NUC740 Programming Guide

Revision 1.2
5/8/2003

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Table No.: 2002-NUC740-11-A

Revision History

Revision	Date	Comment
1.0	12/17/2002	Initial Version for NUC740A
1.1	05/08/2003	Replace the Chinese Font to English Font. Update the WatchDog Timer Information.
1.2	5/15/2003	Update WAITVT, and Typing Error on ROM/Flash Interface.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Table of Contents

1 Overview 11
1.1 Features 13
1.1.1 Architecture 13
1.1.2 External Bus Interface 13
1.1.3 Instruction and Data Cache 13
1.1.4 Timers 13
1.1.5 Advanced Interrupt Controller 13
1.1.6 UART 14
1.1.7 DMA Controller 14
1.1.8 Ethernet Controller 14
1.1.9 NAT Accelerator 15
1.1.10 USB Host Controller 15
1.1.11 Programmable I/Os 15
1.1.12 PLL 15
2 EBI (External Bus Interface) 16
2.1 Overview 16
2.2 Block Diagram 17
2.2.1 SDRAM interface 17
2.3 Registers 18
2.4 Functional Descriptions 18
2.4.1 EBI Control Register (EBICON) 18
2.4.2 ROM/Flash control register 21
2.4.3 SDRAM configuration registers 25
2.4.4 External I/O control registers 29
2.4.5 A system memory initialization example flow chart 34
2.4.6 REMAPPING 35
2.5 Example Code 38
2.5.1 PROJECT FILE 38
2.5.2 SOURCE CODE 39
3 Cache Controller 82
3.1 Overview 82
3.2 Block Diagram 83
3.3 Registers 85
3.4 Functional Descriptions 85
3.4.1 On-Chip RAM 85
3.4.2 Non-Cacheable Area 85
3.4.3 Cache Flushing 86
3.4.4 Cache Enable and Disable 87
3.4.5 Cache Load and Lock 87

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

3.4.6 Cache Unlock 89
4 Timers 90
4.1 Overview 90
4.2 Block Diagram 91
4.3 Registers 91
4.4 IV. Functional Descriptions 92
4.4.1 Interrupt Frequency 92
4.4.2 Initialization 92
4.4.3 Timer Interrupt Service Routine 94
4.4.4 Watchdog Timer 95
4.5 Sample Code 98
4.5.1 Simple Timer Interrupt handler 98
4.5.2 Timer Sample Program 98
4.5.3 Watchdog Timer Sample Program 103
5 AIC (Advanced Interrupt Controller) 109
5.1 Overview 109
5.2 Block Diagram 110
5.3 Registers 111
5.4 Functional Descriptions 113
5.4.1 NUC740 Interrupt Sources 113
5.4.2 Interrupt channel configuration 114
5.4.3 Interrupt Masking 114
5.4.4 Interrupt Clearing and Setting 115
5.4.5 Software Priority Scheme 115
5.4.6 Hardware Priority Scheme 117
5.4.7 Sample Code 120
6 UART 124
6.1 Overview 124
6.2 Block Diagram 124
6.3 Registers 125
6.4 Functional Descriptions 125
6.4.1 Baud Rate 125
6.4.2 Initializations 126
6.4.3 Polled I/O Functions 128
6.4.4 Interrupted I/O Functions 129
6.5 Sample Code 134
7 GDMA 137
7.1 Overview 137
7.2 Block Diagram 138
7.3 Registers 139
7.4 Functional Descriptions 139

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

7.4.1 GDMA Configuration 139
7.4.2 Transfer Count 141
7.4.3 Transfer Termination 142
7.4.4 GDMA operation started by software 142
7.4.5 GDMA operation started by nXDREQ 144
7.4.6 Fixed Address 144
7.4.7 Block Mode Transfer 144
7.4.8 Single Mode Transfer 144
7.4.9 Demand Mode Transfer 145
7.5 Sample Code 145
7.5.1 NUC740.H 146
7.5.2 test740.c 156
7.5.3 GDMA.c 156
8 EMC (Ethernet MAC Controller) 198
8.1 Overview 198
8.2 Block Diagram 199
8.3 Registers 199
8.3.1 EMC 0 Control registers 199
8.3.2 EMC 0 Status Registers 201
8.3.3 EMC 1 Control Registers 201
8.3.4 EMC 1 Status Registers 202
8.4 Functional Descriptions 203
8.4.1 Media Access Control (MAC) 203
8.4.2 Rx Buffer Descriptor (RXBD) 203
8.4.3 Rx Status (RXSTA) : Receive Status 204
8.4.4 NAT INFORMATION (Reserved if disabled) 206
8.4.5 DMA Receive Descriptor Link List Start Address Register 207
8.4.6 Initialize Rx Buffer Descriptors 208
8.4.7 Tx Buffer Descriptor (TXBD) 210
8.4.8 Tx Status (TXSTA) 211
8.4.9 DMA Transmit Descriptor Link-list Start Address Register 212
8.4.10 Initialize Tx Buffer Descriptors 212
8.4.11 CAM Command Register 215
8.4.12 CAM Enable Register 217
8.4.13 CAM Address Registers 217
8.4.14 MAC Interrupt Enable Register 220
8.4.15 MAC Interrupt Status Register 223
8.4.16 MAC Command Register 226
8.4.17 MAC General Status Register 229
8.4.18 MII 230
8.4.19 DMA Receive Frame Control Register 235

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.20 Transmit Start Demand Register 235
8.4.21 Receive Start Demand Register 236
8.4.22 FIFO Threshold Adjustment Register 236
8.4.23 DMA Receive Frame Status Register 238
8.4.24 MAC Missed Packet Count register 239
8.4.25 Control Frames 240
8.4.26 Descriptor Status Registers 242
8.4.27 Interrupts Setting 245
8.4.28 Packet Processing 247
8.5 V. Sample Code 254
9 NATA (Network Address Translation Accelerator) 280
9.1 Overview 280
9.2 Block Diagram 281
9.3 Registers 281
9.3.1 NATA Control and Status Registers 282
9.3.2 Address Lookup and Replacement Entry Registers 282
9.4 Functional Descriptions 284
9.5 NATA Processing Example (LAN to WAN) 286
9.6 NATA Processing Example (WAN to LAN) 287
9.7 NATA Registers 288
9.7.1 MAC Address Registers Format 288
9.7.2 MAC Address Most Significant Word Register(EXMACM, INMACM, LSMACxM, RSMACxM) 288
9.7.3 MAC Address Least Significant Word Register(EXMACL, INMACL, LSMACxL, RSMACxL) 288
9.7.4 IP Address Registers (MASADx, LSADx) 289
9.7.5 Port Number Registers (MASPNx, LSPNx) 289
9.7.6 MAC Command Register (MCMDR_0, MCMDR_1) (0xFFF0308C, 0xFFF0388C) 289
9.7.7 NAT Entry x Configuration Registers (NATCFGx) 290
9.7.8 NAT Entry Configuration Register (NATCFGx) Programming Note : 292
9.7.9 Data Content in the NATA table 292
9.7.10 NATA comparison and replacement table at different port 292
9.7.11 Possible Settings for Different Applications 293
9.8 NAT Command Registers (NATCMD) 293
9.8.1 NAT Counter x Clear Register (NATCCLRx, x : 0-3) 294
9.8.2 EMC Rx Buffer Descriptor 294
9.8.3 Rx Status (RXSTA) : Receive Status 295
9.8.4 NAT INFORMATION 296
9.9 Software Management on NATA 297
9.10 Sample Code 300
10 USB Host Controller 311

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.1 Overview 311
10.2 Registers Map 312
10.3 Block Diagram 313
10.4 Data Structures 315
10.4.1 Endpoint Descriptor (ED) Lists 316
10.4.2 Transfer Descriptor 318
10.4.3 Host Controller Communication Area 320
10.5 Programming Note 321
10.5.1 Initialization 321
10.5.2 USB States 322
10.5.3 Add/Remove Endpoint Descriptors 323
10.5.4 Add/Remove Transfer Descriptors 324
10.5.5 IRP Processing 327
10.5.6 Interrupt Processing 332
10.5.7 Done Queue Processing 335
10.5.8 Root Hub 343

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Table of Figures

Figure 1-1 NUC740 Functional Block Diagram 12
Figure 2-1 SDRAM Interface 17
Figure 2-2 External I/O write operation timing 33
Figure 2-3 External I/O read operation timing 33
Figure 2-4 System Memory Map Setting Flow 34
Figure 2-5 The Project File NUC740.MCP 38
Figure 3-1 Instruction Cache Organization Block Diagram 83
Figure 3-2 Data Cache Organization Block Diagram 84
Figure 3-3 Cache Control Register 86
Figure 3-4 Cache Configuration Register 87
Figure 3-5 Cache Load and Lock 88
Figure 4-1 Timer Block Diagram 91
Figure 4-2 Timer Initialization Sequence 93
Figure 4-3 Timer Interrupt Service Routine 94
Figure 4-4 Enable Watchdog Timer 96
Figure 4-5 Watchdog Timer ISR 97
Figure 5-1 AIC block diagram 110
Figure 5-2 NUC740 Interrupt Sources 113
Figure 5-3 Source Control Register 114
Figure 5-4 Sequential Priority Scheme 116
Figure 5-5 Interrupt Service Routine with Vector 118
Figure 5-6 Using hardware priority scheme 119
Figure 6-1. UART Block Diagram 124
Figure 6-2 UART initialization 126
Figure 6-3 Transmit data in polling mode 128
Figure 6-4 Receive data in polling mode 129
Figure 6-5 Output function in interrupt mode 130
Figure 6-6 Input functions in interrupt mode 131
Figure 6-7 Interrupt Service Routine 133
Figure 7-1 GDMA Block Diagram 138
Figure 7-2 The bit-fields of the GDMA control register. 140
Figure 7-3 GDMA operations 141
Figure 7-4 Software GDMA Transfer 143
Figure 7-5 The project file of the GDMA test program. 145
Figure 8-1 EMC Block Diagram 199
Figure 8-2 Rx Descriptor Initialization 209
Figure 8-3 Tx Descriptor Initialization 214
Figure 8-4 CAM Setting Combination and Received Packets 216
Figure 8-5 Packet Transmission Flow 248

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 8-6 Tx Interrupt Service Routine Flow 250
Figure 8-7 Rx Interrupt Service Routine 252
Figure 9-1 NATA Block Diagram 281
Figure 9-2 NAT lookup and replacement flow 284
Figure 9-3 NAT Process Flow 285
Figure 9-4 LAN to WAN NATA Processing 286
Figure 9-5 WAN to LAN NATA Processing 287
Figure 9-6 NATA Management 299
Figure 10-1 Endpoint Descriptor Format 316
Figure 10-2 General Transfer Descriptor Format 318
Figure 10-3 Isochronous Transfer Descriptor Format 318
Figure 10-4 Remove an Endpoint Descriptor 324
Figure 10-5 ED list and TD queue 325

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

List of Tables

Table 3-1 The size and start address of On-Chip RAM 85
Table 4-1 Timer Reference Setting Values 92
Table 5-1 AIC Register Definition 111
Table 6-1 General Baud Rate Settings 126
Table 10-1 HCCA (Host Controller Communication Area) 320

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

1 Overview

The NUC740 16/32-bit RISC micro-controller is a cost-effective, high-performance micro-controller solution for Ethernet-based system. An integrated Ethernet controller, the NUC740, is designed for use in managed communication hubs and routers.

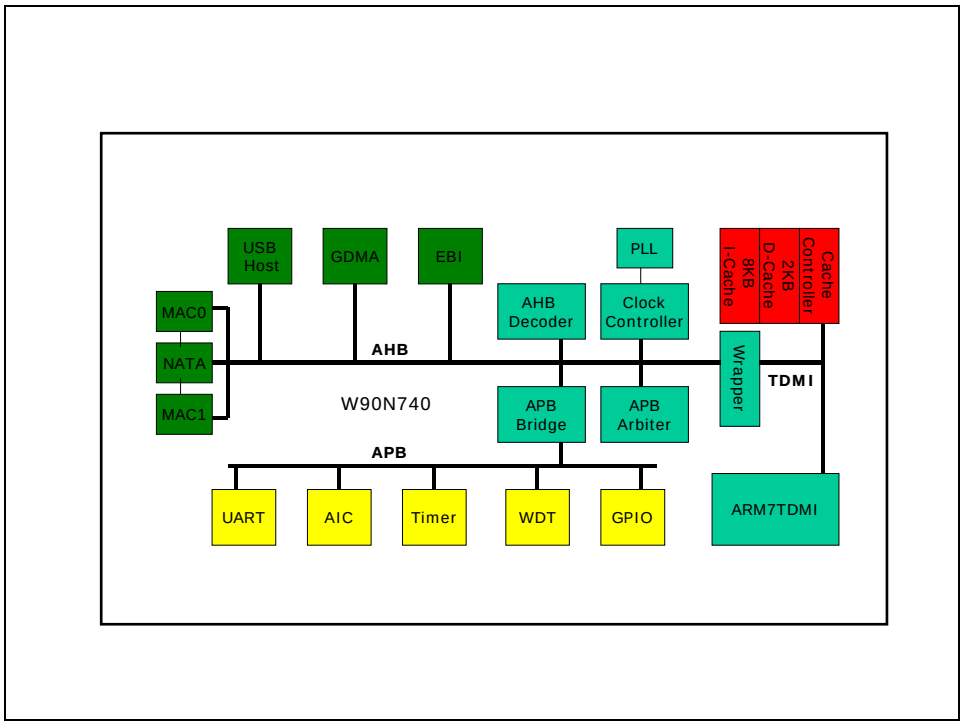
The NUC740 is built around an outstanding CPU core : on the 16/32 ARM7TDMI based RISC processor designed by Advanced RISC Machines, Ltd. The ARM7TDMI core is a low power, general-purpose integrated circuits. Its simple, elegant, and fully static design is particularly suitable for cost-sensitive and power-sensitive applications.

The NUC740 offers an 8K-byte I-cache/SRAM, a 2K-byte D-cache/SRAM and two MACs of Ethernet controller that reduces total system cost. An NAT Accelerator is supported to reduce the software loading for network address translation process. Most of the on-chip function blocks have been designed using an HDL synthesizer and the NUC740 has been fully verified in Nuvoton's state-of-the art ASIC test environment.

Important peripheral functions Other on-chip peripherals include one USB host controller, one UART channel, 2-Channel GDMA(General Purpose DMA), one watch-dog timer, two 24-bit timers with 8-bit pre-scale, 21 programmable I/O ports, and an advanced interrupt controller; The external bus interface (EBI) controller provides support for SDRAM, ROM/SRAM, flash memory and I/O devices. The System Manager includes an internal 32-bit system bus arbiter and a PLL clock controller.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 1-1 NUC740 Functional Block Diagram



The following integrated on-chip functions are described in detail in this document. On the following chapters, each module-programming note will be described in detailed.

- Chapter 2. External Bus Interface Controller
- Chapter 3. 8-K byte I-Cache and 2K-byte D-Cache /SRAM
- Chapter 4. Timers
- Chapter 5. Advanced Interrupt Controller
- Chapter 6. UART
- Chapter 7. GDMA
- Chapter 8. Ethernet MAC Controllers
- Chapter 9. NAT Accelerator
- Chapter 10. USB Host Controller
- Chapter 11. Programmable I/O Ports

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

1.1 Features

1.1.1 Architecture

- Integrated system for embedded Ethernet applications
- Fully 16/32-bit RISC architecture
- Little/Big-Endian mode supported
- Efficient and powerful ARM7TDMI core
- Cost-effective JTAG-based debug solution

1.1.2 External Bus Interface

- 8/16/32-bit external bus support for ROM/SRAM, flash memory, SDRAM and external I/Os
- One external bus master with bus request/acknowledge pins
- Support for SDRAM
- Programmable access cycle (0-7 wait cycle)
- Four-word depth write buffer
- Cost-effective memory-to-peripheral DMA interface
- Support for PCMCIA 16-bit PC Card devices

1.1.3 Instruction and Data Cache

- Two-way, Set-associative, 8K-byte I-cache and 2K-byte D-cache
- Support for LRU (Least Recently Used) Protocol
- Cache is configurable as an internal SRAM
- Support Cache Lock function

1.1.4 Timers

- Two programmable 24-bit timers with 8-bit pre-scalar
- One programmable 24-bit Watch-Dog timer
- One-short mode, period mode or toggle mode operation

1.1.5 Advanced Interrupt Controller

- 18 interrupt sources, including 4 external interrupt sources
- Programmable normal or fast interrupt mode (IRQ, FIQ)
- Programmable as either edge-triggered or level-sensitive for 4 external interrupt sources
- Programmable as either low-active or high-active for 4 external interrupt sources
- Priority methodology is encoded to allow for interrupt daisy-chaining

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

- Automatically mask out the lower priority interrupt during interrupt nesting
- Automatically clear the interrupt flag when the interrupt source is programmed to be edge-triggered

1.1.6 UART

- One UART (serial I/O) blocks with interrupt-based operation
- Support for 5-bit, 6-bit, 7-bit, or 8-bit serial data transmit and receive
- Programmable baud rates
- 1, ½, or 2 stop bits
- Odd or even parity
- Break generation and detection
- Parity, overrun, and framing error detection
- X16 clock mode
- Support for Modem control

1.1.7 DMA Controller

- 2-channel General DMA for memory-to-memory data transfers without CPU intervention
- Initialed by a software or external DMA request
- Increments or decrements a source or destination address in 8-bit, 16-bit or 32-bit data transfers
- 4-data burst mode

1.1.8 Ethernet Controller

- DMA engine with burst mode
- MAC Tx/Rx buffers (256 bytes Tx, 256 bytes Rx)
- Data alignment logic
- Endian translation
- 100/10-Mbit per second operation
- Full compliance with IEE standard 802.3
- MII and RMII interface
- Station Management signaling
- On-Chip CAM (up to 16 destination addresses)
- Full-duplex mode with PAUSE feature
- Long/short packet modes
- PAD generation

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

1.1.9 NAT Accelerator

- Support TCP / UDP packets
- Hardware acceleration on IP address / port number lookup and replacement for network address translation, including MAC address translation
- Provide 64 entries of translation table

1.1.10 USB Host Controller

- USB 1.1 compliant
- Compatible with Open HCI 1.0 specification
- Supports low-speed and full speed devices
- Build-in DMA for real time data transfer
- Option for on-chip USB transceiver or external USB transceiver

1.1.11 Programmable I/Os

- 21 programmable I/O ports
- Pins individually configurable to input, output, or I/O mode for dedicated signals
- I/O ports Configurable for Multiple functions

1.1.12 PLL

- The external clock can be multiplied by on-chip PLL to provide high frequency system clock
- The input frequency range is 3-30MHz; 15MHz is preferred.
- One PLL for both CPU and USB host controller
- Programmable clock frequency

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2 EBI (External Bus Interface)

2.1 Overview

NUC740 supports External Bus Interface (**EBI**), which controls the access to the external memory (ROM/FLASH, SDRAM) and External I/O devices. The **EBI** has seven chip selects to select one ROM/FLASH bank, two SDRAM banks, and four External I/O banks and 25-bit address bus. It supports 8-bit, 16-bit, and 32-bit external data bus width for each bank.

The EBI has the following functions :

- SDRAM controller
- EBI control register
- ROM/FLASH interface
- External I/O interface

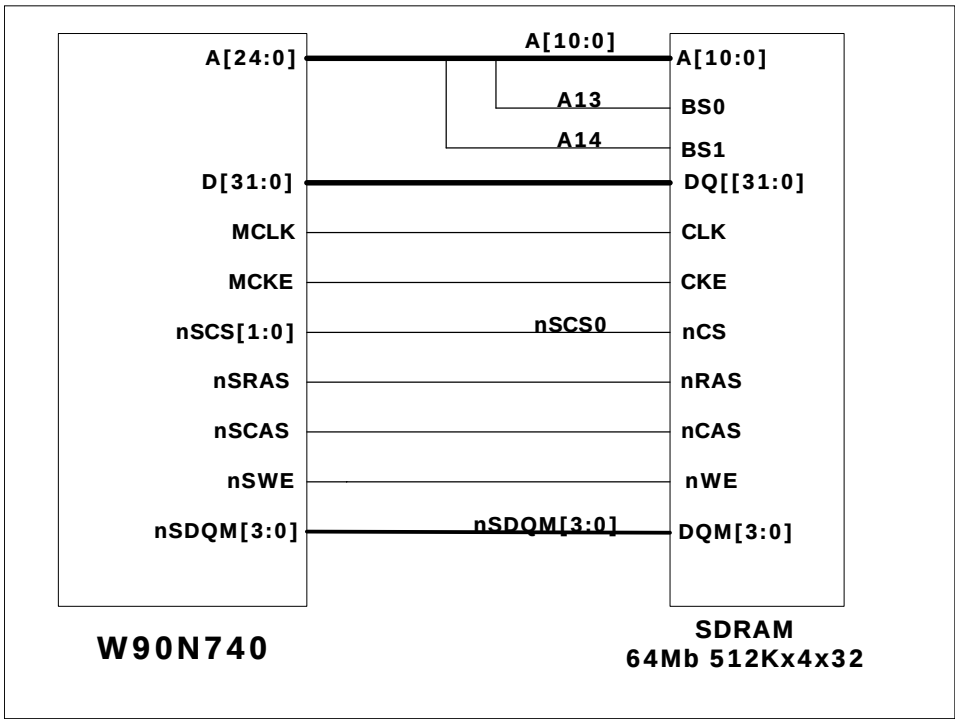
The base addresses of SDRAM, ROM/FLASH, and External I/O are all programmable. Thus they can be set in a specified address ranges in memory. The EBI also offer power-on setting to ensure the system can be boot by from ROM/FLASH.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2.2 Block Diagram

2.2.1 SDRAM interface

Figure 2-1 SDRAM Interface



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2.3 Registers

Register	Address	R/W	Description	Reset Value
EBICON	0xFFF0.1000	R/W	EBI control register	0x0001.0000
ROMCON	0xFFF0.1004	R/W	ROM/FLASH control register	0x0000.0XFC
SDCONF0	0xFFF0.1008	R/W	SDRAM bank 0 configuration register	0x0000.0800
SDCONF1	0xFFF0.100C	R/W	SDRAM bank 1 configuration register	0x0000.0800
SDTIME0	0xFFF0.1010	R/W	SDRAM bank 0 timing control register	0x0000.0000
SDTIME1	0xFFF0.1014	R/W	SDRAM bank 1 timing control register	0x0000.0000
EXT0CON	0xFFF0.1018	R/W	External I/O 0 control register	0x0000.0000
EXT1CON	0xFFF0.101C	R/W	External I/O 1 control register	0x0000.0000
EXT2CON	0xFFF0.1020	R/W	External I/O 2 control register	0x0000.0000
EXT3CON	0xFFF0.1024	R/W	External I/O 3 control register	0x0000.0000
CKSKEW	0xFFF0.1F00	R/W	Clock skew control register (for testing)	0x0000.0000

2.4 Functional Descriptions

2.4.1 EBI Control Register (EBICON)

The major function of EBICON is to control the SDRAM refreshing timing. This register can control is used to set the refresh period, clock, and valid time of nWAIT signal. Additionally, the EBI memory format configuration (Big, or Little Endian) can be known got by reading from the EBI control register. The auto-refresh rate is controlled by the REFRAT , and SDRAM clock is controlled by CLKEN.

There are two SDRAM refresh mode, auto-refresh mode and self-refresh mode. If SDRAM is operated in auto-refresh mode, SDRAM controller refreshes SDRAM every by a period specified by REFRAT. If SDRAM is in self-refresh mode, it is refreshed by SDRAM itself. Thus if SDRAM is operated in self-refresh mode, the CLKEN and REFEN can be disabled to reduce save power consumption. Another way to save reduce power consumption is just to disabling CLKEN, and SDRAM controller still refreshed SDRAM by every each specified refresh period. In this case, SDRAM is closed not functioning by disabling CLKEN to save for power saving, and but SDRAM controller still refreshes it to prevent from data lost. In sum, SDRAM is operated as follows :

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

NORMAL MODE :
REFEN=1
REFMOD=0
CLKEN=1
REFRAT=(proper period)

POWER SAVING MODE 1 :
REFEN=1
REFMOD=0
CLKEN=1
REFRAT=(proper period)

POWER SAVING MODE 2 :
REFEN=0
REFMOD=1
CLKEN=0
REFRAT=(don't care)

The detail description of EBI control register is as follows.

Register	Address	R/W	Description	Reset Value
EBICON	0xFFF0.1000	R/W	EBI control register	0x0001.0000

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED					REFEN	REFMOD	CLKEN
15	14	13	12	11	10	9	8
REFRAT							
7	6	5	4	3	2	1	0
REFRAT					WAITVT		LITTLE

- **REFEN [18] : Enable SDRAM refresh cycle for SDRAM bank0 & bank1**
  This bit set will start the auto-refresh cycle to SDRAM. The refresh rate is according to REFRAT bits.

- **REFMOD [17] : The refresh mode of SDRAM for SDRAM bank**
  Defines the refresh mode type of external SDRAM bank

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

0 = Auto refresh mode
1 = Self refresh mode

- **CLKEN [16] : Clock enable for SDRAM**
  Enables the SDRAM clock enable (**CKE**) control signal
  0 = Disable (power down mode)
  1 = Enable （Default）

- **REFRAT [15 : 3] : Refresh count value for SDRAM**
  The refresh period is calculated as $period = \frac{value}{fMCLK}$
  The SDRAM Controller automatically provides an auto refresh cycle for every refresh period programmed into the **REFRAT** bits when the **REFEN** bit of each bank is set.

- **WAITVT [2 : 1] : Valid time of nWAIT signal**
  This setting is used to recognized the valid nWAIT signal on the setting WAITVT time after the nWAIT signal goes low., such as, if the WAITVT set to 2 MCLKs, then NUC740 will sample the nWAIT signal at the 2nd MCLK after the nWAIT goes low to check the nWAIT signal actual valid or not. This setting is used to eliminate the bounce effect of nWAIT signal.

WAITVT [2 : 1]		Number of MCLK
0	0	1
0	1	2
1	0	3
1	1	4

- **LITTLE [0] : Read only, Little Endian mode**
  0 = EBI memory format is Big Endian mode
  1 = EBI memory format is Little Endian mode

  After power on reset, the content of **LITTLE** is the Power-On Setting value from D14 pin.
  If pin D14 is pull-down, the external memory format is Big Endian mode.
  If pin D14 is pull-up, the external memory format is Little Endian mode.
  For more detail, refer to **Power-On Setting** of **System Manager**.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2.4.2 ROM/Flash control register

ROM/Flash control register is used to control the configuration of the boot ROM. In this register, the size, base address, access type and access timing are specified. The base address of the boot ROM can be set by BASADDR. Although the width of BASADDR is only 13 bits, the real start address of the boot ROM is calculated as BASADDR << 18. Thus the range of the start address of the boot ROM is from 0x0 to (2^13-1)*2^18. However, the system memory map should be concerned together when setting the base address the base address setting should be checked to prevent from using RESERVED memory address. The system memory map can be found in NUC740 spec data sheet.

After system reset, the EBI controller has uses the special power-on setting to ensure the boot ROM to be bootable. These setting are as follows :

- The EBI controller is select to the boot ROM was selected by EBI controller after reset.
- The reset value of BASADDR of ROM/Flash control register is 0.
- The default size of the boot ROM is 256Kb256KB.
- The default value of tACC is the longest value. This value is supposed to suit support any kind of ROM/Flash.
- The boot ROM/Flash data bus width is determined by the data bus signals D[13 : 12] in power-on setting. The external hardware has the responsibility to weak needs to do the pull-up, or pull-down setting on the D[13 : 12] according to the boot ROM/Flash types.
- PGMODE is set in normal ROM mode.

By the configurations shown above, the instruction fetch can be sure to be performed can be fetched from the start of the boot ROM. However, if the boot ROM/Flash has more others functions, ex : such as PGMODE, or more with larger size, the software has the responsibility to correct the setting bootup program should configure the of ROM/Flash control register to let it work correctly after boot.

The ROM/Flash interface is designed for the boot ROM and it is supposed only to before read operations. However, if a flash is attached to the ROM/Flash interface, it still can be written by the writing programming command provided by of the flash. The ROM/Flash interface doesn't hold the writing command to the ROM/Flash. Thus the boot ROM/Flash is still programmable if the boot ROM/Flash allows to be written Thus; the attached Flash can be updated also by the programming interface/sequence provided by the Flash.

The detail descriptions of ROM/Flash control register is as follows :

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Register	Address	R/W	Description	Reset Value
ROMCON	0xFFF0.1004	R/W	ROM/FLASH control register	0x0000.0XFC

31	30	29	28	27	26	25	24
BASADDR							
23	22	21	20	19	18	17	16
BASADDR					SIZE		
15	14	13	12	11	10	9	8
RESERVED				tPA			
7	6	5	4	3	2	1	0
tACC				BTSIZE		PGMODE	

- **BASADDR [31 : 19] : Base address pointer of ROM/Flash bank**
  The start address is calculated as ROM/Flash bank base pointer << 18. The base address pointer together with the "**SIZE**" bits constitutes the whole address range of each bank.

- **SIZE [18 : 16] : The size of ROM/FLASH memory**

SIZE [10 : 8]			Byte
0	0	0	**256K**
0	0	1	**512K**
0	1	0	**1M**
0	1	1	**2M**
1	0	0	**4M**
1	0	1	**8M**
1	1	0	**16M**
1	1	1	**32M**

- **tPA [11 : 8] : Page mode access cycle time**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

tPA [11 : 8]				MCLK
0	0	0	0	**1**
0	0	0	1	**2**
0	0	1	0	**3**
0	0	1	1	**4**
0	1	0	0	**5**
0	1	0	1	**6**
0	1	1	0	**7**
0	1	1	1	**8**

tPA [11 : 8]				MCLK
1	0	0	0	**10**
1	0	0	1	**12**
1	0	1	0	**14**
1	0	1	1	**16**
1	1	0	0	**18**
1	1	0	1	**20**
1	1	1	0	**22**
1	1	1	1	**24**

- **tACC [7 : 4] : Access cycle time**

tACC [7 : 4]				MCLK
0	0	0	0	**1**
0	0	0	1	**2**
0	0	1	0	**3**
0	0	1	1	**4**
0	1	0	0	**5**
0	1	0	1	**6**
0	1	1	0	**7**
0	1	1	1	**8**

tACC [7 : 4]				MCLK
1	0	0	0	**10**
1	0	0	1	**12**
1	0	1	0	**14**
1	0	1	1	**16**
1	1	0	0	**18**
1	1	0	1	**20**
1	1	1	0	**22**
1	1	1	1	**24**

- **BTSIZE [3 : 2] : Read only, the boot ROM/FLASH data bus width**
  This ROM/Flash bank is designed for a boot ROM. **BASADDR** bits determine its start address. The external data bus width is determined by the data bus signals D [13 : 12] power-on setting.

BTSIZE [3 : 2]		Bus Width
0	0	8-bit
0	1	16-bit
1	0	32-bit
1	1	RESERVED

D [13 : 12]		Bus Width
Pull-down	Pull-down	8-bit
Pull-down	Pull-up	16-bit
Pull-up	Pull-down	32-bit
Pull-up	Pull-up	RESERVED

- **PGMODE [1 : 0] : Page mode configuration**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

PGMODE [1 : 0]		Mode
0	0	**Normal ROM**
0	1	**4 word page**
1	0	**8 word page**
1	1	**16 word page**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2.4.3 SDRAM configuration registers

The SDRAM configuration registers enable software to set a number of operating parameters for the SDRAM controller. There are two configuration registers SDCONF0, SDCONF1 for SDRAM bank 0, bank 1 respectively. Each bank can have been set to different configurations. NUC740 also offers the flexible timing control registers to control the generation and processing of the control signal and can suit to control the timing of different speed type of SDRAMs. These timing control registers are SDTIME0 and SDTIME1 for SDRAM bank 0, bank 1 respectively each.

The configurations of SDCONF and SDTIME are dependent on the SDRAM types attached to the EBI interface. Thus the software should have the information about the SDRAM attached to the EBI interface before set the SDCONF and SDTIME according to the timing of SDRAM types. The SDRAM components supported by NUC740 can be found in NUC740 spec data sheet.

The base address of SDRAM bank 0 and bank 1 are also programmable. By BASADDR of SDCONF, the SDRAM bank can be place in a specific address location. BASADDR is 13 bits, and the base address is calculated as BASADDR << 18. Thus the range of the base address each SDRAM bank is from 0x0 to (2^13-1)*2^18. Whenever setting the SDCONF register, the MRSET bit should be set. If this bit doesn't set when setting SDCONF, the SDRAM controller won't issue a mode register set command to SDRAM and the setting will be invalid. The SDRAM controller offers auto pre-charge mode of SDRAM for SDRAM bank0/1. If this mode is enabled, the SDRAM will issue a pre-charge command to SDRAM when for each access. The more detail descriptions of the SDRAM control registers are as follows :

Register	Address	R/W	Description	Reset Value
SDCONF0	0xFFF0.1008	R/W	SDRAM bank 0 configuration register	0x0000.0800
SDCONF1	0xFFF0.100C	R/W	SDRAM bank 1 configuration register	0x0000.0800

31	30	29	28	27	26	25	24
BASADDR							
23	22	21	20	19	18	17	16
BASADDR					RESERVED		
15	14	13	12	11	10	9	8
MRSET	RESERVED	AUTOPR	LATENCY		RESERVED		
7	6	5	4	3	2	1	0
COMPBK	DBWD		COLUMN		SIZE		

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

- **BASADDR [31 : 19] : Base address pointer of SDRAM bank 0/1**
  The start address is calculated as SDRAM bank 0/1 base pointer << 18. The SDRAM base address pointer together with the "**SIZE**" bits constitutes the whole address range of each SDRAM bank.

- **MRSET [15] : SDRAM Mode register set command for SDRAM bank 0/1**
  This bit set will issue a mode register set command to SDRAM.

- **AUTOPR [13] : Auto pre-charge mode of SDRAM for SDRAM bank 0/1**
  Enable the auto pre-charge function of external SDRAM bank 0/1
  0 = Auto pre-charge
  1 = No auto pre-charge

- **LATENCY [12 : 11] : The CAS Latency of SDRAM bank 0/1**
  Defines the CAS latency of external SDRAM bank 0/1

LATENCY [12 : 11]		MCLK
0	0	**1**
0	1	**2**
1	0	**3**
1	1	**REVERSED**

- **COMPBK [7] : Number of component bank in SDRAM bank 0/1**
  Indicates the number of component bank (2 or 4 banks) in external SDRAM bank 0/1.
  0 = 2 banks
  1 = 4 banks

- **DBWD [6 : 5] : Data bus width for SDRAM bank 0/1**
  Indicates the external data bus width connect with SDRAM bank 0/1
  If DBWD = 00, the assigned SDRAM access signal is not generated i.e. disable.

DBWD [6 : 5]		Bits
0	0	**Bank disable**
0	1	**8-bit (byte)**
1	0	**16-bit (half-word)**
1	1	**32-bit (word)**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

- **COLUMN [4 : 3] : Number of column address bits in SDRAM bank 0/1**
  Indicates the number of column address bits in external SDRAM bank 0/1.

COLUMN [4 : 3]		Bits
0	0	**8**
0	1	**9**
1	0	**10**
1	1	**REVERSED**

- **SIZE [2 : 0] : Size of SDRAM bank 0/1**
  Indicates the memory size of external SDRAM bank 0/1

SIZE [2 : 0]			Size of SDRAM （Byte）
0	0	0	**Bank disable**
0	0	1	**2M**
0	1	0	**4M**
0	1	1	**8M**
1	0	0	**16M**
1	0	1	**32M**
1	1	0	**64M**
1	1	1	**REVERSED**

Register	Address	R/W	Description	Reset Value
SDTIME0	0xFFF0.1010	R/W	SDRAM bank 0 timing control register	0x0000.0000
SDTIME1	0xFFF0.1014	R/W	SDRAM bank 1 timing control register	0x0000.0000

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

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

- **tRCD [10 : 8] : SDRAM bank 0/1, /RAS to /CAS delay**

tRCD [10 : 8]			MCLK
0	0	0	**1**
0	0	1	**2**
0	1	0	**3**
0	1	1	**4**
1	0	0	**5**
1	0	1	**6**
1	1	0	**7**
1	1	1	**8**

- **tRDL [7 : 6] : SDRAM bank 0/1, Last data in to pre-charge command**

TRDL [7 : 6]		MCLK
0	0	**1**
0	1	**2**
1	0	**3**
1	1	**4**

- **tRP [5 : 3] : SDRAM bank 0/1, Row pre-charge time**

tRP [5 : 3]			MCLK
0	0	0	**1**
0	0	1	**2**
0	1	0	**3**
0	1	1	**4**
1	0	0	**5**
1	0	1	**6**
1	1	0	**7**
1	1	1	**8**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

- **tRAS [2 : 0] : SDRAM bank 0/1, Row active time**

tRAS [2 : 0]			MCLK
0	0	0	**1**
0	0	1	**2**
0	1	0	**3**
0	1	1	**4**
1	0	0	**5**
1	0	1	**6**
1	1	0	**7**
1	1	1	**8**

2.4.4 External I/O control registers

The NUC740 supports an external device control without glue logic. It is very cost effective because provides address decoding and control signals timing logic are not needed. The control registers can control special external I/O devices for providing the low cost external devices control solution. For instance, if there is a SRAM is attached to the external I/O bank 0. Then the SRAM can be access as memory after setting the external I/O control register of external I/O bank 0. By the way, the flash ROM also can be attached to the external I/O. There are four external I/O banks relative to four control registers called EXT0CON, EXT1CON, EXT2CON, and EXT3CON. The base address of each external I/O bank can be set by BASADDR of external I/O control register. BASADDR is 13 bits and the base address is calculated as BASADDR << 18. The more detailed descriptions of the external I/O control registers are listed as follows following :

Register	Address	R/W	Description	Reset Value
EXT0CON	0xFFF0.1018	R/W	External I/O 0 control register	0x0000.0000
EXT1CON	0xFFF0.101C	R/W	External I/O 1 control register	0x0000.0000
EXT2CON	0xFFF0.1020	R/W	External I/O 2 control register	0x0000.0000
EXT3CON	0xFFF0.1024	R/W	External I/O 3 control register	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

31	30	29	28	27	26	25	24
BASADDR							
23	22	21	20	19	18	17	16
BASADDR					SIZE		
15	14	13	12	11	10	9	8
RESERVED	tACC				tCOH		
7	6	5	4	3	2	1	0
tACS			tCOS			DBWD	

- **BASADDR [31 : 19] : Base address pointer of external I/O bank 0~3**
  The start address of each external I/O bank is calculated as "**BASADDR"** base pointer << 18. Each external I/O bank base address pointer together with the "**SIZE**" bits constitutes the whole address range of each external I/O bank.

- **SIZE [18 : 16] : The size of the external I/O bank 0~3**

SIZE [18 : 16]			Byte
0	0	0	**256K**
0	0	1	**512K**
0	1	0	**1M**
0	1	1	**2M**
1	0	0	**4M**
1	0	1	**8M**
1	1	0	**16M**
1	1	1	**32M**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

- **tACC [1411] : Access cycles （nOE or nWE active time）for external I/O bank 0~3**

tACC [14 : 11]				MCLK
0	0	0	0	**Reserved**
0	0	0	1	**1**
0	0	1	0	**2**
0	0	1	1	**3**
0	1	0	0	**4**
0	1	0	1	**5**
0	1	1	0	**6**
0	1	1	1	**7**

tACC [14 : 11]				MCLK
1	0	0	0	**9**
1	0	0	1	**11**
1	0	1	0	**13**
1	0	1	1	**15**
1	1	0	0	**17**
1	1	0	1	**19**
1	1	1	0	**21**
1	1	1	1	**23**

- **tCOH [10 : 8] : Chip selection hold-on time on nOE or nWBE for external I/O bank 0~3**

tCOH [10 : 8]			MCLK
0	0	0	**0**
0	0	1	**1**
0	1	0	**2**
0	1	1	**3**
1	0	0	**4**
1	0	1	**5**
1	1	0	**6**
1	1	1	**7**

- **tACS [7 : 5] : Address set-up before nECS for external I/O bank 0~3**

tACS [7 : 5]			MCLK
0	0	0	**0**
0	0	1	**1**
0	1	0	**2**
0	1	1	**3**
1	0	0	**4**
1	0	1	**5**
1	1	0	**6**
1	1	1	**7**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

- **tCOS [4 : 2] : Chip selection set-up time on nOE or nWBE for external I/O bank 0~3**
  When ROM/Flash memory bank is configured, the access to its bank stretches chip selection time before the nOE or new signal is activated.

tCOS [4 : 2]			MCLK
0	0	0	**0**
0	0	1	**1**
0	1	0	**2**
0	1	1	**3**
1	0	0	**4**
1	0	1	**5**
1	1	0	**6**
1	1	1	**7**

- **DBWD [1 : 0] : Programmable data bus width for external I/O bank 0~3**

DBWD [1 : 0]		Width of Data Bus
0	0	**Disable bus**
0	1	**8-bit**
1	0	**16-bit**
1	1	**32-bit**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

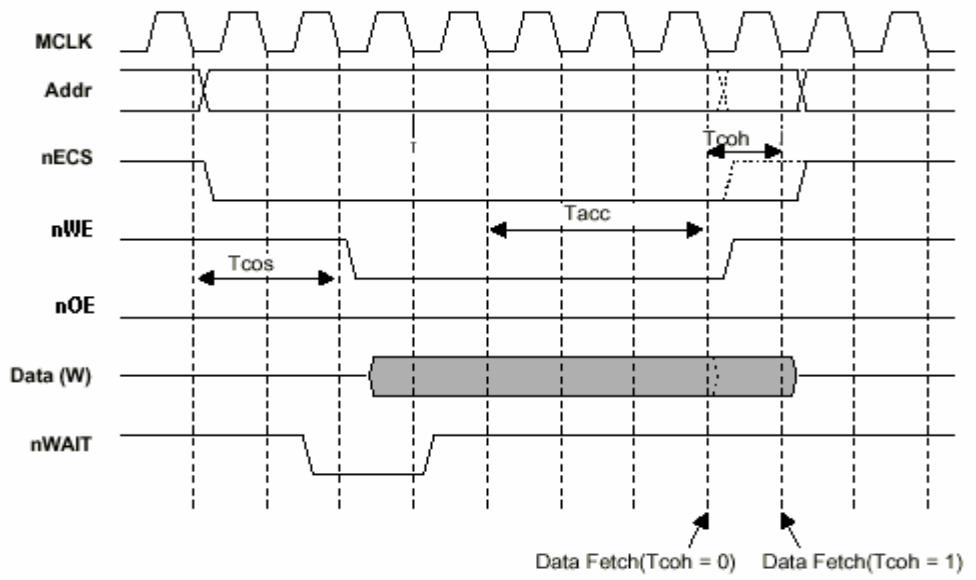


Figure 2-2 External I/O write operation timing

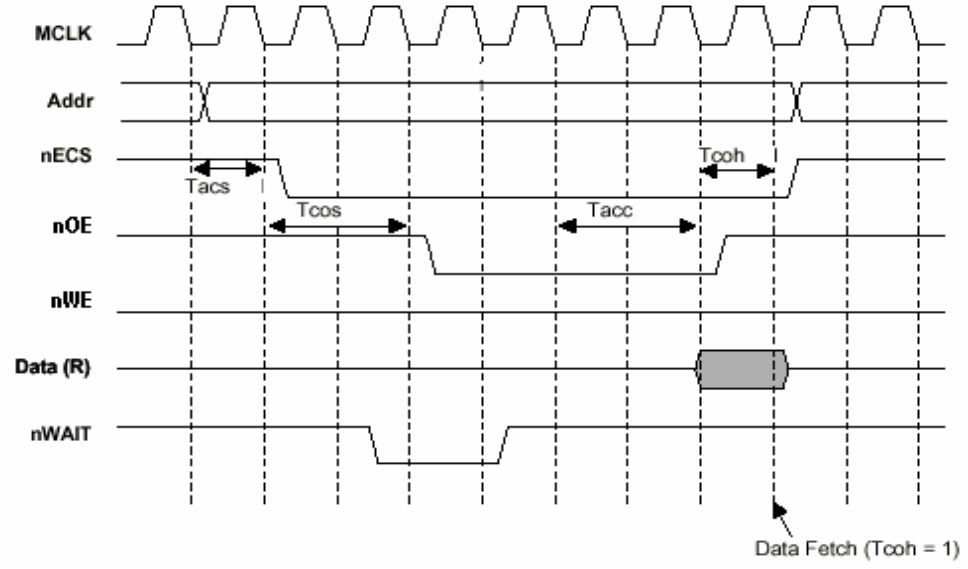


Figure 2-3 External I/O read operation timing

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2.4.5 A system memory initialization example flow chart

Figure 2-4 System Memory Map Setting Flow

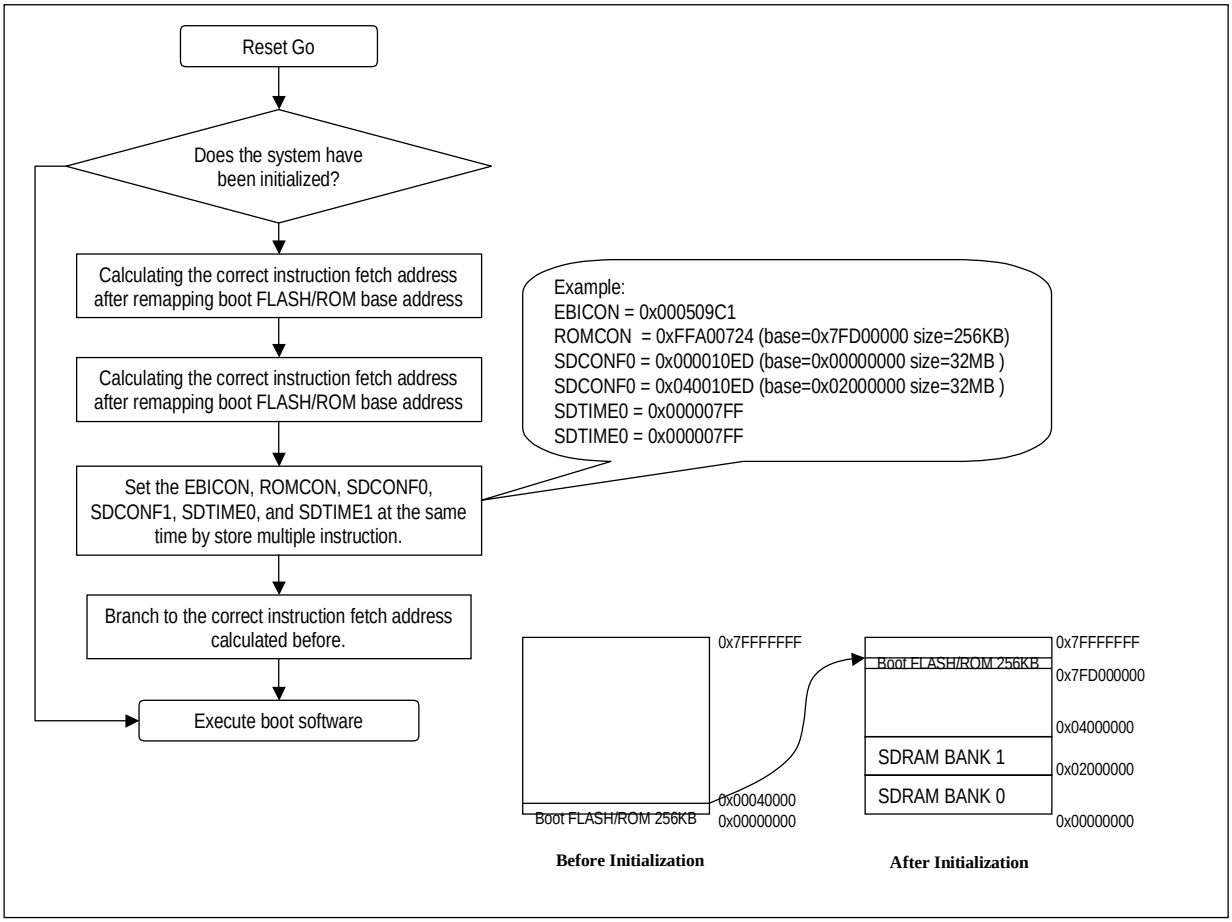


Figure 2-4 is the boot flow of Boot Monitor with remapping. The flow chart shows that most of the EBI control registers, EBICON, ROMCON, SDCONF, SDTIME should beware initialized as soon as possible immediately after reset. Each value of these control register must be known before these registers were configured. Because on doing of remapping, the control registers should be set by store multiple instructions(*STMIA*). The store multiple instructions guarantee to complete the memory initialization before next instruction execution. The system memory maps before initialization and after initialization are shown as above, too. After system reset, the system can access the 256KB boot FLASH/ROM. After the system initialization the memory map becomes the **After Initialization of** Figure 2-4

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2.4.6 REMAPPING

RAM is normally with faster access speed and wider than ROM. For this reason, it is better to store for the vector table and interrupt handlers if the memory on system address at 0x0 is of RAM. However, if RAM is located at address 0x0 on power-up, there is not a valid instruction in the reset vector(0x0) entry. Therefore, you must allow ROM to be located at 0x0 during normal execution. The changeover remapping from the reset to the normal memory map is normally caused by writing to a memory-mapped register.

In NUC740 the memory remapping can be achieved by setting EBI control registers. The following example is a MACRO, which achieves performs the remapping when booting. The program flow of this example is as Figure 2-4.

In general, the memory remapping only needs to be preformed once at reset. Thus the reset value of SDCONF0 is used to check if the system has been initialized. If the system memory needs to do remapping, the correct instructions to be fetched after remapping is important are critical. Therefore, the MACRO will calculate the correct instruction fetch address after remapping. It does this by using labels and program counter to know get the current execution position and execution position decided at link-time. If the current execution position is different to from execution position decided at link-time, the MACRO will calculate the correct instruction fetch address after remapping according to their address relation. Finally, the value of pc PC will be update immediately after remapping. Because the memory bases can be controlled by registers, what we called remapping means setting the EBI control registers. The configuration values are predefined as rEBICON, rROMCON, rSDCONF0, rSDCONF1, rSDTIME0, rSDTIME1 ,and are stored to relative control registers by store multiple instruction(STMIA). The source code is listed as follows :

```
; ------------------------------------------------------------------
; UNMAPROM
; --------
; Provide code to deal with mapping the reset ROM away from zero

     MACRO
$label  UNMAPROM     $w1,$w2

; This macro needs to test if the system has already been initialized
; The reset value of SDCONF0 is used to check if the system has been
initialized

     LDR   $w1, =SDCONF0 ;SDCONF0=0xFFF01008
     LDR   $w1, [$w1]
     LDR  $w2, =0x800
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
     CMP   $w1, $w2
     BNE   %FT0

; Set mode to SVC, interrupts disabled (just paranoid)
     MRS   $w1, cpsr
     BIC   $w1, $w1, #0x1F
     ORR   $w1, $w1, #0xD3
     MSR   cpsr_fc, $w1

; Configure the EBI controller to remap the flash

; The EBI Control Registers must be set using store multiples
; Set up a stack in internal SRAM to preserve the original register
contents

; Disable Cache and use the on-chip SRAM to be stack
     LDR  $w1, =CAHCNF         ; CAHCNF=0xFFF02000
     LDR  $w2, =0x0       ; SetValue = 0x0
     STR  $w2, [$w1]      ; Cache,WB disable

; NUC740_SRAM_BASE = 0x7FE00000
; NUC740_SRAM_SIZe = 10 Kbytes

     MOV  $w1, sp
     LDR  sp,  =(NUC740_SRAM_BASE+NUC740_SRAM_SIZE)

     STR  $w1, [sp, #-4]!; preserve previous sp on new stack
     STMFDsp!, {r0 - r12,lr}

; The labels “$label.temp” and “$label.EndSysMapJump are absolute
addresses
; calculated by linker according to the RO base.

     LDR  r2, =$label.temp
; The value of current pc is the run-time address of “$label.temp”
     MOV  r1, pc
     LDR r3, =$label.EndSysMapJump
$label.temp
     MOV lr, #0
; If r2 > r1, the system needs to remapping.
; The RO_base is NUC740_FLASH_BASE at link-time, thus we needs it to
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
; Calculate the correct instruction fetch address after remapping
     CMP  r2, r1
     LDRGTlr, =NUC740_FLASH_BASE

; Calculate the actual fetch address where is the location of
; the label “$label.EndSysMapJump” after remapping

     SUB  r3, r3, r2
     ADD  r1, r1, r3
     ADD  lr, lr, r1

; Load in the target values into the control registers
     ADRL    r0, $label.SystemInitData
     LDMIA   r0, {r1-r6}
     LDR  r0, =EBICON

; Now run critical jump code
     STMIA   r0, {r1-r6}
     MOV  pc, lr
$label.EndSysMapJump

; Now running from new PROM location, since code no longer exists in low
memory
; Restore registers
     LDMFDsp!, {r0 - r12,lr}
     LDR  $w1, [sp], #4
     MOV  sp,  $w1
     B    %FT0

$label.SystemInitData
     DCD rEBICON    ; REFEN=1,REFMOD=0,CLKEN=1,REFRAT=0x138,WAITVT=0

     DCD rROMCON ; base=NUC740_FLASH_BASE,size=256KB,BTSIZE=32bit,
                 ; tPA=8MCLK,tACC=8MCLK

     DCD rSDCONF0; base=0x0,size=32MB,MRSET=1,AUTOPR=1,LATENCY=3MCLK,
                 ; LENGTH=1Byte,COMPBK=2bank,DBWD=32bit,COLUM=8bit

     DCD rSDCONF1;
base=0x2000000,size=32MB,MRSET=1,AUTOPR=1,LATENCY=3MCLK,
                 ; LENGTH=1Byte,COMPBK=2bank,DBWD=32bit,COLUM=8bit
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

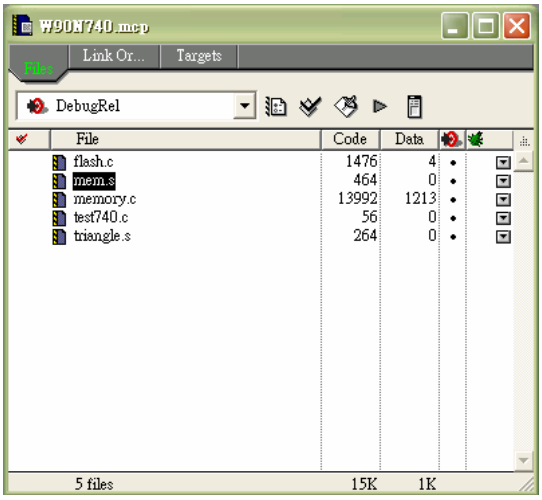
```
      DCD rSDTIME0 ; tRCD=8MCLK,tRDL=4MCLK,tRP=8MCLK,tRAS=8MCLK
      DCD rSDTIME1 ; tRCD=8MCLK,tRDL=4MCLK,tRP=8MCLK,tRAS=8MCLK
      ALIGN
0
         MEND
```

2.5 Example Code

2.5.1 PROJECT FILE

The Example Code is used to test if the demonstrate the EBI hardware functions of NUC740 works. For the EBI controller, the test example program is build by Code Warrior of ADS 1.1. The project name is called NUC740.mcp as Figure 2-5.

Figure 2-5 The Project File NUC740.MCP



The source files in this test program are described as follows :

NUC740.H :	The header file of this project.
flash.c :	The flash read/write command is implemented in this file.
mem.s :	Low-level memory access routines.
memory.c :	High-level memory access routines and EBI initialization codes.
test740.c :	Main function is in this file.
triangle.s :	Implementation of Pascal triangle function. This is used to test instruction fetch.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2.5.2 SOURCE CODE

2.5.2.1 NUC740.H

```
#ifndef NUC740_H
#define NUC740_H
//---------------------------------------------------------------------------
#define DEBUG_INFO
//---------------------------------------------------------------------------
#define BASE   0xc0000000

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int   DWORD;
typedef int    BOOL;
typedef void (*fptr)(void);  /* function pointer */
#define FALSE 0;
#define TRUE 1;

/* io function definition */
#define outp(port,value)       *((volatile BYTE *)(port))=value
#define outpdw(port,value)     *((volatile DWORD *)(port))=value
#define outpw(port,value)      outpdw(port,value)
#define inp(port)              *((volatile BYTE *)(port))
#define inpdw(port)    *((volatile DWORD *)(port))
#define inpw(port)     inpdw(port)

#ifdef DEBUG_INFO
#define msg(x) printf("\n");\
                              printf(x);
#else
#define msg(x)
#endif


/* AP Board registers */
#define AP_IRQ0_STATUS         0x14000000
#define AP_IRQ0_RAWSTAT        0x14000004
#define AP_IRQ0_ENABLESET      0x14000008
#define AP_IRQ0_ENABLECLR      0x1400000c

/* IRQ register bit assigments, 22 bits */
#define EXPINT0 0x000200 /* Logic module 0 interrupt */


/* LM registers */
#define LM_CLK0        0x0000000      /* 19 bits */
#define LM_CLK1        0x0000004      /* 19 bits */
#define LM_CLK_LOCK    0x0000008      /* 17 bits */
#define LM_LEDS        0x000000c      /* 9 bits  */
#define LM_BUTTON      0x0000010      /* 1 bit   */
#define LM_SWITCH      0x0000014      /* 8 bits  */
/* Clock control signal, Freq=48MHz*(nVCO_DIV+8)/(nREF_DIV+2)/nOUT_DIV */
                                                  /* LM_CLK[18 : 16],default=110b */
#define OUT_DIV2       0x10000
#define OUT_DIV4       0x30000
#define OUT_DIV5       0x40000
#define OUT_DIV7       0x50000
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define OUT_DIV8        0x20000
#define OUT_DIV9        0x60000 /* default */
#define OUT_DIV6        0x70000
#define OUT_DIV10       0x00000
#define REF_DIV         0x0fe00 /* LM_CLK[15 : 9],default=0111110b */
#define VCO_DIV         0x001ff /* LM_CLK[8 : 0],default=000000100b */



/* AIC registers */
#ifdef MAIN_PROG
DWORD AIC_SCR[32]={     BASE+0x0ff82004,
                        BASE+0x0ff82008,
                        BASE+0x0ff8200c,
                        BASE+0x0ff82010,
                        BASE+0x0ff82014,
                        BASE+0x0ff82018,
                        BASE+0x0ff8201c,
                        BASE+0x0ff82020,
                        BASE+0x0ff82024,
                        BASE+0x0ff82028,
                        BASE+0x0ff8202c,
                        BASE+0x0ff82030,
                        BASE+0x0ff82034,
                        BASE+0x0ff82038,
                        BASE+0x0ff8203c,
                        BASE+0x0ff82040,
                        BASE+0x0ff82044,
                        BASE+0x0ff82048,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                                        };
#else
extern AIC_SCR[32];
#endif
#define AIC_SCR1        (BASE+0x0FF82004)
#define AIC_SCR2        (BASE+0x0FF82008)
#define AIC_SCR3        (BASE+0x0FF8200C)
#define AIC_SCR4        (BASE+0x0FF82010)
#define AIC_SCR5        (BASE+0x0FF82014)
#define AIC_SCR6        (BASE+0x0FF82018)
#define AIC_SCR7        (BASE+0x0FF8201C)
#define AIC_SCR8        (BASE+0x0FF82020)
#define AIC_SCR9        (BASE+0x0FF82024)
#define AIC_SCR10       (BASE+0x0FF82028)
#define AIC_SCR11       (BASE+0x0FF8202C)
#define AIC_SCR12       (BASE+0x0FF82030)
#define AIC_SCR13       (BASE+0x0FF82034)
#define AIC_SCR14       (BASE+0x0FF82038)
#define AIC_SCR15       (BASE+0x0FF8203C)
#define AIC_SCR16       (BASE+0x0FF82040)
#define AIC_SCR17       (BASE+0x0FF82044)
#define AIC_SCR18       (BASE+0x0FF82048)

#define AIC_IRSR        (BASE+0x0ff82100)
#define AIC_IASR        (BASE+0x0ff82104)
#define AIC_ISR         (BASE+0x0ff82108)
#define AIC_IPER        (BASE+0x0ff8210c)
#define AIC_ISNR        (BASE+0x0ff82110)
#define AIC_IMR         (BASE+0x0ff82114)
#define AIC_OISR        (BASE+0x0ff82118)
#define AIC_MECR        (BASE+0x0ff82120)
#define AIC_MDCR        (BASE+0x0ff82124)
#define AIC_SSCR        (BASE+0x0ff82128)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define AIC_SCCR       (BASE+0x0ff8212c)
#define AIC_EOSCR      (BASE+0x0ff82130)


/* Source type */
#define LOW_LEVEL      0x0
#define HIGH_LEVEL     0x1
#define NEG_EDGE       0x2
#define POS_EDGE       0x3


/* Timer registers */
#define T_TCR0  (BASE+0x0ff81000)
#define T_TCR1  (BASE+0x0ff81004)
#define T_TICR0 (BASE+0x0ff81008)
#define T_TICR1 (BASE+0x0ff8100c)
#define T_TDR0  (BASE+0x0ff81010)
#define T_TDR1  (BASE+0x0ff81014)
#define T_TISR  (BASE+0x0ff81018)
#define T_WTCR  0x0ff8101c
/* Status flag */
#define TIF0    0x1
#define TIF1    0x2
/* Mode */
#define ONE_SHOT       0x00000000
#define PERIOD  0x08000000
#define TOGGLE  0x10000000
#define IE      0x40000000
#define CE      0x20000000
#define WTTME   0x00000100
#define WTE     0x00000080
#define WTIE    0x00000040
#define WTIS    0x00000030
#define WTIF    0x00000008
#define WTRF    0x00000004
#define WTRE    0x00000002
#define WTR     0x00000001

/*----- EBI control register definitions -----*/
#define EBICON         (BASE+0xFF01000)             /* EBI control register */
#define ROMCON         (BASE+0xFF01004)             /* ROM/FLASH control register */
#define SDCONF0        (BASE+0xFF01008)             /* SDRAM bank 0 configuration register */
#define SDCONF1        (BASE+0xFF0100C)             /* SDRAM bank 1 configuration register */
#define SDTIME0        (BASE+0xFF01010)             /* SDRAM bank 0 timing control register */
#define SDTIME1        (BASE+0xFF01014)             /* SDRAM bank 1 timing control register */
#define EXT0CON        (BASE+0xFF01018)             /* External I/O 0 control register */
#define EXT1CON        (BASE+0xFF0101C)             /* External I/O 1 control register */
#define EXT2CON        (BASE+0xFF01020)             /* External I/O 2 control register */
#define EXT3CON        (BASE+0xFF01024)             /* External I/O 3 control register */
#define CKSKEW         (BASE+0xFF01F00)             /* clock skew control register (for testing)*/
/*----- GDMA control register -----*/
#define GDMA_CTL0      (BASE+0xFF04000)
#define GDMAERR (0x1<<20)
#define AUTOIEN (0x1<<19)
#define TC             (0x1<<18)
#define BLOCK          (0x1<<17)
#define SOFTREQ (0x1<<16)
#define DM             (0x1<<15)
#define TWS            (0x3<<12)
#define SBMS           (0x1<<11)
#define BME            (0x1<<9)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define SIEN            (0x1<<8)
#define SAFIX           (0x1<<7)
#define DAFIX           (0x1<<6)
#define SADIR           (0x1<<5)
#define DADIR           (0x1<<4)
#define GDMAMS          (0x3<<2)
#define GDMAEN          (0x1)


#define GDMA_SRCB0      (BASE+0xFF04004)
#define GDMA_DSTB0      (BASE+0xFF04008)
#define GDMA_TCNT0      (BASE+0xFF0400C)
#define GDMA_CSRC0      (BASE+0xFF04010)
#define GDMA_CDST0      (BASE+0xFF04014)
#define GDMA_CTCNT0     (BASE+0xFF04018)
#define GDMA_CTL1       (BASE+0xFF04020)
#define GDMA_SRCB1      (BASE+0xFF04024)
#define GDMA_DSTB1      (BASE+0xFF04028)
#define GDMA_TCNT1      (BASE+0xFF0402C)
#define GDMA_CSRC1      (BASE+0xFF04030)
#define GDMA_CDST1      (BASE+0xFF04034)
#define GDMA_CTCNT1     (BASE+0xFF04038)


#define GPIO_CFG        (BASE+0xFF83000)
#define GPIO_DIR        (BASE+0xFF83004)
#define GPIO_DATAOUT    (BASE+0xFF83008)
#define GPIO_DATAIN     (BASE+0xFF8300C)
#define DEBNCE_CTRL     (BASE+0xFF83010)

//---------------------------------------------------------------------------
#endif
```

2.5.2.2 FLASH.C

```
#include <stdio.h>
#include "NUC740.H"
#define FLASH_ADDR_OFFSET 1
unsigned int flash_id=0xFFFF;
extern void *beginAddr;
extern void *endAddr;
extern unsigned int bit;

union unit_t{
        unsigned int word;
        unsigned char b[4];
        unsigned short h[2];
};
extern union unit_t unit;

void ChipErase(void);
void FlashMemoryContent(void);
void ProgramFlash(void);
void ProdId(void);
void FlashWrite(void *addr,unsigned int data,unsigned int w);
unsigned int FlashRead(void *addr,unsigned int r);


void ProdId(void)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
        unsigned int cnt;
        unsigned int rdata1,rdata2,rdata3;
        unsigned int flash_io_base;
        if(flash_id==0xFFFF)return;
        flash_io_base=(unsigned int) beginAddr;
        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0xAAAA;
        *((volatile unsigned short *)(flash_io_base+(0x2AAA<<FLASH_ADDR_OFFSET)))=0x5555;
        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x9090;
        cnt=0;
        while(cnt++<0x100);
        rdata1=*((volatile unsigned short *)beginAddr);
        rdata2=*((volatile unsigned short *)beginAddr+1);
        rdata3=*((volatile unsigned short *)beginAddr+2);

        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0xAAAA;
        *((volatile unsigned short *)(flash_io_base+(0x2AAA<<FLASH_ADDR_OFFSET)))=0x5555;
        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0xF0F0;
        cnt=0;
        while(cnt++<0x100);

        printf("\nManufacturer code=0x%08x",rdata1);
        printf("\nDevice code=0x%08x",rdata2);
        printf("\nBoot block lockout=0x%08x",(rdata3&0x1));
}

void FlashMemoryContent(void)
{

        int cnt=0x1000;
        int i=0;
        volatile unsigned short *addr;

        if(flash_id==0xFFFF)return;

        printf("\nPlease input the size to be showed(hex) : ");
        scanf("%x",&cnt);

        cnt=cnt/2;
        addr=(unsigned short *)beginAddr;
        for(addr=(unsigned short *)beginAddr;addr<(unsigned short *)endAddr;addr++)
        {
                if( (i&0x7)== 0 )
                {
                        printf("\n0x%08x : ",(unsigned int)addr);
                }
                printf(" %04x",*addr);
                i++;
                if(i>=cnt)break;
        }
}

void ProgramFlash(void)
{
        volatile unsigned int *addr;
        unsigned int base;
        unsigned int ecnt=0;
        int cnt;
        unsigned int data,rdata;
        FILE *fp;

        if(flash_id==0xFFFF)return;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        if(beginAddr==endAddr)
        {printf("\nMemory not initialized....");return;}

        ChipErase();


        if( (fp=fopen("..\\..\\ROM_Data\\ROM\\ROM.bin","rb"))==NULL )
        {
                printf("\nCan't open file ROM.bin");
                return;
        }

        rewind(fp);
        base=(unsigned int )beginAddr;
        while( fread(&rdata,4,1,fp)!=0 )
        {
                printf("\naddr=0x%08x write=0x%08x",base,rdata);
                FlashWrite((void *)base,rdata,2);
                base+=4;
        }
        printf("\nEnd of programming...");
        printf("\nVerifying...");
        base=(unsigned int )beginAddr;
        rewind(fp);
        while( fread(&rdata,4,1,fp)!=0 )
        {
                data=*((volatile unsigned int *)base);
                printf("\naddr=0x%08x data=0x%08x",base,data);
                if( data!=rdata )
                {
                        printf("\nError :  addr=0x%08x write=0x%08x read=0x%08x",base,rdata,data);
                        ecnt++;
                }
                base+=4;
        }
        printf("\nEnd of verifying...");

        fclose(fp);
        return;


        printf("\nProgramming...");
        data=0xC6000000;
        for(addr=(unsigned int *)beginAddr;addr<(unsigned int *)endAddr;addr++)
        {
                data++;
                FlashWrite((void *)addr,data,2);
        }

        printf("\nVerifying...");
        cnt=0;
        data=0xC6000000;
        for(addr=(unsigned int *)beginAddr;addr<(unsigned int *)endAddr;addr++)
        {
                data++;
                rdata=FlashRead((void *)addr,2);
                if( data!=rdata )
                {
                        printf("\nError :  Address=0x%08x write=0x%08x read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        }


        if(cnt==0)
                printf("\n****** Flash Programming PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);

}



void ChipErase(void)
{
        unsigned int flash_io_base;
        unsigned short data;

        if(flash_id==0xFFFF)return;
        flash_io_base=(unsigned int)beginAddr;
        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x00AA;
        *((volatile unsigned short *)(flash_io_base+(0x2AAA<<FLASH_ADDR_OFFSET)))=0x0055;
        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x0080;
        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x00AA;
        *((volatile unsigned short *)(flash_io_base+(0x2AAA<<FLASH_ADDR_OFFSET)))=0x0055;
        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x0010;
        do
        {
                data=*((volatile unsigned short *)(flash_io_base));
        }while(!(data&0x80));/* Data Polling */
}


void FlashWrite(void *addr,unsigned int data,unsigned int w)
{
        unsigned int flash_io_base;
        flash_io_base=(unsigned int)beginAddr;
        unit.word=data;
        switch(w)
        {
                case 1 :
                        printf("\nUn-supported!!");
                        break;
                case 2 :
                        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x00AA;
                        *((volatile unsigned short *)(flash_io_base+(0x2AAA<<FLASH_ADDR_OFFSET)))=0x0055;
                        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x00A0;
                        *((volatile unsigned short *)addr)=unit.h[1];
                        while( ((*((volatile unsigned short *)addr))&0x80)!=(unit.h[1]&0x80) );

                        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x00AA;
                        *((volatile unsigned short *)(flash_io_base+(0x2AAA<<FLASH_ADDR_OFFSET)))=0x0055;
                        *((volatile unsigned short *)(flash_io_base+(0x5555<<FLASH_ADDR_OFFSET)))=0x00A0;
                        *((volatile unsigned short *)addr+1)=unit.h[0];
                        while( ((*((volatile unsigned short *)addr+1))&0x80)!=(unit.h[0]&0x80) );

                        break;
                case 4 :
                        printf("\nUn-supported!!");
                        break;
                default :
                        printf("\nUn-supported!!");
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                        break;
        }

}

unsigned int FlashRead(void *addr,unsigned int r)
{
        unsigned int rdata;

        switch(r)
        {
                case 1 :
                        unit.b[0]=*((volatile unsigned char *)addr);
                        unit.b[1]=*((volatile unsigned char *)addr+1);
                        unit.b[2]=*((volatile unsigned char *)addr+2);
                        unit.b[3]=*((volatile unsigned char *)addr+3);
                        rdata=unit.word;
                        break;
                case 2 :
                        unit.h[0]=*((volatile unsigned short *)addr);
                        unit.h[1]=*((volatile unsigned short *)addr+1);
                        rdata=unit.word;
                        break;
                case 4 :
                        rdata=*((volatile unsigned int *)addr);
                        rdata=unit.word;
                        break;
                default :
                        return 0;
        }

        return rdata;

}
```

2.5.2.3 MEM.S

```
        EXPORT  SeqReadWrite32
        EXPORT  StmFill
        EXPORT  SwapMemory


        AREA |Memory$$code$$NUC740|, CODE, READONLY
SeqReadWrite32
        STMFD   sp!,{r2-r12,lr}
 IF {TRUE}
        LDR     r2,=0x3456789A
        LDR     r3,=0x456789AB
        LDR     r4,=0x56789ABC
        LDR     r5,=0x6789ABCD
        LDR     r6,=0x789ABCDE
        LDR     r7,=0x89ABCDEF
        LDR     r8,=0x9ABCDEF0
        LDR     r9,=0xABCDEF01
 ENDIF
 IF {FALSE}
        LDR     r2,=0x55AAFF11
        LDR     r3,=0x55AAFF11
        LDR     r4,=0x55AAFF11
        LDR     r5,=0x55AAFF11
        LDR     r6,=0x55AAFF11
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        LDR     r7,=0x55AAFF11
        LDR     r8,=0x55AAFF11
        LDR     r9,=0x55AAFF11
 ENDIF

 IF {FALSE}
        LDR     r2,=0
        LDR     r3,=0
        LDR     r4,=0
        LDR     r5,=0
        LDR     r6,=0
        LDR     r7,=0
        LDR     r8,=0
        LDR     r9,=0
 ENDIF


loop
        STMIA   r0,{r2-r9}
        LDMIA   r0!,{r2-r9}
        SUBS    r1,r1,#1
        BNE     loop

        LDMFD   sp!,{r2-r12,pc}


        ;r0=start addrss
        ;r1=n bytes
        ;r2=fill word

; r0=source r1=destination r2=fill word
StmFill
        STMFD   sp!,{r3-r12,lr}
        BIC     r0,r0,#3
        BIC     r1,r1,#3

        MOV     r3,r2
        MOV     r4,r2
        MOV     r5,r2
        MOV     r6,r2
        MOV     r7,r2
        MOV     r8,r2
        MOV     r9,r2
        MOV     r10,r2
        MOV     r11,r2
        MOV     r12,r2
0
        SUBS    r1,r1,#44
        STMGTIA r0!,{r2-r12}
        BGT     %0
        BEQ     %3
1
        ADD     r1,r1,#44
2
        STR     r2,[r0],#4
        SUBS    r1,r1,#4
        BNE     %2
3
        LDMFD   sp!,{r3-r12,pc}

;verify ok return 0
;verify fail return -1
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
;r1 is counted in word

VerifyFillMemory
        STMFD   sp!,{r3-r12,lr}

        BIC     r0,r0,#3
        BIC     r1,r1,#3

4
        LDMIA   r0!,{r3-r10}
        CMP     r2,r3
        CMPEQ   r2,r4
        CMPEQ   r2,r5
        CMPEQ   r2,r6
        CMPEQ   r2,r7
        CMPEQ   r2,r8
        CMPEQ   r2,r9
        CMPEQ   r2,r10
        MOVNE   r0,#-1
        BNE     %7
        SUBS    r1,r1,#32
        BLO     %5
        BNE     %4
        MOVEQ   r0,#0
        BEQ     %7
5
        ADD     r1,r1,#32
6
        LDR     r3,[r0],#4
        CMP     r2,r3
        MOVNE   r0,#-1
        BNE     %7
        SUBS    r1,r1,#4
        BNE     %6
        MOVEQ   r0,#0
7
        LDMFD   sp!,{r3-r12,pc}


VerifyFillMemoryH
        STMFD   sp!,{r3-r12,lr}
        BIC     r0,r0,#1
        BIC     r1,r1,#1
9
        LDRH    r3,[r0],#2
        CMP     r2,r3
        MOVNE   r0,#-1
        BNE     %8
        SUBS    r1,r1,#2
        BNE     %9
        MOVEQ   r0,#0
8
        LDMFD   sp!,{r3-r12,pc}

VerifyFillMemoryB
        STMFD   sp!,{r3-r12,lr}
        BIC     r0,r0,#1
        BIC     r1,r1,#1
11
        LDRB    r3,[r0],#1
        CMP     r2,r3
        MOVNE   r0,#-1
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        BNE     %10
        SUBS    r1,r1,#1
        BNE     %11
        MOVEQ   r0,#0
10
        LDMFD   sp!,{r3-r12,pc}

VerifyMemory
        STMFD   sp!,{r3-r12,lr}
13
        LDRB    r3,[r0],#1
        LDRB    r4,[r1],#1
        CMP     r3,r4
        MOVNE   r0,#-1
        BNE     %12
        SUBS    r2,r2,#1
        BNE     %13
        MOVEQ   r0,#0
12
        LDMFD   sp!,{r3-r12,pc}

; r0=addr1 r1=addr2 r2=size

SwapMemory
        STMFD   sp!,{r4-r8,lr}
        BIC     r8,r2,#3
        BIC     r7,r1,#3
        BIC     r6,r0,#3
13
        LDR     r4,[r6]
        SWP     r5,r4,[r7]
        SWP     r4,r5,[r6]
        ADD     r6,r6,#4
        ADD     r7,r7,#4
        SUBS    r8,r8,#4
        BNE     %b13

        LDMFD   sp!,{r4-r8,pc}



        END
```

2.5.2.4 MEMORY.C

```
#include <stdio.h>
//#include <rt_misc.h>

#include "NUC740.H"

void InitExtIo(void);
void InitSdram(void);
void InitRom(void);
void SingleAccess(void);
void WnRn(void);
void MemBitsTest(void);
void EntireRamTest(void);
void AccessTypeTest(void);
void SerialAccess(void);
void SingleAccessLoop(void);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
void SetSystemClk(void);
void SpecialSramTest1(void);
void SpecialSramTest2(void);
void RWTypeTest(void);
void RandomRWTest(void);
void BaseAddrTest(void);
void SelfPrechargeTest(void);
void ChangeBurstLength(void);
void WordBoundaryTest(void);
void LoadStoreMultTest(void);
void AutoPrechargeEnable(void);
void InstructionFetchTest(void);
void ResetValueTest(void);
void PowerDownTest(void);
void AutoInit(void);
void SwapMemoryTest(void);
void EBIregisterRWTest(void);
void EndianFlagTest(void);
void debug(void);
void ButterflyTest(void);
unsigned int ReadMem(volatile void *addr,int r);
unsigned int WriteMem(volatile void *addr,unsigned int data,int w);
int vpascal_generator (int n, int i);
void vpascal_triangle (int depth,int tube);
extern void FlashMemoryContent(void);
extern void ProgramFlash(void);
extern unsigned int flash_id;
extern void ProdId(void);
extern void ChipErase(void);
extern void SeqReadWrite32(unsigned int *addr, unsigned int n);
extern void SwapMemory(unsigned int addr1,unsigned int addr2,unsigned int size);
extern void far_exe(unsigned int base,unsigned int rwbase,unsigned int depth);

volatile void * beginAddr=0;
volatile void * endAddr=0;
#define bit 14

union unit_t{
        unsigned int word;
        unsigned char b[4];
        unsigned short h[2];
} unit;

typedef struct _test_func{
        int func;
        char *name;
        void (*prog)(void);
} testfunc_t;

testfunc_t TestFuncTable[]=
{
        {1,"Initialize External IO.",InitExtIo},
        {2,"Initialize SDRAM.",InitSdram},
        {3,"FLASH/ROM Initialization.",InitRom},
        {4,"Reset value test.",ResetValueTest},
        {5,"Power down test.",PowerDownTest},
        {6,"RAM R/W type tests.",RWTypeTest},
        {7,"Entire RAM test.",EntireRamTest},
        {8,"Random R/W test.",RandomRWTest},
        {9,"Self-precharge test.",SelfPrechargeTest},
        {10,"Load/Store multiple test.",LoadStoreMultTest},
        {11,"Instruction fetch test.",InstructionFetchTest},
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        {12,"Word boundary test.",WordBoundaryTest},
        {13,"Auto precharge enable",AutoPrechargeEnable},
        {14,"BASADDR test.",BaseAddrTest},
        {15,"Product ID",ProdId},
        {16,"Flash memory content.",FlashMemoryContent},
        {17,"Chip Erase.",ChipErase},
        {18,"Programming flash memory.",ProgramFlash},
        {19,"Read memory controller endian mode.",EndianFlagTest},
        {20,"EBI control registers read/write test.",EBIregisterRWTest},
        {21,"Butterfly test.",ButterflyTest}
};


void ButterflyTest(void)
{
        unsigned int addr1,addr2,midsize;
        unsigned int data[7]={0,0x55555555,0xaaaaaaaa,0xffffffff,0xaaaaaaaa,0x55555555,0};
        unsigned int i;
        unsigned int ecnt=0;
        unsigned int rdata;

        addr1=(unsigned int)beginAddr;
        addr2=(unsigned int)endAddr;
        midsize=(addr2-addr1)/2;


        //-------------------------------------------------------------------
        printf("\nButterfly test 1...");
        for(i=0;i<midsize;i+=4)
        {
                *((volatile unsigned int *)(addr1+i))=data[(i>>2)%4];
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr1+i);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                *((volatile unsigned int *)(addr2-i-4))=data[(i>>2)%4];
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr2-i-4);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                rdata=*((volatile unsigned int *)(addr1+i));
                if(rdata!=data[(i>>2)%4])
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr1+i,data[(i>>2)%4],rdata);
                        ecnt++;
                }
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr1+i);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                rdata=*((volatile unsigned int *)(addr2-i-4));
                if(rdata!=data[(i>>2)%4])
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr2-i-4,data[(i>>2)%4],rdata);
                        ecnt++;
                }
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr2-i-4);
                }
        }
        //-------------------------------------------------------------------
        printf("\nButterfly test 2...");
        for(i=0;i<midsize;i+=4)
        {
                *((volatile unsigned int *)(addr1+midsize-i-4))=data[(i>>2)%4+1];
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr1+midsize-i-4);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                *((volatile unsigned int *)(addr2-midsize+i))=data[(i>>2)%4+1];
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr2-midsize+i);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                rdata=*((volatile unsigned int *)(addr1+midsize-i-4));
                if(rdata!=data[(i>>2)%4+1])
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr1+midsize-i-
4,data[(i>>2)%4+1],rdata);
                        ecnt++;
                }
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr1+midsize-i-4);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                rdata=*((volatile unsigned int *)(addr2-midsize+i));
                if(rdata!=data[(i>>2)%4+1])
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr2-
midsize+i,data[(i>>2)%4],rdata);
                        ecnt++;
                }
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x",addr2-midsize+i);
                }
        }
        //-------------------------------------------------------------------
        printf("\nButterfly test 3...");
        for(i=0;i<midsize;i+=4)
        {
                *((volatile unsigned int *)(addr1+midsize-i-4))=data[(i>>2)%4+2];
                *((volatile unsigned int *)(addr2-midsize+i))=data[(i>>2)%4+2];
                if( (i&0x000FFFFF)==0 )
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                {
                        printf("\naddr=0x%08x 0x%08x",addr1+midsize-i-4,addr2-midsize+i);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                rdata=*((volatile unsigned int *)(addr1+midsize-i-4));
                if(rdata!=data[(i>>2)%4+2])
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr1+midsize-i-
4,data[(i>>2)%4+2],rdata);
                        ecnt++;
                }
                        rdata=*((volatile unsigned int *)(addr2-midsize+i));
                if(rdata!=data[(i>>2)%4+2])
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr2-
midsize+i,data[(i>>2)%4+2],rdata);
                        ecnt++;
                }
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x 0x%08x",addr1+midsize-i-4,addr2-midsize+i);
                }
        }
        //-----------------------------------------------------------------
        printf("\nButterfly test 4...");
        for(i=0;i<midsize;i+=4)
        {
                *((volatile unsigned int *)(addr1+i))=data[(i>>2)%4+3];
                *((volatile unsigned int *)(addr2-i-4))=data[(i>>2)%4+3];
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x 0x%08x",addr1+i,addr2-i-4);
                }
        }
        for(i=0;i<midsize;i+=4)
        {
                rdata=*((volatile unsigned int *)(addr1+i));
                if(rdata!=data[(i>>2)%4+3])
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr1+i,data[(i>>2)%4+3],rdata);
                        ecnt++;
                }
                        rdata=*((volatile unsigned int *)(addr2-i-4));
                if(rdata!=data[(i>>2)%4+3])
                {
                        printf("\nError : addr=0x%08x write=0x%08x read=0x%08x",addr2-i-
4,data[(i>>2)%4+3],rdata);
                        ecnt++;
                }
                if( (i&0x000FFFFF)==0 )
                {
                        printf("\naddr=0x%08x 0x%08x",addr1+i,addr2-i-4);
                }
        }

        if( ecnt== 0)
                printf("\nButterfly test PASSED!!");
        else
                printf("\nThere are %u failures!!",ecnt);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
}



void EndianFlagTest(void)
{
        if( *((volatile unsigned int *)EBICON)&1 )
                printf("\nMemory contoller is Little Endian.");
        else
                printf("\nMemory contoller is Big Endian.");
}


void EBIregisterRWTest(void)
{
        volatile unsigned int *p;
        unsigned int i;
        unsigned int data;
        char RegName[11][8]={"EBICON","ROMCON","SDCONF0","SDCONF1",
        "SDTIME0","SDTIME1","EXT0CON","EXT1CON","EXT2CON","EXT3CON","CKSKEW"};

        data=0xFFFFFFFF;



        p=(unsigned int *)EBICON;

        printf("\nReset value of EBI control registers");
        for(i=0;i<11;i++)
        {
                printf("\n%s=0x%08x",RegName[i],*(p+i));
        }
        printf("\nWrite 0x%08x to each register...",data);
        for(i=0;i<11;i++)
        {
                *(p+i)=data;
                printf("\n%s=0x%08x",RegName[i],*(p+i));
        }
}


void SwapMemoryTest(void)
{
        unsigned int addr1,addr2;
        unsigned int size,i;
        unsigned int data1,data2,ecnt=0;

        addr1=(unsigned int)beginAddr;
        addr2=addr1+0x10000;

        size=0x2048;
        for(i=0;i<size;i+=4)
        {
                *((unsigned int *)(addr1+i))=i;
                *((unsigned int *)(addr2+i))=size-i-4;
        }
        SwapMemory(addr1,addr2,size);
        for(i=0;i<size;i+=4)
        {
                data1=*((unsigned int *)(addr1+i));
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                data2=*((unsigned int *)(addr2+i));

                if( (data1!=size-i-4)||(data2!=i) )
                {
                        printf("\nMemory swap error");
                        ecnt++;
                }
        }

        if( ecnt==0 )
                printf("\nMemory swap test PASSED!!");
        else
                printf("\nThere are %u failures!!",ecnt);
}


void debug(void)
{
        unsigned int data0,data1,data2;
        data0=(unsigned int)beginAddr;
        data1=0x12345678;
        data2=0x87654321;
        __asm{
                MOV             r0,data0
                MOV             r1,data1
                MOV             r2,data2
                STMIA   r0!,{r1,r2}
                LDMDB   r0!,{r3,r4}
        }
}

void ResetValueTest(void)
{
        unsigned int cnt;

        cnt=0;
        if(*((volatile unsigned int *)EBICON)!=0x00010000)
        {
                printf("\nEBICON=0x%08x Reset Value=0x00010000",*((volatile unsigned int *)EBICON));
                cnt++;
        }
        if(*((volatile unsigned int *)ROMCON)!=0x000000FC)
        {
                printf("\nROMCON=0x%08x Reset Value=0x000000FC",*((volatile unsigned int *)ROMCON));
                cnt++;
        }
        if(*((volatile unsigned int *)SDCONF0)!=0x00000800)
        {
                printf("\nSDCONF0=0x%08x Reset Value=0x00000800",*((volatile unsigned int *)SDCONF0));
                cnt++;
        }
        if(*((volatile unsigned int *)SDCONF1)!=0x00000800)
        {
                printf("\nSDCONF1=0x%08x Reset Value=0x00000800",*((volatile unsigned int *)SDCONF1));
                cnt++;
        }
        if(*((volatile unsigned int *)SDTIME0)!=0x00000000)
        {
                printf("\nSDTIME0=0x%08x Reset Value=0x00000000",*((volatile unsigned int *)SDTIME0));
                cnt++;
        }
        if(*((volatile unsigned int *)SDTIME1)!=0x00000000)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        {
                printf("\nSDTIME1=0x%08x Reset Value=0x00000000",*((volatile unsigned int *)SDTIME1));
                cnt++;
        }
        if(*((volatile unsigned int *)EXT0CON)!=0x00000000)
        {
                printf("\nEXT0CON=0x%08x Reset Value=0x00000000",*((volatile unsigned int *)EXT0CON));
                cnt++;
        }
        if(*((volatile unsigned int *)EXT1CON)!=0x00000000)
        {
                printf("\nEXT1CON=0x%08x Reset Value=0x00000000",*((volatile unsigned int *)EXT1CON));
                cnt++;
        }
        if(*((volatile unsigned int *)EXT2CON)!=0x00000000)
        {
                printf("\nEXT2CON=0x%08x Reset Value=0x00000000",*((volatile unsigned int *)EXT2CON));
                cnt++;
        }
        if(*((volatile unsigned int *)EXT3CON)!=0x00000000)
        {
                printf("\nEXT3CON=0x%08x Reset Value=0x00000000",*((volatile unsigned int *)EXT3CON));
                cnt++;
        }
        if(*((volatile unsigned int *)CKSKEW)!=0x00000000)
        {
                printf("\nCKSKEW=0x%08x Reset Value=0x00000000",*((volatile unsigned int *)CKSKEW));
                cnt++;
        }
        if(cnt==0)
                printf("\n**** Reset Value Test PASSED!! ****");
        else
                printf("\nThere were %u failures occurred.",cnt);
}


void InstructionFetchTest(void)
{
        unsigned base,rwbase,depth;

        base=(unsigned int)beginAddr;
        rwbase=(unsigned int)beginAddr+0x1000;
        depth=6;
        far_exe(base,rwbase,depth);
        vpascal_triangle(depth,rwbase);
}

void PowerDownTest(void)
{
        volatile unsigned int cnt;
        unsigned int data=0x1a3b5c7d;
        unsigned int rdata,i,ecnt=0;
        for(i=0;i<0x100;i++)
        {
                *((volatile unsigned int *)beginAddr+i)=data;
        }
        *((volatile unsigned int *)SDCONF0)|=(1<<18); // REFEN
        *((volatile unsigned int *)SDCONF0)&=(~(1<<17));// auto refresh mode
        *((volatile unsigned int *)SDCONF0)&=(~(1<<16));// disable CLKEN
        *((volatile unsigned int *)SDCONF1)|=(1<<18); // REFEN
        *((volatile unsigned int *)SDCONF1)&=(~(1<<17));// auto refresh mode
        *((volatile unsigned int *)SDCONF1)&=(~(1<<16));// disable CLKEN
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        for(cnt=0;cnt<50000000;)
        {
                cnt++;
        }
        printf("\nSDCONF0=0x%08x  SDCONF1=0x%08x",*((volatile unsigned int *)SDCONF0),*((volatile unsigned int
*)SDCONF1));
        *((volatile unsigned int *)SDCONF0)|=(1<<16);// enable CLKEN
        *((volatile unsigned int *)SDCONF1)|=(1<<16);// enable CLKEN

        for(i=0;i<0x100;i++)
        {
                rdata=*((volatile unsigned int *)beginAddr+i);
                if( rdata!=data )
                {
                        printf("\nError :  Addr=0x%08x Write=0x%08x Read=0x%08x",(unsigned
int)beginAddr+i*4,data,rdata);
                        ecnt++;
                }
        }

        if( ecnt==0 )
                printf("\nPower down test PASSED!!");
        else
                printf("\nThere are %u failures",ecnt);
}

void AutoPrechargeEnable(void)
{
        if(*((volatile unsigned int *)SDCONF0)&(1<<13))
        {
                *((volatile unsigned int *)SDCONF0)&=(~(1<<13));
                *((volatile unsigned int *)SDCONF1)&=(~(1<<13));
        }
        else
        {
                *((volatile unsigned int *)SDCONF0)|=(1<<13);
                *((volatile unsigned int *)SDCONF1)|=(1<<13);
        }

        printf("\nSDCONF0=0x%08x",*((volatile unsigned int *)SDCONF0));
        printf("\nSDCONF1=0x%08x",*((volatile unsigned int *)SDCONF1));

}

void LoadStoreMultTest(void)
{
        extern void StmFill(unsigned int addr,unsigned int size, unsigned int data);
        unsigned int i;
        unsigned int cnt=0;
        unsigned int data=0;

        printf("\nInput fill data : ");
        scanf("%x",&data);
        /* Fill memory test */
        StmFill((unsigned int)beginAddr,4096,data);
        for(i=0;i<4096;i+=4)
        {
                if( *((unsigned int *)((unsigned int)beginAddr+i))!=data )
                {
                        printf("\nError :  addr=0x%08x write=0x%08x read=0x%08x",(unsigned
int)beginAddr+i,data,*((unsigned int *)((unsigned int)beginAddr+i)));
                        cnt++;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                }
        }



        if(cnt==0)
                printf("\n****** Store multiple test PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);

}


void WordBoundaryTest(void)
{
        unsigned int addr;
        unsigned int data,rdata0,rdata1,rdata2,rdata3;


        addr=(unsigned int)beginAddr;
        data=0x12345678;
        *((volatile unsigned int *)addr)=data;
        printf("\naddr(0x%08x)=data(0x%08x)",addr,data);
        printf("\nReading with offset #0,#1,#2,#3");
        rdata0=*((volatile unsigned int *)(addr+0));
        printf("\naddr(0x%08x)=data(0x%08x)",addr+0,rdata0);
        rdata1=*((volatile unsigned int *)(addr+1));
        printf("\naddr(0x%08x)=data(0x%08x)",addr+1,rdata1);
        rdata2=*((volatile unsigned int *)(addr+2));
        printf("\naddr(0x%08x)=data(0x%08x)",addr+2,rdata2);
        rdata3=*((volatile unsigned int *)(addr+3));
        printf("\naddr(0x%08x)=data(0x%08x)",addr+3,rdata3);

        printf("\nWrite with offset #1,#2,#3");
        *((volatile unsigned int *)(addr+1))=data;
        rdata0=*((volatile unsigned int *)addr);
        printf("\nWrite to addr(0x%08x) :  Read without offset data(0x%08x)",addr+1,rdata0);
        *((volatile unsigned int *)(addr+2))=data;
        rdata0=*((volatile unsigned int *)addr);
        printf("\nWrite to addr(0x%08x) :  Read without offset data(0x%08x)",addr+2,rdata0);
        *((volatile unsigned int *)(addr+3))=data;
        rdata0=*((volatile unsigned int *)addr);
        printf("\nWrite to addr(0x%08x) :  Read without offset data(0x%08x)",addr+3,rdata0);

        data=0x1234;
        *((volatile unsigned short *)(addr+0))=0x1234;
        printf("\naddr(0x%08x)=data(0x%08x)",addr,data);
        printf("\nReading with offset #0,#1");
        rdata0=*((volatile unsigned short *)(addr+0));
        printf("\naddr(0x%04x)=data(0x%04x)",addr+0,rdata0);
        rdata1=*((volatile unsigned short *)(addr+1));
        printf("\naddr(0x%04x)=data(0x%04x)",addr+1,rdata1);

        printf("\nWrite with offset #1");
        *((volatile unsigned short *)(addr+1))=data;
        rdata0=*((volatile unsigned short *)addr);
        printf("\nWrite to addr(0x%04x) :  Read without offset data(0x%04x)",addr+1,rdata0);


}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
void InitRom(void)
{
        unsigned int romcfg;
        unsigned int BASADDR,size,BTSIZE,tPA,tACC,PGMODE;

        BASADDR=0xC6D00000;
        size=0; // 0=256k,1=512k,2=1M,3=2M,4=4M,5=8M,6=16M,7=32M
        BTSIZE=1;//0=8bits,1=16bits,2=32bits
        tPA=0;//0=1MCLK,...7=8MCLK
        tACC=5;//0=1MCLK,...7=8MCLK
        PGMODE=0; // 0=normal ROM,1=4 word page,2=8 word page,3=16 word page

        romcfg=(BASADDR<<1)+(size<<16)+(BTSIZE<<2)+(tPA<<8)+(tACC<<3)+PGMODE;
        outpw(ROMCON,romcfg);
        printf("\n--- FLASH/ROM Initialization ---");
        printf("\nBase Address=0x%08x",BASADDR);
        printf("\nSize=%u Kbytes",(0x100<<size));
        printf("\nBTSIZE=%u bits",(8<<BTSIZE));
        printf("\ntPA=%u MCLK  tACC=%u MCLK",tPA+1,tACC+1);
        printf("\n--- FLASH/ROM Initialization Finished ---");

        beginAddr=(void *)BASADDR;
        endAddr=(void *)(BASADDR+(0x40000<<size));
        flash_id=0x002F;
        printf("\nbeginAddr=0x%08x endAddr=0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);

}


void SdramTypeDetect32(void)
{
        unsigned int BASADDR,MRSET,AUTOPR,LATENCY,LENGTH,COMPBK,DBWD,COLUMN,size;

        unsigned int rdata,data;
        volatile unsigned int *addr;
        volatile unsigned int *zeroaddr;

        BASADDR=0xC2000000;
        MRSET=1;
        AUTOPR=0;
         /* 0=1MCLK,1=2MCLK,2=3MCLK */
        LATENCY=1;
        /* 0=1 burst length,1=2,2=4,3=8 */
        LENGTH=0;
        /* 0=2 banks, 1=4 banks */
        COMPBK=1;
        /* 1=8bits,2=16bits,3=32bits */
        DBWD=3;
        /* 0=8bits,1=9bits,2=10bits */
        COLUMN=2;
        /* 1=2M,2=8M,3=16M,4=32M */
        size=4;

        zeroaddr=(unsigned int *)0x0;
        data=0x12345678;
//-------------------------------------------------------
        *((volatile unsigned int *)SDCONF0)=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(LENGTH<<8)+(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
        *zeroaddr=0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        addr=(unsigned int *)(1<<26);
        *addr=data;
        rdata=*zeroaddr;
        if( rdata==0 )
        {
                printf("\nType :  32Mx8");
                return;
        }

//-------------------------------------------------------
        COLUMN=1;
        *((volatile unsigned int *)SDCONF0)=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(LENGTH<<8)+(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
        *zeroaddr=0;
        addr=(unsigned int *)(1<<25);
        *addr=data;
        rdata=*zeroaddr;
        if( rdata==0 )
        {
                *zeroaddr=0;
                addr=(unsigned int *)(1<<24);
                *addr=data;
                rdata=*zeroaddr;
                if( rdata==0 )
                {
                        printf("\nType :  16Mx16");
                        return;
                }
                else
                {
                        printf("\nType :  16Mx8");
                        return;
                }
        }
//-------------------------------------------------------
        COLUMN=1;
        *((volatile unsigned int *)SDCONF0)=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(LENGTH<<8)+(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
        *zeroaddr=0;
        addr=(unsigned int *)(1<<24);
        *addr=data;
        rdata=*zeroaddr;
        if( rdata==0 )
        {
                printf("\nType :  8Mx8 or 8Mx16");
                return;
        }
//-------------------------------------------------------
        COLUMN=0;
        size=3;
        *((volatile unsigned int *)SDCONF0)=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(LENGTH<<8)+(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
        *zeroaddr=0;
        addr=(unsigned int *)(1<<23);
        *addr=data;
        rdata=*zeroaddr;
        if( rdata==0 )
        {
                printf("\nType :  4Mx16 or 4Mx32");
                return;
        }
//-------------------------------------------------------
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        COLUMN=1;
        COMPBK=0;
        size=2;
        *((volatile unsigned int *)SDCONF0)=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(LENGTH<<8)+(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
        *zeroaddr=0;
        addr=(unsigned int *)(1<<22);
        *addr=data;
        rdata=*zeroaddr;
        if( rdata==0 )
        {
                printf("\nType :  2Mx8");
                return;
        }
//-------------------------------------------------------
        COLUMN=0;
        COMPBK=0;
        size=1;
        *((volatile unsigned int *)SDCONF0)=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(LENGTH<<8)+(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
        *zeroaddr=0;
        addr=(unsigned int *)(1<<21);
        *addr=data;
        rdata=*zeroaddr;
        if( rdata==0 )
        {
                printf("\nType :  1Mx16");
                return;
        }
//-------------------------------------------------------

}

void SelfPrechargeTest(void)
{
        unsigned int data,rdata;
        unsigned int *addr;
        unsigned int delay;
        int cnt;

        cnt=0;
        data=0x12345678;
        for(addr=(unsigned int *)beginAddr;addr<(unsigned int *)beginAddr+0x100;addr++)
        {
                WriteMem((void *)addr,data,4);
        }

        *((volatile unsigned int *)EBICON)&=(~0x00070000);
        *((volatile unsigned int *)EBICON)|=0x00020000;
        delay=0x1000000;
        while(delay--);
        *((volatile unsigned int *)EBICON)&=(~0x00070000);
        *((volatile unsigned int *)EBICON)|=0x00050000;
        for(addr=(unsigned int *)beginAddr;addr<(unsigned int *)beginAddr+0x100;addr++)
        {
                rdata=ReadMem((void *)addr,4);
                if(rdata!=data)
                {
                        printf("\nError :  address=0x%08x write=0x%08x read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        }

        if(cnt==0)
                printf("\n**** Precharge Test PASSED!! ****");
        else
                printf("\nThere were %u failures occurred.",cnt);

}


void BaseAddrTest(void)
{
        unsigned int BASADDR;
        unsigned int data,rdata;
        unsigned int config;

        printf("\nNote :  This procedure only for SDRAM bank0.");
        data=0x12345678;
        *((unsigned int *)beginAddr)=data;
        BASADDR=0xC6000000;
        config=*((volatile unsigned int *)SDCONF0);
        *((volatile unsigned int *)SDCONF0)&=0x0007FFFF;
        *((volatile unsigned int *)SDCONF0)|=(BASADDR<<1);
        rdata=*((volatile unsigned int *)BASADDR);
        if(data!=rdata)
                printf("\nError :  Base address test fail.");
        else
                printf("\n**** Base address test PASSED ****");
        printf("\nbase addr=0x%08x write=0x%08x",(unsigned int)beginAddr,data);
        printf("\nnew base =0x%08x read =0x%08x",BASADDR,rdata);
        *((volatile unsigned int *)SDCONF0)=config;
}

void ChangeBurstLength(void)
{
        unsigned int length;
        unsigned int LENGTH;
        printf("\nPlease enter burst length(1,2,4,8) : ");
        scanf("%d",&length);

        switch(length)
        {
                case 1 :
                        LENGTH=0;
                        break;
                case 2 :
                        LENGTH=1;
                        break;
                case 4 :
                        LENGTH=2;
                        break;
                case 8 :
                        LENGTH=3;
                        break;
                default :
                        printf("\nWrong length!!");
                        return;
        }

        *((volatile unsigned int *)SDCONF0)&=0xFFFFF8FF;
        *((volatile unsigned int *)SDCONF0)|=(LENGTH<<8);
        *((volatile unsigned int *)SDCONF1)&=0xFFFFF8FF;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        *((volatile unsigned int *)SDCONF1)|=(LENGTH<<8);
}


void RandomRWTest(void)
{
        volatile unsigned int *addr;
        unsigned int addrcnt;
        unsigned int rdata,data;
        unsigned int range;
        unsigned int seed;
        int r,w,cnt;
        int randomaccesstimes;

        printf("\nPlease enter random seed : ");
        scanf("%u",&seed);
        addrcnt=seed;
        w=4;
        r=4;

        range=(((unsigned int)endAddr)-((unsigned int)beginAddr));
        randomaccesstimes=500000;
        addrcnt=0;
        cnt=0;
        while(randomaccesstimes--)
        {
                addr=(unsigned int *)((addrcnt%range)+(unsigned int)beginAddr);
                //printf("\nAddress = 0x%08x",(unsigned int)addr);
                data=addrcnt;
                WriteMem((void *)addr,data,w);
                rdata=ReadMem((void *)addr,r);
                if(data!=rdata)
                {
                        printf("\nTest number(count down)=%d",randomaccesstimes);
                        printf("\nError :  address=0x%08x write=0x%08x read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }

                data=addrcnt;
                WriteMem((void *)addr,data,w);
                rdata=ReadMem((void *)addr,r);
                if(data!=rdata)
                {
                        printf("\nTest number(count down)=%d",randomaccesstimes);
                        printf("\nError :  address=0x%08x write=0x%08x read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }

                addrcnt+=0x0a0d0304;
        }

        if(cnt==0)
                printf("\n**** Random Access Test PASSED!! ****");
        else
                printf("\nThere were %u failures occurred.",cnt);

}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
void SpecialSramTest2(void)
{
        volatile unsigned int *addr, *p;
        unsigned int data,rdata,tmp;
        unsigned cnt=0;
        unsigned int init;
        int w,r;

        init=0x0;
        w=4;
        r=4;

        printf("\nBegin Address=0x%08x  End Address=0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
                WriteMem((volatile void *)addr,init,w);

        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  address=0x%08x",(unsigned int)addr);

                data=(unsigned int)addr;
                WriteMem((volatile void *)addr,data,w);
                rdata=ReadMem((volatile void *)addr,r);
                if( rdata!=data )
                {
                                unit.word=data;
                                *((volatile unsigned short *)addr)=unit.h[0];
                                *((volatile unsigned short *)addr)=unit.h[0];
                                *((volatile unsigned short *)addr+1)=unit.h[1];
                                *((volatile unsigned short *)addr+1)=unit.h[1];
                }
                rdata=ReadMem((volatile void *)addr,r);
                WriteMem((volatile void *)addr,init,w);

                if( rdata!=data )
                {
                        printf("\nError :  Address=0x%08x  Write=0x%08x  Read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                        for(p=(volatile unsigned int *)beginAddr;p<(volatile unsigned int *)endAddr;p++)
                        {
                                tmp=ReadMem((volatile void *)p,r);
                                if( tmp!=init )
                                {
                                        printf("\n     *Error :  Address=0x%08x  Init=0x%08x
Read=0x%08x",(unsigned int)p,init,tmp);
                                        cnt++;
                                }
                        }
                }
        }

        if(cnt==0)
                printf("\n****** Special test 2 PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);



}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
void SpecialSramTest1(void)
{

        volatile unsigned int *addr, *p;
        unsigned int data,rdata,tmp;
        unsigned cnt=0;
        unsigned int init;
        int w,r;

        init=0xc600087c;
        w=2;
        r=2;

        printf("\nBegin Address = 0x%08x  End Address = 0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);

        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                WriteMem((volatile void *)addr,init,w);
                rdata=ReadMem((volatile void *)addr,r);
                if( rdata!=init)
                {
                        printf("\nError :  Address=0x%08x  Init=0x%08x  Read=0x%08x",(unsigned
int)addr,init,rdata);
                }

        }

        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  address=0x%08x",(unsigned int)addr);

                data=(unsigned int)addr;
                WriteMem((volatile void *)addr,data,w);
                rdata=ReadMem((volatile void *)addr,r);
                if( rdata!=data )
                {
                        printf("\nError :  Address=0x%08x  Write=0x%08x  Read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                        for(p=(volatile unsigned int *)beginAddr;p<(volatile unsigned int *)endAddr;p++)
                        {
                                tmp=ReadMem((volatile void *)p,r);
                                if( tmp!=init )
                                {
                                        printf("\n     *Error :  Address=0x%08x  Init=0x%08x
Read=0x%08x",(unsigned int)p,init,tmp);
                                        cnt++;
                                }
                        }
                }
                WriteMem((volatile void *)addr,init,w);
        }

        if(cnt==0)
                printf("\n****** Special test 1 PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
}


void SetSystemClk(void)
{
        unsigned int SystemClkCfg,sysclk;

        printf("\nSDRAM bank : ");
        scanf("%d",&sysclk);

        SystemClkCfg=(sysclk-2)*4;
        outpw(0x1100001C,0x0000A05F);// unlock SC_LOCK
        outpw(0x11000004,SystemClkCfg);
        outpw(0x1100001C,0x00000000);// lock SC_LOCK
        printf("\nSystem Clock=%dMHz",sysclk);

}

void InitSdram(void)
{
        unsigned int config,tcfg;
        unsigned int REFEN,REFMOD,CLKEN,REFRAT,WAITVT,LITTLE;
        unsigned int BASADDR,MRSET,AUTOPR,LATENCY,COMPBK,DBWD,COLUMN,size;
        unsigned int tRCD,tRDL,tRP,tRAS;
        unsigned int memory_size;
        int bank;
        int module;
        unsigned int sdram_type[4][9]={

        /*BASADDR,MRSET,AUTOPR,LATENCY,LENGTH,COMPBK,DBWD,COLUMN,SIZE*/

        {0xc2000000,1,0,1,0,0,2,0,1},/* 2M,16bits,2bank,8 */

        {0xc2000000,1,0,1,0,1,1,2,5},/* 32M,8bits,4bank,10 */

        {0xc2000000,1,0,1,0,1,3,0,3},/* 8M,32bits,4bank,8 */

        {0xc2000000,1,0,1,0,1,3,1,5} /* 32M,32bits,4bank,9 */
                                                                        };


        printf("\nSDRAM bank : ");
        scanf("%d",&bank);

        config=inpw(EBICON);
        printf("EBICON=0x%08x",config);
        config=inpw(SDCONF0);
        printf("\nSDCONF0=0x%08x",config);
        config=inpw(SDCONF1);
        printf("\nSDCONF1=0x%08x",config);
        config=inpw(SDTIME0);
        printf("\nSDTIME0=0x%08x",config);
        config=inpw(SDTIME1);
        printf("\nSDTIME1=0x%08x",config);
        printf("\nSetting EBICON,SDRAM...");

        REFEN=1;
        REFMOD=0;
        CLKEN=1;
        REFRAT=0x138;
        WAITVT=0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
LITTLE=0;
config=(REFEN<<18)+(REFMOD<<17)+(CLKEN<<16)+(REFRAT<<3)+(WAITVT<<1)+LITTLE;
printf("\nEBICON=0x%08x",config);
printf("\nREFEN=%d REFMOD=%d CLKEN=%d REFRAT=%d WAITVT=%d LITTLE=%d" \
               ,REFEN,REFMOD,CLKEN,REFRAT,WAITVT,LITTLE);
outpw(EBICON,config);


BASADDR=0xC2000000;
MRSET=1;
AUTOPR=0;
LATENCY=1;
/* 0=2 banks, 1=4 banks */
COMPBK=1;
/* 1=8bits,2=16bits,3=32bits */
DBWD=3;
/* 0=8bits,1=9bits,2=10bits */
COLUMN=1;
/* 1=2M,2=8M,3=16M,4=32M */
size=4;
#if 0
printf("\nSIZE(1=2M,2=4M,3=8M,4=16M,5=32M,6=64M) : ");
scanf("%u",&size);
printf("\nCOMPBK(0=2banks,1=4banks) : ");
scanf("%u",&COMPBK);
printf("DBWD(1=8bits,2=16bits,3=32bits) : ");
scanf("%u",&DBWD);
printf("COLUMN(0=8bits,1=9bits,2=10bits) : ");
scanf("%u",&COLUMN);
#else
printf("\nPlease select a module(0=#1,1=#2-1,2=#2-2,3=#3) : ");
scanf("%d",&module);
BASADDR=sdram_type[module][0];
MRSET=sdram_type[module][1];
AUTOPR=sdram_type[module][2];
LATENCY=sdram_type[module][3];
COMPBK=sdram_type[module][5];
DBWD=sdram_type[module][6];
COLUMN=sdram_type[module][7];
size=sdram_type[module][8];
#endif
switch(size)
{
        case 1 :
                memory_size=0x200000;
                break;
        case 2 :
                memory_size=0x400000;
                break;
        case 3 :
                memory_size=0x800000;
                break;
        case 4 :
                memory_size=0x1000000;
                break;
        case 5 :
                memory_size=0x2000000;
                break;
        default :
                memory_size=0;
                break;
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        /* 0=1,1=2....7=8 */
        tRCD=1;
        /* 0=1,1=2....3=4 */
        tRDL=0;
        /* 0=1,1=2....7=8 */
        tRP=0;
        /* 0=1,...7=8 */
        tRAS=7;

        tcfg=(tRCD<<8)+(tRDL<<6)+(tRP<<3)+(tRAS);
        config=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;

        printf("\nSDCONF%d : ",bank);
        printf("\nRBASADDR=0x%08x MRSET=%d AUTOPR=%d \
        \nLATENCY=%d COMPBK=%d DBWD=%d COLUMN=%d SIZE=%d"\
                        ,BASADDR,MRSET,AUTOPR,LATENCY,COMPBK,DBWD,COLUMN,size);
        printf("\nSDTIME%d : ",bank);
        printf("\ntRCD=%d tRDL=%d tRP=%d tRAS=%d",tRCD,tRDL,tRP,tRAS);

        switch(bank)
        {
                case 0 :
                        outpw(SDCONF0,config);
                        outpw(SDTIME0,tcfg);
                        break;
                case 1 :
                        BASADDR=BASADDR+(memory_size);
                        config=((BASADDR)<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                                +(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
                        outpw(SDCONF1,config);
                        outpw(SDTIME1,tcfg);
                        break;
                case 2 :
                        outpw(SDCONF0,config);
                        outpw(SDTIME0,tcfg);
                        config=((BASADDR+(memory_size))<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                                +(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
                        outpw(SDCONF1,config);
                        outpw(SDTIME1,tcfg);
                        break;
        }

        beginAddr=(volatile void *)BASADDR;
        endAddr=(volatile void *)(BASADDR+(memory_size));
        if(bank==2)endAddr=(volatile void *)((unsigned int)endAddr+(memory_size));
        printf("\nSDRAM Address : \nBase=0x%08x End=0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);

}

unsigned int DataAbortFlag=0;

__irq void DataAbort(void)
{
        DataAbortFlag=1;
        *((volatile unsigned int *)0x10)=0xE59FF018;
}

#define MODULE_NUM 4
#define BASE_ADDR 0xC2000000
void AutoInit(void)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
        __irq void DataAbort(void);
        unsigned int i;
        unsigned int sdram0[MODULE_NUM]={0x840088ED,0x840088B5,0x840088E3,0x84008841};
        unsigned int sdram1[MODULE_NUM]={0x880088ED,0x880088B5,0x850088E3,0x84408841};
        unsigned int shift_bits[MODULE_NUM]={25,24,22,20};

        *((volatile unsigned int *)0x30)=(unsigned int)DataAbort;

        *((volatile unsigned int *)EBICON)=0x000509C0;
        *((volatile unsigned int *)SDTIME0)=0x0000107;
        *((volatile unsigned int *)SDTIME1)=0x0000107;
        for(i=0;i<MODULE_NUM;i++)
        {
                *((volatile unsigned int *)SDCONF0)=sdram0[i];
                *((volatile unsigned int *)BASE_ADDR)=0x0;
                *((volatile unsigned int *)((1<<shift_bits[i])+BASE_ADDR))=0xAA553311;
                if( DataAbortFlag!=1)
                {
                if( *((volatile unsigned int *)BASE_ADDR)==0x0 )
                {
                        if( DataAbortFlag!= 1)
                        {
                        if( *((volatile unsigned int *)((1<<shift_bits[i])+BASE_ADDR))==0xAA553311)
                        {
                                if( DataAbortFlag!=1)
                                {
                                switch(i)
                                {
                                        case 0 :
                                                printf("\n16Mx16 4banks col=9 base=0xC2000000 total size=64M");
                                                break;
                                        case 1 :
                                                printf("\n32Mx8 4banks col=10 base=0xC2000000 total size=64M");
                                                break;
                                        case 2 :
                                                printf("\n2Mx32 4banks col=8 base=0xC2000000 total size=8M");
                                                break;
                                        case 3 :
                                                printf("\n1Mx16 2banks col=8 base=0xC2000000 total size=2M");
                                                break;
                                }
                                *((volatile unsigned int *)SDCONF1)=sdram1[i];
                                break;
                                }
                                DataAbortFlag=0;
                        }
                        }
                        DataAbortFlag=0;
                }
                }
                DataAbortFlag=0;
        }
        if( i==(MODULE_NUM+1)  )
        {
                printf("\nUnknow SDRAM module type!!!!");
        }

        /* Init External IO */
        printf("\nExtern I/O 0 base=0xc6000000 size=128k");
        *((volatile unsigned int *)EXT0CON)=0x8C007FFE;
        printf("\nExtern I/O 1 base=0xc6040000 size=128k");
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        *((volatile unsigned int *)EXT1CON)=0x8C087FFE;
        printf("\nExtern I/O 2 base=0xc6080000 size=128k");
        *((volatile unsigned int *)EXT2CON)=0x8C107FFE;
        printf("\nExtern I/O 3 base=0xc60c0000 size=128k");
        *((volatile unsigned int *)EXT3CON)=0x8C187FFE;


}



void RWTypeTest(void)
{
        #define RW_TEST_SIZE (0x200)
        volatile unsigned int *addr;
        unsigned int data,rdata;
        unsigned cnt=0;
        void * endaddr;
        int w_type[]={4,4,4,2,2,2,1,1,1};
        int r_type[]={4,2,1,4,2,1,4,2,1};
        int i;
        int w,r;

        endaddr=(void *)((unsigned int)beginAddr+RW_TEST_SIZE);

        i=0;
        while(i<9)
        {
        w=w_type[i];
        r=r_type[i];
        printf("\nAccess type :  Write=%d Bytes  Read=%d Bytes",w,r);
        i++;
        printf("\nBegin Address=0x%08x  End Address=0x%08x",(unsigned int)beginAddr,(unsigned
int)endaddr+RW_TEST_SIZE);
        data=0;
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endaddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  write address=0x%08x",(unsigned int)addr);
                data+=0x01020304;
                WriteMem((volatile void *)addr,data,w);
        }
        data=0;
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endaddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  read address=0x%08x",(unsigned int)addr);
                rdata=ReadMem((volatile void *)addr,r);
                data+=0x01020304;
#ifndef _BIG_ENDIAN
                switch(r+w)
                {
                        case 6 :
                                rdata=(rdata>>16)+(rdata<<16);
                                break;
                        case 5 :
                                rdata=(rdata>>24)+((rdata>>8)&0x0000FF00)+((rdata<<8)&0x00FF0000)+(rdata<<24);
                                break;
                        case 3 :
                                rdata=((rdata>>8)&0x00FF00FF)+((rdata<<8)&0xFF00FF00);
                                break;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                        default :
                                break;
                }
#endif
                if( rdata!=data )
                {
                        printf("\nError :  Writing or Reading or Addressing error  : ");
                        printf("\nAddress=0x%08x  Write=0x%08x  Read=0x%08x",(unsigned int)addr,data,rdata);
                        cnt++;
                }
        }

        }

        if(cnt==0)
                printf("\n****** SDRAM test PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);
}



int MemTest(void)
{
        int func=-1;
        int fnum;
        int i;
        fnum=sizeof(TestFuncTable)/sizeof(testfunc_t);
        printf("\n--------------------");
        printf("\nMemory test program.");
        printf("\n--------------------");
        for(i=0;i<fnum;i++)
                printf("\n%d. %s",TestFuncTable[i].func,TestFuncTable[i].name);
        printf("\nPlease select a function : ");
        scanf("\n%d",&func);
        for(i=0;i<fnum;i++)
        {
                if( TestFuncTable[i].func == func )
                {
                        printf("\n%s",TestFuncTable[i].name);
                        TestFuncTable[i].prog();
                        return 1;
                }
        }

        printf("\nUnknown function. Program terminated.");
        return 0;
}

void SingleAccessLoop(void)
{
        volatile unsigned int *addr;
        unsigned int rdata,data;
        unsigned int ecnt,cnt;
        ecnt=0;
        cnt=0;

        addr=(volatile unsigned int *)0xc6001800;
        data=0xAAAAAAAA;
        while(cnt++<0x1000000)
        {
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                if( (cnt&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  cnt=0x%08x",cnt);

                data=0xc6000000+cnt;
                *addr=data;
                rdata=*addr;
                if( rdata!=data )
                {
                        printf("\nError :  Address=0x%08x  Write=0x%08x  Read=0x%08x",(unsigned
int)addr,data,rdata);
                        ecnt++;
                }
        }

        if(cnt==0)
                printf("\n****** Single access loop PASSED ******");
        else
                printf("\nThere were %u failures occurred in %u loops.",ecnt,cnt);

}


void SerialAccess(void)
{
        volatile unsigned int *addr;
        unsigned int data,rdata;
        unsigned cnt=0;
        int w,r;

        w=4;
        r=4;

        printf("\nBegin Address=0x%08x  End Address=0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  address=0x%08x",(unsigned int)addr);
                data=(unsigned int)addr;
                WriteMem((void *)addr,data,w);
                rdata=ReadMem((void *)addr,r);
                if( rdata!=data )
                {
                        printf("\nError :  Address=0x%08x  Write=0x%08x  Read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }
        }

        if(cnt==0)
                printf("\n****** Serial access test PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);

}


void AccessTypeTest(void)
{
        volatile unsigned int *addr;
        unsigned int data,rdata;
        unsigned cnt=0;
        int w,r;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        printf("\nWrite byte number : ");
        scanf("%d",&w);
        printf("Read byte number : ");
        scanf("%d",&r);


        printf("\nBegin Address=0x%08x  End Address=0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);

        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  write address=0x%08x",(unsigned int)addr);
                data=(unsigned int)0xFFFF-((unsigned int)addr)&0xFFFF;
                data+=((unsigned int)0xFFFF-((unsigned int)addr+2)&0xFFFF)<<16;
                WriteMem((volatile void *)addr,data,w);
        }
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  read address=0x%08x",(unsigned int)addr);
                rdata=ReadMem((volatile void *)addr,r);
                data=0xFFFF-(unsigned int)addr&0xFFFF;
                data+=(0xFFFF-((unsigned int)addr+2)&0xFFFF)<<16;
                if( rdata!=data )
                {
                        printf("\nError :  Writing or Reading or Addressing error  : ");
                        printf("\nAddress=0x%08x  Write=0x%08x  Read=0x%08x",(unsigned int)addr,data,rdata);
                        cnt++;
                }
        }

        if(cnt==0)
                printf("\n****** Access type test PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);


}


void EntireRamTest(void)
{
        volatile unsigned int *addr;
        unsigned int data,rdata,init;
        unsigned cnt=0;
        int w,r;

        w=4;
        r=4;
        printf("\nBegin Address=0x%08x  End Address=0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);

        init=0x01020304;
        data=0;
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  write address=0x%08x",(unsigned int)addr);
                #if 0
                data=(unsigned int)0xFFFF-((unsigned int)addr)&0xFFFF;
                data+=((unsigned int)0xFFFF-((unsigned int)addr+2)&0xFFFF)<<16;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                WriteMem((volatile void *)addr,data,w);
                #else
                data+=init;
                *((volatile unsigned int *)addr)=data;
                #endif
        }
        data=0;
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  read address=0x%08x",(unsigned int)addr);
                #if 0
                rdata=ReadMem((volatile void *)addr,r);
                data=0xFFFF-(unsigned int)addr&0xFFFF;
                data+=(0xFFFF-((unsigned int)addr+2)&0xFFFF)<<16;
                #else
                data+=init;
                rdata=*((volatile unsigned int *)addr);
                #endif

                if( rdata!=data )
                {
                        printf("\nError :  Address=0x%08x  Write=0x%08x  Read=0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }
        }

        if(cnt==0)
                printf("\n****** Addressing test PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);

}

void MemBitsTest(void)
{
        volatile unsigned int *addr;
        unsigned int data,rdata;
        unsigned cnt=0;
        int w,r;

        w=4;
        r=4;

        printf("\nBegin Address = 0x%08x  End Address = 0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);


        data=0x55555555;
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  write address=0x%08x",(unsigned int)addr);
                WriteMem((volatile void *)addr,data,w);
        }
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  read address=0x%08x",(unsigned int)addr);
                rdata=ReadMem((volatile void *)addr,w);
                if(rdata!=data)
                {
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                        printf("\nError :  Writing or Reading error  : ");
                        printf("\nAddress = 0x%08x  Write = 0x%08x  Read = 0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }
        }
        data=0xAAAAAAAA;
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  write address=0x%08x",(unsigned int)addr);
                WriteMem((volatile void *)addr,data,r);
        }
        for(addr=(volatile unsigned int *)beginAddr;addr<(volatile unsigned int *)endAddr;addr++)
        {
                if( ((unsigned int)addr&(0xFFFFFFFF>>bit))==0x0 )
                        printf("\nProcessing :  read address=0x%08x",(unsigned int)addr);
                rdata=ReadMem((volatile void *)addr,r);
                if(rdata!=data)
                {
                        printf("\nError :  Writing or Reading error  : ");
                        printf("\nAddress = 0x%08x  Write = 0x%08x  Read = 0x%08x",(unsigned
int)addr,data,rdata);
                        cnt++;
                }
        }

        if(cnt==0)
                printf("\n****** Memory bits test PASSED ******");
        else
                printf("\nThere were %u failures occurred.",cnt);

}

void WnRn(void)
{
        unsigned int n,i;
        unsigned int v[4]={0x12345678,0x55aa55aa,0x1234aa55,0xaa551234};
        unsigned int rv[4]={0};
        int w,r;
        unsigned int addr;

        printf("\nWrite n units : ");
        scanf("%u",&n);
        if(n>4)
        {
                printf("\nError :  n > 4.");
                return;
        }


        printf("\nWrite byte number : ");
        scanf("%d",&w);
        printf("\nRead byte number : ");
        scanf("%d",&r);
        printf("\nRW starting address(hex) : ");
        scanf("%x",&addr);

        for(i=0;i<n;i++)
        {
                WriteMem((volatile void *)(addr+i*4),v[i],w);
        }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        for(i=0;i<n;i++)
        {
                rv[i]=ReadMem((volatile void *)(addr+i*4),r);
        }

        for(i=0;i<n;i++)
                printf("\nWrite :  Address=0x%08x  Data=0x%08x",(unsigned int)(addr+i*4),v[i]);
        for(i=0;i<n;i++)
                printf("\nRead  :  Address=0x%08x  Data=0x%08x",(unsigned int)(addr+i*4),rv[i]);
}

void SingleAccess(void)
{
        int w,r;
        unsigned int addr,data;

        printf("\nWrite byte number : ");
        scanf("%d",&w);
        printf("\nRead byte number : ");
        scanf("%d",&r);
        printf("\nRW Address(hex) : ");
        scanf("%x",&addr);
        printf("\nRW data(hex) : ");
        scanf("%x",&data);
        printf("\nThe value read from memory = 0x%08x",RWMem((volatile void *)addr,data,w,r));
}

void InitExtIo(void)
{
        unsigned int iobank,buswidth,memory_size;
        unsigned int config,BASADDR,size,tACC,tCOH,tACS,tCOS,DBWD;

        printf("\nExternal I/O bank : ");
        scanf("%u",&iobank);
        printf("Bus width : ");
        scanf("%u",&buswidth);
        if( (iobank&0xFFFFFFFC) )
        {printf("\nExternal I/O bank error! Should be 0/1/2/3.");return;}
        if( buswidth!=8 && buswidth!=16 && buswidth!=32 )
        {printf("\nBus Width error! Should be 8/16/32.");return;}

        BASADDR=0xC6000000+0x40000*iobank;
        size=0;/* 256k */
        tACC=7;/* 7MLK */
        tCOH=7;/* 7MLK */
        tACS=7;/* 7MLK */
        tCOS=7;/* 7MLK */
        DBWD=(buswidth==8)?1 : ((buswidth==16)?2 : 3);
        config=(BASADDR<<1)+(size<<16)+(tACC<<11)+(tCOH<<8)+(tACS<<5)+(tCOS<<2)+DBWD;
        outpw(EXT0CON+4*iobank,config);
        printf("\nEXT%dCON=0x%08x",iobank,config);
        switch(iobank)
        {
                case 0 :
                case 1 :
                case 2 :
                        memory_size=0x20000;
                        flash_id=0xFFFF;
                        break;
                case 3 :
                        flash_id=0x002F;
                        printf("\nflash_id=0x%04x",flash_id);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                        memory_size=0x20000;
                        break;
                default :
                        flash_id=0xFFFF;
                        memory_size=0;
                        break;
        }
        beginAddr=(volatile void *)BASADDR;
        endAddr=(volatile void *)(BASADDR+memory_size);
        printf("\nbeginAddr=0x%08x endAddr=0x%08x",(unsigned int)beginAddr,(unsigned int)endAddr);

}

unsigned int ReadMem(volatile void *addr,int r)
{
#ifdef _BIG_ENDIAN
#error "ReadMem is just for little endian."
#endif
        unsigned int rdata;
        switch(r)
        {
                case 1 :
                        unit.b[0]=*((volatile unsigned char *)addr);
                        unit.b[1]=*((volatile unsigned char *)addr+1);
                        unit.b[2]=*((volatile unsigned char *)addr+2);
                        unit.b[3]=*((volatile unsigned char *)addr+3);
                        rdata=unit.word;
                        break;
                case 2 :
                        unit.h[0]=*((volatile unsigned short *)addr);
                        unit.h[1]=*((volatile unsigned short *)addr+1);
                        rdata=unit.word;
                        break;
                case 4 :
                        rdata=*((volatile unsigned int *)addr);
                        break;
                default :
                        return 0;
        }

        return rdata;
}

unsigned int WriteMem(volatile void *addr,unsigned int data,int w)
{
#ifdef _BIG_ENDIAN
#error "WriteMem is just for little endian."
#endif
        unit.word=data;
        switch(w)
        {
                case 1 :
                        *((volatile unsigned char *)addr)=unit.b[0];
                        *((volatile unsigned char *)addr+1)=unit.b[1];
                        *((volatile unsigned char *)addr+2)=unit.b[2];
                        *((volatile unsigned char *)addr+3)=unit.b[3];
                        break;
                case 2 :
                        *((volatile unsigned short *)addr)=unit.h[0];
                        *((volatile unsigned short *)addr+1)=unit.h[1];
                        break;
                case 4 :
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                        *((volatile unsigned int*)addr)=data;
                        break;
                default :
                        return 0;
        }
        return 0;
}



unsigned int RWMem(volatile void *addr,unsigned int data,int w,int r)
{
        WriteMem(addr,data,w);
        return(ReadMem(addr,r));
}



void vpascal_triangle (int depth,int tube)
{
        int     i,j;
        unsigned int *p;
        unsigned int ecnt=0;

p=(unsigned int *)tube;

        for (j = 1; j < depth; j++) {

                for (i = 1; i <= j; i++) {
                        if( *p!=vpascal_generator(j,i) )
                {
                                        ecnt++;
                        }
                        p++;
                }
        }

        if( ecnt==0 )
                printf("\nPASCAL TRIANGLE execution OK!");
        else
                printf("\nPASCAL TRIANGLE execution FAIL!");


}

int vpascal_generator (int n, int i)
{

        if (i == 1 || i == n) return 1;

        return vpascal_generator(n-1,i-1) + vpascal_generator(n-1,i);
}



/*
//#include <rt_misc.h>
__value_in_regs struct __initial_stackheap __user_initial_stackheap(
        unsigned R0, unsigned SP, unsigned R2, unsigned SL)
{
    struct __initial_stackheap config;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    config.heap_base = 0xC2000000;
    config.stack_base = 0xC6000000;

    return config;
}

*/
```

2.5.2.5 TEST740.C

```
#define MAIN_PROG
#include <stdio.h>

extern int MemTest(void);
int main(void)
{


        *((volatile unsigned int *)0x1100001c)=0xa05f;
        *((volatile unsigned int *)0x11000004)=0x20;
        *((volatile unsigned int *)0x1100001c)=0x0;
        while(MemTest());


}
```

2.5.2.6 TRIANGLE.S

```
; generated by ARM C Compiler, ADS1.1 [Build 723]
        CODE32

EXE_BASE        EQU     0xc2000000

        EXPORT pascal_triangle
        EXPORT far_exe

        IMPORT _printf
        IMPORT putc
        IMPORT __stdout

        AREA PASCAL_TRIANGLE, CODE, READONLY

far_exe ; r0=base r1=r/w base r2=depth
        STMFD   sp!,{r4-r9,lr}
        LDR     r4,=CopyStartPoint
        LDR     r5,=CopyStopPoint
        MOV     r6,r0
0
        LDR     r8,[r4],#4
        STR     r8,[r6],#4
        CMP     r4,r5
        BNE     %0

        LDR     r4,=code16_start_point
        BIC     r4,r4,#1
        LDR     r5,=CopyStartPoint
        SUB     r9,r4,r5
        LDR     r5,=code16_stop_point
        BIC     r5,r5,#1
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        ADD     r6,r0,r9
1
        LDRH    r8,[r4],#2
        STRH    r8,[r6],#2
        CMP     r4,r5
        BNE     %1



        MOV     r4,r0
        MOV     r0,r2
        MOV     lr,pc
        MOV     pc,r4
        LDMFD   sp!,{r4-r9,pc}


CopyStartPoint
pascal_triangle PROC
        STMFD    sp!,{r4-r9,lr}
        MOV      r7,r0
        MOV      r8,r1
        MOV      r6,r8
        MOV      r4,#1
|L1.96|
        CMP      r4,r7
        BGE      |L1.164|
        B        |L1.116|
|L1.108|
        ADD      r4,r4,#1
        B        |L1.96|
|L1.116|
        MOV      r5,#1
|L1.120|
        CMP      r5,r4
        BGT      |L1.108|
        B        |L1.140|
|L1.132|
        ADD      r5,r5,#1
        B        |L1.120|
|L1.140|
        MOV      r1,r5
        MOV      r0,r4
        ADR              r9,pascal_generator+1
        MOV              lr,pc
        BX               r9
        ;BL       pascal_generator
        STR      r0,[r6,#0]
        ADD      r6,r6,#4
        B        |L1.132|
|L1.164|
        LDMFD    sp!,{r4-r9,lr}
        BX       lr
        ENDP

               CODE16
code16_start_point
pascal_generator PROC
        PUSH     {r4-r6,lr}
        MOV      r5,r0

        MOV      r4,r1
        CMP      r4,#1
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        BEQ     |L1.14|
        CMP     r4,r5

        BNE     |L1.22|
|L1.14|
        MOV     r0,#1

|L1.16|
        POP     {r4-r6}
        POP     {r3}

        BX      r3
|L1.22|
        SUB     r1,r4,#1

        SUB     r0,r5,#1
        BL      pascal_generator

        MOV     r6,r0
        MOV     r1,r4

        SUB     r0,r5,#1
        BL      pascal_generator

        ADD     r0,r6,r0
        B       |L1.16|

        ENDP
              CODE32

CopyStopPoint
code16_stop_point
        END
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

3 Cache Controller

3.1 Overview

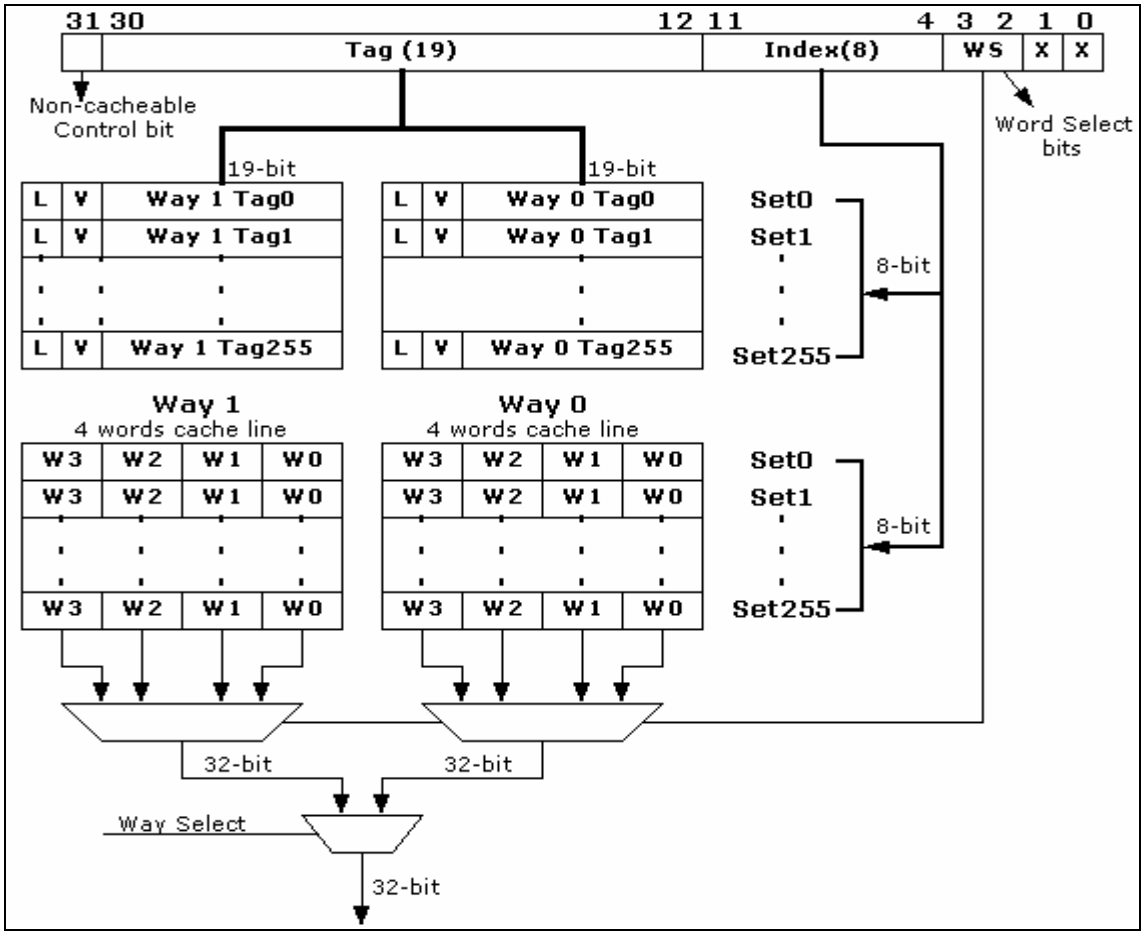
The NUC740 incorporates a 8KB Instruction cache, a 2KB Data cache, and 8 words write buffer to improve the system performance. The caches consist of high-speed SRAM that provides quicker access time than external memory. If cache is enabled, the CPU tries to fetch instructions from I-cache instead of external memory. Similarly, the CPU tries to read data from D-cache instead of external memory. But note that the CPU will write data into both D-cache and write buffer (**write-through** mode). If I-Cache / D-Cache were disabled, these cache memories can be treated as On-Chip SRAM.

For NUC740, the I-Cache and D-Cache have similar configuration except the cache size. To raise the cache-hit ratio, these two caches are configured as two-way set associative addressing. The I-cache organization is 256 sets, two lines per set. The D-cache organization is 64 sets, two lines per set. Each cache has four words cache line size. When a miss occurs, four words must be fetched consecutively from external memory. The replacement algorithm is a LRU (Least Recently Used).

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

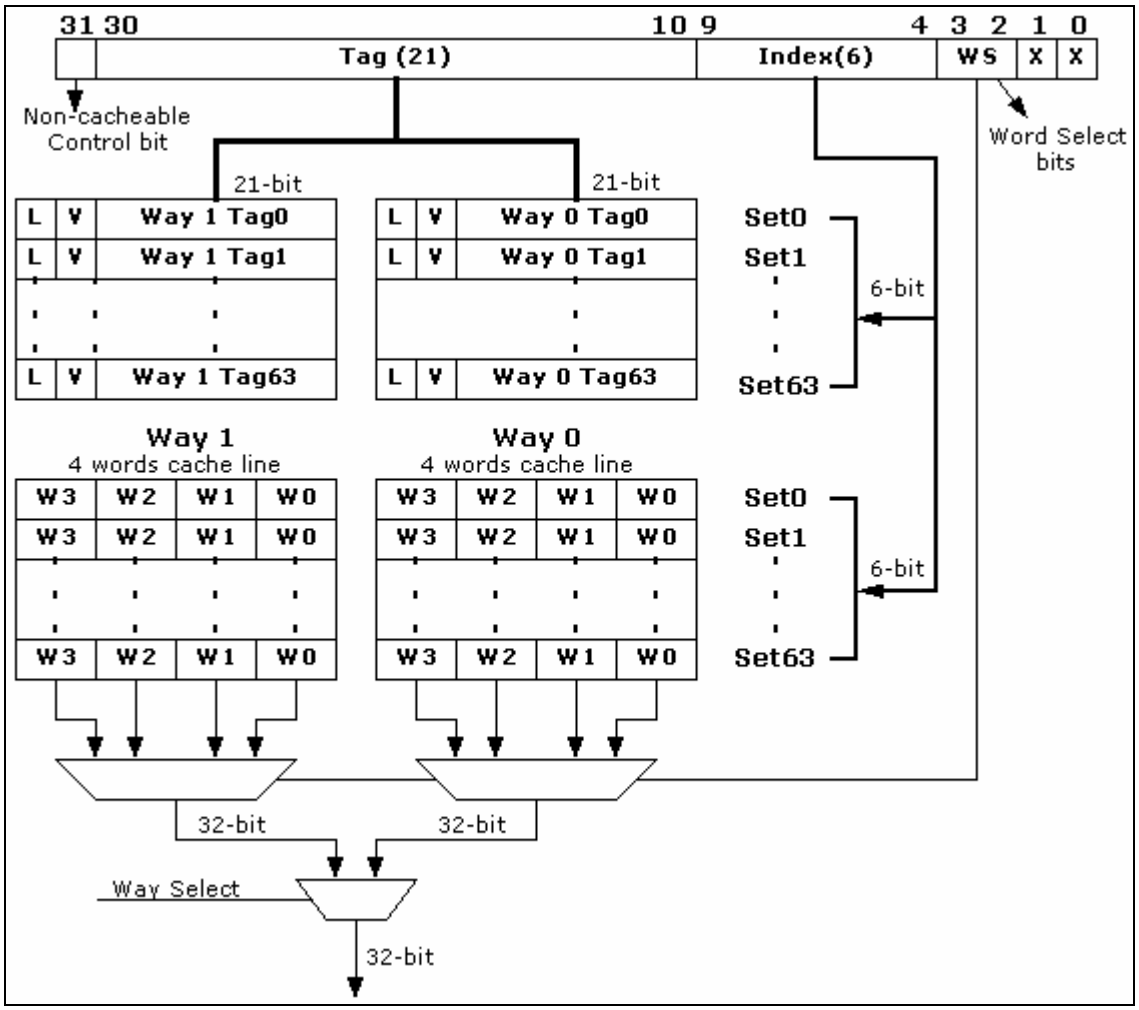
3.2 Block Diagram

Figure 3-1 Instruction Cache Organization Block Diagram



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 3-2 Data Cache Organization Block Diagram



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

3.3 Registers

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
CAHCNF	0xFFF0.2000	R/W	Cache configuration register	0x0000.0000
CAHCON	0xFFF0.2004	R/W	Cache control register	0x0000.0000
CAHADR	0xFFF0.2008	R/W	Cache address register	0x0000.0000
CTEST0	0xFFF6.0000	R/W	Cache test register 0	0x0000.0000
CTEST1	0xFFF6.0004	R	Cache test register 1	0x0000.0000

3.4 Functional Descriptions

3.4.1 On-Chip RAM

If I-Cache or D-Cache is disabled, it can be used as On-Chip SRAM. The size of On-Chip RAM depends on the I-Cache and D-Cache enable bits **ICAEN**, **DCAEN** in Cache Configuration Register **(CAHCNF)**. Table 3-1lists the details.

Table 3-1 The size and start address of On-Chip RAM

ICAEN	DCAEN	On-Chip RAM	
		Size	Start Address
0	0	**10KB**	**0x7FE0.0000**
0	1	**8KB**	**0x7FE0.0000**
1	0	**2KB**	**0x7FE0.2000**
1	1	**Unavailable**	

3.4.2 Non-Cacheable Area

The cache affects the first 2GB system memory. Sometimes it is necessary to define non-cacheable areas when the consistency of data stored in memory and the cache can't be ensured. To support this feature, the NUC740 provides a non-cacheable area control bit in the address field**,**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

A [31]. If A [31] in the ROM/FLASH, SDRAM, or external I/O bank's access address is "0", then the accessed data is cacheable. If the A [31] value is "1", the accessed data is non-cacheable.

Cache Control Register

The Cache controller supports one Control register **(CAHCON)** to control cache flushing, lock/unlock and drain write buffer. All the command set bits of CAHCON register are auto-clear bit. At the end of execution, the command set bit will be cleared to "0" automatically. Figure 3-3 lists the bit definition of CAHCON register. The detail description of each bit filed can be found in NUC740 specification.

Figure 3-3 Cache Control Register

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED							
7	6	5	4	3	2	1	0
DRWB	ULKS	ULKA	LDLK	FLHS	FLHA	DCAH	ICAH

3.4.3 Cache Flushing

To prevent unpredictable error, it's better to flush cache before enable it. Both I-Cache and D-Cache can be entirely flushed in one operation, or be flushed one line at a time. The bit **FLHA** and **FLHS** of register CAHCON are used to flush entire cache and single line, respectively. Bit **DCAH** or **ICAH** of register CAHCON is used to select D-Cache or I-Cache for the flush operation. The Cache Address Register **(CAHADR)** must be set before flush a single cache line.

Due to NUC740 does not support external memory snooping; it is necessary to flush cache if the force consistency of cache and memory is required. For example, The I-Cache should be flushed after a self-modifying code is executed. Similarly, the D-Cache should be flushed before an external device starts a DMA transfer with a cacheable memory region.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

3.4.4 Cache Enable and Disable

After the cache was flushed, the cache can be enabled, bit **ICAEN** and **DCAEN** of register **CAHCNF** is used to enable D-Cache and I-Cache. The D-Cache and I-Cache can be enabled individually, or enabled at the same time. The write buffer can be enabled by setting the **WRBEN.** Most of the time, **ICAEN**, **DCAEN** and **WRBEN** are enabled at the same time. Figure 3-4 list the register definition of Cache Configuration Register (**CAHCNF**).

Figure 3-4 Cache Configuration Register

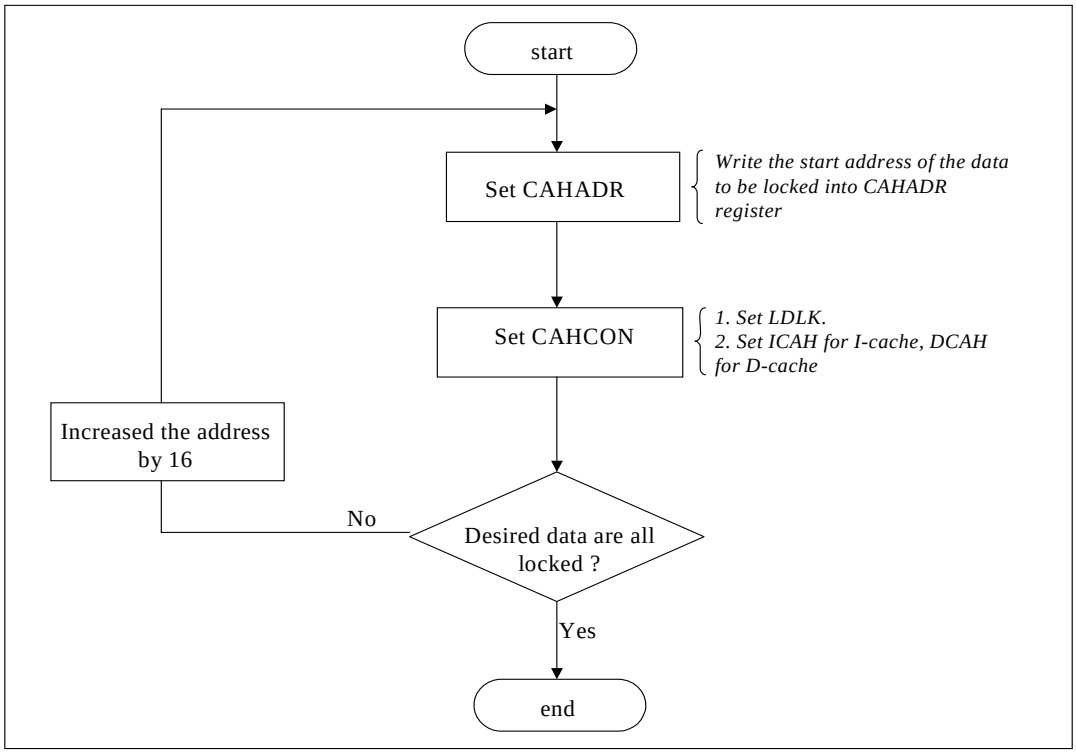
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED							
7	6	5	4	3	2	1	0
RESERVED					WRBEN	DCAEN	ICAEN

3.4.5 Cache Load and Lock

The NUC740 cache controller supports a cache-locking feature that locks critical sections of code or data into I-Cache or D-Cache. This guarantees the quick access to these critical sections. Lockdown operation can be performed with a granularity of one cache line (4 words). The smallest size, which can be locked down, is 4 words. After a line is locked, it operates as a regular instruction SRAM. Locked lines don't be replaced either cache misses or flush per line command. Figure 3-5 shows the steps for locking instructions or data.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 3-5 Cache Load and Lock



There are some limitations during the locking cache line into the I-Cache or D-Cache.

The code that executes load and lock operation should be held in a non-cacheable area of memory.

The cache should be enabled and interrupts should be disabled.

Flushed the cache before execute load and lock to ensure that the data to be locked down is not already in the cache.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

3.4.6 Cache Unlock

The unlock operation is used to unlock previously locked cache lines. The cache controller provides two unlock command, unlock line and unlock all.

The unlock line operation is performed on a cache line granularity. In case the line is found in the cache, it is unlocked and starts to operate as a regular valid cache line. In case the line is not found in the cache, no operation is done and the command terminates with no exception. To unlock one line, write the address of the line to be unlocked into the **CAHADR** Register, and then set the **ULKS** and **ICAH** bits in the CAHCON register for I-cache or set the **ULKS** and **DCAH** bits for D-cache.

The unlock all operation is performed on all cache lines of I-Cache or D-Cache. In case a line is locked, it is unlocked and starts to operate as regular valid cache line. In case a line is not locked or if it is invalid, no operation is performed. To unlock the whole instruction cache, set the **ULKA** and **ICAH** bits. To unlock the whole data cache, set the **ULKA** and **DCAH** bits.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

4 Timers

4.1 Overview

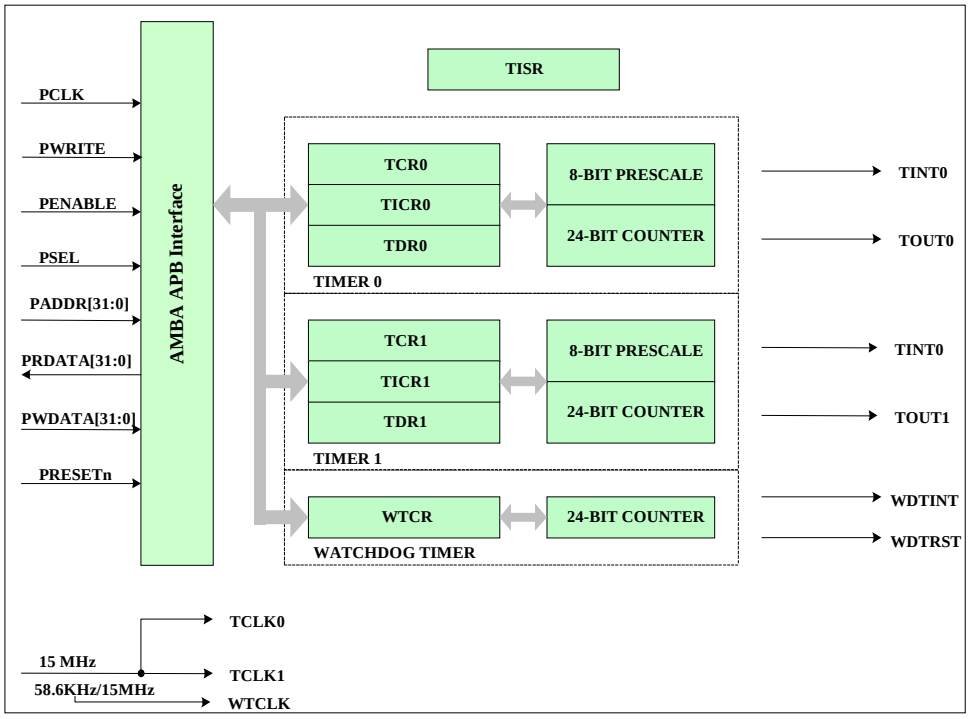
The NUC740 **Timer** module includes two channels, TIMER0 and TIMER1, which allow user to easily implement a counting scheme. Each channel has independent clock source. The input clock is divided by an 8-bit pre-scalar and then referenced by a 24-bit down counter. When the counter counts down to zero, the Timer will assert an interrupt request if the interrupt is enabled. A general software-counting scheme is to set a software counter, and add 1 to it upon every interrupt.

The Timer module also includes a watchdog timer. It supports the system restart if the system goes into problem. This prevents system from hanging for an infinite period of time. The watchdog timer is a free running counter with programmable time-out intervals. When the specified time internal interval expires, it asserts an interrupt to inform software to reset the counter. If the counter doesn't be reset during 512 WDT clocks, the watchdog timer will generate a system restart signals to reset the whole system. Normally, the program should implement a task to periodically reset the counter if the watchdog timer is enabled.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

4.2 Block Diagram

Figure 4-1 Timer Block Diagram



4.3 Registers

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

Register	Address	R/W/C	Description	Reset Value
TCR0	0xFFF8.1000	R/W	Timer Control Register 0	0x0000.0005
TCR1	0xFFF8.1004	R/W	Timer Control Register 1	0x0000.0005
TICR0	0xFFF8.1008	R/W	Timer Initial Control Register 0	0x0000.00FF
TICR1	0xFFF8.100C	R/W	Timer Initial Control Register 1	0x0000.00FF
TDR0	0xFFF8.1010	R	Timer Data Register 0	0x0000.00FF
TDR1	0xFFF8.1014	R	Timer Data Register 1	0x0000.00FF
TISR	0xFFF8.1018	R/C	Timer Interrupt Status Register	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

WTCR	0xFFF8.101C	R/W	Watchdog Timer Control Register	0x0000.0000

4.4 IV. Functional Descriptions

4.4.1 Interrupt Frequency

The frequency of timer interrupt depends on the following equation :

Freq. = Crystal clock / ((pre-scaler+1) * counter))

For NUC740, the crystal clock input is 15 MHZ. According to the equation, user can decide the values of pre-scalar and counter to get the desired interrupt frequency. Table 4-1demonstrates several reference values.

Table 4-1 Timer Reference Setting Values

Frequency (1/sec)	[Pre-Scalar]	[Counter]
18	0	0xCB735
40	0	0x5B8D8
100	0	0x249F0

4.4.2 Initialization

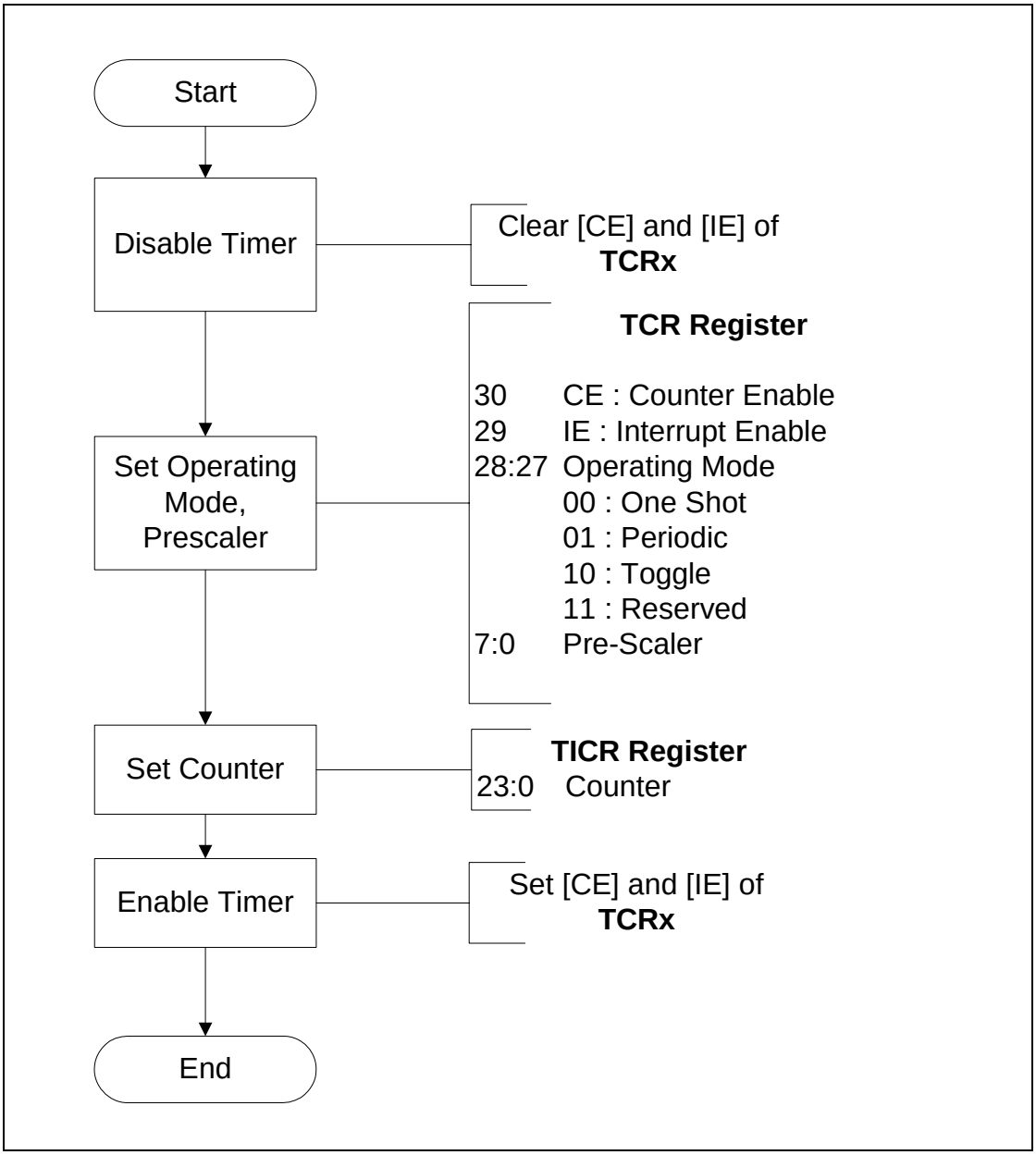
The driver should set the operating mode, pre-scalar and counter before enable the timer interrupt. The Timer supports *one-shot*, *periodic* and *toggle* mode for user to implement the counting scheme.

- In *one-shot* mode, the interrupt signal is generated once and it's not happen again unless the timer is re-enabled later.
- In *periodic* mode, the interrupt signal is generated periodically.
- In *toggle* mode, the interrupt signal is generated on each low-to-high or high-to-low transition with 50% duty cycle.

Figure 4-2 shows the initialization sequence.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 4-2 Timer Initialization Sequence

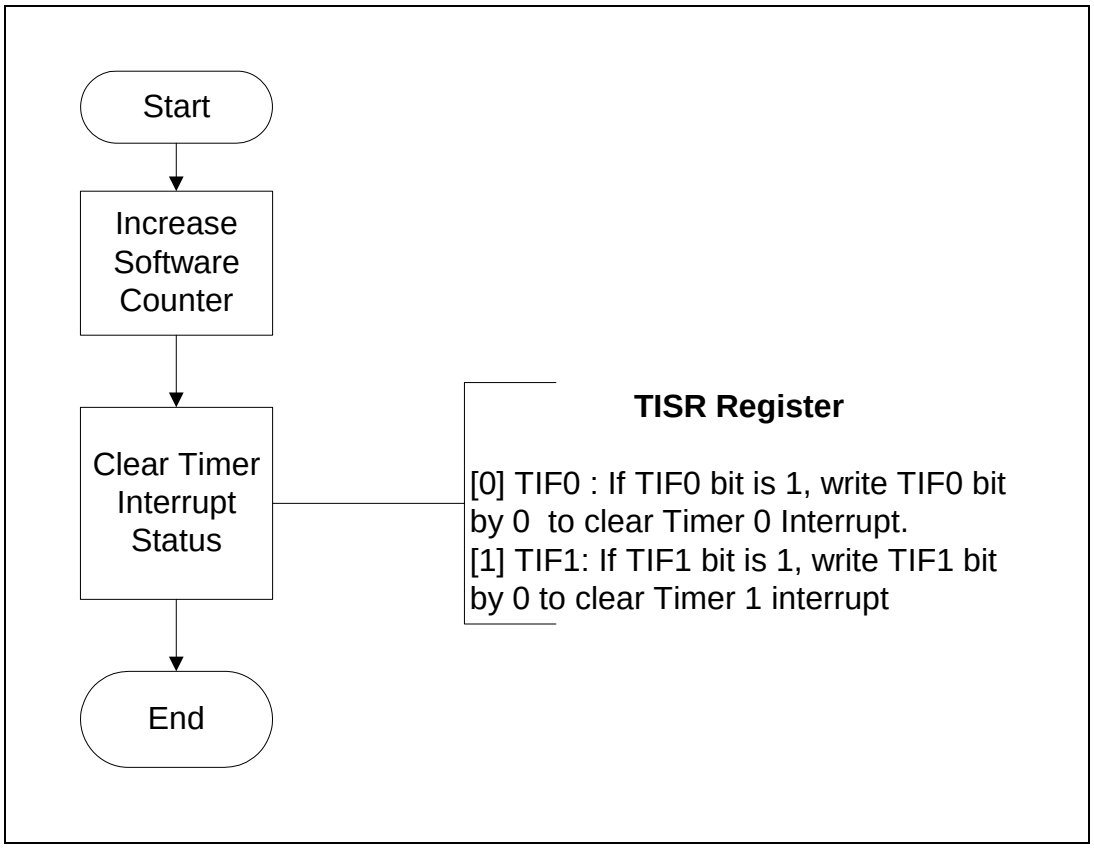


The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

4.4.3 Timer Interrupt Service Routine

A common timer interrupt service routine is very simple. It increases the software counter and clear the timer interrupt status. Figure 4-3 shows the flow chart of such an interrupt service routine.

Figure 4-3 Timer Interrupt Service Routine



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

4.4.4 Watchdog Timer

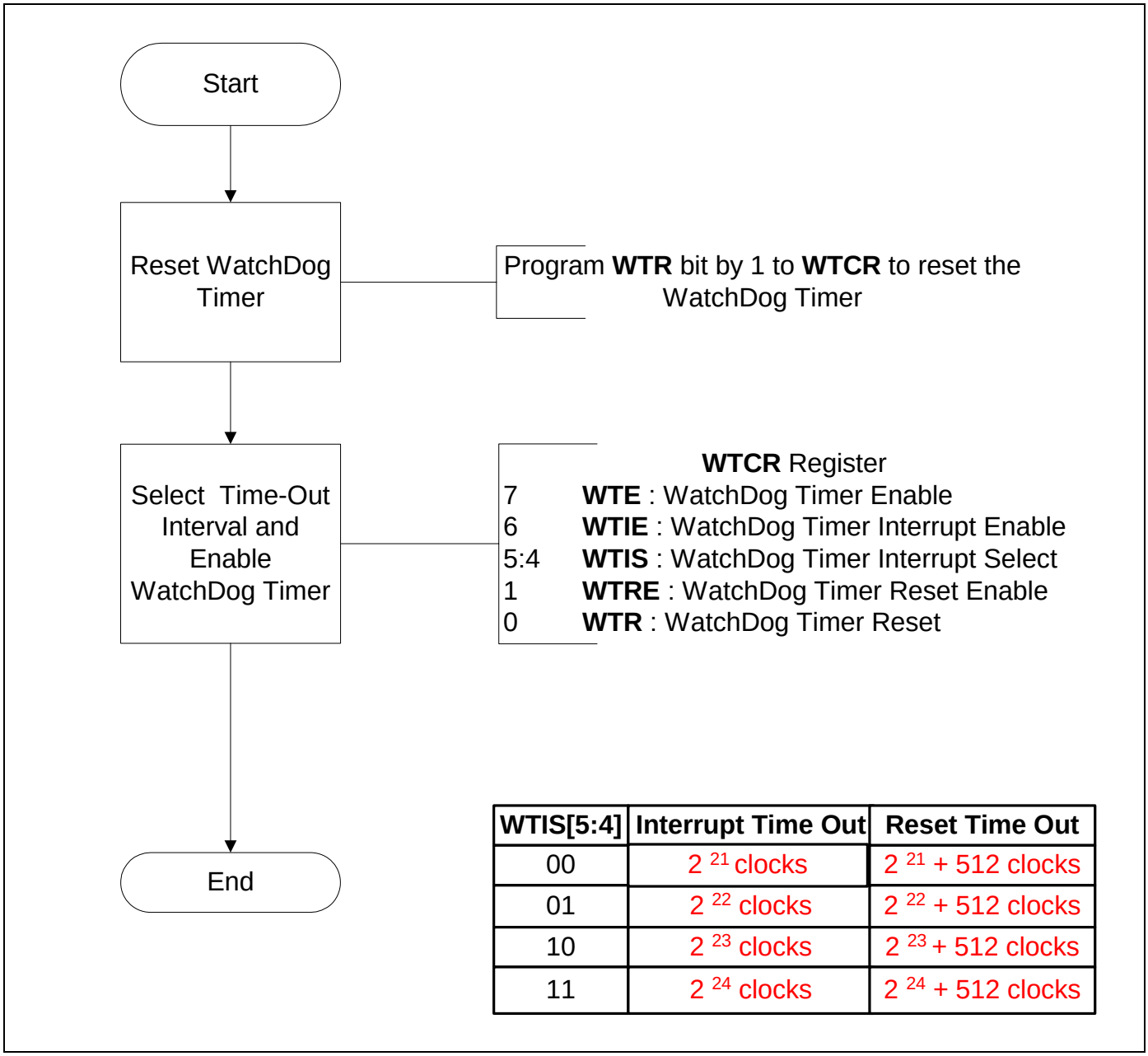
The register **WTCR** is used to control watchdog timer. The bit **[WTR]** should be set before enable watchdog timer. It ensures that the watchdog timer restarts from a known state. Figure 4-4 and Figure 4-5illustrate how to use watchdog timer.
Table 4-2 list the WatchDog Timeout period.

Table 4-2 WatchDog Timer Reset Time

WTIS[5:4]	Interrupt Time-out	Reset Time-out	Actual time
00	2^{21} clocks	2^{21} + 512 clocks	35.79 sec
01	2^{22} clocks	2^{22} + 512 clocks	71.58 sec
10	2^{23} clocks	2^{23} + 512 clocks	143.16 sec
11	2^{24} clocks	2^{24} + 512 clocks	286.33 sec

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

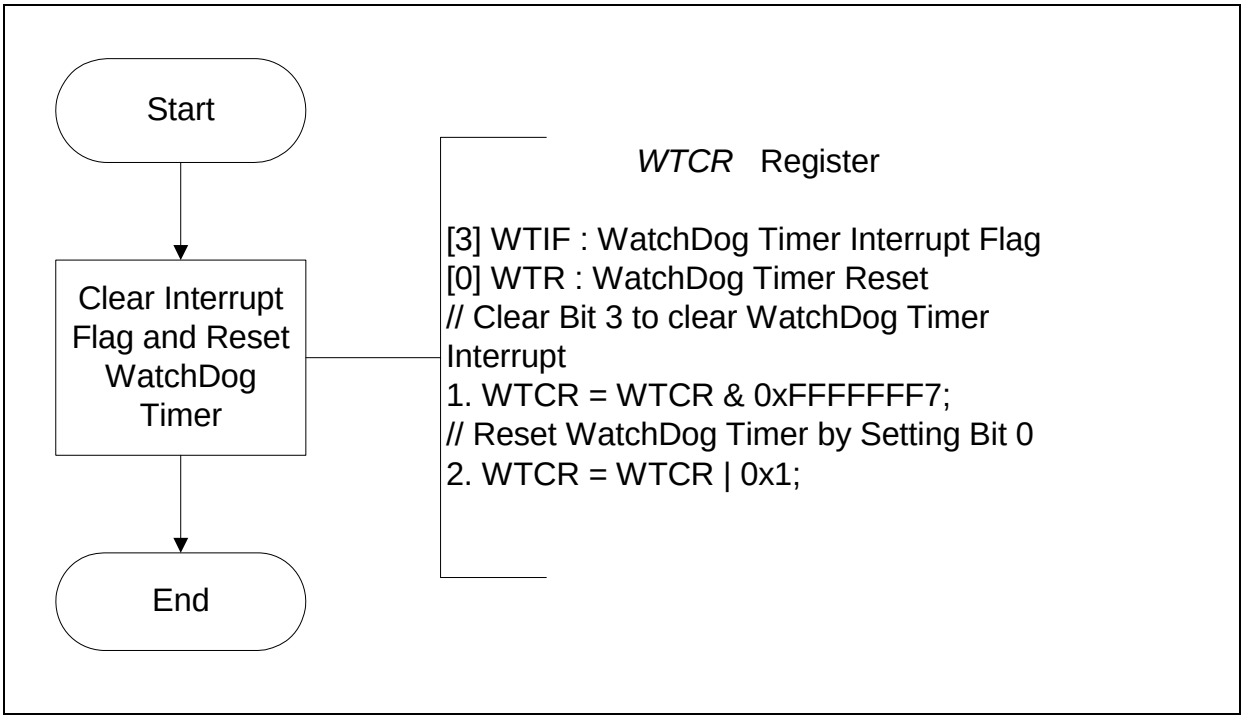
Figure 4-4 Enable Watchdog Timer



WTIS[5:4]	Interrupt Time Out	Reset Time Out
00	2^{21} clocks	2^{21} + 512 clocks
01	2^{22} clocks	2^{22} + 512 clocks
10	2^{23} clocks	2^{23} + 512 clocks
11	2^{24} clocks	2^{24} + 512 clocks

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 4-5 Watchdog Timer ISR



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

4.5 Sample Code

4.5.1 Simple Timer Interrupt handler

The ISR increases the software counter and clear the timer interrupt status.

```
/*---------------------------------------------------*/
/* Function :  TimerIntHandler()                     */
/*          Timer interrupt service routine          */
/*---------------------------------------------------*/
__irq void TimerIntHandler()
{
    volatile unsigned int data;

    data = inpw(TISR);

    /*----- check timer 0 -----*/
    if (data & 0x00000001)
    {
        count0++;
        outpw(TISR, 0x02);          /* clear TIF0 */
    }

    /*----- check timer 1 -----*/
    if (data & 0x00000002)
    {
        count1++;
        outpw(TISR, 0x01); /* clear TIF1 */
    }

} /* end TimerIntHandler */
```

4.5.2 Timer Sample Program

This program tests timer's three operation modes.

1. *One-shot* mode : Enable timer 0 and timer 1. Set one is one-shot mode, and the other is periodic mode. When periodic mode timer interrupts 5 times, check whether or not the one-shot mode timer only interrupt 1 time.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

2. *Periodic* mode : Set timer 0 is 2-ticks per second, and timer 1 is 1-tick per second. Check whether the frequency is correct or not with your watch.

3. *Toggle* mode : Set timer 0 is 2-ticks per second, and timer 1 is 1-tick per second. Check whether the frequency is correct or not with your watch.

```
/*-----------------------------------------------------*/
/* Function :  OneShotModeTest ()          */
/*-----------------------------------------------------*/
void OneShotModeTest()
{
    int loop = 5;

    printf("One-shot mode test .....>\n");

    /*----- disable timer -----*/
    outpw(TCR0, 0);
    outpw(TCR1, 0);

    /*----- clear interrupt flag bit -----*/
    outpw(TISR, 0);             /* clear for safty */
    count0 = count1 = 0;

    /*----- timer 0  :  one-shot mode, 8 tick/sec (pclk = 10MHz) -----*/
    outpw (TICR0, 0x30d4);
    outpw (TCR0, 0x60000063);

    /*----- timer 1  :  periodic mode, 8 tick/sec (LM clk= 1MHz) -----*/
    outpw (TICR1, 0x1E848);
    outpw (TCR1, 0x68000000);

    while (count1 < loop) ;
    if (count0 == 1 && !(inpw(TCR0) & 0x40000000))
       printf("    channel 0 OK.\n");
    else
       printf("    channel 0 Fail.\n");

    /*----- disable timer -----*/
    outpw(TCR0, 0);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    outpw(TCR1, 0);

    /*----- clear interrupt flag bit -----*/
    outpw(TISR, 0);             /* clear for safty */
    count0 = count1 = 0;

    /*----- timer 0  :  periodic mode, 8 tick/sec (pclk = 10MHz) -----*/
    outpw (TICR0, 0x30d4);
    outpw (TCR0, 0x68000063);

    /*----- timer 1  :  one-shot mode, 8 tick/sec (LM clk= 1MHz) -----*/
    outpw (TICR1, 0x1E848);
    outpw (TCR1, 0x60000000);

    while (count0 < loop) ;
    if (count1 == 1 && !(inpw(TCR1) & 0x40000000))
       printf("     channel 1 OK.\n");
    else
       printf("     channel 1 Fail.\n");

} /* end OneShotMode test */


/*-----------------------------------------------------*/
/* Function :  PeriodicModeTest ()             */
/*-----------------------------------------------------*/
void PeriodicModeTest()
{
    int i, loop = 5;
    unsigned int begin0, begin1;

    printf("Periodic mode test (timer0  :  1 tick/sec, timer1  :  2 tick/sec) .....>\n");

    /*----- disable timer -----*/
    outpw(TCR0, 0);
    outpw(TCR1, 0);

    /*----- clear interrupt flag bit -----*/
    outpw(TISR, 0);             /* clear for safty */
    begin0 = begin1 = count0 = count1 = 0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    /*----- timer 0  :  periodic mode, 2 tick/sec (pclk = 10MHz) -----*/
    outpw(TICR0, 0xC350);
    outpw(TCR0, 0x70000063);

    /*----- timer 1  :  periodic mode, 1 tick/sec (LM clk = 1MHz) -----*/
    outpw(TICR1, 0xF4240);
    outpw(TCR1, 0x70000000);

    for (i=0; i<loop; i++)
    {
        printf ("Timer1 :  %d,  Timer2 :  %d\n", begin0=count0, begin1=count1);
        while (begin0 == count0 && begin1 == count1);
    }

    if (begin0 == begin1 * 2)
        printf("     frequency OK.\n");
    else
        printf("     frequency Error.\n");

} /* end PeriodicModeTest */


/*---------------------------------------------------*/
/* Function :  ToggleModeTest ()                     */
/*---------------------------------------------------*/
void ToggleModeTest()
{
    int i, loop = 5;
    unsigned int begin0, begin1;

    printf("Toggle mode test (timer0  :  2 tick/sec, timer1  :  1 tick/sec) .....>\n");

    /*----- disable timer -----*/
    outpw(TCR0, 0);
    outpw(TCR1, 0);

    /*----- clear interrupt flag bit -----*/
    outpw(TISR, 0);          /* clear for safty */
    begin0 = begin1 = count0 = count1 = 0;

    /*----- timer 0  :  periodic mode, 2 tick/sec (pclk = 10MHz) -----*/
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    outpw(TICR0, 0xC350);
    outpw(TCR0, 0x70000063);

    /*----- timer 1  :  periodic mode, 1 tick/sec (LM clk = 1MHz) -----*/
    outpw(TICR1, 0xF4240);
    outpw(TCR1, 0x70000000);

    for (i=0; i<loop; i++)
    {
        printf ("Timer1 :  %d,  Timer2 :  %d\n", begin0=count0, begin1=count1);
        while (begin0 == count0 && begin1 == count1);
    }

    if (begin0 == begin1 * 2)
        printf("     frequency OK.\n");
    else
        printf("     frequency Error.\n");

} /* end ToggleModeTest */


/*---------------------------------------------------*/
/* Function :  main ()                               */
/*---------------------------------------------------*/
int main()
{
    int temp;
    unsigned int oldvect;

    /*----- disable timer -----*/
    outpw(TCR0, 0);
    outpw(TCR1, 0);

    /*----- enable CPSR I bit -----*/
    __asm
    {
        MRS    temp, CPSR
        AND    temp, temp, 0x7F
        MSR    CPSR_c, temp
    }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    /*----- replace IRQ exception entry with timer ISR -----*/
    oldvect = *(volatile unsigned int *)0x38;
    *(volatile unsigned int *)0x38 = (unsigned int) TimerIntHandler;

    /*----- set AIC bit 7 & 8 for timer channel set timer 0 and timer 1 */
    outpw (AIC_SCR7, 0x41);  /* high-level sensitive, priority level 1 */
    outpw (AIC_SCR8, 0x41);  /* high-level sensitive, priority level 1 */

    outpw (AIC_MDCR, 0x7FFFE);      /* disable all interrupts */
    outpw (AIC_MECR, 0x180);        /* trun on bit 7 & 8 */
    if (inpw (AIC_IMR) != 0x180)
        printf("AIC mask enable fail !!\n");

    /*----- Begin testing -----*/
    printf(">>>>> Timer Test Program <<<<<\n");

    OneShotModeTest();

    PeriodicModeTest();

    ToggleModeTest();

    return 0;

} /* end main */
```

4.5.3 Watchdog Timer Sample Program

This program has two features :

1. Time-out interval test : Watchdog timer has four time-out interval levels. Check whether the time-out interval is correct or not.
2. Reset test : When interrupt occurred, we clear the watchdog three times. After we don't clear watchdog timer, the system will reset.

```
/*-----------------------------------------------------*/
/* Function :  WatchDog_ISR ()                         */
/*-----------------------------------------------------*/
void WatchDog_ISR()
{
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    volatile unsigned int wtcr_val;

    wtcr_val = inpw(WTCR);
    wtcr_val &= 0xFFFFFFF7;     /* clear WTIF */

    IsWdtTimeOut = TRUE;

    if (IsClearWdt == TRUE)  wtcr_val |= 0x01;  /* WTR */
    outpw(WTCR, wtcr_val);

} /* end WatchDog_ISR() */

/*---------------------------------------------------*/
/* Function :  Timer1_I ()                           */
/*---------------------------------------------------*/
void Timer1_ISR()
{
  tTicks++;

  outpw(TISR, 0x01);      /* clear TIF1 */

  if (inpw(WTCR) & 0x04)   IsWdtResetFlagSet = TRUE;    /* cehck WDT reset flag bit */

} /* end Timer1_ISR */

/*---------------------------------------------------*/
/* Function :  IRQIntHandler ()                      */
/*---------------------------------------------------*/
__irq void IRQIntHandler()
{
  volatile unsigned int isr;

  isr = inpw(AIC_ISR);

  if (isr & 0x02)  /* WDT */
     WatchDog_ISR();

  if (isr & 0x100)  /* General timer 1 */
     Timer1_ISR();

} /* end IRQIntHandler() */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
/*---------------------------------------------------*/
/* Function :  interval_test ()                      */
/*---------------------------------------------------*/
void interval_test()
{
  volatile unsigned int btime, etime;
  unsigned int wtcr_data;
  int i;

  printf("Time-out interval test .....>\n");

  /*----- disable timer -----*/
  outpw(TCR0, 0);
  outpw(TCR1, 0);

  /*----- clear interrupt flag bit -----*/
  outpw(TISR, 0);  /* clear for safty */

  /*----- timer 1  :  periodic mode, 100 tick/sec (LM clk= 1MHz) -----*/
  outpw(TICR1, 0x2710);
  outpw(TCR1, 0x68000000);

  /*----- */
  printf("     (Note that  :  1 second has 100 ticks)\n");
  IsClearWdt = TRUE;
  for (i=0; i<4; i++)
  {
    btime = tTicks;
    printf("btime = %x\n", tTicks);
    wtcr_data = (0xC3 | ((i & 0x03) << 4));  /* WTE, WTIE, WTIS=i, WTRE, WTR */
    printf("wtcr_data = %x\n", wtcr_data);
    outpw(WTCR, wtcr_data);
    while (IsWdtTimeOut != TRUE)  ;
    etime = tTicks;
    printf("etime = %x\n", tTicks);
    outpw(WTCR, 0);                    /* disable and clear watchdog timer */
    IsWdtTimeOut = FALSE;
    printf("WTIS = %x  :  start_ticks = %d, end_ticks = %d, total = %d ticks\n",
            i, btime, etime, etime-btime);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    }

} /* end interval_test() */


/*---------------------------------------------------------*/
/* Function :  reset_test ()                               */
/*---------------------------------------------------------*/
void reset_test()
{
    int i, loop = 3;

    /*----- disable timer -----*/
    outpw(TCR0, 0);
    outpw(TCR1, 0);

    /*----- clear interrupt flag bit -----*/
    outpw(TISR, 0);  /* clear for safty */

    /*----- timer 1  :  periodic mode, 100 tick/sec (LM clk= 1MHz) -----*/
    outpw(TICR1, 0x2710);
    outpw(TCR1, 0x68000000);

    /*----- */
    outpw(WTCR, 0xC3);          /* WTE, WTIE, WTIS=0, WTRE, WTR */
    printf("\nTime-out reset test .....>\n");
    printf("     (Note that  :  1 second has 100 ticks)\n");

    IsClearWdt = TRUE;
    IsWdtTimeOut = FALSE;
    i = 0;
    while (IsWdtResetFlagSet == FALSE)
    {
      if (IsWdtTimeOut == TRUE)
      {
            i++;
            IsWdtTimeOut = FALSE;
            printf("     Time-out(%d)  :  ticks = %d\n", i, tTicks);

            if (i >= loop)    IsClearWdt = FALSE;
      }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
  }
  printf("     Reset Flag Set !  :  ticks = %d\n", tTicks);
  if (i < loop)
       printf("     ERROR ! reset flag is set during normal operation, i = %d, loop = %d\n", i, loop);
  else
       printf("     Reset flag OK.\n");

} /* end reset_test() */


/*-----------------------------------------------------*/
/* Function :  main ()                                 */
/*-----------------------------------------------------*/
int main()
{
  int temp;
  unsigned int oldvect;

  /*----- enable CPSR I bit -----*/
  __asm
  {
     MRS    temp, CPSR
     AND    temp, temp, 0x7F
     MSR    CPSR_c, temp
  }

  /*----- replace IRQ exception entry with timer ISR -----*/
  oldvect = *(volatile unsigned int *)0x38;
  *(volatile unsigned int *)0x38 = (unsigned int) IRQIntHandler;

  /*----- set AIC bit 1 & 8 for wtachdog timer and general timer 1 */
  outpw (AIC_SCR1, 0x41);  /* high-level sensitive, priority level 1 */
  outpw (AIC_SCR8, 0x41);  /* high-level sensitive, priority level 1 */

  outpw (AIC_MDCR, 0x7FFFE); /* disable all interrupts */
  outpw (AIC_MECR, 0x102);         /* turn bit 1, 8 */
  if (inpw (AIC_IMR) != 0x102)  printf("AIC mask enable fail !!\n");

  /*----- Begin testing -----*/
  printf(">>>>> Watch Dog Timer Test <<<<<\n");
  interval_test();
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    reset_test();

    /*----- restore IRQ exception entry -----*/
    *(volatile unsigned int *)0x38 = oldvect;
    return 0;

} /* end main */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5 AIC (Advanced Interrupt Controller)

5.1 Overview

The NUC740 incorporates the **advanced interrupt controller (AIC)** that is capable of dealing with the interrupt requests from 32 different interrupt sources. Currently, only 18 interrupt sources are defined. Each interrupt source is uniquely assigned to an *interrupt channel (1 to 18)*. Every interrupt channel can be enabled or disabled individually. A channel is treated as active if its corresponding interrupt source has an interrupt request. Several status registers are used to distinguish the state of these interrupt channels. The AIC will assert an interrupt request to CPU core (ARM7TDMI) only if there's at least one interrupt channel is active and enabled.

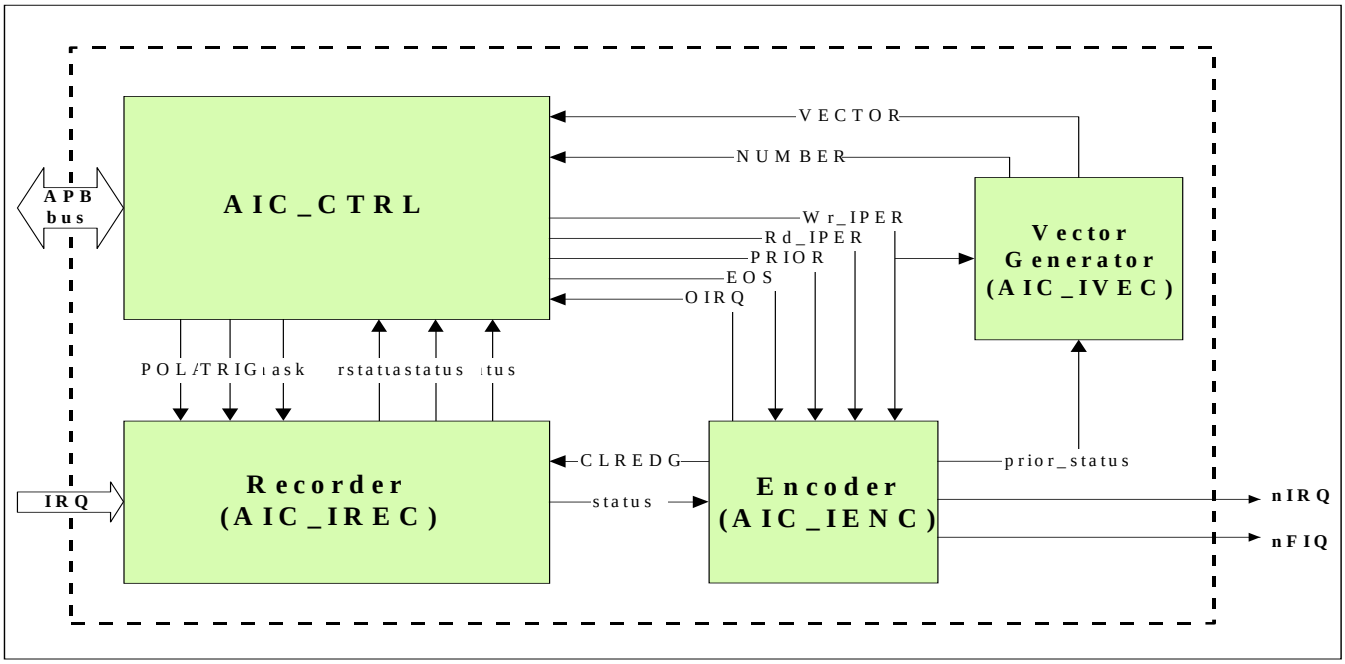
The software driver can implement a priority scheme based on the status register. However, the AIC itself implements a proprietary eight-level priority scheme to improve the interrupt dispatch time. It differentiates the available 18 interrupt sources into eight priority levels, level 0 is the highest one and the level 7 is the lowest. Within each priority level, a lower channel number interrupt source has a higher priority. The AIC will assert the FIQ request if the active and enabled interrupt channel is assigned to priority level 0. For the interrupt channels that are assigned to other priority level, AIC will assert an IRQ request. The IRQ can be preempted by the occurrence of the FIQ. Interrupt nesting is performed automatically by the AIC.

Although the internal interrupt sources of NUC740 are intrinsically high-level sensitive, the driver can configure each interrupt source to be either low-level sensitive, high-level sensitive, negative-edge triggered, or positive-edge triggered.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5.2 Block Diagram

Figure 5-1 AIC block diagram



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5.3 Registers

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

Table 5-1 AIC Register Definition

Register	Address	R/W	Description	Reset Value
AIC_SCR1	0xFFF8.2004	R/W	Source Control Register 1	0x0000.0047
AIC_SCR2	0xFFF8.2008	R/W	Source Control Register 2	0x0000.0047
AIC_SCR3	0xFFF8.200C	R/W	Source Control Register 3	0x0000.0047
AIC_SCR4	0xFFF8.2010	R/W	Source Control Register 4	0x0000.0047
AIC_SCR5	0xFFF8.2014	R/W	Source Control Register 5	0x0000.0047
AIC_SCR6	0xFFF8.2018	R/W	Source Control Register 6	0x0000.0047
AIC_SCR7	0xFFF8.201C	R/W	Source Control Register 7	0x0000.0047
AIC_SCR8	0xFFF8.2020	R/W	Source Control Register 8	0x0000.0047
AIC_SCR9	0xFFF8.2024	R/W	Source Control Register 9	0x0000.0047
AIC_SCR10	0xFFF8.2028	R/W	Source Control Register 10	0x0000.0047
AIC_SCR11	0xFFF8.202C	R/W	Source Control Register 11	0x0000.0047
AIC_SCR12	0xFFF8.2030	R/W	Source Control Register 12	0x0000.0047
AIC_SCR13	0xFFF8.2034	R/W	Source Control Register 13	0x0000.0047
AIC_SCR14	0xFFF8.2038	R/W	Source Control Register 14	0x0000.0047
AIC_SCR15	0xFFF8.203C	R/W	Source Control Register 15	0x0000.0047
AIC_SCR16	0xFFF8.2040	R/W	Source Control Register 16	0x0000.0000
AIC_SCR17	0xFFF8.2044	R/W	Source Control Register 17	0x0000.0000
AIC_SCR18	0xFFF8.2048	R/W	Source Control Register 18	0x0000.0000
AIC_IRSR	0xFFF8.2100	R	Interrupt Raw Status Register	0x0000.0000
AIC_IASR	0xFFF8.2104	R	Interrupt Active Status Register	0x0000.0000
AIC_ISR	0xFFF8.2108	R	Interrupt Status Register	0x0000.0000
AIC_IPER	0xFFF8.210C	R	Interrupt Priority Encoding Register	0x0000.0000
AIC_ISNR	0xFFF8.2110	R	Interrupt Source Number Register	0x0000.0000
AIC_IMR	0xFFF8.2114	R	Interrupt Mask Register	0x0000.0000
AIC_OISR	0xFFF8.2118	R	Output Interrupt Status Register	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

AIC_MECR	0xFFF8.2120	W	Mask Enable Command Register	Undefined
AIC_MDCR	0xFFF8.2124	W	Mask Disable Command Register	Undefined
AIC_SSCR	0xFFF8.2128	W	Source Set Command Register	Undefined
AIC_SCCR	0xFFF8.212C	W	Source Clear Command Register	Undefined
AIC_EOSCR	0xFFF8.2130	W	End of Service Command Register	Undefined
AIC_TEST	0xFFF8.2200	W	ICE/Debug mode Register	Undefined

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5.4 Functional Descriptions

5.4.1 NUC740 Interrupt Sources

Interrupt channel 1, 6 – 18 are mapped to internal peripheral devices, and 2 – 5 are mapped to to 18 are defined to internal peripherals and external devices. The channel 0 and all others beyond channel 18 are undefined in current implementation. Figure 5-2 lists the interrupt source and its channel number.

Figure 5-2 NUC740 Interrupt Sources

Channel	**Name**	Priority	**Mode**	Source
		Highest		
1	WDTINT		Positive Level	Watch Dog Timer Interrupt
2	nIRQ0		Programmable	External Interrupt 0
3	nIRQ1		Programmable	External Interrupt 1
4	nIRQ2		Programmable	External Interrupt 2
5	nIRQ3		Programmable	External Interrupt 3
6	UARTINT		Positive Level	UART Interrupt
7	TINT0		Positive Level	Timer Interrupt 0
8	TINT1		Positive Level	Timer Interrupt 1
9	USBINT0		Positive Level	USB Interrupt 0
10	USBINT1		Positive Level	USB Interrupt 1
11	Reserved		Reserved	Reserved
12	Reserved		Reserved	Reserved
13	EMCTXINT0		Positive Level	EMC TX Interrupt 0
14	EMCTXINT1		Positive Level	EMC TX Interrupt 1
15	EMCRXINT0		Positive Level	EMC RX Interrupt 0
16	EMCRXINT1		Positive Level	EMC RX Interrupt 1
17	GDMAINT0		Positive Level	GDMA Channel Interrupt 0
18	GDMAINT1		Positive Level	GDMA Channel Interrupt 1
19 ~ 31	-	Lowest	-	Reserved for further use

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5.4.2 Interrupt channel configuration

Each interrupt channel has an independent source control register to set its type and priority. The interrupt type of all NUC740 internal peripherals is positive-level triggered. This shouldn't be changed during normal operation. For the channel 2, 3, 4 and 5, 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. Figure 5-3 shows the content of source control register.

Figure 5-3 Source Control Register

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

5.4.3 Interrupt Masking

The NUC740 AIC provides a set of registers to mask individual interrupt channel. The **Mask Enable Command Register (AIC_MECR)** is used to enable interrupt. Write 1 to a bit of MECR will enable the corresponding interrupt channel. Oppositely, the **Mask Disable Command Register (AIC_MDCR)** is used to disable the interrupt. Write 1 to a bit of MDCR will disable the corresponding interrupt channel. Write 0 to a bit of AIC_MECR or AIC_MDCR has no effect. Therefore, the device driver can arbitrarily change these two registers without keeping their original values. If it's necessary, the device driver can read the **Interrupt Mask Register (AIC_IMR)** to know whether the interrupt channel is enabled or disabled. If the interrupt channel is enabled, its corresponding bit is read as 1, otherwise 0.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5.4.4 Interrupt Clearing and Setting

For the interrupt channels that are level sensitive, the device driver doesn't need to write the **Source Clear Command Register (AIC_SCCR)** or **End of Service Command Register (AIC_EOSCR) to** clear any AIC status. As soon as the device's interrupt status has been was cleared, the AIC de-asserts the interrupt request. For the interrupt channels that are edge-triggered, the device driver must clear AIC status to de-assert the interrupt request. To clear AIC status, the device driver may either write **Source Clear Command Register (AIC_SCCR)** or **End of Service Command Register (AIC_EOSCR)**. Write 1 to a bit of AIC_SCCR will clear the corresponding interrupt. The usage of AIC_EOSCR will be discussed in the section *Hardware Priority Scheme*. The register **Source Set Command Register (AIC_SSCR)** is used to set an interrupt channel when it is programmed to edge-triggered. Write 1 to a bit of AIC_SSCR will set the corresponding interrupt. This feature is useful in auto-testing or software debugging

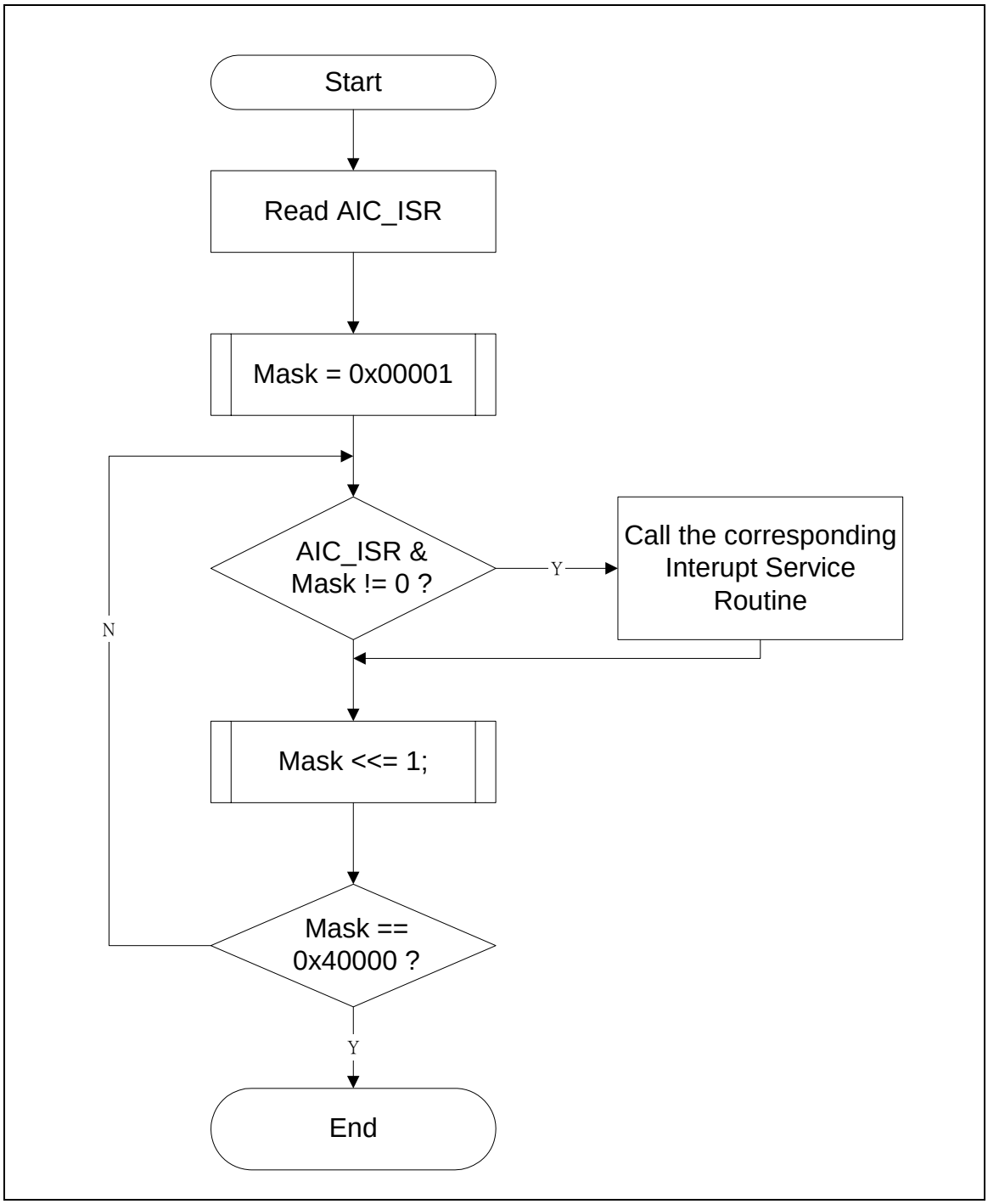
5.4.5 Software Priority Scheme

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

Generally, there's a function table to keep the interrupt service routines of internal peripherals and external devices. When the interrupt is recognized by CPU core, the FIQ or IRQ exception handler is executed firstly. Then it will call the proper interrupt service routine according to the AIC_ISR content. Figure 5-4 demonstrates a sequential priority scheme where channel 1 has the highest priority and channel 17 18 has the lowest priority.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 5-4 Sequential Priority Scheme



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5.4.6 Hardware Priority Scheme

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

1. Reads IRQ Priority Encoding Register (AIC_IPER) to get the Vector(IRQ Channel x 4), and at this mean time, the AIC_ISNR will be loaded by the current interrupt channel number, the Vector (IRQ Channel Number x 4) represents the interrupt channel number that is active, enabled, and has the highest priority, multiplied by 4, then stored on the AIC_IPER. The data (Vector) got from AIC_IPER is convenient for the following interrupt service route address calculation ., enabled, and has the highest priority.

2. Branch to the corresponding interrupt service routine by adding Vector to the base of interrupt service routine table.

3. Write any value to AIC_EOSCR to finish the interrupt.

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

In contrast with the software priority scheme, the Vector provides a quicker method to reach the interrupt service routine. The branch address can be easily got that adds Vector to the base of interrupt service routine table. Figure 5-5 shows an example assembly code.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 5-5 Interrupt Service Routine with Vector

```
STMFD   SP!, {R0-R2}             ; Push registers on stack

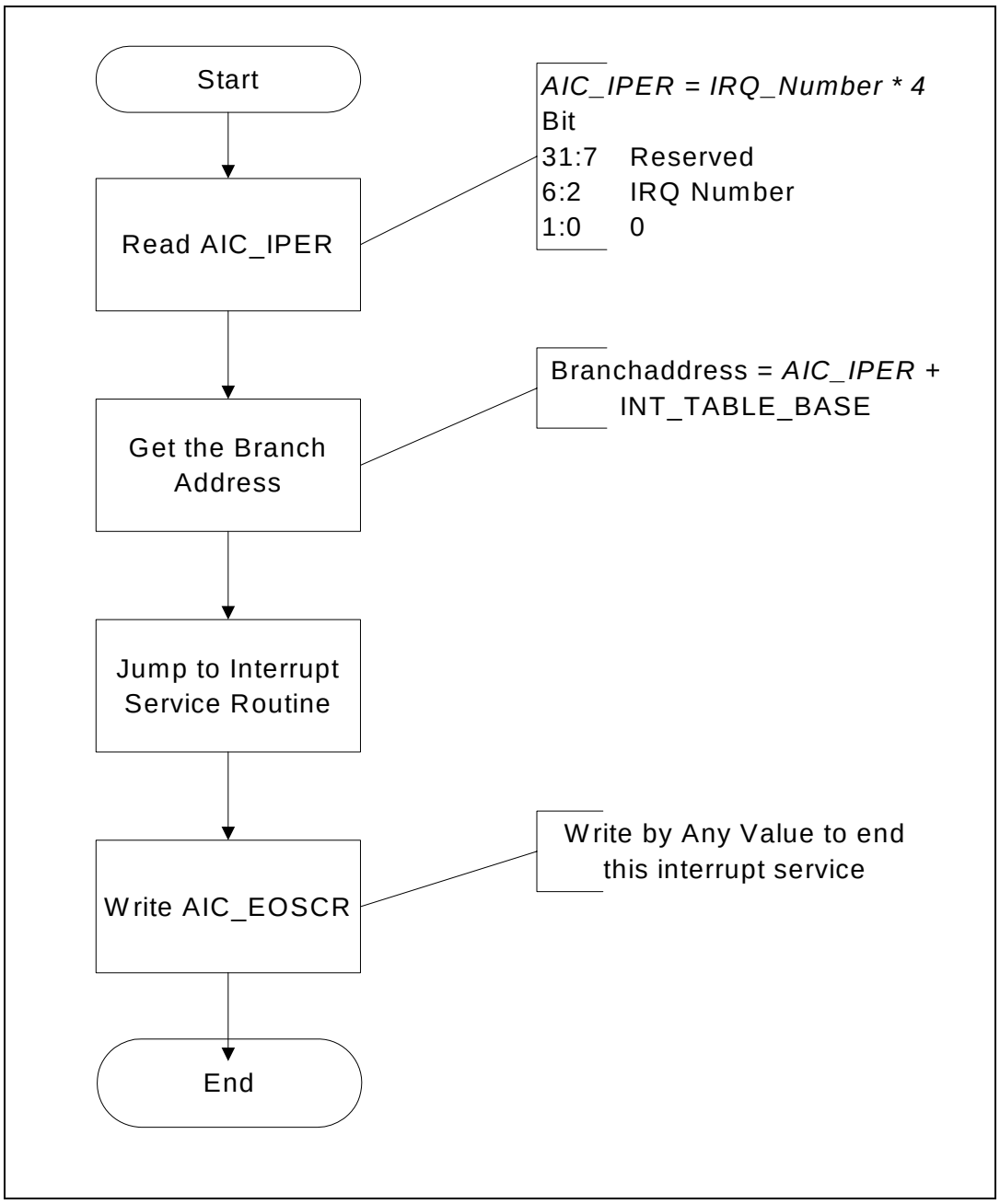
;Goto_Handler, jump to the correct handler
LDR     R2, =AICBase
LDR     R1, [R2,#AIC_IPER]       ; gets the highest pending
vector
LDR     PC, [PC, R1]             ; jump to correct handler
NOP

;table of handler start address
DCD     Fake_Interrupt
DCD     Int1_Interrupt
DCD     Int2_Interrupt
DCD     Int3_Interrupt
DCD     Int4_Interrupt
DCD     Int5_Interrupt
DCD     Int6_Interrupt
DCD     Int7_Interrupt
DCD     Int8_Interrupt
DCD     Int9_Interrupt
DCD     Int10_Interrupt
DCD     Int11_Interrupt
DCD     Int12_Interrupt
DCD     Int13_Interrupt
DCD     Int14_Interrupt
DCD     Int15_Interrupt
DCD     Int16_Interrupt
DCD     Int17_Interrupt
DCD     Int18_Interrupt
DCD     Int19_Interrupt
DCD     Int20_Interrupt
DCD     Int21_Interrupt
DCD     Int22_Interrupt
DCD     Int23_Interrupt
DCD     Int24_Interrupt
DCD     Int25_Interrupt
DCD     Int26_Interrupt
DCD     Int27_Interrupt
DCD     Int28_Interrupt
DCD     Int29_Interrupt
DCD     Int30_Interrupt
DCD     Int31_Interrupt
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

It is very important that ISR must write AIC_EOSCR to restore to normal interrupt state once it read the AIC_IPER. Otherwise, the next interrupt may not be serviced correctly Figure 5-6 shows the programming flow of using hardware priority scheme.

Figure 5-6 Using hardware priority scheme



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

5.4.7 Sample Code

5.4.7.1 NUC740 Interrupt Service routine

```
/* Interrupt Handler Table */
typedef void (*fptr)();  /* function pointer */
fptr IRQ_HandlerTable[] = {         IRQ_Int1_Handler,
                                    IRQ_Int2_Handler,
                                    IRQ_Int3_Handler,
                                    IRQ_Int4_Handler,
                                    IRQ_Int5_Handler,
                                    IRQ_Int6_Handler,
                                    IRQ_Int7_Handler,
                                    IRQ_Int8_Handler,
                                    IRQ_Int9_Handler,
                                    IRQ_Int10_Handler,
                                    IRQ_Int11_Handler,
                                    IRQ_Int12_Handler,
                                    IRQ_Int13_Handler,
                                    IRQ_Int14_Handler,
                                    IRQ_Int15_Handler,
                                    IRQ_Int16_Handler,
                                    IRQ_Int17_Handler,
                                    IRQ_Int18_Handler
                    };


/*---------------------------------*/
/* interrupt service routine */
/*---------------------------------*/
void IRQ_Int1_Handler()
{
   outpw(AIC_SCCR, 0x02);
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
void IRQ_Int2_Handler()
{
  outpw(AIC_SCCR, 0x04);
}

void IRQ_Int3_Handler()
{
  outpw(AIC_SCCR, 0x08);
}

void IRQ_Int4_Handler()
{
  outpw(AIC_SCCR, 0x10);
}

void IRQ_Int5_Handler()
{
  outpw(AIC_SCCR, 0x20);
}

void IRQ_Int6_Handler()
{
  outpw(AIC_SCCR, 0x40);
}

void IRQ_Int7_Handler()
{
  outpw(AIC_SCCR, 0x80);
}

void IRQ_Int8_Handler()
{
  outpw(AIC_SCCR, 0x100);
}

void IRQ_Int9_Handler()
{
  outpw(AIC_SCCR, 0x200);
}

void IRQ_Int10_Handler()
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
  outpw(AIC_SCCR, 0x400);
}

void IRQ_Int11_Handler()
{
  outpw(AIC_SCCR, 0x800);
}

void IRQ_Int12_Handler()
{
  outpw(AIC_SCCR, 0x1000);
}

void IRQ_Int13_Handler()
{
  outpw(AIC_SCCR, 0x2000);
}

void IRQ_Int14_Handler()
{
  outpw(AIC_SCCR, 0x4000);
}

void IRQ_Int15_Handler()
{
  outpw(AIC_SCCR, 0x8000);
}

void IRQ_Int16_Handler()
{
  outpw(AIC_SCCR, 0x10000);
}

void IRQ_Int17_Handler()
{
  outpw(AIC_SCCR, 0x20000);
}

void IRQ_Int18_Handler()
{
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
  outpw(AIC_SCCR, 0x40000);
}

/*---------------------------------------------*/
/* Function  :  IRQ_IntHandler()               */
/*        interrupt service routine            */
/*---------------------------------------------*/
__irq void IRQ_IntHandler()
{
    unsigned int IPER, ISNR;

    IPER = inpw (AIC_IPER) >> 2;
    ISNR = inpw (AIC_ISNR);

    if (ISNR == IPER)
       (*IRQ_HandlerTable[ISNR-1])();

    outpw (AIC_EOSCR, 0);

} /* end IRQ_IntHandler */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

6 UART

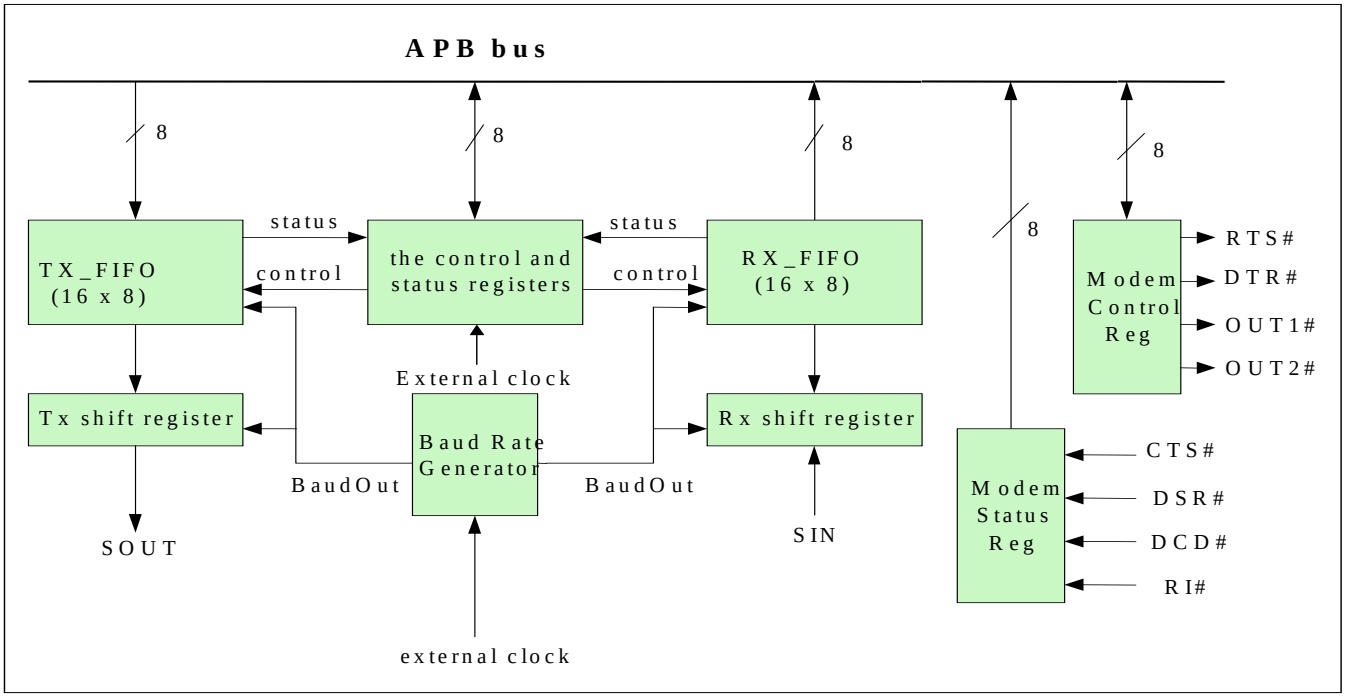
6.1 Overview

The NUC740 **Universal Asynchronous Receiver/Transmitter (UART)** performs a serial-to-parallel conversion for the input data. The character bits received from UART receive pin **(SIN)** are shifted to Receive FIFO one after by one. The driver reads the receive FIFO to get the input character. The UART also performs a parallel-to-serial conversion for output data. The driver writes the output characters to the transmitter FIFO. Then the character bits are shifted to UART transmit pin **(SOUT)** in sequence.

The UART provides control/status registers and an interrupt for device driver to control the transmitting operation, receiving operation and error handling. There are five types of interrupts including *line status interrupt*, *transmitter FIFO empty interrupt*, *receiver threshold level reaching interrupt*, *time out interrupt*, and *MODEM status interrupt*. The provided status information includes the type and condition of the transfer operations being performed by the UART, as well as any found error conditions (parity, overrun, framing, or break interrupt).

6.2 Block Diagram

Figure 6-1. UART Block Diagram



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

6.3 Registers

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
RBR	0xFFF8.0000	R	Receive Buffer Register (DLAB = 0)	Undefined
THR	0xFFF8.0000	W	Transmit Holding Register (DLAB = 0)	Undefined
IER	0xFFF8.0004	R/W	Interrupt Enable Register (DLAB = 0)	0x0000.0000
DLL	0xFFF8.0000	R/W	Divisor Latch Register (LS) (DLAB = 1)	0x0000.0000
DLM	0xFFF8.0004	R/W	Divisor Latch Register (MS) (DLAB = 1)	0x0000.0000
IIR	0xFFF8.0008	R	Interrupt Identification Register	0x8181.8181
FCR	0xFFF8.0008	W	FIFO Control Register	Undefined
LCR	0xFFF8.000C	R/W	Line Control Register	0x0000.0000
MCR	0xFFF8.0010	R/W	Modem Control Register	0x0000.0000
LSR	0xFFF8.0014	R	Line Status Register	0x6060.6060
MSR	0xFFF8.0018	R	MODEM Status Register	0x0000.0000
TOR	0xFFF8.001C	R/W	Time Out Register	0x0000.0000

6.4 Functional Descriptions

6.4.1 Baud Rate

The UART includes a programmable baud rate generator. The crystal clock input is divided by divisor to produce the clock that transmitter and receiver need. The equation is

Baud Rate = Crystal clock / (16 * [Divisor + 2])

For NUC740, the crystal clock input is 15 MHZ. The DLL and DLM registers consist of the low byte and high byte of the divisor. The DLL and DLM registers aren't accessible until the DLAB bit of LCR register is set 1. The driver should program, the correct value into the DLL/DLM registers according to the desired baud rate. Table 6-1 lists some general baud rate settings.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Table 6-1 General Baud Rate Settings

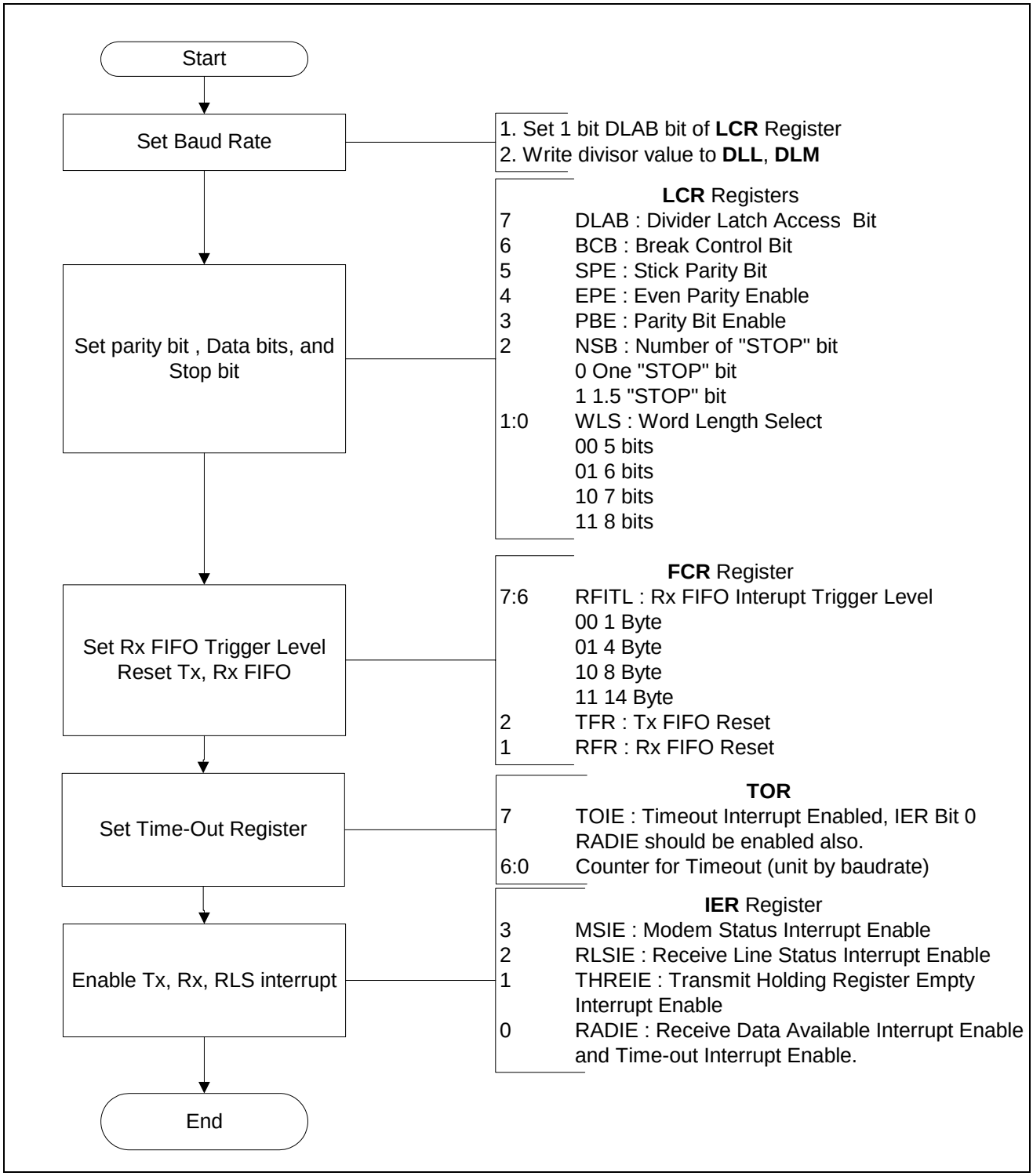
Baud Rate	DLM	DLL	Real	Error rate [%]
115200	0	6	117187.5	1.725
57600	0	14	58593.75	1.725
38400	0	22	39062.5	1.725
19200	0	47	19132.65	-0.35
9600	0	96	9566.33	-0.35

6.4.2 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 6-2 shows the initialization flow of UART.

Figure 6-2 UART initialization

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

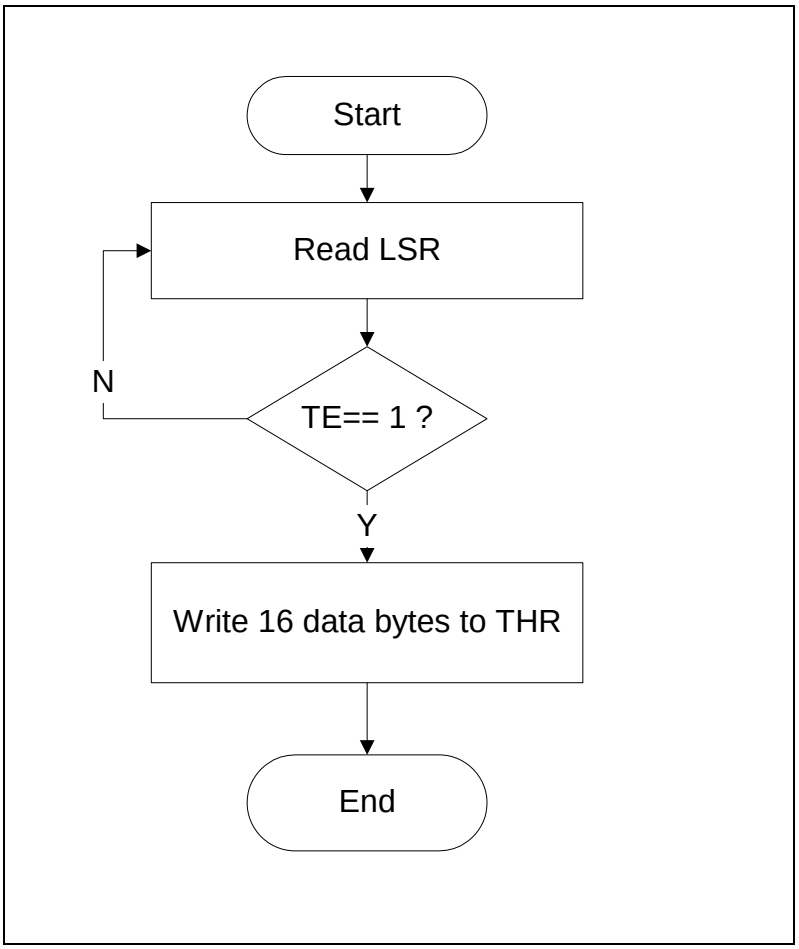


The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

6.4.3 Polled I/O Functions

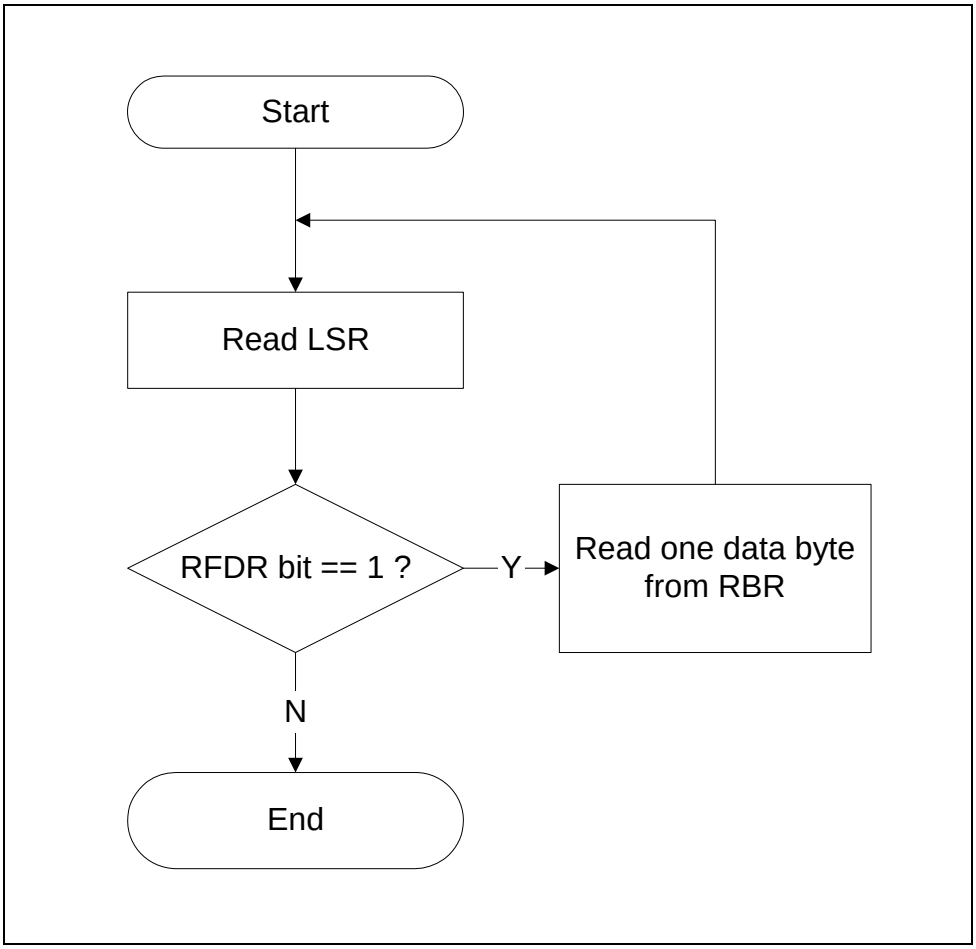
The driver can transmit and receive data through UART by polling mode. The poll functions check UART buffer by reading status register. If there's at least one data byte available in receive FIFO, the [RFDR] bit is set 1. It indicates that driver can read receive FIFO to get new data bytes. If the transmitter is empty, the [TE] bit is set 1. Then the data bytes can be written into the transmit FIFO. The data bytes in the transmit FIFO will be shifted to SOUT serially. Figure 6-3 and Figure 6-4 show the programming flow of transmit data and receive data in polling mode.

Figure 6-3 Transmit data in polling mode



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 6-4 Receive data in polling mode



6.4.4 Interrupted I/O Functions

The data bytes also can be transmitted and received through UART by interrupt control. The interrupt service routine is responsible to move data bytes from driver's buffer to transmit FIFO whenever the THRE interrupt happens. If RDA or TOUT interrupts occurs, the interrupt service routine should move the data bytes from receive FIFO to driver's buffer.
On interrupt mode, the input and output functions are different from the polling functions. They read or write the driver's buffer instead of Tx /Rx FIFO. The output function writes the data bytes into driver's buffer and then enables THRE interrupt. The ISR will read the data bytes from driver's

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

buffer and write them to the Tx FIFO when the transmitter FIFO empty interrupt occurs, or get the data bytes from Rx FIFO the driver receiving buffer when the *receiver threshold level reaching interrupt* occurs. When the input function is called, it reads the data bytes from driver's receiving buffer and then return. The Figure 6-5, Figure 6-6 and Figure 6-7 show the flow of output function, input function, and interrupt service routine.

Figure 6-5 Output function in interrupt mode

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

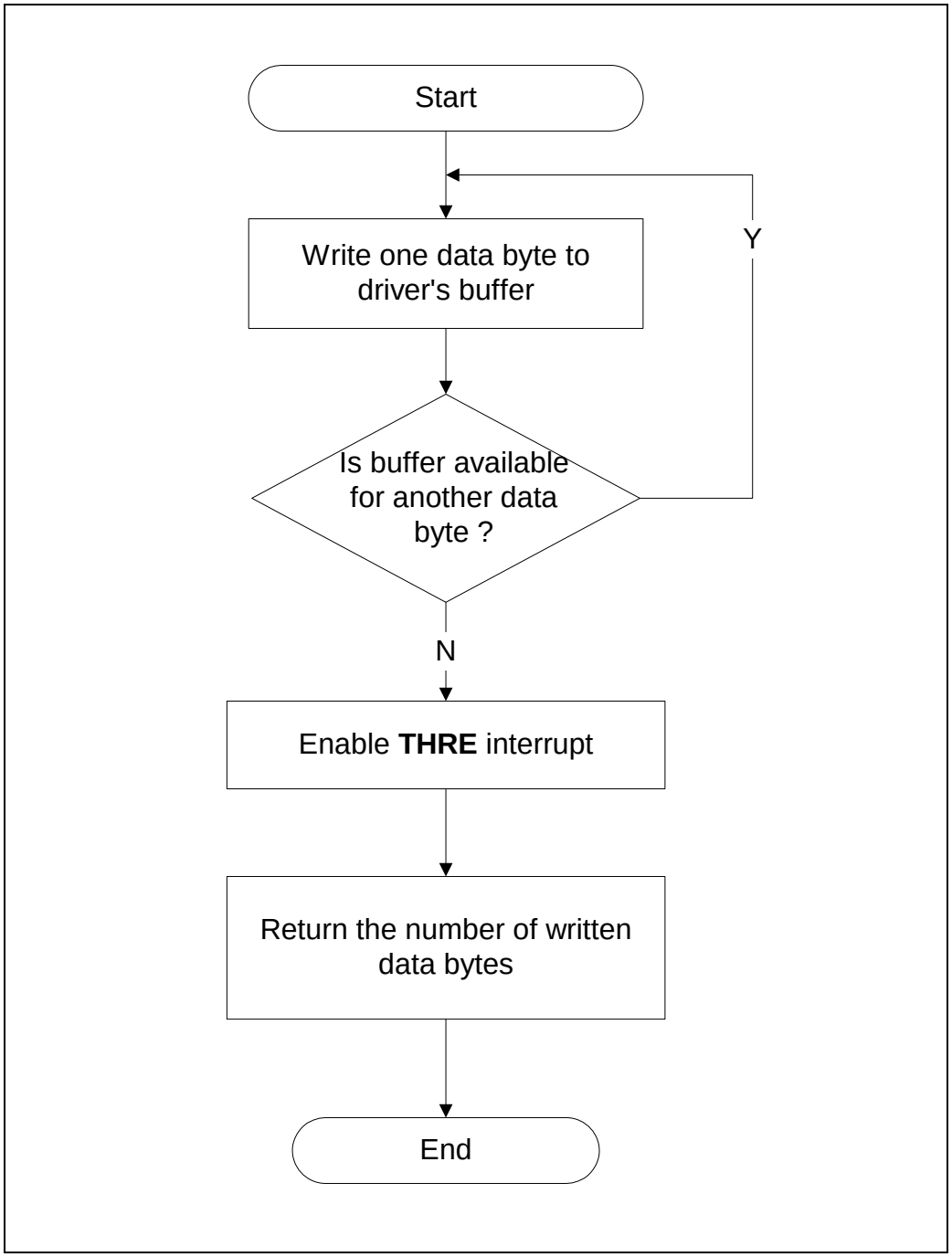
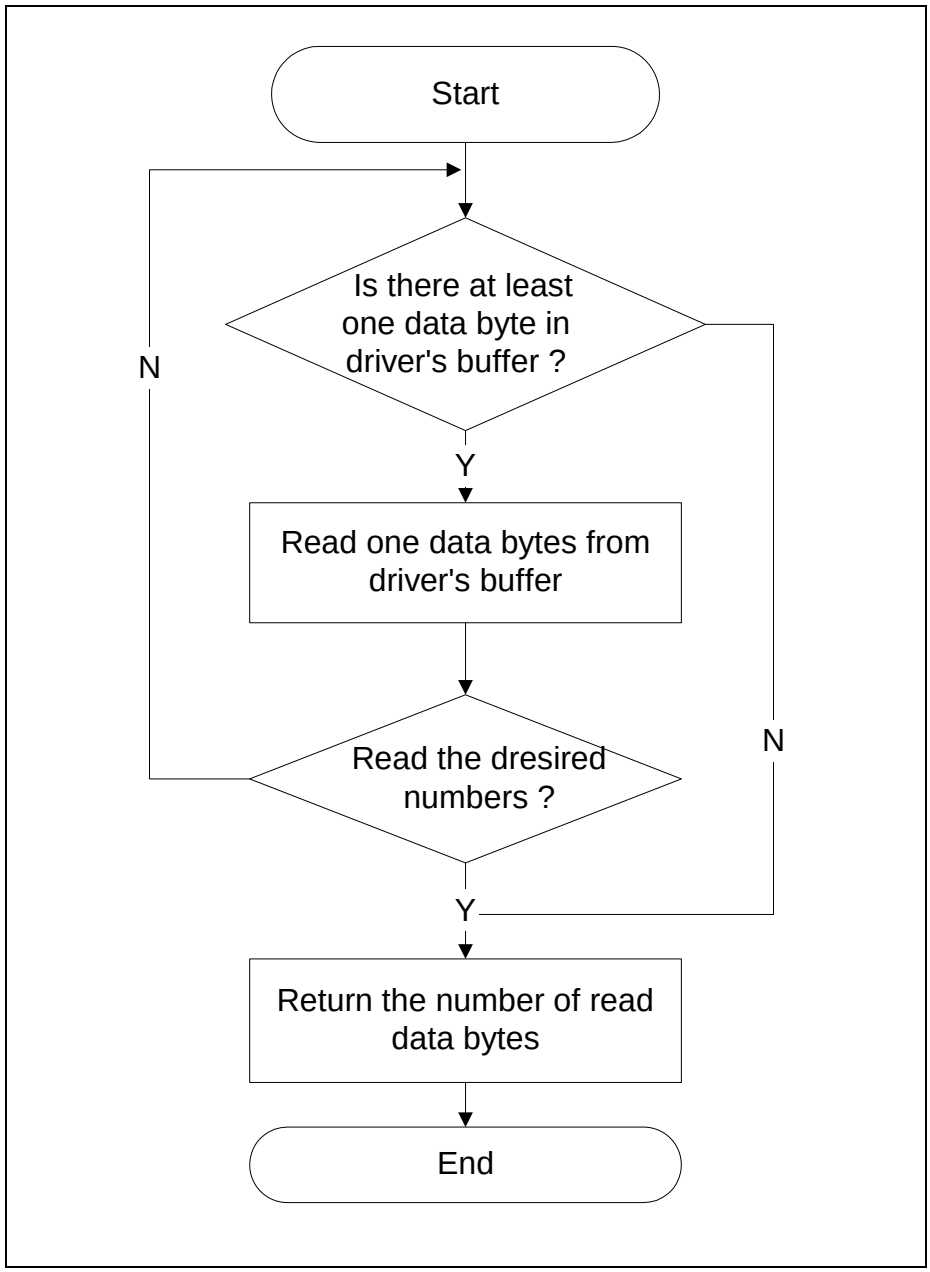


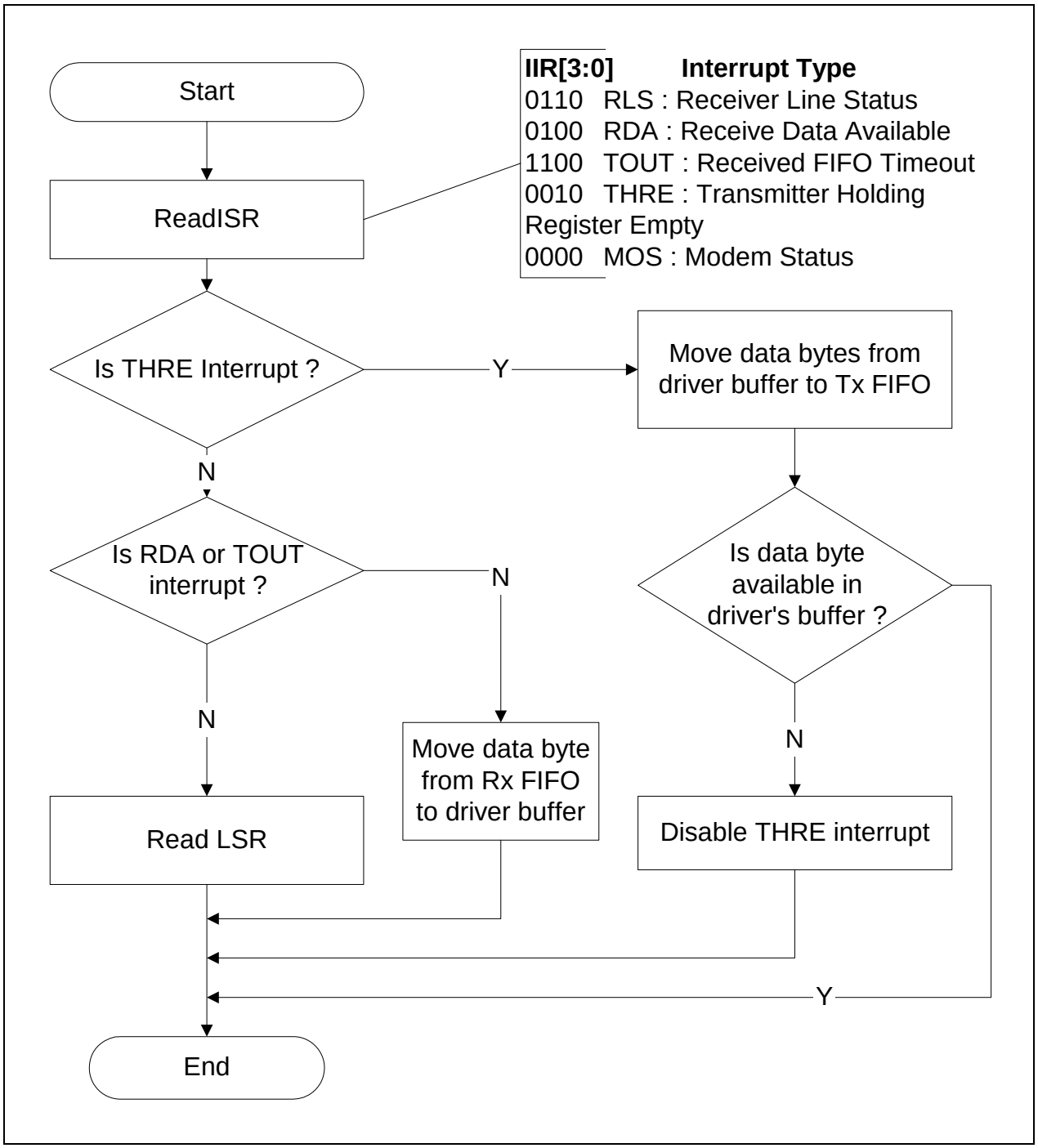
Figure 6-6 Input functions in interrupt mode

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 6-7 Interrupt Service Routine



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

6.5 Sample Code

This program receives the data from keyboard, and then transmits it into hyper terminal.

```
/*---------------------------------------------------*/
/* Function :  _commIntHandler ()                    */
/*---------------------------------------------------*/
__irq void _commIntHandler()
{
  volatile unsigned char inChar;
  volatile unsigned char comIIR;

  /*----- Determine interrupt source -----*/
  comIIR = inp(COM_IIR) & 0x0F;

  if (comIIR==4 || comIIR==12)    /* Received data available INT? */
  {
    /*----- Get all the input characters -----*/
    while (inp (COM_LSR) & 0x01)          /* Receiver data ready? */
    {
      /*----- Get the character from UART -----*/
      inChar = inp (COM_RX);

      /*----- Check if buffer full -----*/
      if (comRbytes < RXBUFSIZE)
      {
            /*----- Enqueue the character -----*/
            comRbuf[comRtail] = inChar;
            comRtail = (comRtail == (RXBUFSIZE-1)) ? 0  :  (comRtail+1);
            comRbytes++;
          }
    }
  }
  else if (comIIR == 2)            /* Transmitter holding register INT? */
  {
    if (comRhead != comRtail)
    {
      outp (COM_TX, comRbuf[comRhead]);
      comRhead = (comRhead == (RXBUFSIZE-1)) ? 0  :  (comRhead+1);
      comRbytes--;
    }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
  }

} /* end _commIntHandler */


/*---------------------------------------------------*/
/* Function :  ComInit ()                            */
/*---------------------------------------------------*/
void ComInit ()
{
    volatile char inChar;

    outp (COM_LCR, 0x80);           /* select divisor latch registers */

    outp (COM_LSB, 6);              /* Baud rate = 115200  :  15,000,000/(16*(6+2))  */
    outp (COM_MSB, 0);

    outp (COM_LCR, 0x03);           /* none parity, 8 data bits, 1 stop bits */

    outp (COM_IER, 0x01);           /* RDA interrupt source on / LOOP disable */
    outp (COM_MCR, 0x00);

    outp (COM_TOR, 0x80+0x20);    /* Timeout if more than ??? bits xfer time */

    outp (COM_FCR, 0x8F);           /* 8-byte FIFO trigger level, reset Tx and Rx FIFO */

    inChar = inp (COM_RX);          /* Reset interrupt status */

} /* end ComInit */


/*---------------------------------------------------*/
/* Function :  main ()                               */
/*---------------------------------------------------*/
int main()
{
    unsigned int oldvect;
    int temp;

    printf(">>>>> UART Tx/Rx ECHO Terminal Test Program <<<<<\n\n");
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    /*----- enable CPSR I bit -----*/
    __asm
    {
        MRS    temp, CPSR
        AND    temp, temp, 0x7F
        MSR    CPSR_c, temp
    }

    /*----- Replace IRQ exception entry with UART ISR -----*/
    oldvect = *(volatile unsigned int *)0x38;
    *(volatile unsigned int *)0x38 = (unsigned int) _commIntHandler;

    /*----- Set UART INT  :  high-level sensitive and priority level 1 ------*/
    outpw (AIC_SCR6, 0x41);

    /*----- Turn on bit 6 in AIC for UART INT -----*/
    outpw (AIC_MDCR, 0x7FFFF); /* disable all interrupts */
    outpw (AIC_MECR, 0x40);
    if (inpw (AIC_IMR) != 0x40)      printf("AIC mask enable fail !!\n");

    /*----- Begining test -----*/
    while (! (inp(COM_LSR) & 0x40)); /* wait Transmitter Empty */

    ComInit();
    outp (COM_MCR, 0x0b);                   /* OUT2, RTS, DTR */
    outp (COM_IER, 4+2+1);                  /* RLS, THRE, RDA */

    /*----- Waiting for get data -----*/
    while (1);

    outp (COM_IER, 0x00);
    *(volatile unsigned int *)0x38 = (unsigned int) oldvect;

    return 0;

} /* end main */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

7 GDMA

7.1 Overview

The NUC740 **GDMA** controller provides a data transfer mechanism without the need of CPU intervention. It can move data between two memory regions, or between memory and external devices. The GDMA has two independent channels that support single and block mode transfer. When GDMA is programmed to *single* mode, it requires a request **(nXDREQ)** for each data transfer that may be one byte, one half-word or one word. When GDMA is programmed to *block* mode, a single GDMA request will make all of the data to be transferred.

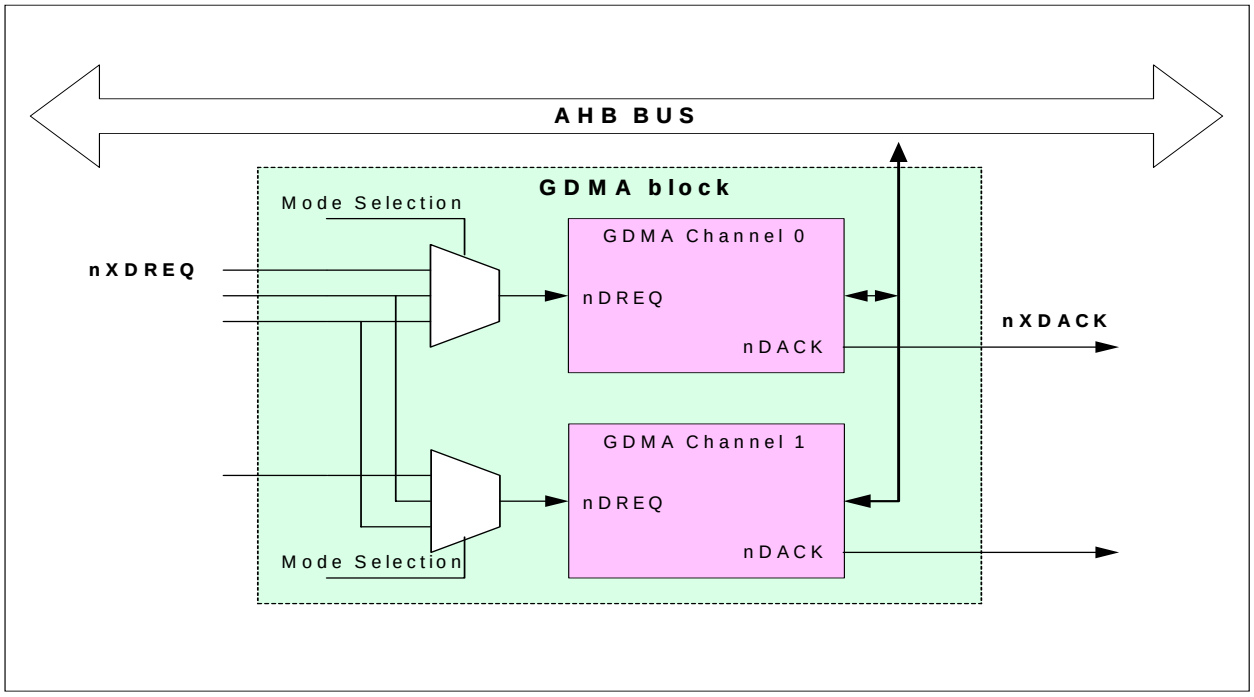
The data transfer can be started after write the control register or receive an external DMA request (nXDREQ). The GDMA will try to finish the data transfer according to the transfer mode, source address, destination address and transfer count. The device driver can recognize the completion of a GDMA operation by polling control register or when it receives a GDMA interrupt.

The NUC740 GDMA controller implements many flexible features to support the data transfers. It can increment or decrement source or destination address during the data transfer, and conduct with 8-bit (byte), 16-bit (half-word), or 32-bit (word) size data transfers. The source or destination address of the GDMA can be fixed also. Furthermore, the GDMA supports 4-data burst mode to boost performance and supports demand mode to speed up external GDMA operations.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

7.2 Block Diagram

Figure 7-1 GDMA Block Diagram



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

7.3 Registers

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
GDMA_CTL0	0xFFF0.4000	R/W	Channel 0 Control Register	0x0000.0000
GDMA_SRCB0	0xFFF0.4004	R/W	Channel 0 Source Base Address Register	0x0000.0000
GDMA_DSTB0	0xFFF0.4008	R/W	Channel 0 Destination Base Address Register	0x0000.0000
GDMA_TCNT0	0xFFF0.400C	R/W	Channel 0 Transfer Count Register	0x0000.0000
GDMA_CSRC0	0xFFF0.4010	R	Channel 0 Current Source Address Register	0x0000.0000
GDMA_CDST0	0xFFF0.4014	R	Channel 0 Current Destination Address Register	0x0000.0000
GDMA_CTCNT0	0xFFF0.4018	R	Channel 0 Current Transfer Count Register	0x0000.0000
GDMA_CTL1	0xFFF0.4020	R/W	Channel 1 Control Register	0x0000.0000
GDMA_SRCB1	0xFFF0.4024	R/W	Channel 1 Source Base Address Register	0x0000.0000
GDMA_DSTB1	0xFFF0.4028	R/W	Channel 1 Destination Base Address Register	0x0000.0000
GDMA_TCNT1	0xFFF0.402C	R/W	Channel 1 Transfer Count Register	0x0000.0000
GDMA_CSRC1	0xFFF0.4030	R	Channel 1 Current Source Address Register	0x0000.0000
GDMA_CDST1	0xFFF0.4034	R	Channel 1 Current Destination Address Register	0x0000.0000
GDMA_CTCNT1	0xFFF0.4038	R	Channel 1 Current Transfer Count Register	0x0000.0000

7.4 Functional Descriptions

7.4.1 GDMA Configuration

Each GDMA channel has one control register, two base address registers and one transfer count register. These registers should be correctly programmed before the data transfer starts. The most important one is the control register **(GDMA_CTL)**. It is used to control the transfer behavior of the GDMA operation, such as the transfer mode, transfer count, transfer width and interrupt mask. Figure 7-2 lists the content of GDMA_CTL. The detail description of each bit-field can be found in NUC740 data sheet.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

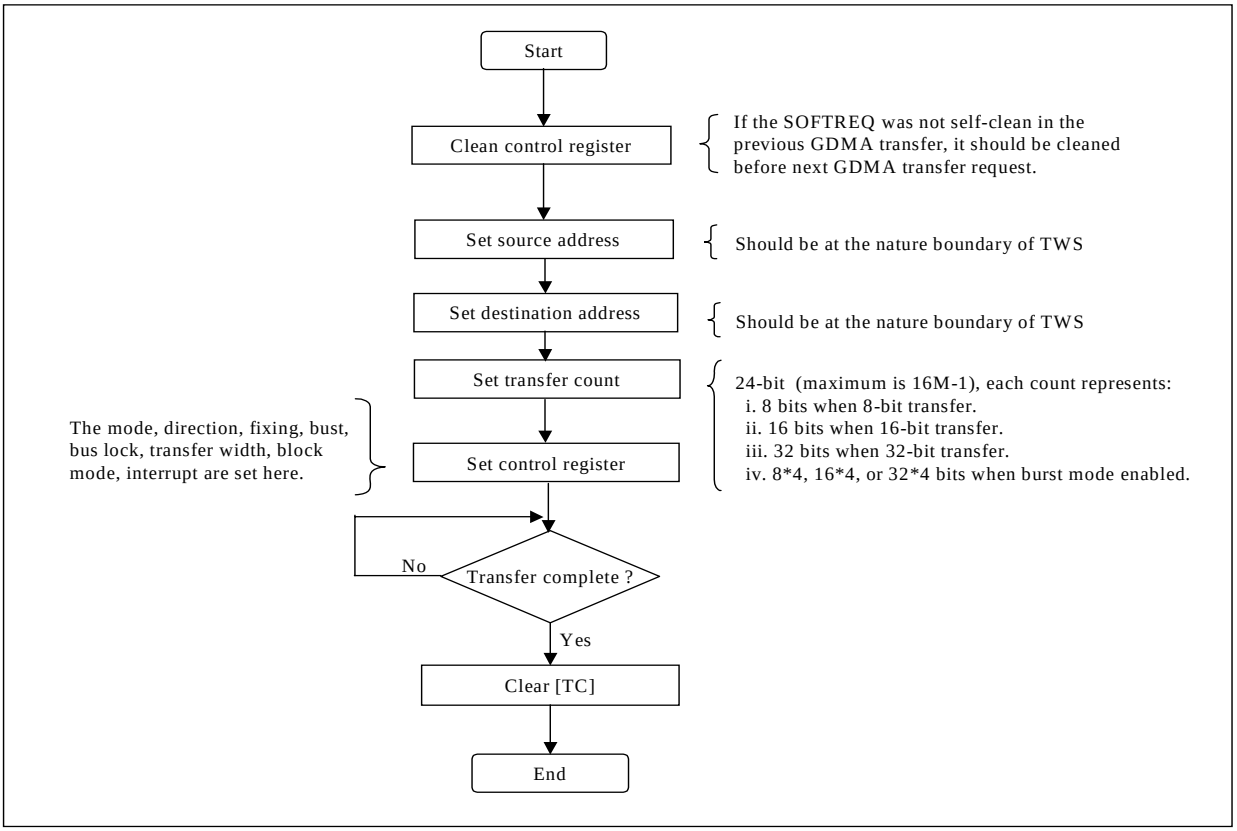
Figure 7-2 The bit-fields of the GDMA control register.

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

The source base address register **(GDMA_SRCB)** is used to set the base address of source data. The destination base address register **(GDMA_DSTB)** is used to set the starting address where the source data to be stored. The number of the GDMA transfer is set by programming the transfer count register **(GDMA_TCNT)**. The GDMA operation is continued until the transfer count register is counted down to zero. Figure 7-3 shows the programming flow for GDMA operation.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 7-3 GDMA operations



7.4.2 Transfer Count

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

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

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

Transferred bytes = [GDMA_TCNT] * Transfer width * 4 /* burst mode is enabled */

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

In case of burst mode is enabled, the transferred bytes of the above example should be 16 * 2 * 4 = 128

7.4.3 Transfer Termination

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

7.4.4 GDMA operation started by software

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

(1) Set the GDMA to software mode (GDMAMS=00b).
(2) Set the GDMA to Block Mode.
(3) After all configuration of the GDMA, set [SOFTREQ] = 1 and [GDMAEN] = 1 to start the GDMA operation.
(4) Single mode is invalid.
(5) Demand mode is invalid.

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

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

Figure 7-4 shows an example code for software GDMA transfer.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 7-4 Software GDMA Transfer

```
#define BASE 0xc0000000
#define GDMA_SRCB0              (BASE+0xFF04004)
#define GDMA_DSTB0              (BASE+0xFF04008)
#define GDMA_TCNT0              (BASE+0xFF0400C)
#define GDMA_CSRC0              (BASE+0xFF04010)
#define GDMA_CDST0              (BASE+0xFF04014)
#define GDMA_CTCNT0             (BASE+0xFF04018)


void main(void)
{
   *((volatile UINT *)GDMA_CTL0)=0x0;
   *((volatile UINT *)GDMA_SRCB0)=0xc2000000;
   *((volatile UINT *)GDMA_DSTB0)=0xc2001000;
   *((volatile UINT *)GDMA_TCNT0)=0x10;
   *((volatile UINT *)GDMA_CTL0)=0x12801;

   while(!(*((volatile UINT *)GDMA_CTL0) & 0x40000))
                ;
}
```

In this example, source base address is set as 0xC2000000 and destination base address is 0xC2001000. The transfer count is 0x10. The burst mode is not turned on in this example. Both source and destination base addresses are increment. The transfer width is 32-bit. The program waits GDMA transfer completed by polling TC flag. The first line of main routine that clears the GDMA control register is used to avoid that the SOFTREQ did not be self-cleared in the previous GDMA transfer. Once these code executed, the GDMA will copy 0x10*4 bytes data from 0xC2000000 to 0xC2001000. After the transfer completed, the current source, destination, and transfer count status can be read from current status registers. These current status registers are GDMA_CSRC, GDMA_CDST, and GDMA_CTNT respectively. However, if the AUTOIEN is 1, the current status registers will be updated to the value stored in GDMA_SRCB, GDMA_DSTB, and GDMA_TCNT. The GDMAEN did not be self-cleared if AUTOIEN is 1. Therefore, it only needs to

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

set SOFTREQ to request the GDMA transfer next time if the AUTOIEN is 1 by the same source address, destination address, and transfer count.

7.4.5 GDMA operation started by nXDREQ

The GDMA can accept the request from external device. The external device requests the GDMA transfer by asserting signal nXDREQ. When nXDREQ is used to request the GDMA transfer, it is called external nXDREQ mode. The programming method of external nXDREQ mode is the same as software mode except for the followings.

- The GDMA transfer is requested by nXDREQ pin.
- The GDMA is operated in external nXDREQ mode (GDMANS=01b).
- Single mode is valid.
- Demand mode is valid.

7.4.6 Fixed Address

Generally the GDMA continually increase or decrease the source and destination address during data transfer. The NUC740 GDMA controller provides another feature to support the fixed source/destination address to perform data transfer between system memory and external device. To do a Memory-to-I/O transfer, the bit **DAFIX** in register GDMA_CTL should be set. In case of I/O-to-Memory transfer, the bit **SAFIX** in register GDMA_CTL should be set.

7.4.7 Block Mode Transfer

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

7.4.8 Single Mode Transfer

The single mode transfer (**[SBMS]** = 0) is different to block mode. It can't be started via setting bit SOFTREQ. Besides, Single Mode Transfer requires an **nXDREQ** request for each data transfer that may be one byte, one-halftword, or one word. When receiving **nXDREQ** request, GDMA performs a single data transfer and then wait for next **nXDREQ**. After the numbers of data

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

specified by register **GDMA_TCNT** have been transferred, the GDMA set the bit **TC** and generates an interrupt if it is enabled.

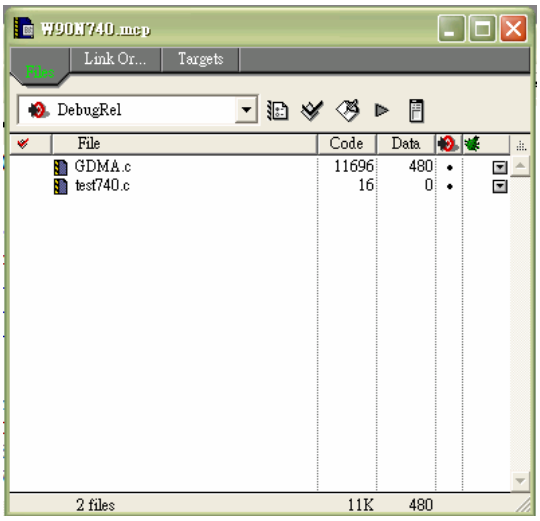
7.4.9 Demand Mode Transfer

The GDMA controller supports the demand mode feature to speed up external DMA transfer. When bit **DM** of register GDML_CTL is set to 1, GDMA controller transfers data as long as the signal nXDREQ is active. The amount of data transferred depends on how long the nXDREQ is active. When nXDREQ is active and GDMA gets the bus in Demand mode, GDMA controller holds the system bus until the nXDREQ signal becomes non-active. Therefore, the period of the active nXDREQ signal should be carefully tuned such that the entire operation does not exceed an acceptable interval (for example, in a DRAM refresh operation).

7.5 Sample Code

The NUC740 Emulation Test Code is used to test if the hardware functions of NUC740 works. The test code for GDMA is as follows. It is build by Code Warrior for ADS 1.1 and its project file is as Figure 7-5 .

Figure 7-5 The project file of the GDMA test program.



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

The project file is called NUC740.MCP. The main function of the GDMA test program is in test740.c and all GDMA test functions are in GDMA.c. The included file is NUC740.H. The source codes of the three files are as follows :

7.5.1 NUC740.H

```
#ifndef NUC740_H

#define NUC740_H

//---------------------------------------------------------------------------

#define DEBUG_INFO

//---------------------------------------------------------------------------

#define BASE          0xc0000000


typedef unsigned char BYTE;

typedef unsigned short WORD;

typedef unsigned intDWORD;

typedef int     BOOL;

typedef void (*fptr)(void);  /* function pointer */

#define FALSE 0;

#define TRUE 1;


/* io function definition */

#define outp(port,value)     *((volatile BYTE *)(port))=value
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define outpdw(port,value) *((volatile DWORD *)(port))=value
#define outpw(port,value)   outpdw(port,value)
#define inp(port)               *((volatile BYTE *)(port))
#define inpdw(port)  *((volatile DWORD *)(port))
#define inpw(port)    inpdw(port)

#ifdef DEBUG_INFO
#define msg(x) printf("\n");\
                                printf(x);
#else
#define msg(x)
#endif


/* AP Board registers */
#define AP_IRQ0_STATUS                  0x14000000
#define AP_IRQ0_RAWSTAT      0x14000004
#define AP_IRQ0_ENABLESET   0x14000008
#define         AP_IRQ0_ENABLECLR     0x1400000c

/* IRQ register bit assigments, 22 bits */
#define         EXPINT0       0x000200 /* Logic module 0 interrupt */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
/* LM registers */
#define LM_CLK0    0x0000000    /* 19 bits */
#define LM_CLK1    0x0000004    /* 19 bits */
#define LM_CLK_LOCK     0x0000008    /* 17 bits */
#define LM_LEDS    0x000000c    /* 9 bits  */
#define LM_BUTTON         0x0000010    /* 1 bit   */
#define LM_SWITCH         0x0000014    /* 8 bits  */
/* Clock control signal, Freq=48MHz*(nVCO_DIV+8)/(nREF_DIV+2)/nOUT_DIV */
                                                /* LM_CLK[18 : 16],default=110b */
#define OUT_DIV2  0x10000
#define OUT_DIV4  0x30000
#define OUT_DIV5  0x40000
#define OUT_DIV7  0x50000
#define          OUT_DIV8    0x20000
#define OUT_DIV9  0x60000        /* default */
#define OUT_DIV6  0x70000
#define OUT_DIV10          0x00000
#define REF_DIV     0x0fe00 /* LM_CLK[15 : 9],default=0111110b */
#define VCO_DIV    0x001ff /* LM_CLK[8 : 0],default=000000100b */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
/* AIC registers */
#ifdef MAIN_PROG
DWORD AIC_SCR[32]={   BASE+0x0ff82004,
                      BASE+0x0ff82008,
                      BASE+0x0ff8200c,
                      BASE+0x0ff82010,
                      BASE+0x0ff82014,
                      BASE+0x0ff82018,
                      BASE+0x0ff8201c,
                      BASE+0x0ff82020,
                      BASE+0x0ff82024,
                      BASE+0x0ff82028,
                      BASE+0x0ff8202c,
                      BASE+0x0ff82030,
                      BASE+0x0ff82034,
                      BASE+0x0ff82038,
                      BASE+0x0ff8203c,
                      BASE+0x0ff82040,
                      BASE+0x0ff82044,
                      BASE+0x0ff82048,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                                        };
#else
extern AIC_SCR[32];
#endif
#define AIC_SCR1  (BASE+0x0FF82004)
#define AIC_SCR2  (BASE+0x0FF82008)
#define AIC_SCR3  (BASE+0x0FF8200C)
#define AIC_SCR4  (BASE+0x0FF82010)
#define AIC_SCR5  (BASE+0x0FF82014)
#define AIC_SCR6  (BASE+0x0FF82018)
#define AIC_SCR7  (BASE+0x0FF8201C)
#define AIC_SCR8  (BASE+0x0FF82020)
#define AIC_SCR9  (BASE+0x0FF82024)
#define AIC_SCR10       (BASE+0x0FF82028)
#define AIC_SCR11       (BASE+0x0FF8202C)
#define AIC_SCR12       (BASE+0x0FF82030)
#define AIC_SCR13       (BASE+0x0FF82034)
#define AIC_SCR14       (BASE+0x0FF82038)
#define AIC_SCR15       (BASE+0x0FF8203C)
#define AIC_SCR16       (BASE+0x0FF82040)
#define AIC_SCR17       (BASE+0x0FF82044)
#define AIC_SCR18       (BASE+0x0FF82048)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define AIC_IRSR   (BASE+0x0ff82100)
#define AIC_IASR   (BASE+0x0ff82104)
#define AIC_ISR    (BASE+0x0ff82108)
#define AIC_IPER   (BASE+0x0ff8210c)
#define AIC_ISNR   (BASE+0x0ff82110)
#define AIC_IMR    (BASE+0x0ff82114)
#define AIC_OISR   (BASE+0x0ff82118)
#define AIC_MECR (BASE+0x0ff82120)
#define AIC_MDCR (BASE+0x0ff82124)
#define AIC_SSCR  (BASE+0x0ff82128)
#define AIC_SCCR  (BASE+0x0ff8212c)
#define AIC_EOSCR        (BASE+0x0ff82130)


/* Source type */
#define LOW_LEVEL       0x0
#define HIGH_LEVEL      0x1
#define NEG_EDGE        0x2
#define POS_EDGE        0x3
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
/* Timer registers */
#define T_TCR0      (BASE+0x0ff81000)
#define T_TCR1      (BASE+0x0ff81004)
#define T_TICR0     (BASE+0x0ff81008)
#define T_TICR1     (BASE+0x0ff8100c)
#define T_TDR0      (BASE+0x0ff81010)
#define         T_TDR1      (BASE+0x0ff81014)
#define T_TISR      (BASE+0x0ff81018)
#define T_WTCR     0x0ff8101c
/* Status flag */
#define TIF0 0x1
#define TIF1 0x2
/* Mode */
#define ONE_SHOT 0x00000000
#define PERIOD      0x08000000
#define TOGGLE      0x10000000
#define IE      0x40000000
#define         CE      0x20000000
#define WTTME       0x00000100
#define WTE         0x00000080
#define WTIE        0x00000040
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define WTIS        0x00000030
#define WTIF        0x00000008
#define WTRF        0x00000004
#define WTRE        0x00000002
#define WTR         0x00000001

/*----- EBI control register definitions -----*/
#define EBICON          (BASE+0xFF01000)        /* EBI control register */
#define ROMCON          (BASE+0xFF01004)        /* ROM/FLASH control register */
#define SDCONF0  (BASE+0xFF01008)       /* SDRAM bank 0 configuration register */
#define SDCONF1  (BASE+0xFF0100C)               /* SDRAM bank 1 configuration register */
#define SDTIME0  (BASE+0xFF01010)       /* SDRAM bank 0 timing control register */
#define SDTIME1  (BASE+0xFF01014)       /* SDRAM bank 1 timing control register */
#define EXT0CON  (BASE+0xFF01018)       /* External I/O 0 control register */
#define EXT1CON  (BASE+0xFF0101C)               /* External I/O 1 control register */
#define EXT2CON  (BASE+0xFF01020)       /* External I/O 2 control register */
#define EXT3CON  (BASE+0xFF01024)       /* External I/O 3 control register */
#define CKSKEW          (BASE+0xFF01F00)                /* clock skew control register (for testing)*/
/*----- GDMA control register -----*/
#define GDMA_CTL0       (BASE+0xFF04000)
#define GDMAERR (0x1<<20)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define AUTOIEN   (0x1<<19)
#define TC        (0x1<<18)
#define BLOCK     (0x1<<17)
#define SOFTREQ   (0x1<<16)
#define DM        (0x1<<15)
#define TWS       (0x3<<12)
#define SBMS      (0x1<<11)
#define BME       (0x1<<9)
#define SIEN      (0x1<<8)
#define SAFIX     (0x1<<7)
#define DAFIX     (0x1<<6)
#define SADIR     (0x1<<5)
#define DADIR     (0x1<<4)
#define GDMAMS    (0x3<<2)
#define GDMAEN    (0x1)
```

```
#define GDMA_SRCB0   (BASE+0xFF04004)
#define GDMA_DSTB0   (BASE+0xFF04008)
#define GDMA_TCNT0   (BASE+0xFF0400C)
#define GDMA_CSRC0   (BASE+0xFF04010)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define GDMA_CDST0      (BASE+0xFF04014)
#define GDMA_CTCNT0     (BASE+0xFF04018)
#define GDMA_CTL1       (BASE+0xFF04020)
#define GDMA_SRCB1      (BASE+0xFF04024)
#define GDMA_DSTB1      (BASE+0xFF04028)
#define GDMA_TCNT1      (BASE+0xFF0402C)
#define GDMA_CSRC1      (BASE+0xFF04030)
#define GDMA_CDST1      (BASE+0xFF04034)
#define GDMA_CTCNT1     (BASE+0xFF04038)


#define GPIO_CFG (BASE+0xFF83000)
#define GPIO_DIR   (BASE+0xFF83004)
#define GPIO_DATAOUT  (BASE+0xFF83008)
#define GPIO_DATAIN      (BASE+0xFF8300C)
#define DEBNCE_CTRL     (BASE+0xFF83010)


//-------------------------------------------------------------------------
//extern unsigned int RWMem(volatile void *addr,unsigned int data,int w,int r);
//extern int MemTest(void);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
//---------------------------------------------------------------------------

#endif
```

7.5.2 test740.c

```
#define MAIN_PROG

#include <stdio.h>


void GdmaTest(void);

int main(void)

{

        GdmaTest() ;

}
```

7.5.3 GDMA.c

```
#include "NUC740.H"
#include <stdio.h>
#define ROBUST_TEST_NUM 10
typedef unsigned int UINT;

volatile UINT IsGdmaChannel0IntOk=0;
volatile UINT IsGdmaChannel1IntOk=0;
volatile UINT          InterruptCnt=0;
volatile UINT aic_iasr=0;
volatile UINT aic_isr=0;
volatile UINT aic_oisr=0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
void GdmaInitChannel0(UINT src, UINT dest, UINT tcnt);
void GdmaInitChannel1(UINT src, UINT dest, UINT cnt);
void GdmaCtrlChannel0(UINT config);
void GdmaCtrlChannel1(UINT config);
UINT GdmaTcnt(UINT size, UINT config);
void GdmaTest(void);
void GdmaTest1(UINT width, UINT burst, UINT fix);
void GdmaTest2(UINT src, UINT dest, UINT size);
UINT InitMemory(UINT addr, UINT size);
UINT ZeroMemory(UINT addr, UINT size);
UINT CompareMemory(UINT src, UINT dest, UINT size, UINT fix,UINT width);
void InitExtIoMemory(void);
void InstallGdmaHandler(void);
void AutoInitTest(void);
void BusLockTest(void);
void SingleModeTest(void);
void AlignmentErrorFlagTest(void);
void GdmaResetValueTest(void);
void AddrFixTest(void);
void DemindModeTest(void);
void SingleModeTest(void);
void BME1ktest(void);
void GPIOResetValueTest(void);

__irq void GdmaIrqHandler(void);
__irq void GdmaIrqHandler2(void);
__irq void AicIrqHandler(void);

void GdmaInitSdram(void);
void AICTest(void);


void GdmaTest(void)
{


        /* slow down the system clock */
        *((volatile UINT *)0x1100001c)=0xa05f;
        *((volatile UINT *)0x11000004)=0x20;
        *((volatile UINT *)0x1100001c)=0x0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
InstallGdmaHandler();
GdmaInitSdram();
GdmaResetValueTest();
printf("\n32bits, no burst test...");
GdmaTest1((2<<12),0,0);
printf("\n16bits, no burst test...");
GdmaTest1((1<<12),0,0);
printf("\n8bits, no burst test...");
GdmaTest1((0<<12),0,0);
printf("\n32bits, burst test...");
GdmaTest1((2<<12),BME,0);
printf("\n16bits, burst test...");
GdmaTest1((1<<12),BME,0);
printf("\n8bits, burst test...");
GdmaTest1((0<<12),BME,0);
printf("\nSource address fixed test...");
GdmaTest1((2<<12),BME,SAFIX);
printf("\nDestination address fixed test...");
GdmaTest1((2<<12),BME,DAFIX);
printf("\nSource & destination address fixed test...");
GdmaTest1((2<<12),BME,SAFIX+DAFIX);
printf("\nSource & destination direction test...");
GdmaTest2(0xc200100C,0xc201000C,1024);
printf("\nBus lock test...");
BusLockTest();
printf("\nAlignment error flag test...");
AlignmentErrorFlagTest();
printf("\nAuto-initialze test...");
AutoInitTest();
printf("\nChecking GPIO...");
GPIOResetValueTest();
printf("\nSet GPIO_0 as output pin.");
*((volatile unsigned int *)GPIO_DIR)=0x1;
printf("\nDemand mode test...");
DemindModeTest();
printf("\nSingle mode test...");
SingleModeTest();
printf("\nBurst mode cross 1k boundary test...");
BME1ktest();
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
}

void GPIOResetValueTest(void)
{
        volatile unsigned int *p;
        unsigned int i;
        unsigned int data;
        char
RegName[5][13]={"GPIO_CFG","GPIO_DIR","GPIO_DATAOUT","GPIO_DATAIN","DEBNCE_CTR
L"};

        data=0xFFFFFFFF;

        p=(unsigned int *)GPIO_CFG;

        printf("\nReset value of GPIO control registers");
        for(i=0;i<5;i++)
        {
                printf("\n%s=0x%08x",RegName[i],*(p+i));
        }
        printf("\nWrite 0x%08x to each register...",data);
        for(i=0;i<5;i++)
        {
                *(p+i)=data;
                printf("\n%s=0x%08x",RegName[i],*(p+i));
        }
        data=0x00000000;
        printf("\nWrite 0x%08x to each register...",data);
        for(i=0;i<5;i++)
        {
                *(p+i)=data;
                printf("\n%s=0x%08x",RegName[i],*(p+i));
        }


}



void SingleModeTest(void)
{
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
UINT i;
UINT tc=0;
UINT ecnt=0;

printf("\nSingle mode test with no Burst mode....");
for(i=0;i<0x8;i++)
{
        *((volatile UINT *)0xc2000000+i)=i+1;
        *((volatile UINT *)0xc2001000+i)=0;
}
*((volatile UINT *)GDMA_CTL0)=0x0;
*((volatile UINT *)GDMA_SRCB0)=0xc2000000;
*((volatile UINT *)GDMA_DSTB0)=0xc2001000;
*((volatile UINT *)GDMA_TCNT0)=0x8;
/* Demand,non-burst,nXDREQ mode,single mode */
*((volatile UINT *)GDMA_CTL0)=0x0a005;
do{
        tc=(*((volatile UINT *)GDMA_CTL0)&0x40000)>>18;
        printf("\nTC=%x CSRCB0=0x%08x CDSTB0=0x%08x CTCNT0=0x%08x",tc,\
        *((volatile UINT *)GDMA_CSRC0),*((volatile UINT *)GDMA_CDST0),\
        *((volatile UINT *)GDMA_CTCNT0));
/* nXDREQ !!*/
*((volatile unsigned int *)GPIO_DATAOUT)=0x1;
*((volatile unsigned int *)GPIO_DATAOUT)=0x0;
}while( tc==0 );
for(i=0;i<0x8;i++)
{
        if(*((volatile UINT *)0xc2001000+i)!=i+1)
        {
                printf("\nnXDREQ data error");
                ecnt++;
        }
}

printf("\nSingle mode test with Burst mode enable....");
tc=0;
for(i=0;i<0x20;i++)
{
        *((volatile UINT *)0xc20000FC+i)=i+1;
        *((volatile UINT *)0xc20010FC+i)=0;
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        *((volatile UINT *)GDMA_CTL1)=0x0;
        *((volatile UINT *)GDMA_SRCB1)=0xc20000FC;
        *((volatile UINT *)GDMA_DSTB1)=0xc20010FC;
        *((volatile UINT *)GDMA_TCNT1)=0x8;
        /* Demand,Burst,nXDREQ mode,single mode */
        *((volatile UINT *)GDMA_CTL1)=0x0a205;
        do{
                tc=(*((volatile UINT *)GDMA_CTL1)&0x40000)>>18;
                printf("\nTC=%x CSRCB1=0x%08x CDSTB1=0x%08x CTCNT1=0x%08x",tc,\
                *((volatile UINT *)GDMA_CSRC1),*((volatile UINT *)GDMA_CDST1),\
                *((volatile UINT *)GDMA_CTCNT1));
        /* nXDREQ !!*/
        *((volatile unsigned int *)GPIO_DATAOUT)=0x1;
        *((volatile unsigned int *)GPIO_DATAOUT)=0x0;
        }while( tc==0 );
        for(i=0;i<0x20;i++)
        {
                if(*((volatile UINT *)0xc20010FC+i)!=i+1)
                {
                        printf("\nnXDREQ data error");
                        ecnt++;
                }
        }


        if( ecnt== 0 )
                printf("\nSingle mode test PASSED!");
        else
                printf("\nThere are %u failures!!!",ecnt);

}

void BME1ktest(void)
{
        UINT i;
        UINT ecnt=0;

        printf("\nBME 1k for 32bits test...");
        for(i=0;i<0x8;i++)
        {
                *((volatile UINT *)0xc20000FC+i)=i+1;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        *((volatile UINT *)0xc20010FC+i)=0;
}
*((volatile UINT *)GDMA_CTL0)=0x0;
*((volatile UINT *)GDMA_SRCB0)=0xc20000FC;
*((volatile UINT *)GDMA_DSTB0)=0xc20010FC;
*((volatile UINT *)GDMA_TCNT0)=0x2;
/* Demand,Burst,nXDREQ mode */
*((volatile UINT *)GDMA_CTL0)=0x0aa05;
/* nXDREQ !!*/
*((volatile unsigned int *)GPIO_DATAOUT)=0x1;
while( !(*((volatile UINT *)GDMA_CTL0)&0x40000) );
*((volatile unsigned int *)GPIO_DATAOUT)=0x0;
for(i=0;i<0x8;i++)
{
        if(*((volatile UINT *)0xc20010FC+i)!=i+1)
        {
                printf("\nnXDREQ data error");
                ecnt++;
        }
}

printf("\nBME 1k for 16bits test...");
for(i=0;i<0x20;i++)
{
        *((volatile unsigned short *)0xc20000FC+i)=i+1;
        *((volatile unsigned short *)0xc20010FC+i)=0;
}
*((volatile UINT *)GDMA_CTL0)=0x0;
*((volatile UINT *)GDMA_SRCB0)=0xc20000FC;
*((volatile UINT *)GDMA_DSTB0)=0xc20010FC;
*((volatile UINT *)GDMA_TCNT0)=0x8;
/* Demand,Burst,nXDREQ mode,16bits */
*((volatile UINT *)GDMA_CTL0)=0x09a05;
/* nXDREQ !!*/
*((volatile unsigned int *)GPIO_DATAOUT)=0x1;
while( !(*((volatile UINT *)GDMA_CTL0)&0x40000) );
*((volatile unsigned int *)GPIO_DATAOUT)=0x0;
for(i=0;i<0x20;i++)
{
        if(*((volatile unsigned short *)0xc20010FC+i)!=i+1)
        {
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                printf("\nnXDREQ data error");
                ecnt++;
            }
        }

        printf("\nBME 1k for 8bits test...");
        for(i=0;i<0x20;i++)
        {
            *((volatile unsigned char *)(0xc20000FE)+i)=i+1;
            *((volatile unsigned char *)(0xc20010FE)+i)=0;
        }
        *((volatile UINT *)GDMA_CTL0)=0x0;
        *((volatile UINT *)GDMA_SRCB0)=0xc20000FE;
        *((volatile UINT *)GDMA_DSTB0)=0xc20010FE;
        *((volatile UINT *)GDMA_TCNT0)=0x8;
        /* Demand,Burst,nXDREQ mode,16bits */
        *((volatile UINT *)GDMA_CTL0)=0x08a05;
        /* nXDREQ !! */
        *((volatile unsigned int *)GPIO_DATAOUT)=0x1;
        while( !(*((volatile UINT *)GDMA_CTL0)&0x40000) );
        *((volatile unsigned int *)GPIO_DATAOUT)=0x0;
        for(i=0;i<0x20;i++)
        {
            if(*((volatile unsigned char *)(0xc20010FE)+i)!=i+1)
            {
                printf("\nnXDREQ data error");
                ecnt++;
            }
        }

        if( ecnt== 0 )
            printf("\nBME 1k test PASSED!");
        else
            printf("\nThere are %u failures!!!",ecnt);
}

void DemindModeTest(void)
{
        UINT i,ecnt=0;

        for(i=0;i<0x10;i++)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
        *((volatile UINT *)0xc2000000+i)=i+1;
        *((volatile UINT *)0xc2001000+i)=0;
}
*((volatile UINT *)GDMA_CTL0)=0x0;
*((volatile UINT *)GDMA_SRCB0)=0xc2000000;
*((volatile UINT *)GDMA_DSTB0)=0xc2001000;
*((volatile UINT *)GDMA_TCNT0)=0x10;
/* no-Demand,non-burst,nXDREQ mode */
*((volatile UINT *)GDMA_CTL0)=0x0a805;
/* nXDREQ !!*/
*((volatile unsigned int *)GPIO_DATAOUT)=0x1;
while( !(*((volatile UINT *)GDMA_CTL0)&0x40000) );
*((volatile unsigned int *)GPIO_DATAOUT)=0x0;
for(i=0;i<0x10;i++)
{
        if(*((volatile UINT *)0xc2001000+i)!=i+1)
        {
                printf("\nnXDREQ data error");
                ecnt++;
        }
}

for(i=0;i<0x10;i++)
{
        *((volatile UINT *)0xc2000000+i)=i+1;
        *((volatile UINT *)0xc2001000+i)=0;
}
*((volatile UINT *)GDMA_CTL1)=0x0;
*((volatile UINT *)GDMA_SRCB1)=0xc2000000;
*((volatile UINT *)GDMA_DSTB1)=0xc2001000;
*((volatile UINT *)GDMA_TCNT1)=0x10;
/* Demand,non-burst,nXDREQ mode */
*((volatile UINT *)GDMA_CTL1)=0x0a805;
/* nXDREQ !!*/
*((volatile unsigned int *)GPIO_DATAOUT)=0x1;
while( !(*((volatile UINT *)GDMA_CTL1)&0x40000) );
*((volatile unsigned int *)GPIO_DATAOUT)=0x0;
for(i=0;i<0x10;i++)
{
        if(*((volatile UINT *)0xc2001000+i)!=i+1)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        {
                printf("\nnXDREQ data error");
                ecnt++;
        }
}


for(i=0;i<0x40;i++)
{
        *((volatile UINT *)0xc2000000+i)=i+1;
        *((volatile UINT *)0xc2001000+i)=0;
}
*((volatile UINT *)GDMA_CTL0)=0x0;
*((volatile UINT *)GDMA_SRCB0)=0xc2000000;
*((volatile UINT *)GDMA_DSTB0)=0xc2001000;
*((volatile UINT *)GDMA_TCNT0)=0x10;
/* Demand,Burst,nXDREQ mode */
*((volatile UINT *)GDMA_CTL0)=0x0aa05;
/* nXDREQ !!*/
*((volatile unsigned int *)GPIO_DATAOUT)=0x1;
while( !(*((volatile UINT *)GDMA_CTL0)&0x40000) );
*((volatile unsigned int *)GPIO_DATAOUT)=0x0;
for(i=0;i<0x40;i++)
{
        if(*((volatile UINT *)0xc2001000+i)!=i+1)
        {
                printf("\nnXDREQ data error");
                ecnt++;
        }
}

for(i=0;i<0x40;i++)
{
        *((volatile UINT *)0xc2000000+i)=i+1;
        *((volatile UINT *)0xc2001000+i)=0;
}
*((volatile UINT *)GDMA_CTL1)=0x0;
*((volatile UINT *)GDMA_SRCB1)=0xc2000000;
*((volatile UINT *)GDMA_DSTB1)=0xc2001000;
*((volatile UINT *)GDMA_TCNT1)=0x10;
/* Demand,Burst,nXDREQ mode */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        *((volatile UINT *)GDMA_CTL1)=0x0aa05;
        /* nXDREQ !!*/
        *((volatile unsigned int *)GPIO_DATAOUT)=0x1;
        while( !(*((volatile UINT *)GDMA_CTL1)&0x40000) );
        *((volatile unsigned int *)GPIO_DATAOUT)=0x0;
        for(i=0;i<0x40;i++)
        {
                if(*((volatile UINT *)0xc2001000+i)!=i+1)
                {
                        printf("\nnXDREQ data error");
                        ecnt++;
                }
        }

        if( ecnt== 0 )
                printf("\nDemand mode test PASSED!");
        else
                printf("\nThere are %u failures!!!",ecnt);

}



void AddrFixTest(void)
{
        unsigned int src,dest,n,i;


        dest=src+0x1000;
        n=16;
        /* sourec address fixed test */
        for(i=0;i<n;i++)
        {
                *((volatile unsigned int *)src+i)=i+1;
                *((volatile unsigned int *)dest+i)=0;
        }
        *((volatile UINT *)GDMA_SRCB0)=src;
        *((volatile UINT *)GDMA_DSTB0)=dest;
        *((volatile UINT *)GDMA_TCNT0)=n;
        *((volatile UINT *)GDMA_CTL0)=0x12801;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
}


void GdmaResetValueTest(void)
{
        volatile unsigned int *p;
        unsigned int i;
        unsigned int data;
        char
RegName[15][12]={"GDMA_CTL0","GDMA_SRCB0","GDMA_DSTB0","GDMA_TCNT0",
        "GDMA_CSRC0","GDMA_CDST0","GDMA_CTCNT0","N/A","GDMA_CTL1","GDMA_SRCB
1",
        "GDMA_DSTB1","GDMA_TCNT1","GDMA_CSRC1","GDMA_CDST1","GDMA_CTCNT1"};

        data=0xFFFFFFFF;

        p=(unsigned int *)GDMA_CTL0;

        printf("\nReset value of GDMA control registers");
        for(i=0;i<15;i++)
        {
                printf("\n%s=0x%08x",RegName[i],*(p+i));
        }
        printf("\nWrite 0x%08x to each register...",data);
        for(i=0;i<15;i++)
        {
                *(p+i)=data;
                printf("\n%s=0x%08x",RegName[i],*(p+i));
        }

}


void AICTest(void)
{

        *((volatile unsigned int *)0x38)=(unsigned int)AicIrqHandler;
        /* High-level sensitive, priority 2 */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        *((volatile unsigned int *)AIC_SCR17)=0x00000041;
        /* High-level sensitive, priority 1 */
        *((volatile unsigned int *)AIC_SCR18)=0x00000041;
/* enable AP board interrupt */
        *((volatile unsigned int *)AIC_MDCR)=0x0007FFFE;
        *((volatile unsigned int *)AIC_MECR)=0x00060000;
*(volatile unsigned int *)0x14000008 = 0x200;                   /* IRQ LM 0 */
/* enable CPSR I bit */
__asm
{
    MRS    r9, CPSR
    BIC    r9, r9,0x80
    MSR    CPSR_c, r9
}


        IsGdmaChannel0IntOk=0;
        IsGdmaChannel1IntOk=0;

        *((volatile UINT *)AIC_MDCR)=0x7FFFE;
        *((volatile UINT *)AIC_MECR)=0x60000;
        *((volatile UINT *)AIC_SCCR)=0x7FFFE;
        *((volatile UINT *)AIC_SCR17)=0xc1;
        *((volatile UINT *)AIC_SCR18)=0xc1;
        printf("\nEnable source17...");
        *((volatile UINT *)AIC_SSCR)=0x20000;
        while( !(*((volatile UINT *)AIC_OISR)&0x2) )
        {
                if( IsGdmaChannel0IntOk || IsGdmaChannel1IntOk )break;
        }

        if( IsGdmaChannel0IntOk!=0 )
        {
                printf("\n Int source 17 ok");
        }
        else
        {
                printf("\n Int source 17 failed");
        }
        printf("\nAIC_IASR=0x%08x",aic_iasr);
        printf("\nAIC_ISR =0x%08x",aic_isr);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
printf("\nAIC_OISR=0x%08x",aic_oisr);

IsGdmaChannel0IntOk=0;
IsGdmaChannel1IntOk=0;
*((volatile UINT *)AIC_SCCR)=0x7FFFE;
printf("\nEnable source18...");
*((volatile UINT *)AIC_SSCR)=0x40000;
while( !(*((volatile UINT *)AIC_OISR)&0x2) )
{
        if( IsGdmaChannel0IntOk || IsGdmaChannel1IntOk )break;
}
if( IsGdmaChannel1IntOk )
{
        printf("\n Int source 18 ok");
}
else
{
        printf("\n Int source 18 failed");
}
printf("\nAIC_IASR=0x%08x",aic_iasr);
printf("\nAIC_ISR =0x%08x",aic_isr);
printf("\nAIC_OISR=0x%08x",aic_oisr);
*((volatile UINT *)AIC_SCCR)=0x7FFFE;


printf("\nGDMA Channel 0 test ... ");
IsGdmaChannel0IntOk=0;
IsGdmaChannel1IntOk=0;
*((volatile UINT *)AIC_MDCR)=0x7FFFE;
*((volatile UINT *)AIC_MECR)=0x60000;
*((volatile UINT *)AIC_SCCR)=0x7FFFE;
*((volatile UINT *)AIC_SCR17)=0x41;
*((volatile UINT *)AIC_SCR18)=0x41;


*((UINT *)0xc1000000)=0x12345678;
*((UINT *)0xc1000004)=0x12345678*2;
*((UINT *)0xc1000008)=0x12345678*3;
*((UINT *)0xc100000c)=0x12345678*4;
*((UINT *)0xc1000010)=0x12345678*5;
*((UINT *)0xc1000014)=0x12345678*6;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        *((volatile UINT *)GDMA_SRCB0)=0xc1000000;
        *((volatile UINT *)GDMA_DSTB0)=0xc1002000;
        *((volatile UINT *)GDMA_TCNT0)=0x6;
        *((volatile UINT *)GDMA_CTL0)=0x12901;
        while( !(*((volatile UINT *)AIC_OISR)&0x2) )
        {
                if( IsGdmaChannel0IntOk || IsGdmaChannel1IntOk )break;
                if( (*((volatile UINT *)AIC_SCR17)!=0x41) || (*((volatile UINT *)AIC_SCR18)!=0x41) )
                {
                        printf("\nError :  AIC_SCR17=0x%x  AIC_SCR18=0x%x",*((volatile UINT
*)AIC_SCR17),*((volatile UINT *)AIC_SCR18));
                        //*((volatile UINT *)AIC_SCR17)=0x41;
                        //*((volatile UINT *)AIC_SCR18)=0x41;
                        break;
                }
        }
        if( IsGdmaChannel0IntOk )
        {
                printf("\n Int source 17 ok");
        }
        else
        {
                printf("\n Int source 17 failed");
        }
        printf("\nAIC_IASR=0x%08x",aic_iasr);
        printf("\nAIC_ISR =0x%08x",aic_isr);
        printf("\nAIC_OISR=0x%08x",aic_oisr);



        printf("\nGDMA Channel 1 test ... ");
        IsGdmaChannel0IntOk=0;
        IsGdmaChannel1IntOk=0;
        *((volatile UINT *)GDMA_SRCB1)=0xc1000000;
        *((volatile UINT *)GDMA_DSTB1)=0xc1002000;
        *((volatile UINT *)GDMA_TCNT1)=0x6;
        *((volatile UINT *)GDMA_CTL1)=0x12901;
        while( !(*((volatile UINT *)AIC_OISR)&0x2) )
        {
                if( IsGdmaChannel0IntOk || IsGdmaChannel1IntOk )break;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                if( (*((volatile UINT *)AIC_SCR17)!=0x41) || (*((volatile UINT *)AIC_SCR18)!=0x41) )
                {
                        printf("\nError :  AIC_SCR17=0x%x AIC_SCR18=0x%x",*((volatile UINT
*)AIC_SCR17),*((volatile UINT *)AIC_SCR18));
                        //*((volatile UINT *)AIC_SCR17)=0x41;
                        //*((volatile UINT *)AIC_SCR18)=0x41;
                        break;
                }
        }
        if( IsGdmaChannel1IntOk )
        {
                printf("\n Int source 18 ok");
        }
        else
        {
                printf("\n Int source 18 failed");
        }
        printf("\nAIC_IASR=0x%08x",aic_iasr);
        printf("\nAIC_ISR =0x%08x",aic_isr);
        printf("\nAIC_OISR=0x%08x",aic_oisr);

}


void GdmaInitChannel0(UINT src, UINT dest, UINT tcnt)
{
        *((volatile UINT *)GDMA_SRCB0)=src;
        *((volatile UINT *)GDMA_DSTB0)=dest;
        *((volatile UINT *)GDMA_TCNT0)=tcnt;
}

void GdmaInitChannel1(UINT src, UINT dest, UINT tcnt)
{
        *((volatile UINT *)GDMA_SRCB1)=src;
        *((volatile UINT *)GDMA_DSTB1)=dest;
        *((volatile UINT *)GDMA_TCNT1)=tcnt;
}

void GdmaCtrlChannel0(UINT config)
{
        *((volatile UINT *)GDMA_CTL0)=config;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
}

void GdmaCtrlChannel1(UINT config)
{
        *((volatile UINT *)GDMA_CTL1)=config;
}

UINT GdmaTcnt(UINT size, UINT config)
{
        UINT tcnt;
        switch( ((config&TWS)>>12) )
        {
                case 0 :
                        tcnt=size;
                        break;
                case 1 :
                        tcnt=size>>1;
                        break;
                case 2 :
                        tcnt=size>>2;
                        break;
                default :
                        tcnt=size;
                        break;
        }
        if( (config&BME) )
                tcnt=tcnt>>2;
        return tcnt;
}


void RobustTest(void)
{
        IsGdmaChannel0IntOk=0;
        IsGdmaChannel1IntOk=0;
        InstallGdmaHandler();
        *((volatile UINT *)0x38)=(UINT)GdmaIrqHandler2;
        GdmaTest1((2<<12),BME,0);
        printf("\nChannel 0 Int count=%d",IsGdmaChannel0IntOk);
        printf("\nChannel 1 Int count=%d",IsGdmaChannel0IntOk);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
}



/*
GdmaTest1 :
 1. Enable GDMA 0/1 at the same time.
 2. Wait for interrupt.
 3. Use EXTIO SRAM.
 4. Base function test
 5. Cross 1k AMBA boundary
*/
void GdmaTest1(UINT width, UINT burst, UINT fix)
{
        UINT src;
        UINT dest;
        UINT size;
        UINT tcnt;
        UINT config;
        UINT errCnt;

        IsGdmaChannel0IntOk=0;
        IsGdmaChannel1IntOk=0;
        if( *((volatile UINT *)GDMA_CTL0)&(GDMAEN+SOFTREQ) )
        {
                printf("\nGDMAEN 0 is not clear automatically.");
                printf("\nClean it...");
                *((volatile UINT *)GDMA_CTL0)&=(~(GDMAEN+SOFTREQ));
        }
        if( *((volatile UINT *)GDMA_CTL1)&(GDMAEN+SOFTREQ) )
        {
                printf("\nGDMAEN 1 is not clear automatically.");
                printf("\nClean it...");
                *((volatile UINT *)GDMA_CTL1)&=(~(GDMAEN+SOFTREQ));
        }

        src=0xC2000FFC;
        dest=0xC2002FFC;
        size=1024;
        IsGdmaChannel0IntOk=0;
        IsGdmaChannel1IntOk=0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        if(InitMemory(src,size+16))
                printf("\nCan't initialize memory!");
        if(ZeroMemory(dest,size+16))
                printf("\nCan't initialize memory to zero!");
        /* burst, 32bits, int enable */
        config=SOFTREQ+width+SBMS+SIEN+GDMAEN+burst+fix;
        tcnt=GdmaTcnt(size,config);
        GdmaInitChannel0(src,dest,tcnt);
        GdmaInitChannel1(src,dest,tcnt);
        GdmaCtrlChannel0(config);
        GdmaCtrlChannel1(config);
        do
        {
                if( IsGdmaChannel0IntOk || IsGdmaChannel1IntOk )
                        break;
                __asm{

                        MOV         r5,GDMA_CTL0
                        LDMIA       r5,{r6-r12}
                }

        }while( !(*((volatile unsigned int *)AIC_OISR)&0x2) );

        while( !IsGdmaChannel0IntOk );
        printf("\nChannel 0 Interrupt Ok!");
        while( !IsGdmaChannel1IntOk );
        printf("\nChannel 1 Interrupt Ok!");

        errCnt=CompareMemory(src,dest,size,fix,width);
        if(errCnt!=0)
                printf("\nThere are %u failures!!!",errCnt);
        else
                printf("\nMemory copy OK!!");

        if( fix&DAFIX )
        {
                if( (*((UINT *)(dest+size))+*((UINT *)(dest+size+4))+*((UINT *)(dest+size+8))+*((UINT
*)(dest+size+12)))!=0 )
                        printf("\nDestination fix may be wrong...!!");
        }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        printf("\nSRC0= 0x%08x DST0= 0x%08x TCNT0= 0x%08x",*((volatile UINT
*)GDMA_SRCB0),*((volatile UINT *)GDMA_DSTB0),*((volatile UINT *)GDMA_TCNT0));
        printf("\nCSRC0=0x%08x CDST0=0x%08x CTCNT0=0x%08x",*((volatile UINT
*)GDMA_CSRC0),*((volatile UINT *)GDMA_CDST0),*((volatile UINT *)GDMA_CTCNT0));
        printf("\nSRC1= 0x%08x DST1= 0x%08x TCNT1= 0x%08x",*((volatile UINT
*)GDMA_SRCB1),*((volatile UINT *)GDMA_DSTB1),*((volatile UINT *)GDMA_TCNT1));
        printf("\nCSRC1=0x%08x CDST1=0x%08x CTCNT1=0x%08x",*((volatile UINT
*)GDMA_CSRC1),*((volatile UINT *)GDMA_CDST1),*((volatile UINT *)GDMA_CTCNT1));
}


void GdmaTest2(UINT src, UINT dest, UINT size)
{
        UINT config,tcnt;
        UINT i,n;
        UINT ecnt=0;

        n=size>>2;
        for(i=0;i<n;i++)
        {
                *((UINT *)src+i)=0x01010101*i;
        }

        config=SOFTREQ+BME+SBMS+GDMAEN+(2<<12)+DADIR;
        tcnt=GdmaTcnt(size,config);
        GdmaInitChannel0(src,dest,tcnt);
        GdmaCtrlChannel0(config);

        while( !(*((volatile UINT *)GDMA_CTL0)&TC) );

        for(i=0;i<n;i++)
        {
                if( *((UINT *)src+i)!=*((UINT *)dest-i) )
                {
                        printf("\nError :  SRC addr=0x%08x data=0x%08x DEST addr=0x%08x
data=0x%08x",src+i*4,*((UINT *)src+i),dest-i*4,*((UINT *)dest-i));
                        ecnt++;
                }

        }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        printf("\nSRC0= 0x%08x DST0= 0x%08x TCNT0= 0x%08x",*((volatile UINT
*)GDMA_SRCB0),*((volatile UINT *)GDMA_DSTB0),*((volatile UINT *)GDMA_TCNT0));
        printf("\nCSRC0=0x%08x CDST0=0x%08x CTCNT0=0x%08x",*((volatile UINT
*)GDMA_CSRC0),*((volatile UINT *)GDMA_CDST0),*((volatile UINT *)GDMA_CTCNT0));



        config=SOFTREQ+BME+SBMS+GDMAEN+(2<<12)+SADIR;
        tcnt=GdmaTcnt(size,config);
        GdmaInitChannel1(src,dest,tcnt);
        GdmaCtrlChannel1(config);

        while( !(*((volatile UINT *)GDMA_CTL1)&TC) );

        for(i=0;i<n;i++)
        {
                if( *((UINT *)src-i)!=*((UINT *)dest+i) )
                {
                        printf("\nError :  SRC addr=0x%08x data=0x%08x DEST addr=0x%08x
data=0x%08x",src-i*4,*((UINT *)src-i),dest+i*4,*((UINT *)dest+i));
                        ecnt++;
                }

        }
        printf("\nSRC1= 0x%08x DST1= 0x%08x TCNT1= 0x%08x",*((volatile UINT
*)GDMA_SRCB1),*((volatile UINT *)GDMA_DSTB1),*((volatile UINT *)GDMA_TCNT1));
        printf("\nCSRC1=0x%08x CDST1=0x%08x CTCNT1=0x%08x",*((volatile UINT
*)GDMA_CSRC1),*((volatile UINT *)GDMA_CDST1),*((volatile UINT *)GDMA_CTCNT1));

        if( ecnt== 0 )
                printf("\nSource & destination direction test PASSED!");
        else
                printf("\nThere were %u failures!!",ecnt);

}


void AutoInitTest(void)
{
        UINT data,ecnt=0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
*((volatile UINT *)0xCFF04040)=0x01;


/* Test the count value */
*((volatile UINT *)GDMA_CTL0)=0x0;
*((volatile UINT *)GDMA_SRCB0)=0xc2000000;
*((volatile UINT *)GDMA_DSTB0)=0xc2001000;
*((volatile UINT *)GDMA_TCNT0)=0x10;
*((volatile UINT *)GDMA_CTL0)=0x12801;
while( !(*((volatile UINT *)GDMA_CTL0)&0x40000) );
data=*((volatile UINT *)GDMA_CSRC0);
printf("\nGDMA_CSRC0=0x%08x(should be : 0x%08x)",data,0xc2000000+0x10*4);
if( data!=(0xc2000000+0x10*4) ){ecnt++;}
data=*((volatile UINT *)GDMA_CDST0);
printf("\nGDMA_CDST0=0x%08x(should be : 0x%08x)",data,0xc2001000+0x10*4);
if( data!=(0xc2001000+0x10*4) ){ecnt++;}
data=*((volatile UINT *)GDMA_CTCNT0);
printf("\nGDMA_CTCNT0=0x%08x(should be : 0x%08x)",data,0);
if( data!=0 ){ecnt++;}


/* Test the count value */
*((volatile UINT *)GDMA_CTL1)=0x0;
*((volatile UINT *)GDMA_SRCB1)=0xc2000000;
*((volatile UINT *)GDMA_DSTB1)=0xc2001000;
*((volatile UINT *)GDMA_TCNT1)=0x10;
*((volatile UINT *)GDMA_CTL1)=0x12a01;
while( !(*((volatile UINT *)GDMA_CTL1)&0x40000) );
data=*((volatile UINT *)GDMA_CSRC1);
printf("\nGDMA_CSRC1=0x%08x(should be : 0x%08x)",data,0xc2000000+0x10*4*4);
if( data!=(0xc2000000+0x10*4*4) ){ecnt++;}
data=*((volatile UINT *)GDMA_CDST1);
printf("\nGDMA_CDST1=0x%08x(should be : 0x%08x)",data,0xc2001000+0x10*4*4);
if( data!=(0xc2001000+0x10*4*4) ){ecnt++;}
data=*((volatile UINT *)GDMA_CTCNT1);
printf("\nGDMA_CTCNT1=0x%08x(should be : 0x%08x)",data,0);
if( data!=0 ){ecnt++;}


/* Test the auto-initial value */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
*((volatile UINT *)GDMA_SRCB0)=0xc2000000;
*((volatile UINT *)GDMA_DSTB0)=0xc2001000;
*((volatile UINT *)GDMA_TCNT0)=0x10;
*((volatile UINT *)GDMA_CTL0)=0x92a01;
do{

        __asm{

                MOV         r5,GDMA_CTL0
                LDMIA       r5,{r6-r12}

        }

}while( !(*((volatile UINT *)GDMA_CTL0)&0x40000) );
data=*((volatile UINT *)GDMA_CSRC0);
printf("\nGDMA_CSRC0=0x%08x(should be : 0x%08x)",data,0xc2000000);
if( data!=0xc2000000 ){ecnt++;}
data=*((volatile UINT *)GDMA_CDST0);
printf("\nGDMA_CDST0=0x%08x(should be : 0x%08x)",data,0xc2001000);
if( data!=0xc2001000 ){ecnt++;}
data=*((volatile UINT *)GDMA_CTCNT0);
printf("\nGDMA_CTCNT0=0x%08x(should be : 0x%08x)",data,0x1000);
if( data!=0x10 ){ecnt++;}


*((volatile UINT *)GDMA_SRCB1)=0xc2000000;
*((volatile UINT *)GDMA_DSTB1)=0xc2001000;
*((volatile UINT *)GDMA_TCNT1)=0x10;
*((volatile UINT *)GDMA_CTL1)=0x92a01;
do{

        __asm{

                MOV         r7,GDMA_CSRC1
                LDMIA       r7,{r8-r10}
        }


}while( !(*((volatile UINT *)GDMA_CTL1)&0x40000) );
data=*((volatile UINT *)GDMA_CSRC1);
printf("\nGDMA_CSRC1=0x%08x(should be : 0x%08x)",data,0xc2000000);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        if( data!=0xc2000000 ){ecnt++;}
        data=*((volatile UINT *)GDMA_CDST1);
        printf("\nGDMA_CDST1=0x%08x(should be : 0x%08x)",data,0xc2001000);
        if( data!=0xc2001000 ){ecnt++;}
        data=*((volatile UINT *)GDMA_CTCNT1);
        printf("\nGDMA_CTCNT1=0x%08x(should be : 0x%08x)",data,0x10);
        if( data!=0x10 ){ecnt++;}


        if( ecnt== 0 )
                printf("\nSource & destination direction test PASSED!");
        else
                printf("\nThere were %u failures!!",ecnt);


}


void BusLockTest(void)
{
        UINT ecnt=0;
        UINT config;
        volatile UINT i;

        for(i=0;i<0x10000;i+=4)
        {
                *((volatile UINT *)(0xc2000000+i))=0x55;
                *((volatile UINT *)(0xc2010000+i))=0x0;
        }
        *((volatile UINT *)GDMA_SRCB0)=0xc2000000;
        *((volatile UINT *)GDMA_DSTB0)=0xc2010000;
        *((volatile UINT *)GDMA_TCNT0)=0x10000;
        *((volatile UINT *)GDMA_CTL0)=0x32801;
        i=0;
        do{
                config=*((volatile UINT *)GDMA_CTL0);
                *((volatile UINT *)0xc2030000)=0xaf;
                i++;
        }while( config&0x40000 );
        printf("\nChannel 0 transfer completed. Polling count=%d",i);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
/*
        if( *((volatile UINT *)0xc2010000)== 0x55 )
        {
                printf("\nSomething wrong 1!!");
                ecnt++;
        }
*/
        for(i=0;i<0x10000;i+=4)
        {
                *((volatile UINT *)(0xc2000000+i))=0x55;
                *((volatile UINT *)(0xc2010000+i))=0x0;
        }
        *((volatile UINT *)GDMA_SRCB1)=0xc2000000;
        *((volatile UINT *)GDMA_DSTB1)=0xc2010000;
        *((volatile UINT *)GDMA_TCNT1)=0x10000;
        *((volatile UINT *)GDMA_CTL1)=0x32801;
        i=0;
        do{
                config=*((volatile UINT *)GDMA_CTL1);
                *((volatile UINT *)0xc2010000+i)=i+1;
                i++;
        }while( config&0x40000 );
        printf("\nChannel 1 transfer completed. Polling count=%d",i);
/*
        if( *((volatile UINT *)0xc2010004)!= 0x55 )
        {
                printf("\nSomething wrong 2!!");
                ecnt++;
        }
*/

        if( ecnt== 0 )
                printf("\nSource & destination direction test PASSED!");
        else
                printf("\nThere were %u failures!!",ecnt);


}
```

void _SingleModeTest(void)

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
        UINT config;
        UINT i;
        /* Test the single mode without burst */
        for(i=0;i<0x10*4;i+=4)
        {
                *((volatile UINT *)(0xc2000000+i))=i;
                *((volatile UINT *)(0xc2001000+i))=0;
        }
        *((volatile UINT *)GDMA_SRCB0)=0xc2000000;
        *((volatile UINT *)GDMA_DSTB0)=0xc2001000;
        *((volatile UINT *)GDMA_TCNT0)=0x10;
        *((volatile UINT *)GDMA_CTL0)=0x12001;
        do{
                config=*((volatile UINT *)GDMA_CTL0);
                if( config&0x40000 )
                        break;
                else
                        *((volatile UINT *)GDMA_CTL0)=0x12001;
        }while(1);


        /* Test the single mode with burst */
        for(i=0;i<0x10*4*4;i+=4)
        {
                *((volatile UINT *)(0xc2000000+i))=i;
                *((volatile UINT *)(0xc2001000+i))=0;
        }
        *((volatile UINT *)GDMA_SRCB0)=0xc2000000;
        *((volatile UINT *)GDMA_DSTB0)=0xc2001000;
        *((volatile UINT *)GDMA_TCNT0)=0x10;
        *((volatile UINT *)GDMA_CTL0)=0x12201;
        do{
                config=*((volatile UINT *)GDMA_CTL0);
                if( config&0x40000 )
                        break;
                else
                        *((volatile UINT *)GDMA_CTL0)=0x12201;
        }while(1);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
}

void AlignmentErrorFlagTest(void)
{
        UINT flag;
        UINT ecnt=0;

        *((volatile UINT *)GDMA_SRCB0)=0xc2000000;
        *((volatile UINT *)GDMA_DSTB0)=0xc2001000;
        *((volatile UINT *)GDMA_TCNT0)=0x10;
        *((volatile UINT *)GDMA_CTL0)=0x02801;
        flag=(*((volatile UINT *)GDMA_CTL0))&(0x3<<21);
        if( flag!=0 )
        {
                printf("\nSrcErrFlag=%u(0) DestErrFlag=%u(0)",(flag>>1),flag&1);
                ecnt=0;
        }
        *((volatile UINT *)GDMA_SRCB0)=0xc2000002;
        flag=(*((volatile UINT *)GDMA_CTL0)>>21)&0x3;
        if( flag!=2 )
        {
                printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(0)",(flag>>1),flag&1);
                ecnt++;
        }
        *((volatile UINT *)GDMA_DSTB0)=0xc2001002;
        flag=(*((volatile UINT *)GDMA_CTL0)>>21)&0x3;
        if( flag!=3 )
        {
                printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(1)",(flag>>1),flag&1);
                ecnt++;
        }
        *((volatile UINT *)GDMA_CTL0)=0x01801;
        flag=(*((volatile UINT *)GDMA_CTL0)>>21)&0x3;
        if( flag!=0 )
        {
                printf("\nSrcErrFlag=%u(0) DestErrFlag=%u(0)",(flag>>1),flag&1);
                ecnt++;
        }
        *((volatile UINT *)GDMA_SRCB0)=0xc2000001;
        flag=(*((volatile UINT *)GDMA_CTL0)>>21)&0x3;
        if( flag!=2 )
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
        printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(0)",(flag>>1),flag&1);
        ecnt++;
}
*((volatile UINT *)GDMA_DSTB0)=0xc2001001;
flag=(*((volatile UINT *)GDMA_CTL0)>>21)&0x3;
if( flag!=3 )
{
        printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(1)",(flag>>1),flag&1);
        ecnt++;
}
*((volatile UINT *)GDMA_CTL0)=0x00801;
flag=(*((volatile UINT *)GDMA_CTL0)>>21)&0x3;
if( flag!=0 )
{
        printf("\nSrcErrFlag=%u(0) DestErrFlag=%u(0)",(flag>>1),flag&1);
        ecnt++;
}


/* test channel 1 */
*((volatile UINT *)GDMA_SRCB1)=0xc2000000;
*((volatile UINT *)GDMA_DSTB1)=0xc2001000;
*((volatile UINT *)GDMA_TCNT1)=0x10;
*((volatile UINT *)GDMA_CTL1)=0x02801;
flag=(*((volatile UINT *)GDMA_CTL1))&(0x3<<21);
if( flag!=0 )
{
        printf("\nSrcErrFlag=%u(0) DestErrFlag=%u(0)",(flag>>1),flag&1);
        ecnt=0;
}
*((volatile UINT *)GDMA_SRCB1)=0xc2000002;
flag=(*((volatile UINT *)GDMA_CTL1)>>21)&0x3;
if( flag!=2 )
{
        printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(0)",(flag>>1),flag&1);
        ecnt++;
}
*((volatile UINT *)GDMA_DSTB1)=0xc2001002;
flag=(*((volatile UINT *)GDMA_CTL1)>>21)&0x3;
if( flag!=3 )
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    {
        printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(1)",(flag>>1),flag&1);
        ecnt++;
    }
    *((volatile UINT *)GDMA_CTL1)=0x01801;
    flag=(*((volatile UINT *)GDMA_CTL1)>>21)&0x3;
    if( flag!=0 )
    {
        printf("\nSrcErrFlag=%u(0) DestErrFlag=%u(0)",(flag>>1),flag&1);
        ecnt++;
    }
    *((volatile UINT *)GDMA_SRCB1)=0xc2000001;
    flag=(*((volatile UINT *)GDMA_CTL1)>>21)&0x3;
    if( flag!=2 )
    {
        printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(0)",(flag>>1),flag&1);
        ecnt++;
    }
    *((volatile UINT *)GDMA_DSTB1)=0xc2001001;
    flag=(*((volatile UINT *)GDMA_CTL1)>>21)&0x3;
    if( flag!=3 )
    {
        printf("\nSrcErrFlag=%u(1) DestErrFlag=%u(1)",(flag>>1),flag&1);
        ecnt++;
    }
    *((volatile UINT *)GDMA_CTL1)=0x00801;
    flag=(*((volatile UINT *)GDMA_CTL1)>>21)&0x3;
    if( flag!=0 )
    {
        printf("\nSrcErrFlag=%u(0) DestErrFlag=%u(0)",(flag>>1),flag&1);
        ecnt++;
    }



    if( ecnt == 0 )
        printf("\nAlignment error test PASSED!!");
    else
        printf("\nThere are %u failures...",ecnt);

}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
void BlockTest0(void)
{
        UINT src;
        UINT dest;
        UINT size;
        UINT tcnt;
        UINT config;
        UINT errCnt;

        IsGdmaChannel0IntOk=0;
        if( *((volatile UINT *)GDMA_CTL0)&GDMAEN )
        {
                printf("\nGDMAEN is not clear automatically.");
                printf("\nClean it...");
                *((volatile UINT *)GDMA_CTL0)&=(~GDMAEN);
        }

        src=0xC6000000;
        dest=0xC6002000;
        size=0x1024;
        if(InitMemory(src,size))
                printf("\nCan't initialize memory!");
        if(ZeroMemory(dest,size))
                printf("\nCan't initialize memory to zero!");
        /* burst, 32bits, int enable */
        config=SOFTREQ+(2<<12)+SIEN+GDMAEN+BME+BLOCK+AUTOIEN;
        tcnt=GdmaTcnt(size,config);
        GdmaInitChannel0(src,dest,tcnt);
        printf("\nWaiting for Channel 0 Interrupt...");
        GdmaCtrlChannel0(config);
        do{
                *((volatile UINT *)src)=0xFF;
        }while( !IsGdmaChannel0IntOk );

        errCnt=CompareMemory(src,dest,size,0,(2<<12));
        if(errCnt!=0)
                printf("\nThere are %d failures!!!",errCnt);

        if( *((volatile UINT *)GDMA_CSRC0)!=src )
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
            printf("\nError :  GDMA_CSRC0=0x%08x",*((volatile UINT *)GDMA_CSRC0));
        if( *((volatile UINT *)GDMA_CDST0)!=dest )
            printf("\nError :  GDMA_CDST0=0x%08x",*((volatile UINT *)GDMA_CDST0));
        if( *((volatile UINT *)GDMA_CTCNT0)!=tcnt )
            printf("\nError :  GDMA_CTCNT0=0x%08x",*((volatile UINT *)GDMA_CTCNT0));


        if( *((volatile UINT *)src)!=0xFF )
            printf("\nChannel 0 BLOCK mode test ok!!");
        else
            printf("\nChannel 0 BLOCK mode test fail!!");

}

void BlockTest1(void)
{
        UINT src;
        UINT dest;
        UINT size;
        UINT tcnt;
        UINT config;
        UINT errCnt;

        IsGdmaChannel1IntOk=0;
        if( *((volatile UINT *)GDMA_CTL1)&GDMAEN )
        {
            printf("\nGDMAEN is not clear automatically.");
            printf("\nClean it...");
            *((volatile UINT *)GDMA_CTL1)&=(~GDMAEN);
        }

        src=0xC6000000;
        dest=0xC6002000;
        size=0x1024;
        if(InitMemory(src,size))
            printf("\nCan't initialize memory!");
        if(ZeroMemory(dest,size))
            printf("\nCan't initialize memory to zero!");
        /* burst, 32bits, int enable */
        config=SOFTREQ+(2<<12)+SIEN+GDMAEN+BME+BLOCK+AUTOIEN;
        tcnt=GdmaTcnt(size,config);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        GdmaInitChannel1(src,dest,tcnt);
        printf("\nWaiting for Channel 1 Interrupt...");
        GdmaCtrlChannel1(config);
        do{
                *((volatile UINT *)src)=0xFF;
        }while( !IsGdmaChannel1IntOk );

        errCnt=CompareMemory(src,dest,size,0,(2<<12));
        if(errCnt!=0)
                printf("\nThere are %d failures!!!",errCnt);

        if( *((volatile UINT *)GDMA_CSRC1)!=src )
                printf("\nError :  GDMA_CSRC1=0x%08x",*((volatile UINT *)GDMA_CSRC1));
        if( *((volatile UINT *)GDMA_CDST1)!=dest )
                printf("\nError :  GDMA_CDST1=0x%08x",*((volatile UINT *)GDMA_CDST1));
        if( *((volatile UINT *)GDMA_CTCNT1)!=tcnt )
                printf("\nError :  GDMA_CTCNT1=0x%08x",*((volatile UINT *)GDMA_CTCNT1));


        if( *((volatile UINT *)src)!=0xFF )
                printf("\nChannel 1 BLOCK mode test ok!!");
        else
                printf("\nChannel 1 BLOCK mode test fail!!");

}



UINT CompareMemory(UINT src, UINT dest, UINT size, UINT fix, UINT width)
{
        UINT i;
        UINT cnt;
        UINT mod;
        unsigned char srcData,destData;

        mod=(1<<(width>>12));
        cnt=0;
        for(i=0;i<size;i++)
        {
                if( fix&SAFIX )
                        srcData=*((volatile unsigned char *)(src+i%mod));
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
            else
                srcData=*((volatile unsigned char *)(src+i));
            if( fix&DAFIX )
            {
              if( fix&SAFIX)
              {
                switch( (width>>12) )
                {
                    case 0 :
                        srcData=*((unsigned char *)(src));
                        destData=*((unsigned char *)dest);
                        if( srcData==destData )
                            return 0;
                        else
                        {
                            printf("\n* src : 0x%08x 0x%02x  dest : 0x%08x
0x%02x",src+size-1,srcData,dest,destData);
                            return 1;
                        }
                        break;
                    case 1 :
                        if( *((unsigned short *)src)==*((unsigned short *)dest) )
                            return 0;
                        else
                        {
                            printf("\n* src : 0x%08x 0x%04x  dest : 0x%08x
0x%04x",src+size-1,*((unsigned short *)(src+size-2)),dest,*((unsigned short *)dest));
                            return 1;
                        }
                        break;
                    case 2 :
                        if( *((unsigned int *)src)==*((unsigned int *)dest) )
                            return 0;
                        else
                        {
                            printf("\n* src : 0x%08x 0x%08x  dest : 0x%08x
0x%08x",src+size-1,*((unsigned int *)(src+size-2)),dest,*((unsigned int *)dest));
                            return 1;
                        }
                        break;
                    default :
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                        printf("\nSome error~!!!!!!!!!!");
                        break;
                }

        }
        else
        {
                switch( (width>>12) )
                {
                        case 0 :
                                srcData=*((unsigned char *)(src+size-1));
                                destData=*((unsigned char *)dest);
                                if( srcData==destData )
                                        return 0;
                                else
                                {
                                        printf("\n* src : 0x%08x 0x%02x  dest : 0x%08x
0x%02x",src+size-1,srcData,dest,destData);
                                        return 1;
                                }
                                break;
                        case 1 :
                                if( *((unsigned short *)(src+size-2))==*((unsigned short *)dest) )
                                        return 0;
                                else
                                {
                                        printf("\n* src : 0x%08x 0x%04x  dest : 0x%08x
0x%04x",src+size-1,*((unsigned short *)(src+size-2)),dest,*((unsigned short *)dest));
                                        return 1;
                                }
                                break;
                        case 2 :
                                if( *((unsigned int *)(src+size-4))==*((unsigned int *)dest) )
                                        return 0;
                                else
                                {
                                        printf("\n* src : 0x%08x 0x%08x  dest : 0x%08x
0x%08x",src+size-1,*((unsigned int *)(src+size-2)),dest,*((unsigned int *)dest));
                                        return 1;
                                }
                                break;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                        default :
                                printf("\nSome error~!!!!!!!!!!");
                                break;
                        }
                  }
                }
                else
                        destData=*((volatile unsigned char *)(dest+i));
                if( srcData!=destData )
                {
                        printf("\n* src : 0x%08x 0x%02x  dest : 0x%08x
0x%02x",src+i%mod,srcData,dest+i%mod,destData);
                        cnt++;
                }
        }

        return cnt;
}



UINT InitMemory(UINT addr, UINT size)
{
        UINT i;
        unsigned char data;

        data=0;
        for(i=0;i<size;i++)
        {
                data++;
                *((unsigned char *)(addr+i))=data;
        }
        data=0;
        for(i=0;i<size;i++)
        {
                data++;
                if(data!=*((unsigned char *)(addr+i)))
                        return -1;
        }
        return 0;
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
UINT ZeroMemory(UINT addr, UINT size)
{
        UINT i;

        for(i=0;i<size;i++)
                *((unsigned char *)(addr+i))=0x0;
        for(i=0;i<size;i++)
                if(0x0!=*((unsigned char *)(addr+i)))
                        return -1;
        return 0;
}
```

```
void InitExtIoMemory(void)
{
        unsigned int config,BASADDR,size,tACC,tCOH,tACS,tCOS,DBWD;

        BASADDR=0xC6000000;
        /* 256k */
        size=0;
        /* 7MLK */
        tACC=1;
        /* 7MLK */
        tCOH=1;
        /* 7MLK */
        tACS=1;
        /* 7MLK */
        tCOS=1;
        DBWD=2;
        config=(BASADDR<<1)+(size<<16)+(tACC<<11)+(tCOH<<8)+(tACS<<5)+(tCOS<<2)+DBW
D;
        outpw(EXT0CON,config);
        BASADDR=0xC6020000;
        config=(BASADDR<<1)+(size<<16)+(tACC<<11)+(tCOH<<8)+(tACS<<5)+(tCOS<<2)+DBW
D;
        outpw(EXT1CON,config);
        BASADDR=0xC6040000;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        config=(BASADDR<<1)+(size<<16)+(tACC<<11)+(tCOH<<8)+(tACS<<5)+(tCOS<<2)+DBW
D;
        outpw(EXT2CON,config);

}


void InstallGdmaHandler(void)
{
        *((volatile unsigned int *)0x38)=(unsigned int)GdmaIrqHandler;
        /* High-level sensitive, priority 1 */
        *((volatile unsigned int *)AIC_SCR17)=0x00000041;
        /* High-level sensitive, priority 1 */
        *((volatile unsigned int *)AIC_SCR18)=0x00000041;
   /* enable AP board interrupt */
        *((volatile unsigned int *)AIC_MDCR)=0x0007FFFE;
        *((volatile unsigned int *)AIC_MECR)=0x00060000;
   *(volatile unsigned int *)0x14000008 = 0x200;                /* IRQ LM 0 */
   /* enable CPSR I bit */
   __asm
   {
      MRS    r9, CPSR
      BIC    r9, r9,0x80
      MSR    CPSR_c, r9
   }
}

__irq void GdmaIrqHandler(void)
{
        unsigned int intSource;

        intSource=*((volatile unsigned int *)AIC_IPER);

        if( intSource==0x44 )
        {
                IsGdmaChannel0IntOk=1;
                *((volatile UINT *)GDMA_CTL0)&=(~TC);
        }
        if( intSource==0x48 )
        {
                IsGdmaChannel1IntOk=1;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                *((volatile UINT *)GDMA_CTL1)&=(~TC);
        }

        *((volatile unsigned int *)AIC_EOSCR)=0;
}

__irq void GdmaIrqHandler2(void)
{
        unsigned int intSource;

        intSource=*((volatile unsigned int *)AIC_IPER);

        if( intSource==0x44 )
        {
                IsGdmaChannel0IntOk++;
                *((volatile UINT *)GDMA_CTL0)&=(~TC);
                if( IsGdmaChannel0IntOk < ROBUST_TEST_NUM )
                        *((volatile UINT *)GDMA_CTL0)|=GDMAEN;
        }
        if( intSource==0x48 )
        {
                IsGdmaChannel1IntOk++;
                *((volatile UINT *)GDMA_CTL1)&=(~TC);
                if( IsGdmaChannel1IntOk < ROBUST_TEST_NUM )
                        *((volatile UINT *)GDMA_CTL1)|=GDMAEN;
        }

        *((volatile unsigned int *)AIC_EOSCR)=0;
}


__irq void AicIrqHandler(void)
{
        unsigned int intSource;
        aic_iasr=*((volatile UINT *)AIC_IASR);
        aic_isr=*((volatile UINT *)AIC_ISR);
        aic_oisr=*((volatile UINT *)AIC_OISR);

        intSource=*((volatile unsigned int *)AIC_IPER);

        if( intSource==0x44 )
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        {
                IsGdmaChannel0IntOk++;
                *((volatile UINT *)GDMA_CTL0)&=(~TC);
        }
        if( intSource==0x48 )
        {
                IsGdmaChannel1IntOk++;
                *((volatile UINT *)GDMA_CTL1)&=(~TC);
        }

        *((volatile unsigned int *)AIC_EOSCR)=0;
}


void GdmaInitSdram(void)
{
        unsigned int config,tcfg;
        unsigned int REFEN,REFMOD,CLKEN,REFRAT,WAITVT,LITTLE;
        unsigned int BASADDR,MRSET,AUTOPR,LATENCY,COMPBK,DBWD,COLUMN,size;
        unsigned int tRCD,tRDL,tRP,tRAS;
        unsigned int memory_size;
        int bank;
        int module;
        unsigned int sdram_type[4][9]={

        /*BASADDR,MRSET,AUTOPR,LATENCY,LENGTH,COMPBK,DBWD,COLUMN,SIZE*/


        {0xc2000000,1,0,1,0,0,2,0,1},/* 2M,16bits,2bank,8 */

        {0xc2000000,1,0,1,0,1,1,2,5},/* 32M,8bits,4bank,10 */

        {0xc2000000,1,0,1,0,1,3,0,3},/* 8M,32bits,4bank,8 */

        {0xc2000000,1,0,1,0,1,3,1,5} /* 32M,32bits,4bank,9 */
                                                                        };


        printf("\nInitialize SDRAM (16Mx16) ...");
        bank=0;
        REFEN=1;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        REFMOD=0;
        CLKEN=1;
        REFRAT=0x138;
        WAITVT=0;
        LITTLE=0;
        config=(REFEN<<18)+(REFMOD<<17)+(CLKEN<<16)+(REFRAT<<3)+(WAITVT<<1)+LITT
LE;

        outpw(EBICON,config);


        BASADDR=0xC2000000;
        MRSET=1;
        AUTOPR=0;
        LATENCY=1;
        /* 0=2 banks, 1=4 banks */
        COMPBK=1;
        /* 1=8bits,2=16bits,3=32bits */
        DBWD=3;
        /* 0=8bits,1=9bits,2=10bits */
        COLUMN=1;
        /* 1=2M,2=8M,3=16M,4=32M */
        size=4;
        #if 0
        printf("\nSIZE(1=2M,2=4M,3=8M,4=16M,5=32M,6=64M) : ");
        scanf("%u",&size);
        printf("\nCOMPBK(0=2banks,1=4banks) : ");
        scanf("%u",&COMPBK);
        printf("DBWD(1=8bits,2=16bits,3=32bits) : ");
        scanf("%u",&DBWD);
        printf("COLUMN(0=8bits,1=9bits,2=10bits) : ");
        scanf("%u",&COLUMN);
        #else
        module=3;
        BASADDR=sdram_type[module][0];
        MRSET=sdram_type[module][1];
        AUTOPR=sdram_type[module][2];
        LATENCY=sdram_type[module][3];
        COMPBK=sdram_type[module][5];
        DBWD=sdram_type[module][6];
        COLUMN=sdram_type[module][7];
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
size=sdram_type[module][8];
#endif
switch(size)
{
        case 1 :
                memory_size=0x200000;
                break;
        case 2 :
                memory_size=0x400000;
                break;
        case 3 :
                memory_size=0x800000;
                break;
        case 4 :
                memory_size=0x1000000;
                break;
        case 5 :
                memory_size=0x2000000;
                break;
        default :
                memory_size=0;
                break;
}

/* 0=1,1=2....7=8 */
tRCD=1;
/* 0=1,1=2....3=4 */
tRDL=1;
/* 0=1,1=2....7=8 */
tRP=1;
/* 0=1,...7=8 */
tRAS=7;

tcfg=(tRCD<<8)+(tRDL<<6)+(tRP<<3)+(tRAS);
config=(BASADDR<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                +(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;

switch(bank)
{
        case 0 :
                outpw(SDCONF0,config);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                outpw(SDTIME0,tcfg);
                break;
        case 1 :
                BASADDR=BASADDR+(memory_size);
                config=((BASADDR)<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11)\
                        +(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
                outpw(SDCONF1,config);
                outpw(SDTIME1,tcfg);
                break;
        case 2 :
                outpw(SDCONF0,config);
                outpw(SDTIME0,tcfg);

    config=((BASADDR+(memory_size))<<1)+(MRSET<<15)+(AUTOPR<<13)+(LATENCY<<11
)\
                        +(COMPBK<<7)+(DBWD<<5)+(COLUMN<<3)+size;
                outpw(SDCONF1,config);
                outpw(SDTIME1,tcfg);
                break;
    }
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8 EMC (Ethernet MAC Controller)

8.1 Overview

The NUC740 provides two Ethernet MAC Controllers (EMC) for WAN/LAN application. Each EMC consists of

- DMA controller
- Transmit FIFO
- Receive FIFO
- IEEE 802.3/Ethernet protocol engine with internal CAM for entry address comparison
- TX/RX state machine controller
- and status controller.

The EMC supplies selectable MII (Media Independent Interface) or RMII (Reduced MII) interfaces for 10/100M external PHY to operate with 25M/2.5M Hz TXCLK/RXCLK.

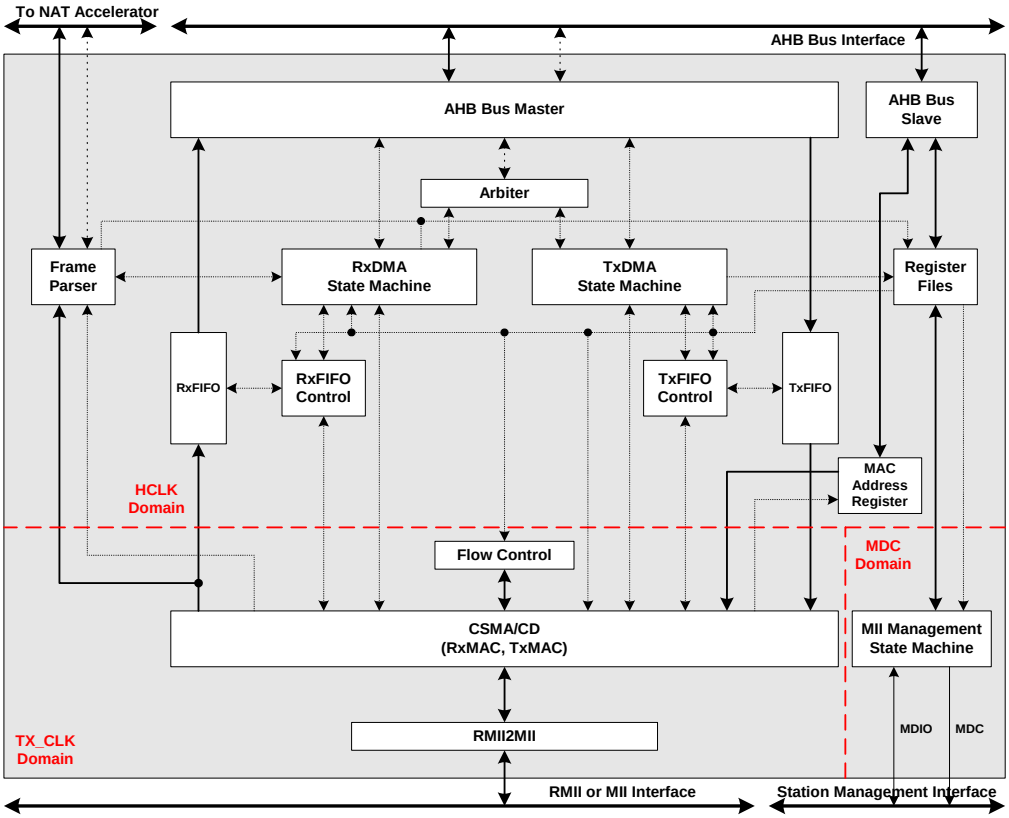
Features :

- IEEE 802.3 protocol engine with programmable MII or RMII interface.
- 256 bytes transmit and receive FIFO for MAC protocol engine and DMA access.
- Can accept long frame (more than 1518 bytes) and short frame (less than 64 bytes).
- Re-transmit (during collision) the frame without DMA access.
- 16-entry CAM configuration.
- Half or full duplex function option.
- Pad generator on chip.
- Support pause and remote pause function for flow control.
- Support station management for external PHY.
- Internal loop back mode for testing.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.2 Block Diagram

Figure 8-1 EMC Block Diagram



8.3 Registers

8.3.1 EMC 0 Control registers

Register	Address	R/W	Description	Reset Value
CAM REGISTERS				
CAMCMR_0	0xFFF0.3000	R/W	CAM Command Register	0x0000.0000
CAMEN_0	0xFFF0.3004	R/W	CAM enable register	0x0000.0000
CAM0M_0	0xFFF0.3008	R/W	CAM0 Most Significant Word Register	0x0000.0000
CAM0L_0	0xFFF0.300C	R/W	CAM0 Least Significant Word Register	0x0000.0000
CAM1M_0	0xFFF0.3010	R/W	CAM1 Most Significant Word Register	0x0000.0000
CAM1L_0	0xFFF0.3014	R/W	CAM1 Least Significant Word Register	0x0000.0000
CAM2M_0	0xFFF0.3018	R/W	CAM2 Most Significant Word Register	0x0000.0000
CAM2L_0	0xFFF0.301C	R/W	CAM2 Least Significant Word Register	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

CAM3M_0	0xFFF0.3020	R/W	CAM3 Most Significant Word Register	0x0000.0000
CAM3L_0	0xFFF0.3024	R/W	CAM3 Least Significant Word Register	0x0000.0000
CAM4M_0	0xFFF0.3028	R/W	CAM4 Most Significant Word Register	0x0000.0000
CAM4L_0	0xFFF0.302C	R/W	CAM4 Least Significant Word Register	0x0000.0000
CAM5M_0	0xFFF0.3030	R/W	CAM5 Most Significant Word Register	0x0000.0000
CAM5L_0	0xFFF0.3034	R/W	CAM5 Least Significant Word Register	0x0000.0000
CAM6M_0	0xFFF0.3038	R/W	CAM6 Most Significant Word Register	0x0000.0000
CAM6L_0	0xFFF0.303C	R/W	CAM6 Least Significant Word Register	0x0000.0000
CAM7M_0	0xFFF0.3040	R/W	CAM7 Most Significant Word Register	0x0000.0000
CAM7L_0	0xFFF0.3044	R/W	CAM7 Least Significant Word Register	0x0000.0000
CAM8M_0	0xFFF0.3048	R/W	CAM8 Most Significant Word Register	0x0000.0000
CAM8L_0	0xFFF0.304C	R/W	CAM8 Least Significant Word Register	0x0000.0000
CAM9M_0	0xFFF0.3050	R/W	CAM9 Most Significant Word Register	0x0000.0000
CAM9L_0	0xFFF0.3054	R/W	CAM9 Least Significant Word Register	0x0000.0000
CAM10M_0	0xFFF0.3058	R/W	CAM10 Most Significant Word Register	0x0000.0000
CAM10L_0	0xFFF0.305C	R/W	CAM10 Least Significant Word Register	0x0000.0000
CAM11M_0	0xFFF0.3060	R/W	CAM11 Most Significant Word Register	0x0000.0000
CAM11L_0	0xFFF0.3064	R/W	CAM11 Least Significant Word Register	0x0000.0000
CAM12M_0	0xFFF0.3068	R/W	CAM12 Most Significant Word Register	0x0000.0000
CAM12L_0	0xFFF0.306C	R/W	CAM12 Least Significant Word Register	0x0000.0000
CAM13M_0	0xFFF0.3070	R/W	CAM13 Most Significant Word Register	0x0000.0000
CAM13L_0	0xFFF0.3074	R/W	CAM13 Least Significant Word Register	0x0000.0000
CAM14M_0	0xFFF0.3078	R/W	CAM14 Most Significant Word Register	0x0000.0000
CAM14L_0	0xFFF0.307C	R/W	CAM14 Least Significant Word Register	0x0000.0000
CAM15M_0	0xFFF0.3080	R/W	CAM15 Most Significant Word Register	0x0000.0000
CAM15L_0	0xFFF0.3084	R/W	CAM15 Least Significant Word Register	0x0000.0000
MAC REGISTERS				
MIEN_0	0xFFF0.3088	R/W	MAC Interrupt Enable Register	0x0000.0000
MCMDR_0	0xFFF0.308C	R/W	MAC Command Register	0x0000.0000
MIID_0	0xFFF0.3090	R/W	MII Management Data Register	0x0000.0000
MIIDA_0	0xFFF0.3094	R/W	MII Management Data Control and Address Register	0x0090.0000
MPCNT_0	0xFFF0.3098	R/W	Missed Packet counter register	0x0000.7FFF
DMA REGISTERS				
TXDLSA_0	0xFFF0.309C	R/W	Transmit Descriptor Link List Start Address register	0xFFFF.FFFC
RXDLSA_0	0xFFF0.30A0	R/W	Receive Descriptor Link List Start Address register	0xFFFF.FFFC
DMARFC_0	0xFFF0.30A4	R/W	DMA Receive Frame Control Register	0x0000.0800
TSDR_0	0xFFF0.30A8	W	Transmit Start Demand Register	Undefined
RSDR_0	0xFFF0.30AC	W	Receive Start Demand Register	Undefined
FIFOTHD_0	0xFFF0.30B0	R/W	FIFO Threshold Adjustment Register	0x0000.0101

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.3.2 EMC 0 Status Registers

Register	Address	R/W	Description	Reset Value
MAC REGISTERS				
MISTA_0	0xFFF0.30B4	R/W	MAC Interrupt Status Register	0x0000.0000
MGSTA_0	0xFFF0.30B8	R/W	MAC General Status Register	0x0000.0000
MRPC_0	0xFFF0.30BC	R	MAC Receive Pause count register	0x0000.0000
MRPCC_0	0xFFF0.30C0	R	MAC Receive Pause Current Count Register	0x0000.0000
MREPC_0	0xFFF0.30C4	R	MAC Remote pause count register	0x0000.0000
DMA REGISTERS				
DMARFS_0	0xFFF0.30C8	R/W	DMA Receive Frame Status Register	0x0000.0000
CTXDSA_0	0xFFF0.30CC	R	Current Transmit Descriptor Start Address Register	0x0000.0000
CTXBSA_0	0xFFF0.30D0	R	Current Transmit Buffer Start Address Register	0x0000.0000
CRXDSA_0	0xFFF0.30D4	R	Current Receive Descriptor Start Address Register	0x0000.0000
CRXBSA_0	0xFFF0.30D8	R	Current Receive Buffer Start Address Register	0x0000.0000

8.3.3 EMC 1 Control Registers

Register	Address	R/W	Description	Reset Value
CAM REGISTERS				
CAMCMR_1	0xFFF0.3800	R/W	CAM Command Register	0x0000.0000
CAMEN_1	0xFFF0.3804	R/W	CAM enable register	0x0000.0000
CAM0M_1	0xFFF0.3808	R/W	CAM0 Most Significant Word Register	0x0000.0000
CAM0L_1	0xFFF0.380C	R/W	CAM0 Least Significant Word Register	0x0000.0000
CAM1M_1	0xFFF0.3810	R/W	CAM1 Most Significant Word Register	0x0000.0000
CAM1L_1	0xFFF0.3814	R/W	CAM1 Least Significant Word Register	0x0000.0000
CAM2M_1	0xFFF0.3818	R/W	CAM2 Most Significant Word Register	0x0000.0000
CAM2L_1	0xFFF0.381C	R/W	CAM2 Least Significant Word Register	0x0000.0000
CAM3M_1	0xFFF0.3820	R/W	CAM3 Most Significant Word Register	0x0000.0000
CAM3L_1	0xFFF0.3824	R/W	CAM3 Least Significant Word Register	0x0000.0000
CAM4M_1	0xFFF0.3828	R/W	CAM4 Most Significant Word Register	0x0000.0000
CAM4L_1	0xFFF0.382C	R/W	CAM4 Least Significant Word Register	0x0000.0000
CAM5M_1	0xFFF0.3830	R/W	CAM5 Most Significant Word Register	0x0000.0000
CAM5L_1	0xFFF0.3834	R/W	CAM5 Least Significant Word Register	0x0000.0000
CAM6M_1	0xFFF0.3838	R/W	CAM6 Most Significant Word Register	0x0000.0000
CAM6L_1	0xFFF0.383C	R/W	CAM6 Least Significant Word Register	0x0000.0000
CAM7M_1	0xFFF0.3840	R/W	CAM7 Most Significant Word Register	0x0000.0000
CAM7L_1	0xFFF0.3844	R/W	CAM7 Least Significant Word Register	0x0000.0000
CAM8M_1	0xFFF0.3848	R/W	CAM8 Most Significant Word Register	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

CAM8L_1	0xFFF0.384C	R/W	CAM8 Least Significant Word Register	0x0000.0000
CAM9M_1	0xFFF0.3850	R/W	CAM9 Most Significant Word Register	0x0000.0000
CAM9L_1	0xFFF0.3854	R/W	CAM9 Least Significant Word Register	0x0000.0000
CAM10M_1	0xFFF0.3858	R/W	CAM10 Most Significant Word Register	0x0000.0000
CAM10L_1	0xFFF0.385C	R/W	CAM10 Least Significant Word Register	0x0000.0000
CAM11M_1	0xFFF0.3860	R/W	CAM11 Most Significant Word Register	0x0000.0000
CAM11L_1	0xFFF0.3864	R/W	CAM11 Least Significant Word Register	0x0000.0000
CAM12M_1	0xFFF0.3868	R/W	CAM12 Most Significant Word Register	0x0000.0000
CAM12L_1	0xFFF0.386C	R/W	CAM12 Least Significant Word Register	0x0000.0000
CAM13M_1	0xFFF0.3870	R/W	CAM13 Most Significant Word Register	0x0000.0000
CAM13L_1	0xFFF0.3874	R/W	CAM13 Least Significant Word Register	0x0000.0000
CAM14M_1	0xFFF0.3878	R/W	CAM14 Most Significant Word Register	0x0000.0000
CAM14L_1	0xFFF0.387C	R/W	CAM14 Least Significant Word Register	0x0000.0000
CAM15M_1	0xFFF0.3880	R/W	CAM15 Most Significant Word Register	0x0000.0000
CAM15L_1	0xFFF0.3884	R/W	CAM15 Least Significant Word Register	0x0000.0000
MAC REGISTERS				
MIEN_1	0xFFF0.3888	R/W	MAC Interrupt Enable Register	0x0000.0000
MCMDR_1	0xFFF0.388C	R/W	MAC Command Register	0x0000.0000
MIID_1	0xFFF0.3890	R/W	MII Management Data Register	0x0000.0000
MIIDA_1	0xFFF0.3894	R/W	MII Management Data Control and Address Register	0x0090.0000
MPCNT_1	0xFFF0.3898	R/W	Missed Packet counter register	0x0000.7FFF
DMA REGISTERS				
TXDLSA_1	0xFFF0.389C	R/W	Transmit Descriptor Link List Start Address register	0xFFFF.FFFC
RXDLSA_1	0xFFF0.38A0	R/W	Receive Descriptor Link List Start Address register	0xFFFF.FFFC
DMARFC_1	0xFFF0.38A4	R/W	DMA Receive Frame Control Register	0x0000.0800
TSDR_1	0xFFF0.38A8	W	Transmit Start Demand Register	Undefined
RSDR_1	0xFFF0.38AC	W	Receive Start Demand Register	Undefined
FIFOTHD_1	0xFFF0.38B0	R/W	FIFO Threshold Adjustment Register	0x0000.0101

8.3.4 EMC 1 Status Registers

Register	Address	R/W	Description	Reset Value
MAC REGISTERS				
MISTA_1	0xFFF0.38B4	R/W	MAC Interrupt Status Register	0x0000.0000
MGSTA_1	0xFFF0.38B8	R/W	MAC General Status Register	0x0000.0000
MRPC_1	0xFFF0.38BC	R	MAC Receive Pause count register	0x0000.0000
MRPCC_1	0xFFF0.38C0	R	MAC Receive Pause Current Count Register	0x0000.0000
MREPC_1	0xFFF0.38C4	R	MAC Remote pause count register	0x0000.0000
DMA REGISTERS				
DMARFS_1	0xFFF0.38C8	R/W	DMA Receive Frame Status Register	0x0000.0000
CTXDSA_1	0xFFF0.38CC	R	Current Transmit Descriptor Start Address Register	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

CTXBSA_1	0xFFF0.38D0	R	Current Transmit Buffer Start Address Register	0x0000.0000
CRXDSA_1	0xFFF0.38D4	R	Current Receive Descriptor Start Address Register	0x0000.0000
CRXBSA_1	0xFFF0.38D8	R	Current Receive Buffer Start Address Register	0x0000.0000

8.4 Functional Descriptions

8.4.1 Media Access Control (MAC)

The function of NUC740 MAC fully meets the requirements defined by the IEEE802.3u specification. The transmission data frame sent from the transmit DMA will be encapsulated by the MAC before transmitting onto the MII bus. The sent data will be assembled with the preamble, the start frame delimiter (SFD), the frame check sequence ,and the padding for enforcing those packets less than 64 bytes, to meet the requirement of minimum size of frame and CRC sequence. The outgoing frame format will be :

10101010 ---- 10101010	1010111	d0	d1	d2	---	dn	Padding	CRC31	CRC30	---	CRC0

As mentioned by the above format, the preamble is a consecutive 7-byte long with the pattern “10101010” and the SFD is a one byte 10101011 data. The padding data will be all 0 value if the length of the data frame is less than 64 bytes. The bits CRC0 ... CRC31 are the 32 bits cyclic redundancy check (CRC) sequence.

8.4.2 Rx Buffer Descriptor (RXBD)

3 1	**3 0**	**2 9**	**1 6**	**1 5**	**0**
O	**Rx Status**		**Frame Length**		
Data Buffer Starting Address					
NAT Information (Reserved)					
Next Descriptor Starting Address					

O : Ownership bits

Bit[31 : 30]	Ownership bits
00b	CPU
10b	DMA
11b	NATA
01b	Undefined

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

EMC receive DMA is allowed to access current descriptor if bit 31 is set to 1 by the user driver program. If the entire frame is received successfully, then the ownership bit 31 is cleared and the ownership is granted to CPU. If NATA is enabled, NATA is also allowed to access current descriptor and bit 30 is set to 1 by NATA when NATA is processing.

Rx Status : Receive Status

This field is updated by EMC after reception completed.

Frame Length : Received Frame Length

This field is the size of the received frame.

Data Buffer Starting Address

This field is the starting address of the frame data to be received.

Next Descriptor Start Address

This field is the start address of the next frame descriptor.

NAT Information

This field is reserved for MAC Rx to set information for NAT processing. For user driver program, it is forbidden to modify these bits.

8.4.3 Rx Status (RXSTA) : Receive Status

		29	28	27	26	25	24
		Hit	IPHit	PortHit	Inverse	NATFSH	Nop
23	22	21	20	19	18	17	16
Reserved	RP	ALIE	RXGD	PTLE	Reserved	CRCE	RXINTR

Bits 29-24 are NAT information for the NAT accelerator, and reserved if the NATA is disabled.

Hit : current packet is hit with NAT entry table

1 : Current packet IP/Port is in the entry list.

If NATA is disabled, the bit is reserved.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

IPHit : current packet is hit on IP address

1 : Current packet IP/Port is hit by the IP address.

PortHit : current packet is hit on Port Number

1 : Current packet IP/port is hit by the Port number.

Inverse : current hit entry is setting on inverse mode

1 : Current hit entry is on inverse mode.

NATFSH : NAT Processing Finish

1 : Current packet NAT processing is finished and successful.

NOP : No Operation

This bit indicates the packet is hit in NAT table but no need to be processed by NATA. This bit will be set to 1 if the packet was hit on the NAT table and the corresponding NOP and Discard bit of hit entry is 2'b10.

RP : Runt Packet

Set if the length of the received packet is less than 64 bytes.

ALIE : Alignment Error

Set if the received bit length of are not a multiple of eight.

RXGD : Receiving Good packet received

Set if the MAC successfully receives a packet with no errors. If **EnRXGD** = 1, an interrupt is generated when this bit is set.

PTLE : Packet Too Long Error

Set if the length of a received frame is longer than 1518 bytes. Not to set if the **ALP** (Accept Long Packet) bit is set.

CRCE : CRC Error

Set if the CRC at end of packet does not match the computed value, or the PHY asserts Rx_er during packet reception.

RXINTR : Interrupt on receive

Set if reception of packet caused an interrupt condition. This includes Good received, if the **EnRXGD** is set.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.4 NAT INFORMATION (Reserved if disabled)

31	30	29	28	27	26	25	24
Reserved		TCP information					
23	22	21	20	19	18	17	16
Reserved	UCK_Err	TU_Err	NH_Err	IP Header Length			
15	14	13	12	11	10	9	8
Reserved		Hit Entry Number					
7	6	5	4	3	2	1	0
Reserved	PPPCaps	PPPoE	UCKS	UDP	TCP	L/W	Hit

TCP information : URG (bit 29), ACK, PSH, RST, SYN, FIN (bit 24)

The six bit values show current TCP status, and are the same as the six bits in TCP header. The values are valid only if current packet is TCP type and **Hit** is set.

UCK_Err : TCP/UCKS Error
TU_Err : TCP/UDP Error
NH_Err : No Hit Error

These bits record error status if NAT processing error is occurred and are wrote by NATA.

UCK_Err	TCP = 1 and UCKS = 1
TU_Err	TCP = 1 and UDP = 1
NH_Err	No hit error

IP Header Length : TCP/UDP header location offset

The offset value lets the NAT accelerator to identify the starting address of TCP or UDP header, which is used for NAT to parsing port data. The value is valid only when **Hit** is set.

Hit Entry Number : the entry number hit with the input address

The value indicates which entry is hit to let NAT accelerator to take corresponding data. The value is valid only when Hit is set.

PPPCaps : PPPoE datagram encapsulated

1 : PPP encapsulation is used(8 bits protocol field).
0 : Encapsulation is not used (16 bits protocol field).

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

PPPoE : PPPoE protocol

1 : The packet takes PPPoE protocol instead of IP protocol.
0 : The packet takes IP protocol.

UCKS : UDP protocol with skip checksum replacement

The bit was set when the packet takes UDP protocol, and its checksum is zero. The NAT accelerator will skip the checksum replacement procedure.

UDP : apply UDP protocol

1 : if the packet takes UDP protocol.
0 : if the packet takes non-UDP protocol.
The value is valid only if **Hit** is set to notify the NAT engine this is a UDP packet

TCP : apply TCP protocol

1 : if the packet takes TCP protocol.
0 : if the packet takes non-TCP protocol.
The value is valid only if **Hit** is set to notify the NAT engine this packet is the TCP packet.

L/W : hit port; the value is 1 if internal (LAN) port gets hit, and 0 if external (WAN) port is hit

Program must specify LAN port and WAN port with the two EMCs. The L/W value is 1 if the hit port is internal port, and 0 if the hit port is external port. The value is valid if Hit is set.

Hit : current packet is hit with NAT entry table

The value is 1 if current packet IP/Port is in the entry list. If NAT is disabled, the bit is reserved.

8.4.5 DMA Receive Descriptor Link List Start Address Register

Register	Address	R/W	Description	Reset Value
RXDLSA_0	0xFFF0.30A0	R/W	Receive Descriptor Link List Start Address register	0xFFFF.FFFC
RXDLSA_1	0xFFF0.38A0	R/W	Receive Descriptor Link List Start Address register	0xFFFF.FFFC

31	30	29	28	27	26	25	24
RXDLSA							
23	22	21	20	19	18	17	16
RXDLSA							
15	14	13	12	11	10	9	8

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

RXDLSA							
7	6	5	4	3	2	1	0
RXDLSA							

RXDLSA [31 : 0] : DMA receive descriptor link list start address
Default value : *0xFFFF.FFFC*

This register defines the receive descriptor link list start address for frame reception process. While software first turn on RxDMA after the hardware reset or software reset, the content of this register will be loaded into the current receive descriptor start address. The value of this register will not be updated by any hardware operation. The system would ignore the least 2 significant bits to fit the word alignment requirement.

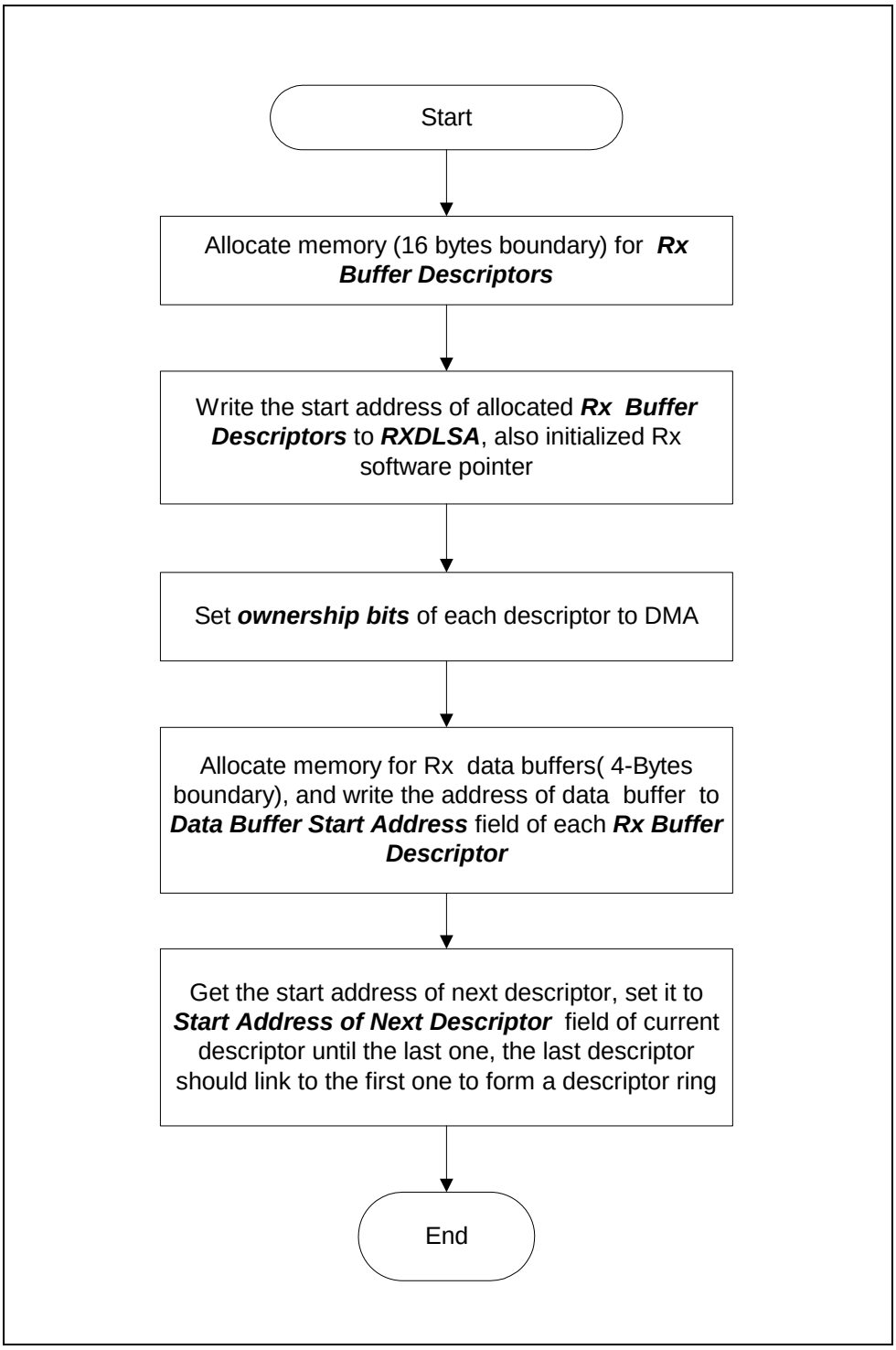
8.4.6 Initialize Rx Buffer Descriptors

(1) Allocate memory for Rx descriptors.
(2) Write start address of (1) to *RXDLSA* register and let Rx software pointer point to this address.
(3) Set ownership bits of each descriptor to DMA.
(4) Allocate memory as data buffer and write the address to *data buffer start address* field of Rx descriptor.
(5) Set start address of next descriptor, this field of the last descriptor should set to the address of the first descriptor.
(6) The start address of descriptor and data buffer are suggested to be aligned to 16 bytes address boundary.

Figure 8-2 lists the Rx Descriptor initialization flow.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 8-2 Rx Descriptor Initialization



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.7 Tx Buffer Descriptor (TXBD)

31	30	16	15	3	2	1	0
O					I	C	P
Data Buffer Starting Address							
Tx Status			Frame Length				
Next Descriptor Starting Address							

O : Ownership bit

0 = CPU

1 = DMA

NUC740 transmit DMA is allowed to access current descriptor if this bit is set to '1' (Owned by DMA) by the driver program. If the entire frame is transmitted successfully, then the ownership bit is cleared by MAC and the ownership is switched to CPU.

I : MAC transmit interrupt enable after transmission complete of the frame

0 = Disable

1 = Enable

C : CRC mode bit

0 = Disable CRC mode

1 = Enable CRC mode

P : Padding mode bit

0 = Disable padding mode

1 = Enable padding mode

Data Buffer Starting Address

Starting address of the frame data to be transmitted.

Tx Status : Transmit Status

This field is updated by the EMC after the packet was transmitted.

Frame Length

Length of the transmitting frame.

Next Descriptor Starting Address

Starting address of the next frame descriptor.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.8 Tx Status (TXSTA)

31	30	29	28	27	26	25	24
CCNT					SEQ	PAU	TXHA
23	22	21	20	19	18	17	16
LC	TXABT	NCS	EXDEF	TXCP	Reserved	DEF	TXINTR

TXINTR : Interrupt on Transmit

Set if transmission of packet causes an interrupt condition. It includes **TXCP (Transmission Complete)**.

DEF : Transmit deferred

Set when MAC has to defer to transmit packet, if MAC is ready to transmit a frame, but the carrier sense input is asserted.

TXCP : Transmission Completion

Set when MAC completes a transmission, or one packet was discarded.

EXDEF : Exceed Deferral

Set if MAC deferring time to transmit exceeds 0.32768ms for 100Mbit/s, or 3.2768ms for 10Mbit/s.

NCS : No Carrier Sense Error

Set if carrier sense is not detected during the entire transmission time of a packet..

TXABT : Transmission Abort

Set if transmitting aborted, due to there are 16 collisions occurred during the transmission of a packet.

LC : Late Collision

Set if there is collision occurs after 64 bytes of collision window.

TXHA : Transmission halted

Transmission was halted, due to the **TXON** bit in the **MCMDR was cleared**.

PAU : Paused

Transmission is paused by a remote flow control command of full-duplex mode.

SQE : SQE error

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

After transmitting a frame, set if the fake collision signal did not come from the PHY for 1.6 µs on 10Mbps, half-duplex mode.

CCNT : Transmit Collision Count

Count of collisions during transmission of a single packet. After 16 collisions, **CCNT** is 1111, and **TXABT** is set.

8.4.9 DMA Transmit Descriptor Link-list Start Address Register

Register	Address	R/W	Description	Reset Value
TXDLSA_0	0xFFF0.309C	R/W	Transmit Descriptor Link-list Start Address register	0xFFFF.FFFC
TXDLSA_1	0xFFF0.389C	R/W	Transmit Descriptor Link-list Start Address register	0xFFFF.FFFC

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

TXDLSA [31 : 0] : DMA transmit descriptor link list start address

Default value : 0xFFFF.FFFC

This register defines the transmit descriptor link list start address for transmission process. While software first turn on TxDMA after the hardware reset or software reset, the content of this register will be loaded into the current transmit descriptor start address. The value of this register will not be updated by any hardware operation. The system would ignore the least 2 significant bits to fit the requirement of word alignment.

8.4.10 Initialize Tx Buffer Descriptors

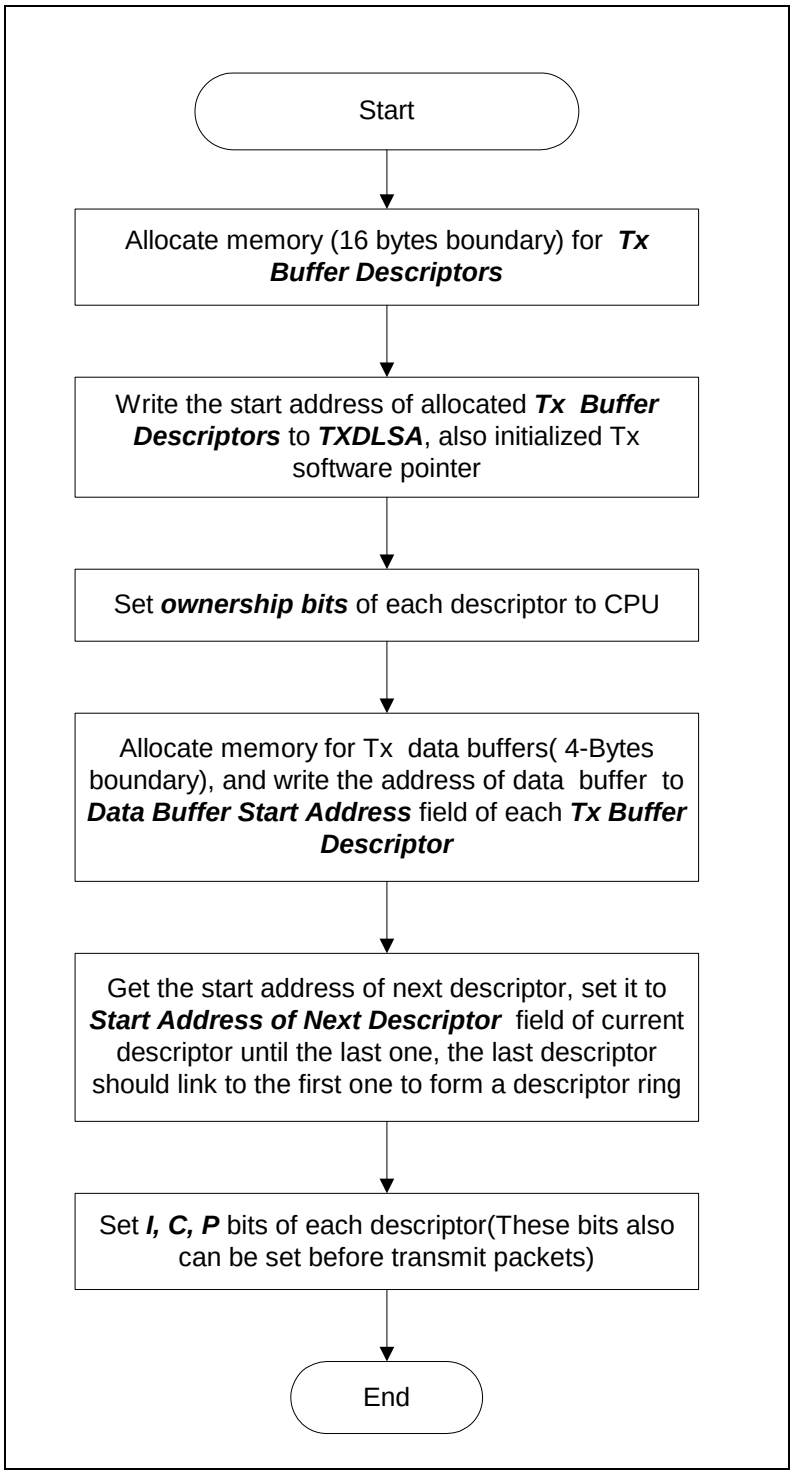
The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

(1) Allocate memory for Tx descriptors.
(2) Write start address of (1) to *TXDLSA* register and let Tx software pointer point to this address.
(3) Set ownership bits of each descriptor to CPU.
(4) Allocate memory to save frame data and write the address to *data buffer start address* field of Tx descriptor.
(5) Set start address of next descriptor, this field of the last descriptor should set to the address of the first descriptor.
(6) Set *I,C,P* bits of each descriptor(The bits can also be set before transmitting packets).
(7) The start address of descriptor and data buffer are suggested to be 16 bytes alignment.

Figure 8-3 lists the Tx Descriptor initialization flow.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 8-3 Tx Descriptor Initialization



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.11 CAM Command Register

The ABP, AMP, and AUP three accept bits in the CAMCMR_x are used to override CAM rejections or accept ion operations. To place the MAC in promiscuous mode, use CAMCMR_x settings to accept packets with these three types of destination address. The three types of destination address packets are listed as followed :

1. Station packets, *xxxxxxx0-xxxxxxxx-xxxxxxxx-xxxxxxxx-xxxxxxxx-xxxxxxxx*
2. Multicast packet, *xxxxxxx1-xxxxxxxx-xxxxxxxx-xxxxxxxx-xxxxxxxx-xxxxxxxx*. (but *x* not all *1*)
3. Broadcast packet, *11111111-11111111-11111111-11111111-11111111-11111111*

Register	Address	R/W	Description	Reset Value
CAMCMR_0	0xFFF0.3000	R/W	CAM Command Register	0x0000.0000
CAMCMR_1	0xFFF0.3800	R/W	CAM Command Register	0x0000.0000

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

ECMP [4] : Enable CAM Compare

Set this bit to enable compare mode.

CCAM [3] : Complement CAM

Set this bit to do complement CAM compare logic, and data packets rejected when the destination address can be recognized by CAM.

ABP [2] : Accept Broadcast Packet

Set this bit to accept any packet with a Broadcast address.

AMP [1] : Accept Multicast Packet

Set this bit to accept any packet with a Multicast address.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

AUP [0] : Accept Unicast Packet

Set this bit to accept any packet with a Unicast address.

Figure 8-4 CAM Setting Combination and Received Packets

Setting	Reception	Setting	Reception	Setting	Reception	Setting	Reception
		N	C U M B	C	C	CN	U M B
B	B	N B	C U M B	C B	C B	CN B	U M B
M	M	N M	C U M B	C M	C M	CN M	U M B
MB	M B	N MB	C U M B	C MB	C M B	CN MB	U M B
U	C U	NU	C U M B	C U	C U	CNU	C U M B
U B	C U B	NU B	C U M B	C U B	C U B	CNU B	C U M B
UM	C U M	NUM	C U M B	C UM	C U M	CNUM	C U M B
UMB	C U M B	NUMB	C U M B	C UMB	C U M B	CNUMB	C U M B

Setting : C = compare mode, N = negative compare mode,
U = accept unicast address, M = accppt multicast address, B = accept broadcast address

Reception : C = address set in CAM, U = unicast address,
M = multicast address, B = broadcast address

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.12 CAM Enable Register

Register	Address	R/W	Description	Reset Value
CAMEN_0	0xFFF0.3004	R/W	CAM enable register	0x0000.0000
CAMEN_1	0xFFF0.3804	R/W	CAM enable register	0x0000.0000

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
CAM15EN	CAM14EN	CAM13EN	CAM12EN	CAM11EN	CAM10EN	CAM9EN	CAM8EN
7	6	5	4	3	2	1	0
CAM7EN	CAM6EN	CAM5EN	CAM4EN	CAM3EN	CAM2EN	CAM1EN	CAM0EN

CAMxEN (x : 15 ~ 0) [15 : 0] : CAM Enable bits

Set the bits in this 16-bit value to selectively enable CAM entry locations from 0 through 15. To disable a CAM entry, clear the corresponding bit.

Note :
The CAM13EN, CAM14EN, and CAM15EN has to be set on sending pause control packet**.**

8.4.13 CAM Address Registers

There are 16 entries of CAM Address Registers, 13 for the Destination Address (entries 0~12) , and 3 for the Pause Control Packet (entries 13~15). For each of the destination address(6 bytes each), it should be programmed into one pair of the CAM address registers *(CAMxM, CAMxL)*.

To send a Pause Control Packet, write the register set **{CAM13M, CAM13L}** with the destination address, the **{CAM14M, CAM14L}** entry with the source address, and the **{CAM15M, CAM15L}** entry with length/type, op-code, and operand, then set the **SDPZ** bit in the **MCMDR** (MAC Command Register).

To activate the CAM function, the corresponding enable bit should be set in the **CAMEN** register.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Register	Address	R/W	Description	Reset Value
CAM0M_0	0xFFF0.3008	R/W	CAM0 Most Significant Word Register	0x0000.0000
CAM0L_0	0xFFF0.300C		CAM0 Least Significant Word Register	0x0000.0000
|	|		|	|
CAM15M_0	0xFFF0.3080		CAM15 Most Significant Word Register	0x0000.0000
CAM15L_0	0xFFF0.3084		CAM15 Least Significant Word Register	0x0000.0000
CAM0M_1	0xFFF0.3808	R/W	CAM0 Most Significant Word Register	0x0000.0000
CAM0L_1	0xFFF0.380C		CAM0 Least Significant Word Register	0x0000.0000
|	|		|	|
CAM15M_1	0xFFF0.3880		CAM15 Most Significant Word Register	0x0000.0000
CAM15L_1	0xFFF0.3884		CAM15 Least Significant Word Register	0x0000.0000

CAMxM (CAM15M excluded)

31	30	29	28	27	26	25	24
Destination Address Byte 6 (Most Significant Byte)							
23	22	21	20	19	18	17	16
Destination Address Byte 5							
15	14	13	12	11	10	9	8
Destination Address Byte 4							
7	6	5	4	3	2	1	0
Destination Address Byte 3							

CAMxL (CAM15L excluded)

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

{CAMxM, CAMxL} : destination address (6 byte), with 2 bytes in CAMxL and 4 bytes in CAMxM, (CAM15M and CAM15L excluded). For example, if the address of Entry CAM 1 is desired to store 12-34-56-78-90-13, then the content of CAM1M is 12-34-56-78, and the content of CAM1L is 90-13-00-00.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

CAM15M (for Pause Control Packet)

31	30	29	28	27	26	25	24
Length / Type (2 bytes) (Most Significant Byte)							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Length / Type (2 bytes)							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Op-code (2 bytes) (Most Significant Byte)							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Op-code (2 bytes)							

CAM15L (for Pause Control Packet)

31	30	29	28	27	26	25	24
Operand (2 bytes) (Most Significant Byte)							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Operand (2 bytes)							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Reserved							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved							

{CAM15M, CAM15L} entry is the Length/Type, Op-code, and operand of the Pause Control Frame. The Length/Type field is 88-08(h), and the Op-code is 00-01(h).

Note :

1. The MAC address of the network device should be filled to CAM and enable CAM compare mode.
2. The Multicast address “01-80-c2-00-00-01” (Address for Flow Control Frame) should be filled to CAM for control frame receiving if AMP (Accept Multicast Frame) not set.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.14 MAC Interrupt Enable Register

Register	Address	R/W	Description	Reset Value
MIEN_1	0xFFF0.3088	R/W	MAC Interrupt Enable Register	0x0000.0000
MIEN_2	0xFFF0.3888	R/W	MAC Interrupt Enable Register	0x0000.0000

31	30	29	28	27	26	25	24
Reserved							EnTxBErr
23	22	21	20	19	18	17	16
EnTDU	EnLC	EnTXABT	EnNCS	EnEXDEF	EnTXCP	EnTXEMP	EnTXINTR
15	14	13	12	11	10	9	8
Reserved	EnCFR	EnNATErr	EnNATOK	EnRxBErr	EnRDU	EnDEN	EnDFO
7	6	5	4	3	2	1	0
EnMMP	EnRP	EnALIE	EnRXGD	EnPTLE	EnRXOV	EnCRCE	EnRXINTR

EnTxBErr [24] : Enable Transmit Bus ERROR interrupt

Set this bit to enable the interrupt when system bus access error occurred on transmitting data from system memory. When the system bus access error occurred, the Tx state machine will stay at Halt state.

MAC software reset is recommended while this interrupt occurred.

EnTDU [23] : Enable Transmit Descriptor Unavailable interrupt

Set this bit to enable the interrupt, which is generated if transmit descriptors owned to the TxDMA is unavailable. That means, if the TxDMA finds the ownership of descriptors is not belonged to TxDMA, it will generate an interrupt and Tx operation will be ceased till the user issues a write command to Transmit Start Demand register to restart Tx operation.

EnLC [22] : Enable Late Collision interrupt

Set this bit to enable the late collision interrupt, which is generated if a collision occurs after 512 bit times.

EnTXABT [21] : Enable Transmit Abort interrupt

Set this bit to enable the interrupt when 16 collisions occur on transmitting a packet.

EnNCS [20] : Enable No Carrier Sense interrupt

Set this bit to enable the interrupt on no carrier sense is presented during transmission.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

EnEXDEF [19] : Enable Defer interrupt

Set this bit to enable the interrupt when the deferred time exceeding 0.32768ms operated at 100Mbs/s or 3.2768ms operated at 10Mbs/s.

EnTXCP [18] : Enable Transmit Completion interrupt

Set this bit to enable the interrupt on a packet was transmitted, or discarded by MAC.

EnTXEMP [17] : Enable Transmit FIFO Empty interrupt

Set this bit to enable the interrupt on MAC transmit FIFO becomes empty (underflow) during a packet transmitting.

EnTXINTR [16] : Enable Interrupt on Transmit interrupt

Set this bit to enable the interrupt on transmitting a packet that causes an interrupt condition.

EnCFR [14] : Enable Control Frame Receive Interrupt

Set this bit to enable the interrupt, when the incoming frame is a MAC control frame.

EnNATErr [13] : Enable NAT Processing Error Interrupt

Set this bit to enable the interrupt, when there is any error occurred during NATA do the NAT processing.

EnNATOK [12] : Enable NAT Processing OK Interrupt

Set this bit to enable the interrupt, when no error occurred during NATA do the NAT processing.

EnRxBErr [11] : Enable Receive Bus ERROR interrupt

Set this bit to enable the interrupt, when system bus access error on transferring data from MAC to system memory. When received bus error occurred, the Rx state machine will stay at Halt state.

The MAC software reset is recommended when receive bus error occurred.

EnRDU [10] : Enable Receive Descriptor Unavailable interrupt

Set this bit to enable the interrupt, when the receive descriptors owned to the RxDMA is unavailable. That means, if the RxDMA finds the ownership of descriptors is not belonged to RxDMA, it will generate an interrupt and Rx operation will be ceased till the user issues a write command to Receive Start Demand register to restart the Rx operation.

EnDEN [9] : Enable DMA early notification interrupts

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Set this bit to enable the interrupt, once the length field of the current frame is received.

EnDFO [8] : Enable DMA receive frame over maximum size interrupt

Set this bit to enable the interrupt, when the received frame size is larger than the value stored in **DMARFC**.

EnMMP [7] : Enable More Missed Packets interrupt

Set this bit to enable the interrupt, when the missed error counter rolls over.

EnRP [6] : Enable Runt Packet on Receive interrupt

Set this bit to enable the interrupt, when the length of the received frame is shorter than 64 bytes.

EnALIE [5] : Enable Alignment Error interrupt

Set this bit to enable the interrupt, when the bit length of received frame was not a multiple of eight.

EnRXGD [4] : Enable Receive Good interrupt

Set this bit to enable the interrupt, when a packet was successfully received with no errors.

EnPTLE [3] : Enable Packet Too Long interrupt

Set this bit to enable the interrupt, when the MAC received a frame, its length is longer than 1518 bytes (unless ALP in MCMDR is set).

EnRXOV [2] : Enable Receive FIFO Overflow interrupt

Set this bit to enable the interrupt, when the MAC receive FIFO was full when receiving a frame.

EnCRCE [1] : Enable CRC Error interrupt

Set this bit to enable the interrupt, when the CRC at the end of a packet is not correct, or the PHY asserted Rx_er during packet reception.

EnRXINTR [0] : Enable Interrupt on Receive interrupt

Set this bit to enable the interrupt, when the reception of a packet caused an interrupt to be generated. This includes a good received interrupt, if the EnRXGD bit is set.

Note :

1. EnRXINTR and EnTXINTR should be set to enable Rx and Tx interrupts.
2. EnRXGD and EnTXCP should be set for Rx and Tx processing.
3. EnRxBErr and EnTxBErr should be set to handle bus error case.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.15 MAC Interrupt Status Register

When an event is recognized, the MAC controller sets the corresponding MISTA bit. The MISTA bits are cleared by write ones; writing zeros has no effect.

Register	Address	R/W	Description	Reset Value
MISTA_0	0xFFF0.30B4	R/W	MAC Interrupt Status Register	0x0000.0000
MISTA_1	0xFFF0.38B4	R/W	MAC Interrupt Status Register	0x0000.0000

31	30	29	28	27	26	25	24
Reserved							TxBErr
23	22	21	20	19	18	17	16
TDU	LC	TXABT	NCS	EXDEF	TXCP	TXEMP	TXINTR
15	14	13	12	11	10	9	8
Reserved	CFR	NATErr	NATOK	RxBErr	RDU	DENI	DFOI
7	6	5	4	3	2	1	0
MMP	RP	ALIE	RXGD	PTLE	RXOV	CRCE	RXINTR

TxBErr [24] : Transmit Bus Error interrupt

This field will be set if access error from EMC to memory (for example, address undefined in system) is occurred. If the status and **EnTxBErr** in **MIEN** are both set, the EMC_TxINT will be triggered. If the status is set, the Tx operation will be ceased, and the MAC software reset is recommended to reset the EMC.

TDU [23] : Transmit Descriptor Unavailable interrupt

This field will be set if the transmit descriptors owned to the TxDMA is unavailable. If the status and **EnTDU** in **MIEN** are both set, the EMC_TxINT will be triggered. When it occurred, the TxDMA operation will be ceased till the user issues a write command to Transmit Start Demand register to restart the Tx operation.

LC [22] : Late Collision

This bit will be set if a collision occurs after 512 bit times.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

TXABT [21] : Transmit Abort

The bit is set to indicate that there are 16 collisions occurred while transmitting a packet.

NCS [20] : No Carrier Sense

Set to indicate no carrier sense is presented during transmission.

EXDEF [19] : Defer

This bit is set to indicate that defer time exceeded 0.32768ms on 100Mbs/s and 3.2768ms on 10Mbs/s. When the **EnEXDEF** in **MIEN** is set, the internal 15-bit counter will automatically count the deferred bit time and generate interrupt when the counter overflows.

TXCP [18] : Transmit Completion

This bit is set when a packet was transmitted by MAC transmit, or discarded.

TXEMP [17] : Transmit FIFO Empty

Set when MAC transmitting FIFO becomes empty (underflow) during a packet transmission period.

TXINTR [16] : Interrupt on Transmit

This bit is set if there is any interrupt condition occurred during packet transmission.

CFR [14] : Control Frame Receive

This field will be set if the incoming frame is a MAC control frame (type==8808h).

NATErr [13] : NAT Processing Error

This field will be set if there is any error during NATA do the NAT processing. If the status and EnNATErr in MIEN are both set, the EMC_RxINT will be triggered.

NATOK [12] : NAT Processing OK

This field will be set if there is no error during NATA do the NAT processing. If the status and EnNATOK in MIEN are both set, the EMC_RxINT will be triggered.

RxBErr [11] : Receive Bus Error interrupt

This field will be set if the access error from EMC to memory (for example, address undefined in system) is occurred. If the status and **EnBErr** in **MIEN** are both set, the EMC_RxINT will be triggered. If the status is set, the Rx operation will be ceased and a software reset is needed to reset the EMC.

RDU [10] : Receive Descriptor Unavailable interrupt

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

This field will be set if the receive descriptors owned to the RxDMA is unavailable. If the status and **ENRDU** in **MIEN** are both set, the EMC_RxINT will be triggered. When the status is set, the RxDMA operation will be ceased till the user issues a write command to *Receive Start Demand* register to restart Rx operation.

DENI [9] : DMA early notification interrupt

If **EnDEN** is reset, an interrupt will be generated when the length field of the current frame was received.

DFOI [8] : DMA receive frame over maximum size interrupt

If **ENDFO** is set, an interrupt will be generated when the received frame size is larger than the value stored in **RXMS**.

MMP [7] : More Missed Packets than miss rolling over counter flag

This bit is automatically set when the missed error counter rolls over.

RP [6] : Runt Packet

This bits is set, it indicates that the received packet length is less than 64 bytes (unless ARP in MCMDR is set).

ALIE [5] : Alignment Error

This bit is set if the frame length in bits was not a multiple of eight.

RXGD [4] : Receive Good

This bit is set if a packet was successfully received with no errors.

PTLE [3] : Packet Too Long Error

This bit is set if the MAC received a frame longer than 1518 bytes (unless ALP in MCMDR is set).

RXOV [2] : Receive FIFO Overflow error

This bit is set if the MAC receive FIFO was overflow on receiving a frame.

CRCE [1] : CRC Error

This bit is set if the CRC at the end of a packet is not correct, or the PHY asserted Rx_er during packet reception.

RXINTR [0] : Interrupt on Receive

This bit is set if the reception of a packet caused an interrupt to be generated. This includes a good received interrupt, if the EnRXGD bit in MIEN is set.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.16 MAC Command Register

Register	Address	R/W	Description	Reset Value
MCMDR_0	0xFFF0.308C	R/W	MAC Command Register	0x0000.0000
MCMDR_1	0xFFF0.388C	R/W	MAC Command Register	0x0000.0000

31	30	29	28	27	26	25	24
Reserved						Reserved	LAN
23	22	21	20	19	18	17	16
LPCS	EnRMII	LBK	OPMOD	EnMDC	FDUP	EnSQE	SDPZ
15	14	13	12	11	10	9	8
Reserved						NDEF	TXON
7	6	5	4	3	2	1	0
Reserved		SPCRC	AEP	ACP	ARP	ALP	RXON

LAN [24] : LAN Port Setting Mode

Set this bit to set this EMC port as a LAN port, and clear this bit to set the EMC port as a WAN port. The S/W program must initial an EMC as a LAN port, and another EMC as a WAN port (for example, EMC 0 as a LAN port and EMC 1 as a WAN port) to let the NAT accelerator work properly.

LPCS [23] : Low Pin Count Package Switch

Set this bit to switch for low pin count (LPC) package. It is used for chip package with limited pins, such as 128-pin or 160-pin package. If users select low pin count package and use two MACs, users have to use both RMII interface and set this bit to switch internal circuit.

EnRMII [22] : Enable RMII

MAC 1 Bit [23 : 22] LPCS : EnRMII	MAC 0 Bit [23 : 22] LPCS*1 : EnRMII	MAC 1 Interface	MAC 0 Interface	Note
00	X0	MII	MII	*1 : the LPCS of MAC0 bit23 is undefined, which not affect MAC 0 Interface.
00	X1	MII	RMII	
01	X0	RMII	MII	

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

01	**X1**	RMII	RMII	*2 : the RMII interface of MAC1 is shared from pins of MAC0
1X	**X0**	Disable	MII	
10	**X1**	Disable	RMII	
11	**X1**	RMII *2	RMII	

LBK [21] : Loop Back

Set this bit to enable MAC internal loop back mode.

OPMOD [20] : Operation Mode

Set this bit to enable MAC to be operated at 100Mb/s. Clear this bit to enable MAC to be operated at 10Mb/s.

EnMDC [19] : Enable MDC signal

Set this bit to enable MDC clock generation. Clear this bit to disable MDC clock generation. If users want to access the MII management data, the EnMDC bit should be set to enable MDC clock.

FDUP [18] : Full Duplex

Set this bit to perform the full duplex function.

EnSQE [17] : Enable SQE test

Set this bit to enable MII 10-Mb/s SQE test mode.

SDPZ [16] : Send Pause

Set this bit to send a pause command or other MAC control packet. The SDPZ bit will be automatically cleared after the MAC control packet has been transmitted. Write zero to this bit has no effect.

NDEF [9] : No defer

Set this bit to disable defer counter.

TXON [8] : Transmit On

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

When this bit is set, the transmission process will be started. If the bit is clear, transmissions will stop after the current packet is transmitted completely. Users should set the bit restart MAC transmit operation.

SPCRC [5] : Accept Strip CRC Value

Set this bit to enable MAC to check the CRC and then strip it from the message.

AEP [4] : Accept Error Packet

Set this bit to enable MAC to accept error (CRC error) packet.

ACP [3] : Accept Control Packet

Set this bit to enable accept control packets.

ARP [2] : Accept Runt Packet

Set this bit to enable accepting frames with lengths less than 64 bytes.

ALP [1] : Accept Long Packet

Set this bit to enable accepting frames with lengths greater than 1518 bytes.

RXON [0] : Receive ON

This bit is set to enable MAC reception operation. If the bit is clear, receptions will stop after the current packet is received completely. Users should set this bit to restart MAC receive operation.

Note :

1. If users change setting during operation, the updated setting will take effect after current frame is completed.
2. RXON and TXON should be set to enable reception and transmission.
3. FDUP and OPMOD should be set depend on the operation mode of PHY.
4. If SPCRC was set,CRC still is checked, but it will be striped, the received length in descriptor doesn't include the length of CRC (4 bytes).
5. EnMDC should be set to enable MDC clock for accessing MII.
6. LAN bit should be set to appropriate value(LAN/WAN) to let NATA work properly.
7. Before activate the RMII interface, the EnRMII, LPCS and OPMOD should be set, and then aMAC software reset is required to activate it.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.17 MAC General Status Register

Register	Address	R/W	Description	Reset Value
MGSTA_0	0xFFF0.30B8	R/W	MAC General Status Register	0x0000.0000
MGSTA_1	0xFFF0.38B8	R/W	MAC General Status Register	0x0000.0000

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
Reserved				TXHA	SQE	PAU	DEF
7	6	5	4	3	2	1	0
CCNT				Reserved		RXHA	CFR

All the MGSTA bits are write ***ones*** to clear.

TXHA [11] : Transmission Halted

Set to indicate that the transmission is halt by clearing the TXON bit.

SQE [10] : Signal Quality Error

Set to indicate a SQE.

PAU [9] : Pause Bit

Set when transmission was delayed due to a remote Pause command.

DEF [8] : Deferred transmission

This bit is set to indicate the network is busy.

CCNT [7 : 4] : Collision Count

Four bits counter to indicate the number of collisions occurred before the frame is transmitted.

RXHA [1] : Reception Halted

This bit is set if reception is halted by clearing RXON bit in the MAC Command Register (MCMDR).

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

CFR [0] : Control Frame Received

This bit is set if
(1) the packet received is a MAC control frame (type = 8808H),
(2) if the CAM recognizes the packet address, and
(3) if the frame length is 64 bytes.

8.4.18 MII

8.4.18.1 MAC MII Management Data Register

NUC740 provides MII management function to let user access the registers of the external physical layer device. Setting options in MII management registers does not affect the MAC controller operation.

Register	Address	R/W	Description	Reset Value
MIID_0	0xFFF0.3090	R/W	MII Management Data Register	0x0000.0000
MIID_1	0xFFF0.3890	R/W	MII Management Data Register	0x0000.0000

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

MIIData [15 : 0] : MII station management data

This register contains a 16-bit data value for the MII station management function.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.18.2 MAC MII Management Data Control and Address Register

Register	Address	R/W	Description	Reset Value
MIIDA_0	0xFFF0.3094	R/W	MII Management Data Control and Address Register	0x0090.0000
MIIDA_1	0xFFF0.3894	R/W	MII Management Data Control and Address Register	0x0090.0000

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
MDCCR				MDCON	PreSP	BUSY	WR
15	14	13	12	11	10	9	8
Reserved			PHYAD				
7	6	5	4	3	2	1	0
Reserved			PHYRAD				

MDCCR [23 : 20] : MDC clock rating

The 4-bit value is to set the MDC clock period.

MDCCR [23 : 20]				MDC clock period
0	0	0	0	4 x (1/Fmclk)
0	0	0	1	6 x (1/Fmclk)
0	0	1	0	8 x (1/Fmclk)
0	0	1	1	12 x (1/Fmclk)
0	1	0	0	16 x (1/Fmclk)
0	1	0	1	20 x (1/Fmclk)
0	1	1	0	24 x (1/Fmclk)
0	1	1	1	28 x (1/Fmclk)
1	0	0	0	30 x (1/Fmclk)
1	0	0	1	32 x (1/Fmclk)
1	0	1	0	36 x (1/Fmclk)
1	0	1	1	40 x (1/Fmclk)
1	1	0	0	44 x (1/Fmclk)
1	1	0	1	48 x (1/Fmclk)
1	1	1	0	54 x (1/Fmclk)
1	1	1	1	60 x (1/Fmclk)

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Users should set the MDC clock setting to meet the PHY requirement (maximum 2.5MHz). Besides, the MCLK (HCLK) frequency ranges from 10 MHz to 150 MHz (set MDC 2.5MHz). (“System clock”/”MDC clock rating” <= 2.5)

MDCON [19] : MDC Clock On Always

If this bit was set, the MDC clock will always active. Otherwise, the MDC clock will active only when the EnMDC of MCMDR and BUSY of MIIDA are both set. In other words, the MDC clock will be turned off after the station management command finished. This bit is only for debug.

PreSP [18] : Preamble Suppress

If this bit is set, then the preamble is not sent to PHY.

BUSY [17] : Busy bit

Set this bit to start a MII management read or write-operation. The MAC controller clears this bit automatically when the operation is completed.

WR [16] : Write/Read

Set this bit for a MII management write-operation. Reset the bit for a read operation.

PHYAD [12 : 8] : PHY Address

The 5-bit address is the address of PHY device to be accessed.

PHYRAD [4 : 0] : PHY Register Address
The 5-bit address is the register address in the PHY to be accessed.

8.4.18.3 Read/Write MII Management Data

Read
(1) Write the following items to **MIIDA**
PHY address
PHY register address
MDC clock rating
Read operation bit (WR[16])
Busy bit
(2) After MAC controller clear the busy bit, the read operation is completed.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

(3) Read MII data from **MIID**

Write
(1) Write MII data to **MIID** register
(2) Write the following items to **MIIDA**
PHY address
PHY register address
MDC clock rating
Write operation bit (WR[16])
Busy bit
(3) After MAC controller clear the busy bit, the write operation is completed.

Note : EnMDC bit of MCMDR should be set to access MII management data.

8.4.18.4 PHY Registers Programming

Control Register(0x00).

Bit	Function
15	Reset
14	Loopback
13	Speed (1=100MB, 0=10MB)
12	Auto-negotiation Enable
11	Power-Down
10	Isolate
09	Restart auto-negotiation
08	Duplex Mode (1=Full, 0=Half)
07	Collision test

Status Register #1(0x01)

Bit	Function
15	100BASE-T4 capable
14	100BASE-TX full duplex capable
13	100BASE-TX half duplex capable
12	10BASE-T full duplex capable

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

11	10BASE-T half duplex capable
06	Accept management frames with preamble suppressed
05	Auto-negotiation complete
04	Remote fault
03	Auto-negotiation capable
02	Link status(1=Up, 0=Down)
01	Jabber condition detected
00	Extended register capable

Auto-negotiation Advertisement Register(0x04)

Protocol selection (00001-IEEE802.3)

Bit	Function
15	Next page available
13	Remote fault
10	Flow control support
09	100BASE-T4 support
08	100BASE-TX full duplex support
07	100BASE-TX half duplex support
06	10BASE-T full duplex support
05	10BASE-T half duplex support
04 ~ 00	Protocol selection (00001-IEEE802.3)

Status Register #2(0x11)

Current speed(10M/100M) and operation(full/half duplex) can read from this register, the exact bit position should refer to the PHY datasheet.

Example for auto-negotiation

(1) Set “auto-negotiation enable” and “restart auto-negotiation”(bits 12 and 9) of control register
(2) Wait auto-negotiation complete by reading “auto-negotiation complete”(bit 5) of status register #1 until it is set
(3) Read status register #2 to get speed and operation mode that is the result of auto-negotiation.
(4) Set speed and operation mode of MAC

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.19 DMA Receive Frame Control Register

Register	Address	R/W	Description	Reset Value
DMARFC_0	0xFFF0.30A4	R/W	DMA Receive Frame Control Register	0x0000.0800
DMARFC_1	0xFFF0.38A4	R/W	DMA Receive Frame Control Register	0x0000.0800

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

RXMS [15 : 0] : DMA receive frame maximum size

This value controls the maximum bytes for a received frame can be saved to system memory. If the received frame size exceeds the value specified in this register and the EnDFO is set, an error interrupt is reported. The default maximum size is 2K bytes.

8.4.20 Transmit Start Demand Register

Register	Address	R/W	Description	Reset Value
TSDR_0	0xFFF0.30A8	W	Transmit Start Demand Register	Undefined
TSDR_1	0xFFF0.38A8	W	Transmit Start Demand Register	Undefined

31	30	29	28	27	26	25	24
TSDR							
23	22	21	20	19	18	17	16
TSDR							
15	14	13	12	11	10	9	8
TSDR							

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

7	6	5	4	3	2	1	0
TSDR							

TSDR [31 : 0] : Transmit Start Demand Register

While the transmit descriptor is unavailable, the Tx engine will be halt. The software driver needs to issue a write command with any data to Transmit Start Demand register to restart the Tx operation. Only while Tx engine was halted, the write command to Transmit Start Demand register take affect on the Tx operation.

8.4.21 Receive Start Demand Register

Register	Address	R/W	Description	Reset Value
RSDR_0	0xFFF0.30AC	W	Receive Start Demand Register	Undefined
RSDR_1	0xFFF0.38AC	W	Receive Start Demand Register	Undefined

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

RSDR [31 : 0] : Receive Start Demand Register

While the receive descriptor is unavailable, the Rx engine will be halted. The software driver needs to issue a write command with any data to Receive Start Demand register to restart the Rx operation. Only while Rx engine was halted, the write command to Receive Start Demand register can take affect on the Rx operation.

8.4.22 FIFO Threshold Adjustment Register

Register	Address	R/W	Description	Reset Value
FIFOTHD_0	0xFFF0.30B0	R/W	FIFO Threshold Adjustment Register	0x0000.0101
FIFOTHD_1	0xFFF0.38B0	R/W	FIFO Threshold Adjustment Register	0x0000.0101

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

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

Blength [21 : 20] : DMA burst length

Blength[21 : 20]	Words(32 bits)
00b	4
01b	8
10b	16
11b	16

SWR [16] : software reset

Software reset. After a reset is complete, the MAC controller will clear the SWR reset bit.

TxTHD [9 : 8] : Transmit FIFO Lower threshold Register

TxTHD[9 : 8]	Threshold Value
00b	forbidden
01b	32 bytes (i.e. high threshold 64 bytes)
10b	64 bytes (i.e. high threshold 128 bytes)
11b	96 bytes (i.e. high threshold 192 bytes)

This value controls the transmit FIFO low threshold. If transmitting packet number is less than the setting value, Tx DMA will request the arbiter to get data from memory.

RxTHD [1 : 0] : Receive FIFO Upper threshold Register

RxTHD[1 : 0]	Threshold Value

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

00b	Depend on the burst length setting
01b	64 bytes (i.e. low threshold 32 bytes)
10b	128 bytes (i.e. low threshold 64 bytes)
11b	192 bytes (i.e. low threshold 96 bytes)

This value controls the receive FIFO high threshold. If receiving packet number is greater than the setting value, Rx DMA will request the arbiter to send data into memory.

8.4.23 DMA Receive Frame Status Register

Register	Address	R/W	Description	Reset Value
DMARFS_0	0xFFF0.30C8	R/W	DMA Receive Frame Status Register	0x0000.0000
DMARFS_1	0xFFF0.38C8	R/W	DMA Receive Frame Status Register	0x0000.0000

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

RXFS [15 : 0] : DMA receive frame length

When an early notification interrupt occurs, the frame length/type field contains the size of the frame that is currently being received, and stores in **RXFS**.

To save space in the frame memory buffer, users can determine the current frame length by :

1. Enable the early notification interrupt (set **EnDEN** bit in **MIEN**)
2. Read the RXFS field when the interrupt occurs

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Note : all the DMARFS bits are cleared by write ones; writing zeros has no effect.

8.4.24 MAC Missed Packet Count register

The value indicates the number of packets that were discarded due to various types of errors. Users can read the MPCNT to get current missed packet counter value and clears the register (read clear). It is the responsibility of software to maintain a global count. However, users can write the MPCNT to set the initial value of counter overflow and start to count. The counter overflow value ranges from 0x0000 to 0xFFFF (default value : 0x7FFF). It sets the corresponding bit (MMP) in the MISTA and generates an interrupt if overflow is occurred and the corresponding interrupt enable bit is set.

This counter indicates the following kinds of error :

Dribbling Bits error count (AECnt) :

The number of packets received with alignment errors. The counter will be increment at the end of packet reception if the MISTA indicates the alignment errors.

Frame discarded error count (RXFDCnt) :

The number of packets discarded by MAC because of receive FIFO overflows, or because the RxON bit is cleared. This count does not include the number of packets rejected by the CAM.

CRC error count (CECnt) :

The number of packets received with a CRC error. The counter will be increment at the end of packet reception if the MISTA indicates the CRC errors.

Register	Address	R/W	Description	Reset Value
MPCNT_0	0xFFF0.3098	R/W	Missed Packet counter register	0x0000.7FFF
MPCNT_1	0xFFF0.3898	R/W	Missed Packet counter register	0x0000.7FFF

31	30	29	28	27	26	25	24
Reserved							

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
MPCNT							
7	6	5	4	3	2	1	0
MPCNT							

MPCNT [15 : 0] : MAC Missed Packet Count

It indicates the number of packets that were discarded due to various types of errors.

8.4.25 Control Frames

8.4.25.1 Receive Control Pause Frame

(1) Set ACP bit in MCMDR
(2) The Multicast address “01-80-c2-00-00-01” should fill to CAM if AMP not set.
(3) Set EnCFR in MIEN if want to handle control frame receive interrupt

8.4.25.2 Send Control Pause Frame

(1) Fill the destination MAC address to CAM#13.
(2) Fill the source MAC address to CAM#14.
(3) Fill length/type(0x8808), opcode(0x0001) and operand(timeslot) to CAM#15.
(4) Set SDPZ bit in MCMDR.
(5) Wait control pause frame transmission complete by reading SDPZ bit until it is 0.

8.4.25.3 MAC Received Pause Count Register

Register	Address	R/W	Description	Reset Value
MRPC_0	0xFFF0.30BC	R	MAC Receive Pause count register	0x0000.0000
MRPC_1	0xFFF0.38BC	R	MAC Receive Pause count register	0x0000.0000

31	30	29	28	27	26	25	24

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
MRPC							
7	6	5	4	3	2	1	0
MRPC							

MRPC [15 : 0] : MAC Received Pause Count Register

The count value indicates the number of time slots the transmitter was paused due to the receipt of control pause operation packets from the MAC.

8.4.25.4 MAC Received Pause Current Count Register

The received pause current count register, **MRPCC**, stores the current value of the 16-bit received pause counter. It is read only.

Register	Address	R/W	Description	Reset Value
MRPCC_0	0xFFF0.30C0	R	MAC Receive Pause Current Count register	0x0000.0000
MRPCC_1	0xFFF0.38C0	R	MAC Receive Pause Current Count register	0x0000.0000

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

MRPC [15 : 0] : MAC Received Pause Count Register

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

The count value indicates the number of time slots the receiver was paused due to the receipt of control pause operation packets from the MAC. (Finally will count down to 0)

8.4.25.5 MAC Remote Pause Count Register

The remote pause count register, MREPC, stores the current value of the remote pause counter. It is read only.

Register	Address	R/W	Description	Reset Value
MREPC_0	0xFFF0.30C4	R	MAC Remote pause count register	0x0000.0000
MREPC_1	0xFFF0.38C4	R	MAC Remote pause count register	0x0000.0000

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

MREPC [15 : 0] : MAC Remote Pause Count Register

The count value indicates the number of time slots that a remote MAC was paused, as a result of its sent control pause operation packets. (Finally will count down to 0)

8.4.26 Descriptor Status Registers

8.4.26.1 Current Transmit Descriptor Start Address Register

Register	Address	R/W	Description	Reset Value
CTXDSA_0	0xFFF0.30CC	R	Current Transmit Descriptor Start Address Register	0x0000.0000
CTXDSA_1	0xFFF0.38CC	R	Current Transmit Descriptor Start Address Register	0x0000.0000

31	30	29	28	27	26	25	24

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

CTXDSA							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
CTXDSA							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
CTXDSA							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CTXDSA							

CTXDSA [31 : 0] : Current Transmit Descriptor Start Address

This register reports the start address of the current transmit descriptor used by EMC.

8.4.26.2 Current Transmit Buffer Start Address Register

Register	Address	R/W	Description	Reset Value
CTXBSA_0	0xFFF0.30D0	R	Current Transmit Buffer Start Address Register	0x0000.0000
CTXBSA_1	0xFFF0.38D0	R	Current Transmit Buffer Start Address Register	0x0000.0000

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

CTXBSA [31 : 0] : Current Transmit Buffer Start Address

This register reports the start address of the current transmit buffer used by EMC.

8.4.26.3 Current Receive Descriptor Start Address Register

Register	Address	R/W	Description	Reset Value
CRXDSA_0	0xFFF0.30D4	R	Current Receive Descriptor Start Address Register	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

CRXDSA_1	0xFFF0.38D4	R	Current Receive Descriptor Start Address Register	0x0000.0000

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

CRXDSA [31 : 0] : Current Receive Descriptor Start Address

This register reports the start address of the current receive descriptor used by EMC.

8.4.26.4 Current Receive Buffer Start Address Register

Register	Address	R/W	Description	Reset Value
CRXBSA_0	0xFFF0.30D8	R	Current Receive Buffer Start Address Register	0x0000.0000
CRXBSA_1	0xFFF0.38D8	R	Current Receive Buffer Start Address Register	0x0000.0000

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

CRXBSA [31 : 0] : Current Receive Buffer Start Address

This register reports the start address of the current receive buffer used by EMC.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.27 Interrupts Setting

8.4.27.1 Interrupt Sources

Channel	Name	Mode	Source
13	EMCTXINT0	Positive Level	EMC TX Interrupt 0
14	EMCTXINT1	Positive Level	EMC TX Interrupt 1
15	EMCRXINT0	Positive Level	EMC RX Interrupt 0
16	EMCRXINT1	Positive Level	EMC RX Interrupt 1

8.4.27.2 AIC Source Control Registers

Register	Address	R/W	Description	Reset Value
AIC_SCR13	0xFFF8.2034	R/W	Source Control Register 13	0x0000.0047
AIC_SCR14	0xFFF8.2038	R/W	Source Control Register 14	0x0000.0047
AIC_SCR15	0xFFF8.203C	R/W	Source Control Register 15	0x0000.0047
AIC_SCR16	0xFFF8.2040	R/W	Source Control Register 16	0x0000.0047

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

SRCTYPE [7 : 6] : Interrupt Source Type

Whether an interrupt source is considered active or not by the AIC is subject to the settings of this field. Interrupt sources other than nIRQ0, nIRQ1, nIRQ2, nIRQ3, should be configured as level sensitive during normal operation unless in the testing situation.

SRCTYPE [7 : 6]		Interrupt Source Type
0	0	Low-level Sensitive
0	1	High-level Sensitive
1	0	Negative-edge Triggered
1	1	Positive-edge Triggered

PRIORITY [2 : 0] : Priority Level

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Every interrupt source must be assigned a priority level during initiation. Among them, priority level 0 has the highest priority and priority level 7 the lowest. Interrupt sources with priority level 0 are promoted to FIQ. Interrupt sources with priority level other than 0 belong to IRQ. For interrupt sources of the same priority level, that located in the lower channel number has higher priority.

8.4.27.3 AIC Mask Enable Command Register

Register	Address	R/W	Description	Reset Value
AIC_MECR	0xFFF8.2120	W	Mask Enable Command Register	Undefined

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED					MEC18	MEC17	MEC16
15	14	13	12	11	10	9	8
MEC15	MEC14	MEC13	MEC12	MEC11	MEC10	MEC9	MEC8
7	6	5	4	3	2	1	0
MEC7	MEC6	MEC5	MEC4	MEC3	MEC2	MEC1	RESERVED

MEC *x* : Mask Enable Command

0 = No effect

1 = Enables the corresponding interrupt channel

8.4.27.4 AIC Mask Disable Command Register

Register	Address	R/W	Description	Reset Value
AIC_MDCR	0xFFF8.2124	W	Mask Disable Command Register	Undefined

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED					MDC18	MDC17	MDC16
15	14	13	12	11	10	9	8
MDC15	MDC14	MDC13	MDC12	MDC11	MDC10	MDC9	MDC8
7	6	5	4	3	2	1	0
MDC7	MDC6	MDC5	MDC4	MDC3	MDC2	MDC1	RESERVED

MDC *x* : Mask Disable Command

0 = No effect

1 = Disables the corresponding interrupt channel

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.27.5 Initialize Interrupts

(1) Set interrupt source type(high level sensitive) and priority level to AIC_SCR registers.
(2) Register interrupt service routine to associative interrupts.
(3) Enable interrupt by set bits in AIC_MECR register.

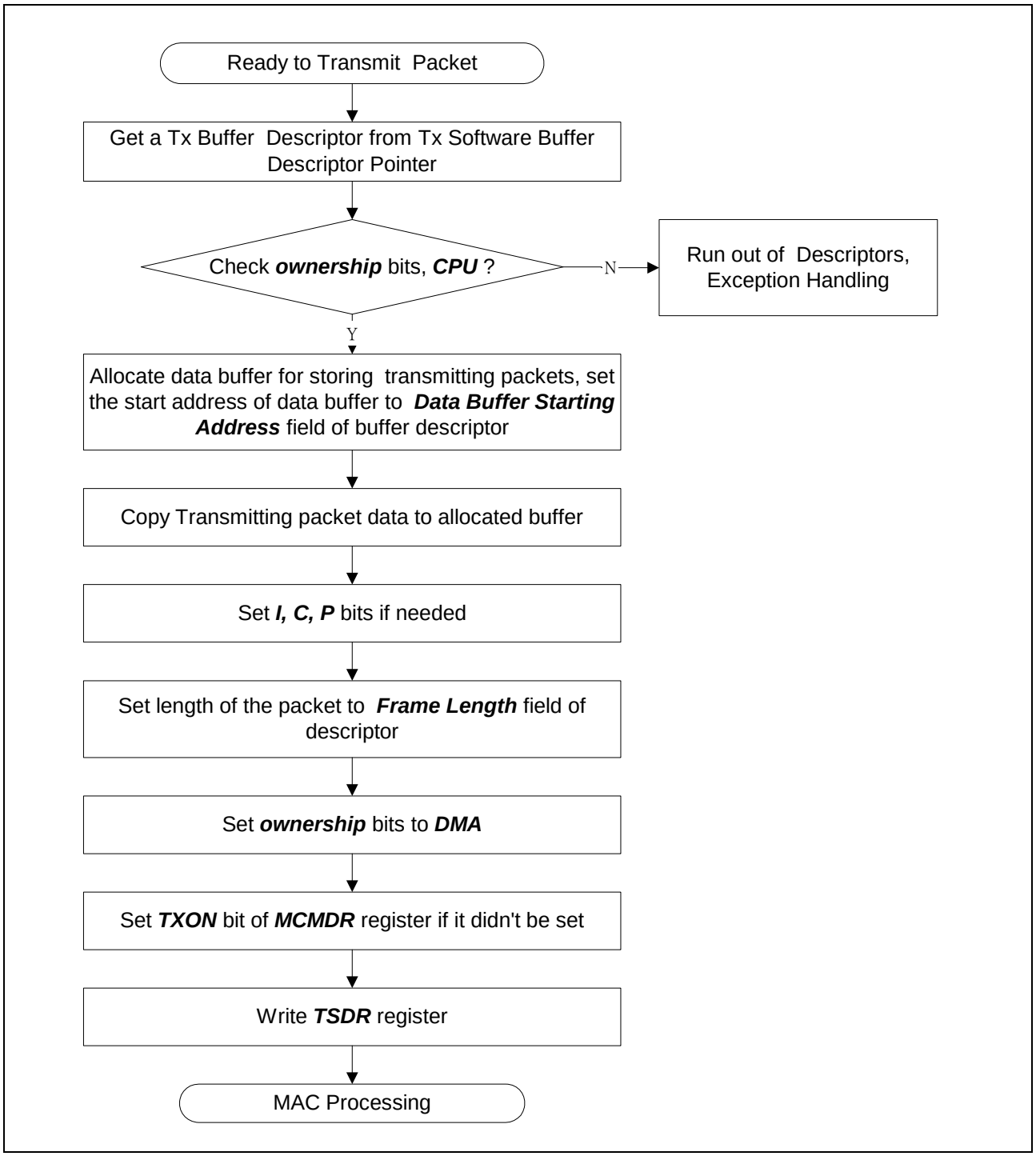
8.4.28 Packet Processing

8.4.28.1 Packet Transmission

(1) Get Tx buffer descriptor from Tx software pointer.
(2) Check ownership of (1), do nothing if ownership is DMA.
(3) Allocate data buffer and set start address to data buffer starting address field of (1).
(4) Copy packet data to data buffer.
(5) Set I,C,P bits if need.
(6) Set packet length to frame length field of (1).
(7) Set ownership to DMA.
(8) Set TXON bit of MCMDR register if it is not set.
(9) Write TSDR register.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 8-5 Packet Transmission Flow



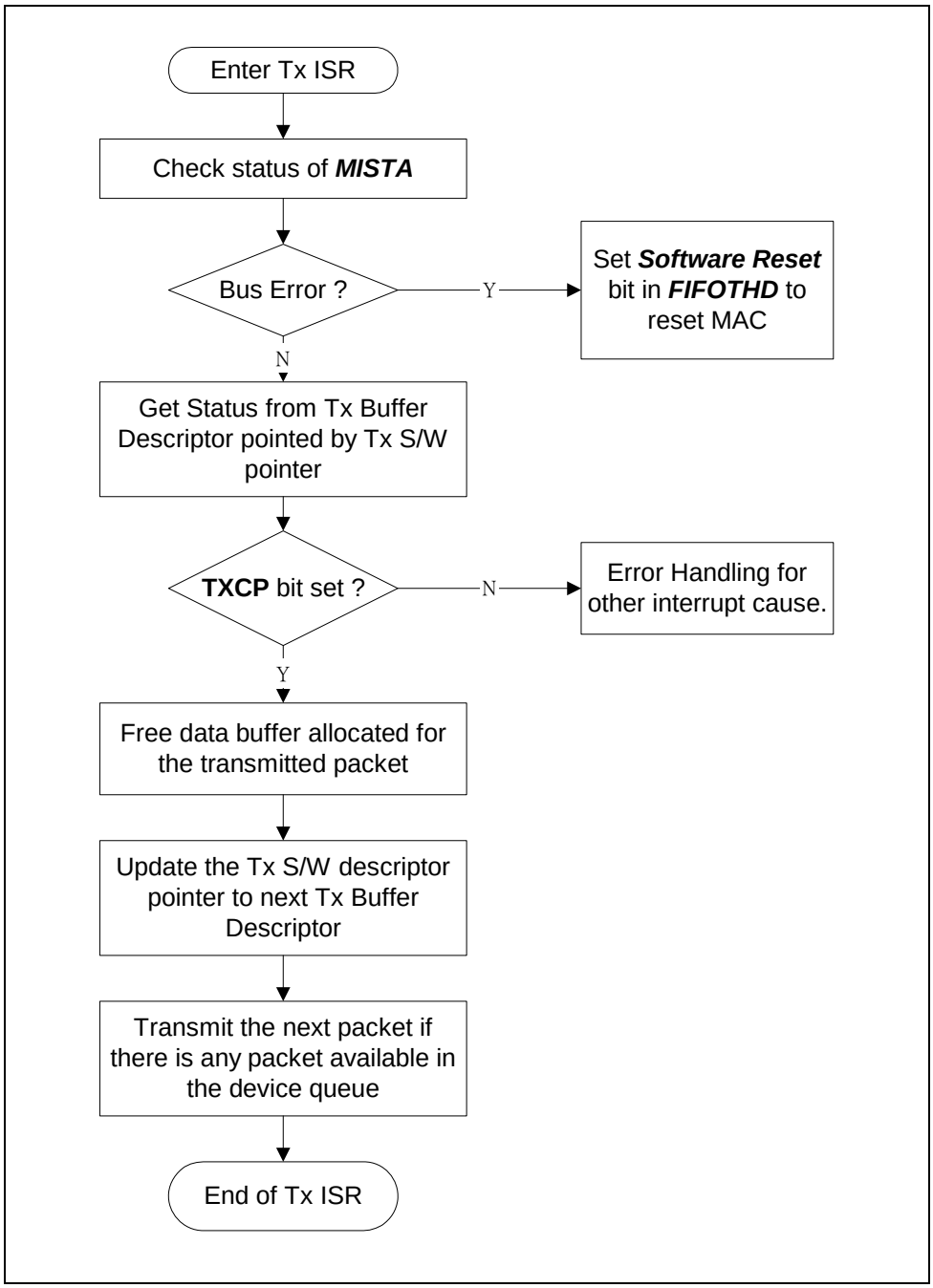
The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.28.2 Tx Interrupt Service Routine

(1) Get and check status in MISTA.
(2) Set software reset bit in FIFOTHD and re-initialize MAC if bus error occur. Do the following step if no error occur.
(3) Get status from the descriptor of Tx software pointer. Do the following steps if TXCP bit is set.
(4) Free data buffer allocated to this descriptor.
(5) Set the next descriptor to Tx software pointer.
(6) Transmit the next packet if there is packet available in device queue.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 8-6 Tx Interrupt Service Routine Flow



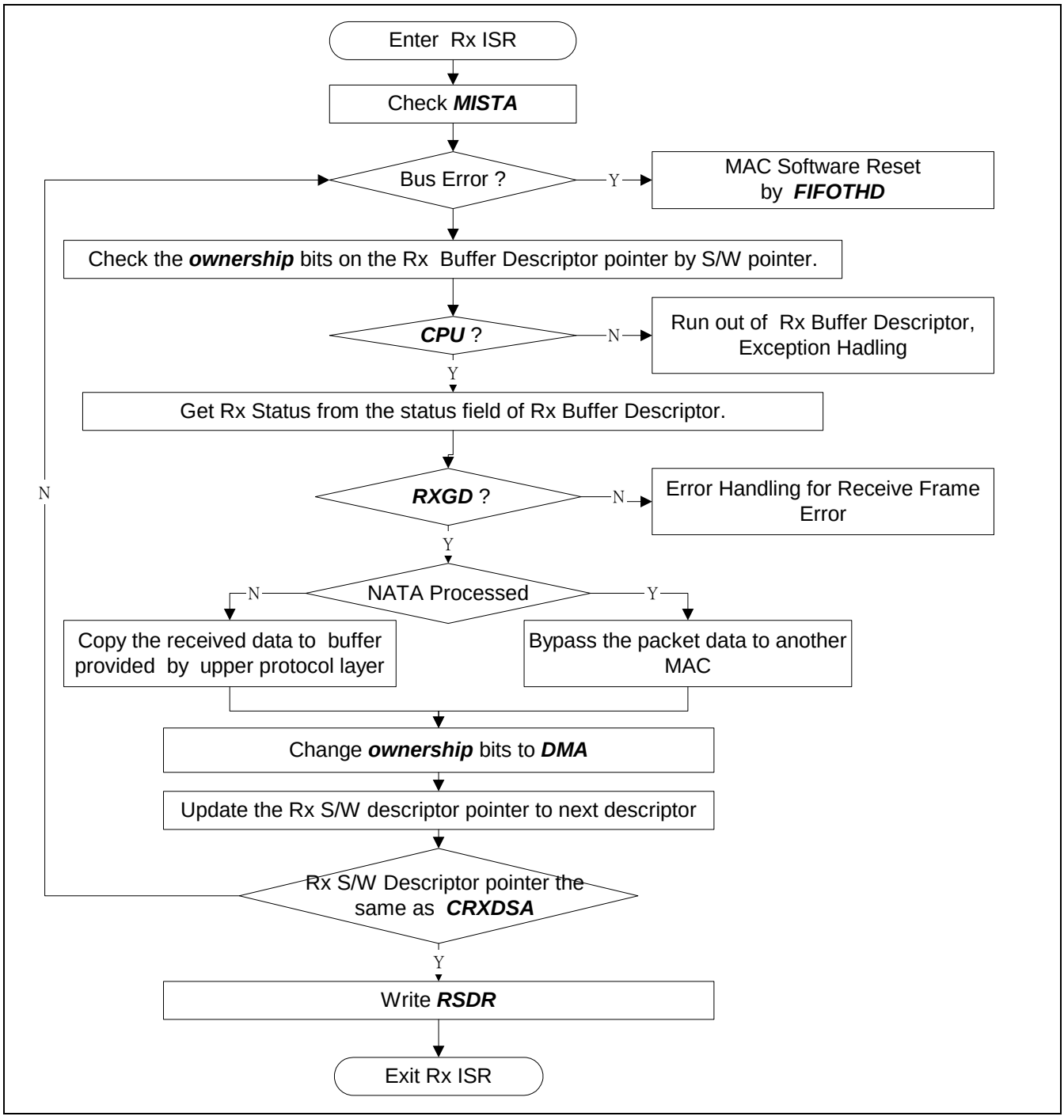
The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.28.3 Rx Interrupt Service Routine

(1) Get and check status in MISTA.
(2) Set software reset bit in FIFOTHD and re-initialize MAC if bus error occur. Do the following step if no error occur.
(3) Get ownership from the descriptor of Rx software pointer. Do the following step if ownership is CPU.
(4) Get status from the descriptor of Rx software pointer. Do the following steps if RXGD bit is set.
(5) Copy receive packet to net buffer of upper layer if the packet was not processed by NATA. Bypass the packet to another MAC if it is processed by NATA.
(6) Change ownership to DMA.
(7) Set the next descriptor to Rx software pointer.
(8) Re-start from step (3) if descriptor of Rx software pointer is not the same as the one of CRXDSA register.
(9) Write RSDR register.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 8-7 Rx Interrupt Service Routine



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.4.28.4 Long Packets

ALP bit of MCMDR is set

(1) Long packet will be received.
(2) PTLE bit in Rx status of descriptor will be set
(3) PTLE interrupt will occur if EnPTLE bit of MIEN is set.

ALP bit of MCMDR is not set

(1) Long packet will not be received.
(2) PTLE interrupt will occur if EnPTLE bit of MIEN is set.

8.4.28.5 Runt Packets

ARP bit of MCMDR is set

(1) Runt packet will be received.
(2) RP bit in Rx status of descriptor will be set
(3) RP interrupt will occur if EnRP bit of MIEN is set.

ARP bit of MCMDR is not set

(1) Runt packet will not be received.
(2) RP interrupt will occur if EnRP bit of MIEN is set.

8.4.28.6 CRCE&ALIE Packets

AEP bit of MCMDR is set

(1) CRC error or alignment error packet will be received.
(2) CRCE or ALIE bit in Rx status of descriptor will be set
(3) CRCE or ALIE interrupt will occur if EnCRCE or EnALIE bit of MIEN is set.

ALP bit of MCMDR is not set

(1) CRC error or alignment error packet will not be received.
(2) CRCE or ALIE interrupt will occur if EnCRCE or EnALIE bit of MIEN is set.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

8.5 V. Sample Code

```
#define VPint   *(volatile unsigned int *)
#define VPshort *(volatile unsigned short *)
#define VPchar  *(volatile unsigned char *)

// Special Register Start Address After System Reset
#define Base_Addr 0xfff00000

// Advanced Interrupt Controller Registers
#define AIC_SCR_EMCTX0 (VPint(Base_Addr+0x00082034))
#define AIC_SCR_EMCTX1 (VPint(Base_Addr+0x00082038))
#define AIC_SCR_EMCRX0 (VPint(Base_Addr+0x0008203c))
#define AIC_SCR_EMCRX1 (VPint(Base_Addr+0x00082040))
#define AIC_MECR       (VPint(Base_Addr+0x00082120))
#define AIC_MDCR       (VPint(Base_Addr+0x00082124))
// MAC Interrupt Sources
#define EMCTXINT0 13
#define EMCTXINT1 14
#define EMCRXINT0 15
#define EMCRXINT1 16
// Routines to Enable/Disable Interrupts
#define Enable_Int(n)     AIC_MECR = (1<<(n))
#define Disable_Int(n)    AIC_MDCR = (1<<(n))
#define Enable_Int_All()  AIC_MECR = 0xffff
#define Disable_Int_All() AIC_MDCR = 0xffff

// EMC 0 Control Registers
#define CAMCMR_0       (VPint(Base_Addr+0x3000))     // CAM Registers
#define CAMEN_0        (VPint(Base_Addr+0x3004))
#define CAM0M_Base_0   Base_Addr+0x3008
#define CAM0L_Base_0   Base_Addr+0x300c
#define CAMxM_Reg_0(x) (VPint(CAM0M_Base_0+x*0x8))
#define CAMxL_Reg_0(x) (VPint(CAM0L_Base_0+x*0x8))
#define MIEN_0         (VPint(Base_Addr+0x3088))     // MAC Registers
#define MCMDR_0        (VPint(Base_Addr+0x308c))
#define MIID_0         (VPint(Base_Addr+0x3090))
#define MIIDA_0        (VPint(Base_Addr+0x3094))
#define MPCNT_0        (VPint(Base_Addr+0x3098))
#define TXDLSA_0       (VPint(Base_Addr+0x309c))     // DMA Registers
#define RXDLSA_0       (VPint(Base_Addr+0x30a0))
#define DMARFC_0       (VPint(Base_Addr+0x30a4))
#define TSDR_0         (VPint(Base_Addr+0x30a8))
#define RSDR_0         (VPint(Base_Addr+0x30ac))
#define FIFOTHD_0      (VPint(Base_Addr+0x30b0))     // Test Registers
// EMC 0 Status Registers
#define MISTA_0        (VPint(Base_Addr+0x30b4))     // MAC Registers
#define MGSTA_0        (VPint(Base_Addr+0x30b8))
#define MRPC_0         (VPint(Base_Addr+0x30bc))
#define MRPCC_0        (VPint(Base_Addr+0x30c0))
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define MREPC_0       (VPint(Base_Addr+0x30c4))
#define DMARFS_0      (VPint(Base_Addr+0x30c8))    // DMA Registers
#define CTXDSA_0      (VPint(Base_Addr+0x30cc))
#define CTXBSA_0      (VPint(Base_Addr+0x30d0))
#define CRXDSA_0      (VPint(Base_Addr+0x30d4))
#define CRXBSA_0      (VPint(Base_Addr+0x30d8))

// EMC 1 Control Registers
#define CAMCMR_1      (VPint(Base_Addr+0x3800))    // CAM Registers
#define CAMEN_1       (VPint(Base_Addr+0x3804))
#define CAM0M_Base_1  Base_Addr+0x3808
#define CAM0L_Base_1  Base_Addr+0x380c
#define CAMxM_Reg_1(x) (VPint(CAM0M_Base_1+x*0x8))
#define CAMxL_Reg_1(x) (VPint(CAM0L_Base_1+x*0x8))
#define MIEN_1        (VPint(Base_Addr+0x3888))    // MAC Registers
#define MCMDR_1       (VPint(Base_Addr+0x388c))
#define MIID_1        (VPint(Base_Addr+0x3890))
#define MIIDA_1       (VPint(Base_Addr+0x3894))
#define MPCNT_1       (VPint(Base_Addr+0x3898))
#define TXDLSA_1      (VPint(Base_Addr+0x389c))    // DMA Registers
#define RXDLSA_1      (VPint(Base_Addr+0x38a0))
#define DMARFC_1      (VPint(Base_Addr+0x38a4))
#define TSDR_1        (VPint(Base_Addr+0x38a8))
#define RSDR_1        (VPint(Base_Addr+0x38ac))
#define FIFOTHD_1     (VPint(Base_Addr+0x38b0))    // Test Registers
// EMC 1 Status Registers
#define MISTA_1       (VPint(Base_Addr+0x38b4))    // MAC Registers
#define MGSTA_1       (VPint(Base_Addr+0x38b8))
#define MRPC_1        (VPint(Base_Addr+0x38bc))
#define MRPCC_1       (VPint(Base_Addr+0x38c0))
#define MREPC_1       (VPint(Base_Addr+0x38c4))
#define DMARFS_1      (VPint(Base_Addr+0x38c8))    // DMA Registers
#define CTXDSA_1      (VPint(Base_Addr+0x38cc))
#define CTXBSA_1      (VPint(Base_Addr+0x38d0))
#define CRXDSA_1      (VPint(Base_Addr+0x38d4))
#define CRXBSA_1      (VPint(Base_Addr+0x38d8))

// CAM Command Register(CAMCMR)
#define CAM_AUP  0x0001  // Accept Packets with Unicast Address
#define CAM_AMP  0x0002  // Accept Packets with Multicast Address
#define CAM_ABP  0x0004  // Accept Packets with Broadcast Address
#define CAM_CCAM 0x0008  // 0 :  Accept Packets CAM Recognizes and Reject Others
                         // 1 :  Reject Packets CAM Recognizes and Accept Others
#define CAM_ECMP 0x0010  // Enable CAM Compare

// MAC Interrupt Enable Register(MIEN)
#define EnRXINTR 0x00000001  // Enable Interrupt on Receive Interrupt
#define EnCRCE   0x00000002  // Enable CRC Error Interrupt
#define EnRXOV   0x00000004  // Enable Receive FIFO Overflow Interrupt
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define EnPTLE   0x00000008  // Enable Packet Too Long Interrupt
#define EnRXGD   0x00000010  // Enable Receive Good Interrupt
#define EnALIE   0x00000020  // Enable Alignment Error Interrupt
#define EnRP     0x00000040  // Enable Runt Packet on Receive Interrupt
#define EnMMP    0x00000080  // Enable More Missed Packets Interrupt
#define EnDFO    0x00000100  // Enable DMA receive frame over maximum size Interrupt
#define EnDEN    0x00000200  // Enable DMA early notification Interrupt
#define EnRDU    0x00000400  // Enable Receive Descriptor Unavailable Interrupt
#define EnRxBErr 0x00000800  // Enable Receive Bus ERROR interrupt
#define EnNATOK  0x00001000  // Enable NAT Processing OK Interrupt
#define EnNATErr 0x00002000  // Enable NAT Processing Error Interrupt
#define EnCFR    0x00004000  // Enable Control Frame Receive Interrupt
#define EnTXINTR 0x00010000  // Enable Interrupt on Transmit Interrupt
#define EnTXEMP  0x00020000  // Enable Transmit FIFO Empty Interrupt
#define EnTXCP   0x00040000  // Enable Transmit Completion Interrupt
#define EnEXDEF  0x00080000  // Enable Defer Interrupt
#define EnNCS    0x00100000  // Enable No Carrier Sense Interrupt
#define EnTXABT  0x00200000  // Enable Transmit Abort Interrupt
#define EnLC     0x00400000  // Enable Late Collision Interrupt
#define EnTDU    0x00800000  // Enable Transmit Descriptor Unavailable Interrupt
#define EnTxBErr 0x01000000  // Enable Transmit Bus ERROR Interrupt

// MAC Command Register(MCMDR)
#define MCMDR_RXON    0x00000001  // Receive ON
#define MCMDR_ALP     0x00000002  // Accept Long Packet
#define MCMDR_ARP     0x00000004  // Accept Runt Packet
#define MCMDR_ACP     0x00000008  // Accept Control Packet
#define MCMDR_AEP     0x00000010  // Accept Error Packet
#define MCMDR_SPCRC   0x00000020  // Accept Strip CRC Value
#define MCMDR_TXON    0x00000100  // Transmit On
#define MCMDR_NDEF    0x00000200  // No defer
#define MCMDR_SDPZ    0x00010000  // Send Pause
#define MCMDR_EnSQE   0x00020000  // Enable SQE test
#define MCMDR_FDUP    0x00040000  // Full Duplex
#define MCMDR_EnMDC   0x00080000  // Enable MDC signal
#define MCMDR_OPMOD   0x00100000  // Operation Mode
#define MCMDR_LBK     0x00200000  // Loop Back
#define MCMDR_EnRMII  0x00400000  // Enable RMII
#define MCMDR_LPCS    0x00800000  // Low Pin Count Package Switch
#define MCMDR_LAN     0x01000000  // LAN Port Setting Mode
#define MCMDR_EnTMode 0x02000000  // Enable Test Mode

// MAC MII Management Data Control and Address Register(MIIDA)
#define MDCCR    0x00300000  // MDC clock rating
#define PHYAD    0x00000100  // PHY Address
#define PHYWR    0x00010000  // Write Operation
#define PHYBUSY  0x00020000  // Busy Bit
#define PHYPreSP 0x00040000  // Preamble Suppress
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
// FIFO Threshold Adjustment Register(FIFOTHD)
#define SWR        0x00010000  // Software Reset
#define TxTHD_1    0x00000100  // 1/4 Transmit FIFO Threshold
#define TxTHD_2    0x00000200  // 2/4 Transmit FIFO Threshold
#define TxTHD_3    0x00000300  // 3/4 Transmit FIFO Threshold
#define RxTHD_1    0x00000001  // 1/4 Receive FIFO Threshold
#define RxTHD_2    0x00000002  // 2/4 Receive FIFO Threshold
#define RxTHD_3    0x00000003  // 3/4 Receive FIFO Threshold
#define Blength_8  0x00100000  // DMA burst length 8 beats
#define Blength_12 0x00200000  // DMA burst length 12 beats
#define Blength_16 0x00300000  // DMA burst length 16 beats

// MAC Interrupt Status Register(MISTA)
#define MISTA_RXINTR 0x00000001  // Interrupt on Receive
#define MISTA_CRCE   0x00000002  // CRC Error
#define MISTA_RXOV   0x00000004  // Receive FIFO Overflow error
#define MISTA_PTLE   0x00000008  // Packet Too Long Error
#define MISTA_RXGD   0x00000010  // Receive Good
#define MISTA_ALIE   0x00000020  // Alignment Error
#define MISTA_RP     0x00000040  // Runt Packet
#define MISTA_MMP    0x00000080  // More Missed Packets than miss rolling over counter flag
#define MISTA_DFOI   0x00000100  // DMA receive frame over maximum size interrupt
#define MISTA_DENI   0x00000200  // DMA early notification interrupt
#define MISTA_RDU    0x00000400  // Receive Descriptor Unavailable interrupt
#define MISTA_RxBErr 0x00000800  // Receive Bus Error interrupt
#define MISTA_NATOK  0x00001000  // NAT Processing OK
#define MISTA_NATErr 0x00002000  // NAT Processing Error
#define MISTA_CFR    0x00004000  // Control Frame Receive
#define MISTA_TXINTR 0x00010000  // Interrupt on Transmit
#define MISTA_TXEMP  0x00020000  // Transmit FIFO Empty
#define MISTA_TXCP   0x00040000  // Transmit Completion
#define MISTA_EXDEF  0x00080000  // Defer
#define MISTA_NCS    0x00100000  // No Carrier Sense
#define MISTA_TXABT  0x00200000  // Transmit Abort
#define MISTA_LC     0x00400000  // Late Collision
#define MISTA_TDU    0x00800000  // Transmit Descriptor Unavailable interrupt
#define MISTA_TxBErr 0x01000000  // Transmit Bus Error interrupt

// MAC General Status Register(MGSTA)
#define MGSTA_CFR  0x00000001  // Control Frame Received
#define MGSTA_RXHA 0x00000002  // Reception Halted
#define MGSTA_DEF  0x00000010  // Deferred transmission
#define MGSTA_PAU  0x00000020  // Pause Bit
#define MGSTA_SQE  0x00000040  // Signal Quality Error
#define MGSTA_TXHA 0x00000080  // Transmission Halted

// PHY(DM9161) Register Description
#define PHY_CNTL_REG    0x00
#define PHY_STATUS_REG  0x01
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define PHY_ID1_REG     0x02
#define PHY_ID2_REG     0x03
#define PHY_ANA_REG     0x04
#define PHY_ANLPA_REG   0x05
#define PHY_ANE_REG     0x06

#define PHY_DSC_REG     0x10
#define PHY_DSCS_REG    0x11
#define PHY_10BTCS_REG  0x12
#define PHY_SINT_REG    0x15
#define PHY_SREC_REG    0x16
#define PHY_DISC_REG    0x17

//PHY Control Register
#define RESET_PHY       1 << 15
#define ENABLE_LOOPBACK 1 << 14
#define DR_100MB        1 << 13
#define ENABLE_AN       1 << 12
#define PHY_MAC_ISOLATE 1 << 10
#define RESTART_AN      1 << 9
#define PHY_FULLDUPLEX  1 << 8
#define PHY_COL_TEST    1 << 7

// PHY Status Register
#define AN_COMPLETE     1 << 5

// PHY Auto-negotiation Advertisement Register
#define DR100_TX_FULL   1 << 8
#define DR100_TX_HALF   1 << 7
#define DR10_TX_FULL    1 << 6
#define DR10_TX_HALF    1 << 5
#define IEEE_802_3_CSMA_CD   1

// Ethernet Definitions
#define HW_Address_Size    6  // Length of Ethernet Addresses
#define TYPE_LENGTH        2  // Length of Header Type/Length Field
#define MTU_Size        1500  // MAX Transmission Unit, excluding media
                              //  header length for ethernet

// Ethernet header size :   dest addr, source addr, Type/Length
#define ETH_HeaderSize ((HW_Address_Size * 2) + TYPE_LENGTH)

#define MaxRxFrameSize 1518  // Max Size Rx Packet
#define MaxTxFrameSize 1518  // Max Size Tx Packet

#define MaxRxFrameDescriptors 32   // Max number of Rx Frame Descriptors
#define MaxTxFrameDescriptors 16   // Max number of Tx Frame Descriptors
#define MaxRxFrameData        32   // Max number of Rx Frame Data
#define MaxTxFrameData        16   // Max number of Tx Frame Data
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
// RX Frame Descriptor's Owner bit
#define RXfOwnership_DMA 0x80000000  // 10 = DMA
#define RXfOwnership_NAT 0xc0000000  // 11 = NAT
#define RXfOwnership_CPU 0x3fffffff  // 00 = CPU

// TX Frame Descriptor's Owner bit
#define TXfOwnership_DMA 0x80000000  // 1 = DMA
#define TXfOwnership_CPU 0x7fffffff  // 0 = CPU

// Rx Frame Descriptor Status
#define RXFD_Hit     0x2000  // current packet is hit with NAT entry table
#define RXFD_IPHit   0x1000  // current packet is hit on IP address
#define RXFD_PortHit 0x0800  // current packet is hit on Port Number
#define RXFD_Inverse 0x0400  // current hit entry is setting on inverse mode
#define RXFD_NATFSH  0x0200  // NAT Processing Finish
#define RXFD_NOP     0x0100  // NAT No Operation
#define RXFD_RP      0x0040  // Runt Packet
#define RXFD_ALIE    0x0020  // Alignment Error
#define RXFD_RXGD    0x0010  // Receiving Good packet received
#define RXFD_PTLE    0x0008  // Packet Too Long Error
#define RXFD_CRCE    0x0002  // CRC Error
#define RXFD_RXINTR  0x0001  // Interrupt on receive

// Rx Frame Descriptor NAT Information
#define RXFD_NAT_URG     0x20000000  // TCP status(URG)
#define RXFD_NAT_ACK     0x10000000  // TCP status(ACK)
#define RXFD_NAT_PSH     0x08000000  // TCP status(PSH)
#define RXFD_NAT_RST     0x04000000  // TCP status(RST)
#define RXFD_NAT_SYN     0x02000000  // TCP status(SYN)
#define RXFD_NAT_FIN     0x01000000  // TCP status(FIN)
#define RXFD_NAT_UCKERR  0x00400000  // TCP/UCKS Error
#define RXFD_NAT_TUERR   0x00200000  // TCP/UDP Error
#define RXFD_NAT_NHERR   0x00100000  // No Hit Error
#define RXFD_NAT_PPPCaps 0x00000040  // PPPoE datagram encapsulated
#define RXFD_NAT_PPPoE   0x00000020  // PPPoE protocol
#define RXFD_NAT_UCKS    0x00000010  // UDP protocol with skip checksum replacement
#define RXFD_NAT_UDP     0x00000008  // apply UDP protocol
#define RXFD_NAT_TCP     0x00000004  // apply TCP protocol
#define RXFD_NAT_LAN     0x00000002  // internal (LAN) port gets hit
#define RXFD_NAT_Hit     0x00000001  // current packet is hit with NAT entry table

// Tx Frame Descriptor's Control bits
#define MACTxIntEn    0x04
#define CRCMode       0x02
#define NoCRCMode     0x00
#define PaddingMode   0x01
#define NoPaddingMode 0x00

// Tx Frame Descriptor Status
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#define TXFD_TXINTR 0x0001  // Interrupt on Transmit
#define TXFD_DEF    0x0002  // Transmit deferred
#define TXFD_TXCP   0x0008  // Transmission Completion
#define TXFD_EXDEF  0x0010  // Exceed Deferral
#define TXFD_NCS    0x0020  // No Carrier Sense Error
#define TXFD_TXABT  0x0040  // Transmission Abort
#define TXFD_LC     0x0080  // Late Collision
#define TXFD_TXHA   0x0100  // Transmission halted
#define TXFD_PAU    0x0200  // Paused
#define TXFD_SQE    0x0400  // SQE error

// Tx/Rx buffer descriptor structure
typedef struct FrameDescriptor
{
 UINT32 Status1; // RX - Ownership(2bits),RxStatus(14bits),Length(16bits)
                 // TX - Ownership(1bits),Control bits(3bits),Reserved(28bits)
 UINT32 FrameDataPtr;
 UINT32 Status2; // RX - NAT Information/Reserved(32bits)
                 // TX - TxStatus(16bits),Length(16bits)
 UINT32 NextFD;
} sFrameDescriptor;

// MAC Frame Structure
typedef struct ETH_HEADER
{
 UINT8 DestinationAddr[6] ;
 UINT8 SourceAddr[6] ;
 UINT8 LengthOrType[2] ;
} etheader ;

typedef struct MACFrame
{
 etheader Header ;
 UINT8    LLCData[1506] ;
} sMACFrame;

// MAC Tx Error Structure
typedef struct MACTxStatus
{
 UINT32 DEF;
 UINT32 TXCP;
 UINT32 EXDEF;
 UINT32 NCS;
 UINT32 TXABT;
 UINT32 LC;
 UINT32 TXHA;
 UINT32 PAU;
 UINT32 SQE;
} pMACTxStatus;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
// MAC Rx Error Structure
typedef struct MACRxStatus
{
 UINT32 RP;
 UINT32 ALIE;
 UINT32 RXGD;
 UINT32 PTLE;
 UINT32 CRCE;
 UINT32 CFR;
 UINT32 NATFSH;
 UINT32 NOP;
} pMACRxStatus;

#define MACNUM0    0
#define MACNUM1    1
#define CAM_FULL  -1

#define RxFDBaseAddr0   0x2000000
#define TxFDBaseAddr0   0x2000200
#define RxFDBaseAddr1   0x2080000
#define TxFDBaseAddr1   0x2080200

#ifndef USE_NETBUF
#define RxFBABaseAddr0  0x2000300
#define TxFBABaseAddr0  0x200c100
#define RxFBABaseAddr1  0x2080300
#define TxFBABaseAddr1  0x208c100
#endif

// Global variables  used for MAC driver
volatile UINT32 gMCMDR = MCMDR_SPCRC | MCMDR_EnMDC | MCMDR_ACP;

volatile UINT32 gMIEN = EnTXINTR | EnRXINTR | EnRXGD | EnTXCP |
                        EnTxBErr | EnRxBErr;

volatile UINT32 gCAMCMR = CAM_ECMP | CAM_AMP | CAM_ABP;

volatile UINT32 gCTxFDPtr[2], gCRxFDPtr[2];
volatile int MacTxDoneFlag[2] ;

// Global variable structure for store status
volatile pMACTxStatus gsMacTxStatus[2];
volatile pMACRxStatus gsMacRxStatus[2];

volatile int NATCheckFlag0=0, NATCheckFlag1=0;

volatile UINT32 et_data_ptr0, et_data_ptr1;
volatile UINT32 et_data_len0, et_data_len1;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
UINT8 MAC0_Address[6],MAC1_Address[6];

STATUS MacInitialize(int num)
{
 void (*old_lisr)(INT);
 UINT8 PauseMultiAddr[6]={0x01,0x80,0xc2,0x00,0x00,0x01};

 MacTxDoneFlag[num] = 0;

 if (num==0)
   {
#ifdef USE_NETBUF
    NetBuf_Init0();
#endif

    // Tx interrupt vector setup.
    AIC_SCR_EMCTX0 = 0x41;
    NU_Register_LISR(EMCTXINT0, MAC0_Tx_LISR, &old_lisr);

    // Rx interrupt vector setup.
    AIC_SCR_EMCRX0 = 0x41;
    NU_Register_LISR(EMCRXINT0, MAC0_Rx_LISR, &old_lisr);

    // Set the Tx and Rx Frame Descriptor
    TxFDInitialize(num) ;
    RxFDInitialize(num) ;

    // Set the CAM Control register
    CAMCMR_0 = gCAMCMR ;

    // set hardware address into the CAM filter
    FillCamEntry(num, 0, MAC0_Address);    // fill to CAM entry 0
    EnableCamEntry(num, 0);
    FillCamEntry(num, 1, PauseMultiAddr);  // fill multiaddr for control frame
    EnableCamEntry(num, 1);                // to CAM entry 1

    // Enable MAC Tx and Rx interrupt.
    Enable_Int(EMCTXINT0);
    Enable_Int(EMCRXINT0);

    // Configure the MAC control registers.
    MIEN_0 = gMIEN ;
    MCMDR_0 = gMCMDR ;

    // set MAC0 as LAN port
    MCMDR_0 |= MCMDR_LAN ;
   }
 else if (num==1)
   {
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
#ifdef USE_NETBUF
    NetBuf_Init1();
#endif

    // Tx interrupt vector setup.
    AIC_SCR_EMCTX1 = 0x41;
    NU_Register_LISR(EMCTXINT1, MAC1_Tx_LISR, &old_lisr);

    // Rx interrupt vector setup.
    AIC_SCR_EMCRX1 = 0x41;
    NU_Register_LISR(EMCRXINT1, MAC1_Rx_LISR, &old_lisr);

    // Set the Tx and Rx Frame Descriptor
    TxFDInitialize(num) ;
    RxFDInitialize(num) ;

    // Set the CAM Control register
    CAMCMR_1 = gCAMCMR ;

    // set hardware address into the CAM filter
    FillCamEntry(num, 0, MAC1_Address);
    EnableCamEntry(num, 0);
    FillCamEntry(num, 1, PauseMultiAddr);
    EnableCamEntry(num, 1);

    // Enable MAC Tx and Rx interrupt.
    Enable_Int(EMCTXINT1);
    Enable_Int(EMCRXINT1);

    // Configure the MAC control registers.
    MIEN_1 = gMIEN ;
    MCMDR_1 = gMCMDR ;

    // set MAC1 as WAN port
    MCMDR_1 &= ~MCMDR_LAN ;
   }

 //Reset PHY and set communication properties
 ResetPhyChip(num);

 if (num==MACNUM0)
   MCMDR_0 |= MCMDR_RXON;
 else if (num==MACNUM1)
   MCMDR_1 |= MCMDR_RXON;

 return (NU_SUCCESS);
}

void TxFDInitialize(int num)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
 sFrameDescriptor *pFrameDescriptor;
 sFrameDescriptor *pStartFrameDescriptor;
 sFrameDescriptor *pLastFrameDescriptor = NULL;
 UINT32 i;
#ifndef USE_NETBUF
 UINT32 FrameDataAddr;
#endif

 if (num==0)
   {
    // Get Frame descriptor's base address.
    TXDLSA_0 = (UINT32)TxFDBaseAddr0;
    gCTxFDPtr[num] = TXDLSA_0;

#ifndef USE_NETBUF
    // Get Transmit buffer base address.
    FrameDataAddr = (UINT32)TxFBABaseAddr0;
#endif

    // Generate linked list.
    pFrameDescriptor = (sFrameDescriptor *) gCTxFDPtr[num];
    pStartFrameDescriptor = pFrameDescriptor;

    for(i=0; i < MaxTxFrameDescriptors; i++)
    {
     if (pLastFrameDescriptor == NULL)
       pLastFrameDescriptor = pFrameDescriptor;
     else
       pLastFrameDescriptor->NextFD = (UINT32)pFrameDescriptor;

     pFrameDescriptor->Status1 = 0;
#ifdef USE_NETBUF
     pFrameDescriptor->FrameDataPtr = (UINT32)0x0;
#else
     pFrameDescriptor->FrameDataPtr = (UINT32)FrameDataAddr;
#endif
     pFrameDescriptor->Status2 = (UINT32)0x0;
     pFrameDescriptor->NextFD = NULL;

     pLastFrameDescriptor = pFrameDescriptor;
     pFrameDescriptor++;
#ifndef USE_NETBUF
     FrameDataAddr += sizeof(sMACFrame);
#endif
    }

    // Make Frame descriptor to ring buffer type.
    pFrameDescriptor--;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    pFrameDescriptor->NextFD = (UINT32)pStartFrameDescriptor;
   }
 else if (num==1)
   {
    // Get Frame descriptor's base address.
    TXDLSA_1 = (UINT32)TxFDBaseAddr1;
    gCTxFDPtr[num] = TXDLSA_1;

#ifndef USE_NETBUF
    // Get Transmit buffer base address.
    FrameDataAddr = (UINT32)TxFBABaseAddr1;
#endif

    // Generate linked list.
    pFrameDescriptor = (sFrameDescriptor *) gCTxFDPtr[num];
    pStartFrameDescriptor = pFrameDescriptor;

    for(i=0; i < MaxTxFrameDescriptors; i++)
    {
     if (pLastFrameDescriptor == NULL)
       pLastFrameDescriptor = pFrameDescriptor;
     else
       pLastFrameDescriptor->NextFD = (UINT32)pFrameDescriptor;

     pFrameDescriptor->Status1 = 0;
#ifdef USE_NETBUF
     pFrameDescriptor->FrameDataPtr = (UINT32)0x0;
#else
     pFrameDescriptor->FrameDataPtr = (UINT32)FrameDataAddr;
#endif
     pFrameDescriptor->Status2 = (UINT32)0x0;
     pFrameDescriptor->NextFD = NULL;

     pLastFrameDescriptor = pFrameDescriptor;
     pFrameDescriptor++;
#ifndef USE_NETBUF
     FrameDataAddr += sizeof(sMACFrame);
#endif
    }

    // Make Frame descriptor to ring buffer type.
    pFrameDescriptor--;
    pFrameDescriptor->NextFD = (UINT32)pStartFrameDescriptor;
   }
}

void RxFDInitialize(int num)
{
 sFrameDescriptor *pFrameDescriptor;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
 sFrameDescriptor *pStartFrameDescriptor;
 sFrameDescriptor *pLastFrameDescriptor = NULL;
 UINT32 i;
#ifndef USE_NETBUF
 UINT32 FrameDataAddr;
#endif

 if (num==0)
  {
   // Get Frame descriptor's base address.
   RXDLSA_0 = (UINT32)RxFDBaseAddr0;
   gCRxFDPtr[num] = RXDLSA_0;

#ifndef USE_NETBUF
   // Get Transmit buffer base address.
   FrameDataAddr = (UINT32)RxFBABaseAddr0;
#endif

   // Generate linked list.
   pFrameDescriptor = (sFrameDescriptor *) gCRxFDPtr[num];
   pStartFrameDescriptor = pFrameDescriptor;

   for(i=0; i < MaxRxFrameDescriptors; i++)
   {
    if (pLastFrameDescriptor == NULL)
      pLastFrameDescriptor = pFrameDescriptor;
    else
      pLastFrameDescriptor->NextFD = (UINT32)pFrameDescriptor;

    pFrameDescriptor->Status1 = RXfOwnership_DMA;
#ifdef USE_NETBUF
    pFrameDescriptor->FrameDataPtr = (UINT32)(NetBuf_Allocate0());
#else
    pFrameDescriptor->FrameDataPtr = (UINT32)FrameDataAddr;
#endif
    pFrameDescriptor->Status2 = (UINT32)0x0;
    pFrameDescriptor->NextFD = NULL;

    pLastFrameDescriptor = pFrameDescriptor;
    pFrameDescriptor++;
#ifndef USE_NETBUF
    FrameDataAddr += sizeof(sMACFrame);
#endif
   }

   // Make Frame descriptor to ring buffer type.
   pFrameDescriptor--;
   pFrameDescriptor->NextFD = (UINT32)pStartFrameDescriptor;
  }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
 else if (num==1)
   {
    // Get Frame descriptor's base address.
    RXDLSA_1 = (UINT32)RxFDBaseAddr1;
    gCRxFDPtr[num] = RXDLSA_1;

#ifndef USE_NETBUF
    // Get Transmit buffer base address.
    FrameDataAddr = (UINT32)RxFBABaseAddr1;
#endif

    // Generate linked list.
    pFrameDescriptor = (sFrameDescriptor *) gCRxFDPtr[num];
    pStartFrameDescriptor = pFrameDescriptor;

    for(i=0; i < MaxRxFrameDescriptors; i++)
    {
     if (pLastFrameDescriptor == NULL)
       pLastFrameDescriptor = pFrameDescriptor;
     else
       pLastFrameDescriptor->NextFD = (UINT32)pFrameDescriptor;

     pFrameDescriptor->Status1 = RXfOwnership_DMA;
#ifdef USE_NETBUF
     pFrameDescriptor->FrameDataPtr = (UINT32)(NetBuf_Allocate1());
#else
     pFrameDescriptor->FrameDataPtr = (UINT32)FrameDataAddr;
#endif
     pFrameDescriptor->Status2 = (UINT32)0x0;
     pFrameDescriptor->NextFD = NULL;

     pLastFrameDescriptor = pFrameDescriptor;
     pFrameDescriptor++;
#ifndef USE_NETBUF
     FrameDataAddr += sizeof(sMACFrame);
#endif
    }

    // Make Frame descriptor to ring buffer type.
    pFrameDescriptor--;
    pFrameDescriptor->NextFD = (UINT32)pStartFrameDescriptor;
   }
}

void ResetPhyChip(int num)
{
 UINT32 RdValue;

 MiiStationWrite(num, PHY_CNTL_REG, PHYAD, ENABLE_AN | RESTART_AN);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
 while (1) // wait for auto-negotiation complete
   {
    RdValue = MiiStationRead(num, PHY_STATUS_REG, PHYAD) ;
    if ((RdValue&AN_COMPLETE)!=0)
      break;
   }

 // read the result of auto-negotiation
 RdValue = MiiStationRead(num, PHY_CNTL_REG, PHYAD) ;
 if ((RdValue&DR_100MB)!=0) // 100MB
   {
    if (num==1)
      MCMDR_1 |= MCMDR_OPMOD;
    else
      MCMDR_0 |= MCMDR_OPMOD;
   }
  else
   {
    if (num==1)
      MCMDR_1 &= ~MCMDR_OPMOD;
    else
      MCMDR_0 &= ~MCMDR_OPMOD;
   }
 if ((RdValue&PHY_FULLDUPLEX)!=0) // Full Duplex
   {
    if (num==1)
      MCMDR_1 |= MCMDR_FDUP;
    else
      MCMDR_0 |= MCMDR_FDUP;
   }
  else
   {
    if (num==1)
      MCMDR_1 &= ~MCMDR_FDUP;
    else
      MCMDR_0 &= ~MCMDR_FDUP;
   }
}

void MiiStationWrite(int num,UINT32 PhyInAddr,UINT32 PhyAddr,UINT32 PhyWrData)
{
 int i=1000;

 if (num==0)
   {
    MIID_0 = PhyWrData ;
    MIIDA_0 = PhyInAddr | PhyAddr | PHYBUSY | PHYWR | MDCCR;
    while(i--) ;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    while( (MIIDA_0 & PHYBUSY) )  ;
   }
 else if (num==1)
   {
    MIID_1 = PhyWrData ;
    MIIDA_1 = PhyInAddr | PhyAddr | PHYBUSY | PHYWR | MDCCR;
    while(i--) ;
    while( (MIIDA_1 & PHYBUSY) )  ;
   }
}

UINT32 MiiStationRead(int num, UINT32 PhyInAddr, UINT32 PhyAddr)
{
 UINT32 PhyRdData ;
 if (num==0)
   {
    MIIDA_0 = PhyInAddr | PhyAddr | PHYBUSY | MDCCR;
    while( (MIIDA_0 & PHYBUSY) )  ;
    PhyRdData = MIID_0 ;
   }
 else if (num==1)
   {
    MIIDA_1 = PhyInAddr | PhyAddr | PHYBUSY | MDCCR;
    while( (MIIDA_1 & PHYBUSY) )  ;
    PhyRdData = MIID_1 ;
   }
 return PhyRdData ;
}

void FillCamEntry(int num, int entry, UINT8 *CAM_addr)
{
 UINT32 msw=0,lsw=0;

 msw =  (CAM_addr[0] << 24) |
        (CAM_addr[1] << 16) |
        (CAM_addr[2] << 8) |
         CAM_addr[3];

 lsw |= (CAM_addr[4] << 24) |
        (CAM_addr[5] << 16);

 if (num==0)
   {
    CAMxM_Reg_0(entry) = msw;
    CAMxL_Reg_0(entry) = lsw;
   }
 else if (num==1)
   {
    CAMxM_Reg_1(entry) = msw;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    CAMxL_Reg_1(entry) = lsw;
   }
}

void EnableCamEntry(int num, int entry)
{
 if (num==0)
   CAMEN_0 |= 0x00000001<<entry ;
 else if (num==1)
   CAMEN_1 |= 0x00000001<<entry ;
}

STATUS MAC0_Transmit(DV_DEVICE_ENTRY *dev, NET_BUFFER *buf_ptr)
{
 sFrameDescriptor *psTxFD;
 volatile UINT32  *pFDPtr;
 UINT8            *FramePtr;
 UINT32           *PTxFDStatus1;

 // to handle packets that are bigger than one NET Buffer can hold
 UINT32   total_length;

 while (MacTxDoneFlag[MACNUM0]) ; // wait on any outgoing Tx

 MacTxDoneFlag[MACNUM0] = 1 ;

 // Get Current TX Frame Descriptor Pointer
 psTxFD = (sFrameDescriptor *)gCTxFDPtr[MACNUM0];

#ifdef USE_NETBUF
 psTxFD->FrameDataPtr = (UINT32)(NetBuf_Allocate0());
#endif

 // Get Current TX Frame data Pointer
 pFDPtr = (UINT32 *)&psTxFD->FrameDataPtr;
 FramePtr = (UINT8 *)*pFDPtr;
 PTxFDStatus1 = (UINT32 *)&psTxFD->Status1;

 if ( (*PTxFDStatus1 & TXfOwnership_DMA) )
   return(NU_SUCCESS);

 total_length = 0;

 // copy frame data to Frame data buffer area
 do
 {
  memcpy(FramePtr, buf_ptr->data_ptr, buf_ptr->data_len);
  total_length += buf_ptr->data_len;
  FramePtr += buf_ptr->data_len;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
  // Move on to the next buffer.
  buf_ptr = buf_ptr->next_buffer;
 } while (buf_ptr != 0);

 psTxFD->Status1 |= (PaddingMode | CRCMode | MACTxIntEn);

 // Set TX Frame flag & Length Field
 psTxFD->Status2 = (UINT32)(total_length & 0xffff);

 // Cheange ownership to DMA
 psTxFD->Status1 |= TXfOwnership_DMA;

 // Enable MAC Tx control register
 if (!(MCMDR_0&MCMDR_TXON))
   MCMDR_0 |= MCMDR_TXON ;

 TSDR_0 = 0;

 return(NU_SUCCESS);
}

void MAC0_Tx_LISR(INT vector)
{
 sFrameDescriptor *psTxFD;
 UINT32 IntMACTxStatus, TxStatus;
 UINT32 NumBytes;

 if (MacTxDoneFlag[MACNUM0])
 {
 // Read MAC Tx Status Reg
 IntMACTxStatus = MISTA_0;
 MISTA_0 = IntMACTxStatus&0xffff0000;

 if (IntMACTxStatus & MISTA_TxBErr)
   {
    FIFOTHD_0|=SWR;
    MacInitialize(MACNUM0);
   }
 else
   {
    psTxFD = (sFrameDescriptor *)gCTxFDPtr[MACNUM0];
    TxStatus = (psTxFD->Status2 >> 16) & 0xffff;

    if (TxStatus & TXFD_TXCP)
      {
#ifdef USE_NETBUF
       NetBuf_Free0((NETBUF *)psTxFD->FrameDataPtr);
#endif
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    NumBytes = CheckMacTxStatus(MACNUM0);

    if (NumBytes > 0)
      {
       // Activate the transmit HISR.
       NU_Activate_HISR (&Ether_Xmit0);
       return;
      }
    else
      if (TxStatus & ( TXFD_TXABT | TXFD_DEF | TXFD_PAU | TXFD_EXDEF |
                       TXFD_NCS | TXFD_SQE | TXFD_LC | TXFD_TXHA) )
        {
         // Save Error status
         if (TxStatus & TXFD_TXABT) gsMacTxStatus[MACNUM0].TXABT++ ;
         if (TxStatus & TXFD_DEF)   gsMacTxStatus[MACNUM0].DEF++ ;
         if (TxStatus & TXFD_PAU)   gsMacTxStatus[MACNUM0].PAU++ ;
         if (TxStatus & TXFD_EXDEF) gsMacTxStatus[MACNUM0].EXDEF++ ;
         if (TxStatus & TXFD_NCS)   gsMacTxStatus[MACNUM0].NCS++ ;
         if (TxStatus & TXFD_SQE)   gsMacTxStatus[MACNUM0].SQE++ ;
         if (TxStatus & TXFD_LC)    gsMacTxStatus[MACNUM0].LC++ ;
         if (TxStatus & TXFD_TXHA)  gsMacTxStatus[MACNUM0].TXHA++ ;
        }
    }
  }
 }
}

VOID MAC0_TxHisr()
{
 STATUS         previous_int_value;
 DV_DEVICE_ENTRY *device;

 // Get a pointer to the device.
 device = DEV_Get_Dev_By_Name("740MAC0");

 if (device == NU_NULL)
   return;

 // Lock out interrupts.
 previous_int_value = NU_Control_Interrupts(NU_DISABLE_INTERRUPTS);

 DEV_Recover_TX_Buffers(device);

 // If there is another item on the list, transmit it.
 if (device -> dev_transq.head)
 {
  // Re-enable interrupts
  NU_Control_Interrupts(previous_int_value);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
  // Transmit the next packet.
  MAC0_Transmit(device, device->dev_transq.head);
 }

 // Re-enable interrupts.
 NU_Control_Interrupts(previous_int_value);
}

UINT32 CheckMacTxStatus(int num)
{
 sFrameDescriptor *psTxFD;
 UINT32 Status;
 UINT32 NumBytes;

 psTxFD = (sFrameDescriptor *) gCTxFDPtr[num];
 Status = (psTxFD->Status2 >> 16) & 0xffff;
 NumBytes = (psTxFD->Status2 & 0xffff) + ETH_HeaderSize;

 gCTxFDPtr[num] = (UINT32)(psTxFD->NextFD);

 MacTxDoneFlag[num] = 0;

 if (Status & TXFD_TXCP)
   return NumBytes;
 else
   return 0;
}

STATUS SendPacket(int num, UINT32 *buf_ptr, int Size)
{
 sFrameDescriptor *psTxFD;
 UINT32           *PTxFDStatus1;

 if (num==MACNUM0)
   {
    while (MacTxDoneFlag[MACNUM0]) ; // wait on any outgoing Tx

    MacTxDoneFlag[MACNUM0] = 1 ;

    // Get Current TX Frame Descriptor Pointer
    psTxFD = (sFrameDescriptor *)gCTxFDPtr[MACNUM0];
    PTxFDStatus1 = (UINT32 *)&psTxFD->Status1;

    if ( (*PTxFDStatus1 & TXfOwnership_DMA) )
      return(NU_SUCCESS);

    psTxFD->FrameDataPtr = (UINT32)buf_ptr;
    psTxFD->Status1 |= (PaddingMode | CRCMode | MACTxIntEn);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    psTxFD->Status2 = (UINT32)(Size & 0xffff);

    // Cheange ownership to DMA
    psTxFD->Status1 |= TXfOwnership_DMA;

    // Enable MAC Tx control register
    if (!(MCMDR_0&MCMDR_TXON))
      MCMDR_0 |= MCMDR_TXON ;

    TSDR_0 = 0;
   }
  else
   {
    while (MacTxDoneFlag[MACNUM1]) ; // wait on any outgoing Tx

    MacTxDoneFlag[MACNUM1] = 1 ;

    // Get Current TX Frame Descriptor Pointer
    psTxFD = (sFrameDescriptor *)gCTxFDPtr[MACNUM1];
    PTxFDStatus1 = (UINT32 *)&psTxFD->Status1;

    if ( (*PTxFDStatus1 & TXfOwnership_DMA) )
      return(NU_SUCCESS);

    psTxFD->FrameDataPtr = (UINT32)buf_ptr;
    psTxFD->Status1 |= (PaddingMode | CRCMode | MACTxIntEn);
    psTxFD->Status2 = (UINT32)(Size & 0xffff);

    // Cheange ownership to DMA
    psTxFD->Status1 |= TXfOwnership_DMA;

    // Enable MAC Tx control register
    if (!(MCMDR_1&MCMDR_TXON))
      MCMDR_1 |= MCMDR_TXON ;

    TSDR_1 = 0;
   }

 return(NU_SUCCESS);
}

void MAC0_Rx_LISR(INT vector)
{
 UINT32          IntMACRxStatus;

 IntMACRxStatus = MISTA_0;
 MISTA_0 = IntMACRxStatus & 0x0000ffff;

 if (IntMACRxStatus & MISTA_RxBErr)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
  {
   FIFOTHD_0|=SWR;
   MacInitialize(MACNUM0);
  }
 else
  {
   if (IntMACRxStatus & (MISTA_CFR | MISTA_CRCE | MISTA_PTLE |
                         MISTA_ALIE | MISTA_RP))
     {
      if (IntMACRxStatus & MISTA_CFR)  gsMacRxStatus[MACNUM0].CFR++ ;
      if (IntMACRxStatus & MISTA_CRCE) gsMacRxStatus[MACNUM0].CRCE++ ;
      if (IntMACRxStatus & MISTA_PTLE) gsMacRxStatus[MACNUM0].PTLE++ ;
      if (IntMACRxStatus & MISTA_ALIE) gsMacRxStatus[MACNUM0].ALIE++ ;
      if (IntMACRxStatus & MISTA_RP)   gsMacRxStatus[MACNUM0].RP++ ;
     }
    else
      NU_Activate_HISR(&Ether_Rcv0);
  }
}

STATUS MAC0_Receive(DV_DEVICE_ENTRY *device)
{
 NET_BUFFER  *buf_ptr;
 UINT32      pkt_size, bytes_left;

 // Copy the Rx Packet into as many NET Buffers as it takes!
 pkt_size = et_data_len0;
 if (pkt_size <= NET_PARENT_BUFFER_SIZE)  // One buffer case!
   buf_ptr = MEM_Buffer_Dequeue(&MEM_Buffer_Freelist);
 else  // More than one buffer needed
   buf_ptr = MEM_Buffer_Chain_Dequeue(&MEM_Buffer_Freelist, pkt_size);

 if (buf_ptr == NU_NULL)
   {
    NU_Tcp_Log_Error (TCP_PARTITION_ERROR, TCP_RECOVERABLE,
                      __FILE__, __LINE__);

    // Return that there was an error with a packet reception
    return(-1);
   }
 else
   {
    // Put head of the chain(or only one) on the incoming packet buffer list
    MEM_Buffer_Enqueue (&MEM_Buffer_List, buf_ptr);

    // Get the total number of bytes that must be copied
    bytes_left = pkt_size;

    // This sets Total Packet Length in the first (or only) buffer
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
buf_ptr->mem_total_data_len = pkt_size;

// Set the device into the buffer header
buf_ptr->mem_buf_device = device;

// Clear the flags field
buf_ptr->mem_flags = 0;

if (pkt_size <= NET_PARENT_BUFFER_SIZE) // One buffer case
  {
   // The MEM Buffer Dequeue function returns a Ptr to the Buffer's
   // Data area at the TOP of the Buffer.
   buf_ptr->data_ptr = (UINT8 *) buf_ptr;

   // Data in this buffer, same as total for this case.
   buf_ptr->data_len = pkt_size;

   // Do the copy.
   memcpy(buf_ptr->data_ptr, (UINT8 *)et_data_ptr0, pkt_size);
  }
else  // More than one NET Buffer required!  Pack the first one and
  {   // enter the while loop to fill the remaining ones needed.
   // Data area in the first buffer is less than in the rest.
   buf_ptr->data_len = NET_PARENT_BUFFER_SIZE;

   // The Ptr is to the Buffer's Data area at the TOP of the
   // Buffer, so copy the buf_ptr to the Buffer's Data Ptr.
   buf_ptr->data_ptr = (UINT8 *) buf_ptr;

   // Do the copy.
   memcpy(buf_ptr->data_ptr,(UINT8 *)et_data_ptr0,NET_PARENT_BUFFER_SIZE);

   bytes_left -= NET_PARENT_BUFFER_SIZE;

   // Advance data ptr to next block start.
   et_data_ptr0 += NET_PARENT_BUFFER_SIZE;

   while (bytes_left > 0)
   {
    // Move to the next buffer in the chain
    buf_ptr = (NET_BUFFER *) buf_ptr->next_buffer;

    // The Ptr is to the Buffer's Header start, NOT the Data area
    // at the TOP of the Buffer, so copy the Adjusted Ptr to the
    // Buffer's Data Ptr.
    buf_ptr->data_ptr = (UINT8 *) buf_ptr;

    // See if a Full Buffer is needed!  Note that these buffers
    // hold more data (0x218/536) since the me_bufhdr is not
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        // needed!
        if (bytes_left > NET_MAX_BUFFER_SIZE)
          {
           buf_ptr->data_len = NET_MAX_BUFFER_SIZE;
           bytes_left -= NET_MAX_BUFFER_SIZE;

           // Do the copy.
           memcpy(buf_ptr->data_ptr,(UINT8 *)et_data_ptr0,buf_ptr->data_len);

           // Advance data ptr to next block start!
           et_data_ptr0 += NET_MAX_BUFFER_SIZE;
          }
        else
          {
           buf_ptr->data_len = bytes_left;
           bytes_left = 0;

           // Do the copy.
           memcpy(buf_ptr->data_ptr,(UINT8 *)et_data_ptr0,buf_ptr->data_len);
          }
       }
      }

    // Let the upper layer know a good packet is here.
    NU_Set_Events(&Buffers_Available, (UNSIGNED)2, NU_OR);

    return (NU_SUCCESS);
   }
}

void MAC0_RxHisr ()
{
 sFrameDescriptor *pRxFDptr;
 UINT32          RxStatus;
 UINT32          TempAddr;
 DV_DEVICE_ENTRY  *device;
 UINT32          *FDataPtr;
#ifdef USE_NETBUF
 UINT32          FrameLength ;
#endif

 TempAddr = CRXDSA_0;

 // Process all packets received.
 do
 {
  // Get Rx Frame Descriptor
  pRxFDptr = (sFrameDescriptor *)gCRxFDPtr[MACNUM0];
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
  // Check Receive Ownership
  if ((pRxFDptr->Status1|RXfOwnership_CPU)==RXfOwnership_CPU)
    {
     RxStatus = (pRxFDptr->Status1 >> 16) & 0xffff;

     if (RxStatus & RXFD_RXGD)
       {
        gsMacRxStatus[MACNUM0].RXGD++ ;

#ifdef USE_NETBUF
        if ((RxStatus&RXFD_NOP)||(RxStatus&RXFD_NATFSH))
          {
           FrameLength = pRxFDptr->Status1 & 0xffff ;
           FDataPtr =  (UINT32 *)pRxFDptr->FrameDataPtr;

           pRxFDptr->FrameDataPtr = (UINT32)(NetBuf_Allocate0());

           SendPacket(MACNUM0,FDataPtr,FrameLength);
           //SendPacket(MACNUM1,FDataPtr,FrameLength);  // for two MACs

           if (RxStatus & RXFD_NATFSH) gsMacRxStatus[MACNUM0].NATFSH++ ;
           if (RxStatus & RXFD_NOP)    gsMacRxStatus[MACNUM0].NOP++ ;
          }
        else
#endif
         {
          et_data_ptr0 =  (UINT32)pRxFDptr->FrameDataPtr;
          et_data_len0 = pRxFDptr->Status1 & 0xffff ;

          device = DEV_Get_Dev_By_Name("740MAC0");
          device->dev_receive(device);
         }
       }
    }
  else
   {
    if ((pRxFDptr->Status1&RXfOwnership_NAT)==RXfOwnership_NAT) // ownership=NAT
      {
       if (NATCheckFlag0==0)
         {
          NATCheckFlag0=1;
          break;
         }
       else
         NATCheckFlag0=0;
      }
    else // ownership=DMA
      {
       NATCheckFlag0=0;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
      break;
     }
   }

  // Change ownership to DMA for next use
  pRxFDptr->Status1 = RXfOwnership_DMA;

  // Set Next Frame Descriptor pointer
  gCRxFDPtr[MACNUM0] = (UINT32)(pRxFDptr->NextFD);
 } while ((TempAddr = CRXDSA_0) != gCRxFDPtr[MACNUM0]);

 RSDR_0 = 0;
}
```

*** All the above program referenced from Nucleus ethernet driver for NUC740.**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9 NATA (Network Address Translation Accelerator)

9.1 Overview

The Network Address Translation Accelerator (NATA) provides IP address and port number translations. An inside (local) IP address is mapped to an outside (global) IP address, meaning that an LAN IP address is replaced by the appropriate WAN IP address to do the IP sharing, and vice versa. Users can use NATA on NUC740 to speed up fixed (static) address translation and reduce software loading on processing layer-3 IP replacement. The NATA provides 64 entries to do the IP/Port content replacement, it works with the 2 EMC ports to do the NAT operations.

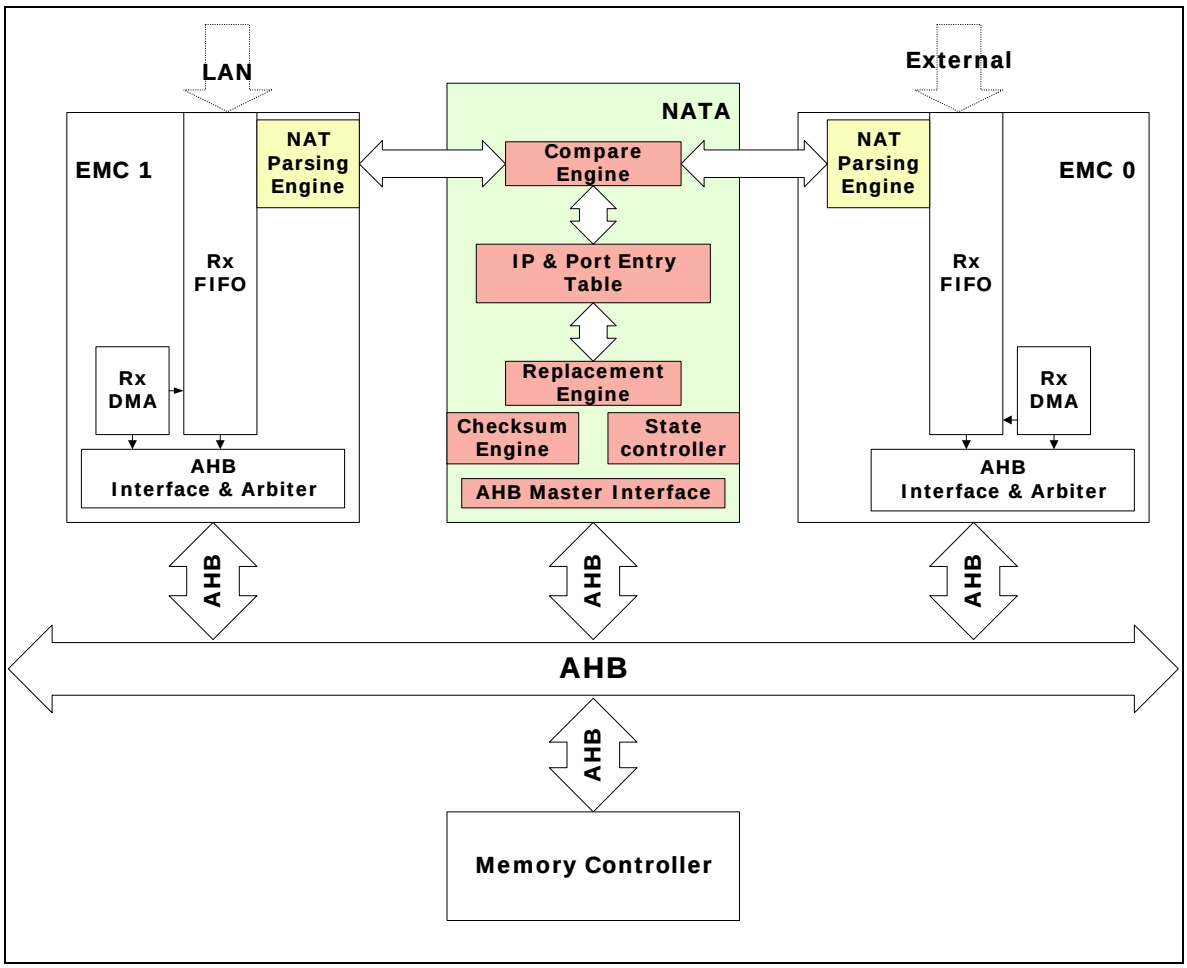
The NATA includes the following features :

- Support TCP/UDP packets (also support PPPoE with L/T 0x8864)
- Hardware acceleration on IP address / port number lookup and replacement for network address translation, including MAC address translation
- Provide 64 entries of translation table
- Filter packets with specified IP

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.2 Block Diagram

Figure 9-1 NATA Block Diagram



9.3 Registers

This set of registers is used to access status/control information to/from the NAT engine. These registers are used for loading commands generated by user, indicating network translation status, and providing interrupt control. The registers used by NUC740 NATA controller are divided into two groups :

- NATA Control and Status Registers
- Address Lookup and Replacement Entry Registers

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.3.1 NATA Control and Status Registers

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

Register	Offset	R/W	Description	Reset Value
NATA Control and Status Registers				
NATCMD	0xFFF0.6000	R/W	NAT Command Register	0x0000.0000
NATCCLR0	0xFFF0.6010	W	NAT Counter 0 Clear Register	0x0000.0000
NATCCLR1	0xFFF0.6014	W	NAT Counter 1 Clear Register	0x0000.0000
NATCCLR2	0xFFF0.6018	W	NAT Counter 2 Clear Register	0x0000.0000
NATCCLR3	0xFFF0.601C	W	NAT Counter 3 Clear Register	0x0000.0000
NATCFG0	0xFFF0.6100	R/W	NAT Entry 0 Configuration Register	0x0000.0000
NATCFG1	0xFFF0.6104	R/W	NAT Entry 1 Configuration Register	0x0000.0000
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
NATCFG63	0xFFF0.61FC	R/W	NAT Entry 63 Configuration Register	0x0000.0000

9.3.2 Address Lookup and Replacement Entry Registers

Register	Offset	R/W	Description	Reset Value
Address Lookup and Replacement Registers				
EXMACM	0xFFF0.6200	R/W	External MAC Address Most Significant Word Register	0x0000.0000
EXMACL	0xFFF0.6204	R/W	External MAC Address Least Significant Word Register	0x0000.0000
INMACM	0xFFF0.6208	R/W	Internal MAC Address Most Significant Word Register	0x0000.0000
INMACL	0xFFF0.620C	R/W	Internal MAC Address Least Significant Word Register	0x0000.0000
MASAD0	0xFFF0.6800	R/W	NAT Masquerading IP Address Entry 0	0x0000.0000
MASPN0	0xFFF0.6804	R/W	NAT Masquerading Port Number Entry 0	0x0000.0000
LSAD0	0xFFF0.6808	R/W	Local Station IP Address Entry 0	0x0000.0000
LSPN0	0xFFF0.680C	R/W	Local Station Port Number Entry 0	0x0000.0000

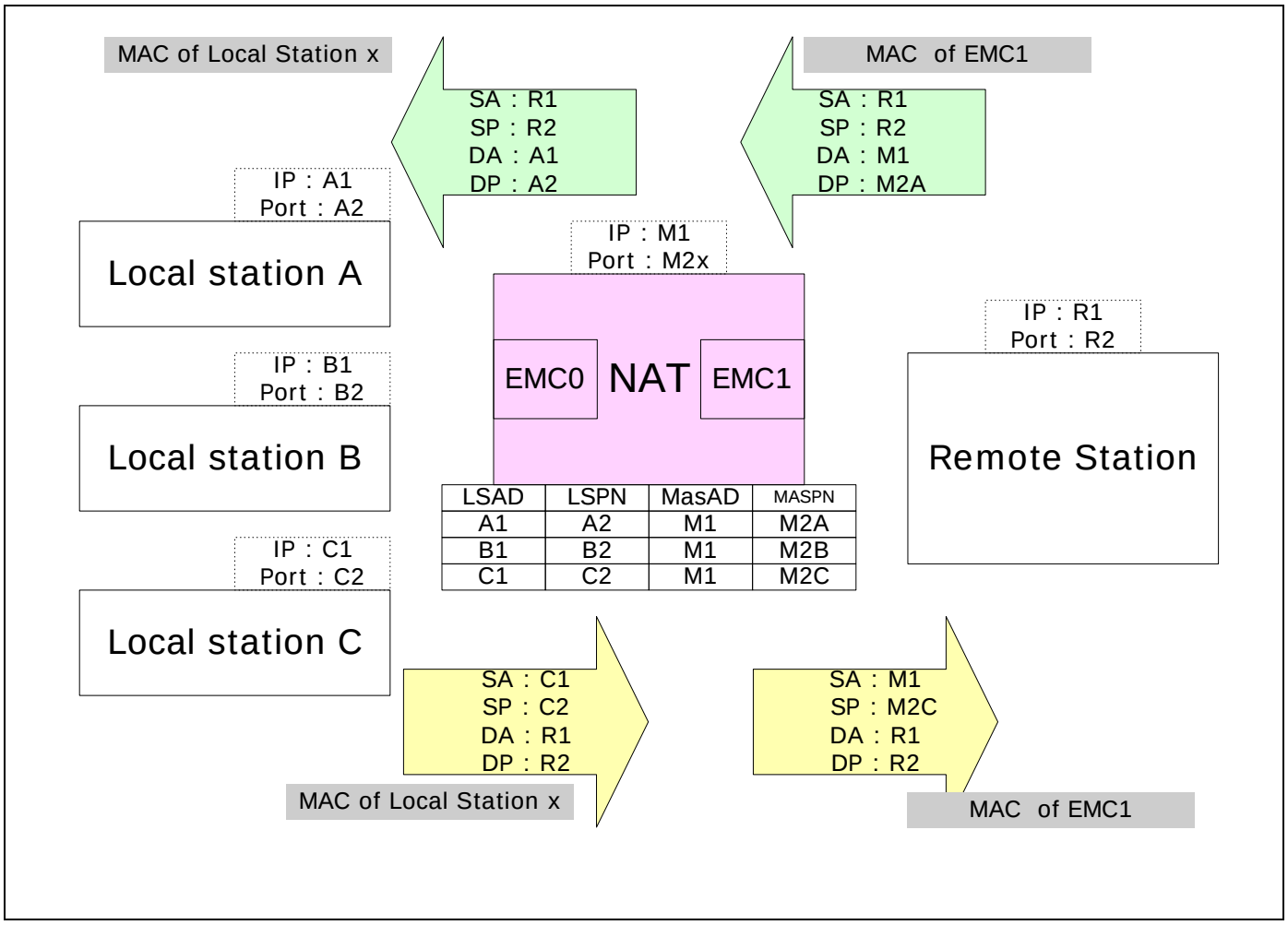
The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

LSMAC0M	0xFFF0.6810	R/W	Local Station MAC Address Most Significant Word Register for Entry 0	0x0000.0000
LSMAC0L	0xFFF0.6814	R/W	Local Station MAC Address Least Significant Word Register for Entry 0	0x0000.0000
RSMAC0M	0xFFF0.6818	R/W	Remote Station MAC Address Most Significant Word Register for Entry 0	0x0000.0000
RSMAC0L	0xFFF0.681C	R/W	Remote Station MAC Address Least Significant Word Register for Entry 0	0x0000.0000
MASAD1	0xFFF0.6820	R/W	NAT Masquerading IP Address Entry 1	0x0000.0000
MASPN1	0xFFF0.6824	R/W	NAT Masquerading Port Number Entry 1	0x0000.0000
LSAD1	0xFFF0.6828	R/W	Local Station IP Address Entry 1	0x0000.0000
LSPN1	0xFFF0.682C	R/W	Local Station Port Number Entry 1	0x0000.0000
LSMAC1M	0xFFF0.6830	R/W	Local Station MAC Address Most Significant Word Register for Entry 1	0x0000.0000
LSMAC1L	0xFFF0.6834	R/W	Local Station MAC Address Least Significant Word Register for Entry 1	0x0000.0000
RSMAC1M	0xFFF0.6838	R/W	Remote Station MAC Address Most Significant Word Register for Entry 1	0x0000.0000
RSMAC1L	0xFFF0.683C	R/W	Remote Station MAC Address Least Significant Word Register for Entry 1	0x0000.0000
. . .	. . .	. . .	. . .	. . .
MASAD63	0xFFF0.6FE0	R/W	NAT Masquerading IP Address Entry 63	0x0000.0000
MASPN63	0xFFF0.6FE4	R/W	NAT Masquerading Port Number Entry 63	0x0000.0000
LSAD63	0xFFF0.6FE8	R/W	Local Station IP Address Entry 63	0x0000.0000
LSPN63	0xFFF0.6FEC	R/W	Local Station Port Number Entry 63	0x0000.0000
LSMAC63M	0xFFF0.6FF0	R/W	Local Station MAC Address Most Significant Word Register for Entry 63	0x0000.0000
LSMAC63L	0xFFF0.6FF4	R/W	Local Station MAC Address Least Significant Word Register for Entry 63	0x0000.0000
RSMAC63M	0xFFF0.6FF8	R/W	Remote Station MAC Address Most Significant Word Register for Entry 63	0x0000.0000
RSMAC63L	0xFFF0.6FFC	R/W	Remote Station MAC Address Least Significant Word Register for Entry 63	0x0000.0000

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

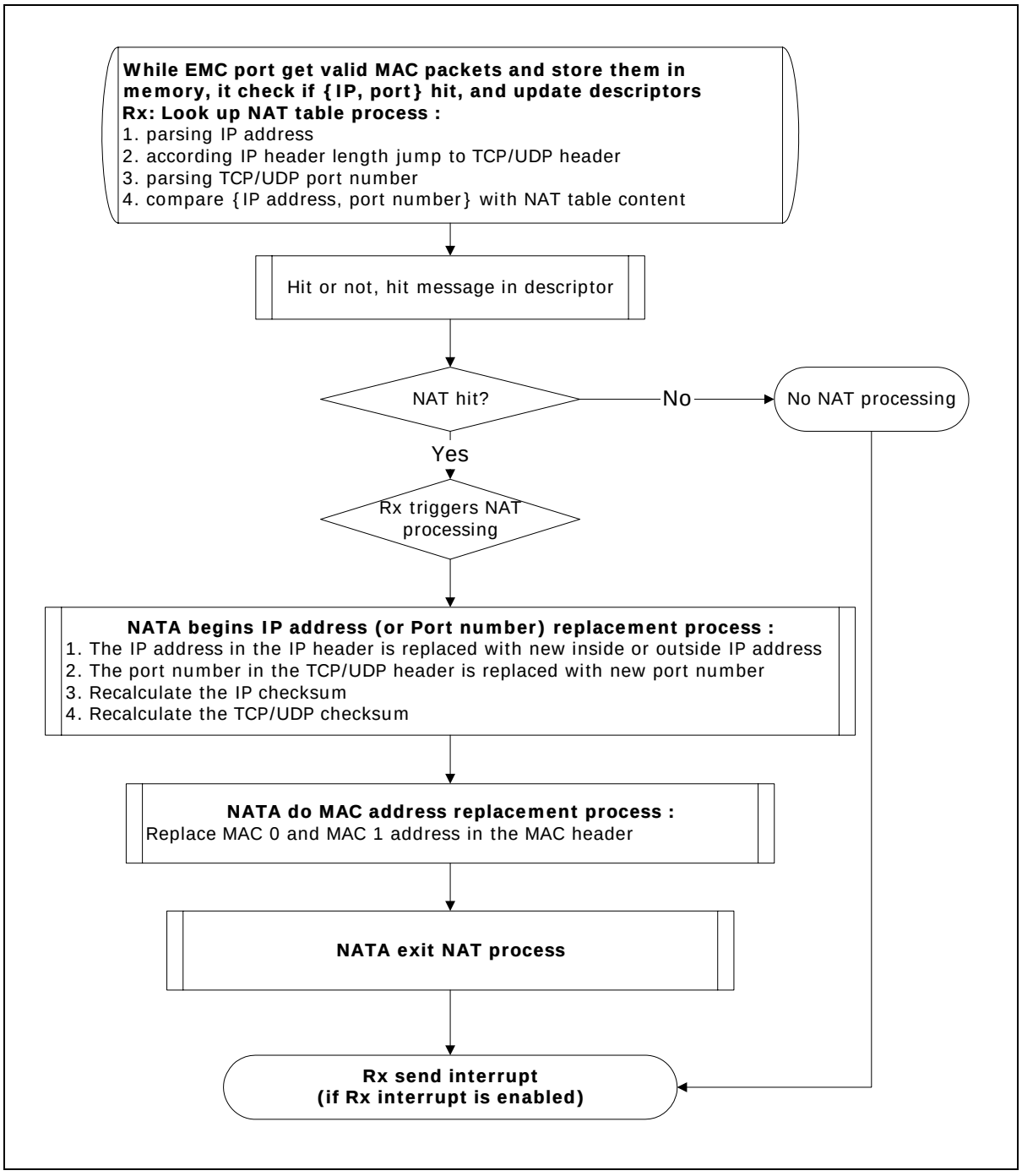
9.4 Functional Descriptions

Figure 9-2 NAT lookup and replacement flow



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 9-3 NAT Process Flow



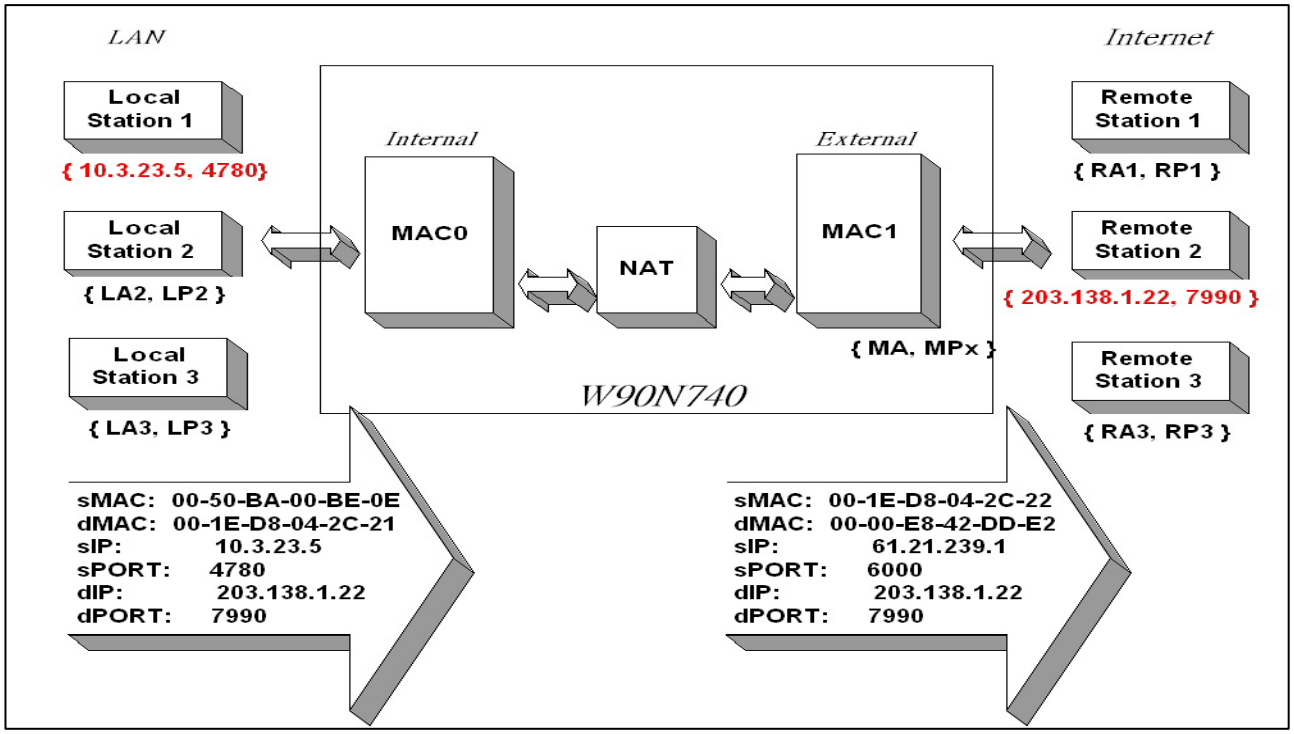
The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.5 NATA Processing Example (LAN to WAN)

INMAC : 00-1E-D8-04-2C-21 EXMAC : 00-1E-D8-04-2C-22

	MA	MP	LA	LP	LSMAC	RSMAC
Entry 0	61.21.239.1	6000	10.3.23.5	4780	00-50-BA-00-BE-0E	00-00-E8-42-DD-E2
Entry 1	61.21.239.1	6001	10.3.23.33	7900	00-50-BA-00-A3-90	00-00-E8-42-DD-E2
…						
Entry 63	61.21.239.1	6090	10.3.23.19	2001	00-1E-99-30-00-1E	00-00-E8-42-DD-E2

Figure 9-4 LAN to WAN NATA Processing



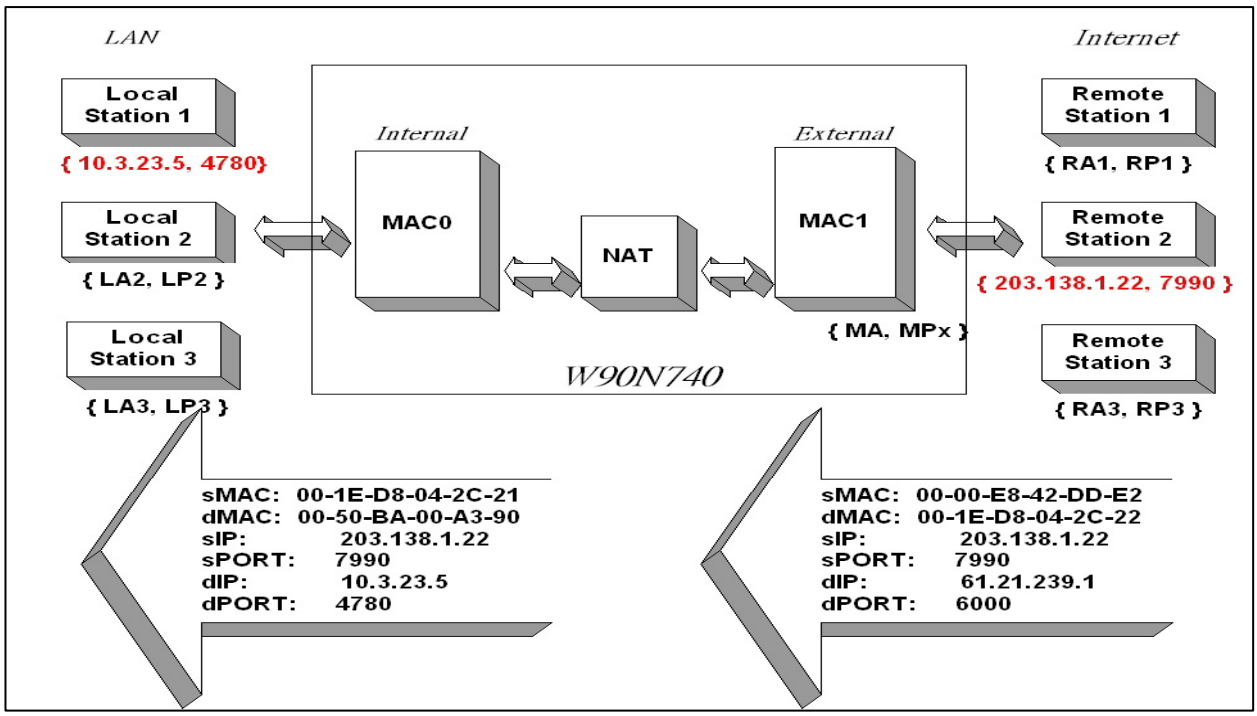
The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.6 NATA Processing Example (WAN to LAN)

INMAC : 00-1E-D8-04-2C-21 EXMAC : 00-1E-D8-04-2C-22

	MA	MP	LA	LP	LSMAC	RSMAC
Entry 0	61.21.239.1	6000	10.3.23.5	4780	00-50-BA-00-BE-0E	00-00-E8-42-DD-E2
Entry 1	61.21.239.1	6001	10.3.23.33	7900	00-50-BA-00-A3-90	00-00-E8-42-DD-E2
…						
Entry 63	61.21.239.1	6090	10.3.23.19	2001	00-1E-99-30-00-1E	00-00-E8-42-DD-E2

Figure 9-5 WAN to LAN NATA Processing



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.7 NATA Registers

9.7.1 MAC Address Registers Format

(EXMACM, EXMACL, INMACM, INMACL, LSMACxM, LSMACxL, RSMACxM, RSMACxL)

9.7.2 MAC Address Most Significant Word Register(EXMACM, INMACM, LSMACxM, RSMACxM)

31	30	29	28	27	26	25	24
Address Byte 6 (Most Significant Byte)							
23	22	21	20	19	18	17	16
Address Byte 5							
15	14	13	12	11	10	9	8
Address Byte 4							
7	6	5	4	3	2	1	0
Address Byte 3							

9.7.3 MAC Address Least Significant Word Register(EXMACL, INMACL, LSMACxL, RSMACxL)

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

For example, if the External MAC address is 12-34-56-78-90-13, then EXMACM is 12-34-56-78 and EXMACL is 90-13-00-00.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.7.4 IP Address Registers (MASADx, LSADx)

31	30	29	28	27	26	25	24
IP Address Byte 4 (Most Significant Byte)							
23	22	21	20	19	18	17	16
IP Address Byte 3							
15	14	13	12	11	10	9	8
IP Address Byte 2							
7	6	5	4	3	2	1	0
IP Address Byte 1							

For example, if the masquerading address is 140.112.2.100, then the value in MASAD is 8C-70-02-64.

9.7.5 Port Number Registers (MASPNx, LSPNx)

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

For example, if masquerading port number is 7500, then the value in MASPN is 00-00-1D-4C.

9.7.6 MAC Command Register (MCMDR_0, MCMDR_1) (0xFFF0308C, 0xFFF0388C)

31	30	29	28	27	26	25	24
Reserved						EnTMode	LAN
23	22	21	20	19	18	17	16
LPCS	EnRMII	LBK	OPMOD	EnMDC	FDUP	EnSQE	SDPZ
15	14	13	12	11	10	9	8
Reserved						NDEF	TXON
7	6	5	4	3	2	1	0
Reserved		SPCRC	AEP	ACP	ARP	ALP	RXON

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

LAN [24] : LAN Port Setting Mode

Default value : 0

Set this bit to set this EMC port as a LAN port, and clear this bit to set the EMC port as a WAN port. The S/W program must initial an EMC as a LAN port, and another EMC as a WAN port (for example, EMC 0 as a LAN port and EMC 1 as a WAN port) to let the NAT accelerator work properly.

9.7.7 NAT Entry x Configuration Registers (NATCFGx)

31	30	29	28	27	26	25	24
EHCNTx							
23	22	21	20	19	18	17	16
EHCNTx							
15	14	13	12	11	10	9	8
Reserved					CNTx		
7	6	5	4	3	2	1	0
Nop	Discard	Inverse	PxRE	AxRE	PxCE	AxCE	ExEN

EHCNTx [31 : 16] : Counter of Packets Hit with Entry x

Default value : 0

The register is read-only, to indicate that how many packets are hit with entry x. When a new packet is hit with entry x, the value on corresponding EHCNTx, will increase. The value can be reset to zero by writing one to corresponding CLREHx in NATCCLR register.

CNTx [10 : 8] : Number of Entry x Hit Packets to be processed

Default value : 0

The register is read-only, to indicate that how many packet is hit with entry x and still not processed by NAT. When a new packet is hit with entry x, the corresponding entry counter, CNTx, will increase. On the other hand, when a packet hit with entry x is process, the corresponding CNTx will be decrease.

Nop [7] : Packet Nop bit

Default value : 0

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Set this bit to receive current hit packet as usual. The Rx descriptor will be updated for this packet and set the Nop bit in descriptor. This bit will be valid while Discard[6]=0.

Discard [6] : Packet Discard bit

Default value : 0

Set this bit to automatically discard current hit packet. The Rx descriptor will not be updated for this packet. It is applicable for IP filter.

Inverse [5] : Inverse Comparison and Replacement bit

Default value : 0

Set this bit to change comparison and replacement field in packets. For example, at the WAN port, destination address and port {DA, DP} is compared for inverse bit (I bit) clear. If I bit is set, source address and port {SA, SP} is compared instead of {DA, DP}. It is applicable for IP filter.

PxRE [4] : Port Number Replacement Enable at Entry x

Default value : 0

Set the bits to selectively enable port number replacement at the entry location. This bit is valid when ExEN is set.

AxRE [3] : IP Address Replacement Enable at Entry x

Default value : 0

Set the bits to selectively enable IP address replacement at the entry location. This bit is valid when ExEN is set.

PxCE [2] : Port Number Comparison Enable at Entry x

Default value : 0

Set the bits to selectively enable port number comparison at the entry location. This bit is valid when ExEN is set.

AxCE [1] : IP Address Comparison Enable at Entry x

Default value : 0

Set the bits to selectively enable IP address comparison at the entry location. This bit is valid when ExEN is set.

ExEN [0] : Entry x Comparison Enable bit

Default value : 0

Set the bits to selectively enable specified entry location comparison. To disable an entry location, clear the bit. If S/W wants to change some entry data, it has to disable the selected

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

entry and monitor corresponding CNTx till the value is 0, then it is acceptable to change new entry data.

9.7.8 NAT Entry Configuration Register (NATCFGx) Programming Note :

In the following sections, these symbols are defined as

MA : Masquerading IP address
MP : Masquerading port number
LA : Local mapping IP address
LP : Local mapping port number
SA : (IP) address at source field
SP : Port number at source field
DA : (IP) address at destination field
DP : Port number at destination field

9.7.9 Data Content in the NATA table

	MA	MP	LA	LP
Entry 0	MA 0	MP 0	LA 0	LP 0
Entry 1	MA 1	MP 1	LA 1	LP 1
….	….	….	….	….
Entry 63	MA 63	MP 63	LA 63	LP 63

9.7.10 NATA comparison and replacement table at different port

	Inverse bit is reset		Inverse bit is set	
	Comparison	**Replacement**	**Comparison**	**Replacement**
WAN port (for receiving remote packets)	DA with MA (if AxCE set)	DA with LA (if AxRE set)	SA with MA (if AxCE set)	SA with LA (if AxRE set)
	DP with MP (if PxCE set)	DP with LP (if PxRE set)	SP with MP (if PxCE set)	SP with LP (if PxRE set)
LAN port (for receiving local station packets)	SA with LA (if AxCE set)	SA with MA (if AxRE set)	DA with LA (if AxCE set)	DA with MA (if AxRE set)
	SP with LP (if PxCE set)	SP with MP (if PxRE set)	DP with LP (if PxCE set)	DP with MP (if PxRE set)

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.7.11 Possible Settings for Different Applications

	AxCE	PxCE	AxRE	PxRE	Inverse	Discard
Comparison IP address and port number	1	1	1	1	0	0
Comparison IP address	1	0	1	0	0	0
Discard the hit packets for IP filtering	1	1	0	0	1	1

9.8 NAT Command Registers (NATCMD)

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

NATEN [0] : NAT enable bit

Default value : 0

Set this bit to start NAT function. The EMC Rx will begin packet parsing and lookup procedure if this bit is set. Clear this bit will stop all NAT operations.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.8.1 NAT Counter x Clear Register (NATCCLRx, x : 0-3)

NATCCLR0 (entry 0-15) (NATCCLR1 : entry 16-31, NATCCLR2 : entry 32-47, NATCCLR3 : entry 48-63)

31	30	29	28	27	26	25	24
CLREH15	CLREH14	CLREH13	CLREH12	CLREH11	CLREH10	CLREH 9	CLREH 8
23	22	21	20	19	18	17	16
CLREH 7	CLREH 6	CLREH 5	CLREH 4	CLREH 3	CLREH 2	CLREH 1	CLREH 0
15	14	13	12	11	10	9	8
CLRCNT15	CLRCNT14	CLRCNT13	CLRCNT12	CLRCNT11	CLRCNT10	CLRCNT9	CLRCNT8
7	6	5	4	3	2	1	0
CLRCNT7	CLRCNT6	CLRCNT5	CLRCNT4	CLRCNT3	CLRCNT2	CLRCNT1	CLRCNT0

CLREHx (x : 63 ~ 0) [31 : 16] : Entry x hit counter clear bit

Default value : 0

Set the appropriate bit to clear the corresponding entry x hit counter (EHCNTx); writing zero to the bit has no effect.

CLRCNTx (x : 63 ~ 0) [15 : 0] : NATA Entry x counter clear bit

Default value : 0

Set the appropriate bit to clear the corresponding entry counter (CNTx); writing zero to the bit has no effect.

9.8.2 EMC Rx Buffer Descriptor

O	Rx Status	Frame Length
Data Buffer Starting Address		
NAT Information (Reserved)		
Next Descriptor Starting Address		

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Bit[31 : 30] O	Ownership
00b	CPU
10b	DMA
11b	NATA
01b	Undefined

NUC740 EMC receive DMA is allowed to access current descriptor if bit 31 is set to 1 by the user driver program. If the entire frame is received successfully, then the ownership bit 31 is cleared and the ownership is granted to CPU.

If NATA is enabled, NATA is also allowed to access current descriptor and bit 30 is set to 1 by NATA when NATA is processing.

9.8.3 Rx Status (RXSTA) : Receive Status

		29	28	27	26	25	24
		Hit	IPHit	PortHit	Inverse	NATFSH	Nop
23	22	21	20	19	18	17	16
Reserved	RP	ALIE	RXGD	PTLE	Reserved	CRCE	RXINTR

Bits 29-24 are NAT information for the NAT accelerator, and reserved if the NATA is disabled.

Hit : current packet is hit with NAT entry table

The value is 1 if current packet IP/Port is in the entry list.

IPHit : current packet is hit on IP address

The value is 1 if current packet IP/Port is hit in the IP address location.

PortHit : current packet is hit on Port Number

The value is 1 if current packet IP/Port is hit in the port number location.

Inverse : current hit entry is setting on inverse mode

The value is 1 if current hit entry is on inverse mode.

NATFSH : NAT Processing Finish

The value is 1 if current packet NAT processing is finished and successful. This bit will be written while NATA finish the NAT processing.

N**OP : No Operation**

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

This bit indicates the packet is hit in NAT table but no need to be replaced by NATA.

9.8.4 NAT INFORMATION

31	30	29	28	27	26	25	24
Reserved		TCP information					
23	22	21	20	19	18	17	16
Reserved	UCK_Err	TU_Err	NH_Err	IP Header Length			
15	14	13	12	11	10	9	8
Reserved		Hit Entry Number					
7	6	5	4	3	2	1	0
Reserved	PPPCaps	PPPoE	UCKS	UDP	TCP	L/W	Hit

TCP information : URG (bit 29), ACK, PSH, RST, SYN, FIN (bit 24)

The six bit values show current TCP status, and are to the copy of the six bits in TCP header. The values are valid if current packet is TCP type and Hit is set.

UCK_Err : TCP/UCKS Error

TU_Err : TCP/UDP Error

NH_Err : No Hit Error

These bits records error status if NAT processing error is occurred and are wrote by NATA.

UCK_Err	TCP = 1 and UCKS = 1
TU_Err	TCP = 1 and UDP = 1
NH_Err	No hit error

IP Header Length : TCP/UDP header location offset

The offset value lets the NAT accelerator to identify the starting address of TCP or UDP header, which is used for NAT to parsing port data. The value is valid if Hit is set.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Hit Entry Number : the entry number hit with the input address

The value indicates which entry is hit to let NAT accelerator to take corresponding data. The value is valid if Hit is set.

PPPCaps : PPPoE datagram encapsulated

The value is 1 if PPP encapsulation is used (8 bits protocol field), and 0 if encapsulation is not used (16 bits protocol field).

PPPoE : PPPoE protocol

The value is 1 if the packet takes PPPoE protocol instead of IP protocol. The value is 0 if the packet takes IP protocol.

UCKS : UDP protocol with skip checksum replacement

The value is 1 if the packet takes UDP protocol, and its checksum is zero. The NAT accelerator will skip the checksum replacement procedure.

UDP : apply UDP protocol

Notify NAT engine to apply UDP protocol. The value is 1 if the packet takes UDP protocol, and 0 if the packet takes non-UDP protocol. The value is valid if Hit is set.

TCP : apply TCP protocol

Notify NAT engine to apply TCP protocol. The value is 1 if the packet takes TCP protocol, and 0 if the packet takes non-TCP protocol. The value is valid if Hit is set.

L/W : hit port; the value is

1 if internal (**LAN**) port gets hit, and

0 if external (**WAN**) port is hit

The S/W program must specify LAN port and WAN port with the two EMCs. For example, EMC 0 is connected to WAN port, and EMC 1 is connected to LAN port. If NAT is enabled, EMC 0 is connected to external port for {MA, MP} comparison, and EMC 1 is connected to internal port for {LA, LP} comparison. The L/W value is 1 if the hit port is internal port, and 0 if the hit port is external port. The value is valid if Hit is set.

Hit : current packet is hit with NAT entry table

The value is 1 if current packet IP/port is in the entry list.

9.9 Software Management on NATA

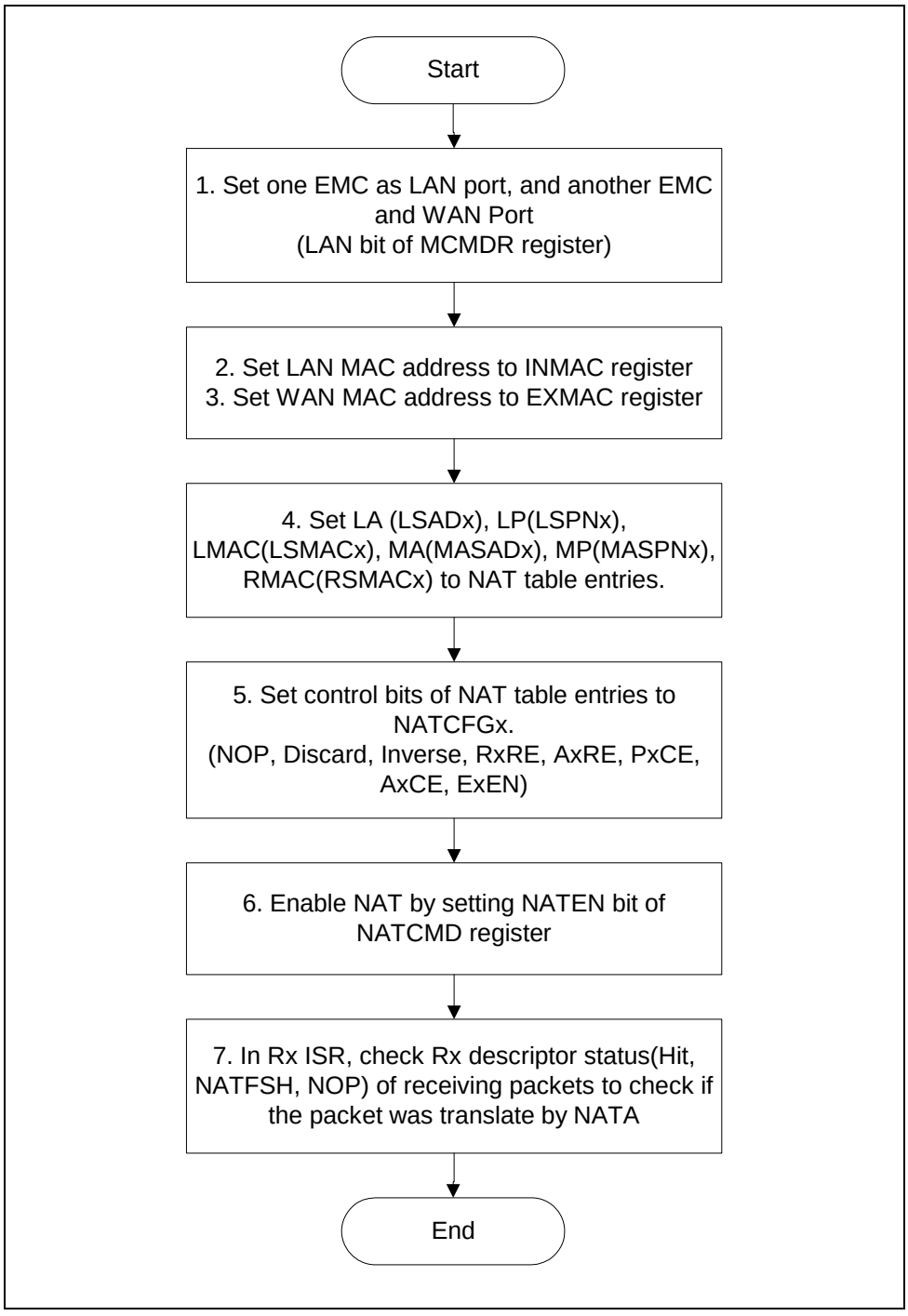
1. Set one EMC as LAN port, and another EMC as WAN port

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

(LAN bit of MCMDR register).
2. Set LAN port EMC MAC address to INMAC register.
3. Set WAN port EMC MAC address to EXMAC register.
4. Set LA, LP, LMAC, MA, MP, RMAC to NAT table entries
(registers LSADx, LSPNx, LSMACx, MASADx, MASPNx, RSMACx)
5. set control bits of NAT table entries to register NATCFGx
(bits Nop, Discard, Inverse, PxRE, AxRE, PxCE, AxCE, ExEN)
6. Enable NAT by setting NATEN bit of NATCMD register
7. In Rx ISR, check Rx descriptor status (Hit, NATFSH, NOP) of receiving packets to see if the packet is translated by NATA or not.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Figure 9-6 NATA Management



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

9.10 Sample Code

```
#define NATTbl_Entry_Num  64
#define MAC_ADDR_SIZE     6
#define IP_ADDR_SIZE          4

#define VPint     *(volatile unsigned int *)
#define VPshort *(volatile unsigned short *)
#define VPchar  *(volatile unsigned char *)

// Special Register Start Address After System Reset
#define Base_Addr 0xfff00000

// NATA Control and Status Registers
#define NATCMD       (VPint(Base_Addr+0x6000))
#define NATCCLR0   Base_Addr+0x6010
#define NATCCLR(x) (VPint(NATCCLR0+x*0x4))
#define NATCFG0     Base_Addr+0x6100
#define NATCFG(x)   (VPint(NATCFG0+x*0x4))
// NATA Address Lookup and Replacement Registers
#define EXMACM      (VPint(Base_Addr+0x6200))
#define EXMACL       (VPint(Base_Addr+0x6204))
#define INMACM       (VPint(Base_Addr+0x6208))
#define INMACL        (VPint(Base_Addr+0x620c))
#define MASAD0       Base_Addr+0x6800
#define MASPN0       Base_Addr+0x6804
#define LSAD0          Base_Addr+0x6808
#define LSPN0          Base_Addr+0x680c
#define LSMAC0M    Base_Addr+0x6810
#define LSMAC0L     Base_Addr+0x6814
#define RSMAC0M    Base_Addr+0x6818
#define RSMAC0L     Base_Addr+0x681c
#define MASAD(x)     (VPint(MASAD0 +x*0x20))
#define MASPN(x)     (VPint(MASPN0 +x*0x20))
#define LSAD(x)        (VPint(LSAD0  +x*0x20))
#define LSPN(x)        (VPint(LSPN0  +x*0x20))
#define LSMACM(x)  (VPint(LSMAC0M+x*0x20))
#define LSMACL(x)   (VPint(LSMAC0L+x*0x20))
#define RSMACM(x)  (VPint(RSMAC0M+x*0x20))
#define RSMACL(x)   (VPint(RSMAC0L+x*0x20))

// NAT Command Register(NATCMD)
#define NATEN       0x00000001  // NAT Enable bit

// NAT Entry Configuration Register(NATCFG)
#define NAT_NOP     0x00000080  // No Operation bit
#define NAT_Discard 0x00000040  // Packet Discard bit
#define NAT_Inverse 0x00000020  // Inverse Comparison and Replacement bit
#define NAT_PxRE    0x00000010  // Port Number Replacement Enable
#define NAT_AxRE    0x00000008  // IP Address Replacement Enable
#define NAT_PxCE    0x00000004  // Port Number Comparison Enable
#define NAT_AxCE    0x00000002  // IP Address Comparison Enable
#define NAT_ExEN    0x00000001  // Entry Enable bit

// Enable NAT Function
void NAT_Enable()
{
 NATCMD = NATEN;
}

// Disable NAT Function
void NAT_Disable()
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
 NATCMD = 0;
}

// Clear NAT entry Hit Counter
void NAT_HitCount_Clear(int entry)
{
 NATCCLR(entry/16) = 1<<((entry%16)+16);
}

// Clear All NAT Hit Counters
void NAT_HitCount_ClearAll()
{
 int i=0;

 for (i=0;i<4;i++)
    NATCCLR(i) = 0xffff0000;
}

// Clear NAT entry Processed Hit Counter
void NAT_ProcCount_Clear(int entry)
{
 NATCCLR(entry/16) = 1<<(entry%16);
}

// Clear All NAT Processed Hit Counters
void NAT_ProcCount_ClearAll()
{
 int i=0;

 for (i=0;i<4;i++)
    NATCCLR(i) = 0x0000ffff;
}



// Read NAT entry Hit Counter
int  NAT_HitCount_Read(int entry)
{
 int count;

 count = (int)(NATCFG(entry)>>16);

 return count;
}

// Get the entry with Minimum NAT Hit Counter
int  NAT_HitCount_MinEntry()
{
 int count, min, entry, i;

 entry = 0;
 min = (int)(NATCFG(entry)>>16);

 for (i=1;i<NATTbl_Entry_Num;i++)
    {
     count = (int)(NATCFG(i)>>16);
     if (count<min)
       {
        min=count;
        entry=i;
       }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    }

 return entry;
}

// Read NAT entry Processed Hit Counter
int  NAT_ProcCount_Read(int entry)
{
 int count;

 count = (int)((NATCFG(entry)&0x00000700)>>8);

 return count;
}

// Get the entry with Minimum NAT Processed Hit Counter
int  NAT_ProcCount_MinEntry()
{
 int count, min, entry, i;

 entry = 0;
 min = (int)((NATCFG(entry)&0x00000700)>>8);

 for (i=1;i<NATTbl_Entry_Num;i++)
    {
     count = (int)((NATCFG(i)&0x00000700)>>8);
     if (count<min)
       {
        min=count;
        entry=i;
       }
    }

 return entry;
}

// Enable entry x of NAT Table
void NAT_Entry_Config(int entry, UINT32 value)
{
 NATCFG(entry) = value;
}

// Enable entry x of NAT Table
void NAT_Entry_Enable(int entry)
{
 NATCFG(entry) |= NAT_ExEN;
}

// Disable entry x of NAT Table
void NAT_Entry_Disable(int entry)
{
 NATCFG(entry) &= ~NAT_ExEN;
}

// IP Address Comparison Enable at entry x
void NAT_AddrCompare_Enable(int entry)
{
 NATCFG(entry) |= NAT_AxCE;
}

// IP Address Comparison Disable at entry x
void NAT_AddrCompare_Disable(int entry)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
{
 NATCFG(entry) &= ~NAT_AxCE;
}

// Port Number Comparison Enable at entry x
void NAT_PortCompare_Enable(int entry)
{
 NATCFG(entry) |= NAT_PxCE;
}

// Port Number Comparison Disable at entry x
void NAT_PortCompare_Disable(int entry)
{
 NATCFG(entry) &= ~NAT_PxCE;
}

// IP Address Replacement Enable at entry x
void NAT_AddrReplace_Enable(int entry)
{
 NATCFG(entry) |= NAT_AxRE;
}

// IP Address Replacement Disable at entry x
void NAT_AddrReplace_Disable(int entry)
{
 NATCFG(entry) &= ~NAT_AxRE;
}

// Port Number Replacement Enable at entry x
void NAT_PortReplace_Enable(int entry)
{
 NATCFG(entry) |= NAT_PxRE;
}

// Port Number Replacement Disable at entry x
void NAT_PortReplace_Disable(int entry)
{
 NATCFG(entry) &= ~NAT_PxRE;
}

// Inverse Comparison and  Replacement at entry x
void NAT_Inverse_Enable(int entry)
{
 NATCFG(entry) |= NAT_Inverse;
}

// Disable Inverse Comparison and  Replacement at entry x
void NAT_Inverse_Disable(int entry)
{
 NATCFG(entry) &= ~NAT_Inverse;
}

// Packet Discard at entry x
void NAT_Discard_Enable(int entry)
{
 NATCFG(entry) |= NAT_Discard;
}


// Disable Packet Discard at entry x
void NAT_Discard_Disable(int entry)
{
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
 NATCFG(entry) &= ~NAT_Discard;
}

// Packet Discard at entry x
void NAT_NOP_Enable(int entry)
{
 NATCFG(entry) |= NAT_NOP;
}

// Disable Packet Discard at entry x
void NAT_NOP_Disable(int entry)
{
 NATCFG(entry) &= ~NAT_NOP;
}

// External MAC Address Configuration
void NAT_EXMAC_Config(UINT8 *addr)
{
 UINT32 addrtemp;
 int i;

 addrtemp = 0;
 for (i=0;i<(int)MAC_ADDR_SIZE-2;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 EXMACM = addrtemp;

 addrtemp = 0;
 for (i=(int)(MAC_ADDR_SIZE-2);i<(int)MAC_ADDR_SIZE;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 addrtemp = (addrtemp << 16) ;
 EXMACL = addrtemp;
}

// Internal MAC Address Configuration
void NAT_INMAC_Config(UINT8 *addr)
{
 UINT32 addrtemp;
 int i;

 addrtemp = 0;
 for (i=0;i<(int)MAC_ADDR_SIZE-2;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 INMACM = addrtemp;

 addrtemp = 0;
 for (i=(int)(MAC_ADDR_SIZE-2);i<(int)MAC_ADDR_SIZE;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 addrtemp = (addrtemp << 16) ;
 INMACL = addrtemp;
}

// Local Station MAC Address Configuration
void NAT_LSMAC_Config(int entry, UINT8 *addr)
{
 UINT32 addrtemp;
 int i;

 addrtemp = 0;
 for (i=0;i<(int)MAC_ADDR_SIZE-2;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 LSMACM(entry) = addrtemp;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
 addrtemp = 0;
 for (i=(int)(MAC_ADDR_SIZE-2);i<(int)MAC_ADDR_SIZE;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 addrtemp = (addrtemp << 16) ;
 LSMACL(entry) = addrtemp;
}

// Remote Station MAC Address Configuration
void NAT_RSMAC_Config(int entry, UINT8 *addr)
{
 UINT32 addrtemp;
 int i;

 addrtemp = 0;
 for (i=0;i<(int)MAC_ADDR_SIZE-2;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 RSMACM(entry) = addrtemp;

 addrtemp = 0;
 for (i=(int)(MAC_ADDR_SIZE-2);i<(int)MAC_ADDR_SIZE;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 addrtemp = (addrtemp << 16) ;
 RSMACL(entry) = addrtemp;
}

// Masquerading IP Address Configuration at entry x
void NAT_MASAD_Config(int entry, UINT8 *addr)
{
 UINT32 addrtemp;
 int i;

 addrtemp = 0;
 for (i=0;i<(int)IP_ADDR_SIZE;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 MASAD(entry) = addrtemp;
}

// Masquerading Port Number Configuration at entry x
void NAT_MASPN_Config(int entry, UINT16 *port)
{
 MASPN(entry) = *port;
}

// Local Station IP Address Configuration at entry x
void NAT_LSAD_Config(int entry, UINT8 *addr)
{
 UINT32 addrtemp;
 int i;

 addrtemp = 0;
 for (i=0;i<(int)IP_ADDR_SIZE;i++)
    addrtemp = (addrtemp << 8) | addr[i] ;
 LSAD(entry) = addrtemp;
}

// Local Station Port Number Configuration at entry x
void NAT_LSPN_Config(int entry, UINT16 *port)
{
 LSPN(entry) = *port;
}

// External MAC
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
UINT8 NATTBL_EXMAC[MAC_ADDR_SIZE] =
{0x00, 0x50, 0xBA, 0x33, 0xBE, 0x88};

// Internal MAC
UINT8 NATTBL_INMAC[MAC_ADDR_SIZE] =
{0x00, 0x50, 0xBA, 0x33, 0xBE, 0x44};

// Table Configuration
int NATTBL_NOP[NATTbl_Entry_Num] =
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, …, 0};

int NATTBL_Discard[NATTbl_Entry_Num] =
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, …, 0};

int NATTBL_Inverse[NATTbl_Entry_Num] =
{0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, …, 0};

int NATTBL_PxRE[NATTbl_Entry_Num] =
{1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, …, 0};

int NATTBL_AxRE[NATTbl_Entry_Num] =
{1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, …, 0};

int NATTBL_PxCE[NATTbl_Entry_Num] =
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, …, 0};

int NATTBL_AxCE[NATTbl_Entry_Num] =
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, …, 0};

int NATTBL_ExEN[NATTbl_Entry_Num] =
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, …, 0};

// MA
UINT8 NATTBL_MA[NATTbl_Entry_Num][IP_ADDR_SIZE] =
{{0x0A, 0x03, 0x93, 0x47},
 {0x0A, 0x03, 0x93, 0x37},
 {0x0A, 0x03, 0x93, 0x46},
 {0x0A, 0x03, 0x93, 0x36},
 {0x0A, 0x03, 0x93, 0x45},
 {0x0A, 0x03, 0x93, 0x35},
 {0x0A, 0x03, 0x93, 0x44},
 {0x0A, 0x03, 0x93, 0x34},
 {0x0A, 0x03, 0x93, 0x43},
 {0x0A, 0x03, 0x93, 0x33},
 {0x0A, 0x03, 0x93, 0x42},
 {0x0A, 0x03, 0x93, 0x32},
 {0x0A, 0x03, 0x93, 0x41},
 {0x0A, 0x03, 0x93, 0x31},
 {0x0A, 0x03, 0x93, 0x40},
 {0x0A, 0x03, 0x93, 0x30},
  …};

// MP
UINT16 NATTBL_MP[NATTbl_Entry_Num] =
{0x0047,
 0x0037,
 0x0046,
 0x0036,
 0x0045,
 0x0035,
 0x0044,
 0x0034,
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
 0x0043,
 0x0033,
 0x0042,
 0x0032,
 0x0041,
 0x0031,
 0x0040,
 0x0030,
  …};

// LA
UINT8 NATTBL_LA[NATTbl_Entry_Num][IP_ADDR_SIZE] =
{{0x0A, 0x03, 0x93, 0x57},
 {0x0A, 0x03, 0x93, 0x27},
 {0x0A, 0x03, 0x93, 0x56},
 {0x0A, 0x03, 0x93, 0x26},
 {0x0A, 0x03, 0x93, 0x55},
 {0x0A, 0x03, 0x93, 0x25},
 {0x0A, 0x03, 0x93, 0x54},
 {0x0A, 0x03, 0x93, 0x24},
 {0x0A, 0x03, 0x93, 0x53},
 {0x0A, 0x03, 0x93, 0x23},
 {0x0A, 0x03, 0x93, 0x52},
 {0x0A, 0x03, 0x93, 0x22},
 {0x0A, 0x03, 0x93, 0x51},
 {0x0A, 0x03, 0x93, 0x21},
 {0x0A, 0x03, 0x93, 0x50},
 {0x0A, 0x03, 0x93, 0x20},
  …};

// LP
UINT16 NATTBL_LP[NATTbl_Entry_Num] =
{0x0057,
 0x0027,
 0x0056,
 0x0026,
 0x0055,
 0x0025,
 0x0054,
 0x0024,
 0x0053,
 0x0023,
 0x0052,
 0x0022,
 0x0051,
 0x0021,
 0x0050,
 0x0020,
  …};

// LMAC
UINT8 NATTBL_LMAC[NATTbl_Entry_Num][MAC_ADDR_SIZE] =
{{0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x22},
  …};

// RMAC
UINT8 NATTBL_RMAC[NATTbl_Entry_Num][MAC_ADDR_SIZE] =
{{0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
 {0x00, 0x50, 0xBA, 0x33, 0xBE, 0x66},
  …};

void NAT_Table_Conf()
{
 int i;

 NAT_EXMAC_Config((UINT8 *)NATTBL_EXMAC);
 NAT_INMAC_Config((UINT8 *)NATTBL_INMAC);

 for (i=0;i<NATTbl_Entry_Num;i++)
    {
     NAT_LSMAC_Config(i, (UINT8 *)NATTBL_LMAC+MAC_ADDR_SIZE*i);
     NAT_RSMAC_Config(i, (UINT8 *)NATTBL_RMAC+MAC_ADDR_SIZE*i);
     NAT_MASAD_Config(i, (UINT8 *)NATTBL_MA+IP_ADDR_SIZE*i);
     NAT_MASPN_Config(i, (UINT16 *)NATTBL_MP+i);
     NAT_LSAD_Config(i, (UINT8 *)NATTBL_LA+IP_ADDR_SIZE*i);
     NAT_LSPN_Config(i, (UINT16 *)NATTBL_LP+i);
    }

 for (i=0;i<NATTbl_Entry_Num;i++)
    {
     if (NATTBL_AxCE[i])
       NAT_AddrCompare_Enable(i);
     if (NATTBL_PxCE[i])
       NAT_PortCompare_Enable(i);
     if (NATTBL_AxRE[i])
       NAT_AddrReplace_Enable(i);
     if (NATTBL_PxRE[i])
       NAT_PortReplace_Enable(i);
     if (NATTBL_Inverse[i])
       NAT_Inverse_Enable(i);
     if (NATTBL_Discard[i])
       NAT_Discard_Enable(i);
     if (NATTBL_NOP[i])
       NAT_NOP_Enable(i);
     if (NATTBL_ExEN[i])
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
       NAT_Entry_Enable(i);
    }
 NAT_Enable();
}

void MAC0_Rx_LISR(INT vector)
{
 UINT32           IntMACRxStatus;
 sFrameDescriptor *pRxFDptr;
 UINT32           RxStatus;
 UINT32           TempAddr;
 DV_DEVICE_ENTRY  *device;
 UINT32           *FDataPtr;
#ifdef USE_NETBUF
 UINT32           FrameLength ;
#endif

 IntMACRxStatus = MISTA_0;
 MISTA_0 = IntMACRxStatus & 0x0000ffff;

 if (IntMACRxStatus & MISTA_RxBErr)
   {
    FIFOTHD_0|=SWR;
    MacInitialize(MACNUM0);
   }
 else
   {
    if (IntMACRxStatus & (MISTA_CFR | MISTA_CRCE | MISTA_PTLE |
                          MISTA_ALIE | MISTA_RP))
      {
       if (IntMACRxStatus & MISTA_CFR)  gsMacRxStatus[MACNUM0].CFR++ ;
       if (IntMACRxStatus & MISTA_CRCE) gsMacRxStatus[MACNUM0].CRCE++ ;
       if (IntMACRxStatus & MISTA_PTLE) gsMacRxStatus[MACNUM0].PTLE++ ;
       if (IntMACRxStatus & MISTA_ALIE) gsMacRxStatus[MACNUM0].ALIE++ ;
       if (IntMACRxStatus & MISTA_RP)   gsMacRxStatus[MACNUM0].RP++ ;
      }
    else
      {
       TempAddr = CRXDSA_0;

       // Process all packets received.
       do
       {
        // Get Rx Frame Descriptor
        pRxFDptr = (sFrameDescriptor *)gCRxFDPtr[MACNUM0];

        // Check Receive Ownership
        if ((pRxFDptr->Status1|RXfOwnership_CPU)==RXfOwnership_CPU)
          {
           RxStatus = (pRxFDptr->Status1 >> 16) & 0xffff;

           if (RxStatus & RXFD_RXGD)
             {
              gsMacRxStatus[MACNUM0].RXGD++ ;

#ifdef USE_NETBUF
              if ((RxStatus&RXFD_NOP)||(RxStatus&RXFD_NATFSH))
                {
                 FrameLength = pRxFDptr->Status1 & 0xffff ;
                 FDataPtr =  (UINT32 *)pRxFDptr->FrameDataPtr;

                 pRxFDptr->FrameDataPtr = (UINT32)(NetBuf_Allocate0());
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                SendPacket(MACNUM0,FDataPtr,FrameLength);
                //SendPacket(MACNUM1,FDataPtr,FrameLength);  // for two MACs

                if (RxStatus & RXFD_NATFSH) gsMacRxStatus[MACNUM0].NATFSH++ ;
                if (RxStatus & RXFD_NOP)    gsMacRxStatus[MACNUM0].NOP++ ;
              }
            else
#endif
              {
               et_data_ptr0 =  (UINT32)pRxFDptr->FrameDataPtr;
               et_data_len0 = pRxFDptr->Status1 & 0xffff ;

               device = DEV_Get_Dev_By_Name("740MAC0");
               device->dev_receive(device);
              }
          }
        }
       else
        {
         if ((pRxFDptr->Status1&RXfOwnership_NAT)==RXfOwnership_NAT) // ownership=NAT
           {
            if (NATCheckFlag0==0)
              {
               NATCheckFlag0=1;
               break;
              }
            else
              NATCheckFlag0=0;
           }
         else // ownership=DMA
           {
            NATCheckFlag0=0;
            break;
           }
        }

       // Change ownership to DMA for next use
       pRxFDptr->Status1 = RXfOwnership_DMA;

       // Set Next Frame Descriptor pointer
       gCRxFDPtr[MACNUM0] = (UINT32)(pRxFDptr->NextFD);
      } while ((TempAddr = CRXDSA_0) != gCRxFDPtr[MACNUM0]);
     }
  if (IntMACRxStatus & MISTA_RDU)
    RSDR_0 = 0;
 }

}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10 USB Host Controller

10.1 Overview

The **Universal Serial Bus (USB)** is a low-cost, low-to mid-speed peripheral interface standard intended for modem, scanners, PDAs, keyboards, mice, and other devices that do not require a high-bandwidth parallel interface. 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 NUC740 USB *Host Controller* has the following features :

- **Open Host Controller Interface (OHCI)** Revision 1.0 compatible.
- USB Revision 1.1 compatible
- Supports both low-speed (1.5 Mbps) and full-speed (12Mbps) USB devices.
- Handles all the USB protocol.
- Built-in DMA for real-time data transfer
- Multiple low power modes for efficient power management

The *Host Controller Driver* has the following responsibilities :

- *Host Controller* Management
- Bandwidth Allocation
- List Management
- List Management
- *Root Hub* Management
- Multiple low power modes for efficient power management

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

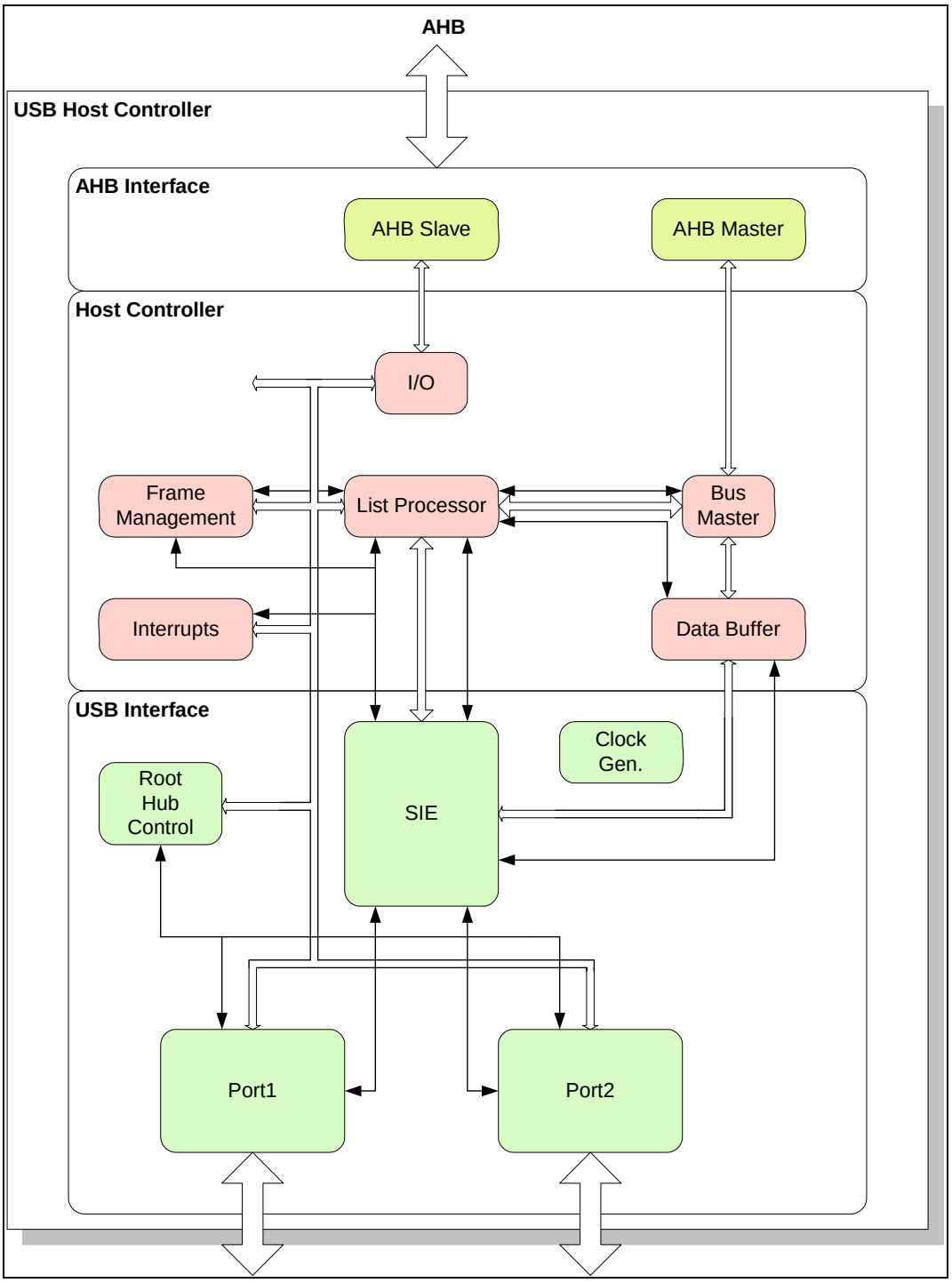
10.2 Registers Map

Register	Address	R/W	Description	Reset Value
OpenHCI Registers				
HcRevision	0xFFF0.5000	R	Host Controller Revision Register	0x0000.0010
HcControl	0xFFF0.5004	R/W	Host Controller Control Register	0x0000.0000
HcCommandStatus	0xFFF0.5008	R/W	Host Controller Command Status Register	0x0000.0000
HcInterruptStatus	0xFFF0.500C	R/W	Host Controller Interrupt Status Register	0x0000.0000
HcInterruptEnable	0xFFF0.5010	R/W	Host Controller Interrupt Enable Register	0x0000.0000
HcInterruptDisable	0xFFF0.5014	R/W	Host Controller Interrupt Disable Register	0x0000.0000
HcHCCA	0xFFF0.5018	R/W	Host Controller Communication Area Register	0x0000.0000
HcPeriodCurrentED	0xFFF0.501C	R/W	Host Controller Period Current ED Register	0x0000.0000
HcControlHeadED	0xFFF0.5020	R/W	Host Controller Control Head ED Register	0x0000.0000
HcControlCurrentED	0xFFF0.5024	R/W	Host Controller Control Current ED Register	0x0000.0000
HcBulkHeadED	0xFFF0.5028	R/W	Host Controller Bulk Head ED Register	0x0000.0000
HcBulkCurrentED	0xFFF0.502C	R/W	Host Controller Bulk Current ED Register	0x0000.0000
HcDoneHead	0xFFF0.5030	R/W	Host Controller Done Head Register	0x0000.0000
HcFmInterval	0xFFF0.5034	R/W	Host Controller Frame Interval Register	0x0000.2EDF
HcFrameRemaining	0xFFF0.5038	R	Host Controller Frame Remaining Register	0x0000.0000
HcFmNumber	0xFFF0.503C	R	Host Controller Frame Number Register	0x0000.0000
HcPeriodicStart	0xFFF0.5040	R/W	Host Controller Periodic Start Register	0x0000.0000
HcLSThreshold	0xFFF0.5044	R/W	Host Controller Low Speed Threshold Register	0x0000.0628
HcRhDescriptorA	0xFFF0.5048	R/W	Host Controller Root Hub Descriptor A Register	0x0100.0002
HcRhDescriptorB	0xFFF0.504C	R/W	Host Controller Root Hub Descriptor B Register	0x0000.0000
HcRhStatus	0xFFF0.5050	R/W	Host Controller Root Hub Status Register	0x0000.0000
HcRhPortStatus [1]	0xFFF0.5054	R/W	Host Controller Root Hub Port Status [1]	0x0000.0000
HcRhPortStatus [2]	0xFFF0.5058	R/W	Host Controller Root Hub Port Status [2]	0x0000.0000
USB Configuration Registers				
TestModeEnable	0xFFF0.5200	R/W	USB Test Mode Enable Register	0x0XXX.XXXX
OperationalModeEnable	0xFFF0.5204	R/W	USB Operational Mode Enable Register	0x0000.0000

According to the function of these registers, they are divided into four partitions, specifically for Control and Status, Memory Pointer, Frame Counter and *Root Hub*. All of the registers should be read and written as Dwords.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.3 Block Diagram



The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

AHB Master

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

AHB Slave

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

List Processing

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

Frame Management

Frame Management is responsible for managing the frame specific tasks required by the USB specification and the OpenHCI specification.

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.

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.

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.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.4 Data Structures

Except direct access to *Host Controller* by registers, *Host Controller Driver* must maintain the following memory blocks to communicate with *Host Controller* :

- *Endpoint Descriptor* Lists
- *Transfer Descriptor* Lists
- *Host Controller Communication Area (HCCA)*

Note1 : *All these data structures are located in system memory. The Host Controller will access these memory blocks by DMA transfer. All Endpoint Descriptors, Transfer Descriptors, HCCA, and transfer buffers must be set to non-cacheable region.*

Note2 : *Endpoint Descriptors and Transfer Descriptors must be aligned with 32 bytes address boundary. Host Controller Communication Area must be aligned with 256 bytes address boundary.*

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.4.1 Endpoint Descriptor (ED) Lists

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 :

Figure 10-1 Endpoint Descriptor Format

	31	26	16	15	14	13	12	11	10	07	06	05	04	03	02	01	00
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)													—			

The *Endpoint Descriptor* format of NUC740 USB *Host Controller* is compliant to OpenHCI Specification 1.0a. In this document, you can find detail descriptions about each field in *Endpoint Descriptor*.

The Control *ED* list is created by *Host Controller Driver* (*HCD*), which should add any new *ED*s 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 *ED*s.

Similarly, all Bulk *ED*s 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. That is, you may have 32 Interrupt *ED* lists., it The structure of Interrupt *ED* lists will be explained in the *HCCA* section.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

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 *ED*s 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*** register.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.4.2 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 Descriptor*s (*TD*s) to describe the details of a 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 Descriptor*s are linked to queues attached to *ED*s. The *ED* provides the endpoint address to/from where the *TD* data is to be transferred. The *Host Controller Driver* adds to the queue and the *Host Controller* removes from the queue. When the *Host Controller* removes a *TD* from a queue, it links the *TD* to the *Done Queue*. There are two *TD* types in OpenHCI, General *TD* and Isochronous *TD*. The *TD* formats are listed below :

Figure 10-2 General Transfer Descriptor Format

	31–28	27–26	25–24	23–21	20–19	18	03–00
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)						

Figure 10-3 Isochronous Transfer Descriptor Format

	31–28	27	26–24	23–21	20–16	15–12	11–05	04–00
Dword 0	CC	–	FC	DI	—	SF		
Dword 1	Buffer Page 0 (BP0)					—		
Dword 2	NextTD							0
Dword 3	Buffer End (BE)							
Dword 4	Offset1/PSW1					Offset0/PSW0		
Dword 5	Offset3/PSW3					Offset2/PSW2		
Dword 6	Offset5/PSW5					Offset4/PSW4		
Dword 7	Offset7/PSW7					Offset6/PSW6		

The General and Isochronous *Transfer Descriptor* formats of NUC740 USB *Host Controller* are compliant to OpenHCI Specification 1.0a. You can find detail descriptions about each field in a General/Isochronous *Transfer Descriptor* in this document.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

*Transfer Descriptor*s are created and filled by *HCD*. After receiving an *IRP* (I/O Request Packet) from USB Driver (refer to USB 1.1 Specification), according to the pipe, *HCD* must create appropriate number of *TD*s to describe the data transfer. For example, for a control pipe *IRP*, *HCD* may create three *TD*s for the SETUP stage, DATA stage, and STATUS stage transfers. The *TD*s must be linked to *TD* list of the corresponding *ED* of the pipe specified in the original *IRP*. The *TD* list of an *ED* is maintained by the *HeadP* and *TailP* fields of the *ED* itself.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.4.3 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*.

Table 10-1 HCCA (Host Controller Communication Area)

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 HC

The *Host Controller Communication Area* format of NUC740 USB *Host Controller* is compliant to OpenHCI Specification 1.0a. Detail descriptions about each field in *HCCA* can be found in this document.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5 Programming Note

This section will demonstrate how to write a *Host Controller Driver*, including

- Initialization,
- Lists management,
- Interrupt processing, and
- *Root hub* management.

10.5.1 Initialization

The initialization of *Host Controller* contain the following steps :

1. Disable *Host Controller* interrupts by setting ***MasterInterruptEnable*** bit of ***HcInterruptDisable*** register.
2. Issue a software reset command by setting ***HostControllerReset*** bit of ***HcCommandStatus*** register and waiting for 10ms until the read value of ***HostControllerReset*** become **0.**
3. Allocate and create all necessary list structures and memory blocks, including *HCCA*, and empty all driver-maintained lists, including *InterruptTable* of *HCCA* (Note that *HCCA* must be aligned with *256-bytes* address boundary, while *ED*s and *TD*s must be aligned with *32-bytes* address boundary).
4. Clear ***HcControlHeadED*** and ***HcBulkHeadED*** register
5. Program the physical address of driver-allocated *HCCA* to ***HcHCCA*** register
6. Programe frame interval value (11,999 $\pm$ 6) to ***HcFmInterval*** register and 90% of this value (recommended) to ***HcPeriodicStart*** register
7. Programe *0x628* to ***HcThreshold*** register (0x628 is the reset default value of ***HcThreshold*** register)
8. Enable all transfer list by setting ***PeriodicListEnable***, ***IsochronousEnable***, ***ControlListEnable***, and ***BulkListEnable*** bits of ***HcControl*** register
9. Let *Host Controller* transit to *USBOPERATIONAL* state by writing *10b* to ***HostControllerFunctionalState*** field of ***HcControl*** register
10. Enable desired interrupts by programming corresponding bits to ***HcInterruptEnable*** register and clear interrupt status of these interrupts by programming corresponding bits to ***HcInterruptStatus*** register
11. Turn on the *Root Hub* port power by issuing *SetGlobalPower* command (writing *0x10000* to ***HcRhStatus*** register) (Note that NUC740 USB *Root Hub* uses global power switching mode)

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

12. Enable NUC740 **AIC** (Advanced Interrupt Controller) USB interrupt, which is *IRQ9*
13. Connect Hub device driver

10.5.2 USB States

The *Host Controller* has four USB states visible to the *Host Controller Driver* via the Operational Registers : USBOPERATIONAL, USBRESET, USBSUSPEND, and USBRESUME. These USB states are stored in the ***HostControllerFunctionalState*** field of the ***HcControl*** register. The *Host Controller Driver* can perform some state transitions by modifying the ***HostControllerFunctionalState*** field of ***HcControl*** register. The meanings of two bits ***HostControllerFunctionalState*** field is listed in Table 10-2.

Table 10-2 HostControllerFunctionalState

HostControllerFunctionalState	
00b	USBRESET
01b	USBRESUME
10b	USBOPERATIONAL
11b	USBSUSPEND

You can find possible transitions of USB states in Table 10-3. The followings are some notes about the USB state transitions :

- After hardware reset, the *Host Controller* will enter USBRESET state.
- In any state, by programming 1 to the ***HostControllerReset*** bit ***HcCommandStatus*** register, a software reset command will force the *Host Controller* to perform software reset. After software reset, the HC will enter USBSUSPEND state, instead of USBRESET state.
- If HC is in USBSUSPEND state, it will enter USBRESUME state either by *HCD* writing 0x1 to ***HostControllerFunctionalState*** or by remote wakeup. To enable HC resume by remote wakeup, *HCD* must enable the ***DeviceRemoteWakeupEnable*** bit of ***HcRhStatus*** register. *HCD* can enable ***ResumeDetected*** interrupt to sense the case.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

Table 10-3 USB state transition table

From state	*Convert to state*	*Conditions*
	USBRESET	Hardware Reset
USBRESET	USBOPERATIONAL	Writing 0x2 to ***HostControllerFunctionalState***
	USBSUSPEND	1. Writing 0x3 to ***HostControllerFunctionalState*** 2. Issue a Software Reset command
USBOPERATIONAL	USBSUSPEND	***1.*** Writing 0x3 to ***HostControllerFunctionalState*** 2. Issue a Software Reset command
	USBRESET	Writing 0 to ***HostControllerFunctionalState***
USBSUSPEND	USBRESUME	1. Writing 0x3 to ***HostControllerFunctionalState*** 2. Resumed by device
	USBOPERATIONAL	Writing 0x2 to ***HostControllerFunctionalState***
	USBRESET	Writing 0 to ***HostControllerFunctionalState***
USBRESUME	USBOPERATIONAL	Writing 0x2 to ***HostControllerFunctionalState***
	USBRESET	Writing 0 to ***HostControllerFunctionalState***

10.5.3 Add/Remove Endpoint Descriptors

In *Host Controller* architecture, a device endpoint is described by an *ED* (*Endpoint Descriptor*). *Host Controller* has Control, Bulk, Interrupt, and Isochronous *Endpoint Descriptor* lists. The Control and Bulk *ED* lists are referred to by ***HcControlHeadED*** register and ***HcBulkHeadED*** register respectively. The Interrupt endpoints are organized into 32 Interrupt *ED* lists with each list pointed by one of the *HCCA* InterruptTable entries. The Isochronous *ED* list is linked to the last *ED* of the Interrupt *ED* list.

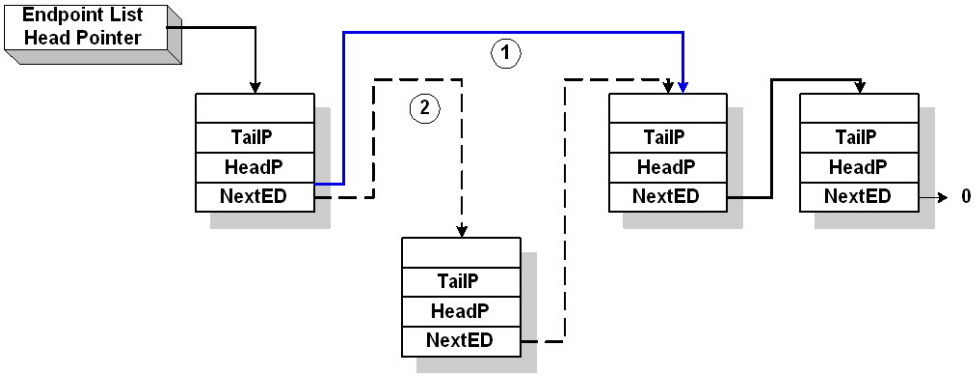
While receiving an *IRP* from USB Driver, *HCD* must identify the target endpoint of this *IRP* and try to find out the *ED* corresponding to the endpoint. If the *ED* does not exist, *HCD* must create a new *ED* and link it to the appropriate *ED* list. The *Endpoint Descriptor*s in an *ED* list are linked together by the *NextED* field of each *ED*. Each *NextED* links to the very next *ED* in an *ED* list. The *NextED* of the last *Endpoint Descriptor* must points to *zero* to signify the end of an *ED* list.

To add an *Endpoint Descriptor* to an *ED* list, *HCD* needs to program the physical address of the new *ED* to the *NextED* of the last *ED* and write *zero* to the *NextED* of the new *ED*.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

To remove an *Endpoint Descriptor*, *HCD* must find the previous *ED* of the *ED* to be removed. *HCD* modified the *NextED* of the previous *ED* to point to the next *ED* of the *ED* to be removed, or clear it if the *ED* to be removed is the last *ED*. However, before removing an *ED*, *HCD* must make sure that *Host Controller* is not processing this *ED* and there's no any *TD* waiting for service. *HCD* can temporarily disable the processing of target *ED* list by configuring ***HcControl*** register and enable the SOF interrupt. In the next the SOF interrupt, *HCD* can guarantee that the *ED* list is not processed by *Host Controller*.

Figure 10-4 Remove an Endpoint Descriptor



10.5.4 Add/Remove Transfer Descriptors

The diagram of *Endpoint Descriptor* list linked with *Transfer Descriptor* queue is shown inFigure 10-1. The *Transfer Descriptor* queue of an *ED* is linked by its *HeadP* and *TailP* field.

According to OpenHCI specification, if *HeadP* is equal to *TailP*, then the *TD* queue is configured as empty. In Figure 10-2, the *HeadP* and *TailP* pointer of last *ED* has pointed to the same *Transfer Descriptor*, that is, the *TD* queue of this *ED* is empty. The *TD* there under the *ED* is a dummy *TD*. While creating a new *ED*, *HCD* must also create a dummy *TD* for it, and let both *HeadP* and *TailP* pointer point to this dummy *TD*.

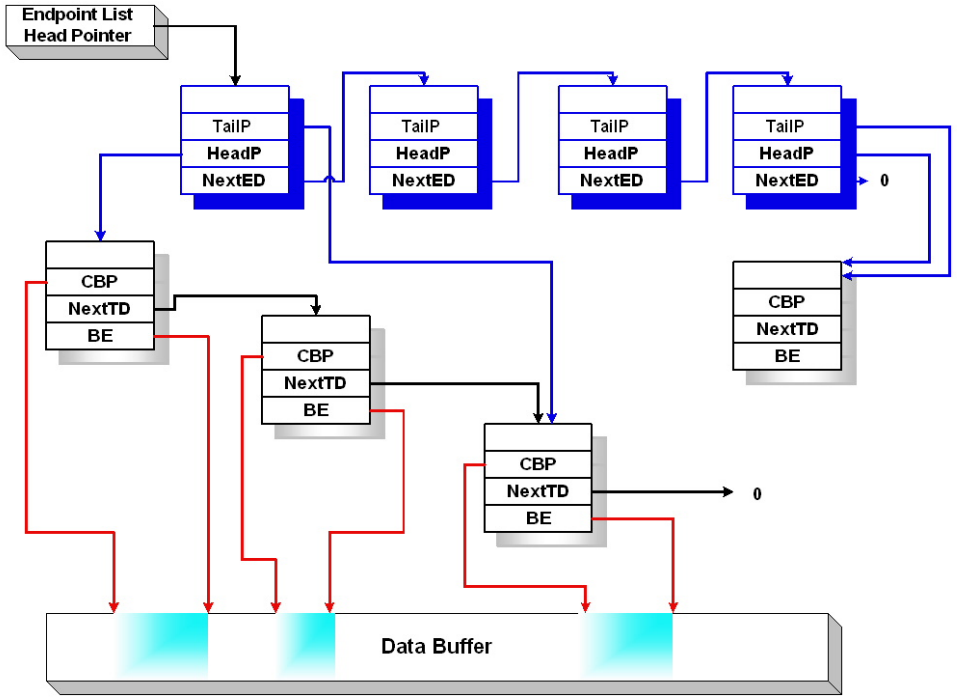
To add a new *TD* into the *TD* queue, *HCD* can use the dummy *TD*. *HCD* writes information and data buffer link to the dummy *TD*, and creates a new dummy *TD*. This can be accomplished by the following steps :

1. Writing *TD* information and data buffer link to the current dummy *TD*
2. Creating a new dummy *TD*
3. Let the *NextTD* of the current *TD* point to the new dummy *TD*

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

4. Let TailP of *ED* point to the new dummy *TD*

Figure 10-5 ED list and TD queue



Once the *Host Controller* has accomplished the processing of a *TD*, in spite of success or failure, *Host Controller* will remove the *TD* from *TD* queue and put it into the *Done Queue*. *Host Controller* will follow the following steps to remove a *TD* :

1. Modify the *HeadP* pointer of the *Endpoint Descriptor* (HC always service the first *TD* of the *TD* queue) to link to the next *TD*, HC can obtain the link of the next *TD* by reading the *NextTD* pointer of the first *TD*
2. Now the *TD* has been unlinked from *TD* queue, HC moves the *TD* to the head of *Done Queue*, which is pointed by ***HcDoneHead*** register
3. HC moves the retired *TD* to the *Done Queue* by writing the contains of ***HcDoneHead*** to the *NextTD* field of the retired *TD*, and then have the ***HcDoneHead*** point to the retired *TD*

In some situation, the client software may cancel an *IRP* before the *IRP* was completed. This

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

would result in canceling the *TD*s, which has been created for the *IRP*. To canceling *TD*s, *HCD* must ensure that the *TD*s are not being processed by HC. To achieve this, *HCD* can set the *skip* bit of the *Endpoint Descriptor* and wait for the next SOF by enabling SOF interrupt. If *HCD* can ensure the target endpoint is temporary skipped by HC, it can safely remove any *TD*s of the endpoint. After removing the *TD*s, *HCD* can clear the *sKip* bit and enable processing on the endpoint again.

In some other situation, transfer errors or endpoint stall may make an endpoint being halted, then *HCD* must remove the residual *TD*s. Under these situations, *Host Controller* will stop processing on this endpoint, because the *Halted* bit has been set. Thus, *HCD* can safely remove the *TD*s. After removing the *TD*s and overcoming the error conditions, *HCD* can clear *Halted* bit and enable processing on the endpoint again.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.5 IRP Processing

The data structure of *IRP* is operation system dependent. *Host Controller driver* should be able to interpret the content of any *IRP*. The processing on *IRP*s are different for each transfer type.

10.5.5.1 Control Transfer

For Control Transfer, *HCD* may create two or three *TD*s for a Control Transfer, depend on specific request command. For a request command without DATA stage, *HCD* will create two *TD*s for it. The first *TD* is created for SETUP stage, which has DATA0 toggle setting. It must have an eight bytes data buffer to accommodate the request command. The second *TD* is created for STATUS stage, which has DATA1 toggle setting. It must contain a zero bytes buffer.

For a request command with DATA stage, *HCD* will create three *TD*s for it. The first *TD* is created for SETUP stage, which has DATA0 toggle setting and has an eight bytes buffer to accommodate the request command. The second *TD* is created for DATA stage, which has DATA1 toggle setting and has a data buffer to accommodate the transferred data for this command. The third *TD* is created for STATUS stage, which has DATA1 toggle setting. It must contain a zero bytes buffer.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

The following is an example code of Control Transfer :

```
info = TD_CC | TD_DP_SETUP | TD_T_DATA0;
td_fill(info, ctrl, 8, urb, cnt++);
if (data_len > 0)
{
    info = usb_pipeout(urb->pipe)? (TD_CC | TD_R | TD_DP_OUT | TD_T_DATA1)  :
                                   (TD_CC | TD_R | TD_DP_IN | TD_T_DATA1);
    td_fill(info, data, data_len, urb, cnt++);
}
info = usb_pipeout(urb->pipe)? (TD_CC | TD_DP_IN | TD_T_DATA1) :
                               (TD_CC | TD_DP_OUT | TD_T_DATA1);
td_fill(info, NULL, 0, urb, cnt++);
writel(OHCI_CLF, &ohci->regs->HcCommandStatus); /* start Control list */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.5.2 Bulk Transfer

The maximum buffer size for a bulk *Transfer Descriptor* is 4096 bytes. Thus, for a Bulk Transfer, *HCD* simply generates a *TD* for each 4096 bytes data length. For example, if the transfer buffer length of an *IRP* is 9KB, then *HCD* will generate three *TD*s for this *IRP*.

Because OHCI handles the data toggles by itself, it just deed to set the toggle bits for the first *TD*. The data toggle setting of the subsequent *TD*s were processed by OHCI controller. OHCI controller can get the toggle value from the DataToggle bit of *Endpoint Descriptor*.

The following is an example code of Bulk Transfer :

```
info = usb_pipeout(urb->pipe)? (TD_CC | TD_DP_OUT)  :  (TD_CC | TD_DP_IN);
while(data_len > 4096)
{
    td_fill(info | (cnt? TD_T_TOGGLE : toggle), data, 4096, urb, cnt);
    data = (VOID *)((UINT32)data + 4096);
    data_len -= 4096;
    cnt++;
}
info = usb_pipeout(urb->pipe)? (TD_CC | TD_DP_OUT)  :  (TD_CC | TD_R | TD_DP_IN);
td_fill(info | (cnt? TD_T_TOGGLE : toggle), data, data_len, urb, cnt);
cnt++;
writel (OHCI_BLF, &ohci->regs->HcCommandStatus); /* start bulk list */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.5.3 Interrupt Transfer

The maximum buffer size for an Interrupt *Transfer Descriptor* is 64 bytes. The USB Client Software will not deliver an *IRP* with transfer length exceeding 64 bytes. *HCD* makes only one *TD* for an Interrupt Transfer, which is one-shot. On completion of the *TD*, *HCD* may re-submit the identical *TD* to implement the next Interrupt Transfer. Thus it can fulfill the periodic polling of an Interrupt Endpoint.

The following is an example code of Interrupt Transfer :

```
info = usb_pipeout (urb->pipe)? (TD_CC | TD_DP_OUT | toggle)  :
                            (TD_CC | TD_R  | TD_DP_IN | toggle);
td_fill(info, data, data_len, urb, cnt++);
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.5.4 Isochronous Transfer

An Isochronous *TD* may contain one to eight consecutive packets with specified starting frame number. Depending on implementation of operating system, several isochronous packets to be transferred may be carried in a single *IRP*. *HCD* must prepare appropriate Isochronous *TD*s for these isochronous packets. For example, in Linux's implementation, *HCD* will generate a single Isochronous *TD* for each isochronous packet. According to the starting frame specified in the *IRP*, *HCD* will increase the starting frame of each consecutive Isochronous *TD*. The transfer length of each isochronous packet is specified by Client Software and should not be larger than 1023 bytes.

The following is an example code of Isochronous Transfer :

```
for (cnt = 0; cnt < urb->number_of_packets / ISO_FRAME_COUNT; cnt++)
{
    iso_td_fill(TD_CC | ((ISO_FRAME_COUNT - 1) << 24) | TD_ISO |
            ((urb->start_frame + cnt * ISO_FRAME_COUNT) & 0xffff ),
               (UINT8 *) data, urb, cnt);
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.6 Interrupt Processing

NUC740 USB *Host Controller* may raise the following interrupts :

- SchedulingOverrun
- WritebackDoneHead
- StartOfFrame
- ResumeDetected
- UnrecoverableError
- FrameNumberOverflow
- RootHubStatusChange
- OwnershipChange

10.5.6.1 SchedulingOverrun 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.

10.5.6.2 WritebackDoneHead Interrupt

This interrupt is set after *Host Controller* has written ***HcDoneHead*** 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 *TD*s were retired in stack order. Then *HCD* can start processing on each *TD*. More detailed description is introduced in the next section.

10.5.6.3 StartOfFrame Interrupt

This interrupt is set at 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 ***StartOfFrame*** interrupt. In the next coming StartOfFrame interrupt, *HCD* can ensure that the endpoint is not currently processed by *Host Controller*, and it can remove the *TD*.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.6.4 ResumeDetected 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. Note that if you want to make ***ConnectStatusChange*** event being treated as a resume event, you must have written a ***SetRemoteWakeupEnable*** command to ***HcRhStatus*** register.

10.5.6.5 UnrecoverableError 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.

10.5.6.6 FrameNumberOverflow Interrupt

The *Host Controller* will raise this interrupt when the MSB bit of ***FrameNumber*** (bit 15) of ***HcFmNumber*** register toggles value from 0 to 1 or 1 to 0, and after ***HccaFrameNumber*** 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 ***FrameNumberOverflow*** interrupt.

10.5.6.7 RootHubStatusChange Interrupt

When the ***OverCurrentIndicatorChange*** bit of ***HcRhStatus*** register, or the ***ConnectStatusChange***, ***PortEnableStatusChange***, ***PortSuspendStatusChange***, or ***PortResetStatusChange*** bit of ***HcRhPortStatus[1/2]*** set, the *Host Controller* would raise the ***RootHubStatusChange*** interrupt.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.6.8 OwnershipChange Interrupt

This *Host Controller* would raise this interrupt when *HCD* set the ***OwnershipChangeRequest*** field of ***HcCommandStatus*** register. Due to the characteristics of embedded system, almost all applications would not have a SMM driver, the ***OwnershipChange*** interrupt would not be used.

The following is an example implementation of *Host Controller* interrupt service routine :

```
VOID hc_interrupt(int vector)
{
    OHCI_T  *ohci = _NUC740_OHCI;
    OHCI_REGS_T *regs = ohci->regs;
    INT     ints;

    _InUsbInterrupt = 1;

    ints = ohci->regs->HcInterruptStatus;

    if ((ohci->hcca->done_head != 0) &&
        !((UINT32)(ohci->hcca->done_head) & 0x01))
    {
        ints =  OHCI_INTR_WDH;
    }
    else if ((ints = (readl(&regs->HcInterruptStatus) &
                      readl(&regs->HcInterruptEnable))) == 0)
    {
        USB_printf("Not the wanted interrupts :  %x\n", ints);
    }

    if (ints & OHCI_INTR_UE)
    {
        ohci->disabled++;
        USB_printf("Error! - OHCI Unrecoverable Error, controller disabled\n");
        hc_reset (ohci);
    }

    if (ints & OHCI_INTR_WDH)
    {
        writel(OHCI_INTR_WDH, &regs->HcInterruptDisable);
        dl_done_list(ohci, dl_reverse_done_list (ohci));
        writel(OHCI_INTR_WDH, &regs->HcInterruptEnable);
    }

    if (ints & OHCI_INTR_SO)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    {
```

```
        USB_printf("Error! - USB Schedule overrun, count : %d\n",
                    (readl(&ohci->regs->HcCommandStatus) >> 16) & 0x3);
        writel(OHCI_INTR_SO, &regs->HcInterruptEnable);
    }

    if (ints & OHCI_INTR_SF)
    {
        UINT32 frame = ohci->hcca->frame_no & 1;

        writel(OHCI_INTR_SF, &regs->HcInterruptDisable);
        if (ohci->ed_rm_list[!frame] != NULL)
        {
            dl_del_list(ohci, !frame);
        }
        if (ohci->ed_rm_list[frame] != NULL)
            writel(OHCI_INTR_SF, &regs->HcInterruptEnable);
    }

    writel(ints, &regs->HcInterruptStatus);
    writel(OHCI_INTR_MIE, &regs->HcInterruptEnable);

    _InUsbInterrupt = 0;
}
```

10.5.7 Done Queue Processing

The *Done Queue* is built by the *Host Controller* and referred to by the ***HcDoneHead*** register. No matter successful or failed, the retired *Transfer Descriptor*s 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 ***WritebackDoneHead*** interrupt. *HCD* can take the *Done Queue* by servicing the ***WritebackDoneHead*** interrupt.

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.7.1 Reverse Done Queue

Note that the *TD*s are queued into the *Done Queue* in stack 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 *TD*s. The following is an example routine of reversing *Done Queue* :

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
static TD_T  *dl_reverse_done_list(OHCI_T * ohci)
{
    UINT32  td_list_hc;
    TD_T    *td_rev = NULL;
    TD_T    *td_list = NULL;
    URB_PRIV_T *urb_priv = NULL;

    td_list_hc = (UINT32)(ohci->hcca->done_head) & 0xfffffff0;
    ohci->hcca->done_head = 0;

    while (td_list_hc)
    {
        td_list = (TD_T *)td_list_hc;

        if (TD_CC_GET((UINT32)td_list->hwINFO))
        {
            urb_priv = (URB_PRIV_T *)td_list->urb->hcpriv;
            TD_CompletionCode(TD_CC_GET((UINT32)(td_list->hwINFO)));
            if (td_list->ed->hwHeadP & 0x1)
            {
                if (urb_priv && ((td_list->index + 1) < urb_priv->length))
                {
                    td_list->ed->hwHeadP =
                   (urb_priv->td[urb_priv->length - 1]->hwNextTD & 0xfffffff0) |
                   (td_list->ed->hwHeadP & 0x2);
                    urb_priv->td_cnt += urb_priv->length - td_list->index - 1;
                }
                else
                    td_list->ed->hwHeadP &= 0xfffffff2;
            }
        }

        if ((td_list->ed->type == PIPE_ISOCHRONOUS) &&
            (td_list->hwPSW[0] >> 12) &&
            ((td_list->hwPSW[0] >> 12) != TD_DATAUNDERRUN))
        {
```

```
            /* PSW error */
            TD_CompletionCode(td_list->hwPSW[0] >> 12);
        }

        td_list->next_dl_td = td_rev;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        td_rev = td_list;
        td_list_hc = (UINT32)(td_list->hwNextTD) & 0xfffffff0;
    }  /* end of while */

    return td_list;
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.7.2 Processing Done Queue

Now that the *Done Queue* is inversed into its original order, *HCD* can start to process the *TD*s one by one. For each *TD*, *HCD* checks whether the *TD* was completed with any errors. While processing each *TD*, the *HCD* must determine whether an *IRP* is completed. *HCD* recorded the *TD*s linked to a specific *IRP* and would know whether all *TD*s belong to the same *IRP* have been completed. In the example code, once all *TD*s of an *IRP* have been completed, *HCD* would invoke ***sochi_return_urb( )*** to reclaim the *IRP*.

In the ***sochi_return_urb( )*** routine, *HCD* would invoke the complete routine of the *IRP*. The complete routine is a callback routine provided by *Client Software* or *USB Driver* and used to notify the completion of an *IRP*. The *Client Software* or *USB Driver* may collect data received by *Host Controller* or do nothing, it depends on the implementation. In addition to invoke complete routine, *HCD* may release the *IRP* or re-submit the *IRP* if it is the Interrupt Transfer type.

The following is an example implementation of *Done Queue* consuming routine :

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
static VOID  dl_done_list(OHCI_T *ohci, TD_T *td_list)
{
    TD_T    *td_list_next = NULL;
    ED_T    *ed;
    INT     cc = 0;
    URB_T   *urb;
    URB_PRIV_T *urb_priv;
    UINT32  tdINFO, edHeadP, edTailP;

    while (td_list)
    {
        td_list_next = td_list->next_dl_td;

        urb = td_list->urb;
        urb_priv = urb->hcpriv;
        tdINFO = (UINT32)(td_list->hwINFO);

        ed = td_list->ed;

        dl_transfer_length(td_list);

        /* error code of transfer */
        cc = TD_CC_GET (tdINFO);
        if (cc == TD_CC_STALL)
            usb_endpoint_halt(urb->dev, usb_pipeendpoint(urb->pipe),
                              usb_pipeout(urb->pipe));

        if (!(urb->transfer_flags & USB_DISABLE_SPD) && (cc == TD_DATAUNDERRUN))
            cc = TD_CC_NOERROR;

        if (++(urb_priv->td_cnt) == urb_priv->length)
        {
            if ((ed->state & (ED_OPER | ED_UNLINK)) &&
                    (urb_priv->state != URB_DEL))
            {
                urb->status = cc_to_error[cc];
                sohci_return_urb(urb);
            }
            else
                dl_del_urb(urb);
        }
        if (ed->state != ED_NEW)
        {
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
            edHeadP = (UINT32)(ed->hwHeadP) & 0xfffffff0;
            edTailP = (UINT32)(ed->hwTailP);

            /* unlink eds if they are not busy */
            if ((edHeadP == edTailP) && (ed->state == ED_OPER))
                ep_unlink(ohci, ed);
        }
        td_list = td_list_next;
    }
}


static INT sohci_return_urb(URB_T * urb)
{
    URB_PRIV_T  *urb_priv = urb->hcpriv;
    URB_T  *urbt;
    INT    i;

    if (!urb_priv)
        return -1; /* urb already unlinked */
    /* just to be sure */
    if (!urb->complete)
    {
        urb_rm_priv(urb);
        return -1;
    }

    switch (usb_pipetype (urb->pipe))
    {
        case PIPE_INTERRUPT :
            urb->complete(urb);    /* call complete and requeue URB */
            urb->actual_length = 0;
            urb->status = USB_ST_URB_PENDING;
            if (urb_priv->state != URB_DEL)
                td_submit_urb(urb);
            break;

        case PIPE_ISOCHRONOUS :
            for (urbt = urb->next; urbt && (urbt != urb); urbt = urbt->next);
            if (urbt)
            {                           /* send the reply and requeue URB */
                urb->complete(urb);
                urb->actual_length = 0;
                urb->status = USB_ST_URB_PENDING;
                urb->start_frame = urb_priv->ed->last_iso + 1;
                if (urb_priv->state != URB_DEL)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                {
                    for (i = 0; i < urb->number_of_packets; i++)
                    {
                        urb->iso_frame_desc[i].actual_length = 0;
                        urb->iso_frame_desc[i].status = -EXDEV;
                    }
                    td_submit_urb (urb);
                }
            }
            else
            {                           /* unlink URB, call complete */
                urb_rm_priv (urb);
                urb->complete (urb);
            }
            break;

        case PIPE_BULK :
        case PIPE_CONTROL :  /* unlink URB, call complete */
            urb_rm_priv(urb);
            urb->complete(urb);
            break;
    }
    return 0;
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.8 Root Hub

The *Root Hub* is integrated into *Host Controller* and the control of *Root Hub* is done by accessing register files. NUC740 *Host Controller* has provided several *Root Hub* related registers. The ***HcRhDescriptorA*** and ***HcRhDescriptorB*** registers are informative registers, which are used to describe the characteristics and capabilities of *Root Hub*. The ***HcRhStatus*** register presents the current status and reflects the change of status of *Root Hub*. The ***HcRhPortStatus*** register presents the current status and reflects the change of status of a *Root Hub* port. NUC740 *Root Hub* has two hub ports, the ***HcRhPortStatus[1]*** and ***HcRhPortStatus[2]*** are respectively dedicated to port 0 and port 1.

10.5.8.1 HcRhDescriptorA and HcRhDescriptorB

The ***HcRhDescriptorA*** and ***HcRhDescriptorB*** registers are informative registers, which are used to describe the characteristics and capabilities of *Root Hub*. The characteristics and capabilities of NUC740 *Root Hub* are listed in the followings :

- Two downstream ports
- Ports are power switched
- Power switching mode is global power switch
- Is not a compound device
- Over-current status is reported collectively for all downstream ports
- Power-on-to-power-good-time is 2ms
- Devices attached to any ports are removable

10.5.8.2 HcRhStatus

The ***HcRhStatus*** 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 bit 0 to turn off power to all ports
- ***SetRemoteWakeupEnable*** – write '1' to bit 15 to enable device remote wakeup
- ***SetGlobalPower*** – write '1' to bit 16 to turn off power to all ports
- ***ClearRemoteWakeupEnable*** – write '1' to bit 31 to disable device remote wakeup
  In addition, the ***HcRhStatus*** register also indicates the following status :
- ***OverCurrentIndicator*** – bit 2 indicates the over-current status
- ***DeviceRemoteWakeupEnable*** – bit 15 indicates the remote wakeup status. If this bit was set, the ConnectStatusChange is determined as a remote wakeup event
- ***OverCurrentIndicatorChange*** – This bit was set when the ***OverCurrentIndicator*** bit

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

changed

10.5.8.3 HcRhPortStatus[1] and HcRhPortStatus[2]

The ***HcRhPortStatus[1]*** and ***HcRhPortStatus[1]*** register is used to control and monitor the *Root Hub* port status. ***HcRhPortStatus[1]*** is dedicated to port 0 and ***HcRhPortStatus[2]*** dedicated to port 1 respectively. The lower word 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 bit 0 to clear the ***PortEnableStatus*** bit
- ***SetPortEnable*** – write '1' to bit 1 to set the ***PortEnableStatus*** bit
- ***SetPortSuspend*** – write '1' to bit 2 to clear the ***PortSuspendStatus*** bit
- ***ClearSuspendStatus*** – write '1' to bit 3 to clear the ***PortSuspendStatus*** bit
- ***SetPortReset*** – write '1' to bit 4 to set port reset signaling
- ***SetPortPower*** – write '1' to bit 8 to set the ***PortPowerStatus*** bit
- ***ClearPortPower*** – write '1' to bit 9 to clear the ***PortPowerStatus*** bit

You can get the current status of the *Root Hub* port by reading the following bits :

- ***CurrentConnectStatus*** – bit 0, reflect the current connect status of the *Root Hub* port
- ***PortEnableStatus*** – bit 1, indicate whether the port is enabled or disabled
- ***PortSuspendStatus*** – bit 2, indicate the port is suspended or not
- ***PortResetStatus*** – bit 4, indicate the *Root Hub* is asserting reset signal on this port
- ***PortPowerStatus*** – bit 8, reflect the port's power status
- ***LowSpeedDeviceAttached*** – bit 9, indicate the speed of the device attached to this port

The following bits indicate the change of status bits. Write '1' to these bits will clear the events :

- ***ConnectStatusChange*** – bit 16, indicate the connect or disconnect events
- ***PortEnableStatusChange*** – bit 17, set when hardware event clear the ***PortEnableStatus*** bit
- ***PortSuspendStatusChange*** – bit 18, set when the full resume sequence has been completed
- ***PortResetStatusChange*** – bit 20, set at the end of the 10-ms port reset signal

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

10.5.8.4 Virtual Root Hub

Obviously, the *Root Hub* control is quite different from a hub device. The hardware dependent parts of *Root Hub* will increase the complexity of implementing the USB Driver. Thus, to make the *Root Hub* appear as a normal hub device seems be a better solution. To accomplish this, *HCD* must be able to determine the standard request to *Root Hub* and reply the request to make the upper layer software feel like that they are communicating with a real hub device.

First, the standard request to the *Root Hub* must be intercepted. Refer to the following code :

```
static INT  sohci_submit_urb(URB_T * urb)
{
    /* some code assertted here */
    ... ... ...
    ... ... ...

    /* handle a request to the virtual root hub */
    if (usb_pipedevice(pipe) == ohci->rh.devnum)
        return rh_submit_urb(urb);

    /* some code assertted here */
    ... ... ...
    ... ... ...
}
```

As it illustrated, all *IRP*s are forwarded to *HCD*'s ***sohci_submit_urb( )*** routine. This routine will further translate *IRP*s into *Transfer Descriptor*s, which finally make *Host Controller* issue transactions on USB bus. But *Root Hub* is not a real device on USB bus, it's embedded in *Host Controller* itself. So, in the previous program segment, the requests to the *Root Hub* must be intercepted and forwarded them to the dedicated routine ***rh_submit_urb( )***. The followings code are the sample implementation of ***rh_submit_urb( ) for illustration :***

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
/* Device descriptor */
static UINT8 root_hub_dev_des[] =
{
        0x12,       /*  UINT8  bLength; */
        0x01,       /*  UINT8  bDescriptorType; Device */
        0x10,       /*  UINT16 bcdUSB; v1.1 */
        0x01,
        0x09,       /*  UINT8  bDeviceClass; HUB_CLASSCODE */
        0x00,       /*  UINT8  bDeviceSubClass; */
        0x00,       /*  UINT8  bDeviceProtocol; */
        0x08,       /*  UINT8  bMaxPacketSize0; 8 Bytes */
        0x00,       /*  UINT16 idVendor; */
        0x00,
        0x00,       /*  UINT16 idProduct; */
        0x00,
        0x00,       /*  UINT16 bcdDevice; */
        0x00,
        0x00,       /*  UINT8  iManufacturer; */
        0x02,       /*  UINT8  iProduct; */
        0x01,       /*  UINT8  iSerialNumber; */
        0x01        /*  UINT8  bNumConfigurations; */
};


/* Configuration descriptor */
static UINT8 root_hub_config_des[] =
{
        0x09,       /*  UINT8  bLength; */
        0x02,       /*  UINT8  bDescriptorType; Configuration */
        0x19,       /*  UINT16 wTotalLength; */
        0x00,
        0x01,       /*  UINT8  bNumInterfaces; */
        0x01,       /*  UINT8  bConfigurationValue; */
        0x00,       /*  UINT8  iConfiguration; */
        0x40,       /*  UINT8  bmAttributes;
                        Bit 7 :  Bus-powered, 6 :  Self-powered, 5 Remote-wakwup,
                        4..0 :  resvd */
        0x00,       /*  UINT8  MaxPower; */

        /* interface */
        0x09,       /*  UINT8  if_bLength; */
        0x04,       /*  UINT8  if_bDescriptorType; Interface */
        0x00,       /*  UINT8  if_bInterfaceNumber; */
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        0x00,       /*  UINT8  if_bAlternateSetting; */
        0x01,       /*  UINT8  if_bNumEndpoints; */
        0x09,       /*  UINT8  if_bInterfaceClass; HUB_CLASSCODE */
        0x00,       /*  UINT8  if_bInterfaceSubClass; */
        0x00,       /*  UINT8  if_bInterfaceProtocol; */
        0x00,       /*  UINT8  if_iInterface; */

        /* endpoint */
        0x07,       /*  UINT8  ep_bLength; */
        0x05,       /*  UINT8  ep_bDescriptorType; Endpoint */
        0x81,       /*  UINT8  ep_bEndpointAddress; IN Endpoint 1 */
        0x03,       /*  UINT8  ep_bmAttributes; Interrupt */
        0x02,       /*  UINT16 ep_wMaxPacketSize; ((MAX_ROOT_PORTS + 1) / 8 */
        0x00,
        0xff        /*  UINT8  ep_bInterval; 255 ms */
};

/* Hub class-specific descriptor is constructed dynamically */


/*-------------------------------------------------------------------------*/

/* prepare Interrupt pipe data; HUB INTERRUPT ENDPOINT */

static INT  rh_send_irq(OHCI_T *ohci, VOID *rh_data, INT rh_len)
{
    INT     num_ports;
    INT     i;
    INT     ret;
    INT     len;
    UINT8   data[8];

    num_ports = readl(&ohci->regs->roothub.a) & RH_A_NDP;
    if (num_ports > MAX_ROOT_PORTS)
    {
        USB_printf("Error! - bogus NDP=%d", num_ports);
        return 0;
    }

    *(UINT8 *)data = (readl (&ohci->regs->roothub.status) &
                            (RH_HS_LPSC | RH_HS_OCIC)) ? 1 :  0;
    ret = *(UINT8 *)data;

    for ( i = 0; i < num_ports; i++)
    {
        *(UINT8 *)(data + (i + 1) / 8) |=
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                ((readl (&ohci->regs->roothub.portstatus[i]) &
                (RH_PS_CSC | RH_PS_PESC | RH_PS_PSSC | RH_PS_OCIC | RH_PS_PRSC))
                 ? 1 :  0) << ((i + 1) % 8);
        ret += *(UINT8 *)(data + (i + 1) / 8);
    }
    len = i/8 + 1;

    if (ret > 0)
    {
        memcpy(rh_data, data, min(len, min (rh_len, sizeof(data))));
        return len;
    }
    return 0;
}

/*-------------------------------------------------------------------------*/

static NU_TIMER  _RootHubTimer;

static URB_T  *_RootHubIntUrb;

/* Virtual Root Hub INTs are polled by this timer every "interval" ms */

VOID  rh_int_timer_do(UINT32 ptr)
{
    INT     len;

    URB_T   *urb = _RootHubIntUrb;
    OHCI_T  *ohci = urb->dev->bus->hcpriv;

    if (ohci->disabled)
        return;

    if (ohci->rh.send)
    {
        len = rh_send_irq (ohci, urb->transfer_buffer,
                           urb->transfer_buffer_length);
        if (len > 0)
        {
            urb->actual_length = len;
            if (urb->complete)
                urb->complete(urb);  /* call the hub_irq() in hub.c */
        }
    }
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
/*-------------------------------------------------------------------------*/

#define RH_OK(x)                len = (x); break
#define WR_RH_STAT(x)           writel((x), &ohci->regs->roothub.status)
#define WR_RH_PORTSTAT(x)       writel((x),
                                     &ohci->regs->roothub.portstatus[wIndex-1])
#define RD_RH_STAT              readl(&ohci->regs->roothub.status)
#define RD_RH_PORTSTAT          readl(&ohci->regs->roothub.portstatus[wIndex-1])

/* request to virtual root hub */

static INT  rh_submit_urb(URB_T *urb)
{
    USB_DEV_T *usb_dev = urb->dev;
    OHCI_T *ohci = usb_dev->bus->hcpriv;
    UINT32  pipe = urb->pipe;
    DEV_REQ_T  *cmd = (DEV_REQ_T *) urb->setup_packet;
    VOID    *data = urb->transfer_buffer;
    INT     leni = urb->transfer_buffer_length;
    INT     len = 0;
    INT     status = TD_CC_NOERROR;

    UINT32  datab[4];
    UINT8   *data_buf = (UINT8 *) datab;

    UINT16  bmRType_bReq;
    UINT16  wValue;
    UINT16  wIndex;
    UINT16  wLength;

    if (usb_pipeint(pipe))  /* 如果這個 URB 是屬於 interrupt pipe */
    {
        ohci->rh.urb =  urb;
        ohci->rh.send = 1;
        ohci->rh.interval = urb->interval;

        len = urb->interval / (NU_PLUS_Tick_Rate / 1000);   /* ticks count */
        NU_Create_Timer(&_RootHubTimer, "RHtimer", rh_int_timer_do, 1,
                        len * 2, len, NU_ENABLE_TIMER);
        _RootHubIntUrb = urb;
        urb->status = cc_to_error[TD_CC_NOERROR];
        return 0;
    }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    bmRType_bReq  = cmd->requesttype | (cmd->request << 8);
    wValue        = USB_SWAP16(cmd->value);
    wIndex        = USB_SWAP16(cmd->index);
    wLength       = USB_SWAP16(cmd->length);

    switch (bmRType_bReq)
    {
        /* Request Destination :
           without flags :  Device,
           RH_INTERFACE :  interface,
           RH_ENDPOINT :  endpoint,
           RH_CLASS means HUB here,
           RH_OTHER | RH_CLASS  almost ever means HUB_PORT here
        */

        case RH_GET_STATUS :
            *(UINT16 *) data_buf = USB_SWAP16(1);
            RH_OK(2);

        case RH_GET_STATUS | RH_INTERFACE :
            *(UINT16 *) data_buf = 0;
            RH_OK(2);

        case RH_GET_STATUS | RH_ENDPOINT :
            *(UINT16 *) data_buf = 0;
            RH_OK(2);

        case RH_GET_STATUS | RH_CLASS :
            *(UINT32 *) data_buf = USB_SWAP32(RD_RH_STAT &
                                    ~(RH_HS_CRWE | RH_HS_DRWE));
            RH_OK(4);

        case RH_GET_STATUS | RH_OTHER | RH_CLASS :
            *(UINT32 *) data_buf = USB_SWAP32(RD_RH_PORTSTAT);
            RH_OK(4);

        case RH_CLEAR_FEATURE | RH_ENDPOINT :
            switch (wValue)
            {
                case (RH_ENDPOINT_STALL) :
                    RH_OK(0);
            }
            break;

        case RH_CLEAR_FEATURE | RH_CLASS :
            switch (wValue)
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
            {
                case RH_C_HUB_LOCAL_POWER :
                    RH_OK(0);
                case (RH_C_HUB_OVER_CURRENT) :
                    WR_RH_STAT(RH_HS_OCIC); RH_OK(0);
            }
            break;

        case RH_CLEAR_FEATURE | RH_OTHER | RH_CLASS :
            switch (wValue)
            {
                case (RH_PORT_ENABLE) :
                    WR_RH_PORTSTAT (RH_PS_CCS );
                    RH_OK(0);
                case (RH_PORT_SUSPEND) :
                    WR_RH_PORTSTAT (RH_PS_POCI);
                    RH_OK(0);
                case (RH_PORT_POWER) :
                    WR_RH_PORTSTAT (RH_PS_LSDA);
                    RH_OK(0);
                case (RH_C_PORT_CONNECTION) :
                    WR_RH_PORTSTAT (RH_PS_CSC );
                    RH_OK(0);
                case (RH_C_PORT_ENABLE) :
                    WR_RH_PORTSTAT (RH_PS_PESC);
                    RH_OK(0);
                case (RH_C_PORT_SUSPEND) :
                    WR_RH_PORTSTAT (RH_PS_PSSC);
                    RH_OK(0);
                case (RH_C_PORT_OVER_CURRENT) :
                    WR_RH_PORTSTAT (RH_PS_OCIC);
                    RH_OK(0);
                case (RH_C_PORT_RESET) :
                    WR_RH_PORTSTAT (RH_PS_PRSC);
                    RH_OK(0);
            }
            break;

        case RH_SET_FEATURE | RH_OTHER | RH_CLASS :
            switch (wValue)
            {
                case (RH_PORT_SUSPEND) :
                    WR_RH_PORTSTAT (RH_PS_PSS );
                    RH_OK(0);

                case (RH_PORT_RESET) :  /* BUG IN HUP CODE *********/
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
                if (RD_RH_PORTSTAT & RH_PS_CCS)
                    WR_RH_PORTSTAT (RH_PS_PRS);
                RH_OK(0);

            case (RH_PORT_POWER) :
                WR_RH_PORTSTAT (RH_PS_PPS );
                RH_OK(0);

            case (RH_PORT_ENABLE) :  /* BUG IN HUP CODE *********/
                if (RD_RH_PORTSTAT & RH_PS_CCS)
                    WR_RH_PORTSTAT (RH_PS_PES );
                RH_OK(0);
        }
        break;

    case RH_SET_ADDRESS :
        ohci->rh.devnum = wValue;
        RH_OK(0);

    case RH_GET_DESCRIPTOR :
        switch ((wValue & 0xff00) >> 8)
        {
            case (0x01) :  /* device descriptor */
                len = min (leni, min (sizeof (root_hub_dev_des), wLength));
                data_buf = root_hub_dev_des;
                RH_OK(len);

            case (0x02) :  /* configuration descriptor */
                len = min (leni, min (sizeof(root_hub_config_des),
                                      wLength));
                data_buf = root_hub_config_des;
                RH_OK(len);

            case (0x03) :  /* string descriptors */
                len = USB_RootHubString(wValue & 0xff,
                   (int)(long) ohci->regs, "OHCI", data, wLength);
                if (len > 0)
                {
                    data_buf = data;
                    RH_OK(min (leni, len));
                }
            default :
                status = TD_CC_STALL;
        }
        break;
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
        case RH_GET_DESCRIPTOR | RH_CLASS :
            {
                UINT32 temp = readl (&ohci->regs->roothub.a);

                data_buf [0] = 9;           // min length;
                data_buf [1] = 0x29;
                data_buf [2] = temp & RH_A_NDP;
                data_buf [3] = 0;
                if (temp & RH_A_PSM)        /* per-port power switching? */
                    data_buf [3] |= 0x1;
                if (temp & RH_A_NOCP)       /* no overcurrent reporting? */
                    data_buf [3] |= 0x10;
                else if (temp & RH_A_OCPM)
                    data_buf [3] |= 0x8;

                datab [1] = 0;
                data_buf [5] = (temp & RH_A_POTPGT) >> 24;
                temp = readl (&ohci->regs->roothub.b);
                data_buf [7] = temp & RH_B_DR;
                if (data_buf [2] < 7)
                {
                    data_buf [8] = 0xff;
                }
                else
                {
                    data_buf [0] += 2;
                    data_buf [8] = (temp & RH_B_DR) >> 8;
                    data_buf [10] = data_buf [9] = 0xff;
                }
                len = min (leni, min (data_buf [0], wLength));
                RH_OK(len);
            }

        case RH_GET_CONFIGURATION :
            *(UINT8 *) data_buf = 0x01;
            RH_OK(1);

        case RH_SET_CONFIGURATION :
            WR_RH_STAT (0x10000);
            RH_OK(0);

        default :
            USB_printf("Error! - unsupported root hub command\n");
            status = TD_CC_STALL;
    }
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton

```
    len = min(len, leni);
    if (data != data_buf)
        memcpy(data, data_buf, len);
    urb->actual_length = len;
    urb->status = cc_to_error[status];

    urb->hcpriv = NULL;
    USB_DecreaseDeviceUser(usb_dev);
    urb->dev = NULL;
    if (urb->complete)
        urb->complete (urb);

    return 0;
}
```

The above information is the exclusive intellectual property of Nuvoton Tech. and shall not be disclosed, distributed or reproduced without permission from Nuvoton