ARM926EJ-S Based

32-bit Microprocessor

NUC980 Programming Guide

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 NUC980 microprocessor 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

1 OVERVIEW 13

2 SYSTEM MANAGER 14

2.1 Overview 14

2.2 Functional Description 14

2.2.1 Register Write-Protection Control 14

2.2.2 Multiple Function Control 14

2.2.3 System Reset 14

2.2.4 Low Voltage Detect / Reset 14

2.2.5 IP Reset 15

2.2.6 Power Mode And Wake Up Source 15

2.2.7 USB ID Detection 16

2.3 Register Map 17

3 CLOCK CONTROLLER 19

3.1 Overview 19

3.2 Features 19

3.3 Block Diagram 20

3.4 Functional Description 21

3.4.1 Pre-Scalar Counter 21

3.4.2 Module Clock On/Off 21

3.4.3 Clock Divider 21

3.4.4 PLL Setting 21

3.5 Register Map 23

4 ADVANCED INTERRUPT CONTROLLER (AIC) 24

4.1 Overview 24

4.2 Features 24

4.3 Block Diagram 25

4.4 Functional Description 26

4.4.1 Interrupt channel configuration 26

4.4.2 Interrupt Masking 26

4.4.3 Interrupt Handling Operation Flow 26
4.4.4 ENIT Usage 27
4.4.5 Interrupt Source 29
4.5 Register Map 32
5 EXTERNAL BUS INTERFACE (EBI) 34
5.1 Overview 34
5.2 Features 34
5.3 Block Diagram 34
5.4 Functional Description 35
5.4.1 Basic Configuration 35
5.4.2 Operation and Access Time Control 36
5.5 Register Map 38
6 GENERAL-PURPOSE INPUT/OUTPUT (GPIO) 39
6.1 Overview 39
6.2 Features 39
6.3 Block Diagram 40
6.4 Functional Description 41
6.4.1 Multiple function pin Configuration 41
6.5 Register Map 44
7 PERIPHERAL DMA CONTROLLER (PDMA) 47
7.1 Overview 47
7.2 Features 47
7.3 Block Diagram 47
7.4 Functional Description 48
7.4.1 Descriptor Functional Descriptions 48
7.5 Register Map 55
8 TIMER CONTROLLER (EMR) 57
8.1 Overview 57

8.2 Features 57
8.3 Block Diagram 58
8.4 Functional Description 59
8.4.1 Timer Initialization 59
8.4.2 Timer Capture Initialization 59
8.4.3 INTERRUPT HANDLING 59
8.4.4 TIEMR FREQUENCY 59
8.4.5 ONE-SHOT MODE 60
8.4.6 PERIODIC MODE 60
8.4.7 TOGGLE MODE 61
8.4.8 CONTINUOUS MODE 61
8.4.9 Event Counting Mode 62
8.4.10 FREE COUNTING MODE 62
8.4.11 TRIGGER COUNTING MODE 63
8.4.12 COUNTER RESET MODE 63
8.4.13 CAPTURE DEBOUNCE 64
8.4.14 Inter-Timer Trigger Mode 64
8.5 Register Map 66
9 PULSE WIDTH MODULATION (PWM) 67
9.1 Overview 67
9.2 Features 67
9.3 Block Diagram 68
9.4 Functional Description 69
9.4.1 PWM Timer Operation 69
9.4.2 PWM double buffer 69
9.4.3 Periodic and One-Shot Operation 70
9.4.4 Dead-Zone Generator 70
9.4.5 PWM Timer Start Procedure 71
9.4.6 PWM Timer Stop Procedure 71
9.5 Register Map 72

10 WATCHDOG TIMER (WDT) 73
10.1 Overview 73
10.2 Features 73
10.3 Block Diagram 73
10.4 Functional Description 74
10.4.1 WDT Configuration 74
10.4.2 WDT Wakeup 75
10.5 Register Map 76
11 WINDOW WATCHDOG TIMER (WWDT) 77
11.1 Overview 77
11.2 Features 77
11.3 Block Diagram 77
11.4 Functional Description 78
11.4.1 Timeout Setting 78
11.4.2 WWDT Interrupt 78
11.4.3 System Reset 79
11.4.4 WWDT Window Setting Limitations 79
11.5 Register Map 80
12 REAL TIME CLOCK (RTC) 81
12.1 Overview 81
12.2 Features 81
12.3 Block Diagram 82
12.4 Functional Description 83
12.4.1 RTC Initiation 83
12.4.2 RTC write enable 83
12.4.3 12/24 hour Time scale Selection 83
12.4.4 Set Calendar and Time 84
12.4.5 Set Calendar and Time Alarm (Absolute) 85
12.4.6 Set Time Alarm (Relative) 86

12.4.7 Set wake-up function 87
12.4.8 Set tick interrupt 88
12.4.9 Frequency Compensation 89
12.5 Register Map 91
13 UART 92
13.1 Overview 92
13.2 Features 94
13.3 Block Diagram 95
13.4 Functional Description 97
13.4.1 Initializations 97
13.4.2 IrDA Mode 97
13.4.3 RS485 Function Mode 98
13.4.4 LIN (Local Interconnection Network) Mode 99
13.4.5 PDMA Transfer Function 100
13.4.6 UART Controller Wake-up Function 101
13.5 Register Map 104
14 SMART CARD HOST INTERFACE (SC) 105
14.1 Overview 105
14.2 Features 105
14.3 Block Diagram 106
14.4 Functional Description 107
14.4.1 Activation (Cold Reset) 107
14.4.2 Warm Reset 108
14.4.3 Deactivation 109
14.4.4 Data Format 110
14.4.5 Data Transfer 111
14.4.6 Error Signal and Character Repetition 111
14.4.7 Internal Time-out Counter 112
14.4.8 Smartcard Insert/Remove Detection 114

14.4.9 Miscellaneous Transmission Settings 114
14.4.10 UART Mode 114
14.5 Register Map 116
15 I²C 117
15.1 Overview 117
15.2 Features 117
15.3 Block Diagram 118
15.4 Functional Description 119
15.4.1 I²C Protocol 119
15.4.2 Operation Modes 119
15.4.3 Example for Random Read on EEPROM 125
15.5 Register Map 127
16 QSPI 128
16.1 Overview 128
16.2 Features 128
16.3 Block Diagram 129
16.4 Functional Description 130
16.4.1 Slave Selection 130
16.4.2 Automatic Slave Select 130
16.4.3 Dual / Quad Mode 130
16.4.4 QSPI Interrupt 133
16.4.5 Slave mode 133
16.4.6 PDMA Transfer function 133
16.4.7 QSPI Programming Example 134
16.5 Register Map 135
17 SPI 136
17.1 Overview 136
17.2 Features 136
17.3 Block Diagram 137

17.4 Functional Description 137
17.4.1 Slave Selection 137
17.4.2 Automatic Slave Select 138
17.4.3 SPI Interrupt 138
17.4.4 Slave mode 138
17.4.5 PDMA Transfer function 139
17.4.6 SPI Programming Example 139
17.5 Register Map 140
18 I²S CONTROLLER (I²S) 141
18.1 Overview 141
18.2 Features 141
18.3 Block Diagram 142
18.4 Functional Description 143
18.4.1 I²S Master/Slave Mode 143
18.4.2 I²S Source Clock Configuration 143
18.4.3 I²S Calculation and Configuration of Clock 144
18.4.4 DMA 144
18.4.5 Sequence of DMA Data 145
18.4.6 Interface Selection 146
18.4.7 PCM Interface 146
18.4.8 Data Split 147
18.5 Register Map 148
19 ETHERNET MAC CONTROLLER (EMAC) 149
19.1 Overview 149
19.2 Features 149
19.3 Block Diagram 150
19.4 Functional Description 151
19.4.1 PHY Control 151
19.4.2 CAM Configuration 152

19.4.3 Control Frame 152
19.4.4 Wake on Lan (WoL) 152
19.4.5 Packet Receive 153
19.4.6 Packet Transmit 158
19.4.7 Network Timing 164
19.4.8 Error Handling 167
19.5 Register Map 169
20 USB 2.0 DEVICE CONTROLLER (USBD) 172
20.1 Overview 172
20.2 Features 172
20.3 Block Diagram 172
20.4 Functional Description 173
20.4.1 Initialization 173
20.4.2 Interrupt Service Routine 174
20.4.3 Standard Request 174
20.4.4 Set Address Request 174
20.4.5 Get Descriptor 175
20.4.6 IN Transmission 176
20.4.7 OUT Transmission 176
20.5 Register Map 177
21 USB HOST CONTROLLER 182
21.1 Overview 182
21.2 Features 182
21.3 Block Diagram 183
21.3.1 Basic Configuration 183
21.3.2 USB Host Port 0 183
21.3.3 EHCI Controller 183
21.3.4 OHCI Controller 184
21.4 Functional Description 186

21.4.1 Initialization 186
21.4.2 Root Hub Port Routing Logic 186
21.4.3 OHCI 186
21.4.4 EHCI 193
21.5 Register Map 200
22 CAN 202
22.1 Overview 202
22.2 Features 202
22.3 Block Diagram 202
22.4 Functional Description 204
22.4.1 CAN Protocol 204
22.4.2 CAN Baud Rate Setting 204
22.4.3 CAN Module Register 206
22.4.4 Receive CAN Message 209
22.4.5 Wakeup Function 210
22.5 Register Map 210
23 FLASH MEMORY INTERFACE (FMI) 213
23.1 Overview 213
23.2 Features 213
23.3 Block Diagram 213
23.4 Functional Description 214
23.4.1 DMA and FMI Global Control 214
23.4.2 NAND Flash 214
23.4.3 SD/eMMC 217
23.5 Register Map 221
24 SECURE DIGITAL HOST CONTROLLER (SDH) 223
24.1 Overview 223
24.2 Features 223
24.3 Block Diagram 223

24.4 Functional Description 224
24.4.1 Global Control 225
24.4.2 Send Command 226
24.4.3 Get Response 226
24.4.4 Read SD Card 226
24.4.5 Write SD Card 226
24.5 Register Map 228
25 CRYPTOGRAPHIC ACCELERATOR 229
25.1 Overview 229
25.2 Features 229
25.3 Block Diagram 231
25.3.1 Data Access 232
25.4 Functional Description 233
25.4.1 PRNG 233
25.4.2 AES 233
25.4.3 SHA 234
25.4.4 ECC 235
25.4.5 RSA 238
25.5 Register Map 240
26 CAPTURE SENSOR INTERFACE CONTROLLER 248
26.1 Overview 248
26.2 Features 248
26.3 Block Diagram 248
26.4 Functional Description 249
26.4.1 Basic Configuration 249
26.4.2 Image Capture Flow Chart 249
26.4.3 Polarity and Input Data Order 249
26.4.4 Sensor Data Input Order 250
26.4.5 Input and Output Data Format 250

26.4.6 Downscale Factor 250
26.4.7 Cropping Window and Start Position 251
26.4.8 One Shutter Mode (Single Frame) 251
26.4.9 Motion detection 251
26.5 Register Map 253
27 ANALOG TO DIGITAL CONVERTER (ADC) 254
27.1 Overview 254
27.2 Features 254
27.3 Functional Description 254
27.3.1 Basic Configuration 254
27.3.2 ADC Transfer Function 254
27.3.3 ADC Timing Diagram 255
27.3.4 Normal Detection 256
27.4 Register Map 258
28 REVISION HISTORY 259

1 OVERVIEW

The NUC980 series 32-bit microprocessor is powered by the Arm926EJ-S™ processor core with 16 KB I-cache, 16 KB D-cache and MMU running up to 300 MHz. Its SDRAM interface supports SDR/DDR/DDR2/LPDDR type SDRAM running up to 150 MHz. The NUC980 series supports built-in 16 KB embedded SRAM and 16.5 KB IBR (Internal Boot ROM) for booting from USB, NAND, SD/eMMC and SPI Flash, and industrial operating temperature from -40°C to 85°C. In addition, the NUC980 series provides built-in DDR in LQFP package to ease PCB design and reduce the BOM cost.

The NUC980 series is equipped with a large number of high speed digital peripherals, such as two 10/100 Mbps Ethernet MAC supporting RMII, a USB 2.0 high speed host/device and a USB 2.0 high speed host controller, up to six USB 2.0 full speed host lite interfaces, two CMOS sensor interfaces supporting CCIR601 and CCIR656 type sensor, two SD interfaces supporting SD/SDHC/SDIO card, a NAND Flash interface supporting SLC and MLC type NAND Flash, an I^2S interface supporting I^2S and PCM protocol. Also, the NUC980 series offers a built-in hardware cryptography accelerator supporting RSA, ECC, AES, SHA, HMAC and a random number generator (RNG).

The NUC980 series provides up to ten UART interfaces, two ISO-7816-3 interfaces, a Quad-SPI interface, two SPI interfaces, up to four I^2C interfaces, four CAN 2.0B interfaces, eight channels PWM output, 8-channel 12-bit SAR ADC, six 32-bit timers, WDT (Watchdog Timer), WWDT (Window Watchdog Timer), 32.768 kHz XTL and RTC (Real Time Clock). The NUC980 series also supports two 10-channel peripheral DMA (PDMA) for automatic data transfer between memories and peripherals.

This document describes the system configuration including clock control and interrupt handling as well as the usage of each peripheral from a software engineer's perspective.

2 SYSTEM MANAGER

2.1 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

2.2 Functional Description

2.2.1 Register Write-Protection Control

Some of the system control registers need to be protect to avoid inadvertent write and disturb the chip operation. These system control registers are protect after the power-on reset until user disables register protection. For user to program these protected registers, a register protection disable sequence needs to be followed by a special programming. The register protection disable sequence is writing the data “59h”, “16h” “88h” to the register SYS_REGWPCTL continuously. Any different data value, different sequence or any other write to other address during these three data writing will abort the whole sequence.

After the protection is disabled, user can check the protection disable bit at address 0xB000_01FC bit0, 1 is protection disable, and 0 is protection enable. Then user can update the target protected register value and then write any data to the address “0xB000_01FC” to enable register protection.

2.2.2 Multiple Function Control

Each module should be set the multiple function control before starting it. Such as SPI0, user should be set the GPD2 ~ 5 for SPI0. GPD3 is SPI0_SS0, GPD2 is SPI0_CLK, GPD4 is SPI0_DATAO, and GPD5 is SPI0_DATAI. Therefore, SYS_GPD_MFPL should be filled with 0x00111100. (For each pin setting, refer to NUC980 Technical Reference Manual.)

2.2.3 System Reset

The system reset can be issued by one of the below listed events. System reset should be configured CHIPRST(SYS_AHBIPRST[0]), and CPU reset should be configured CPURST(SYS_AHBIPRST[2]). For these reset event flags can be read by SYS_RSTSTS register. Write 1 to clear this bit to 0.

- Power-On Reset, PORRSTS (SYS_RSTSTS[0])
- Low level on the /RESET pin, PINRSTS (SYS_RSTSTS[1])
- Low Voltage Reset, LVRRSTS (SYS_RSTSTS[2]) (Please refer to the 1.3.4)
- Chip Reset, CHIPRSTS (SYS_RSTSTS[3])
- CPU Reset, CPURSTS (SYS_RSTSTS[4])
- Watchdog Time Out Reset, WDTRSTS (SYS_RSTSTS[5])

2.2.4 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.

2.2.5 IP Reset

In the three registers of SYS_AHBIPRST, SYS_APBIPRST0, or SYS_APBIPRST1, the peripheral can be reset by writing 1 and writing 0 to the bit corresponding to the peripheral setting. Using TIMER0 as an example, the user can write TIMER0RST (SYS_APBIPRST0[8]) 1 then write 0 to reset TIMER0.

2.2.6 Power Mode And Wake Up Source

- Normal mode: All clocks are ON
- Power-down mode: All clocks are disabled except LXT

Chip enters to power-down mode, system waits wake-up source occurred and returns to Normal Mode.

Users can get wake-up status from SYS_WKUPSSR0 or SYS_WKUPSSR1. Power-down mode wake-up sources include: WDT, GPIO, EINT, Timer, UART, I²C, RTC, CAN, LVD, EMAC, USBH, USBD, SDH, and ADC.

SYS_WKUPSER0 or SYS_WKUPSER1 must be enabled and need to be configured each peripheral registers.

The following steps show how to use TIMER0 to wake up system.

1. Enable TIMER0 clock and select the clock source as LXT
2. Enable TIMER0 time-out interrupt and configured time-out is 100 LXT clocks.
3. Set TMR0WKEN (SYS_WKUPSER0[8]) to 1
4. Set the WKEN(TIMER0_CTL[2]) to 1, enable TIMER0 wake-up function.
5. TIMER0 starts counting, CNTEN(TIMER0_CTL[0])=1
6. Execute WFI to enter power-down mode

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

7. Wait for TIMER0 time-out interrupt to wake up system.
8. Read TMR0WK(SYS_WKUPSSR0[8]) to get wake up source.

2.2.7 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.

2.3 Register Map

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

Register	Offset	R/W	Description	Reset Value
SYS Base Address: **SYS_BA = 0xB000_0000**				
SYS_PDID	SYS_BA+0x000	R	Product Identifier Register	0x1030_D016[1]
SYS_PWRON	SYS_BA+0x004	R/W	Power-On Setting Register	0xXXXX_XXXX[2]
SYS_LVRDCR	SYS_BA+0x020	R/W	Low Voltage Reset & Detect Control Register	0x0000_0001
SYS_MISCFCR	SYS_BA+0x030	R/W	Miscellaneous Function Control Register	0x0000_0200
SYS_MISCIER	SYS_BA+0x040	R/W	Miscellaneous Interrupt Enable Register	0x0000_0000
SYS_MISCISR	SYS_BA+0x044	R/W	Miscellaneous Interrupt Status Register	0x0001_0000
SYS_WKUPSER0	SYS_BA+0x050	R/W	System Wakeup Source Enable Register0	0x0000_0000
SYS_WKUPSER1	SYS_BA+0x054	R/W	System Wakeup Source Enable Register1	0x0000_0000
SYS_WKUPSSR0	SYS_BA+0x058	R/W	System Wakeup Source Status Register 0	0x0000_0000
SYS_WKUPSSR1	SYS_BA+0x05C	R/W	System Wakeup Source Status Register 1	0x0000_0000
SYS_AHBIPRST	SYS_BA+0x060	R/W	AHB IP Reset Control Register	0x0000_0000
SYS_APBIPRST0	SYS_BA+0x064	R/W	APB IP Reset Control Register 0	0x0000_0000
SYS_APBIPRST1	SYS_BA+0x068	R/W	APB IP Reset Control Register 1	0x0000_0000
SYS_RSTSTS	SYS_BA+0x06C	R/W	Reset Source Active Status Register	0x0000_00XX
SYS_GPA_MFPL	SYS_BA+0x070	R/W	GPIOA Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPA_MFPH	SYS_BA+0x074	R/W	GPIOA High Byte Multiple Function Control Register	0x0000_0000
SYS_GPB_MFPL	SYS_BA+0x078	R/W	GPIOB Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPB_MFPH	SYS_BA+0x07C	R/W	GPIOB High Byte Multiple Function Control Register	0x0000_0000
SYS_GPC_MFPL	SYS_BA+0x080	R/W	GPIOC Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPC_MFPH	SYS_BA+0x084	R/W	GPIOC High Byte Multiple Function Control Register	0x0000_0000
SYS_GPD_MFPL	SYS_BA+0x088	R/W	GPIOD Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPD_MFPH	SYS_BA+0x08C	R/W	GPIOD High Byte Multiple Function Control Register	0x0000_0000
SYS_GPE_MFPL	SYS_BA+0x090	R/W	GPIOE Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPE_MFPH	SYS_BA+0x094	R/W	GPIOE High Byte Multiple Function Control Register	0x0000_0000
SYS_GPF_MFPL	SYS_BA+0x098	R/W	GPIOF Low Byte Multiple Function Control Register	0x0000_0000
SYS_GPF_MFPH	SYS_BA+0x09C	R/W	GPIOF High Byte Multiple Function Control Register	0x0000_0000
SYS_GPG_MFPL	SYS_BA+0x0A0	R/W	GPIOG Low Byte Multiple Function Control Register	0x0000_0000

SYS_GPG_MFPH	SYS_BA+0x0A4	R/W	GPIOG High Byte Multiple Function Control Register	0x7777_7000
SYS_DDR_DSCTL	SYS_BA+0x0F0	R/W	DDR I/O Driving Strength Control Register	0x0000_0000
SYS_PORDISCR	SYS_BA+0x100	R/W	Power-On-Reset Disable Control Register	0x0000_00XX
SYS_RSTDEBCTL	SYS_BA+0x10C	R/W	Reset Pin De-bounce Control Register	0x0000_04B0
SYS_REGWPCTL	SYS_BA+0x1FC	R/W	Register Write-Protection Control Register	0x0000_0000

Note: [1] Dependents on part number.

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

3 CLOCK CONTROLLER

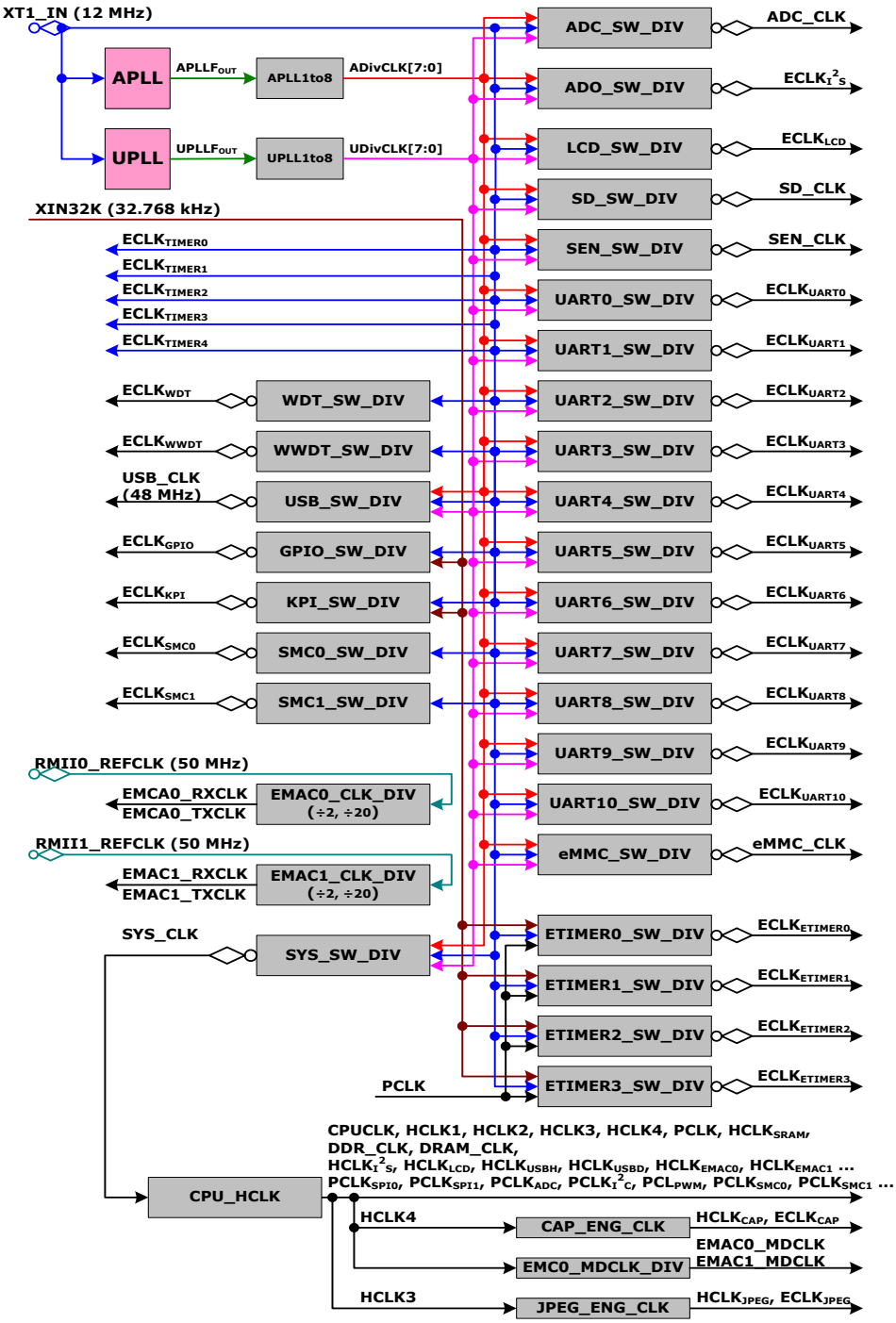
3.1 Overview

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 functionality 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 registers can also be used to control the clock enable or disable for power control.

3.2 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.

3.3 Block Diagram



Note: Before clock switching, both the pre-selected and newly selected clock sources must be turned on and stable.

Figure 3.3-1 Clock Controller Block Diagram

3.4 Functional Description

3.4.1 Pre-Scalar Counter

To avoid outputting an unstable clock to system, clock controller implements a pre-scalar counter PRESCALE(CLK_PMCON[23:8]). Crystal is stable after the PRESCALE x 256 crystal cycles, the clock controller starts to output the clock to system.

3.4.2 Module Clock On/Off

Each module of NUC980 has independent clock control. Before read/write module 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.

Using NAND controller as example, 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. Using UART0 as example, before print message from UART0, enable UART0 clock first. That is set the CLK_PCLKEN0 register UART0 bit.

3.4.3 Clock Divider

The modules which can connect external device has their 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 300 MHz, 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 300 kHz. SDH_SDIV fill 4 and SDH_N fill 199 is for this setting.
4. Data transfer frequency to 50 MHz – UPLL (300 MHz) divides by 5 becomes 50 MHz. SDH_SDIV fill 0 and SDH_N fill 5 is for this setting.

3.4.4 PLL Setting

NUC980 PLL default setting is 264 MHz. PLL frequency adjustment needs to meet the following formula.

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

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

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

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

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

4	$4.0 \leq F_{pfd} \leq 80$
5	$3.5 \leq F_{pfd} \leq 80$
6	$3.0 \leq F_{pfd} \leq 80$
7 ~ 8	$2.5 \leq F_{pfd} \leq 80$
9 ~ 10	$3.5 \leq F_{pfd} \leq 80$
11 ~ 40	$3.0 \leq F_{pfd} \leq 80$
41 ~ 128	$2.5 \leq F_{pfd} \leq 80$

Table 3.4-1 PLL Configuration

3.5 Register Map

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

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

4 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, 63 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 63 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

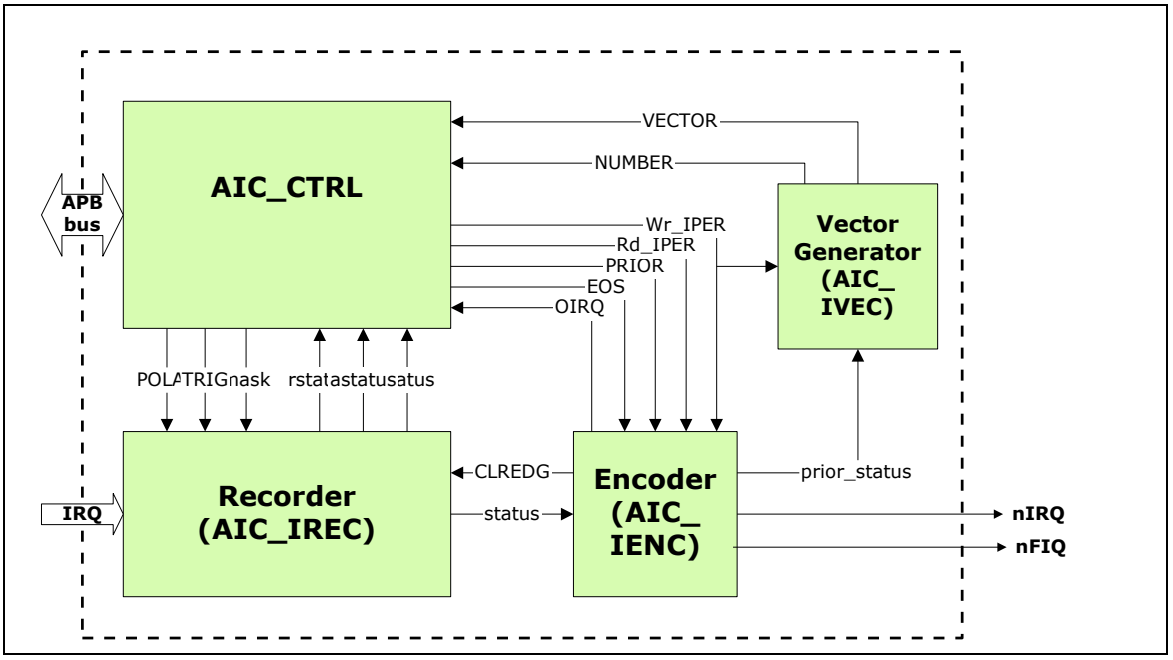


Figure 4.3-1 AIC Block Diagram

4.4 Functional Description

4.4.1 Interrupt channel configuration

Each interrupt channel has an independent source control register to set its type and priority. The interrupt type of all NUC980 Series microprocessor internal peripherals is positive-level triggered. This shouldn't be changed during normal operation. For the channel 2 and 3 that combine with 8 external interrupts nIRQ0~nIRQ7. 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(AIC_SRCCTLx).

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

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

Figure 4.4-1 Interrupt Type and Priority

4.4.2 Interrupt Masking

The NUC980 Series MCU AIC provides a set of registers to mask individual interrupt channel. The Mask Enable Command Register (AIC_INTEN0 and AIC_INTEN1) is used to enable interrupt. Write 1 to a bit of AIC_INTENx will enable the corresponding interrupt channel. Oppositely, the Mask Disable Command Register (AIC_INTDIS0 and AIC_INTDIS1) is used to disable the interrupt. Write 1 to a bit of AIC_INTDISx will disable the corresponding interrupt channel. Write 0 to a bit of AIC_INTENx or AIC_INTDISx 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_INTMSK0 or AIC_INTMSK1) 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。

4.4.3 Interrupt Handling Operation Flow

The AIC implements a proprietary 8-level priority scheme. To use this mechanism, AIC_SRCCTLx needs to be programmed before enable the interrupt channels. The AIC provides individual AIC_FIQNUM and AIC_IRQNUM interrupt source number registers to identify the source of the interrupt that is the FIQ or IRQ. The FIQ or IRQ interrupt handler can obtain its own interrupt source by reading the AIC_FIQNUM and the AIC_IRQNUM registers, respectively. Under normal circumstances there is a function table to maintain the internal device and external device interrupt service routine. When the interrupt is approved by the CPU core, the FIQ or IRQ exception handler first executes and then it calls the appropriate interrupt service routine based on the contents of AIC_FIQNUM or AIC_IRQNUM. The exception handler and interrupt service routine should follow certain rules to let this mechanism work correctly. The rules are listed below.

1. Configure interrupt source (AIC_SRCCTLx)
2. Write 1 to a bit of AIC_INTENx will enable the corresponding interrupt channel
3. Wait for the interrupt occur
4. Enter the corresponding interrupt service routine.
5. Read the AIC_FIQNUM or AIC_IRQNUM to obtain the vector, which is the current encoding interrupt channel number, which prevents a lower priority interrupt request
6. The AIC_EOIS (End of IRQ Service Command Register) must be written any value at the end of the interrupt service routine.

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

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

4.4.4 ENIT Usage

The NUC980 series provides four external interrupts (EINT0~EINT4). The following table lists the GPIOs that can be configured as EINT functions. The use of external interrupts requires the configuration of GPIO registers.

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

Table 4.4-1 EINT GPIO Mapping

The following steps to describe how to configure EINT0.

1. Set GPIO (CLK_HCLKEN[11]) as 1 to enable GPIO clock.
2. The PA.0 multi-function pin is set as the EINT0 function.
3. Set PA->MODE as 0(PA0 is input mode)
4. Set PA->INTTYPE as 1 (Level trigger interrupt).

5. Set PA->INTEN as 1 (Low level).
6. Install EINT0's IRQ service routine
7. Set the interrupt type to IRQ
8. Enable interrupt (AIC_INTENx)

In addition to the above steps, the corresponding interrupt source register PA->INTSRC needs to be cleared in the IRQ service routine.

```
void UserEINT0Init(void)
{
    /* Enable GPIO engine clock */
    outpw(REG_CLK_HCLKEN,inpw(REG_CLK_HCLKEN) | (1<<11));
    /* Configure PA.0 as EINT0 pin and enable interrupt by low level trigger */
    outpw(REG_SYS_GPA_MFPL, 0x5);
    PA->MODE = 0x0; //input mode
    PA->INTTYPE = (1 << 0); //LEVEL
    PA->INTEN = (1 << 0); // LOW LEVEL


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


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

4.4.5 Interrupt Source

The following is a list of all NUC980 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	EXT_INT0	Positive Level	External Interrupt 0
5	EXT_INT1	Positive Level	External Interrupt 1
6	EXT_INT2	Positive Level	External Interrupt 2
7	EXT_INT3	Positive Level	External Interrupt 3
8	GPA_INT	Positive Level	GPIO Port A Interrupt
9	GPB_INT	Positive Level	GPIO Port B Interrupt
10	GPC_INT	Positive Level	GPIO Port C Interrupt
11	GPD_INT	Positive Level	GPIO Port D Interrupt
12	I2S_INT	Positive Level	Audio Controller Interrupt
13	Reserved	Reserved	Reserved
14	VCAP0_INT	Positive Level	Sensor Interface Controller 0 Interrupt
15	RTC_INT	Positive Level	RTC Interrupt
16	TIMER0_INT	Positive Level	Timer 0 Interrupt
17	TIMER1_INT	Positive Level	Timer 1 Interrupt
18	ADC_INT	Positive Level	ADC Interrupt
19	EMC0RX_INT	Positive Level	EMC 0 RX Interrupt
20	EMC1RX_INT	Positive Level	EMC 1 RX Interrupt
21	EMC0TX_INT	Positive Level	EMC 0 TX Interrupt
22	EMC1TX_INT	Positive Level	EMC 1 TX Interrupt
23	EHCI_INT	Positive Level	USB 2.0 Host Controller Interrupt
24	OHCI_INT	Positive Level	USB 1.1 Host Controller Interrupt
25	PDMA0_INT	Positive Level	PDMA Channel 0 Interrupt
26	PDMA1_INT	Positive Level	PDMA Channel 1 Interrupt
27	SDH_INT	Positive Level	SD/SDIO Host Interrupt
28	FMI_INT	Positive Level	SIC Interrupt

29	UDC_INT	Positive Level	USB Device Controller Interrupt
30	TIMER2_INT	Positive Level	Timer 2 Interrupt
31	TIMER3_INT	Positive Level	Timer 3 Interrupt
32	TIMER4_INT	Positive Level	Timer 4 Interrupt
33	VCAP1_INT	Positive Level	Sensor Interface Controller 1 Interrupt
34	TIMER5_INT	Positive Level	Timer5 Interrupt
35	CRYPTO_INT	Positive Level	CRYPTO Engine Interrupt
36	UR0_INT	Positive Level	UART 0 Interrupt
37	UR1_INT	Positive Level	UART 1 Interrupt
38	UR2_INT	Positive Level	UART 2 Interrupt
39	UR4_INT	Positive Level	UART 4 Interrupt
40	UR6_INT	Positive Level	UART 6 Interrupt
41	UR8_INT	Positive Level	UART 8 Interrupt
42	CAN3_INT	Positive Level	CAN 3 Interrupt
43	UR3_INT	Positive Level	UART 3 Interrupt
44	UR5_INT	Positive Level	UART 5 Interrupt
45	UR7_INT	Positive Level	UART 7 Interrupt
46	UR9_INT	Positive Level	UART 9 Interrupt
47	I2C2_INT	Positive Level	I^2C 2 Interrupt
48	I2C3_INT	Positive Level	I^2C 3 Interrupt
49	GPE_INT	Positive Level	GPIO Port E Interrupt
50	SPI2_INT	Positive Level	SPI 2 Interrupt
51	SPI0_INT	Positive Level	SPI 0 Interrupt
52	SPI1_INT	Positive Level	SPI 1 Interrupt
53	I2C0_INT	Positive Level	I^2C 0 Interrupt
54	I2C1_INT	Positive Level	I^2C 1 Interrupt
55	SMC0_INT	Positive Level	Smart Card 0 Interrupt
56	SMC1_INT	Positive Level	Smart Card 1 Interrupt
57	GPF_INT	Positive Level	GPIO Port F Interrupt
58	CAN0_INT	Positive Level	CAN 0 Interrupt
59	CAN1_INT	Positive Level	CAN 1 Interrupt

60	PWM0_INT	Positive Level	PWM 0 Interrupt
61	PWM1_INT	Positive Level	PWM 1 Interrupt
62	CAN2_INT	Positive Level	CAN 2 Interrupt
63	GPG_INT	Positive Level	GPIO Port G Interrupt

Table 4.4-2 Interrupt Source

4.5 Register Map

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

Register	Address	R/W	Description	Reset Value
AIC_BA = 0xB800_2000				
AIC_SRCCTL0	AIC_BA+0x000	R/W	Source Control Register 0	0x4747_4747
AIC_SRCCTL1	AIC_BA+0x000	R/W	Source Control Register 1	0x4747_4747
AIC_SRCCTL2	AIC_BA+0x004	R/W	Source Control Register 2	0x4747_4747
AIC_SRCCTL3	AIC_BA+0x008	R/W	Source Control Register 3	0x4747_4747
AIC_SRCCTL4	AIC_BA+0x00C	R/W	Source Control Register 4	0x4747_4747
AIC_SRCCTL5	AIC_BA+0x010	R/W	Source Control Register 5	0x4747_4747
AIC_SRCCTL6	AIC_BA+0x014	R/W	Source Control Register 6	0x4747_4747
AIC_SRCCTL7	AIC_BA+0x018	R/W	Source Control Register 7	0x4747_4747
AIC_SRCCTL8	AIC_BA+0x01C	R/W	Source Control Register 8	0x4747_4747
AIC_SRCCTL9	AIC_BA+0x020	R/W	Source Control Register 9	0x4747_4747
AIC_SRCCTL10	AIC_BA+0x024	R/W	Source Control Register 10	0x4747_4747
AIC_SRCCTL11	AIC_BA+0x028	R/W	Source Control Register 11	0x4747_4747
AIC_SRCCTL12	AIC_BA+0x02C	R/W	Source Control Register 12	0x4747_4747
AIC_SRCCTL13	AIC_BA+0x030	R/W	Source Control Register 13	0x4747_4747
AIC_SRCCTL14	AIC_BA+0x034	R/W	Source Control Register 14	0x4747_4747
AIC_SRCCTL15	AIC_BA+0x038	R/W	Source Control Register 15	0x4747_4747
AIC_RAWSTS0	AIC_BA+0x100	R	Interrupt Raw Status Register 0	0x0000_0000
AIC_RAWSTS1	AIC_BA+0x104	R	Interrupt Raw Status Register 1	0x0000_0000
AIC_ACTSTS0	AIC_BA+0x108	R	Interrupt Active Status Register 0	0x0000_0000
AIC_ACTSTS1	AIC_BA+0x10C	R	Interrupt Active Status Register 1	0x0000_0000
AIC_INTSTS0	AIC_BA+0x110	R	Interrupt Status Register 0	0x0000_0000
AIC_INTSTS1	AIC_BA+0x114	R	Interrupt Status Register 1	0x0000_0000
AIC_IRQNUM	AIC_BA+0x120	R	IRQ Source Number Register	0x0000_0000
AIC_FIQNUM	AIC_BA+0x124	R	FIQ Source Number Register	0x0000_0000
AIC_INTMSK0	AIC_BA+0x128	R	Interrupt Mask Register 0	0x0000_0000
AIC_INTMSK1	AIC_BA+0x12C	R	Interrupt Mask Register 1	0x0000_0000
AIC_INTEN0	AIC_BA+0x130	W	Interrupt Mask Enable Command Register 0	Undefined
AIC_INTEN1	AIC_BA+0x134	W	Interrupt Mask Enable Command Register 1	Undefined
AIC_INTDIS0	AIC_BA+0x138	W	Interrupt Mask Disable Command Register 0	Undefined
AIC_INTDIS1	AIC_BA+0x13C	W	Interrupt Mask Disable Command Register 1	Undefined
AIC_EOIS	AIC_BA+0x150	W	End of IRQ Service Command Register	Undefined

AIC_EOFS	AIC_BA+0x154	W	End of FIQ Service Command Register	Undefined

5 EXTERNAL BUS INTERFACE (EBI)

5.1 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 3 chip select signals to select different devices with 20-bit address bus. It supports 8-bit and 16-bit external data bus width for each bank

5.2 Features

- Support SRAM and external I/O devices.
- Supports up to three memory banks.
- Supports dedicated external chip select pin with polarity control for each bank
- Supports accessible space up to 1 Mbytes for each bank, actually external addressable space is dependent on package pin out
- Supports 8-/16-bit data width
- Supports Timing parameters individual adjustment for each memory block
- Supports LCD interface i80 mode
- Supports PDMA mode
- Supports variable external bus base clock (MCLK) which based on HCLK
- Supports configurable idle cycle for different access condition: Idle of Write command finish (W2X) and Idle of Read-to-Read (R2R)
- Supports address bus and data bus separate mode

5.3 Block Diagram

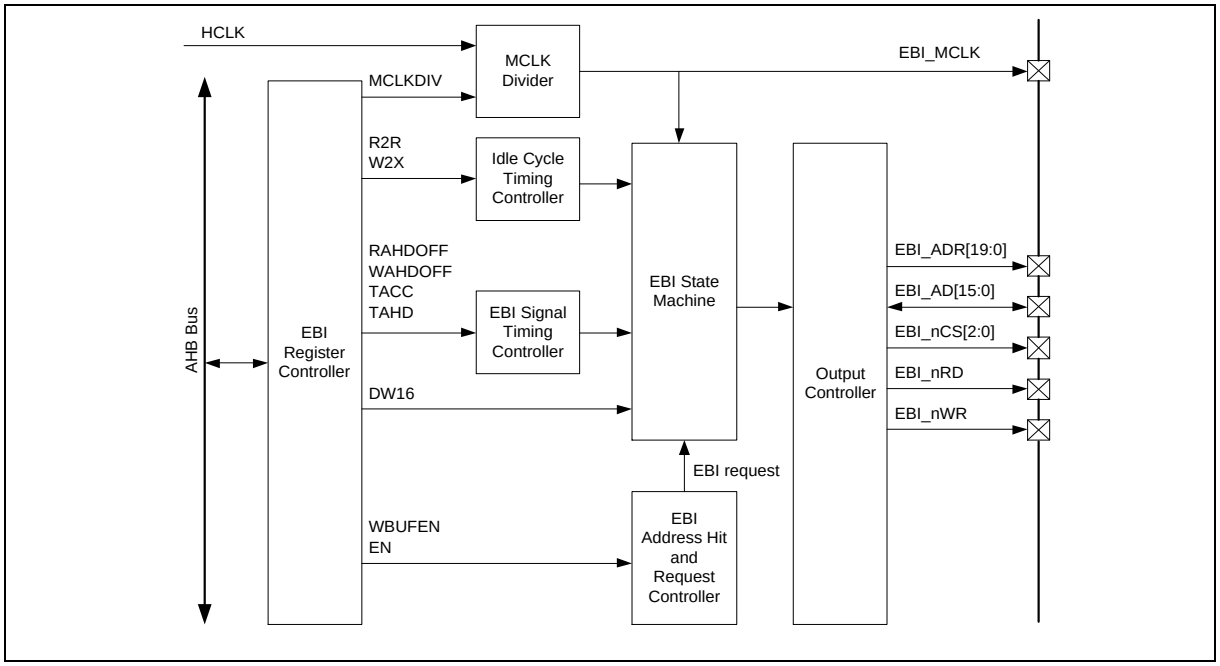


Figure 5.3-1 EBI Block Diagram

5.4 Functional Description

5.4.1 Basic Configuration

Before using External Bus Interface, it is necessary to configure related pins as the EBI function and enable EBI's clock. For EBI related pin configuration, please refer to following table 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.

Group	Pin Name	GPIO	MFP
EBI	EBI_AD0	PG. 10, PC. 0	MFP1
	EBI_AD1	PC. 1	MFP1
		PD. 12	MFP8
	EBI_AD2	PC. 2	MFP1
		PD. 13	MFP8
	EBI_AD3	PC. 3	MFP1
		PD. 14	MFP8
	EBI_AD4	PC. 4	MFP1
		PD. 15	MFP8
	EBI_AD5	PC. 5	MFP1
		PF. 0	MFP8
	EBI_AD6	PC. 6	MFP1
		PF. 1	MFP8
	EBI_AD7	PC. 7	MFP1
		PF. 2	MFP8
	EBI_AD8	PC. 8	MFP1
		PF. 3	MFP8
	EBI_AD9	PC. 9	MFP1
		PF. 4	MFP8
	EBI_AD10	PC. 10	MFP1
		PF. 5	MFP8
	EBI_AD11	PC. 11	MFP1
		PF. 6	MFP8
	EBI_AD12	PC. 12	MFP1
		PF. 7	MFP8
	EBI_AD13	PC. 13	MFP1
		PF. 8	MFP8
	EBI_AD14	PC. 14	MFP1
		PF. 9	MFP8

	EBI_AD15	PC. 15	MFP1
		PF. 10	MFP8
	EBI_ADR0	PG. 0	MFP1
	EBI_ADR1	PG. 1	MFP1
	EBI_ADR2	PG. 2	MFP1
	EBI_ADR3	PG. 3	MFP1
	EBI_ADR4	PG. 6	MFP1
	EBI_ADR5	PG. 7	MFP1
	EBI_ADR6	PG. 8	MFP1
	EBI_ADR7	PG. 9	MFP1
	EBI_ADR8	PA. 12	MFP1
	EBI_ADR9	PA. 11	MFP1
	EBI_ADR10	PA. 10	MFP1
	EBI_ADR11	PB. 8	MFP1
	EBI_ADR12	PB. 0, PG. 5	MFP1
	EBI_ADR13	PA. 13, PB. 6	MFP1
	EBI_ADR14	PA. 14, PB. 4	MFP1
	EBI_ADR15	PB. 7	MFP1
	EBI_ADR16	PB. 5	MFP1
	EBI_ADR17	PB. 1	MFP1
	EBI_ADR18	PB. 3, PG. 4	MFP1
	EBI_ADR19	PA. 5, PB. 2	MFP1
	EBI_MCLK	PA. 1	MFP2
	EBI_nCS0	PA. 9	MFP1
	EBI_nCS1	PA. 6	MFP1
	EBI_nCS2	PA. 1	MFP1
	EBI_nRD	PA. 8	MFP1
	EBI_nWR	PA. 7	MFP1

Table 5.4-1 EBI Multi Function Pin List

5.4.2 Operation and Access Time Control

The EBI mapping address is located at 0x6000_0000 ~ 0x602F_FFFF and the total memory space is 3 Mbytes. When system request address hits EBI's memory space, the corresponding EBI chip select signal is assert and EBI state machine operates.

At the start of EBI access, chip select (EBI_nCS0, EBI_nCS1 and EBI_nCS2) asserts to low and wait one EBI_MCLK for address setup time (tASU) for address stable.Then EBI_nRD asserts to low when read access or EBI_nWR asserts to low when write access. Then EBI_nRD or EBI_nWR asserts to high after keeps access time (tACC) for reading output stable or writing finish. After that, EBI signals

keep for data access hold time (tAHD) and chip select asserts to high, address is released by current access control.

The EBI controller provides a flexible timing control for different external device. In EBI timing control, tASU is fixed to 1 EBI_MCLK cycle, tAHD can modulate to 1~8 EBI_MCLK cycles by setting TAHD (EBI_TCTLx[10:8]), tACC can modulate to 1~32 EBI_MCLK cycles by setting TACC (EBI_TCTLx[7:3]). Some external device can support zero data access hold time accessing, the EBI controller can skipped tAHD to increase access speed by setting WAHDOFF (EBI_TCTLx[23]) and RAHDOFF (EBI_TCTLx[22]).

For each chip select, the EBI provides individual register with timing control, please refer to following table.

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

Table 5.4-2 EBI Timing Description

5.5 Register Map

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

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

6 GENERAL-PURPOSE INPUT/OUTPUT (GPIO)

6.1 Overview

The NUC980 series have up to 104 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 and PG. PA, PC, PD, and PG have 16 pins on port, PB has 14 pins on port, PE and PF has 13 pins on port. Each of the 104 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 104 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 kΩ ~ 82 kΩ) and a pull-down resistor (37 kΩ ~ 91 kΩ). The enable of pull-up/pull-down resistor is controllable.

6.2 Features

- Four I/O modes:
  - Quasi-bidirectional mode
  - Push-Pull Output mode
  - Open-Drain Output mode
  - Input only with high impendence mode
- TTL/Schmitt trigger input selectable
- I/O pin can be configured as interrupt source with edge/level setting
- Supports High Drive and High Slew Rate I/O mode
- I/O pin internal pull-up resistor enabled only in Quasi-bidirectional I/O mode
- Enabling the pin interrupt function will also enable the wake-up function

6.3 Block Diagram

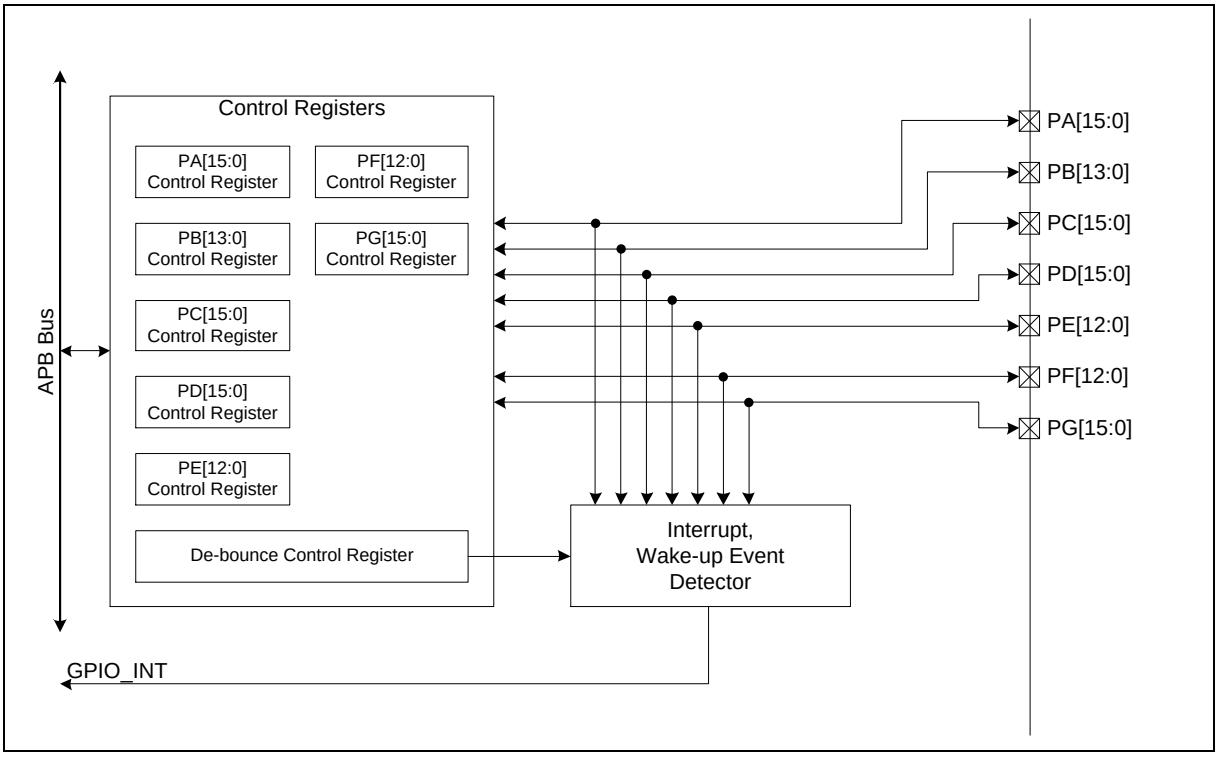


Figure 6.3-1 GPIO Block Diagram

6.4 Functional Description

6.4.1 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 and SYS_GPG_MFPH 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);
```

6.4.1.1 Input Mode

Set MODEn (Px_MODE[2n+1:2n]) to 00 as the Px.n pin is in Input mode and the I/O pin is in tri-state (high impedance) without output drive capability. The PIN (Px_PIN[n]) value reflects the status of the corresponding port pins.

6.4.1.2 Push-pull Output Mode

Set MODEn (Px_MODE[2n+1:2n]) to 01 as Px.n pin is in Push-pull Output mode and the I/O pin supports digital output function with source/sink current capability. The bit value in the corresponding DOUT (Px_DOUT[n]) is driven on the pin.

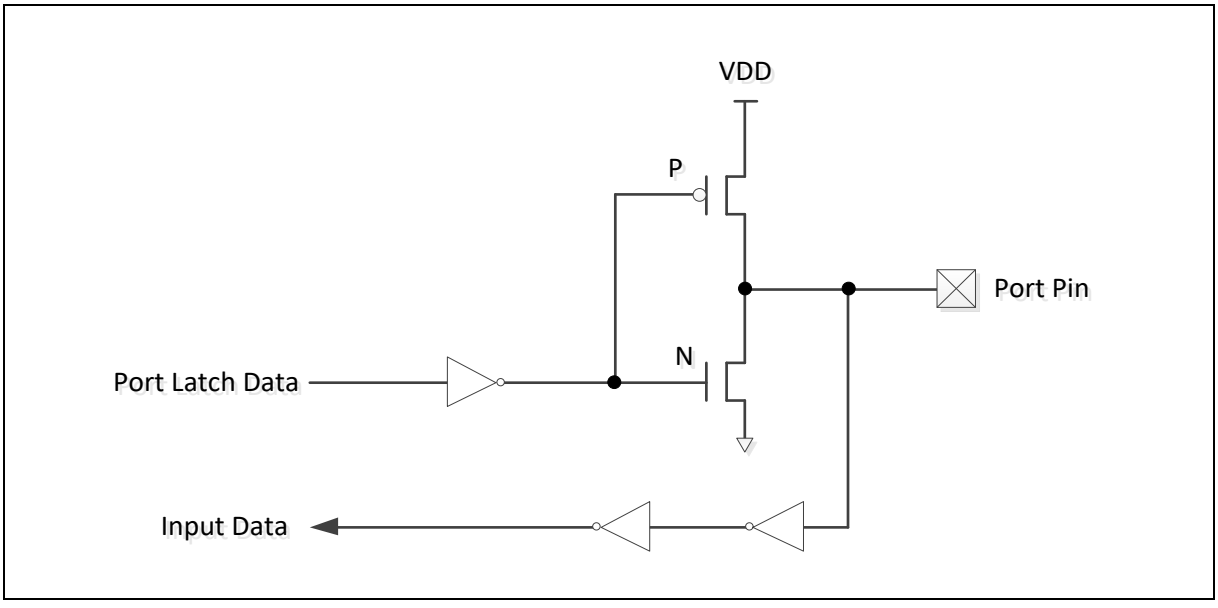


Figure 6.4-1 Push-Pull Output

6.4.1.3 Open-drain Mode

Set MODEn (Px_MODE[2n+1:2n]) to 10 the Px.n pin is in Open-drain mode and the digital output function of I/O pin supports only sink current capability, an external pull-up register is needed for

driving high state. If the bit value in the corresponding DOUT (Px_DOUT[n]) bit is 0, the pin drive a low output on the pin. If the bit value in the corresponding DOUT (Px_DOUT[n]) bit is 1, the pin output drives high that is controlled by external pull high resistor.

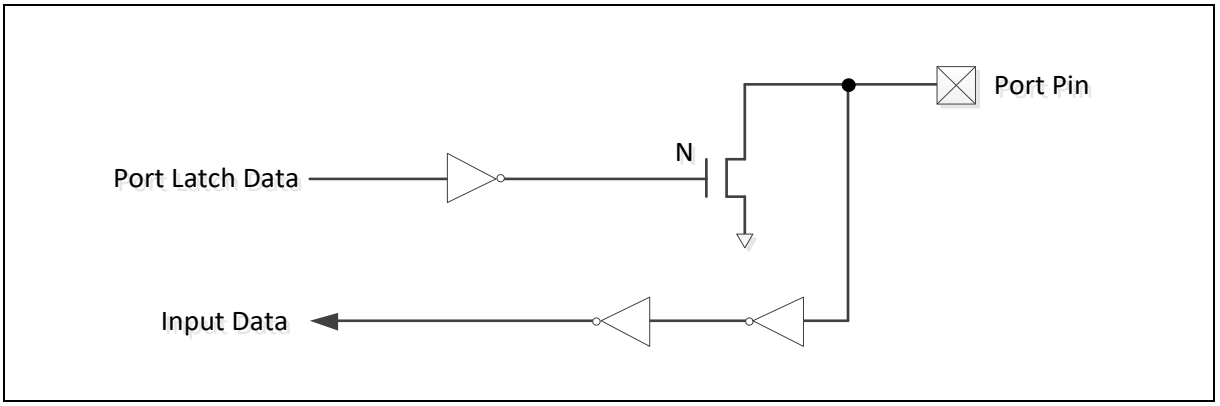


Figure 6.4-2 Open-Drain Output

6.4.1.4 Quasi-bidirectional Mode

Set MODEn (Px_MODE[2n+1:2n]) to 11 as the Px.n pin is in Quasi-bidirectional mode and the I/O pin supports digital output and input function at the same time but the source current is only up to hundreds uA. Before the digital input function is performed the corresponding DOUT (Px_DOUT[n]) bit must be set to 1. The quasi-bidirectional output is common on the 80C51 and most of its derivatives. If the bit value in the corresponding DOUT (Px_DOUT[n]) bit is 0, the pin drive a low output on the pin. If the bit value in the corresponding DOUT (Px_DOUT[n]) bit is 1, the pin will check the pin value. If pin value is high, no action takes. If pin state is low, the pin will drive strong high with 2 clock cycles on the pin and then disable the strong output drive. Meanwhile, the pin status is controlled by internal pull-up resistor. Note that the source current capability in quasi-bidirectional mode is only about 200 uA to 30 uA for V_{DD} is form 5.0 V to 2.5 V.

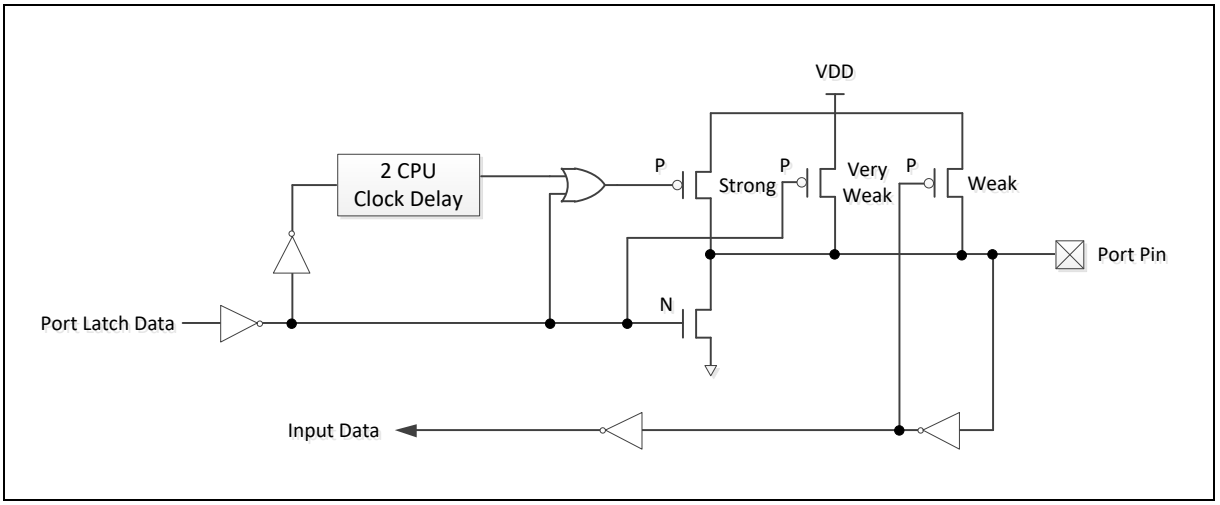


Figure 6.4-3 Quasi-Bidirectional I/O Mode

6.4.1.5 Schmitt trigger mode

The Schmitt Trigger is a logic input type that provides two different threshold voltage levels for rising and falling edge. This is useful when we have noisy input signals from which we want to get square wave signals.

6.4.1.6 Pull-up/Pull-down mode

Using pull-up/pull-down control to ensure the input state. The pull-up control only valid when MODE (Px_MODE[n])) set as tri-state and open-drain mode. The pull-down control only valid when MODE (Px_MODE[n])) set as tri-state mode.

6.4.1.7 GPIO Digital Input Path Disable Control

User can disable GPIO digital input path by setting DINOFF (Px_DINOFF[n]) to avoid input current leakage. When GPIO digital input path is disabled, the digital input pin value PIN (Px_PIN[n]) is tied to low.

6.4.1.8 GPIO Interrupt and Wake-up Function

Each GPIO pin can be set as chip interrupt source by setting correlative RHIEN (Px_INTEN[n+16])/ FLIEN (Px_INTEN[n]) bit and TYPE (Px_INTTYPE[n]). There are five types of interrupt condition can be selected: low level trigger, high level trigger, falling edge trigger, rising edge trigger and both rising and falling edge trigger. For edge trigger condition, user can enable input signal de-bounce function to prevent unexpected interrupt happened which caused by noise. The de-bounce clock source and sampling cycle period can be set through DBCLKSRC (GPIO_DBCTL[4]) and DBCLKSEL (GPIO_DBCTL[3:0]) register.

The GPIO can also be the chip wake-up source when chip enters Idle/Power-down mode. The setting of wake-up trigger condition is the same as GPIO interrupt trigger.

6.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
GPIO Base Address: **GPIO_BA = 0xB000_4000**				
PA_MODE	GPIO_BA+0x000	R/W	PA I/O Mode Control	0xXXXX_XXXX
PA_DINOFF	GPIO_BA+0x004	R/W	PA Digital Input Path Disable Control	0x0000_0000
PA_DOUT	GPIO_BA+0x008	R/W	PA Data Output Value	0x0000_FFFF
PA_DATMSK	GPIO_BA+0x00C	R/W	PA Data Output Write Mask	0x0000_0000
PA_PIN	GPIO_BA+0x010	R	PA Pin Value	0x0000_XXXX
PA_DBEN	GPIO_BA+0x014	R/W	PA De-Bounce Enable Control Register	0x0000_0000
PA_INTTYPE	GPIO_BA+0x018	R/W	PA Interrupt Trigger Type Control	0x0000_0000
PA_INTEN	GPIO_BA+0x01C	R/W	PA Interrupt Enable Control Register	0x0000_0000
PA_INTSRC	GPIO_BA+0x020	R/W	PA Interrupt Source Flag	0x0000_XXXX
PA_SMTEN	GPIO_BA+0x024	R/W	PA Input Schmitt Trigger Enable Register	0x0000_0000
PA_SLEWCTL	GPIO_BA+0x028	R/W	PA High Slew Rate Control Register	0x0000_0000
PB_MODE	GPIO_BA+0x040	R/W	PB I/O Mode Control	0xXXXX_XXXX
PB_DINOFF	GPIO_BA+0x044	R/W	PB Digital Input Path Disable Control	0x0000_0000
PB_DOUT	GPIO_BA+0x048	R/W	PB Data Output Value	0x0000_FFFF
PB_DATMSK	GPIO_BA+0x04C	R/W	PB Data Output Write Mask	0x0000_0000
PB_PIN	GPIO_BA+0x050	R	PB Pin Value	0x0000_XXXX
PB_DBEN	GPIO_BA+0x054	R/W	PB De-Bounce Enable Control Register	0x0000_0000
PB_INTTYPE	GPIO_BA+0x058	R/W	PB Interrupt Trigger Type Control	0x0000_0000
PB_INTEN	GPIO_BA+0x05C	R/W	PB Interrupt Enable Control Register	0x0000_0000
PB_INTSRC	GPIO_BA+0x060	R/W	PB Interrupt Source Flag	0x0000_XXXX
PB_SMTEN	GPIO_BA+0x064	R/W	PB Input Schmitt Trigger Enable Register	0x0000_0000
PB_SLEWCTL	GPIO_BA+0x068	R/W	PB High Slew Rate Control Register	0x0000_0000
PC_MODE	GPIO_BA+0x080	R/W	PC I/O Mode Control	0xXXXX_XXXX
PC_DINOFF	GPIO_BA+0x084	R/W	PC Digital Input Path Disable Control	0x0000_0000
PC_DOUT	GPIO_BA+0x088	R/W	PC Data Output Value	0x0000_FFFF
PC_DATMSK	GPIO_BA+0x08C	R/W	PC Data Output Write Mask	0x0000_0000
PC_PIN	GPIO_BA+0x090	R	PC Pin Value	0x0000_XXXX

PC_DBEN	GPIO_BA+0x094	R/W	PC De-Bounce Enable Control Register	0x0000_0000
PC_INTTYPE	GPIO_BA+0x098	R/W	PC Interrupt Trigger Type Control	0x0000_0000
PC_INTEN	GPIO_BA+0x09C	R/W	PC Interrupt Enable Control Register	0x0000_0000
PC_INTSRC	GPIO_BA+0x0A0	R/W	PC Interrupt Source Flag	0x0000_XXXX
PC_SMTEN	GPIO_BA+0x0A4	R/W	PC Input Schmitt Trigger Enable Register	0x0000_0000
PC_SLEWCTL	GPIO_BA+0x0A8	R/W	PC High Slew Rate Control Register	0x0000_0000
PD_MODE	GPIO_BA+0x0C0	R/W	PD I/O Mode Control	0xXXXX_XXXX
PD_DINOFF	GPIO_BA+0x0C4	R/W	PD Digital Input Path Disable Control	0x0000_0000
PD_DOUT	GPIO_BA+0x0C8	R/W	PD Data Output Value	0x0000_FFFF
PD_DATMSK	GPIO_BA+0x0CC	R/W	PD Data Output Write Mask	0x0000_0000
PD_PIN	GPIO_BA+0x0D0	R	PD Pin Value	0x0000_XXXX
PD_DBEN	GPIO_BA+0x0D4	R/W	PD De-Bounce Enable Control Register	0x0000_0000
PD_INTTYPE	GPIO_BA+0x0D8	R/W	PD Interrupt Trigger Type Control	0x0000_0000
PD_INTEN	GPIO_BA+0x0DC	R/W	PD Interrupt Enable Control Register	0x0000_0000
PD_INTSRC	GPIO_BA+0x0E0	R/W	PD Interrupt Source Flag	0x0000_XXXX
PD_SMTEN	GPIO_BA+0x0E4	R/W	PD Input Schmitt Trigger Enable Register	0x0000_0000
PD_SLEWCTL	GPIO_BA+0x0E8	R/W	PD High Slew Rate Control Register	0x0000_0000
PE_MODE	GPIO_BA+0x100	R/W	PE I/O Mode Control	0xXXXX_XXXX
PE_DINOFF	GPIO_BA+0x104	R/W	PE Digital Input Path Disable Control	0x0000_0000
PE_DOUT	GPIO_BA+0x108	R/W	PE Data Output Value	0x0000_FFFF
PE_DATMSK	GPIO_BA+0x10C	R/W	PE Data Output Write Mask	0x0000_0000
PE_PIN	GPIO_BA+0x110	R	PE Pin Value	0x0000_XXXX
PE_DBEN	GPIO_BA+0x114	R/W	PE De-Bounce Enable Control Register	0x0000_0000
PE_INTTYPE	GPIO_BA+0x118	R/W	PE Interrupt Trigger Type Control	0x0000_0000
PE_INTEN	GPIO_BA+0x11C	R/W	PE Interrupt Enable Control Register	0x0000_0000
PE_INTSRC	GPIO_BA+0x120	R/W	PE Interrupt Source Flag	0x0000_XXXX
PE_SMTEN	GPIO_BA+0x124	R/W	PE Input Schmitt Trigger Enable Register	0x0000_0000
PE_SLEWCTL	GPIO_BA+0x128	R/W	PE High Slew Rate Control Register	0x0000_0000
PE_DRVCTL	GPIO_BA+0x12C	R/W	PE High Drive Strength Control Register	0x0000_0000
PF_MODE	GPIO_BA+0x140	R/W	PF I/O Mode Control	0x0000_XXXX
PF_DINOFF	GPIO_BA+0x144	R/W	PF Digital Input Path Disable Control	0x0000_0000

PF_DOUT	GPIO_BA+0x148	R/W	PF Data Output Value	0x0000_00FF
PF_DATMSK	GPIO_BA+0x14C	R/W	PF Data Output Write Mask	0x0000_0000
PF_PIN	GPIO_BA+0x150	R	PF Pin Value	0x0000_00XX
PF_DBEN	GPIO_BA+0x154	R/W	PF De-Bounce Enable Control Register	0x0000_0000
PF_INTTYPE	GPIO_BA+0x158	R/W	PF Interrupt Trigger Type Control	0x0000_0000
PF_INTEN	GPIO_BA+0x15C	R/W	PF Interrupt Enable Control Register	0x0000_0000
PF_INTSRC	GPIO_BA+0x160	R/W	PF Interrupt Source Flag	0x0000_00XX
PF_SMTEN	GPIO_BA+0x164	R/W	PF Input Schmitt Trigger Enable Register	0x0000_0000
PF_SLEWCTL	GPIO_BA+0x168	R/W	PF High Slew Rate Control Register	0x0000_0000
PG_MODE	GPIO_BA+0x180	R/W	PG I/O Mode Control	0x0000_XXXX
PG_DINOFF	GPIO_BA+0x184	R/W	PG Digital Input Path Disable Control	0x0000_0000
PG_DOUT	GPIO_BA+0x188	R/W	PG Data Output Value	0x0000_00FF
PG_DATMSK	GPIO_BA+0x18C	R/W	PG Data Output Write Mask	0x0000_0000
PG_PIN	GPIO_BA+0x190	R	PG Pin Value	0x0000_00XX
PG_DBEN	GPIO_BA+0x194	R/W	PG De-Bounce Enable Control Register	0x0000_0000
PG_INTTYPE	GPIO_BA+0x198	R/W	PG Interrupt Trigger Type Control	0x0000_0000
PG_INTEN	GPIO_BA+0x19C	R/W	PG Interrupt Enable Control Register	0x0000_0000
PG_INTSRC	GPIO_BA+0x1A0	R/W	PG Interrupt Source Flag	0x0000_00XX
PG_SMTEN	GPIO_BA+0x1A4	R/W	PG Input Schmitt Trigger Enable Register	0x0000_0000
PG_SLEWCTL	GPIO_BA+0x1A8	R/W	PG High Slew Rate Control Register	0x0000_0000
GPIO_DBCTL	GPIO_BA+0x440	R/W	Interrupt De-bounce Control Register	0x0000_0020
PAn_PDIO n=0,1..15	GPIO_BA+0x800+(0x04 * n)	R/W	GPIO PA.n Pin Data Input/Output Register	0x0000_000X
PBn_PDIO n=0,1..15	GPIO_BA+0x840+(0x04 * n)	R/W	GPIO PB.n Pin Data Input/Output Register	0x0000_000X
PCn_PDIO n=0,1..15	GPIO_BA+0x880+(0x04 * n)	R/W	GPIO PC.n Pin Data Input/Output Register	0x0000_000X
PDn_PDIO n=0,1..15	GPIO_BA+0x8C0+(0x04 * n)	R/W	GPIO PD.n Pin Data Input/Output Register	0x0000_000X
PEn_PDIO n=0,1..14	GPIO_BA+0x900+(0x04 * n)	R/W	GPIO PE.n Pin Data Input/Output Register	0x0000_000X
PFn_PDIO n=0,1..7	GPIO_BA+0x940+(0x04 * n)	R/W	GPIO PF.n Pin Data Input/Output Register	0x0000_000X

7 PERIPHERAL DMA CONTROLLER (PDMA)

7.1 Overview

The peripheral direct memory access (PDMA) controller is used to provide high-speed data transfer. The PDMA controller can transfer data from one address to another without CPU intervention. This has the benefit of reducing the workload of CPU and keeps CPU resources free for other applications. The PDMA controller has a total of 10 channels and each channel can perform transfer between memory and peripherals or between memory and memory.

7.2 Features

- Supports 2 PDMA controller, PDMA0 and PDMA1
- Supports 10 independently configurable channels
- Supports selectable 2 level of priority (fixed priority or round-robin priority)
- Supports transfer data width of 8, 16, and 32 bits
- Supports source and destination address increment size can be byte, half-word, word or no increment
- Supports software and UART, SPI, I^2C and Timer request
- Supports Scatter-Gather mode to perform sophisticated transfer through the use of the descriptor link list table
- Supports single and burst transfer type
- Supports time-out function
- Supports stride function from channel 0 to channel 5

7.3 Block Diagram

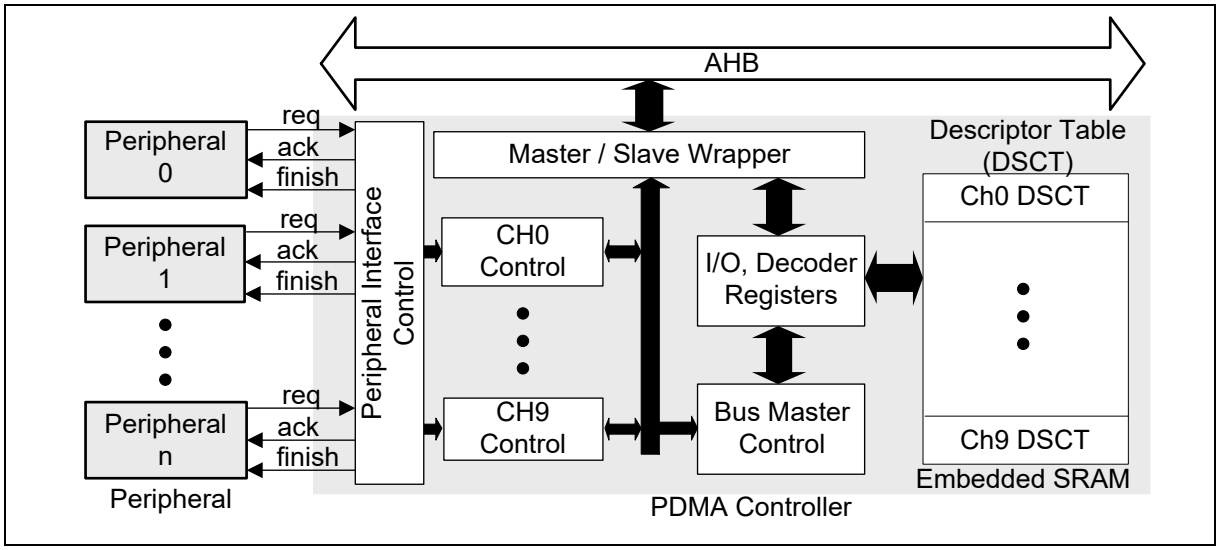


Figure 7.3-1 PDMA Block Diagram

7.4 Functional Description

7.4.1 Descriptor Functional Descriptions

7.4.1.1 Channel Priority

The PDMA controller supports two level channel priorities including fixed and round-robin priority. The fixed priority channel has higher priority than round-robin priority channel. If multiple channels are set as fixed or round-robin priority, the higher channel will have higher priority.

PDMA_PRISET	Channel Number	Priority Setting	Arbitration Priority In Descending Order
1	9	Channel9, Fixed Priority	Highest
1	8	Channel8, Fixed Priority	---
---	---	---	---
1	0	Channel0, Fixed Priority	---
0	9	Channel9, Round-Robin Priority	---
0	8	Channel8, Round-Robin Priority	---
---	---	---	---
0	0	Channel0, Round-Robin Priority	Lowest

Table 7.4-1 Descriptor Table Entry Structure

7.4.1.2 PDMA Operation Mode

The PDMA controller supports two operation modes including Basic mode and Scatter-Gather mode.

Basic Mode

Basic mode is used to perform one descriptor table transfer mode. This mode can be used to transfer data between memory and memory or peripherals and memory. PDMA controller operation mode can be set from OPMODE (PDMA_DSCTn_CTL[1:0], n denotes PDMA channel), the default setting is in idle state (OPMODE (PDMA_DSCTn_CTL[1:0]) = 0x0) and recommend user configure the descriptor table in idle state. If operation mode is not in idle state, user re-configure channel setting may make some operation error.

User must fill the transfer count TXCNT (PDMA_DSCTn_CTL[31:16]) register and select transfer width TXWIDTH (PDMA_DSCTn_CTL[13:12]), destination address increment size DAINC (PDMA_DSCTn_CTL[11:10]), source address increment size SAINC (PDMA_DSCTn_CTL[9:8]), burst size BURSIZE (PDMA_DSCTn_CTL[6:4]) and transfer type TXTYPE (PDMA_DSCTn_CTL[2]), then the PDMA controller will perform transfer operation in transfer state after receiving request signal. Finishing this task will generate an interrupt to CPU if corresponding PDMA interrupt bit INTENn (PDMA_INTEN[9:0]) is enabled and the operation mode will be updated to idle state as shown in Figure 7.4-1. If software configures the operation mode to idle state, the PDMA controller will not perform any transfer and then clear this operation request. Finishing this task will also generate an interrupt to CPU if corresponding PDMA interrupt bit is enabled.

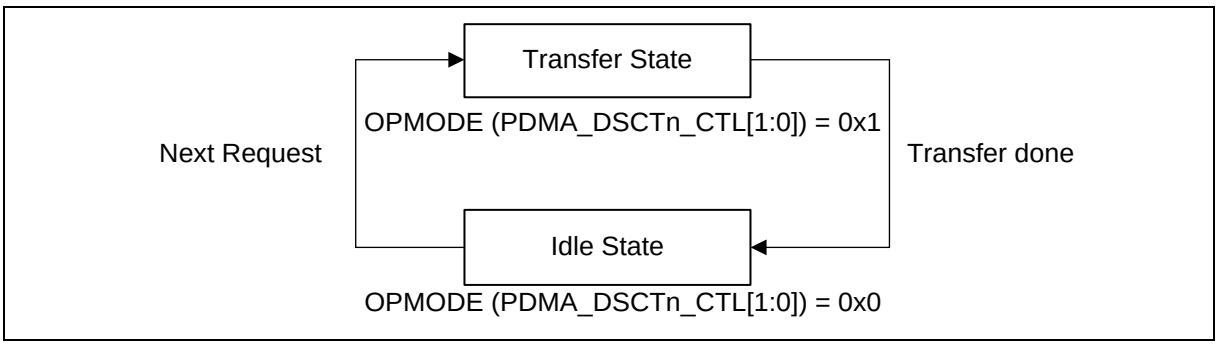


Figure 7.4-1 Basic Mode Finite State Machine

Scatter-Gather Mode

Scatter-Gather mode is a complex mode and can perform sophisticated transfer through the use of the description link list table as shown in Figure 7.4-2. Through operation mode user can perform peripheral wrapper-around, and multiple PDMA task can be used for data transfer between varied locations in system memory instead of a set of contiguous locations. Scatter-gather mode only needs a request to finish all table entries task till the last task with OPMODE (PDMA_DSCTn_CTL[1:0]) is idle state without ack. It also means scatter-gather mode can only be use to transfer data between memory to memory without handshaking.

In Scatter-Gather mode, the table is just used for jumping to the next table entry. The first task will not perform any operation transfer. Finishing each task will generate an interrupt to CPU if corresponding PDMA interrupt bit is enabled and TBINTDIS (PDMA_DSCTn_CTL[7]) bit is “0” (when finishing task and TBINTDIS bit is “0”, corresponding TDIFn (PDMA_TDSTS[9:0]) flag will be asserted and if this bit is “1” TDIFn will not be active).

If channel 9 has been triggered, and the operation mode is in Scatter-Gather mode (OPMODE (PDMA_DSCTn_CTL[1:0]) = 0x2), the hardware will load the real PDMA information task from the address generated by PDMA_DSCTn_NEXT (link address). For example, the current link address is 0x02000_0100 (32bits without last two bits [1:0] valid in PDMA_DSCTn_NEXT), and then the next DSCT entry start address is 0x2000_0100.

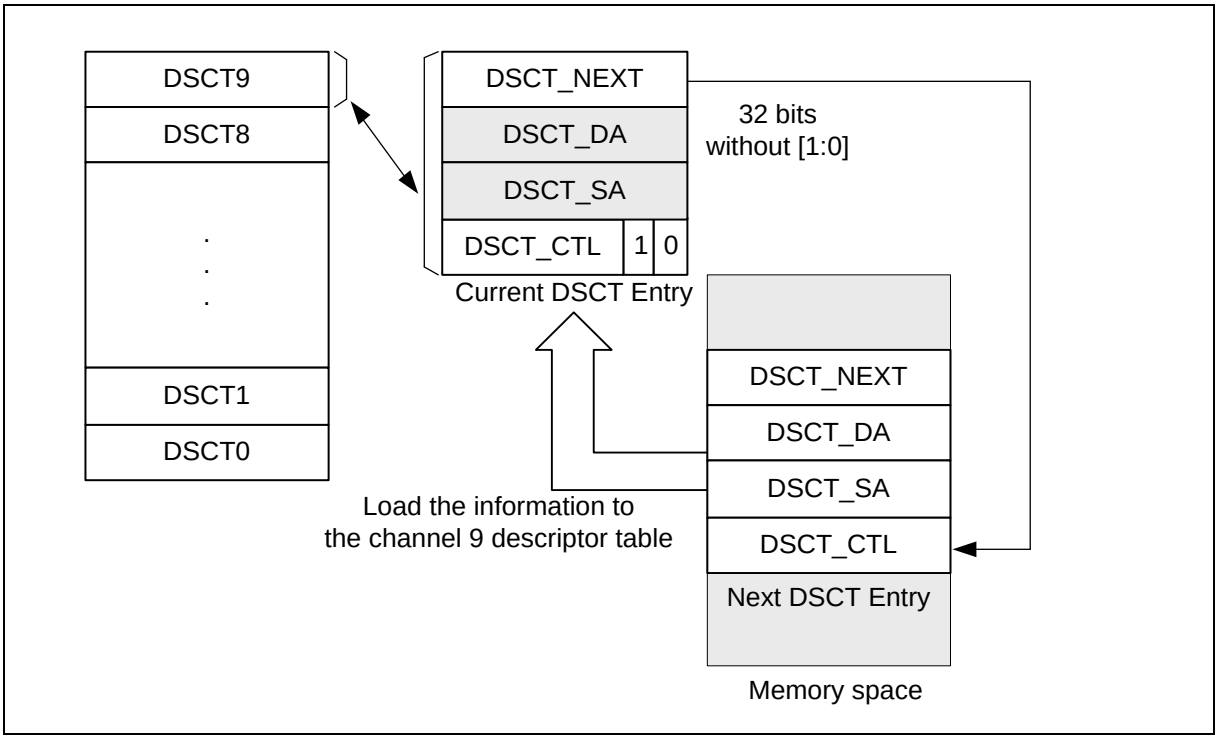


Figure 7.4-2 Descriptor Table Link List Structure

The above link list table operation is DSCT state in Scatter-Gather Mode as shown in Figure 6.7 5. When loading the information is finished, it will go to transfer state and start transfer by this information automatically. However, if the next PDMA information is also in the Scatter-Gather mode, the hardware will catch the next PDMA information block when the current task is finished. The Scatter-Gather mode switches to basic mode when doing the next task. Then, the basic mode switches to Idle state when the last task is finished.

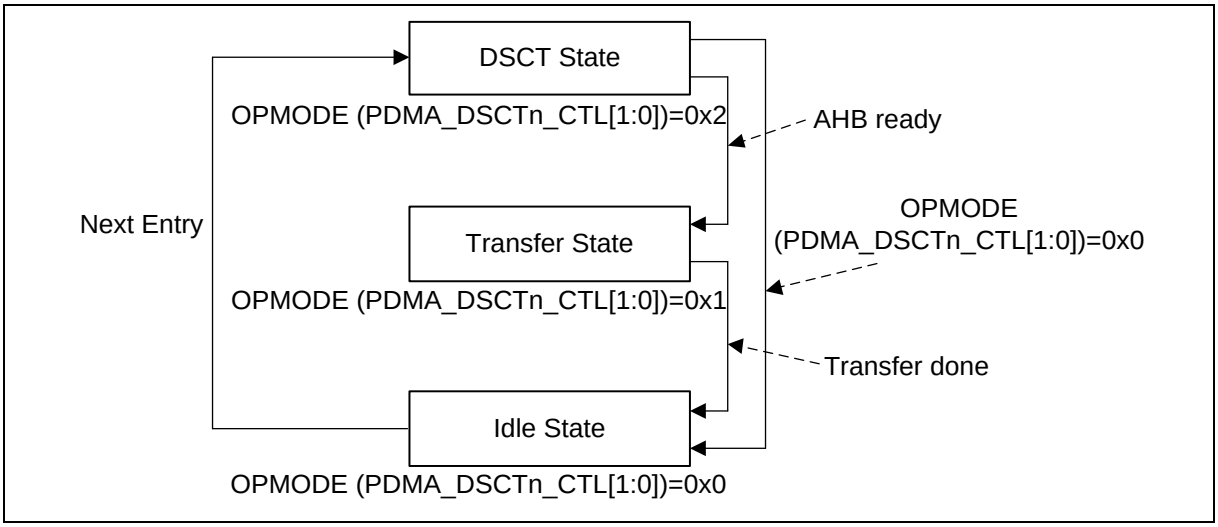


Figure 7.4-3 Scatter-Gather Mode Finite State Machine

7.4.1.3 Transfer Type

The PDMA controller supports two transfer types: single transfer type and burst transfer type, configure by setting TXTYPE (PDMA_DSCTn_CTL[2]).

When the PDMA controller is operated in single transfer type, each transfer data needs one request signal for one transfer, after transferred data, TXCNT (PDMA_DSCTn_CTL[31:16]) will decrease 1. Transfer will be finished after the TXCNT (PDMA_DSCTn_CTL[31:16]) decreases to 0. In this mode, the BURSIZE (PDMA_DSCTn_CTL[6:4]) is not useful to control the transfer size. The BURSIZE (PDMA_DSCTn_CTL[6:4]) will be fixed as one.

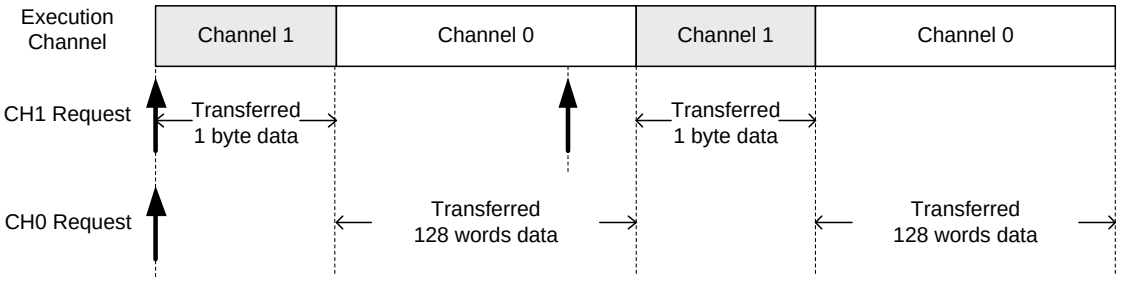
For the burst transfer type, the PDMA controller transfers TXCNT (PDMA_DSCTn_CTL[31:16]) of data and need only one request signal. After transferred BURSIZE (PDMA_DSCTn_CTL[6:4]) of data, TXCNT (PDMA_DSCTn_CTL[31:16]) will decrease BURSIZE number. Transfer will be done after the transfer count TXCNT (PDMA_DSCTn_CTL[31:16]) decreases to 0. Note that burst transfer type can only be used for PDMA controller to do burst transfer between memory and memory. User must use single request type for memory-to-peripheral and peripheral-to-memory transfers. Please note that, PDMA transfer data between Flash and memory should finish before MCU enter idle mode or power done mode to prevent access wrong data.

Figure 6.7 6 shows an example about single and burst transfer type in basic mode. In this example, channel 1 uses single transfer type and TXCNT (PDMA_DSCTn_CTL[31:16]) = 127. Channel 0 uses burst transfer type, BURSIZE (PDMA_DSCTn_CTL[6:4]) = 128 and TXCNT (PDMA_DSCTn_CTL[31:16]) = 255. The operation sequence is described below:

1. Channel 0 and channel 1 get the trigger signal at the same time.
2. Channel 1 has higher priority than channel 0 by default; the PDMA controller will load the channel 1 descriptor table first and executing. But channel 1 is single transfer type, and thus the PDMA controller will only transfer one transfer data.
3. Then, the PDMA controller turns to the channel 0 and loads channel 0's descriptor table. The channel 0 is burst transfer type and the burst size selected to 128. Therefore, the PDMA controller will transfer 128 transfer data.
4. When channel 0 transfers 128 data, channel 1 gets another request signal, then after channel 0 finishes 128 transfer data, the PDMA controller will turn to channel 1 and transfer next one data.
5. After channel 1 transfers data, the PDMA controller switches to low priority channel 0 to continuous next 128 data transfer. If no channel 1 request receives, PDMA will start next

channel 0, 128 data transfer.

6. The PDMA controller will complete transfer when channel 0 finishes data transfer 256 times, and channel 1 finishes transferring 128 times.



	CH1	CH0
TXCNT (PDMA_DSCTn_CTL[31:16])	127	255
BURSIZE (PDMA_DSCTn_CTL[6:4])	0 (1 byte)	2 (1 word)
BURSIZE (PDMA_DSCTn_CTL[6:4])	Non-useful	0 (128 transfers)
TXTYPE (PDMA_DSCTn_CTL[2])	1 (Single Transfer)	0 (Burst Transfer)

Figure 7.4-4 Example of Single Transfer Type and Burst Transfer Type in Basic Mode

7.4.1.4 Channel Time-out

When the transfer channel is enabled and selected to the peripheral, corresponding channel time-out TOUTENn (PDMA_TOUTEN [n], n=0,1..9) is enabled, then channel's corresponding time-out counter will start count up from 0 while the channel has received trigger signal from the peripheral.

The time-out counter is based on output of HCLK prescaler, which is setting by corresponding channel's TOUTPSCn (PDMA_TOUTPSC [2+4n:4n], n=0,1..9). If time-out counter counts up from 0 to corresponding channel's TOCn (PDMA_TOC0_1 [16(n+1)-1:16n], n=0,1..9), the PDMA controller will generate interrupt signal when corresponding TOUTIENn (PDMA_TOUTIEN [n], n=0,1..9) is enabled. When time-out occurred, corresponding channel's REQTOFn (PDMA_INTSTS [n+8], n=0,1..9) will be set to indicate channel time-out is happened.

Time-out counter resets to 0 while counter count to TOCn (PDMA_TOC0_1 [16(n+1)-1:16n], n=0,1..9), received trigger signal, time-out function is disabled or chip enters Power-down mode.

Figure 7.4-5 shows an example about time-out counter operation. The operation sequence is described below:

1. The channel 0 time-out counter is not counting when time-out function is enabled by setting TOUTEN0(PDMA_TOUTEN[0]) bit to 1.
2. Time-out counter starts counting from 0 to the value of TOC0(PDMA_TOC0_1[15:0]) bits when receiving the first peripheral request.

3. Time-out counter is reset to 0 by received second peripheral request.
4. Channel 0 request time-out flag(REQTOF0(PDMA_INTSTS[8])) is set to high when time-out counter counts to 5. The counter will keep counting from 0 to 5, and user can clear REQTOF0 flag and then poll REQTOF0 flag to check the next time-out occurred.
5. Time-out counter is reset to 0 when time-out function is disabled.

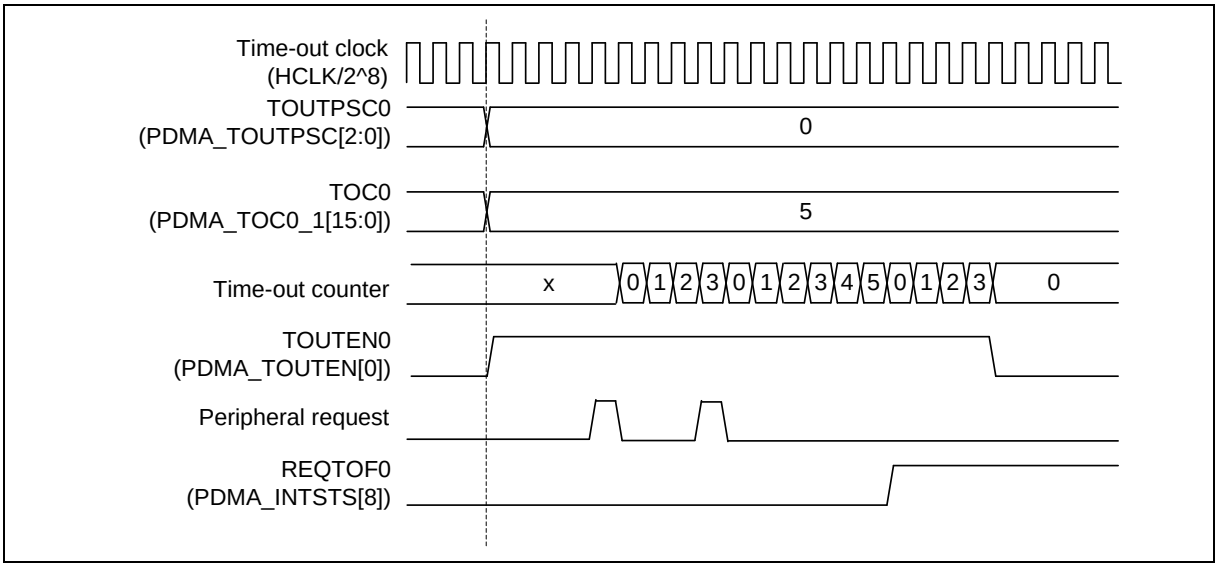


Figure 7.4-5 Example of PDMA Channel 0 Time-out Counter Operation

7.4.1.5 Stride Function

The PDMA supports channel 0 to channel 5 six channels with stride function. The stride function can transfer data from one address to another address and support block transfer with stride. When operating with stride function, the transfer address can be fixed or incremented successively.

Set STRIDEEN (PDMA_DSCTn_CTL[15]) to enable the stride function, and then write a valid source address to the PDMA_DSCTn_SA register and a source address offset count to SASOL (PDMA_ASOCRn[15:0]) register, a destination address to the PDMA_DSCTn_DA register and a destination address offset count to DASOL (PDMA_ASOCRn[31:16]), and a transfer count to the TXCNT (PDMA_DSCTn_CTL) register and a stride transfer count to STC (PDMA_STCn[15:0]). Next, trigger the SWREQn (PDMA_SWREQ[5:0]). The PDMA will start and then stop the transfer after TXCNT (PDMA_DSCTn_CTL) counts down to 0. Figure 7.4-6 shows the block transfer relationship between source memory and destination memory. The stride function also supports peripheral to memory or memory to peripheral transfer.

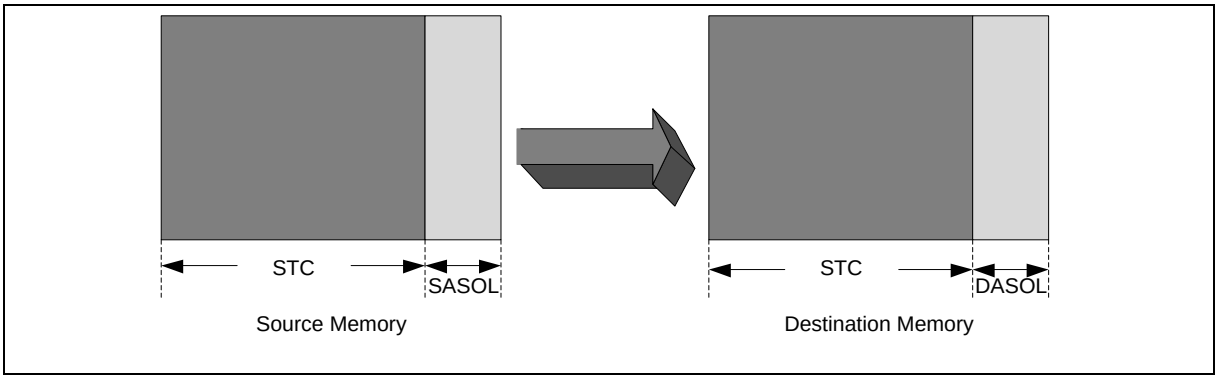


Figure 7.4-6 Stride Function Block Transfer

7.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
PDMA Base Address: **PDMAx_BA = 0xB000_8000 + (0x1000*x)** **x=0, 1** **DSCT_CTL_BA = PDMAx_BA** **DSCT_SA_BA = PDMAx_BA+4** **DSCT_DA_BA = PDMAx_BA+8** **DSCT_NEXT_BA = PDMAx_BA+c** **CURSCAT_BA = PDMAx_BA+80**				
PDMAx_DSCTn_CTL **n = 0,1..9**	DSCT_CTL_BA + 0x10 * n	R/W	Descriptor Table Control Register of PDMA Channel n	0xXXXX_XXXX
PDMAx_DSCTn_SA **n = 0,1..9**	DSCT_SA_BA + 0x10 * n	R/W	Source Address Register of PDMA Channel n	0xXXXX_XXXX
PDMAx_DSCTn_DA **n = 0,1..9**	DSCT_DA_BA + 0x10 * n	R/W	Destination Address Register of PDMA Channel n	0xXXXX_XXXX
PDMAx_DSCTn_NEXT **n = 0,1..9**	DSCT_NEXT_BA + 0x10 * n	R/W	Next Scatter-Gather Descriptor Table Offset Address of PDMA Channel n	0xXXXX_XXXX
PDMAx_CURSCATn **n = 0,1..9**	CURSCAT_BA + 0x004 * n	R	Current Scatter-Gather Descriptor Table Address of PDMA Channel n	0xXXXX_XXXX
PDMAx_CHCTL	PDMAx_BA + 0x400	R/W	PDMA Channel Control Register	0x0000_0000
PDMAx_PAUSE	PDMAx_BA + 0x404	W	PDMA Transfer Pause Control Register	0x0000_0000
PDMAx_SWREQ	PDMAx_BA + 0x408	W	PDMA Software Request Register	0x0000_0000
PDMAx_TRGSTS	PDMAx_BA + 0x40C	R	PDMA Channel Request Status Register	0x0000_0000
PDMAx_PRISET	PDMAx_BA + 0x410	R/W	PDMA Fixed Priority Setting Register	0x0000_0000
PDMAx_PRICLR	PDMAx_BA + 0x414	W	PDMA Fixed Priority Clear Register	0x0000_0000
PDMAx_INTEN	PDMAx_BA + 0x418	R/W	PDMA Interrupt Enable Register	0x0000_0000
PDMAx_INTSTS	PDMAx_BA + 0x41C	R/W	PDMA Interrupt Status Register	0x0000_0000
PDMAx_ABTSTS	PDMAx_BA + 0x420	R/W	PDMA Channel Read/Write Target Abort Flag Register	0x0000_0000
PDMAx_TDSTS	PDMAx_BA + 0x424	R/W	PDMA Channel Transfer Done Flag Register	0x0000_0000
PDMAx_ALIGN	PDMAx_BA + 0x428	R/W	PDMA Transfer Alignment Status Register	0x0000_0000
PDMAx_TACTSTS	PDMAx_BA + 0x42C	R	PDMA Transfer Active Flag Register	0x0000_0000
PDMAx_TOUTPSC	PDMAx_BA + 0x430	R/W	PDMA Time-out Prescaler Register	0x0000_0000
PDMAx_TOUTEN	PDMAx_BA + 0x434	R/W	PDMA Time-out Enable Register	0x0000_0000
PDMAx_TOUTIEN	PDMAx_BA + 0x438	R/W	PDMA Time-out Interrupt Enable Register	0x0000_0000
PDMAx_SCATBA	PDMAx_BA + 0x43C	R/W	PDMA Scatter-Gather Descriptor Table Base Address Register	0x2000_0000
PDMAx_TOC0_1	PDMAx_BA + 0x440	R/W	PDMA Time-out Counter Ch1 and Ch0 Register	0x0000_0000

PDMAx_CHRST	PDMAx_BA + 0x460	R/W	PDMA Channel Reset Register	0x0000_0000
PDMAx_REQSEL0_3	PDMAx_BA + 0x480	R/W	PDMA Request Source Select Register 0	0x0000_0000
PDMAx_REQSEL4_7	PDMAx_BA + 0x484	R/W	PDMA Request Source Select Register 1	0x0000_0000
PDMAx_REQSEL8_11	PDMAx_BA + 0x488	R/W	PDMA Request Source Select Register 2	0x0000_0000
PDMAx_STCR0	PDMAx_BA + 0x500	R/W	Stride Transfer Count Register of PDMA Channel 0	0x0000_0000
PDMAx_ASOCR0	PDMAx_BA + 0x504	R/W	Address Stride Offset Register of PDMA Channel 0	0x0000_0000
PDMAx_STCR1	PDMAx_BA + 0x508	R/W	Stride Transfer Count Register of PDMA Channel 1	0x0000_0000
PDMAx_ASOCR1	PDMAx_BA + 0x50C	R/W	Address Stride Offset Register of PDMA Channel 1	0x0000_0000
PDMAx_STCR2	PDMAx_BA + 0x510	R/W	Stride Transfer Count Register of PDMA Channel 2	0x0000_0000
PDMAx_ASOCR2	PDMAx_BA + 0x514	R/W	Address Stride Offset Register of PDMA Channel 2	0x0000_0000
PDMAx_STCR3	PDMAx_BA + 0x518	R/W	Stride Transfer Count Register of PDMA Channel 3	0x0000_0000
PDMAx_ASOCR3	PDMAx_BA + 0x51C	R/W	Address Stride Offset Register of PDMA Channel 3	0x0000_0000
PDMAx_STCR4	PDMAx_BA + 0x520	R/W	Stride Transfer Count Register of PDMA Channel 4	0x0000_0000
PDMAx_ASOCR4	PDMAx_BA + 0x524	R/W	Address Stride Offset Register of PDMA Channel 4	0x0000_0000
PDMAx_STCR5	PDMAx_BA + 0x528	R/W	Stride Transfer Count Register of PDMA Channel 5	0x0000_0000
PDMAx_ASOCR5	PDMAx_BA + 0x52C	R/W	Address Stride Offset Register of PDMA Channel 5	0x0000_0000

8 TIMER CONTROLLER (TMR)

8.1 Overview

The timer controller includes four 32-bit timers, Timer0 ~ Timer5, allowing user to easily implement a timer control applications. The timer can perform functions, such as frequency measurement, delay timing, clock generation, and event counting by external input pins, and interval measurement by external capture pins.

8.2 Features

- Independent Clock Enable Control for each Timer (TMRx, x= 0~5)
- Time-out period = (Period of TMRx clock input) * (8-bit pre-scale counter + 1) * (24-bit CMP)
- Counting cycle time = (1 / TMRx_CLK) * (2^8) * (2^{24})
- Internal 8-bit pre-scale counter
- Internal 24-bit up counter is readable through CNT (TMRx_CNT[23:0])
- Provides one-shot, periodic, toggle-output and continuous counting operation modes
- Supports external pin capture for interval measurement
- Supports external pin capture for timer counter reset
- 24-bit capture value is readable through CAPDAT (TMRx _CAP[23:0])
- Supports event counting function to count input event from pin TMx_CNT (x = 0~5)
- Supports chip wake-up from Idle/Power-down mode if a timer interrupt signal is generated
- Supports time-out interrupt or capture interrupt to trigger PDMA.
- Supports Inter-Timer trigger that Timer 0 can trigger Timer 1, Timer 2 can trigger Timer 3 and Timer 4 can trigger Timer 5

8.3 Block Diagram

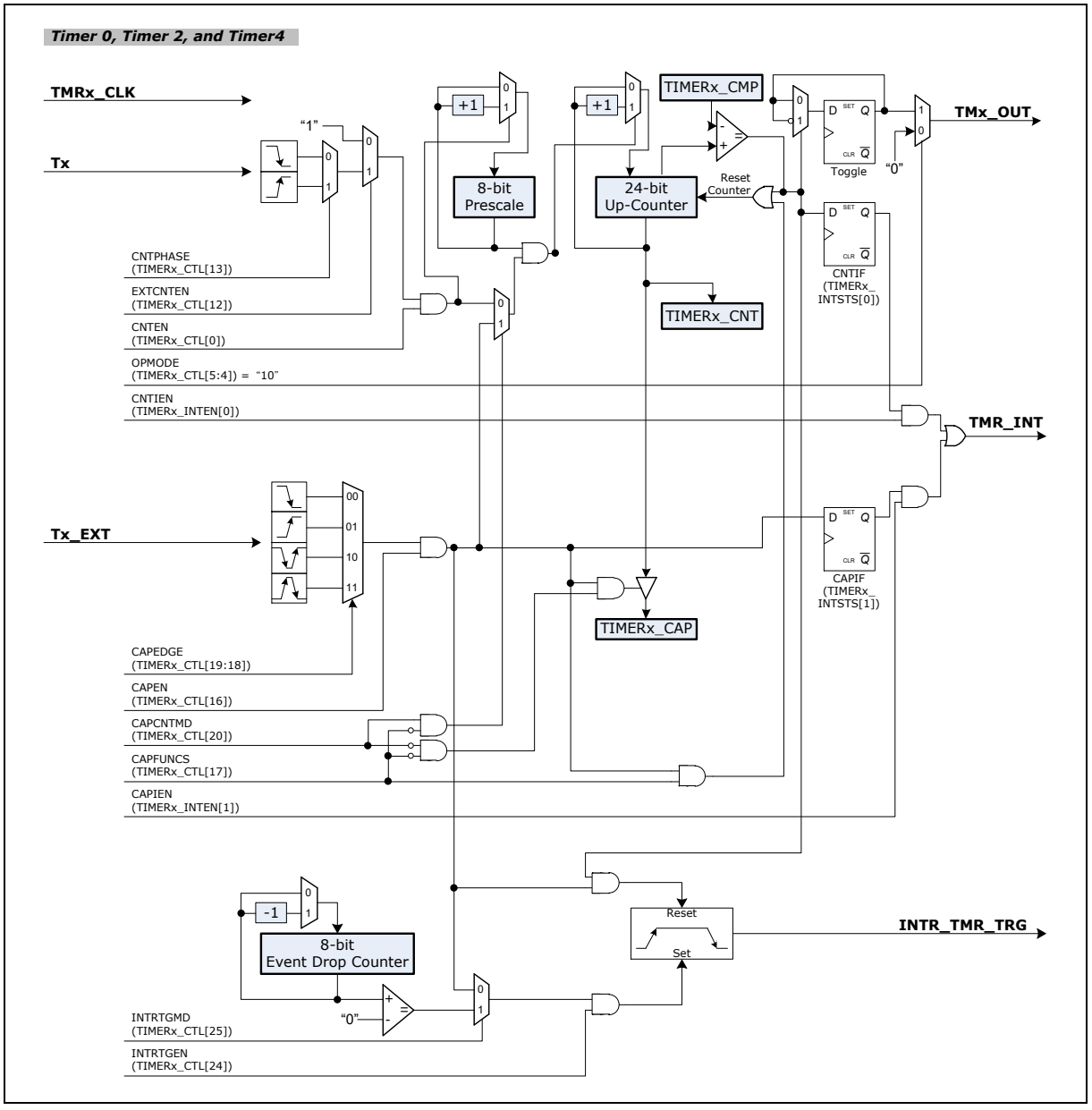


Figure 8.3-1 Timer Controller Block Diagram

8.4 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.

8.4.1 Timer Initialization

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

1. Stop timer counting by clear CNTEN (TMRx_CTL[0]) to 0.
2. Configure OPMODE (TMRx_CTL [5: 4]) to set operating mode.
3. Set CNTIEN (TMRx_INTEN [0]) to 1 for enable interrupt, otherwise clear to 0.
4. Set prescaler in PSC (TMRx_PRECNT [7: 0]).
5. Set timer compare value in CMPDAT (TMRx_CMP [24:0]).
6. Set CNTEN (TMRx_CTL[0]) 1 to enable timer counting.

8.4.2 Timer Capture Initialization

Below list the procedure to initialize timer capture mode:

1. Stop timer by clear both CNTEN (TMRx_CTL[0]) and CAPEN (TMRx_CTL [16]) to 0.
2. Configure OPMODE (TMRx_CTL [5: 4]) to set operating mode.
3. Set CAPIEN (TMRx_INTEN [1]) 1 to enable capture interrupt, otherwise clear to 0.
4. Set CAPCNTMD (TMRx_CTL [20]) and CAPFUNCS (TMRx_CTL [17]) to select the capture mode.
5. Configure CAPEDGE (TMRx_CTL [19:18]) to select capture trigger condition.
6. To enable capture debounce, set CAPDBEN (TMRx_CTL [22]) to 1, Otherwise clear to 0.
7. Set prescaler in PSC (TMRx_PRECNT [7: 0]).
8. Set timer compare value in CMPDAT (TMRx_CMP [24:0])
9. Set CAPEN (TMRx_CTL [16]) 1 to enable capture mode.
10. Set CNTEN (TMRx_CTL[0]) 1 to enable counter.

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

8.4.3 INTERRUPT HANDLING

Every timer has individual interrupt source and interrupt could be ether timeout interrupt or capture interrupt. While timeout interrupt triggers, CNTIF (TMRx_INTSTS[0]) will be set 1. While capture interrupt triggers, CAPIF (TMRx_INTSTS[1]) will be set 1. These two bits could be cleared by write 1 to them.

If new capture event occurs before CAPIF cleared in capture mode, CAPDATOF (TMRx_INTSTS[5]) will be set 1. This bit will be cleared after when CAPIF cleared.

8.4.4 TIEMR FREQUENCY

Formular below can be used to calculate timer timeout frequency:

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

Where TMRx_CLK is timer clock source frequency. Could be HXT (12 MHz external frequency), PCLK, PCLK/4096, or LXT (32.768 kHz external crystal). PRESCALE prescaler defined in PRECNT (TMRx_PRECNT[7:0]), TCMP timer compare value defined in CMPDAT (TMRx_CMP[24:0]). Following table shows some example of timer setting to generate 1Hz, 10Hz, 100Hz, and 1000Hz frequency.

Timer Frequency	Timer Clock Source	PRECNT (TMRx_PRECNT[7:0])	TMR_CMP (TMRx_CMP[24:0])
1Hz	LXT	0	0x8000
10Hz	HXT	0	0x124F80
10Hz	HXT	9	0x1D4C0
100Hz	HXT	9	0x2EE0
100Hz	HXT	19	0x1770
1000Hz	PCLK (75 MHz)	4	0x3A98
1000Hz	HXT	9	0x4B0

Table 8.4-1 Timer Frequency Setting Example

8.4.5 ONE-SHOT MODE

If the timer is operated in One-shot mode (TMRx_CTL[5:4] is 00) and CNTEN (TMRx_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, the CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1. If CNTIEN (TMRx_INTEN[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 CNTIEN (TMRx_INTEN[0] timer interrupt enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1, timer counting operation stops and the timer counter value (TMRx_CNT value) goes back to counting initial value then CNTEN (TMRx_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 TMRx_CMP value function only one time after programming the timer compare register (TMRx_CMP) value and CNTEN (TMRx_CTL[0] timer counter enable bit) is set to 1. So, this operating mode is called One-Shot mode.

8.4.6 PERIODIC MODE

If the timer is operated in Periodic mode (TMRx_CTL[5:4] is 01) and TMR_EN (TMRx_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, the CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1. If CNTIEN (TMRx_INTEN[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 CNTIEN (TMRx_INTEN[0] timer interrupt enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1, the timer counter value (TMRx_CNT value) goes back to counting initial value and TMR_EN (TMRx_CTL[0] timer counter enable bit) is kept at 1 (counting enable continuously) and timer counter operates up counting again. If CNTIF (TMRx_INTSTS[0] timer interrupt status) is cleared by software, once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value again, CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1 also. That is to say, timer operates timer counting and compares with TMRx_CMP value function periodically. The timer counting operation does not stop until the TMR_EN (TMRx_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.

8.4.7 TOGGLE MODE

If the timer is operated in Toggle mode (TMRx_CTL[5:4] is 10) and TMR_EN (TMRx_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, the CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1. If CNTIEN (TMRx_INTEN[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 CNTIEN (TMRx_INTEN[0] timer interrupt enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, CNTIF (TMRx_INTSTS[0] timer interrupt status) and toggle out signal will set to 1, the timer counter value (TMRx_CNT value) goes back to counting initial value and TMR_EN (TMRx_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 (TMRx_CNT value) reaches timer compare register value again, toggle out signal is set to 0, and CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1 also. The timer counting operation does not stop until the TMR_EN (TMRx_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.

8.4.8 CONTINUOUS MODE

If the timer is operated in Continuous Counting mode (MODE_SEL[1:0] is 11) and TMR_EN (TMRx_CTL[0] timer counter enable bit) is set to 1, the timer counter starts up counting. Once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, the CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1. If CNTIEN (TMRx_INTEN[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 CNTIEN (TMRx_INTEN[0] timer counter enable bit) is set to 0, no interrupt signal is generated.

In this operating mode, once the timer counter value (TMRx_CNT value) reaches timer compare register (TMRx_CMP) value, CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1 and TMR_EN (TMRx_CTL[0] timer counter enable bit) is kept at 1 (counting enable continuously) and timer counter continuous counting without reload the timer counter value (TMRx_CNT value) to counting initial value. User can change different timer compare register (TMRx_CMP) value immediately without disabling timer counter and restarting timer counter counting.

For example, the timer compare register (TMRx_CMP) value is set as 80, first. (The timer compare register (TMRx_CMP) should be less than 224 and be greater than 1). Once the timer counter value (TMRx_CNT value) reaches to 80, CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1 and TMR_EN (TMRx_CTL[0] timer counter enable bit) is still kept at 1 (counting enable continuously). Next, user clears the CNTIF (TMRx_INTSTS[0] timer interrupt status) and reprograms timer compare register (TMRx_CMP) value as 200, then CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1 again when timer counter value (TMRx_CNT value) reaches to 200. At last, user clears CNTIF

(TMRx_INTSTS[0] timer interrupt status) and reprograms timer compare register (TMRx_CMP) value as 500, then CNTIF (TMRx_INTSTS[0] timer interrupt status) will set to 1 again when timer counter value (TMRx_CNT value) reaches to 500. In this mode, when the timer counter value (TMRx_CNT value) continues counting up to 224 -1, then recount up from 0 continuously. The timer counter value (TMRx_CNT value) is always keeping up counting even if CNTIF (TMRx_INTSTS[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.

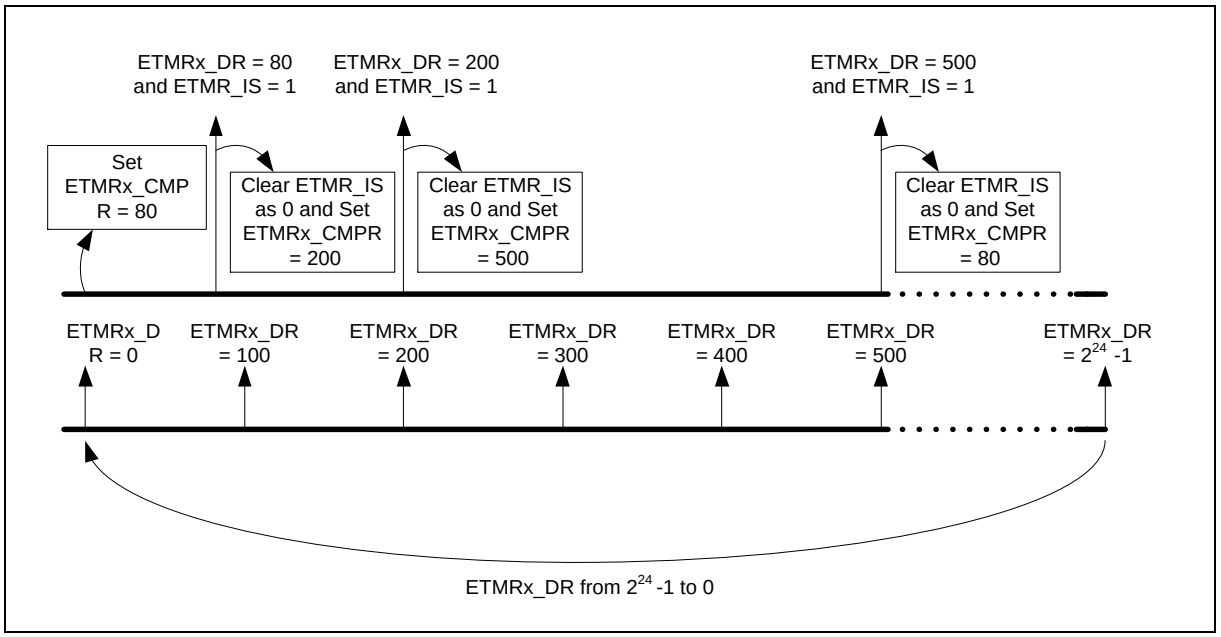


Figure 8.4-1 Timer Continuous Mode Operation

8.4.9 Event Counting Mode

The timer controller also provides an application which can count the input event from TMx_CNT

(x= 0~5) pin and the number of event will reflect to CNT (TMRx _CNT[23:0]) value. It is also alled as event counting function. In this function, EXTCNTEN (TMRx_CTL[12]) should be set and the timer peripheral clock source should be set as HCLK User can enable or disable TMx_CNT pin de-bounce circuit by setting CNTDBEN(TMRx_CTL[14]). The input event frequency should be less than 1/6HCLK if TMx_CNT pin debounce disabled, or less than 1/16HCLK if TMx_CNT pin de-bounce enabled to assure the returned CNT (TMRx_CNT[23:0]) value is correct, and user can also select edge detection phase of TMx_CNT pin by setting CNTPHASE (TMRx_CTL[13]) bit. In event counting mode, the timer counting operation mode can be selected as One-shot, Periodic, Toggle-output and Continuous Counting mode to counts the counter value CNT (TMRx_CNT[23:0]) for TMx_CNT pin.

8.4.10 FREE COUNTING MODE

In this mode, timer monitors the capture pin toggle event to save current counter value. If both CAPFUNCS (TMRx_CTL[17]) and CAPCNTMD (TMRX_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 CAPEDGE (TMRX_CTL[19:18]), current 24 up counter value will be stored in TMRx_CAP register. At the mean time, if CAPIEN (TMRx_INTEN[1]) is 1, CAPIF(TMRX_INTSTS [1]) will set 1 and trigger interrupt.

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

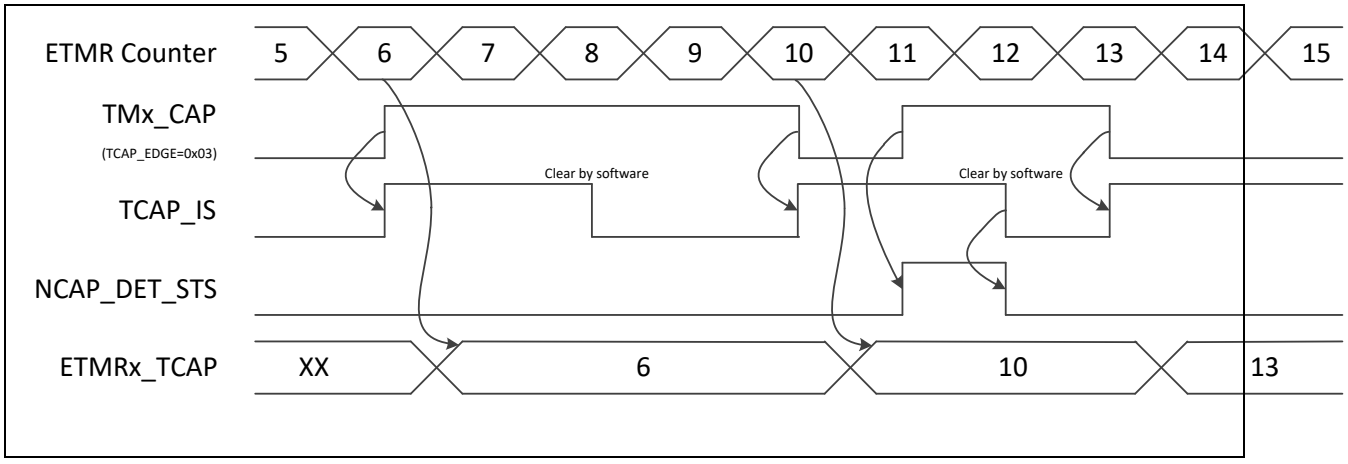


Figure 8.4-2 Timer Free Counting Mode

8.4.11 TRIGGER COUNTING MODE

In this mode, timer monitors the capture pin toggle event to start/stop timer counter and save captured value. If CAPFUNCS (TMRx_CTL[17]) is 0 and CAPCNTMD (TMRX_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 CAPEDGE (TMRx_CTL[19:18]) first trigger condition, 24 counter starts counting. And timer counter stops counting and store current counter value to TMRx_CAP register. When the external capture pin toggle state matches the second trigger condition set in CAPEDGE. If CAPIEN (TMRx_INTEN[1]) is 1, CAPIF(TMRx_INTSTS [1]) will be set 1 and triggers interrupt.

In trigger counting mode, if CAPEDGE is 0, first falling edge on TMx_CAP starts timer up counting, second falling edge stops timer counter. If CAPEDGE is 1, first rising edge on TMx_CAP starts timer up counting, second rising edge stops timer counter. If CAPEDGE is 2, falling edge on TMx_CAP starts timer counter, and rising edge stops counter. If CAPEDGE is 3, rising edge on TMx_CAP starts timer counter, and falling edge stops counter.

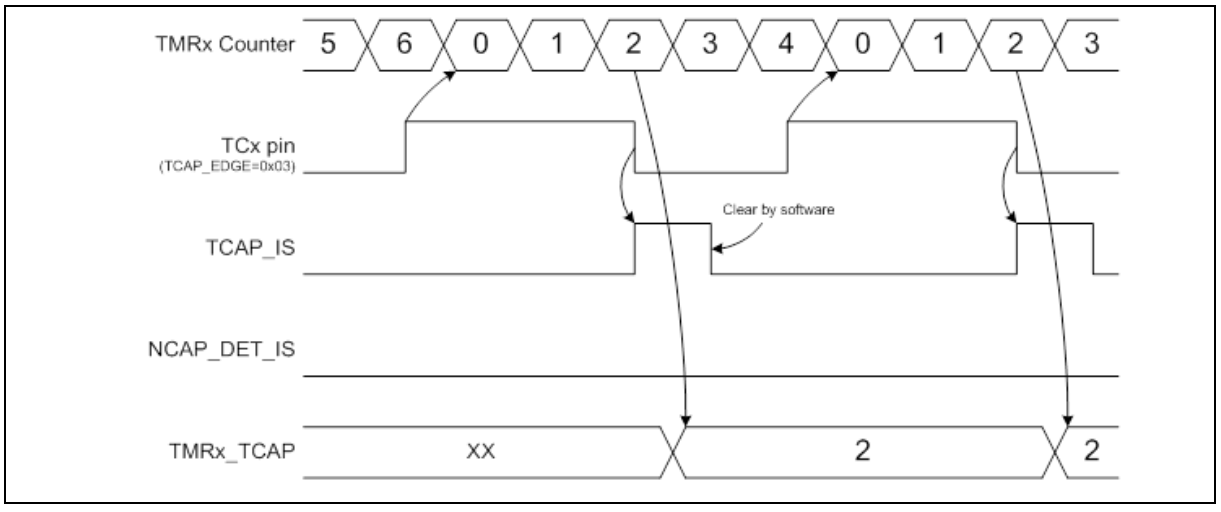


Figure 8.4-3 Timer Trigger Counting Mode

8.4.12 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 CAPFUNCS (TMRx_CTL[17]) is 1. In

this mode, while external capture pin toggle status matches the setting in CAPEDGE (TMRx_CTL[19:18]), timer counter will be reset an keep up counting. If CAPIEN (TMRx_INTEN[1]) is 1, CAPIF(TMRx_INTSTS [1]) will be set 1 and trigger interrupt.

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

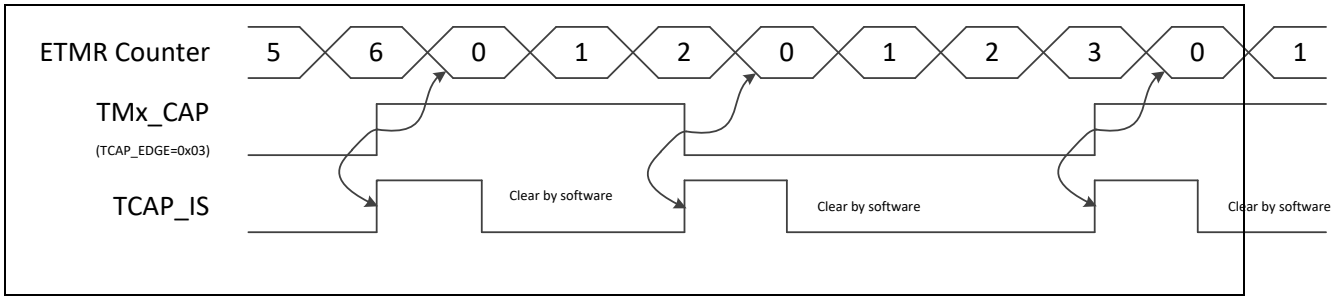


Figure 8.4-4 Timer Counter Reset Mode

8.4.13 CAPTURE DEBOUNCE

Timer capture supports debounce function for detecting capture pin toggle. Timer can use ether original signal or debounced signal to detect capture pin status. Default state of debounce circuit is disabled. And only will be enabled if both CAPDBEN (TMRx_CTL[22]) and CAPEN (TMRx_CTL[16]) are set 1. So if capture pin level is 1, and CAPEDGE (TMRx_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 TMRx_CAP 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.

8.4.14 Inter-Timer Trigger Mode

The timer controller provides inter-timer trigger function to measure input frequency precisely. In inter-timer trigger function, TMR0 can trigger TMR1, TMR2 can trigger TMR3, and TMR4 can trigger TMR5.

If TMR0 and TMR1 are configured in Inter-timer trigger mode (INTRTGEN (TMR0_CTL[24]) is 1), TMR0 is operating at event counting mode to count the input event from T0 pin and generate an internal signal (INTR_TMR_TRG) to Timer1. Timer0 transit internal signal INTR_TMR_TRG from low to high if 1st input event detected on T0 pin and then transit internal signal INTR_TMR_TRG from high to low if CNT (TMR0_CNT[23:0]) value reaches CMPDAT (TMRx_CMP[23:0]) value. Timer1 is operating at external capture trigger-counting mode to starts the timer counter up counting if rising edge transition on INTR_TMR_TRG detected and load CNT (TMR1_CNT[23:0]) value to CAPDAT (TMR1_CAP[23:0]) if falling edge transition on INTR_TMR_TRG detected.

If INTRTGEN (TMR2_CTL[24]) is set to 1, TMR2 and TMR3 are configured in Inter-timer Trigger mode. The operation behavior of TMR2 and TMR3 in inter-timer trigger function is the same as the operation behavior of TMR0 and TMR1. The inter-timer trigger settings for TMR4 and TMR5 are the same.

The following figure describes how inter-timer trigger function operated with TMR0 and TMR1.In the end of inter-timer trigger function, the CNTIF (TMR0_INTSTS[0]) will set to 1 and INTRTGEN (TMR0_CTL[24]) is cleared automatically. In the meantime, if the CNTIEN (TMR0_INTEN[0]) bit is set to 1, the timer interrupt signal is generated and sent to NVIC to inform CPU as well.

By using Inter-timer trigger function, the frequency of input event from T0 pin could be measured more precisely.In following figure when TMR0 counts 100 input events, the counter of TMR1 counts to 999. If TMR1's clock frequency is 10 MHz, then we know the time for 100 events is 99900ns. Therefore, the period of an input event is 999ns and the frequency of input event will be 1.001 MHz.

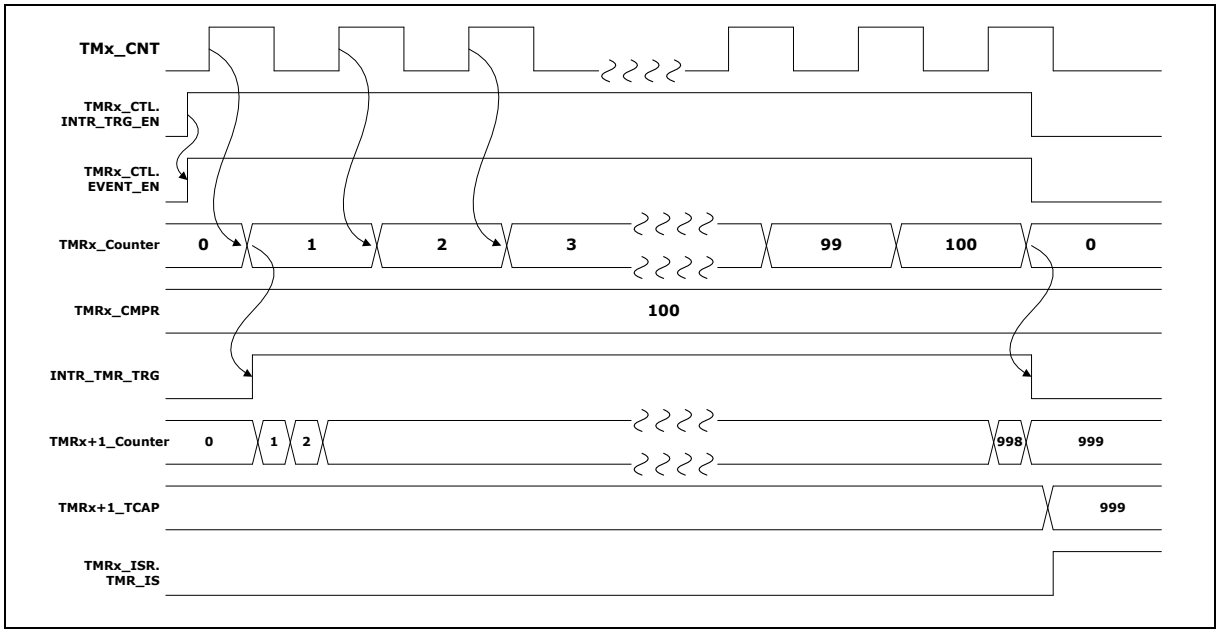


Figure 8.4-5 Inter-Timer Trigger Mode

8.5 Register Map

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

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

9 PULSE WIDTH MODULATION (PWM)

9.1 Overview

This chip has two PWM controllers, and each one 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.

9.2 Features

- Two PWM controllers, each one has 4 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.

9.3 Block Diagram

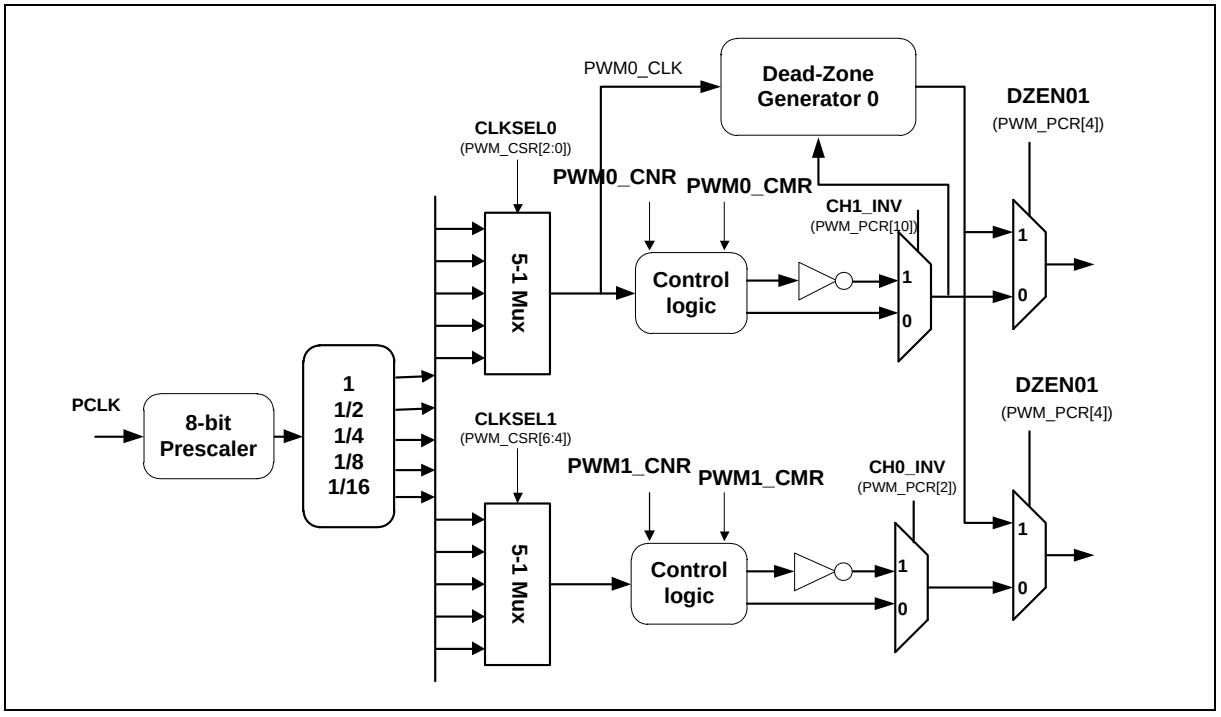


Figure 9.3-1 PWM Block Diagram

9.4 Functional Description

9.4.1 PWM Timer Operation

The PWM period and duty control are decided by register PWMx_CNR and PWMx_CMR registers. The PWM-timer timing operation is shown in following figure. The pulse width

modulation follows the formula below:

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

PWM duty ratio = (CMR + 1) / (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..

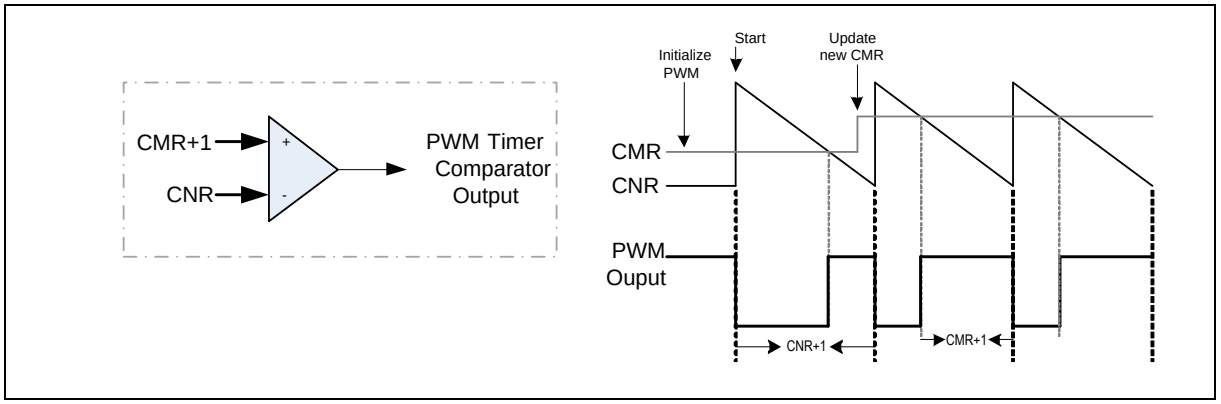


Figure 9.4-1 PWM Internal Comperator Output

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

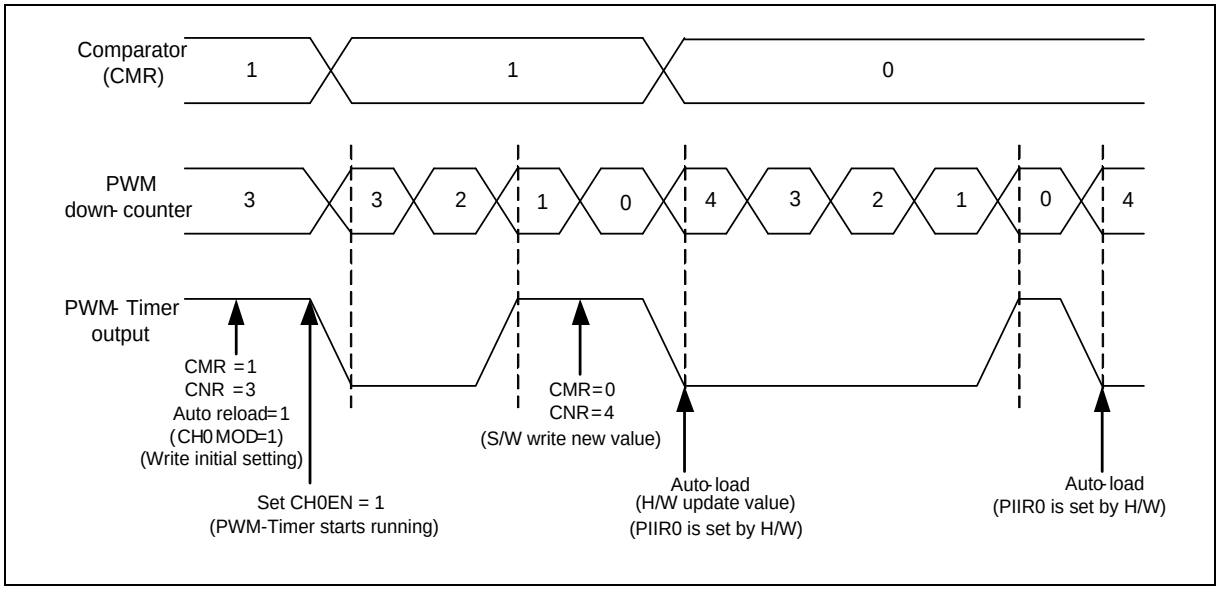


Figure 9.4-2 PWM Counter Reload

9.4.2 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]).

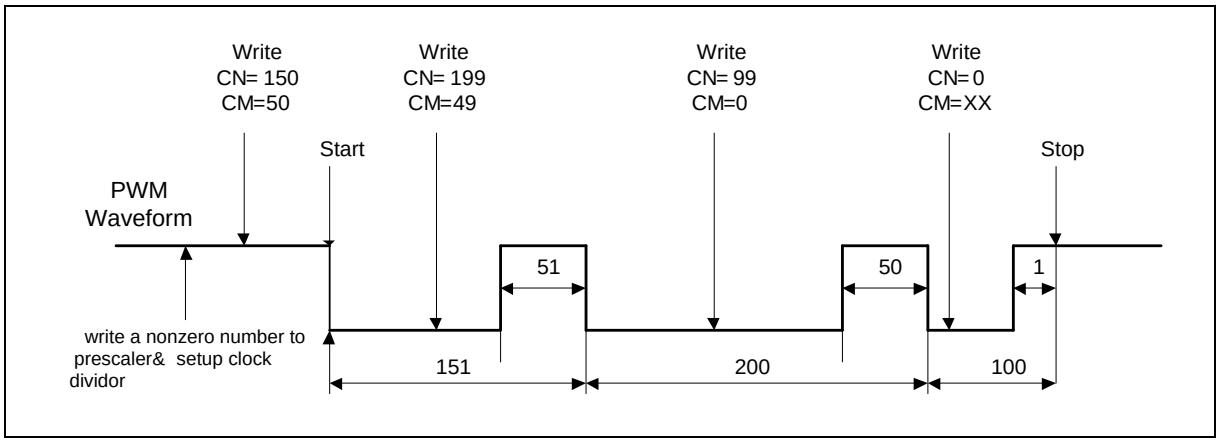


Figure 9.4-3 PWM Double Buffer Illustration (I)

Following figure is an example of using double buffer feature.

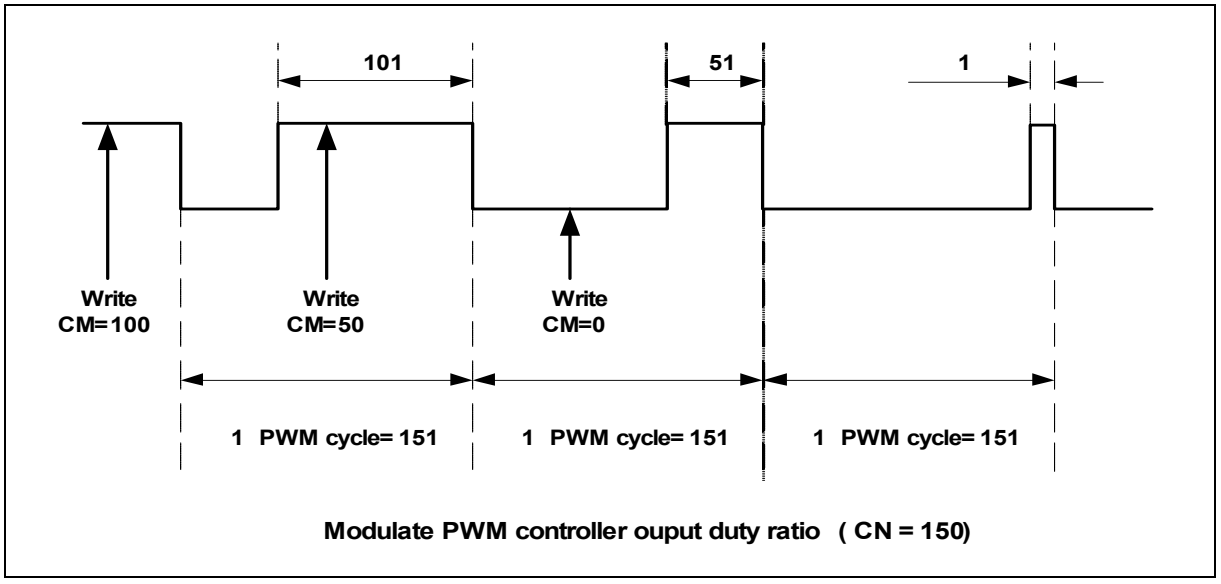


Figure 9.4-4 PWM Double Buffer Illustration (II)

9.4.3 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

9.4.4 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.

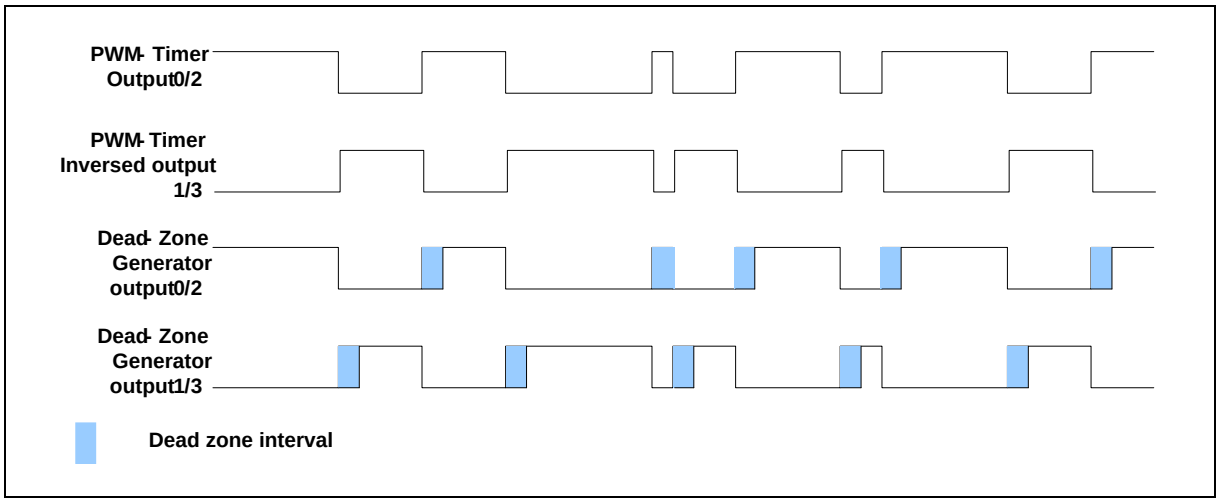


Figure 9.4-5 PWM Output with Dead Zone Operation

9.4.5 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 GPI/O pins to PWM function
8. Enable PWM down-counter by set CH0EN((PWM_PCR[0])) to 1.
9. Setup PWM comparator register CMR (PWM_CMR[15:0]) and PWM counter register CNR
10. (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 75 MHz / (74 + 1) = 1 MHz
PWM->CSR = 4;         // Prescale output divide by 1
PWM->PCR = 9;         // Enable PWM0 in periodic mode

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

9.4.6 PWM 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 an abruptly change in PWM duty ration. This may damage the motor connected with PWM.

9.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
PWM Base Address: **PWM0_BA = 0xB005_8000** **PWM1_BA = 0xB005_9000**				
PWM_PPR	PWM_BA+0x000	R/W	PWM Pre-scale Register	0000_0000
PWM_CSR	PWM_BA+0x004	R/W	PWM Clock Select Register	0000_0000
PWM_PCR	PWM_BA+0x008	R/W	PWM Control Register	0000_0000
PWM0_CNR	PWM_BA+0x00C	R/W	PWM Counter Register 0	0000_0000
PWM0_CMR	PWM_BA+0x010	R/W	PWM Comparator Register 0	0000_0000
PWM0_PDR	PWM_BA+0x014	R	PWM Data Register 0	0000_0000
PWM1_CNR	PWM_BA+0x018	R/W	PWM Counter Register 1	0000_0000
PWM1_CMR	PWM_BA+0x01C	R/W	PWM Comparator Register 1	0000_0000
PWM1_PDR	PWM_BA+0x020	R	PWM Data Register 1	0000_0000
PWM2_CNR	PWM_BA+0x024	R/W	PWM Counter Register 2	0000_0000
PWM2_CMR	PWM_BA+0x028	R/W	PWM Comparator Register 2	0000_0000
PWM2_PDR	PWM_BA+0x02C	R	PWM Data Register 2	0000_0000
PWM3_CNR	PWM_BA+0x030	R/W	PWM Counter Register 3	0000_0000
PWM3_CMR	PWM_BA+0x034	R/W	PWM Comparator Register 3	0000_0000
PWM3_PDR	PWM_BA+0x038	R	PWM Data Register 3	0000_0000
PWM_PIER	PWM_BA+0x03C	R/W	PWM Timer Interrupt Enable Register	0000_0000
PWM_PIIR	PWM_BA+0x040	R/W	PWM Timer Interrupt Indication Register	0000_0000

10 WATCHDOG TIMER (WDT)

10.1 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.

10.2 Features

- 18-bit free running up counter for WDT time-out interval.
- Selectable time-out interval (2^4~ 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 / 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.

10.3 Block Diagram

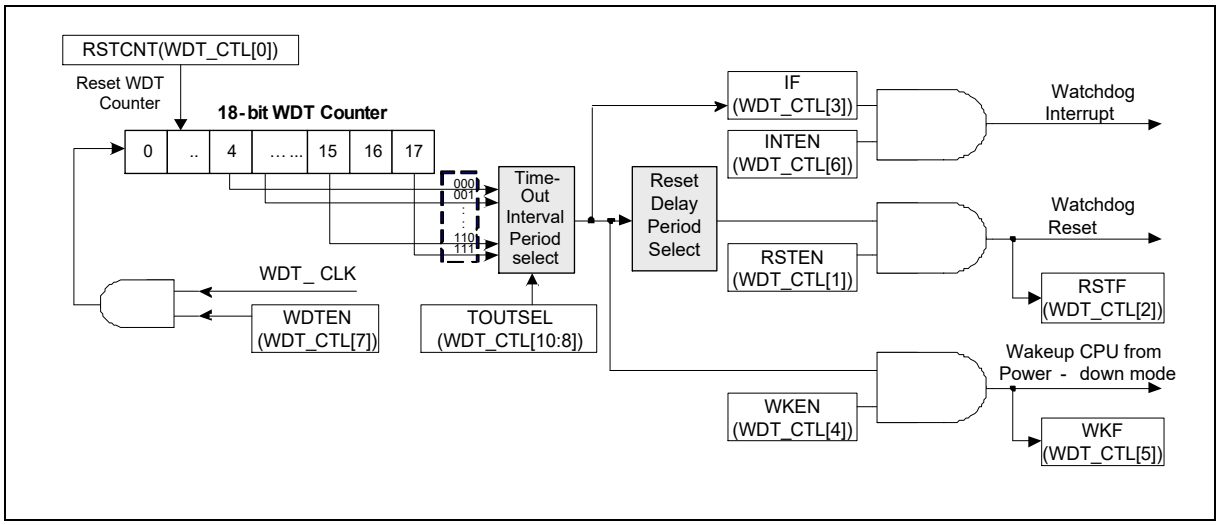


Figure 10.3-1 WDT Block Diagram

10.4 Functional Description

10.4.1 WDT Configuration

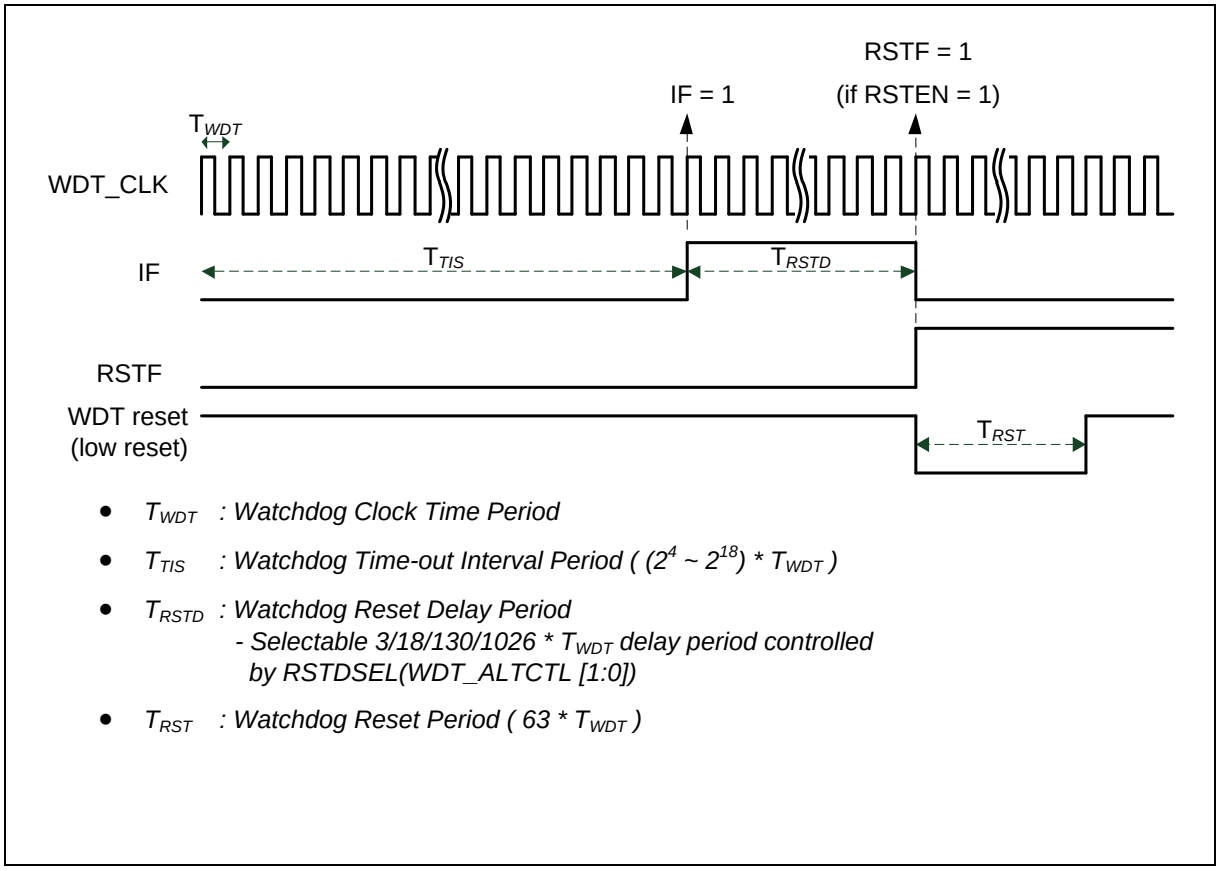
Watchdog timer is used to trigger a system reset while the software execute to an abnormal state, this prevents the system stays in an uncontrollable state for unlimited duration. Besides, WDT support wakeup function that can wake up CPU from power-down state and right before CPU enters power-down state, 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, T_{WDT} in the table depends on the peripheral clock source selection through WDT_S(CLK_DIVCTL8[9:8]), it can be HXT (external high speed 12 MHz crystal), 12 MHz/512, PCLK/4096, or LXT (external low speed 32 kHz 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

Table 10.4-1 WDT Timeout Period

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 or write 0x00005AA5 to WDT_RSTCNT register 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.



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

Figure 10.4-1 WDT Timeout 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.

10.4.2 **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 entries 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.

10.5 Register Map

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

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

11 WINDOW WATCHDOG TIMER (WWDT)

11.1 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.

11.2 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.

11.3 Block Diagram

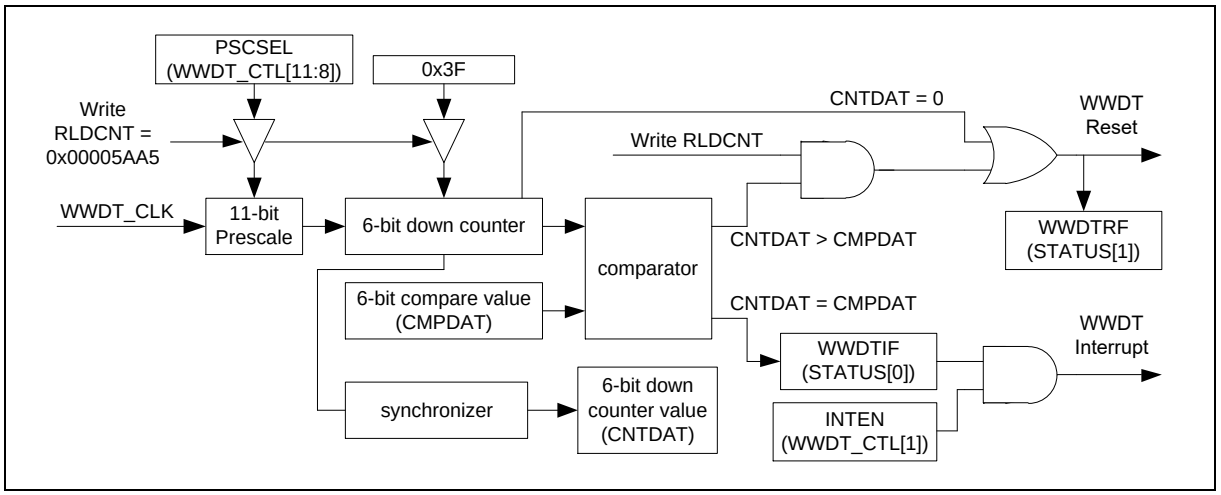


Figure 11.3-1 WWDT Block Diagram

11.4 Functional Description

11.4.1 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 divided by 4096 (PCLK/4096), external 12 MHz oscillator, external 12 MHz oscillator divided by 512, or internal 32 kHz oscillator with a programmable 11-bitprescale 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 uS	1.95 mS
0001	2	2 * 64 * T_{WWDT}	10.66 uS	3.91 mS
0010	4	4 * 64 * T_{WWDT}	21.33 uS	7.81 mS
0011	8	8 * 64 * T_{WWDT}	42.67 uS	15.63 mS
0100	16	16 * 64 * T_{WWDT}	85.33 uS	31.25 mS
0101	32	32 * 64 * T_{WWDT}	170.67 uS	62.50 mS
0110	64	64 * 64 * T_{WWDT}	341.33 uS	125.00 mS
0111	128	128 * 64 * T_{WWDT}	682.67 uS	250.00 mS
1000	192	192 * 64 * T_{WWDT}	1.02 mS	375.00 mS
1001	256	256 * 64 * T_{WWDT}	1.37 mS	500.00 mS
1010	384	384 * 64 * T_{WWDT}	2.05 mS	750.00 mS
1011	512	512 * 64 * T_{WWDT}	2.73 mS	1.00 S
1100	768	768 * 64 * T_{WWDT}	4.10 mS	1.50 S
1101	1024	1024 * 64 * T_{WWDT}	5.46 mS	2.00 S
1110	1536	1536 * 64 * T_{WWDT}	8.19 mS	3.00 S
1111	2048	2048 * 64 * T_{WWDT}	10.09 mS	4.00 S

Table 11.4-1WWDT Timeout Period

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 whiel 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.

11.4.2 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..

11.4.3 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.

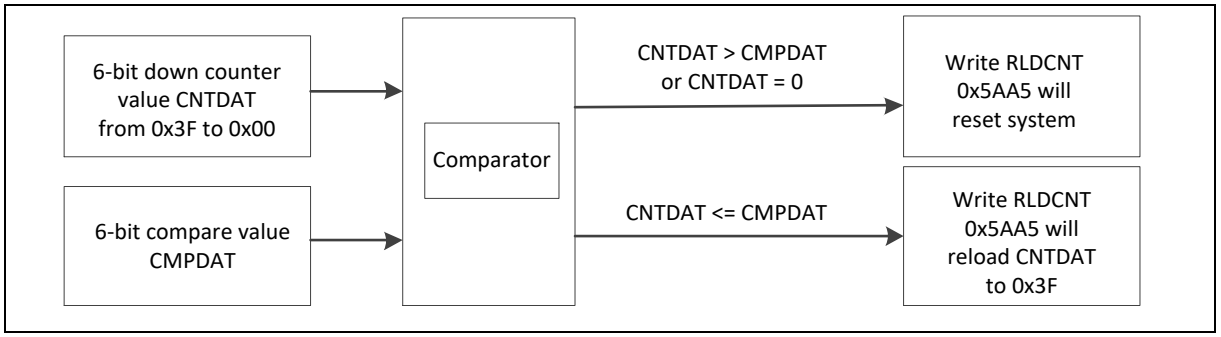


Figure 11.4-1 WWDT Reset and Reload Behavior

11.4.4 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

Table 11.4-2 WWDT CMPDAT Setting Limitation

And also, after system enter power-down mode, WWDT stop counting. So it is not possible to wake up system using WWDT.

11.5 Register Map

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

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

12 REAL TIME CLOCK (RTC)

12.1 Overview

Real Time Clock (RTC) block can operate with independent power supply (RTC_V_{DD}) 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

12.2 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

12.3 Block Diagram

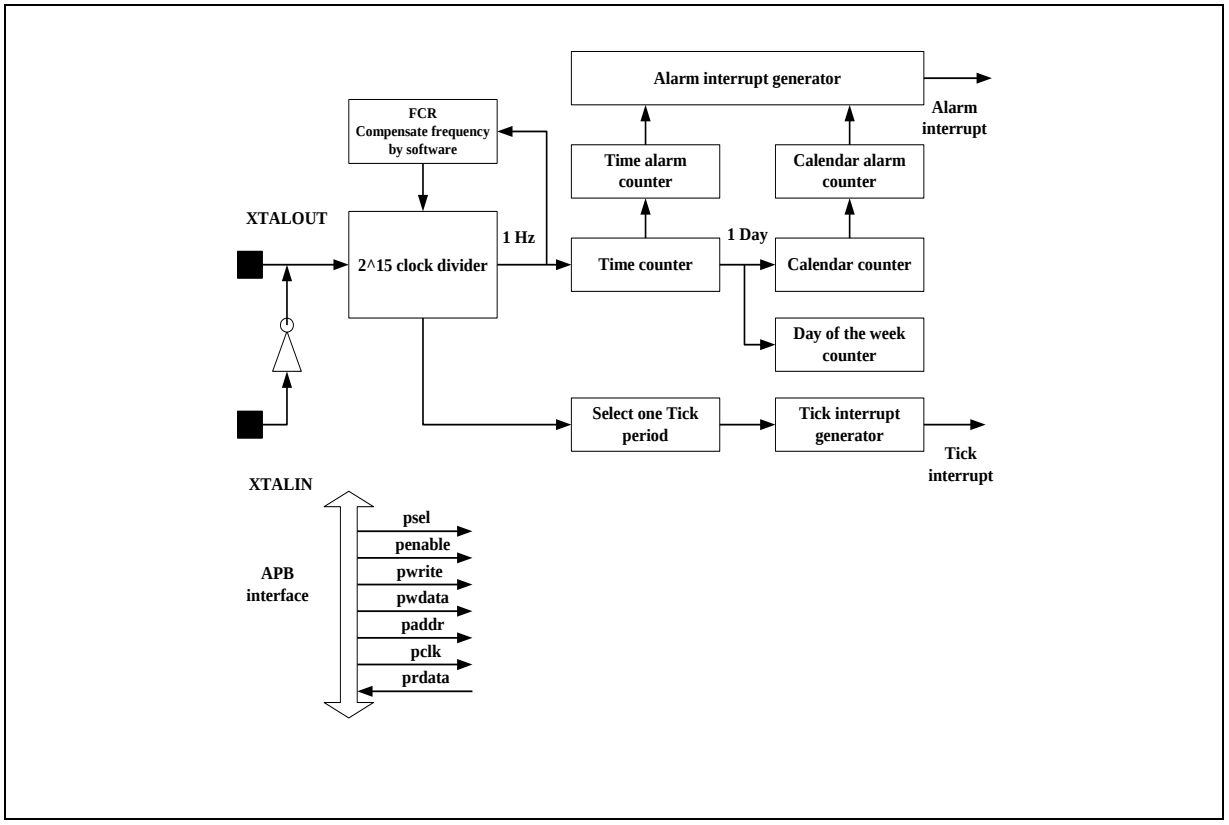


Figure 12.3-1 RTC Block Diagram

12.4 Functional Description

12.4.1 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.

12.4.2 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

Table 12.4-1 RTC Reset Value

12.4.3 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)

Table 12.4-2 RTC 12/24 Hour Mode

12.4.4 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

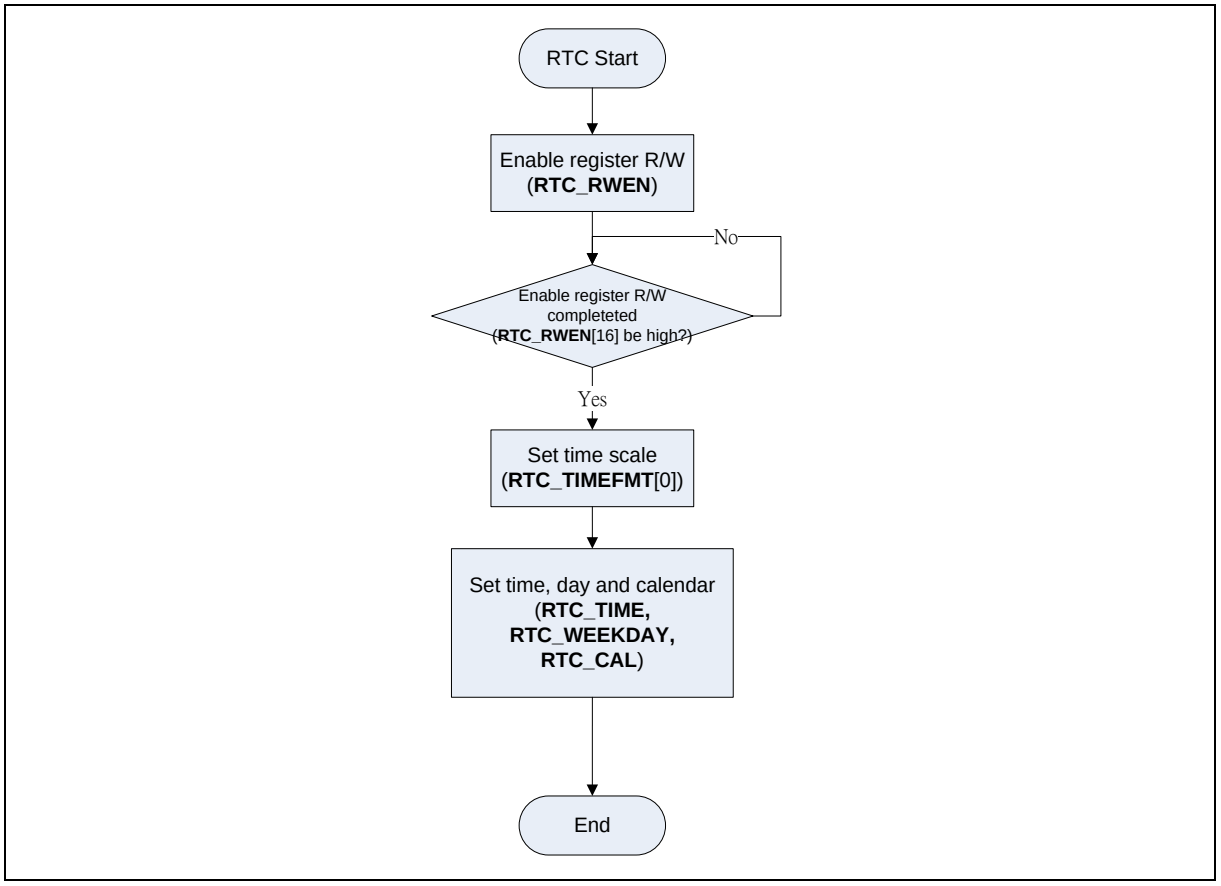


Figure 12.4-1 Set RTC Calander and Time Flow

12.4.5 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

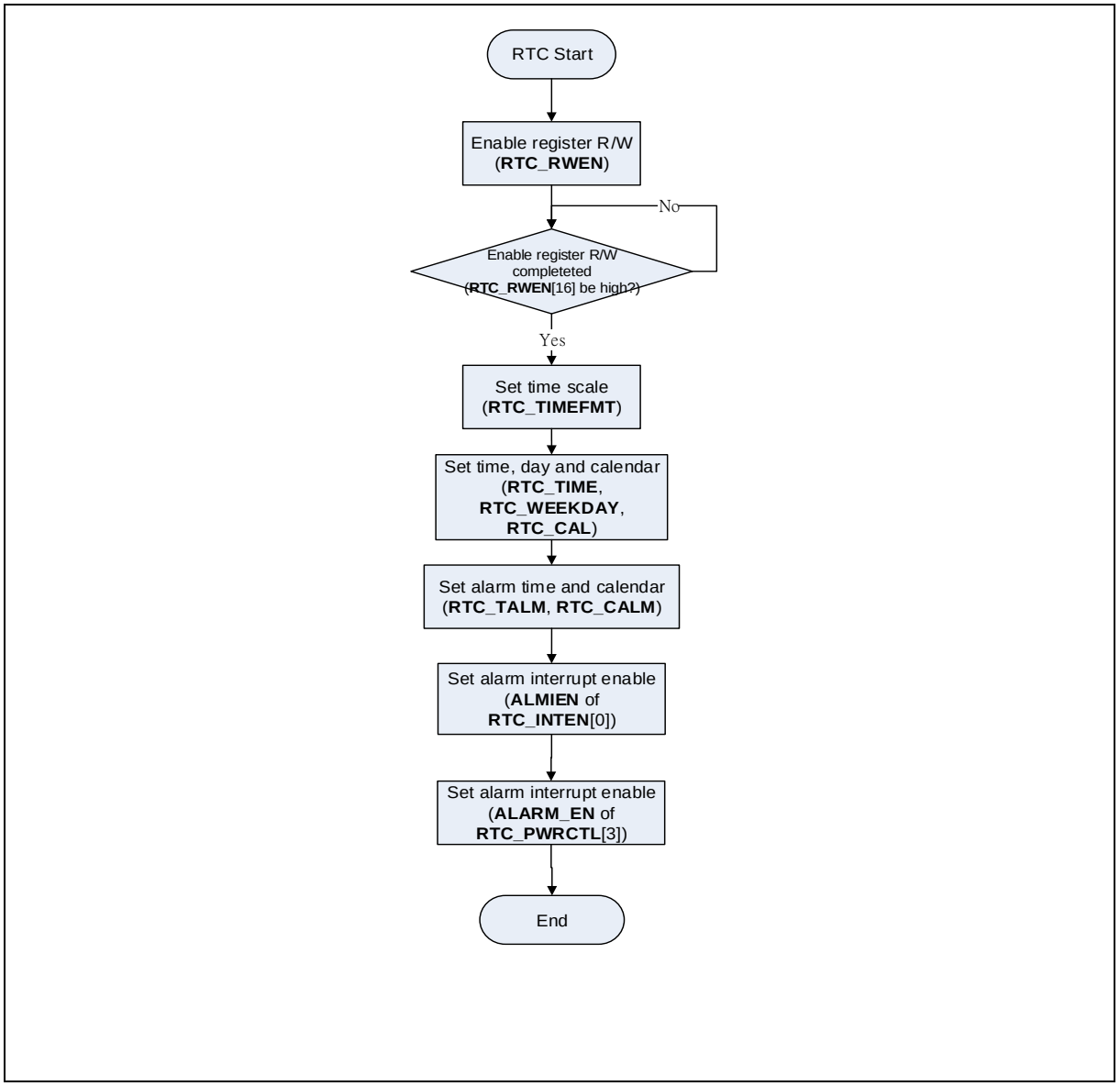


Figure 12.4-2 Set RTC Calander and Timer Alarm Flow

12.4.6 **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

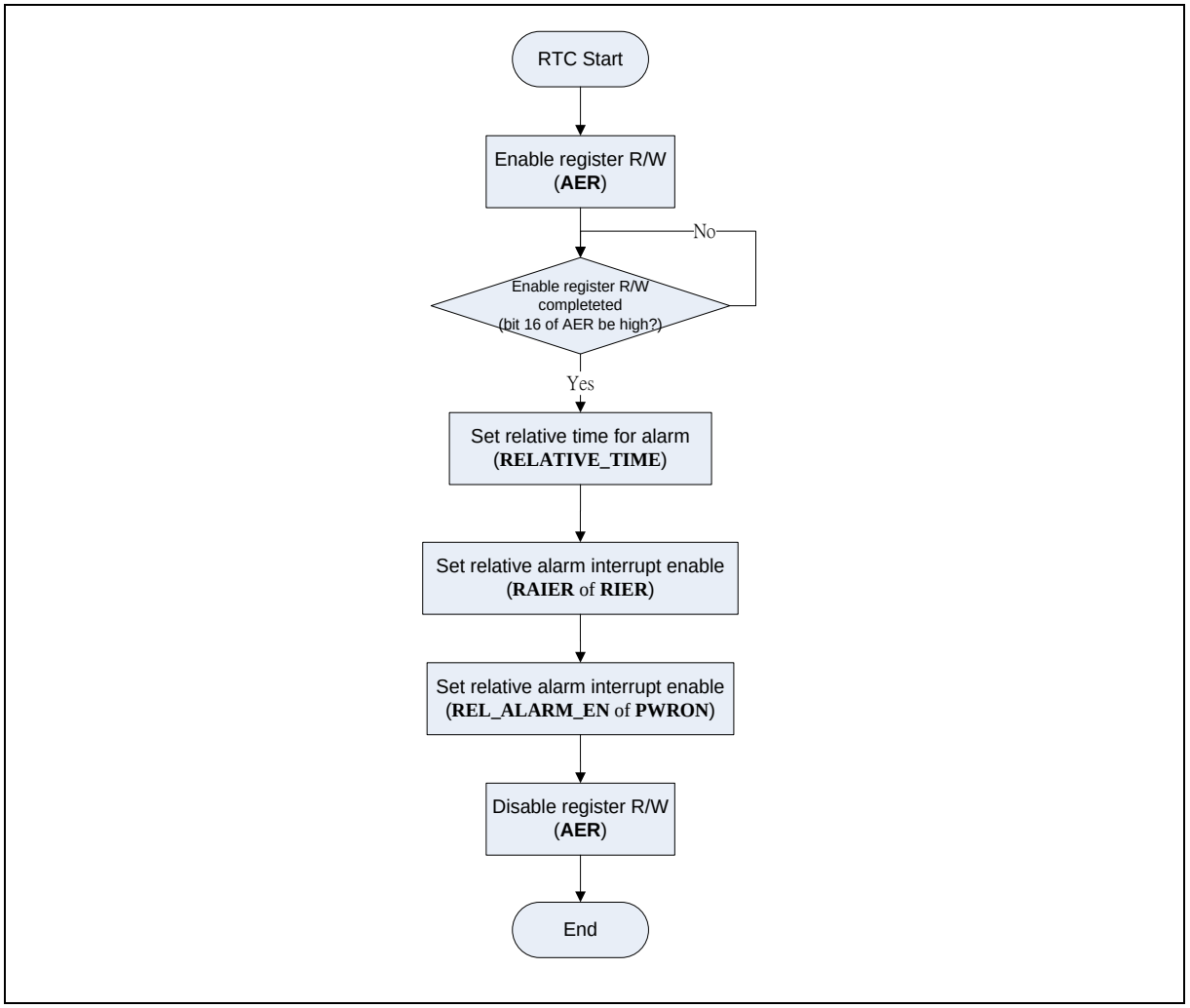


Figure 12.4-3 Set Relative Alarm Flow

12.4.7 **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 do not 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.

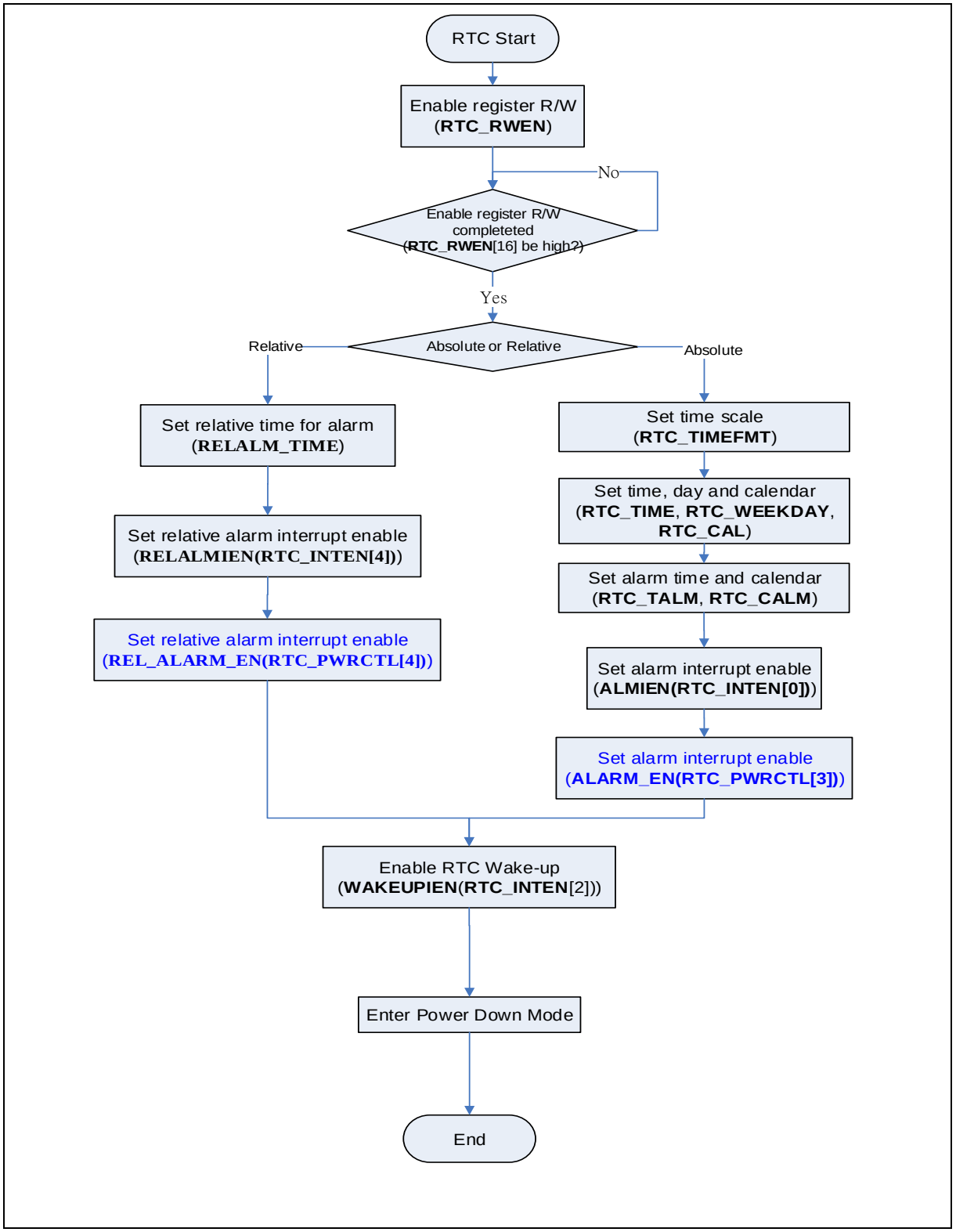


Figure 12.4-4 Set Wake-up Function Flow

12.4.8 **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

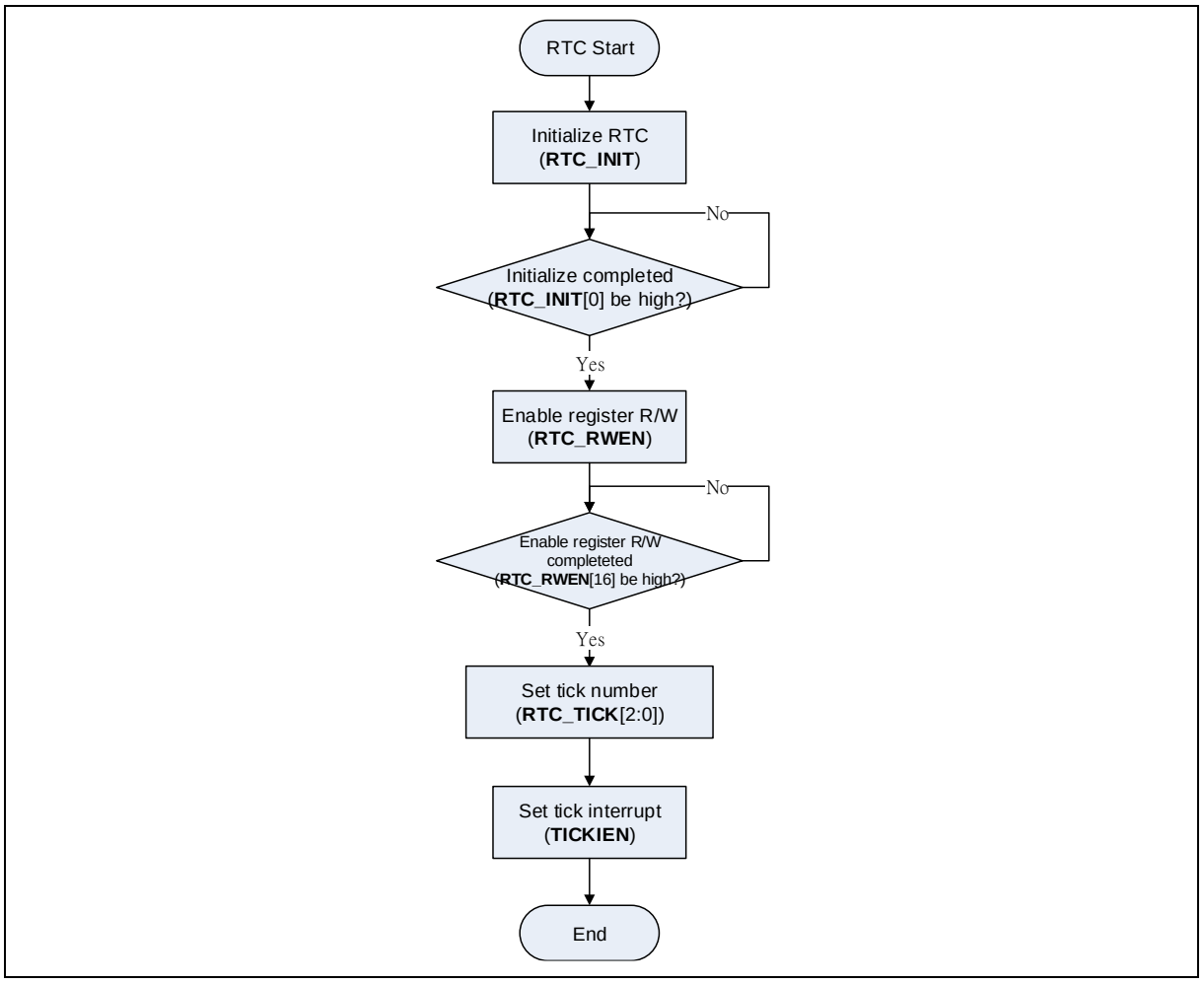


Figure 12.4-5 Set Tick Interrupt Flow

12.4.9 **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 enable SRCSEL(TIMERx_ECTL[16]) bit to measure RTC 1Hz clock during manufacture, and store the value in Flash memory for retrieval when the product is first power on.

Following are the compensation examples:

Frequency counter measurement : 32773.65Hz

Integer part: 32773

Search Following Table:

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

Table 12.4-3 RTC Frequency Compensation

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

12.5 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
RTC_BA = 0xB004_1000				
RTC_INIT	RTC_BA+0x000	R/W	RTC Initiation Register	Undefined
RTC_RWEN	RTC_BA+0x004	R/W	RTC Access Enable Register	0x0000_0000
RTC_FREQADJ	RTC_BA+0x008	R/W	RTC Frequency Compensation Register	0x0000_0700
RTC_TIME	RTC_BA+0x00C	R/W	RTC Time Counter Register	0x0000_0000
RTC_CAL	RTC_BA+0x010	R/W	RTC Calendar Counter Register	0x0005_0101
RTC_TIMEFMT	RTC_BA+0x014	R/W	RTC Time Format Selection Register	0x0000_0001
RTC_WEEKDAY	RTC_BA+0x018	R/W	RTC Day of the Week Register	0x0000_0006
RTC_TALM	RTC_BA+0x01C	R/W	RTC Time Alarm Register	0x0000_0000
RTC_CALM	RTC_BA+0x020	R/W	RTC Calendar Alarm Register	0x0000_0000
RTC_LEAPYEAR	RTC_BA+0x024	R	RTC Leap year Indicator Register	0x0000_0000
RTC_INTEN	RTC_BA+0x028	R/W	RTC Interrupt Enable Register	0x0000_0000
RTC_INTSTS	RTC_BA+0x02C	R/C	RTC Interrupt Status Register	0x0000_0000
RTC_TICK	RTC_BA+0x030	R/W	RTC Time Tick Register	0x0000_0000
RTC_PWRCTL	RTC_BA+0x034	R/W	RTC Power Control Register	0x0000_7000
RTC_PWRCNT	RTC_BA+0x038	R	RTC Power Control Counter Register	0x0000_0000
RTC_SPR0 ~ RTC_SPR15	RTC_BA+0x040 ~ RTC_BA+0x07C	R/W	RTC Spare Register 0 ~ 15	0x0000_0000

13 UART

13.1 Overview

The chip provides 10 channels of Universal Asynchronous Receiver/Transmitters (UART). The UART controller performs Normal Speed UART and supports flow control function. The UART controller 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 controller channel supports ten types of interrupts. The UART controller also supports IrDA SIR, LIN(Only UART1 /UART2 with LIN function) and RS-485 function modes and auto-baud rate measuring function.

There are ten types of interrupts, Receive Data Available Interrupt (RDAINT), Transmit Holding Register Empty Interrupt (THERINT), Transmitter Empty Interrupt (TXENDINT), Receive Line Status Interrupt (parity error or framing error or break interrupt) (RLSINT), MODEM Status Interrupt (MODEMINT), Receiver Buffer Time-out Interrupt (RXTOINT), Buffer Error Interrupt (BUFERRINT), LIN Bus Interrupt (LININT), Wake-up Interrupt (WKINT) and Auto-Baud Rate Interrupt (ABRINT). Interrupt enable register (UART_INTEN) enable or disable the responding interrupt and interrupt status register (UART_INTSTS) identifying the occurrence of the responding interrupt.

The UART0 ~ UART9 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. The wake-up function includes nCTS pin, incoming data wake-up, Received Data FIFO reached threshold wake-up, RS-485 Address Match (AAD mode) wake-up and Received Data FIFO threshold time-out wake-up function. CTSWKF (UART_WKSTS[0]), DATWKF (UART_WKSTS[1]), RFRTWKF (UART_WKSTS[2]), RS485WKF (UART_WKSTS[3]) or TOUTWKF (UART_WKSTS[4]) cause the wake-up interrupt flag WKIF(UART_INTSTS[6]) is generated. If the WKIEN (UART_INTEN[6]) is enabled, the wake-up interrupt flag WKIF(UART_INTSTS[6]) cause the wake-up interrupt WKINT (UART_INTSTS[14]) is generated.

The UART controller includes a programmable baud rate generator capable of dividing clock input by divisors to produce the serial clock that transmitter and receiver need. Table 13.1-1 list the UART baud rate equations in the various conditions. Table 13.1-2 and Table 13.1-3 list the UART baud rate parameter and register setting example. In IrDA function mode, the baud rate generator must be set in mode 0. More detail register description is shown in UART_BAUD register. There are three setting mode. Mode 0 is set by UART_BAUD[29:28] with 00. Mode 1 is set by UART_BAUD[29:28] with 10. Mode 2 is set by UART_BAUD[29:28] with 11.

The following tables list the UART baud rate equations in the various conditions and UART baud rate parameter settings.

Mode	BAUDM1	BAUDM0	Baud Rate Equation
Mode 0	0	0	UART_CLK / [16 * (BRD+2)].
Mode 1	1	0	UART_CLK / [(EDIVM1+1) * (BRD+2)], EDIVM1 must >= 8.
Mode 2	1	1	UART_CLK / (BRD+2) If UART_CLK <= 3*HCLK, BRD must >= 9. If UART_CLK > 3*HCLK, BRD must >= 3*N – 1. N is the smallest integer larger than or equal to the ratio of UART_CLK /HCLK.

			For example, if 3*HCLK < UART_CLK =< 4*HCLK, BRD must >=11. if 4*HCLK < UART_CLK =< 5*HCLK, BRD must >=14. (If the UART_CLK is selected from LXT, BRD can be greater than or equal to 1)

Table 13.1-1 UART controller Baud Rate Equation Table

UART Peripheral Clock = 12 MHz			
Baud Rate	**Mode 0**	**Mode 1**	**Mode 2**
921600	Not support	Not recommended	BRD=11
460800	Not recommended	BRD=0, EDIVM1 =13	BRD=24
230400	Not recommended	BRD =2, EDIVM1 =13	BRD =50
115200	Not recommended	BRD =6, EDIVM1 =13	BRD =102
57600	BRD =11	BRD =14, EDIVM1 =13	BRD =206
38400	BRD =18	BRD =22, EDIVM1 =13	BRD =311
19200	BRD =37	BRD =123, EDIVM1 =5	BRD =623
9600	BRD =76	BRD =123, EDIVM1 =10	BRD =1248
4800	BRD =154	BRD =248, EDIVM1 =10	BRD =2498

Table 13.1-2 UART controller Baud Rate Parameter Setting Example Table

UART Peripheral Clock = 12 MHz			
Baud Rate	**UART_BAUD Value**		
	Mode 0	**Mode 1**	**Mode 2**
921600	Not support	Not recommended	0x3000_000B
460800	Not recommended	0x2D00_0000	0x3000_0018
230400	Not recommended	0x2D00_0002	0x3000_0032
115200	Not recommended	0x2D00_0006	0x3000_0066
57600	0x0000_000B	0x2D00_000E	0x3000_00CE
38400	0x0000_0012	0x2D00_0016	0x3000_0137
19200	0x0000_0025	0x2500_007B	0x3000_026F
9600	0x0000_004C	0x2A00_007B	0x3000_04E0
4800	0x0000_009A	0x2A00_00F8	0x3000_09C2

Table 13.1-3 UART controller Baud Rate Register Setting Example Table

The UART controller supports baud rate compensation function. It is used to optimize the precision in

each bit. The precision of the compensation is half of UART module clock because there is BRCOMDEC bit (UART_BRCOMP[31]) to define the positive or negative compensation in each bit. If the BRCOMPDEC (UART_BRCOMP[31]) = 0, it is positive compensation for each bit, one more module clock will be append in the compensated bit. If the BRCOMPDEC (UART_BRCOMP[31]) = 1, it is negative compensation for each bit, decrease one module clock in the compensated bit.

There is 9-bits location, BRCOMP[8:0] (UART_BRCOMP[8:0]), can be configured by user to define the relative bit is compensated or not. BRCOMP[7:0] is used to define the compensation of UART_DAT[7:0] and BRCOMP[8] is used to define the parity bit

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_FIFO [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 (FUNCSEL(UART_FUNCSEL[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 NUC980 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 (FUNCSEL(UART_FUNCSEL[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 (FUNCSEL(UART_FUNCSEL[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.

13.2 Features

- Full-duplex asynchronous communications
- Separates receive and transmit 16/16 bytes entry FIFO for data payloads
- Supports hardware auto-flow control
- Programmable receiver buffer trigger level
- Supports programmable baud rate generator for each channel individually
- Supports nCTS, incoming data, Received Data FIFO reached threshold and RS-485 Address Match (AAD mode) 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 (UART_TOUT [15:8])
- Supports Auto-Baud Rate measurement and baud rate compensation function
- Support 9600 bps for UART_CLK is selected LXT.
- Supports break error, frame error, parity error and receive/transmit buffer overflow detection 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 for 3/16 bit duration for normal mode
- Supports LIN function mode (Only UART1 /UART2 with LIN function)
- Supports LIN master/slave mode
- Supports programmable break generation function for transmitter
- Supports break detection function for receiver
- Supports RS-485 function mode
- Supports RS-485 9-bit mode
- Supports hardware or software enables to program nRTS pin to control RS-485 transmission direction
- Supports PDMA transfer function

13.3 Block Diagram

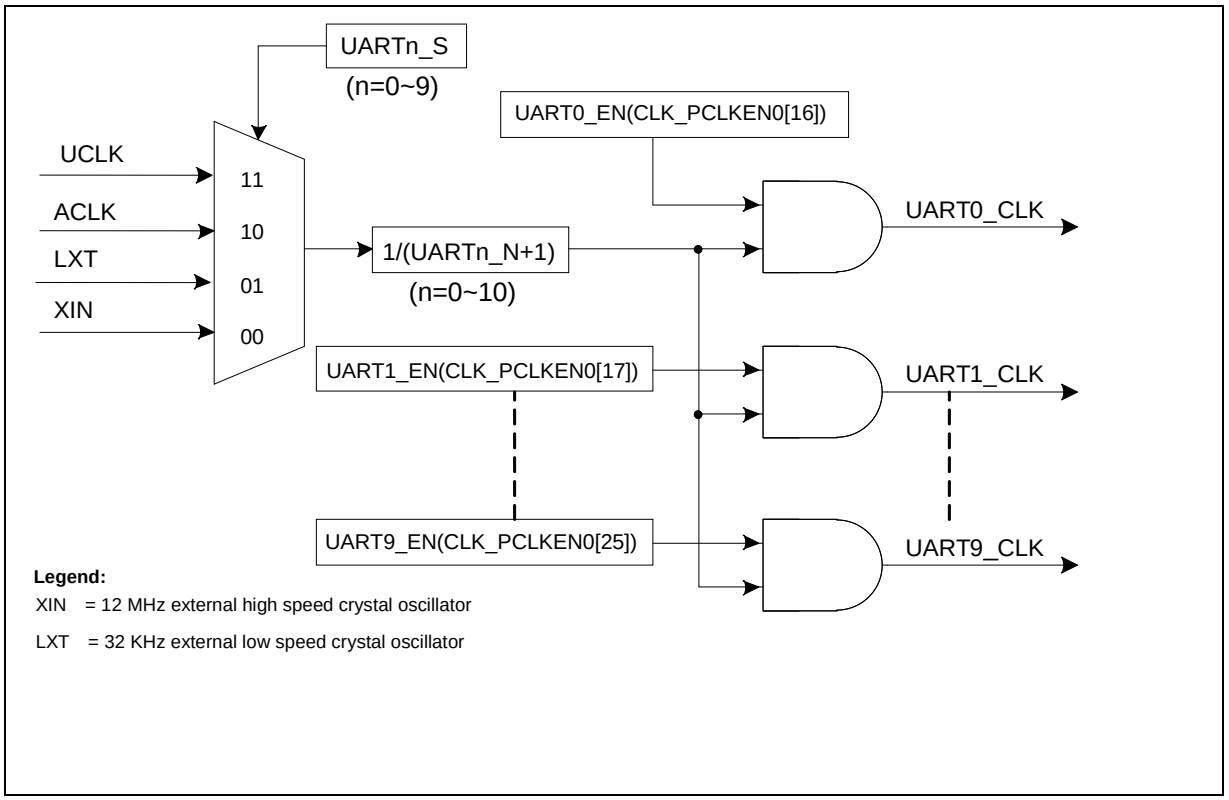


Figure 13.3-1 UART Clock Source

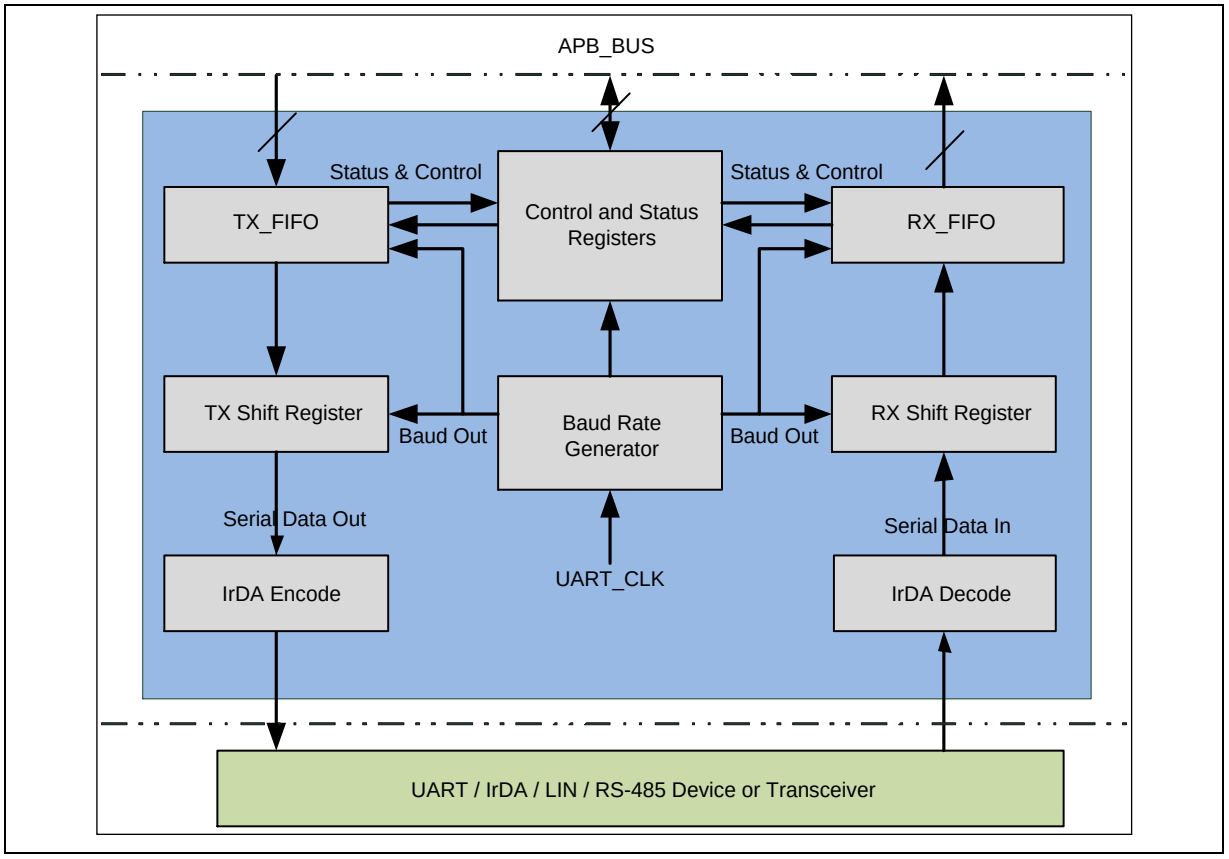


Figure 13.3-2 UART Block Diagram

13.4 Functional Description

13.4.1 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.

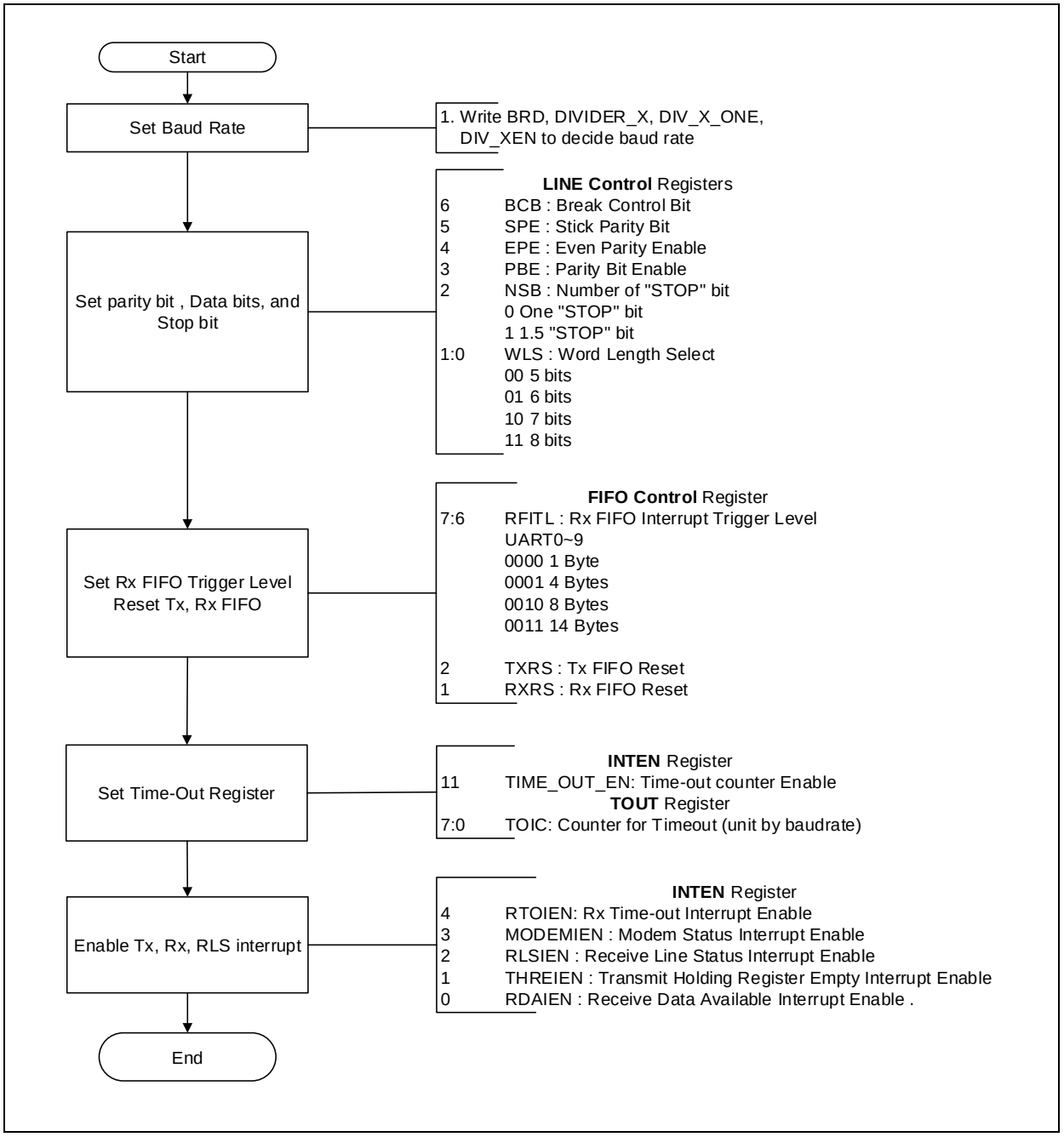


Figure 13.4-1 UART Initialization Flow

13.4.2 IrDA Mode

The UART Controller provides Serial IrDA (SIR, Serial Infrared) transmit encoder and receive decoder

function. The IrDA_EN(UART_FUNCSEL[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(UART_FIFO[7:4]) = 0000 and the BAUDM1(UART_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(UART_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_FUNCSEL[2:0] = 010) =1, to select IrDA function.
2. Set TXINV(UART_IRDA[5]) = 0. (Not inverse TX output signal)
3. Set RXINV(UART_IRDA[6]) = 1. (Inverse RX input signal)
4. Setting TX_SELECT (UART_IRDA [2]) to select half- tandem as TX or RX.
   - TXEN(UART_IRDA[2]) = 1 ➔ select TX.
   - TXEN(UART_IRDA[2]) = 0 ➔ select RX.

13.4.3 RS485 Function Mode

The UART supports RS-485 9-bit mode function. The RS-485 mode is selected by setting the FUNCSEL(UART_FUNCSEL[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 UART_LINE register to control the 9-th bit (When the PBE(UART_LINE[3]), EPE(UART_LINE[4]) and SPE(UART_LINE[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 UART_ALTCTL 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(UART_TOUT [15:8]).

13.4.3.1 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 RXOFF(UART_FIFO [8]) then enable RS485NMM(UART_ALTCTL[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 RXOFF.

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 (RXOFF(UART_FIFO[8]) is low, all received byte data will be accepted and stored in the RX-FIFO, and if the receiver is disabled (RXOFF(UART_FIFO[8]) is high, all received byte data will be ignore until the next address byte be detected.

If software disable receiver by setting (RXOFF(UART_FIFO[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 FUNCSEL(UART_FUNCSEL[2:0]) to select RS-485 function.
2. Program the RXOFF(UART_FIFO[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 RS485NMM(UART_ALTCTL[8]).
4. When an address byte is detected (bit 9 = “1”), hardware will set RLSIF(UART_INTSTS[2]) and ADDRDETF(UART_FIFOSTS[3]) flag.
5. Software can decide whether to accept the following data byte by setting RXOFF(UART_FIFO[8]).
6. Repeat step 4 and step 5.

13.4.3.2 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 ADDRMV(UART_ALTCTL[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 ADDRMV(UART_ALTCTL[31:24]) value. In RS-485 AAD mode, don"t fill any value to RXOFF(UART_FIFO[8]) bit.

Program Sequence example：

1. Program FUNCSEL(UART_FUNCSEL[1:0]) to select RS-485 function.
2. Program the RS485AAD(UART_ALTCTL[9]).
3. When an address byte is detected (bit9 = “1”), hardware will compare the address byte and the ADDRMV (UART_ALTCTL[31:24]) value.
4. If the address byte matches the ADDRMV(UART_ALTCTL[31:24]) value, hardware will set RLSIF(UART_INTSTS[2]) and ADDRDETF(UART_FIFOSTS[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 ADDRMV(UA_ALTCTL[31:24]) value, hardware will ignored the address byte data and ignored the following data transfer.
6. Respect step 3 and step 4.

13.4.3.3 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 setting RTSACTLV(UART_MODEM[9]) to change the RTS driving level.

13.4.4 LIN (Local Interconnection Network) Mode

The UART supports LIN function. The LIN mode is selected by setting the (FUNCSEL(UART_FUNCSEL[2:0]) = 001).

According to the LIN protocol, all information is transmitted packed as frames; a frame consist (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.

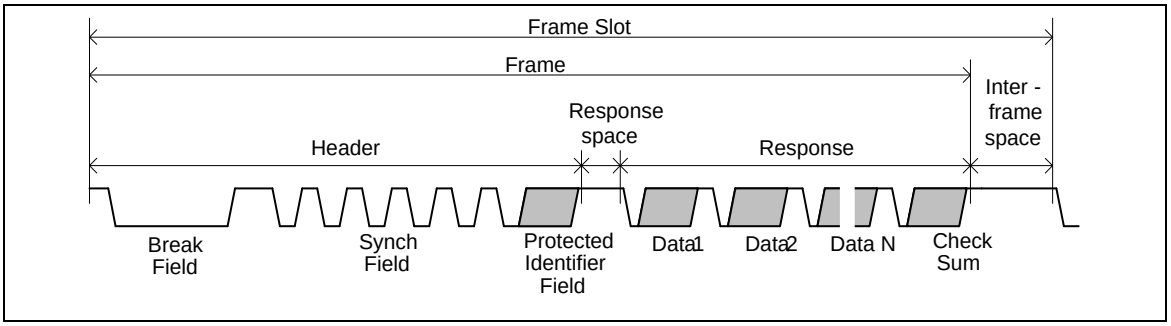


Figure 13.4-2 UART LIN Mode

LIN Transmission (TX) Program Sequence：

1. Select LIN function mode by setting UART_FUNCSEL register
2. Select Break filed and Delimiter Length by setting BRKFL(UART_LINCTL[19:16]) and BSL(UART_LINCTL[21:20]).
3. Set SENDH(UART_LINCTL[8]) to start transfer. (When transmitter header field (it may be “break” or “break + sync” or “break + sync + frame ID” selected by HSEL(UART_LINCTL[23:22]) field) transfer operation finished, this bit will be cleared automatically).
4. Request sync field transmission by writing 0x55 into UART_DAT register.
5. Request header frame ID transmission by writing the protected identifier value in the UART_DAT register.
6. Wait for the TXEMPTYF(UART_FIFOSTS[28]) flag
7. Write N bytes data and checksum value to UART_DAT register. Repeat step 5 and step 6.

LIN Receive(RX) Program Sequence：

1. Select LIN function mode by setting UART_FUNCSEL register
2. Set SLVEN(UART_ALTCTL[0]) = 1 to enable LIN Slave mode
3. Wait BRKDETF(UART_LINSTS[8]) flag. (This bit is set by hardware when a break is detected).
4. Wait for the RDAIF(UART_INTSTS[0]) flag and read back the UART_DAT register

13.4.5 PDMA Transfer Function

The UART controller supports PDMA transfer function.

By configuring PDMA parameter and set UART_DAT as the PDMA destination address. When TXPDMAEN (UART_INTEN[14]) is set to 1, the controller will issue request to PDMA controller to start the PDMA transmission process automatically.

By configuring PDMA parameter and set UART_DAT as the PDMA source address. When RXPDMAEN (UART_INTEN[15]) is set to 1, the controller will start the PDMA reception process. UART controller will issue request to PDMA controller automatically when there is data in the RX FIFO buffer.

Note: If STOPn (PDMA_STOP[n]) is set to stop UART RXPDMA task and the UART receive is not finish. UART controller will complete the transfer and stored current receive data in receive buffer. By

reading RXEMPTY (UART_FIFOSTS[14]) to check there is valid data in receive buffer or not.

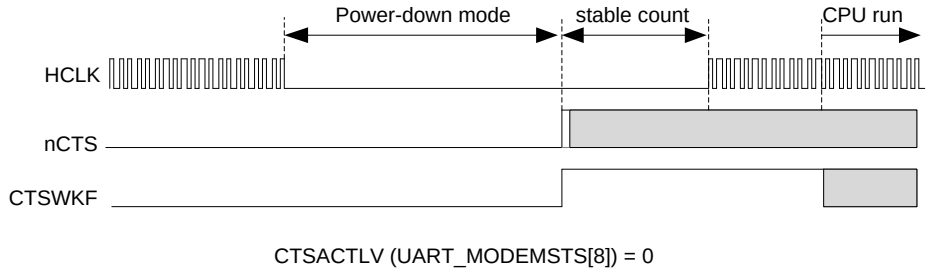
13.4.6 UART Controller Wake-up Function

The UART controller supports wake-up system function. The wake-up function includes nCTS pin, incoming data wake-up, Received Data FIFO reached threshold wake-up, RS-485 Address Match (AAD mode) wake-up and Received Data FIFO threshold time-out wake-up function. CTSWKF (UART_WKSTS[0]), DATWKF (UART_WKSTS[1]), RFRTWKF (UART_WKSTS[2]), RS485WKF (UART_WKSTS[3]) or TOUTWKF (UART_WKSTS[4]) cause the wake-up interrupt flag WKIF(UART_INTSTS[6]) is generated. If the WKIEN (UART_INTEN[6]) is enabled, the wake-up interrupt flag WKIF(UART_INTSTS[6]) cause the wake-up interrupt WKINT (UART_INTSTS[14]) is generated.

nCTS pin wake-up :

When the system is in Power-down mode and WKCTSEN (UART_WKCTL[0]) is set, the toggle of nCTS pin can wake-up system. If the WKCTSEN (UART_WKCTL[0]) is enabled, the toggle of nCTS pin cause the nCTS wake-up flag CTSWKF (UART_WKSTS[0]) is generated. The nCTS wake-up is shown in Figure 13.4-3 and Figure 13.4-4.

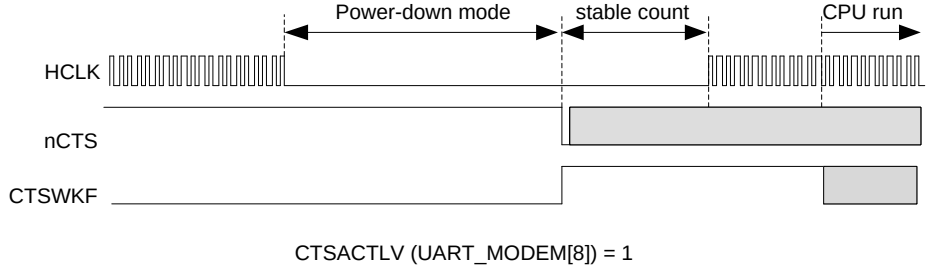
nCTS Wake-up Case 1 (nCTS transition from low to high)



Note: Stable count means HCLK source recovery stable count.

Figure 13.4-3 UART nCTS Wake-up Case1

nCTS Wake-up Case 2 (nCTS transition from high to low)



Note: Stable count means HCLK source recovery stable count.

Figure 13.4-4 UART nCTS Wake-up Case2

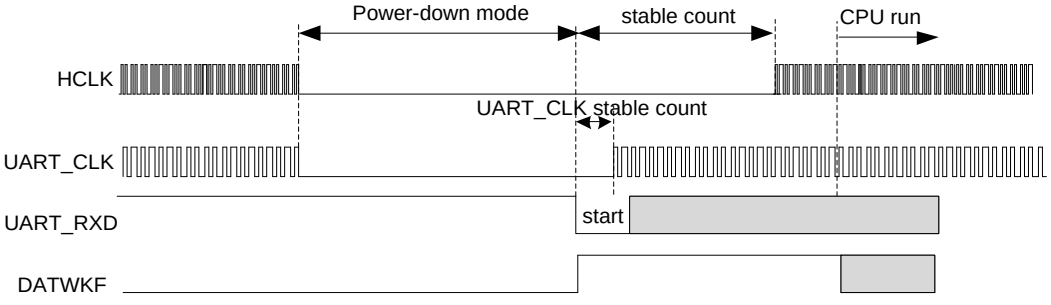
Incoming Data Wake-up

When system is in Power-down mode and the WKDATEN (UART_WKCTL [1]) is set, the toggle of incoming data (UART_RXD) pin can wake-up the system. In order to receive the incoming data after the system wake-up, the STCOMP (UART_DWKCOMP[15:0]) shall be set. These bits field of STCOMP indicate how many clock cycle selected by UART_CLK do the UART controller can get the 1st bit (start bit) when the system is wakeup from Power-down mode.

When incoming data wakes system up, the incoming data will be received and stored in FIFO. If the WKDATEN (UART_WKCTL[1]) is enabled, the toggle of incoming data (UART_RXD) pin cause the incoming data wake-up flag DATWKF (UART_WKSTS[1]) is generated. The imcoing data wake-up is shown in Figure 13.4-5.

Note1: The UART controller clock source should be selected as HIRC and the compensation time for start bit is about 10.865us. It means that the value of STCOMP (UART_DWKCOMP[15:0]) can be set as 0x207.

Note2: The value of BRD(UART_BAUD[15:0]) should be greater than STCOMP (UART_DWKCOMP[15:0]).



Note1: Stable count means HCLK source recovery stable count.

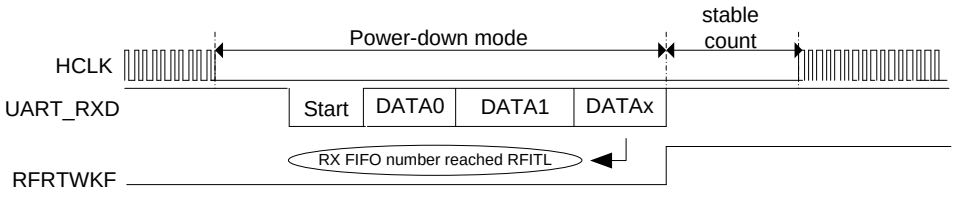
Note2: UART_CLK stable count means UART clock source recovery stable count.

Figure 13.4-5 UART Data Wake-up

Received Data FIFO Reached Threshold Wake-up

The received data FIFO threshold reached wake-up function is enabled by setting WKRFRTEN (UART_WKCTL[2]). In Power-down mode, when the number of received data in RX FIFO reaches the threshold value RFITL (UART_FIFO[7:4]), it can wake-up the system. If the WKRFRTEN (UART_WKCTL[2]) is enabled, the number of received data in RX FIFO reaches the threshold value RFITL (UART_FIFO[7:4]) cause the received data FIFO reached threshold wake-up flag RFRTWKF (UART_WKSTS[2]) is generated. The Received Data FIFO reached threshold wake-up is shown in Figure 13.4-6.

Note: The UART controller clock source should be selected as LXT in Power-down mode to receive data.



Note: Stable count means HCLK source recovery stable count.

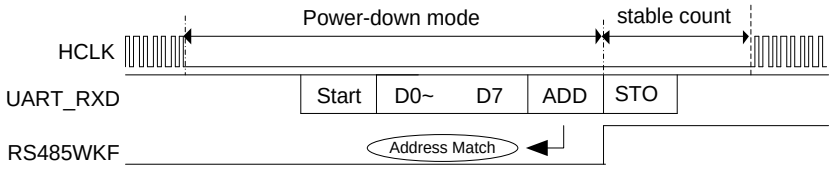
Figure 13.4-6 UART Received Data FIFO reached threshold wake-up

RS-485 Address Match (AAD Mode) Wake-up

The RS-485 address match wake-up function is enabled by setting WKRFRTEN (UART_WKCTL[2]) and WKRS485EN (UART_WKCTL[3]). This function is used for RS-485 Auto Address Detection (AAD) mode in RS-485 function mode and ADDRDEN (UART_ALTCTL[15]) is set to 1. In Power-down mode, when an address byte is detected and matches the ADDRMV (UART_ALTCTL[31:24]) or

the number of received data in RX FIFO reaches the threshold value RFITL (UART_FIFO[7:4]), it can wake-up the system. If the WKRS485EN (UART_WKCTL[3]) is enabled, when an address byte is detected and matches the ADDRMV (UART_ALTCTL[31:24]) that cause the RS485 address match (AAD mode) wake-up flag RS485WKF (UART_WKSTS[3]) is generated. The RS-485 Address Match (AAD mode) wake-up is shown in Figure 13.4-7.

Note: The UART controller clock source should be selected as LXT in Power-down mode to receive data.



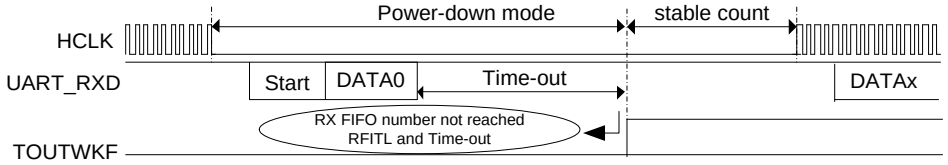
Note: Stable count means HCLK source recovery stable count.

Figure 13.4-7 UART RS-485 AAD Mode Address Match Wake-up

Received Data FIFO Threshold Time-out Wake-up

The received data FIFO threshold time-out wake-up function is enabled by setting WKRFRTEN (UART_WKCTL[2]) and WKTOUTEN (UART_WKCTL[4]). Setting TOCNTEN (UART_INTEN[11]) to enable receiver buffer time-out counter. In Power-down mode, when the number of received data in RX FIFO does not reach the threshold value RFITL (UART_FIFO[7:4]) and the time-out counter equals to the time-out value TOIC (UART_TOUT[7:0]), it can wake-up the system. If the WKTOUTEN (UART_WKCTL[4]) is enabled, when the time-out counter equals to the time-out value TOIC (UART_TOUT[7:0]) that cause the Received Data FIFO threshold time-out wake-up wake-up flag TOUTWKF (UART_WKSTS[4]) is generated. The Received Data FIFO threshold time-out wake-up is shown in Figure 13.4-8.

Note: The UART controller clock source should be selected as LXT in Power-down mode to receive data.



Note: Stable count means HCLK source recovery stable count.

Figure 13.4-8 UART Received Data FIFO threshold time-out wake-up

13.5 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 = 0xB007_0000** **Channel1 : UART1_BA = 0xB007_1000** **Channel2 : UART2_BA = 0xB007_2000** **Channel3 : UART3_BA = 0xB007_3000** **Channel4 : UART4_BA = 0xB007_4000** **Channel5 : UART5_BA = 0xB007_5000** **Channel6 : UART6_BA = 0xB007_6000** **Channel7 : UART7_BA = 0xB007_7000** **Channel8 : UART8_BA = 0xB007_8000** **Channel9 : UART9_BA = 0xB007_9000**				
UART_DAT	UART_BA+0x00	R/W	UART Receive/Transmit Buffer Register	Undefined
UART_INTEN	UART_BA+0x04	R/W	UART Interrupt Enable Register	0x0000_0000
UART_FIFO	UART_BA+0x08	R/W	UART FIFO Control Register	0x0000_0101
UART_LINE	UART_BA+0x0C	R/W	UART Line Control Register	0x0000_0000
UART_MODEM	UART_BA+0x10	R/W	UART Modem Control Register	0x0000_0200
UART_MODEMSTS	UART_BA+0x14	R/W	UART Modem Status Register	0x0000_0110
UART_FIFOSTS	UART_BA+0x18	R/W	UART FIFO Status Register	0xB040_4000
UART_INTSTS	UART_BA+0x1C	R/W	UART Interrupt Status Register	0x0040_0002
UART_TOUT	UART_BA+0x20	R/W	UART Time-out Register	0x0000_0000
UART_BAUD	UART_BA+0x24	R/W	UART Baud Rate Divisor Register	0x0F00_0000
UART_IRDA	UART_BA+0x28	R/W	UART IrDA Control Register	0x0000_0040
UART_ALTCTL	UART_BA+0x2C	R/W	UART Alternate Control/Status Register	0x0000_000C
UART_FUNCSEL	UART_BA+0x30	R/W	UART Function Select Register	0x0000_0000
UART_LINCTL	UART_BA+0x34	R/W	UART LIN Control Register	0x000C_0000
UART_LINSTS	UART_BA+0x38	R/W	UART LIN Status Register	0x0000_0000
UART_BRCOMP	UART_BA+0x3C	R/W	UART Baud Rate Compensation Register	0x0000_0000
UART_WKCTL	UART_BA+0x40	R/W	UART Wake-up Control Register	0x0000_0000
UART_WKSTS	UART_BA+0x44	R/W	UART Wake-up Status Register	0x0000_0000
UART_DWKCOMP	UART_BA+0x48	R/W	UART Incoming Data Wake-up Compensation Register	0x0000_0000

14 SMART CARD HOST INTERFACE (SC)

14.1 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.

14.2 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

14.3 Block Diagram

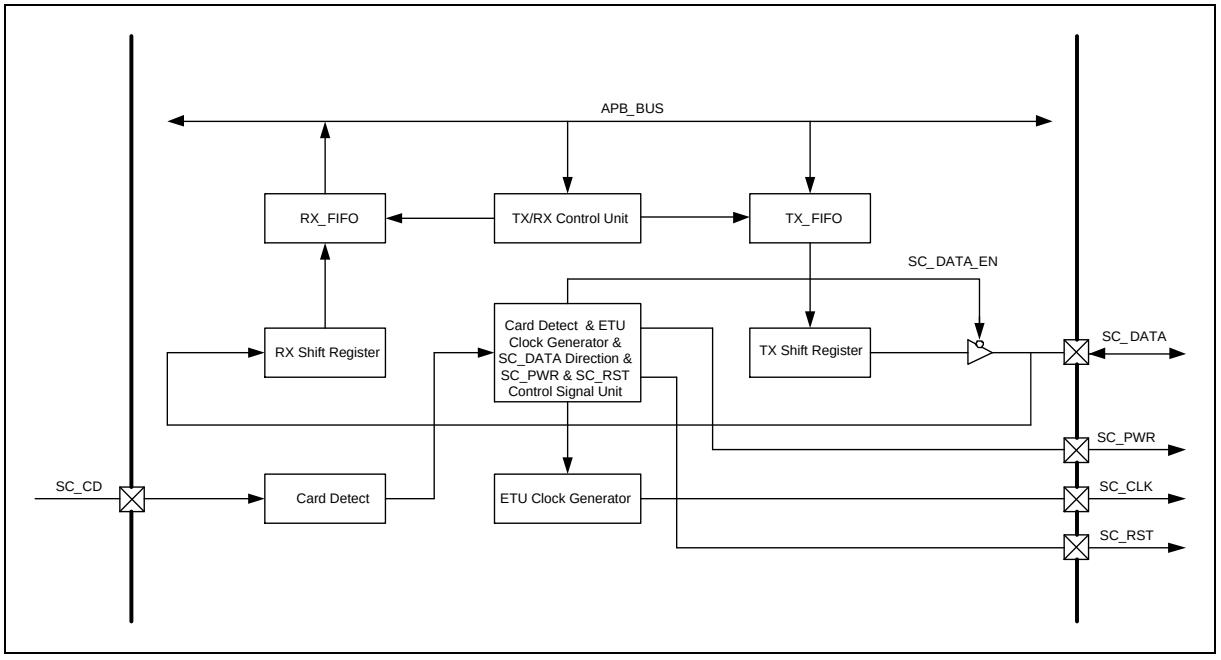


Figure 14.3-1 Smart Card Host Interface Block Diagram

14.4 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.

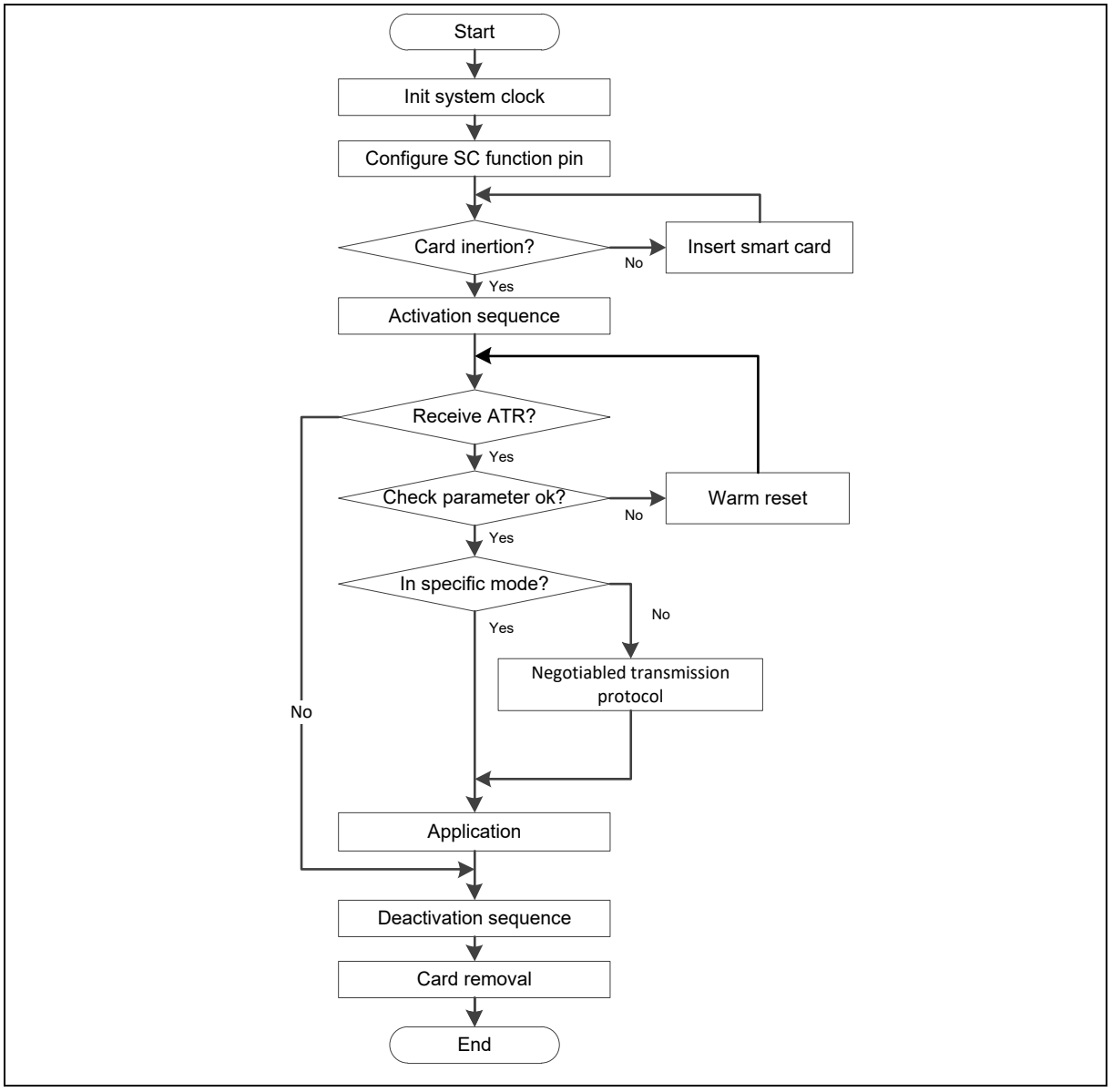


Figure 14.4-1 Smart Card Activate/Deactive Flow

14.4.1 Activation (Cold Reset)

The Smart Card Interface controller supports hardware activation, warm reset and deactivation sequence. The activation sequence is 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:

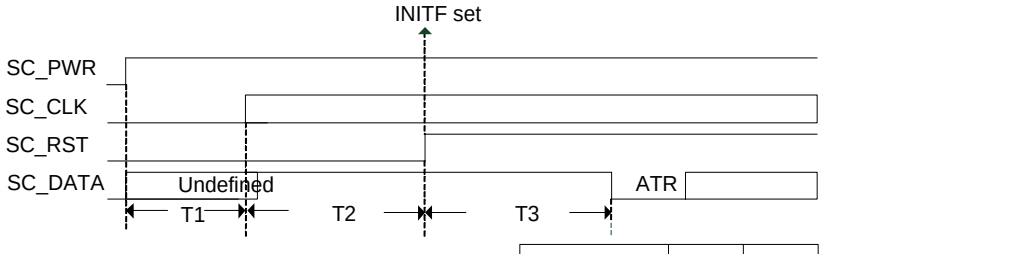
Set SC_PINCTL and SC_TMRCTLx (x = 0, 1, 2) to process the activation sequence. SC_PWR, SC_CLK, SC_RST and SC_DATA pin state can be programmed by SC_PINCTL. The programming method is shown in Activation description. The activation sequence timing can be controlled by setting SC_TMRCTLx (x = 0, 1, 2). 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]). The SCn_PWR to SCn_CLK length can be configure by setting T1EXT(SCn_ACTCTL[4:0]). 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.



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

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

Unit: SC Clock

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

Figure 14.4-2 Smart Card Cold Reset Timing

14.4.2 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:

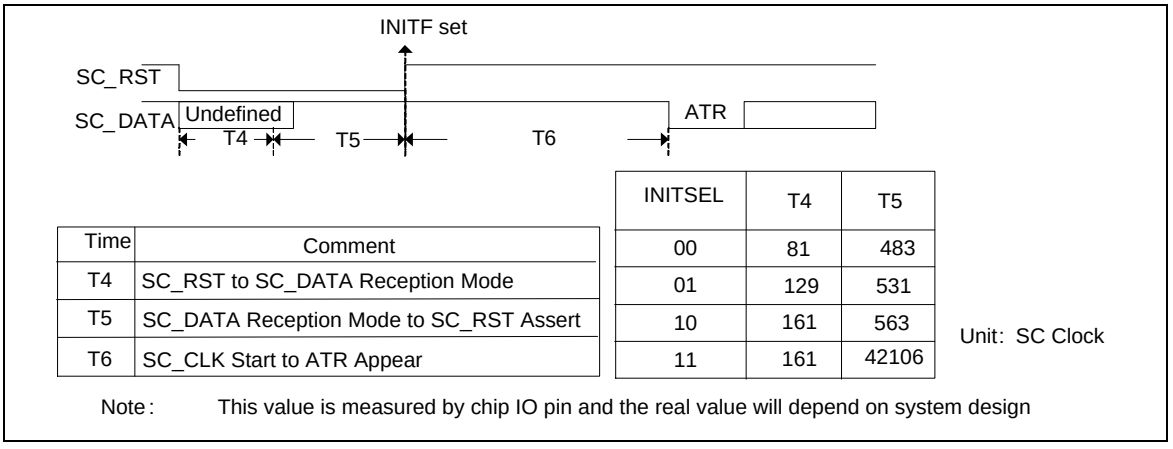
Set SC_PINCTL and SC_TMRCTLx (x = 0, 1, 2) to process the warm reset sequence. SC_RST and SC_DATA pin state can be programmed by SC_PINCTL. The warm reset sequence timing can be controlled by setting SC_TMRCTLx (x = 0, 1, 2). This programming procedure provides user has a flexible timing setting for 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



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

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

Unit: SC Clock

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

Figure 14.4-3 Smart Card Warm Reset Timeing

14.4.3 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.

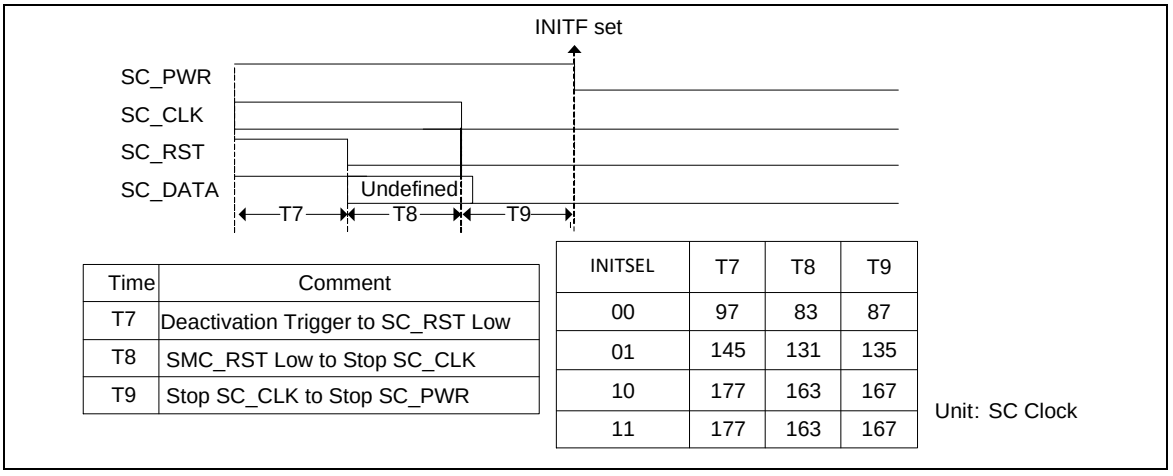
Software Timing Control:

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]).



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

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

Unit: SC Clock

Figure 14.4-4 Smart Card Deactivation Timing

14.4.4 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.

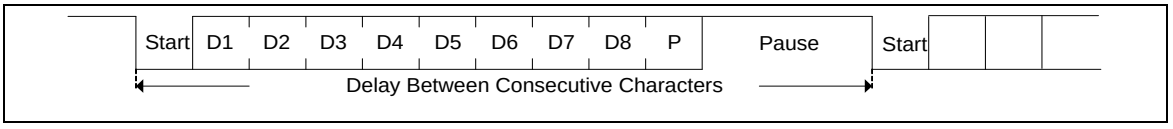


Figure 14.4-5 Smart Card Character Format

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 AUTOCEN (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 AUTOCEN (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 decided 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.

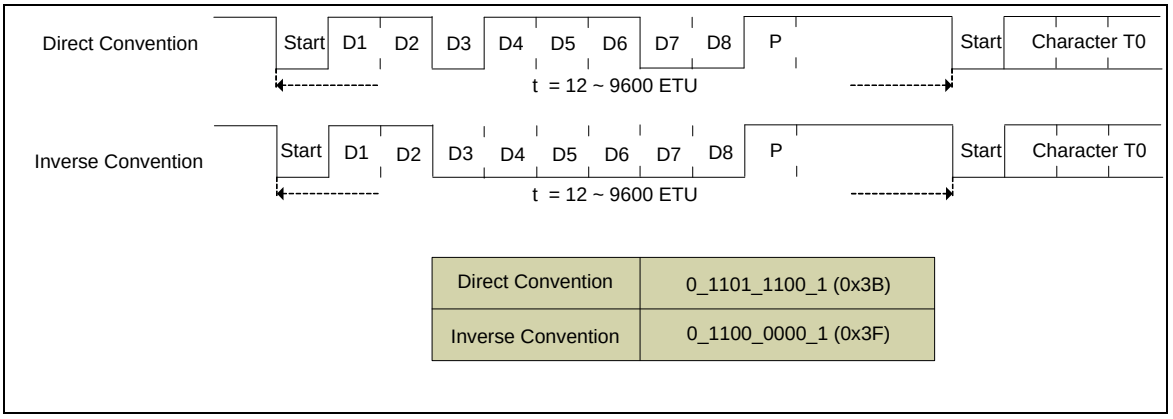


Figure 14.4-6 Smart Card Convention

14.4.5 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 then 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_RXTOUT register using ETU as time unit. Except configure proper value in SC_RXTOUT, 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.

14.4.6 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.

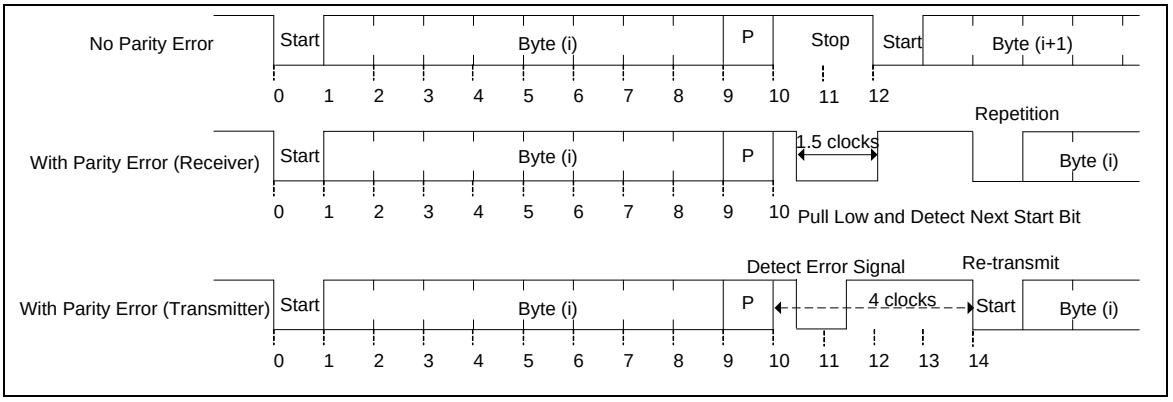


Figure 14.4-7 Smart Card Parity Error Handling

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

14.4.7 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 CNTEN0 (SC_ALTCTL[5]) automatically.
0100	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]). 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. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.	
0101	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]). 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. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.	
0110	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]). 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. The time-out value will be CNT (SC_TMRCTL0[23:0], SC_TMRCTL1[7:0], SC_TMRCTL2[7:0]) + 1.	
0111	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. 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.	
	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.
1111	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. 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.	
	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”.

Table 14.4-1 Smartcard Internal Timer Operating Mode

14.4.8 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

14.4.9 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 BGTIF (SC_INTSTS[6]) will be set 1. Write 1 can clear BGTIF bit.

- Extra Guard Time (EGT)

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

14.4.10 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 AUTOCEN (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.

14.5 Register Map

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

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

15 I²C

15.1 Overview

I²C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. The I²C 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.

There are four sets of I²C controllers which support Power-down wake-up function

15.2 Features

- Supports up to four I²C ports
- Master/Slave mode
- Bidirectional data transfer between masters and slaves
- Multi-master bus (no central master)
- Supports Standard mode (100 kbps), Fast mode (400 kbps) and Fast mode plus (1 Mbps)
- Arbitration between simultaneously transmitting masters without corruption of serial data on the bus
- Serial clock synchronization allow devices with different bit rates to communicate via one serial bus
- Serial clock synchronization used as a handshake mechanism to suspend and resume serial transfer
- Built-in 14-bit time-out counter requesting the I²C interrupt if the I²C bus hangs up and timer-out counter overflows
- Programmable clocks allow for versatile rate control
- Supports 7-bit addressing and 10-bit addressing mode
- Supports multiple address recognition (four slave address with mask option)
- Supports Power-down wake-up function
- Supports setup/hold time programmable

15.3 Block Diagram

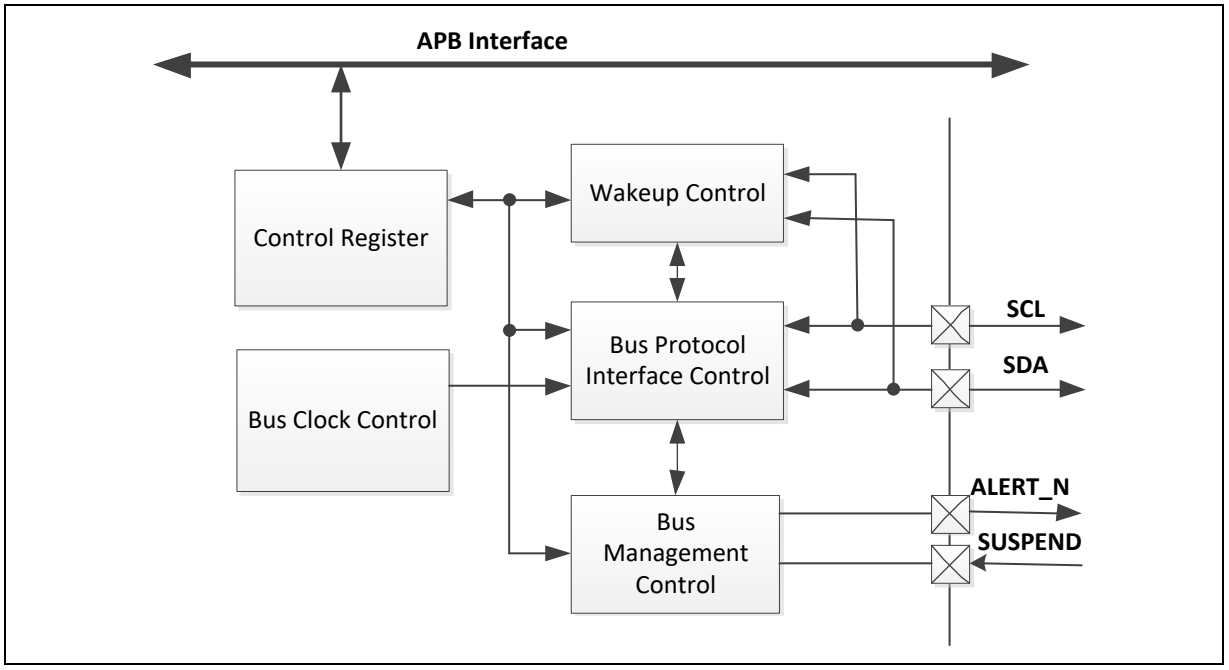


Figure 15.3-1 I²C Controller Block Diagram

15.4 Functional Description

On I²C bus, data is transferred between a Master and a Slave. Data bits transfer on the SCL and SDA lines are synchronously on a byte-by-byte basis. Each data byte is 8-bit long. There is one SCL clock pulse for each data bit with the MSB being transmitted first, and 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). Please refer to Figure 15.4-1 for more detailed I²C BUS Timing.

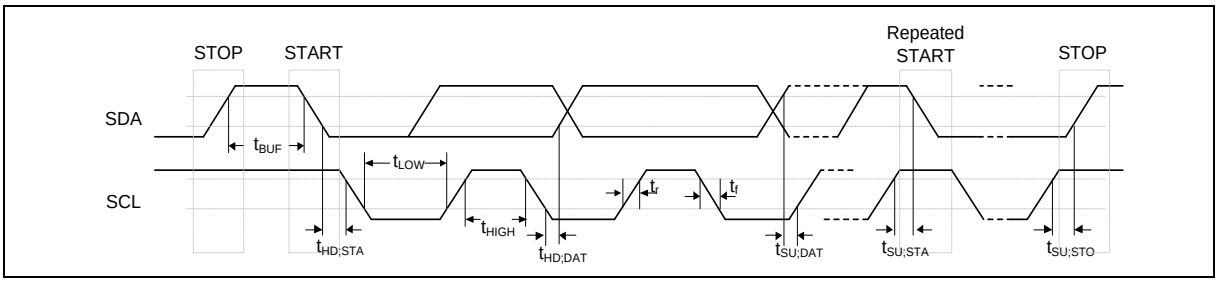


Figure 15.4-1 I²C Bus Timing

The device's on-chip I²C provides the serial interface that meets the I²C bus standard mode specification. The I²C port handles byte transfers autonomously. To enable this port, the bit I2CEN in I2C_CTL0 should be set to '1'. The I²C hardware interfaces to the I²C bus via two pins: SDA and SCL. When I/O pins are used as I²C ports, user must set the pins function to I²C in advance.

Note: Pull-up resistor is needed for I²C operation as the SDA and SCL are open-drain pins.

15.4.1 I²C Protocol

The following figure shows the typical I²C 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

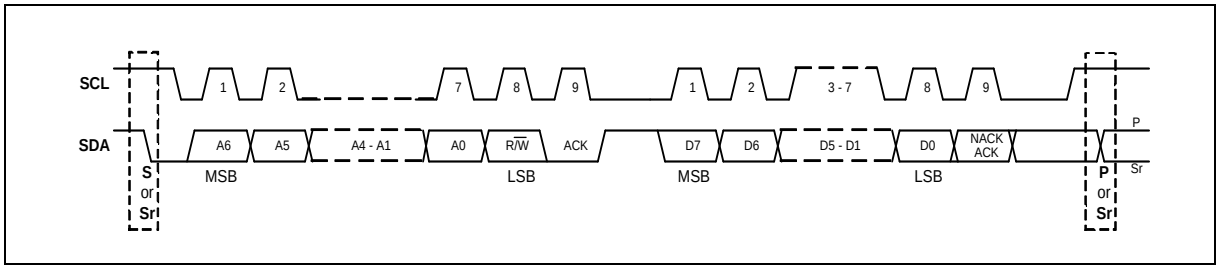


Figure 15.4-2 I2C Bus Protocol

15.4.2 Operation Modes

The on-chip I²C ports support three operation modes, Master, Slave, and General Call Mode.

In a given application, I²C port may operate as a master or as a slave. In Slave mode, the I²C port hardware looks for its own slave address and the general call address. If one of these addresses is detected, and if the slave is willing to receive or transmit data from/to master(by setting the AA bit), acknowledge pulse will be transmitted out on the 9th clock, hence an interrupt is requested on both

master and slave devices if interrupt is enabled. When the microprocessor wishes to become the bus master, hardware waits until the bus is free before entering Master mode so that a possible slave action is not be interrupted. If bus arbitration is lost in Master mode, I²C port switches to Slave mode immediately and can detect its own slave address in the same serial transfer.

To control the I²C bus transfer in each mode, user needs to set I2C_CTL0, I2C_DAT registers according to current status code of I2C_STATUS0 register. In other words, for each I²C bus action, user needs to check current status by I2C_STATUS0 register, and then set I2C_CTL0, I2C_DAT registers to take bus action. Finally, check the response status by I2C_STATUS0.

The bits, STA, STO and AA in I2C_CTL0 register are used to control the next state of the I²C hardware after SI flag of I2C_CTL0 [3] register is cleared. Upon completion of the new action, a new status code will be updated in I2C_STATUS0 register and the SI flag of I2C_CTL0 register will be set. But the SI flag will not be set when I²C STOP. If the I²C interrupt control bit INTEN (I2C_CTL0 [7]) is set, appropriate action or software branch of the new status code can be performed in the Interrupt service routine.

Figure 15.4-3 shows the current I²C status code is 0x08, and then set I2C_DATA=SLA+W and (STA,STO,SI,AA) = (0,0,1,x) to send the address to I²C bus. If a slave on the bus matches the address and response ACK, the I2C_STATUS0 will be updated by status code 0x18.

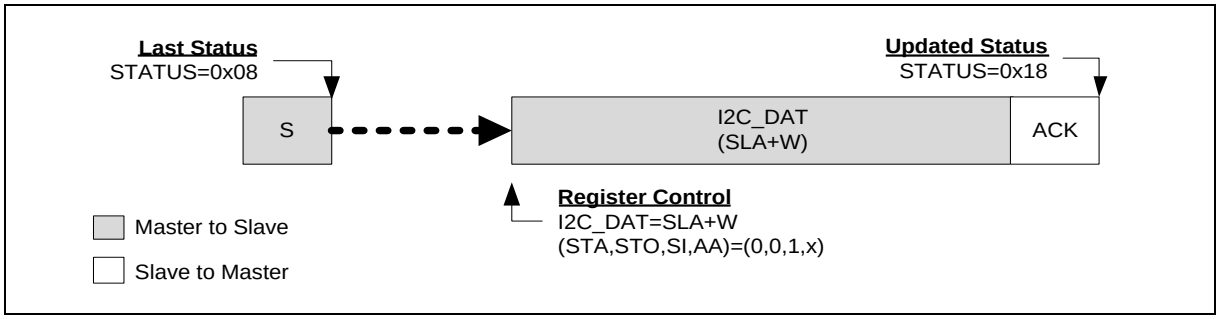


Figure 15.4-3 Control I²C Bus according to the Current I²C Status

Master Mode

In Figure 15.4-4 and Figure 15.4-5, all possible protocols for I²C master are shown. User needs to follow proper path of the flow to implement required I²C protocol.

In other words, user can send a START signal to bus and I²C will be in Master Transmitter (MT) mode (Figure 15.4-4) or Master receiver (MR) mode (Figure 15.4-5) after START signal has been sent successfully and new status code would be 0x08. Followed by START signal, user can send slave address, read/write bit, data and Repeat START, STOP to perform I²C protocol.

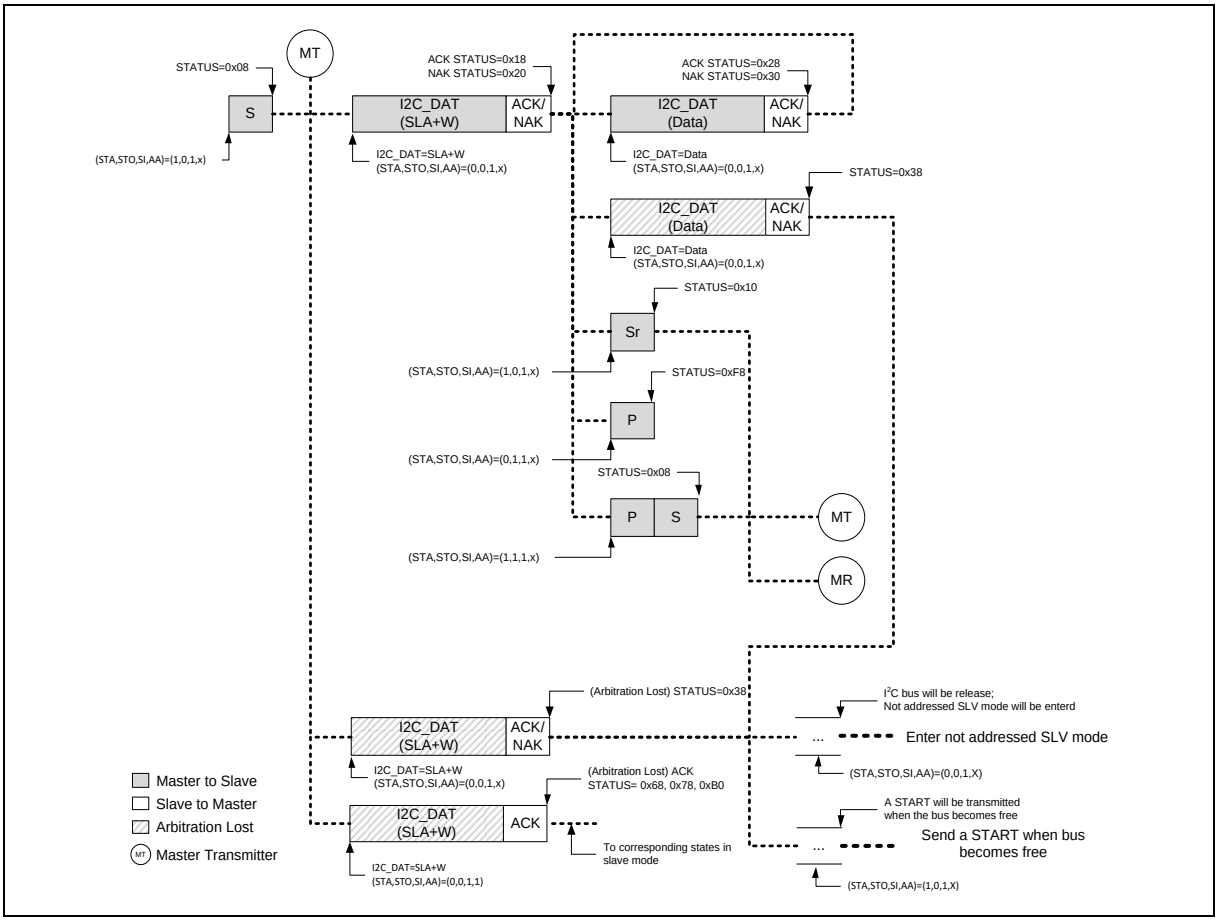


Figure 15.4-4 Master Transmitter Mode Control Flow

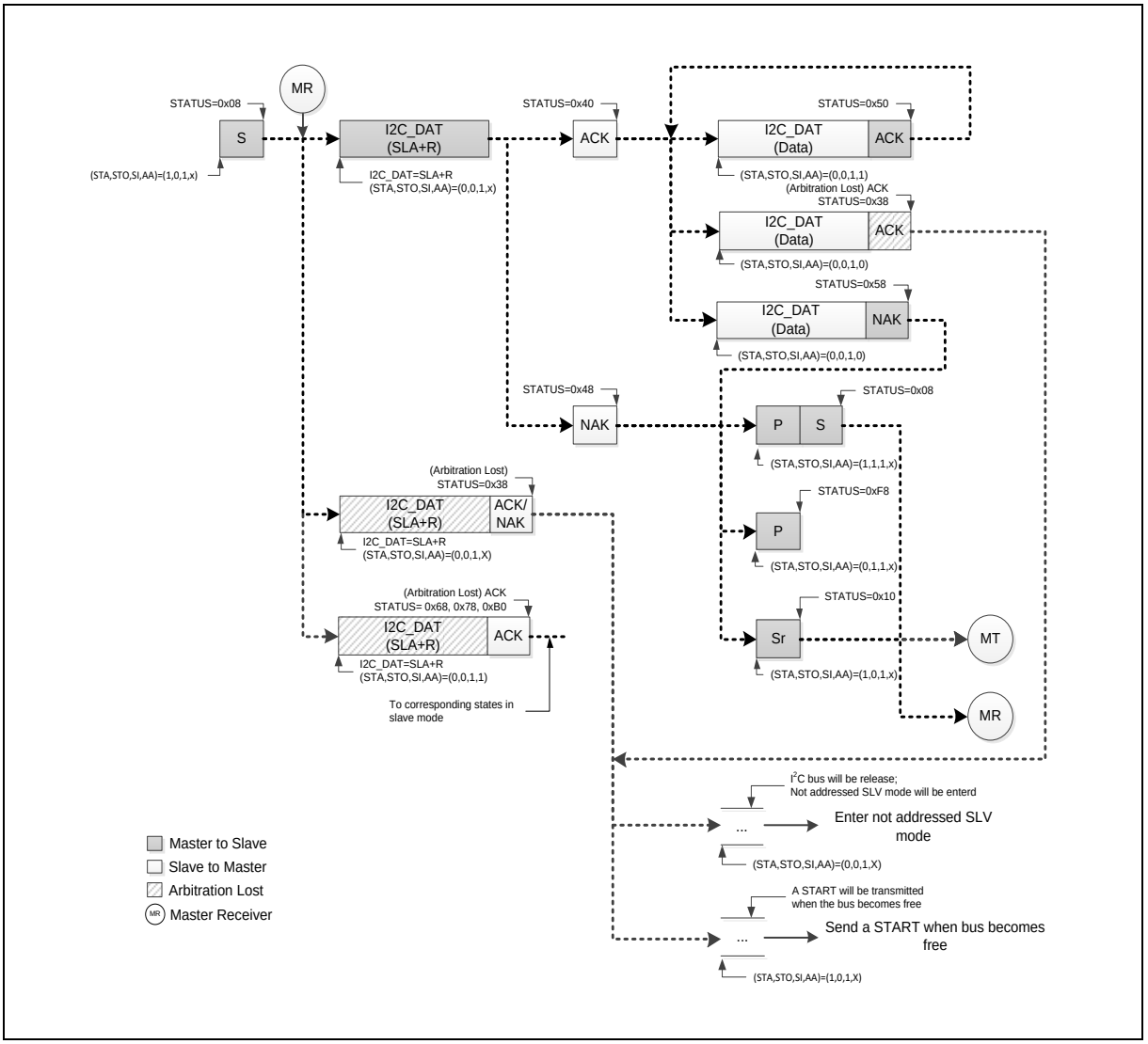


Figure 15.4-5 Master Receiver Mode Control Flow

If the I²C is in Master mode and gets arbitration lost, the status code will be 0x38. In status 0x38, user may set (STA, STO, SI, AA) = (1, 0, 1, X) to send START to re-start Master operation when bus become free. Otherwise, user may set (STA, STO, SI, AA) = (0, 0, 1, X) to release I²C bus and enter not addressed Slave mode.

Slave Mode

When reset default, I²C is not addressed and will not recognize the address on I²C bus. User can set slave address by I2C_ADDRn (n=0~3) and set (STA, STO, SI, AA) = (0, 0, 1, 1) to let I²C recognize the address sent by master. Figure 15.4-6 shows all the possible flow for I²C in Slave mode. Users need to follow a proper flow (as shown in Figure 15.4-6 to implement their own I²C protocol.

If bus arbitration is lost in Master mode, I²C port switches to Slave mode immediately and can detect its own slave address in the same serial transfer. If the detected address is SLA+W (Master want to write data to Slave) after arbitration lost, the status code is 0x68. If the detected address is SLA+R (Master want to read data from Slave) after arbitration lost, the status code is 0xB0.

Note: During I²C communication, the SCL clock will be released when writing '1' to clear SI flag in Slave mode.

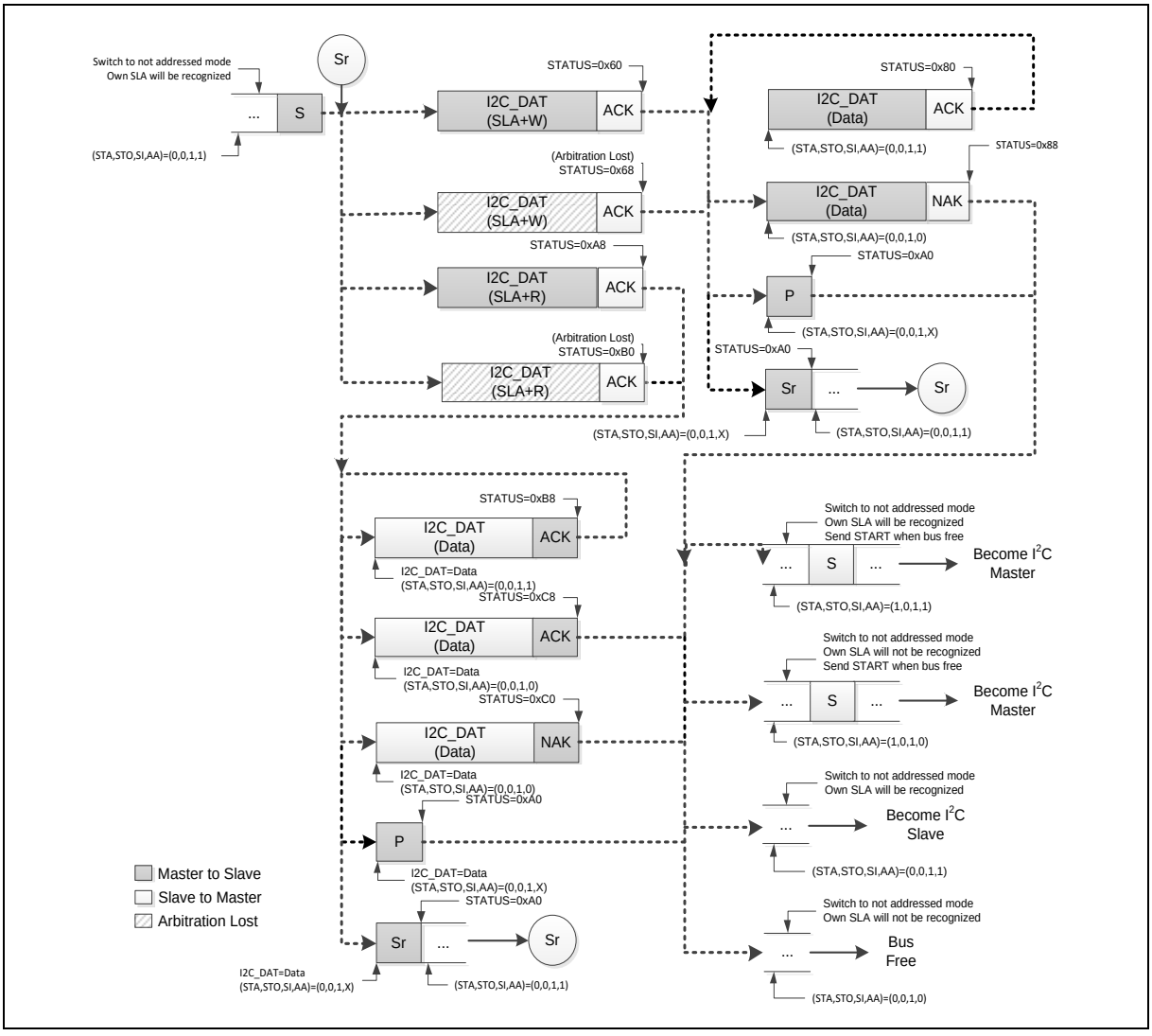


Figure 15.4-6 Slave Mode Control Flow

If I²C is still receiving data in addressed Slave mode but got a STOP or Repeat START, the status code will be 0xA0. User could follow the action for status code 0x88 as shown in the above figure when getting 0xA0 status.

If I²C is still transmitting data in addressed Slave mode but got a STOP or Repeat START, the status code will be 0xA0. User could follow the action for status code 0xC8 as shown in the above figure when getting 0xA0 status.

Note: After slave gets status of 0x88, 0xC8, 0xC0 and 0xA0, slave can switch to not address mode and own SLA will not be recognized. If entering this status, slave will not receive any I²C signal or address from master. At this status, I²C should enter idle mode.

General Call (GC) Mode

If the GC bit (I2C_ADDRn [0]) is set, the I²C port hardware will respond to General Call address (00H). User can clear GC bit to disable general call function. When the GC bit is set and the I²C in Slave mode, it can receive the general call address by 0x00 after master send general call address to I²C

bus, then it will follow status of GC mode.

The GC mode can wake up when address matched. Note that the default address is 0x00, but user must set an address except for 0x00.

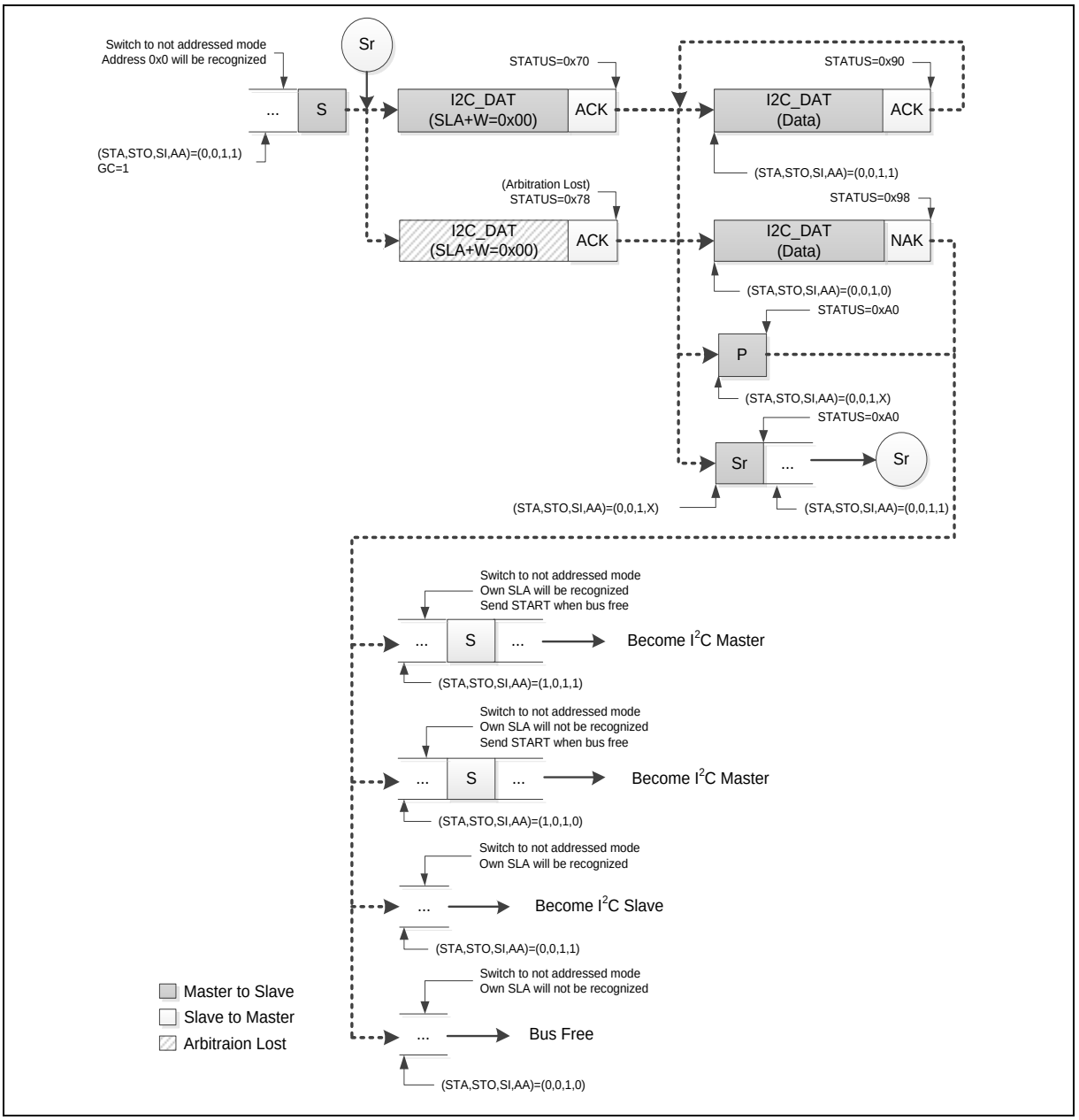


Figure 15.4-7 GC Mode

If I²C is still receiving data in GC mode but got a STOP or Repeat START, the status code will be 0xA0. User could follow the action for status code 0x98 in above figure when getting 0xA0 status.

Note: After slave gets status of 0x98 and 0xA0, slave can switch to not address mode and own SLA will not be recognized. If entering this status, slave will not receive any I²C signal or address from master. At this time, the I²C controller should enter idle mode.

Multi-Master

In some applications, there are two or more masters on the same I²C bus to access slaves, and the masters may transmit data simultaneously. The I²C supports multi-master by including collision detection and arbitration to prevent data corruption.

If for some reason two masters initiate command at the same time, the arbitration procedure determines which master wins and can continue with the command. Arbitration is performed on the SDA signal while the SCL signal is high. Each master checks if the SDA signal on the bus corresponds to the generated SDA signal. If the SDA signal on the bus is low but it should be high, then this master has lost arbitration. The device that has lost arbitration can generate SCL pulses until the byte ends and must then release the bus and go into slave mode. The arbitration procedure can continue until all the data is transferred. This means that in multi-master system each master must monitor the bus for collisions and act accordingly.

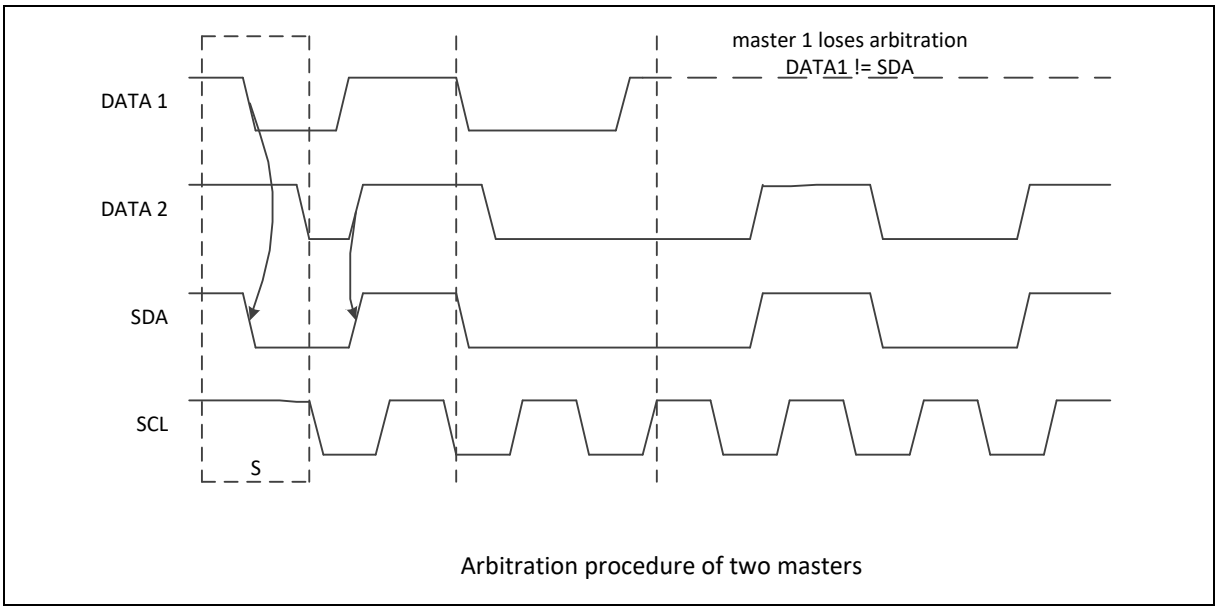


Figure 15.4-8 Arbitration Lost

- When I2C_STATUS0 = 0x38, an “Arbitration Lost” is received. Arbitration lost event maybe occur during the send START bit, data bits or STOP bit. User could set (STA, STO, SI, AA) = (1, 0, 1, X) to send START again when bus free, or set (STA, STO, SI, AA) = (0, 0, 1, X) to not addressed Slave mode. User can detect bus free by ONBUSY (I2C_STATUS1 [8]).
- When I2C_STATUS0 = 0x00, a “Bus Error” is received. To recover I²C bus from a bus error, STO should be set and SI should be cleared, and then STO is cleared to release bus.
  - Set (STA, STO, SI, AA) = (0, 1, 1, X) to stop current transfer
  - Set (STA, STO, SI, AA) = (0, 0, 1, X) to release bus

15.4.3 Example for Random Read on EEPROM

The following steps are used to configure the I2C0 related registers when using I²C to read data from EEPROM.

1. Set I2C0 the multi-function pin as SCL and SDA pins.
2. Enable I2C0 APB clock.
3. Set I2C0RST=1 to reset I2C0 controller then set I2C0 controller to normal operation.
4. Set I2CEN=1 to enable I2C0 controller in the “I2C_CTL0” register.

5. Give I2C0 clock a divided register value for I²C clock rate in the “I2C_CLKDIV”.
6. Install I2C0 IRQ.
7. Set INTEN=1 to enable I2C0 Interrupt in the “I2C_CTL0” register.
8. Set I2C0 address registers “I2C_ADDR0 ~ I2C_ADDR3”.

Random read operation is one of the methods of access EEPROM. The method allows the master to access any address of EEPROM space. Figure 15.4-9 shows the EEPROM random read operation.

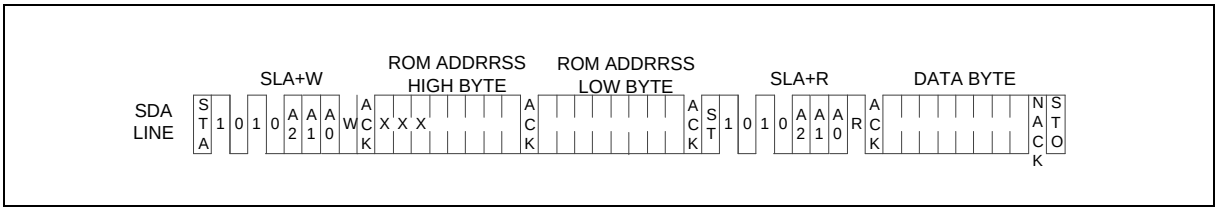


Figure 15.4-9 EEPROM Random Read

Figure 15.4-10 shows how to use the I²C controller to implement the protocol of EEPROM random read.

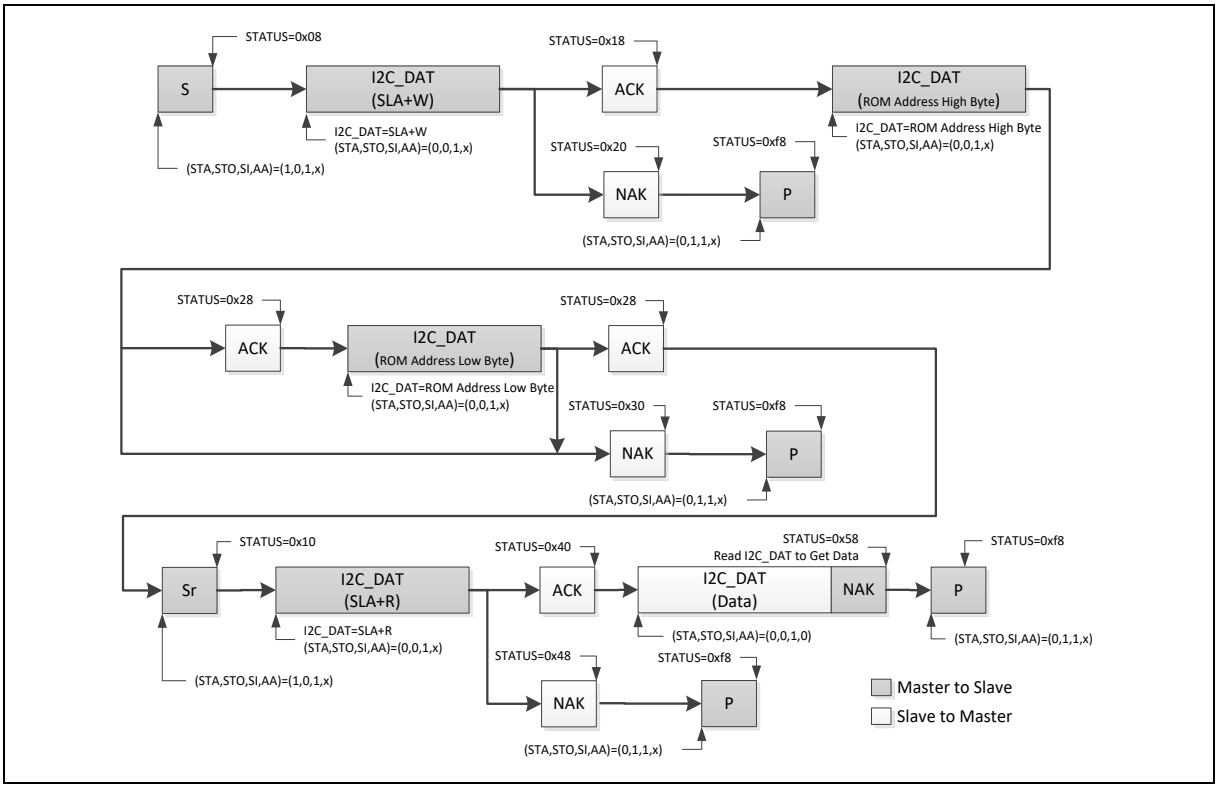


Figure 15.4-10 Protocol of EEPROM Random Read

The I²C controller, which is a master, sends START to bus. Then, it sends a SLA+W (Slave address + Write bit) to EERPOM followed by two bytes data address to set the EEPROM address to read. Finally, a Repeat START followed by SLA+R is sent to read the data from EEPROM.

15.5 Register Map

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

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

16 QSPI

16.1 Overview

The Quad Serial Peripheral Interface (QSPI) applies to synchronous serial data communication and allows full duplex transfer. Devices communicate in Master/Slave mode with the 4-wire bi-direction interface. The NUC980 series contains one QSPI controller performing 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.

The QSPI controller supports 2-bit Transfer mode to perform full-duplex 2-bit data transfer and also supports Dual and Quad I/O Transfer mode and the controller supports the PDMA function to access the data buffer.

16.2 Features

- Supports Master or Slave mode operation
- Supports 2-bit Transfer mode
- Supports Dual and Quad I/O Transfer mode
- Configurable bit length of a transaction word from 8 to 32-bit
- Provides separate 8-level depth transmit and receive FIFO buffers
- Supports MSB first or LSB first transfer sequence
- Supports Byte Reorder function
- Supports Byte or Word Suspend mode
- Supports PDMA transfer
- Supports 3-Wire, no slave selection signal, bi-direction interface
- Supports one data channel half-duplex transfer
- Supports receive-only mode

16.3 Block Diagram

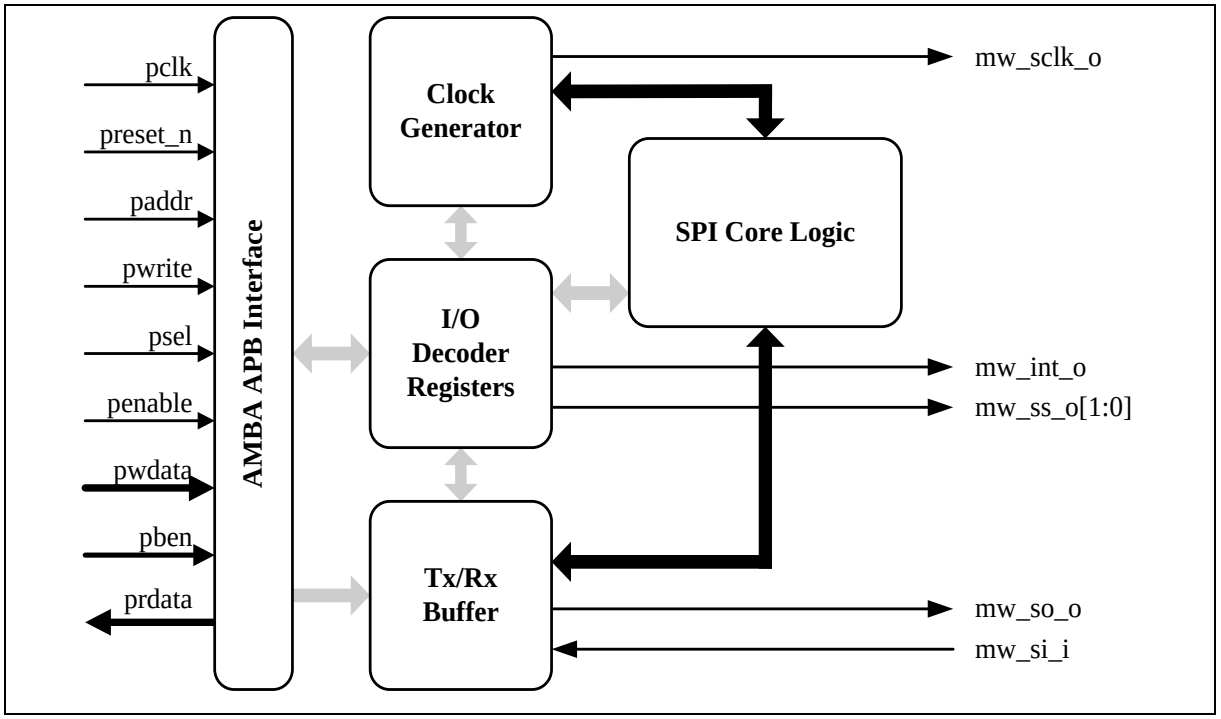


Figure 16.3-1 QSPI Block Diagram

16.4 Functional Description

16.4.1 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.

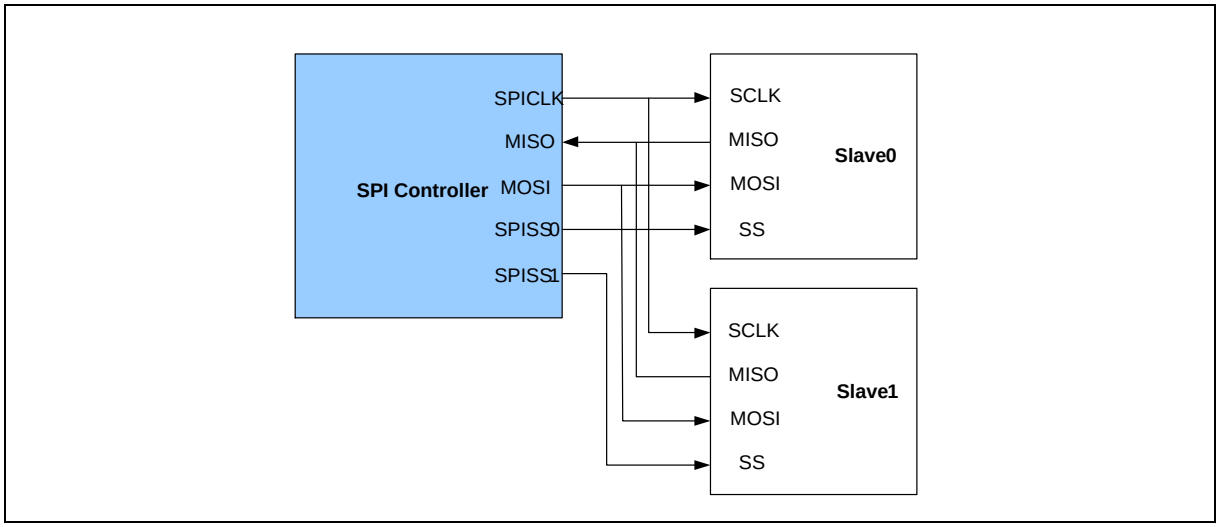


Figure 16.4-1 Slave Selection

Configure SS0(QSPI0_SSCTL[0]) or SS1(QSPI0_SSCTL [1]) can control SS0 or SS1 output data.

```
QSPI0_SSCTL |= 0x1;         //Enable SS0 output pin
QSPI0_SSCTL |= 0x2;         //Enable SS1 output pin
QSPI0_SSCTL |= 0x3;         //Enable SS0 and SS1 output pin at the same time
```

In master mode, user can configure SSACTPOL(QSPI0_SSCTL [2]) bit to let SS signal to active at high or low level. The trigger condition is based on type of slave device.

16.4.2 Automatic Slave Select

In Master mode, if AUTOSS (QSPI0_SSCTL[3]) is set, the slave selection signal will be generated automatically and output to the QSPI0_SS pin according to whether SS0 (QSPI0_SSCTL[0]) and SS1 (QSPI0_SSCTL[1]) is enabled or not. The slave selection signal will be set to active state by the SPI controller when the QSPI data transfer is started by writing to FIFO. It will be set to inactive state when QSPI bus is idle.

```
// Enable automatic slave select function on SS0 pin
QSPI0_SSCTL |= 0x1;              //Enable SS0 output pin
QSPI0_SSCTL |= (0x1 << 3); //Enable automatic slave select function
```

16.4.3 Dual / Quad Mode

QSPI controller supports dual IO transmit when DUALIOEN (QSPI0_CTL[21]) bit is set to 1.

The following figure is dual output mode:

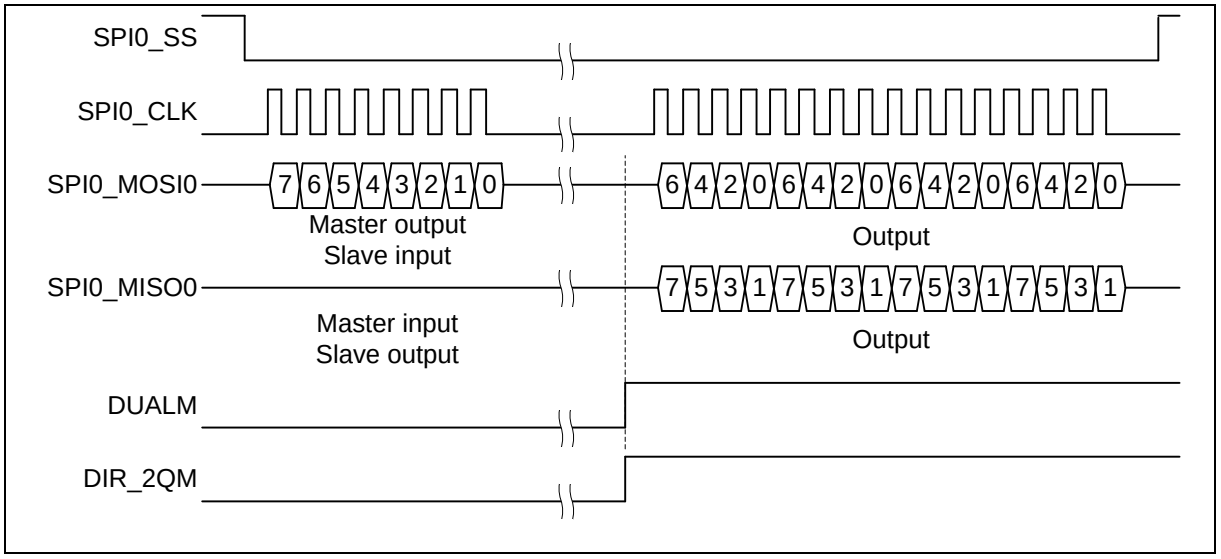


Figure 16.4-2 Dual Output Mode

And the following figure is dual input mode:

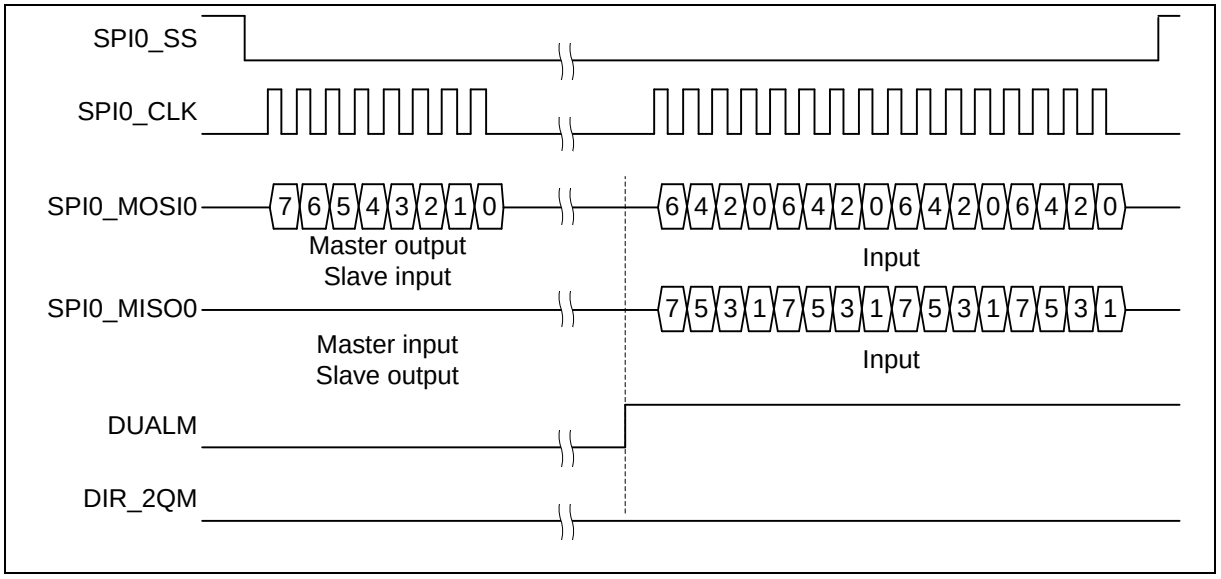


Figure 16.4-3 Dual Input Mode

DATDIR(QSPI0_CTL[20]) is defined as the direction of data transmission. When DATDIR bit is set to 1, SPI controller will output data to external device, otherwise when DATDIR bit is 0, QSPI controller will get the data from external device.

```
//Use dual IO function, MOSI/MISO pin output the data
QSPI0_CTL |= (0x1 << 21);        //Enable dual IO mode
QSPI0_CTL |= (0x1 << 20);        //Direction is output
QSPI0_TX = 0x12;
```

SPI controller supports quad IO mode when QUADIOEN(QSPI0_CTL[22]) bit is set to 1.

The following figure is quad output mode:

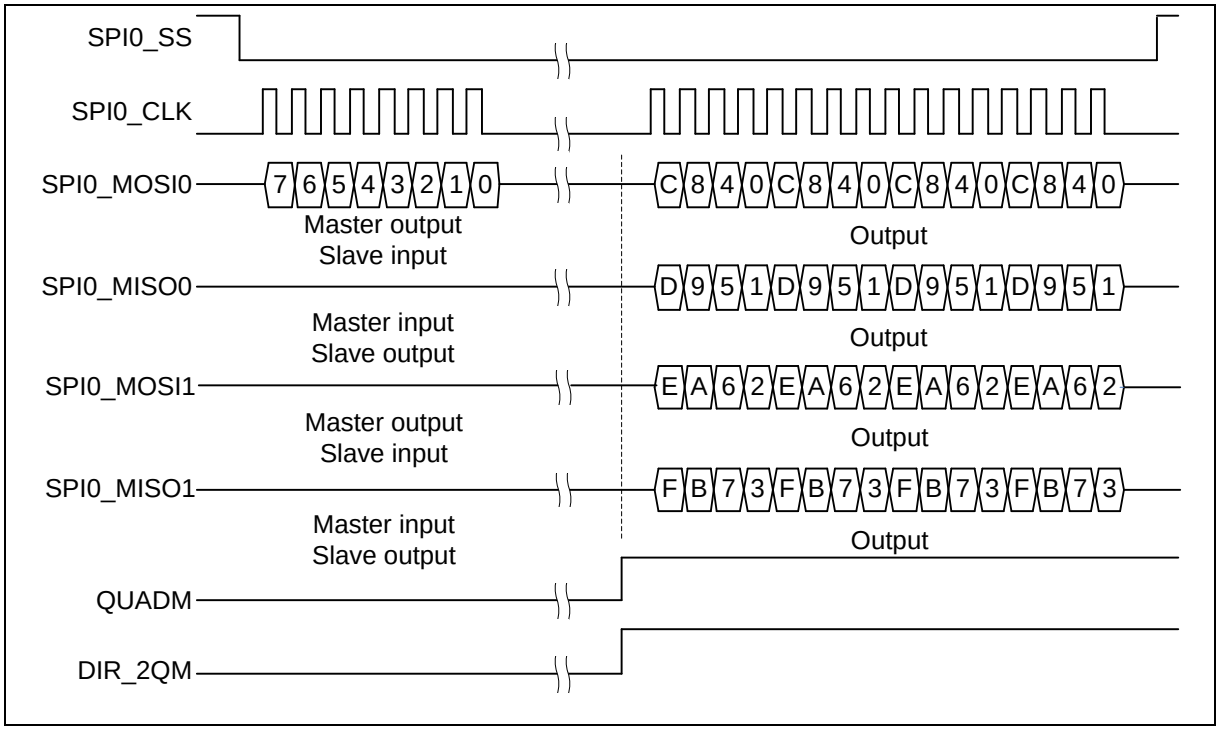


Figure 16.4-4 Quad Output Mode

And the following figure is quad input mode:

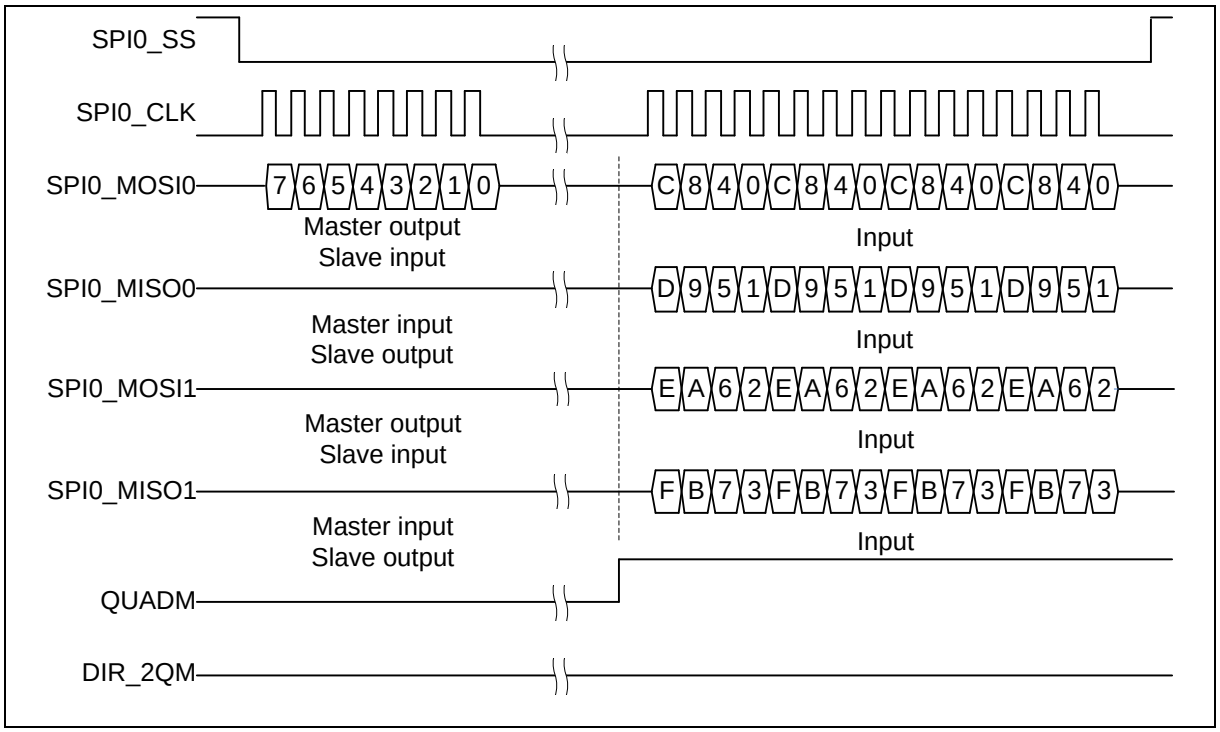


Figure 16.4-5 Quad Input Mode

DATDIR(QSPI0_CTL[20]) is defined as the direction of data transmission. When DATDIR bit is set to 1, SPI controller will output data to external device, otherwise when DATDIR bit is 0, SPI controller will get the data from external device.

```
// Use quad IO function, MOSI/MISO pin output the data
```

```
QSPI0_CTL |= (0x1 << 22);        //Enable quad mode
QSPI0_CTL |= (0x1 << 20);        //Direction is output
QSPI0_TX = 0x12;
```

16.4.4 QSPI Interrupt

The interrupt flag UNITIF(QSPI0_STATUS[1]) bit will be set to 1 after QSPI controller finished transmit or receive. If interrupt enable bit UNITIEN(QSPI0_CTL[17]) is also set to 1 and interrupt will occur. UNIT IF bit can be cleared by writing 1 to itself.

```
QSPI0_CTL |= 0x20000;       //Enable interrupt
QSPI0_TX  |= 0x5A5A5A5A;    //Write data to TX
while(!spi_isr);      //Wait for interrupt
QSPI0_STATUS |= 0x2;        //Clear UNITIF bit
```

If UNITIF bit doesn't be set to 1, user still can poll BUSY(QSPI0_STATUS[0]) bit to check SPI controller finishes transmit or not.

```
QSPI0_TX  |= 0x5A5A5A5A;    //Write data to TX
while(QSPI0_STATUS & 0x1); //Wait for SPI's job is done
```

16.4.5 Slave mode

QSPI controller supports slave mode.

When SLAVE(QSPIx_CTL[18]) is set to 1, QSPI acts as slave mode.

SSACTPOL(QSPIx_SSCTL[2]) defines the active polarity of slave selection signal.

Enable slave 3-wire mode, set SLV3WIRE(QSPIx_SSCTL[4]) to 1, QSPI controller can work with 3-wire interface including QSPIx_CLK, QSPIx_MOSI and QSPIx_MISO.

While slave select interrupt event occurred, SSACTIF(QSPIx_STATUS[2]) is set to 1.

Below is an example that QSPI acts as slave mode, low level active, wait for slave select event and read a data from RX FIFO.

```
unsigned int data;
QSPI0_CTL  |= (1 << 18);    //Set to slave mode
QSPI0_SSCTL &= ~(1 << 2); //low level active
while(!(QSPI0_STATUS & 0x4));    //Wait for slave select event
while(!(QSPI0_STATUS & (1 << 8))); //Wait Rx empty
data = QSPI0_RX; //read received data in FIFO
```

16.4.6 PDMA Transfer function

QSPI controller supports PDMA transfer function.

When TXPDMAEN (QSPIx_PDMACTL[0]) is set to 1, the controller will issue request to PDMA controller to start the PDMA transmission process automatically.

When RXPDMAEN (QSPIx_PDMACTL[1]) is set to 1, the controller will start the PDMA reception process. QSPI controller will issue request to PDMA controller automatically when there is data in the

RX FIFO buffer.

```
QSPIx_PDMACTL |= 1;         //Request PDMA controller to start PDMA TX
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_TX_DMA_CH)); //Wait for PDMA
transfer done


QSPIx_PDMACTL |= 2;         //Request PDMA controller to start PDMA RX
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_RX_DMA_CH)); //Wait for PDMA
transfer done
```

16.4.7 QSPI Programming Example

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

1. Write a divisor into QSPI0_CLKDIV to determine the frequency of serial clock.
2. Write in QSPI0_SSCTL, set AUTOSS = 0, SSACTPOL = 0 and SS0(QSPI0_SSCTL[0]) or SS1(QSPI0_SSCTL[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 QSPI0_TX.
5. When receive (read) data from device:
6. Write in QSPI0_CTL, set RXNEG = 0, TXNEG = 1, DWIDTH = 0x08, LSB = 0, SUSPITV = 0x0 and Write 0 into QSPI0_TX to start the transfer. Wait for interrupt (if UNITIE = 1) or polling the BUSY(QSPI0_STATUS[0]) bit until it turns to 0.
7. Read out the received data from QSPI_RX.
8. Go to step 3 to continue data transfer or set QSPI0_SSCTL[0] or QSPI0_SSCTL[1] to 0 to inactivate the device

16.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
QSPI_BA = 0xB000_6000				
QSPIx_CTL	QSPIx_BA+0x00	R/W	QSPI Control Register	0x0000_0034
QSPIx_CLKDIV	QSPIx_BA+0x04	R/W	QSPI Clock Divider Register	0x0000_0000
QSPIx_SSCTL	QSPIx_BA+0x08	R/W	QSPI Slave Select Control Register	0x0000_0000
QSPIx_PDMACTL	QSPIx_BA+0x0C	R/W	QSPI PDMA Control Register	0x0000_0000
QSPIx_FIFOCTL	QSPIx_BA+0x10	R/W	QSPI FIFO Control Register	0x4400_0000
QSPIx_STATUS	QSPIx_BA+0x14	R/W	QSPI Status Register	0x0005_0110
QSPIx_TX	QSPIx_BA+0x20	W	QSPI Data Transmit Register	0x0000_0000
QSPIx_RX	QSPIx_BA+0x30	R	QSPI Data Receive Register	0x0000_0000

17 SPI

17.1 Overview

The Serial Peripheral Interface (SPI) applies to synchronous serial data communication and allows full duplex transfer. Devices communicate in Master/Slave mode with the 4-wire bi-direction interface. The NUC980 series contains two sets of SPI controllers performing 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. Each SPI controller can be configured as a master or a slave device and supports the PDMA function to access the data buffer.

17.2 Features

- Supports two sets of SPI controllers
- Supports Master or Slave mode operation
- Configurable bit length of a transaction word from 8 to 32-bit
- Provides separate 4-level depth transmit and receive FIFO buffers
- Supports MSB first or LSB first transfer sequence
- Supports Byte Reorder function
- Supports Byte or Word Suspend mode
- Supports PDMA transfer
- Supports one data channel half-duplex transfer
- Supports receive-only mode

17.3 Block Diagram

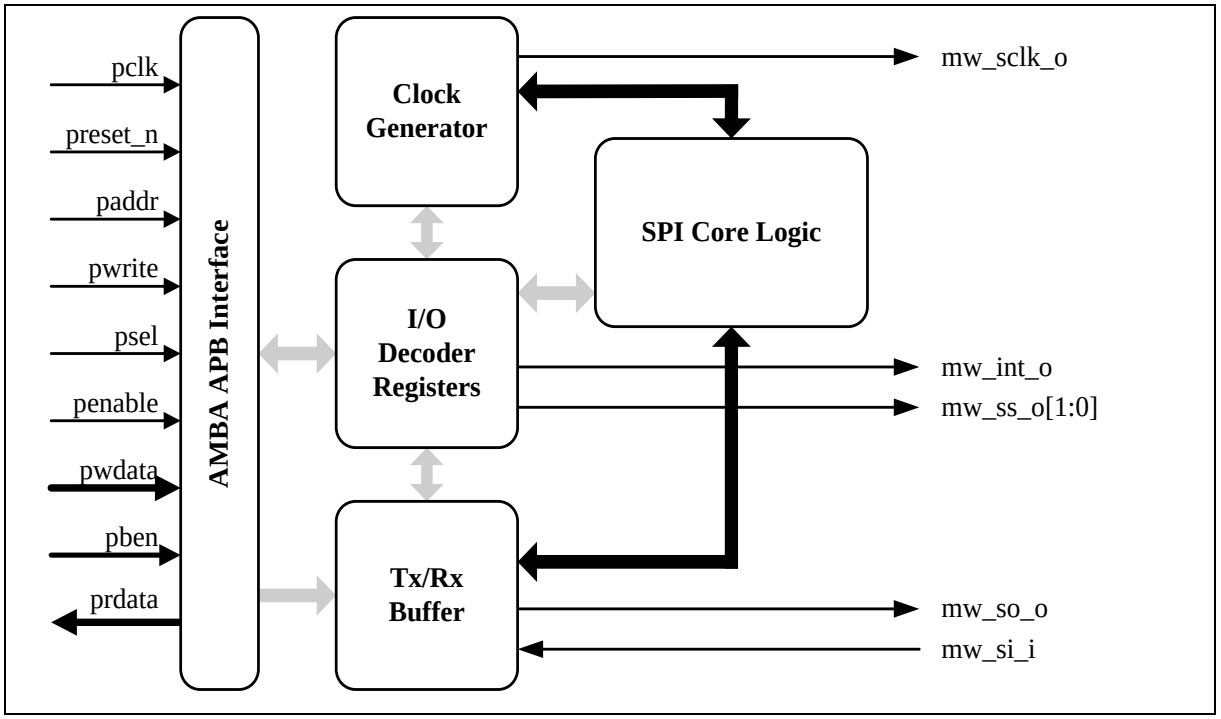


Figure 17.3-1 SPI Block Diagram

17.4 Functional Description

17.4.1 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.

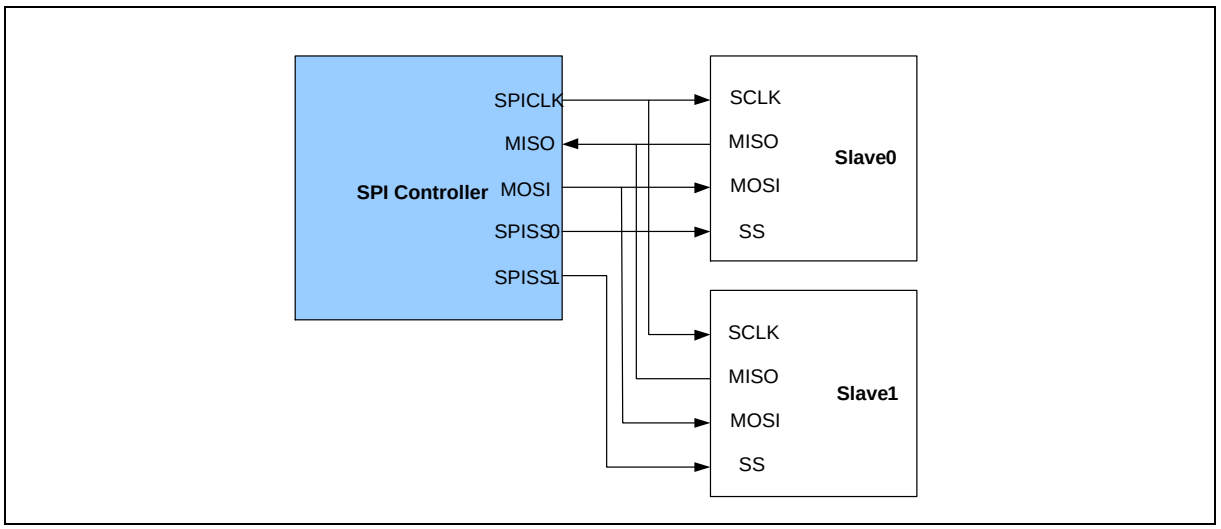


Figure 17.4-1 Slave Selection

Configure SS0(SPIx_SSCTL[0]) or SS1(SPIx_SSCTL [1]) can control SS0 or SS1 output data.

```
SPIx_SSCTL |= 0x1;          //Enable SS0 output pin
```

```
SPIx_SSCTL |= 0x2;          //Enable SS1 output pin
SPIx_SSCTL |= 0x3;          //Enable SS0 and SS1 output pin at the same time
```

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

17.4.2 Automatic Slave Select

In Master mode, if AUTOSS (SPIx_SSCTL[3]) is set, the slave selection signal will be generated automatically and output to the SPIx_SS pin according to whether SS0 (SPIx_SSCTL[0]) and SS1 (SPIx_SSCTL[1]) is enabled or not. The slave selection signal will be set to active state by the SPI controller when the SPI data transfer is started by writing to FIFO. It will be set to inactive state when SPI bus is idle.

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

17.4.3 SPI Interrupt

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

```
SPIx_CTL |= 0x20000;        //Enable interrupt
SPIx_TX  |= 0x5A5A5A5A;     //Write data to TX
while(!spi_isr);      //Wait for interrupt
SPIx_STATUS |= 0x2;         //Clear UNITIF bit
```

If UNITIF bit doesn't be set to 1, user still can poll BUSY(SPIx_STATUS[0]) bit to check SPI controller finishes transmit or not.

```
SPIx_TX  |= 0x5A5A5A5A;     //Write data to TX
while(SPIx_STATUS & 0x1);   //Wait for SPI's job is done
```

17.4.4 Slave mode

SPI controller supports slave mode.

When SLAVE(SPIx_CTL[18]) is set to 1, SPI acts as slave mode.

SSACTPOL(SPIx_SSCTL[2]) defines the active polarity of slave selection signal.

While slave select interrupt event occurred, SSACTIF(SPIx_STATUS[2]) is set to 1.

Below is an example that SPI acts as slave mode, low level active, wait for slave select event and read a data from RX FIFO.

```
unsigned int data;
SPIx_CTL  |= (1 << 18);     //Set to slave mode
SPIx_SSCTL &= ~(1 << 2); //low level active
while(!(SPIx_STATUS & 0x4));     //Wait for slave select event
while(!(SPIx_STATUS & (1 << 8))); //Wait Rx empty
data = SPIx_RX; //read received data in FIFO
```

17.4.5 PDMA Transfer function

SPI controller supports PDMA transfer function.

When TXPDMAEN (SPIx_PDMACTL[0]) is set to 1, the controller will issue request to PDMA controller to start the PDMA transmission process automatically.

When RXPDMAEN (SPIx_PDMACTL[1]) is set to 1, the controller will start the PDMA reception process. SPI controller will issue request to PDMA controller automatically when there is data in the RX FIFO buffer.

```
SPIx_PDMACTL |= 1;          //Request PDMA controller to start PDMA TX
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_TX_DMA_CH)); //Wait for PDMA
transfer done


SPIx_PDMACTL |= 2;          //Request PDMA controller to start PDMA RX
while (PDMA_GET_TD_STS(PDMA0) & (1 << SPI_MASTER_RX_DMA_CH)); //Wait for PDMA
transfer done
```

17.4.6 SPI Programming Example

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

1. Write a divisor into SPIx_CLKDIV to determine the frequency of serial clock.
2. Write in SPIx_SSCTL, set AUTOSS = 0, SSACTPOL = 0 and SS0(SPIx_SSCTL[0]) or SS1(SPIx_SSCTL[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 SPIx_TX.
5. When receive (read) data from device:
6. Write in SPIx_CTL, set RXNEG = 0, TXNEG = 1, DWIDTH = 0x08, LSB = 0, SUSPITV = 0x0 and Write 0 into SPIx_TX to start the transfer.
7. Wait for interrupt (if UNITIE = 1) or polling the BUSY(SPIx_STATUS[0]) bit until it turns to 0.
8. Read out the received data from SPI_RX.
9. Go to step 3 to continue data transfer or set SPIx_SSCTL[0] or SPIx_SSCTL[1] to 0 to inactivate the device

17.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
SPI_BA = 0xB000_6000 + (0x0000_1000 * x) **x = 0,1**				
SPIx_CTL	SPIx_BA+0x00	R/W	SPI Control Register	0x0000_0034
SPIx_CLKDIV	SPIx_BA+0x04	R/W	SPI Clock Divider Register	0x0000_0000
SPIx_SSCTL	SPIx_BA+0x08	R/W	SPI Slave Select Control Register	0x0000_0000
SPIx_PDMACTL	SPIx_BA+0x0C	R/W	SPI PDMA Control Register	0x0000_0000
SPIx_FIFOCTL	SPIx_BA+0x10	R/W	SPI FIFO Control Register	0x2200_0000
SPIx_STATUS	SPIx_BA+0x14	R/W	SPI Status Register	0x0005_0110
SPIx_TX	SPIx_BA+0x20	W	SPI Data Transmit Register	0x0000_0000
SPIx_RX	SPIx_BA+0x30	R	SPI Data Receive Register	0x0000_0000

18 I²S CONTROLLER (I²S)

18.1 Overview

The I²S controller consists of I²S and PCM protocols to interface with external audio CODEC. The I²S 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.

18.2 Features

- Support I²S 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
  - 8, 16, 20, 24-bit data precision
  - Master mode
- Use DMA to playback and record data, with interrupt

18.3 Block Diagram

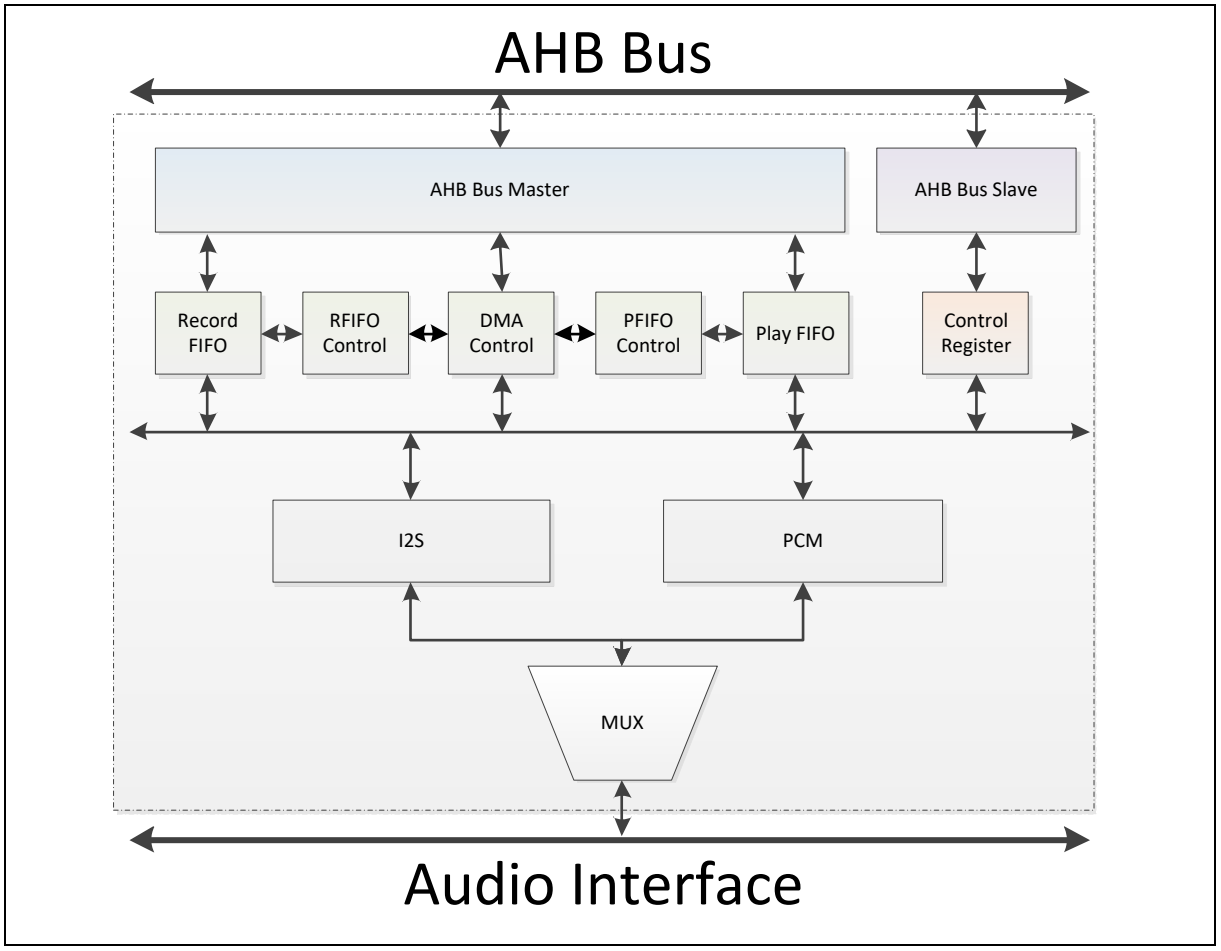


Figure 18.3-1 I²S Controller Block Diagram

18.4 Functional Description

18.4.1 I²S Master/Slave Mode

To use slave mode, user can set SLAVE(I2S_CON[20]) bit to 1 otherwise set 0 to be as master mode.

Note that slave only can be chosen when use I²S interface and only use master mode if PCM interface is used.

Master mode connection between controller and audio codec:

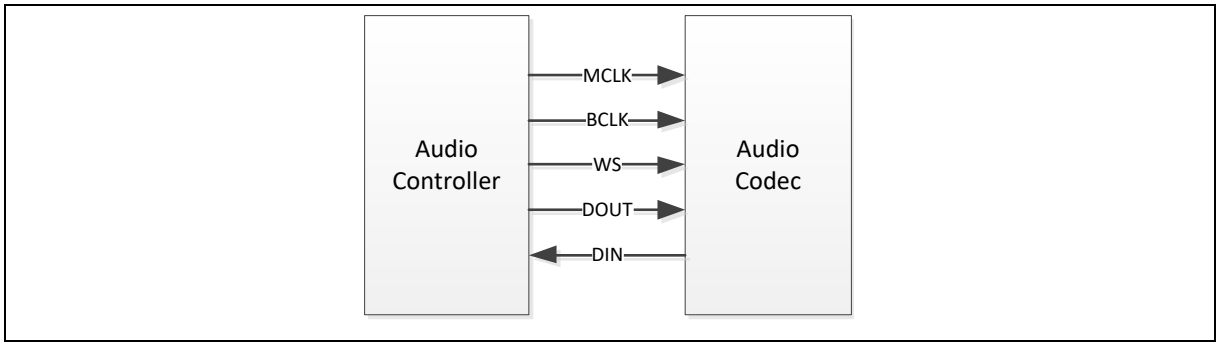


Figure 18.4-1 I²S Master Mode Connection

Slave mode connection between controller and audio codec:

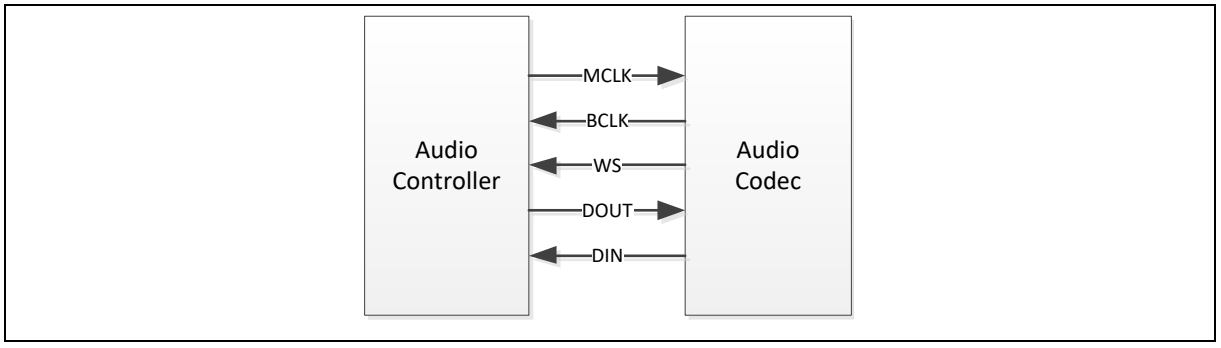


Figure 18.4-2 I²S Slave Mode Selection

18.4.2 I²S Source Clock Configuration

Software can choose APLL, UPLL or external crystal as source clock of I²S by configuring I2S_S(CLK_DIVCTL1[20:19])

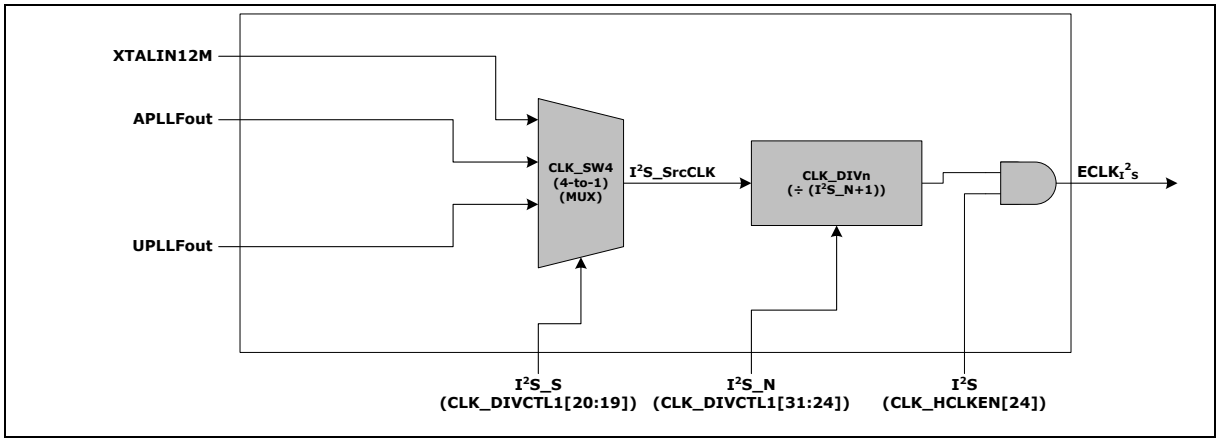


Figure 18.4-3 I²S Clock Configuration

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

18.4.3 I²S Calculation and Configuration of Clock

The clocks in I²S need to be configured are MCLK and BCLK. Only MCLK needs to be configured when using I²S slave mode.

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

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:

MCLK = 256 * 48000 =11288000 Hz = 12.288 MHz

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

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

18.4.4 DMA

I²S 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 use 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_GLBCON[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_CNTER_EN(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 I²S 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

18.4.5 Sequence of DMA Data

When use I²S 18, 20, 24-bits, each data stored in DMA buffer will all use 32-bit width.

Take I²S 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
…	…

Table 18.4-1 Stereo Mode DMA Buffer Layout

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
…	…

Table 18.4-2 Mono Mode DMA Buffer Layout

18.4.6 Interface Selection

Software can choose I²S or PCM interface by setting BLOCK_EN(I2S_GLBCON[0]) bit.

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

18.4.7 PCM Interface

The following figure is PCM timing wave form,

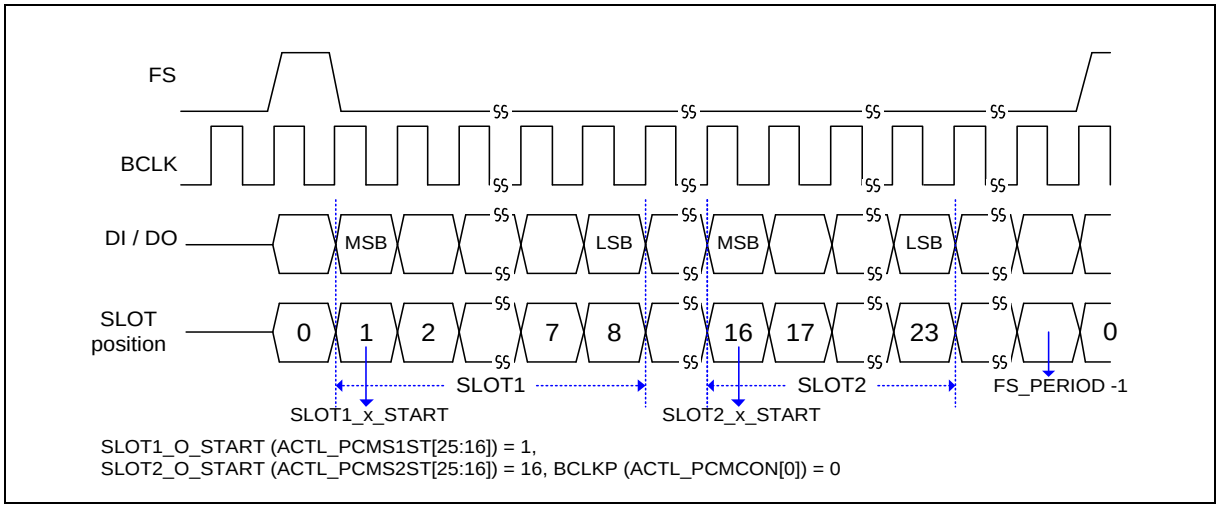


Figure 18.4-4 I²S PCM Interface

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

```
//BCLK=24.576 MHz/48 = 512k
```

```
I2S_PCMCON = I2S_PCMCON | (23<<8));

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

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

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

18.4.8 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 I²S left channel or PCM slot1. The data in second DMA address is I²S right channel or PCM slot2.

To reach the target mentions 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 I²S 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
…	…

Table 18.4-3 Stereo Mode Data Split Left Channel DMA Buffer

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
…	…

Table 18.4-4 Stereo Mode Data Split Right Channel DMA Buffer

18.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
I²S Base Address: **I2S_BA = 0xB000_2000**				
I2S_GLBCON	I2S_BA+0x000	R/W	I²S Global Control Register	0x0000_0000
I2S_RESET	I2S_BA+0x004	R/W	I²S Sub Block Reset Control Register	0x0000_0000
I2S_RDESB	I2S_BA+0x008	R/W	I²S Record DMA Destination Base Address Register	0x0000_0000
I2S_RDES_LENGTH	I2S_BA+0x00C	R/W	I²S Record DMA Destination Length Register	0x0000_0000
I2S_RDESC	I2S_BA+0x010	R	I²S Record DMA Destination Current Address Register	0x0000_0000
I2S_PDESB	I2S_BA+0x014	R/W	I²S Play DMA Destination Base Address Register	0x0000_0000
I2S_PDES_LENGTH	I2S_BA+0x018	R/W	I²S Play DMA Destination Length Register	0x0000_0000
I2S_PDESC	I2S_BA+0x01C	R	I²S Play DMA Destination Current Address Register	0x0000_0000
I2S_RSR	I2S_BA+0x020	R/W	I²S Record Status Register	0x0000_0000
I2S_PSR	I2S_BA+0x024	R/W	I²S Play Status Register	0x0000_0000
I2S_CON	I2S_BA+0x028	R/W	I²S Control Register	0x0000_0000
I2S_COUNTER	I2S_BA+0x02C	R/W	I²S Play DMA Down Counter Register	0xFFFF_FFFF
I2S_PCMCON	I2S_BA+0x030	R/W	I²S PCM Mode Control Register	0x0000_0000
I2S_PCMS1ST	I2S_BA+0x034	R/W	I²S PCM Mode Slot 1 Start Register	0x0000_0000
I2S_PCMS2ST	I2S_BA+0x038	R/W	I²S PCM Mode Slot 2 Start Register	0x0000_0000
I2S_RDESB2	I2S_BA+0x040	R/W	I²S Record DMA Destination Base Address 2 Register	0x0000_0000
I2S_PDESB2	I2S_BA+0x044	R/W	I²S Play DMA Destination Base Address 2 Register	0x0000_0000

19 ETHERNET MAC CONTROLLER (EMAC)

19.1 Overview

The NUC980 provides 2 Ethernet MAC Controllers (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.

19.2 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

19.3 Block Diagram

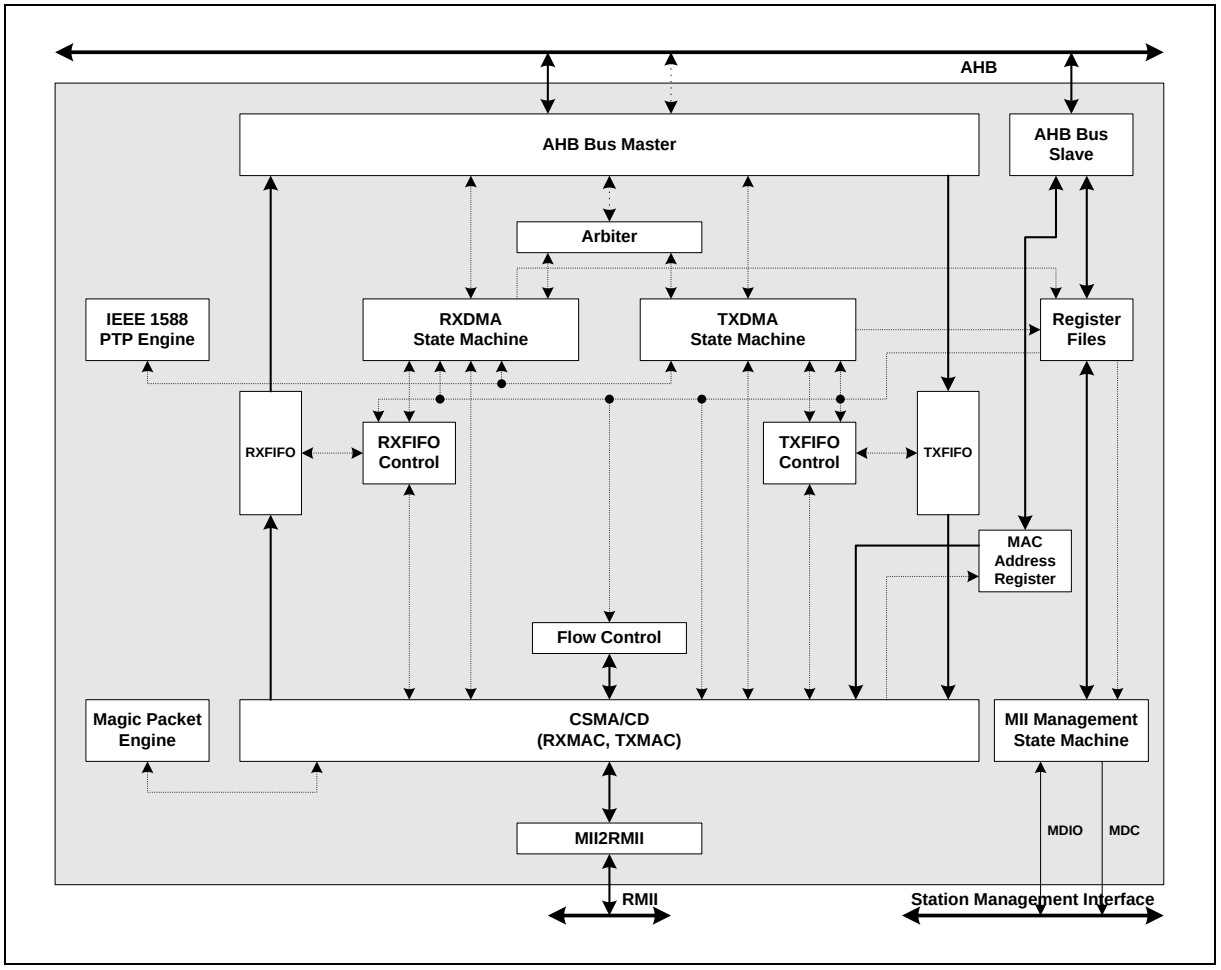


Figure 19.3-1 EMAC Block Diagram

19.4 Functional Description

19.4.1 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_MIIDA[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 FDUP (EMAC_MCMDR[18]) accordingly. EMAC and PHY must have the same operating mode setting to transmit and receive packet correctly.

19.4.2 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. NUC980 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, 0xFFFF0000 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 NUC980 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.

19.4.3 Control Frame

IEEE 802.3 defines control frame used for flow control. NUC980 supports transmit and receive pause frame for flow control. NUC980 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.

19.4.4 Wake on Lan (WoL)

NUC980 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 contains6 continuous 0xFFs anyplace in the packet follow 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.

19.4.5 Packet Receive

EMAC use a link-list structure named as descriptors to receive Ethernet packets. Driver needs to prepare RX descriptors in advance before enabled receive function. After CAM decides a packet needs to be received, packet will be received to a memory space describes 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 occupied 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 switch 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.

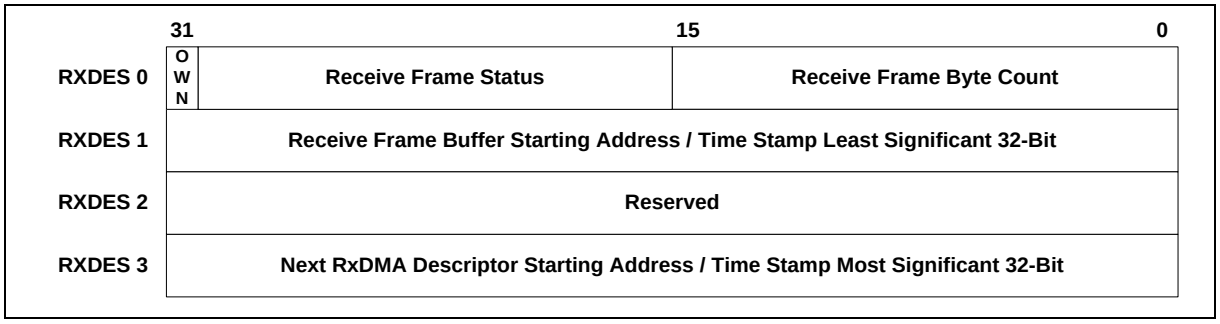


Figure 19.4-1 EMAC Rx Descriptor

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_RXDLSA 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.

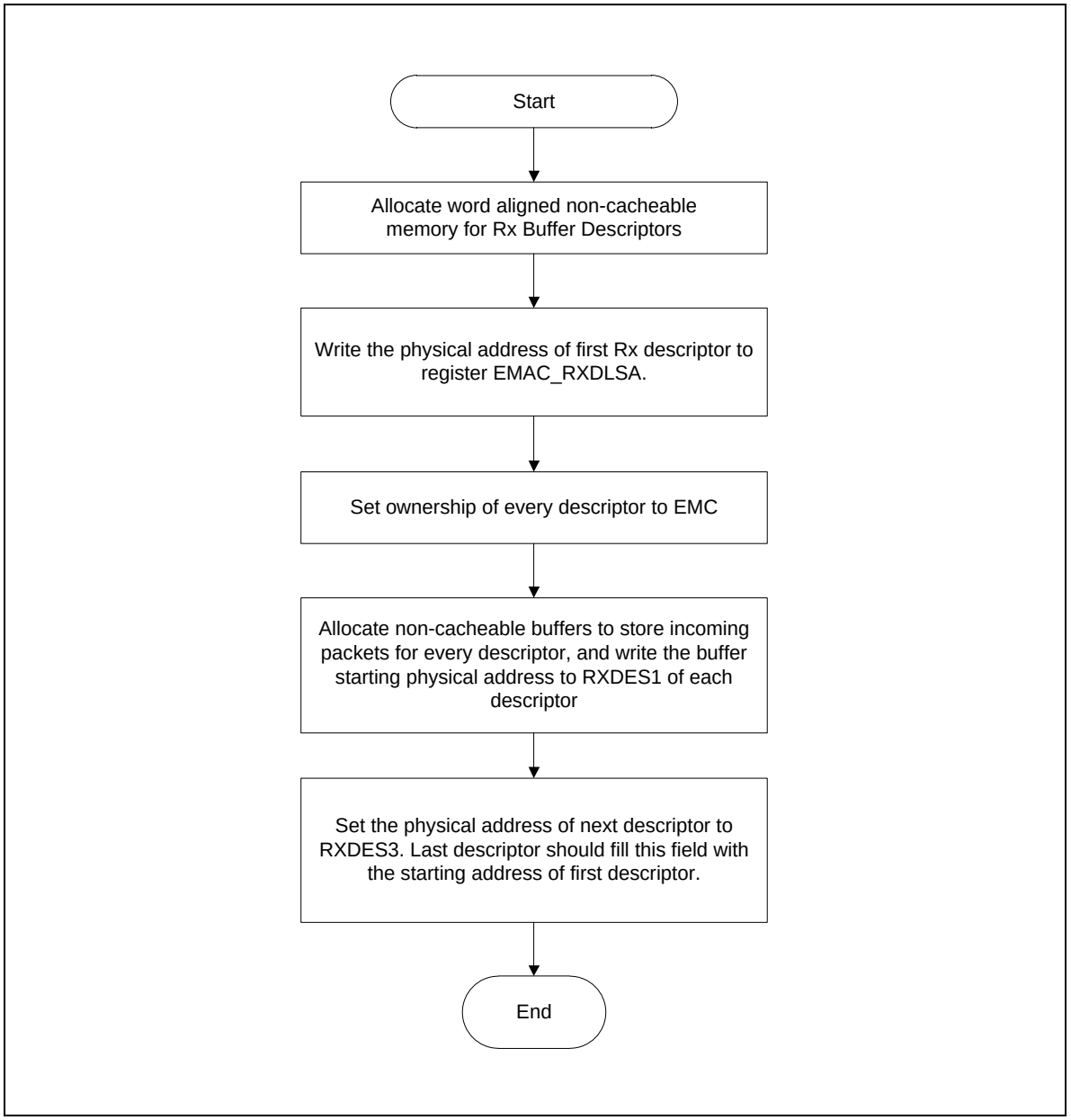


Figure 19.4-2 Rx Descriptor Init Flow

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.

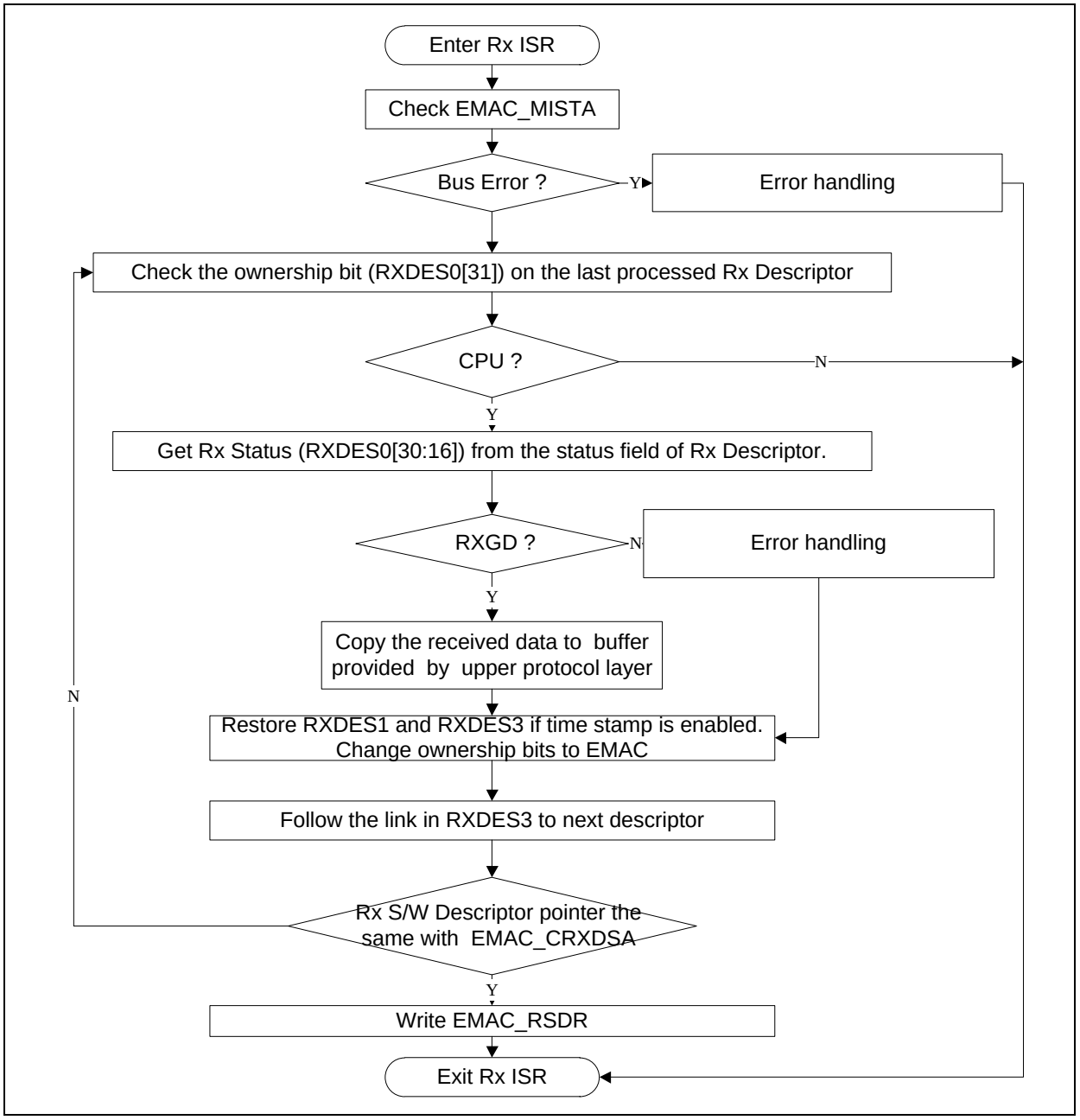


Figure 19.4-3 EMAC Rx ISR Flow

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;
}
```

19.4.6 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 it 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	OWN	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					

Figure 19.4-4 EMAC Tx Descriptor

If network timestamp enabled, TXDES1 and TXDES3 will be used to store packet transmit 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 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.

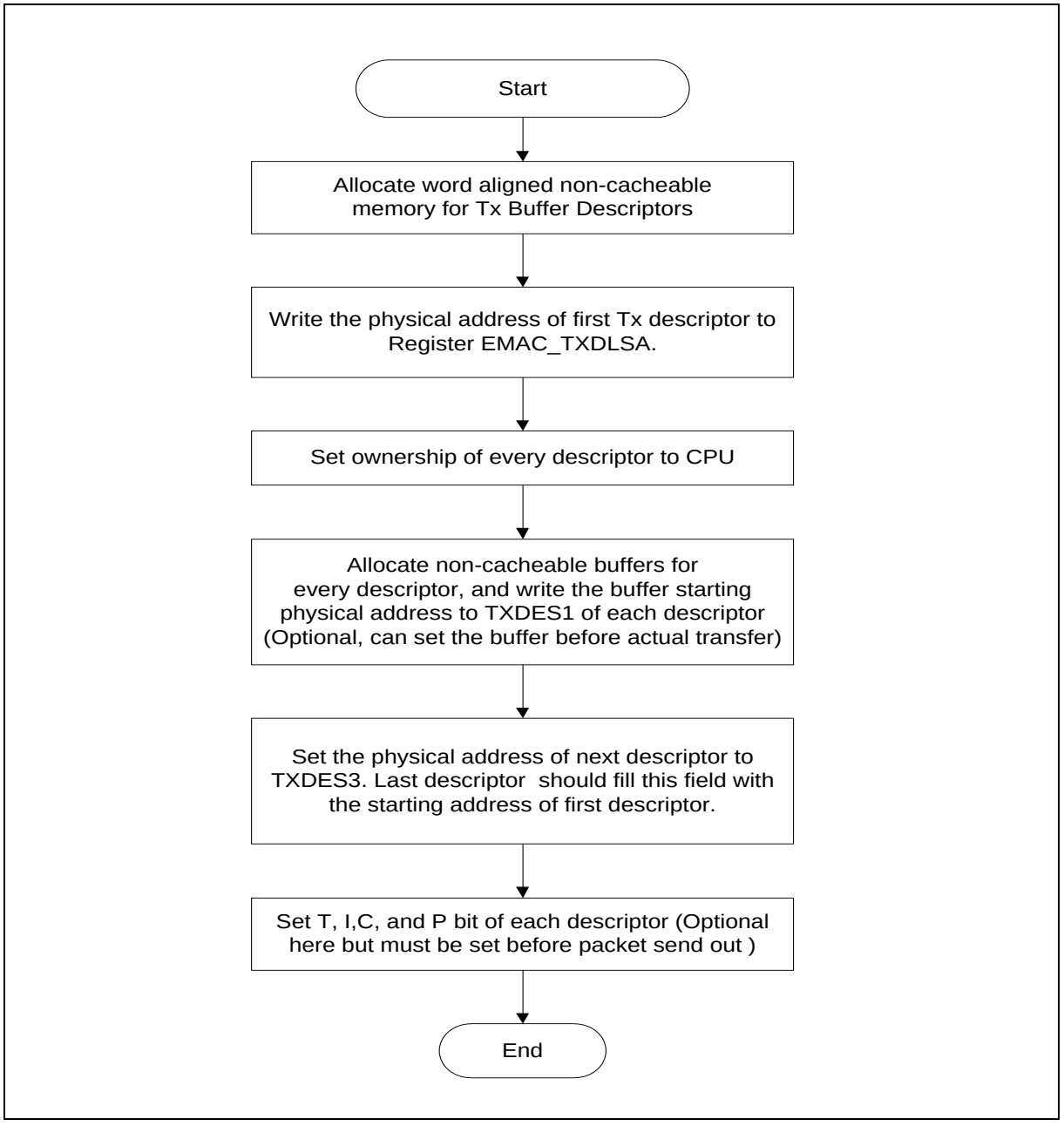


Figure 19.4-5 Tx Descriptor Init Flow

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 transmitting procedure with time stamping enabled.

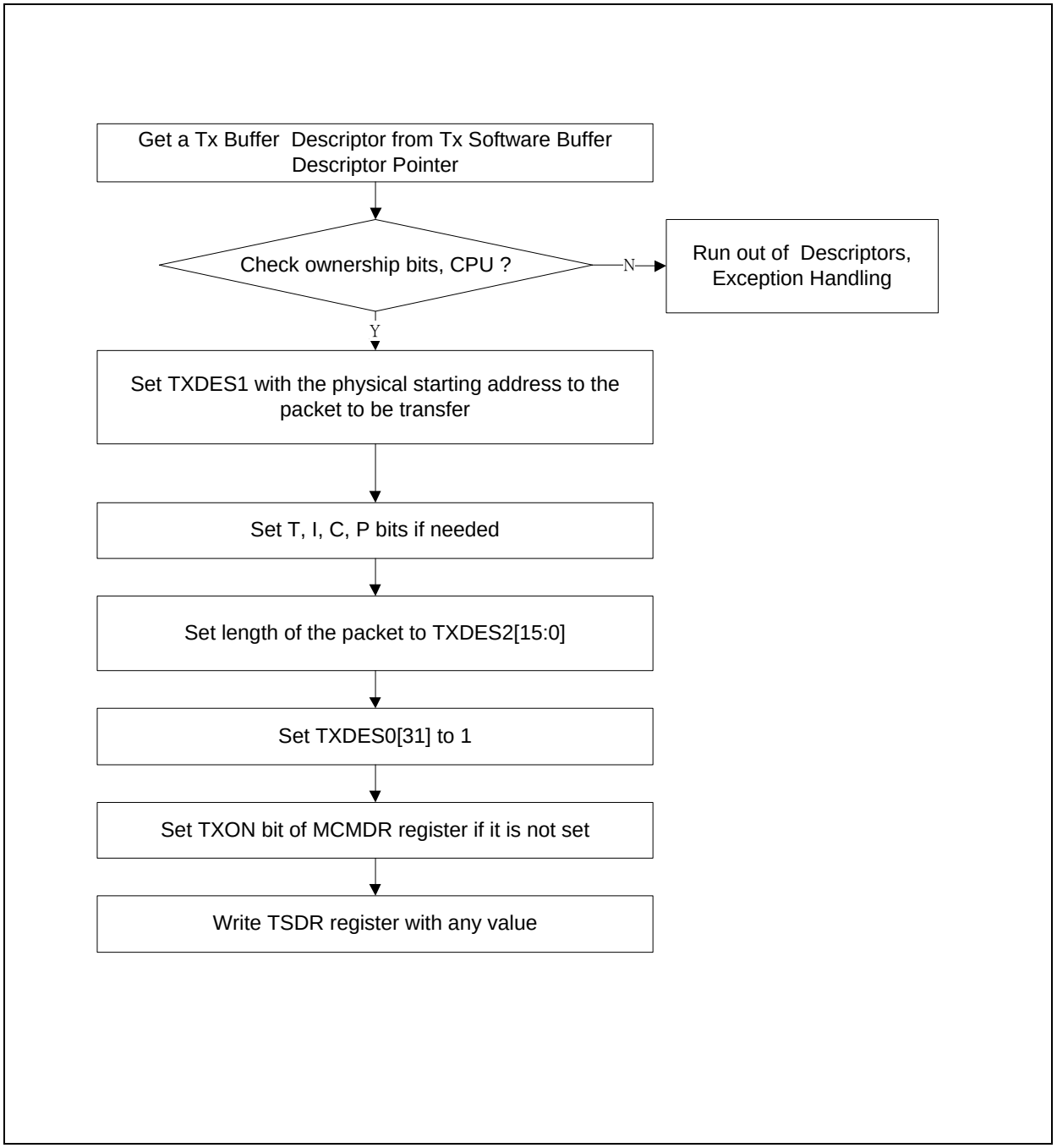


Figure 19.4-6 EMAC Packet Transmit

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;

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

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

    Return 0;
}
```

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 wills 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.

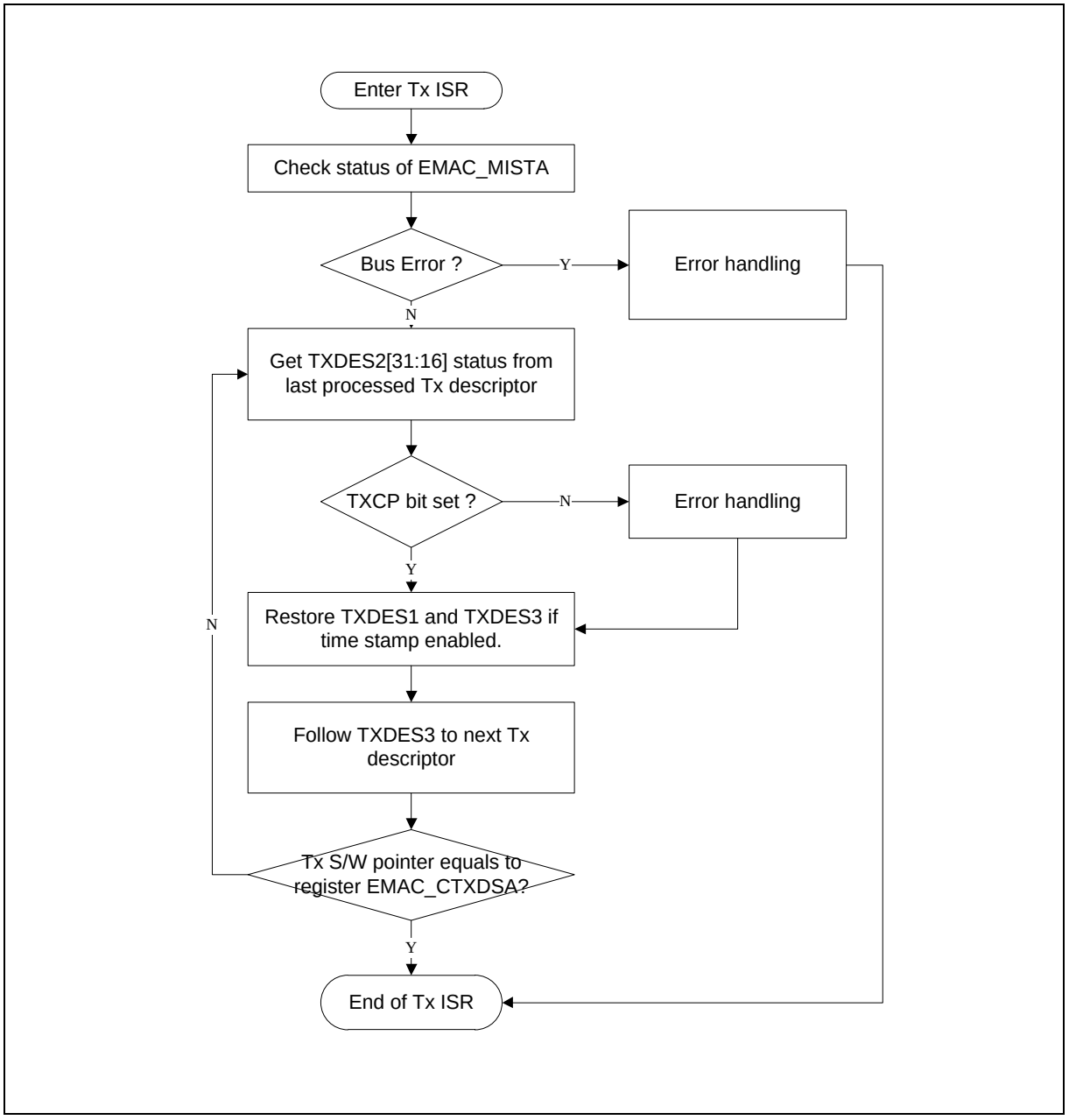


Figure 19.4-7 EMAC Tx ISR Flow

Below is an example of interrupt handler 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 & 0xffff0000;

      last_tx_desc = EMAC_CTXDSA;
      desc = current_tx_desc;

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

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

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

}
```

19.4.7 **Network Timing**

To support more accurate IEEE1588 network timing on NUC980, 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 150 MHz 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.

NUC980 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 150 MHz as example to calculate the relative registers setting here. Since the clock rate is 150 MHz, sub-second file 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 ~= 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 150 MHz.

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 find update mode is more accurate than cores update mode.

Here use EMAC clock frequency 150 MHz 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 ~= 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 ~= 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 150 MHz 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.

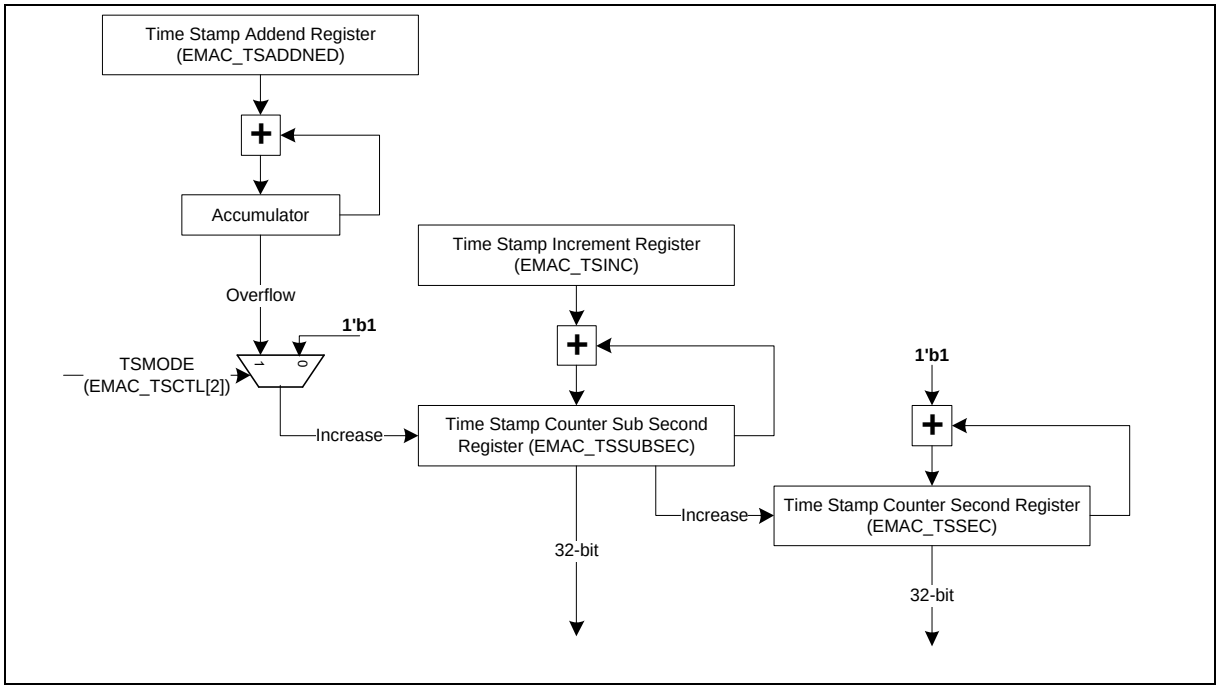


Figure 19.4-8 EMAC Time Stamp Calculation

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 = 100000000011 * 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 NUC980'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 store in tow 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 an 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.

19.4.8 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.

Table 19.4-1 EMAC Erro Handling

19.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
EMAC0_BA = 0xB001_2000 **EMAC1_BA = 0xB002_2000**				
EMAC_CAMCMR	EMAC_BA+0x000	R/W	CAM Command Register	0x0000_0000
EMAC_CAMEN	EMAC_BA+0x004	R/W	CAM Enable Register	0x0000_0000
EMAC_CAM0M	EMAC_BA+0x008	R/W	CAM0 Most Significant Word Register	0x0000_0000
EMAC_CAM0L	EMAC_BA+0x00C	R/W	CAM0 Least Significant Word Register	0x0000_0000
EMAC_CAM1M	EMAC_BA+0x010	R/W	CAM1 Most Significant Word Register	0x0000_0000
EMAC_CAM1L	EMAC_BA+0x014	R/W	CAM1 Least Significant Word Register	0x0000_0000
EMAC_CAM2M	EMAC_BA+0x018	R/W	CAM2 Most Significant Word Register	0x0000_0000
EMAC_CAM2L	EMAC_BA+0x01C	R/W	CAM2 Least Significant Word Register	0x0000_0000
EMAC_CAM3M	EMAC_BA+0x020	R/W	CAM3 Most Significant Word Register	0x0000_0000
EMAC_CAM3L	EMAC_BA+0x024	R/W	CAM3 Least Significant Word Register	0x0000_0000
EMAC_CAM4M	EMAC_BA+0x028	R/W	CAM4 Most Significant Word Register	0x0000_0000
EMAC_CAM4L	EMAC_BA+0x02C	R/W	CAM4 Least Significant Word Register	0x0000_0000
EMAC_CAM5M	EMAC_BA+0x030	R/W	CAM5 Most Significant Word Register	0x0000_0000
EMAC_CAM5L	EMAC_BA+0x034	R/W	CAM5 Least Significant Word Register	0x0000_0000
EMAC_CAM6M	EMAC_BA+0x038	R/W	CAM6 Most Significant Word Register	0x0000_0000
EMAC_CAM6L	EMAC_BA+0x03C	R/W	CAM6 Least Significant Word Register	0x0000_0000
EMAC_CAM7M	EMAC_BA+0x040	R/W	CAM7 Most Significant Word Register	0x0000_0000
EMAC_CAM7L	EMAC_BA+0x044	R/W	CAM7 Least Significant Word Register	0x0000_0000
EMAC_CAM8M	EMAC_BA+0x048	R/W	CAM8 Most Significant Word Register	0x0000_0000
EMAC_CAM8L	EMAC_BA+0x04C	R/W	CAM8 Least Significant Word Register	0x0000_0000
EMAC_CAM9M	EMAC_BA+0x050	R/W	CAM9 Most Significant Word Register	0x0000_0000
EMAC_CAM9L	EMAC_BA+0x054	R/W	CAM9 Least Significant Word Register	0x0000_0000
EMAC_CAM10M	EMAC_BA+0x058	R/W	CAM10 Most Significant Word Register	0x0000_0000
EMAC_CAM10L	EMAC_BA+0x05C	R/W	CAM10 Least Significant Word Register	0x0000_0000
EMAC_CAM11M	EMAC_BA+0x060	R/W	CAM11 Most Significant Word Register	0x0000_0000
EMAC_CAM11L	EMAC_BA+0x064	R/W	CAM11 Least Significant Word Register	0x0000_0000
EMAC_CAM12M	EMAC_BA+0x068	R/W	CAM12 Most Significant Word Register	0x0000_0000

EMAC_CAM12L	EMAC_BA+0x06C	R/W	CAM12 Least Significant Word Register	0x0000_0000
EMAC_CAM13M	EMAC_BA+0x070	R/W	CAM13 Most Significant Word Register	0x0000_0000
EMAC_CAM13L	EMAC_BA+0x074	R/W	CAM13 Least Significant Word Register	0x0000_0000
EMAC_CAM14M	EMAC_BA+0x078	R/W	CAM14 Most Significant Word Register	0x0000_0000
EMAC_CAM14L	EMAC_BA+0x07C	R/W	CAM14 Least Significant Word Register	0x0000_0000
EMAC_CAM15M	EMAC_BA+0x080	R/W	CAM15 Most Significant Word Register	0x0000_0000
EMAC_CAM15L	EMAC_BA+0x084	R/W	CAM15 Least Significant Word Register	0x0000_0000
EMAC_TXDLSA	EMAC_BA+0x088	R/W	Transmit Descriptor Link List Start Address Register	0xFFFF_FFFC
EMAC_RXDLSA	EMAC_BA+0x08C	R/W	Receive Descriptor Link List Start Address Register	0xFFFF_FFFC
EMAC_MCMDR	EMAC_BA+0x090	R/W	MAC Command Register	0x0040_0000
EMAC_MIID	EMAC_BA+0x094	R/W	MII Management Data Register	0x0000_0000
EMAC_MIIDA	EMAC_BA+0x098	R/W	MII Management Control and Address Register	0x0000_0000
EMAC_FFTCR	EMAC_BA+0x09C	R/W	FIFO Threshold Control Register	0x0000_0000
EMAC_TSDR	EMAC_BA+0x0A0	W	Transmit Start Demand Register	Undefined
EMAC_RSDR	EMAC_BA+0x0A4	W	Receive Start Demand Register	Undefined
EMAC_DMARFC	EMAC_BA+0x0A8	R/W	Maximum Receive Frame Control Register	0x0000_0800
EMAC_MIEN	EMAC_BA+0x0AC	R/W	MAC Interrupt Enable Register	0x0000_0000
EMAC_MISTA	EMAC_BA+0x0B0	R/W	MAC Interrupt Status Register	0x0000_0000
EMAC_MGSTA	EMAC_BA+0x0B4	R/W	MAC General Status Register	0x0000_0000
EMAC_MPCNT	EMAC_BA+0x0B8	R/W	Missed Packet Count Register	0x0000_7FFF
EMAC_MRPC	EMAC_BA+0x0BC	R	MAC Receive Pause Count Register	0x0000_0000
EMAC_DMARFS	EMAC_BA+0x0C8	R/W	DMA Receive Frame Status Register	0x0000_0000
EMAC_CTXDSA	EMAC_BA+0x0CC	R	Current Transmit Descriptor Start Address Reg.	0x0000_0000
EMAC_CTXBSA	EMAC_BA+0x0D0	R	Current Transmit Buffer Start Address Register	0x0000_0000
EMAC_CRXDSA	EMAC_BA+0x0D4	R	Current Receive Descriptor Start Address Reg.	0x0000_0000
EMAC_CRXBSA	EMAC_BA+0x0D8	R	Current Receive Buffer Start Address Register	0x0000_0000
EMAC_TSCTL	EMAC_BA+0x100	R/W	Time Stamp Control Register	0x0000_0000
EMAC_TSSEC	EMAC_BA+0x110	R	Time Stamp Counter Second Register	0x0000_0000
EMAC_TSSUBSEC	EMAC_BA+0x114	R	Time Stamp Counter Sub Second Register	0x0000_0000
EMAC_TSINC	EMAC_BA+0x118	R/W	Time Stamp Increment Register	0x0000_0000
EMAC_TSADDEND	EMAC_BA+0x11C	R/W	Time Stamp Addend Register	0x0000_0000

EMAC_UPDSEC	EMAC_BA+0x120	R/W	Time Stamp Update Second Register	0x0000_0000
EMAC_UPDSUBSEC	EMAC_BA+0x124	R/W	Time Stamp Update Sub Second Register	0x0000_0000
EMAC_ALMSEC	EMAC_BA+0x128	R/W	Time Stamp Alarm Second Register	0x0000_0000
EMAC_ALMSUBSEC	EMAC_BA+0x12C	R/W	Time Stamp Alarm Sub Second Register	0x0000_0000

20 USB 2.0 DEVICE CONTROLLER (USBD)

20.1 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.

20.2 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

20.3 Block Diagram

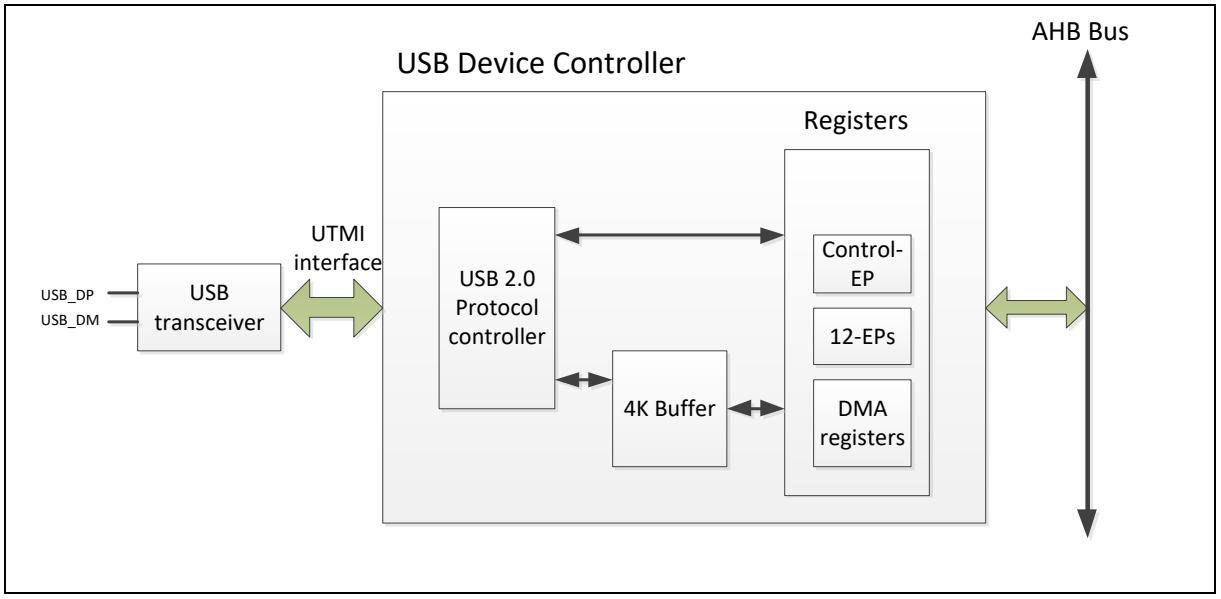


Figure 20.3-1 USB Device Controller Block Diagram

20.4 Functional Description

The USB device controller is compliant with USB 2.0 specification. User can simulate 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.

20.4.1 Initialization

USB device controller initialize, please follow the steps below:

1. Set multiple function pin GPE11. Fill 0x1 to SYS_GPE_MFPL register GPE11 bit.
2. Set CLK_HCLKEN register USBD bit.
3. Set HSUSBD_PHYCTL register PHYEN bit to enable USB PHY.
4. Fill 0x8 to HSUSBD_EPAMPS register. Polling HSUSBD_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 HSUSBD_EPARSPCTL register MODE bit to 0 to select auto-validation mode.
    2) Fill 512 to HSUSBD_EPAMPS register. It means the maximum packet size is 512 bytes
    3) Set HSUSBD_EPACFG register EPNUM bit to 1, EPDIR bit to 1, EPTYPE bit to 01b and EPEN bit to 1
    4) Fill 0x200 to HSUSBD_EPABUFSTART register. 0x3FF to HSUSBD_EPABUFEND register. It means this endpoint FIFO length is 512 bytes
6. Configure endpoint B to Bulk-Out type, endpoint number 2
    1) Set HSUSBD_EPBINTEN register RXPKIEN bit to enable data receive interrupt.
    2) Set HSUSBD_EPBRSPCTL register MODE bit to 0 to select auto-validation mode
    3) Fill 512 to HSUSBD_EPBMPS register. It means the maximum packet size is 512 bytes
    4) Set HSUSBD_EPBCFG register EPNUM bit to 2, EPDIR bit to 0, EPTYPE bit to 01b and EPEN bit to 1
    5) Fill 0x400 to HSUSBD_EPBBUFSTART register. 0x5FF to HSUSBD_EPBBUFEND register. It means this endpoint FIFO length is 512 bytes
7. Set HSUSBD_GINTEN register USBIEN, CEPIEN, EPAIEN and EPBIEN bit to enable USB bus, control endpoint, endpoint A and endpoint B interrupt
8. Set HSUSBD_BUSINTEN register RSTIEN, RESUMEIEN, DMADONEIEN and VBUSDETIEN bit to enable USB reset, resume, DMA complete and floating detect interrupt
9. Set HSUSBD_OPER register HISPDEN bit to enable high speed mode

10. Clear HSUSBD_FADDR register
11. Configure control endpoint (EP0)
    1) Fill 0x0 to HSUSBD_CEPBUFSTART register. 0x7F to HSUSBD_CEPBUFEND register. It means this endpoint FIFO length is 128 bytes.
    2) Set HSUSBD_CEPINTEN register SETUPPKIEN and STSDONEIEN bit to enable EP0 setup packet and setup status done interrupt
12. Polling HSUSBD_PHYCTL register VBUSDET bit until it was set. It means device connect to host. Set HSUSBD_PHYCTL register DPPUEN bit to clear SE0

20.4.2 Interrupt Service Routine

USBD control interrupt processing as follow:

1. Read HSUSBD_GINTSTS register and HSUSBD_GINTEN register to mask. User can get the interrupt information.
2. Read HSUSBD_BUSINTSTS register and HSUSBD_BUSINTEN register to mask. If match, it means USB BUS interrupt occurred.
3. Read HSUSBD_CEPINTSTS register and HSUSBD_CEPINTEN register to mask. If match, it means control endpoint interrupt occurred.
4. Read HSUSBD_EPAINTSTS register and HSUSBD_EPAINTEN register to mask. If match, it means endpoint A interrupt occurred.
5. Read HSUSBD_EPBINTSTS register and HSUSBD_EPBINTEN register to mask. If match, it means endpoint B interrupt occurred.

20.4.3 Standard Request

USBD Controller process the Standard Request

1. Control endpoint setup packet interrupt occurred.
2. Read HSUSBD_SETUP1_0, HSUSBD_SETUP3_2, HSUSBD_SETUP5_4 and HSUSBD_SETUP7_6 register to get the setup packet information
3. Analysis of the request. It supports, clear the NAK. (Set HSUSBD_CEPCTL register NAKCLR bit) and wait the status complete. Otherwise, send STALL to host. (Set HSUSBD_CEPCTL register STALLEN bit)

20.4.4 Set Address Request

USBD controller processes the “Set Address” request:

1. Control endpoint setup packet interrupt occurred.
2. Read HSUSBD_SETUP1_0, HSUSBD_SETUP3_2, HSUSBD_SETUP5_4 and HSUSBD_SETUP7_6 register to get the setup packet information
    1) Get bmRequestType from HSUSBD_SETUP1_0 register low byte
    2) Get bRequest from HSUSBD_SETUP1_0 register high byte
    3) Get wValue from HSUSBD_SETUP3_2 register
    4) Get wIndex from HSUSBD_SETUP5_4 register
    5) Get wLength from HSUSBD_SETUP7_6 register
3. Get the address from wValue

4. Clear NAK (Set HSUSBD_CEPCTL register NAKCLR bit)
5. Set HSUSBD_CEPINTSTS register STSDONEIF bit to 1 to clear status complete interrupt. Set HSUSBD_CEPINTEN register STSDONEIEN bit to 1 to enable interrupt
6. Waiting for status complete interrupt occurred
    1) Set HSUSBD_CEPINTEN register SETUPPKIEN bit to enable setup packet interrupt
    2) Fill address to HSUSBD_FADDR register
    3) Set HSUSBD_CEPINTSTS register STSDONEIF bit to 1 to clear status complete interrupt

20.4.5 Get Descriptor

USBD controller processes the “Get Descriptor” request:

1. Control endpoint setup packet interrupt occurred.
2. Read HSUSBD_SETUP1_0, HSUSBD_SETUP3_2, HSUSBD_SETUP5_4 and HSUSBD_SETUP7_6 register to get the setup packet information
    1) Get bmRequestType from HSUSBD_SETUP1_0 register low byte
    2) Get bRequest from HSUSBD_SETUP1_0 register high byte
    3) Get wValue from HSUSBD_SETUP3_2 register
    4) Get wIndex from HSUSBD_SETUP5_4 register
    5) Get wLength from HSUSBD_SETUP7_6 register
3. Get the descriptor type from wValue
4. Check wLength and descriptor length
5. Fill 1 to HSUSBD_CEPINTSTS register STSDONEIF and INTKIF bit to clear interrupt. Set HSUSBD_CEPINTEN register STSDONEIEN and INTKIEN bit to enable interrupt
6. Waiting for IN-token interrupt occurred
    1) Fill 1 to HSUSBD_CEPINTSTS register STSDONEIF and TXPKIF bit to clear interrupt. Set HSUSBD_CEPINTEN register STSDONEIEN and TXPKIEN bit to enable interrupt.
    2) Write descriptor data into HSUSBD_CEPDAT register
    3) Write descriptor length into HSUSBD_CEPTXCNT register to trigger data out
7. Fill 1 to HSUSBD_CEPINTSTS register INTKIF bit to clear interrupt
8. Waiting for TX interrupt occurred
    1) Fill HSUSBD_CEPINTSTS register STSDONEIF and TXPKIF bit to clear interrupt.
    2) Clear NAK (Set HSUSBD_CEPCTL register NAKCLR bit)
    3) Fill 1 to HSUSBD_CEPINTSTS register STSDONEIF bit to clear status complete interrupt.
    4) Set HSUSBD_CEPINTEN register STSDONEIEN and SETUPPKIEN bit to enable interrupt
9. Waiting for status complete interrupt occurred
    1) Set HSUSBD_CEPINTEN register SETUPPKIEN bit to enable setup packet interrupt
    2) Fill 1 to HSUSBD_CEPINTSTS register STSDONEIF bit to clear status complete interrupt

20.4.6 IN Transmission

USBD controller handles IN transmission through DMA:

1. Set HSUSBD_DMACTL register DMARD bit to 1 for DMA read. Fill the endpoint number to HSUSBD_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 HSUSBD_EPxINTEN register TXPKIEN bit to enable the data transmit interrupt
4. Check whether FIFO empty or not. (HSUSBD_EPxINTSTS register BUFEMPTYIF bit is 1)
5. Set HSUSBD_BUSINTEN register RSTIEN, SUSPENDIEN and DMADONEIEN bit to enable USB reset, suspend and DMA complete interrupts
6. Write physical source address to HSUSBD_DMAADDR register
7. Write transfer count to HSUSBD_DMACNT register
8. Set SBD_DMACTL register DMAEN bit to trigger DMA
9. Waiting for DMA complete interrupt occurred
    1) Set HSUSBD_BUSINTSTS register DMADONEIF bit to clear interrupt.
    2) Check whether last packet is less than maximum packet size. If so, set HSUSBD_EPxRSPCTL register SHORTTXEN bit to output the last packet

20.4.7 OUT Transmission

USBD controller handles OUT transmission through DMA:

1. Set HSUSBD_DMACTL register DMARD bit to 0 for DMA write. Fill endpoint number to HSUSBD_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 HSUSBD_BUSINTEN register RSTIEN, SUSPENDIEN and DMADONEIEN bit to enable USB reset, suspend and DMA complete interrupts
4. Write physical target address to HSUSBD_DMAADDR register
5. Write transfer count to HSUSBD_DMACNT register
6. Set SBD_DMACTL register DMAEN bit to trigger DMA
7. Waiting for DMA complete interrupt occurred
    1) Set HSUSBD_BUSINTSTS register DMADONEIF bit to clear interrupt
    2) Check whether received length is mass storage CBW or data length
    3) Set HSUSBD_EPxINTEN register RXPKIEN bit to enable receive packet interrupt
8. Waiting for received data interrupt occurred. Clear HSUSBD_EPxINTEN register RXPKIEN bit to disable receive data

20.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
HSUSBD Base Address: **HSUSBD_BA = 0xB001_6000**				
HSUSBD_GINTSTS	HSUSBD_BA+0x000	R	Global Interrupt Status Register	0x0000_0000
HSUSBD_GINTEN	HSUSBD_BA+0x008	R/W	Global Interrupt Enable Register	0x0000_0001
HSUSBD_BUSINTSTS	HSUSBD_BA+0x010	R/W	USB Bus Interrupt Status Register	0x0000_0000
HSUSBD_BUSINTEN	HSUSBD_BA+0x014	R/W	USB Bus Interrupt Enable Register	0x0000_0040
HSUSBD_OPER	HSUSBD_BA+0x018	R/W	USB Operational Register	0x0000_0002
HSUSBD_FRAMECNT	HSUSBD_BA+0x01C	R	USB Frame Count Register	0x0000_0000
HSUSBD_FADDR	HSUSBD_BA+0x020	R/W	USB Function Address Register	0x0000_0000
HSUSBD_TEST	HSUSBD_BA+0x024	R/W	USB Test Mode Register	0x0000_0000
HSUSBD_CEPDAT	HSUSBD_BA+0x028	R/W	Control Endpoint Data Buffer	0x0000_0000
HSUSBD_CEPCTL	HSUSBD_BA+0x02C	R/W	Control Endpoint Control Register	0x0000_0000
HSUSBD_CEPINTEN	HSUSBD_BA+0x030	R/W	Control Endpoint Interrupt Enable	0x0000_0000
HSUSBD_CEPINTSTS	HSUSBD_BA+0x034	R/W	Control Endpoint Interrupt Status	0x0000_1800
HSUSBD_CEPTXCNT	HSUSBD_BA+0x038	R/W	Control Endpoint In Transfer Data Count	0x0000_0000
HSUSBD_CEPRXCNT	HSUSBD_BA+0x03C	R	Control Endpoint Out Transfer Data Count	0x0000_0000
HSUSBD_CEPDATCNT	HSUSBD_BA+0x040	R	Control Endpoint Data Count	0x0000_0000
HSUSBD_SETUP1_0	HSUSBD_BA+0x044	R	Setup1 & Setup0 bytes	0x0000_0000
HSUSBD_SETUP3_2	HSUSBD_BA+0x048	R	Setup3 & Setup2 Bytes	0x0000_0000
HSUSBD_SETUP5_4	HSUSBD_BA+0x04C	R	Setup5 & Setup4 Bytes	0x0000_0000
HSUSBD_SETUP7_6	HSUSBD_BA+0x050	R	Setup7 & Setup6 Bytes	0x0000_0000
HSUSBD_CEPBUFSTART	HSUSBD_BA+0x054	R/W	Control Endpoint RAM Start Address Register	0x0000_0000
HSUSBD_CEPBUFEND	HSUSBD_BA+0x058	R/W	Control Endpoint RAM End Address Register	0x0000_0000
HSUSBD_DMACTL	HSUSBD_BA+0x05C	R/W	DMA Control Status Register	0x0000_0000
HSUSBD_DMACNT	HSUSBD_BA+0x060	R/W	DMA Count Register	0x0000_0000
HSUSBD_EPADAT	HSUSBD_BA+0x064	R/W	Endpoint A Data Register	0x0000_0000
HSUSBD_EPAINTSTS	HSUSBD_BA+0x068	R/W	Endpoint A Interrupt Status Register	0x0000_0003
HSUSBD_EPAINTEN	HSUSBD_BA+0x06C	R/W	Endpoint A Interrupt Enable Register	0x0000_0000
HSUSBD_EPADATCNT	HSUSBD_BA+0x070	R	Endpoint A Data Available Count Register	0x0000_0000
HSUSBD_EPARSPCTL	HSUSBD_BA+0x074	R/W	Endpoint A Response Control Register	0x0000_0000

HSUSBD_EPAMPS	HSUSBD_BA+0x078	R/W	Endpoint A Maximum Packet Size Register	0x0000_0000
HSUSBD_EPATXCNT	HSUSBD_BA+0x07C	R/W	Endpoint A Transfer Count Register	0x0000_0000
HSUSBD_EPACFG	HSUSBD_BA+0x080	R/W	Endpoint A Configuration Register	0x0000_0012
HSUSBD_EPABUFSTART	HSUSBD_BA+0x084	R/W	Endpoint A RAM Start Address Register	0x0000_0000
HSUSBD_EPABUFEND	HSUSBD_BA+0x088	R/W	Endpoint A RAM End Address Register	0x0000_0000
HSUSBD_EPBDAT	HSUSBD_BA+0x08C	R/W	Endpoint B Data Register	0x0000_0000
HSUSBD_EPBINTSTS	HSUSBD_BA+0x090	R/W	Endpoint B Interrupt Status Register	0x0000_0003
HSUSBD_EPBINTEN	HSUSBD_BA+0x094	R/W	Endpoint B Interrupt Enable Register	0x0000_0000
HSUSBD_EPBDATCNT	HSUSBD_BA+0x098	R	Endpoint B Data Available Count Register	0x0000_0000
HSUSBD_EPBRSPCTL	HSUSBD_BA+0x09C	R/W	Endpoint B Response Control Register	0x0000_0000
HSUSBD_EPBMPS	HSUSBD_BA+0x0A0	R/W	Endpoint B Maximum Packet Size Register	0x0000_0000
HSUSBD_EPBTXCNT	HSUSBD_BA+0x0A4	R/W	Endpoint B Transfer Count Register	0x0000_0000
HSUSBD_EPBCFG	HSUSBD_BA+0x0A8	R/W	Endpoint B Configuration Register	0x0000_0022
HSUSBD_EPBBUFSTART	HSUSBD_BA+0x0AC	R/W	Endpoint B RAM Start Address Register	0x0000_0000
HSUSBD_EPBBUFEND	HSUSBD_BA+0x0B0	R/W	Endpoint B RAM End Address Register	0x0000_0000
HSUSBD_EPCDAT	HSUSBD_BA+0x0B4	R/W	Endpoint C Data Register	0x0000_0000
HSUSBD_EPCINTSTS	HSUSBD_BA+0x0B8	R/W	Endpoint C Interrupt Status Register	0x0000_0003
HSUSBD_EPCINTEN	HSUSBD_BA+0x0BC	R/W	Endpoint C Interrupt Enable Register	0x0000_0000
HSUSBD_EPCDATCNT	HSUSBD_BA+0x0C0	R	Endpoint C Data Available Count Register	0x0000_0000
HSUSBD_EPCRSPCTL	HSUSBD_BA+0x0C4	R/W	Endpoint C Response Control Register	0x0000_0000
HSUSBD_EPCMPS	HSUSBD_BA+0x0C8	R/W	Endpoint C Maximum Packet Size Register	0x0000_0000
HSUSBD_EPCTXCNT	HSUSBD_BA+0x0CC	R/W	Endpoint C Transfer Count Register	0x0000_0000
HSUSBD_EPCCFG	HSUSBD_BA+0x0D0	R/W	Endpoint C Configuration Register	0x0000_0032
HSUSBD_EPCBUFSTART	HSUSBD_BA+0x0D4	R/W	Endpoint C RAM Start Address Register	0x0000_0000
HSUSBD_EPCBUFEND	HSUSBD_BA+0x0D8	R/W	Endpoint C RAM End Address Register	0x0000_0000
HSUSBD_EPDDAT	HSUSBD_BA+0x0DC	R/W	Endpoint D Data Register	0x0000_0000
HSUSBD_EPDINTSTS	HSUSBD_BA+0x0E0	R/W	Endpoint D Interrupt Status Register	0x0000_0003
HSUSBD_EPDINTEN	HSUSBD_BA+0x0E4	R/W	Endpoint D Interrupt Enable Register	0x0000_0000
HSUSBD_EPDDATCNT	HSUSBD_BA+0x0E8	R	Endpoint D Data Available Count Register	0x0000_0000
HSUSBD_EPDRSPCTL	HSUSBD_BA+0x0EC	R/W	Endpoint D Response Control Register	0x0000_0000
HSUSBD_EPDMPS	HSUSBD_BA+0x0F0	R/W	Endpoint D Maximum Packet Size Register	0x0000_0000

HSUSBD_EPDTXCNT	HSUSBD_BA+0x0F4	R/W	Endpoint D Transfer Count Register	0x0000_0000
HSUSBD_EPDCFG	HSUSBD_BA+0x0F8	R/W	Endpoint D Configuration Register	0x0000_0042
HSUSBD_EPDBUFSTART	HSUSBD_BA+0x0FC	R/W	Endpoint D RAM Start Address Register	0x0000_0000
HSUSBD_EPDBUFEND	HSUSBD_BA+0x100	R/W	Endpoint D RAM End Address Register	0x0000_0000
HSUSBD_EPEDAT	HSUSBD_BA+0x104	R/W	Endpoint E Data Register	0x0000_0000
HSUSBD_EPEINTSTS	HSUSBD_BA+0x108	R/W	Endpoint E Interrupt Status Register	0x0000_0003
HSUSBD_EPEINTEN	HSUSBD_BA+0x10C	R/W	Endpoint E Interrupt Enable Register	0x0000_0000
HSUSBD_EPEDATCNT	HSUSBD_BA+0x110	R	Endpoint E Data Available Count Register	0x0000_0000
HSUSBD_EPERSPCTL	HSUSBD_BA+0x114	R/W	Endpoint E Response Control Register	0x0000_0000
HSUSBD_EPEMPS	HSUSBD_BA+0x118	R/W	Endpoint E Maximum Packet Size Register	0x0000_0000
HSUSBD_EPETXCNT	HSUSBD_BA+0x11C	R/W	Endpoint E Transfer Count Register	0x0000_0000
HSUSBD_EPECFG	HSUSBD_BA+0x120	R/W	Endpoint E Configuration Register	0x0000_0052
HSUSBD_EPEBUFSTART	HSUSBD_BA+0x124	R/W	Endpoint E RAM Start Address Register	0x0000_0000
HSUSBD_EPEBUFEND	HSUSBD_BA+0x128	R/W	Endpoint E RAM End Address Register	0x0000_0000
HSUSBD_EPFDAT	HSUSBD_BA+0x12C	R/W	Endpoint F Data Register	0x0000_0000
HSUSBD_EPFINTSTS	HSUSBD_BA+0x130	R/W	Endpoint F Interrupt Status Register	0x0000_0003
HSUSBD_EPFINTEN	HSUSBD_BA+0x134	R/W	Endpoint F Interrupt Enable Register	0x0000_0000
HSUSBD_EPFDATCNT	HSUSBD_BA+0x138	R	Endpoint F Data Available Count Register	0x0000_0000
HSUSBD_EPFRSPCTL	HSUSBD_BA+0x13C	R/W	Endpoint F Response Control Register	0x0000_0000
HSUSBD_EPFMPS	HSUSBD_BA+0x140	R/W	Endpoint F Maximum Packet Size Register	0x0000_0000
HSUSBD_EPFTXCNT	HSUSBD_BA+0x144	R/W	Endpoint F Transfer Count Register	0x0000_0000
HSUSBD_EPFCFG	HSUSBD_BA+0x148	R/W	Endpoint F Configuration Register	0x0000_0062
HSUSBD_EPFBUFSTART	HSUSBD_BA+0x14C	R/W	Endpoint F RAM Start Address Register	0x0000_0000
HSUSBD_EPFBUFEND	HSUSBD_BA+0x150	R/W	Endpoint F RAM End Address Register	0x0000_0000
HSUSBD_EPGDAT	HSUSBD_BA+0x154	R/W	Endpoint G Data Register	0x0000_0000
HSUSBD_EPGINTSTS	HSUSBD_BA+0x158	R/W	Endpoint G Interrupt Status Register	0x0000_0003
HSUSBD_EPGINTEN	HSUSBD_BA+0x15C	R/W	Endpoint G Interrupt Enable Register	0x0000_0000
HSUSBD_EPGDATCNT	HSUSBD_BA+0x160	R	Endpoint G Data Available Count Register	0x0000_0000
HSUSBD_EPGRSPCTL	HSUSBD_BA+0x164	R/W	Endpoint G Response Control Register	0x0000_0000
HSUSBD_EPGMPS	HSUSBD_BA+0x168	R/W	Endpoint G Maximum Packet Size Register	0x0000_0000
HSUSBD_EPGTXCNT	HSUSBD_BA+0x16C	R/W	Endpoint G Transfer Count Register	0x0000_0000
HSUSBD_EPGCFG	HSUSBD_BA+0x170	R/W	Endpoint G Configuration Register	0x0000_0072

HSUSBD_EPGBUFSTART	HSUSBD_BA+0x174	R/W	Endpoint G RAM Start Address Register	0x0000_0000
HSUSBD_EPGBUFEND	HSUSBD_BA+0x178	R/W	Endpoint G RAM End Address Register	0x0000_0000
HSUSBD_EPHDAT	HSUSBD_BA+0x17C	R/W	Endpoint H Data Register	0x0000_0000
HSUSBD_EPHINTSTS	HSUSBD_BA+0x180	R/W	Endpoint H Interrupt Status Register	0x0000_0003
HSUSBD_EPHINTEN	HSUSBD_BA+0x184	R/W	Endpoint H Interrupt Enable Register	0x0000_0000
HSUSBD_EPHDATCNT	HSUSBD_BA+0x188	R	Endpoint H Data Available Count Register	0x0000_0000
HSUSBD_EPHRSPCTL	HSUSBD_BA+0x18C	R/W	Endpoint H Response Control Register	0x0000_0000
HSUSBD_EPHMPS	HSUSBD_BA+0x190	R/W	Endpoint H Maximum Packet Size Register	0x0000_0000
HSUSBD_EPHTXCNT	HSUSBD_BA+0x194	R/W	Endpoint H Transfer Count Register	0x0000_0000
HSUSBD_EPHCFG	HSUSBD_BA+0x198	R/W	Endpoint H Configuration Register	0x0000_0082
HSUSBD_EPHBUFSTART	HSUSBD_BA+0x19C	R/W	Endpoint H RAM Start Address Register	0x0000_0000
HSUSBD_EPHBUFEND	HSUSBD_BA+0x1A0	R/W	Endpoint H RAM End Address Register	0x0000_0000
HSUSBD_EPIDAT	HSUSBD_BA+0x1A4	R/W	Endpoint I Data Register	0x0000_0000
HSUSBD_EPIINTSTS	HSUSBD_BA+0x1A8	R/W	Endpoint I Interrupt Status Register	0x0000_0003
HSUSBD_EPIINTEN	HSUSBD_BA+0x1AC	R/W	Endpoint I Interrupt Enable Register	0x0000_0000
HSUSBD_EPIDATCNT	HSUSBD_BA+0x1B0	R	Endpoint I Data Available Count Register	0x0000_0000
HSUSBD_EPIRSPCTL	HSUSBD_BA+0x1B4	R/W	Endpoint I Response Control Register	0x0000_0000
HSUSBD_EPIMPS	HSUSBD_BA+0x1B8	R/W	Endpoint I Maximum Packet Size Register	0x0000_0000
HSUSBD_EPITXCNT	HSUSBD_BA+0x1BC	R/W	Endpoint I Transfer Count Register	0x0000_0000
HSUSBD_EPICFG	HSUSBD_BA+0x1C0	R/W	Endpoint I Configuration Register	0x0000_0092
HSUSBD_EPIBUFSTART	HSUSBD_BA+0x1C4	R/W	Endpoint I RAM Start Address Register	0x0000_0000
HSUSBD_EPIBUFEND	HSUSBD_BA+0x1C8	R/W	Endpoint I RAM End Address Register	0x0000_0000
HSUSBD_EPJDAT	HSUSBD_BA+0x1CC	R/W	Endpoint J Data Register	0x0000_0000
HSUSBD_EPJINTSTS	HSUSBD_BA+0x1D0	R/W	Endpoint J Interrupt Status Register	0x0000_0003
HSUSBD_EPJINTEN	HSUSBD_BA+0x1D4	R/W	Endpoint J Interrupt Enable Register	0x0000_0000
HSUSBD_EPJDATCNT	HSUSBD_BA+0x1D8	R	Endpoint J Data Available Count Register	0x0000_0000
HSUSBD_EPJRSPCTL	HSUSBD_BA+0x1DC	R/W	Endpoint J Response Control Register	0x0000_0000
HSUSBD_EPJMPS	HSUSBD_BA+0x1E0	R/W	Endpoint J Maximum Packet Size Register	0x0000_0000
HSUSBD_EPJTXCNT	HSUSBD_BA+0x1E4	R/W	Endpoint J Transfer Count Register	0x0000_0000
HSUSBD_EPJCFG	HSUSBD_BA+0x1E8	R/W	Endpoint J Configuration Register	0x0000_00A2
HSUSBD_EPJBUFSTART	HSUSBD_BA+0x1EC	R/W	Endpoint J RAM Start Address Register	0x0000_0000
HSUSBD_EPJBUFEND	HSUSBD_BA+0x1F0	R/W	Endpoint J RAM End Address Register	0x0000_0000

HSUSBD_EPKDAT	HSUSBD_BA+0x1F4	R/W	Endpoint K Data Register	0x0000_0000
HSUSBD_EPKINTSTS	HSUSBD_BA+0x1F8	R/W	Endpoint K Interrupt Status Register	0x0000_0003
HSUSBD_EPKINTEN	HSUSBD_BA+0x1FC	R/W	Endpoint K Interrupt Enable Register	0x0000_0000
HSUSBD_EPKDATCNT	HSUSBD_BA+0x200	R	Endpoint K Data Available Count Register	0x0000_0000
HSUSBD_EPKRSPCTL	HSUSBD_BA+0x204	R/W	Endpoint K Response Control Register	0x0000_0000
HSUSBD_EPKMPS	HSUSBD_BA+0x208	R/W	Endpoint K Maximum Packet Size Register	0x0000_0000
HSUSBD_EPKTXCNT	HSUSBD_BA+0x20C	R/W	Endpoint K Transfer Count Register	0x0000_0000
HSUSBD_EPKCFG	HSUSBD_BA+0x210	R/W	Endpoint K Configuration Register	0x0000_00B2
HSUSBD_EPKBUFSTART	HSUSBD_BA+0x214	R/W	Endpoint K RAM Start Address Register	0x0000_0000
HSUSBD_EPKBUFEND	HSUSBD_BA+0x218	R/W	Endpoint K RAM End Address Register	0x0000_0000
HSUSBD_EPLDAT	HSUSBD_BA+0x21C	R/W	Endpoint L Data Register	0x0000_0000
HSUSBD_EPLINTSTS	HSUSBD_BA+0x220	R/W	Endpoint L Interrupt Status Register	0x0000_0003
HSUSBD_EPLINTEN	HSUSBD_BA+0x224	R/W	Endpoint L Interrupt Enable Register	0x0000_0000
HSUSBD_EPLDATCNT	HSUSBD_BA+0x228	R	Endpoint L Data Available Count Register	0x0000_0000
HSUSBD_EPLRSPCTL	HSUSBD_BA+0x22C	R/W	Endpoint L Response Control Register	0x0000_0000
HSUSBD_EPLMPS	HSUSBD_BA+0x230	R/W	Endpoint L Maximum Packet Size Register	0x0000_0000
HSUSBD_EPLTXCNT	HSUSBD_BA+0x234	R/W	Endpoint L Transfer Count Register	0x0000_0000
HSUSBD_EPLCFG	HSUSBD_BA+0x238	R/W	Endpoint L Configuration Register	0x0000_00C2
HSUSBD_EPLBUFSTART	HSUSBD_BA+0x23C	R/W	Endpoint L RAM Start Address Register	0x0000_0000
HSUSBD_EPLBUFEND	HSUSBD_BA+0x240	R/W	Endpoint L RAM End Address Register	0x0000_0000
HSUSBD_DMAADDR	HSUSBD_BA+0x700	R/W	AHB DMA Address Register	0x0000_0000
HSUSBD_PHYCTL	HSUSBD_BA+0x704	R/W	USB PHY Control Register	0x0000_0420

21 USB HOST CONTROLLER

21.1 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.

21.2 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.
- Support up to 6 USB Lite full-speed ports.
- Built-in DMA for real-time data transfer.

21.3 Block Diagram

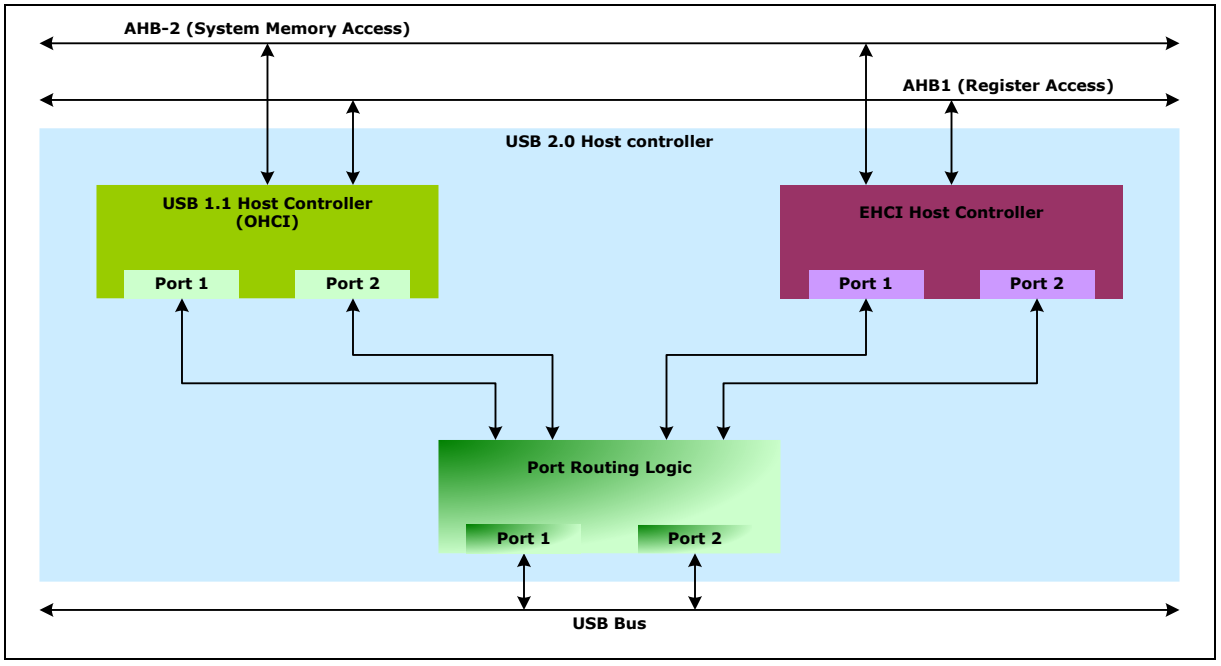


Figure 21.3-1 USB Host Controller Block Diagram

21.3.1 Basic Configuration

Set the USBH(CLK_HCLKEN[18]) bit to enable USB host clock.

USB 1.1 Host 48 MHz clock source is come from USB 2.0 USB PHY 480 MHz divided by 10. Driver programs USB_S(CLK_DIVCTL2[4:3]) select USB 1.1 engine clock source come from USB 2.0 PHY 0 or PHY1.

Set SUSPEND(USBPCR0[8]) bit to enable USB PHY0 and set SUSPEND(USBPCR1[8]) bit to enable USB PHY1. User has to check if CLKVALID(USBPCR0[11]) is high before starting to use USB host controller.

21.3.2 USB Host Port 0

USB port 0 is a dual-role port. It can be a USB device port or a USB host port. To work as host or device role is depend on USB ID pin. If ID pin is low, USB port 0 works as a USB device port. If ID pin is high, USB port 0 works as a USB host port. USBID[SYS_PWRON[16]] indicates the current role of USB port 0.

Driver can force to change the USB port0 host/device role. First, set USRHDSEN(SYS_MISCFCR[11]) bit, thus USBID[SYS_PWRON[16]] becomes writable. Then, driver can write 1 to USBID[SYS_PWRON[16]] to force USB port 0 be a host port, or write 0 to USBID[SYS_PWRON[16]] to force USB port 0 be a device port.

21.3.3 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.

21.3.4 OHCI Controller

21.3.4.1 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.

21.3.4.2 AHB Master

The master issues the address and data onto the bus when granted.

21.3.4.3 AHB Slave

The configuration of the Host Controller is through the slave interface.

21.3.4.4 List Processor

The List Processor manages the data structures from the Host Controller Driver and coordinates all activity within the Host Controller.

21.3.4.5 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.

21.3.4.6 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.

21.3.4.7 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.

21.3.4.8 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.

21.3.4.9 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.

21.3.4.10 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.

21.3.4.11 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.

21.3.4.12 USB Lite

NUC980 supports 6 USB Lite full-speed ports.Several GPIO pins can be slected for D+/D- pins of USB Lite ports. USB Lite 0 ~ 5 are mapped to OHCI (USB 1.1) port 2 ~ 7. OHCI port 0 and 1 are the root hub ports.

21.4 Functional Description

21.4.1 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.11 multifunction pin for USBH0_VBUSVLS and PE.12 for USBH_PWREN respectively. NUC980 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.
- If USBH Lite ports were used, configure the corresponding multifunction pins.
- Initialize EHCI Host Controller, which services USB 2.0 high-speed devices.

21.4.2 Root Hub Port Routing Logic

NUC980 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.

21.4.3 OHCI

The NUC980 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 NUC980.

21.4.3.1 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.

21.4.3.2 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:

	31–16	15–13	12	11	10–7	6–0	31–11	10–0
Dword 0	—	MPS	F	K	S	D	EN	FA
Dword 1	TD Queue Tail Pointer (TailP)							—
Dword 2	TD Queue Head Pointer (HeadP)							0 C H
Dword 3	Next Endpoint Descriptor (NextED)							—

Table 21.4-1 End Point Descriptor Format

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

21.4.3.3 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

	31–28	27–26	25–24	23–21	20–19	18	17–0
Dword 0	CC	EC	T	DI	DP	R	—
Dword 1	Current Buffer Pointer (CBP)						
Dword 2	Next TD (NextTD)						0
Dword 3	Buffer end (BE)						

Table 21.4-2 General Transfer Descriptor Format

Isochronous Transfer Descriptor

	31–28	27–24	23–21	20–16	15–12	11–10	9–0
Dword 0	CC		FC	DI	—	SF	
Dword 1	Buffer Page 0 (BP0)					—	
Dword 2	Next TD						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	

Table 21.4-3 Isochronous Transfer Descriptor Format

21.4.3.4 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	HccaInterrruptTable	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 **HccaFrameNumber**, 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 **HccaDoneHead** was written.
0x88	116	reserved	Reserved for use by Host Controller.

Table 21.4-4 HCCA Structure

21.4.3.5 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 HCC A, 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 ± 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 NUC980 Series MCU USB Root Hub uses global power switching mode)
12. Enable AIC (NUC980 Advanced Interrupt Controller) OHCI interrupt. The IRQ number of OHCI is 24.

21.4.3.6 Interrupt Processing

NUC980 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(HcRhPtrx[16]), PESC(HcRhPtrx[17]), or PSSC(HcRhPtrx[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]).

21.4.3.7 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.

21.4.3.8 Root Hub

The Root Hub is integrated into Host Controller and the control of Root Hub is done by accessing register files. NUC980 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[1:8] register presents the current status and reflects the change of status of a Root Hub port. NUC980 Series MCU OHCI Root Hub has 8 hub ports, the HcRhPrt[1] ~ HcRhPrt[8] are respectively dedicated to port 1 ~ 8.

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 NUC980 OHCI Root Hub are listed in the followings:

- 8 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]).

21.4.4 EHCI

NUC980 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 NUC980.

21.4.4.1 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 (iTD)
- Split Transaction Isochronous Transfer Descriptor (siTD)
- Queue Element Transfer Descriptor (qTD)
- Queue Head
- Periodic Frame Span Traversal Node (FSTN)

21.4.4.2 Isochronous Transfer Descriptor (iTD)

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 iTD 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				

Table 21.4-5 Isochronous Transfer Descriptor Structure

21.4.4.3 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

Table 21.4-6 Split Transfer Isochronous Transfer Descriptor Structure

21.4.4.4 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				

Table 21.4-7 Queue Element Transfer Descriptor Structure

21.4.4.5 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.

21.4.4.6 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
00h	Reserved
01h	1 micro-frame
02h	2 micro-frames
04h	4 micro-frames
08h	8 micro-frames (default, equates to 1 ms)
10h	16 micro-frames (2 ms)
20h	32 micro-frames (4 ms)
40h	64 micro-frames (8 ms)

21.4.4.7 Interrupt Processing

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 elinquished 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.

Reclaimation (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.

21.4.4.8 Root Hub

NUC980 Series MCU EHCI host controller implements two port registers, UPSCR0 and UPSCR1. NUC980 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

Table 21.4-8 Port Status Indication

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.

21.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
USBH Base Address: **USBH_BA = 0xB001_7000** **HSUSBH_BA = 0xB001_5000**				
HcRevision	USBH_BA+0x000	R	Host Controller Revision Register	0x0000_0110
HcControl	USBH_BA+0x004	R/W	Host Controller Control Register	0x0000_0000
HcCommandStatus	USBH_BA+0x008	R/W	Host Controller Command Status Register	0x0000_0000
HcInterruptStatus	USBH_BA+0x00C	R/W	Host Controller Interrupt Status Register	0x0000_0000
HcInterruptEnable	USBH_BA+0x010	R/W	Host Controller Interrupt Enable Register	0x0000_0000
HcInterruptDisable	USBH_BA+0x014	R/W	Host Controller Interrupt Disable Register	0x0000_0000
HcHCCA	USBH_BA+0x018	R/W	Host Controller Communication Area Register	0x0000_0000
HcPeriodCurrentED	USBH_BA+0x01C	R/W	Host Controller Period Current ED Register	0x0000_0000
HcControlHeadED	USBH_BA+0x020	R/W	Host Controller Control Head ED Register	0x0000_0000
HcControlCurrentED	USBH_BA+0x024	R/W	Host Controller Control Current ED Register	0x0000_0000
HcBulkHeadED	USBH_BA+0x028	R/W	Host Controller Bulk Head ED Register	0x0000_0000
HcBulkCurrentED	USBH_BA+0x02C	R/W	Host Controller Bulk Current ED Register	0x0000_0000
HcDoneHead	USBH_BA+0x030	R/W	Host Controller Done Head Register	0x0000_0000
HcFmInterval	USBH_BA+0x034	R/W	Host Controller Frame Interval Register	0x0000_2EDF
HcFmRemaining	USBH_BA+0x038	R	Host Controller Frame Remaining Register	0x0000_0000
HcFmNumber	USBH_BA+0x03C	R	Host Controller Frame Number Register	0x0000_0000
HcPeriodicStart	USBH_BA+0x040	R/W	Host Controller Periodic Start Register	0x0000_0000
HcLSThreshold	USBH_BA+0x044	R/W	Host Controller Low-speed Threshold Register	0x0000_0628
HcRhDescriptorA	USBH_BA+0x048	R/W	Host Controller Root Hub Descriptor A Register	0x0000_0908
HcRhDescriptorB	USBH_BA+0x04C	R/W	Host Controller Root Hub Descriptor B Register	0x0000_0000
HcRhStatus	USBH_BA+0x050	R/W	Host Controller Root Hub Status Register	0x0000_0000
HcRhPortStatus0	USBH_BA+0x054	R/W	Host Controller Root Hub Port Status [0]	0x0000_0000
HcRhPortStatus1	USBH_BA+0x058	R/W	Host Controller Root Hub Port Status [1]	0x0000_0000
HcRhPortStatus2	USBH_BA+0x05C	R/W	Host Controller Root Hub Port Status [2]	0x0000_0000
HcRhPortStatus3	USBH_BA+0x060	R/W	Host Controller Root Hub Port Status [3]	0x0000_0000
HcRhPortStatus4	USBH_BA+0x064	R/W	Host Controller Root Hub Port Status [4]	0x0000_0000

Register	Offset	R/W	Description	Reset Value
HcRhPortStatus5	USBH_BA+0x068	R/W	Host Controller Root Hub Port Status [5]	0x0000_0000
HcRhPortStatus6	USBH_BA+0x06C	R/W	Host Controller Root Hub Port Status [6]	0x0000_0000
HcRhPortStatus7	USBH_BA+0x070	R/W	Host Controller Root Hub Port Status [7]	0x0000_0000
HcPhyControl	USBH_BA+0x200	R/W	Host Controller PHY Control Register	0x0000_0000
HcMiscControl	USBH_BA+0x204	R/W	Host Controller Miscellaneous Control Register	0x0000_0000
EHCVNR	HSUSBH_BA+0x000	R	EHCI Version Number Register	0x0095_0020
EHCSPR	HSUSBH_BA+0x004	R	EHCI Structural Parameters Register	0x0000_0012
EHCCPR	HSUSBH_BA+0x008	R	EHCI Capability Parameters Register	0x0000_0000
UCMDR	HSUSBH_BA+0x020	R/W	USB Command Register	0x0008_0000
USTSR	HSUSBH_BA+0x024	R/W	USB Status Register	0x0000_1000
UIENR	HSUSBH_BA+0x028	R/W	USB Interrupt Enable Register	0x0000_0000
UFINDR	HSUSBH_BA+0x02C	R/W	USB Frame Index Register	0x0000_0000
UPFLBAR	HSUSBH_BA+0x034	R/W	USB Periodic Frame List Base Address Register	0x0000_0000
UCALAR	HSUSBH_BA+0x038	R/W	USB Current Asynchronous List Address Register	0x0000_0000
UASSTR	HSUSBH_BA+0x03C	R/W	USB Asynchronous Schedule Sleep Timer Register	0x0000_0BD6
UCFGR	HSUSBH_BA+0x060	R/W	USB Configure Flag Register	0x0000_0000
UPSCR0	HSUSBH_BA+0x064	R/W	USB Port 0 Status and Control Register	0x0000_2000
UPSCR1	HSUSBH_BA+0x068	R/W	USB Port 1 Status and Control Register	0x0000_2000
USBPCR0	HSUSBH_BA+0x0C4	R/W	USB PHY 0 Control Register	0x0000_0060
USBPCR1	HSUSBH_BA+0x0C8	R/W	USB PHY 1 Control Register	0x0000_0020

22 CAN

22.1 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.

22.2 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

22.3 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.

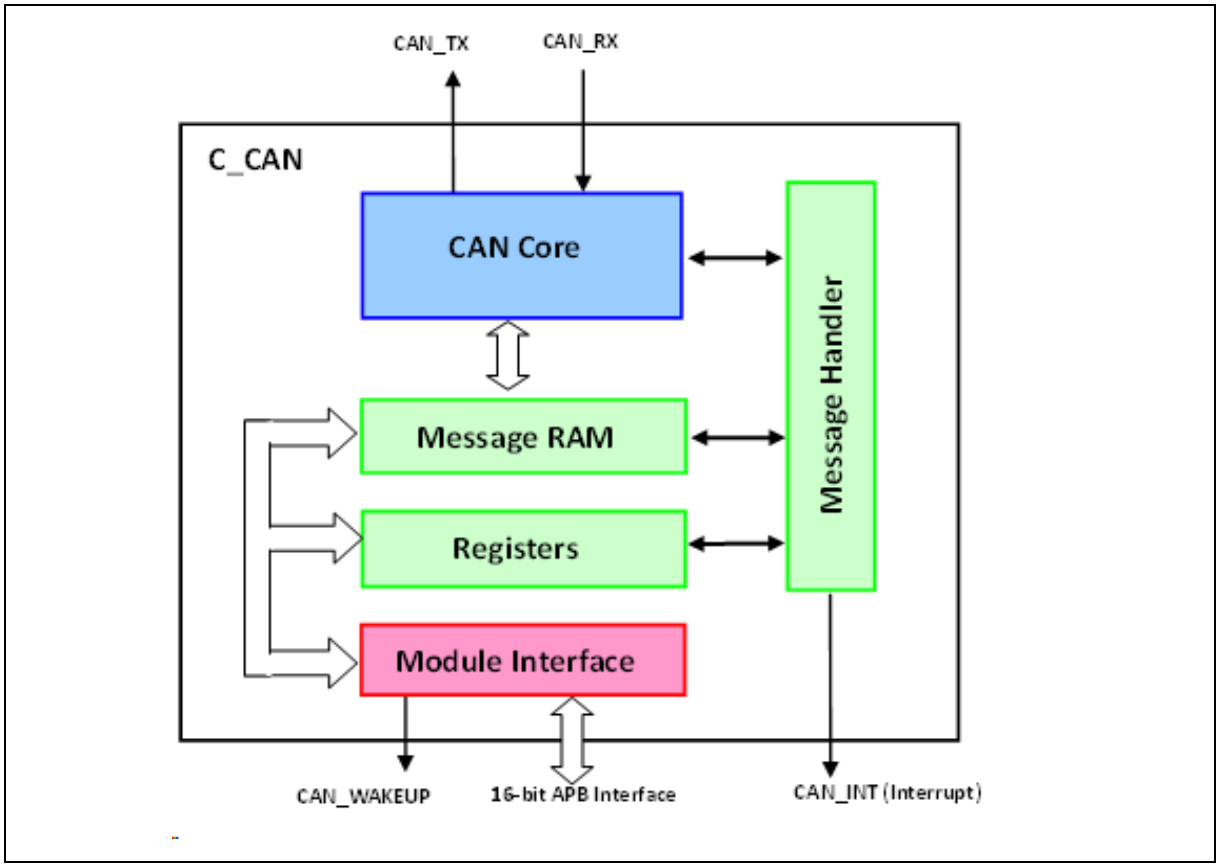


Figure 22.3-1 CAN Controller Block Diagram

22.4 Functional Description

22.4.1 CAN Protocol

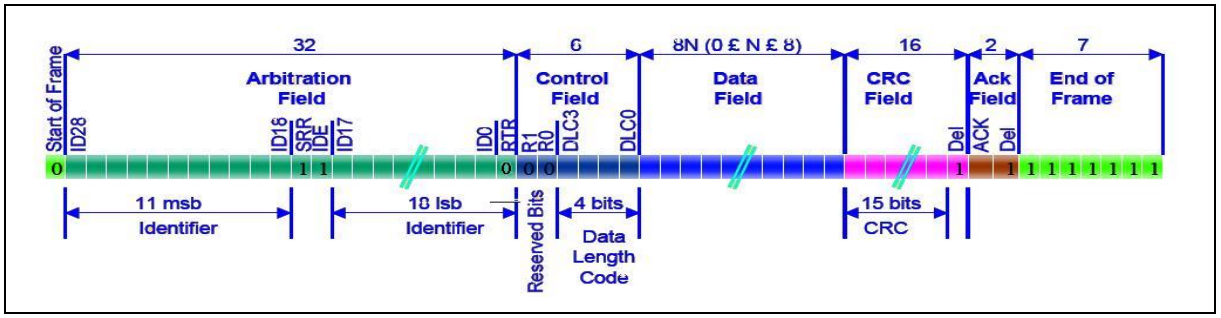


Figure 22.4-1 CAN Protocol

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

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

22.4.2 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：$f_{speed} = {}^{1}/_{t_{NBT}}$

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.

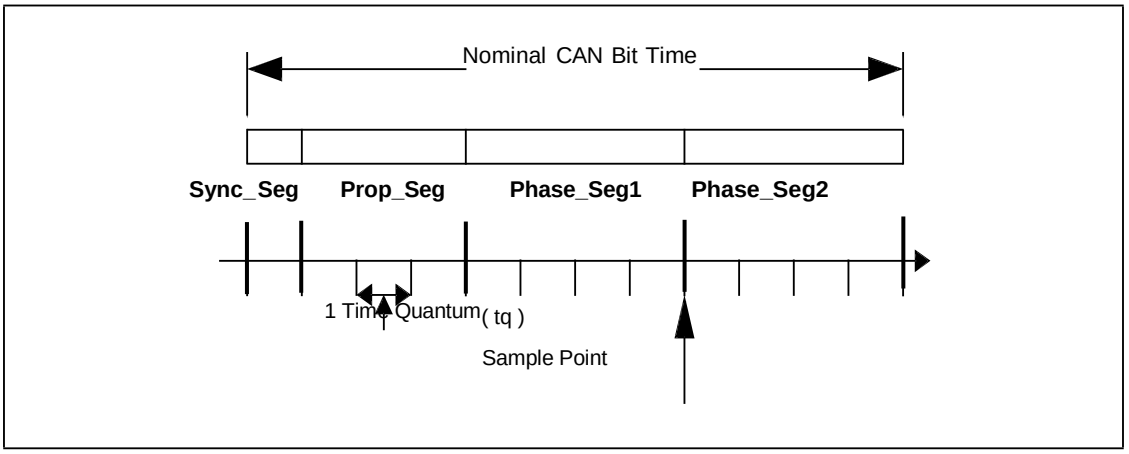


Figure 22.4-2 CAN Bit Time

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

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

BRP: Baud Rate PreScaler Value

f_{APB_CLK} : System Clock

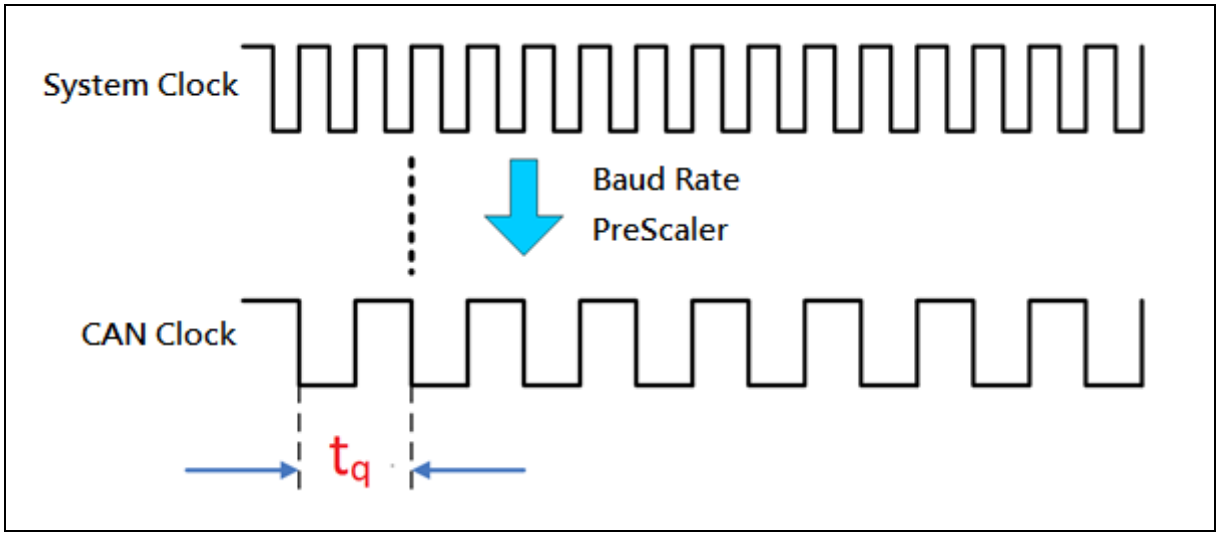


Figure 22.4-3 CAN Clock

In these bit timing registers of CAN controller

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

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

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

TSEG1, TSEG2 are the control bit of register CAN_BTIME.

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

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

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

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

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

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.

22.4.3 CAN Module Register

CAN module register address base is CAN0_BA=0xB800_0000，There are three module 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 or 2

Table 22.4-1 CAN Module Register

- 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) is status information provided by the Message Handler FSM.

  These registers relationship show in follow fig:

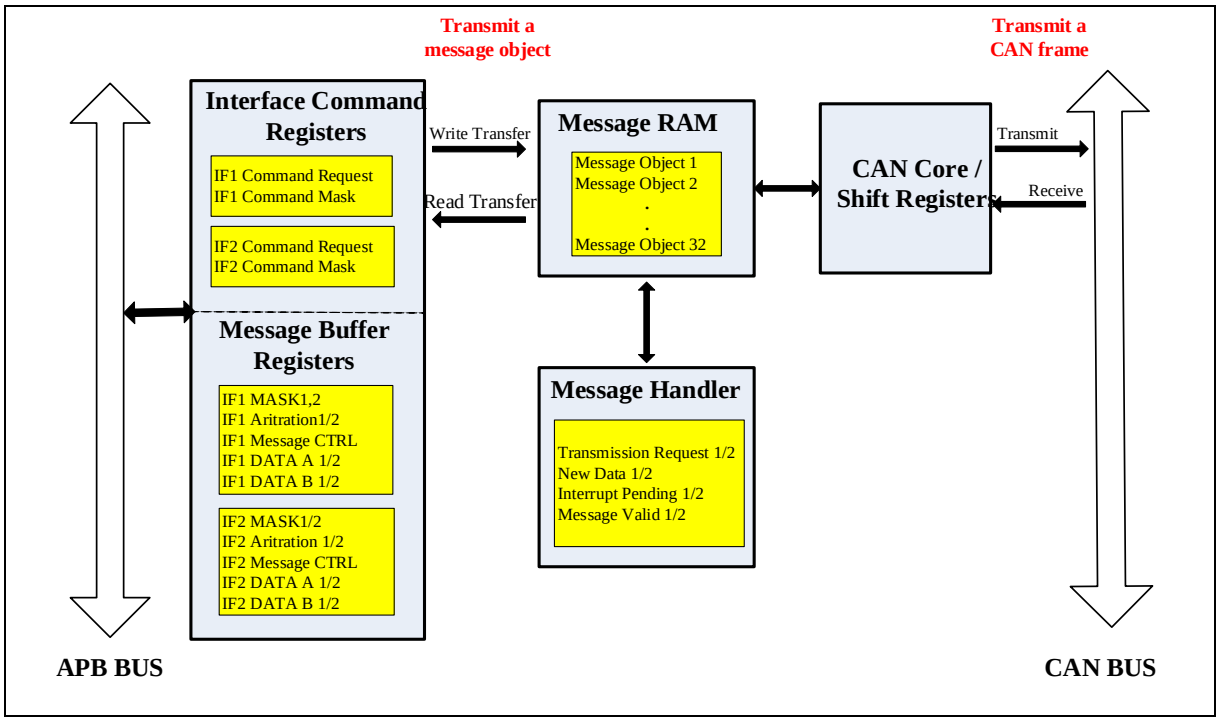


Figure 22.4-4 CAN Message Handling

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 interface 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.

Transfer CAN Message

The C_CAN Module include two Mode : 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 fig. :

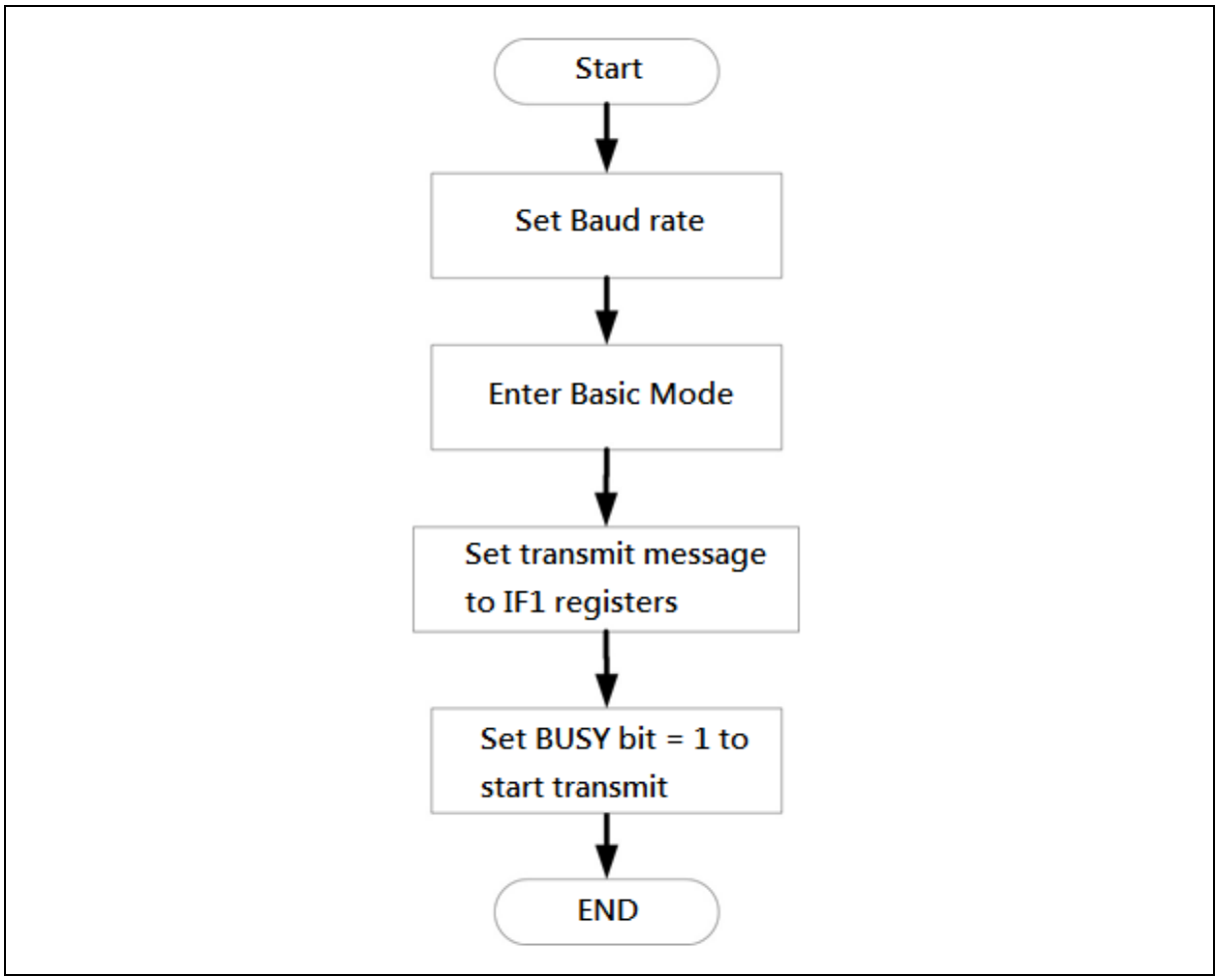


Figure 22.4-5 CAN Transmit Message

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

1. Set CAN bus Baud rate. (Reference sector 22.4.2)
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”.

22.4.4 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 fig. means the flow of polling bitNEWDAT(CAN_IFn_MCON[15]):

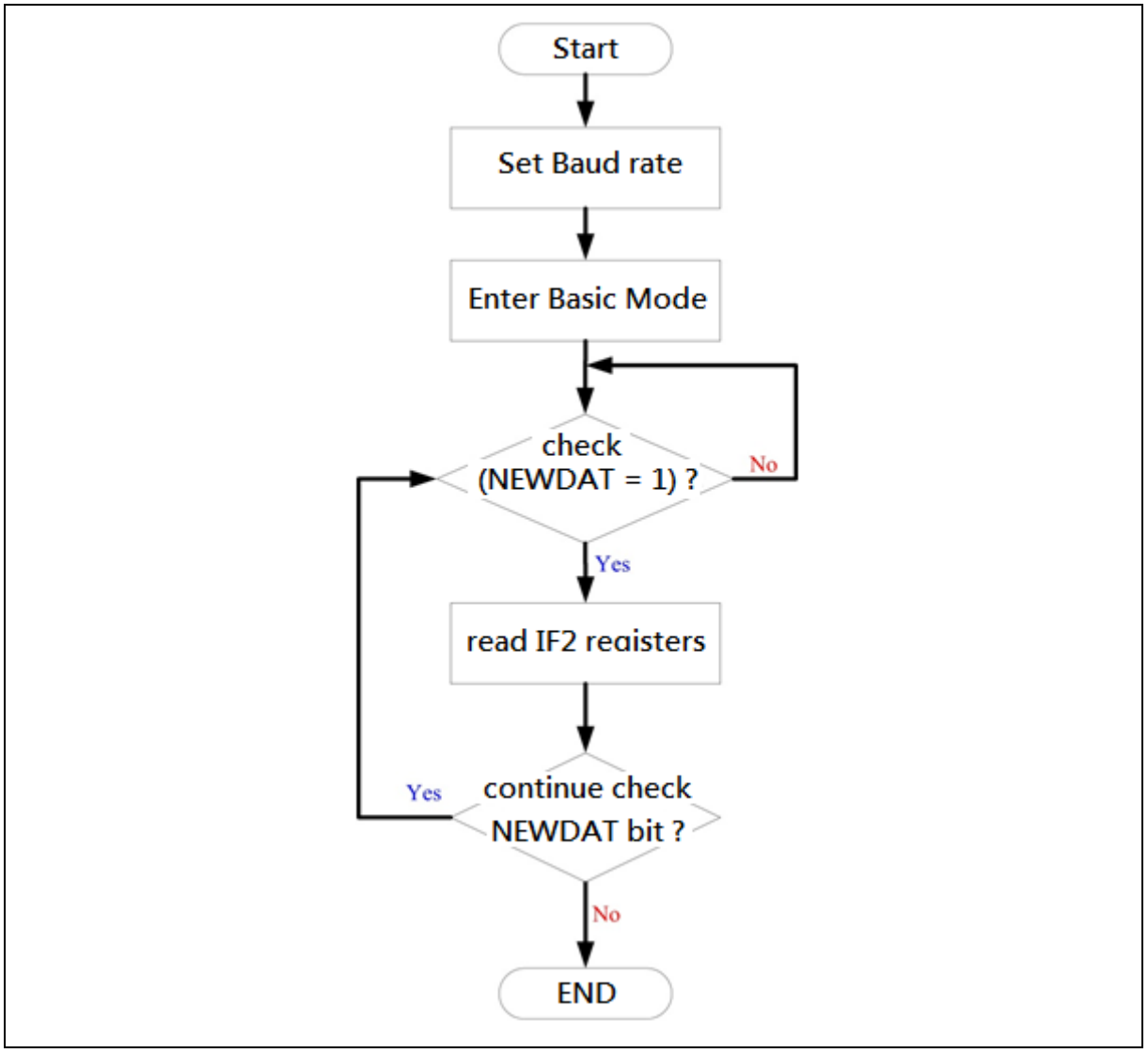


Figure 22.4-6 CAN Receive Message Using Polling Mode

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

1. Set CAN bus Baud rate. (Reference sector 22.4.2)

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 fig. shows the flow that use RX_OK interrupt to receive message:

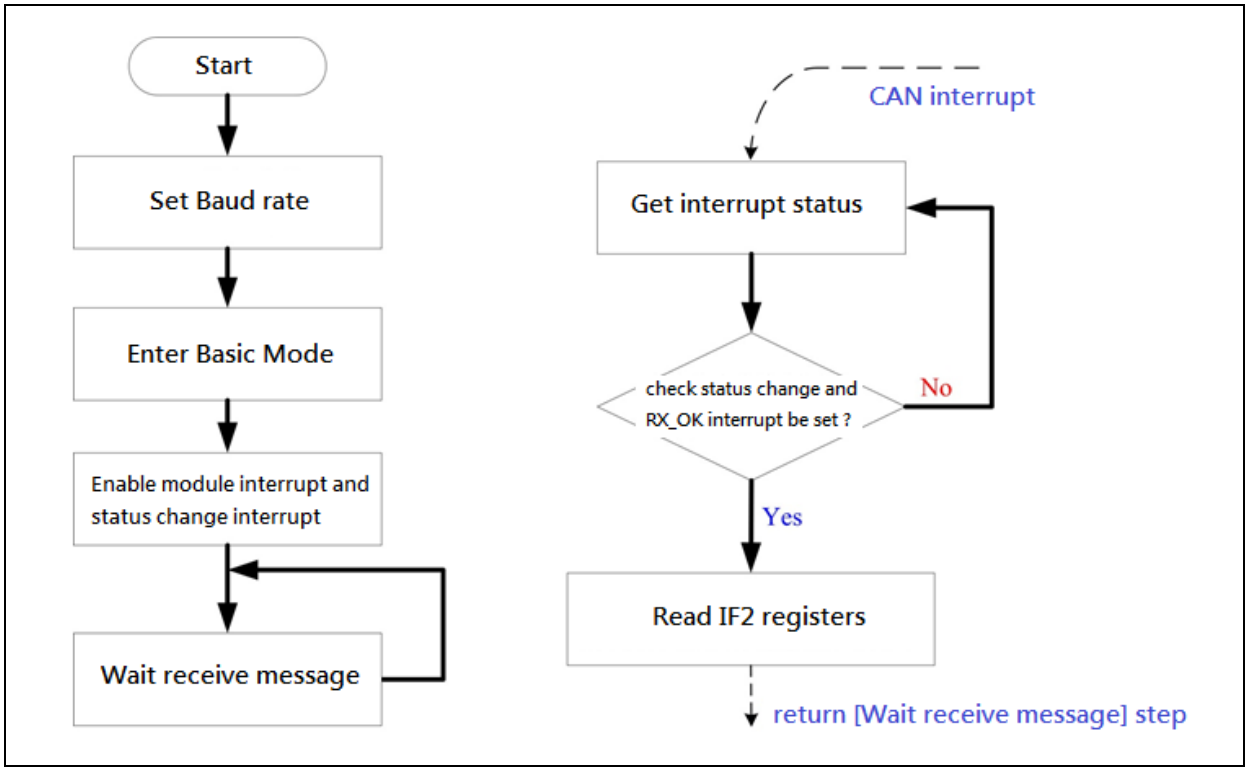


Figure 22.4-7 CAN Receive Message Using Interrupt Mode

In Basic Mode, use RX_OK interrupt to receive message flow as follow:

1. Set CAN bus Baud rate. (Reference sector 22.4.2)
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.

22.4.5 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.

22.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
CAN0_BA = 0xB00A_0000 **CAN1_BA = 0xB80A_1000** **CAN1_BA = 0xB80A_2000** **CAN1_BA = 0xB80A_3000**				
CAN_CON	CANx_BA+0x00	R/W	Control Register	0x0000_0001
CAN_STATUS	CANx_BA+0x04	R/W	Status Register	0x0000_0000
CAN_ERR	CANx_BA+0x08	R	Error Counter	0x0000_0000
CAN_BTIME	CANx_BA+0x0C	R/W	Bit Timing Register	0x0000_2301
CAN_IIDR	CANx_BA+0x10	R	Interrupt Identifier Register	0x0000_0000
CAN_TEST	CANx_BA+0x14	R/W	Test Register	*(1)
CAN_BRPE	CANx_BA+0x18	R/W	BRP Extension Register	0x0000_0000
CAN_IF1_CREQ **CAN_IF2_CREQ**	CANx_BA+0x20 CANx_BA+0x80	R/W	IFn (*2) Command Request Registers	0x0000_0001
CAN_IF1_CMASK **CAN_IF2_CMASK**	CANx_BA+0x24 CANx_BA+0x84	R/W	IFn Command Mask Registers	0x0000_0000
CAN_IF1_MASK1 **CAN_IF2_MASK1**	CANx_BA+0x28 CANx_BA+0x88	R/W	IFn Mask 1 Register	0x0000_FFFF
CAN_IF1_MASK2 **CAN_IF2_MASK2**	CANx_BA+0x2C CANx_BA+0x8C	R/W	IFn Mask 2 Register	0x0000_FFFF
CAN_IF1_ARB1 **CAN_IF2_ARB1**	CANx_BA+0x30 CANx_BA+0x90	R/W	IFn Arbitration 1 Register	0x0000_0000
CAN_IF1_ARB2 **CAN_IF2_ARB2**	CANx_BA+0x34 CANx_BA+0x94	R/W	IFn Arbitration 2 Register	0x0000_0000
CAN_IF1_MCON **CAN_IF2_MCON**	CANx_BA+0x38 CANx_BA+0x98	R/W	IFn Message Control Registers	0x0000_0000
CAN_IF1_DAT_An/ **CAN_IF1_DAT_Bn/** **CAN_IF2_DAT_An/** **CAN_IF2_DAT_Bn/**	CANx_BA+0x3C~40 CANx_BA+0x44~48 CANx_BA+0x9C~A0 CANx_BA+0xA4~A8	R/W	IFn Data An (*3) and Data Bn (*3) Registers eg: CAN_IF1_DAT_A1 = CAN_BA+0x3Ch CAN_IF1_DAT_A2 = CAN_BA+0x40h	0x0000_0000
CAN_TXREQ1 **CAN_TXREQ2**	CANx_BA+0x100 CANx_BA+0x104	R	Transmission Request Registers 1 & 2	0x0000_0000
CAN_NDAT1 **CAN_NDAT2**	CANx_BA+0x120 CANx_BA+0x124	R	New Data Registers 1 & 2	0x0000_0000
CAN_IPND1 **CAN_IPND2**	CANx_BA+0x140 CANx_BA+0x144	R	Interrupt Pending Registers 1 & 2	0x0000_0000
CAN_MVLD1 **CAN_MVLD2**	CANx_BA+0x160 CANx_BA+0x164	R	Message Valid Registers 1 & 2	0x0000_0000
CAN_WU_EN	CANx_BA+0x168	R/W	Wake-up Function Enable	0x0000_0000

CAN_WU_STATUS	CANx_BA+0x16C	R/W	Wake-up Function Status	0x0000_0000

23 FLASH MEMORY INTERFACE (FMI)

23.1 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 SD0/eMMC0 or NAND Flash. The interface controller can support SD0/eMMC0 and NAND-type Flash and the FMI is cooperated with DMAC to provide a fast data transfer between system memory and cards.

23.2 Features

- Supports single DMA channel and address in non-word boundary
- Supports hardware Scatter-Gather function
- Supports 128 Bytes shared buffer for data exchange between system memory and Flash device. (Separate into two 64 bytes ping pong FIFO)
- Supports SD0/eMMC0 Flash device
- Supports SLC and MLC NAND type Flash
- Adjustable NAND page sizes. (2048B+spare area, 4096B+spare area, and 8192B+spare area)
- Supports up to 8-bit/12-bit/24-bit hardware ECC calculation circuit to protect data communication
- Supports programmable NAND timing cycle

23.3 Block Diagram

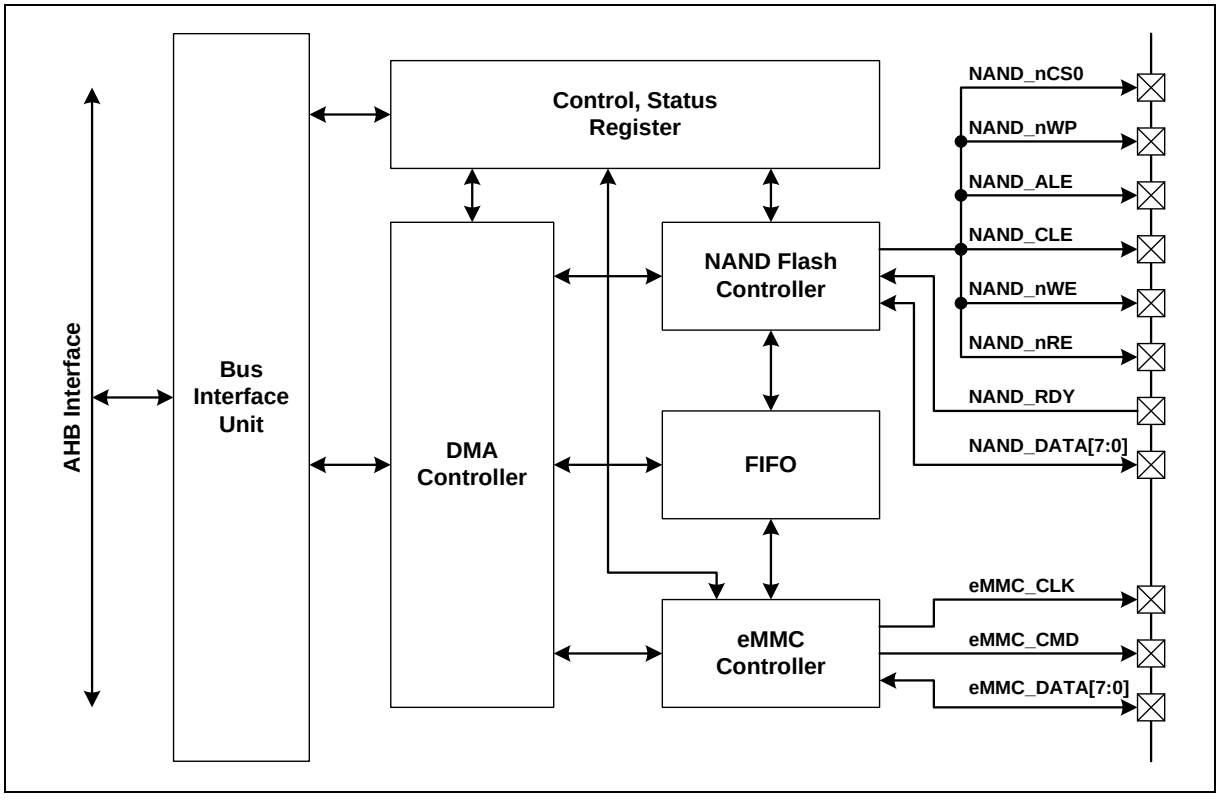


Figure 23.3-1 FMI Block Diagram

23.4 Functional Description

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

23.4.1 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 SD card 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. SD and NAND are not coexisted.

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

1. Set FMI_DMACTL register DMACEN bit and DMARST bit.
2. Polling FMI_DMACTL register DMARST bit until it was cleared.
3. Set FMI_GCTL register GCTLRST bit.
4. Polling FMI_GCTL register GCTLRST bit until it was cleared.

23.4.2 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, 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 8-bit, 12-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_NANDECCEDx register to correct it.

For 2K/4K/8K Page size NAND Flash with BCH algorithm, T can be t8, t12 or t24. Based on the page size and T setting, FMI generate different size of parity data. The number of byte for parity data in different page size and T setting listed in the table shown below. The data arrangement of redundant area is as figure shown below.

It's recommended to choose appropriate T based on NAND Flash page size and redundant area size.

BCH algorithm	Parity (Byte) 512 Page size	Parity (Byte) 2048 Page size	Parity (Byte) 4096 Page size	Parity (Byte) 8192 Page size
BCH T8	15	60	120	240
BCH T12	23	92	184	368
BCH T24	No support	90	180	360

Table 23.4-1 BCH Aolgorithm redundant area usage

For example:

1. page size 2048+64 bytes, BCH T8, the spare area layout is

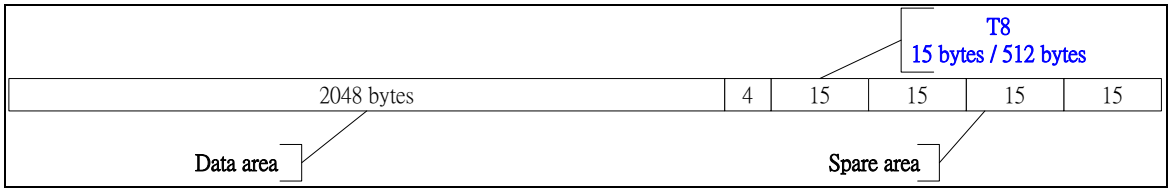


Figure 23.4-1 2048 + 64 Page Size Spare Area Layout

2. Page size 2048+128 bytes, BCH T24, the spare area layout is

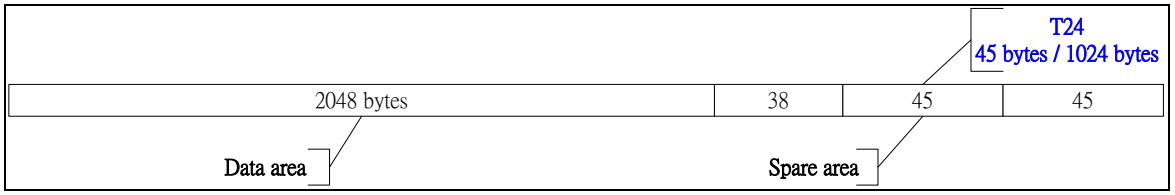


Figure 23.4-2 2048 + 128 Page Size Spare Area Layout

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

23.4.2.1 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. Set the value 0x33333330 into SYS_GPC_MFPL register, and 0x33333333 into SYS_GPC_MFPH register.
3. Set FMI_GCTL 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 bit to 0 to select NAND chip.

23.4.2.2 Reset NAND 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.

23.4.2.3 Identify NAND 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.

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

Table 23.4-2 NAND Access Address Cycle

23.4.2.5 Write NAND 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 and ECC_FLD_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.

23.4.2.6 Read NAND 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. According to the size of redundant area, read out one by one by FMI_NANDDATA register to write redundant area data into FMI_NANDRAn 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).

23.4.2.7 NAND Flash ECC Correction

BCH error correction algorithm can correct up to 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.

23.4.3 **SD/eMMC**

FMI provides an SD/eMMC control interface. This SD/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 SD/eMMC card, user should control the CLKDIV3 register. Detailed procedural rules relate to the card, please refer to "SD Memory Card Specifications Part 1" and "The MultiMediaCard System Specification" , “JEDEC Standard No. 84-A441” and the manufactures eMMC datasheet.

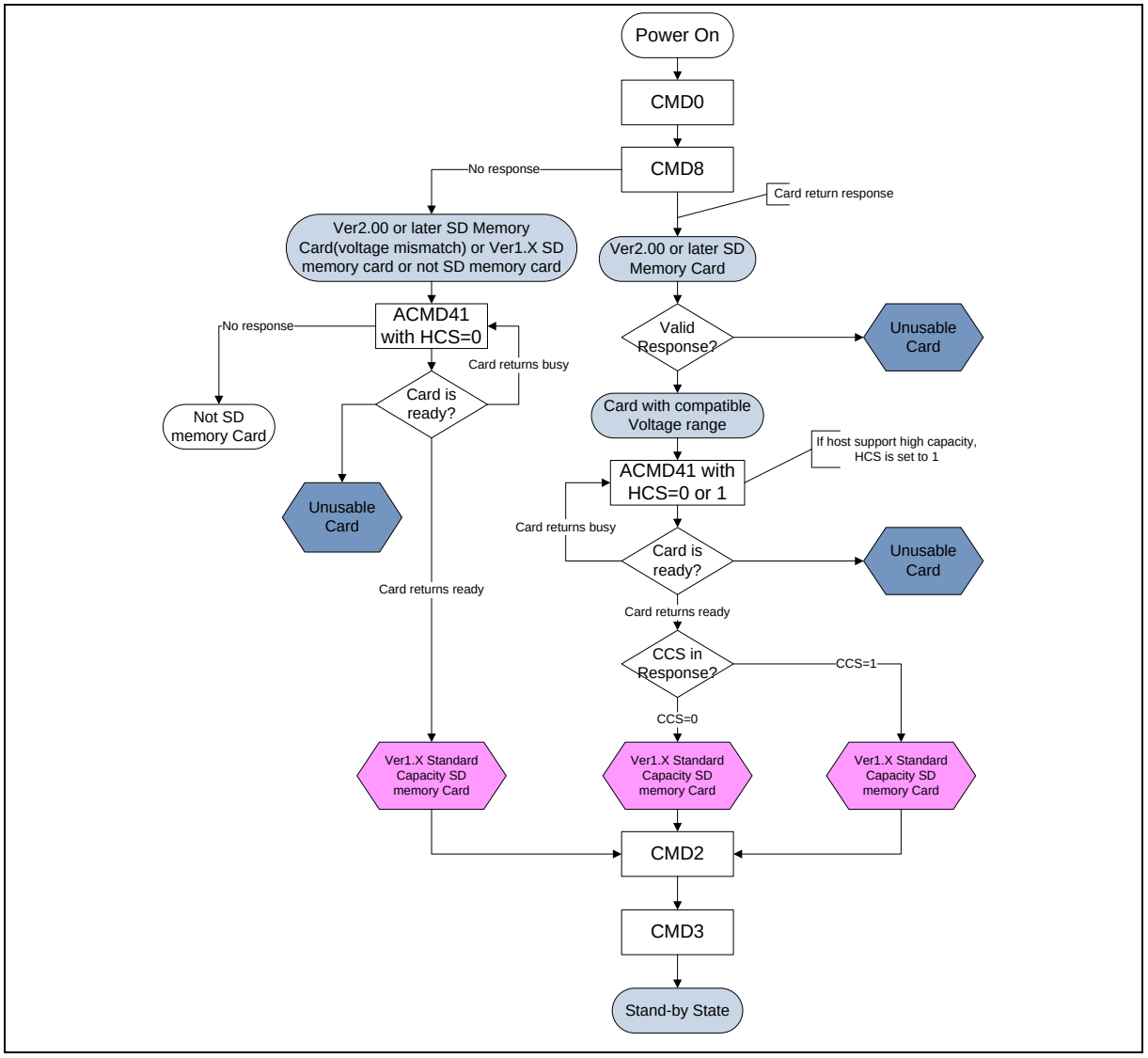


Figure 23.4-3 SD Memory Card State Diagram (Card Identification Mode)

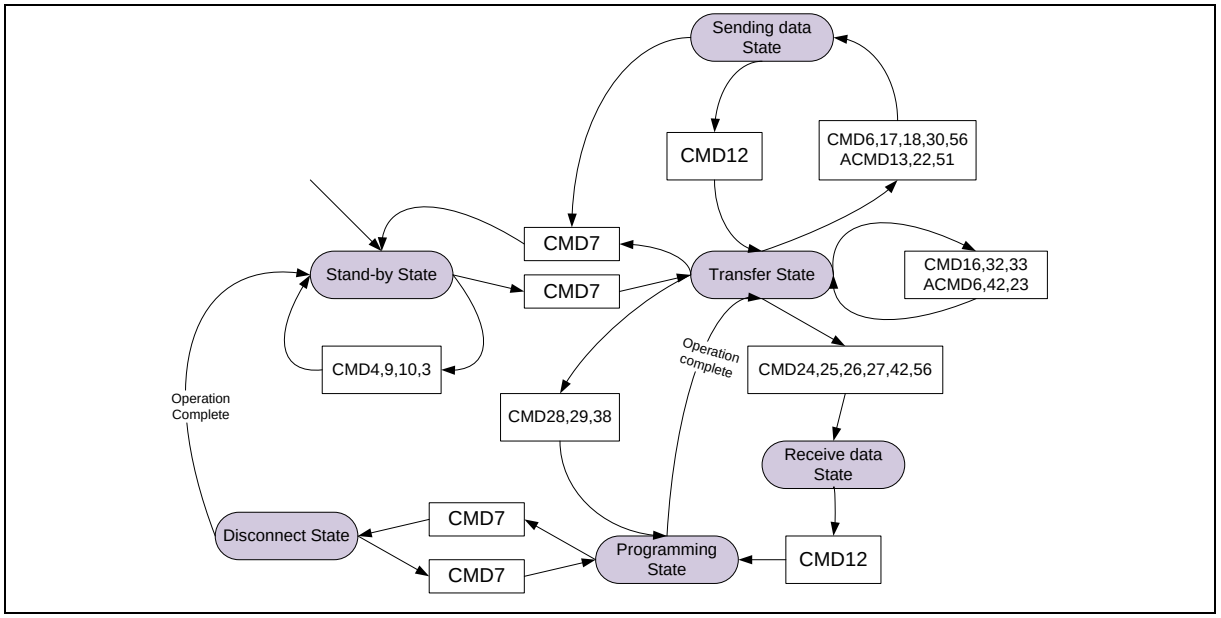


Figure 23.4-4 SD Memory Card State Diagram (Date Transfer Mode)

23.4.3.1 SD/eMMC Initialize

SD/eMMC initialize, please follow the steps below

1. Set CLK_HCLKEN register FMI, NAND and eMMC bit.
2. Set the value 0x66600000 to SYS_GPC_MFPL register, and 0x00060666 to SYS_GPC_MFPH register.
3. Set FMI_GCTL register eMMCEN bit to enable SD/eMMC.
4. Set FMI_EMMCCTL register CTLRST bit.
5. Polling FMI_EMMCCTL register CTLRST bit until it was cleared
6. Set SD/eMMC initial output frequency to 300 kHz, and 1-bit bus for SD/eMMC interface
7. Set FMI_EMMCCTL register CLK74OE bit
8. Polling FMI_EMMCCTL register CLK74OE bit until it was cleared
9. According to devices programming rule to send command to SD/eMMC
10. When device get into Data Transfer Mode, the output frequency can set to suitable clock. Such as 25 MHz. And the bus width is 4-bit mode

23.4.3.2 Send Command

Send command to SD/eMMC, please follow the steps below:

1. Set the argument to FMI_EMMCCMD register.
2. Set command to FMI_EMMCCTL register CMDCODE bit.
3. Set FMI_EMMCCTL register COEN bit to enable command out.
4. Polling FMI_EMMCCTL register COEN bit until it was cleared

23.4.3.3 Get Response

Get response from SD/eMMC, please follow the steps below:

1. Set FMI_EMMCCTL register RIEN bit to enable response in.
2. Set FMI_EMMCCTL register RIEN bit to enable response in

3. Check FMI_EMMCINTSTS register CRC7 bit
4. Get the response from FMI_EMMCRESP0 and FMI_EMMCRESP1 register

23.4.3.4 Read SD/eMMC

SD/eMMC read access, please follow the steps below:

1. Send CMD7 to enter transfer state.
2. Set FMI_EMMCCTL register CLK8OE bit to output 8 clock cycles. Check FMI_EMMCINTSTS register DAT0 bit. Repeat step 2 until SD/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 BLKCNT bit. (255 is the limitation)
7. Send CMD18 for multiple read. (Set 18 to FMI_EMMCCTL register CMDCODE bit)
8. Set FMI_EMMCCTL register COEN, RIEN and DIEN bit to enable command out, response in and data in.
9. Polling DIEN 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 CLK8OE bit to output 8 clock cycles. Check FMI_EMMCINTSTS register DAT0 bit. Repeat step 12 until SD/eMMC is ready.
13. Send CMD7 to idle state.

23.4.3.5 Write SD/eMMC

SD/eMMC wrote access, please follow the steps below:

1. Send CMD7 to enter transfer state.
2. Set FMI_EMMCCTL register CLK8OE bit to output 8 clock cycles. Check FMI_EMMCINTSTS register DAT0 bit. Repeat step 2 until SD/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 BLKCNT bit. (255 is the limitation).
7. Set CMD25 for multiple write. (Set 25 to FMI_EMMCCTL register CMDCODE bit).
8. Set FMI_EMMCCTL register COEN, RIEN and DOEN bit to enable command out, response in and data out.
9. Polling DOEN bit until it was cleared. Or waiting the interrupt (FMI_EMMCINTSTS register BLKDIF bit).
10. Check FMI_EMMCINTSTS register CRCIF bit. If CRC error occurred, the state machine should software reset. (Set FMI_EMMCCTL register CTLRST bit)
11. Send CMD12 to stop transfer.
12. Set FMI_EMMCCTL register CLK8OE bit to output 8 clock cycles. Check FMI_EMMCINTSTS

register DAT0 bit. Repeat step 12 until SD/eMMC is ready.

13. Send CMD7 to Idle state.

23.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
FMI Base Address: **FMI_BA = 0xB001_9000**				
FMI_BUFFERn **n = 0, 1..31**	FMI_BA+0x000+0x4*n	R/W	FMI Embedded Buffer Word n n = 0, 1..31	0x0000_0000
FMI_DMACTL	FMI_BA+0x400	R/W	FMI DMA Control Register	0x0000_0000
FMI_DMASA	FMI_BA+0x408	R/W	FMI DMA Transfer Starting Address Register	0x0000_0000
FMI_DMABCNT	FMI_BA+0x40C	R	FMI DMA Transfer Byte Count Register	0x0000_0000
FMI_DMAINTEN	FMI_BA+0x410	R/W	FMI DMA Interrupt Enable Register	0x0000_0001
FMI_DMAINTSTS	FMI_BA+0x414	R/W	FMI DMA Interrupt Status Register	0x0000_0000
FMI_GCTL	FMI_BA+0x800	R/W	FMI Global Control and Status Register	0x0000_0000
FMI_GINTEN	FMI_BA+0x804	R/W	FMI Global Interrupt Control Register	0x0000_0001
FMI_GINTSTS	FMI_BA+0x808	R/W	FMI Global Interrupt Status Register	0x0000_0000
FMI_EMMCCTL	FMI_BA+0x820	R/W	SD0/eMMC0 Control Register	0x0101_0000
FMI_EMMCCMDARG	FMI_BA+0x824	R/W	SD0/eMMC0 Command Argument Register	0x0000_0000
FMI_EMMCINTEN	FMI_BA+0x828	R/W	SD0/eMMC0 Interrupt Enable Register	0x0000_0000
FMI_EMMCINTSTS	FMI_BA+0x82C	R/W	SD0/eMMC0 Interrupt Status Register	0x00XX_008C
FMI_EMMCRESP0	FMI_BA+0x830	R	SD0/eMMC0 Receiving Response Token Register 0	0x0000_0000
FMI_EMMCRESP1	FMI_BA+0x834	R	SD0/eMMC0 Receiving Response Token Register 1	0x0000_0000
FMI_EMMCBLEN	FMI_BA+0x838	R/W	SD0/eMMC0 Block Length Register	0x0000_01FF
FMI_EMMCTOUT	FMI_BA+0x83C	R/W	SD0/eMMC0 Response/Data-in Time-out Register	0x0000_0000
FMI_EMMCECR	FMI_BA+0x840	R/W	SD0/eMMC0 Extend Control Register	0x0000_0003
FMI_NANDCTL	FMI_BA+0x8A0	R/W	NAND Flash Control Register	0x0288_0090
FMI_NANDTMCTL	FMI_BA+0x8A4	R/W	NAND Flash Timing Control Register	0x0001_0105
FMI_NANDINTEN	FMI_BA+0x8A8	R/W	NAND Flash Interrupt Enable Register	0x0000_0000
FMI_NANDINTSTS	FMI_BA+0x8AC	R/W	NAND Flash Interrupt Status Register	0x00XX_0000
FMI_NANDCMD	FMI_BA+0x8B0	W	NAND Flash Command Port Register	0xXXXX_XXXX
FMI_NANDADDR	FMI_BA+0x8B4	W	NAND Flash Address Port Register	0xXXXX_XXXX
FMI_NANDDATA	FMI_BA+0x8B8	R/W	NAND Flash Data Port Register	0xXXXX_XXXX
FMI_NANDRACTL	FMI_BA+0x8BC	R/W	NAND Flash Redundant Area Control Register	0x0000_0000
FMI_NANDECTL	FMI_BA+0x8C0	R/W	NAND Flash Extend Control Register	0x0000_0000

FMI_NANDECCES0	FMI_BA+0x8D0	R	NAND Flash ECC Error Status 0 Register	0x0000_0000
FMI_NANDECCES1	FMI_BA+0x8D4	R	NAND Flash ECC Error Status 1 Register	0x0000_0000
FMI_NANDECCES2	FMI_BA+0x8D8	R	NAND Flash ECC Error Status 2 Register	0x0000_0000
FMI_NANDECCES3	FMI_BA+0x8DC	R	NAND Flash ECC Error Status 3 Register	0x0000_0000
FMI_NANDPROTA0	FMI_BA+0x8E0	R/W	NAND Flash Protect Region End Address 0 Register	0x0000_0000
FMI_NANDPROTA1	FMI_BA+0x8E4	R/W	NAND Flash Protect Region End Address 1 Register	0x0000_0000
FMI_NANDECCEA0	FMI_BA+0x900	R	NAND Flash ECC Error Byte Address 0 Register	0x0000_0000
FMI_NANDECCEA1	FMI_BA+0x904	R	NAND Flash ECC Error Byte Address 1 Register	0x0000_0000
FMI_NANDECCEA2	FMI_BA+0x908	R	NAND Flash ECC Error Byte Address 2 Register	0x0000_0000
FMI_NANDECCEA3	FMI_BA+0x90C	R	NAND Flash ECC Error Byte Address 3 Register	0x0000_0000
FMI_NANDECCEA4	FMI_BA+0x910	R	NAND Flash ECC Error Byte Address 4 Register	0x0000_0000
FMI_NANDECCEA5	FMI_BA+0x914	R	NAND Flash ECC Error Byte Address 5 Register	0x0000_0000
FMI_NANDECCEA6	FMI_BA+0x918	R	NAND Flash ECC Error Byte Address 6 Register	0x0000_0000
FMI_NANDECCEA7	FMI_BA+0x91C	R	NAND Flash ECC Error Byte Address 7 Register	0x0000_0000
FMI_NANDECCEA8	FMI_BA+0x920	R	NAND Flash ECC Error Byte Address 8 Register	0x0000_0000
FMI_NANDECCEA9	FMI_BA+0x924	R	NAND Flash ECC Error Byte Address 9 Register	0x0000_0000
FMI_NANDECCEA10	FMI_BA+0x928	R	NAND Flash ECC Error Byte Address 10 Register	0x0000_0000
FMI_NANDECCEA11	FMI_BA+0x92C	R	NAND Flash ECC Error Byte Address 11 Register	0x0000_0000
FMI_NANDECCED0	FMI_BA+0x960	R	NAND Flash ECC Error Data Register 0	0x8080_8080
FMI_NANDECCED1	FMI_BA+0x964	R	NAND Flash ECC Error Data Register 1	0x8080_8080
FMI_NANDECCED2	FMI_BA+0x968	R	NAND Flash ECC Error Data Register 2	0x8080_8080
FMI_NANDECCED3	FMI_BA+0x96C	R	NAND Flash ECC Error Data Register 3	0x8080_8080
FMI_NANDECCED4	FMI_BA+0x970	R	NAND Flash ECC Error Data Register 4	0x8080_8080
FMI_NANDECCED5	FMI_BA+0x974	R	NAND Flash ECC Error Data Register 5	0x8080_8080
FMI_NANDRAn **n = 0, 1..117**	FMI_BA+0xA00+0x4*n	R/W	NAND Flash Redundant Area Word n n = 0, 1..117	Undefined

24 SECURE DIGITAL HOST CONTROLLER (SDH)

24.1 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.

24.2 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

24.3 Block Diagram

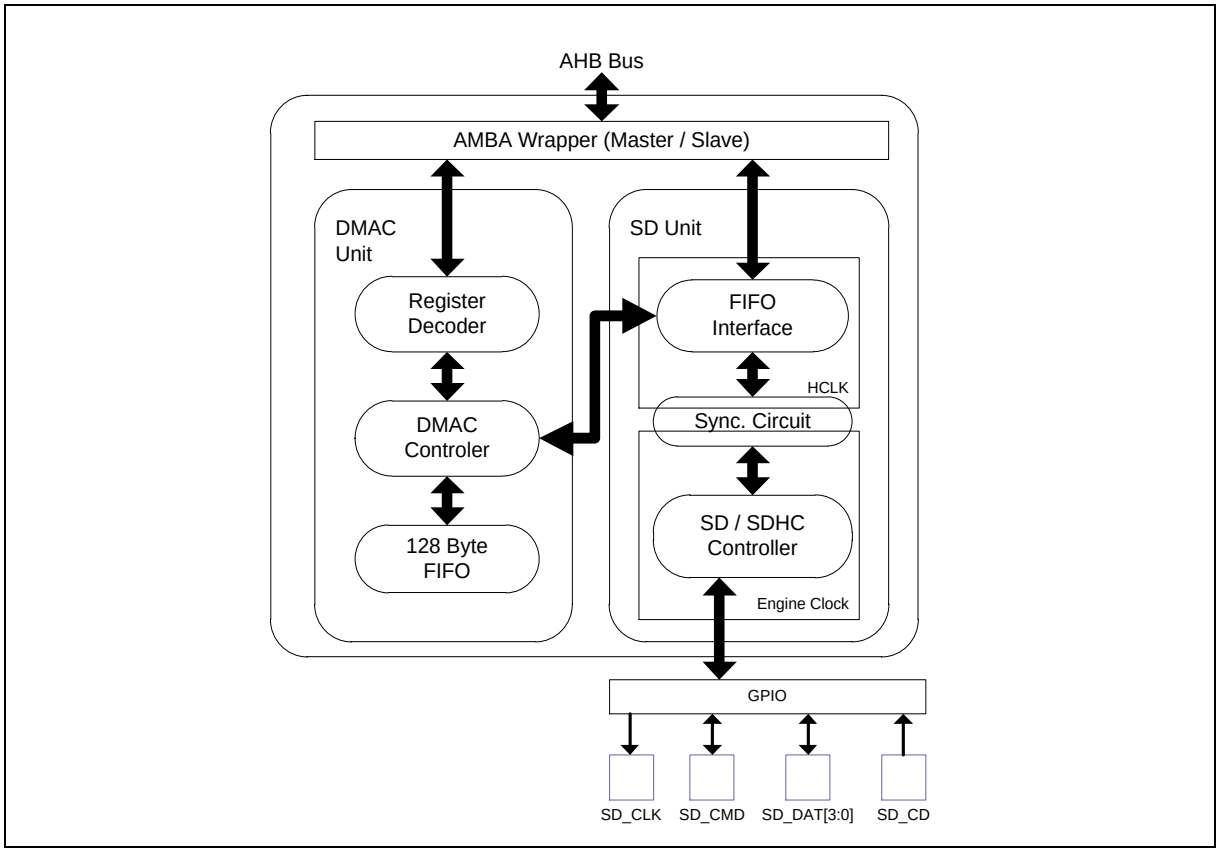


Figure 24.3-1 SD Host Controller Block Diagram

24.4 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.

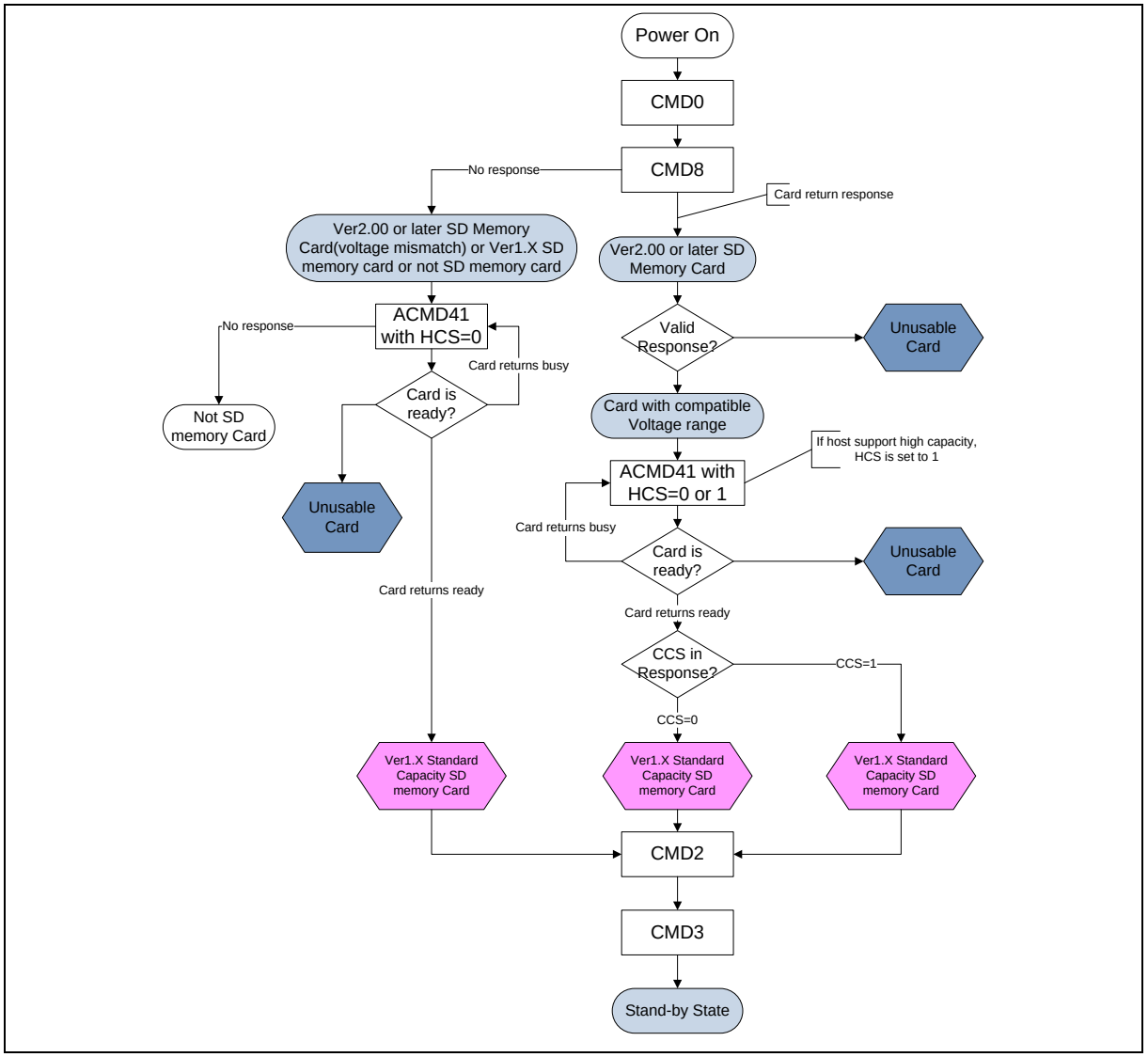


Figure 24.4-1 SD Memory Card State Diagram (Card Identification Mode)

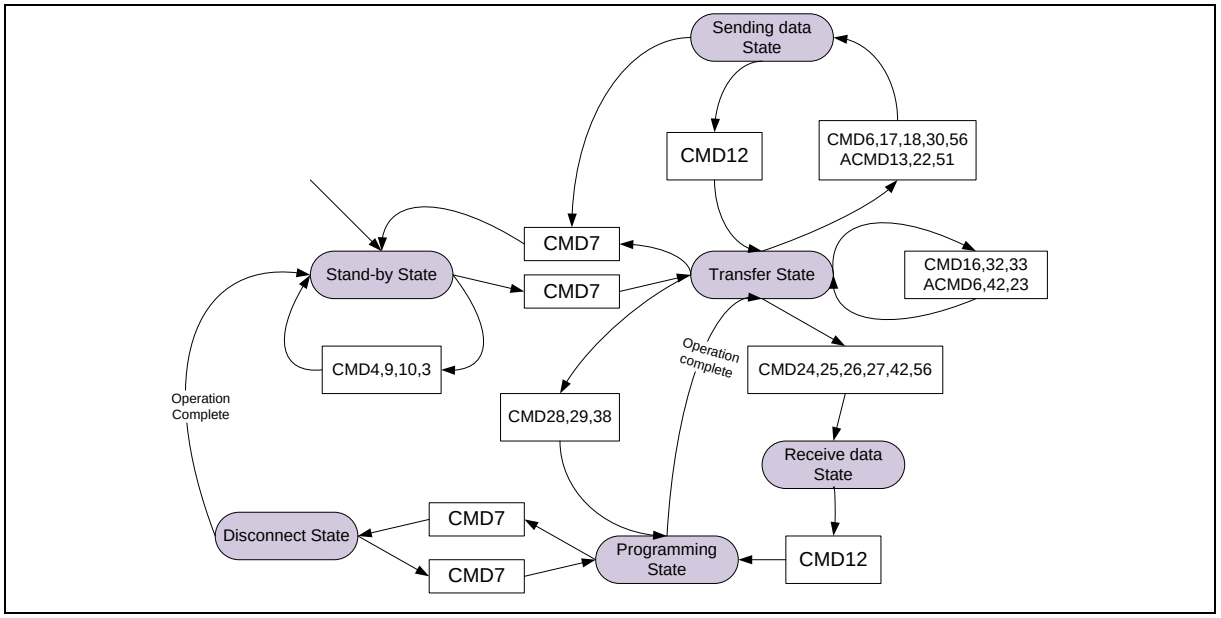


Figure 24.4-2 SD Memory Card State Diagram (Date Transfer Mode)

24.4.1 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 one SD ports – port0. 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 GCTLRST bit.
5. Polling SDH_GCTL register GCTLRST bit until it was cleared.
6. Port 0 only has one set of multiple function pin (GPF0～6). Fill value 0x02222222 into SYS_GPF_MFPL register to select Port 0.
7. Clear SDH_ECTL register PWROFF0 bit to enable the power control.
8. Set SDH_INTEN register CDxSRC bit to select SD card detection source.(DAT3 or GPIO).
9. Set SDH initial output frequency is 300 kHz, bus width is 1-bit mode for Card Identification Mode.
10. Set SDH_CTL register CLK74_OE bit.
11. Polling SDH_CTL register CLK74_OE bit until it was cleared.
12. Follow standard programming rule to send command to SD device.
13. When device get into Data Transfer Mode, the output frequency can set to suitable clock. Such as 25 MHz. And the bus width is 4-bit mode.

24.4.2 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.

24.4.3 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.

24.4.4 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.

24.4.5 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.

24.5 Register Map

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

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

25 CRYPTOGRAPHIC ACCELERATOR

25.1 Overview

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

ECC accelerator support ECC elliptic curve binary field GF(2^m) multiplication and addition operations, and support prime field GF(p) module multiplication、division、addition and subtraction operations。
NUC980 ECC supports up to 571 bits ECC calculation. Support NIST P-192, P-224, P-256, P-384, P-521, B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, and K-571 elliptic curves. It also supports Koblitz secp192k1, secp224k1, and secp256k1 elliptic curves, and Brainpool P256r1, P384r1, and P512r1 elliptic curves.

NUC980 RSA accelerator supports RSA modulus multiplication operation. It supports up to 2048 bits RSA.

25.2 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
- 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
- ECC
  - Supports both prime field GF(p) and binary filed GF(2^m)
  - Supports NIST P-192, P-224, P-256, P-384, and P-521
  - Supports NIST B-163, B-233, B-283, B-409, and B-571
  - Supports NIST K-163, K-233, K-283, K-409, and K-571
  - Supports point multiplication, addition and doubling operations in GF(p) and GF(2^m)
  - Supports modulus division, multiplication, addition and subtraction operations in GF(p)

- RSA
  - Supports both encryption and decryption
  - Supports up to 2048 bits

25.3 Block Diagram

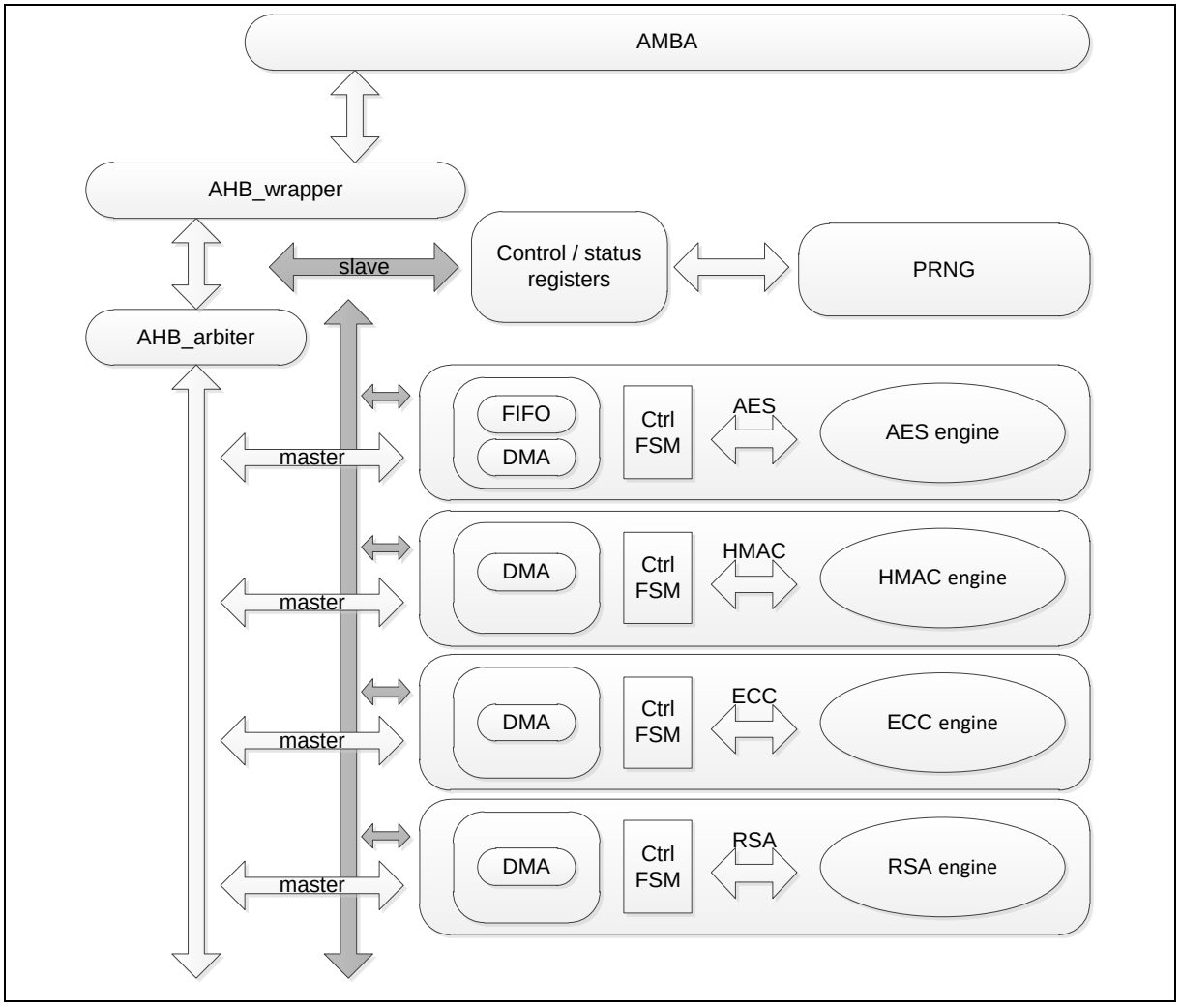


Figure 25.3-1 Cryptographic Accelerator Block Diagram

The cryptographic accelerator includes a secure pseudo random number generator (PRNG) core and supports AES, SHA/HMAC, ECC, and RSA 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.

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 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.

The ECC accelerator is a fully compliant implementation of the prime field GF(p) and binary field GF(2^m) algorithm. The prime field GF(p) supports NIST P-192, P-224, P-256, P-384 and P-521. The binary field GF(2^m) supports NIST B-163, B-233, B-283, B-409, B-571 and NIST K-163, K-233, K-283, K-409 and K-571.

The RSA accelerator is a fully compliant implementation of RSA cryptography with 1024-bit and 2048-bit encryption/decryption.

25.3.1 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, 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 mode. The total channel number can exceed the limit of four DMA channels. The intermediate data from feedback registers (CRPT_AES_FDBCKx) 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_AES0_IVx) 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.

25.4 Functional Description

25.4.1 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]).

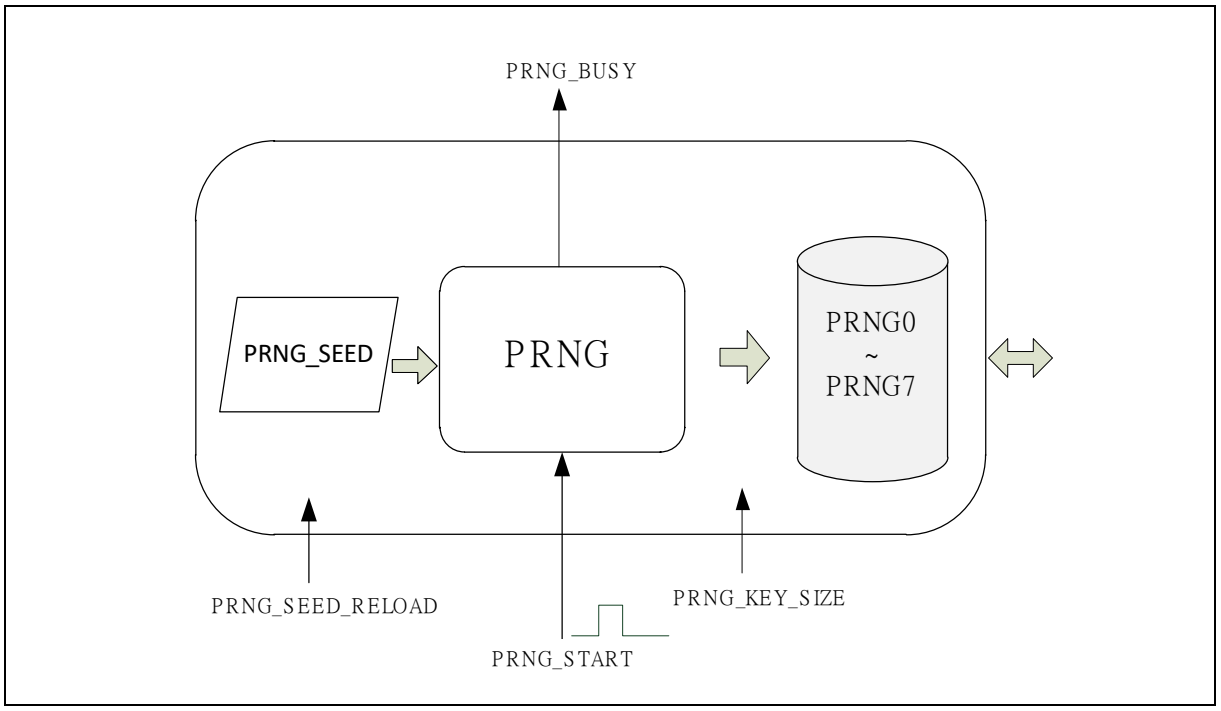


Figure 25.4-1 PRNG Block Diagram

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.

25.4.2 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. NUC980 AES accelerator is fully compliant with AES standards.

Users can refer to the following steps to learn how to use NUC980 AES accelerator.

25.4.2.1 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_AES0_KEY0 ~ CRPT_AES0_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_AES0_KEY0 ~ CRPT_AES0_KEY7 registers.
4. Program initial vectors to registers CRPT_AES0_IV0 ~ CRPT_AES0_IV3. If user selects AES ECB mode, there's no need to write initial vectors.
5. Write DMA source address to register CRPT_AES0_SADDR and write DMA destination address to register CRPT_AES0_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_AES0_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.

25.4.2.2 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_AES0_KEY0 ~ CRPT_AES0_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_AES0_KEY0 ~ CRPT_AES0_KEY7 registers.
4. Program initial vectors to registers CRPT_AES0_IV0 ~ CRPT_AES0_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.

25.4.3 **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 NUC980 SHA accelerator.

25.4.3.1 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).

25.4.3.2 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).

25.4.4 **ECC**

ECC（Elliptic Curve Cryptography）is a famous approach of public-key cryptosystems. It utilizes the algebraic cyclic group characters of elliptic curves over finite field to build cryptographic systems. All points of an elliptic curve will follow the formula of elliptic curve : $y2 \equiv x3+A*x+B \pmod N$ in GF(p) and $y2+x*y \equiv x3+A*x2+B \pmod N$ in $GF(2^m)$.

NIST published several elliptic curves: P-192, P-224, P-256, P-384, P-521, B-163, B-233, B-283, B-409, B571, K-163, K-233, K-283, K-409, and K571. NUC980 ECC can pass ECC test vectors published on NIST official website.

Using NU980 ECC accelerator, the base point and curve parameters of selected elliptic curve must be written to corresponding registers. Users can refer to the following steps to learn how to use NUC980

ECC accelerator.

25.4.4.1 Using Prime Field Elliptic Curves

To use prime field elliptic curves, the base point and curve parameters of selected elliptic curve must be written to corresponding registers:

- Gx： CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17
- Gy： CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17
- p： CRPT_ECC_N_00 ~ CRPT_ECC_N_17
- p-3： CRPT_ECC_A_00 ~ CRPT_ECC_A_17
- b： CRPT_ECC_B_00 ~ CRPT_ECC_B_17
- key length： CURVEM(CRPT_ECC_CTL[31:22])

25.4.4.2 Using Binary Field Elliptic Curves

To use binary field elliptic curves, the base point and curve parameters of selected elliptic curve must be written to corresponding registers:

- Gx： CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17
- Gy： CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17
- P(t)：CRPT_ECC_N_00 ~ CRPT_ECC_N_17
- a： CRPT_ECC_A_00 ~ CRPT_ECC_A_17
- b： CRPT_ECC_B_00 ~ CRPT_ECC_B_17
- key length： CURVEM(CRPT_ECC_CTL[31:22])

25.4.4.3 Point Multiplication

Using NUC980 ECC accelerator for elliptic curve point multiplication as the following steps:

1. Write the base point and curve parameters of selected elliptic curve to the corresponding registers.
2. Write the x and y coordinate of multiplied points to registers CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
3. Write the multiplier to register CRPT_ECC_K_00 ~ CRPT_ECC_K_17.
4. Execute NUC980 ECC point multiplication operation.
5. Get calculation output from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.

25.4.4.4 Generate the Public Key

Using NUC980 ECC accelerator to generate the public key as the following steps:

1. Write the base point and curve parameters of selected elliptic curve to the corresponding registers.
2. Write the private key to register CRPT_ECC_K_00 ~ CRPT_ECC_K_17.
3. Execute NUC980 ECC point multiplication operation.
4. Get the public key from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.

25.4.4.5 Support ECC ECDH

ECDH(Diffie–Hellman key exchange) is a famous key exchange protocol. It supports to establish a share secret over an insecure channel. Using NUC980 ECC accelerator to support ECDH secret generation as the following steps:

1. Write the base point and curve parameters of selected elliptic curve to the corresponding registers.
2. Write your own private key to register CRPT_ECC_K_00 ~ CRPT_ECC_K_17.
3. Write the partner's public key to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
4. Execute NUC980 ECC point multiplication operation.
5. Get the secret from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.

25.4.4.6 Support ECC ECDSA

1. ECDSA (Elliptic Curve Digital Signature Algorithm) is a digital signature generation/verification algorithm based on ECC elliptic curve. Using NUC980 ECC to generate a digital signature as the following steps:
2. Write the base point and curve parameters of selected elliptic curve to the corresponding registers.
3. Select a random integer k and write it to register CRPT_ECC_K_00 ~ CRPT_ECC_K_17.
4. Execute NUC980 ECC point multiplication operation. Keep output coordinate x in register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17. Clear register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17 to 0.
5. Write n of the selected curve to register CRPT_ECC_N_00 ~ CRPT_ECC_N_17.
6. Execute NUC980 ECC point multiplication addition operation. Get the calculation result r from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.
7. Write point (0,1) to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
8. Execute NUC980 ECC module division operation. Get the calculation result K-1 from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.
9. Write r to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17, and write private key d to register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
10. Execute NUC980 ECC module multiplication operation.
11. Calculate the hash value e of messages to be transmitted. Write to register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
12. Execute NUC980 ECC module addition operation.
13. Write K-1 to register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
14. Execute NUC980 ECC module multiplication operation.
15. Get the calculation result s from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.
16. (r, s) is the generated digital signature.

Using NUC980 ECC to verify a digital signature as the following steps:

1. Write the base point and curve parameters of selected elliptic curve to the corresponding registers.
2. Write n of the selected curve to register CRPT_ECC_N_00 ~ CRPT_ECC_N_17.

3. Write signature s to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17. Write 0x1 to register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
4. Execute NUC980 ECC module division operation. Get the calculation result w from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.
5. Write the hash value e to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17. Write w to register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
6. Execute NUC980 ECC module multiplication operation. Get the calculation result u1 from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.
7. Write r to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17. Write w to register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
8. Execute NUC980 ECC module multiplication operation. Get the calculation result u2 from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.
9. Write the base point and curve parameters of selected elliptic curve to the corresponding registers.
10. Write u1 to register CRPT_ECC_K_00 ~ CRPT_ECC_K_17.
11. Execute NUC980 ECC point multiplication operation. Get the calculation result u1 * G from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
12. Write the public key to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
13. Write u2 to register CRPT_ECC_K_00 ~ CRPT_ECC_K_17.
14. Execute NUC980 ECC point multiplication operation. Get the calculation result u2 * Q from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
15. Write u2 * Q to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
16. Write u1 * G to register CRPT_ECC_X2_00 ~ CRPT_ECC_X2_17 and CRPT_ECC_Y2_00 ~ CRPT_ECC_Y2_17.
17. Execute NUC980 ECC point addition operation. Get the calculation result (x', y') from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17 and CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
18. Write x' to register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17, and Write 0x0 to register CRPT_ECC_Y1_00 ~ CRPT_ECC_Y1_17.
19. Execute NUC980 ECC module addition operation. Get the calculation result X1 from register CRPT_ECC_X1_00 ~ CRPT_ECC_X1_17.
20. If X1 is equal to r, the signature verification passed.

25.4.5 **RSA**

RSA is a widely used public-key cryptosystems. In RSA, this asymmetry is based on the practical difficulty of the factorization of the product of two large prime numbers, the "factoring problem". The most time expensive calculation in RSA should be modulus exponentiation calculation. NUC980 RSA accelerator supports hardware accelerating of modulus exponentiation calculation.

25.4.5.1 Generate RSA Private Key and Public Key

The operation flow of RSA key generation as the followings:

1. Choose two distinct huge prime numbers p and q. Calculate p * q to obtain N.

2. Calculate (p – 1) * (q – 1) to obtain r ◦
3. Choose an integer E < r such that E and r are coprime.
4. Determine a d such that d is the modular multiplicative inverse of E.
5. Destroy p and q.

As a result, (N, E) is the RSA public key and (N, d) is the RSA private key.

25.4.5.2 Montgomery domain constant

Using NUC980 RSA accelerator to encrypt/decrypt messages, NUC980 RSA executes M^E % N or M^d % N calculation. N, E and d are obtained in the process of RSA key generation. M represents the messages to be encrypted/decrypted.

In addtion to be given necessary M, E(d) and N, NUC980 RSA accelerator requires the constant of Montgomery domain. The constant of Montgomery domain can be obtained by formula C = $2^{(key\_length+2)*2}$ % N. It requires software to caculate it and write to NUC980 RSA register. The calculation of C is very time expensive. However, once a C is obtained, there's no need to calculate in the next time used unless N was changed.

25.4.5.3 RSA Encryption

The operating flow of using RSA public key to encrypt messages:

1. Write N to register CRPT_RSA_N_00 ~ CRPT_RSA_N_63.
2. Calculate C, and write C to register CRPT_RSA_C_00 ~ CRPT_RSA_C_63.
3. Write E to register CRPT_RSA_E_00 ~ CRPT_RSA_E_63.
4. Write the message to be encrypted to register CRPT_RSA_M_00 ~ CRPT_RSA_M_63.
5. Start NUC980 RSA modulus exponentiation calculation.
6. Get the encrypted messages from register CRPT_RSA_M_00 ~ CRPT_RSA_M_63.

25.4.5.4 RSA Decryption

The operating flow of using RSA private key to decrypt messages:

1. Write N to register CRPT_RSA_N_00 ~ CRPT_RSA_N_63.
2. Calculate C, and write C to register CRPT_RSA_C_00 ~ CRPT_RSA_C_63.
3. Write d to register CRPT_RSA_E_00 ~ CRPT_RSA_E_63.
4. Write the message to be decrypted to register CRPT_RSA_M_00 ~ CRPT_RSA_M_63.
5. Start NUC980 RSA modulus exponentiation calculation.
6. Get the decrypted messages from register CRPT_RSA_M_00 ~ CRPT_RSA_M_63.

25.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
CRYP 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_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_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 Register	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
CRPT_ECC_CTL	CRYP_BA+0x800	R/W	ECC Control Register	0x0000_0000
CRPT_ECC_STS	CRYP_BA+0x804	R	ECC Status Register	0x0000_0000
CRPT_ECC_X1_00	CRYP_BA+0x808	R/W	ECC the X-coordinate Word0 of the First Point	0x0000_0000
CRPT_ECC_X1_01	CRYP_BA+0x80C	R/W	ECC the X-coordinate Word1 of the First Point	0x0000_0000
CRPT_ECC_X1_02	CRYP_BA+0x810	R/W	ECC the X-coordinate Word2 of the First Point	0x0000_0000

CRPT_ECC_X1_03	CRYP_BA+0x814	R/W	ECC the X-coordinate Word3 of the First Point	0x0000_0000
CRPT_ECC_X1_04	CRYP_BA+0x818	R/W	ECC the X-coordinate Word4 of the First Point	0x0000_0000
CRPT_ECC_X1_05	CRYP_BA+0x81C	R/W	ECC the X-coordinate Word5 of the First Point	0x0000_0000
CRPT_ECC_X1_06	CRYP_BA+0x820	R/W	ECC the X-coordinate Word6 of the First Point	0x0000_0000
CRPT_ECC_X1_07	CRYP_BA+0x824	R/W	ECC the X-coordinate Word7 of the First Point	0x0000_0000
CRPT_ECC_X1_08	CRYP_BA+0x828	R/W	ECC the X-coordinate Word8 of the First Point	0x0000_0000
CRPT_ECC_X1_09	CRYP_BA+0x82C	R/W	ECC the X-coordinate Word9 of the First Point	0x0000_0000
CRPT_ECC_X1_10	CRYP_BA+0x830	R/W	ECC the X-coordinate Word10 of the First Point	0x0000_0000
CRPT_ECC_X1_11	CRYP_BA+0x834	R/W	ECC the X-coordinate Word11 of the First Point	0x0000_0000
CRPT_ECC_X1_12	CRYP_BA+0x838	R/W	ECC the X-coordinate Word12 of the First Point	0x0000_0000
CRPT_ECC_X1_13	CRYP_BA+0x83C	R/W	ECC the X-coordinate Word13 of the First Point	0x0000_0000
CRPT_ECC_X1_14	CRYP_BA+0x840	R/W	ECC the X-coordinate Word14 of the First Point	0x0000_0000
CRPT_ECC_X1_15	CRYP_BA+0x844	R/W	ECC the X-coordinate Word15 of the First Point	0x0000_0000
CRPT_ECC_X1_16	CRYP_BA+0x848	R/W	ECC the X-coordinate Word16 of the First Point	0x0000_0000
CRPT_ECC_X1_17	CRYP_BA+0x84C	R/W	ECC the X-coordinate Word17 of the First Point	0x0000_0000
CRPT_ECC_Y1_00	CRYP_BA+0x850	R/W	ECC the Y-coordinate Word0 of the First Point	0x0000_0000
CRPT_ECC_Y1_01	CRYP_BA+0x854	R/W	ECC the Y-coordinate Word1 of the First Point	0x0000_0000
CRPT_ECC_Y1_02	CRYP_BA+0x858	R/W	ECC the Y-coordinate Word2 of the First Point	0x0000_0000
CRPT_ECC_Y1_03	CRYP_BA+0x85C	R/W	ECC the Y-coordinate Word3 of the First Point	0x0000_0000
CRPT_ECC_Y1_04	CRYP_BA+0x860	R/W	ECC the Y-coordinate Word4 of the First Point	0x0000_0000
CRPT_ECC_Y1_05	CRYP_BA+0x864	R/W	ECC the Y-coordinate Word5 of the First Point	0x0000_0000
CRPT_ECC_Y1_06	CRYP_BA+0x868	R/W	ECC the Y-coordinate Word6 of the First Point	0x0000_0000
CRPT_ECC_Y1_07	CRYP_BA+0x86C	R/W	ECC the Y-coordinate Word7 of the First Point	0x0000_0000
CRPT_ECC_Y1_08	CRYP_BA+0x870	R/W	ECC the Y-coordinate Word8 of the First Point	0x0000_0000
CRPT_ECC_Y1_09	CRYP_BA+0x874	R/W	ECC the Y-coordinate Word9 of the First Point	0x0000_0000
CRPT_ECC_Y1_10	CRYP_BA+0x878	R/W	ECC the Y-coordinate Word10 of the First Point	0x0000_0000
CRPT_ECC_Y1_11	CRYP_BA+0x87C	R/W	ECC the Y-coordinate Word11 of the First Point	0x0000_0000
CRPT_ECC_Y1_12	CRYP_BA+0x880	R/W	ECC the Y-coordinate Word12 of the First Point	0x0000_0000
CRPT_ECC_Y1_13	CRYP_BA+0x884	R/W	ECC the Y-coordinate Word13 of the First Point	0x0000_0000
CRPT_ECC_Y1_14	CRYP_BA+0x888	R/W	ECC the Y-coordinate Word14 of the First Point	0x0000_0000
CRPT_ECC_Y1_15	CRYP_BA+0x88C	R/W	ECC the Y-coordinate Word15 of the First Point	0x0000_0000
CRPT_ECC_Y1_16	CRYP_BA+0x890	R/W	ECC the Y-coordinate Word16 of the First Point	0x0000_0000
CRPT_ECC_Y1_17	CRYP_BA+0x894	R/W	ECC the Y-coordinate Word17 of the First Point	0x0000_0000
CRPT_ECC_X2_00	CRYP_BA+0x898	R/W	ECC the X-coordinate Word0 of the Second Point	0x0000_0000
CRPT_ECC_X2_01	CRYP_BA+0x89C	R/W	ECC the X-coordinate Word1 of the Second Point	0x0000_0000

CRPT_ECC_X2_02	CRYP_BA+0x8A0	R/W	ECC the X-coordinate Word2 of the Second Point	0x0000_0000
CRPT_ECC_X2_03	CRYP_BA+0x8A4	R/W	ECC the X-coordinate Word3 of the Second Point	0x0000_0000
CRPT_ECC_X2_04	CRYP_BA+0x8A8	R/W	ECC the X-coordinate Word4 of the Second Point	0x0000_0000
CRPT_ECC_X2_05	CRYP_BA+0x8AC	R/W	ECC the X-coordinate Word5 of the Second Point	0x0000_0000
CRPT_ECC_X2_06	CRYP_BA+0x8B0	R/W	ECC the X-coordinate Word6 of the Second Point	0x0000_0000
CRPT_ECC_X2_07	CRYP_BA+0x8B4	R/W	ECC the X-coordinate Word7 of the Second Point	0x0000_0000
CRPT_ECC_X2_08	CRYP_BA+0x8B8	R/W	ECC the X-coordinate Word8 of the Second Point	0x0000_0000
CRPT_ECC_X2_09	CRYP_BA+0x8BC	R/W	ECC the X-coordinate Word9 of the Second Point	0x0000_0000
CRPT_ECC_X2_10	CRYP_BA+0x8C0	R/W	ECC the X-coordinate Word10 of the Second Point	0x0000_0000
CRPT_ECC_X2_11	CRYP_BA+0x8C4	R/W	ECC the X-coordinate Word11 of the Second Point	0x0000_0000
CRPT_ECC_X2_12	CRYP_BA+0x8C8	R/W	ECC the X-coordinate Word12 of the Second Point	0x0000_0000
CRPT_ECC_X2_13	CRYP_BA+0x8CC	R/W	ECC the X-coordinate Word13 of the Second Point	0x0000_0000
CRPT_ECC_X2_14	CRYP_BA+0x8D0	R/W	ECC the X-coordinate Word14 of the Second Point	0x0000_0000
CRPT_ECC_X2_15	CRYP_BA+0x8D4	R/W	ECC the X-coordinate Word15 of the Second Point	0x0000_0000
CRPT_ECC_X2_16	CRYP_BA+0x8D8	R/W	ECC the X-coordinate Word16 of the Second Point	0x0000_0000
CRPT_ECC_X2_17	CRYP_BA+0x8DC	R/W	ECC the X-coordinate Word17 of the Second Point	0x0000_0000
CRPT_ECC_Y2_00	CRYP_BA+0x8E0	R/W	ECC the Y-coordinate Word0 of the Second Point	0x0000_0000
CRPT_ECC_Y2_01	CRYP_BA+0x8E4	R/W	ECC the Y-coordinate Word1 of the Second Point	0x0000_0000
CRPT_ECC_Y2_02	CRYP_BA+0x8E8	R/W	ECC the Y-coordinate Word2 of the Second Point	0x0000_0000
CRPT_ECC_Y2_03	CRYP_BA+0x8EC	R/W	ECC the Y-coordinate Word3 of the Second Point	0x0000_0000
CRPT_ECC_Y2_04	CRYP_BA+0x8F0	R/W	ECC the Y-coordinate Word4 of the Second Point	0x0000_0000
CRPT_ECC_Y2_05	CRYP_BA+0x8F4	R/W	ECC the Y-coordinate Word5 of the Second Point	0x0000_0000
CRPT_ECC_Y2_06	CRYP_BA+0x8F8	R/W	ECC the Y-coordinate Word6 of the Second Point	0x0000_0000
CRPT_ECC_Y2_07	CRYP_BA+0x8FC	R/W	ECC the Y-coordinate Word7 of the Second Point	0x0000_0000
CRPT_ECC_Y2_08	CRYP_BA+0x900	R/W	ECC the Y-coordinate Word8 of the Second Point	0x0000_0000
CRPT_ECC_Y2_09	CRYP_BA+0x904	R/W	ECC the Y-coordinate Word9 of the Second Point	0x0000_0000
CRPT_ECC_Y2_10	CRYP_BA+0x908	R/W	ECC the Y-coordinate Word10 of the Second Point	0x0000_0000
CRPT_ECC_Y2_11	CRYP_BA+0x90C	R/W	ECC the Y-coordinate Word11 of the Second Point	0x0000_0000
CRPT_ECC_Y2_12	CRYP_BA+0x910	R/W	ECC the Y-coordinate Word12 of the Second Point	0x0000_0000
CRPT_ECC_Y2_13	CRYP_BA+0x914	R/W	ECC the Y-coordinate Word13 of the Second Point	0x0000_0000
CRPT_ECC_Y2_14	CRYP_BA+0x918	R/W	ECC the Y-coordinate Word14 of the Second Point	0x0000_0000
CRPT_ECC_Y2_15	CRYP_BA+0x91C	R/W	ECC the Y-coordinate Word15 of the Second Point	0x0000_0000
CRPT_ECC_Y2_16	CRYP_BA+0x920	R/W	ECC the Y-coordinate Word16 of the Second Point	0x0000_0000
CRPT_ECC_Y2_17	CRYP_BA+0x924	R/W	ECC the Y-coordinate Word17 of the Second Point	0x0000_0000
CRPT_ECC_A_00	CRYP_BA+0x928	R/W	ECC the Parameter CURVEA Word0 of Elliptic Curve	0x0000_0000

CRPT_ECC_A_01	CRYP_BA+0x92C	R/W	ECC the Parameter CURVEA Word1 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_02	CRYP_BA+0x930	R/W	ECC the Parameter CURVEA Word2 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_03	CRYP_BA+0x934	R/W	ECC the Parameter CURVEA Word3 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_04	CRYP_BA+0x938	R/W	ECC the Parameter CURVEA Word4 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_05	CRYP_BA+0x93C	R/W	ECC the Parameter CURVEA Word5 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_06	CRYP_BA+0x940	R/W	ECC the Parameter CURVEA Word6 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_07	CRYP_BA+0x944	R/W	ECC the Parameter CURVEA Word7 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_08	CRYP_BA+0x948	R/W	ECC the Parameter CURVEA Word8 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_09	CRYP_BA+0x94C	R/W	ECC the Parameter CURVEA Word9 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_10	CRYP_BA+0x950	R/W	ECC the Parameter CURVEA Word10 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_11	CRYP_BA+0x954	R/W	ECC the Parameter CURVEA Word11 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_12	CRYP_BA+0x958	R/W	ECC the Parameter CURVEA Word12 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_13	CRYP_BA+0x95C	R/W	ECC the Parameter CURVEA Word13 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_14	CRYP_BA+0x960	R/W	ECC the Parameter CURVEA Word14 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_15	CRYP_BA+0x964	R/W	ECC the Parameter CURVEA Word15 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_16	CRYP_BA+0x968	R/W	ECC the Parameter CURVEA Word16 of Elliptic Curve	0x0000_0000
CRPT_ECC_A_17	CRYP_BA+0x96C	R/W	ECC the Parameter CURVEA Word17 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_00	CRYP_BA+0x970	R/W	ECC the Parameter CURVEB Word0 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_01	CRYP_BA+0x974	R/W	ECC the Parameter CURVEB Word1 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_02	CRYP_BA+0x978	R/W	ECC the Parameter CURVEB Word2 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_03	CRYP_BA+0x97C	R/W	ECC the Parameter CURVEB Word3 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_04	CRYP_BA+0x980	R/W	ECC the Parameter CURVEB Word4 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_05	CRYP_BA+0x984	R/W	ECC the Parameter CURVEB Word5 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_06	CRYP_BA+0x988	R/W	ECC the Parameter CURVEB Word6 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_07	CRYP_BA+0x98C	R/W	ECC the Parameter CURVEB Word7 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_08	CRYP_BA+0x990	R/W	ECC the Parameter CURVEB Word8 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_09	CRYP_BA+0x994	R/W	ECC the Parameter CURVEB Word9 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_10	CRYP_BA+0x998	R/W	ECC the Parameter CURVEB Word10 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_11	CRYP_BA+0x99C	R/W	ECC the Parameter CURVEB Word11 of Elliptic Curve	0x0000_0000

CRPT_ECC_B_12	CRYP_BA+0x9A0	R/W	ECC the Parameter CURVEB Word12 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_13	CRYP_BA+0x9A4	R/W	ECC the Parameter CURVEB Word13 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_14	CRYP_BA+0x9A8	R/W	ECC the Parameter CURVEB Word14 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_15	CRYP_BA+0x9AC	R/W	ECC the Parameter CURVEB Word15 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_16	CRYP_BA+0x9B0	R/W	ECC the Parameter CURVEB Word16 of Elliptic Curve	0x0000_0000
CRPT_ECC_B_17	CRYP_BA+0x9B4	R/W	ECC the Parameter CURVEB Word17 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_00	CRYP_BA+0x9B8	R/W	ECC the Parameter CURVEN Word0 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_01	CRYP_BA+0x9BC	R/W	ECC the Parameter CURVEN Word1 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_02	CRYP_BA+0x9C0	R/W	ECC the Parameter CURVEN Word2 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_03	CRYP_BA+0x9C4	R/W	ECC the Parameter CURVEN Word3 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_04	CRYP_BA+0x9C8	R/W	ECC the Parameter CURVEN Word4 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_05	CRYP_BA+0x9CC	R/W	ECC the Parameter CURVEN Word5 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_06	CRYP_BA+0x9D0	R/W	ECC the Parameter CURVEN Word6 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_07	CRYP_BA+0x9D4	R/W	ECC the Parameter CURVEN Word7 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_08	CRYP_BA+0x9D8	R/W	ECC the Parameter CURVEN Word8 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_09	CRYP_BA+0x9DC	R/W	ECC the Parameter CURVEN Word9 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_10	CRYP_BA+0x9E0	R/W	ECC the Parameter CURVEN Word10 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_11	CRYP_BA+0x9E4	R/W	ECC the Parameter CURVEN Word11 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_12	CRYP_BA+0x9E8	R/W	ECC the Parameter CURVEN Word12 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_13	CRYP_BA+0x9EC	R/W	ECC the Parameter CURVEN Word13 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_14	CRYP_BA+0x9F0	R/W	ECC the Parameter CURVEN Word14 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_15	CRYP_BA+0x9F4	R/W	ECC the Parameter CURVEN Word15 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_16	CRYP_BA+0x9F8	R/W	ECC the Parameter CURVEN Word16 of Elliptic Curve	0x0000_0000
CRPT_ECC_N_17	CRYP_BA+0x9FC	R/W	ECC the Parameter CURVEN Word17 of Elliptic Curve	0x0000_0000
CRPT_ECC_K_00	CRYP_BA+0xA00	W	ECC the Scalar SCALARK Word0 of Point Multiplication	0x0000_0000
CRPT_ECC_K_01	CRYP_BA+0xA04	W	ECC the Scalar SCALARK Word1 of Point Multiplication	0x0000_0000

CRPT_ECC_K_02	CRYP_BA+0xA08	W	ECC the Scalar SCALARK Word2 of Point Multiplication	0x0000_0000
CRPT_ECC_K_03	CRYP_BA+0xA0C	W	ECC the Scalar SCALARK Word3 of Point Multiplication	0x0000_0000
CRPT_ECC_K_04	CRYP_BA+0xA10	W	ECC the Scalar SCALARK Word4 of Point Multiplication	0x0000_0000
CRPT_ECC_K_05	CRYP_BA+0xA14	W	ECC the Scalar SCALARK Word5 of Point Multiplication	0x0000_0000
CRPT_ECC_K_06	CRYP_BA+0xA18	W	ECC the Scalar SCALARK Word6 of Point Multiplication	0x0000_0000
CRPT_ECC_K_07	CRYP_BA+0xA1C	W	ECC the Scalar SCALARK Word7 of Point Multiplication	0x0000_0000
CRPT_ECC_K_08	CRYP_BA+0xA20	W	ECC the Scalar SCALARK Word8 of Point Multiplication	0x0000_0000
CRPT_ECC_K_09	CRYP_BA+0xA24	W	ECC the Scalar SCALARK Word9 of Point Multiplication	0x0000_0000
CRPT_ECC_K_10	CRYP_BA+0xA28	W	ECC the Scalar SCALARK Word10 of Point Multiplication	0x0000_0000
CRPT_ECC_K_11	CRYP_BA+0xA2C	W	ECC the Scalar SCALARK Word11 of Point Multiplication	0x0000_0000
CRPT_ECC_K_12	CRYP_BA+0xA30	W	ECC the Scalar SCALARK Word12 of Point Multiplication	0x0000_0000
CRPT_ECC_K_13	CRYP_BA+0xA34	W	ECC the Scalar SCALARK Word13 of Point Multiplication	0x0000_0000
CRPT_ECC_K_14	CRYP_BA+0xA38	W	ECC the Scalar SCALARK Word14 of Point Multiplication	0x0000_0000
CRPT_ECC_K_15	CRYP_BA+0xA3C	W	ECC the Scalar SCALARK Word15 of Point Multiplication	0x0000_0000
CRPT_ECC_K_16	CRYP_BA+0xA40	W	ECC the Scalar SCALARK Word16 of Point Multiplication	0x0000_0000
CRPT_ECC_K_17	CRYP_BA+0xA44	W	ECC the Scalar SCALARK Word17 of Point Multiplication	0x0000_0000
CRPT_ECC_SADDR	CRYP_BA+0xA48	R/W	ECC DMA Source Address Register	0x0000_0000
CRPT_ECC_DADDR	CRYP_BA+0xA4C	R/W	ECC DMA Destination Address Register	0x0000_0000
CRPT_ECC_STARTREG	CRYP_BA+0xA50	R/W	ECC Starting Address of Updated Registers	0x0000_0000
CRPT_ECC_WORDCNT	CRYP_BA+0xA54	R/W	ECC DMA Word Count	0x0000_0000
CRPT_RSA_CTL	CRYP_BA+0x1000	R/W	RSA Control Register	0x0000_0000
CRPT_RSA_STS	CRYP_BA+0x1004	R	RSA Status Register	0x0000_0000
CRPT_RSA_M_i **i=0,1..63**	CRYP_BA+0x1008+ 0x4*i	R/W	RSA the Base of Exponentiation Word i	0x0000_0000
CRPT_RSA_E_i **i=0,1..63**	CRYP_BA+0x1208+ 0x4*i	W	RSA the Exponent of Exponentiation Word i	0x0000_0000

CRPT_RSA_N_i **i=0,1..63**	CRYP_BA+0x1408+ 0x4*i	R/W	RSA the Base of Modulus Operation Word i	0x0000_0000
CRPT_RSA_C_i **i=0,1..63**	CRYP_BA+0x1608+ 0x4*i	R/W	RSA the Constant Value of Montgomery Domain Word i	0x0000_0000
CRPT_RSA_SADDR	CRYP_BA+0x1808	R/W	RSA DMA Source Address Register	0x0000_0000
CRPT_RSA_DADDR	CRYP_BA+0x180C	R/W	RSA DMA Destination Address Register	0x0000_0000
CRPT_RSA_STARTREG	CRYP_BA+0x1810	R/W	RSA Starting Address of Updated Registers	0x0000_0000
CRPT_RSA_WORDCNT	CRYP_BA+0x1814	R/W	RSA DMA Word Count	0x0000_0000

26 CAPTURE SENSOR INTERFACE CONTROLLER

26.1 Overview

NUC980 series has two CAP controls. 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.

26.2 Features

- Supports 2 CAP controller, CAP0 and CAP1
- 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/sepia/posterization color effect
- Supports two independent capture interfaces

26.3 Block Diagram

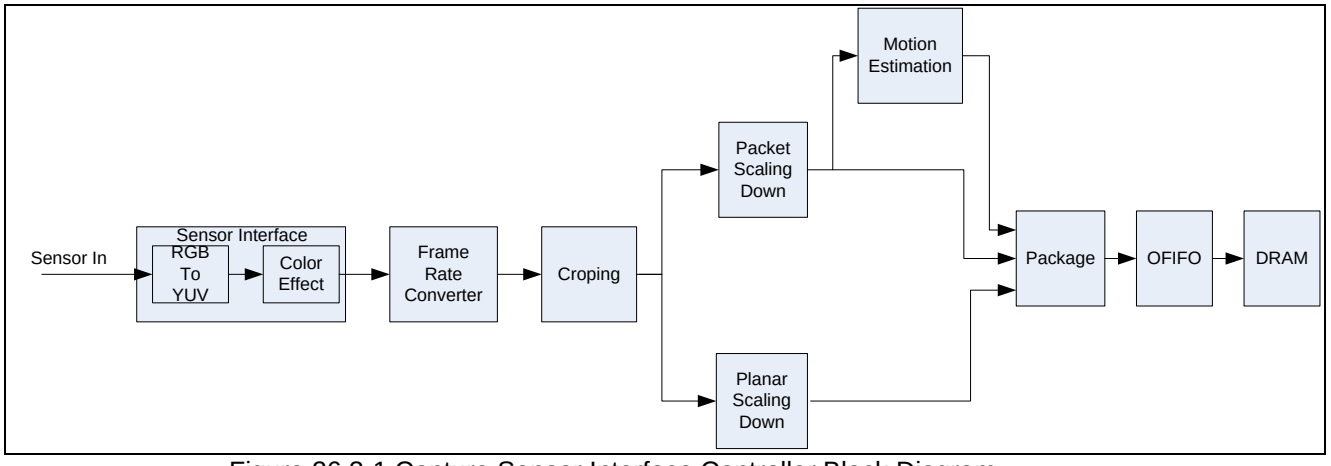


Figure 26.3-1 Capture Sensor Interface Controller Block Diagram

26.4 Functional Description

26.4.1 Basic Configuration

The CAP0 peripheral clock can be enabled in CAP0(HCLKEN1[26]) and SENSOR(HCLKEN[27]). The CAP0 engine clock source is selected by SENSOR0_S (CLKDIV3[23:16]) and CAP0 engine clock divider is determined by SENSOR0_N (CLKDIV3[27:24]).

The CAP1 peripheral clock can be enabled in CAP1(HCLKEN1[31]) and SENSOR(HCLKEN[27]). The CAP1 engine clock source is selected by SENSOR1_S (CLKDIV2[23:16]) and CAP1 engine clock divider is determined by SENSOR1_N (CLKDIV2[27:24]).

26.4.2 Image Capture Flow Chart

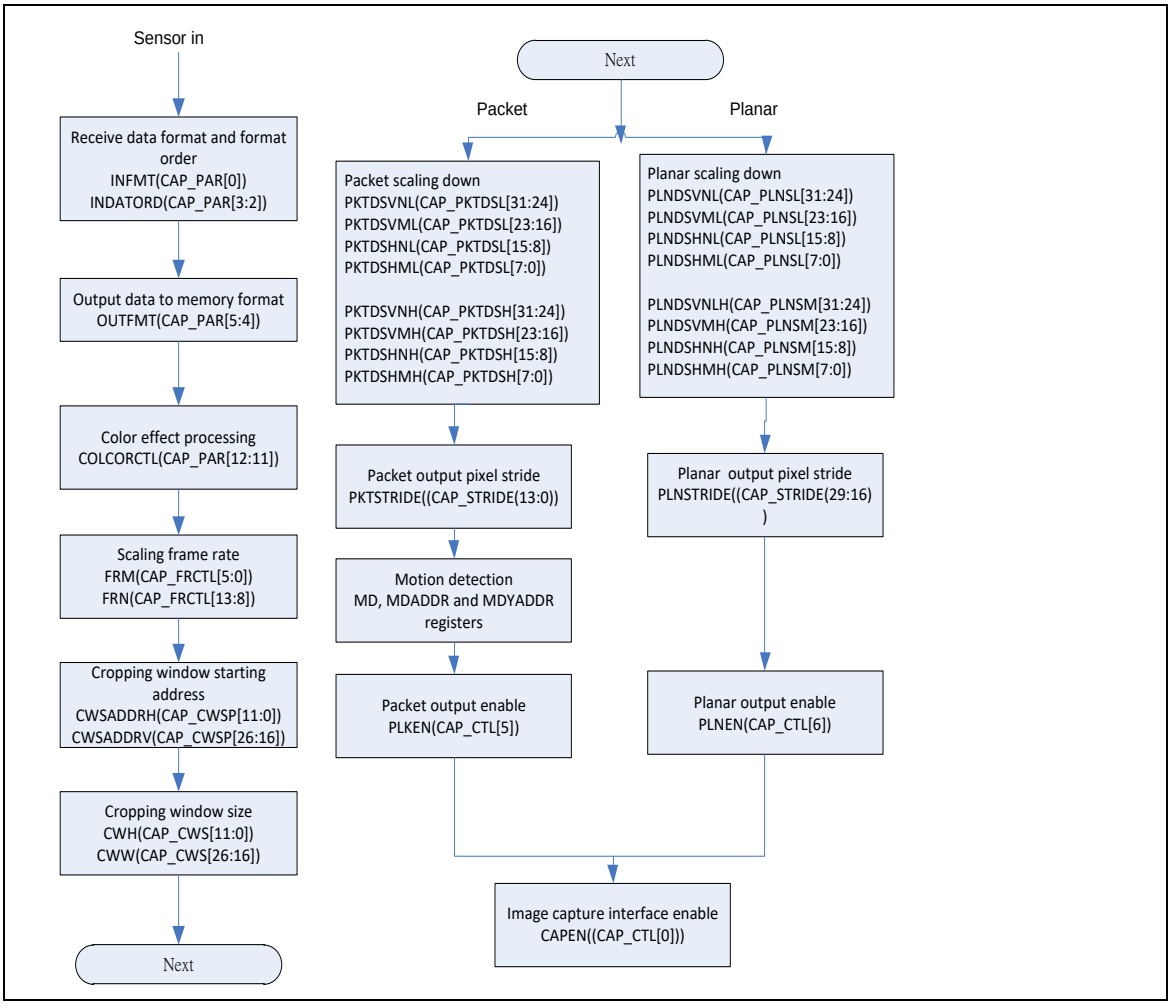


Figure 26.4-1 Image Capture Flow

26.4.3 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.

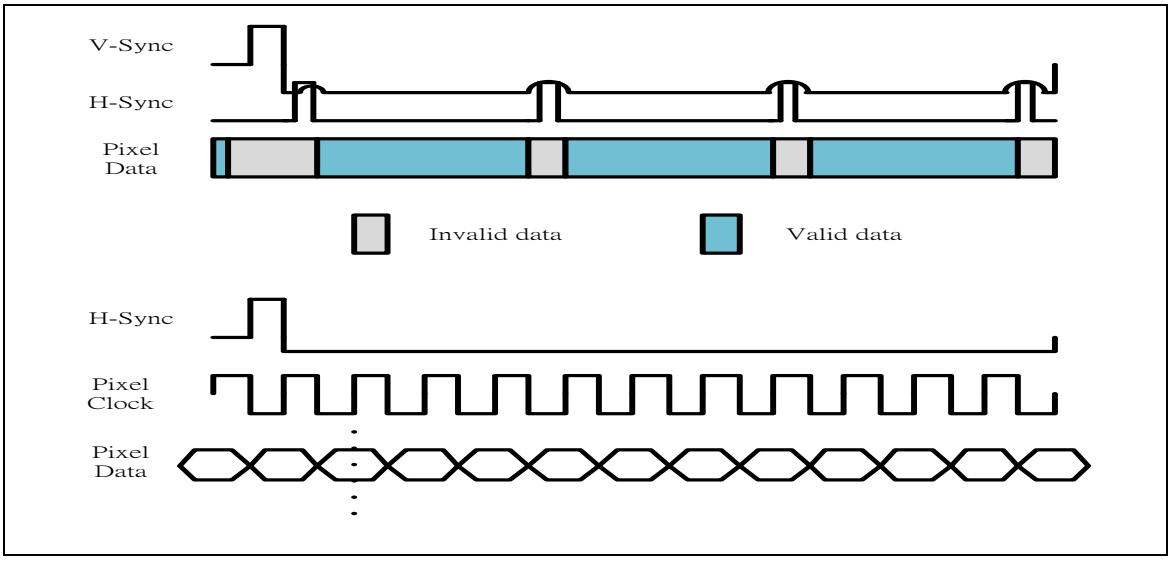


Figure 26.4-2 Control Signal Polarity

26.4.4 Sensor Data Input Order

Input data order may be U0Y0VY1, Y0U0Y1V0, V0Y0U0Y1 or Y0V0Y1U0 after cropping the input data.

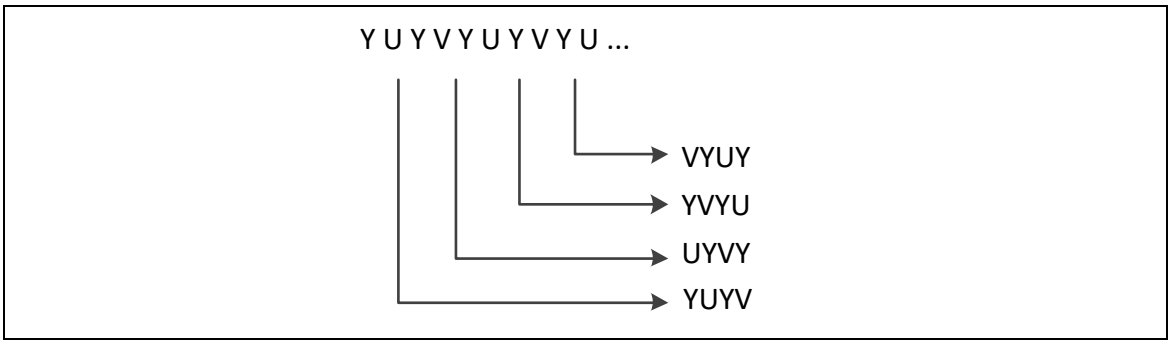


Figure 26.4-3 Sensor Data Input Order

26.4.5 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])**.**

26.4.6 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 =(PKTSHNH<<8+PKTSHNL)/(PKTSHMH<<8+PKTSHML)

The vertical downscale factor =

288/480 =(PKTSVNH<<8+PKTSVNL)/(PKTSVMH<<8+PKTSVML)

26.4.7 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.

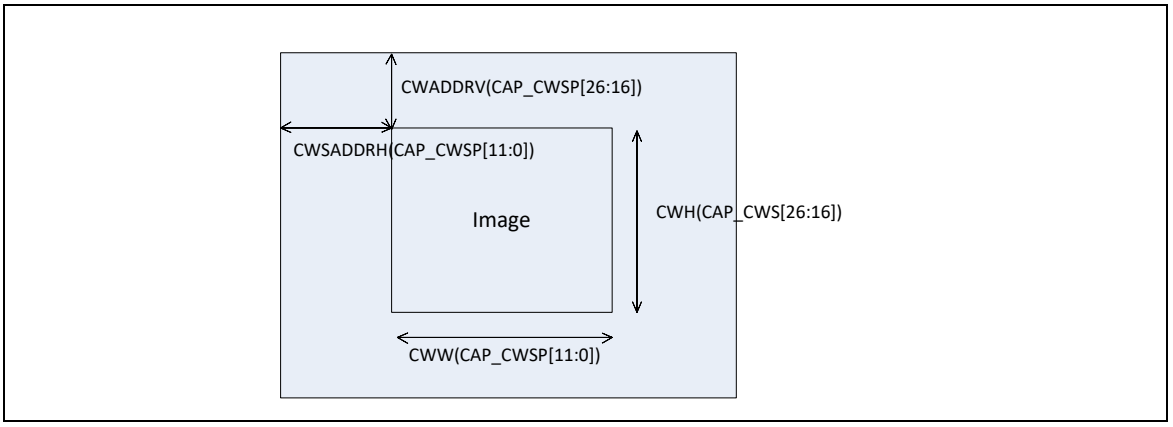


Figure 26.4-4 Cropping Window Configuration

26.4.8 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.

26.4.9 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.

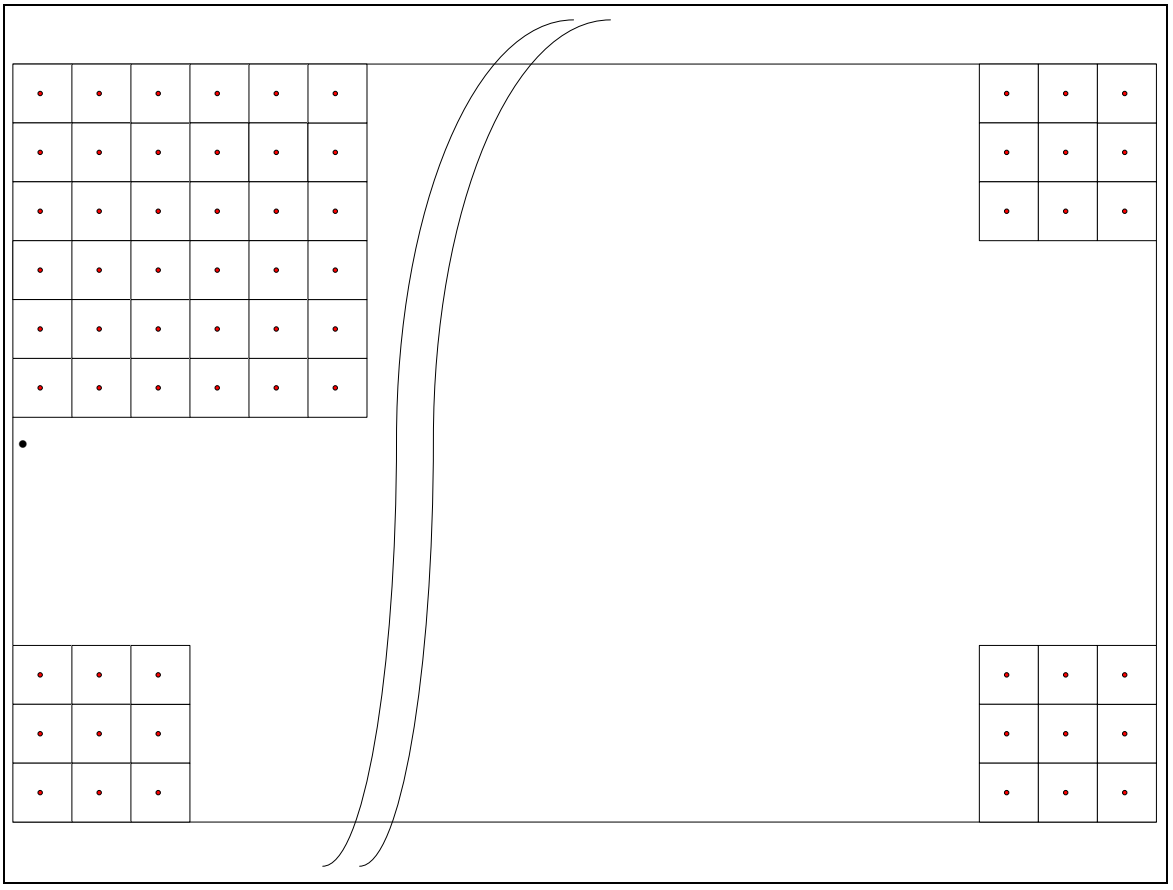


Figure 26.4-5 Motion Detection

The format of motion detection output buffer lists as following.

- One bit mode (Captured Width = 640 for block size 16x16)

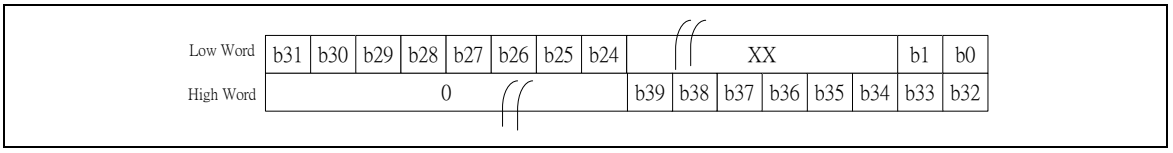


Figure 26.4-6 One Bit Mode Motion Detection Output

- 1 bit DIFF(MSB) + 7 bit Y Differential (Captured Width = 352 for block size 16x16)

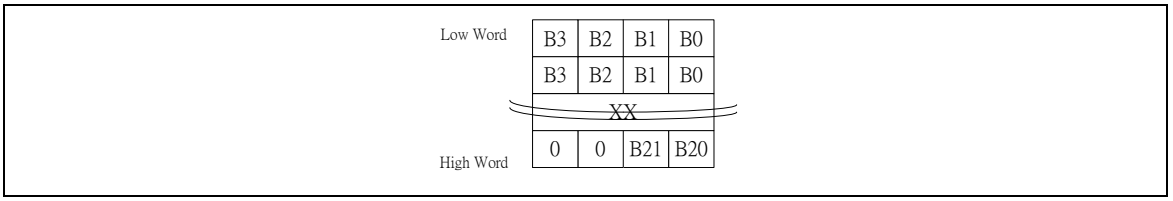


Figure 26.4-7 Y Differentional Motion Detection Output

26.5 Register Map

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

Register	Offset	R/W	Description	Reset Value
Capture Base Address: **CAPx_BA = 0xB002_4000 - (0x10000*x)** **x=0, 1**				
CAPx_CTL	CAPx_BA+0x00	R/W	Image Capture Interface Control Register	0x0000_0040
CAPx_PAR	CAPx_BA+0x04	R/W	Image Capture Interface Parameter Register	0x0000_0000
CAPx_INT	CAPx_BA+0x08	R/W	Image Capture Interface Interrupt Register	0x0000_0000
CAPx_POSTERIZE	CAPx_BA+0x0C	R/W	YUV Component Posterizing Factor Register	0x0000_0000
CAPx_MD	CAPx_BA+0x10	R/W	Motion Detection Register	0x0000_0000
CAPx_MDADDR	CAPx_BA+0x14	R/W	Motion Detection Output Address Register	0x0000_0000
CAPx_MDYADDR	CAPx_BA+0x18	R/W	Motion Detection Temp Y Output Address Register	0x0000_0000
CAPx_SEPIA	CAPx_BA+0x1C	R/W	Sepia Effect Control Register	0x0000_0000
CAPx_CWSP	CAPx_BA+0x20	R/W	Cropping Window Starting Address Register	0x0000_0000
CAPx_CWS	CAPx_BA+0x24	R/W	Cropping Window Size Register	0x0000_0000
CAPx_PKTSL	CAPx_BA+0x28	R/W	Packet Scaling Vertical/Horizontal Factor Register (LSB)	0x0000_0000
CAPx_PLNSL	CAPx_BA+0x2C	R/W	Planar Scaling Vertical/Horizontal Factor Register (LSB)	0x0000_0000
CAPx_FRCTL	CAPx_BA+0x30	R/W	Scaling Frame Rate Factor Register	0x0000_0000
CAPx_STRIDE	CAPx_BA+0x34	R/W	Frame Output Pixel Stride Width Register	0x0000_0000
CAPx_FIFOTH	CAPx_BA+0x3C	R/W	FIFO Threshold Register	0x070D_0507
CAPx_CMPADDR	CAPx_BA+0x40	R/W	Compare Memory Base Address Register	0xFFFF_FFFC
CAPx_PKTSM	CAPx_BA+0x48	R/W	Packet Scaling Vertical/Horizontal Factor Register (MSB)	0x0000_0000
CAPx_PLNSM	CAPx_BA+0x4C	R/W	Planar Scaling Vertical/Horizontal Factor Register (MSB)	0x0000_0000
CAPx_CURADDRP	CAPx_BA+0x50	R	Current Packet System Memory Address Register	0x0000_0000
CAPx_CURADDRY	CAPx_BA+0x54	R	Current Planar Y System Memory Address Register	0x0000_0000
CAPx_CURADDRU	CAPx_BA+0x58	R	Current Planar U System Memory Address Register	0x0000_0000
CAPx_CURVADDR	CAPx_BA+0x5C	R	Current Planar V System Memory Address Register	0x0000_0000
CAPx_PKTBA0	CAPx_BA+0x60	R/W	System Memory Packet Base Address 0 Register	0x0000_0000
CAPx_PKTBA1	CAPx_BA+0x64	R/W	System Memory Packet Base Address 1 Register	0x0000_0000
CAPx_YBA	CAPx_BA+0x80	R/W	System Memory Planar Y Base Address Register	0x0000_0000
CAPx_UBA	CAPx_BA+0x84	R/W	System Memory Planar U Base Address Register	0x0000_0000
CAPx_VBA	CAPx_BA+0x88	R/W	System Memory Planar V Base Address Register	0x0000_0000

27 ANALOG TO DIGITAL CONVERTER (ADC)

27.1 Overview

The NUC980 contains one 12-bit Successive Approximation Register analog-to-digital converter (SAR A/D converter) with 9 input channels. NUC980 ADC can convert the voltage of input channel to a 12-bit digital conversion result.

27.2 Features

- Resolution: 12-bit resolution.
- DNL: +/-1.5 LSB, INL: +/-3 LSB.
- Data Rate: 200 KSPS.
- Analog Input Range: V_{REF} to AGND, could be rail-to-rail.
- Analog Supply: 2.7-3.6V.
- Digital Supply: 1.2V.
- 9 Single-Ended Analog inputs.

27.3 Functional Description

27.3.1 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]).

27.3.2 ADC Transfer Function

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

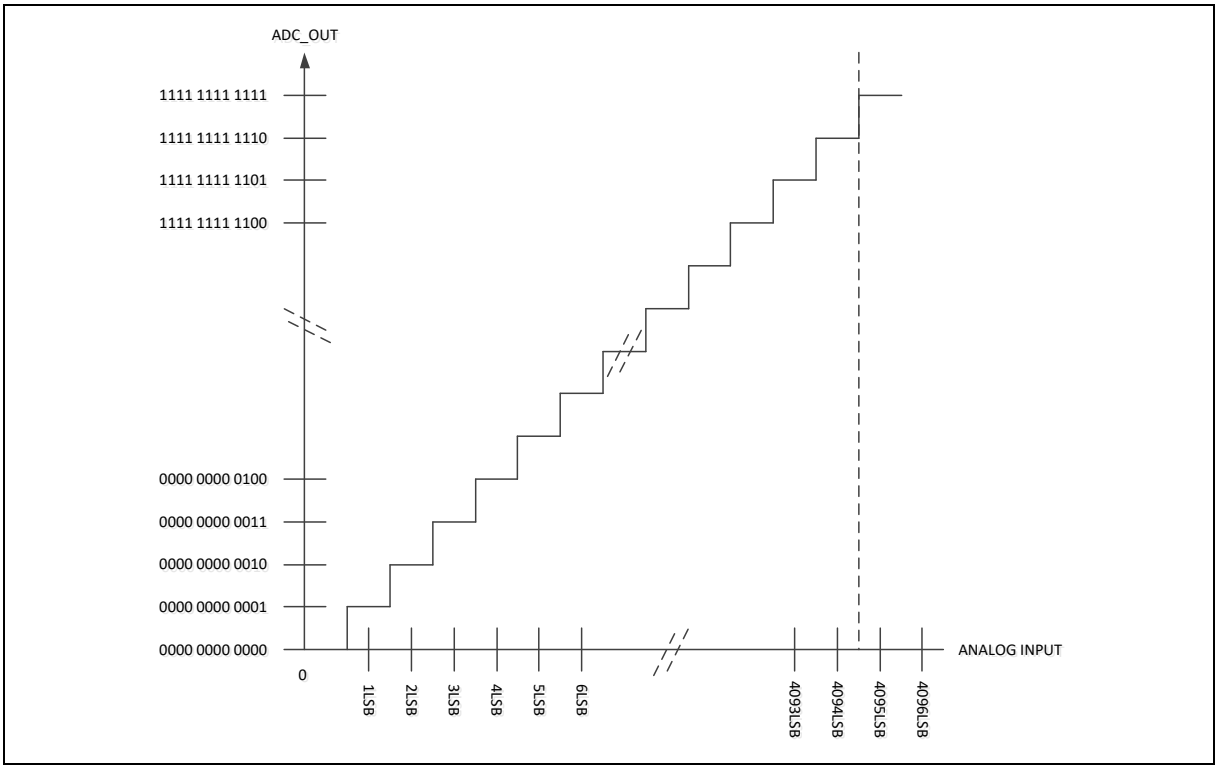


Figure 27.3-1 ADC Transfer Function

27.3.3 **ADC Timing Diagram**

When A/D conversion for each enabled function is completed, the result is transferred to the A/D data register corresponding to each functional, by the way, setting ADC reference voltage stable counter REFCNT (ADC_CONF[19:16]) to add additional wait cycle (n), use sampling counter registers ADCSAMPCNT (ADC_CONF[31:24]) to add additional sample and hold cycle (n).

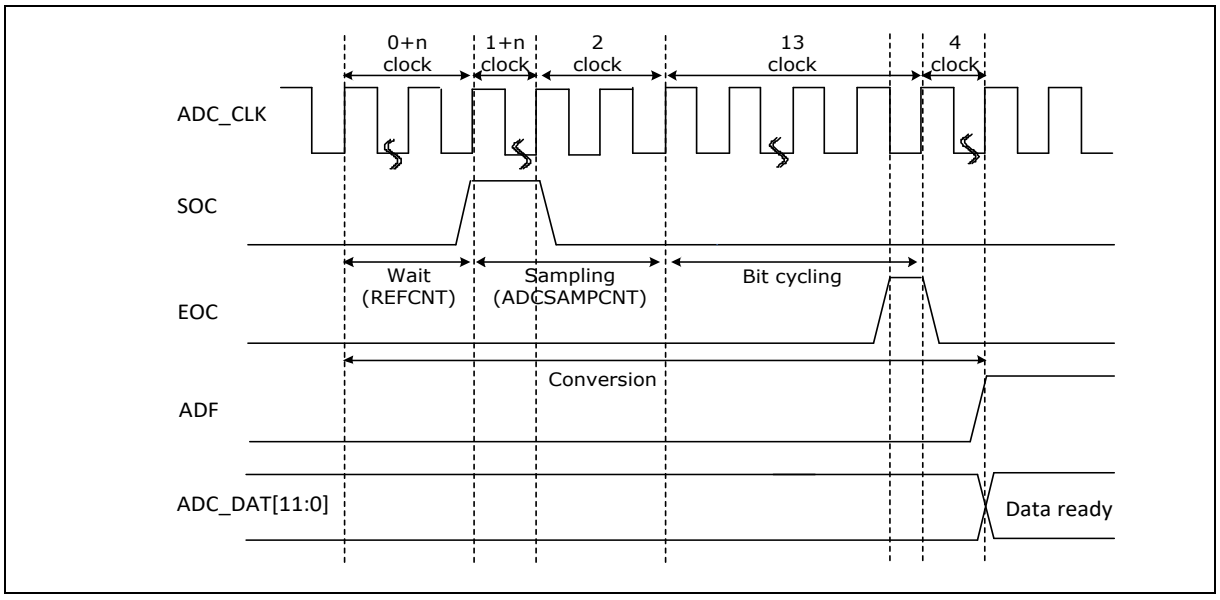


Figure 27.3-2 ADC Timing Diagram

Demonstrations of ADC timing configuration software program as follows:

```
unsigned int refcnt, samplecnt;
refcnt = 5;
samplecnt = 10;
rREG_CONF = (rREG_CONF & ~0xF0000) | (refcnt << 16); /* set REFCNT */
rREG_CONF = (rREG_CONF & ~0xFF000000) | (refcnt << 24); /* set SAMPLECNT */
```

27.3.4 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 0~7:\n");
num=getchar();
switch(num)
{
      case '0': val=0; break;
      case '1': val=1; break;
      case '2': val=2; break;
      case '3': val=3; break;
      case '4': val=4; break;
      case '5': val=5; break;
      case '6': val=6; break;
      case '7': val=7; break;
}
rREG_CONF |= ADC_CONF_NACEN | val<<12 | (3<<6);
rREG_ISR = ADC_ISR_MF | ADC_ISR_NACF;
/* narmal_test interrupt mode */
rREG_IER |= ADC_IER_MIEN;
do{
      complete = 0;
      rREG_CTL |= ADC_CTL_MST;
      UART_printf("Waiting for Normal mode Interrupt\n");
      while(!(rREG_ISR & ADC_ISR_MF));
      rREG_ISR = ADC_ISR_MF; //Clear MF flag
      if(rREG_ISR & ADC_ISR_NACF)
      {
            data=rREG_DATA;
```

```
            n=(33*data*100)>>12;
            d1=n/1000;
            d2=n%1000;
            printf("DATA=0x%08x,voltage=%d.%dv\n",data,d1,d2);
      }
      else
            UART_printf("interrupt error\n");
}while(1);
```

27.4 Register Map

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

Register	Offset	R/W	Description	Reset Value
ADC Base Address: **ADC_BA = 0xB004_3000**				
ADCON	ADC_BA+0x00	R/W	ADC Control	0x0000_0000
ADC_CONF	ADC_BA+0x04	R/W	ADC Configure	0x0000_0000
ADC_IER	ADC_BA+0x08	R/W	ADC Interrupt Enable Register	0x0000_0000
ADC_ISR	ADC_BA+0x0C	R/W	ADC Interrupt Status Register	0x0000_0000
ADC_DATA	ADC_BA+0x28	R	ADC Normal Conversion Data	0x0000_0000

28 REVISION HISTORY

Date	Revision	Description
2018.07.1	1.00	Initial version.
2019.05.02	1.01	Editorial change.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

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.