

NUC970 Family Programming Guide

Document Information

Abstract	This document introduces the control sequence of NUC970 family peripherals.
Apply to	NUC970 family, including NUC972, NUC972, NUC976 and NUC977.

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

*Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design.
Nuvoton assumes no responsibility for errors or omissions.*

All data and specifications are subject to change without notice.

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

www.nuvoton.com

Table of Contents

1 系統管理器 (SYSTEM MANAGER).....	13
1.1 概述.....	13
1.2 寄存器	13
1.3 功能描述	14
1.3.1端口多功能控制 (Multiple Function Control).....	14
1.3.2低電壓偵測/復位	15
1.3.3USB ID 偵測.....	15
2 時鐘控制器 (CLOCK CONTROLLER).....	16
2.1 概述.....	16
2.2 特性.....	16
2.3 方塊圖	16
2.4 寄存器	18
2.5 功能描述	18
2.5.1模組時鐘開關	18
2.5.2時鐘除頻器	18
2.5.3PLL 設置	19
3 模擬數字轉換((ADC)	20
3.1 概述.....	20
3.2 特性.....	20
3.3 方塊圖	20
3.4 寄存器	21
3.5 功能描述	22
3.5.1基本配置	22
3.5.2ADC傳送模式	22
3.5.3一般偵測(Normal Detection)	23
3.5.4電池電壓檢測 (Battery Voltage Detection).....	24
3.5.5鍵盤掃描(Key Pad Scan)	25
3.5.64線和5線觸摸屏(4-wire and 5-wire Touch Screen)	27
3.5.74線壓力測量(4-wire Pressure Measurement)	29
4 中斷控制器 (AIC)	30

4.1 概述.....	30
4.2 特性.....	30
4.3 方塊圖	30
4.4 寄存器	31
4.5 功能描述.....	32
4.5.1中斷通道配置	32
4.5.2中斷屏蔽	33
4.5.3中斷清除與設置	33
4.5.4軟件優先規則	33
4.5.5硬件優先規則	35
4.5.6中斷源	35
5 CAN.....	38
5.1 概述.....	38
5.2 特性.....	38
5.3 方塊圖	38
5.4 寄存器	39
5.5 功能描述	40
5.5.1CAN協定的幀編碼格式.....	40
5.5.2CAN硬體設定	41
5.5.3CAN傳送速率的設定	41
5.5.4CAN模塊的寄存器	43
5.5.5發送CAN報文	45
5.5.6接收CAN報文	47
5.5.7喚醒功能	48
6 加密加速器.....	49
6.1 概述.....	49
6.2 特性.....	49
6.3 方塊圖	50
6.4 寄存器	50
6.5 功能描述	56
6.5.1數據訪問	56
6.5.2通道擴展	57
6.5.3PRNG	57
6.5.4AES	58
6.5.5DES/TDES	60
6.5.6SHA	61
7 外部總線 (EXTERNAL BUS INTERFACE).....	63

7.1 概述	63
7.2 特性	63
7.3 方塊圖	63
7.4 寄存器	64
7.5 功能描述	64
7.5.1 基本配置	64
7.5.2 外部I/O控制	64
8 乙太網路控制器 (EMAC)	66
8.1 概述	66
8.2 特性	66
8.3 方塊圖	66
8.4 寄存器	67
8.5 功能描述	69
8.5.1 PHY 控制	69
8.5.2 CAM 設置	71
8.5.3 控制封包	71
8.5.4 遠端喚醒 (WoL)	71
8.5.5 封包接收	72
8.5.6 封包傳送	77
8.5.7 網路時鐘	83
8.5.8 錯誤處理	86
9 加強型計時器控制器 (ETMR)	89
9.1 概述	89
9.2 特性	89
9.3 方塊圖	89
9.4 寄存器	89
9.5 功能描述	90
9.5.1 計時器計時初始化	90
9.5.2 計時器捕獲初始化	90
9.5.3 中斷處理	91
9.5.4 計時器頻率	91
9.5.5 單次模式	92
9.5.6 週期模式	92
9.5.7 翻轉模式	93
9.5.8 連續計數模式	93
9.5.9 自由計數模式	94
9.5.10 觸發計數模式	95
9.5.11 計數器重定模式	96

9.5.12 捕獲模式去抖動.....	97
10 閃存接口 (FMI)	98
10.1 概述.....	98
10.2 特性.....	98
10.3 方塊圖	98
10.4 寄存器	99
10.5 功能描述	101
10.5.1 DMA 以及 FMI 全域控制.....	101
10.5.2 NAND Flash.....	101
10.5.3 eMMC.....	105
11 通用 DMA 控制器 (GDMA).....	110
11.1 概述.....	110
11.2 特性.....	110
11.3 方塊圖	110
11.4 寄存器	111
11.5 功能描述	112
11.5.1 Non-Descriptor功能描述.....	112
11.5.2 Descriptor功能描述.....	116
12 2D 圖形加速	120
12.1 概述.....	120
12.2 特性.....	120
12.3 方塊圖	121
12.4 寄存器	121
12.5 功能描述	122
12.5.1 初始化2D控制器.....	122
12.5.2 三元光柵操作(ROP).....	123
12.5.3 位塊傳送(BitBLT)	125
12.5.4 Bresenham畫線	129
12.5.5 α 混合.....	133
12.5.6 剪裁.....	134
12.5.7 旋轉.....	134
12.5.8 放大與縮小.....	135
13 通用 I/O 控制器 (GPIO).....	138
13.1 概述.....	138

13.2 特性	138
13.3 方塊圖	138
13.4 寄存器	139
13.5 功能描述	143
13.5.1 多功能引腳設置	143
13.5.2 GPIO 輸出模式	144
13.5.3 GPIO 輸入模式	144
13.5.4 GPIO 中斷	145
14 I²C	147
14.1 概述	147
14.2 特性	147
14.3 方塊圖	147
14.4 寄存器	148
14.5 功能描述	148
14.5.1 I ² C協定	148
14.5.2 連續傳送	149
14.5.3 中斷	149
14.5.4 軟體控制模式	149
14.5.5 I ² C命令寄存器操作	150
14.5.6 I ² C EEPROM操作使用示例(24LC64為例)	151
15 I²S	153
15.1 概述	153
15.2 特性	153
15.3 方塊圖	153
15.4 寄存器	154
15.5 功能描述	155
15.5.1 I ² S主/從模式設定	155
15.5.2 I ² S 時鐘源設定	155
15.5.3 I ² S 輸出入時鐘計算及設置	156
15.5.4 設置DMA	156
15.5.5 DMA數據在內存存放順序	157
15.5.6 介面選擇及設置	158
15.5.7 PCM介面的配置	158
15.5.8 數據分割	159
16 JPEG 編解碼器	162
16.1 概述	162

16.2 特性	162
16.3 方塊圖	162
16.4 寄存器	163
16.5 功能說明	165
16.5.1 訪問記憶體(Memory Access)	165
16.5.2 JPEG 編碼	167
16.5.3 一般編碼	167
16.5.4 影像放大編碼	168
16.5.5 JPEG 解碼	171
17 鍵盤接口 (KPI)	179
17.1 概述	179
17.2 特性	179
17.3 方塊圖	179
17.4 寄存器	180
17.5 功能描述	180
17.5.1 控制器配置	181
17.5.2 喚醒功能	182
18 顯示控制器 (LCM)	183
18.1 概述	183
18.2 特性	183
18.3 方塊圖	184
18.4 寄存器	184
18.5 功能描述	185
18.5.1 LCD控制器編程流程	186
18.5.2 初始化和配置LCD控制器	188
18.5.3 配置OSD控制器	190
18.5.4 配置硬體游標功能	191
19 MTP 控制器	193
19.1 概述	193
19.2 特性	193
19.3 方塊圖	193
19.4 寄存器	194
19.5 功能描述	195
19.5.1 使用MTP	195
19.5.2 MTP 密鑰	195

19.5.3	用戶定義數據.....	196
19.5.4	MTP使能	196
19.5.5	MTP 密鑰燒錄	197
19.5.6	MTP 上鎖	197
19.5.7	MTP 密鑰用於 AES 加密/解密	198
19.5.8	MTP 密鑰用於 SHA/HMAC 比對.....	199
20	脈寬調製 (PWM)	200
20.1	概述.....	200
20.2	特性	200
20.3	方塊圖	201
20.4	寄存器	201
20.5	功能描述	202
20.5.1	PWM 計時器操作	202
20.5.2	PWM 雙緩存功能	203
20.5.3	連續以及單次操作	204
20.5.4	死區發生器	204
20.5.5	PWM 計時器開啟過程	205
20.5.6	PWM 計時器停止過程	206
21	實時時鐘 (RTC).....	207
21.1	概述.....	207
21.2	特性	207
21.3	方塊圖	207
21.4	寄存器	208
21.5	功能描述	209
21.5.1	初始化.....	209
21.5.2	訪問(讀/寫)限制	209
21.5.3	12/24 小時格式顯示切換	210
21.5.4	設定日期和時間	211
21.5.5	絕對時間鬧鐘設定	211
21.5.6	相對時間鬧鐘設定	212
21.5.7	喚醒功能	213
21.5.8	時鐘節拍	215
21.5.9	系統電源控制流程	216
21.5.10	頻率補償:.....	222
22	智能卡接口 (SC).....	224
22.1	概述.....	224
22.2	特性	224

22.3 方塊圖	225
22.4 寄存器	225
22.5 功能描述	226
22.5.1 啟動 (Cold Reset)	227
22.5.2 暖復位 (Warm Reset)	228
22.5.3 釋放 (Deactivation)	229
22.5.4 資料格式	230
22.5.5 資料傳輸	231
22.5.6 錯誤信號和字元重複	231
22.5.7 內部超時計數器	232
22.5.8 智能卡插拔偵測	234
22.5.9 其他傳輸 相關設置	235
22.5.10 串口(UART) 模式	235
23 SD 控制器 (SDH)	237
23.1 概述	237
23.2 特性	237
23.3 方塊圖	237
23.4 寄存器	238
23.5 功能描述	239
23.5.1 全域控制	241
23.5.2 發送命令	242
23.5.3 取得響應	242
23.5.4 讀取SD 卡	242
23.5.5 寫入 SD 卡	243
24 SPI	245
24.1 概述	245
24.2 特性	245
24.3 方塊圖	245
24.4 寄存器	246
24.5 功能描述	246
24.5.1 從機選擇	246
24.5.2 自動從機選擇	247
24.5.3 雙/四 IO 模式	247
24.5.4 連續傳送	249
24.5.5 中斷	250
24.5.6 SPI控制器完整使用示例	250
25 計時器控制器 (TIMER)	251

25.1 概述	251
25.2 特性	251
25.3 方塊圖	251
25.4 寄存器	252
25.5 功能描述	252
25.5.1 計時器初始化	252
25.5.2 中斷處理	253
25.5.3 計時器頻率	253
25.5.4 單次模式	253
25.5.5 週期模式	254
25.5.6 連續計數模式	254
26 通用異步收發器 (UART)	256
26.1 概述	256
26.2 特性	257
26.3 方塊圖	258
26.4 寄存器	259
26.5 功能描述	260
26.5.1 初始化	260
26.5.2 IrDA功能模式	261
26.5.3 RS485功能模式	262
26.5.4 LIN功能模式	263
27 USB 2.0 設備控制器	265
27.1 概述	265
27.2 特性	265
27.3 方塊圖	265
27.4 寄存器	266
27.5 功能描述	270
27.5.1 初始	270
27.5.2 中斷處理程序	271
27.5.3 Standard Request	272
27.5.4 Set Address Request	272
27.5.5 Get Descriptor	273
27.5.6 IN 傳輸	274
27.5.7 OUT 傳輸	274
28 USB 主機控制器	276
28.1 概述	276

28.2 特性	276
28.3 方塊圖	276
28.3.1 基本配置	277
28.3.2 EHCI 控制器	277
28.3.3 OHCI 控制器	278
28.4 寄存器	279
28.5 功能描述	281
28.5.1 初始設置	281
28.5.2 根集線器端口路由	281
28.5.3 OHCI	281
28.5.4 EHCI	286
29 圖像擷取接口 (CAPTURE SENSOR INTERFACE CONTROLLER)	294
29.1 概述	294
29.2 特性	294
29.3 方塊圖	294
29.4 寄存器	294
29.5 功能描述	295
29.5.1 基本配置	295
29.5.2 圖像捕捉流程圖	296
29.5.3 極性和輸入的數據	296
29.5.4 傳感器數據輸入順序	297
29.5.5 功輸入和輸出數據格式	297
29.5.6 縮小功能	297
29.5.7 裁剪功能	298
29.5.8 快門模式 (單張拍攝)	298
29.5.9 運動檢測	298
30 看門狗計時器控制器 (WDT)	300
30.1 概述	300
30.2 特性	300
30.3 方塊圖	300
30.4 寄存器	300
30.5 功能描述	301
30.5.1 WDT 設置	301
30.5.2 WDT喚醒功能	302
31 窗口看門狗定時控制器 (WWDT)	303
31.1 概述	303

31.2 特性	303
31.3 方塊圖	303
31.4 寄存器	303
31.5 功能描述	304
31.5.1 超時設置.....	304
31.5.2 WWDT 中斷.....	305
31.5.3 系統復位.....	305
31.5.4 使用限制.....	305

1 系統管理器 (System Manager)

1.1 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

1.2 寄存器

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

Register	Offset	R/W	Description	Reset Value
SYS_BA = 0xB000_0000				
SYS_P DID	SYS_BA+0x000	R	Product Identifier Register	0x0X30_D008 ^[1]
SYS_PWRON	SYS_BA+0x004	R/W	Power-On Setting Register	Undefined ^[2]
SYS_ARBCON	SYS_BA+0x008	R/W	Arbitration Control Register	0x0000_0000
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_WKUPSER	SYS_BA+0x058	R/W	System Wakeup Source Enable Register	0x0000_0000
SYS_WKUPSSR	SYS_BA+0x05C	R/W	System Wakeup Source Status Register	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_0007
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	0x0000_0000
SYS_GPH_MFPL	SYS_BA+0x0A8	R/W	GPIOH Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPH_MFPH	SYS_BA+0x0AC	R/W	GPIOH High Byte Multiple Function Control Register	0x0000_0000
SYS_GPI_MFPL	SYS_BA+0x0B0	R/W	GPIOI low Byte Multiple Function Control Register	0x0000_0000
SYS_GPI_MFPH	SYS_BA+0x0B4	R/W	GPIOI High Byte Multiple Function Control Register	0x0000_0000
SYS_GPJ_MFPL	SYS_BA+0x0B8	R/W	GPIOJ low Byte Multiple Function Control Register	Undefined ^[2]
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_ICEDBGCR	SYS_BA+0x104	R/W	ICE Debug Interface Control Register	0x0000_0001
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.

1.3 功能描述

1.3.1 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GPB6~9設定成SPI0所使用的pin腳。GPB6是SPI0_SS0，GPIO7是SPI0_CLK，GPIO8是SPI0_DATA0，GPIO9是SPI0_DATA1，因此，SYS_GPB_MFPL要填入0xBB000000，SYS_GPB_MFPH要填入0xBB。

1.3.2 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。

1.3.3 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：

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

2 時鐘控制器 (Clock Controller)

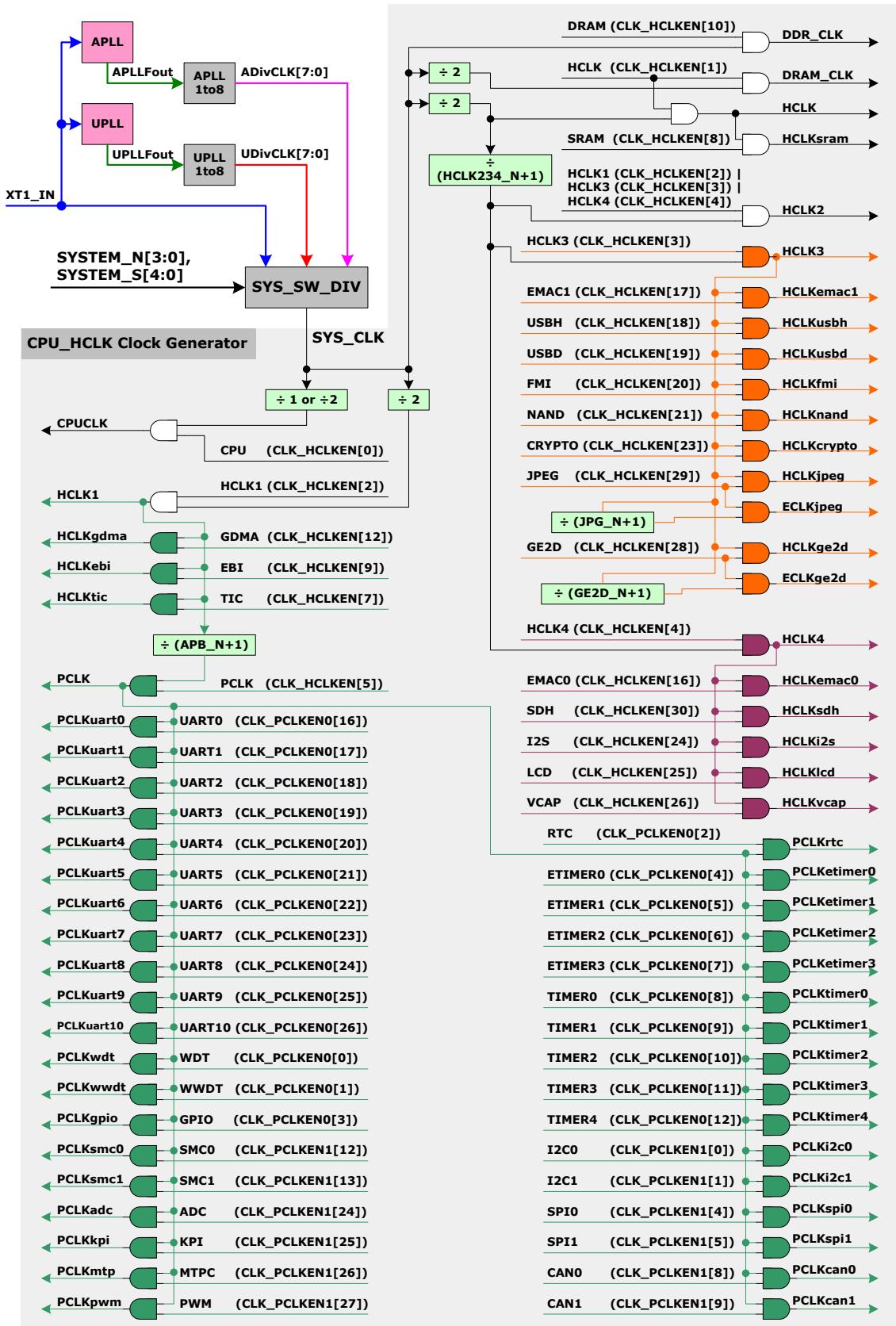
2.1 概述

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。

2.2 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

2.3 方塊圖



2.4 寄存器

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

Register	Address	R/W	Description	Reset Value
CLK_BA = 0xB000_0200				
CLK_PMCN	CLK_BA+0x00	R/W	Power Management Control Register	0xFFFF_FF03
CLK_HCLKEN	CLK_BA+0x10	R/W	AHB IP Clock Enable Control Register	0x0000_0527
CLK_PCLKEN0	CLK_BA+0x18	R/W	APB IP Clock Enable Control Register 0	0x0000_0000
CLK_PCLKEN1	CLK_BA+0x1C	R/W	APB IP Clock Enable Control Register 1	0x0000_0000
CLK_DIVCTL0	CLK_BA+0x20	R/W	Clock Divider Control Register 0	0x0100_00XX
CLK_DIVCTL1	CLK_BA+0x24	R/W	Clock Divider Control Register 1	0x0000_0000
CLK_DIVCTL2	CLK_BA+0x28	R/W	Clock Divider Control Register 2	0x0000_0000
CLK_DIVCTL3	CLK_BA+0x2C	R/W	Clock Divider Control Register 3	0x0000_0000
CLK_DIVCTL4	CLK_BA+0x30	R/W	Clock Divider Control Register 4	0x0000_0000
CLK_DIVCTL5	CLK_BA+0x34	R/W	Clock Divider Control Register 5	0x0000_0000
CLK_DIVCTL6	CLK_BA+0x38	R/W	Clock Divider Control Register 6	0x0000_0000
CLK_DIVCTL7	CLK_BA+0x3C	R/W	Clock Divider Control Register 7	0x0000_0000
CLK_DIVCTL8	CLK_BA+0x40	R/W	Clock Divider Control Register 8	0x0000_0500
CLK_DIVCTL9	CLK_BA+0x44	R/W	Clock Divider Control Register 9	0x0000_0000
CLK_APLLCON	CLK_BA+0x60	R/W	APLL Control Register	0x1000_0015
CLK_UPLLCON	CLK_BA+0x64	R/W	UPLL Control Register	0xX000_0015
CLK_PLLSTBCNTR	CLK_BA+0x80	R/W	PLL Stable Counter and Test Clock Control Register	0x0000_1800

2.5 功能描述

2.5.1 模組時鐘開關

NUC970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。

2.5.2 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. SDH時鐘除頻寄存器為CLKDIV9，需要控制SDH_N，SDH_S，SDH_SDIV。
2. 設定SDH時鐘來源是UPLL，SDH_S位要填入11b。
3. 初始化頻率為300KHz，要先將UPLL（300MHz）除5變成60MHz，就是SDH_SDIV填入4，再除以200，就是SDH_N填入199，即可得到300KHz輸出。
4. 數據傳輸頻率為50MHz，SDH_SDIV填入0，代表UPLL不需要先除頻，直接設定SDH_N為5，代表300MHz除6，就能輸出50MHz。

2.5.3 PLL 設置

NUC970 PLL默認值是設定264MHz，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。

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

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

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

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

N	F _{pfld} Range
1	11.0 ≤ F _{pfld} ≤ 80
2	7.0 ≤ F _{pfld} ≤ 80
3	5.0 ≤ F _{pfld} ≤ 80
4	4.0 ≤ F _{pfld} ≤ 80
5	3.5 ≤ F _{pfld} ≤ 80
6	3.0 ≤ F _{pfld} ≤ 80
7 ~ 8	2.5 ≤ F _{pfld} ≤ 80
9 ~ 10	3.5 ≤ F _{pfld} ≤ 80
11 ~ 40	3.0 ≤ F _{pfld} ≤ 80
41 ~ 128	2.5 ≤ F _{pfld} ≤ 80

3 模擬數字轉換((ADC)

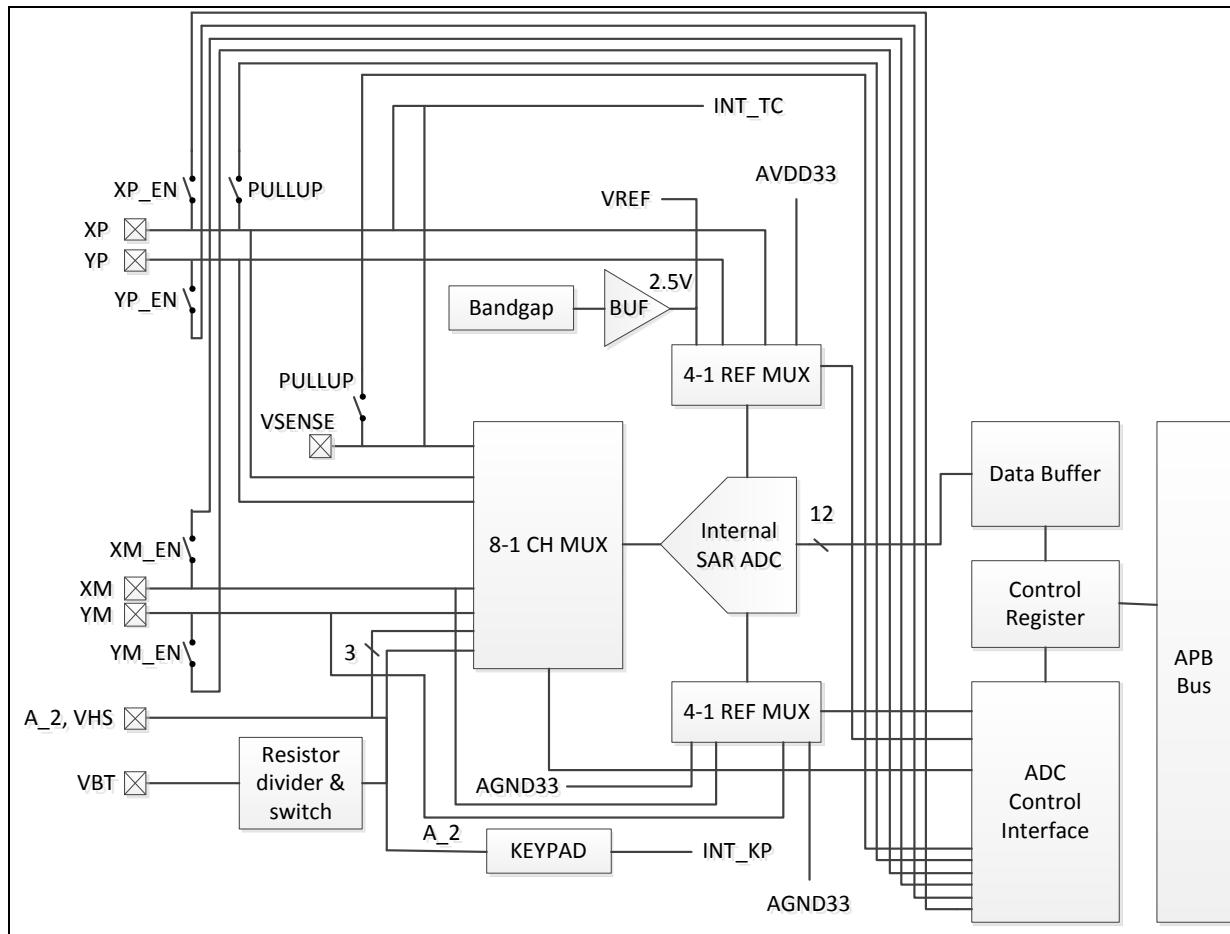
3.1 概述

NUC970系列包含 一個8通道12位的SAR型模擬 / 數字比較器 (SAR A/D比較器)。在A / D轉換器支持兩種操作模式：四線或五線的模式。NUC9xx系列的ADC是特別適合於作為觸摸屏控制器。電池電壓檢測可以由SAR ADC很容易地完成。也支援鍵盤中斷信號發生器。

3.2 特性

- 分辨率：12位分辨率。
- DNL : +/-1.5 LSB , INL : +/-3 LSB 。
- 雙速率：1MSPS/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 。
- 8個類比輸入。
- 兼容4線或5線觸摸屏接口。
- 在4線觸摸屏支援觸摸壓力。
- 電池測量。
- 鍵盤中斷發生器。
- 自動關機。
- 低功耗：4850uW (@1MSPS) /2170uW (@200KSPS) , <1uA

3.3 方塊圖



3.4 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 = 0xB800_A000				
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_WKISR	ADC_BA+0x10	R	ADC Wake Up Interrupt Status Register	0x0000_0000
ADC_XYDATA	ADC_BA+0x20	R	ADC Touch X,Y Position Data	0x0000_0000
ADC_ZDATA	ADC_BA+0x24	R	ADC Touch Z Pressure Data	0x0000_0000
ADC_DATA	ADC_BA+0x28	R	ADC Normal Conversion Data	0x0000_0000
ADC_VBATDATA	ADC_BA+0x2C	R	ADC Battery Detection Data	0x0000_0000
ADC_KPDATA	ADC_BA+0x30	R	ADC Key Pad Data	0x0000_0000

ADC_SELFDATA	ADC_BA+0x34	R	ADC Self-Test Data	0x0000_0000
ADC_XYSORT0	ADC_BA+0x1F4	R	ADC Touch XY Position Mean Value Sort 0	0x0000_0000
ADC_XYSORT1	ADC_BA+0x1F8	R	ADC Touch XY Position Mean Value Sort 1	0x0000_0000
ADC_XYSORT2	ADC_BA+0x1FC	R	ADC Touch XY Position Mean Value Sort 2	0x0000_0000
ADC_XYSORT3	ADC_BA+0x200	R	ADC Touch XY Position Mean Value Sort 3	0x0000_0000
ADC_ZSORT0	ADC_BA+0x204	R	ADC Touch Z Pressure Mean Value Sort 0	0x0000_0000
ADC_ZSORT1	ADC_BA+0x208	R	ADC Touch Z Pressure Mean Value Sort 1	0x0000_0000
ADC_ZSORT2	ADC_BA+0x20C	R	ADC Touch Z Pressure Mean Value Sort 2	0x0000_0000
ADC_ZSORT3	ADC_BA+0x210	R	ADC Touch Z Pressure Mean Value Sort 3	0x0000_0000
MTMULCK	ADC_BA+0x220	W	ADC Manual Test Mode Unlock	0x0000_0000
MTCNF	ADC_BA+0x224	R/W	ADC Manual Test Mode Configure	0x0000_0000
MTCN	ADC_BA+0x228	R/W	ADC Manual Test Mode Control	0x0000_0000
ADCAII	ADC_BA+0x22C	R	ADC Analog Interface Information	0x0000_0000
ADCAIIRLT	ADC_BA+0x230	R	ADC Analog Interface Information Result	0xFFFF_FFFF

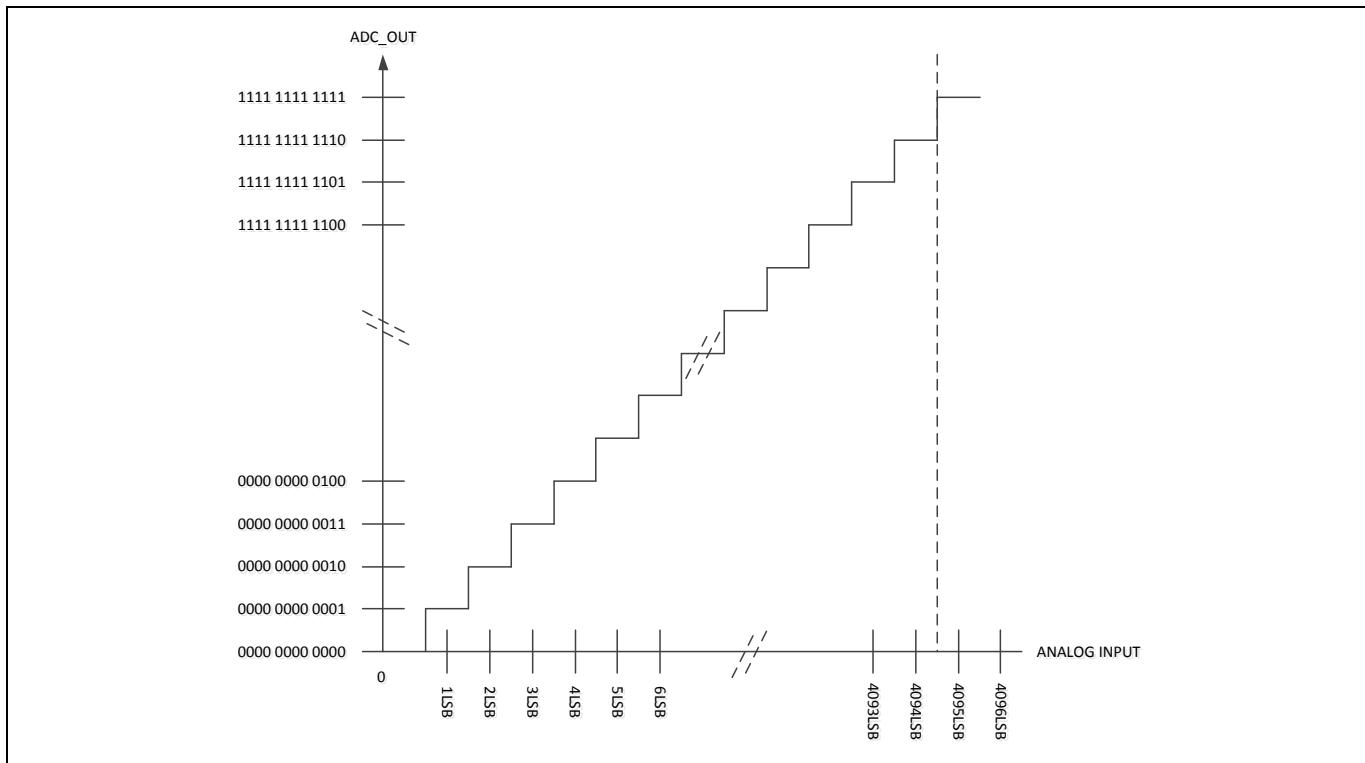
3.5 功能描述

3.5.1 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[23:16]) 設置，ADC 引擎時鐘分頻器由 ADC_N(CLKDIV7[31:24]) 設置。

3.5.2 ADC 傳送模式

ADC 輸出編碼在二進制偏移， $1\text{ LSB} = \text{VREF}/4096$ ，傳遞特性顯示在下面的圖表所示：



3.5.3 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:VBT(A0), 1:VHS(A1), 2:A2, 3:A3, 4:YM(A4), 5:YP(A5), 6:XM(A6),
7:XP(A7)\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<<3 | (3<<6) | (1<<22);
```

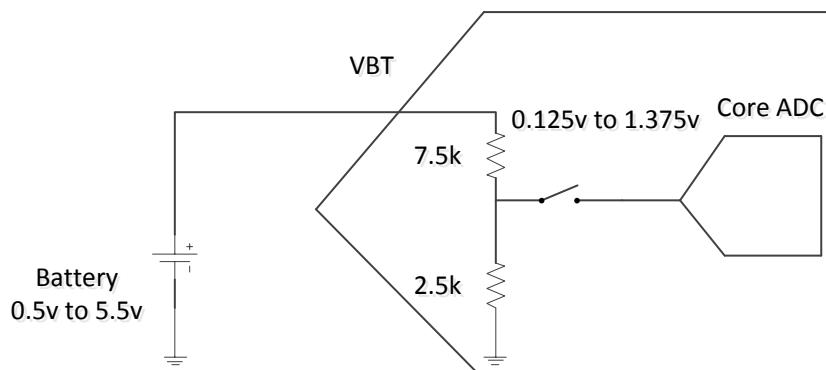
```

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);

```

3.5.4 電池電壓檢測 (Battery Voltage Detection)

用 VBT 作為輸入，並選擇內部 Bandgap 當作參考值。用於 ADC 的配置寄存器 VBAT_EN(ADC_CONF[8])應設置為1。電池電壓檢測內部電路如下：



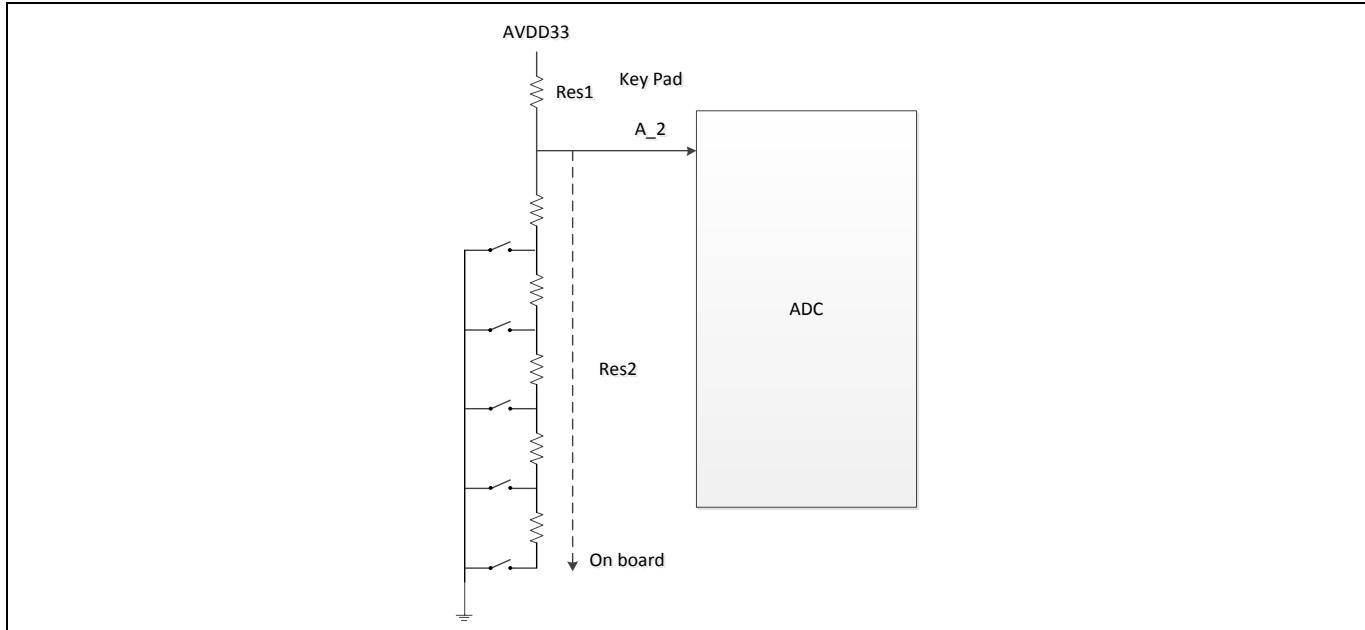
示範一個電池電壓檢測的軟件程序，如下：

```
unsigned int n,vbadata;
```

```
int d1,d2;
rREG_CTL |= ADC_CTL_ADEN | ADC_CTL_VBGEN;
rREG_CONF |= ADC_CONF_VBATEN;
rREG_ISR |= ADC_ISR_MF | ADC_ISR_VBF;
/* menu complete enable */
rREG_IER |= ADC_IER_MIEN;
do{
    complete = 0;
    rREG_CTL |= ADC_CTL_MST;
    printf("Waiting for bettrey Interrupt A0<inputer pin>\n");
    while(!(rREG_ISR & ADC_ISR_MF));
    rREG_ISR = ADC_ISR_MF; //Clear MF flag
    if(rREG_ISR & ADC_ISR_VBF )
    {
        vbadata=rREG_VBADATA;
        n=(25*vbadata*100)>>12;
        d1=n/1000;
        d2=n%1000;
        printf("VBATDATA=0x%08x, voltage=%d.%dv\n",vbadata,d1,d2);
        rREG_ISR = ADC_ISR_VBF; //clear VBF flag
    }
    else
        printf("interrupt error\n");
}while(1);
```

3.5.5 鍵盤掃描(Key Pad Scan)

用A_2當作輸入，並選擇AVDD33和AGND33作為參考值。用於ADC的配置寄存器KPC_EN(ADC_CONF[9])應設置為1。其設計電路如下：



如果使用上述結構的鍵盤，同時需要中斷產生，請務必將**Res1** \leq 20K歐姆和**Res2** $<$ 5.6***RES1**。此外，一個0.01uF的上限，建議在**A_2**在船上。然而如果不需要中斷產生器，請忽略**RES1**和的**Res2**的要求。

示範一個鍵盤掃描的軟件程序，如下：

```
rREG_CTL |= ADC_CTL_ADEN | ADC_CTL_PKWPEN;
rREG_CONF = ADC_CONF_KPCEN ;
rREG_ISR |= 0x0000FFFF;
rREG_IER |= ADC_IER_KPEIEN;

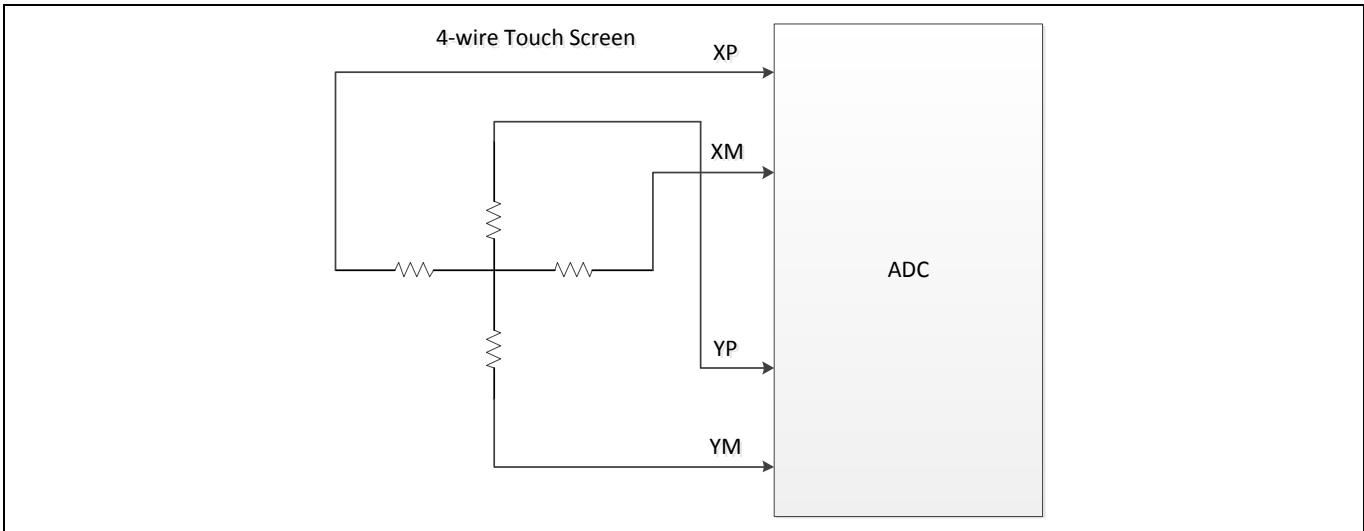
rREG_CONF |= ADC_CONF_KPCEN;
rREG_ISR = ADC_ISR_MF | ADC_ISR_KPCF;
/* keypad interrupt mode */
rREG_IER |= ADC_IER_MIEN;
do{
    while(!(rREG_ISR & ADC_ISR_KPEF)); // Waiting for Interrupt
    rREG_ISR = ADC_ISR_KPEF; //Clear KPEF flag
    rREG_CTL |= ADC_CTL_MST;
    while(!(rREG_ISR & ADC_ISR_MF));
    rREG_ISR = ADC_ISR_MF; //Clear MF flag
    if(rREG_ISR & ADC_ISR_KPCF)
    {
        rREG_ISR = ADC_ISR_KPCF; //Clear KPCF flag
        printf("interrupt correct REG_KPDATA=0x%08x\n",rREG_KPDATA);
```

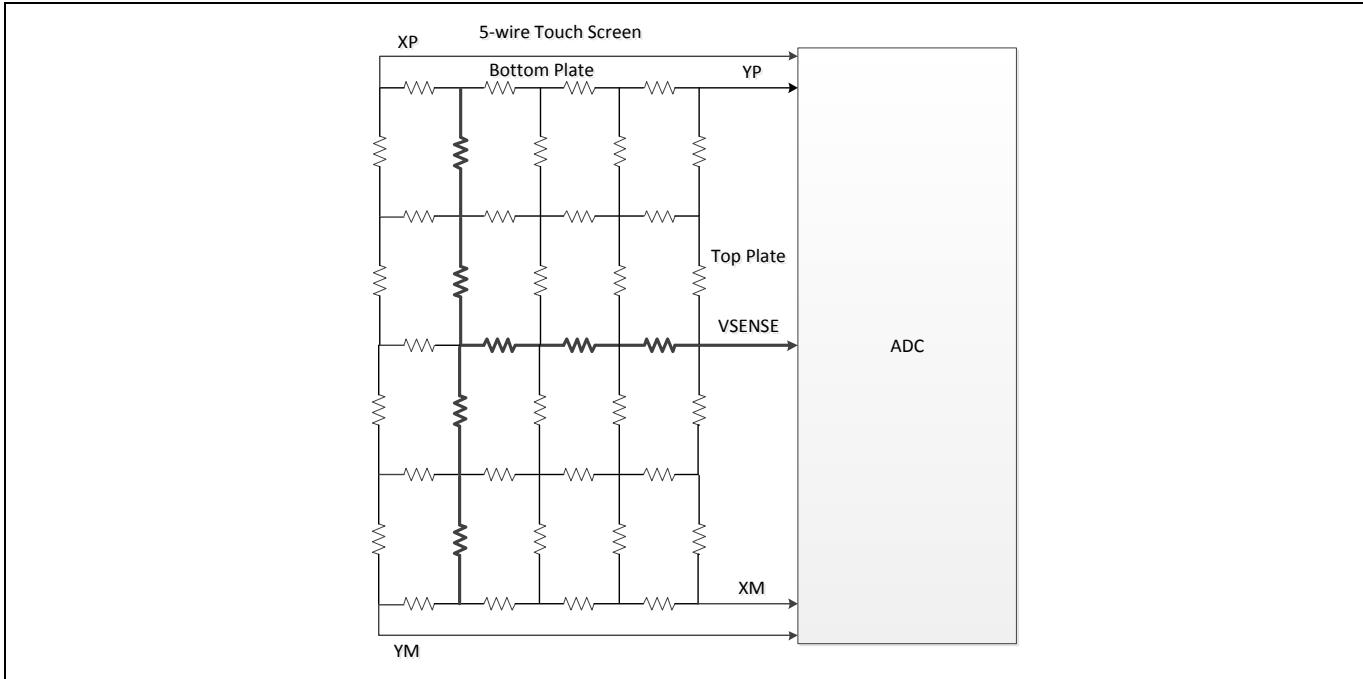
```
rREG_IER |= ADC_IER_KPUEIEN;  
while( !( rREG_ISR & ADC_ISR_KPUEF));  
rREG_ISR = ADC_ISR_KPUEF; //Clear KPUEF flag  
rREG_IER &= ~ADC_IER_KPUEIEN;  
}  
else  
    printf("interrupt error\n");  
}while(1);
```

3.5.6 4 線和 5 線觸摸屏(4-wire and 5-wire Touch Screen)

觸摸屏控制邏輯和開關可以控制的 4 線和五線式觸摸屏。對於 ADC 配置寄存器 T_EN(ADC_CONF[0])應設置為 1。ADC 控制寄存器 WMSWCH(ADCON[16])可以設定 5 線/4 線配置。下圖顯示了 4 線、5 線觸摸屏接口。

需要注意的是，四個開關以偏壓 XP，XM，YP，YM 具有下 5 欧姆傳導電阻。和上拉 PMOS 有 200K 欧姆。





示範一個4線觸摸屏的軟件程序，如下：

```
unsigned short x, y,i;
rREG_CTL |= ADC_CTL_ADEN ;
rREG_CONF |= ADC_CONF_TEN | ADC_CONF_DISTMAVEN;
rREG_IER |= ADC_IER_PDEIEN ;
rREG_ISR = ADC_ISR_TF | ADC_ISR_MF;
/* touch_xy_test interrupt mode */
rREG_IER |= ADC_IER_MIEN;

do{
    rREG_CTL |= ADC_CTL_MST;
    printf("Waiting for Interrupt\n");
    while(!(rREG_ISR & ADC_ISR_MF));
    rREG_ISR = ADC_ISR_MF; //Clear MF flag
    if(rREG_ISR & ADC_ISR_TF)
        printf("interrupt correct\n");
    else
        printf("interrupt error\n");
    rREG_ISR = ADC_ISR_TF; //Clear TF flag
    x = rREG_XYDATA & 0xFFFF;
    y = (rREG_XYDATA >> 16) & 0xFFFF;
    printf("x = %08x, y = %08x\n", x,y);
}while(1);
```

3.5.7 4 線壓力測量(4-wire Pressure Measurement)

為了區分筆或手指的觸摸，所以需要觸摸的壓力來分辨。ADC提供了兩種解決方案。第一種方法需要知道X-plate電阻和X-Position的測量，並在觸摸屏的兩個附加交叉面板測量(Z1和Z2)。使用下面的公式來計算觸摸電阻：

$$R_{\text{Touch}} = R_{\text{(X-plate)}} \times X_{\text{position}} / 4096 (Z_2/Z_1 - 1)$$

第二種方法需要知道X-plate和Y-plate的電阻，X-Position和Y-Position，以及Z1的測量。使用下面的公式來計算觸摸電阻：

$$R_{\text{Touch}} = (R_{\text{(X-plate)}} \times X_{\text{position}} / 4096) (4096/Z_1 - 1) - R_{\text{(Y-plate)}} (1 - Y_{\text{position}}/4096) \quad (1)$$

用於ADC的寄存器Z_EN(ADC_CONF[1])置為1，觸摸壓力測量將存儲在此ADC_ZDATA寄存器。

示範一個4線壓力測量的軟件程序，如下：

```
unsigned short x, y, z1, z2;
rREG_CTL |= ADC_CTL_ADEN ;
rREG_CONF |= ADC_CONF_TEN|ADC_CONF_DISTMAVEN|ADC_CONF_ZEN|ADC_CONF_DISZMAVEN;
rREG_ISR = ADC_ISR_MF | ADC_ISR_TF | ADC_ISR_ZF;
/* touch_xy_test interrupt mode */
rREG_IER |= ADC_IER_MIEN;
do{
    rREG_CTL |= ADC_CTL_MST;
    printf("Waiting for Interrupt\n");
    while(!(rREG_ISR & ADC_ISR_MF));
    rREG_ISR = ADC_ISR_MF; //Clear MF flag
    if((rREG_ISR&(ADC_ISR_TF|ADC_ISR_ZF))==(ADC_ISR_TF|ADC_ISR_ZF))
        printf("interrupt correct\n");
    else
        printf("interrupt error\n");
    x = rREG_XYDATA & 0FFF;
    y = (rREG_XYDATA >> 16) & 0FFF;
    z1 = rREG_ZDATA & 0FFF;
    z2 = (rREG_ZDATA >> 16) & 0FFF;
    printf("x = %d, y = %d, z1 = %d, z2 = %d\n", x,y,z1,z2);
}while(1);
```

4 中斷控制器 (AIC)

4.1 概述

中斷是臨時改變程序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，發送/從以太網MAC控制器接收請求，依此類推。該CPU處理器提供了兩種模式中斷，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。當前程序狀態寄存器(CPSR)的F和I bit時，它可以忽略FIQ和IRQ通過設置。

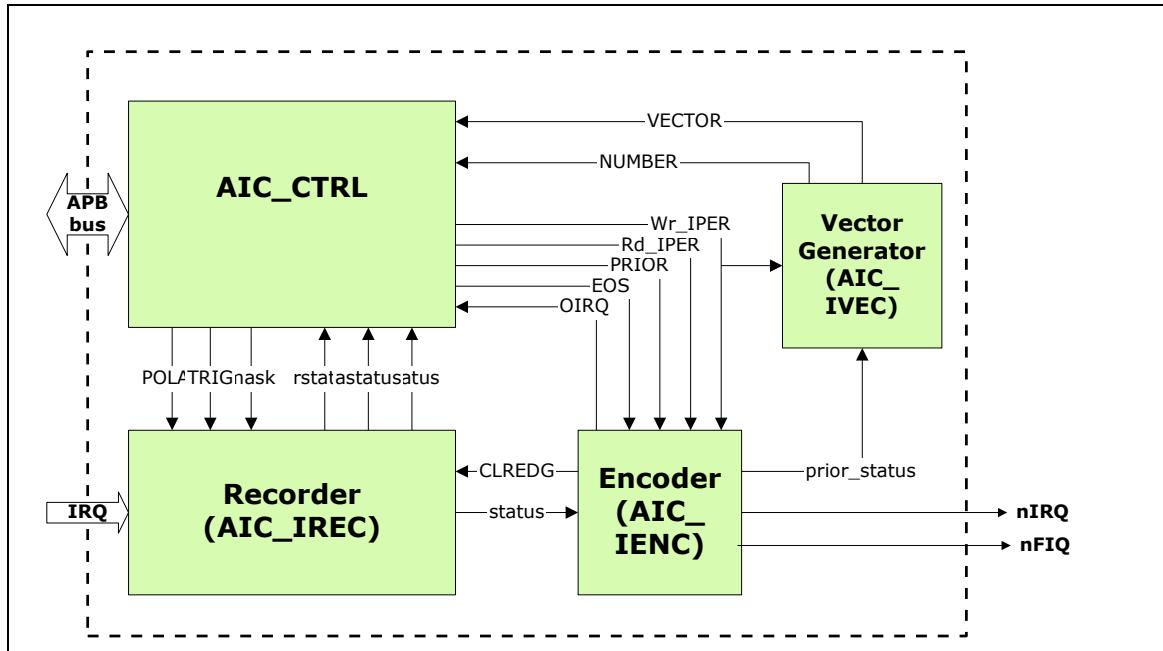
高級中斷控制器(AIC)能夠處理的中斷請求多達64個不同的中斷源。目前，61個中斷源被定義。每個中斷源被分配給一個中斷通道。例如，看門狗定時器中斷被分配到通道1。AIC實現了61個中斷源和8個優先級。優先級0中的中斷源為最高和優先級7的中斷源是最低的。為了使優先級正常工作，優先級必須指定給每個中斷源在上電時初始化;否則，系統會產生意外行為。每個優先級在一個較低的中斷通道具有更高的優先級。中斷源為最高優先級0會被晉升為FIQ。中斷源除0之外的優先級都會是IRQ。而IRQ可由FIQ的發生時被搶占。

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

4.2 特性

- AMBA APB 總線接口
- 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激活和高激活
- 利用標誌反映每個中斷源的狀態
- 每個中斷源都有獨立的屏蔽
- 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)
- 自動屏蔽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源被編程為邊沿觸發

4.3 方塊圖



4.4 寄存器

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

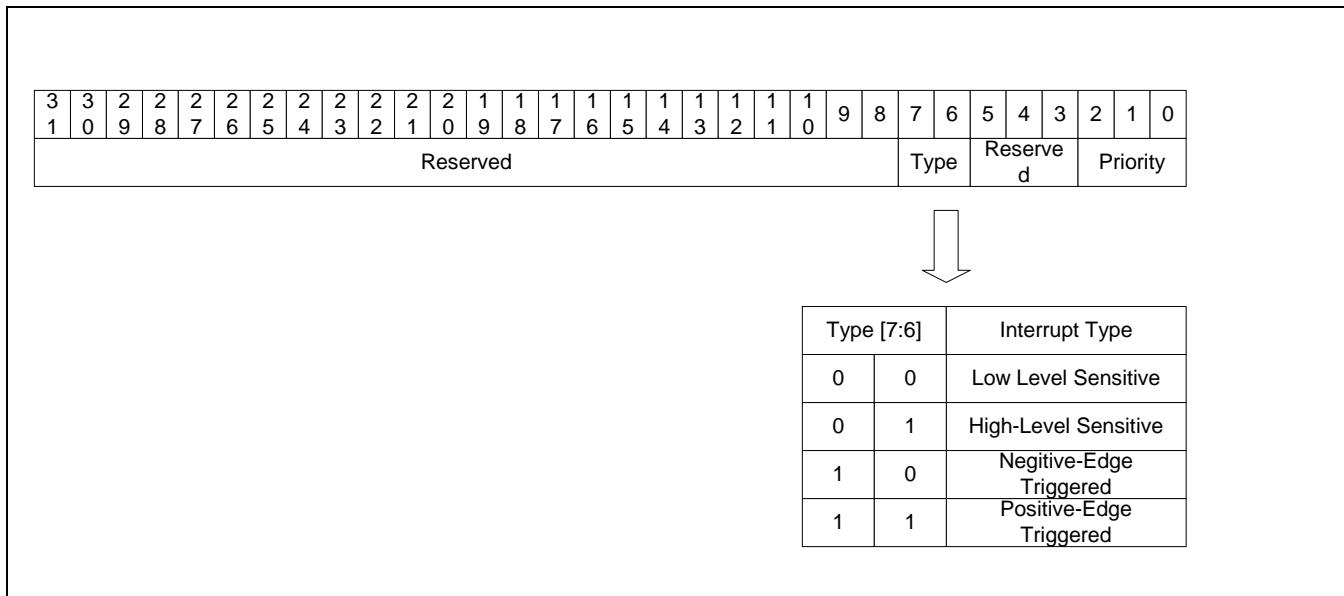
Register	Address	R/W	Description	Reset Value
AIC_BA = 0xB800_2000				
AIC_SCR1	AIC_BA+0x000	R/W	Source Control Register 1	0x4747_4747
AIC_SCR2	AIC_BA+0x004	R/W	Source Control Register 2	0x4747_4747
AIC_SCR3	AIC_BA+0x008	R/W	Source Control Register 3	0x4747_4747
AIC_SCR4	AIC_BA+0x00C	R/W	Source Control Register 4	0x4747_4747
AIC_SCR5	AIC_BA+0x010	R/W	Source Control Register 5	0x4747_4747
AIC_SCR6	AIC_BA+0x014	R/W	Source Control Register 6	0x4747_4747
AIC_SCR7	AIC_BA+0x018	R/W	Source Control Register 7	0x4747_4747
AIC_SCR8	AIC_BA+0x01C	R/W	Source Control Register 8	0x4747_4747
AIC_SCR9	AIC_BA+0x020	R/W	Source Control Register 9	0x4747_4747
AIC_SCR10	AIC_BA+0x024	R/W	Source Control Register 10	0x4747_4747
AIC_SCR11	AIC_BA+0x028	R/W	Source Control Register 11	0x4747_4747
AIC_SCR12	AIC_BA+0x02C	R/W	Source Control Register 12	0x4747_4747
AIC_SCR13	AIC_BA+0x030	R/W	Source Control Register 13	0x4747_4747
AIC_SCR14	AIC_BA+0x034	R/W	Source Control Register 14	0x4747_4747
AIC_SCR15	AIC_BA+0x038	R/W	Source Control Register 15	0x4747_4747
AIC_SCR16	AIC_BA+0x03C	R/W	Source Control Register 16	0x0000_0047

AIC_IRSR	AIC_BA+0x100	R	Interrupt Raw Status Register	0x0000_0000
AIC_IRSRH	AIC_BA+0x104	R	Interrupt Raw Status Register (High)	0x0000_0000
AIC_IASR	AIC_BA+0x108	R	Interrupt Active Status Register	0x0000_0000
AIC_IASRH	AIC_BA+0x10C	R	Interrupt Active Status Register (High)	0x0000_0000
AIC_ISR	AIC_BA+0x110	R	Interrupt Status Register	0x0000_0000
AIC_ISRH	AIC_BA+0x114	R	Interrupt Status Register (High)	0x0000_0000
AIC_IPER	AIC_BA+0x118	R	Interrupt Priority Encoding Register	0x0000_0000
AIC_ISNR	AIC_BA+0x120	R	Interrupt Source Number Register	0x0000_0000
AIC_OISR	AIC_BA+0x124	R	Output Interrupt Status Register	0x0000_0000
AIC_IMR	AIC_BA+0x128	R	Interrupt Mask Register	0x0000_0000
AIC_IMRH	AIC_BA+0x12C	R	Interrupt Mask Register (High)	0x0000_0000
AIC_MECR	AIC_BA+0x130	W	Mask Enable Command Register	Undefined
AIC_MECH	AIC_BA+0x134	W	Mask Enable Command Register (High)	Undefined
AIC_MDCR	AIC_BA+0x138	W	Mask Disable Command Register	Undefined
AIC_MDCRH	AIC_BA+0x13C	W	Mask Disable Command Register (High)	Undefined
AIC_SSCR	AIC_BA+0x140	W	Source Set Command Register	Undefined
AIC_SSCRH	AIC_BA+0x144	W	Source Set Command Register (High)	Undefined
AIC_SCCR	AIC_BA+0x148	W	Source Clear Command Register	Undefined
AIC_SCCRH	AIC_BA+0x14C	W	Source Clear Command Register (High)	Undefined
AIC_EOSCR	AIC_BA+0x150	W	End of Service Command Register	Undefined

4.5 功能描述.

4.5.1 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NUC970系列MCU內部中斷類型都是正電平觸發，這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_SRCx)的內容。



4.5.2 中斷屏蔽

NUC970 系列 MCU AIC 提供一組寄存器來遮蔽每個中斷通道。屏蔽啟用命令寄存器 (AIC_MEGR)。寫 1 到 MECR 將使相對應的中斷通道啟用中斷。相反，屏蔽禁用命令寄存器 (AIC_MDCR)。寫 1 到 MDCR 會禁止相對應的中斷通道禁用中斷。寫 0 到 AIC_MEGR 或 AIC_MDCR 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屏蔽寄存器 (AIC_IMR) 知道中斷通道是否被啟用或禁用。如果中斷通道被啟用，則相應的元能讀取到 1，否則為 0。

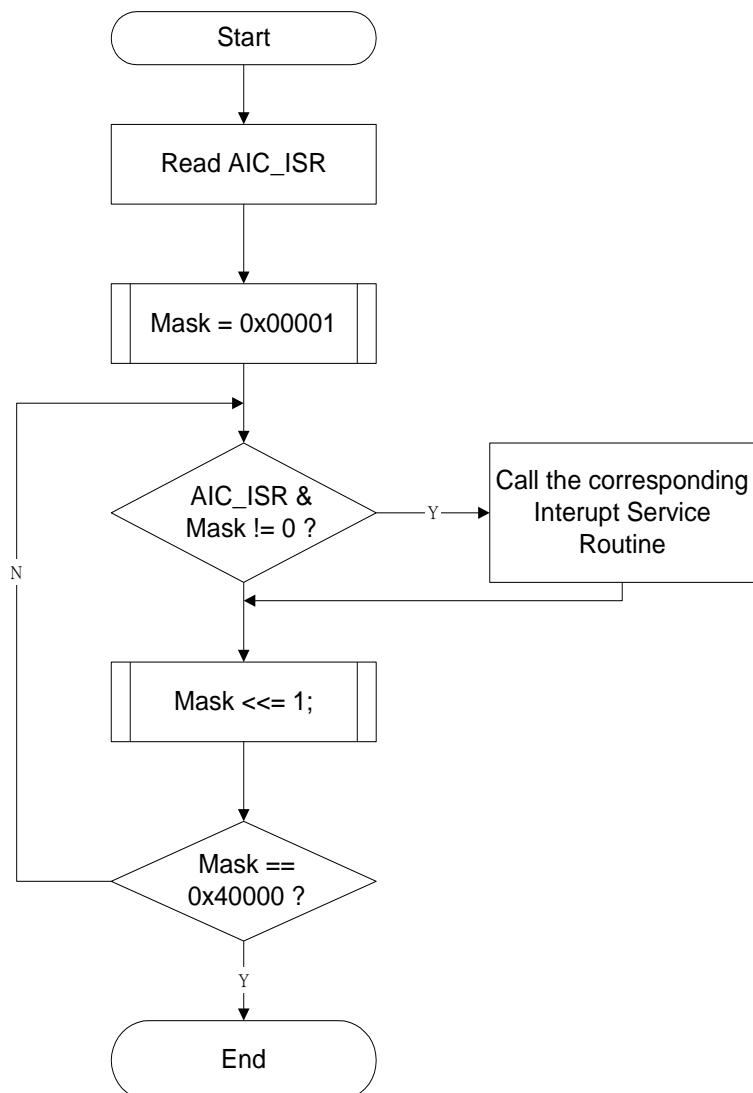
4.5.3 中斷清除與設置

所有的中斷源(包括FIQ)可單獨設置，或分別寫入寄存器AIC_SSCR和AIC_SCCR，當他們被編程為邊沿觸發。在AIC的這個功能在自動測試或軟件調試。

4.5.4 軟件優先規則

AIC 提供了一個中斷狀態寄存器 (AIC_ISR) 來識別中斷源。如果中斷通道激活或啟動時，在 AIC_ISR 其相應的位被設置為 1 FIQ 或 IRQ 的中斷處理程序可以通過讀取 AIC_ISR 得到中斷源。與該服務序列是完全由軟件算法決定。

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序。當中斷是由 CPU 內核的認可，FIQ 或 IRQ 異常處理程序首先執行。然後，它會根據 AIC_ISR 內容調用適當的中斷服務程序。下圖展示了順序優先級方案，其中通道 1 具有最高的優先級和通道 17~18 具有低的優先級。



```
__irq void sysIrqHandler()
{
    UINT32 volatile _mISR, _mISRH, i;

    _mISR = inpw(REG_AIC_ISR);
    _mISRH = inpw(REG_AIC_ISRH);

    for (i = 1; i <= 31; i++)
        if (_mISR & (1 << i))
            (*sysIrqHandlerTable[i])();

    for (i = 32; i <= WB_MAX_INT_SOURCE; i++)

```

```
if (_mISRH & (1 << (i-32)))
    (*sysIrqHandlerTable[i])();
}
```

4.5.5 硬件優先規則

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編程 AIC_SCRx。同樣，FIQ或IRQ異常處理程序時首先執行中斷確認。異常處理程序和中斷服務程序應遵循一定的規則，讓這個機制工作正常。規則在下面列出。

1. 讀取IRQ優先權編碼寄存器(AIC_IPER)來獲得向量(IRQ通道x4)，並在同時AIC_ISNR將由當前中斷通道號碼被加載，所述向量(IRQ信道號x4)表示中斷通道號碼也就是激活狀態和使能狀態，並且具有最高優先級，再乘以4，然後將其存儲在AIC_IPER。從AIC_IPER得到的數據為便於下面的中斷服務路由地址計算完成後，啟用並且具有最高優先級。
2. 增加中斷服務程序表來分配相對應中斷服務程序。
3. 寫任何值到AIC_EOSCR寄存器，來完成中斷。

當中斷通道是激活時，將被視為當前的優先級，將推入到優先級編碼器中，在同一時間 AIC_IPER被讀取。AIC_ISNR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。因此，當服務命令寄存器結束(AIC_EOSCR)寫入，當前中斷級別與從堆棧(如果有的話)的最後一個存儲中斷級別更新。因此，在具有較高優先級的中斷結束時，AIC返回到對應於前一較低優先級的中斷已被中斷之前的狀態。這種硬件優先級控制是很有幫助的實施嵌套中斷系統。

```
_irq void sysIrqHandler()
{
    UINT32 volatile _mISNR;

    _mISNR = inpw(REG_AIC_ISNR);
    (*sysIrqHandlerTable[_mISNR])();
    outpw(REG_AIC_EOSCR, 1);
}
```

4.5.6 中斷源

以下為 NUC970 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	External Interrupt 0	Positive Level	External Interrupt 0
5	External Interrupt 1	Positive Level	External Interrupt 1
6	External Interrupt 2	Positive Level	External Interrupt 2
7	External Interrupt 3	Positive Level	External Interrupt 3
8	External Interrupt 4	Positive Level	External Interrupt 4
9	External Interrupt 5	Positive Level	External Interrupt 5
10	External Interrupt 6	Positive Level	External Interrupt 6
11	External Interrupt 7	Positive Level	External Interrupt 7
12	ACTL_INT	Positive Level	Audio Controller Interrupt
13	LCD_INT	Positive Level	LCD Controller Interrupt
14	CAP_INT	Positive Level	Sensor Interface Controller Interrupt
15	RTC_INT	Positive Level	RTC Interrupt
16	TMR0_INT	Positive Level	Timer 0 Interrupt
17	TMR1_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	GDMA0_INT	Positive Level	GDMA Channel 0 Interrupt
26	GDMA1_INT	Positive Level	GDMA Channel 1 Interrupt
27	SDH_INT	Positive Level	SD/SDIO Host Interrupt
28	SIC_INT	Positive Level	SIC Interrupt
29	UDC_INT	Positive Level	USB Device Controller Interrupt
30	TMR2_INT	Positive Level	Timer 2 Interrupt
31	TMR3_INT	Positive Level	Timer 3 Interrupt
32	TMR4_INT	Positive Level	Timer 4 Interrupt
33	JPEG_INT	Positive Level	JPEG Engine Interrupt
34	GE2D_INT	Positive Level	2D Graphic Engine 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	UART10_INT	Positive Level	UART 10 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	ETMR0_INT	Positive Level	Enhanced Timer 0 Interrupt
48	ETMR1_INT	Positive Level	Enhanced Timer 1 Interrupt
49	ETMR2_INT	Positive Level	Enhanced Timer 2 Interrupt
50	ETMR3_INT	Positive Level	Enhanced Timer 3 Interrupt
51	USI0_INT	Positive Level	USI 0 Interrupt
52	USI1_INT	Positive Level	USI 1 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	GPIO_INT	Positive Level	GPIO Interrupt
58	CAN0_INT	Positive Level	CAN 0 Interrupt
59	CAN1_INT	Positive Level	CAN 1 Interrupt
60	PWM_INT	Positive Level	PWM Interrupt
61	KPI_INT	Positive Level	KPI Interrupt

5 CAN

5.1 概述

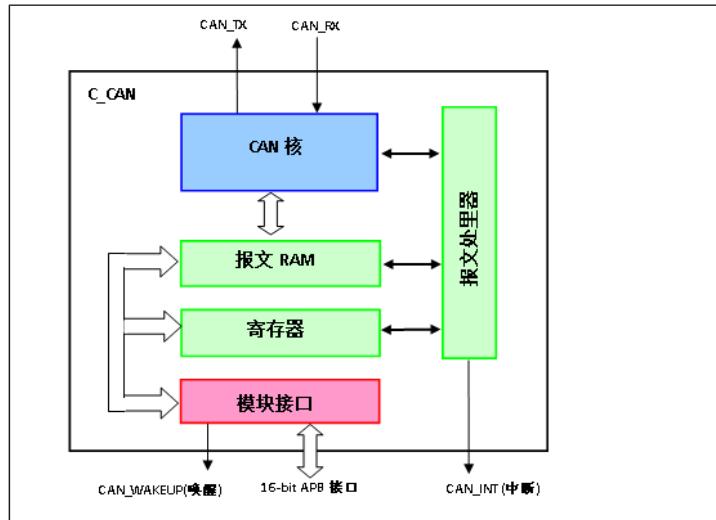
控制區域網路CAN(Controller Area Network)，首先由Robert Bosch公司提出，常用於汽車與航太工業，是以種差動式序列傳輸方式，並將車上多個控制器整合在區域網路中，藉由各個MCU分擔或分享資訊，如此便能大幅減少電線的使用量，最快的傳送速率為1Mbps。

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 。

5.2 特性

- 支持 2 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

5.3 方塊圖



上圖為NuMicro MCU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。

5.4 寄存器

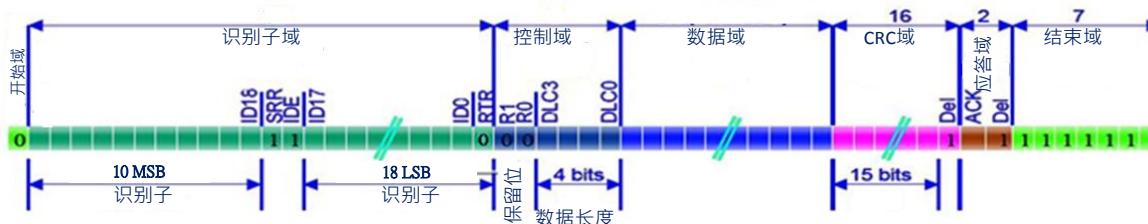
Register	Offset	R/W	Description	Reset Value
CAN0_BA = 0xB800_0000				
CAN1_BA = 0xB800_4000				
CAN_CON	CANx_BA+0x00	R/W	Control Register	0x0000_0001
CAN_STATUS	CANx_BA+0x04	R/W	Status Register	0x0000_0000
CAN_ERR	CANx_BA+0x08	R	Error Counter	0x0000_0000
CAN_BTIME	CANx_BA+0x0C	R/W	Bit Timing Register	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	CANx_BA+0x20	R/W	IFn (*2) Command Request Registers	0x0000_0001
CAN_IF2_CREQ	CANx_BA+0x80			
CAN_IF1_CMASK	CANx_BA+0x24	R/W	IFn Command Mask Registers	0x0000_0000
CAN_IF2_CMASK	CANx_BA+0x84			
CAN_IF1_MASK1	CANx_BA+0x28	R/W	IFn Mask 1 Register	0x0000_FFFF
CAN_IF2_MASK1	CANx_BA+0x88			
CAN_IF1_MASK2	CANx_BA+0x2C	R/W	IFn Mask 2 Register	0x0000_FFFF
CAN_IF2_MASK2	CANx_BA+0x8C			
CAN_IF1_ARB1	CANx_BA+0x30	R/W	IFn Arbitration 1 Register	0x0000_0000
CAN_IF2_ARB1	CANx_BA+0x90			

CAN_IF1_ARB2	CANx_BA+0x34	R/W	IFn Arbitration 2 Register	0x0000_0000
CAN_IF2_ARB2	CANx_BA+0x94			
CAN_IF1_MCON	CANx_BA+0x38	R/W	IFn Message Control Registers	0x0000_0000
CAN_IF2_MCON	CANx_BA+0x98			
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	CANx_BA+0x100	R	Transmission Request Registers 1 & 2	0x0000_0000
CAN_TXREQ2	CANx_BA+0x104			
CAN_NDAT1	CANx_BA+0x120	R	New Data Registers 1 & 2	0x0000_0000
CAN_NDAT2	CANx_BA+0x124			
CAN_IPND1	CANx_BA+0x140	R	Interrupt Pending Registers 1 & 2	0x0000_0000
CAN_IPND2	CANx_BA+0x144			
CAN_MVLD1	CANx_BA+0x160	R	Message Valid Registers 1 & 2	0x0000_0000
CAN_MVLD2	CANx_BA+0x164			
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

5.5 功能描述

5.5.1 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：

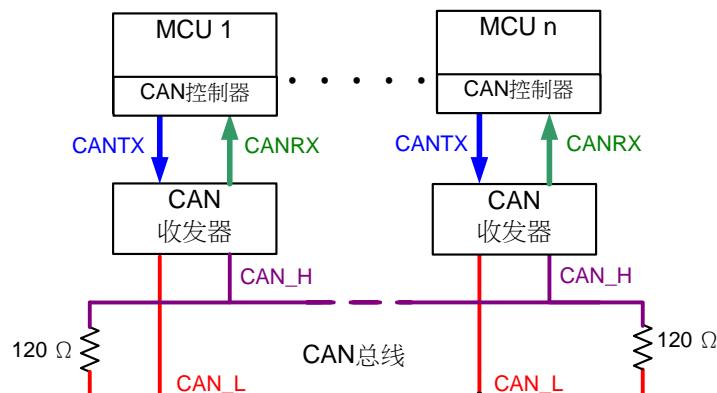
- 起始域 (SOF, Start of Frame) 一幀的開頭
- 仲裁域 (Arbitration Field) 用以確定該幀的優先順序
- 控制域 (Control Field) 包含著兩個保留位元和資料欄內數據的長度
- 資料欄 (Data Field) 0~8 個位元組的被傳送資料
- CRC域(CRC Field) 從 SOF 到 Data Field 的 CRC 值，用於檢測傳輸錯誤

- 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

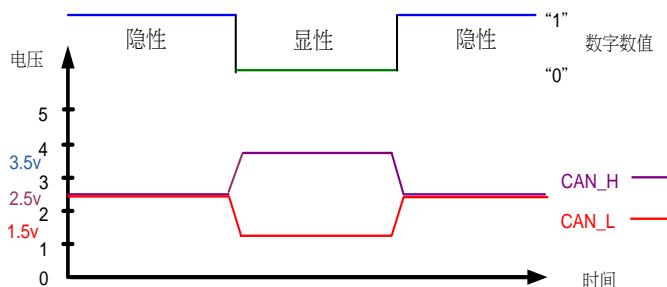
對於CAN協定的詳細內容，用於可以參考BOSCH公司的CAN技術檔。

5.5.2 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 5-1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 $120\ \Omega$ 電阻。由於在 NUC970 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。



5.5.3 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： $f_{speed} = 1/t_{NBT}$

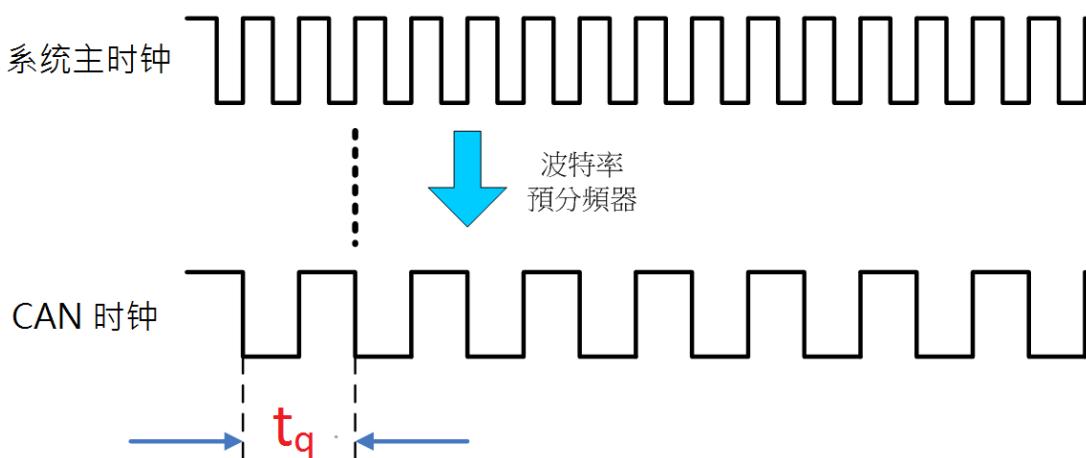
其中 t_{NBT} 為位時間。位時間可被分為四個不重疊的時間段，包含同步段、延遲時間段、相位緩衝段1和相位緩衝段2，詳細定義如下：

- 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 5-2 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

$$TSEG1 + 1 = (t_{PROP_SEG} + t_{PHASE_SEG1})t_q$$

$$TSEG2 + 1 = (t_{PHASE_SEG2})t_q$$

$$t_{SYNC_SEG} = 1 t_q$$

其中，TSEG1 和 TSEG2 是 CAN_BTIME 寄存器中的控制位。

綜上可得：

$$\begin{aligned} f_{speed} &= 1/t_{NBT} = 1/(t_{SYNC_SEG} + t_{PROP_SEG} + t_{PHASE_SEG1} + t_{PHASE_SEG2}) \\ &= 1/(1 + (TSEG1 + 1) + (TSEG2 + 1))t_q \end{aligned}$$

$$= 1/(TSEG1 + TSEG2 + 3) \left((BPR + 1) / f_{APB_CLK} \right)$$

$$= f_{APB_CLK} / (TSEG1 + TSEG2 + 3) (BPR + 1)$$

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

例如，欲設定 CAN 總線速度為 1000 kbps，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。

5.5.4 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。

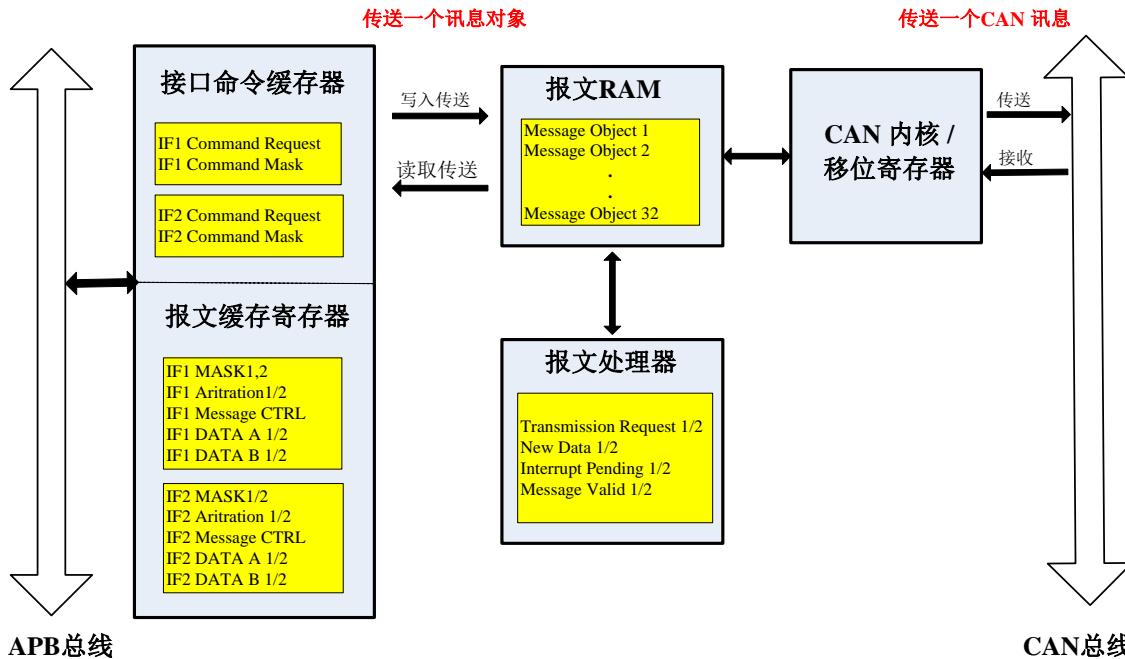
● 報文介面寄存器集合

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。

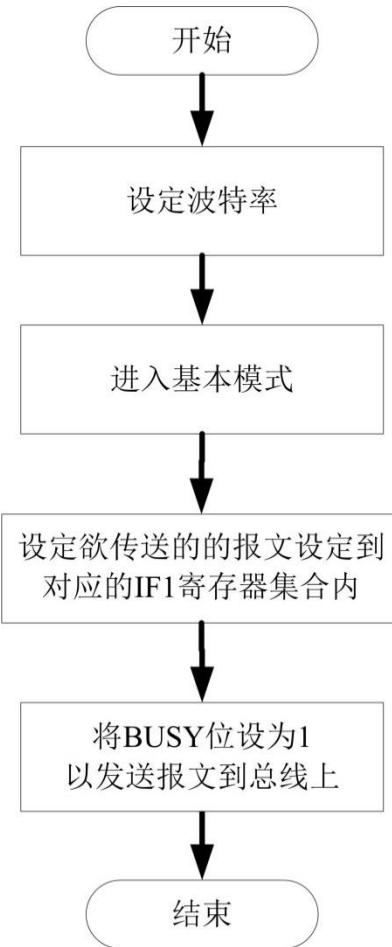
首先，使用者需要通過介面命令寄存器，來設定或取得報文資訊到報文RAM中，並通過介面命令寄存器傳送命令給CAN，報文緩存寄存器用以儲存所需要的命令資料。

5.5.5 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。

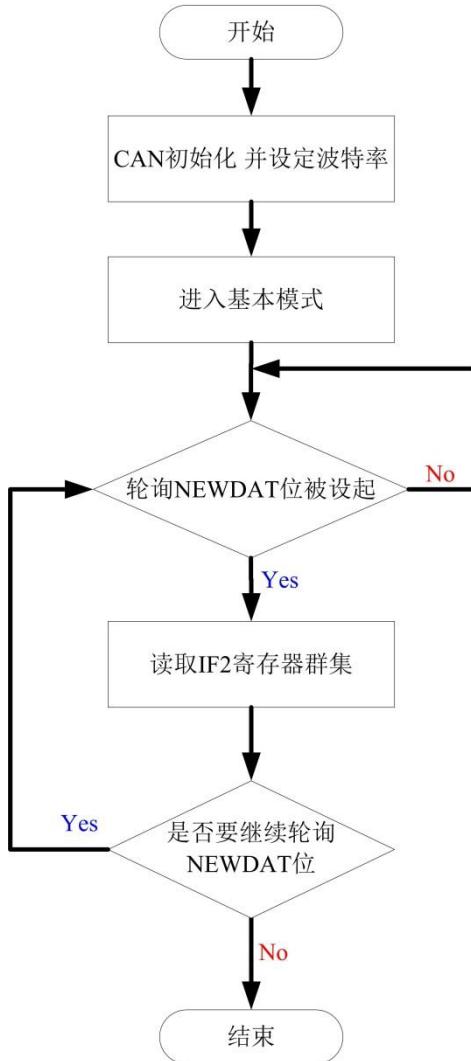
1. 參考 5.5.3 小節，完成 CAN 串列傳輸速率的設定。
2. 接著允許 CAN_CON 寄存器中的 TEST 位、CAN_TEST 寄存器中的 BASIC 位，即可讓 CAN 控制器進入基本模式。
3. 根據 CAN 報文格式，在 IF1 寄存器中填入對應的值。
4. 將 CAN_CREQ 寄存器中的 BUSY 置 “1” ，即可發送根據 IF1 寄存器集合中的 CAN 報文 資訊，待發送完畢後 BUSY 位會自動清除。

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 汇流排上。

5.5.6 接收 CAN 報文

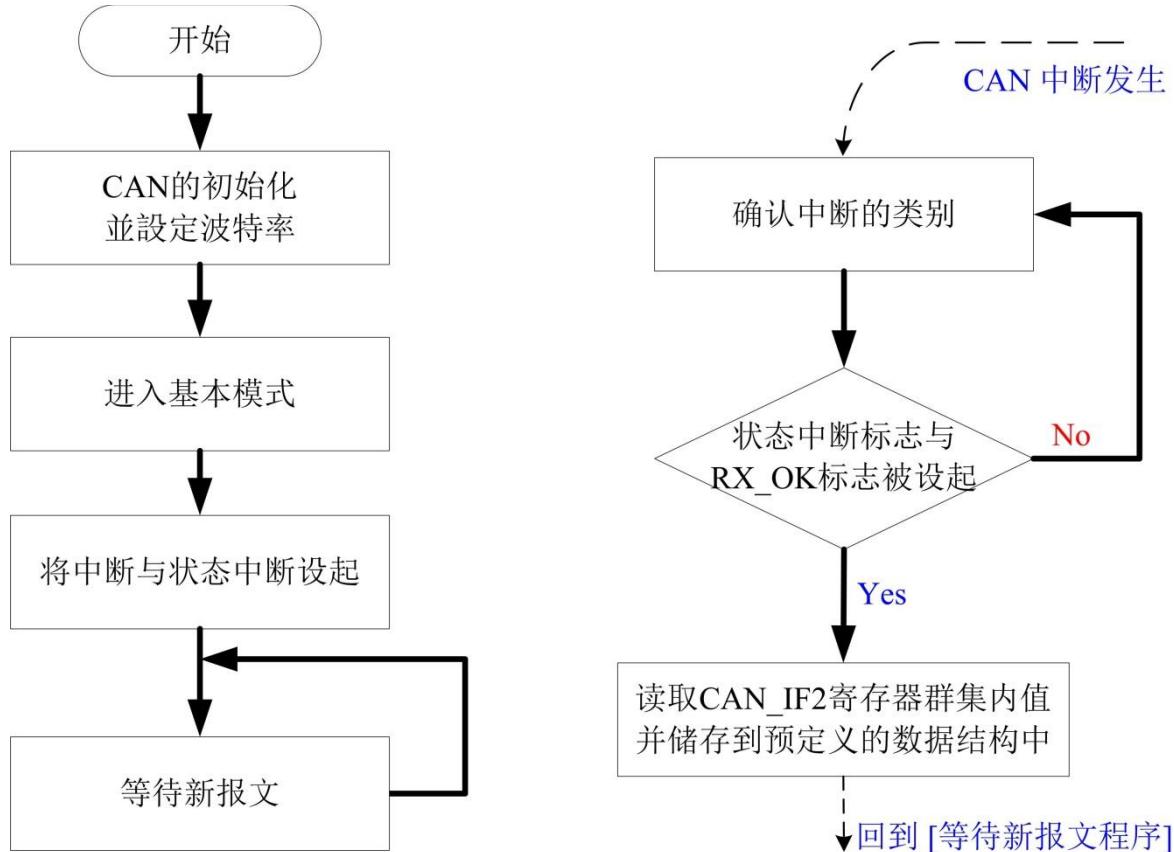
接收報文有兩種方式，一種是使用輪詢方式確認NEWDAT位元是否被置位，另一種則是利用RxOK的狀態中斷，兩者皆可確認是否已接收到新報文，新報文會被儲存到CAN_IF2寄存器集合中。下圖是採用輪詢方式接收CAN匯流排上報文的流程。



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 5.5.3 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。

- 首先，先對 CAN 做初始化，並請參考 5.5.3 小節完成 CAN 串列傳輸速率設定。
- 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。
- 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。
- 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。

5.5.7 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

6 加密加速器

6.1 概述

NUC970 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, DES / TDES 加密加速器, SHA 和 HMAC 運算加速器。

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。

DES / TDES 加速器是完全符合 DES 和三重DES 加密／解密算法。DES / TDES 加速器支持 ECB , CBC , CFB , OFB 和 CTR 等加密模式。

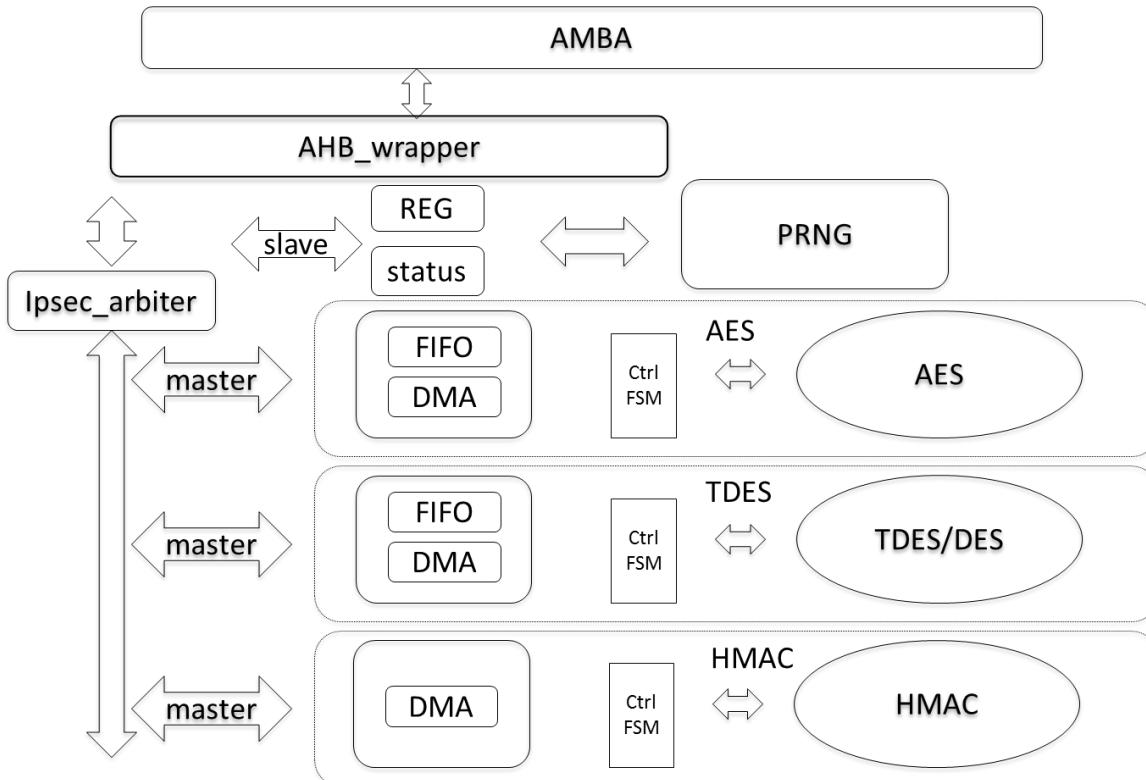
SHA / HMAC 加速器完全符合 SHA-160 , SHA-224 , SHA-256 , SHA-384 和 SHA-512 和相應的 HMAC 算法。

6.2 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) 。
- DES
 - 支持 FIPS 46-3 標準規範。
 - 支持加密和解密。
 - 支持 ECB , CBC , CFB , OFB , 和 CTR 模式。
- TDES
 - 支持 FIPS NIST 800-67 標準規範。
 - 符合 X9.52 標準規範。
 - 支持雙密鑰及三密鑰模式。
 - 支持加密和解密。
 - 支持 ECB , CBC , CFB , OFB , 和 CTR 模式。
- 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。

6.3 方塊圖



6.4 寄存器

Register	Offset	R/W	Description	Reset Value
CRYPTO Base Address:				
CRYP_BA	= 0xB000_C000			
CRPT_INTEN	CRYP_BA+0x000	R/W	Crypto Interrupt Enable Control Register	0x0000_0000
CRPT_INTSTS	CRYP_BA+0x004	R/W	Crypto Interrupt Flag	0x0000_0000
CRPT_PRNG_CTL	CRYP_BA+0x008	R/W	PRNG Control Register	0x0000_0000
CRPT_PRNG_SEED	CRYP_BA+0x00C	W	Seed for PRNG	Undefined

CRPT_PRNG_KEY0	CRYP_BA+0x010	R	PRNG Generated Key0	Undefined
CRPT_PRNG_KEY1	CRYP_BA+0x014	R	PRNG Generated Key1	Undefined
CRPT_PRNG_KEY2	CRYP_BA+0x018	R	PRNG Generated Key2	Undefined
CRPT_PRNG_KEY3	CRYP_BA+0x01C	R	PRNG Generated Key3	Undefined
CRPT_PRNG_KEY4	CRYP_BA+0x020	R	PRNG Generated Key4	Undefined
CRPT_PRNG_KEY5	CRYP_BA+0x024	R	PRNG Generated Key5	Undefined
CRPT_PRNG_KEY6	CRYP_BA+0x028	R	PRNG Generated Key6	Undefined
CRPT_PRNG_KEY7	CRYP_BA+0x02C	R	PRNG Generated Key7	Undefined
CRPT_AES_FDBCK0	CRYP_BA+0x050	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000
CRPT_AES_FDBCK1	CRYP_BA+0x054	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000
CRPT_AES_FDBCK2	CRYP_BA+0x058	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000
CRPT_AES_FDBCK3	CRYP_BA+0x05C	R	AES Engine Output Feedback Data after Cryptographic Operation	0x0000_0000
CRPT_TDES_FDBCKH	CRYP_BA+0x060	R	TDES/DES Engine Output Feedback High Word Data after Cryptographic Operation	0x0000_0000
CRPT_TDES_FDBCKL	CRYP_BA+0x064	R	TDES/DES Engine Output Feedback Low Word Data after Cryptographic Operation	0x0000_0000
CRPT_AES_CTL	CRYP_BA+0x100	R/W	AES Control Register	0x0000_0000
CRPT_AES_STS	CRYP_BA+0x104	R	AES Engine Flag	0x0001_0100
CRPT_AES_DATIN	CRYP_BA+0x108	R/W	AES Engine Data Input Port Register	0x0000_0000
CRPT_AES_DATOUT	CRYP_BA+0x10C	R	AES Engine Data Output Port Register	0x0000_0000
CRPT_AES0_KEY0	CRYP_BA+0x110	R/W	AES Key Word 0 Register for Channel 0	0x0000_0000
CRPT_AES0_KEY1	CRYP_BA+0x114	R/W	AES Key Word 1 Register for Channel 0	0x0000_0000
CRPT_AES0_KEY2	CRYP_BA+0x118	R/W	AES Key Word 2 Register for Channel 0	0x0000_0000
CRPT_AES0_KEY3	CRYP_BA+0x11C	R/W	AES Key Word 3 Register for Channel 0	0x0000_0000
CRPT_AES0_KEY4	CRYP_BA+0x120	R/W	AES Key Word 4 Register for Channel 0	0x0000_0000
CRPT_AES0_KEY5	CRYP_BA+0x124	R/W	AES Key Word 5 Register for Channel 0	0x0000_0000
CRPT_AES0_KEY6	CRYP_BA+0x128	R/W	AES Key Word 6 Register for Channel 0	0x0000_0000
CRPT_AES0_KEY7	CRYP_BA+0x12C	R/W	AES Key Word 7 Register for Channel 0	0x0000_0000

CRPT_AES0_IV0	CRYP_BA+0x130	R/W	AES Initial Vector Word 0 Register for Channel 0	0x0000_0000
CRPT_AES0_IV1	CRYP_BA+0x134	R/W	AES Initial Vector Word 1 Register for Channel 0	0x0000_0000
CRPT_AES0_IV2	CRYP_BA+0x138	R/W	AES Initial Vector Word 2 Register for Channel 0	0x0000_0000
CRPT_AES0_IV3	CRYP_BA+0x13C	R/W	AES Initial Vector Word 3 Register for Channel 0	0x0000_0000
CRPT_AES0_SADDR	CRYP_BA+0x140	R/W	AES DMA Source Address Register for Channel 0	0x0000_0000
CRPT_AES0_DADDR	CRYP_BA+0x144	R/W	AES DMA Destination Address Register for Channel 0	0x0000_0000
CRPT_AES0_CNT	CRYP_BA+0x148	R/W	AES Byte Count Register for Channel 0	0x0000_0000
CRPT_AES1_KEY0	CRYP_BA+0x14C	R/W	AES Key Word 0 Register for Channel 1	0x0000_0000
CRPT_AES1_KEY1	CRYP_BA+0x150	R/W	AES Key Word 1 Register for Channel 1	0x0000_0000
CRPT_AES1_KEY2	CRYP_BA+0x154	R/W	AES Key Word 2 Register for Channel 1	0x0000_0000
CRPT_AES1_KEY3	CRYP_BA+0x158	R/W	AES Key Word 3 Register for Channel 1	0x0000_0000
CRPT_AES1_KEY4	CRYP_BA+0x15C	R/W	AES Key Word 4 Register for Channel 1	0x0000_0000
CRPT_AES1_KEY5	CRYP_BA+0x160	R/W	AES Key Word 5 Register for Channel 1	0x0000_0000
CRPT_AES1_KEY6	CRYP_BA+0x164	R/W	AES Key Word 6 Register for Channel 1	0x0000_0000
CRPT_AES1_KEY7	CRYP_BA+0x168	R/W	AES Key Word 7 Register for Channel 1	0x0000_0000
CRPT_AES1_IV0	CRYP_BA+0x16C	R/W	AES Initial Vector Word 0 Register for Channel 1	0x0000_0000
CRPT_AES1_IV1	CRYP_BA+0x170	R/W	AES Initial Vector Word 1 Register for Channel 1	0x0000_0000
CRPT_AES1_IV2	CRYP_BA+0x174	R/W	AES Initial Vector Word 2 Register for Channel 1	0x0000_0000
CRPT_AES1_IV3	CRYP_BA+0x178	R/W	AES Initial Vector Word 3 Register for Channel 1	0x0000_0000
CRPT_AES1_SADDR	CRYP_BA+0x17C	R/W	AES DMA Source Address Register for Channel 1	0x0000_0000
CRPT_AES1_DADDR	CRYP_BA+0x180	R/W	AES DMA Destination Address Register for Channel 1	0x0000_0000
CRPT_AES1_CNT	CRYP_BA+0x184	R/W	AES Byte Count Register for Channel 1	0x0000_0000
CRPT_AES2_KEY0	CRYP_BA+0x188	R/W	AES Key Word 0 Register for Channel 2	0x0000_0000
CRPT_AES2_KEY1	CRYP_BA+0x18C	R/W	AES Key Word 1 Register for Channel 2	0x0000_0000
CRPT_AES2_KEY2	CRYP_BA+0x190	R/W	AES Key Word 2 Register for Channel 2	0x0000_0000
CRPT_AES2_KEY3	CRYP_BA+0x194	R/W	AES Key Word 3 Register for Channel 2	0x0000_0000
CRPT_AES2_KEY4	CRYP_BA+0x198	R/W	AES Key Word 4 Register for Channel 2	0x0000_0000
CRPT_AES2_KEY5	CRYP_BA+0x19C	R/W	AES Key Word 5 Register for Channel 2	0x0000_0000
CRPT_AES2_KEY6	CRYP_BA+0x1A0	R/W	AES Key Word 6 Register for Channel 2	0x0000_0000

CRPT_AES2_KEY7	CRYP_BA+0x1A4	R/W	AES Key Word 7 Register for Channel 2	0x0000_0000
CRPT_AES2_IV0	CRYP_BA+0x1A8	R/W	AES Initial Vector Word 0 Register for Channel 2	0x0000_0000
CRPT_AES2_IV1	CRYP_BA+0x1AC	R/W	AES Initial Vector Word 1 Register for Channel 2	0x0000_0000
CRPT_AES2_IV2	CRYP_BA+0x1B0	R/W	AES Initial Vector Word 2 Register for Channel 2	0x0000_0000
CRPT_AES2_IV3	CRYP_BA+0x1B4	R/W	AES Initial Vector Word 3 Register for Channel 2	0x0000_0000
CRPT_AES2_SADDR	CRYP_BA+0x1B8	R/W	AES DMA Source Address Register for Channel 2	0x0000_0000
CRPT_AES2_DADDR	CRYP_BA+0x1BC	R/W	AES DMA Destination Address Register for Channel 2	0x0000_0000
CRPT_AES2_CNT	CRYP_BA+0x1C0	R/W	AES Byte Count Register for Channel 2	0x0000_0000
CRPT_AES3_KEY0	CRYP_BA+0x1C4	R/W	AES Key Word 0 Register for Channel 3	0x0000_0000
CRPT_AES3_KEY1	CRYP_BA+0x1C8	R/W	AES Key Word 1 Register for Channel 3	0x0000_0000
CRPT_AES3_KEY2	CRYP_BA+0x1CC	R/W	AES Key Word 2 Register for Channel 3	0x0000_0000
CRPT_AES3_KEY3	CRYP_BA+0x1D0	R/W	AES Key Word 3 Register for Channel 3	0x0000_0000
CRPT_AES3_KEY4	CRYP_BA+0x1D4	R/W	AES Key Word 4 Register for Channel 3	0x0000_0000
CRPT_AES3_KEY5	CRYP_BA+0x1D8	R/W	AES Key Word 5 Register for Channel 3	0x0000_0000
CRPT_AES3_KEY6	CRYP_BA+0x1DC	R/W	AES Key Word 6 Register for Channel 3	0x0000_0000
CRPT_AES3_KEY7	CRYP_BA+0x1E0	R/W	AES Key Word 7 Register for Channel 3	0x0000_0000
CRPT_AES3_IV0	CRYP_BA+0x1E4	R/W	AES Initial Vector Word 0 Register for Channel 3	0x0000_0000
CRPT_AES3_IV1	CRYP_BA+0x1E8	R/W	AES Initial Vector Word 1 Register for Channel 3	0x0000_0000
CRPT_AES3_IV2	CRYP_BA+0x1EC	R/W	AES Initial Vector Word 2 Register for Channel 3	0x0000_0000
CRPT_AES3_IV3	CRYP_BA+0x1F0	R/W	AES Initial Vector Word 3 Register for Channel 3	0x0000_0000
CRPT_AES3_SADDR	CRYP_BA+0x1F4	R/W	AES DMA Source Address Register for Channel 3	0x0000_0000
CRPT_AES3_DADDR	CRYP_BA+0x1F8	R/W	AES DMA Destination Address Register for Channel 3	0x0000_0000
CRPT_AES3_CNT	CRYP_BA+0x1FC	R/W	AES Byte Count Register for Channel 3	0x0000_0000
CRPT_TDES_CTL	CRYP_BA+0x200	R/W	TDES/DES Control Register	0x0000_0000
CRPT_TDES_STS	CRYP_BA+0x204	R	TDES/DES Engine Flag	0x0001_0100
CRPT_TDES0_KEY1H	CRYP_BA+0x208	R/W	TDES/DES Key 1 High Word Register for Channel 0	0x0000_0000
CRPT_TDES0_KEY1L	CRYP_BA+0x20C	R/W	TDES/DES Key 1 Low Word Register for Channel 0	0x0000_0000
CRPT_TDES0_KEY2H	CRYP_BA+0x210	R/W	TDES Key 2 High Word Register for Channel 0	0x0000_0000
CRPT_TDES0_KEY2L	CRYP_BA+0x214	R/W	TDES Key 2 Low Word Register for Channel 0	0x0000_0000

CRPT_TDES0_KEY3H	CRYP_BA+0x218	R/W	TDES Key 3 High Word Register for Channel 0	0x0000_0000
CRPT_TDES0_KEY3L	CRYP_BA+0x21C	R/W	TDES Key 3 Low Word Register for Channel 0	0x0000_0000
CRPT_TDES0_IVH	CRYP_BA+0x220	R/W	TDES/DES Initial Vector High Word Register for Channel 0	0x0000_0000
CRPT_TDES0_IVL	CRYP_BA+0x224	R/W	TDES/DES Initial Vector Low Word Register for Channel 0	0x0000_0000
CRPT_TDES0_SADDR	CRYP_BA+0x228	R/W	TDES/DES DMA Source Address Register for Channel 0	0x0000_0000
CRPT_TDES0_DADDR	CRYP_BA+0x22C	R/W	TDES/DES DMA Destination Address Register for Channel 0	0x0000_0000
CRPT_TDES0_CNT	CRYP_BA+0x230	R/W	TDES/DES Byte Count Register for Channel 0	0x0000_0000
CRPT_TDES_DATIN	CRYP_BA+0x234	R/W	TDES/DES Engine Input data Word Register	0x0000_0000
CRPT_TDES_DATOUT	CRYP_BA+0x238	R	TDES/DES Engine Output data Word Register	0x0000_0000
CRPT_TDES1_KEY1H	CRYP_BA+0x248	R/W	TDES/DES Key 1 High Word Register for Channel 1	0x0000_0000
CRPT_TDES1_KEY1L	CRYP_BA+0x24C	R/W	TDES/DES Key 1 Low Word Register for Channel 1	0x0000_0000
CRPT_TDES1_KEY2H	CRYP_BA+0x250	R/W	TDES Key 2 High Word Register for Channel 1	0x0000_0000
CRPT_TDES1_KEY2L	CRYP_BA+0x254	R/W	TDES Key 2 Low Word Register for Channel 1	0x0000_0000
CRPT_TDES1_KEY3H	CRYP_BA+0x258	R/W	TDES Key 3 High Word Register for Channel 1	0x0000_0000
CRPT_TDES1_KEY3L	CRYP_BA+0x25C	R/W	TDES Key 3 Low Word Register for Channel 1	0x0000_0000
CRPT_TDES1_IVH	CRYP_BA+0x260	R/W	TDES/DES Initial Vector High Word Register for Channel 1	0x0000_0000
CRPT_TDES1_IVL	CRYP_BA+0x264	R/W	TDES/DES Initial Vector Low Word Register for Channel 1	0x0000_0000
CRPT_TDES1_SADDR	CRYP_BA+0x268	R/W	TDES/DES DMA Source Address Register for Channel 1	0x0000_0000
CRPT_TDES1_DADDR	CRYP_BA+0x26C	R/W	TDES/DES DMA Destination Address Register for Channel 1	0x0000_0000
CRPT_TDES1_CNT	CRYP_BA+0x270	R/W	TDES/DES Byte Count Register for Channel 1	0x0000_0000
CRPT_TDES2_KEY1H	CRYP_BA+0x288	R/W	TDES/DES Key 1 High Word Register for Channel 2	0x0000_0000
CRPT_TDES2_KEY1L	CRYP_BA+0x28C	R/W	TDES/DES Key 1 Low Word Register for Channel 2	0x0000_0000
CRPT_TDES2_KEY2H	CRYP_BA+0x290	R/W	TDES Key 2 High Word Register for Channel 2	0x0000_0000
CRPT_TDES2_KEY2L	CRYP_BA+0x294	R/W	TDES Key 2 Low Word Register for Channel 2	0x0000_0000
CRPT_TDES2_KEY3H	CRYP_BA+0x298	R/W	TDES Key 3 High Word Register for Channel 2	0x0000_0000
CRPT_TDES2_KEY3L	CRYP_BA+0x29C	R/W	TDES Key 3 Low Word Register for Channel 2	0x0000_0000

CRPT_TDES2_IVH	CRYP_BA+0x2A0	R/W	TDES/DES Initial Vector High Word Register for Channel 2	0x0000_0000
CRPT_TDES2_IVL	CRYP_BA+0x2A4	R/W	TDES/DES Initial Vector Low Word Register for Channel 2	0x0000_0000
CRPT_TDES2_SADDR	CRYP_BA+0x2A8	R/W	TDES/DES DMA Source Address Register for Channel 2	0x0000_0000
CRPT_TDES2_DADDR	CRYP_BA+0x2AC	R/W	TDES/DES DMA Destination Address Register for Channel 2	0x0000_0000
CRPT_TDES2_CNT	CRYP_BA+0x2B0	R/W	TDES/DES Byte Count Register for Channel 2	0x0000_0000
CRPT_TDES3_KEY1H	CRYP_BA+0x2C8	R/W	TDES/DES Key 1 High Word Register for Channel 3	0x0000_0000
CRPT_TDES3_KEY1L	CRYP_BA+0x2CC	R/W	TDES/DES Key 1 Low Word Register for Channel 3	0x0000_0000
CRPT_TDES3_KEY2H	CRYP_BA+0x2D0	R/W	TDES Key 2 High Word Register for Channel 3	0x0000_0000
CRPT_TDES3_KEY2L	CRYP_BA+0x2D4	R/W	TDES Key 2 Low Word Register for Channel 3	0x0000_0000
CRPT_TDES3_KEY3H	CRYP_BA+0x2D8	R/W	TDES Key 3 High Word Register for Channel 3	0x0000_0000
CRPT_TDES3_KEY3L	CRYP_BA+0x2DC	R/W	TDES Key 3 Low Word Register for Channel 3	0x0000_0000
CRPT_TDES3_IVH	CRYP_BA+0x2E0	R/W	TDES/DES Initial Vector High Word Register for Channel 3	0x0000_0000
CRPT_TDES3_IVL	CRYP_BA+0x2E4	R/W	TDES/DES Initial Vector Low Word Register for Channel 3	0x0000_0000
CRPT_TDES3_SADDR	CRYP_BA+0x2E8	R/W	TDES/DES DMA Source Address Register for Channel 3	0x0000_0000
CRPT_TDES3_DADDR	CRYP_BA+0x2EC	R/W	TDES/DES DMA Destination Address Register for Channel 3	0x0000_0000
CRPT_TDES3_CNT	CRYP_BA+0x2F0	R/W	TDES/DES Byte Count Register for Channel 3	0x0000_0000
CRPT_HMAC_CTL	CRYP_BA+0x300	R/W	SHA/HMAC Control Register	0x0000_0000
CRPT_HMAC_STS	CRYP_BA+0x304	R	SHA/HMAC Status Flag	0x0000_0000
CRPT_HMAC_DGST0	CRYP_BA+0x308	R	SHA/HMAC Digest Message 0	0x0000_0000
CRPT_HMAC_DGST1	CRYP_BA+0x30C	R	SHA/HMAC Digest Message 1	0x0000_0000
CRPT_HMAC_DGST2	CRYP_BA+0x310	R	SHA/HMAC Digest Message 2	0x0000_0000
CRPT_HMAC_DGST3	CRYP_BA+0x314	R	SHA/HMAC Digest Message 3	0x0000_0000
CRPT_HMAC_DGST4	CRYP_BA+0x318	R	SHA/HMAC Digest Message 4	0x0000_0000
CRPT_HMAC_DGST5	CRYP_BA+0x31C	R	SHA/HMAC Digest Message 5	0x0000_0000
CRPT_HMAC_DGST6	CRYP_BA+0x320	R	SHA/HMAC Digest Message 6	0x0000_0000
CRPT_HMAC_DGST7	CRYP_BA+0x324	R	SHA/HMAC Digest Message 7	0x0000_0000

CRPT_HMAC_DGST8	CRYP_BA+0x328	R	SHA/HMAC Digest Message 8	0x0000_0000
CRPT_HMAC_DGST9	CRYP_BA+0x32C	R	SHA/HMAC Digest Message 9	0x0000_0000
CRPT_HMAC_DGST10	CRYP_BA+0x330	R	SHA/HMAC Digest Message 10	0x0000_0000
CRPT_HMAC_DGST11	CRYP_BA+0x334	R	SHA/HMAC Digest Message 11	0x0000_0000
CRPT_HMAC_DGST12	CRYP_BA+0x338	R	SHA/HMAC Digest Message 12	0x0000_0000
CRPT_HMAC_DGST13	CRYP_BA+0x33C	R	SHA/HMAC Digest Message 13	0x0000_0000
CRPT_HMAC_DGST14	CRYP_BA+0x340	R	SHA/HMAC Digest Message 14	0x0000_0000
CRPT_HMAC_DGST15	CRYP_BA+0x344	R	SHA/HMAC Digest Message 15	0x0000_0000
CRPT_HMAC_KEYCNT	CRYP_BA+0x348	R/W	SHA/HMAC Key Byte Count	0x0000_0000
CRPT_HMAC_SADDR	CRYP_BA+0x34C	R/W	SHA/HMAC DMA Source Address Register	0x0000_0000
CRPT_HMAC_DMACNT	CRYP_BA+0x350	R/W	SHA/HMAC Byte Count Register	0x0000_0000
CRPT_HMAC_DATIN	CRYP_BA+0x354	R/W	SHA/HMAC Engine Non-DMA Mode Data Input Port Register	0x0000_0000

6.5 功能描述

加密加速器包括一個安全偽隨機數發生器（PRNG）和支持 AES，DES / TDES，SHA 和 HMAC 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。

DES / TDES 加速器是完全符合 DES 和三重DES 加密／解密算法。DES / TDES 加速器支持 ECB，CBC，CFB，OFB 和 CTR 等加密模式。DES / TDES 加速器還支持 DMA 功能以減少 CPU的干預。它只有兩種突發長度，4字和8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

6.5.1 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：

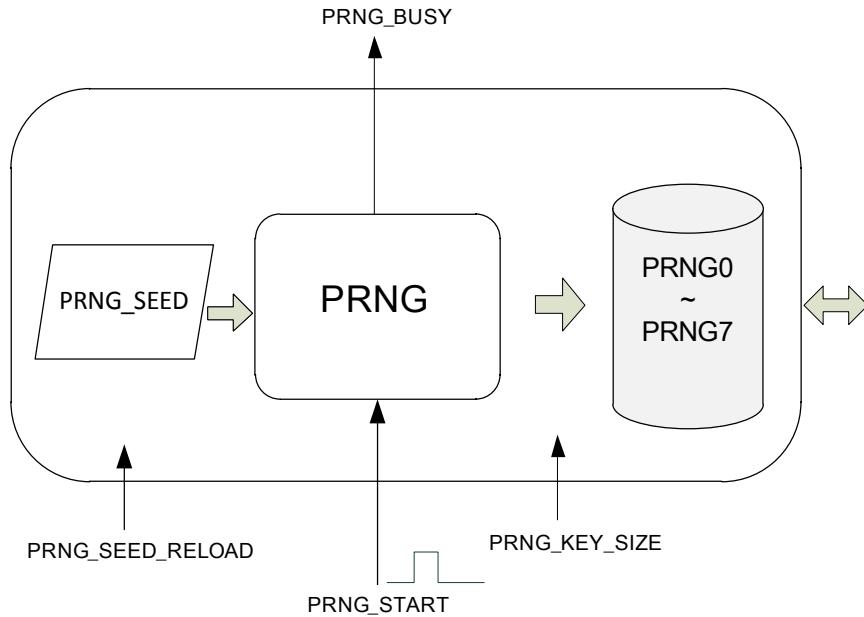
1. DMA 模式：軟件依照數據緩衝區地址，配置好 DMA 源地址寄存器，目的地址寄存器和字節計數寄存器，之後便完全由 DMA 邏輯單元負責對加密加速器的數據寫入及讀出。這種模式可卸去 CPU 的搬運數據的負荷。加密加速器內嵌四個硬件 DMA 通道給 AES 引擎，四個硬件 DMA 通道給 DES / TDES 引擎，以及一個硬件 DMA 通道給 SHA / HMAC 引擎。
2. DMA 級聯模式：在加密數據量龐大，或加密數據流無法一次到位的情況下，或是有其他任務需要即時處理的情況下，可以採用 DMA 級聯（DMA Cascade）模式。在這種模式下，軟件可以分段處理數據加密，一開始的第一段加密指定為 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。

6.5.2 通道擴展

AES 及 DES / TDES 加速器各自均提供了 4 個 DMA 通道。然而，在任一 DMA 通道上擴展虛擬通道是可行的。軟件記錄反饋寄存器（CRPT_AES_FDBCKx，CRPT_TDES_FDBCKH 和 CRPT_TDES_FDBCKL）的中間數據，並記錄目前 DMA 通道上的加密設置，包括操作模式，加密控制寄存器，密鑰，密鑰大小，初始向量等參數，是為虛擬通道。軟件可以建立多組虛擬通道，令彼此共享任一個 DMA 通道，達到實際擴展通道數量的目的。

6.5.3 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。

6.5.4 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 NUC970 AES 加速器。

6.5.4.1 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_AESn_KEY0 ~ CRPT_AESn_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_AESn_KEY0 ~ CRPT_AESn_KEY7。
4. 如果有初始向量，將其寫到 CRPT_AESn_IV0~CRPT_AESn_IV3。
5. 將加密／解密來源數據緩存的實體地址，寫到 CRPT_AESn_SADDR 寄存器。並將輸出數據緩存的實體地址寫到 CRPT_AESn_DADDR 寄存器。將 DMA 傳輸字節數寫到 CRPT_AESn_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。

6.5.4.2 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_AESn_KEY0 ~ CRPT_AESn_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_AESn_KEY0 ~ CRPT_AESn_KEY7。
4. 如果有初始向量，將其寫到 CRPT_AESn_IV0~CRPT_AESn_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。

6.5.5 DES/TDES

FIPS 46-3 指定兩個加密算法：數據加密標準（DES）和三重數據加密算法（TDEA）。NUC970 加密加速器支持 FIPS 46-3 加密和解密，以及 ECB，CBC，CFB，OFB 和 CTR模式。

用戶可以參考下面的步驟，了解如何使用 NUC970 DES/TDES 加速器。

6.5.5.1 DES/TDES DMA 模式的操作程序

1. 寫 1 到 TDESIEN (CRPT_INTEN[8])，使能 DES/TDES 中斷。
2. 從 DES/TDES 的 4 個 DMA 通道選擇一個閒置的通道。
3. 將 DES/TDES 密鑰寫入到 CRPT_TDESn_KEY1H, CRPT_TDESn_KEY1L, CRPT_TDESn_KEY2H, CRPT_TDESn_KEY2L, CRPT_TDESn_KEY3H 和 CRPT_TDESn_KEY3L 寄存器（其中，n 是所選擇的頻道數）。
4. 如果有初始向量，將其寫到 CRPT_TDESn_IVH 和 CRPT_TDESn_IVL。
5. 將加密／解密來源數據緩存的實體地址，寫到 CRPT_TDESn_SADDR 寄存器。並將輸出數據緩存的實體地址寫到 CRPT_TDESn_DADDR 寄存器。將 DMA 傳輸字節數寫到 CRPT_TDESn_CNT 寄存器。
6. 在 DES/TDES 控制寄存器 CRPT_TDES_CTL 選擇通道，加密/解密，加密模式，DMA 模式，DES/TDES 選擇，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_TDES_CTL[0]) 開始 DES/TDES 加密/解密。
9. 等待 DES/TDES 中斷標誌 TDESIF (CRPT_INTSTS[8]) 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。

6.5.5.2 DES/TDES 非DMA 模式的操作程序

1. 寫 1 到 TDESIEN (CRPT_INTEN[8])，使能 DES/TDES 中斷。
2. 從 DES/TDES 的 4 個 DMA 通道選擇一個閒置的通道。
3. 將 DES/TDES 密鑰寫入到 CRPT_TDESn_KEY1H, CRPT_TDESn_KEY1L, CRPT_TDESn_KEY2H, CRPT_TDESn_KEY2L, CRPT_TDESn_KEY3H 和 CRPT_TDESn_KEY3L 寄存器（其中，n 是所選擇的頻道數）。
4. 如果有初始向量，將其寫到 CRPT_TDESn_IVH 和 CRPT_TDESn_IVL。
5. 將加密／解密來源數據緩存的實體地址，寫到 CRPT_TDESn_SADDR 寄存器。並將輸出數據緩存的實體地址寫到 CRPT_TDESn_DADDR 寄存器。將 DMA 傳輸字節數寫到 CRPT_TDESn_CNT 寄存器。

6. 寫 1 到 START (CRPT_TDES_CTL[0]) 開始 DES/TDES 加密/解密。
7. 輪詢 INBUFFULL (CRPT_TDES_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_TDES_DATIN 寄存器。
8. 輪詢 OUTBUFEMPTY (CRPT_TDES_STS[16]) 位，如果此位為 0，便從 CRPT_TDES_DATOUT 讀取 32 bits 加密/解密輸出數據。
9. 重複步驟 7~8，直到從 CRPT_TDES_DATOUT 寄存器讀取達 64 bits 的數據（8 字節）為止。
10. 對 DMALAST (CRPT_TDES_CTL[5]) 位寫 1，表示完成一個 DES/TDES 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。

6.5.6 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 NUC970 SHA 加速器。

6.5.6.1 SHA 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. 將欲進行 SHA 運算的數據寫到 DMA 來源數據緩存區。
5. 寫 1 到 START (CRPT_HMAC_CTL[0]) 開始 SHA 運算。
6. 等待 SHA 中斷標誌 HMACIF (CRPT_INTSTS[24]) 被設置。
7. 直接從 CRPT_HMAC_DGST0~CRPT_HMAC_DGST7 寄存器讀取 SHA 運算結果。

6.5.6.2 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。

7 外部總線 (External Bus Interface)

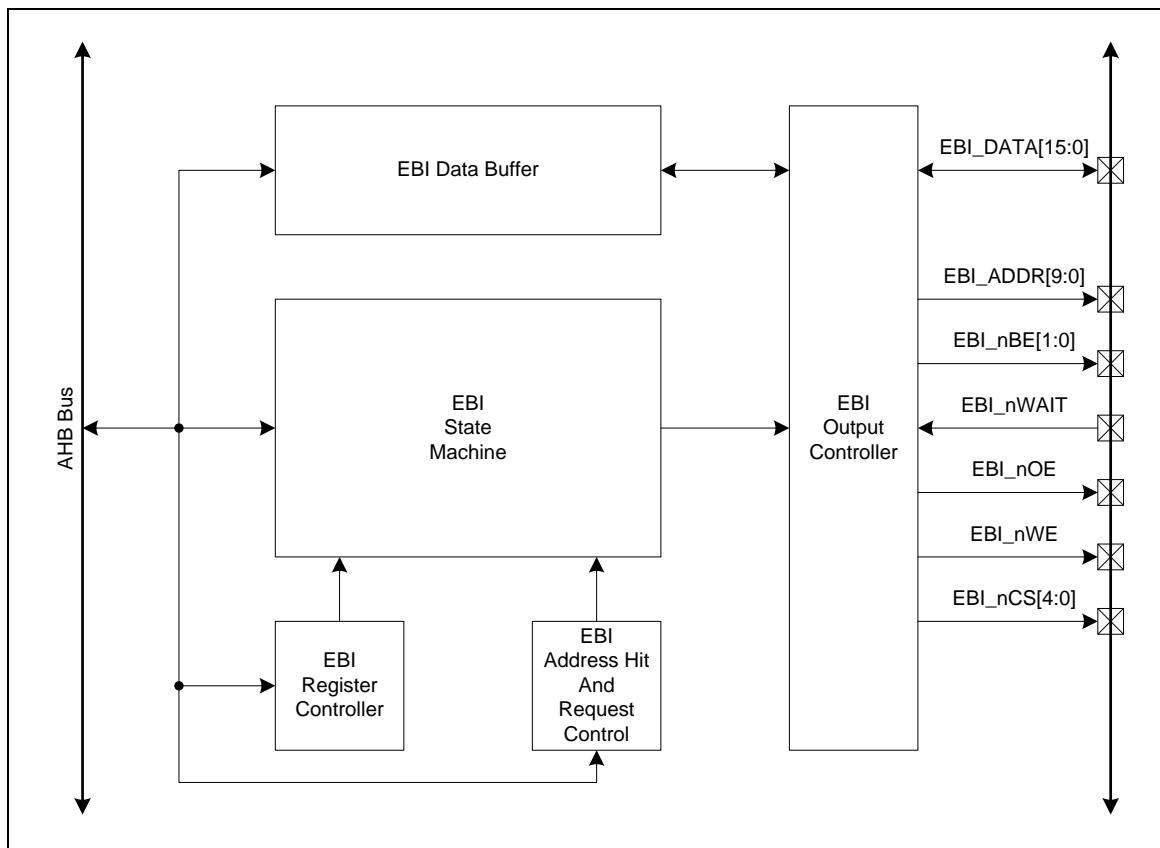
7.1 概述

NUC970系列外部總線接口 (EBI) ，控制訪問外部存儲器(SRAM)和外部I/O設備。在EBI具有最多5芯片選擇信號以選擇具有10位地址總線不同的設備。它支持8位和16位的外部資料總線寬度。

7.2 特性

- 支持SRAM和外部I/O設備。
- 支持8 bit/16bit數據總線寬度。
- 支持80和68模式的接口信號。
- 支持最高5 SRAM和外部I/O設備。
- 支持可編程訪問週期。
- 支持4個32位的寫緩存。

7.3 方塊圖



7.4 寄存器

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

Register	Offset	R/W	Description	Reset Value
(EBI_BA=0xB000_1000)				
EBI_CTL	EBI_BA+0x000	R/W	EBI Control Register	0x0001_0001
EBI_BNKCTL0	EBI_BA+0x018	R/W	External Bus Bank 0 Control Register	0x0000_0000
EBI_BNKCTL1	EBI_BA+0x01C	R/W	External Bus Bank 1 Control Register	0x0000_0000
EBI_BNKCTL2	EBI_BA+0x020	R/W	External Bus Bank 2 Control Register	0x0000_0000
EBI_BNKCTL3	EBI_BA+0x024	R/W	External Bus Bank 3 Control Register	0x0000_0000
EBI_BNKCTL4	EBI_BA+0x028	R/W	External Bus Bank 4 Control Register	0x0000_0000

7.5 功能描述.

7.5.1 基本配置

當 EBI 使用之前，必須將 EBI(HCLKEN[9]) 設置為 1。並將多功能腳位 SYS_MFP_GPDH, SYS_MFP_GPHL, SYS_MFP_GPHH, SYS_MFP_GPIL 和 SYS_MFP_GPIH 相關腳位設定給 EBI 使用。

7.5.2 外部 I/O 控制

NUC970 系列 MCU 支持外部設備控制。這是非常符合成本效益，因為提供了不需要地址譯碼和控制信號的時序邏輯。控制寄存器可用於提供低成本的外部設備控制外部 I/O 設備。舉例來說，如果有一個 SRAM 連接到外部 I/O 組 0。然後外部控制寄存器 0 (EBI_BNKCTL0) 設置後，即可經由外部 I/O 訪問 SRAM，ROM 也可以連接到外部 I/O。有五個外部 I/O 相對於五個控制寄存器，稱為 EBI_BNKCTL0, EBI_BNKCTL1, EBI_BNKCTL2, EBI_BNKCTL3 和 EBI_BNKCTL4。每個外部 I/O 組的基地址可以通過外部 I/O 控制寄存器的 BASADDR (EBI_BNKCTLx[31:19], x is 0, 1, 2, 3 and 4) 進行設置。BASADDR 是 13 元並且地址被計算為 BASADDR << 18。EBI_CTL 依據外部 I/O 設備的控制方式來決定使用 EXB 模式或 M68 模式。透過 EXB0 (EBI_CTL0[24]) 和 M68E0 (EBI_CTL0[19]) 定義了 EBI_nBE1 腳位、EBI_nBE0 腳位和 EBI_nWE 腳位，如何訪問外部 I/O 設備，設定方式如下表：

EXBE0	M68E0	Description
0	0	Pin EBI_nBE1 and EBI_nBE0 used as byte write strobe signal.

1	0	Pin EBI_nBE1 and EBI_nBE0 used as byte enable signals while EBI_nWE used as write strobe signal to external device.
0	1	EBI_nCS0 pin is the enable signal, EBI_nWE used as read/write strobe signal
1	1	Reserved

示範一個將外部SRAM經由EBI_BNKCTL0映射到0x20000000的軟件程序，如下：

```
unsigned int value, bank = 0;
unsigned int BASEADDR=0x20000000;
unsigned int SIZE=1;    //512K
unsigned int DBWD=2;   //16bit

/* External Bus Bank 4 Byte Enable for EBI_BNKCTL0 */
rEBI_CTL |= (0x1<<(bank+24));

/* Set timing, base address and sram size */
rEBI_BNKCTL0 = (unsigned int)( (BASEADDR<<1)      |
                               (SIZE << 16)       |
                               (DBWD << 0)        |
                               (6<<11 /*tACC*/) |
                               (3<<2 /*tCos*/));
```

8 乙太網路控制器 (EMAC)

8.1 概述

NUC970 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.

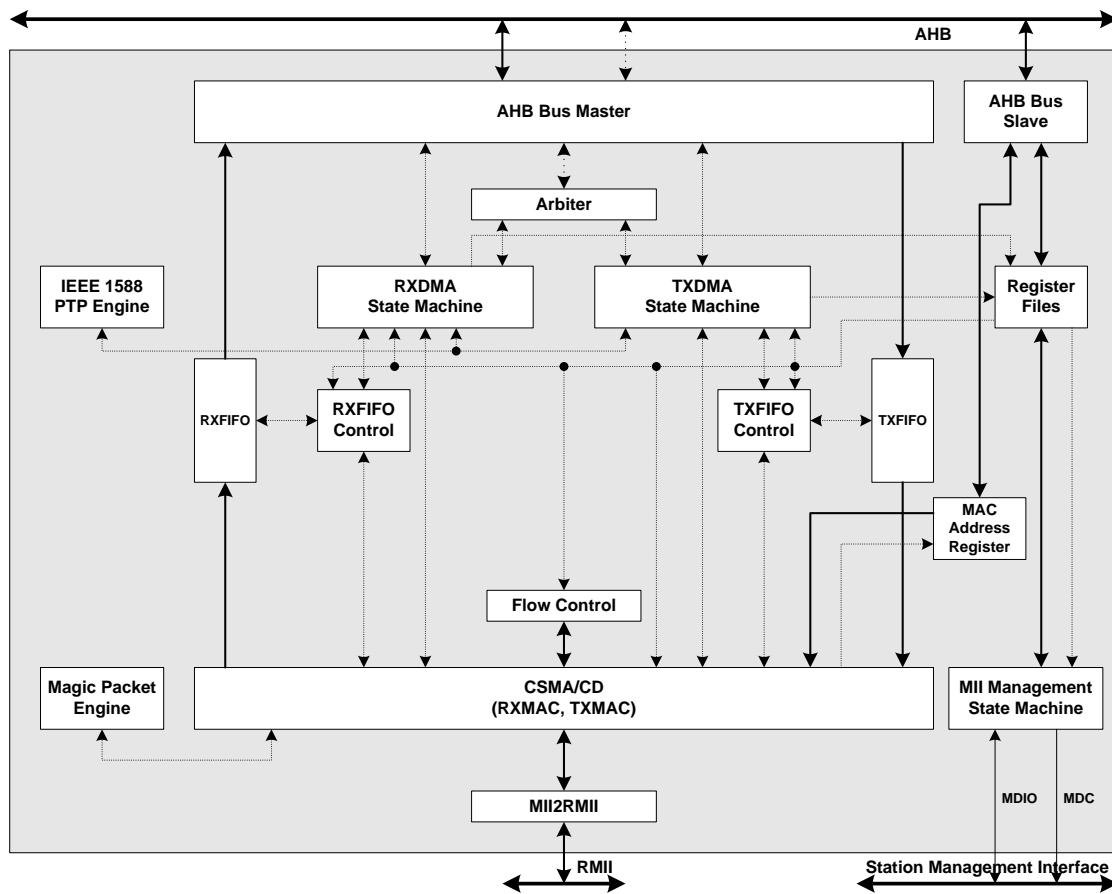
乙太網路控制器包含了 IEEE 802.3 乙太網通訊協定引擎, 並且內建了結合存儲 (CAM) 用於 MAC 位址比對. 另外還包含傳送 FIFO, 接收 FIFO, Tx/Rx 狀態機控制單元, IEEE 1588 時間戳記, 並且支持魔法封包分析的功能.

乙太網路控制器透過 RMII 介面與外部的 PHY 連接.

8.2 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

8.3 方塊圖



8.4 寄存器

Register	Offset	R/W	Description	Reset Value
EMAC0_BA = 0xB000_2000				
EMAC0_BA = 0xB000_3000				
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_CRXBBA	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

8.5 功能描述

8.5.1 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_MIID[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_MIID[12:8]) 以及 PHY 寄存器地址 PHYRAD (EMAC_MIID[4:0]).
2. 將 BUSY (EMAC_MIID[17]), 以及 MDCON (EMAC_MIID[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_MIID = reg | (addr << 8) | BUSY | MDCON;
    while(EMAC_MIID & 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_MIID[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_MIID = reg | (addr << 8) | BUSY | MDCON | WRITE;
    while(EMAC_MIID & 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 FDPU (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.

8.5.2 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. NUC970 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, 0xFFFFF0000, 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.

8.5.3 控制封包

IEEE 802.3 定義了用於流控的暫停封包. NUC970 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: 只能在全雙工模式下使用暫停封包做流控.

8.5.4 遠端喚醒 (WoL)

NUC970 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。

8.5.5 封包接收

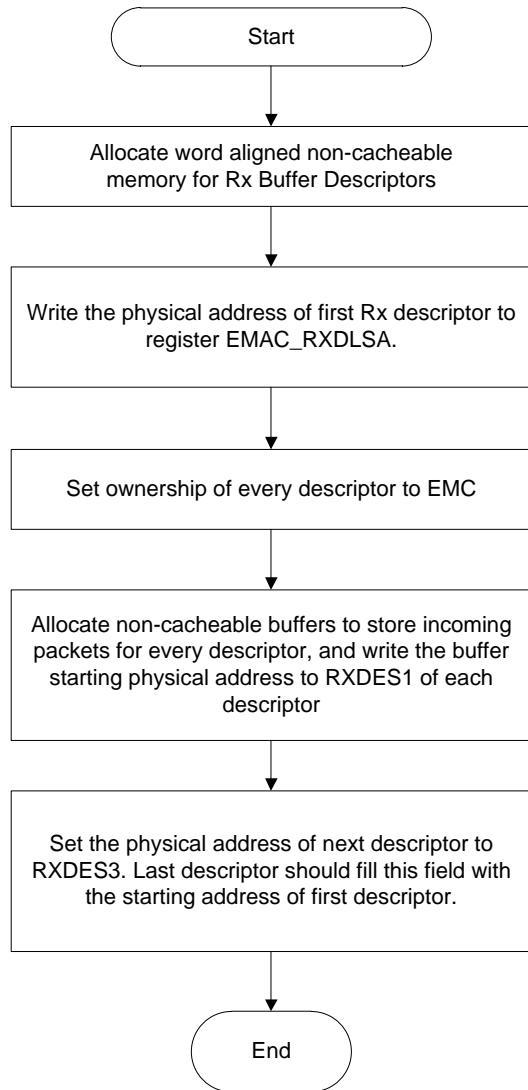
網路封包的接收，是透過預先定義好格式的接收描述符來進行。驅動程式需要預先準備好一些接收描述符，當 CAM 決定偵測到的封包需要接收後，封包就會接收至描述符所指定的內存地址，然後封包的狀態以及長度會被儲存至描述符中，接著乙太網路控制器會使用下一個接收描述符來接收封包。所以 CPU 以及 EMAC 透過接收描述符彼此交換接收封包的資訊。

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	O W N	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_RXDLA 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];

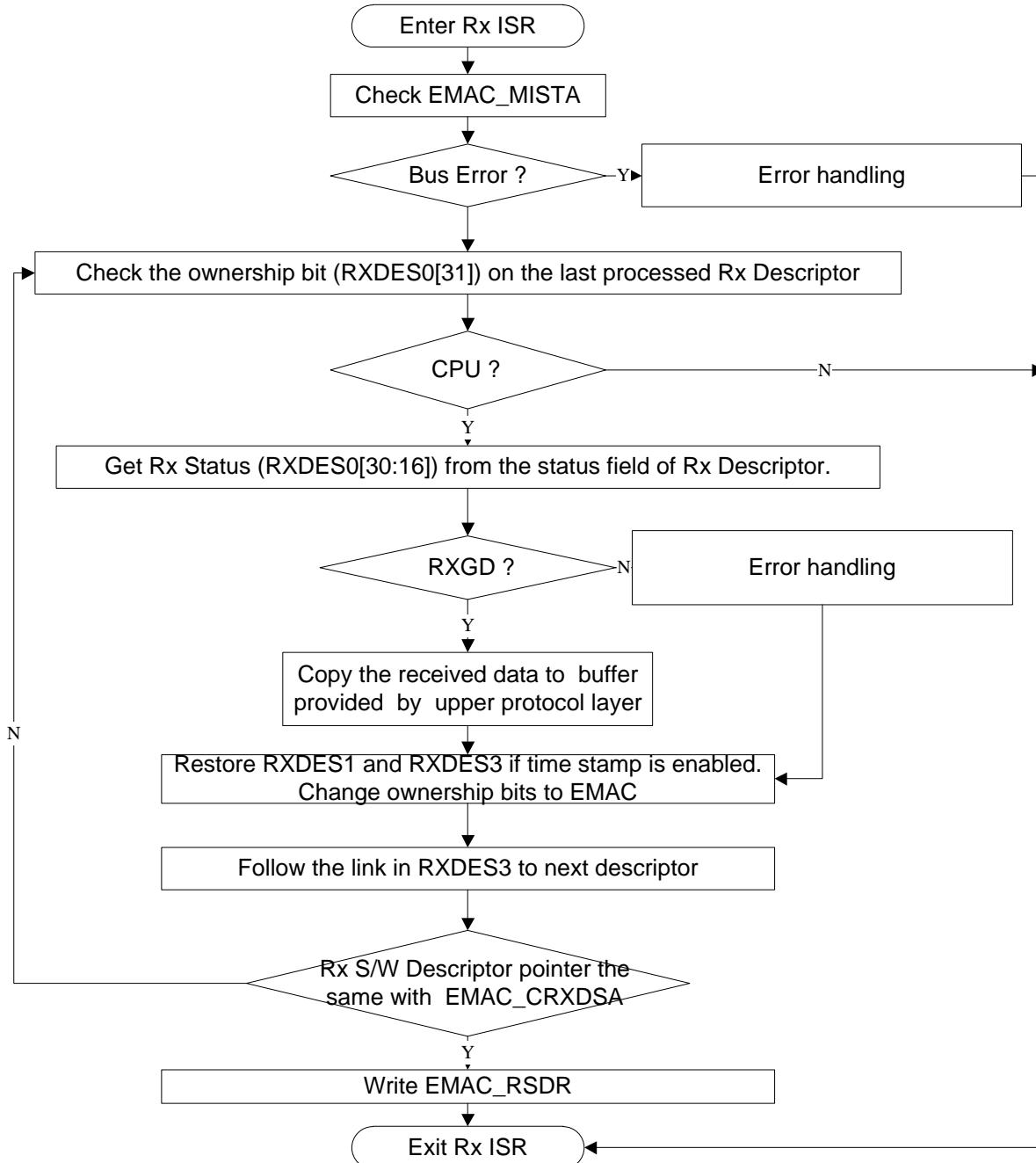
void rx_desc_init(void)
{
    unsigned int i;

    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;
}

```

8.5.6 封包傳送

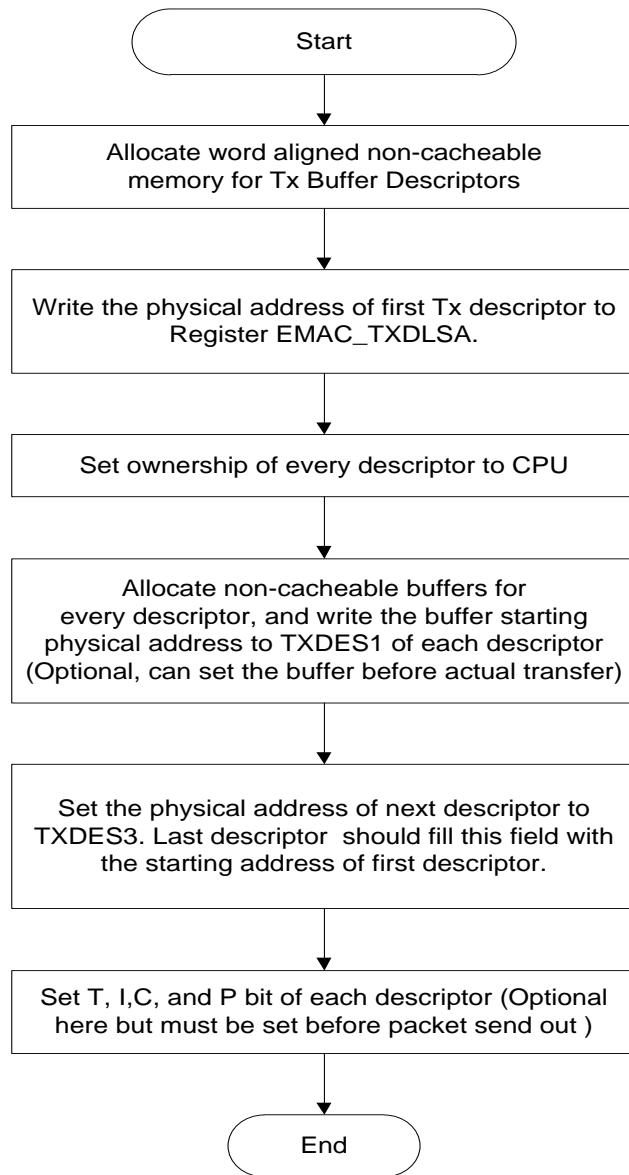
網路封包的傳送與接收相同，是透過預先定義好格式的描述符來進行。驅動程式需要預先準備好一些傳送述符，當決定要傳送封包出去時，將封包放在傳送描述符所指定的內存地址，然後填寫封包的長度，觸發傳送狀態機，乙太網路控制器就會開始傳送此封包，傳送完成後，接著乙太網路控制器會使用下一個傳送描述符來傳送封包。所以CPU 以及 EMAC 透過傳送描述符彼此交換傳送封包的資訊。

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 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 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。

	31	15	3 2 1 0
TXDES 0	O W N	Reserved	T I C P
TXDES 1	Transmit Frame Buffer Starting Address / Time Stamp Least Significant 32-Bit		
TXDES 2	Transmit Frame Status		Transmit Frame Byte Count
TXDES 3	Next TxDMA Descriptor Starting Address / Time Stamp Most Significant 32-Bit		

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];

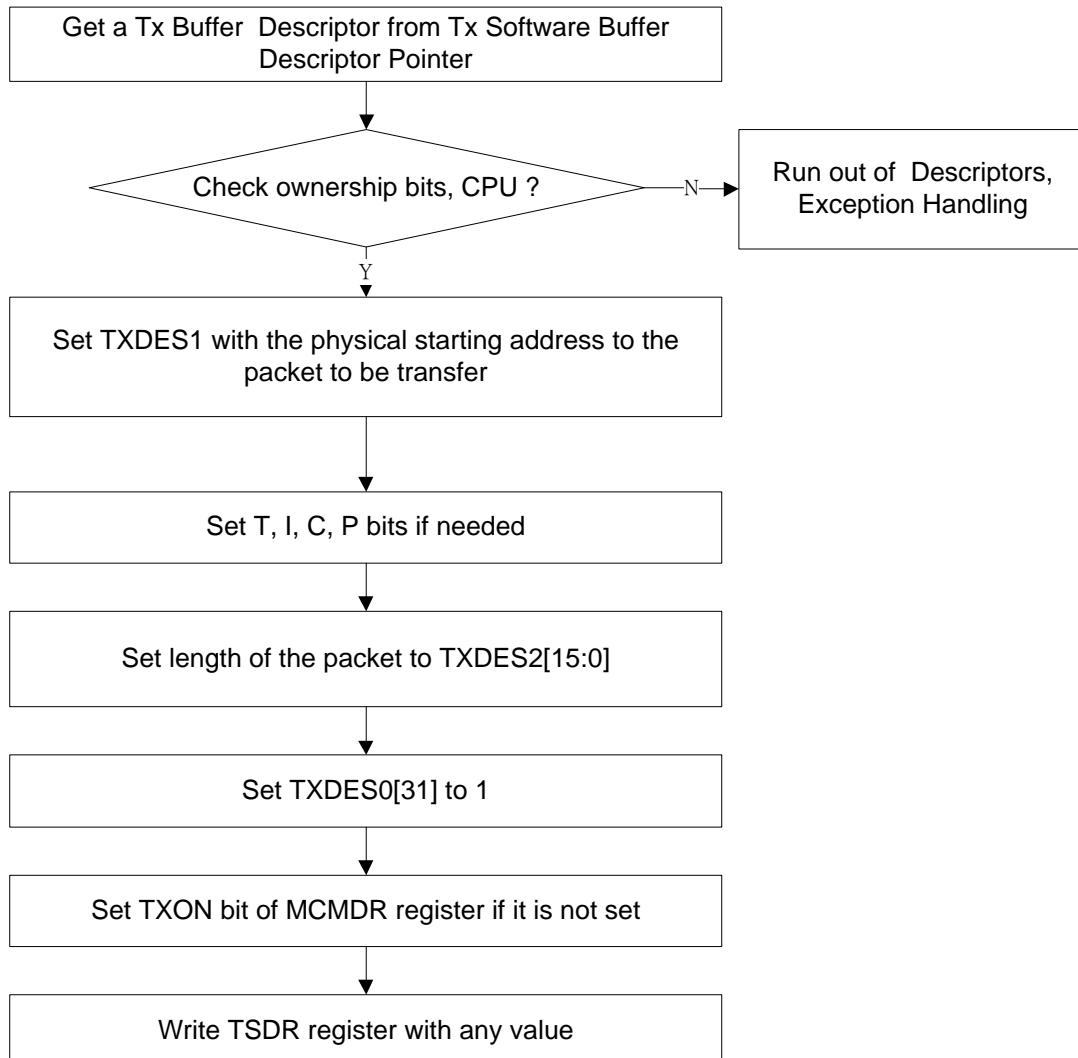
void tx_desc_init(void)
{
    unsigned int i;

    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，除了先前提到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);

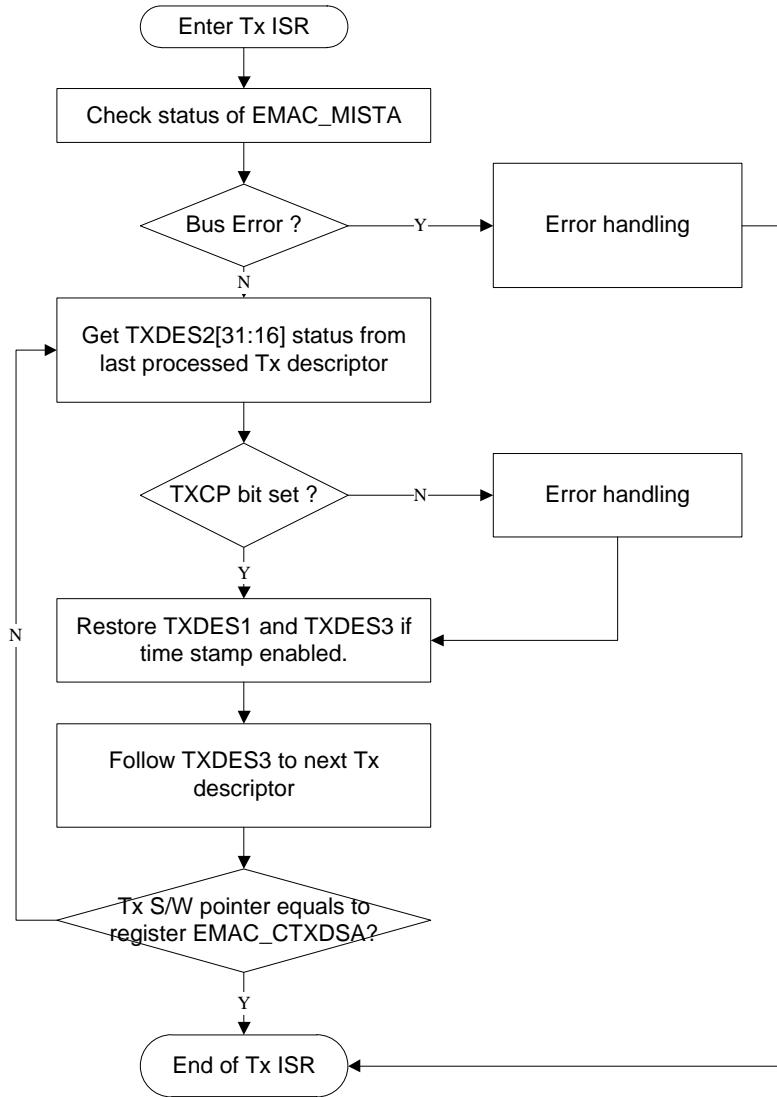
// Set TX Frame flag & Length Field
desc->txdes0 |= (P | C | I | T);
desc->txdes2 = size;

// Change 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 & 0xfffff0000;

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;

}
```

8.5.7 網路時鐘

為了使 IEEE1588 在 NUC970 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 $(2^{31}) / 150M = 14.31 \approx 14 = 0x0E$ ，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。

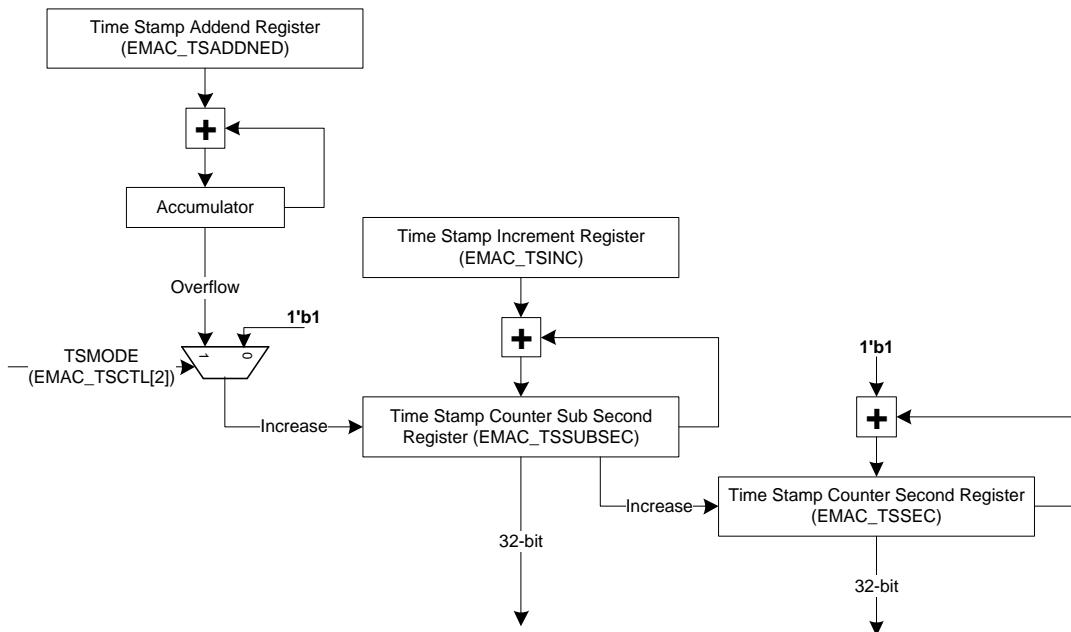
試算 EMAC 時鐘頻率為 150MHz 下，精細更新的時間戳記相關寄存器的設定。假設我們希望每 100 奈秒要對次秒寄存器加上一次 EMAC_TSINC 的值。代表累加 10^7 次之後，次秒需要溢位，所以 EMAC_TSINC 要填入 $2^{31} / 10^7 = 214.71 \approx 215 = 0xD7$ 。此時，實際進位的頻率不是希望的 10⁷ Hz，而是 $2^{31} / 215$ Hz。由此反推回 EMAC_TSADDNED 所需填的值必須能讓累加器在 $2^{31} / 215$ Hz 的頻率下溢位，時間才會準確。也就是 EMAC_TSADDNED 要填入 $2^{32} * (2^{31} / 215) / 150M = 285996032.15 \approx 285996032 = 0x110BF400$ 。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 = 10000000001l * 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.

8.5.8 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	<p>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.</p> <p>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.</p> <p>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.</p>
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.
--	--	-----------------------

9 加強型計時器控制器 (ETMR)

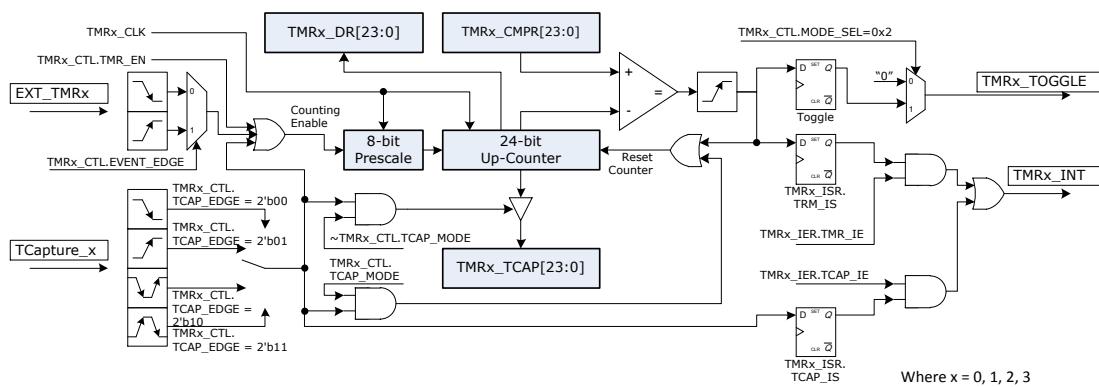
9.1 概述

該晶片帶有四個計時器模組, TIMER0, TIMER1, TIMER2 與 TIMER3, 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.

9.2 特性

- 每個計時器都有獨立的時鐘源 (TMRx_CLK, x=0,1,2,3)
- 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TCMP)
- 計數週期 = $(1 / \text{TMRx_CLK}) * (2^8) * (2^{24})$
- 內部 8 位預分頻計數器
- 內部 24 位上數型計數器通過 TDR 可讀 (計時器資料寄存器)
- 支援單次、週期、輸出翻轉和連續計數操作模式
- 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

9.3 方塊圖



9.4 寄存器

Register	Offset	R/W	Description	Reset Value
----------	--------	-----	-------------	-------------

ETMR Base Address:				
ETMR0_BA = 0x4001_0000				
ETMR1_BA = 0x4001_0100				
ETMR2_BA = 0x4011_0000				
ETMR3_BA = 0x4011_0100				
ETMR_CTL n=0,1,2,3	ETMRn_BA+0x000	R/W	Enhance Timer n Control Register	0x0000_0000
ETMR_PRECNT n=0,1,2,3	ETMRn_BA+0x004	R/W	Enhance Timer n Pre-Scale Counter Register	0x0000_0000
ETMR_CMPR n=0,1,2,3	ETMRn_BA+0x008	R/W	Enhance Timer n Compare Register	0x0000_0000
ETMR_IER n=0,1,2,3	ETMRn_BA+0x00C	R/W	Enhance Timer n Interrupt Enable Register	0x0000_0000
ETMR_ISR n=0,1,2,3	ETMRn_BA+0x010	R/W	Enhance Timer n Interrupt Status Register	0x0000_0000
ETMR_DR n=0,1,2,3	ETMRn_BA+0x014	R	Enhance Timer n Data Register	0x0000_0000
ETMR_TCAP n=0,1,2,3	ETMRn_BA+0x018	R	Enhance Timer n Capture Data Register	0x0000_0000

9.5 功能描述

定时器控制器提供单次，周期，翻转输出和连续计数等操作模式。也提供外部引脚捕捉功能用于时间间隔测量或者复位定时器计数器。另外，定时器控制器提供定时器间互触发模式用于精确测量输入频率。每个操作功能模式如下.

9.5.1 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 ETMR_EN (ETMRx_CTL[0]) 清0, 停止計時器.
2. 設置操作模式 MODE_SEL (ETMRx_CTL[5:4])
3. 若要使能中斷, 將 ETMR_IE (ETMRx_IER[0]) 置 1, 否則清 0.
4. 設置預分頻PRESCALE_CNT (ETMRx_PRECNT[7:0])
5. 設置比較值 ETMR_CMP (ETMRx_CMPR[24:0])
6. 將ETMR EN (ETMRx_CTL[0]) 置 1, 計時器開始上數計數.

9.5.2 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ETMR_EN (ETMRx_CTL[0]) 以及 TCAP_EN (ETMRx_CTL[16]) 清0, 停止計時器.
2. 設置操作模式 MODE_SEL (ETMRx_CTL[5:4])
3. 若要使能中斷, 將 TCAP_IE (ETMRx_IER[1]) 置 1, 否則清 0.
4. 設置 TCAP_MODE(ETMRx_CTL[17]), 以及 CAP_CNT_MOD, 選擇捕獲模式.
5. 設置 TCAP_EDGE, 選擇捕獲的觸發條件
6. 若需要去抖操作, 將 TCAP_DEB_EN(ETMRx_CTL[22]) 置1, 否則清 0.
7. 設置預分頻PRESCALE_CNT (ETMRx_PRECNT[7:0])
8. 設置比較值 ETMR_CMP (ETMRx_CMPr[24:0])
9. 將 TCAP_EN (ETMRx_CTL[16]) 置1, 使能捕獲模式.
10. 將ETMR EN (ETMRx_CTL[0]) 置 1, 使能計時器.

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

9.5.3 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, ETMR_IS(ETMR_ISR[0]) 會置 1. 當觸發捕獲中斷時, TCAP_IS(ETMR_ISR[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 TCAP_IS 之前, 又有新的補獲事件發生, 則 NCAP_DET_STS (ETMR_ISR[5]) 會被置 1. 此標誌位會在清除 TCAP_IS 時, 一併清 0.

9.5.4 計時器頻率

每個計時器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:

$$\text{頻率} = \text{TMRx_CLK} / ((\text{PRESCALE} + 1) * \text{TCMP})$$

其中 TMRx_CLK 為計時器時鐘源, 可以是HXT (12MHz 外部晶振), PCLK, PCLK/4029, 或是LXT (32.768kHz 外部晶振). PRESCALE 預分頻定義在 PRESCALE_CNT (ETMRx_PRECNT[7:0]), TCMP 比較值定義在 ETMR_CMP (ETMRx_CMPr[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	PRESCALE_CNT (ETMRx_PRECNT[7:0])	ETMR_CMP (ETMRx_CMPr[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

9.5.5 單次模式

如果計時器控制器工作在單次模式 MODE_SEL(ETMRx_CTL[5:4] 為 0x0) 且 ETMR_EN (ETMRx_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 (ETMRx_DR 值) 達到計時器比較器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 將會被設置為 1. 如果 ETMR_IE (ETMRx_IER[0] 計時器中斷使能位) 被設置為 1, 且 ETMR_IS (ETMRx_ISR[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 ETMR_IE (ETMRx_IER[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 (ETMRx_DR 的值) 達到計時器比較器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[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 (ETMRx_DR 的值) 恢復到計數初始值, 然後 ETMR_EN (ETMRx_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 (ETMRx_CMPR) 且 ETMR_EN (ETMRx_CTL[0] 計時器計數器使能位) 之後, 計時器計數和與 ETMRx_CMPR 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.

9.5.6 週期模式

如果計時器工作在週期模式 MODE_SEL(ETMRx_CTL[5:4] 為 0x1) 且 ETMR_EN (ETMRx_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 (ETMRx_DR 的值) 計時器達到比較寄存器(ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 將會被設置為 1. 如果 ETMR_IE (ETMRx_IER[0] 計時器中斷使能位) 被設置為 1, 且 ETMR_IS (ETMRx_ISR[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 ETMR_IE (ETMRx_IER[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 (ETMRx_DR 的值) 達到計時器比較寄存器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 將會被設置為 1, 計時器計數器的值 (ETMRx_DR 的值) 恢復到計數初值, 且 ETMR_EN (ETMRx_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 ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 被軟體清除, 一旦計時器計數值 (ETMRx_DR 的值) 再一次達

到計時器比較寄存器(ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[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 ETMRx_CMPR 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ETMR_EN (ETMRx_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.

9.5.7 翻轉模式

如果計時器工作在翻轉模式 MODE_SEL(ETMRx_CTL[5:4] 為 0x2) 且 ETMR_EN (ETMRx_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 (ETMRx_DR 的值) 達到計時器比較寄存器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 將會被設置為 1. 如果 ETMR_IE (ETMRx_IER[0] 計時器中斷使能位) 被設置為 1, 且 ETMR_IS (ETMRx_ISR[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 ETMR_IE (ETMRx_IER[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 (ETMRx_DR 的值) 達到計時器比較寄存器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[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 (ETMRx_DR 的值) 恢復到計數初值, 且 ETMR_EN (ETMRx_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 ETMR_IS (ETMRx_ISR[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 (ETMRx_DR 的值) 再一次達到計時器比較寄存器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[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 ETMR_EN (ETMRx_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.

9.5.8 連續計數模式

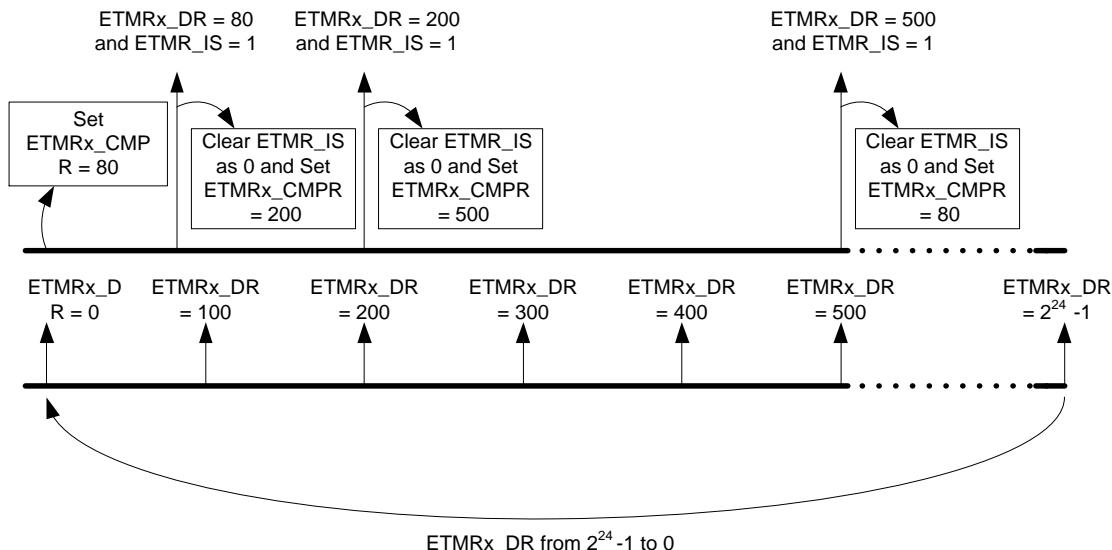
如果計時器工作在連續計數模式 MODE_SEL(ETMRx_CTL[5:4] 為 0x3) 且 ETMR_EN (ETMRx_CTL[0] 計時器計數器使能位) 被設置為 1, 計時器計數器開始向上計數. 一旦計時器計數器的值 (ETMRx_DR 的值) 達到計時器比較寄存器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 將會被設置為 1. 如果 ETMR_IE (ETMRx_IER[0] 計時器中斷使能位) 被設置為 1, 且 ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 為 1, 中斷信號產生並被發送到 AIC 來通知 CPU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

在本操作模式下, 一旦計時器計數器的值 (ETMRx_DR 的值) 達到計時器比較寄存器 (ETMRx_CMPR) 的值, ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 將會被設置為 1, 且 ETMR_EN (ETMRx_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_DR 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 (ETMRx_CMPR) 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 (ETMRx_CMPR) 被設置為 80 (計時器比較寄存器 (ETMRx_CMPR) 的值必須大於 1, 且小於 2^{24}). 一旦計時器計數器的值 (ETMRx_DR 的值) 達到 80, ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 將會被設置為 1, 且 ETMR_EN (ETMRx_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 (ETMRx_DR 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 (ETMRx_CMPR) 的值為 200, 且 ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 被清除為 0, 當計時器比較寄存器的值 (ETMRx_DR 的值) 達到 200, ETMR_IS (ETMRx_ISR[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 (ETMRx_CMPR) 的值為 500, 並清除 ETMR_IS (ETMRx_ISR[0] 計時器中斷狀態) 為 0, 當計時器計數器的值 (ETMRx_DR 的值) 達到 500, ETMR_IS (ETMRx_ISR[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(ETMRx_DR 的值)總是保持持續向上計數, 即便 ETMR_IS (ETMRx_ISR[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。

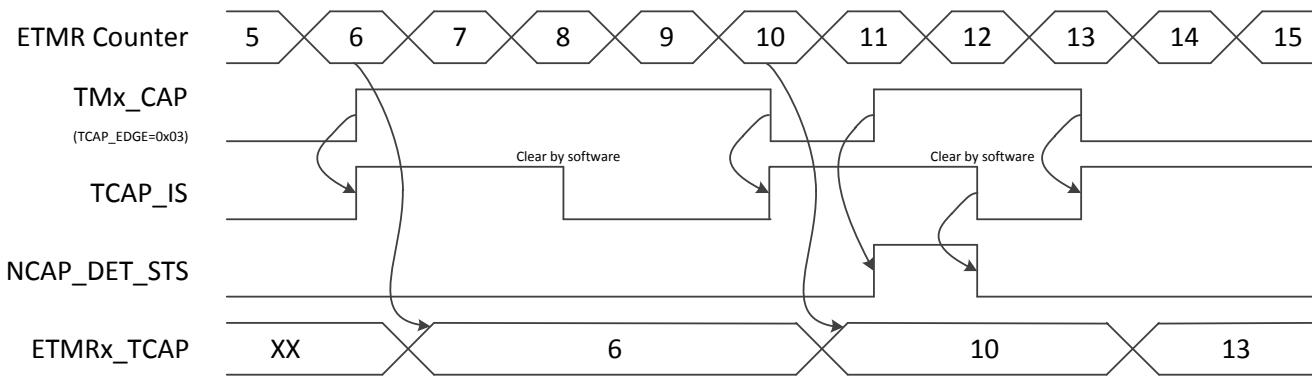


9.5.9 自由計數模式

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 TCAP_MODE (ETMRX_CTL[17]) 為 0, 且 CAP_CNT_MOD (ETMRX_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 TCAP_EDGE (ETMRX_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 ETMRx_TCAP 寄存器。若是TCAP_IE (ETMRx_IER[1]) 為 1, 在此同時 TCAP_IS(ETMR_IS[1]) 會置 1 並觸發中斷。

在自由計數模式下, 當TCAP_EDGE 為 0, TMx_CAP 引腳上的下降緣 (從 1 到 0 翻轉) 是有效邊緣。當TCAP_EDGE 為 1, TMx_CAP 引腳上的上升緣 (從 0 到 1 翻轉) 是有效邊緣。若 TCAP_EDGE 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。

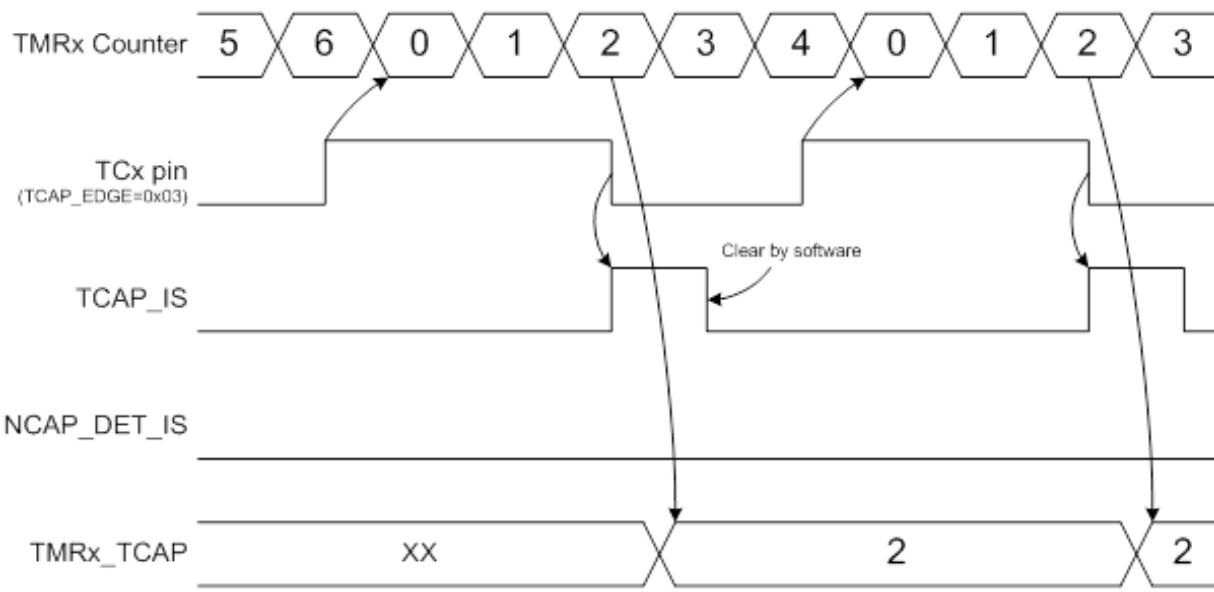


9.5.10 觸發計數模式

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 TCAP_MODE (ETMRX_CTL[17]) 為 0，且 CAP_CNT_MOD (ETMRX_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 TCAP_EDGE (ETMRX_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 TCAP_EDGE 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_TCAP 寄存器中。若是 TCAP_IE (ETMRx_IER[1]) 為 1，在此同時 TCAP_IS (ETMR_ISR[1]) 會置 1 並觸發中斷。

在觸發計數模式下，當 TCAP_EDGE 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 TCAP_EDGE 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 TCAP_EDGE 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 TCAP_EDGE 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。

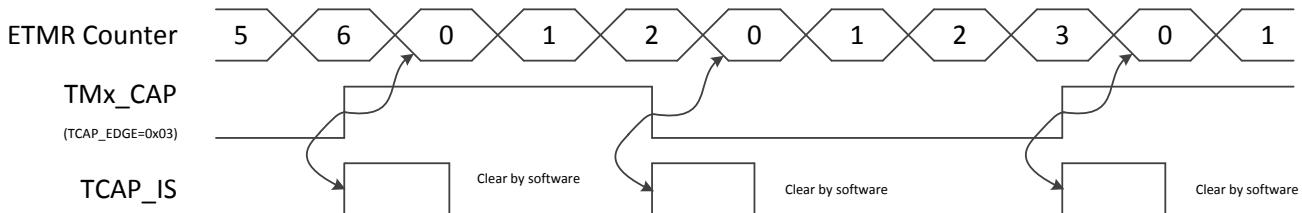


9.5.11 計數器重定模式

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 TCAP_MODE (ETMRX_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 TCAP_EDGE (ETMRX_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TCAP_IE (ETMRx_IER[1]) 為 1，在此同時 TCAP_IS(ETMR_ISR[1]) 會置 1 並觸發中斷。

在計數器重定模式下，當TCAP_EDGE 為 0，TMx_CAP 引腳上的下降緣（從 1 到 0 翻轉）是有效邊緣。當TCAP_EDGE 為 1，TMx_CAP 引腳上的上升緣（從 0 到 1 翻轉）是有效邊緣。若 TCAP_EDGE 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。



9.5.12 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 TCAP_DEB_EN (ETMRX_CTL[22]) 和 TCAP_EN (ETMRX_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 TCAP_EDGE (ETMRX_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 (TCAP_DEB_EN 和 TCAP_EN 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 TCAP_IS (ETMR_ISR[1]) 被置位元時，捕捉的資料不正確 (TMRx_TCAP)，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。

10 閃存接口 (FMI)

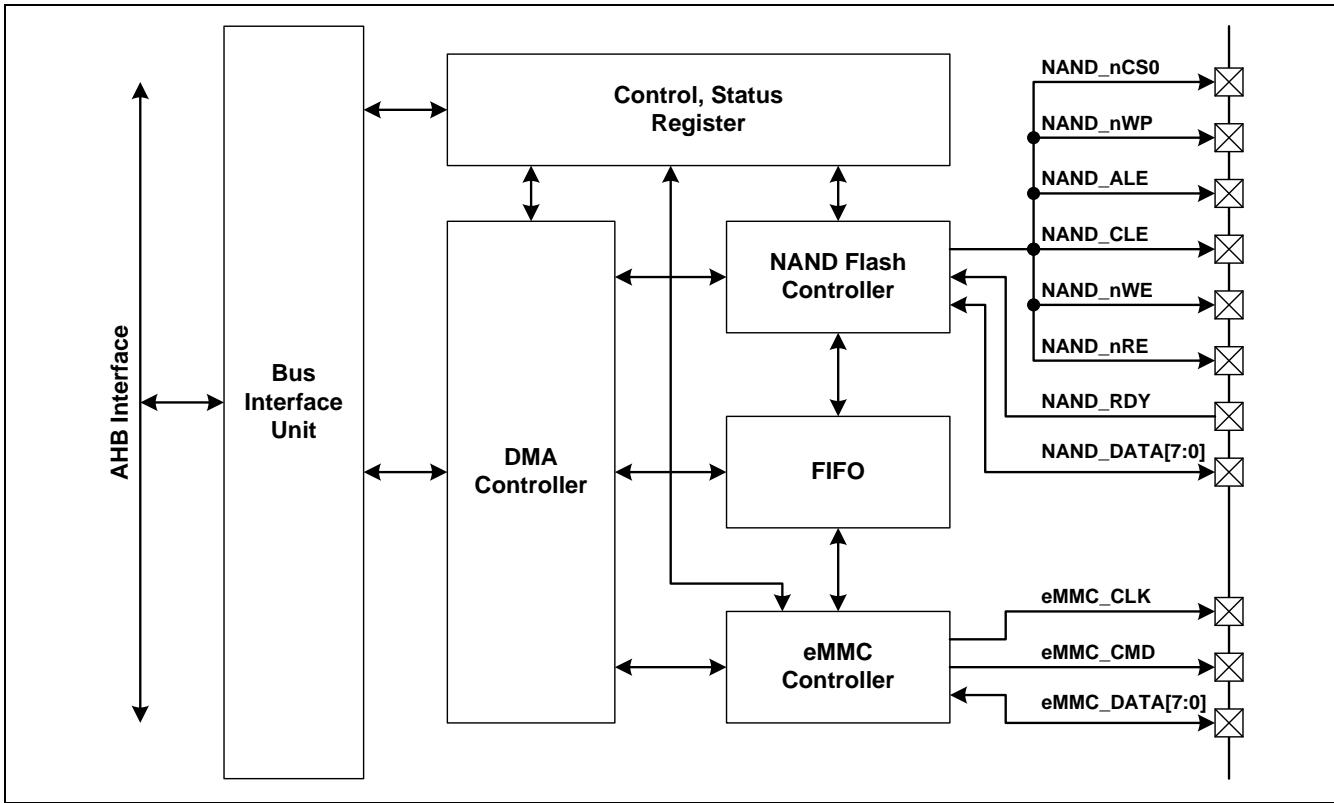
10.1 概述

閃存接口 (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或NAND閃存的接口。閃存接口控制器支持eMMC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。

10.2 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的閃存設備。
- 支持SLC和MLC NAND型閃存。
- 可調NAND頁面大小。512B+備用區域，2048B+備用區，4096B+備用區和8192B+備用區）。
- 支持4位/8位/12位/15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。

10.3 方塊圖



10.4 寄存器

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

Register	Offset	R/W	Description	Reset Value
FMI_BA = 0xB000_D000				
FMI_BUFFERn <i>n = 0, 1..31</i>	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 Enable Register	0x0000_0000
FMI_CTL	FMI_BA+0x800	R/W	FMI Control and Status Register	0x0000_0000
FMI_INTEN	FMI_BA+0x804	R/W	FMI Interrupt Enable Register	0x0000_0001
FMI_INTSTS	FMI_BA+0x808	R/W	FMI Interrupt Status Register	0x0000_0000
FMI_EMMCCTL	FMI_BA+0x820	R/W	eMMC Control Register	0x0101_0000
FMI_EMMCCMD	FMI_BA+0x824	R/W	eMMC Command Argument Register	0x0000_0000
FMI_EMMCINTEN	FMI_BA+0x828	R/W	eMMC Interrupt Enable Register	0x0000_0000

FMI_EMMCINTSTS	FMI_BA+0x82C	R/W	eMMC Interrupt Status Register	0x00XX_008C
FMI_EMMCRESP0	FMI_BA+0x830	R	eMMC Receiving Response Token Register 0	0x0000_0000
FMI_EMMCRESP1	FMI_BA+0x834	R	eMMC Receiving Response Token Register 1	0x0000_0000
FMI_EMMCBLEN	FMI_BA+0x838	R/W	eMMC Block Length Register	0x0000_01FF
FMI_EMMCTOUT	FMI_BA+0x83C	R/W	eMMC Response/Data-in Time-out Register	0x0000_0000
FMI_NANDCTL	FMI_BA+0x8A0	R/W	NAND Flash Control Register	0x1E88_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	N/A
FMI_NANDADDR	FMI_BA+0x8B4	W	NAND Flash Address Port Register	N/A
FMI_NANDDATA	FMI_BA+0x8B8	R/W	NAND Flash Data Port Register	N/A
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_NANDECCE0	FMI_BA+0x960	R	NAND Flash ECC Error Data Register 0	0x8080_8080
FMI_NANDECCE1	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_NANDRA_n n = 0, 1..117	FMI_BA+0xA00+0x4*n	R/W	NAND Flash Redundant Area Word n n = 0, 1..117	Undefined

10.5 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eMMC控制器，下面章節會分開描述程序步驟。

10.5.1 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eMMC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，eMMC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SW_RST位。
2. 等待FMI_DMACTL寄存器SW_RST位清除。
3. 設置FMI_CTL寄存器SW_RST位。
4. 等待FMI_CTL寄存器SW_RST位清除。

10.5.2 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，512字節，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4-bit，8-bit，12-bit，15-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_NANDECCE_n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_NANDECCEA_x和FMI_NANDECCED_x寄存器來手動更正錯誤。

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"。

10.5.2.1 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，NAND有二組：GPC0~14和GPI1~15。
 - (1) GPC 設定為SYS_GPC_MFPL 要填入0x55555，SYS_GPC_MFPH 要填入0x05555555。
 - (2) GPI 設定為SYS_GPI_MFPL 要填入0x55555550，SYS_GPI_MFPH 要填入0x55555555。
3. 設置FMI_CTL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或CS1位，選擇控制哪個NAND。

10.5.2.2 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。

10.5.2.3 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）。

10.5.2.4 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	
1 st Cycle	A7	A6	A5	A4	A3	A2	A1	A0	Column Address
2 nd Cycle	L	L	A13	A12	A11	A10	A9	A8	
3 rd Cycle	A21	A20	A19	A18	A17	A16	A15	A14	Row Address Page address: A14~A21 Block Address: A22 ~ L: must be "Low"
4 th Cycle	A29	A28	A27	A26	A25	A24	A23	A22	
5 th Cycle	L	L	L	L	A33	A32	A31	A30	

10.5.2.5 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。

10.5.2.6 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. 將冗餘數據寫入 FMI_NANDRAAn 寄存器有二種方法：
 - (1) 硬件讀取：設置 FMI_NANDCTL 寄存器 REDUN_REN 位，輪詢等待 REDUN_REN 位清除。
 - (2) 軟件讀取：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)。

10.5.2.7 NAND-type Flash ECC 校驗

BCH算法錯誤校正可以校正多達4-bit、8-bit、12-bit、15-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。

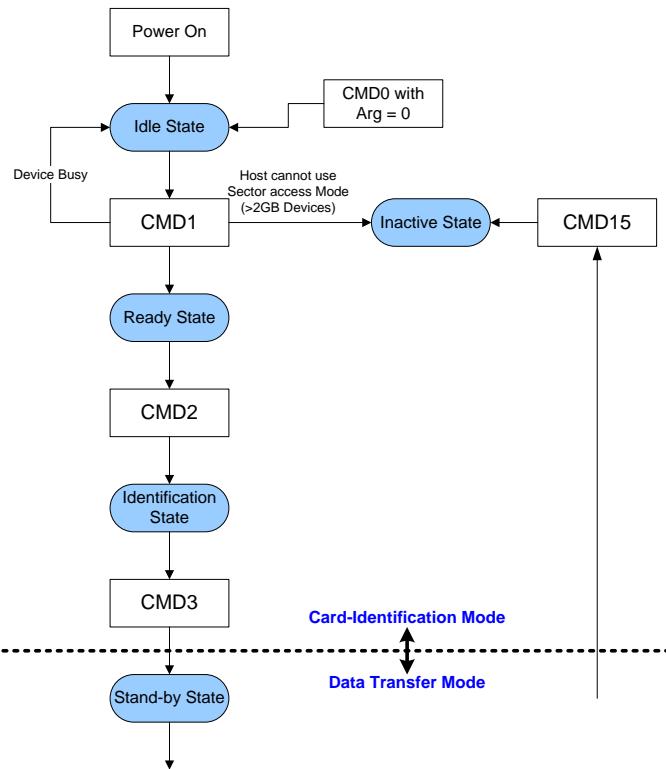
NAND型閃存錯誤校正的步驟：

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_NANDECCEAn和FMI_NANDECEDn寄存器。
4. 從FMI_NANDECEDn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。

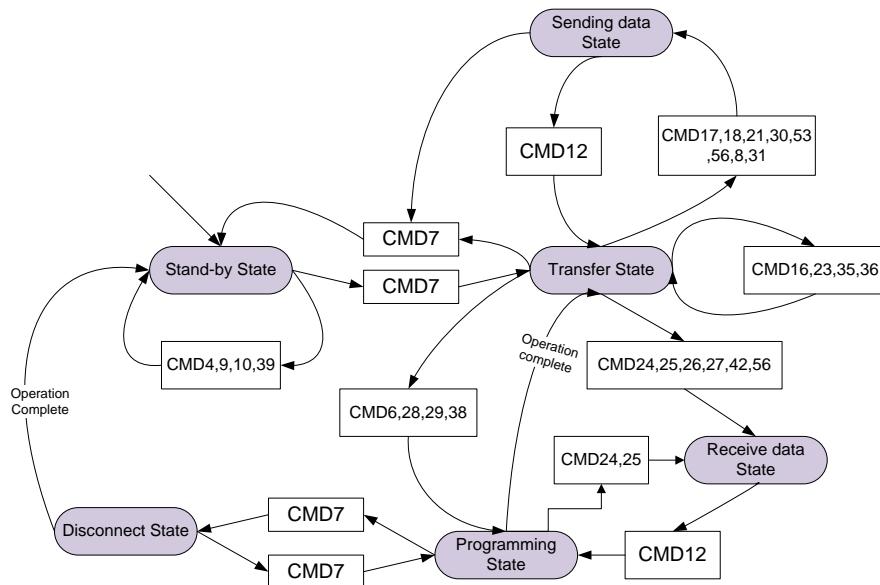
10.5.3 eMMC

FMI提供一個用於eMMC裝置的訪問接口，這個eMMC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_EMMCRSP0和FMI_EMMCRSP1寄存器。關於輸出到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" JEDEC Standard No. 84-A441 "以及各廠商的eMMC datasheet。

eMMC Card Identification Mode :



eMMC Data Transfer Mode :



10.5.3.1 eMMC 初始化

初始化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，eMMC有二組：GPC0~5和GPI5~10。
 - 甲、GPC設定為SYS_GPC_MFPL要填入0x00666666。
 - 乙、GPI設定為SYS_GPI_MFPL要填入0x66600000，SYS_GPI_MFPH要填入0x666。
3. 設置FMI_CTL寄存器eMMC_EN位，始能eMMC功能。
4. 設置FMI_EMMCCTL寄存器SW_RST位。
5. 等待FMI_EMMCCTL寄存器SW_RST位清除。
6. 設定eMMC初始化輸出頻率為300KHz，eMMC接口為1位數據總線模式。
7. 設置FMI_EMMCCTL寄存器CLK74_OE位。
8. 等待FMI_EMMCCTL寄存器CLK74_OE位清除。
9. 根據設備規則發送命令給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，eMMC接口設定為4位數據總線模式。

10.5.3.2 發送 eMMC 命令

發送命令給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CMD_CODE位。
3. 設置FMI_EMMCCTL寄存器CO_EN位，始能命令輸出。
4. 輪詢等待FMI_EMMCCTL寄存器CO_EN位清除。

10.5.3.3 取得 eMMC 響應

得到eMMC響應的步驟如下：

1. 設置FMI_EMMCCTL寄存器RI_EN位，始能響應輸入。
2. 輪詢等待FMI_EMMCCTL寄存器RI_EN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。

10.5.3.4 讀取eMMC

讀取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 CLK8_OE 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BLK_CNT位（一次最多255塊）。
7. 設置CMD18讀取多個塊命令（FMI_EMMCCTL寄存器CMD_CODE位填入18）。
8. 設置FMI_EMMCCTL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 CLK8_OE 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eMMC準備就緒。
13. 發送CMD7使eMMC變成待機狀態。

10.5.3.5 寫入eMMC

寫入 eMMC的步驟如下：

1. 發送CMD7進入傳輸狀態。
2. 設置 FMI_EMMCCTL 寄存器 CLK8_OE 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BLK_CNT位（一次最多255塊）。
7. 設置CMD25寫入多個塊命令（FMI_EMMCCTL寄存器CMD_CODE位填入25）。
8. 設置FMI_EMMCCTL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SW_RST位）。

-
11. 發送CMD12停止傳輸。
 12. 設置 FMI_EMMCCTL 寄存器 CLK8_OE 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 eMMC 準備就緒。
 13. 發送CMD7使eMMC變成待機狀態。

11 通用 DMA 控制器 (GDMA)

11.1 概述

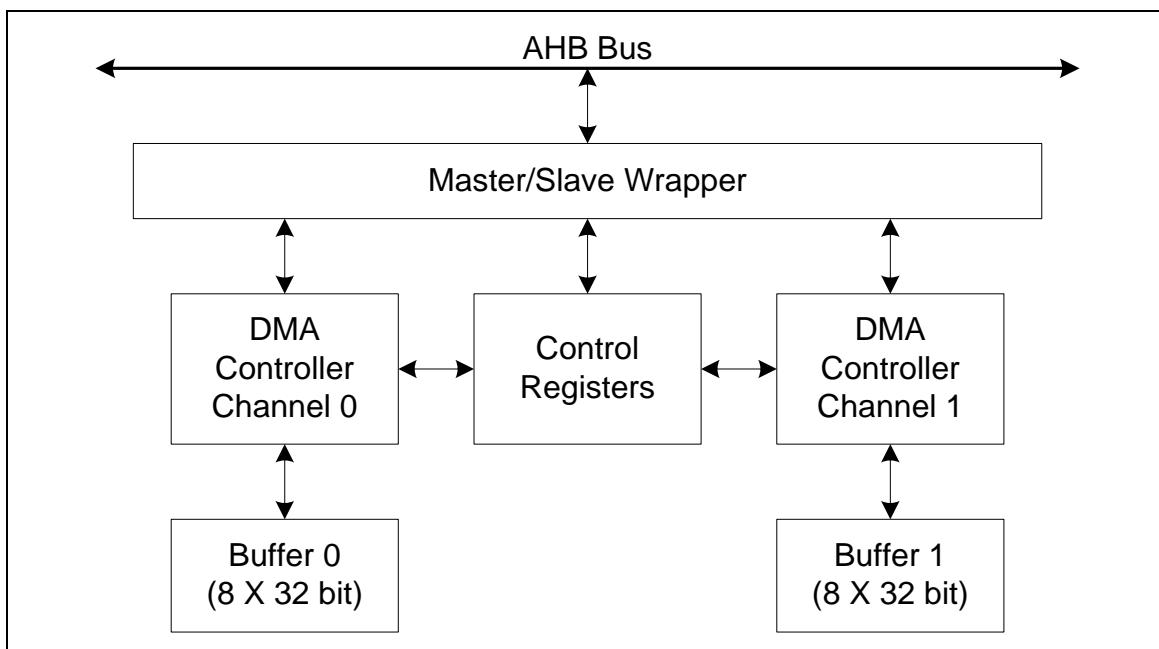
NuMicro™ NUC970 系列 GDMA 有兩個通道控制器，每個通道控制器都支持 Descriptor 或 Non-Descriptor 操作。當 GDMA 通道執行時，不需要 CPU 干預數據傳輸：

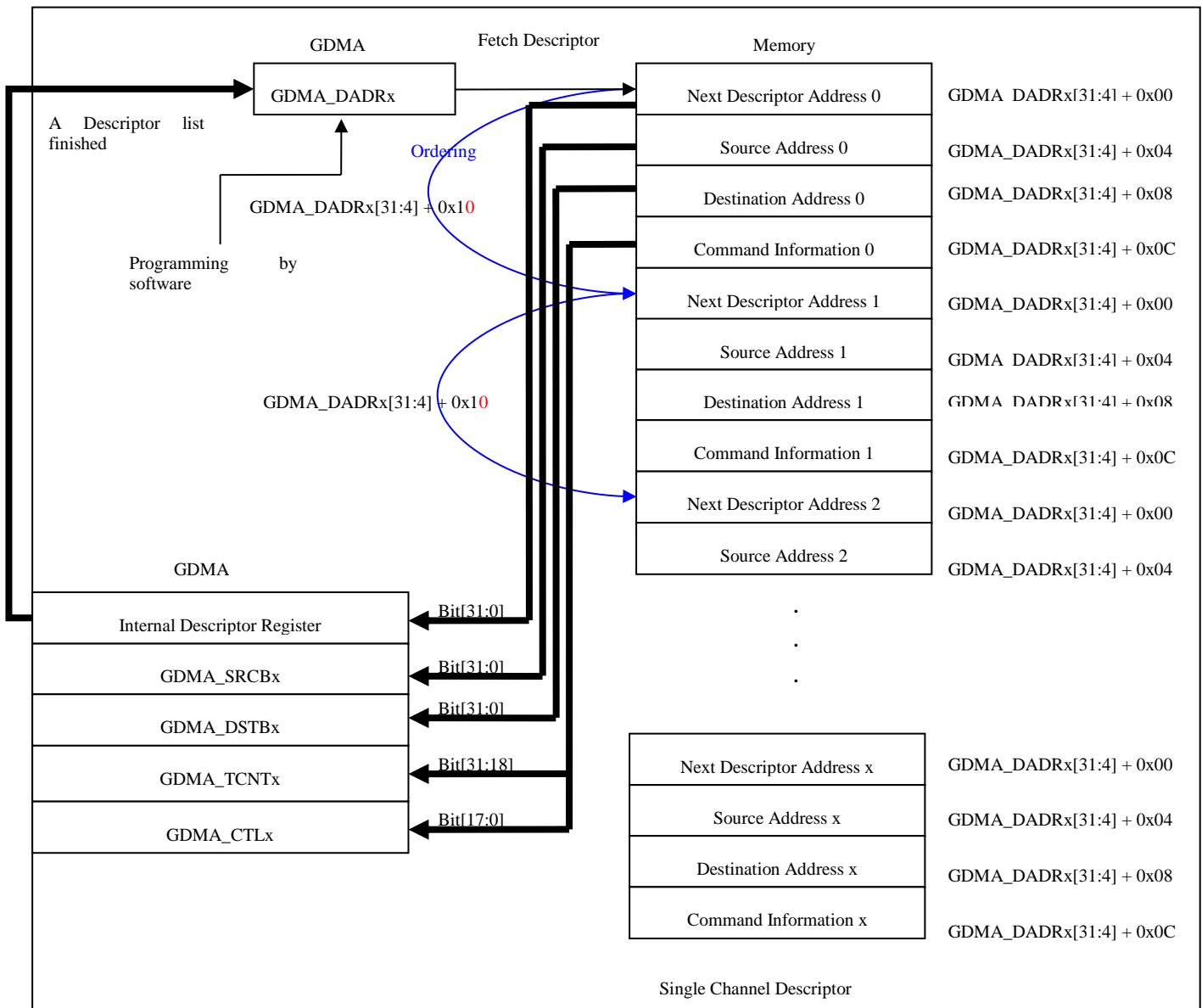
GDMA 可以由軟件來啟動。軟件也可以用它來停止後，重新啟動 GDMA 操作。CPU 可以通過軟件輪詢的方式或接收到一個內部 GDMA 中斷辨識出是否完成 GDMA 操作。在 GDMA 控制器可以對來源位址或目的位址進行遞增或遞減的傳輸，傳輸單位可以支援 8 位元 (byte)，16 位元 (half-word) 或 32 位元 (word) 的數據傳輸。

11.2 特性

- AMBA AHB 兼容
- 支持 Descriptor 和 Non-Descriptor 的功能
- 支持 8 位元的 burst 模式來提高性能
- 提供外部 GDMA 設備

11.3 方塊圖





11.4 寄存器

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

Register	Address	R/W	Description	Reset Value
GDMA_BA = 0xB000_4000				
Channel 0				
GDMA_CTL0	GDMA_BA+0x000	R/W	Channel 0 Control Register	0x0000_0000
GDMA_SRCBA0	GDMA_BA+0x004	R/W	Channel 0 Source Base Address Register	0x0000_0000
GDMA_DSTBA0	GDMA_BA+0x008	R/W	Channel 0 Destination Base Address Register	0x0000_0000

GDMA_TCNT0	GDMA_BA+0x00C	R/W	Channel 0 Transfer Count Register	0x0000_0000
GDMA_CSRCA0	GDMA_BA+0x010	R	Channel 0 Current Source Address Register	0x0000_0000
GDMA_CDSTA0	GDMA_BA+0x014	R	Channel 0 Current Destination Address Register	0x0000_0000
GDMA_CTCNT0	GDMA_BA+0x018	R	Channel 0 Current Transfer Count Register	0x0000_0000
GDMA_DADR0	GDMA_BA+0x01C	R/W	Channel 0 Descriptor Address Register	0x0000_0004
Channel 1				
GDMA_CTL1	GDMA_BA+0x020	R/W	Channel 1 Control Register	0x0000_0000
GDMA_SRCBA1	GDMA_BA+0x024	R/W	Channel 1 Source Base Address Register	0x0000_0000
GDMA_DSTBA1	GDMA_BA+0x028	R/W	Channel 1 Destination Base Address Register	0x0000_0000
GDMA_TCNT1	GDMA_BA+0x02C	R/W	Channel 1 Transfer Count Register	0x0000_0000
GDMA_CSRCA1	GDMA_BA+0x030	R	Channel 1 Current Source Address Register	0x0000_0000
GDMA_CDSTA1	GDMA_BA+0x034	R	Channel 1 Current Destination Address Register	0x0000_0000
GDMA_CTCNT1	GDMA_BA+0x038	R	Channel 1 Current Transfer Count Register	0x0000_0000
GDMA_DADR1	GDMA_BA+0x03C	R/W	Channel 1 Descriptor Address Register	0x0000_0004
GDMA_BUFFER0	GDMA_BA+0x080	R	GDMA Internal Buffer Word 0 Register	0x0000_0000
GDMA_BUFFER1	GDMA_BA+0x084	R	GDMA Internal Buffer Word 1 Register	0x0000_0000
GDMA_BUFFER2	GDMA_BA+0x088	R	GDMA Internal Buffer Word 2 Register	0x0000_0000
GDMA_BUFFER3	GDMA_BA+0x08C	R	GDMA Internal Buffer Word 3 Register	0x0000_0000
GDMA_BUFFER4	GDMA_BA+0x090	R	GDMA Internal Buffer Word 4 Register	0x0000_0000
GDMA_BUFFER5	GDMA_BA+0x094	R	GDMA Internal Buffer Word 5 Register	0x0000_0000
GDMA_BUFFER6	GDMA_BA+0x098	R	GDMA Internal Buffer Word 6 Register	0x0000_0000
GDMA_BUFFER7	GDMA_BA+0x09C	R	GDMA Internal Buffer Word 7 Register	0x0000_0000
GDMA_INTS	GDMA_BA+0x0A0	R/W	GDMA Interrupt Control and Status Register	0x0000_0000

11.5 功能描述.

11.5.1 Non-Descriptor 功能描述

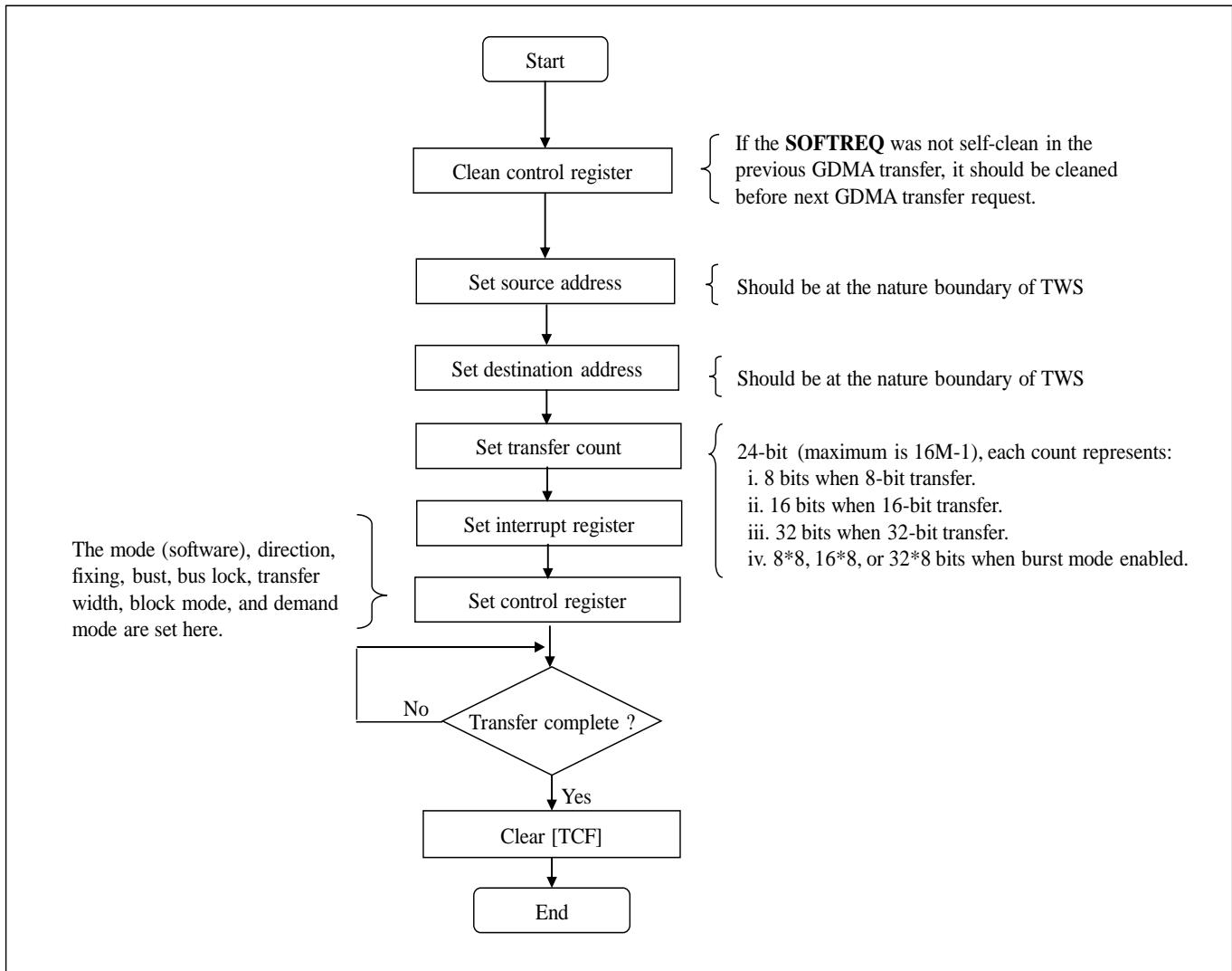
11.5.1.1 GDMA的配置

每個GDMA通道都有一個控制寄存器，一個descriptor地址寄存器，兩個基本寄存器和一個傳輸

計數寄存器。這些寄存器應在數據傳輸開始前被正確的編程。其中最重要的是控制寄存器(GDMAx_CTL)。它用來控制GDMA運作的行為，例如在傳輸模式和傳輸寬度。如下圖列出Non-Descriptor的GDMA_CTL的控制寄存器。每個bit的詳細描述都可以在技術參考手冊中找到。

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVE D	SABNDE RR	DABNDE RR	RESERV ED	AUTOIE N	RESERV ED	BLOCK	SOFTRE Q
15	14	13	12	11	10	9	8
RESERVED		TWS		SBMS	RESERVED		
7	6	5	4	3	2	1	0
SAFIX	DAFIX	SADIR	DADIR	GDMAMS		BME	GDMAE N

來源寄存器(GDMAx_SRCB)用於設定來源位址。目標位址寄存器(GDMAx_DSTB)用於設定起始地址。GDMA傳輸的數量是由傳輸計數寄存器(GDMAx_TCNT)來設置。下圖顯示了編程流程GDMA的Non-Descriptor操作。



11.5.1.2 傳輸數量(Transfer Count)

GDMAx_TCNT主要是設定傳輸的數量，而不是傳輸的bytes數量。

Transferred bytes = [GDMAx_TCNT] * Transfer width /* burst mode is disabled */

假設GDMAx_TCNT= 6和傳送的寬度為16，TWS(GDMAx_CTL[13:12]) = 1。傳輸的bytes數量應該為 $16 \times 2 = 32$ 。但是，如果burst模式被使能則上面的公式將被如下改變。

Transferred bytes = [GDMAx_TCNT] * Transfer width * 8 /* burst mode is enabled */

在此情況下的burst模式被使能，在上述實施例的傳送bytes應為 $16 \times 2 \times 8 = 256$

11.5.1.3 傳輸終止(Transfer Termination)

當GDMA完成傳輸時，它將設置寄存器GDMA_INTCS中的[TCxF],x=0,1，並產生一個中斷請求，並使中斷使能。程序使用者可以輪詢[TCxF]或等待的GDMA中斷發生知道傳送完成。需要注意的是程序使用者必須清除位[TCxF]，清除該中斷請求，讓接下來的GDMA運作繼續。

11.5.1.4 固定地址(Block Mode Transfer)

GDMA以連續增加或減少來源地址和以連續增加或減少目的地址，來做數據傳輸期間。GDMA控制器提供另一功能，以支持固定來源/目的地址來執行系統存儲器和外部設備之間的數據傳輸。做一個儲存器到I/O傳輸，DAFIX(GDMA_CTL[6])應被設置。如果I/O到儲存器傳送，SAFIX(GDMA_CTL[7])應被設置。

11.5.1.5 區塊模式傳輸(Block Mode Transfer)

當GDMA被設定到區塊模式，SBMS(GDMAx_CTL[11]) = 1，它僅需要一個請求給所有的傳輸數據。當接收SOFTREQ被設置，GDMA開始傳輸數據。後已傳輸的寄存器GDMAX_TCNT指定的數據的數量時，GDMA_INTS寄存器中TCxF被設置，並產生一個中斷，則GDMA停止，直到接收到下一個請求。

11.5.1.6 軟件啟動GDMA

GDMA可以配置為軟件模式，以執行存儲器到存儲器的傳送。在此模式下，傳輸操作可以由GDMA控制寄存器(GDMAx_CTL)中設置開始時，來源地址(GDMAx_SRCB)、目的地址(GDMAx_DSTB)和傳輸計數(GDMAx_TCNT)的設置應在先前進行配置。軟件模式的編程方法如下：

1. GDMA以軟件模式，GDMAMS(GDMAx_CTL[3:2])=0x0。
2. 設定來源地址(GDMAx_SRCB)、目的地址(GDMAx_DSTB)和傳輸計數(GDMAx_TCNT)。
3. 設定SOFTREQ(GDMAx_CTL[16])=1。
4. 設定SGDMAEN(GDMAx_CTL[0])=1，GDMA開始運作。

在軟件模式下，SOFTREQ (GDMAx_CTL[16])和GDMAEN (GDMAx_CTL[0])是當GDMA傳送完成則會自動清除為0。然而如果AUTOIEN位被設置，則GDMAEN不會自動清除為0。因此，程序使用者只需要設置位SOFTREQ開始下一個數據傳輸。如果GDMA沒有完成傳輸，它會導致GDMA傳輸錯誤的旗標被設置為1，並且SOFTREQ不會自行清除。在這種情況下，SOFTREQ應該在下一次軟件GDMA請求之前被清除。

來源位址和目的位址必須根據傳輸寬度對齊。例如，如果傳輸寬度為32位，在源和目標基地址應該是word對齊。如果沒有對齊，則GDMA的SABNDERR和DABNDERR，將被設置為1。

示範一個GMDS Non-Descriptor 傳送，如下：

```
#define BASE 0xB0004000
#define GDMA_CTL0    (BASE)
#define GDMA_SRCB0   (BASE+0x04)
#define GDMA_DSTB0   (BASE+0x08)
#define GDMA_TCNT0   (BASE+0x0C)
```

```
#define GDMA_INTCS  (BASE+0xA0)
void main(void)
{
    //Clear GDMA_CTL0 register
*((volatile int *) GDMA_CTL0)=0x0;

    //Set source base address
*((volatile int *) GDMA_SRCB0)= 0xc2000000;

    //Set destination base address
*((volatile int *) GDMA_DSTB0)= 0Xc2001000;

    //Set transfer count
*((volatile int *) GDMA_TCNT0)=0x10;

    //Enable GDMA operation
*((volatile int *) GDMA_CTL0)=0x12001;

    //Waiting for GDMA transfer finish
while(!(*((volatile int *)GDMA_INCS) & 0x100);

    //Clear interrupt flag
*((volatile int *)GDMA_INTCS=0x100;
}
```

11.5.2 Descriptor 功能描述

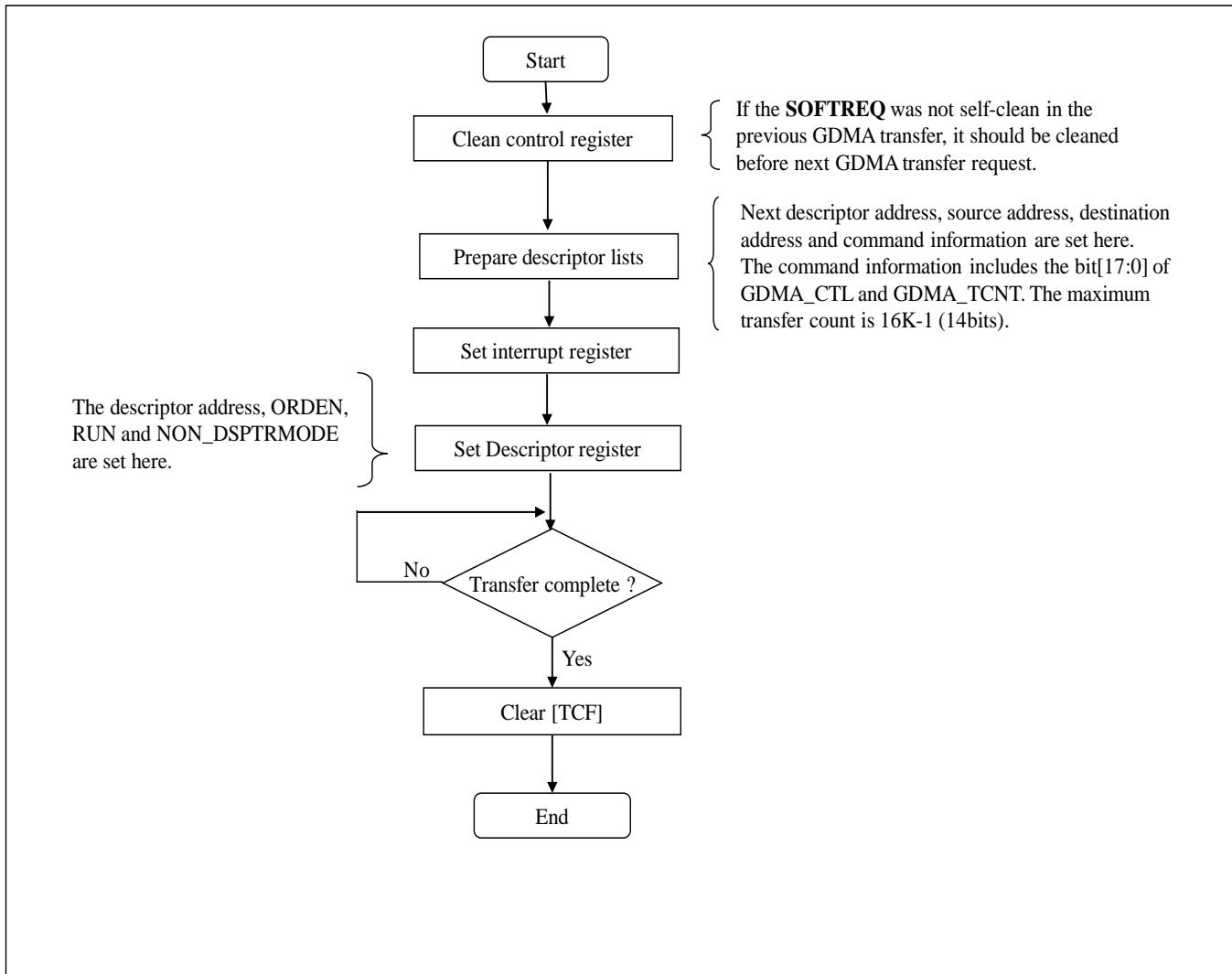
11.5.2.1 GDMA的配置

每個GDMA通道有一個控制寄存器，一個descriptor地址寄存器，兩個基本寄存器和一個傳輸計數寄存器。這些寄存器應在數據傳輸開始前被正確編程。其中最重要的是控制寄存器(GDMAx_CTL)。它用來控制GDMA運作的行為，例如在傳輸模式和傳輸寬度。如下圖列出Descriptor的GDMA_CTL的控制寄存器。每個bit的詳細描述都可以在技術參考手冊中找到。

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16

RESERVE D	SABNDE RR	DABNDE RR	RESERVED			BLOCK	SOFTRE Q
15	14	13	12	11	10	9	8
RESERVED		TWS		RESERV ED	D_INTS	RESERVED	
7	6	5	4	3	2	1	0
SAFIX	DAFIX	SADIR	DADIR	GDMAMS		BME	GDMAE N

31	30	29	28	27	26	25	24
Descriptor Address [31:24]							
23	22	21	20	19	18	17	16
Descriptor Address [23:16]							
15	14	13	12	11	10	9	8
Descriptor Address [15:8]							
7	6	5	4	3	2	1	0
Descriptor Address [7:4]				RUN	NON_DSPTRM ODE	ORDEN	RESET



11.5.2.2 軟件啟動GDMA

GDMA可以配置為軟件模式，以執行存儲器到存儲器的傳送。在此模式下，傳輸操作可以由GDMA控制寄存器(GDMAx_CTL)中設置開始時，Descriptor列表應在開始前先進進行編程。軟件模式的編程方法如下：

1. 設定下一個Descriptor地址寄存器GDMA_DADR。如果GDMA_DADR寄存器的ORDEN被設定，Descriptor列表的下一個Descriptor地址是不必要的。否則，下一個Descriptor地址應填充Descriptor列表。在RUN應該在Descriptor列表中設置，除了最後一個Descriptor。在最後一個Descriptor列表，RUN和ORDEN應該被清為0，NON_DSPRTMODE應設置。
2. 設定來源地址(GDMAx_SRCB)、目的地址(GDMAx_DSTB)和傳輸計數(GDMAx_TCNT)。
3. 設定命令信息(Command Information)，命令信息中 bits([31:18])將被寫回到GDMAx_TCNT寄存器，bits([17:0]) 將被寫回到GDMAx_CTL寄存器。

在軟體模式下，SOFTREQ(GDMAx_CTL[16])和GDMAEN(GDMAx_CTL[0])是當GDMA傳送完

成則會自動清除為0。然而如果AUTOIEN位被設置，則GDMAEN不會自動清除為0。因此，程序使用者只需要設置位SOFTREQ開始下一個數據傳輸。如果GDMA沒有完成傳輸，它會導致GDMA傳輸錯誤的旗標被設置為1，並且SOFTREQ不會自行清除。在這種情況下，SOFTREQ應該在下一次軟件GDMA請求之前被清除。來源位址和目的位址必須根據傳輸寬度對齊。例如，如果傳輸寬度為32位，在源和目標基地址應該是word對齊。如果沒有對齊，則GDMA的SABNDERR和DABNDERR，將被設置為1。

示範一個GMDA Non-Descriptor 傳送，如下：

```
#define BASE 0xB0004000
#define GDMA_DADR0    (BASE+0x1C)
#define GDMA_INCS (BASE+0xA0)

typedef struct_dma_desc
{
    unsigned int nextDescAddr;
    unsigned int srcBufAddr;
    unsigned int dstBufAddr;
    unsigned int comInfo;
}GDMA_DESC;

void main(void)
{
    unsigned int listaddr=0x100000,i;
    GDMA_DESC *descp0 =(GDMA_DESC *)listaddr;
    for(i=0;i<10;i++)
    {
        descp0->nextDescAddr=0x0A;
        descp0->srcBufAddr = 0x200000+(0x10*4*i);
        descp0->dstbufAddr = 0x300000+(0x10*4*i);
        descp0->cominfo = 0x412401;
        descp0++;
    }
    descp0--;
    descp0->nextdescAddr=0x04;
    *((volatile unsigned int *)GDMA_DADR0) = listaddr | 0x0A;
    while(!(*((volatile unsigned int *)GDMA_INTCS)&0x100));
    *((volatile unsigned int *)GDMA_INCS)=0x100;
}
```

12 2D 圖形加速

12.1 概述

32位2D圖形引擎（GE2D）是專門設計來提高圖形處理性能。它可以加速個體的GUI功能，諸如BitBLTs和布氏線繪製的操作，可在8/16/32位每像素進行操作。

像素是最小的可尋址的屏幕元素如Microsoft Windows中所定義，以及線和圖片是由多種像素構成。 GE2D用於加快在像素數據的移動和畫線的圖形的性能，以及通過消除CPU開銷，以加速計算機圖形布爾運算。

同時，旋轉和按比例縮小的功能被實現於一些特殊的應用。在圖像縮小功能方面，提供調整大小的圖像的N/M縮小因子。

二維旋轉，它可以向左旋轉或向右45，90或180度，並且還支持鏡像或上下相反的功能。

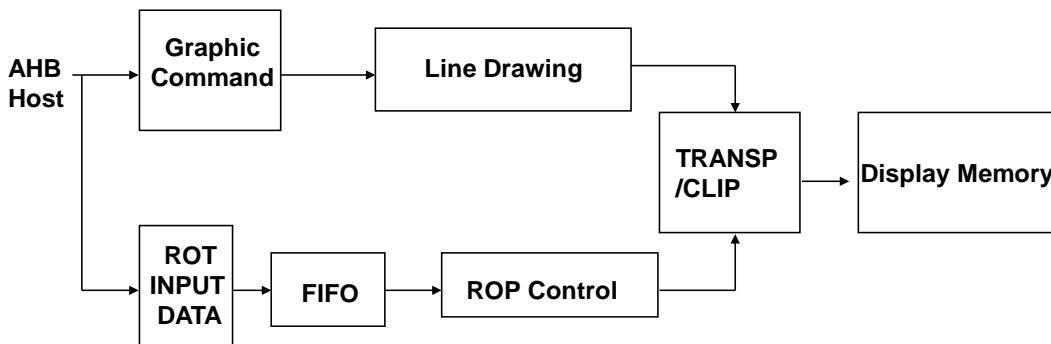
12.2 特性

- 支援由微軟公司定義之2D位塊傳送(BitBLT)功能
- 支援主位塊傳送功能
- 支援圖案位塊傳送功能
- 支援顏色/字型位塊傳送功能
- 支援透明位塊傳送功能
- 支援瓦位塊傳送功能
- 支援矩形填充
- 支援RGB332/565/888數據格式
- 支援前/背景顏色及微軟256光柵操作碼(ROP)
- 支援目標範圍裡/外剪貼功能
- 支援alpha混合
- 支援快速Bresenham畫線演算法
- 支援影像大小調整
- 支援縮小功能(倍率：1/255 ~ 254/255)
- 支援放大功能(倍率：1 ~ 1.996)
- 支援物件旋轉
- 支援左右鏡射功能

- 支援上下翻轉功能

12.3 方塊圖

Example: Host -> Graphics Engine -> Display Memory



12.4 寄存器

Register	Address	R/W	Description	Reset Value
GE2D_BA = 0xB000_B000				
GE2D_TRG	GE2D_BA+0x000	R/W	Graphic Engine Trigger Control Register	0x0000_0000
GE2D_XYSORG	GE2D_BA+0x004	R/W	Graphic Engine XY Mode Source Origin Starting Address Register	0x0000_0000
GE2D_TCNTVHSF	GE2D_BA+0x008	R/W	Graphic Engine Tile Count or Vertical/Horizontal Scale Factor Register	0x0000_0000
GE2D_XYRRP	GE2D_BA+0x00C	R/W	Graphic Engine XY Mode Rotate Reference Pixel Coordinate Register	0x0000_0000
GE2D_INTSTS	GE2D_BA+0x010	R/W	Graphic Engine Interrupt Status Register	0x0000_0000
GE2D_PATSA	GE2D_BA+0x014	R/W	Graphic Engine Pattern Location Starting Address Register	0x0000_0000
GE2D_BETSC	GE2D_BA+0x018	R/W	Graphic Engine Bresenham Error Term Stepping Constant Register	0x0000_0000
GE2D_BIEPC	GE2D_BA+0x01C	R/W	Graphic Engine Bresenham Initial Error Term, Pixel Count Register	0x0000_0000
GE2D_CTL	GE2D_BA+0x020	R/W	Graphic Engine Control Register	0x0000_0000
GE2D_BGCOLR	GE2D_BA+0x024	R/W	Graphic Engine Background Color Register	0x0000_0000
GE2D_FGCOLR	GE2D_BA+0x028	R/W	Graphic Engine Foreground Color Register	0x0000_0000
GE2D_TRNSCOLR	GE2D_BA+0x02C	R/W	Graphic Engine Transparency Color Register	0x0000_0000
GE2D_TCMSK	GE2D_BA+0x030	R/W	Graphic Engine Transparency Color Mask Register	0x0000_0000

GE2D_XYDORG	GE2D_BA+0x034	R/W	Graphic Engine XY Mode Display Memory Origin Starting Register	0x0000_0000
GE2D_SDPICTH	GE2D_BA+0x038	R/W	Graphic Engine Source/Destination Pitch Register	0x0000_0000
GE2D_SRCSPA	GE2D_BA+0x03C	R/W	Graphic Engine Source Start Pixel/Address Register	0x0000_0000
GE2D_DSTSPA	GE2D_BA+0x040	R/W	Graphic Engine Destination Start Pixel/Address Register	0x0000_0000
GE2D_RTGLSZ	GE2D_BA+0x044	R/W	Graphic Engine Rectangle Size Register	0x0000_0000
GE2D_CLPBTL	GE2D_BA+0x048	R/W	Graphic Engine Clipping Boundary Top/Left Register	0x0000_0000
GE2D_CLPBRR	GE2D_BA+0x04C	R/W	Graphic Engine Clipping Boundary Bottom/Right Register	0x0000_0000
GE2D_PTNA	GE2D_BA+0x050	R/W	Graphic Engine Pattern Group A Register	0x0000_0000
GE2D_PTNB	GE2D_BA+0x054	R/W	Graphic Engine Pattern Group B Register	0x0000_0000
GE2D_WRPLNMSK	GE2D_BA+0x058	R/W	Graphic Engine Write Plane Mask Register	0x0000_0000
GE2D_MISCTL	GE2D_BA+0x05C	R/W	Graphic Engine Miscellaneous Control Register	0x0000_0000
GE2D_GEHBDW0	GE2D_BA+0x060	R/W	Graphic Engine HostBLT Data Port 0 Register	0x0000_0000
GE2D_GEHBDW1	GE2D_BA+0x064	R/W	Graphic Engine HostBLT Data Port 1 Register	0x0000_0000
GE2D_GEHBDW2	GE2D_BA+0x068	R/W	Graphic Engine HostBLT Data Port 2 Register	0x0000_0000
GE2D_GEHBDW3	GE2D_BA+0x06C	R/W	Graphic Engine HostBLT Data Port 3 Register	0x0000_0000
GE2D_GEHBDW4	GE2D_BA+0x070	R/W	Graphic Engine HostBLT Data Port 4 Register	0x0000_0000
GE2D_GEHBDW5	GE2D_BA+0x074	R/W	Graphic Engine HostBLT Data Port 5 Register	0x0000_0000
GE2D_GEHBDW6	GE2D_BA+0x078	R/W	Graphic Engine HostBLT Data Port 6 Register	0x0000_0000
GE2D_GEHBDW7	GE2D_BA+0x07C	R/W	Graphic Engine HostBLT Data Port 7 Register	0x0000_0000

12.5 功能描述

12.5.1 初始化 2D 控制器

2D 控制器提供了加速計算機圖形處理，該處理使圖像在屏幕顯示裝置上正確地顯示。因此，2D 控制器還涉及有關屏幕顯示裝置的信息，如屏幕的高度和寬度。

該信息涉及設置某些寄存器的狀態，所以這些信息必須是正確的，讓2D控制器得到正確的數據，並在顯示內存正常地操作。

2D圖形引擎的初始化包括下列步驟：

1. 設置全局變量，包括每像素所需位元數，屏幕的高度和寬度。
2. 分配顯示(display)和圖案(pattern)存儲器的空間。顯示存儲器的分配的大小是根據每像素所需的比特數，屏幕的高度和寬度，存儲器地址的對準以64K字節存儲器邊界來計算。在默認情況下，XY模式下的顯示內存起始地址需設置在2D_GEXYSORG寄存器中，且源地址的內存起始地址應該是64K字節邊界。圖案尺寸是8x8像素，因此，圖案分配的內存大小是8x8乘以每像素為八個比特。該模式的內存地址必須在一個M字節邊界編程。 $M = 8 \times 8 \times BPP / 8$ 個字節，其中 $BPP = 8/16/32$ 。

3. 透過配置GE2D(CLK_HCLKEN[28])，啟動2D控制器的時鐘。
4. 設定GE2DIF(GE2D_INTSTS[0])位，清除中斷旗標。設定GE2IEN(GE2D_CTL[17])位，使用輪詢(非中斷)模式。
5. 將圖案的源地址填入GE2D_PATSA寄存器中，將來源及目的顯示地址填入GE2D_SRCSPA/GE2D_DSTSPA寄存器。
6. 透過設置GE2D_WRPLNMSK寄存器來使能將要顯示的比特數，如RGB888，請填入0xFFFFFFF值。
7. 設置BPP(GE2D_MISCTL[5:4])決定顯示像素值，以便讓2D控制器正確的讀取位元數。

下面列出2D控制器的初始化編程樣本：

```
GFX_BPP = bpp;           // 每像素比特數
GFX_WIDTH = width;      // 顯示寬度
GFX_HEIGHT = height;    // 顯示高度

GFX_PITCH = (GFX_WIDTH*(GFX_BPP/8));
GFX_SIZE = (GFX_HEIGHT*GFX_PITCH);
GFX_START_ADDR = (void *)malloc(GFX_SIZE+65536); // 取得顯示緩存
GFX_START_ADDR = (void *)shift_pointer((int)GFX_START_ADDR, 65536); // 64k字節邊界
GFX_PAT_ADDR = (void *)malloc((8*8*(GFX_BPP/8))*2); // 取得圖案緩存
GFX_PAT_ADDR = (void *)shift_pointer((int)GFX_PAT_ADDR, (8*8*(GFX_BPP/8))*2); // 圖案緩存位址邊界對齊

CLK_HCLKEN |= (0x1 << 28); // 啟動控制器時鐘
GE2D_INTSTS |= 0x1;          // 清除中斷旗標
G2DE_CTL = 0;                // 取消中斷模式(使用輪詢模式)

GE2D_PATSA = GFX_PAT_ADDR;   // 圖案源地址
GE2D_SRCSPA = GFX_START_ADDR; // 顯示源地址
GE2D_DSTSPA = GFX_START_ADDR; // 顯示目的地址
GE2D_WRPLNMSK = 0x00ffff;

GE2D_MISCTL = GE2D_MISCTL & ~(0x3 << 4) | bpp; // 決定顯示bpp數
```

12.5.2 三元光柵操作(ROP)

2D控制器支持所有微軟256三元光柵操作碼。三元光柵操作碼是包含在位塊傳輸功能中，三元光柵操作碼定義了BitBLT如何結合位在源圖中的圖案及位在目標圖中。每一光柵操作代碼表示一個布林操作，來將源像素值，目的地的像素值進行合併。

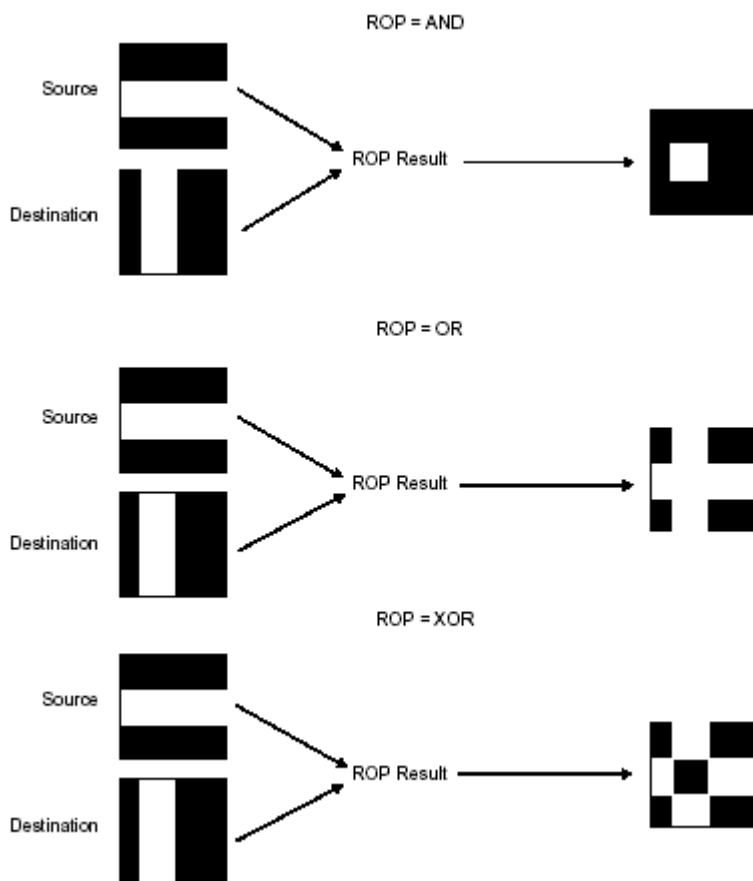
最常用的光柵操作常會被賦予特定的名稱，下表列出比較常用的操作及其名稱。

Boolean function	Common name
------------------	-------------

0x00	BLACKNESS
0x11	NOTSRCERASE
0x33	NOTSRCCOPY
0x44	SRCERASE
0x55	DSTINVERT
0x5A	PATINVERT
0x66	SRCINVERT
0x88	SRCAND
0xBB	MERGEPAINT
0xC0	MERGECOPY
0xCC	SRCCOPY
0xEE	SRCPAINT
0xF0	PATCOPY
0xFB	PATPAINT
0xFF	WHITENESS

光柵操作在BitBLT的操作期間始終有效，必須裝入適當的值至相關暫存器。程序員應該設置ROP(GE2D_CTL[31:24])位，填入適當的值。

下圖舉例光柵操作基本的運作原理。

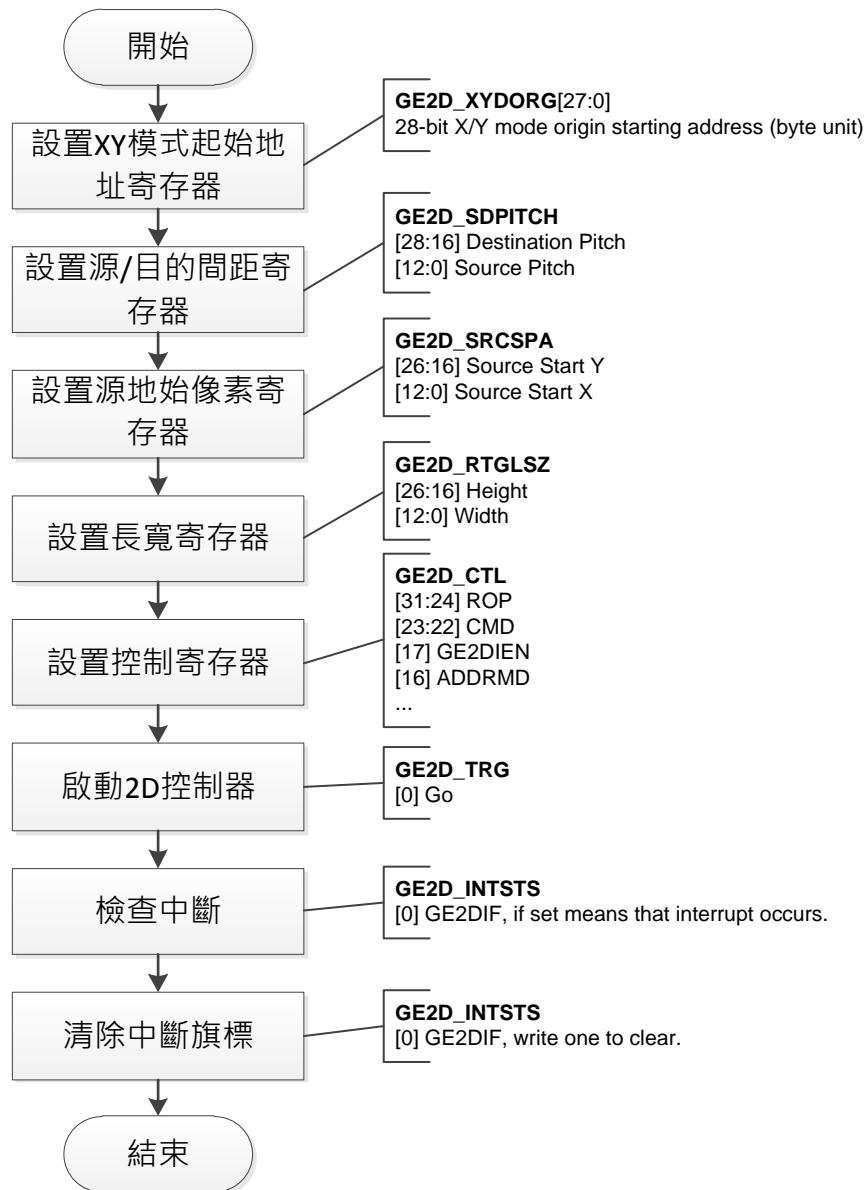


12.5.3 位塊傳送(BitBLT)

BitBLT功能執行了對應於從指定的源上下文到顯示內存像素的矩形的圖像數據內的位塊傳輸。且BitBLT加速了數據顯示存儲器區域之間，或介於系統內存和顯示內存的傳輸操作。所述BitBLT操作三個像素圖(操作數)：源，圖案，和目的地，並提供了所有256個可能的光柵操作(ROP)。圖形引擎可支持多種BitBLT，包括主BLT，圖案BLT，彩色/字體擴大BLT，透明BLT，顏色/字體擴張，矩形填充等。

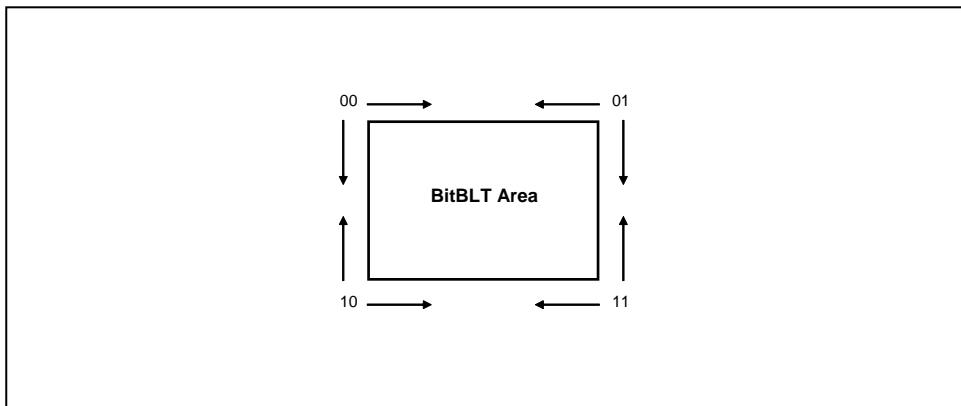
源數據可以位於顯示內存或系統內存，既可以是彩色或黑白。8x8圖案數據如果它是彩色數據時，可以位於顯示內存中，如果它是單色數據時可以來自內部圖案寄存器。目的地數據必須位在顯示內存中。由控制器所產生的所得目的地數據會被寫回到顯示內存中，或者其可以被CPU讀回。

其下為一般操作BitBLT的流程：



在流程圖中，顯示出控制2D控制器的動作和行為，並解釋了哪些寄存器被使用。在實際上，在設置控制寄存器之前，流程圖中的步驟順序是可以改變的。

BitBLT功能的實現幾乎都是使用X/Y尋址模式。在X/Y尋址模式中，所有源啟始X，目標起始X，尺寸X，表示以像素為單位。源間距和目標間距X/Y像素的處理都是相同的。除了BitBLT的方向上，BitBLT的方向表示，在X/Y地址上跨越矩形的方向。它還定義了傳輸的起始角落，這是為了突顯如果目標矩形和源矩形重疊的時候，透過該起始角落操作的進行與定義，使源區數據不會被覆蓋。BitBLT的方向控制顯示在下面的圖中：

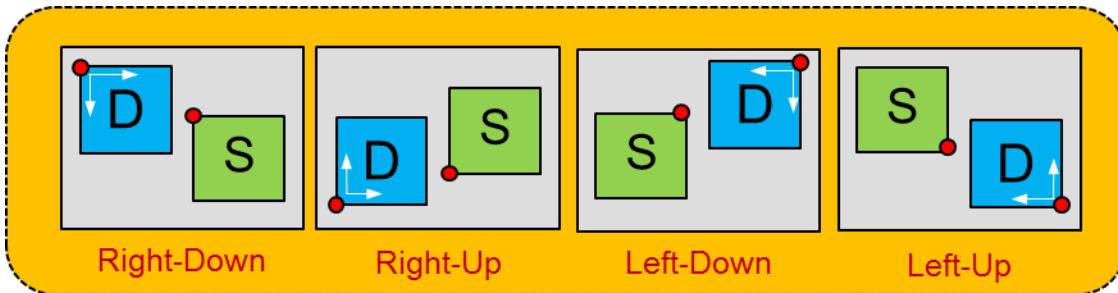


畫的方向的決定與源和目的起始位置相關，下圖為簡易決定的算法及示意圖(S為源位置，D為目的位置)。

```

if((srcx > dstx) && (srcy > dsty)) direction = b'000; // (右-下)
if((srcx > dstx) && (srcy < dsty)) direction = b'100; // (右-上)
if((srcx < dstx) && (srcy > dsty)) direction = b'010; // (左-下)
if((srcx < dstx) && (srcy < dsty)) direction = b'110; // (左-上)
(srcx=源x起始位置，srcy=源y起始位置)
(dstx=目的x起始位置，dsty=目的y起始位置)

```



下面的例子說明了一個圖像是使用BitBLT的功能且數據存放於顯示內存中，其中源和目標數據是來自顯示內存中並且兩者一樣。一個BitBLT功能的動作大致可包含以下步驟：

1. 設定源X/Y起始位置，於2DGE_XYDORG寄存器。
2. 設定基本控制值 0xCC430000，這個控制值表示，0xCC設定光柵操作為源拷貝模式，並設 CMD(GE2D_CTL[23:22]) 位為 BitBLT 加速，GE2DIEN(GE2D_CTL[17]) 位為啟用中斷，ADDRMD(GE2D_CTL[16])位為X/Y模式，最後透過設置DRAWDIR(GE2D_CTL[3:1])確定方向。

3. 設置GE2D_SDPICTH寄存器以決定源和目的在X/Y模式下的使用像素為單位的間距。
4. 設置GE2D_SRCSPA寄存器來決定源在X/Y模式下的使用像素為單位的起始位置，GE2D_DSTSPA寄存器來決定目的在X/Y模式下的使用像素為單位的起始位置。
5. 設置GE2D_RTGLSZ寄存器來決定在X/Y模式下的使用像素為單位的寬和高。
6. 決定其它在GE2D_CTL內需要設置的功能。
7. 設置GO(GE2D_TRG[0])，啟動2D加速控制器。
8. 檢查GE2DIF(GE2D_INTSTS[0])位，以確定控制器已完成工作。
9. 透過寫1至GE2DIF位，清除中斷旗標。

下面是一個簡單的編程樣本：

```
GE2D_SRCSPA = GFX_START_ADDR; //顯示源地址
GE2D_DSTSPA = GFX_START_ADDR; //顯示目的地址

u32cmd = 0xCC430000;

// 決定方向
if (srcx > destx) {      //+X
    if (srcy > desty) {  //+Y
        direction = PP;           // direction is 000
    } else {                //-Y
        u32cmd |= 0x08;         // direction is 100
        srcy = srcy + height - 1;
        desty = desty + height - 1;
    }
}
else {                  //-X
    if (srcy > desty) {  //+Y
        u32cmd |= 0x04;         // direction is 010
        srcx = srcx + width - 1;
        destx = destx + width - 1;
    } else {                //-Y
        u32cmd |= 0xc;          // direction is 110
        srcx = srcx + width - 1;
        destx = destx + width - 1;
        srcy = srcy + height - 1;
        desty = desty + height - 1;
    }
}
```

```
GE2D_CTL = u32cmd;
pitch = GFX_WIDTH << 16 | GFX_WIDTH; // 設置源/目的間隔
GE2D_SDPITCH = pitch;

src_start = srcy << 16 | srcx;           // 設置X/Y源起始位置
GE2D_SRCSPA = src_start;

dest_start = desty << 16 | destx;         // 設置X/Y目的起始位置
GE2D_DSTSPA = dest_start;

dimension = height << 16 | width;        // 設置寬和高
GE2D_RTGLSZ = dimension;

GE2D_CTL = cmd32;
GE2D_TRG = 1;                // 啟動2D加速控制器
while ((GE2D_INTSTS & 0x01)==0);    // 等待動作結束

GE2D_INTSTS = 1;             // 清除中斷旗標
```

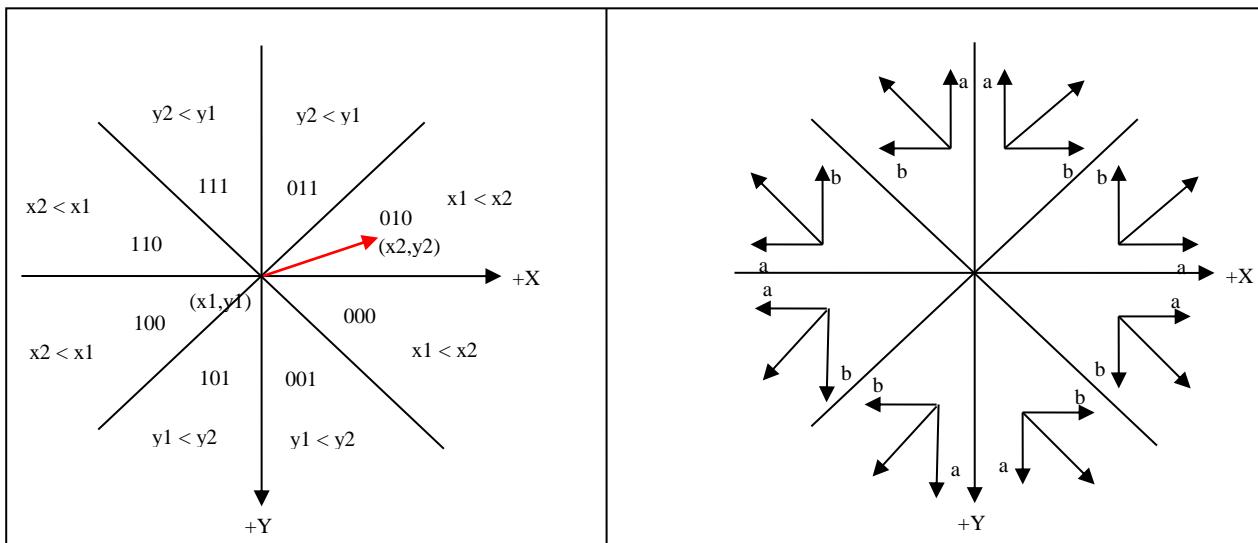
12.5.4 Bresenham 畫線

Bresenham線描是一個演算法則，它決定點對應，以便繪製出兩個給定的點之間最近的直線。這種算法只使用整數加法，減法和移位，對計算機結構來說是一種很成本很低的操作。因此，Bresenham線描有高效率，及反應快速的特性。

布氏線描算法被用來繪製從屏幕坐標X1，Y1到X2，Y2，一個像素寬的固體或紋理化線。要繪製實線，前景色用於指定該行的顏色。繪製一個紋理化線，一個16位的線型圖案用於指定線的圖案，裡面所有的1被擴展到前景顏色而所有的0被擴展至背景顏色或者是透明顏色。16位線型模式位LNEPTN/ABLDFT是在2DGE_MISCTL[31:16]寄存器中。

Bresenham畫線算法的參數操作被標準化到一個八象限中，一個3位八象限代碼顯示如下左圖。

如顯示在下面的右圖可以看出，Bresenham線描可以處理多個象限。所有的(x₁,y₁)到(x₂,y₂)有8個對應象限而Bresenham畫線算法可以工作在其中之一象限中。

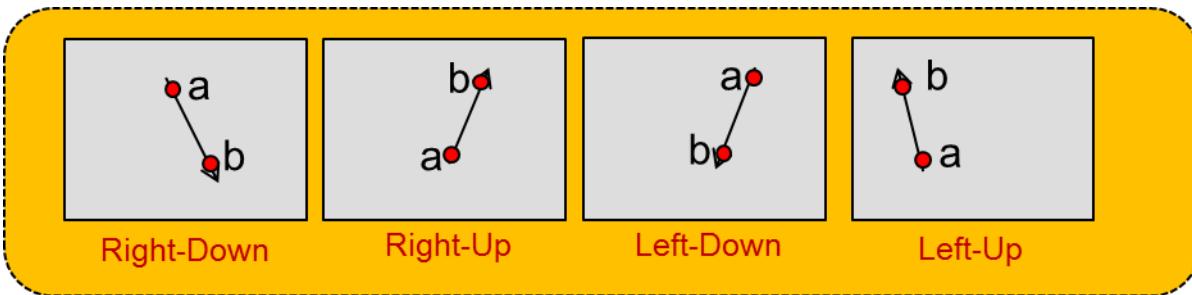


兩個點 $a(x_1, y_1)/b(x_2, y_2)$ 決定了一個象限，同時也決定了一個畫的方向。下圖為簡易決定的算法及示意圖。

```

if((x2 > x1) && (y2 > y1)) direction = b'001;
if((x2 > x1) && (y2 < y1)) direction = b'011;
if((x2 < x1) && (y2 > y1)) direction = b'101;
if((x2 < x1) && (y2 < y1)) direction = b'111;

```



為了避免繪製線路端點兩次，芯片提供了一個功能，抑制該行的最後一個像素的繪製，而這個功能在Bresenham畫線上被使用。Bresenham畫線操作可以是繪製操作，或者僅僅是一個移動操作。在Bresenham線繪製操作完成過程中，目標起始的 (x, y) 的點在操作畫至最後一點時，會被自動更新，在AU(2DGE-CTL[10])位被設置成1時。需要注意的是Bresenham畫線操作僅運作在X/ Y尋址模式時。

下面的例子說明了如何畫出一條實線：

- 找出兩點相對於八象限中的位置，設置基本的控制值，0x008B000加上八象限代碼。基本控制值表示了以下意義：設置CMD(GE2D_CTL[23:22])來執行Bresenham畫線加速功能，設置

BLNMD(GE2D_CTL[19])來執行Bresenham畫線功能，設置GE2DIEN(GE2D_CTL[17])啟動中斷功能，最後設置ADDRMD(GE2D_CTL[16])來決定定位方式為X/Y模式。

2. 在GE2D_BETSC寄存器中設置DIAGINC及AXIALINC位來指定給Diagonal/Axial 的Error Team值。DIAGINC在標準化後的初始值為(2*(Y的差值 – X的差值))。配置GE2D_BIEPC寄存器中的INIT及LINEPC位來決定Error Term的初始值及major axix的像素計數值。初始Error Term的初始值為(2*(Y的差值 – X的差值))。
3. 設置GE2D_FGCOLR寄存器決定前景值。
4. 設置GE2D_XYDORG寄存器決定畫線的起始位置。
5. 設置GE2D_SDPITCH寄存器決定目的間隔值。
6. 設置GE2D_DSTSPA寄存器決定畫線的目的起始位置。
7. 設置GO(GE2D_TRG[0])，啟動2D加速控制器。
8. 檢查GE2DIF(GE2D_INTSTS[0])位，以確定控制器已完成工作。
9. 透過寫1至GE2DIF位，清除中斷旗標。

下面為一簡單畫線的編程樣本：

```
/* octant code of line drawing */
/* #define XpYpX1      (0<<1)   */
/* #define XpYpY1      (1<<1)   */
/* #define XpYmX1      (2<<1)   */
/* #define XpYmY1      (3<<1)   */
/* #define XmYpX1      (4<<1)   */
/* #define XmYpY1      (5<<1)   */
/* #define XmYmX1      (6<<1)   */
/* #define XmYmY1      (7<<1)   */

// 決定象限值
abs_X = ABS(x2-x1); //absolute value
abs_Y = ABS(y2-y1); //absolute value
if (abs_X > abs_Y) { // X major
max = abs_X;
min = abs_Y;

step_constant = (((2*(min-max)) << 16) | (2*min));
initial_error = (((2*(min)-max)) << 16) | (max);

if (x2 > x1) { // +X direction
    if (y2 > y1) // +Y direction
        direction_code = XpYpX1;
```

```
        else                                // -Y direction
            direction_code = XpYmXl;
    } else {                                // -X direction
        if (y2 > y1)                      // +Y direction
            direction_code = XmYpXl;
        else                                // -Y direction
            direction_code = XmYmXl;
    }
} else {                                // Y major
    max = abs_Y;
    min = abs_X;

    step_constant = (((2*(min-max))) << 16) | (2*min);
    initial_error = (((2*(min)-max)) << 16) | (max);

    if (x2 > x1) {                      // +X direction
        if (y2 > y1)                      // +Y direction
            direction_code = XpYpYl;
        else                                // -Y direction
            direction_code = XpYmYl;
    } else {                                // -X direction
        if (y2 > y1)                      // +Y direction
            direction_code = XmYpYl;
        else                                // -Y direction
            direction_code = XmYmYl;
    }
}

GE2D_BETSC = step_constant; // set error term stepping constant
GE2D_BIEPC = initial_error; // set initial error, pixel count major -1

cmd32 = 0x008b0000 | direction_code;

GE2D_BGCOLR = make_color(color);      // 設定前景值
GE2D_SRCSPA = GFX_START_ADDR;        // 設定源起始值

dest_pitch = GFX_WIDTH << 16;         // 設定像素間隔
2DGE_SDPITCH = dest_pitch;

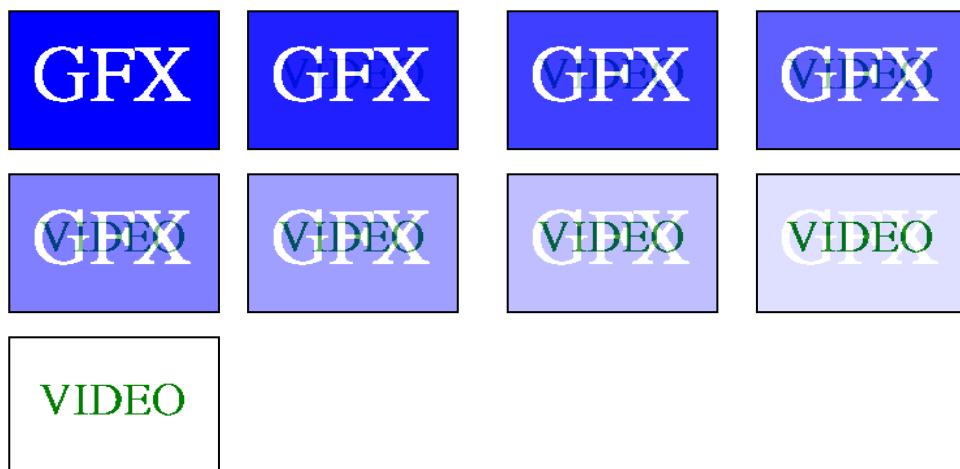
dest_start = y1 << 16 | x1;
2DGE_DSTSPA = dest_start;             //設定目的起始值
```

```
GE2D_CTL = cmd32;
GE2D_TRG = 1;           // 啟動2D加速控制器
while ((GE2D_INTSTS & 0x01)==0);    // 等待動作結束
GE2D_INTSTS = 1;        // 清除中斷旗標
```

12.5.5 α 混合

α 混合是半透明的前景顏色組合與背景顏色，由此產生了新的混合色的過程。前景色的透光度的範圍可以從完全透明到完全不透明。如果前景顏色是完全透明的，混合的顏色將是背景顏色。反之，如果它是完全不透明的，混合的顏色將是前景色。當然，半透明的範圍可在這兩個極端，在這種情況下，混合的顏色被計算為前景和背景顏色的加權平均值之間。

下圖為一 α 混合的實例：



決定 α 混合的動作包括以下步驟：

1. 設置ABLDFT(GE2D_MISCTL[31:16])位，決定前背景顯示因子。
2. 設置APABLDEN(GE2D_CTL[21])位，啟動 α 混合功能。

下面為一簡要的編程樣本：

```
data32 = GE2D_MISCTL & 0x0000ffff;
alpha = (_AlphaKs << 8) | _AlphaKd);
data32 |= (alpha << 16);
GE2D_MISCTL = data32;

cmd32 |= 0x00200000;      // 啟動 alpha 混合
GE2D_CTL = cmd32;
```

12.5.6 剪裁

剪裁功能提供了在繪圖過程中對於指定的矩形裡面或外面在顯示緩存裡進行裁剪的動作。本功能也提供了對BitBLTs進行矩形裁剪或Bresenham畫線時對線進行裁剪。當啟動此功能時，剪裁功能儘盡對矩形裡面或外面進行遮蔽的動作。

需要注意的是，此功能只能運作在X/Y繪圖模式，矩形的範圍由兩個點決定，分別定義為左上點(x_1, y_1)，及右下點(x_2, y_2)。

下圖為一對一矩形裡面和外面進行裁剪的實例：



下面的步驟說明了如何進行裁剪：

1. 設置CLPEN(GE2D_CTL[9])位啟動剪裁的功能，並設置CLPCTL(GE2D_CTL[8])控制是要裁剪矩形裡面或外面。
2. 透過設置GE2D_CLPBTL及GE2D_CLPBRR兩個寄存器來決定剪裁矩形左上及右下點的位置。

以下為一簡單操作的編程樣本：

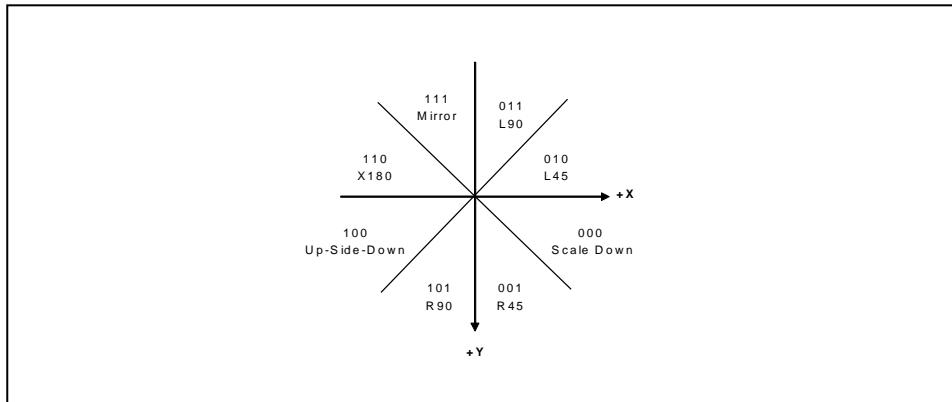
```
cmd32 |= 0x00000200; // enable clipping
if (_OutsideClip)
    cmd32 |= 0x00000100; // clip outside
    GE2D_CTL = cmd32;

    _ClipTL = ((y1 << 16) | x1);
    _ClipBR = ((y2 << 16) | x2);
    GE2D_CLPBTL = _ClipTL;
    GE2D_CLPBRR = _ClipBR;
```

12.5.7 旋轉

在本2D加速控制器裡，其中一個主要的功能就是在顯示緩存中做旋轉圖像加速，它可以向左或向右旋轉45/90/180度，所以它也可以支援左右鏡像或上下顛倒圖像。

旋轉的方向及角度是依據所在的8象限來決定，一個3位元的8象限碼如下圖所示：



以下提供了一個範例，2D控制器在顯示緩存中取得了一個矩形範圍，然後控制器也在此顯示緩存的矩形範圍中使用了指定的角度方向來繪製，旋轉功能最主要的運作過程就是繪製目的的方向，此方向是透過設置DRAWDIR(GE2D_CTL[3:1])位來決定的。

```

GE2D_SRCSPA = GFX_START_ADDR; // 顯示源地址
GE2D_DSTSPA = GFX_START_ADDR; // 顯示目的地址

pitch = GFX_WIDTH << 16 | GFX_WIDTH; // 設置源/目的間隔
GE2D_SDPITCH = pitch;

src_start = srcy << 16 | srcx;           // 設置X/Y源起始位置
GE2D_SRCSPA = src_start;

dest_start = desty << 16 | destx;         // 設置X/Y目的起始位置
GE2D_DSTSPA = dest_start;

dimension = height << 16 | width;        // 設置寬和高
GE2D_RTGLSZ = dimension;

u32cmd = 0xCC030000 | (direction << 1); // 填入8象限碼

GE2D_CTL = cmd32;
GE2D_TRG = 1;                         // 啟動2D加速控制器
while ((GE2D_INTSTS & 0x01)==0);      // 等待動作結束

GE2D_INTSTS = 1;                      // 清除中斷旗標

```

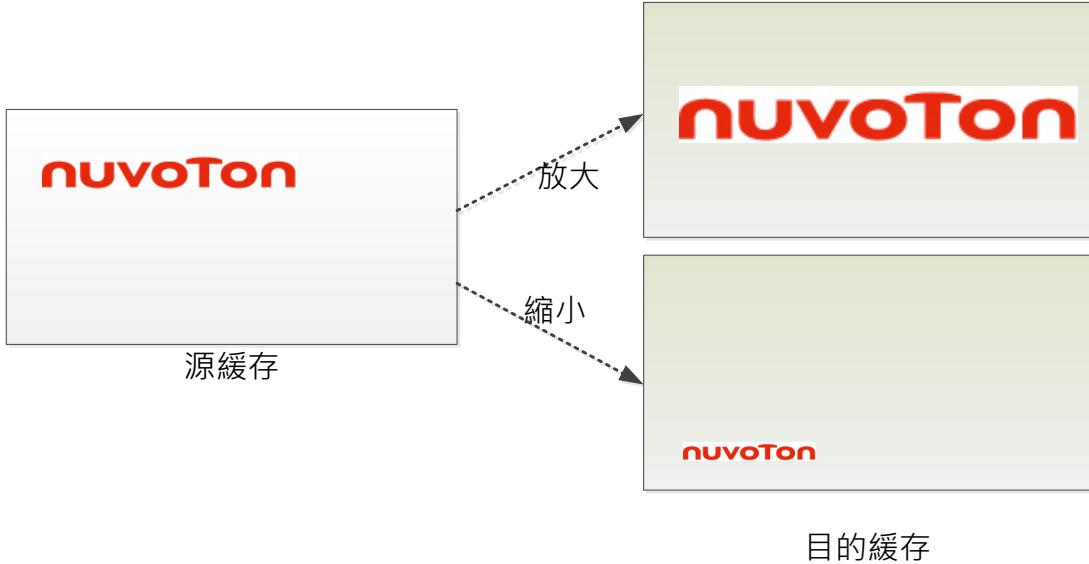
12.5.8 放大與縮小

2D控制器另一項主要的功能就是放大與縮小。這個功能主要就是能支援任何在顯示緩存內矩形進行放大與縮小功能，透過設置水平與重直放大縮小因子(N/M)來進行圖像的放大縮小。

為了要放大($1+N/M$)或縮小(N/M)，N因子數值必需等於或小於M。支援的放大倍率從1.0至1.996

倍。

下面列出一放大與縮小運作示意圖。



下面列出一個在顯示緩存中放大縮小的編程樣本，主要為透過設置GE2D_TCNTVHSF寄存器中的VSF_N，VSF_M，HSF_N，HSF_M位及GE2D_CTL寄存器中的SCLMD位來實現。

```
GE2D_SRCSPA = GFX_START_ADDR; //顯示源地址
GE2D_DSTSPA = GFX_START_ADDR; //顯示目的地址

pitch = GFX_WIDTH << 16 | GFX_WIDTH; // 設置源/目的間隔
GE2D_SDPITCH = pitch;

src_start = srcy << 16 | srcx;           // 設置X/Y源起始位置
GE2D_SRCSPA = src_start;

dest_start = desty << 16 | destx;        // 設置X/Y目的起始位置
GE2D_DSTSPA = dest_start;

dimension = height << 16 | width;         // 設置寬和高
GE2D_RTGLSZ = dimension;

stretch_ctl = (vsfN << 24) | (vsfM << 16) | (hsfN << 8) | hsfM;
GE2D_TCNTVHSF = stretch_ctl;

u32cmd = 0xCC030000;
GE2D_CTL = cmd32;
```

```
GE2D_TRG = 1;           // 啟動2D加速控制器
while ((GE2D_INTSTS & 0x01)==0); // 等待動作結束

GE2D_INTSTS = 1;        // 清除中斷旗標
```

13 通用 I/O 控制器 (GPIO)

13.1 概述

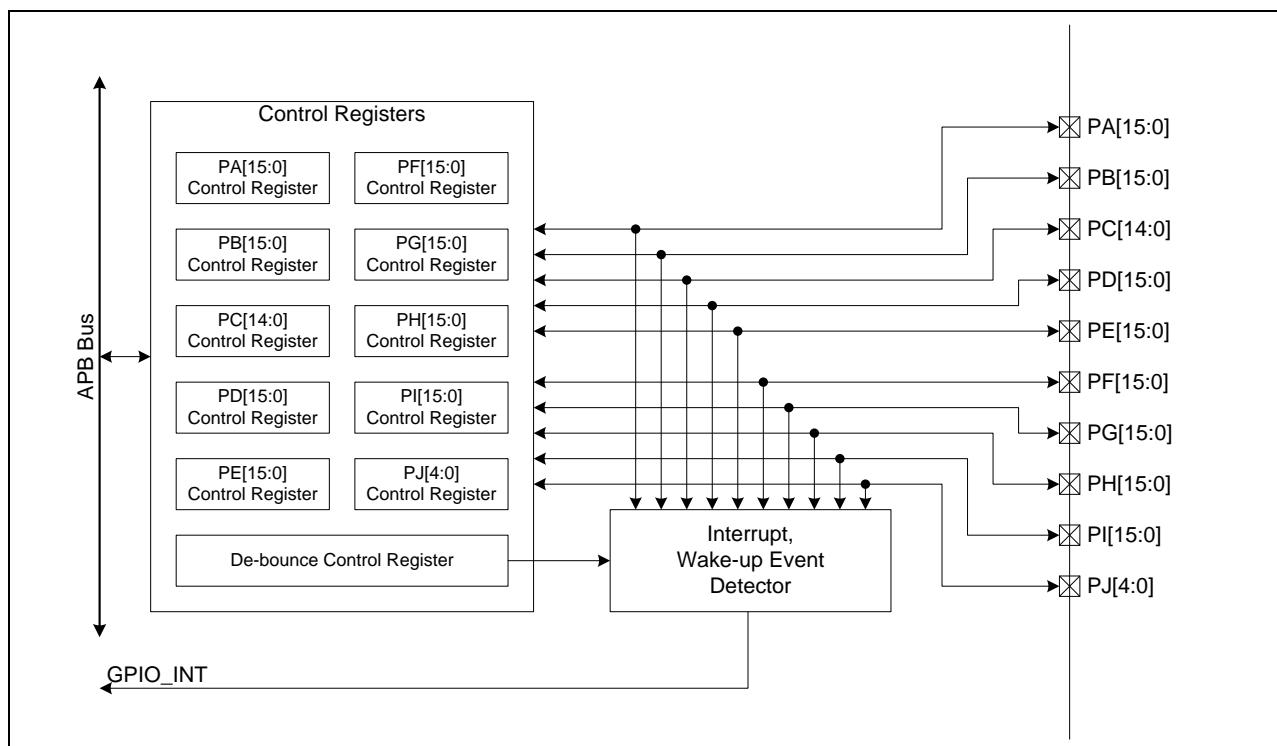
NUC970系列擁有多達148個通用 I/O 引腳，這些管腳可以和其他功能管腳共享。這些148個通用 I/O 引腳分布在PA, PB, PC, PD, PE, PF, PG, PH, PI 和 PJ。PA, PB, PD, PE, PF, PG, PH 和PI 各擁有16個通用 I/O 引腳，PC擁有15個通用 I/O 引腳和PJ擁有5個通用 I/O 引腳。每個I/O 引腳是獨立的，可以很容易地由用戶配置的，以滿足不同的系統配置和設計要求。復位後，所有的I/O 引腳配置為通用I/O輸入模式。

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($45\text{ k}\Omega \sim 82\text{ k}\Omega$)和下拉電阻($37\text{ k}\Omega \sim 91\text{ k}\Omega$)。上拉使能/下拉電阻是可控的。

13.2 特性

- 支持輸入和輸出模式。
- 支持 CMOS 和 Schmitt Trigger 輸入緩衝器
- 支持可控的上拉和下拉電阻
- 每个 I/O 引腳都可以作為中斷源，支持邊緣和電平觸發
- 支持去除抖動電路，濾除噪聲。

13.3 方塊圖



13.4 寄存器

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

Register	Address	R/W	Description	Reset Value
GPIO_BA = 0xB800_3000				
GPIOA_DIR	GPIO_BA+0x000	R/W	GPIO Port A Direction Control Register	0x0000_0000
GPIOA_DATAOUT	GPIO_BA+0x004	R/W	GPIO Port A Data Output Register	0x0000_0000
GPIOA_DATAIN	GPIO_BA+0x008	R	GPIO Port A Data Input Register	0xxxxx_xxxx
GPIOA_IMD	GPIO_BA+0x00C	R/W	GPIO Port A Interrupt Mode Register	0x0000_0000
GPIOA_IREN	GPIO_BA+0x010	R/W	GPIO Port A Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOA_IFEN	GPIO_BA+0x014	R/W	GPIO Port A Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOA_ISR	GPIO_BA+0x018	R/W	GPIO Port A Interrupt Status Register	0xxxxx_xxxx
GPIOA_DBEN	GPIO_BA+0x01C	R/W	GPIO Port A De-bounce Enable Register	0x0000_0000
GPIOA_PUEN	GPIO_BA+0x020	R/W	GPIO Port A Pull-Up Enable Register	0x0000_0000
GPIOA_PDEN	GPIO_BA+0x024	R/W	GPIO Port A Pull-Down Enable Register	0x0000_0000
GPIOA_ICEN	GPIO_BA+0x028	R/W	GPIO Port A CMOS Input Enable Register	0x0000_0000
GPIOA_ISEN	GPIO_BA+0x02C	R/W	GPIO Port A Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOB_DIR	GPIO_BA+0x040	R/W	GPIO Port B Direction Control Register	0x0000_0000
GPIOB_DATAOUT	GPIO_BA+0x044	R/W	GPIO Port B Data Output Register	0x0000_0000
GPIOB_DATAIN	GPIO_BA+0x048	R	GPIO Port B Data Input Register	0xxxxx_xxxx
GPIOB_IMD	GPIO_BA+0x04C	R/W	GPIO Port B Interrupt Mode Register	0x0000_0000
GPIOB_IREN	GPIO_BA+0x050	R/W	GPIO Port B Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOB_IFEN	GPIO_BA+0x054	R/W	GPIO Port B Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOB_ISR	GPIO_BA+0x058	R/W	GPIO Port B Interrupt Status Register	0xxxxx_xxxx
GPIOB_DBEN	GPIO_BA+0x05C	R/W	GPIO Port B De-bounce Enable Register	0x0000_0000
GPIOB_PUEN	GPIO_BA+0x060	R/W	GPIO Port B Pull-Up Enable Register	0x0000_0000
GPIOB_PDEN	GPIO_BA+0x064	R/W	GPIO Port B Pull-Down Enable Register	0x0000_0000
GPIOB_ICEN	GPIO_BA+0x068	R/W	GPIO Port B CMOS Input Enable Register	0x0000_0000

GPIOB_ISEN	GPIO_BA+0x06C	R/W	GPIO Port B Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOC_DIR	GPIO_BA+0x080	R/W	GPIO Port C Direction Control Register	0x0000_0000
GPIOC_DATAOUT	GPIO_BA+0x084	R/W	GPIO Port C Data Output Register	0x0000_0000
GPIOC_DATAIN	GPIO_BA+0x088	R	GPIO Port C Data Input Register	0xxxxx_xxxx
GPIOC_IMD	GPIO_BA+0x08C	R/W	GPIO Port C Interrupt Mode Register	0x0000_0000
GPIOC_IREN	GPIO_BA+0x090	R/W	GPIO Port C Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOC_IFEN	GPIO_BA+0x094	R/W	GPIO Port C Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOC_ISR	GPIO_BA+0x098	R/W	GPIO Port C Interrupt Status Register	0xxxxx_xxxx
GPIOC_DBEN	GPIO_BA+0x09C	R/W	GPIO Port C De-bounce Enable Register	0x0000_0000
GPIOC_PUEN	GPIO_BA+0x0A0	R/W	GPIO Port C Pull-Up Enable Register	0x0000_0000
GPIOC_PDEN	GPIO_BA+0x0A4	R/W	GPIO Port C Pull-Down Enable Register	0x0000_0000
GPIOC_ICEN	GPIO_BA+0x0A8	R/W	GPIO Port C CMOS Input Enable Register	0x0000_0000
GPIOC_ISEN	GPIO_BA+0x0AC	R/W	GPIO Port C Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOD_DIR	GPIO_BA+0x0C0	R/W	GPIO Port D Direction Control Register	0x0000_0000
GPIOD_DATAOUT	GPIO_BA+0x0C4	R/W	GPIO Port D Data Output Register	0x0000_0000
GPIOD_DATAIN	GPIO_BA+0x0C8	R	GPIO Port D Data Input Register	0xxxxx_xxxx
GPIOD_IMD	GPIO_BA+0x0CC	R/W	GPIO Port D Interrupt Mode Register	0x0000_0000
GPIOD_IREN	GPIO_BA+0x0D0	R/W	GPIO Port D Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOD_IFEN	GPIO_BA+0x0D4	R/W	GPIO Port D Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOD_ISR	GPIO_BA+0x0D8	R/W	GPIO Port D Interrupt Status Register	0xxxxx_xxxx
GPIOD_DBEN	GPIO_BA+0x0DC	R/W	GPIO Port D De-bounce Enable Register	0x0000_0000
GPIOD_PUEN	GPIO_BA+0x0E0	R/W	GPIO Port D Pull-Up Enable Register	0x0000_0000
GPIOD_PDEN	GPIO_BA+0x0E4	R/W	GPIO Port D Pull-Down Enable Register	0x0000_0000
GPIOD_ICEN	GPIO_BA+0x0E8	R/W	GPIO Port D CMOS Input Enable Register	0x0000_0000
GPIOD_ISEN	GPIO_BA+0x0EC	R/W	GPIO Port D Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOE_DIR	GPIO_BA+0x100	R/W	GPIO Port E Direction Control Register	0x0000_0000

GPIOE_DATAOUT	GPIO_BA+0x104	R/W	GPIO Port E Data Output Register	0x0000_0000
GPIOE_DATAIN	GPIO_BA+0x108	R	GPIO Port E Data Input Register	0xxxxx_xxxx
GPIOE_IMD	GPIO_BA+0x10C	R/W	GPIO Port E Interrupt Mode Register	0x0000_0000
GPIOE_IREN	GPIO_BA+0x110	R/W	GPIO Port E Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOE_IFEN	GPIO_BA+0x114	R/W	GPIO Port E Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOE_ISR	GPIO_BA+0x118	R/W	GPIO Port E Interrupt Status Register	0xxxxx_xxxx
GPIOE_DBEN	GPIO_BA+0x11C	R/W	GPIO Port E De-bounce Enable Register	0x0000_0000
GPIOE_PUEN	GPIO_BA+0x120	R/W	GPIO Port E Pull-Up Enable Register	0x0000_0000
GPIOE_PDEN	GPIO_BA+0x124	R/W	GPIO Port E Pull-Down Enable Register	0x0000_0000
GPIOE_ICEN	GPIO_BA+0x128	R/W	GPIO Port E CMOS Input Enable Register	0x0000_0000
GPIOE_ISEN	GPIO_BA+0x12C	R/W	GPIO Port E Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOF_DIR	GPIO_BA+0x140	R/W	GPIO Port F Direction Control Register	0x0000_0000
GPIOF_DATAOUT	GPIO_BA+0x144	R/W	GPIO Port F Data Output Register	0x0000_0000
GPIOF_DATAIN	GPIO_BA+0x148	R	GPIO Port F Data Input Register	0xxxxx_xxxx
GPIOF_IMD	GPIO_BA+0x14C	R/W	GPIO Port F Interrupt Mode Register	0x0000_0000
GPIOF_IREN	GPIO_BA+0x150	R/W	GPIO Port F Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOF_IFEN	GPIO_BA+0x154	R/W	GPIO Port F Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOF_ISR	GPIO_BA+0x158	R/W	GPIO Port F Interrupt Status Register	0xxxxx_xxxx
GPIOF_DBEN	GPIO_BA+0x15C	R/W	GPIO Port F De-bounce Enable Register	0x0000_0000
GPIOF_PUEN	GPIO_BA+0x160	R/W	GPIO Port F Pull-Up Enable Register	0x0000_0000
GPIOF_PDEN	GPIO_BA+0x164	R/W	GPIO Port F Pull-Down Enable Register	0x0000_0000
GPIOF_ICEN	GPIO_BA+0x168	R/W	GPIO Port F CMOS Input Enable Register	0x0000_0000
GPIOF_ISEN	GPIO_BA+0x16C	R/W	GPIO Port F Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOG_DIR	GPIO_BA+0x180	R/W	GPIO Port G Direction Control Register	0x0000_0000
GPIOG_DATAOUT	GPIO_BA+0x184	R/W	GPIO Port G Data Output Register	0x0000_0000
GPIOG_DATAIN	GPIO_BA+0x188	R	GPIO Port G Data Input Register	0xxxxx_xxxx

GPIOG_IMD	GPIO_BA+0x18C	R/W	GPIO Port G Interrupt Mode Register	0x0000_0000
GPIOG_IREN	GPIO_BA+0x190	R/W	GPIO Port G Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOG_IFEN	GPIO_BA+0x194	R/W	GPIO Port G Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOG_ISR	GPIO_BA+0x198	R/W	GPIO Port G Interrupt Status Register	0xxxxx_xxxx
GPIOG_DBEN	GPIO_BA+0x19C	R/W	GPIO Port G De-bounce Enable Register	0x0000_0000
GPIOG_PUEN	GPIO_BA+0x1A0	R/W	GPIO Port G Pull-Up Enable Register	0x0000_0000
GPIOG_PDEN	GPIO_BA+0x1A4	R/W	GPIO Port G Pull-Down Enable Register	0x0000_0000
GPIOG_ICEN	GPIO_BA+0x1A8	R/W	GPIO Port G CMOS Input Enable Register	0x0000_0000
GPIOG_ISEN	GPIO_BA+0x1AC	R/W	GPIO Port G Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOH_DIR	GPIO_BA+0x1C0	R/W	GPIO Port H Direction Control Register	0x0000_0000
GPIOH_DATAOUT	GPIO_BA+0x1C4	R/W	GPIO Port H Data Output Register	0x0000_0000
GPIOH_DATAIN	GPIO_BA+0x1C8	R	GPIO Port H Data Input Register	0xxxxx_xxxx
GPIOH_IMD	GPIO_BA+0x1CC	R/W	GPIO Port H Interrupt Mode Register	0x0000_0000
GPIOH_IREN	GPIO_BA+0x1D0	R/W	GPIO Port H Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOH_IFEN	GPIO_BA+0x1D4	R/W	GPIO Port H Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOH_ISR	GPIO_BA+0x1D8	R/W	GPIO Port H Interrupt Status Register	0xxxxx_xxxx
GPIOH_DBEN	GPIO_BA+0x1DC	R/W	GPIO Port H De-bounce Enable Register	0x0000_0000
GPIOH_PUEN	GPIO_BA+0x1E0	R/W	GPIO Port H Pull-Up Enable Register	0x0000_0000
GPIOH_PDEN	GPIO_BA+0x1E4	R/W	GPIO Port H Pull-Down Enable Register	0x0000_0000
GPIOH_ICEN	GPIO_BA+0x1E8	R/W	GPIO Port H CMOS Input Enable Register	0x0000_0000
GPIOH_ISEN	GPIO_BA+0x1EC	R/W	GPIO Port H Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOI_DIR	GPIO_BA+0x200	R/W	GPIO Port I Direction Control Register	0x0000_0000
GPIOI_DATAOUT	GPIO_BA+0x204	R/W	GPIO Port I Data Output Register	0x0000_0000
GPIOI_DATAIN	GPIO_BA+0x208	R	GPIO Port I Data Input Register	0xxxxx_xxxx
GPIOI_IMD	GPIO_BA+0x20C	R/W	GPIO Port I Interrupt Mode Register	0x0000_0000
GPIOI_IREN	GPIO_BA+0x210	R/W	GPIO Port I Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000

GPIOI_IFEN	GPIO_BA+0x214	R/W	GPIO Port I Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOI_ISR	GPIO_BA+0x218	R/W	GPIO Port I Interrupt Status Register	0xxxxx_xxxx
GPIOI_DBEN	GPIO_BA+0x21C	R/W	GPIO Port I De-bounce Enable Register	0x0000_0000
GPIOI_PUEN	GPIO_BA+0x220	R/W	GPIO Port I Pull-Up Enable Register	0x0000_0000
GPIOI_PDEN	GPIO_BA+0x224	R/W	GPIO Port I Pull-Down Enable Register	0x0000_0000
GPIOI_ICEN	GPIO_BA+0x228	R/W	GPIO Port I CMOS Input Enable Register	0x0000_0000
GPIOI_ISEN	GPIO_BA+0x22C	R/W	GPIO Port I Schmitt-Trigger Input Enable Register	0x0000_0000
GPIOJ_DIR	GPIO_BA+0x240	R/W	GPIO Port J Direction Control Register	0x0000_0000
GPIOJ_DATAOUT	GPIO_BA+0x244	R/W	GPIO Port J Data Output Register	0x0000_0000
GPIOJ_DATAIN	GPIO_BA+0x248	R	GPIO Port J Data Input Register	0xxxxx_xxxx
GPIOJ_IMD	GPIO_BA+0x24C	R/W	GPIO Port J Interrupt Mode Register	0x0000_0000
GPIOJ_IREN	GPIO_BA+0x250	R/W	GPIO Port J Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
GPIOJ_IFEN	GPIO_BA+0x254	R/W	GPIO Port J Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
GPIOJ_ISR	GPIO_BA+0x258	R/W	GPIO Port J Interrupt Status Register	0xxxxx_xxxx
GPIOJ_DBEN	GPIO_BA+0x25C	R/W	GPIO Port J De-bounce Enable Register	0x0000_0000
GPIOJ_PUEN	GPIO_BA+0x260	R/W	GPIO Port J Pull-Up Enable Register	0x0000_0000
GPIOJ_PDEN	GPIO_BA+0x264	R/W	GPIO Port J Pull-Down Enable Register	0x0000_0000
GPIOJ_ICEN	GPIO_BA+0x268	R/W	GPIO Port J CMOS Input Enable Register	0x0000_0000
GPIOJ_ISEN	GPIO_BA+0x26C	R/W	GPIO Port J Schmitt-Trigger Input Enable Register	0x0000_0000
GPIO_DBNCECON	GPIO_BA+0x3F0	R/W	GPIO Debounce Control Register	0x0000_0020
GPIO_ISR	GPIO_BA+0x3FC	R	GPIO Port Interrupt Status Register	0x0000_0000

13.5 功能描述.

13.5.1 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, SYS_GPG_MFPH, SYS_GPH_MFPL, SYS_GPH_MFPH, SYS_GPI_MFPL, SYS_GPI_MFPH 和 SYS_GPJ_MFPL 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);
```

13.5.2 GPIO 輸出模式

在使用GPIO引腳當作輸出前，需要配置GPIO方向寄存器(GPIOx_DIR)。配置順序模描述如下：

1. 設定多功能引腳為GPIO 模式。
2. 設定 GPIOx_DIR對應的Px.n值為1，1為輸出模式。

示範一個設定GPIOC[0] 為輸出模式，然後輸出1和0，如下：

```
// Set GPIOC[0] as I/O pin by SYS_GPC_MFPL register
outpw(SYS_GPC_MFPL, inpw(SYS_GPC_MFPL) & (~0x0000000F));

// Set GPIOC[0] as output mode by GPIOC_DIR register
outpw(GPIOC_DIR, inpw(GPIOC_DIR) | 0x00000001);

// Set GPIOC[0] output 1 by GPIOC_DATAOUT
outpw(GPIOC_DATAOUT, inpw(GPIOC_DATAOUT) | 0x00000001);

// Set GPIOC[0] output 0 by GPIOC_DATAOUT
outpw(GPIOC_DATAOUT, inpw(GPIOC_DATAOUT) & (~0x00000001));
```

13.5.3 GPIO 輸入模式

在使用GPIO引腳當作輸入前，需要配置GPIO方向寄存器(GPIOx_DIR)。配置順序模描述如下：

1. 設定多功能引腳為GPIO 模式。
2. 設定 GPIOx_DIR對應的Px.n值為0，0為輸入模式。

示範一個設定GPIOC[0] 為輸入模式，然後取得輸入的值，如下：

```
int GPIO_CFG=0;
int value=0;
// Set GPIOC[0] as I/O pin by SYS_GPC_MFPL register
```

```
outpw(SYS_GPC_MFPL, inpw(SYS_GPC_MFPL) & (~0x0000000F));

// Set GPIOC[0] as output mode by GPIOC_DIR register
outpw(GPIOC_DIR, inpw(GPIOC_DIR) & (~0x00000001));

// Get GPIOC[0] input value by GPIOC_DATAIN register
value = inpw(GPIOC_DATAIN) & 0x00000001;
if(value)
    printf("GPIOC[0] input value is 1.");
else
    printf("GPIOC[0] input value is 0.");
```

13.5.4 GPIO 中斷

每個GPIO引腳都支持中斷機制，需要配置GPIO寄存器。配置順序模描述如下：

1. 設定多功能引腳為GPIO 模式。
2. 設定 GPIOx_DIR 對應的 Px.n 值為 0，0 為輸入模式。
3. 設定 GPIOx_IMD 為 0(Edge trigger interrupt) 或 1(Level trigger interrupt)。
4. 設定 GPIOx_IREN 為 0(Rising disable) 或 1(Rising enable)。
5. 設定 GPIOx_IFEN 為 0(falling disable) 或 1(falling enable)。

除了上述步驟，在ISR中，還需要清除相對應的中斷源寄存器GPIOx_ISR。

示範一個設定GPIOC[0] 為輸入模式，利用輪尋(Polling)的方式來確定上升觸發時中斷旗標發生變化，如下：

```
int GPIO_CFG=0;
int value=0;
// Set GPIOC[0] as I/O pin by SYS_GPC_MFPL register
outpw(SYS_GPC_MFPL, inpw(SYS_GPC_MFPL) & (~0x0000000F));

// Set GPIOC[0] as output mode by GPIOC_DIR register
outpw(GPIOC_DIR, inpw(GPIOC_DIR) & (~0x00000001));

// Set GPIOC[0] as rising-edge trigger interrupt by GPIOC_IMD
outpw(GPIOC_IMD, inpw(GPIOC_IMD) & (~0x00000001));

// Set GPIOC[0] as rising-edge enable by GPIOC_IREN
outpw(GPIOC_IREN, inpw(GPIOC_IREN) & 0x00000001);

// Set GPIOC[0] as falling-edge disable by GPIOC_IFEN
```

```
outpw(GPIOC_IFEN, inpw(GPIOC_IFEN) & (~0x00000001));  
  
// Waitting for GPIOC[0] rising-edge interrupt by GPIOC_ISR  
while(!(inpw(GPIOC_ISR)& 0x1));  
  
// Clear GPIOC[0] interrupt flag by GPIOC_ISR  
outpw(GPIOC_ISR, 0x1);
```

14 I²C

14.1 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設備間的資料交換。I²C的標準是一個多主機匯流排，包括衝突檢測和仲裁以防止兩個或多個主機試圖同時控制匯流排時發生的資料損壞。

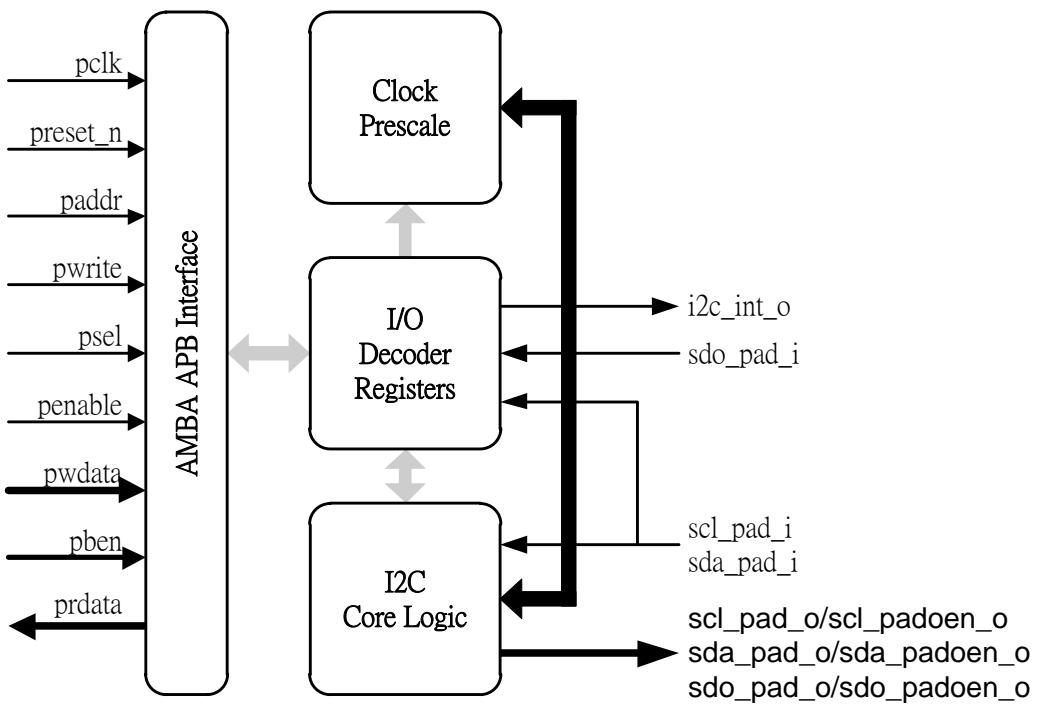
資料在主機與從機之間通過 SCL 時鐘線控制，在 SDA 資料線上按一位元組一位元組的同步傳輸。每個位元組為 8 位元長度，一個 SCL 時鐘脈衝傳輸一個資料位元，資料由最高位元 MSB 開始傳輸，每個傳輸位元組後跟隨一個應答位元。每個位在 SCL 為高時採樣，因此，SDA 線只有在 SCL 為低時才可以改變，在 SCL 為高時 SDA 保持穩定。當 SCL 為高時，SDA 線上的跳變視為命令中斷 (START or STOP)。

設備的片上 I²C邏輯提供符合 I²C匯流排規範的序列介面。I²C埠自動處理位元組傳輸，將 CSR 的 I2C_EN 設置為 '1'，即可使能該埠。I²C H/W 介面通過 SDA 與 SCL兩個引腳連到 I²C匯流排。用於I²C操作的兩個管腳需要上拉電阻，因為這個兩個管腳為開漏腳。

14.2 特性

- 支援 7 位元位址模式
- 支援主機模式操作
- 主從機之間雙向資料傳輸
- 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
- 串列同步時鐘可作為握手方式控制匯流排上資料暫停及恢復傳送
- 總線忙碌偵測
- 支援單一傳輸連續位元組傳送，最多可達個位4個元組
- 軟體控制模式

14.3 方塊圖



14.4 寄存器

Register	Offset	R/W/C	Description	Reset Value
I²C Port0 : I²C_BA = 0xB800_6000				
I²C Port1 : I²C_BA = 0xB800_6100				
CSR	I ² C_BA+0x00	R/W	Control and Status Register	0x0000_0000
DIVIDER	I ² C_BA+0x04	R/W	Clock Prescale Register	0x0000_0000
CMDR	I ² C_BA+0x08	R/W	Command Register	0x0000_0000
SWR	I ² C_BA+0x0C	R/W	Software Mode Control Register	0x0000_003F
RxR	I ² C_BA+0x10	R	Data Receive Register	0x0000_0000
TxR	I ² C_BA+0x14	R/W	Data Transmit Register	0x0000_0000

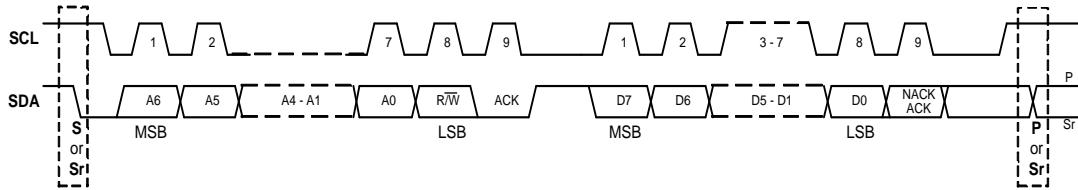
14.5 功能描述

14.5.1 I²C 協定

通常標準的 I²C 通信包含四個部分：

1. 起始信號 (START) 或者重複起始信號(Repeat-START)
2. 從機位址傳輸
3. 資料傳輸

4. 停止信號 (STOP)



14.5.2 連續傳送

此功能只能用在將數據送出時。

設置Tx_NUM(CSR[5:4])位，可決定在一次的I²C啟動中可連續傳輸的數據次數，可設置的範圍從1~4次。

如果Tx_NUM=3，I²C數據傳送順序依序為TxR[31:24]，TxR[23:16]，TxR[15:8]，TxR[7:0]。

```
CSR |= (0x3 << 4); //連續傳送4個8位元數據，依序為0x12, 0x34, 0x56, 0x78
u32Data = 0x12 << 24;
u32Data |= (0x34 << 16);
u32Data |= (0x56 << 8);
u32Data |= 0x78;
TxR = u32Data; // 寫入TxR寄存器
```

14.5.3 中斷

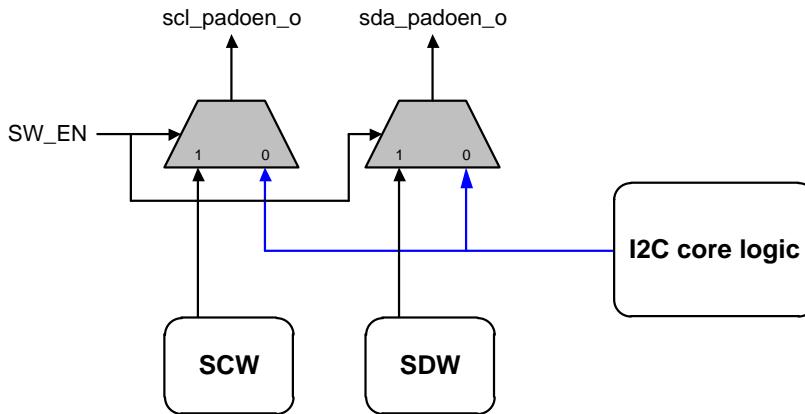
當I²C控制器完成一次單元傳輸，單元傳輸中斷標誌 IF (CSR[2])位將會被設置為1。如果單元傳輸中斷使能位元 IE (CSR[1]) 被設置成1時，單元傳輸中斷事件將會向CPU產生一個中斷。單元傳輸標誌僅可以通過向自身寫1清除。

```
CSR |= 0x2; //使能中斷
```

如果發生仲裁錯誤，I2C_AL標誌(CSR[9])和IF標誌都將會被設置為1，亦會發生中斷通知軟體。此時需送出STOP訊號，將控制權釋放，未完成的讀或寫的動作需再重新執行一次。

14.5.4 軟體控制模式

方塊圖如下：



如要使用I²C軟體控制模式，需將I²C_EN(CSR[0])位設置成0。軟體透過設置SCW(SWR[0])位，SDW(SWR[1])位，可將I²C SCL管腳及SDA管腳拉高或拉低。

軟體可透過讀取SCR(SWR[3])位或SDR(SWR[4])位得知目前管腳的狀態。

```
CSR &= ~0x1; //關閉硬體I2C功能
//送出START訊號
SWR |= 0x3; //SCL=high, SDA=high
sleep();
SWR &= ~0x2; //SCL=high, SDA=low
sleep();
SWR &= ~0x1; // SCL=low, SDA=low
sleep();
...
...
```

14.5.5 I²C 命令寄存器操作

CMDR為可控制I²C START/STOP，READ/WRITE，ACK/NACK動作的寄存器。

一般控制流程如下：

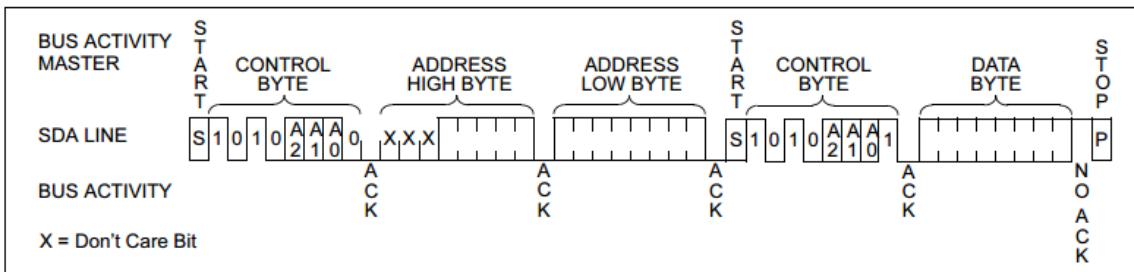
1. 寫入數據至Tx(TxR[7:0])
2. 設置START位(CMDR[4])，及設置寫(CMDR[1])位
3. 等待中斷
4. 檢查確定從裝置回應為ACK 或 NACK (CMDR[0])並設STOP位(CMDR[3])
5. 等待中斷
6. 停止

```
TxR = 0x12; //從裝置位址
CMDR |= (0x1 << 4) | (0x1 << 1); //啟動I2C'寫'程序
while(CSR & (0x1 << 8)); //輪詢I2C狀況
if(!CMDR & 0x1)
    printf("Transfer error!!\n");
```

```
CMMDR |= (0x1 << 3);           //設STOP位，停止傳輸
while(CSR & (0x1 << 8));      //輪詢I2C狀況
```

14.5.6 I²C EEPROM 操作使用示例(24LC64 為例)

Random Read:



以下為編程樣本：

```
CSR &= ~(0x3 << 4);           //Tx_NUM=0
CSR |= 0x1;                      //啟動硬體I2C功能

TxR = 0x50;                      //寫入24LC64位置/寫動作
CMDR = (0x1 << 4) | (0x1 << 1); //設置START及WRITE位
while(CSR & (0x1 << 8));      //輪詢I2C狀況
if(!CMDR & 0x1)
    printf("Transfer error!!\n");

TxR = 0x00;                      //高位位置
CMDR |= (0x1 << 1);            //設置WRITE位
while(CSR & (0x1 << 8));      //輪詢I2C狀況
if(!CMDR & 0x1)
    printf("Transfer error!!\n");

TxR = 0x01;                      //低位位置
CMDR |= (0x1 << 1);            //設置WRITE位
while(CSR & (0x1 << 8));      //輪詢I2C狀況
if(!CMDR & 0x1)
    printf("Transfer error!!\n");

TxR = 0x51;                      //控制位元/讀動作
CMDR |= (0x1 << 4) | (0x1 << 1); //設置START, WRITE位
while(CSR & (0x1 << 8));      //輪詢I2C狀況
if(!CMDR & 0x1)
    printf("Transfer error!!\n");
```

```
CMDR |= (0x1 << 3) | (0x1 << 2) | (0x1 << 2); //設置STOP,READ, ACK位  
u32data = RxR; //讀回數據
```

15 I²S

15.1 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。

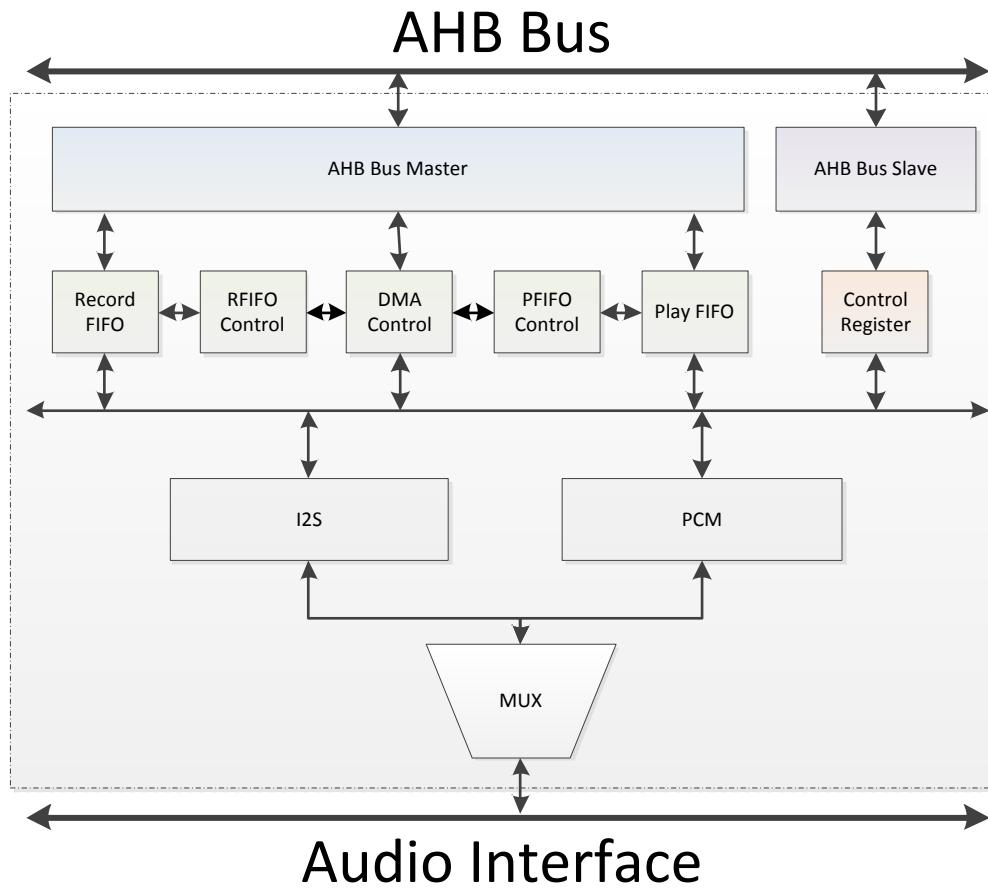
當在18/20/24位精度操作時，每個左/右聲道採樣值是存儲於一個32位的字節內。

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。

15.2 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

15.3 方塊圖



15.4 寄存器

Register	Address	R/W	Description	Reset Value
I2S_BA = 0xB000_9000				
ACTL_CON	I2S_BA + 0x00	R/W	Audio control register	0x0000_0000
ACTL_RESET	I2S_BA + 0x04	R/W	Sub block reset control register	0x0000_0000
ACTL_RDESB	I2S_BA + 0x08	R/W	DMA destination base address register for record	0x0000_0000
ACTL_RDES_LENGTH	I2S_BA + 0x0C	R/W	DMA destination length register for record	0x0000_0000
ACTL_RDESC	I2S_BA + 0x10	R	DMA destination current address register for record	0x0000_0000
ACTL_PDESB	I2S_BA + 0x14	R/W	DMA destination base address register for play	0x0000_0000
ACTL_PDES_LENGTH	I2S_BA + 0x18	R/W	DMA destination length register for play	0x0000_0000
ACTL_PDESC	I2S_BA + 0x1C	R	DMA destination current address register for play	0x0000_0000
ACTL_RSR	I2S_BA + 0x20	R/W	Record status register	0x0000_0000
ACTL_PSR	I2S_BA + 0x24	R/W	Play status register	0x0000_0000

ACTL_I2SCON	I2S_BA + 0x28	R/W	I2S control register	0x0000_0000
ACTL_COUNTER	I2S_BA+ 0x2C	R/W	DMA counter down values	0xFFFF_FFFF
ACTL_PCMCON	I2S_BA + 0x30	R/W	PCM interface control register	0x0000_0000
ACTL_PCMS1ST	I2S_BA + 0x34	R/W	PCM interface slot1 start register	0x0000_0000
ACTL_PCMS2ST	I2S_BA + 0x38	R/W	PCM interface slot2 start register	0x0000_0000
ACTL_RDESB2	I2S_BA + 0x40	R/W	DMA destination base address register for record right channel	0x0000_0000
ACTL_PDESB2	I2S_BA + 0x44	R/W	DMA destination base address register for play right channel	0x0000_0000

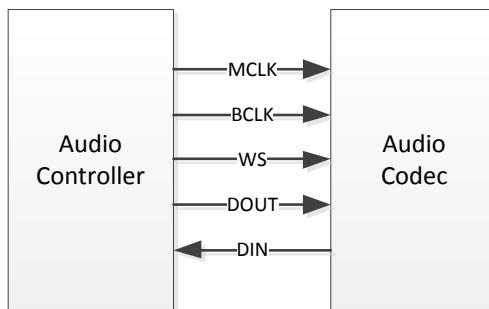
15.5 功能描述

15.5.1 I²S 主/從模式設定

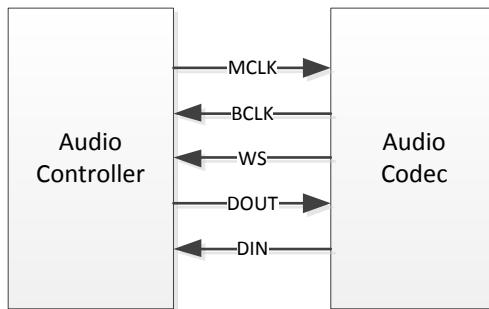
使用從模式，可設置SLAVE(CTL_I2SCON[20])為1，主模式需設為0。

在選擇使用I²S介面時(CTL_I2SCON[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：

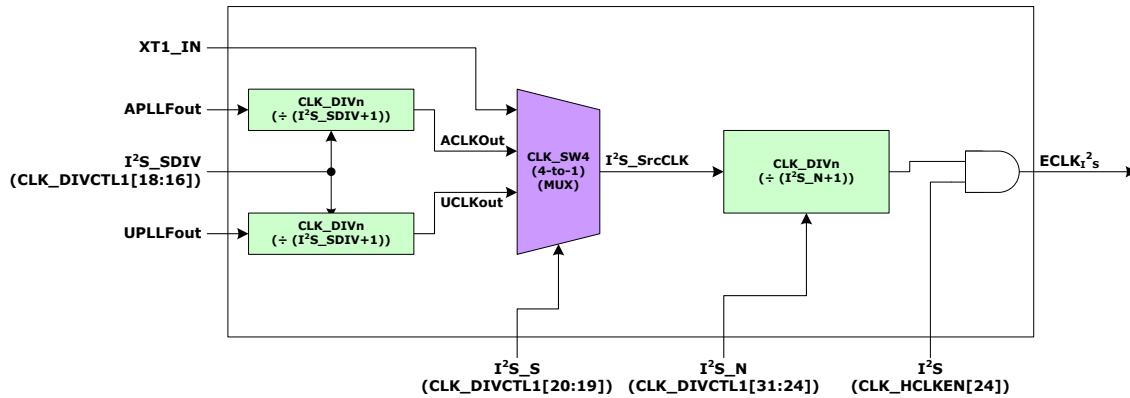


從模式接線示意圖如下：



15.5.2 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。



$\text{CLK_DIVCTL1} = (\text{CLK_DIVCTL1} \& \sim(0x3 << 19)) | 0x2;$ //選擇APLL為I²S時鐘源

15.5.3 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：

$$\text{MCLK} = 256 * 48000 = 11288000 \text{ Hz} = 12.288\text{MHz}$$

且以16位資料精度及雙聲道為例，BCLK的計算方式如下：

$$\text{BCLK} = 48000 * 16 * 2 = 1536000 \text{ Hz} = 1.536\text{MHz}$$

此時可設置分頻值PSR(ACTL_I2SCON[19:16])為 $12.288/12.288 - 1 = 1 - 1 = 0$

```
BCLK_DIV(ACTL_I2SCON[7:5])為  $(12.288/1.536)/2-1 = 8/2-1 = 3$ 
ACTL_I2SCON = ACTL_I2SCON &  $\sim(0xF << 16)$ ; //PRS=0
ACTL_I2SCON = ACTL_I2SCON &  $\sim(0x1 << 4)$ ; //PLL需經過PRS分頻
ACTL_I2SCON = (ACTL_I2SCON &  $\sim(0x1 << 5)$ ) | 0x3; //BCLK_DIV=3
```

15.5.4 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 - ACTL_RDESB和ACTL_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 - ACTL_RDES_LENGTH和ACTL_PDES_LENGTH，指定DMA空間的長度。

DMA目前位置指示寄存器 – ACTL_RDESC和ACTL_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(ectl_CON[15:14])，P_DMA_IRQ_SEL(ectl_CON[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(ectl_CON[21])或P_DMA_IRQ_EN(ectl_CON[20])位。

DMA設置及流程以回放為例如下。

```
ACTL_PDESB = 0x80001000;           //配置非緩存的內存位置
ACTL_PDES_LENGTH = 2*1024;         //長度為2k位元組
ACTL_I2SCON = (ACTL_I2SCON & ~(0x3 << 12)) | (0x1 << 12); //中斷發生為1/2 DMA長度時
ACTL_I2SCON |= (0x1 << 20);       //啟動中斷
```

軟體可透過讀取P_DMA_RIA_SN(ectl_PSR[7:5])或R_DMA_RIA_SN(ectl_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ACTL_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_CNTEN(ectl_CON[4])位發生中斷通知軟體。

```
ACTL_COUNTER = 0x1000;             //設置倒數初始值為0x1000
...
while(ectl_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(ectl_CON[3])位發生中斷通知軟體。

15.5.5 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
...	...

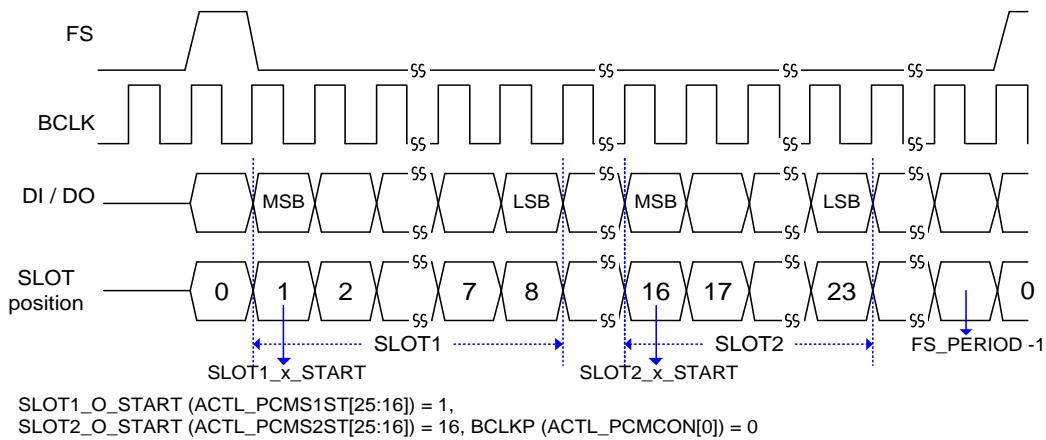
15.5.6 介面選擇及設置

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

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

15.5.7 PCM 介面的配置

一個PCM介面的波形如下：



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(ACTL_PCMCON[25:16])。
2. SLOT1_x_START 及 SLOT2_x_START 距離 FS 的波特數 – ACTL_PCMS1ST 及 ACTL_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
ACTL_PCMCON = ACTL_PCMCON | (23<<8);

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

//SLOT1_O_START = 1
//SLOT1_I_START = 1
ACTL_PCMS1ST = 0x00010001;

//SLOT2_O_START = 33
//SLOT2_I_START = 33
ACTL_PCMS2ST = 0x00210021;
```

15.5.8 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ACTL_RDESB2及ACTL_PDESB2寄存器指定數據分割之後第二個存放位址。原來的ACTL_RDESB及ACTL_PDESB固定放置I²S左聲道或PCM slot1的數據，ACTL_RDESB2

及ACTL_PDESB2則是放置I²S右聲道或PCM slot2的數據。

可設置SPLIT_DATA(ACTL_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



16 JPEG 編解碼器

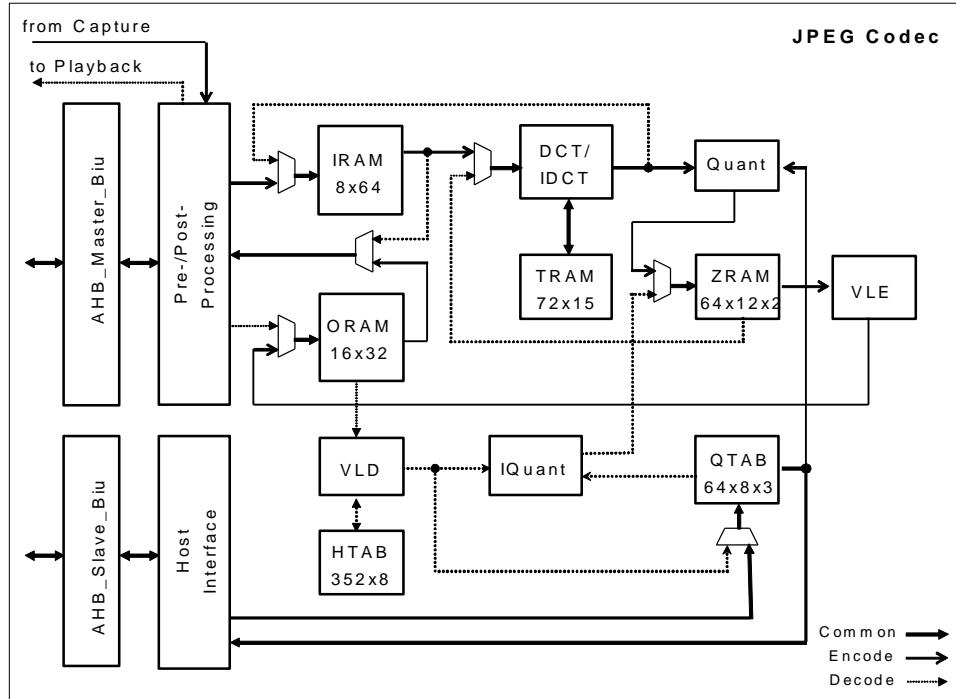
16.1 概述

JPEG 編解碼器支持基線順序模式(Baseline Sequential Mode) 的JPEG靜態影像壓縮和解壓縮,並且完全符合ISO/IEC國際標準10918-1(T.81).

16.2 特性

- 支持編碼交錯 YCbCr 4:2:2/4:2:0 以及灰階(Y only)格式影像
- 支持解碼交錯 YCbCr 4:4:4/4:2:2/4:2:0/4:1:1 以及灰階(Y only)格式影像
- 支持解碼 YCbCr 4:2:2 轉換格式
- 支持完全符合 JFIF 和 EXIF 標準的 JPEG 碼流格式編碼
- 支持 Capture 和 JPEG 硬體即時編碼模式
- 支持 JPEG 解碼和硬體即時播放模式
- 支持軟體即時輸入/輸出的編解碼模式
- 支持任意長寬影像的編碼和解碼
- 支持三種可程式化量化表
- 支持標準霍夫曼表解碼和可程式化霍夫曼表解碼
- 在編碼模式下支持任意(1~8 倍)的影像放大功能
- 在編碼和解碼模式下支持縮小功能
- 支持特定視窗解碼模式
- 在編碼模式下支持量化表調整比特率和質量控制
- 在編碼模式下支持旋轉功能

16.3 方塊圖



16.4 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/C	Description	Reset Value
JPG_BA = 0xB000_A000				
JMCR	JPG_BA + 000	R/W	JPEG Engine Mode Control Register	0x0000_0000
JHEADER	JPG_BA + 004	R/W	JPEG Encode Header Control Register	0x0000_0000
JITCR	JPG_BA + 008	R/W	JPEG Image Type Control Register	0x0000_0000
RESERVED	JPG_BA + 00C	R/W	Reserved	0x0000_0000
JPRIQC	JPG_BA + 010	R/W	JPEG Encode Primary Q-Table Control Register	0x0000_00F4
JTHBQC	JPG_BA + 014	R/W	JPEG Encode Thumbnail Q-Table Control Register	0x0000_00F4
JPRIWH	JPG_BA + 018	R/W	JPEG Primary Width/Height Register	0x0000_0000
JTHBWH	JPG_BA + 01C	R/W	JPEG Encode Thumbnail Width/Height Register(For Planar Format Only)	0x0000_0000
JPRST	JPG_BA + 020	R/W	JPEG Encode Primary Restart Interval Register	0x0000_0004
JTRST	JPG_BA + 024	R/W	JPEG Encode Thumbnail Restart Interval Register	0x0000_0004
JDECWH	JPG_BA + 028	R	JPEG Decode Image Width/Height Register	0x0000_0000
JINTCR	JPG_BA + 02C	R/W	JPEG Interrupt Control and Status Register	0x0020_0000
RESERVED	JPG_BA + 034~ JPG_BA + 038	R/W	Reserved	0x0000_0000

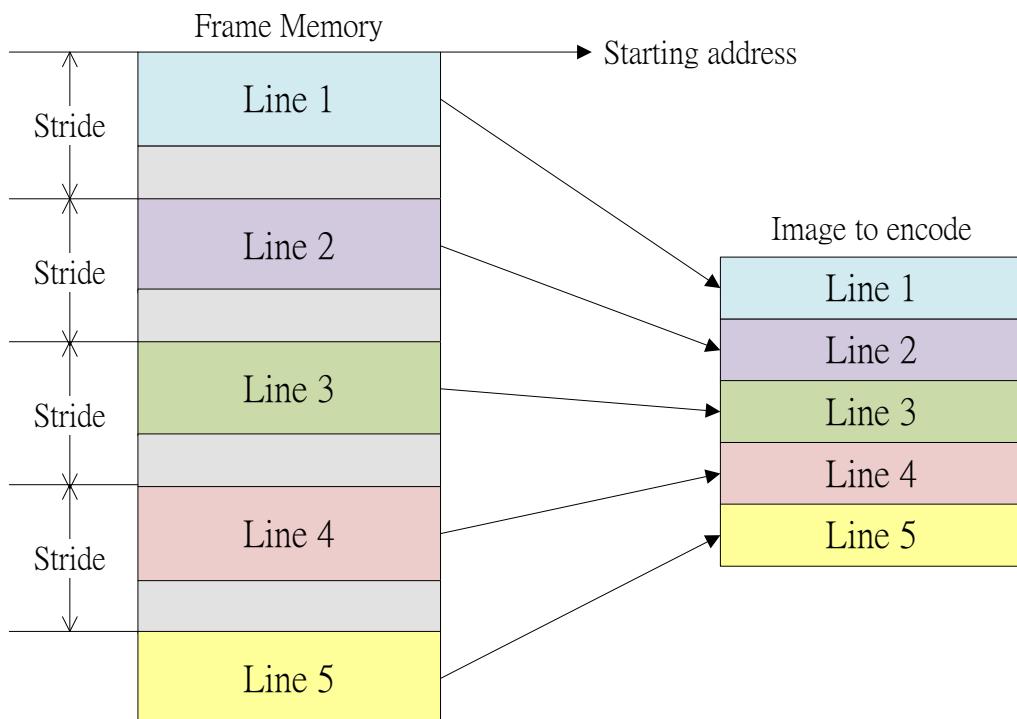
JDOWFBS	JPG_BA + 03C	R/W	Decoding Output Wait Frame Buffer Size	0xFFFF_FFFF
JTEST	JPG_BA + 040	R/W	JPEG Test Control Register	0x0000_0000
JWINDEC0	JPG_BA + 044	R/W	JPEG Window Decode Mode Control Register 0	0x0000_0000
JWINDEC1	JPG_BA + 048	R/W	JPEG Window Decode Mode Control Register 1	0x0000_0000
JWINDEC2	JPG_BA + 04C	R/W	JPEG Window Decode Mode Control Register 2	0x0000_0000
JMACR	JPG_BA + 050	R/W	JPEG Memory Address Mode Control Register	0x0000_0000
JPSCALU	JPG_BA + 054	R/W	JPEG Primary Scaling-Up Control Register	0x0000_0000
JPSCALD	JPG_BA + 058	R/W	JPEG Primary Scaling-Down Control Register	0x0000_0000
JTSCALD	JPG_BA + 05C	R/W	JPEG Thumbnail Scaling-Down Control Register	0x0000_0000
JDBCR	JPG_BA + 060	R/W	JPEG Dual-Buffer Control Register	0x0000_0000
RESERVED	JPG_BA + 064 ~ JPG_BA + 06C	R/W	Reserved	0x0000_0000
JRESERVE	JPG_BA + 070	R/W	Primary Encode Bit-stream Reserved Size Register	0x0000_0000
JOFFSET	JPG_BA + 074	R/W	Address Offset Between Primary/Thumbnail Register	0x0000_0000
JFStride	JPG_BA + 078	R/W	JPEG Encode Bit-stream Frame Stride Register	0x0000_0000
JYADDR0	JPG_BA + 07C	R/W	Y Component or Packet Format Frame Buffer-0 Start Address Register,	0x0000_0000
JUADDR0	JPG_BA + 080	R/W	U Component Frame Buffer-0 Start Address Register	0x0000_0000
JVADDR0	JPG_BA + 084	R/W	V Component Frame Buffer-0 Start Address Register	0x0000_0000
JYADDR1	JPG_BA + 088	R/W	Y Component or Packet Format Frame Buffer-1 Start Address Register	0x0000_0000
JUADDR1	JPG_BA + 08C	R/W	U Component Frame Buffer-1 Start Address Register	0x0000_0000
JVADDR1	JPG_BA + 090	R/W	V Component Frame Buffer-1 Start Address Register	0x0000_0000
JYStride	JPG_BA + 094	R/W	Y Component Frame Buffer Stride Register	0x0000_0000
JUSTRIDE	JPG_BA + 098	R/W	U Component Frame Buffer Stride Register	0x0000_0000
JVStride	JPG_BA + 09C	R/W	V Component Frame Buffer Stride Register	0x0000_0000
JIOADDR0	JPG_BA + 0A0	R/W	Bit-stream Frame Buffer-0 Start Address Register	0x0000_0000
JIOADDR1	JPG_BA + 0A4	R/W	Bit-stream Frame Buffer-1 Start Address Register	0x0000_0000
JPRI_SIZE	JPG_BA + 0A8	R	JPEG Encode Primary Bit-stream Size Register	0x0000_0000
JTHB_SIZE	JPG_BA + 0AC	R	JPEG Encode Thumbnail Bit-stream Size Register	0x0000_0000
JUPRAT	JPG_BA + 0B0	R/W	JPEG Planar Format Encode Up-Scale Ratio and Packet Format Decode Down-Scale Ratio	0x0000_0000
JBSIFO	JPG_BA + 0B4	R/W	JPEG Bit-stream FIFO Control Register	0x0000_0032
JSRCH	JPG_BA + 0B8	R/W	JPEG Encode Source Image Height	0x0000_0FFF
RESERVED	JPG_BA + 0BC ~ JPG_BA + 0FC	R/W	Reserved	0x0000_0000
JQTAB0	JPG_BA + 100 ~ JPG_BA + 13F	R/W	JPEG Quantization-Table 0	0x0000_0000

JQTAB1	JPG_BA + 140 ~ JPG_BA + 17F	R/W	JPEG Quantization-Table 1	0x0000_0000
JQTAB2	JPG_BA + 180 ~ JPG_BA + 1BF	R/W	JPEG Quantization-Table 2	0x0000_0000
RESERVED	JPG_BA + 1C8 ~ JPG_BA + 1FC	R/W	Reserved	0x0000_0000

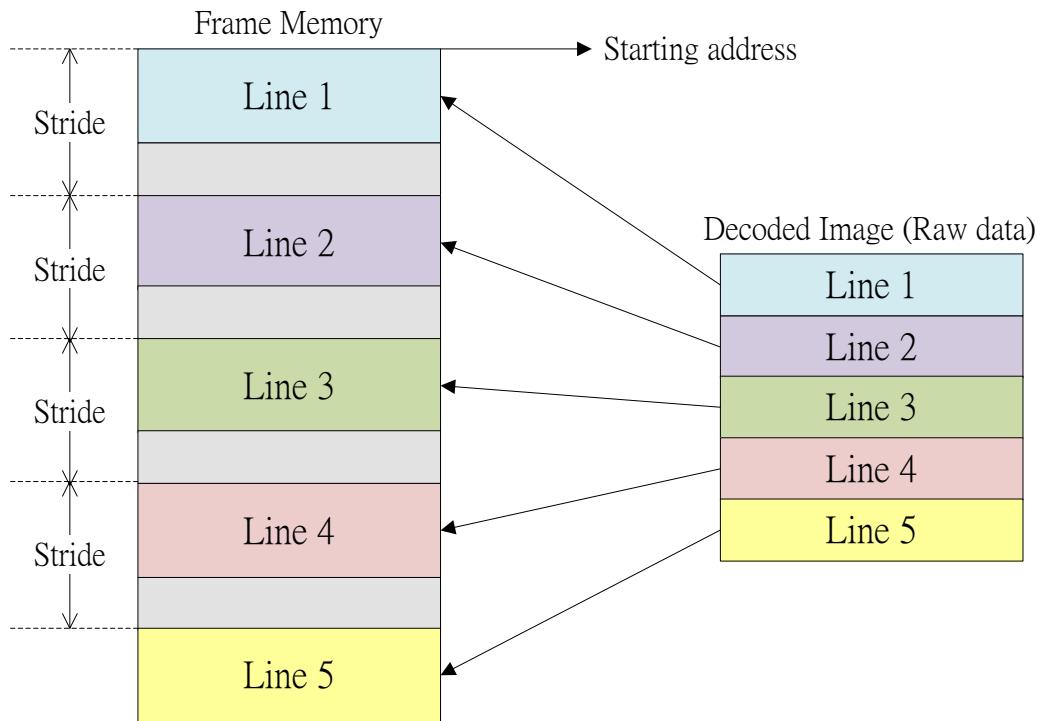
16.5 功能說明

16.5.1 訪問記憶體(Memory Access)

下圖表示在編碼模式下,資料從感測器進入並儲存到SDRAM上.

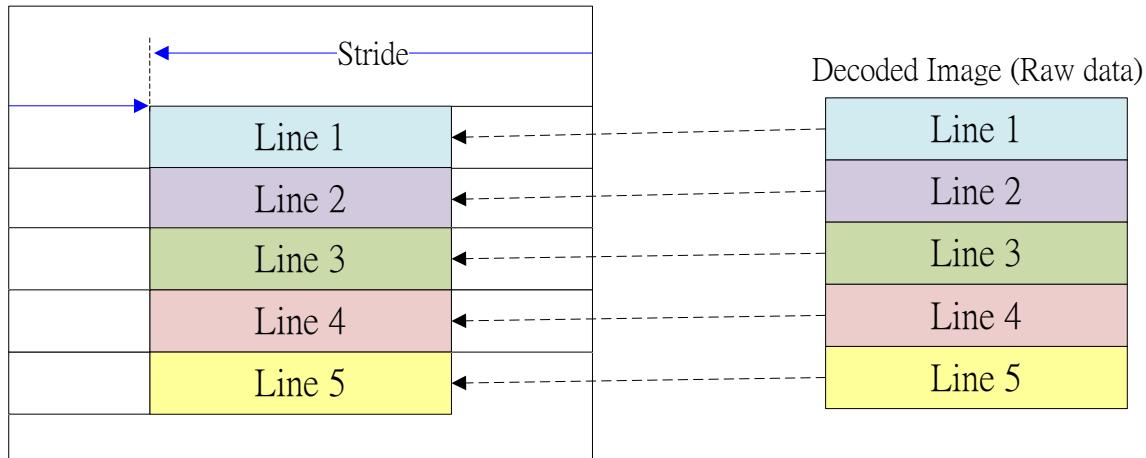


下圖表示,在解碼模式下,將SDRAM內的資料經由解碼輸出.



用戶可使用步幅(stride)功能將解碼影像輸出到顯示用的顯示幀緩衝上的任意位址.下圖表示在解碼模式使用步幅功能輸出解碼原始資料到顯示緩衝幀.

Display Frame Buffer Memory



16.5.2 JPEG 編碼

16.5.2.1 Jpeg復位

JPEG支持編碼/解碼碼流(bit-stream). 在觸發Jpeg的解碼或編碼碼流之前,必須要先將Jpeg復位(先設定JMCR[1] = 1,再設定JMCR[1] = 0).否則,編碼或解碼的結果可能會是錯誤的.

16.5.2.2 量化表

● 量化表順序 Quantization Table Order

量化表在寄存器內的順序和碼流上的順序是不同的.

例如,以下面的量化表來說:

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

這個JPEG量化表,寫入寄存器的順序是0, 1, 5, 6, 14, 15, 27, 28 63.

但是,在碼流上的zig-zag順序卻是0, 1, 2, 3, 4, 5.....63.

● 寫入量化表

在寫入JPEG量化表之前,必須先確認量化表的忙碌狀態是為'0'(JMCR[2] =0).如果JMCR[2] =1時,寫入JPEG量化表(JPB_BA+0x100 ~ 0x17F)將會失敗.

16.5.3 一般編碼

編碼設定的流程如下:

1. 將 JPEG 復位

2. 寫入量化表
3. 設定輸入幀緩衝的起始位址
 - 平面格式(Planar format) : Y/U/V 起始位址 (JYADDR0 / JUADDR0 / JVADDR0)
 - 緊縮格式(Packet format) : 資料起始位址 (JYADDR0)
4. 設定輸出碼流的目標位址. (JIOADDR0)
5. 設定編碼影像的尺寸屬性
 - 設定主要編碼影像的寬和高 (JPRIWH)
 - 設定編碼來源影像的高度 (JSRCH)
6. 設定影像初始資料屬性
 - 平面格式(Planar format)
 - ◆ Y/U/V 緩衝區步幅 (JYSTRIDE / JUSTRIDE / JVSTRIDE)
 - 緊縮格式(Packet format)
 - ◆ 緩衝區步幅 (JYSTRIDE 是影像寬度, JUSTRIDE/JVSTRIDE 是影像寬度除 2)
7. 設定編碼模式和格式
 - 設定編碼模式. (JMCR[7] =1)
 - 主影像編碼 (JMCR[5])
 - 縮略影像編碼 (JMCR[4])
 - 編碼格式 (JMCR[3])
 - ◆ YUV422
 - ◆ YUV420
8. 設定編碼標頭 (JHEADER)
9. 使能編碼中斷 (JINTCR)
 - 使能編碼完成中斷 (ENC_INTE)
10. 觸發開始編碼 (JINTCR)
 - 設定 JMCR[0] = 1, 然後再設定 JMCR[0] = 0.
11. 等待編碼完成中斷

16.5.4 影像放大編碼

在下面的公式裡,Width和Height指的是原始圖片的寬和高, Upscale_Width和Upscale_Height 放大後圖片的寬和高.

$$\text{X ratio} = ((\text{Upscale_Width} - 1) / (\text{Width} - 1)) * 1024$$

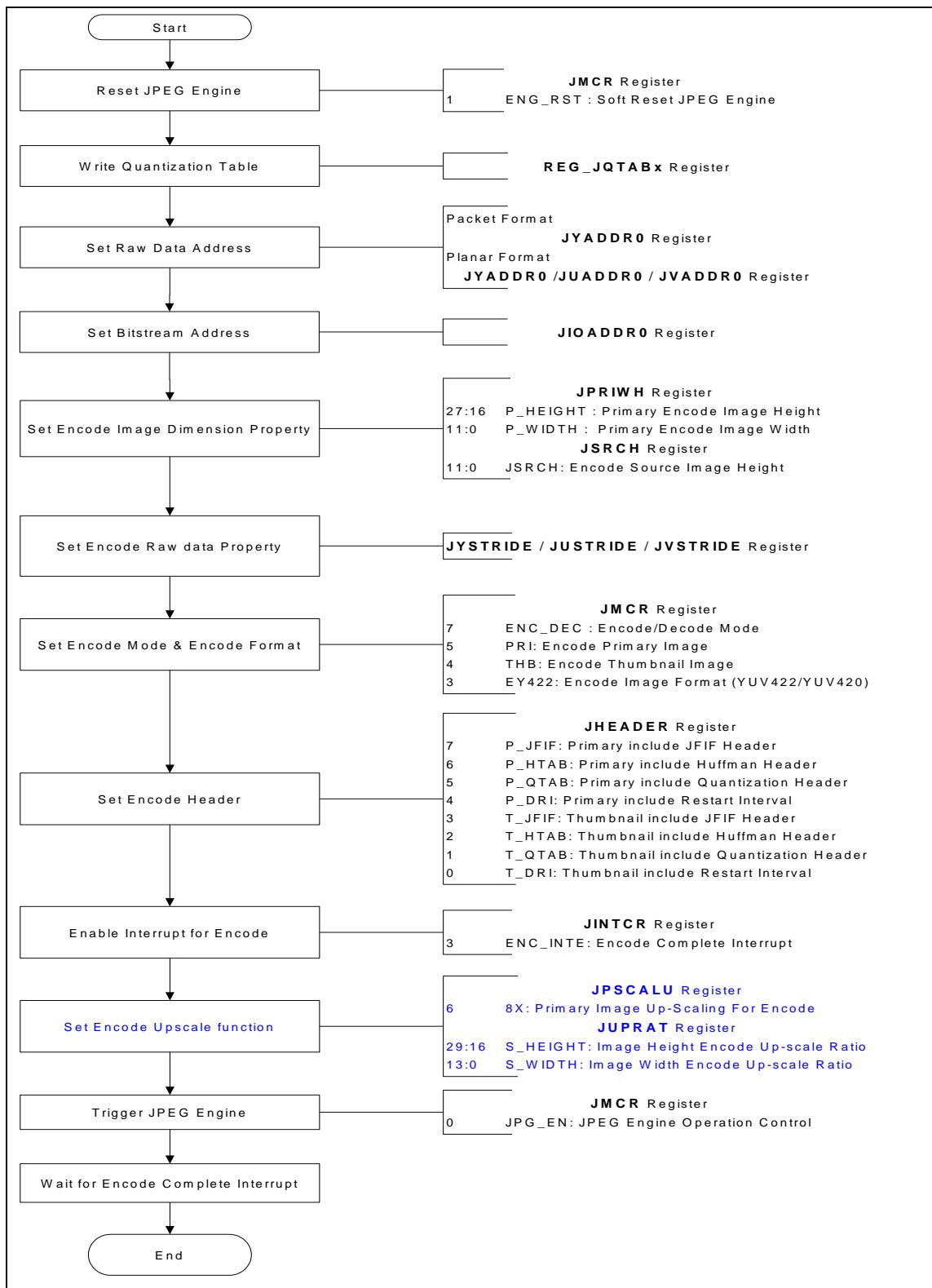
$$\text{Y ratio} = ((\text{Upscale_Height} - 1) / (\text{Height} - 1)) * 1024$$

如果用戶想要使能放大編碼功能,可以在觸發JPEG編碼前,利用以下的設定達成.

1. 設定放大倍率 (JUPRAT)

- S_HEIGHT: 影像高度的放大率
 - S_WIDTH: 影像寬度的放大率
2. 使能編碼放大功能 (JPSCALU[6] = 1)
- 8X: 使能主影像編碼放大功能

當原始資料是緊縮格式時,有時最後一列的資料會和下一行中的第一列資料混合.用戶可以使用較大的比例來避免這種情況.



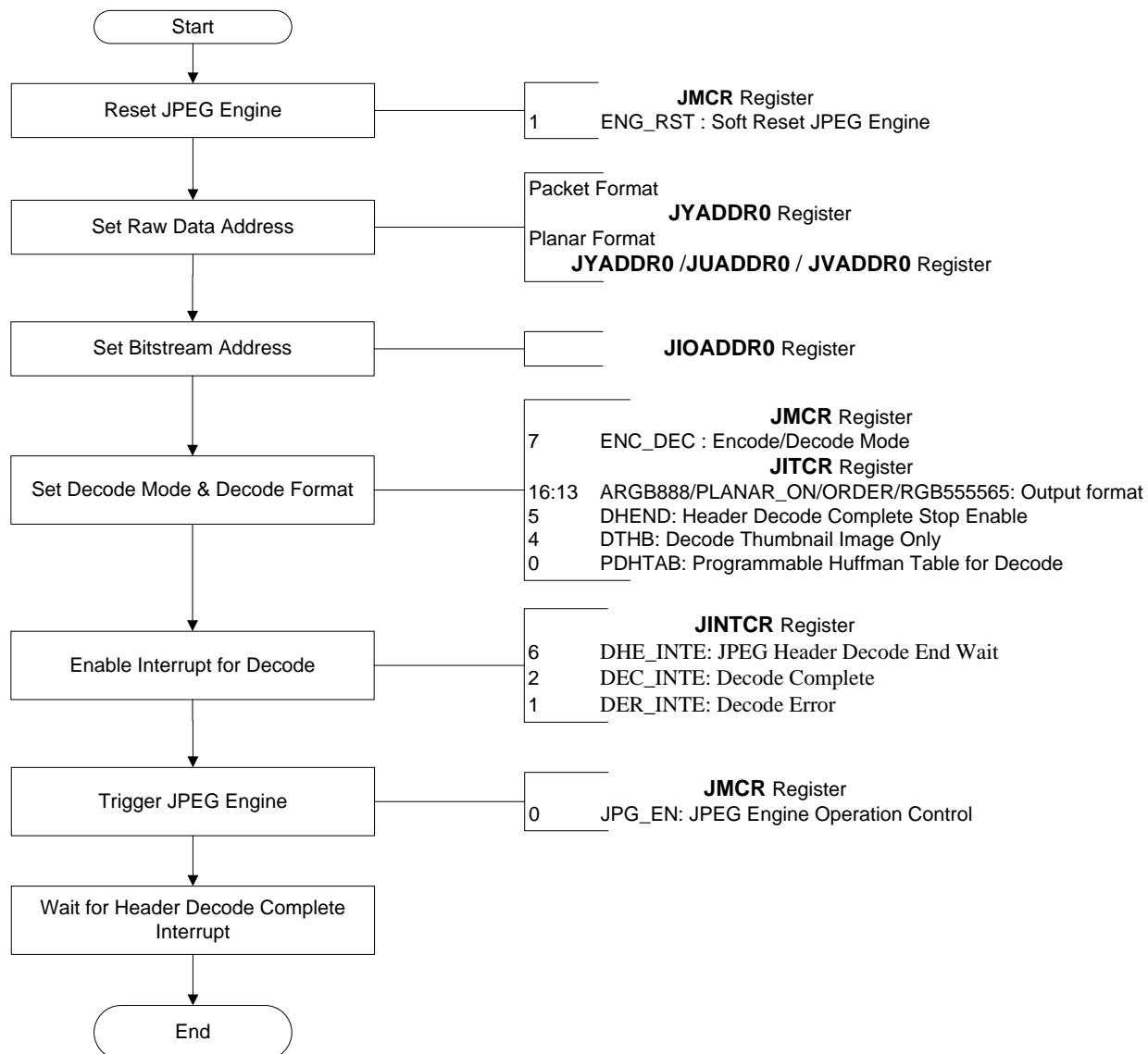
16.5.5 JPEG 解碼

16.5.5.1 一般解碼

解碼的操作流程可以分成兩個部份標頭解碼以及影像解碼.

標頭解碼流程如下:

1. JPEG 復位
2. 設定輸入碼流的起始位址. (JIOADDR0)
3. 設定輸出 YUV 幀緩衝的位址.
 - 平面格式(Planar format)
 - ◆ Y/U/V 起始位址 (JYADDR0 / JUADDR0 / JVADDR0)
 - 緊縮格式(Packet format)
 - ◆ 資料起始位址 (JYADDR0). 當使用平面格式時,YUV 幀緩衝的位址必須在觸發 JPEG 解碼前就要設定好. 當使用緊縮格式時,YUV 幀緩衝位址須要在輸出原始資料(raw data)前設定好.
4. 設定解碼模式和輸出格式
 - 設定解碼模式 (JMCR[7] = 0)
 - 設定解碼主影像或是解碼縮圖影像 (JITCR[4])
 - 使能標頭解碼完成後停止功能 (JITCR[5])
 - 使能解碼時,可編程霍夫曼表功能 (JITCR[0])
 - 設定輸出格式 (JITCR[16:13])
 - ◆ 平面格式(Planar format)
 - ◆ 緊縮格式(Packet format):
RGB888/RGB565/RGB555/YUV422/RGB555R1/RGB555R2/RGB565R1/RGB565R2
5. 使能解碼中斷(JINTCR)
 - 設定 DEC_INTE 為 1
 - 設定 DHE_INTE 為 1
 - 設定 DER_INTE 為 1
6. 觸發開始解碼 (JINTCR)
 - 設定 JMCR[0]為 1,然後再設定 JMCR[0]為 0
7. 等待標頭解碼完成中斷

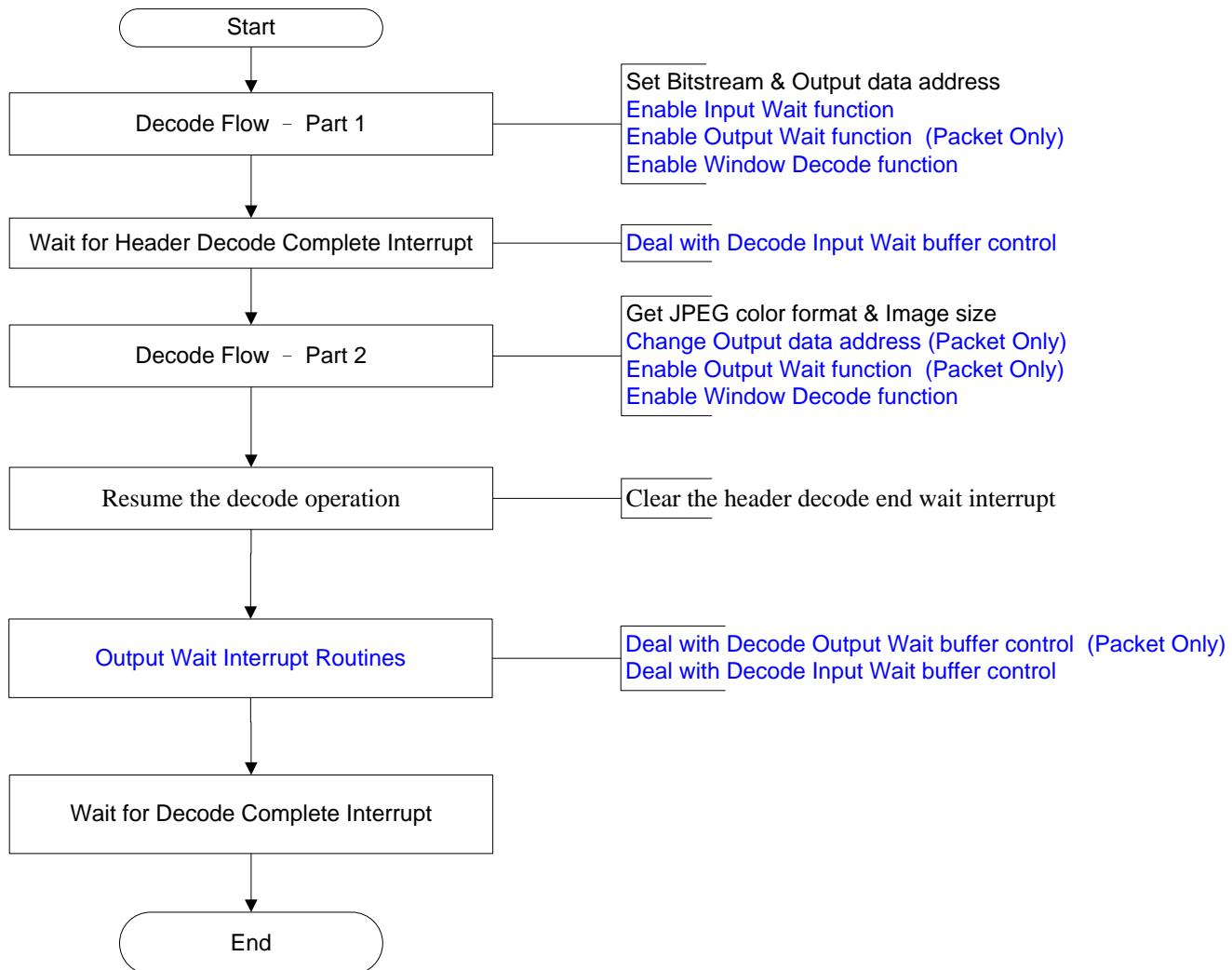


在完成標頭解碼之後,用戶可以得到YUV 顏色格式 (JITCR[10:8]) 以及影像的寬/高 (JDECWH) 的訊息.

影像解碼流程如下:

1. 設定解碼的影像的寬和高(JPRIWH)
 - 寬度的值必須調整成 JPEG 格式或是設定成輸出步幅.
2. 設定資料輸出屬性 (只在緊縮格式(Packet format)下才需要設定)
 - 改變輸出位址 (JYADDR0)
 - 輸出偏移值到 JYSTRIKE . 如果不使能步幅功能,步幅將會等於影像寬度,而偏移值會等於 0.
3. 清除標頭解碼結束的中斷並重新開始解碼

4. 等待解碼完成中斷



16.5.5.2 縮圖解碼

解碼縮圖比例

在下面的公式裡,Width和Height表示原始圖片的寬和高,而Downscale_Width和Downscale_Height表示圖片經過縮小後的寬和高。(縮小比例加1後必須要小於320)

$$X \text{ ratio} = (\text{Downscale_Width} / (\text{Width} - 1)) * 8192$$

$$Y \text{ ratio} = (\text{Downscale_Height} / (\text{Height} - 1)) * 8192$$

例如將18000 x 18000 像素縮到 640 x 480 像素設定如下:

- 水平比例 = $640 / 18000 * 8192 = 291$

- 垂直比例 = $480 / 18000 * 8192 = 218$

在平面格式下(planer format),這個比例如下面公式所列:Width和Height表示原始圖片的寬和高而Downscale_Width和Downscale_Height表示圖片經過縮小後的寬和高. (這個比例最大只能設為8192)

$$X \text{ ratio} = \text{Width} / \text{Downscale_Width} / 2 - 1 \quad (0 \leq X \text{ ratio} \leq 15)$$

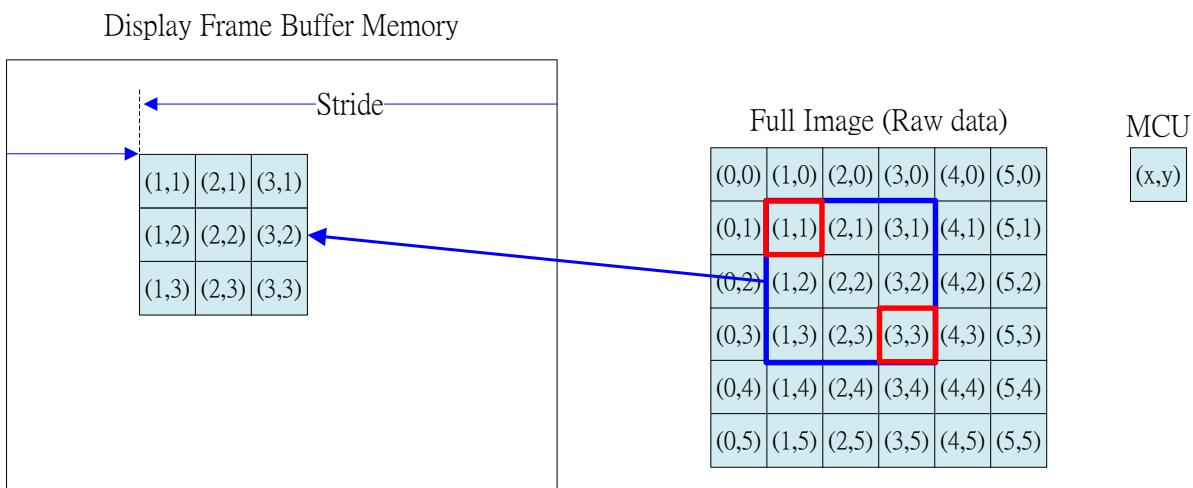
$$Y \text{ ratio} = \text{Height} - \text{Downscale_Height} \quad (0 \leq Y \text{ ratio} \leq 63)$$

如果用戶想要使能解碼縮小功能,用戶可以在清除標頭解碼結束中斷之前使用以下的設定方式:

1. 設定縮小比例
 - 緊縮格式(Packet format) (JUPRAT)
 - 平面格式(Planar format) (JPSCALD)
2. 使能解碼縮小功能 (JPSCALD[15] = 1)

16.5.5.3 視窗解碼

JPEG解碼器支持特定窗解碼模式. 這個功能允許用戶在整張圖中指定特定子視窗區域解碼,如下圖所示. 只有指定的特定視窗區域會被解碼並儲存在幀存儲器裡. 這個功能可以經由設定位WIN_DEC被使能,而且視窗的區域可以在寄存器JWINDEC0~JWINDEC2中被指定. 請注意: 視窗解碼的最小單位是16x16,另外被解碼的圖像區域必須是存在的.

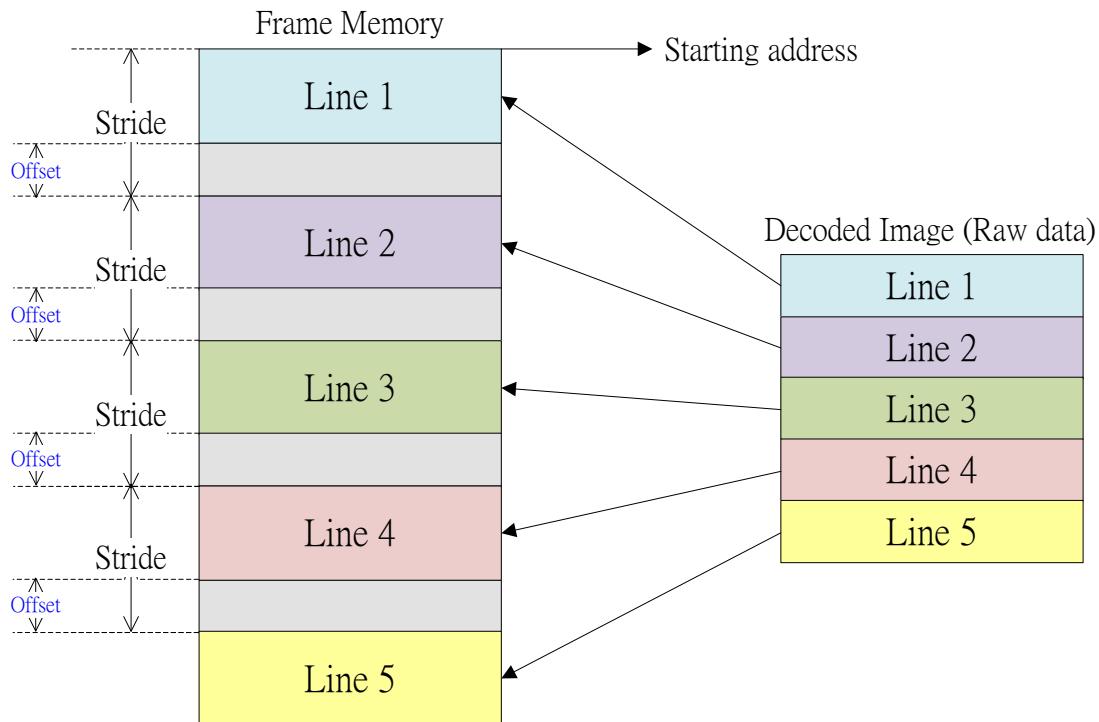


如果用戶想要使能視窗解碼功能時,可以使用下面的方式設定:

1. 設定視窗區域 Set the region window of and End MCU
 - 設定視窗起始座標 (JWINDEC0)
 - 設定視窗結束座標 (JWINDEC1)
 - 設定解碼步幅 (JWINDEC2)

16.5.5.4 解碼步幅功能(只在緊縮格式(Packet Format)時使用)

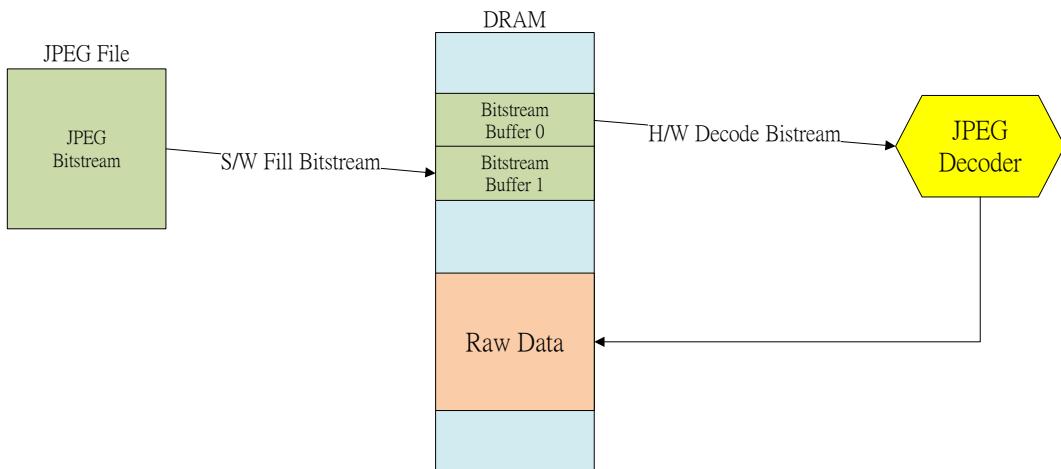
在清除標頭解碼結束中斷之前,必須將步幅設定到JPRIWH取代原始圖片的寬度. 下圖中的Offset指的是步幅和影像寬度的差異(JYStride). 如果Offset為0,表示解碼的原始資料是連續的.



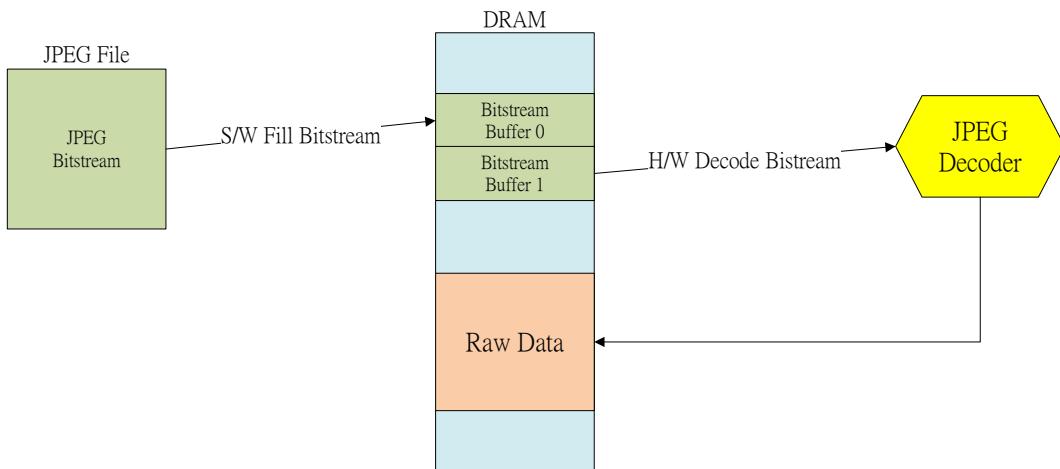
16.5.5.5 軟體解碼輸入等待(Software Decode Input Wait)

當JPEG在解碼模式時,輸入來源是從主機寫入的JPEG碼流.而碼流的緩衝區大小是2K單位的雙緩衝區.全部的緩衝區大小由JMACR裡的BSF_SEL決定. 假設緩衝區大小是2KB(BSF_SEL=1), 當輸入等待中斷發生時,在操作JPEG之前,主機必須先填入1KB的碼流到雙緩衝區的其中一個緩衝區內.

1. JPEG解碼Buffer 0內的資料,同時軟體將碼流資料填到Buffer1內.



2. JPEG 解碼 Buffer 1 的資料,同時軟體將碼流資料填到 Buffer 0.



3. 重複步驟 1 和 2 直到解碼完成中斷發生.

16.5.5.6 解碼輸出等待(Decode Output Wait)

當沒有足夠的空間可以儲存解碼輸出的原始資料時,JPRG支持輸出部份資料的功能. 用戶可以通過指定若干個輸出數據的地址和大小設置得到整個數據. 使用這個功能時, 用戶可以得到的JPEG解碼圖像會比可用連續的存儲器空間更大.

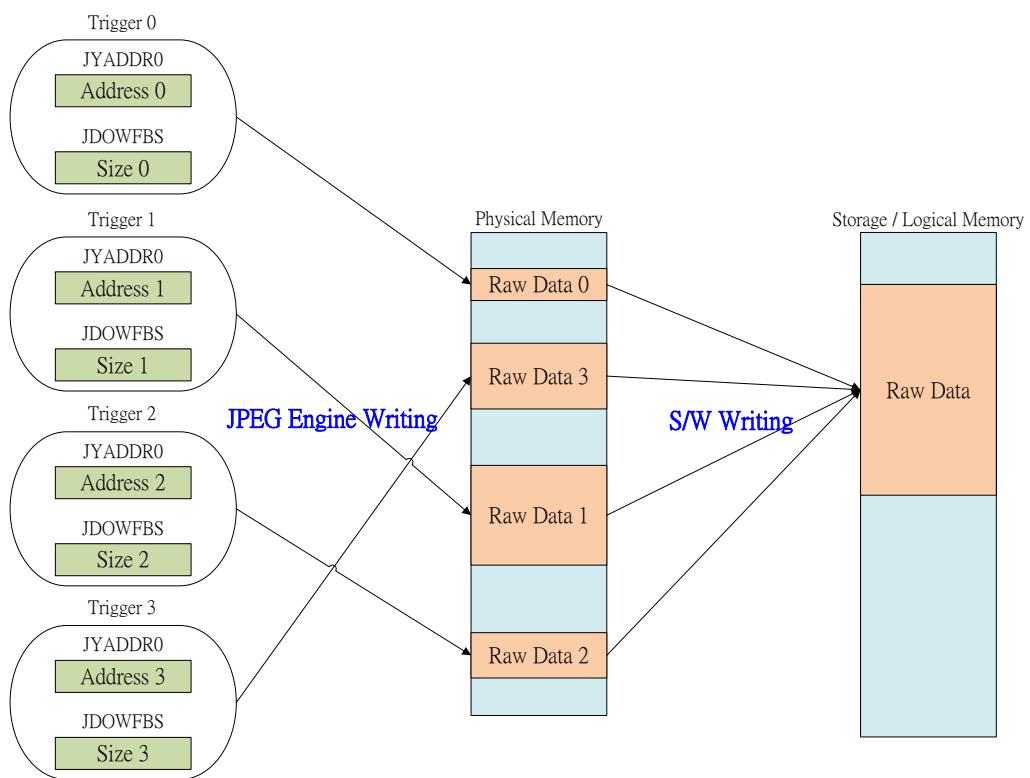
如果用戶想要使能解碼輸出等待功能, 用戶可以在清除標頭解碼結束中斷之前(開始輸出資料之前), 增加額外設定如下:

1. 設定輸出等待的位址和大小
 - 位址: JYADDR0
 - 設定解碼輸出資料大小: JDOWFBS
 - ◆ 單位是 Word.
 - ◆ 資料大小必須是 MCU 線數的倍數
2. 使能解碼輸出等待中斷(JINTCR)
 - 設定 DOW_INTE 為 1 (使能解碼輸出等待中斷)
3. 觸發解碼輸出等待功能
 - 設定 Dec_Scatter_Gather (JITCR[18])為 1

在 Dec_Scatter_Gather 被設置後, 只能在 IP 復位或是當輸出大小等於JDOWFBS 的值時被清除. 假設 Dec_Scatter_Gather被設定時, 用戶可以把JDOWFBS設成0xFFFFFFFF讓JPEG解碼器忽略解碼輸出功能.

16.5.5.7 解碼輸出等待中斷處理

1. 等待解碼輸出中斷(DOW_INTS)
 - 它代表緩衝區已滿.
 - 用戶可以從 JDOWFBS 得到緩衝區的大小,而緩衝區的起始位址則可以從 JYADDR0 得到.
2. 清除解碼輸出中斷(DOW_INTS)
3. 設定下一個輸出等待位址和大小
 - 位址: JYADDR0
 - 設定解碼輸出資料大小: JDOWFBS
 - ◆ 單位是 Word.
 - ◆ 資料大小必須是 MCU 線資料大小的倍數
4. 觸發解碼輸出等待功能
 - 設定 Dec_Scatter_Gather (JITCR[18])為 1
5. 重覆步驟 1~4 直到解碼完成中斷發生.



17 鍵盤接口 (KPI)

17.1 概述

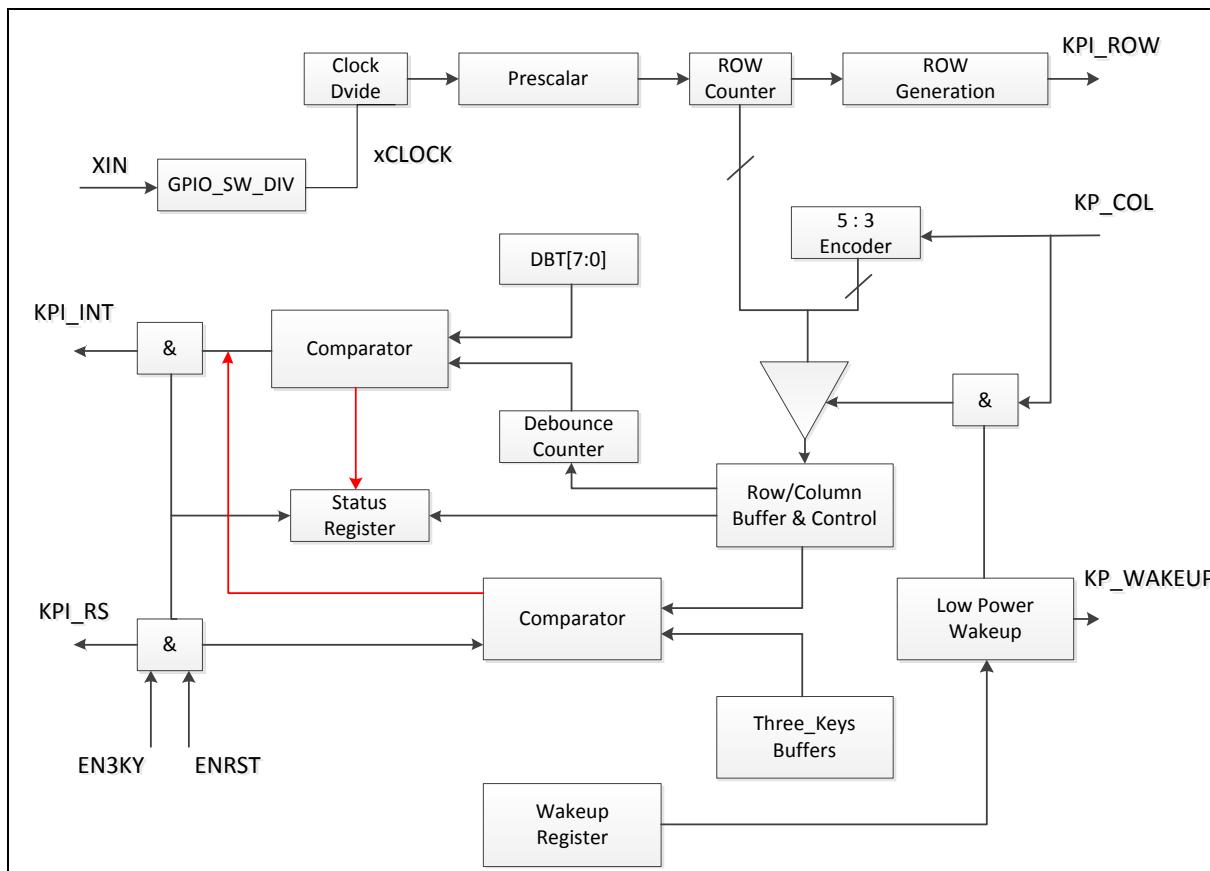
鍵盤接口 (KPI) 是至少2行至16行掃描輸出和至少小1列至4列掃描輸入。按下或釋放的陣列中的任何鍵將產生一個中斷。

在KPI支持多個鍵的按下或釋放的掃描中斷，並指定INT_3KEYs中斷系統復位。如果3個按鍵與在KPI3KCONF定義的3個按鍵匹配，就會產生一個中斷並且會使芯片復位。每當它檢測到鍵盤按下或釋放或從IDLE狀態或三鍵復位醒來的任意鍵中斷。用戶可以知道通過查詢KPISTATUS寄存器中的中斷源。

17.2 特性

- 矩陣鍵盤接口（最大 16x4 陣列，和最小 2x1 陣列）
- 可編程去抖動偵測週期時間
- 低功率喚醒模式
- 可編程三鍵復位
- 產生中斷，並更新所有的按鍵（最多 64 個按鍵，最小的 2 鍵）的狀態(按下/釋放)
- 支持同步型 LCD16 位總線份額 KPI

17.3 方塊圖



17.4 寄存器

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

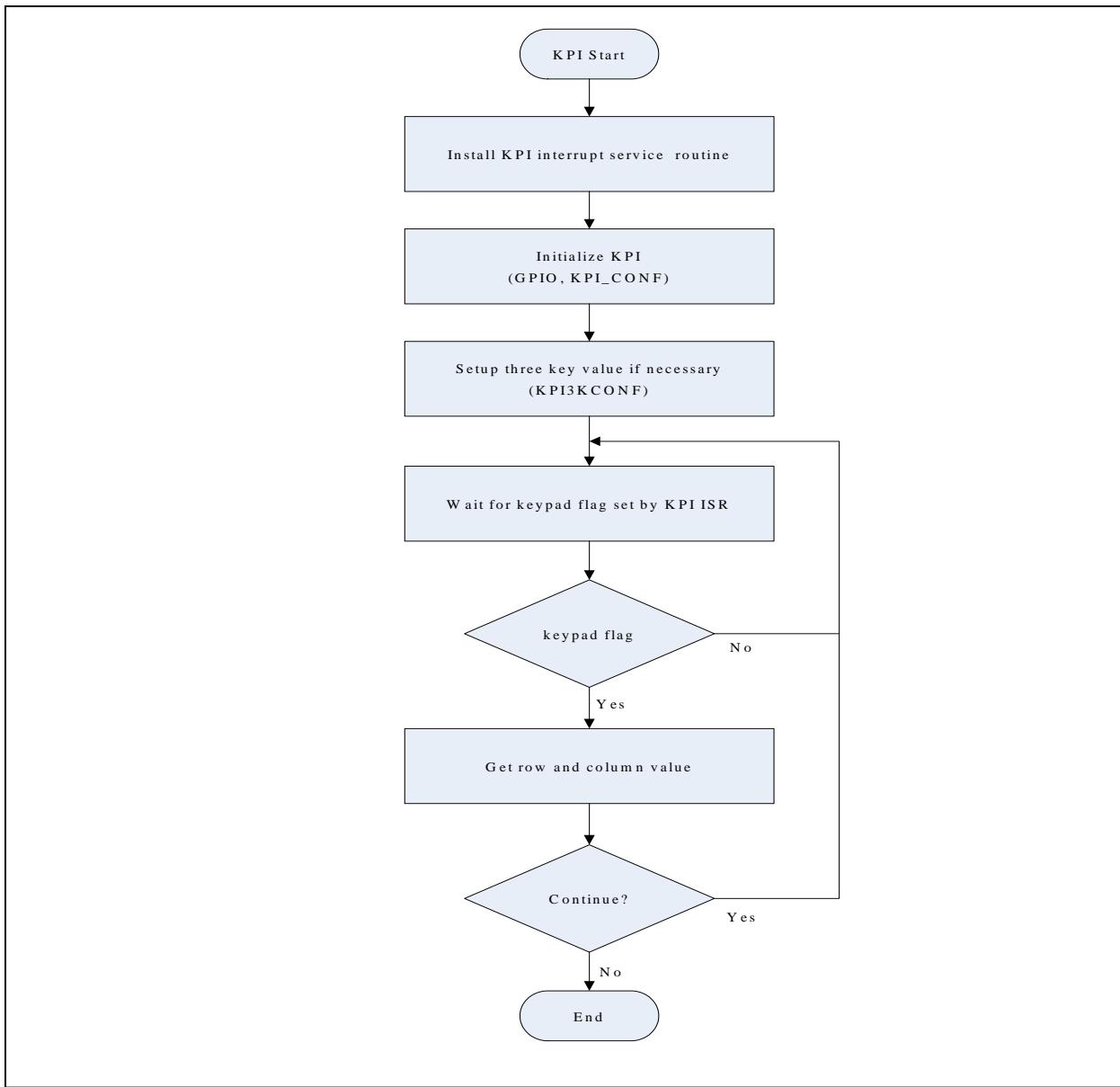
Register	Offset	R/W	Description	Reset Value
KPI Base Address:				
KPI_BA = 0xB800_8000				
KPICONF	KPI_BA+0x00	R/W	Keypad Configuration	0x0000_0000
KPI3KCONF	KPI_BA+0x04	R/W	Keypad 3-Keys Configuration	0x0000_0000
KPISTATUS	KPI_BA+0x08	R	Keypad Status	0x0000_0000
KPIRSTC	KPI_BA+0x0C	R/W	Keypad Reset Period Controller	0x0000_0000
KPIKEST	KPI_BA+0x10	R	Keypad Key State Indicator	0xFFFF_FFFF
KPIKPE	KPI_BA+0x18	R/W	Press Key Event Indicator	0x0000_0000
KPIKRE	KPI_BA+0x20	R/W	Release Key Event Indicator	0x0000_0000
KPIPRESCALDIV	KPI_BA+0x28	R/W	Pre-Scale Divider	0x0000_001F

17.5 功能描述

17.5.1 控制器配置

鍵盤接口(KPI)的用法過程描述如下:

1. 使用 MFP_GPH4~MFP_GPH15 寄存器, 配置 KPI 引腳
2. 設置 KPICONF 的 KROW 和 KCOL 位來設置的行數和列數
3. 設置 KPICONF 寄存器中的 DBCLKSEL 和 PRESCALE 位來設定行的掃描時間，並設置 DB_EN 位來使能去抖動功能
4. 可以設置 KPICONF 的 ODEN 位寄存器，使能開漏輸出模式，防止同一欄上多鍵同時按下時發生短路. 在這種模式下，作為列的 GPIO 需始能內部上拉電阻.
5. 依應用程式需要設置 KPI3KCONF
6. 設置，KPICONF 寄存器中的 INTEN，RKINTEN 和 PKINTEN 位來使能鍵盤的釋放和壓按的中斷
7. 設置 KPICONF 的 ENKP 位寄存器，使鍵盤掃描
8. 等待鍵盤標誌由 KPI 設定 ISR
9. 獲取 KPI 的行和列值寄存器 KPISTATUS，然後再回到第 8 步



17.5.2 喚醒功能

設置KPICONF寄存器的WAKEUP位可以啟用喚醒功能。當芯片在掉電模式，按任意鍵的芯片將被喚醒

18 顯示控制器 (LCM)

18.1 概述

顯示控制器的主要目的是用於對視頻/圖像數據顯示到液晶顯示裝置或連接與外部電視編碼器。視頻/圖像數據源可以來自圖像傳感器，JPEG解碼器和已經存儲在系統存儲器（SDRAM）的OSD圖案。

所述顯示控制器的輸入數據格式可以是packet YUV422，packet RGB444，packet RGB565，packet RGB666，和packet RGB888。

在OSD（屏幕顯示）功能，支持packet YUV422和8/16/24位直接色彩模式。

該LCD控制器支持同步型和MPU型LCDM。

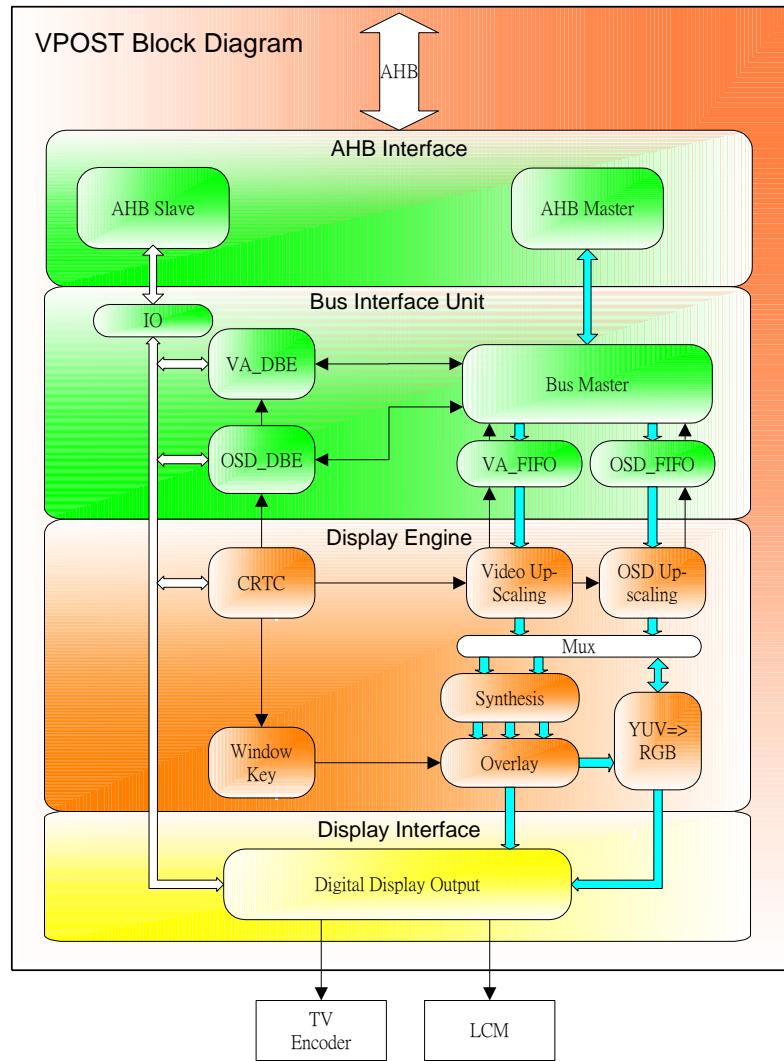
該LCD控制器是一個總線主控，並且可以在無需CPU干預轉移系統內存（SDRAM）的顯示數據。

18.2 特性

- 支援輸入數據格式
 - YUV422, YUV444
 - RGB444, RGB565, RGB666, RGB888
- 支援輸出數據格式
 - YUV422, YUV444
 - RGB444, RGB565, RGB666, RGB888
- 最大支援解析度：2048x2048
- 圖像大小調整
 - 水平調整：1至8倍
 - 垂直調整：1至8倍
- 可轉換RGB565, RGB888, YUV422 的顯示數據至 RGB444, RGB565, RGB666, RGB888
- 可轉換全域YUV至CCIR601格式
- 三個OSD圖形或文本覆蓋窗的支持
- 支持CCIR-656（帶報頭），CCIR-601（與HSYNC和VSYNC）8位/ 16位的YUV數據輸出格式與外部TV編碼器連接
- 支援同步型和MPU 型的屏

- 支援8/9/16/18/24位數據輸出至80/68型屏模組

18.3 方塊圖



18.4 寄存器

Register	Offset	R/W	Description	Reset Value
LCM Base Address:				
LCM_BA = 0xB000_8000				
DCCS	LCM_BA + 0x00	R/W	Display Controller Control and Status Register	0x0000_0000
DEVICE_CTRL	LCM_BA + 0x04	R/W	Display Output Device Control Register	0x0000_00E0
MPULCD_CMD	LCM_BA + 0x08	R/W	MPU-Interface LCD Write Command Register	0x0000_0000

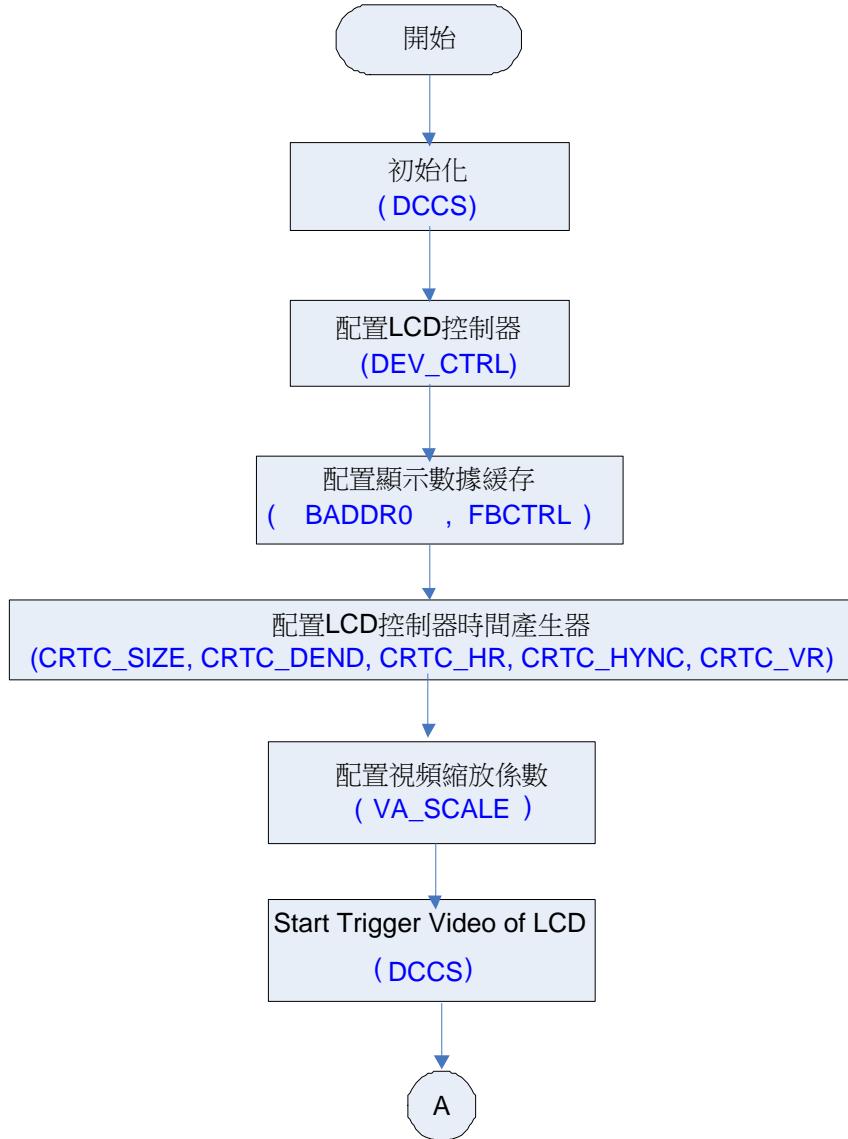
INT_CS	LCM_BA + 0x0C	R/W	Interrupt Control/Status Register	0x0000_0000
CRTC_SIZE	LCM_BA + 0x10	R/W	CRTC Display Size Register	0x0000_0000
CRTC_DEND	LCM_BA + 0x14	R/W	CRTC Display Enable End Register	0x0000_0000
CRTC_HR	LCM_BA + 0x18	R/W	CRTC Internal Horizontal Retrace Timing Register	0x0000_0000
CRTC_HSYNC	LCM_BA + 0x1C	R/W	CRTC Horizontal Sync Timing Register	0x0000_0000
CRTC_VR	LCM_BA + 0x20	R/W	CRTC Internal Vertical Retrace Timing Register	0x0000_0000
VA_BADDR0	LCM_BA + 0x24	R/W	Video Stream Frame Buffer-0 Starting Address Register	0x0000_0000
VA_BADDR1	LCM_BA + 0x28	R/W	Video Stream Frame Buffer-1 Starting Address Register	0x0000_0000
VA_FBCTRL	LCM_BA + 0x2C	R/W	Video Stream Frame Buffer Control Register	0x0000_0000
VA_SCALE	LCM_BA + 0x30	R/W	Video Stream Scaling Control Register	0x0000_0000
VA_TEST	LCM_BA + 0x34	R/W	Test Mode Control Register	0x0000_0000
VA_WIN	LCM_BA + 0x38	R/W	Video Stream Active Window Coordinates Register	0x0001_07FF
VA_STUFF	LCM_BA + 0x3C	R/W	Video Stream Stuff Register	0x0000_0000
OSD_WINS	LCM_BA + 0x40	R/W	OSD Window Starting Coordinates Register	0x0000_0000
OSD_WINE	LCM_BA + 0x44	R/W	OSD Window Ending Coordinates Register	0x0000_0000
OSD_BADDR	LCM_BA + 0x48	R/W	OSD Stream Frame Buffer Starting Address Register	0x0000_0000
OSD_FBCTRL	LCM_BA + 0x4C	R/W	OSD Stream Frame Buffer Control Register	0x0000_0000
OSD_OVERLAY	LCM_BA + 0x50	R/W	OSD Overlay Control Register	0x0000_0000
OSD_CKEY	LCM_BA + 0x54	R/W	OSD Overlay Color-Key Pattern Register	0x0000_0000
OSD_CMASK	LCM_BA + 0x58	R/W	OSD Overlay Color-Key Mask Register	0x0000_0000
OSD_SKIP1	LCM_BA + 0x5C	R/W	OSD Window Skip1 Register	0x0000_0000
OSD_SKIP2	LCM_BA + 0x60	R/W	OSD Window Skip2 Register	0x0000_0000
OSD_SCALE	LCM_BA + 0x64	R/W	OSD Scaling Control Register	0x0000_0000
MPU_VSYNC	LCM_BA + 0x68	R/W	MPU Vsync Control Register	0x0000_0000
HC_CTRL	LCM_BA + 0x6C	R/W	Hardware Cursor Control Register	0x0000_0000
HC_POS	LCM_BA + 0x70	R/W	Hardware Cursor Position Register	0x0000_0000
HC_WBCTRL	LCM_BA + 0x74	R/W	Hardware Cursor Window Buffer Control Register	0x0000_0000
HC_BADDR	LCM_BA + 0x78	R/W	Hardware Cursor Memory Base Address Register	0x0000_0000
HC_COLOR0	LCM_BA + 0x7C	R/W	Hardware Cursor Color RAM 0 Register	0x0000_0000
HC_COLOR1	LCM_BA + 0x80	R/W	Hardware Cursor Color RAM 1 Register	0x0000_0000
HC_COLOR2	LCM_BA + 0x84	R/W	Hardware Cursor Color RAM 2 Register	0x0000_0000
HC_COLOR3	LCM_BA + 0x88	R/W	Hardware Cursor Color RAM 3 Register	0x0000_0000

18.5 功能描述

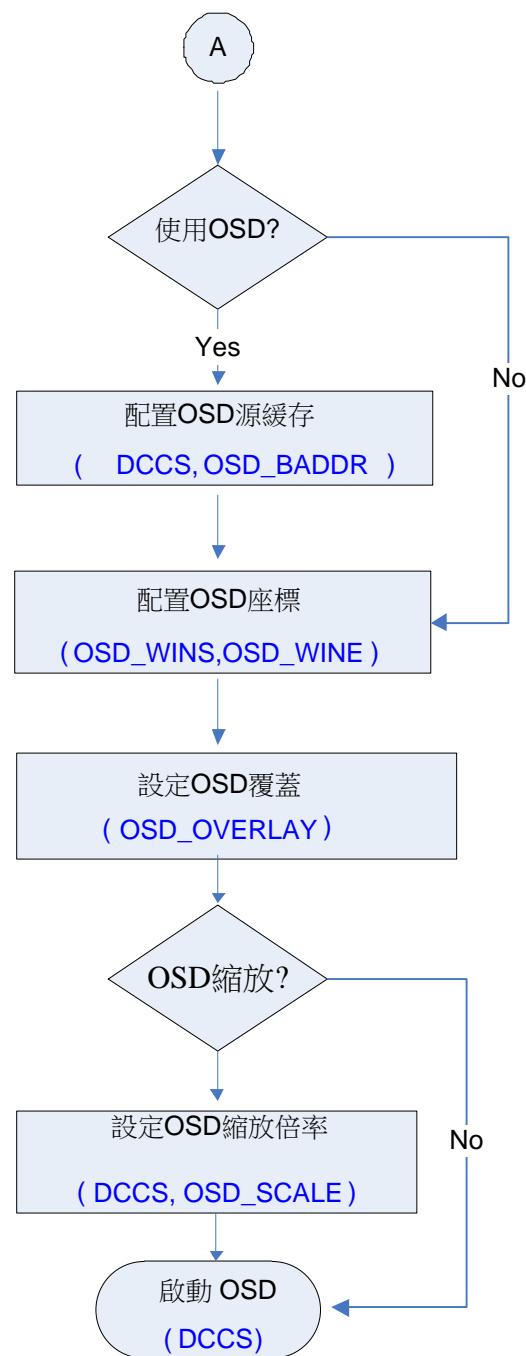
18.5.1 LCD 控制器編程流程

本節介紹LCD控制器的軟體編程流程。跟著以下的步驟，以避免不可預知的情況。

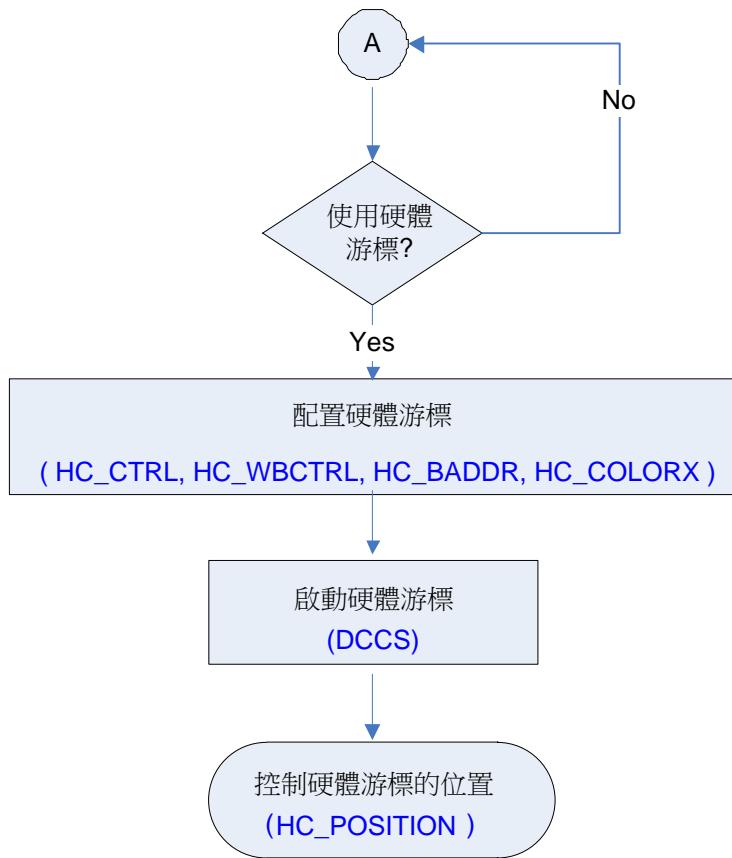
設置Video部份：



設置OSD部份：



設置硬體游標部份：

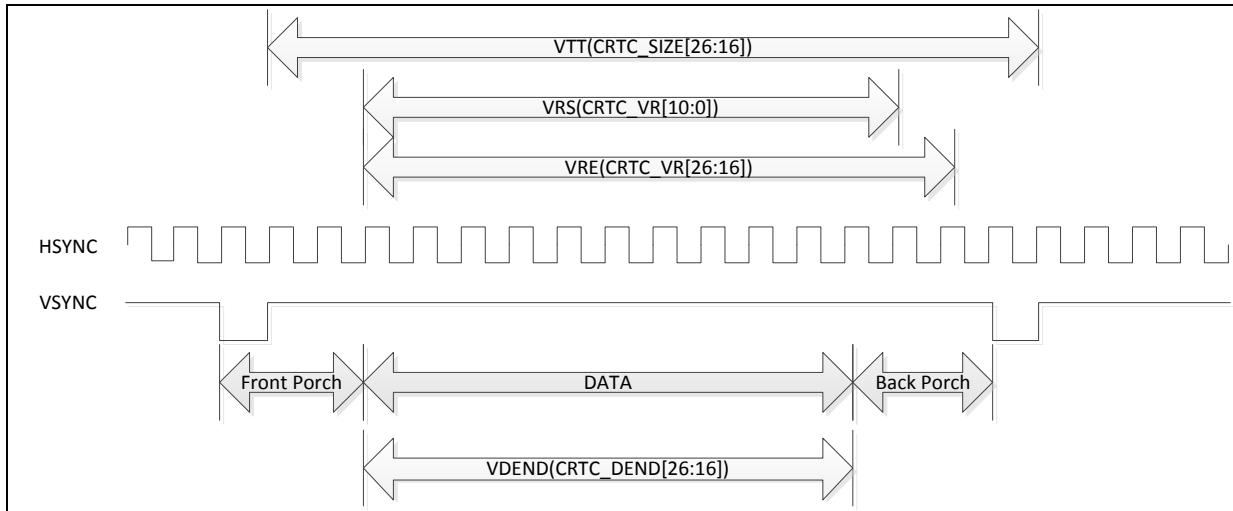
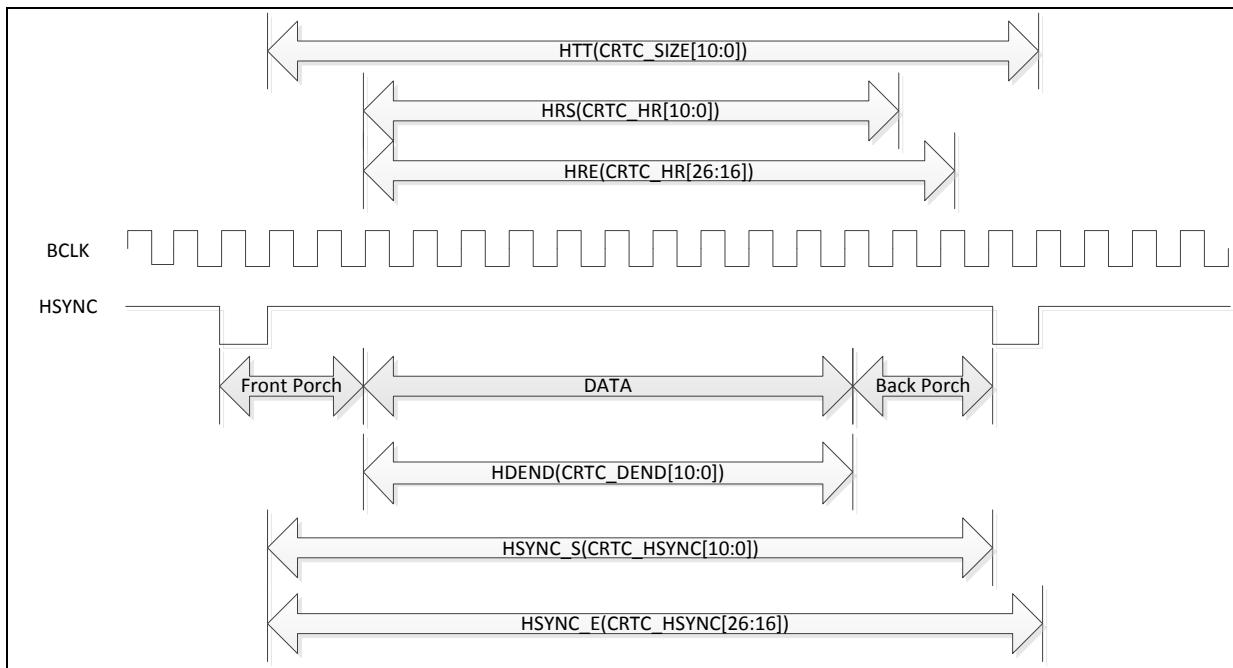


18.5.2 初始化和配置 LCD 控制器

初始化整個系統環境，設置中斷服務程序，然後初始化LCD控制器。

編程過程如下：

1. 配置需要的 LCD 控制器，比如為視頻時鐘，多功能引腳選擇，等全局設置。
 2. 復位 LCD 控制器。
 3. 根據分辨率設置顯示器的幀緩衝 - VA_BADDR0，VA_FCTRL。
 4. 根據 LCM 的模式來設置顯示為持續或單一模式 SINGLE(DCCS[7])。
 5. 設置視頻輸入數據源格式 VA_SRC(DCCS[10 : 8])。
 6. 根據液晶設置 LCM 設備相關寄存器(DEVICE_CTRL)。
 7. 設置時間產生器(CRTC_SIZE，CRTC_DEND，CRTC_HR，CRTC_HSYNC，CRTC_VR)。
- 垂直、水平時間寄存器映射到視頻信號如下圖。



8. 設置視頻縮放因子(VA_SCALE)。
9. 啟動 LCD 控制器 VA_EN(DCCS[1]) 及 DISP_OUT_EN(DCCS[3])。

以連接320x240分辨率的屏舉例說明如下：

```
//配置LCD控制器時鐘源&時鐘
APLLCON = 0xc0004018;
CLKDIV1 = (CLKDIV1 & ~0x1f) | 0x12;           //使用APLL時鐘源並且輸出50MHz

//切換LCD控制管腳
MFP_GPG_L = (MFP_GPG_L & ~0xFF000000) | 0x22000000; //GPG6(CLK), GPG7(HSYNC)
```

```

MFP_GPG_H = (MFP_GPG_H & ~0xFF) | 0x22; //GPG8(VSYNC), GPG9(DEN)

//切換LCD 數據管腳 - 16位元
MFP_GPA_L = 0x22222222; //GPA0 ~ GPA7 (DATA0~7)
MFP_GPA_H = 0x22222222; //GPA8 ~ GPA15 (DATA8~15)

//LCD控制器復位
SYS_AHBIPRST |= (0x1 << 9);
SYS_AHBIPRST &= ~(0x1 << 9);

VA_BADDR0 = 0x80001000;
VA_FCTRL = (VA_FCTRL & ~(0x07FF07FF)) | (320/2 << 16) | (320/2);

DCCS = DCCS & ~(0x7 << 8) | 0x4; //選擇RGB565格式

// 設置同步高色彩、數據管腳寬度為16/18位元
DEVICE_CTRL = 0;
DEVICE_CTRL |= (0x1 << 26) | (0x2 << 24) | (0x6 << 5) | (0x1 << 19);

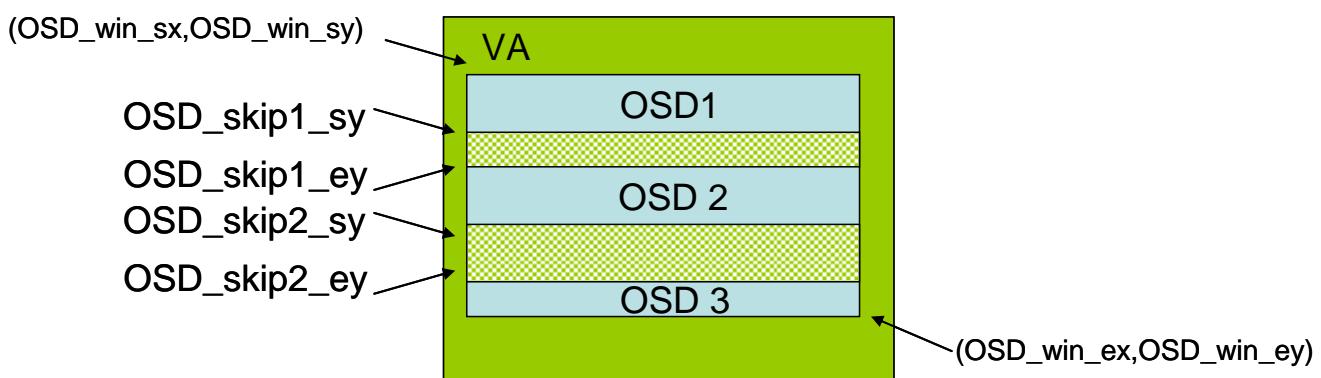
//設置時間
CRTC_SIZE = 0x00F40150; //CRTC_SIZE
CRTC_DEND = 0x00F00140; //CRTC_DEND
CRTC_HR = 0x01450141; //CRTC_HR
CRTC_HSYNC = 0x014F014D; //CRTC_HSYNC
CRTC_VR = 0x00F300F2; //CRTC_VR

DCCS |= 0xA; //啟動LCD控制器

```

18.5.3 配置 OSD 控制器

通過以下步驟可以開啟OSD並覆蓋於視頻數據上，示意圖如下：



1. 設置的 OSD 輸入數據的源格式 OSD_SRC(DCCS[14:12])。
 2. 設置的 OSD 輸入緩衝器的起始緩衝地址，和幀緩衝器操作(OSD_BADDR，OSD_FBCTRL)。
 3. 設置 OSD 的坐標(OSD_WIN_S，OSD_WIN_E_)。
 4. 啟用顏色鍵(OSD_OVERLAY[8]，控制覆蓋區域的顯示效果 (OSD_OVERLAY[3 : 0]))。
- 顯示條件是下表中所描述。

Color-Key	Match	OCR1	OCR0	Display
0	X	X	X	Video
1	0	X	0	Video
1	0	X	1	OSD
1	0	X	2	Video+OSD
1	1	0	X	Video
1	1	1	X	OSD
1	1	2	X	Video+OSD

註：Match是指OSD數據和顏色鍵的數據相符合。

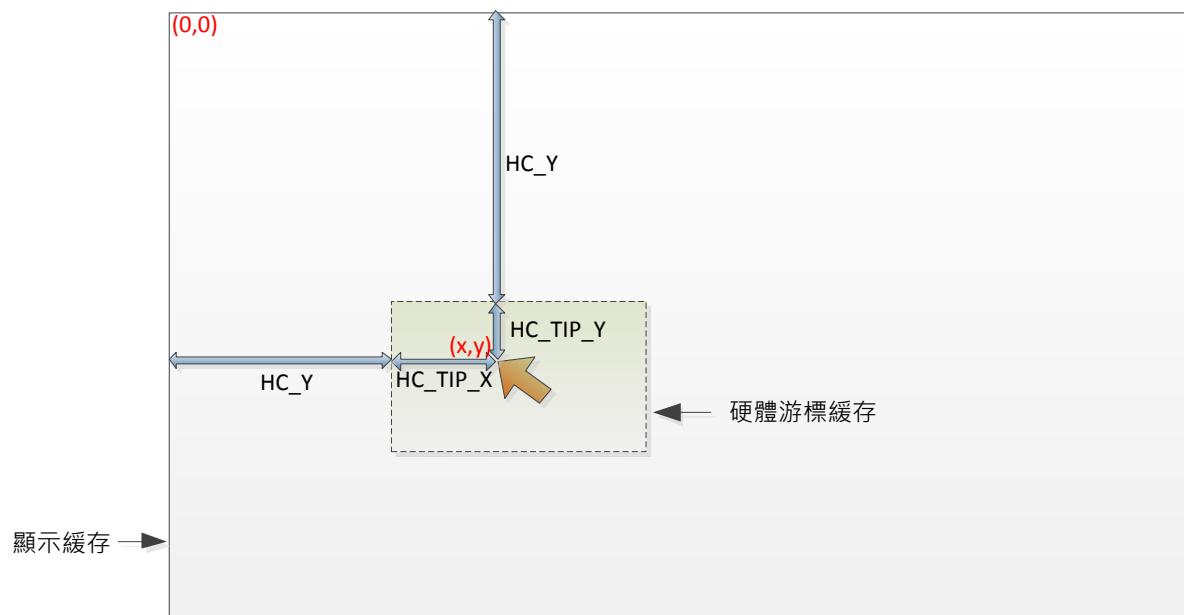
5. 如果需要閃爍的功能，可透過配置 BLI_ON(OSD_OVERLAY[9])位和 BLINK_VCNT(OSD_OVERLAY[23:16])位來實現。

18.5.4 配置硬體游標功能

如果要啟用硬體游標功能，以下過程可啟動其功能：

1. 設置游標的模式，並設置提示的位置。(HC_CTRL)
2. 控制游標緩衝區的操作(HC_WBCTRL，HC_BADDR)
3. 控制硬體游標塊的位置(HC_POS)

下圖顯示顯示緩存及硬體游標緩存的相對位置：



19 MTP 控制器

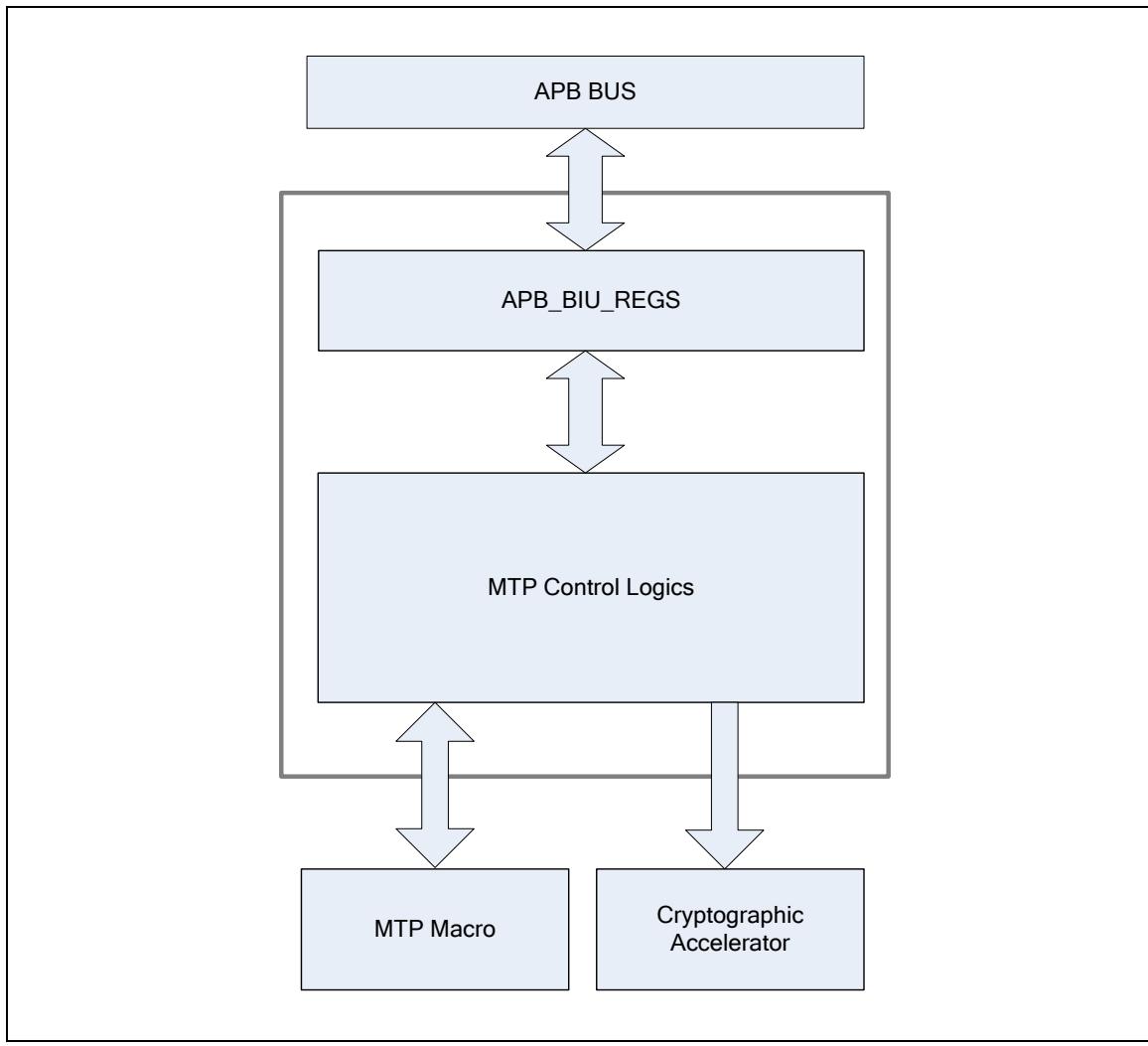
19.1 概述

MTP (Multi-Time Programmable；可多次燒錄) 控制器為NUC970 的加密加速器提供了一個簡便的功能，用以使用和燒錄 256 位密鑰。在 NUC970 芯片中配備有 MTP EEPROM，它可以被燒錄的 15 次，每一次燒錄會覆蓋前一次燒錄，只有最後一次燒錄的內容是有效的。每次用戶可燒錄 256 位密鑰和 8 位用戶定義數據，此 256 位密鑰是只可燒錄不可讀取的，只有 NUC970 加密加速器可以使用 MTP 密鑰。燒錄的 8 位用戶定義數據可供用戶定義，用來識別 MTP 密鑰的用途。MTP 控制器並支持上鎖功能，上鎖之後就不允許再次燒錄（即使燒錄次數未達15次），用以保護燒錄的密鑰和用戶定義字段的內容不被覆寫。

19.2 特性

- 支持MTP EEPROM燒錄
 - ◆ 256位密鑰和8位用戶定義數據。
 - ◆ 可燒錄次數15次。
- 支援上鎖功能

19.3 方塊圖



MTP 控制器方塊圖

19.4 寄存器

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

Register	Offset	R/W	Description	Reset Value
MTP_BA = 0xB800C000				
MTP_KEYEN	MTP_BA+0x00	R/W	Key Enable Register	0x0000_0000
MTP_USERDATA	MTP_BA+0x0c	R/W	MTP User Defined Data Register	0x0000_0000
MTP_KEY0	MTP_BA+0x10	W	MTP KEY 0 Register	0x0000_0000
MTP_KEY1	MTP_BA+0x14	W	MTP KEY 1 Register	0x0000_0000
MTP_KEY2	MTP_BA+0x18	W	MTP KEY 2 Register	0x0000_0000

MTP_KEY3	MTP_BA+0x1c	W	MTP KEY 3 Register	0x0000_0000
MTP_KEY4	MTP_BA+0x20	W	MTP KEY 4 Register	0x0000_0000
MTP_KEY5	MTP_BA+0x24	W	MTP KEY 5 Register	0x0000_0000
MTP_KEY6	MTP_BA+0x28	W	MTP KEY 6 Register	0x0000_0000
MTP_KEY7	MTP_BA+0x2c	W	MTP KEY 7 Register	0x0000_0000
MTP_PCYCLE	MTP_BA+0x30	R/W	MTP Program Cycle Program Count Register	0x0000_60AE
MTP_CTL	MTP_BA+0x34	R/W	MTP Control Register	0x0000_0000
MTP_PSTART	MTP_BA+0x38	R/W	MTP Program Start Register	0x0000_0000
MTP_STATUS	MTP_BA+0x40	R	MTP Status Register	0x0000_0000
MTP_REGLCTL	MTP_BA+0x50	R/W	MTP Register Write-Protection Control Register	0x0000_0000

19.5 功能描述

19.5.1 使用 MTP

MTP 控制器介面連接於 APB，用戶必須先使能 MTPC (CLK_PCLKEN1[26]) 位，才能夠操作 MTP 控制器。

MTP 寄存器是寫入保護的，在開始對 MTP 控制器進行操作之前，必須先對 MTP_REGLCTL 寄存器連續寫入 0x59，0x16，0x88 三個值，才可以解鎖對 MTP 寄存器的寫入保護。對 REGLCTL[0] (MTP_REGLCTL[0]) 寫 1 可以重啟對 MTP 寄存器的寫入保護。

19.5.2 MTP 密鑰

NUC970 提供了 256 位的 MTP 密鑰，經由 MTP 燒錄程序燒錄到 MTP 控制器自帶的 EEPROM 中。此 EEPROM 可以被燒錄的 15 次，每一次燒錄會覆蓋前一次燒錄，只有最後一次燒錄的內容是有效的。也就是說，系統只會存在唯一一把 MTP 密鑰，或者不存在 (EEPROM 空白位燒錄)。

用戶執行 MTP 使能程序之後，可以從 PRGCNT (MTP_STATUS[19:16]) 得知目前 EEPROM 已燒錄次數，從而判斷還有剩餘幾次的燒錄機會。PRGCNT 為 0 表示尚未燒錄 MTP 密鑰，PRGCNT 的最大值為 15。

如果 NUC970 芯片尚未被燒錄 MTP 密鑰，那麼用戶執行 MTP 使能程序之後，可以看到 MTPEN (MTP_STATUS[0]) 位及 NONPRG (MTP_STATUS[2]) 位均被設置為 1。此時，用戶可以執行 MTP 燒錄密鑰程序。

如果 NUC970 芯片已被燒錄 MTP 密鑰，那麼用戶執行 MTP 使能程序無誤之後，可以看到 MTPEN (MTP_STATUS[0]) 位及 KEYVALID (MTP_STATUS[1]) 位均被設置為 1。此時，用戶可以將 MTP 密鑰作為 AES 密鑰或 SHA 比對使用。如果 LOCKED (MTP_STATUS[3]) 位為 0，而且 PRGCNT (MTP_STATUS[19:16]) 小於 15，那麼用戶可以在此時進行 MTP 燒錄密

鑰程序。

19.5.3 用 戶 定 義 數 據

MTP 控制器除了 256 bits 大小的 MTP 密鑰本身之外，還提供了 8 bits 大小的用 戶 定 義 數 據，目的是用來識別 MTP 密鑰的用途。在燒錄 MTP 密鑰的時候，此用 戶 定 義 數 據 也會被同時燒錄。因此，用 戶 在燒錄 MTP 密鑰時，除了需要將密鑰寫入 MTP_KEY0 ~ MTP_KEY7 寄存器之外，同時還需要將用 戶 定 義 數 據寫入 MTP_USERDATA 寄存器。

用 戶 可以自行決定此用 戶 定 義 數 據所表示的意義，在使用 MTP 密鑰時，便可以藉此識別 MTP 密鑰的使用對象。但是，由於 NUC970 的 IBR (Internal Boot ROM) 程式也會透過此用 戶 定 義 數 據，來決定在載入 NAND/SPI/eMMC 內的程式碼的時候是否要做 AES 解密或 SHA 比對，因此，用 戶 必須注意不要去動用到 IBR 所使用的幾個 bit。IBR 對用 戶 定 義 數 據的使用如下：

MTP_USERDATA								Description
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
x	x	x	x	x	1	0	1	IBR 使用 MTP 密鑰對開機程式碼執行 AES 解密之後，將解密後的程式碼載入到內存執行。
x	x	x	x	x	1	1	0	IBR 使用 MTP 密鑰對開機程式碼執行 SHA 比對之後，如果比對成功，將開機程式碼載入到內存執行。
x	x	x	x	x	0	x	x	IBR 不使用 MTP 密鑰。

X : 可為 0 或 1

如果用 戶 定 義 數 據 Bit 2 為 1，IBR 便會認定 MTP 密鑰是要用來做開機程式碼解密或比對使用。因此，如果用 戶 想要利用 MTP 密鑰來記錄 AES 密鑰或 SHA 運算值，將用 戶 定 義 數 據 Bit 2 寫成 0，就不會影響到 IBR 正常開機動作，此時，除了 Bit 2 之外的其他 Bit，用 戶 都可以自訂定義。

19.5.4 MTP 使能

在進行 MTP 密鑰的燒錄之前，必須先執行 MTP 使能程序。欲使用 MTP 密鑰進行 AES 加密/解密，或使用 MTP 密鑰與 SHA 運算比對之前，也必須先執行 MTP 使能程序。MTP 使能的詳細程序如下：

1. 將 MTPC (CLK_PCLKEN1[26]) 位設置為 1，啟用 MTP clock。
2. 對 MTP_REGLCTL 寄存器連續寫入 0x59，0x16，0x88 三個值，然後確認 REGLCTL (MTP_REGLCTL[0]) 為 1，表示已可以寫入 MTP 寄存器。
3. 對 KEYEN (MTP_KEYEN[0]) 寫 1。
4. 檢查 MTPEN (MTP_STATUS[0]) 位，直到它被設置為 1。

5. 如果 NONPRG (MTP_STATUS[2]) 為 1，表示此芯片尚未被寫入過 MTP 密鑰，現在 MTP 已使能成功，可以進行 MTP 密鑰燒錄動作。請注意，由於尚未寫入過 MTP 密鑰，如果用 MTP 來執行 AES 解密或 SHA 比對，將會出現不可預期錯誤。
6. 如果 KEYVALID (MTP_STATUS[2]) 為 1，表示 MTP 已經使能成功，可以用 MTP 來執行 AES 解密或 SHA 比對。如果 LOCKED (MTP_STATUS[3]) 為 0，那麼用戶此時也可以執行 MTP 密鑰燒錄或上鎖程序。如果 LOCKED (MTP_STATUS[3]) 為 1，則表示此芯片 MTP 密鑰已上鎖，不允許再執行 MTP 密鑰燒錄或上鎖程序。

19.5.5 MTP 密鑰燒錄

在進行 MTP 密鑰的燒錄之前，必須先執行 MTP 使能程序，如19.5.4 所述。MTP 使能成功之後，檢查 MTP_STATUS 寄存器，如果出現下列情況，則無法燒錄 MTP 密鑰：

- LOCKED (MTP_STATUS[3]) 為 1：表示此芯片 MTP 密鑰已上鎖，不允許再執行 MTP 密鑰燒錄。
- PRGCNT (MTP_STATUS[19:16]) 等於 15：表示此芯片 MTP 密鑰燒錄次數已到達 15 次上限，無法再進行燒錄。

除上述兩種情況之外，在 MTP 使能之後，用戶可以對 MTP 執行密鑰燒錄程序，燒錄步驟如下：

1. 對 MODE (MTP_CTL[1:0]) 寫入 2。
2. 根據 PCLK 當時的頻率，對 MTP_PCYCLE 寫入 PCLK clock 數，總時間必須大於 330us。例如，當 PCLK 為 75 MHz 的時候，每個 clock 時間為 13.333 ns， $330000/13.333 = 24750$ 。所以必須對 MTP_PCYCLE 寫入大於 24750 的數值，才能滿足 MTP 燒錄時間的要求。
3. 將 256 bits 的密鑰寫入到 MTP_KEY0 ~ MTP_KEY7。需特別注意的是，如果此 MTP 密鑰將用於 AES 密鑰使用，那麼根據使用時的開機模式 (Boot from NAND, SPI, 或 eMMC)，必須將 AES 密鑰做字組位置轉換 (參見 19.5.7)。
4. 將用戶定義數據寫入 MTP_USERDATA。
5. 對 PSTART (MTP_PSTART[0]) 寫入 1，然後檢查等待 PSTART 被 MTP 控制器清除為 0。
6. 如果 PRGFAIL (MTP_STATUS[4]) 為 0，則表示燒錄成功。反之，則表示燒錄失敗。

19.5.6 MTP 上鎖

在燒錄了 MTP 密鑰之後，如果已確定不再更動密鑰，用戶可以選擇對 MTP 上鎖。MTP 上鎖是為了安全目的，當 MTP 被上鎖之後，MTP 控制器便不允許再進行 MTP 燒錄操作，這可以防止 MTP 密鑰被覆寫。由於 MTP 控制器可以允許 15 次的寫入，如果用戶燒錄 MTP 密鑰之後未上鎖，那麼破解者便可能藉由燒錄新的 MTP 密鑰來覆蓋掉原先的密鑰，因此，為了安全起見，最好是對 MTP 進行上鎖。

為了安全理由，MTP 是不提供解鎖功能的，一旦 MTP 上鎖成功之後，該芯片之 MTP 便無法透

過任何方式修改，重新下電上電也不會重置上鎖狀態。

要進行 MTP 上鎖操作之前，同樣必須先執行 MTP 使能程序，如19.5.4 所述。MTP 上鎖的操作程序如下：

1. 對 MODE (MTP_CTL[1:0]) 寫入 3。
2. 根據 PCLK 當時的頻率，對 MTP_PCYCLE 寫入 PCLK clock 數，總時間必須大於 330us。
例如，當 PCLK 為 75 MHz 的時候，每個 clock 時間為 13.333 ns， $330000/13.333 = 24750$ 。
所以必須對 MTP_PCYCLE 寫入大於 24750 的數值，才能滿足 MTP 上鎖時間的要求。
3. 對 PSTART (MTP_PSTART[0]) 寫入 1，然後檢查等待 PSTART 被 MTP 控制器清除為 0。
4. 檢查 LOCKED (MTP_STATUS[3])，KEYVALID (MTP_STATUS[1])，MTPEN (MTP_STATUS[0])，如果此三個位均被設置為 1，則表示 MTP 已成功上鎖。

19.5.7 MTP 密鑰用於 AES 加密/解密

MTP 密鑰可以被 AES 加速器硬件讀取，並做為 AES 密鑰使用，在使用之前，用戶必須先執行 MTP 密鑰使能程序，才能夠使用 MTP 密鑰。

如果要以 MTP 密鑰來做為 AES 加密/解密時的密鑰使用，只需要在進行 AES 加密/解密程序的時候，將 EXTKEY (CRPT_AES_CTL[4]) 設置為 1，AES 加速器在取用密鑰的時候，便會從 MTP 密鑰擷取，而不是從 CRPT_AESn_KEYx 寄存器讀取密鑰。

MTP 密鑰唯一的讀取途徑，只能被 AES 或 SHA/HMAC 加速器硬件直接讀取，使用者無法經由任何的方法取得密鑰，這是為了保護密鑰安全。

必須特別注意的是，AES 加速器擷取 MTP 密鑰的方式並不一定是按順序擷取的，而是會根據執行 AES 解密程序當時所處的開機模式有關。這個設定限制了開機程式只從指定的來源開機，否則便會無法解密出正確的原始碼。MTP 密鑰基於不同開機模式，與 AES 密鑰的對映關係，如下表所列：

AES 密鑰	從 NAND 開機	從 SPI 開機	從 eMMC 開機	從 USB 開機
KEY 0	MTP_KEY 4	MTP_KEY 6	MTP_KEY 2	MTP_KEY 0
KEY 1	MTP_KEY 5	MTP_KEY 7	MTP_KEY 3	MTP_KEY 1
KEY 2	MTP_KEY 6	MTP_KEY 0	MTP_KEY 4	MTP_KEY 2
KEY 3	MTP_KEY 7	MTP_KEY 1	MTP_KEY 5	MTP_KEY 3
KEY 4	MTP_KEY 0	MTP_KEY 2	MTP_KEY 6	MTP_KEY 4
KEY 5	MTP_KEY 1	MTP_KEY 3	MTP_KEY 7	MTP_KEY 5
KEY 6	MTP_KEY 2	MTP_KEY 4	MTP_KEY 0	MTP_KEY 6
KEY 7	MTP_KEY 3	MTP_KEY 5	MTP_KEY 1	MTP_KEY 7

19.5.8 MTP 密鑰用於 SHA/HMAC 比對

MTP 密鑰可以被 SHA/HMAC 加速器硬件讀取，並做為 SHA/HMAC 運算結果比對之用，在使用之前，用戶必須先執行 MTP 密鑰使能程序，才能夠使用 MTP 密鑰。

如果要以 MTP 密鑰來做為 SHA/HMAC 運算結果比對，只需要在進行 SHA/HMAC 運算程序的時候，將 CMPEN (CRPT_HMAC_CTL[15]) 設置為 1，那麼 SHA/HMAC 加速器在執行完運算之後，便會自動將運算結果與 MTP 密鑰做比對，如果運算結果與 MTP 密鑰完全相符，SHA/HMAC 加速器便會將 CMPSTS (CRTP_HMAC_STS[15]) 設置為 1，如果不相符，便會將 CMPSTS 設置為 0。

與做為 AES 解密用途不同的是，當 MTP 密鑰用於 SHA/HMAC 運算結果比對時，SHA/HMAC 加速器在擷取 MTP 密鑰的時候，並不會根據開機模式做順序轉換，而是直接依序比對。

20 脈寬調製 (PWM)

20.1 概述

NUC970 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.

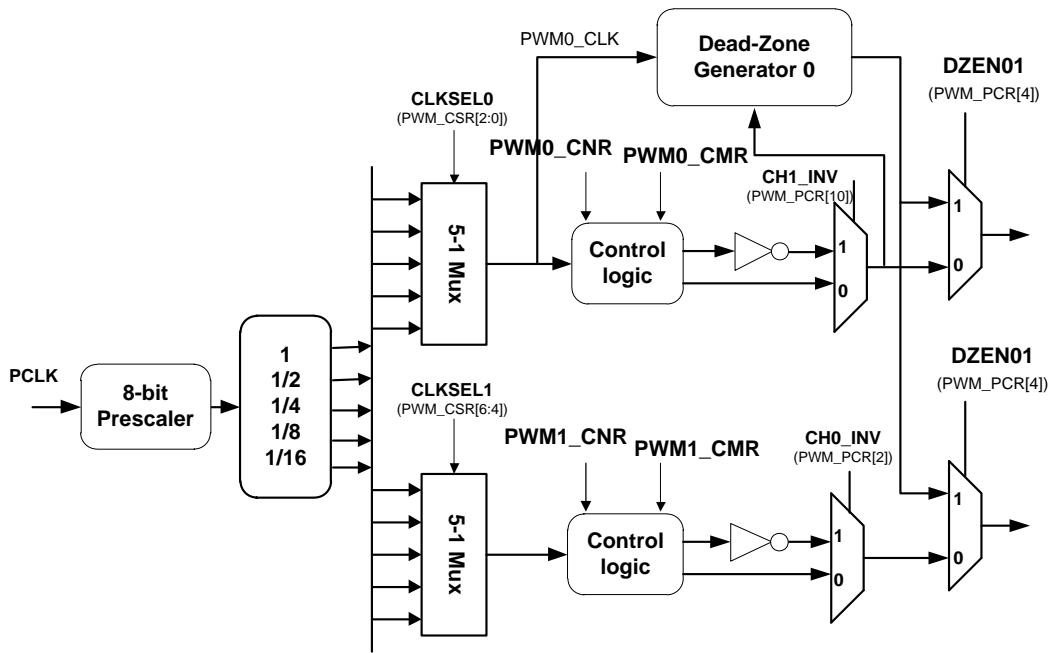
當 16 位下數型計數器達到 0 時, 中斷要求產生. 如果 PWM 輸出被設置為連續模式, 當下數計數器計數到 0 時, 下數計數器會重複自動重新裝載 PWMx_CNR 寄存器中 CNR 的值, 並開始減計數. 如果 PWM 輸出被設置為單次模式, 當下數計數器計數到 0 時, 停止計數, 並產生一個插斷要求.

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.

20.2 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

20.3 方塊圖



20.4 寄存器

Register	Offset	R/W	Description	Reset Value
PWM Base Address:				
PWM_BA = 0xB800_7000				
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_CMNR	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_CMNR	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_CMNR	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_CMRR	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

20.5 功能描述

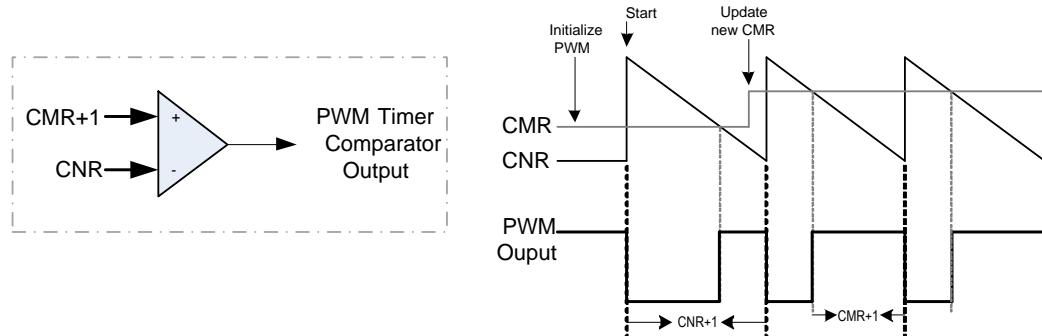
20.5.1 PWM 計時器操作

PWM 週期和占空比控制由 `PWMx_CNR` 寄存器和 `PWMx_CMRR` 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：

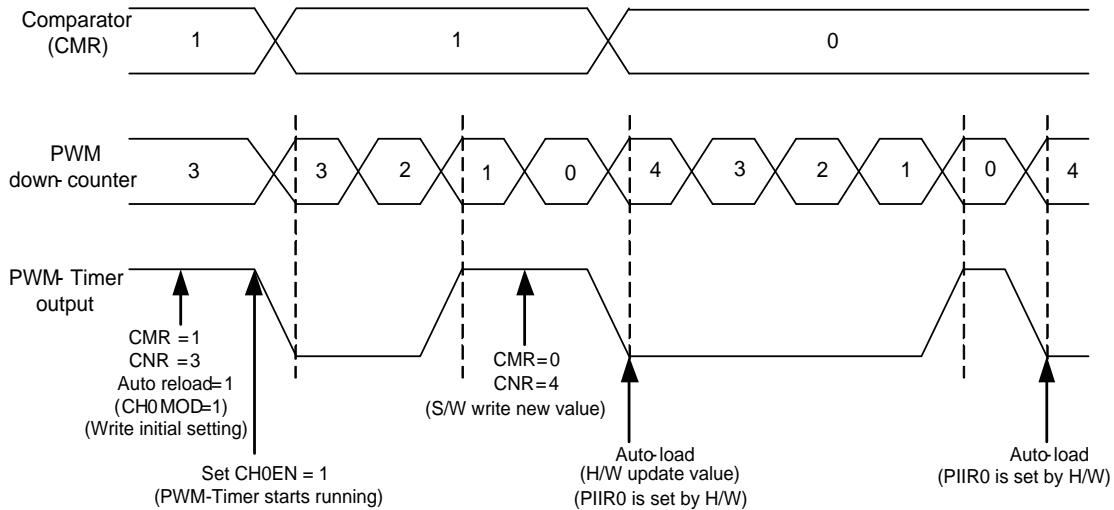
$$\text{PWM 頻率} = \text{PWM_CLK} / ((\text{prescale} + 1) * (\text{clock divider})) / (\text{CNR} + 1)$$

$$\text{PWM 占空比} = (\text{CMR} + 1) / (\text{CNR} + 1)$$

由輸出波形來看，當 $\text{CMR} \geq \text{CNR}$: PWM 輸出總是為高。當 $\text{CMR} < \text{CNR}$: PWM 低脈寬 = $(\text{CNR} - \text{CMR})$ 個 PWM 時鐘，PWM 高脈寬 = $(\text{CMR} + 1)$ 個 PWM 時鐘。而當 $\text{CMR} = 0$: PWM 低脈寬 = CNR 個 PWM 時鐘；PWM 高脈寬 = 1 個 PWM 時鐘。

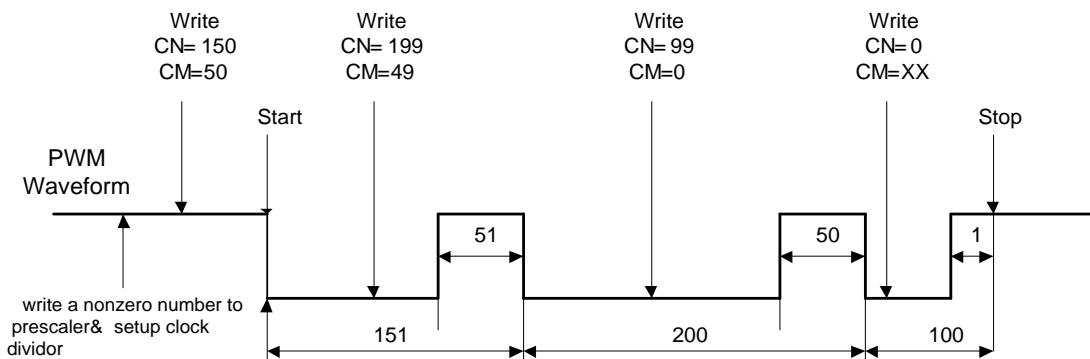


下圖顯示了 PWM 計時器的比較器運作原理。當下數計數器的值與比較寄存器的值匹配時，計數器控制邏輯會改變 PWM 輸出電平。

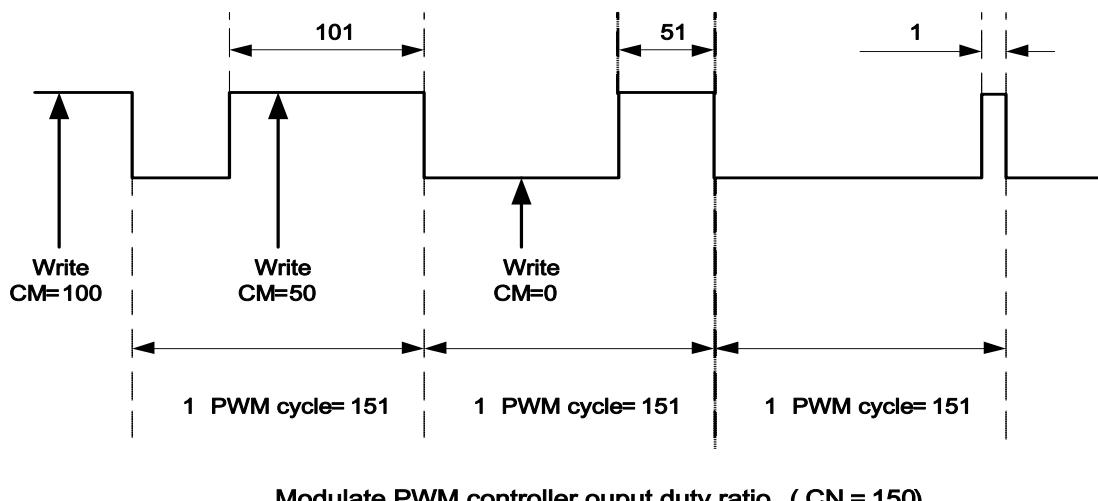


20.5.2 PWM 雙緩存功能

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_CMRR 的 [15:0] 位。



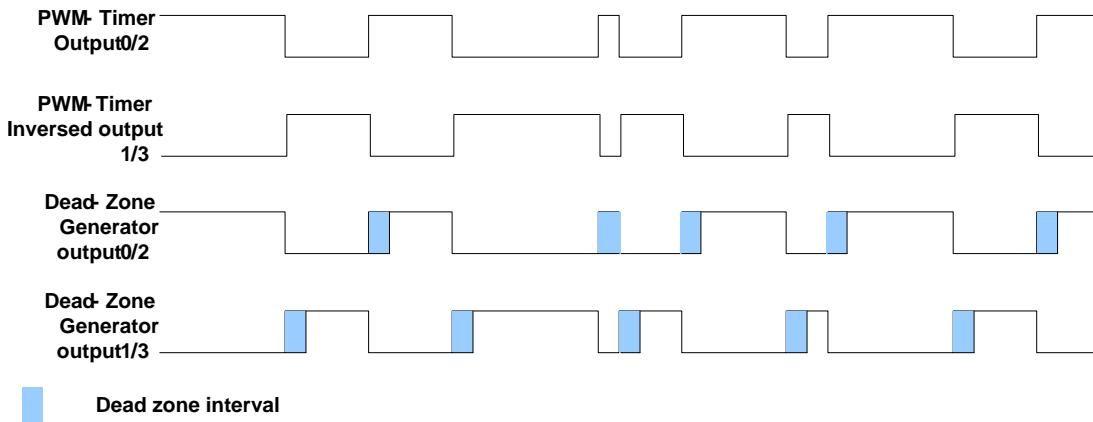
20.5.3 連續以及單次操作

PWM 控制寄存器 (PWM_PCR) 中 CHxMOD 位定義 PWM_x ($x = 0\sim3$) 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.

20.5.4 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.



20.5.5 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_CMRR 寄存器的和 PWMx_CNRR 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->CNRR = (1000 - 1);
```

20.5.6 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.

21 實時時鐘 (RTC)

21.1 概述

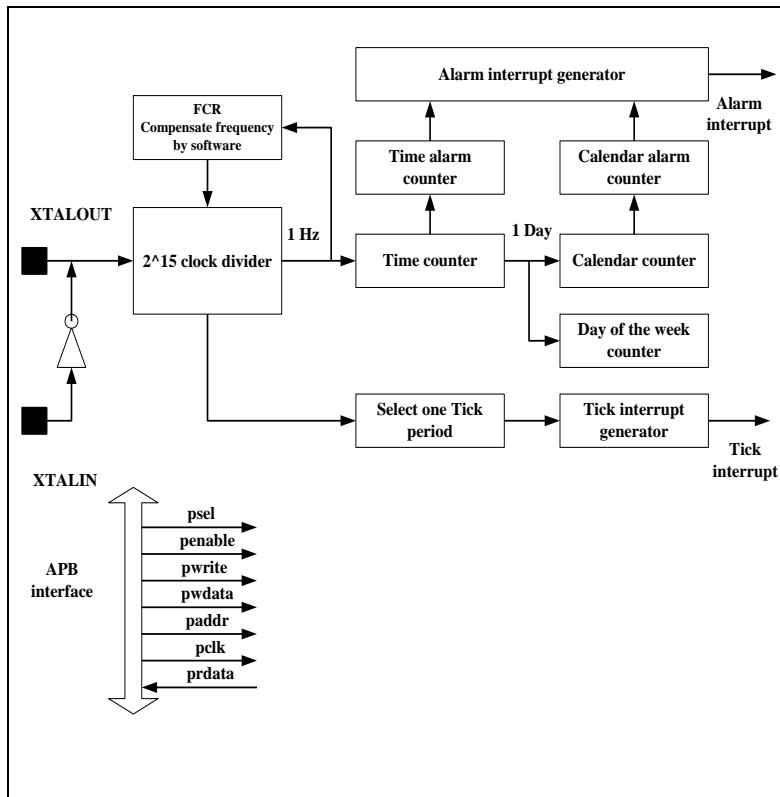
實時時鐘 (RTC) 模塊當系統電源關閉可以由獨立的電源進行供電。該 RTC 模塊採用外部晶振產生 32.768 kHz 時鐘。RTC 可以將數據用 BCD 碼傳輸到 CPU。這些數據包括時間訊息(秒，分，時)和日曆訊息(日，月，年)。此外，為了達到更好的頻率精度，RTC 計數器可以通過軟件進行調整。

21.2 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 小時模式可選擇
- 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

21.3 方塊圖

實時時鐘(RTC)的模塊圖如下所示:



21.4 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 = 0xB800_4000				
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_INEN	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

21.5 功能描述

21.5.1 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。

21.5.2 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

21.5.3 12/24 小時格式顯示切換

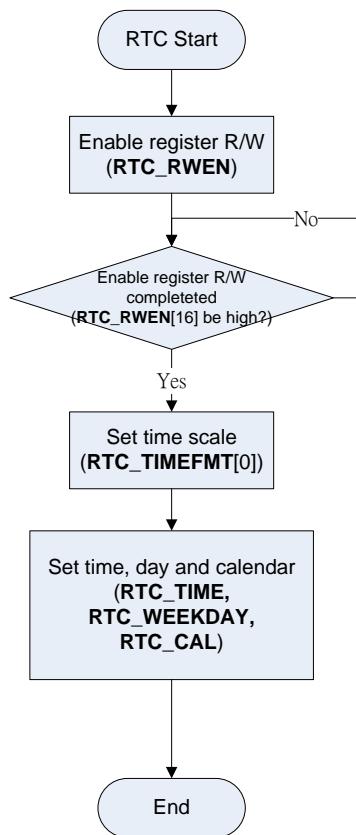
設置RTC_TIMEFMT寄存器的[0]位為0/1，用於切換顯示格式為12/24小時制。

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)

21.5.4 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

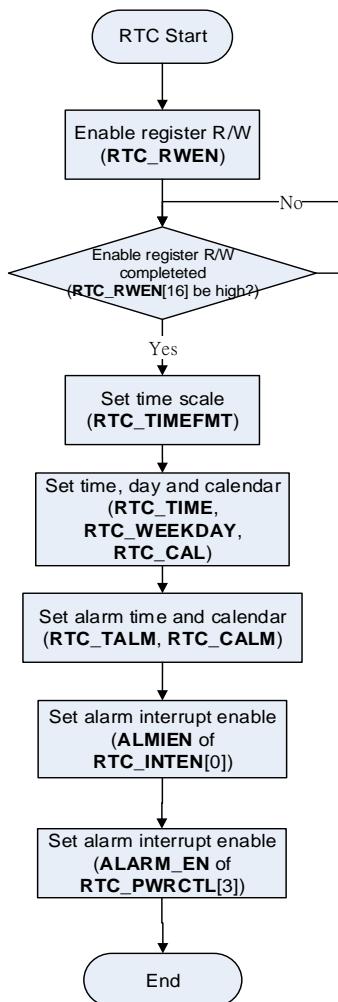


21.5.5 絕對時間鬧鐘設定

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

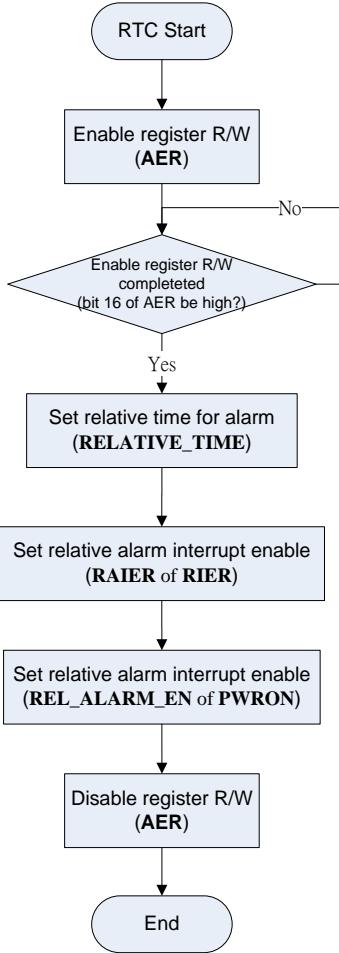


21.5.6 相對時間鬧鐘設定

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：當鬧鐘中斷產生時，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



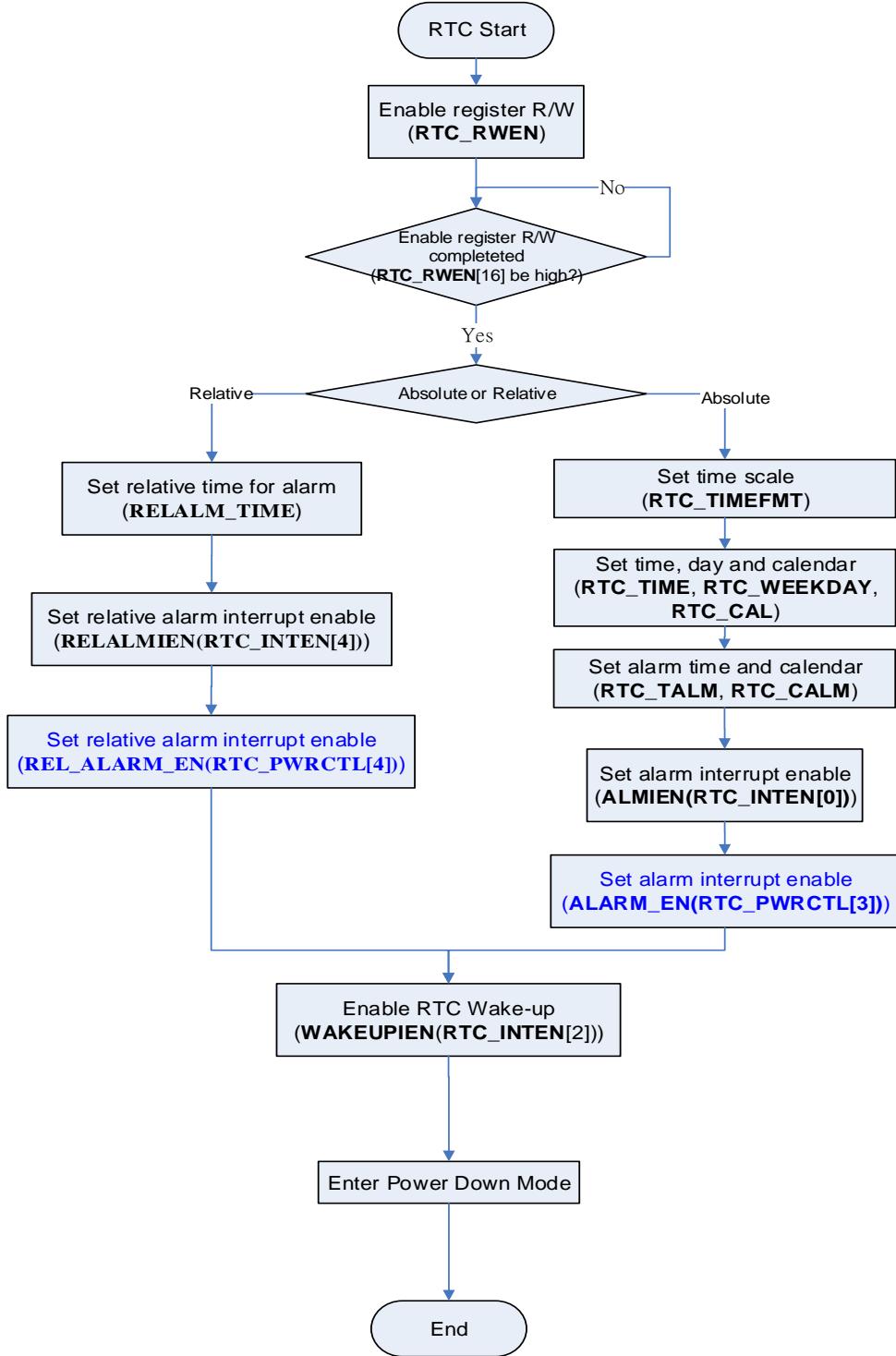
21.5.7 喚醒功能

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.



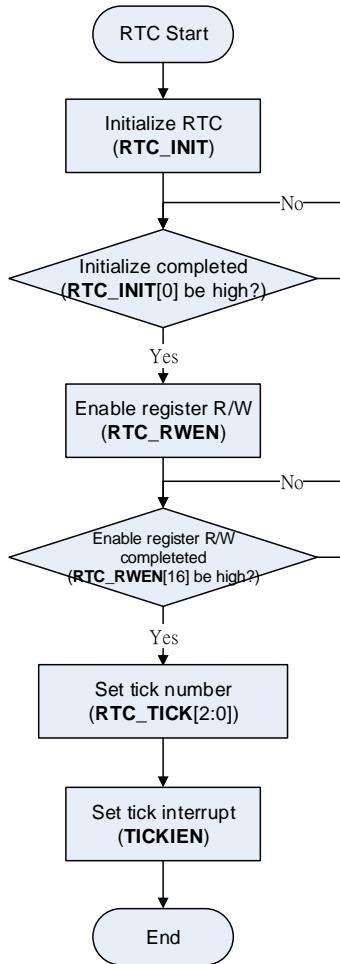
21.5.8 時鐘節拍

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



21.5.9 系統電源控制流程

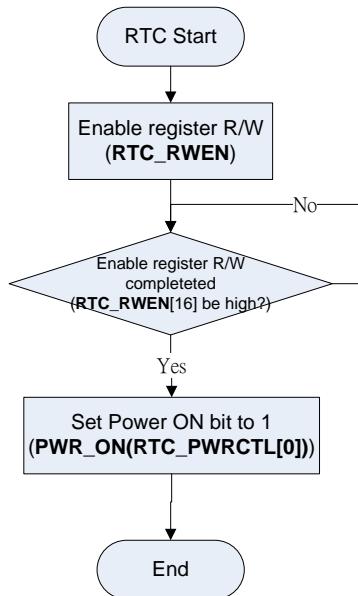
21.5.9.1 上電控制流程

壓按電源鍵(PWR Key),使電源控制訊號(PWCE)變成高電位,如果PWR_ON位(PWROM[0]),被設成1,當電源鍵被放開時,PWCE將會被保持在高電位.如果PWR_ON位(PWRON[0])沒有被設成1,當電源鍵被放開時,PWCE將會變成低電位.以下為控制流程:

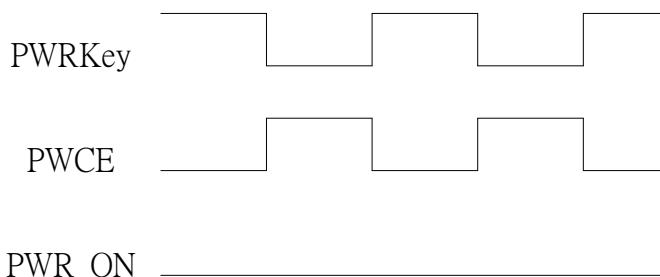
1. 壓按 PWR Key 打開系統電源

2. 當 RTC 上電時,用戶必須寫 0xA5EB1357 到繼存器 RTC_INIT, 初始化 RTC
3. 當 RTC_INIT[0]由變成 1 時,表示 RTC 初始化完成
4. 將繼存器 RTC_RWEN 寫入 0xA965,使能 RTC 繼存器訪問(讀/寫)功能
5. 確認 RWENF(RTC_RWEN[16])變成 1,確認 RTC 繼存器訪問功能已被使能.
6. RWENF(RTC_RWEN[16])會在 1024 個 RTC clock 之後自動被清除成 0
7. 設定 PWR_ON(RTC_PWRCTL[0])為 1,打開系統電源

電源鍵觸發模式有兩種, 可透過 EDGE_TRIG(RTC_PWRCTL[5]) 設置。設為1是邊緣觸發, 經由壓按電源鍵一段足夠長的時間,然後放開電源鍵之後才上電. 設為 0是準位觸發. 經由壓按電源鍵一段足夠長的時間後上電.



當用戶壓按電源鍵使得電源控制訊號(PWCE)變成高電位時,如果沒有把PWR_ON設成1,這時只要用戶放開電源鍵,PWCE就會馬上變為低電位



21.5.9.2 強制下電流程

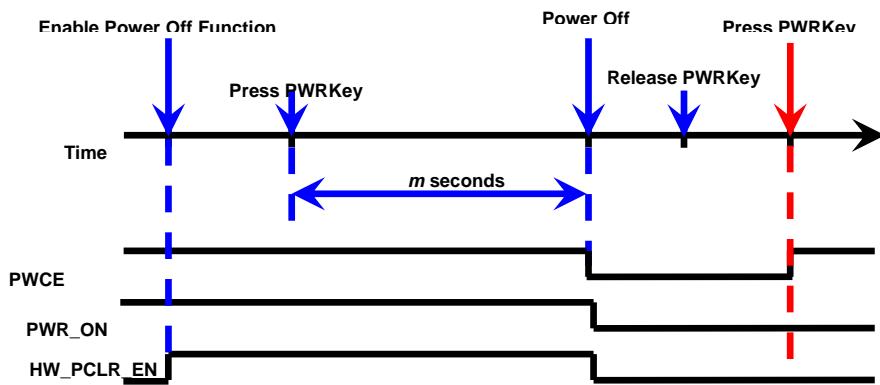
當PWR_ON位為1時,電源鍵如果有任何的壓按動作,系統將會產生中斷訊號(PWRSWINT),此時用戶可以自行決定是否要將PWR_ON位清除成0.如果將PWR_ON位清除成0,PWCE將會變為低電位,在這種情況下,系統電源就會被關閉.如果用戶決定讓PWR_ON位繼續維持為1,這時PWCE將會繼續被保持在高電位,也就是電源不會被關閉.但是要注意的是,如果PWR_ON位,被用戶清除成0時,即使電源鍵沒有任何壓按的動作,PWCE仍然會立刻被變成低電位,此時系統電源也將會被關閉.

當使用硬體自動斷電程序時,用戶只要使能HW_PCLR_EN位,並且壓按電源鍵數秒鐘,便可以將系統電源斷電.壓按電源鍵的時間,可以利用設定PWROFF_TIME(RTC_PWRCTL[23:16])來決定

PWROFF_TIME Setting	Pressed time to power off	PWROFF_TIME Setting	Pressed time to power off
0	3~4 second	8	11~12 seconds
1	4~5 second	9	12~13 seconds
2	5~6 seconds	10	13~14 seconds
3	6~7 seconds	11	14~15 seconds
4	7~8 seconds	12	15~16 seconds
5	8~9 seconds	13	16~17 seconds
6	9~10 seconds	14	17~18 seconds
7	10~11 seconds	15	18~19 seconds

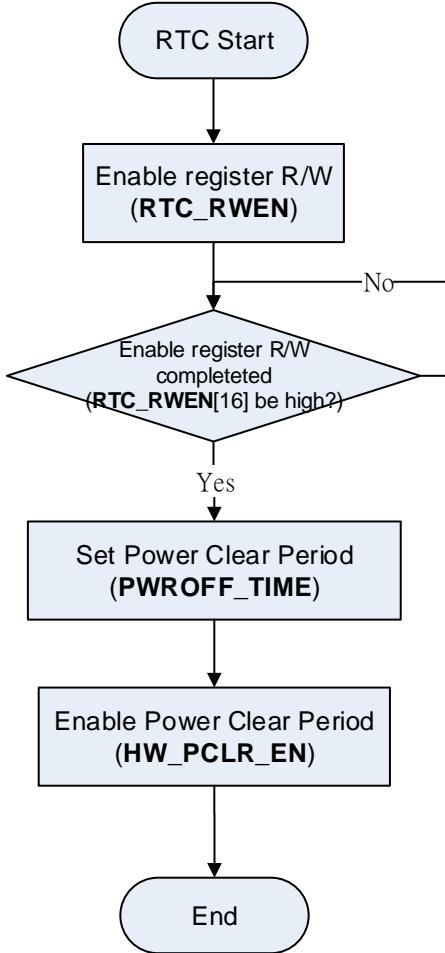
當使用RTC硬體自動斷電程序時,如果用戶壓按電源鍵要關閉系統電源時,此時硬體會去計數用戶持續壓按電源鍵的秒數,當壓按的秒數等於PWROFF_TIME的設定時,硬體會自動將PWCE變成0並且將PWR_ON也清除成0(但HW_PCLR_EN不會被自動清除成0).在系統斷電之後,當電源鍵被壓按時,用戶可以決定是否要將PWR_ON設成1,用來將系統電源再度打開.

下圖是硬體自動斷電時序圖



硬體下電控制流程如下：

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. 設定 PWROFF_TIME(RTC_PWRCTL[15:12])決定壓按電源鍵多少秒後,系統會下電
5. 使能硬體下電功能(HW_PCLR_EN(RTC_PWRCTL[2]))



21.5.9.3 軟體系統下電

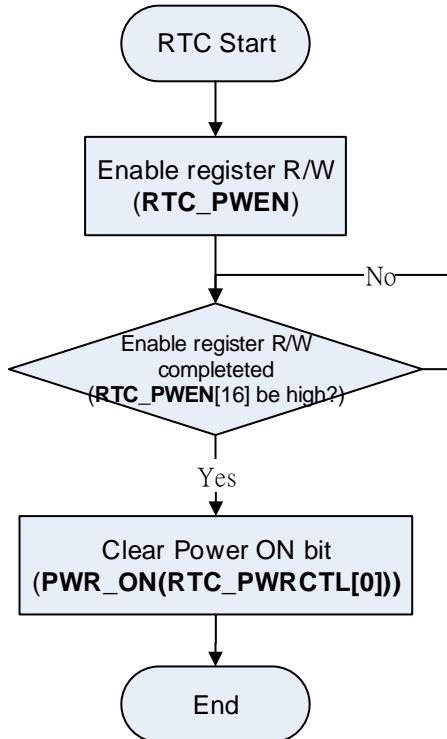
RTC也支持軟體斷電控制程序.用戶可以壓按電源鍵數秒鐘用來關閉系統電源,而壓按電源鍵的秒數可以由用戶自行決定.當PWR_ON被用戶清除成0時,PWCE將會在116us後被設成低電位,而SW_PCLR會在電源鍵被放開時被清除成0.

下圖是軟體控制系統斷電時序圖

以下是程式控制流程:

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. 設定 PWR_ON(RTC_PWRCTL[0])位為 0



下表是全部系統控制流程的說明:

Input			Output	Note
X1	X2	X3	Y	
PWRKey	PWR_ON	RST_	PWCE	
1	0	0	0	RTC powered only (Default state)
0	0	X	1	Press key, Power On
0	1	1	1	keep key & S/W Set X2, Power On

1	1	1	1	Left key, Power keep On
0	1	1	1	Press key, get INT, intend to power Off
1	0	1	0	Left key & S/W clean X2, power Off or S/W clean X2 , don't need press key, power off
X	1	0	1	RST_ active, still keep power whenX2=1

PWCE is open drain output
 X1, internal pull-up
 X2, it is R/W able
 There is Interrupt from key be pressed

21.5.10 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，用戶可以利用一個頻率計數器來測量腳位PH.4和PI.3的輸出，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

22 智能卡接口 (SC)

22.1 概述

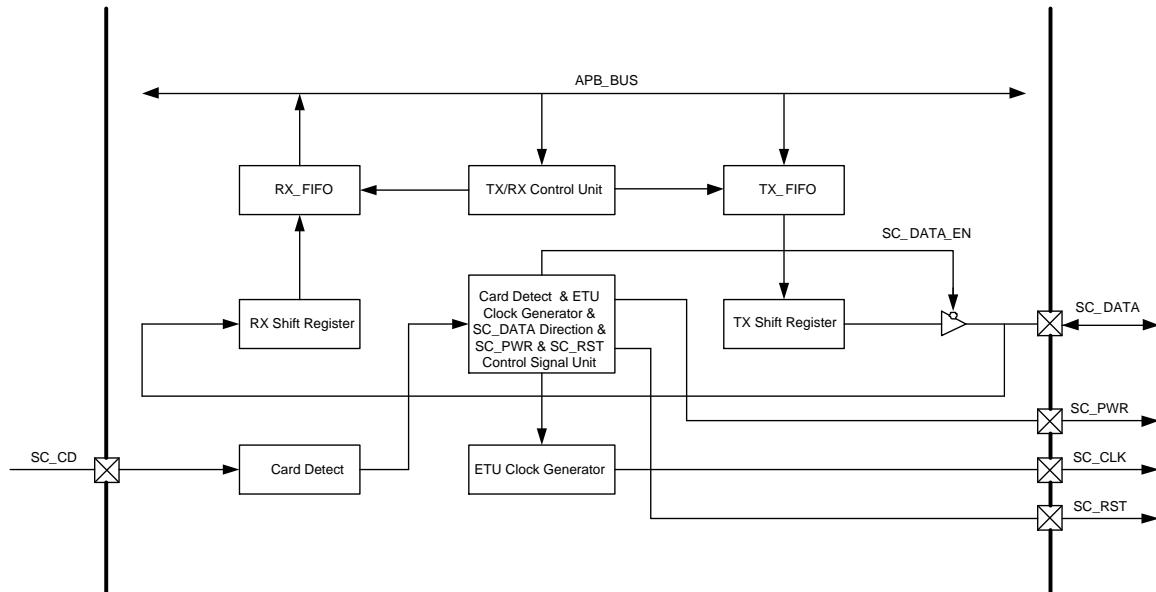
智能卡介面控制器 (SC controller) 是基於 ISO/IEC 7816-3 標準並完全相容 PC/SC 規格. 它也提供卡插入/移除的狀態

22.2 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

22.3 方塊圖



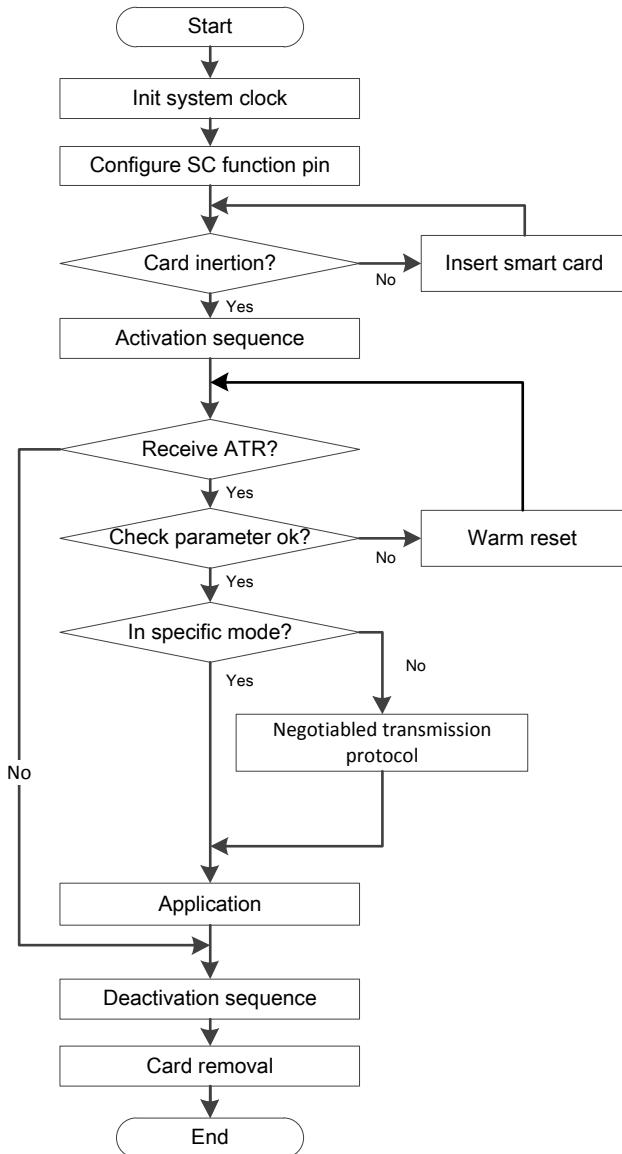
22.4 寄存器

Register	Offset	R/W	Description	Reset Value
SC Base Address:				
SC0_BA = 0xB800_5000				
SC1_BA = 0xB800_5400				
SC_DAT <i>x = 0,1</i>	SCx_BA+0x00	R/W	SC Receiving/Transmit Holding Buffer Register	0xFFFF_FFFF
SC_CTL <i>x = 0,1</i>	SCx_BA+0x04	R/W	SC Control Register	0x0000_0000
SC_ALTCTL <i>x = 0,1</i>	SCx_BA+0x08	R/W	SC Alternate Control Register	0x0000_0000
SC_EGT <i>x = 0,1</i>	SCx_BA+0x0C	R/W	SC Extend Guard Time Register	0x0000_0000
SC_RXTOOUT <i>x = 0,1</i>	SCx_BA+0x10	R/W	SC Receive Buffer Time-out Register	0x0000_0000
SC_ETUCTL <i>x = 0,1</i>	SCx_BA+0x14	R/W	SC ETU Control Register	0x0000_0173
SC_INTEN	SCx_BA+0x18	R/W	SC Interrupt Enable Control Register	0x0000_0000

x = 0,1				
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_TMRDAT0 x = 0,1	SCx_BA+0x38	R	SC Timer Current Data Register A	0x0000_07FF
SC_TMRDAT1_2 x = 0,1	SCx_BA+0x3C	R	SC Timer Current Data Register B	0x0000_7F7F

22.5 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:



22.5.1 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. 啓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.

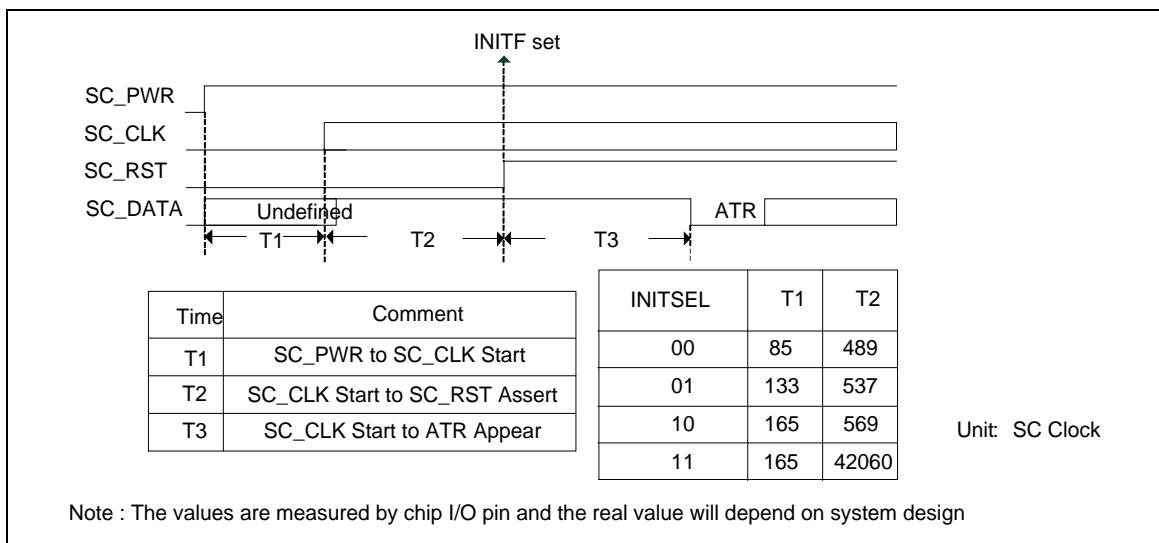


圖 22-1 SC 啟動序列

22.5.2 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.

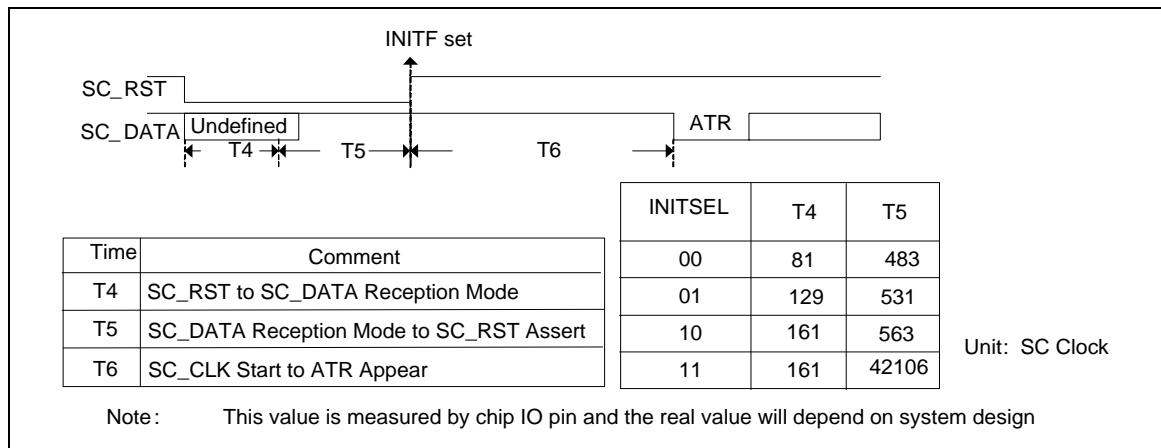


圖 22-2 SC 暖復位序列

22.5.3 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.

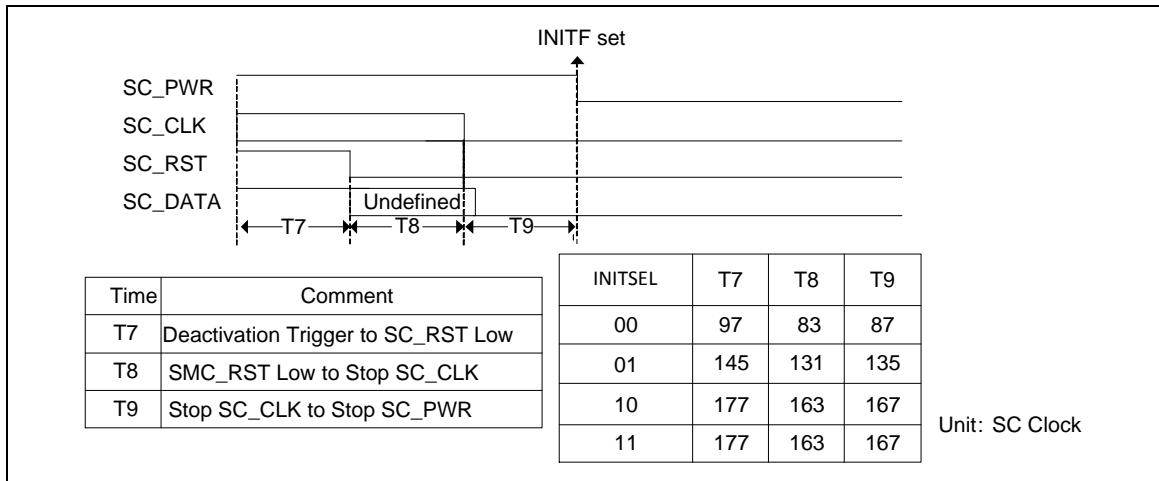


圖 22-3 SC 釋放序列

22.5.4 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.

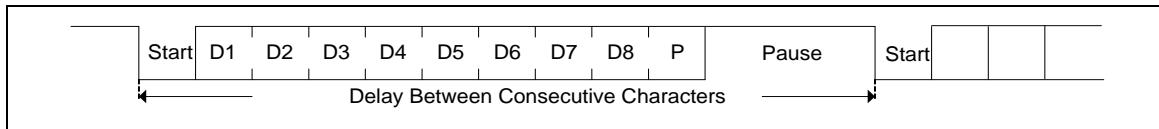


圖 22-4 SC 資料字元

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 AUTOSEN (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 AUTOSEN 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.

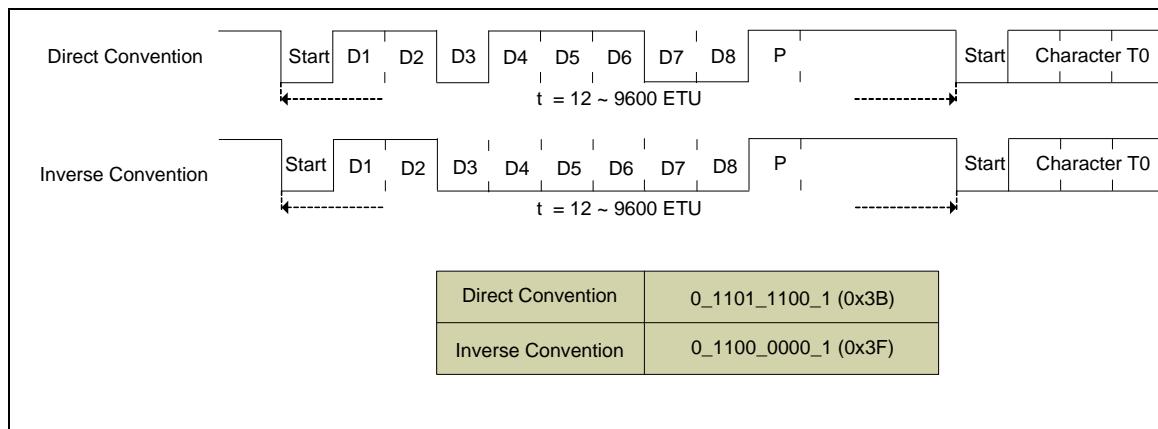


圖 22-5 初始化字元 TS

22.5.5 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_RXTOOUT 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。

22.5.6 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.

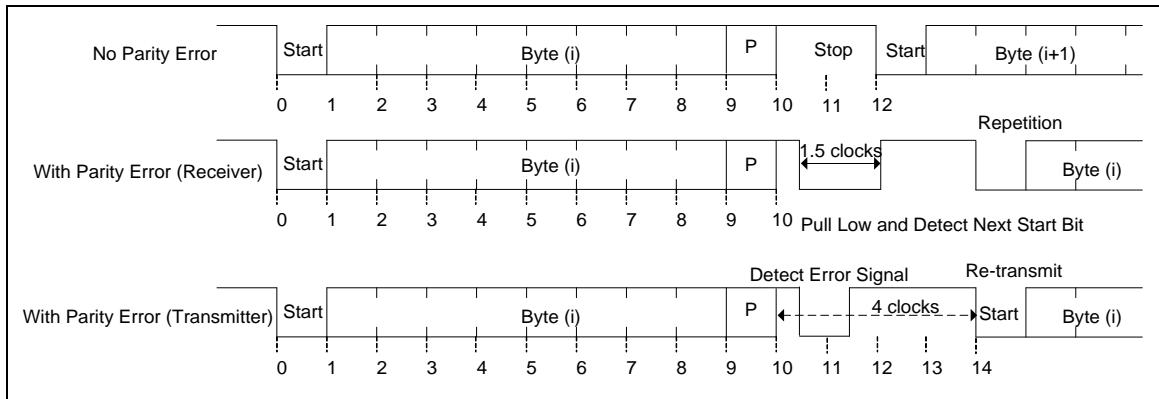


圖 22-6 SC 錯誤信號

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.

22.5.7 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_TMR CTLx[27: 4]) (X=0 ~2)	操作描述	
		向下計數器開始於 CNTENx 使能, 結束於計數器超時, 超時值為 CNT+1.
0000	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.
		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.
0001	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.
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.
		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.
		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.
1000	向上計數器開始於CNTENx 使能，結束於CNTENx 禁用. 計數值將被保存在SC_TMRDATx. 本模式下，硬體將不能產生任何中斷給 CPU. 實際計數值將是 CNT +1
	開始 在CNTENx 設為 1 後，開始計數，開始計數的值為0. 硬體將忽略 CNT 的值.
	結束 CNTENx 被設為 0 後，停止計數，並保存值到 SC_TMRDATx 寄存器.
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.

22.5.8 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.

22.5.9 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 TC₁ 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.

22.5.10 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 AUTOSEN (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0xFFFF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_RXTOOUT[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.

23 SD 控制器 (SDH)

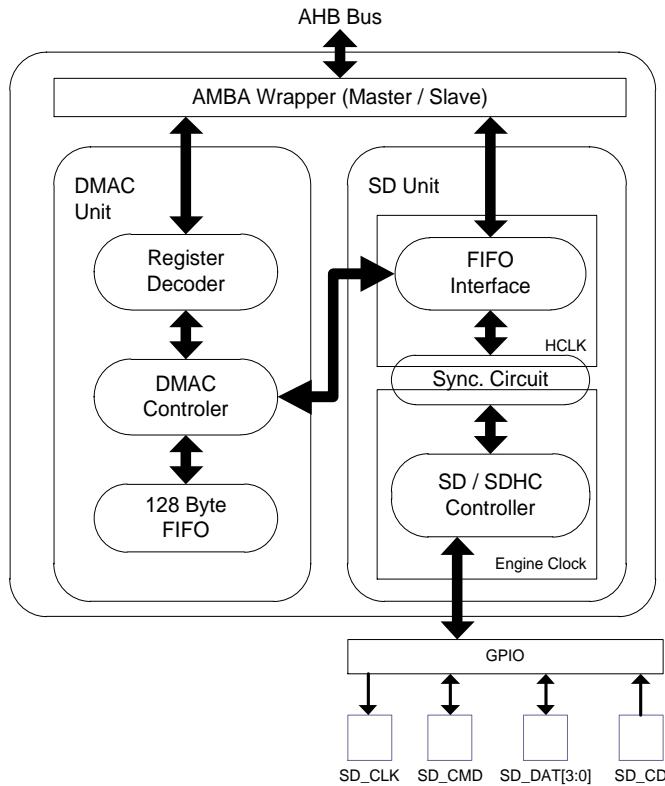
23.1 概述

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。

23.2 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。

23.3 方塊圖



23.4 寄存器

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

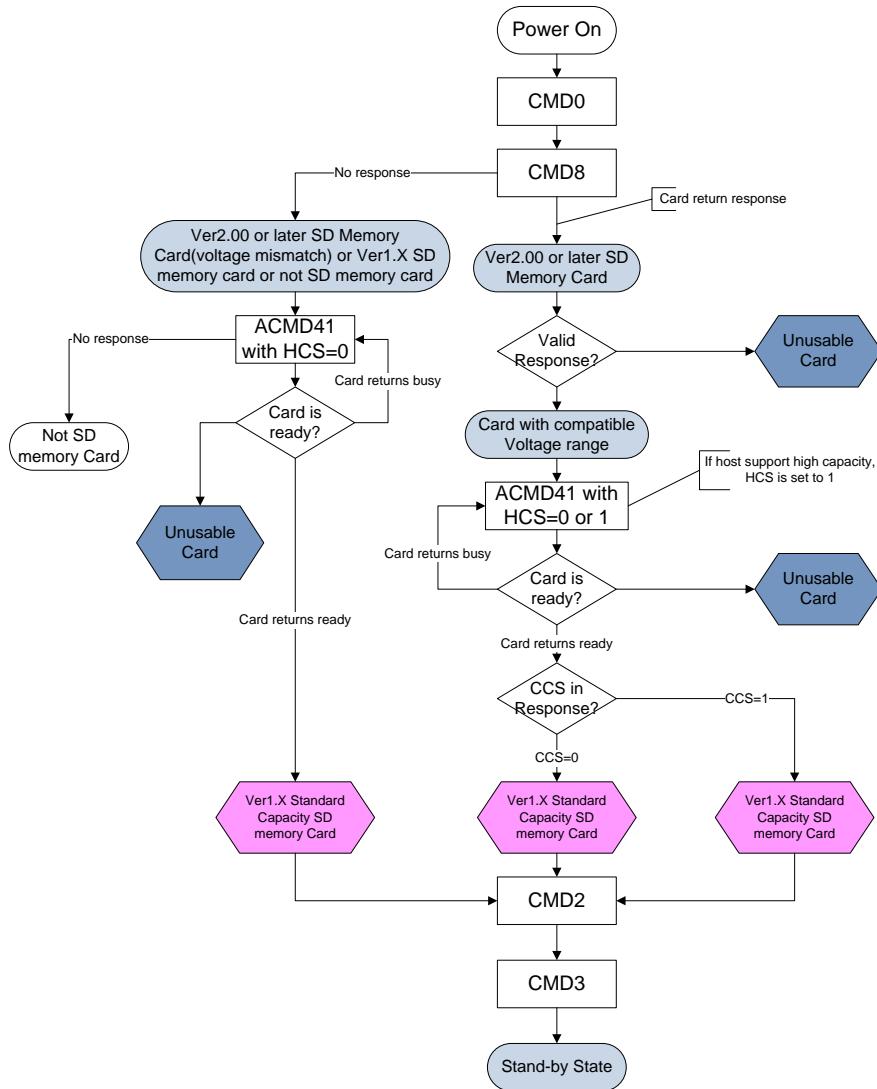
Register	Offset	R/W	Description	Reset Value
SDH_BA = 0xB000_C000				
SDH_FB_n n = 0,1...31	SDH_BA+0x000 + 0x4 * n	R/W	SD Host Embedded Buffer Word n n = 0,1...31	0x0000_0000
SDH_DMACTL	SDH_BA+0x400	R/W	SD Host DMA Control and Status Register	0x0000_0000
SDH_DMASA	SDH_BA+0x408	R/W	SD Host DMA Transfer 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

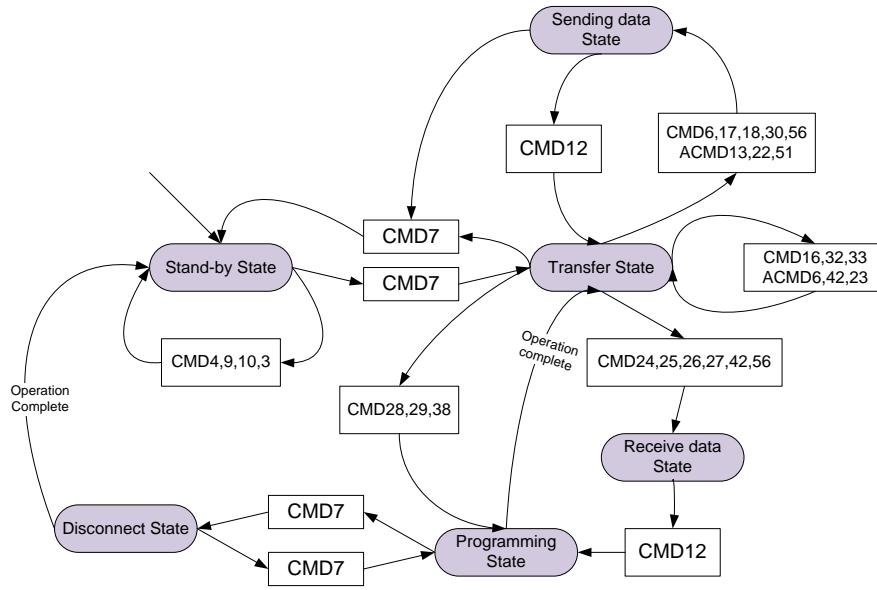
23.5 功能描述

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 Transfer Mode) :



23.5.1 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DMA，然後DMA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。

SD控制器提供2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GCTRLRST位。
5. 等待SDH_GCTL寄存器GCTRLRST位清除。
6. 端口0只有一組多功能控制（GPD0~7），SYS_GPD_MFPL要填入0x66666666。
7. 端口1有三組多功能控制：GPE2~9、GPH6~13和GPI5~10，12~13。
 - (1) GPE設定為SYS_GPE_MFPL要填入0x66666600，SYS_GPE_MFPH要填入0x66。
 - (2) GPH 設定為 SYS_GPH_MFPL 要填入 0x66000000，SYS_GPH_MFPH 要填入 0x666666。

- (3) GPI 設定為 SYS_GPI_MFPL 要填入 0x44400000，SYS_GPI_MFPH 要填入 0x00440444。
8. 清除SDH_ECTL寄存器PWROFF0位，始能電源控制，即供電給SD設備。
 9. 設置SDH_CTL寄存器SDPORT位來選擇SD端口0或端口1接入。
 10. 設置SDH_INTEN寄存器CDxSRC位來選擇SD卡檢測源（DAT3或GPIO）。
 11. 設定SDH初始化輸出頻率為300KHz，SDH接口為1位數據總線模式。
 12. 設置SDH_CTL寄存器CLK74_OE位。
 13. 等待SDH_CTL寄存器CLK74_OE位清除。
 14. 根據設備規則發送命令給SD設備。
 15. 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。

23.5.2 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。

23.5.3 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。

23.5.4 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。

23.5.5 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 SPI

24.1 概述

SPI 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。

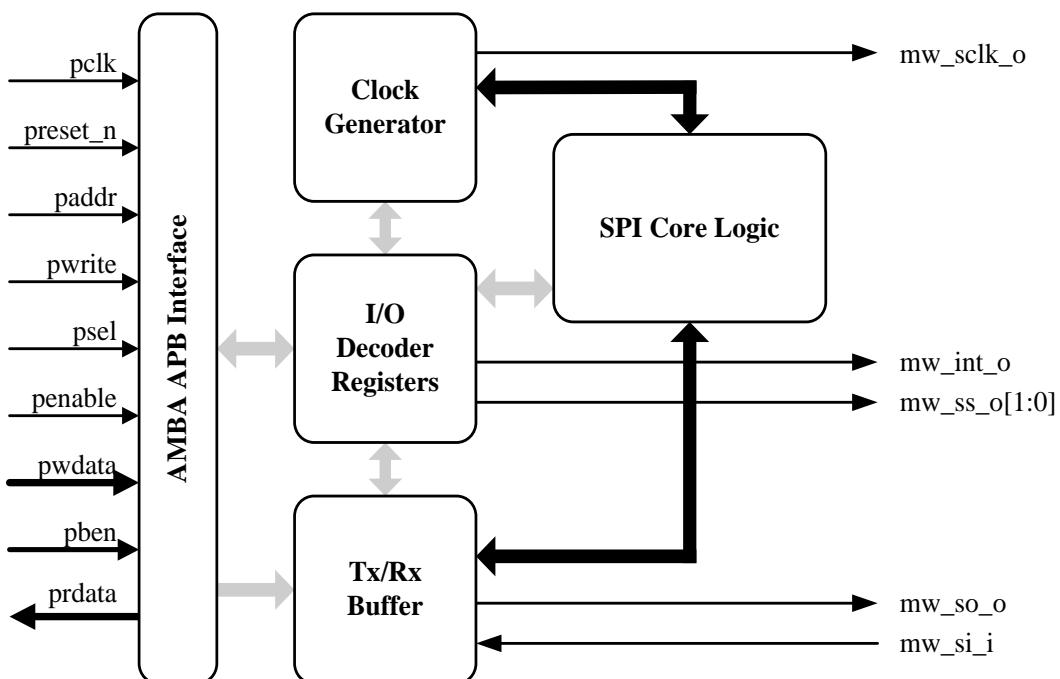
當從一個週邊設備接收資料時，SPI 執行串-並的轉換，而在資料向週邊設備發送時，執行並-串的轉換。

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。

24.2 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

24.3 方塊圖



24.4 寄存器

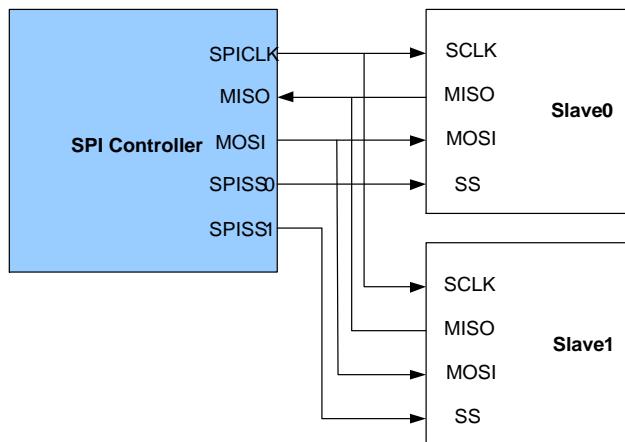
Register	Offset	R/W	Description	Reset Value
SPI_BA = 0xB800_6200				
SPI_BA = 0xB800_6300				
CNTRL	SPI_BA+0x00	R/W	Control and Status Register	0x0000_0004
DIVIDER	SPI_BA+0x04	R/W	Clock Divider Register	0x0000_0000
SSR	SPI_BA+0x08	R/W	Slave Select Register	0x0000_0000
Reserved	SPI_BA+0x0C	N/A	Reserved	N/A
Rx0	SPI_BA+0x10	R	Data Receive Register 0	0x0000_0000
Rx1	SPI_BA+0x14	R	Data Receive Register 1	0x0000_0000
Rx2	SPI_BA+0x18	R	Data Receive Register 2	0x0000_0000
Rx3	SPI_BA+0x1C	R	Data Receive Register 3	0x0000_0000
Tx0	SPI_BA+0x10	W	Data Transmit Register 0	0x0000_0000
Tx1	SPI_BA+0x14	W	Data Transmit Register 1	0x0000_0000
Tx2	SPI_BA+0x18	W	Data Transmit Register 2	0x0000_0000
Tx3	SPI_BA+0x1C	W	Data Transmit Register 3	0x0000_0000

24.5 功能描述

24.5.1 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SSR[0]或設定SSR[1]使能SS1輸出腳或同時輸出。

```
SSR |= 0x1;      //使能SS0輸出腳
SSR |= 0x2;      //使能SS1輸出腳
SSR |= 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 SS_LVL 位 (SSR[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。

24.5.2 自動從機選擇

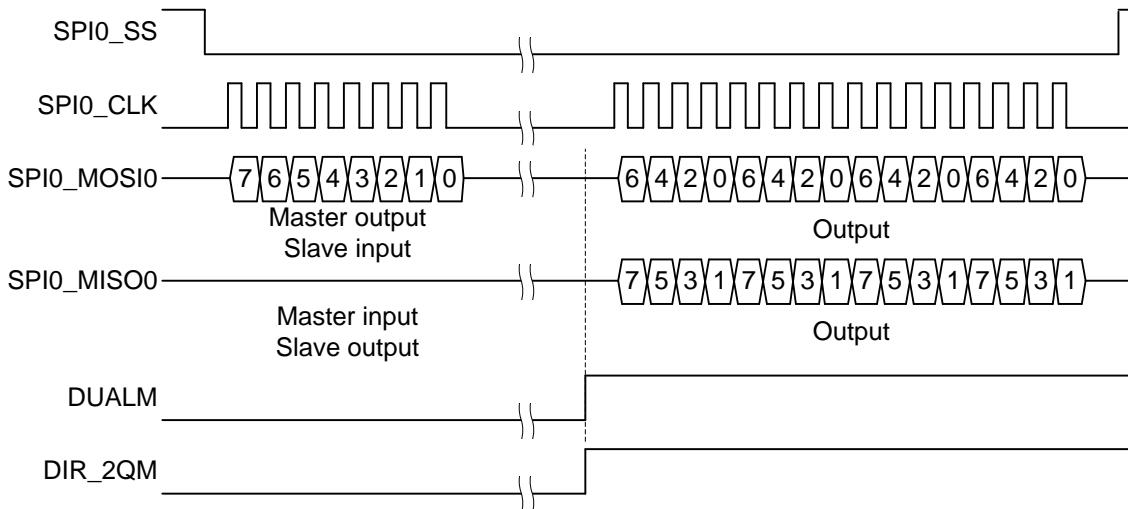
在主機模式下，如果置位元 AUTOSS (SSR[3])，將自動產生從機選擇信號，並根據 SSR[0] 位 (SSR[0]) 與 SSR[1] 位 (SSR[1]) 是否使能，將從機選擇信號輸出到 SSO 與 SS1 管腳上。這意味著當通過設置 GO_BUSY 位 (CNTRL[0]) 來開始資料傳輸時，在 SSR[1:0] 中使能的從機選擇信號將由 SPI 控制器自動設置為有效狀態，在資料傳輸結束後自動被設為無效狀態。

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

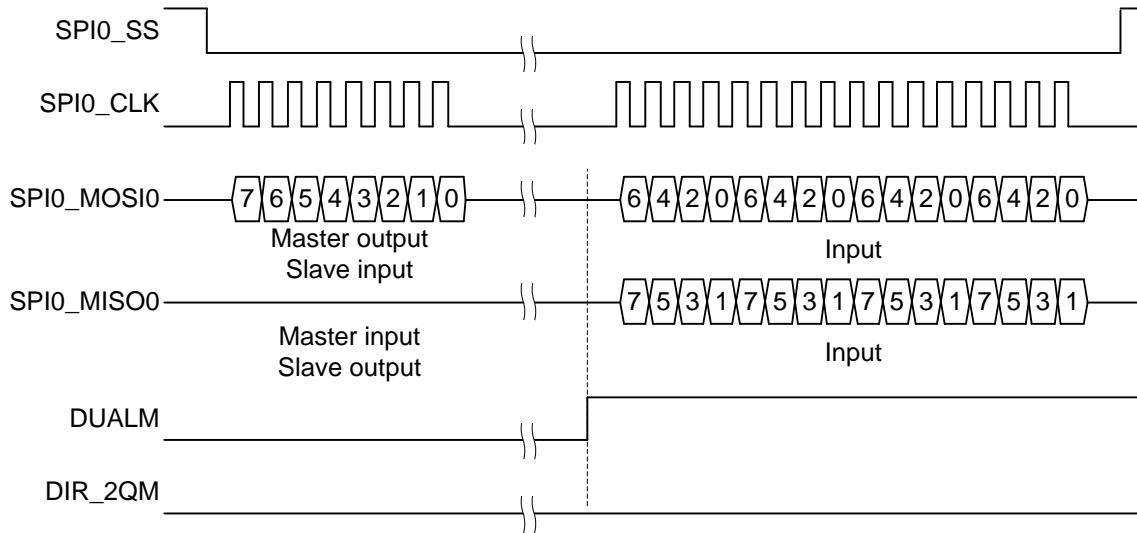
24.5.3 雙/四 IO 模式

當DUALM位 (CNTRL[22])被設置為1時，SPI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：

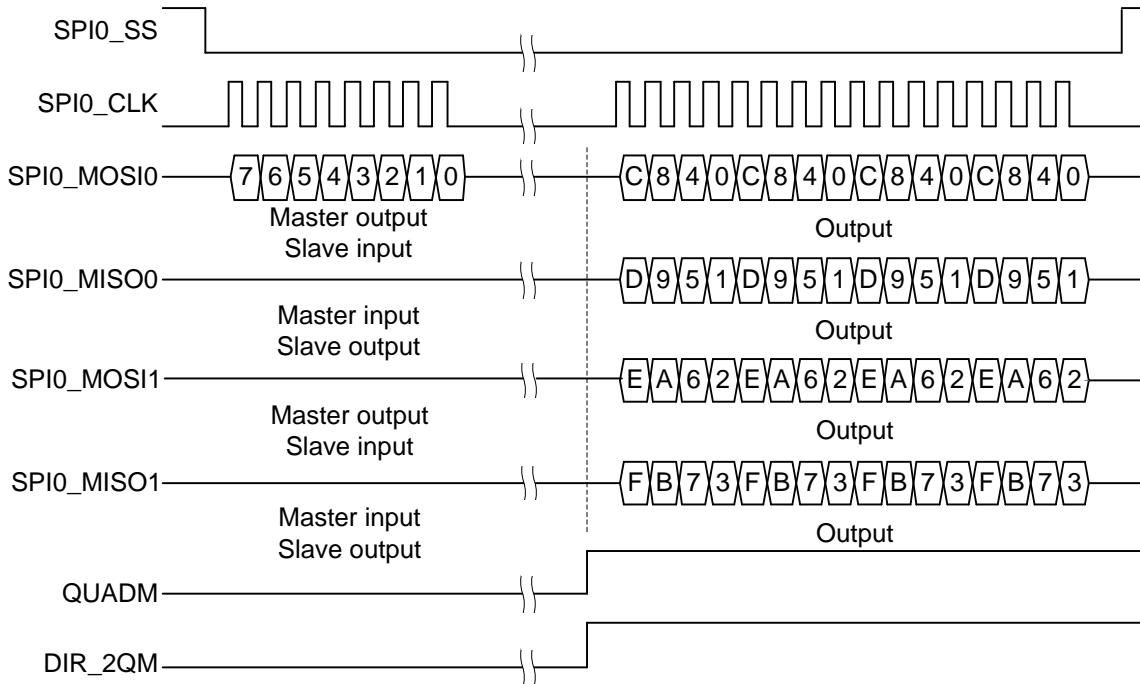


DIR_2QM位(CNTRL[20])用於定義資料傳輸的方向。當設置DIR_2QM位為1，SPI控制器向外部設備發送資料；當設置DIR_2QM位為0，SPI控制器從外部設備讀取數據。

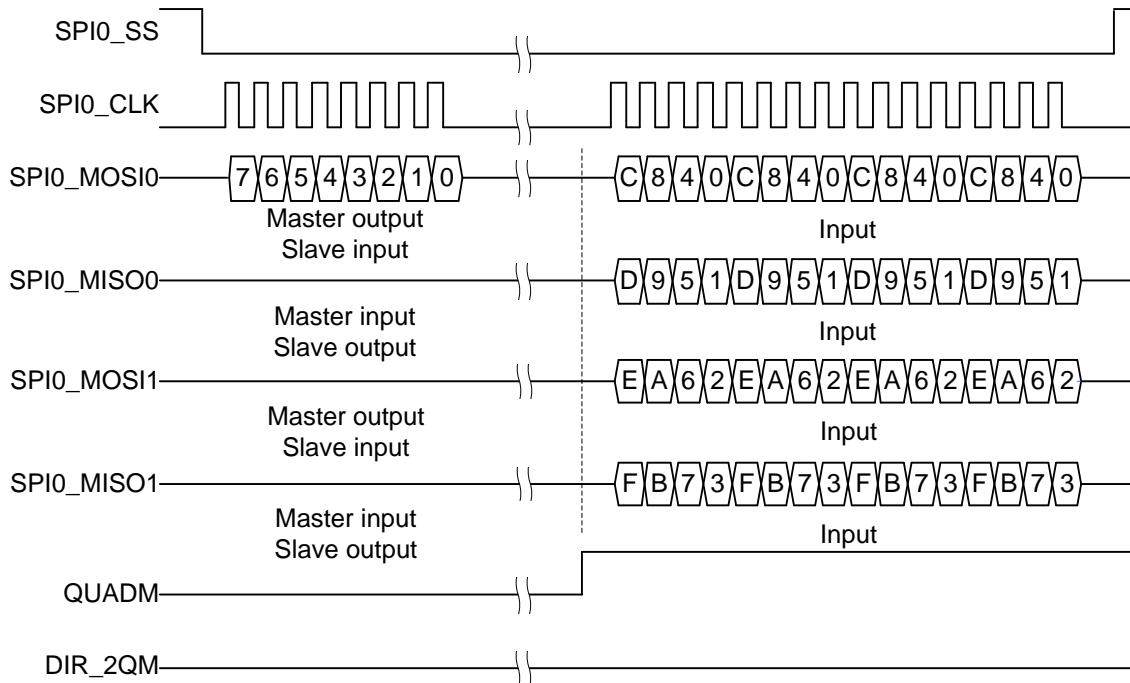
```
//使用雙IO模式，MOSI/MISO同時向外發送資料
CNTRL |= (0x1 << 22); //使能雙IO模式
CNTRL |= (0x1 << 20); //方向為輸出
TX0 = 0x12;
CNTRL |= 0x1; //啟動SPI
```

當QUADM位 (CNTRL[21])被設置為1時，SPI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：



DIR_2QM位(CNTRL[20])用於定義資料傳輸的方向。當設置DIR_2QM位為 1，SPI控制器向外部設備發送資料；當設置DIR_2QM位為 0，SPI控制器從外部設備讀取數據。

```
//使用四IO模式，MOSI/MISO同時向外發送資料
CNTRL |= (0x1 << 21); //使能四IO模式
CNTRL |= (0x1 << 20); //方向為輸出
TX0 = 0x12;
CNTRL |= 0x1; //啟動SPI
```

24.5.4 連續傳送

設置TX_NUM(CNTRL[9:8])，可決定在一次的SPI啟動中可連續傳輸的數據單位次數，範圍從1~4次。

```
//連續傳送4個8位元數據
CNTRL = (CNTRL & ~0xf8) | 0x8 << 3; //8位元傳輸
CNTRL |= (0x3 << 8); //連續傳送個4單位
TX0 = 0x12; //第一個數據
TX1 = 0x34; //第二個數據
TX2 = 0x56; //第三個數據
TX3 = 0x78; //第四個數據
CNTRL |= 0x1; //啟動SPI，連續傳送0x12, 0x34, 0x56, 0x78數據
```

24.5.5 中斷

當 SPI 控制器完成一次單元傳輸，單元傳輸中斷標誌 IF (CNTRL[16]) 將會被設置為 1。如果單元傳輸中斷使能位元 IE (CNTRL[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。

```
CNTRL |= 0x20000; //使能中斷  
CNTRL |= 0x1; //啟動SPI  
while(!spi_isr); //等待中斷  
CNTRL |= 0x10000; //清除IF位
```

如果沒設置IE位，仍可透過輪詢GO_BUSY位得知SPI傳輸動作完成與否。

```
CNTRL |= 0x1; //啟動SPI  
while(CNTRL & 0x1); //等待SPI完成動作
```

24.5.6 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 DIVIDER 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 (SSR[3])
3. 選擇有效電平位 SS_LVL (SSR[2])
4. 通過設定從機選擇寄存器位 SSR[0] 或 SSR[1] (SSR[1:0]) 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. 設定 CLK_POL位(CNTRL[31])以決定串列時鐘空閒狀態為低電平或高電平。
6. 在 TX_NEG 位元(CNTRL[2])選擇在串列時鐘的下降沿或上升沿發送資料。
7. 在 LSB 位 (CNTRL[10] = 0) 設定 MSB或LSB 優先傳輸。
8. 設定TX_BIT_LEN(CNTRL[7:3])，決定每次傳輸數據長度。
9. 決定TX_NUM(CNTRL[9:8])連續傳輸數(1~4)。
10. 將傳輸的數據寫入TX0寄存器。
11. 將 CNTRL寄存器中的GO_BUSY位設定成1，啟動SPI控制器。
12. 等待 SPI 中斷發生 (如果中斷使能位 IE 被置位)，或者輪詢 GO_BUSY 位，直到該位元被硬體自動清零。
13. 從RX0 寄存器讀出接收到的資料
14. 完成一次的傳輸

25 計時器控制器 (TIMER)

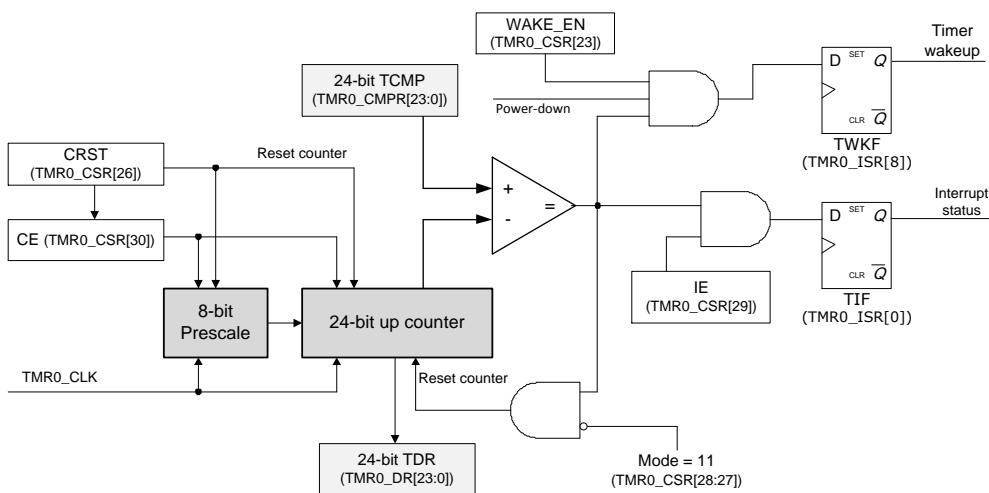
25.1 概述

NUC970 帶有五個計時器模塊.TIMER0, TIMER1, TIMER2, TIMER3, 以及 TIMER4. 用戶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間隔時間測量, 時鐘產生, 延遲時間等功能. 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.

25.2 特性

- 每個通道均有獨立的時鐘源開關.
- 每個通道均有24位上數計數器以及獨立的中斷.
- 內部 8 位預分頻計數器.
- 內部 24 位上數型計數器通過 TDR(TMR_DR[23:0]) 可讀取.
- 支持單次, 週期, 以及連續計數等操作模式.
- 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 TCMP).
- 計數周期= $(1 / \text{TMRx_CLK}) * 2^8 * 2^{24}$

25.3 方塊圖



25.4 寄存器

Register	Offset	R/W	Description	Reset Value
TMR Base Address:				
TMR0_BA = 0xB800_1000				
TMR1_BA = 0xB800_1010				
TMR2_BA = 0xB800_1020				
TMR3_BA = 0xB800_1030				
TMR4_BA = 0xB800_1040				
TMR0_CSR	TMR0_BA+0x000	R/W	Timer Control and Status Register 0	0x0000_0005
TMR0_CMPR	TMR0_BA+0x004	R/W	Timer Compare Register 0	0x0000_0000
TMR0_DR	TMR0_BA+0x008	R	Timer Data Register 0	0x0000_0000
TMR1_CSR	TMR1_BA+0x000	R/W	Timer Control and Status Register 1	0x0000_0005
TMR1_CMPR	TMR1_BA+0x004	R/W	Timer Compare Register 1	0x0000_0000
TMR1_DR	TMR1_BA+0x008	R	Timer Data Register 1	0x0000_0000
TMR2_CSR	TMR2_BA+0x000	R/W	Timer Control and Status Register 2	0x0000_0005
TMR2_CMPR	TMR2_BA+0x004	R/W	Timer Compare Register 2	0x0000_0000
TMR2_DR	TMR2_BA+0x008	R	Timer Data Register 2	0x0000_0000
TMR3_CSR	TMR3_BA+0x000	R/W	Timer Control and Status Register 3	0x0000_0005
TMR3_CMPR	TMR3_BA+0x004	R/W	Timer Compare Register 3	0x0000_0000
TMR3_DR	TMR3_BA+0x008	R	Timer Data Register 3	0x0000_0000
TMR4_CSR	TMR4_BA+0x000	R/W	Timer Control and Status Register 4	0x0000_0005
TMR4_CMPR	TMR4_BA+0x004	R/W	Timer Compare Register 4	0x0000_0000
TMR4_DR	TMR4_BA+0x008	R	Timer Data Register 4	0x0000_0000
TMR_ISR	TMR0_BA+0x060	R/W	Timer Interrupt Status Register	0x0000_0000

25.5 功能描述

25.5.1 計時器初始化

計時器可以依以下流程初始化, 並開始計時:

1. 將 CE (TMRx_CSR[30]) 清0, 停止計時器.
2. 設置操作模式 MODE (TMRx_CSR[28:27])
3. 若要使能中斷, 將 IE (TMRx_CSR[29]) 置 1, 否則清 0.
4. 設置預分頻PRESCALE (TMRx_CSR[7:0])

5. 設置比較值 TCMP (TMRx_CMPR[24:0])
6. 將CE (TMRx_CSR[30]) 置 1, 計時器開始上數計數.

25.5.2 中斷處理

每個計時器都有獨立的中斷, 當觸發時, TMR_ISR[4:0] 裡相對應的位置會置 1. 所以第 0 位置 1 表示 TMR0 中斷發生, 第 1 位置 1 表示 TMR1 中斷發生, 第 2 位置 1 表示 TMR2 中斷發生... 以此類推. 發生中斷後, 這些 TMR_ISR 的相關位可以使用寫 1 的方式清 0.

須注意一點, 若是 IE (TMRx_CSR[29]) 位沒有設 1, 則 TMR_ISR 裡相對應的位置不會被置 1. 所以在使用輪詢模式時, IE 依然要置 1, 軟件才能讀取 TMR_ISR 做判斷. 只要 AIC 裡不使能相關的中斷, CPU 就不會收到中斷.

25.5.3 計時器頻率

每個計時器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:

$$\text{頻率} = \text{TMRx_CLK} / ((\text{PRESCALE} + 1) * \text{TCMP})$$

其中 TMRx_CLK 為計時器時鐘源, PRESCALE 預分頻定義在 TMRx_CSR[7:0], TCMP 比較值定義在 TMRx_CMPR[24:0]. 所以已 12MHz 晶振當作計時器時鐘的話, 最快及最小頻率分別為 $12\text{MHz} / ((0 + 1) * 1)$ 以及 $12\text{MHz} / ((0xFF + 1) * 0xFFFFFFF)$. 以下表格列出了幾組使用 12MHz 當成時鐘源, 設定頻率為 10Hz, 100Hz, 1000Hz 的方式.

輸出頻率	PRESCALE (TMRx_CSR[7:0])	TCMP (TMRx_CMPR[24:0])
10Hz	0	0x124F80
10Hz	9	0x1D4C0
100Hz	9	0x2EE0
100Hz	19	0x1770
1000Hz	4	0x960
1000Hz	9	0x4B0

25.5.4 單次模式

如果計時器控制器工作在單次模式, MODE(TMRx_CSR[28:27]) 為 0x0, 且 CE (TMRx_CSR[30] 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_DR 值) 達到計時器比較器 (TMRx_CMPR) 的值, 且 IE (TMRx_CSR[29] 計時器中斷使能位) 被設置為 1, 則 TMR_ISR 中, 相對應的中斷號將會被設置為 1. 若是 AIC 中, 相對應的中斷有使能, AIC 會通知 CPU 計時器計數溢出發生. 如果 IE 被設置為 0, 則溢出發生時, IS 不會被置 1, 也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_DR 的值) 達到計時器比較器 (TMRx_CMPR) 的值, 計時器計數操作停止, 計時器計數器的值 (TMRx_DR 的值) 恢復到計數初始值 0, 然後 CE 被計時器控制器自動清除為 0. 也即是說, 在程式設定好計時器, 並開始計時後, 計時器計數和與 TMRx_CMPR 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.

25.5.5 週期模式

如果計時器工作在週期模式, MODE(TMRx_CSR[28:27]) 為 0x1, 且 CE (TMRx_CSR[30] 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_DR 的值) 等於計時器比較寄存器 (TMRx_CMPR) 的值, 且 IE (TMRx_CSR[29] 計時器中斷使能位) 被設置為 1, 則 TMR_ISR 中, 相對應的中斷號將會被設置為 1. 若是 AIC 中, 相對應的中斷有使能, AIC 會通知 CPU 計時器計數溢出發生. 如果 IE 被設置為 0, 則溢出發生時, IS 不會被置 1, 也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_DR 的值) 達到計時器比較寄存器 (TMRx_CMPR) 的值, 計時器計數器的值 (TMRx_DR 的值) 會恢復到計數初值 0, 且 CE 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 TMR_ISR 相對應的中斷位被軟體清除, 一旦計時器計數值 (TMRx_DR 的值) 再一次達到計時器比較寄存器 (TMRx_CMPR) 的值, TMR_IS 相對應的中斷位將會被再次設置為 1. 也即是說, 計時器計數和與 TMRx_CMPR 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 CE 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.

25.5.6 連續計數模式

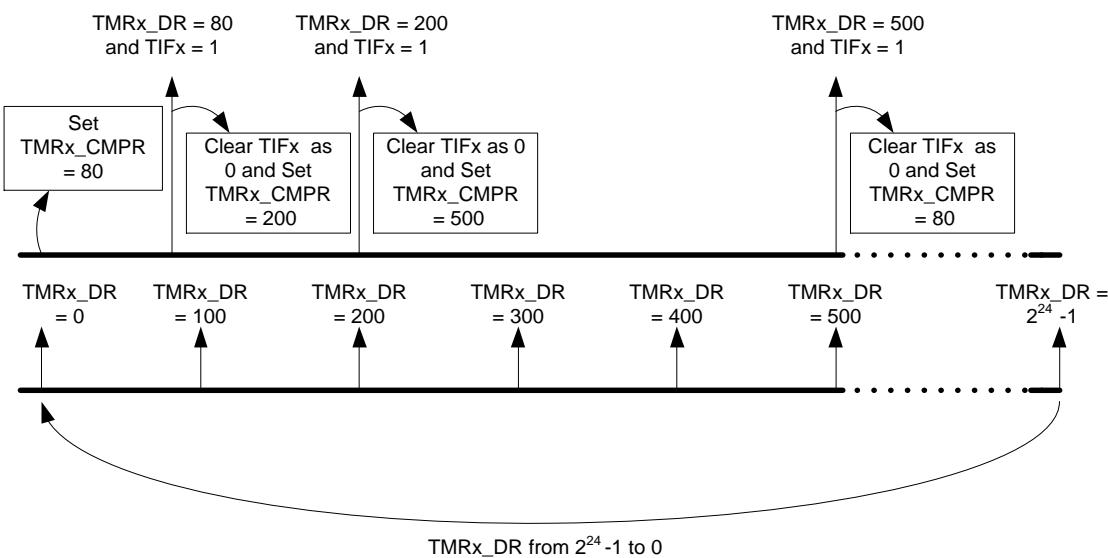
如果計時器工作在連續計數模式, MODE(TMRx_CSR[28:27]) 為 0x3, 且 CE (TMRx_CSR[30] 計時器計數器使能位) 被設置為 1, 計時器計數器開始向上計數. 一旦計時器計數器的值 (TMRx_DR 的值) 達到計時器比較寄存器 (TMRx_CMPR) 的值且 IE (TMRx_CSR[29] 計時器中斷使能位) 被設置為 1, 則 TMR_ISR 中, 相對應的中斷號將會被設置為 1. 若是 AIC 中, 相對應的中斷有使能, AIC 會通知 CPU 計時器計數溢出發生. 如果 IE 被設置為 0, 則溢出發生時, IS 不會被置 1, 也不會有中斷信號產生.

在這種操作模式下, 一旦計時器計數器的值 (TMRx_DR 的值) 達到計時器比較寄存器 (TMRx_CMPR) 的值, CE 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_DR 的值) 到計數初值 0. 用戶立即可以改變不同的計時器比較寄存器 (TMRx_CMPR) 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_CMPR) 被設置為 80 (計時器比較寄存器 (TMRx_CMPR) 的值必須大於 1, 且小於 2^{24}). 一旦計時器計數器的值 (TMRx_DR 的值) 達到 80, TMR_ISR 中相對應的中斷位將會被設置為 1, 且 CE 保持為 1 (繼續計數使能), 計時器計數器的值 (TMRx_DR

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 開始上數 0, 1, 2, 3... 到 $2^{24}-1$, 如此往復。接下來, 如果用戶程式在發生中斷時, 設定計時器比較寄存器 (TMRx_CMPR) 的值為 200, 且中斷位被清除為 0, 當計時器比較寄存器的值 (TMRx_DR 的值) 達到 200, TMR_ISR 裡對應的中斷位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_CMPR) 的值為 500, 並清除中斷位, 當計時器計數器的值 (TMRx_DR 的值) 達到 500, TMR_ISR 裡對應的中斷位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_DR 的值) 總是保持持續向上計數, 即便 TMR_ISR 裡對應的中斷位為 1。所以這個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。



26 通用異步收發器 (UART)

26.1 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(INT_RDA)，發送FIFO空中斷(INT_THRE)，線狀態中斷(break錯誤，校驗錯誤，格式錯誤或者RS-485中斷)(INT_RLS)，超時中斷(INT_TOUT)，MODEM狀態中斷(INT_MODEM)，緩存錯誤中斷(INT_BUF_ERR)，喚醒中斷 (INT_WAKE)，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(INT_ABAUD)和LIN功能中斷(INT_LIN)。

UART1/2/4/6/8/10內嵌一個64位發送FIFO (TX_FIFO)和一個64位接收FIFO (RX_FIFO)來降低向CPU申請中斷的次數。而UART0/3/5/7/9則內嵌一個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控制器支持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CTS_n喚醒功能，當系統處於掉電模式時，一個傳入的CTS_n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 (UART_x_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RTS_TRI_LEV(UA_FCR [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(FUN_SEL(UA_FUN_SEL[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，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.

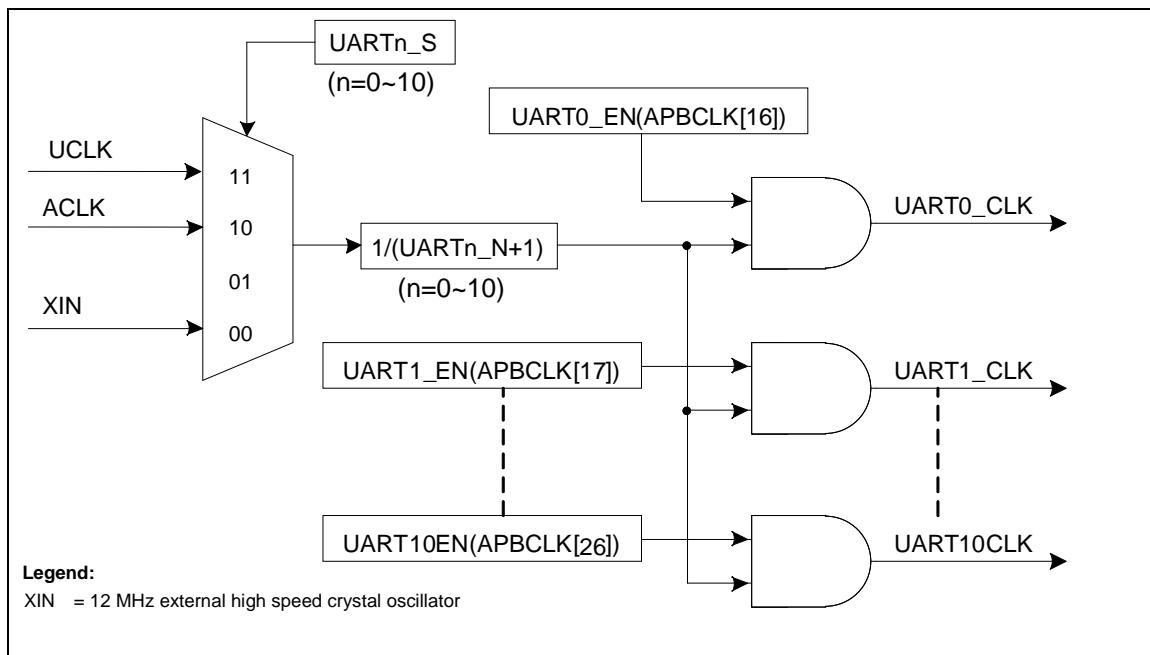
LIN模式可以通過設定UART_FUN_SEL寄存器中的LIN_EN位來選擇。在LIN模式下，依照LIN標準，要求資料格式為1個起始位元，8個資料位元和1個停止位

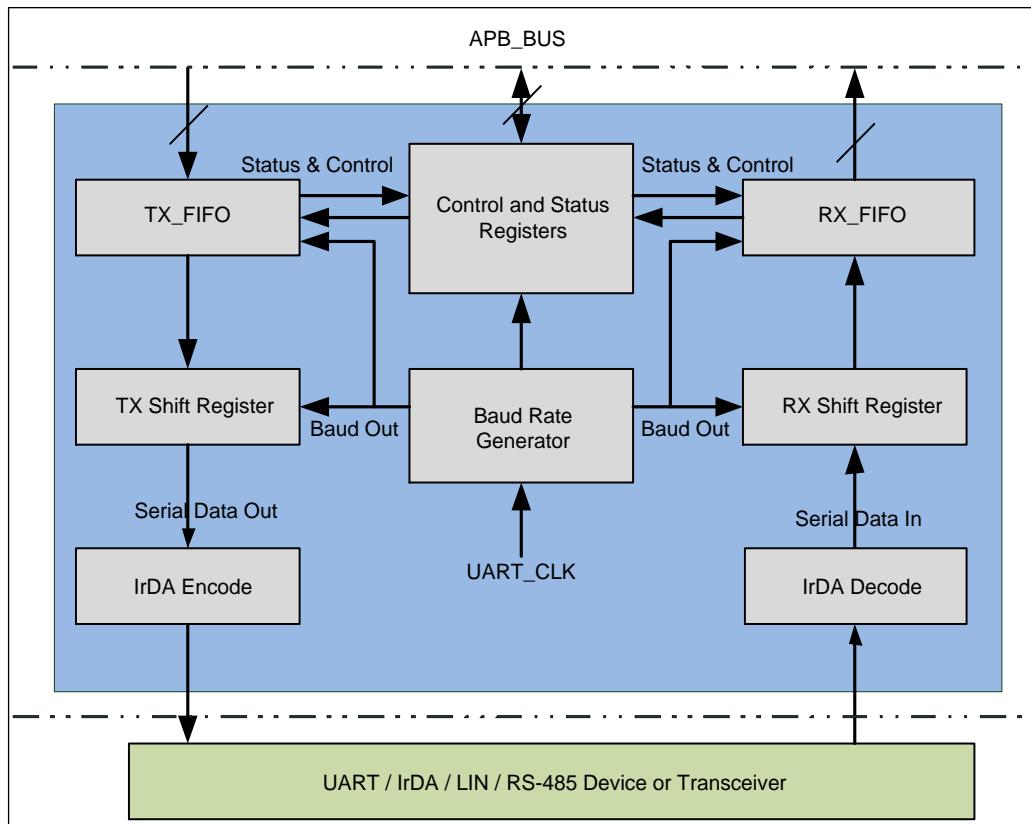
26.2特性

- 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(UA_TOR[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
- 支援RS485功能模式

26.3 方塊圖





26.4 寄存器

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

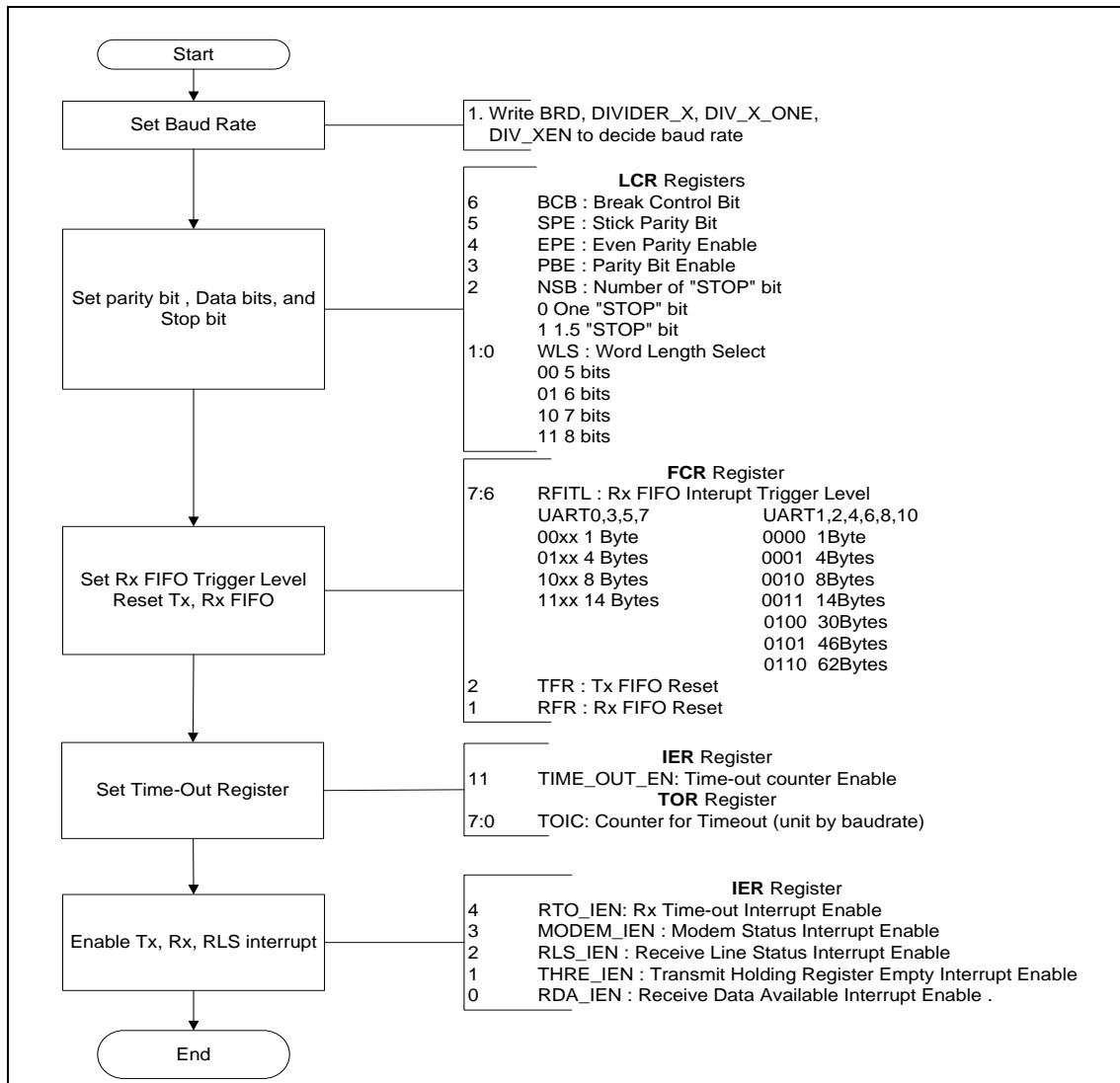
Register	Offset	R/W	Description	Reset Value
UART Base Address :				
Channel0	UART0_BA (Normal Speed)	= 0xB800_0000		
Channel1	UART1_BA (High-speed)	= 0xB800_0100		
Channel2	UART2_BA (High-speed)	= 0xB800_0200		
Channel3	UART3_BA (Normal Speed)	= 0xB800_0300		
Channel4	UART4_BA (High-speed)	= 0xB800_0400		
Channel5	UART5_BA (Normal Speed)	= 0xB800_0500		
Channel6	UART6_BA (High-speed)	= 0xB800_0600		
Channel7	UART7_BA (Normal Speed)	= 0xB800_0700		
Channel8	UART8_BA (High-speed)	= 0xB800_0800		
Channel9	UART9_BA (Normal Speed)	= 0xB800_0900		
Channel10	UART10_BA (High-speed)	= 0xB800_0A00		
UA_RBR	UART_BA+0x00	R	UART Receive Buffer Register	Undefined
UA_THR	UART_BA+0x00	W	UART Transmit Holding Register	Undefined
UA_IER	UART_BA+0x04	R/W	UART Interrupt Enable Register	0x0000_0000
UA_FCR	UART_BA+0x08	R/W	UART FIFO Control Register	0x0000_0000

UA_LCR	UART_BA+0x0C	R/W	UART Line Control Register	0x0000_0000
UA_MCR	UART_BA+0x10	R/W	UART Modem Control Register	0x0000_0000
UA_MSR	UART_BA+0x14	R/W	UART Modem Status Register	0x0000_0000
UA_FSR	UART_BA+0x18	R/W	UART FIFO Status Register	0x1040_4000
UA_ISR	UART_BA+0x1C	R/W	UART Interrupt Status Register	0x0000_0002
UA_TOR	UART_BA+0x20	R/W	UART Time-out Register	0x0000_0000
UA_BAUD	UART_BA+0x24	R/W	UART Baud Rate Divisor Register	0x0F00_0000
UA_IRCR	UART_BA+0x28	R/W	UART IrDA Control Register	0x0000_0040
UA_ALT_CSR	UART_BA+0x2C	R/W	UART Alternate Control/Status Register	0x0000_000C
UA_FUN_SEL	UART_BA+0x30	R/W	UART Function Select Register	0x0000_0000
UA_LIN_CTL	UART_BA+0x34	R/W	UART LIN Control Register	0x000C_0000
UA_LIN_SR	UART_BA+0x38	R/W	UART LIN Status Register	0x0000_0000
UA_SC_CTL	UART_BA+0x40	R/W	UART SC Control Register	0x0000_0000
UA_SC_FSR	UART_BA+0x44	R/W	UART SC Flag Status Register	0x0000_0000

26.5功能描述

26.5.1 初始话

在开始使用UART传输之前，必须要对UART做初始设置，包含波特率，奇偶检验位，数据位，停止位的设置。另外，如果将TX，RX，RLS的中断使能，则在每一次的传输结束时，就可以触发相对的的中断。



26.5.2 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 UA_FUN_SEL 寄存器中的 FUN_SEL 位來選擇，當 UART 控制器工作在 IrDA 模式時，必須通過設定 RFITL(UART_FCR[7:4]) = 000 來把接收 FIFO 觸發閾值設置為 1。

IrDA 模式下，DIV_16_EN(UA_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 UA_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_FUN_SEL 寄存器中的 FUN_SEL 位來選擇 IrDA 功能.
2. 設定 INV_TX(UA_IRCR[5]) = 0 選擇 TX 訊號不反相.
3. 設定 INV_RX(UA_IRCR[6]) = 1 選擇 RX 訊號反相.
4. 通過設定 TX_SELECT(UA_IRCR[2])選擇半雙工串列為 TX 或 RX。當 TX_SELECT(UA_IRCR[2]) = 1 時，是選擇 TX。當 TX_SELECT(UA_IRCR[2]) = 0 時，是選擇 RX。

26.5.3 RS485 功能模式

UART支援RS-485 9位元模式功能。RS-485模式可以通過設置寄存器UA_FUN_SEL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位元將會被配置為位址位元，對於資料字元，第9位要被設置為0，軟體可以程式設計UA_TLCTL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UA_ALT_CTL 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_TOR[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。

26.5.3.1 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RX_DIS(UA_FCR[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(RX_DIS(UA_FCR[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(RX_DIS(UA_FCR[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RX_DIS(UART_CTL[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RX_DIS(UART_CTL[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 UA_FUN_SEL 寄存器的 FUN_SEL 位來選擇 RS-485 功能。
2. 程式設計 UA_FCR 寄存器的 RX_DIS 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 UA_ALT_CTL(UA_ALT_CSR[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 RLS_IF(UA_ISR[2])和 RS485_ADD_DETF(UA_FSR[3])標誌。
5. 軟體可以通過設定 RX_DIS(UA_FCR[8])來決定是否接受接下來的資料。

6. 重複步驟 4 和步驟 5。

26.5.3.2 RS-485 自動位址識別操作模式 (AAD)

在 RS-485 自動位址識別操作模式下，接收器在檢測到位址位元組 (bit9 = “1”)，並且位址位元組資料與 ADDR_MATCH(UA_ALT_CSR[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 ADDR_MATCH(UA_ALT_CSR[31:24]) 的值不匹配。當處於 RS-485 AAD 模式時，不要寫任何值到 RX_DIS(UA_CTL[2]) 位。

程式設計流程示例：

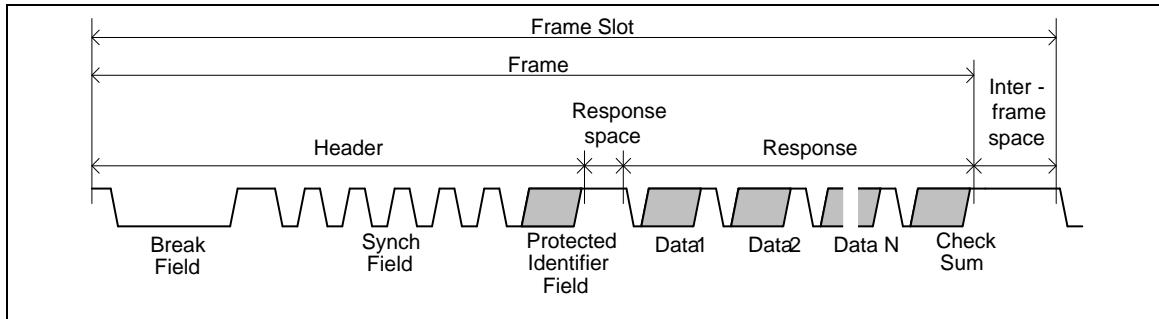
1. 程式設計 UART_FUN_SEL 寄存器中的 FUN_SEL 位來選擇 RS-485 功能.
2. 通過設定 UA_ALT_CSR[9] 寄存器來程式設計設定 RS485_AAD.
3. 當位址位元組被檢測到 (bit9 = “1”)，硬體會比較位址位元組與 ADDR_MATCH(UA_ALT_CSR[31:24]) 的值。
4. 如果位址位元組與 ADDR_MATCH(UA_ALT_CSR[31:24]) 的值相匹配，硬體會置位元 RLS_IS(UART_ISR[2]) 和 RS485_ADD_DETF(UA_FSR[3]) 標誌，接收器會存儲位址位元組到 RX-FIFO，接受接下來的資料，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 ADDR_MATCH(UA_ALT_CSR[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。

26.5.3.3 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 LEV_RTS(UA_MCR[9]) 位來改變 RTSn 驅動電平

26.5.4 LIN 功能模式

UART 支援 LIN 功能，LIN 模式可以通過設定 UA_FUN_SEL 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總線發送傳輸(TX)編程流程：

1. 設置寄存器 UA_FUS_SEL 的 LIN_EN 位使能 LIN 總線模式
2. 寫 LIN_BKFL(UA_LIN_CTL[19:16])選擇 break 域長度(break 域長度是 UA_LIN_BKFL+2)
3. 設置寄存器 LIN_TX_EN(UA_ALT_CSR[8])位開始傳輸(當 break 域操作完成，LIN_TXEN 將被自動清除)
4. 寫 0x55 到 UA THR 請求同步域傳輸
5. 寫保護標識符到 UA THR，請求標識符域傳輸
6. 當最後字節 THR 的停止位被發到總線後，硬件將置 UA_FSR 寄存器中的 TE_FLAG 為 1
7. 寫 N 字節數據和檢驗和(checksum)到 UA THR，然後重復步驟 5 和步驟 6 傳輸數據

LIN總線接收傳輸(RX)編程流程：

1. 設置寄存器 UA_FUS_SEL 的 LIN_EN 位使能 LIN 總線模式
2. 設置寄存器 LIN_RX_EN(UA_ALT_CSR[8]) = 1，使能 LIN RX 模式
3. 等待 UA_LIN_SR 中的標誌位 LIN_BKDET_F，用來檢查 RX 有無 break 域接收
4. 等待 UA_ISR 中的標誌位 RDA_IF 和讀寄存器 UA_RBR

27 USB 2.0 設備控制器

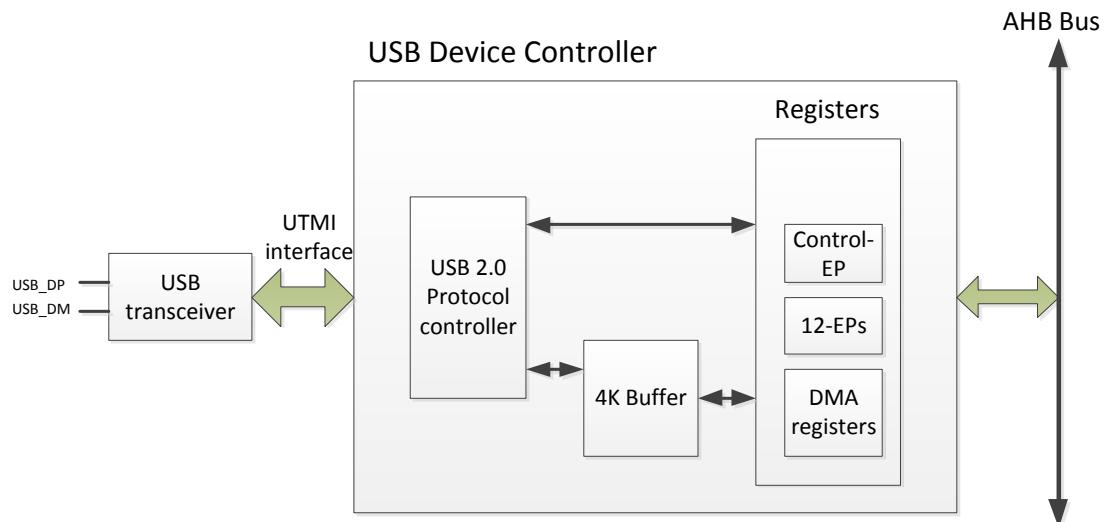
27.1 概述

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。

27.2 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。

27.3 方塊圖



27.4 寄存器

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

Register	Offset	R/W	Description	Reset Value
USBD Base Address:				
USBD_BA = 0x4001_9000				
USBD_GINTSTS	USBD_BA+0x000	R	Interrupt Status Low Register	0x0000_0000
USBD_GINTEN	USBD_BA+0x008	R/W	Interrupt Enable Low Register	0x0000_0001
USBD_BUSINTSTS	USBD_BA+0x010	R/W	USB Bus Interrupt Status Register	0x0000_0000
USBD_BUSINTEN	USBD_BA+0x014	R/W	USB Bus Interrupt Enable Register	0x0000_0040
USBD_OPER	USBD_BA+0x018	R/W	USB Operational Register	0x0000_0002
USBD_FRAMECNT	USBD_BA+0x01C	R	USB Frame Count Register	0x0000_0000
USBD_FADDR	USBD_BA+0x020	R/W	USB Function Address Register	0x0000_0000
USBD_TEST	USBD_BA+0x024	R/W	USB Test Mode Register	0x0000_0000
USBD_CEPDAT	USBD_BA+0x028	R/W	Control-Endpoint Data Buffer	0x0000_0000
USBD_CEPCTL	USBD_BA+0x02C	R/W	Control-Endpoint Control and Status	0x0000_0000
USBD_CEPINTEN	USBD_BA+0x030	R/W	Control-Endpoint Interrupt Enable	0x0000_0000
USBD_CEPINTSTS	USBD_BA+0x034	R/W	Control-Endpoint Interrupt Status	0x0000_1800
USBD_CEPTXCNT	USBD_BA+0x038	R/W	Control-Endpoint In-transfer Data Count	0x0000_0000
USBD_CEPRXCNT	USBD_BA+0x03C	R	Control-Endpoint Out-transfer Data Count	0x0000_0000
USBD_CEPDATCNT	USBD_BA+0x040	R	Control-Endpoint data count	0x0000_0000
USBD_SETUP1_0	USBD_BA+0x044	R	Setup1 & Setup0 bytes	0x0000_0000
USBD_SETUP3_2	USBD_BA+0x048	R	Setup3 & Setup2 Bytes	0x0000_0000
USBD_SETUP5_4	USBD_BA+0x04C	R	Setup5 & Setup4 Bytes	0x0000_0000
USBD_SETUP7_6	USBD_BA+0x050	R	Setup7 & Setup6 Bytes	0x0000_0000
USBD_CEPBUFSTART	USBD_BA+0x054	R/W	Control Endpoint RAM Start Address Register	0x0000_0000
USBD_CEPBUFEND	USBD_BA+0x058	R/W	Control Endpoint RAM End Address Register	0x0000_0000
USBD_DMACTL	USBD_BA+0x05C	R/W	DMA Control Status Register	0x0000_0000
USBD_DMACNT	USBD_BA+0x060	R/W	DMA Count Register	0x0000_0000
USBD_EPADAT	USBD_BA+0x064	R/W	Endpoint A Data Register	0x0000_0000
USBD_EPAINTSTS	USBD_BA+0x068	R/W	Endpoint A Interrupt Status Register	0x0000_0003
USBD_EPAINTEN	USBD_BA+0x06C	R/W	Endpoint A Interrupt Enable Register	0x0000_0000
USBD_EPADATCNT	USBD_BA+0x070	R	Endpoint A Data Available Count Register	0x0000_0000
USBD_EPARSPCTL	USBD_BA+0x074	R/W	Endpoint A Response Control Register	0x0000_0000
USBD_EPAMPS	USBD_BA+0x078	R/W	Endpoint A Maximum Packet Size Register	0x0000_0000
USBD_EPATXCNT	USBD_BA+0x07C	R/W	Endpoint A Transfer Count Register	0x0000_0000

USBD_EPACFG	USBD_BA+0x080	R/W	Endpoint A Configuration Register	0x0000_0012
USBD_EPABUFSTART	USBD_BA+0x084	R/W	Endpoint A RAM Start Address Register	0x0000_0000
USBD_EPABUFEND	USBD_BA+0x088	R/W	Endpoint A RAM End Address Register	0x0000_0000
USBD_EPBDAT	USBD_BA+0x08C	R/W	Endpoint B Data Register	0x0000_0000
USBD_EPBINTSTS	USBD_BA+0x090	R/W	Endpoint B Interrupt Status Register	0x0000_0003
USBD_EPBINTEN	USBD_BA+0x094	R/W	Endpoint B Interrupt Enable Register	0x0000_0000
USBD_EPBDATCNT	USBD_BA+0x098	R	Endpoint B Data Available Count Register	0x0000_0000
USBD_EPBRSPCTL	USBD_BA+0x09C	R/W	Endpoint B Response Control Register	0x0000_0000
USBD_EPBMPMS	USBD_BA+0x0A0	R/W	Endpoint B Maximum Packet Size Register	0x0000_0000
USBD_EPBTXCNT	USBD_BA+0x0A4	R/W	Endpoint B Transfer Count Register	0x0000_0000
USBD_EPBCFG	USBD_BA+0x0A8	R/W	Endpoint B Configuration Register	0x0000_0022
USBD_EPBBUFSTART	USBD_BA+0x0AC	R/W	Endpoint B RAM Start Address Register	0x0000_0000
USBD_EPBBUFEND	USBD_BA+0x0B0	R/W	Endpoint B RAM End Address Register	0x0000_0000
USBD_EPCDAT	USBD_BA+0x0B4	R/W	Endpoint C Data Register	0x0000_0000
USBD_EPCINTSTS	USBD_BA+0x0B8	R/W	Endpoint C Interrupt Status Register	0x0000_0003
USBD_EPCINTEN	USBD_BA+0x0BC	R/W	Endpoint C Interrupt Enable Register	0x0000_0000
USBD_EPCDATCNT	USBD_BA+0x0C0	R	Endpoint C Data Available Count Register	0x0000_0000
USBD_EPCRSPCTL	USBD_BA+0x0C4	R/W	Endpoint C Response Control Register	0x0000_0000
USBD_EPCMPS	USBD_BA+0x0C8	R/W	Endpoint C Maximum Packet Size Register	0x0000_0000
USBD_EPCTXCNT	USBD_BA+0x0CC	R/W	Endpoint C Transfer Count Register	0x0000_0000
USBD_EPCCFG	USBD_BA+0x0D0	R/W	Endpoint C Configuration Register	0x0000_0032
USBD_EPCBUFSTART	USBD_BA+0x0D4	R/W	Endpoint C RAM Start Address Register	0x0000_0000
USBD_EPCBUFEND	USBD_BA+0x0D8	R/W	Endpoint C RAM End Address Register	0x0000_0000
USBD_EPDDAT	USBD_BA+0x0DC	R/W	Endpoint D Data Register	0x0000_0000
USBD_EPDINTSTS	USBD_BA+0x0E0	R/W	Endpoint D Interrupt Status Register	0x0000_0003
USBD_EPDINTEN	USBD_BA+0x0E4	R/W	Endpoint D Interrupt Enable Register	0x0000_0000
USBD_EPDATCNT	USBD_BA+0x0E8	R	Endpoint D Data Available Count Register	0x0000_0000
USBD_EPDRSPCTL	USBD_BA+0x0EC	R/W	Endpoint D Response Control Register	0x0000_0000
USBD_EPDMPMS	USBD_BA+0x0F0	R/W	Endpoint D Maximum Packet Size Register	0x0000_0000
USBD_EPDTXCNT	USBD_BA+0x0F4	R/W	Endpoint D Transfer Count Register	0x0000_0000
USBD_EPDCFG	USBD_BA+0x0F8	R/W	Endpoint D Configuration Register	0x0000_0042
USBD_EPDBUFSTART	USBD_BA+0x0FC	R/W	Endpoint D RAM Start Address Register	0x0000_0000
USBD_EPDBUFEND	USBD_BA+0x100	R/W	Endpoint D RAM End Address Register	0x0000_0000
USBD_EPEDAT	USBD_BA+0x104	R/W	Endpoint E Data Register	0x0000_0000
USBD_EPEINTSTS	USBD_BA+0x108	R/W	Endpoint E Interrupt Status Register	0x0000_0003

USBD_EPEINTEN	USBD_BA+0x10C	R/W	Endpoint E Interrupt Enable Register	0x0000_0000
USBD_EPEDATCNT	USBD_BA+0x110	R	Endpoint E Data Available Count Register	0x0000_0000
USBD_EPERSPCTL	USBD_BA+0x114	R/W	Endpoint E Response Control Register	0x0000_0000
USBD_EPEMPS	USBD_BA+0x118	R/W	Endpoint E Maximum Packet Size Register	0x0000_0000
USBD_EPETXCNT	USBD_BA+0x11C	R/W	Endpoint E Transfer Count Register	0x0000_0000
USBD_EPECFG	USBD_BA+0x120	R/W	Endpoint E Configuration Register	0x0000_0052
USBD_EPEBUFSTART	USBD_BA+0x124	R/W	Endpoint E RAM Start Address Register	0x0000_0000
USBD_EPEBUFEND	USBD_BA+0x128	R/W	Endpoint E RAM End Address Register	0x0000_0000
USBD_EPFDAT	USBD_BA+0x12C	R/W	Endpoint F Data Register	0x0000_0000
USBD_EPFINTSTS	USBD_BA+0x130	R/W	Endpoint F Interrupt Status Register	0x0000_0003
USBD_EPFINTEN	USBD_BA+0x134	R/W	Endpoint F Interrupt Enable Register	0x0000_0000
USBD_EPFDATCNT	USBD_BA+0x138	R	Endpoint F Data Available Count Register	0x0000_0000
USBD_EPFRSPCTL	USBD_BA+0x13C	R/W	Endpoint F Response Control Register	0x0000_0000
USBD_EPFMPS	USBD_BA+0x140	R/W	Endpoint F Maximum Packet Size Register	0x0000_0000
USBD_EPFTXCNT	USBD_BA+0x144	R/W	Endpoint F Transfer Count Register	0x0000_0000
USBD_EPFCFG	USBD_BA+0x148	R/W	Endpoint F Configuration Register	0x0000_0062
USBD_EPFBUFSTART	USBD_BA+0x14C	R/W	Endpoint F RAM Start Address Register	0x0000_0000
USBD_EPFBUFEND	USBD_BA+0x150	R/W	Endpoint F RAM End Address Register	0x0000_0000
USBD_EPGDAT	USBD_BA+0x154	R/W	Endpoint G Data Register	0x0000_0000
USBD_EPGINTSTS	USBD_BA+0x158	R/W	Endpoint G Interrupt Status Register	0x0000_0003
USBD_EPGINTEN	USBD_BA+0x15C	R/W	Endpoint G Interrupt Enable Register	0x0000_0000
USBD_EPGDATCNT	USBD_BA+0x160	R	Endpoint G Data Available Count Register	0x0000_0000
USBD_EPGRSPCTL	USBD_BA+0x164	R/W	Endpoint G Response Control Register	0x0000_0000
USBD_EPGMPS	USBD_BA+0x168	R/W	Endpoint G Maximum Packet Size Register	0x0000_0000
USBD_EPGTXCNT	USBD_BA+0x16C	R/W	Endpoint G Transfer Count Register	0x0000_0000
USBD_EPGCFG	USBD_BA+0x170	R/W	Endpoint G Configuration Register	0x0000_0072
USBD_EPGBUFSTART	USBD_BA+0x174	R/W	Endpoint G RAM Start Address Register	0x0000_0000
USBD_EPGBUFEND	USBD_BA+0x178	R/W	Endpoint G RAM End Address Register	0x0000_0000
USBD_EPHDAT	USBD_BA+0x17C	R/W	Endpoint H Data Register	0x0000_0000
USBD_EPHINTSTS	USBD_BA+0x180	R/W	Endpoint H Interrupt Status Register	0x0000_0003
USBD_EPHINTEN	USBD_BA+0x184	R/W	Endpoint H Interrupt Enable Register	0x0000_0000
USBD_EPHDATCNT	USBD_BA+0x188	R	Endpoint H Data Available Count Register	0x0000_0000
USBD_EPHRSPCTL	USBD_BA+0x18C	R/W	Endpoint H Response Control Register	0x0000_0000
USBD_EPHMPS	USBD_BA+0x190	R/W	Endpoint H Maximum Packet Size Register	0x0000_0000
USBD_EPHTXCNT	USBD_BA+0x194	R/W	Endpoint H Transfer Count Register	0x0000_0000

USBD_EPHCFG	USBD_BA+0x198	R/W	Endpoint H Configuration Register	0x0000_0082
USBD_EPHBUFSTART	USBD_BA+0x19C	R/W	Endpoint H RAM Start Address Register	0x0000_0000
USBD_EPHBUFEND	USBD_BA+0x1A0	R/W	Endpoint H RAM End Address Register	0x0000_0000
USBD_EPIDAT	USBD_BA+0x1A4	R/W	Endpoint I Data Register	0x0000_0000
USBD_EPIINTSTS	USBD_BA+0x1A8	R/W	Endpoint I Interrupt Status Register	0x0000_0003
USBD_EPIINTEN	USBD_BA+0x1AC	R/W	Endpoint I Interrupt Enable Register	0x0000_0000
USBD_EPIDATCNT	USBD_BA+0x1B0	R	Endpoint I Data Available Count Register	0x0000_0000
USBD_EPIRSPCTL	USBD_BA+0x1B4	R/W	Endpoint I Response Control Register	0x0000_0000
USBD_EPIMPS	USBD_BA+0x1B8	R/W	Endpoint I Maximum Packet Size Register	0x0000_0000
USBD_EPITXCNT	USBD_BA+0x1BC	R/W	Endpoint I Transfer Count Register	0x0000_0000
USBD_EPICFG	USBD_BA+0x1C0	R/W	Endpoint I Configuration Register	0x0000_0092
USBD_EPIBUFSTART	USBD_BA+0x1C4	R/W	Endpoint I RAM Start Address Register	0x0000_0000
USBD_EPIBUFEND	USBD_BA+0x1C8	R/W	Endpoint I RAM End Address Register	0x0000_0000
USBD_EPJDAT	USBD_BA+0x1CC	R/W	Endpoint J Data Register	0x0000_0000
USBD_EPJINTSTS	USBD_BA+0x1D0	R/W	Endpoint J Interrupt Status Register	0x0000_0003
USBD_EPJINTEN	USBD_BA+0x1D4	R/W	Endpoint J Interrupt Enable Register	0x0000_0000
USBD_EPJDATCNT	USBD_BA+0x1D8	R	Endpoint J Data Available Count Register	0x0000_0000
USBD_EPJRSPCTL	USBD_BA+0x1DC	R/W	Endpoint J Response Control Register	0x0000_0000
USBD_EPJMPMS	USBD_BA+0x1E0	R/W	Endpoint J Maximum Packet Size Register	0x0000_0000
USBD_EPJTXCNT	USBD_BA+0x1E4	R/W	Endpoint J Transfer Count Register	0x0000_0000
USBD_EPJCFG	USBD_BA+0x1E8	R/W	Endpoint J Configuration Register	0x0000_00A2
USBD_EPJBUFSTART	USBD_BA+0x1EC	R/W	Endpoint J RAM Start Address Register	0x0000_0000
USBD_EPJBUFEND	USBD_BA+0x1F0	R/W	Endpoint J RAM End Address Register	0x0000_0000
USBD_EPKDAT	USBD_BA+0x1F4	R/W	Endpoint K Data Register	0x0000_0000
USBD_EPKINTSTS	USBD_BA+0x1F8	R/W	Endpoint K Interrupt Status Register	0x0000_0003
USBD_EPKINTEN	USBD_BA+0x1FC	R/W	Endpoint K Interrupt Enable Register	0x0000_0000
USBD_EPKDATCNT	USBD_BA+0x200	R	Endpoint K Data Available Count Register	0x0000_0000
USBD_EPKRSPCTL	USBD_BA+0x204	R/W	Endpoint K Response Control Register	0x0000_0000
USBD_EPKMPS	USBD_BA+0x208	R/W	Endpoint K Maximum Packet Size Register	0x0000_0000
USBD_EPKTXCNT	USBD_BA+0x20C	R/W	Endpoint K Transfer Count Register	0x0000_0000
USBD_EPKCFG	USBD_BA+0x210	R/W	Endpoint K Configuration Register	0x0000_00B2
USBD_EPKBUFSTART	USBD_BA+0x214	R/W	Endpoint K RAM Start Address Register	0x0000_0000
USBD_EPKBUFEND	USBD_BA+0x218	R/W	Endpoint K RAM End Address Register	0x0000_0000
USBD_EPLDAT	USBD_BA+0x21C	R/W	Endpoint L Data Register	0x0000_0000
USBD_EPLINTSTS	USBD_BA+0x220	R/W	Endpoint L Interrupt Status Register	0x0000_0003

USBD_EPLINTEN	USBD_BA+0x224	R/W	Endpoint L Interrupt Enable Register	0x0000_0000
USBD_EPLDATCNT	USBD_BA+0x228	R	Endpoint L Data Available Count Register	0x0000_0000
USBD_EPLRSPCTL	USBD_BA+0x22C	R/W	Endpoint L Response Control Register	0x0000_0000
USBD_EPLMPS	USBD_BA+0x230	R/W	Endpoint L Maximum Packet Size Register	0x0000_0000
USBD_EPLTWCNT	USBD_BA+0x234	R/W	Endpoint L Transfer Count Register	0x0000_0000
USBD_EPLCFG	USBD_BA+0x238	R/W	Endpoint L Configuration Register	0x0000_00C2
USBD_EPLBUFSTART	USBD_BA+0x23C	R/W	Endpoint L RAM Start Address Register	0x0000_0000
USBD_EPLBUFEND	USBD_BA+0x240	R/W	Endpoint L RAM End Address Register	0x0000_0000
USBD_DMAADDR	USBD_BA+0x700	R/W	AHB DMA Address Register	0x0000_0000
USBD_PHYCTL	USBD_BA+0x704	R/W	USB PHY Control Register	0x0000_0420

27.5 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。

27.5.1 初始化

初始化USB設備控制器的步驟：

1. 設置多功能控制GPH0，SYS_GPH_MFPL要填入0x7。
2. 設置CLK_HCLKEN寄存器USBD位。
3. 設置USBD_PHYCTL寄存器PHYEN位，始能USB PHY。
4. USBD_EPAMPS寄存器填入0x8，輪詢USBD_EPAMPS寄存器，讀出的值是否等於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USBD_EPARSPCTL寄存器MODE位為0，選擇自動驗證模式。

- (2) USBD_EPAMPS 寄存器填入 512，表示最大数据包为 512 字节。
 - (3) 设置 USBD_EPACFG 寄存器 EPNUM 位为 1，EPDIR 位为 1，EPTYPE 位为 01，EPEN 位为 1。
 - (4) USBD_EPABUFSTART 寄存器填入 0x200，USBD_EPABUFEND 寄存器填入 0x3FF，表示端点 FIFO 长度为 512 字节。
6. 配置端点 B 为 Bulk-Out 类型、端点号为 2。
- (1) 设置 USBD_EPBINTEN 寄存器 RXPKIEN 位，始能接收数据中断。
 - (2) 设置 USBD_EPRSPCTL 寄存器 MODE 位为 0，选择自动验證模式。
 - (3) USBD_EPBMPMS 寄存器填入 512，表示最大数据包为 512 字节。
 - (4) 设置 USBD_EPBCFG 寄存器 EPNUM 位为 2，EPDIR 位为 0，EPTYPE 位为 01，EPEN 位为 1。
 - (5) USBD_EPBBUFSTART 寄存器填入 0x400，USBD_EPBBUFEND 寄存器填入 0x5FF，表示端点 FIFO 长度为 512 字节。
7. 设置 USBD_GINTEN 寄存器 USBIEN、CEPIEN、EPAIEN、EPBIEN 位，始能 USB 控制端点、端点 A、端点 B 的中断。
8. 设置 USBD_BUSINTEN 寄存器 RSTIEN、RESUMEIEN、DMADONEIEN、VBUSDETEN 位，始能 USB 复位、恢复、DMA 完成、插拔的中断。
9. 设置 USBD_OPER 寄存器 HISPDEN 位，始能高速模式。
10. 清除 USBD_FADDR 寄存器。
11. 配置控制端点（端点 0）
- (1) USBD_CEPBUFSTART 寄存器填入 0x0，USBD_CEPBUFEND 寄存器填入 0x7F，表示端点 FIFO 长度为 128 字节。
 - (2) 设置 USBD_CEPINTEN 寄存器 SETUPPKIEN、STSdoneIEN 位，始能控制端点的安装包和状态完成中断。
12. 轮询 USBD_PHYCTL 寄存器 VBUSDET 位，为 1 代表接上主机，设置 USBD_PHYCTL 寄存器 DPPUEN 位，清除 SE0。

27.5.2 中断处理程序

处理 USBD 控制器的中断如下：

1. 读取 USBD_GINTSTS 寄存器和 USBD_GINTEN 寄存器做屏蔽，可以得知是哪种中断产生。
2. 读取 USBD_BUSINTSTS 寄存器和 USBD_BUSINTEN 寄存器做屏蔽，如果相匹配就处理 USB BUS 的中断。

3. 讀取USBD_CEPINTSTS寄存器和USBD_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USBD_EPAINTSTS寄存器和USBD_EPAINTEN寄存器做屏蔽，如果相匹配就處理端點A的中斷。
5. 讀取USBD_EPBINTSTS寄存器和USBD_EPBINTEN寄存器做屏蔽，如果相匹配就處理端點B的中斷。

27.5.3 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USBD_SETUP1_0、USBD_SETUP3_2、USBD_SETUP5_4、USBD_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USBD_CEPCTL寄存器NAKCLR位），並且等待狀態完整；否則發送STALL給主機（設置USBD_CEPCTL寄存器STALLEN位）。

27.5.4 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USBD_SETUP1_0、USBD_SETUP3_2、USBD_SETUP5_4、USBD_SETUP7_6寄存器，獲得請求和相關參數。
 - (1) 從USBD_SETUP1_0寄存器低字節獲得bmRequestType。
 - (2) 從USBD_SETUP1_0寄存器高字節獲得bRequest。
 - (3) 從USBD_SETUP3_2寄存器獲得wValue。
 - (4) 從USBD_SETUP5_4寄存器獲得wIndex。
 - (5) 從USBD_SETUP7_6寄存器獲得wLength。
3. 從wValue獲得地址。
4. 清除NAK（設置USBD_CEPCTL寄存器NAKCLR位）。
5. USBD_CEPINTSTS寄存器STSDONEIF位填1清除狀態完成中斷，USBD_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USBD_CEPINTEN寄存器SETUPPKIEN位，始能安裝包中斷。
 - (2) 填寫地址到USBD_FADDR寄存器。

- (3) USBD_CEPINTSTS寄存器STSDONEIF位填1清除狀態完成中斷。

27.5.5 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USBD_SETUP1_0、USBD_SETUP3_2、USBD_SETUP5_4、USBD_SETUP7_6寄存器，獲得請求和相關參數。
 - (1) 從USBD_SETUP1_0寄存器低字節獲得bmRequestType。
 - (2) 從USBD_SETUP1_0寄存器高字節獲得bRequest。
 - (3) 從USBD_SETUP3_2寄存器獲得wValue。
 - (4) 從USBD_SETUP5_4寄存器獲得wIndex。
 - (5) 從USBD_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. USBD_CEPINTSTS寄存器STSDONEIF和INTKIF位填1清除中斷，USBD_CEPINTEN寄存器STSDONEIEN和INTKIEN位填1始能中斷。
6. 等待IN-token中斷。
 - (1) USBD_CEPINTSTS 寄存器 STSDONEIF 和 TXPKIF 位 填 1 清 除 中 斷 ， USBD_CEPINTEN寄存器STSDONEIEN和TXPKIEN位填1始能中斷。
 - (2) 描述符的數據填入USBD_CEPDAT寄存器。
 - (3) 描述符的長度填入USBD_CEPTXCNT寄存器，輸出數據。
7. USBD_CEPINTSTS寄存器INTKIF位填1清除中斷。
8. 等待TX中斷。
 - (1) USBD_CEPINTSTS寄存器STSDONEIF和TXPKIF位填1清除中斷。
 - (2) 清除NAK（設置USBD_CEPCTL寄存器NAKCLR位）。
 - (3) USBD_CEPINTSTS寄存器STSDONEIF位填1清除狀態完成中斷。
 - (4) USBD_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USBD_CEPINTEN寄存器SETUPPKIEN位，始能安裝包中斷。
 - (2) USBD_CEPINTSTS寄存器STSDONEIF位填1清除狀態完成中斷。

27.5.6 IN 傳輸

USBD控制器透過DMA處理 IN 傳輸：

1. 設置USBD_DMACTL寄存器DMARD位為1，填寫端點號到USBD_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USBD_EPxINTEN寄存器TXPKIEN位，始能數據包傳輸中斷。
4. 檢查FIFO是否清空（USBD_EPxINTSTS寄存器BUFEMPTYIF位為1）。
5. 設置USBD_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USBD_DMAADDR寄存器。
7. 填寫傳輸計數到USBD_DMACNT寄存器。
8. 設置USBD_DMACTL寄存器DMAEN位，觸發DMA。
9. 等待DMA完成中斷發生。
 - (1) 設置USBD_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USBD_EPxRSPCTL寄存器SHORTTXEN位，輸出最後數據。

27.5.7 OUT 傳輸

USBD控制器透過DMA處理 OUT 傳輸：

1. 設置USBD_DMACTL寄存器DMARD位為0，填寫端點號到USBD_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USBD_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USBD_DMAADDR寄存器。
5. 填寫傳輸計數到USBD_DMACNT寄存器。
6. 設置USBD_DMACTL寄存器DMAEN位，觸發DMA。
7. 等待DMA完成中斷發生。
 - (1) 設置USBD_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USBD_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USBD_EPxINTEN寄存器RXPKIEN位，禁止接收數據。

28 USB 主機控制器

28.1 概述

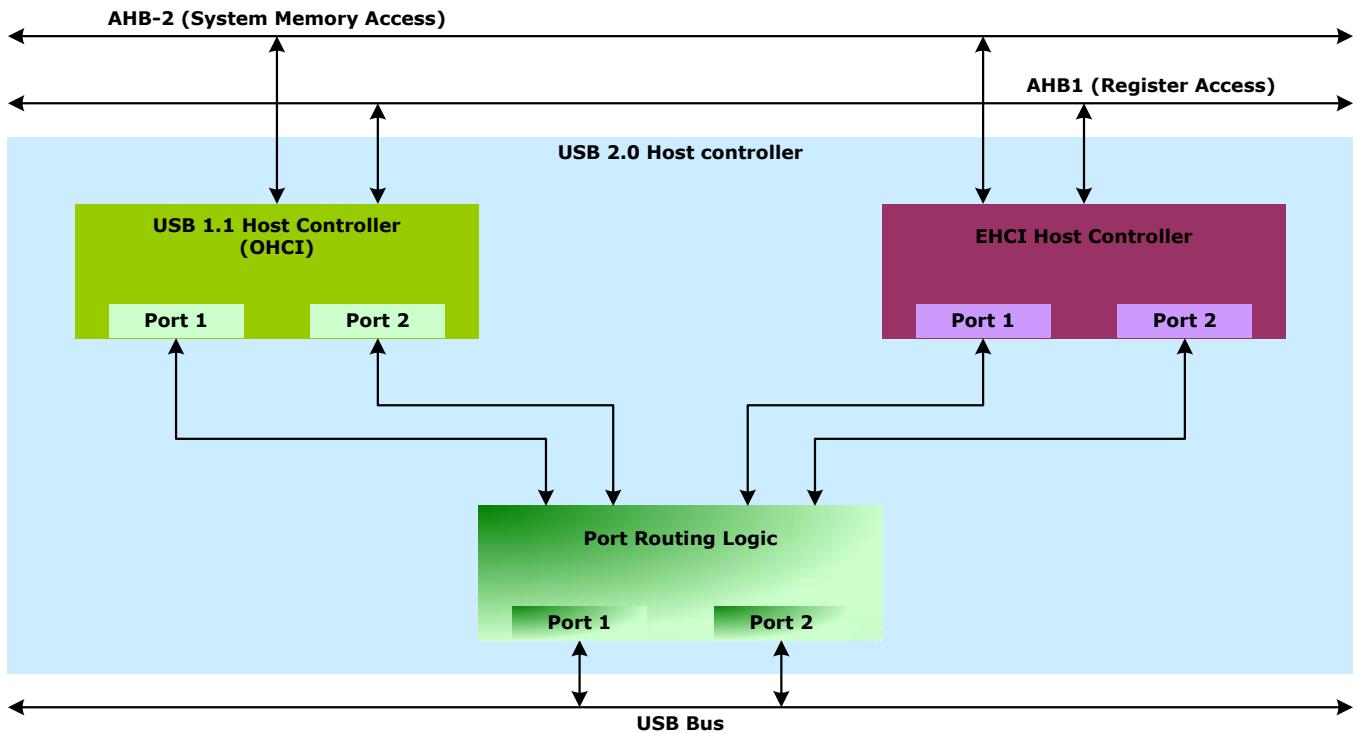
通用串行總線（USB）是一種快速，雙向，同步的，低成本的，動態附接的串行接口標準，用於調製解調器，掃描儀，個人數字助理，鍵盤，鼠標，以及數字成像設備等應用。USB 使用四線串行總線，在主機控制器和外圍設備連接網之間進行串行數據交換。連接的外圍設備通過主機調度，基於 token 封包協議操作，彼此共享 USB 帶寬。外圍設備均可動態地附接，設置，使用或拔除（支持熱插拔），同時間主機控制器與其他保持連接外圍設備間不受影響，仍可繼續正常操作。

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。

28.2 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。
- 內置的 DMA 實時數據傳輸。

28.3 方塊圖



28.3.1 基本配置

USB 主機時鐘源是從 PLL 和 USB PHY 而來。在使能 USB 主機之前，用戶需要事先設置好相關的 PLL 配置。設置 USBH (CLK_HCLKEN[18]) 位，設定 USB 主機時鐘和 4 位預分 USB_N (CLK_DIVCTL2[11 : 8]) 來生成正確的48 MHz 時鐘給 USB 主機。此外，USB 主機需要從 USB PHY 時鐘源，用來支持 USB 2.0 高速運行所需。用戶設置 SUSPEND (USBPCR0[8]和 USBPCR1[8]) 位，以使能 USB PHY。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。

28.3.2 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。

28.3.3 OHCI 控制器

28.3.3.1 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。

28.3.3.2 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。

28.3.3.3 AHB 從機

對主機控制器的配置是通過從機接口。

28.3.3.4 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。

28.3.3.5 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。

28.3.3.6 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。

28.3.3.7 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。

28.3.3.8 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。

28.3.3.9 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。

28.3.3.10 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。

28.3.3.11 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。

28.4 寄存器

Register	Offset	R/W	Description	Reset Value
EHCI Registers (EHCI_BA = 0xB000_5000)				
EHCVNR	EHCI_BA+0x000	R	EHCI Version Number Register	0x0095_0020
EHCSPR	EHCI_BA+0x004	R	EHCI Structural Parameters Register	0x0000_0012
EHCCPR	EHCI_BA+0x008	R	EHCI Capability Parameters Register	0x0000_0000
UCMDR	EHCI_BA+0x020	R/W	USB Command Register	0x0008_0000
USTSR	EHCI_BA+0x024	R/W	USB Status Register	0x0000_1004
UIENR	EHCI_BA+0x028	R/W	USB Interrupt Enable Register	0x0000_0000
UFINDR	EHCI_BA+0x02C	R/W	USB Frame Index Register	0x0000_0000
UPFLBAR	EHCI_BA+0x034	R/W	USB Periodic Frame List Base Address Register	0x0000_0000
UCALAR	EHCI_BA+0x038	R/W	USB Current Asynchronous List Address Register	0x0000_0000

UASSTR	EHCI_BA+0x03C	R/W	USB Asynchronous Schedule Sleep Timer Register	0x0000_0BD6
UCFGR	EHCI_BA+0x060	R/W	USB Configure Flag Register	0x0000_0000
UPSCR0	EHCI_BA+0x064	R/W	USB Port 0 Status and Control Register	0x0000_2000
UPSCR1	EHCI_BA+0x068	R/W	USB Port 1 Status and Control Register	0x0000_2000
USBPCR0	EHCI_BA+0x0C4	R/W	USB PHY 0 Control Register	0x0000_0060
USBPCR1	EHCI_BA+0x0C8	R/W	USB PHY 1 Control Register	0x0000_0020
OHCI Registers (OHCI_BA = 0xB000_7000)				
HcRev	OHCI_BA+0x000	R	Host Controller Revision Register	0x0000_0010
HcControl	OHCI_BA+0x004	R/W	Host Controller Control Register	0x0000_0000
HcComSts	OHCI_BA+0x008	R/W	Host Controller Command Status Register	0x0000_0000
HcIntSts	OHCI_BA+0x00C	R/W	Host Controller Interrupt Status Register	0x0000_0000
HcIntEn	OHCI_BA+0x010	R/W	Host Controller Interrupt Enable Register	0x0000_0000
HcIntDis	OHCI_BA+0x014	R/W	Host Controller Interrupt Disable Register	0x0000_0000
HcHCCA	OHCI_BA+0x018	R/W	Host Controller Communication Area Register	0x0000_0000
HcPerCED	OHCI_BA+0x01C	R/W	Host Controller Period Current ED Register	0x0000_0000
HcCtrHED	OHCI_BA+0x020	R/W	Host Controller Control Head ED Register	0x0000_0000
HcCtrCED	OHCI_BA+0x024	R/W	Host Controller Control Current ED Register	0x0000_0000
HcBlkHED	OHCI_BA+0x028	R/W	Host Controller Bulk Head ED Register	0x0000_0000
HcBlkCED	OHCI_BA+0x02C	R/W	Host Controller Bulk Current ED Register	0x0000_0000
HcDoneH	OHCI_BA+0x030	R/W	Host Controller Done Head Register	0x0000_0000
HcFmIntv	OHCI_BA+0x034	R/W	Host Controller Frame Interval Register	0x0000_2EDF
HcFmRem	OHCI_BA+0x038	R	Host Controller Frame Remaining Register	0x0000_0000
HcFNum	OHCI_BA+0x03C	R	Host Controller Frame Number Register	0x0000_0000
HcPerSt	OHCI_BA+0x040	R/W	Host Controller Periodic Start Register	0x0000_0000
HcLSTH	OHCI_BA+0x044	R/W	Host Controller Low Speed Threshold Register	0x0000_0628
HcRhDeA	OHCI_BA+0x048	R/W	Host Controller Root Hub Descriptor A Register	0x0100_0002
HcRhDeB	OHCI_BA+0x04C	R/W	Host Controller Root Hub Descriptor B Register	0x0000_0000
HcRhSts	OHCI_BA+0x050	R/W	Host Controller Root Hub Status Register	0x0000_0000
HcRhPrt1	OHCI_BA+0x054	R/W	Host Controller Root Hub Port Status [1]	0x0000_0000
HcRhPrt2	OHCI_BA+0x058	R/W	Host Controller Root Hub Port Status [2]	0x0000_0000
OHCI USB Configuration Register				
OpModEn	OHCI_BA+0x204	R/W	USB Operational Mode Enable Register	0X0000_0000

28.5 功能描述

28.5.1 初始設置

要令 USB 主機運作，必須正確執行下列的操作：

- 使能 USB 主機控制器的時鐘源。將 USBH (CLK_HCLKEN[18]) 位設置為 1。
- 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。
- 設置多功能腳位，將 PE.14 及 PE.15 設置為 USBH_PPWR0 及 USBH_PPWR1。
- 對 OHCI 主機控制器進行初始化。
- 對 EHCI 主機控制器進行初始化。

28.5.2 根集線器端口路由

NUC970 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。

28.5.3 OHCI

NUC970 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 NUC970 平台上執行。

28.5.3.1 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。

28.5.3.2 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：

	3 1	2 6	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 7	0 6	0 5	0 4	0 3	0 2	0 1	0 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	下一個端點描述符 (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 HcHCCA 寄存器。

28.5.3.3 傳輸描述符

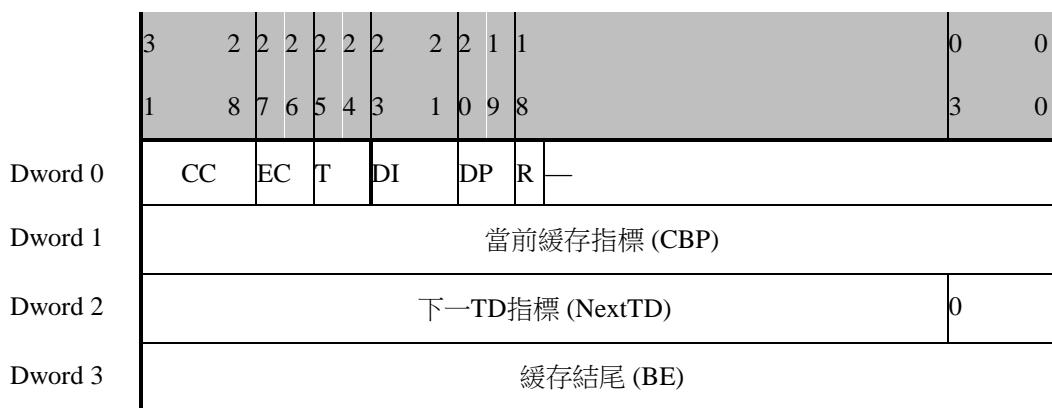
ED 用於描述 USB 端點的特性，僅憑 ED 本身並不足以使主機控制器對 USB 總線進行任何數據傳輸。OHCI 採用傳輸描述符（TD），用來描述一個 USB 數據傳輸的細節，包括了將被用於傳輸的數據緩存的實體位址，目標 USB 設備的裝置地址，目標端點的端點地址，傳輸的類型，以及傳輸的方向等。

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



Isochronous Transfer Descriptor

	3	2	2	2	2	2	2	1	1	1	1	0	0	0
	1	8	7	6	4	3	1	0	6	5	2	1	5	4
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						

28.5.3.4 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	HccaInterruptTable	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	<p>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。</p> <p>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。</p> <p>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。</p>
0x88	116	reserved	保留給主機控制器使用

28.5.3.5 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。（NUC970 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。

28.5.4 EHCI

NUC970 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 NUC970 平台上執行。

28.5.4.1 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)

28.5.4.2 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			

28.5.4.3 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

28.5.4.4 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						

28.5.4.5 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。

28.5.4.6 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)

28.5.4.7 中斷處理

- 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。

- 帖列表翻轉中斷

當帖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])。發生帖列表翻轉的確切指數值，則取決於帖列表的大小。

- 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。

28.5.4.8 根集線器

NUC970 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

29 圖像擷取接口 (Capture Sensor Interface Controller)

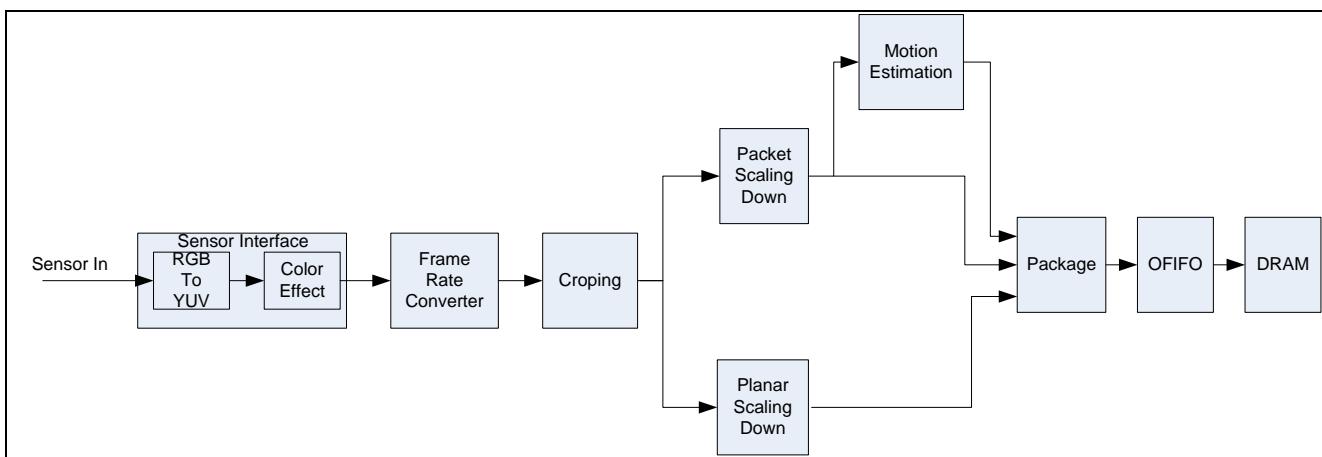
29.1 概述

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。

29.2 特性

- 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

29.3 方塊圖



29.4 寄存器

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

Register	Offset	R/W	Description	Reset Value
----------	--------	-----	-------------	-------------

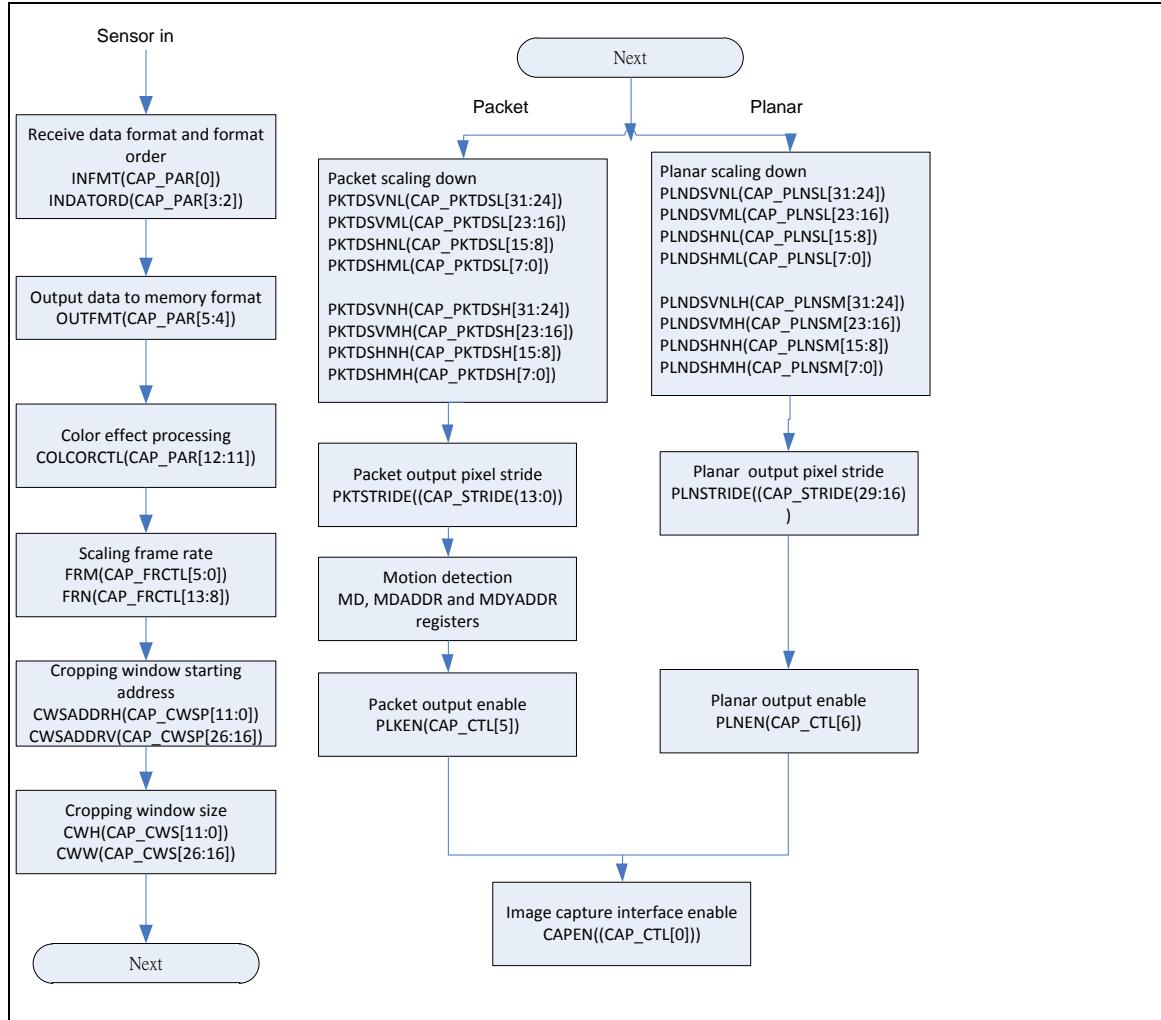
Capture Base Address: CAP_BA = 0xB000_E000				
CAP_CTL	CAP_BA+0x00	R/W	Image Capture Interface Control Register	0x0000_0040
CAP_PAR	CAP_BA+0x04	R/W	Image Capture Interface Parameter Register	0x0000_0000
CAP_INT	CAP_BA+0x08	R/W	Image Capture Interface Interrupt Register	0x0000_0000
CAP_POSTERIZE	CAP_BA+0x0C	R/W	YUV Component Posterizing Factor Register	0x0000_0000
CAP_MD	CAP_BA+0x10	R/W	Motion Detection Register	0x0000_0000
CAP_MDADDR	CAP_BA+0x14	R/W	Motion Detection Output Address Register	0x0000_0000
CAP_MDYADDR	CAP_BA+0x18	R/W	Motion Detection Temp Y Output Address Register	0x0000_0000
CAP_SEPIA	CAP_BA+0x1C	R/W	Sepia Effect Control Register	0x0000_0000
CAP_CWSP	CAP_BA+0x20	R/W	Cropping Window Starting Address Register	0x0000_0000
CAP_CWS	CAP_BA+0x24	R/W	Cropping Window Size Register	0x0000_0000
CAP_PKTSL	CAP_BA+0x28	R/W	Packet Scaling Vertical/Horizontal Factor Register (LSB)	0x0000_0000
CAP_PLNSL	CAP_BA+0x2C	R/W	Planar Scaling Vertical/Horizontal Factor Register (LSB)	0x0000_0000
CAP_FRCTL	CAP_BA+0x30	R/W	Scaling Frame Rate Factor Register	0x0000_0000
CAP_STRIDE	CAP_BA+0x34	R/W	Frame Output Pixel Stride Width Register	0x0000_0000
CAP_FIFOTH	CAP_BA+0x3C	R/W	FIFO Threshold Register	0x070D_0507
CAP_CMPADDR	CAP_BA+0x40	R/W	Compare Memory Base Address Register	0xFFFF_FFFC
CAP_PKTSM	CAP_BA+0x48	R/W	Packet Scaling Vertical/Horizontal Factor Register (MSB)	0x0000_0000
CAP_PLNSM	CAP_BA+0x4C	R/W	Planar Scaling Vertical/Horizontal Factor Register (MSB)	0x0000_0000
CAP_CURADDRP	CAP_BA+0x50	R	Current Packet System Memory Address Register	0x0000_0000
CAP_CURADDRY	CAP_BA+0x54	R	Current Planar Y System Memory Address Register	0x0000_0000
CAP_CURADDRU	CAP_BA+0x58	R	Current Planar U System Memory Address Register	0x0000_0000
CAP_CURVADDR	CAP_BA+0x5C	R	Current Planar V System Memory Address Register	0x0000_0000
CAP_PKTBA0	CAP_BA+0x60	R/W	System Memory Packet Base Address 0 Register	0x0000_0000
CAP_PKTBA1	CAP_BA+0x64	R/W	System Memory Packet Base Address 1 Register	0x0000_0000
CAP_YBA	CAP_BA+0x80	R/W	System Memory Planar Y Base Address Register	0x0000_0000
CAP_UBA	CAP_BA+0x84	R/W	System Memory Planar U Base Address Register	0x0000_0000
CAP_VBA	CAP_BA+0x88	R/W	System Memory Planar V Base Address Register	0x0000_0000

29.5 功能描述.

29.5.1 基本配置

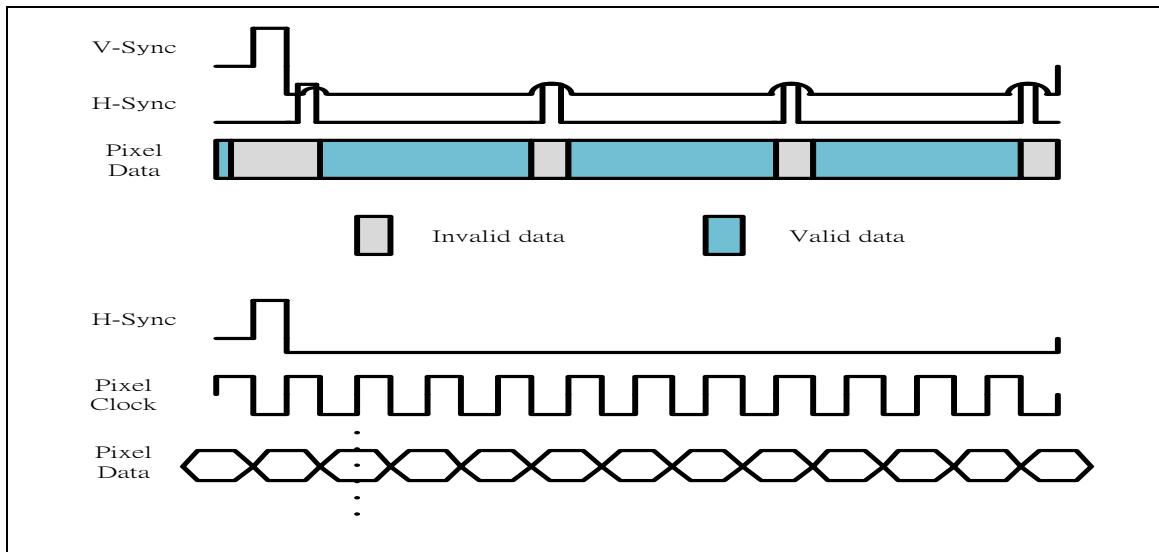
當 CAP 使用之前，必須將 CAPEN(HCLKEN1[26]) 設置為 1。CAP 時鐘來源選擇可以由 SENSOR_S(CLKDIV3[23:16]) 設置，ADC 引擎時鐘分頻器由 SENSOR_N(CLKDIV3[27:24]) 設置。

29.5.2 圖像捕捉流程圖



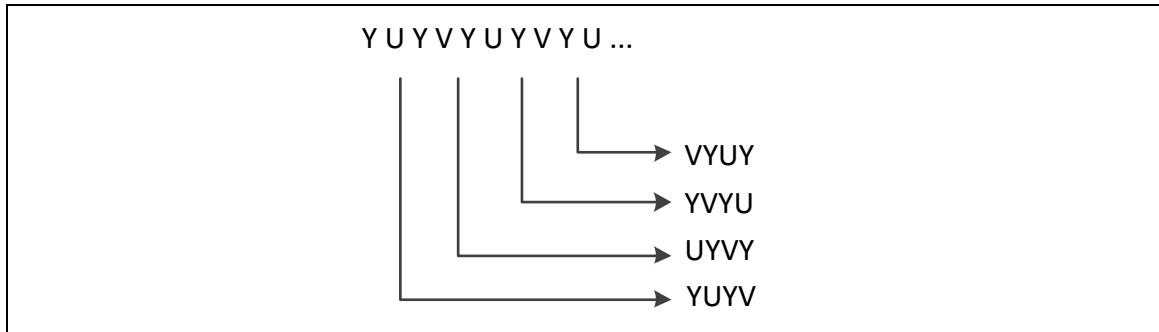
29.5.3 極性和輸入的數據

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。



29.5.4 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])。



29.5.5 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。

29.5.6 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 =

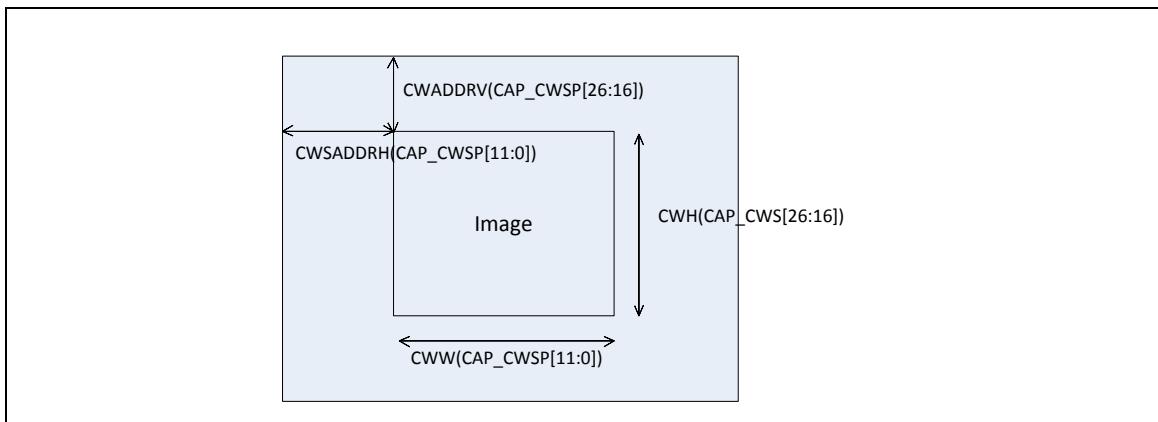
$$352/640 = (\text{PKTSHNH} \ll 8 + \text{PKTSHNL}) / (\text{PKTSHMH} \ll 8 + \text{PKTSHML})$$

The vertical downscale factor =

$$288/480 = (\text{PKTSVNH} << 8 + \text{PKTSVNL}) / (\text{PKTSVMH} << 8 + \text{PKTSVML})$$

29.5.7 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：



29.5.8 快門模式（單張拍攝）

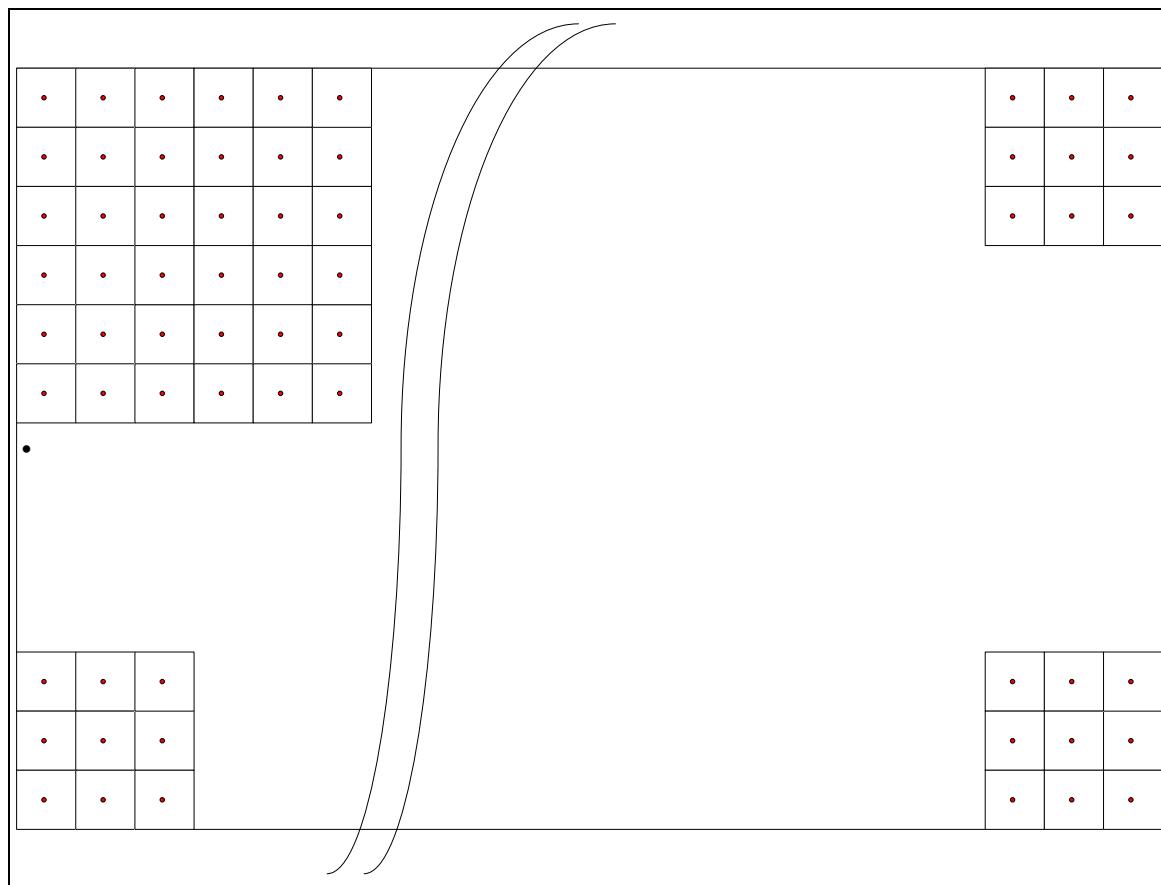
在快門模式下，即SHUTTER (CAP_CTL[16])=1，圖像捕捉接口接收圖像後則會停止捕捉圖像。

29.5.9 運動檢測

該功能用於檢測圖像中有物體移動。功能如下：

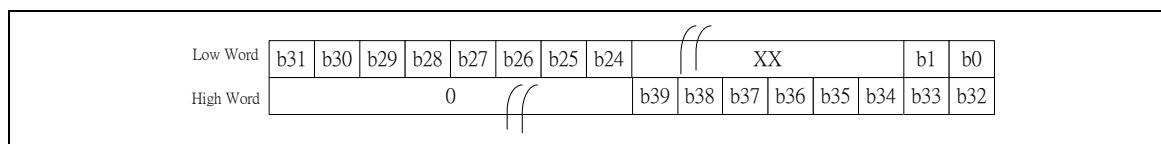
1. 區塊大小支持8x8和16x16
2. 輸出移動偵測支持1位或8位（1位DIFF和7位閾值）

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。

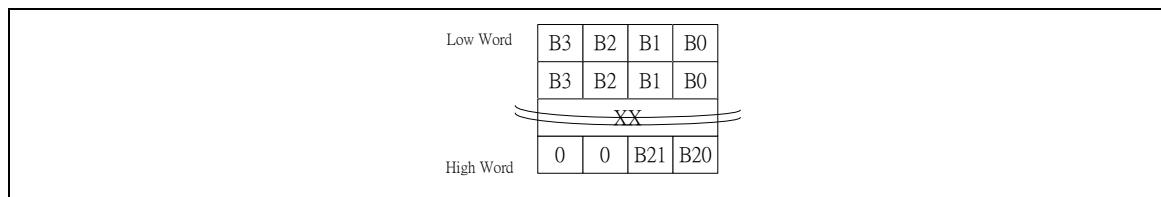


運動檢測輸出緩衝區列表格式如下。

- 一個bit模式（拍攝的寬度=640塊大小 16×16 ）



- 1 bit DIFF (MSB) +7 bit Y的差值（捕獲寬度=352塊大小 16×16 ）



30 看門狗計時器控制器 (WDT)

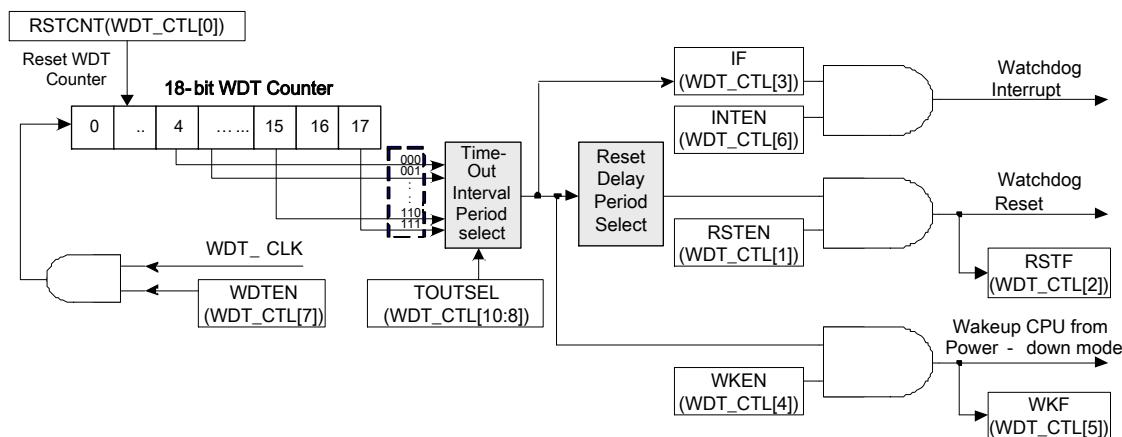
30.1 概述

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。

30.2 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 ($2^4 \sim 2^{18}$)，如果 WDT_CLK = 32.768kHz，則超時間隔為 0.49 mS ~ 8S.
- 復位週期 = $(1 / \text{WDT_CLK}) * 63$
- 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

30.3 方塊圖



30.4 寄存器

Register	Offset	R/W	Description	Reset Value
WDT Base Address:				
WDT_BA = 0xB800_1800				

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

30.5 功能描述

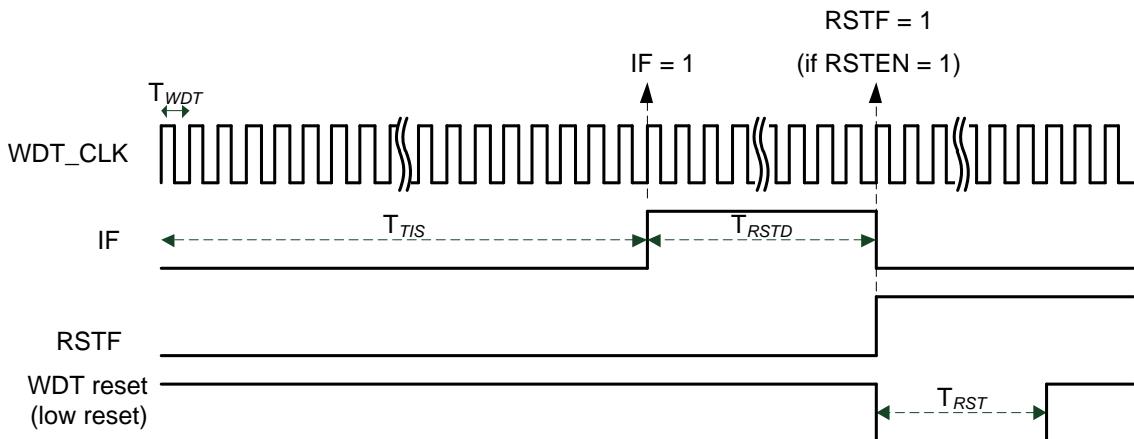
30.5.1 WDT 設置

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/128，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_CTL[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 ($(2^4 \sim 2^{18}) * T_{WDT}$)
- 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.

30.5.2 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.

31 窗口看門狗定時控制器 (WWDT)

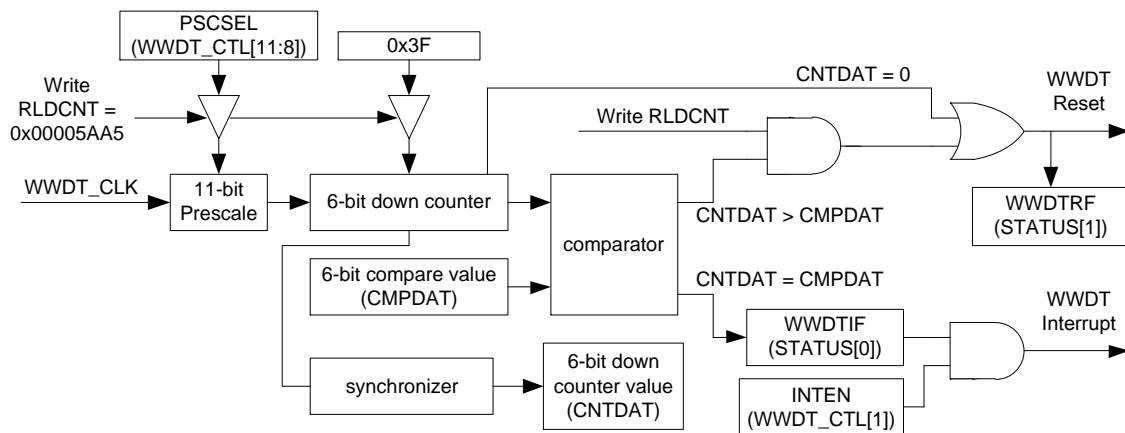
31.1 概述

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。

31.2 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

31.3 方塊圖



31.4 寄存器

Register	Offset	R/W	Description	Reset Value
WWDT Base Address:				
WWDT_BA = 0xB800_1900				
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

31.5 功能描述

31.5.1 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/128, 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。

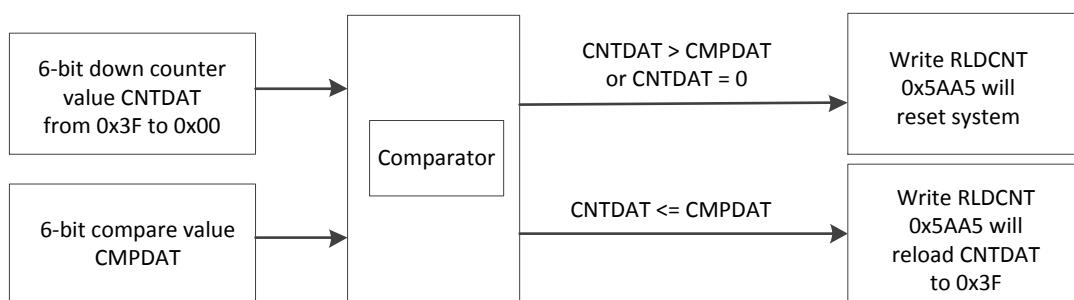
為防止程式跑到非預期代碼禁止視窗看門狗，控制寄存器WWDTCSR和WWDT_IER在晶片上電或重定之後僅能寫一次。一旦視窗看門狗被軟體使能，軟體不能禁止視窗看門狗，改變預分頻週期或改變視窗比較值，除非晶片重定。

31.5.2 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。

31.5.3 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。



31.5.4 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(WWDTCSR[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.

Revision History

Date	Revision	Description
2015.3.31	1.00	1. Initially issued.

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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.