VIA Networking Technologies, Inc.

VT6120/VT6122

Gigabit Ethernet Controller
Programming Guide

Revision 1.60
Jul 8, 2004

VIA Networking Technologies, INC.

Copyright Notice:

 is a registered trademark of VIA Networking Technologies, Incorporated.

All trademarks are the properties of their respective owners.

Disclaimer Notice:

No license is granted, implied or otherwise, under any patent or patent rights of VIA Networking Technologies Inc. VIA Networking Technologies Inc. makes no warranties, implied or otherwise, in regard to this document and to the products described in this document. The information provided by this document is believed to be accurate and reliable as of the publication date of this document. However, VIA Networking Technologies Inc. assumes no responsibility for any errors in this document. Furthermore, VIA Networking Technologies Inc. assumes no responsibility for the use or misuse of the information in this document and for any patent infringements that may arise from the use of this document. The information and product specifications within this document are subject to change at any time, without notice and without obligation to notify any person of such change.

Offices:

USA Office:
940 Mission Court
Fremont, CA 94539
USA
Tel: (510) 683-3300
Fax: (510) 683-3301 -or- (510) 687-4654
Web: http://www.vntek.com

Taipei Office:
8th Floor, No. 533
Chung-Cheng Road, Hsin-Tien
Taipei, Taiwan ROC
Tel: (886-2) 2218-2078
Fax: (886-2) 2219-8461
Web: http://www.vntek.com.tw

REVISION HISTORY

Document Release	Date	Revision	Initials
1.00	08/26/03	Initial release of this document.	YJ Chen
1.10	09/18/03	Add appended sample code files for MII and EEPROM access.	YJ Chen
1.20	11/14/03	Add EEPROM shadow read/write sample code.	YJ Chen
1.30	04/12/04	Modify product name string.	YJ Chen
1.40	05/06/04	Add TXSTL, RXSTL, and LSTE interrupt solution.	YJ Chen
1.50	06/10/04	Add appendix A.1, A.2, A.3, A.4	YJ Chen
1.60	07/08/04	Add Multicast Address receive (MAR, MCAM) ,Pattern Match WOL and Magic Packet WOL registers manipulation description.	YJ Chen

TABLE OF CONTENTS

1 PACKET TRANSMISSION

1.1 Architecture

The transmit descriptor (TD) command blocks used by VT6120/VT6122 is a round-robin ring structure (Figure 1). This ring structure is a physically continuous memory block on host. The maximum of TD's number is 4096(2^{12}) if host memory is enough.

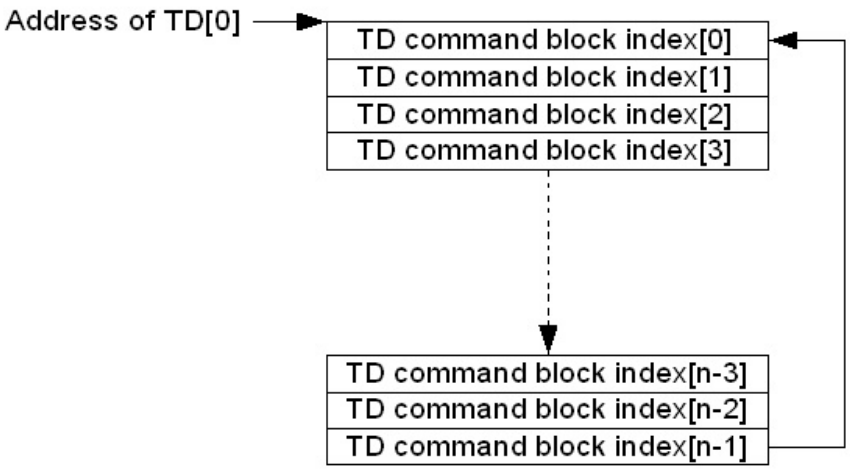


Figure 1. Transmit Descriptor (TD) Ring

1.2 TD Command Block

As Figure 2 shows, each TD command block defines a transmit packet with at most 7 physical segments. It must be allocated in 16-DoubleWord (64-Bytes) base address.

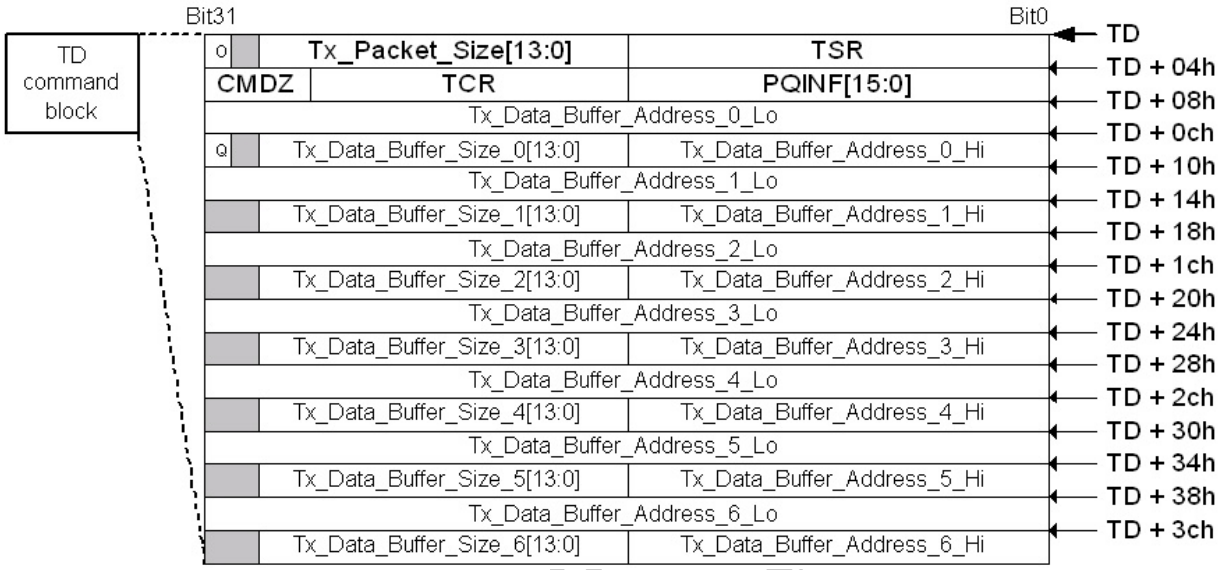


Figure 2. TD Command Block Format

The first two DOUBLE_WORDs of the TD command block is the header. The header contains some information like transmit status, packet size, owner bit, etc. Table 1 and Table 2 define each bit in the header.

Table 1. Bit Definitions of First DOUBLE_WORD of TD Command Block Header

Bit	Symbol	R/W	Description
31	OWN	R/W	Driver set this bit before sending this packet. After Tx DMA of this command block is completed, chip clear this bit.
30	-	-	
29-16	TxPktSize[13:0]	R/W	Total packet length. After Tx DMA of this command block is completed, chip clear this field.
15	TERR	R	Tx error status. TERR = ABT | OWT | OWC | SHDN.
14	FDX	R	"1" means packet is served by Full Duplex Mode. "0" means by Half Duplex Mode.
13	GMII	R	"1" means packet is served by GMII mode. "0" means by MII Mode.
12	LMKFL	R	"1" means packet is served during link down.
11	-	-	
10	SHDN	R	"1" means Tx shutdown case, no guarantee for Tx OK.
9	CRS	R	"1" means Carrier Sense is lost during packet transmission.
8	CDH	R	"1" means Collision Heart Beat detection failure in Half Duplex Mode.
7	ABT	R	"1" means transmission Abort due to excessive collision.
6	OWT	R	"1" means jumbo frame transmission abort.
5	OWC	R	"1" means Out of Window collision during transmission.
4	COLS	R	"1" means collision seen in current Tx OK status.
3-0	NCR[3:0]	R	Collision counts in current Tx OK status.

Table 2. Bit Definitions of Second DOUBLE_WORD of TD Command Block Header

Bit	Symbol	R/W	Description
31-28	CMDZ[3:0]	W	Determine how many segments inside the command block (segment number = CMDZ – 1).

27-26	-	-	
25-24	{TCPLS_SOF, TCPLS_EOF}	W	{1,1}: Normal packet. {1,0}: Start TD of a TCP Large Send packet. {0,0}: Intermediate TD of a TCP Large Send packet. {0,1}: End TD of a TCP Large Send packet.
23	TIC	W	"1" means issuing interrupt while this packet is sent.
22	PIC	W	"1" means Priority Interrupt request.
21	VETAG	W	Set "1" to enable VLAN Tag.
20	IPCK	W	Set "1" to enable IP Checksum calculation.
19	UDPCK	W	Set "1" to enable UDP Checksum calculation.
18	TCPCK	W	Set "1" to enable TCP Checksum calculation.
17	JMBO	W	Set "1" to indicate this is a jumbo packet.
16	CRC	W	Set "1" to disable CRC generation in this packet.
15-13	Priority[2:0]	W	802.1p priority bits.
12	-	-	
11-0	VLANID[11:0]	W	802.1Q VLAN ID.

There is also an important "QUE" control bit in the field of Tx_Data_Buffer_Size_0. The function of this bit is to indicate TD list queuing fetch. Use this bit correctly can improve Tx performance. Please see Figure 5, 6 for detail.

1.3 Transmission Process

To transmit a packet, driver has to setup a TD command block and other related registers of the chip. After that, the DMA engine will move packet data in the buffer indicated by the TD command block into transmit FIFO. The transmit FIFO of VT6120/VT6122 is a 16K Bytes memory, that keep the data from host memory then push it to the media (cable).

There are two ways to transmit a packet: **Direct Transmit** and **Indirect Transmit**. When driver transmits a packet in direct way, it needs to fill the Tx_Data_Buffer_Address (0~7)(Lo/Hi) fields of the TD command block with the physical memory address of the packet. But this packet must have 7 physical memory segment at most, this is a hardware limitation. DMA engine of the chip will move packet data from host memory into Tx FIFO and transmit data to the media. Please see Figure 3 for detail:

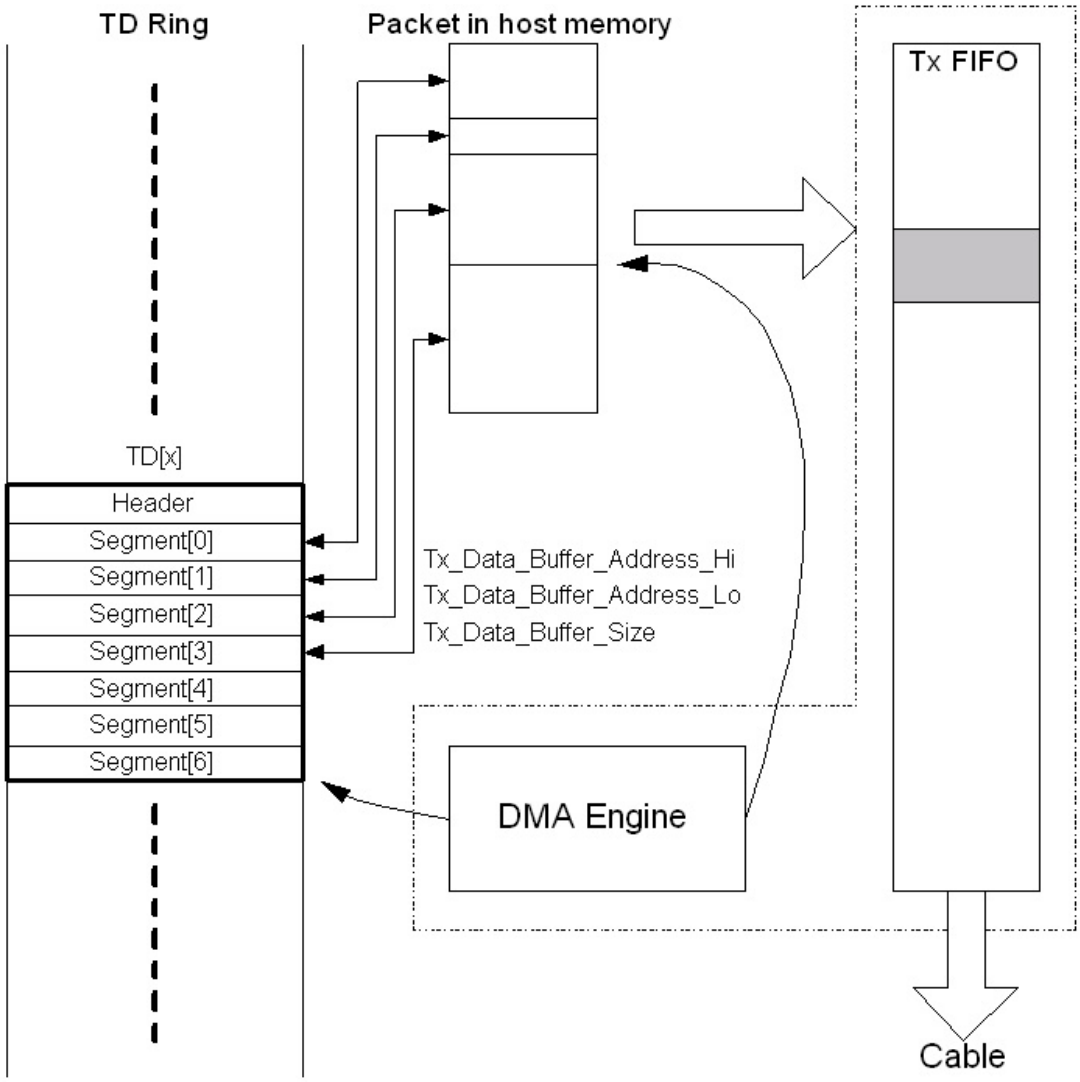


Figure 3. Direct Transmit

As for the other way of transmission, Indirect Transmit, driver needs to allocate extra data buffer first. Driver must fill the Tx_Data_Buffer_Address_0(Hi/Lo) field of the TD command block with the physical memory address of the allocated buffer and copy the packet to the buffer. For the example shown in Figure 4, the physical fragment of the packet is greater than 7, so the driver can copy the packet to extra allocated buffer. DMA engine will move data in the buffer into Tx FIFO and transmit it.

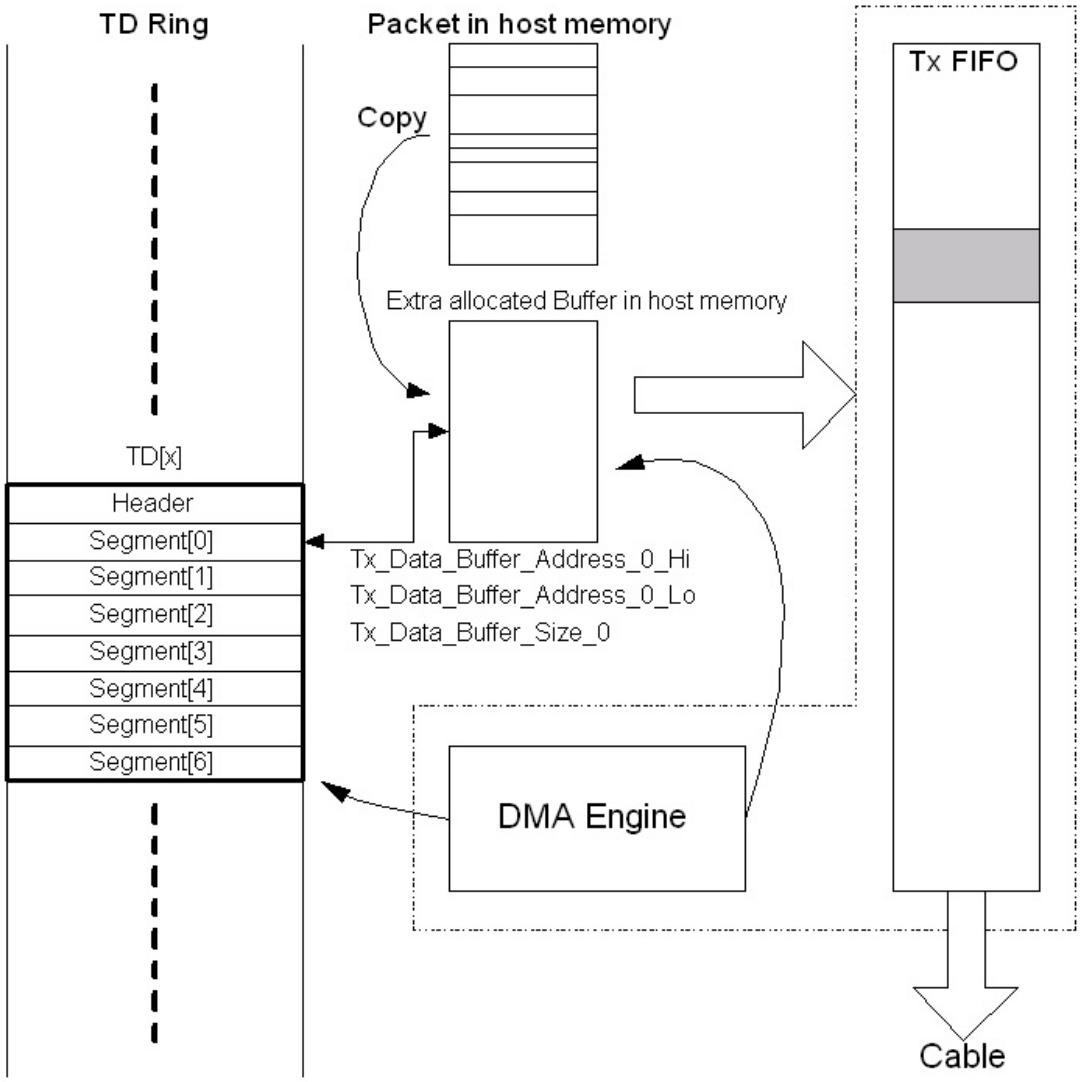


Figure 4. Indirect Transmit

If protocol layer pass packets one by one, driver should prepare one TD then issue send command to send it (Figure 5).

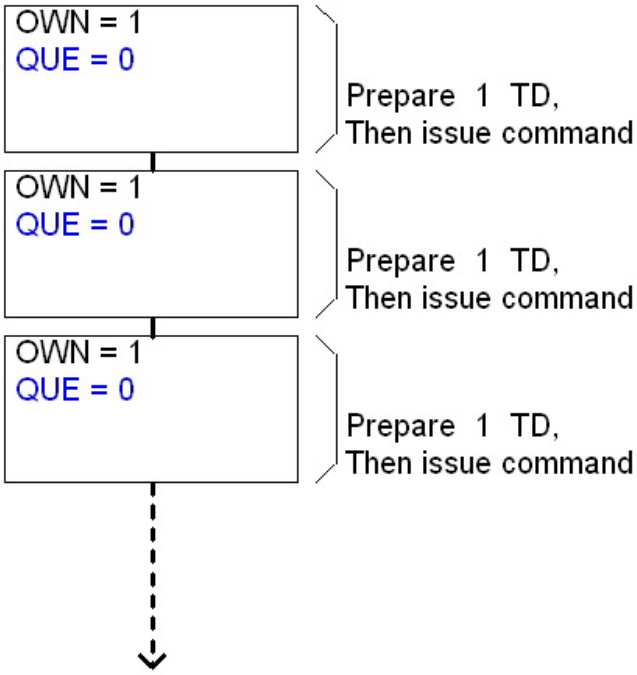


Figure 5. Driver sends a packet by one command

If protocol layer pass a lot of packets one time, driver can prepare all the TDs for the packets then issue one command to send them all (Figure 6), this method need to set QUE bit in each TD carefully, if the QUE is 1 in the list end TD, Tx DMA machine will meet error condition. Basically, if the QUE of current TD is 1, the chip will fetch next TD automatically, but if next TD's OWN = 0, abnormal case will occur. So, QUE bit must be cleared after the TD is sent completely.

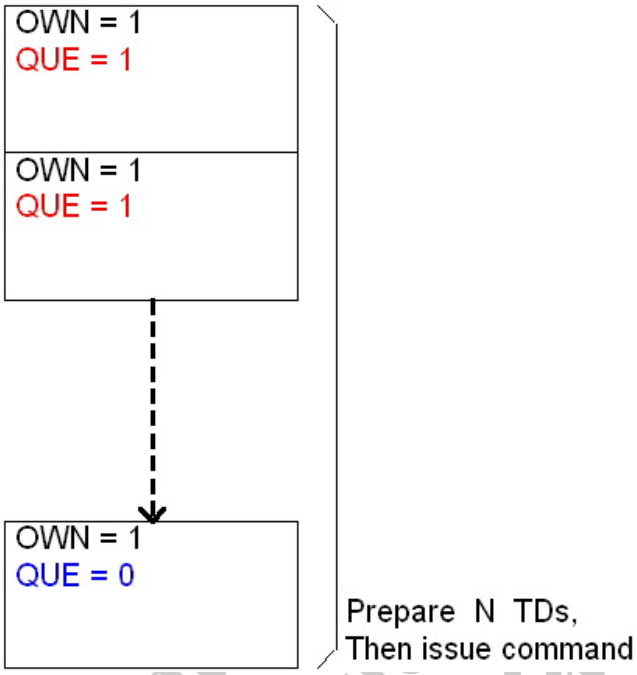


Figure 6. Driver sends many packets by one command

1.4 Related Registers

Table 3 lists related registers for transmission. Detailed layout of each register is in Table 4.

Table 3. Transmit Process Related Registers

	Registers			
Offset	+3h	+2h	+1h	+0h
18h	TDRDBase.Hi[63:32]			
1Ch	Reserved		TDRDBufAddr.Hi[63:48]	
30h			TDCSR.s	
34h			TDCSR.c	
40h	TDBase0.Lo[31:6]			
44h	TDBase1.Lo[31:6]			
48h	TDBase2.Lo[31:6]			
4Ch	TDBase3.Lo[31:6]			
50h	TDCSIZE[11:0]			
54h	TDIdx1[11:0]		TDIdx0[11:0]	
58h	TDIdx3[11:0]		TDIdx2[11:0]	

Table 4. Format of Tx Related Registers

Offset	Register	R/W	Description
18h-1Bh	TDRDBase.Hi[63:32]	R/W	TD/RD high address, R/W if 64-bit addressing, else read as 32'h0 always. This value is the high address of TD[0] in Figure 1. 4 TD queues use the same high address. See Figure 7 for detail.
1Ch-1Dh	TDRDBufAddr.Hi[63:48]	R/W	TD/RD linked data buffer high address, R/W if 64-bit addressing,

			else read as 16'h0 always.
30h-31h	TDCSR.s	R/W	Tx Descriptor Control Status Register Set for 4 TD queue. See Table 5, 6 for detail.
34h-35h	TDCSR.c	R/W	Tx Descriptor Control Status Register Clear for 4 TD queue. See Table 5, 6 for detail.
40h-43h	TDBase0.Lo[31:6]	R/W	TD Base Address Low Register, bits[5:0] R/W as 6'h0 always (64-byte alignment). This value is the low address of TD[0] of each TD queue in Figure 1. See Figure 7 for detail.
44h-47h	TDBase1.Lo[31:6]	R/W	
48h-4Bh	TDBase2.Lo[31:6]	R/W	
4Ch-4Fh	TDBase3.Lo[31:6]	R/W	
52h-53h	TDCSIZE[11:0]	R/W	TD number in every TD ring. 4 TD queues have the same TD number. This value is (TD# – 1). See Figure 7 for detail.
54h-55h	TDIdx0[11:0]	R/W	Current TD Index maintained by hardware in the ring structure. Driver can program it only when respect RUN bit of TDCSR is 0. See Figure 7 for detail.
56h-57h	TDIdx1[11:0]	R/W	
58h-59h	TDIdx2[11:0]	R/W	
5Ah-5Bh	TDIdx3[11:0]	R/W	

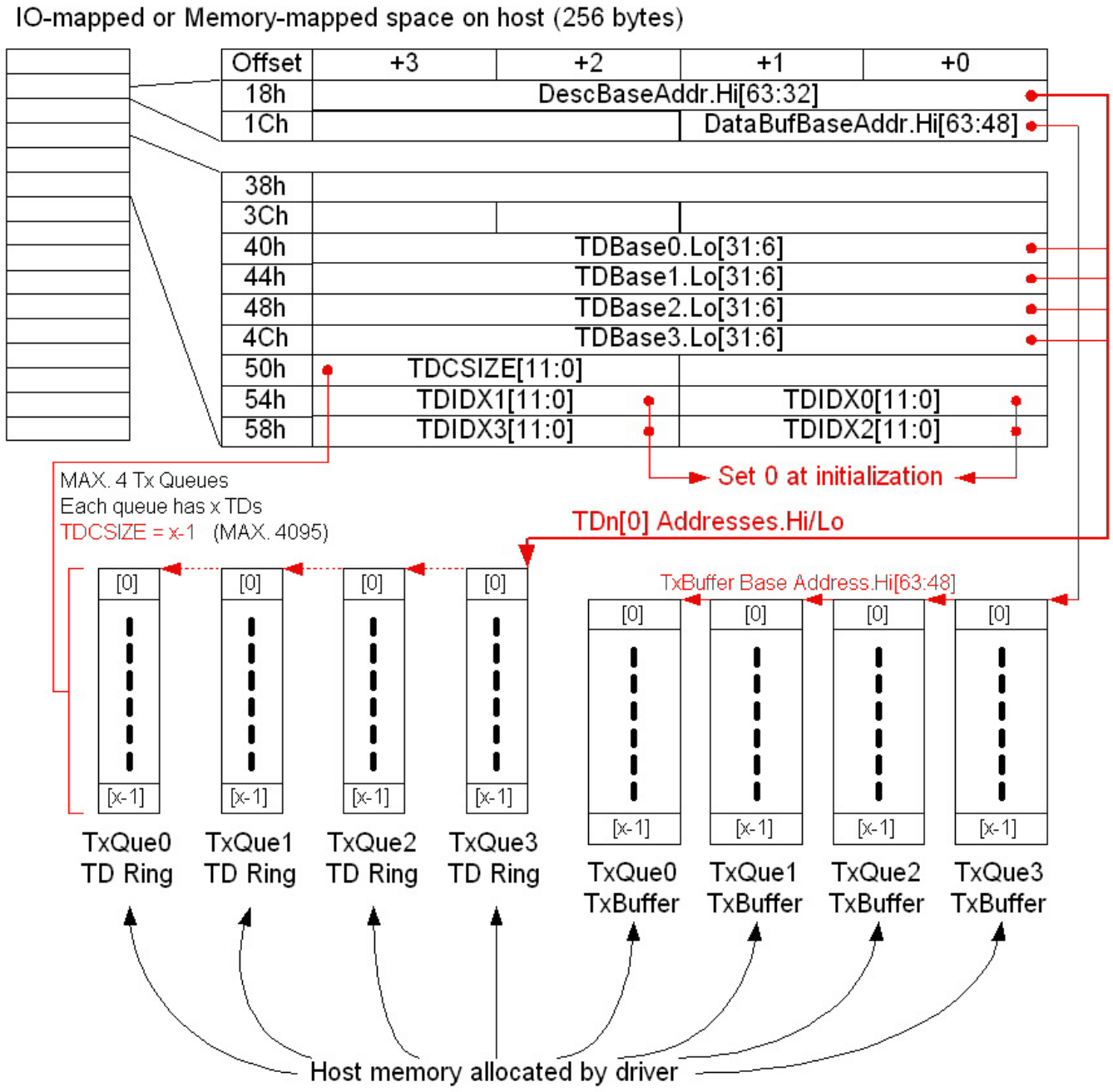


Figure 7. VT6120/VT6122 Memory Allocation for Packet Transmission

Table 5. TDCSR Set(30h-31h) / Clear(34h-35h) Format

Offset	31h / 35h								30h / 34h							
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QUE	3				2				1				0			
Func	D	W	A	R	D	W	A	R	D	W	A	R	D	W	A	R

Table 6. Bit Definitions of TDCSR

Bit	Symbol	Set/Clear	Description
0,4,8,12	**R**UN	Sw/Sw	Enable Tx queue to operate.
1,5,9,13	**A**CT	Hw/Hw	Indicate the end of Tx queue has not been reached when processing descriptors.
2,6,10,14	**W**AKE	Sw/Hw	Wake up Tx queue to see if there is unprocessed descriptor.
3,7,11,15	**D**EAD	Hw/Sw	Indicate Tx queue encounters error conditions.

2 PACKET RECEPTION

2.1 Architecture

The receive portion of VT6120/VT6122 uses ring structure, too (Figure 8). This Receive Descriptor (RD) ring buffer is physically continuous on host memory. Each RD links with an Rx Buffer to contain data come from media (cable). It is not necessary to make all Rx Buffers physically continuous. The maximum number of RD is 252(2^8 - 4) if host memory is enough.

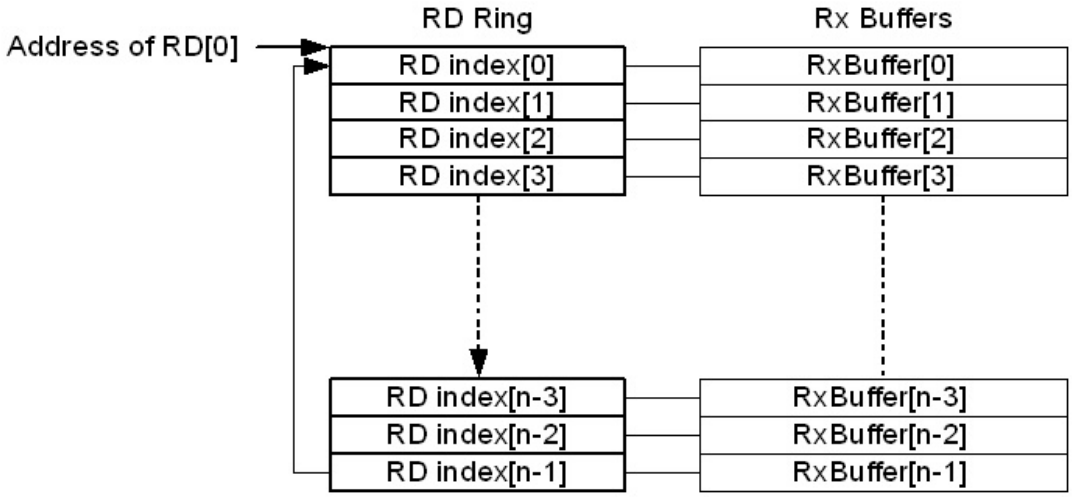


Figure 8. Receive Descriptor (RD) Ring

2.2 RD Command Block

Figure 9 shows the format of RD command block. Just the same as TD command blocks, RD command blocks must be allocated in 16-DOUBLE_WORD (64-Bytes) base address. The most important thing is, RD number must equal to 4X, X > 0. This is also hardware spec.

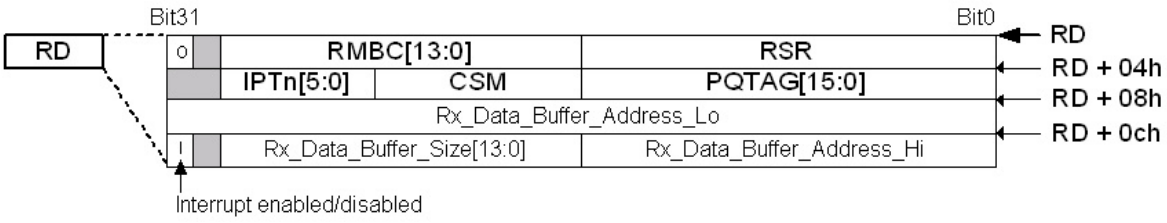


Figure 9. RD Command Block Format

The first two DOUBLE_WORDs of the RD command block is the header. The header contains some information like receive status, packet size, owner bit, 802.1p/Q tag, etc. Table 7 and Table 8 define each bit in the header.

Table 7. Bit Definitions of First DOUBLE_WORD of RD Command Block Header

Bit	Symbol	R/W	Description
31	OWN	R/W	"1" means this RD is owned by NIC, "0" means owned by host.
30	-	-	
29-16	RMBC[13:0]	R	Received packet length.
15	RXOK	R	"1" means Rx OK status. "0" means Rx error status.
14	PFT	R	"1" means perfect filtering address match, check the PQTAG field to see if interesting packet hit.
13	MAR	R	"1" means this received packet is multicast address.
12	BAR	R	"1" means this received packet is broadcast address.
11	PHY	R	"1" means this received packet is unicast address.
10	VTAG	R	"1" means this received packet with 802.1p/Q tag.
9-8	{STP,EDP}	R	{0,0}: Single packet in single RD. {1,0}: Start RD of chained packet. {1,1}: Intermediate RD of chained packet. {0,1}: End RD of chained packet.
7	DETAG	R	"1" means hardware de-tag.
6	SNTAG	R	"1" means 802.1p/Q Tag behind SNAP Tag.
5	RXER	R	"1" means PHY detected PCS symbol error.
4	RL	R	"1" means received SNAP packet length error.
3	CE	R	"1" means received packet Checksum Error.
2	FAE	R	"1" means received packet Frame Alignment Error.
1	CRC	R	"1" means received packet CRC error.
0	VIDM	R	"1" means VID filtering miss.

Table 8. Bit Definitions of Second DOUBLE_WORD of RD Command Block Header

Bit	Symbol	R/W	Description
31-30	-	-	
29-24	IPTn[5:0]	R	Index of perfect matched interesting filtering.
23	-	-	
22	IPOK	R	"1" means received packet IP checksum validation OK.
21	TUPOK	R	"1" means received packet TCP/UDP checksum validation OK.
20	FRAG	R	"1" means fragment IP datagram.
19	UDPZRO	R	"1" means UDP checksum is zero.
18	IPKT	R	"1" means this is IP packet.
17	TPKT	R	"1" means this is TCP packet.
16	UPKT	R	"1" means this is UDP packet.
15-0	PQTAG	R	VID/Priority tag.

2.3 Reception Process

In the initial stage, driver allocates RD ring and RD data buffers then links them together. When a packet comes from cable and passes the filter, it is pushed into the receive FIFO of the chip. According to the

RDINDEX register, the DMA engine then moves the packet in the Rx FIFO to the correct data buffer. Please see Figure 10 for detail.

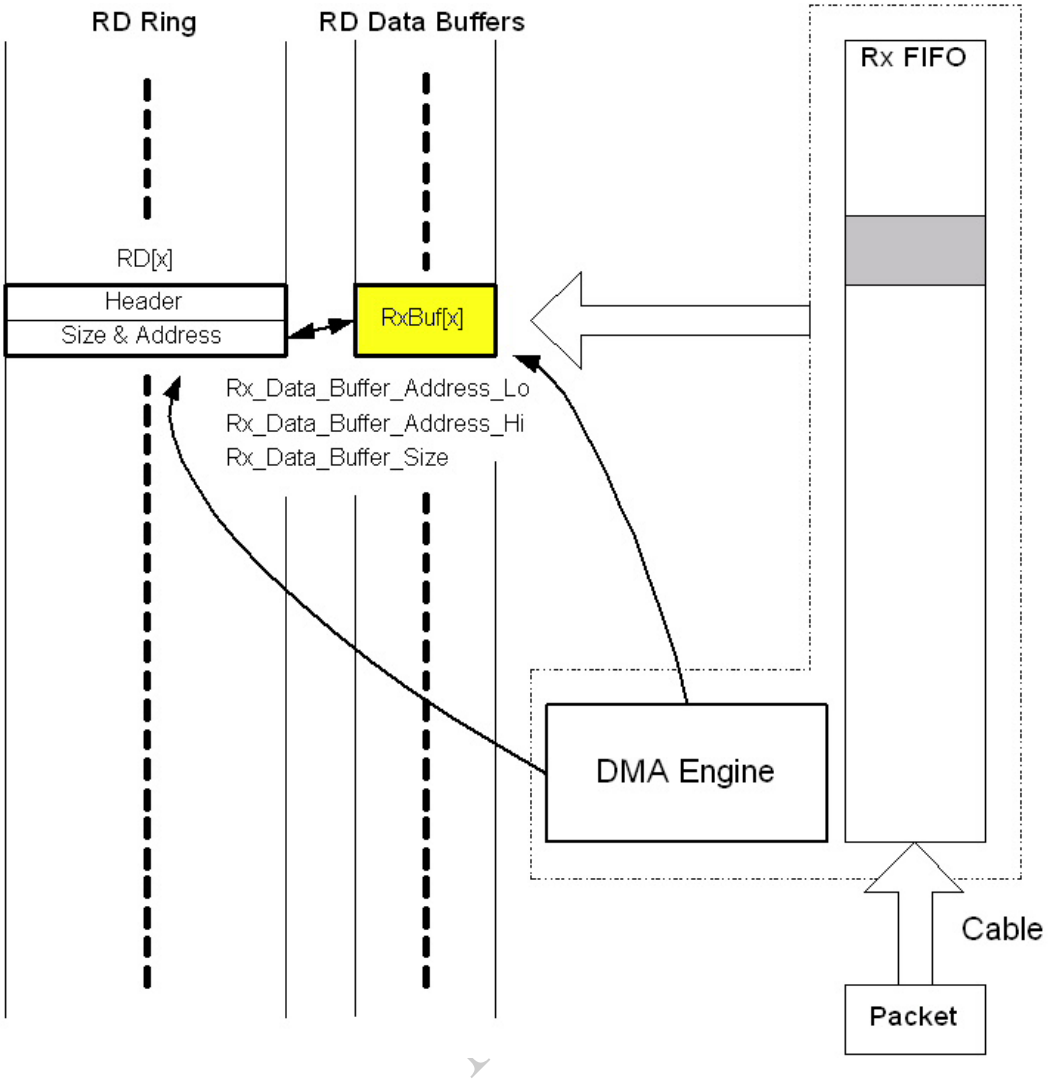


Figure 10. Receive Process

2.4 Related Registers

Table 9 lists related registers for reception. Detailed format of each register is in Table 10.

Table 9. Receive Process Related Registers

	Registers			
Offset	+3h	+2h	+1h	+0h
18h	TDRDBase.Hi[63:32]			
1Ch	Reserved		TDRDBufAddr.Hi[63:48]	
30h		RDCSR.s[3:0]		

34h		RDCSR.c[3:0]	
38h	RDBase.Lo[31:6]		
3Ch			RDINDEX[7:0]
50h			RDCSIZE[7:0]

Table 10. Format of Rx Related Registers

Offset	Register	R/W	Description
18h-1Bh	TDRDBase.Hi[63:32]	R/W	TD/RD high address, R/W if 64-bit addressing, else read as 32'h0 always. This value is the high address of RD[0] in Figure 6. See Figure 11 for detail.
1Ch-1Dh	TDRDBufAddr.Hi[63:48]	R/W	TD/RD linked data buffer high address, R/W if 64-bit addressing, else read as 16'h0 always. See Figure 11 for detail.
32h	RDCSR.s[3:0]	R/W	Rx Descriptor Control Status Register Set. See Table 11, 12 for detail.
36h	RDCSR.c[3:0]	R/W	Rx Descriptor Control Status Register Clear. See Table 11, 12 for detail.
38h-3Bh	RDBase.Lo[31:6]	R/W	RD Base Address Low register, bits[5:0] R/W as 6'h0 always (64-byte alignment). This value is the low address of RD[0] in Figure 8. See Figure 11 for detail.
3Ch-3Dh	RDINDEX[7:0]	R/W	Current RD Index maintained by hardware in ring structure. Driver can program it only when RUN bit of RDCSR is 0. See Figure 11 for detail.
50h-51h	RDCSIZE[7:0]	R/W	RD number in RD ring. This value is (RD# – 1). See Figure 11 for detail.

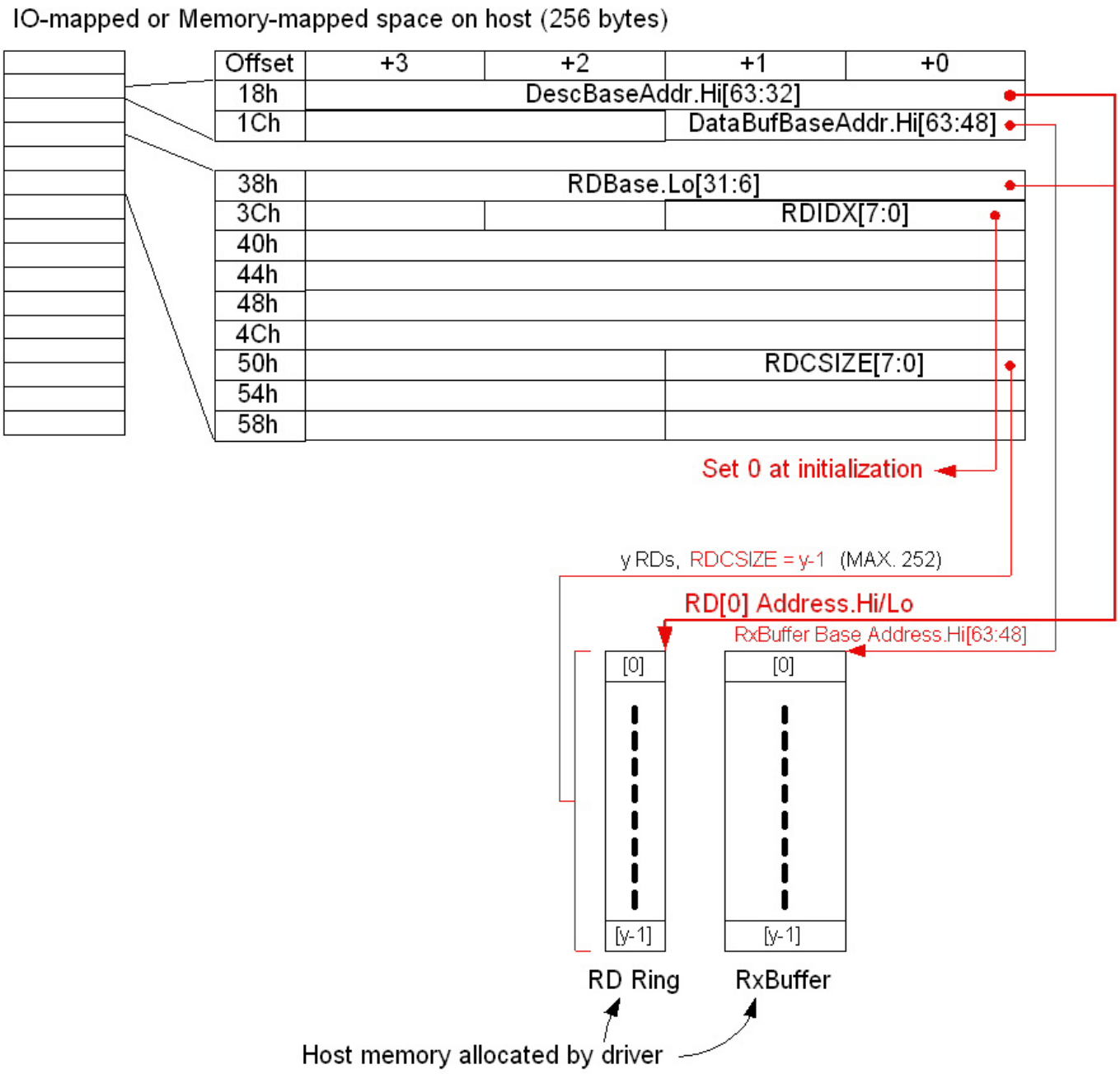


Figure 11. VT6120/VT6122 Memory Allocation for Packet Reception

Table 11. RDCSR Set(32h) / Clear(36h) Format

Offset	32h / 36h							
Bit	7	6	5	4	3	2	1	0
Sym	-	-	-	-	DEAD	WAKE	ACT	RUN

Table 12. Bit Definitions of RDCSR

Bit	Symbol	Set/Clear	Description
0	RUN	Sw/Sw	Enable Rx queue to operate.
1	ACT	Hw/Hw	Indicate the end of Rx queue has not been reached when processing descriptors.
2	WAKE	Sw/Hw	Wake up Rx queue to see if there is unprocessed descriptor.
3	DEAD	Hw/Sw	Indicate Rx queue encounters error conditions.

2.5 Multicast Address Receive

VT6120/VT6122 adopts two mechanisms to filter multicast address packets, one is hashing table, the other is CAM (Content Access Memory). CAM can do perfect multicast address filtering because it compare all the 6 bytes DA of the incoming packet, only perfect match of the 6 bytes let the packet be accepted. If driver uses hashing table, due to the hash value is calculated form the multicast address we want to filter with a specific algorithm, it is not guaranteed the received multicast packet is really what we want.

In general, driver uses CAM first. VT6120/VT6122 supports up to 64 CAMs, this is sufficient to most of the OSs. If driver needs to filter more than 64 multicast addresses, it can use hashing table mechanism after the 64th addresses. The related registers to access multicast address/hash value are listed in Table 13 and Table 14.

Table 13. Registers to access CAM/Multicast Hashing Table

	Registers			
Offset	+3h	+2h	+1h	+0h
10h	MAR3/CAM3	MAR2/CAM2	MAR1/CAM1	MAR0/CAM0
14h	MAR7/CAM7	MAR6/CAM6	MAR5/CAM5	MAR4/CAM4
68h			CAMCR	CAMADR

Table 14. Layout of CAM address and control registers

Register	Bit	Symbol	R/W	Description
CAMCR	15-14	PS1-0	R/W	PS1=0, PS0=0: 0x10~0x17 Function = MAR, 0x10~0x17 Range = MAR7~0 PS1=0, PS0=1: 0x10~0x17 Function = CAM Mask, 0x10~0x17 Range = CAM7~0 PS1=1, PS0=0: 0x10~0x17 Function = CAM Data, 0x10~0x17 Range = Multicast CAM: CAM5~0, VLAN CAM: CAM1~0 PS1=1, PS0=1: 0x10~0x17 Function = MAR, 0x10~0x17 Range = MAR7~0
	13-12	--	--	
	11	CAMRD	R/WC	CAM Read command, self cleared. Work only while CAMEN=1.
	10	CAMWR	R/WC	CAM Write command, self cleared. Work only while CAMEN=1.
	9-8	--	--	
CAMADR	7	CAMEN	R/W	Enable CAM Read/Write command.
	6	VCAMSL	R/W	Hit VLAN CAM while VCAMSL=1, Multicast CAM while VCAMSL=0.
	5-0	CA5-0	R/W	CAM Address.

The flow of reading/writing Multicast CAM is below, please also refer to the sample code in **Section 7.6**.

(1) Select the page of CAM data.
(2) Enable Multicast CAM and set the address (0~63) of CAM to store the multicast address.
(3) Read/Write CAM data.

The flow of setting Multicast hash value is below, please also refer to the sample code in **Section 7.7**. Basically, the hash value is calculated from the 32-bit CRC of the multicast address. The sample code in **Section 7.8** shows the algorithm of calculating 32-bit CRC.

(1) Calculate the multicast address's 32-bit CRC.
(2) Get hash index(0~63) from the CRC value.
(3) Add the hash index into multicast address table.

3 INTERRUPT HANDLING

VT6120/VT6122 issues interrupts in many cases, basically there are 3 kinds of interrupts: Tx, Rx, and others. Interrupt related registers are listed in Table 15.

Table 15. Interrupt Related Registers

	Registers			
Offset	+3h	+2h	+1h	+0h
08h	CR3.s			
0Ch	CR3.c			
20h	RXESR	TXESR		
24h	ISR3	ISR2	ISR1	ISR0
28h	IMR3	IMR2	IMR1	IMR0

All interrupts' on/off are controlled by GintMsk1 (bit 1) of CR3. To enable chip to issue interrupts, CR3_GintMsk1 and specific Interrupt Mask Register (IMR) must set to 1. Some Interrupt Status Registers (ISR) can be cleared by writing 1, write 0 doesn't change ISR's value. Detailed layout of IMR/ISR is listed in Table 16.

Table 16. Layout of Interrupt Status/Mask Register (ISR/IMR)

Register	Bit	Symbol	R/W Type	Description
ISR3	31,30,29,28	ISR3,2,1,0	R only	Interrupt source indication.
	27-26	--	--	
	25	TXSTL	R only	Transmission DMA stall in TXESR.
	24	RXSTL	R only	Reception DMA stall in RXESR.
ISR2	23	--	--	
	22	UDP	R/W1 clear	User defined, soft driven interrupt for diagnosis.
	21	MIBF	R/W1 clear	MIB counter near full warning.
	20	SHDN	R/W1 clear	Software shutdown complete.
	19	PHY	R/W1 clear	If PHYINTEN (bit0 of CHIPGCR, 9fh) is 1, this bit shows PHY interrupt event occurred.
	18	PWE	R/W1 clear	Wake up power events reporting status for test purpose.
	17	TMR1	R/W1 clear	Programmable software Timer 1 expired indication.
	16	TMR0	R/W1 clear	Programmable software Timer 0 expired indication.
ISR1	15	SRC	R/W1 clear	Link status change indication.
	14	LSTPE	R/W1 clear	RD is using up warning.
	13	LSTE	R/W1 clear	RD is used up indication.
	12	OVF	R/W1 clear	Receive FIFO overflow indication, some receive packets might be lost.
	11	FLON	R/W1 clear	Receive flow control mechanism turn on notification.
	10	RACE	R/W1 clear	Receive FIFO packet list queue overflow indication.
	9-8	--	--	
ISR0	7,6,5,4	PTX3,2,1,0	R/W1 clear	Tx service complete indication in TxQueue #3,2,1,0.
	3	PTX	R only	Combination result of PTXn.
	2	PRX	R/W1 clear	Rx service complete indication.
	1	PPTX	R/W1 clear	High priority Tx interrupt service indication. PTXn will be set also.

	0	PPRX	R/W1 clear	High priority Rx interrupt service indication. PRX will be set also.

If ISR3_TXSTL is 1, it means something wrong with Tx process and will cause TDCSR.Dead set. This result is the OR operation of TXESR (22h). Table 17 is the layout of TXESR, clear this register can also clear ISR3_TXSTL.

Table 17. Layout of Transmit Error Status Register (TXESR, 22h)

Bit	Symbol	R/W	Description
23-20	--	--	
19	TFDBS	R/W1 clear	Tx FIFO DMA bus error.
18	TDWBS	R/W1 clear	TD write back host bus error.
17	TDRBS	R/W1 clear	TD fetch host bus error.
16	TDSTR	R/W1 clear	(1)TD link structure error. (2)Valid CMDZ with zero buffer length.

If ISR3_RXSTL is 1, it means something wrong with Rx process and will cause RDCSR.Dead set. This result is the OR operation of RXESR (23h). Table 18 shows the layout of RXESR, clear this register can also clear ISR3_RXSTL.

Table 18. Layout of Receive Error Status Register (RXESR, 23h)

Bit	Symbol	R/W	Description
23-20	--	--	
19	RFDBS	R/W1 clear	Rx FIFO DMA bus error.
18	RDWBS	R/W1 clear	RD write back host bus error.
17	RDRBS	R/W1 clear	RD fetch host bus error.
16	RDSTR	R/W1 clear	Valid RD with linked buffer size zero.

When driver runs into interrupt service routine (ISR) code, after recognizing the interrupt and clear it, ISR code should disable interrupts, handle different interrupt types, then enable interrupts before ISR is ended.

3.1 Tx Interrupt Handling

There should be two software index for each TD queue. One is used for sending packet, the other is used for serving used TD and checking Tx status. As Figure 12 shows, after a packet is sent, the TD's OWN bit will be set as 0. If Tx interrupt occurred, driver should check TDs from TD_Index_For_Interrupt

recorded last time, till the TD which with OWN = 1. Since OWN = 1 means this TD is not transmitted completely, driver should not keep on checking TD.

When ISR3_TXSTL occurs, it means Tx DMA stall, Tx state machine is shut down. Driver should complete pending packets, clear all TDs' RUN, reset TD ring(s), clear all TDs' DEAD. Then set all TDs' RUN again.

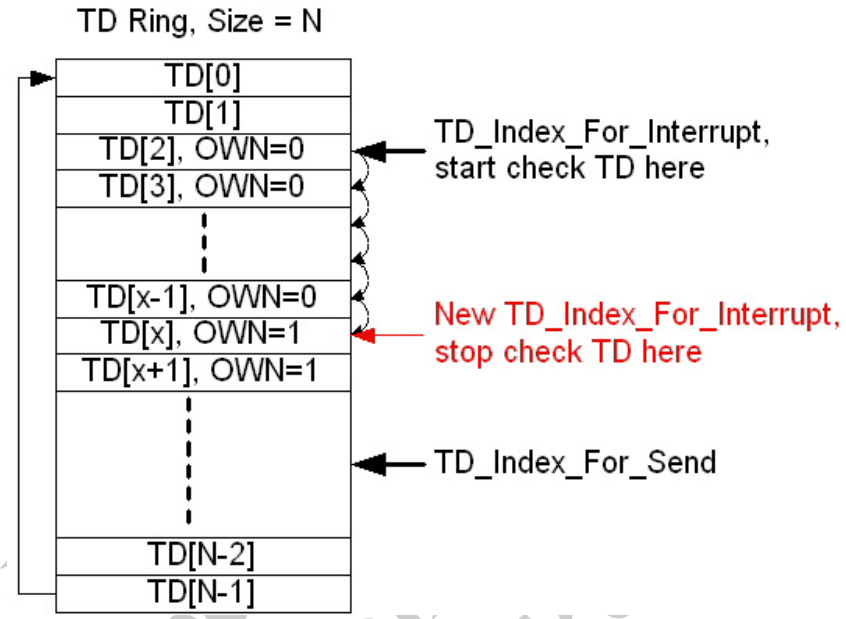


Figure 12. Tx Interrupt Handling

3.2 Rx Interrupt Handling

Basically, the reception procedure of driver is the Rx interrupt service routine of the driver. If a packet is written into a Rx buffer completely, the RD's OWN will be set as 0. If Rx interrupt occurred, driver should check RDs from RD_Index_For_Interrupt recorded last time, till the RD which with OWN = 1. In Rx case, OWN = 1 means this RD is not used yet or the reception procedure of this RD is not completed, driver should stop check RD at this moment. Besides, driver needs to inform upper protocol layer(s) that packet(s) is received, protocol layer(s) will get this packet(s). If protocol(s) gets the packet(s) and informs driver can reuse the RD, driver should set the RD's OWN as 1. Figure 13 shows the concept.

When ISR1_LSTE occurs, it means RDs are used up. In this situation driver should re-wake RD ring again to receive packets. When network traffic is very heavy, ISR1_LSTE may occur frequently. Driver **should not disable** this interrupt, or it cannot know when to re-wake RD ring.

When ISR3_RXSTL occurs, Rx state machine is shut down. Driver should clear RD's RUN, reset RD ring, clear RD's DEAD. Then set RD's RUN and WAKE again.

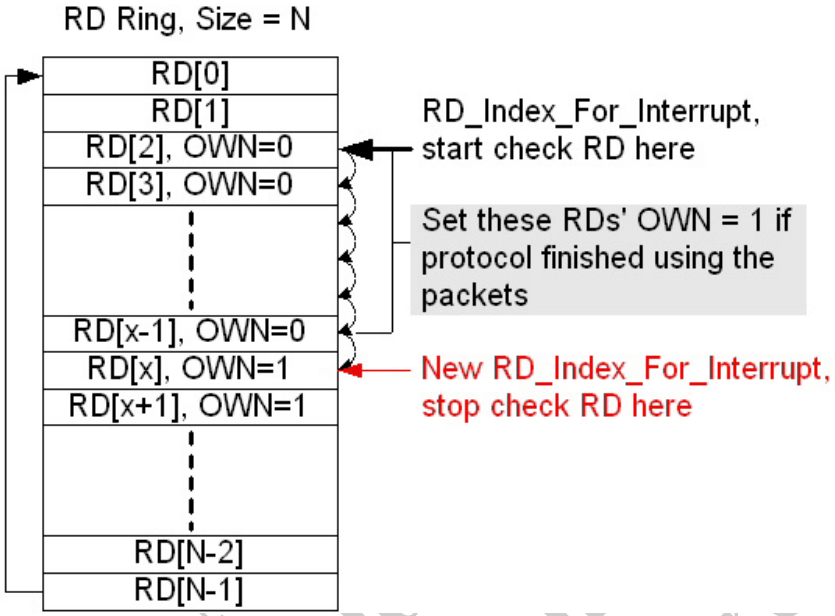


Figure 13. Rx Interrupt Handling

3.3 Other Interrupt Handling

In other interrupts, when Link-Status-Change interrupt (ISR1_SRC) occurs, driver should check media's link status and inform upper protocol layer that media is connected or disconnected.

If timer interrupts occurs (ISR2_TMR1, TSR2_TMR0), driver should run timer callback functions.

4 PHY ACCESS

4.1 Architecture

Driver should access PHY registers via GMII/MII interface just as Figure 14 shows. Besides, the MAC of VT6120/VT6122 can poll the status of PHY continuously through GMII/MII interface. But when link status is changed, auto-polling function will be disabled by MAC automatically. The link status could still be changed after auto-polling shutdown, so when interrupt service routine (ISR) wants to get the updated link status, driver needs to issue software polling MII status. Driver should enable MII auto-polling function again before ending ISR. Table 19 and Table 20 list the related registers to access PHY.

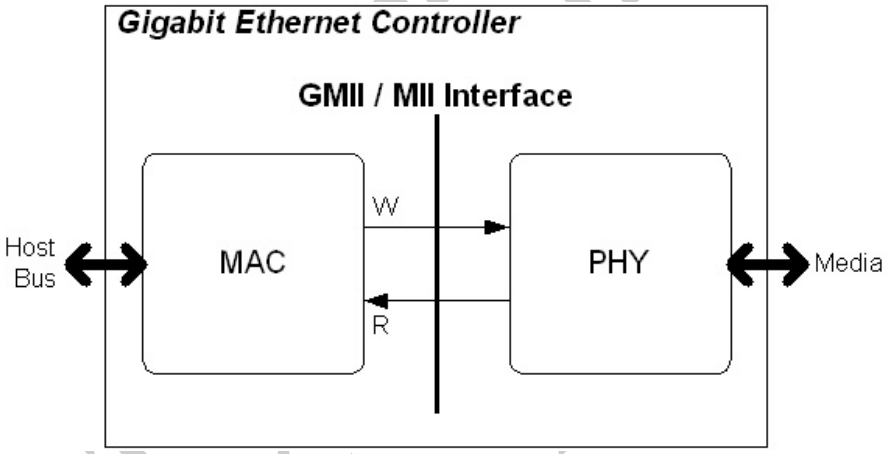


Figure 14. VT6120/VT6122 PHY access architecture

4.2 Related Registers

Table 19. PHY Access Related Registers

	Registers			
Offset	+3h	+2h	+1h	+0h
6Ch			MIISR	
70h	MII_DATA		MIIADR	MIICR

Table 20. Layout of registers for PHY Access

Register	Bit	Symbol	R/W	Description
MIISR	15	MIIDL	R Only	"1" means not at the software/timer polling cycle.
MIICR	7	MAUTO	R/W	"1" means enable MII port auto-polling. MIICR has no effect while MAUTO = 1.

	6	RCMD	R/W	MII port embedded read command. Cleared by hardware while read complete and PHY status is stored in 0x72.
	5	WCMD	R/W	MII port embedded write command. Cleared by hardware while programming complete.
MIIADR	15	SWMPL	R/W	Software initiated MII port polling command. Hardware will clear this bit when the process is completed.
	12-8	MAD[4:0]	R/W	PHY register address for embedded read/write process.
MII_DATA	31-16	[15:0]	R/W	PHY embedded read/write data port.

4.3 Embedded Read PHY Register Procedure

The flow of MII embedded read is below, please also refer to appended C and Assembly sample code files for x86 platform.

(1) Disable MII auto-polling function to enable MIICR register.
(2) Program MIIADR to select the offset to read.
(3) Turn on MIICR_RCMD only.
(4) After MIICR_RCMD is turned off by chip, the read process is completed.
(5) Read MII_DATA for the content of PHY register.
(6) Enable MII auto-polling function to disable MIICR register.

4.4 Embedded Write PHY Register Procedure

The flow of MII embedded write is below, please also refer to appended C and Assembly sample code files for x86 platform.

(1) Disable MII auto-polling function to enable MIICR register.
(2) Program MIIADR to select the offset to write.
(3) Program MII_DATA to set the data to write.
(4) Turn on MIICR_WCMD only.
(5) After MIICR_WCMD is turned off by chip, the write process is completed.
(6) Enable MII auto-polling function to disable MIICR register.

5 EEPROM ACCESS

5.1 Architecture

Figure 15 shows that the access of EEPROM should be achieved via EEPROM interface. There are two modes to EEPROM: embedded programming mode and direct programming mode. Here we only introduce embedded mode. Table 21 and Table 22 list related registers to read/write EEPROM.

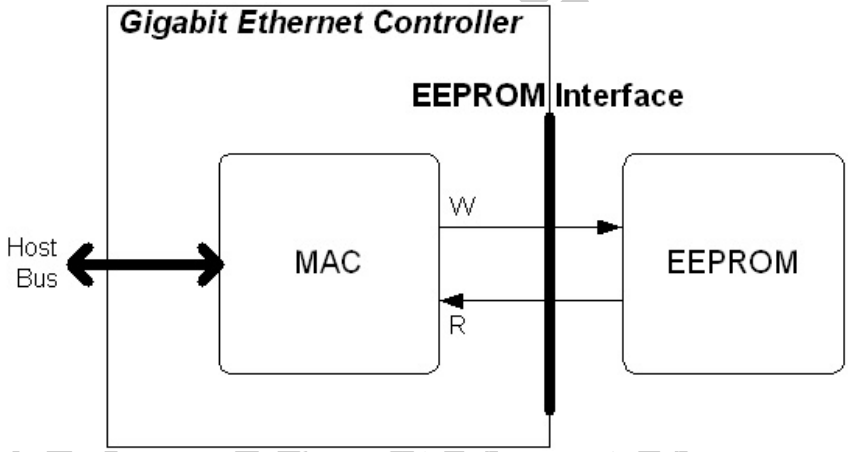


Figure 15. VT6120/VT6122 EEPROM access architecture

5.2 Related Registers

Table 21. EEPROM Access Related Registers

	Registers			
Offset	+3h	+2h	+1h	+0h
78h		CFG_C		
8Ch			EE_WR_DATA	
90h	EECSR			
94h	EMCMD	EADDR	EE_RD_DATA	

Table 22. Layout of Registers for EEPROM Access

Register	Bit	Symbol	R/W	Description
CFG_C	23	EELOAD	R/W	Enable EEPROM embedded and direct programming, always 0 after power on and loading.
EE_WR_DATA	15-0	[15:0]	R/W	EEPROM embedded write data port.
EECSR	30	EMBP	R/W	EEPROM embedded program mode enable. Programmable only when CFGC.EELOAD=1.

	29	RELOAD	R/W	Dynamic reload EEPROM, the Ethernet ID and related chip configuration will be updated. Cleared by hardware while process is completed.
EE_RD_DATA	15-0	[15:0]	R/W	EEPROM embedded read data port.
EADDR	23-16	[7:0]	R/W	EEPROM embedded operation address port.
EMCMD	25	EWR	R/W	Embedded write with programmable address, cleared by hardware while program done.
	24	ERD	R/W	Embedded read with programmable address, cleared by hardware while program done.

5.3 Embedded Read EEPROM Procedure

The flow of EEPROM embedded read is below, please also refer to appended C and Assembly sample code files for x86 platform. Note: the sample code is **NOT** suitable for **EEPROM-less** environment.

(1) Enable embedded programming mode.
(2) Program EADDR to select the offset to read.
(3) Turn on EMCMD_ERD only.
(4) After EMCMD_ERD is turned off by chip, the read process is completed.
(5) Disable embedded programming mode.
(6) Read EE_RD_DATA for EEPROM content.

5.4 Embedded Write EEPROM Procedure

The flow of EEPROM embedded write is below, please also refer to appended C and Assembly sample code files for x86 platform. Note: the sample code is **NOT** suitable for **EEPROM-less** environment.

(1) Enable embedded programming mode.
(2) Program EADDR to select the offset to write.
(3) Program EE_WR_DATA to set the data to write.
(4) Turn on EMCMD_EWR only.
(5) After EMCMD_EWR is turned off by chip, the write process is completed.
(6) Disable embedded programming mode.

5.5 Load EEPROM Contents to MAC

VT6120/VT6122 can reload the contents of EEPROM to specific MAC registers, for example, Ethernet Address. The flow of this function is below:

(1) Turn on EECSR_RELOAD only.

(2) After EECSR_RELOAD is turned off by chip, the reload process is completed.

6 WAKE-ON-LAN (WOL) FUNCTION

6.1 Pattern Match WOL

VT6120/VT6122 can support at most 8 WOL patterns. Patterns and the mask to filter the content of the pattern can be defined by protocols in some OS or just defined by user. In both cases, driver needs to write the "Byte Mask" and "16-bit CRC" in respect to the pattern into MAC's registers. The "16-bit CRC" is calculated from the pattern and the mask. Table 23 and Table 24 list related registers to enable Pattern Match WOL function.

6.1.1 Related Registers

Table 23. Pattern Match WOL Related Registers

	Registers			
Offset	+3h	+2h	+1h	+0h
68h			CAMCR	
A0h				WOLCR0.s
A4h				WOLCR0.c
A8h				WOLSR0.s
ACh				WOLSR0.c
B0h	PatternCRC1[15:0]		PatternCRC0[15:0]	
B4h	PatternCRC3[15:0]		PatternCRC2[15:0]	
B8h	PatternCRC5[15:0]		PatternCRC4[15:0]	
BCh	PatternCRC7[15:0]		PatternCRC6[15:0]	
C0h	Pattern 0/4, ByteMask[31:0]			
C4h	Pattern 0/4, ByteMask[63:32]			
C8h	Pattern 0/4, ByteMask[95:64]			
CCh	Pattern 0/4, ByteMask[127:96]			
D0h	Pattern 1/5, ByteMask[31:0]			
D4h	Pattern 1/5, ByteMask[63:32]			
D8h	Pattern 1/5, ByteMask[95:64]			
DCh	Pattern 1/5, ByteMask[127:96]			
E0h	Pattern 2/6, ByteMask[31:0]			
E4h	Pattern 2/6, ByteMask[63:32]			
E8h	Pattern 2/6, ByteMask[95:64]			
ECh	Pattern 2/6, ByteMask[127:96]			
F0h	Pattern 3/7, ByteMask[31:0]			
F4h	Pattern 3/7, ByteMask[63:32]			
F8h	Pattern 3/7, ByteMask[95:64]			
FCh	Pattern 3/7, ByteMask[127:96]			

Table 24. Layout of Registers for Pattern Match WOL

Register	Bit	Symbol	R/W	Description
CAMCR	6	PS0	R/W	If PS0=0, select Pattern ByteMask (0 ~ 3) If PS0=1, select Pattern ByteMask (4 ~ 7)

WOLCR0.s WOLCR0.c	7	PTNMH7	R/W1	In WOLCR0.s, write 1 means set (enable) pattern 0~7 WOL, write 0 has no effect. In WOLCR0.c, write 1 means clear (disable) pattern 0~7 WOL, write 0 has no effect. Read value is as the same as WOLCR0.s.
	6	PTNMH6	R/W1	
	5	PTNMH5	R/W1	
	4	PTNMH4	R/W1	
	3	PTNMH3	R/W1	
	2	PTNMH2	R/W1	
	1	PTNMH1	R/W1	
	0	PTNMH0	R/W1	
WOLSR0.s WOLSR0.c	7	PTNMH7I	R/W1	Internal updated by hardware for WOL patterns 0~7 are detected, write 1 in WOLSR0.s means set for testing purpose, write 0 has no effect. Write 1 in WOLSR0.c can clear status indication.
	6	PTNMH6I	R/W1	
	5	PTNMH5I	R/W1	
	4	PTNMH4I	R/W1	
	3	PTNMH3I	R/W1	
	2	PTNMH2I	R/W1	
	1	PTNMH1I	R/W1	
	0	PTNMH0I	R/W1	

6.1.2 The Procedure of Enabling Pattern Match WOL

Figure 16 shows the concept of adding a WOL pattern to MAC. Driver gets a pattern and its byte mask, calculates the 16-bit CRC, then writes byte mask and CRC to the corresponding registers. There are 2 pages to store the byte masks, driver can use Bit6(PS0) of Register 0x69(CAMCR) to select page.

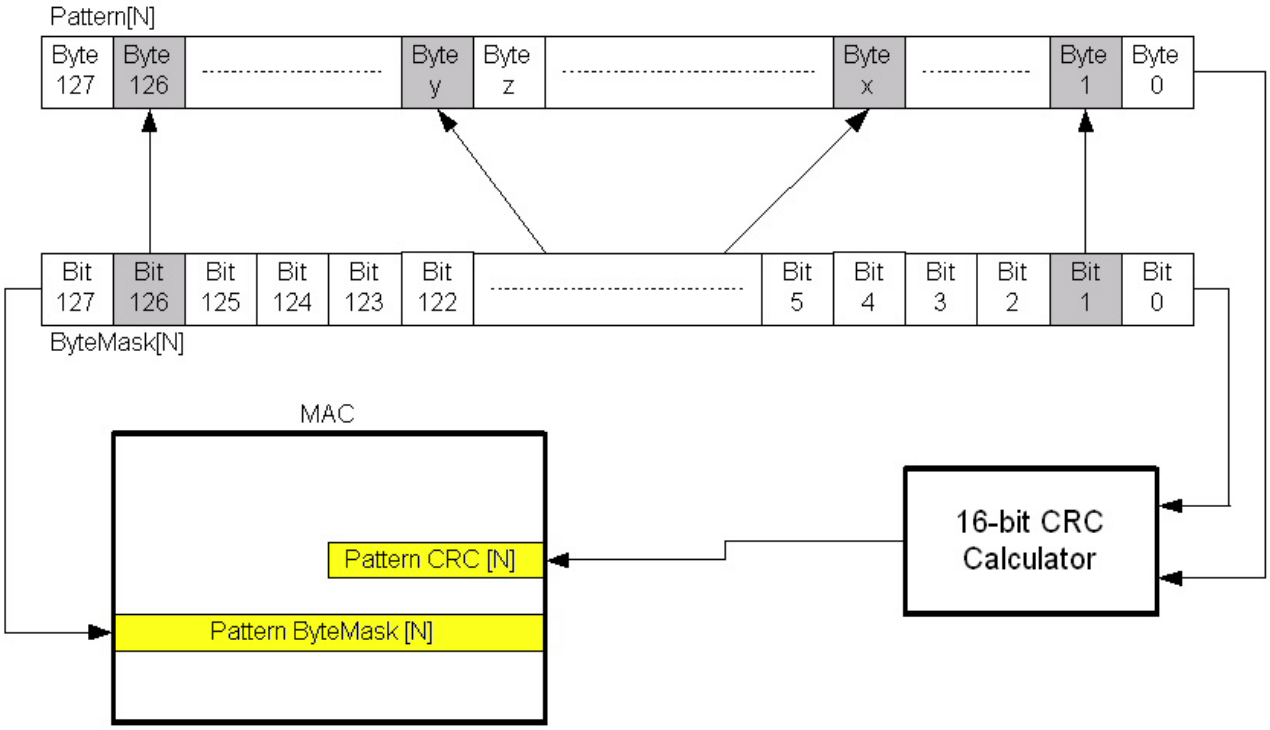


Figure 16. Add a WOL pattern to MAC

Because Byte Mask has only 4 DWORDs (16 bytes, 128 bits) length, that is, it can mask 128 bytes at most. If pattern length is greater than 128, the pattern contents after the 128th byte have no effect.

As for the algorithm of calculating 16-bit CRC from the pattern and its byte mask, please refer to the sample code in **Section 7.9**.

After filling pattern CRC and byte mask, driver must set the corresponding bit in WOLCR0.s as 1 to enable the pattern. Before the machine going into power state, driver should shutdown MAC, save important register contents, clear WOLSR to avoid trigger the WOL event, turn on Bit4(SWPTAG) of Register 0x83(STKSHDW).

If the machine is woke up by one of the patterns, driver should turn off Bit4(SWPTAG), Bit1(DS1), and Bit0(DS0) of Register 0x83(STKSHDW), clear WOLCR to disable all power events, restore important register contents. By checking Register WOLSR0(0xA4 or 0xA8), driver can know which pattern woke up the machine.

6.2 Magic Packet WOL

Magic packet is a special packet with special payload segment that is **16 duplications** of the MAC address of the NIC to be woke up. For example, if the MAC address of the machine we want to wake up is 00:11:22:33:44:55, the magic packet we need is like Table 25. Table 26 and Table 27 list related registers to enable Magic Packet WOL function.

Table 25. Magic Packet Format Example

Destination Address						Source Address						MISC		FF	FF
FF	FF	FF	FF	00	11	22	33	44	55	00	11	22	33	44	55
00	11	22	33	44	55	00	11	22	33	44	55	00	11	22	33
44	55	00	11	22	33	44	55	00	11	22	33	44	55	00	11
22	33	44	55	00	11	22	33	44	55	00	11	22	33	44	55
00	11	22	33	44	55	00	11	22	33	44	55	00	11	22	33
44	55	00	11	22	33	44	55	00	11	22	33	44	55	00	11
22	33	44	55	MISC…								CRC32			

6.2.1 Related Registers

Table 26. Magic Packet WOL Related Registers

	Registers			
Offset	+3h	+2h	+1h	+0h
A0h			WOLCR1.s	
A4h			WOLCR1.c	

A8h			WOLSR1.s	
ACh			WOLSR1.c	

Table 27. Layout of Registers for Magic Packet WOL

Register	Bit	Symbol	R/W	Description
WOLCR1.s WOLCR1.c	1	MAGICEN	R/W1	Write MAGICEN=1 in WOLCR1.s means enable WOL event detection for Magic Packet. Write MAGICEN=1 in WOLCR1.c means disable WOL event detection for Magic Packet. Write 0 has no effect.
WOLSR1.s WOLSR1.c	1	MAGICINT	R/W1	Internal updated by hardware for Magic Packet WOL event detected. Write MAGICINT=1 in WOLSR1.c can clear status indication.

6.2.2 The Procedure of Enabling Magic Packet WOL

The procedure to enable Magic Packet WOL function is simple:

(1) Set the bit MAGICEN=1 in Register 0xA1(WOLCR1.s).

(2) Do the same procedure as Pattern Match WOL does before/after power state of the machine.

(3) By checking WOLSR1, driver can figure out if the machine is woke up by Magic Packet.

7 SAMPLE CODE

7.1 Packet Transmission

```
STATUS
SendPacket(PPACKET pPacket)
{
    // Query this packet for informations
    QueryPacket(
            pPacket,
            &PhysicalSegmentNumber, // Get physical memory segment number
            &BufferList, // Get the addresses of all memory segments
            &PacketSize // Get this packet size
            );

    // Get current TD index
    uCurrDescIdx = TD_Index_For_Send;
    pCurrTD = TD[uCurrDescIdx];

    // Reset Priority and VID field of current TD
    pCurrTD->PQINF = 0;

    // Reset TCR and turn on TIC of current TD
    pCurrTD->TCR = 0;
    pCurrTD->TIC = 1;

    // Set TCPLS field of current TD
    pCurrTD->TCPLS = 0x3;

    // Set TxPktSize field of current TD
    pCurrTD->TxPktSize = PacketSize;

    // If the Packet is too fragmented copy the packet into a single buffer
    if (cbPktPhysSegment > 7) {
        // Copy the Packet into a single Tx Buffer prepared by this driver
        CopyFromPacketToBuffer(
                        pPacket,
                        &TxDataBuffer[uCurrDescIdx],
                        PacketSize,
                        );

        // Set TxBufSeg address and size
        pCurrTD->TxBufAddrLo[0] = Low_Address of TxDataBuffer[uCurrDescIdx];
        pCurrTD->TxBufAddrHi[0] = High_Address of TxDataBuffer[uCurrDescIdx];
        pCurrTD->TxBufSize[0] = PacketSize;

        // Set CMDZ = Segment# + 1
        pCurrTD->CMDZ = 2;
    }
    else {
        for (ii = 0; ii < PhysicalSegmentNumber; ii++) {
            pCurrTD->TxBufAddrLo[ii] = Low_Address of BufferList[ii];
```

```
            pCurrTD->TxBufAddrHi[ii] = High_Address of BufferList[ii];
            pCurrTD->TxBufSize[ii] = Length of BufferList[ii];
        }

        // Set CMDZ = Segment# + 1
        pCurrTD->CMDZ = PhysicalSegmentNumber + 1;
    }

    // Set OWN bit of current TD
    pCurrTD->OWN = 1;

    // Point to the next TD in the ring
    ADD_N_WITH_WRAP_AROUND(TD_Index_For_Send, 1, TD_Size);

    // Record transmitted pPacket
    TxPacketRecord[uCurrDescIdx] = pPacket;

    // Poll Transmit the adapter
    MACvTransmit();

    return TxStatus;
}


void
MACvTransmit()
{
    // Set MAC_Register_TDCSR_SET.WAKE0 = 1
    outportb(MAC_REGISTER_BASE_ADDRESS + 0x34, 0x04);
}
```

7.2 Packet Reception and Interrupt Handling

```
void
RxInterruptHandler()
{
    // Walk down the receive descriptors ring starting at the
    // last known descriptor owned by the adapter
    //
    // Examine each receive ring descriptor for errors.
    //
    // When we have the entire packet (and error processing doesn't
    // prevent us from indicating it), we give the routine that
    // processes the packet through the filter, the buffers virtual
    // address (which is always the lookahead size) and as the
    // MAC context the address of the first data byte.

    RxPktHandled = 0;
    ThisTimeFreeRD = 0;
    uRxFreeRDIdx = RD_Index_For_Interrupt;

    while (TRUE) {
        // If handle too much RD, exit this loop
```

```
if (RxPktHandled >= CB_MAX_RECEIVED_PACKETS) {
    break;
}

uCurrDescIdx = RD_Index_For_Interrupt;
pCurrRD = RD[uCurrDescIdx];
Rsr = pCurrRD->RSR;

// If the RD is owned by the chip, we are done.
if (pCurrRD->OWN == 1) {
    break;
}

// Expect both STP and EDP are zero
if ( IsAnyBitsOn(Rsr, (RSR_STP | RSR_EDP)) ) {
    // Drop this RD, move to next RD
    uRxFreeRD++;
    ADD_N_WITH_WRAP_AROUND(RD_Index_For_Interrupt, 1, RD_Size);
    continue;
}

// get packet size
FrameSize = pCurrRD->RMBC - 4; // CRC length is 4

//
// RX ERROR
//
// error handle...
if ( IsBitOff(Rsr, RSR_RXOK) ) {
    if ( IsAnyBitsOn(Rsr, (RSR_CE | RSR_RL)) ) {
        // Don't drop Checksum Error and Length Error packet here
    }
    else {
        // Drop this RD, move to next RD
        uRxFreeRD++;
        ADD_N_WITH_WRAP_AROUND(RD_Index_For_Interrupt, 1, RD_Size);
        continue;
    }
}

// Packet length error if one of the following condition exists:
if ( (FrameSize < MIN_PACKET_LEN) || (FrameSize > MAX_PACKET_LEN) ) {
    // Drop this RD, move to next RD
    uRxFreeRD++;
    ADD_N_WITH_WRAP_AROUND(RD_Index_For_Interrupt, 1, RD_Size);
    continue;
}

if (Need_To_Check_If_Packet_Length_Is_Valid) {
    if (Rsr0 & RSR_RL) {
        // drop this RD
        uRxFreeRD++;
        ADD_N_WITH_WRAP_AROUND(RD_Index_For_Interrupt, 1, RD_Size);
        continue;
    }
```

```
        }

        //
        // RX OK
        //

        // Indicate Protocol Layer that this driver has received a packet
        IndicateReceive(……);

        // RD handled count increase
        RxPktHandled++;

        // Release this RD
        ThisTimeFreeRD++;
        // move index to the next RD
        ADD_N_WITH_WRAP_AROUND(RD_Index_For_Interrupt, 1, RD_Size);
    } // end while


    // Batch append RD here (4X RDs appended each time)
    // count the difference between uRxDequeDescIdx and uRxAppendDescIdx
    if ((OriginalFreeRD + ThisTimeFreeRD) >= 4) {
        // Calculate the number of RD to be free, it must be 4X
        ii = (OriginalFreeRD + ThisTimeFreeRD) - ((OriginalFreeRD + ThisTimeFreeRD) % 4);
        // Decrease the RD index
        SUB_N_WITH_WRAP_AROUND(uRxFreeRDIdx, 1, RD_Size);
        // Calculate the end RD index to be free in this time
        ADD_N_WITH_WRAP_AROUND(uRxFreeRDIdx, (ii - OriginalFreeRD), RD_Size);

        for (uu = 0; uu < ii; uu++) {
            RD[uRxFreeRDIdx]->OWN = 1;
            SUB_N_WITH_WRAP_AROUND(uRxFreeRDIdx, 1, RD_Size);
        }

        // Calculate to-be-free RD for next time
        OriginalFreeRD = (OriginalFreeRD + ThisTimeFreeRD) % 4;
    }
    else {
        // calculate to-be-free RD for next time
        OriginalFreeRD += ThisTimeFreeRD;
    }
}


void
TxInterruptHandler()
{
    // Walk down the transmit descriptors ring starting at the
    // last known descriptor owned by the adapter
    while (TRUE) {
        uCurrDescIdx = TD_Index_For_Interrupt;
        pCurrTD = TD[uCurrDescIdx];
        Tsr = pCurrTD->TSR;
        CMDZ = pCurrTD->CMDZ;
```

```
        // If current descriptor is not owned by the system, we are done.
        if (pCurrTD->OWN == 1) {
            break;
        }

        // Update the statistics based on the Transmit status
        if ( BITIsOff(Tsr, TSR_TERR) ) {
            TxStatus = STATUS_SUCCESS;
        }
        else {
            TxStatus = STATUS_FAILURE;
        }

        // Get pPacket from record
        pPacket = TxPacketRecord[uCurrDescIdx];
        // Clear record
        TxPacketRecord[uCurrDescIdx] = NULL;

        // Increase index to next
        ADD_N_WITH_WRAP_AROUND(TD_Index_For_Interrupt, 1, TD_Size);

        // Return packet resource and TxStatus to Protocol Layer here
        CompletePacket(pPacket, TxStatus);
    }
}


void
ISR(……)
{
    DWORD       dwIsrStatus;
    BOOL        bLinkPass;

    If (interrupt already disabled)
        return;

    // Read the interrupt field of the adapter's ISR
    InPortD(MAC_REGISTER_BASE_ADDRESS + 0x24, &dwIsrStatus);
    // Clear MAC previous interrupt
    OutPortD(MAC_REGISTER_BASE_ADDRESS + 0x24, dwIsrStatus);

    // If PCI resource was cleared, all ISR is 1, this is abnormal
    if (dwIsrStatus == 0xFFFFFFFFUL)
        return;

    // Check if the shared interrupt is recognized by the adapter
    // use IMR_MASK_VALUE to filter out UNKNOWN irq
    if ((dwIsrStatus & 0x037BFFFFUL) == 0)
        return;

    Disable MAC interrupts

    Handle error interrupts;

    if (ISR_SRCI is on) {
```

```
        Indicate link status change to Protocol;
        Enable MII Auto-Polling;
    }

    RxInterruptHandler();
    TxInterruptHandler();

    Handle other interrupts;

    Enable MAC interrupts;
}
```

7.3 Hardware Initialization

```
void
MACvInitialize(……)
{
    // Clear sticky bits
    Turn off MAC_STICKHW.(DS1 | DS0);

    // Disable force PME-enable
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_WOLCFG_CLR, 0x80);

    // Disable power-event config bit
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_WOLCR0_CLR, 0xFF);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_WOLCR1_CLR, 0x0F);

    // Clear power status
    OutPortW(MAC_REGISTER_BASE_ADDRESS + MAC_WOLSR0_CLR, 0xFFFF);

    // Do reset
    MACbSoftwareReset();

    // Issue AUTOLD in EECSR to reload eeprom
    Turn on MAC_EECSR.RELOAD;
    while (TRUE) {
        if (MAC_EECSR.RELOAD is Off)
            break;
    }

    // EEPROM reloaded will cause bit 0 in MAC_REG_CFGA turned on.
    Turn off MAC_CFGA.PACPI;

    // Suspend-well accept broadcast, multicast
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_WOLCFG_SET, 0x30);

    // Set backoff algorithm

    // set packet filter, don't receive any packet until Protocol set packet filter
    MACvSetPacketFilter(NONE);

    // Fill IMR
    OutPortD(MAC_REGISTER_BASE_ADDRESS + MAC_REG_IMR, 0x037BFFFFUL);
```

```
    Enable MAC interrupts;

    Enable MII Auto-Polling;

    Reset and turn on Tx;
    Reset and turn on Rx;

    Start MAC;
}

BOOL
MACbSoftwareReset(……)
{
    // turn on CR1_SFRST
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR1_SET, CR1_SFRST);

    Wait for MAC_CR1_SET.SFRST off with timeout mechanism;

    if (Wait is timeout) {
        // turn on force reset
        OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR3_SET, CR3_FORSRST);
        Delay 2ms;
    }

    return TRUE;
}

void
MACvSetPacketFilter(WORD wFilterType)
{
    BYTE    byOldRCR;
    BYTE    byNewRCR = 0;
    BOOL    bUnicast = FALSE;

    // if only in DIRECTED mode, multicast-address will set to zero,
    // but if other mode exist (e.g. PROMISCUOUS), multicast-address
    // will be open
    if (wFilterType is DIRECTED mode) {
        // set multicast address to accept none
        OutPortD(MAC_REGISTER_BASE_ADDRESS + MAC_MAR, 0L);
        OutPortD(MAC_REGISTER_BASE_ADDRESS + MAC_MAR + 4, 0L);

        // accept unicast packet
        bUnicast = TRUE;
    }

    if (wFilterType is PROMISCUOUS or ALL_MULTICAST mode) {
        // set multicast address to accept all
        OutPortD(MAC_REGISTER_BASE_ADDRESS + MAC_MAR, 0xFFFFFFFFL);
        OutPortD(MAC_REGISTER_BASE_ADDRESS + MAC_REG_MAR + 4, 0xFFFFFFFFL);
    }

    if (wFilterType is PKT_TYPE_PROMISCUOUS mode) {
        byNewRCR |= (RCR_PROM | RCR_AP | RCR_AM | RCR_AB);
```

```
        // accept unicast packet
        bUnicast = TRUE;
    }

    if (wFilterType is MULTICAST or ALL_MULTICAST mode)
        byNewRCR |= (RCR_AP | RCR_AM);

    if (wFilterType is BROADCAST mode)
        byNewRCR |= RCR_AB;

    InPortB(MAC_REGISTER_BASE_ADDRESS + MAC_RCR,  &byOldRCR);
    if (byNewRCR != (byOldRCR & 0x1F)) {
        byNewRCR |= (byOldRCR & 0xE0);
        OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_RCR, byNewRCR);
    }

    if (bUnicast) {
        // accept unicast packet
        OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR1_CLR, CR1_DISAU);
    }
    else {
        // reject unicast packet
        OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR1_SET, CR1_DISAU);
    }
}

void
vSafeResetTx(……)
{
    // initialize TD index
    TD_Index_For_Send = 0;
    TD_Index_For_Interrupt = 0;

    // init state, all TD is host's
    for (uu = 0; uu < TD#; uu++) {
        TD[uu].OWN = 0;
        TD[uu].QUE = 0;
    }

    Set MAC's TD_Base_Address register;
    Set MAC's TD_Index register = 0;
    Set MAC's TD_Size register = (TD# - 1);
}

void
vSafeResetRx()
{
    // initialize RD index
    TD_Index_For_Interrupt = 0;
    OriginalFreeRD = 0;

    // init state, all RD is chip's
    for (uu = 0; uu < RD#; uu++) {
        RD[uu].OWN = 1;
    }
```

```
    Set MAC's RD_Base_Address register;
    Set MAC's RD_Index register = 0;
    Set MAC's RD_Size register = (RD# - 1);
}

void
MACvRxOn(……)
{
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_CLR, CR0_STOP);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_SET, CR0_STRT);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_SET, CR0_RXON);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_RDCSR0_SET, 0x5);
}

void
MACvTxOn(……)
{
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_CLR, CR0_STOP);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_SET, CR0_STRT);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_SET, CR0_TXON);
    OutPortW(MAC_REGISTER_BASE_ADDRESS + MAC_TDCSR0_SET, 0x1111);
    OutPortW(MAC_REGISTER_BASE_ADDRESS + MAC_TDCSR0_SET, 0x4444);
}

void
MACvStart(……)
{
    OutPortB(dwIoBase + MAC_REG_CR0_CLR, CR0_STOP);
    OutPortB(dwIoBase + MAC_REG_CR0_SET, CR0_STRT);
    OutPortB(dwIoBase + MAC_REG_CR0_SET, (CR0_RXON | CR0_TXON));
}
```

7.4 Hardware Shutdown

```
BOOL
MACbShutdown(……)
{
    // Disable interrupt
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR3_CLR, CR3_GINTMSK1);

    // Disable MII auto-polling
    vDisableMiiAutoPoll(……);

    // Stop the adapter
    if (!MACbSafeStop(……))
        return FALSE;

    return TRUE;
}

BOOL
MACbSafeStop(……)
```

```
{
    if (!MACbSafeTxOff(……))
        return FALSE;
    if (!MACbSafeRxOff(……))
        return FALSE;
    if (!MACbStop(……))
        return FALSE;

    return TRUE;
}

BOOL
MACbSafeTxOff(……)
{
    // Clear RUN Tx
    OutPortW(MAC_REGISTER_BASE_ADDRESS + MAC_TDCSR0_CLR, 0x1111);

    // Try to safe shutdown TX
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_CLR, CR0_TXON);

    Delay a safe period;

    Wait for MAC_CR0_CLR.TXON off with timeout mechanism;

    if (Wait for MAC_CR0_CLR.TXON off is timeout)
        return FALSE;

    return TRUE;
}

BOOL
MACbSafeRxOff(……)
{
    // Clear RUN Rx
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_RDCSR0_CLR, TRDCSR_RUN);

    // Shutdown RXON
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_CLR, CR0_RXON);

    Delay a safe period;

    Wait for MAC_CR0_CLR.RXON off with timeout mechanism;

    if (Wait for MAC_CR0_CLR.RXON off is timeout)
        return FALSE;

    return TRUE;
}

BOOL
MACbStop(……)
{
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_CR0_SET, CR0_STOP);

    Wait for MAC_CR0_SET.(TXON | RXON) off with timeout mechanism;
```

```
    if (Wait is timeout)
        return FALSE;

    return TRUE;
}
```

7.5 Macros

```
#define ADD_N_WITH_WRAP_AROUND(uVar, uNum, uModulo) { \
    if (((uVar) + (uNum)) > ((uModulo) - 1)) \
        (uVar) = (uVar) + (uNum) - (uModulo); \
    else \
        (uVar) += (uNum); \
}


#define SUB_N_WITH_WRAP_AROUND(uVar, uNum, uModulo) { \
    if (((uVar) - (uNum)) < 0) \
        (uVar) = (uModulo) - (uNum) + (uVar); \
    else \
        (uVar) -= (uNum); \
}
```

7.6 Multicast CAM Read/Write

```
// Read Multicast CAM, pbyData should be 6-byte array
void GetMCAM (MAC_REGISTER_BASE_ADDRESS, BYTE byAddress, PBYTE pbyData)
{
    BYTE    byOrgCAMCR, byData;

    // Select CAM DATA regs
    InPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, &byOrgCAMCR);
    byData = (BYTE)((byOrgCAMCR | CAMCR_PS1) & ~CAMCR_PS0);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byData);

    // Enable/Select MCAM
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMADDR, (BYTE)(CAMADDR_CAMEN | byAddress));

    // Issue read command
    MACvRegBitsOn(MAC_REGISTER_BASE_ADDRESS, MAC_REG_CAMCR, CAMCR_CAMRD);

    // Wait for CAMRD self clear
    while (TRUE)
        if (MACbIsRegBitsOff(MAC_REGISTER_BASE_ADDRESS, MAC_REG_CAMCR, CAMCR_CAMRD))
            break;

    // Get Multicast Address
    InPortD(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAM, (PDWORD)pbyData);
    InPortW(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAM + 4, (PWORD)(pbyData + 4));

    // Turn off CAMEN
```

```
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMADDR, 0);

    // Restore original CAMCR
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byOrgCAMCR);
}


// Read Multicast CAM Mask, pbyMask is 8-byte array
void GetMCAMMask (MAC_REGISTER_BASE_ADDRESS, PBYTE pbyMask)
{
    BYTE    byOrgCAMCR, byData;

    // Select CAM MASK regs
    InPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, &byOrgCAMCR);
    byData = (BYTE)((byOrgCAMCR & ~CAMCR_PS1) | CAMCR_PS0);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byData);

    // Select MCAM Mask
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMADDR, 0);

    // Read CAM mask data (don't need to set CAMEN)
    InPortD(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAM, (PDWORD)pbyMask);
    InPortD(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAM + 4, (PDWORD)(pbyMask + 4));

    // Restore original CAMCR
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byOrgCAMCR);
}


// Write Multicast CAM, pbyData should be 6-byte array
void SetMCAM (MAC_REGISTER_BASE_ADDRESS, BYTE byAddress, PBYTE pbyData)
{
    BYTE    byOrgCAMCR, byData;
    UINT    uu;

    // Select CAM DATA regs
    InPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, &byOrgCAMCR);
    byData = (BYTE)((byOrgCAMCR | CAMCR_PS1) & ~CAMCR_PS0);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byData);

    // Enable/Select MCAM
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMADDR, (BYTE)(CAMADDR_CAMEN | byAddress));

    // set MCAM Data
    for (uu = 0; uu < 6; uu++) {
        OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAM + uu, *(pbyData + uu));
    }

    // Issue write command
    MACvRegBitsOn(MAC_REGISTER_BASE_ADDRESS, MAC_REG_CAMCR, CAMCR_CAMWR);

    // Wait for CAMWR self clear
    while (TRUE)
        if (MACbIsRegBitsOff(MAC_REGISTER_BASE_ADDRESS, MAC_REG_CAMCR, CAMCR_CAMWR))
            break;
```

```
    // Turn off CAMEN
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMADDR, 0);

    // Restore original CAMCR
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byOrgCAMCR);
}


// Write Multicast CAM Mask, pbyMask is 8-byte array
void SetMCAMMask (MAC_REGISTER_BASE_ADDRESS, PBYTE pbyMask)
{
    BYTE    byOrgCAMCR, byData;
    UINT    uu;

    // Select CAM MASK regs
    InPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, &byOrgCAMCR);
    byData = (BYTE)((byOrgCAMCR & ~CAMCR_PS1) | CAMCR_PS0);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byData);

    // Select MCAM Mask
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMADDR, 0);

    for (uu = 0; uu < 8; uu++) {
        OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAM + uu, *(pbyMask + uu));
    }

    // Restore original CAMCR
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byOrgCAMCR);
}
```

7.7 Multicast Hash Value Algorithm

```
// The algorithm to calculate hash vale,
// where pbyMultiAddr is the address of the byte array of multicast address
BYTE GetHashIndexByCrc32 (PBYTE pbyMultiAddr)
{
    int     ii;
    BYTE    byTmpHash;
    BYTE    byHash = 0;

    // get the least 6-bits from CRC generator, seed is 0xFFFFFFFFL
    byTmpHash = (BYTE)(CRCdwCrc32(pbyMultiAddr, U_ETHER_ADDR_LEN, 0xFFFFFFFFL) & 0x3F);

    // reverse most bit to least bit
    for (ii = 0; ii < (sizeof(byTmpHash) * 8); ii++) {
        byHash <<= 1;
        if (BITbIsBitOn(byTmpHash, 0x01))
            byHash |= 1;
        byTmpHash >>= 1;
    }

    // adjust 6-bits to the right most
    return (byHash >> 2);
```

```
}


// Set hash index into multicast address register bit
void SetMultiAddrByHash (MAC_REGISTER_BASE_ADDRESS, BYTE byHashIdx)
{
    UINT    uByteIdx;
    BYTE    byBitMask;
    BYTE    byOrgValue;
    BYTE    byOrgCAMCR;
    BYTE    byData;

    // Select MAR regs
    InPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, &byOrgCAMCR);
    byData = byOrgCAMCR & ~(CAMCR_PS1 | CAMCR_PS0);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byData);

    // Calculate byte position
    uByteIdx = byHashIdx / 8;
    ASSERT(uByteIdx < 8);

    // Calculate bit position
    byBitMask = 1;
    byBitMask <<= (byHashIdx % 8);

    // Turn on the bit
    InPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_MAR + uByteIdx, & byOrgValue);
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_MAR + uByteIdx, (BYTE)(byOrgValue | byBitMask));

    // Restore original CAMCR
    OutPortB(MAC_REGISTER_BASE_ADDRESS + MAC_REG_CAMCR, byOrgCAMCR);
}


// Add multicast address hashing table
BYTE byHash = GetHashIndexByCrc32(pbyMultiAddr);
SetMultiAddrByHash(MAC_REGISTER_BASE_ADDRESS, byHashNum);
```


7.8 32-bit CRC Calculation

The table for calculating 32-bit CRC.

```
// 32-bit CRC table
const DWORD Crc32Table[256] = {
    0x00000000L, 0x77073096L, 0xEE0E612CL, 0x990951BAL, 0x076DC419L, 0x706AF48FL, 0xE963A535L, 0x9E6495A3L,
    0x0EDB8832L, 0x79DCB8A4L, 0xE0D5E91EL, 0x97D2D988L, 0x09B64C2BL, 0x7EB17CBDL, 0xE7B82D07L, 0x90BF1D91L,
    0x1DB71064L, 0x6AB020F2L, 0xF3B97148L, 0x84BE41DEL, 0x1ADAD47DL, 0x6DDDE4EBL, 0xF4D4B551L, 0x83D385C7L,
    0x136C9856L, 0x646BA8C0L, 0xFD62F97AL, 0x8A65C9ECL, 0x14015C4FL, 0x63066CD9L, 0xFA0F3D63L, 0x8D080DF5L,
    0x3B6E20C8L, 0x4C69105EL, 0xD56041E4L, 0xA2677172L, 0x3C03E4D1L, 0x4B04D447L, 0xD20D85FDL, 0xA50AB56BL,
    0x35B5A8FAL, 0x42B2986CL, 0xDBBBC9D6L, 0xACBCF940L, 0x32D86CE3L, 0x45DF5C75L, 0xDCD60DCFL, 0xABD13D59L,
    0x26D930ACL, 0x51DE003AL, 0xC8D75180L, 0xBFD06116L, 0x21B4F4B5L, 0x56B3C423L, 0xCFBA9599L, 0xB8BDA50FL,
    0x2802B89EL, 0x5F058808L, 0xC60CD9B2L, 0xB10BE924L, 0x2F6F7C87L, 0x58684C11L, 0xC1611DABL, 0xB6662D3DL,
    0x76DC4190L, 0x01DB7106L, 0x98D220BCL, 0xEFD5102AL, 0x71B18589L, 0x06B6B51FL, 0x9FBFE4A5L, 0xE8B8D433L,
    0x7807C9A2L, 0x0F00F934L, 0x9609A88EL, 0xE10E9818L, 0x7F6A0DBBL, 0x086D3D2DL, 0x91646C97L, 0xE6635C01L,
```

```
    0x6B6B51F4L, 0x1C6C6162L, 0x856530D8L, 0xF262004EL, 0x6C0695EDL, 0x1B01A57BL, 0x8208F4C1L, 0xF50FC457L,
    0x65B0D9C6L, 0x12B7E950L, 0x8BBEB8EAL, 0xFCB9887CL, 0x62DD1DDFL, 0x15DA2D49L, 0x8CD37CF3L, 0xFBD44C65L,
    0x4DB26158L, 0x3AB551CEL, 0xA3BC0074L, 0xD4BB30E2L, 0x4ADFA541L, 0x3DD895D7L, 0xA4D1C46DL, 0xD3D6F4FBL,
    0x4369E96AL, 0x346ED9FCL, 0xAD678846L, 0xDA60B8D0L, 0x44042D73L, 0x33031DE5L, 0xAA0A4C5FL, 0xDD0D7CC9L,
    0x5005713CL, 0x270241AAL, 0xBE0B1010L, 0xC90C2086L, 0x5768B525L, 0x206F85B3L, 0xB966D409L, 0xCE61E49FL,
    0x5EDEF90EL, 0x29D9C998L, 0xB0D09822L, 0xC7D7A8B4L, 0x59B33D17L, 0x2EB40D81L, 0xB7BD5C3BL, 0xC0BA6CADL,
    0xEDB88320L, 0x9ABFB3B6L, 0x03B6E20CL, 0x74B1D29AL, 0xEAD54739L, 0x9DD277AFL, 0x04DB2615L, 0x73DC1683L,
    0xE3630B12L, 0x94643B84L, 0x0D6D6A3EL, 0x7A6A5AA8L, 0xE40ECF0BL, 0x9309FF9DL, 0x0A00AE27L, 0x7D079EB1L,
    0xF00F9344L, 0x8708A3D2L, 0x1E01F268L, 0x6906C2FEL, 0xF762575DL, 0x806567CBL, 0x196C3671L, 0x6E6B06E7L,
    0xFED41B76L, 0x89D32BE0L, 0x10DA7A5AL, 0x67DD4ACCL, 0xF9B9DF6FL, 0x8EBEEFF9L, 0x17B7BE43L, 0x60B08ED5L,
    0xD6D6A3E8L, 0xA1D1937EL, 0x38D8C2C4L, 0x4FDFF252L, 0xD1BB67F1L, 0xA6BC5767L, 0x3FB506DDL, 0x48B2364BL,
    0xD80D2BDAL, 0xAF0A1B4CL, 0x36034AF6L, 0x41047A60L, 0xDF60EFC3L, 0xA867DF55L, 0x316E8EEFL, 0x4669BE79L,
    0xCB61B38CL, 0xBC66831AL, 0x256FD2A0L, 0x5268E236L, 0xCC0C7795L, 0xBB0B4703L, 0x220216B9L, 0x5505262FL,
    0xC5BA3BBEL, 0xB2BD0B28L, 0x2BB45A92L, 0x5CB36A04L, 0xC2D7FFA7L, 0xB5D0CF31L, 0x2CD99E8BL, 0x5BDEAE1DL,
    0x9B64C2B0L, 0xEC63F226L, 0x756AA39CL, 0x026D930AL, 0x9C0906A9L, 0xEB0E363FL, 0x72076785L, 0x05005713L,
    0x95BF4A82L, 0xE2B87A14L, 0x7BB12BAEL, 0x0CB61B38L, 0x92D28E9BL, 0xE5D5BE0DL, 0x7CDCEFB7L, 0x0BDBDF21L,
    0x86D3D2D4L, 0xF1D4E242L, 0x68DDB3F8L, 0x1FDA836EL, 0x81BE16CDL, 0xF6B9265BL, 0x6FB077E1L, 0x18B74777L,
    0x88085AE6L, 0xFF0F6A70L, 0x66063BCAL, 0x11010B5CL, 0x8F659EFFL, 0xF862AE69L, 0x616BFFD3L, 0x166CCF45L,
    0xA00AE278L, 0xD70DD2EEL, 0x4E048354L, 0x3903B3C2L, 0xA7672661L, 0xD06016F7L, 0x4969474DL, 0x3E6E77DBL,
    0xAED16A4AL, 0xD9D65ADCL, 0x40DF0B66L, 0x37D83BF0L, 0xA9BCAE53L, 0xDEBB9EC5L, 0x47B2CF7FL, 0x30B5FFE9L,
    0xBDBDF21CL, 0xCABAC28AL, 0x53B39330L, 0x24B4A3A6L, 0xBAD03605L, 0xCDD70693L, 0x54DE5729L, 0x23D967BFL,
    0xB3667A2EL, 0xC4614AB8L, 0x5D681B02L, 0x2A6F2B94L, 0xB40BBE37L, 0xC30C8EA1L, 0x5A05DF1BL, 0x2D02EF8DL
    };


// Generate a CRC-32 from the data stream
DWORD CRCdwCrc32 (PBYTE pbyData, UINT cbByte, DWORD dwCrcSeed)
{
    DWORD dwCrc;

    dwCrc = dwCrcSeed;
    while (cbByte--) {
        dwCrc = Crc32Table[(BYTE)((dwCrc ^ (*pbyData)) & 0xFF)] ^ (dwCrc >> 8);
        pbyData++;
    }

    return dwCrc;
}
```


7.9 16-bit CRC Calculation

The table for calculating 16-bit CRC.

```
// CCITT CRC16 Table
const WORD CCITTCRC16Table[256] = {
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
    0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
    0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
    0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
    0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
    0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
    0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
    0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
```

```
    0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
    0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
    0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
    0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
    0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
    0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
    0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
    0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
    0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
    0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
    0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
    0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
    0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
    0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
    0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
    0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
    0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
    0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
    0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
    0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
    0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
    0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
    0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
    };
```

The main 16-bit calculator, where pbyMaskData is the address of the byte array of the byte mask, pbyPatData is the address of the byte array of the pattern, dwDataCount is the length of the pattern.

```
WORD wGenPatCrc16 (BYTE *pbyMaskData, BYTE *pbyPatData, DWORD dwDataCount)
{
    UINT    uu, uBitIdx;
    BYTE    byMask;
    WORD    wCrc = 0xFFFF;


    for (uu = 0; uu < ((DWORD)dwDataCount / 8); uu++) {
        byMask = pbyMaskData[uu];
        for (uBitIdx = 0; uBitIdx < 8; uBitIdx++) {
            if ((byMask & 0x01) == 0) {
                byMask >>= 1;
                continue;
            }
            byMask >>= 1;

            wCrc = CRC16(wCrc, &(pbyPatData[uu * 8 + uBitIdx]), 1);
        }
    }

    // if pattern-data-count is not the multiple of 8
    if ((dwDataCount % 8) != 0) {
        byMask = pbyMaskData[uu];
        for (uBitIdx = 0; uBitIdx < (dwDataCount % 8); uBitIdx++) {
            if ((byMask & 0x01) == 0) {
                byMask >>= 1;
```

```
                continue;
            }
            byMask >>= 1;

            wCrc = CRC16(wCrc, &(pbyPatData[uu * 8 + uBitIdx]), 1);
        }
    }

    // finally, must inversion once to get the correct data
    wCrc = ~wCrc;
    // for match hardware, reverse bit order
    wCrc = wBitReverseForWord(wCrc);

    return wCrc;
}


WORD CRC16 (WORD wCRC, BYTE* cp, int len)
{
    while (len--)
        wCRC = (wCRC >> 8) ^ CCITTCRC16Table[(wCRC ^ *cp++) & 0xff];

    return wCRC;
}


WORD wBitReverseForWord (WORD wData)
{
    DWORD   dwNewData = 0x00000000;
    WORD    wMask = 0x0001;
    int     ii;


    for (ii = 0; ii < 16; ii++) {
        dwNewData |= ((DWORD)(wData & (wMask << ii)) << (31 - ii*2));
    }

    return (WORD)(dwNewData >> 16);
}
```

APPENDIX

A.1 VIA Networking Velocity Family Models Reference Table

This table lists the functions comparison between all Rhine Family models.

Model	VT6120	VT6122
VendorID	0x1106	
DeviceID	0x3119	
RevisionID	0x10~1F	
Shadow EEPROM	●	
Boot ROM Interface	●	
802.1p/Q Tagging	●	
Checksum Offload	●	
TCP Large Send Offload	●	
Priority Transmit	●	
Jumbo Frame	●	
Full-duplex Tx Flow Control	●	
Full-duplex Rx Flow Control	●	
Tx Buffer Byte Alignment	●	
CAM for Multiple VLAN	●	
CAM for Multicast Address perfect match	●	
Hardware MIB Counter	●	
Adaptive Interrupt	●	
WOL Pattern Match Slots	8	
Rx Fifo	48KB	
Tx Fifo	16KB	

A.2 Suspend Well Registers

This table lists the registers that are still alive in power suspend mode.

Model			VT6120	VT6122
Offset	Register(s)	Bit(s)		
MAC Registers				
0x00~05	PAR[0~5]	All	●	
0x10~17	MAR/CAM[0~7]	All	●	
0x6C	MIICFG	All	●	
0x78	CFGA	Bit 0~5	●	
0x79	CFGB	Bit 5~7	●	
0x83	STICKHW	Bit 0~4	●	
0x9F	CHIP_GCR	Bit 3~7	●	
0xA0~A1/0xA4~A5	WOLCR.S/C	All	●	
0xA2/0xA6	PWCFG.S/C	All	●	
0xA3/0xA7	WOLCFG.S/C	All	●	
0xA8~A9/0xAC~AD	WOLSR.S/C	All	●	
0xAA~AB/0xAE~AF	Reserved	All	●	
0xB0~BF	PatternCRC[0~7]	All	●	
0xC0~FF	ByteMask[0~7]	All	●	
PCI Configuration Space				
0x55	PMCSR1	Bit 0 Bit 7	●	
0x5F	VEE_CTL	Bit 0	●	

A.3 Software Reset Un-Modified Registers

This table lists the registers that are **NOT** modified by software reset.

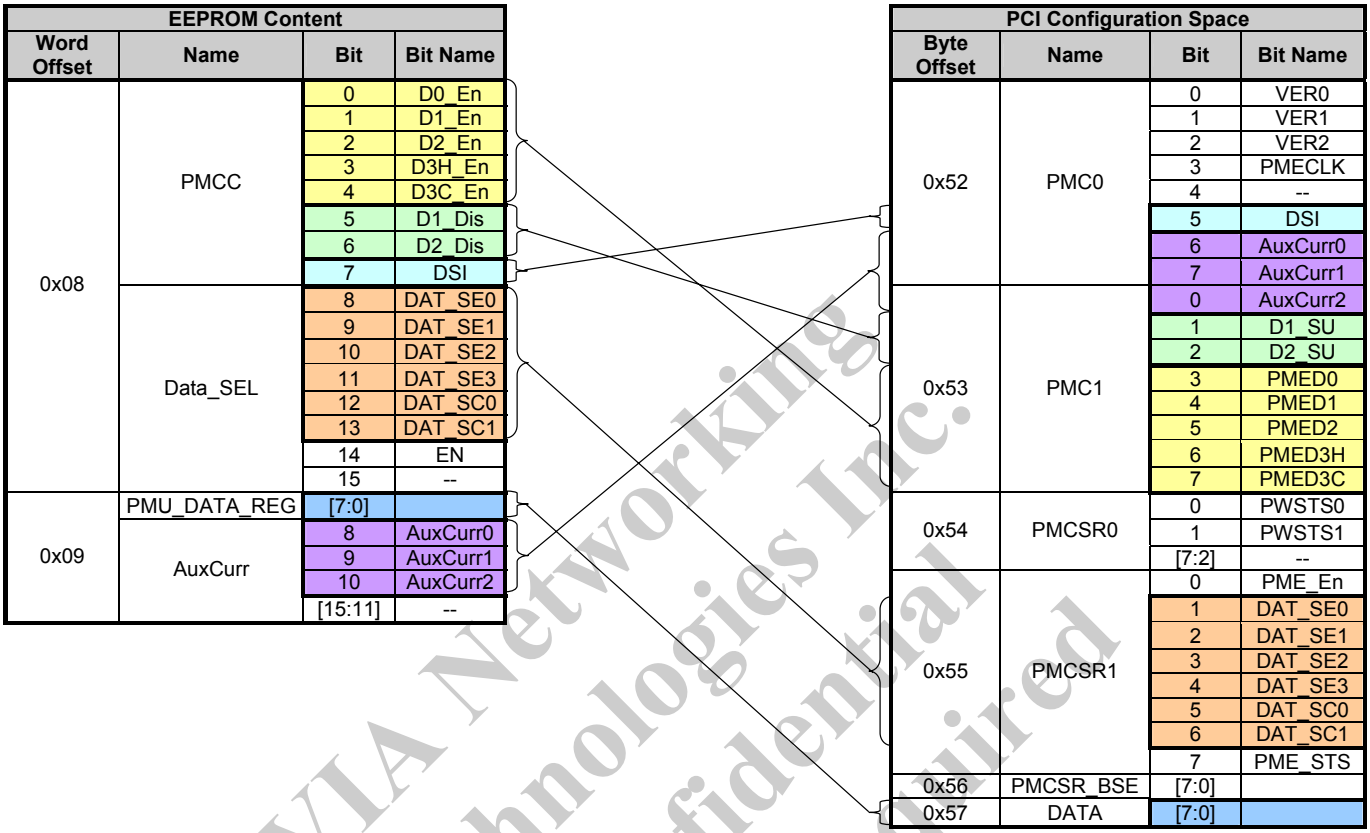
Model			VT6120	VT6122
Offset	**Register(s)**	**Bit(s)**		
MAC Registers				
0x00~05	PAR[0~5]	All	●	
0x6C	MIICFG	All	●	
0x78	CFGA	Bit 1~7	●	
0x79	CFGB	All	●	
0x7A	CFGC	All	●	
0x7B	CFGD	All	●	
0x7C	DCFG0	All	●	
0x7D	DCFG1	All	●	
0x7E	MCFG0	All	●	
0x7F	MCFG1	All	●	
0x83	STICKHW	Bit 0~4	●	
0x92	CHKSUM	All	●	
0x9C	CHIP_GSR	Bit 4	●	
0x9D	TESTCFG	Bit 4~7	●	
0x9F	CHIP_GCR	Bit 3~7	●	
0xA0~A1/0xA4~A5	WOLCR.S/C	All	●	
0xA2/0xA6	PWCFG.S/C	Bit 0~3	●	
0xA3/0xA7	WOLCFG.S/C	All	●	
PCI Configuration Space				
All		All	●	

A.4 EEPROM Layout Reference

This table lists the MAC Registers / PCI Configuration Space that can be loaded from EEPROM

Model / PCI or MAC Register			VT6120	VT6122
EEPROM Content				
Word Offset	Bit	Name		
0x00~02	[47:0]	ETHER_ID	MAC[0x00~05]	
0x03	[7:0]	MII_PHYAD	MAC[0x6C]	
	[15:8]	SW_Reserved		
0x04	[15:0]	SUB_SID	PCI[0x2E~2F]	
0x05	[15:0]	SUB_VID	PCI[0x2C~2D]	
0x06	[15:0]	DEV_ID		
0x07	[15:0]	VEN_ID		
0x08	[7:0]	PMCC	MAC[0x82] / *PCI(see next table)	
	[15:8]	Data_SEL	*PCI(see next table)	
0x09	[7:0]	PMU_DATA_REG	*PCI(see next table)	
	[15:8]	AuxCurr	*PCI(see next table)	
0x0A	[7:0]	MCFG0	MAC[0x7E]	
	[15:8]	MCFG1	MAC[0x7F]	
0x0B	[7:0]	Min_GNT	PCI[0x3E]	
	[15:8]	Max_LAT	PCI[0x3F]	
0x0C	[7:0]	DCFG0	MAC[0x7C]	
	[15:8]	DCFG1	MAC[0x7D]	
0x0D	[7:0]	CFG_A	MAC[0x78]	
	[15:8]	CFG_B	MAC[0x79]	
0x0E	[7:0]	CFG_B	MAC[0x7A]	
	[15:8]	CFG_D	MAC[0x7B]	
0x0F	[7:0]	PROG_STATUS		
	[15:8]	CHKSUM		

This table shows the mapping between EEPROM word offset [0x08~0x09] and PCI Configuration Space



EEPROM Content			
Word Offset	Name	Bit	Bit Name
0x08	PMCC	0	D0_En
		1	D1_En
		2	D2_En
		3	D3H_En
		4	D3C_En
		5	D1_Dis
		6	D2_Dis
		7	DSI
	Data_SEL	8	DAT_SE0
		9	DAT_SE1
		10	DAT_SE2
		11	DAT_SE3
		12	DAT_SC0
		13	DAT_SC1
		14	EN
		15	--
0x09	PMU_DATA_REG	[7:0]	
	AuxCurr	8	AuxCurr0
		9	AuxCurr1
		10	AuxCurr2
		[15:11]	--

PCI Configuration Space			
Byte Offset	Name	Bit	Bit Name
0x52	PMC0	0	VER0
		1	VER1
		2	VER2
		3	PMECLK
		4	--
		5	DSI
		6	AuxCurr0
		7	AuxCurr1
0x53	PMC1	0	AuxCurr2
		1	D1_SU
		2	D2_SU
		3	PMED0
		4	PMED1
		5	PMED2
		6	PMED3H
		7	PMED3C
0x54	PMCSR0	0	PWSTS0
		1	PWSTS1
		[7:2]	--
0x55	PMCSR1	0	PME_En
		1	DAT_SE0
		2	DAT_SE1
		3	DAT_SE2
		4	DAT_SE3
		5	DAT_SC0
		6	DAT_SC1
		7	PME_STS
0x56	PMCSR_BSE	[7:0]	
0x57	DATA	[7:0]	