

## NUC970/N9H30 Family Programming Guide

### Document Information

<b>Abstract</b>	This document introduces the control sequence of NUC970/N9H30 family peripherals.
<b>Apply to</b>	NUC970 and N9H30 family.

*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](http://www.nuvoton.com)

---

## Table of Contents

---

<b>1</b>	<b>SYSTEM MANAGER.....</b>	<b>12</b>
1.1	Overview.....	12
1.2	Register Map.....	12
1.3	Functional Description.....	13
1.3.1	Multiple Function Control .....	13
1.3.2	Low Voltage Detect / Reset.....	14
1.3.3	USB ID Detection .....	14
<b>2</b>	<b>CLOCK CONTROLLER.....</b>	<b>15</b>
2.1	Overview.....	15
2.2	Features.....	15
2.3	Block Diagram .....	15
2.4	Register Map.....	17
2.5	Functional Description.....	17
2.5.1	Module Clock On/Off .....	17
2.5.2	Clock Divider.....	18
2.5.3	PLL Setting.....	18
<b>3</b>	<b>ANALOG TO DIGITAL CONVERTER (ADC).....</b>	<b>20</b>
3.1	Overview.....	20
3.2	Features.....	20
3.3	Block Diagram .....	20
3.4	Register Map.....	21
3.5	Functional Description.....	22
3.5.1	Basic Configuration.....	22
3.5.2	ADC Transfer Function.....	22
3.5.3	Normal Detection .....	23
3.5.4	Battery Voltage Detection .....	24
3.5.5	Key Pad Scan .....	25
3.5.6	4-wire and 5-wire Touch Screen.....	27
3.5.7	4-wire Pressure Measurement .....	29
<b>4</b>	<b>ADVANCED INTERRUPT CONTROLLER (AIC).....</b>	<b>31</b>
4.1	Overview.....	31

---

<b>4.2 Features</b> .....	31
<b>4.3 Block Diagram</b> .....	32
<b>4.4 Register Map</b> .....	32
<b>4.5 Functional Description</b> .....	33
4.5.1 Interrupt channel configuration .....	33
4.5.2 Interrupt Masking .....	34
4.5.3 Interrupt Clearing and Setting .....	34
4.5.4 Software Priority Scheme .....	35
4.5.5 Hardware Priority Scheme.....	36
4.5.6 Interrupt Sources.....	37
<b>5 CAN</b> .....	40
<b>5.1 Overview</b> .....	40
<b>5.2 Features</b> .....	40
<b>5.3 Block Diagram</b> .....	40
<b>5.4 Register Map</b> .....	41
<b>5.5 Functional Description</b> .....	43
5.5.1 CAN Protocol .....	43
5.5.2 CAN Baud Rate Setting .....	43
5.5.3 CAN Module Register.....	46
5.5.4 Transfer CAN Message.....	47
5.5.5 Receive CAN Message .....	49
5.5.6 Wakeup Function .....	50
<b>6 CRYPTOGRAPHIC ACCELERATOR</b> .....	51
<b>6.1 Overview</b> .....	51
<b>6.2 Features</b> .....	51
<b>6.3 Block Diagram</b> .....	52
<b>6.4 Register Map</b> .....	52
<b>6.5 Functional Description</b> .....	57
6.5.1 Data Access .....	58
6.5.2 Channel Expansion.....	59
6.5.3 PRNG.....	59
6.5.4 AES .....	60
6.5.5 DES/TDES .....	61
6.5.6 SHA .....	63
<b>7 EXTERNAL BUS INTERFACE (EBI)</b> .....	66
<b>7.1 Overview</b> .....	66
<b>7.2 Features</b> .....	66
<b>7.3 Block Diagram</b> .....	66

<b>7.4 Register Map</b> .....	67
<b>7.5 Functional Description</b> .....	67
7.5.1 Basic Configuration.....	67
7.5.2 Memory Space and Control.....	67
<b>8 ETHERNET MAC CONTROLLER (EMAC)</b> .....	69
<b>8.1 Overview</b> .....	69
<b>8.2 Features</b> .....	69
<b>8.3 Block Diagram</b> .....	69
<b>8.4 Register Map</b> .....	70
<b>8.5 Functional Description</b> .....	72
8.5.1 PHY Control .....	72
8.5.2 CAM Configuration .....	74
8.5.3 Control Frame .....	74
8.5.4 Wake on Lan (WoL).....	75
8.5.5 Packet Receive.....	75
8.5.6 Packet Transmit .....	80
8.5.7 Network Timing.....	86
8.5.8 Error Handling.....	90
<b>9 ENHANCED TIMER CONTROLLER (ETMR)</b> .....	91
<b>9.1 Overview</b> .....	91
<b>9.2 Features</b> .....	91
<b>9.3 Block Diagram</b> .....	91
<b>9.4 Register Map</b> .....	91
<b>9.5 Functional Description</b> .....	92
9.5.1 Timer Initialization .....	92
9.5.2 Timer Capture Initialization .....	92
9.5.3 Interrupt Handling.....	93
9.5.4 Timer Frequency .....	93
9.5.5 One-Shot Mode .....	94
9.5.6 Periodic Mode.....	94
9.5.7 Toggle Mode .....	95
9.5.8 Continuous Mode .....	95
9.5.9 Free Counting Mode .....	96
9.5.10 Trigger Counting Mode .....	97
9.5.11 Counter Reset Mode .....	98
9.5.12 Capture Debounce .....	99
<b>10 FLASH MEMORY INTERFACE</b> .....	100
<b>10.1 Overview</b> .....	100
<b>10.2 Features</b> .....	100

<b>10.3 Block Diagram .....</b>	<b>100</b>
<b>10.4 Register Map.....</b>	<b>101</b>
<b>10.5 Functional Description.....</b>	<b>103</b>
10.5.1    DMA and FMI Global Control.....	103
10.5.2    NAND Flash.....	103
10.5.3    eMMC.....	107
 <b>11 GENERAL DMA CONTROLLER (GDMA) .....</b>	 <b>112</b>
<b>11.1 Overview.....</b>	<b>112</b>
<b>11.2 Features.....</b>	<b>112</b>
<b>11.3 Block Diagram .....</b>	<b>112</b>
<b>11.4 Register Map.....</b>	<b>113</b>
<b>11.5 Functional Description.....</b>	<b>114</b>
11.5.1    Non-Descriptor Functional Descriptions.....	114
11.5.2    Descriptor Functional Descriptions .....	118
 <b>12 2D GRAPHIC ENGINE (GE2D).....</b>	 <b>122</b>
<b>12.1 Overview.....</b>	<b>122</b>
<b>12.2 Features.....</b>	<b>122</b>
<b>12.3 Block Diagram .....</b>	<b>123</b>
<b>12.4 Register Map.....</b>	<b>123</b>
<b>12.5 Function Description.....</b>	<b>124</b>
12.5.1    2D Graphic Engine Initialization .....	124
12.5.2    Ternary Raster Operations (ROP) .....	126
12.5.3    Bit Block Transfer (BitBLT).....	127
12.5.4    Bresenham Line Drawing .....	131
12.5.5 $\alpha$ Blending.....	135
12.5.6    Clipping .....	136
12.5.7    Rotation.....	137
12.5.8    Scale Up/Down .....	138
 <b>13 GENERAL-PURPOSE INPUT/OUTPUT (GPIO) .....</b>	 <b>141</b>
<b>13.1 Overview.....</b>	<b>141</b>
<b>13.2 Features.....</b>	<b>141</b>
<b>13.3 Block Diagram .....</b>	<b>141</b>
<b>13.4 Register Map.....</b>	<b>142</b>
<b>13.5 Functional Description.....</b>	<b>146</b>
13.5.1    Multiple function pin Configuration.....	146
13.5.2    GPIO Output Mode.....	147
13.5.3    GPIO Input Mode.....	147
13.5.4    GPIO Interrupt.....	148

---

<b>14 I<sup>2</sup>C .....</b>	<b>150</b>
<b>14.1 Overview.....</b>	<b>150</b>
<b>14.2 Features.....</b>	<b>150</b>
<b>14.3 Function Block .....</b>	<b>150</b>
<b>14.4 Register Map.....</b>	<b>151</b>
<b>14.5 Function Description.....</b>	<b>151</b>
14.5.1 I <sup>2</sup> C Protocol.....	151
14.5.2 Data Transmission Continuously .....	152
14.5.3 Interrupt.....	152
14.5.4 Software Mode.....	152
14.5.5 I <sup>2</sup> C Operation Using CMDR Register.....	153
14.5.6 I <sup>2</sup> C EEPROM Operation Example.....	154
<b>15 I<sup>2</sup>S.....</b>	<b>156</b>
<b>15.1 Overview.....</b>	<b>156</b>
<b>15.2 Features.....</b>	<b>156</b>
<b>15.3 Function Block .....</b>	<b>156</b>
<b>15.4 Register Map.....</b>	<b>157</b>
<b>15.5 Functional Description.....</b>	<b>158</b>
15.5.1 I <sup>2</sup> S Master/Slave Mode .....	158
15.5.2 I <sup>2</sup> S Source Clock Configuration .....	158
15.5.3 I <sup>2</sup> S Calculation and Configuration of Clock.....	159
15.5.4 DMA .....	159
15.5.5 Sequence of DMA Data.....	161
15.5.6 Interface Selection.....	161
15.5.7 PCM Interface .....	162
15.5.8 Data Split .....	163
<b>16 JPEG CODEC.....</b>	<b>164</b>
<b>16.1 Overview.....</b>	<b>164</b>
<b>16.2 Feature.....</b>	<b>164</b>
<b>16.3 Block Diagram .....</b>	<b>164</b>
<b>16.4 Register Map.....</b>	<b>165</b>
<b>16.5 Functional Description.....</b>	<b>167</b>
16.5.1 Memory Access .....	167
16.5.2 JPEG Encoding.....	169
16.5.3 Normal Encoding .....	169
16.5.4 Encoding Scaling up .....	170
16.5.5 JPEG Decoding .....	173

---

---

<b>17 KEYPAD INTERFACE .....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>
<b>17.1 Overview.....</b>	.....Error! Bookmark not defined.
<b>17.2 Features.....</b>	.....Error! Bookmark not defined.
<b>17.3 Block Diagram .....</b>	.....Error! Bookmark not defined.
<b>17.4 Register Map.....</b>	.....Error! Bookmark not defined.
<b>17.5 Functional Description.....</b>	.....Error! Bookmark not defined.
17.5.1     Keypad Controller Configuration .....	.....Error! Bookmark not defined.
17.5.2     Wake UP Function .....	.....Error! Bookmark not defined.
<b>18 LCD DISPLAY INTERFACE CONTROLLER (LCM) .....</b>	<b>181</b>
<b>18.1 Overview.....</b>	<b>181</b>
<b>18.2 Features.....</b>	<b>181</b>
<b>18.3 Block Diagram .....</b>	<b>182</b>
<b>18.4 Register Map.....</b>	<b>182</b>
<b>18.5 Functional Description.....</b>	<b>183</b>
18.5.1     LCD Configuration Flow .....	.....183
18.5.2     LCD Controller Initialization and Configuration .....	.....186
18.5.3     Configure OSD Controller .....	.....188
18.5.4     Hardware Cursor .....	.....189
<b>19 MTP CONTROLLER .....</b>	<b>191</b>
<b>19.1 Overview.....</b>	<b>191</b>
<b>19.2 Features.....</b>	<b>191</b>
<b>19.3 Block Diagram .....</b>	<b>191</b>
<b>19.4 Register Map.....</b>	<b>192</b>
<b>19.5 Functional Description.....</b>	<b>193</b>
19.5.1     Use MTP Controller .....	.....193
19.5.2     MTP Key .....	.....193
19.5.3     User Defined Data.....	.....194
19.5.4     MTP Enable.....	.....195
19.5.5     Program MTP Key .....	.....195
19.5.6     Lock MTP Key.....	.....196
19.5.7     MTP Key for AES Encrypt/Decrypt .....	.....197
19.5.8     MTP Key for SHA/HMAC Comparison .....	.....198
<b>20 PULSE WIDTH MODULATION (PWM) .....</b>	<b>199</b>
<b>20.1 Overview.....</b>	<b>199</b>
<b>20.2 Features.....</b>	<b>199</b>
<b>20.3 Block Diagram .....</b>	<b>199</b>

---

<b>20.4 Register Map.....</b>	<b>200</b>
<b>20.5 Functional Description.....</b>	<b>201</b>
20.5.1    PWM Timer Operation .....	201
20.5.2    PWM double buffer.....	202
20.5.3    Periodic and One-Shot Operation.....	203
20.5.4    Dead-Zone Generator.....	203
20.5.5    PWM Timer Start Procedure.....	204
20.5.6    PWM Timer Stop Procedure .....	205
<b>21 REAL TIME CLOCK (RTC).....</b>	<b>206</b>
<b>21.1 Overview.....</b>	<b>206</b>
<b>21.2 Features.....</b>	<b>206</b>
<b>21.3 Block Diagram .....</b>	<b>206</b>
<b>21.4 Register Map.....</b>	<b>207</b>
<b>21.5 Functional Description.....</b>	<b>208</b>
21.5.1    RTC Initiation.....	208
21.5.2    RTC write enable .....	208
21.5.3    12/24 hour Time scale selection.....	209
21.5.4    Set Calendar and Time.....	209
21.5.5    Set Calendar and Time Alarm (Absolute) .....	210
21.5.6    Set Time Alarm (Relative) .....	211
21.5.7    Set wake-up function.....	212
21.5.8    Set tick interrupt.....	214
21.5.9    System Power Control Flow.....	215
21.5.10    Frequency Compensation: .....	220
<b>22 SMART CARD HOST INTERFACE (SC).....</b>	<b>222</b>
<b>22.1 Overview.....</b>	<b>222</b>
<b>22.2 Features.....</b>	<b>222</b>
<b>22.3 Block Diagram .....</b>	<b>222</b>
<b>22.4 Register Map.....</b>	<b>223</b>
<b>22.5 Functional Description.....</b>	<b>224</b>
22.5.1    Activation (Cold Reset) .....	225
22.5.2    Warm Reset.....	226
22.5.3    Deactivation.....	228
22.5.4    Data Format.....	228
22.5.5    Data Transfer .....	229
22.5.6    Error Signal and Character Repetition .....	230
22.5.7    Internal Time-out Counter .....	231
22.5.8    Smartcard Insert/Remove Detection .....	232
22.5.9    Miscellaneous Transmission Settings .....	233
22.5.10    UART Mode.....	233
<b>23 SECURE DIGITAL HOST CONTROLLER (SDH).....</b>	<b>235</b>

---

<b>23.1 Overview.....</b>	<b>235</b>
<b>23.2 Features.....</b>	<b>235</b>
<b>23.3 Block Diagram .....</b>	<b>235</b>
<b>23.4 Register Map.....</b>	<b>236</b>
<b>23.5 Functional Description.....</b>	<b>236</b>
23.5.1    Global Control .....	238
23.5.2    Send Command.....	239
23.5.3    Get Response .....	239
23.5.4    Read SD Card.....	240
23.5.5    Write SD Card.....	240
<b>24 SPI .....</b>	<b>242</b>
<b>24.1 Overview.....</b>	<b>242</b>
<b>24.2 Features.....</b>	<b>242</b>
<b>24.3 Function Block .....</b>	<b>242</b>
<b>24.4 Register Map.....</b>	<b>242</b>
<b>24.5 Function Description.....</b>	<b>243</b>
24.5.1    Slave Selection .....	243
24.5.2    Automatic Slave Select.....	244
24.5.3    Dual / Quad Mode .....	244
24.5.4    Burst Mode .....	247
24.5.5    SPI Interrupt .....	247
24.5.6    SPI Programming Example.....	247
<b>25 TIMER CONTROLLER.....</b>	<b>249</b>
<b>25.1 Overview.....</b>	<b>249</b>
<b>25.2 Features.....</b>	<b>249</b>
<b>25.3 Block Diagram .....</b>	<b>249</b>
<b>25.4 Register Map.....</b>	<b>250</b>
<b>25.5 Functional Description.....</b>	<b>250</b>
25.5.1    Timer Initialization.....	250
25.5.2    Interrupt Handling .....	251
25.5.3    Timeout Frequency .....	251
25.5.4    One-shot Mode .....	252
25.5.5    Periodic Mode .....	252
25.5.6    Continuous Mode .....	252
<b>26 UART .....</b>	<b>254</b>
<b>26.1 Overview.....</b>	<b>254</b>
<b>26.2 Features.....</b>	<b>256</b>

---

<b>26.3 Block Diagram .....</b>	<b>256</b>
<b>26.4 Register Map.....</b>	<b>257</b>
<b>26.5 Functional Description.....</b>	<b>258</b>
26.5.1   Initializations .....	258
26.5.2   IrDA Mode.....	259
26.5.3   RS485 Function Mode .....	260
26.5.4   LIN (Local Interconnection Network) Mode .....	262
<b>27 USB 2.0 DEVICE CONTROLLER .....</b>	<b>264</b>
27.1 Overview.....	264
27.2 Features .....	264
27.3 Block Diagram .....	264
27.4 Register Map.....	265
27.5 Functional Description.....	269
27.5.1   Initialize .....	270
27.5.2   Interrupt Service Routine.....	271
27.5.3   Standard Request.....	271
27.5.4   Set Address Request .....	271
27.5.5   Get Descriptor .....	272
27.5.6   IN Transmission.....	273
27.5.7   OUT Transmission .....	273
<b>28 USB HOST CONTROLLER.....</b>	<b>275</b>
28.1 Overview.....	275
28.2 Features .....	275
28.3 Block Diagram .....	275
28.3.1   Basic Configuration .....	276
28.3.2   EHCI Controller.....	276
28.3.3   OHCI Controller .....	277
28.4 Register Map.....	278
28.5 Functional Description.....	281
28.5.1   Initialization .....	281
28.5.2   Root Hub Port Routing Logic .....	281
28.5.3   OHCI.....	281
28.5.4   EHCI .....	289
<b>29 CAPTURE SENSOR INTERFACE CONTROLLER .....</b>	<b>298</b>
29.1 Overview.....	298
29.2 Features .....	298
29.3 Block Diagram .....	298
29.4 Register Map.....	298

---

<b>29.5 Functional Description.....</b>	<b>299</b>
29.5.1 Basic Configuration .....	299
29.5.2 Image Capture Flow Chart .....	300
29.5.3 Polarity and Input Data Order .....	300
29.5.4 Sensor Data Input Order .....	301
29.5.5 Input and Output Data Format.....	301
29.5.6 Downscale Factor.....	301
29.5.7 Cropping Window and Start Position.....	302
29.5.8 One Shutter Mode (Single Frame).....	302
29.5.9 Motion detection .....	302
<b>30 WATCHDOG TIMER (WDT).....</b>	<b>304</b>
30.1 Overview.....	304
30.2 Features.....	304
30.3 Block Diagram .....	304
30.4 Register Map.....	304
30.5 Functional Description.....	305
30.5.1 WDT Configuration.....	305
30.5.2 WDT Wakeup.....	306
<b>31 WINDOW WATCHDOG TIMER (WWDT).....</b>	<b>307</b>
31.1 Overview.....	307
31.2 Features.....	307
31.3 Block Diagram .....	307
31.4 Register Map.....	307
31.5 Function Description.....	308
31.5.1 Timeout Setting.....	308
31.5.2 WWDT Interrupt.....	309
31.5.3 System Reset.....	309
31.5.4 WWDT Window Setting Limitations .....	309

---

# 1 System Manager

## 1.1 Overview

The system management describes following information and functions.

- System Resets
- System Memory Map
- System management registers for Product Identifier (PDID), Power-On Setting, System Wake-Up, Reset Control for on-chip controllers/peripherals, and multi-function pin control.
- System Control registers

## 1.2 Register Map

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

Register	Offset	R/W	Description	Reset Value
<b>SYS_BA = 0xB000_0000</b>				
SYS_P DID	SYS_BA+0x000	R	Product Identifier Register	0x0X30_D008 <sup>[1]</sup>
SYS_PWRON	SYS_BA+0x004	R/W	Power-On Setting Register	Undefined <sup>[2]</sup>
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

<b>SYS_GPC_MFPL</b>	SYS_BA+0x080	R/W	GPIOC Low Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPC_MFPH</b>	SYS_BA+0x084	R/W	GPIOC High Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPD_MFPL</b>	SYS_BA+0x088	R/W	GPIOD Low Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPD_MFPH</b>	SYS_BA+0x08C	R/W	GPIOD High Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPE_MFPL</b>	SYS_BA+0x090	R/W	GPIOE Low Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPE_MFPH</b>	SYS_BA+0x094	R/W	GPIOE High Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPF_MFPL</b>	SYS_BA+0x098	R/W	GPIOF Low Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPF_MFPH</b>	SYS_BA+0x09C	R/W	GPIOF High Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPG_MFPL</b>	SYS_BA+0x0A0	R/W	GPIOG Low Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPG_MFPH</b>	SYS_BA+0x0A4	R/W	GPIOG High Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPH_MFPL</b>	SYS_BA+0x0A8	R/W	GPIOH Low Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPH_MFPH</b>	SYS_BA+0x0AC	R/W	GPIOH High Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPI_MFPL</b>	SYS_BA+0x0B0	R/W	GPIOI low Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPI_MFPH</b>	SYS_BA+0x0B4	R/W	GPIOI High Byte Multiple Function Control Register	0x0000_0000
<b>SYS_GPJ_MFPL</b>	SYS_BA+0x0B8	R/W	GPIOJ low Byte Multiple Function Control Register	Undefined <sup>[2]</sup>
<b>SYS_DDR_DSCTL</b>	SYS_BA+0x0F0	R/W	DDR I/O Driving Strength Control Register	0x0000_0000
<b>SYS_PORDISCR</b>	SYS_BA+0x100	R/W	Power-On-Reset Disable Control Register	0x0000_00XX
<b>SYS_ICEDBGCR</b>	SYS_BA+0x104	R/W	ICE Debug Interface Control Register	0x0000_0001
<b>SYS_REGWPCTL</b>	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 Functional Description

### 1.3.1 Multiple Function Control

Each Module should set the multiple function control before starting it. Such as SPI0, user should set the GPB6~9 for SPI0 before starting it. GPIO6 is for SPI0\_SS0, GPIO7 is for SPI0\_CLK, GPIO8 is for SPI0\_DATAO, and GPIO9 is for SPI0\_DATAL. Therefore, fill 0xBB000000 into SYS\_GPB\_MFPL register and 0xBB to SYS\_GPB\_MFPH register to do this setting. (For each pin's setting, please reference NUC970/N9H30 Technical Reference Manual).

### 1.3.2 Low Voltage Detect / Reset

When voltage is lower than 2.6V or 2.8V, SYS\_MISCISR register LVD\_IS bit will be set. If the interrupt is enabled, the interrupt will occur. When the voltage rises from 2.3V to 2.4V, the system will reset.

Low voltage reset or detection, please follow the steps below.

1. Set SYS\_LVRDCR register LVD\_SEL bit to select the detect voltage – 2.6V or 2.8V.
2. Detect. Enable SYS\_LVRDCR register LVD\_EN bit.
3. Detect and reset. Enable SYS\_LVRDCR register LVD\_EN and LVR\_EN bit.
4. Active interrupt. Enable SYS\_MISCIER register LVD\_EN bit.
5. Confirm interrupt status. Checking SYS\_MISCISR register LVD\_IS bit.
6. Clear SYS\_MISCISR register LVD\_IS bit.
7. Repeat steps 2~6.

### 1.3.3 USB ID Detection

USB Host and USB Device share one port. When A-type cable (Host) plug in, user can detect it and run the Host program. Otherwise, when B-type cable (Device) plug in, user can run the Device program.

The steps as below:

1. Enable SYS\_MISCIER register USBIDC\_IEN bit.
2. Cable plug in, interrupt occurred, checking SYS\_MISCISR register USBIDC\_IS bit.
3. Checking SYS\_MISCISR register USB0\_IDS bit. 1 is USB Host connect; 0 is USB Device connect.
4. Clear SYS\_MISCISR register USBIDC\_IS bit.
5. Repeat steps 2~4.

## 2 Clock Controller

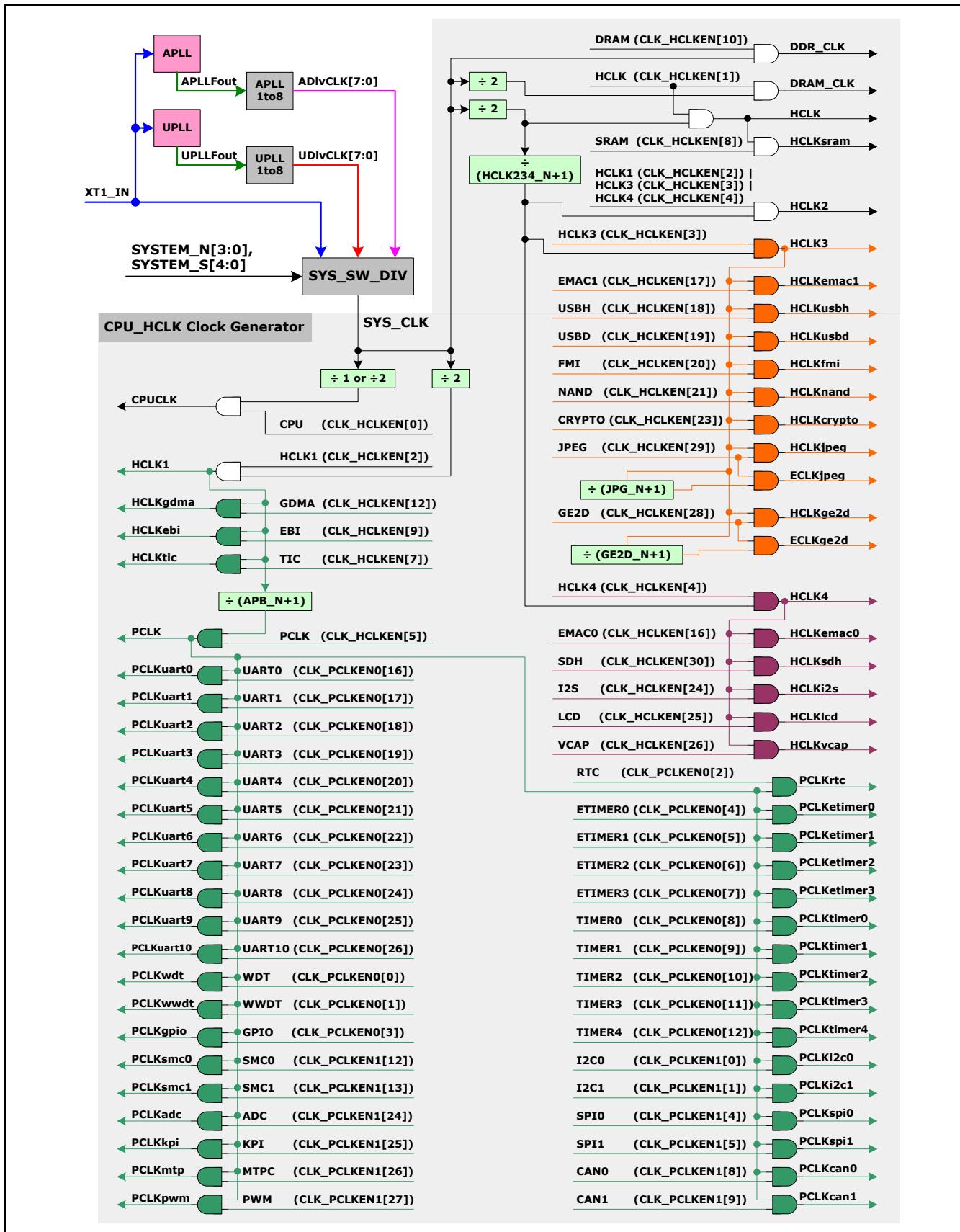
### 2.1 Overview

The clock controller generates all clocks for Video, Audio, CPU, system bus and all functionalities. This chip includes two PLL modules. The clock source for each peripherals comes from the PLL or from the external crystal input directly. For each clock there is a bit on the CLKEN register to control the clock ON or OFF individually, and the divider setting is in the CLK\_DIVCTL register. The register can also be used to control the clock enable or disable for power control.

### 2.2 Features

- Supports two PLLs, up to 500 MHz, for high performance system operation.
- External 12 MHz high speed crystal input for precise timing operation.
- External 32.768 kHz low speed crystal input for RTC function and low speed clock source.

### 2.3 Block Diagram



## 2.4 Register Map

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

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

## 2.5 Functional Description

### 2.5.1 Module Clock On/Off

Each module of NUC970/N9H30 has independent clock control. Before read/write registers, the module clock should be enabled first. Clock On/Off setting is in CLK\_HCLKEN, CLK\_PCLKEN0 and CLK\_PCLKEN1 registers. Module of AHB bus should use CLK\_HCLKEN register. And module of APB bus should use CLK\_PCLKEN0 or CLK\_PCLKEN1 register.

In NAND case, NAND is controlled by AHB bus FMI module. To enable NAND should set FMI and NAND. That is set the CLK\_HCLKEN register FMI and NAND bit. In UART0 case, before print message from UART0, enable UART0 clock first. That is set the CLK\_PCLKEN0 register UART0 bit.

## 2.5.2 Clock Divider

The module which can connect external device has its own clock frequency divider to provide the correct clock output. Each frequency divider in addition to set the divisor, user can also select the clock source. Use an external SD card for example: clock source is UPLL, SD card needs 300 KHz for card identification mode, 50 MHz for data transfer mode. If UPLL is 300MHz, the divider setting is as follow:

1. SDH clock divider register is CLKDIV9, user should control SDH\_N, SDH\_S and SDH\_SDIV.
2. Set the SDH clock source is UPLL, this means fill the 11b to SDH\_S bit.
3. Initialize the frequency to 300 KHz – first UPLL (300 MHz) divides by 5 becomes 60 MHz, and then divides by 200 becomes 300KHz. SDH\_SDIV fill 4 and SDH\_N fill 199 is for this setting.
4. Data transfer frequency to 50MHz – UPLL (300 MHz) divides by 6 becomes 50MHz. SDH\_SDIV fill 0 and SDH\_N fill 5 is for this setting.

## 2.5.3 PLL Setting

NUC970/N9H30 PLL default setting is 264MHz. PLL frequency adjustment needs to meet the following formula.

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

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

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

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

N	F <sub>pfd</sub> Range
1	11.0 ≤ F <sub>pfd</sub> ≤ 80
2	7.0 ≤ F <sub>pfd</sub> ≤ 80
3	5.0 ≤ F <sub>pfd</sub> ≤ 80
4	4.0 ≤ F <sub>pfd</sub> ≤ 80
5	3.5 ≤ F <sub>pfd</sub> ≤ 80
6	3.0 ≤ F <sub>pfd</sub> ≤ 80
7 ~ 8	2.5 ≤ F <sub>pfd</sub> ≤ 80
9 ~ 10	3.5 ≤ F <sub>pfd</sub> ≤ 80
11 ~ 40	3.0 ≤ F <sub>pfd</sub> ≤ 80

41 ~ 128

 $2.5 \leq F_{pfq} \leq 80$

### 3 Analog to Digital Converter (ADC)

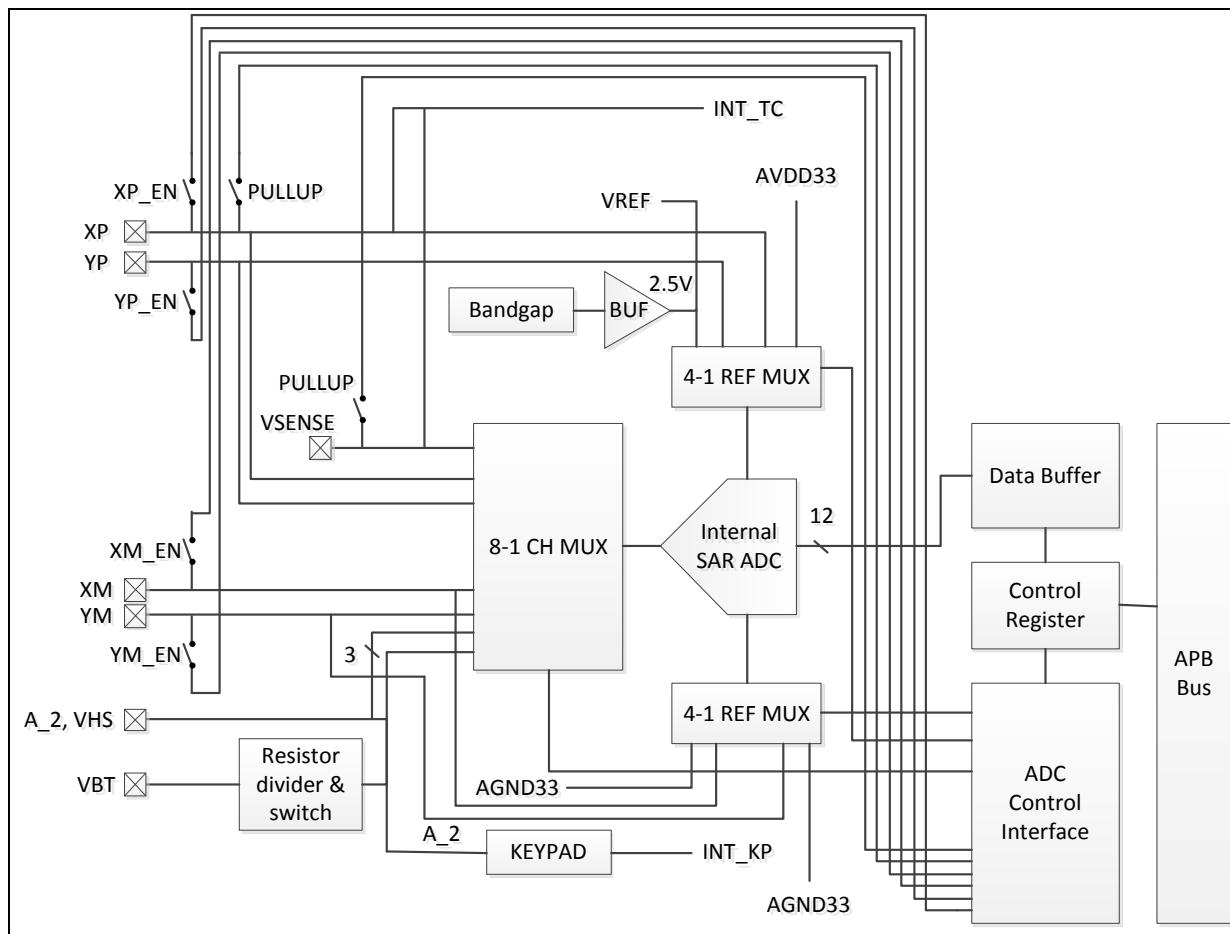
#### 3.1 Overview

The NUC970/N9H30 family contains one 12-bit Successive Approximation Register analog-to-digital converter (SAR A/D converter) with eight input channels. The A/D converter supports two operation modes: 4-wire or 5-wire mode. The ADC is especially suitable to act as touch screen controller. Battery voltage detection could be easily accomplished by the SAR ADC. It has keypad interrupt signal generator.

#### 3.2 Features

- Resolution: 12-bit resolution.
- DNL: +/-1.5 LSB, INL: +/-3 LSB.
- Dual Data Rates: 1MSPS/200KSPS.
- Analog Input Range: VREF to AGND, could be rail-to-rail.
- Analog Supply: 2.7-3.6V.
- Digital Supply: 1.2V.
- 8 Single-Ended Analog inputs.
- Compatible with 4-wire or 5-wire Touch Screen Interface.
- Touch Pressure Measurement for 4-wire touch screen application.
- Direct Battery Measurement.
- Keypad Interrupt Generator.
- Auto Power Down.
- Low Power Consumption: 4850uW(@1MSPS) / 2170uW(@200KSPS), < 1uA

#### 3.3 Block Diagram



### 3.4 Register Map

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

Register	Offset	R/W	Description	Reset Value
<b>ADC Base Address:</b>				
<b>ADC_BA = 0xB800_A000</b>				
<b>ADCON</b>	ADC_BA+0x00	R/W	ADC Control	0x0000_0000
<b>ADC_CONF</b>	ADC_BA+0x04	R/W	ADC Configure	0x0000_0000
<b>ADC_IER</b>	ADC_BA+0x08	R/W	ADC Interrupt Enable Register	0x0000_0000
<b>ADC_ISR</b>	ADC_BA+0x0C	R/W	ADC Interrupt Status Register	0x0000_0000
<b>ADC_WKISR</b>	ADC_BA+0x10	R	ADC Wake Up Interrupt Status Register	0x0000_0000
<b>ADC_XYDATA</b>	ADC_BA+0x20	R	ADC Touch X,Y Position Data	0x0000_0000
<b>ADC_ZDATA</b>	ADC_BA+0x24	R	ADC Touch Z Pressure Data	0x0000_0000
<b>ADC_DATA</b>	ADC_BA+0x28	R	ADC Normal Conversion Data	0x0000_0000
<b>ADC_VBATDATA</b>	ADC_BA+0x2C	R	ADC Battery Detection Data	0x0000_0000
<b>ADC_KPDATA</b>	ADC_BA+0x30	R	ADC Key Pad Data	0x0000_0000

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

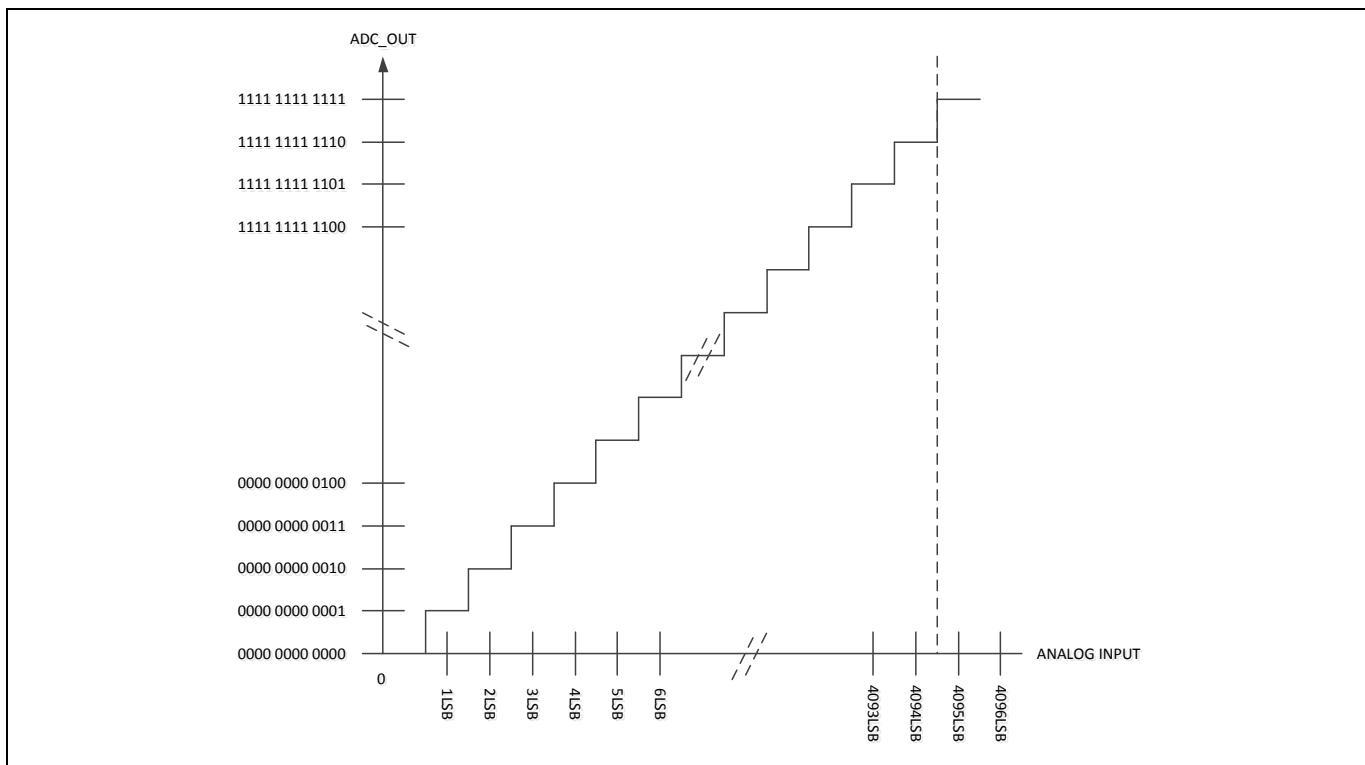
## 3.5 Functional Description

### 3.5.1 Basic Configuration

The ADC peripheral clock can be enabled in ADC (PCLKEN1[24]). The ADC engine clock source is selected by ADC\_S (CLKDIV7[23:16]) and ADC engine clock divider is determined by ADC\_N (CLKDIV7[31:24]).

### 3.5.2 ADC Transfer Function

The ADC output coding is offset in binary, 1LSB=VREF/4096, the transfer characteristic is shown in the following graph:



### 3.5.3 Normal Detection

In normal mode, A/D conversion is performed only once on the specified single channel. Demonstration of a normal detection software program as follows.

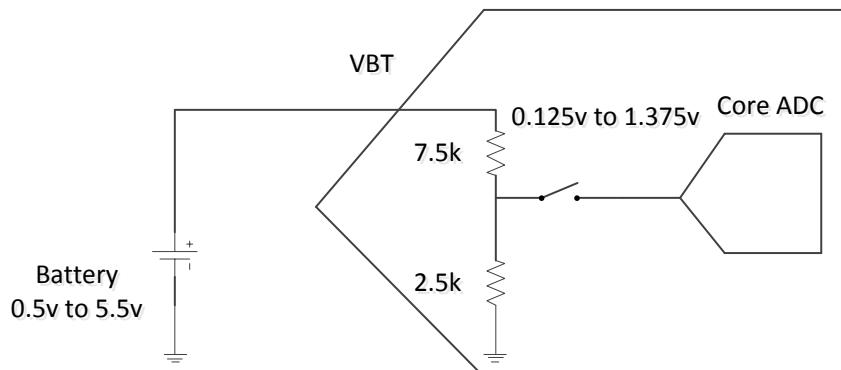
```
char c,num;
unsigned int data,n;
unsigned int d1,d2,val=0;
rREG_CTL |= ADC_CTL_ADEN;
printf("select channel 0:VBT(A0), 1:VHS(A1), 2:A2, 3:A3, 4:YM(A4), 5:YP(A5), 6:XM(A6),
7:XP(A7)\n");
num=getchar();
switch(num)
{
    case '0': val=0; break;
    case '1': val=1; break;
    case '2': val=2; break;
    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 Battery Voltage Detection

Take VBT as input, and select internal buffer's output as the reference. For ADC configure register VBAT\_EN (ADC\_CONF[8]) should be set to 1.

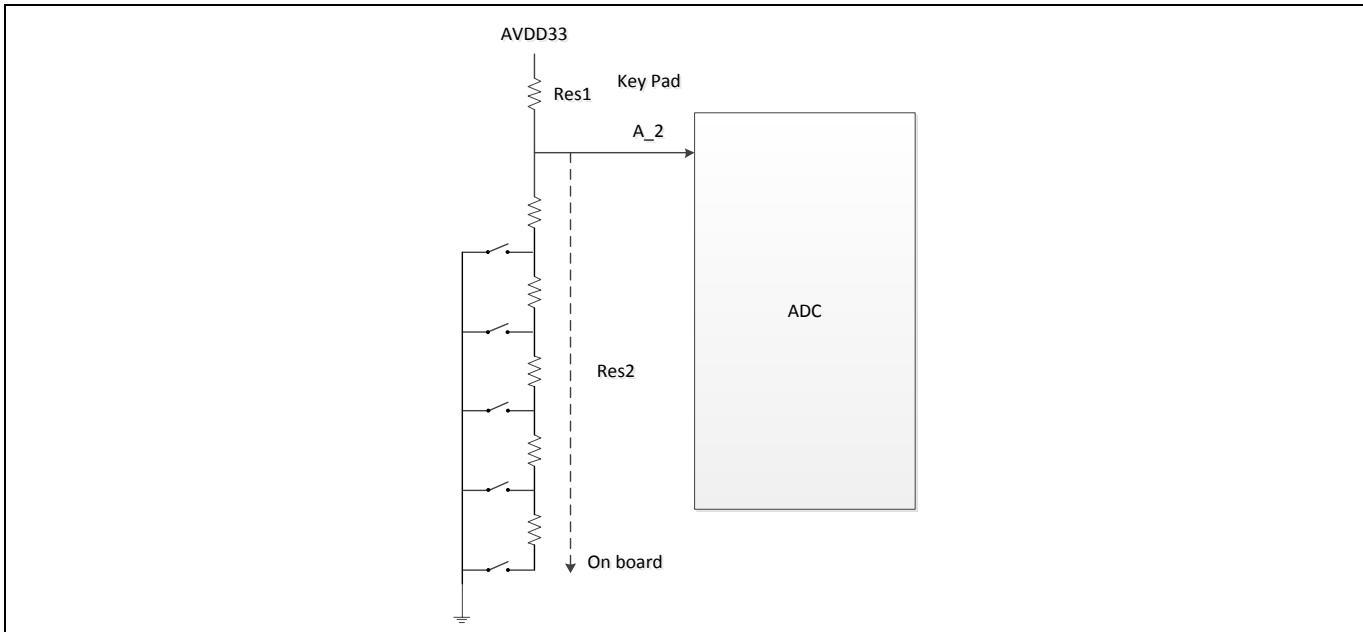


Demonstrations of a battery voltage detection software program as follows:

```
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 Key Pad Scan

Take A\_2 as input, and select AVDD33 and AGND33 as the reference. For ADC configure register KPC\_EN (ADC\_CONF[9]) should be set to 1.



If a user applies the Keypad using this structure and meanwhile, he/she needs the interrupt generator, please make sure  $\text{Res1} \leq 20\text{K}$  ohm and  $\text{Res2} < 5.6 * \text{Res1}$ . Moreover, a  $0.01\mu\text{F}$  cap is recommended at A\_2 on board. If a user doesn't need the interrupt generator, please ignore the requirement for Res1 and Res2.

Demonstration of a key pay scan software program as follows:

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

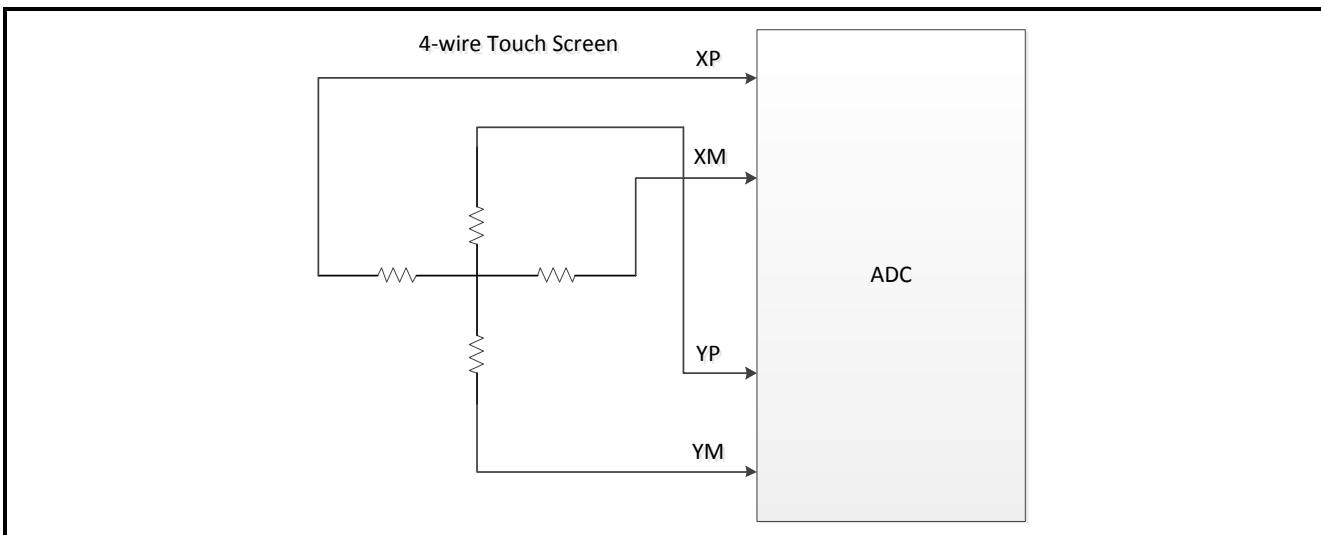
rREG_CONF |= ADC_CONF_KPCEN;
rREG_ISR = ADC_ISR_MF | ADC_ISR_KPCF;
/* keypad interrupt mode */
rREG_IER |= ADC_IER_MIEN;
do{
    while(!(rREG_ISR & ADC_ISR_KPEF)); // Waiting for Interrupt
    rREG_ISR = ADC_ISR_KPEF; //Clear KPEF flag
    rREG_CTL |= ADC_CTL_MST;
    while(!(rREG_ISR & ADC_ISR_MF));
    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;
    }
}
```

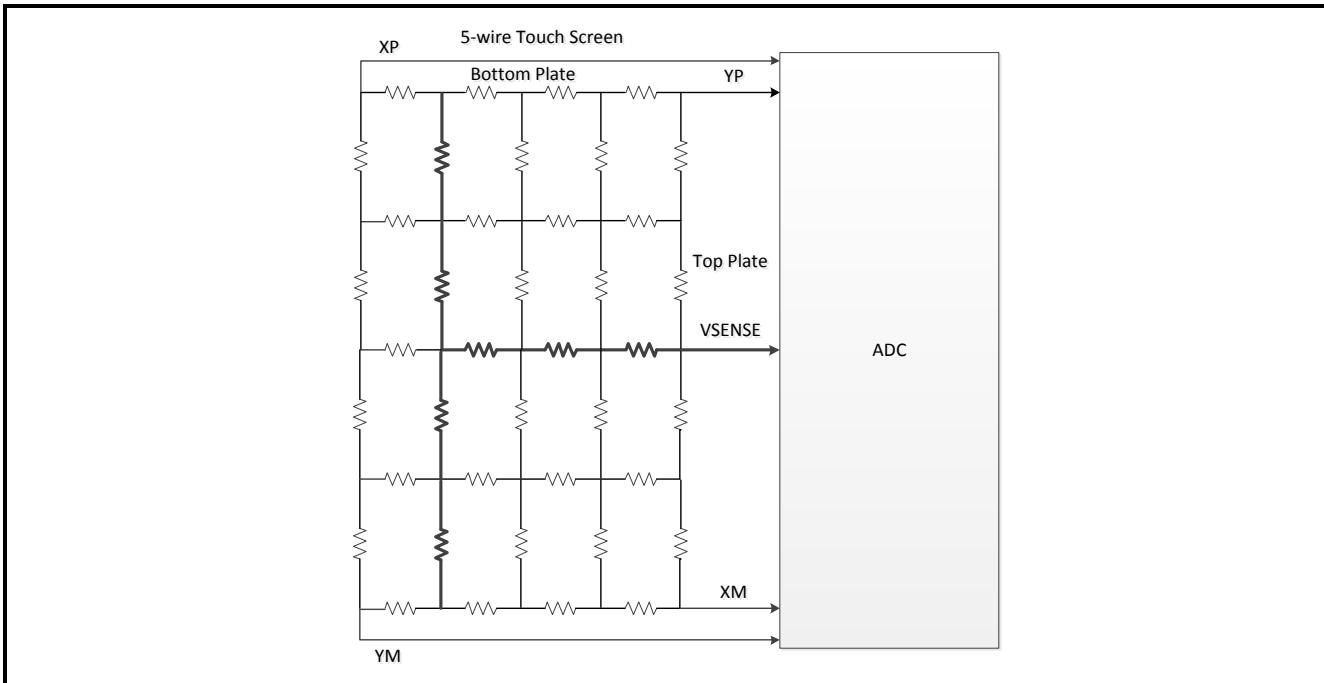
```
while( !( rREG_ISR & ADC_ISR_KPUEF));
rREG_ISR = ADC_ISR_KPUEF; //Clear KPUEF flag
rREG_IER &= ~ADC_IER_KPUEIEN;
}
else
    printf("interrupt error\n");
}while(1);
```

### 3.5.6 4-wire and 5-wire Touch Screen

The touch screen control logic and the switch could control the 4-wire and 5-wire type touch screen. For ADC configure register T\_EN (ADC\_CONF[0]) should be set to 1. ADC control register WMSWCH (ADCON[16]) (Wire Mode Switch) for 5-wire/4-wire configuration. The following figures show the interface for 4-wire, 5-wire touch screen respectively.

Note that, the four switches to bias XP, XM, YP, YM have conduction resistance under 5 ohm. And the pull up PMOS have 200K ohm typically.





Demonstration of a 4-wire touch screen software program as follows:

```
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-wire Pressure Measurement

To distinguish pen or finger touch, the pressure of the touch needs to be determined. The IP provides two solutions. The first method requires knowing the X-plate resistance, measurement of the X-Position, and two additional cross-panel measurements (Z1 and Z2) of the touch screen. Use the following Equation to calculate the touch resistance:

$$R_{Touch} = R_{X-plate} \times \frac{X_{position}}{4096} \left( \frac{Z_2}{Z_1} - 1 \right)$$

The second method requires knowing both the X-plate and Y-plate resistance, measurement of X-Position and Y-Position, and Z1. Use the following Equation to calculate the touch resistance:

$$R_{Touch} = \frac{R_{X-plate} \times X_{position}}{4096} \left( \frac{4096}{Z_1} - 1 \right) - R_{Y-plate} \left( 1 - \frac{Y_{position}}{4096} \right)$$

For ADC configure register Z\_EN (ADC\_CONF[1]) should be set to 1. When Z\_EN (ADC\_CONF[1]) in ADC\_FM register is set; the touch pressure measure Z will be stored in this ADC\_ZDATA register.

Demonstration of a 4-wire pressure measurement software program as follows:

```
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 & 0xFFFF;
    y = (rREG_XYDATA >> 16) & 0xFFFF;
    z1 = rREG_ZDATA & 0xFFFF;
    z2 = (rREG_ZDATA >> 16) & 0xFFFF;
```

```
printf("x = %d, y = %d,z1 = %d, z2 = %d\n", x,y,z1,z2);
}while(1);
```

## 4 Advanced Interrupt Controller (AIC)

### 4.1 Overview

An interrupt temporarily changes the sequence of program execution to react to a particular event such as power failure, watchdog timer timeout, transmit/receive request from Ethernet MAC Controller, and so on. The CPU processor provides two modes of interrupt, the Fast Interrupt (FIQ) mode for critical session and the Interrupt (IRQ) mode for general purpose. The IRQ request is occurred when the nIRQ input is asserted. Similarly, the FIQ request is occurred when the nFIQ input is asserted. The FIQ has privilege over the IRQ and can preempt an ongoing IRQ. It is possible to ignore the FIQ and the IRQ by setting the F and I bits in the current program status register (CPSR).

The Advanced Interrupt Controller (AIC) is capable of processing the interrupt requests up to 64 different sources. Currently, 61 interrupt sources are defined. Each interrupt source is uniquely assigned to an interrupt channel. For example, the watchdog timer interrupt is assigned to channel 1. The AIC implements a proprietary eight-level priority scheme that categories the available 61 interrupt sources into eight priority levels. Interrupt sources within the priority level 0 is the highest priority and the priority level 7 is the lowest. In order to make this scheme work properly, a certain priority level must be specified to each interrupt source during power-on initialization; otherwise, the system shall behave unexpectedly. Within each priority level, interrupt source that is positioned in a lower channel has a higher priority. Interrupt source that is active, enabled, and positioned in the lowest channel with priority level 0 is promoted to the FIQ. Interrupt sources within the priority levels other than 0 are routed to the IRQ. The IRQ can be preempted by the occurrence of the FIQ. Interrupt nesting is performed automatically by the AIC.

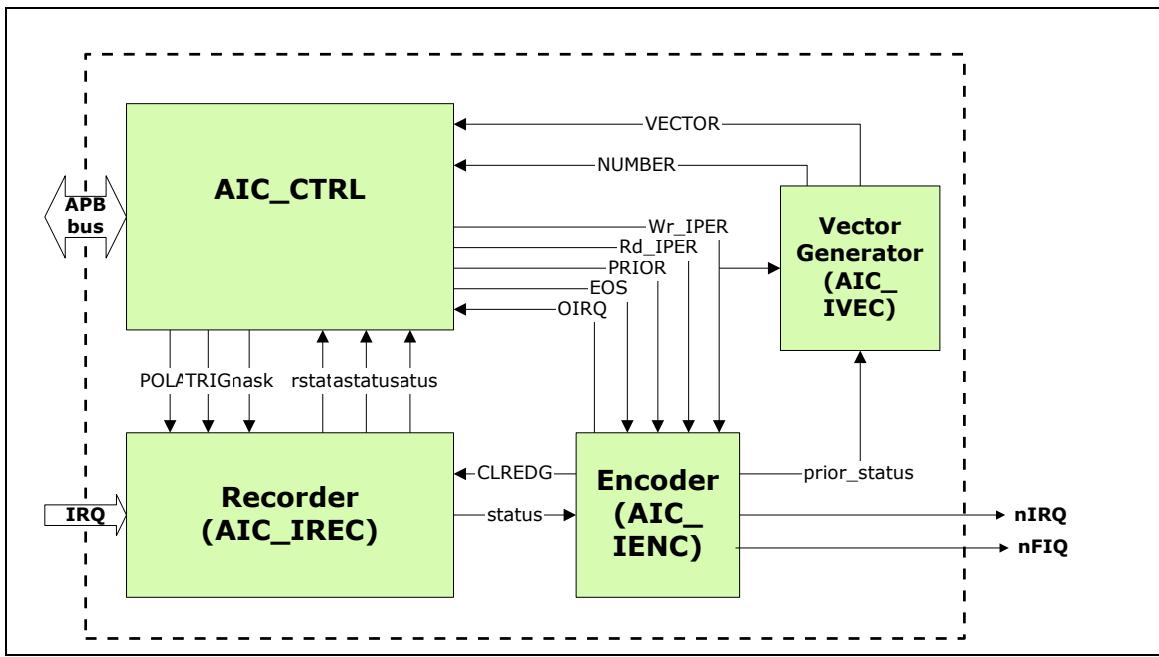
Though interrupt sources originated from the chip itself are intrinsically high-level sensitive, the AIC can be configured as either low-level sensitive, high-level sensitive, negative-edge triggered, or positive-edge triggered to each interrupt source.

### 4.2 Features

- AMBA APB bus interface
- External interrupts can be programmed as either edge-triggered or level-sensitive
- External interrupts can be programmed as either low-active or high-active
- Flags to reflect the status of each interrupt source
- Individual mask for each interrupt source
- Support proprietary 8-level interrupt scheme to employ the priority scheme.
- Priority methodology is adopted to allow for interrupt daisy-chaining
- Automatically masking out the lower priority interrupt during interrupt nesting
- Automatically clearing the interrupt flag when the external interrupt source is

programmed to be edge-triggered

### 4.3 Block Diagram



### 4.4 Register Map

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

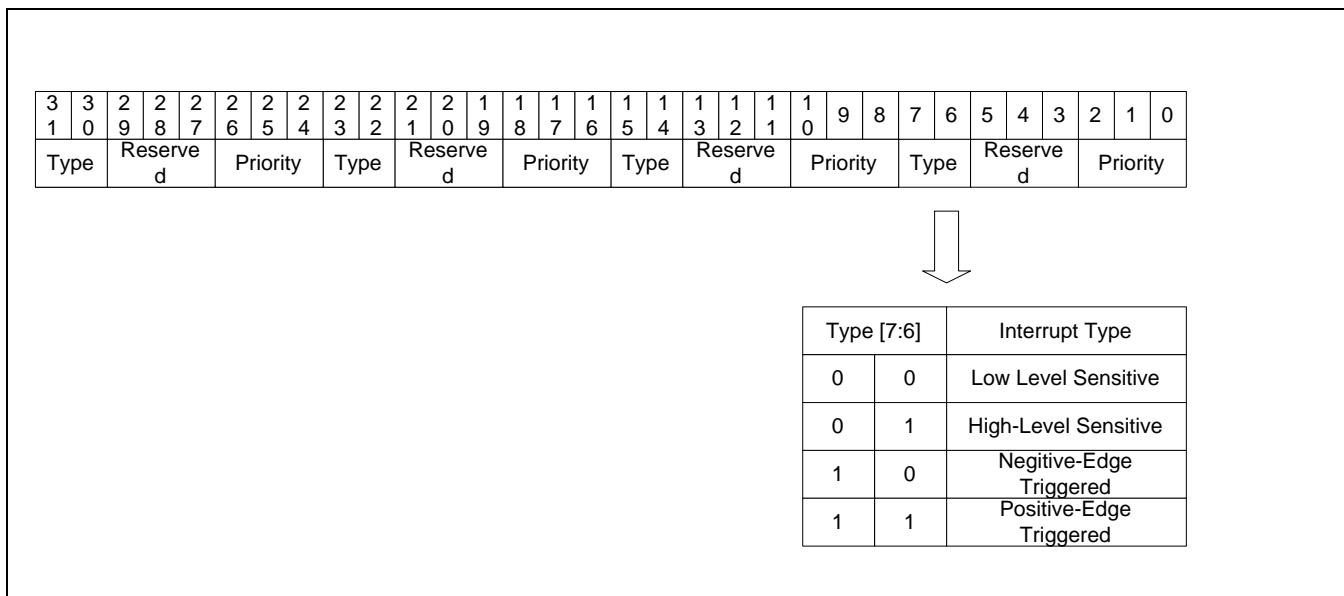
Register	Address	R/W	Description	Reset Value
<b>AIC_BA = 0xB800_2000</b>				
<b>AIC_SCR1</b>	AIC_BA+0x000	R/W	Source Control Register 1	0x4747_4747
<b>AIC_SCR2</b>	AIC_BA+0x004	R/W	Source Control Register 2	0x4747_4747
<b>AIC_SCR3</b>	AIC_BA+0x008	R/W	Source Control Register 3	0x4747_4747
<b>AIC_SCR4</b>	AIC_BA+0x00C	R/W	Source Control Register 4	0x4747_4747
<b>AIC_SCR5</b>	AIC_BA+0x010	R/W	Source Control Register 5	0x4747_4747
<b>AIC_SCR6</b>	AIC_BA+0x014	R/W	Source Control Register 6	0x4747_4747
<b>AIC_SCR7</b>	AIC_BA+0x018	R/W	Source Control Register 7	0x4747_4747
<b>AIC_SCR8</b>	AIC_BA+0x01C	R/W	Source Control Register 8	0x4747_4747
<b>AIC_SCR9</b>	AIC_BA+0x020	R/W	Source Control Register 9	0x4747_4747
<b>AIC_SCR10</b>	AIC_BA+0x024	R/W	Source Control Register 10	0x4747_4747
<b>AIC_SCR11</b>	AIC_BA+0x028	R/W	Source Control Register 11	0x4747_4747
<b>AIC_SCR12</b>	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_MECRH	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 Functional Description

### 4.5.1 Interrupt channel configuration

Each interrupt channel has an independent source control register to set its type and priority. The interrupt type of all NUC970/N9H30 Series MCU internal peripherals is positive-level triggered. This shouldn't be changed during normal operation. The device driver must set the pertinent interrupt type according to the external devices. The priority level of each interrupt channel is completely decided by the interrupted device. After power-on or reset, all the channels are assigned to priority level 0~7 by AIC. The following figure shows the content of source control register.



#### 4.5.2 Interrupt Masking

The NUC970/N9H30 Series MCU AIC provides a set of registers to mask individual interrupt channel. The Mask Enable Command Register (AIC\_MECR) is used to enable interrupt. Write 1 to a bit of MECR will enable the corresponding interrupt channel. Oppositely, the Mask Disable Command Register (AIC\_MDCR) is used to disable the interrupt. Write 1 to a bit of MDCR will disable the corresponding interrupt channel. Write 0 to a bit of AIC\_MEGR or AIC\_MDCR has no effect. Therefore, the device driver can arbitrarily change these two registers without keeping their original values. If it's necessary, the device driver can read the Interrupt Mask Register (AIC\_IMR) to know whether the interrupt channel is enabled or disabled. If the interrupt channel is enabled, its corresponding bit is read as 1, otherwise 0.

#### 4.5.3 Interrupt Clearing and Setting

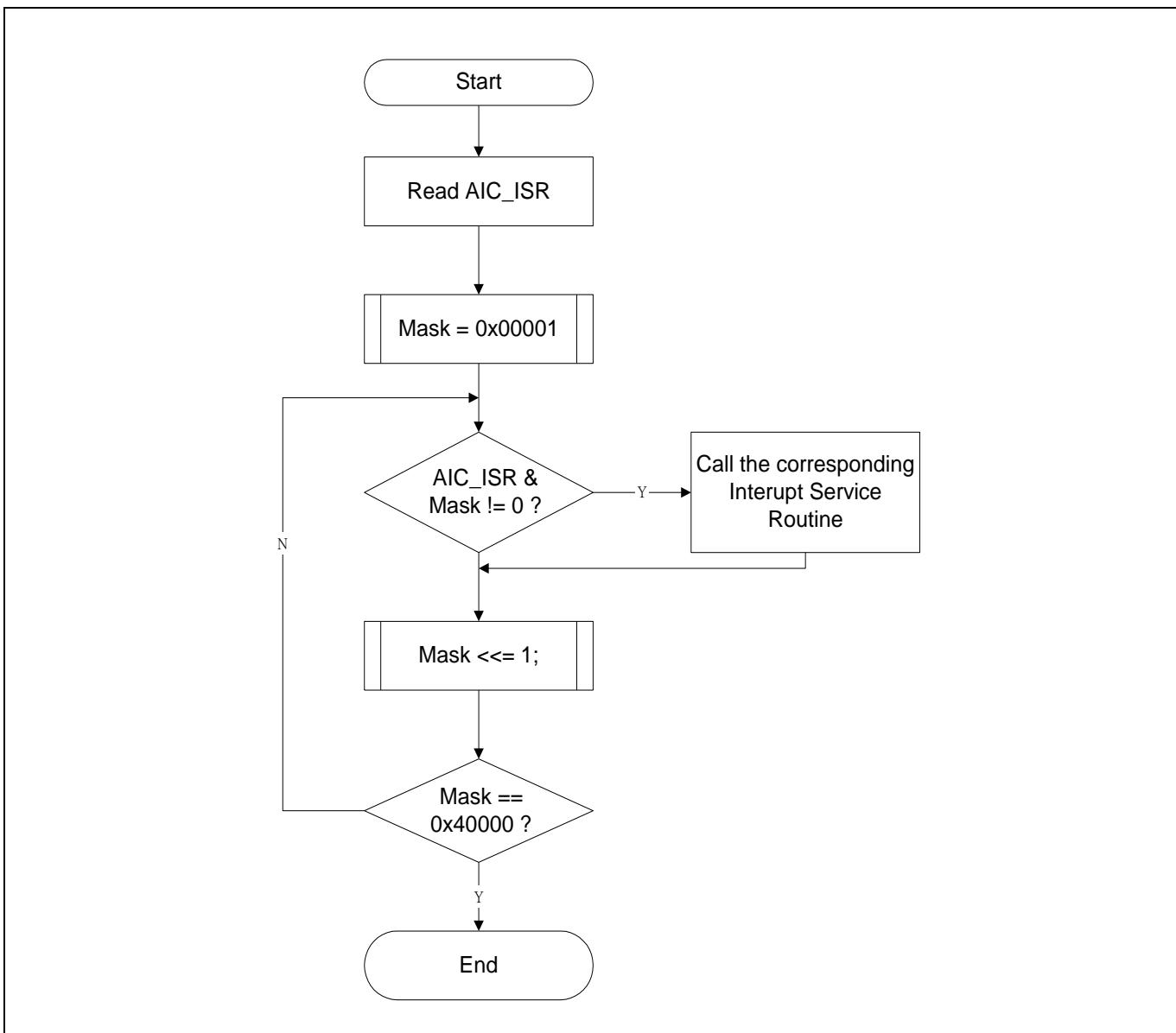
For the interrupt channels that are level sensitive, the device driver doesn't need to write the Source Clear Command Register (AIC\_SCCR) or End of Service Command Register (AIC\_EOSCR) to clear any AIC status. As soon as the device's interrupt status has been cleared, the AIC de-asserts the interrupt request. For the interrupt channels that are edge-triggered, the device driver must clear AIC status to de-assert the interrupt request. To clear AIC status, the device driver may either write Source Clear Command Register (AIC\_SCCR) or End of Service Command Register (AIC\_EOSCR). Write 1 to a bit of AIC\_SCCR will clear the corresponding interrupt. The usage of AIC\_EOSCR will be discussed in the section Hardware Priority Scheme.

The register Source Set Command Register (AIC\_SSCR) is used to active an interrupt channel when it is programmed to edge-triggered. Write 1 to a bit of AIC\_SSCR will set the corresponding interrupt. This feature is useful in auto-testing or software debugging.

#### 4.5.4 Software Priority Scheme

The AIC provides an Interrupt Status Register (AIC\_ISR) to identify the interrupt sources. If an interrupt channel is both active and enabled, its corresponding bit in AIC\_ISR is set as 1. The interrupt handler of FIQ or IRQ can get the interrupt sources by reading AIC\_ISR. And the service sequence is completely decided by software algorithm.

Generally, there's a function table to keep the interrupt service routines of internal peripherals and external devices. When the interrupt is recognized by CPU core, the FIQ or IRQ exception handler is executed firstly. Then it will call the proper interrupt service routine according to the AIC\_ISR content. The following figure demonstrates a sequential priority scheme where channel 1 has the highest priority and channel 17 18 has the lowest priority.



```
__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 Hardware Priority Scheme

The AIC implements a proprietary 8-level priority scheme. To use this mechanism, the proper AIC\_SCRx should be programmed before enable the interrupt channels. Similarly, the FIQ or IRQ exception handler is executed firstly when the interrupt is recognized. The exception handler and interrupt service routine should follow certain rules to let this mechanism work correctly. The rules are listed below.

1. Reads IRQ Priority Encoding Register (AIC\_IPER) to get the Vector (IRQ Channel x 4), and at this mean time, the AIC\_ISNR will be loaded by the current interrupt channel number, the Vector (IRQ Channel Number x 4) represents the interrupt channel number that is active, enabled, and has the highest priority, multiplied by 4, then stored on the AIC\_IPER. The data (Vector) got from AIC\_IPER is convenient for the following interrupt service route address calculation. Enabled, and has the highest priority.
2. Branch to the corresponding interrupt service routine by adding Vector to the base of interrupt service routine table.
3. Write any value to AIC\_EOSCR to finish the interrupt.

The priority level of the interrupt channel that is active and enabled is treated as current priority level. It is pushed into the Priority Encoder when AIC\_IPER is read. In the same time, the AIC\_ISNR was loaded by the current encoded interrupt channel number. This prevents AIC from asserting an interrupt request if the following active and enabled interrupt has lower priority level. Therefore, the interrupt service routine must write AIC\_EOSCR to pop the current priority level from priority Encoder to let AIC service the interrupt channel with lower priority. This hardware priority control is helpful to implement a nesting interrupt system.

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

```

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

```

It is very important that ISR must write AIC\_EOSCR to restore to normal interrupt state once it read the AIC\_IPER. Otherwise, the next interrupt may not be serviced correctly

#### 4.5.6 Interrupt Sources

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 Overview

The C\_CAN consists of the CAN Core, Message RAM, Message Handler, Control Registers and Module Interface. The CAN Core performs communication according to the CAN protocol version 2.0 part A and B. The bit rate can be programmed to values up to 1MBit/s. For the connection to the physical layer, additional transceiver hardware is required.

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM. All functions concerning the handling of messages are implemented in the Message Handler. These functions include acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.

The register set of the C\_CAN can be accessed directly by the software through the module interface. These registers are used to control/configure the CAN Core and the Message Handler and to access the Message RAM.

### 5.2 Features

- Supports CAN protocol version 2.0 part A and B
- Bit rates up to 1 MBit/s
- 32 Message Objects
- Each Message Object has its own identifier mask
- Programmable FIFO mode (concatenation of Message Objects)
- Maskable interrupt
- Disabled Automatic Re-transmission mode for Time Triggered CAN applications
- Programmable loop-back mode for self-test operation
- 16-bit module interfaces to the AMBA APB bus
- Supports wake-up function

### 5.3 Block Diagram

The C\_CAN interfaces with the AMBA APB bus. The following figure shows the block diagram of the C\_CAN.

- CAN Core

CAN Protocol Controller and Rx/Tx Shift Register for serial/parallel conversion of messages.

- Message RAM

Stores Message Objects and Identifier Masks

- Registers

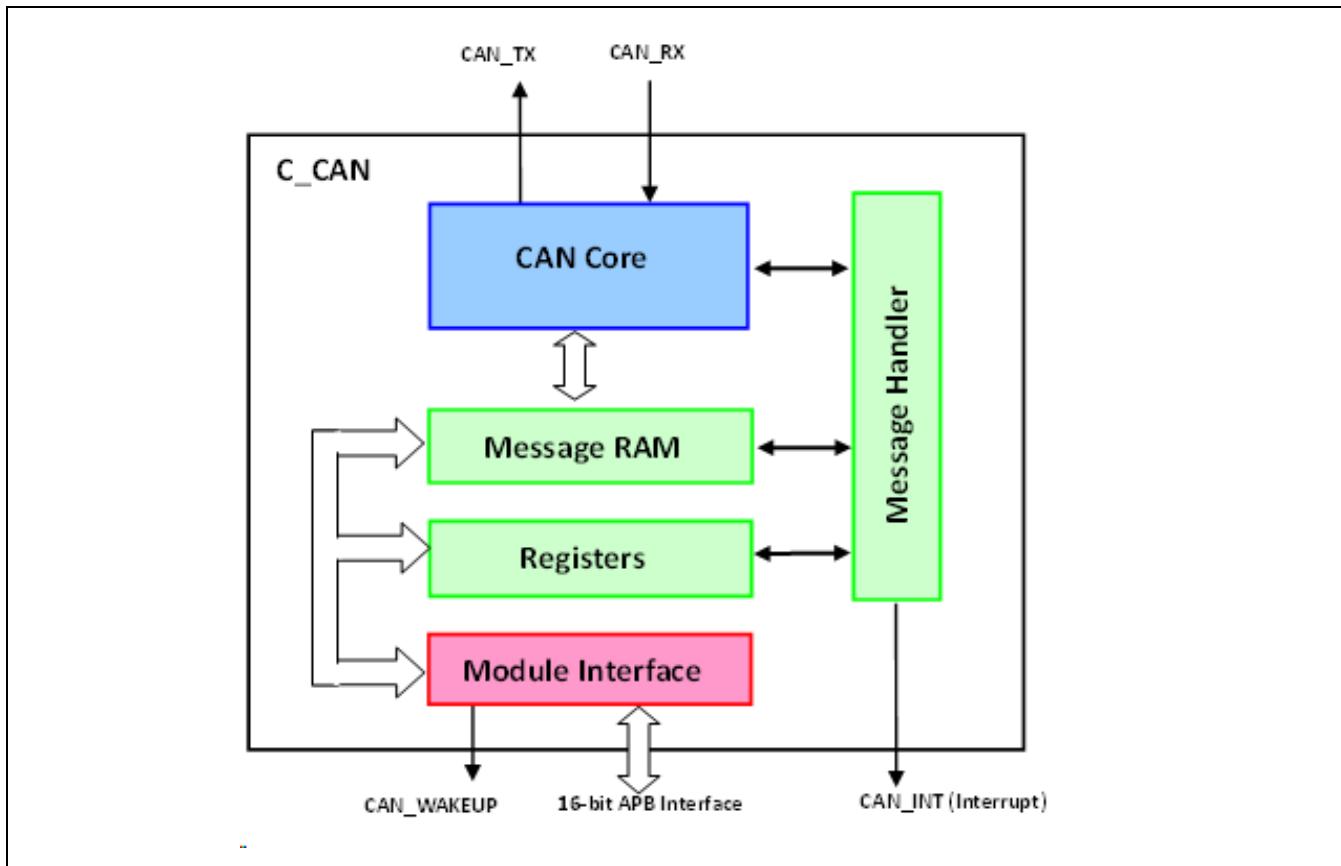
All registers used to control and to configure the C\_CAN.

- Message Handler

State Machine that controls the data transfer between the Rx/Tx Shift Register of the CAN Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers.

- Module Interface

C\_CAN interfaces to the AMBA APB 16-bit bus from ARM.



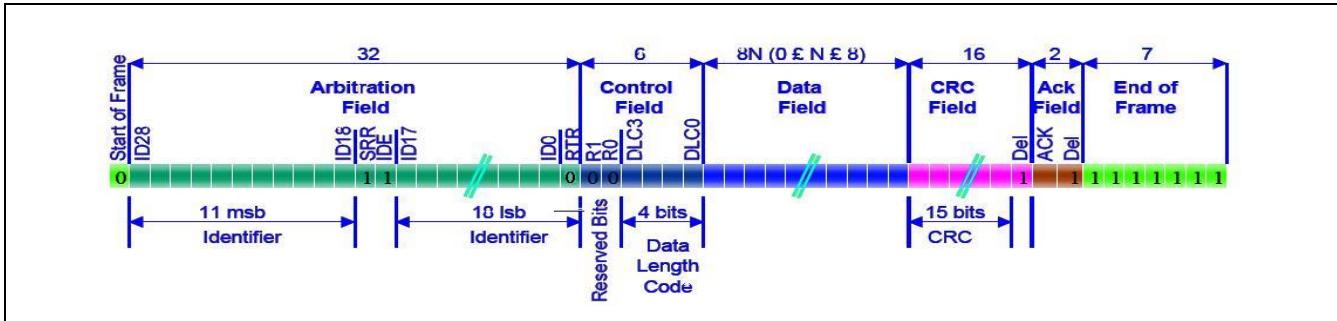
## 5.4 Register Map

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

<b>CAN0_BA = 0xB800_0000</b> <b>CAN1_BA = 0xB800_4000</b>				
<b>CAN_CON</b>	CANx_BA+0x00	R/W	Control Register	0x0000_0001
<b>CAN_STATUS</b>	CANx_BA+0x04	R/W	Status Register	0x0000_0000
<b>CAN_ERR</b>	CANx_BA+0x08	R	Error Counter	0x0000_0000
<b>CAN_BTIME</b>	CANx_BA+0x0C	R/W	Bit Timing Register	0x0000_2301
<b>CAN_IIDR</b>	CANx_BA+0x10	R	Interrupt Identifier Register	0x0000_0000
<b>CAN_TEST</b>	CANx_BA+0x14	R/W	Test Register	*(1)
<b>CAN_BRPE</b>	CANx_BA+0x18	R/W	BRP Extension Register	0x0000_0000
<b>CAN_IF1_CREQ</b> <b>CAN_IF2_CREQ</b>	CANx_BA+0x20 CANx_BA+0x80	R/W	IFn (*2) Command Request Registers	0x0000_0001
<b>CAN_IF1_CMASK</b> <b>CAN_IF2_CMASK</b>	CANx_BA+0x24 CANx_BA+0x84	R/W	IFn Command Mask Registers	0x0000_0000
<b>CAN_IF1_MASK1</b> <b>CAN_IF2_MASK1</b>	CANx_BA+0x28 CANx_BA+0x88	R/W	IFn Mask 1 Register	0x0000_FFFF
<b>CAN_IF1_MASK2</b> <b>CAN_IF2_MASK2</b>	CANx_BA+0x2C CANx_BA+0x8C	R/W	IFn Mask 2 Register	0x0000_FFFF
<b>CAN_IF1_ARB1</b> <b>CAN_IF2_ARB1</b>	CANx_BA+0x30 CANx_BA+0x90	R/W	IFn Arbitration 1 Register	0x0000_0000
<b>CAN_IF1_ARB2</b> <b>CAN_IF2_ARB2</b>	CANx_BA+0x34 CANx_BA+0x94	R/W	IFn Arbitration 2 Register	0x0000_0000
<b>CAN_IF1_MCON</b> <b>CAN_IF2_MCON</b>	CANx_BA+0x38 CANx_BA+0x98	R/W	IFn Message Control Registers	0x0000_0000
<b>CAN_IF1_DAT_An/</b> <b>CAN_IF1_DAT_Bn/</b> <b>CAN_IF2_DAT_An/</b> <b>CAN_IF2_DAT_Bn/</b>	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
<b>CAN_TXREQ1</b> <b>CAN_TXREQ2</b>	CANx_BA+0x100 CANx_BA+0x104	R	Transmission Request Registers 1 & 2	0x0000_0000
<b>CAN_NDAT1</b> <b>CAN_NDAT2</b>	CANx_BA+0x120 CANx_BA+0x124	R	New Data Registers 1 & 2	0x0000_0000
<b>CAN_IPND1</b> <b>CAN_IPND2</b>	CANx_BA+0x140 CANx_BA+0x144	R	Interrupt Pending Registers 1 & 2	0x0000_0000
<b>CAN_MVLD1</b> <b>CAN_MVLD2</b>	CANx_BA+0x160 CANx_BA+0x164	R	Message Valid Registers 1 & 2	0x0000_0000
<b>CAN_WU_EN</b>	CANx_BA+0x168	R/W	Wake-up Function Enable	0x0000_0000
<b>CAN_WU_STATUS</b>	CANx_BA+0x16C	R/W	Wake-up Function Status	0x0000_0000

## 5.5 Functional Description

### 5.5.1 CAN Protocol



The CAN Data Frame format consist SOF, Arbitration Field, Control Field, Data Field, CRC field, ACK Field and EOF.

Each field describe as follow:

- SOF: Start of Frame
- Arbitration Field: Any potential bus conflicts are resolved by bitwise arbitration
- Control Field: Include 4 bits Data Length Code and 2 bits Reserved Bits
- Data Field: Containing from zero to eight bytes
- CRC Field: Containing a fifteen bit cyclic redundancy check code
- ACK Field: An empty slot which will be filled by every node that receives the frame it does NOT say that the node you intended the data for got it, just that at least one node on the whole network got it
- EOF: End of Frame

### 5.5.2 CAN Baud Rate Setting

CAN supports bit rates in the range of lower than 1 Kbit/s up to 1000 Kbit/s .

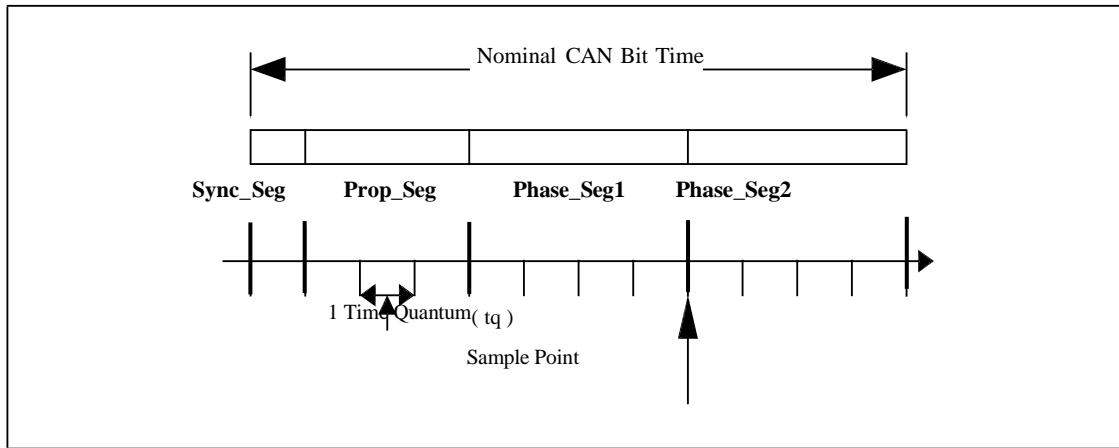
CAN transfer rate  $f_{speed}$  can be show :  $f_{speed} = 1/t_{NBT}$ , where  $t_{NBT}$  is bit time.

According to the CAN specification, the bit time is divided into four segments (see the following figure). The Synchronization Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2 :

- The Synchronization Segment, Sync\_Seg, is that part of the bit time where edges of the

CAN bus level are expected to occur. The distance between an edge that occurs outside of Sync\_Seg, and the Sync\_Seg is called the phase error of that edge.

- The Propagation Time Segment, Prop\_Seg, is intended to compensate for the physical delay times within the CAN network.
- The Phase Buffer Segments Phase\_Seg1 and Phase\_Seg2 surround the Sample Point. The (Re-)Synchronization Jump Width (SJW) defines how far a re-synchronization may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.



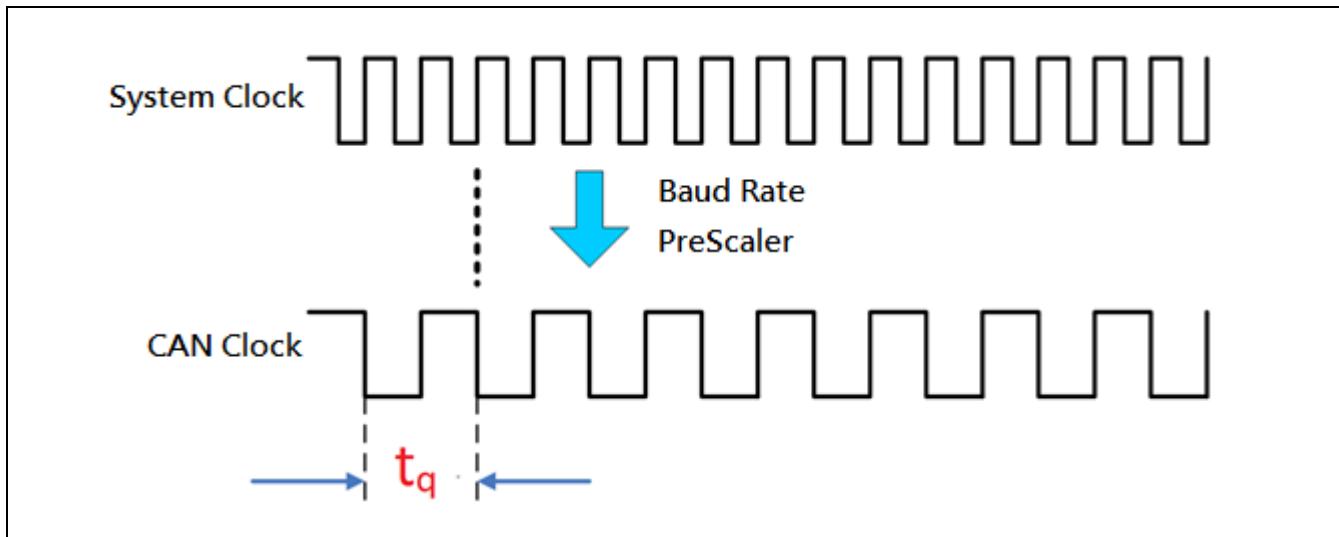
The length of the bit time is [Sync\_Seg + Prop\_Seg + Phase\_Seg1 + Phase\_Seg2] \* tq

The bit time may consist of 4 to 25 time quanta.

The length of the time quantum tq is  $t_q = \frac{(BPR + 1)}{f_{APB\_CLK}}$

BRP: Baud Rate PreScaler Value

$f_{APB\_CLK}$  : System Clock



In these bit timing registers of CAN controller

$$TSEG1 + 1 = (t_{PROP\_SEG} + t_{PHASE\_SEG1})t_q$$

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

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

TSEG1, TSEG2 are the control bit of register CAN\_BTME.

According above describe, we can find the baud-rate function :

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

$$= 1/(1 + (TSEG1 + 1) + (TSEG2 + 1))t_q$$

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

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

$f_{APB\_CLK}$  : System clock

TSEG1, TSEG2 and BPR are the control bit filed of register CAN\_BITME

For Example:

If CAN bus baud-rate is 1000kbps, CPU APB clock is 75 MHz , we can set TSEG1 =6, TSEG2 =6, BPR =4. The speed is :

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

$$= 75000000 / (6 + 6 + 3) (4 + 1)$$

= 1000 kbps

We also can set TSEG1 =7, TSEG2 =5, other parameter not change, the CAN speed will keep on 1000 kbps, but the sample point will be changed.

### 5.5.3 CAN Module Register

CAN module register address base is CAN0\_BA=0xB800\_0000 , There are three modules of CAN registers: CAN Protocol Related Registers, Message Interface Registers and Message Handler Registers. These registers address base show as follow:

Register Module	Offset	Register name	
CAN Protocol Related Registers	0x00 ~ 0x18	CAN_CON	CAN_STATUS
		CAN_ERR	CAN_BTIME
		CAN_IIDR	CAN_TEST
		CAN_BRPE	
Message Interface Registers	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*	
Message Handler Registers	0x100 ~ 0x164	CAN_TXREQn*	CAN_NDATn*
		CAN_IPNDn*	CAN_MVLD1n*
			* : n=1或2

- CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

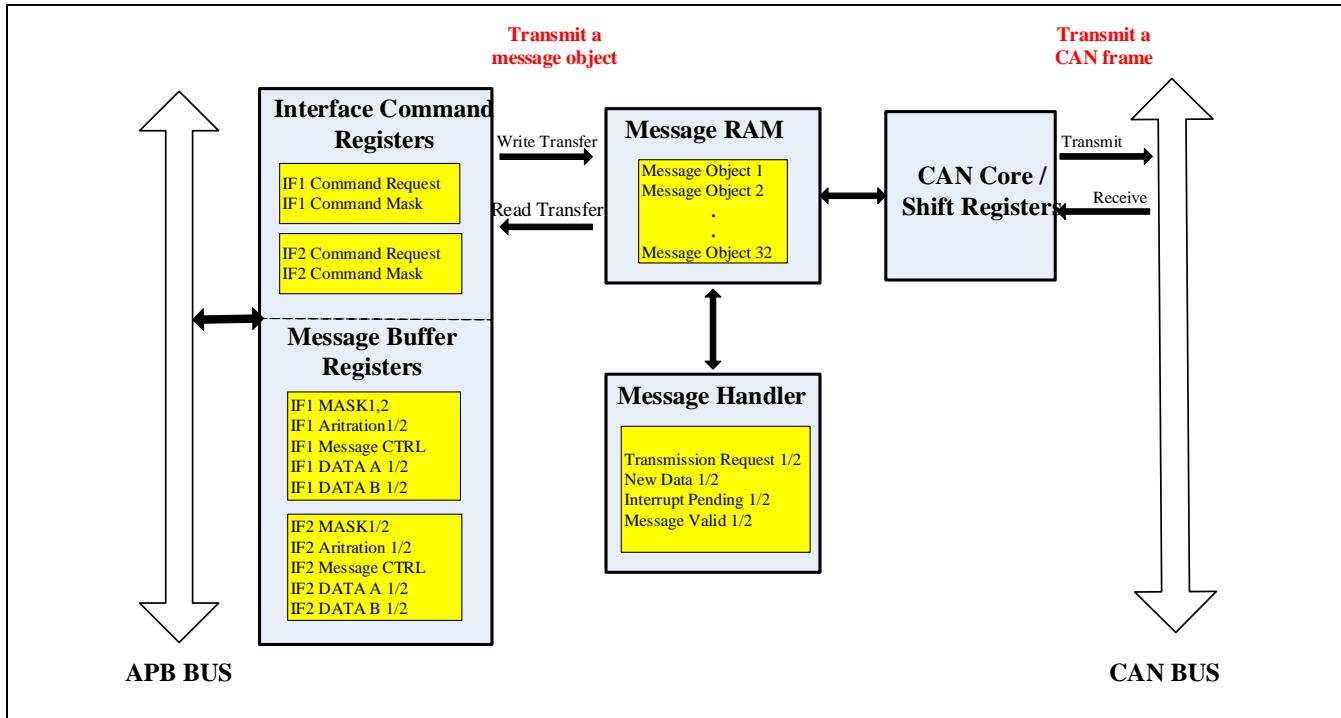
- Message Interface Register Sets

There are two sets of Interface Registers which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred.

- Message Handler Registers

All Message Handler registers are read-only. Their contents (TxRqst, NewDat, IntPnd, and MsgVal bits of each Message Object and the Interrupt Identifier) are status information provided by the Message Handler FSM.

These registers relationship show in follow figure:



The configuration of the Message Objects in the Message RAM will (with the exception of the bits MsgVal, NewDat, IntPnd, and TxRqst) not be affected by resetting the chip. All the Message Objects must be initialized by the CPU or they must be not valid (MsgVal = '0') and the bit timing must be configured before the CPU clears the Init bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data field of one of the two interfaces register sets to the desired values. By writing to the corresponding IFx Command Request Register, the IFx Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the Init bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN\_Core and the Message Handler State Machine control the C\_CAN's internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, messages with pending transmission request are loaded into the CAN\_Core's Shift Register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFx Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

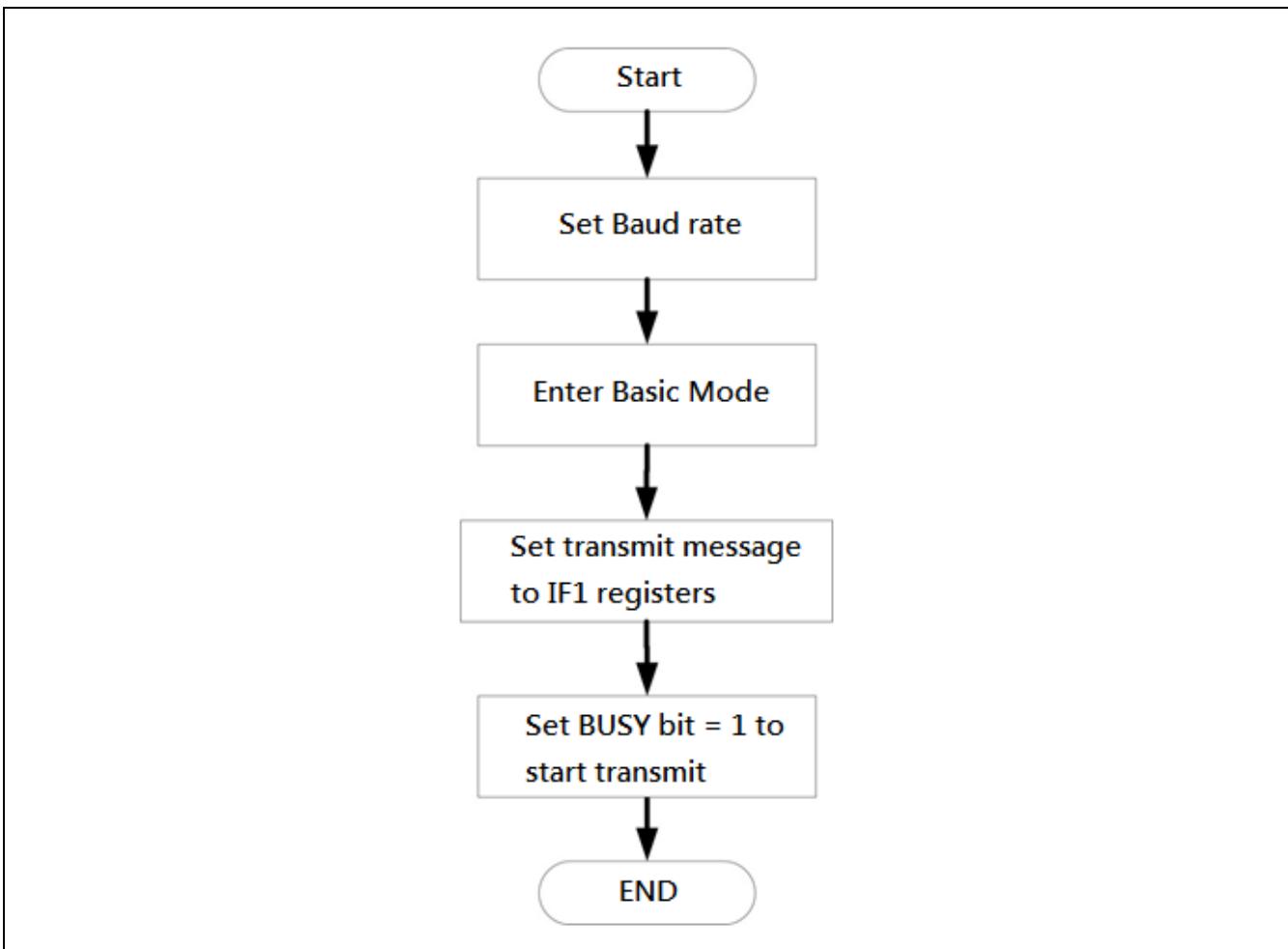
#### 5.5.4 Transfer CAN Message

The C\_CAN Module includes two Modes: Normal Mode and Basic Mode.

In Basic mode:

The C\_CAN module runs without the Message RAM. The IF1 Registers are used as Transmit Buffer. The IF2 Registers are used as Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers.

In Basic Mode, the transmit message flow as below figure:



We can follow below steps to transfer a message to CAN bus:

1. Set CAN bus Baud rate.
2. Enter Basic Mode: Set bit TEST(CAN\_CON[7]) and bit BASIC(CAN\_TEST[2])
3. Set transmit message to IF1 registers.
4. Set bit BUSY(CAN\_CREQ[15]) to start transfer message. This bit will be auto-cleared when finish transmitting.

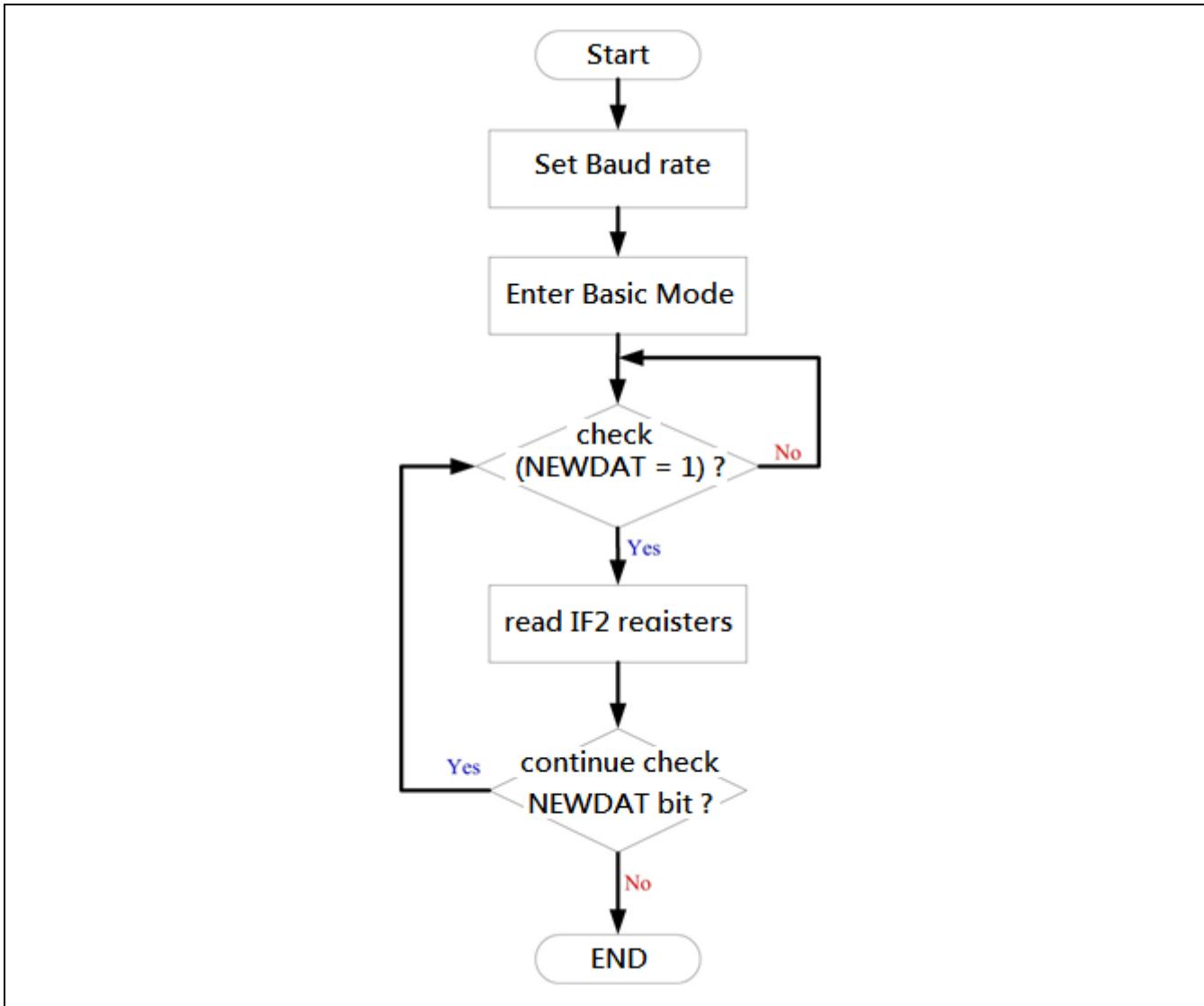
When use Basic Mode, please care following status:

- Make sure CAN Module enter Test Mode
- Make sure the bit BUSY(CAN\_CREQ[15]) is set “1”.

### 5.5.5 Receive CAN Message

There two method to receive CAN message: one is polling the bit NEWDAT (CAN\_IFn\_MCON[15]) the other is use Rx interrupt. The received message will be store to IF2 registers.

Following figure means the flow of polling bitNEWDAT(CAN\_IFn\_MCON[15]):

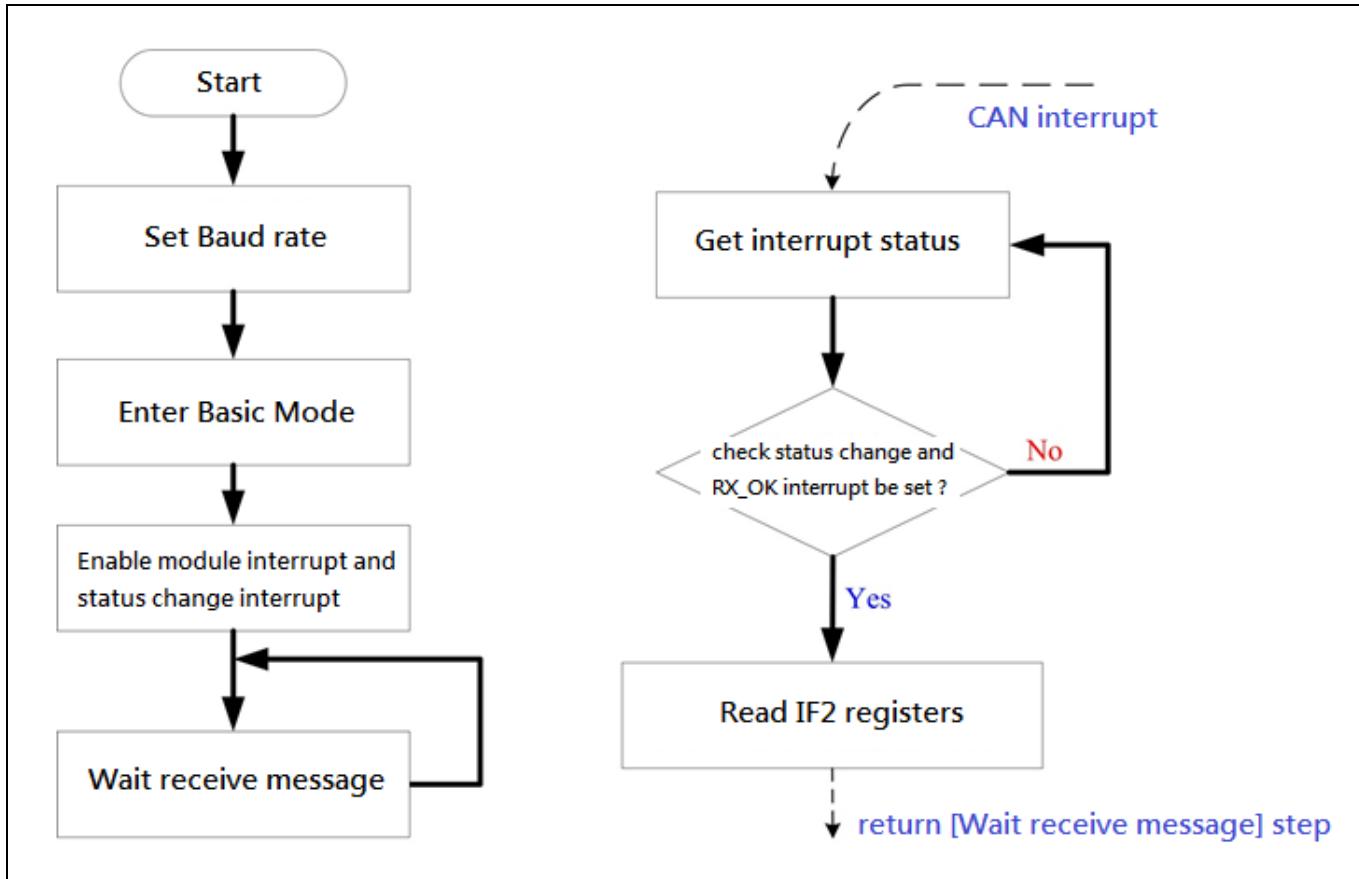


In Basic Mode, use polling mode to receive message flow as below:

1. Set CAN bus Baud rate.
2. Enter Basic Mode: Set bit TEST(CAN\_CON[7]) and bit BASIC(CAN\_TEST[2])
3. Polling bit NEWDAT(CAN\_IF2\_MCON[15]) until this bit be set “1”

4. Read CAN\_IF2 registers can get received message.

Following figure shows the flow that use RX\_OK interrupt to receive message:



In Basic Mode, use RX\_OK interrupt to receive message flow as follow:

1. Set CAN bus Baud rate.
2. Enter Basic Mode: Set bit TEST(CAN\_CON[7]) and bit BASIC(CAN\_TEST[2])
3. Enable interrupt and status change interrupt: Set bit IE(CAN\_CON[1]) and bit SIE(CAN\_CON[2]).
4. Wait interrupt happened. If bit RX\_OK(CAN\_STATUS[4]) is “1”, means CAN module receive a message. Read IF2 registers can get this message.

### 5.5.6 Wakeup Function

Set bit WAKEUP\_EN(CAN\_WU\_EN[0]) can enable wakeup function. And User can wake-up system when there is a falling edge in the CAN\_Rx pin.

## 6 Cryptographic Accelerator (NUC970 only)

### 6.1 Overview

The Crypto (Cryptographic Accelerator) includes a secure pseudo random number generator (PRNG) core and supports AES, DES/TDES, SHA and HMAC algorithms.

The PRNG core supports 64 bits, 128 bits, 192 bits, and 256 bits random number generation.

The AES accelerator is an implementation fully compliant with the AES (Advance Encryption Standard) encryption and decryption algorithm. The AES accelerator supports ECB, CBC, CFB, OFB, CTR, CBC-CS1, CBC-CS2, and CBC-CS3 mode.

The DES/TDES accelerator is an implementation fully compliant with the DES and Triple DES encryption/decryption algorithm. The DES/TDES accelerator supports ECB, CBC, CFB, OFB, and CTR mode.

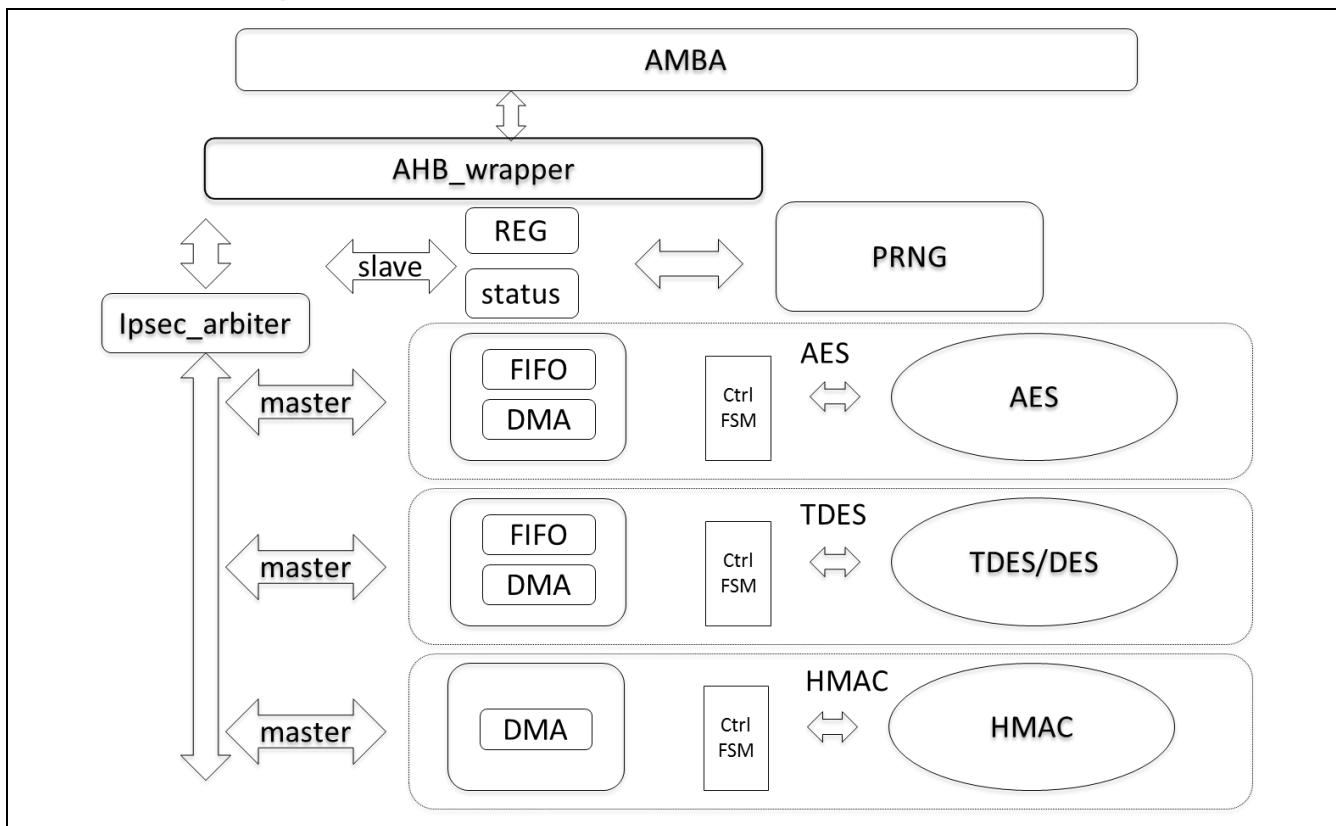
The SHA accelerator is an implementation fully compliant with the SHA-160, SHA-224, SHA-256, SHA-384, and SHA-512 and corresponding HMAC algorithms

### 6.2 Features

- PRNG
  - Supports 64 bits, 128 bits , 192 bits, and 256 bits random number generation
- AES
  - Supports FIPS NIST 197
  - Supports SP800-38A and addendum
  - Supports 128, 192, and 256 bits key
  - Supports both encryption and decryption
  - Supports ECB, CBC, CFB, OFB , CTR, CBC-CS1, CBC-CS2, and CBC-CS3 mode
  - Supports external key (AES key from MTP)
- DES
  - Supports FIPS 46-3
  - Supports both encryption and decryption
  - Supports ECB, CBC, CFB, OFB, and CTR mode
- TDES
  - Supports FIPS NIST 800-67
  - Implemented according to the X9.52 standard

- Supports two keys or three keys mode
- Supports both encryption and decryption
- Supports ECB, CBC, CFB, OFB, and CTR mode
- SHA
  - Supports FIPS NIST 180, 180-2
  - Supports SHA-160, SHA-224, SHA-256, SHA-384, and SHA-512
- HMAC
  - Supports FIPS NIST 180, 180-2
  - Supports HMAC-SHA-160, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512

### 6.3 Block Diagram



### 6.4 Register Map

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 Functional Description

The cryptographic accelerator includes a secure pseudo random number generator (PRNG) core and supports AES, DES/TDES, SHA, and HMAC algorithms. The accelerator can be used in different data security applications, such as secure communications that need cryptographic protection and integrity.

The PRNG core supports 64 bits, 128 bits, 192 bits, and 256 bits random number generation configured by KEYSZ.

The AES accelerator is a fully compliant implementation of the AES (Advance Encryption Standard) encryption and decryption algorithm. The AES accelerator supports ECB, CBC, CFB, OFB, CTR, CBC-CS1, CBC-CS2, and CBC-CS3 mode. The AES accelerator provides

the DMA function to reduce the CPU intervention, and supports three burst lengths, sixteen-words, eight-words, and four-words.

The DES/TDES accelerator is a fully compliant implementation of the DES and Triple DES encryption/decryption algorithm. The DES/TDES accelerator supports ECB, CBC, CFB, OFB, and CTR mode. The DES/TDES accelerator also supports the DMA function to reduce the CPU intervention. Only two burst lengths, four words and eight words, are supported.

The SHA/HMAC accelerator is a fully compliant implementation of the SHA-160, SHA-224, SHA-256, SHA-384, SHA-512, and corresponding HMAC algorithm. The SHA/HMAC accelerator also supports the DMA function to reduce the CPU intervention. It supports three burst lengths, sixteen-words, eight-words, and four-words.

### 6.5.1 Data Access

The cryptographic accelerator supports the following features to enhance the performance:

1. **DMA mode:** Once DMA source address register, destination address register, and byte count register are configured by CPU, moving data from and to accelerator is done by DMA logic totally. This mode can off-load the loading from the CPU. The cryptographic accelerator embeds four hardware DMA channels for AES engine, four hardware DMA channels for DES/TDES engine, and one hardware DMA channel for SHA/HMAC engine.
2. **DMA Cascade mode:** In the case that the data SRAM resource is tight, or another peripheral is scheduled to switch, the data source or sink needs an update, while the setting for the accelerator operation is planned to be kept. In this mode, software can update DMA source address register, destination address register, and byte count register during a cascade operation, without finishing the accelerator operation.
3. **Non-DMA mode:** In the case that the input data is small in size, DMA mode is not preferred. This mode can reduce the processing time for the accelerator, since no DMA related register needs a configuration, and no latency in DMA logic is introduced. Input data was feeding to cryptographic engine via writing to data input register.
4. **Channel Expansion mode:** In this mode, several virtual channels in one of four DMA channels are feasible in AES or DES/TDES mode. The total channel number can exceed the limit of four DMA channels. The intermediate data from feedback registers (CRPT\_AES\_FDBCKx, CRPT\_TDES\_FDBCKH, and CRPT\_TDES\_FDBCKL) should be stored temporarily in data SRAM. And switch to another configuration setting of accelerator operation that includes operational mode, encryption/decryption, key, key size, IV, and other parameters. Once switching back, the intermediate data from feedback registers should be written to initial vectors (CRPT\_AESn\_IVx, CRPY\_TDESx\_IVH, and CRPT\_TDESx\_IVL) for the accelerator to continue the

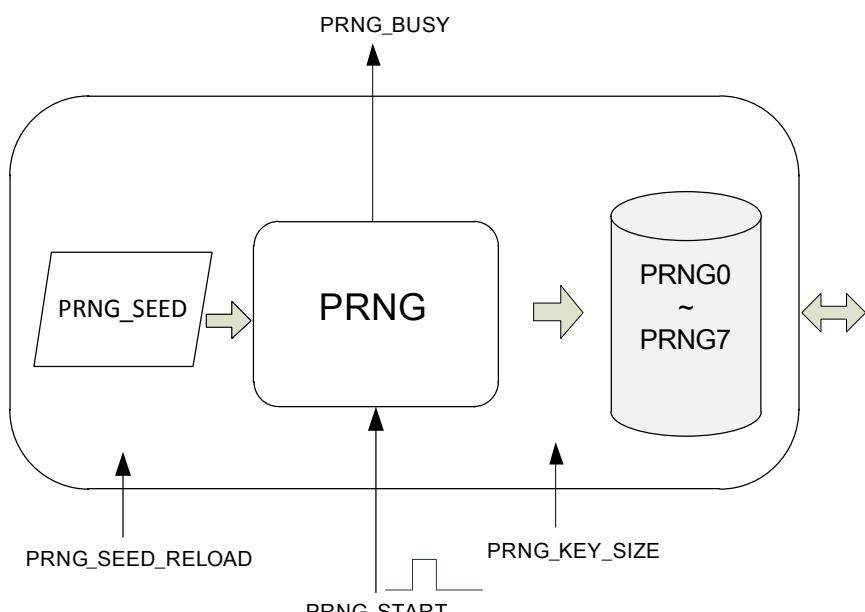
operation with the original configuration setting. Note that, in ECB mode, there is no need to move the intermediate data from feedback registers to IV.

### 6.5.2 Channel Expansion

AES and DES/TDES accelerator each supports four DMA channels. However, the extended virtual channel on either DMA channel is feasible if the user requires more than four cryptographic channels. By recording the feedback register and the current DMA channel's cryptographic settings, including control registers, key, and initial vectors, driver can create a snapshot of a physical cryptographic channel. Once a snapshot is created, this physical channel can be migrated to service other encrypt/decrypt requests. Driver can later recover the cryptographic channel by writing that snapshot back to control registers. Driver can create a lot of snapshots of cryptographic channel. In this way, number of cryptographic channel is not limited by four physical channels. The user can have as many cryptographic channels as memory is enough.

### 6.5.3 PRNG

The PRNG block diagram is depicted below. The core supports 64 bits, 128 bits, 192 bits, and 256 bits random number generation configured by KEYSZ(CRPT\_PRNG\_CTL[3:2] ).



Program steps to get the pseudo random number are depicted below:

1. Check BUSY(CRPT\_PRNG\_CTL[8]) until it comes to 0.

2. Initialize PRNG parameters. Select key size by KEYSZ (CRPT\_PRNG\_CTL[3:2]), and write a random seed to CRPT\_PRNG\_SEED. Note that CRPT\_PRNG\_SEED should be initialized since it's not initialized as the chip powers up.
3. Write setting value to PRNG control register CRPT\_PRNG\_CTL. At the same, set START(CRPT\_PRNG\_CTL[0]) as 1 to trigger PRNG.
4. Check BUSY(CRPT\_PRNG\_CTL[8]) until it comes to 0, or waits for the PRNG done interrupt (must enable the corresponding interrupt enable register). User can then get the output random numbers from CRPT\_PRNG\_KEY0 ~ CRPT\_PRNG\_KEY7 registers.
5. User can repeat step 3~4 to get a sequence of random numbers.

#### 6.5.4 AES

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. NUC970 AES accelerator is fully compliant with AES standards.

Users can refer to the following steps to learn how to use NUC970 AES accelerator.

##### 6.5.4.1 AES DMA Mode Operating Flow

1. Write 1 to AESIEN (CRPT\_INTEN[0]) to enable AES interrupt.
2. Select an available channel from four AES channels.
3. Program AES key to registers CRPT\_AESn\_KEY0 ~ CRPT\_AESn\_KEY7 (where n is the selected channel number). If user wants to use MTP key as AES key, just write 1 to EXTKEY(CRPT\_AES\_CTL[4]) instead of writing CRPT\_AESn\_KEY0 ~ CRPT\_AESn\_KEY7 registers.
4. Program initial vectors to registers CRPT\_AESn\_IV0 ~ CRPT\_AESn\_IV3. If user selects AES ECB mode, there's no need to write initial vectors.
5. Write DMA source address to register CRPT\_AESn\_SADDR and write DMA destination address to register CRPT\_AESn\_DADDR respectively. If CPU data cache is enabled, the DMA address must be located in a non-cacheable address area.
6. Write DMA byte count to register CRPT\_AESn\_CNT.
7. Configure AES control register CRPT\_AES\_CTL for channel selection, encryption/decryption, operational mode, DMA mode, key size, and DMA input/output swap.
8. Write input data to DMA source address. The byte count of data must be the same as selected DMA byte count.
9. Write 1 to START(CRPT\_AES\_CTL[0]) to start AES encryption/decryption.

10. Waits for the AES interrupt flag AESIF (CRPT\_INTSTS[0]) be set.
11. Read output data from DMA destination address. The byte count of output data is the same as selected DMA byte count.
12. Repeat step 8 to step 11 until all data processed.

#### 6.5.4.2 AES Non-DMA Mode Operating Flow

1. Write 1 to AESIEN (CRPT\_INTEN[0]) to enable AES interrupt.
2. Select an available channel from four AES channels.
3. Program AES key to registers CRPT\_AESn\_KEY0 ~ CRPT\_AESn\_KEY7 (where n is the selected channel number). If user wants to use MTP key as AES key, just write 1 to EXTKEY(CRPT\_AES\_CTL[4]) instead of writing CRPT\_AESn\_KEY0 ~ CRPT\_AESn\_KEY7 registers.
4. Program initial vectors to registers CRPT\_AESn\_IV0 ~ CRPT\_AESn\_IV3. If user selects AES ECB mode, there's no need to write initial vectors.
5. Configure AES control register CRPT\_AES\_CTL for channel selection, encryption/decryption, operational mode, DMA mode, and key size.
6. Write 1 to START(CRPT\_AES\_CTL[0]) to start AES encryption/decryption.
7. If INBUFFULL(CRPT\_AES\_STS[9]) bit is 0, write an input data word to CRPT\_AES\_DATIN register.
8. If OUTBUFEMPTY(CRPT\_AES\_STS[16]) bit is 0, read an output data word from CRPT\_AES\_DATOUT register.
9. Repeat steps 7~8 until there's 4 words read from CRPT\_AES\_DATOUT register.
10. Write 1 to DMALAST(CRPT\_AES\_CTL[5]). It means the completion of an AES block.
11. Repeat steps 7~10, until all encrypt/decrypt data are processed.

#### 6.5.5 DES/TDES

FIPS 46-3 specifies two cryptographic algorithms, the Data Encryption Standard (DES) and the Triple Data Encryption Algorithm (TDEA). The cryptographic accelerator supports FIPS 46-3, both encryption and decryption, and ECB, CBC, CFB, OFB and CTR modes.

Users can refer to the following steps to learn how to use NUC970 DES/TDES accelerator.

#### 6.5.5.1 DES/TDES DMA Mode Operating Flow

1. Write 1 to TDESIEN (CRPT\_INTEN[8]) to enable DES/TDES interrupt.
2. Select an available channel from four DES/TDES channels.

3. Program DES/TDES key to registers CRPT\_TDESn\_KEY1H, CRPT\_TDESn\_KEY1L, CRPT\_TDESn\_KEY2H, CRPT\_TDESn\_KEY2L, CRPT\_TDESn\_KEY3H, and CRPT\_TDESn\_KEY3L (where n is the selected channel number).
4. Program initial vectors to registers CRPT\_TDESn\_IVH and CRPT\_TDESn\_IVL. If user selects AES ECB mode, there's no need to write initial vectors.
5. Write DMA source address to register CRPT\_TDESn\_SADDR and write DMA destination address to register CRPT\_TDESn\_DADDR respectively. If CPU data cache is enabled, the DMA address must be located in a non-cacheable address area.
6. Write DMA byte count to register CRPT\_TDESn\_CNT.
7. Configure DES/TDES control register CRPT\_TDES\_CTL for channel selection, encryption/decryption, operational mode, DMA mode, key size, and DMA input/output swap.
8. Write input data to DMA source address. The byte count of data must be the same as selected DMA byte count.
9. Write 1 to START(CRPT\_TDES\_CTL[0]) to start DES/TDES encryption/decryption.
10. Waits for the DES/TDES interrupt flag TDESIF (CRPT\_INTSTS[8]) be set.
11. Read output data from DMA destination address. The byte count of output data is the same as selected DMA byte count.
12. Repeat step 8 to step 11 until all data processed.

#### 6.5.5.2 DES/TDES Non-DMA Mode Operating Flow

1. Write 1 to TDESIEN (CRPT\_INTEN[8]) to enable DES/TDES interrupt.
2. Select an available channel from four DES/TDES channels.
3. Program DES/TDES key to registers CRPT\_TDESn\_KEY1H, CRPT\_TDESn\_KEY1L, CRPT\_TDESn\_KEY2H, CRPT\_TDESn\_KEY2L, CRPT\_TDESn\_KEY3H, and CRPT\_TDESn\_KEY3L (where n is the selected channel number).
4. Program initial vectors to registers CRPT\_TDESn\_IVH and CRPT\_TDESn\_IVL. If user selects AES ECB mode, there's no need to write initial vectors.
5. Configure DES/TDES control register CRPT\_TDES\_CTL for channel selection, encryption/decryption, operational mode, DMA mode, and key size.
6. Write 1 to START(CRPT\_TDES\_CTL[0]) to start DES/TDES encryption/decryption.
7. If INBUFFULL(CRPT\_TDES\_STS[9]) bit is 0, write an input data word to CRPT\_TDES\_DATIN register.
8. If OUTBUFEMPTY(CRPT\_TDES\_STS[16]) bit is 0, read an output data word from CRPT\_TDES\_DATOUT register.

9. Repeat steps 7~8 until there's 2 words read from CRPT\_TDES\_DATOUT register.
10. Write 1 to DMALAST(CRPT\_TDES\_CTL[5]). It means the completion of an DES/TDES block.
11. Repeat steps 7~10, until all encrypt/decrypt data are processed.

## 6.5.6 SHA

The Secure Hash Algorithm is a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS).

Users can refer to the following steps to learn how to use NUC970 DES/TDES accelerator.

### 6.5.6.1 SHA DMA Mode Operating Flow

1. Write 1 to HMACIEN(CRPT\_INTEN[24]) to enable SHA/HMAC interrupt.
2. Configure SHA/HMAC control register CRPT\_HMAC\_CTL for SHA/HMAC engine input/output data swap, DMA mode, and SHA operation mode. Clear HMACEN(CRPT\_HMAC\_CTL[4]) to select SHA mode.
3. Program DMA source address to register CRPT\_HMAC\_SADDR.
4. Program DMA byte count to register CRPT\_HMAC\_DMACNT.
5. Write input data to DMA source address with selected DMA byte count.
6. Write 1 to START(CRPT\_HMAC\_CTL[0]) to start SHA encryption.
7. Waits for the SHA interrupt flag HMACIF(CRPT\_INTSTS[24]) be set.
8. Read output digest (SHA160: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST4, SHA224: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST6, SHA256: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST7, SHA384: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST11, SHA512: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST15).

### 6.5.6.2 SHA Non-DMA Mode Operating Flow

1. Configure SHA/HMAC control register CRPT\_HMAC\_CTL for SHA/HMAC engine input/output data swap, DMA mode, and SHA operation mode. Clear HMACEN(CRPT\_HMAC\_CTL[4]) to select SHA mode.
2. Write 1 to START(CRPT\_HMAC\_CTL[0]) to start SHA encryption.
3. If it's the last input word, set DMALAST(CRPT\_HMAC\_CTL[5]).
4. Write 1 to START(CRPT\_HMAC\_CTL[0]) to start SHA encryption.
5. Waits for the SHA data input request DATINREQ(CRPT\_HMAC\_STS[16]) be set.

6. Write one word of input data to CRPT\_HMAC\_DATIN.
7. Repeat step 2 to 5 until all input words are written into SHA engine.
8. Waits for the BUSY (CRPT\_HMAC\_STS[0]) be cleared.
9. Read output digest (SHA160: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST4, SHA224: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST6, SHA256: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST7, SHA384: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST11, SHA512: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST15).

#### 6.5.6.3 SHA DMA Mode Operating Flow

1. Write 1 to HMACIEN(CRPT\_INTEN[24]) to enable SHA/HMAC interrupt.
2. Configure SHA/HMAC control register CRPT\_HMAC\_CTL for SHA/HMAC engine input/output data swap, DMA mode, and SHA operation mode. Clear HMACEN(CRPT\_HMAC\_CTL[4]) to select SHA mode.
3. Program DMA source address to register CRPT\_HMAC\_SADDR.
4. Program DMA byte count to register CRPT\_HMAC\_DMACNT.
5. Write input data to DMA source address with selected DMA byte count.
6. Write 1 to START(CRPT\_HMAC\_CTL[0]) to start HMAC encryption.
7. Waits for the HMAC interrupt flag HMACIF(CRPT\_INTSTS[24]) be set.
8. Read output digest (HMAC-SHA160: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST4, HMAC-SHA224: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST6, HMAC-SHA256: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST7, HMAC-SHA384: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST11, HMAC-SHA512: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST15).

#### 6.5.6.4 SHA Non-DMA Mode Operating Flow

1. Configure SHA/HMAC control register CRPT\_HMAC\_CTL for SHA/HMAC engine input/output data swap, DMA mode, and SHA operation mode. Set HMACEN(CRPT\_HMAC\_CTL[4]) as 1 to select HMAC mode.
2. If it's the last input word, set DMALAST(CRPT\_HMAC\_CTL[5]).
3. Write 1 to START(CRPT\_HMAC\_CTL[0]) to start HMAC encryption.
4. Waits for the HMAC data input request DATINREQ(CRPT\_HMAC\_STS[16]) be set.
5. Write one word of input data to CRPT\_HMAC\_DATIN.
6. Repeat step 2 to 5 until all input words are written into SHA engine.
7. Waits for the BUSY (CRPT\_HMAC\_STS[0]) be cleared.

- 
8. Read output digest (HMAC-SHA160: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST4, HMAC-SHA224: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST6, HMAC-SHA256: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST7, HMAC-SHA384: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST11, HMAC-SHA512: CRPT\_HMAC\_DGST0 ~ CRPT\_HMAC\_DGST15).

## 7 External Bus Interface (EBI)

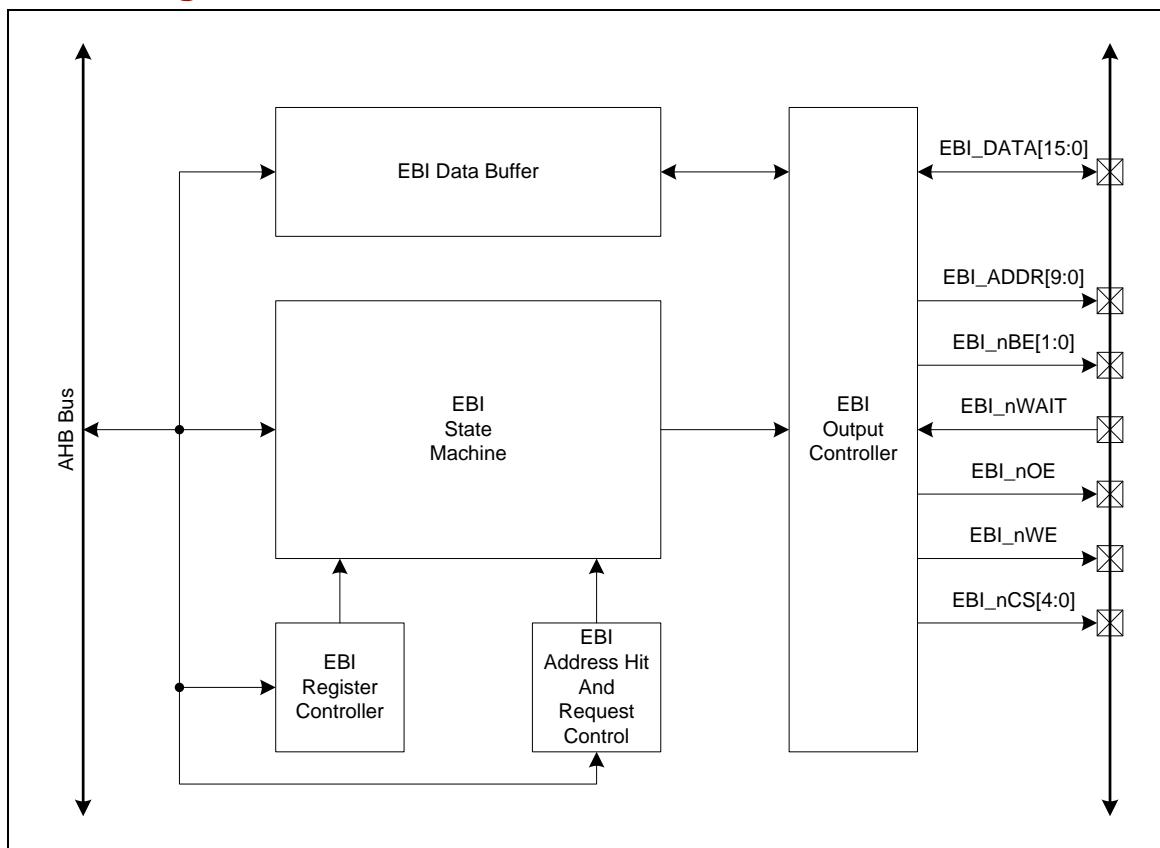
### 7.1 Overview

This chip supports External Bus Interface (EBI), which controls the access to the external memory (SRAM) and External I/O devices. The EBI has up to 5 chip select signals to select different devices with 10-bit address bus. It supports 8-bit and 16-bit external data bus width for each bank.

### 7.2 Features

- Support SRAM and external I/O devices.
- Support 8/16-bit data bus width.
- Support 80 and 68 mode interface signals.
- Support up to 5 chip selects for SRAM and external I/O devices.
- Support programmable access cycle.
- Support four 32-bit write buffers.

### 7.3 Block Diagram



## 7.4 Register Map

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

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

## 7.5 Functional Description

### 7.5.1 Basic Configuration

Before using External Bus Interface, it's necessary to configure related pins as the EBI function and enable EBI's clock. For EBI related pin configuration, please refer to the register SYS\_MFP\_GPDH, SYS\_MFP\_GPHL, SYS\_MFP\_GPHH, SYS\_MFP\_GPIL and SYS\_MFP\_GPIH to know how to configure related pins as the EBI function. To enable EBI's clock for operation, please set EBI (CLK\_HCLKEN[9]) high.

### 7.5.2 Memory Space and Control

In this chip, two system memory spaces, 1st memory space is from 0x20000000 to 0x2FFFFFFF and the other memory space is from 0xA0000000 to 0xAFEEEEEE, are defined to EBI. By programming BASADDR (EBI\_BNKCTLx[31:19], x is 0, 1, 2, 3 and 4) appropriately, user could map external device to these two system memory spaces. When system request address hit the EBI's memory space, the corresponding EBI chip select assert and EBI state machine operates. The BASADDR (EBI\_BNKCTLx [31:19]) of each external I/O bank is calculated as “BASADDR” base pointer << 18. User can set M68Ex (EBI\_CTL[23:19], x is 0, 1, 2, 3 and 4) and EXBEx (EBI\_CTL[28:24] , x is 0, 1, 2, 3 and 4) to defines how the pins EBI\_nBE1, EBI\_nBE0 and EBI\_nWE are used when external bus bank accessed. Refer to the table shown below for detail information:

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

Demonstrate how to map an external SRAM to 0x20000000 memory spaces, as follows:

```
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 Ethernet MAC Controller (EMAC)

### 8.1 Overview

NUC970/N9H30 provides 2 Ethernet MAC Controller (EMAC) for Network application.

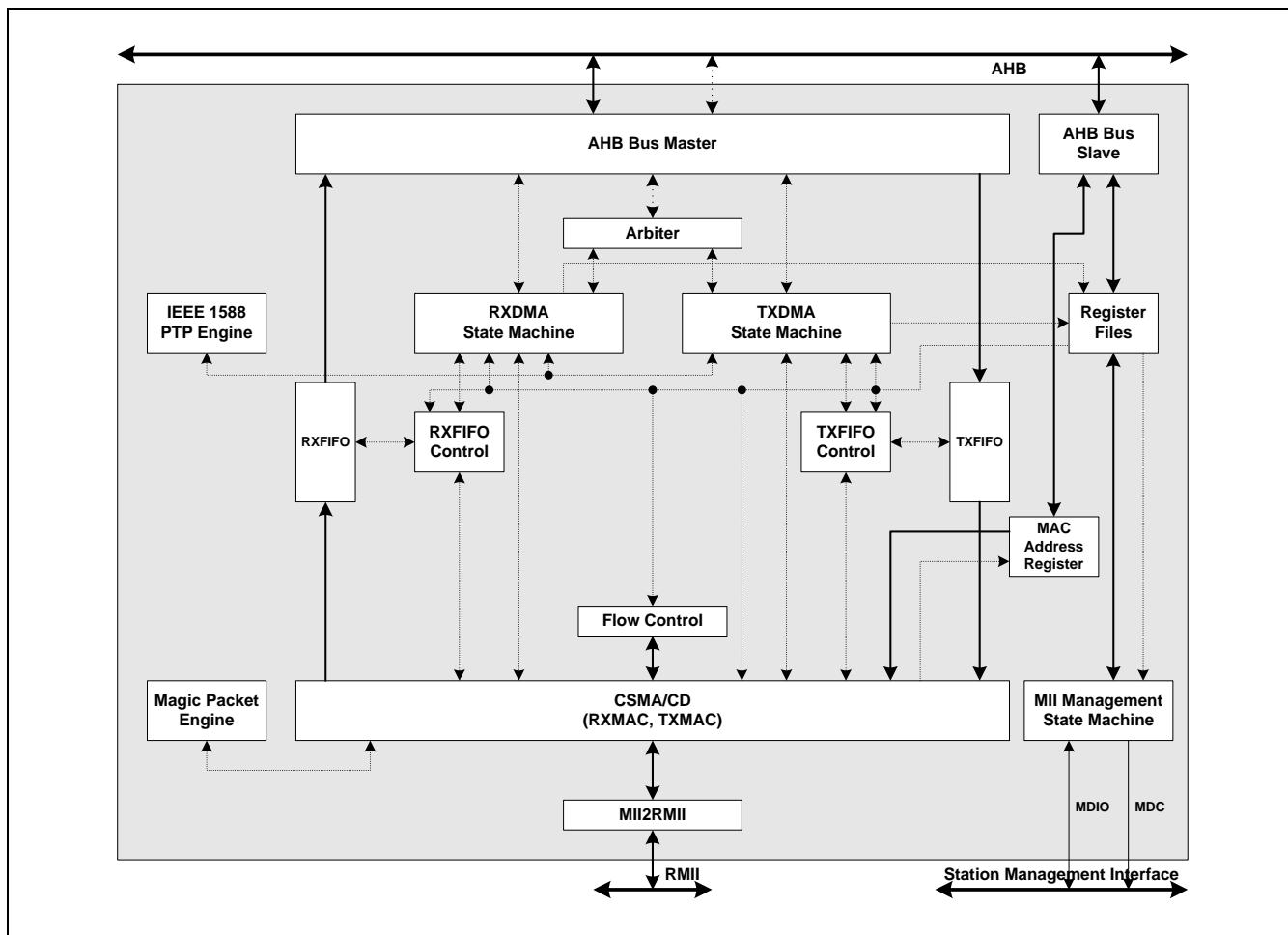
The Ethernet MAC controller consists of IEEE 802.3/Ethernet protocol engine with internal CAM function for recognizing Ethernet MAC addresses; Transmit-FIFO, Receive-FIFO, TX/RX state machine controller, time stamping engine for IEEE 1588, Magic Packet parsing engine and status controller.

The EMAC supports RMII (Reduced MII) interface to connect with external Ethernet PHY.

### 8.2 Features

- Supports IEEE Std. 802.3 CSMA/CD protocol
- Supports Ethernet frame time stamping for IEEE Std. 1588 – 2002 protocol
- Supports both half and full duplex for 10 Mbps or 100 Mbps operation
- Supports RMII interface
- Supports MII Management function to control external Ethernet PHY
- Supports pause and remote pause function for flow control
- Supports long frame (more than 1518 bytes) and short frame (less than 64 bytes) reception
- Supports 16 entries CAM function for Ethernet MAC address recognition
- Supports Magic Packet recognition to wake system up from power-down mode
- Supports 256 bytes transmit FIFO and 256 bytes receive FIFO
- Supports DMA function

### 8.3 Block Diagram



## 8.4 Register Map

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

<b>EMAC_CAM3M</b>	EMAC_BA+0x020	R/W	CAM3 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM3L</b>	EMAC_BA+0x024	R/W	CAM3 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM4M</b>	EMAC_BA+0x028	R/W	CAM4 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM4L</b>	EMAC_BA+0x02C	R/W	CAM4 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM5M</b>	EMAC_BA+0x030	R/W	CAM5 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM5L</b>	EMAC_BA+0x034	R/W	CAM5 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM6M</b>	EMAC_BA+0x038	R/W	CAM6 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM6L</b>	EMAC_BA+0x03C	R/W	CAM6 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM7M</b>	EMAC_BA+0x040	R/W	CAM7 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM7L</b>	EMAC_BA+0x044	R/W	CAM7 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM8M</b>	EMAC_BA+0x048	R/W	CAM8 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM8L</b>	EMAC_BA+0x04C	R/W	CAM8 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM9M</b>	EMAC_BA+0x050	R/W	CAM9 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM9L</b>	EMAC_BA+0x054	R/W	CAM9 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM10M</b>	EMAC_BA+0x058	R/W	CAM10 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM10L</b>	EMAC_BA+0x05C	R/W	CAM10 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM11M</b>	EMAC_BA+0x060	R/W	CAM11 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM11L</b>	EMAC_BA+0x064	R/W	CAM11 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM12M</b>	EMAC_BA+0x068	R/W	CAM12 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM12L</b>	EMAC_BA+0x06C	R/W	CAM12 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM13M</b>	EMAC_BA+0x070	R/W	CAM13 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM13L</b>	EMAC_BA+0x074	R/W	CAM13 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM14M</b>	EMAC_BA+0x078	R/W	CAM14 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM14L</b>	EMAC_BA+0x07C	R/W	CAM14 Least Significant Word Register	0x0000_0000
<b>EMAC_CAM15M</b>	EMAC_BA+0x080	R/W	CAM15 Most Significant Word Register	0x0000_0000
<b>EMAC_CAM15L</b>	EMAC_BA+0x084	R/W	CAM15 Least Significant Word Register	0x0000_0000
<b>EMAC_TXDLSA</b>	EMAC_BA+0x088	R/W	Transmit Descriptor Link List Start Address Register	0xFFFF_FFFC
<b>EMAC_RXDLSA</b>	EMAC_BA+0x08C	R/W	Receive Descriptor Link List Start Address Register	0xFFFF_FFFC
<b>EMAC_MCMDR</b>	EMAC_BA+0x090	R/W	MAC Command Register	0x0040_0000
<b>EMAC_MIID</b>	EMAC_BA+0x094	R/W	MII Management Data Register	0x0000_0000
<b>EMAC_MIIDA</b>	EMAC_BA+0x098	R/W	MII Management Control and Address Register	0x0000_0000

<b>EMAC_FFTCR</b>	EMAC_BA+0x09C	R/W	FIFO Threshold Control Register	0x0000_0000
<b>EMAC_TSDR</b>	EMAC_BA+0x0A0	W	Transmit Start Demand Register	Undefined
<b>EMAC_RSDR</b>	EMAC_BA+0x0A4	W	Receive Start Demand Register	Undefined
<b>EMAC_DMARFC</b>	EMAC_BA+0x0A8	R/W	Maximum Receive Frame Control Register	0x0000_0800
<b>EMAC_MIEN</b>	EMAC_BA+0x0AC	R/W	MAC Interrupt Enable Register	0x0000_0000
<b>EMAC_MISTA</b>	EMAC_BA+0x0B0	R/W	MAC Interrupt Status Register	0x0000_0000
<b>EMAC_MGSTA</b>	EMAC_BA+0x0B4	R/W	MAC General Status Register	0x0000_0000
<b>EMAC_MPCNT</b>	EMAC_BA+0x0B8	R/W	Missed Packet Count Register	0x0000_7FFF
<b>EMAC_MRPC</b>	EMAC_BA+0x0BC	R	MAC Receive Pause Count Register	0x0000_0000
<b>EMAC_DMARFS</b>	EMAC_BA+0x0C8	R/W	DMA Receive Frame Status Register	0x0000_0000
<b>EMAC_CTXDSA</b>	EMAC_BA+0x0CC	R	Current Transmit Descriptor Start Address Reg.	0x0000_0000
<b>EMAC_CTXBSA</b>	EMAC_BA+0x0D0	R	Current Transmit Buffer Start Address Register	0x0000_0000
<b>EMAC_CRXDSA</b>	EMAC_BA+0x0D4	R	Current Receive Descriptor Start Address Reg.	0x0000_0000
<b>EMAC_CRXBSA</b>	EMAC_BA+0x0D8	R	Current Receive Buffer Start Address Register	0x0000_0000
<b>EMAC_TSCTL</b>	EMAC_BA+0x100	R/W	Time Stamp Control Register	0x0000_0000
<b>EMAC_TSSEC</b>	EMAC_BA+0x110	R	Time Stamp Counter Second Register	0x0000_0000
<b>EMAC_TSSUBSEC</b>	EMAC_BA+0x114	R	Time Stamp Counter Sub Second Register	0x0000_0000
<b>EMAC_TSINC</b>	EMAC_BA+0x118	R/W	Time Stamp Increment Register	0x0000_0000
<b>EMAC_TSADDEND</b>	EMAC_BA+0x11C	R/W	Time Stamp Addend Register	0x0000_0000
<b>EMAC_UPDSEC</b>	EMAC_BA+0x120	R/W	Time Stamp Update Second Register	0x0000_0000
<b>EMAC_UPDSUBSEC</b>	EMAC_BA+0x124	R/W	Time Stamp Update Sub Second Register	0x0000_0000
<b>EMAC_ALMSEC</b>	EMAC_BA+0x128	R/W	Time Stamp Alarm Second Register	0x0000_0000
<b>EMAC_ALMSUBSEC</b>	EMAC_BA+0x12C	R/W	Time Stamp Alarm Sub Second Register	0x0000_0000

## 8.5 Functional Description

### 8.5.1 PHY Control

Ethernet MAC controllers read and write PHY internal registers to communicate with PHY via EMAC\_MDC and EMAC\_MDIO pins. EMAC\_MDC clock rate is AHB clock divide by MDCLK\_N (CLK\_DIVCTL8) + 1. The maximum clock setting depends on PHY's datasheet. EMAC\_MDC starts output clock after MDCON (EMAC\_MIID[19]) set 1. This clock is only

used for access PHY's registers, and not used for packet transmit or receive. So it could be stopped while not accessing PHY registers.

Both PHY's address and PHY's register address must be known to access PHY registers. PHY address may be configurable depends on PHY's power-on setting circuit. Taking IC Plus IP101G PHY as example, its PHY address is configured by the pull-up or pull-down state of PHY\_AD0, PHY\_AD1, PHY\_AD2, PHY\_AD3 pins. IEEE 802.3 defines some base PHY registers and most PHYs support these registers, so different PHYs could share a driver sometimes.

Besides these basic registers, PHYs usually has their proprietary registers as well, but the definition of these registers is varies between each PHYs. Please check PHYs' technical document for the meaning of these proprietary registers.

Below list the steps to read PHY internal registers:

1. Fill PHY address in PHYAD (EMAC\_MIIDA[12:8]) and PHY register address in PHYRAD (EMAC\_MIIDA[4:0]).
2. Set both BUSY (EMAC\_MIIDA[17]) bit and MDCON (EMAC\_MIIDA[18]) bit to 1 to send out read command.
3. Poll BUSY bit until it clear to 0.
4. Read the PHY register value from EMAC\_MIID register.

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

And below are the steps to write PHY internal registers:

1. Fill the value to program into EMAC\_MIID register.
2. Fill PHY address in PHYAD and PHY register address in PHYRAD.
3. Set WRITE (EMAC\_MIIDA[16]), BUSY, and MDCON bits to 1. This will trigger EMAC send write command to PHY.
4. Poll BUSY bit, this bit alto clear to 0 after write complete.

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

The main purpose of reading PHY registers is to get the network operating mode and speed. PHY starts Auto-Negotiation (AN) after network cable properly connected to decide to working in full duplex mode or half duplex mode, and also the speed, 10Mbps or 100Mbps. Driver need to check the value o PHY register, Auto-Negotiation Link Partner Base Page Ability to decide link partner ability and then set OPMOD (EMAC\_MCMDR[20]) and FDPU (EMAC\_MCMDR[18]) accordingly. EMAC and PHY must have the same operating mode setting to transmit and receive packet correctly.

### 8.5.2 CAM Configuration

CAM is used for Ethernet MAC address comparison, avoiding EMAC received all Ethernet packets including those destined to other machines and drag down system performance. NUC970/N9H30 built-in 16 set of CAMs. Among them, 13 (CAM0~CAM12) are actually used for address comparison. And the reset 3 sets (CAM13~CAM15) are reserved for sending control frame. ECMP (EMAC\_CAMCMR[4]), the main switch needs to be set 1 to enable CAM function. And fill the compared MAC address to one of the entries in CAM0~CAM12. For example, if the Ethernet MAC address is 00:00:00:59:16:88, and then EMAC\_CAM0M should filled with 0x00000059 and EMAC\_CAM0L will with 0x1688000000. And last, set CAM0EN(EMAC\_CAMEN[0]) to 1 and enable CAM0.

To receive broadcast packets, drivers could ether use one of the CAM entries and set EMAC\_CAMxM and EMAC\_CAMxL to 0xFFFFFFFF, 0xFFFFF0000 respectively, enable CAMxEN to receive broadcast packets. Or simply set ABP (EMAC\_CAMCMR[2]) bit to 1.

To receive multicast packets, drivers could ether use one of the CAM entries and set EMAC\_CAMxM, EMAC\_CAMxL to the mapping MAC address of the multicast IP address, enable CAMxEN to receive specific multicast packets. Or set AMP (EMAC\_CAMCMR[1]) to 1 to receive all multicast packets.

Driver could put NUC970/N9H30 Ethernet MAC into Promiscuous Mode by setting AUP (EMAC\_CAMCMR[0]), AMP (EMAC\_CAMCMR[1]), and ABP (EMAC\_CAMCMR[2]) to 1. Under this mode, all Ethernet packets will be received.

### 8.5.3 Control Frame

IEEE 802.3 defines control frame used for flow control. NUC970/N9H30 supports transmit and receive pause frame for flow control. NUC970/N9H30 will receive pause frame after ACP (EMAC\_MCMDR[3]) set 1. After received pause frame, EMAC will temporarily postpone packet transmission for a designated duration. During this period, PAU(EMAC\_MGSTA[12]) will be set 1 automatically, and clear to 0 automatically afterwards 0. While received pause frame CFR (EMAC\_MISTA[14]) will be set to 1. At the meantime, interrupt will be triggered if CFRIEN (EMAC\_MIEN[14]) is 1.

To transmit control frame, fill the MAC address 01:80:C2:00:00:01 into EMAC\_CAM13M and EMAC\_CAM13L registers, fill local MAC address into EMAC\_CAM14M and EMAC\_CAM14L registers. Fill 0x88080001 into EMAC\_CAM15M, and fill pause duration into OPERAND(EMAC\_CAM15L[31:24]). Pause time uses 512 but time as unit. Finally, set SDPZ

(EMAC\_MCMDR[16]) to 1 to send this pause frame. SPDZ automatically clears to 0 after transmit complete.

Note: Pause frame could only be used in full-duplex mode.

#### 8.5.4 Wake on Lan (WoL)

NUC970/N9H30 supports Wake on Lan feature. System could wake up from power-down state after received magic packet. Magic packet format is defined in AMD's white paper, Magic Packet Technology. It contains 6 continuous 0xFFs anywhere in the packet followed by 16 duplications of local MAC address. While both MGPWAKE(EMCA\_MCMDR[6]) and WOLIEN (EMAC\_MIEN[15]) set 1, EMAC will wake up system from power-down mode after received a magic packet its duplicated MAC address matched the address in CAM0, and set MGPR (EMAC\_MISTA[15]) to 1. Software can clear MGPR by writing 1 to it.

#### 8.5.5 Packet Receive

EMAC uses a link-list structure named as descriptors to receive Ethernet packets. Driver needs to prepare RX descriptors in advance before enabling receive function. After CAM decides a packet needs to be received, packet will be received to a memory space described in the descriptor. The status and length of received packet is recorded in the descriptor. And then EMAC will use next descriptor in the link list to receive next packet. So Rx descriptor is where CPU and EMAC used to exchange the information of received packets.

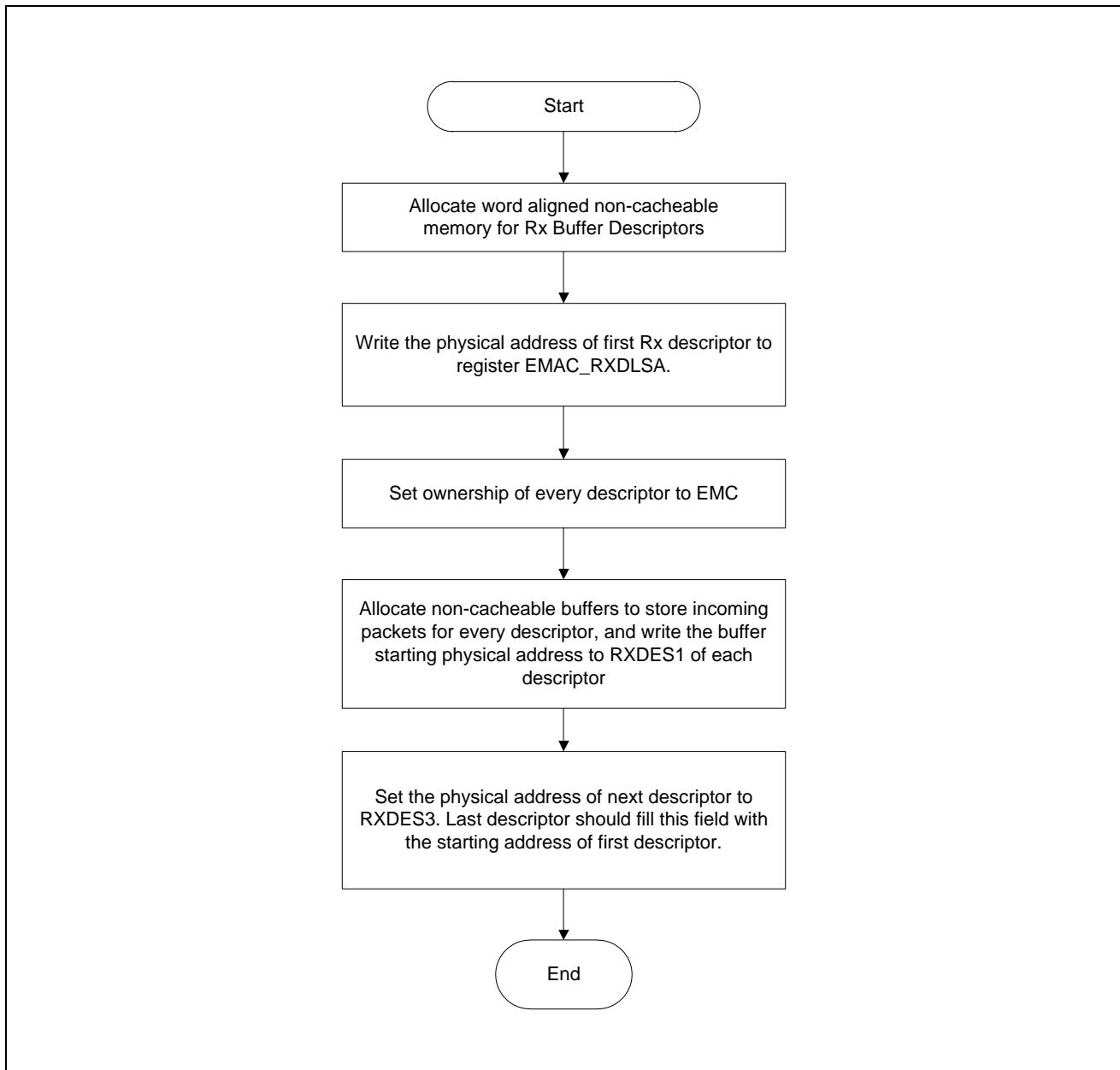
Each descriptor occupies 4 words. All descriptors form a link list. Figure below shows the structure of RX descriptor. The MSB of RXDES0, RXDES0[31] shows the current owner of this descriptor. Descriptor owner is EMAC while set 1, this means EMAC will put received packet to the address points by RXDES1, put received packet length in RXDES0 [15:0], and store received packet status to RXDES0 [30:16]. After packet received complete, EMAC automatically clears RXDES0[31] to 0, means the owner of this descriptor now switches to CPU. And EMAC will follow the link in RXDES3 to fetch next descriptor. If the owner of next descriptor is 0, then all Rx descriptor are unavailable. EMAC will stop its RX state machine until Rx descriptors' ownership given back to EMAC and RX state machine restart.

	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		

If network timestamp enabled, RXDES1 and RXDES3 will be used to store packet receive

time for user level application calculate network time. So after packet received, driver needs to restore the pointer value in RXDES1, RXDES3 before set RXDES0[31] to 1. This also means driver needs to find some memory space to backup these pointers.

After driver initialized RX descriptors, it needs to fill the starting address of first descriptor into register EMAC\_RXDLZA to notify EMAC where the descriptors are. EMAC RX state machine will start working and receive packet after driver set RXON(EMAC\_MCMDR[0]) to 1, and write any value into register EMAC\_RSDR. Following figure shows the RX descriptor initial flow.



In the sample code below, it also reserved two unsigned integer to back up the initial value of

RXDES1 and RXDES3. So the driver can restore the setting after they are overwritten by time stamp.

```
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;

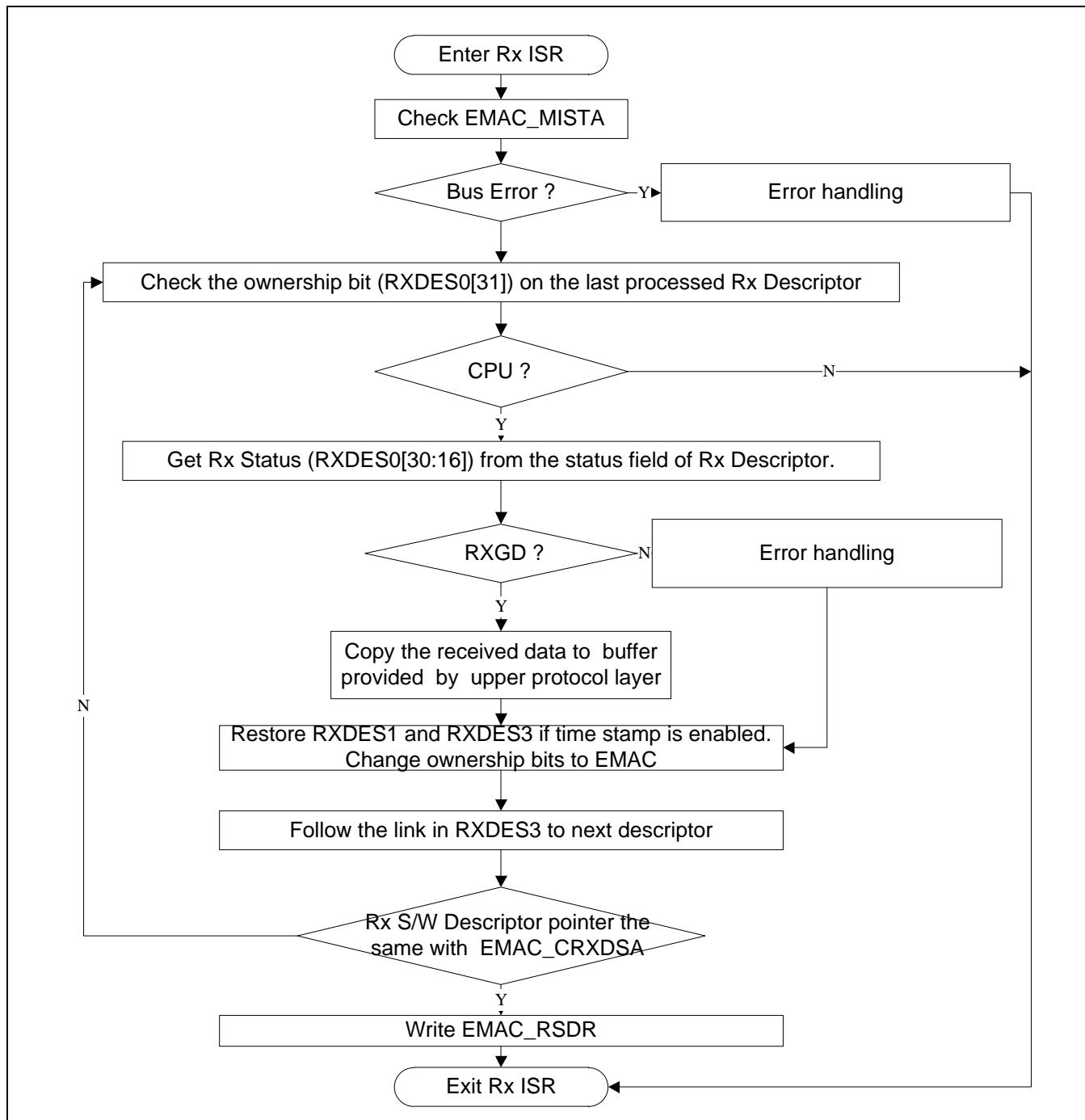
    for(i = 0; i < RX_DESC_SIZE; i++) {
        rx_desc[i].rxdes0 = (1 << 31);
        rx_desc[i].rxdes1 = (unsigned int)&rx_buf[i][0];
        rx_desc[i].backup0 = rx_desc[i].rxdes1;
        rx_desc[i].rxdes2 = 0;
        rx_desc[i].rxdes3 = (unsigned int)&rx_desc[(i + 1) % RX_DESC_SIZE];
        rx_desc[i].backup1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

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

}
```

Driver can detect packet arrival using ether polling or interrupt mode. In polling mode, driver can detect packet received by checking if RXGD (EMAC\_MISTA[4]) is set 1. At least one packet is received using Rx descriptor if RXGD is 1. To use interrupt mode, driver needs to

set both RXGDIEN (EMAC\_MIEN[4]) and RXIEN(EMAC\_MIEN[0]) to 1. EMAC will trigger interrupt whenever new packet received and also set both RXGD and RXINTR(EMAC\_MISTA[0]) to 1. RXGD and RXINTEN can both write 1 to clear them. Flow chart below shows what driver should do after RXGD set 1. If driver is working in interrupt mode, this is what driver should do in the interrupt handler.



Below is an interrupt handler sample supports time stamp function. If time stamping is not enabled, the codes to restore descriptor pointers could be omitted.

```
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 Packet Transmit

The same with packet receiving, EMAC also needs descriptors to transmit Ethernet packets. Driver needs to prepare Tx descriptor in advance. When receive a command to send out a packet from protocol stack, driver put the packet to where a pointer in descriptor points to, set the packet length in descriptor, and then trigger EMAC to transmit the packet. After transmit complete EMAC will use next descriptor to transmit packet. So Tx descriptor is where CPU and EMAC used to exchange the information of transmitted packets

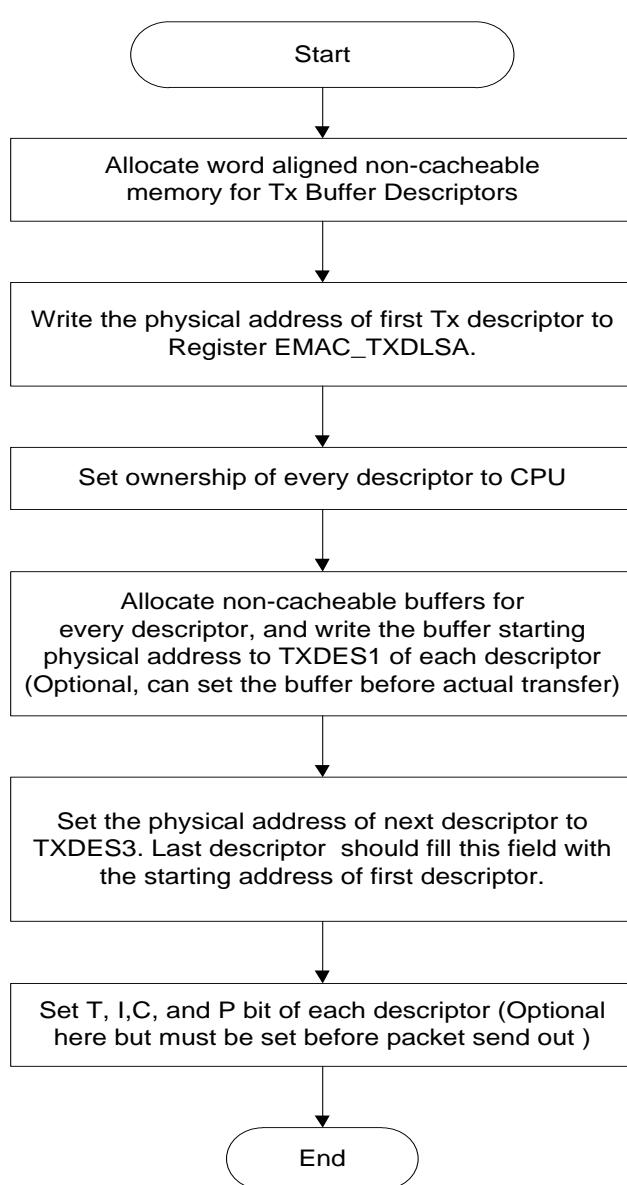
Each descriptor occupied 4 words. All descriptors form a link list. Figure below shows the structure of RX descriptor. The MSB of TXDES0, TXDES0[31] shows the current owner of this descriptor. If this bit set 1, EMAC will transmit the packet points to by TXDES1 for total TXDES2[15:0] bytes. After transmit complete, no matter success or failed, the result will be stored in RXDES2 [30:16] and TXDES0[31] will be cleared to 0, which means the packet is processed. EMAC then will follow the pointer stored in TXDES3 to fetch next Tx descriptor. If the TXDES0[31] of next descriptor is 0, it means all transmit packets are processed, no more packet need to be send. In this case, EMAC will halt its transmit state machine. But if TXDES0[31] is 1, EMAC will repeat the transmit procedure list above.

	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		

If network timestamp enabled, TXDES1 and TXDES3 will be used to store packet transmit time stamp. Aug. 14, 2018 Page 80 of 312 Rev 1.02

time for user level application calculate network time. So after packet transmitted, driver needs to restore the pointer value in TXDES1, TXDES3 before set TXDES0[31] to 1. This also means driver needs to find some memory space to backup these pointers.

After driver initialized TX descriptors, it needs to fill the starting address of first descriptor into register EMAC\_TXDLSA to notify EMAC where the descriptors are. EMAC TX state machine will start working and start transmit packet packet after driver set TXON(EMAC\_MCMDR[8]) to 1, and write any value into register EMAC\_TSDR. Following figure shows the TX descriptor initial flow.



In the TX descriptor initialize sample code below, it reserved the space for backup TXDES1 and TXDES3 initial value, so driver can restore them after overwritten by time stamp.

```
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];

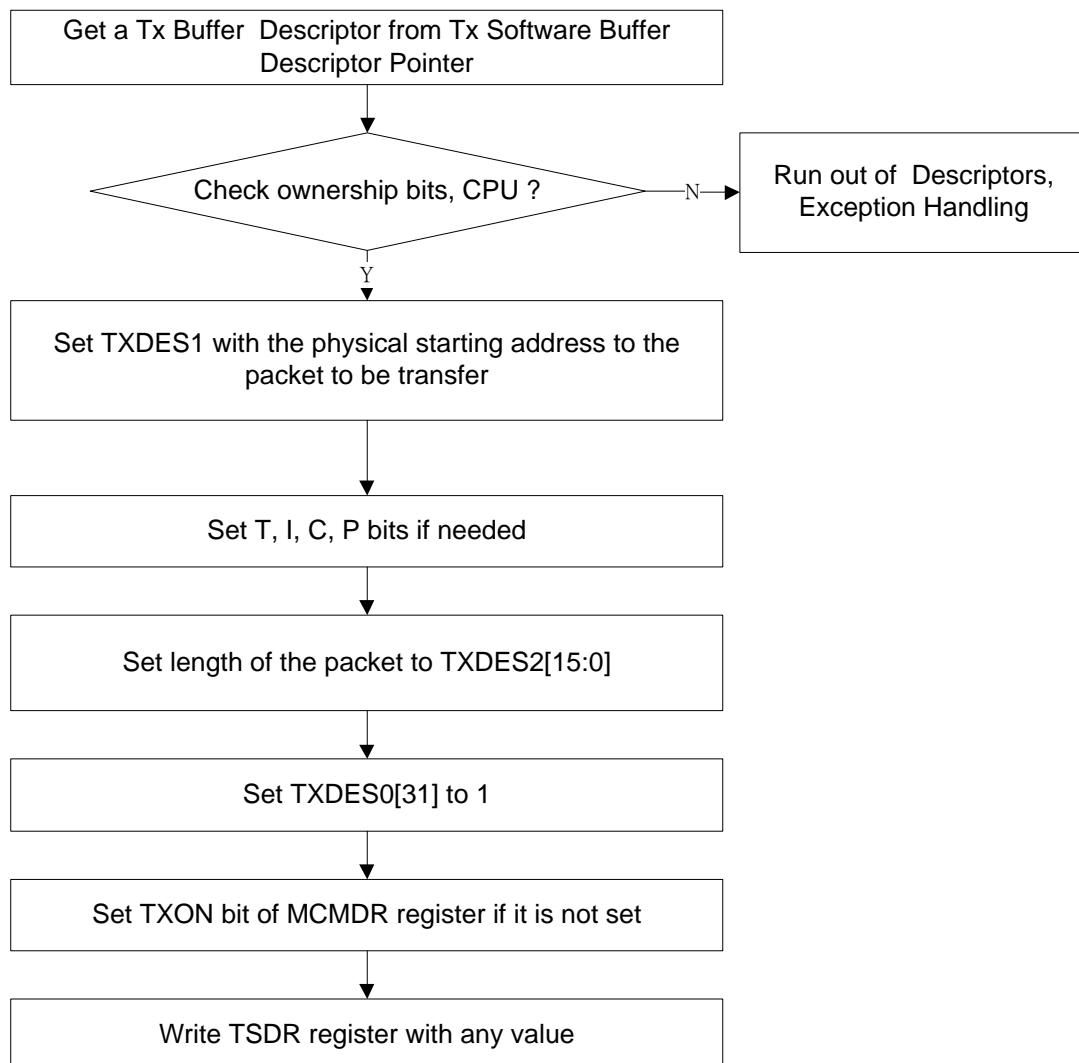
void tx_desc_init(void)
{
    unsigned int i;

    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];
}
```

Except the descriptor setting mentions previously, TXDES0[3:0] also needs to be configured to send a packet. TTSEN(TXDES0[3]) is used to enable time stamp function. If this bit set to 1, the network time after transmit complete will be recorded in TXDES1 and TXDES3. INTEN(TXDES0[2]) used to configure if interrupt should be triggered after transmit this packet. EMAC only triggers interrupt if this bit is 1 and the setting in EMAC\_MISTA enabled transmit interrupt. CRCAPP(TXDES0[1]) controls if EMAC calculate the CRC for transmitted

packet or not. This bit should set to 1 in normal operation. Minimum Ethernet packet size is 60 bytes (without 4 bytes CRC). EMAC will help to pad packet to 60 bytes if the packet length is shorter than 60 bytes if PADEN (TXDES0[1]) is set to 1. This bit should set to 1 during normal operation. Following flow chart shows the network transmit procedure with time stamping enabled.



Below is a sample code shows the procedure to send a packet.

```
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);

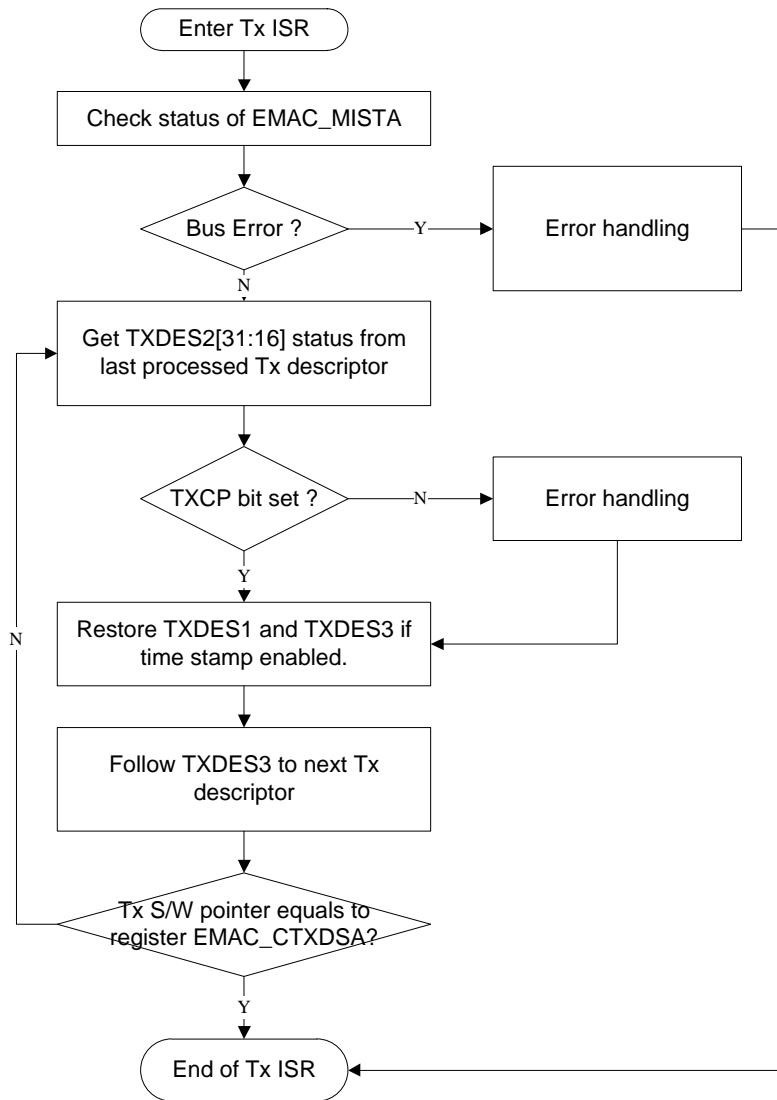
    // Set TX Frame flag & Length Field
    desc->txdes0 |= (P | C | I | T);
    desc->txdes2 = size;

    // 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;
}
```

Network transmit result could be checked by polling mode or interrupt mode. In polling mode, driver checks TXCP (EMAC\_MISTA[18]) bit. Whenever this bit set 1, at least one packet was transmitted, no matter success or not. In interrupt mode, both TXCPIEN (EMAC\_MIEN[18]) and TXIEN(EMAC\_MIEN[16]) needs to be set 1. Whenever packet transmit complete, EMAC will trigger interrupt, and set both TXCP and TXINTR(EMAC\_MISTA[16]) to 1. These two bits, TXCP and TXINTR could be cleared by writing 1 to them. Following figure shows what driver should do after TXCP set 1. In interrupt mode, these are what interrupt handler should do.



Below is an interrupt handler sample that supports time stamping function.

```

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 Network Timing

To support more accurate IEEE1588 network timing on NUC970/N9H30, both EMAC built in time stamping module. The time stamping module could record the exact packet received and

transmitted time and reduces the bias error if time stamp get by software in interrupt handler. The time stamping modules update their time every EMAC clock. So it is 150MHz at most, which is the finest clock in this system. Time stamp modules supports two update methods, fine update and cores update, which is configurable by TSMODE (EMAC\_TSCTL[2]) bit. Clear 0 to select cores update, set 1 to select fine update.

NUC970/N9H30 uses second and sub-second as time unit in time stamp module. Current time increased 1 second every time sub-second overflows. In cores update mode, current sub-second value increased by the value stored in register EMAC\_TSINC on every EMAC clock tick.

Taking EMAC clock frequency 150MHz as example to calculate the relative registers setting here. Since the clock rate is 150MHz, sub-second field should overflow every 150M clock tick to increase the second field. Sub-second register use 31 bits to store sub-second, so EMAC\_TSINC should fill with  $(2^{31}) / 150M = 14.31 \approx 14 = 0x0E$ , this is the sub-second value should increase on every EMAC clock tick. But if 0x0E is used, clock bias will be  $0.31/14 = 2.2\%$  which is impractical to use. So it is not recommend using cores update while EMAC clock is 150MHz.

In fine update mode, an internal 32-bit counter will add the value stored in register EMAC\_TSADDNED on every EMAC clock tick. When this counter overflows, sub-second value will increase the value stored in register EMAC\_TSINC. So fine update mode is more accurate than cores update mode.

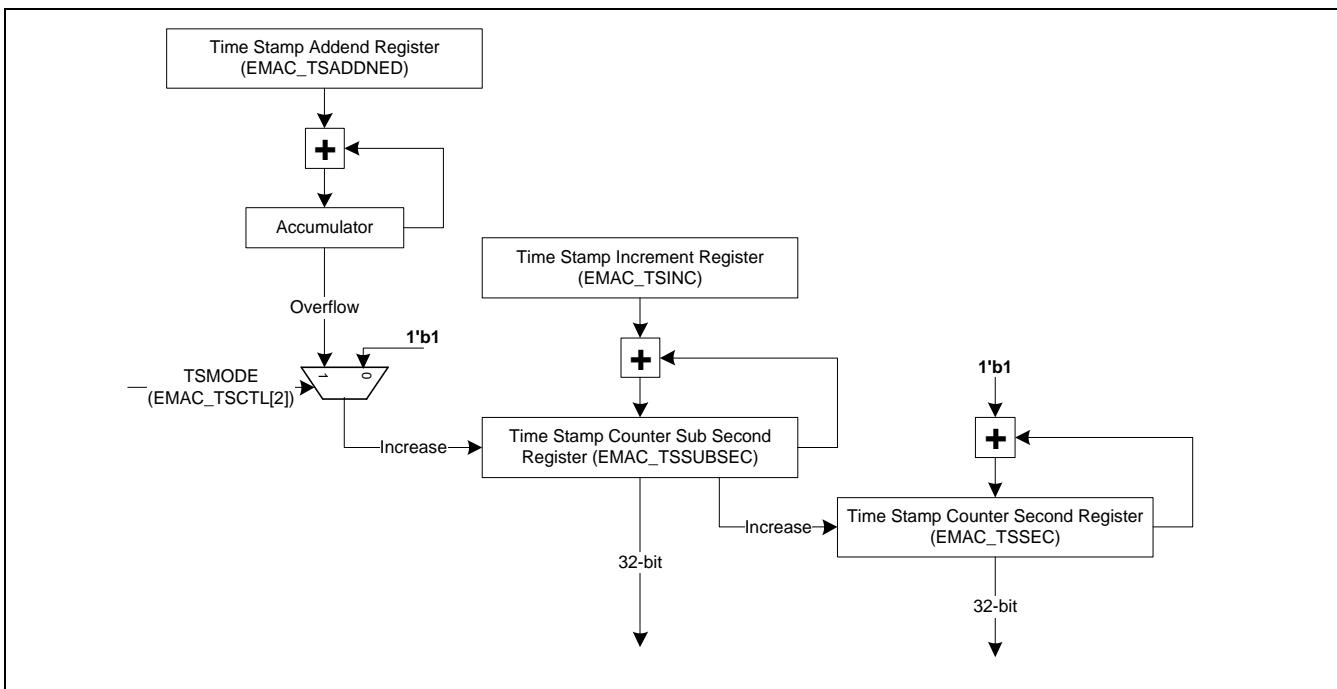
Here use EMAC clock frequency 150MHz as condition to calculate time stamp registers setting in fine update mode. Assuming we want to increase EMAC\_TSINC value every 100ns, so sub-second needs to overflow after added for  $10^7$  times, so EMAC\_TSINC needs to fill in  $2^{31} / 10^7 = 214.71 \approx 215 = 0xD7$ . The actual overflow frequency is not exact  $10^7$  Hz as expected but  $2^{31} / 215$  Hz. So needs to fill EMAC\_TSADDNED with a value that makes counter overflow at the frequency of  $2^{31} / 215$  Hz to make timing accurate. This means value fill to EMAC\_TSADDNED has to be  $2^{32} * (2^{31} / 215) / 150M = 285996032.15 \approx 285996032 = 0x110BF400$ . In this case, the bias error is  $5.26 * 10^{-10}$ , which is much better comparing with cores update. It is recommended using fine update mode for timing update.

Based on the calculation above, the setting of EMAC\_TSINC and EMAC\_TSADDEND while EMAC clock is 150MHz listed below. (Different from cores update mode, the setting is not the only valid value. But different EMAC\_TSINC needs to use different EMAC\_TSADDNED value):

EMAC\_TSINC = 0xD7;

EMAC\_TSADDEND = 0x110BF400;

Following figure shows network timestamp update block diagram.



In sub-second operation, every overflow (bit 31 becomes 1) means 1 second elapsed. In other words, every  $2^{31}$  sub-second is 1 second or  $10^9$  nanoseconds. The functions below show how to convert between sub-second and nanosecond using the calculation above.

```

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

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

Steps toward network timestamp initialization listed below:

1. Set TSEN(EMAC\_TSCTL[0]) to 1, enable network timestamp circuit.
2. Fill initial second and sub-second value into EMAC\_TSSEC and EMAC\_TSSUBSEC registers
3. Configure EMAC\_TSINC register, and configure EMAC\_TSADDEND to use fine update mode.
4. Set TSIEN (EMAC\_TSCTL[1]) 1 to start network timestamp counting, to use fine update, set TSMODE (EMAC\_TSCTL[2]) to 1 too.

According to IEEE 1588 specification, when a device has multiple network interfaces, they must share the same clock source. So if timestamp modules on both NUC970/N9H30's EMACs are enabled, driver must set PTP\_SRC (EMAC\_MCMDR[7]) of EMAC1 to 1, this will let EMAC1 use EMAC0's timestamp module instead of using its own timestamp module.

Software can read current network time via registers. Current network time is stored in two 32-bit registers, EMAC\_TSSEC and EMAC\_TSSUBSEC. There is a circuit to avoid the situation that sub-second overflow while reading sub-second register. While read EMAC\_TSSUBSEC register, the current second value will be locked in EMAC\_TSSEC register at the same time, to avoid software uses incorrect time value for operation. Below is a sample code for reading current time value:

```
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));
```

Software can adjust current network time after time stamp module initialized. To maintain accurate timing, network time adjust current time using a offset value. For example, if current time is 3 second too fast, the adjust method does not require a read-modify-write. The executing time of software also needs to take into consideration and it could be affected by some factors. This unpredictable brings error to current time, and has bad impact to PTP which requires precise timing. The time stamp module can add or subtract an offset from current time. Second level offset fills into register EMAC\_UPDSEC, sub-second level offset fills into EMAC\_UPDSUBSEC[30:0]. If this is positive offset, EMAC\_UPDSUBSEC[31] keep 0, on the contrary set EMAC\_UPDSUBSEC[31] to 1 for negative offset. After both EMAC\_UPDSEC and EMAC\_UPDSUBSEC fill with proper offset setting, set TSUPDATE(EMAC\_TSCTL[3]) to 1 to trigger time stamp module to update network time. This bit auto clears to 0 after update complete.

Time stamp module also has alarm feature. Alarm trigger time fills in EMAC\_ALMSEC and EMAC\_ALMSUBSEC registers. They store the alarm trigger second and sub-second respectively. After alarm time configured, set TSALMEN (EMAC\_TSCTL[5]) to 1 to enable alarm function. If TSALMIEN (EMAC\_MIEN[28]) is 1, an interrupt will be triggered when alarm

occurs and TSALS(EMAC\_MISTA[28]) will be set 1. Software can write 1 to clear this bit.  
Note: This interrupt is designed as a Tx interrupt, and needs to be processed in Tx interrupt handler instead of Rx interrupt handler.

### 8.5.8 Error Handling

Some status bits in EMAC\_MISTA register reflect status error during normal operation. Following table lists error status and the solutions.

Error Bit Name	Bit Number	Status Description	Solution
TXBERR	24	Transmit bus error	Check driver. This error flag can only be triggered when EMAC follows incorrect pointer TX descriptor in to fetch data.
TDU	23	Transmit description unavailable	Do not need to take action. This flag means no more packets need to be sent.
TXABT	21	Transmit abort	Probably caused by heavy network loading.
TXEMP	17	Transmit FIFO unavailable	If this flag set frequently, set TXTHD(EMAC_FFTCR[9:8]) to a higher trigger level.
RXBERR	11	Receive bus error	Check driver. This error flag can only be triggered when EMAC follows incorrect pointer RX descriptor in to fetch data.
RDU	10	Receive description unavailable	This flag set 1 because software cannot process received packets in RX descriptor thus EMAC has no more free descriptor to receive further incoming packets.  To trigger EMAC RX state machine to receive packet after RDU state occurred, software needs to receive all received packets, set ownership of RX descriptors to EMAC, and write any value into EMAC_RSDR.  Packet comes in too fast before driver can process them, or RX interrupt blocked too long and could not get a chance to execute before RX descriptors runs out can both trigger this state.
RP	6	Received short packet ( < 64 bytes)	Simply drop this packet. Does not occur during normal operation unless ARP (EMAC_MCMDR[2]) set 1.
ALIE	5	Alignment error	Should not occur during normal operation. Please check RMII related circuit on PCB board or try another Ethernet cable if this flag set frequently.
PTLE	3	Received long packet ( > 1518 bytes)	Simply drop this packet. Does not occur during normal operation unless ALP (EMAC_MCMDR[1]) set 1.
RXOV	2	Receive FIFO overflow	If this flag set frequently, set RXTHD(EMAC_FFTCR[1:0]) to a higher trigger level.
CRCE	1	CRC error	Simply drop this packet. Does not occur during normal operation unless AEP (EMAC_MCMDR[4]) set 1.

## 9 Enhanced Timer Controller (ETMR)

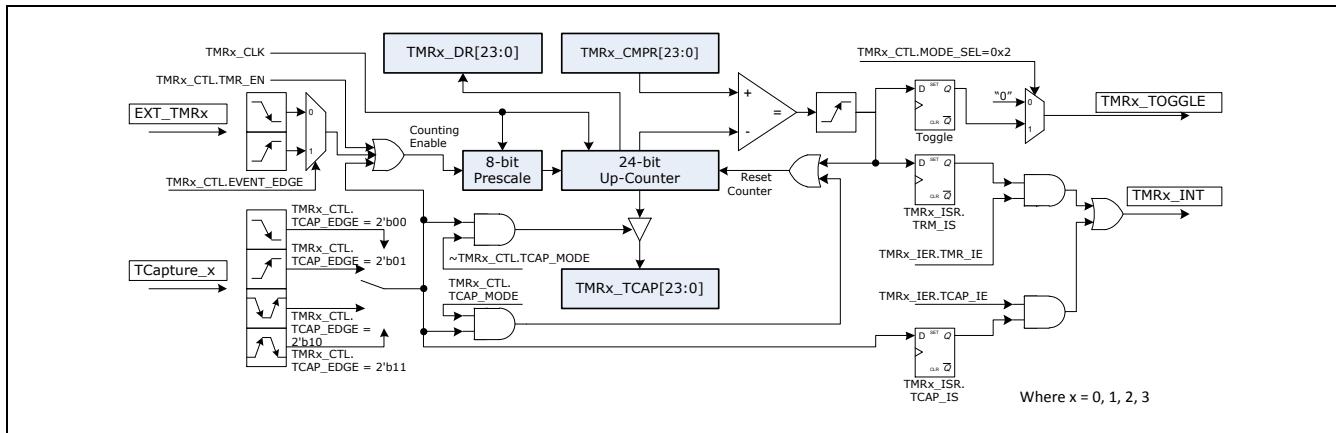
### 9.1 Overview

This chip is equipped with four enhance timer modules including ETIMER0, ETIMER1, ETIMER2 and ETIMER3, which allow user to easily implement a counting scheme or timing control for applications. The timer can perform functions like frequency measurement, event counting, interval measurement, clock generation, delay timing, and so on. The timer can generate an interrupt signal upon timeout, or provide the current value of count during operation.

### 9.2 Features

- Independent Clock Enable Control for each Timer (ECLKetmr0, ECLKetmr1, ECLKetmr2 and ECLKetmr3)
- Time-out period = (Period of timer clock input) \* (8-bit pre-scale counter + 1) \* (24-bit TCMP)
- Counting cycle time =  $(1 / \text{ECLKetmr}) * (2^8) * (2^{24})$
- Internal 8-bit pre-scale counter
- Internal 24-bit up counter is readable through TDR (Timer Data Register)
- Supports One-shot, Periodic, Output Toggle and Continuous Counting Operation mode
- Supports external pin capture for interval measurement
- Supports external pin capture for timer counter reset

### 9.3 Block Diagram



### 9.4 Register Map

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

## 9.5 Functional Description

Enhanced timer supports one-shot, periodic, toggle out, and continuous operation mode. And it also supports external capture functions to measure input signal frequency or reset counter.

### 9.5.1 Timer Initialization

Below list the procedure to initialize timer counter and start counting:

1. Stop timer counting by clear ETMR\_EN (ETMRx\_CTL[0]) to 0.
2. Configure MODE\_SEL (ETMR\_CTL[5:4]) to set operating mode.
3. Set ETMR\_IE (ETMRx\_IER[0]) to 1 for enable interrupt, otherwise clear to 0.
4. Set prescaler in PRESCALE\_CNT (ETMRx\_PRECNT[7:0]).
5. Set timer compare value in ETMR\_CMP (ETMRx\_CMPR[24:0]).
6. Set ETMR\_EN (ETMRx\_CTL[0]) 1 to enable timer counting.

### 9.5.2 Timer Capture Initialization

Below list the procedure to initialize timer capture mode:

1. Stop timer by clear both ETMR\_EN (ETMRx\_CTL[0]) and TCAP\_EN (ETMRx\_CTL[16]) to 0.
2. Configure MODE\_SEL (ETMRx\_CTL[5:4]) to set operating mode.
3. Set TCAP\_IE (ETMRx\_IER[1]) 1 to enable capture interrupt, otherwise clear to 0.
4. Configure TCAP\_MODE(ETMRx\_CTL[17]) and CAP\_CNT\_MOD to select capture mode.
5. Configure TCAP\_EDGE to select capture trigger condition.
6. To enable capture debounce, set TCAP\_DEB\_EN(ETMRx\_CTL[22]) to 1, Otherwise clear to 0.
7. Set prescaler in PRESCALE\_CNT (ETMRx\_PRECNT[7:0])
8. Set timer compare value in ETMR\_CMP (ETMRx\_CMPPR[24:0])
9. Set TCAP\_EN (ETMRx\_CTL[16]) 1 to enable capture mode.
10. Set ETMR EN (ETMRx\_CTL[0]) 1 to enable counter.

Note: MODE\_SEL setting is ignored in trigger counting mode. This setting only affect timer operating mode in free counting mode and counter reset mode.

### 9.5.3 Interrupt Handling

Every timer has individual interrupt source and interrupt could be ether timeout interrupt or capture interrupt. While timeout interrupt triggers, ETMR\_IS(ETMRx\_ISR[0]) will be set 1. While capture interrupt triggers, TCAP\_IS(ETMRx\_ISR[1]) will be set 1. These two bits could be cleared by write 1 to them.

If new capture event occurs before TCAP\_IS cleared in capture mode, NCAP\_DET\_STS (ETMRx\_ISR[5]) will be set 1. This bit will be cleared after when TCAP\_IS cleared.

### 9.5.4 Timer Frequency

Fulmar below can be used to calculate timer timeout frequency:

$$\text{Frequency} = \text{TMRx\_CLK} / ((\text{PRESCALE} + 1) * \text{TCMP})$$

Where TMRx\_CLK is the timer clock source frequency. Could be HXT (12MHz external frequency), PCLK, PCLK/4029, or LXT (32.768kHz external crystal). PRESCALE prescaler defined in PRESCALE\_CNT (ETMRx\_PRECNT[7:0]), TCMP timer compare value defined in ETMR\_CMP (ETMRx\_CMPPR[24:0]). Following table shows some example of timer setting to generate 1Hz, 10Hz, 100Hz, and 1000Hz frequency.

Timer Frequency	Timer Clock Source	PRESCALE_CNT (ETMRx_PRECNT[7:0])	ETMR_CMP (ETMRx_CMPPR[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 One-Shot Mode

If the timer is operated in One-shot mode (MODE\_SEL[1:0] is 00) and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, the ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1. If ETMR\_IE (ETMRx\_IER[0] timer interrupt enable bit) is set to 1 then the interrupt signal is generated and sent to AIC to inform CPU for indicating that the timer counting overflow happens. If ETMR\_IE (ETMRx\_IER[0] timer interrupt enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1, timer counting operation stops and the timer counter value (ETMRx\_DR value) goes back to counting initial value then ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is cleared to 0 by timer controller automatically. That is to say, timer operates timer counting and compares with ETMRx\_CMPR value function only one time after programming the timer compare register (ETMRx\_CMPR) value and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is set to 1. So, this operating mode is called One-Shot mode.

### 9.5.6 Periodic Mode

If the timer is operated in Periodic mode (MODE\_SEL[1:0] is 01) and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, the ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1. If ETMR\_IE (ETMRx\_IER[0] timer interrupt enable bit) is set to 1 then the interrupt signal is generated and sent to AIC to inform CPU for indicating that the timer counting overflow happens. If ETMR\_IE (ETMRx\_IER[0] timer interrupt enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1, the timer counter value (ETMRx\_DR value) goes back to counting initial value and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is kept at 1 (counting enable continuously) and timer counter operates up counting again. If ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) is cleared by software, once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value again, ETMR\_IS (ETMRx\_ISR[0]

timer interrupt status) will set to 1 also. That is to say, timer operates timer counting and compares with ETMRx\_CMPR value function periodically. The timer counting operation does not stop until the ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is set to 0. The interrupt signal is also generated periodically. So, this operating mode is called Periodic mode.

### 9.5.7 Toggle Mode

If the timer is operated in Toggle mode (MODE\_SEL[1:0] is 10) and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, the ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1. If ETMR\_IE (ETMRx\_IER[0] timer interrupt enable bit) is set to 1 then the interrupt signal is generated and sent to AIC to inform CPU for indicating that the timer counting overflow happens. If ETMR\_IE (ETMRx\_IER[0] timer interrupt enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) and toggle out signal will set to 1, the timer counter value (ETMRx\_DR value) goes back to counting initial value and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is still kept at 1 (counting enable continuously), and timer counter operates up counting again. When the timer counter value (ETMRx\_DR value) reaches timer compare register value again, toggle out signal is set to 0, and ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1 also. The timer counting operation does not stop until the ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is set to 0. Thus, the toggle output signal changes back and forth with 50% duty cycle. So, this operating mode is called Toggle mode.

### 9.5.8 Continuous Mode

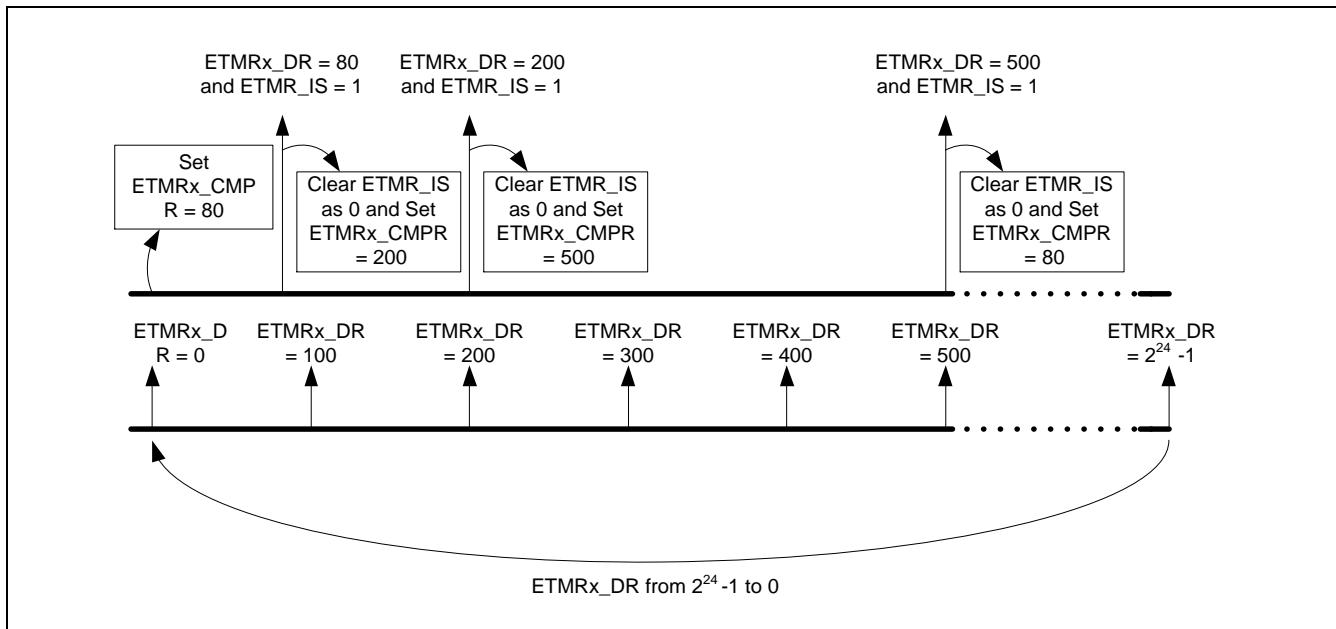
If the timer is operated in Continuous Counting mode (MODE\_SEL[1:0] is 11) and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, the ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1. If ETMR\_IE (ETMRx\_IER[0] timer counter enable bit) is set to 1 then the interrupt signal is generated and sent to AIC to inform CPU for indicating that the timer counting overflow happens. If ETMR\_IE (ETMRx\_IER[0] timer counter enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (ETMRx\_DR value) reaches timer compare register (ETMRx\_CMPR) value, ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1 and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is kept at 1 (counting enable continuously) and timer counter continuous counting without reload the timer counter value (ETMRx\_DR value) to counting initial value. User can change different timer compare register (ETMRx\_CMPR) value immediately without disabling timer counter and restarting timer counter counting.

For example, the timer compare register (ETMRx\_CMPR) value is set as 80, first. (The timer compare register (ETMRx\_CMPR) should be less than  $2^{24}$  and be greater than 1). Once the

timer counter value (ETMRx\_DR value) reaches to 80, ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1 and ETMR\_EN (ETMRx\_CTL[0] timer counter enable bit) is still kept at 1 (counting enable continuously). Next, user clears the ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) and reprograms timer compare register (ETMRx\_CMPR) value as 200, then ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1 again when timer counter value (ETMRx\_DR value) reaches to 200. At last, user clears ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) and reprograms timer compare register (ETMRx\_CMPR) value as 500, then ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) will set to 1 again when timer counter value (ETMRx\_DR value) reaches to 500. In this mode, when the timer counter value (ETMRx\_DR value) continues counting up to  $2^{24} - 1$ , then recount up from 0 continuously. The timer counter value (ETMRx\_DR value) is always keeping up counting even if ETMR\_IS (ETMRx\_ISR[0] timer interrupt status) is 1. Therefore, this operation mode is called as Continuous Counting mode.

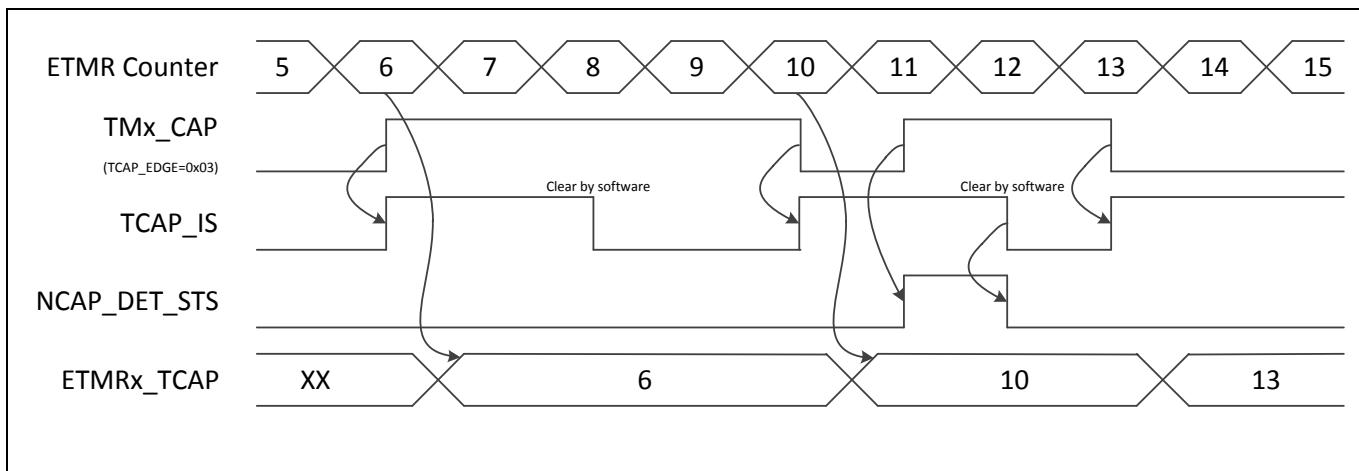
Following figure shows an enhanced timer continuous mode sample.



### 9.5.9 Free Counting Mode

In this mode, timer monitors the capture pin toggle event to save current counter value. If both TCAP\_MODE (ETMRx\_CTL[17]) and CAP\_CNT\_MOD (ETMRx\_CTL[20]) is 0, timer is working in free counting mode. 24 up counter keeps counting, when the external pin toggle state matches the setting in TCAP\_EDGE (ETMRx\_CTL[19:18]), current 24 up counter value will be stored in TMRx\_TCAP register. At the mean time, if TCAP\_IE (ETMRx\_IER[1]) is 1, TCAP\_IS(ETMR\_IS[1]) will set 1 and trigger interrupt.

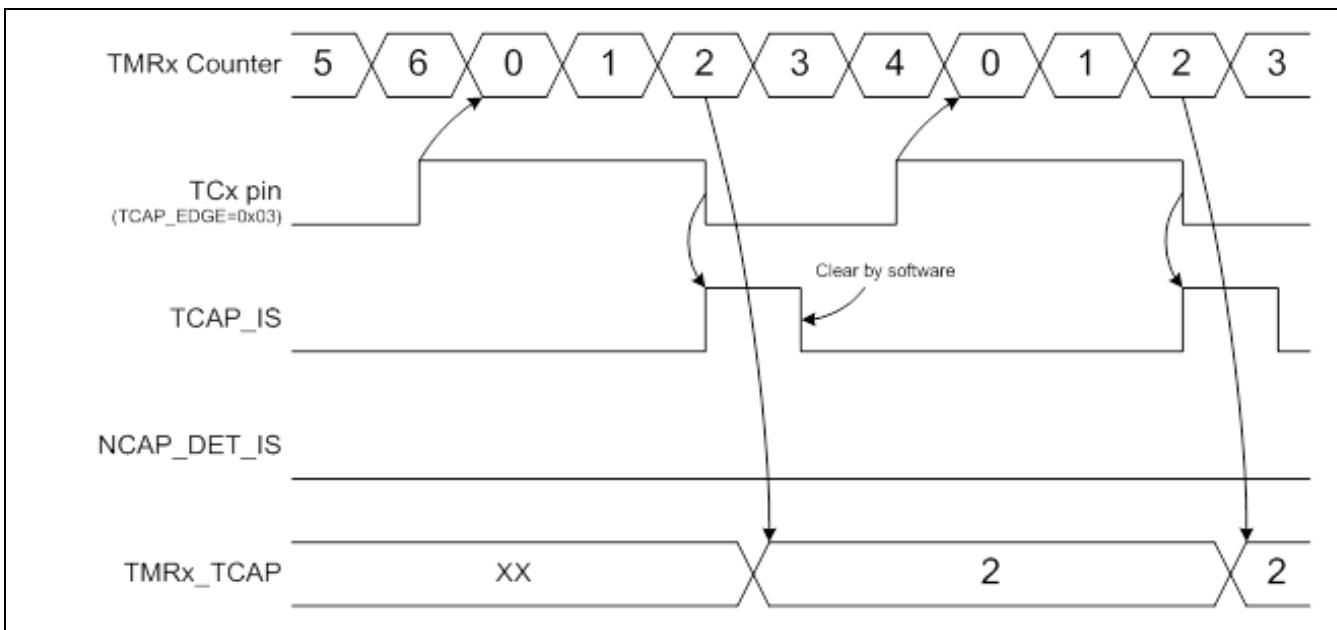
In free counting mode, when TCAP\_EDGE is 0, falling edge on TMx\_CAP triggers capture event. When TCAP\_EDGE is 1, rising edge on TMx\_CAP triggers capture event. And both falling and rising edge trigger capture event if TCAP\_EDGE is ether 2 or 3, TMx\_CAP. Following figure is timing diagram of free counting mode.



### 9.5.10 Trigger Counting Mode

In this mode, timer monitors the capture pin toggle event to start/stop timer counter and save captured value. If TCAP\_MODE (ETMRx\_CTL[17]) is 0 and CAP\_CNT\_MOD (ETMRx\_CTL[20]) is 1, counter will work in trigger counting mode. 24 up counting counter will keep 0. Until external capture pin toggle state matches TCAP\_EDGE (ETMRx\_CTL[19:18]) first trigger condition, 24 counter starts counting. And timer counter stops counting and store current counter value to ETMRx\_TCAP register. When the external capture pin toggle state matches the second trigger condition set in TCAP\_EDGE. If TCAP\_IE (ETMRx\_IER[1]) is 1, TCAP\_IS(ETMR\_ISR[1]) will be set 1 and triggers interrupt.

In trigger counting mode, if TCAP\_EDGE is 0, first falling edge on TMx\_CAP starts timer up counting, second falling edge stops timer counter. If TCAP\_EDGE is 1, first rising edge on TMx\_CAP starts timer up counting, second rising edge stops timer counter. If TCAP\_EDGE is 2, falling edge on TMx\_CAP starts timer counter, and rising edge stops counter. If TCAP\_EDGE is 3, rising edge on TMx\_CAP starts timer counter, and falling edge stops counter.

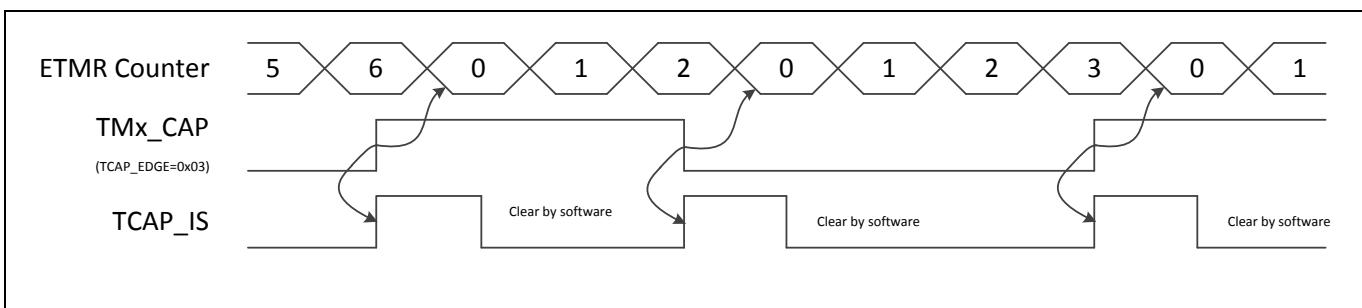


### 9.5.11 Counter Reset Mode

In this mode, timer monitors the capture pin toggle event to reset timer counter. The timer value before reset will not be saved.

External capture toggle pin will be used to reset timer counter if TCAP\_MODE (ETMRx\_CTL[17]) is 1. In this mode, while external capture pin toggle status matches the setting in TCAP\_EDGE (ETMRx\_CTL[19:18]), timer counter will be reset and keep up counting. If TCAP\_IE (ETMRx\_IER[1]) is 1, TCAP\_IS(ETMRx\_ISR[1]) will be set 1 and trigger interrupt.

In counter reset mode, if TCAP\_EDGE is 0, falling edge on TMx\_CAP pin will reset timer counter. Rising edge on TMx\_CAP reset counter if TCAP\_EDGE is 1, TMx\_CAP. Both rising and falling edge reset timer counter if TCAP\_EDGE if 2 or 3, TMx\_CAP. Following figure illustrate the reset timer mode operation.



### 9.5.12 Capture Debounce

Timer capture supports debounce function for detecting capture pin toggle. Timer can use either original signal or debounced signal to detect capture pin status. Default state of debounce circuit is disabled. And only will be enabled if both TCAP\_DEB\_EN (ETMRx\_CTL[22]) and TCAP\_EN (ETMRX\_CTL[16]) are set 1. So if capture pin level is 1, and TCAP\_EDGE (ETMRx\_CTL[19:18]) configured to detect rising level and debounce circuit enabled, a false rising event will be detected. This will means the value in ETMRx\_TCAP is incorrect on first capture interrupt. To avoid using this wrong value for frequency calculation, it is recommended to ignore first capture data that maybe incorrect capture value.

## 10 Flash Memory Interface

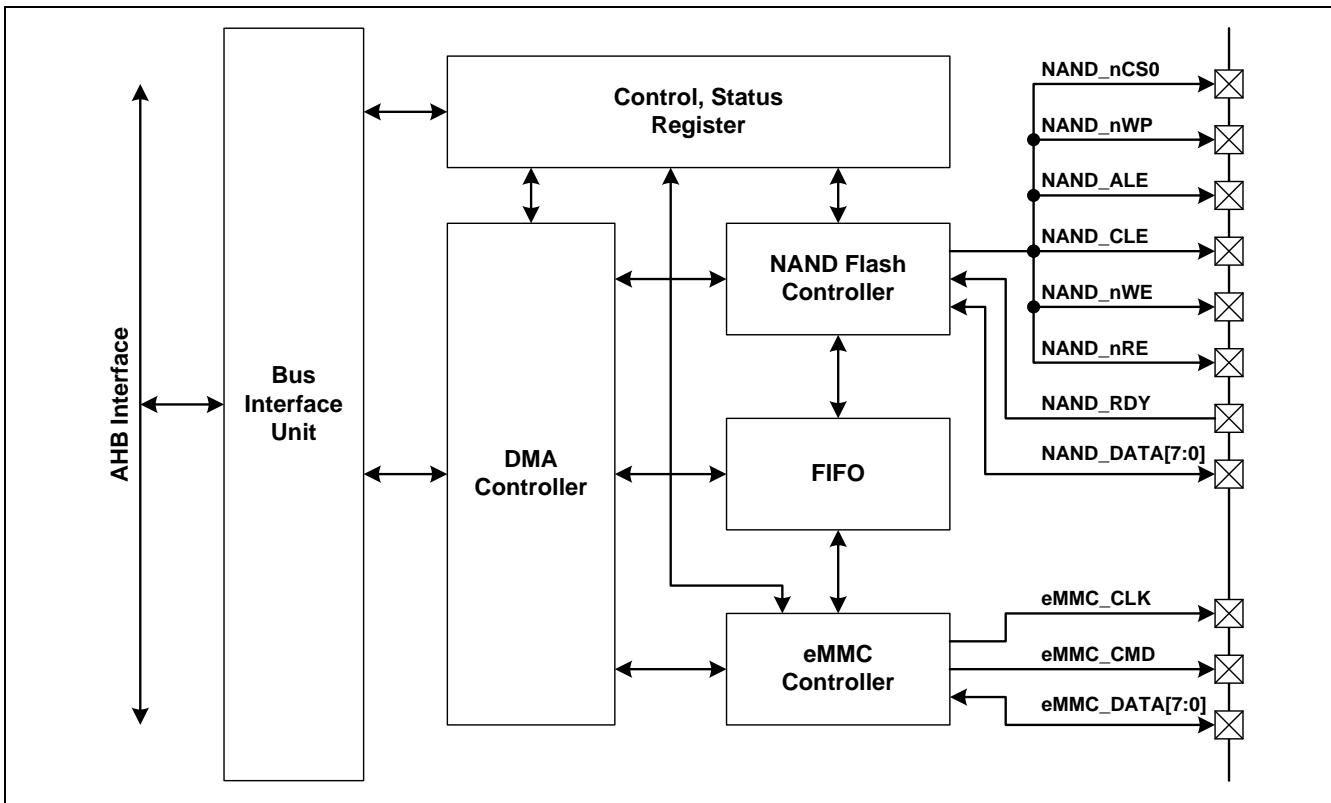
### 10.1 Overview

The Flash Memory Interface (FMI) of this Chip has DMA unit and FMI unit. The DMA unit provides a DMA (Direct Memory Access) function for FMI to exchange data between system memory (ex. SDRAM) and shared buffer (128 bytes), and the FMI unit control the interface of eMMC or NAND flash. The interface controller can support eMMC and NAND-type flash and the FMI is cooperated with DMAC to provide a fast data transfer between system memory and cards.

### 10.2 Features

- Support single DMA channel and address in non-word boundary
- Support hardware Scatter-Gather function
- Support 128Bytes shared buffer for data exchange between system memory and flash device. (Separate into two 64 bytes ping pong FIFO)
- Support eMMC Flash device
- Supports SLC and MLC NAND type Flash
- Adjustable NAND page sizes. (512B+spare area, 2048B+spare area, 4096B+spare area and 8192B+spare area)
- Support up to 4bit/8bit/12bit/15bit/24bit hardware ECC calculation circuit to protect data communication
- Support programmable NAND timing cycle

### 10.3 Block Diagram



## 10.4 Register Map

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

Register	Offset	R/W	Description	Reset Value
<b>FMI_BA = 0xB000_D000</b>				
<b>FMI_BUFFERn</b> <i>n = 0, 1..31</i>	FMI_BA+0x000+0x4*n	R/W	FMI Embedded Buffer Word n N = 0, 1..31	0x0000_0000
<b>FMI_DMACTL</b>	FMI_BA+0x400	R/W	FMI DMA Control Register	0x0000_0000
<b>FMI_DMASA</b>	FMI_BA+0x408	R/W	FMI DMA Transfer Starting Address Register	0x0000_0000
<b>FMI_DMABCNT</b>	FMI_BA+0x40C	R	FMI DMA Transfer Byte Count Register	0x0000_0000
<b>FMI_DMAINTEN</b>	FMI_BA+0x410	R/W	FMI DMA Interrupt Enable Register	0x0000_0001
<b>FMI_DMAINTSTS</b>	FMI_BA+0x414	R/W	FMI DMA Interrupt Enable Register	0x0000_0000
<b>FMI_CTL</b>	FMI_BA+0x800	R/W	FMI Control and Status Register	0x0000_0000
<b>FMI_INTEN</b>	FMI_BA+0x804	R/W	FMI Interrupt Enable Register	0x0000_0001
<b>FMI_INTSTS</b>	FMI_BA+0x808	R/W	FMI Interrupt Status Register	0x0000_0000
<b>FMI_EMMCCTL</b>	FMI_BA+0x820	R/W	eMMC Control Register	0x0101_0000
<b>FMI_EMMCCMD</b>	FMI_BA+0x824	R/W	eMMC Command Argument Register	0x0000_0000
<b>FMI_EMMCINTEN</b>	FMI_BA+0x828	R/W	eMMC Interrupt Enable Register	0x0000_0000

<b>FMI_EMMCINTSTS</b>	FMI_BA+0x82C	R/W	eMMC Interrupt Status Register	0x00XX_008C
<b>FMI_EMMCRESP0</b>	FMI_BA+0x830	R	eMMC Receiving Response Token Register 0	0x0000_0000
<b>FMI_EMMCRESP1</b>	FMI_BA+0x834	R	eMMC Receiving Response Token Register 1	0x0000_0000
<b>FMI_EMMCBLEN</b>	FMI_BA+0x838	R/W	eMMC Block Length Register	0x0000_01FF
<b>FMI_EMMCTOUT</b>	FMI_BA+0x83C	R/W	eMMC Response/Data-in Time-out Register	0x0000_0000
<b>FMI_NANDCTL</b>	FMI_BA+0x8A0	R/W	NAND Flash Control Register	0x1E88_0090
<b>FMI_NANDTMCTL</b>	FMI_BA+0x8A4	R/W	NAND Flash Timing Control Register	0x0001_0105
<b>FMI_NANDINTEN</b>	FMI_BA+0x8A8	R/W	NAND Flash Interrupt Enable Register	0x0000_0000
<b>FMI_NANDINTSTS</b>	FMI_BA+0x8AC	R/W	NAND Flash Interrupt Status Register	0x00XX_0000
<b>FMI_NANDCMD</b>	FMI_BA+0x8B0	W	NAND Flash Command Port Register	N/A
<b>FMI_NANDADDR</b>	FMI_BA+0x8B4	W	NAND Flash Address Port Register	N/A
<b>FMI_NANDDATA</b>	FMI_BA+0x8B8	R/W	NAND Flash Data Port Register	N/A
<b>FMI_NANDRACTL</b>	FMI_BA+0x8BC	R/W	NAND Flash Redundant Area Control Register	0x0000_0000
<b>FMI_NANDECTL</b>	FMI_BA+0x8C0	R/W	NAND Flash Extend Control Register	0x0000_0000
<b>FMI_NANDECCES0</b>	FMI_BA+0x8D0	R	NAND Flash ECC Error Status 0 Register	0x0000_0000
<b>FMI_NANDECCES1</b>	FMI_BA+0x8D4	R	NAND Flash ECC Error Status 1 Register	0x0000_0000
<b>FMI_NANDECCES2</b>	FMI_BA+0x8D8	R	NAND Flash ECC Error Status 2 Register	0x0000_0000
<b>FMI_NANDECCES3</b>	FMI_BA+0x8DC	R	NAND Flash ECC Error Status 3 Register	0x0000_0000
<b>FMI_NANDPROTA0</b>	FMI_BA+0x8E0	R/W	NAND Flash Protect Region End Address 0 Register	0x0000_0000
<b>FMI_NANDPROTA1</b>	FMI_BA+0x8E4	R/W	NAND Flash Protect Region End Address 1 Register	0x0000_0000
<b>FMI_NANDECCEA0</b>	FMI_BA+0x900	R	NAND Flash ECC Error Byte Address 0 Register	0x0000_0000
<b>FMI_NANDECCEA1</b>	FMI_BA+0x904	R	NAND Flash ECC Error Byte Address 1 Register	0x0000_0000
<b>FMI_NANDECCEA2</b>	FMI_BA+0x908	R	NAND Flash ECC Error Byte Address 2 Register	0x0000_0000
<b>FMI_NANDECCEA3</b>	FMI_BA+0x90C	R	NAND Flash ECC Error Byte Address 3 Register	0x0000_0000
<b>FMI_NANDECCEA4</b>	FMI_BA+0x910	R	NAND Flash ECC Error Byte Address 4 Register	0x0000_0000
<b>FMI_NANDECCEA5</b>	FMI_BA+0x914	R	NAND Flash ECC Error Byte Address 5 Register	0x0000_0000
<b>FMI_NANDECCEA6</b>	FMI_BA+0x918	R	NAND Flash ECC Error Byte Address 6 Register	0x0000_0000
<b>FMI_NANDECCEA7</b>	FMI_BA+0x91C	R	NAND Flash ECC Error Byte Address 7 Register	0x0000_0000
<b>FMI_NANDECCEA8</b>	FMI_BA+0x920	R	NAND Flash ECC Error Byte Address 8 Register	0x0000_0000
<b>FMI_NANDECCEA9</b>	FMI_BA+0x924	R	NAND Flash ECC Error Byte Address 9 Register	0x0000_0000
<b>FMI_NANDECCEA10</b>	FMI_BA+0x928	R	NAND Flash ECC Error Byte Address 10 Register	0x0000_0000
<b>FMI_NANDECCEA11</b>	FMI_BA+0x92C	R	NAND Flash ECC Error Byte Address 11 Register	0x0000_0000
<b>FMI_NANDECCED0</b>	FMI_BA+0x960	R	NAND Flash ECC Error Data Register 0	0x8080_8080
<b>FMI_NANDECCED1</b>	FMI_BA+0x964	R	NAND Flash ECC Error Data Register 1	0x8080_8080
<b>FMI_NANDECCED2</b>	FMI_BA+0x968	R	NAND Flash ECC Error Data Register 2	0x8080_8080

<b>FMI_NANDECCED3</b>	FMI_BA+0x96C	R	NAND Flash ECC Error Data Register 3	0x8080_8080
<b>FMI_NANDECCED4</b>	FMI_BA+0x970	R	NAND Flash ECC Error Data Register 4	0x8080_8080
<b>FMI_NANDECCED5</b>	FMI_BA+0x974	R	NAND Flash ECC Error Data Register 5	0x8080_8080
<b>FMI_NANDRAAn n = 0, 1..117</b>	FMI_BA+0xA00+0x4*n	R/W	NAND Flash Redundant Area Word n n = 0, 1..117	Undefined

## 10.5 Functional Description

Flash Memory Interface (FMI) has DMA unit and FMI unit. The FMI unit has NAND controller and eMMC controller. The following sections will separate to describe each process steps.

### 10.5.1 DMA and FMI Global Control

DMA controller provides a direct memory access function. User only needs to fill the starting address and enable it, and DMAC can handle the data transmission automatically. DMA controller has a 128 bytes share buffer – separate to two 64 bytes ping pong FIFO. It can use the ping pong mechanism to provide multi-block transfer. When FMI is idle, the share buffer can be accessed directly by software.

FMI interface supports eMMC flash and NAND-type flash. FMI and DMAC provide fast data transfer between system memory and the card. Since DMAC only a single channel, which means that only one interface can be activated at the same time. eMMC and NAND are not co-exist.

To enable FMI and DMAC, please follow the steps below:

1. Set FMI\_DMACTL register DMACEN bit and SW\_RST bit.
2. Polling FMI\_DMACTL register SW\_RST bit until it was cleared.
3. Set FMI\_CTL register SW\_RST bit.
4. Polling FMI\_CTL register SW\_RST bit until it was cleared.

### 10.5.2 NAND Flash

FMI provides NAND-type flash memory access interface. This NAND-type flash memory controller provides all the necessary signals. User can easily generate the signals based on device specification. (Such as command port, address port and data port). It supports four different page size, 512 bytes, 2048 bytes, 4096 bytes and 8192 bytes. For different NAND, user needs to adjust the timing parameters (FMI\_NANDTMCTL register) to meet the NAND flash memory device specification. Periodic to adjust the timing parameters can also improve the performance of data transmission.

NAND-type flash memory controller provides a BCH error correction algorithm. This ECC

calculation circuit supports up to 4-bit, 8-bit, 12-bit, 15-bit or 24-bit error. User can check the error from reading FMI\_NANDINTSTS register ECC\_FLD\_IF bit, and also can get the error information from reading FMI\_NANDECCESn register. If needs doing correction, user should read the FMI\_NANDECCEAx and FMI\_NANDECCECx register to correct it.

About the device detail programming rule, please reference "Software Driver of SmartMedia", "SmartMedia Electrical Specifications", "SmartMedia Physical Format Specifications" and "SmartMedia Logical Format Specifications".

#### 10.5.2.1 NAND Initialize

To initial NAND controller, please follow the steps below:

1. Set CLK\_HCLKEN register FMI and NAND bit.
2. Select the multiple function pin. NAND has two set: GPC0~14 and GPI1~15.
  - (1) GPC: Set the value 0x55555 into SYS\_GPC\_MFPL register, and 0x05555555 into SYS\_GPC\_MFPH register.
  - (2) GPI: Set the value 0x55555550 into SYS\_GPI\_MFPL register, and 0x55555555 into SYS\_GPI\_MFPH register.
3. Set FMI\_CTL register NAND\_EN bit to enable NAND function.
4. Set FMI\_NANDECTL register WP bit to disable NAND-type flash memory write-protect.
5. Set FMI\_NANDCTL register CS0 or CS1 bit to select NAND chip select.

#### 10.5.2.2 Reset NAND-type Flash

Reset NAND-type flash memory, please follow the steps below:

1. Send “RESET” command 0xFF to FMI\_NANDCMD register.
2. Polling RB#. Check FMI\_NANDINTSTS register RB0\_IF bit until it was set. And then clear FMI\_NANDINTSTS register RB0\_IF bit.

#### 10.5.2.3 Identify NAND-type Flash

Identify NAND-type flash, please follow the steps below:

1. Send “Read ID” command 0x90 to FMI\_NANDCMD register.
2. FMI\_NANDADDR register ADDRESS bit fill address 0x00, and set EOA bit.
3. Get the ID from FMI\_NANDDATA register.
4. Get the NAND page size, ECC correct information from ID. And then set the FMI\_NANDCTL register PSIZE and BCH\_TSEL bit.
5. Set the redundant area depend on ID or specification. FMI\_NANDRACTL register RA128EN bit.

#### 10.5.2.4 Erase NAND-type Flash

Erase NAND-type flash, please follow the steps below:

1. Send “Block Erase” command 0x60 to FMI\_NANDCMD register.
2. Fill the row address from low to high into FMI\_NANDADDR register. Please reference the figure below.
3. Set FMI\_NANDADDR register EOA bit.
4. Send “Erase” command 0xD0 to FMI\_NANDCMD register.
5. Polling RB#. Check the FMI\_NANDINTSTS register RB0\_IF bit until it was set. And then clear FMI\_NANDINTSTS register RB0\_IF bit.
6. Send “Read Status” command 0x70 to FMI\_NANDCMD register.
7. Get the status from FMI\_NANDDATA register, and check the bit 0. 1: Fail; 0: Pass.

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

#### 10.5.2.5 Write NAND-type Flash

NAND-type flash page write access, please follow the steps below:

1. Fill target address to FMI\_DMASA register.
2. Fill 0x0000FFFF to FMI\_NANDRA0 register. It means this page was used.
3. Send “Serial Input” command 0x80 to FMI\_NANDCMD register.
4. Fill column address from low to high into FMI\_NANDADDR register. The column address usually fills 0, start from one page.
5. Fill row address from low to high into FMI\_NANDADDR register.
6. Set FMI\_NANDADDR register EOA bit.
7. Clear FMI\_NANDINTSTS register DMA\_IF, ECC\_FLD\_IF, PROT\_REGION\_WR\_IF bit.
8. Set FMI\_NANDCTL register REDUN\_AUTO\_WEN bit to enable auto-write redundant area.
9. Set FMI\_NANDCTL register DWR\_EN bit to enable DMA output data to NAND.

10. Polling DWR\_EN bit until it was cleared. Or polling FMI\_NANDINTSTS register DMA\_IF bit.
11. Send “Program” command 0x10 to FMI\_NANDCMD register.
12. Polling RB#. Check FMI\_NANDINTSTS register RB0\_IFbit until it was set. And then clear FMI\_NANDINTSTS register RB0\_IF bit.
13. Send “Read Status” command 0x70 to FMI\_NANDCMD register.
14. Get the status from FMI\_NANDDATA register, and check the bit 0. 1: Fail; 0: Pass.

#### 10.5.2.6 Read NAND-type Flash

Before NAND-type flash page read, user should read the redundant area first. NAND controller needs the redundant area ECC parity bytes for error correction. All page read access, please follow the steps below:

1. Get redundant area size from FMI\_NANDRACTL register RA128EN bit.
2. Send “Read” command 0x00 to FMI\_NANDCMD register.
3. Fill column address from low to high into FMI\_NANDADDR register.
4. Fill row address from low to high into FMI\_NANDADDR register.
5. Set FMI\_NANDADDR register EOA bit.
6. Send “Read Data” command 0x30 to FMI\_NANDCMD register.
7. Polling RB#. Check FMI\_NANDINTSTS register RB0\_IF bit until it was set. And then clear FMI\_NANDINTSTS register RB0\_IF bit.
8. There are two ways to write redundant area data into FMI\_NANDRA register:
  - (1) Hardware Read: Set FMI\_NANDCTL register REDUN\_REN bit. Polling REDUN\_REN bit until it was cleared.
  - (2) Software Read: According to the size of redundant area, read out one by one by FMI\_NANDDATA register.
9. Read data. Repeat step 2 ~ 7. The column address should be 0 for each page starting.
10. Fill target address to FMI\_DMASA register.
11. Clear FMI\_NANDINTSTS register DMA\_IF and ECC\_FLD\_IF bit.
12. Set FMI\_NANDCTL register DRD\_EN bit to enable DMA to get NAND data.
13. Polling DRD\_EN bit until it was cleared. Or polling FMI\_NANDINTSTS register DMA\_IF bit.
14. If FMI\_NANDINTSTS register ECC\_FLD\_IF bit was set, it means that data error. User should active error correction. (Refer to the error correction step for more detail).

#### 10.5.2.7 NAND-type Flash ECC Correction

BCH error correction algorithm can correct up to 4-bit, 8-bit, 12-bit, 15-bit or 24-bit errors. In addition to 24-bit computing unit is 1024 bytes, others are 512 bytes.

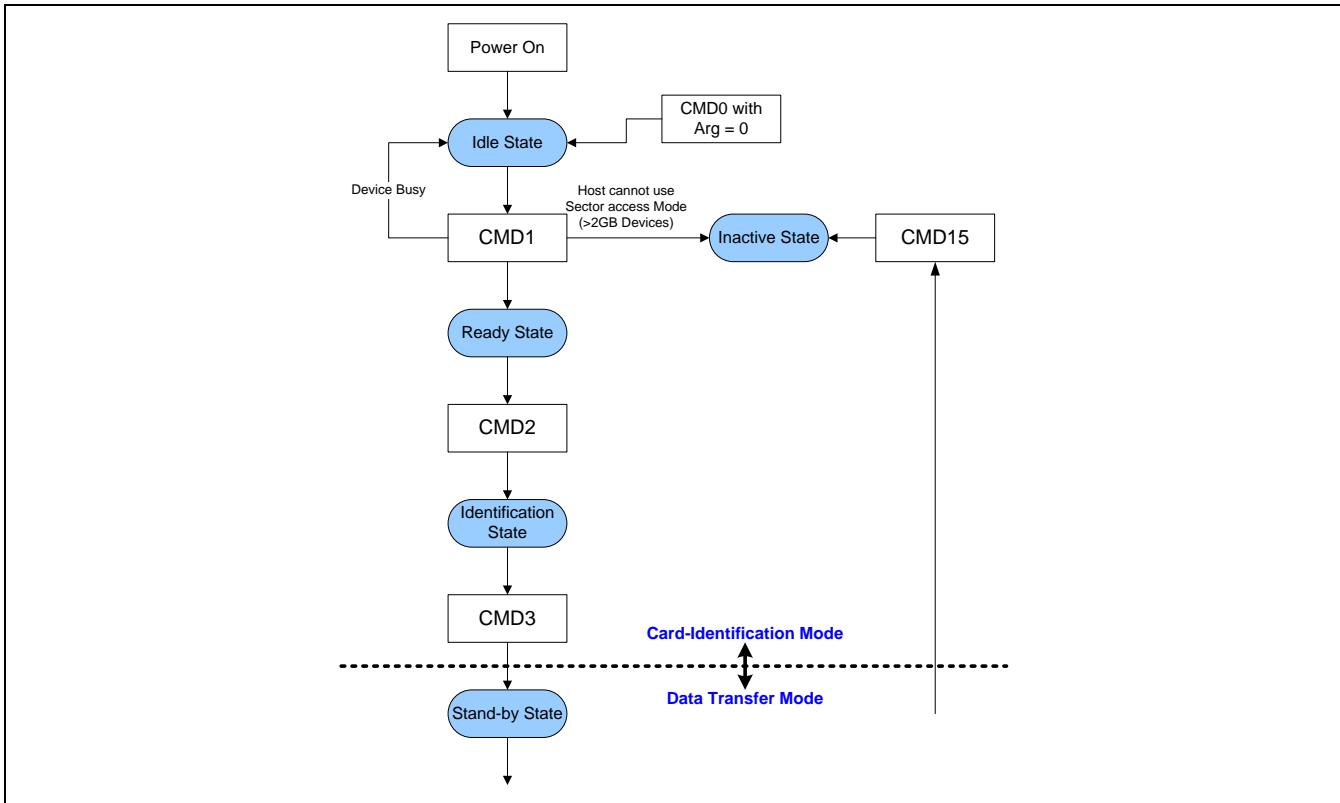
NAND-type flash memory error correction, please follow the steps below:

1. Read FMI\_NANDECCESn register Fx\_STAT bit to check whether the error can be corrected.
2. If errors can be corrected, read FMI\_NANDECCESn register Fx\_ECNT bit to get the number of errors.
3. According to the page size and BCH algorithm to calculate the correct region. Get the legal FMI\_NANDECCEDn and FMI\_NANDECCEAn register.
4. Reads FMI\_NANDECCEDn register to get incorrect data. Then get the wrong data address according to FMI\_NANDECCEAn register and obtain input data. These two data do XOR. The result is the correct data.

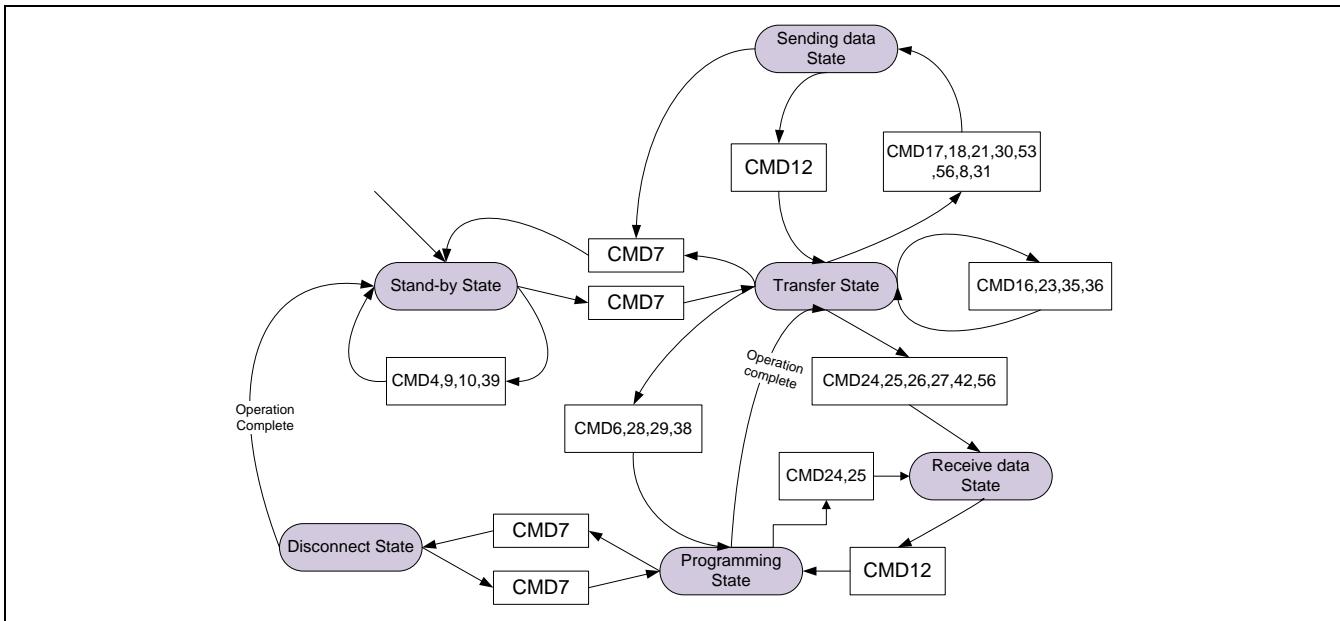
#### 10.5.3 eMMC

FMI provides an eMMC device interface. This eMMC controller supports 1-bit / 4-bit bus width. The controller can generates all types command and response. The response content will save at FMI\_EMMCRESP0 and FMI\_EMMCRESP1 register. About output frequency to eMMC device, user should control the CLKDIV3 register. Detailed procedural rules relating to the device, please refer to “JEDEC Standard No. 84-A441” and the manufactures eMMC datasheet.

eMMC Card Identification Mode:



eMMC Data Transfer Mode :



### 10.5.3.1 eMMC Initialize

eMMC initialize, please follow the steps below:

1. Set CLK\_HCLKEN register FMI, NAND and eMMC bit.
2. Select multiple function pin. eMMC has two set. GPC0~5和GPI5~10。
  - GPC: Set the value 0x00666666 to SYS\_GPC\_MFPL register.
  - GPI: Set the value 0x66600000 to SYS\_GPI\_MFPL register, and 0x666 to SYS\_GPI\_MFPH register.
3. Set FMI\_CTL register eMMC\_EN bit to enable eMMC.
4. Set FMI\_EMMCCTL register SW\_RST bit.
5. Polling FMI\_EMMCCTL register SW\_RST bit until it was cleared.
6. Set eMMC initial output frequency to 300 KHz, and 1-bit bus for eMMC interface.
7. Set FMI\_EMMCCTL register CLK74\_OE bit.
8. Polling FMI\_EMMCCTL register CLK74\_OE bit until it was cleared.
9. According to devices programming rule to send command to eMMC.
10. When device get into Data Transfer Mode, the output frequency can set to suitable clock. Such as 25MHz. And the bus width is 4-bit mode.

#### 10.5.3.2 Send Command

Send command to eMMC, please follow the steps below:

1. Set the argument to FMI\_EMMCCMD register.
2. Set command to FMI\_EMMCCTL register CMD\_CODE bit.
3. Set FMI\_EMMCCTL register CO\_EN bit to enable command out.
4. Polling FMI\_EMMCCTL register CO\_EN bit until it was cleared.

#### 10.5.3.3 Get Response

Get response from eMMC, please follow the steps below:

1. Set FMI\_EMMCCTL register RI\_EN bit to enable response in.
2. Polling FMI\_EMMCCTL register RI\_EN bit until it was cleared.
3. Check FMI\_EMMCINTSTS register CRC7 bit.
4. Get the response from FMI\_EMMCRESP0 and FMI\_EMMCRESP1 register.

#### 10.5.3.4 Read eMMC

eMMC read access, please follow the steps below:

1. Send CMD7 to enter transfer state.
2. Set FMI\_EMMCCTL register CLK8\_OE bit to output 8 clock cycles. Check FMI\_EMMCINTSTS register DAT0 bit. Repeat step 2 until eMMC is ready.

3. Set block size to FMI\_EMMCBLEN register. Such as 0x1FF is for 512 bytes.
4. Set the read starting sector address to FMI\_EMMCCMD register.
5. Set the data target address to FMI\_DMASA register.
6. Check the read sector count. If the count is greater than 255, user should separate it. Set the sector count to FMI\_EMMCCTL register BLK\_CNT bit. (255 is the limitation).
7. Send CMD18 for multiple read. (Set 18 to FMI\_EMMCCTL register CMD\_CODE bit).
8. Set FMI\_EMMCCTL register CO\_EN, RI\_EN and DI\_EN bit to enable command out, response in and data in.
9. Polling DI\_EN bit until it was cleared. Or waiting the interrupt (FMI\_EMMCINTSTS register BLKD\_IF bit).
10. Check FMI\_EMMCINTSTS register CRC7 and CRC16 bit.
11. Send CMD12 to stop transfer.
12. Set FMI\_EMMCCTL register CLK8\_OE bit to output 8 clock cycles. Check FMI\_EMMCINTSTS register DAT0 bit. Repeat step 12 until eMMC is ready.
13. Send CMD7 to idle state.

#### 10.5.3.5 Write eMMC

eMMC write access, please follow the steps below:

1. Send CMD7 to enter transfer state.
2. Set FMI\_EMMCCTL register CLK8\_OE bit to output 8 clock cycles. Check FMI\_EMMCINTSTS register DAT0 bit. Repeat step 2 until eMMC is ready.
3. Set block size to FMI\_EMMCBLEN register. Such as 0x1FF is for 512 bytes.
4. Set the write starting sector address to FMI\_EMMCCMD register.
5. Set the data source address to FMI\_DMASA register.
6. Check the write sector count. If the count is greater than 255, user should separate it. Set the sector count to FMI\_EMMCCTL register BLK\_CNT bit. (255 is the limitation).
7. Set CMD25 for multiple write. (Set 25 to FMI\_EMMCCTL register CMD\_CODE bit).
8. Set FMI\_EMMCCTL register CO\_EN, RI\_EN and DO\_EN bit to enable command out, response in and data out.
9. Polling DO\_EN bit until it was cleared. Or waiting the interrupt (FMI\_EMMCINTSTS register BLKD\_IF bit).
10. Check FMI\_EMMCINTSTS register CRC\_IF bit. If CRC error occurred, the state machine should software reset. (Set FMI\_EMMCCTL register SW\_RST bit).
11. Send CMD12 to stop transfer.
12. Set FMI\_EMMCCTL register CLK8\_OE bit to output 8 clock cycles. Check FMI\_EMMCINTSTS register DAT0 bit. Repeat step 12 until eMMC is ready.

- 
13. Send CMD7 to Idle state.

## 11 General DMA Controller (GDMA)

### 11.1 Overview

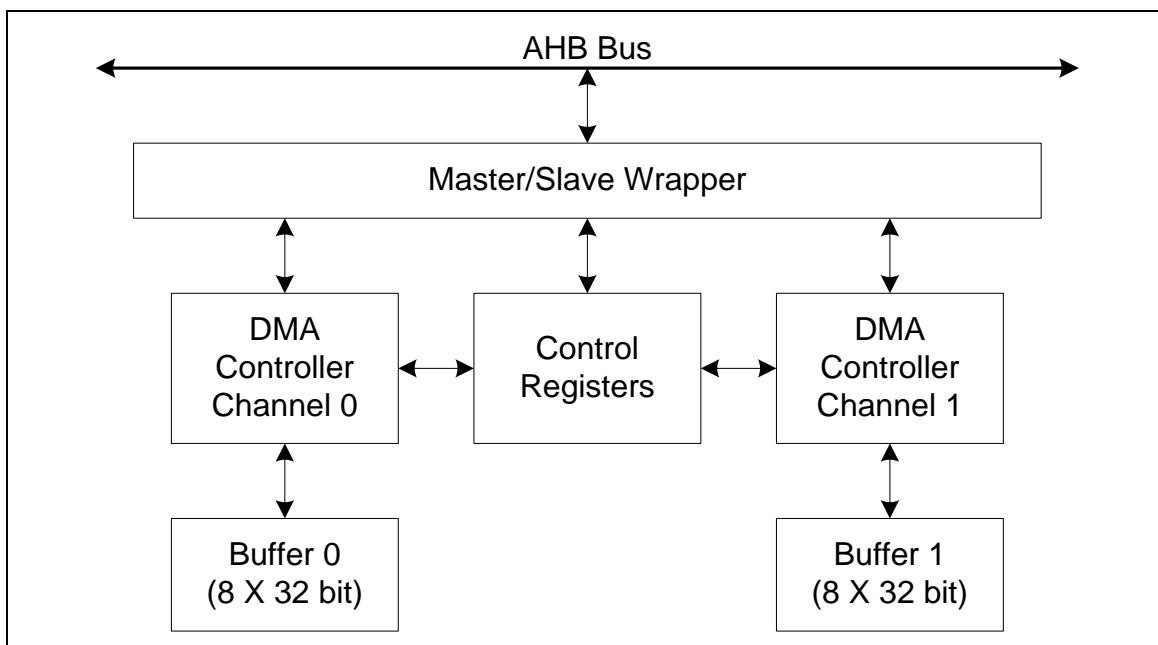
The chip has a two-channel general DMA controller with or without descriptor fetch operation, called the GDMA. The two-channel GDMA performs the memory-to-memory data transfers without the CPU intervention:

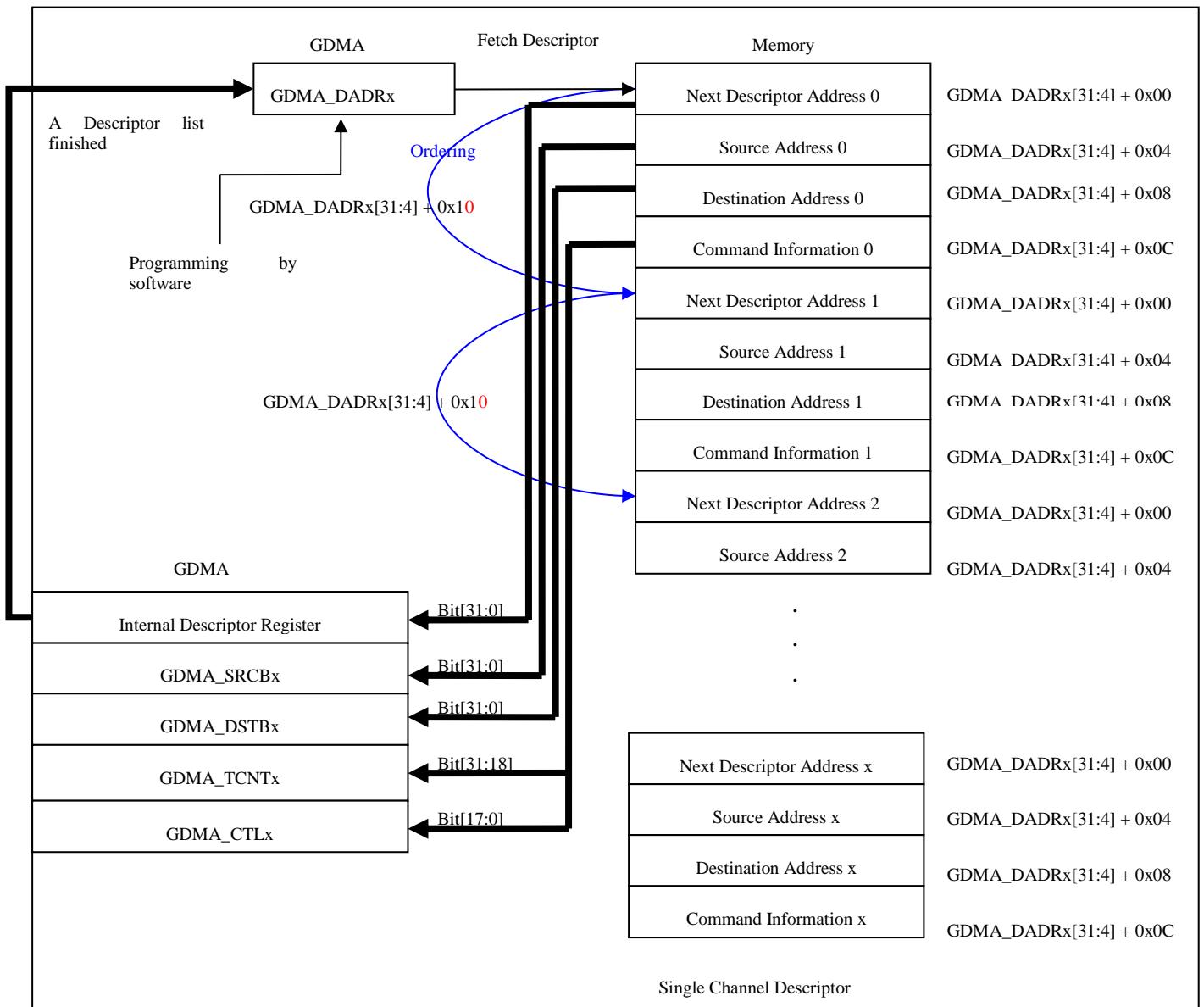
The on-chip GDMA can be started by the software. Software can also be used to restart the GDMA operation after it has been stopped. The CPU can recognize the completion of a GDMA operation by software polling or when it receives an internal GDMA interrupt. The GDMA controller can increment source or destination address, decrement them as well, and conduct 8-bit (byte), 16-bit (half-word), or 32-bit (word) data transfers.

### 11.2 Features

- AMBA AHB compliant
- Descriptor and Non-Descriptor based function
- Supports 8-data burst mode to boost performance
- Provides support for external GDMA device
- Demand mode speeds up external GDMA operations

### 11.3 Block Diagram





## 11.4 Register Map

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

Register	Address	R/W	Description	Reset Value
<b>GDMA_BA = 0xB000_4000</b>				
<b>Channel 0</b>				
<b>GDMA_CTL0</b>	GDMA_BA+0x000	R/W	Channel 0 Control Register	0x0000_0000
<b>GDMA_SRCBA0</b>	GDMA_BA+0x004	R/W	Channel 0 Source Base Address Register	0x0000_0000
<b>GDMA_DSTBA0</b>	GDMA_BA+0x008	R/W	Channel 0 Destination Base Address Register	0x0000_0000
<b>GDMA_TCNT0</b>	GDMA_BA+0x00C	R/W	Channel 0 Transfer Count Register	0x0000_0000

<b>GDMA_CSRCA0</b>	GDMA_BA+0x010	R	Channel 0 Current Source Address Register	0x0000_0000
<b>GDMA_CDSTA0</b>	GDMA_BA+0x014	R	Channel 0 Current Destination Address Register	0x0000_0000
<b>GDMA_CTCNT0</b>	GDMA_BA+0x018	R	Channel 0 Current Transfer Count Register	0x0000_0000
<b>GDMA_DADDR0</b>	GDMA_BA+0x01C	R/W	Channel 0 Descriptor Address Register	0x0000_0004
<b>Channel 1</b>				
<b>GDMA_CTL1</b>	GDMA_BA+0x020	R/W	Channel 1 Control Register	0x0000_0000
<b>GDMA_SRCBA1</b>	GDMA_BA+0x024	R/W	Channel 1 Source Base Address Register	0x0000_0000
<b>GDMA_DSTBA1</b>	GDMA_BA+0x028	R/W	Channel 1 Destination Base Address Register	0x0000_0000
<b>GDMA_TCNT1</b>	GDMA_BA+0x02C	R/W	Channel 1 Transfer Count Register	0x0000_0000
<b>GDMA_CSRCA1</b>	GDMA_BA+0x030	R	Channel 1 Current Source Address Register	0x0000_0000
<b>GDMA_CDSTA1</b>	GDMA_BA+0x034	R	Channel 1 Current Destination Address Register	0x0000_0000
<b>GDMA_CTCNT1</b>	GDMA_BA+0x038	R	Channel 1 Current Transfer Count Register	0x0000_0000
<b>GDMA_DADR1</b>	GDMA_BA+0x03C	R/W	Channel 1 Descriptor Address Register	0x0000_0004
<b>GDMA_BUFFER0</b>	GDMA_BA+0x080	R	GDMA Internal Buffer Word 0 Register	0x0000_0000
<b>GDMA_BUFFER1</b>	GDMA_BA+0x084	R	GDMA Internal Buffer Word 1 Register	0x0000_0000
<b>GDMA_BUFFER2</b>	GDMA_BA+0x088	R	GDMA Internal Buffer Word 2 Register	0x0000_0000
<b>GDMA_BUFFER3</b>	GDMA_BA+0x08C	R	GDMA Internal Buffer Word 3 Register	0x0000_0000
<b>GDMA_BUFFER4</b>	GDMA_BA+0x090	R	GDMA Internal Buffer Word 4 Register	0x0000_0000
<b>GDMA_BUFFER5</b>	GDMA_BA+0x094	R	GDMA Internal Buffer Word 5 Register	0x0000_0000
<b>GDMA_BUFFER6</b>	GDMA_BA+0x098	R	GDMA Internal Buffer Word 6 Register	0x0000_0000
<b>GDMA_BUFFER7</b>	GDMA_BA+0x09C	R	GDMA Internal Buffer Word 7 Register	0x0000_0000
<b>GDMA_INTS</b>	GDMA_BA+0x0A0	R/W	GDMA Interrupt Control and Status Register	0x0000_0000

## 11.5 Functional Description

### 11.5.1 Non-Descriptor Functional Descriptions

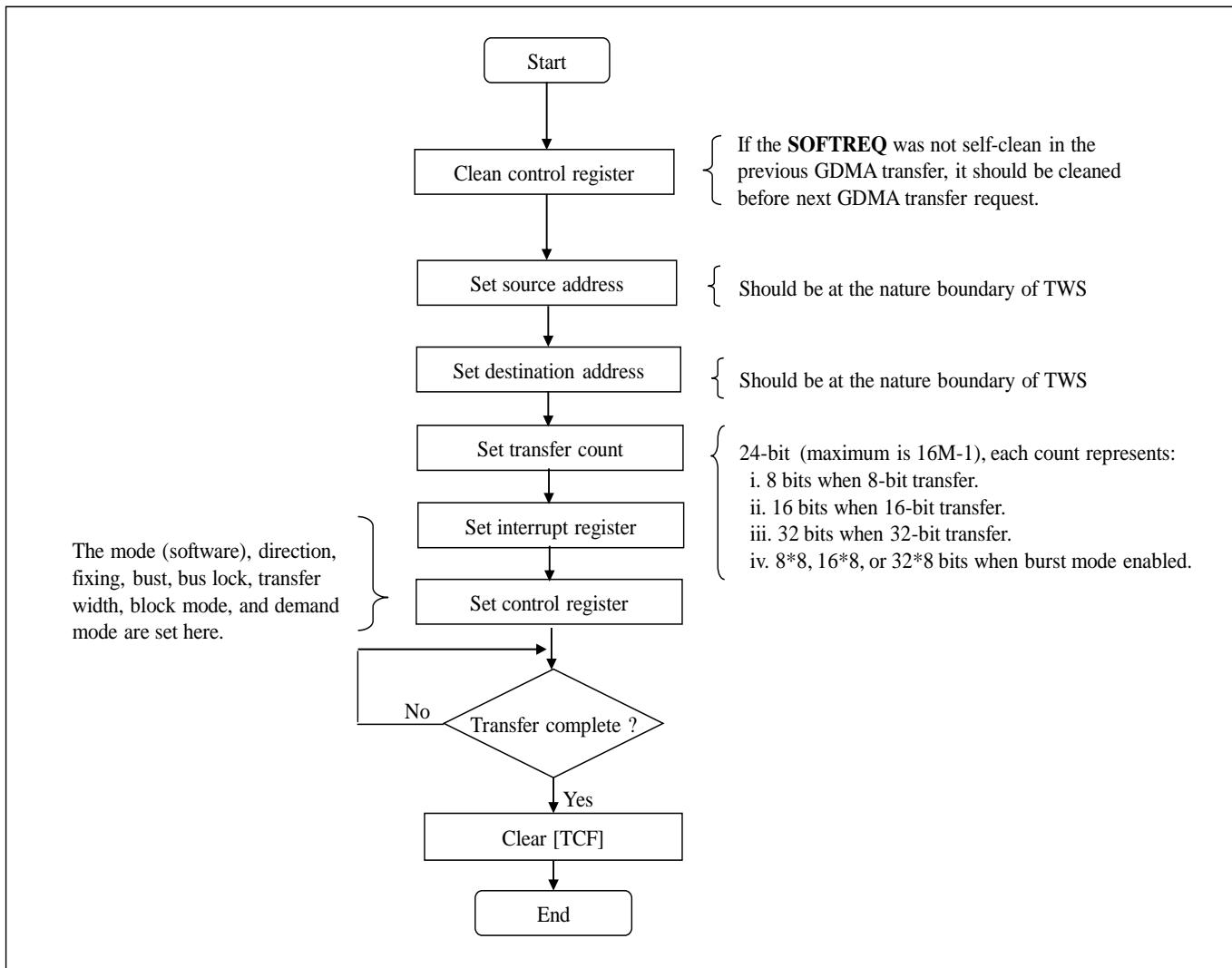
#### 11.5.1.1 GDMA Configuration

Each GDMA channel has one control register, one descriptor address register, two base address registers and one transfer count register. These registers should be correctly programmed before the data transfer starts. The most important one is the control register (GDMA\_CTL). It is used to control the transfer behavior of the GDMA operation, such as the transfer mode and transfer width. The following figure lists the content of GDMA\_CTL of non-descriptor fetches mode. The detail description of each bit-field can be found in data sheet.

31	30	29	28	27	26	25	24
----	----	----	----	----	----	----	----

RESERVED							
23	22	21	20	19	18	17	16
RESERVED	SABNDERR	DABNDERR	RESERVED	AUTOIEN	RESERVED	BLOCK	SOFTREQ
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	GDMAEN

The source base address register (GDMA\_SRCB) is used to set the base address of source data. The destination base address register (GDMA\_DSTB) is used to set the starting address where the source data to be stored. The number of the GDMA transfer is set by programming the transfer count register (GDMA\_TCNT). The following figure shows the programming flow for GDMA non-descriptor operation



### 11.5.1.2 Transfer Count

The value in register GDMA\_TCNT is the transfer count, not the byte count. Normally, the number of final transferred bytes is calculated by the following equation.

Transferred bytes = [GDMAx\_TCNT] \* Transfer width /\* burst mode is disabled \*/

For example, supposes that [GDMA\_TCNT] = 16 and the transfer width is half-word (16-bit). The number of transferred bytes should be  $16 * 2 = 32$ . But if the burst mode is enabled, the above equation will be changed as below.

Transferred bytes = [GDMAx\_TCNT] \* Transfer width \* 8 /\* burst mode is enabled \*/

In case of burst mode is enabled, the transferred bytes of the above example should be  $16 * 2 * 8 = 256$ .

### 11.5.1.3 Transfer Termination

When GDMA finishes the transfer, it will set the bit [TCF] of register GDMA\_INTCS and generate an interrupt request if the interrupt is enabled. The device driver can either poll the bit [TCF] or wait the GDMA interrupt occurs to know the transfer is completed. Note that the device driver must clear bit [TCF] to clear this interrupt request to let the next GDMA operation to continue

### 11.5.1.4 Fixed Address

Generally the GDMA continually increase or decrease the source and destination address during data transfer. The GDMA controller provides another feature to support the fixed source/destination address to perform data transfer between system memory and external device. To do a Memory-to-I/O transfer, the bit DAFIX(GDMA\_CTL[6]) should be set. In case of I/O-to-Memory transfer, the bit SAFIX(GDMA\_CTL[7]) should be set

### 11.5.1.5 Block Mode Transfer

When GDMA is programmed to block mode SBMS (GDMAx\_CTL[11]) = 1, it needs only one request to transfer all the data. When receiving the bit SOFTREQ is set, the GDMA begins to transfer data. After the numbers of data specified on register GDMAx\_TCNT have been transferred, the GDMA set the bit TCxF and generates an interrupt if it is enabled. Then the GDMA stops until next request is received.

### 11.5.1.6 GDMA operation started by software

The GDMA can be configured as software mode to perform memory-to-memory transfer. In this mode, the transfer operation starts as soon as the setting of the GDMA control registers are set, the setting of source address, destination address, and transfer count should be programmed in advanced. The programming method of software mode is listed below:

1. Set the GDMA to software mode GDMAMS (GDMAX\_CTL[3:2])=0x0.
2. Set source base address (GDMAX\_SRCB), destination base Address (GDMAX\_DSTB) and transfer Count (GDMAX\_TCNT).
3. Set SOFTREQ (GDMAX\_CTL[16])=1
4. Set SGDMAEN (GDMAX\_CTL[0])=1 to start.

In software mode, bit SOFTREQ and GDMAEN are self-cleared. The GDMA controller automatically clears these 2 bits after transfer completed. However, GDMAEN won't be self-clear if AUTOIEN bit is set. Hence, the driver only needs to set bit SOFTREQ to start next data transfer. If the GDMA didn't complete this transfer, it will cause the GDMA transfer error bit to be set, and the SOFTREQ won't be self-cleared. In this case, the SOFTREQ bit should be cleared before next software GDMA request.

It should be note that the source and destination base address must be in the right alignment according to its transfer width. For example, if the transfer width is 32-bit, the source and destination base address should be word-alignment. If each one is not aligned, the GDMA will read from and write to wrong addresses and the alignment error flags, SABNDERR and DABNDERR, will be set.

Shows an example code for software GDMA non-descriptor transfer.

```
#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;
}

```

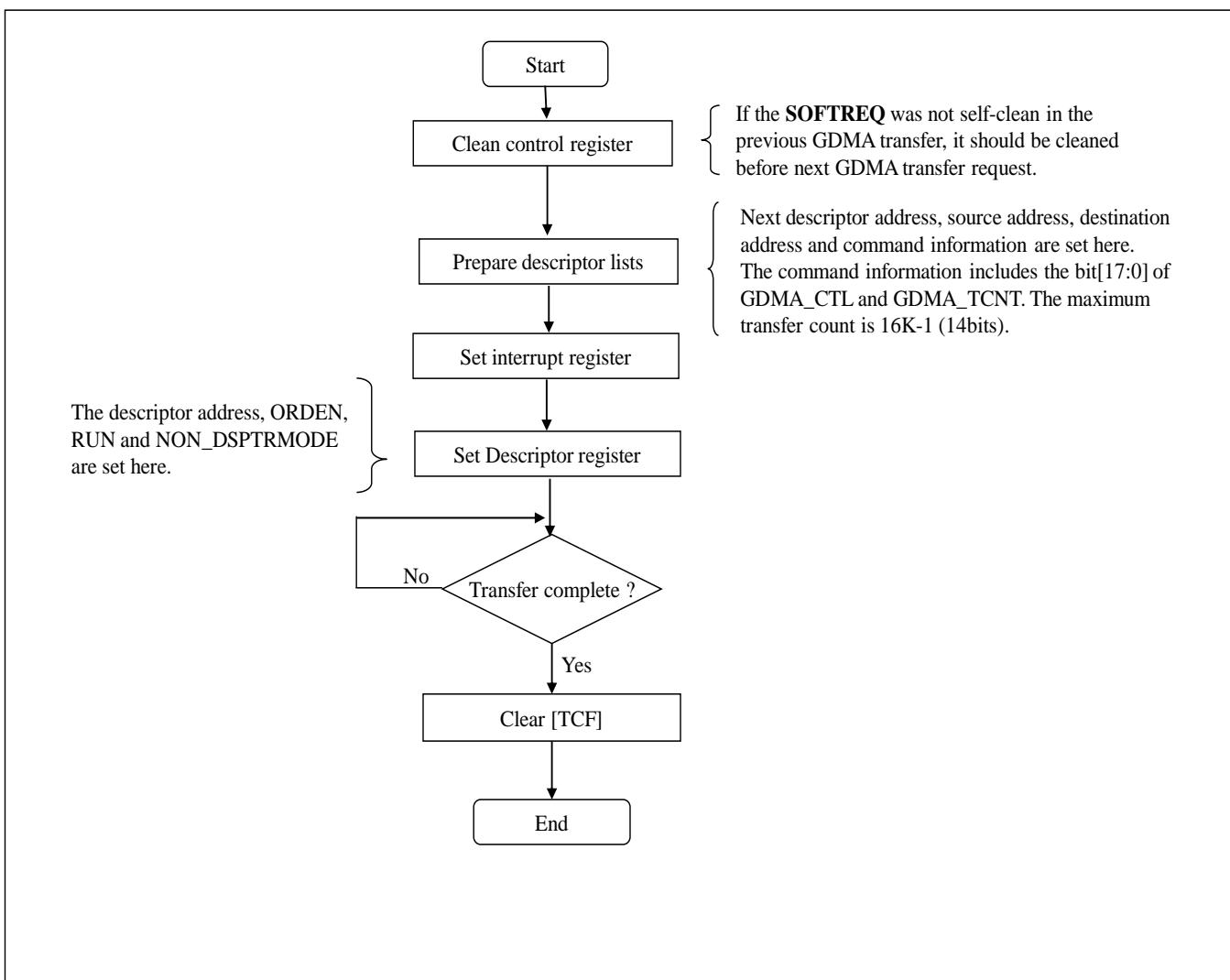
## 11.5.2 Descriptor Functional Descriptions

### 11.5.2.1 GDMA Configuration

Each GDMA channel has one control register, one descriptor address register, two base address registers and one transfer count register. Program should prepare the descriptor lists before the data transfer starts. And then control the descriptor address register to enable the descriptor based function. Figure 5-6 lists the content of GDMA\_CTL register of descriptor fetches mode and GDMA\_DADR register. The detail description of each bit-field can be found in data sheet. The following figure shows the programming flow for GDMA descriptor operation

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED	SABNDERR	DABNDERR	RESERVED			BLOCK	SOFTREQ
15	14	13	12	11	10	9	8
RESERVED		TWS		RESERVED	D_INTS	RESERVED	
7	6	5	4	3	2	1	0
SAFIX	DAFIX	SADIR	DADIR	GDMAMS		BME	GDMAEN

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_DSPTRMODE	ORDEN	RESET



### 11.5.2.2 GDMA operation started by software

The GDMA can be configured as software mode to perform memory-to-memory transfer. In this mode, the transfer operation starts as soon as the setting of the GDMA descriptor register is set, the descriptor lists should be programmed in advanced. The descriptor list of software mode is described below:

1. Prepare the next descriptor address for GDMA\_DADR register. If the ORDEN bit of GDMA\_DADR register is set, the next descriptor address of descriptor list is unnecessarily. Otherwise, the next descriptor address should be filled in descriptor list. The RUN bit should be set in descriptor lists except the last one. On the last descriptor list, the RUN and ORDEN bit should be cleared and the NON\_DSPRTMODE should be set.

2. Set source base address (GDMAx\_SRCB), destination base Address (GDMAx\_DSTB) and transfer Count (GDMAx\_TCNT).
3. Prepare the command information. The first fourteen bits ([31:18]) of the MSB of the command information will be written back to the GDMAx\_TCNT register and the others bits ([17:0]) will be written back to the GDMAx\_CTL register.

In software mode, bit SOFTREQ and GDMAEN are self-cleared. The GDMA controller automatically clears these 2 bits after transfer completed. However, GDMAEN won't be self-clear if AUTOIEN bit is set. Hence, the driver only needs to set bit SOFTREQ of the command information within each descriptor list. If the GDMA didn't complete this transfer, it will cause the GDMA transfer error bit to be set, and the SOFTREQ won't be self-cleared. In this case, the SOFTREQ bit should be cleared before next software GDMA request.

It should be note that the source and destination base address must be in the right alignment according to its transfer width. For example, if the transfer width is 32-bit, the source and destination base address should be word-alignment. If each one is not aligned, the GDMA will read from and write to wrong addresses and the alignment error flags, SABNDERR and DABNDERR, will be set.

Shows an example code for software GDMA Descriptor transfer

```
#define BASE 0xB0004000
#define GDMA_DADDR0      (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 Graphic Engine (GE2D)

### 12.1 Overview

A 32-bit 2D Graphics Engine (GE2D) is specially designed to improve the performance of graphic processing. It can accelerate the operation of individual GUI functions such as BitBLTs and Bresenham Line Draw to operate at all pixel depths including 8/16/32 bit-per-pixel.

A pixel is the smallest addressable screen element as defined in Microsoft Windows, and lines and pictures are made up by a variety of pixels. GE2D is used to speed up graphic performance in pixel data moving and line drawing, as well as to accelerate almost all computer graphic Boolean operations by eliminating the CPU overhead. Meanwhile, the functions of rotation and scaling down are implemented for some special applications. In image scaling down function, both programmable horizontal and vertical N/M scaling down factors are provided for resizing the image. For the 2D rotation, it can rotate left or right 45, 90 or 180 degrees, and also supports the flip/flop, mirror or up-side-down pictures.

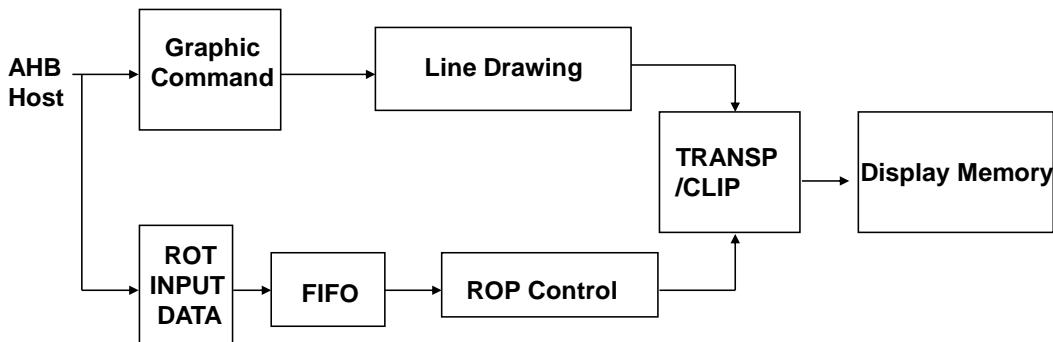
### 12.2 Features

- Support 2D Bit Block Transfer (BitBLT) functions defined in Microsoft GDI
- Support Host BLT
- Support Pattern BLT
- Support Color/Font Expanding BLT
- Support Transparent BLT
- Support Tile BLT
- Support Block Move BLT
- Support Copy File BLT
- Support Color/Font Expansion
- Support Rectangle Fill
- Support RGB332/RGB565/RGB888 data format.
- Support fore/background colors and all Microsoft 256 ternary raster-operation codes (ROP)
- Support both inside and outside clipping function
- Support alpha-blending for source/destination picture overlaying
- Support fast Bresenham line drawing algorithm to draw solid/textured line
- Support rectangular border and frame drawing
- Support picture re-sizing
- Support down-scaling from 1/255 to 254/255
- Support up-scaling from 1 to 1.996 (1+254/255)
- Support object rotation with different degree
- Support L45 (45 degree left rotation) and L90 (90 degree left rotation)

- Support R45 (45 degree right rotation) and R90 (90 degree right rotation)
- Support M180 (mirror/flop)
- Support F180 (up-side-down (flip) and X180 (180 degree rotation)

## 12.3 Block Diagram

Example: Host -> Graphics Engine -> Display Memory



## 12.4 Register Map

Register	Address	R/W	Description	Reset Value
<b>GE2D_BA = 0xB000_B000</b>				
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

<b>GE2D_TRNSCOLR</b>	GE2D_BA+0x02C	R/W	Graphic Engine Transparency Color Register	0x0000_0000
<b>GE2D_TCMSK</b>	GE2D_BA+0x030	R/W	Graphic Engine Transparency Color Mask Register	0x0000_0000
<b>GE2D_XYDORG</b>	GE2D_BA+0x034	R/W	Graphic Engine XY Mode Display Memory Origin Starting Register	0x0000_0000
<b>GE2D_SDPITCH</b>	GE2D_BA+0x038	R/W	Graphic Engine Source/Destination Pitch Register	0x0000_0000
<b>GE2D_SRCSPA</b>	GE2D_BA+0x03C	R/W	Graphic Engine Source Start Pixel/Address Register	0x0000_0000
<b>GE2D_DSTSPA</b>	GE2D_BA+0x040	R/W	Graphic Engine Destination Start Pixel/Address Register	0x0000_0000
<b>GE2D_RTGLSZ</b>	GE2D_BA+0x044	R/W	Graphic Engine Rectangle Size Register	0x0000_0000
<b>GE2D_CLPBTL</b>	GE2D_BA+0x048	R/W	Graphic Engine Clipping Boundary Top/Left Register	0x0000_0000
<b>GE2D_CLPBBR</b>	GE2D_BA+0x04C	R/W	Graphic Engine Clipping Boundary Bottom/Right Register	0x0000_0000
<b>GE2D_PTNA</b>	GE2D_BA+0x050	R/W	Graphic Engine Pattern Group A Register	0x0000_0000
<b>GE2D_PTNB</b>	GE2D_BA+0x054	R/W	Graphic Engine Pattern Group B Register	0x0000_0000
<b>GE2D_WRPLNMSK</b>	GE2D_BA+0x058	R/W	Graphic Engine Write Plane Mask Register	0x0000_0000
<b>GE2D_MISCTL</b>	GE2D_BA+0x05C	R/W	Graphic Engine Miscellaneous Control Register	0x0000_0000
<b>GE2D_GEHBDW0</b>	GE2D_BA+0x060	R/W	Graphic Engine HostBLT Data Port 0 Register	0x0000_0000
<b>GE2D_GEHBDW1</b>	GE2D_BA+0x064	R/W	Graphic Engine HostBLT Data Port 1 Register	0x0000_0000
<b>GE2D_GEHBDW2</b>	GE2D_BA+0x068	R/W	Graphic Engine HostBLT Data Port 2 Register	0x0000_0000
<b>GE2D_GEHBDW3</b>	GE2D_BA+0x06C	R/W	Graphic Engine HostBLT Data Port 3 Register	0x0000_0000
<b>GE2D_GEHBDW4</b>	GE2D_BA+0x070	R/W	Graphic Engine HostBLT Data Port 4 Register	0x0000_0000
<b>GE2D_GEHBDW5</b>	GE2D_BA+0x074	R/W	Graphic Engine HostBLT Data Port 5 Register	0x0000_0000
<b>GE2D_GEHBDW6</b>	GE2D_BA+0x078	R/W	Graphic Engine HostBLT Data Port 6 Register	0x0000_0000
<b>GE2D_GEHBDW7</b>	GE2D_BA+0x07C	R/W	Graphic Engine HostBLT Data Port 7 Register	0x0000_0000

## 12.5 Function Description

### 12.5.1 2D Graphic Engine Initialization

2-D GE provides the acceleration in computer graphics processing. The processing makes the images show on the screen display device correctly. Therefore, 2-D GE also concerns about the information of screen display device, such as the width of screen and height of screen. The information relates to the operations that 2-D GE uses it to set the state of certain registers. The information must be right to let 2-D GE get the correct datum and handle it in display memory normally.

The initialization of 2-D Graphic Engine contains the following steps:

1. Set global variables, including bit-per-pixel, width of screen and height of screen.
2. Allocate the space of display and pattern memory. The size of allocation of display memory is calculated according to bit-per-pixel, width of screen and height of screen and

memory address aligned to 64K bytes memory boundary. In the default, the XY mode source memory origin starting address of GE2D\_GEXYSORG register is set with display memory address and then source memory origin starting address should be 64K bytes boundary.

The pattern size is 8x8 pixels. Therefore, the allocated memory size of pattern that 8x8 multiplied by bit-per-pixel into eight. The pattern memory address must be programmed on an M-byte boundary.  $M=8x8xBPP/8$  bytes, where  $BPP=8/16/32$ .

3. Enable the 2-D GE clock, GE2D(CLK\_HCLKEN[28]).
4. Setting GE2DIF(GE2D\_INTSTS[0]) bit is zero in order to clear interrupt. And setting GE2IEN(GE2D\_CTL[17]) bit is zero in order to disable the interrupt.
5. Set the pattern location address into the register GE2D\_PATSA. In the default, setting the XY mode source and destination memory origin starting address take the display memory address. And set the bits Write Plane Mask of GE2D\_WRPLNMSK register to enable writing to the corresponding bit plane.
6. Set the BPP(GE2D\_MISCTL[5:4]) bits to let GE know how to get the byte count correctly.  
Note that BitBLT type is according to 2D\_GEC control bits setting.

A sample code initiates 2-D GE when essential datum input from caller function. The initial process is given below.

```
GFX_BPP = bpp;                                // bit per pixel
GFX_WIDTH = width;                            // width of screen
GFX_HEIGHT = height;                           // height of screen

GFX_PITCH = (GFX_WIDTH*(GFX_BPP/8));
GFX_SIZE = (GFX_HEIGHT*GFX_PITCH);
GFX_START_ADDR = (void *)malloc(GFX_SIZE+65536); // allocate memory for display
GFX_START_ADDR = (void *)shift_pointer((int)GFX_START_ADDR, 65536); // align 64K bytes
GFX_PAT_ADDR = (void *)malloc((8*8*(GFX_BPP/8))*2); // allocate memory for pattern
GFX_PAT_ADDR = (void *)shift_pointer((int)GFX_PAT_ADDR, (8*8*(GFX_BPP/8))*2); // align
appropriate bytes

CLK_HCLKEN |= (0x1 << 28); // enable 2D clock
GE2D_INTSTS |= 0x1;           // clear interrupt flag
G2DE_CTL = 0;                // disable interrupt

GE2D_PATSA = GFX_PAT_ADDR;    //pattern source address
GE2D_SRCSPA = GFX_START_ADDR; //display source address
GE2D_DSTSPA = GFX_START_ADDR; //display destination address
GE2D_WRPLNMSK = 0x00ffff;
```

```
GE2D_MISCTL = GE2D_MISCTL & ~(0x3 << 4) | bpp; // bpp
```

### 12.5.2 Ternary Raster Operations (ROP)

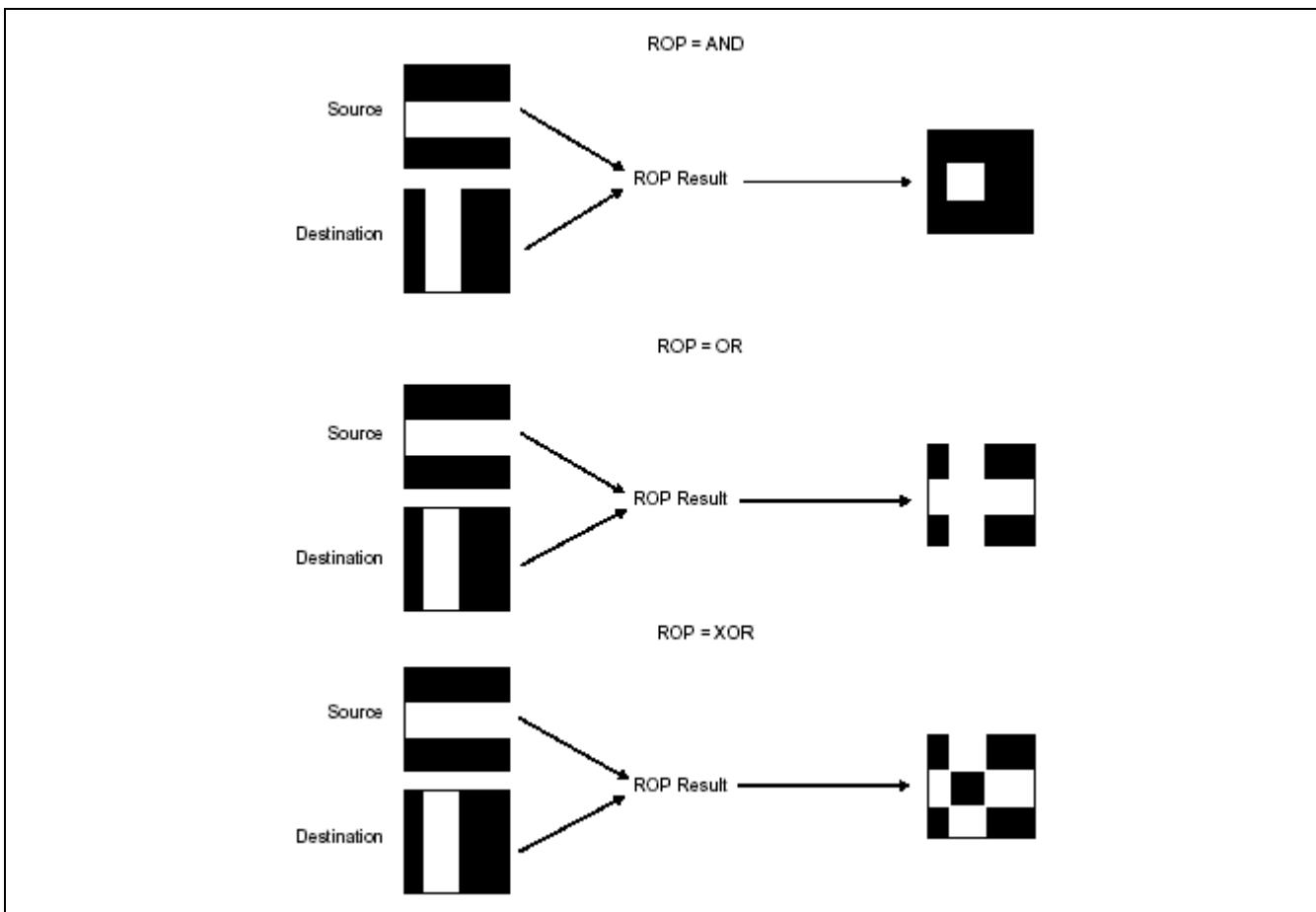
The 2-D GE supports all Microsoft 256 ternary raster-operation codes. The ternary raster-operation codes are used by the BitBLT functions. Ternary raster-operation codes define how BitBLT combines the bits in a source bitmap with the bits in a pattern map or the bits in the destination bitmap. Each raster-operation code represents a Boolean operation in which the values of the pixels in the source, the selected brush, and the destination are combined.

The most commonly used raster operations have been given special names. The table shows that 15 raster-operations codes have the common names.

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	SRCOPY
0xEE	SRCPAINT
0xF0	PATCOPY
0xFB	PATPAINT
0xFF	WHITENESS

Raster operation is always active during BitBLT and must be loaded with appropriate value. Programmer should set the ROP(GE2D\_CTL[31:24]) to fill with appropriate value.

The following figure are basic operations of ROP.

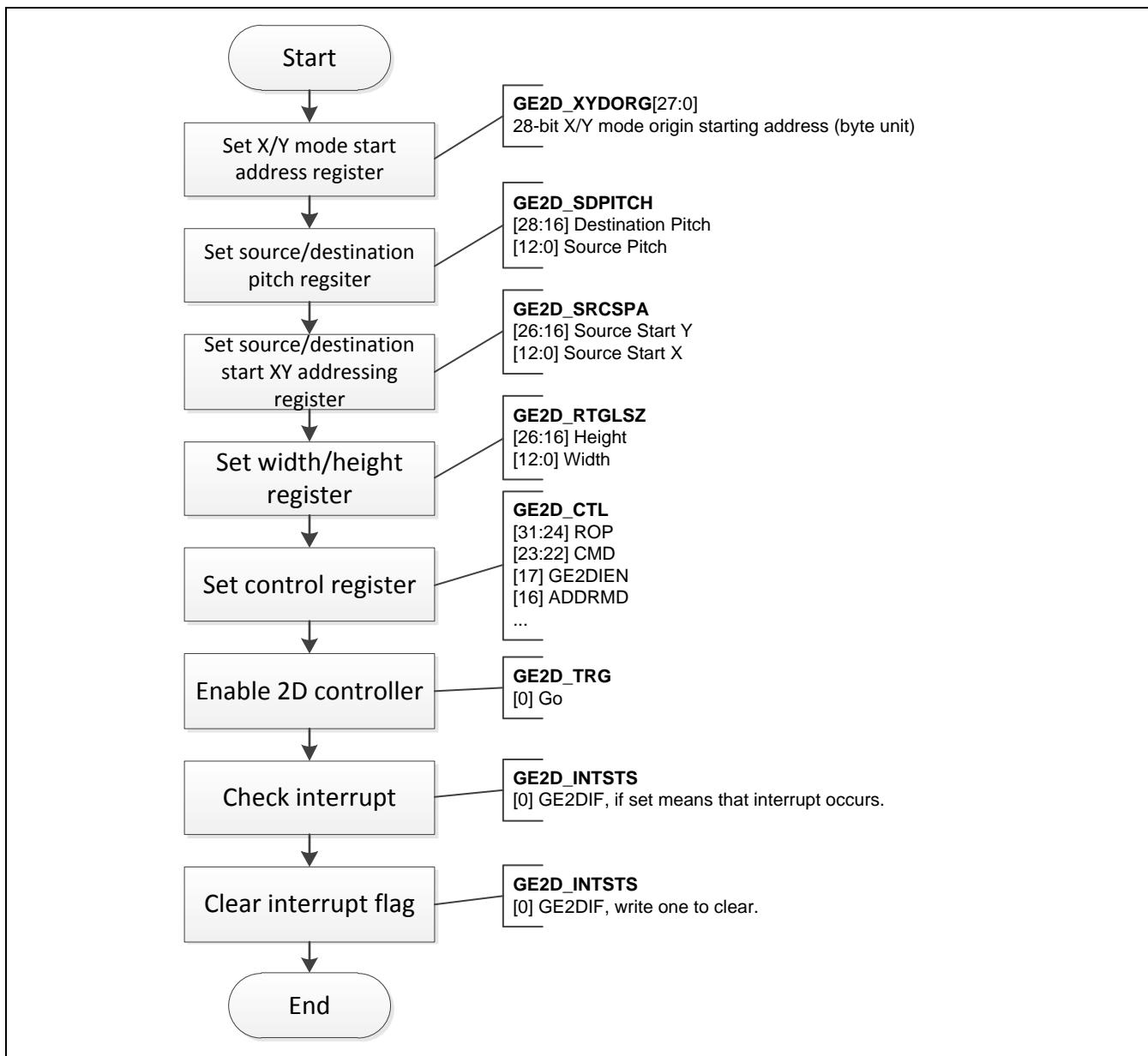


### 12.5.3 Bit Block Transfer (BitBLT)

The BitBLT function performs a bit-block transfer of the image data corresponding to a rectangle of pixels from the specified source context into the display memory. And BitBLT is the operation that accelerates the transfers of data between regions of display memory, or between system memory and display memory. The BitBLT engine operates on three pixel maps (operands): source, pattern, and destination, with all 256 possible raster operations (ROP). The graphic engine can support several kinds of BitBLT including HostBLT, Pattern BLT, Color/Font Expanding BLT, Transparent BLT, Color/Font Expansion, and Rectangle Fill, etc.

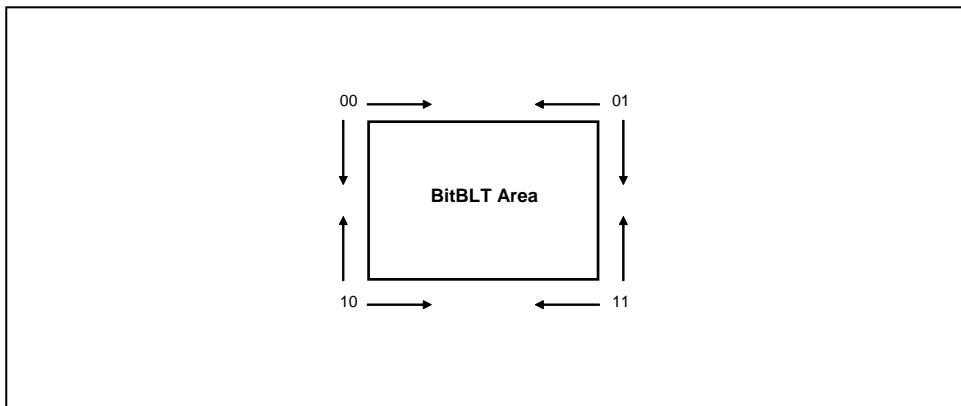
The source map data may reside in display memory or system memory, both can be color or monochrome. The 8x8 pattern map data may reside in display memory when it is color data, or may come from internal pattern register when it is monochrome data. The destination map data must reside in display memory. The resultant destination data generated by the engine is normally written back to the display memory, or it may be read back by the CPU.

The following figure is the flow of BitBLT.



The flowchart indicates that programmer could design the common BitBLT function with these essential registers. In the flowchart, it illustrates the behavior of actions and explains which registers are used. In the fact, before setting control register, the order of steps could be changed. Because of starting 2-D GE, it refers to the information according to GE control register (2D\_CTL).

The implementation of BitBLT function almost uses the X/Y addressing mode. In X/Y addressing mode, all source start X, destination start X, and dimension X are expressed in pixels. Source pitch and destination pitch are the same in X/Y addressing by pixels. In addition to BitBLT direction, the BitBLT direction indicates the direction in which the X/Y address is stepped across the rectangle. It also defines the starting corner of the transfer. This is significant if the destination rectangle overlaps the source rectangle. One must be certain that the operation progresses so that the source area is not overwritten prior to being used. BitBLT direction is controlled as shown in the following diagram.

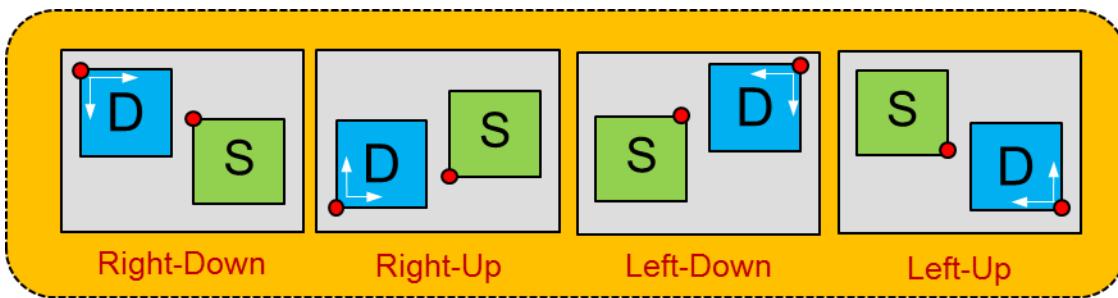


Drawing direction has relationship with starting address of destination, the following shows the easy method to decide the direction. (S is source address, D is destination address)

```

if((srcx > dstx) && (srcy > dsty)) direction = b'000; //right-down
if((srcx > dstx) && (srcy < dsty)) direction = b'100; //right-up
if((srcx < dstx) && (srcy > dsty)) direction = b'010; //left-down
if((srcx < dstx) && (srcy < dsty)) direction = b'110; //left-up
(srcx= source x starting address , srcy= source y starting address)
(dstx= destination x starting address , dsty= destination y starting address)

```



The example below explains that an image is drawn by BitBLT in display memory, which source and destination data is the same from the display memory. A BitBLT operation includes the following steps:

1. Set the bits X/Y source origin starting address of 2DGE\_XYSORG register to specify the starting address of display memory.  
Set the bits X/Y display origin starting address of 2DGE\_XYDORG register to specify the starting address of display memory

2. Set the basic control value, 0xCC4300, means that CC of raster-operation is source copy, set the CMD(GE2D\_CTL[23:22]) bits to start BitBLT acceleration up, set the GE2DIEN(GE2D\_CTL[17]) bit to enable interrupt and set the ADDRMD(GE2D\_CTL[16]) to make X/Y addressing mode. To estimate the BitBLT direction and set the DRAWDIR(GE2D\_CTL[3:1]) bits to determine the BitBLT directions.
3. Set the bits Destination Pitch and Source Pitch of GE2D\_SDPICTH register to specify the pitch in X/Y addressing mode by pixels.
4. Set the bits Source Start X and Source Start Y of GE2D\_SRCSPA register to specify the source start X/Y in pixels.  
Set the bits Destination Start X and Destination Start Y of GE2D\_DSTSPA register to specify the destination start X/Y in pixels.
5. Set the bits Dimension X and Dimension Y of GE2D\_RTGLSZ register to specify the width and height of rectangle in X/Y addressing by pixels.
6. Set the value of variable cmd32 into GE2D\_CTL register.
7. Set the GO(GE2D\_TRG[0]) bit to trigger 2-D GE acceleration.
8. Check the GE2DIF(GE2D\_INTSTS[0]) bit to confirm the drawing is done.
9. Write one to the GE2DIF(GE2D\_INTSTS) bit for clearing GE2DIF if drawing is done.

The example code is given blow.

```
GE2D_SRCSPA = GFX_START_ADDR; //display source address
GE2D_DSTSPA = GFX_START_ADDR; //display destination address

u32cmd = 0xCC430000;

// drawing direction
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; // set source/destination pitch
GE2D_SDPITCH = pitch;

src_start = srcy << 16 | srcx;           // set XY source starting address
GE2D_SRCSPA = src_start;

dest_start = desty << 16 | destx;         // set XT destination starting address
GE2D_DSTSPA = dest_start;

dimension = height << 16 | width;         // set width and height
GE2D_RTGLSZ = dimension;

GE2D_CTL = cmd32;
GE2D_TRG = 1;           // enable 2D
while ((GE2D_INTSTS & 0x01)==0); // wait for finish

GE2D_INTSTS = 1; // clear interrupt flag
```

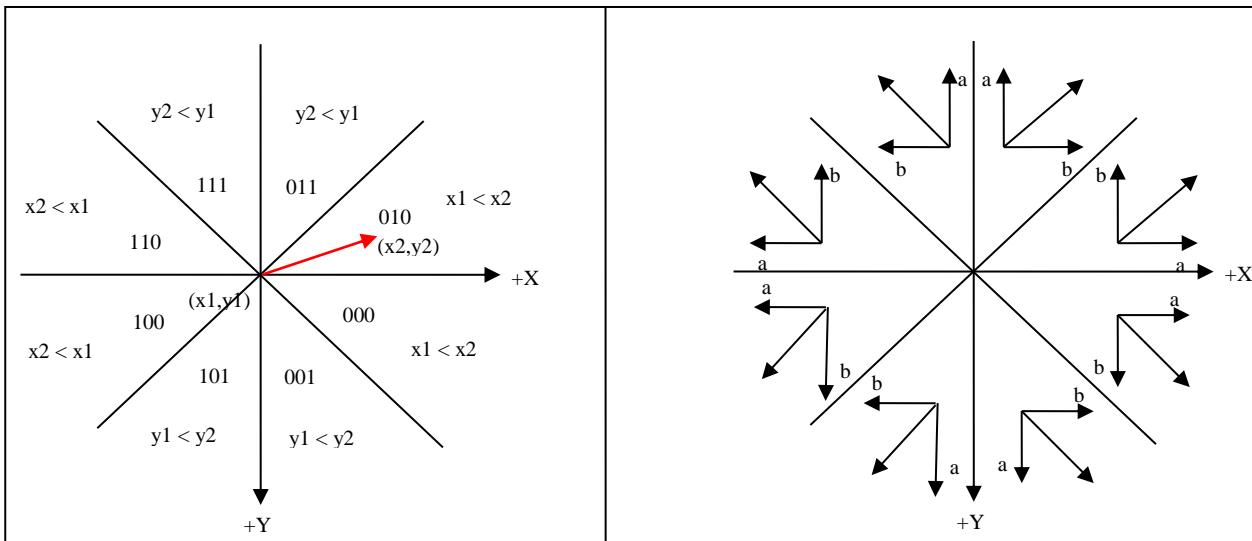
## 12.5.4 Bresenham Line Drawing

The Bresenham line drawing is an algorithm that determines which points should be plotted in order to form a close approximation to a straight line between two given points. This algorithm uses only integer addition, subtraction and bit shifting all of which are very cheap operations in computer architectures. Hence, there is efficient and fast reaction in Bresenham line drawing.

The Bresenham line drawing algorithm is used to draw a pixel wide solid or textured line from screen coordinates  $x_1, y_1$  to  $x_2, y_2$ . To draw a solid line, the foreground color is used to specify color of the line. To draw a textured line, a 16-bit line style pattern is used to specify the pattern of line, with all ones in the style being expanded to a pixel of foreground color and all zeros being either expanded to a pixel of background color or transparent. The 16-bit line style pattern (LNEPTN/ABLDFT) is in 2DGE\_MISCTL[31:16] register.

The Bresenham line drawing algorithm operates with all parameters normalized to the first

octant. A 3-bit octant code is specified as shown in the following left figure. The second data point has been assumed to be in the first octant with respect to first data point. If the second data point is located at another octant, an (a, b) coordinate system is again chosen with origin at the first data point, but with the axes oriented individually for each octant. As shown in the following right figure. Thus it could be seen, the Bresenham line drawing could handle the multiple slopes. All the vectors from  $x_1, y_1$  to  $x_2, y_2$  that there are eight regions and Bresenham line drawing algorithm could work in only one of them.

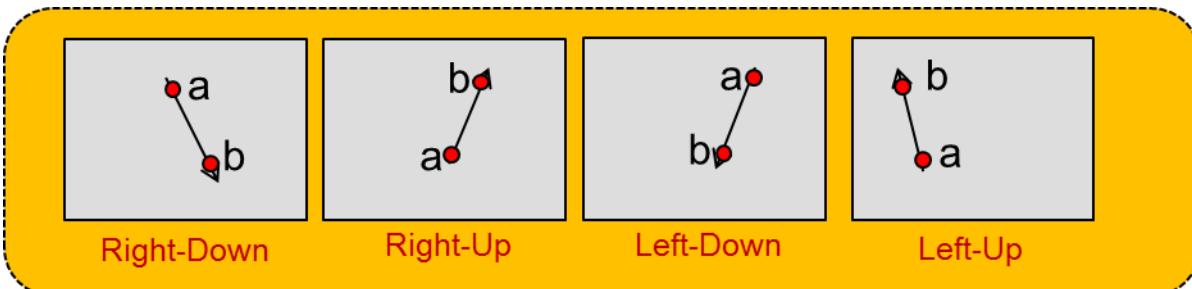


Two points decide a octant and direction at the same time. The following shows the easy method to decide a octant and direction

```

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;

```



In order to avoid drawing the endpoints of poly-lines twice, this chip provides a function that inhibits the drawing of the last pixel of the line, and this function is provided for the Bresenham Line Draw. The Bresenham Line Draw operation may be either a draw operation, or mere a move operation. On completion of the Bresenham Line Draw operation, destination

start X, Y normally points at the last pixel of the line by setting Auto Destination Update to be 1. It may also points at the original position of the line when Auto Destination Update is 0. Note that the Bresenham Line Draw operation is available only in X/Y addressing mode.

The example provides the function of drawing solid line blow.

1. Find out the octant code according to two given points. Set the basic control value, 0x008B0000 plus octant code, this value means that set the CMD(GE2D\_CTL[23:22]) bits to start Bresenham Line Draw acceleration, set the BLNMD(GE2D\_CTL[19]) bit to make Bresenham Line Draw, set the GE2DIEN(GE2D\_CTL[17]) bit to enable interrupt and set the bit ADDR\_MD to do X/Y addressing mode.
2. Set the bits Diagonal Error Increment and Axial Error Increment of GE2D\_BETSC register to specify the constant to be added to the error term for diagonal and axial stepping. The initial value of Diagonal Error Increment is  $(2 * (\text{delta Y} - \text{delta X}))$  after normalization to first octant. The initial value of Axial Error Increment is  $(2 * \text{delta Y})$  after normalization to first octant.  
Set the bits Initial Error Term and Line Pixel Count Major -1 of GE2D\_BIEPC register to specify the initial error term and the pixel count of major axis. The initial value of Initial Error Term is  $(2 * (\text{delta Y}) - \text{delta X})$  after normalization to first octant.
3. Set the Foreground Color of GE2D\_FGCOLR register to specify the foreground color.
4. Set the bits X/Y Origin Starting Address of GE2D\_XYDORG register to specify the starting address of display memory.
5. Set the bits Destination Pitch of GE2D\_SDPICTH register to specify the destination pitch in X/Y addressing by pixels.
6. Set the bits Destination Start X and Destination Start Y of GE2D\_DSTSPA register to specify the destination start X/Y in pixels.
7. Set the value of variable cmd32 into 2D\_CTL register.
8. Set the GO(GE2D\_TRG[0]) bit to trigger 2-D GE acceleration.
9. Check the GE2DIF(GE2D\_INTSTS[0]) bit to confirm the drawing is done.
10. Write one to the GE2DIF(GE2D\_INTSTS) bit for clearing GE2DIF if drawing is done.

The following is an example of drawing a solid line.

```
/* 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 XmYmYl (7<<1) */

// Calculate quadrant
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 = XpYpXl;
    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); // set background color
GE2D_SRCSPA = GFX_START_ADDR; // set source starting address

dest_pitch = GFX_WIDTH << 16; // set display pitch
2DGE_SDPITCH = dest_pitch;

dest_start = y1 << 16 | x1;
2DGE_DSTSPA = dest_start; // set destination starting address

GE2D_CTL = cmd32;
GE2D_TRG = 1; // enable 2D
while ((GE2D_INTSTS & 0x01)==0); // wait for finish
GE2D_INTSTS = 1; // clear interrupt flag
```

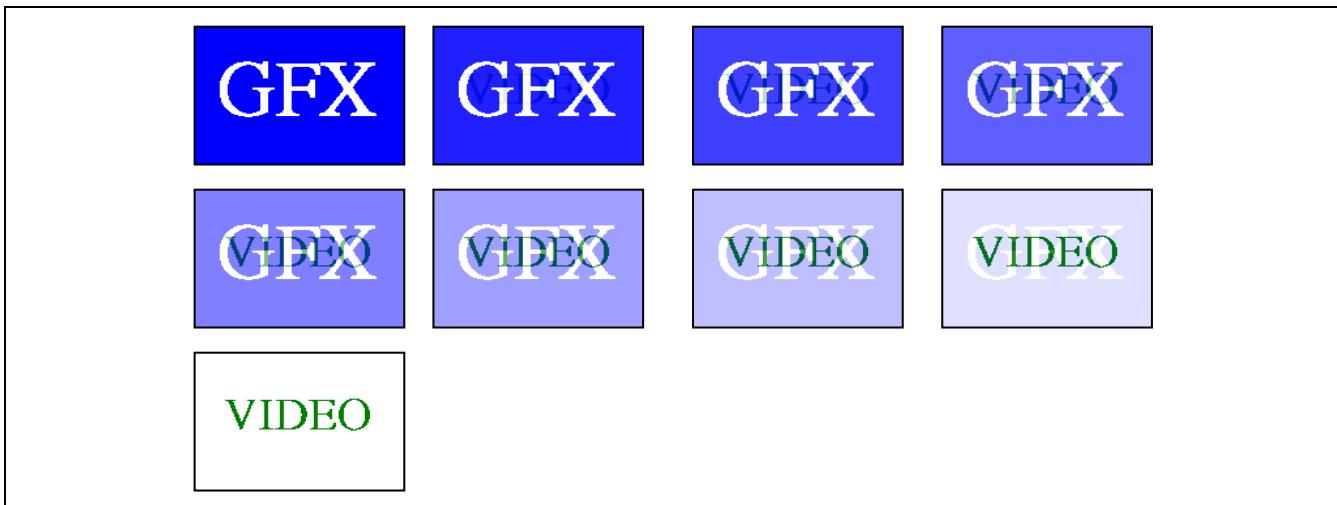
## 12.5.5 $\alpha$ Blending

In computer graphics, the combining of the alpha channel is in an image in order to show translucency. The alpha channel is an additional eight bits used with each pixel in a 32-bit graphics system that can represent 256 levels of translucency. Black and white represent opaque and fully transparent, while various gray levels represent levels of translucency.

In order to accomplish the function of alpha blending, programmer would properly program the two 8-bit alpha blending factors Ks and Kd in Miscellaneous Control register. Ks indicate the 8-bit alpha value of source stream, and Kd indicates the 8-bit alpha value of destination stream respectively. Note that Ks adding to Kd would be smaller than is equal to 256. Which the alpha blending factor is smaller, it expresses the image more translucency. Oppositely, alpha blending factor is bigger, expressing the image more opaque. The blending equation is:  $[Ps \times Ks + Pd \times Kd]/256$ , where Ps means the source stream pixels and Pd means the destination stream pixels. Note that the alpha blending function can't be used in pattern BLT.

The behavior of alpha blending is executed by the requirement. The action of alpha blending includes setting and decision. First, before doing the function of alpha blending, programmer would know the information about alpha blending factors. This information is stored to set the alpha mode.

The following figure is an example of alpha blending.



The action of alpha blending about the decision contains the following steps:

1. Alpha blending source Ks and destination Kd fill in the ABLDFCT(GE2D\_MISCTL[31:16]) bit.
2. Set APABLDEN(GE2D\_CTL[21]) bit to enable alpha blending function.

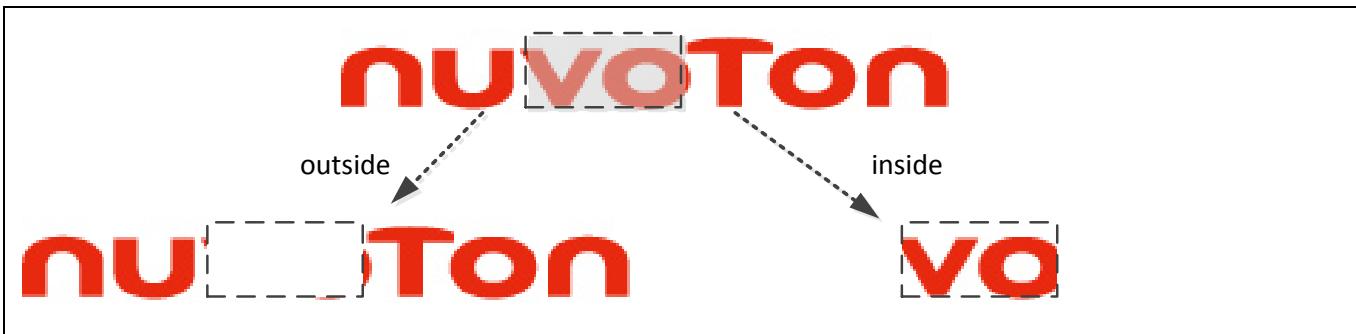
A sample code explains that this is inserted in display processing. The code is shown blow.

```
data32 = GE2D_MISCTL & 0x0000ffff;  
alpha = ((_AlphaKs << 8) | _AlphaKd); // fill with alpha blending factors  
data32 |= (alpha << 16);  
GE2D_MISCTL = data32;  
  
cmd32 |= 0x00200000; // enable alpha blending  
GE2D_CTL = cmd32;
```

## 12.5.6 Clipping

The clipping function supports clipped drawing writes inside or outside of any rectangular region in display memory during Graphics Engine operation. This chip supports both rectangle clipping for BitBLTs and line clipping for Bresenham Line. When enabled, the clipping function simply masks writes within or outside of the clipping window. Note that the clipping function is available only in X/Y addressing mode. The clipping area is assigned by up-left point (x1,y1) and right-down point (x2,y2).

The following figure shows clipping in inside and outside of the pattern.



The action of clipping function contains the following steps:

1. Set the CLPEN(GE2D\_CTL[9]) bit to enable the clipping. And set the CLPEN(GE2D\_CTL[9]) bit to clip outside of rectangle if the outside clip flag is set. Otherwise, the pixels inside the clipping rectangle are drawn
2. The top and left limit coordinates fill in the 11-bit clipping boundary top and left bits of GE2D\_CLPBTL register respectively. The bottom and right limit coordinates fill in the 11-bit clipping boundary bottom and right bits of GE2D\_CLPBBR register respectively.

A sample code explains that this is inserted in display processing. The code is shown blow.

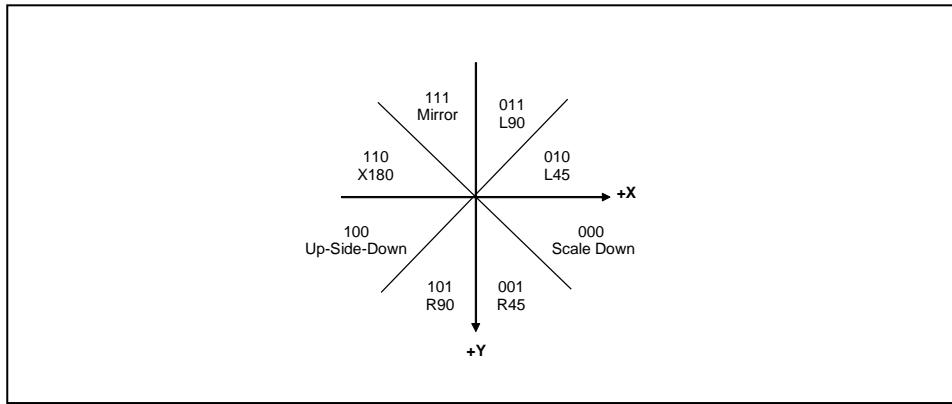
```
cmd32 |= 0x00000200; // enable clipping
if (_OutsideClip)
    cmd32 |= 0x00000100; // clip outside
    GE2D_CTL = cmd32;

    _ClipTL = ((y1 << 16) | x1);
    _ClipBR = ((y2 << 16) | x2);
    GE2D_CLPBTL = _ClipTL;
    GE2D_CLPBBR = _ClipBR;
```

### 12.5.7 Rotation

In the 2-D GE, one of main function is to support the rotation acceleration in any rectangular region in display memory during GE operation. For the 2-D GE rotation, it can rotate left or right 45, 90 or 180 degrees, and it also supports the flip/flop, mirror or up-side-down images.

The graphic engine rotation control operates with all parameters with reference to the first octant (octant 0). A 3-bit octant code is specified as shown in following figure.



About rotation function, here provides an example that 2-D GE acquires a rectangular region in display memory, and then 2-D GE also draws the rectangular region which been acquired in display memory with the drawing directions. Obviously, the rotation function that most main different point is involved the drawing directions in the DRAWDIR(GE2D\_CTL[3:1]) bits.

```

GE2D_SRCSPA = GFX_START_ADDR; // display source starting address
GE2D_DSTSPA = GFX_START_ADDR; // display destination starting address

pitch = GFX_WIDTH << 16 | GFX_WIDTH; // set display pitch
GE2D_SDPITCH = pitch;

src_start = srcy << 16 | srcx; // set X/Y source start address
GE2D_SRCSPA = src_start;

dest_start = desty << 16 | destx; // set X/Y destination start address
GE2D_DSTSPA = dest_start;

dimension = height << 16 | width; // set width and height
GE2D_RTGLSZ = dimension;

u32cmd = 0xCC030000 | (direction << 1); //set the drawing direction

GE2D_CTL = cmd32;
GE2D_TRG = 1; // enable 2D
while ((GE2D_INTSTS & 0x01)==0); // wait for finish
GE2D_INTSTS = 1; // clear interrupt flag

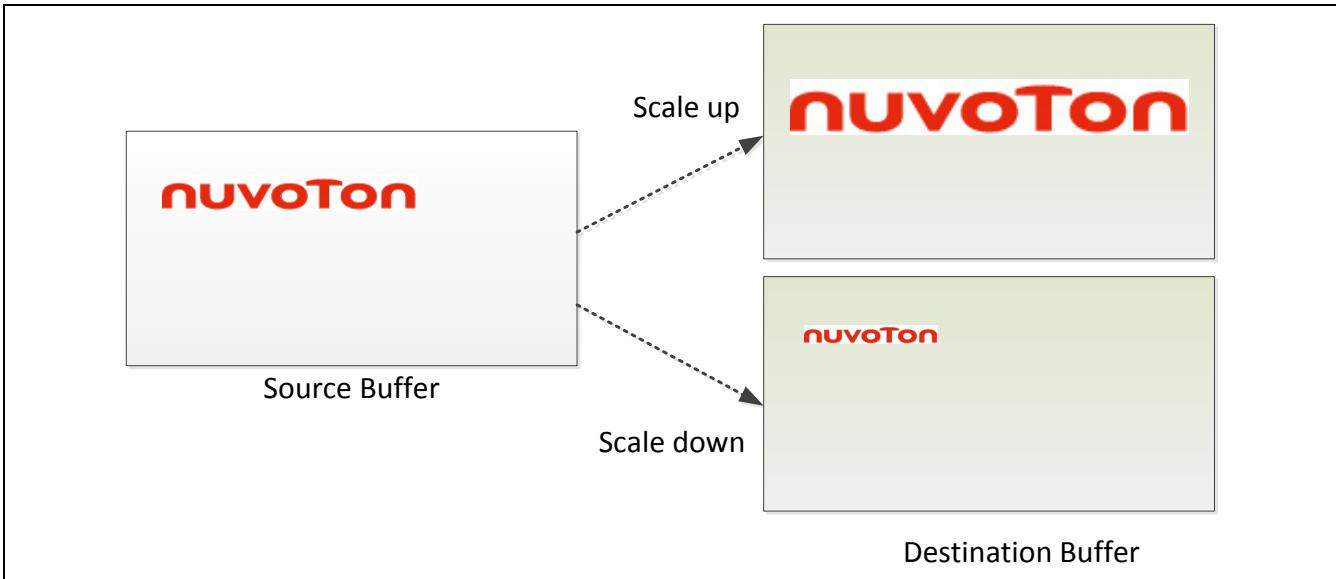
```

## 12.5.8 Scale Up/Down

One of main function is scaling capability in the 2-D GE. And this function is to support scaling up/down in any rectangular region in display memory. In image scale up/down function, both programmable horizontal and vertical N/M scaling up/down factors are provided for resizing the image. In order to scale up (1+N/M) or scale down (N/M), the value of N must be equal or

less than M. Scale up supports for 1.0 ~ 1.996.

The following figure is an effect of scale up/down.



The following is an example that an image could be scaled up/down in display memory. The source data is from display memory.

```
GE2D_SRCSPA = GFX_START_ADDR; // display source starting address  
GE2D_DSTSPA = GFX_START_ADDR; // display destination starting address  
  
pitch = GFX_WIDTH << 16 | GFX_WIDTH; // set display pitch  
GE2D_SDPITCH = pitch;  
  
src_start = srcy << 16 | srcx; // set X/Y source start address  
GE2D_SRCSPA = src_start;  
  
dest_start = desty << 16 | destx; // set X/Y destination start address  
GE2D_DSTSPA = dest_start;  
  
dimension = height << 16 | width; // set width and height  
GE2D_RTGLSZ = dimension;  
  
stretch_ctl = (vsfN << 24) | (vsfM << 16) | (hsfN << 8) | hsfM; // set vertical and horizontal scaling factor  
GE2D_TCNTVHSF = stretch_ctl;  
  
u32cmd = 0xCC030000;  
GE2D_CTL = cmd32;
```

```
GE2D_TRG = 1;           // enable 2D  
while ((GE2D_INTSTS & 0x01)==0);    // wait for finish  
GE2D_INTSTS = 1;        // clear interrupt flag
```

## 13 General-Purpose Input/Output (GPIO)

### 13.1 Overview

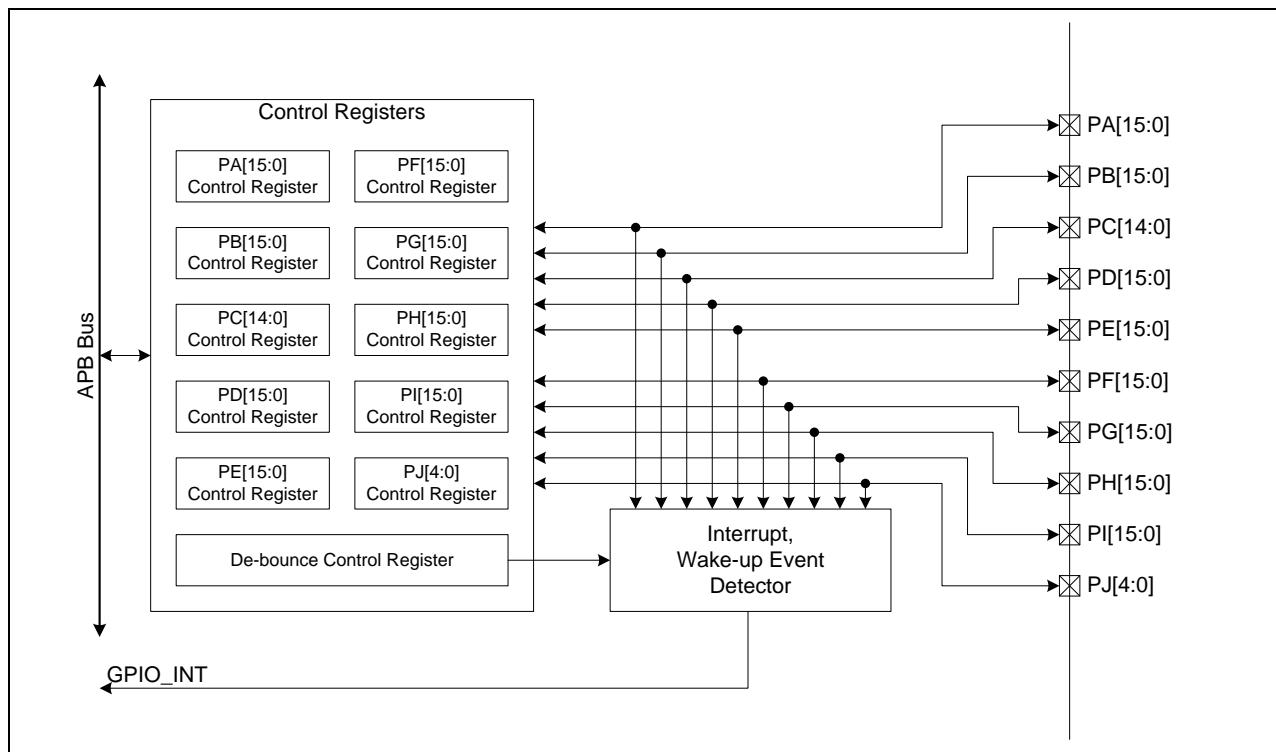
The NUC970/N9H30 series have up to 148 General-Purpose I/O (GPIO) pins and can be shared with other function pins depending on the chip configuration. These 148 pins are arranged in 10 ports named as PA, PB, PC, PD, PE, PF, PG, PH, PI and PJ. PA, PB, PD, PE, PF, PG, PH and PI have 16 pins on port, PC has 15 pins on port and PJ has 5 pins on port. Each of the 148 I/O pins is independent and can be easily configured by user to meet various system configurations and design requirements. After reset, all 148 I/O pins are configured in General-Purpose I/O Input mode.

When any of the 148 I/O pins used as a General-Purpose I/O, its I/O type can be configured by user individually as Input or Output mode. In Input mode, the input buffer type could be selected as CMOS input buffer or Schmitt trigger input buffer. Each I/O pin also equips a pull-up resistor ( $45\text{ k}\Omega \sim 82\text{ k}\Omega$ ) and a pull-down resistor ( $37\text{ k}\Omega \sim 91\text{ k}\Omega$ ). The enable of pull-up/pull-down resistor is controllable.

### 13.2 Features

- Support input and output mode.
- Support CMOS and Schmitt trigger input buffer.
- Support controllable pull-up and pull-down resistor.
- Support both edge and level interrupt.
- Support de-bounce circuit to filter the noise.

### 13.3 Block Diagram



## 13.4 Register Map

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

Register	Address	R/W	Description	Reset Value
<b>GPIO_BA = 0xB800_3000</b>				
<b>GPIOA_DIR</b>	GPIO_BA+0x000	R/W	GPIO Port A Direction Control Register	0x0000_0000
<b>GPIOA_DATAOUT</b>	GPIO_BA+0x004	R/W	GPIO Port A Data Output Register	0x0000_0000
<b>GPIOA_DATAIN</b>	GPIO_BA+0x008	R	GPIO Port A Data Input Register	0xxxxx_xxxx
<b>GPIOA_IMD</b>	GPIO_BA+0x00C	R/W	GPIO Port A Interrupt Mode Register	0x0000_0000
<b>GPIOA_IREN</b>	GPIO_BA+0x010	R/W	GPIO Port A Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOA_IFEN</b>	GPIO_BA+0x014	R/W	GPIO Port A Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOA_ISR</b>	GPIO_BA+0x018	R/W	GPIO Port A Interrupt Status Register	0xxxxx_xxxx
<b>GPIOA_DBEN</b>	GPIO_BA+0x01C	R/W	GPIO Port A De-bounce Enable Register	0x0000_0000
<b>GPIOA_PUEN</b>	GPIO_BA+0x020	R/W	GPIO Port A Pull-Up Enable Register	0x0000_0000
<b>GPIOA_PDEN</b>	GPIO_BA+0x024	R/W	GPIO Port A Pull-Down Enable Register	0x0000_0000
<b>GPIOA_ICEN</b>	GPIO_BA+0x028	R/W	GPIO Port A CMOS Input Enable Register	0x0000_0000
<b>GPIOA_ISEN</b>	GPIO_BA+0x02C	R/W	GPIO Port A Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOB_DIR</b>	GPIO_BA+0x040	R/W	GPIO Port B Direction Control Register	0x0000_0000
<b>GPIOB_DATAOUT</b>	GPIO_BA+0x044	R/W	GPIO Port B Data Output Register	0x0000_0000

<b>GPIOB_DATAIN</b>	GPIO_BA+0x048	R	GPIO Port B Data Input Register	0xxxxx_xxxx
<b>GPIOB_IMD</b>	GPIO_BA+0x04C	R/W	GPIO Port B Interrupt Mode Register	0x0000_0000
<b>GPIOB_IREN</b>	GPIO_BA+0x050	R/W	GPIO Port B Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOB_IFEN</b>	GPIO_BA+0x054	R/W	GPIO Port B Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOB_ISR</b>	GPIO_BA+0x058	R/W	GPIO Port B Interrupt Status Register	0xxxxx_xxxx
<b>GPIOB_DBEN</b>	GPIO_BA+0x05C	R/W	GPIO Port B De-bounce Enable Register	0x0000_0000
<b>GPIOB_PUEN</b>	GPIO_BA+0x060	R/W	GPIO Port B Pull-Up Enable Register	0x0000_0000
<b>GPIOB_PDEN</b>	GPIO_BA+0x064	R/W	GPIO Port B Pull-Down Enable Register	0x0000_0000
<b>GPIOB_ICEN</b>	GPIO_BA+0x068	R/W	GPIO Port B CMOS Input Enable Register	0x0000_0000
<b>GPIOB_ISEN</b>	GPIO_BA+0x06C	R/W	GPIO Port B Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOC_DIR</b>	GPIO_BA+0x080	R/W	GPIO Port C Direction Control Register	0x0000_0000
<b>GPIOC_DATAOUT</b>	GPIO_BA+0x084	R/W	GPIO Port C Data Output Register	0x0000_0000
<b>GPIOC_DATAIN</b>	GPIO_BA+0x088	R	GPIO Port C Data Input Register	0xxxxx_xxxx
<b>GPIOC_IMD</b>	GPIO_BA+0x08C	R/W	GPIO Port C Interrupt Mode Register	0x0000_0000
<b>GPIOC_IREN</b>	GPIO_BA+0x090	R/W	GPIO Port C Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOC_IFEN</b>	GPIO_BA+0x094	R/W	GPIO Port C Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOC_ISR</b>	GPIO_BA+0x098	R/W	GPIO Port C Interrupt Status Register	0xxxxx_xxxx
<b>GPIOC_DBEN</b>	GPIO_BA+0x09C	R/W	GPIO Port C De-bounce Enable Register	0x0000_0000
<b>GPIOC_PUEN</b>	GPIO_BA+0x0A0	R/W	GPIO Port C Pull-Up Enable Register	0x0000_0000
<b>GPIOC_PDEN</b>	GPIO_BA+0x0A4	R/W	GPIO Port C Pull-Down Enable Register	0x0000_0000
<b>GPIOC_ICEN</b>	GPIO_BA+0x0A8	R/W	GPIO Port C CMOS Input Enable Register	0x0000_0000
<b>GPIOC_ISEN</b>	GPIO_BA+0x0AC	R/W	GPIO Port C Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOD_DIR</b>	GPIO_BA+0x0C0	R/W	GPIO Port D Direction Control Register	0x0000_0000
<b>GPIOD_DATAOUT</b>	GPIO_BA+0x0C4	R/W	GPIO Port D Data Output Register	0x0000_0000
<b>GPIOD_DATAIN</b>	GPIO_BA+0x0C8	R	GPIO Port D Data Input Register	0xxxxx_xxxx
<b>GPIOD_IMD</b>	GPIO_BA+0x0CC	R/W	GPIO Port D Interrupt Mode Register	0x0000_0000
<b>GPIOD_IREN</b>	GPIO_BA+0x0D0	R/W	GPIO Port D Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOD_IFEN</b>	GPIO_BA+0x0D4	R/W	GPIO Port D Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOD_ISR</b>	GPIO_BA+0x0D8	R/W	GPIO Port D Interrupt Status Register	0xxxxx_xxxx
<b>GPIOD_DBEN</b>	GPIO_BA+0x0DC	R/W	GPIO Port D De-bounce Enable Register	0x0000_0000
<b>GPIOD_PUEN</b>	GPIO_BA+0x0E0	R/W	GPIO Port D Pull-Up Enable Register	0x0000_0000

<b>GPIOD_PDEN</b>	GPIO_BA+0x0E4	R/W	GPIO Port D Pull-Down Enable Register	0x0000_0000
<b>GPIOD_ICEN</b>	GPIO_BA+0x0E8	R/W	GPIO Port D CMOS Input Enable Register	0x0000_0000
<b>GPIOD_ISEN</b>	GPIO_BA+0x0EC	R/W	GPIO Port D Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOE_DIR</b>	GPIO_BA+0x100	R/W	GPIO Port E Direction Control Register	0x0000_0000
<b>GPIOE_DATAOUT</b>	GPIO_BA+0x104	R/W	GPIO Port E Data Output Register	0x0000_0000
<b>GPIOE_DATAIN</b>	GPIO_BA+0x108	R	GPIO Port E Data Input Register	0xxxxx_xxxx
<b>GPIOE_IMD</b>	GPIO_BA+0x10C	R/W	GPIO Port E Interrupt Mode Register	0x0000_0000
<b>GPIOE_IREN</b>	GPIO_BA+0x110	R/W	GPIO Port E Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOE_IFEN</b>	GPIO_BA+0x114	R/W	GPIO Port E Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOE_ISR</b>	GPIO_BA+0x118	R/W	GPIO Port E Interrupt Status Register	0xxxxx_xxxx
<b>GPIOE_DBEN</b>	GPIO_BA+0x11C	R/W	GPIO Port E De-bounce Enable Register	0x0000_0000
<b>GPIOE_PUEN</b>	GPIO_BA+0x120	R/W	GPIO Port E Pull-Up Enable Register	0x0000_0000
<b>GPIOE_PDEN</b>	GPIO_BA+0x124	R/W	GPIO Port E Pull-Down Enable Register	0x0000_0000
<b>GPIOE_ICEN</b>	GPIO_BA+0x128	R/W	GPIO Port E CMOS Input Enable Register	0x0000_0000
<b>GPIOE_ISEN</b>	GPIO_BA+0x12C	R/W	GPIO Port E Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOF_DIR</b>	GPIO_BA+0x140	R/W	GPIO Port F Direction Control Register	0x0000_0000
<b>GPIOF_DATAOUT</b>	GPIO_BA+0x144	R/W	GPIO Port F Data Output Register	0x0000_0000
<b>GPIOF_DATAIN</b>	GPIO_BA+0x148	R	GPIO Port F Data Input Register	0xxxxx_xxxx
<b>GPIOF_IMD</b>	GPIO_BA+0x14C	R/W	GPIO Port F Interrupt Mode Register	0x0000_0000
<b>GPIOF_IREN</b>	GPIO_BA+0x150	R/W	GPIO Port F Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOF_IFEN</b>	GPIO_BA+0x154	R/W	GPIO Port F Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOF_ISR</b>	GPIO_BA+0x158	R/W	GPIO Port F Interrupt Status Register	0xxxxx_xxxx
<b>GPIOF_DBEN</b>	GPIO_BA+0x15C	R/W	GPIO Port F De-bounce Enable Register	0x0000_0000
<b>GPIOF_PUEN</b>	GPIO_BA+0x160	R/W	GPIO Port F Pull-Up Enable Register	0x0000_0000
<b>GPIOF_PDEN</b>	GPIO_BA+0x164	R/W	GPIO Port F Pull-Down Enable Register	0x0000_0000
<b>GPIOF_ICEN</b>	GPIO_BA+0x168	R/W	GPIO Port F CMOS Input Enable Register	0x0000_0000
<b>GPIOF_ISEN</b>	GPIO_BA+0x16C	R/W	GPIO Port F Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOG_DIR</b>	GPIO_BA+0x180	R/W	GPIO Port G Direction Control Register	0x0000_0000
<b>GPIOG_DATAOUT</b>	GPIO_BA+0x184	R/W	GPIO Port G Data Output Register	0x0000_0000
<b>GPIOG_DATAIN</b>	GPIO_BA+0x188	R	GPIO Port G Data Input Register	0xxxxx_xxxx
<b>GPIOG_IMD</b>	GPIO_BA+0x18C	R/W	GPIO Port G Interrupt Mode Register	0x0000_0000

<b>GPIOG_IREN</b>	GPIO_BA+0x190	R/W	GPIO Port G Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOG_IFEN</b>	GPIO_BA+0x194	R/W	GPIO Port G Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOG_ISR</b>	GPIO_BA+0x198	R/W	GPIO Port G Interrupt Status Register	0xxxxx_xxxx
<b>GPIOG_DBEN</b>	GPIO_BA+0x19C	R/W	GPIO Port G De-bounce Enable Register	0x0000_0000
<b>GPIOG_PUEN</b>	GPIO_BA+0x1A0	R/W	GPIO Port G Pull-Up Enable Register	0x0000_0000
<b>GPIOG_PDEN</b>	GPIO_BA+0x1A4	R/W	GPIO Port G Pull-Down Enable Register	0x0000_0000
<b>GPIOG_ICEN</b>	GPIO_BA+0x1A8	R/W	GPIO Port G CMOS Input Enable Register	0x0000_0000
<b>GPIOG_ISEN</b>	GPIO_BA+0x1AC	R/W	GPIO Port G Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOH_DIR</b>	GPIO_BA+0x1C0	R/W	GPIO Port H Direction Control Register	0x0000_0000
<b>GPIOH_DATAOUT</b>	GPIO_BA+0x1C4	R/W	GPIO Port H Data Output Register	0x0000_0000
<b>GPIOH_DATAIN</b>	GPIO_BA+0x1C8	R	GPIO Port H Data Input Register	0xxxxx_xxxx
<b>GPIOH_IMD</b>	GPIO_BA+0x1CC	R/W	GPIO Port H Interrupt Mode Register	0x0000_0000
<b>GPIOH_IREN</b>	GPIO_BA+0x1D0	R/W	GPIO Port H Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOH_IFEN</b>	GPIO_BA+0x1D4	R/W	GPIO Port H Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOH_ISR</b>	GPIO_BA+0x1D8	R/W	GPIO Port H Interrupt Status Register	0xxxxx_xxxx
<b>GPIOH_DBEN</b>	GPIO_BA+0x1DC	R/W	GPIO Port H De-bounce Enable Register	0x0000_0000
<b>GPIOH_PUEN</b>	GPIO_BA+0x1E0	R/W	GPIO Port H Pull-Up Enable Register	0x0000_0000
<b>GPIOH_PDEN</b>	GPIO_BA+0x1E4	R/W	GPIO Port H Pull-Down Enable Register	0x0000_0000
<b>GPIOH_ICEN</b>	GPIO_BA+0x1E8	R/W	GPIO Port H CMOS Input Enable Register	0x0000_0000
<b>GPIOH_ISEN</b>	GPIO_BA+0x1EC	R/W	GPIO Port H Schmitt-Trigger Input Enable Register	0x0000_0000
<b>GPIOI_DIR</b>	GPIO_BA+0x200	R/W	GPIO Port I Direction Control Register	0x0000_0000
<b>GPIOI_DATAOUT</b>	GPIO_BA+0x204	R/W	GPIO Port I Data Output Register	0x0000_0000
<b>GPIOI_DATAIN</b>	GPIO_BA+0x208	R	GPIO Port I Data Input Register	0xxxxx_xxxx
<b>GPIOI_IMD</b>	GPIO_BA+0x20C	R/W	GPIO Port I Interrupt Mode Register	0x0000_0000
<b>GPIOI_IREN</b>	GPIO_BA+0x210	R/W	GPIO Port I Interrupt Rising-Edge or Level-High Enable Register	0x0000_0000
<b>GPIOI_IFEN</b>	GPIO_BA+0x214	R/W	GPIO Port I Interrupt Falling-Edge or Level-Low Enable Register	0x0000_0000
<b>GPIOI_ISR</b>	GPIO_BA+0x218	R/W	GPIO Port I Interrupt Status Register	0xxxxx_xxxx
<b>GPIOI_DBEN</b>	GPIO_BA+0x21C	R/W	GPIO Port I De-bounce Enable Register	0x0000_0000
<b>GPIOI_PUEN</b>	GPIO_BA+0x220	R/W	GPIO Port I Pull-Up Enable Register	0x0000_0000
<b>GPIOI_PDEN</b>	GPIO_BA+0x224	R/W	GPIO Port I Pull-Down Enable Register	0x0000_0000
<b>GPIOI_ICEN</b>	GPIO_BA+0x228	R/W	GPIO Port I CMOS Input Enable Register	0x0000_0000

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

## 13.5 Functional Description

### 13.5.1 Multiple function pin Configuration

To configure pin Px.n as a General-Purpose I/O, set the corresponding field of register 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 and SYS\_GPJ\_MFPL to 0.

For example, if user want to configure pin PA.0 as a General-Purpose I/O, it's necessary to set MFP\_GPA0 (SYS\_GPA\_MFPL[4:7]) to 0.

```
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 Output Mode

Before the system use the GPIO pin as output pin, program need to configure the GPIO direction register (GPIOx\_DIR). The configuration sequence is described as follows

1. Set Multiple function pin to GPIO purpose according to the above method of multiple function configuration.
2. Set the GPIOx\_DIR Px.n value as 1(output mode).

A sample code set GPIOC[0] as GPIO output mode, then change the output between high and low is given below:

```
// 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 Input Mode

Before the system use the GPIO pin as output pin, program need to configure the GPIO direction register (GPIOx\_DIR). The configuration sequence is described as follows

1. Set multiple function pin to GPIO purpose according to the above method of multiple function configurations.
2. Set the GPIOx\_DIR Px.n value as 0(input mode).

A sample code set GPIOC[0] as GPIO input mode, then get the input value is given below:

```
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 Interrupt

The GPIO pin all supported interrupt, program need to configure the GPIO direction register(GPIOx\_DIR) and the GPIO interrupt register(GPIOx\_IMD, GPIOx\_IREN, GPIO\_IFEN).

1. Set multiple function pin to GPIO purpose according to the above method of multiple function configurations.
2. Set the GPIOx\_DIR Px.n value as 0(input mode).
3. Set the GPIOx\_IMD value as 0(Edge trigger interrupt) or 1(Level trigger interrupt).
4. Set the GPIO\_IREN value as 0(Rising disable) or 1(Rising enable).
5. Set the GPIO\_IFEN value as 0(Falling disable) or 1(Falling enable).

In addition to these steps, It is also necessary to clear the corresponding interrupt source register GPIOx\_ISR after interrupt occurs.

A sample code set GPIOC[0] as GPIO input mode, then get interrupt flag when change input value. As below:

```
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));

// Waiting 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 I<sup>2</sup>C

### 14.1 Overview

I<sup>2</sup>C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. The I<sup>2</sup>C standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously.

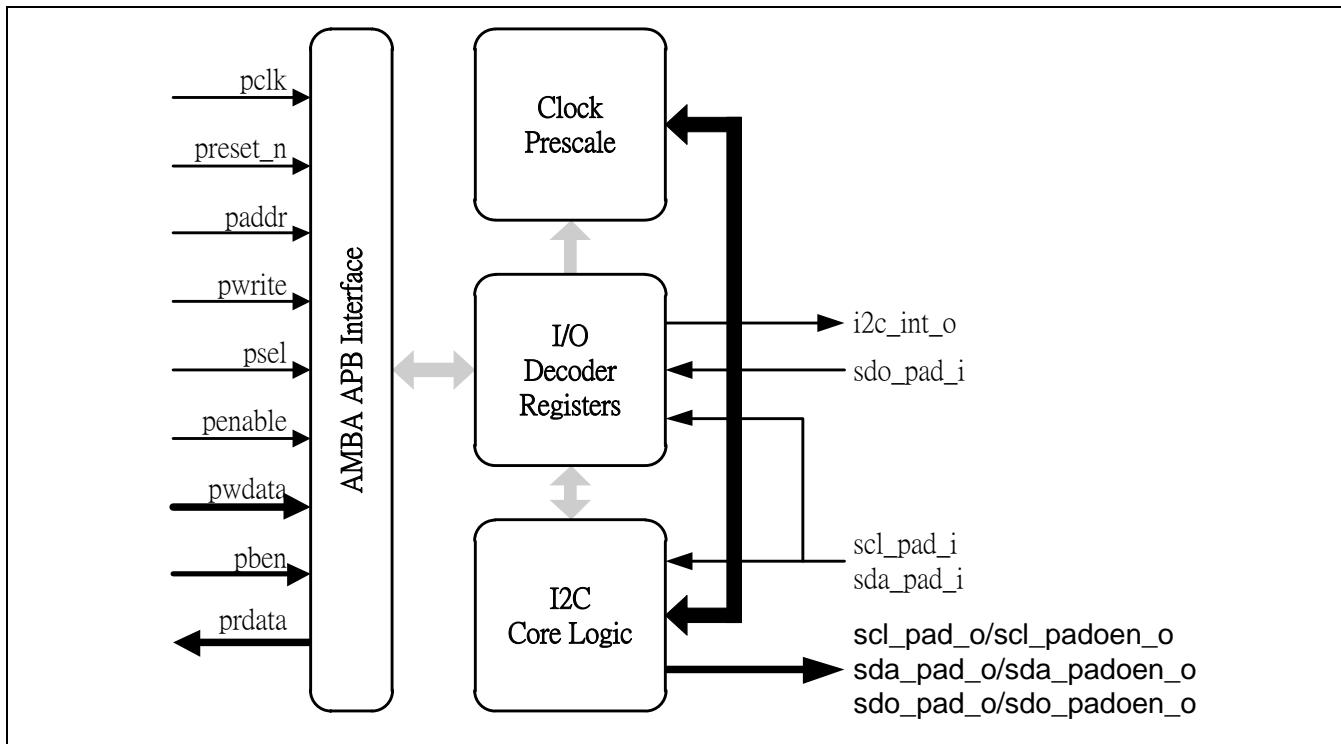
Serial, 8-bit oriented bi-directional data transfers can be up to 100 KBit/s in Standard-mode, 400 KBit/s in the Fast-mode, or 3.4 Mbit/s in the High-speed mode. Only 100kbps and 400kbps modes are supported directly in this chip.

Data transfer is synchronized to SCL signal between a Master and a Slave with byte-by-byte basis. Each data byte is 8 bits long. There is one SCL clock pulse for each data bit with the MSB being transmitted first. An acknowledge bit follows each transferred byte. Each bit is sampled during the high period of SCL; therefore, the SDA line may be changed only during the low period of SCL and must be held stable during the high period of SCL. A transition on the SDA line while SCL is high is interpreted as a command (START or STOP).

### 14.2 Features

- Compatible with Philips I<sup>2</sup>C standard, support master mode
- Supports 7 bit addressing mode
- Multi Master Operation
- Arbitration lost interrupt, with automatic transfer cancellation.
- Provide multi-byte transmit operation, up to 4 bytes can be transmitted in a single transfer
- Software programmable acknowledge bit
- Bus busy detection
- Clock stretching and wait state generation

### 14.3 Function Block



## 14.4 Register Map

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

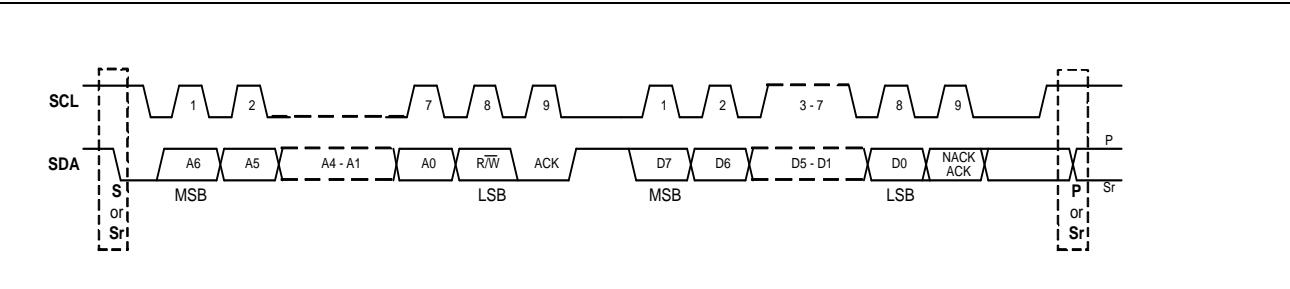
## 14.5 Function Description

### 14.5.1 I<sup>2</sup>C Protocol

The following figure shows the typical I<sup>2</sup>C protocol. Normally, a standard communication consists of four parts:

1. START or Repeated START signal generation
2. Slave address transfer

3. Data transfer
4. STOP signal generation



#### 14.5.2 Data Transmission Continuously

For the data transmission, the I<sup>2</sup>C core used 32-bit transmit buffer and provide multi-byte transmit function. Set CSR[Tx\_NUM] to a value that you want to transmit. I<sup>2</sup>C core will always issue a transfer from the highest byte first. For example, if CSR[Tx\_NUM] = 0x3, Tx[31:24] will be transmitted first, then Tx[23:16], and so on.

```
CSR |= (0x3 << 4); //Four bytes transmission, data sequence is 0x12, 0x34, 0x56, 0x78
u32Data = 0x12 << 24;
u32Data |= (0x34 << 16);
u32Data |= (0x56 << 8);
u32Data |= 0x78;
TxR = u32Data; //Write 4 bytes in TX register
```

#### 14.5.3 Interrupt

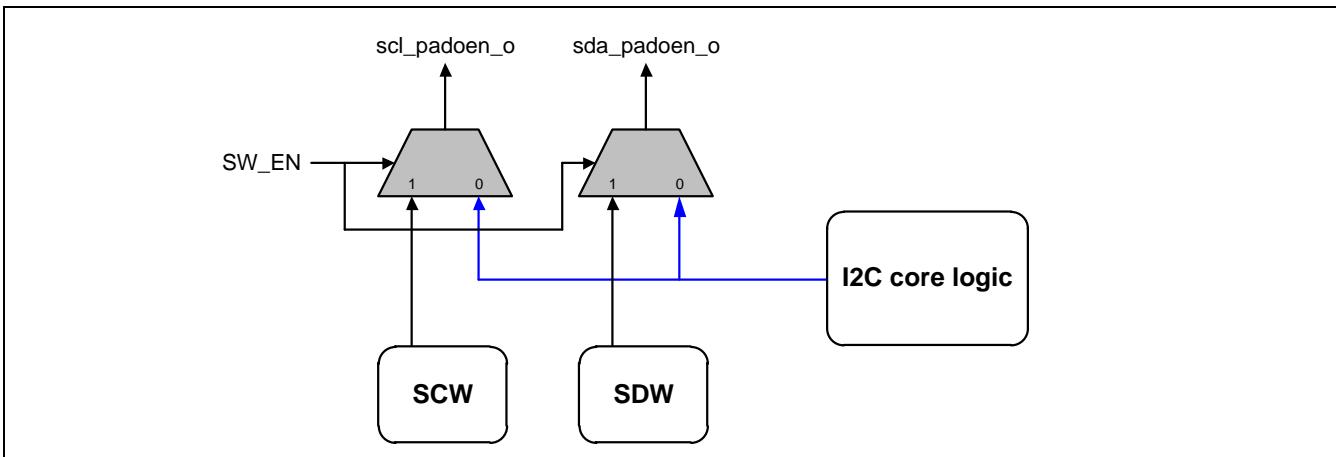
The interrupt flag IF(CSR [2]) bit will be set to 1 after I<sup>2</sup>C controller finished transmit or receive. If interrupt enable bit IE(CSR[1]) is also set to 1 and interrupt will occur. IF bit can be cleared by writing 1 to itself.

```
CSR |= 0x2; //Enable Interrupt
```

I<sup>2</sup>C\_AL(CSR[9]) flag will be set to 1 if I<sup>2</sup>C arbitration lost error happens. If IE bit is also set to 1, interrupt will also happen. At this time, software needs to send STOP command to release bus and unfinished job needs to restart.

#### 14.5.4 Software Mode

Function block is shown below,



To use I<sup>2</sup>C software control mode, need to set I2C\_EN(CSR[0]) to 0. And software can configure SCW(SWR[0]) bit, SDW(SWR[1]) bit and pull I<sup>2</sup>C SCL or SDA pin high or low.

Software also can know the status of pins by reading SCR(SWR[3]) bit or SDR(SWR[4]) bit.

```
CSR &= ~0x1; //Disable hardware I2C function
//START
SWR |= 0x3; //SCL=high, SDA=high
sleep();
SWR &= ~0x2; //SCL=high, SDA=low
sleep();
SWR &= ~0x1; // SCL=low, SDA=low
sleep();
...
...
```

#### 14.5.5 I<sup>2</sup>C Operation Using CMDR Register

CMDR is used to control I<sup>2</sup>C START/STOP , READ/WRITE , ACK/NACK command.

Control flow is list below:

1. Write data to Tx(TxR[7:0]) register
2. Set START(CMDR[4])bit and WRITE (CMDR[1]) bit
3. Wait for interrupt
4. Check ACK or NACK and set STOP(CMDR[3]) bit
5. Wait for interrupt
6. Finish

```
TxR = 0x12; //Slave address
CMDR |= (0x1 << 4) | (0x1 << 1); //Enable I2C WRITE
while(CSR & (0x1 << 8)); //Polling I2C status
if(!CMDR & 0x1)
```

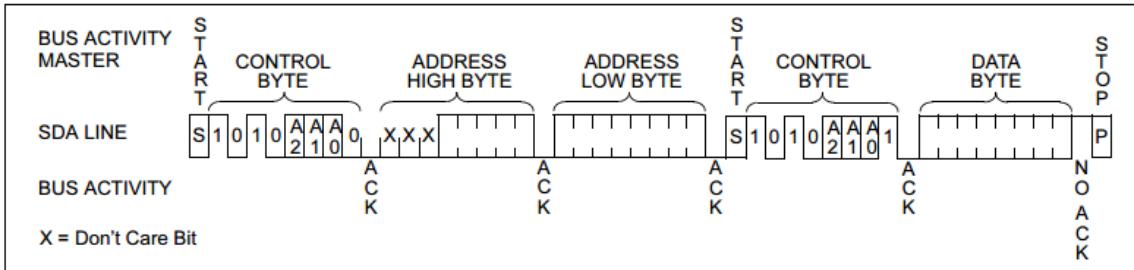
```

printf("Transfer error!!\n");
CMMDR |= (0x1 << 3);           //Set STOP bit, stop operation
while(CSR & (0x1 << 8));      //Polling I2C status

```

#### 14.5.6 I<sup>2</sup>C EEPROM Operation Example

Random Read:



Example code :

```

CSR &= ~(0x3 << 4);          //Tx_NUM=0
CSR |= 0x1;                   //Enable hardware I2C function

TxR = 0x50;                  //24LC64 slave address / write
CMDR = (0x1 << 4) | (0x1 << 1); //Set START and WRITE bit
while(CSR & (0x1 << 8));    //Polling I2C status
if(!CMDR & 0x1)
    printf("Transfer error!!\n");

TxR = 0x00;                  //High byte address
CMDR |= (0x1 << 1);        //Set WRITE bit
while(CSR & (0x1 << 8));    //Polling I2C status
if(!CMDR & 0x1)
    printf("Transfer error!!\n");

TxR = 0x01;                  //Low byte address
CMDR |= (0x1 << 1);        //Set WRITE bit
while(CSR & (0x1 << 8));   //Polling I2C status
if(!CMDR & 0x1)
    printf("Transfer error!!\n");

TxR = 0x51;                  //24LC64 slave address / read
CMDR |= (0x1 << 4) | (0x1 << 1); //Set START and WRITE bit (repeat START)
while(CSR & (0x1 << 8));    //Polling I2C status
if(!CMDR & 0x1)
    printf("Transfer error!!\n");

```

```
CMDR |= (0x1 << 3) | (0x1 << 2) | (0x1 << 2); //Set STOP, READ, ACK bit  
u32data = RxR; //Read data back
```

## 15 I<sup>2</sup>S

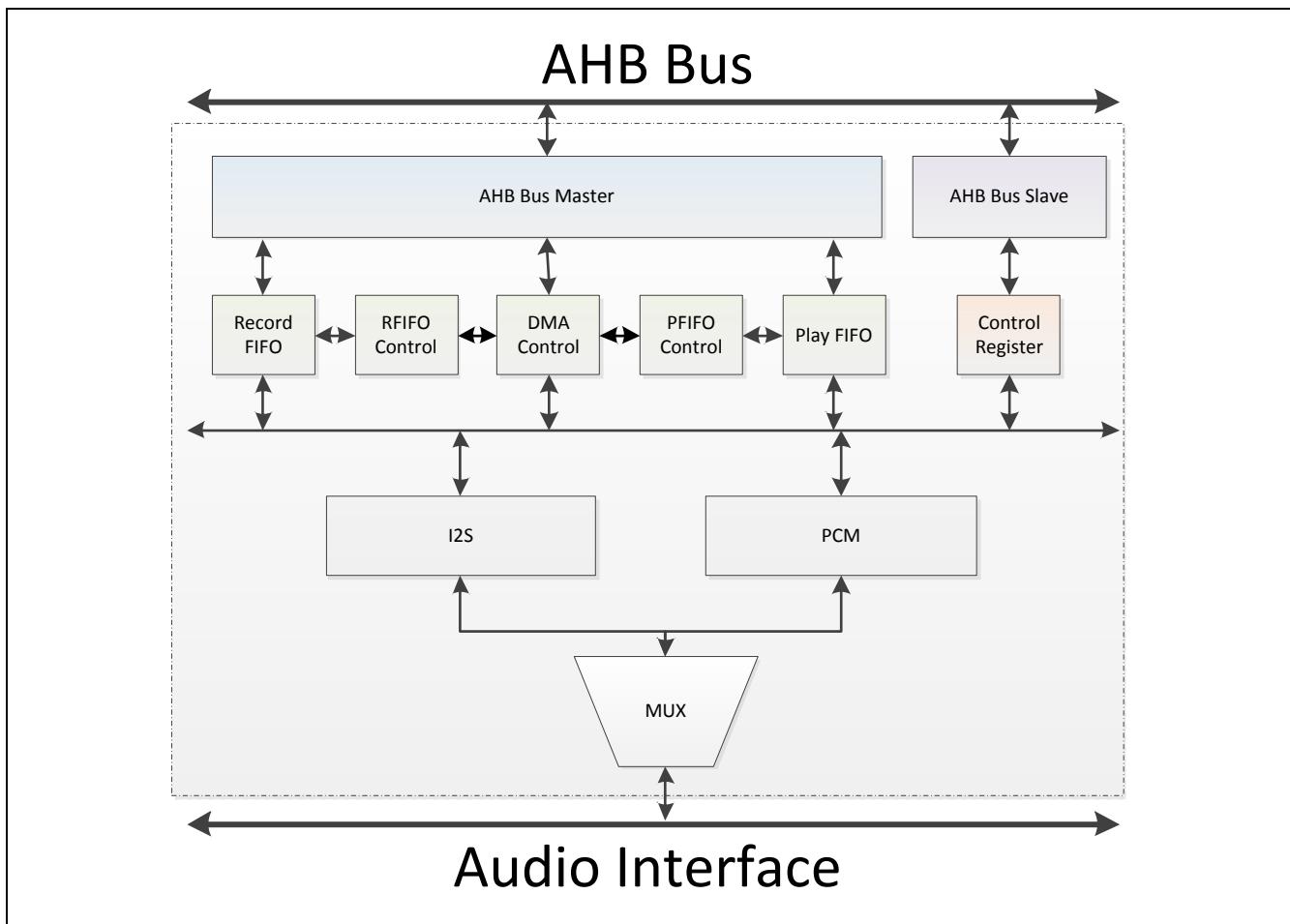
### 15.1 Overview

The I<sup>2</sup>S controller consists of I<sup>2</sup>S and PCM protocols to interface with external audio CODEC. The I<sup>2</sup>S and PCM interface supports 8, 16, 18, 20 and 24-bit left/right precision in record and playback. When operating in 18/20/24-bit precision, each left/right-channel sample is stored in a 32-bit word. Each left/right-channel sample has 24/20/18 MSB bits of valid data and other LSB bits are the padding zeros. When operating in 16-bit precision, right-channel sample is stored in MSB of a 32-bit word and left-channel sample is stored in LSB of a 32-bit word.

### 15.2 Features

- Support I<sup>2</sup>S interface record and playback
  - Left/right channel
  - 8, 16, 20, 24-bit data precision
  - Support master and slave mode
- Support PCM interface record and playback
  - Two slots
  - Support 8,16,18, 20,24-bit data precision
  - Support master mode
  - Support two addresses for left/right channel data and different slots
- Use DMA to playback and record data, with interrupt

### 15.3 Function Block



## 15.4 Register Map

Register	Offset	R/W	Description	Reset Value
<b>I<sup>2</sup>S Base Address:</b>				
I <sup>2</sup> S_BA = 0xB000_9000				
I <sup>2</sup> S_GLBCON	I <sup>2</sup> S_BA+0x000	R/W	I <sup>2</sup> S Global Control Register	0x0000_0000
I <sup>2</sup> S_RESET	I <sup>2</sup> S_BA+0x004	R/W	I <sup>2</sup> S Sub Block Reset Control Register	0x0000_0000
I <sup>2</sup> S_RDESB	I <sup>2</sup> S_BA+0x008	R/W	I <sup>2</sup> S Record DMA Destination Base Address Register	0x0000_0000
I <sup>2</sup> S_RDES_LENGTH	I <sup>2</sup> S_BA+0x00C	R/W	I <sup>2</sup> S Record DMA Destination Length Register	0x0000_0000
I <sup>2</sup> S_RDESC	I <sup>2</sup> S_BA+0x010	R	I <sup>2</sup> S Record DMA Destination Current Address Register	0x0000_0000
I <sup>2</sup> S_PDESB	I <sup>2</sup> S_BA+0x014	R/W	I <sup>2</sup> S Play DMA Destination Base Address Register	0x0000_0000
I <sup>2</sup> S_PDES_LENGTH	I <sup>2</sup> S_BA+0x018	R/W	I <sup>2</sup> S Play DMA Destination Length Register	0x0000_0000
I <sup>2</sup> S_PDESC	I <sup>2</sup> S_BA+0x01C	R	I <sup>2</sup> S Play DMA Destination Current Address Register	0x0000_0000
I <sup>2</sup> S_RSR	I <sup>2</sup> S_BA+0x020	R/W	I <sup>2</sup> S Record Status Register	0x0000_0000
I <sup>2</sup> S_PSR	I <sup>2</sup> S_BA+0x024	R/W	I <sup>2</sup> S Play Status Register	0x0000_0000

I2S_CON	I2S_BA+0x028	R/W	I2S Control Register	0x0000_0000
I2S_COUNTER	I2S_BA+0x02C	R/W	I2S Play DMA Down Counter Register	0xFFFF_FFFF
I2S_PCMCON	I2S_BA+0x030	R/W	I2S PCM Mode Control Register	0x0000_0000
I2S_PCMS1ST	I2S_BA+0x034	R/W	I2S PCM Mode Slot 1 Start Register	0x0000_0000
I2S_PCMS2ST	I2S_BA+0x038	R/W	I2S PCM Mode Slot 2 Start Register	0x0000_0000
I2S_RDESB2	I2S_BA+0x040	R/W	I2S Record DMA Destination Base Address 2 Register	0x0000_0000
I2S_PDESB2	I2S_BA+0x044	R/W	I2S Play DMA Destination Base Address 2 Register	0x0000_0000

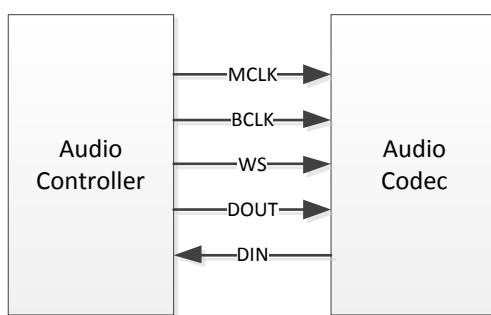
## 15.5 Functional Description

### 15.5.1 I<sup>2</sup>S Master/Slave Mode

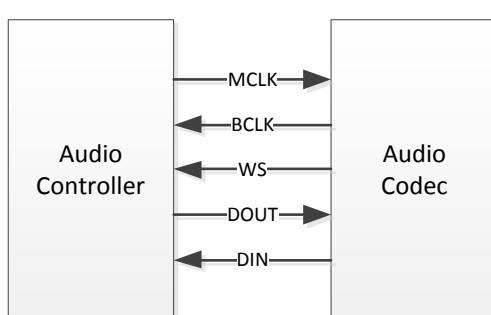
To use slave mode, user can set SLAVE(I2S\_CON[20]) bit to 1 otherwise set 1 to be as master mode.

Note that slave only can be chosen when use I<sup>2</sup>S interface and only use master mode if PCM interface is used.

Master mode connection between controller and audio codec:



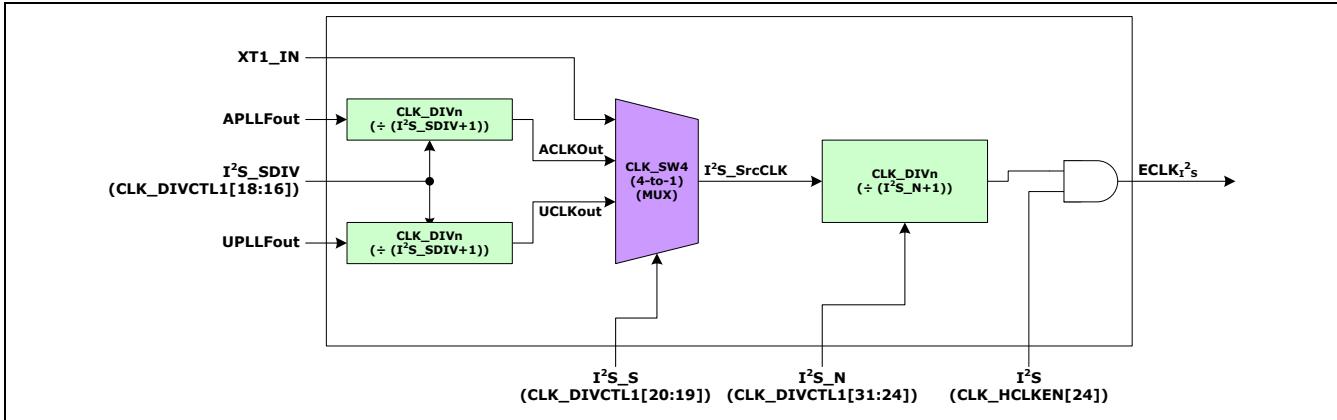
Slave mode connection between controller and audio codec:



### 15.5.2 I<sup>2</sup>S Source Clock Configuration

Software can choose APLL, UPLL or external crystal as source clock of I<sup>2</sup>S by configuring

I<sup>2</sup>S\_S(CLK\_DIVCTL1[20:19]).



```
CLK_DIVCTL1 = (CLK_DIVCTL1 & ~(0x3 << 19)) | 0x2; //I2S source clock is APLL
```

### 15.5.3 I<sup>2</sup>S Calculation and Configuration of Clock

The clocks in I<sup>2</sup>S need to be configured are MCLK and BCLK. Only MCLK needs to be configured when using I<sup>2</sup>S slave mode.

In general, to get the accurate clock, suggest using PLL and set speed to 12.288MHz, 16.934MHz or 11.285MHz.

The following is an example to let user know how to get 48 kHz sampling rate when 16-bit data and stereo channel are used.

If audio codec supports 256x sampling rate, the calculation of MCLK is as below:

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

And if use 16-bit data width and stereo channel, the calculation of BCLK is as below:

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

So the divider PSR(I2S\_CON[19:16]) is  $12.288/12.288 - 1 = 1 - 1 = 0$

And BCLK\_DIV(I2S\_CON[7:5]) is  $(12.288/1.536)/2-1 = 8/2-1 = 3$

```
I2S_CON = I2S_CON & ~(0xF << 16);           //PRS=0
I2S_CON = I2S_CON & ~(0x1 << 4);            //MCLK comes from divide PLL by PRS
I2S_CON = (I2S_CON & ~(0x1 << 5)) | 0x3;     //BCLK_DIV=3
```

### 15.5.4 DMA

I<sup>2</sup>S use DMA to implement playing and recording. The description of DMA operation and

configuration list as below:

- Play and record DMA base address (I2S\_RDESB and I2S\_PDESB). All the play and record data will be put in the address, in general, this space is somewhere in RAM which is continuous and non-cacheable.
- DMA length register (I2S\_RDES\_LENGTH and I2S\_PDES\_LENGTH), is the total length of DMA space.
- DMA current address register (I2S\_RDESC and I2S\_PDESC) will show the current DMA address which is playing or recording. Software can use this to determine how much buffer can be used at this time.
- Software can decide when (1/2, 1/4 or 1/8 of DMA length) interrupt will occur by configuring R\_DMA\_IRQ\_SEL(I2S\_GLBCON[15:14]), P\_DMA\_IRQ\_SEL(I2S\_CON[13:12]) and enabling DMA\_IRQ\_EN(I2S\_GLBCON[21]) or P\_DMA\_IRQ\_EN(I2S\_GLBCON[20]) bit.

DMA configuration example list as below:

```
I2S_PDESB = 0x80001000;           //Assign play base address
I2S_PDES_LENGTH = 2*1024;        //DMA length is 2048 bytes
I2S_CON = (I2S_CON & ~(0x3 << 12)) | (0x1 << 12); //Interrupt will occur when DMA reach
1/2 of DMA length
I2S_CON |= (0x1 << 20);       //Enable interrupt
```

- DMA section number: Software can read P\_DMA\_RIA\_SN(I2S\_PSR[7:5]) or R\_DMA\_RIA\_SN(I2S\_RSR[7:5]) bit to know which DMA section that DMA is playing or recording. If the value read from P\_DMA\_RIA\_SN is 2 and P\_DMA\_IRQ\_SEL is b'11, that means that DMA is playing at the 2/8 section.
- DMA down counter: Software can read down counter register(I2S\_COUNTER) to know how much data had been played or recorded. When DMA transfers one data and down counter register will decrease one until it becomes zero. When down counter value becomes zero, software can enable IRQ\_DMA\_CNTEN(I2S\_GLBCON[4]) bit to let interrupt happen.

```
I2S_COUNTER = 0x1000;           //Set down count value to x1000
...
while(I2S_COUNTER>0x30);      //Test if the value is smaller than 0x30
...
```

- Zero crossing detection: When playing the audio by I<sup>2</sup>S function, the output data comes from the memory by DMA. However, it may result some pop noise if the playing gain level is changed by user at any time. Because, the output data is not zero, and the output data cross the gain change will generate a sharp pop noise. Therefore, the zero crossing

function will help to reduce this situation. Software can enable this function by setting DMA\_DATA\_ZERO\_EN(I2S\_RESET[3]) to 1 and also interrupt can be enabled by setting IRQ\_DMA\_DATA\_ZERO\_EN(I2S\_GLBCON[3]) to 1.

### 15.5.5 Sequence of DMA Data

When use I<sup>2</sup>S 18, 20, 24-bits, each data stored in DMA buffer will all use 32-bit width.

Take I<sup>2</sup>S 16-bit as an example:

Dual channels(Stereo) :

Base Address	DMA Buffer
0x1000	Left channel – LSB byte
0x1001	Left channel – MSB byte
0x1002	Right channel – LSB byte
0x1003	Right channel – MSB byte
0x1004	Left channel – LSB byte
0x1005	Left channel – MSB byte
0x1006	Right channel – LSB byte
0x1007	Right channel – MSB byte
...	...

Single channel(Mono) :

Base Address	DMA Buffer
0x1000	Left channel – LSB byte
0x1001	Left channel – MSB byte
0x1002	Left channel – LSB byte
0x1003	Left channel – MSB byte
0x1004	Left channel – LSB byte
0x1005	Left channel – MSB byte
0x1006	Left channel – LSB byte
0x1007	Left channel – MSB byte
...	...

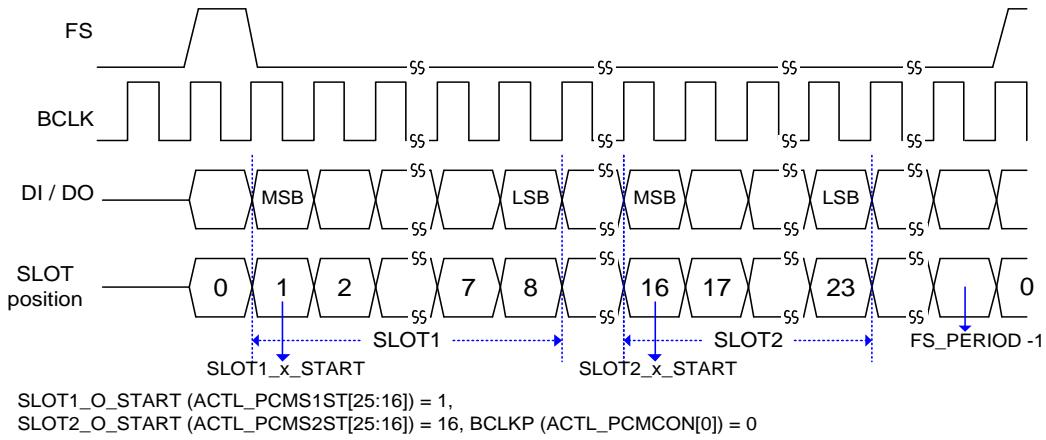
### 15.5.6 Interface Selection

Software can choose I<sup>2</sup>S or PCM interface by setting BLOCK\_EN(I2S\_GLBCON[0]) bit.

```
I2S_GLBCON = (I2S_GLBCON & ~0x3) | 0x2; //Choose PCM interface
```

### 15.5.7 PCM Interface

The following figure is PCM timing wave form,



And arguments that software can configure are:

1. Bit number between two FS – FS\_PERIOD(I2S\_PCMCON[25:16]).
2. Bit number between SLOT1\_x\_START or SLOT2\_x\_START and FS – I2S\_PCMS1ST or I2S\_PCMS2ST.

Take 8KHz sampling rate and data width is 32-bit for example. If two slots are used and assume clock speed of source clock is 24.576MHz.

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

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

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

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

### 15.5.8 Data Split

Data split function can put the continuous data into different DMA buffer by channel or slot. Software can process these data in single buffer address easier than two different addresses.

Software needs to set the second DMA base address register (I2S\_RDESB2 and I2S\_PDESB2). The data in first DMA address which specified by I2S\_RDESB and I2S\_PDESB register is I<sup>2</sup>S left channel or PCM slot1. The data in second DMA address is I<sup>2</sup>S right channel or PCM slot2.

To reach the target mentioned before, software can set SPLIT\_DATA(I2S\_RESET[20]) bit to 1 to enable this function. After enabling data split function, layout of data stored in buffer will like the following table (take I<sup>2</sup>S interface for example).

Base address-1	DMA Buffer
0x1000	Left channel – LSB byte
0x1001	Left channel – MSB byte
0x1002	Left channel – LSB byte
0x1003	Left channel – MSB byte
0x1004	Left channel – LSB byte
0x1005	Left channel – MSB byte
0x1006	Left channel – LSB byte
0x1007	Left channel – MSB byte
...	...

Base address-2	DMA Buffer
0x2000	Right channel – LSB byte
0x2001	Right channel – MSB byte
0x2002	Right channel – LSB byte
0x2003	Right channel – MSB byte
0x2004	Right channel – LSB byte
0x2005	Right channel – MSB byte
0x2006	Right channel – LSB byte
0x2007	Right channel – MSB byte
...	...

## 16 JPEG Codec

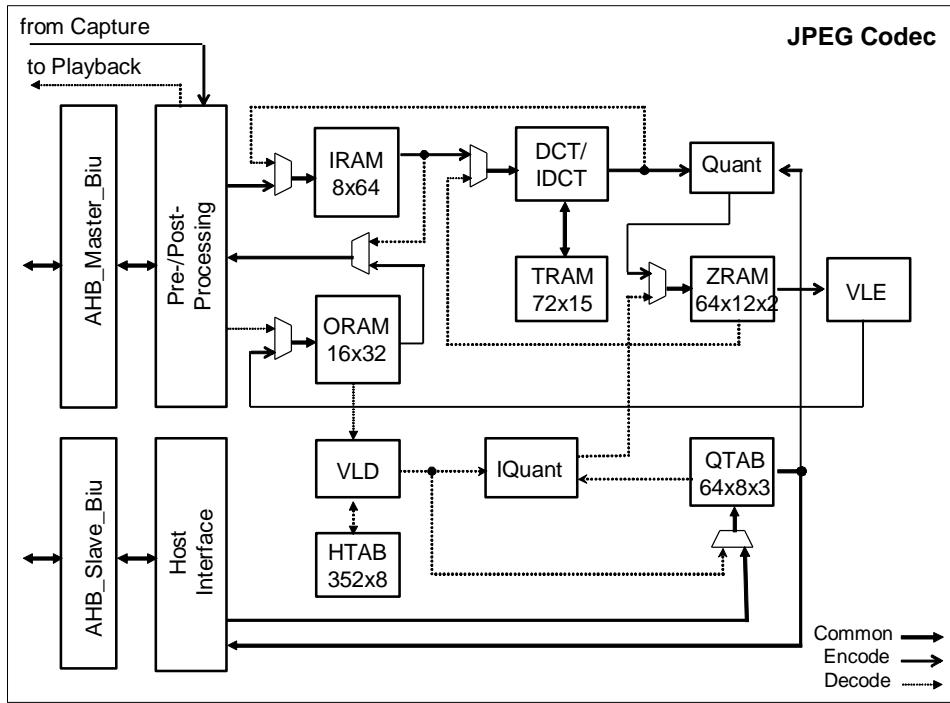
### 16.1 Overview

The JPEG Codec supports Baseline Sequential Mode JPEG still image compression and decompression that is fully compliant with ISO/IEC International Standard 10918-1 (T.81).

### 16.2 Feature

- Support to encode interleaved YCbCr 4:2:2/4:2:0 and gray-level (Y only) format image
- Support to decode interleaved YCbCr 4:4:4/4:2:2/4:2:0/4:1:1 and gray-level (Y only) format image
- Support to decode YCbCr 4:2:2 transpose format
- The encoded JPEG bit-stream format is fully compatible with JFIF and EXIF standards
- Support Capture and JPEG hardware on-the-fly access mode for encode
- Support JPEG and Playback hardware on-the-fly access mode for decode
- Support software input/output on-the-fly access mode for both encode and decode
- Support arbitrary width and height image encode and decode
- Support three programmable quantization-tables
- Support standard default Huffman-table and programmable Huffman-table for decode
- Support arbitrarily 1X~8X image up-scaling function for encode mode
- Support down-scaling function for encode and decode modes
- Support specified window decode mode
- Support quantization-table adjustment for bit-rate and quality control in encode mode
- Support rotate function in encode mode

### 16.3 Block Diagram



## 16.4 Register Map

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
<b>JPG_BA = 0xB000_A000</b>				
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

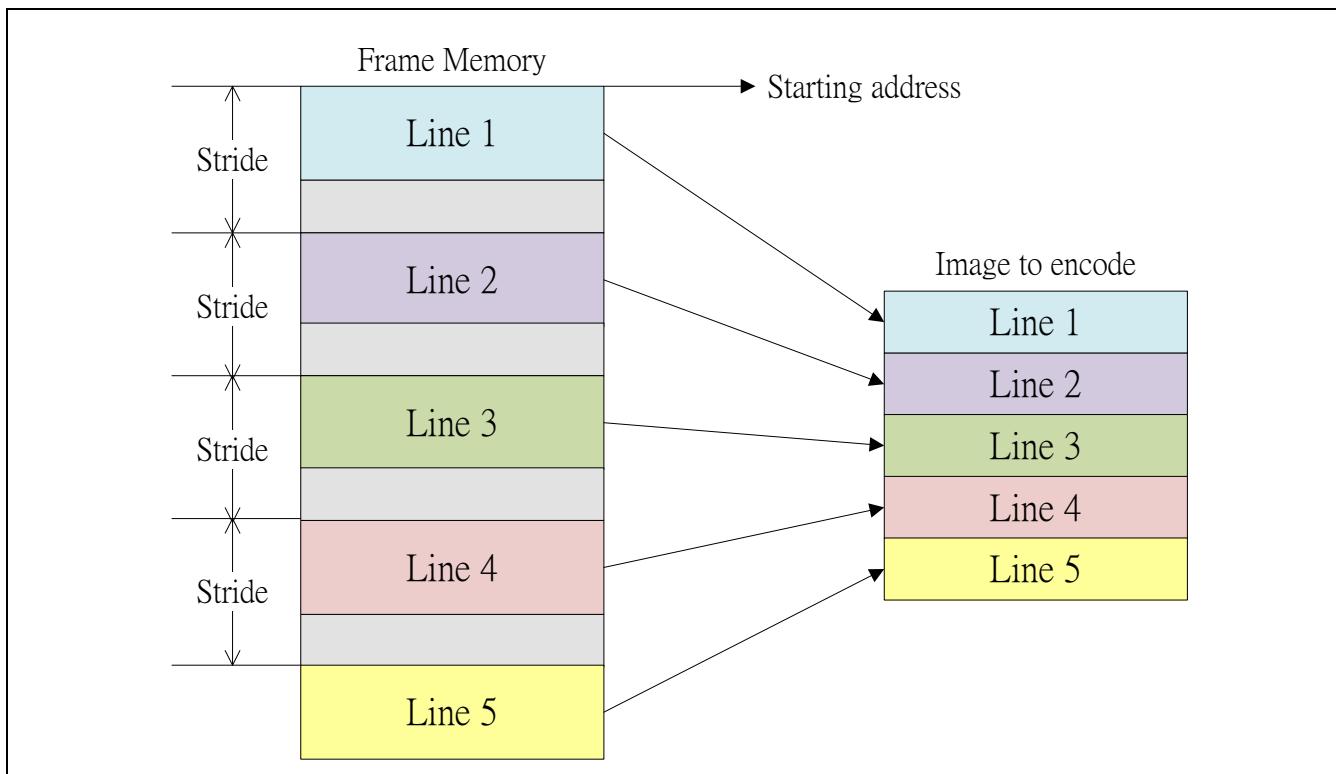
<b>JDOWFBS</b>	JPG_BA + 03C	R/W	Decoding Output Wait Frame Buffer Size	0xFFFF_FFFF
<b>JTEST</b>	JPG_BA + 040	R/W	JPEG Test Control Register	0x0000_0000
<b>JWINDEC0</b>	JPG_BA + 044	R/W	JPEG Window Decode Mode Control Register 0	0x0000_0000
<b>JWINDEC1</b>	JPG_BA + 048	R/W	JPEG Window Decode Mode Control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

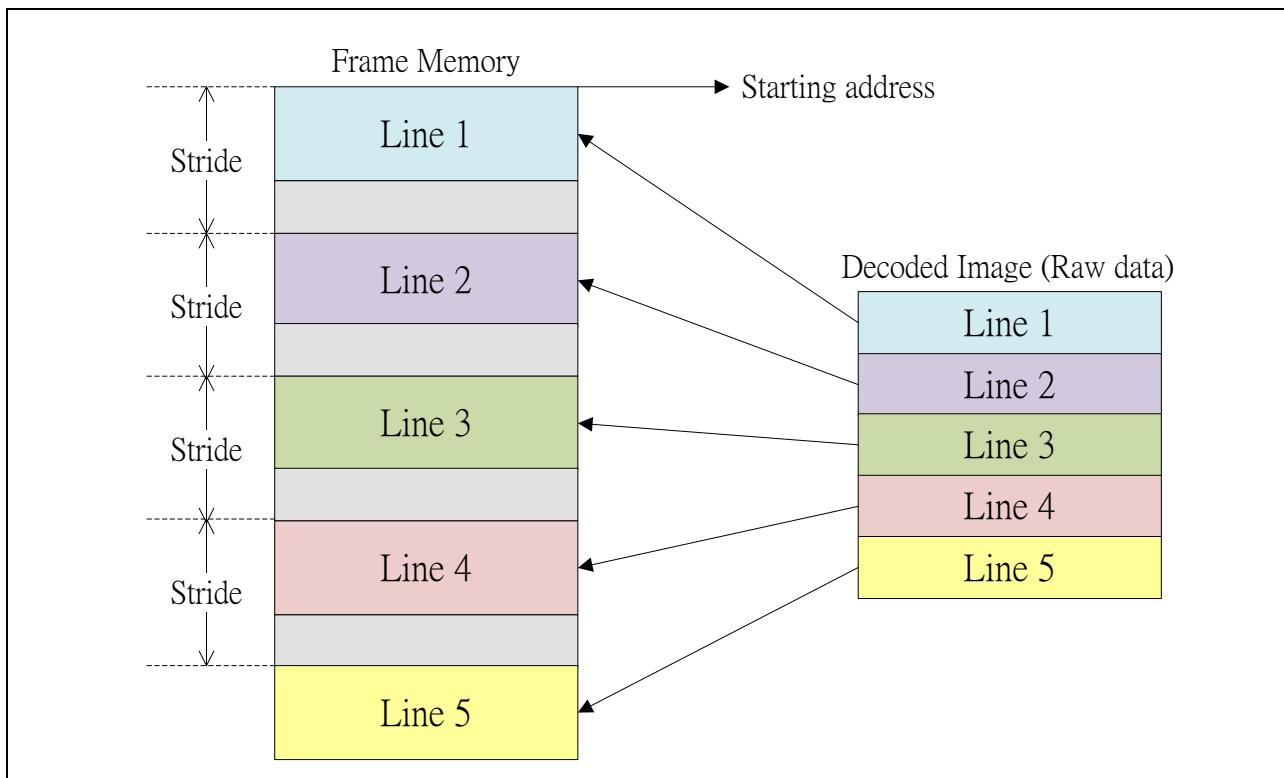
## 16.5 Functional Description

### 16.5.1 Memory Access

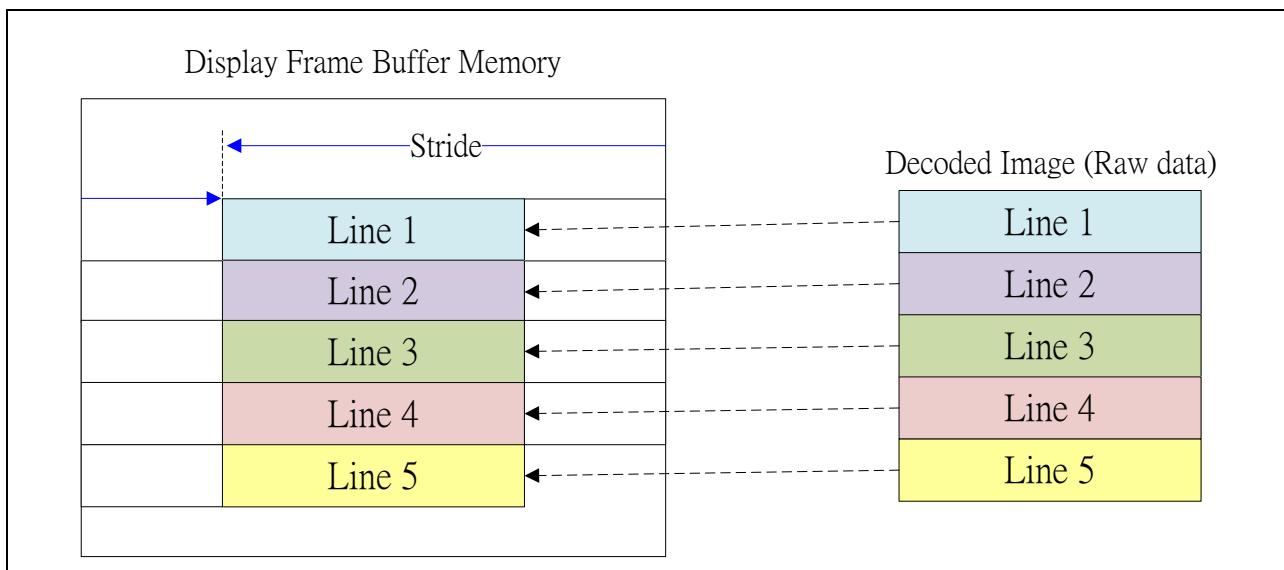
Following figure shows the encode mode to access the source data which are from sensor normally and stored on the RAM.



Following figure shows the decode mode to output the decoded raw data on the RAM.



User can use stride function to output decoded image to any position on the Display Frame Buffer for Display. Following figure shows the decode mode with stride to output the decoded raw data on the Display Frame Buffer.



## 16.5.2 JPEG Encoding

### 16.5.2.1 Reset Jpeg Engine

The JPEG engine supports to encode/decode JPEG bit-stream. Before trigger Jpeg engine to decode or encode jpeg bit-stream, remember to reset Jpeg engine by set JMCR[1] = 1,then JMCR[1] = 0 first. Otherwise, the result may be wrong.

### 16.5.2.2 Quantization Table

- Quantization Table Order

The Quantization table for the register order is different from the order in bit-stream.

For example, the Quantization table is as following table:

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

The JPEG Quantization table 0 is written in the order 0, 1, 5, 6, 14, 15, 27, 28 ..... 63. However, the Quantization table in bit-stream is in the zig-zag order 0, 1, 2, 3, 4, 5.....63.

- Write Quantization Table

Before writing the JPEG Quantization table, be sure that the Quantization-Table Busy Status equal to 0 (JMCR[2] =0). If writing JPEG Quantization-table (JPB\_BA+0x100 ~ 0x17F) when JMCR[2] =1, the writing may be fail.

## 16.5.3 Normal Encoding

The setting sequence of encode process will be:

- Reset JPEG engine
- Write Quantization tables
- Set input frame buffer start address

- Planar format : Y/U/V start address (JYADDR0 / JUADDR0 / JVADDR0)
  - Packet format : data start address (JYADDR0)
4. Set output bit-stream target address. (JIOADDR0)
  5. Set Encode Image Dimension Property
    - Set encode primary image width/height (JPRIWH)
    - Set image height to Encode Source Image Height (JSRCH)
  6. Set Encode Image Raw data property
    - Planar format
      - ◆ Y/U/V buffer stride (JYSTRIDE / JUSTRIDE / JVSTRIDE)
    - Packet format
      - ◆ buffer stride (JYSTRIDE is image width and, JUSTRIDE/JVSTRIDE is image width/2)
  7. Set Encode mode and format
    - Set encode mode. (JMCR[7] =1)
    - Encode Primary image (JMCR[5])
    - Encode Thumbnail image (JMCR[4])
    - Encode format (JMCR[3])
      - ◆ YUV422
      - ◆ YUV420
  8. Set Encode header (JHEADER)
  9. Enable interrupt for Encode (JINTCR)
    - Encode complete interrupt (ENC\_INTE)
  10. Trigger for Encode (JINTCR)
    - Set JMCR[0] to 1, then clear to 0.
  11. Wait for Encode Complete Interrupt

#### 16.5.4 Encoding Scaling up

##### Encode Scaling up ratio

The Width and Height are the original picture, Upscale\_Width and Upscale\_Height are the scaling-up picture:

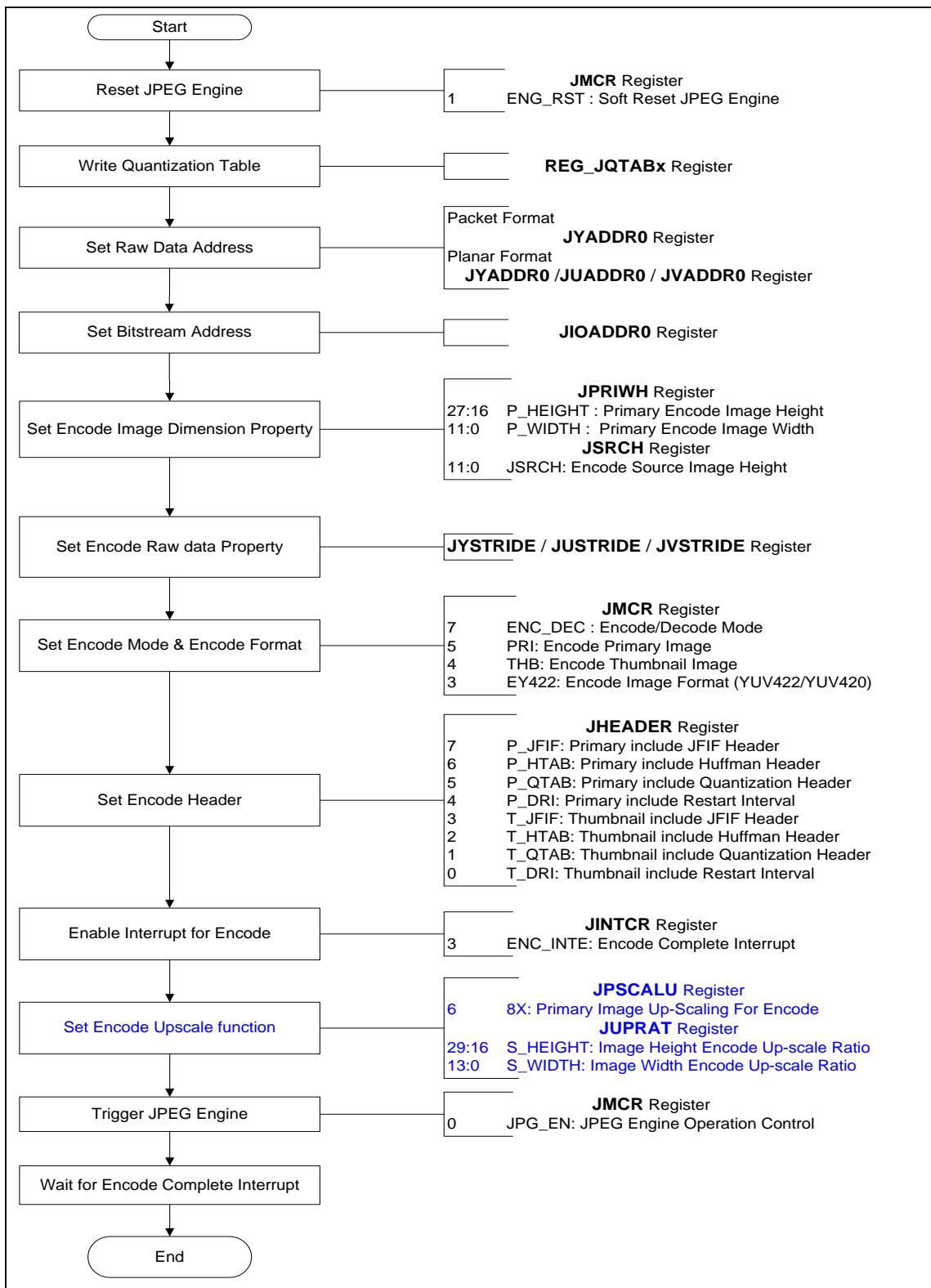
$$\text{X ratio} = ((\text{Upscale\_Width} - 1) / (\text{Width} - 1)) * 1024$$

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

If user wants to enable encode upscale function, user can do the following settings any time before trigger JPEG engine.

1. Set the Scaling up ratio (JUPRAT)
  - S\_HEIGHT: Image Height Encode Up-scale Ratio
  - S\_WIDTH: Image Width Encode Up-scale Ratio
2. Enable encode scaling up (JPSCALU[6] = 1)
  - 8X: Primary Image Up-Scaling For Encode

Due to the source data is packet format, sometimes the final column data may mix with the first column in next row. User should use larger ratio to avoid it.

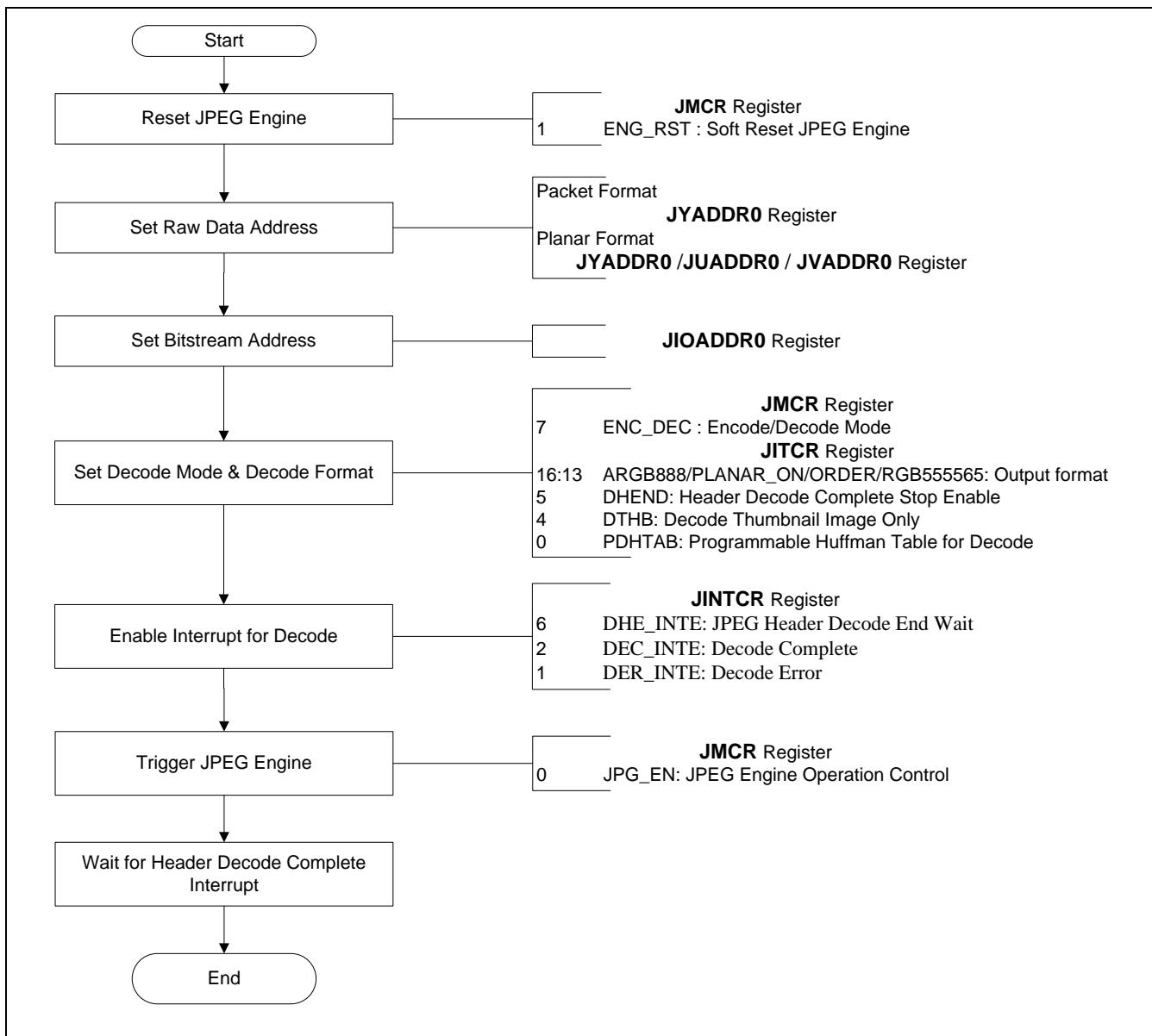


## 16.5.5 JPEG Decoding

### 16.5.5.1 Normal Decode

The Decode operation sequence can be divided two parts:

1. Reset JPEG engine
2. Set input bit-stream start address. (JIOADDR0)
3. Set Output YUV frame buffer address.
  - Planar format
    - ◆ Y/U/V start address (JYADDR0 / JUADDR0 / JVADDR0)
  - Packet format
    - ◆ Data start address (JYADDR0).  
YUV frame buffer address must be set before trigger JPEG engine for planar format.
    - YUV frame buffer address can be set before starting to output raw data for packet format.
4. Set Decode mode and Output format
  - Set decode mode (JMCR[7] = 0)
  - Set decode primary or thumbnail image (JITCR[4])
  - Enable Header Decode Complete Stop (JITCR[5])
  - Enable Programmable Huffman Table function for Decode (JITCR[0])
  - Set output format (JITCR[16:13])
    - ◆ Planar format
    - ◆ Packet format:  
RGB888/RGB565/RGB555/YUV422/RGB555R1/RGB555R2/RGB565R1/RGB565R2
5. Enable interrupt for decode (JINTCR)
  - Set DEC\_INTE to 1
  - Set DHE\_INTE to 1
  - Set DER\_INTE to 1
6. Trigger for Encode (JINTCR)
  - Set JMCR[0] to 1, then clear to 0
7. Wait for Header Decode Complete interrupt

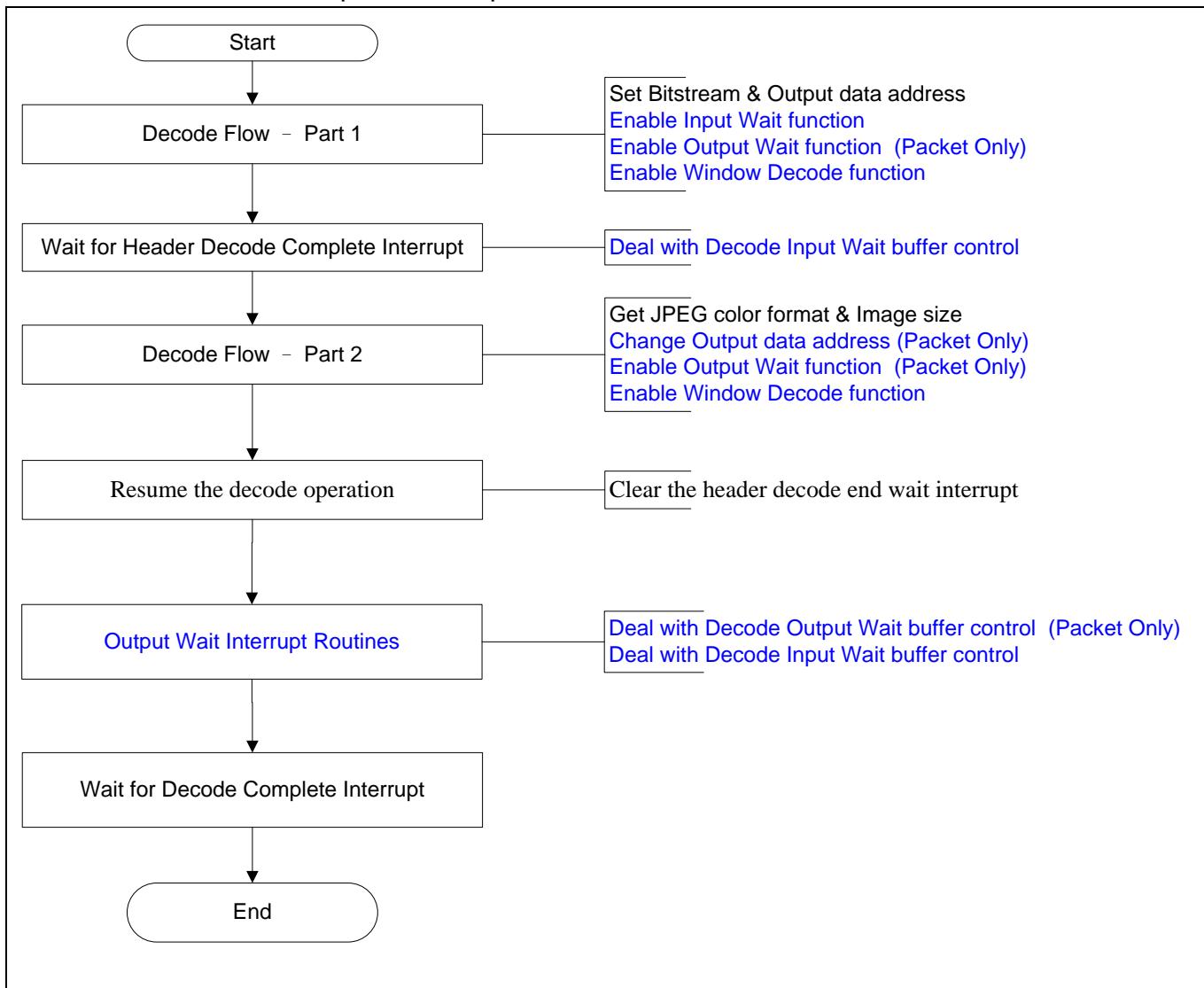


After getting the header decode complete interrupt, user can get YUV color format (JITCR[10:8]) and Image Width/Height (JDECWH).

Image Decode Flow as Follow:

1. Set decode image width/height (JPRIWH)
  - The value of width must adjust for JPEG format or set to the output stride.
2. Set the output data property (Packet format only)
  - Change output address (JYADDR0)
  - Output offset value to JYSTRIDE . If stride function is disabled, Stride is equal to width, the value of offset is 0.
3. Clear the header decode end wait interrupt to resume the decode operation

#### 4. Wait for Decode complete interrupt



##### 16.5.5.2 Decoding Scaling down

###### Decoding Scaling down ratio

Width and Height are for the original picture and Downscale\_Width, Downscale\_Height is for the picture after scaling-down

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

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

For example, the setting of downscaling 18000 x 18000 pixels to 640 x 480 pixels as follow:

- Horizontal factor =  $640 / 18000 * 8192 = 291$
- Vertical factor =  $480 / 18000 * 8192 = 218$

The ratio is as following equation for planar format: Width, Height is for the original picture and Downscale\_Width, Downscale\_Height is for the picture after scaling-down. (If the ratio is larger than 8192, please set it to 8192.)

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

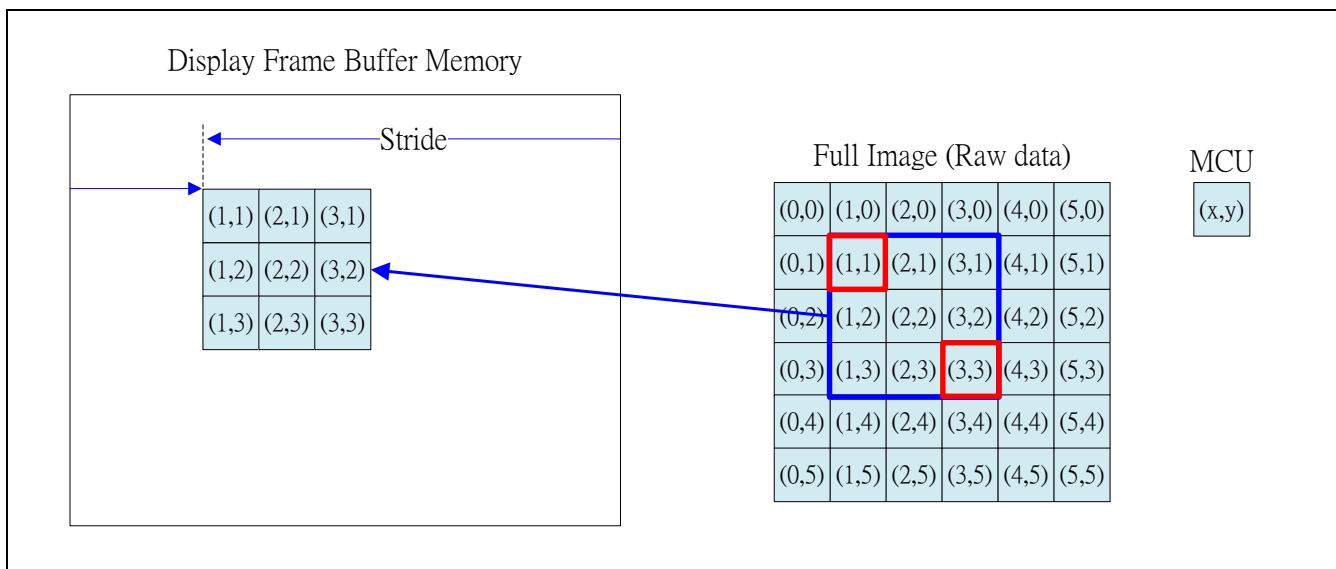
$$Y \text{ ratio} = \text{Height} - \text{Downscale\_Height} \quad (0 \leq Y \text{ ratio} \leq 63)$$

If user wants to enable decode down scale function, user can do the following settings any time before clearing the header decode end wait interrupt (starting to output data):

1. Set the Scaling down ratio
  - Packet format (JUPRAT)
  - Planar format (JPSCALD)
2. Enable decode scaling down (JPSCALD[15] = 1)

#### 16.5.5.3 Window Decode

The JPEG decoder supports specified window decode mode. This function allows user to specify a sub-window region within the whole image to be decoded as shown in the following figure. Only the specified window region image will be decoded and stored to frame memory. This function can be enabled by setting register bit WIN\_DEC, and the window region can be specified in registers JWINDEC0~JWINDEC2.



[Note 1]. The minimum window decode unit is 16x16.

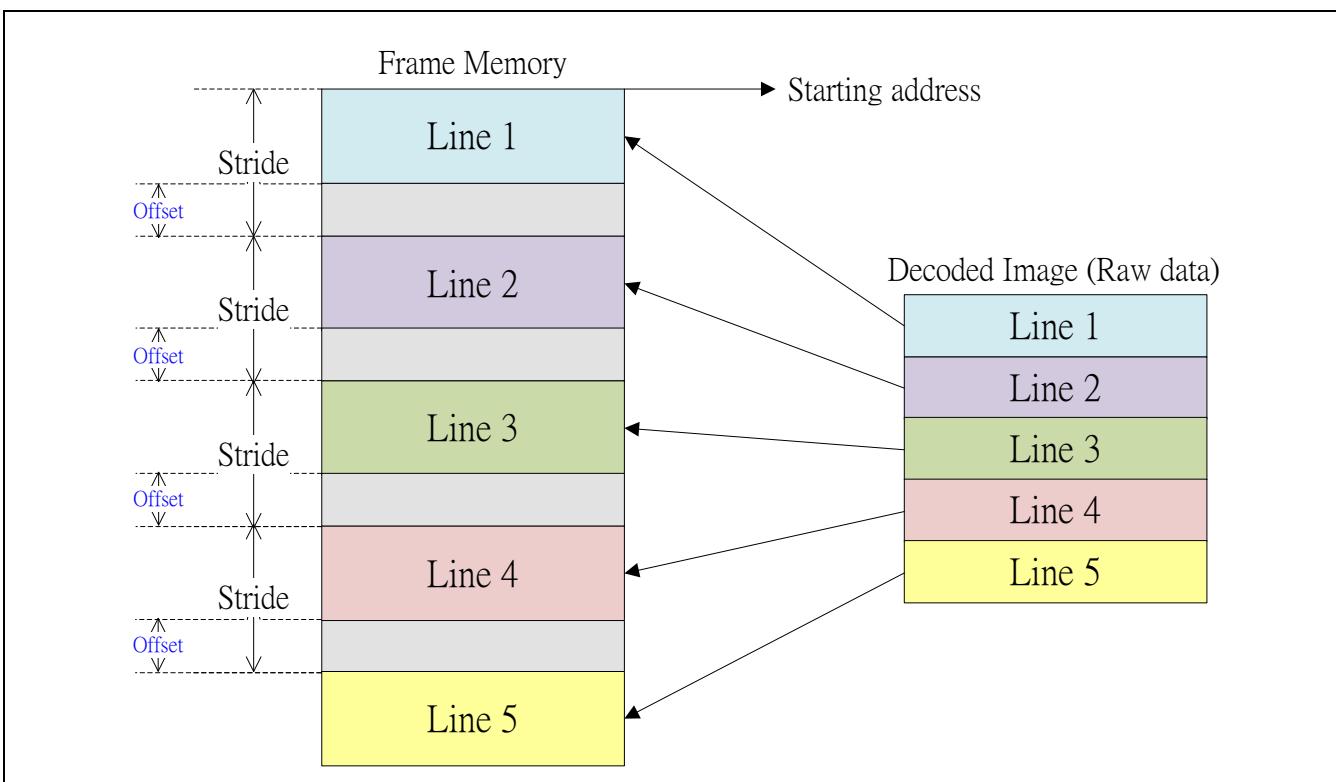
[Note 2]. The image region to be decoded must exists.

If user wants to enable decode down scale function, user can do the following settings any time before clearing the header decode end wait interrupt (starting to output data):

1. Set the region window of and End MCU
  - Start coordinate in MCU to JWINDEC0
  - End coordinate in MCU to JWINDEC1
  - Decode Stride to JWINDEC2

#### 16.5.5.4 Decode Stride Function (Packet Format Only)

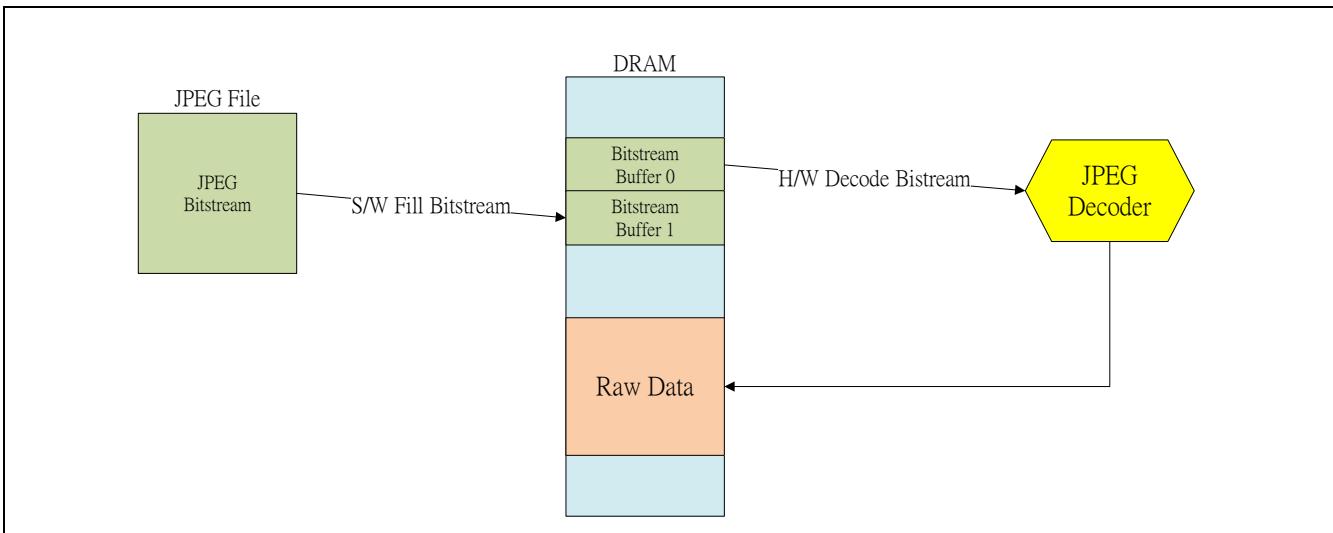
Before clearing Header Decode End interrupt, the value of stride must be set to JPRIWH instead of original width. Offset is the difference between Stride and Image width (JYSTRIDE). If Offset is 0, the decoded raw data is continuous.



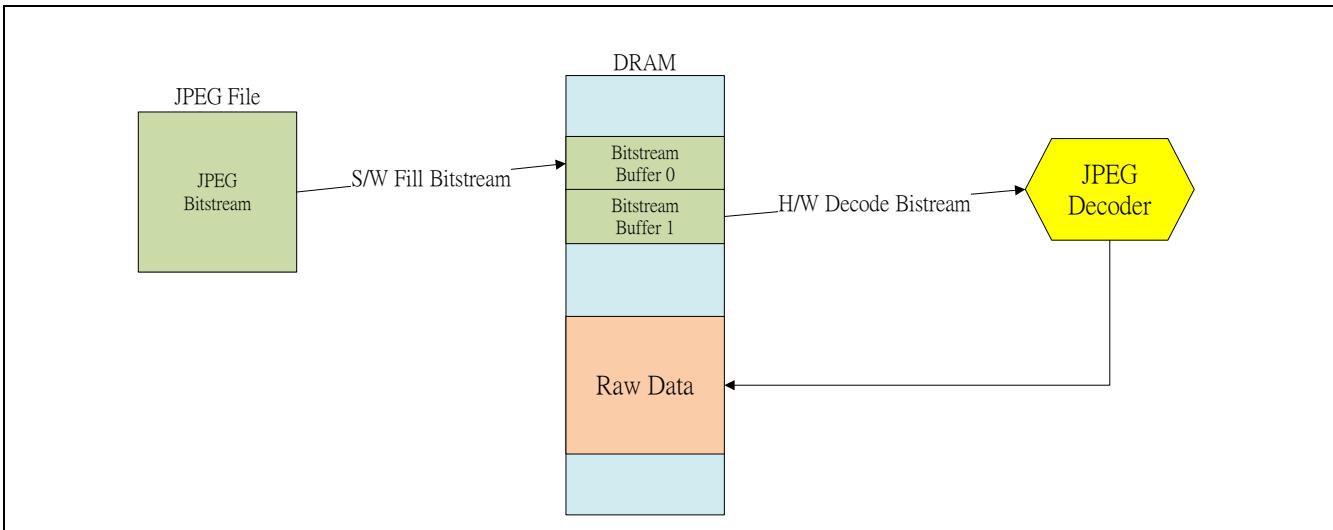
#### 16.5.5.5 Software Decode Input Wait

When the JPEG is in decoding mode, the input source is the JPEG bit-stream written by host. The bit-stream buffer size is in 2K unit dual-buffer manner. The total buffer size is decided by the BSF\_SEL of JMCR. If the buffer-size is 2KB (BSF\_SEL=1), host need to fill 1KB bit-stream into one of the half buffer region before resuming JPEG operation when an input-wait interrupt is generated.

1. JPEG engine decodes the data in Buffer 0 and S/W fills the data into Buffer 1.



2. JPEG engine decodes the data in Buffer 1 and S/W fills the data into Buffer 0.



3. Iterate Step 1 & 2 until getting Decode Complete interrupt.

#### 16.5.5.6 Decode Output Wait

When there is not enough continuous space to store the decode output raw data, JPEG engine support a function to output data partially. User can get the whole data by assigning several output data address and size settings. Using this function, user can get the JPEG decoded image that larger than the available continuous memory space.

If user wants to enable decode output wait function, user can do the following settings any time before clearing the header decode end wait interrupt (starting to output data).

Extra setting before clearing the Header Decode end wait interrupt is as follows:

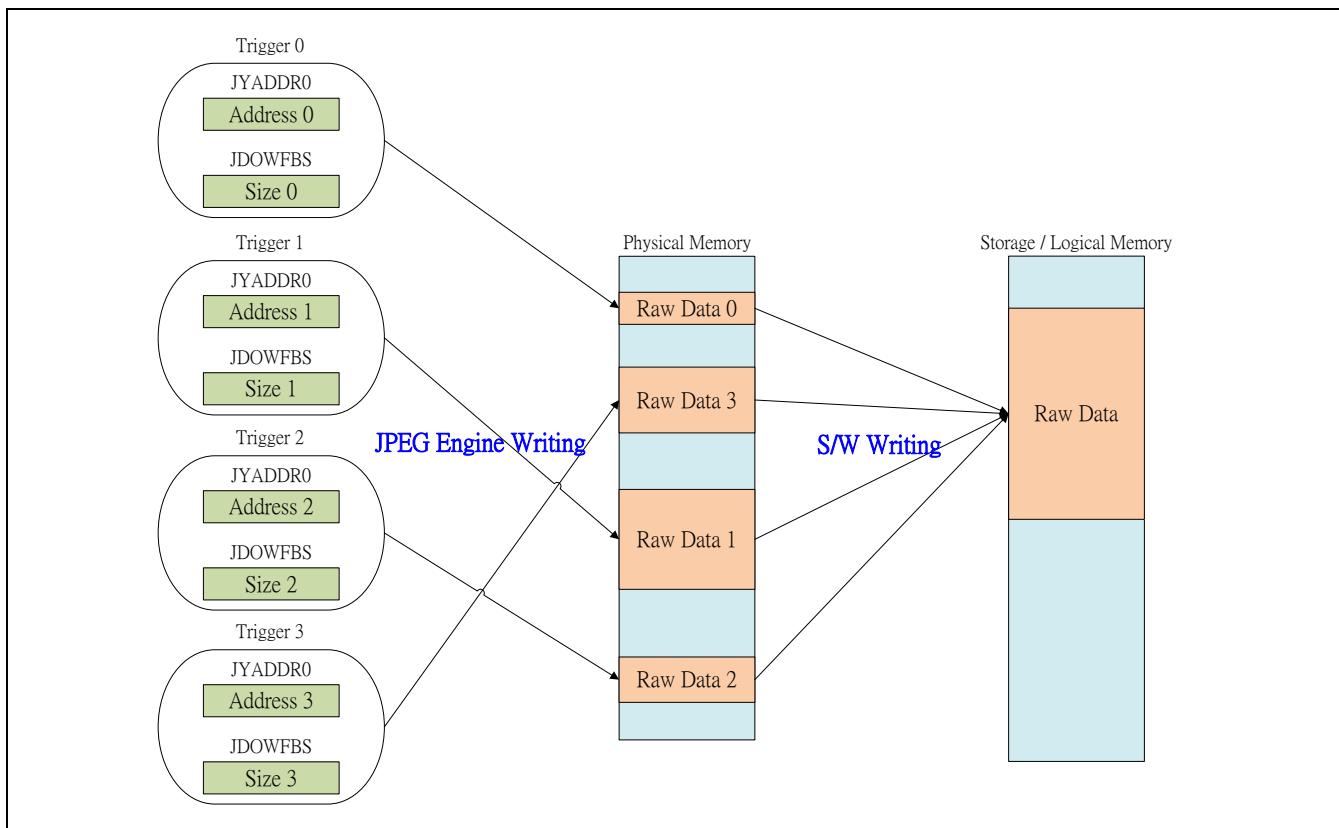
1. Set Output Wait address & size

- Address: JYADDR0
  - Set decoded output data size: JDOWFBS
    - ◆ Unit is Word.
    - ◆ Must multiple of MCU line data size
  - 2. Set interrupt enable for decode output wait (JINTCR)
    - Set DOW\_INTE to 1 (Decoding Output Wait Interrupt Enable)
  - 3. Trigger Decode Output Wait
    - Set Dec\_Scatter\_Gather (JITCR[18]) to 1
- [Note1] Dec\_Scatter\_Gather only be clear
- When output size is equal to the value of JDOWFBS
  - IP Reset.

[Note2] If Dec\_Scatter\_Gather is set, user can set JDOWFBS to 0xFFFFFFFF to let JPEG engine to ignore the Decode Output wait function.

#### 16.5.5.7 Decode Output wait service routine

1. Wait for the Decode Output interrupt (DOW\_INTS)
  - It represents the buffer is full.
  - User can get the data size that set to JDOWFBS from the address set to JYADDR0.
2. Clear the Decode Output interrupt status (DOW\_INTS)
3. Set Next Output Wait address & size
  - Address: JYADDR0
  - Set decoded output data size: JDOWFBS
    - ◆ Unit is Word.
    - ◆ Must multiple of MCU line data size
4. Trigger Decode Output Wait
  - Set Dec\_Scatter\_Gather (JITCR[18]) to 1
5. Go to Step 1 till getting the decode complete interrupt.



## 17 LCD Display Interface Controller (LCM)

### 17.1 Overview

The main purpose of Display Controller is used to display the video/image data to LCD device or connect with external TV-encoder. The video/image data source may come from the image sensor, JPEG decoder and the OSD pattern which have been stored in system memory (SDRAM). The input data format of the display controller can be packet YUV422, packet YUV444, packet RGB444, packet RGB565, packet RGB666, and packet RGB888. The OSD (On Screen Display) function supports packet YUV422 and 8/16/24-bit direct-color mode. The LCD controller supports both sync-type and MPU-type LCDM. This LCD Controller is a bus master and can transfer display data from system memory (SDRAM) without CPU intervention.

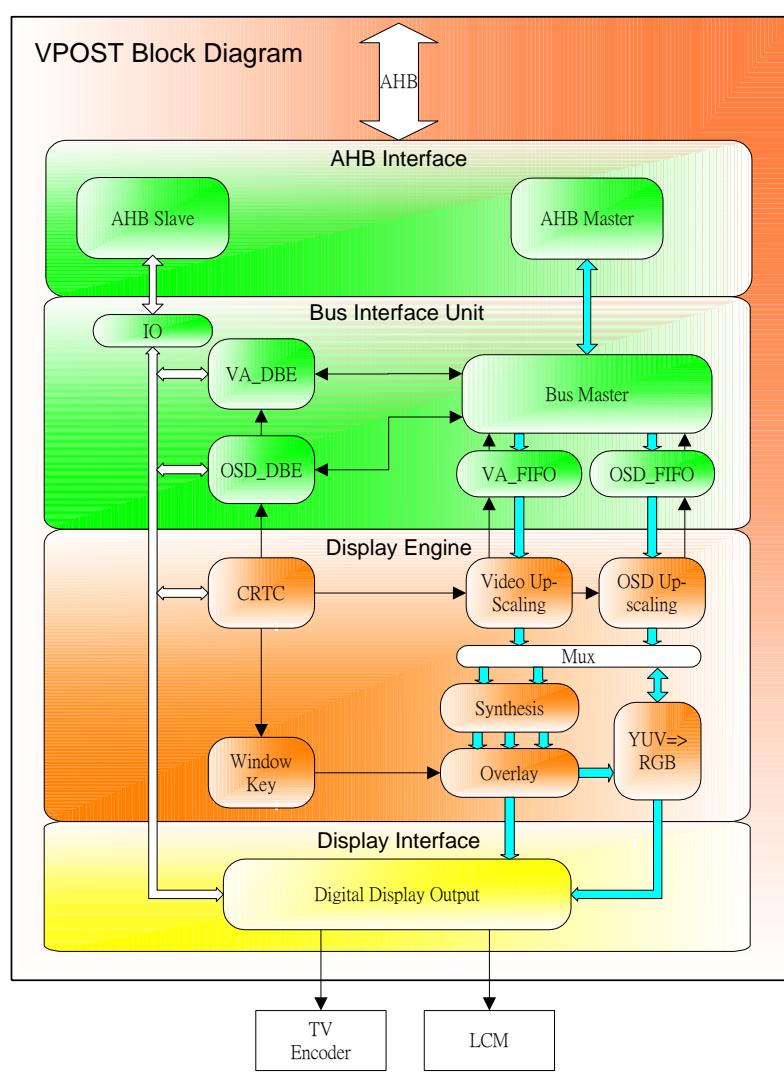
### 17.2 Features

- Input data format
  - YUV422, YUV444
  - RGB444, RGB565, RGB666, RGB888
- Output format
  - YUV422, YUV444
  - RGB444, RGB565, RGB666, RGB888
- Input size: Maximum size 1024 \* 768
- Image resize
  - Horizontal up-scaling 1~8X in fractional steps
  - Vertical up-scaling 1~8X in fractional steps
- Convert full range YUV to CCIR601
- Windowing support for three OSD graphic or text overlay
- Support CCIR-656 (with header), CCIR-601 (with hsync and vsync) 8/16-bit YUV data output format to connect with external TV encoder
- Support both sync-type and MPU-type LCM (with v-sync or not)
- Support the 8/9/16/18/24-bit data output to connect with 80/68 series MPU type LCM module

The LCD Controller includes the following main functions :

- Video post-processing
- Display & overlay control
- Video output control
- Hardware cursor control

## 17.3 Block Diagram



## 17.4 Register Map

Register	Offset	R/W	Description	Reset Value
<b>LCM Base Address:</b>				
<b>LCM_BA = 0xB000_8000</b>				
<b>DCCS</b>	LCM_BA + 0x00	R/W	Display Controller Control and Status Register	0x0000_0000
<b>DEVICE_CTRL</b>	LCM_BA + 0x04	R/W	Display Output Device Control Register	0x0000_00E0
<b>MPULCD_CMD</b>	LCM_BA + 0x08	R/W	MPU-Interface LCD Write Command Register	0x0000_0000
<b>INT_CS</b>	LCM_BA + 0x0C	R/W	Interrupt Control/Status Register	0x0000_0000
<b>CRTC_SIZE</b>	LCM_BA + 0x10	R/W	CRTC Display Size Register	0x0000_0000

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

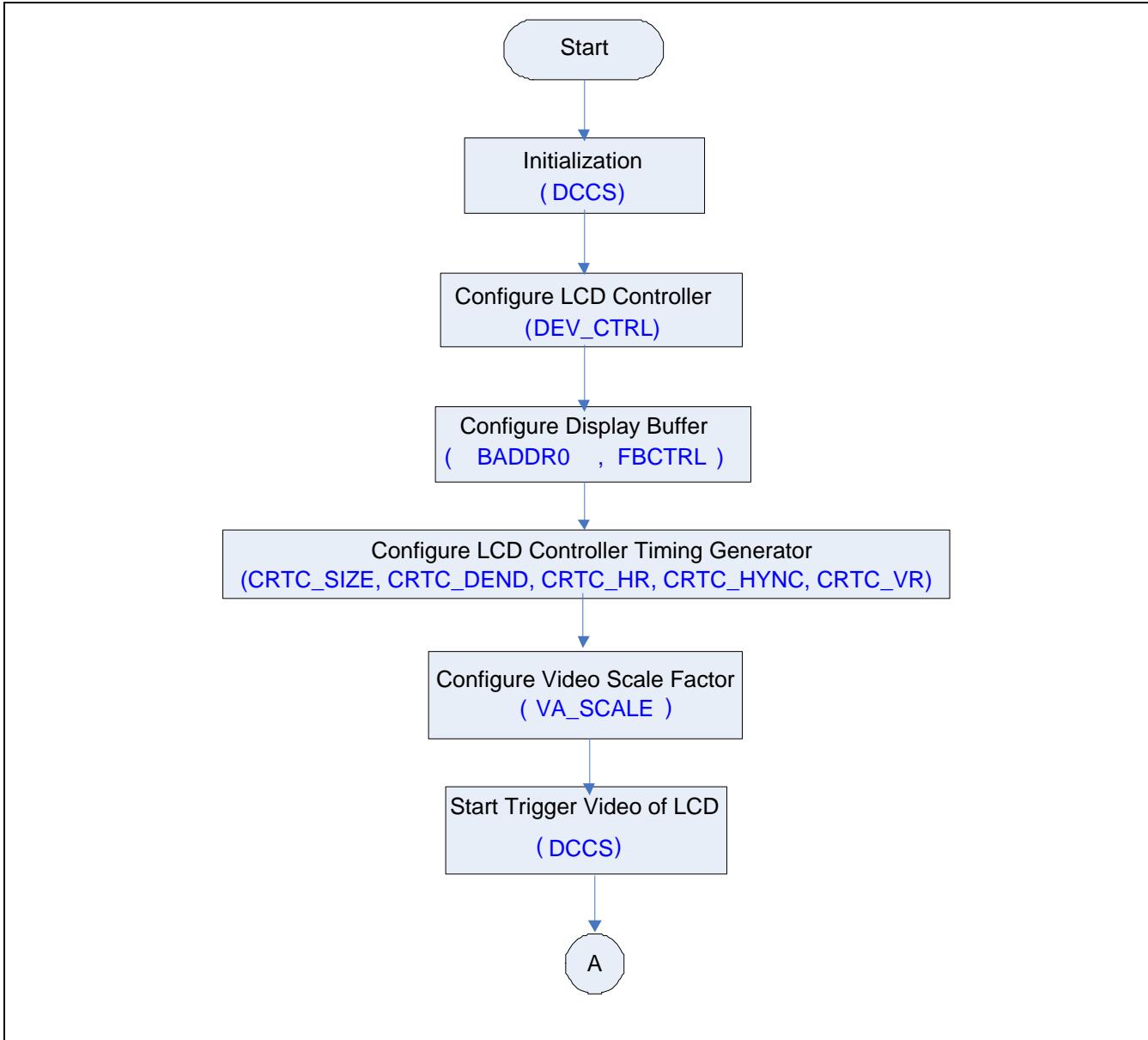
## 17.5 Functional Description

### 17.5.1 LCD Configuration Flow

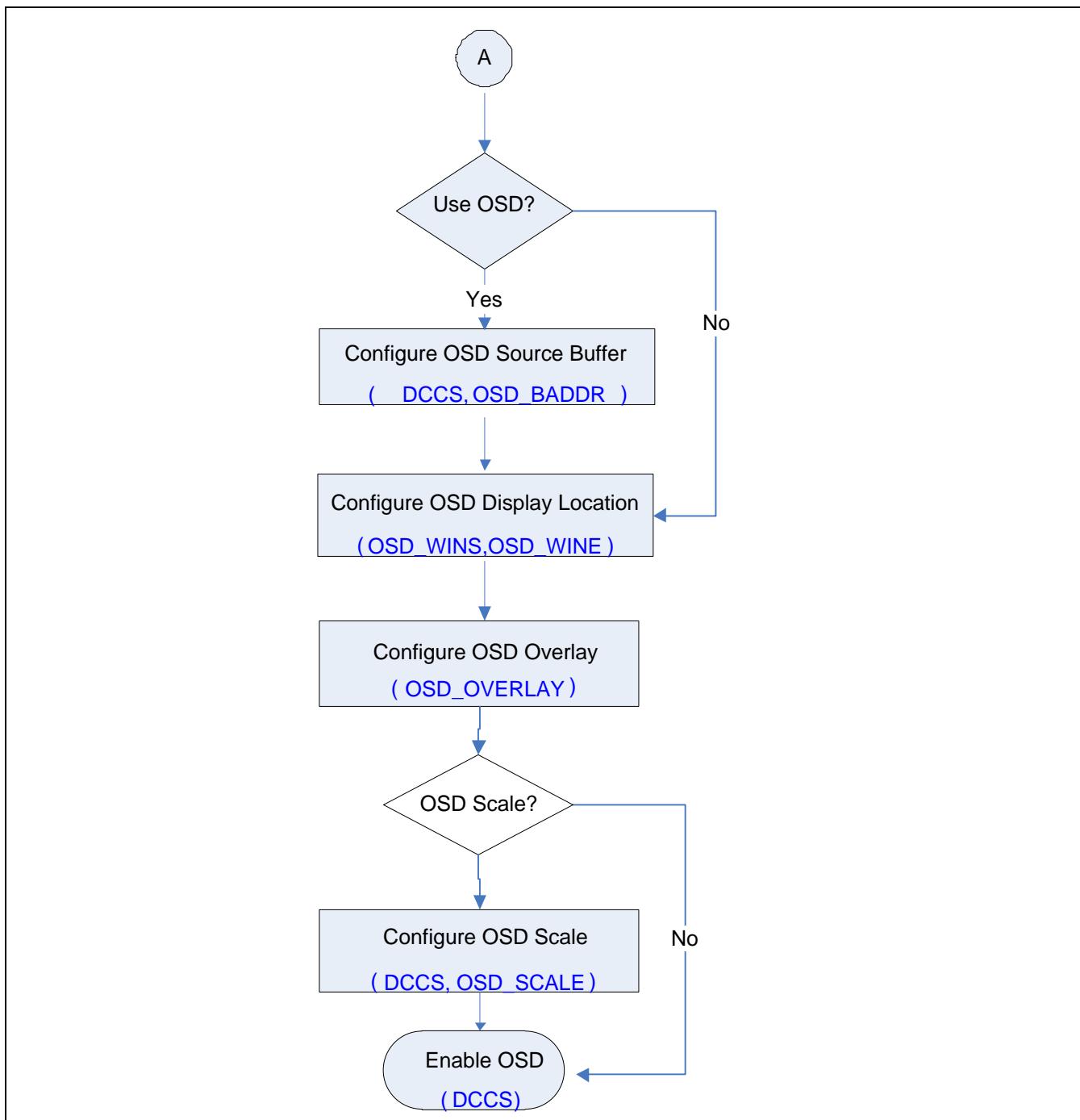
Software control flow will be introduced in this section. Please follow the steps described

below to avoid unpredicted situation.

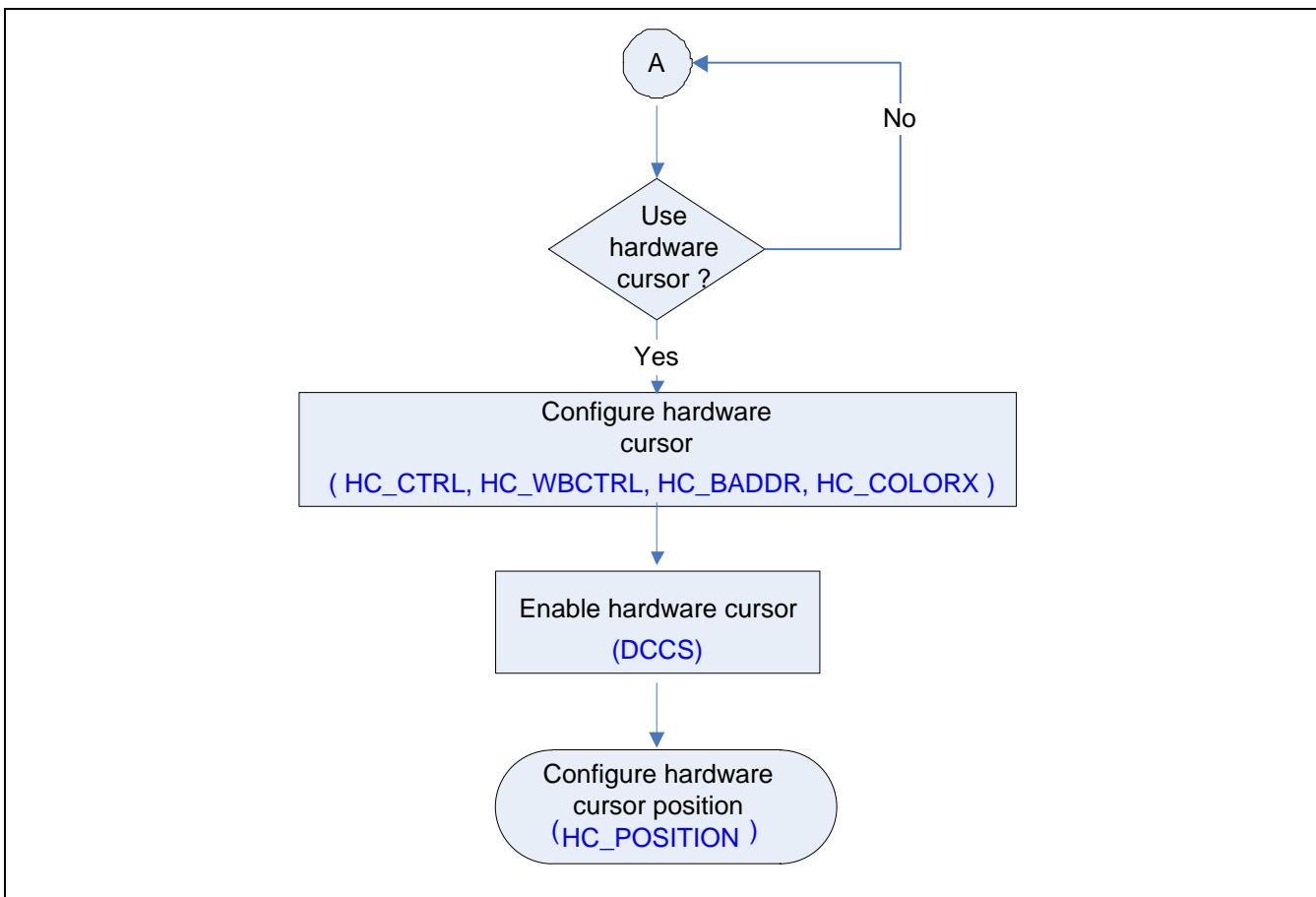
Configure Video:



Configure OSD:



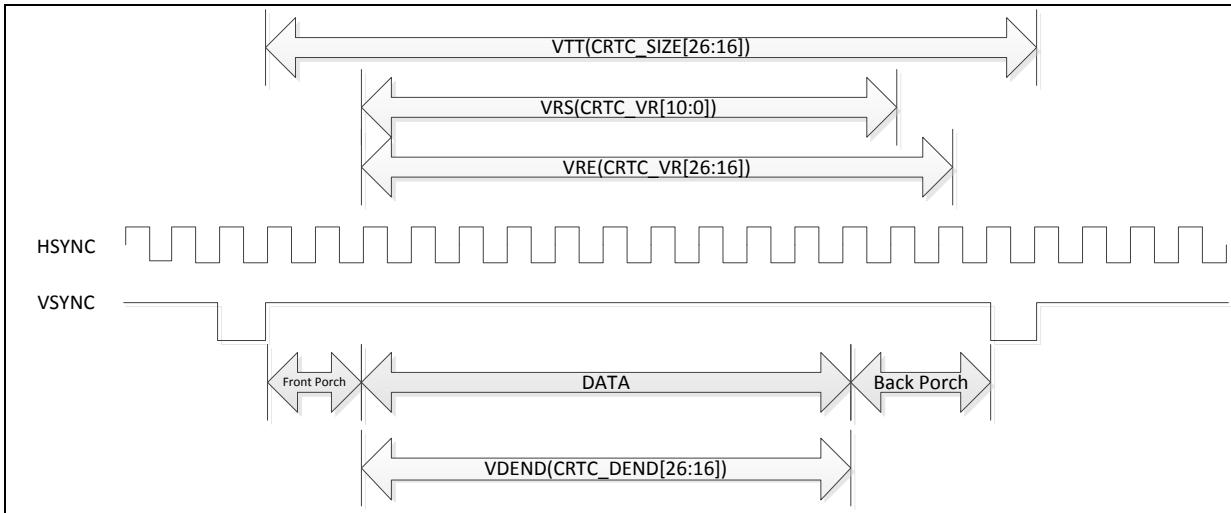
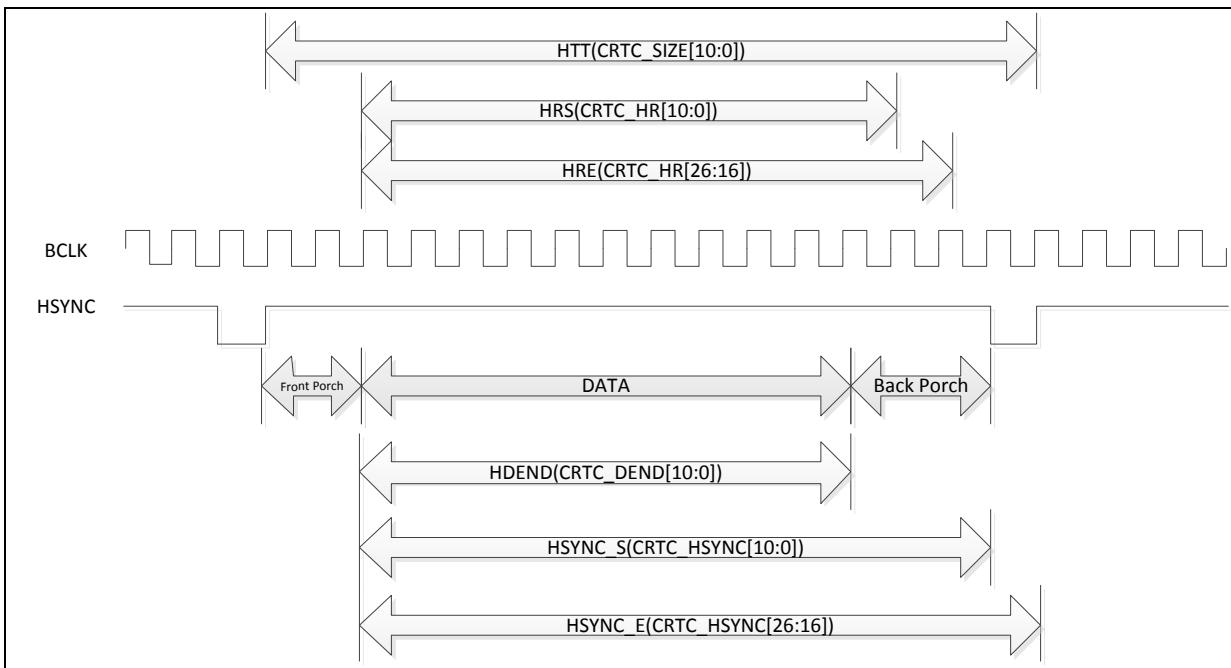
Configure hardware cursor:



### 17.5.2 LCD Controller Initialization and Configuration

Initialization and configuration flow:

1. Configure necessary global registers. (Ex. LCD clock register, multi-function pins and global registers.)
2. Reset LCD controller.
3. Allocate frame buffer according to display resolution - VA\_BADDR0 , VA\_FBCTRL.
4. Configure continuous display mode or single display mode according to application SINGLE(DCCS[7]).
5. Select input source format VA\_SRC(DCCS[10:8]).
6. Configure LCD related register according to type of LCM module (DEVICE\_CTRL).
7. Configure timing generator (CRTC\_SIZE, CRTC\_DEND, CRTC\_HR, CRTC\_HSYNC, CRTC\_VR). Vertical, horizontal timing maps to video display waveform illustrate as the following figure:



1. Configure video scale factor (VA\_SCALE).
2. Enable LCD controller VA\_EN(DCCS[1]) and DISP\_OUT\_EN(DCCS[3]).

The following is an example for 320\*240 display resolution,

```
//Configure LCD source clock and clock speed
APLLCON = 0xc0004018;
CLKDIV1 = (CLKDIV1 & ~0x1f) | 0x12;      //Use APLL and output 50MHz clock

//Switch LCD multi-function pins
MFP_GPG_L = (MFP_GPG_L & ~0xFF000000) | 0x22000000; //GPG6(CLK), GPG7(HSYNC)
```

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

//Configure LCD data pins - 16-bits
MFP_GPA_L = 0x22222222; //GPA0 ~ GPA7 (DATA0~7)
MFP_GPA_H = 0x22222222; //GPA8 ~ GPA15 (DATA8~15)

//Reset LCD
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; //Select RGB565 format

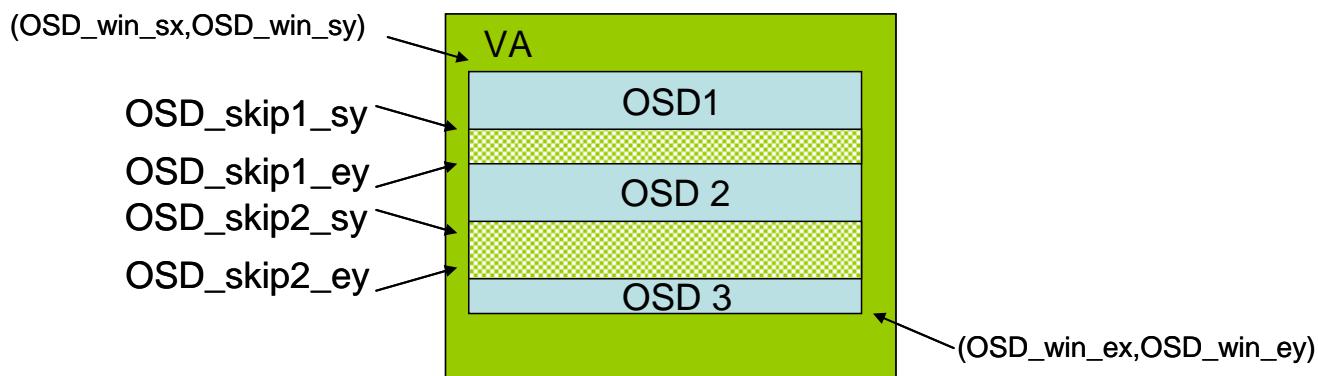
//Configure sync high color type and width of data bus is 16/18-bits
DEVICE_CTRL = 0;
DEVICE_CTRL |= (0x1 << 26) | (0x2 << 24) | (0x6 << 5) | (0x1 << 19);

//Configure timing registers
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; //Enable LCD
```

### 17.5.3 Configure OSD Controller

Follow the below steps to enable OSD and overlay on the video layer, illustrate as the following figure.



1. Configure OSD input source format OSD\_SRC(DCCS[14:12]).
2. Configure OSD source buffer address and frame buffer operation (OSD\_BADDR, OSD\_FBCTRL).
3. Configure OSD position.(OSD\_WIN\_S, OSD\_WIN\_E).
4. Enable color key (OSD\_OVERLAY[8]), configure display effect of overlay area (OSD\_OVERLAY[3 : 0]). Overlay display condition is list as following table.

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

Note: "Match" means OSD data matches with the value of color key.

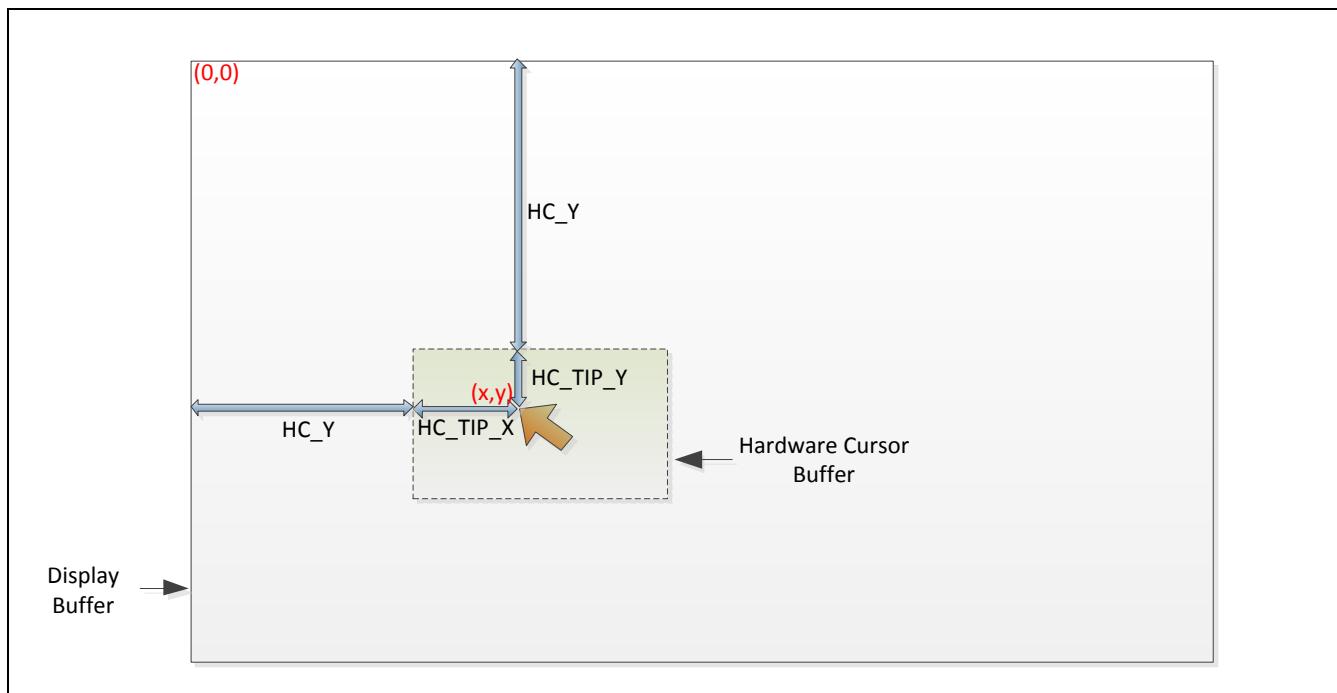
5. Enable blinking function by setting BLI\_ON(OSD\_OVERLAY[9]) bit and BLINK\_VCNT(OSD\_OVERLAY[23:16]) bit.

#### 17.5.4 Hardware Cursor

The following steps can enable hardware cursor function.

1. Configure cursor mode (1 - 4 color) and X/Y position. (HC\_CTRL)
2. Configure cursor buffer related registers. (HC\_WBCTRL , HC\_BADDR)
3. Modify hardware cursor position. (HC\_POS)

The following figure shows relationship between cursor position and display buffer.



## **18 MTP Controller (NUC970 only)**

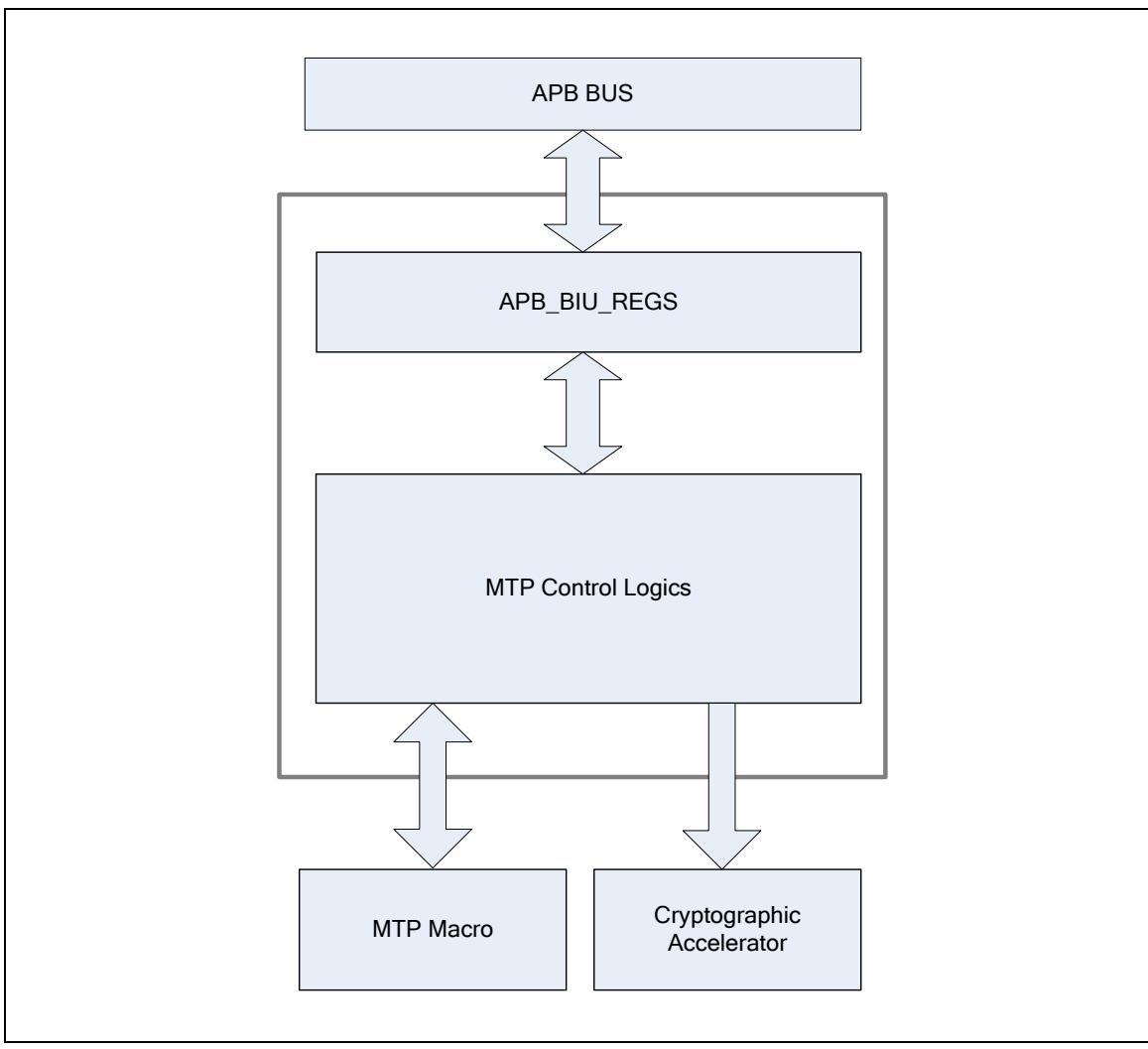
### **18.1 Overview**

The MTP (Multi-Time Programmable) controller performs an easy way to use and program the 256-bit Key for IP Security Engine. There is a MTP EEPROM in this chip, and it can be programmed 15 times. User can program 256-bit key and 8-bit user defined fields each time. The 256-bit key is program-only, and only can be used by IP Security Engine. User can use the 8-bit user defined field for special purpose. The MTP also supports a LOCK function to protect the content of programmed key and user defined field.

### **18.2 Features**

- Support MTP EEPROM programming
  - ◆ 256-bit Key and 8-bit user defined data field.
  - ◆ Up to 15 times programming.
- Support LOCK function.
- Support MTP key for AES encryption and decryption.
- Support MTP key for SHA comparison.

### **18.3 Block Diagram**



MTP Controller Block Diagram

## 18.4 Register Map

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

Register	Offset	R/W	Description	Reset Value
<b>MTP_BA = 0xB800C000</b>				
<b>MTP_KEYEN</b>	MTP_BA+0x00	R/W	Key Enable Register	0x0000_0000
<b>MTP_USERDATA</b>	MTP_BA+0x0c	R/W	MTP User Defined Data Register	0x0000_0000
<b>MTP_KEY0</b>	MTP_BA+0x10	W	MTP KEY 0 Register	0x0000_0000
<b>MTP_KEY1</b>	MTP_BA+0x14	W	MTP KEY 1 Register	0x0000_0000
<b>MTP_KEY2</b>	MTP_BA+0x18	W	MTP KEY 2 Register	0x0000_0000
<b>MTP_KEY3</b>	MTP_BA+0x1c	W	MTP KEY 3 Register	0x0000_0000

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

## 18.5 Functional Description

### 18.5.1 Use MTP Controller

MTP controller interface is connected to APB bus. User must enable MTPC(CLK\_PCLKEN1[26]) bit to operate MTP controller.

MTP registers are write-protected. Before starting operate MTP controller, user must unlock it first. By programming three continuous words 0x59, 0x16, and 0x88 to MTP\_REGLCTL register can unlock MTP registers. At any time, program 0x1 to REGLCTL[0](MTP\_REGLCTL[0]) can re-enable the write-protection of MTP registers.

### 18.5.2 MTP Key

NUC970 provides a 256 bits width MTP key function. Following the MTP programming flow, user can program a MTP key into the MTP controller internal EEPROM. This EEPROM supports up-to 15 times programming opportunities.

Each time to program a new MTP key will also invalidate the existing MTP key. Only the last programmed MTP key is effective. That is, the new MTP key always overwrites the old MTP key. At any time, NUC970 system can have only one MTP key or no MTP. After executing the MTP key enable flow, user can learn the remaining times of program opportunity by reading PRGCNT(MTP\_STATUS[19:16]). The available time is (15 – PRGCNT). If PRGCNT is 15, it means this NUC970 chip's MTP has been programmed up to 15 times and cannot be programmed any more.

After executing the MTP enable flow, user can learn MTP key status via reading MTP control registers. If the NUC970 chip has never been programmed with any MTP key, user can observe both MTPEN(MTP\_STATUS[0]) bit and NONPRG(MTP\_STATUS[2]) are 1. At this time, user can execute MTP key program flow to program a new MTP key.

After executing the MTP enable flow, if both MTPEN(MTP\_STATUS[0]) bit and KEYVALID(MTP\_STATUS[1]) bit are 1, there's exist a valid MTP key. At this time, user can employ the MTP key for AES encrypt/decrypt key or SHA comparison. If LOCKED(MTP\_STATUS[3]) bit is 0 and PRGCNT(MTP\_STATUS[19:16]) is smaller than 15, user is allowed to program a new MTP key.

### 18.5.3 User Defined Data

In addition to the 256 bits size MTP key itself, MTP controller also provides User Defined Data size which size is 8 bits. User Defined Data is used to identify the purpose of MTP key. When programming MTP key, User Defined Data will be programmed at the same time. On executing MTP key program flow, in addition to program a 256 bits key to MTP\_KEY0 ~ MTP\_KEY7 registers, user must also program an 8 bits User Defined Data to MTP\_USERDATA register.

Users can determine the significance of this User Defined Data represented themselves. By interpreting User Defined Data, MTP key user can learn the purpose of MTP key and determine it can use the MTP key or not.

However, NUC970 IBR(Internal Boot ROM) program reads User Defined Data to determine whether to execute AES decrypt or SHA comparison on loading the program code in NAND/SPI/eMMC. To prevent from IBR mislead by User Defined Data, user must not to use those bits meaningful to IBR.

IBR interprets only bit[2:0] of User Defined Data. IBR defines User Defined Data as the following table:

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 uses MTP key as AES decrypt key. IBR will execute AES decrypt operation on program code in boot flash and load the decrypted code to system memory for running.
x	x	x	x	x	1	1	0		IBR will execute SHA calculation on program code in boot flash. Then, IBR compares the SHA calculation result with MTP key. If key matched, IBR will load the program code to system memory for running. If not matched, IBR will halt there.
x	x	x	x	x	0	x	x		IBR ignores the MTP key.

X : don't care

Supposed user set the bit 2 of User Defined Data as 1, IBR will think of the MTP key is used for boot code AES decryption or boot code SHA validation. This will result in IBR false decrypt boot code or halt on boot code validation error. So, if user wants to define his own User Defined Data, always set bit 2 as 0 can safely passed IBR.

#### 18.5.4 MTP Enable

To perform any operations on MTP key, MTP key must be enabled first. To employ MTP key for AES encrypt/decrypt or SHA comparison, MTP key must also be enabled first. The MTP enable flow is listed in the flowing steps:

1. Set MTPC(CLK\_PCLKEN1[26]) bit as 1 to enable MTP controller clock.
2. Program three contiguous words 0x59, 0x16, and 0x88 to MTP\_REGLCTL. Confirm REGLCTL (MTP\_REGLCTL [0]) to 1, it indicates that you can write MTP register.
3. Writing 1 to KEYEN(MTP\_KEYEN[0]) bit.
4. Check MTPEN(MTP\_STATUS [0]) bit until it is set to 1.
5. If NONPRG (MTP\_STATUS [2]) is 1, indicating that this chip's MTP has not yet been programmed. MTP has been enabled successfully now. It's allowed to program a MTP key. Please note that due to MTP has never been programmed yet, if use MTP key to perform AES decryption or SHA comparison at this time, there will be unexpected errors.
6. If KEYVALID (MTP\_STATUS [2]) is 1, indicating that MTP is enabled. You can use MTP key to perform AES decryption or SHA comparison. If LOCKED (MTP\_STATUS [3]) is 0, then the user can also program a new MTP key or lock MTP key at this time. If LOCKED (MTP\_STATUS [3]) is 1, it means the chip MTP key is locked, no longer allowed to perform MTP key programming or locked procedure.

#### 18.5.5 Program MTP Key

Before programming a MTP key, the MTP key must have been enabled. The MTP enable procedure is described in section [18.5.4](#). Once MTP enable procedure success, check MTP\_STATUS register. You cannot program a MTP Key in the following conditions:

- LOCKED(MTP\_STATUS[3]) bit is 1: Indicates chip MTP key is locked, no longer allowed to perform MTP key programming.
- PRGCNT(MTP\_STATUS[19:16]) is equal to 15: Indicated the MTP programming times has reach up-limit. No longer allowed to perform MTP key programming.

In addition to above two cases, once MTP is enabled, the user can perform key programming procedure for MT. The programming procedure is as follows:

1. Write 0x2 to MODE(MTP\_CTL[1:0]).
2. Based on the current frequency of PCLK, write a PCLK clock count to MTP\_PCYCLE register. The corresponding time of this PCLK clock count must be longer than 330us.

For example, if PCLK frequency is 75 MHz, the time per clock will be 13.333 ns. We can come out the clock count of 330us is 24750 by “ $330\text{us} / 13.333\text{ns} = 330000\text{ns} / 13.333\text{ns} = 24750$ ”. In this case, the user must write a clock count larger than 24750 to MTP\_PCYCLE register to meet the MTP key programming time requirement.

3. Program the 256 bits MTP key to MTP\_KEY0 ~ MTP\_KEY7. If this MTP key is used for AES key, there's a key rotation rule must be noted. The rotation rule of AES purpose MTP key is determined by the current boot mode (boot from NAND, SPI, or eMMC). Refer to section [18.5.7](#) for detailed descriptions.
4. Write the User Defined Data to MTP\_USERDAT. (refer to section [19.5.3](#))
5. Write 0x1 to PSTART(MTP\_PSTART[0]) and then check PSTART(MTP\_PSTART[0]) until it was cleared as 0 by MTP controller.
6. Check PRGFAIL(MTP\_STATUS[4]) bit. This bit 0 represents MTP key be successfully programmed. A none-zero bit indicates MTP key programming failed.

### 18.5.6 Lock MTP Key

After programmed the MTP key, if it's confirmed this key will never be changed, the user can consider of locking this MTP key. Locking MTP key is for security purposes. Once the MTP key is locked, MTP controller will not allow future MTP key programming operations on this chip. This mechanism is to prevent MTP key be overwritten occasionally or be hacked.

Consider about MTP can be programmed 15 times, if MTP key is not locked, then the attacker will likely overwrite this MTP key by programming a new MTP key. Therefore, for safety sake, it's best to lock MTP key.

Import attention should be noted. Once the MTP key is successfully programmed, there's no way to unlock the MTP key. Even the chip producer cannot unlock it.

Before perform MTP key lock procedure, MTP key must be enabled first. Refer to section [19.5.3](#) for MTP key enable procedure. MTP locked operating procedures are as follows:

1. Write 0x3 to MODE(MTP\_CTL[1:0]).
2. Based on the current frequency of PCLK, write a PCLK clock count to MTP\_PCYCLE register. The corresponding time of this PCLK clock count must be longer than 330us. For example, if PCLK frequency is 75 MHz, the time per clock will be 13.333 ns. We can come out the clock count of 330us is 24750 by “ $330\text{us} / 13.333\text{ns} = 330000\text{ns} / 13.333\text{ns} = 24750$ ”. In this case, the user must write a clock count larger than 24750 to MTP\_PCYCLE register to meet the MTP key programming time requirement.
3. Write 0x1 to PSTART(MTP\_PSTART[0]) and then check PSTART(MTP\_PSTART[0]) until it was cleared as 0 by MTP controller.

4. Check if LOCKED(MTP\_STATUS[3]), KEYVALID(MTP\_STATUS[1]), and MTPEN(MTP\_STATUS[0]) are all set as 1, which means MTP key is locked successfully.

### 18.5.7 MTP Key for AES Encrypt/Decrypt

MTP key can be read by Cryptographic Accelerator hardware, and can be used as AES encrypt/decrypt keys. Before using MTP key for AES encrypt/decrypt, the user must enable MTP key first. Cryptographic Accelerator hardware will not get correct MTP key if MTP key is not enabled.

If user wants to use MTP key for AES encrypt/decrypt, just set EXTKEY(CRPT\_AES\_CTL[4]) as 1 on performing AES encrypt/decrypt operation. If EXTKEY (CRPT\_AES\_CTL [4]) is 1, the AES accelerator will retrieve AES encrypt/decrypt keys from MTP key, instead of from CRPT\_AESn\_KEYx registers.

Only AES or SHA / HMAC hardware accelerator is allowed to retrieve MTP key. There's no any other ways to retrieve MTP key. This is for MTP key security purpose.

Special attention is must, AES accelerator does not retrieve MTP key in sequential. When retrieving MTP key, AES accelerator rotates the MTP key depending on the current boot mode. The key retrieving rule is listed in the following table:

AES Key	Boot from NAND	Boot from SPI	Boot from eMMC	Boot from USB
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

This key rotation rule is for security purpose. Suppose the user wants to publish a firmware and makes it be booted from NAND flash with AES encrypted. The user must first encrypt this firmware with AES and program the encrypted firmware to NAND flash. The user then program the AES key to MTP key in order KEY4, KEY5, KEY6, KEY7, KEY0, KEY1, KEY2, KEY3. Thus, when booting from NAND flash, NUC970 IBR(Internal Boot ROM) can successfully decrypt the firmware image and load to system memory. Should the encrypted firmware be programmed to NOR flash and booting from it, IBR will decrypt out a wrong image due to incorrect AES key.

### 18.5.8 MTP Key for SHA/HMAC Comparison

MTP key can be read by SHA/HMAC accelerator hardware for SHA/HMAC calculation output comparison. Using MTP key for SHA/HMAC comparison, the user must first enable MTP key. If MTP key is not enabled, SHA/HMAC accelerator will get wrong MTP key and will get comparison failed.

To use MTP key SHA/HMAC comparison, the user just set CMPEN(CRPT\_HMAC\_CTL[15]) as 1 on performing SHA/HMAC calculation. If CMPEN(CRPT\_HMAC\_CTL[15]) is 1, after SHA/HMAC calculation done, SHA/HMAC accelerator will automatically retrieve MTP key and compare it with SHA/HMAC output digest. If MTP key is matched with SHA/HMAC output digest, SHA/HMAC accelerator sets CMPSTS(CRTP\_HMAC\_STS[15]) as 1. Otherwise, it clears CMPSTS(CRTP\_HMAC\_STS[15]) as 0.

Different from “MTP key for AES decrypt”, in case of “MTP key for AES comparison”, SHA/HMAC accelerator does not rotate MTP key by different boot mode. It always compares MTP key with SHA/HMAC output digest in sequential.

## 19 Pulse Width Modulation (PWM)

### 19.1 Overview

This chip has one PWM controller, and it has 4 independent PWM outputs, CH0~CH3, or as 2 complementary PWM pairs, (CH0, CH1), (CH2, CH3) with 2 programmable dead-zone generators. Each PWM pair has one Prescaler, one clock divider, two clock selectors, two 16-bit PWM counters, two 16-bit comparators, and one Dead-Zone generator. They are all driven by APB system clock (PCLK) in chip. Each PWM channel can be used as a timer and issue interrupt independently.

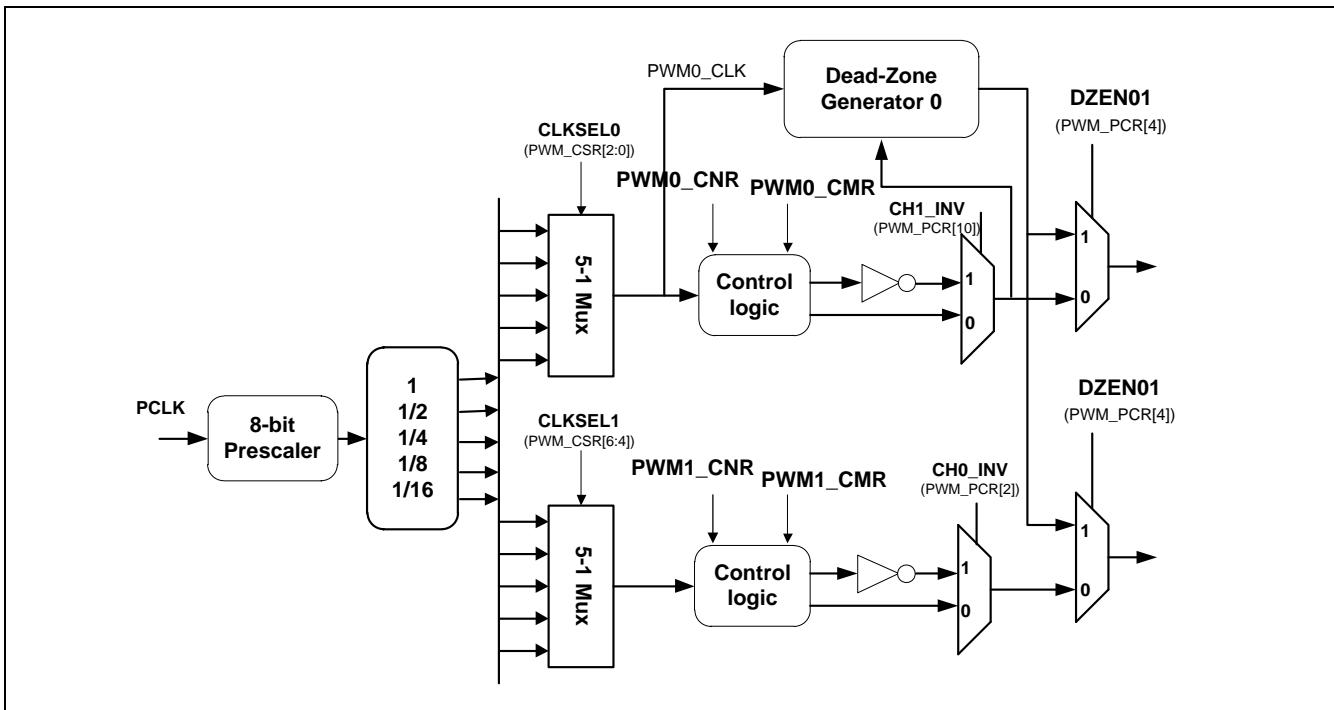
Two channels PWM Timers in one pair share the same prescaler. The Clock divider provides each PWM channel with 5 divided clock sources (1, 1/2, 1/4, 1/8, 1/16). Each channel receives its own clock signal from clock divider which receives clock from 8-bit prescaler. The 16-bit down-counter in each channel receive clock signal from clock selector and can be used to handle one PWM period. The 16-bit comparator compares PWM counter value with threshold value in register CMR (PWM\_CMR[15:0]) loaded previously to generate PWM duty cycle. The clock signal from clock divider is called PWM clock. Dead-Zone generator utilize PWM clock as clock source. Once Dead-Zone generator is enabled, two outputs of the corresponding PWM channel pair will be replaced by the output of Dead Zone generator. The Dead-Zone generator is used to control off-chip power device.

To prevent PWM driving output pin with unsteady waveform, 16-bit down-counter and 16-bit comparator are implemented with double buffering feature. User can feel free to write data to counter buffer register and comparator buffer register without generating glitch. When 16-bit down-counter reaches zero, the interrupt request is generated to inform CPU that time is up. When counter reaches zero, if counter is set as periodic mode, it is reloaded automatically and start to generate next cycle. User can set PWM counter as one-shot mode instead of periodic mode. If counter is set as one-shot mode, counter will stop and generate one interrupt request when it reaches zero. The value of comparator is used for pulse width modulation. The counter control logic changes the output level when down-counter value matches the value of compare register.

### 19.2 Features

- 4 PWM channels with a 16-bit down counter and an interrupt each.
- 2 complementary PWM pairs, (CH0, CH1), (CH2, CH3), with a programmable dead-zone generator each.
- Internal 8-bit prescaler and a clock divider for each PWM paired channel.
- Independent clock source selection for each PWM channel.
- Internal 16-bit down counter and 16-bit comparator for each independent PWM channel.
- PWM down-counter supports One-shot or Periodic mode.

### 19.3 Block Diagram



## 19.4 Register Map

Register	Offset	R/W	Description	Reset Value
<b>PWM Base Address:</b>				
<b>PWM_BA = 0xB800_7000</b>				
<b>PWM_PPR</b>	PWM_BA+0x000	R/W	PWM Pre-scale Register	0000_0000
<b>PWM_CSR</b>	PWM_BA+0x004	R/W	PWM Clock Select Register	0000_0000
<b>PWM_PCR</b>	PWM_BA+0x008	R/W	PWM Control Register	0000_0000
<b>PWM0_CNR</b>	PWM_BA+0x00C	R/W	PWM Counter Register 0	0000_0000
<b>PWM0_CMRR</b>	PWM_BA+0x010	R/W	PWM Comparator Register 0	0000_0000
<b>PWM0_PDR</b>	PWM_BA+0x014	R	PWM Data Register 0	0000_0000
<b>PWM1_CNR</b>	PWM_BA+0x018	R/W	PWM Counter Register 1	0000_0000
<b>PWM1_CMRR</b>	PWM_BA+0x01C	R/W	PWM Comparator Register 1	0000_0000
<b>PWM1_PDR</b>	PWM_BA+0x020	R	PWM Data Register 1	0000_0000
<b>PWM2_CNR</b>	PWM_BA+0x024	R/W	PWM Counter Register 2	0000_0000
<b>PWM2_CMRR</b>	PWM_BA+0x028	R/W	PWM Comparator Register 2	0000_0000
<b>PWM2_PDR</b>	PWM_BA+0x02C	R	PWM Data Register 2	0000_0000
<b>PWM3_CNR</b>	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

## 19.5 Functional Description

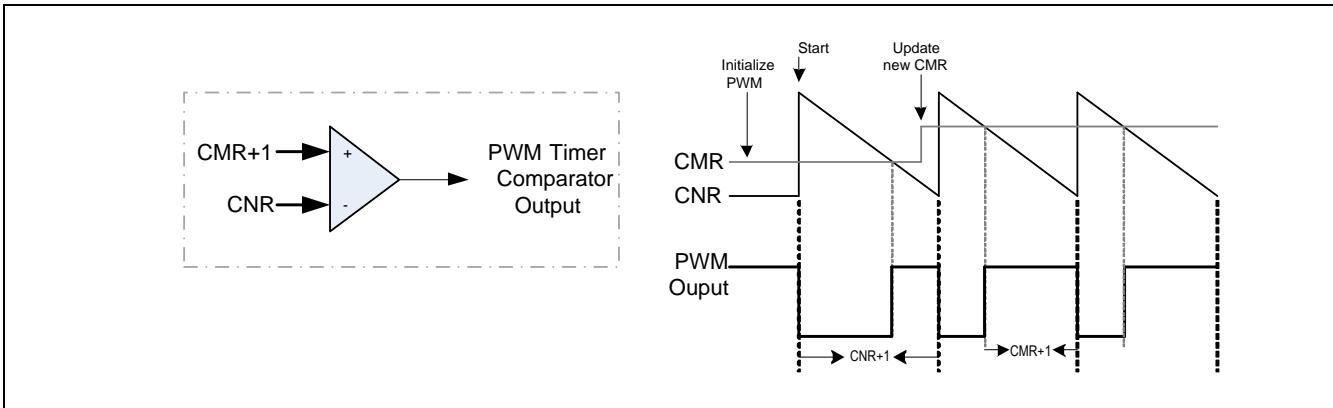
### 19.5.1 PWM Timer Operation

The PWM period and duty control are decided by register PWMx\_CNR and PWMx\_CMRR registers.. The PWM-timer timing operation is shown in following figure. The pulse width modulation follows the formula below:

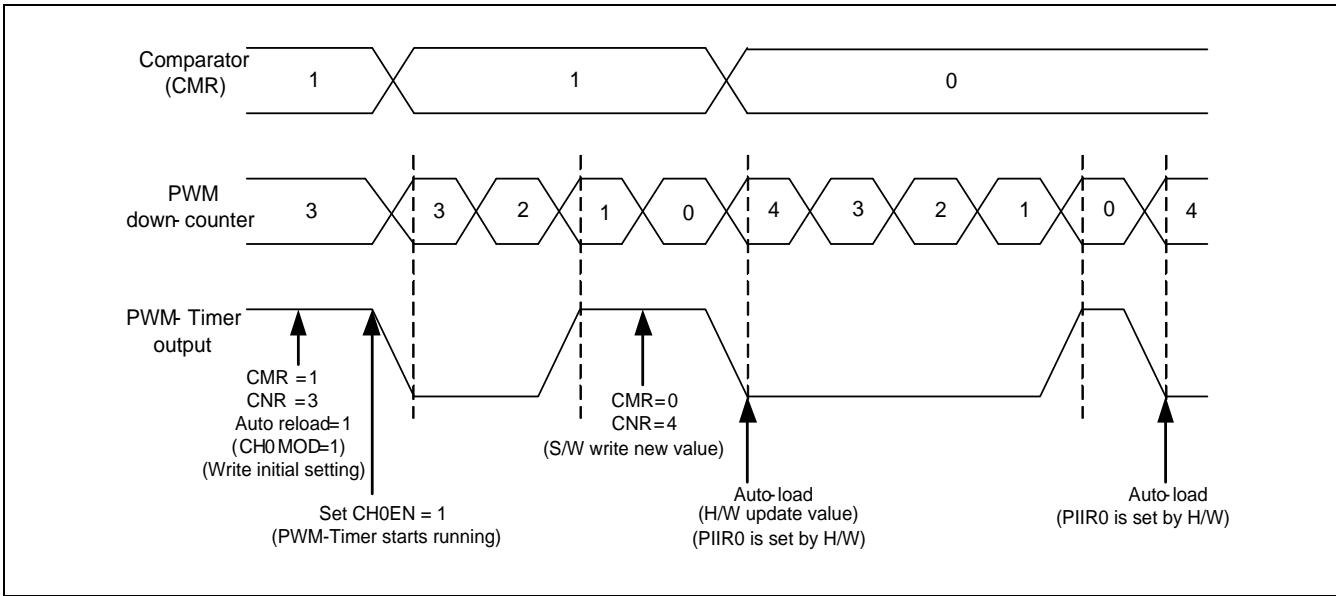
$$\text{PWM frequency} = \text{PWM\_CLK} / ((\text{prescale} + 1) * (\text{clock divider})) / (\text{CNR} + 1)$$

$$\text{PWM duty ratio} = (\text{CMR} + 1) / (\text{CNR} + 1)$$

When CMR >= CNR: PWM output is always high. When CMR < CNR: PWM outputs low for (CNR - CMR) PWM clocks, and PWM outputs high for (CMR + 1) PWM clocks. If CMR = 0, then PWM output low for CNR PWM clocks and output high for 1 PWM clock..

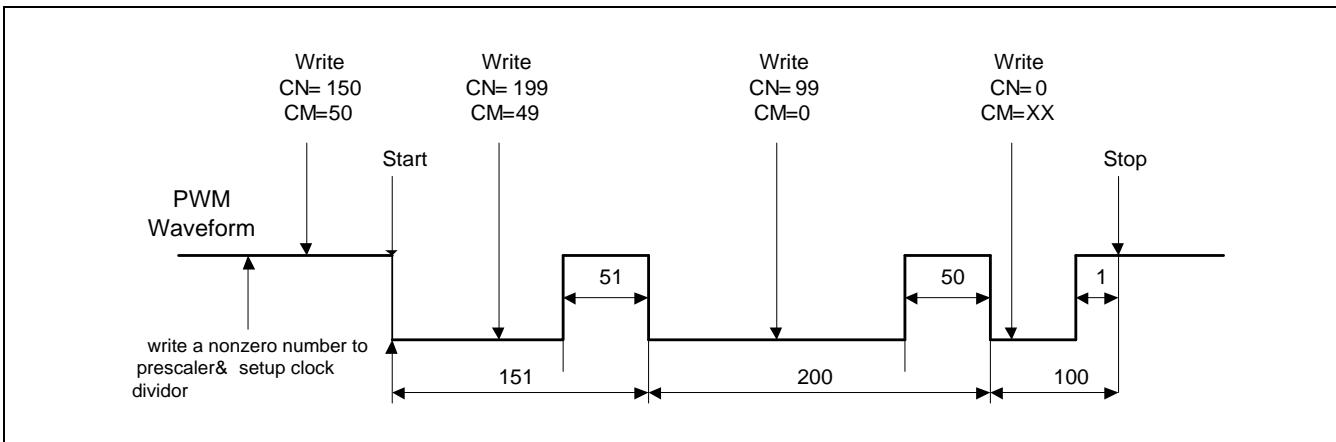


Following waveform illustrate the operation of PWM. Whenever the current counter equals to compare register or reaches 0, output level toggles.

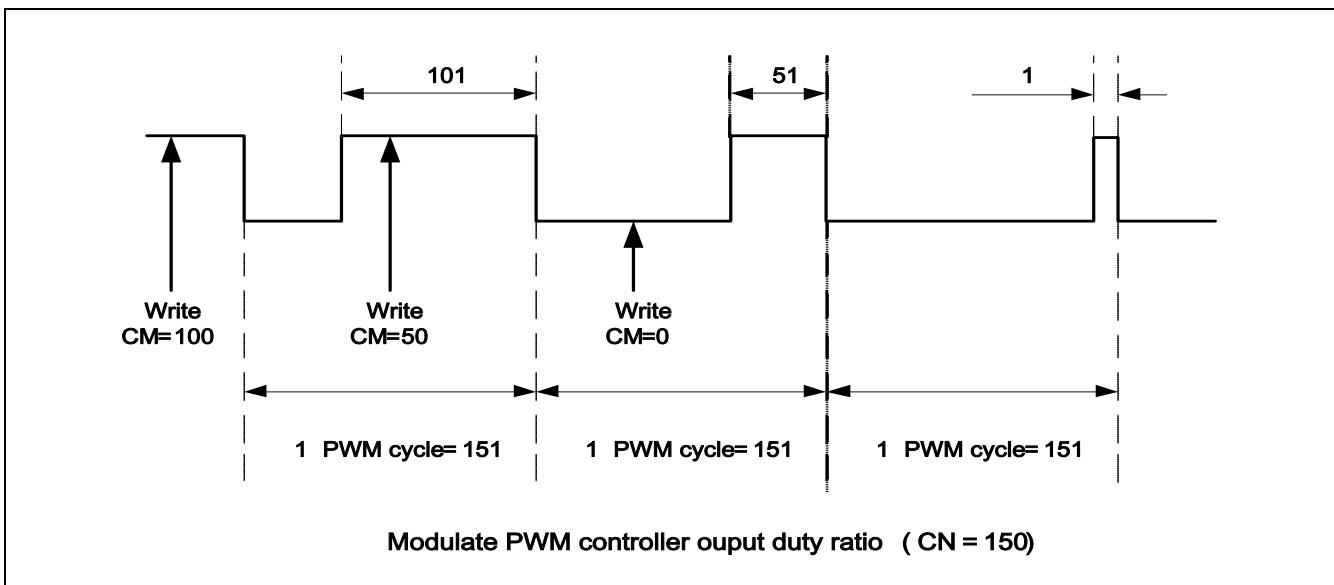


### 19.5.2 PWM double buffer

The PWM timers have double buffering function; the reload value is updated at the start of next period without affecting current timer operation. The PWM counter value can be written into CNR (PWM\_CNR[15:0]).



Following figure is an example of using double buffer feature.



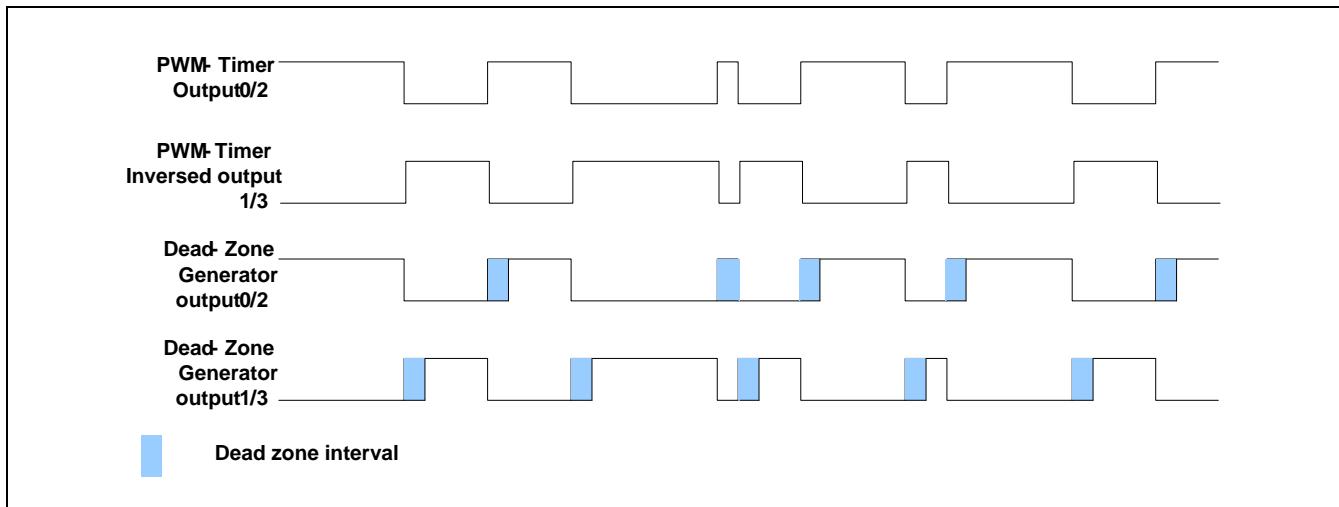
### 19.5.3 Periodic and One-Shot Operation

The CHxMOD bits defines PWM operation in Periodic or One-shot mode. If CHxMOD is set to one (periodic mode), the controller loads CNR (PWM\_CNR[15:0]) to PWM counter when PWM counter reaches zero. If CNR is set to zero, PWM counter will be halt when PWM counter counts to zero.

In one-shot mode (CHxMOD=0), the corresponding channel will output only one cycle of duty waveform and then PWM counter will be stopped if no further corresponding duty register updated. When PWM counter is running, updating corresponding duty register will engage the next

### 19.5.4 Dead-Zone Generator

PWM implements Dead Zone generator. They are built for power device protection. This function generates a programmable time gap called “Dead-Zone” to delay PWM rising output, and it is in order to prevent damage for the power switch devices that connected to the PWM output pins. User can program Dead-Zone counter to determine the Dead Zone interval. Following figure shows Dead-Zone operation.



### 19.5.5 PWM Timer Start Procedure

Take PWM channel 0 for example, and the following procedure is for starting a PWM.

1. Setup clock selector CLKSEL0 (PWM\_CSR[2:0]).
2. Setup prescaler PRESCALE (PWM\_PPR[7:0]).
3. Setup inverter on/off CH0INV (PWM\_PCR[2]).
4. Setup dead zone generator on/off DZEN01 (PWM\_PCR[4]), also set dead-zone interval in DZL01 (PWM\_PPR[23:16]) if dead-zone enabled.
5. Setup CH0MOD (PWM\_PCR[3]) to select operation mode.
6. Setup interrupt enable register PIER0 (PWM\_PIER[0])
7. Setup the corresponding GPIO pins to PWM function
8. Enable PWM down-counter by set CH0EN((PWM\_PCR[0])) to 1.
9. Setup PWM comparator register CMR (PWM\_CMRA[15:0]) and PWM counter register CNR (PWM\_CNR[15:0]) for setting PWM period and duty length

Step1~8 may be execute in other order without affect the behavior of PWM timer. Below is a sample setting PWM0 frequency to 1000Hz and 40% duty ratio.

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

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

```
PWM->CNR = (1000 - 1);
```

### 19.5.6 PWM Timer Stop Procedure

There are two methods to stop PWM timer, here using channel 0 as example.

Method 1:

Set 16-bit down counter CNR (PWM\_CNR[15:0]) as 0. When interrupt request happen or polling interrupt bit PIIR0(PWM\_PIIR[0]) until set 1, disable PWM Timer by setting CH0EN = 0 (PWM\_PCR[0] = 0). (Recommended).

Method 2:

Disable PWM Timer by setting CH0EN = 0 (Not recommended)

Method 2 is not recommended because clear CH0EN will stop PWM output immediately and cause a abruptly change in PWM duty ration. This may damage the motor connected with PWM.

## 20 Real Time Clock (RTC)

### 20.1 Overview

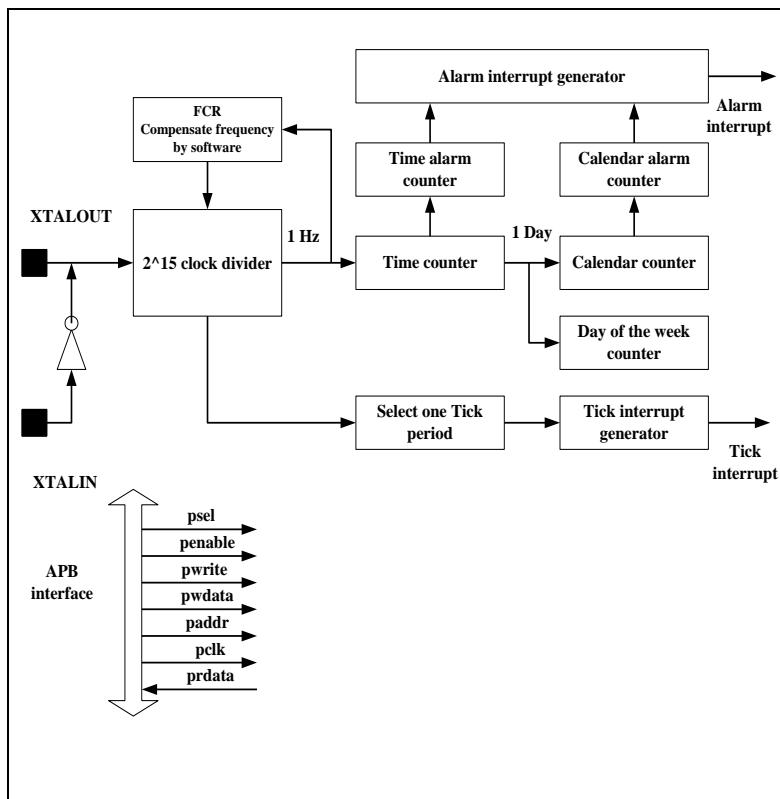
Real Time Clock (RTC) block can operate with independent power supply (RTC\_VDD) while the system power is off. The clock source of RTC controller is from an external 32.768 kHz low-speed crystal. The RTC controller provides the real time clock and calendar information. The data format of RTC time and calendar message are all expressed in BCD (Binary Coded Decimal) format. The RTC also provide frequency compensation mechanism for 32.768 kHz clock source

### 20.2 Features

- Supports real time counter and calendar counter for RTC time and calendar check.
- Supports time (hour, minute, second) and calendar (year, month, day) alarm and alarm mask settings
- Selectable 12-hour or 24-hour time scale
- Supports Leap Year indication
- Supports Day of the Week counter
- Supports frequency compensation mechanism for 32.768 kHz clock source
- All time and calendar message expressed in BCD format
- Supports periodic RTC Time Tick interrupt with 8 period interval options 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2 and 1 second
- Supports RTC Time Tick and Alarm match interrupt
- Supports chip wake-up from Idle or Power-down mode while alarm or relative alarm interrupt is generated
- Supports 64 bytes spare registers to store user's important information
- Supports power on/off control mechanism to control system core power

### 20.3 Block Diagram

The block diagram of Real Time Clock depicted is as following:



## 20.4 Register Map

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
<b>RTC_BA = 0xB800_4000</b>				
RTC_INIT	RTC_BA+0x000	R/W	RTC Initiation Register	Undefined
RTC_RWEN	RTC_BA+0x004	R/W	RTC Access Enable Register	0x0000_0000
RTC_FREQADJ	RTC_BA+0x008	R/W	RTC Frequency Compensation Register	0x0000_0700
RTC_TIME	RTC_BA+0x00C	R/W	RTC Time Counter Register	0x0000_0000
RTC_CAL	RTC_BA+0x010	R/W	RTC Calendar Counter Register	0x0005_0101
RTC_TIMEFMT	RTC_BA+0x014	R/W	RTC Time Format Selection Register	0x0000_0001
RTC_WEEKDAY	RTC_BA+0x018	R/W	RTC Day of the Week Register	0x0000_0006
RTC_TALM	RTC_BA+0x01C	R/W	RTC Time Alarm Register	0x0000_0000
RTC_CALM	RTC_BA+0x020	R/W	RTC Calendar Alarm Register	0x0000_0000
RTC_LEAPYEAR	RTC_BA+0x024	R	RTC Leap year Indicator Register	0x0000_0000
RTC_INTEN	RTC_BA+0x028	R/W	RTC Interrupt Enable Register	0x0000_0000
RTC_INTSTS	RTC_BA+0x02C	R/C	RTC Interrupt Status Register	0x0000_0000

<b>RTC_TICK</b>	RTC_BA+0x030	R/W	RTC Time Tick Register	0x0000_0000
<b>RTC_PWRCTL</b>	RTC_BA+0x034	R/W	RTC Power Control Register	0x0000_7000
<b>RTC_PWCNT</b>	RTC_BA+0x038	R	RTC Power Control Counter Register	0x0000_0000
<b>RTC_SPR0 ~</b> <b>RTC_SPR15</b>	RTC_BA+0x040 ~ RTC_BA+0x07C	R/W	RTC Spare Register 0 ~ 15	0x0000_0000

## 20.5 Functional Description

### 20.5.1 RTC Initiation

When RTC block is power on, programmer has to write a number (0xa5eb1357) to RTC\_INIT to reset all logic. RTC\_INIT act as hardware reset circuit. Once RTC\_INIT has been set as 0xa5eb1357, user cannot reload any other value.

### 20.5.2 RTC write enable

Register RTC\_RWEN bit 15~0 is RTC read /write password. It is used to avoid signal interference from system during system power off. RTC\_RWEN bit 15~0 has to be set as 0xa965 before user want to write new data into all registers besides RTC\_INIT. If user set RTC\_RWEN as 0xa965, RWENF will be raised high. Then user can feel free to write data into register. RWENF will keep high for a short period (about 24ms) and it will be pull low by internal state machine automatically. User can disable RTC clock (CLK\_PCLKEN0[2]) to reduce the Power Consumption.

RTC\_TALM, RTC\_CALM, RTC\_TIME and RTC\_CAL are all BCD counter, but RTC\_FREQADJ is not a BCD counter.

Programmer must be aware that the RTC block does not check whether the loaded value is reasonable. For example, Load RTC\_CAL as 201a (year), 13 (month), 00 (day), or RTC\_CAL does not match with RTC\_WEEKDAY, etc.

Reset Status:

Register	Value	Description
RTC_RWEN	0	RTC register read/write disable
RTC_CAL	05 , 1 ,1	2005-1-1
RTC_TIME	00 00 00	00 hour, 00 minute, 00 second
RTC_CALM	00,00,00	2000-0-0
RTC_TALM	00,00,00	00 hour, 00 minute, 00 second
RTC_TIMEFMT	1	24 hour time scale
RTC_WEEKDAY	6	Saturday

RTC_INTEN	0	Tick interrupt disable Alarm interrupt disable
RTC_INTSTS	0	Tick interrupt not occur Alarm interrupt not occur
RTC_LEAPYEAR	0	This year not leap year
RTC_TICK	0	Time tick enable

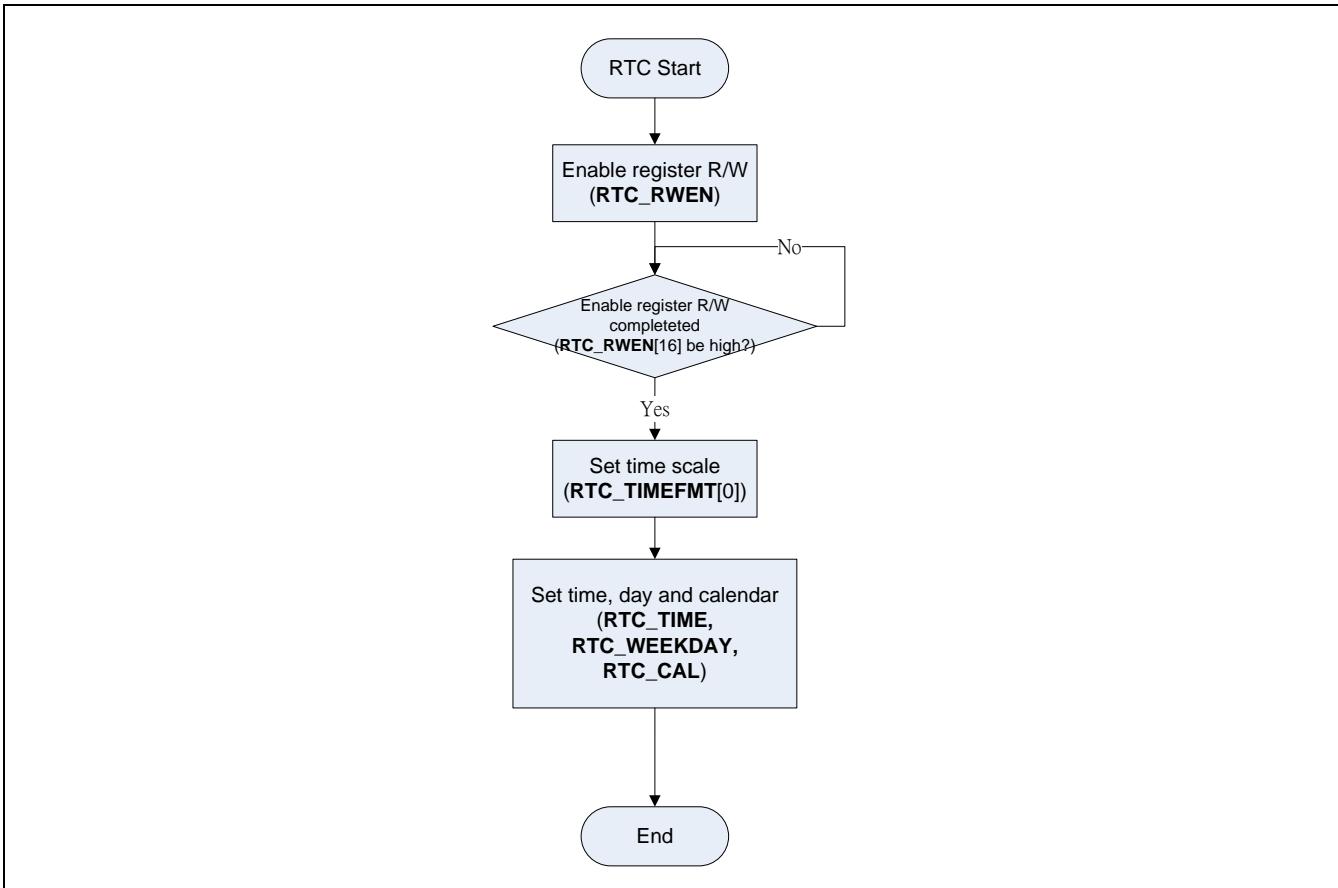
### 20.5.3 12/24 hour Time scale selection

The 12/24 hour time scale selection decided by 24HEN (RTC\_TIMEFMT[0]).

24-hour time scale	12-hour time scale	24-hour time scale	12-hour time scale
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)

### 20.5.4 Set Calendar and Time

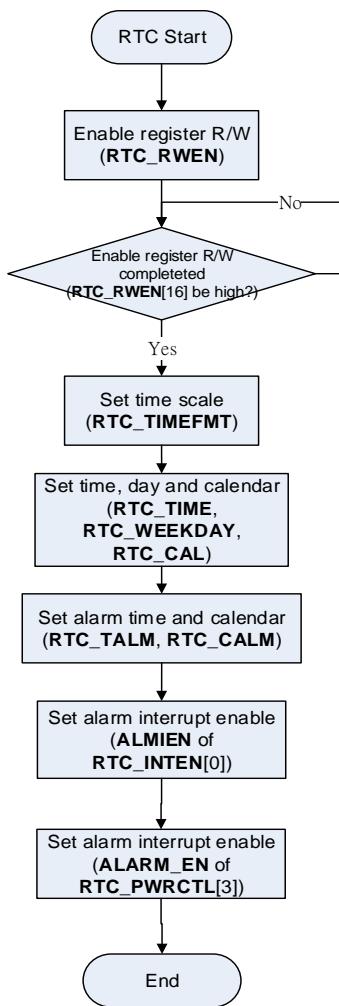
1. Write 0xa965 to RTC\_RWEN means enable RTC access enable/disable password
2. Read register RWENF(RTC\_RWEN[16]), RTC is read/write enable if it's equal to 1.
3. RWENF(RTC\_RWEN[16]) will be cleared automatically after 1024 RTC clock
4. Set register 24HEN(RTC\_TIMEFMT[0]) bit. (select to 24/12-hour time scale).
5. Set year, month and day to register RTC\_CAL
6. Set day of week to register RTC\_WEEKDAY
7. Set hour, minute and second to register RTC\_TIME



### 20.5.5 Set Calendar and Time Alarm (Absolute)

1. Set ALMINT(RTC\_INTSTS[0]) = 1 to clear alarm interrupt.
2. Set time and calendar same as above step 1-7
3. Set alarm year, month and day to register RTC\_CALM.
4. Set alarm hour, minute and second to register RTC\_TALM
5. Set the bit ALMIEN(RTC\_INTEN[0]) for alarm interrupt enable.
6. Set the bit ALARM\_EN(RTC\_PWRCTL[3]) for alarm function enable.

Note: Week of Day also the alarm condition

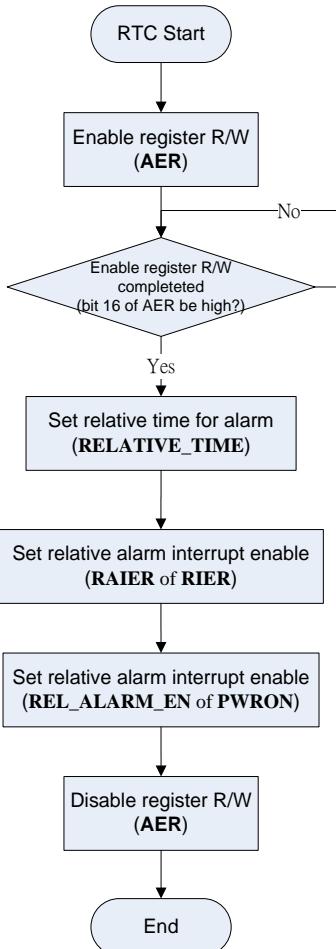


### 20.5.6 Set Time Alarm (Relative)

1. Set and prepare the RTC\_INTSTS of RTC alarm
2. Write 0xA965 to RTC\_RWEN means enable RTC access enable password
3. Read register bit RWENF(RTC\_RWEN[16]), RTC is read/write enable if it's equal to 1.
4. Set the relative time to RELALM\_TIME(RTC\_PWRCTL[27:16])
5. Maximum relative time is 1800 (about 30 minutes)
6. Set the bit RELALMIEN(RTC\_INTEN[4]) for alarm interrupt enable.
7. Set the bit REL\_ALARM\_EN(RTC\_PWRCTL[4]) for relative alarm interrupt enable

Note: Please disable relative alarm interrupt enable (RELALMIEN(RTC\_INTEN[4])) after the

alarm occurs. Otherwise, it will issue interrupt again after 30 minutes



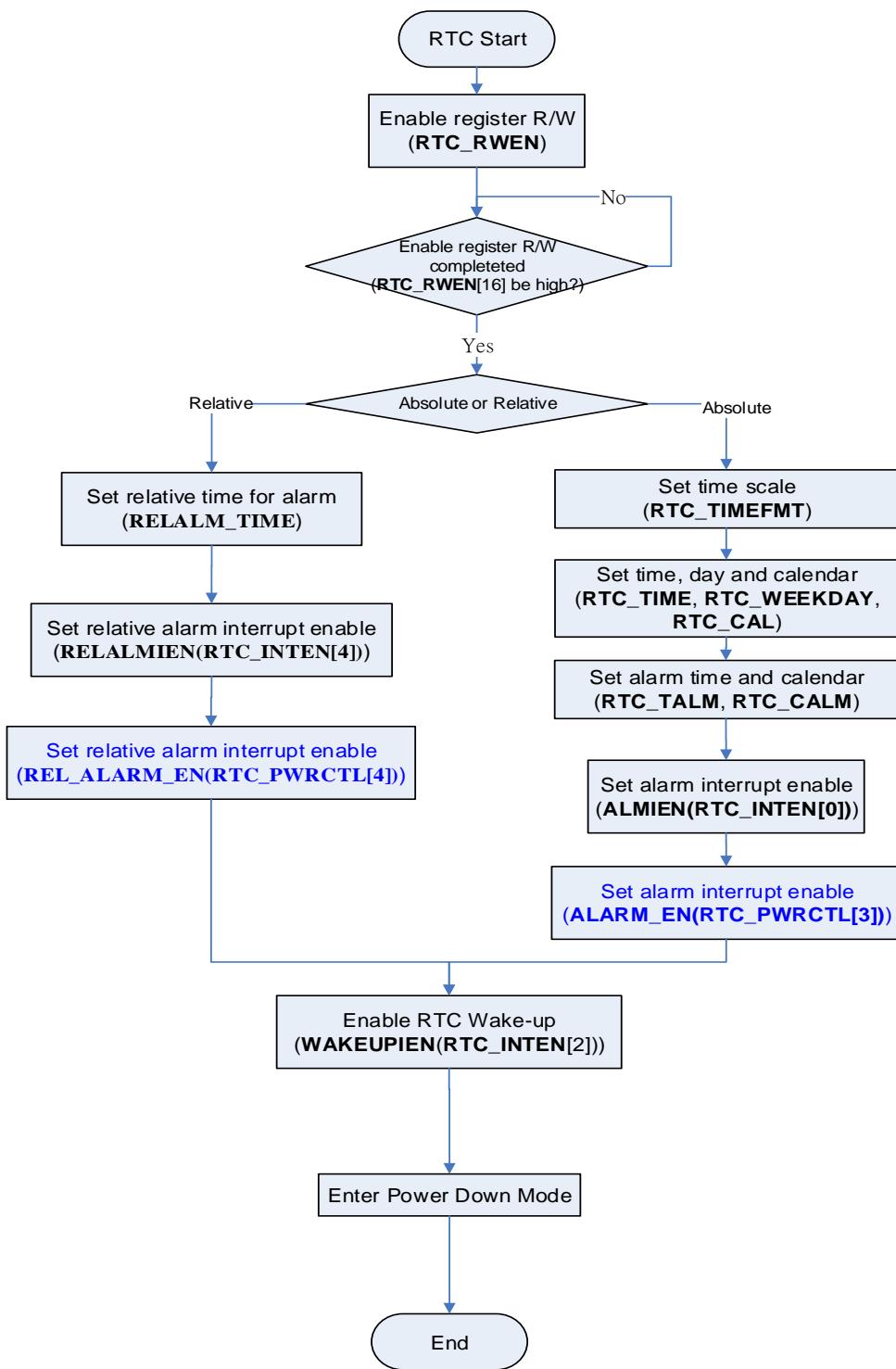
### 20.5.7 Set wake-up function

The programming procedure listed is as follows:

1. Set and prepare the RTC\_INTSTS of RTC wake-up interrupt
2. Set absolute or relative alarm
3. Enable RTC Wakeup enable (WAKEUPIEN(RTC\_INTEN[2]))
4. Let system enter power down mode
5. When RTC reach alarm time, system will wake-up system

If user won't enable wakeup function, please don't enable the alarm enable bit (WAKEUPIEN(RTC\_INTEN[2])).

Please disable relative alarm interrupt enable (RELALMIEN(RTC\_INTEN[4])) after the alarm occurs. Otherwise, it will issue interrupt again after 30 minutes.

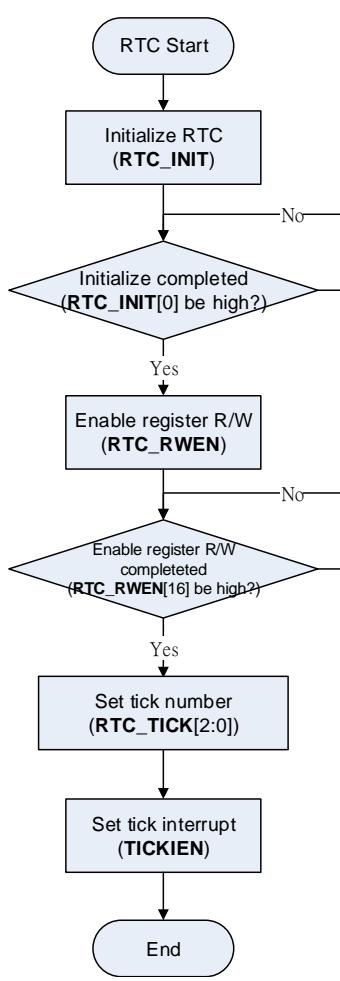


### 20.5.8 Set tick interrupt

The periodic RTC Time Tick interrupt has 8 period interval options 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2 and 1 second which are selected by RTC\_TICK (RTC\_TICK[2:0] Time Tick Register).

The programming procedure listed is as follows:

1. Set and prepare the RTC\_INTSTS of RTC tick interrupt
2. Write 0xA965 to AER means enable RTC access enable password.
3. Read register bit RWENF(RTC\_RWEN[16]), RTC is read/write enable if it's equal to 1.
4. Set the RTC\_TICK[2:0] for tick interrupt happen time interval per second
5. Set the bit TICKIEN(RTC\_INTEN[1]) for alarm interrupt enable



## 20.5.9 System Power Control Flow

### 20.5.9.1 Set System Power On

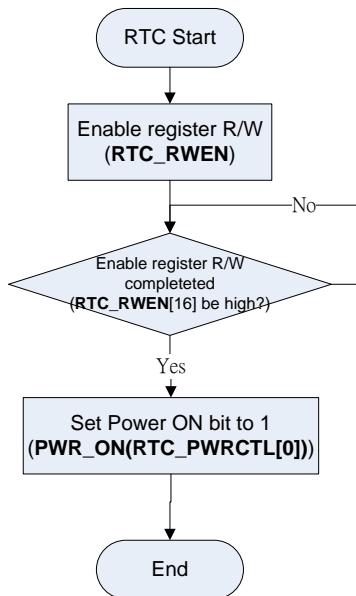
User presses the Power Key to make the Power Control Signal, PWCE to high. If PWR\_ON(PWRON[0]) set to 1, the PWCE will keep on when the Power Key is released. If PWR\_ON(PWRON[0]) doesn't be set to 1, the PWCE will back to low when the Power Key is released.

The power control flow listed is as follows:

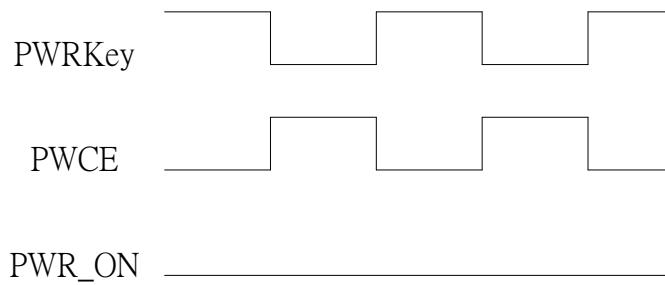
1. Press PWR Key to force the System power on
2. When RTC block is power on, programmer has to write a number 0xA5EB1357 to INIR to reset all logic RTC
3. Read register bit RTC\_INIT[0] if this bit equals to 1 means RTC has been set.
4. Write 0xA965 to RTC\_RWEN means enable RTC access enable password
5. Read register bit RWENF(RTC\_RWEN[16]), RTC is read/write enable if it's equal to 1.
6. RWENF(RTC\_RWEN[16]) will be cleared automatically after 1024 RTC clock
7. Set bit PWR\_ON(RTC\_PWRCTL[0]) to 1 to make system kept power on.

There are two Power key trigger mode:

- 1: EDGE TRIGE
  - RTC is powered on while power key is pressed longer than programmed duration and then released
- 0: LEVEL TRIGGER
  - RTC is powered on while power key is pressed longer programmed duration



User presses the Power Key to make the Power Control Signal, PWCE to high. If **PWR\_ON** doesn't be set to 1, the PWCE will back to low when the Power Key is released



#### 20.5.9.2 Force Power off Flow

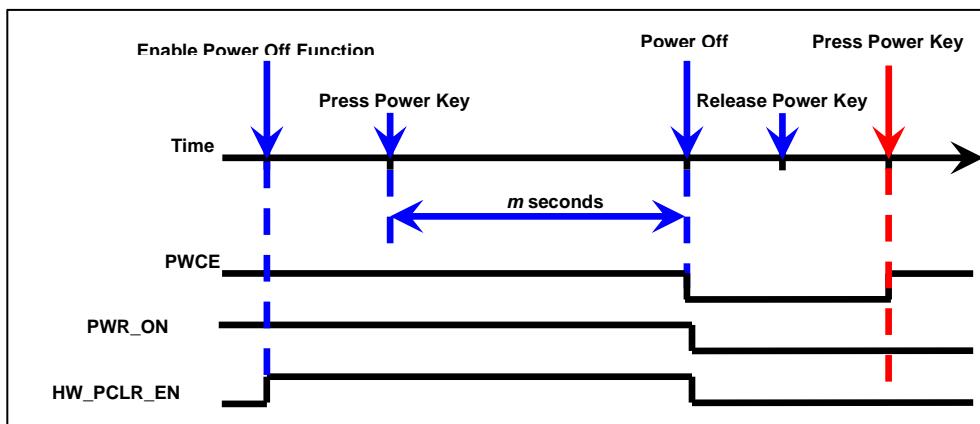
If there is another pulse on power key when the PWR\_ON bit is set, the system will get an interrupt signal (PWRSWINT). User can decide to clear the PWR\_ON or not. If this bit is clear, the PWCE will go to low to turn off the core power. If the PWR\_ON bit is also kept high, the PWCE pin will keep in high level. If there is not any pulse on the power key and the PWR\_ON bit is clear by user, the PWCE pin is also set to low at this time.

For hardware power off function, it can be enable and disable in HW\_PCLR\_EN bit and the user presses the power button for a few seconds to power off system. The time to press power the button to power off is configured in PWROFF\_TIME(RTC\_PWRCTL[23:16])

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

The RTC supports a hardware power off function to provide the power off flow like Notebook. The user presses the power button for a few seconds to power off the system. The time to press power key to power off is counted by hardware. After the time, hardware will set the PWCE to low and clear the PWR\_ON (But HW\_PCLR\_EN will not be clear). After power off, user can decide to set the PWR\_ON bit to power on system or not when the Power Key is pressed.

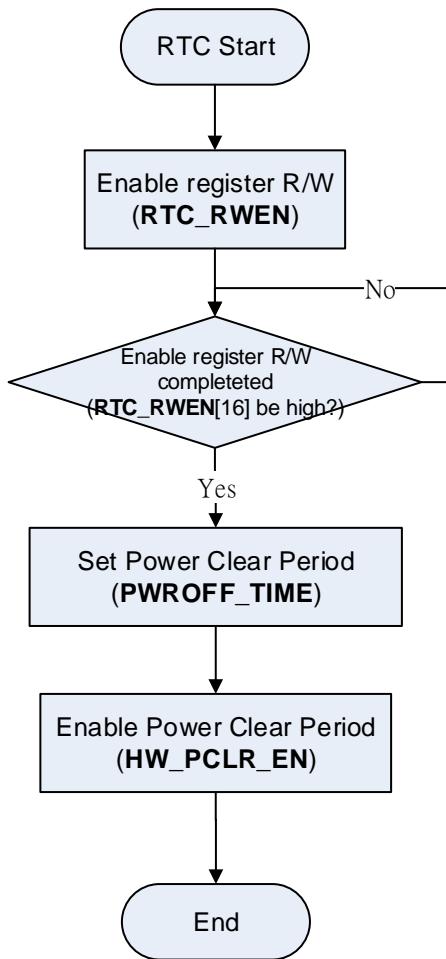
The timing of the hardware power off function is following



Hardware power off system flow:

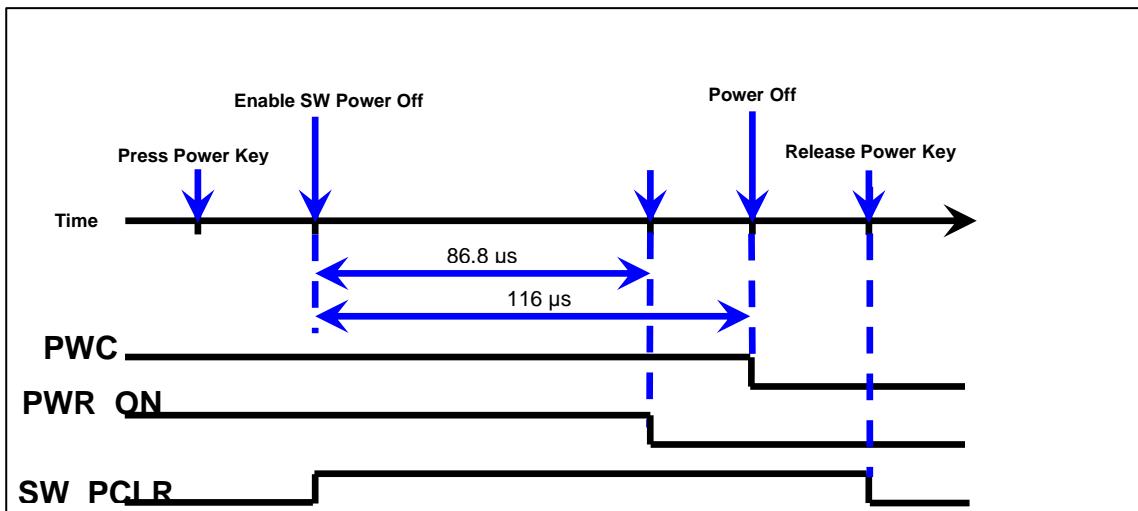
1. Write 0xA965 to RTC\_RWEN means enable RTC access enable password
2. Read register bit RWENF(RTC\_RWEN[16]), RTC is read/write enable if it's equal to 1
3. RWENF(RTC\_RWEN[16]) will be cleared after 1024 RTC clocks
4. Set Power Clear Period (WROFF\_TIME(RTC\_PWRCTL[15:12])) to set desired time to pressing Power Key for power off

## 5. Enable Power Clear Period (HW\_PCLR\_EN(RTC\_PWRCTL[2]))



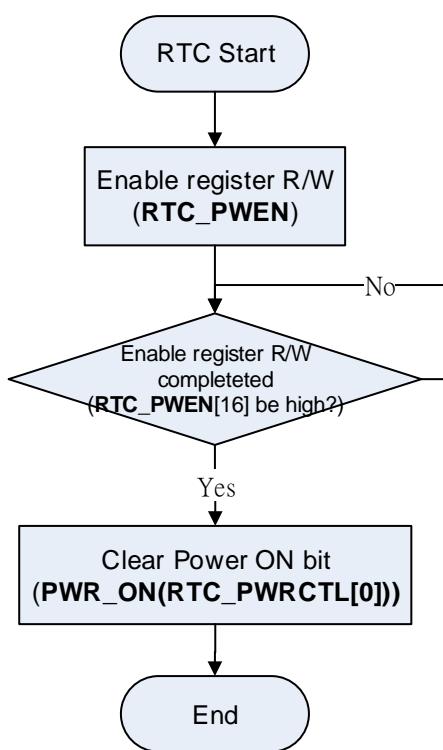
### 20.5.9.3 Software Power off System

The RTC also supports a software power off function. The user presses the power button for a few seconds to power off the system. The time to press power key to power off is counted by user. When the **PWR\_ON** bit is cleared by user, the PWCE outputs low after 116us and the **SW\_PCLR** bit is cleared when the power key is released. See the timing Figure as following.



Software power off system flow:

1. Write 0xA965 to RTC\_RWEN means enable RTC access enable password
2. Read register bit RWENF(RTC\_RWEN[16]), RTC is read/write enable if it's equal to 1.
3. This bit RWENF(RTC\_RWEN[16]) will be cleared automatically after 1024 RTC clock.
4. Clear PWR\_ON(RTC\_PWRCTL[0]) bit to zero



Following is the Power Control Flow:

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

### 20.5.10 Frequency Compensation:

The RTC\_FREQADJ allows software to make digital compensation to a clock input. The frequency of clock input must be in the range from 32776Hz to 32761Hz. User can utilize a frequency counter to measure RTC clock on one of PH.4 and PI.3 pin during manufacture, and store the value in Flash memory for retrieval when the product is first power on. Following are the compensation example:

Frequency counter measurement: 32773.65Hz

Integer part: 32773

Search Following Table:

Integer part of detected	RTC_FREQADJ[11:8]	Integer part of detected	RTC_FREQADJ[11:8]
--------------------------	-------------------	--------------------------	-------------------

value		value	
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[11:8](integer part) is 0xC

Fraction part :  $0.65 * 60 = 39 = 0x27$ , RTC\_FREQADJ[7:0](Fraction part) is 0x27

The register RTC\_FREQADJ is 0xC27

## 21 Smart Card Host Interface (SC)

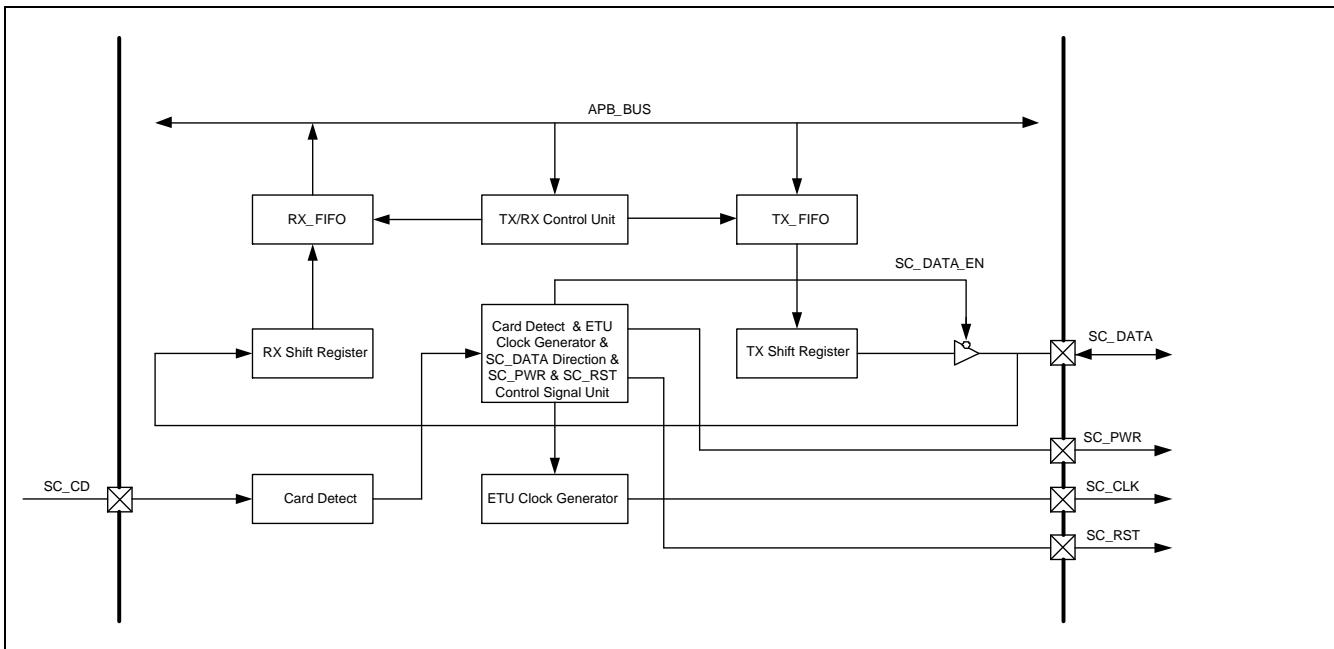
### 21.1 Overview

The Smart Card Interface controller (SC controller) is based on ISO/IEC 7816-3 standard and fully compliant with PC/SC Specifications. It also provides status of card insertion/removal.

### 21.2 Features

- ISO-7816-3 T = 0, T = 1 compliant
- EMV2000 compliant
- Up to two ISO-7816-3 ports
- Separates receive/transmit 4 byte entry FIFO for data payloads
- Programmable transmission clock frequency
- Programmable receiver buffer trigger level
- Programmable guard time selection (11 ETU ~ 267 ETU)
- A 24-bit and two 8-bit timers for Answer to Request (ATR) and waiting times processing
- Supports auto inverse convention function
- Supports transmitter and receiver error retry and error number limiting function
- Supports hardware activation sequence, hardware warm reset sequence and hardware deactivation sequence process
- Supports hardware auto deactivation sequence when detected the card removal
- Supports UART mode
  - Full duplex, asynchronous communications
  - Separates receiving / transmitting 4 bytes entry FIFO for data payloads
  - Supports programmable baud rate generator for each channel
  - Supports programmable receiver buffer trigger level
  - Programmable transmitting data delay time between the last stop bit leaving the TX-FIFO and the de-assertion by setting EGT (SC\_EGT[7:0])
  - Programmable even, odd or no parity bit generation and detection
  - Programmable stop bit, 1- or 2- stop bit generation

### 21.3 Block Diagram



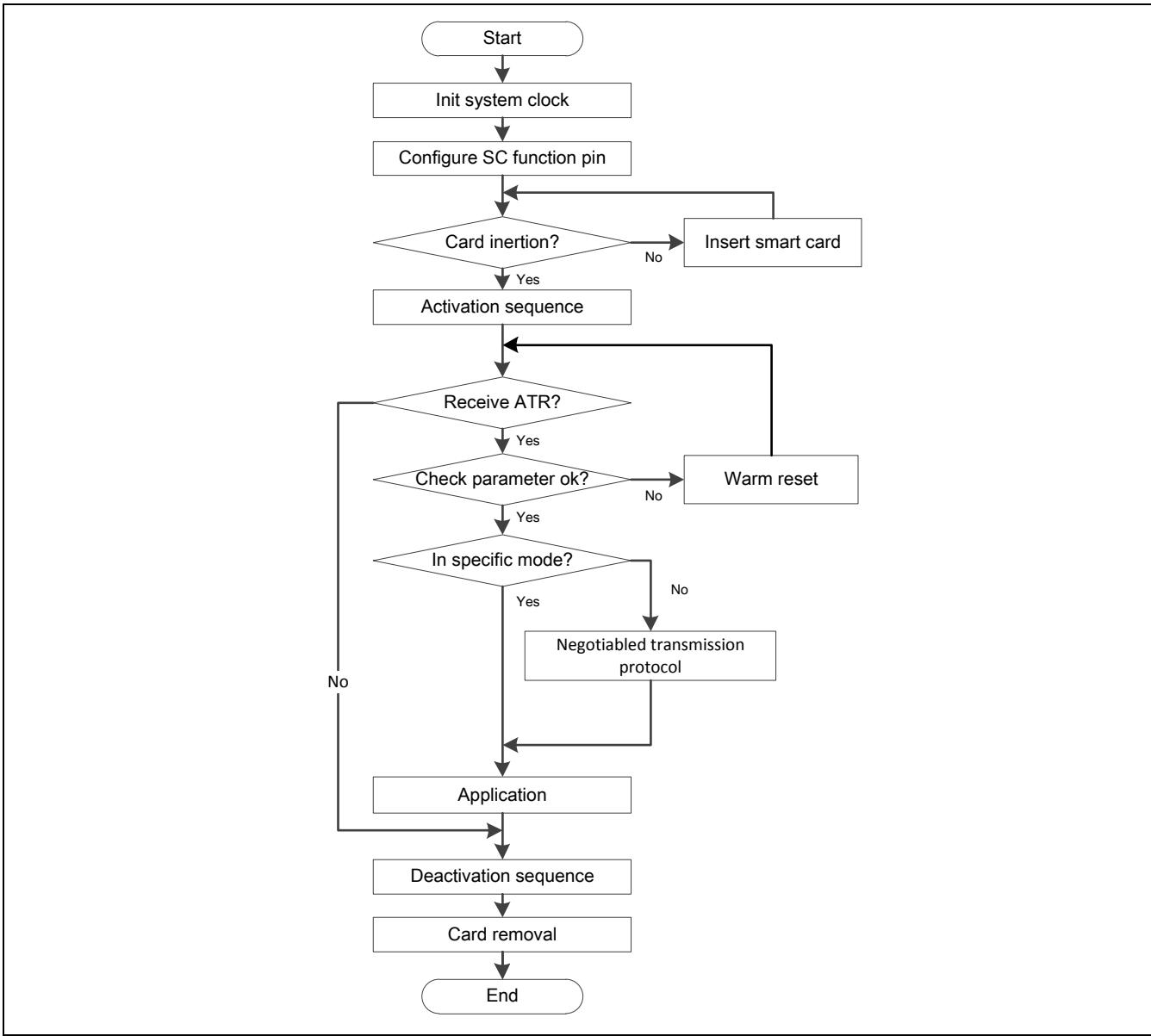
## 21.4 Register Map

Register	Offset	R/W	Description	Reset Value
<b>SC Base Address:</b>				
<b>SC0_BA</b>	= 0xB800_5000			
<b>SC1_BA</b>	= 0xB800_5400			
<b>SC_DAT x = 0,1</b>	SCx_BA+0x00	R/W	SC Receiving/Transmit Holding Buffer Register	0xFFFF_FFFF
<b>SC_CTL x = 0,1</b>	SCx_BA+0x04	R/W	SC Control Register	0x0000_0000
<b>SC_ALTCTL x = 0,1</b>	SCx_BA+0x08	R/W	SC Alternate Control Register	0x0000_0000
<b>SC_EGT x = 0,1</b>	SCx_BA+0x0C	R/W	SC Extend Guard Time Register	0x0000_0000
<b>SC_RXTOOUT x = 0,1</b>	SCx_BA+0x10	R/W	SC Receive Buffer Time-out Register	0x0000_0000
<b>SC_ETUCTL x = 0,1</b>	SCx_BA+0x14	R/W	SC ETU Control Register	0x0000_0173
<b>SC_INTEN x = 0,1</b>	SCx_BA+0x18	R/W	SC Interrupt Enable Control Register	0x0000_0000
<b>SC_INSTS x = 0,1</b>	SCx_BA+0x1C	R/W	SC Interrupt Status Register	0x0000_0002
<b>SC_STATUS x = 0,1</b>	SCx_BA+0x20	R/W	SC Status Register	0x0000_0202

<b>SC_PINCTL x = 0,1</b>	SCx_BA+0x24	R/W	SC Pin Control State Register	0x0000_00x0
<b>SC_TMRCTL0 x = 0,1</b>	SCx_BA+0x28	R/W	SC Internal Timer Control Register 0	0x0000_0000
<b>SC_TMRCTL1 x = 0,1</b>	SCx_BA+0x2C	R/W	SC Internal Timer Control Register 1	0x0000_0000
<b>SC_TMRCTL2 x = 0,1</b>	SCx_BA+0x30	R/W	SC Internal Timer Control Register 2	0x0000_0000
<b>SC_UARTCTL x = 0,1</b>	SCx_BA+0x34	R/W	SC UART Mode Control Register	0x0000_0000
<b>SC_TMRDAT0 x = 0,1</b>	SCx_BA+0x38	R	SC Timer Current Data Register A	0x0000_07FF
<b>SC_TMRDAT1_2 x = 0,1</b>	SCx_BA+0x3C	R	SC Timer Current Data Register B	0x0000_7F7F

## 21.5 Functional Description

This section describes the control of smartcard interface. But the content of ISO7816 and EMV is not in the scope of this document. A smartcard control flow is shown in the figure below. It is highly suggested to have basic knowledge of ISO7816 and EVM specification before develop smartcard driver.



### 21.5.1 Activation (Cold Reset)

The Smart Card Interface controller supports hardware activation, warm reset and deactivation sequence. The activation sequence are shown as follows:

- Set SC\_RST to low by programming RSTSTS (SC\_PINCTL[18]) to 0.
- Set SC\_PWR at high level by programming PWRSTS (SC\_PINCTL[18]) to 1 and SC\_DAT at high level (reception mode) by programming DATSTS (SC\_PINCTL[16]) to 1.
- Enable SC\_CLK clock by programming CLKKEEP (SC\_PINCTL[6]) to 1.
- De-assert SC\_RST to high by programming RSTSTS (SC\_PINCTL[18]) to 1.

The activation sequence can be controlled in two ways. The procedure is shown as follows:

### Software Timing Control:

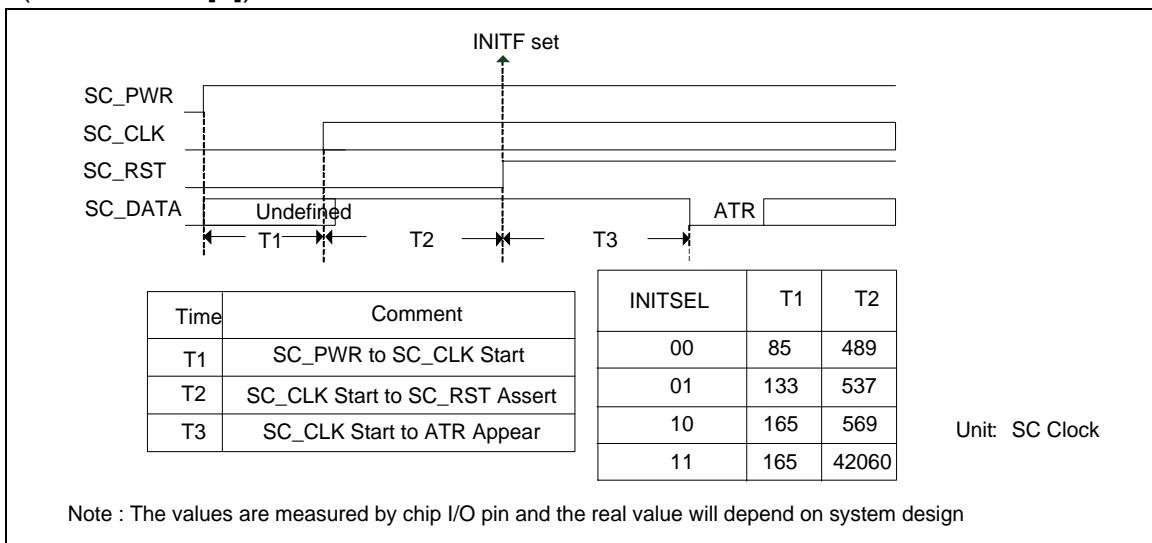
Set SC\_PINCTL and SC\_TMRCTL $x$  ( $x = 0, 1, 2$ ) to process the activation sequence. SC\_PWR, SC\_CLK, SC\_RST and SC\_DATA pin state can be programmed by SC\_PINCTL. The programming method is shown in Activation description. The activation sequence timing can be controlled by setting SC\_TMRCTL $x$  ( $x = 0, 1, 2$ ). This programming procedure provides user has a flexible timing setting for activation sequence

### Hardware Timing Control:

Set ACTEN (SC\_ALTCTL[3]) to 1 and the interface will perform the activation sequence by hardware. The SC\_PWR to SC\_CLK start (T1) and SC\_CLK start to SC\_RST assert (T2) can be selected by programming INITSEL(SC\_ALTCTL[9:8]). This programming procedure provides user has a simple setting for activation sequence.

Following is the activation control sequence generated by hardware:

1. Set activation timing by setting INITSEL (SC\_ALTCTL[9:8]).
2. TMR0 can be selected by setting TMRSEL (SC\_CTL[14:13]) is 01, 10 or 11.
3. Set operation mode OPMODE (SC\_TMRCTL0[27:24]) to 0011 and give an Answer to Request (ATR) value by setting CNT (SC\_TMRCTL0[23:0]) register.
4. When hardware de-asserts SC\_RST to high, hardware will generator an interrupt INTIF (SC\_INTSTS[8]) to CPU at the same time INITIEN (SC\_INTEN[8]) = 1.
5. If the TMR0 decreases the counter to “0” (start from SC\_RST de-assert) and the card does not response ATR before that time, hardware will generate interrupt TMR0IF (SC\_INTSTS[3]) to CPU.



### 21.5.2 Warm Reset

The warm reset sequence is showed as follows.

1. Set SC\_RST to low by programming RSTSTS (SC\_PINCTL[18]) to 0.

2. Set SC\_DAT to high by programming DATSTS (SC\_PINCTL[16]) to 1.
3. Set SC\_RST to high by programming RSTSTS (SC\_PINCTL[18]) to 1.

The warm reset sequence can be controlled in two ways. The procedure is shown as follows.

#### Software Timing Control:

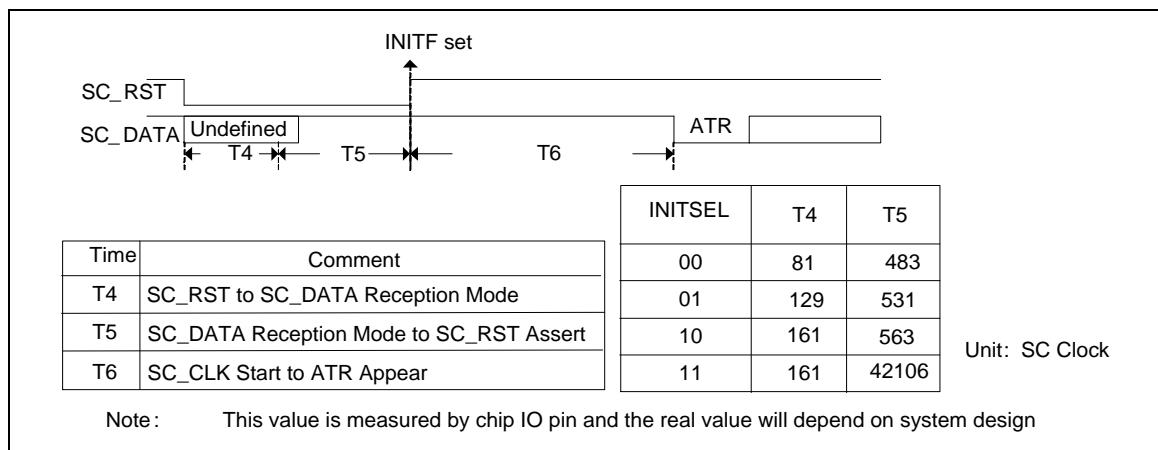
Set SC\_PINCTL and SC\_TMRCTLx ( $x = 0, 1, 2$ ) to process the warm reset sequence. SC\_RST and SC\_DATA pin state can be programmed by SC\_PINCTL. The warm reset sequence timing can be controlled by setting SC\_TMRCTLx ( $x = 0, 1, 2$ ). This programming procedure provides user has a flexible timing setting for warm reset sequence.

#### Hardware Timing Control:

Set WARSTEN (SC\_ALTCTL[4]) to 1 and the interface will perform the warm reset sequence by hardware. The SC\_RST to SC\_DATA reception mode (T4) and SC\_DATA reception mode to SC\_RST assert (T5) can be selected by programming INITSEL (SC\_ALTCTL[9:8]). This programming procedure provides user has a simple setting for warm reset sequence.

Following is THE warm reset control sequence by hardware:

1. Set warm reset timing by setting INITSEL (SC\_ALTCTL[9:8]).
2. Select TMR0 by setting TMRSEL (SC\_CTL[14:13]) register (TMRSEL can be set to 01, 10, or 11).
3. Set operation mode OPMODE (SC\_TMRCTL0[27:24]) to 0011 and give an Answer to Request value by setting CNT (SC\_TMRCTL0[23:0]) register.
4. SetCNTEN0 (SC\_ALTCTL[5]) and WARSTEN (SC\_ALTCTL[4]) to start counting.
5. When hardware de-asserts SC\_RST to high, hardware will generate an interrupt INTIF (SC\_INTSTS[8]) to CPU at the same time (INITIEN (SC\_INTEN[8]) = 1).
6. If the TMR0 decrease the counter to “0” (start from SC\_RST) and the card does not response ATR before that time, hardware will generate interrupt TMR0IF (SC\_INTSTS[3]) to CPU



### 21.5.3 Deactivation

The deactivation sequence is showed as follows:

1. Set SC\_RST to low by programming RSTSTS (SC\_PINCTL[18]) to 0.
2. Stop SC\_CLK by programming CLKKEEP (SC\_PINCTL[6]) to 0.
3. Set SC\_DATA to state low by programming DATSTS (SC\_PINCTL[16]) to 0.
4. Deactivate SC\_PWR by programming PWRSTS (SC\_PINCTL[18]) to 0.

The deactivation sequence can be controlled in two ways. The procedure is shown as follows.

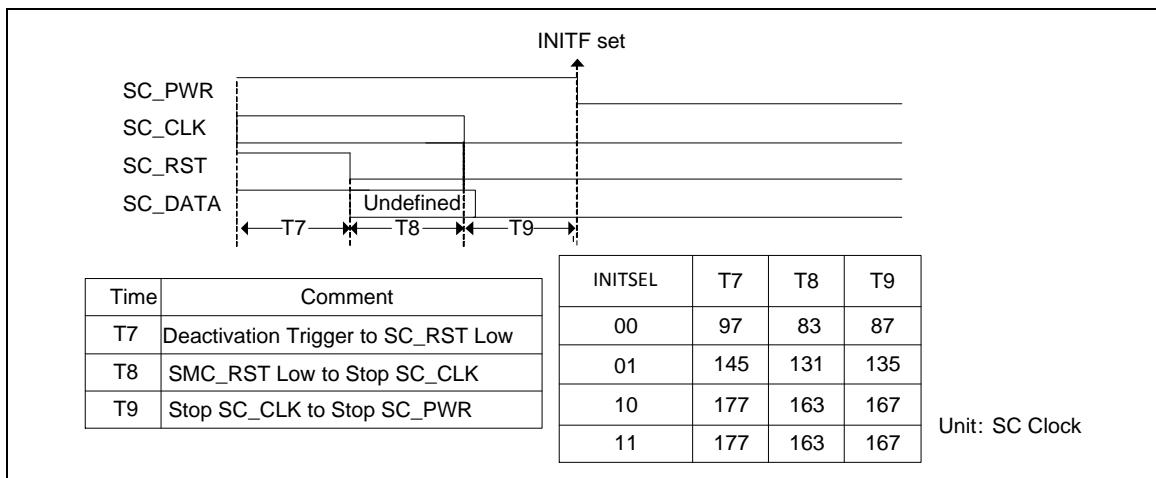
**Software Timing Control:**

Set SC\_PINCTL and SC\_TMRCTL0 to process the deactivation sequence. SC\_PWR, SC\_CLK, SC\_RST and SC\_DATA pin state can be programmed by SC\_PINCTL. The deactivation sequence timing can be controlled by setting SC\_TMRCTL0. This programming procedure provides user has a flexible timing setting for deactivation sequence.

**Hardware Timing Control:**

DACTEN (SC\_ALTCTL[2]) to '1' and the interface will perform the deactivation sequence by hardware. The Deactivation Trigger to SC\_RST low (T7), SMC\_RST low to SC\_CLK (T8) and stop SC\_CLK to stop SC\_PWR (T9) time can be selected by programming INITSEL (SC\_ALTCTL[9:8]). This programming procedure provides user has a simple setting for deactivation sequence.

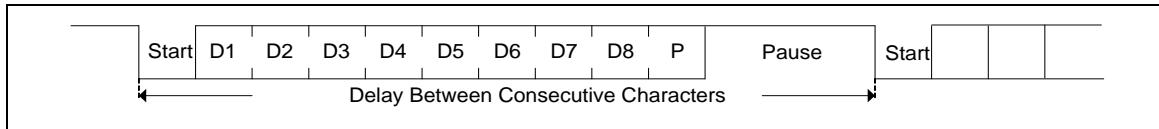
The SC controller also supports auto deactivation sequence when the card removal detection is enabled by setting ADAC\_CDEN (SC\_ALTCTL[11]).



### 21.5.4 Data Format

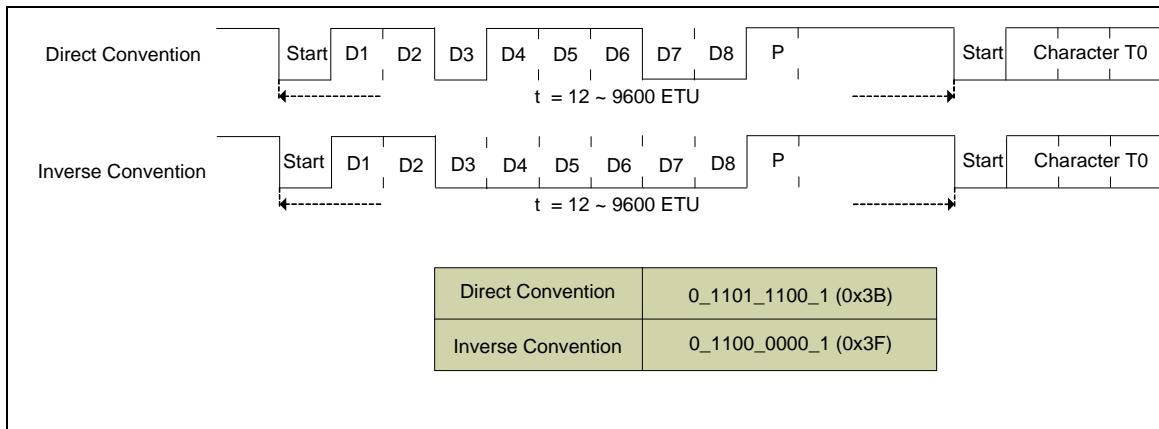
Basically, the smart card interface acts as a half-duplex asynchronous communication port

and its data format is composed of ten consecutive bits, which is show as follows.



According to 7816-3, the initial character TS has two possible patterns shown in the following figure. If the TS pattern is 1100\_0000, it is inverse convention. When decoded by inverse convention, the conveyed byte is equal to 0x3F. If the TS pattern is 1101\_1100, it is direct convention. When decoded by direct convention, the conveyed byte is equal to 0x3B. Software can set AUTOSEN (SC\_CTL[3]) and then the operating convention will be decided by hardware. Software can also set the CONSEL (SC\_CTL[5:4]) register (set to „00“ or „11“) to change the operating convention after SC received TS of answer to request (ATR).

If auto convention function is enabled by setting AUTOSEN (SC\_CTL[3]) register, the setting step must be done before Answer to Request state and the first data must be 0x3B or 0x3F. After hardware received first data and stored it at buffer, the hardware will decide the convention and change the CONSEL (SC\_CTL[5:4]) register automatically. If the first data is neither 0x3B nor 0x3F, the hardware will generate an interrupt (if ACERRIEN (SC\_INTEN[10]) = „1“) to CPU.



## 21.5.5 Data Transfer

Smartcard interface transmit and receive data through SC\_DAT register. Driver should write output data to SC\_DAT register, and read received data from SC\_DAT.

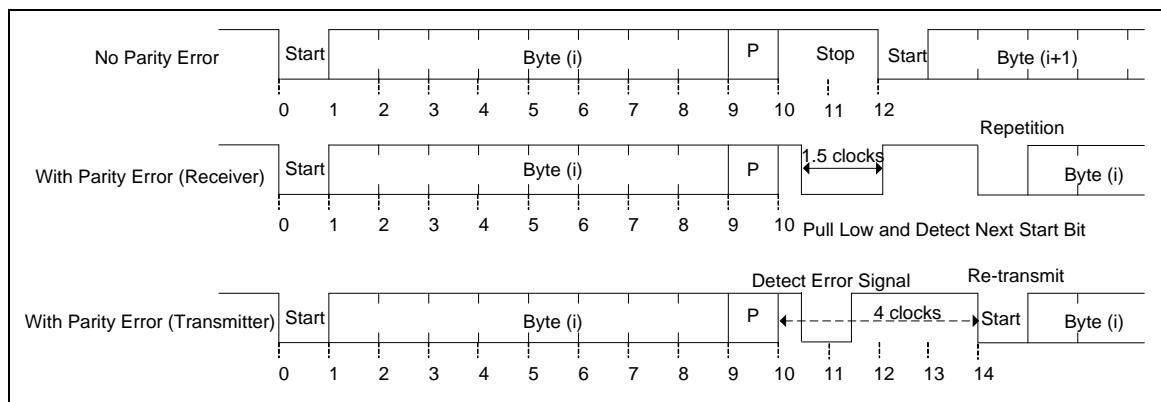
Both transmit (TX) and receive (RX) has 4 level FIFO. Driver must make sure TX FIFO is not full (TXFULL (SC\_STATUS[10]) is 0) before write any data to SC\_DAT. Otherwise TXOV(SC\_STATUS[8]) will be set 1 to indicate TX FIFO overflow. While there's data available in RX FIFO, RXEMPTY (SC\_STATUS[1]) will be cleared to 0, driver can keep read SC\_DAT until RXEMPTY (SC\_STATUS[1]) set 1 again. If RX FILL is FULL and further data comes in, RXOV(SC\_STATUS[0]) will be set 1 to indicate RX FIFO overflow.

Except polling mode, driver can use interrupt to detect the status change of TX/RX FIFO. If

TBEIEN (SC\_INTEN[1]) set 1, interrupt will be triggered when TX FIFO is empty, and TBEIF (SC\_INTSTS[1]) will be set to 1. Driver can repeatedly write at most 4 bytes data into TX FIFO until next interrupt. If RDAIEN (SC\_INTEN[0]) is 1, interrupt will be triggered if data in RX FIFO is no less than the interrupt trigger level configured in RXTRGLV (SC\_CTL[7:6]), and RDAIF (SC\_INTSTS[0]) will be set 1. Driver can keep reading SC\_DAT until RXEMPTY set 1 again. To avoid the situation that data count less than interrupt trigger level and stays in RX FIFO without trigger interrupt, driver could set a timeout duration to trigger interrupt if there is data in RX FIFO longer than the duration and does not reach the level to trigger. This timeout duration is configured in SC\_RXTOOUT register using ETU as time unit. Except configure proper value in SC\_RXTOOUT, RXTOIEN (SC\_INTEN[9]) also needs to set 1. Then smartcard controller will trigger interrupt and set RXTOIF (SC\_INTSTS[9]) to 1, to notify driver there's data available in RX FIFO.

### 21.5.6 Error Signal and Character Repetition

According to ISO7816-3 T=0 mode description, as shown in following, if the receiver receives a wrong parity bit, it will pull the SC\_DAT to low by 1.5 bit period to inform the transmitter parity error. Then the transmitter will retransmit the character. The SC interface controller supports hardware error detection function in receiver and supports hardware re-transmit function in transmitter. Software can enable re-transmit function by setting TXRTYEN (SC\_CTL[23]). Software can also define the retry (re-transmit) number limitation in TXRTY (SC\_CTL[22:20]). The re-transmit number is up to TXRTY +1 and if the re-transmit number is equal to TXRTY +1, TXOVERR flag will be set by hardware and if TERRIEN (SC\_INTEN [2]), SC controller will generate a transfer error interrupt to CPU. Software can also define the received retry number limitation in RXRTY (SC\_CTL[18:16]) register. The receiver retry number is up to RXRTY +1, if the number of received errors by receiver is equal to RXRTY +1, receiver will receive this error data to buffer and RXOVERR flag will be set by hardware and if TERRIEN (SC\_INTEN[2]), SC controller will generate a transfer error interrupt to CPU.



While working in T=1 mode, error detection is implemented in upper layer protocol. If transfer error is detected, an R-Block is sent to notify counterpart an error occurred instead of pulling SC\_DAT low. So while working in T=1 mode, both TXRTYEN and RXRTYEN must clear to 0.

## 21.5.7 Internal Time-out Counter

The smart card interface includes a 24-bit time-out counter (SC\_TMR0) and two 8 bit time-out counters (SC\_TMR1, SC\_TMR2). These counters help the controller in processing different real-time interval (ATR, WBT, WWT...). Each counter can be set to start counting once the trigger enable bit has been written or a START bit has been detected..

The following is the programming flow:

Enable counter by setting TMRSEL (SC\_CTL[14:13]). Select operation mode OPMODE (SC\_TMRCTLx[27:24]) and give a count value CNT (SC\_TMRCTLx[23:0]) by setting SC\_TMRCTLx register. Set CNTEN0 (SC\_ALTCTL[5]), CNTEN1 (SC\_ALTCTL[6]) or CNTEN2 (SC\_ALTCTL[7]) is to start counting.

The SC\_TMRCTL0, SC\_TMRCTL1 and SC\_TMRCTL2 timer operation mode are listed in below table.

Note: Only SC\_TMRCTL0 supports mode 0011

OPMODE (SC_TMRCTLx[27:24]) (X=0 ~2)	Operation Description	
0000	The down counter started when CNTENx (SC_ALTCTL[7:5]) enabled and ended when counter time-out. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.	
	Start	Start counting when CNTENx (SC_ALTCTL[7:5]) enabled
	End	When the down counter equals to 0, hardware will set TMRxIF (SC_INTSTS[5:3]) and clear CNTENx (SC_ALTCTL[7:5]) automatically.
0001	The down counter started when the first START bit (reception or transmission) detected and ended when counter time-out. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.	
	Start	Start counting when the first START bit (reception or transmission) detected after CNTENx (SC_ALTCTL[7:5]) set to 1.
	End	When the down counter equals to 0, hardware will set TMRxIF (SC_INTSTS[5:3]) and clear CNTENx (SC_ALTCTL[7:5]) automatically.
0010	The down counter started when the first START bit (reception) detected and ended when counter time-out. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.	
	Start	Start counting when the first START bit (reception) detected bit after CNTENx (SC_ALTCTL[7:5]) set to 1.
	End	Start counting when the first START bit (reception) detected bit after CNTENx (SC_ALTCTL[7:5]) set to 1.
0011	The down counter is only used for hardware activation, warm reset sequence to measure ATR timing. The timing starts when SC_RST de-assertion and ends when ATR response received or time-out. If the counter decreases to 0 before ATR response received, hardware will generate an interrupt to CPU. The time-out value will be CNT (SC_TMRCTL0[23:0]) + 1.	
	Start	Start counting when SC_RST de-assertion after CNTEN0 (SC_ALTCTL[5]) set to 1. It is used for hardware activation, warm reset mode.
	End	When the down counter equals to 0 before ATR response received, hardware will set TMR0IF (SC_INTSTS[3]) and clear CNTEN0 (SC_ALTCTL[5]) automatically.

		When ATR received and down counter does not equal to 0, hardware will clear CNTENx (SC_ALTCTL[5]) automatically.
0100		<p>Same as 0000, but when the down counter equals to 0, hardware will set TMRxIF (SC_INTSTS[5:3]) and counter will re-load the CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) value and re-count until software clears CNTENx (SC_ALTCTL[7:5]).</p> <p>When ACTSTSx (SC_ALTCTL[15:13]) = 1, software can change CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) value at any time. When the down counter equals to 0, counter will reload the new value of CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) and re-count.</p> <p>The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.</p>
0101		<p>Same as 0001, but when the down counter equals to 0, hardware will set TMRxIF (SC_INTSTS[5:3]) and counter will re-load the CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) value. When the next START bit is detected, counter will re-count until software clears CNTENx (SC_ALTCTL[7:5]).</p> <p>When ACTSTSx (SC_ALTCTL[15:13]) = 1 software can change CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) value at any time. When the down counter equal to 0, it will reload the new value of CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) and re-counting.</p> <p>The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.</p>
0110		<p>Same as 0010, but when the down counter equals to 0, it will set TMRxIF (SC_INTSTS[5:3]) and counter will re-load the CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) value. When the next START bit is detected, counter will re-count until software clears CNTENx (SC_ALTCTL[7:5]).</p> <p>When ACTSTSx (SC_ALTCTL[15:13]) = 1, software can change CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) value at any time. When the down counter equals to 0, counter will reload the new value of CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) and re-count.</p> <p>The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.</p>
0111		<p>The down counter started when the first START bit (reception or transmission) detected and ended when software clears CNTENx (SC_ALTCTL[7:5]) bit. If next START bit detected, counter will reload the new value of CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) and re-counting.</p> <p>If the counter decreases to 0 before the next START bit detected, hardware will generate an interrupt to CPU. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.</p>
	Start	Start counting when the first START bit detected after CNTENx (SC_ALTCTL[7:5]) set to 1.
	End	Stop counting after CNTENx (SC_ALTCTL[7:5]) set to 0.
1000		<p>The up counter starts when CNTENx (SC_ALTCTL[7:5]) enabled and ends when CNTENx (SC_ALTCTL[7:5]) disabled. This count value will be stored in CNTx (SC_TMRDAT0[23:0], SC_TMRDAT1_2[7:0], SC_TMRDAT1_2[15:8]). In this mode, hardware cannot generate any interrupt to CPU. The real count value will be CNTx (SC_TMRDAT0[23:0], SC_TMRDAT1_2[7:0], SC_TMRDAT1_2[15:8]) + 1.</p>
	Start	Start counting after CNTENx (SC_ALTCTL[7:5]) set to 1, and the start count value is 0 (hardware will ignore CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) value).
	End	Stop counting after CNTENx (SC_ALTCTL[7:5]) set to 0 and the value stored to CNTx (SC_TMRDAT0[23:0], SC_TMRDAT1_2[7:0], SC_TMRDAT1_2[15:8]) register
1111		<p>Down counter starts when software set CNTENx (SC_ALTCTL[7:5]) bit or any START bit been detected and ends when software clears CNTENx (SC_ALTCTL[7:5]) bit. If next START bit detected, counter will reload the new value of CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) and re-counting.</p> <p>If the counter decreases to "0" before the next START bit be detected, hardware will generate an interrupt to CPU. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0])+1.</p>
	Start	Start count when the CNTENx (SC_ALTCTL[7:5]) set to "1" or any START bit (CNTENx (SC_ALTCTL[7:5]) must be set) be detected.
	End	Stop count after CNTENx (SC_ALTCTL[7:5]) set to "0".

### 21.5.8 Smartcard Insert/Remove Detection

Smartcard interface can detect the presence if smartcard. But to correctly detect the status,

CDLV (SC\_CTL[26]) must be configured according the card slot in use. When set to 1, SC\_CD high means card inserted, low means card removed. When clear to 0, SC\_CD high means card removed, low means card inserted. Smartcard interface also support four level de-bounce function which can be configured by CDDBSEL (SC\_CTL[25:24]) bits.

Current level of SC\_CD pin can be checked by polling CDPINSTS(SC\_STATUS) bit. This bit reflects current level of SC\_CD pin regardless of the setting of CDLV bit. During normal operation, driver could use interrupt to detect card status change. If CDIEN (SC\_INTEN[7]) set to 1, every time card presence state change will trigger an interrupt to CPU, and set CDIF (SC\_INTSTS[7]) to 1. In the interrupt service routine, software can check CINSERT (SC\_STATUS[12]) and CREMOVE (SC\_STATUS[11]) to know current card detection status. Writing 1 to them can clear CDIF, CINSERT, and CREMOVE bits

### 21.5.9 Miscellaneous Transmission Settings

Here introduce some transmission relative settings

- Elementary Time Unit (ETU)

ETU is the elementary time unit used in smartcard data transmission. And its default value is 372 clocks. After PPS exchange, ETU can change to other value by setting ETURDIV (SC\_ETUCTL[11:0]). Actual ETU is ETURDIV + 1 clocks.

- Stop Bit

While receiving ATR or working in T=0 mode, NSB(SC\_CTL[15]) needs to clear to 0 to make the interface communicate using 2 stop bits. Only 1 Stop bit is used when working in T=1 mode, so NSB(SC\_CTL[15]) needs clear to 0.

- Block Guard Time (BGT)

According to ISO 7816-3, BGT, the minimum delay between transfer from different directions is 11 ETU while working in T=1 mode. BGT is configured in BGT (SC\_CTL[12:8]) bits. If smartcard sends response within BGT time, and BGTIEN (SC\_INTEN[6]) is 1, an interrupt will be triggered and BGTF (SC\_INTSTS[6]) will be set 1. Write 1 can clear BGTF bit.

- Extra Guard Time (EGT)

According to ISO 7816-3, if TC1 exist in ATR and does not equal to 255, guard time is  $12\text{ETU} + \text{F/D} * \text{N} / \text{f} = (12 + \text{N})$ . Where N is the EGT. EGT is set in SC\_EGT register.

### 21.5.10 UART Mode

When the UARTEN (SC\_UARTCTL[0]) bit set, the Smart Card Interface controller can also be used as base UART function. The following is the program example for UART mode. Below is a programming example:

1. Set UARTEN (SC\_UARTCTL[0]) bit to enter UART mode.
2. Do software reset by setting RXRST (SC\_ALTCTL[1]) and TXRST (SC\_ALTCTL[0]) bit to ensure that all state machine return idle state.

3. Fill “0” to CONSEL (SC\_CTL[5:4]) and AUTOSEN (SC\_CTL[3]) field. (In UART mode, those fields must be “0”)
4. Select the UART baud rate by setting ETURDIV (SC\_ETUCR[11:0]) fields. For example, if smartcard module clock is 12 MHZ and target baud rate is 115200bps, ETURDIV should fill with (12000000 / 115200 - 1).
5. Select the data format include data length (by setting WLS (SC\_UARTCTL[5:4])), parity format (by setting OPE (SC\_UARTCTL[7]) and PBOFF (SC\_UARTCTL[6])) and stop bit length (by setting NSB (SC\_CTL[15]) or EGT (SC\_EGT[7:0])).
6. Select the receiver buffer trigger level by setting RXTRGLV (SC\_CTL[7:6]) field and select the receiver buffer time-out value by setting RFTM (SC\_RXTOUT[8:0]) field.
7. Write SC\_DAT (SC\_DAT[7:0]) (TX) register or read the SC\_DAT (SC\_DAT[7:0]) (RX) register can perform UART function.

## 22 Secure Digital Host Controller (SDH)

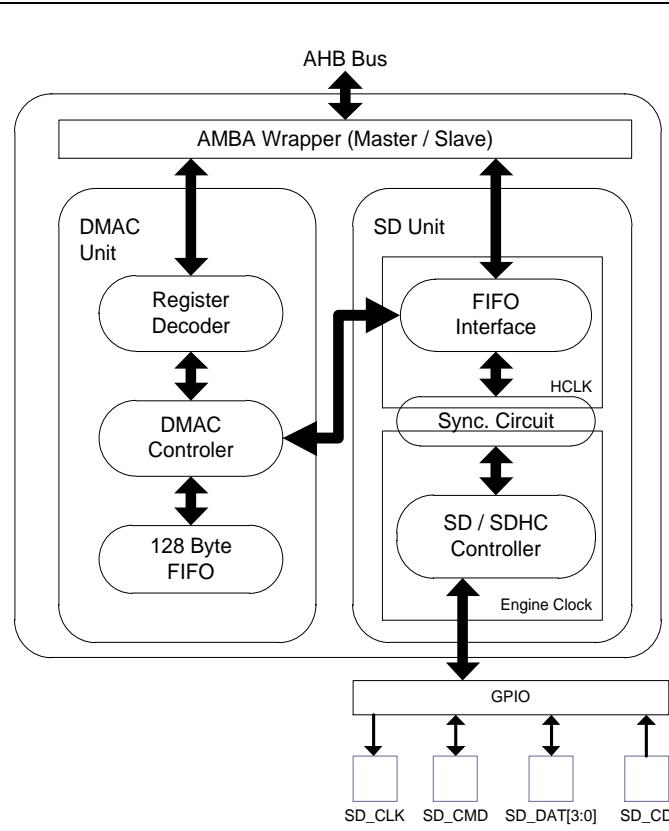
### 22.1 Overview

The Secure-Digital Card Host Controller (SDH) equips DMAC unit and SD unit. The DMAC unit provides a DMA (Direct Memory Access) function for SD to exchange data between system memory and shared buffer (128 bytes), and the SD unit controls the interface of SD / SDHC / SDIO. The SDH controller supports SD / SDHC / SDIO card and cooperates with DMAC to provide a fast data transfer between system memory and cards.

### 22.2 Features

- Supports single DMA channel
- Supports hardware Scatter-Gather functionality
- Supports 128 Bytes shared buffer for data exchange between system memory and cards
- Supports SD, SDHC and SDIO card

### 22.3 Block Diagram



## 22.4 Register Map

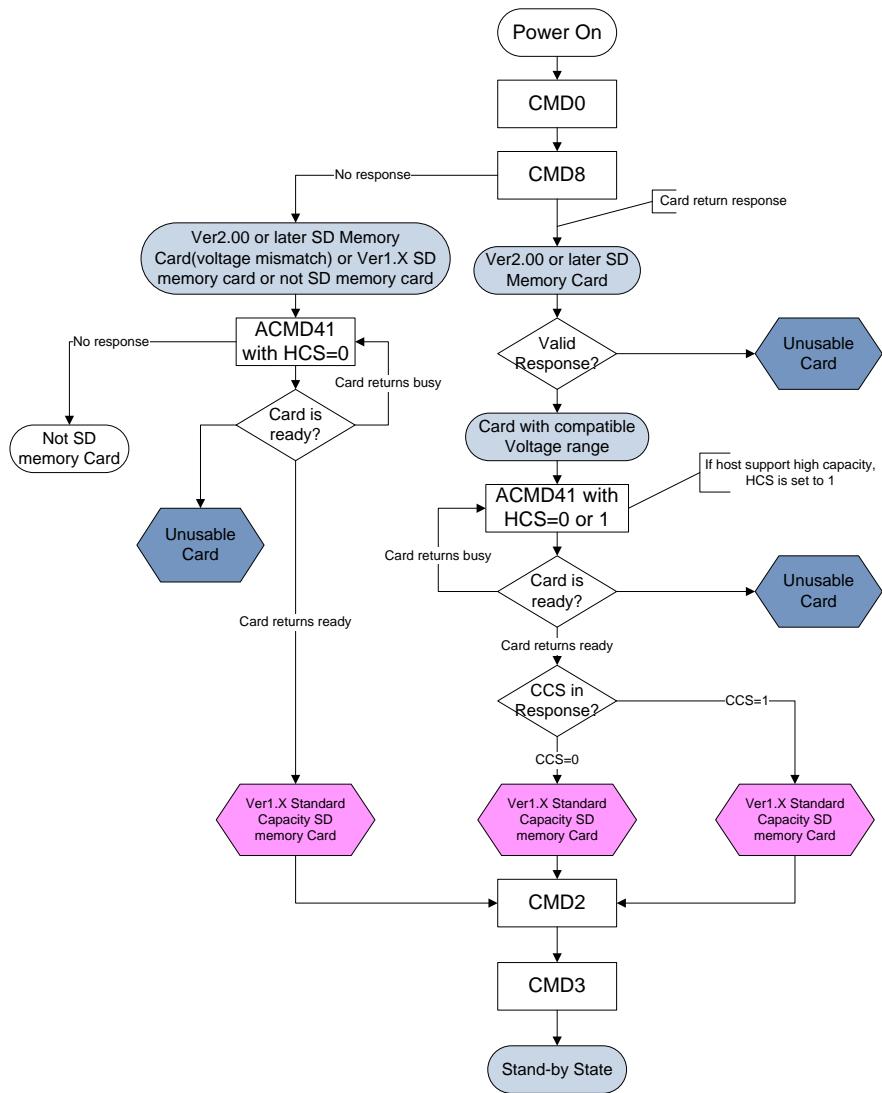
R: read only, W: write only, R/W: both read and write

Register	Offset	R/W	Description	Reset Value
<b>SDH_BA = 0xB000_C000</b>				
<b>SDH_FB_n</b> <b>n = 0,1...31</b>	SDH_BA+0x000 + 0x4 * n	R/W	SD Host Embedded Buffer Word n n = 0,1...31	0x0000_0000
<b>SDH_DMACTL</b>	SDH_BA+0x400	R/W	SD Host DMA Control and Status Register	0x0000_0000
<b>SDH_DMASA</b>	SDH_BA+0x408	R/W	SD Host DMA Transfer Starting Address Register	0x0000_0000
<b>SDH_DMABCNT</b>	SDH_BA+0x40C	R	SD Host DMA Transfer Byte Count Register	0x0000_0000
<b>SDH_DMAINTEN</b>	SDH_BA+0x410	R/W	SD Host DMA Interrupt Enable Register	0x0000_0001
<b>SDH_DMAINTSTS</b>	SDH_BA+0x414	R/W	SD Host DMA Interrupt Status Register	0x0000_0000
<b>SDH_GCTL</b>	SDH_BA + 0x800	R/W	SD Host Global Control and Status Register	0x0000_0000
<b>SDH_GINTEN</b>	SDH_BA + 0x804	R/W	SD Host Global Interrupt Control Register	0x0000_0001
<b>SDH_GINTSTS</b>	SDH_BA + 0x808	R/W	SD Host Global Interrupt Status Register	0x0000_0000
<b>SDH_CTL</b>	SDH_BA + 0x820	R/W	SD Host Control and Status Register	0x0101_0000
<b>SDH_CMD</b>	SDH_BA + 0x824	R/W	SD Host Command Argument Register	0x0000_0000
<b>SDH_INTEN</b>	SDH_BA + 0x828	R/W	SD Host Interrupt Enable Register	0x0000_0A00
<b>SDH_INTSTS</b>	SDH_BA + 0x82C	R/W	SD Host Interrupt Status Register	0x000X_008C
<b>SDH_RESP0</b>	SDH_BA + 0x830	R	SD Host Receiving Response Token Register 0	0x0000_0000
<b>SDH_RESP1</b>	SDH_BA + 0x834	R	SD Host Receiving Response Token Register 1	0x0000_0000
<b>SDH_BLEN</b>	SDH_BA + 0x838	R/W	SD Host Block Length Register	0x0000_01FF
<b>SDH_TMOUT</b>	SDH_BA + 0x83C	R/W	SD Host Response/Data-in Time-out Register	0x0000_0000
<b>SDH_ECTL</b>	SDH_BA + 0x840	R/W	SD Host Extend Control Register	0x0000_0003

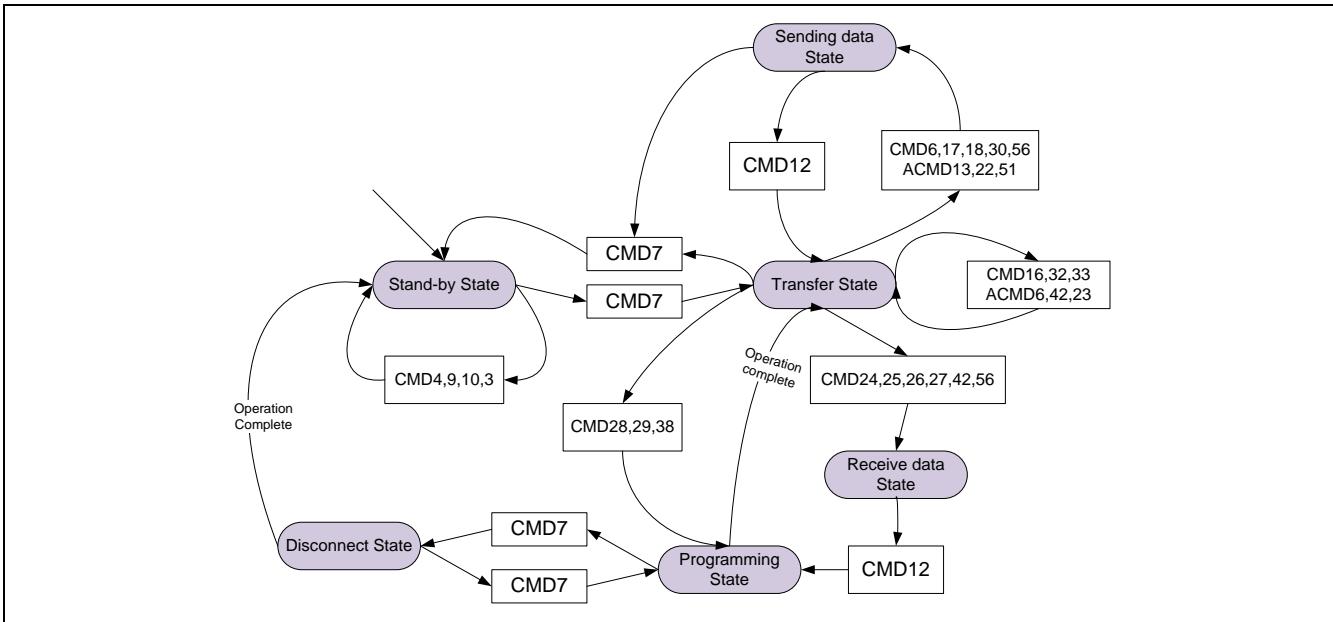
## 22.5 Functional Description

The Secure-Digital Card Host Controller (SDH) equips DMAC unit and SD unit. SDH provides a control interface for SD/SDHC/SDIO/MMC card access. The following sections have more detail description.

SD Memory Card State Diagram (Card Identification Mode) :



SD Memory Card State Diagram (Data Transfer Mode) :



### 22.5.1 Global Control

DMA Controller provides a Direct Memory Access function. After filling in the starting address and enables DMA, DMA would handle the data transfer automatically. There is a 128 bytes shared buffer inside DMA. This 128 bytes buffer is directly accessible when SDH is not in busy.

This SDH controller provides two SD ports – port0 and port1. Each port can provide 1-bit / 4-bit data bus mode, card detect function and SDIO interrupt. User should set the output frequency to SD device by control CLKDIV9 register. About the device detail programming rule, please reference "SD Memory Card Specifications Part 1" and "The MultiMediaCard System Specification".

To enable the SDH, please follow the steps below:

1. Set CLK\_HCLKEN register SDH bit.
2. Set SDH\_DMACTL register DMACEN and DMARST bit.
3. Polling SDH\_DMACTL register DMARST bit until it was cleared.
4. Set SDH\_GCTL register SDEN and GCTRLRST bit.
5. Polling SDH\_GCTL register GCTRLRST bit until it was cleared.
6. Port 0 only has one set of multiple function pin (GPD0~7). Fill value 0x66666666 into SYS\_GPD\_MFPL register to select Port 0.
7. Port 1 has three set of multiple function pin: GPE2~9, GPH6~13 and GPI5~10, 12~13.
  - (1) Set value 0x66666600 into SYS\_GPE\_MFPL register, and 0x66 into

- SYS\_GPE\_MFPH register to select GPE Port 1.
- (2) Set value 0x66000000 into SYS\_GPH\_MFPL register, and 0x666666 into SYS\_GPH\_MFPH register to select GPH Port 1.
  - (3) Set value 0x44400000 into SYS\_GPI\_MFPL register, and 0x00440444 into SYS\_GPI\_MFPH register to select GPI Port 1.
8. Clear SDH\_ECTL register PWROFF0 bit to enable the power control.
  9. Set SDH\_CTL register SDPORT bit to select SD Port 0 or Port 1.
  10. Set SDH\_INTEN register CDxSRC bit to select SD card detection source.(DAT3 or GPIO).
  11. Set SDH initial output frequency is 300 KHz, bus width is 1-bit mode for Card Identification Mode.
  12. Set SDH\_CTL register CLK74\_OE bit.
  13. Polling SDH\_CTL register CLK74\_OE bit until it was cleared.
  14. Follow standard programming rule to send command to SD device.
  15. When device get into Data Transfer Mode, the output frequency can set to suitable clock. Such as 25MHz. And the bus width is 4-bit mode.

### 22.5.2 Send Command

Send command to SD card, please follow the steps below:

1. Set the argument into SDH\_CMD register.
2. Set command into SDH\_CTL register CMD\_CODE bit.
3. Set SDH\_CTL register CO\_EN bit to enable the command out.
4. Polling SDH\_CTL register CO\_EN bit until it was cleared.

### 22.5.3 Get Response

Get response from SD card, please follow the steps below:

1. Set SDH\_CTL register RI\_EN bit to enable response in.
2. Polling SDH\_CTL register RI\_EN bit until it was cleared.
3. Check SDH\_INTSTS register CRC7 bit.
4. Get the response from SDH\_RESP0 and SDH\_RESP1 register.

### 22.5.4 Read SD Card

SD card read access, please follow the steps below:

1. Send CMD7 to enter transfer state.
2. Set SDH\_CTL register CLK8\_OE bit to output 8 clock cycles. Check SDH\_INTSTS register SDDAT0 bit. Repeat step 2 until the SD card is ready.
3. Set block size to SDH\_BLEN register. Such as 0x1FF is for 512 bytes.
4. Set the read starting sector address to SDH\_CMD register.
5. Set the data target address to SDH\_DMASA register.
6. Check the read sector count. If the count is greater than 255, user should separate it. Set the sector count to SDH\_CTL register BLK\_CNT bit. (255 is the limitation).
7. Send CMD18 for multiple read. (Set 18 to SDH\_CTL register CMD\_CODE bit).
8. Set SDH\_CTL register CO\_EN, RI\_EN and DI\_EN bit to enable command out, response in and data in.
9. Polling DI\_EN bit until it was cleared. Or waiting the interrupt (SDH\_INTSTS register BLKD\_IF bit).
10. Check SDH\_INTSTS register CRC7 and CRC16 bit.
11. Send CMD12 to stop transfer.
12. Set SDH\_CTL register CLK8\_OE bit to output 8 clock cycles. Check SDH\_INTSTS register SDDAT0 bit. Repeat step 12 until the SD card is ready.
13. Send CMD7 to Idle state.

### 22.5.5 Write SD Card

SD card write access, please follow the steps below:

1. Send CMD7 to enter transfer state.
2. Set SDH\_CTL register CLK8\_OE bit to output 8 clock cycles. Check SDH\_INTSTS register SDDAT0 bit. Repeat step 2 until the SD card is ready.
3. Set block size to SDH\_BLEN register. Such as 0x1FF is for 512 bytes.
4. Set the write starting sector address to SDH\_CMD register.
5. Set the data source address to SDH\_DMASA register.
6. Check the write sector count. If the count is greater than 255, user should separate it. Set the sector count to SDH\_CTL register BLK\_CNT bit. (255 is the limitation).
7. Send CMD25 for multiple write. (Set 25 to SDH\_CTL register CMD\_CODE bit).
8. Set SDH\_CTL register CO\_EN, RI\_EN and DO\_EN bit to enable command out, response in and data out.

9. Polling DO\_EN bit until it was cleared. Or waiting the interrupt (SDH\_INTSTS register BLKD\_IF bit).
10. Check SDH\_INTSTS register CRC\_IF bit. If CRC error occurred, the state machine should software reset. (Set SDH\_CTL register SW\_RST bit)
11. Send CMD12 to stop transfer.
12. Set SDH\_CTL register CLK8\_OE bit to output 8 clock cycles. Check SDH\_INTSTS register SDDAT0 bit. Repeat step 12 until the SD card is ready.
13. Send CMD7 to Idle state.

## 23 SPI

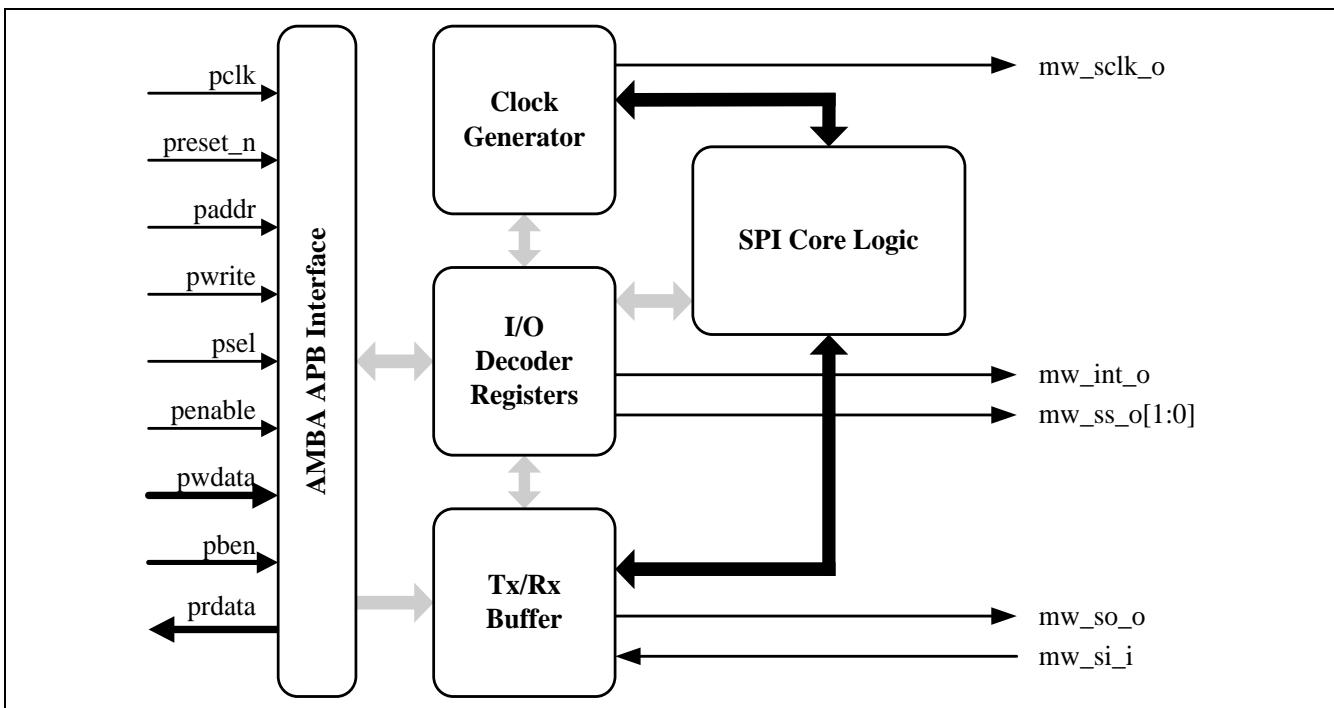
### 23.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial data communication protocol. Devices communicate in Master/Slave mode with 4-wire bi-direction interface. It is used to perform a serial-to-parallel conversion on data received from a peripheral device, and a parallel-to-serial conversion on data transmitted to a peripheral device.

### 23.2 Features

- Supports Master mode operation
- Configurable bit length of a transaction from 1 to 32-bit and can be configured as burst mode, totally 128-bit can be transmitted at one time.
- Supports MSB first or LSB first transfer sequence
- Two slave select lines supported in Master mode
- Support dual / quad mode

### 23.3 Function Block



### 23.4 Register Map

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

SPI\_BA = 0xB800\_6200

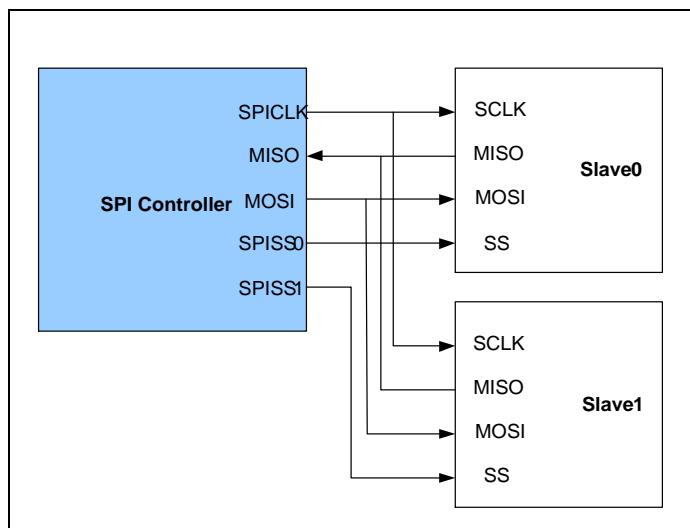
SPI\_BA = 0xB800\_6300

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

## 23.5 Function Description

### 23.5.1 Slave Selection

In Master mode, this SPI controller can drive up to two off-chip slave devices through the slave select output signals SPISS0 and SPISS1, but it is a time-sharing operation and it can not operate with two slave devices simultaneously.



Configure SSR[0] or SSR[1] can control SS0 or SS1 output data.

```

SSR |= 0x1;          //Enable SS0 output pin
SSR |= 0x2;          //Enable SS1 output pin
  
```

```
SSR |= 0x3; //Enable SS0 and SS1 output pin at the same time
```

In master mode, user can configure SS\_LVL(SSR[2]) bit to let SS signal to active at high or low level. The trigger condition is based on type of slave device.

### 23.5.2 Automatic Slave Select

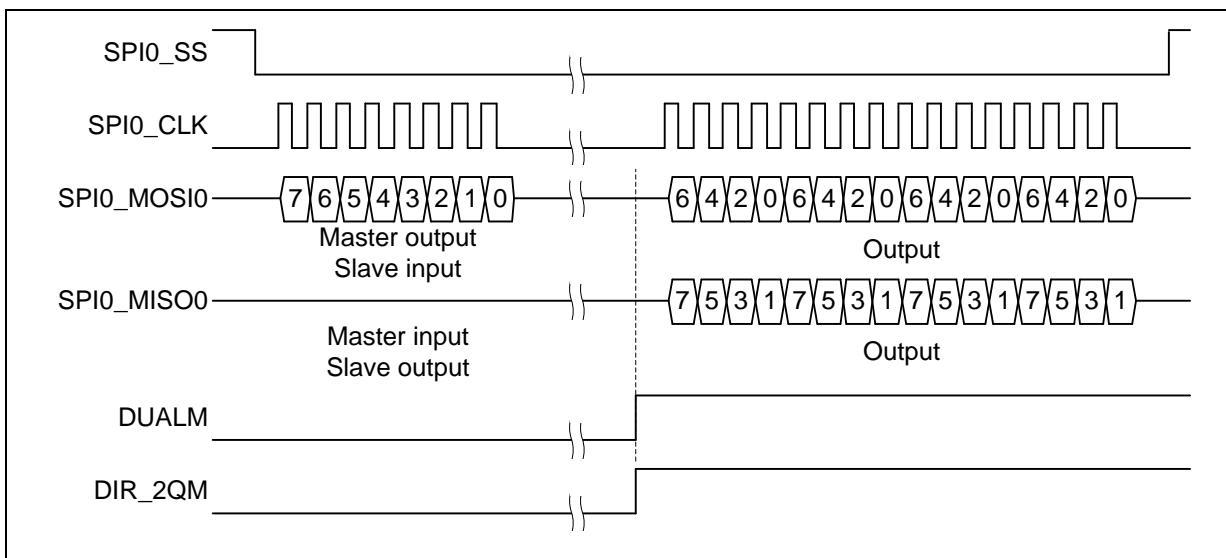
In Master mode, if the AUTOSS bit (SSR[3]) is set as 1, the slave select signals will be generated automatically and output to SPISS0 and SPISS1 ports according to SSR[0] (SSR0) and SSR[1] (SSR1) whether it is enabled or not. It means that the slave select signals, which is enabled in SSR[1:0] register is asserted by the SPI controller when the SPI data transfer is started by setting the GO\_BUSY bit (CNTRL[0]) and is de-asserted after the data transfer is finished.

```
// Enable automatic slave select function on SS0 pin
SSR |= 0x1; //Enable SS0 output pin
SSR |= (0x1 << 3); //Enable automatic slave select function
```

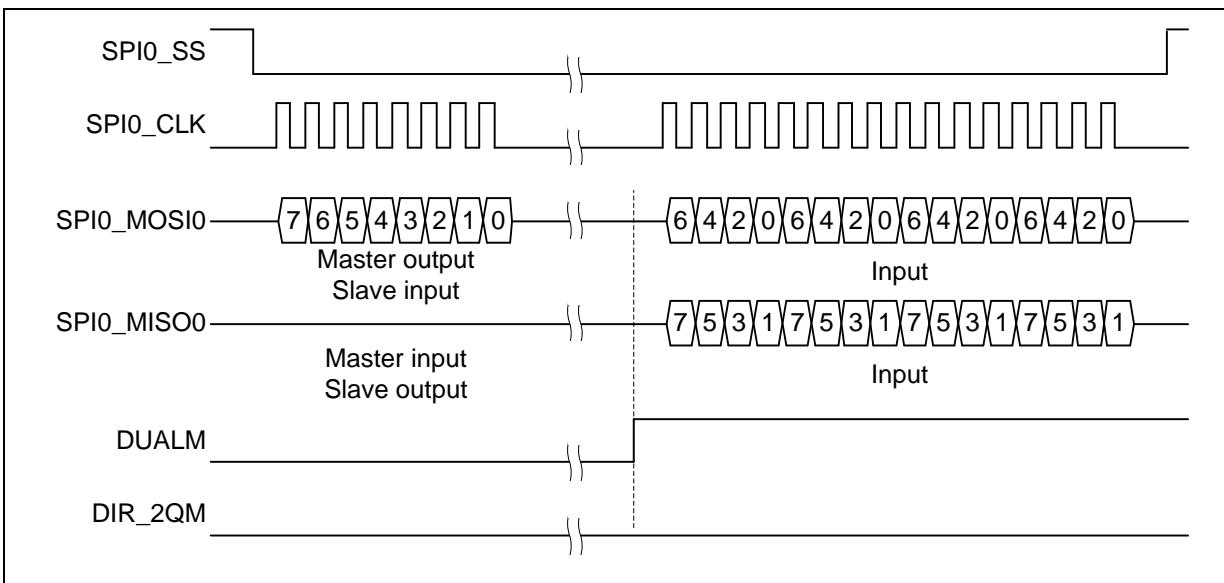
### 23.5.3 Dual / Quad Mode

SPI controller supports dual IO transmit when DUALM (CNTRL[22]) bit is set to 1.

The following figure is dual output mode:



And the following figure is dual input mode:

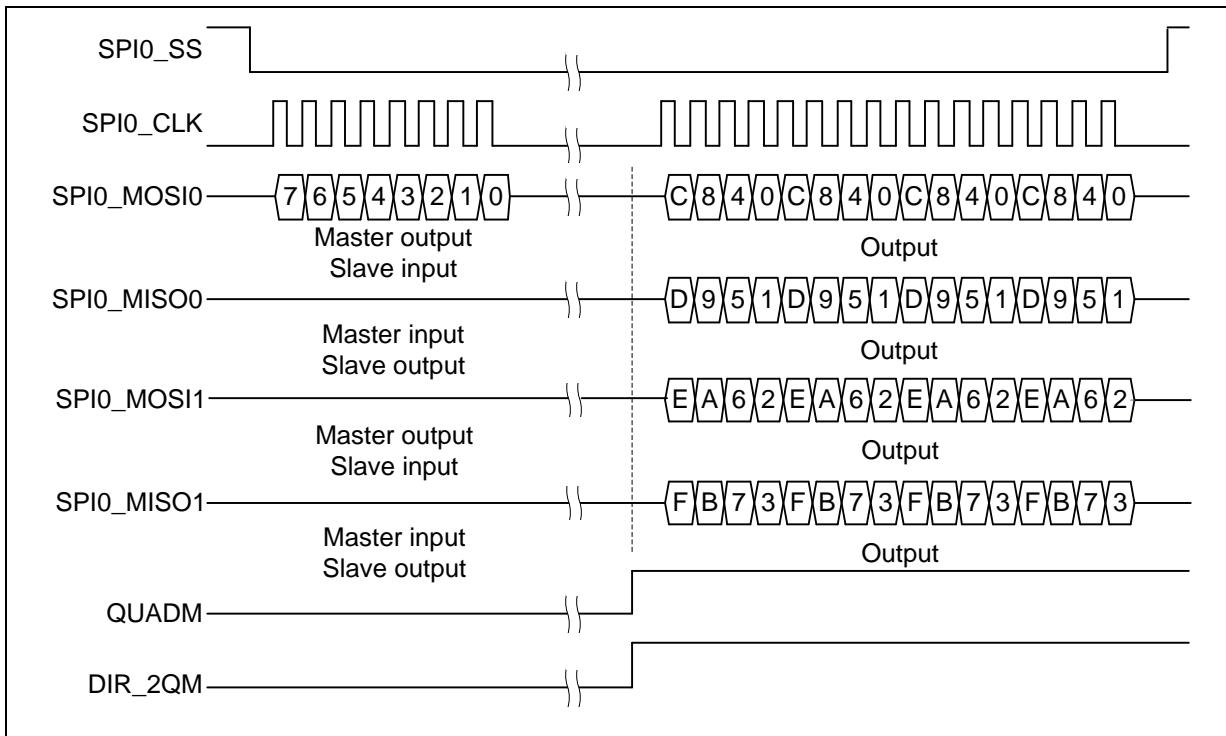


DIR\_2QM(CNTRL[20]) is defined as the direction of data transmission. When DIR\_2QM bit is set to 1, SPI controller will output data to external device, otherwise when DIR\_2QM bit is 0, SPI controller will get the data from external device.

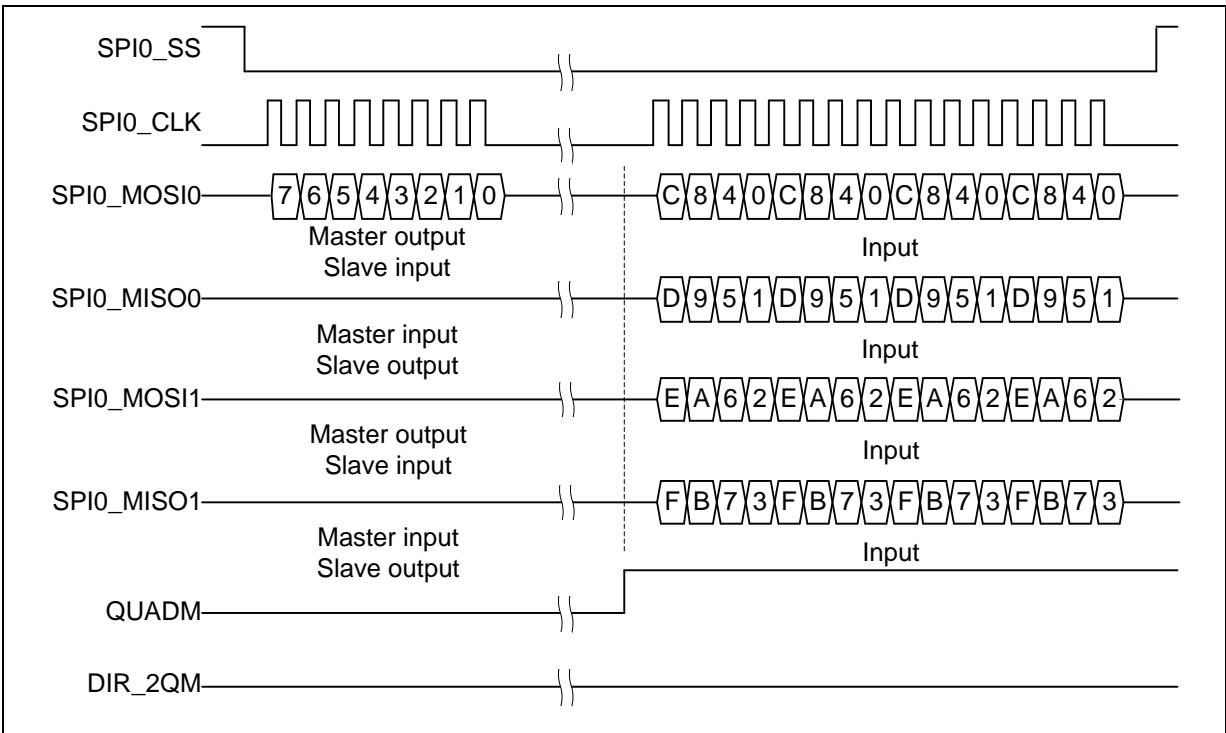
```
//Use dual IO function, MOSI/MISO pin output the data
CNTRL |= (0x1 << 22);      //Enable dual IO mode
CNTRL |= (0x1 << 20);      //Direction is output
TX0 = 0x12;
CNTRL |= 0x1;                //Enable SPI controller
```

SPI controller supports quad IO mode when QUADM(CNTRL[21]) bit is set to 1.

The following figure is quad output mode:



And the following figure is quad input mode:



DIR\_2QM(CNTRL[20]) is defined as the direction of data transmission. When DIR\_2QM bit is set to 1, SPI controller will output data to external device, otherwise when DIR\_2QM bit is 0, SPI controller will get the data from external device.

```
// Use quad IO function, MOSI/MISO pin output the data
CNTRL |= (0x1 << 21);      //Enable quad mode
CNTRL |= (0x1 << 20);      //Direction is output
TX0 = 0x12;
CNTRL |= 0x1;                //Enable SPI controller
```

### 23.5.4 Burst Mode

SPI controller can transfer/receive one to four data at one transfer by configuring TX\_NUM(CNTRL[9:8]).

```
//Transfer four 8-bit data continuously
CNTRL = (CNTRL & ~(0xf8)) | 0x8 <<3;    //Configure 8-bit data width
CNTRL |= (0x3 << 8);                      //Transfer four data
TX0 = 0x12;        //Write the first data
TX1 = 0x34;        //Write the second data
TX2 = 0x56;        //Write the third data
TX3 = 0x78;        //Write the fourth data
CNTRL |= 0x1;      //Enable SPI and transfer 0x12, 0x34, 0x56, 0x78 continuously
```

### 23.5.5 SPI Interrupt

The interrupt flag IF(CNTRL[16]) bit will be set to 1 after SPI controller finished transmit or receive. If interrupt enable bit IE(CNTRL[17]) is also set to 1 and interrupt will occur. IF bit can be cleared by writing 1 to itself.

```
CNTRL |= 0x20000;      //Enable interrupt
CNTRL |= 0x1;          //Enable SPI
while(!spi_isr);       //Wait for interrupt
CNTRL |= 0x10000;      //Clear IF bit
```

If IE bit doesn't be set to 1, user still can poll GO\_BUSY bit to check SPI controller finishes transmit or not.

```
CNTRL |= 0x1;          //Enable SPI
while(CNTRL & 0x1);    //Wait for SPI's job is done
```

### 23.5.6 SPI Programming Example

Do following actions basically (Should refer to the specification of device for the detailed steps):

1. Write a divisor into DIVIDER to determine the frequency of serial clock.
2. Write in SSR, set ASS = 0, SS\_LVL = 0 and SSR[0] or SSR[1] to 1 to activate the device you want to access.
3. When transmit (write) data to device:

4. Write the data you want to transmit into Tx0[7:0].
5. When receive (read) data from device:
6. Write 0xFFFFFFFF into Tx0.
7. Write in CNTRL, set Rx\_NEG = 0, Tx\_NEG = 1, Tx\_BIT\_LEN = 0x08, Tx\_NUM = 0x0, LSB = 0, SLEEP = 0x0 and GO\_BUSY = 1 to start the transfer.  
Wait for interrupt (if IE = 1) or polling the GO\_BUSY bit until it turns to 0.
8. Read out the received data from Rx0.
9. Go to step 3 to continue data transfer or set SSR[0] or SSR[1] to 0 to deactivate the device

## 24 TIMER CONTROLLER

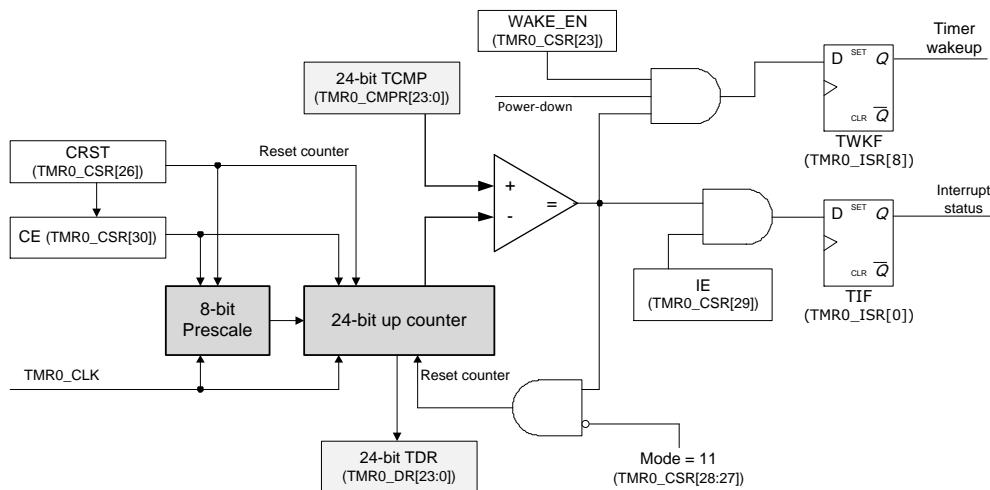
### 24.1 Overview

The general timer controller includes five channels, TIMER0, TIMER1, TIMER2, TIMER3, and TIMER4, which allow user to easily implement a counting scheme or timing control for applications. The timer can perform functions like frequency measurement, event counting, interval measurement, pulse generation, delay timing, and so on. The timer possesses features such as adjustable resolution, programmable counting period, and detailed information. The timer can generate an interrupt signal upon timeout, or provide the current value of count during operation.

### 24.2 Features

- Independent Clock Source for each Timer channel (TMRx\_CLK, x= 0, 1, 2, 3, 4).
- Five channels with a 24-bit up counter and an interrupt request each.
- Internal 8-bit pre-scale counter.
- Internal 24-bit up counter is readable through Timer Data Register, TDR (TMR\_DR[23:0]).
- Supports One-shot, Periodic, and Continuous operation mode.
- Time-out period = (Period of timer clock input) \* (8-bit pre-scale counter + 1) \* (24-bit TCMP setting value).
- Maximum counting time =  $(1 / \text{TMRx\_CLK}) * 256 * 2^{24}$

### 24.3 Block Diagram



## 24.4 Register Map

Register	Offset	R/W	Description	Reset Value
<b>TMR Base Address:</b>				
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_CMNR	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_CMNR	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_CMNR	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_CMNR	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_CMNR	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

## 24.5 Functional Description

### 24.5.1 Timer Initialization

Timer can be initialized using following procedure and start timer counting.

1. Clear CE (TMRx\_CSR[30]) to 0 to stop timer counting.
2. Set MODE (TMRx\_CSR[28:27]) to configure timer operating mode
3. Set IE (TMRx\_CSR[29]) to 1 to enable interrupt, otherwise clear to 0.

4. Set prescaler in PRESCALE (TMRx\_CSR[7:0])
5. Set timer compare value in TCMP (TMRx\_CMPR[24:0])
6. Set CE (TMRx\_CSR[30]) 1 to start timer up counting.

#### 24.5.2 Interrupt Handling

Each timer has individual timeout interrupt source, when timer interrupt triggered, TMR\_ISR[4:0] corresponding bit will be set 1. Bit 0 set 1 means TMR0 interrupt triggered, bit 1 set 1 means TMR1 interrupt triggered, bit 2 set 1 means TMR2 interrupt triggered... so on so forth. Writing 1 to them can clear these bits.

Please note if IE (TMRx\_CSR[29]) is cleared to 0 then TMR\_ISR relative bits will not set to 1 even timeout event occurs. So if software wants to check timeout event using polling mode, IE bit still needs to set 1 to check TMR\_ISR status. Timer will not trigger as long as timer interrupt is not enabled in AIC.

#### 24.5.3 Timeout Frequency

The timeout frequency can be calculated using formula below:

$$\text{Frequency} = \text{TMRx\_CLK} / ((\text{PRESCALE} + 1) * \text{TCMP})$$

Where TMRx\_CLK is the timer clock source frequency, PRESCALE is the prescaler defined in TMRx\_CSR[7:0], TCMP compare value defined in TMRx\_CMPR[24:0]. So using external 12MHz crystal as timer clock source, fastest and slowest frequency timer can generate are  $12\text{MHz} / ((0 + 1) * 1)$  and  $12\text{MHz} / ((0xFF + 1) * 0xFFFFFFF)$ . Following table list some possible setting for generating 10Hz, 100Hz, 1000Hz timeout frequency while using 12MHz as clock source.

Frequency	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

#### 24.5.4 One-shot Mode

If the timer is operated in One-shot mode when MODE (TMRx\_CSR[28:27]) is 0x0 and the timer counter enable bit CE (TMRx\_CSR[30]) is set to 1, the timer counter starts up counting. Once the timer counter value TDR (TMRx\_DR[23:0]) reaches timer compare register TCMP (TMRx\_CMPR[23:0]) value, the timer interrupt signal will be asserted. If the timer interrupt signal is asserted and the timer interrupt enable bit IE (TMRx\_CSR[29]) is set to 1, the timer interrupt flag TIF (TMRx\_ISR[0]) will be asserted by hardware and then software can write 1 into TIF (TMRx\_ISR[0]) bit to clear it.

In this operating mode, once the timer counter value TDR (TMRx\_DR[23:0]) reaches timer compare register TCMP (TMRx\_CMPR[23:0]) value will set the timer interrupt flag TIF (TMRx\_ISR[0]) to 1, then timer counting operation stops and the timer counter value TDR (TMRx\_DR[23:0]) goes back to counting initial value then timer counter enable bit CE (TMRx\_CSR[30]) is cleared to 0 by timer controller automatically. That is to say, timer operates timer counting and compares with TCMP (TMRx\_CMPR[23:0]) value function only one time after programming the timer compare register TCMP (TMRx\_CMPR[23:0]) value and timer counter enable bit CE (TMRx\_CSR[30]) is set to 1. So, this operating mode is called One-shot mode.

#### 24.5.5 Periodic Mode

If the timer is operated in Periodic mode (MODE (TMRx\_CSR[28:27]) is 0x1) and timer counter enable bit CE (TMRx\_CSR[30]) is set to 1, the timer counter starts up counting. Once the timer counter value (TMRx\_DR) reaches timer compare register (TMRx\_CMPR) value, the interrupt signal will be asserted then timer interrupt flag TIF (TMRx\_ISR[0]) will set to 1 if timer interrupt enable bit IE=0 (TMRx\_CSR[29]=1). If IE (TMRx\_CSR[29]) is set to 0, the timer interrupt flag TIF (TMRx\_ISR[0]) will not be set to 1.

In this operating mode, once the timer counter value TDR (TMRx\_DR[23:0]) reaches timer compare register TCMP (TMRx\_CMPR[23:0]) value and IE (TMRx\_CSR[29]) set to 1, timer interrupt flag TIF (TMRx\_ISR[0]) will set to 1, the timer counter value TDR (TMRx\_DR[23:0]) goes back to counting initial value and timer counter enable bit CE (TMRx\_CSR[30]) is kept at 1 (counting enable Periodically) and timer counter operates up counting again. If timer interrupt flag TIF (TMRx\_ISR[0]) is cleared by software, once the timer counter value TDR (TMRx\_DR[23:0]) reaches timer compare register TCMP (TMRx\_CMPR[23:0]) value again, TIF (TMRx\_ISR[0]) will set to 1 also. That is to say, timer operates timer counting and compares with TCMP (TMRx\_CMPR[23:0]) value function periodically. The timer counting operation does not stop until the timer counter enable bit CE (TMRx\_CSR[30]) is set to 0. The interrupt signal is also generated periodically. So, this operating mode is called Periodic mode.

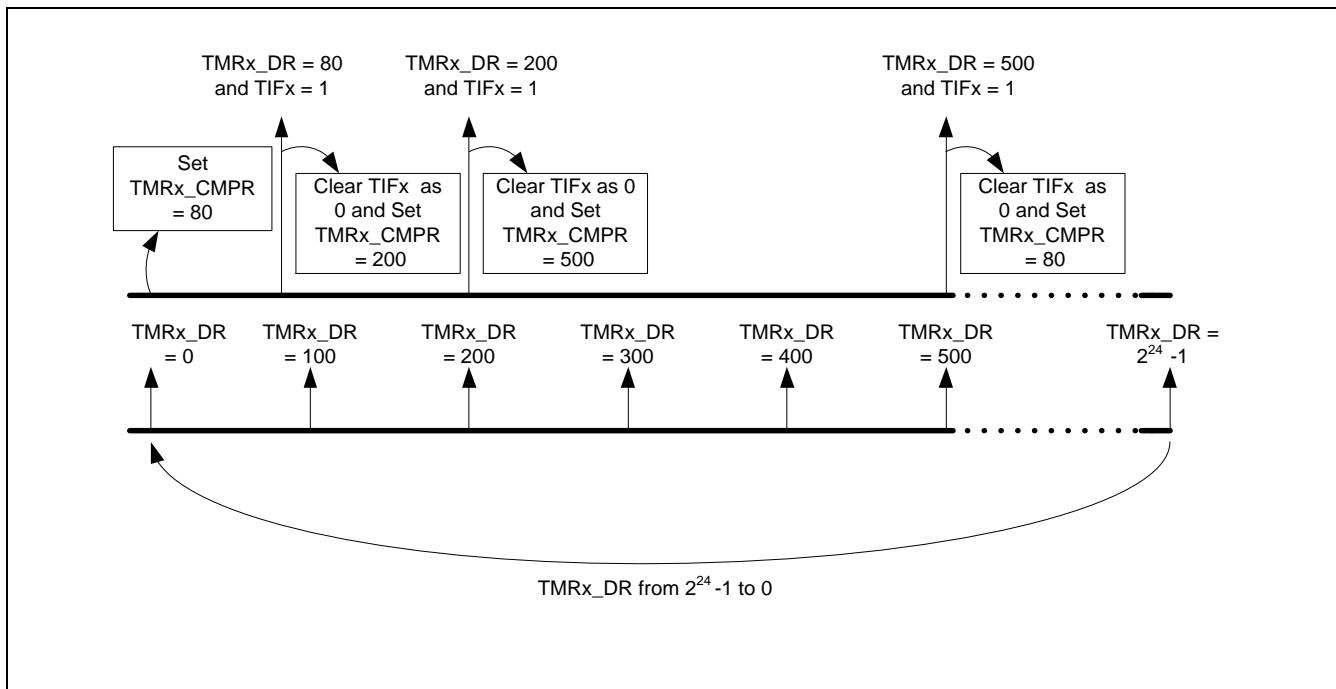
#### 24.5.6 Continuous Mode

If the timer is operated in Continuous mode (MODE (TMRx\_CSR[28:27]) is 0x3) and timer counter enable bit CE (TMRx\_CSR[30]) is set to 1, the timer counter starts up counting. The 24-bit timer counter in Continuous mode will keep counting until the counting value reaches to the maximum value (0xFFFFFFF), then timer counter value TDR (TMRx\_DR[23:0]) will be

reset to 0x0 and keep up counting while CE=1 (TMRx\_CSR[30]=1).

Once the timer counter value TDR (TMRx\_DR[23:0]) is equal to the timer compare register TCMP (TMRx\_CMPR[23:0]) value and timer interrupt enable bit IE (TMRx\_CSR[29]) is set to 1, the timer interrupt flag TIF (TMRx\_ISR[0]) will set to 1. If IE (TMRx\_CSR[29]) is set to 0, TIF (TMRx\_ISR[0]) will not be asserted. In Continuous mode, changing the value of compare register TCMP (TMRx\_CMPR[23:0]) will not affect the current value of TDR (TMRx\_DR[23:0]), but it will clear timer counter value TDR (TMRx\_DR[23:0]) to 0x0 in other operation modes. User can change the compare register TCMP (TMRx\_CMPR[23:0]) at any time and the timer counter will keep up counting continuously to generate the new interrupt event. So, this operating mode is called Continuous mode.

Following figure shows a Timer continuous mode sample.



## 25 UART

### 25.1 Overview

The Universal Asynchronous Receiver/Transmitter (UART) performs a serial-to-parallel conversion on data received from the peripheral, and a parallel-to-serial conversion on data transmitted from the CPU.

Each UART channel supports 7 types of interrupts including (1). transmitter FIFO empty interrupt (INT\_THRE), (2). receiver threshold level reaching interrupt (INT\_RDA), (3). line status interrupt (parity error or framing error or break interrupt) (INT\_RLS), (4). receiver buffer time-out interrupt (INT\_TOUT), (5). MODEM/Wake-up status interrupt (INT\_MODEM), (6). Buffer error interrupt (INT\_BUF\_ERR), and (7). LIN interrupt (INT\_LIN).

The UART1/ 2/ 4/ 6/ 8/ 10 is built-in with a 64-byte transmitter FIFO (TX\_FIFO) and a 64-byte receiver FIFO (RX\_FIFO) that reduces the number of interrupts presented to the CPU and the UART0/ 3/ 5/ 7/ 9 are equipped 16-byte transmitter FIFO (TX\_FIFO) and 16-byte receiver FIFO (RX\_FIFO). The CPU can read the status of the UART at any time during the operation. The reported status information includes the type and condition of the transfer operations being performed by the UART, as well as 4 error conditions (parity error, framing error, break interrupt and buffer error) probably occur while receiving data.

The UART Controller supports wake-up system function. In power condition, when the WAKE\_CTS\_EN(UA\_CTL[8]) is set and the toggle of CTSn pin can wake-up the system.

The UART Controller includes a programmable baud rate generator capable of dividing clock input by dividers to produce the serial clock that transmitter and receiver need.

The baud rate equation is: Baud Rate =  $\text{UART\_CLK} / M * [\text{BRD} + 2]$

where M and BRD are defined in Baud Rate Divider Register (UA\_BAUD).

The following tables list the UART baud rate equations in the various conditions and UART baud rate parameter settings.

Mode	DIV_X_EN	DIV_X_ONE	DIVIDER X	BRD	Baud Rate Equation
0	Disable	0	Don't Care	A	$\text{UART\_CLK} / [16 * (A+2)]$
1	Enable	0	B	A	$\text{UART\_CLK} / [(B+1) * (A+2)]$ , B must $\geq 8$
2	Enable	1	Don't care	A	$\text{UART\_CLK} / (A+2)$ , A must $\geq 9$

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

The UART controllers support auto-flow control function that uses two low-level signals, CTSn (clear-to-send) and RTSn (request-to-send) to control the flow of data transfer between the UART and external devices (ex: Modem). When auto-flow is enabled, the UART is not allowed to receive data until the UART asserts RTSn (RTSn high) to external device. When the number of bytes in the RX-FIFO equals the value of RTS\_TRI\_LEV (UA\_FCR [19:16]), the RTSn is de-asserted. The UART sends data out when UART controller detects CTSn is asserted (CTSn high) from external device. If a valid asserted CTSn is not detected the UART controller will not send data out.

The UART controllers also provides Serial IrDA (SIR, Serial Infrared) function (The IrDA mode is selected by setting the (FUN\_SEL(UA\_FUN\_SEL[2:0]) = 010) to select IrDA function). The SIR specification defines a short-range infrared asynchronous serial transmission mode with one start bit, 8 data bits, and 1 stop bit. The maximum data rate is 115.2 Kbps (half duplex). The IrDA SIR block contains an IrDA SIR Protocol encoder/decoder. The IrDA SIR protocol is half-duplex only. So it cannot transmit and receive data at the same time. The IrDA SIR physical layer specifies a minimum 10ms transfer delay between transmission and reception. This delay feature must be implemented by software.

For the NUC970/N9H30 series, another alternate function of UART controllers is RS-485 9-bit mode function, and direction control provided by RTS pin to implement the function by software. The RS-485 mode is selected by setting the (FUN\_SEL(UA\_FUN\_SEL[2:0]) = 011) to select RS-485 function. The RS-485 driver control is implemented using the RTS control signal from an asynchronous serial port to enable the RS-485 driver. In RS-485 mode, many characteristics of the RX and TX are the same as UART.

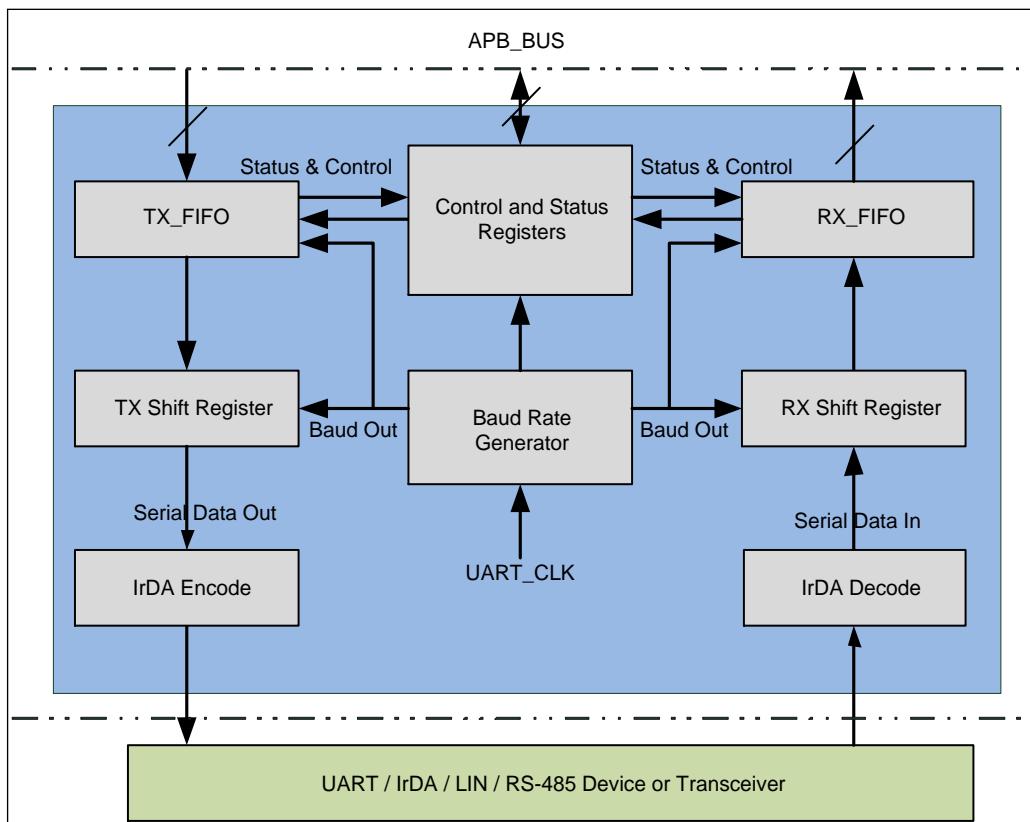
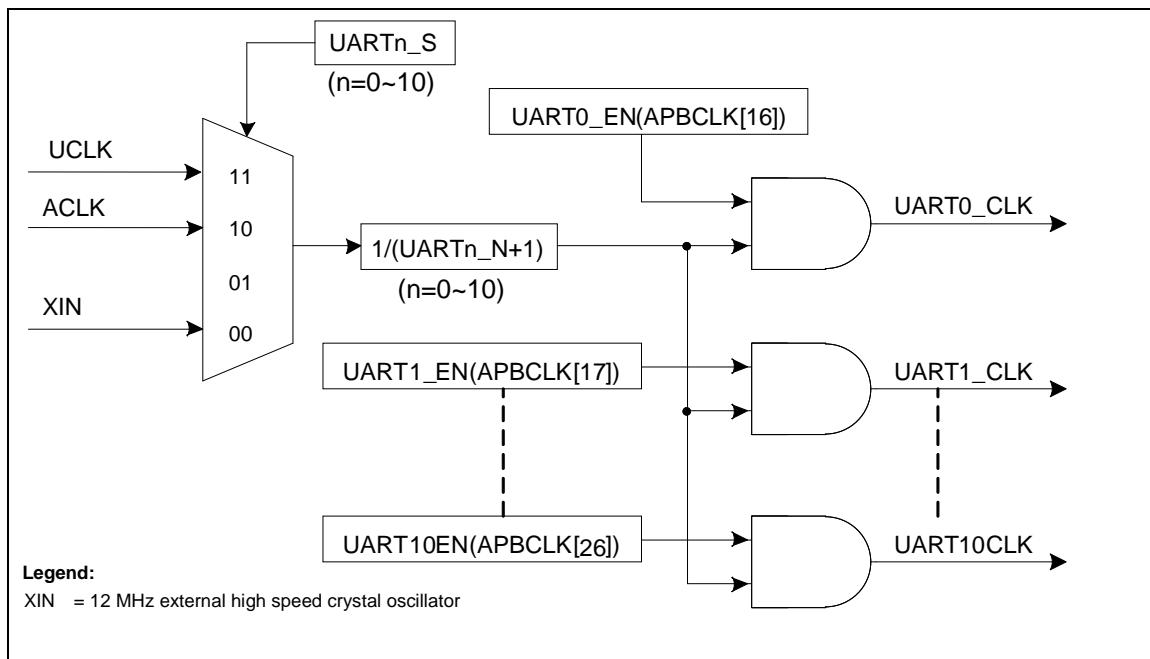
The alternate function of UART controllers is LIN (Local Interconnect Network) function. The LIN mode is selected by setting the (FUN\_SEL(UA\_FUN\_SEL[2:0]) = 001) to select LIN

mode. In LIN mode, one start bit and 8-bit data format with 1-bit stop bit are required in accordance with the LIN standard.

## 25.2 Features

- Full duplex, asynchronous communications
- Separate receive / transmit 64/16 bytes entry FIFO for data payloads
- Supports hardware auto flow control/flow control function (CTS, RTS) and programmable RTS flow control trigger level
- Programmable receiver buffer trigger level
- Supports programmable baud-rate generator for each channel individually
- Supports CTS wake-up function
- Supports 8-bit receiver buffer time-out detection function
- Programmable transmitting data delay time between the last stop and the next start bit by setting DLY(UA\_TOR[15:8]) register
- Supports break error, frame error, parity error and receive / transmit buffer overflow detect function
- Fully programmable serial-interface characteristics
- Programmable number of data bit, 5-, 6-, 7-, 8-bit character
- Programmable parity bit, even, odd, no parity or stick parity bit generation and detection
- Programmable stop bit, 1, 1.5, or 2 stop bit generation
- Supports IrDA SIR function mode
- Supports LIN function mode
- Supports RS-485 function mode

## 25.3 Block Diagram



## 25.4 Register Map

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  
 ChannelA : UART10\_BA (High-speed) = 0xB800\_0A00

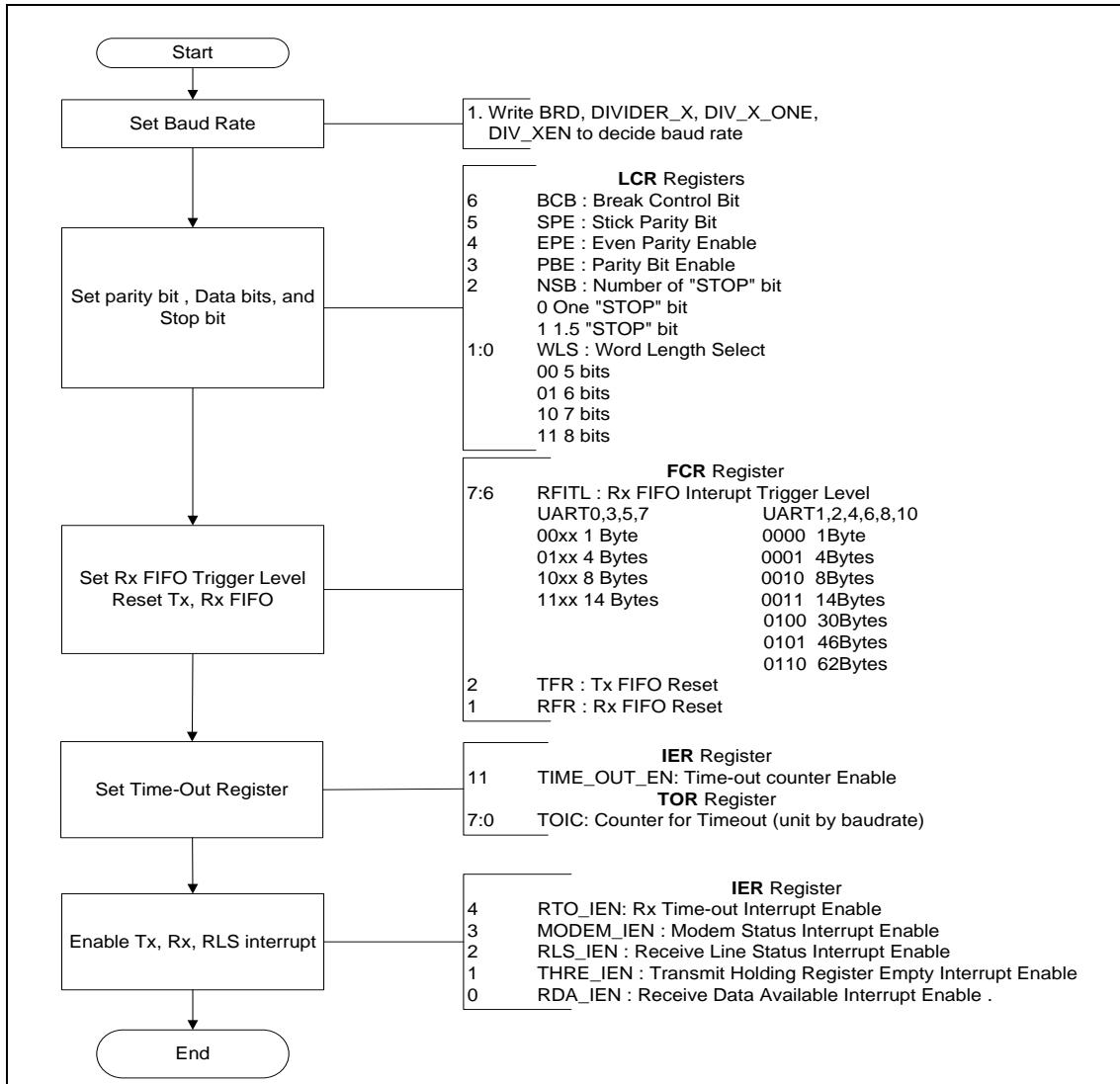
<b>UA_RBR</b>	UART_BA+0x00	R	UART Receive Buffer Register	Undefined
<b>UA_THR</b>	UART_BA+0x00	W	UART Transmit Holding Register	Undefined
<b>UA_IER</b>	UART_BA+0x04	R/W	UART Interrupt Enable Register	0x0000_0000
<b>UA_FCR</b>	UART_BA+0x08	R/W	UART FIFO Control Register	0x0000_0000
<b>UA_LCR</b>	UART_BA+0x0C	R/W	UART Line Control Register	0x0000_0000
<b>UA_MCR</b>	UART_BA+0x10	R/W	UART Modem Control Register	0x0000_0000
<b>UA_MSR</b>	UART_BA+0x14	R/W	UART Modem Status Register	0x0000_0000
<b>UA_FSR</b>	UART_BA+0x18	R/W	UART FIFO Status Register	0x1040_4000
<b>UA_ISR</b>	UART_BA+0x1C	R/W	UART Interrupt Status Register	0x0000_0002
<b>UA_TOR</b>	UART_BA+0x20	R/W	UART Time-out Register	0x0000_0000
<b>UA_BAUD</b>	UART_BA+0x24	R/W	UART Baud Rate Divisor Register	0x0F00_0000
<b>UA_IRCR</b>	UART_BA+0x28	R/W	UART IrDA Control Register	0x0000_0040
<b>UA_ALT_CSR</b>	UART_BA+0x2C	R/W	UART Alternate Control/Status Register	0x0000_000C
<b>UA_FUN_SEL</b>	UART_BA+0x30	R/W	UART Function Select Register	0x0000_0000
<b>UA_LIN_CTL</b>	UART_BA+0x34	R/W	UART LIN Control Register	0x000C_0000
<b>UA_LIN_SR</b>	UART_BA+0x38	R/W	UART LIN Status Register	0x0000_0000
<b>UA_SC_CTL</b>	UART_BA+0x40	R/W	UART SC Control Register	0x0000_0000
<b>UA_SC_FSR</b>	UART_BA+0x44	R/W	UART SC Flag Status Register	0x0000_0000

## 25.5 Functional Description

### 25.5.1 Initializations

Before the transfer operation starts, the serial interface of UART must be programmed. The

driver should set the baud rate, parity bit, data bit and stop bit. If the transfer operation is done triggered by interrupt, the TX, RX and RLS interrupts need to be enabled.



### 25.5.2 IrDA Mode

The UART Controller provides Serial IrDA (SIR, Serial Infrared) transmit encoder and receive decoder function. The IrDA\_EN(UART\_FUN\_SEL[2:0] = 010) bit are used to select IrDA function.

In IrDA Operation mode, the receive FIFO trigger level must be “1” by setting RFITL(UA\_FCR[7:4]) = 0000 and the DIV\_X\_EN(UA\_BAUD[29]) bit must be disabled in IrDA mode operation (Mode 1).

Baud Rate = Clock / (16 \* BRD ), where BRD is Baud Rate Divider in BRD(UA\_BAUD[15:0]).

The IrDA SIR Encoder/Decoder provides functionality which converts between UART data stream and half duplex serial SIR interface.

Programming Sequence Example:

1. Set IrDA\_EN(UART\_FUN\_SEL[2:0] = 010) =1, to select IrDA function.
2. Set INV\_TX(UA\_IRCR[5]) = 0. (Not inverse TX output signal)
3. Set INV\_RX(UA\_IRCR[6]) = 1. (Inverse RX input signal)
4. Setting TX\_SELECT (UA\_IRCR [2]) to select half- tandem as TX or RX.
  - TX\_SELECT(UA\_IRCR[2]) = 1 → select TX.
  - TX\_SELECT(UA\_IRCR[2]) = 0 → select RX.

### 25.5.3 RS485 Function Mode

The UART supports RS-485 9-bit mode function. The RS-485 mode is selected by setting the FUN\_SEL(UA\_FUN\_SEL[2:0]) to select RS-485 function. The RS-485 driver control is implemented using the RTS control signal from an asynchronous serial port to enable the RS-485 driver. In RS-485 mode, many characteristics of the RX and TX are same as UART.

In RS-485 mode, the bit 9 will be configured as address bit. The controller can configuration of it as an RS-485 addressable slave and the RS-485 master transmitter will identify an address character by setting the parity (9th bit) to 1.

For data characters, the bit 9 is set to “0”. Software can use UA\_LCR register to control the 9-th bit (When the PBE(UA\_LCR[3]), EPE(UA\_LCR[4]) and SPE(UA\_LCR[5]) are set, the 9-th bit is transmitted 0 and when PBE and SPE are set and EPE is cleared, the 9-th bit is transmitted 1).

The Controller support three operation mode that is RS-485 Normal Multi-drop Operation Mode (NMM), RS-485 Auto Address Detection Operation Mode (AAD) and RS-485 Auto Direction Control Operation Mode (AUD), software can choose any operation mode by programming UA\_ALT\_CSR register, and software can driving the transfer delay time between the last stop bit leaving the TX-FIFO and the de-assertion of by setting DLY(UA\_TOR [15:8]).

#### 25.5.3.1 RS-485 Normal Multidrop Operation Mode (NMM)

In RS-485 Normal Multi-drop operation mode, software must decide whether receiver will ignore data before an address byte is detected (bit 9 = “1”).

If software wants to receive any data before address byte detected, the flow is disable RX\_DIS(UA\_FCR [8]) then enable RS485\_NMM(UA\_ALT\_CSR[8]) and the receiver will received any data. If an address byte is detected (bit9 =1), it will generator an interrupt to CPU and software can decide whether enable or disable receiver to accept the following data byte by setting RX\_DIS.

When an address byte be detected (bit 9 = “1”) by hardware, the address byte data will be stored in the RX-FIFO. If the receiver is be enabled (RX\_DIS(UA\_FCR[8]) is low, all received byte data will be accepted and stored in the RX-FIFO, and if the receiver is disabled (RX\_DIS(UA\_FCR[8]) is high, all received byte data will be ignore until the next address byte be detected.

If software disable receiver by setting (RX\_DIS(UA\_FCR[8]) bit, when a next address byte be detected, the controller will clear the RX\_DIS bit and the address byte data will be stored in the RX-FIFO.

Program Sequence Example :

1. Program FUN\_SEL(UA\_FUN\_SEL[2:0]) to select RS-485 function.
2. Program the RX\_DIS(UA\_FCR[8]) bit to determine whether to store the received data before an address byte is detected (bit 9 = “1”).
3. Program the RS485\_NMM by setting RS485\_NMM(UA\_ALT\_CSR[8]).
4. When an address byte is detected (bit 9 = “1”), hardware will set RLS\_IS(UA\_ISR[2]) and RS485\_ADD\_DETF(UA\_FSR[3]) flag.
5. Software can decide whether to accept the following data byte by setting RX\_DIS(UA\_FCR[8]).
6. Repeat step 4 and step 5.

#### 25.5.3.2 RS-485 Auto Address Detection Operation Mode (AAD)

In RS-485 Auto Address Detection Operation Mode, the receiver will ignore any data until an address byte is detected (bit9 =1) and the address byte data match the ADDR\_MATCH(UA\_ALT\_CSR[31:24]) value. The address byte data will be stored in the RX-FIFO. The all received byte data will be accepted and stored in the RX-FIFO until and address byte data not match the ADDR\_MATCH(UA\_ALT\_CSR[31:24]) value. In RS-485 AAD mode, don“t fill any value to RX\_DIS(UA\_CTL[2]) bit.

Program Sequence example :

1. Program FUN\_SEL(UART\_FUN\_SEL[1:0]) to select RS-485 function.
2. Program the RS485\_AAD(UA\_ALT\_CSR[9]).
3. When an address byte is detected (bit9 = “1”), hardware will compare the address byte and the ADDR\_MATCH (UA\_ALT\_CSR[31:24]) value.
4. If the address byte matches the ADDR\_MATCH(UA\_ALT\_CSR[31:24]) value, hardware will set RLS\_IS(UART\_ISR[2]) and RS485\_ADD\_DETF(UA\_FSR[3]). And the receiver will sorted address byte to FIFO and accept the following data transfer and stored data in FIFO until next address byte be detected.

5. However if the address byte does not match the ADDR\_MATCH(UA\_ALT\_CSR[31:24]) value, hardware will ignore the address byte data and ignore the following data transfer.
6. Respect step 3 and step 4.

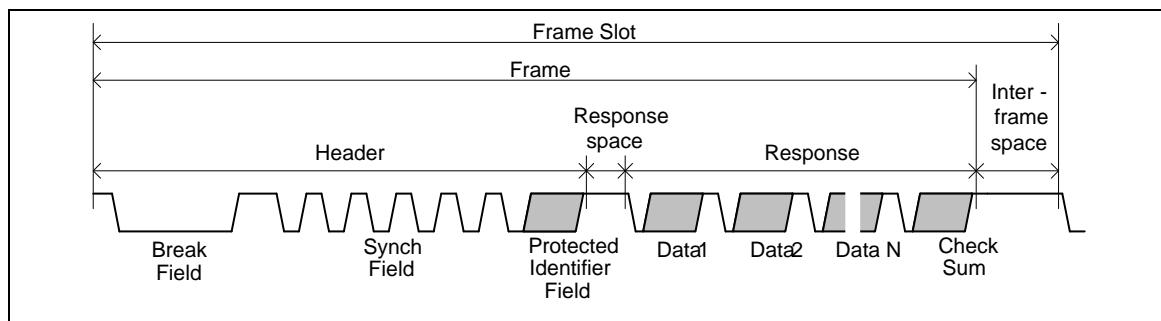
#### 25.5.3.3 RS-485 Auto Direction Mode (AUD)

Another option function of RS-485 controllers is RS-485 auto direction control function. The RS-485 driver control is implemented using the RTS control signal from an asynchronous serial port to enable the RS-485 driver. The RTS line is connected to the RS-485 driver enable such that setting the RTS line to high (logic 1) enables the RS-485 driver. Setting the RTS line to low (logic 0) puts the driver into the tri-state condition. User can set LEV\_RTS(UA\_MCR[9]) to change the RTS driving level.

#### 25.5.4 LIN (Local Interconnection Network) Mode

The UART supports LIN function. The LIN mode is selected by setting the (FUN\_SEL(UA\_FUN\_SEL[2:0]) = 001).

According to the LIN protocol, all information is transmitted packed as frames; a frame consists (provided by the master task) a header and a response (provided by a slave task). That is any communication on the LIN bus is started by the master sending a header, followed by the response. The header (provided by the master task) consists of a break field and sync field followed by a frame identifier (frame ID). The frame identifier uniquely defines the purpose of the frame. The slave task appointed for providing the response associated with the frame ID and the response consists of a data field and a checksum field. The following diagram is the structure of LIN function mode.



#### LIN Transmission (TX) Program Sequence :

1. Select LIN function mode by setting UA\_FUN\_SEL register
2. Select Break field and Delimiter Length by setting LIN\_BKFL(UA\_LIN\_CTL[19:16]) and LIN\_BS\_LEN(UA\_LIN\_CTL[21:20]).

3. Set LIN\_TX\_EN(UA\_ALT\_CSR[8]) to start transfer. (When transmitter header field (it may be “break” or “break + sync” or “break + sync + frame ID” selected by LIN\_HEAD\_SEL(UA\_LIN\_CTL[23:22]) field) transfer operation finished, this bit will be cleared automatically).
4. Request sync field transmission by writing 0x55 into UA\_THR register.
5. Request header frame ID transmission by writing the protected identifier value in the UA\_THR register.
6. Wait for the TE\_FLAG(UA\_FSR[28]) flag
7. Write N bytes data and checksum value to UA\_THR register. Repeat step 5 and step 6.

LIN Receive (RX) Program Sequence :

1. Select LIN function mode by setting UA\_FUN\_SEL register
2. Set LIN\_TX\_EN(UA\_ALT\_CSR[8]) = 1 to enable LIN RX mode
3. Wait LIN\_BKDET\_F(UA\_LIN\_SR[8]) flag. (This bit is set by hardware when a break is detected).
4. Wait for the RDA\_IF(UA\_ISR[0]) flag and read back the UA\_RBR register

## 26 USB 2.0 Device Controller

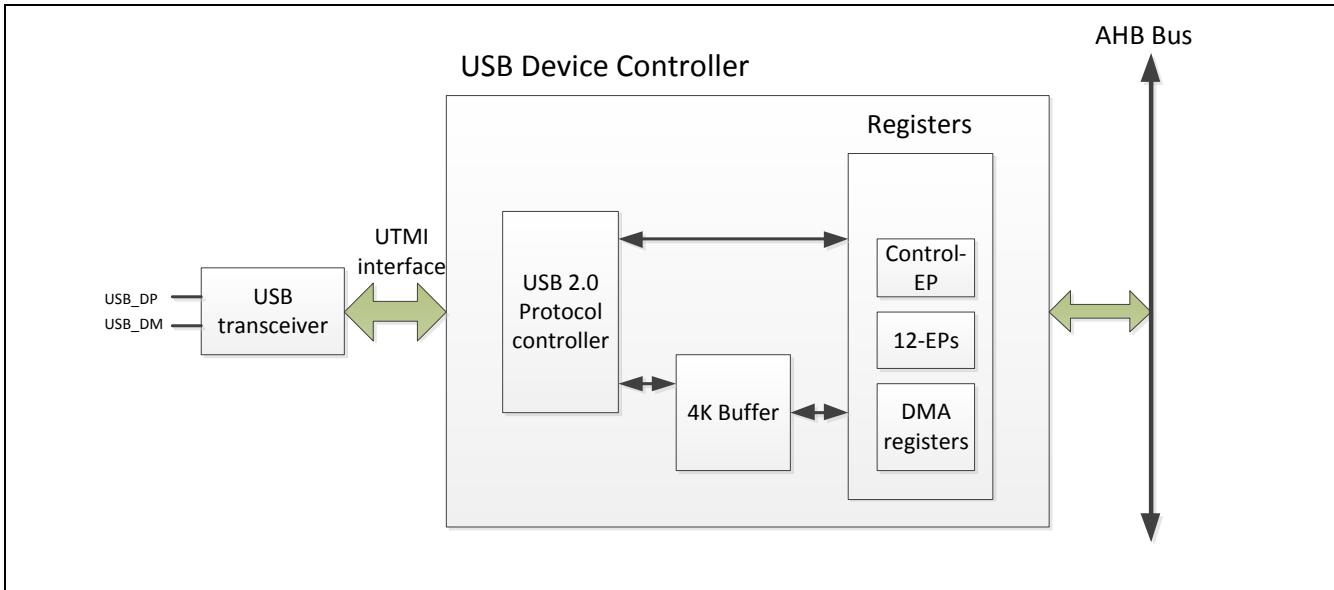
### 26.1 Overview

The USB device controller interfaces the AHB bus and the UTMI bus. The USB controller contains both the AHB master interface and AHB slave interface. CPU programs the USB controller registers through the AHB slave interface. For IN or OUT transfer, the USB device controller needs to write data to memory or read data from memory through the AHB master interface. The USB device controller is compliant with USB 2.0 specification and it contains 12 configurable endpoints in addition to control endpoint. These endpoints could be configured to BULK, INTERRUPT or ISO. The USB device controller has a built-in DMA to relieve the load of CPU.

### 26.2 Features

- USB Specification reversion 2.0 compliant
- Supports 12 configurable endpoints in addition to Control Endpoint
- Each of the endpoints can be Isochronous, Bulk or Interrupt and either IN or OUT direction
- Three different operation modes of an in-endpoint — Auto Validation mode, Manual Validation mode, Fly mode
- Supports DMA operation
- 4096 Bytes Configurable RAM used as endpoint buffer
- Supports Endpoint Maximum Packet Size up to 1024 bytes

### 26.3 Block Diagram



## 26.4 Register Map

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

Register	Offset	R/W	Description	Reset Value
<b>USBD Base Address:</b>				
<b>USBD_BA = 0x4001_9000</b>				
<b>USBD_GINTSTS</b>	USBD_BA+0x000	R	Interrupt Status Low Register	0x0000_0000
<b>USBD_GINTEN</b>	USBD_BA+0x008	R/W	Interrupt Enable Low Register	0x0000_0001
<b>USBD_BUSINTSTS</b>	USBD_BA+0x010	R/W	USB Bus Interrupt Status Register	0x0000_0000
<b>USBD_BUSINTEN</b>	USBD_BA+0x014	R/W	USB Bus Interrupt Enable Register	0x0000_0040
<b>USBD_OPER</b>	USBD_BA+0x018	R/W	USB Operational Register	0x0000_0002
<b>USBD_FRAMECNT</b>	USBD_BA+0x01C	R	USB Frame Count Register	0x0000_0000
<b>USBD_FADDR</b>	USBD_BA+0x020	R/W	USB Function Address Register	0x0000_0000
<b>USBD_TEST</b>	USBD_BA+0x024	R/W	USB Test Mode Register	0x0000_0000
<b>USBD_CEPDAT</b>	USBD_BA+0x028	R/W	Control-Endpoint Data Buffer	0x0000_0000
<b>USBD_CEPCTL</b>	USBD_BA+0x02C	R/W	Control-Endpoint Control and Status	0x0000_0000
<b>USBD_CEPINTEN</b>	USBD_BA+0x030	R/W	Control-Endpoint Interrupt Enable	0x0000_0000
<b>USBD_CEPINTSTS</b>	USBD_BA+0x034	R/W	Control-Endpoint Interrupt Status	0x0000_1800
<b>USBD_CEPTXCNT</b>	USBD_BA+0x038	R/W	Control-Endpoint In-transfer Data Count	0x0000_0000
<b>USBD_CEPRXCNT</b>	USBD_BA+0x03C	R	Control-Endpoint Out-transfer Data Count	0x0000_0000
<b>USBD_CEPDATCNT</b>	USBD_BA+0x040	R	Control-Endpoint data count	0x0000_0000
<b>USBD_SETUP1_0</b>	USBD_BA+0x044	R	Setup1 & Setup0 bytes	0x0000_0000

<b>USBD_SETUP3_2</b>	USBD_BA+0x048	R	Setup3 & Setup2 Bytes	0x0000_0000
<b>USBD_SETUP5_4</b>	USBD_BA+0x04C	R	Setup5 & Setup4 Bytes	0x0000_0000
<b>USBD_SETUP7_6</b>	USBD_BA+0x050	R	Setup7 & Setup6 Bytes	0x0000_0000
<b>USBD_CEPBUFSTART</b>	USBD_BA+0x054	R/W	Control Endpoint RAM Start Address Register	0x0000_0000
<b>USBD_CEPBUFEND</b>	USBD_BA+0x058	R/W	Control Endpoint RAM End Address Register	0x0000_0000
<b>USBD_DMACTL</b>	USBD_BA+0x05C	R/W	DMA Control Status Register	0x0000_0000
<b>USBD_DMACNT</b>	USBD_BA+0x060	R/W	DMA Count Register	0x0000_0000
<b>USBD_EPADAT</b>	USBD_BA+0x064	R/W	Endpoint A Data Register	0x0000_0000
<b>USBD_EPAINTSTS</b>	USBD_BA+0x068	R/W	Endpoint A Interrupt Status Register	0x0000_0003
<b>USBD_EPAINTEN</b>	USBD_BA+0x06C	R/W	Endpoint A Interrupt Enable Register	0x0000_0000
<b>USBD_EPADATCNT</b>	USBD_BA+0x070	R	Endpoint A Data Available Count Register	0x0000_0000
<b>USBD_EPARSPCTL</b>	USBD_BA+0x074	R/W	Endpoint A Response Control Register	0x0000_0000
<b>USBD_EPAMPS</b>	USBD_BA+0x078	R/W	Endpoint A Maximum Packet Size Register	0x0000_0000
<b>USBD_EPATXCNT</b>	USBD_BA+0x07C	R/W	Endpoint A Transfer Count Register	0x0000_0000
<b>USBD_EPACFG</b>	USBD_BA+0x080	R/W	Endpoint A Configuration Register	0x0000_0012
<b>USBD_EPABUFSTART</b>	USBD_BA+0x084	R/W	Endpoint A RAM Start Address Register	0x0000_0000
<b>USBD_EPABUFEND</b>	USBD_BA+0x088	R/W	Endpoint A RAM End Address Register	0x0000_0000
<b>USBD_EPBBDAT</b>	USBD_BA+0x08C	R/W	Endpoint B Data Register	0x0000_0000
<b>USBD_EPBINTSTS</b>	USBD_BA+0x090	R/W	Endpoint B Interrupt Status Register	0x0000_0003
<b>USBD_EPBINTEN</b>	USBD_BA+0x094	R/W	Endpoint B Interrupt Enable Register	0x0000_0000
<b>USBD_EPBDATCNT</b>	USBD_BA+0x098	R	Endpoint B Data Available Count Register	0x0000_0000
<b>USBD_EPBRSRSPCTL</b>	USBD_BA+0x09C	R/W	Endpoint B Response Control Register	0x0000_0000
<b>USBD_EPBMPMS</b>	USBD_BA+0x0A0	R/W	Endpoint B Maximum Packet Size Register	0x0000_0000
<b>USBD_EPBTXCNT</b>	USBD_BA+0x0A4	R/W	Endpoint B Transfer Count Register	0x0000_0000
<b>USBD_EPBBCFG</b>	USBD_BA+0x0A8	R/W	Endpoint B Configuration Register	0x0000_0022
<b>USBD_EPBBUFSTART</b>	USBD_BA+0x0AC	R/W	Endpoint B RAM Start Address Register	0x0000_0000
<b>USBD_EPBBUFEND</b>	USBD_BA+0x0B0	R/W	Endpoint B RAM End Address Register	0x0000_0000
<b>USBD_EPCDAT</b>	USBD_BA+0x0B4	R/W	Endpoint C Data Register	0x0000_0000
<b>USBD_EPCINTSTS</b>	USBD_BA+0x0B8	R/W	Endpoint C Interrupt Status Register	0x0000_0003
<b>USBD_EPCINTEN</b>	USBD_BA+0x0BC	R/W	Endpoint C Interrupt Enable Register	0x0000_0000
<b>USBD_EPCDATCNT</b>	USBD_BA+0x0C0	R	Endpoint C Data Available Count Register	0x0000_0000
<b>USBD_EPCRSPCTL</b>	USBD_BA+0x0C4	R/W	Endpoint C Response Control Register	0x0000_0000
<b>USBD_EPCMPS</b>	USBD_BA+0x0C8	R/W	Endpoint C Maximum Packet Size Register	0x0000_0000
<b>USBD_EPCTXCNT</b>	USBD_BA+0x0CC	R/W	Endpoint C Transfer Count Register	0x0000_0000
<b>USBD_EPCCFG</b>	USBD_BA+0x0D0	R/W	Endpoint C Configuration Register	0x0000_0032

<b>USBD_EPCBUFSTART</b>	USBD_BA+0x0D4	R/W	Endpoint C RAM Start Address Register	0x0000_0000
<b>USBD_EPCBUFEND</b>	USBD_BA+0x0D8	R/W	Endpoint C RAM End Address Register	0x0000_0000
<b>USBD_EPDDAT</b>	USBD_BA+0x0DC	R/W	Endpoint D Data Register	0x0000_0000
<b>USBD_EPDINTSTS</b>	USBD_BA+0x0E0	R/W	Endpoint D Interrupt Status Register	0x0000_0003
<b>USBD_EPDINTEN</b>	USBD_BA+0x0E4	R/W	Endpoint D Interrupt Enable Register	0x0000_0000
<b>USBD_EPDDATCNT</b>	USBD_BA+0x0E8	R	Endpoint D Data Available Count Register	0x0000_0000
<b>USBD_EPDRSPCTL</b>	USBD_BA+0x0EC	R/W	Endpoint D Response Control Register	0x0000_0000
<b>USBD_EPDMPMS</b>	USBD_BA+0x0F0	R/W	Endpoint D Maximum Packet Size Register	0x0000_0000
<b>USBD_EPDTXCNT</b>	USBD_BA+0x0F4	R/W	Endpoint D Transfer Count Register	0x0000_0000
<b>USBD_EPDCFG</b>	USBD_BA+0x0F8	R/W	Endpoint D Configuration Register	0x0000_0042
<b>USBD_EPDBUFSTART</b>	USBD_BA+0x0FC	R/W	Endpoint D RAM Start Address Register	0x0000_0000
<b>USBD_EPDBUFEND</b>	USBD_BA+0x100	R/W	Endpoint D RAM End Address Register	0x0000_0000
<b>USBD_EPEDAT</b>	USBD_BA+0x104	R/W	Endpoint E Data Register	0x0000_0000
<b>USBD_EPEINTSTS</b>	USBD_BA+0x108	R/W	Endpoint E Interrupt Status Register	0x0000_0003
<b>USBD_EPEINTEN</b>	USBD_BA+0x10C	R/W	Endpoint E Interrupt Enable Register	0x0000_0000
<b>USBD_EPEDATCNT</b>	USBD_BA+0x110	R	Endpoint E Data Available Count Register	0x0000_0000
<b>USBD_EPERSPCTL</b>	USBD_BA+0x114	R/W	Endpoint E Response Control Register	0x0000_0000
<b>USBD_EPEMPS</b>	USBD_BA+0x118	R/W	Endpoint E Maximum Packet Size Register	0x0000_0000
<b>USBD_EPETXCNT</b>	USBD_BA+0x11C	R/W	Endpoint E Transfer Count Register	0x0000_0000
<b>USBD_EPECFG</b>	USBD_BA+0x120	R/W	Endpoint E Configuration Register	0x0000_0052
<b>USBD_EPEBUFSTART</b>	USBD_BA+0x124	R/W	Endpoint E RAM Start Address Register	0x0000_0000
<b>USBD_EPEBUFEND</b>	USBD_BA+0x128	R/W	Endpoint E RAM End Address Register	0x0000_0000
<b>USBD_EPFDAT</b>	USBD_BA+0x12C	R/W	Endpoint F Data Register	0x0000_0000
<b>USBD_EPFINTSTS</b>	USBD_BA+0x130	R/W	Endpoint F Interrupt Status Register	0x0000_0003
<b>USBD_EPFINTEN</b>	USBD_BA+0x134	R/W	Endpoint F Interrupt Enable Register	0x0000_0000
<b>USBD_EPFDATCNT</b>	USBD_BA+0x138	R	Endpoint F Data Available Count Register	0x0000_0000
<b>USBD_EPFRSPCTL</b>	USBD_BA+0x13C	R/W	Endpoint F Response Control Register	0x0000_0000
<b>USBD_EPFMPS</b>	USBD_BA+0x140	R/W	Endpoint F Maximum Packet Size Register	0x0000_0000
<b>USBD_EPFTXCNT</b>	USBD_BA+0x144	R/W	Endpoint F Transfer Count Register	0x0000_0000
<b>USBD_EPFCFG</b>	USBD_BA+0x148	R/W	Endpoint F Configuration Register	0x0000_0062
<b>USBD_EPFBUFSTART</b>	USBD_BA+0x14C	R/W	Endpoint F RAM Start Address Register	0x0000_0000
<b>USBD_EPFBUFEND</b>	USBD_BA+0x150	R/W	Endpoint F RAM End Address Register	0x0000_0000
<b>USBD_EPGDAT</b>	USBD_BA+0x154	R/W	Endpoint G Data Register	0x0000_0000
<b>USBD_EPGINTSTS</b>	USBD_BA+0x158	R/W	Endpoint G Interrupt Status Register	0x0000_0003
<b>USBD_EPGINTEN</b>	USBD_BA+0x15C	R/W	Endpoint G Interrupt Enable Register	0x0000_0000

<b>USBD_EPGDATCNT</b>	USBD_BA+0x160	R	Endpoint G Data Available Count Register	0x0000_0000
<b>USBD_EPGRSPCTL</b>	USBD_BA+0x164	R/W	Endpoint G Response Control Register	0x0000_0000
<b>USBD_EPGMPS</b>	USBD_BA+0x168	R/W	Endpoint G Maximum Packet Size Register	0x0000_0000
<b>USBD_EPGTXCNT</b>	USBD_BA+0x16C	R/W	Endpoint G Transfer Count Register	0x0000_0000
<b>USBD_EPGCFG</b>	USBD_BA+0x170	R/W	Endpoint G Configuration Register	0x0000_0072
<b>USBD_EPGBUFSTART</b>	USBD_BA+0x174	R/W	Endpoint G RAM Start Address Register	0x0000_0000
<b>USBD_EPGBUFEND</b>	USBD_BA+0x178	R/W	Endpoint G RAM End Address Register	0x0000_0000
<b>USBD_EPHDAT</b>	USBD_BA+0x17C	R/W	Endpoint H Data Register	0x0000_0000
<b>USBD_EPHINTSTS</b>	USBD_BA+0x180	R/W	Endpoint H Interrupt Status Register	0x0000_0003
<b>USBD_EPHINTEN</b>	USBD_BA+0x184	R/W	Endpoint H Interrupt Enable Register	0x0000_0000
<b>USBD_EPHDATCNT</b>	USBD_BA+0x188	R	Endpoint H Data Available Count Register	0x0000_0000
<b>USBD_EPHRSPCTL</b>	USBD_BA+0x18C	R/W	Endpoint H Response Control Register	0x0000_0000
<b>USBD_EPHMPS</b>	USBD_BA+0x190	R/W	Endpoint H Maximum Packet Size Register	0x0000_0000
<b>USBD_EPHTXCNT</b>	USBD_BA+0x194	R/W	Endpoint H Transfer Count Register	0x0000_0000
<b>USBD_EPHCFG</b>	USBD_BA+0x198	R/W	Endpoint H Configuration Register	0x0000_0082
<b>USBD_EPHBUFSTART</b>	USBD_BA+0x19C	R/W	Endpoint H RAM Start Address Register	0x0000_0000
<b>USBD_EPHBUFEND</b>	USBD_BA+0x1A0	R/W	Endpoint H RAM End Address Register	0x0000_0000
<b>USBD_EPIDAT</b>	USBD_BA+0x1A4	R/W	Endpoint I Data Register	0x0000_0000
<b>USBD_EPIINTSTS</b>	USBD_BA+0x1A8	R/W	Endpoint I Interrupt Status Register	0x0000_0003
<b>USBD_EPIINTEN</b>	USBD_BA+0x1AC	R/W	Endpoint I Interrupt Enable Register	0x0000_0000
<b>USBD_EPIDATCNT</b>	USBD_BA+0x1B0	R	Endpoint I Data Available Count Register	0x0000_0000
<b>USBD_EPIRSPCTL</b>	USBD_BA+0x1B4	R/W	Endpoint I Response Control Register	0x0000_0000
<b>USBD_EPIMPS</b>	USBD_BA+0x1B8	R/W	Endpoint I Maximum Packet Size Register	0x0000_0000
<b>USBD_EPITXCNT</b>	USBD_BA+0x1BC	R/W	Endpoint I Transfer Count Register	0x0000_0000
<b>USBD_EPICFG</b>	USBD_BA+0x1C0	R/W	Endpoint I Configuration Register	0x0000_0092
<b>USBD_EPIBUFSTART</b>	USBD_BA+0x1C4	R/W	Endpoint I RAM Start Address Register	0x0000_0000
<b>USBD_EPIBUFEND</b>	USBD_BA+0x1C8	R/W	Endpoint I RAM End Address Register	0x0000_0000
<b>USBD_EPJDAT</b>	USBD_BA+0x1CC	R/W	Endpoint J Data Register	0x0000_0000
<b>USBD_EPJINTSTS</b>	USBD_BA+0x1D0	R/W	Endpoint J Interrupt Status Register	0x0000_0003
<b>USBD_EPJINTEN</b>	USBD_BA+0x1D4	R/W	Endpoint J Interrupt Enable Register	0x0000_0000
<b>USBD_EPJDATCNT</b>	USBD_BA+0x1D8	R	Endpoint J Data Available Count Register	0x0000_0000
<b>USBD_EPJRSPECTL</b>	USBD_BA+0x1DC	R/W	Endpoint J Response Control Register	0x0000_0000
<b>USBD_EPJMPS</b>	USBD_BA+0x1E0	R/W	Endpoint J Maximum Packet Size Register	0x0000_0000
<b>USBD_EPJTXCNT</b>	USBD_BA+0x1E4	R/W	Endpoint J Transfer Count Register	0x0000_0000
<b>USBD_EPJCFG</b>	USBD_BA+0x1E8	R/W	Endpoint J Configuration Register	0x0000_00A2

<b>USBD_EPJBUFSTART</b>	USBD_BA+0x1EC	R/W	Endpoint J RAM Start Address Register	0x0000_0000
<b>USBD_EPJBUFEND</b>	USBD_BA+0x1F0	R/W	Endpoint J RAM End Address Register	0x0000_0000
<b>USBD_EPKDAT</b>	USBD_BA+0x1F4	R/W	Endpoint K Data Register	0x0000_0000
<b>USBD_EPKINTSTS</b>	USBD_BA+0x1F8	R/W	Endpoint K Interrupt Status Register	0x0000_0003
<b>USBD_EPKINTEN</b>	USBD_BA+0x1FC	R/W	Endpoint K Interrupt Enable Register	0x0000_0000
<b>USBD_EPKDATCNT</b>	USBD_BA+0x200	R	Endpoint K Data Available Count Register	0x0000_0000
<b>USBD_EPKRSPCTL</b>	USBD_BA+0x204	R/W	Endpoint K Response Control Register	0x0000_0000
<b>USBD_EPKMPS</b>	USBD_BA+0x208	R/W	Endpoint K Maximum Packet Size Register	0x0000_0000
<b>USBD_EPKTXCNT</b>	USBD_BA+0x20C	R/W	Endpoint K Transfer Count Register	0x0000_0000
<b>USBD_EPKCFG</b>	USBD_BA+0x210	R/W	Endpoint K Configuration Register	0x0000_00B2
<b>USBD_EPKBUFSTART</b>	USBD_BA+0x214	R/W	Endpoint K RAM Start Address Register	0x0000_0000
<b>USBD_EPKBUFEND</b>	USBD_BA+0x218	R/W	Endpoint K RAM End Address Register	0x0000_0000
<b>USBD_EPLDAT</b>	USBD_BA+0x21C	R/W	Endpoint L Data Register	0x0000_0000
<b>USBD_EPLINTSTS</b>	USBD_BA+0x220	R/W	Endpoint L Interrupt Status Register	0x0000_0003
<b>USBD_EPLINTEN</b>	USBD_BA+0x224	R/W	Endpoint L Interrupt Enable Register	0x0000_0000
<b>USBD_EPLDATCNT</b>	USBD_BA+0x228	R	Endpoint L Data Available Count Register	0x0000_0000
<b>USBD_EPLRSPCTL</b>	USBD_BA+0x22C	R/W	Endpoint L Response Control Register	0x0000_0000
<b>USBD_EPLMPS</b>	USBD_BA+0x230	R/W	Endpoint L Maximum Packet Size Register	0x0000_0000
<b>USBD_EPLTXCNT</b>	USBD_BA+0x234	R/W	Endpoint L Transfer Count Register	0x0000_0000
<b>USBD_EPLCFG</b>	USBD_BA+0x238	R/W	Endpoint L Configuration Register	0x0000_00C2
<b>USBD_EPLBUFSTART</b>	USBD_BA+0x23C	R/W	Endpoint L RAM Start Address Register	0x0000_0000
<b>USBD_EPLBUFEND</b>	USBD_BA+0x240	R/W	Endpoint L RAM End Address Register	0x0000_0000
<b>USBD_DMAADDR</b>	USBD_BA+0x700	R/W	AHB DMA Address Register	0x0000_0000
<b>USBD_PHYCTL</b>	USBD_BA+0x704	R/W	USB PHY Control Register	0x0000_0420

## 26.5 Functional Description

The USB device controller is compliant with USB 2.0 specification. User can simulates the device as a mass storage card reader, virtual COM port, etc. Refer to “USB Class Specification” for more detail.

There are three different modes for IN-transfer operation:

- Auto-Validation Mode – When transfer data length is equal to the maximum packet size, user can choose this mode. (Such as Bulk pipe transfer).
- Manual-Validation Mode – This mode requires intervention of CPU for each transfer. When transfer data length is not fixed, user can choose this mode. (Such as Interrupt pipe transfer).

- Fly Mode – This mode is best suited for isochronous data transfer, where the speed of data transfer is more important than the packet size. (Such as Isochronous pipe transfer).

The following sections will be a USB mass storage device as an example. This device needs two endpoints – Endpoint A is Bulk IN, Endpoint B is Bulk Out.

### 26.5.1 Initialize

USB device controller initialize, please follow the steps below:

1. Set multiple function pin GPH0. Fill 0x7 to SYS\_GPH\_MFPL register GPH0 bit.
2. Set CLK\_HCLKEN register USBD bit.
3. Set USBD\_PHYCTL register PHYEN bit to enable USB PHY.
4. Fill 0x8 to USBD\_EPAMPS register. Polling USBD\_EPAMPS register until read data is 0x8. It means PHY clock stable.
5. Configure endpoint A to Bulk-IN type, endpoint number 1.
  - (1) Set USBD\_EPARSPCTL register MODE bit to 0 to select auto-validation mode.
  - (2) Fill 512 to USBD\_EPAMPS register. It means the maximum packet size is 512 bytes.
  - (3) Set USBD\_EPACFG register EPNUM bit to 1, EPDIR bit to 1, EPTYPE bit to 01b and EPEN bit to 1.
  - (4) Fill 0x200 to USBD\_EPABUFSTART register. 0x3FF to USBD\_EPABUFEND register. It means this endpoint FIFO length is 512 bytes.
6. Configure endpoint B to Bulk-Out type, endpoint number 2.
  - (1) Set USBD\_EPBINTEN register RXPKIEN bit to enable data receive interrupt.
  - (2) Set USBD\_EPBRSPCTL register MODE bit to 0 to select auto-validation mode.
  - (3) Fill 512 to USBD\_EPBMPMS register. It means the maximum packet size is 512 bytes.
  - (4) Set USBD\_EPBCFG register EPNUM bit to 2, EPDIR bit to 0, EPTYPE bit to 01b and EPEN bit to 1.
  - (5) Fill 0x400 to USBD\_EPBBUFSTART register. 0x5FF to USBD\_EPBBUFEND register. It means this endpoint FIFO length is 512 bytes.
7. Set USBD\_GINTEN register USBIEN, CEPIEN, EPAIEN and EPBIEN bit to enable USB bus, control endpoint, endpoint A and endpoint B interrupt.
8. Set USBD\_BUSINTEN register RSTIEN, RESUMEIEN, DMADONEIEN and VBUSDETIEN bit to enable USB reset, resume, DMA complete and floating detect interrupt.
9. Set USBD\_OPER register HISPDEN bit to enable high speed mode.
10. Clear USBD\_FADDR register.

11. Configure control endpoint (EP0)
  - (1) Fill 0x0 to USBD\_CEPBUFSTART register. 0x7F to USBD\_CEPBUFEND register. It means this endpoint FIFO length is 128 bytes.
  - (2) Set USBD\_CEPINTEN register SETUPPKIEN and STSDONEIEN bit to enable EP0 setup packet and setup status done interrupt.
12. Polling USBD\_PHYCTL register VBUSDET bit until it was set. It means device connect to host. Set USBD\_PHYCTL register DPPUEN bit to clear SE0.

### 26.5.2 Interrupt Service Routine

USBD control interrupt processing as follow:

1. Read USBD\_GINTSTS register and USBD\_GINTEN register to mask. User can get the interrupt information.
2. Read USBD\_BUSINTSTS register and USBD\_BUSINTEN register to mask. If match, it means USB BUS interrupt occurred.
3. Read USBD\_CEPINTSTS register and USBD\_CEPINTEN register to mask. If match, it means control endpoint interrupt occurred.
4. Read USBD\_EPAINTSTS register and USBD\_EPAINTEN register to mask. If match, it means endpoint A interrupt occurred.
5. Read USBD\_EPBIINTSTS register and USBD\_EPBIINTEN register to mask. If match, it means endpoint B interrupt occurred.

### 26.5.3 Standard Request

USBD Controller processes Standard Request:

1. Control endpoint setup packet interrupt occurred.
2. Read USBD\_SETUP1\_0, USBD\_SETUP3\_2, USBD\_SETUP5\_4 and USBD\_SETUP7\_6 register to get the setup packet information.
3. Analysis of the request. It supports, clear the NAK. (Set USBD\_CEPCTL register NAKCLR bit) and wait the status complete. Otherwise, send STALL to host. (Set USBD\_CEPCTL register STALLEN bit)

### 26.5.4 Set Address Request

USBD controller processes the “Set Address” request:

1. Control endpoint setup packet interrupt occurred.
2. Read USBD\_SETUP1\_0, USBD\_SETUP3\_2, USBD\_SETUP5\_4 and USBD\_SETUP7\_6

register to get the setup packet information.

- (1) Get bmRequestType from USBD\_SETUP1\_0 register low byte.
- (2) Get bRequest from USBD\_SETUP1\_0 register high byte.
- (3) Get wValue from USBD\_SETUP3\_2 register.
- (4) Get wIndex from USBD\_SETUP5\_4 register.
- (5) Get wLength from USBD\_SETUP7\_6 register. °
3. Get the address from wValue.
4. Clear NAK (Set USBD\_CEPCTL register NAKCLR bit)
5. Set USBD\_CEPINTSTS register STSDONEIF bit to 1 to clear status complete interrupt.  
Set USBD\_CEPINTEN register STSDONEIEN bit to 1 to enable interrupt.
6. Waiting for status complete interrupt occurred.
  - (1) Set USBD\_CEPINTEN register SETUPPKIEN bit to enable setup packet interrupt.
  - (2) Fill address to USBD\_FADDR register.
  - (3) Set USBD\_CEPINTSTS register STSDONEIF bit to 1 to clear status complete interrupt.

### 26.5.5 Get Descriptor

USBD controller processes the “Get Descriptor” request:

1. Control endpoint setup packet interrupt occurred.
2. Read USBD\_SETUP1\_0, USBD\_SETUP3\_2, USBD\_SETUP5\_4 and USBD\_SETUP7\_6 register to get the setup packet information.
  - (1) Get bmRequestType from USBD\_SETUP1\_0 register low byte.
  - (2) Get bRequest from USBD\_SETUP1\_0 register high byte.
  - (3) Get wValue from USBD\_SETUP3\_2 register.
  - (4) Get wIndex from USBD\_SETUP5\_4 register.
  - (5) Get wLength from USBD\_SETUP7\_6 register.
3. Get the descriptor type from wValue.
4. Check wLength and descriptor length.
5. Fill 1 to USBD\_CEPINTSTS register STSDONEIF and INTKIF bit to clear interrupt. Set USBD\_CEPINTEN register STSDONEIEN and INTKIEN bit to enable interrupt.
6. Waiting for IN-token interrupt occurred.
  - (1) Fill 1 to USBD\_CEPINTSTS register STSDONEIF and TXPKIF bit to clear interrupt.  
Set USBD\_CEPINTEN register STSDONEIEN and TXPKIEN bit to enable interrupt.
  - (2) Write descriptor data into USBD\_CEPDAT register.

- (3) Write descriptor length into USBD\_CEPINTSTS register to trigger data out.
7. Fill 1 to USBD\_CEPINTSTS register INTKIF bit to clear interrupt.
8. Waiting for TX interrupt occurred.
  - (1) Fill USBD\_CEPINTSTS register STSDONEIF and TXPKIF bit to clear interrupt.
  - (2) Clear NAK (Set USBD\_CEPCTL register NAKCLR bit).
  - (3) Fill 1 to USBD\_CEPINTSTS register STSDONEIF bit to clear status complete interrupt.
  - (4) Set USBD\_CEPINTEN register STSDONEIEN and SETUPPKIEN bit to enable interrupt.
9. Waiting for status complete interrupt occurred.
  - (1) Set USBD\_CEPINTEN register SETUPPKIEN bit to enable setup packet interrupt.
  - (2) Fill 1 to USBD\_CEPINTSTS register STSDONEIF bit to clear status complete interrupt.

### 26.5.6 IN Transmission

USBD controller handles IN transmission through DMA:

1. Set USBD\_DMACTL register DMARD bit to 1 for DMA read. Fill the endpoint number to USBD\_DMACTL register EPNUM bit.
2. Check the transfer length. If transfer length is greater than DMA count, user needs to separate the transmission.
3. Set USBD\_EPxINTEN register TXPKIEN bit to enable the data transmit interrupt.
4. Check whether FIFO empty or not. (USBD\_EPxINTSTS register BUFEMPTYIF bit is 1).
5. Set USBD\_BUSINTEN register RSTIEN, SUSPENDIEN and DMADONEIEN bit to enable USB reset, suspend and DMA complete interrupts.
6. Write physical source address to USBD\_DMAADDR register.
7. Write transfer count to USBD\_DMACNT register.
8. Set SBD\_DMACTL register DMAEN bit to trigger DMA.
9. Waiting for DMA complete interrupt occurred.
  - (1) Set USBD\_BUSINTSTS register DMADONEIF bit to clear interrupt.
  - (2) Check whether last packet is less than maximum packet size. If so, set USBD\_EPxRSPCTL register SHORTTXEN bit to output the last packet.

### 26.5.7 OUT Transmission

USBD controller handles OUT transmission through DMA:

1. Set USBD\_DMACTL register DMARD bit to 0 for DMA write. Fill endpoint number to USBD\_DMACTL register EPNUM bit.
2. Check the transfer length. If transfer length is greater than DMA count, user needs to separate the transmission.
3. Set USBD\_BUSINTEN register RSTIEN, SUSPENDIEN and DMADONEIEN bit to enable USB reset, suspend and DMA complete interrupts.
4. Write physical target address to USBD\_DMAADDR register.
5. Write transfer count to USBD\_DMACNT register.
6. Set SBD\_DMACTL register DMAEN bit to trigger DMA.
7. Waiting for DMA complete interrupt occurred.
  - (1) Set USBD\_BUSINTSTS register DMADONEIF bit to clear interrupt.
  - (2) Check whether received length is mass storage CBW or data length.
  - (3) Set USBD\_EPxINTEN register RXPKIEN bit to enable receive packet interrupt.
8. Waiting for received data interrupt occurred. Clear USBD\_EPxINTEN register RXPKIEN bit to disable receive data.

## 27 USB Host Controller

### 27.1 Overview

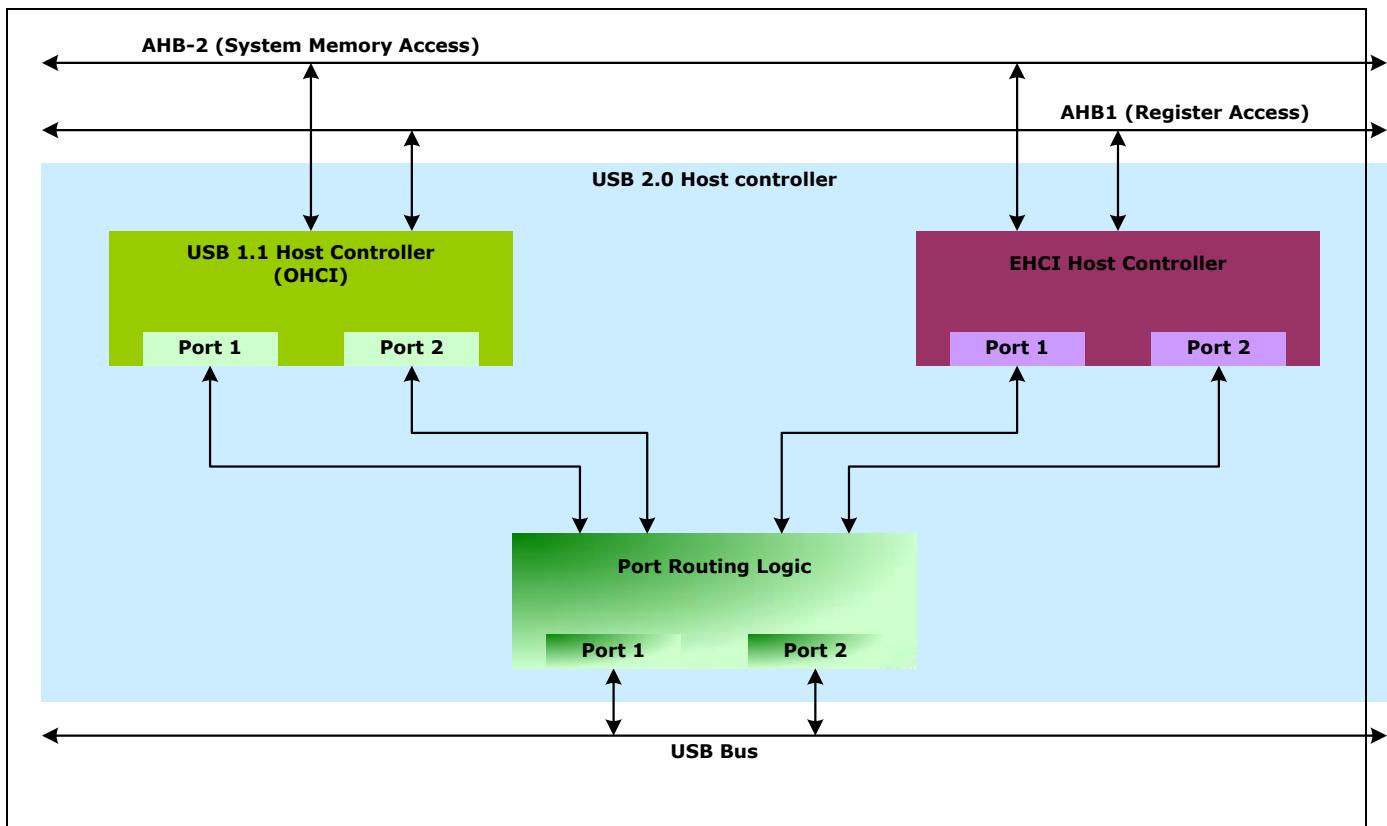
The Universal Serial Bus (USB) is a fast, bi-directional, isochronous, low-cost, dynamically attachable serial interface standard intended for modem, scanners, PDAs, keyboards, mice, and digital imaging devices. The USB is a 4-wire serial cable bus that supports serial data exchange between a Host Controller and a network of peripheral devices. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. Peripherals may be attached, configured, used, and detached, while the host and other peripherals continue operation (i.e. hot plug and unplug is supported).

The design purpose of USB standard is to achieve a flexible, plug-and-play of the USB device connectivity network. In any USB connectivity network, there will only be an USB host controller, but can connect up to 127 USB devices and hubs.

### 27.2 Features

- Fully compliant with USB Revision 2.0 specification.
- Enhanced Host Controller Interface (EHCI) Revision 1.0 compatible.
- Open Host Controller Interface (OHCI) Revision 1.0 compatible.
- Supports high-speed (480Mbps), full-speed (12Mbps) and low-speed (1.5Mbps) USB devices.
- Supports Control, Bulk, Interrupt, Isochronous and Split transfers.
- Integrated a port routing logic to route full/low speed device to OHCI controller.
- Built-in DMA for real-time data transfer.

### 27.3 Block Diagram



### 27.3.1 Basic Configuration

USB host clock source is derived from PLL and USB PHY. User has to set the PLL related configurations before USB host enabled. Set the USBH(CLK\_HCLKEN[18]) bit to enable USB host clock and 4-bit pre-scale USB\_N(CLK\_DIVCTL2[11:8]) to generate proper 48 MHz clock to USB host. In addition, USB host needs a clock from USB PHY for USB 2.0 high speed operation. User has to set SUSPEND(USBPCR0[8] and USBPCR1[8]) high to enable USB PHY. Then, user has to check if

CLKVALID(USBPCR0[11]) is high before starting to use USB host controller.

### 27.3.2 EHCI Controller

The EHCI is interfaced with the system through AHB interface. Whenever the CPU wants to initiate a register read or register write, it uses the AHB slave I/F signals and performs the necessary operation (register read writes). The CPU acts as a bus master, having initiated this transfer. At that time, EHCI acts as a target and responds to the transfer initiated by the system software. For example, if the CPU wants to write into one of the memory mapped registers of EHCI, it says the address and value to be written into that addressed register. EHCI targets the register by using that address and fills the register with the value specified

by the software. If it is a register read, EHCI gets the value from the addressed register and puts it on the system bus.

Likewise, when the EHCI wants to perform a data transfer, it acts as a master and initiates a data transfer. At that time, the system memory acts as a bus target. EHCI, as a master can perform two types of data transfers, from EHCI to the system memory and from system memory to the EHCI. When the EHCI wants the data to be moved from the downstream USB2.0 device to the system memory, it initiates a memory write transfer by accessing the memory interfacing signals. EHCI writes the control word (write), data and data count to be moved to the system memory. The memory controller accepts the data and moves it to the memory. If the data has to be moved from memory to the downstream device, the EHCI issue a read transfer to system bus. The memory controller gives data through the memory interfacing signals. EHCI accepts the data and moves them to the downstream device.

### 27.3.3 OHCI Controller

#### 27.3.3.1 AHB Interface

The OpenHCI Host Controller is connected to the system by the AHB bus. The design requires both master and slave bus operations. As a master, the Host Controller is responsible for running cycles on the AHB bus to access EDs and TDs as well as transferring data between memory and the local data buffer. As a slave, the Host Controller monitors the cycles on the AHB bus and determines when to respond to these cycles. Configuration and non-real-time control access to the Host Controller operational registers are through the AHB bus slave interface.

#### 27.3.3.2 AHB Master

The master issues the address and data onto the bus when granted.

#### 27.3.3.3 AHB Slave

The configuration of the Host Controller is through the slave interface.

#### 27.3.3.4 List Processing

The List Processor manages the data structures from the Host Controller Driver and coordinates all activity within the Host Controller.

#### 27.3.3.5 Frame Management

Frame Management is responsible for managing the frame specific tasks required by the USB specification and the OpenHCI specification. These tasks are:

- Management of the OpenHCI frame specific Operational Registers.
- Operation of the Largest Data Packet Counter.

- Performing frame qualifications on USB Transaction requests to the SIE.
- Generate SOF token requests to the SIE.

### 27.3.3.6 Interrupt Processing

Interrupts are the communication method for HC-initiated communication with the Host Controller Driver. There are several events that may trigger an interrupt from the Host Controller. Each specific event sets a specific bit in the HcInterruptStatus register.

### 27.3.3.7 Host Controller Bus Master

The Host Controller Bus Master is the central block in the data path. The Host Controller Bus Master coordinates all access to the AHB Interface. There are two sources of bus mastering within Host Controller: the List Processor and the Data Buffer Engine.

### 27.3.3.8 Data Buffer

The Data Buffer serves as the data interface between the Bus Master and the SIE. It is a combination of a 64-byte latched based bi-directional asynchronous FIFO and a single Dword AHB Holding Register.

### 27.3.3.9 USB Interface

The USB interface includes the integrated Root Hub with two external ports, Port 1 and Port 2 as well as the Serial Interface Engine (SIE) and USB clock generator. The interface combines responsibility for executing bus transactions requested by the HC as well as the hub and port management specified by USB.

### 27.3.3.10 Series Interface Engine (SIE)

The SIE is responsible for managing all transactions to the USB. It controls the bus protocol, packet generation/extraction, data parallel-to-serial conversion, CRC coding, bit stuffing, and NRZI encoding. All transactions on the USB are requested from the List Processor and Frame Manager.

### 27.3.3.11 Root Hub

The Root Hub is a collection of ports that are individually controlled and a hub that maintains control/status over functions common to all ports.

## 27.4 Register Map

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

## 27.5 Functional Description

### 27.5.1 Initialization

To initialize USB Host Controller, the following operations must be performed correctly:

- Set USBH(CLK\_HCLKEN [18]) bit as 1 to enable USB Host Controller clock source.
- Write 0x160 and 0x520 to USBPCR0 and USBPCR1 respectively to enable USB PHY0 and PHY1.
- Configure PE.14 and PE.15 multifunction pins for USBH\_PPWR0 and USBH\_PPWR1 respectively. NUC970/N9H30 UBS Host Controller uses these two pins to control external power switch IC, which provides power to USB port 0 and port1.
- Initialize OHCI Host Controller, which services USB 1.1 full-speed and low-speed devices.
- Initialize EHCI Host Controller, which services USB 2.0 high-speed devices.

### 27.5.2 Root Hub Port Routing Logic

NUC970/N9H30 series MCU equips EHCI (USB2.0) and OHCI (USB1.1) Host controller. Both Host Controllers share the two USB ports of root hub. If EHCI is enabled and in the activated state (UCFGR [0] is set to 1), it will be the default owner EHCI USB port. If EHCI is not enabled, OHCI will be only owner of root hub ports until EHCI is enabled. The ownership of root hub ports can be assigned to EHCI or OHCI individually.

EHCI Host Controller is designed for USB 2.0 devices. If an USB 2.0 device was plugged into the USB port, EHCI Host Controller will perform the standard USB device enumeration procedure to reset and enable the device. If success, user can perform USB data transfers on this device. Because EHCI has ownership of that USB port, there's any no port related status changes to OHCI. OHCI is totally unaware of the connection of USB 2.0 device.

However, if an USB 1.1 full-speed or low-speed device is plugged into a USB port, EHCI will fail to reset and enable it. In this case, EHCI driver or hub driver should change the ownership of that port to OHCI Host Controller. Set PO(UPSCRx [13]) bit as 1 can transfer port ownership to companion OHCI Host Controller. In the following, OHCI root hub ports will receive the status change of device connected. And it's OHCI Host Controller's turn to reset and enable the USB device. Once success, user can perform full/low-speed transfer on that USB device.

When the USB device is connected, OHCI driver cannot transfer port ownership to the EHCI until device disconnected. Once the USB device is disconnected, the ownership of the port is automatically returned to the EHCI, and PO(UPSCRx [13]) bit will be cleared by root hub.

### 27.5.3 OHCI

NUC970/N9H30 OHCI host controller is fully compliant with the Open Host Controller (OHCI),

version 1.0 standard. OHCI drivers running on other platforms can be easily ported to NUC970/N9H30.

### 27.5.3.1 Data Structure

In addition to direct access to the OHCI registers, system software interworks with OCHI controller via the following structured memory blocks:

- Endpoint Descriptor List
- Transfer Descriptor List
- Host Controller Communication Area(HCCA)

These data structures are defined by OHCI standard. System software allocates memory blocks to create these data structures. OHCI Host Controller has the ability to access these memory blocks by way of DMA transfer. All endpoint descriptors, transfer descriptors, HCCA and transmission buffers must be set to non-cacheable area. Endpoint descriptors and transfer descriptors must be aligned with the 32-byte address boundary. Host controller communication area must be aligned with 256-byte address boundary.

### 27.5.3.2 Endpoint Descriptor

The OpenHCI Host Controller fulfills USB transfers by classifying Endpoints into four types of Endpoint Descriptor lists. The Control ED list is pointed by HcControlHeadED register, the Bulk ED list is pointed by HcBulkHeadED register, the Interrupt ED lists are pointed by InterruptTable of HCCA, and the Isochronous ED list is linked behind the last 1m interval Interrupt ED. HCD must create and maintain an ED for each endpoint of a USB device.

For all transfer types, they have the same Endpoint Descriptor format. The common format is listed below:

	3			2						1	1	1	1	1	1	1		0	0	0	0	0	0	0																		
	1			6						6	5	4	3	2	1	0		7	6	5	4	3	2	1																		
Dword 0	—	MPS										F	K	S	D	EN		FA																								
Dword 1	TD Queue Tail Pointer (TailP)																																									
Dword 2	TD Queue Head Pointer (HeadP)																																									
Dword 3	Next Endpoint Descriptor (NextED)																																									

The Control ED list is created by Host Controller Driver (HCD), which should add any new EDs to the end of the Control ED list. HCD must write the physical address of the first ED of Control ED list to HcControlHeadED register. Thus, the HC can find the Control ED list and process all Control EDs. Similarly, all Bulk EDs are placed in the Bulk ED list, which must be pointed by the HcBulkHeadED register. And it's the responsibility of HCD to maintain Bulk ED list and link HcBulkHeadED.

The Interrupt ED lists are not directly pointed by any Host Controller operation registers, instead, they are pointed by the InterruptTable of HCCA (Host Controller Communication Area), which is a memory area created by HCD. In the HCCA, there are 32 entries InterruptTable with each entry points to an Interrupt ED list. The structure of Interrupt ED lists will be explained in the HCCA section.

The end of each Interrupt ED list must be linked to the identical 1ms-polling interval Interrupt ED list, which is also a part of each Interrupt ED list. You may have no any 1ms-polling interval Interrupt EDs in some of the real scenes. If it was the case, then you will have a placeholder on the node a 1ms interval Interrupt ED should be inserted. It is also true for 2m, 4m, 8m, 16ms, and 32ms polling interval Interrupt ED lists. In fact, an Interrupt ED list is composed of these various polling interval Interrupt ED lists.

The Isochronous ED list must be linked to the end of the 1ms-polling interval Interrupt ED list, that is, the end of any one Interrupt ED list. Host Controller Driver must maintain the Interrupt ED lists and Isochronous ED list, including the maintenance of HCCA and InterruptTable. The HCCA is pointed by HcHCCA register. Of course, HCD is responsible for creating HCCA and writing the physical address of HCCA to HcHCCA

### 27.5.3.3 Transfer Descriptor

ED is used to describe the characteristics of a specific endpoint. ED itself does not make HC to start any data transfer on USB bus. OpenHCI employs Transfer Descriptors (TDs) to describe the details of an USB data transfer. A Transfer Descriptor (TD) is a system memory data structure that is used by the Host Controller to define a buffer of data that will be moved to or from an endpoint.

Transfer Descriptors are linked to queues attached to EDs. The ED provides the endpoint address to/from where the TD data is to be transferred. Host Controller Driver adds TDs to the queue and Host

Controller removes TDs from the queue. Once the transfer of a TD was completed, Host Controller removed it from TD queue to the Done Queue.

There are two TD types in OpenHCI, General TD and Isochronous TD. The TD formats are listed below:

#### General Transfer Descriptor

	3		2	2	2	2	2	2	2	2	2	1	1											0		0
	1		8	7	6	5	4	3		1	0	9	8											3		0
Dword 0		CC		EC	T		DI		DP	R	—															
Dword 1																										
Dword 2																										0
Dword 3																										

#### Isochronous Transfer Descriptor

	3		2	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		0
Dword 0	CC	-	FC	DI	—													SF				
Dword 1	Buffer Page 0 (BPO)														—							
Dword 2	NextTD														0							
Dword 3	Buffer End (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														

#### 27.5.3.4 Host Controller Communication Area

The Host Controller Communications Area (HCCA) is a 256-byte structure of system memory, which is used by HCD to communicate with HC. HCCA must be aligned to 256 bytes address boundary. This memory block must be set to non-cacheable memory region, because HC accesses this memory block by DMA transfer. HCD must claim the physical address of HCCA by writing the physical address to HchCCA register to notify HC the address of HCCA.

Offset	Size (bytes)	Name	Description
0	128	HccaInterruptTable	These 32 Dwords are pointers to interrupt EDs.
0x80	2	HccaFrameNumber	Contains the current frame number. This value is updated by the HC before it begins processing the periodic lists for the frame.
0x82	2	HccaPad1	When the HC updates <b>HccaFrameNumber</b> , it sets this word to 0.
0x84	4	HccaDoneHead	When the HC reaches the end of a frame and its deferred interrupt register is 0, it writes the current value of its HcDoneHead to this location and generates an interrupt if interrupts are enabled. This location is not written by the HC again until software clears the WD bit in the HcInterruptStatus register. The Lsb of this entry is set to 1 to indicate whether an unmasked HcInterruptStatus was set when <b>HccaDoneHead</b> was written.
0x88	116	Reserved	Reserved for use by Host Controller.

#### 27.5.3.5 OHCI Initialization

The initialization of Host Controller may contain the following steps:

1. Disable Host Controller interrupts by writing 1 to MIE(HcIntDis[31]).

2. Issue a software reset command by writing 1 to HCR(HcComSts[0]) and waiting for 10ms until the HCR be cleared as 0 by Host Controller.
3. Allocate and create all necessary list structures and memory blocks, including HCCA, and initialize all driver-maintained lists, including InterruptTable of HCCA (Note that HCCA must be aligned with 256-bytes address boundary, while EDs and TDs must be aligned with 32-bytes address boundary).
4. Clear HcCtrHED and HcBlkHED register.
5. Write the physical address of HCCA memory block to HcHCCA register.
6. Write frame interval value ( $11,999 \pm 6$ ) to HcFmIntv register, and write 90% of this frame interval value (recommended) to HcPerSt register.
7. Write 0x628 to HcLSTH register (0x628 is also the reset default value of HcLSTH register).
8. Write 1 to BLE(HcControl[5]), CLE(HcControl[4]), IE(HcControl[3]), PLE(HcControl[2]) to enable Bulk, Control, Interrupt, and Isochronous transfers.
9. Write 10b to HCFS(HcControl[7:6]) to make Host Controller enter operational state.
10. Enable desired interrupts by writing corresponding bits to HcIntEn register and clear interrupt status of these interrupts by writing corresponding bits to HcIntSts register.
11. Turn on the Root Hub port power by writing 1 to LPSC(HcRhSts[16]) (Note that NUC970/N9H30 Series MCU USB Root Hub uses global power switching mode)
12. Enable AIC (NUC970/N9H30 Advanced Interrupt Controller) OHCI interrupt. The IRQ number of OHCI is 24.

#### 27.5.3.6 Interrupt Processing

NUC970/N9H30 Series MCU OHCI Host Controller may raise the following interrupts:

- Scheduling Overrun
- Write Back Done Head
- Start of Frame
- Resume Detected
- Unrecoverable Error
- Frame Number Overflow
- Root Hub Status Change
- Ownership Change

#### Scheduling Overrun Interrupt

This interrupt is set when the USB schedule for the current frame overruns. The presence of

this interrupt means that HCD has scheduled too many transfers. HCD may temporarily stop one or more endpoints to reduce bandwidth.

### Write Back Done Head Interrupt

This interrupt is set after Host Controller has written HcDoneH to HccaDoneHead. On this interrupt, HCD can obtain the TD done queue by reading HccaDoneHead. HCD may first reverse the done queue by traveling the done queue, because the TDs were retired in stack order. Then HCD can start processing on each TD.

### Start of Frame Interrupt

This interrupt is set on each start of a frame. Generally, HCD will not enable this interrupt. This interrupt is generally used to identify the starting of a next frame. For example, if you are going to remove a TD, you must ensure that the endpoint is not currently processed by Host Controller. To accomplish this, HCD can temporarily set the sKip bit of its ED and enable Start of Frame interrupt. In the next coming Start of Frame interrupt, HCD can ensure that the endpoint is not currently processed by Host Controller, and it can remove the TD.

### Resumed Detected Interrupt

This interrupt is set when Host Controller detects that a device on the USB bus is asserting a resume signal. If Host Controller is in USBSUSPEND state, the resume signal will make Host Controller automatically enter USBRESUME state.

### Unrecoverable Error Interrupt

The Host Controller will raise this interrupt when it detects a system error not related to USB or an error that cannot be reported in any other way. HCD may try to reset Host Controller in this case.

### Frame Number Overflow Interrupt

The Host Controller will raise this interrupt when the MSB bit of FN(HcFNum[15:0]) toggles value from 0 to 1 or 1 to 0, and after HcFNum register has been updated. Because the Host Controller has only 16-bits frame counter, the HCD may want to maintain a wider range frame counter. If the HCD want to maintain a 32-bits frame counter, it can increase the upper 16-bits value by each two Frame Number Overflow interrupt.

### Root Hub Status Change Interrupt

Once OCIC(HcRhsts[17]), CSC(HcRhPtx[16]), PESC(HcRhPtx[17]), or PSSC(HcRhPtx[20]) is set, the Host Controller would raise this interrupt.

## Ownership Change Interrupt

Host Controller would raise this interrupt when HCD write 1 to OCR(HcComSts[3]).

### 27.5.3.7 Done Queue Processing

The Done Queue is built by the Host Controller and referred to by the HcDoneH register. No matter successful or failed, the retired Transfer Descriptors must be put into the Done Queue by Host Controller. When Host Controller reaches the end of a frame (1ms) and its internal deferred interrupt register is 0, it writes the location of Done Queue to HccaDoneHead and raises a Write Back Done Head interrupt. HCD can take the Done Queue by servicing the Write Back Done Head interrupt.

### Reverse Done Queue

Host Controller queues TDs into the Done Queue by first-in-last-out order. The latest queued TD is linked at the head of the Done Queue, while the earliest queued TD is linked at the end of the Done Queue. HCD must reverse the Done Queue before it can start to process the retired TDs.

### Processing Done Queue

Once TDs in Done Queue are reversed into their original order, HCD can start to process these TDs one by one. For each TD, HCD checks whether the TD was completed with any errors.

### 27.5.3.8 Root Hub

The Root Hub is integrated into Host Controller and the control of Root Hub is done by accessing register files. NUC970/N9H30 OHCI Host Controller has provided several Root Hub related registers. The HcRhDeA and HcRhDeB registers are informative registers, which are used to describe the characteristics and capabilities of Root Hub. The HcRhSts register presents the current status and reflects the change of status of Root Hub. The HcRhPrt[1:2] register presents the current status and reflects the change of status of a Root Hub port. NUC970/N9H30 Series MCU OHCI Root Hub has two hub ports, the HcRhPrt[1] and HcRhPrt[2] are respectively dedicated to port 0 and port 1.

### HcRhDeA and HcRhDeB

HcRhDeA and HcRhDeB registers are informative registers, which are used to describe the characteristics and capabilities of Root Hub. The characteristics and capabilities of NUC970/N9H30 OHCI Root Hub are listed in the followings:

- Two downstream ports
- Ports are power switched

- Power switching mode is global power switch
- Is not a compound device
- Over-current status is reported collectively for all downstream ports
- Power-on-to-power-good-time is 2ms
- Devices attached to any ports are removable

### HcRhsts

The HcRhsts register is used to control and monitor the Root Hub status. The Root Hub can be controlled by the following actions:

- ClearGlobalPower - write 1 to LPS(HcRhsts[0]).
- SetRemoteWakeUpEnable - write 1 to LPS(HcRhsts[15]).
- SetGlobalPower - write 1 to LPSC(HcRhsts[16]).
- ClearRemoteWakeUpEnable - write 1 to CRWE(HcRhsts[31]).

In addition, HcRhsts register also indicates the following status:

- OCI(HcRhsts[0]) indicates overcurrent condition.
- DRWE(HcRhsts[15]) indicates the remote wakeup status. If this bit is 1, Connect Status Change is determined as a remote wakeup event
- OCIC(HcRhsts[15]) - This bit was set when the OverCurrentIndicator bit changed

### HcRhPrt[1] and HcRhPrt[2]

HcRhPrt[1] and HcRhPrt[2] registers are used to control and monitor the status Root Hub ports.

HcRhPrt[1] is used to indicate port 1 status and HcRhPrt[2] for port 2 respectively. The lower word of HcRhPrt is used to reflect the port status, whereas the upper word is used to reflect the changing of lower word status bits. Some status bits are implemented with special write behavior. You can do the following actions to control the Root Hub port:

- ClearPortEnable - write 1 to CCS(HcRhPrtx[0]).
- SetPortEnable - write 1 to PES(HcRhPrtx[1]).
- SetPortSuspend - write 1 to PES(HcRhPrtx[2]).
- ClearPortSuspend - write 1 to PES(HcRhPrtx[3]).
- SetPortReset - write 1 to PRS(HcRhPrtx[4]).

You can get the current status of the Root Hub port by reading the following bits:

- CCS(HcRhPrtx[0]) indicates the current connect status of the Root Hub port.
- PES(HcRhPrtx[1]) indicates whether the port is enabled.
- PSS(HcRhPrtx[2]) indicates the port is suspended.
- PRS(HcRhPrtx[4]) indicates the Root Hub is asserting reset signal on this port.
- PPS(HcRhPrtx[8]) indicates the port's power state.
- LSDA(HcRhPrtx[9]) indicates a low-speed device is attached to this port.

The following bits indicate the change of status bits. Write '1' to these bits will clear the events:

- CSC(HcRhPrtx[16]) indicates change of CCS(HcRhPrtx[0]).
- PESC(HcRhPrtx[17]) indicates change of PES(HcRhPrtx[1]).
- PSSC(HcRhPrtx[18]) indicates change of PSS(HcRhPrtx[2]).
- PRSC(HcRhPrtx[20]) indicates change of PRS(HcRhPrtx[4]).

## 27.5.4 EHCI

NUC970/N9H30 EHCI host controller is fully compliant with the Enhanced Host Controller Interface (EHCI), version 1.0 standard. EHCI drivers running on other platforms, can be easily ported to NUC970/N9H30.

### 27.5.4.1 Data Structure

Except direct access to Host Controller by registers, Host Controller Driver must maintain the following memory blocks to communicate with Host Controller:

- Isochronous (High-Speed) Transfer Descriptor (iTД)
- Split Transaction Isochronous Transfer Descriptor (siTD)
- Queue Element Transfer Descriptor (qTD)
- Queue Head
- Periodic Frame Span Traversal Node (FSTN)

### 27.5.4.2 Isochronous Transfer Descriptor (iTД)

This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. Isochronous TDs must be aligned on a 32-byte boundary. Note that siTD must be located in non-cacheable memory.

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						

#### 27.5.4.3 Split Transaction Isochronous Transfer Descriptor (siTD)

All Full-speed isochronous transfers through Transaction Translators are managed using the siTD data structure. This data structure satisfies the operational requirements for managing the split transaction protocol. Note that siTD must be located in non-cacheable memory.

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	

#### 27.5.4.4 Queue Element Transfer Descriptor (qTD)

This data structure is only used with a queue head. This data structure is used for one or more USB transactions. This data structure is used to transfer up to 20480 (5\*4096) bytes. The structure contains two structure pointers used for queue advancement, a Dword of transfer state and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Note that qTD must be located in non-cacheable memory.

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						

#### 27.5.4.5 EHCI Initialization

The initialization of EHCI Host Controller may contain the following steps :

1. Write 1 to USBH(CLK\_HCLKEN[18]) to enable USB Host clock.
2. Enable PHY 0 by writing 0x160 to USBPCR0 register, and enable PHY 1 by writing 0x120 to USBPCR1 register.
3. Force EHCI to halt state. It can be done by writing 0 to RUN(UCMDR[0]).
4. Write 1 to HCRST(UCMDR[1]) to reset EHCI Host Controller. This bit will be cleared by Host Controller once reset process completed.
5. Allocated non-cacheable memory for Periodic Frame List, which is an array of 32-bits pointers. Writing 0x01 (means end-of-list) to all entries of Periodic Frame List. And then writing the physical address of Periodic Frame List to UPFLBAR register.
6. Enable EHCI interrupts by writing corresponding bits to UIENR register.
7. Allocate main memory to create a dummy Queue Head for the asynchronous ring head. And writing physical address of the Queue Head to UCALAR register.
8. Write 1 to UCFGR register. This will make the port routing logic to default-route all ports to EHCI controller.
9. Write 1 to PP(UPSCR[0]) and PP(UPSCR[1]) to enable port power of root hub port 0 and port1. Once an USB device was connected, the port status register UPSCR0/1 can reflect it.

#### 27.5.4.6 USB Commands

EHCI driver issues commands to Host Controller by writing commands to UCMDR register.

##### Run/Stop

Write 1 to RUN(UCMDR[0]) can make Host Controller enter operational state. Host Controller keeps operating as long as this bit is 1. Once RUN(UCMDR[0]) is cleared to 0, Host Controller completes the current and any actively pipelined transactions on the USB and then enter Halted state. HCHalted(USTSR[12]) indicates whether Host Controller has finished its transactions and has entered Halted state. EHCI driver must not write a one to this field unless Host Controller is in Halted state, it will yield unexpected results.

##### Host Controller Reset

Write 1 to HCRST(UCMDR[1]) can reset EHCI Host Controller. The effects of this on Root Hub registers are similar to a Chip Hardware Reset. HCRST(UCMDR[1]) will be cleared as 0 by Host Controller when the reset process is completed. Writing 0 to HCRST(UCMDR[1])

cannot cancel the reset process. If the HCHalted(USTSR[12]) is 0, it's not legal to write 1 to HCRST(UCMDR[1]). Attempting to reset an actively running host controller will result in unexpected errors.

### Frame List Size

FLSZ(UCMDR[3:2] indicates size of the frame list. The size the frame list controls which bits in the Frame Index Register should be used for the Frame List Current index. Values mean:

- 00b** 1024 elements (4096 bytes) Default value
- 01b** 512 elements (2048 bytes)
- 10b** 256 elements (1024 bytes) – for resource-constrained environments
- 11b** Reserved

### Periodic Schedule Enable

PSEN(UCMDR[4]) controls whether Host Controller skips processing the Periodic Schedule. Values mean:

- 0b** Do not process the Periodic Schedule
- 1b** Use the PERIODICLISTBASE register to access the Periodic Schedule.

### Asynchronous Schedule Enable

ASEN(UCMDR[5]) controls whether Host Controller skips processing the Asynchronous Schedule. Values mean:

- 0b** Do not process the Asynchronous Schedule
- 1b** Use the ASYNCLISTADDR register to access the Asynchronous Schedule.

### Interrupt on Async Advance Doorbell

IAAD(UCMDR[6]) is used as a doorbell by software to ask Host Controller to issue an interrupt the next time it advances asynchronous schedule. EHCI driver writes 1 IAAD(UCMDR[6]) to ring the doorbell. It's illegal to write 1 to IAAD(UCMDR[6]) if asynchronous schedule is disabled.

### Interrupt Threshold Control

ITC(UCMDR[23:16]) determines the maximum rate at which Host Controller will issue interrupts. The only valid values are defined as the followings. Any other value is illegal.

Value	Maximum Interrupt Interval
<b>00h</b>	Reserved

<b>01h</b>	1 micro-frame
<b>02h</b>	2 micro-frames
<b>04h</b>	4 micro-frames
<b>08h</b>	8 micro-frames (default, equates to 1 ms)
<b>10h</b>	16 micro-frames (2 ms)
<b>20h</b>	32 micro-frames (4 ms)
<b>40h</b>	64 micro-frames (8 ms)

#### 27.5.4.7 Interrupt Status

##### USB Interrupt (USBINT)

Host Controller sets USBINT(USTSR[0]) to 1 on the completion of a USB transaction, which implies the retirement of a Transfer Descriptor that had its IOC bit set. Host Controller also sets USBINT to 1 when a short packet is detected (actual number of bytes received was less than the expected number of bytes).

##### USB Error Interrupt (UERRINT)

Host Controller sets UERRINT(USTSR[1]) to 1 when completion of a USB transaction is caused by errors. If the TD on which the error interrupt occurred also had its IOC bit set, both UERRINT(USTSR[1]) and USBINT(USTSR[0]) will be set.

##### Port change Detect (PCD)

Host Controller sets PCD(USTSR[2]) to 1 when any port has a change bit transition from a zero to a one or a Force Port Resume bit transition from a zero to a one. PCD(USTSR[2]) will also be set as a result of the Connect Status Change being set to a one after system software has relinquished ownership of a connected port by writing a one to a port's Port Owner bit.

##### Frame List Rollover (FLR)

Host Controller sets FLR(USTSR[3]) to a one when the Frame List Index rolls over from its maximum value to zero. The exact value at which the rollover occurs depends on the frame list size.

##### Host System Error (HSERR)

Host Controller sets HSERR(USTSR[4]) to 1 when a serious error occurs during Host Controller accessing system memory. When this error occurs, the Host Controller clears the Run(USTSR[0]) to prevent further execution of the scheduled TDs.

##### Interrupt on Async Advance (IAA)

System software can ask Host Controller to issue an interrupt the next time the host controller advances the asynchronous schedule by writing 1 to IAA(USTSR[5]). This status bit indicates the assertion of that interrupt source.

### HcHalted

HcHalted(USTSR[12]) should be 0 if Run(USTSR[0]) is a one. Host Controller sets HcHalted(USTSR[12]) to 1 after it has stopped executing as a result of Run(USTSR[0]) being set to 0, either by software or by Host Controller.

### Reclamation (RCLA)

RECLA(USTSR[13]) is a read-only status bit, which is used to detect an empty asynchronous schedule.

### Periodic Schedule Status (PSS)

PSS(USTSR[14]) indicates the current status of the Periodic Schedule. If PSS is 0, the Periodic Schedule is disabled. If PSS is 1, the Periodic Schedule is enabled.

### Asynchronous Schedule Status

ASS(USTSR[15]) indicates the current real status of Asynchronous Schedule. If ASS is 0, Asynchronous Schedule is disabled. If ASS is 1, Asynchronous Schedule is enabled.

#### 27.5.4.8 Root Hub

NUC970/N9H30 Series MCU EHCI host controller implements two port registers, UPSCR0 and UPSCR1. NUC970/N9H30 Series MCU EHCI root hub ports has port power control, software cannot change the state of the port until after it applies power to the port by writing 1 to PP(UPSCR[12]). Software must not attempt to change the state of the port until power is stable on the port. Host Controller will make port power stable within 20 milliseconds. The root hub ports control and status bits are listed as the following:

### Current Connect Status (CCS)

CCS(UPSCR[0]) indicates the current connect state of the port, and may not correspond directly to the event that caused the Connect Status Change bit (Bit 1) to be set.

### Connect Status Change (CSC)

CSC(UPSCR[1]) indicates a change has occurred in the port's Current Connect Status. This status bit can be cleared by writing 1 to 1.

### Port Enable/Disable (PE)

Ports can only be enabled by Host Controller as a part of the reset and enable. Software cannot enable a port by writing a one to PE(UPSCR[2]). Host Controller will only set this bit to a one when the reset sequence determines that the attached device is a high-speed device.

Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by host software. Note that the bit status does not change until the port state actually changes.

When the port is disabled (0b) downstream propagation of data is blocked on this port, except for reset.

### Port Enable/Disable Change (PEC)

For the root hub, PEC(UPSCR[3]) is set to a one only when a port is disabled by Host Controller. Software can clear PEC(UPSCR[3]) by writing 1 to it.

### Over-current Active (OCA)

OCA(UPSCR[4]) indicates port over-current condition. Host Controller updates this it when port over-current condition changed. OCA is a read-only bit to software.

### Over-current Change (OCC)

When there is a change to OCA(UPSCR[4]), Host Controller will set OCC(UPSCR[5]) as 1.

Software can clear OCC(UPSCR[5]) by writing 1 to it.

### Force Port Resume (FPR)

Writing 1 to FPR(UPSCR[6]) makes Host Controller drive resume signal on that port. Host Controller sets this bit to a 1 if a J-to-K transition is detected if the port is in the Suspend state. When this bit transitions to 1 by J-to-K transition being detected, PCD(USTSR[2]) is also set to 1 by Host Controller. If software issues FPR, Host Controller will not set PCD(USTSR[2]).

### Suspend

Both PE(UPSCR[2]) and SUSPEND(UPSCR[7]) together define the port states as follows:

Bits [Port Enabled, Suspend]	Port State
0X	Disable
10	Enable
11	Suspend

When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction, if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Note that the bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB.

### **Port Reset (PRST)**

When software writes a one to PRST(UPSCR[8]), the bus reset sequence as defined in the USB Specification Revision 2.0 is started by Host Controller. Software writes 0 to PRST(UPSCR[8]) to terminate the bus reset sequence after 10ms later. Software must keep this bit as 1 long enough to ensure the reset sequence, as specified in the USB Specification Revision 2.0, completes. When software writes this bit to 1, it must also write 0 to the Port Enable bit.

Note that when software writes 0 to PRST there may be a delay before the bit status changes to a zero. Host Controller will not clear PRST to 0 until the reset process is completed. If the port is in high-speed mode after reset completed, Host Controller will automatically enable this port and set PE(UPSCR[2]) as 1.

Before writing 1 to PRST, software must make sure HCHalted(USTSR[12]) be 0.

### **Port Power (PP)**

Software writes 1 to PP(UPSCR[12]) to turn on the port power, and writes 0 to turn off the port power. When an over-current condition is detected on a powered port, Host Controller will force to turn off the port power and clear PP(UPSCR[12]) as 0.

### **Port Owner (PO)**

PO(UPSCR[13]) indicates the port owner is OHCI or EHCI Host Controller. PO(UPSCR[13]) unconditionally goes to 0 when software writes 1 to UCFGR register. PO(UPSCR[13]) unconditionally goes to 1 whenever the UCFGR is cleared.

System software uses PO(UPSCR[13]) to release ownership of the port to OHCI Host Controller (in the event that the attached device is not a high-speed device). Software writes 1 to PO when the attached device is not a high-speed device.

## 28 Capture Sensor Interface Controller

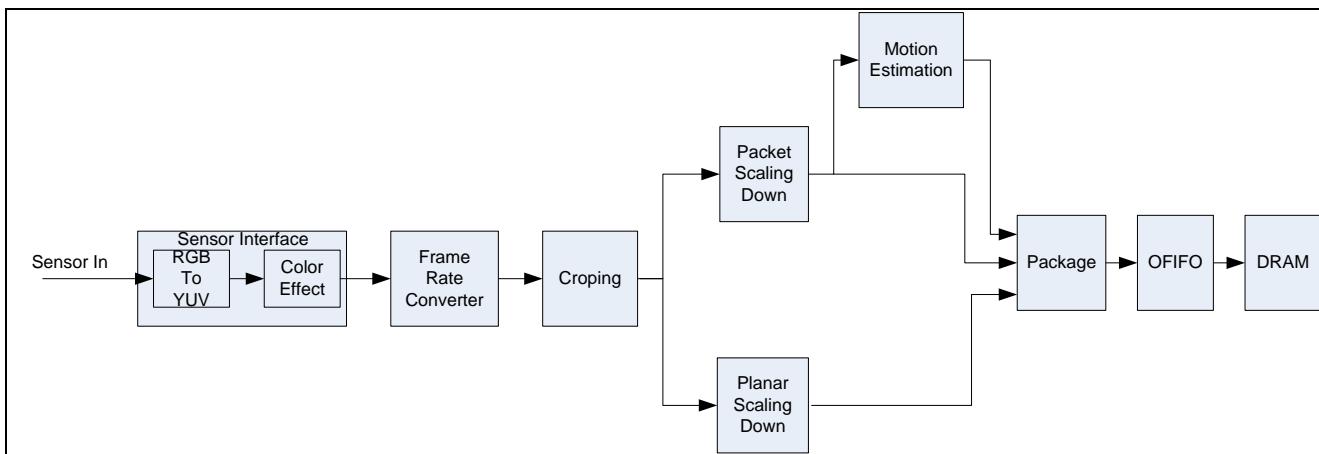
### 28.1 Overview

The Image Capture Interface is designed to capture image data from a sensor. After capturing or fetching image data, it will process the image data, and then FIFO output them into frame buffer.

### 28.2 Features

- 8-bit RGB565 sensor
- 8-bit YUV422 sensor
- Supports CCIR601 YCbCr color range scale to full YUV color range
- Supports 4 packaging format for packet data output: YUYV, Y only, RGB565, RGB555
- Supports YUV422 planar data output
- Supports the CROP function to crop input image to the required size for digital application.
- Supports the down scaling function to scale input image to the required size for digital application.
- Supports frame rate control
- Supports field detection and even/odd field skip mechanism
- Supports packet output dual buffer control through hardware buffer controller
- Supports negative/sephia/posterization color effect
- Supports two independent capture interfaces

### 28.3 Block Diagram



### 28.4 Register Map

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

Register	Offset	R/W	Description	Reset Value
<b>Capture Base Address:</b>				
<b>CAP_BA = 0xB000_E000</b>				
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_CURADDY	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

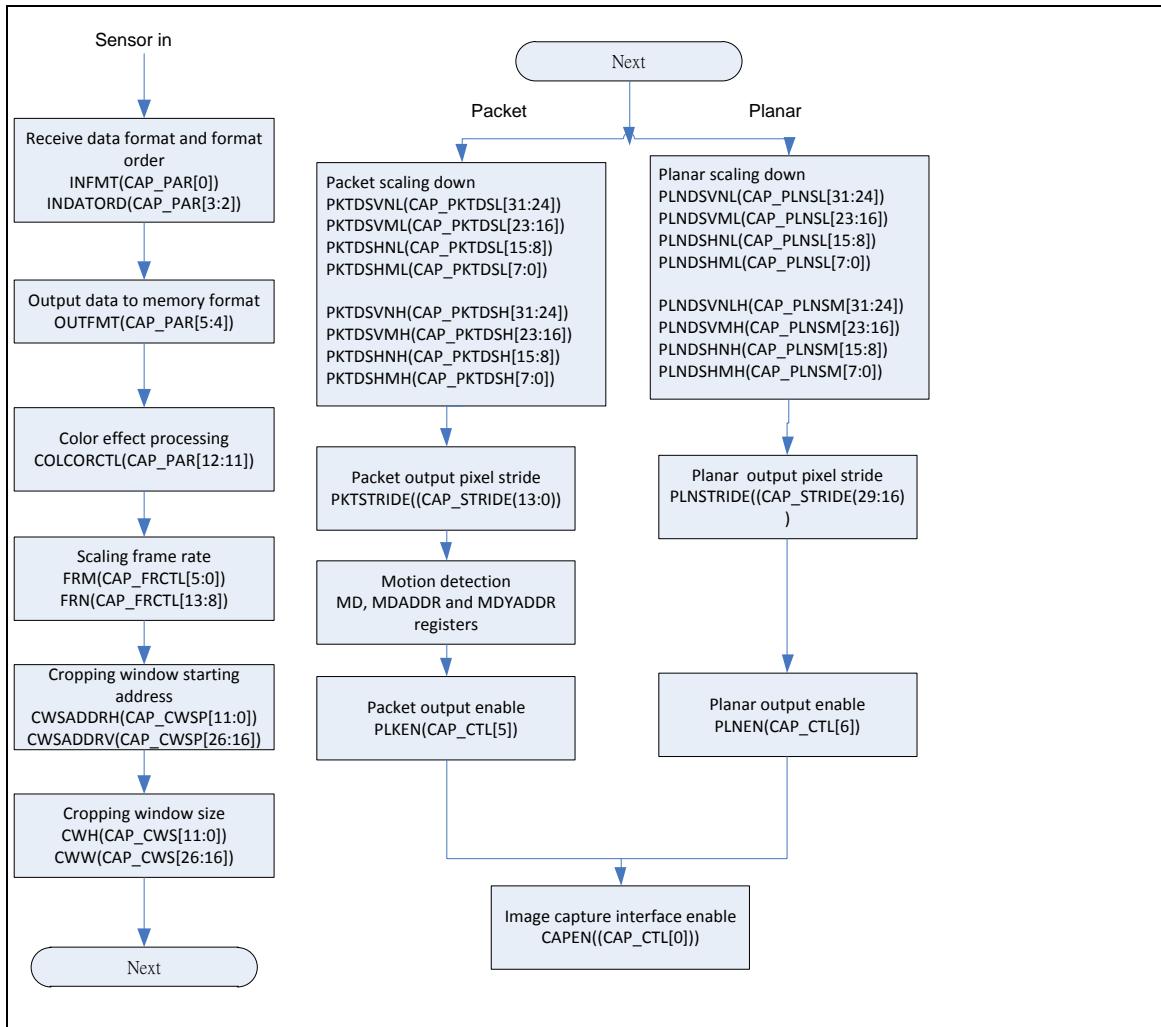
## 28.5 Functional Description

### 28.5.1 Basic Configuration

The CAP peripheral clock can be enabled in CAP (HCLKEN1[26]). The CAP engine clock

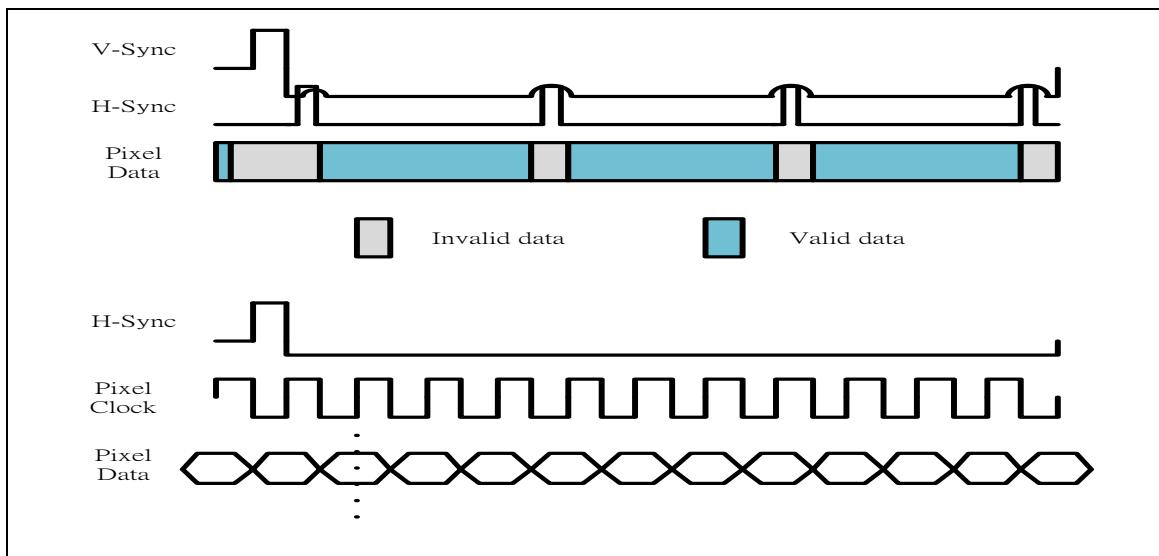
source is selected by SENSOR\_S (CLKDIV3[23:16]) and CAP engine clock divider is determined by SENSOR\_N (CLKDIV3[27:24])

### 28.5.2 Image Capture Flow Chart



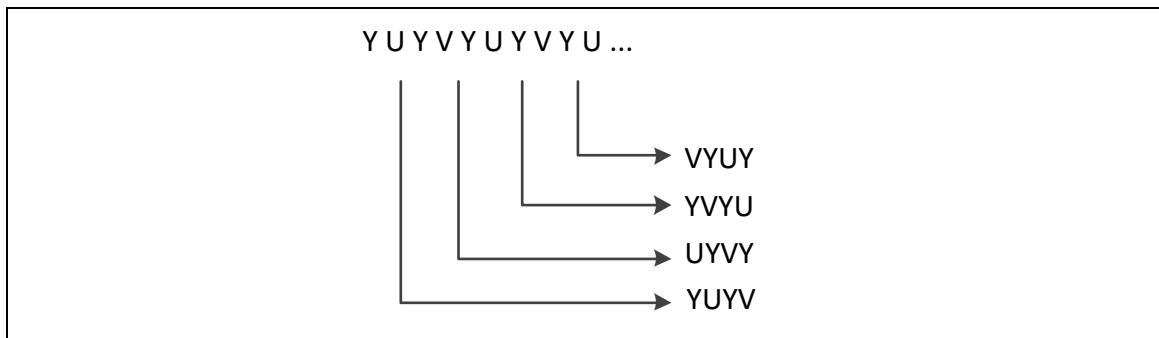
### 28.5.3 Polarity and Input Data Order

Sensor uses three control pins to notify the Video-In engine for a new frame, a new horizontal line or a new pixel. These pins are VSYNC, HSYNC and PCLK respectively. The polarity of VSYNC and HSYNC define positive or negative level that is the synchronization period. In addition, rising or falling edge of PCLK latches the image data. The following figures illustrate vertical synchronization polarity in high horizontal synchronization polarity, in high and rising edge latch data.



#### 28.5.4 Sensor Data Input Order

Input data order may be U0Y0VY1, Y0U0Y1V0, V0Y0U0Y1 or Y0V0Y1U0 after cropping the input data.



#### 28.5.5 Input and Output Data Format

Sensor could output YcbCr422, RGB565, Bayer format or JPEG bit-stream. However, Capture sensor interface only support YcbCr422 and RGB565. The input format depends on the sensor initial table. Programmer can specify the output format by INFMT(CAP\_PAR[0]).

Output format depends on the display device or the input data of JPEG encoder. Programmer can specify the output format by OUTFMT(CAP\_PAR[5:4]).

#### 28.5.6 Downscale Factor

Capture sensor interface controller supports to resize the input data by direct-drop algorithm (DDA).

For example:

If the dimension of cropping window is equal to 640x480 and target dimension is equal to 352x288.

The horizontal downscale factor =

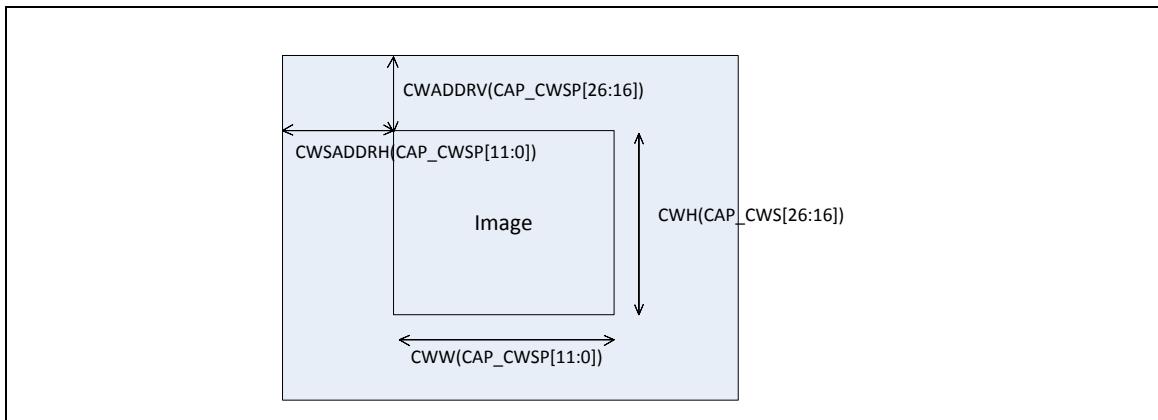
$$352/640 = (\text{PKTSHNH} << 8 + \text{PKTSHNL}) / (\text{PKTSHMH} << 8 + \text{PKTSHML})$$

The vertical downscale factor =

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

### 28.5.7 Cropping Window and Start Position

The capture interface can select a window from the received image. The size of the window is specified by the number of pixel clocks (horizontal dimension) and the number of lines (vertical dimension). The start (left upper corner) coordinates can be specified by the CAP\_CWSP register. The size (vertical dimension in number of lines and horizontal dimension in number of pixel clocks) can be specified by the CAP\_CWS register.



### 28.5.8 One Shutter Mode (Single Frame)

In this mode, a single frame is captured. After the SHUTTER (CAP\_CTL[16]) bit is set, the Image Capture interface automatically disables the capture interface after a frame is captured.

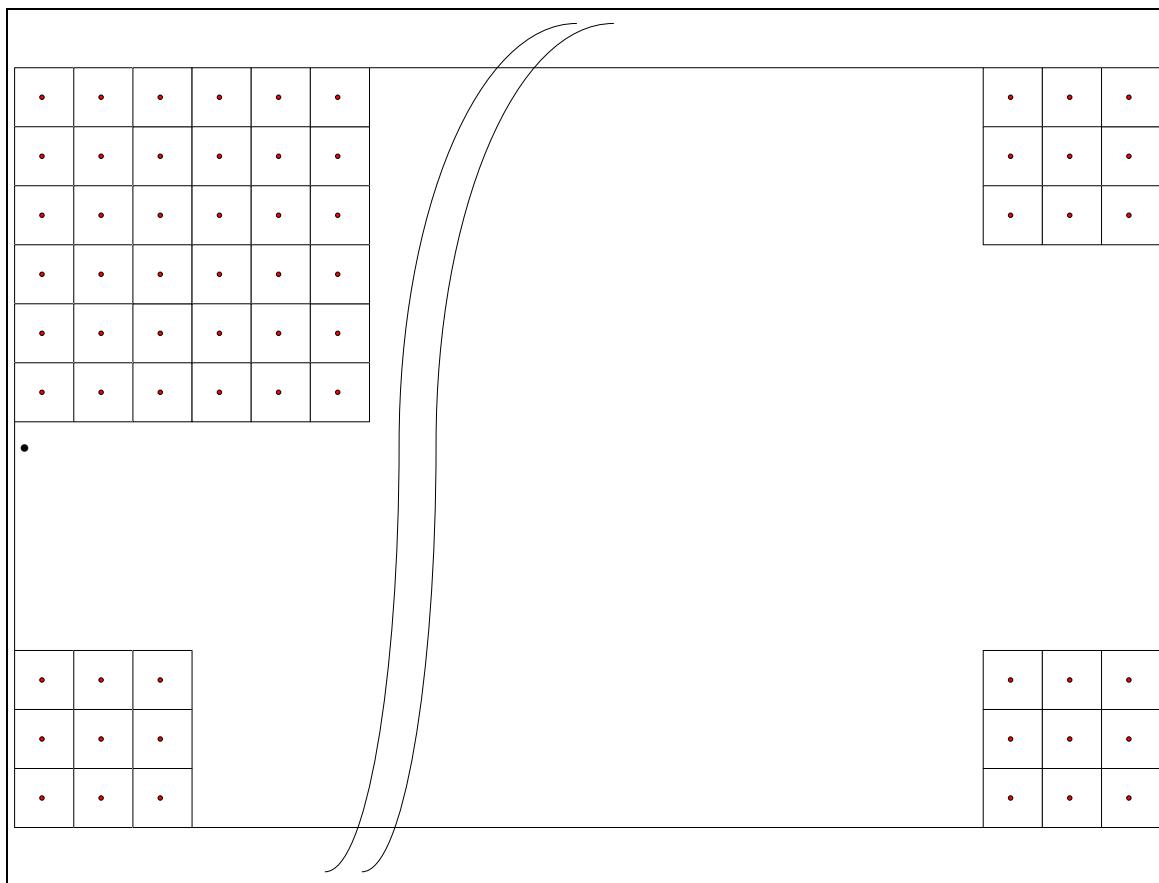
### 28.5.9 Motion detection

Capture sensor interface controller supports **in-door** motion detection. The feature lists as follows

1. Block size supports 8x8 and 16x16
2. Output motion detection supports 1 bit or 8 bit(1 bit DIFF and 7 bit threshold)

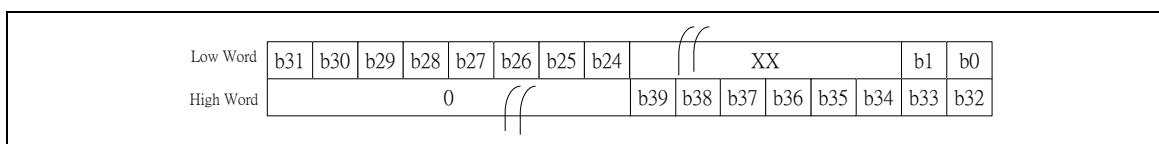
The following figure illustrates how motion detection block works. Motion detection block separates whole frame into 8x8 or 16x16 blocks. Get the central pixel (4,4) or (8,8) for block size 8x8 or 16x16. Then compare with previous frame for same position –**MDYADDR** (the temporary Y buffer of motion detection). If the difference is over the threshold set 1 to motion detection output buffer- **MDADDR**, otherwise set to be 0 to motion detection output buffer- **MDADDR**. Output the central pixel to the temporary Y buffer of motion detection. It will be

padding 0 if the output stream is not enough one word.

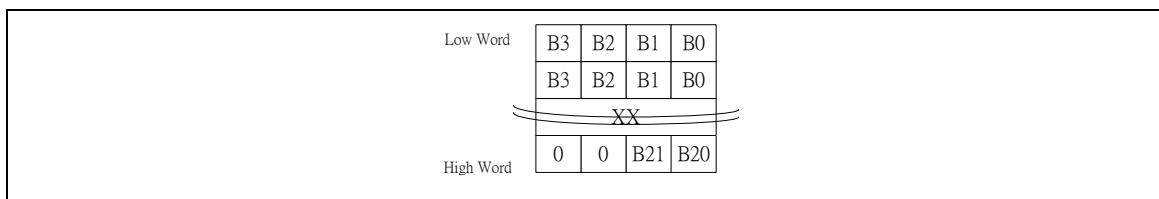


The format of motion detection output buffer lists as following.

- One bit mode (Captured Width = 640 for block size 16x16 )



- 1 bit DIFF(MSB) + 7 bit Y Differential (Captured Width = 352 for block size 16x16 )



## 29 Watchdog Timer (WDT)

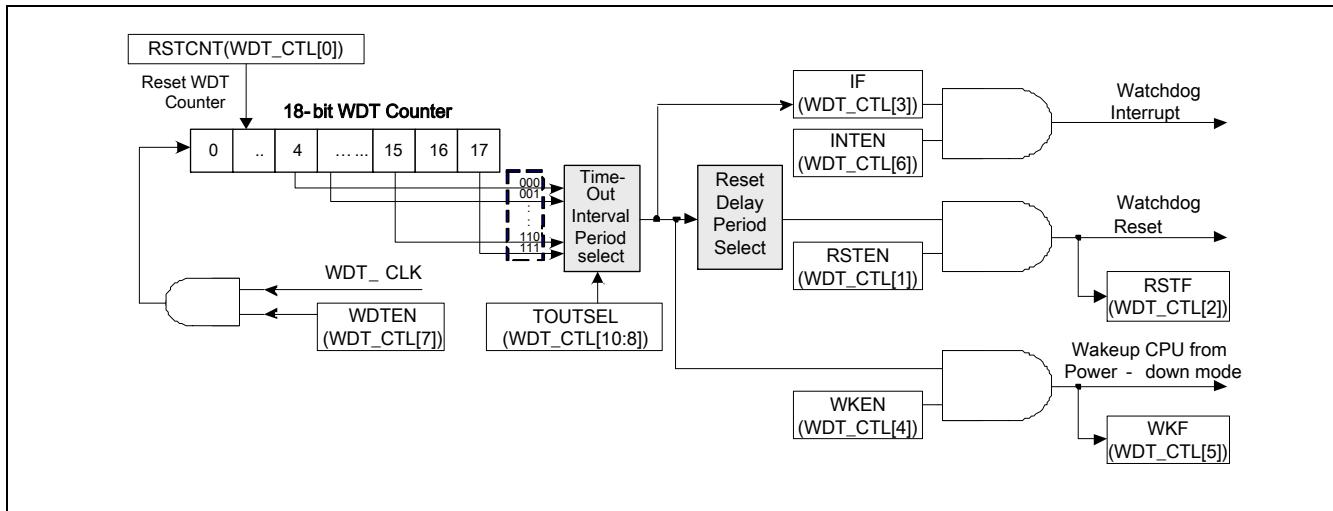
### 29.1 Overview

The purpose of Watchdog Timer (WDT) is to perform a system reset when system runs into an unknown state. This prevents system from hanging for an infinite period of time. Besides, this Watchdog Timer supports the function to wake-up system from Idle/Power-down mode.

### 29.2 Features

- 18-bit free running up counter for WDT time-out interval.
- Selectable time-out interval ( $2^4 \sim 2^{18}$ ) and the time-out interval is 0.48828125 ms ~ 8 s if WDT\_CLK = 32.768 kHz.
- System kept in reset state for a period of  $(1 / \text{WDT\_CLK}) * 63$ .
- Supports selectable WDT reset delay period, including 1026, 130, 18 or 3 WDT\_CLK reset delay period.
- Supports to force WDT enabled after chip powered on or reset by setting WDTON in PWRON register.
- Supports WDT time-out wake-up function only if WDT clock source is selected as 32 kHz.

### 29.3 Block Diagram



### 29.4 Register Map

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

<b>WDT Base Address:</b> <b>WDT_BA = 0xB800_1800</b>				
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

## 29.5 Functional Description

### 29.5.1 WDT Configuration

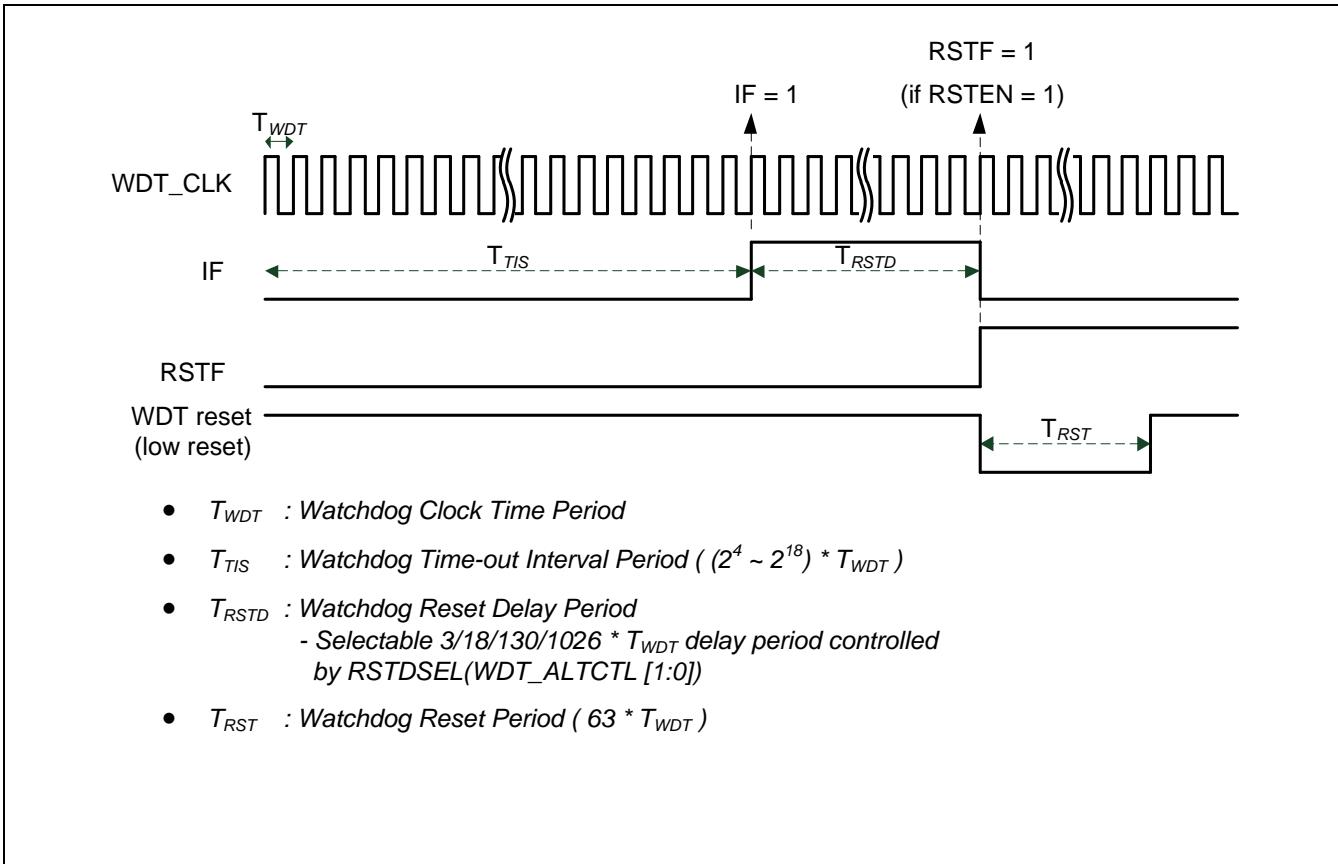
Watchdog timer is used to trigger a system reset while the software execute to an abnormal state, this prevents the system stays in an uncontrollable state for unlimited duration. Besides, WDT support wakeup function that can wake up CPU from power-down state and right before CPU enters power-down state, its counter will reset automatically, so the wakeup duration is predictable. The WDT includes an 18-bit free running up counter with programmable time-out intervals. Table below shows the WDT time-out interval period selection, Twdt in the table depends on the peripheral clock source selection through WDT\_S(CLK\_DIVCTL8[9:8]), it can be HXT (external high speed 12MHz crystal), 12MHz/128, PCLK/4096, or LXT (external low speed 32KHz crystal).

TOUTSEL (WDT_CTL[10:8])	Timeout Interval Period	WDT_CLK=HXT	WDT_CLK=LXT
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

Setting WDTEN (WDT\_CTL[7]) to 1 will enable the WDT function and the WDT counter to start counting up. When the WDT up counter reaches the TOUTSEL (WDT\_CTL[10:8]) settings, WDT time-out interrupt will occur then WDT time-out interrupt flag IF (WDT\_CTL[3]) will be set to 1 immediately, if INTEN(WDT\_CTL[6]) is set 1, timeout event will also triggers interrupt. Software must set RSTCNT (WDT\_CTL[0]) bit to reset WDT counter within the reset delay period which is configured by RSTDSEL (WDT\_ALTCTL[1:0]) to prevent system reset. There are eight time-out interval period can be selected by setting TOUTSEL (WDT\_CTL[10:8]). If RSTEN (WDT\_CTL[1]) is 1, and WDT counter is not reset before reset delay period times out, WDT will set WDTRSTS (SYS\_RSTSTS[5]) but and reset CPU. This reset signal will last for 63 WDT clocks (TRST), and then CPU resets. WDTRSTS flag will not be cleared by WDT reset signal. User could check the status of this flag to decide if the

system source is WDT or not.

Next figure shows the Watchdog Timer Time-out Interval and Reset Period Timing.



NOTE: If WDT is not enabled by power-on setting but rather enabled by software after system boot up, then WDT timeout cannot reset the system successfully.

### 29.5.2 WDT Wakeup

If WDT clock source is selected to 32 kHz, system can be waken-up from Power-down mode while WDT time-out interrupt signal is generated and WKEN (WDT\_CTL[4]) enabled. Notice that user should set XTAL\_EN (CLK\_PMCON [0]) to enable crystal clock source before system enters power down mode because the system peripheral clock are disabled when system is power down mode. In the meanwhile, the WDT (SYS\_WKUPSSR[28]) will set to 1 automatically, user can check WDT (SYS\_WKUPSSR[28]) status by software to recognize the system has been waken-up by WDT time-out interrupt or not.

## 30 Window Watchdog Timer (WWDT)

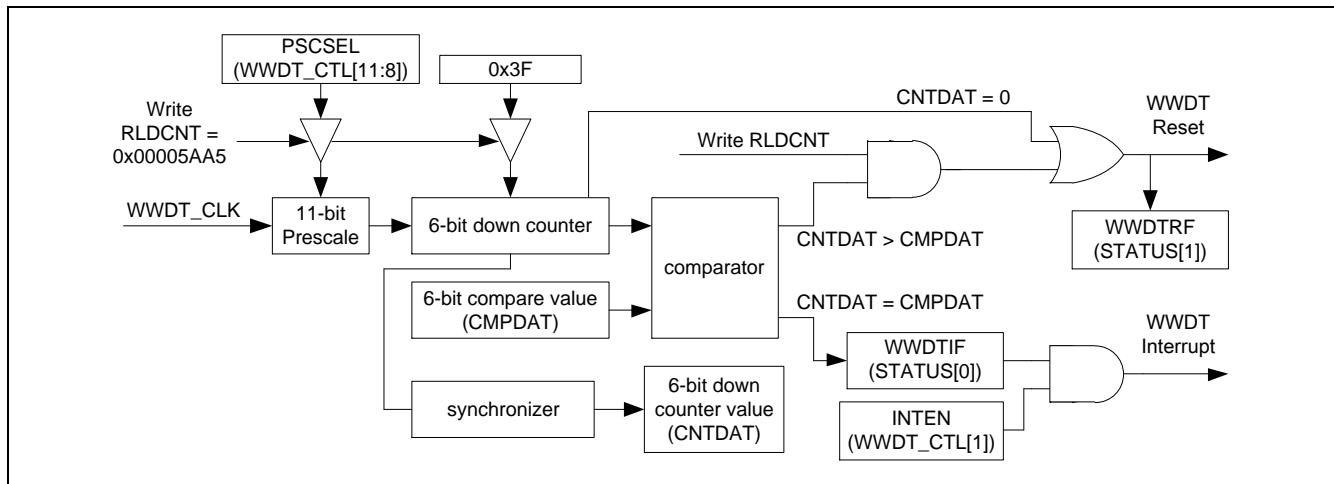
### 30.1 Overview

The Window Watchdog Timer (WWDT) is used to perform a system reset within a specified window period to prevent software run to uncontrollable status by any unpredictable condition.

### 30.2 Features

- 6-bit down counter value (CNTDAT) and 6-bit compare value (CMPDAT) to make the WWDT time-out window period flexible.
- Supports 4-bit value (PSCSEL) to programmable maximum 11-bit prescale counter period of WWDT counter.

### 30.3 Block Diagram



### 30.4 Register Map

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

## 30.5 Function Description

### 30.5.1 Timeout Setting

The WWDT includes a 6-bit down counter with programmable prescale value to define different WWDT time-out intervals. The clock source of 6-bit WWDT is based on system clock divide 4096 (PCLK/4096), external 12 MHz oscillator or internal 32 kHz oscillator with a programmable 11-bit prescale counter value which controlled by PSCSEL (WWDT\_CTL[11:8]). Also, the correlate of PSCSEL (WWDT\_CTL[11:8]) and prescale value are listed in the following table. :

PSCSEL	Prescaler Value	Max. Timeout Period	Max. Time-out Interval WWDT_CLK=HXT	Max. Time-out Interval WWDT_CLK=LXT
0000	1	$1 * 64 * T_{WWDT}$	5.33 $\mu$ s	1.95 ms
0001	2	$2 * 64 * T_{WWDT}$	10.66 $\mu$ s	3.91 ms
0010	4	$4 * 64 * T_{WWDT}$	21.33 $\mu$ s	7.81 ms
0011	8	$8 * 64 * T_{WWDT}$	42.67 $\mu$ s	15.63 ms
0100	16	$16 * 64 * T_{WWDT}$	85.33 $\mu$ s	31.25 ms
0101	32	$32 * 64 * T_{WWDT}$	170.67 $\mu$ s	62.50 ms
0110	64	$64 * 64 * T_{WWDT}$	341.33 $\mu$ s	125.00 ms
0111	128	$128 * 64 * T_{WWDT}$	682.67 $\mu$ s	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

When the WWDTEN (WWDT\_CTL[0]) is set, WWDT down counter will start counting from 0x3F to 0 and cannot be stopped. Software can read current counter value from WWDT\_CNT register.

If WWDT counter reaches 0, WWDT will trigger a system reset. Before WWDT counter reaches 0, software can write a specific value, 0x00005AA5, to register WWDTRLD to reload counter to its initial value 0x3F and prevent WWDT reset. This reload can only be set while counter value is smaller or equal to WINCMP. If software write WWDTRLD will cause system reset when WWDT counter is greater than WINCMP.

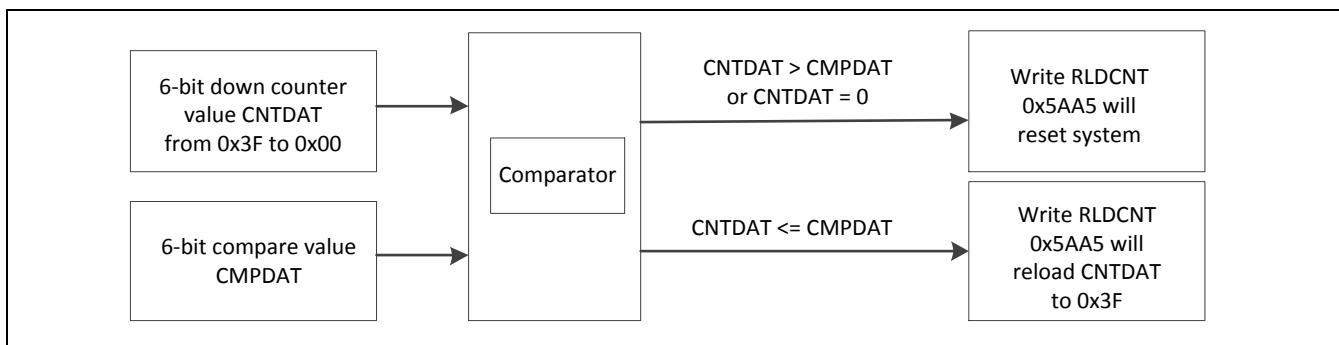
To prevent program runs to disable WWDT counter counting unexpected, the WWDT\_CTL register can only be written once after chip is powered on or reset. User cannot disable WWDT counter counting (WWDTEN[0]), change counter prescale period (PSCSEL) or change window compare value (CMPDAT) while WWDTEN (WWDT\_CTL[0]) has been enabled by user unless chip is reset.

### 30.5.2 WWDT Interrupt

During down counting by the WWDT counter, the WWDTIF (WWDT\_STATUS[0]) is set to 1 while the WWDT counter value (CNTDAT) is equal to window compare value (CMPDAT) and WWDTIF can be cleared by user by writing 1 to this bit. If INTEN (WWDT\_CTL[1]) is also set to 1 by user, the WWDT compare match interrupt signal is generated also while WWDTIF is set to 1 by hardware..

### 30.5.3 System Reset

When WWDTIF (WWDT\_STATUS[0]) is generated, user must reload WWDT counter value to 0x3F by writing 0x00005AA5 to WWDT\_RLDCNT register, and also to prevent WWDT counter value reached to 0 and generate WWDT reset system signal to info system reset. If current CNTDAT (WWDT\_CNT[5:0]) is larger than CMPDAT (WWDT\_CTL[21:16]) and user writes 0x00005AA5 to the WWDT\_RLDCNT register, the WWDT reset system signal will be generated immediately to cause chip reset also. User can check if WWDT caused system reset or not be checking WWDTRF (WWDT\_STATUS[1]) bit. If this bit is set 1, it means system was reset by WWDT. Software can write 1 to clear this bit.



### 30.5.4 WWDT Window Setting Limitations

When user writes 0x00005AA5 to WWDT\_RLDCNT register to reload WWDT counter value to 0x3F, it needs 3 WWDT clocks to sync the reload command to actually perform reload action. Notice that if user set PSCSEL (WWDT\_CTL[11:8]) to 0000, the counter prescale value should be as 1, and the CMPDAT (WWDT\_CTL[21:16]) must be larger than 2. Otherwise, writing WWDT\_RLDCNT register to reload WWDT counter value to 0x3F is unavailable, WWDTIF(WWDT\_STATUS[0]) is generated, and WWDT reset system event

always happened. Following table list the prescale value and CMPDAT setting limitations:

PSCSEL (WWDT_CTL[11:8])	Prescale Value	Valid CMPDAT (WWDT_CTL[21:16]) Value
0000	1	0x3 ~ 0x3F
0001	2	0x2 ~ 0x3F
Others	Others	0x0 ~ 0x3F

And also, after system enter power-down mode, WWDT stop counting. So it is not possible to wake up system using WWDT.

**Revision History**

Date	Revision	Description
2015.7.10	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.