLK to Stop SC_PWR

INITSEL	T7	T8	T9
00	97	83	87
01	145	131	135
10	177	163	167
11	177	163	167

Unit: SC Clock

圖 14-3 SC 釋放序列

14.4.4 資料格式

基本上, 智慧卡介面扮演的是一個半雙工非同步通訊連接埠的角色, 它的資料格式如下圖所示的10個連續位組成.

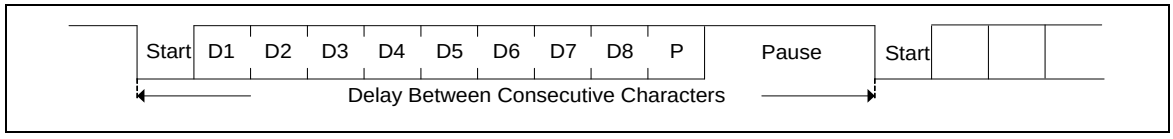


圖 14-4 SC 資料字元

依據 ISO 7816-3, 初始化 ATR 的字元 TS 有如下兩個可能的模式（如下圖所示）. 如果 TS 模式是0_1100_0000_1, 則是反向約定. 當按反向約定進行解碼後, 則傳送的位元組等於 0x3F, 如果 TS 模式是 0_1101_1100_1, 則是直接約定. 當按直接約定解碼後, 則傳送的位元組等於 0x3B. 軟體可以設置 AUTOCEN (SC_CTL[3]), 由硬體來決定操作的約定. 軟體也可以設置CONSEL (SC_CTL[5:4]) 寄存器（設為 00 或 11）, 在 SC 收到 ATR 的 TS 後去改變操作約定.

如果軟體通過設置 AUTOCEN 位使能自動約定功能, 則設置步驟必須在ATR狀態之前完成,而且第一個資料必須是 0x3B 或者 0x3F. 在硬體收到第一個資料並存放到緩存中之後, 硬體將決定約定模式並自動改變 CONSEL 位. 如果第一個資料既不是 0x3B 也不是且 ACERRIEN (SC_INTEN[10]) 被置1, 則硬體將產生一個中斷給 CPU, 並將 ACERRIF (SC_INTSTS[10]) 置1.

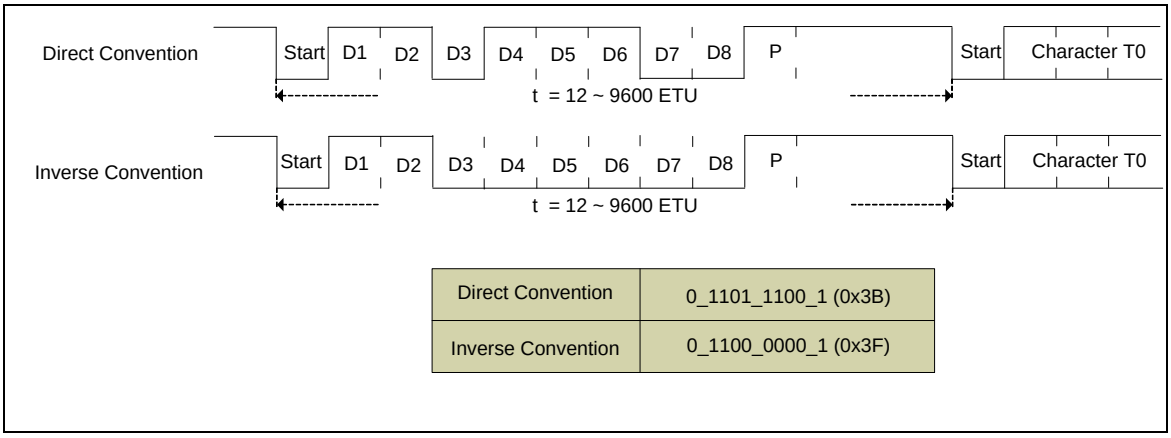


圖 14-5 初始化字元 TS

14.4.5 資料傳輸

資料的傳送與接收, 都是透過 SC_DAT 寄存器進行. 要送出時, 將要送出的字元寫入 SC_DAT, 要接收時, 則讀取 SC_DAT.

傳送與接收各有四個字節的 FIFO. 傳送時, 需要讀取 TXFULL (SC_STATUS[10]), 卻認為 0, 表示 TX FIFO 仍有空間, 才可寫入, 否則會發生傳送溢出錯誤, TXOV(SC_STATUS[8]) 會被置 1. 當有自料可接收時, RXEMPTY (SC_STATUS[1]) 會被清 0, 軟體可一直讀取 SC_DAT 直到RXEMPTY (SC_STATUS[1]) 會被置 1. RX FIFO 已滿, 軟體來不及讀取又有新資料進來, 則接收溢出錯誤, RXOV(SC_STATUS[0]) 會被置 1.

除了可使用輪詢模式外, 軟體也可透過中斷來得知傳送接收 FIFO 的狀態變化. 當 TBEIEN (SC_INTEN[1]) 設1, 當 TX FIFO 為空時, 會發生中斷, 且TBEIF (SC_INTSTS[1]) 被置 1, 此時可以一次對 TX FIFO 寫入最多四個字節. 直到下次中斷再寫入最多四個字節. 當 RDAIEN (SC_INTEN[0]) 設1, 當 RX FIFO 裡的資料不少於 RXTRGLV (SC_CTL[7:6]) 定義的觸發準位時, 會發生中斷, 且RDAIF (SC_INTSTS[0]) 被置 1, 軟體可一直讀取 SC_DAT 直到 RXEMPTY 被置 1. 為了避免 RX FIFO 裡的資料少於 RXTRGLV, 無法觸發中斷, 可以透過設置接收超時設置, 使得 RX FIFO 有資料, 且超過一段時間都沒有達到觸發準位時, 發生中斷通知軟體. 要使能此功能, 可透過 SC_RXTOUT 設置以 ETU 為單位的超時時間, 並將 RXTOIEN (SC_INTEN[9]) 設 1, 則當超時發生時, RXTOIF (SC_INTSTS[9]) 會被置 1, 通知 FIFO 中有資料可讀.

14.4.6 錯誤信號和字元重複

依據 ISO 7816-3 T=0 模式描述, 如果接收器端收到一個錯誤的校驗位元, 接收端必須拉低 SC_DATA 1 到 2 個位週期去通知發送端校驗錯誤. 然後發送端將重傳該字元. 智能卡介面控制器支援接收器硬體錯誤檢測功能和發送器硬體重傳功能. 軟體能通過設置TXRTYEN (SC_CTL[23]) 來使能重傳功能。軟體也能夠在 TXRTY (SC_CTL[22:20]) 寄存器中定義重傳的次數限制. 如果重傳的次數等於 TXRTY + 1, TXOVERR (SC_STATUS[30]) 標誌位將被置 1. 若是 TERRIEN (SC_INTEN [2]) 為 1, 則硬體將產生一個中斷給 CPU, 並將 TERRIF (SC_INTSTS[2]) 置1.

軟體也能通過設置 RXRTYEN (SC_CTL[19]) 以及 RXRTY(SC_CTL[18:16]) 來定義重傳的次數限

制. 如果重傳的次數等於 RXRTY + 1, RXOVERR (SC_STATUS[22]) 標誌位將被置 1. 若是 TERRIEN (SC_INTEN [2]) 為 1, 則硬體將產生一個中斷給 CPU, 並將 TERRIF (SC_INTSTS[2]) 置1.

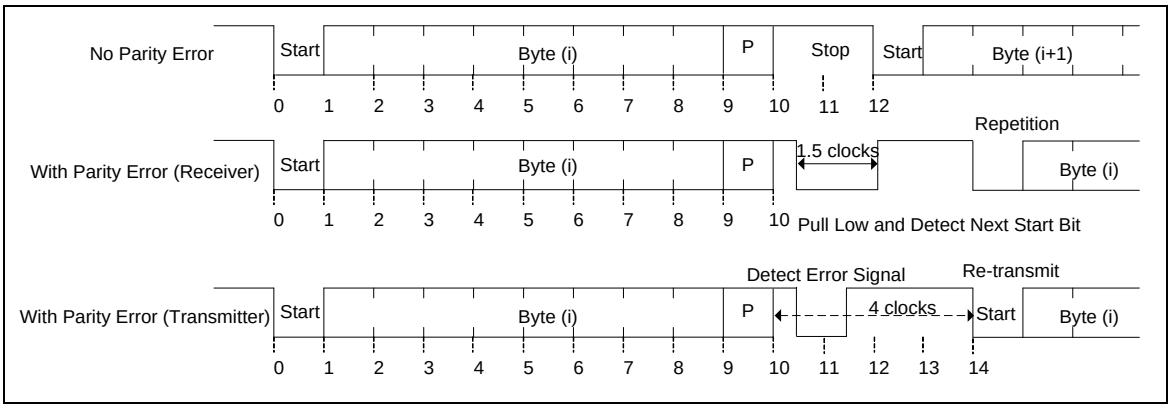


圖 14-6 SC 錯誤信號

在 T=1 模式下時, 資料校驗是透過上層傳輸協議的 R-Block 通知對方有傳輸錯誤發生, 而不是拉錯誤訊號, 所以當工作在 T=1 模式時, TXRTYEN 以及RXRTYEN 均需清為 0.

14.4.7 內部超時計數器

智慧卡介面包括一個24-位元超時計數器 (SC_TMR0)和兩個 8-位 超時計數器 (SC_TMR1, SC_TMR2), 這些計數器幫助控制器處理不同的即時間隔 (ATR, WWT, BWT, 等待). 每個計數器可以被設置成一旦觸發使能位被寫或者檢測到一個START 位就開始進行計數.

如下是程式設計流程：

- 通過設置 TMRSEL (SC_CTL[14:13])使能/禁用 計數器
- 通過設置SC_TMRx 寄存器選擇操作模式 OPMODE (SC_TMRCTLx[27:24]) 並設置一個計數值 CNT (SC_TMRCTLx[23:0])
- 設置 CNTEN0 (SC_ALTCTL[5]), CNTEN1 (SC_ALTCTL[6]) 或 CNTEN2 (SC_ALTCTL[7]) 開始進行計數.

OPMODE (SC_TMRCTLx[27:24]) (X=0 ~2)	操作描述	
0000	向下計數器開始於 CNTENx 使能, 結束于計數器超時, 超時值為 CNT+1.	
	開始	當 CNTENx 使能, 開始計數.
	結束	當向下計數器等於 “0”, 硬體將自動設置 TMRxIF (SC_INTSTS[5:3])

		並清除CNTENx.
0001	向下計數器開始於第一個 START 位被檢測到, 結束于計數器超時. 超時值為 CNT+1.	
	開始	在CNTENx 設為 1 後, 當檢測到第一個 START 位(接收或發送), 開始計數.
	結束	當向下計數器等於 0, 硬體將自動設置 TMR0_IS 並清除CNTENx.
0010	向下計數器開始於第一個 START 位（接收）被檢測到, 結束于計數器超時出現. 超時值為 CNT+1.	
	開始	在CNTENx 設為 1 後，當檢測到第一個 START 位(接收), 開始計數.
	結束	當向下計數器等於 0, 硬體將自動設置 TMRxIF 並清除CNTENx.
0011	向下計數器只用於硬體啟動, 暖重定序列來測量 ATR 時序. 時間開始於SC_RST 釋放, 結束於 ATR 應答收到或超時. 如果在 ATR 應答收到之前, 計數器減到 0, 則硬體將產生一個中斷給CPU. 超時值為 CNT+1. 注意: 只有 SC_TMRCTL0 支持本模式.	
	開始	在CNTENx 設為 1後, 當SC_RST 釋放, 開始計數. 用於硬體啟動, 暖重定模式.
	結束	如果在 ATR 應答收到之前, 計數器減到 0, 則硬體將自動設置 TMRxIF 並清除 CNTENx. 當 ATR 接收到而且向下計數器沒有等於 0, 硬體將自動清除 CNTENx.
0100	和模式 0000 一樣, 但是當向下計數器等於 0 時, 硬體將設置 TMRxIF 而且計數器將重載 CNT 的值, 並重新計數直到軟體清除 CNTENx. 當 ACTSTSx (SC_ALTCTL[15:13]) 為 1, 軟體能在任何時候改變 CNT 的值. 當向下計數器等於 0, 計數器將重載 CNT 的新值並重新計數. 超時值為 CNT+1.	
0101	和模式 0001 一樣, 但是當向下計數器等於 0 時, 硬體將設置 TMRx_IF 而且計數器將重載 CNT 的值. 當檢測到下一個 START 位，計數器將重新計數直到軟體清除 CNTENx. 當 ACTSTSx為 1, 軟體能在任何時候改變CNT 的值。當向下計數器等於 0, 計數器將重載 CNT 的新值並重新計數. 超時值為 CNT+1.	
0110	和模式 0010 一樣，但是當向下計數器等於 0 時, 硬體將設置 TMRx_IF 而且計	

	數器將重載 CNT 的值。當檢測到下一個 START 位，計數器將重新計數直到軟體清除 CNTENx. 當 ACTSTSx為 1, 軟體能在任何時候改變 CNT的值. 當向下計數器等於 0, 計數器將重載 CNT 的新值並重新計數. 超時值為 CNT+1.	
0111	向下計數器開始於第一個 START 位被檢測到, 結束於軟體清除CNTENx 位. 如果下一個START 位被檢測到, 計數器將重載 CNT 的新值並重新計數. 如果在下一個 START 位被檢測到之前, 計數器減到 0, 則硬體將產生一個中斷給CPU. 超時值為 CNT+1.	
	開始	在CNTENx 設為 1 後, 當檢測到第一個 START 位, 開始計數.
	結束	CNTENx被設為 0 後, 停止計數.
1111	向下計數器開始於CNTENx 使能或是第一個 START 位被檢測到, 結束於軟體清除CNTENx 位. 如果下一個START 位被檢測到, 計數器將重載 CNT 的新值並重新計數. 如果在下一個 START 位被檢測到之前, 計數器減到 0, 則硬體將產生一個中斷給CPU. 超時值為 CNT+1.	
	開始	在CNTENx 設為 1 後, 或當檢測到第一個 START 位, 開始計數
	結束	CNTENx被設為 0 後, 停止計數.

14.4.8 智能卡插拔偵測

智能卡接口可以偵測卡的插拔狀態. 但要能正確偵測, 首先需要設置偵測電平 CDLV (SC_CTL[26]), 當此位為 1, 帶表SC_CD 高電平為卡插入, 低電平為卡移除. 當此位為 0, 帶表 SC_CD 高電平為卡移除, 低電平為卡插入. 這位的設置跟卡槽設計有關. 請查詢使用卡槽的規格書設置此位. 智能卡接口同時支持了插拔偵測的去抖動功能. 共有四級可以透過 CDDBSEL (SC_CTL[25:24]) 來設置.

SC_CD 的當前狀態可以透過輪詢 CDPINSTS(SC_STATUS), 這位直接反映當前 SC_CD 的電平, 跟 CDLV 設置無關. 平時使用時, 則可透過中斷來提示智能卡插拔狀態的改變. 當 CDIEN (SC_INTEN[7]) 設 1 後, 每當插拔狀態產生改變, 則硬體將產生一個中斷給 CPU, 並將 CDIF (SC_INTSTS[7]) 置1. 之後, 軟體可透過查詢 CINSERT (SC_STATUS[12]) 以及 CREMOVE (SC_STATUS[11]) 得知卡的當前狀態. CDIF, CINSERT, 以及 CREMOVE 都可透過寫 1的方式清 0.

14.4.9 其他傳輸 相關設置

在此介紹一些傳輸相關的設置

- ETU 設置

ETU 是智能卡傳輸的基本時間單位, 缺省值是 372 個時鐘長度. 當經過 PPS 交換協議後, 可透過設置 ETURDIV (SC_ETUCTL[11:0]) 更改為其他值. 實際的 ETU 會是 ETURDIV + 1 個時鐘.

- 停止位設置

讀取 ATR, 或是工作在 T=0 模式時, NSB(SC_CTL[15]) 需清 0, 設定兩個停止位. 當工作在 T=1模式時, NSB(SC_CTL[15]) 需置 1 , 設定一個停止位

- 塊保護時間 (BGT)

當工作在 T=1模式時根據 ISO 7816-3 規範, 不同方向傳輸的開始位最小間隔, 即塊保護時間, 為 22 個 ETU. 塊保護時間可透過 BGT (SC_CTL[12:8]) 設置. 若是智能卡沒有遵循塊保護時間的規範, 提早發送, 且 BGTIEN (SC_INTEN[6]) 為 1, 則硬體將產生一個中斷給 CPU, 並將 BGTIF (SC_INTSTS[6]) 置1. BGTIF 標誌位可透過寫 1 的方式清除.

- 擴展保護時間 (Extra Guard Time)

根據 ISO 7816-3 規範, 當 TC_1 存在於 ATR, 且不為 255 時, 保護時間為 12ETU + F/D * N / f = (12 + N). 這裡的 N 值, 即為擴展保護時間. 擴展保護時間可透過 SC_EGT 寄存器控制.

14.4.10 串口(UART) 模式

若是系統中的串口不敷使用, 且並無使用智能卡的需求, 可以將 UARTEN (SC_UARTCTL[0]) 置 1, 將智能卡介面控制器作為一個基本的 UART 功能來使用, 此時不具有自動流控制. 在串口模式下, 智能介面資料(SCx_DATA)管腳將會作為UART RXD, 而智慧介面時鐘(SCx_CLK)管腳將會作為UART TXD. 如下是串口模式下的程式設計示例:

1. 軟體可以通過設定 UARTEN (SC_UARTCTL[0]) 位元來進入 UART 模式.
2. 通過設定 RXRST (SC_ALTCTL[1]) 和 TXRST (SC_ALTCTL[0]) 位元進行軟體重定, 以確保所有的狀態機處於空閒狀態.
3. 填充 0 到 CONSEL (SC_CTL[5:4]) 和 AUTOCEN (SC_CTL[3]) 域. (當工作在 UART 模式時，這些位元必須為 “0”).
4. 通過設定 ETURDIV (SC_ETUCR[11:0])來選擇 UART 串列傳輸速率.
   - 串列傳輸速率 = f / (ETURDIV +1) , 此時 f 為智慧卡控制器介面運行時鐘頻率 (SC_CLK), 有效的ETURDIV 的有效值介於0x04與0xFFF之間, 當其值小於0x04將會被視為0x04.
   - 例如當使用者欲設定串列傳輸速率115200, 智慧卡時鐘頻率為12MHz, ETURDIV應設為 0x67其錯誤率大約為0.16%.
5. 選擇資料格式, 包括資料長度 (通過設定 WLS (SC_UARTCTL[5:4]), 校驗位格式 (通過設定 OPE (SC_UARTCTL[7]) 和 PBOFF (SC_UARTCTL[6]) 位) 和停止位長度 (通過設定 NSB (SC_CTL[15])).
6. 通過設定 RXTRGLV (SC_CTL[7:6]) 域來選擇接收器緩存觸發水準, 並通過設定 RFTM (SC_RXTOUT[8:0]) 域來選擇接收器緩存超時間隔.

7. 寫 SC_DAT (SC_DAT[7:0]) 寄存器或者讀 SC_DAT (SC_DAT[7:0]) 寄存器可以執行 UART 功能.

14.5 寄存器

Register	Offset	R/W	Description	Reset Value
SC Base Address: **SC0_BA = 0xB009_0000** **SC1_BA = 0xB009_1000**				
SC_DAT **x = 0,1**	SCx_BA+0x00	R/W	SC Receiving/Transmit Holding Buffer Register	0xXXXX_XXXX
SC_CTL **x = 0,1**	SCx_BA+0x04	R/W	SC Control Register	0x0000_0000
SC_ALTCTL **x = 0,1**	SCx_BA+0x08	R/W	SC Alternate Control Register	0x0000_0000
SC_EGT **x = 0,1**	SCx_BA+0x0C	R/W	SC Extend Guard Time Register	0x0000_0000
SC_RXTOUT **x = 0,1**	SCx_BA+0x10	R/W	SC Receive Buffer Time-out Register	0x0000_0000
SC_ETUCTL **x = 0,1**	SCx_BA+0x14	R/W	SC ETU Control Register	0x0000_0173
SC_INTEN **x = 0,1**	SCx_BA+0x18	R/W	SC Interrupt Enable Control Register	0x0000_0000
SC_INTSTS **x = 0,1**	SCx_BA+0x1C	R/W	SC Interrupt Status Register	0x0000_0002
SC_STATUS **x = 0,1**	SCx_BA+0x20	R/W	SC Status Register	0x0000_0202
SC_PINCTL **x = 0,1**	SCx_BA+0x24	R/W	SC Pin Control State Register	0x0000_00x0
SC_TMRCTL0 **x = 0,1**	SCx_BA+0x28	R/W	SC Internal Timer Control Register 0	0x0000_0000
SC_TMRCTL1 **x = 0,1**	SCx_BA+0x2C	R/W	SC Internal Timer Control Register 1	0x0000_0000
SC_TMRCTL2 **x = 0,1**	SCx_BA+0x30	R/W	SC Internal Timer Control Register 2	0x0000_0000
SC_UARTCTL **x = 0,1**	SCx_BA+0x34	R/W	SC UART Mode Control Register	0x0000_0000
SC_ACTCTL **x = 0,1**	SCx_BA+0x4C	R/W	SC Activation Control Register	0x0000_0000

15 I²C 序列介面控制器 (I²C)

15.1 概述

I2C 為雙線，雙向串列匯流排，通過簡單有效的連線方式實現器件間的資料交換。I2C標準是多主機匯流排，包括衝突檢測和仲裁，以防止在兩個或多個主機同時嘗試控制匯流排時發生資料損壞。

有四組I2C控制器，都支援掉電喚醒功能。

15.2 特性

- 支援最多四個I²C介面
- 支援主機/從機 模式
- 主從機之間雙向資料傳輸
- 匯流排支援多主機 (無中心主機)
- 多主機間同時傳輸資料仲裁，避免匯流排上串列資料損壞
- 匯流排採用串列同步時鐘，可實現設備之間以不同的速率傳輸
- 內建14位溢出計時器，當I2C匯流排中止且計時器溢出，產生I2C中斷
- 可配置不同時鐘以適用於可變速率控制
- 支援7位元及10位元從機位址模式
- I²C 匯流排控制器支援多位址識別（4組從機位址帶mask 選項）
- 支援喚醒功能

15.3 方塊圖

I2C控制器的框圖如下：

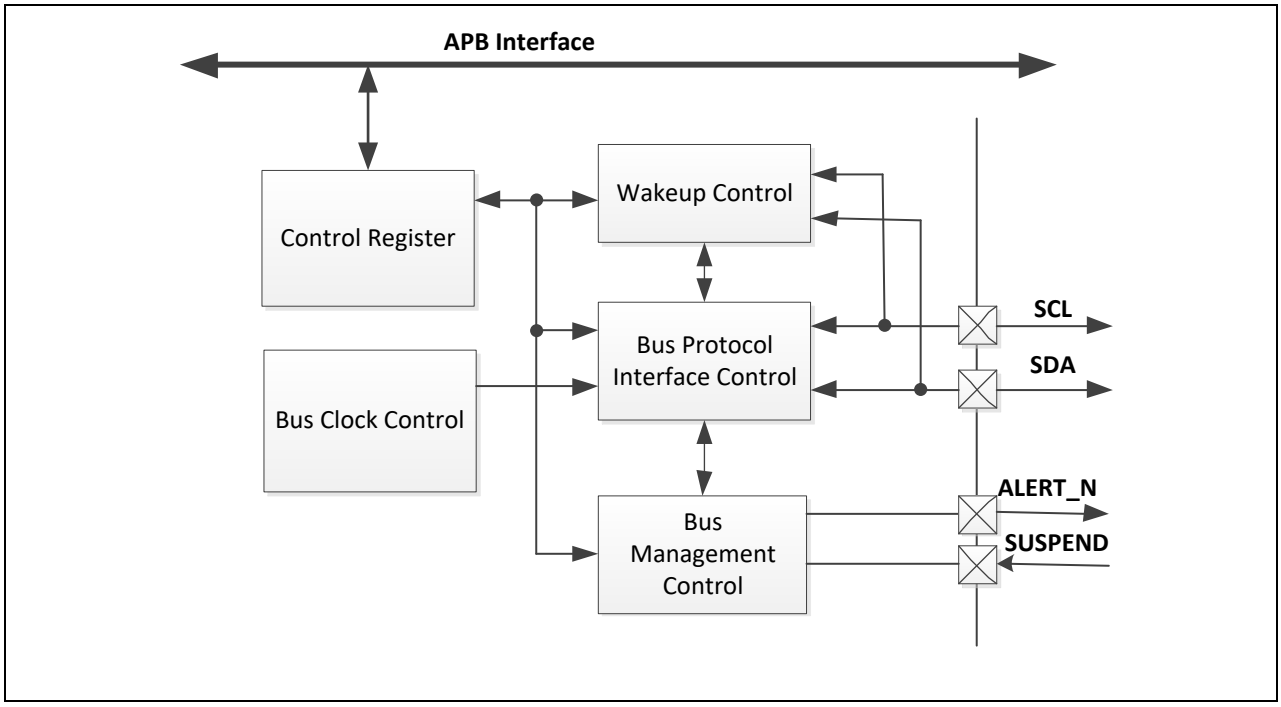
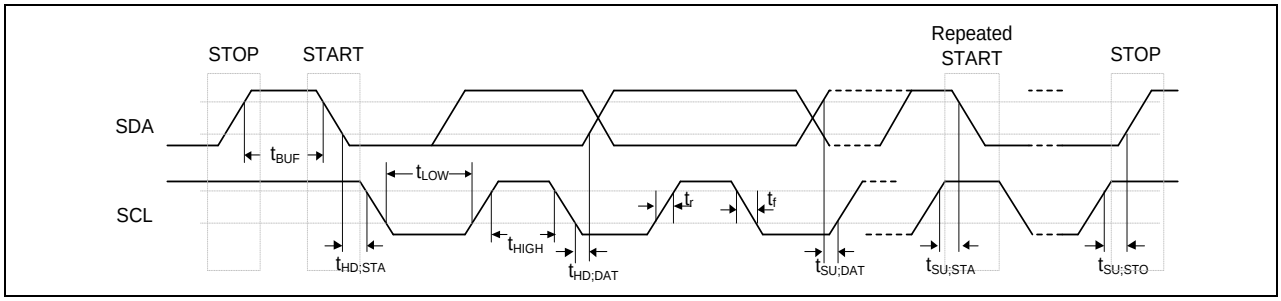


圖 15.3−1 I²C控制器框圖

15.4 I²C 功能描述

在I2C匯流排上，資料通過時鐘線SCL和資料線SDA在主從機間逐一位元組同步傳送。每個位元組資料長度是8位元. 一個SCL 時鐘脈衝傳輸一個資料位元，資料由最高位元MSB 開始傳輸，每個位元組傳輸後跟隨一個應答位，每個位在SCL 為高時採樣；因此，SDA 線可能在SCL 為低時改變，但在SCL 為高時 必須保持穩定。當SCL 為高時，SDA 線上的跳變視為一個命令(START or STOP)。更多關於I2C匯流排時序的細節請參考 下圖。



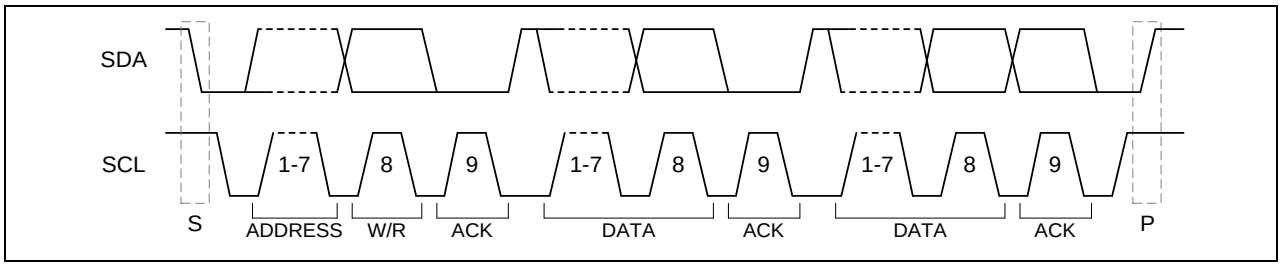
片上I2C外設提供了一個符合I2C匯流排規範的序列介面。I2C埠自動處理位元組傳輸。通過設置寄存器I2C_CTL的I2CEN為'1'可使能該埠。I2C 硬體介面通過資料線SDA 和時鐘線 SCL兩個管腳連到I2C匯流排。當I/O 管腳作為I2C 埠使用時，用戶必須事先設定I/O 管腳為I2C功能。

注意： SDA 和 SCL兩個管腳需要上拉電阻，因為這個兩個管腳為開漏腳.

15.4.1.1 協議

標準I2C 協定如下圖，通常標準通訊有以下4部分:

- 起始信號(START) 或者重複起始信號(Repeated START)
- 從機位址傳輸和R/W 位傳輸
- 資料傳輸
- 停止信號(STOP)



15.4.1.2 工作模式

片上I2C埠支援三種工作模式：主機模式，從機模式和廣播呼叫模式。

應用中，I2C 埠可以作為主機和從機。在從機模式，I2C埠硬體會查找自身從機位址和廣播呼叫位址，如果這兩個位址的任一個被檢測到，並且從機想要從主機接收或向主機發送資料(通過設置AA位元)，應答脈衝將會在第9個時鐘發出，此時，如果中斷使能，則主機和從機設備上都會發生一次插斷要求。在微控制器想要成為匯流排主機時，在進入主機模式之前，硬體需等待到匯流排空閒，以保證合理的從機動作不會被打斷。在主機模式下，如果匯流排仲裁丟失，I2C埠立即切換到從機模式，並可以在同一次序列傳輸過程中檢測自身的從機位址。

為控制I2C匯流排的各種模式傳輸，使用者需要按照寄存器I2C_STATUS的當前狀態碼來設置I2C_CTL, I2C_DAT寄存器。換句話說，每個I2C匯流排動作都要檢查I2C_STATUS寄存器的當前狀態，然後再設置I2C_CTL, I2C_DAT寄存器執行匯流排動作。最後，通過I2CSTATUS檢查回應狀態。

在寄存器I2C_CTL [3]的SI標誌清除後，寄存器I2C_CTL的這些位STA, STO 和 AA 用來控制I2C硬體的下一個狀態。當完成一個新的動作，I2C_STATUS的狀態碼將被更新，I2C_CTL 寄存器的SI標誌將被設置。如果I2C中斷控制位INTEN (I2C_CTL [7])被設置，新狀態碼對應的動作或者軟體將在中斷服務程式中被執行。

下圖顯示當前I2C狀態碼是0x08，然後通過設置I2C_DATA=SLA+W和(STA,STO,SI,AA) = (0,0,1,x)發送位址到I2C匯流排。如果匯流排上有從機匹配相應的位址且返回ACK，I2C_STATUS狀態碼更新為0x18。

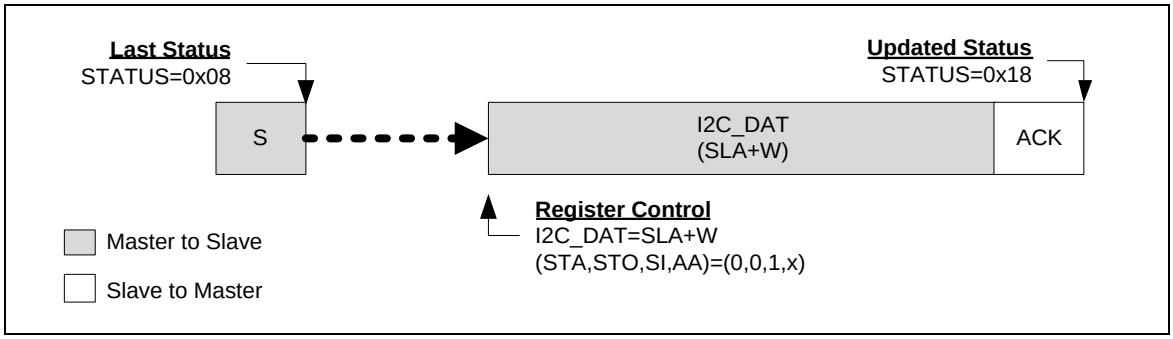


圖 15.4-1 通過當前狀態控制I²C 匯流排

主機模式

下圖中，是I2C主機模式所有可能的協定。使用者需要遵循恰當的流程來實現I2C協議。

換句話說，使用者可以發送一個起始信號到匯流排，I2C匯流排將設置成主機傳輸模式或主機接收模式。起始信號設置成功後新的狀態碼將是0x08。起始信號後，使用者可以發送從機位址，讀/寫位元，資料和重複起始，停止來執行I2C協議。

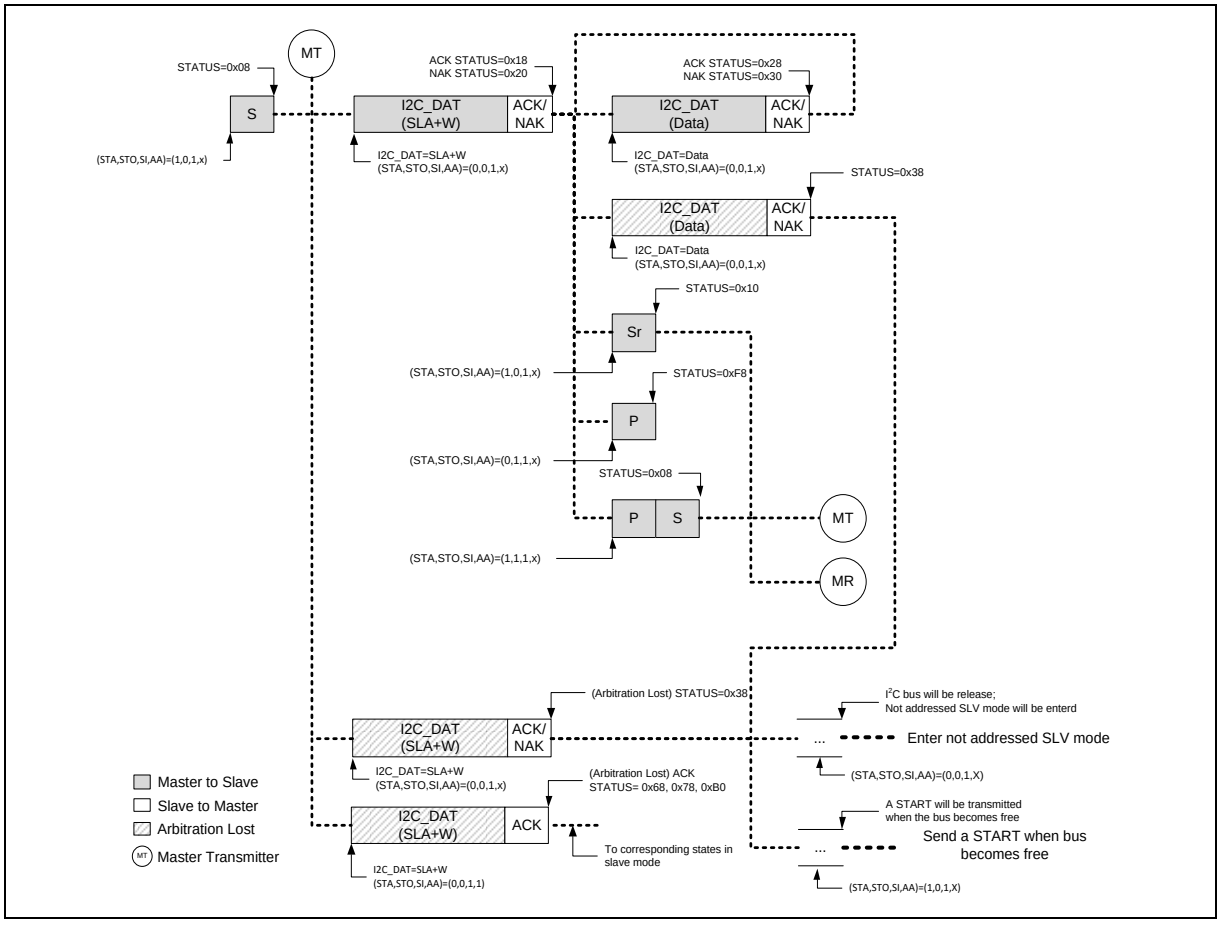


Figure 15.4-2主機發送模式控制流程

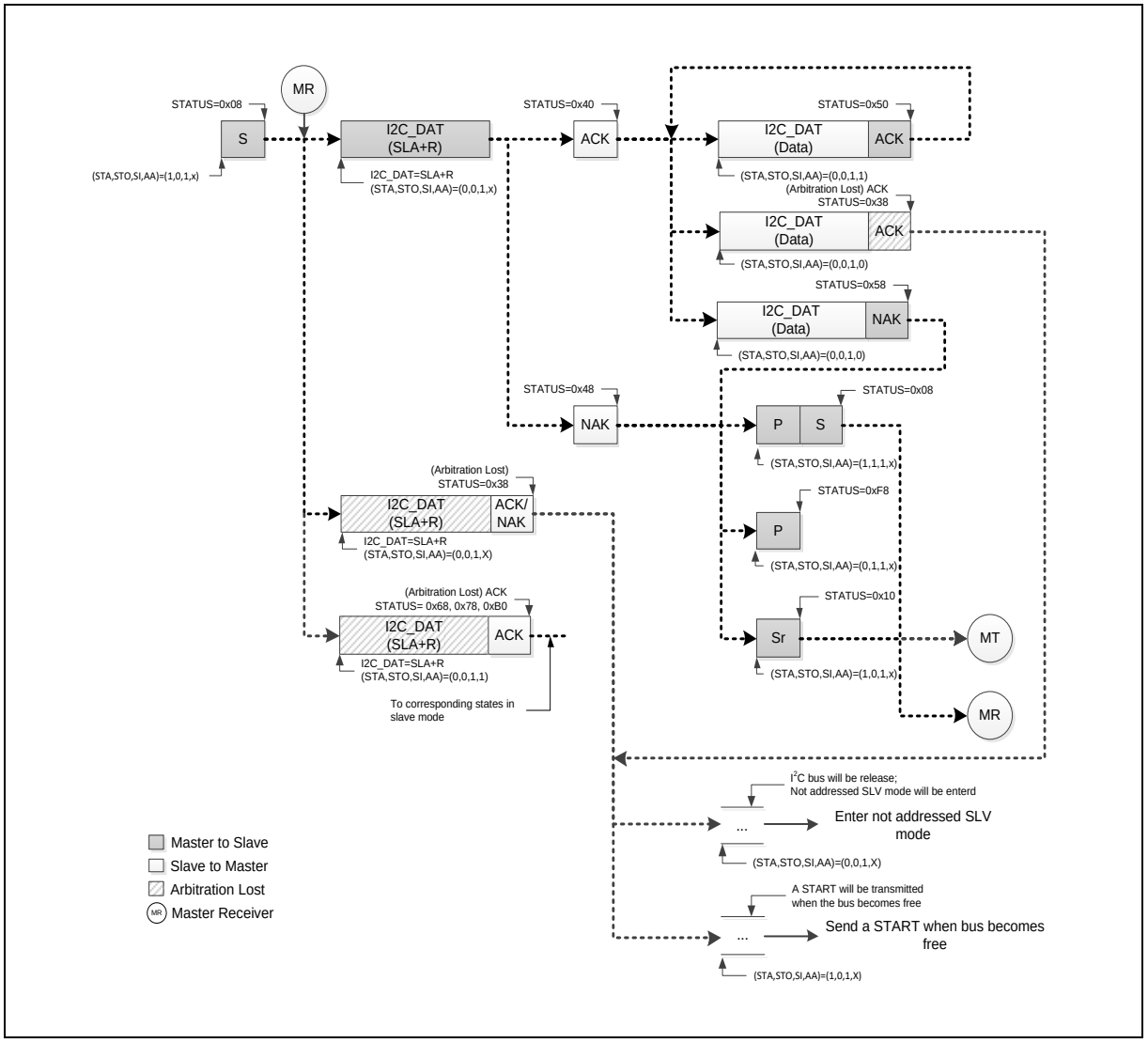


Figure 15.4−3 主機接收模式控制流程

如果I2C在主機模式並且仲裁丟失，狀態碼將置為0x38。在狀態為0x38時，當匯流排空閒時，使用者可以設置(STA, STO, SI, AA) = (1, 0, 1, X)發送起始信號來重新開始主機操作。否則，用戶可以設置(STA, STO, SI, AA) = (0, 0, 1, X)來釋放匯流排，並進入無位元元址從機模式。

從機模式

重定後預設情況下，I2C不會被定址，並且不會識別I2C匯流排上的位址。用戶可以通過I2C_ADDRn(n=0~3)設置從機位址和設置 (STA, STO, SI, AA) = (0, 0, 1, 1)來讓I2C識別主機發送的位址。圖6.15-12所示為I2C從機模式的所有可能的流程。使用者需要遵循下圖的流程來實現他們的I2C協議。

如果在主機模式匯流排仲裁丟失，I2C埠立即切換到從機模式，並且在同一序列傳輸中識別自有的從機位址。如果在仲裁丟失後識別到位址是SLA+W（主機想寫資料到從機），狀態碼是0x68。如果在仲裁丟失後識別到位址是SLA+R（主機向從機讀數據），狀態碼是0xB0。

注意：I2C通信期間，在從機模式下，當對SI標誌寫‘1’清除時，SCL 時鐘將被釋放。

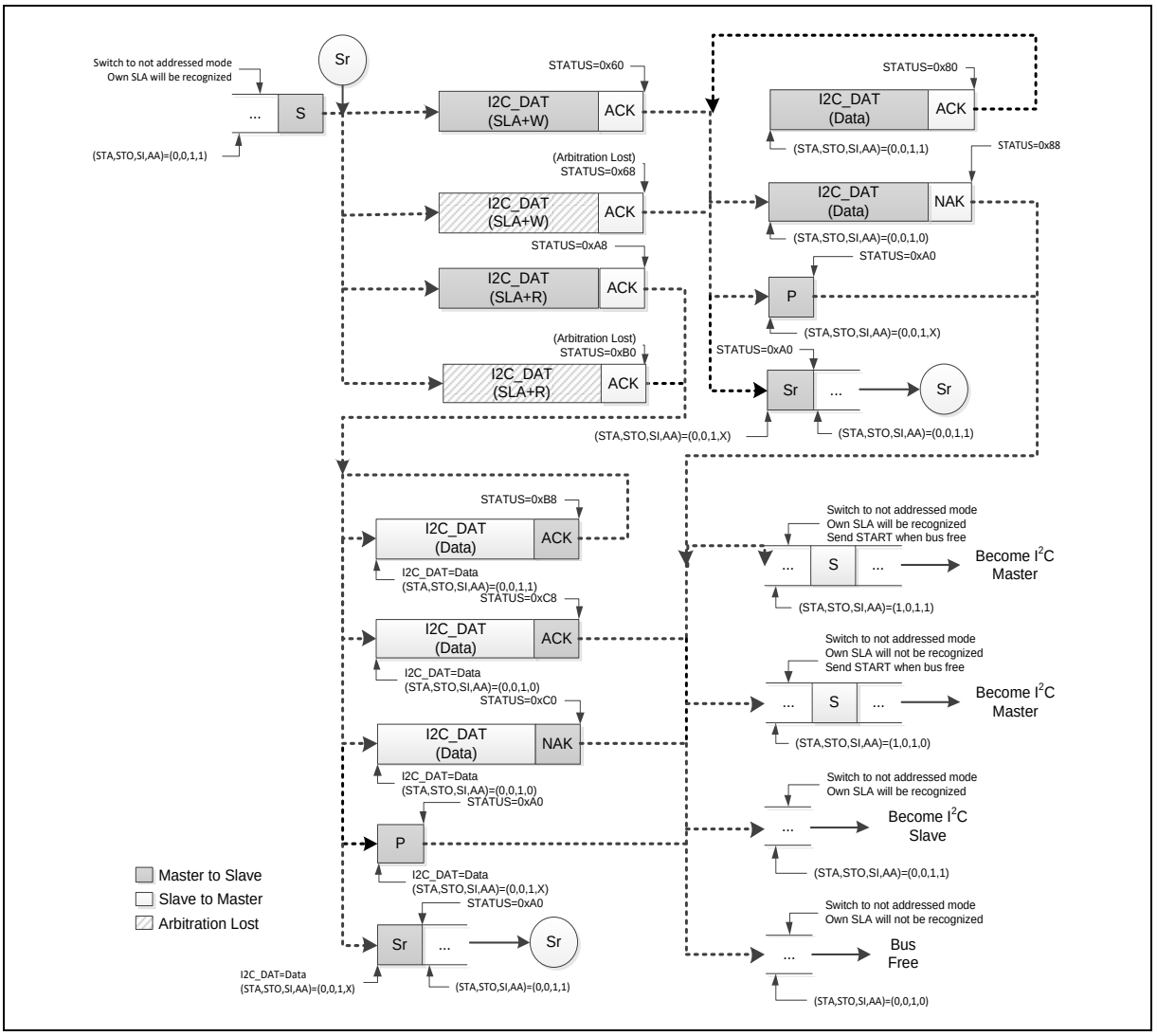


Figure 15.4−4 從機模式控制流程

如果I2C在可定址從機模式接收資料時，收到停止或重複起始信號，狀態碼是0xA0。當狀態碼是0xA0時，使用者可以遵循如上圖狀態碼是0x88的操作。

如果I2C在可定址從機模式傳輸資料時卻收到停止或重複起始信號，狀態碼是0xA0。當狀態碼是0xA0時，使用者可以遵循如上圖狀態碼是0xC8的操作。

注意：從機獲得0x88, 0xC8, 0xC0 和0xA0狀態後,從機可以切換到無位元元址模式，自身SLA不會被辨識。如果進入這種狀態，從機不再接收主機任何信號或位址。在這種狀態，需要重定才能離開這種狀態。

廣播呼叫模式 (GC)

如果 GC位(I2C_ADDRn [0]) 被設, I^{2}C埠硬體將回應廣播呼叫位址(0x00).用戶可以通過清GC位來禁止廣播呼叫功能。當I^{2}C在從機模式時且GC 位元元元被設置，可以接收主機位址(0x00)的廣播呼叫，將遵循廣播模式狀態。

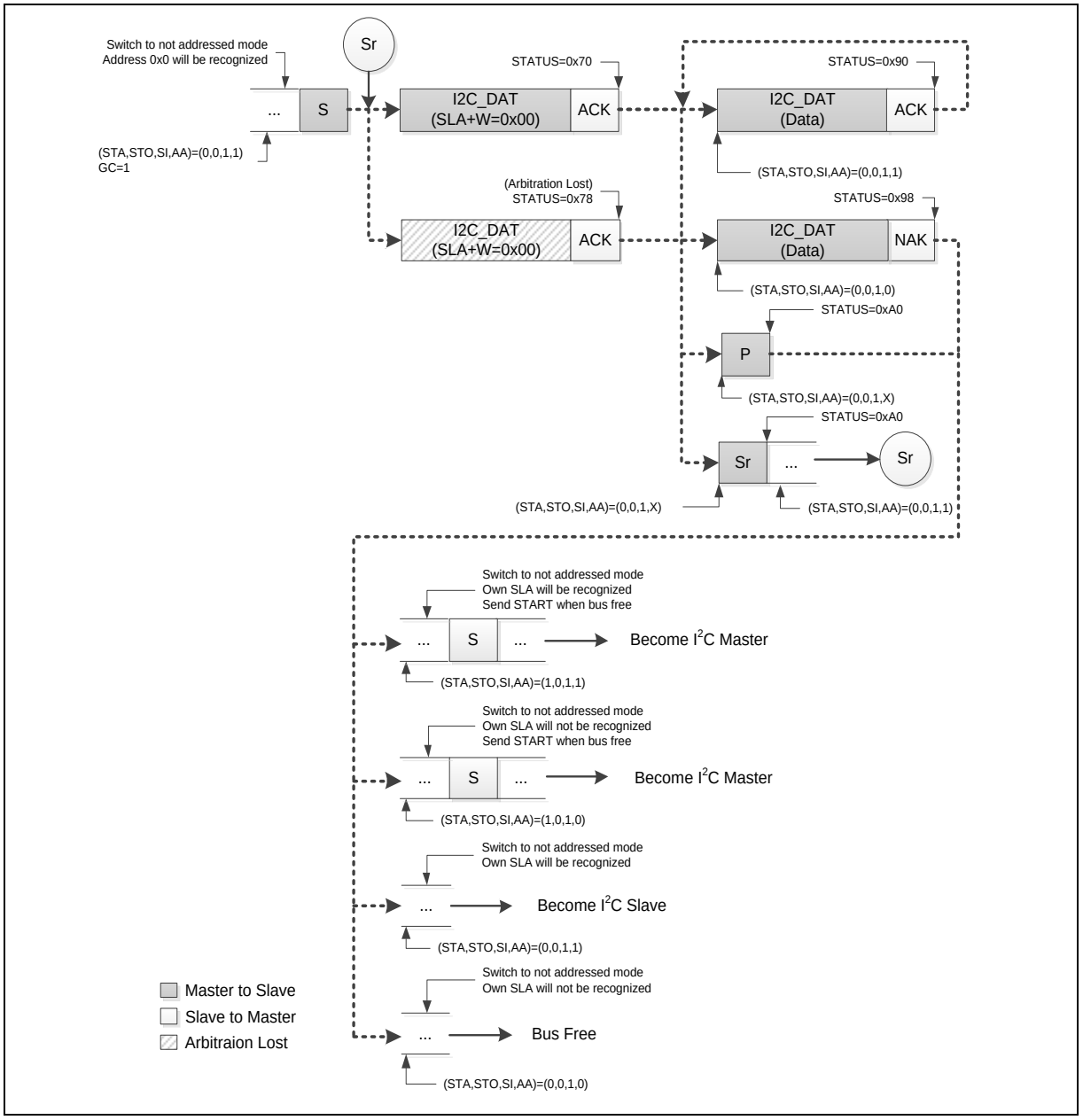


Figure 15.4−5 廣播呼叫模式

如果I2C在廣播呼叫模式接收資料時，卻收到停止或重複起始信號，狀態碼是0xA0。當狀態碼是0xA0時，使用者可以遵循如上圖狀態碼是0x98的流程處理。

注意：從機獲得0x98 和0xA0 狀態後，從機會切換到無位元元址模式並且自身SLA將不被辨識。如果進入這種狀態，從機不再接收主機的任何信號或位址。需要重定才能離開這種狀態。

多主機模式

在一些應用中，一個I2C匯流排上有多個主機同時訪問從機，並有可能同時在傳送資料。I2C是支援多主機模式，並包含有衝突檢測和仲裁，防止資料損壞。

如果兩個主機同時發命令，通過仲裁來決定哪個優先並繼續發命令。仲裁是在SCL為高時在SDA上執行的。每一個主機都會檢測匯流排上的SDA信號是否符合它產生的SDA信號。如果檢測到匯流排上SDA為低，但應該為高，則這個主機將失去仲裁。設備在仲裁丟失後會產生SCL脈衝直到本位元組結束，然後釋放匯流排進入從機模式。仲裁可以一直進行到所有資料被傳輸完。這樣意味著多主機系統中主機必須監控匯流排和做相應的處理。

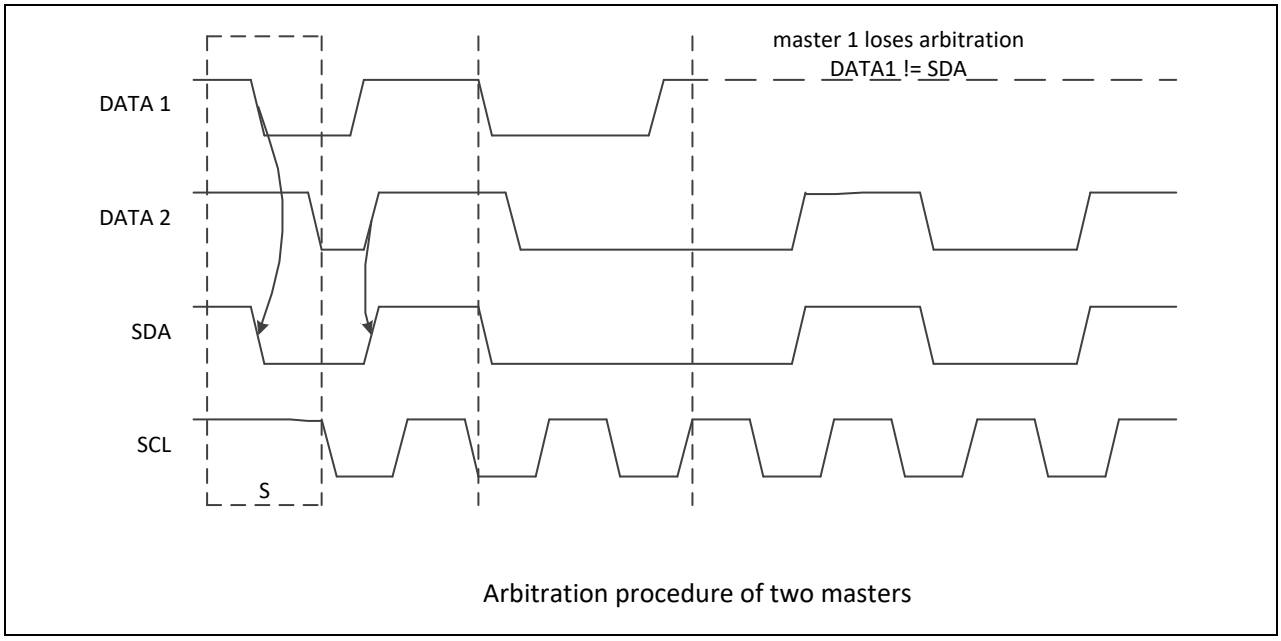


圖 15.4-6 仲裁丟失

- 當I2C_STATUS = 0x38代表接收到一個仲裁丟失。仲裁丟失事件可能發生在發送起始位元，資料位元或者停止位元期間。使用者可以在匯流排空閒時設置(STA, STO, SI, AA) = (1, 0, 1, X)來再次發送起始信號或者設置(STA, STO, SI, AA) = (0, 0, 1, X)發送停止信號來返回到無位址從機模式。

- 當I2C_STATUS = 0x00，接收到一個“匯流排錯誤”，為了從錯誤匯流排恢復到I2C匯流排，STO應被設置且SI應將被清0，然後對STO清0來釋放匯流排。

  - 設置(STA, STO, SI, AA) = (0, 1, 1, X)停止當前傳輸。

  - 設置(STA, STO, SI, AA) = (0, 0, 1, X)釋放匯流排。

15.4.1.3 EEPROM隨機讀取例子

通過下面的步驟來配置I2C0相關寄存器，來使用I2C從EEPROM讀取資料。

1. 在system寄存器中將多功能管腳設置成SCL和SDA管腳。
2. 通過設置寄存器CLK_APBCLK0的I2C0CKEN為1使能I2C0 APB時鐘。
3. 通過設置I2C0RST = 1來復位I2C0控制器，然後通過I2C0RST= 0將I2C控制器設置成普通操

作。I2C0RST位在SYS_IPRST1寄存器中。

4. 通過設置寄存器I2C_CTL中的I2CEN為1使能I2C0控制器。
5. 通過在I2C_CLKDIV寄存器寫I2C0時鐘分頻值。
6. 設置I2C0 IRQ.
7. 設置寄存器I2C_CTL的INTEN為1使能I2C0中斷
8. 設置I2C0位址寄存器“I2C_ADDR0~I2C_ADDR3”。

隨機讀操作是訪問EEPROM的其中一種方法。這個方法是允許主機訪問EEPROM的任何一個位址。下圖顯示對EEPROM隨機讀取操作。

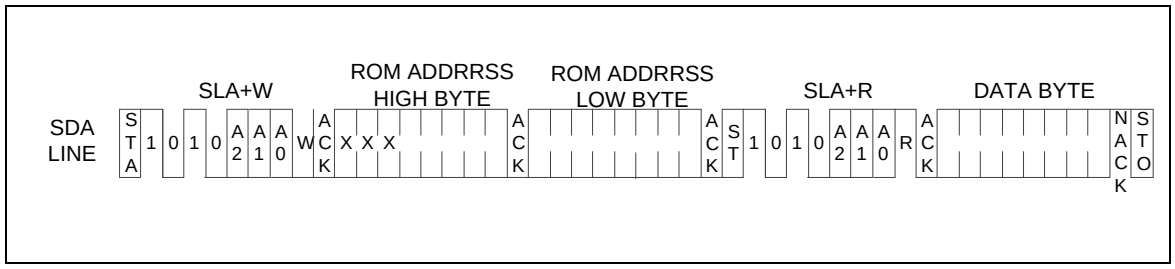


圖 15.4−7 EEPROM 隨機讀取

下圖顯示怎樣使用I2C控制器執行EEPROM隨機讀取操作協議

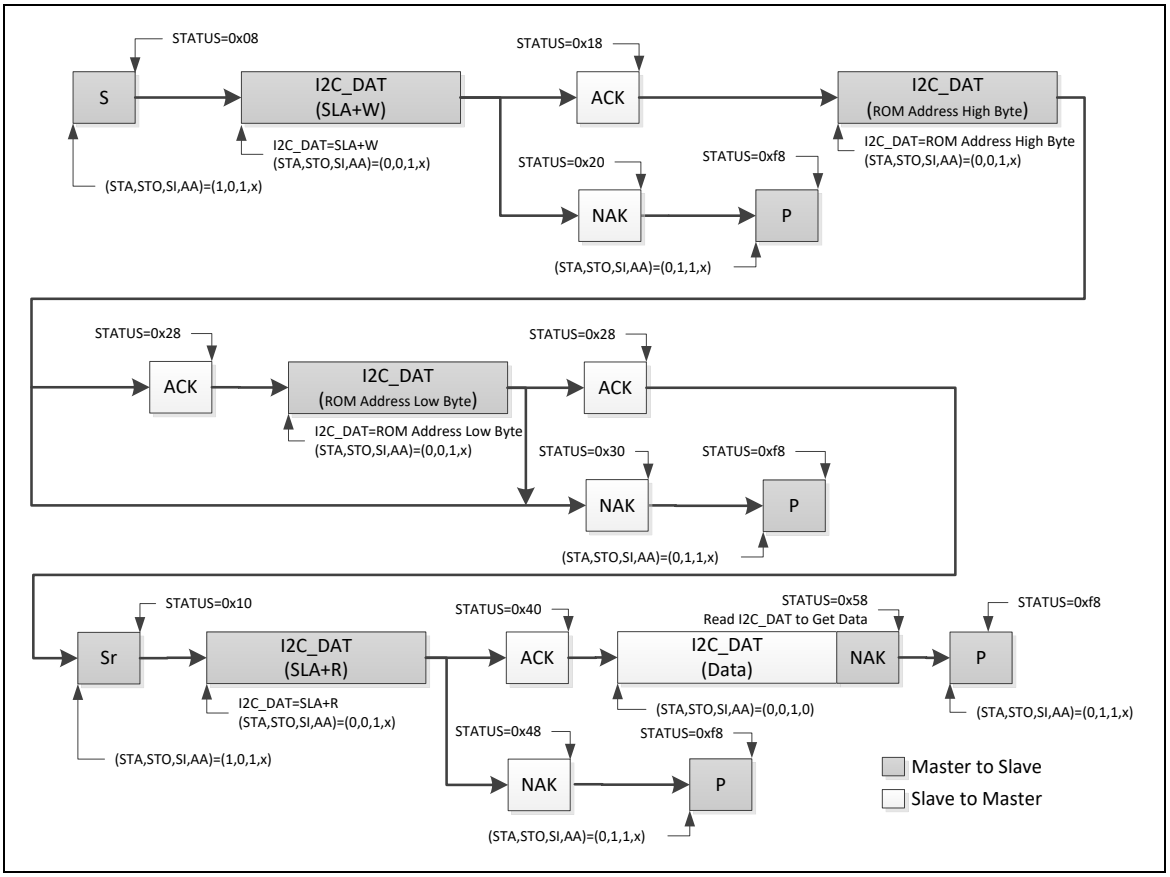


圖 15.4−8 EEPROM 隨機讀取協議

I2C控制器作為主機發送起始信號到匯流排，然後發送SLA+W (從機位址+寫位)到EERPOM，跟著由兩個位元組資料位址來設置EEPROM被讀的位址。最後，重複起始信號跟著SLA+R被發送來向EEPROM 讀取資料。

15.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
I²C Base Address: **I2Cn_BA = 0xB008_0000 + (0x1000 *n)** **n= 0,1,2,3**				
I2C_CTL0	I2Cn_BA+0x00	R/W	I²C Control Register 0	0x0000_0000
I2C_ADDR0	I2Cn_BA+0x04	R/W	I²C Slave Address Register0	0x0000_0000
I2C_DAT	I2Cn_BA+0x08	R/W	I²C Data Register	0x0000_0000
I2C_STATUS0	I2Cn_BA+0x0C	R	I²C Status Register 0	0x0000_00F8
I2C_CLKDIV	I2Cn_BA+0x10	R/W	I²C Clock Divided Register	0x0000_0000
I2C_TOCTL	I2Cn_BA+0x14	R/W	I²C Time-out Control Register	0x0000_0000
I2C_ADDR1	I2Cn_BA+0x18	R/W	I²C Slave Address Register1	0x0000_0000
I2C_ADDR2	I2Cn_BA+0x1C	R/W	I²C Slave Address Register2	0x0000_0000
I2C_ADDR3	I2Cn_BA+0x20	R/W	I²C Slave Address Register3	0x0000_0000
I2C_ADDRMSK0	I2Cn_BA+0x24	R/W	I²C Slave Address Mask Register0	0x0000_0000
I2C_ADDRMSK1	I2Cn_BA+0x28	R/W	I²C Slave Address Mask Register1	0x0000_0000
I2C_ADDRMSK2	I2Cn_BA+0x2C	R/W	I²C Slave Address Mask Register2	0x0000_0000
I2C_ADDRMSK3	I2Cn_BA+0x30	R/W	I²C Slave Address Mask Register3	0x0000_0000
I2C_WKCTL	I2Cn_BA+0x3C	R/W	I²C Wake-up Control Register	0x0000_0000
I2C_WKSTS	I2Cn_BA+0x40	R/W	I²C Wake-up Status Register	0x0000_0000
I2C_CTL1	I2Cn_BA+0x44	R/W	I²C Control Register 1	0x0000_0000
I2C_STATUS1	I2Cn_BA+0x48	R/W	I²C Status Register 1	0x0000_0000
I2C_TMCTL	I2Cn_BA+0x4C	R/W	I2C Timing Configure Control Register	0x0000_0000
I2C_BUSCTL	I2Cn_BA+0x50	R/W	I²C Bus Management Control Register	0x0000_0000

I2C_BUSTCTL	I2Cn_BA+0x54	R/W	I²C Bus Management Timer Control Register	0x0000_0000
I2C_BUSSTS	I2Cn_BA+0x58	R/W	I²C Bus Management Status Register	0x0000_0000
I2C_PKTSIZE	I2Cn_BA+0x5C	R/W	I²C Packet Error Checking Byte Number Register	0x0000_0000
I2C_PKTCRC	I2Cn_BA+0x60	R	I²C Packet Error Checking Byte Value Register	0x0000_0000
I2C_BUSTOUT	I2Cn_BA+0x64	R/W	I²C Bus Management Timer Register	0x0000_0005
I2C_CLKTOUT	I2Cn_BA+0x68	R/W	I²C Bus Management Clock Low Timer Register	0x0000_0005

16 QSPI

16.1 概述

QSPI 介面是一個同步串列資料通訊協定，利用 4 線雙向介面進行相互通訊。

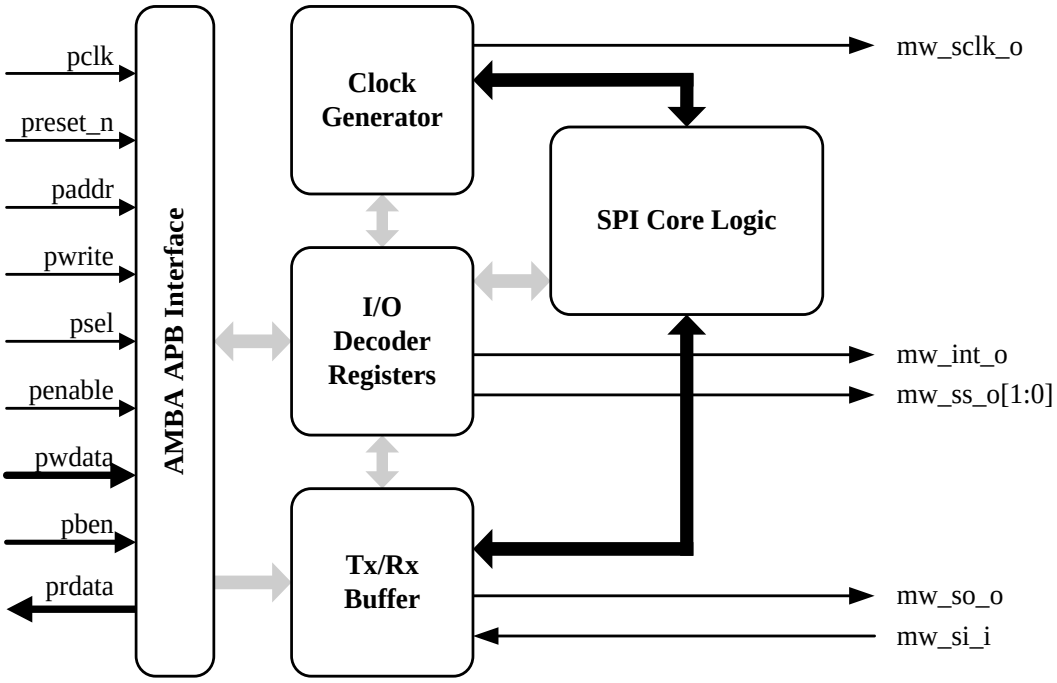
當從一個週邊設備接收資料時，QSPI 執行串-並的轉換，而在資料向週邊設備發送時，執行並-串的轉換。

QSPI 控制器可以被設置為主機模式，驅動多達 2 個週邊從設備。

16.2 特性

- 支援主機模式操作
- 一個事務傳輸的資料長度可配置為 1 到32 位，一次傳輸在高載模式下可被配置為傳輸 2 個事務，因此在高載模式下，一次傳輸的最大資料長度是 64 位
- 支援 MSB 或 LSB 優先傳輸
- 主機模式下支援 2 條從機選擇線，從機模式下僅支援 1 條從機選擇線
- 支援雙 / 四 IO 傳輸模式
- 支援 PDMA

16.3 方塊圖

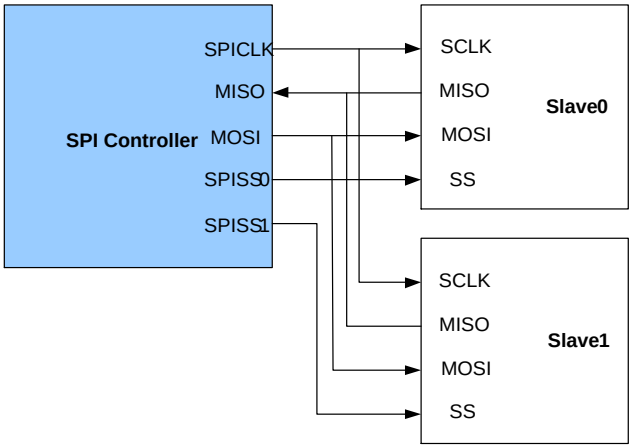


16.4 功能描述

16.4.1 從機選擇

在主機模式下，該 QSPI 控制器能通過從機選擇輸出腳 SS0 與 SS1 來驅動多達兩個片外從機設備，但是這種驅動兩個片外從機設備的操作是一種分時操作，不能同時與兩個週邊從設備進行通訊。

下面為一連線示意圖。



透過設定SS0(QSPI0_SSCTL[0])或設定SS1(QSPI0_SSCTL [1])使能SS1輸出腳或同時輸出。

```
QSPI0_SSCTL |= 0x1;        //使能SS0輸出腳
QSPI0_SSCTL |= 0x2;        //使能SS1輸出腳
QSPI0_SSCTL |= 0x3;        //同時使能SS0/SS1輸出腳
```

在主機模式下，從機選擇信號的有效電平可以通過設定 SSACTPOL(QSPI0_SSCTL [2]) 來設定為低有效或高有效，觸發條件的選擇取決於所連接的從機/主機設備的類型。

16.4.2 自動從機選擇

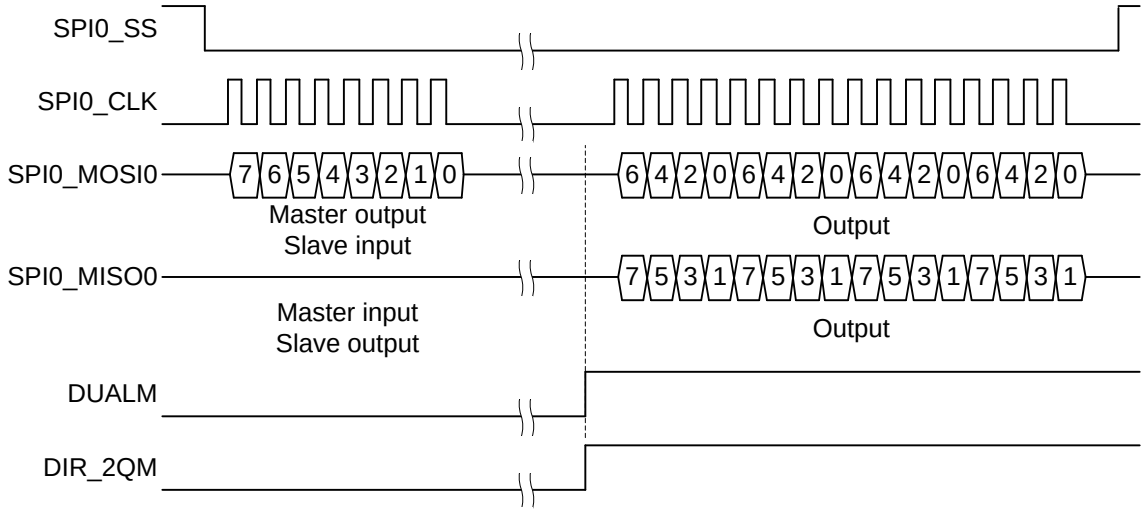
在主機模式下，如果置位元 AUTOSS (QSPI0_SSCTL[3])，將自動產生從機選擇信號，並根據 SS0 (QSPI0_SSCTL[0]) 與 SS1 (QSPI0_SSCTL[1]) 是否使能，將從機選擇信號輸出到 SS0 與 SS1 管腳上。這意味著當通過寫資料到 QSPI0_TX 來開始資料傳輸時，在 SSR[1:0] 中使能的從機選擇信號將由 QSPI 控制器自動設置為有效狀態，在資料傳輸結束後自動被設為無效狀態。

```
// 設定自動從機選擇在SS0管腳上
QSPI0_SSCTL |= 0x1;      //使能SS0輸出腳
QSPI0_SSCTL |= (0x1 << 3);      //使能自動從機選擇
```

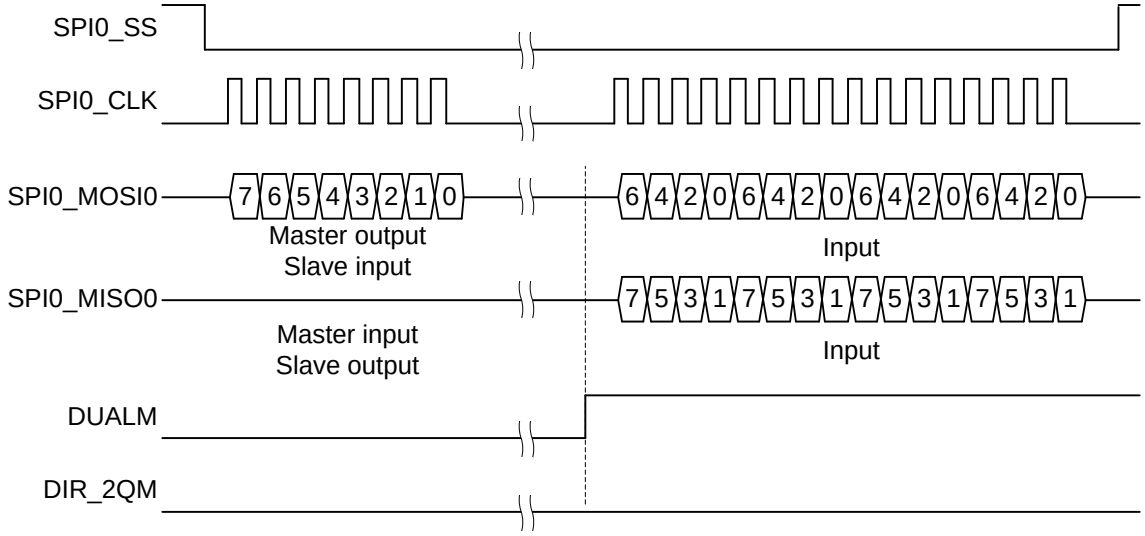
16.4.3 雙/四 IO 模式

當 DUALIOEN (QSPI0_CTL[21]) 被設置為1時，QSPI控制器支援雙IO傳輸。

雙IO輸出示意圖如下：



雙IO輸入示意圖如下：

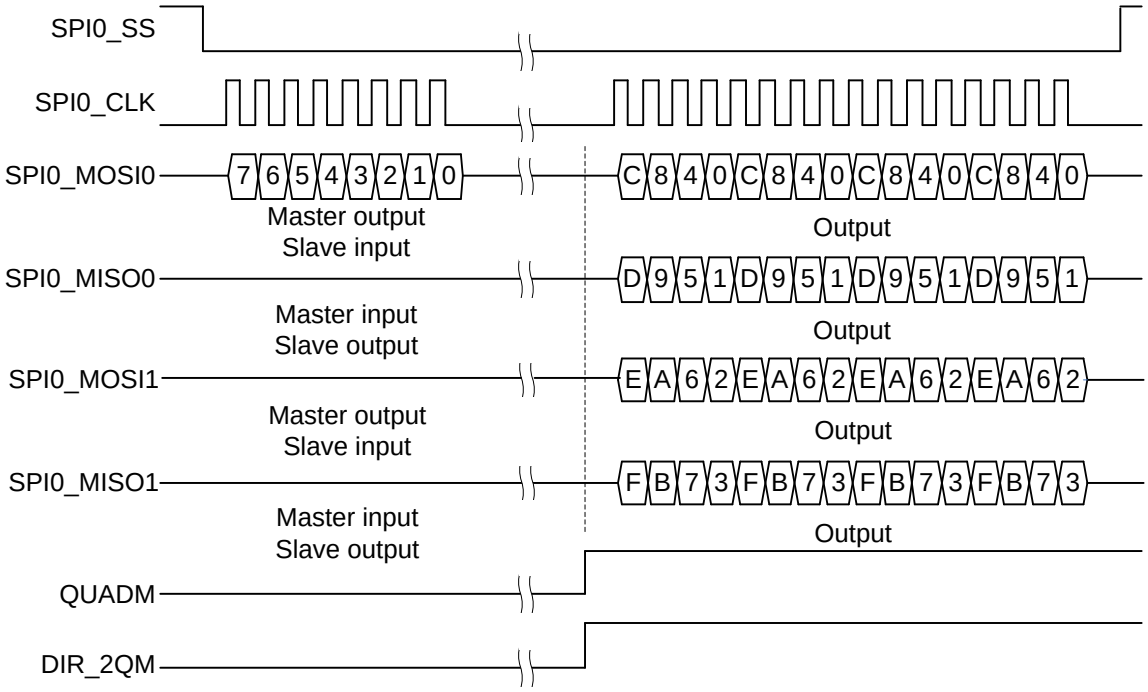


DATDIR(QSPI0_CTL[20]) 用於定義資料傳輸的方向。當設置 DATDIR位為 1，QSPI控制器向外部設備發送資料； 當設置 DATDIR位為 0，QSPI控制器從外部設備讀取數據。

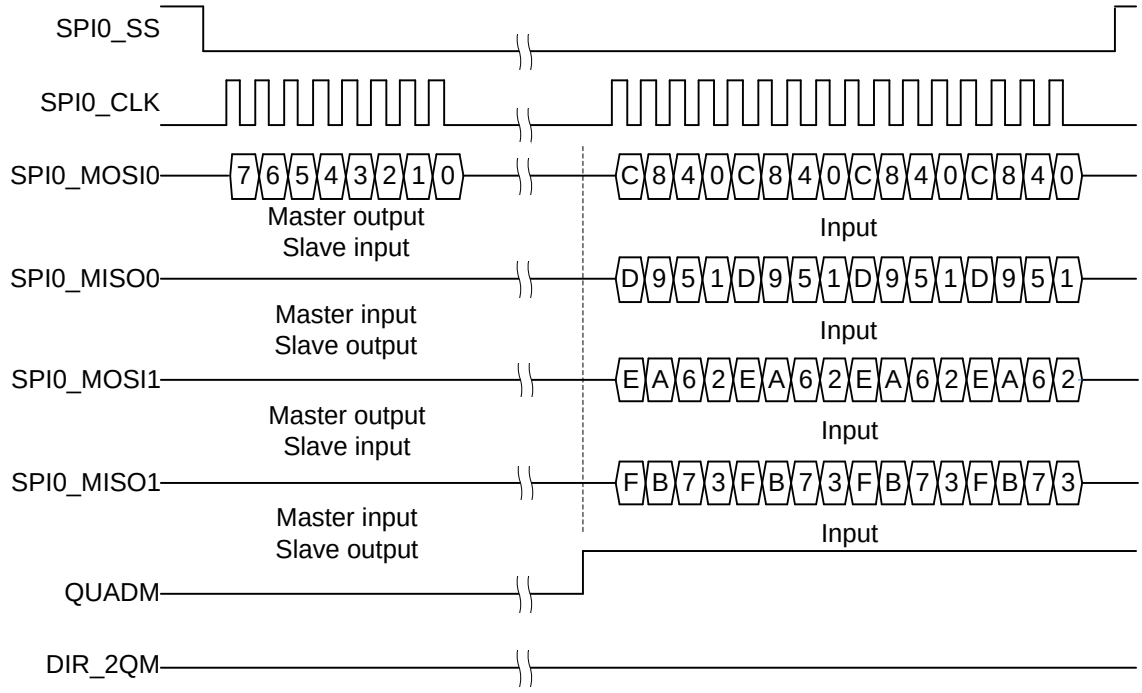
```
//使用雙IO模式，MOSI/MISO同時向外部發送資料
QSPI0_CTL |= (0x1 << 21);    //使能雙IO模式
QSPI0_CTL |= (0x1 << 20);  //方向為輸出
QSPI0_TX = 0x12;
```

當 QUADIOEN(QSPI0_CTL[22])被設置為1時，QSPI控制器支援四IO傳輸。

四IO輸出示意圖如下：



四IO輸入示意圖如下：



DATDIR(QSPI0_CTL[20])用於定義資料傳輸的方向。當設置 DATDIR位為 1，QSPI控制器向外部設備發送資料； 當設置 DATDIR位為 0，QSPI控制器從外部設備讀取數據。

```
//使用四IO模式，MOSI/MISO同時向外部發送資料
QSPI0_CTL |= (0x1 << 22);    //使能四IO模式
```

```
QSPI0_CTL |= (0x1 << 20);  //方向為輸出
QSPI0_TX = 0x12;
```

16.4.4 中斷

當 QSPI 控制器完成一次單元傳輸，單元傳輸中斷標誌 UNITIF(QSPI0_STATUS[1]) 將會被設置為 1。如果單元傳輸中斷使能位元 UNITIEN(QSPI0_CTL[17]) 被置位元，單元傳輸中斷事件將會向 CPU 產生一個中斷。單元傳輸標誌僅可以通過向自身寫 1 清除。

```
QSPI0_CTL |= 0x20000;  //使能中斷
QSPI0_TX  |= 0x5A5A5A5A;     //寫資料到 TX FIFO
while(!QSPI_isr);  //等待中斷
QSPI0_STATUS |= 0x2;  //清除IF位
```

如果沒設置UNITIEN位，仍可透過輪詢 BUSY(QSPI0_STATUS[0])位得知QSPI傳輸動作完成與否。

```
QSPI0_TX  |= 0x5A5A5A5A;     //寫資料到 TX FIFO
while(QSPI0_STATUS & 0x1);  //等待QSPI完成動作
```

16.4.5 從模式

QSPI 控制器支持從模式.

當 SLAVE(QSPIx_CTL[18]) 設為 1 時， QSPI 控制器就會運作在從模式。

SSACTPOL(QSPIx_SSCTL[2]) 定義從選擇信號的極性。

SLV3WIRE(QSPIx_SSCTL[4]) 設為 1 時， QSPI 控制器運作在三線模式，介面包括 QSPIx_CLK, QSPIx_MOSI and QSPIx_MISO。

當從選擇中斷發生時， SSACTIF(QSPIx_STATUS[2]) 位會變為 1。

下面這個範例 QSPI 控制器運作在從模式，等待從選擇信號變成低位後，從 FIFO 讀取資料。

```
unsigned int data;
QSPI0_CTL  |= (1 << 18);  // QSPI 控制器就會運作在從模式
QSPI0_SSCTL &= ~(1 << 2); // QSPI 控制器選擇低位運作
while(!(QSPI0_STATUS & 0x4));  //等待從選擇信號變成低位
while(!(QSPI0_STATUS & (1 << 8))); //等待 FIFO 還是空的
data = QSPI0_RX; //從 FIFO 讀取資料
```

16.4.6 PDMA 模式傳輸

QSPI 控制器支援 PDMA 傳輸功能。

當 TXPDMAEN (QSPIx_PDMACTL[0]) 位設置為 1 時, QSPI 控制器會自動向 PDMA 控制器發出一個傳送請求。

當 RXPDMAEN (QSPIx_PDMACTL[1]) 位設置為 1 時, 控制器會開始進行 PDMA 接收. 當 RX

FIFO 有資料時，QSPI 會自動向 PDMA 發出一個接收的請求。

```
QSPI0_PDMACTL |= 1;        //向 PDMA 控制器發出傳送請求
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_TX_DMA_CH)); //等待 PDMA 傳送完成

QSPIx_PDMACTL |= 2;        //向 PDMA 控制器發出接收請求
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_RX_DMA_CH)); //等待 PDMA 接收完成
```

16.4.7 QSPI 控制器完整使用示例

依照下列步驟使用QSPI控制器

1. 設定寄存器 QSPI0_CLKDIV 來決定串列時鐘輸出頻率
2. 選擇是否使用自動從選擇位 AUTOSS
3. 選擇有效電平位SSACTPOL
4. 通過設定從機選擇寄存器位 SS0(QSPI0_SSCTL[0]) 或 SS1(QSPI0_SSCTL[1]) 選擇哪一個從機選擇信號會在相應的 IO 管腳上輸出，以啟動片外從設備。
5. 設置 QSPI0_CTL，在 TXNEG 位元選擇在串列時鐘的下降沿或上升沿發送資料。
6. 在 LSB 位設定 MSB或LSB 優先傳輸。
7. 設定DWIDTH，決定每次傳輸數據長度。
8. 將傳輸的數據寫入QSPI0_TX寄存器。
9. 等待 QSPI 中斷發生（如果中斷使能位 UNITIEN 被置位），或者輪詢 BUSY(QSPI0_STATUS[0]) 位，直到該位元被硬體自動清零。
10. 從 QSPI0_RX 寄存器讀出接收到的資料
11. 完成一次的傳輸

16.5 寄存器

Register	Offset	R/W	Description	Reset Value
QSPI0_BA = 0xB000_6000				
QSPI0_CTL	QSPI0_BA+0x00	R/W	QSPI Control Register	0x0000_0034
QSPI0_CLKDIV	QSPI0_BA+0x04	R/W	QSPI Clock Divider Register	0x0000_0000
QSPI0_SSCTL	QSPI0_BA+0x08	R/W	QSPI Slave Select Control Register	0x0000_0000
QSPI0_PDMACTL	QSPI0_BA+0x0C	R/W	QSPI PDMA Control Register	0x0000_0000

QSPI0_FIFOCTL	QSPI0_BA+0x10	R/W	QSPI FIFO Control Register	0x4400_0000
QSPI0_STATUS	QSPI0_BA+0x14	R/W	QSPI Status Register	0x0005_0110
QSPI0_TX	QSPI0_BA+0x20	W	QSPI Data Transmit Register	0x0000_0000
QSPI0_RX	QSPI0_BA+0x30	R	QSPI Data Receive Register	0x0000_0000

17 SPI

17.1 概述

SPI 介面是一個同步串列資料通訊協定，利用 4 線介面進行相互通訊。

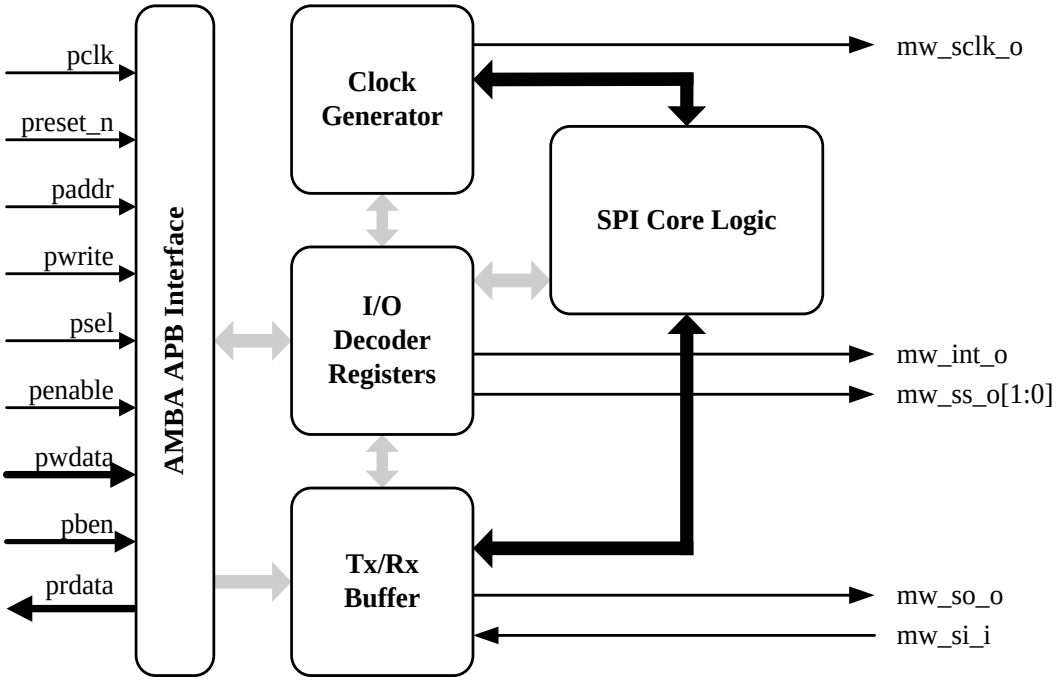
當從一個週邊設備接收資料時，SPI 執行串-並的轉換，而在資料向週邊設備發送時，執行並-串的轉換。

SPI 控制器可以被設置為主機模式，驅動多達 2 個週邊從設備。

17.2 特性

- 支援主機模式操作
- 一個事務傳輸的資料長度可配置為 1 到32 位，一次傳輸在高載模式下可被配置為傳輸 2 個事務，因此在高載模式下，一次傳輸的最大資料長度是 64 位
- 支援 MSB 或 LSB 優先傳輸
- 主機模式下支援 2 條從機選擇線，從機模式下僅支援 1 條從機選擇線
- 支援 PDMA

17.3 方塊圖

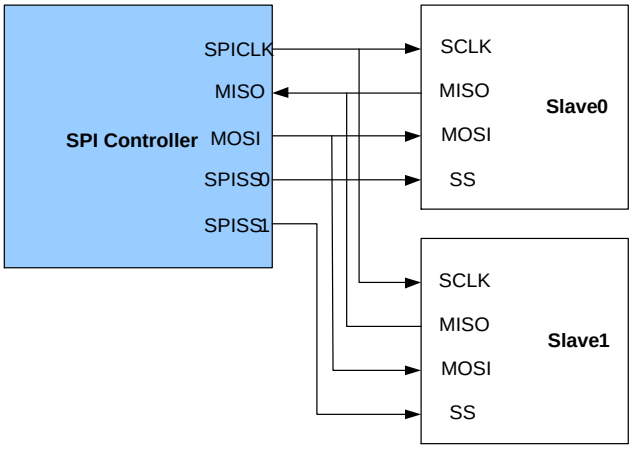


17.4 功能描述

17.4.1 從機選擇

在主機模式下，該 SPI 控制器能通過從機選擇輸出腳 SS0 與 SS1 來驅動多達兩個片外從機設備，但是這種驅動兩個片外從機設備的操作是一種分時操作，不能同時與兩個週邊從設備進行通訊。

下面為一連線示意圖。



透過設定SS0(SPI0_SSCTL[0])或設定SS1(SPI0_SSCTL [1])使能SS1輸出腳或同時輸出。

```
SPI0_SSCTL |= 0x1;          //使能SS0輸出腳
SPI0_SSCTL |= 0x2;          //使能SS1輸出腳
SPI0_SSCTL |= 0x3;          //同時使能SS0/SS1輸出腳
```

在主機模式下，從機選擇信號的有效電平可以通過設定 SSACTPOL(SPI0_SSCTL [2]) 來設定為低有效或高有效，觸發條件的選擇取決於所連接的從機/主機設備的類型。

17.4.2 自動從機選擇

在主機模式下，如果置位元 AUTOSS (SPI0_SSCTL[3])，將自動產生從機選擇信號，並根據 SS0 (SPI0_SSCTL[0]) 與 SS1 (SPI0_SSCTL[1]) 是否使能，將從機選擇信號輸出到 SS0 與 SS1 管腳上。這意味著當通過寫資料到 SPI0_TX 來開始資料傳輸時，在 SSR[1:0] 中使能的從機選擇信號將由 SPI 控制器自動設置為有效狀態，在資料傳輸結束後自動被設為無效狀態。

```
// 設定自動從機選擇在SS0管腳上
SPI0_SSCTL |= 0x1;      //使能SS0輸出腳
SPI0_SSCTL |= (0x1 << 3);      //使能自動從機選擇
```

17.4.3 中斷

當 SPI 控制器完成一次單元傳輸，單元傳輸中斷標誌 UNITIF(SPI0_STATUS[1]) 將會被設置為 1。如果單元傳輸中斷使能位元 UNITIEN(SPI0_CTL[17]) 被置位元，單元傳輸中斷事件將會向

CPU 產生一個中斷。單元傳輸標誌僅可以通過向自身寫 1 清除。

```
SPI0_CTL |= 0x20000;  //使能中斷
SPI0_TX  |= 0x5A5A5A5A;      //寫資料到 TX FIFO
while(!SPI_isr);  //等待中斷
SPI0_STATUS |= 0x2;  //清除IF位
```

如果沒設置UNITIEN位，仍可透過輪詢 BUSY(SPI0_STATUS[0])位得知SPI傳輸動作完成與否。

```
SPI0_TX  |= 0x5A5A5A5A;      //寫資料到 TX FIFO
while(SPI0_STATUS & 0x1);  //等待SPI完成動作
```

17.4.4 從模式

SPI 控制器支持從模式.

當 SLAVE(SPIx_CTL[18]) 設為 1 時， SPI 控制器就會運作在從模式。

SSACTPOL(SPIx_SSCTL[2]) 定義從選擇信號的極性。

當從選擇中斷發生時， SSACTIF(SPIx_STATUS[2]) 位會變為 1。

下面這個範例 SPI 控制器運作在從模式，等待從選擇信號變成低位後，從 FIFO 讀取資料。

```
unsigned int data;
SPIx_CTL  |= (1 << 18);    // SPI 控制器就會運作在從模式
SPIx_SSCTL &= ~(1 << 2); // SPI 控制器選擇低位運作
while(!(SPIx_STATUS & 0x4));   //等待從選擇信號變成低位
while(!(SPIx_STATUS & (1 << 8))); //等待 FIFO 還是空的
data = SPIx_RX; //從 FIFO 讀取資料
```

17.4.5 PDMA 模式傳輸

SPI 控制器支援 PDMA 傳輸功能。

當 TXPDMAEN (SPIx_PDMACTL[0]) 位設置為 1 時, SPI 控制器會自動向 PDMA 控制器發出一個傳送請求。

當 RXPDMAEN (SPIx_PDMACTL[1]) 位設置為 1 時, 控制器會開始進行 PDMA 接收. 當 RX FIFO 有資料時，SPI 會自動向 PDMA 發出一個接收的請求。

```
SPI0_PDMACTL |= 1;         //向 PDMA 控制器發出傳送請求
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_TX_DMA_CH)); //等待 PDMA 傳送完成

SPIx_PDMACTL |= 2;         //向 PDMA 控制器發出接收請求
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_RX_DMA_CH)); //等待 PDMA 接收完成
```

17.4.6 SPI 控制器完整使用示例

依照下列步驟使用SPI控制器

1. 設定寄存器 SPI0_CLKDIV 來決定串列時鐘輸出頻率
2. 選擇是否使用自動從選擇位 AUTOSS
3. 選擇有效電平位SSACTPOL
4. 通過設定從機選擇寄存器位 SS0(SPI0_SSCTL[0]) 或 SS1(SPI0_SSCTL[1]) 選擇哪一個從機選擇信號會在相應的 IO 管腳上輸出，以啟動片外從設備。
5. 設置 SPI0_CTL，在 TXNEG 位元選擇在串列時鐘的下降沿或上升沿發送資料。
6. 在 LSB 位設定 MSB或LSB 優先傳輸。
7. 設定DWIDTH，決定每次傳輸數據長度。
8. 將傳輸的數據寫入SPI0_TX寄存器。
9. 等待 SPI 中斷發生 (如果中斷使能位 UNITIEN 被置位)，或者輪詢 BUSY(SPI0_STATUS[0]) 位，直到該位元被硬體自動清零。
10. 從 SPI0_RX 寄存器讀出接收到的資料
11. 完成一次的傳輸

17.5 寄存器

Register	Offset	R/W	Description	Reset Value
SPI0_BA = 0xB000_6100 **SPI1_BA = 0xB000_6200**				
SPIx_CTL	SPIx_BA+0x00	R/W	SPI Control Register	0x0000_0034
SPIx_CLKDIV	SPIx_BA+0x04	R/W	SPI Clock Divider Register	0x0000_0000
SPIx_SSCTL	SPIx_BA+0x08	R/W	SPI Slave Select Control Register	0x0000_0000
SPIx_PDMACTL	SPIx_BA+0x0C	R/W	SPI PDMA Control Register	0x0000_0000
SPIx_FIFOCTL	SPIx_BA+0x10	R/W	SPI FIFO Control Register	0x4400_0000
SPIx_STATUS	SPIx_BA+0x14	R/W	SPI Status Register	0x0005_0110
SPIx_TX	SPIx_BA+0x20	W	SPI Data Transmit Register	0x0000_0000
SPIx_RX	SPIx_BA+0x30	R	SPI Data Receive Register	0x0000_0000

18 I²S

18.1 概述

音頻控制器包括了I2S及PCM協議與外部音頻CODEC接口。I2S和PCM接口，支持錄音和回放8，16，18，20和24位的精度在左或右聲道上。

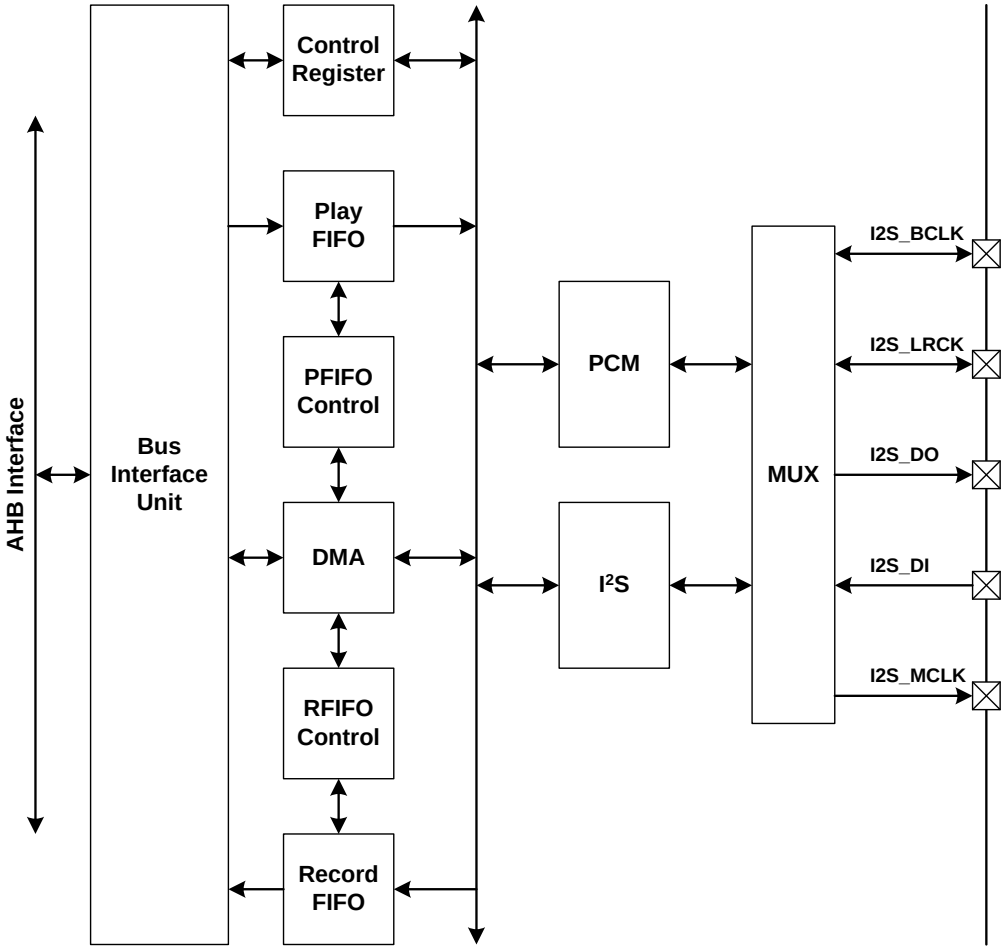
當在18/20/24位精度操作時，每個左/右聲道採樣值是存儲於一個32位的字節內。

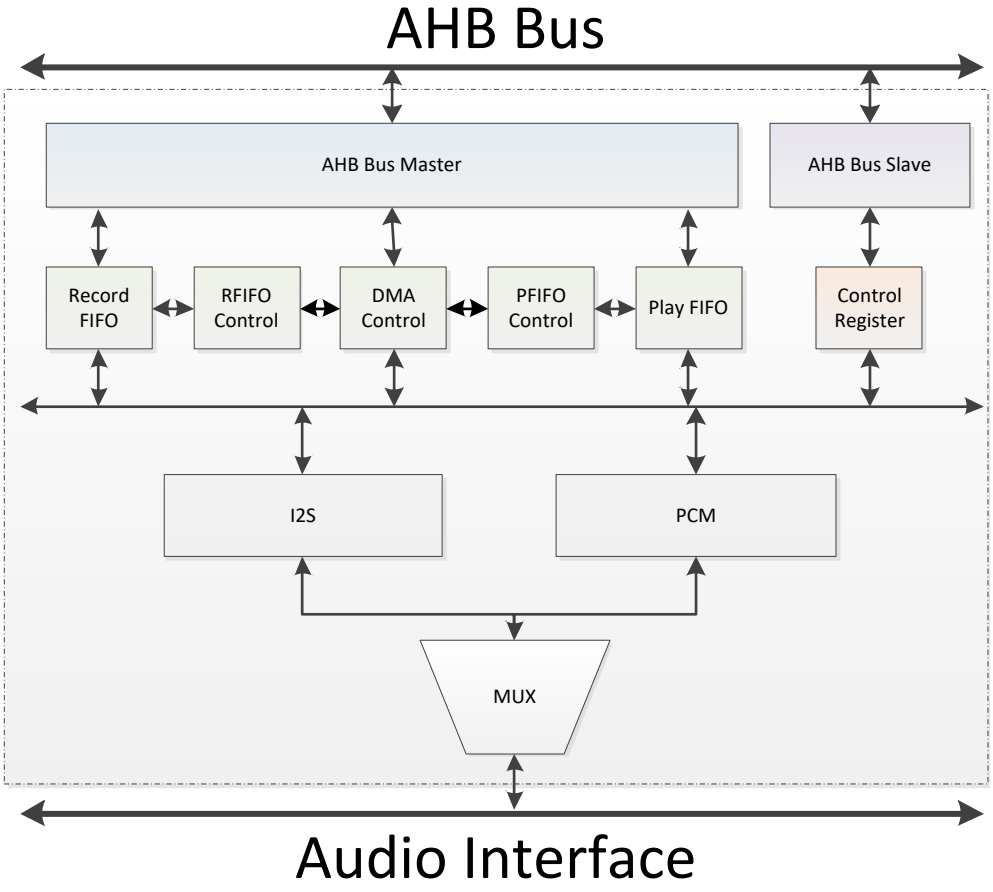
每個左/右聲道有效的數據為 MSB 24/20/18 比特其他的LSB比特則是填充為零。當在16位精度操作，右聲道採樣值存儲在一個32位字的MSB和左聲道採樣值存儲在32位字的LSB。

18.2 特性

- 支援I2S 介面錄音和回放功能
  - 支援左/右聲道
  - 支援 8,16,18, 20,24位數據精度
  - 支援主/從模式
- 支援PCM介面錄音和回放功能
  - 支援左/右聲道
  - 支援 8,16,18, 20,24位數據精度
  - 支援主模式
  - 支援不同聲道數據可存放於不同內存位址
- 支援配合DMA來實現錄音和回放
- 支援中斷功能

18.3 方塊圖


AHB Interface
Bus Interface Unit
Control Register
Play FIFO
PFIFO Control
DMA
RFIFO Control
Record FIFO
PCM
I²S
MUX
I2S_BCLK
I2S_LRCK
I2S_DO
I2S_DI
I2S_MCLK



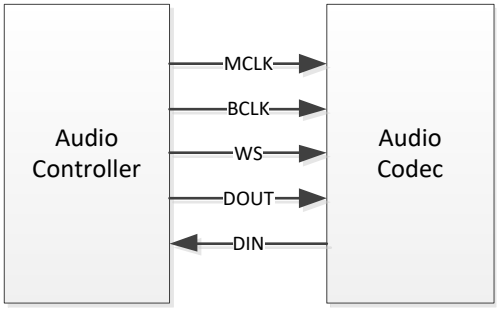
18.4 功能描述

18.4.1 I²S 主/從模式設定

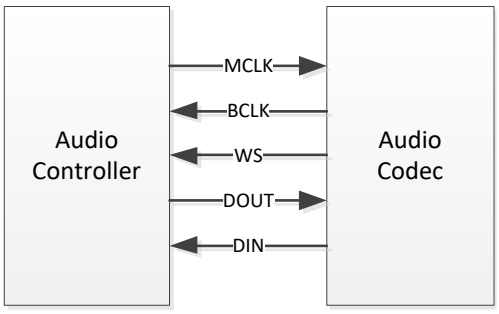
使用從模式，可設置SLAVE(I2S_CON[20])為1，主模式需設為0。

在選擇使用I²S介面時(I2S_CON[1:0] = 1)才能選擇主或從模式，PCM介面只能使用主模式。

主模式接線示意圖如下：

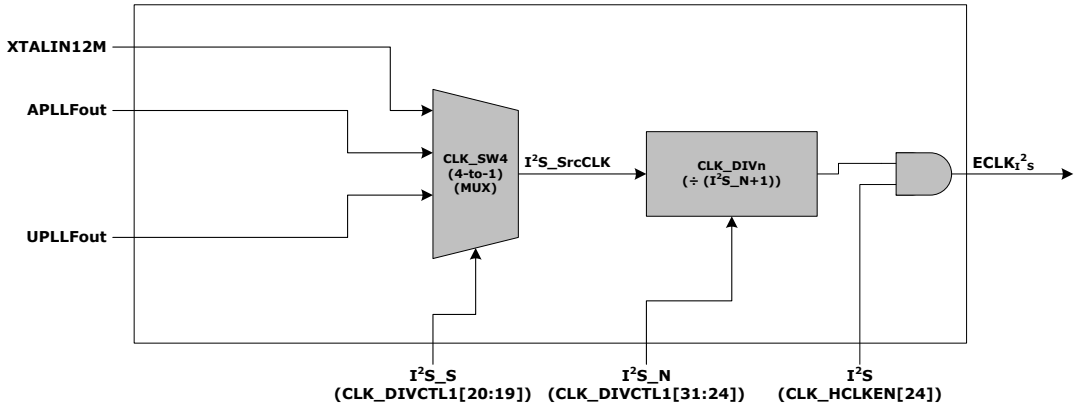


從模式接線示意圖如下：



18.4.2 I²S 時鐘源設定

I²S的時鐘來源可為APLL，UPLL或外部晶振，可透過設置I2S_S(CLK_DIVCTL1[20:19])來做選擇。



CLK_DIVCTL1 = (CLK_DIVCTL1 & ~(0x3 << 19)) | 0x2; //選擇APLL為I²S時鐘源

18.4.3 I²S 輸出入時鐘計算及設置

I²S需要設置的輸出入時鐘有MCLK及BCLK兩種，如果是使用I²S從模式，只需要設置MCLK即可。

一般來說為了輸出精準的時鐘，時鐘源會選擇PLL且PLL會設置成輸出12.288MHz，16.934MHz 或 11.285Mhz，下面以48000Hz採樣率，16位資料精度及雙聲道為例，舉例如何計算及設置：

假設音頻codec支援256倍採樣率，MCLK的計算方式如下：

MCLK = 256 * 48000 =11288000 Hz = 12.288MHz

且以16位資料精度及雙聲道為例，BCLK的計算方式如下：

BCLK = 48000 * 16 * 2 = 1536000 Hz = 1.536MHz

此時可設置分頻值PSR(I2S_CON[19:16])為 12.288 / 12.288 - 1 = 1 - 1 = 0

和BCLK_DIV(I2S_CON[7:5])為 (12.288/1.536) / 2 - 1 = 8 / 2 - 1 = 3

```
I2S_CON = I2S_CON & ~(0xF << 16);          //PRS=0
I2S_CON = I2S_CON & ~(0x1 << 4);           //PLL需經過PRS分頻
I2S_CON = (I2S_CON & ~(0x1 << 5)) | 0x3; //BCLK_DIV=3
```

18.4.4 設置 DMA

I²S使用DMA來實現錄音和回放，相關的DMA設置說明如下：

- 設置錄音和回放 DMA 基礎位置寄存器（I2S_RDESB 和 I2S_PDESB），所有錄音和回放的數據會放在這個基礎位置，一般來說都會是在內存中一塊非緩存的連續空間。
- DMA 數據長度寄存器（I2S_RDES_LENGTH 和 I2S_PDES_LENGTH），指定 DMA 空間的長度。
- DMA 目前位置指示寄存器（I2S_RDESC 和 I2S_PDESC），用於指示目前 DMA 存放或讀取數據的位置。可用於決定目前 DMA 空間內所剩未使用的部份有多少，以利軟體做相對應的動作。
- 與中斷相關部份，可透過設置 R_DMA_IRQ_SEL(I2S_GLBCON[15:14])，P_DMA_IRQ_SEL(I2S_GLBCON[13:12])決定在 DMA 已經讀或寫了 1/2，1/4，1/8 的長度時發生中斷通知軟體，開啟中斷可透過設置 R_DMA_IRQ_EN(I2S_GLBCON[21])或 P_DMA_IRQ_EN(I2S_GLBCON[20])位。

DMA設置及流程以回放為例如下。

```
I2S_PDESB = 0x80001000;              //配置非緩存的內存位置
I2S_PDES_LENGTH = 2*1024;            //長度為2k位元組
I2S_CON = (I2S_CON  & ~(0x3 << 12)) | (0x1 << 12);  //中斷發生為1/2 DMA長度時
I2S_CON |= (0x1 << 20);           //啟動中斷
```

- 軟體可透過讀取 P_DMA_RIA_SN(I2S_PSR[7:5])或 R_DMA_RIA_SN(I2S_RSR[7:5])位得知 DMA 正在讀取或寫入哪一個內存區段。假設軟體讀到的值為 2，選擇是 1/8 時通知軟體，則此意思是 DMA 正在讀取或寫入第 2/8 個區段。
- 遞減計數器主要用於軟體需知道 DMA 傳了幾筆數據時使用，軟體需先寫入一初始值於 I2S_COUNTER 寄存器中，當 DMA 傳了一筆數據時，寄存器中的值會遞減 1，直到值為 0 時，另可設置 IRQ_DMA_CNTER_EN(I2S_GLBCON[4])位發生中斷通知軟體。

```
I2S_COUNTER = 0x1000;              //設置倒數初始值為0x1000
…
while(I2S_COUNTER>0x30);           //檢測值是否小於0x30
…
```

- 過零偵測(zero crossing detection)用於檢測在 DMA 緩衝內的數據是否全為 0 或數據符號比特有改變時，可設置 IRQ_DMA_DATA_ZERO_EN(I2S_GLBCON[3])位發生中斷通知軟體。

18.4.5 DMA 數據在內存存放順序

使用I²S 18, 20, 24位元時，每筆音效數據在DMA緩衝內都會使用至32位元。

下面舉I2S 16位元為例：

雙通道(立體音)：

基礎位址	DMA 緩衝 (雙通道)
0x1000	左聲道 – LSB 位元組
0x1001	左聲道 – MSB 位元組
0x1002	右聲道 – LSB 位元組
0x1003	右聲道 – MSB 位元組
0x1004	左聲道 – LSB 位元組
0x1005	左聲道 – MSB 位元組
0x1006	右聲道 – LSB 位元組
0x1007	右聲道 – MSB 位元組
...	...

單通道(單音)：

基礎位址	DMA 緩衝 (單通道)
0x1000	左聲道 – LSB 位元組
0x1001	左聲道 – MSB 位元組
0x1002	左聲道 – LSB 位元組
0x1003	左聲道 – MSB 位元組

0x1004	左聲道 – LSB 位元組
0x1005	左聲道 – MSB 位元組
0x1006	左聲道 – LSB 位元組
0x1007	左聲道 – MSB 位元組
…	…

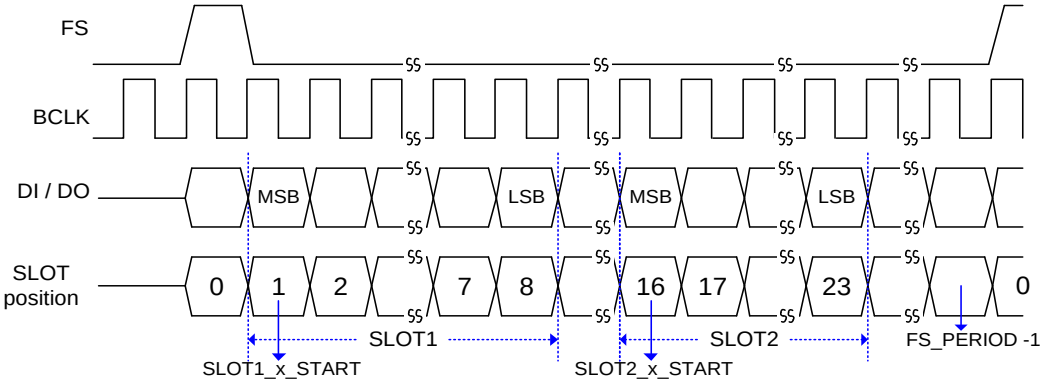
18.4.6 介面選擇及設置

可選擇的介面有I²S及PCM，透過設置BLOCK_EN(I2S_GLBCON[0])，來選擇所使用的介面。

```
I2S_GLBCON = (I2S_GLBCON & ~0x3) | 0x2;  //選擇PCM介面
```

18.4.7 PCM 介面的配置

一個PCM介面的波形如下：



SLOT1_O_START (ACTL_PCMS1ST[25:16]) = 1,
SLOT2_O_START (ACTL_PCMS2ST[25:16]) = 16, BCLKP (ACTL_PCMCON[0]) = 0

如上圖，可提供進行配置的部份有：

1. 兩個FS之間的波特數 – FS_PERIOD(I2S_PCMCON[25:16])。
2. SLOT1_x_START 及 SLOT2_x_START 距離 FS 的波特數 – I2S_PCMS1ST 及 I2S_PCMS2ST。

舉例配置一個採樣率為8kHz，一個槽的數據精度為32bits，兩個槽都用的情形如下(假設I²S時鐘源的時鐘為24.576MHz)：

```
//BCLK=24.576Mhz/48 = 512k
I2S_PCMCON = I2S_PCMCON | (23<<8));
```

```
//FS_PERIOD = 32+32
I2S_PCMCON = (63<<16) | 0;    //FS= 512/64=8k, //BCLKP = 0

//SLOT1_O_START = 1
//SLOT1_I_START = 1
I2S_PCMS1ST = 0x00010001;

//SLOT2_O_START = 33
//SLOT2_I_START = 33
I2S_PCMS2ST = 0x00210021;
```

18.4.8 數據分割

數據分割可將原本連續的聲道數據依據聲道不同存放於不同的DMA緩存內，方便軟體針對單一聲道內數據進行連續的讀取或寫入及運算。

此功能可透過設置I2S_RDESB2及I2S_PDESB2寄存器指定數據分割之後第二個存放位址。原來的I2S_RDESB及I2S_PDESB固定放置I2S左聲道或PCM slot1的數據，I2S_RDESB2及I2S_PDESB2則是放置I²S右聲道或PCM slot2的數據。

可設置SPLIT_DATA(I2S_RESET[20])來啟動此功能。啟動數據分割後，數據存放狀況(以I²S介面為例)如下：

基礎位址-1	DMA 緩衝
0x1000	左聲道 – LSB 位元組
0x1001	左聲道 – MSB 位元組
0x1002	左聲道 – LSB 位元組
0x1003	左聲道 – MSB 位元組
0x1004	左聲道 – LSB 位元組
0x1005	左聲道 – MSB 位元組
0x1006	左聲道 – LSB 位元組
0x1007	左聲道 – MSB 位元組
…	…

基礎位址-2	DMA 緩衝
0x2000	右聲道 – LSB 位元組
0x2001	右聲道 – MSB 位元組
0x2002	右聲道 – LSB 位元組
0x2003	右聲道 – MSB 位元組
0x2004	右聲道 – LSB 位元組
0x2005	右聲道 – MSB 位元組
0x2006	右聲道 – LSB 位元組
0x2007	右聲道 – MSB 位元組
…	…

18.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
I2S Base Address: **I2S_BA = 0xB000_2000**				
I2S_GLBCON	I2S_BA+0x000	R/W	I2S Global Control Register	0x0000_0000
I2S_RESET	I2S_BA+0x004	R/W	I2S Sub Block Reset Control Register	0x0000_0000
I2S_RDESB	I2S_BA+0x008	R/W	I2S Record DMA Destination Base Address Register	0x0000_0000
I2S_RDES_LENGTH	I2S_BA+0x00C	R/W	I2S Record DMA Destination Length Register	0x0000_0000
I2S_RDESC	I2S_BA+0x010	R	I2S Record DMA Destination Current Address Register	0x0000_0000
I2S_PDESB	I2S_BA+0x014	R/W	I2S Play DMA Destination Base Address Register	0x0000_0000
I2S_PDES_LENGTH	I2S_BA+0x018	R/W	I2S Play DMA Destination Length Register	0x0000_0000
I2S_PDESC	I2S_BA+0x01C	R	I2S Play DMA Destination Current Address Register	0x0000_0000
I2S_RSR	I2S_BA+0x020	R/W	I2S Record Status Register	0x0000_0000
I2S_PSR	I2S_BA+0x024	R/W	I2S Play Status Register	0x0000_0000
I2S_CON	I2S_BA+0x028	R/W	I2S Control Register	0x0000_0000
I2S_COUNTER	I2S_BA+0x02C	R/W	I2S Play DMA Down Counter Register	0xFFFF_FFFF

I2S_PCMCON	I2S_BA+0x030	R/W	I2S PCM Mode Control Register	0x0000_0000
I2S_PCMS1ST	I2S_BA+0x034	R/W	I2S PCM Mode Slot 1 Start Register	0x0000_0000
I2S_PCMS2ST	I2S_BA+0x038	R/W	I2S PCM Mode Slot 2 Start Register	0x0000_0000
I2S_RDESB2	I2S_BA+0x040	R/W	I2S Record DMA Destination Base Address 2 Register	0x0000_0000
I2S_PDESB2	I2S_BA+0x044	R/W	I2S Play DMA Destination Base Address 2 Register	0x0000_0000

19 乙太網路控制器 (EMAC)

19.1 概述

NUC980 支持了兩個獨立的乙太網路控制器可提供給網路相關的應用使用.

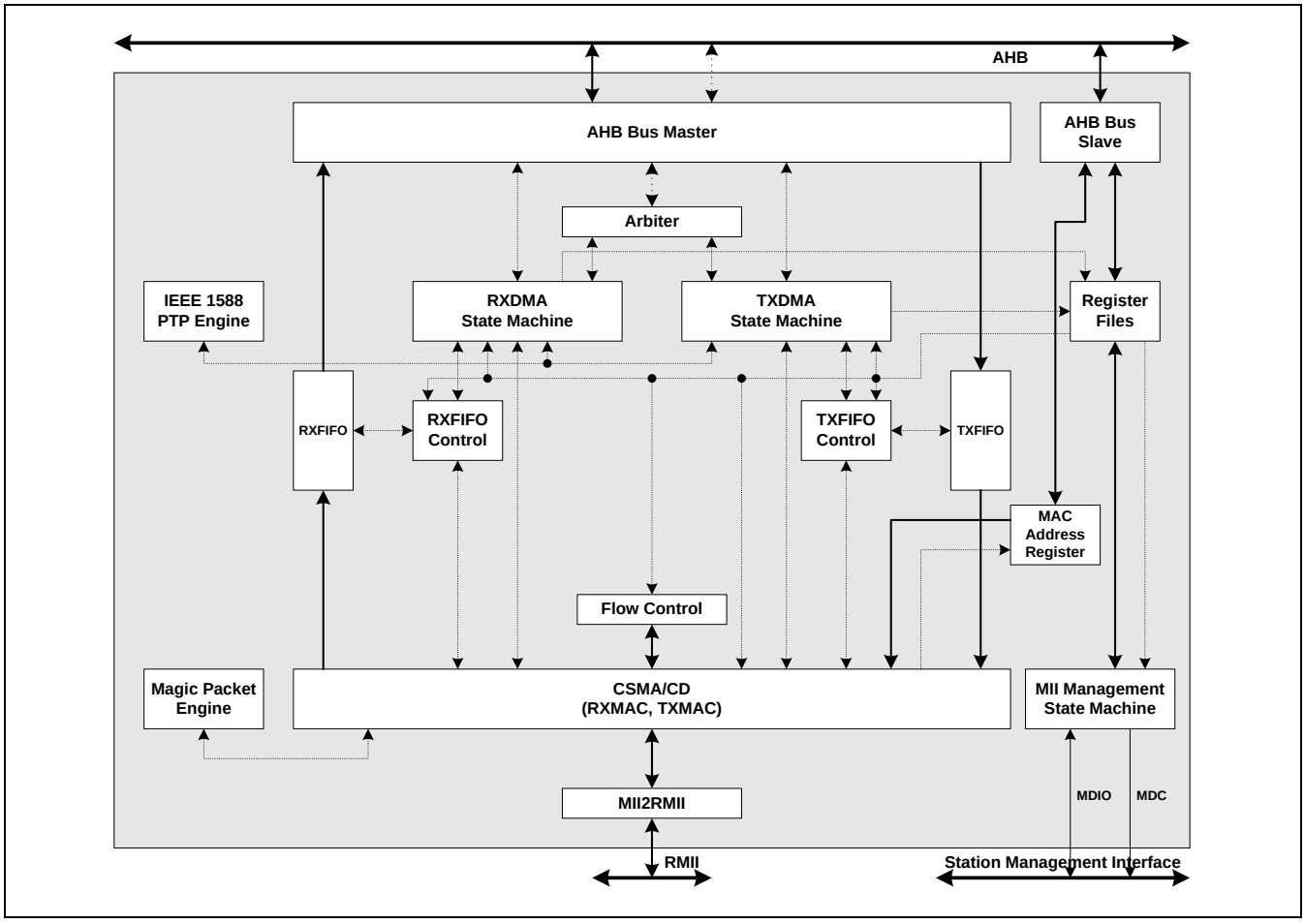
乙太網路控制器包含了IEEE 802.3 乙太網通訊協定引擎, 並且內建了結合存儲 (CAM) 用於MAC 位址比對. 另外還包含傳送 FIFO, 接收 FIFO, Tx/Rx狀態機控制單元, IEEE 1588 時間戳記, 並且支持魔法封包分析的功能.

乙太網路控制器透過 RMII 介面與外部的 PHY 連接.

19.2 特性

- 支持IEEE Std. 802.3 CSMA/CD 通訊協定.
- 支持IEEE Std. 1588 協定的網路封包時間戳記.
- 支持全雙工, 半雙工, 以及 10Mbps, 100Mbps 兩種傳輸速率.
- 支持RMII 介面
- 支持MII 管理功能, 以控制外部的 Ethernet PHY
- 支持暫停封包, 用於流控功能.
- 可接收長封包 (> 1518 位元) 與短封包 (< 64 位元).
- 內建 16 組結合存儲作為MAC位址比對
- 支持魔法封包, 可用於系統喚醒
- 支持256位元傳送FIFO 與 256位元接收FIFO
- 支持DMA 功能

19.3 方塊圖



19.4 功能描述

19.4.1 PHY 控制

乙太網路控制器透過 EMAC_MDC, EMAC_MDIO 兩根管腳讀寫 PHY 內部的寄存器, 與 PHY 溝通. EMAC_MDC 的時鐘由 AHB 除上 MDCLK_N (CLK_DIVCTL8) + 1 而定, 實際能輸出多快的頻率須根據 PHY 的技術文件決定. 當 MDCON (EMAC_MIIDA[19]) 設 1 後, EMAC_MDC 即開始輸出時鐘. 這個時鐘不須一直持續, 就算停止也不影響封包的收送.

對 PHY 寄存器的存取, 需要知道 PHY 的地址, 以及PHY內部寄存器的地址. PHY 的地址依據不同的 PHY , 以及實際的外部線路, 可能會有不同的設定. 以 IC Plus 的 IP101G 為例, PHY 的地址是由 PHY_AD0, PHY_AD1, PHY_AD2, PHY_AD3 這四根腳位的上拉或是下拉決定. PHY 內部寄存器的地址則需要查閱各廠商 PHY 的技術文件. 但是IEEE 802.3 內定義了一些基本寄存器, 不同廠家的 PHY 都會支持這些寄存器, 所以不同廠家的 PHY 有可能使用相同的驅動程式控制.

以下是讀取PHY 內部寄存器的步驟以及範例程式:

1. 設置 PHY 地址 PHYAD (EMAC_MIIDA[12:8]) 以及 PHY 寄存器地址 PHYRAD (EMAC_MIIDA[4:0]).

2. 將 BUSY (EMAC_MIIDA[17]), 以及 MDCON (EMAC_MIIDA[18]) 置 1, 以送出讀取命令.
3. 輪詢 BUSY 位, 直到自動清 0
4. 自 EMAC_MIID 讀取 PHY 寄存器的值.

```
unsigned int mdio_read(unsigned int reg, unsigned int addr)
{
    EMAC_MIIDA = reg | (addr << 8) | BUSY | MDCON;
    while(EMAC_MIIDA & BUSY);
    return EMAC_MIID;
}
```

以下則是寫入 PHY 內部寄存器的步驟以及範例程式:

1. 將要寫入的值填進 EMAC_MIID 寄存器.
2. 設置PHY 地址 PHYAD 以及 PHY 寄存器地址 PHYRAD.
3. 將 WRITE (EMAC_MIIDA[16]), BUSY 以及 MDCON 置 1, 以送出寫入命令.
4. 輪詢 BUSY 位, 直到自動清 0 則寫入完成.

```
unsigned int mdio_write(unsigned int reg, unsigned int addr, unsigned int data)
{
    EMAC_MIID = data;
    EMAC_MIIDA = reg | (addr << 8) | BUSY | MDCON | WRITE;
    while(EMAC_MIIDA & BUSY);
}
```

讀取 PHY 寄存器的主要目的是要設置乙太網正確的工作模式與設定. PHY 在正確接上網線後, 會自動進行自動協商(Auto-Negotiation), 以決定要工作在全雙工, 半雙工中哪一種模式, 以及 10Mbps, 100Mbps 中的哪一種傳輸速率. 驅動程式需要根據讀回的PHY 寄存器 Auto-Negotiation Link Partner Base Page Ability 值, 設置相對應的 OPMOD (EMAC_MCMDR[20]) 以及 FDUP (EMAC_MCMDR[18]), 讓 EMAC 跟 PHY 工作在相同的狀態, 這樣才可正確的收送封包.

19.4.2 CAM 設置

結合存儲是用來執行 MAC 地址的比對. 避免所有網路上的包都被接收進來處理, 降低系統效能. NUC980 系列帶有 16 組 ACM, 其中 13組(CAM0~CAM12)可實際用來比對位址, 剩下的3 組 (CAM13~CAM15)則是保留給控制封包使用. 要使能 CAM, ECMP (EMAC_CAMCMR[4]) 這個總開關需要被設 1. 接著設定要接收的 MAC 地址到 CAM0~CAM12 其中一組的地址寄存器中. 例如本機的網口地址為 00:00:00:59:16:88, 則將 EMAC_CAM0M 設定為 0x00000059, 將 EMAC_CAM0L 設定為 0x1688000000. 最後, 將 CAM0EN(EMAC_CAMEN[0]) 設定為 1, 啟動 CAM0 的功能.

若是要接收廣播封包, 可以挪用其中一組 CAM, 將 EMAC_CAMxM, EMAC_CAMxL 分別設置成 0xFFFFFFFF, 0xFFFF0000, 使能 CAMxEN, 或是直接將 ABP (EMAC_CAMCMR[2]) 設為 1, 即可接收廣播封包.

接收組播封包的方法類似, 可以挪用其中一組 CAM, 將 EMAC_CAMxM, EMAC_CAMxL 設置成組播封包對應的網路地址, 使能 CAMxEN, 或是直接將 AMP (EMAC_CAMCMR[1]) 設為 1. 第一種方式只會接收特定地址的組播封包, 第二個方式則是會將網路上所有的組播封包都收下來.

網口也可以透過設置 CAM 寄存器 進入混雜模式 (Promiscuous Mode), 在這種模式下, 不論封包的目的地只是誰, 全部都會被收下來. 較進入這個模式, 只需將 AUP (EMAC_CAMCMR[0]), AMP (EMAC_CAMCMR[1]), 以及 ABP (EMAC_CAMCMR[2]) 都設為 1, 即可工作在混雜模式.

19.4.3 控制封包

IEEE 802.3 定義了用於流控的暫停封包. NUC980 支持傳送暫停封包, 也可以接生暫停封包. 將 ACP (EMAC_MCMDR[3]) 置 1 後, 即可接收控制封包. 收到控制封包後, EMAC 的封包傳送會暫停指定的時間. 在暫停的時間內, PAU(EMAC_MGSTA[12]) 會被置 1, 之後自動清 0. 若是 CFRIEN (EMAC_MIEN[14]) 為 1, 則接收到控制封包的同時, 中斷會被觸發, 且 CFR (EMAC_MISTA[14]) 會被置 1.

若是要傳送暫停封包, 將 01:80:C2:00:00:01 這個地址填進 EMAC_CAM13M 以及 EMAC_CAM13L 寄存器, 將本機的網路地址填進 EMAC_CAM14M 以及 EMAC_CAM14L 寄存器. 將0x88080001 填進EMAC_CAM15M, 將暫停時間填入 OPERAND(EMAC_CAM15L[31:24]). 暫停時間是以 512 比特時間為單位. 最後, 將SDPZ (EMAC_MCMDR[16]) 寫1, 即可傳送暫停封包. 當傳送完成後, SDPZ 會自動清 0.

注意: 只能在全雙功模式下使用暫停封包做流控.

19.4.4 遠端喚醒 (WoL)

NUC980 支持遠端喚醒的功能. 當接收到魔術封包後, 可以將系統由休眠模式中喚醒. 魔術封包的格式是定義在一份AMD 所出的白皮書之中. 魔術封包的的格式是封包任意位置有連續六字節的 0xFF, 緊接著重複 16 次共 102 字節的 MAC 地址. 當 MGPWAKE(EMCA_MCMDR[6]) 置 1, 以及 WOLIEN (EMAC_MIEN[15]) 為 1, 在休眠狀態中接收到魔術封包, 且其中重複 16 次的 MAC 地址與 CAM0 中的設置吻合, 則會喚醒系統, 並將 MGPR (EMAC_MISTA[15]) 置 1. MGPR 可透過軟體寫 1 的方式清 0.

19.4.5 封包接收

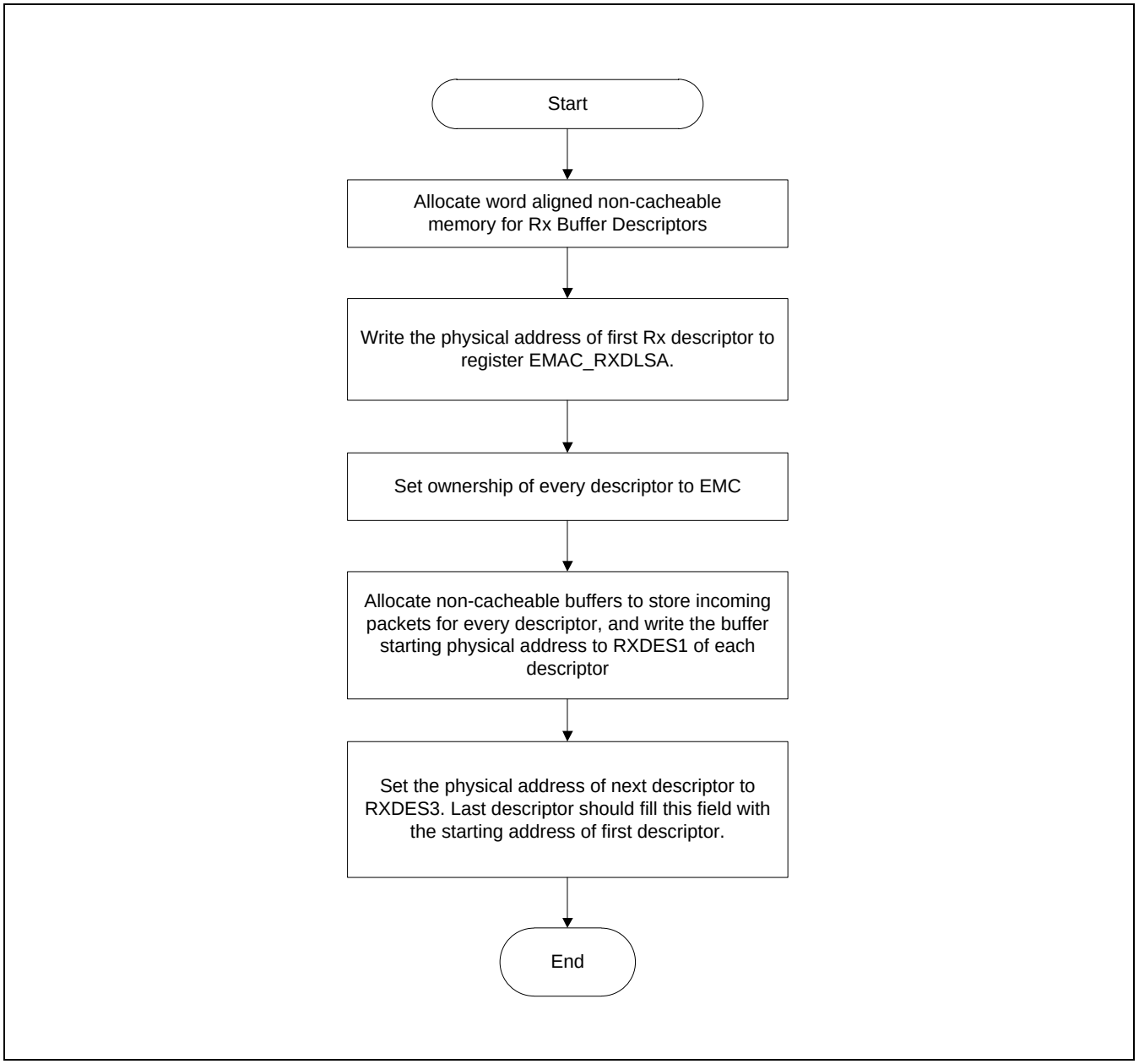
網路封包的接收, 是透過預先定義好格式的接收描述符來進行. 驅動程式需要預先準備好一些接收描述符, 當 CAM 決定偵測到的封包需要接收後, 封包就會接收至描述符所指定的內存地址, 然後封包的狀態以及長度會被儲存至描述符中, 接著乙太網路控制器會使用下一個接收描述符來接收封包. 所以CPU 以及 EMAC 透過接收描述符彼此交換接收封包的資訊.

每個接收描述符佔了四個 32位字組, 所有的接收描述符會組成一個單向循環鏈表, 下表顯示的是每個描述符的結構. 描述符字組RXDES0 的最高位 RXDES0[31] 記錄這描述符由誰使用. 當設成 1 時, 代表此描述符可由 EMAC 使用, EMAC 會依著 RXDES1 的指針擺放接收到的封包, 並將封包的長度放到 RXDES0 [15:0], 將接收到的封包狀態, 例如有無錯誤等… 顯示在 RXDES0 [30:16] 的標誌位內. 並將 RXDES0[31] 清為 0, 表示此描述符有收到封包. 最後, EMAC 會順著 RXDES3 指針找到下一個描述符. 若是下一個描述符的 RXDES0[31] 為 0, 代表所有的接收描述符都已被使用, 此時, EMAC 會停止接收狀態機, 直到描述符被釋放, 並重新啟動狀態機為止.

	31		15	0
RXDES 0	OWN	Receive Frame Status	Receive Frame Byte Count	
RXDES 1	Receive Frame Buffer Starting Address / Time Stamp Least Significant 32-Bit			
RXDES 2	Reserved			
RXDES 3	Next RxDMA Descriptor Starting Address / Time Stamp Most Significant 32-Bit			

若是網路時鐘的功能有使能, 接收到封包的同時, RXDES1, 以及 RXDES3 會記錄收到封包的時間供上層軟體計算之用. 所以若是在處理完封包, 要重新使用這個描述符的話, 除了 RXDES0[31] 需要重新設置為 1, 驅動程式要在這之前先將 RXDES1, RXDES3 填回正確的指針值. 也就是說, 驅動程式需要在其他位置備份這兩個指針, 以便之後回填.

當在內存中的接收描述符初始化完成後, 需要將第一個描述符的位址填進EMAC_RXDLSA 寄存器, 通知 EMAC描述符的所在. 之後將 RXON(EMAC_MCMDR[0]) 設 1, 並填寫任意值進 EMAC_RSDR 寄存器, 即可起始接收狀態機, 開始接收封包. 以下流程顯示著初始化接收描述符的步驟.

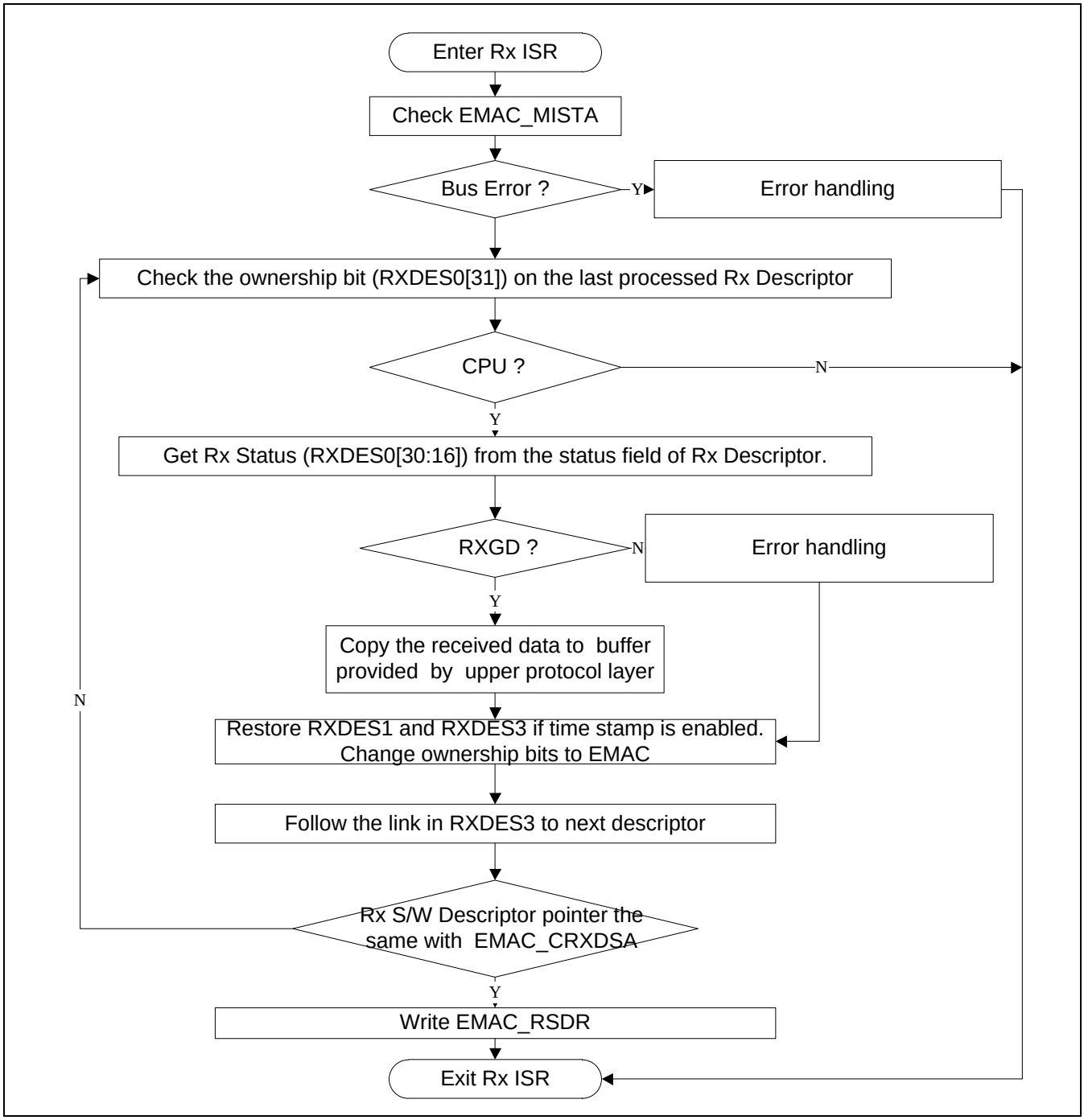


在此的範例在初始化描述符的同時, 預留了空間保存 RXDES1, RXDES3 的值, 在被時間戳記覆蓋後, 可以回填.

```
typedef struct _emac_descriptor
{
    unsigned int  rxdes0;
    unsigned int  rxdes1;
    unsigned int  rxdes2;
    unsigned int  rxdes3;
    // for backup descriptor fields over written by time stamp
```

```
    unsigned int  backup0;
    unsigned int  backup1;
} rx_descriptor;

#define RX_DESC_SIZE      4        // Number of Rx Descriptors
#define RX_BUF_SIZE            1518 // MAX Ethernet packet size

rx_descriptor rx_desc[RX_DESC_SIZE];
unsigned char rx_buf[RX_DESC_SIZE][ RX_BUF_SIZE];

void rx_desc_init(void)
{
    unsigned int i;

    for(i = 0; i < RX_DESC_SIZE; i++) {
        rx_desc[i].rxdes0 = (1 << 31);
        rx_desc[i].rxdes1 = (unsigned int)(&rx_buf[i][0]);
        rx_desc[i].backup0 = rx_desc[i].rxdes1;
        rx_desc[i].rxdes2 = 0;
        rx_desc[i].rxdes3 = (unsigned int)&rx_desc[(i + 1) % RX_DESC_SIZE];
        rx_desc[i].backup1 = rx_desc[i].rxdes3;
    }

    // Set Frame descriptor's base address.
    EMAC_RXDLSA = (unsigned int)&rx_desc[0];

}
```

網路封包的接收可透過輪詢模式或是中斷模式達成. 若是使用輪詢模式, 須檢測 RXGD (EMAC_MISTA[4]) 標誌位. 每當這個位被置 1, 代表至少有一個封包透過描述符接收進內存. 若是使用中斷模式, 則需要將 RXGDIEN (EMAC_MIEN[4]) 以及 RXIEN(EMAC_MIEN[0]) 都置 1. 每當有封包接收, 就會觸發中斷, 並將 RXGD 以及 RXINTR(EMAC_MISTA[0]) 均置 1. RXGD 以及 RXINTEN 都可透過寫 1 的方式清除. 以下的流程圖說明了在 RXGD 被置 1 之後的處理方式. 若是在中斷模式接收, 那這就是中斷處理函數中所需做的處理.



以下是假設時間戳記使能的接收中斷程式的範例，若是沒有使能時間戳記，將指針復原的部分移除即可.

```
void RX_IRQHandler(void)
{
    rx_descriptor *desc;
    unsigned int status, len, reg;
```

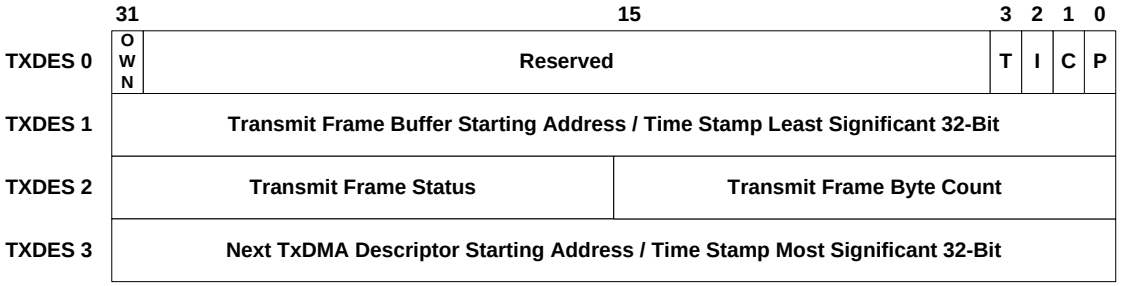
```
    reg = EMAC_MISTA;

    // Get last Rx Descriptor
    desc = (rx_descriptor *)current_rx_desc;
    do {

        if(EMAC0->CRXDSA == (unsigned int)desc)
            break;

        if((desc->rxdes0 | (1<<31)) == (1<<31)) { // ownership=CPU
            status = (desc->rxdes0 >> 16) & 0xffff;

            // If Rx frame is good, then process received frame
            if(status & RXFD_RXGD) {
                len = desc->rxdes0 & 0xffff;
                recv_pkt(desc->backup0, len);
            } else {
                // error handling
            }
        } else
            break;

        if(status & RTSAS) {
            // store time stamp
            log_time_stamp(desc->rxdes1, desc->rxdes3);
        }

        // restore descriptor link list
        desc->rxdes1 = desc->backup0;
        desc->rxdes3 = desc->backup1;

        // Change ownership to EMAC for next use
        desc->rxdes0 |= (1 << 31);
        // Get Next Frame Descriptor pointer to process
        desc = (mac_descriptor *)desc->rxdes3;
    } while (1);

    // store last processed descriptor. Next interrupt needs it
    current_rx_desc = (unsigned int)desc;

    // Trigger Rx
```

```
    EMAC_RDSR = 0;
    // Clear Rx related interrupt status
    EMAC_MISTA = reg & 0x0000ffff;
}
```

19.4.6 封包傳送

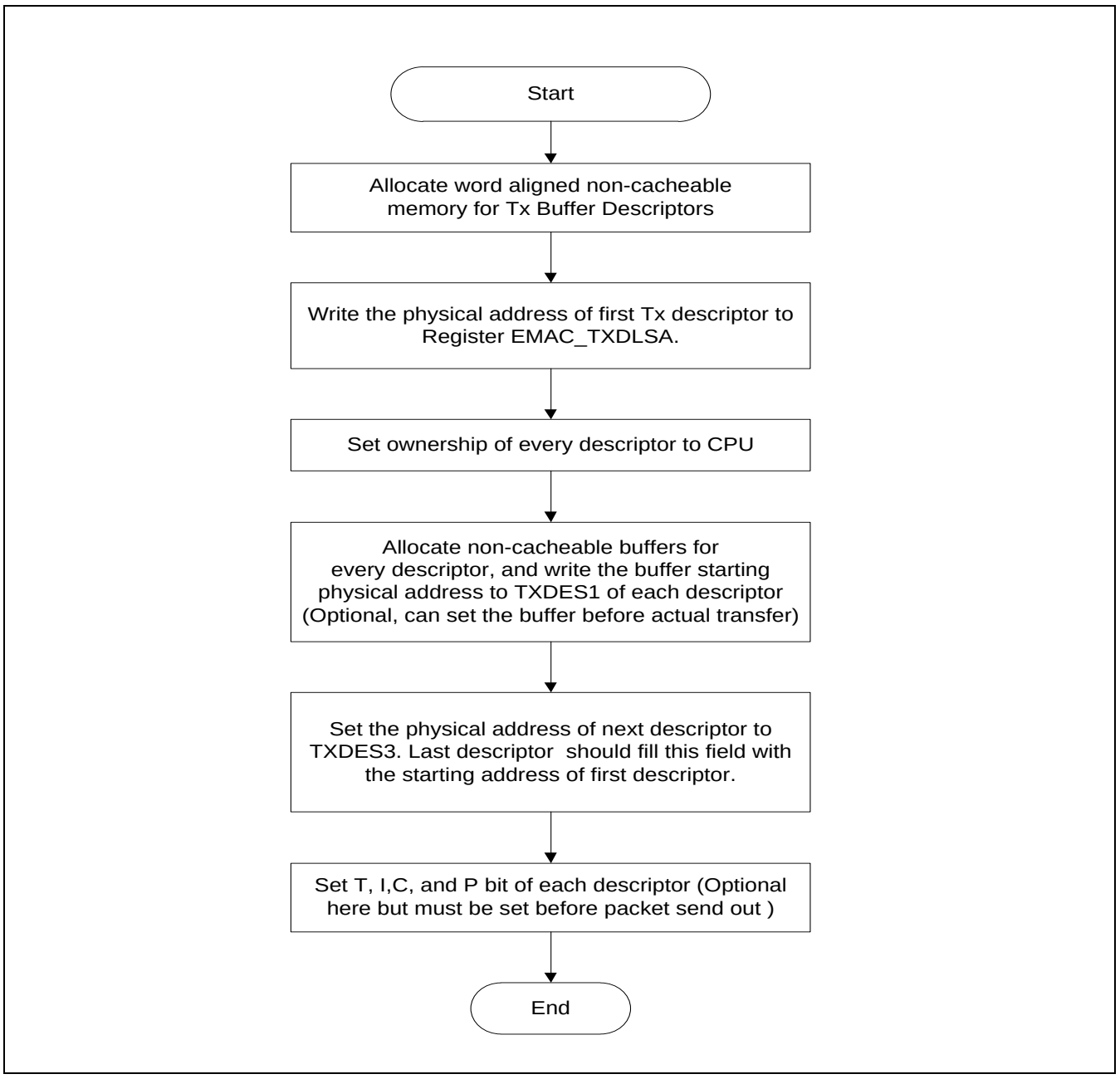
網路封包的傳送與接收相同, 是透過預先定義好格式的描述符來進行. 驅動程式需要預先準備好一些傳送述符, 當決定要傳送封包出去時, 將封包放在傳送描述符所指定的內存地址, 然後填寫封包的長度, 觸發傳送狀態機, 乙太網路控制器就會開始傳送此封包, 傳送完成後, 接著乙太網路控制器會使用下一個傳送描述符來傳送封包. 所以CPU 以及 EMAC 透過傳送描述符彼此交換傳送封包的資訊.

每個傳送描述符佔了四個 32位字組, 所有的傳送描述符會組成一個單向循環鏈表, 下表顯示的是每個描述符的結構. 描述符字組TXDES0 的最高位 TXDES0[31] 記錄這描述符由誰使用. 當設成 1 時, 代表此描述符可由 EMAC 使用, EMAC 從 TXDES1 的指針擺位址, 開始送出長度為 TXDES2[15:0] 個字節的封包, 並將傳送的結果, 例如有無錯誤等… 顯示在 RXDES2 [30:16] 的標誌位內. 並將 TXDES0[31] 清為 0, 表示此描述符的封包處理完成. 最後, EMAC 會順著 TXDES3 指針找到下一個描述符. 若是下一個描述符的 TXDES0[31] 為 0, 代表所有的傳送描述符都已處理完成, 沒有封包需要傳送, 此時, EMAC 會停止傳送狀態機. 若是TXDES0[31] 為 1, 則 EMAC 會繼續重複傳輸封包的步驟.



若是網路時鐘的功能有使能, 傳送完封包的同時, TXDES1, 以及 TXDES3 會記錄收到封包的時間供上層軟體計算之用. 所以若是要重新使用這個描述符傳送封包的話, 除了 TXDES0[31] 需要重新設置為 1, 驅動程式要在這之前先將 TXDES1, TXDES3 填回正確的指針值. 也就是說, 驅動程式需要在其他位置備份這兩個指針, 以便之後回填.

當在內存中的傳送描述符初始化完成後, 需要將第一個描述符的位址填進EMAC_TXDLSA 寄存器, 通知 EMAC 描述符的所在. 之後將 TXON(EMAC_MCMDR[0]) 設 1, 並填寫任意值進 EMAC_TSDR 寄存器, 即可起始傳送狀態機, 開始傳送封包. 以下流程顯示著初始化傳送描述符的步驟.

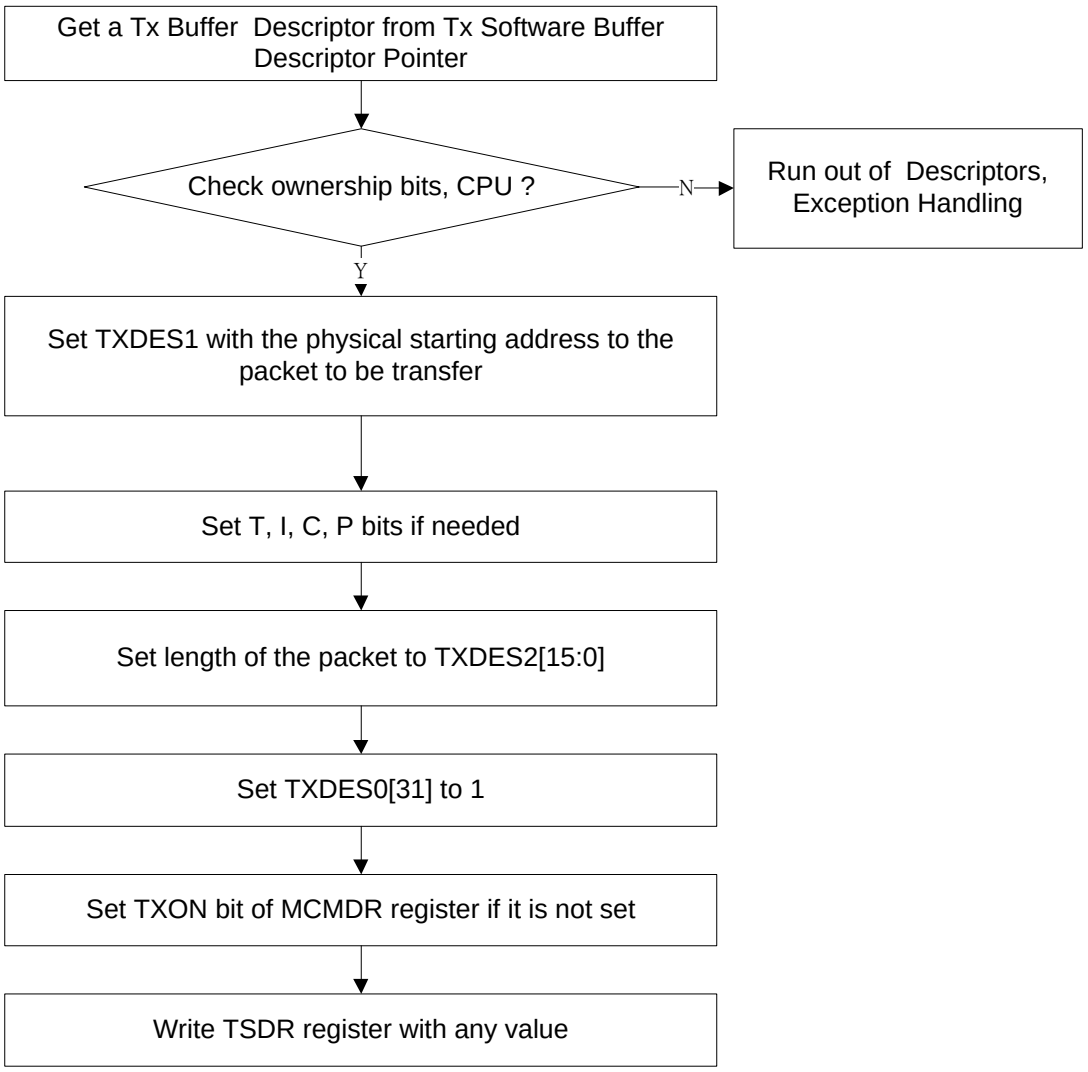


在此的範例在初始化描述符的同時, 預留了空間保存 TXDES1, TXDES3 的值, 在被時間戳記覆蓋後, 可以回填.

```
typedef struct _emac_descriptor
{
    unsigned int  txdes0;
    unsigned int  txdes1;
    unsigned int  txdes2;
    unsigned int  txdes3;
```

```
    // for backup descriptor fields over written by time stamp
    unsigned int  backup0;
    unsigned int  backup1;
} tx_descriptor;

#define TX_DESC_SIZE      4        // Number of Tx Descriptors

tx_descriptor tx_desc[TX_DESC_SIZE];

void tx_desc_init(void)
{
    unsigned int i;

    for(i = 0; i < TX_DESC_SIZE; i++) {
        tx_desc[i].txdes0 = (1 << 31);
        tx_desc[i].txdes1 = 0;
        tx_desc[i].backup0 = tx_desc[i].txdes1;
        tx_desc[i].txdes2 = 0;
        tx_desc[i].txdes3 = (unsigned int)&tx_desc[(i + 1) % TX_DESC_SIZE];
        tx_desc[i].backup1 = tx_desc[i].rxdes3;
    }

    // Set Frame descriptor's base address.
    EMAC_TXDLSA = (unsigned int)&tx_desc[0];

}
```

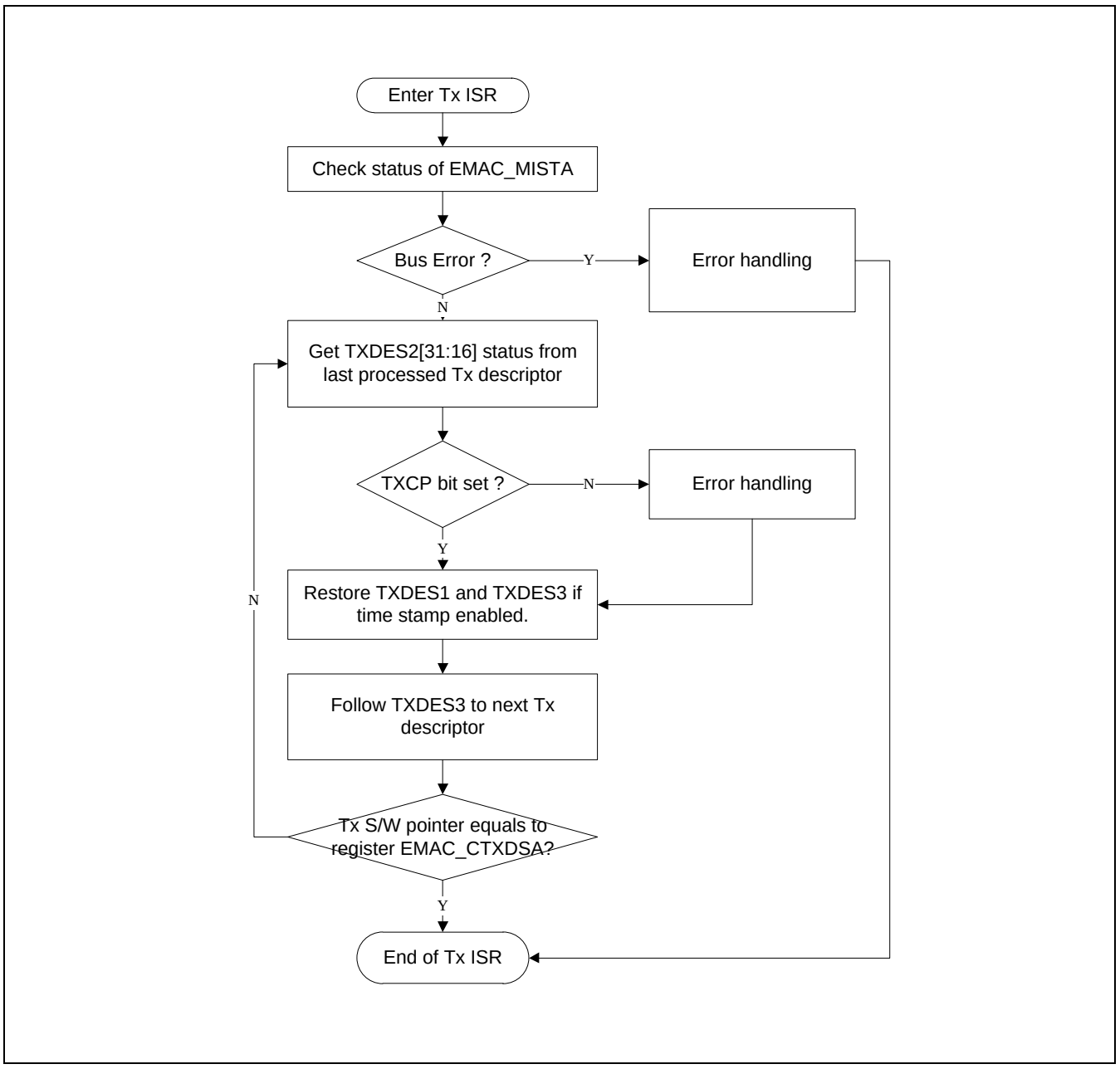
當要傳送封包時, 除了先前提的的一些傳送描述符控制位需要設置外, 再傳送前 TXDES0[3:0] 也需要做相對應的設置. TTSEN(TXDES0[3]) 用來使能傳輸時間戳記, 若是此位置 1, 則封包傳送完成的時間會被記錄在 TXDES1 以及 TXDES3. INTEN(TXDES0[2]) 控制封包傳輸完成後是否要觸發中斷. 必須要這個控制為以及 EMAC_MISTA 寄存器裡的設定都是使能接收中斷, 傳送完成後才會觸發中斷. CRCAPP(TXDES0[1]) 控制是否須由 EMAC 幫忙運算封包的 CRC 校驗值並幫忙寄出, 正常情況須設定為 1. PADEN (TXDES0[1]) 控制當封包長度不足 60 位時, 是否由 EMAC 自動補足, 以符合乙太網路的規範. 正常情況下, 此位必須置 1. 以下是封包傳送的流程圖以及使能網路時鐘的範例程式.




```
int send_pkt(unsigned char *data, unsigned int size)
{
    tx_descriptor *desc;
```

```
    unsigned int status;

    // Get Tx frame descriptor & data pointer
    desc = (tx_descriptor *)next_tx_desc;

    status = desc->txdes0;

    // Check ownership, return if owner is EMAC
    if(status & (1 << 31))
        return -1;
    // Fill data pointer
    desc->txdes1 = (unsigned int)(data);

    // Set TX Frame flag & Length Field
    desc->txdes0 |= (P | C | I | T);
    desc->txdes2 = size;

    // Cheange ownership to DMA
    desc->txdes0 |= (1 << 31);

    // Find next Tx descriptor, do it here before time stamp update pointers
    next_tx_desc = desc->txdes3;
    // Trigger TX
    EMAC_TDSR = 0;

    Return 0;
}
```

網路封包的傳送結果可透過輪詢模式或是中斷模式得知. 若是使用輪詢模式, 須檢測 TXCP (EMAC_MISTA[18]) 標誌位. 每當這個位被置 1, 代表至少有一個封包透過描述符傳送完成. 若是使用中斷模式, 則需要將 TXCPIEN (EMAC_MIEN[18]) 以及 TXIEN(EMAC_MIEN[16]) 都置 1. 每當有封包傳送完成, 就會觸發中斷, 並將 TXCP 以及 TXINTR(EMAC_MISTA[16]) 均置 1. TXCP以及 TXINTR都可透過寫 1 的方式清除. 以下的流程圖說明了在 TXCP被置 1 之後的處理方式. 若是在中斷模式下, 那這就是中斷處理函數中所需做的處理.



以下是開啟網路時鐘的傳送中斷處理函數範例.

```
void TX_IRQHandler(void)
{
    tx_descriptor *desc;
    unsigned int status, reg;
    unsigned int last_tx_desc;
```

```
    reg = EMAC0->MISTA;
    // Time stamp alarm interrupt
    if(reg & MISTA_TSALS) {
        // Do something here
    }
    // Clear Tx related interrupt flags
    EMAC0->MISTA = reg & 0xffff0000;

    last_tx_desc = EMAC_CTXDSA;
    desc = current_tx_desc;

    while (last_tx_desc != (unsigned int)desc) {
    // we have packet to process
        status = desc->txdes2 >> 16;
        if (status & TXCP) {
            // Success.
        } else {
            // Failed, error handling
        }
        if(status & TTSAS) {
            // process time stamp
            log_time_stamp(desc->txdes1, desc->txdes3);
        }

        // restore descriptor link list and data pointer
        desc->txdes1 = desc->backup0;
        desc->txdes3 = desc->backup1;

        // find next Tx descriptor
        desc = (mac_descriptor *)desc->txdes3;
    }
    // store last processed descriptor. Next interrupt needs it
    current_tx_desc = (unsigned int)desc;

}
```

19.4.7 網路時鐘

為了使 IEEE1588 在 NUC980 上運行能得到更精準的時鐘, 在兩個乙太網口都有件網路時間模塊.

可自動幫忙在接收或是傳送封包時, 加上時間, 減少軟體之後讀取時間再來運算的誤差. 時間的更新是每個網口時鐘周期都會做一次, 最快每秒會更新 150M 次, 可說是系統中最精密的時鐘. 時間的更新共支持了兩種方式, 粗略更新以及精細更新. 可透過 TSMODE (EMAC_TSCTL[2]) 控制. 清 0 時使用粗略更新, 置 1 時使用精細更新.

在網路時間的運算上, 可分為秒級 (second) 的運算以及次秒級 (sub-second)的運算. 每當次秒溢位時, 時間就加上一秒. 使用粗略更新時, 每個網口時鐘周期, 次秒時間都會加上儲存在 EMAC_TSINC 寄存器的值.

我們在此試算 EMAC 時鐘頻率為 150MHz 下, 粗略更新的時間戳記相關寄存器的設定. 當時鐘是 150MHz 時, 次秒每經過 150M 個週期就需要溢位1 秒. 次秒寄存器 (EMAC_TSSUBSEC) 共有 31 位, 所以 EMAC_TSINC 要填入 (2^{31}) / 150M = 14.31 ~= 14 = 0x0E, 這就是每個時間週期次秒所需要增加的值. 但填入 0x0E 的話, 時鐘誤差會有 0.31/14 = 2.2%. 由此可知, 當 EMAC 時鐘頻率為 150MHz 下, 粗略更新的誤差非常的大, 並不適合使用.

若是使用精細更新, 則每個時鐘週期, 有一個 32 位的累加器會不斷加上存在 EMAC_TSADDNED 寄存器中的值, 每當累加器溢位, 次秒時間會加上儲存在 EMAC_TSINC 寄存器的值. 因此使用精細更新的結果會比粗略更新準確.

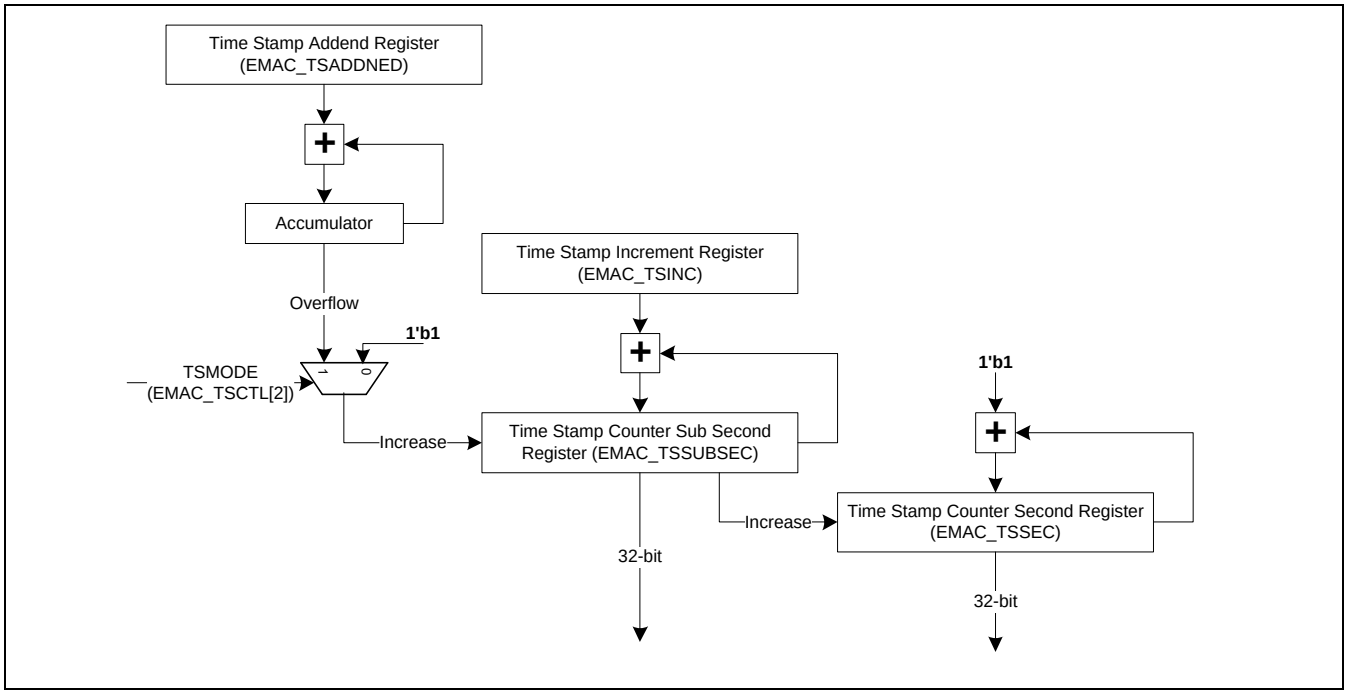
試算EMAC 時鐘頻率為 150MHz 下, 精細更新的時間戳記相關寄存器的設定. 假設我們希望每 100 奈秒要對次秒寄存器加上一次 EMAC_TSINC 的值. 代表累加 10^7 次之後, 次秒需要溢位, 所以 EMAC_TSINC 要填入 2^{31} / 10^7 = 214.71 ~= 215 = 0xD7. 此時, 實際進位的頻率不是希望的 10^7 Hz, 而是 2^{31} / 215 Hz. 由此反推回 EMAC_TSADDNED 所需填的值必須能讓累加器在2^{31} / 215 Hz 的頻率下溢位, 時間才會準確. 也就是EMAC_TSADDNED 要填入 2^{32} * (2^{31} / 215) / 150M = 285996032.15 ~= 285996032 = 0x110BF400. 實際誤差只有 5.26 * 10^{-10}, 相較粗略更新準確很多. 建議正常使用時, 選擇使用精細更新.

依剛才的試算結果, 當 EMAC 時鐘頻率為 150MHz 下, 精細更新的設置可以如下 (設置不同於粗略更新, 不是唯一. 選擇不同的 EMAC_TSINC 需要使用不同的 EMAC_TSADDNED):

EMAC_TSINC = 0xD7;

EMAC_TSADDEND = 0x110BF400;

下圖顯示了網路時間更新的結構圖.



在網路時間次秒級的運算上, 每次溢位 (第 31 位為 1), 代表時間過了一秒. 也就是每 2^{31} 個次秒等於 1 秒, 也等於 10^9 個奈秒. 以下兩隻函數正是依據這個公式在次秒與奈秒之間做轉換.

```
static unsigned int subsec2nsec(unsigned int subsec)
{
    // 2^31 subsec == 10^9 ns
    unsigned long long i;
    i = 1000000000ll * subsec;
    i >>= 31;
    return(i);
}

static unsigned int nsec2subsec(unsigned int nsec)
{
    // 10^9 ns =  2^31 subsec
    unsigned long long i;
    i = (1ll << 31) * nsec;
    i /= 1000000000;
    return(i);
}
```

初始化網路時鐘的步驟如下:

1. 將TSEN(EMAC_TSCTL[0]) 設 1 始能網路時鐘模塊.
2. 將初始秒及次秒填入 EMAC_TSSEC 以及 EMAC_TSSUBSEC 寄存器
3. 設置 EMAC_TSINC寄存器, 若是使用精細更新, 也須設置 EMAC_TSADDEND寄存器.
4. 將 TSIEN (EMAC_TSCTL[1]) 設 1 開始網路時鐘計時, 若要使用精細更新, 一併將 TSMODE (EMAC_TSCTL[2]) 置 1.

依照 IEEE 1588 規範, 當同一裝置有多個網口時, 必須能共用時鐘. 所以若是兩個乙太網口同時開啟網路時鐘功能, 必須要將 EMAC1 的 PTP_SRC (EMAC_MCMDR[7]) 置 1, 讓 EMAC1 使用 EMAC0 內部的時鐘模塊而不要用自己的.

軟體可以透過寄存器讀取當前網路時鐘的時間. 當前網路時間共用了兩個 32 位寄存器, EMAC_TSSEC 以及 EMAC_TSSUBSEC 表示. 為了避免讀取時正好次秒寄存器溢位, 這兩個寄存器在讀取時有做了保護線路. 只要軟體先讀取 EMAC_TSSUBSEC, 則當前的秒值會被鎖定在 EMAC_TSSEC 寄存器內, 避免產生誤判. 以下是讀取當前時間的範例:

```
unsigned int s, subs;

// Read sub second first.
subs = EMAC_TSSUBSEC;
s = EMAC_TSSEC;

printf("current time is %d second %d nano-second\n", s, subsec2nsec(subs));
```

在初始化設置時鐘後, 當前的網路時間可透過軟體調整. 為了維持時間的精準度, 時間的調整是透過偏移量的方式來調整. 若是知道當前時間快了3 秒, 調整方式並不是讀回當前時間, 減去 3 秒, 再回填. 這樣會有不可掌握的軟體執行時間帶來誤差, 對於要求精準的 PTP 有不好的影響. 實際的方式是填寫寄存器觸發網路時間模塊, 加或是減一個偏移量. 秒級偏移量填進 EMAC_UPDSEC 寄存器, 次秒級的偏移量則是填到 EMAC_UPDSUBSEC[30:0]. 若是正偏移量, 將 EMAC_UPDSUBSEC[31] 清 0, 負偏移量則將 EMAC_UPDSUBSEC[31] 置 1. 之後再將 TSUPDATE(EMAC_TSCTL[3]) 置1, 即可觸發觸發網路時間模塊更新時間. 當更新完成後, TSUPDATE(EMAC_TSCTL[3]) 會自動清 0.

網路時間模塊也支援了鬧鐘的功能. 觸發鬧鐘的時間填在兩個寄存器 EMAC_ALMSEC 以及 EMAC_ALMSUBSEC 內. 它們分別儲存著觸發鬧鐘的秒及次秒. 將這兩寄存器設置好後, 將 TSALMEN (EMAC_TSCTL[5]) 置 1, 即啟動了鬧鐘功能. 若是要使能鬧鐘中斷, 將 TSALMIEN (EMAC_MIEN[28]) 設 1, 則當鬧鐘觸發時, 將產生中斷, 並將 TSALS(EMAC_MISTA[28]) 置 1. 軟體可以將此位寫 1 清 0. 請注意, 這個中斷是規在 TX 中斷, 所以須在 TX 中斷處理函數中攔截, 而不是 RX 中斷處理函數.

19.4.8 錯誤處理

寄存器 EMAC_MISTA 裡的一些標誌位可以反映一些在 網路運行上發生的錯誤狀態. 以下列出

了發生錯誤時, 可做的檢查或是解決方式.

錯誤位名稱	比特	狀態描述	檢查或是解決方式
TXBERR	24	傳送總線錯誤	檢查驅動程式. 造成本錯誤的原因只可能是傳送描述符帶有不合法的指針. 正確的程式設計不會造成這個問題.
TDU	23	傳送描述符不足	不須特別處理. 這個標誌位表示乙太網路控制器已經將所有要傳送的封包送出.
TXABT	21	傳送退出	很可能是由於網路踏繁忙造成.
TXEMP	17	傳送FIFO 不足	若是這個標誌位頻繁的被設起, 可將 TXTHD(EMAC_FFTCR[9:8]) 調到較高的觸發准位.
RXBERR	11	接收總線錯誤	檢查驅動程式. 造成本錯誤的原因只可能是傳送描述符帶有不合法的指針. 正確的程式設計不會造成這個問題.
RDU	10	接收描述符不足	這標誌位被置1 的原因是軟體沒有及時地將收到的封包讀走, 導致接收描述符都已被收到的封包佔滿. 造成 EMAC 無法再接收任何封包. 若要繼續接收封包, 需要將掛上收描述符的封包都讀走, 收描述符的所有權設置成 EMAC, 並在 EMAC_RSDR 寄存器填寫任意值, 啟動接收狀態機. 觸發的原因有可能是網路真的有大量封包同時湧現, 但也有可能是整個系統中, 有中斷獨佔了太多時間, 造成接收中斷沒有機會執行.
RP	6	接收過短封包 (< 64 字節)	丟棄此封包即可. 正常操作不會發生. 除非 ARP (EMAC_MCMDR[2]) 被置 1.
ALIE	5	對其錯誤	正常操作中不應發生. 若是時常發生, 請檢查 PCB 版上 RMII 相關的走線, 或是更換網路線試試.
PTLE	3	接收過長封包 (> 1518 字節)	丟棄此封包即可. 正常操作不會發生. 除非 ALP (EMAC_MCMDR[1]) 被置 1.
RXOV	2	接收 FIFO 溢出	若是這個標誌位頻繁的被設起, 可將 RXTHD(EMAC_FFTCR[1:0]) 調到較高的觸發准位

CRCE	1	CRC 較驗錯誤	丟棄此封包即可. 正常操作不會發生. 除非 AEP (EMAC_MCMDR[4]) 被置 1.

19.5 寄存器

Register	Offset	R/W	Description	Reset Value
EMAC0_BA = 0xB001_2000 **EMAC1_BA = 0xB002_2000**				
EMAC_CAMCMR	EMAC_BA+0x000	R/W	CAM Command Register	0x0000_0000
EMAC_CAMEN	EMAC_BA+0x004	R/W	CAM Enable Register	0x0000_0000
EMAC_CAM0M	EMAC_BA+0x008	R/W	CAM0 Most Significant Word Register	0x0000_0000
EMAC_CAM0L	EMAC_BA+0x00C	R/W	CAM0 Least Significant Word Register	0x0000_0000
EMAC_CAM1M	EMAC_BA+0x010	R/W	CAM1 Most Significant Word Register	0x0000_0000
EMAC_CAM1L	EMAC_BA+0x014	R/W	CAM1 Least Significant Word Register	0x0000_0000
EMAC_CAM2M	EMAC_BA+0x018	R/W	CAM2 Most Significant Word Register	0x0000_0000
EMAC_CAM2L	EMAC_BA+0x01C	R/W	CAM2 Least Significant Word Register	0x0000_0000
EMAC_CAM3M	EMAC_BA+0x020	R/W	CAM3 Most Significant Word Register	0x0000_0000
EMAC_CAM3L	EMAC_BA+0x024	R/W	CAM3 Least Significant Word Register	0x0000_0000
EMAC_CAM4M	EMAC_BA+0x028	R/W	CAM4 Most Significant Word Register	0x0000_0000
EMAC_CAM4L	EMAC_BA+0x02C	R/W	CAM4 Least Significant Word Register	0x0000_0000
EMAC_CAM5M	EMAC_BA+0x030	R/W	CAM5 Most Significant Word Register	0x0000_0000
EMAC_CAM5L	EMAC_BA+0x034	R/W	CAM5 Least Significant Word Register	0x0000_0000
EMAC_CAM6M	EMAC_BA+0x038	R/W	CAM6 Most Significant Word Register	0x0000_0000
EMAC_CAM6L	EMAC_BA+0x03C	R/W	CAM6 Least Significant Word Register	0x0000_0000
EMAC_CAM7M	EMAC_BA+0x040	R/W	CAM7 Most Significant Word Register	0x0000_0000
EMAC_CAM7L	EMAC_BA+0x044	R/W	CAM7 Least Significant Word Register	0x0000_0000
EMAC_CAM8M	EMAC_BA+0x048	R/W	CAM8 Most Significant Word Register	0x0000_0000
EMAC_CAM8L	EMAC_BA+0x04C	R/W	CAM8 Least Significant Word Register	0x0000_0000
EMAC_CAM9M	EMAC_BA+0x050	R/W	CAM9 Most Significant Word Register	0x0000_0000
EMAC_CAM9L	EMAC_BA+0x054	R/W	CAM9 Least Significant Word Register	0x0000_0000
EMAC_CAM10M	EMAC_BA+0x058	R/W	CAM10 Most Significant Word Register	0x0000_0000

EMAC_CAM10L	EMAC_BA+0x05C	R/W	CAM10 Least Significant Word Register	0x0000_0000
EMAC_CAM11M	EMAC_BA+0x060	R/W	CAM11 Most Significant Word Register	0x0000_0000
EMAC_CAM11L	EMAC_BA+0x064	R/W	CAM11 Least Significant Word Register	0x0000_0000
EMAC_CAM12M	EMAC_BA+0x068	R/W	CAM12 Most Significant Word Register	0x0000_0000
EMAC_CAM12L	EMAC_BA+0x06C	R/W	CAM12 Least Significant Word Register	0x0000_0000
EMAC_CAM13M	EMAC_BA+0x070	R/W	CAM13 Most Significant Word Register	0x0000_0000
EMAC_CAM13L	EMAC_BA+0x074	R/W	CAM13 Least Significant Word Register	0x0000_0000
EMAC_CAM14M	EMAC_BA+0x078	R/W	CAM14 Most Significant Word Register	0x0000_0000
EMAC_CAM14L	EMAC_BA+0x07C	R/W	CAM14 Least Significant Word Register	0x0000_0000
EMAC_CAM15M	EMAC_BA+0x080	R/W	CAM15 Most Significant Word Register	0x0000_0000
EMAC_CAM15L	EMAC_BA+0x084	R/W	CAM15 Least Significant Word Register	0x0000_0000
EMAC_TXDLSA	EMAC_BA+0x088	R/W	Transmit Descriptor Link List Start Address Register	0xFFFF_FFFC
EMAC_RXDLSA	EMAC_BA+0x08C	R/W	Receive Descriptor Link List Start Address Register	0xFFFF_FFFC
EMAC_MCMDR	EMAC_BA+0x090	R/W	MAC Command Register	0x0040_0000
EMAC_MIID	EMAC_BA+0x094	R/W	MII Management Data Register	0x0000_0000
EMAC_MIIDA	EMAC_BA+0x098	R/W	MII Management Control and Address Register	0x0000_0000
EMAC_FFTCR	EMAC_BA+0x09C	R/W	FIFO Threshold Control Register	0x0000_0000
EMAC_TSDR	EMAC_BA+0x0A0	W	Transmit Start Demand Register	Undefined
EMAC_RSDR	EMAC_BA+0x0A4	W	Receive Start Demand Register	Undefined
EMAC_DMARFC	EMAC_BA+0x0A8	R/W	Maximum Receive Frame Control Register	0x0000_0800
EMAC_MIEN	EMAC_BA+0x0AC	R/W	MAC Interrupt Enable Register	0x0000_0000
EMAC_MISTA	EMAC_BA+0x0B0	R/W	MAC Interrupt Status Register	0x0000_0000
EMAC_MGSTA	EMAC_BA+0x0B4	R/W	MAC General Status Register	0x0000_0000
EMAC_MPCNT	EMAC_BA+0x0B8	R/W	Missed Packet Count Register	0x0000_7FFF
EMAC_MRPC	EMAC_BA+0x0BC	R	MAC Receive Pause Count Register	0x0000_0000
EMAC_DMARFS	EMAC_BA+0x0C8	R/W	DMA Receive Frame Status Register	0x0000_0000
EMAC_CTXDSA	EMAC_BA+0x0CC	R	Current Transmit Descriptor Start Address Reg.	0x0000_0000
EMAC_CTXBSA	EMAC_BA+0x0D0	R	Current Transmit Buffer Start Address Register	0x0000_0000
EMAC_CRXDSA	EMAC_BA+0x0D4	R	Current Receive Descriptor Start Address Reg.	0x0000_0000
EMAC_CRXBSA	EMAC_BA+0x0D8	R	Current Receive Buffer Start Address Register	0x0000_0000
EMAC_TSCTL	EMAC_BA+0x100	R/W	Time Stamp Control Register	0x0000_0000

EMAC_TSSEC	EMAC_BA+0x110	R	Time Stamp Counter Second Register	0x0000_0000
EMAC_TSSUBSEC	EMAC_BA+0x114	R	Time Stamp Counter Sub Second Register	0x0000_0000
EMAC_TSINC	EMAC_BA+0x118	R/W	Time Stamp Increment Register	0x0000_0000
EMAC_TSADDEND	EMAC_BA+0x11C	R/W	Time Stamp Addend Register	0x0000_0000
EMAC_UPDSEC	EMAC_BA+0x120	R/W	Time Stamp Update Second Register	0x0000_0000
EMAC_UPDSUBSEC	EMAC_BA+0x124	R/W	Time Stamp Update Sub Second Register	0x0000_0000
EMAC_ALMSEC	EMAC_BA+0x128	R/W	Time Stamp Alarm Second Register	0x0000_0000
EMAC_ALMSUBSEC	EMAC_BA+0x12C	R/W	Time Stamp Alarm Sub Second Register	0x0000_0000

20 USB 設備控制器 (USBD Device Controller)

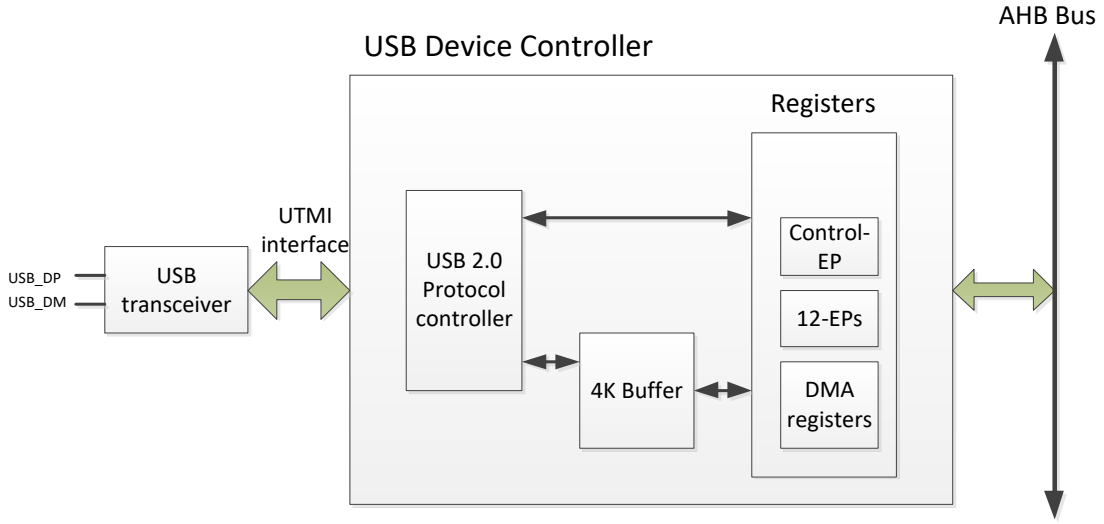
20.1 概述

USB設備控制器是AHB總線和UTMI總線的接口。USB控制器包含了AHB主接口和AHB從接口。CPU透過AHB從接口讀寫USB控制器寄存器。對於IN或OUT傳輸時，USB設備控制器需要通過AHB主接口將數據寫入存儲器，或從存儲器讀取數據。USB設備控制器符合USB2.0的規範，它含有12個可配置的endpoint，除了Control endpoint。這些端點可被配置成Bulk，Interrupt或Isochronous。USB設備控制器有一個內置的DMA，用以減輕CPU的負擔。

20.2 特性

- 兼容 USB 2.0 規格。
- 支持 12 個可配置端點，除了控制端點。
- 每個端點可以是 Isochronous，Bulk 或 Interrupt，搭配 IN 或 OUT 方向。
- 一個 IN 端點支持三種不同的操作模式 - 自動驗證模式，手動驗證模式，飛行模式。
- 支持 DMA 操作。
- 配置 4096 字節 RAM 供端點緩衝區使用。
- 支持端點最大數據包最大可達 1024 字節。

20.3 方塊圖



20.4 功能描述

USB設備控制器符合USB2.0規範，用戶可以設定模擬成一個大容量存儲卡讀卡器、虛擬COM端口等。詳細可以參考"USB Class Specification"。

USB設備控制器針對"IN-transfer"提供三種不同的操作模式：

- 自動驗證模式 – 當發送到主機的數據量等於最大數據包大小時，就可以選擇這個模式（例如 Bulk pipe transfer）。
- 手動驗證模式 – 這個模式需要 CPU 介入，當每次發送的數據量都不固定時，就可以選擇這個模式（例如 Interrupt pipe transfer）。
- 飛行模式 – 此模式最適合於同步數據傳輸，數據傳送的速度比包大小更重要（例如 Isochronous pipe transfer）。

下面會以USB大容量存儲設備（mass storage device）為範例，這個設備需要配置二個端點 – 端點A為Bulk IN，端點B為Bulk Out。

20.4.1 初始化 (Initialization)

初始化USB設備控制器的步驟：

1. 設置多功能控制 GPE11，SYS_GPE_MFPL 要填入 0x1。
2. 設置 CLK_HCLKEN 寄存器 USBD 位。
3. 設置 HSUSBD_PHYCTL 寄存器 PHYEN 位，始能 USB PHY。
4. HSUSBD_EPAMPS 寄存器填入 0x8，輪詢 HSUSBD_EPAMPS 寄存器，讀出的值是否同等於 0x8，用以確認 PHY 時鐘穩定。
5. 配置端點 A 為 Bulk-IN 類型、端點號為 1。
   (1) 設置 HSUSBD_EPARSPCTL 寄存器 MODE 位為 0，選擇自動驗證模式。
   (2) HSUSBD_EPAMPS 寄存器填入 512，表示最大數據包為 512 字節。
   (3) 設置 HSUSBD_EPACFG 寄存器 EPNUM 位為 1，EPDIR 位為 1，EPTYPE 位為 01，EPEN 位為 1。
   (4) HSUSBD_EPABUFSTART 寄存器填入 0x200，HSUSBD_EPABUFEND 寄存器填入 0x3FF，表示端點 FIFO 長度為 512 字節。
6. 配置端點 B 為 Bulk-Out 類型、端點號為 2。
   (1) 設置 HSUSBD_EPBINTEN 寄存器 RXPKIEN 位，始能接收數據中斷。
   (2) 設置 HSUSBD_EPBRSPCTL 寄存器 MODE 位為 0，選擇自動驗證模式。
   (3) HSUSBD_EPBMPS 寄存器填入 512，表示最大數據包為 512 字節。
   (4) 設置 HSUSBD_EPBCFG 寄存器 EPNUM 位為 2，EPDIR 位為 0，EPTYPE 位為 01，EPEN 位為 1。
   (5) HSUSBD_EPBBUFSTART 寄存器填入 0x400，HSUSBD_EPBBUFEND 寄存器填入 0x5FF，表示端點 FIFO 長度為 512 字節。
7. 設置 HSUSBD_GINTEN 寄存器 USBIEN、CEPIEN、EPAIEN、EPBIEN 位，始能 USB 控制端點、端點 A、端點 B 的中斷。

8. 設置 HSUSBD_BUSINTEN 寄存器 RSTIEN、RESUMEIEN、DMADONEIEN、VBUSDETIEN 位，始能 USB 復位、恢復、DMA 完成、插拔的中斷。
9. 設置 HSUSBD_OPER 寄存器 HISPDEN 位，始能高速模式。
10. 清除 HSUSBD_FADDR 寄存器。
11. 配置控制端點（端點 0）
    (1) HSUSBD_CEPBUFSTART 寄存器填入 0x0，HSUSBD_CEPBUFEND 寄存器填入 0x7F，表示端點 FIFO 長度為 128 字節。
    (2) 設置 HSUSBD_CEPINTEN 寄存器 SETUPPKIEN、STSDONEIEN 位，始能控制端點的安裝包和狀態完成中斷。
12. 輪詢 HSUSBD_PHYCTL 寄存器 VBUSDET 位，為 1 代表接上主機，設置 HSUSBD_PHYCTL 寄存器 DPPUEN 位，清除 SE0。

20.4.2 中斷處理程序

處理USBD控制器的中斷如下：

1. 讀取HSUSBD_GINTSTS寄存器和HSUSBD_GINTEN寄存器做屏蔽，可以得知是哪種中斷產生。
2. 讀取HSUSBD_BUSINTSTS寄存器和HSUSBD_BUSINTEN寄存器做屏蔽，如果相匹配就處理USB BUS的中斷。
3. 讀取HSUSBD_CEPINTSTS寄存器和HSUSBD_CEPINTEN寄存器做屏蔽，如果相匹配就處理控制端點的中斷。
4. 讀取HSUSBD_EPAINTSTS寄存器和HSUSBD_EPAINTEN寄存器做屏蔽，如果相匹配就處理端點A的中斷。
5. 讀取HSUSBD_EPBINTSTS寄存器和HSUSBD_EPBINTEN寄存器做屏蔽，如果相匹配就處理端點B的中斷。

20.4.3 Standard Request

USBD控制器處理標準請求的步驟如下：

1. 發生控制端點安裝包中斷。
2. 讀取 HSUSBD_SETUP1_0 、 HSUSBD_SETUP3_2 、 HSUSBD_SETUP5_4 、 HSUSBD_SETUP7_6寄存器，獲得請求和相關參數。
3. 分析請求。如果支持，清除NAK（設置HSUSBD_CEPCTL寄存器NAKCLR位），並且等待狀態完整；否則發送STALL給主機（設置HSUSBD_CEPCTL寄存器STALLEN位）。

20.4.4 Set Address Request

USBD控制器處理"Set Address"請求：

1. 發生控制端點安裝包中斷。
2. 讀取 HSUSBD_SETUP1_0、HSUSBD_SETUP3_2、HSUSBD_SETUP5_4、HSUSBD_SETUP7_6 寄存器，獲得請求和相關參數。
   (1) 從 HSUSBD_SETUP1_0 寄存器低字節獲得 bmRequestType。
   (2) 從HSUSBD_SETUP1_0寄存器高字節獲得bRequest。
   (3) 從HSUSBD_SETUP3_2寄存器獲得wValue。
   (4) 從HSUSBD_SETUP5_4寄存器獲得wIndex。
   (5) 從 HSUSBD_SETUP7_6 寄存器獲得 wLength。
3. 從 wValue 獲得地址。
4. 清除 NAK（設置 HSUSBD_CEPCTL 寄存器 NAKCLR 位）。
5. HSUSBD_CEPINTSTS 寄存器 STSDONEIF 位填 1 清除狀態完成中斷，HSUSBD_CEPINTEN 寄存器 STSDONEIEN 位填 1 始能中斷。
6. 等待狀態完成中斷產生。
   (1) 設置HSUSBD_CEPINTEN寄存器SETUPPKIEN位，始能安裝包中斷。
   (2) 填寫地址到 HSUSBD_FADDR 寄存器。
   (3) HSUSBD_CEPINTSTS 寄存器 STSDONEIF 位填 1 清除狀態完成中斷。

20.4.5 Get Descriptor

USBD控制器處理"Get Descriptor"請求：

1. 發生控制端點安裝包中斷。
2. 讀取 HSUSBD_SETUP1_0、HSUSBD_SETUP3_2、HSUSBD_SETUP5_4、HSUSBD_SETUP7_6 寄存器，獲得請求和相關參數。
   (1) 從HSUSBD_SETUP1_0寄存器低字節獲得bmRequestType。
   (2) 從HSUSBD_SETUP1_0寄存器高字節獲得bRequest。
   (3) 從HSUSBD_SETUP3_2寄存器獲得wValue。
   (4) 從HSUSBD_SETUP5_4寄存器獲得wIndex。
   (5) 從 HSUSBD_SETUP7_6 寄存器獲得 wLength。
3. 從wValue獲得描述符類型。
4. 檢查比較wLength和準備的描述符長度。

5. HSUSBD_CEPINTSTS 寄存器 STSDONEIF 和 INTKIF 位填 1 清除中斷，HSUSBD_CEPINTEN寄存器STSDONEIEN和INTKIEN位填1始能中斷。
6. 等待 IN-token 中斷。
   (1) HSUSBD_CEPINTSTS 寄存器 STSDONEIF 和 TXPKIF 位填 1 清除中斷，HSUSBD_CEPINTEN寄存器STSDONEIEN和TXPKIEN位填1始能中斷。
   (2) 描述符的數據填入HSUSBD_CEPDAT寄存器。
   (3) 描述符的長度填入 HSUSBD_CEPTXCNT 寄存器，輸出數據。
7. HSUSBD_CEPINTSTS寄存器INTKIF位填1清除中斷。
8. 等待 TX 中斷。
   (1) HSUSBD_CEPINTSTS寄存器STSDONEIF和TXPKIF位填1清除中斷。
   (2) 清除NAK（設置HSUSBD_CEPCTL寄存器NAKCLR位）。
   (3) HSUSBD_CEPINTSTS寄存器STSDONEIF位填1清除狀態完成中斷。
   (4) HSUSBD_CEPINTEN 寄存器 STSDONEIEN、SETUPPKIEN 位填 1 始能中斷。
9. 等待狀態完成中斷產生。
   (1) 設置HSUSBD_CEPINTEN寄存器SETUPPKIEN位，始能安裝包中斷。
   (2) HSUSBD_CEPINTSTS 寄存器 STSDONEIF 位填 1 清除狀態完成中斷。

20.4.6 IN 傳輸

USBD控制器透過DMA處理 IN 傳輸：

1. 設置HSUSBD_DMACTL寄存器DMARD位為1，填寫端點號到HSUSBD_DMACTL寄存器EPNUM位。
2. 檢查傳輸長度，如果大於DMA計數就要分次傳輸。
3. 設置HSUSBD_EPxINTEN寄存器TXPKIEN位，始能數據包傳輸中斷。
4. 檢查FIFO是否清空（HSUSBD_EPxINTSTS寄存器BUFEMPTYIF位為1）。
5. 設置HSUSBD_BUSINTEN寄存器RSTIEN、SUSPENDIEN、DMADONEIEN位，始能USB復位、暫停和DMA完成中斷。
6. 填寫實體位置到HSUSBD_DMAADDR寄存器。
7. 填寫傳輸計數到HSUSBD_DMACNT寄存器。
8. 設置HSUSBD_DMACTL寄存器DMAEN位，觸發DMA。
9. 等待 DMA 完成中斷發生。
   (1) 設置HSUSBD_BUSINTSTS寄存器DMADONEIF位，清除中斷。

(2) 檢查最後數據長度是否小於最大數據包，如果是，設置 HSUSBD_EPxRSPCTL 寄存器 SHORTTXEN 位，輸出最後數據。

20.4.7 OUT 傳輸

USBD控制器透過DMA處理 OUT 傳輸：

1. 設置HSUSBD_DMACTL寄存器DMARD位為0，填寫端點號到HSUSBD_DMACTL寄存器EPNUM位。
2. 檢查傳輸長度，如果大於DMA計數就要分次傳輸。
3. 設置HSUSBD_BUSINTEN寄存器RSTIEN、SUSPENDIEN、DMADONEIEN位，始能USB復位、暫停和DMA完成中斷。
4. 填寫實體位置到HSUSBD_DMAADDR寄存器。
5. 填寫傳輸計數到HSUSBD_DMACNT寄存器。
6. 設置HSUSBD_DMACTL寄存器DMAEN位，觸發DMA。
7. 等待 DMA 完成中斷發生。
   (1) 設置HSUSBD_BUSINTSTS寄存器DMADONEIF位，清除中斷。
   (2) 檢查數據長度是否為mass storage命令或資料長度。
   (3) 設置 HSUSBD_EPxINTEN 寄存器 RXPKIEN 位，始能接收數據中斷。
8. 等待接收數據中斷發生。清除 HSUSBD_EPxINTEN 寄存器 RXPKIEN 位，禁止接收數據。

20.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
HSUSBD Base Address: **HSUSBD_BA = 0xB001_6000**				
HSUSBD_GINTSTS	HSUSBD_BA+0x000	R	Global Interrupt Status Register	0x0000_0000
HSUSBD_GINTEN	HSUSBD_BA+0x008	R/W	Global Interrupt Enable Register	0x0000_0001
HSUSBD_BUSINTSTS	HSUSBD_BA+0x010	R/W	USB Bus Interrupt Status Register	0x0000_0000
HSUSBD_BUSINTEN	HSUSBD_BA+0x014	R/W	USB Bus Interrupt Enable Register	0x0000_0040
HSUSBD_OPER	HSUSBD_BA+0x018	R/W	USB Operational Register	0x0000_0002
HSUSBD_FRAMECNT	HSUSBD_BA+0x01C	R	USB Frame Count Register	0x0000_0000

HSUSBD_FADDR	HSUSBD_BA+0x020	R/W	USB Function Address Register	0x0000_0000
HSUSBD_TEST	HSUSBD_BA+0x024	R/W	USB Test Mode Register	0x0000_0000
HSUSBD_CEPDAT	HSUSBD_BA+0x028	R/W	Control Endpoint Data Buffer	0x0000_0000
HSUSBD_CEPCTL	HSUSBD_BA+0x02C	R/W	Control Endpoint Control Register	0x0000_0000
HSUSBD_CEPINTEN	HSUSBD_BA+0x030	R/W	Control Endpoint Interrupt Enable	0x0000_0000
HSUSBD_CEPINTSTS	HSUSBD_BA+0x034	R/W	Control Endpoint Interrupt Status	0x0000_1800
HSUSBD_CEPTXCNT	HSUSBD_BA+0x038	R/W	Control Endpoint In Transfer Data Count	0x0000_0000
HSUSBD_CEPRXCNT	HSUSBD_BA+0x03C	R	Control Endpoint Out Transfer Data Count	0x0000_0000
HSUSBD_CEPDATCNT	HSUSBD_BA+0x040	R	Control Endpoint Data Count	0x0000_0000
HSUSBD_SETUP1_0	HSUSBD_BA+0x044	R	Setup1 & Setup0 bytes	0x0000_0000
HSUSBD_SETUP3_2	HSUSBD_BA+0x048	R	Setup3 & Setup2 Bytes	0x0000_0000
HSUSBD_SETUP5_4	HSUSBD_BA+0x04C	R	Setup5 & Setup4 Bytes	0x0000_0000
HSUSBD_SETUP7_6	HSUSBD_BA+0x050	R	Setup7 & Setup6 Bytes	0x0000_0000
HSUSBD_CEPBUFSTART	HSUSBD_BA+0x054	R/W	Control Endpoint RAM Start Address Register	0x0000_0000
HSUSBD_CEPBUFEND	HSUSBD_BA+0x058	R/W	Control Endpoint RAM End Address Register	0x0000_0000
HSUSBD_DMACTL	HSUSBD_BA+0x05C	R/W	DMA Control Status Register	0x0000_0000
HSUSBD_DMACNT	HSUSBD_BA+0x060	R/W	DMA Count Register	0x0000_0000
HSUSBD_EPADAT	HSUSBD_BA+0x064	R/W	Endpoint A Data Register	0x0000_0000
HSUSBD_EPAINTSTS	HSUSBD_BA+0x068	R/W	Endpoint A Interrupt Status Register	0x0000_0003
HSUSBD_EPAINTEN	HSUSBD_BA+0x06C	R/W	Endpoint A Interrupt Enable Register	0x0000_0000
HSUSBD_EPADATCNT	HSUSBD_BA+0x070	R	Endpoint A Data Available Count Register	0x0000_0000
HSUSBD_EPARSPCTL	HSUSBD_BA+0x074	R/W	Endpoint A Response Control Register	0x0000_0000
HSUSBD_EPAMPS	HSUSBD_BA+0x078	R/W	Endpoint A Maximum Packet Size Register	0x0000_0000
HSUSBD_EPATXCNT	HSUSBD_BA+0x07C	R/W	Endpoint A Transfer Count Register	0x0000_0000
HSUSBD_EPACFG	HSUSBD_BA+0x080	R/W	Endpoint A Configuration Register	0x0000_0012
HSUSBD_EPABUFSTART	HSUSBD_BA+0x084	R/W	Endpoint A RAM Start Address Register	0x0000_0000
HSUSBD_EPABUFEND	HSUSBD_BA+0x088	R/W	Endpoint A RAM End Address Register	0x0000_0000

HSUSBD_EPBDAT	HSUSBD_BA+0x08C	R/W	Endpoint B Data Register	0x0000_0000
HSUSBD_EPBINTSTS	HSUSBD_BA+0x090	R/W	Endpoint B Interrupt Status Register	0x0000_0003
HSUSBD_EPBINTEN	HSUSBD_BA+0x094	R/W	Endpoint B Interrupt Enable Register	0x0000_0000
HSUSBD_EPBDATCNT	HSUSBD_BA+0x098	R	Endpoint B Data Available Count Register	0x0000_0000
HSUSBD_EPBRSPCTL	HSUSBD_BA+0x09C	R/W	Endpoint B Response Control Register	0x0000_0000
HSUSBD_EPBMPS	HSUSBD_BA+0x0A0	R/W	Endpoint B Maximum Packet Size Register	0x0000_0000
HSUSBD_EPBTXCNT	HSUSBD_BA+0x0A4	R/W	Endpoint B Transfer Count Register	0x0000_0000
HSUSBD_EPBCFG	HSUSBD_BA+0x0A8	R/W	Endpoint B Configuration Register	0x0000_0022
HSUSBD_EPBBUFSTART	HSUSBD_BA+0x0AC	R/W	Endpoint B RAM Start Address Register	0x0000_0000
HSUSBD_EPBBUFEND	HSUSBD_BA+0x0B0	R/W	Endpoint B RAM End Address Register	0x0000_0000
HSUSBD_EPCDAT	HSUSBD_BA+0x0B4	R/W	Endpoint C Data Register	0x0000_0000
HSUSBD_EPCINTSTS	HSUSBD_BA+0x0B8	R/W	Endpoint C Interrupt Status Register	0x0000_0003
HSUSBD_EPCINTEN	HSUSBD_BA+0x0BC	R/W	Endpoint C Interrupt Enable Register	0x0000_0000
HSUSBD_EPCDATCNT	HSUSBD_BA+0x0C0	R	Endpoint C Data Available Count Register	0x0000_0000
HSUSBD_EPCRSPCTL	HSUSBD_BA+0x0C4	R/W	Endpoint C Response Control Register	0x0000_0000
HSUSBD_EPCMPS	HSUSBD_BA+0x0C8	R/W	Endpoint C Maximum Packet Size Register	0x0000_0000
HSUSBD_EPCTXCNT	HSUSBD_BA+0x0CC	R/W	Endpoint C Transfer Count Register	0x0000_0000
HSUSBD_EPCCFG	HSUSBD_BA+0x0D0	R/W	Endpoint C Configuration Register	0x0000_0032
HSUSBD_EPCBUFSTART	HSUSBD_BA+0x0D4	R/W	Endpoint C RAM Start Address Register	0x0000_0000
HSUSBD_EPCBUFEND	HSUSBD_BA+0x0D8	R/W	Endpoint C RAM End Address Register	0x0000_0000
HSUSBD_EPDDAT	HSUSBD_BA+0x0DC	R/W	Endpoint D Data Register	0x0000_0000
HSUSBD_EPDINTSTS	HSUSBD_BA+0x0E0	R/W	Endpoint D Interrupt Status Register	0x0000_0003
HSUSBD_EPDINTEN	HSUSBD_BA+0x0E4	R/W	Endpoint D Interrupt Enable Register	0x0000_0000
HSUSBD_EPDDATCNT	HSUSBD_BA+0x0E8	R	Endpoint D Data Available Count Register	0x0000_0000
HSUSBD_EPDRSPCTL	HSUSBD_BA+0x0EC	R/W	Endpoint D Response Control Register	0x0000_0000
HSUSBD_EPDMPS	HSUSBD_BA+0x0F0	R/W	Endpoint D Maximum Packet Size Register	0x0000_0000

HSUSBD_EPDTXCNT	HSUSBD_BA+0x0F4	R/W	Endpoint D Transfer Count Register	0x0000_0000
HSUSBD_EPDCFG	HSUSBD_BA+0x0F8	R/W	Endpoint D Configuration Register	0x0000_0042
HSUSBD_EPDBUFSTART	HSUSBD_BA+0x0FC	R/W	Endpoint D RAM Start Address Register	0x0000_0000
HSUSBD_EPDBUFEND	HSUSBD_BA+0x100	R/W	Endpoint D RAM End Address Register	0x0000_0000
HSUSBD_EPEDAT	HSUSBD_BA+0x104	R/W	Endpoint E Data Register	0x0000_0000
HSUSBD_EPEINTSTS	HSUSBD_BA+0x108	R/W	Endpoint E Interrupt Status Register	0x0000_0003
HSUSBD_EPEINTEN	HSUSBD_BA+0x10C	R/W	Endpoint E Interrupt Enable Register	0x0000_0000
HSUSBD_EPEDATCNT	HSUSBD_BA+0x110	R	Endpoint E Data Available Count Register	0x0000_0000
HSUSBD_EPERSPCTL	HSUSBD_BA+0x114	R/W	Endpoint E Response Control Register	0x0000_0000
HSUSBD_EPEMPS	HSUSBD_BA+0x118	R/W	Endpoint E Maximum Packet Size Register	0x0000_0000
HSUSBD_EPETXCNT	HSUSBD_BA+0x11C	R/W	Endpoint E Transfer Count Register	0x0000_0000
HSUSBD_EPECFG	HSUSBD_BA+0x120	R/W	Endpoint E Configuration Register	0x0000_0052
HSUSBD_EPEBUFSTART	HSUSBD_BA+0x124	R/W	Endpoint E RAM Start Address Register	0x0000_0000
HSUSBD_EPEBUFEND	HSUSBD_BA+0x128	R/W	Endpoint E RAM End Address Register	0x0000_0000
HSUSBD_EPFDAT	HSUSBD_BA+0x12C	R/W	Endpoint F Data Register	0x0000_0000
HSUSBD_EPFINTSTS	HSUSBD_BA+0x130	R/W	Endpoint F Interrupt Status Register	0x0000_0003
HSUSBD_EPFINTEN	HSUSBD_BA+0x134	R/W	Endpoint F Interrupt Enable Register	0x0000_0000
HSUSBD_EPFDATCNT	HSUSBD_BA+0x138	R	Endpoint F Data Available Count Register	0x0000_0000
HSUSBD_EPFRSPCTL	HSUSBD_BA+0x13C	R/W	Endpoint F Response Control Register	0x0000_0000
HSUSBD_EPFMPS	HSUSBD_BA+0x140	R/W	Endpoint F Maximum Packet Size Register	0x0000_0000
HSUSBD_EPFTXCNT	HSUSBD_BA+0x144	R/W	Endpoint F Transfer Count Register	0x0000_0000
HSUSBD_EPFCFG	HSUSBD_BA+0x148	R/W	Endpoint F Configuration Register	0x0000_0062
HSUSBD_EPFBUFSTART	HSUSBD_BA+0x14C	R/W	Endpoint F RAM Start Address Register	0x0000_0000
HSUSBD_EPFBUFEND	HSUSBD_BA+0x150	R/W	Endpoint F RAM End Address Register	0x0000_0000
HSUSBD_EPGDAT	HSUSBD_BA+0x154	R/W	Endpoint G Data Register	0x0000_0000
HSUSBD_EPGINTSTS	HSUSBD_BA+0x158	R/W	Endpoint G Interrupt Status Register	0x0000_0003
HSUSBD_EPGINTEN	HSUSBD_BA+0x15C	R/W	Endpoint G Interrupt Enable Register	0x0000_0000

HSUSBD_EPGDATCNT	HSUSBD_BA+0x160	R	Endpoint G Data Available Count Register	0x0000_0000
HSUSBD_EPGRSPCTL	HSUSBD_BA+0x164	R/W	Endpoint G Response Control Register	0x0000_0000
HSUSBD_EPGMPS	HSUSBD_BA+0x168	R/W	Endpoint G Maximum Packet Size Register	0x0000_0000
HSUSBD_EPGTXCNT	HSUSBD_BA+0x16C	R/W	Endpoint G Transfer Count Register	0x0000_0000
HSUSBD_EPGCFG	HSUSBD_BA+0x170	R/W	Endpoint G Configuration Register	0x0000_0072
HSUSBD_EPGBUFSTART	HSUSBD_BA+0x174	R/W	Endpoint G RAM Start Address Register	0x0000_0000
HSUSBD_EPGBUFEND	HSUSBD_BA+0x178	R/W	Endpoint G RAM End Address Register	0x0000_0000
HSUSBD_EPHDAT	HSUSBD_BA+0x17C	R/W	Endpoint H Data Register	0x0000_0000
HSUSBD_EPHINTSTS	HSUSBD_BA+0x180	R/W	Endpoint H Interrupt Status Register	0x0000_0003
HSUSBD_EPHINTEN	HSUSBD_BA+0x184	R/W	Endpoint H Interrupt Enable Register	0x0000_0000
HSUSBD_EPHDATCNT	HSUSBD_BA+0x188	R	Endpoint H Data Available Count Register	0x0000_0000
HSUSBD_EPHRSPCTL	HSUSBD_BA+0x18C	R/W	Endpoint H Response Control Register	0x0000_0000
HSUSBD_EPHMPS	HSUSBD_BA+0x190	R/W	Endpoint H Maximum Packet Size Register	0x0000_0000
HSUSBD_EPHTXCNT	HSUSBD_BA+0x194	R/W	Endpoint H Transfer Count Register	0x0000_0000
HSUSBD_EPHCFG	HSUSBD_BA+0x198	R/W	Endpoint H Configuration Register	0x0000_0082
HSUSBD_EPHBUFSTART	HSUSBD_BA+0x19C	R/W	Endpoint H RAM Start Address Register	0x0000_0000
HSUSBD_EPHBUFEND	HSUSBD_BA+0x1A0	R/W	Endpoint H RAM End Address Register	0x0000_0000
HSUSBD_EPIDAT	HSUSBD_BA+0x1A4	R/W	Endpoint I Data Register	0x0000_0000
HSUSBD_EPIINTSTS	HSUSBD_BA+0x1A8	R/W	Endpoint I Interrupt Status Register	0x0000_0003
HSUSBD_EPIINTEN	HSUSBD_BA+0x1AC	R/W	Endpoint I Interrupt Enable Register	0x0000_0000
HSUSBD_EPIDATCNT	HSUSBD_BA+0x1B0	R	Endpoint I Data Available Count Register	0x0000_0000
HSUSBD_EPIRSPCTL	HSUSBD_BA+0x1B4	R/W	Endpoint I Response Control Register	0x0000_0000
HSUSBD_EPIMPS	HSUSBD_BA+0x1B8	R/W	Endpoint I Maximum Packet Size Register	0x0000_0000
HSUSBD_EPITXCNT	HSUSBD_BA+0x1BC	R/W	Endpoint I Transfer Count Register	0x0000_0000
HSUSBD_EPICFG	HSUSBD_BA+0x1C0	R/W	Endpoint I Configuration Register	0x0000_0092
HSUSBD_EPIBUFSTART	HSUSBD_BA+0x1C4	R/W	Endpoint I RAM Start Address Register	0x0000_0000

HSUSBD_EPIBUFEND	HSUSBD_BA+0x1C8	R/W	Endpoint I RAM End Address Register	0x0000_0000
HSUSBD_EPJDAT	HSUSBD_BA+0x1CC	R/W	Endpoint J Data Register	0x0000_0000
HSUSBD_EPJINTSTS	HSUSBD_BA+0x1D0	R/W	Endpoint J Interrupt Status Register	0x0000_0003
HSUSBD_EPJINTEN	HSUSBD_BA+0x1D4	R/W	Endpoint J Interrupt Enable Register	0x0000_0000
HSUSBD_EPJDATCNT	HSUSBD_BA+0x1D8	R	Endpoint J Data Available Count Register	0x0000_0000
HSUSBD_EPJRSPCTL	HSUSBD_BA+0x1DC	R/W	Endpoint J Response Control Register	0x0000_0000
HSUSBD_EPJMPS	HSUSBD_BA+0x1E0	R/W	Endpoint J Maximum Packet Size Register	0x0000_0000
HSUSBD_EPJTXCNT	HSUSBD_BA+0x1E4	R/W	Endpoint J Transfer Count Register	0x0000_0000
HSUSBD_EPJCFG	HSUSBD_BA+0x1E8	R/W	Endpoint J Configuration Register	0x0000_00A2
HSUSBD_EPJBUFSTART	HSUSBD_BA+0x1EC	R/W	Endpoint J RAM Start Address Register	0x0000_0000
HSUSBD_EPJBUFEND	HSUSBD_BA+0x1F0	R/W	Endpoint J RAM End Address Register	0x0000_0000
HSUSBD_EPKDAT	HSUSBD_BA+0x1F4	R/W	Endpoint K Data Register	0x0000_0000
HSUSBD_EPKINTSTS	HSUSBD_BA+0x1F8	R/W	Endpoint K Interrupt Status Register	0x0000_0003
HSUSBD_EPKINTEN	HSUSBD_BA+0x1FC	R/W	Endpoint K Interrupt Enable Register	0x0000_0000
HSUSBD_EPKDATCNT	HSUSBD_BA+0x200	R	Endpoint K Data Available Count Register	0x0000_0000
HSUSBD_EPKRSPCTL	HSUSBD_BA+0x204	R/W	Endpoint K Response Control Register	0x0000_0000
HSUSBD_EPKMPS	HSUSBD_BA+0x208	R/W	Endpoint K Maximum Packet Size Register	0x0000_0000
HSUSBD_EPKTXCNT	HSUSBD_BA+0x20C	R/W	Endpoint K Transfer Count Register	0x0000_0000
HSUSBD_EPKCFG	HSUSBD_BA+0x210	R/W	Endpoint K Configuration Register	0x0000_00B2
HSUSBD_EPKBUFSTART	HSUSBD_BA+0x214	R/W	Endpoint K RAM Start Address Register	0x0000_0000
HSUSBD_EPKBUFEND	HSUSBD_BA+0x218	R/W	Endpoint K RAM End Address Register	0x0000_0000
HSUSBD_EPLDAT	HSUSBD_BA+0x21C	R/W	Endpoint L Data Register	0x0000_0000
HSUSBD_EPLINTSTS	HSUSBD_BA+0x220	R/W	Endpoint L Interrupt Status Register	0x0000_0003
HSUSBD_EPLINTEN	HSUSBD_BA+0x224	R/W	Endpoint L Interrupt Enable Register	0x0000_0000
HSUSBD_EPLDATCNT	HSUSBD_BA+0x228	R	Endpoint L Data Available Count Register	0x0000_0000
HSUSBD_EPLRSPCTL	HSUSBD_BA+0x22C	R/W	Endpoint L Response Control Register	0x0000_0000
HSUSBD_EPLMPS	HSUSBD_BA+0x230	R/W	Endpoint L Maximum Packet Size	0x0000_0000

			Register	
HSUSBD_EPLTXCNT	HSUSBD_BA+0x234	R/W	Endpoint L Transfer Count Register	0x0000_0000
HSUSBD_EPLCFG	HSUSBD_BA+0x238	R/W	Endpoint L Configuration Register	0x0000_00C2
HSUSBD_EPLBUFSTART	HSUSBD_BA+0x23C	R/W	Endpoint L RAM Start Address Register	0x0000_0000
HSUSBD_EPLBUFEND	HSUSBD_BA+0x240	R/W	Endpoint L RAM End Address Register	0x0000_0000
HSUSBD_DMAADDR	HSUSBD_BA+0x700	R/W	AHB DMA Address Register	0x0000_0000
HSUSBD_PHYCTL	HSUSBD_BA+0x704	R/W	USB PHY Control Register	0x0000_0420

21 USB 主機控制器

21.1 概述

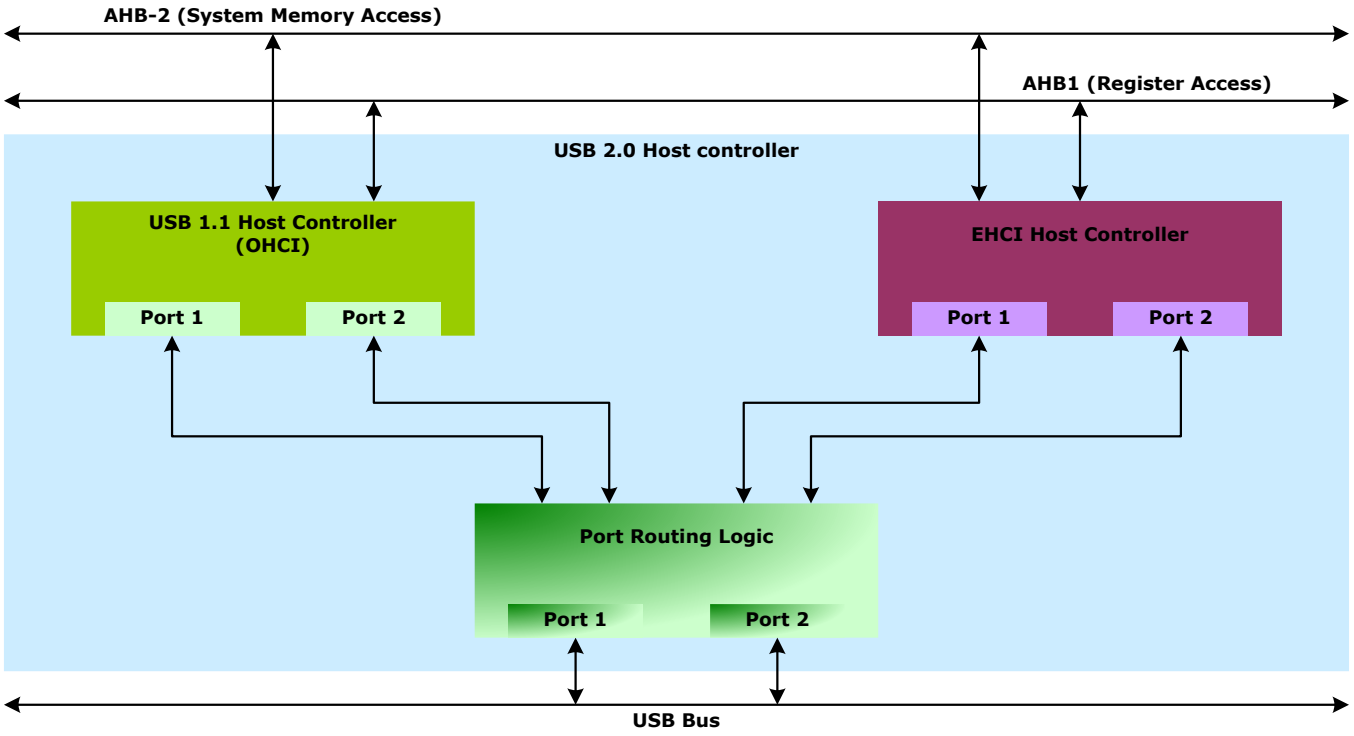
通用串行總線（USB）是一種快速，雙向，同步的，低成本的，動態附接的串行接口標準，用於調製解調器，掃描儀，個人數字助理，鍵盤，鼠標，以及數字成像設備等應用。USB 使用四線串行總線，在主機控制器和外圍設備連接網之間進行串行數據交換。連接的外圍設備通過主機調度，基於 token 封包協議操作，彼此共享 USB 帶寬。外圍設備均可動態地附接，設置，使用或拔除（支持熱插拔），同時間主機控制器與其他保持連接外圍設備間不受影響，仍可繼續正常操作。

USB 標準的主要設計目標是實現靈活的，隨插即放 (plug-and-play) 的 USB 設備連接網。在任何 USB 連接網中，只會有一個主機控制器，但可以同時連接最多達 127 個 USB 設備和集線器。

21.2 特性

- 完全符合 USB2.0 版規範。
- 兼容於 EHCI 1.0（增強主機控制器接口）。
- 兼容於 OHCI 1.0（開放主機控制器接口）修訂版。
- 支持高速（480Mbps 的），全速（12Mbps 的）和低速（1.5Mbps 的）USB 設備。
- 支持控制（Control），批量（Bulk），中斷（Interrupt），同步（Isochronous）和分割轉讓（Split Transfer）傳輸。
- 集成一個端口路由邏輯電路，支持將全速及低速設備路由至 OHCI 控制器。
- 可同支援 6 個 USB Lite 全速埠。
- 內置的 DMA 實時數據傳輸。

21.3 方塊圖



21.3.1 基本配置

將 USBH(CLK_HCLKEN[18]) 位設置為 1，以開啟 USB Host 時鐘源。

USB 1.1 主機的 48 MHz時鐘源，是來自於 USB 2.0 USB PHY 的 480MHz 時鐘除以10 而來的。 驅動程式規劃 USB_S(CLK_DIVCTL2[4:3]) 以選擇 USB 1.1主機的 48 MHz時鐘源是要從 USB 2.0 PHY 0 或 PHY1 而來。

用戶設置 SUSPEND（USBPCR0[8]）位，以使能 USB PHY 0，並且設置 SUSPEND（USBPCR1[8]）位，以使能 USB PHY 1。然後，用戶在開始使用 USB 主控制器之前，必須先檢查 CLKVALID（USBPCR0[11]）是否為高位。

21.3.2 USB 主機埠 0

USB 埠 0 是一個雙重角色埠，它可以成為 USB 裝置埠，或是成為 USB 主機埠 0。成為USB 主機埠或是裝置埠，是取決於 USB ID 腳位。如果 USB ID 為低位，那麼 USB 埠 0 就會成為裝置埠，如果 USB ID 為高位，那麼 USB 埠 0 就會成為主機埠。通過 USBID(SYS_PWRON[16]) 可以得知當前 USB 埠 0 是扮演主機埠或裝置埠的角色。

驅動程式是可以強制設置 USB 埠 0 角色的。首先，將 USRHDSEN(SYS_MISCFCR[11]) 位寫為1，如此就可以用 USBID(SYS_PWRON[16]) 來強制設定 USB 埠 0 工作為 USB 主機埠獲裝

置埠。

21.3.3 EHCI 控制器

EHCI 控制器通過 AHB 接口與系統連接。每當 CPU 想要發起寄存器讀取或寫入寄存器，便會透過 AHB 從機 I/F 信號來執行操作（寄存器讀取寫入）。CPU 充當總線主控器，啟動了這一轉移。在這個時候，EHCI 作為一個傳輸目標，並響應由系統軟件所發起的傳輸。例如，如果 CPU 要寫入的 EHCI 的某一個存儲器映射寄存器，它只要直接給出寄存器地址和值，即可寫入到指定的寄存器。EHCI 使用該地址來標定寄存器位置，並用軟件指定的值來填寫寄存器。如果是寄存器讀取操作，EHCI 從寄存器位址取得值，並把它的發到系統總線。

當 EHCI 想要執行數據傳送，它便會充當總線主控器並啟動數據傳輸。在這個時候，系統存儲器便充當為總線目標。 EHCI，作為總線主控器可以執行兩種類型的數據傳輸，從 EHCI 到系統存儲器，以及從系統存儲器向 EHCI。當 EHCI 希望數據從下游 USB 2.0 設備向系統存儲器傳輸時，它通過訪問系統存儲器的接口信號，發起對存儲器的寫入操作。EHCI 將控制字組，數據和數據計數發給系統存儲器，存儲器控制器接收數據並將其移動到內存。如果數據必須從存儲器移動到下游設備，EHCI 便會對系統總線發出讀傳輸，內存控制器通過存儲器接口的信號給出數據， EHCI 接受數據，並將它們移動到下游設備。

21.3.4 OHCI 控制器

21.3.4.1 AHB 介面

OpenHCI 主機控制器通過 AHB 總線連接到系統。設計上需要用到主機和從機總線操作。作為主機，主機控制器負責 AHB 總線上運行週期，以訪問 EDs 和 TDs 以及內存和本地數據緩存之間的數據傳輸。作為從機，主機控制器監控 AHB 總線上的週期，並確定何時對這些週期作出回應。對主機控制器操作寄存器的設定和非實時控制，都可以通過 AHB 總線從屬接口來完成。

21.3.4.2 AHB 主機

在獲得控制同意之後（granted），主機便會將地址和數據發到總線。

21.3.4.3 AHB 從機

對主機控制器的配置是通過從機接口。

21.3.4.4 列表處理器（List Processor）

列表處理器管理來自主機控制器驅動程序的數據結構，並協調主機控制器內的所有活動。

21.3.4.5 幀管理（Frame Management）

幀管理單元負責管理由 USB 規範和 OpenHCI 規範所需的幀的特定任務。這些任務包括：

- 對於 OpenHCI 幀相關寄存器的管理。
- 最大數據包計數器的操作。
- 當有 USB 請求對 SIE 發出時，執行幀可行性確認。
- 定時每幀對 SIE 發出 SOF 令牌請求。

21.3.4.6 中斷處理

中斷是主機控制器驅動程序與主機控制器發起的傳輸之間的通信方法。有幾個事件可能觸發來自主控制器的中斷。每一個具體的事件會設置在 HcInterruptStatus 寄存器中對應的特定位。

21.3.4.7 主機控制器總線主機

主機控制器總線主機位於數據路徑的中央塊。主機控制器總線主機協調所有對 AHB 接口的訪問。在主機控制器中的總線主機操作有兩個來源：列表處理器和數據緩衝引擎。

21.3.4.8 數據緩衝區

數據緩衝區作為總線主機與SIE之間的數據接口。它是一個 64 字節的雙向異步鎖存型 FIFO 和一個雙字的 AHB 保持寄存器的組合。

21.3.4.9 USB 接口

USB 接口包括集成的根集線器與兩個外部端口，端口 1 和 2，以及串行接口引擎（SIE）和 USB 時鐘產生器。USB 接口負責執行來自主機控制器所發出的總線交易請求，並實現 USB 規範的集線器與端口管理。

21.3.4.10 串行接口引擎（SIE）

SIE 負責管理所有的 USB 交易。它控制總線協議，封包生成與提取，數據並行到串行轉換，CRC 編碼，比特填充和 NRZI 編碼。在 USB 總線上面的所有交易，都是從列表處理器及幀管理器發出請求的。

21.3.4.11 根集線器（Root Hub）

根集線器包含了數個可單獨控制的端口，和一個維護所有的端口控制/狀態的集線器。

21.3.4.12 USB Lite 埠

NUC980 支援了 6 個 USB Lite 全速埠。每個 USB Lite port 都有多組 GPIO 腳位可選為 USB D+/D- 腳。USB Lite 0 ~ 5 分別依序對應到 OHCI (USB 1.1) 主機根集線器埠 2 ~ 7。

21.4 功能描述

21.4.1 初始設置

要令 USB 主機運作，必須正確執行下列的操作:

1. 使能 USB 主機控制器的時鐘源。將 USBH（CLK_HCLKEN[18]）位設置為 1。
2. 始能 USB PHY，USB 主機才能進行 USB 總線傳輸。分別對 USBPCR0 及 USBPCR1 寫入 0x160 及 0x520。
3. 設置多功能腳位，將 PE.11 設置為 USB0_VBUSVLD，PE.12 設置為 USBH_PWREN。
4. 如果有使用 USB Lite port，將選擇做為 USBH DP 及 DM 的 GPIO pin 設置為 USBHLx_DP 及 USBHLx_DM (x = 0~5)。
5. 對 OHCI 主機控制器進行初始化。
6. 對 EHCI 主機控制器進行初始化。

21.4.2 根集線器端口路由

NUC980 系列 MCU 同時具備了 EHCI（USB2.0）和 OHCI（USB1.1）主機控制器。這兩個主機控制器共享了根集線器上的兩個 USB 端口。如果 EHCI 在啟動狀態下（UCFGR[0] 設置為 1），則 EHCI 將是 USB 端口的默認擁有者。如果沒有啟用 EHCI，則 OHCI 將是 USB 端口唯一擁有者。根集線器上的兩個 USB 端口可以各別被 EHCI 或 OHCI 主機控制器擁有。

EHCI 主機控制器專用於 USB 2.0 設備。如果一個非 USB 2.0 設備插入到 USB 端口，EHCI 主機控制器將依照正常 USB 設備列舉程序，重置並使能此設備，然後對此設備進行資料傳輸。由於 EHCI 擁有端口控制權， OHCI 是完全不知道有 USB 設備被接上來的。

然而，如果是一個 USB 1.1 的設備被插入到 USB 端口，那麼 EHCI 將會無法成功地重置此 USB 設備。在這種情況下，EHCI 驅動程序或集線器驅動程式，應該要將該端口的擁有權轉移給 OHCI。驅動程序將 PO（UPSCRx[13]）位設置為 1，便可以將端口擁有權轉移給 OHCI。接下來，OHCI 的根集線器端口將獲得一個連接設備的狀態。然後，OHCI 驅動程序便可以開始重置及使能該 USB 設備。

當 USB 設備在連接中，OHCI 驅動程式無法主動將端口擁有權轉移給 EHCI。在 USB 設備斷開之後，該端口的擁有權會自動歸還給 EHCI，PO（UPSCRx[13]）位會被主機控制器自動清除為 0。

21.4.3 OHCI

NUC980 OHCI 主機控制器，完全符合開放主機控制器接口（OHCI）1.0 版標準。其他平台上的 OHCI 驅動程式，可以輕易地移植到 NUC980 平台上執行。

21.4.3.1 數據結構

除了直接訪問主機控制器的寄存器，OHCI 驅動程序必須保持以下的內存塊與主機控制器進行通信：

- 端點描述符（Endpoint Descriptor）列表
- 傳輸描述符（Transfer Descriptor）列表
- 主機控制器通信區（HCCA）

上述這些數據結構都是由驅動程序建立於系統內存中。主機控制器將通過 DMA 傳輸方式去訪問這些存儲塊。所有端點描述符，傳輸描述符，HCCA 和傳輸緩衝器，都必須設置為不可緩存區域。

端點描述符和傳輸描述符必須與 32 字節地址邊界對齊。主機控制器通信區必須與 256 字節地址邊界對齊。

21.4.3.2 端點描述符

OHCI 主機控制器通過端點（Endpoints）劃分為符合 USB 傳輸四種類型的端點描述符（以下簡稱 ED）。HcCtrHED 寄存器指向了控制（Control）ED 串列，HcBlkHED 寄存器指向了批量（Bulk）ED 串列，HCCA 的 InterruptTable 則指向由中斷（Interrupt）ED 及同步（Isochronous）ED 共同組成的樹狀列表。驅動程序必須為 USB 設備的每個端點，創建和操作一個對應的 ED。

所有的傳輸類型均具有相同的端點描述符格式。通用格式如下所列：

端點描述符

	31	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dword 0	—		MPS										F	K	S	D		EN			FA							
Dword 1	TD 隊列尾指針 (TailP)																								—			
Dword 2	TD 隊列頭指針 (HeadP)																								0		C	H
Dword 3	下一端點描述符 (NextED)																								—			

控制（Control）ED 列表是由主機控制器驅動程序（HCD）創建，所有新建的控制 ED 均應被添加到控制 ED 列表的末尾。 HCD 必須將控制 ED 列表中的第一 ED 的實體位址寫入到 HcCtrHED 寄存器。如此，主機控制器便可以找到控制 ED 列表，並沿著列表依序處理所有控制 ED。

同樣地，所有的批量（Bulk）ED 均應被放置到批量 ED 列表，HCD 將批量 ED 列表的第一個 ED 的實體位址寫入到 HcBlkHED 寄存器。主機控制器便可以找到批量 ED 列表，並沿著列表依序處理所有批量 ED。

中斷（Interrupt）ED 列表與控制或批量 ED 列表不同，並不是由主機控制器的操作寄存器指出位址。中斷 ED 列表是由 HCCA（主機控制器通信區域）的 InterruptTable 所指向，這是一個完全由驅動程序創建的存儲區。在 HCCA 的 InterruptTable 中，總計有 32 個指標，每個指標均指向一個中斷 ED 列表節點串。中斷 ED 表的結構將在後面的 HCCA 章節中說明。

所有 32 個中斷 ED 列表節點串的末尾，最終均會指向同一個中斷 ED 列表節點，此中斷 ED 列表節點為 1ms 輪詢間隔的中斷 ED 列表，這也是每一個中斷 ED 列表節點串的最後一個節點。在一些實際狀況下，可能不存在任何 1ms 輪詢間隔的中斷 ED，如果是這種情況的話，那麼在 1ms 輪詢間隔的中斷 ED 列表節點上，仍然應該放置佔位符，對於 2ms，4ms，8ms，16ms，及 32ms 的列表節點亦是如此。事實上，整個中斷 ED 表就是由這些不同的輪詢間隔中斷 ED 列表節點所組成的。

同步（Isochronous）ED 列表必須被鏈接到 1ms 輪詢間隔中斷 ED 列表的末尾。主機控制器驅動程序必須負責維護中斷 ED 列表和同步 ED 列表，包括 HCCA 和 InterruptTable 的維護。HCCA 的實體位址是由 HcHCCA 寄存器指出，主機控制器透過此寄存器找到 HCCA。當然，驅動程序必須負責創建 HCCA 並將 HCCA 的實體地址寫入到到 HcHCCA 寄存器。

21.4.3.3 傳輸描述符

ED 用於描述 USB 端點的特性，僅憑 ED 本身並不足以使主機控制器對 USB 總線進行任何數據傳輸。OHCI 採用傳輸描述符（TD），用來描述一個 USB 數據傳輸的細節，包括了將被用於傳輸的數據緩存的實體位址，目標 USB 設備的裝置地址，目標端點的端點地址，傳輸的類型，以及傳輸的方向等。

TD 是由驅動程序建立及維護的，驅動程序按照將要進行之傳輸的細節，填寫到 TD 中，然後將之附加到傳輸對應 ED 底下的 TD 列表中。當主機控制器處理此 ED 時，將會看到新加入的 TD，並按照 TD 描述內容進行真正的 USB 數據傳輸。

當 TD 所描述的 USB 數據傳輸被主機控制器完成後，不管傳輸過程是否有錯誤發生，該 TD 便會被主機控制器從 TD 隊列中移出，並放置到完成隊列（Done Queue）中。然後主機控制器會發出中斷請求，驅動程序經由 HCCA 的 HccaDoneH 指標，可取得完成隊列的起始實體位址。然後，驅動程序逐一檢視完成隊列中的 TD，確認傳輸的結果並回收 TD。

OHCI 規範了兩款 TD 類型，一般 TD 和 TD 同步。TD 格式如下：

通用傳輸描述符

	31–28	27–26	25–24	23–21	20–19	18	17–0
Dword 0	CC	EC	T	DI	DP	R	—
Dword 1	當前緩存指標 (CBP)						
Dword 2	下一 TD 指標 (NextTD)						0
Dword 3	緩存結尾 (BE)						

Isochronous Transfer Descriptor

	31–28	27	26–24	23–21	20–16	15–12	11–5	4–0
Dword 0	CC		FC	DI	—	SF		
Dword 1	緩存頁 0 (BP0)					—		
Dword 2	下一 TD							0
Dword 3	緩存結尾 (BE)							
Dword 4	Offset1/PSW1					Offset0/PSW0		
Dword 5	Offset3/PSW3					Offset2/PSW2		
Dword 6	Offset5/PSW5					Offset4/PSW4		
Dword 7	Offset7/PSW7					Offset6/PSW6		

21.4.3.4 主機控制器通信區

主機控制器通信區（HCCA）是一個 256 字節的數據結構，由驅動程序從系統內存配置建立，用作驅動程序與主機控制器之間的通信區。HCCA 必須對齊到 256 字節地址邊界，這個存儲塊必須被設置為不可緩存區域。

偏移量	大小 (字節)	名稱	說明
0	128	HccaInterrruptTable	這 32 個字組指向中斷 ED 列表節點。
0x80	2	HccaFrameNumber	包含當前幀號。主機控制器在開始處理當前幀的週期列表前，會更新此值。
0x82	2	HccaPad1	當主機控制器更新 HccaFrameNumber 時，它會將此值清除為 0。。
0x84	4	HccaDoneHead	在到達一 USB 幀的結束，主機控制器便會將 HcDoneH 寄存器的當前值寫入此項目，並產生一個中斷。 在驅動程式清除 WD（HcIntSts[1]）位之前，主機控制器不會再覆寫此項目。 當此項目的最低位被設置為 1 時，表示此項目被寫入的當時，有一個非屏蔽的 HcIntSts 被設置了。
0x88	116	reserved	保留給主機控制器使用

21.4.3.5 OHCI 初始化

主機控制器的初始化可包含以下步驟：

1. 通過設置 MIE（HcIntDis[31]），禁止主機控制器的所有中斷。
2. 執行軟件軟件復位命令：設置 HCR（HcComSts[0]）等待 10 毫秒，HCR 位會被 OHCI 清除為 0。如果沒有，則復位失敗。
3. 配置並創建一切必要的列表結構和存儲塊，包括 HCCA，並初始化所有由驅動程序維護的列表，包括 HCCA 的 InterruptTable（HCCA 必須對齊 256 字節地址邊界，而 ED 和 TD 必須對齊 32 字節地址邊界）。
4. 清除 HcCtrHED 和 HcBlkHED 寄存器。
5. 將 HCCA 的實體地址寫入到 HcHCCA 寄存器。
6. 將幀間隔值（11999 ± 6）寫入 HcFmIntv 寄存器，並將此值的 90% 寫入 HcPerSt 寄存器。
7. 將 0x628 寫入到 HcLSTH 寄存器（0x628 是 HcLSTH 寄存器的復位默認值）。
8. 設置 BLE（HcControl[5]），CLE（HcControl[4]），IE（HcControl[3]），PLE（HcControl[2]），以激活 Bulk，Control，Interrupt，及 Isochronous 傳輸。
9. 對 HCFS（HcControl[7:6]）寫入 10b，令主機控制器轉移至工作狀態。
10. 設置 HcIntEn 寄存器，啟用所需中斷，並對 HcIntSts 寄存器相應位寫 1 ，以清除中斷狀態。
11. 設置 LPSC（HcRhSts[16]），以打開根集線器端口電源。（NUC980 系列 MCU 的 OHCI 根集線器是使用全局電源開關模式）
12. 使能 AIC 的 OHCI 中斷。
13. 連接集線器的設備驅動程序。

21.4.4 EHCI

NUC980 EHCI 主機控制器，完全符合增強主機控制器接口（EHCI）1.0 版標準。其他平台上的 EHCI 驅動程式，可以輕易地移植到 NUC980 平台上執行。

21.4.4.1 數據結構

除了直接訪問主機控制器的寄存器，EHCI 驅動程序必須保持以下的內存塊與主機控制器進行通信：

- 同步傳輸描述符（iTD）
- 拆分交易同步傳輸描述符（siTD）
- 隊列項傳輸描述符（qTD）
- 隊列頭描述符（QH）
- 週期性跨幀節點（FSTN）

21.4.4.2 同步傳輸描述符

同步傳輸描述符只用於高速同步端點。所有其他傳輸類型應使用隊列項傳輸描述符。iTD 必須對齊 32 字節邊界。需要注意的是 iTD 必須要位於非緩存內存。以下為 iTD 的結構內容：

0x00	Next Link Pointer						0	Typ	T
0x04	Status	Transaction 0 Length		ioc	PG	Transaction 0 Offset			
0x08	Status	Transaction 1 Length		ioc	PG	Transaction 1 Offset			
0x0C	Status	Transaction 2 Length		ioc	PG	Transaction 2 Offset			
0x10	Status	Transaction 3 Length		ioc	PG	Transaction 3 Offset			
0x14	Status	Transaction 4 Length		ioc	PG	Transaction 4 Offset			
0x18	Status	Transaction 5 Length		ioc	PG	Transaction 5 Offset			
0x1C	Status	Transaction 6 Length		ioc	PG	Transaction 6 Offset			
0x20	Status	Transaction 7 Length		ioc	PG	Transaction 7 Offset			
0x24	Buffer Pointer (Page 0)					EndPt	R	Device Address	
0x28	Buffer Pointer (Page 1)					I/O	Maximum Packet Size		
0x2C	Buffer Pointer (Page 2)					Reserved			Mult

0x30	Buffer Pointer (Page 3)	Reserved
0x34	Buffer Pointer (Page 4)	Reserved
0x38	Buffer Pointer (Page 5)	Reserved
0x3C	Buffer Pointer (Page 6)	Reserved

21.4.4.3 拆分交易同步傳輸描述符

siTD 是 EHCI 用來實現全速交易同步傳輸所使用的。

當一個全速的 USB 1.1 設備連接在一個 高速的 USB 2.0 集線器底下，EHCI 可以透過拆分交易傳輸協議與該 USB 1.1 設備進行傳輸。在 EHCI 與 USB 2.0 集線器之間進行的是拆分交易傳輸，集線器會將此傳輸內容轉譯為 USB 1.1 傳輸協議並與 USB 1.1 設備進行傳輸。

需要注意的是 siTD 必須位於非緩存存儲塊內，而且 siTD 必須對齊 32 字節邊界。以下為 siTD 的結構內容：

0x00	Next Link Pointer													0	Typ	T
0x04	I/O	Port Number				R	Hub Addr				R	EndPt		R	Device Address	
0x08	Reserved							uFrame C-mask						uFrame S-mask		
0x0C																
0x10	Buffer Pointer (Page 0)											Current Offset				
0x14	Buffer Pointer (Page 1)											Reserved			TP	T-count
0x18	Back Pointer														0	T

21.4.4.4 隊列項傳輸描述符

隊列項傳輸描述符需搭配隊列頭描述符，它可實現一個或多個 USB 交易，一個隊列項傳輸描述符可表示高達 20480（5 *4096）個字節的數據傳輸。該結構包含用兩個用於隊列推進的指針，表示傳輸狀態的雙字，和五個數據緩衝區指針。

此描述符傳輸相關的傳輸緩衝區，在虛擬地址上必須是連續的。緩衝區可以開始任何字節邊界。整個傳輸緩衝區可以分配在不連續的實體內存頁上，各 Buffer Pointer 緩衝區指針被用來指向傳輸緩衝區的每個實體內存頁地址。qTD 必須對齊 32 字節邊界。需要注意的是 qTD 必須要位於非緩存內存。以下為 qTD 的結構內容：

0x00	Next qTD Pointer					0	T
0x04	Alternate Next qTD Pointer					0	T
0x08	dt	Total Bytes To Transfer	ioc	C_Page	Cerr	PID Code	Status
0x0C	Buffer Pointer (Page 0)					Current Offset	
0x10	Buffer Pointer (Page 1)					Reserved	
0x14	Buffer Pointer (Page 2)					Reserved	
0x18	Buffer Pointer (Page 3)					Reserved	
0x1C	Buffer Pointer (Page 4)					Reserved	

21.4.4.5 EHCI 初始化

EHCI 主機控制器的初始化可包含以下步驟：

1. 設置 USBH（CLK_HCLKEN[18]）位，以使能 USB 主機時鐘。
2. 對 USBPCR0 寄存器寫入 0x160，以啟用 PHY0。對 USBPCR1 寄存器寫入 0x120，以啟用 PHY1。
3. 清除 RUN（UCMDR[0]）位，令 EHCI 進入停止狀態。
4. 重置 EHCI 主機控制器。設置 HCRST（UCMDR[1]）位，等待 10ms 之後，EHCI 主機控制器會清除 HCRST，表示 EHCI 重置完成。
5. 分配非緩存存儲塊以建立週期幀列表，它是一個 32 位的指針數組。將此週期幀列表的實體地址寫到 UPFLBAR 寄存器。
6. 寫入 UIENR 寄存器，使能中斷。
7. 分配一個隊列頭描述符（QH）作為非同步環狀隊列的頭，將此 QH 的實體地址寫到 UCALAR 寄存器。
8. 對 UCFGR 寄存器寫入 1。這將會使端口的路由邏輯默認路由到 EHCI 控制器。
9. 設置 UPSCR0 和 UPSCR1 寄存器的 PP 位，以開啟端口 0 和端口 1 端口的電源。一旦 USB 設備連接，端口狀態寄存器 UPSCR0/1 便可以反映端口連接狀態。

21.4.4.6 USB 命令

通過 USBCMD 寄存器可以向 EHCI 主控制器執行下達命令。

運行／停止

對 RUN（USBCMD[0]）位寫入 1 表示令 EHCI 運行，寫入 0 表示令 EHCI 停止運行。

重置 EHCI

將 HCRST（UCMDR[1]）位設置為 1，可以重置 EHCI 主機控制器。此重置對根集線器寄存器的作用類似於一個芯片硬件復位。當主機控制器完成重置，HCRST 便會被清除為零。

但是在 HCHalted（USTSR[12]）位為 0 的時候，軟件不可以將 HCRST 設置為 1。對運行中的主機控制器進行重置，都將會導致無法預期的結果。

幀列表大小

FLSZ（UCMDR[3:2]）決定幀列表的大小。內容值定義為：

00b	1024 個（4096字節）默認值
01b	512 個（2048字節）
10b	256 個（1024字節） 適用於資源有限的環境
11b	保留

啟用週期調度

設置 PSEN（UCMDR[4]）位可令 EHCI 主機控制器啟動週期調度，也就是開始服務中斷（Interrupt）及同步（Isochronous）傳輸。

啟用非同步調度

設置 ASEN（UCMDR[5]）位可令 EHCI 主機控制器啟動非同步調度，也就是開始服務控制（Control）及批量（Bulk）傳輸。

非同步前進門鈴中斷

設置 IAAD（UCMDR[6]）位可要求 EHCI 主機控制器，在下一次將要進入非同步調度之前發出一個中斷。當非同步調度被禁用時，軟件不應該設置此為位，這樣做將產生不確定的結果。

中斷控制閥

ITC（UCMDR[23:16] ）可用來控制 EHCI 主機控制器發出中斷的最大速率。有效的值定義如下：

值	最大中斷間隔

00b	保留
01h	1 微幀
02h	2 微幀
04h	4 微幀
08h	8 微幀 （默認值，相當於1 毫秒）
10h	16 微幀 （2 毫秒）
20h	32 微幀 （4 毫秒）
40h	64 微幀 （8 毫秒）

21.4.4.7 中斷處裡

USB 中斷

當USB 交易完成，並且傳輸描述符有設置 IOC 位，則主機控制器便會設置USBINT（USTSR[0]）中斷狀態。當檢測到短封包（short packet）時，EHCI 主機控制器也會設置此中斷狀態。

USB 錯誤中斷

當USB 交易完成並有錯誤發生，則主機控制器便會設置 UERRINT（USTSR[1]）中斷狀態。如果發生錯誤的傳輸描述符同時有設置 IOC 位，那麼主機控制器會同時設置 USBINT（USTSR[0]）中斷狀態。

端口變化檢測中斷

當 PO（UPSCRx[13]）由 0 變為 1，或是 FPR（UPSCRx[6]）由 0 變為 1，則主機控制器便會設置 PCD（USTSR[2]）中斷狀態。當 CSC（UPSCRx[1]）位有變化，主機控制器便也會設置 PCD 中斷狀態

幀列表翻轉中斷

當幀列表指數從最大值翻轉為零的時候，主機控制器便會設置 FLR（USTSR[3]）。發生幀列表翻轉的確切指數值，則取決於幀列表的大小。

主機系統錯誤中斷

當涉及主機控制器模塊的主機系統訪問發生了嚴重的錯誤時，主機控制器便會設置HSERR

（USTSR[4]）中斷狀態。當這個錯誤發生時，主機控制器將會清除 RUN（UCMDR[0]），以防止主機控制器繼續服務已排程的 TD 。

非同步推進中斷

系統軟件可以設置 IAAD（UCMDR[6]）位，以求 EHCI 主機控制器在下一次將要進入非同步調度之前發出一個 IAA（USTSR[5]）中斷。

21.4.4.8 根集線器

NUC980 EHCI 主機控制器包含了兩個端口寄存器，UPSCR0 和 UPSCR1，分別反映端口 0 及端口 1 的實際狀態：

端口當前連接狀態

CCS（UPSCRx[0]）位反映了根集線器端口的當前連接狀態，1 表示有當前有 USB 設備連接，0 表示當前沒有 USB 設備連接。

端口連接狀態變化

當 CCS 從 0 變化為 1，或者從 1 變化為 0，主機控制器便會設置 CSC（UPSCRx[1]）位，以表示連接狀態發生變化。

端口啟用／禁用

PE（UPSCRx[2]）位，1 = 端口啟用，0 = 端口禁用。端口不能由軟件直接啟用，只能經由主機控制器對端口執行復位程序後啟用。經由復位程序確定連接的設備是高速設備，主機控制器將便會將 PE 位設置為 1。

端口可以通過故障狀態（斷開事件或其他故障情況），或者通過主機軟件被禁用。需要注意的是，該位的狀態必須等到端口實際狀態改變之後才會變為 0。

當端口被禁用時，此端口往下游的 USB 傳播會被阻斷，除非重新執行復位啟用。

端口啟用／禁用變化

PEC（UPSCRx[3]）位，1 = 端口啟用／禁用狀態已經改變， 0 = 沒有變化。對於根集線器而言，只有當 EOF2 點（見 USB 規範第 11 章的端口錯誤定義）存在適當的條件致使端口被禁的情況下，此位才會被設置為 1。而軟件則可以寫入 1 來清除此位。

端口過電流激活

OCA（UPSCRx[4]）位，1 = 該端口目前存在過電流（over-current）狀況，0 = 該端口目前無過電流狀況。當過電流狀況解除時，此位將自動由 1 轉為 0。

過電流變化

當 OCC（UPSCRx[5]）位被設置為 1 時，表示有過電流狀態變化。軟件可通過寫 1 清除此位。

強制喚醒端口

FPR（UPSCRx[6]）位，1 = 端口偵測到喚醒或被驅動喚醒，0 = 無喚醒。操縱此位的功能定義取決於 SUSPEND（UPSCRx[7]）位當時的值。例如，如果端口並未掛起，而軟件對此位為寫入 1，那麼將無法確定會對 USB 總線造成甚麼樣的影響。

軟件設置 FPR 位為 1 可驅動喚醒信號。當端口處在掛起狀態下，如果主機控制器檢測到總線有 J-到-K 的轉換，便會設置此位為 1。如果 FPR 位轉為 1 是因為 J-到-K 的轉換，那麼 PCD（USTSR[2]）位也會被設置為 1。如果是軟件將 FPR 位設置為 1 的，那麼主機控制器就不可以設置 PCD 位。

端口掛起

SUSPEND（UPSCRx[7]）位，1 = 端口處於掛起狀態，0 = 端口不處於掛起狀態。

在掛起狀態下，該端口下游的數據傳播會被阻斷，除非端口復位。如果 SUSPEND 位被寫 1 的時候有交易正在進行，那麼在該交易結束後才會進行阻斷。在掛起狀態下，端口會持續檢測喚醒訊號。注意，直到端口真正被掛起後，SUSPEND 位的狀態才會改變，如果有交易目前正在進行，那麼端口的掛起可能會有延遲。

端口復位

PRST（UPSCRx[8]）位，1 = 端口處於復位狀態，0 = 端口不處於復位狀態。當軟件將 PRST 位從 0 寫為 1，如 USB 2.0 規範中定義的，總線將會被啟動復位程序。軟件寫 0 到 PRST 位，便會終止總線復位程序。軟件必須維持 PRST 為 1 足夠長的時間，以確保復位程序能夠如 USB 2.0 規範所規定的程序完成。注：當軟件對 PRST 寫 1，同時也必須對 PE（UPSCRx[2]）寫 0。

需要注意的是，當軟件對 PRST 位寫 0，可能有狀態變化到 0 的延遲。在復位程序完成之前，PRST 位都會讀到 0。如果端口在復位完成之後處於高速模式下，則主機控制器會自動啟用該端口（也就是將 PE（UPSCRx[2]）設置為1）。

端口供電

PP（UPSCRx[12]）位控制了端口供電的開啟／關閉。對 PP 寫 1 為打開端口供電，寫 0 則是關閉端口供電。

在供電端口上檢測到過電流情況下，每個受過電流影響的端口，PP 位都可能會被主機控制器從 1 變更為 0。

端口所有者

PO（UPSCRx[13]）位，0 = EHCI 主機控制器擁有端口，1 = OHCI 主機控制器擁有端口。

當 CF（UCFGR[0]）位為 1 時，PO 位會無條件為 0。當 CF（UCFGR[0]）位為 0 時，PO 位會無條件為 1

21.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
USBH Base Address: **USBH_BA = 0xB001_7000** **HSUSBH_BA = 0xB001_5000**				
HcRevision	USBH_BA+0x000	R	Host Controller Revision Register	0x0000_0110
HcControl	USBH_BA+0x004	R/W	Host Controller Control Register	0x0000_0000
HcCommandStatus	USBH_BA+0x008	R/W	Host Controller Command Status Register	0x0000_0000
HcInterruptStatus	USBH_BA+0x00C	R/W	Host Controller Interrupt Status Register	0x0000_0000
HcInterruptEnable	USBH_BA+0x010	R/W	Host Controller Interrupt Enable Register	0x0000_0000
HcInterruptDisable	USBH_BA+0x014	R/W	Host Controller Interrupt Disable Register	0x0000_0000
HcHCCA	USBH_BA+0x018	R/W	Host Controller Communication Area Register	0x0000_0000
HcPeriodCurrentED	USBH_BA+0x01C	R/W	Host Controller Period Current ED Register	0x0000_0000
HcControlHeadED	USBH_BA+0x020	R/W	Host Controller Control Head ED Register	0x0000_0000
HcControlCurrentED	USBH_BA+0x024	R/W	Host Controller Control Current ED Register	0x0000_0000
HcBulkHeadED	USBH_BA+0x028	R/W	Host Controller Bulk Head ED Register	0x0000_0000
HcBulkCurrentED	USBH_BA+0x02C	R/W	Host Controller Bulk Current ED Register	0x0000_0000
HcDoneHead	USBH_BA+0x030	R/W	Host Controller Done Head Register	0x0000_0000
HcFmInterval	USBH_BA+0x034	R/W	Host Controller Frame Interval Register	0x0000_2EDF

Register	Offset	R/W	Description	Reset Value
HcFmRemaining	USBH_BA+0x038	R	Host Controller Frame Remaining Register	0x0000_0000
HcFmNumber	USBH_BA+0x03C	R	Host Controller Frame Number Register	0x0000_0000
HcPeriodicStart	USBH_BA+0x040	R/W	Host Controller Periodic Start Register	0x0000_0000
HcLSThreshold	USBH_BA+0x044	R/W	Host Controller Low-speed Threshold Register	0x0000_0628
HcRhDescriptorA	USBH_BA+0x048	R/W	Host Controller Root Hub Descriptor A Register	0x0000_0908
HcRhDescriptorB	USBH_BA+0x04C	R/W	Host Controller Root Hub Descriptor B Register	0x0000_0000
HcRhStatus	USBH_BA+0x050	R/W	Host Controller Root Hub Status Register	0x0000_0000
HcRhPortStatus0	USBH_BA+0x054	R/W	Host Controller Root Hub Port Status [0]	0x0000_0000
HcRhPortStatus1	USBH_BA+0x058	R/W	Host Controller Root Hub Port Status [1]	0x0000_0000
HcRhPortStatus2	USBH_BA+0x05C	R/W	Host Controller Root Hub Port Status [2]	0x0000_0000
HcRhPortStatus3	USBH_BA+0x060	R/W	Host Controller Root Hub Port Status [3]	0x0000_0000
HcRhPortStatus4	USBH_BA+0x064	R/W	Host Controller Root Hub Port Status [4]	0x0000_0000
HcRhPortStatus5	USBH_BA+0x068	R/W	Host Controller Root Hub Port Status [5]	0x0000_0000
HcRhPortStatus6	USBH_BA+0x06C	R/W	Host Controller Root Hub Port Status [6]	0x0000_0000
HcRhPortStatus7	USBH_BA+0x070	R/W	Host Controller Root Hub Port Status [7]	0x0000_0000
HcPhyControl	USBH_BA+0x200	R/W	Host Controller PHY Control Register	0x0000_0000
HcMiscControl	USBH_BA+0x204	R/W	Host Controller Miscellaneous Control Register	0x0000_0000
EHCVNR	HSUSBH_BA+0x000	R	EHCI Version Number Register	0x0095_0020
EHCSPR	HSUSBH_BA+0x004	R	EHCI Structural Parameters Register	0x0000_0012
EHCCPR	HSUSBH_BA+0x008	R	EHCI Capability Parameters Register	0x0000_0000
UCMDR	HSUSBH_BA+0x020	R/W	USB Command Register	0x0008_0000
USTSR	HSUSBH_BA+0x024	R/W	USB Status Register	0x0000_1000
UIENR	HSUSBH_BA+0x028	R/W	USB Interrupt Enable Register	0x0000_0000
UFINDR	HSUSBH_BA+0x02C	R/W	USB Frame Index Register	0x0000_0000
UPFLBAR	HSUSBH_BA+0x034	R/W	USB Periodic Frame List Base Address Register	0x0000_0000
UCALAR	HSUSBH_BA+0x038	R/W	USB Current Asynchronous List Address Register	0x0000_0000
UASSTR	HSUSBH_BA+0x03C	R/W	USB Asynchronous Schedule Sleep Timer Register	0x0000_0BD6
UCFGR	HSUSBH_BA+0x060	R/W	USB Configure Flag Register	0x0000_0000
UPSCR0	HSUSBH_BA+0x064	R/W	USB Port 0 Status and Control Register	0x0000_2000
UPSCR1	HSUSBH_BA+0x068	R/W	USB Port 1 Status and Control Register	0x0000_2000

Register	Offset	R/W	Description	Reset Value
USBPCR0	HSUSBH_BA+0x0C4	R/W	USB PHY 0 Control Register	0x0000_0060
USBPCR1	HSUSBH_BA+0x0C8	R/W	USB PHY 1 Control Register	0x0000_0020

22 CAN

22.1 概述

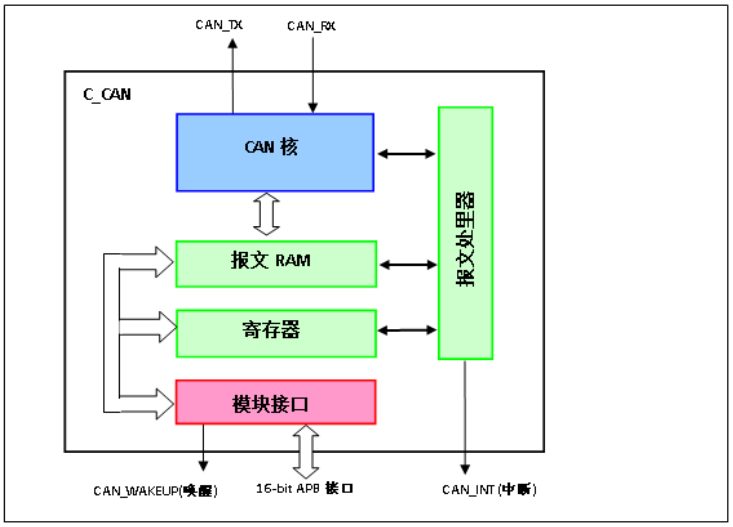
控制區域網路CAN(Controller Area Network)，首先由Robert Bosch公司提出，常用於汽車與航太工業，是以種差動式序列傳輸方式，並將車上多個控制器整合在區域網路中，藉由各個MCU分擔或分享資訊，如此便能大幅減少電線的使用量，最快的傳送速率為1Mbps。

C_CAN 由 CAN 內核，報文 RAM，報文處理器，控制寄存器和模組介面構成。CAN 內核通信符合 CAN 協定規範2.0A 和 2.0B。位元速率可達到 1MBit/s。為了和實體層相連，需要另外的收發器硬體。在 CAN 網路中通信，各個報文物件是可配置的。報文對象和用於接收報文過濾的識別字遮罩存儲在報文 RAM 中。所有關於報文處理的功能在報文處理器中執行。這些功能包括接收過濾、CAN 內核與報文 RAM 之間的報文傳輸和傳送請求以及模組中斷的產生。C_CAN 的寄存器組可以通過模組介面被軟體直接訪問，這些寄存器用來控制/配置 CAN 內核和報文處理器，以及訪問報文 RAM。

22.2 特性

- 支持 4 組 CAN
- CAN 內核通信符合 CAN 協定規範 2.0A 和 2.0B
- 位元速率可達到 1MBit/s
- 支持 32 個報文 RAM
- 每一個報文 RAM 都有個別的識別字遮罩
- 可程式化 FIFO 模式
- 可遮罩中斷
- 可不使能自動重傳功能
- 在自我測試模式下可程式化環回模式
- 支援 16 位元模組介面到 AMBA APB 匯流排
- 支援喚醒功能

22.3 方塊圖

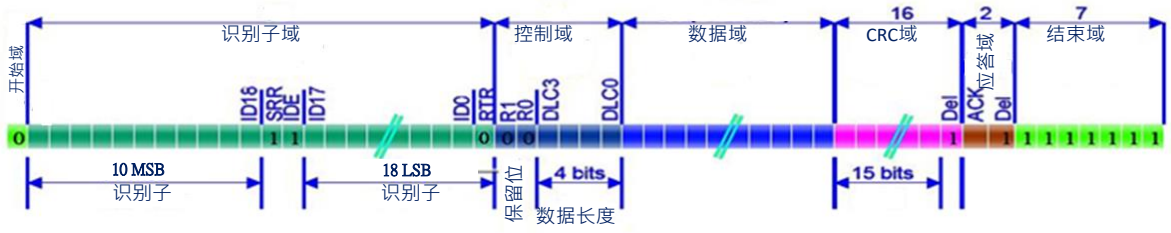


上圖為NUC980內部CAN模組的內部結構，主要包含CAN核、報文RAM、模組介面和報文處理器四個部分，其中CAN核負責錯誤偵測與處理，是CAN的主要核心；報文RAM為傳送與接收的緩衝器；模組介面為與CAN核與CPU溝通的主要介面。報文處理器為傳送與接收命令的控制中樞。

22.4 功能描述

22.4.1 CAN 協定的幀編碼格式

基本的協議可分為CAN 2.0A和2.0B兩個版本，A與B版本的差別在於識別子長度分別為11位與29位。下圖所示為一個CAN 2.0協定傳輸的資料幀。



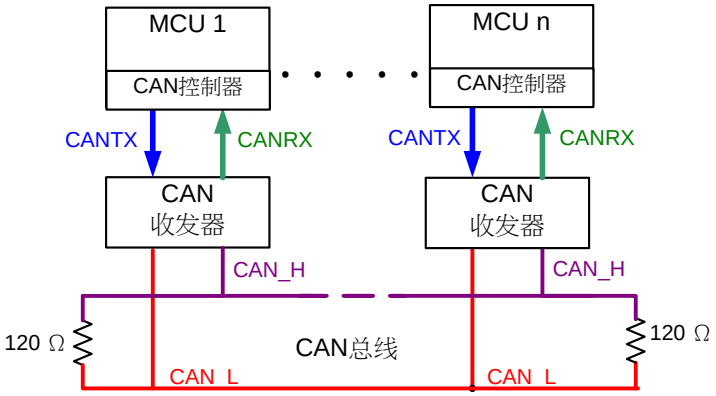
CAN資料幀編碼格式，可分為起始域、仲裁域、控制域、資料欄、 CRC域、應答域和結束域。每幀最長可以達到128位，以下分別說明各域的意義：

- 起始域 (SOF，Start of Frame)　一幀的開頭
- 仲裁域 (Arbitration Field)　用以確定該幀的優先順序
- 控制域 (Control Field)　包含著兩個保留位元和資料欄內數據的長度
- 資料欄 (Data Field)　0~8 個位元組的被傳送資料
- CRC 域(CRC Field)　從 SOF 到 Data Field 的 CRC 值，用於檢測傳輸錯誤
- 應答域 (ACK Field）　用以確認對方是否正常接收
- 結束域 (EOF，End of Frame)　相對於 SOF，是幀的結尾

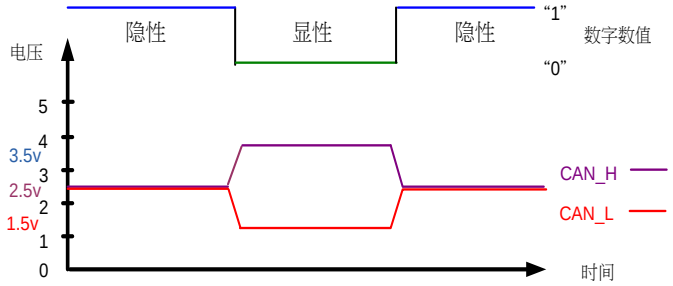
對於CAN協定的詳細內容，用於可以參考BOSCH公司的CAN技術檔。

22.4.2 CAN 硬體設定

在使用CAN控制器之前，需要正確地配置硬體，典型配置如圖 22-1所示。ISO-11898是高速硬體的硬體規範，其定義了CAN匯流排在40米內最高的傳送速率為1 Mbps。CAN匯流排通過CAN_H和CAN_L兩條線來傳送差動信號。防止終端信號反射，終端必須搭配合適的匹配阻抗，一般為120 Ω電阻。由於在 NUC980 的CAN控制器中，未包含CAN的收發器，用戶須自行選配。



收發器用於控制與偵測匯流排上資料的收送，進行匯流排上的差動信號與單端邏輯信號之間的轉換。單端邏輯信號有兩種：顯性(邏輯為1)和隱性(邏輯為0)。如下所示，當CAN_H - CAN_L > 0.9v時，判斷為隱性，當CAN_H - CAN_L < 0.6v時，判斷為顯性。



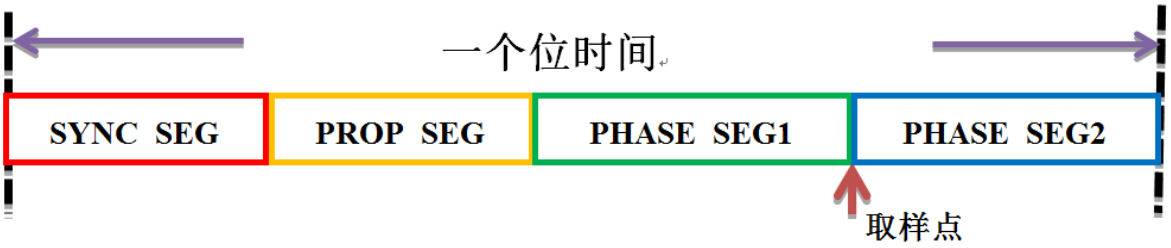
22.4.3 CAN 傳送速率的設定

CAN匯流排支援的傳送速率為1kbps~1000 kbps，根據CAN協定，CAN匯流排串列傳輸速率f_speed可表示為：$f_{speed} = {}^{1}/_{t_{NBT}}$

其中t_NBT為位時間。位時間可被分為四個不重疊的時間段，包含同步段、延遲時間段、相位

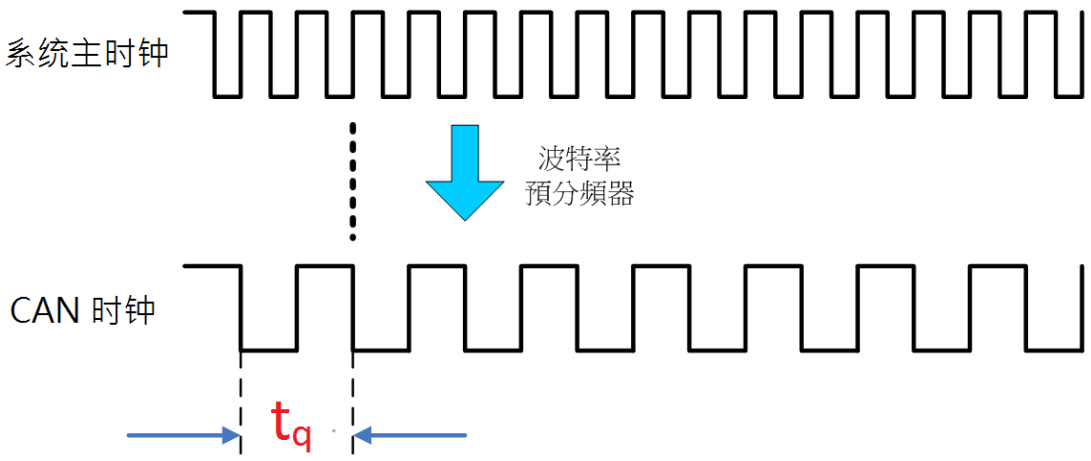
緩衝段1和相位緩衝段2，詳細定義如下：

- 同步區段，SYNC_SEG：這部分時間是用來同步在匯流排上多個節點。
- 延遲時間段，PROP_SEG：用以補償信號在網路中實體傳輸的延遲，補償的時間包括兩倍線路的延遲、輸入比較器的延遲以及輸出驅動器的延遲。
- 相位緩衝段 1/相位緩衝段 2，PHASE_SEG1 / PHASE_SEG2：用以再次同步，為延長或縮短的相位同步。



實際取樣點位置是落在相位緩衝段1與相位緩衝段2之間。位元時間是上述四個部分的時間總和，表示為：$t_{NBT} = t_{SYNC\_SEG} + t_{PROP\_SEG} + t_{PHASE\_SEG1} + t_{PHASE\_SEG2}$

其中t_{SYNC_SEG}、t_{PROP_SEG}、t_{PHASE_SEG1}與t_{PHASE_SEG2}表示各部份所占的時間，時間單位為time quantum(t_q)，通常一個位時間包含著8到25個單位時間，使用者通過控制單位時間總數來決定傳送速率。下圖為單位時間的示意圖，圖中上半部為系統主頻率，通過串列傳輸速率預分頻器分頻後，可得圖下半部的CAN時鐘源頻率，串列傳輸速率預分頻器可通過CAN_BTIME寄存器中的BRP控制位來設定。



如圖 22-2可知，單位時間t_q可表示為：

$$t_q = (BPR + 1) / f_{APB\_CLK}$$

其中BPR為串列傳輸速率預分頻器的分頻值，由CAN_BTIME寄存器中的BPR控制位來設置，f_{APB_CLK}為系統主頻率。

在CAN控制器中

$\text{TSEG1} + 1 = (t_{PROP_{SEG}} + t_{PHASE\_SEG1})t_q$

$\text{TSEG2} + 1 = (t_{PHASE\_SEG2})t_q$

$t_{SYNC\_SEG} = 1\ t_q$

其中，TSEG1和TSEG2是CAN_BTIME寄存器中的控制位。

綜上可得：

$f_{speed} = {}^{1}/_{t_{NBT}} = {}^{1}/_{(\ t_{SYNC\_SEG}\ + t_{PROP\_SEG} + t_{PHASE\_SEG1} + t_{PHASE\_SEG2})}$

$= {}^{1}/_{(1 + (TSEG1 + 1) + (TSEG2 + 1))t_q}$

$= {}^{1}/_{(TSEG1 + TSEG2 + 3)\ ({}^{(BPR + 1)}/_{f_{APB\_CLK}})}$

$= {}^{f_{APB\_CLK}}/_{(TSEG1 + TSEG2 + 3)\ (BPR + 1)}$

其中：f_{APB_CLK}為系統主時鐘，TSEG1、TSEG2與BPR皆為CAN_BITME 寄存器中的控制位域。

例如，欲設定CAN 總線速度為1000kbps，而CPU APB時鐘為75 MHz，則可設定TSEG1 =6、TSEG2 =6, BPR =4，即可得：

$f_{speed} = {}^{f_{APB\_CLK}}/_{(TSEG1 + TSEG2 + 3)\ (BPR + 1)}$

$= {}^{75000000}/_{(6 + 6 + 3)\ (4 + 1)}$

$= 1000\ kbps$

也可把TSEG1設為7、TSEG2設為5，其餘參數不變，亦可將CAN速度設為1000 kbps，與前者不同的是取樣點位置的不同.

22.4.4 CAN 模塊的寄存器

CAN模組寄存器組基底位址為CAN0_BA=0xB800_0000，寄存器可分為三類：CAN協定相關寄存器、報文介面寄存器與報文處理寄存器，CAN模組寄存器組基底位址為，如下表所列：

寄存器組	偏移位址	所包含寄存器名稱	
CAN協議相關寄存器	0x00 ~ 0x18	CAN_CON	CAN_STATUS
		CAN_ERR	CAN_BTIME
		CAN_IIDR	CAN_TEST

		CAN_BRPE	
報文介面寄存器	0x20 ~ 0xA8	CAN_IFn_CREQ*	CAN_IFn_CMASK*
		CAN_IFn_MASK1*	CAN_IFn_MASK2*
		CAN_IFn_ARB1*	CAN_IFn_ARB2*
		CAN_IFn_MCON*	CAN_IFn_DAT_An*
		CAN_IFn_DAT_Bn*	
報文處理寄存器	0x100 ~ 0x164	CAN_TXREQn*	CAN_NDATn*
		CAN_IPNDn*	CAN_MVLD1n*

*： n=1或2

- CAN 協定相關寄存器集合

這些寄存器用於CAN核控制工作模式和設定CAN串列傳輸速率，並包含著一些與CAN控制器協定相關狀態報文。

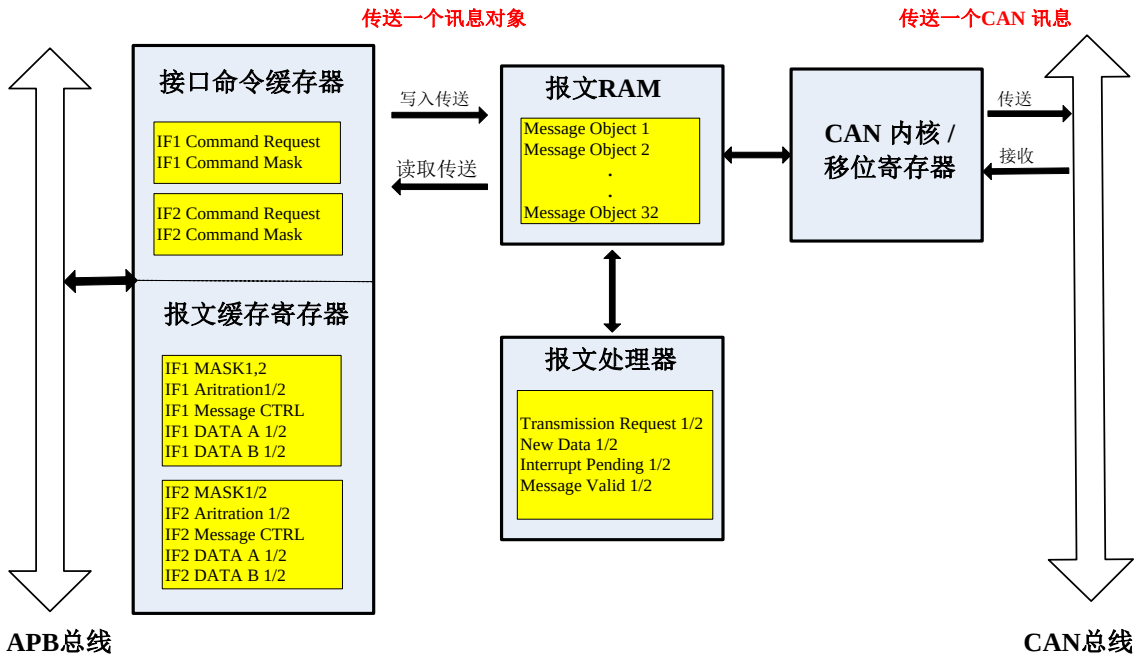
- 報文介面寄存器集合

可根據介面CAN_IF1和CAN_IF2分為兩組，通常用於控制CPU去存取報文RAM中的資料，這些介面寄存器是為了防止CPU同時去存取報文RAM，避免接收或發送CAN報文時可能的衝突。

- 報文處理寄存器集合

所有的報文處理寄存器都是唯讀的，包含報文物件中的TxRqst、NewDat、IntPnd和MsgVal位。中斷ID號也提供報文處理機制中的狀態資訊。

以上這些寄存器在CAN模組內部之間的關係如下圖所示。



介面命令寄存器和報文緩衝寄存器屬於報文介面寄存器集合。介面命令寄存器用於控制資料進出報文RAM；報文緩衝寄存器用來儲存從報文RAM接收或向報文RAM發送的報文，其中存儲了報文物件與遮罩ID。報文處理器用於控制Rx/Tx移位暫存器和報文RAM之間的動作，並可以產生相關中斷。CAN內核/移位暫存器用於報文RAM和CAN匯流排之間的串並轉換。

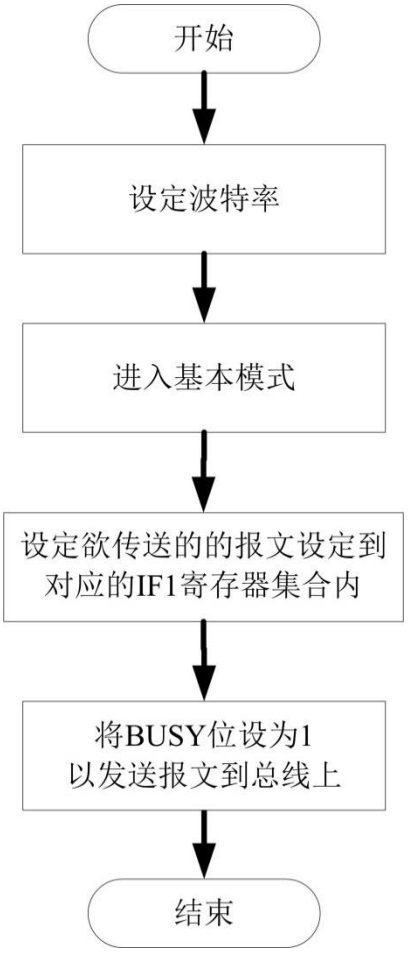
首先，使用者需要通過介面命令寄存器，來設定或取得報文資訊到報文RAM中，並通過介面命令寄存器傳送命令給CAN，報文緩存寄存器用以儲存所需要的命令資料。

22.4.5 發送 CAN 報文

CAN有兩種模式：正常模式(Normal Mode)與基本模式(Basic Mode)，其中基本模式是CAN的測試模式。

在基本模式中，報文緩衝的資料不會被寫到報文RAM中，且資料不需要被報文處理器控制，不用決定何時發送與接收，因此使用者可以先使用此模式做基本調適。換言之，CAN可以直接接收或發送CAN報文，而不通過報文RAM。當設定為基本模式時，CAN_IF1寄存器集合是用來傳送報文，CAN_IF2寄存器集合則是用來接收報文，當CAN內核接收到任何報文後，都會將報文儲存到IF2寄存器集合中。

基本模式發送報文的流程如下圖所示。



可依照以下步驟完成一個通過基本模式發送一個CAN報文到匯流排上的動作。

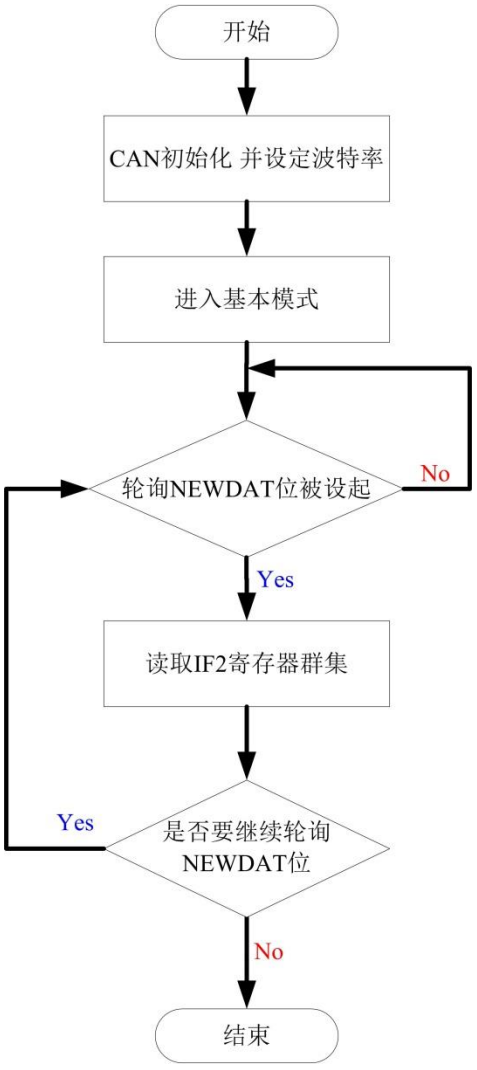
1. 參考 22.4.3 小節，完成 CAN 串列傳輸速率的設定。
2. 接著允許 CAN_CON 寄存器中的 TEST 位、CAN_TEST 寄存器中的 BASIC 位，即可讓 CAN 控制器進入基本模式。
3. 根據 CAN 報文格式，在 IF1 寄存器中填入對應的值。
4. 將 CAN_CREQ 寄存器中的 BUSY 置“1”，即可發送根據 IF1 寄存器集合中的 CAN 報文資訊，待發送完畢後 BUSY 位會自動清除。

當使用基本模式發送報文時，請特別注意以下事情：

- 確認 CAN 是否已進入測試模式。
- 確認是否 CAN_CREQ 寄存器中的 BUSY 位已被置“1”，以便要求發送，從而將 CAN 報文發送到 CAN 匯流排上。

22.4.6 接收 CAN 報文

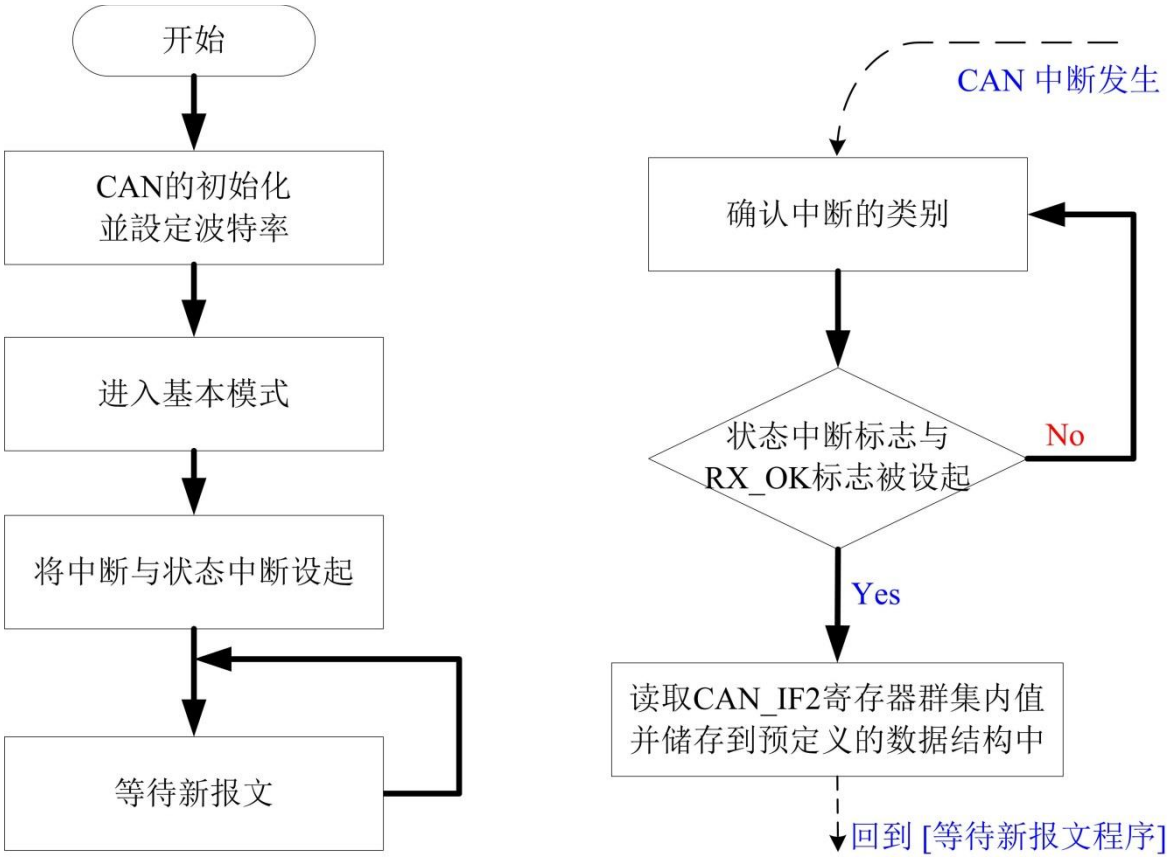
接收報文有兩種方式，一種是使用輪詢方式確認NEWDAT位元是否被置位，另一種則是利用RxOK的狀態中斷，兩者皆可確認是否已接收到新報文，新報文會被儲存到CAN_IF2寄存器集合中。下圖是採用輪詢方式接收CAN匯流排上報文的流程。



可依照以下步驟，在基本模式下利用輪詢方式接收在匯流排上的報文。

1. 參考 22.4.3 小節，完成 CAN 串列傳輸速率的設定。
2. 允許 CAN_CON 寄存器中的 TEST 位與 CAN_TEST 寄存器中的 BASIC 位，即可讓 CAN 控制器進入基本模式。
3. 輪詢 CAN_IF2_MCON 寄存器中的 NEWDAT 位，直到該位已被置“1”，表示收到 CAN 報文並已儲存到 CAN_IF2 寄存器集合中。
4. 讀取 CAN_IF2 寄存器集合內相對應欄位的值，可獲取已接收的報文。

下圖所示為如何用中斷來接收報文的流程。



依照下面步驟來完成基本模式下使用中斷來接收資料幀的程式。

1. 首先，先對 CAN 做初始化，並請參考 22.4.3 小節完成 CAN 串列傳輸速率設定。
2. 允許 CAN_CON 寄存器中的 TEST 位與 CAN_TEST 寄存器中的 BASIC 位，即可讓 CAN 控制器進入基本模式。
3. 允許 CAN_CON 寄存器中的 IE 和 SIE 位，即表示已經開啟狀態中斷的功能。
4. 直到狀態中斷發生。如果狀態中斷發生且 CAN_STATUS 寄存器中的 RxOK 標誌被置“1”時，則表示已接收到來自匯流排的報文，此時報文會被儲存到 CAN_IF2 寄存器集合中，使用者可讀取 CAN_IF2 寄存器集合來獲取已接收的報文。

22.4.7 喚醒功能

設置CAN_WU_EN寄存器的WAKEUP_EN位可以啟用喚醒功能。當晶片在掉電模式，發送任一個CAN報文到匯流排上，晶片將被喚醒

22.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
CAN0_BA = 0xB00A_0000 **CAN1_BA = 0xB00A_1000** **CAN2_BA = 0xB00A_2000** **CAN3_BA = 0xB00A_3000**				
CAN_CON	CANx_BA+0x00	R/W	Control Register	0x0000_0001
CAN_STATUS	CANx_BA+0x04	R/W	Status Register	0x0000_0000
CAN_ERR	CANx_BA+0x08	R	Error Counter	0x0000_0000
CAN_BTIME	CANx_BA+0x0C	R/W	Bit Timing Register	0x0000_2301
CAN_IIDR	CANx_BA+0x10	R	Interrupt Identifier Register	0x0000_0000
CAN_TEST	CANx_BA+0x14	R/W	Test Register	*(1)
CAN_BRPE	CANx_BA+0x18	R/W	BRP Extension Register	0x0000_0000
CAN_IF1_CREQ **CAN_IF2_CREQ**	CANx_BA+0x20 CANx_BA+0x80	R/W	IFn (*2) Command Request Registers	0x0000_0001
CAN_IF1_CMASK **CAN_IF2_CMASK**	CANx_BA+0x24 CANx_BA+0x84	R/W	IFn Command Mask Registers	0x0000_0000
CAN_IF1_MASK1 **CAN_IF2_MASK1**	CANx_BA+0x28 CANx_BA+0x88	R/W	IFn Mask 1 Register	0x0000_FFFF
CAN_IF1_MASK2 **CAN_IF2_MASK2**	CANx_BA+0x2C CANx_BA+0x8C	R/W	IFn Mask 2 Register	0x0000_FFFF
CAN_IF1_ARB1 **CAN_IF2_ARB1**	CANx_BA+0x30 CANx_BA+0x90	R/W	IFn Arbitration 1 Register	0x0000_0000
CAN_IF1_ARB2 **CAN_IF2_ARB2**	CANx_BA+0x34 CANx_BA+0x94	R/W	IFn Arbitration 2 Register	0x0000_0000
CAN_IF1_MCON **CAN_IF2_MCON**	CANx_BA+0x38 CANx_BA+0x98	R/W	IFn Message Control Registers	0x0000_0000
CAN_IF1_DAT_An/ **CAN_IF1_DAT_Bn/** **CAN_IF2_DAT_An/** **CAN_IF2_DAT_Bn/**	CANx_BA+0x3C~40 CANx_BA+0x44~48 CANx_BA+0x9C~A0 CANx_BA+0xA4~A8	R/W	IFn Data An (*3) and Data Bn (*3) Registers eg: CAN_IF1_DAT_A1 = CAN_BA+0x3Ch CAN_IF1_DAT_A2 = CAN_BA+0x40h	0x0000_0000
CAN_TXREQ1 **CAN_TXREQ2**	CANx_BA+0x100 CANx_BA+0x104	R	Transmission Request Registers 1 & 2	0x0000_0000
CAN_NDAT1 **CAN_NDAT2**	CANx_BA+0x120 CANx_BA+0x124	R	New Data Registers 1 & 2	0x0000_0000
CAN_IPND1 **CAN_IPND2**	CANx_BA+0x140 CANx_BA+0x144	R	Interrupt Pending Registers 1 & 2	0x0000_0000

CAN_MVLD1 **CAN_MVLD2**	CANx_BA+0x160 CANx_BA+0x164	R	Message Valid Registers 1 & 2	0x0000_0000
CAN_WU_EN	CANx_BA+0x168	R/W	Wake-up Function Enable	0x0000_0000
CAN_WU_STATUS	CANx_BA+0x16C	R/W	Wake-up Function Status	0x0000_0000

23 閃存接口控制器 (FMI)

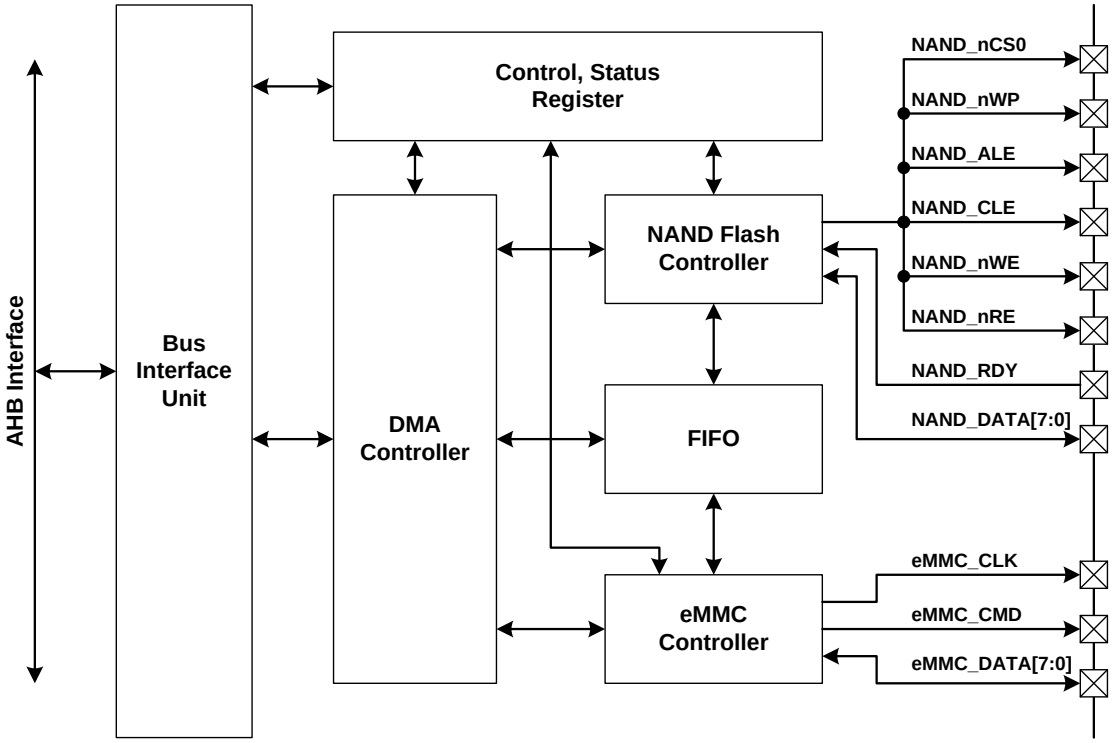
23.1 概述

閃存接口控制器（FMI）分為DMA和FMI二個單元。DMA單元提供了一個DMA（直接存儲器存取）功能用於FMI交換系統存儲器（例如SDRAM）和共享緩衝器（128字節）的數據，而FMI單元主要控制eMMC/SD或NAND閃存的接口。閃存接口控制器支持eMMC/SD和NAND型閃存和FMI與DMAC合作，以提供系統存儲器和卡之間的快速數據傳送。

23.2 特性

- 支持單一 DMA 通道和 non-word boundary 地址。
- 支持硬件分散收集功能。
- 支持 128 字節共享緩存於系統內存和閃存設備之間的數據交換。（分成兩個 64 字節乒乓 FIFO）。
- 支持 eMMC/SD 的閃存設備。
- 支持 SLC 和 MLC NAND 型閃存。
- 可調 NAND 頁面大小。2048B+備用區，4096B+備用區和 8192B+備用區。
- 支持 8 位/12 位/24 位硬件 ECC 運算電路來保護數據通信。
- 支持可編程的 NAND 定時週期。

23.3 方塊圖



23.4 功能描述

閃存接口（FMI）分為DMA和FMI二個單元，在FMI單元內又分成NAND控制器和SD控制器，下面章節會分開描述程序步驟。

23.4.1 DMA 以及 FMI 全域控制

DMA控制器提供一個直接存儲器存取功能，用戶只需簡單地填寫起始地址和使能DMAC，然後DMAC就可以自動處理數據傳輸。DMA控制器裡有一個128字節的共享緩存，分成兩個64字節乒乓FIFO（總共128字節）。它可以使用乒乓機制提供多塊傳輸。當FMI不忙，這些共享緩衝器可以通過軟件直接訪問。

閃存接口支持SD和NAND型閃存。FMI與DMAC合作，提供系統內存和卡之間的快速數據傳輸。由於DMAC只提供單一個通道，這意味著只有一個接口可以在同一時間是啟動的，SD和NAND是不能同時並存。

始能FMI和DMAC的步驟如下：

1. 設置FMI_DMACTL寄存器DMACEN位和DMARST位。
2. 等待FMI_DMACTL寄存器DMARST位清除。
3. 設置FMI_GCTL寄存器GCTLRST位。
4. 等待FMI_GCTL寄存器GCTLRST位清除。

23.4.2 NAND Flash

FMI提供NAND型閃存的訪問的接口。這個NAND型閃存控制器提供了所有必需的信號，可以由軟件依據設備規範任意的產生（例如命令端口，地址端口和數據端口）。它支持四種不同的頁面大小，2048字節，4096字節和8192字節。對於不同的NAND，必需要調整時序參數（FMI_NANDTMCTL寄存器）來符合NAND閃存設備規範，定時的調整參數還可以提高數據傳輸的性能。

NAND型閃存控制器配備一個BCH算法錯誤校正。該BCH算法可以校正多達 8-bit，12-bit 或 24-bit 的錯誤。通過讀取FMI_NANDINTSTS寄存器ECC_FLD_IF位來檢查錯誤發生，而通過讀取FMI_NANDECCESn寄存器得知有多少錯誤，而這些錯誤是否可以校正。如果可以校正，就必需讀取FMI_NANDECCEAx和FMI_NANDECCEDx寄存器來手動更正錯誤。

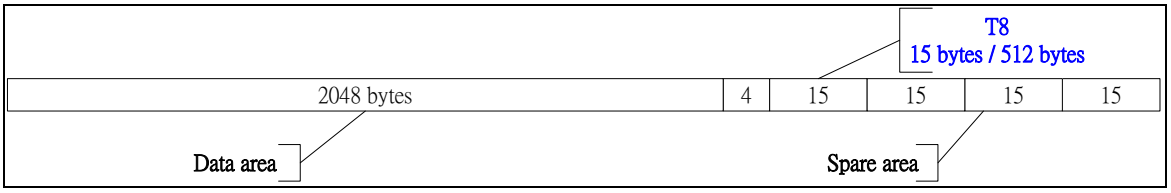
閃存接口FMI對於不同的頁面BCH算法有 8-bit，12-bit 或 24-bit等多種選擇，根據頁面大小和T設定，FMI會生成不同大小的奇偶校驗數據。下表中列出的不同頁面大小和T設置的奇偶校驗數據的字節數。 冗餘區域的數據排列如下圖所示。

建議根據NAND閃存頁面大小和冗餘區域大小選擇合適的T算法。

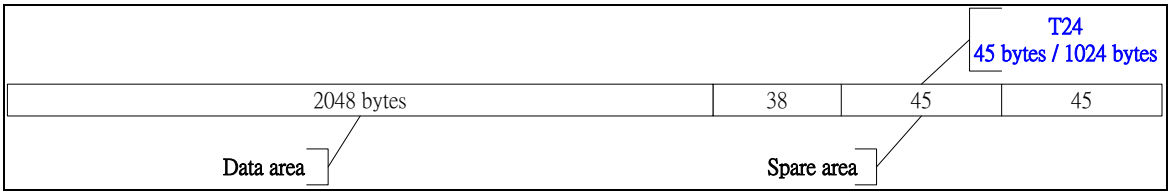
BCH algorithm	Parity (Byte) 2048 Page size	Parity(Byte)4096 Page size	Parity (Byte) 8192 Page size
BCH T8	60	120	240
BCH T12	92	184	368
BCH T24	90	180	360

下面舉例說明冗餘區域數據的分布：

1. 頁面大小 2048 + 64 字節，BCH T8，備用區域佈局為



2. 頁面大小 2048 + 128 字節，BCH T24，備用區域佈局為



有關設備的詳細程序規則，請參考"Software Driver of SmartMedia"，"SmartMedia Electrical Specifications"，"SmartMedia Physical Format Specifications"和"SmartMedia Logical Format

Specifications"。

23.4.2.1 NAND 初始化

初始化NAND的步驟如下：

1. 設置CLK_HCLKEN寄存器FMI，NAND位。
2. 選擇多功能控制，設定SYS_GPC_MFPL填入0x33333330，SYS_GPC_MFPH要填入0x33333333。
3. 設置FMI_GCTL寄存器NAND_EN位，始能NAND功能。
4. 設置FMI_NANDECTL寄存器WP位，解除NAND閃存寫保護。
5. 設置FMI_NANDCTL寄存器CS0位為0來控制NAND。

23.4.2.2 復位 NAND-type Flash

復位NAND型閃存包含以下步驟：

1. 發送RESET命令0xFF到FMI_NANDCMD寄存器。
2. 等待RB＃。檢查FMI_NANDINTSTS寄存器RB0_IF位，直到它設置。然後清除FMI_NANDINTSTS寄存器RB0_IF位。

23.4.2.3 識別 NAND-type Flash

識別NAND型閃存包含以下步驟：

1. 發送讀取ID命令0x90到FMI_NANDCMD寄存器。
2. FMI_NANDADDR寄存器ADDRESS位填入地址0x00，並設置EOA位。
3. 從FMI_NANDDATA寄存器讀取ID。
4. 從ID判斷NAND的頁面大小，需要多少bit的校正，而設定FMI_NANDCTL寄存器PSIZE和BCH_TSEL位。
5. 依據ID或是規格設定冗餘區域大小（FMI_NANDRACTL寄存器RA128EN位）。

23.4.2.4 擦除 NAND-type Flash

NAND型閃存擦除塊的步驟：

1. 發送擦除命令0x60到FMI_NANDCMD寄存器。
2. 將行地址（Row Address）從低到高依序填入FMI_NANDADDR寄存器，請參考下表。
3. 設置FMI_NANDADDR寄存器EOA位。

4. 發送擦除命令0xD0到FMI_NANDCMD寄存器。
5. 等待RB＃。檢查FMI_NANDINTSTS寄存器RB0_IF位，直到它設置。然後清除FMI_NANDINTSTS寄存器RB0_IF位。
6. 發送讀取狀態命令0x70到FMI_NANDCMD寄存器。
7. 從FMI_NANDDATA寄存器讀取狀態並檢查0位，1是失敗；0是通過。

Addr Cycle	D7	D6	D5	D4	D3	D2	D1	D0	
1st Cycle	A7	A6	A5	A4	A3	A2	A1	A0	Column Address
2nd Cycle	L	L	A13	A12	A11	A10	A9	A8	
3rd Cycle	A21	A20	A19	A18	A17	A16	A15	A14	Row Address Page address: A14~A21 Block Address: A22 ~
4th Cycle	A29	A28	A27	A26	A25	A24	A23	A22	
5th Cycle	L	L	L	L	A33	A32	A31	A30	L: must be "Low"

23.4.2.5 寫入 NAND-type Flash

NAND型閃存Page Write的步驟：

1. 將目標地址填入FMI_DMASA寄存器。
2. FMI_NANDRA0寄存器填入0x0000FFFF，代表這頁被使用了。
3. 發送串列輸入數據命令0x80到FMI_NANDCMD寄存器。
4. 將列地址（Column Address，這裡通常是填0，都是以一頁為起始點）從低到高依序填入FMI_NANDADDR寄存器。
5. 將行地址（Row Address）從低到高依序填入FMI_NANDADDR寄存器。
6. 設置FMI_NANDADDR寄存器EOA位。
7. 清除FMI_NANDINTSTS寄存器DMA_IF、ECC_FLD_IF、PROT_REGION_WR_IF位。
8. 設置FMI_NANDCTL寄存器REDUN_AUTO_WEN位，始能自動寫入冗餘區域（Redundant Area）。
9. 設置FMI_NANDCTL寄存器DWR_EN位，始能DMA輸出數據到NAND。
10. 輪詢DWR_EN位直到被清除，或等待FMI_NANDINTSTS寄存器DMA_IF位。
11. 發送自動程序命令0x10到FMI_NANDCMD寄存器。
12. 等待RB＃。檢查FMI_NANDINTSTS寄存器RB0_IF位，直到它設置。然後清除FMI_NANDINTSTS寄存器RB0_IF位。

13. 發送讀取狀態命令0x70到FMI_NANDCMD寄存器。
14. 從FMI_NANDDATA寄存器讀取狀態並檢查0位，1是失敗；0是通過。

23.4.2.6 讀取 NAND-type Flash

NAND型閃存Page Read之前要先將冗餘區域先讀出，NAND控制器才能夠處理錯誤校正。整個Page Read的步驟如下：

1. 從FMI_NANDRACTL寄存器RA128EN位獲得冗餘區域大小。
2. 發送讀取數據命令0x00到FMI_NANDCMD寄存器。
3. 將列地址（Column Address，這裡通常是填入頁大小）從低到高依序填入FMI_NANDADDR寄存器。
4. 將行地址（Row Address）從低到高依序填入FMI_NANDADDR寄存器。
5. 設置FMI_NANDADDR寄存器EOA位。
6. 發送第二讀取數據命令0x30到FMI_NANDCMD寄存器。
7. 等待RB＃。檢查FMI_NANDINTSTS寄存器RB0_IF位，直到它設置。然後清除FMI_NANDINTSTS寄存器RB0_IF位。
8. 根據冗餘區域大小，通過FMI_NANDDATA寄存器一一讀出。
9. 讀取數據。重複步驟2～7，列地址（Column Address）此時應填0，以一頁為起始點。
10. 將目標地址填入FMI_DMASA寄存器。
11. 清除FMI_NANDINTSTS寄存器DMA_IF和ECC_FLD_IF位。
12. 設置FMI_NANDCTL寄存器DRD_EN位，始能DMA從NAND輸入數據。
13. 輪詢DRD_EN位直到被清除，或等待FMI_NANDINTSTS寄存器DMA_IF位。
14. 如果FMI_NANDINTSTS寄存器ECC_FLD_IF位被置1，表示數據有錯，要啟動校正檢測（詳細請參考錯誤校正步驟）。

23.4.2.7 NAND-type Flash ECC 校驗

BCH算法錯誤校正可以校正多達 8-bit、12-bit或24-bit的錯誤，除了24-bit是以1024字節為單位計算外，其他的都是以512字節為單位。

NAND型閃存錯誤校正的步驟：

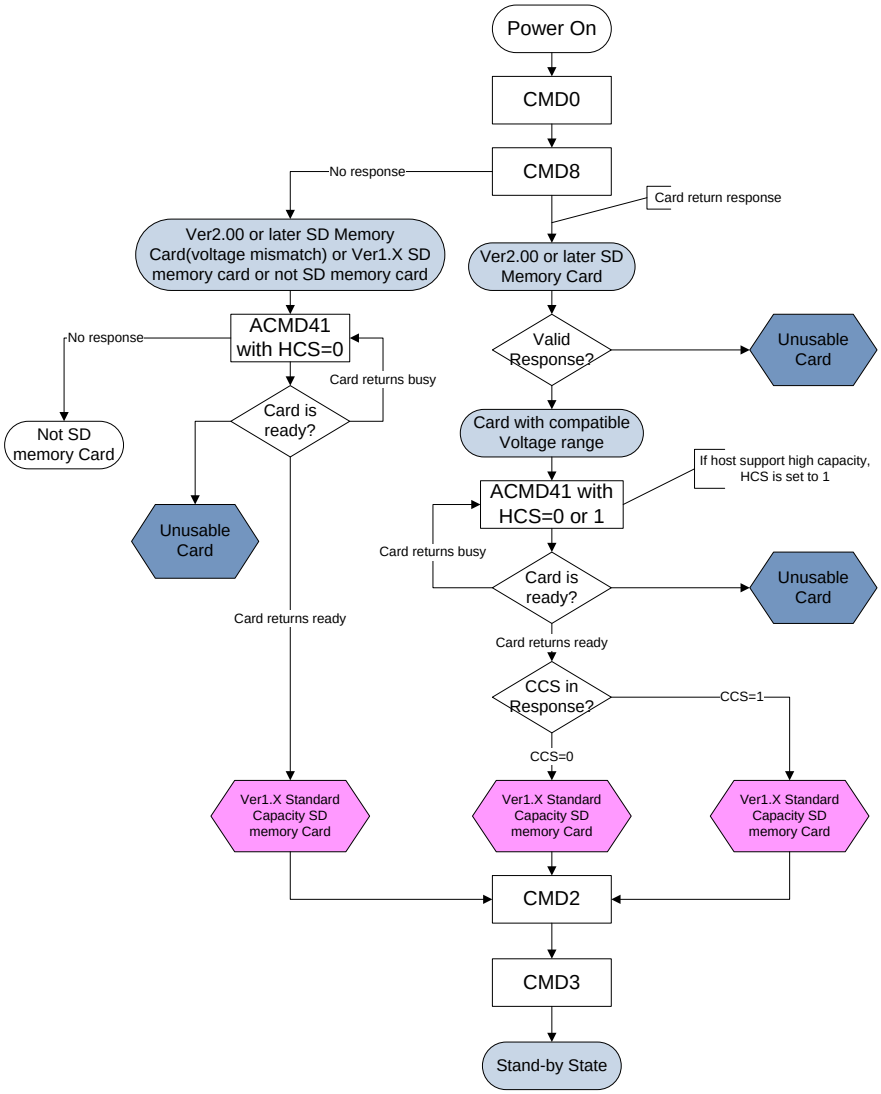
1. 讀取FMI_NANDECCESn寄存器Fx_STAT位，檢查錯誤是否可以校正。
2. 如果能夠校正，讀取FMI_NANDECCESn寄存器Fx_ECNT位，得知錯誤數量。
3. 根據頁面大小和BCH幾bit校正換算區域，計算合法的FMI_NANDECCEDn和FMI_NANDECCEAn寄存器。

4. 從FMI_NANDECCEDn寄存器讀取錯誤的數據，然後根據FMI_NANDECCEAn寄存器得到數據錯誤的地址並取得輸入的數據，兩個數據做XOR，其結果就是正確的數據。

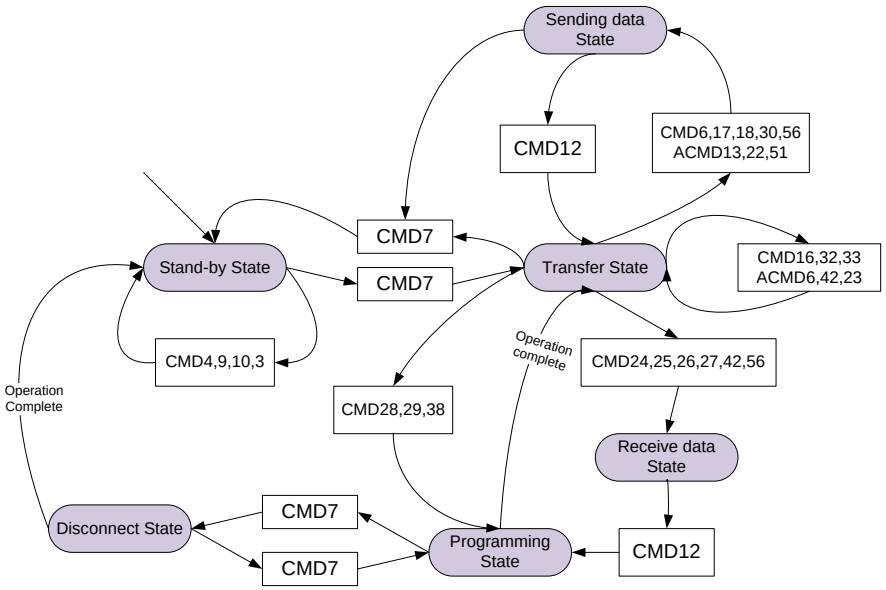
23.4.3 SD/eMMC

FMI提供一個用於SD/eMMC裝置的訪問接口，這個SD/MMC控制器支持1位/ 4位數據總線模式，該控制器可以生成所有類型的48位命令以及獲得設備響應，響應的內容將被存儲在FMI_EMMCRESP0和FMI_EMMCRESP1寄存器。關於輸出到SD/eMMC設備的頻率則需要透過CLKDIV3寄存器控制，有關設備的詳細程序規則，請參考"SD Memory Card Specifications Part 1"，"The MultiMediaCard System Specification"，" JEDEC Standard No. 84-A441 "以及各廠商的eMMC datasheet。

SD存儲卡狀態圖（卡識別模式）：



SD存儲卡狀態圖（數據傳輸模式）：



23.4.3.1 SD/eMMC 初始化

初始化SD/eMMC的步驟如下：

1. 設置CLK_HCLKEN寄存器FMI，NAND和eMMC位。
2. 選擇多功能控制，設定SYS_GPC_MFPL填入0x66600000，SYS_GPC_MFPH填入0x00060666。
3. 設置FMI_GCTL寄存器eMMCEN位，始能SD/eMMC功能。
4. 設置FMI_EMMCCTL寄存器CTLRST位。
5. 等待FMI_EMMCCTL寄存器CTLRST位清除。
6. 設定SD/eMMC初始化輸出頻率為300KHz，SD/eMMC接口為1位數據總線模式。
7. 設置FMI_EMMCCTL寄存器CLK74OE位。
8. 等待FMI_EMMCCTL寄存器CLK74OE位清除。
9. 根據設備規則發送命令給SD/eMMC。
10. 進入傳輸狀態，根據設備規則將輸出頻率設定成所需要的頻率（例如25MHz），另外，SD/eMMC接口設定為4位數據總線模式。

23.4.3.2 發送命令

發送命令給SD/eMMC的步驟如下：

1. 設置參數於FMI_EMMCCMD寄存器。
2. 設置命令於FMI_EMMCCTL寄存器CMDCODE位。

3. 設置FMI_EMMCCTL寄存器COEN位，始能命令輸出。
4. 輪詢等待FMI_EMMCCTL寄存器COEN位清除。

23.4.3.3 取得響應

得到SD/eMMC響應的步驟如下：

1. 設置FMI_EMMCCTL寄存器RIEN位，始能響應輸入。
2. 輪詢等待FMI_EMMCCTL寄存器RIEN位清除。
3. 檢查FMI_EMMCINTSTS寄存器CRC7位。
4. 響應的信息會放置在FMI_EMMCRESP0和FMI_EMMCRESP1寄存器。

23.4.3.4 讀取SD/eMMC

讀取SD/eMMC處理的步驟如下：

1. 發送CMD7進入傳輸狀態。
2. 設置FMI_EMMCCTL寄存器CLK8OE位，發送8個時鐘週期，然後檢查FMI_EMMCINTSTS寄存器的DAT0位，等待卡準備就緒。反覆輪詢直到SD/eMMC準備就緒。
3. 設置塊大小FMI_EMMCBLEN寄存器，例如512字節應填寫0x1FF。
4. 將讀取起始的扇區地址填入FMI_EMMCCMD寄存器。
5. 設置數據源地址到FMI_DMASA寄存器。
6. 檢查寫入扇區計數。如果超過255，用戶應該分次傳輸。多塊計數要填入FMI_EMMCCTL寄存器BLKCNT位（一次最多255塊）。
7. 設置CMD18讀取多個塊命令（FMI_EMMCCTL寄存器CMDCODE位填入18）。
8. 設置FMI_EMMCCTL寄存器COEN位、RIEN位和DIEN位，來使能命令輸出，響應輸入和數據輸入。
9. 輪詢DIEN位，直到被清除，或等待FMI_EMMCINTSTS寄存器的BLKD_IF中斷位。
10. 檢查FMI_EMMCINTSTS寄存器的CRC7和CRC16位。
11. 發送CMD12停止傳輸。
12. 設置FMI_EMMCCTL寄存器CLK8OE位，發送8個時鐘週期，然後檢查FMI_EMMCINTSTS寄存器的DAT0位，等待卡準備就緒。反覆輪詢直到SD/eMMC準備就緒。
13. 發送CMD7使SD/eMMC變成待機狀態。

23.4.3.5 寫入SD/eMMC

寫入SD/eMMC的步驟如下：

1. 發送CMD7進入傳輸狀態。
2. 設置FMI_EMMCCTL寄存器CLK8OE位，發送8個時鐘週期，然後檢查FMI_EMMCINTSTS寄存器的DAT0位，等待卡準備就緒。反覆輪詢直到SD/eMMC準備就緒。
3. 設置塊大小FMI_EMMCBLEN寄存器，例如512字節應填寫0x1FF。
4. 將讀取起始的扇區地址填入FMI_EMMCCMD寄存器。
5. 設置數據源地址到FMI_DMASA寄存器。
6. 檢查寫入扇區計數。如果超過255，用戶應該分次傳輸。多塊計數要填入FMI_EMMCCTL寄存器BLKCNT位（一次最多255塊）。
7. 設置CMD25寫入多個塊命令（FMI_EMMCCTL寄存器CMDCODE位填入25）。
8. 設置FMI_EMMCCTL寄存器COEN位、RIEN位和DOEN位，來使能命令輸出，響應輸入和數據輸出。
9. 輪詢DOEN位，直到被清除，或等待FMI_EMMCINTSTS寄存器的BLKD_IF中斷位。
10. 檢查FMI_EMMCINTSTS寄存器的CRC_IF位。如果CRC校驗發生錯誤，需要做軟件復位（設置FMI_EMMCCTL寄存器CTLRST位）。
11. 發送CMD12停止傳輸。
12. 設置FMI_EMMCCTL寄存器CLK8OE位，發送8個時鐘週期，然後檢查FMI_EMMCINTSTS寄存器的DAT0位，等待卡準備就緒。反覆輪詢直到SD/eMMC準備就緒。
13. 發送CMD7使SD/eMMC變成待機狀態。

23.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
FMI Base Address: **FMI_BA = 0xB001_9000**				
FMI_BUFFERn **n = 0, 1..31**	FMI_BA+0x000+0x4*n	R/W	FMI Embedded Buffer Word n n = 0, 1..31	0x0000_0000
FMI_DMACTL	FMI_BA+0x400	R/W	FMI DMA Control Register	0x0000_0000
FMI_DMASA	FMI_BA+0x408	R/W	FMI DMA Transfer Starting Address Register	0x0000_0000
FMI_DMABCNT	FMI_BA+0x40C	R	FMI DMA Transfer Byte Count Register	0x0000_0000

FMI_DMAINTEN	FMI_BA+0x410	R/W	FMI DMA Interrupt Enable Register	0x0000_0001
FMI_DMAINTSTS	FMI_BA+0x414	R/W	FMI DMA Interrupt Status Register	0x0000_0000
FMI_GCTL	FMI_BA+0x800	R/W	FMI Global Control and Status Register	0x0000_0000
FMI_GINTEN	FMI_BA+0x804	R/W	FMI Global Interrupt Control Register	0x0000_0001
FMI_GINTSTS	FMI_BA+0x808	R/W	FMI Global Interrupt Status Register	0x0000_0000
FMI_EMMCCTL	FMI_BA+0x820	R/W	SD0/eMMC0 Control Register	0x0101_0000
FMI_EMMCCMDARG	FMI_BA+0x824	R/W	SD0/eMMC0 Command Argument Register	0x0000_0000
FMI_EMMCINTEN	FMI_BA+0x828	R/W	SD0/eMMC0 Interrupt Enable Register	0x0000_0000
FMI_EMMCINTSTS	FMI_BA+0x82C	R/W	SD0/eMMC0 Interrupt Status Register	0x00XX_008C
FMI_EMMCRESP0	FMI_BA+0x830	R	SD0/eMMC0 Receiving Response Token Register 0	0x0000_0000
FMI_EMMCRESP1	FMI_BA+0x834	R	SD0/eMMC0 Receiving Response Token Register 1	0x0000_0000
FMI_EMMCBLEN	FMI_BA+0x838	R/W	SD0/eMMC0 Block Length Register	0x0000_01FF
FMI_EMMCTOUT	FMI_BA+0x83C	R/W	SD0/eMMC0 Response/Data-in Time-out Register	0x0000_0000
FMI_EMMCECR	FMI_BA+0x840	R/W	SD0/eMMC0 Extend Control Register	0x0000_0003
FMI_NANDCTL	FMI_BA+0x8A0	R/W	NAND Flash Control Register	0x0288_0090
FMI_NANDTMCTL	FMI_BA+0x8A4	R/W	NAND Flash Timing Control Register	0x0001_0105
FMI_NANDINTEN	FMI_BA+0x8A8	R/W	NAND Flash Interrupt Enable Register	0x0000_0000
FMI_NANDINTSTS	FMI_BA+0x8AC	R/W	NAND Flash Interrupt Status Register	0x00XX_0000
FMI_NANDCMD	FMI_BA+0x8B0	W	NAND Flash Command Port Register	0xXXXX_XXXX
FMI_NANDADDR	FMI_BA+0x8B4	W	NAND Flash Address Port Register	0xXXXX_XXXX
FMI_NANDDATA	FMI_BA+0x8B8	R/W	NAND Flash Data Port Register	0xXXXX_XXXX
FMI_NANDRACTL	FMI_BA+0x8BC	R/W	NAND Flash Redundant Area Control Register	0x0000_0000
FMI_NANDECTL	FMI_BA+0x8C0	R/W	NAND Flash Extend Control Register	0x0000_0000
FMI_NANDECCES0	FMI_BA+0x8D0	R	NAND Flash ECC Error Status 0 Register	0x0000_0000
FMI_NANDECCES1	FMI_BA+0x8D4	R	NAND Flash ECC Error Status 1 Register	0x0000_0000
FMI_NANDECCES2	FMI_BA+0x8D8	R	NAND Flash ECC Error Status 2 Register	0x0000_0000
FMI_NANDECCES3	FMI_BA+0x8DC	R	NAND Flash ECC Error Status 3 Register	0x0000_0000
FMI_NANDPROTA0	FMI_BA+0x8E0	R/W	NAND Flash Protect Region End Address 0 Register	0x0000_0000
FMI_NANDPROTA1	FMI_BA+0x8E4	R/W	NAND Flash Protect Region End Address 1 Register	0x0000_0000
FMI_NANDECCEA0	FMI_BA+0x900	R	NAND Flash ECC Error Byte Address 0 Register	0x0000_0000
FMI_NANDECCEA1	FMI_BA+0x904	R	NAND Flash ECC Error Byte Address 1 Register	0x0000_0000
FMI_NANDECCEA2	FMI_BA+0x908	R	NAND Flash ECC Error Byte Address 2 Register	0x0000_0000
FMI_NANDECCEA3	FMI_BA+0x90C	R	NAND Flash ECC Error Byte Address 3 Register	0x0000_0000
FMI_NANDECCEA4	FMI_BA+0x910	R	NAND Flash ECC Error Byte Address 4 Register	0x0000_0000
FMI_NANDECCEA5	FMI_BA+0x914	R	NAND Flash ECC Error Byte Address 5 Register	0x0000_0000

FMI_NANDECCEA6	FMI_BA+0x918	R	NAND Flash ECC Error Byte Address 6 Register	0x0000_0000
FMI_NANDECCEA7	FMI_BA+0x91C	R	NAND Flash ECC Error Byte Address 7 Register	0x0000_0000
FMI_NANDECCEA8	FMI_BA+0x920	R	NAND Flash ECC Error Byte Address 8 Register	0x0000_0000
FMI_NANDECCEA9	FMI_BA+0x924	R	NAND Flash ECC Error Byte Address 9 Register	0x0000_0000
FMI_NANDECCEA10	FMI_BA+0x928	R	NAND Flash ECC Error Byte Address 10 Register	0x0000_0000
FMI_NANDECCEA11	FMI_BA+0x92C	R	NAND Flash ECC Error Byte Address 11 Register	0x0000_0000
FMI_NANDECCED0	FMI_BA+0x960	R	NAND Flash ECC Error Data Register 0	0x8080_8080
FMI_NANDECCED1	FMI_BA+0x964	R	NAND Flash ECC Error Data Register 1	0x8080_8080
FMI_NANDECCED2	FMI_BA+0x968	R	NAND Flash ECC Error Data Register 2	0x8080_8080
FMI_NANDECCED3	FMI_BA+0x96C	R	NAND Flash ECC Error Data Register 3	0x8080_8080
FMI_NANDECCED4	FMI_BA+0x970	R	NAND Flash ECC Error Data Register 4	0x8080_8080
FMI_NANDECCED5	FMI_BA+0x974	R	NAND Flash ECC Error Data Register 5	0x8080_8080
FMI_NANDRAn **n = 0, 1..117**	FMI_BA+0xA00+0x4*n	R/W	NAND Flash Redundant Area Word n n = 0, 1..117	Undefined

24 SD 控制器 (SDH)

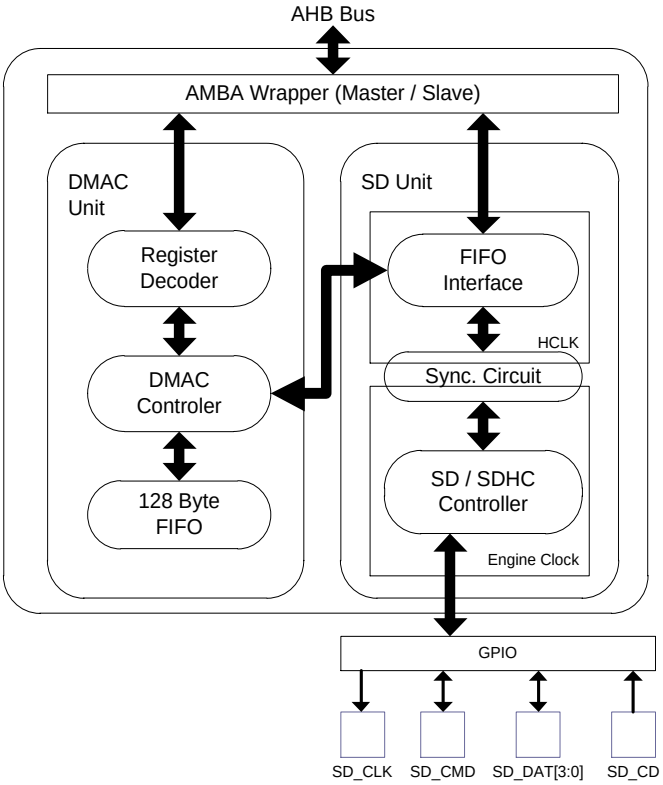
24.1 概述

安全數碼卡主機控制器（SDH）分為DMAC和SD二個單元。DMAC單元提供了一個DMA（直接存儲器存取）功能用於SD交換系統存儲器和共享緩衝器（128字節）的數據，而SD單元主要控制SD / SDHC / SDIO接口。SDH控制器支持SD/ SDHC/ SDIO卡和DMAC的合作，以提供系統內存和卡之間的快速數據傳輸。

24.2 特性

- AMBA AHB主/從接口兼容，用於數據傳輸和寄存器讀/寫。
- 支持單DMA通道。
- 支持硬件分散收集功能。
- 支持128字節的共享系統內存和卡之間的數據交換緩衝區。
- 支持SD，SDHC和SDIO卡。

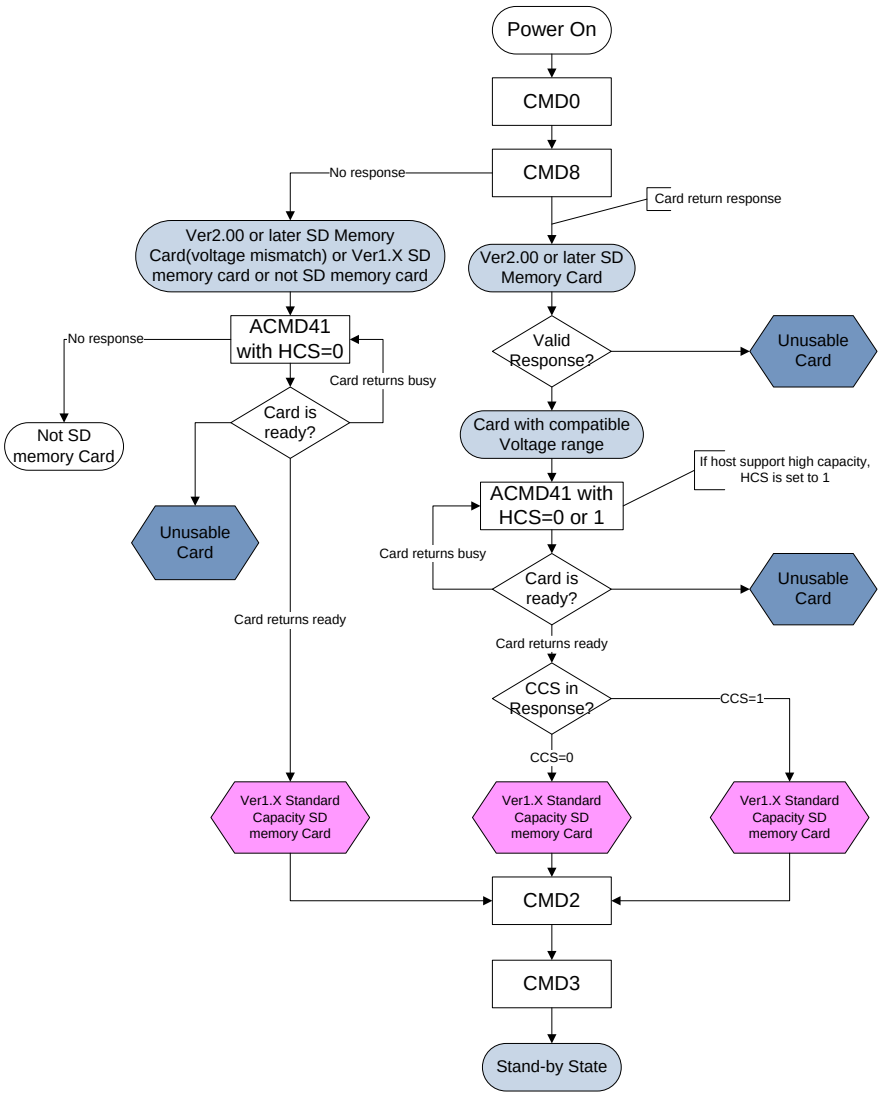
24.3 方塊圖



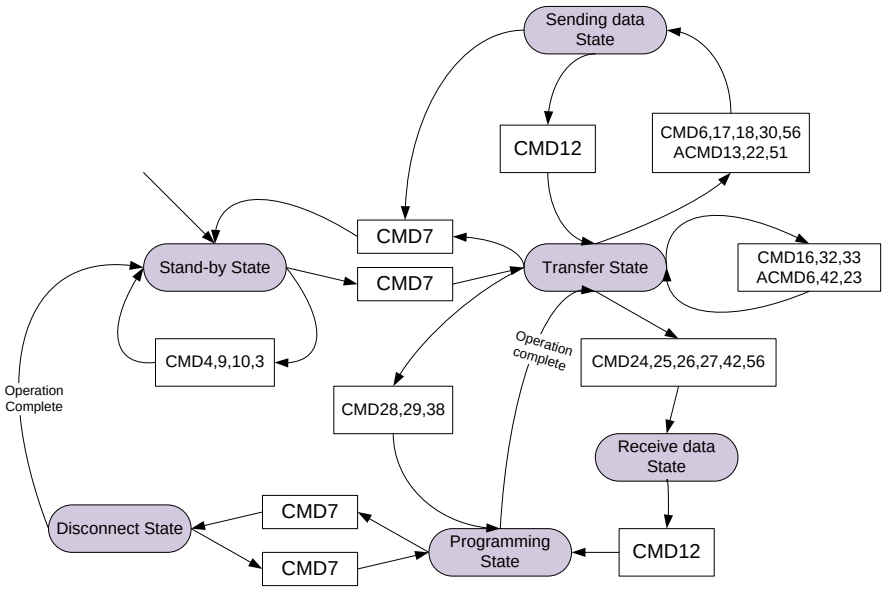
24.4 功能描述

安全數碼卡主機控制器（SDH）分為DMAC和SD二個單元，而SD單元主要控制SD / SDHC / SDIO接口，下面章節會分開描述程序步驟。

SD Memory Card State Diagram（Card Identification Mode）：



SD Memory Card State Diagram（Data Trasnfer Mode）：



24.4.1 全域控制

DMA控制器提供一個直接存儲器存取功能，用戶只需簡單地填寫起始地址和使能DMAC，然後DMAC就可以自動處理數據傳輸。DMA控制器裡有一個128字節的共享緩存，分成兩個64字節乒乓FIFO（總共128字節）。它可以使用乒乓機制提供多塊傳輸。當SDH不忙，這些共享緩衝器可以通過軟件直接訪問。

SD控制器提供1個SD端口，所有端口具有卡檢測功能和SDIO中斷。每個端口可以提供1位/ 4位的數據總線模式，輸出到SD設備的頻率需要透過CLKDIV9寄存器控制，有關設備的詳細程序規則，請參考"SD Memory Card Specifications Part 1" and "The MultiMediaCard System Specification"。

始能SDH的步驟如下：

1. 設置 CLK_HCLKEN 寄存器 SDH 位。
2. 設置SDH_DMACTL寄存器DMACEN位和DMARST位。
3. 等待SDH_DMACTL寄存器DMARST位清除。
4. 設置SDH_GCTL寄存器SDEN位和GCTLRST位。
5. 等待SDH_GCTL寄存器GCTLRST位清除。
6. 端口0只有一組多功能控制（GPF0～6），SYS_GPF_MFPL要填入0x02222222。
7. 清除SDH_ECTL寄存器PWROFF0位，始能電源控制，即供電給SD設備。
8. 設置SDH_INTEN寄存器CDxSRC位來選擇SD卡檢測源（DAT3或GPIO）。
9. 設定SDH初始化輸出頻率為300KHz，SDH接口為1位數據總線模式。
10. 設置SDH_CTL寄存器CLK74_OE位。

11. 等待SDH_CTL寄存器CLK74_OE位清除。
12. 根據設備規則發送命令給SD設備。
13. 進入傳輸狀態，根據設備規則將輸出頻率設定成所需要的頻率（例如25MHz），另外，SDH接口設定為4位數據總線模式。

24.4.2 發送命令

發送命令給SD 卡的步驟如下：

1. 設置參數於SDH_CMD寄存器。
2. 設置命令於SDH_CTL寄存器CMD_CODE位。
3. 設置SDH_CTL寄存器CO_EN位，始能命令輸出。
4. 輪詢等待SDH_CTL寄存器CO_EN位清除。

24.4.3 取得響應

得到SD 卡響應的步驟如下：

1. 設置SDH_CTL寄存器RI_EN位，始能響應輸入。
2. 輪詢等待SDH_CTL寄存器RI_EN位清除。
3. 檢查SDH_INTSTS寄存器CRC7位。
4. 響應的信息會放置在SDH_RESP0和SDH_RESP1寄存器。

24.4.4 讀取 SD 卡

讀取SD處理的步驟如下：

1. 發送CMD7進入傳輸狀態。
2. 設置SDH_CTL寄存器CLK8_OE位，發送8個時鐘週期，然後檢查SDH_INTSTS寄存器的SDDAT0位，等待卡準備就緒。反覆輪詢直到SD設備準備就緒。
3. 設置塊大小SDH_BLEN寄存器，例如512字節應填寫0x1FF。
4. 將讀取起始的扇區地址填入SDH_CMD寄存器。
5. 設置數據源地址到SDH_DMASA寄存器。
6. 檢查寫入扇區計數。如果超過255，用戶應該分次傳輸。多塊計數要填入SDH_CTL寄存器BLK_CNT位（一次最多255塊）。
7. 設置CMD18讀取多個塊命令（SDH_CTL寄存器CMD_CODE位填入18）。
8. 設置SDH_CTL寄存器CO_EN位、RI_EN位和DI_EN位，來使能命令輸出，響應輸入和數

據輸入。

9. 輪詢DI_EN位，直到被清除，或等待SDH_INTSTS寄存器的BLKD_IF中斷位。
10. 檢查SDH_INTSTS寄存器的CRC7和CRC16位。
11. 發送CMD12停止傳輸。
12. 設置SDH_CTL寄存器CLK8_OE位，發送8個時鐘週期，然後檢查SDH_INTSTS寄存器的SDDAT0位，等待卡準備就緒。反覆輪詢直到SD設備準備就緒。
13. 發送CMD7使SD變成待機狀態。

24.4.5 寫入 SD 卡

寫入SD卡的步驟如下：

1. 發送CMD7進入傳輸狀態。
2. 設置SDH_CTL寄存器CLK8_OE位，發送8個時鐘週期，然後檢查SDH_INTSTS寄存器的SDDAT0位，等待卡準備就緒。反覆輪詢直到SD設備準備就緒。
3. 設置塊大小SDH_BLEN寄存器，例如512字節應填寫0x1FF。
4. 將讀取起始的扇區地址填入SDH_CMD寄存器。
5. 設置數據源地址到SDH_DMASA寄存器。
6. 檢查寫入扇區計數。如果超過255，用戶應該分次傳輸。多塊計數要填入SDH_CTL寄存器BLK_CNT位（一次最多255塊）。
7. 設置CMD25寫入多個塊命令（SDH_CTL寄存器CMD_CODE位填入25）。
8. 設置SDH_CTL寄存器CO_EN位、RI_EN位和DO_EN位，來使能命令輸出，響應輸入和數據輸出。
9. 輪詢DO_EN位，直到被清除，或等待SDH_INTSTS寄存器的BLKD_IF中斷位。
10. 檢查SDH_INTSTS寄存器的CRC_IF位。如果CRC校驗發生錯誤，需要做軟件復位（設置SDH_CTL寄存器SW_RST位）。
11. 發送CMD12停止傳輸。
12. 設置SDH_CTL寄存器CLK8_OE位，發送8個時鐘週期，然後檢查SDH_INTSTS寄存器的SDDAT0位，等待卡準備就緒。反覆輪詢直到SD設備準備就緒。
13. 發送CMD7使SD變成待機狀態。

24.5 寄存器

R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
SDH_BA = 0xB001_8000				
SDH_FB_n **n = 0,1…31**	SDH_BA+0x000 + 0x4 * n	R/W	SD Host Embedded Buffer Word n n = 0,1…31	0x0000_0000
SDH_DMACTL	SDH_BA+0x400	R/W	SD Host DMA Control and Status Register	0x0000_0000
SDH_DMASA	SDH_BA+0x408	R/W	SD Host DMA Transfer Starting Address Register	0x0000_0000
SDH_DMABCNT	SDH_BA+0x40C	R	SD Host DMA Transfer Byte Count Register	0x0000_0000
SDH_DMAINTEN	SDH_BA+0x410	R/W	SD Host DMA Interrupt Enable Register	0x0000_0001
SDH_DMAINTSTS	SDH_BA+0x414	R/W	SD Host DMA Interrupt Status Register	0x0000_0000
SDH_GCTL	SDH_BA + 0x800	R/W	SD Host Global Control and Status Register	0x0000_0000
SDH_GINTEN	SDH_BA + 0x804	R/W	SD Host Global Interrupt Control Register	0x0000_0001
SDH_GINTSTS	SDH_BA + 0x808	R/W	SD Host Global Interrupt Status Register	0x0000_0000
SDH_CTL	SDH_BA + 0x820	R/W	SD Host Control and Status Register	0x0101_0000
SDH_CMD	SDH_BA + 0x824	R/W	SD Host Command Argument Register	0x0000_0000
SDH_INTEN	SDH_BA + 0x828	R/W	SD Host Interrupt Enable Register	0x0000_0A00
SDH_INTSTS	SDH_BA + 0x82C	R/W	SD Host Interrupt Status Register	0x000X_008C
SDH_RESP0	SDH_BA + 0x830	R	SD Host Receiving Response Token Register 0	0x0000_0000
SDH_RESP1	SDH_BA + 0x834	R	SD Host Receiving Response Token Register 1	0x0000_0000
SDH_BLEN	SDH_BA + 0x838	R/W	SD Host Block Length Register	0x0000_01FF
SDH_TMOUT	SDH_BA + 0x83C	R/W	SD Host Response/Data-in Time-out Register	0x0000_0000
SDH_ECTL	SDH_BA + 0x840	R/W	SD Host Extend Control Register	0x0000_0003

25 加密加速器

25.1 概述

NUC980 的 Crypto（加密加速器），包括一個安全偽隨機數生成器（Pseudo Random Number Generator）， AES 加密加速器，SHA 和 HMAC 運算加速器，ECC 算法加速器，以及 RSA 算法加速器。

偽隨機數生成器支持 64 位，128 位，192 位和 256 位的隨機數生成。

AES 加速器完全符合 AES（高級加密標準）加密和解密算法。AES 加速器支持 ECB，CBC，CFB，OFB，CTR，CBC-CS1，CBC-CS2 和 CBC-CS3 等加密模式。

SHA / HMAC 加速器完全符合 SHA-160，SHA-224，SHA-256，SHA-384 和 SHA-512 和相應的HMAC 算法。

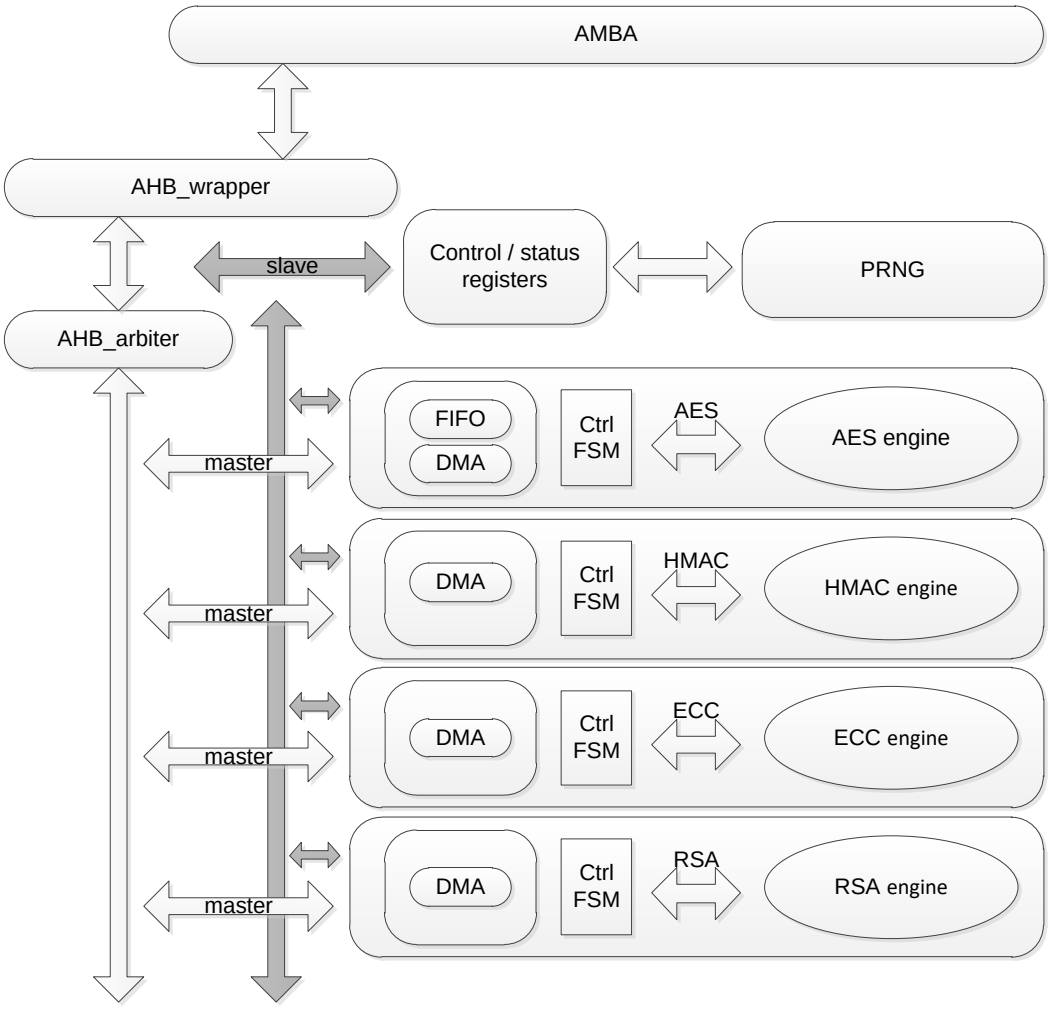
ECC 算法加速器支持 ECC 曲線二元場 GF(2^m) 乘法加法運算，支持質數場 GF(p) 的模數乘法、除法、加減運算。最高支援 571 bits ECC 運算，可支援 NIST P-192、P-224、P-256、P-384、P-521 曲線，B-163、B-233、B-283、B-409、B-571 曲線，以及 K-163、K-233、K-283、K-409、K-571 曲線。並支援 Koblitz secp192k1、secp224k1、secp256k1 曲線，以及 Brainpool P256r1、P384r1、P512r1 曲線。

RSA 算法加速器支持 RSA 模數乘法，最高支持達 2048 bits。

25.2 特性

- 偽隨機數生成器（PRNG）
  - 支持 64 位，128 位，192 位和 256 位的隨機數生成。
- AES
  - 支持 FIPS NIST 197 標準規範。
  - 支持 SP800-38A 標準和增編規範。
  - 支持 128，192 和 256 位密鑰。
  - 支持加密和解密。
  - 支持 ECB，CBC，CFB，OFB，CTR，CBC-CS1，CBC-CS2 和 CBC-CS3 模式。
  - 支持外部密鑰（來自 MTP）。
- SHA
  - 支持 FIPS NIST180，180-2 標準規範。
  - 支持 HMAC-SHA-160，HMAC-SHA-224，HMAC-SHA-256，HMAC-SHA-384 和 HMAC-SHA-512 。
- HMAC
  - 支持 FIPS NIST180，180-2 標準規範。
- ECC
  - 支持橢圓曲線二元場 GF(2^m) 及質數場 GF(p) 運算。
  - 支持 NIST P-192、P-224、P-256、P-384、P-521 曲線。
  - 支持 NIST B-163、B-233、B-283、B-409、B-571 曲線。
  - 支持 NIST K-163、K-233、K-283、K-409、K-571 曲線。
- RSA
  - 支持 RSA 加密及解密。
  - 支持高達 2048 bits RSA。

25.3 方塊圖



加密加速器包括一個安全偽隨機數發生器（PRNG）和支持 AES，SHA/HMAC，ECC，與 RSA 算法。加速器可以運用於不同的數據安全應用上，例如運用於需要加密保護與保證完整性的安全通信上。

PRNG 內核支持 64 位，128 位，192 位和 256 位的隨機數生成。

AES 加速器是完全兼容的實現了AES（高級加密標準）加密和解密算法。它支持 ECB，CBC，CFB，OFB，CTR，CBC-CS1，CBC-CS2 和 CBC-CS3 等模式。AES 加速器提供的 DMA 功能，減少了 CPU 的干預，並支持三種突發長度，16 字，8 字，和 4 字。

SHA / HMAC 加速器完全符合 SHA-160，SHA-224，SHA-256，SHA-384 和 SHA-512 和相應的HMAC 算法。SHA / HMAC加速器支持 DMA 功能，減少了CPU的干預。它支持三種突發長度，16 字，8 字，和 4 字。

ECC 加速器完全符合二元場 GF(2^m) 及質數場 GF(p) 演算法。 質數場GF(p) 演算法支持 NIST

P-192，P-224，P-256，P-384 及 P-521 曲線。二元場 GF(2^m) 演算法支持 NIST B-163，B-233，B-283，B-409，B-571 及 K-163，K-233，K-283，K-409，K-571 曲線。

RSA 加速器提供了 RSA 算法裡面指數乘法與模數除法的硬件加速。

25.3.1 數據訪問

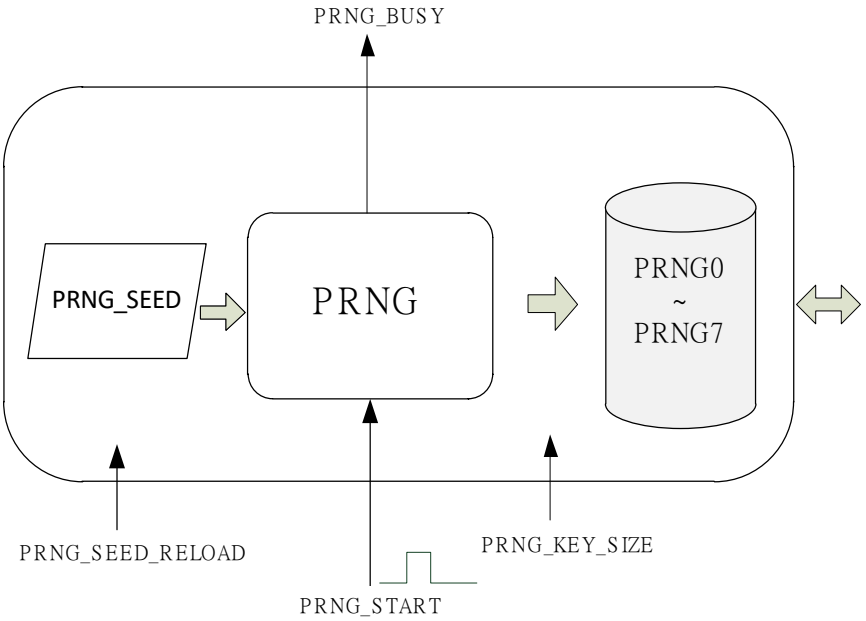
加密加速器提供了下列三種數據訪問的操作方式，提供效率的彈性的操作：

1. **DMA 模式：**軟件依照數據緩衝區地址，配置好 DMA 源地址寄存器，目的地址寄存器和字節計數寄存器，之後便完全由 DMA 邏輯單元負責對加密加速器的數據寫入及讀出。這種模式可卸去 CPU 的搬運數據的負荷。加密加速器內嵌四個硬件 DMA 通道給 AES 引擎，以及一個硬件 DMA 通道給 SHA / HMAC 引擎。
2. **DMA 級聯模式：**在加密數據量龐大，或加密數據流無法一次到位的情況下，或是有其他任務需要即時處理的情況下，可以採用 DMA 級聯（DMA Cascade）模式。在這種模式下，軟件可以分段處理數據加密，一開始的第一段加密指定為 DMA 模式，後續段的加密均指定為 DMA 級聯模式，並且必須使用同一個加密通道，便可以接續加密數據。每一段的加密均需要重新指定 DMA 源地址寄存器，目的地址寄存器，和字節計數寄存器。
3. **非 DMA 模式：**在輸入數據量極少的情況下，非 DMA 模式是另一種選擇。此模式可以減少操作加速器的時間，因為不需要填寫 DMA 相關的寄存器，也沒有引入 DMA 邏輯造成的延遲。在此模式下，加密數據是由軟件直接填寫寄存器給加密加速器。

25.4 功能描述

25.4.1 PRNG

PRNG支持64位，128位，192位和256位的隨機數生成，由KEYSZ（CRPT_PRNG_CTL[3：2]）指定。



PRNG 的操作程序如下：

1. 檢查 BUSY（CRPT_PRNG_CTL[8]），直到此位為 0。
2. 初始化 PRNG 參數。設置 KEYSZ（CRPT_PRNG_CTL[3：2]），並將隨機種子寫到 CRPT_PRNG_SEED 寄存器。需要注意的是 CRPT_PRNG_SEED 應該被初始化，因為芯片上電的時候它並不會被初始化。
3. 設置 PRNG 控制寄存器 CRPT_PRNG_CTL，同時寫入 STRAT（CRPT_PRNG_CTL[0]）觸發隨機數生成。
4. 檢查 BUSY（CRPT_PRNG_CTL[8]），直到它為 0，或等待中斷完成的 PRNG（必須啟用相應的中斷使能寄存器）。然後，軟件便可以從 CRPT_PRNG_KEY0～CRPT_PRNG_KEY7 讀取新生成的隨機數。
5. 重複步驟 3 ~ 4，可以不斷地獲得隨機數。

25.4.2 AES

進階加密標準（Advanced Encryption Standard），是美國聯邦政府採用的一種區塊加密標準。這個標準用來替代原先的DES，已經被多方分析且廣為全世界所使用。

用戶可以參考下面的步驟，了解如何使用 NUC980 AES 加速器。

25.4.2.1 AES DMA 模式的操作程序

1. 寫 1 到 AESIEN（CRPT_INTEN[0]），使能 AES 中斷。
2. 從 4 個 DMA 通道選擇一個閒置的通道。
3. 將 AES 密鑰寫入到 CRPT_AES0_KEY0 ～ CRPT_AES0_KEY7 寄存器。（其中，n 是所選擇的頻道數）。如果是選擇以 MTP key 做為密鑰，那麼就只需要設置 EXTKEY（CRPT_AES_CTL[4]）位，不必填寫 CRPT_AES0_KEY0 ~ CRPT_AES0_KEY7。
4. 如果有初始向量，將其寫到 CRPT_AES0_IV0～CRPT_AES0_IV3。
5. 將加密／解密來源數據緩存的實體地址，寫到 CRPT_AES0_SADDR 寄存器。並將輸出數據緩存的實體地址寫到 CRPT_AES0_DADDR 寄存器。將 DMA 傳輸字節數寫到 CRPT_AES0_CNT 寄存器。
6. 在 AES 控制寄存器 CRPT_AES_CTL 選擇通道，加密/解密，加密模式，DMA 模式，密鑰大小，和 DMA 的輸入/輸出交換等設定。
7. 將欲進行加密/解密的數據寫到 DMA 來源數據緩存區。
8. 寫 1 到 START（CRPT_AES_CTL[0]）開始 AES 加密/解密。
9. 等待 AES 中斷標誌 AESIF（CRPT_INTSTS[0]）被置位。
10. 從 DMA 輸出數據緩存區取得加密/解密後的數據。
11. 如果是採用 DMA 級聯（DMA Cascade）模式，則重複步驟 5 ~ 10 直到所有的數據處理完畢。

25.4.2.2 AES 非DMA 模式的操作程序

1. 寫 1 到 AESIEN（CRPT_INTEN[0]），使能 AES 中斷。
2. 從 AES 的 4 個 DMA 通道選擇一個閒置的通道。
3. 將 AES 密鑰寫入到 CRPT_AES0_KEY0 ～ CRPT_AES0_KEY7 寄存器。（其中，n 是所選擇的頻道數）。如果是選擇以 MTP key 做為密鑰，那麼就只需要設置 EXTKEY（CRPT_AES_CTL[4]）位，不必填寫 CRPT_AES0_KEY0 ~ CRPT_AES0_KEY7。
4. 如果有初始向量，將其寫到 CRPT_AES0_IV0～CRPT_AES0_IV3。
5. 在 AES 控制寄存器 CRPT_AES_CTL 選擇通道，加密/解密，加密模式，DMA 模式，密鑰大小，和 DMA 的輸入/輸出交換等設定。
6. 寫 1 到 START（CRPT_AES_CTL[0]）開始 AES 加密/解密。
7. 輪詢 INBUFFULL（CRPT_AES_STS[9]）位，如果此位為 0 ，便將 32 bits 加密/解密來源數據寫入到 CRPT_AES_DATIN 寄存器。
8. 輪詢 OUTBUFEMPTY（CRPT_AES_STS[16]）位，如果此位為 0，便從 CRPT_AES_DATOUT 讀取 32 bits 加密/解密輸出數據。

9. 重複步驟 7~8，直到從 CRPT_AES_DATOUT 寄存器讀取達 128 bits 的數據（16 字節）為止。
10. 對 DMALAST（CRPT_AES_CTL[5]）位寫 1，表示完成一個 AES 區段加密/解密。
11. 重複步驟 7~10，直到全部的加密/解密數據處理完成。

25.4.3 SHA

安全散列算法（Secure Hash Algorithm），是一系列由美國國家標準與技術研究院（NIST）為美國聯邦信息處理標準（FIPS）公佈的加密散列函數。

用戶可以參考下面的步驟，了解如何使用 NUC980 SHA 加速器。

25.4.3.1 SHA DMA 模式的操作程序

- 寫 1 到 HMACIEN（CRPT_INTEN[24]），使能 SHA/HMAC 中斷。
- 在 SHA/HMAC 控制寄存器 CRPT_HMAC_CTL，清除 HMACEN 位，選擇 SHA 演算模式，DMA 模式，和 DMA 的輸入/輸出交換等設定。
- 將 SHA 來源數據緩存的實體地址，寫到 CRPT_HMAC_SADDR 寄存器。將 DMA 傳輸字節數寫到 CRPT_HMAC_DMACNT 寄存器。
- 將欲進行 SHA 運算的數據寫到 DMA 來源數據緩存區。
- 寫 1 到 START（CRPT_HMAC_CTL[0]）開始 SHA 運算。
- 等待 SHA 中斷標誌 HMACIF（CRPT_INTSTS[24]）被設置。
- 直接從 CRPT_HMAC_DGST0～CRPT_HMAC_DGST7 寄存器讀取 SHA 運算結果。

25.4.3.2 SHA 非 DMA 模式的操作程序

1. 寫 1 到 HMACIEN（CRPT_INTEN[24]），使能 SHA/HMAC 中斷。
2. 在 SHA/HMAC 控制寄存器 CRPT_HMAC_CTL，清除 HMACEN 位，選擇 SHA 演算模式，DMA 模式，和 DMA 的輸入/輸出交換等設定。
3. 將 SHA 來源數據緩存的實體地址，寫到 CRPT_HMAC_SADDR 寄存器。將 DMA 傳輸字節數寫到 CRPT_HMAC_DMACNT 寄存器。
4. 寫 1 到 START（CRPT_HMAC_CTL[0]）開始 SHA 運算。
5. 等待 SHA 數據輸入請求 DATINREQ（CRPT_HMAC_STS[16]）位被設為 1。
6. 將 32 bits 數據寫到 CRPT_HMAC_DATIN。如果是最後一筆 32 bits 數據，則在此筆數據寫到 CRPT_HMAC_DATIN 之前，必須先將 DMALAST（CRPT_HMAC_CTL[5]）位寫 1。
7. 重複步驟 5 ~ 6，直到所有數據都被寫到 SHA 加速器。
8. 等待 BUSY（CRPT_HMAC_STS[0]）位被清除為 0，表示整個 SHA 運算完成。

9. 直接從 CRPT_HMAC_DGST0～CRPT_HMAC_DGST7 寄存器讀取 SHA 運算結果。

25.4.4 ECC

橢圓曲線密碼學（Elliptic Curve Cryptography），為知名的公開密鑰加密演算法，利用了橢圓曲線在有限域上的代數循環群特性，從而建立起的密碼系統。NUC980 ECC 支持質數場 y2≡x3+A*x+B (mod N) 橢圓曲線，以及二元場的 y2+x*y≡x3+A*x2+B (mod N) 橢圓曲線。

NIST 發佈了幾個公開的橢圓曲線，P-192、P-224、P-256、P-384、P-521、B-163、B-233、B-283、B-409、B571、K-163、K-233、K-283、K-409、及 K571，NUC980 ECC 可通過 NIST 官網發佈的相關測試範例。使用 NU980 ECC 硬件加速，必須將選定橢圓曲線的參考座標點及參數填入到對應的暫存器，然後再啟動硬件運算。

用戶可以參考下面的步驟，了解如何使用 NUC980 ECC 加速器。

25.4.4.1 使用質數場橢圓曲線

使用質數場橢圓曲線，必須先將選定曲線的基底座標點及各項參數填入 NUC980 ECC 對應的暫存器內：

- Gx： CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17
- Gy： CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17
- p： CRPT_ECC_N_00 ~ CRPT_ECC_N_17
- p-3： CRPT_ECC_A_00 ~ CRPT_ECC_A_17
- b： CRPT_ECC_B_00 ~ CRPT_ECC_B_17
- key length： CURVEM(CRPT_ECC_CTL[31:22])

25.4.4.2 使用二元場橢圓曲線

使用二元場橢圓曲線，必須先將選定曲線的基底座標點及各項參數填入 NUC980 ECC 對應的暫存器內：

- Gx： CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17
- Gy： CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17
- P(t)：CRPT_ECC_N_00 ~ CRPT_ECC_N_17
- a： CRPT_ECC_A_00 ~ CRPT_ECC_A_17
- b： CRPT_ECC_B_00 ~ CRPT_ECC_B_17
- key length： CURVEM(CRPT_ECC_CTL[31:22])

25.4.4.3 點乘法運算

使用 NUC980 ECC 進行橢圓曲線點乘法，操作流程如下：

1. 將選定曲線的各項橢圓曲線參數寫入到對應的暫存器。
2. 將被乘點的 x、y 座標分別寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
3. 將乘數寫入暫存器 CRPT_ECC_K_00 ~ CRPT_ECC_K_17。
4. 將 MODOP(CRPT_ECC_CTL[10:9]) 寫 0，將 STRAT(CRPT_ECC_CTL[0]) 寫 1 啟動 NUC980 ECC 硬件加速器，然後等待 ECC 中斷通知。
5. 從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17 讀取運算結果。

25.4.4.4 產生公鑰

使用 NUC980 ECC 加速產生公鑰之運算，操作流程如下：

1. 將選定曲線的各項橢圓曲線參數寫入到對應的暫存器。
2. 將私鑰寫入暫存器 CRPT_ECC_K_00 ~ CRPT_ECC_K_17。
3. 將 MODOP(CRPT_ECC_CTL[10:9]) 寫 0，將 STRAT(CRPT_ECC_CTL[0]) 寫 1 啟動 NUC980 ECC 硬件加速器，然後等待 ECC 中斷通知。
4. 從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17 讀取公鑰。

25.4.4.5 實現 ECC ECDH

ECDH(Diffie–Hellman key exchange) 是一種被廣泛使用的密鑰交換協定，可以讓通信雙方在不安全的通信通道下建立一個共有的金鑰，此金鑰稱之為 Secret Z。 做為 ECDH 協定一方，使用 NUC980 ECC 加速產生 Secret Z 之運算，操作流程如下：

1. 將選定曲線的各項橢圓曲線參數寫入到對應的暫存器。
2. 將自選的私鑰寫入暫存器 CRPT_ECC_K_00 ~ CRPT_ECC_K_17。
3. 將通信對方傳遞過來的公鑰，寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
4. 將 MODOP(CRPT_ECC_CTL[10:9]) 寫 0，將 STRAT(CRPT_ECC_CTL[0]) 寫 1 啟動 NUC980 ECC 硬件加速器，然後等待 ECC 中斷通知。
5. 從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 讀取金鑰 Secret Z。

25.4.4.6 實現 ECC ECDSA

ECDSA (Elliptic Curve Digital Signature Algorithm) 是採用橢圓曲線的數位簽章演算法。使用 NUC980 ECC 加速產生數位簽章之運算，操作流程如下：

1. 將選定曲線的各項橢圓曲線參數寫入到對應的暫存器。
2. 將隨機整數 k 寫入暫存器 CRPT_ECC_K_00 ~ CRPT_ECC_K_17。
3. 執行 ECC 點乘法，也就是將 k 乘以曲線基底座標點。產生的座標點 x 直接保留於暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17，而 y 則寫為 0（將 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17 全部清除為 0）。
4. 將選定曲線的 n 寫入暫存器 CRPT_ECC_N_00 ~ CRPT_ECC_N_17。
5. 執行 ECC 點乘加運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 讀取運算結果 r 。
6. 將選定曲線的 n 寫入暫存器 CRPT_ECC_N_00 ~ CRPT_ECC_N_17。
7. 將座標點 (0,1) 寫到暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
8. 進行 NUC980 ECC 模數除法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 得到運算結果 K-1。
9. 將前面運算結果 r 寫入到暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17，並將密鑰 d 寫入到暫存器 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
10. 進行 NUC980 ECC 模數乘法運算。
11. 計算欲傳遞訊息的 hash 值，通信雙方必須先確定採用何種 hash 算法（例如 SHA-1），計算得到 e 。將 e 寫到暫存器 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
12. 進行 NUC980 ECC 模數加法運算。
13. 將 K-1 寫到暫存器 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
14. 進行 NUC980 ECC 模數乘法運算。
15. 從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 讀取運算結果 s。
16. (r, s) 便是數位簽章。

使用 NUC980 ECC 加速驗證數位簽章之運算，操作流程如下：

1. 將選定曲線的各項橢圓曲線參數寫入到對應的暫存器。
2. 將選定曲線的 n 寫入暫存器 CRPT_ECC_N_00 ~ CRPT_ECC_N_17。
3. 將簽章 s 寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17。將 0x1 寫到 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
4. 進行 NUC980 ECC 模數除法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 得到運算結果 w。

5. 將 hash 值 e 寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17。將 w 寫到 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
6. 執行 ECC 模數乘法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 讀取運算結果 u1。
7. 將 r 寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17。將 w 寫到 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
8. 執行 ECC 模數乘法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 讀取運算結果 u2。
9. 重新將選定曲線的各項橢圓曲線參數寫入到對應的暫存器。
10. 將 u1 寫入暫存器 CRPT_ECC_K_00 ~ CRPT_ECC_K_17。
11. 執行 ECC 點乘法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17 讀取運算結果 u1 * G。
12. 將公鑰寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
13. 將 u2 寫入暫存器 CRPT_ECC_K_00 ~ CRPT_ECC_K_17。
14. 執行 ECC 點乘法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17 讀取運算結果 u2 * Q。
15. 將 u2 * Q 寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
16. 將 u1 * G 寫入暫存器 CRPT_ECC_X2_00 ~ CRPT_ECC_X2_17 及 CRPT_ECC_Y2_00 ~ CRPT_ECC_Y2_17。
17. 執行 ECC 點加法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 及 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17 讀取運算結果 (x', y')。
18. 將 x' 寫入暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17，將 0x0 寫入暫存器 CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17。
19. 執行 ECC 模數加法運算。從暫存器 CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 讀取運算結果 X1。
20. 如果 X1 等於 r ，則表示簽章驗證成功。

25.4.5 RSA

RSA 是一種被廣泛使用的公開密鑰加密演算法。其安全性是基於，對極大整數做因數分解是極其困難的。RSA 裡面最花費時間的運算是模指數運算(modulus exponentiation operation)，NUC980 RSA 加速器提供了對此運算的硬件加速。

25.4.5.1 RSA 公鑰與私鑰

RSA 產生公鑰及私鑰的流程如下：

1. 隨意選擇兩個極大的質數 p 和 q，計算 p * q 得到 N。
2. 計算 (p – 1) * (q – 1) 得到 r。
3. 選擇一個小於 r 的整數 E，E 必須與 r 互質。
4. 計算求得 E 關於 r 的模反元素 d。
5. p 和 q 不再需要，為安全性，予以銷毀。

至此，(N, E) 就是 RSA 的公鑰，而 (N, d) 則是 RSA 的私鑰。

25.4.5.2 Montgomery domain 常數

進行訊息加密，首先需要 NUC980 RSA 加速器提供的運算式為 M^E % N 或 M^d % N，其中 N, E, d 在 RSA 產生公鑰及私鑰程序中獲得，而 M 則是要加密或解密的訊息。

使用 NUC980 RSA 除了需要給予 M, E(d), N 之外，還必須給予 Montgomery domain 的常數值，Montgomery domain 的運算式為 $C = 2^{(key\_length+2)*2}$ % N ，軟件運算得到 C 值之後，填入 NUC980 RSA 暫存器，才能正確地進行 RSA 運算。

這個 C 值的計算比較花費軟件時間，然而只需計算一次得到常數值，爾後便不需要再次運算，除非更換了 N 值。

25.4.5.3 RSA 加密

使用 RSA 公鑰加密的流程如下：

1. 將 N 寫入暫存器 CRPT_RSA_N_00 ~ CRPT_RSA_N_63。
2. 計算 C 值，將 C 寫入暫存器 CRPT_RSA_C_00 ~ CRPT_RSA_C_63。
3. 將 E 寫入暫存器 CRPT_RSA_E_00 ~ CRPT_RSA_E_63。
4. 將欲加密的訊息寫入暫存器 CRPT_RSA_M_00 ~ CRPT_RSA_M_63。
5. 啟動 NUC980 RSA 模指數運算。
6. 從暫存器 CRPT_RSA_M_00 ~ CRPT_RSA_M_63 取得加密後的訊息。

25.4.5.4 RSA 解密

使用 RSA 私鑰解密的流程如下：

1. 將 N 寫入暫存器 CRPT_RSA_N_00 ~ CRPT_RSA_N_63。
2. 計算 C 值，將 C 寫入暫存器 CRPT_RSA_C_00 ~ CRPT_RSA_C_63。
3. 將 d 寫入暫存器 CRPT_RSA_E_00 ~ CRPT_RSA_E_63。
4. 將欲解密的訊息寫入暫存器 CRPT_RSA_M_00 ~ CRPT_RSA_M_63。
5. 啟動 NUC980 RSA 模指數運算。
6. 從暫存器 CRPT_RSA_M_00 ~ CRPT_RSA_M_63 取得解密後的訊息。

25.5 寄存器

Register	Offset	R/W	Description	Reset Value
CRYPTO Base Address: CRYPTO_BA = 0xB001_C000				
CRYPTO_INTEN	CRYPTO_BA+0x000	R/W	Crypto Interrupt Enable Control Register	0x0000_0000
CRYPTO_INTSTS	CRYPTO_BA+0x004	R/W	Crypto Interrupt Flag	0x0000_0000
CRYPTO_PRNG_CTL	CRYPTO_BA+0x008	R/W	PRNG Control Register	0x0000_0000
CRYPTO_PRNG_SEED	CRYPTO_BA+0x00C	W	Seed for PRNG	Undefined
CRYPTO_PRNG_KEY0	CRYPTO_BA+0x010	R	PRNG Generated Key0	Undefined
CRYPTO_PRNG_KEY1	CRYPTO_BA+0x014	R	PRNG Generated Key1	Undefined
CRYPTO_PRNG_KEY2	CRYPTO_BA+0x018	R	PRNG Generated Key2	Undefined
CRYPTO_PRNG_KEY3	CRYPTO_BA+0x01C	R	PRNG Generated Key3	Undefined
CRYPTO_PRNG_KEY4	CRYPTO_BA+0x020	R	PRNG Generated Key4	Undefined
CRYPTO_PRNG_KEY5	CRYPTO_BA+0x024	R	PRNG Generated Key5	Undefined
CRYPTO_PRNG_KEY6	CRYPTO_BA+0x028	R	PRNG Generated Key6	Undefined
CRYPTO_PRNG_KEY7	CRYPTO_BA+0x02C	R	PRNG Generated Key7	Undefined
CRYPTO_AES_FDBCK0	CRYPTO_BA+0x050	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000
CRYPTO_AES_FDBCK1	CRYPTO_BA+0x054	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000

CRYPTO_AES_FDBCK2	CRYPTO_BA+0x058	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000
CRYPTO_AES_FDBCK3	CRYPTO_BA+0x05C	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000
CRYPTO_AES_CTL	CRYPTO_BA+0x100	R/W	AES Control Register	0x0000_0000
CRYPTO_AES_STS	CRYPTO_BA+0x104	R	AES Engine Flag	0x0001_0100
CRYPTO_AES_DATIN	CRYPTO_BA+0x108	R/W	AES Engine Data Input Port Register	0x0000_0000
CRYPTO_AES_DATOUT	CRYPTO_BA+0x10C	R	AES Engine Data Output Port Register	0x0000_0000
CRYPTO_AES0_KEY0	CRYPTO_BA+0x110	R/W	AES Key Word 0 Register for Channel 0	0x0000_0000
CRYPTO_AES0_KEY1	CRYPTO_BA+0x114	R/W	AES Key Word 1 Register for Channel 0	0x0000_0000
CRYPTO_AES0_KEY2	CRYPTO_BA+0x118	R/W	AES Key Word 2 Register for Channel 0	0x0000_0000
CRYPTO_AES0_KEY3	CRYPTO_BA+0x11C	R/W	AES Key Word 3 Register for Channel 0	0x0000_0000
CRYPTO_AES0_KEY4	CRYPTO_BA+0x120	R/W	AES Key Word 4 Register for Channel 0	0x0000_0000
CRYPTO_AES0_KEY5	CRYPTO_BA+0x124	R/W	AES Key Word 5 Register for Channel 0	0x0000_0000
CRYPTO_AES0_KEY6	CRYPTO_BA+0x128	R/W	AES Key Word 6 Register for Channel 0	0x0000_0000
CRYPTO_AES0_KEY7	CRYPTO_BA+0x12C	R/W	AES Key Word 7 Register for Channel 0	0x0000_0000
CRYPTO_AES0_IV0	CRYPTO_BA+0x130	R/W	AES Initial Vector Word 0 Register for Channel 0	0x0000_0000
CRYPTO_AES0_IV1	CRYPTO_BA+0x134	R/W	AES Initial Vector Word 1 Register for Channel 0	0x0000_0000
CRYPTO_AES0_IV2	CRYPTO_BA+0x138	R/W	AES Initial Vector Word 2 Register for Channel 0	0x0000_0000
CRYPTO_AES0_IV3	CRYPTO_BA+0x13C	R/W	AES Initial Vector Word 3 Register for Channel 0	0x0000_0000
CRYPTO_AES0_SADDR	CRYPTO_BA+0x140	R/W	AES DMA Source Address Register for Channel 0	0x0000_0000
CRYPTO_AES0_DADDR	CRYPTO_BA+0x144	R/W	AES DMA Destination Address Register for Channel 0	0x0000_0000
CRYPTO_AES0_CNT	CRYPTO_BA+0x148	R/W	AES Byte Count Register for Channel 0	0x0000_0000
CRYPTO_AES1_KEY0	CRYPTO_BA+0x14C	R/W	AES Key Word 0 Register for Channel 1	0x0000_0000
CRYPTO_AES1_KEY1	CRYPTO_BA+0x150	R/W	AES Key Word 1 Register for Channel 1	0x0000_0000
CRYPTO_AES1_KEY2	CRYPTO_BA+0x154	R/W	AES Key Word 2 Register for Channel 1	0x0000_0000
CRYPTO_AES1_KEY3	CRYPTO_BA+0x158	R/W	AES Key Word 3 Register for Channel 1	0x0000_0000

CRYPTO_AES1_KEY4	CRYPTO_BA+0x15C	R/W	AES Key Word 4 Register for Channel 1	0x0000_0000
CRYPTO_AES1_KEY5	CRYPTO_BA+0x160	R/W	AES Key Word 5 Register for Channel 1	0x0000_0000
CRYPTO_AES1_KEY6	CRYPTO_BA+0x164	R/W	AES Key Word 6 Register for Channel 1	0x0000_0000
CRYPTO_AES1_KEY7	CRYPTO_BA+0x168	R/W	AES Key Word 7 Register for Channel 1	0x0000_0000
CRYPTO_AES1_IV0	CRYPTO_BA+0x16C	R/W	AES Initial Vector Word 0 Register for Channel 1	0x0000_0000
CRYPTO_AES1_IV1	CRYPTO_BA+0x170	R/W	AES Initial Vector Word 1 Register for Channel 1	0x0000_0000
CRYPTO_AES1_IV2	CRYPTO_BA+0x174	R/W	AES Initial Vector Word 2 Register for Channel 1	0x0000_0000
CRYPTO_AES1_IV3	CRYPTO_BA+0x178	R/W	AES Initial Vector Word 3 Register for Channel 1	0x0000_0000
CRYPTO_AES1_SADDR	CRYPTO_BA+0x17C	R/W	AES DMA Source Address Register for Channel 1	0x0000_0000
CRYPTO_AES1_DADDR	CRYPTO_BA+0x180	R/W	AES DMA Destination Address Register for Channel 1	0x0000_0000
CRYPTO_AES1_CNT	CRYPTO_BA+0x184	R/W	AES Byte Count Register for Channel 1	0x0000_0000
CRYPTO_AES2_KEY0	CRYPTO_BA+0x188	R/W	AES Key Word 0 Register for Channel 2	0x0000_0000
CRYPTO_AES2_KEY1	CRYPTO_BA+0x18C	R/W	AES Key Word 1 Register for Channel 2	0x0000_0000
CRYPTO_AES2_KEY2	CRYPTO_BA+0x190	R/W	AES Key Word 2 Register for Channel 2	0x0000_0000
CRYPTO_AES2_KEY3	CRYPTO_BA+0x194	R/W	AES Key Word 3 Register for Channel 2	0x0000_0000
CRYPTO_AES2_KEY4	CRYPTO_BA+0x198	R/W	AES Key Word 4 Register for Channel 2	0x0000_0000
CRYPTO_AES2_KEY5	CRYPTO_BA+0x19C	R/W	AES Key Word 5 Register for Channel 2	0x0000_0000
CRYPTO_AES2_KEY6	CRYPTO_BA+0x1A0	R/W	AES Key Word 6 Register for Channel 2	0x0000_0000
CRYPTO_AES2_KEY7	CRYPTO_BA+0x1A4	R/W	AES Key Word 7 Register for Channel 2	0x0000_0000
CRYPTO_AES2_IV0	CRYPTO_BA+0x1A8	R/W	AES Initial Vector Word 0 Register for Channel 2	0x0000_0000
CRYPTO_AES2_IV1	CRYPTO_BA+0x1AC	R/W	AES Initial Vector Word 1 Register for Channel 2	0x0000_0000
CRYPTO_AES2_IV2	CRYPTO_BA+0x1B0	R/W	AES Initial Vector Word 2 Register for Channel 2	0x0000_0000
CRYPTO_AES2_IV3	CRYPTO_BA+0x1B4	R/W	AES Initial Vector Word 3 Register for Channel 2	0x0000_0000
CRYPTO_AES2_SADDR	CRYPTO_BA+0x1B8	R/W	AES DMA Source Address Register for Channel 2	0x0000_0000
CRYPTO_AES2_DADDR	CRYPTO_BA+0x1BC	R/W	AES DMA Destination Address Register for Channel 2	0x0000_0000
CRYPTO_AES2_CNT	CRYPTO_BA+0x1C0	R/W	AES Byte Count Register for Channel 2	0x0000_0000

CRYPTO_AES3_KEY0	CRYPTO_BA+0x1C4	R/W	AES Key Word 0 Register for Channel 3	0x0000_0000
CRYPTO_AES3_KEY1	CRYPTO_BA+0x1C8	R/W	AES Key Word 1 Register for Channel 3	0x0000_0000
CRYPTO_AES3_KEY2	CRYPTO_BA+0x1CC	R/W	AES Key Word 2 Register for Channel 3	0x0000_0000
CRYPTO_AES3_KEY3	CRYPTO_BA+0x1D0	R/W	AES Key Word 3 Register for Channel 3	0x0000_0000
CRYPTO_AES3_KEY4	CRYPTO_BA+0x1D4	R/W	AES Key Word 4 Register for Channel 3	0x0000_0000
CRYPTO_AES3_KEY5	CRYPTO_BA+0x1D8	R/W	AES Key Word 5 Register for Channel 3	0x0000_0000
CRYPTO_AES3_KEY6	CRYPTO_BA+0x1DC	R/W	AES Key Word 6 Register for Channel 3	0x0000_0000
CRYPTO_AES3_KEY7	CRYPTO_BA+0x1E0	R/W	AES Key Word 7 Register for Channel 3	0x0000_0000
CRYPTO_AES3_IV0	CRYPTO_BA+0x1E4	R/W	AES Initial Vector Word 0 Register for Channel 3	0x0000_0000
CRYPTO_AES3_IV1	CRYPTO_BA+0x1E8	R/W	AES Initial Vector Word 1 Register for Channel 3	0x0000_0000
CRYPTO_AES3_IV2	CRYPTO_BA+0x1EC	R/W	AES Initial Vector Word 2 Register for Channel 3	0x0000_0000
CRYPTO_AES3_IV3	CRYPTO_BA+0x1F0	R/W	AES Initial Vector Word 3 Register for Channel 3	0x0000_0000
CRYPTO_AES3_SADDR	CRYPTO_BA+0x1F4	R/W	AES DMA Source Address Register for Channel 3	0x0000_0000
CRYPTO_AES3_DADDR	CRYPTO_BA+0x1F8	R/W	AES DMA Destination Address Register for Channel 3	0x0000_0000
CRYPTO_AES3_CNT	CRYPTO_BA+0x1FC	R/W	AES Byte Count Register for Channel 3	0x0000_0000
CRYPTO_HMAC_CTL	CRYPTO_BA+0x300	R/W	SHA/HMAC Control Register	0x0000_0000
CRYPTO_HMAC_STS	CRYPTO_BA+0x304	R	SHA/HMAC Status Flag	0x0000_0000
CRYPTO_HMAC_DGST0	CRYPTO_BA+0x308	R	SHA/HMAC Digest Message 0	0x0000_0000
CRYPTO_HMAC_DGST1	CRYPTO_BA+0x30C	R	SHA/HMAC Digest Message 1	0x0000_0000
CRYPTO_HMAC_DGST2	CRYPTO_BA+0x310	R	SHA/HMAC Digest Message 2	0x0000_0000
CRYPTO_HMAC_DGST3	CRYPTO_BA+0x314	R	SHA/HMAC Digest Message 3	0x0000_0000
CRYPTO_HMAC_DGST4	CRYPTO_BA+0x318	R	SHA/HMAC Digest Message 4	0x0000_0000
CRYPTO_HMAC_DGST5	CRYPTO_BA+0x31C	R	SHA/HMAC Digest Message 5	0x0000_0000
CRYPTO_HMAC_DGST6	CRYPTO_BA+0x320	R	SHA/HMAC Digest Message 6	0x0000_0000
CRYPTO_HMAC_DGST7	CRYPTO_BA+0x324	R	SHA/HMAC Digest Message 7	0x0000_0000
CRYPTO_HMAC_DGST8	CRYPTO_BA+0x328	R	SHA/HMAC Digest Message 8	0x0000_0000

CRYPTO_HMAC_DGST9	CRYPTO_BA+0x32C	R	SHA/HMAC Digest Message 9	0x0000_0000
CRYPTO_HMAC_DGST10	CRYPTO_BA+0x330	R	SHA/HMAC Digest Message 10	0x0000_0000
CRYPTO_HMAC_DGST11	CRYPTO_BA+0x334	R	SHA/HMAC Digest Message 11	0x0000_0000
CRYPTO_HMAC_DGST12	CRYPTO_BA+0x338	R	SHA/HMAC Digest Message 12	0x0000_0000
CRYPTO_HMAC_DGST13	CRYPTO_BA+0x33C	R	SHA/HMAC Digest Message 13	0x0000_0000
CRYPTO_HMAC_DGST14	CRYPTO_BA+0x340	R	SHA/HMAC Digest Message 14	0x0000_0000
CRYPTO_HMAC_DGST15	CRYPTO_BA+0x344	R	SHA/HMAC Digest Message 15	0x0000_0000
CRYPTO_HMAC_KEYCNT	CRYPTO_BA+0x348	R/W	SHA/HMAC Key Byte Count Register	0x0000_0000
CRYPTO_HMAC_SADDR	CRYPTO_BA+0x34C	R/W	SHA/HMAC DMA Source Address Register	0x0000_0000
CRYPTO_HMAC_DMACNT	CRYPTO_BA+0x350	R/W	SHA/HMAC Byte Count Register	0x0000_0000
CRYPTO_HMAC_DATIN	CRYPTO_BA+0x354	R/W	SHA/HMAC Engine Non-DMA Mode Data Input Port Register	0x0000_0000
CRPT_ECC_CTL	CRYPTO_BA+0x800	R/W	ECC Control Register	0x0000_0000
CRPT_ECC_STS	CRYPTO_BA+0x804	R	ECC Status Register	0x0000_0000
CRPT_ECC_X1_00	CRYPTO_BA+0x808	R/W	ECC The X-coordinate word0 of the first point	0x0000_0000
CRPT_ECC_X1_01	CRYPTO_BA+0x80C	R/W	ECC The X-coordinate word1 of the first point	0x0000_0000
CRPT_ECC_X1_02	CRYPTO_BA+0x810	R/W	ECC The X-coordinate word2 of the first point	0x0000_0000
CRPT_ECC_X1_03	CRYPTO_BA+0x814	R/W	ECC The X-coordinate word3 of the first point	0x0000_0000
CRPT_ECC_X1_04	CRYPTO_BA+0x818	R/W	ECC The X-coordinate word4 of the first point	0x0000_0000
CRPT_ECC_X1_05	CRYPTO_BA+0x81C	R/W	ECC The X-coordinate word5 of the first point	0x0000_0000
CRPT_ECC_X1_06	CRYPTO_BA+0x820	R/W	ECC The X-coordinate word6 of the first point	0x0000_0000
CRPT_ECC_X1_07	CRYPTO_BA+0x824	R/W	ECC The X-coordinate word7 of the first point	0x0000_0000
CRPT_ECC_X1_08	CRYPTO_BA+0x828	R/W	ECC The X-coordinate word8 of the first point	0x0000_0000
CRPT_ECC_X1_09	CRYPTO_BA+0x82C	R/W	ECC The X-coordinate word9 of the first point	0x0000_0000
CRPT_ECC_X1_10	CRYPTO_BA+0x830	R/W	ECC The X-coordinate word10 of the first point	0x0000_0000
CRPT_ECC_X1_11	CRYPTO_BA+0x834	R/W	ECC The X-coordinate word11 of the first point	0x0000_0000
CRPT_ECC_X1_12	CRYPTO_BA+0x838	R/W	ECC The X-coordinate word12 of the first point	0x0000_0000

CRPT_ECC_X1_13	CRYPTO_BA+0x83C	R/W	ECC The X-coordinate word13 of the first point	0x0000_0000
CRPT_ECC_X1_14	CRYPTO_BA+0x840	R/W	ECC The X-coordinate word14 of the first point	0x0000_0000
CRPT_ECC_X1_15	CRYPTO_BA+0x844	R/W	ECC The X-coordinate word15 of the first point	0x0000_0000
CRPT_ECC_X1_16	CRYPTO_BA+0x848	R/W	ECC The X-coordinate word16 of the first point	0x0000_0000
CRPT_ECC_X1_17	CRYPTO_BA+0x84C	R/W	ECC The X-coordinate word17 of the first point	0x0000_0000
CRPT_ECC_Y1_00	CRYPTO_BA+0x850	R/W	ECC The Y-coordinate word0 of the first point	0x0000_0000
CRPT_ECC_Y1_01	CRYPTO_BA+0x854	R/W	ECC The Y-coordinate word1 of the first point	0x0000_0000
CRPT_ECC_Y1_02	CRYPTO_BA+0x858	R/W	ECC The Y-coordinate word2 of the first point	0x0000_0000
CRPT_ECC_Y1_03	CRYPTO_BA+0x85C	R/W	ECC The Y-coordinate word3 of the first point	0x0000_0000
CRPT_ECC_Y1_04	CRYPTO_BA+0x860	R/W	ECC The Y-coordinate word4 of the first point	0x0000_0000
CRPT_ECC_Y1_05	CRYPTO_BA+0x864	R/W	ECC The Y-coordinate word5 of the first point	0x0000_0000
CRPT_ECC_Y1_06	CRYPTO_BA+0x868	R/W	ECC The Y-coordinate word6 of the first point	0x0000_0000
CRPT_ECC_Y1_07	CRYPTO_BA+0x86C	R/W	ECC The Y-coordinate word7 of the first point	0x0000_0000
CRPT_ECC_Y1_08	CRYPTO_BA+0x870	R/W	ECC The Y-coordinate word8 of the first point	0x0000_0000
CRPT_ECC_Y1_09	CRYPTO_BA+0x874	R/W	ECC The Y-coordinate word9 of the first point	0x0000_0000
CRPT_ECC_Y1_10	CRYPTO_BA+0x878	R/W	ECC The Y-coordinate word10 of the first point	0x0000_0000
CRPT_ECC_Y1_11	CRYPTO_BA+0x87C	R/W	ECC The Y-coordinate word11 of the first point	0x0000_0000
CRPT_ECC_Y1_12	CRYPTO_BA+0x880	R/W	ECC The Y-coordinate word12 of the first point	0x0000_0000
CRPT_ECC_Y1_13	CRYPTO_BA+0x884	R/W	ECC The Y-coordinate word13 of the first point	0x0000_0000
CRPT_ECC_Y1_14	CRYPTO_BA+0x888	R/W	ECC The Y-coordinate word14 of the first point	0x0000_0000
CRPT_ECC_Y1_15	CRYPTO_BA+0x88C	R/W	ECC The Y-coordinate word15 of the first point	0x0000_0000
CRPT_ECC_Y1_16	CRYPTO_BA+0x890	R/W	ECC The Y-coordinate word16 of the first point	0x0000_0000
CRPT_ECC_Y1_17	CRYPTO_BA+0x894	R/W	ECC The Y-coordinate word17 of the first point	0x0000_0000
CRPT_ECC_X2_00	CRYPTO_BA+0x898	R/W	ECC The X-coordinate word0 of the second point	0x0000_0000
CRPT_ECC_X2_01	CRYPTO_BA+0x89C	R/W	ECC The X-coordinate word1 of the second point	0x0000_0000
CRPT_ECC_X2_02	CRYPTO_BA+0x8A0	R/W	ECC The X-coordinate word2 of the second point	0x0000_0000
CRPT_ECC_X2_03	CRYPTO_BA+0x8A4	R/W	ECC The X-coordinate word3 of the second point	0x0000_0000

CRPT_ECC_X2_04	CRYPTO_BA+0x8A8	R/W	ECC The X-coordinate word4 of the second point	0x0000_0000
CRPT_ECC_X2_05	CRYPTO_BA+0x8AC	R/W	ECC The X-coordinate word5 of the second point	0x0000_0000
CRPT_ECC_X2_06	CRYPTO_BA+0x8B0	R/W	ECC The X-coordinate word6 of the second point	0x0000_0000
CRPT_ECC_X2_07	CRYPTO_BA+0x8B4	R/W	ECC The X-coordinate word7 of the second point	0x0000_0000
CRPT_ECC_X2_08	CRYPTO_BA+0x8B8	R/W	ECC The X-coordinate word8 of the second point	0x0000_0000
CRPT_ECC_X2_09	CRYPTO_BA+0x8BC	R/W	ECC The X-coordinate word9 of the second point	0x0000_0000
CRPT_ECC_X2_10	CRYPTO_BA+0x8C0	R/W	ECC The X-coordinate word10 of the second point	0x0000_0000
CRPT_ECC_X2_11	CRYPTO_BA+0x8C4	R/W	ECC The X-coordinate word11 of the second point	0x0000_0000
CRPT_ECC_X2_12	CRYPTO_BA+0x8C8	R/W	ECC The X-coordinate word12 of the second point	0x0000_0000
CRPT_ECC_X2_13	CRYPTO_BA+0x8CC	R/W	ECC The X-coordinate word13 of the second point	0x0000_0000
CRPT_ECC_X2_14	CRYPTO_BA+0x8D0	R/W	ECC The X-coordinate word14 of the second point	0x0000_0000
CRPT_ECC_X2_15	CRYPTO_BA+0x8D4	R/W	ECC The X-coordinate word15 of the second point	0x0000_0000
CRPT_ECC_X2_16	CRYPTO_BA+0x8D8	R/W	ECC The X-coordinate word16 of the second point	0x0000_0000
CRPT_ECC_X2_17	CRYPTO_BA+0x8DC	R/W	ECC The X-coordinate word17 of the second point	0x0000_0000
CRPT_ECC_Y2_00	CRYPTO_BA+0x8E0	R/W	ECC The Y-coordinate word0 of the second point	0x0000_0000
CRPT_ECC_Y2_01	CRYPTO_BA+0x8E4	R/W	ECC The Y-coordinate word1 of the second point	0x0000_0000
CRPT_ECC_Y2_02	CRYPTO_BA+0x8E8	R/W	ECC The Y-coordinate word2 of the second point	0x0000_0000
CRPT_ECC_Y2_03	CRYPTO_BA+0x8EC	R/W	ECC The Y-coordinate word3 of the second point	0x0000_0000
CRPT_ECC_Y2_04	CRYPTO_BA+0x8F0	R/W	ECC The Y-coordinate word4 of the second point	0x0000_0000
CRPT_ECC_Y2_05	CRYPTO_BA+0x8F4	R/W	ECC The Y-coordinate word5 of the second point	0x0000_0000
CRPT_ECC_Y2_06	CRYPTO_BA+0x8F8	R/W	ECC The Y-coordinate word6 of the second point	0x0000_0000
CRPT_ECC_Y2_07	CRYPTO_BA+0x8FC	R/W	ECC The Y-coordinate word7 of the second point	0x0000_0000
CRPT_ECC_Y2_08	CRYPTO_BA+0x900	R/W	ECC The Y-coordinate word8 of the second point	0x0000_0000

CRPT_ECC_Y2_09	CRYPTO_BA+0x904	R/W	ECC The Y-coordinate word9 of the second point	0x0000_0000
CRPT_ECC_Y2_10	CRYPTO_BA+0x908	R/W	ECC The Y-coordinate word10 of the second point	0x0000_0000
CRPT_ECC_Y2_11	CRYPTO_BA+0x90C	R/W	ECC The Y-coordinate word11 of the second point	0x0000_0000
CRPT_ECC_Y2_12	CRYPTO_BA+0x910	R/W	ECC The Y-coordinate word12 of the second point	0x0000_0000
CRPT_ECC_Y2_13	CRYPTO_BA+0x914	R/W	ECC The Y-coordinate word13 of the second point	0x0000_0000
CRPT_ECC_Y2_14	CRYPTO_BA+0x918	R/W	ECC The Y-coordinate word14 of the second point	0x0000_0000
CRPT_ECC_Y2_15	CRYPTO_BA+0x91C	R/W	ECC The Y-coordinate word15 of the second point	0x0000_0000
CRPT_ECC_Y2_16	CRYPTO_BA+0x920	R/W	ECC The Y-coordinate word16 of the second point	0x0000_0000
CRPT_ECC_Y2_17	CRYPTO_BA+0x924	R/W	ECC The Y-coordinate word17 of the second point	0x0000_0000
CRPT_ECC_A_00	CRYPTO_BA+0x928	R/W	ECC The parameter CURVEA word0 of elliptic curve	0x0000_0000
CRPT_ECC_A_01	CRYPTO_BA+0x92C	R/W	ECC The parameter CURVEA word1 of elliptic curve	0x0000_0000
CRPT_ECC_A_02	CRYPTO_BA+0x930	R/W	ECC The parameter CURVEA word2 of elliptic curve	0x0000_0000
CRPT_ECC_A_03	CRYPTO_BA+0x934	R/W	ECC The parameter CURVEA word3 of elliptic curve	0x0000_0000
CRPT_ECC_A_04	CRYPTO_BA+0x938	R/W	ECC The parameter CURVEA word4 of elliptic curve	0x0000_0000
CRPT_ECC_A_05	CRYPTO_BA+0x93C	R/W	ECC The parameter CURVEA word5 of elliptic curve	0x0000_0000
CRPT_ECC_A_06	CRYPTO_BA+0x940	R/W	ECC The parameter CURVEA word6 of elliptic curve	0x0000_0000
CRPT_ECC_A_07	CRYPTO_BA+0x944	R/W	ECC The parameter CURVEA word7 of elliptic curve	0x0000_0000
CRPT_ECC_A_08	CRYPTO_BA+0x948	R/W	ECC The parameter CURVEA word8 of elliptic curve	0x0000_0000

CRPT_ECC_A_09	CRYPTO_BA+0x94C	R/W	ECC The parameter CURVEA word9 of elliptic curve	0x0000_0000
CRPT_ECC_A_10	CRYPTO_BA+0x950	R/W	ECC The parameter CURVEA word10 of elliptic curve	0x0000_0000
CRPT_ECC_A_11	CRYPTO_BA+0x954	R/W	ECC The parameter CURVEA word11 of elliptic curve	0x0000_0000
CRPT_ECC_A_12	CRYPTO_BA+0x958	R/W	ECC The parameter CURVEA word12 of elliptic curve	0x0000_0000
CRPT_ECC_A_13	CRYPTO_BA+0x95C	R/W	ECC The parameter CURVEA word13 of elliptic curve	0x0000_0000
CRPT_ECC_A_14	CRYPTO_BA+0x960	R/W	ECC The parameter CURVEA word14 of elliptic curve	0x0000_0000
CRPT_ECC_A_15	CRYPTO_BA+0x964	R/W	ECC The parameter CURVEA word15 of elliptic curve	0x0000_0000
CRPT_ECC_A_16	CRYPTO_BA+0x968	R/W	ECC The parameter CURVEA word16 of elliptic curve	0x0000_0000
CRPT_ECC_A_17	CRYPTO_BA+0x96C	R/W	ECC The parameter CURVEA word17 of elliptic curve	0x0000_0000
CRPT_ECC_B_00	CRYPTO_BA+0x970	R/W	ECC The parameter CURVEB word0 of elliptic curve	0x0000_0000
CRPT_ECC_B_01	CRYPTO_BA+0x974	R/W	ECC The parameter CURVEB word1 of elliptic curve	0x0000_0000
CRPT_ECC_B_02	CRYPTO_BA+0x978	R/W	ECC The parameter CURVEB word2 of elliptic curve	0x0000_0000
CRPT_ECC_B_03	CRYPTO_BA+0x97C	R/W	ECC The parameter CURVEB word3 of elliptic curve	0x0000_0000
CRPT_ECC_B_04	CRYPTO_BA+0x980	R/W	ECC The parameter CURVEB word4 of elliptic curve	0x0000_0000
CRPT_ECC_B_05	CRYPTO_BA+0x984	R/W	ECC The parameter CURVEB word5 of elliptic curve	0x0000_0000
CRPT_ECC_B_06	CRYPTO_BA+0x988	R/W	ECC The parameter CURVEB word6 of elliptic curve	0x0000_0000
CRPT_ECC_B_07	CRYPTO_BA+0x98C	R/W	ECC The parameter CURVEB word7 of elliptic curve	0x0000_0000
CRPT_ECC_B_08	CRYPTO_BA+0x990	R/W	ECC The parameter CURVEB word8 of elliptic curve	0x0000_0000

CRPT_ECC_B_09	CRYPTO_BA+0x994	R/W	ECC The parameter CURVEB word9 of elliptic curve	0x0000_0000
CRPT_ECC_B_10	CRYPTO_BA+0x998	R/W	ECC The parameter CURVEB word10 of elliptic curve	0x0000_0000
CRPT_ECC_B_11	CRYPTO_BA+0x99C	R/W	ECC The parameter CURVEB word11 of elliptic curve	0x0000_0000
CRPT_ECC_B_12	CRYPTO_BA+0x9A0	R/W	ECC The parameter CURVEB word12 of elliptic curve	0x0000_0000
CRPT_ECC_B_13	CRYPTO_BA+0x9A4	R/W	ECC The parameter CURVEB word13 of elliptic curve	0x0000_0000
CRPT_ECC_B_14	CRYPTO_BA+0x9A8	R/W	ECC The parameter CURVEB word14 of elliptic curve	0x0000_0000
CRPT_ECC_B_15	CRYPTO_BA+0x9AC	R/W	ECC The parameter CURVEB word15 of elliptic curve	0x0000_0000
CRPT_ECC_B_16	CRYPTO_BA+0x9B0	R/W	ECC The parameter CURVEB word16 of elliptic curve	0x0000_0000
CRPT_ECC_B_17	CRYPTO_BA+0x9B4	R/W	ECC The parameter CURVEB word17 of elliptic curve	0x0000_0000
CRPT_ECC_N_00	CRYPTO_BA+0x9B8	R/W	ECC The parameter CURVEN word0 of elliptic curve	0x0000_0000
CRPT_ECC_N_01	CRYPTO_BA+0x9BC	R/W	ECC The parameter CURVEN word1 of elliptic curve	0x0000_0000
CRPT_ECC_N_02	CRYPTO_BA+0x9C0	R/W	ECC The parameter CURVEN word2 of elliptic curve	0x0000_0000
CRPT_ECC_N_03	CRYPTO_BA+0x9C4	R/W	ECC The parameter CURVEN word3 of elliptic curve	0x0000_0000
CRPT_ECC_N_04	CRYPTO_BA+0x9C8	R/W	ECC The parameter CURVEN word4 of elliptic curve	0x0000_0000
CRPT_ECC_N_05	CRYPTO_BA+0x9CC	R/W	ECC The parameter CURVEN word5 of elliptic curve	0x0000_0000
CRPT_ECC_N_06	CRYPTO_BA+0x9D0	R/W	ECC The parameter CURVEN word6 of elliptic curve	0x0000_0000
CRPT_ECC_N_07	CRYPTO_BA+0x9D4	R/W	ECC The parameter CURVEN word7 of elliptic curve	0x0000_0000
CRPT_ECC_N_08	CRYPTO_BA+0x9D8	R/W	ECC The parameter CURVEN word8 of elliptic curve	0x0000_0000

CRPT_ECC_N_09	CRYPTO_BA+0x9DC	R/W	ECC The parameter CURVEN word9 of elliptic curve	0x0000_0000
CRPT_ECC_N_10	CRYPTO_BA+0x9E0	R/W	ECC The parameter CURVEN word10 of elliptic curve	0x0000_0000
CRPT_ECC_N_11	CRYPTO_BA+0x9E4	R/W	ECC The parameter CURVEN word11 of elliptic curve	0x0000_0000
CRPT_ECC_N_12	CRYPTO_BA+0x9E8	R/W	ECC The parameter CURVEN word12 of elliptic curve	0x0000_0000
CRPT_ECC_N_13	CRYPTO_BA+0x9EC	R/W	ECC The parameter CURVEN word13 of elliptic curve	0x0000_0000
CRPT_ECC_N_14	CRYPTO_BA+0x9F0	R/W	ECC The parameter CURVEN word14 of elliptic curve	0x0000_0000
CRPT_ECC_N_15	CRYPTO_BA+0x9F4	R/W	ECC The parameter CURVEN word15 of elliptic curve	0x0000_0000
CRPT_ECC_N_16	CRYPTO_BA+0x9F8	R/W	ECC The parameter CURVEN word16 of elliptic curve	0x0000_0000
CRPT_ECC_N_17	CRYPTO_BA+0x9FC	R/W	ECC The parameter CURVEN word17 of elliptic curve	0x0000_0000
CRPT_ECC_K_00	CRYPTO_BA+0xA00	W	ECC The scalar SCALARK word0 of point multiplication	0x0000_0000
CRPT_ECC_K_01	CRYPTO_BA+0xA04	W	ECC The scalar SCALARK word1 of point multiplication	0x0000_0000
CRPT_ECC_K_02	CRYPTO_BA+0xA08	W	ECC The scalar SCALARK word2 of point multiplication	0x0000_0000
CRPT_ECC_K_03	CRYPTO_BA+0xA0C	W	ECC The scalar SCALARK word3 of point multiplication	0x0000_0000
CRPT_ECC_K_04	CRYPTO_BA+0xA10	W	ECC The scalar SCALARK word4 of point multiplication	0x0000_0000
CRPT_ECC_K_05	CRYPTO_BA+0xA14	W	ECC The scalar SCALARK word5 of point multiplication	0x0000_0000
CRPT_ECC_K_06	CRYPTO_BA+0xA18	W	ECC The scalar SCALARK word6 of point multiplication	0x0000_0000
CRPT_ECC_K_07	CRYPTO_BA+0xA1C	W	ECC The scalar SCALARK word7 of point multiplication	0x0000_0000
CRPT_ECC_K_08	CRYPTO_BA+0xA20	W	ECC The scalar SCALARK word8 of point multiplication	0x0000_0000

CRPT_ECC_K_09	CRYPTO_BA+0xA24	W	ECC The scalar SCALARK word9 of point multiplication	0x0000_0000
CRPT_ECC_K_10	CRYPTO_BA+0xA28	W	ECC The scalar SCALARK word10 of point multiplication	0x0000_0000
CRPT_ECC_K_11	CRYPTO_BA+0xA2C	W	ECC The scalar SCALARK word11 of point multiplication	0x0000_0000
CRPT_ECC_K_12	CRYPTO_BA+0xA30	W	ECC The scalar SCALARK word12 of point multiplication	0x0000_0000
CRPT_ECC_K_13	CRYPTO_BA+0xA34	W	ECC The scalar SCALARK word13 of point multiplication	0x0000_0000
CRPT_ECC_K_14	CRYPTO_BA+0xA38	W	ECC The scalar SCALARK word14 of point multiplication	0x0000_0000
CRPT_ECC_K_15	CRYPTO_BA+0xA3C	W	ECC The scalar SCALARK word15 of point multiplication	0x0000_0000
CRPT_ECC_K_16	CRYPTO_BA+0xA40	W	ECC The scalar SCALARK word16 of point multiplication	0x0000_0000
CRPT_ECC_K_17	CRYPTO_BA+0xA44	W	ECC The scalar SCALARK word17 of point multiplication	0x0000_0000
CRPT_ECC_SADDR	CRYPTO_BA+0xA48	R/W	ECC DMA Source Address Register	0x0000_0000
CRPT_ECC_DADDR	CRYPTO_BA+0xA4C	R/W	ECC DMA Destination Address Register	0x0000_0000
CRPT_ECC_STARTREG	CRYPTO_BA+0xA50	R/W	ECC Starting Address of Updated Registers	0x0000_0000
CRPT_ECC_WORDCNT	CRYPTO_BA+0xA54	R/W	ECC DMA Word Count	0x0000_0000
CRYPTO_RSA_CTL	CRYP_BA+0x1000	R/W	RSA Control Register	0x0000_0000
CRYPTO_RSA_STS	CRYP_BA+0x1004	R	RSA Status Register	0x0000_0000
CRYPTO_RSA_M_i i = 0,1,…,63	CRYP_BA+0x1008 + 0x4 * i	R/W	RSA the base of exponentiation word i	0x0000_0000
CRYPTO_RSA_E_i i = 0,1,…,63	CRYP_BA+0x1208 + 0x4 * i	W	RSA the exponent of exponentiation word i	0x0000_0000
CRYPTO_RSA_N_i i = 0,1,…,63	CRYP_BA+0x1408 + 0x4 * i	R/W	RSA the base of modulus operation word i	0x0000_0000
CRYPTO_RSA_C_i i = 0,1,…,63	CRYP_BA+0x1608 + 0x4 * i	R/W	RSA the constant value of Montgomery domain word i	0x0000_0000
CRYPTO_RSA_SADDR	CRYP_BA+0x1808	R/W	RSA DMA Source Address Register	0x0000_0000

CRYPTO_RSA_DADDR	CRYP_BA+0x180C	R/W	RSA DMA Destination Address Register	0x0000_0000
CRYPTO_RSA_STARTREG	CRYP_BA+0x1810	R/W	RSA Starting Address of Updated Registers	0x0000_0000
CRYPTO_RSA_WORDCNT	CRYP_BA+0x1814	R/W	RSA Data Word Count	0x0000_0000

26 圖像擷取接口 (Capture Sensor Interface Controller)

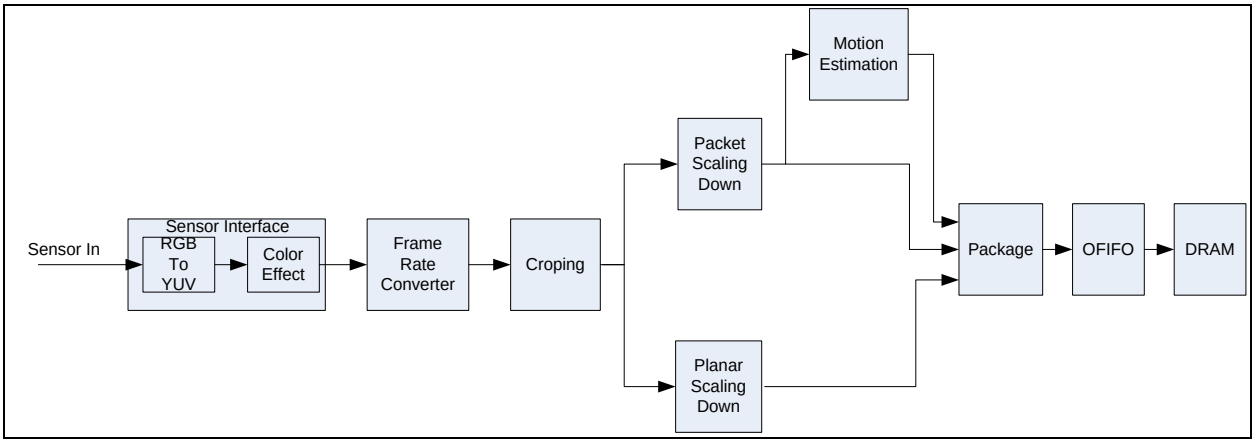
26.1 概述

NUC980 系列有兩個 CAP 控制器，圖像擷取接口是從傳感器捕捉到圖像數據。捕捉或獲取的圖像數據之後，將處理的圖像數據經過 FIFO 輸出到幀緩衝器。

26.2 特性

- 支持 2 個 CAP 控制器, CAP0 和 CAP1
- 8 位元 RGB565 傳感器
- 8 位元 YUV422 傳感器
- 支持 CCIR601 YCbCr 色彩範圍擴大為全 YUV 色彩範圍
- 支持 4 種格式的分組數據輸出：YUYV，Y only，RGB565，RGB555
- 支持 YUV422 planar 數據輸出
- 支持裁剪輸入圖像
- 支持縮放輸入圖像
- 支持幀率控制
- 支持通過硬件緩存控制器包輸出雙緩衝控制
- 支持 negative/sepia/posterization 的色彩效果
- 支持兩個獨立的捕捉接口

26.3 方塊圖

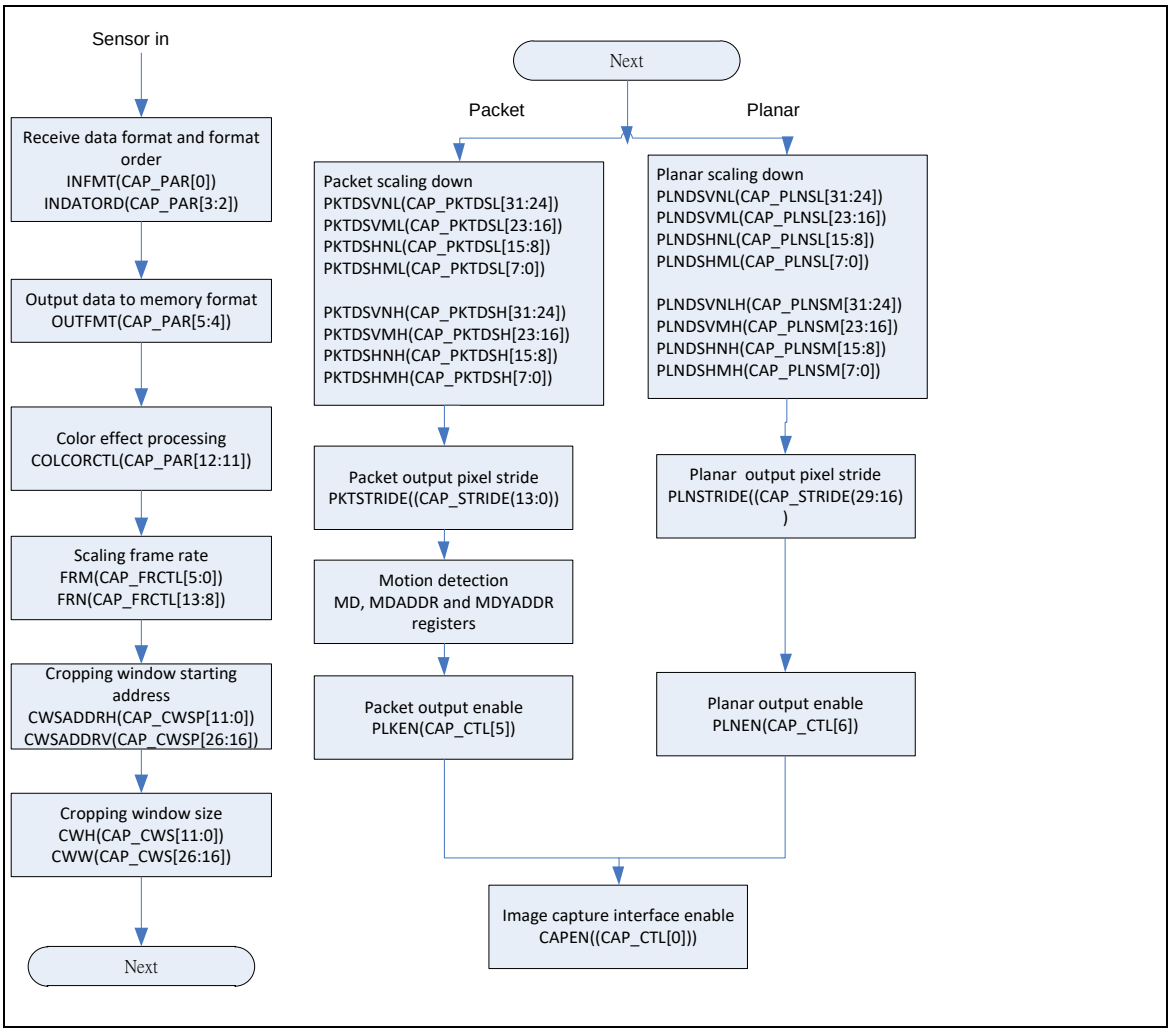


26.4 功能描述

26.4.1 基本配置

當 CAP0 使用之前，必須將 CAP0EN(HCLKEN1[26])設置為 1 和 SENSOR(HCLKEN[27])設置為 1, CAP0 時鐘來源選擇可以由 SENSOR0_S(CLKDIV3[23:16])設置，CAP0 引擎時鐘分頻器由 SENSOR0_N(CLKDIV3[27:24])設置。
當 CAP1 使用之前，必須將 CAP1EN(HCLKEN1[31])設置為 1 和 SENSOR(HCLKEN[27])設置為 1。CAP1 時鐘來源選擇可以由 SENSOR1_S(CLKDIV2[23:16])設置，CAP1 引擎時鐘分頻器由 SENSOR0_N(CLKDIV2[27:24])設置。

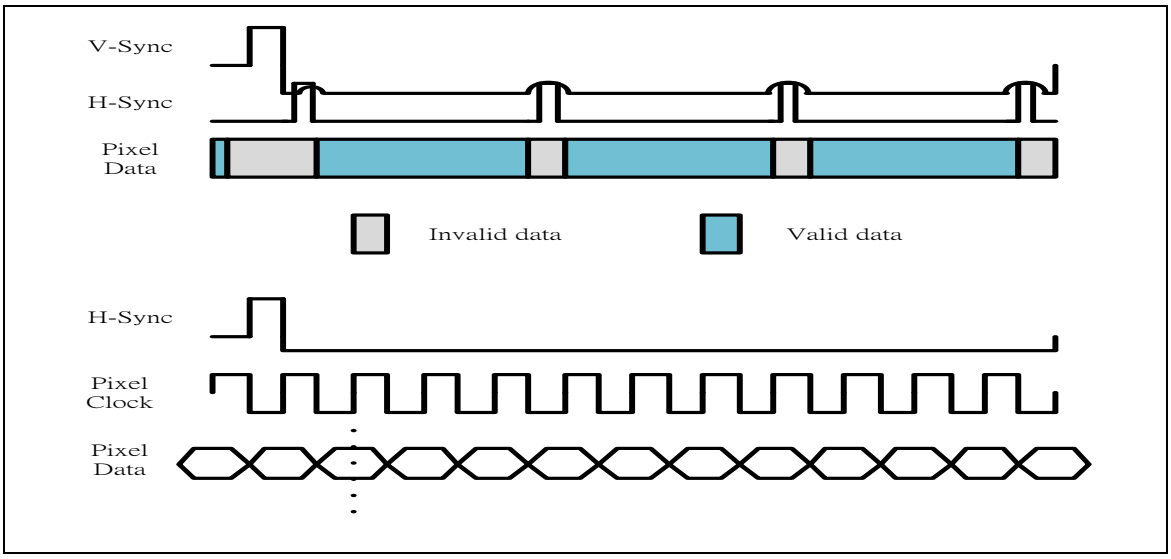
26.4.2 圖像捕捉流程圖



26.4.3 極性和輸入的數據

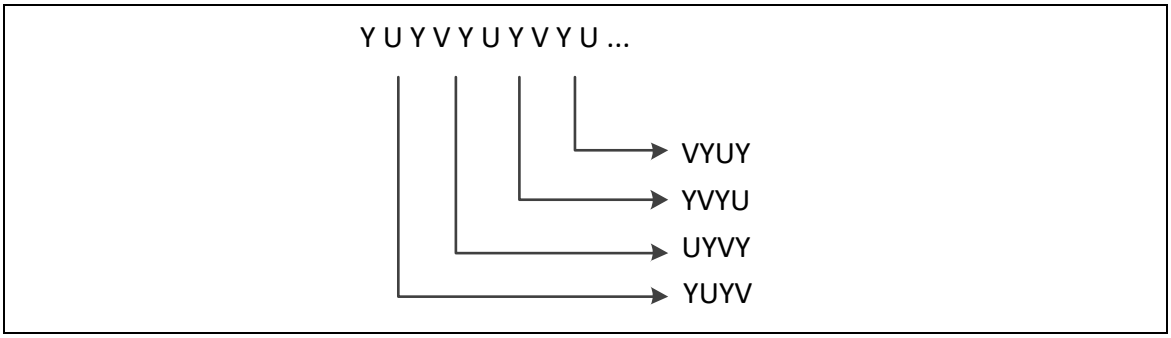
傳感器使用三個控制引腳去擷取一個新幀，一個新的水平線或一個新的像素。這些引腳分別 VSYNC，HSYNC和PCLK。 VSYNC和HSYNC定義的正或負時去同步。此外，PCLK的上升沿

或下降沿時擷取圖像數據。下圖顯示了垂直同步極性在高水，平同步的極性在高，在上升沿鎖時擷取數據。



26.4.4 傳感器數據輸入順序

輸入數據順序可能U0Y0VY1，Y0U0Y1V0，V0Y0U0Y1或Y0V0Y1U0裁剪輸入數據後。寄存器INDATORD(CAP_PAR[3：2])。



26.4.5 功輸入和輸出數據格式

傳感器可以輸出YcbCr422，RGB565，Only Y。然而，視頻引擎只支持YcbCr422和RGB565。輸入格式取決於傳感器初始表。程序員可以指定由INFMT(CAP_PAR[0])輸入格式

輸出格式取決於顯示裝置，程序員可以指定由OUTFMT(CAP_PAR[5:4])輸出格式。

26.4.6 縮小功能

視頻處理器支持圖像縮小演算法，Direct-Drop Algorithm (DDA)。例如：

如果截取窗口的尺寸等於640x480和目標尺寸等於352x288

The horizontal downscale factor =

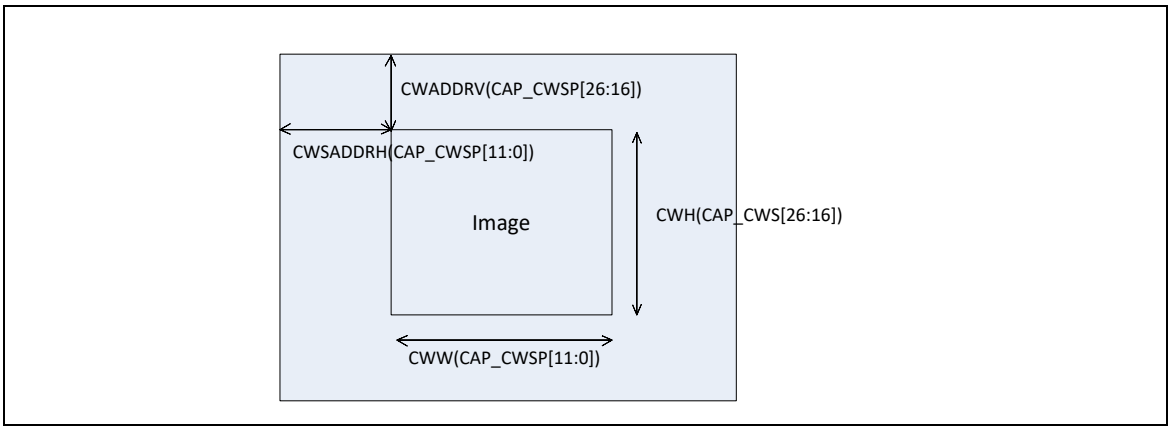
352/640 =(PKTSHNH<<8+PKTSHNL)/(PKTSHMH<<8+PKTSHML)

The vertical downscale factor =

288/480 =(PKTSVNH<<8+PKTSVNL)/(PKTSVMH<<8+PKTSVML)

26.4.7 裁剪功能

CAP接口可以設定接收到的圖像的窗口大小。窗口是由像素時鐘數（水平尺寸）和行數（垂直尺寸）中指定的窗口的大小。開始（左上角）坐標可以通過CAP_CWSP寄存器來指定。的大小（以線和在像素時鐘數水平維度的數目的垂直尺寸）可以通過CAP_CWS寄存器指定。如下圖表示：



26.4.8 快門模式（單張拍攝）

在快門模式下，即SHUTTER (CAP_CTL[16])=1，圖像捕捉接口接收圖像後則會停止捕捉圖像。

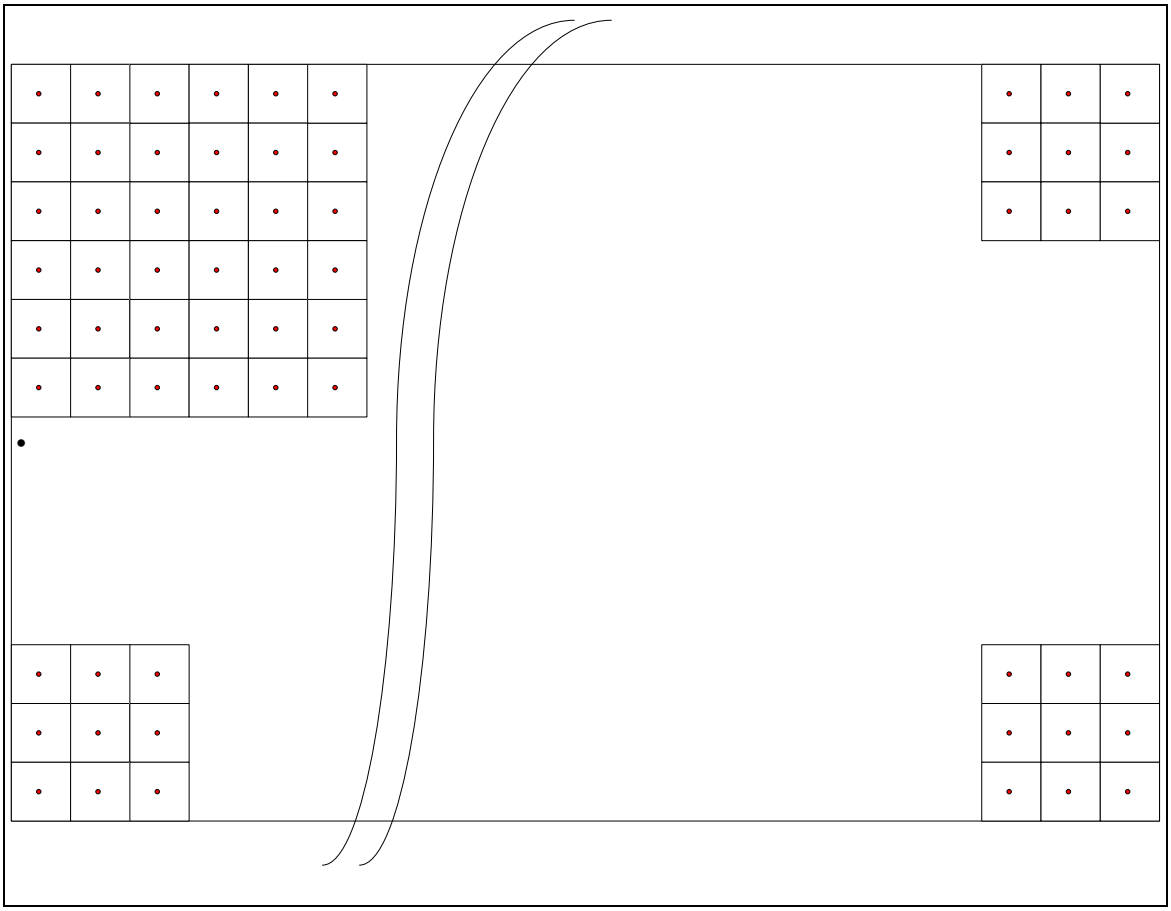
26.4.9 運動檢測

該功能用於檢測圖像中有物體移動。 功能如下：

1. 區塊大小支持8x8和16x16
2. 輸出移動偵測支持1位或8位（1位DIFF和7位閾值）

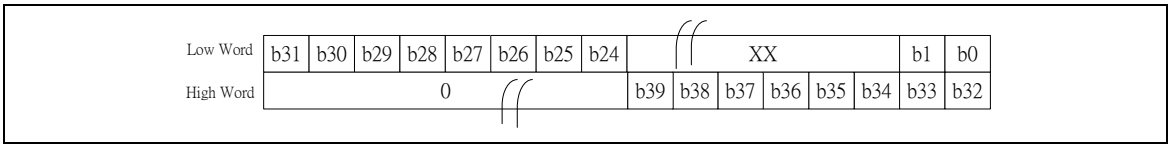
下圖說明了運動檢測模塊的工作原理。運動檢測模塊分離全幀分成8x8或16x16塊。獲得中心像素(4,4)或(8,8)，用於塊大小8x8或16x16。然後與前一幀為同一位置-MDYADDR(運動檢測的臨時Y緩衝區)進行比較。如果差值超過閾值設為1到運動檢測輸出buffer- MDADDR，否則設定為0至運動檢測輸出buffer- MDADDR。輸出中心像素的運動檢測的臨時Y緩衝區。如果輸出流是不

夠的一個字，它將被填充0。

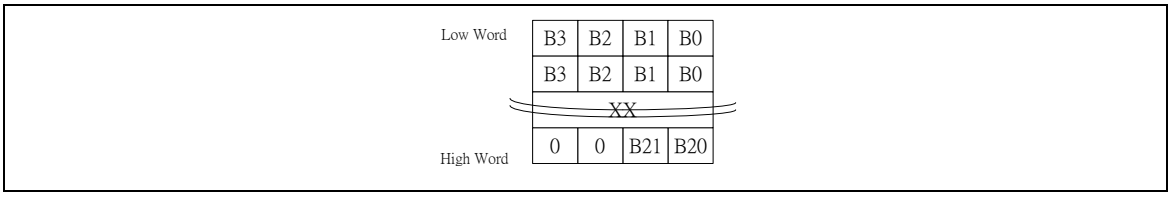


運動檢測輸出緩衝區列表格式如下。

- 一個bit模式（拍攝的寬度=640塊大小16x16）



- 1 bit DIFF（MSB）+7 bit Y的差值（捕獲寬度=352塊大小16x16）



26.5 寄存器

R: read only, **W**: write only, **R/W**: both read and write

Register	Offset	R/W	Description	Reset Value
Capture Base Address: **CAPx_BA = 0xB002_4000 - (0x10000*x)** **x=0, 1**				
CAPx_CTL	CAPx_BA+0x00	R/W	Image Capture Interface Control Register	0x0000_0040
CAPx_PAR	CAPx_BA+0x04	R/W	Image Capture Interface Parameter Register	0x0000_0000
CAPx_INT	CAPx_BA+0x08	R/W	Image Capture Interface Interrupt Register	0x0000_0000
CAPx_POSTERIZE	CAPx_BA+0x0C	R/W	YUV Component Posterizing Factor Register	0x0000_0000
CAPx_MD	CAPx_BA+0x10	R/W	Motion Detection Register	0x0000_0000
CAPx_MDADDR	CAPx_BA+0x14	R/W	Motion Detection Output Address Register	0x0000_0000
CAPx_MDYADDR	CAPx_BA+0x18	R/W	Motion Detection Temp Y Output Address Register	0x0000_0000
CAPx_SEPIA	CAPx_BA+0x1C	R/W	Sepia Effect Control Register	0x0000_0000
CAPx_CWSP	CAPx_BA+0x20	R/W	Cropping Window Starting Address Register	0x0000_0000
CAPx_CWS	CAPx_BA+0x24	R/W	Cropping Window Size Register	0x0000_0000
CAPx_PKTSL	CAPx_BA+0x28	R/W	Packet Scaling Vertical/Horizontal Factor Register (LSB)	0x0000_0000
CAPx_PLNSL	CAPx_BA+0x2C	R/W	Planar Scaling Vertical/Horizontal Factor Register (LSB)	0x0000_0000
CAPx_FRCTL	CAPx_BA+0x30	R/W	Scaling Frame Rate Factor Register	0x0000_0000
CAPx_STRIDE	CAPx_BA+0x34	R/W	Frame Output Pixel Stride Width Register	0x0000_0000
CAPx_FIFOTH	CAPx_BA+0x3C	R/W	FIFO Threshold Register	0x070D_0507
CAPx_CMPADDR	CAPx_BA+0x40	R/W	Compare Memory Base Address Register	0xFFFF_FFFC
CAPx_PKTSM	CAPx_BA+0x48	R/W	Packet Scaling Vertical/Horizontal Factor Register (MSB)	0x0000_0000
CAPx_PLNSM	CAPx_BA+0x4C	R/W	Planar Scaling Vertical/Horizontal Factor Register (MSB)	0x0000_0000
CAPx_CURADDRP	CAPx_BA+0x50	R	Current Packet System Memory Address Register	0x0000_0000
CAPx_CURADDRY	CAPx_BA+0x54	R	Current Planar Y System Memory Address Register	0x0000_0000
CAPx_CURADDRU	CAPx_BA+0x58	R	Current Planar U System Memory Address Register	0x0000_0000
CAPx_CURVADDR	CAPx_BA+0x5C	R	Current Planar V System Memory Address Register	0x0000_0000
CAPx_PKTBA0	CAPx_BA+0x60	R/W	System Memory Packet Base Address 0 Register	0x0000_0000
CAPx_PKTBA1	CAPx_BA+0x64	R/W	System Memory Packet Base Address 1 Register	0x0000_0000
CAPx_YBA	CAPx_BA+0x80	R/W	System Memory Planar Y Base Address Register	0x0000_0000
CAPx_UBA	CAPx_BA+0x84	R/W	System Memory Planar U Base Address Register	0x0000_0000
CAPx_VBA	CAPx_BA+0x88	R/W	System Memory Planar V Base Address Register	0x0000_0000

27 模擬數字轉換 (ADC)

27.1 概述

NUC980系列包含 一個 9 通道12位的SAR型模擬 / 數字比較器 (SAR A/D比較器)。

NUC980 SAR ADC 可將 9 通道的輸入電壓轉換為 12 位的數字並且將轉換結果存放在寄存器中。

27.2 特性

- 分辨率：12位分辨率。
- DNL：+/-1.5 LSB，INL：+/-3 LSB。
- 速率：200KSPS。
- 類比輸入範圍：VREF到AGND，也是輸入輸出的範圍。
- 類比電源：2.7-3.6V。
- 數位電源：1.2V。
- 9個類比輸入。

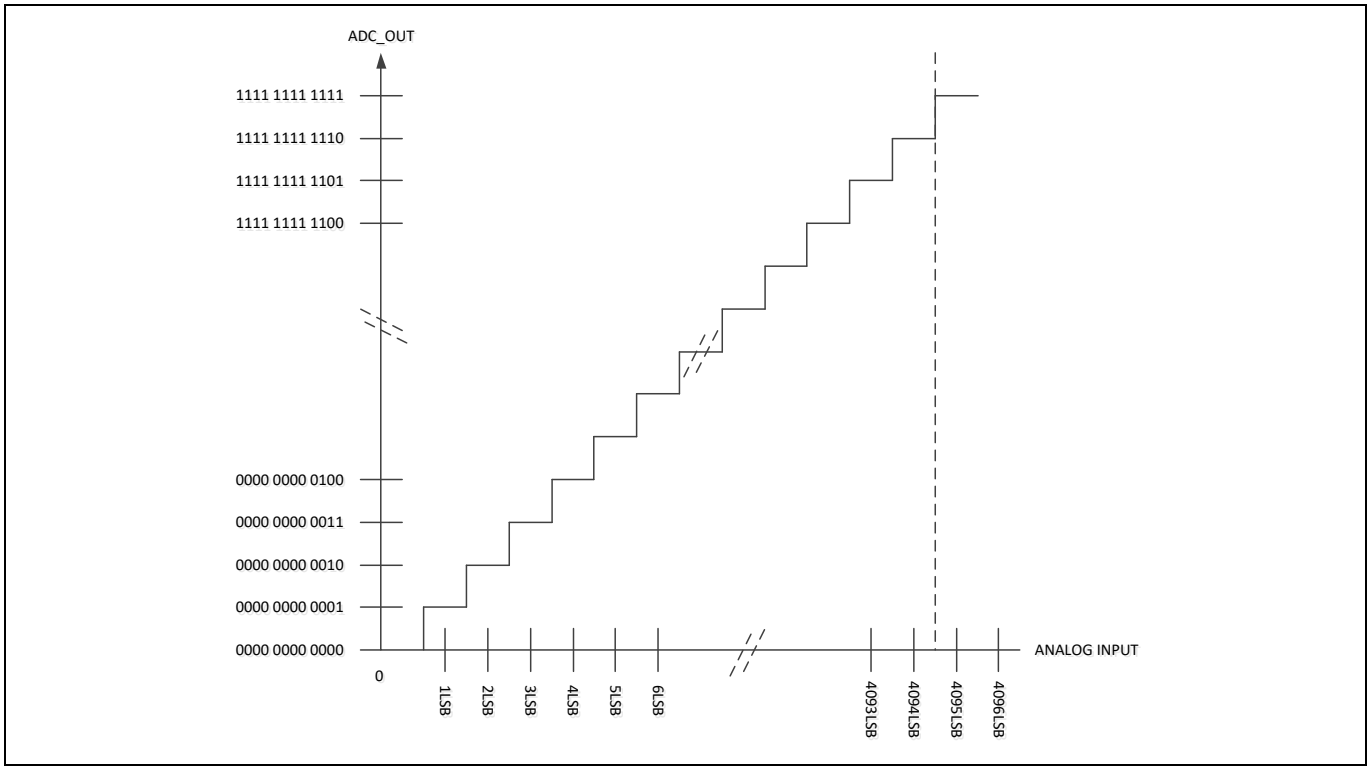
27.3 功能描述

27.3.1 基本配置

當ADC使用之前，必須將ADCEN(PCLKEN[24])設置為1。ADC 時鐘來源選擇可以由ADC_S(CLKDIV7[20:19])設置，ADC引擎時鐘分頻器由ADC_N(CLKDIV7[31:24]) 設置。

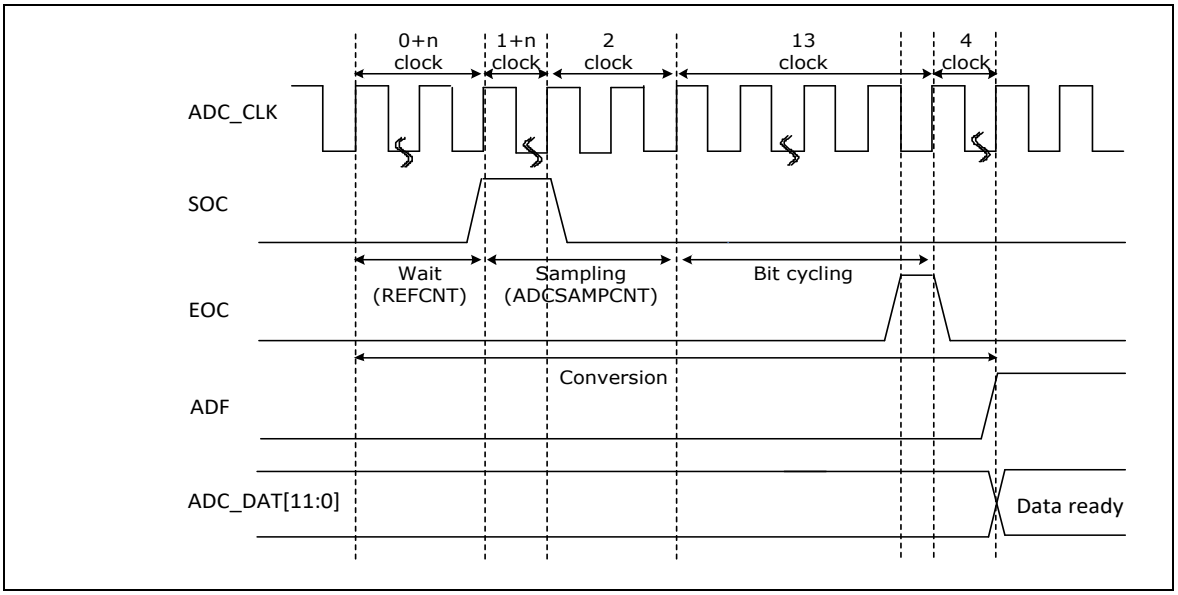
27.3.2 ADC 傳送模式

ADC 輸出編碼在二進制偏移，1LSB= VREF/ 4096，傳遞特性顯示在下面的圖表所示：



27.3.3 ADC 轉換時間

當 ADC 完成一筆類比/數字轉換後，會將轉換結果存放在寄存器中。另外，使用者可以設定等待參考電壓穩定的時間 REFCNT (ADC_CONF[19:16])，以及延長轉換時間ADCSAMPCNT (ADC_CONF[31:24])。



下面這個範例是將等待參考電壓穩定時間 REFCNT(ADC_CONF[19:16]) 設為 5，延長轉換時間 ADCSAMPLECNT(ADC_CONF[31:24]) 設為 10。

```
unsigned int refcnt, samplecnt;
refcnt = 5;
samplecnt = 10;
rREG_CONF = (rREG_CONF & ~0xF0000) | (refcnt << 16); /* set REFCNT */
rREG_CONF = (rREG_CONF & ~0xFF000000) | (samplecnt << 24); /* set SAMPLECNT */
```

27.3.4 一般偵測(Normal Detection)

一般偵測是指在單一通道下，完成一次ADC 轉換，示範一個一般偵測的軟件程序，如下：

```
char c,num;
unsigned int data,n;
unsigned int d1,d2,val=0;
rREG_CTL |= ADC_CTL_ADEN;
printf("select channel 0~7\n");
num=getchar();
switch(num)
{
    case '0': val=0; break;
    case '1': val=1; break;
    case '2': val=2; break;
    case '3': val=3; break;
    case '4': val=4; break;
    case '5': val=5; break;
    case '6': val=6; break;
    case '7': val=7; break;
}
rREG_CONF |= ADC_CONF_NACEN | val<<12 | (3<<6);
rREG_ISR = ADC_ISR_MF | ADC_ISR_NACF;
/* narmal_test interrupt mode */
rREG_IER |= ADC_IER_MIEN;
do{
    complete = 0;
    rREG_CTL |= ADC_CTL_MST;
    UART_printf("Waiting for Normal mode Interrupt\n");
    while(!(rREG_ISR & ADC_ISR_MF));
    rREG_ISR = ADC_ISR_MF; //Clear MF flag
    if(rREG_ISR & ADC_ISR_NACF)
    {
        data=rREG_DATA;
        n=(33*data*100)>>12;
```

```
        d1=n/1000;
        d2=n%1000;
        printf("DATA=0x%08x,voltage=%d.%dv\n",data,d1,d2);
    }
    else
        UART_printf("interrupt error\n");
}while(1);
```

27.4 寄存器

R: read only, W: write only, R/W: both read and write.

Register	Offset	R/W	Description	Reset Value
ADC Base Address: **ADC_BA = 0xB004_3000**				
ADCON	ADC_BA+0x00	R/W	ADC Control	0x0000_0000
ADC_CONF	ADC_BA+0x04	R/W	ADC Configure	0x0000_0000
ADC_IER	ADC_BA+0x08	R/W	ADC Interrupt Enable Register	0x0000_0000
ADC_ISR	ADC_BA+0x0C	R/W	ADC Interrupt Status Register	0x0000_0000
ADC_DATA	ADC_BA+0x28	R	ADC Normal Conversion Data	0x0000_0000

Revision History

Date	Revision	Description
2018.7.22	1.00	1. Initially issued.
2019.6.26	1.01	1. Editorial change.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.