รายงานโครงการวิจัยเรื่อง

การออกแบบซอฟท์แวร์ควบคุมระบบโลคัลแอเรียเนตเวอร์ค

Protocol Development for Wireless Local Area Network

หัวหน้าโครงการ

นายสุวิพล สิทธิชีวภาค

สารบัญ

บทคัดย่อ

โครงการวิจัยเรื่อง การออกแบบซอฟท์แวร์ควบคุมระบบโลคัลแอเรียเนตเวอร์ค

Protocol Development for Wireless Local Area Network

หัวหน้าโครงการ นายสุวิพล สิทธิชีวภาค

บทคัดย่อภาษาไทย

ระบบโลคัลแอเรียเนตเวอร์คที่ใช้กันแพร่หลายในปัจจุบันนั้นก็คือระบบมีสายแบบอีเทอร์เนต สาเหตุที่ได้รับความนิยมอย่างแพร่หลายนั้นก็เนื่องมาจาก 1)อุปกรณ์ควบคุมมีราคาถูกเมื่อเปรียบเทียบกับระบบอื่นๆเช่นระบบโทเกนริงค์ 2)ส่วนควบคุมของแต่ละสถานีทำงานได้อย่างอิสระ ดังนั้นเมื่ออุปกรณ์ใดอุปกรณ์หนึ่งเกิดชำรุดเสียหาย ก็ไม่ทำให้ระบบทั้งหมดหยุดทำงาน 3)ระบบการควบคุมสื่อสารจะไม่มีสถานีส่วนกลางมาควบคุม ดังนั้นจะลดความยุ่งยากทางอิเลคทรอนิกส์ อย่างไรก็ตามในสภาพของโครงสร้างอาคารที่ยุ่งยากต่อการเชื่อมโยงสายเช่นในระบบโรงงานหรือสำนักงานที่อาคารแยกจากกันอยู่นั้นทำให้ระบบเนตเวอร์คนั้นจำเป็นที่จะต้องใช้แบบไร้สาย อย่างไรก็ตาม เมื่อระบบควบคุมแบบอิเตอร์เนตเดิมถูกพัฒนามาใช้กับระบบไร้สายนั้น จะมีปัญหาสองประการที่จะต้องแก้ไขคือ 1)ระบบสัญญาณส่งผ่านตัวกลางจำเป็นต้องเปลี่ยนแปลงให้เหมาะสมกับระบบวิทยุ 2)ซอฟท์แวร์ที่ใช้ควบคุมการทำงานของส่วนควบคุมของแต่ละสถานีจำเป็นที่จะต้องมีการดัดแปลงใช้เข้ากับระบบการจายข่าว

ดังนั้นสำหรับโครงงานวิจัยนี้จึงมีความประสงค์ที่จะพัฒนาซอฟท์แวร์โดยอาศัยชิ้นส่วนฮาร์ดแวร์เดิมของอิเตอร์เนตให้สามารใช้งานได้ทั้งระบบใช้สายและไร้สาย ซึ่งหมายถึงการเพิ่มฮาร์ดแวร์ที่จำเป็นและการเขียนโปรแกรมเพื่อใช้งานในระบบมีสายและไร้สาย

Abstract

Local network that widely used today is a cable-typed Ethernet network The reasons of popularity are that 1) Control equipments are cheaper compared to other system 2) Each station performs data transmission independently, then, system is not interrupted as station is broken down 3) Special control is not necessary, then, electronic equipment is simple. However, in some buildings are not convenience to arrange wire or in the case of temporary utilazation, the network should be changed in to wireless system. When the existed Ethernet system is developed to use in wireless enviornment, there are two problems that must be considered 1) Signalling system must be developed appropriate with wireless system 2) Control software of each station should be adapted into broadcasting system.

In this reseach, we combine some essential hardware in to the existed Ethernet system for wireless operation and design appropriate software for wireless LAN.

บทที่ 1 ETHERNET LAN

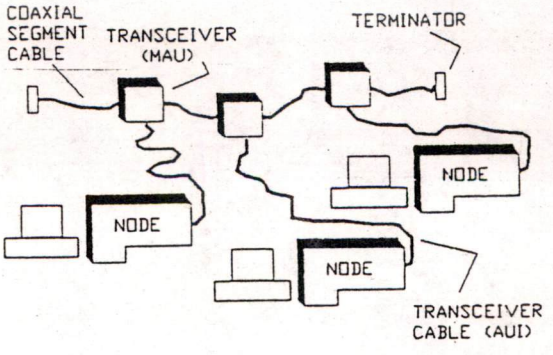
NETWORK คือเซตของการเชื่อมต่อเพื่อการสื่อสารซึ่งกันและกันระหว่างอุปกรณ์ต่างๆเช่น เทอร์มินัล , คอมพิวเตอร์,โทรศัพท์,พรินเตอร์ หรืออุปกรณ์อย่างอื่นที่สามารถสื่อสารข้อมูลและ จัดการข้อมูลได้ เราจะเรียก อุปกรณ์เหล่านี้ว่า Node

LOCAL AREA NETWORK (LAN) หมายถึง เนทเวอร์คที่มีขนาดอยู่ในระยะทางที่จำกัด (โดยทั่วไปจะไม่เกิน 10 km) ซึ่งมีอัตราการส่งข้อมูลตั้งแต่ 100 kb/s ถึง 100 Mb/s หรืออาจมากกว่านั้น LAN สามารถแบ่งออกได้เป็นหลายชนิด ตามรูปแบบของการเชื่อมต่อระหว่าง Node หรือที่เราเรียกว่า Network Topology

ETHERNET เป็น LAN ที่ใช้ topology แบบบัส ซึ่งพัฒนาโดย Xerox Palo Alto Reseach Center และเป็นไปตามมาตรฐาน IEEE 802.3 Type 10-Base-5 ซึ่งมาตรฐานนี้จะครอบคลุมชั้น data-link และ physical layers ของ OSI Model หรือชั้น Medium Access Control และ physical layers ของมาตรฐาน IEEE 802.3

มาตรฐาน IEEE 802.3 จะใช้สำหรับ LAN ที่มี access method แบบ CSMA/CD คำว่า Type 10-Base-5 ,10 หมายถึงมีอัตราการส่งข้อมูลบนเคเบิล 10 Mb/s , Base หมายถึงการส่งสัญญาณเป็นแบบ Baseband คือ สัญญาณแชนแนลเดียวจะใช้แบนด์วิธของสายทั้งหมด . 5 หมายถึง ความยาวของ cable segment ในหน่วย 100 เมตร ใน 1 segment จะมี Node ได้ไม่เกิน 100 Nodes และต้องอยู่ห่างกันอย่างน้อย 2.5 เมตร

Topology ของ Ethernet มีลักษณะดังรูป



โดยส่วนประกอบต่างๆจะมีหน้าที่และข้อกำหนดดังนี้

1.Node Board หรือ Ethernet Controller Board

เป็นส่วนอินเทอร์เฟสระหว่างNodeกับเนทเวอร์ค ให้สามารถติดต่อสื่อสารกันได้ มีหน้าที่ในส่วนของ MAC และ Physical layers ดังนี้

- ควบคุมการรับส่งข้อมูลให้เป็นไปอย่างถูกต้อง
- แปลงสัญญาณข้อมูลแบบ NRZ ให้เป็นแบบ MANCHESTER แล้วส่งไปบนเนทเวอร์คโดยผ่านทาง Transceiver
- รับสัญญาณข้อมูลแบบ MANCHESTER จากเนทเวอร์คผ่านทางTransceiverแล้วแปลงเป็นสัญญาณข้อมูลแบบ NRZ
- ตรวจสอบความผิดพลาดของข้อมูล

2.Tranceiver

จะรับข้อมูลจากเนทเวอร์คแล้วส่งไปให้ Node Board มีหน้าที่ในส่วนของ Physical layer ดังนี้

- ปรับระดับสัญญาณที่เข้ามาจากเนทเวอร์คให้มีความชัดเจนถูกต้อง
- รับสัญญาณแบบ DIFFERENTIAL MANCHESTER จาก Node board แล้วส่งออกไปแบบ open collector current driver
- ตรวจสอบการชนกันของสัญญาณบนเนทเวอร์ค
- มีวงจรป้องกันสัญญาณรบกวนที่เกิดขึ้นในสาย (Squelching circuit) ขณะที่ไม่มีการรับส่งข้อมูลบนสาย coaxial เพื่อป้องกันการรับสัญญาณผิดพลาด

3.Coaxial Segment Cable

เป็นสาย coaxial เส้นผ่านศูนย์กลาง 0.4 นิ้ว shield 2 ชั้น มีความยาวสูงสุดในแต่ละ segment ได้ 500 เมตรและใช้ Repeater เป็นตัวเชื่อมต่อระหว่าง segment

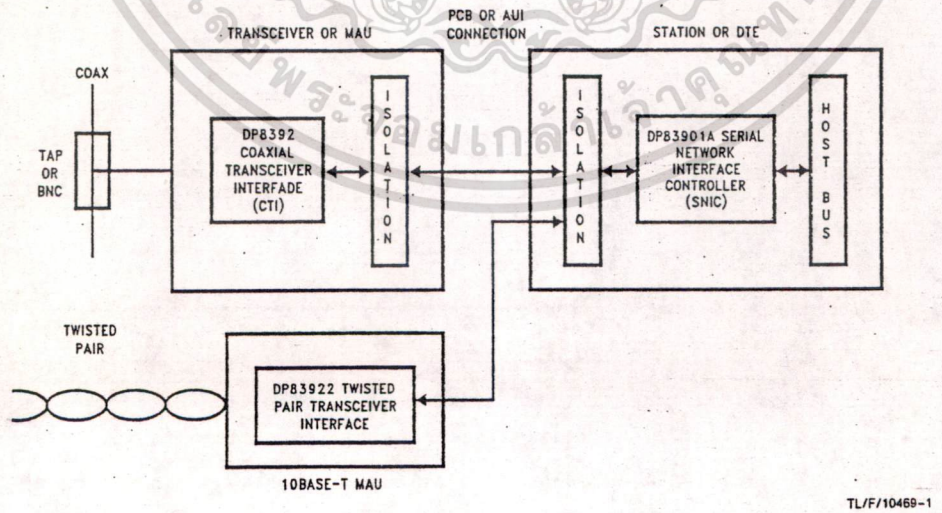
4.Transceiver Cable (Attachment Unit Interface : AUI)

เป็นสายเชื่อมต่อระหว่าง Node Board กับ Transceiver มีความยาวสูงสุดได้ 50 เมตร

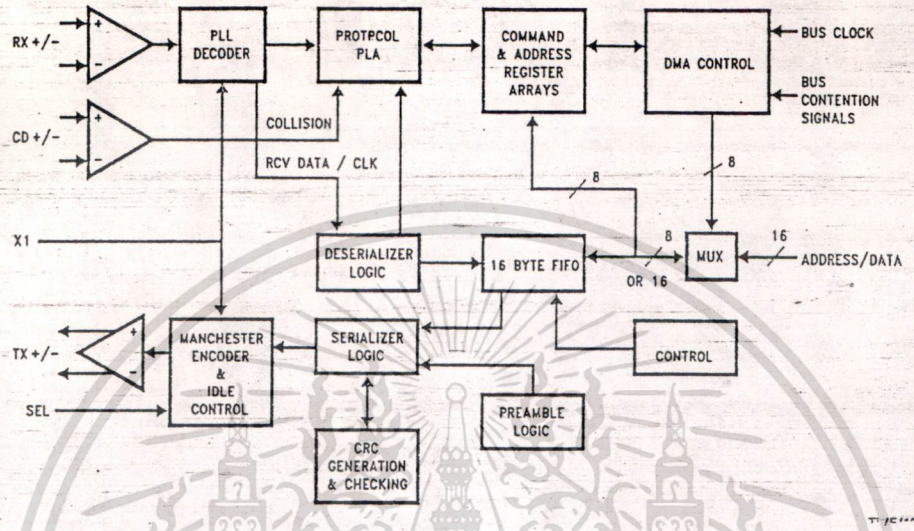
บทที่ 2. การทำงานของส่วนอินเทอร์เฟสระหว่างคอมพิวเตอร์กับ ETHERNET

ส่วนอินเทอร์เฟสที่ใช้กับคอมพิวเตอร์ใน ETHERNET จะมีอยู่หลายลักษณะตามบริษัทผู้ผลิตซึ่งจะมีลักษณะการทำงานที่คล้ายคลึงกันในส่วนที่เป็นมาตราฐานของ IEEE 802.3 และจะแตกต่างกันใน ส่วนที่ไม่ได้เป็นมาตราฐานของ IEEE 802.3 เช่นรูปแบบการเก็บข้อมูลที่รับเข้ามาใน บัฟเฟอร์ เป็นต้น

ฮาร์ดแวร์ที่ใช้เลือกศึกษาการทำงานของส่วนอินเทอร์เฟสครั้งนี้คือ DP83901 Serial Network Interface Controller และ DP8392CN Coaxial Transciever Interface ชิพทั้ง 2 ตัวนี้จะทำหน้าที่เป็นส่วนสำคัญของส่วนอินเทอร์เฟส ซึ่งเมื่อนำมาประกอบกันกับส่วนอื่นๆแล้วจะได้ System Diagram ดังรูปข้างล่าง



DP 83901 Serial Network Interface Controller



Block diagram ข้างบนสามารถแบ่งได้เป็น ๒ โมดูลหลักคือ

1. ENCODER/DECODER (ENDEC) MODULE
2. NIC (Network Interface Controller) MODULE

ENDEC MODULE

แบ่งออกเป็น ๓ ส่วนดังนี้

1) **MANCHESTER ENCODER AND DIFFERENTIAL DRIVER** รับสัญญาณแบบ NRZ จาก NIC MODULE แล้วเปลี่ยนเป็นแบบ MANCHESTER จากนั้นจึงส่งไปยัง TRANCEIVER ภายในนั้นจะมีหม้อแปลงเพื่อแยกกระแส DC ระหว่าง TRANCEIVER กับ NODE BOARD โดยทางด้าน secondary ของหม้อแปลงนี้สามารถจ่ายกระแสได้ระยะทาง ๕๐ เมตรบน Twisted pair AUI cable

2) **MANCHESTER DECODER** ทำหน้าที่รับข้อมูลซึ่งเป็นสัญญาณแบบ DIFFERENTIAL MANCHESTER จาก TRANCEIVER แล้วแปลงให้เป็นสัญญาณแบบ NRZ และสร้าง clock แล้วส่งไปยัง NIC MODULE นอกจากนั้นยังทำหน้าที่ตรวจสอบจุดสิ้นสุดของ Frame โดยตรวจจาก mid-bit transition

ในส่วนนี้ประกอบไปด้วยตัวรับสัญญาณ DIFFERENTIAL (Differential Receiver) กับ PLL (Phase-locked loop) ทั้งสองส่วนนี้ทำงานร่วมกันเพื่อใช้แยกสัญญาณ MANCHESTEที่เข้ามาให้เป็นสัญญาณ clock กับสัญญาณข้อมูลแบบ NRZ

3) **COLLISION TRANSLATOR** เมื่อ TRANCEIVER ตรวจพบ collosion signal บน network cable มันจะสร้างสัญญาณ sqaure wave ความถี่ 10 MHz เข้ามาที่ขา Differential input ของ NODE BOARD เมื่อขานี้ active SNIC จะยกเลิกการส่งในครั้งนั้นและเตรียมการส่งใหม่

NIC MODULE* หรือ *MEDIA ACCESS CONTROLLER

ประกอบไปด้วย 4 ส่วนคือ

1) **RECEIVE DESERIALIZER**

จะทำงานเมื่อมีสัญญาณอินพุท carrier sense เข้ามากระตุ้นเพื่อให้บิตที่เข้ามาถูกชิพเข้าไปยัง shift register โดย clock ที่ได้มาจากส่วน MANCHESTER DECODER และข้อมูลที่เข้ามาจะถูกส่งไปยังส่วน CRC GENERATOR/CHECKER ต่อไป ในส่วนนี้ประกอบไปด้วย

- Synch Detecter ทำหน้าที่ตรวจจับStartofFrameDelimiter(SFD)Frameที่เข้ามาเพื่อให้ทราบตำแหน่งของไบต์เริ่มต้นตั้งแต่ Destination Address Field
- 16-byte FIFO Buffer
- Receive Byte Count ทำหน้าที่นับจำนวนไบต์ที่เข้ามาโดยจะนับเพิ่มทีละหนึ่งเมื่อมีข้อมูล1ไบต์เพื่อให้เข้าไปใน 16-Byte FIFO

2) **TRANSMIT SERIALZER**

จะอ่านข้อมูลแบบขนานใน 16-Byte FIFO แล้วแปลงให้เป็นแบบอนุกรมเพื่อส่งออกไปผ่านส่วน CRC GENERATOR/CHECKER ซึ่งจะถูกกำหนดอัตราเร็วด้วย Transmit clock (20 MHz) ซึ่งสร้างจากภายใน

3) **CRC GENERATOR/CHECKER**

-ขณะส่งข้อมูล

CRC logic จะเพิ่ม Preamble field ,SFD bit และสร้าง CRC field จากข้อมูลแบบ อนุกรมที่ส่งผ่านตัวมันไปซึ่งรับมาจากส่วน TRANSMIT SERIALIZER CRC field จะถูกส่งตามหลังข้อมูล ไบต์สุดท้ายออกไป

-ขณะรับข้อมูล

CRC logic จะสร้าง CRC field จาก Frame ที่ถูกส่งผ่านเข้ามาแล้วนำค่าที่สร้างขึ้นไป เปรียบเทียบกับ CRC feild ของ Frame ที่รับเข้ามา ถ้าตรงกันก็แสดงว่า Frame ที่รับเข้ามาถูกต้อง ถ้าไม่ตรงก็จะทำการยกเลิกข้อมูลที่รับเข้ามาใน Frame นั้นทั้งหมด (สามารถกำหนดให้ รับหรือไม่รับข้อมูลได้ในกรณี CRC feild ไม่ตรงกัน)

4) **ADDRESS RECOGNITION LOGIC**

เปรียบเทียบ Destination Address Feild (6 ไบต์แรก ของFrame) ที่รับเข้ามากับค่า ใน Physical Address Registers (PAR), Multicast Address Registers (MAR) ถ้าไม่ตรงกัน ส่วน Protocol Control Logic (PCL) จะทำการยกเลิกการรับ Frame นั้น ในกรณีที่ทุกบิตของ Destination Address เป็น 1 ซึ่งหมายถึง Broadcast PCL ก็จะรับ Frame นั้นเข้ามา

PROTOCOL PLA

ทำหน้าที่จัดรูปแบบของ packet ระหว่างการส่งและนำPreamble feild ออกจาก packet

DMA AND BUFFER CONTROL LOGIC

BUFFER CONTROL LOGIC ใช้ในการควบคุม DMA ขนาด 16-bits 2 ตัว คือ local DMA กับ remote DMA

-local DMA ใช้รับส่งข้อมูลระหว่าง local memory กับ FIFO

-remote DMA ใช้รับส่งข้อมูลระหว่าง local memory กับ main memory

INTERNAL REGISTER

รีจิสเตอร์ภายในทุกตัวมีขนาด 8 บิต แบ่งเป็น 4 pages เลือกโดยบิต PS0 และ PS1ใน command register ขา RA0 - RA3 ใช้เลือกรีจิสเตอร์ในแต่ละเพจ

Page 0 Address Assignments (PS1 = 0, PS0 = 0)

RA0–RA3	RD	WR
00H	Command (CR)	Command (CR)
01H	Current Local DMA Address 0 (CLDA0)	Page Start Register (PSTART)
02H	Current Local DMA Address 1 (CLDA1)	Page Stop Register (PSTOP)
03H	Boundary Pointer (BNRY)	Boundary Pointer (BNRY)
04H	Transmit Status Register (TSR)	Transmit Page Start Address (TPSR)
05H	Number of Collisions Register (NCR)	Transmit Byte Count Register 0 (TBCR0)
06H	FIFO (FIFO)	Transmit Byte Count Register 1 (TBCR1)
07H	Interrupt Status Register (ISR)	Interrupt Status Register (ISR)
08H	Current Remote DMA Address 0 (CRDA0)	Remote Start Address Register 0 (RSAR0)
09H	Current Remote DMA Address 1 (CRDA1)	Remote Start Address Register 1 (RSAR1)
0AH	Reserved	Remote Byte Count Register 0 (RBCR0)
0BH	Reserved	Remote Byte Count Register 1 (RBCR1)
0CH	Receive Status Register (RSR)	Receive Configuration Register (RCR)
0DH	Tally Counter 0 (Frame Alignment Errors) (CNTR0)	Transmit Configuration Register (TCR)
0EH	Tally Counter 1 (CRC Errors) (CNTR1)	Data Configuration Register (DCR)
0FH	Tally Counter 2 Missed Packet Errors) (CNTR2)	Interrupt Mask Register (IMR)

Page 1 Address Assignments (PS1 = 0, PS0 = 1)

RA0–RA3	RD	WR
00H	Command (CR)	Command (CR)
01H	Physical Address Register 0 (PAR0)	Physical Address Register 0 (PAR0)
02H	Physical Address Register 1 (PAR1)	Physical Address Register 1 (PAR1)
03H	Physical Address Register 2 (PAR2)	Physical Address Register 2 (PAR2)
04H	Physical Address Register 3 (PAR3)	Physical Address Register 3 (PAR3)
05H	Physical Address Register 4 (PAR4)	Physical Address Register 4 (PAR4)
06H	Physical Address Register 5 (PAR5)	Physical Address Register 5 (PAR5)
07H	Current Page Register (CURR)	Current Page Register (CURR)
08H	Multicast Address Register 0 (MAR0)	Multicast Address Register 0 (MAR0)
09H	Multicast Address Register 1 (MAR1)	Multicast Address Register 1 (MAR1)
0AH	Multicast Address Register 2 (MAR2)	Multicast Address Register 2 (MAR2)
0BH	Multicast Address Register 3 (MAR3)	Multicast Address Register 3 (MAR3)
0CH	Multicast Address Register 4 (MAR4)	Multicast Address Register 4 (MAR4)
0DH	Multicast Address Register 5 (MAR5)	Multicast Address Register 5 (MAR5)
0EH	Multicast Address Register 6 (MAR6)	Multicast Address Register 6 (MAR6)
0FH	Multicast Address Register 7 (MAR7)	Multicast Address Register 7 (MAR7)

Page 2 Address Assignments (PS1 = 1, PS0 = 0)

RA0–RA3	RD	WR
00H	Command (CR)	Command (CR)
01H	Page Start Register (PSTART)	Current Local DMA Address 0 (CLDA0)
02H	Page Stop Register (PSTOP)	Current Local DMA Address 1 (CLDA1)
03H	Remote Next Packet Pointer	Remote Next Packet Pointer
04H	Transmit Page Start Address (TPSR)	Reserved
05H	Local Next Packet Pointer	Local Next Packet Pointer
06H	Address Counter (Upper)	Address Counter (Upper)
07H	Address Counter (Lower)	Address Counter (Lower)
08H	Reserved	Reserved
09H	Reserved	Reserved
0AH	Reserved	Reserved
0BH	Reserved	Reserved
0CH	Receive Configuration Register (RCR)	Reserved
0DH	Transmit Configuration Register (TCR)	Reserved
0EH	Data Configuration Register (DCR)	Reserved
0FH	Interrupt Mask Register (IMR)	Reserved

Note: Page 2 registers should only be accessed for diagnostic purposes. They should not be modified during normal operation.
Page 3 should never be modified.

COMMAND REGISTER (CR) 00H (READ/WRITE)

ใช้เริ่มต้นการส่ง , ควบคุมการทำงานของ remote DMA และเลือกเพจของรีจิสเตอร์ ในการสั่งคำสั่งมายัง CR จะต้องทำการเซตบิตต่อไปนี้คือ RD2-RD0, TxP บางคำสั่งสามารถทำงานในช่วงเวลาคาบเกี่ยวกันแต่ต้องเป็นไปตามข้อบังคับต่อไปนี้

- ถ้าคำสั่ง transmit มาซ้อนทับกับการทำงานของ remote DMA bit RD0-RD2จะยังคงเป็นไปตามใน Remote DMA command เมื่อมีการเซต TxP bit
- ถ้าการทำงานของ remote DMA มาซ้อนทับกับการส่ง บิต RD0-RD2 สามารถเขียนได้ตามต้องการ และ '0' อาจถูกเขียนลงใน TxP bit ซึ่งจะไม่มีผลใดๆ
- ถ้า remote DMA มีการทำงานอยู่ (Read/Write) จะต้องถูกทำให้เสร็จสิ้นหรือยกเลิกก่อนที่การทำงานอื่น จะเริ่มขึ้น Bit PS0,PS1,RD2,STP สามารถเซตได้ตลอดเวลา

7	6	5	4	3	2	1	0
PS1	PS0	RD2	RD1	RD0	TXP	STA	STP

STP - Stop : เป็นคำสั่งซอฟต์แวร์รีเซต เมื่อเซตเป็น 1 มีผลทำให้ controllerตัดขาดจากทำงานคือเข้าสู่สถานะรีเซต จะไม่มีการรับหรือส่ง packet การส่งหรือการรับที่กำลัง ทำอยู่จะถูกทำให้เสร็จสิ้นก่อนเข้าสู่สถานะรีเซต การออกจากสถานะนี้ทำได้โดยการเซตบิตนี้เป็น 0 และเซตบิต STA (Start) เป็น 1

STA - Start Transmit Packet : บิตนี้ใช้กระตุ้นให้SNICทำงานหลังจากเปิดเครื่องหรืออยู่ในสถานะรีเซ็ต TXPใช้เซตเพื่อเริ่มการส่งและจะรีเซตเองเมื่อการส่งเสร็จสิ้นหรือ ถูกยกเลิก บิตนี้ควร ถูกเซตหลังจากโปรแกรมค่าใน Transmit Byte Count กับ Transmit Page Start Register แล้ว

RD0-RD2 - remote DMA command : กำหนดการทำงานของ remote DMA ดังนี้

R2	RD1	RD0	
0	0	0	Not allowed
0	0	1	Remote read
0	1	0	Remote write (Note 2)
0	1	1	Send Packet
1	X	X	Abort/Complete Remote DMA (Note 1)

PS0-PS1 - Page Select : ใช้เลือกเพจของรีจิสเตอร์ดังนี้

PS0	PS1	
0	0	Register Page 0
0	1	Register Page 1
1	0	Register Page 2
1	1	Reserved

INTERRUPT STATUS REGISTER (ISR) 07H (READ/WRITE)

จะถูกอ่านโดย Host processor หรือ CPU เพื่อหาสาเหตุของการอินเทอร์รัพท์

7	6	5	4	3	2	1	0
RST	RDC	CNT	OVW	TXE	RXE	PTX	PRX

PRx - Packet Receive : แจ้งว่า packet ถูกรับเข้ามาโดยไม่มี error

PTx - Packet Transmitted : แจ้งว่า packet ถูกส่งออกไปโดยไม่มี error

RxE - Receive Error : แจ้งว่า packet ที่รับเข้ามามีข้อผิดพลาดซึ่งสามารถเกิดขึ้นได้
มากกว่า 1 ลักษณะดังนี้

- CRC Error
- Frame Alignment Error
- FIFO Overrun
- Missed Packet

TxE - Transmit Error : เซตเป็น 1 เมื่อ packet ที่ส่งออกไปเกิด error ซึ่งมีดังนี้

- Exceesive Collision
- FIFO Underrun

OVW - Overwrite Warning : เซตเป็น 1 เมื่อ receive buffer ring ถูกใช้จนหมด

CNT - Counter Overflow : เซ็ตเมื่อ MSB ของ NETWORK TALLY COUNTER หนึ่งหรือมาก
กว่าหนึ่งถูกเซ็ต

RDC - Remote DMA Complete : เซตเป็น 1เมื่อการทำงานของ remote DMA เสร็จสิ้นลง

RST - Reset Status : จะเซตเป็น 1 เมื่อ

- SNIC เข้าสู่สถานะรีเซตและเคลียร์เมื่อมีคำสั่ง START ไปยังCR
- เกิด Receive Buffer Ring Overflow และเคลียร์เมื่อนำ packet ตั้งแต่ 1 ขึ้น
ไป ออกจากบัฟเฟอร์

บิตนี้ไม่สร้างอินเทอร์รัพท์จะเป็นเพียงแค่ตัวบอกสถานะเท่านั้น

INTERRUPT MASK REGISTER (IMR) 0FH (WRITE)

ใช้สำหรับการกำหนดให้เกิด interrupt แต่ละบิตจะตรงกับ Interrupt Status Register (ISR) ถ้าบิตใดถูกเซ็ต สัญญาณ interrupt จะเกิดถ้าค่าในบิตของ ISR ที่ตรงกันถูกเซ็ต และ ถ้าบิตใดๆใน IMR เซ็ตเป็น 0 ก็จะไม่มี interrupt เกิดขึ้น ถึงแม้ว่าบิตใน ISR เซ็ต

7	6	5	4	3	2	1	0
-	RDCE	CNTE	OVWE	TXEE	RXEE	PTXE	PRXE

PRXE - Packet Received Interrupt Enable :

0 : Interrupt Disable

1 : Enable Interrupt เมื่อได้รับ packet

PTXE - Packet Transmitted Interrupt Enable :

0 : Interrupt Disable

1 : Enable Interrupt เมื่อ packet ถูกส่งออก

RXEE - Receive Error Interrupt Enable :

0 : Interrupt Disable

1 : Enable Interrupt เมื่อ packet ที่ได้รับมาเกิด error

TXEE - Transmit Error Interrupt :

0 : Interrupt Disable

1 : Enable Interrupt เมื่อ packet ที่ส่งออกเกิด error

OVWE - Overwrite Warning Interrupt Enable :

0 : Interrupt Disable

1 : Enable Interrupt เมื่อมีที่ไม่พอสำหรับข้อมูลที่รับมา

CNTE - Counter Overflow Interrupt Enable

0 : Interrupt Disable

1 : Enable Interrupt เมื่อ MSB ของ Network Statistics counter หนึ่ง หรือ มากกว่าหนึ่ง counter ถูกเซ็ต

RDCE - DMA Complete Interrupt Enable :

0 : Interrupt Disable

1 : Enable Interrupt เมื่อ Remote DMA ทำการย้ายข้อมูลเสร็จเรียบร้อย

DATA CONFIGURATION REGISTER (DCR) 0EH (WRITE)

รีจิสเตอร์นี้จะถูกโปรแกรมเพื่อกำหนด

- การอินเทอร์เฟสระหว่าง SNIC กับ memory ว่าจะให้เป็นแบบ 8,16 บิต
- เลือกลำดับของไบต์ถ้าใช้เป็นแบบ 16 บิตและกำหนดรูปแบบของ FIFO DCR จะต้องถูกโปรแกรมก่อนการโหลดค่าให้กับ Remote Byte Count Register

7	6	5	4	3	2	1	0
-	FT1	FT0	ARM	LS	LAS	BOS	WTS

WTS - Word Transfer Select : กำหนดจำนวนข้อมูลในการเคลื่อนย้ายแต่ละครั้งของทั้ง local และ remote DMA

- 0 = DMA เคลื่อนย้ายข้อมูลทีละ Byte
- 1 = DMA เคลื่อนย้ายข้อมูลทีละ Word

BOS - Byte Order Select : มีผลเฉพาะเมื่อเลือกรูปแบบการเคลื่อนย้ายข้อมูลเป็น Word

- 0 : ไบต์สูงอยู่บน AD15-AD8 และไบต์ต่ำอยู่บน AD7-AD0
- 1 : ไบต์สูงอยู่บน AD7-AD0 และไบต์ต่ำอยู่บน AD15-AD8

LAS - Long Address Select

- 0 : Dual 16-bit DMA mode
- 1 : Single 32-bit DMA mode

LS - Loopback Select

- 0 : เลือกการทำงานในโหมด loopback บิต D1 และ D2 ของ TCR จะต้องถูกโปรแกรม สำหรับการทำงาน loopback
- 1 : การทำงานปกติ

ARM - Auto Initialize Remote

- 0 : คำสั่ง SEND ไม่ทำงาน การนำ packet ออกจากบัฟเฟอร์จะเป็นไปตามขั้นตอนที่โปรแกรมไว้
- 1 : คำสั่ง SEND ทำงาน Remote DMA จะถูกตั้งค่าเริ่มต้นเองเพื่อนำ packet ออกจาก buffer ring

** 680x0 processor ไม่สามารถใช้คำสั่ง SEND ได้

FT0,FT1 - FIFO Thershold Select : ถ้ากำหนดจำนวน byte ที่เข้าไปใน FIFO ในขณะรับถ้า FIFO มีข้อมูลเข้ามาถึงระดับ byte ที่ตั้งไว้นี้ SNIC จะทำการส่งสัญญาณ BREQ เพื่อใช้ DMA ย้ายข้อมูลจาก FIFO ไปไว้ใน local memory

Receive Thresholds

FT1	FT0	Word Wide	Byte Wide
0	0	1 Word	2 Bytes
0	1	2 Words	4 Bytes
1	0	3 Words	8 Bytes
1	1	4 Words	12 bytes

TRANSMIT CONFIGURATION REGIGTER (TCR) 0D (WRITE)

ค่าใน TCR จะทำให้ส่วน transmitter section ทำงานระหว่างการส่ง packet บนเนทเวอร์ค

7	6	5	4	3	2	1	0
-	-	-	OFST	ATD	LB1	LB0	CRC

CRC - Inhibit CRC

- 0 : CRC ถูกเติมโดย Transmitter
- 1 : ไม่เติม CRC

ใน mode loopback CRC สามารถถูก enable หรือ disable เพื่อทำการคำนวณCRC ได้

LB0, LB1 - Encode Loopback Control : เป็นบิตที่ใช้ตั้งชนิดของ loopback ให้สังเกตว่า loopback ใน mode 2 บิต D3 ของ DCR ต้องถูกเซ็ตเป็น 0

	LB1	LB0	
mode 0	0	0	Normal Operation
mode 1	0	1	Internal NIC Module Loopback
mode 2	1	0	Internal ENDEC Module Loopback
mode 3	1	1	External Loopback

ATD - Auto Transmit Disable : บิตนี้จะทำให้ station อื่นสามารถ disable SNIC ของเครื่องส่งโดยการส่ง multicast packet เฉพาะออกมา เครื่องส่งสามารถถูก enableใหม่โดยการ reset บิตนี้หรือโดยรับ multicast packet เฉพาะอันที่สอง

1 : การรับ multicast address ซึ่งจะใส่ลงในบิตที่ 62 จะ disable เครื่องส่ง การรับ multicast address ซึ่งจะใส่ลงในบิตที่ 63 จะ enable เครื่องส่ง

OFST - Collision Offset Enable : เปลี่ยน backoff อัลกอลิทึ่มเพื่อให้เกิด priority ระหว่าง node

0 : Backoff Logic ใช้ backoff อัลกอลิทึ่มแบบธรรมดา

1 : ใช้ backoff อัลกอลิทึ่มเป็น 0-{2ยกกำลัง[min(3+n,10)]} slot time สำหรับการชนกัน 3 ครั้งแรก จากนั้นจึงใช้อัลกอลิทึ่มแบบธรรมดา

TRANSMIT STATUS REGISTER (TSR) 04H (READ)

บอกสภาพที่เกิดขึ้นบนตัวกลางที่ใช้ส่งข้อมูลระหว่างการส่ง packet ค่าในรีจิสเตอร์จะถูกเคลียร์เมื่อการส่งครั้งต่อไปเริ่มขึ้นใหม่ ก่อนการส่งแต่ละครั้งควรมีการอ่านค่าในรีจิสเตอร์เพื่อตรวจสอบสถานะของการส่ง

7	6	5	4	3	2	1	0
OWC	CDH	FU	CRS	ABT	COL	-	PTX

PTX - Packet Transmitted : แสดงว่าการส่งไม่มี error (ไม่เกิด collision หรือ FIFO Underflow) (ABT=0,FU=0)

COL - Transmit Collided : แสดงว่าการส่งเกิดการชนกันอย่างน้อยที่สุด 1 ครั้งกับ station อื่น จำนวนของการชนจะใส่ใน NCR

ABT - Transmit Aborted : แสดงว่า SNIC ยกเลิกการส่งเพราะเกิด collision เกินจำนวนที่กำหนด คือ 16 ครั้ง

CRS - Carrier Sense Lost : บิตนี้จะถูกเซ็ตเมื่อ carrier เสียหายในระหว่างการส่ง การส่งจะไม่ถูกยกเลิก

FU - FIFO Underrun : ถ้า SNIC ไม่สามารถเข้าถึง bus ได้ก่อนที่ FIFO ว่าง บิตนี้จะถูกเซ็ต การส่ง packet จะถูกยกเลิก

CDH - CD Heartbeat : transceiver ส่งสัญญาณแสดงการเกิด collision ล้มเหลว บิตนี้จะถูกเซ็ต CD-Heartbeat signal ต้องเริ่มต้นส่งระหว่าง ช่องว่างแรกระหว่าง frame (6.4 microsec.) หลังจากการส่ง frame ในการเกิด collision ในบางครั้งบิตนี้จะถูกเซ็ต แม้ว่า transcieve ไม่ได้เซ็ตไว้เพื่อให้ทำ CD Heartbeat Test

OWC - Out Of Window Collision : แสดงการเกิด collision ขึ้นหลังจาก slot time (51.2 microsec.)

RECEIVE CONFIGURATION REGISTER_ (RCR) 0CH (WRITE)

กำหนดการทำงานของ SNIC ขณะรับ packet และกำหนดชนิดของ packet ที่จะรับ

7	6	5	4	3	2	1	0
-	-	MON	PRO	AM	AB	AR	SEP

SEP - Save Error Packet

0 : packet ที่รับมาเกิด error จะถูกทิ้งไป

1 : packet ที่รับมาเกิด error จะรับไว้ (ข้อมูลเกด CRC error และ Frame Alignment error)

AR - Accept Runt Packet : บิตนี้จะตัวบอกให้รับ packet ที่น้อยกว่า 64 byte แต่ต้องน้อยสุดไม่ต่ำกว่า 8 byte

0 : packet น้อยกว่า 64 byte ไม่รับ

1 : packet น้อยกว่า 64 byte รับ

AB - Accept Broadcast

0 : packet ที่เป็น broadcast จะถูกทิ้งไป

1 : packet ที่เป็น broadcast จะรับไว้

AM - Accept Multicast : Enable เครื่องรับให้ทำการรับ packet ที่เป็น multicast

0 : packet ที่เป็น Multicast Destination Address จะไม่ถูกตรวจ

1 : packet ที่เป็น Multicast Destination Address จะถูกตรวจ

PRO - Promicuous Physical

0 : packet ที่มี Physical Address ของ node ตรงกับ address ของ station ซึ่งถูกเก็บใน PAR0-PAR5

1 : ทุก packet ที่ถูกส่งแบบ Physical Address จะถูกรับหมด

MON - Moniter Mode : ใช้เซ็ตให้รับ packet เข้ามาพร้อมกับตรวจ address และ CRC แต่ไม่เก็บ packet ลงใน buffer

0 : เก็บ packet ลงใน buffer

1 : ตรวจ address, CRC, Frame Alignment แต่ไม่เก็บ packet ลงใน buffer

RECEIVE STATUS REGISTER (RSR) 0CH (READ)

บันทึกสถานะของ packet ที่รับ,ข้อมูลเกี่ยวกับชนิดของ error และแอดเดรสที่รับเข้ามา ค่าในรีจิสเตอร์นี้จะถูกเขียนลงใน header packet ซึ่งเก็บอยู่ใน buffer memory โดย DMA เมื่อ packet ที่รับเข้ามาถูกต้องแต่ถ้ามีการกำหนดให้รับ packet ที่มี error ค่าในรีจิสเตอร์นี้ก็จะถูกเขียนลงใน header เช่นกัน ค่าในรีจิสเตอร์จะถูกเคลียร์เมื่อ packet ต่อไปถูกรับเข้ามา

7	6	5	4	3	2	1	0
DFR	DIS	PHY	MPA	FO	FAE	CRC	PRX

PRX - Packet Receive Intact : แสดงว่า packet ที่รับเข้ามาไม่มี error

CRC - CRC Error : แสดงว่า packet ที่รับมามี CRC error และบิตนี้จะถูกเซ็ตด้วยเมื่อมี Frame Alignment Error และ Tally Counter (CNTER1) จะเพิ่มขึ้น

FAE - Frame Alignment Error : แสดงว่า packet ที่รับมาไม่มีจุดสิ้นสุด และค่า CRC ไม่ตรงกันที่ byte สุดท้าย และทำให้ Tally Counter (CNTR0) เพิ่มขึ้นด้วย

FO - FIFO Overrun : บิตนี้จะถูกเซ็ตเมื่อไม่สามารถใช้ FIFO ได้เนื่องมาจากการรับเกิด overflow และการรับจะถูกยกเลิก

MPA - Missed Packet : เซ็ตเมื่อ SNIC ไม่ได้รับ packet เนื่องมาจากว่าไม่สามารถใช้ receive buffer ได้ หรือถ้า controller อยู่ใน mode monitor และไม่สามารถที่จะพักข้อมูลชั่วคราวก่อนที่จะเข้า memory ได้ และทำให้ Tally Counter (CNTR2) เพิ่มขึ้น

PHY - Physical / Multicast Address : แสดงว่า packet ที่รับเข้ามาจะเป็นแบบ physical หรือ multicast address

0 : Physical Address match

1 : Multicast / Physical Address match

DIS - Receiver Disabled : เซ็ตเมื่อเครื่องรับถูก disable โดยการเข้าสู่ monitor mode และจะรีเซ็ตเมื่อถูก enable ใหม่เมื่อออกจาก monitor mode

DFR - Deferring : เซ็ตเมื่อมีสัญญาณ Carrier Sense หรือ Collision Signal เข้ามาที่ ENDEC module หรือเซ็ตเมื่อกำลังทำ jabbering procedure

FAE	CRC	Type of Error
0	0	No Error (Good CRC and < 6 Dribble Bits)
0	1	CRC Error
1	0	Illegal, will not occur
1	1	Frame Alignment Error and CRC Error

TRANSMIT/RECEIVE PACKET ENCAPSULATION/DECAPSULATION

ในมาตราฐาน IEEE 802.3 1 packet จะประกอบด้วย preamble, start of frame (SFD), destination address, source address, length data และ Frame Check Sequence (FCS) รูปแบบของ packet แสดงดังรูป

preamble	SFP	destination	source	length	data	FCS
62b	2b	6b	6b	2b	46b-1500b	4b

packet จะถูกเข้ารหัสหรือถอดรหัสแบบ Manchester โดย ENDEC module และก็จะทำการย้าย packet แบบอนุกรมไปยัง NIC module โดยใช้ข้อมูลแบบ NRZ ด้วย clock ทุกๆ field จะมีความยางคงที่ยกเว้น data field SNIC จะเป็นตัวใส่ preamble, SFD และ FCS ในตอนส่ง ส่วนในตอนรับ preamble และ SFD จะถูกตัดออก ส่วน CRC จะถูกส่งผ่านไปยัง buffer memory

PREAMBLE AND START OF FRAME DELIMITER (SFD)

ในการเข้ารหัสแบบ Manchester สัญญาณ preamble จะเป็นสัญญาณที่เปลี่ยนจาก 0 เป็น 1 และ 1 เป็น 0 สลับกันไป เพื่อใช้ในการ synchronization สัญญาณ preamble จะมีขนาด 62 บิต และเมื่อถึงเครื่องรับจะถูกกำจัดออกโดย NIC module ข้อมูลที่ต้องการสำหรับเครื่องรับจะเริ่มหลัง จาก SFD (ซึ่งเป็นสัญญาณ 1 ติดกัน 2 บิต) และจะนำ preamble ที่รับมานั้นทำการสร้าง clock เพื่อใช้ในการแปลง Manchesterกลับไปเป็น NRZ

DESTINATION ADDRESS

เป็น field ที่ใช้เก็บ address ของ network ที่เราต้องการส่งไป มีรูปแบบของ address 3 แบบ

- physical address : เป็น address ของ node ที่ต้องการส่งไปเพียง node เดียวเท่านั้น ถูกกำหนดโดย MSB มีค่าเป็น 0 ซึ่งค่า address นี้จะนำไปเปรียบเทียบกับ physical address register ของ network นั้น ซึ่งถ้าตรงกัน SNIC จะรับข้อมูลนั้น
- multicast address : กำหนดโดย MSB มีค่าเป็น 1
- broadcast address : กำหนดโดยทุกบิตมีค่าเป็น 1 หมด ซึ่ง packet นี้จะถูกรับโดยทุก network

SOURCE ADDRESS

เป็น physical address หรือ address ของ network ที่ทำการส่ง packet นั้น ซึ่งใน field นี้ ไม่สามารถเขียนเป็น multicast หรือ broadcast ได้ และ field นี้จะถูกเก็บใน buffer memory

LENGTH FIELD

มีขนาด 2 byte ใช้แสดงจำนวน byte ของข้อมูลจริงๆ

DATA FIELD

มีขนาด 46-1500 byte ถ้าข้อมูลมีค่ามากกว่า 1500 byte ก็จะทำการแบ่งออกเป็นอีก packet หนึ่ง และถ้าข้อมูลมีค่าน้อยกว่า 46 byte จะทำการเติม pad เข้าไปเพื่อให้ครบ 46 byte และจำนวน byte ของข้อมูลใน LENGTH FIELD จะเป็นจำนวนของข้อมูลจริงๆ

FCS FIELD

มีขนาด 32 บิต ไว้ใส่ค่า CRC ซึ่งถูกคำนวณและใส่ลงไปในขั้นตอนการส่ง เพื่อทำการตรวจสอบ error ในตอนรับข้อมูล ซึ่งถ้าค่า CRC ไม่ถูกต้องก็จะไม่รับข้อมูลนั้น

DIRECT MEMORY ACCESS CONTROL (DMA)

Local DMA จะย้ายข้อมูลระหว่าง FIFO กับ buffer memory ในระหว่างการส่งข้อมูลจะถูกย้ายออกจาก memory ไป FIFO และส่งออก และถ้าเกิด collision packet จะถูกส่งใหม่ โดย processor จะไม่ยุ่งเกี่ยว แต่ถ้ายังเกิด collision จนถึงครั้งที่ 15 การส่งก็จะหยุดลง ในระหว่างการรับ packet จะถูกย้ายจาก FIFO ไปยัง receive buffer ring

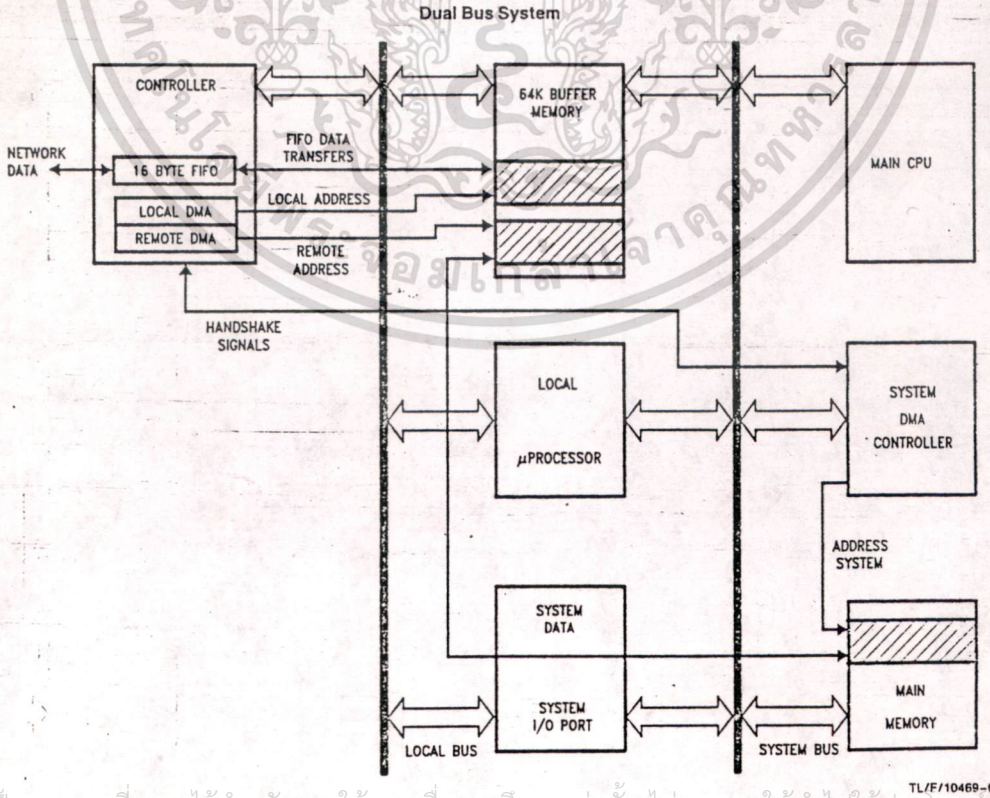
Remote DMA จะทำการย้ายข้อมูลระหว่าง buffer memory กับ main memory DMA ทั้ง 2อันสามารถรวมกันใช้เป็น 32 bit address ด้วย 8 หรือ 16 บิต data

DUAL DMA CONFIGURATION

รูปแบบที่ยกตัวอย่างดังรูปจะใช้ DMA ทั้ง 2 อันแยกกัน โดยการทำงานของเนทเวอร์คจะถูกแยกโดย local bus ซึ่งการทำงานทั้ง 2 อัน จะทำหน้าที่ดังที่กล่าวมาแล้ว การทำงานของ local DMA และ remote DMA สามารถถูกแทรกแซงได้โดย SNIC

SINGLE CHANNEL DMA OPERATION

DMA ทั้ง 2 อันสามารถรวมกันใช้เป็น 32-bit address DMA ได้ 16 บิตบนจะถูกใช้เพื่อชี้ page ของ memory 64-k byte (32-k word)

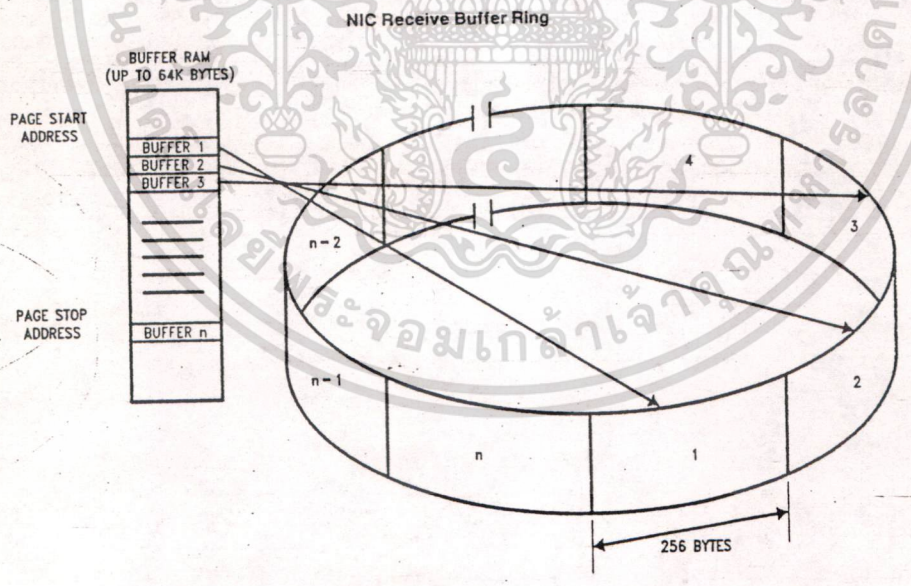


PACKET RECEPTION

ข้อมูลที่รับมาจาก FIFO โดย local DMA จะถูกนำมาเก็บที่บัฟเฟอร์ ซึ่งมีโครงสร้างแบบวงแหวน (Buffer Ring Structure) ใน local memory เรียกว่า receive buffer ring โครงสร้างในลักษณะนี้จะประกอบไปด้วยบัฟเฟอร์ย่อยๆขนาด 256 ไบต์เรียงติดต่อกัน ตำแหน่งเริ่มต้นและสิ้นสุดของ memory ที่ใช้เป็นบัฟเฟอร์จะถูกโปรแกรมใน Page Start Register และ Page Stop Register เมื่อ DMA อ้างอิงมาถึงตำแหน่ง page stop จะรีเซตไปยังตำแหน่ง page start ใหม่ ขนาดของ receive buffer ring จะถูกสงวนไว้ใน local memory 64 KB

การเก็บ packet ที่รับเข้ามาลงในบัฟเฟอร์จะถูกจัดการโดย Buffer Management Logic ซึ่งมีหน้าที่ 3 อย่างคือ

- Link บัฟเฟอร์ย่อยหลายๆอันเข้าด้วยกันเมื่อ packet มีขนาดมากกว่า 1 บัฟเฟอร์ย่อย
- นำบัฟเฟอร์กลับมาใช้ใหม่ถ้าบัฟเฟอร์นั้นเก็บ packet ที่ถูก reject
- นำบัฟเฟอร์กลับมาใช้ใหม่ถ้าบัฟเฟอร์นั้นเก็บ packet ที่ CPU ได้อ่านไปแล้ว



INITIALIZATION OF THE BUFFER RING

เราใช้รีจิสเตอร์ 4 ตัวในการควบคุมการทำงานของ buffer ring คือ

1. Page Start Register (PSTART)

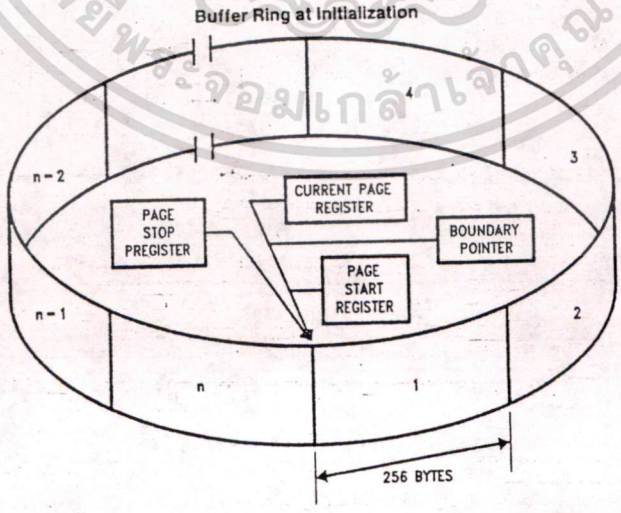
2. Page stop Register (PSTOP)

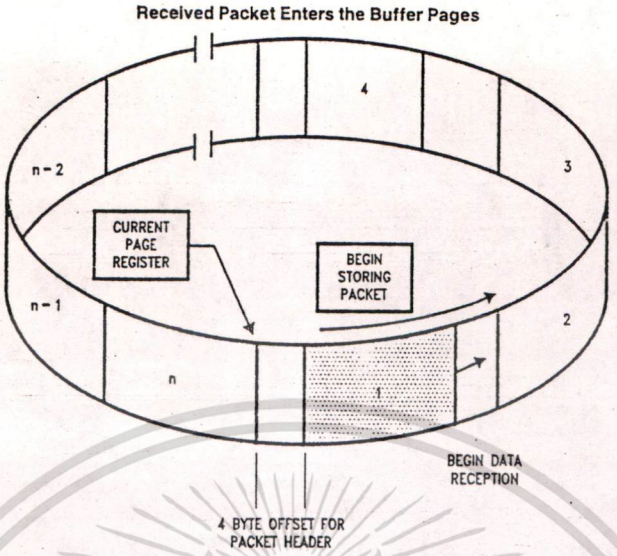
3. Current Page Register

4. Boundary Pointer Register

PSTART และ PSTOP - ใช้เป็นตัวกำหนดขอบเขตของ memory ส่วนที่เป็น buffer ring ดังที่ได้กล่าวมาแล้ว

Current Page Register - ชี้ไปยังบัฟเฟอร์แรกที่ใช้เก็บ packet DMA จะใช้ค่านี้เพื่อ หาตำแหน่งสำหรับเขียนสถานะของ packet ลงในบัฟเฟอร์และใช้ตั้งค่า DMA ใหม่ในกรณีที่เกิด frame error (Runt packet error,Frame alignment error,CRC error) Boundary Pointer Register - ชี้ไปยัง packet แรกใน buffer ring ที่ยังไม่ได้ถูก อ่านไป เมื่อ local DMA ชี้มาถึงค่าในรีจิสเตอร์นี้การรับจะถูกยกเลิกเพื่อป้องกันการเขียนทับข้อมูลที่ยังไม่ได้ถูกอ่านไปและยังใช้สำหรับ remote DMA ในการย้าย packet ไปยัง main memory เมื่อข้อมูลในบัฟเฟอร์ตำแหน่งนั้นถูกอ่านไปแล้ว ค่าในรีจิสเตอร์จะชี้ไปยัง packet ถัดไป





BEGINNING OF RECEPTION

เมื่อ packet แรกเข้ามา SNIC จะเก็บ packet ไว้ในตำแหน่งที่ชี้โดย Current Page Register เนื้อที่ 4 ไบต์แรกของบัฟเฟอร์แรกจะใช้สำหรับเก็บ status information ของ packet นั้น

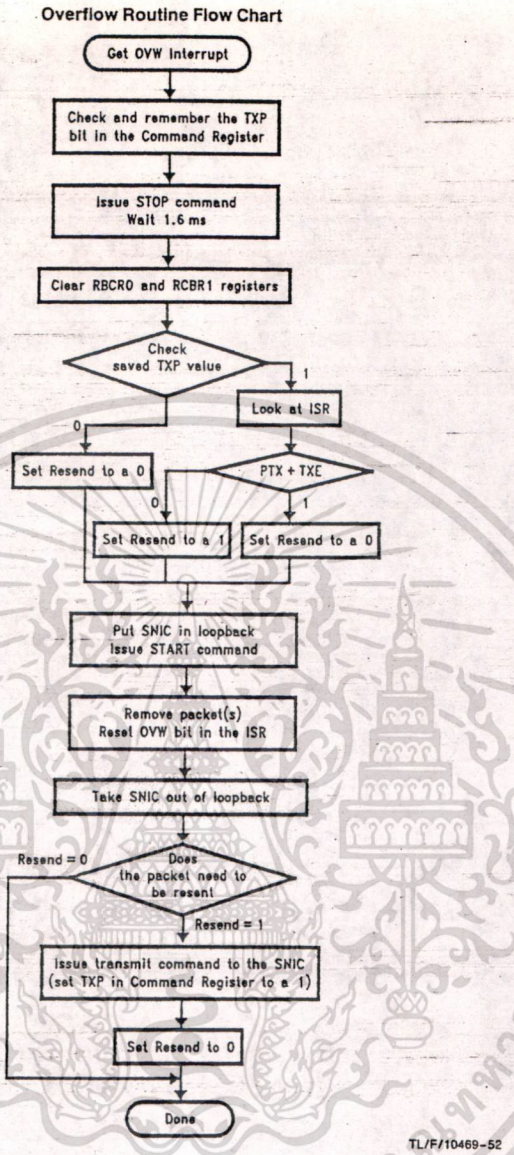
ถ้าความยาวของ packet ที่เข้ามาเกินขนาดของบัฟเฟอร์แรกคือ 256 ไบต์ DMA จะทำ forward link เพื่อเก็บส่วนที่เหลือของ packet ไว้ในบัฟเฟอร์ถัดไปซึ่งจะใช้ไม่เกิน 6 อัน โดยต้องเรียงติดต่อกันไปไม่มีการข้ามหรือสลับ การเชื่อมบัฟเฟอร์เข้าด้วยกัน Buffer Management Logic ต้องทำการเปรียบเทียบ 2 ครั้งดังนี้

1.เปรียบเทียบ DMA address ของบัฟเฟอร์ชุดถัดไปกับค่าใน PSTOP ถ้ามีค่าเท่ากันก็จะใส่ค่าใน DMA ด้วยค่า PSTART

2.เปรียบเทียบ DMA address ของบัฟเฟอร์ชุดถัดไปกับค่าใน Boundary Pointer Register ถ้ามีค่าเท่ากันการรับจะถูกยกเลิกทั้งนี้เพื่อป้องกันการเขียนทับข้อมูลในส่วนที่ยังไม่ได้ถูกอ่านไป

BUFFER RING OVERFLOW

เมื่อบัฟเฟอร์ใน memory ถูกใช้หมดและ DMA มาถึงค่าใน Boundary Pointer Register การรับ packet ในขณะนั้นจะถูกยกเลิกโดย SNIC แต่ packet ที่รับเข้ามาแล้วจะยังคงอยู่ใน memory SNIC จะทำการอ่าน interrupt เพื่อบอกเข้ามาว่าเกิด overflow แล้ว ดังนั้นจึงต้องมี Interrupt Handle เพื่อจัดการกับ overflow ของ buffer ring ซึ่งมี flowchart ดังรูปหน้าถัดไป



(Overflow Routine Flow Chart)

ถ้า buffer ring เต็มและค่า DMA ไปถึงค่า boundary การรับก็จะถูกยกเลิกมิฉะนั้นเมื่อรับเข้ามาก็จะทับข้อมูลเดิมซึ่งยังไม่ได้ถูกอ่านออกไป เมื่อสิ้นสุดการรับ packet SNIC จะดูว่าpacket ที่รับเข้ามาถูกรับไว้แล้วหรือถูกทิ้งไป จากนั้นก็จะไปทำรูทีนซึ่งมีหน้าที่ในการเก็บ buffer header หรือรูทีนอื่นๆเพื่อทำการนำบัฟเฟอร์ที่ใช้แล้วกลับมาใช้ใหม่

ENABLING THE SNIC ON AN ACTIVE NETWORK

หลังจากหยุดการทำงานของ SNIC เนื่องจากเกิด overflow ก็ต้องมีการเซ็ตให้ SNIC ทำงานใหม่ซึ่งมีขั้นตอนดังนี้

1. โปรแกรมค่า CR สำหรับ page 0 โดยเขียนค่า 21H ลงใน CR
2. เตรียม DCR (Data Configuration Register) เริ่มต้นสำหรับการทำงาน
3. เคลียร์ Remote Byte Count Register (DBCR0,DBCR1)
4. เตรียม RCR (Receive Configuration Register) เริ่มต้นสำหรับการทำงาน
5. ทำให้ SNIC ทำงานแบบ loopback ใน mode 1 หรือ 2 (โดยการเขียน 02H หรือ 04H ลงใน TCR)
6. เตรียม receive buffer ring โดยการทำ BNDRY (Boundary Pointer), PSTART (Page Start) และ PSTOP (Page Stop) ให้พร้อมสำหรับเริ่มต้นทำงาน
7. เคลียร์ ISR โดยการเขียน 0FFH ลงใน ISR
8. เตรียม IMR ให้พร้อมสำหรับเริ่มต้นทำงาน
9. โปรแกรม CR สำหรับ page 1 โดยการเขียน 61H ลงไป
   - เตรียม Physical Address Register (PAR0-PAR5)
   - เตรียม Mulicast Address Register (MAR0-MAR7)
   - เตรียม CURRent pointer
10. สั่งให้ SNIC อยู่ใน start mode โดยเขียน 22H ลงใน CR local DMA ยังไม่ active เพราะว่า SNIC ยังอยู่ในการทำงานแบบ loopback
11. เตรียม Transmit Configuration ให้พร้อม

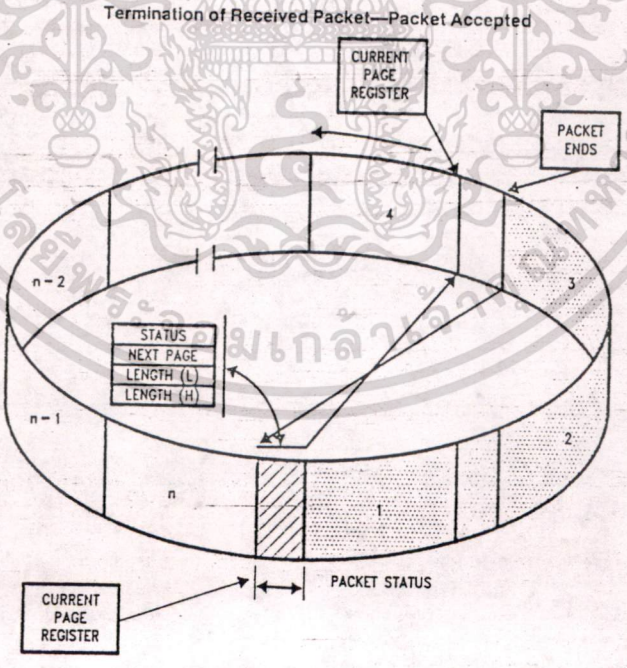
เมื่อทำตามขั้นตอนเสร็จ SNIC ก็จะพร้อมที่จะส่งหรือรับข้อมูลแล้ว

END OF PACKET OPERATIONS

หลังจาก SNIC ได้รับ packet สุดท้ายแล้ว มันจะตัดสินใจว่าข้อมูลที่รับมาจะรับเอาไว้หรือทิ้งไป มันจะไปทำ routine เพื่อเก็บ buffer header หรือมิฉะนั้นก็จะไปทำ routine เพื่อเก็บข้อมูลเอาไว้

SUCCESSFUL RECEPTION

ถ้าการรับประสบผลสำเร็จ DMA จะทำการเก็บค่าของ receive status ลงไปที่บัฟเฟอร์แรกของบัฟเฟอร์ที่ใช้เก็บข้อมูลนั้น (ซึ่งถูกชี้โดย Current Page Register) ซึ่ง statusจะเป็นตัวชี้บอกบัฟเฟอร์ที่เก็บแพ็คเกตถัดไป (ไบต์ที่ 4) และจำนวนไบต์ของข้อมูล เป็นที่น่าสังเกตว่าจำนวนไบต์ที่ยังคงมีอยู่ในบัฟเฟอร์สุดท้ายจะถูกทิ้งไป และการรับข้อมูลชุดต่อไปก็จะเริ่มขึ้นที่บัฟเฟอร์ถัดไปซึ่ง Current Page Registerจะชี้ไปยังบัฟเฟอร์ที่ว่างถัดไปใน buffer ring

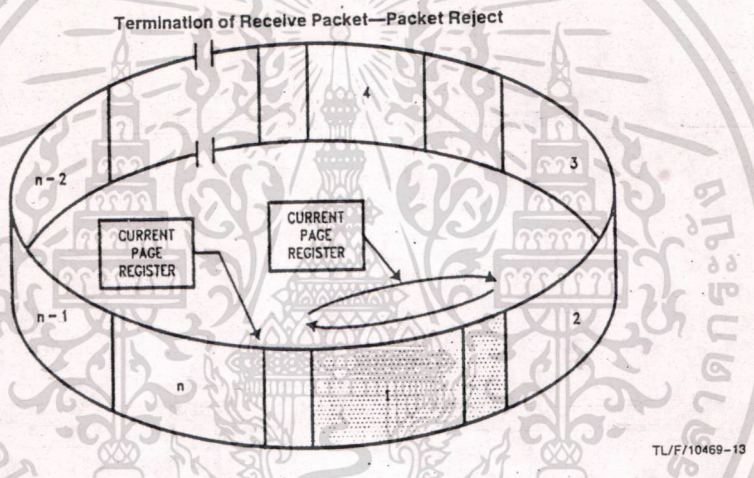


BUFFER RECOVERY FOR REJECTED PACKETS

ถ้า packet ที่รับเข้ามาเกิด Frame error ซึ่งมี 3 อย่างดังนี้

- Runt packet คือ packet ที่มีขนาดความยาวต่ำกว่าความยาวต่ำสุดที่กำหนดไว้ (64 ไบต์)
- Frame Alignment Error คือ packet มีขนาดความยาวไม่ตรงกับที่กำหนดไว้ใน length field
- CRC Error

packet นั้นจะถูกทิ้งไป (ถ้าไม่ได้โปรแกรมให้รับ packet ที่เกิด error) Buffer Management Logic จะทำการรีเซตค่าใน DMA ไปยังตำแหน่งบัฟเฟอร์แรกที่ใช้เก็บ packet นั้น (ชี้โดย Current Page Register) เพื่อนำบัฟเฟอร์ซึ่งเก็บ packet ที่เกิด error กลับมาใช้งานใหม่



การนำ packet ออกจากบัฟเฟอร์

packet จะถูกนำออกจากบัฟเฟอร์โดย remote DMA เราต้องสั่งคำสั่ง SEND ไปเพื่อให้ remote DMA ทำการย้าย packet ออกจากบัฟเฟอร์โดยอัตโนมัติซึ่งทราบตำแหน่งเริ่มต้นที่จะย้ายได้จาก ค่าใน Boundary Pointer Register และโหลดค่าใน Remote Byte Count Register ด้วยค่าใน ReceiveStatusเมื่อการเคลื่อนย้ายเสร็จสิ้นลง SNIC ก็จะทำการเปลี่ยนค่าใน Boundary Pointer Register ไปยัง ตำแหน่งถัดไปและนำบัฟเฟอร์ชุดนั้นกลับมาใช้งานใหม่

ขั้นตอนต่อไปนี้ใช้จัดการพอยน์เตอร์ต่างๆสำหรับ buffer ring

1.เมื่อตอนเริ่มต้นให้สร้างตัวแปรชื่อ Next_Ptr สำหรับชี้ไปยัง packet ถัดไปที่จะถูกอ่านโดยทุกๆครั้งที่ remote DMA เริ่มอ่านข้อมูลจากบัฟเฟอร์ ค่าของตัวแปร Next_Ptr จะถูกโหลดไปยัง RSAR0 และ RSAR1 (Remote Start Address Register)

2.เมื่อตอนเริ่มแรกให้ตั้งค่าตัวแปรต่างๆดังนี้

BNDRY = PSTRAT;

CURR = PSTART + 1;

Next_Ptr = PSTART + 1;

3.เมื่อ packet ถูก remote DMA ย้ายออกไปจากบัฟเฟอร์แล้ว Next Page Pointer ที่อยู่ในไบต์ที่ 2 ของ packet header (header อยู่ในบัฟเฟอร์แรก) ซึ่งชี้ไปยังบัฟเฟอร์ของ packet อันถัดไปจะถูกใช้ เพื่อกำหนดค่า BNDRY และ Next_Ptr ดังนี้

BNDRY = Next Page Pointer - 1;

Next_Ptr = Next Page Pointer;

IF BNDRY = PSTART THEN BNDRY = PSTOP - 1;

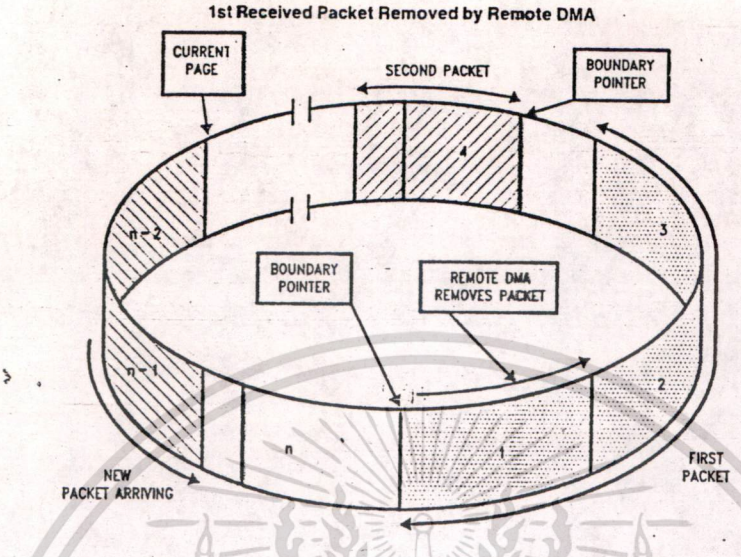
รูปแบบของการเก็บ packet จากบัฟเฟอร์ลงใน memory

รูปแบบจะขึ้นอยู่กับค่าใน Data Configuration Register (DCR)

AD15	AD8	AD7	ADO
Next Packet Pointer		Receive Status	
Receive Byte Count 1		Receive Byte Count O	
Byte 2		Byte 1	

AD15	AD8	AD7	ADO
Next Packet Pointer		Receive Status	
Receive Byte Count O		Receive Byte Count 1	
Byte 1		Byte 2	

Receive Status
Next Packet Pointer
Receive Byte Count O
Receive Byte Count 1
Byte O
Byte 1



PACKET TRANSMISSION

Local DMA มีหน้าที่ในการส่ง packet ออกไปจาก local memory โดยมีรีจิสเตอร์ 3 ตัวควบคุมการทำงานของ local DMA คือ

- Transmit Page Start Register (TPSR) บอกตำแหน่งเริ่มต้นของข้อมูลที่ต้องการส่ง
- Transmit Byte Count Registers (TBCR0, TBSR1) บอกจำนวนไบต์ที่จะส่ง

ข้อมูลที่ต้องการส่งที่เก็บใน local memory จะอยู่ในรูปแบบดังนี้

General Transmit Packet Format

Transmit	Destination Address	6 byte
Byte	Source Address	6 byte
Count	Type Length	2 byte
TBCRO,1	Data	>=46 byte

TRANSMIT PACKET ASSEMBLY

SNIC จะต้องส่ง packet ที่รวมกันอยู่ด้วยกันดังรูปข้างบน ค่าใน Transmit Byte Count จะเป็นความยาวของ destination address, source address, length field และข้อมูลรวมกันอยู่ เมื่อต้องการส่งข้อมูลที่มีขนาดน้อยกว่า 46 ไบต์ packet จะต้องถูกใส่จำนวนไบต์ (เรียกว่า pad) ให้ครบ 64 ไบต์ ซึ่งการนำ pad bytes ออกจาก packet หรือใส่ใน packet เป็นหน้าที่ของโปรแกรมที่อยู่ในระดับสูงก่อนทำการส่ง รีจิสเตอร์ทั้ง 3 ตัวจะต้องถูกตั้งค่าก่อนแล้วจึงโปรแกรมค่าบิต TxP ใน CR โดยการเซตเป็น 1 TSR (บอกสถานะของเนทเวอร์คขณะส่ง)จะถูกเคลียร์ จากนั้น SNIC จะเริ่ม fetch ข้อมูล memory ยกเว้นขณะทำการรับ packet อยู่

เงื่อนไขสำหรับการเริ่มต้นการส่งข้อมูล

1. ช่องว่างระหว่าง frame จะต้องกินเวลา 6.4 microsec.
2. ใน FIFO จะต้องถูกใส่ byte ลงไป อย่างต่ำที่สุด 1 byte (เพื่อเป็นการแสดงว่าการเคลื่อนย้ายข้อมูลจะถูกเริ่มขึ้น)
3. ถ้า collision ถูกตรวจพบก่อนการส่ง การเคลื่อนย้ายข้อมูลก็จะหยุดชั่วคราว

ซึ่งในระบบที่กล่าวมานี้ SNIC จะทำการอ่าน byte กลุ่มแรกก่อนที่เวลา 6.4 microsec. จะหมดลงในระหว่างเวลาที่ SNIC ทำการส่ง preamble ช่วงเวลานั้นสามารถถูกใช้เพื่อทำการ load ค่า FIFO ได้

COLLISION RECOVERY

ในระหว่างการส่ง buffer management logic จะเป็นวงจรที่ควบคุมตรวจสอบการเกิด collision ถ้าเกิดการ collision buffer management จะทำการรีเซ็ต FIFO และทำการเก็บค่าลงใน Transmit DMA Pointer ใหม่อีกครั้ง บิต COL ใน TSR จะถูกเซ็ตและ NCR (Number of Collision Register) จะถูกเพิ่มขึ้น ถ้าการส่งใหม่เกิด collision อีก ก็จะทำการส่งใหม่ ซึ่งถ้า collision ทำให้เกิดการส่งใหม่จนถึงครั้งที่ 15 การส่งก็จะถูกยกเลิกและบิต ABT ใน TSR จะถูกเซ็ต

TRANSMIT PACKET ASSEMBLY FORMAT

รูปหน้าถัดไปแสดงรูปแบบต่างๆของ packet เพื่อส่งออก ซึ่งรูปแบบต่างๆเหล่านี้สามารถเลือกได้โดยใส่ค่าลงใน Data Configuration Register

D15 D8	D7 D0
Desstination Address 1	Destination Address 0
Destination Address 3	Destination Address 2
Destination Address 5	Destination Address 4
Source Address 1	Source Address 0
Source Address 3	Source Address 2
Source Address 5	Source Address 4
Typu/Length 1	Type/Length o
Data 1	Data 0

D15 D8	D7 D0
Destination Address 0	Destination Address 1
Destination Address 2	Destination Address 3
Destination Address 4	Destination Address 5
Source Address 0	Source Address 1
Source Address 2	Source Address 3
Source Address 4	Source Address 5
Type/Length 0	Type/Length 2
Data 0	Data 1

D7 D0
Destination Address 0
Destination Address 1
Destination Address 2
Destination Address 3
Destination Address 4
Destination Address 5
Source Address 0
Source Address 1
Source Address 2
Source Address 3
Source Address 4
Source Address 5

REMOTE DMA

Remote DMA จะทำหน้าที่ทั้ง 2 อย่างคือ รวบรวมและเตรียม packet สำหรับการส่งและเคลื่อนย้าย packet ที่รับเข้ามาจาก receive buffer ring มันอาจจะถูกใช้ในงานปกติ คือใช้ในการเคลื่อนย้าย block ของข้อมูลหรือคำสั่งระหว่าง host memory และ local buffer memory ซึ่งจะมี 3 โหมดในการทำงานคือ

- Remote Write
- Remote Read
- Send Packet

Remote DMA ถูกควบคุมโดย register pair 2 คู่ คือ Remote Start Address (RSAR0, RSAR1) และ Remote Byte Count (RBCR0,RBCR1)

Full handshake logic จะถูกใช้ในการย้ายข้อมูลระหว่าง local buffer memory และ bidirectional I/O port

REMOTE WRITE

mode นี้จะใช้ในการย้ายข้อมูลจาก host memory ไปยัง local buffer memory Remote DMA จะอ่านข้อมูลจาก I/O port และจะเขียนลงไปใน local buffer memory เรียงตามลำดับโดย address เริ่มต้นถูกเก็บอยู่ใน Remote Start Address ในการเคลื่อนย้ายแต่ละครั้ง Remote Start Address จะเพิ่มขึ้นและ Remote Byte Count จะลดลง และการเคลื่อนย้ายจะสิ้นสุดลงเมื่อ Remote Byte Count ลดลงถึง 0

REMOTE READ

mode นี้จะใช้ในการย้ายข้อมูลจาก local buffer memory ไปยัง host memory และ DMA จะเริ่มอ่านข้อมูลที่ address ซึ่งเก็บอยู่ใน Remote Start Address และจะเขียนข้อมูลลงใน I/O port ในการย้ายข้อมูลแต่ละครั้ง Remote Start Address จะเพิ่มขึ้น และ Remote Byte Count จะลดลง และจะสิ้นสุดการย้ายข้อมูลเมื่อ Remote Byte Count ลดลงถึง 0

SEND PACKET COMMAND

เป็นคำสั่งให้ Remote DMA ทำการย้าย packet ออกจาก buffer โดยอัตโนมัติ ซึ่งจะ load ค่าเริ่มต้นของ Remote DMA ด้วยค่าของ BNDRY และ load ค่าใน RBCR0,1 ด้วยค่าใน Receive Byte Count ในbuffer header ของ packet

INITIALIZATION PROCEDURES

SNIC ต้องถูกทำการเตรียมพร้อมก่อนที่จะทำการส่ง หรือการรับ packet จาก network สัญญาณรีเซ็ต จะถูกใส่ที่ขา reset ของ SNIC ซึ่งการเตรียมพร้อม SNIC จะทำการเซ็ตหรือรีเซ็ตบิตต่อไปนี้

Register	Reset Bits	Set Bits
CR	TXP, STA	RD2, STP
ISR		RST
IMR	All Bits	
DCR		LAS
TCR	LB1, LB0	

SNIC จะอยู่ในสภาพรีเซ็ตจนกระทั่งคำสั่ง start ถูกส่งออกมา ซึ่งการทำอย่างนี้จะเป็นข้อยืนยันได้ว่า จะไม่มี packet ถูกส่งออกไปหรือรับเข้ามาก่อนที่เราจะทำการส่งหรือรับ หลังจาก SNIC ถูกทำการเริ่ม ต้นแล้ว และบิต STP ของ CR ถูกรีเซ็ต ก็จะทำให้เริ่มส่งหรือรับข้อมูลได้

INITIALIZATION SEQUENCE

1. โปรแกรม CR สำหรับ page 0 ด้วยค่า 21H ลงใน CR
2. ตั้งค่า DCR
3. เคลียร์ RBCR0,RBCR1
4. ตั้งค่า RCR
5. ให้ SNIC ทำงานแบบ loopback mode1 หรือ 2 (โดยให้ TCR มีค่า 02H หรือ 04H)
6. ตั้งค่า receive buffer ring โดยทำให้ Boundary Pointer (BNDRY), Page Start (PSTART) และ Page Stop (PSTOP) ให้พร้อมสำหรับการเริ่มทำงาน
7. เคลียร์ ISR โดยเขียน 0FFH ลงใน ISR
8. มาร์กบิตใน IMR
9. โปรแกรม CR สำหรับ page 1 (โดยเขียนค่า 61H ลงใน CR)
   - เซต Physical Address Register (PAR0-PAR5)
   - เซต Multicast Address Register (MAR0-MAR5)
   - เซต CURRent pointer
10. ทำให้ SNIC อยู่ใน start mode (โดยใส่ค่า 22H ลงใน CR)
11. เตรียม Transmit Configuration

เมื่อทำตามขั้นตอนเสร็จ SNIC ก็พร้อมที่จะทำการส่งและรับข้อมูล ก่อนที่จะทำการรับข้อมูลผู้ใช้ควรจะทำการกำหนดตำแหน่ง receive buffer ring ก่อนโดยการเขียน ลงใน page start และ page stop register รวมทั้ง Boundary และ Current Page Register ต้องถูกใส่ค่าเริ่มต้นจาก page start register ด้วย ซึ่ง register เหล่านี้จะเปลี่ยนแปลงเองใน ระหว่างการรับข้อมูล

LOOPBACK DIAGNOSTICS

SNIC จะมีรูปแบบการทำงานแบบ loopback ได้ 3 รูปแบบ ผู้ใช้สามารถกำหนดรูปแบบของ loopback โดยผ่าน loopback โดยผ่าน deserializer บน controller หรือผ่าน ENDEC module หรือ Coax Transceiver บน DP83901A SNIC (ถ้าผ่านอันไหนก็คือตรวจสอบอันนั้น)

เพราะว่ารูปแบบการทำงานของ SNIC เป็นแบบ half duplex ดังนั้น loopback testing จะเป็น mode พิเศษสำหรับการทำงานด้วยข้อจำกัดดังต่อไปนี้

ข้อจำกัดระหว่างการทำ loopback

FIFO จะถูกแบ่งเป็น 2 ส่วน ส่วนหนึ่งสำหรับการส่ง และอีกส่วนสำหรับการรับ เพียง 8 บิตเท่านั้นที่สามารถถูกอ่านจาก memory ดังนั้นการตรวจสอบ 2 ครั้งจึงเป็นสิ่งจำเป็นสำหรับระบบ 16 บิต เพื่อที่จะตรวจสอบความถูกต้องแน่นอนของเส้นทางของข้อมูลทั้งหมด

Destination Address	= (6 Bytes) Station Physical Address
Source Address	
length	2 Bytes
Data	= 46 to 1500 Bytes
CRC	Appended by SNIC if CRC = "0" in TCR

เมื่ออยู่ใน mode word-wide และ Byte Order Select ถูกเซ็ต loopback packet จะต้องถูกทำให้อยู่ในตำแหน่งของ 7 byte ดังรูปข้างล่าง (loopback จะเพียงแต่ทำงานกับการย้าย byte wide เท่านั้น)

เมื่ออยู่ใน mode word-wide และ byte order select เป็น low จะทำให้รูปแบบของ loopback packet เป็นดังรูป

LS Byte (AD8-15)	MS Byte (AD0-7)
	Destination
	Source
	Length
	Data
	CRC

WTS="1" BOS="1" (DCR Bits)

MS Byte (AD8-15)	LS Byte (AD0-7)
Destination	
Source	
Length	
Data	
CRC	

WTS="1" BOS="0" (DCR Bits)

เพื่อทำการเริ่มต้นการทำงานของ loopback ขั้นแรก ผู้ใช้ต้องกำหนดรูปแบบของ loopback packet ก่อนและหลังจากนั้นก็เลือกชนิดของ loopback โดยใช้ บิต LB0,LB1 ใน Transmit Configuration Register Transmit Configuration Register ต้องถูกเซ็ต เพื่อให้ enable หรือ disable CRC Generation ในระหว่างการส่งด้วย ต่อมาผู้ใช้ก็ให้คำสั่งส่งข้อมูลเพื่อทำการส่งข้อมูลออก ในระหว่าง loopback ตัวรับจะทำการ check address ว่าตรงกันหรือเปล่า และถ้าบิต CRC ใน TCR ถูกเซ็ต ตัวรับก็จะ check CRC ด้วย 8 byte สุดท้ายของ loopback จะถูกเก็บใน buffer และสามารถ อ่านออกมาได้โดยใช้ FIFO read port

LOOPBACK MODE

- mode 1 : loopback ผ่าน NIC module (LB1 = 0, LB0 = 1) ถ้า modeนี้ถูกใช้จะทำให้ serializer ของ NIC module ถูกต่อกับ deserializer
- mode 2 : loopback ผ่าน ENDEC module (LS1 =1, LB0 =0) ถ้า loopback ถูกกำหนดโดยผ่าน SNI จะทำให้ SNIC ทำการควบคุม ซึ่งจะบังคับให้ ENDEC module ถูกตรวจสอบโดย loopback ทุกสัญญาณ

- mode 3 : loopback กับ Coax (LB1 =1,LB0 =1) ใน mode นี้ packet จะถูกส่งผ่านไปยัง Coax เพื่อทำการตรวจสอบเส้นทางการส่งและรับ

READING THE LOOPBACK PACKET

8 byte สุดท้ายของ packet ที่รับเข้ามาโดยการผ่านของ FIFO ตามลำดับ 8 ลำดับ FIFO pointer จะเพิ่มค่าขึ้นหลังจากขอบขาขึ้นของสัญญาณ read strobe ของ CPU ซึ่งมันจะกินเวลา 4 clock cycle และถ้า pointer เปลี่ยนค่าไม่ทันในขณะที่ CPU ทำการอ่านค่า FIFO register อีกครั้ง SNIC ก็จะเข้าสู่สภาวะ wait states

ALIGNMENT OF THE RECEIVED PACKET IN THE FIFO

การรับของ packet ใน FIFO จะเริ่มต้นที่ตำแหน่ง 0 และหลังจากที่ pointer ไปถึงตำแหน่งสุดท้ายแล้วพอย์นเตอร์จะวนกลับขึ้นไปที่ขอดของ FIFO และเขียนทับข้อมูลที่รับมาก่อนแล้ว ขบวนการนี้จะต่อเนื่องกันไปจนกระทั่งได้รับ byte สุดท้าย และต่อมา SNIC จะเพิ่มจำนวน byte ที่รับมา (byte count) ลงไปที่ 2 location ถัดไป ซึ่งข้อมูลที่บรรจุอยู่ใน upper byte count จะถูก copy ลงใน location ของ FIFO ถัดไปด้วย จำนวนของ byte ซึ่งถูกใช้ใน loopback packet จะเป็นตัวที่ใช้ตัดสิน alignment ของ packet ใน FIFO alignment สำหรับ packet 64 bytes แสดงดังรูปข้างล่าง

FIFO Location	FIFO Contents	
0	Lower ByteCount	First Byte Read
1	Upper Byte Count	Second Byte Read
2	Upper Byte Count	
3	Last Byte	
4	CRC1	
5	CRC2	
6	CRC3	
7	CRC4	Last Byte Read

สำหรับ alignment ใน FIFO ต่อไปนี้ ความยาวของ packet จะเป็น (Nx8) + 5 byte ให้สังเกตว่าถ้าบิต CRC ใน TCR ถูกเซ็ต CRC จะไม่ถูกใส่เพิ่มลงไปโดยเครื่องส่ง ถ้า CRC ถูกเพิ่มเติมโดยเครื่อง ส่ง 4 byte แรกคือ byte N-3 ถึง byte N จะตรงกับ CRC

FIFO Location	FIFO Contents	
0	Byte N-4	First Byte Read
1	Byte N-3 (CRC1)	Second Byte Read
2	Byte N-2 (CRC2)	
3	Byte N-1 (CRC3)	
4	Byte N (CRC4)	
5	Lower Byte Count	
6	Upper Byte Count	Last Byte Read
7	Upper Byte Count	

LOOPBACK TEST

ความสามารถของ loopback นั้นไว้ใช้ในการตรวจสอบการทำงานต่างๆของ DP83901 SNIC ก่อนที่จะทำการส่งหรือรับข้อมูล ซึ่งการตรวจสอบสามารถตรวจสอบสิ่งต่อไปนี้

1. ตรวจสอบความถูกต้องของเส้นทางเดินของข้อมูล ข้อมูลที่รับเข้ามาจะถูกตรวจสอบกับข้อมูลที่ส่งออกไป
2. ตรวจสอบการทำงานของ CRC logic ของเครื่องส่งและเครื่องรับ
3. ตรวจสอบความสามารถของ Address Recognition Logic มีการทำงานดังนี้
   - การทำงานถ้า address ตรงกัน
   - ไม่รับ packet ถ้า address ไม่ตรงกัน

LOOPBACK OPERATION IN SNIC

loopback เป็นรูปแบบอย่างหนึ่งสำหรับการตรวจสอบคุณภาพของการส่งซึ่งใช้ได้เพียงครึ่งหนึ่งของ FIFO ซึ่งสิ่งนี้เป็นข้อจำกัดอย่างหนึ่งของการตรวจสอบแบบ loopback เมื่อทำการเลือก mode ของ loopback โดยใช้ TCR แล้ว FIFO จะถูกแบ่งแยกออก packet ควรจะถูกกำหนดรูปแบบการเก็บลงใน memory โดย ใช้ register TPSR และ TBCR0,TBCR1 และเมื่อให้คำสั่งส่งออกไป การทำงานต่อไปนี้จะเกิดขึ้น

TRANSMITTER ACTION

1. ข้อมูลจะถูกย้ายจาก memory โดย DMA จนกระทั่ง FIFO เต็ม ในการย้ายแต่ละครั้ง TBCR0, TBCR1 จะลดลง

2. SNIC จะส่งสัญญาณ preamble 56 บิต และตามด้วย 1 synch pattern 8 บิต

3. ข้อมูลถูกย้ายจาก FIFO ไปยัง serializer

4. ถ้าบิต CRC ใน TCR เป็น 1 SNIC จะไม่คำนวณ CRC และ byte สุดท้ายที่ส่งไปจะเป็น byte สุด ท้ายใน FIFO และถ้า CRC เป็น 0 SNIC จะทำการคำนวณ CRC และก็จะส่ง 4 byte ของ CRC ต่อท้ายไป

5. ในตอนสุดท้ายของการส่ง บิต PTX ใน ISR จะถูกเซ็ต

RECEIVER ACTION

1. คอยรับ synch และกำจัด preamble ทั้งหมดออก

2. เก็บ packet ลงใน FIFO และในแต่ละครั้งที่รับ byte เข้ามาก็จะเพิ่ม receive byte count

3. ถ้าบิต CRC=1 เครื่องรับจะทำการตรวจสอบ CRC error ของ packet ที่รับเข้ามา ถ้าบิต CRC=0 เครื่องไม่ต้องตรวจสอบ CRC error บิตที่แสดง CRC error จะอยู่ใน RSR

4. ในตอนสุดท้ายของการรับ receive byte count จะถูกเขียนลงใน FIFO และ receive status register จะถูกเขียนตามค่าของสถานะต่างๆ บิต PRX ใน RSR จะถูกเซ็ต แม้ว่า address จะไม่ตรงกัน ถ้าเกิด CRC error packet จะต้องมี address ตรงกัน เพื่อให้ CRC error bit ใน RSR ถูกเซ็ต

CRC AND ADD RECOMBINATION

การตรวจสอบ 3 แบบที่จะกล่าวถึง เป็นการตรวจสอบ address recognition logic และ CRC การตรวจสอบเหล่านี้จะใช้ loopback ภายในเพื่อว่าจะได้ไม่รบกวน network ในส่วนที่ไม่ต้องการจะ test ซึ่งการกรวจสอบเหล่านี้ยังต้องการ การผลิต CRC ทาง software ด้วย

address recognition logic ไม่สามารถถูกตรวจสอบได้โดยตรง บิต CRC และ FAC ใน RSR จะถูก set ถ้า address ใน packet ตรงกับ address ของ filter ซึ่งการแสดง error ของ address recognition logic ก็คือ มันจะไม่ set บิต CRC และ FAC ในขณะที่เราคิดว่ามันต้อง set ซึ่ง error นี้ จะเป็นผลทำให้ packet ถูกทิ้งไป

ลำดับของ packet ต่อไปนี้จะตรวจสอบ address recognition logic DCR set เป็น 40H และ TCR set เป็น 03H พร้อมทั้งผลิต CRC ทาง software

Packet Contents			Results
Test	Address	CRC	RSR
Test A	Matching	Good	O1
Test B	Matching	Bad	O2
Test C	Non-Matching	Bad	O1

DP 8392CN Coaxial Transceiver Interface

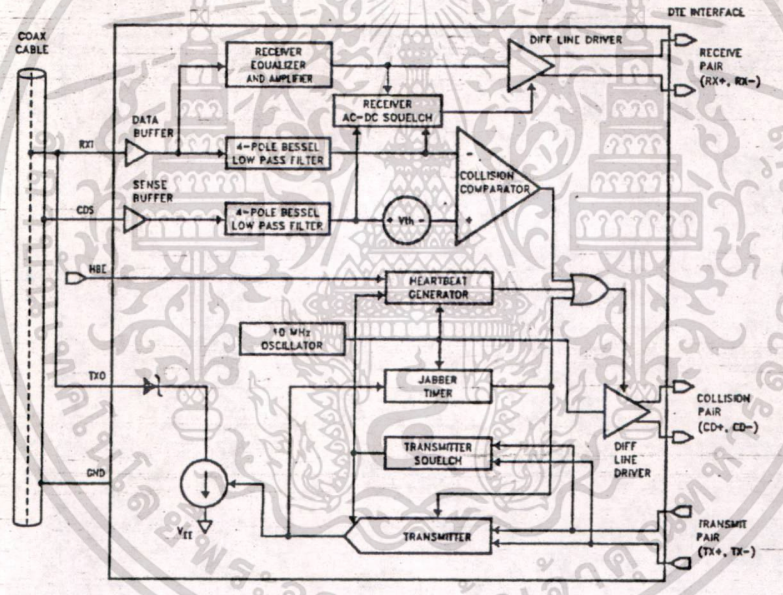


FIGURE 1. DP8392C Block Diagram

CTI ประกอบด้วยส่วนหลัก 4 ส่วน คือ

1. Receiver - รับข้อมูลจากสาย Coaxial แล้วส่งไปยัง DTE
2. Transmitter - รับข้อมูลจาก DTE แล้วส่งไปบนสาย Coaxial
3. Collision Detect Circuitry - บอก DTE ว่ามีการชนเกิดขึ้นบนเนทเวอร์ค
4. Jabber timer เป็นตัวจำกัดเวลาของการส่ง

1. ส่วนรับ แยกออกได้เป็น 4 ส่วนดังนี้

1. บัฟเฟอร์ สัญญาณในสายจะมองเห็นเป็น High input Impedance และ Low input Impedence ทำให้ลดโหลดของกระแสและลดการ Reflection บนสาย
2. Equrlizer เป็นวงจรกรองความถี่สูง
3. 4 Pole Bessel LPF จะกรองระดับเฉลี่ยของกระแส DC บนสาย Coaxial เพื่อนำไปใช้สำหรับวงจร squelch และวงจรตรวจจับการชนกัน
4. Reciever Squelch cct. ป้องกันไม่ให้ noise บนสายเข้าวงจรขณะที่ไม่มีการส่งสัญญาณบนสาย ตอนเริ่มเมื่อ packet เข้ามา ส่วนรับจะเริ่มทำงานเมื่อระดับDCของสัญญาณที่เข้ามาต่ำกว่าค่าที่ตั้งไว้ เมื่อตอนสิ้นสุด packet AC Timing Circuit ซึ่งคอยดูว่าถ้าระดับสัญญาณ High Impedance นานกว่า 200 ns ส่วนรับจะหยุดทำงานทันที

Differential Line Driver จะส่งสัญญาณระดับ Ecb ไปยัง DTE โดยมี rise time และ fall time เป็น 3 ns ถ้าไม่มีการรับข้อมูลสัญญาณที่ออกจากขานี้ จะเป็นศูนย์

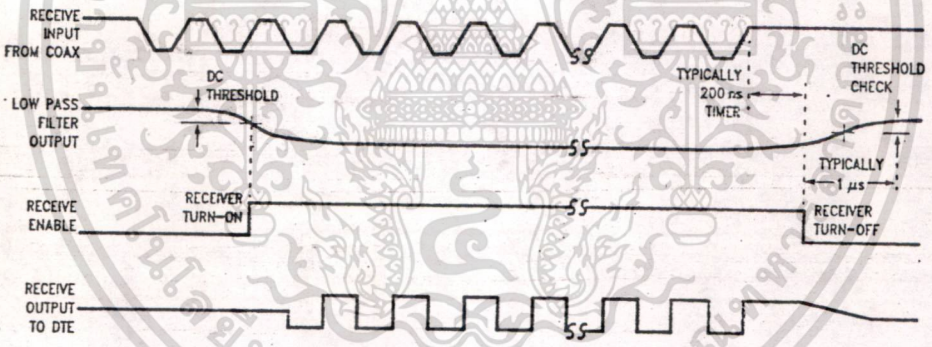


FIGURE 2. Receiver Timing

2. Transmitter ประกอบไปด้วย differntial input และ open collecter output current driver

สัญญาณทางขาเข้ามี rise และ fall time เป็น 25 ns บวกหรือลบ 15 ns trannsmitter squelch circuit จะกำจัดที่มีความกว้างของ pulseน้อยกว่า20nsหรือระดับ 0 ถึง --175 mv transmitter จะหยุดการทำงานเมื่อสัญญาณอยู่ในระดับสูงกว่า --175 mv นานเกิน 300 ns

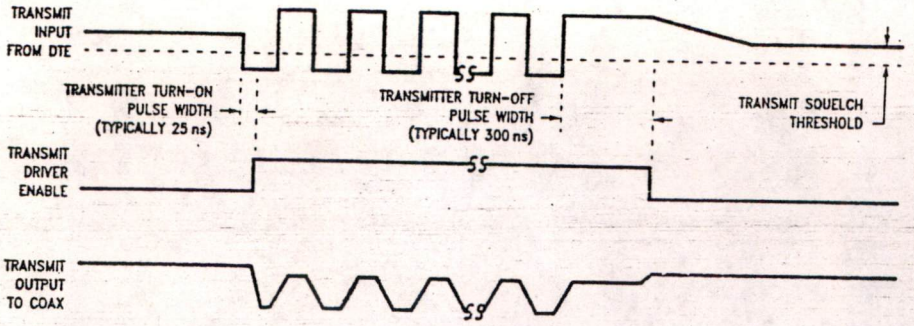


FIGURE 3. Transmitter Timing

3. Collision Functions

ประกอบด้วยBuffer 2 ชุด , 4 pole Bessle LPF 2 ชุด , Comparator , Heartbeat Generator 10 MHz Oscillator และ Differential Line Driver

Buffer และ LPF จะใช้เพื่อแยกระดับ DC ของ CENTER CONDUCTOR (Data) และที่ Shield (Sense)ของสาย Coaxial ระดับทั้งสองจะถูกนำมาเปรียบเทียบโดย Comparator ถ้าระดับ DATA เป็นลบ มาก ๆ กว่าระดับ Sense บวก Vth หมายความว่ามีการชนเกิดขึ้น สัญญาณ 10 MHz จะถูกส่งออกไปยัง DTE เพื่อแจ้งให้ทราบว่ามีการชนกัน

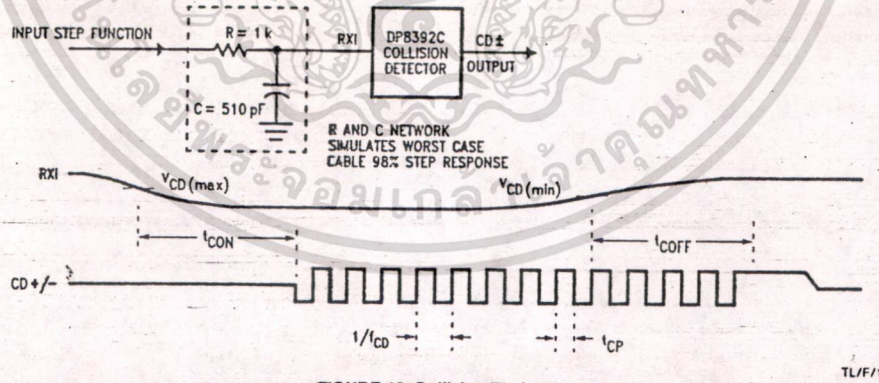


FIGURE 10. Collision Timing

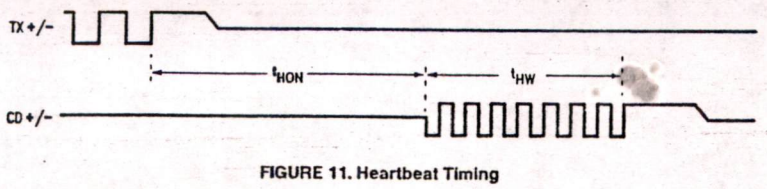


FIGURE 11. Heartbeat Timing

ทุกครั้งเมื่อสิ้นสุดการส่งประมาณ 1.1 ไมโครวินาที Heartbeat Generator จะสร้างสัญญาณ Collision ขึ้นมานาน 1 ไมโครวินาที เพื่อทดสอบว่าวงจร Collision Detect จะทำงานอย่างถูกต้องถ้าไม่ต้องการ ให้นำขา HBE ต่อกับ VEE ถ้าให้ทำงานก็ต่อกับ Ground

4. Jabber Functions.

Jabber Timer จะคอยดูการส่งให้ไม่เกิน 20 ms เมื่อถึง 20 ms จะส่งสัญญาณ Collision ไปยัง DTE จากนั้นก็จะรอประมาณ 500 ms (random)ก่อนจะทำให้ Transmitter ทำการส่งได้ใหม่ ในช่วงเวลา 500 ms ที่ขา Transmit Input จะไม่ทำงาน

บทที่ 3
โปรแกรมรับส่งข้อมูลผ่าน ETHERNET ADAPTER

โปรแกรมนี้เขียนขึ้นด้วยภาษา C เพื่อรับส่งไฟล์ข้อมูลระหว่างคอมพิวเตอร์สองเครื่องที่ใช้COMPEX ENET 16/U Ethernet Adapter Card โปรแกรมนี้จะติดต่อกับการ์ดโดยรันผ่านดอส หน้าที่ของโปรแกรมมีดังนี้

1) เริ่มต้นและหยุดการทำงานของการ์ด
2) ทำการรับและส่งข้อมูล

โปรแกรมจะประกอบด้วยไฟล์หลัก 3 ไฟล์ คือ

- ether.h
- dp901.h
- ether.c

แต่ละไฟล์ทำหน้าที่ดังนี้

1. ether.h : กำหนดชนิดของโครงสร้างของข้อมูล, ค่าคงที่, function, prototype ซึ่งใช้สำหรับซอฟ-แวร์ไดรเวอร์เมื่อเรียกฟังก์ชั่นใน ether.c
2. DP901.h : กำหนดชนิดของโครงสร้างของข้อมูลและค่าคงที่สำหรับส่งและรับเฟรมผ่านการ์ดซึ่งมี DP83901 เป็น Network Controller Chip
3. ether.c : ทำหน้าที่เริ่มและหยุดการทำงานของการ์ดและรับส่งไฟล์ข้อมูล ประกอบด้วยฟังก์ชั่นที่สำคัญดังนี้
   - StartCard ทำหน้าที่เริ่มการทำงานของการ์ดและชิพ ,เซ็ต interrupt handler
   - StopCard ทำหน้าที่หยุดการทำงานของการ์ดและชิพ
   - ReceiveInterrupt เป็น interrupt handler สำหรับจัดการกับอินเทอร์รัพท์ที่เกิดขึ้นเมื่อมีเฟรมถูกรับเข้ามาในบัฟเฟอร์
   - ExtractFrames ทำหน้าที่นำเฟรมที่เข้ามาออกจากบัฟเฟอร์มาใช้ในหน่วยความจำ
   - SendFrame มีหน้าที่นำข้อมูลจากหน่วยความจำออกไปยังบัฟเฟอร์และส่งออกไป
   - ReceiveFrame เป็นฟังก์ชันที่อยู่ใน interrupt handler ทำหน้าที่นำแพ็คเก็ตที่รับเข้ามามารวมกลับเป็นไฟล์

โปรแกรมและโฟล์วชาร์ตการทำงานของฟังก์ชันทั้งหมดจะอยู่หน้าถัดไป

```
    /*
*   Ether.c - Ethernet access functions for ENET16/U 16-bit Ethernet
*    Adapter
*   StartCard - start network card.
*   StopCard - stop network card.
*   ReceiveInterrupt - accept packet reception interrupt and call
*   ReceiveFrame.
*   SendFrame - send frame over ethernet.
*/

#define  _NEDRVR_

#include <io.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <ctype.h>
#include "ether.h"
#include "dp901.h"

int etherdebug = 0;

static interruptVector PreviousHandler; /* place holder for old
                                           handler */
static unsigned char sav_ss, sav_sp;  /* interrupt user stack */
static void interrupt ReceiveInterrupt(void);  /* pointer to handler */
int CheckInterrupt(void);

#define STKLEN 2048              /* length of stack */
static unsigned char RxStack[STKLEN];  /* receive interrupt stack */
static unsigned  nextpage;   /* software counter for buffer page */

/*
 *  StartCard - start network card
 *  argument : local Ethernet address returned in 'addr'
 *  return : 0 on success, -1 on failure
 *
 *  - reset ENET card
 *  - reset SNIC chip
 *  - read Ethernet address from  card ROM
 *  - set it in chip and in 'addr'
 *  - intialize buffer ring
 *  - connect card interrupt to my handler
 *  - start chip for send/receive operation
 */

int StartCard(Address *addr)  /* 'addr'-- local Ethernet address
                                 return */
{
 unsigned char *fp;
 int i;

 /* reset card and chip */

 outportb(ENETBANK, SB_RESET);  /* reset SNIC card */

 outportb(ENETBANK, SB_RUN|SB_ROM);
 if( inportb(ENETBANK) != SB_RUN|SB_ROM)
 { cprintf("ENET16/U board does not respond \r\n");
   cprintf("Press any key to continue \r\n");
   return -1;
 }
 outportb(SNICCOM, NC_STOP|NC_PAGE1);
```

```
    if(inportb(SNICCOM) != (NC_STOP|NC_PAGE1))
    { printf("SNIC chip does not respond \r\n");
      return -1;
    }

    fp = (unsigned char*) MK_FP(ENETMEM,0);
    for( i = 0; i < ALEN; i++, fp += 1)
    { addr->bytes[i] = *fp;
      outportb(SNICPAR0 + i), *fp);
    }

    /* set configuration */
    outportb(SNICCOM, NC_STOP|NC_PAGE0);     /* disabled chip,select
                                                pageO means that chip
                                                programmed during SNIC
                                                operation */
    outportb(SNICDCR, ND_NORMAL);            /* set configuration */
    outportb(SNICRCR, NR_NORMAL);            /* set receive mode */
    outportb(SNICTCR, NT_LOOPBACK);     /* switch to internal loopback */
    outportb(ENETBANK, SB_RUN|SB_RAM0);      /* switch to on board RAM */

    /* initialize buffer ring */

    outportb(SNICCOM, NC_STOP|NC_PAGE0);      /* select register page0 */
    outportb(SNICTPSR, NT_START);             /* send buffer at d000:0000 */
    outportb(SNICPSTART, NP_START);           /* receive buffer at
    outportb(SNICBNDRY, NP_START);            /* BNDRY == PSTART */
    outportb(SNICPSTOP, NP_STOP);        /* receive buffer ends at c000:80C
*/
    outportb(SNICCOM, NC_STOP|NC_PAGE1); /* select register page1 */
    outportb(SNICCURR, NP_START+1);      /* curr == pstart + 1; */

    nextpage = NP_START + 1;             /* software counter == curr */

    /* install interrupt handler */
    PreviousHandler = getvect(ENETINT);  /* save previous handler */
    setvect(ENETINT, ReceiveInterrupt);  /* set my handler */

    /* start chip */
    disable();
    outportb(SNICCOM, NC_STOP|NC_PAGE0); /* select register page0 */
    outportb(SNICIMR, NI_RX|NI_FULL);    /* accept receive interrupt */
    outportb(SNICISR, 0xff);             /* clear pending interrupt */
    outportb(SNICCOM, NC_START|NC_PAGE0); /* start SNIC */
    enable();

    if(etherdebug) cprintf("StartCard : done\r\n");
    return 0;
    }   /* End of StartCard */

/* ////////////////////////////////////////////////////////////////////// */
/* StopCard - stop network card                                           */
/* argument : card chip remain as is if "reset" is false                  */
/*            (for debug)                                                 */
/*                                                                        */
/* - restore old interrupt handler                                        */
/* - stop card and chip if reset is true                                  */
/* ////////////////////////////////////////////////////////////////////// */

void StopCard(bool reset)   /* reset -- full shutdown */
{
 if(etherdebug) cprintf( "StopCard: %s/r/n ",
                         reset ? "reset" : "no reset" );
 setvect(ENETINT, PreviousHandler);  /* restore old handler */
 if(reset)
  { outportb(SNICCOM, NC_STOP|NC_PAGE0);   /* stop SNIC */
```

```
    outportb(ENETBANK, SB_RESET);          /* reset SNIC */
   }
  }

/* ////////////////////////////////////////////////////////////////////// */
/* ReceiveInterrupt - SNIC recieve interrupt handler                      */
/*                                                                        */
/* -switch stack                                                          */
/* -check status                                                          */
/* -extract frame one by one and call ReceiveFrame with it                */
/* -call previous handler ( interrupt controller reenable )               */
/* -switch stack back                                                     */
/* -enable receive interrupt                                              */
/* ////////////////////////////////////////////////////////////////////// */

static void ExtractFrames(void);

static void interrupt ReceiveInterrupt()
{
 /* switch to my stack */
 disable();
 sav_ss = _SS;
 sav_sp = _SP;
 _SS = FP_SEG(RxStack);
 _SP = FP_OFF(&RxStack[STKLEN]);
 enable();

 ExtractFrames();

 /* switch back to user stack */
 disable();
 _SS = sav_ss;
 _SP = sav_sp;
 enable();

 (*PreviousHandler)(); /* old handler does EOI */
}

static void ExtractFrames(void)
{
 unsigned char isr, curr, bndry;
 SNICFrame *nfp;
 void *fp2;
 unsigned len1, len2;
 bool full;

 isr = inportb(SNICISR);  /* get SNIC status */

 if(etherdebug && isr )
     cprintf(" ReceiveInterrupt : ISR = %02x/r/n", isr);

 /* check alive ? */
 if (isr & NI_RESET)
 { cprintf("!!! SNIC abort , ISR = %02x !!! \r\n", isr);
   StopCard(false);
   return;
  }
 /* buffer overflow? */
 if (isr & NI_FULL)
  {  if(etherdebug) cprintf("ReceiveInterrupt : Overflow \r\n");
     outportb(SNICCOM,NC_STOP|NC_PAGE0);          /* stop SNIC */
     while (!(inportb(SNICISR) & NI_RESET));  /* wait until loop */
     outportb(SNICTCR, NT_LOOPBACK);              /* disable reception */
     outportb(SNICCOM, NT_START|NC_PAGE0);    /* start again */
```

```
        full = true;
    }
    else full = false;

    if (isr & NI_RX)    /* I have one or more frame */
     {
       while(true)
       {  outportb(SNICCOM,NC_START|NC_PAGE1);   /* switch to page 1 */
          curr = inportb(SNICCURR);     /* get current page pointer */
          outportb(SNICCOM,NC_START|NC_PAGE0); /* return to page0 */
          if(etherdebug)
          cprintf("ReceiveInterrupt : CURR = %02x \r\n",curr,nextpage);
          if(curr == nextpage)  break;        /* all frame taken */

          while(curr != nextpage)         /* I still have some */
           { /* extract frame */
             nfp = (SNICFrame*)MK_FP(ENETMEM,nextpage << 8);
             if(nfp->length > (NP_STOP - nextpage) >> 8  )
              { /* frame wrap around to the beginning */
                len1 = (( NP_STOP - nextpage) << 8) - 4;
                len2 = nfp->length - len1;
                fp2 = (void*) MK_FP(ENETMEM,NP_START << 8);
              }
             else
              { /* no wrap around */
                len1 = nfp->length;
                len2 = 0;
                fp2 = (void*)0;
              }
             if(etherdebug)
              cprintf("ReceiveInterrupt : Status %02x, length %d\r\n",
                      nfp->status, nfp->length);
             ReceiveFrame(nfp->frame, len1, fp2, len2);   /* call
                                                          driver */
             nextpage = nfp->next;    /* advance to next frame */
             bndry = (nextpage == NP_START) ? NP_STOP-1:nextpage-1;
             outportb(SNICBNDRY,bndry); /* free this frame */
           }   /* End of loop while(curr!=nextpage) */
       }  /* End of loop while(true) */
     }    /* End of loop if */


  if(full) /* stop has been stopped, restart it */
   { outportb(SNICTCR,NI_NORMAL);  /* enable reception */
     if(etherdebug)  cprintf("ReceiveInterrupt : Restored \r\n");
   }
  outportb(SNICISR,NI_RX|NI_FULL); /* acknowledge interrupt */

}

/* ////////////////////////////////////////////////////////////////// */
/* SendFrame - send frame over Ethernet                               */
/* arguments : two buffer containing protocol header and their        */
/*             length                                                 */
/* return : 0 on success , -1 on failure                              */
/*                                                                    */
/* -disable receive interrupt                                         */
/* -pass data to chip and kick it                                     */
/* -wait completion or time out                                       */
/* -reenable receive interrupt                                        */
/* ////////////////////////////////////////////////////////////////// */

int SendFrame(void *header, unsigned hlen, void *data, unsigned dlen)
{
  unsigned char isr, tsr;
  int i, error = -1;
```

```
        unsigned off, len;

        /* mask out receive interrupts */
        disable();
        outportb(SNICIMR,0);
        enable();

* copy user data over send buffer */
        off = NT_START << 8;
        movedata(FP_SEG(header), FP_OFF(header), ENETMEM, off, hlen);
        movedata(FP_SEG(data), FP_OFF(data), ENETMEM, off + hlen, dlen);

        /* inform SNIC of frame length */
        len = hlen + dlen;
        if(len < 60) len = 60;  /* Ethernet maximum frame size */
        outportb(SNICTBCR0, (unsigned char)(len & 0xff));  /* lower byte */
        outportb(SNICTBCR1, (unsigned char)(len >> 8));    /* upper byte */
        if (etherdebug)
        cprintf("SendFrame : TBCR1 = %02\r\n",
                (unsigned char)(len >> 8),
                (unsigned char)(len & 0xff));
        outportb(SNICCOM,NC_START|NC_SEND|NC_PAGE0);     /* start sending */
        for(i = 1; i != 0;i++)     /* while 16-bit increment wraps around */
         {
           isr = inport(SNICISR);     /* get status */
           if(etherdebug && isr)
            {
              tsr = inportb(SNICTSR);  /* get send-specific status */
              cprintf("SendFrame : ISR = %02x, TSR = %02x\r\n",isr,tsr);
            }
           if(isr & NI_TX)
            {
              outportb(SNICISR,NI_TX);  /* acknowledge interrupt */
              if(isr & NI_TXOK) error = 0;
              break;
            }
         }
        if(etherdebug) cprintf("SendFrame : Error = %d\r\n",error);
        disable();
        outportb(SNICIMR, NI_RX|NI_FULL);
        enable();
        return error;
       }

       #define FileOverhead 17
       #define MaxPacketSize 1483
       FILE *fpr,*fpt;
       static FrameHeader *header;
       static Info *fileinfo;
       void ReceiveFrame(void *buf1, unsigned len1, void *buf2,unsigned len2)
       {
         static unsigned char *RxPtr;
         char select;
         static int i;
         int fhand,count;

         clrscr();
         printf("There is packet now, Do you want to receive?(Y/N) : ");
         do{ select = getch();
             select = toupper(select);
         }while(select != 'Y' && select != 'N');
         switch(select)
         { case 'N' :
             return;
           case 'Y' :
             while(fileinfo->number==1)
```

```
            movedata(FP_SEG(buf1),FP_OFF(buf1)+FileOverhead,FP_SEG(RxPtr),
                     FP_OFF(RxPtr),fileinfo->size-len2);
            movedata(FP_SEG(buf2),FP_OFF(buf2),FP_SEG(RxPtr),
                     FP_OFF(RxPtr)+len1-FileOverhead,len2);
            RxPtr = RxPtr +len1+len2-FileOverhead;
            if(fileinfo->number != fileinfo->totalnumber) return;

            if(i==fileinfo->totalnumber)  /* open file */
             { if((fpr = fopen("&fileinfo->fname","wb")) == NULL)
                { puts("Cannot receive a file\n");
                  puts("Press any key to continue\n");
                  getch();
                  return;
                 }
               printf(" file name received : %s\n",&fileinfo->fname);
               printf(" file size : %d\n",fileinfo->totalsize);
               fhand = _open("&fileinfo->fname",O_WRONLY | O_BINARY);
               count = fileinfo->totalsize;
               _write(fhand,&RxPtr,count);
               puts("File is received successfully");
               puts("Press any key to continue");
               getch();
               return;
             }
      } /* switch */

     }

    int CheckInterrupt(void)
     {
       int isr;
       isr = inport(SNICISR);
       if(etherdebug && isr)
       return 1;
       else return 0;
     }

    Address myaddr = { 0x00,0x80,0x48,     /*  for local Ethernet address */
                0x81,0x1C,0x11   };

    char  desaddr[] = {0x00,0x80,0x48,     /*  remote Ethernet address */
                0x81,0x1C,0x16   };

    main()
    {  void  *data;
       unsigned hsize, dsize,fsize;
       unsigned char  select,filename[11],*name,*buffer,packet[1500];
       int i,fhand,piece,piecesize;
       unsigned int totalpacket;

       etherdebug = 1;

       clrscr();
       if(StartCard(&myaddr) < 0)
       {
         getch();
         exit(1);
       }
       printf("Local address : %x \r\n", &myaddr);
       while(1)
       {
        /* begin to send */

        printf("\n Enter a filename to send : ");
        gets(filename);
        if((fpt = fopen(filename,"rb")) == NULL)  /* cannot open file */
          printf("\n Cannot open file %s\n", filename);
```

```
else
 { fhand = _open(filename,O_RDONLY | O_BINARY);
   dsize = (unsigned)filelength(fhand);  /* find length of file */
   hsize = 14;                           /* length of frame header */

   &fileinfo->fname = filename;
   fileinfo->totalsize = dsize;
   if(dsize%MaxPacketSize != 0)
    {  totalpacket =  (dsize/MaxPacketSize) +1;
       piece = 1;
    }
   else
    {  totalpacket = (dsize/MaxPacketSize);
       piece = 0;
    }

   setbuf(fpt,buffer);                   /* allocate memory for file */
   data = (void*)buffer;
   _read(fhand,&data,dsize);             /* write open file to memory */
   fileinfo->number = 0;
   fileinfo->totalnumber = totalpacket;

if(!piece)
 {
   for(i=0;i<totalpacket;i++)
    {
     (fileinfo->number)++;
     movedata(FP_SEG(fileinfo),FP_OFF(fileinfo),FP_SEG(packet),
              FP_OFF(packet),FileOverhead);
     movedata(FP_SEG(data),FP_SEG(data)+(MaxPacketSize*i),
              FP_SEG(packet),FileOverhead,MaxPacketSize);
     SendFrame(header,hsize,(void*)packet,1500);
    }
 }

else if(dsize < MaxPacketSize)
  { fileinfo->totalnumber = 1;
    movedata(FP_SEG(fileinfo),FP_OFF(fileinfo),FP_SEG(packet),
             FP_OFF(packet),16);
    movedata(FP_SEG(data),FP_SEG(data)+(MaxPacketSize*i),
             FP_SEG(packet),FileOverhead,MaxPacketSize);
    SendFrame(header,hsize,(void*)packet,dsize);
  }
    else
     { for(i = 0;i<totalpacket;i++)
       {
        (fileinfo->number)++;
         movedata(FP_SEG(fileinfo),FP_OFF(fileinfo),
                  FP_SEG(packet),FP_OFF(packet),FileOverhead);
         movedata(FP_SEG(data),FP_SEG(data)+(MaxPacketSize*i),
                  FP_SEG(packet),FileOverhead,MaxPacketSize);
        SendFrame(header,hsize,(void*)packet,1500);
       if(i==totalpacket-1)
        {peicesize = dsize - (MaxPacketSize*(i+1));
         (fileinfo->number)++;
          movedata(FP_SEG(fileinfo),FP_OFF(fileinfo),
                   FP_SEG(packet),FP_OFF(packet),FileOverhead);
          movedata(FP_SEG(data),FP_SEG(data)+(MaxPacketSize*i),
                   FP_SEG(packet),FileOverhead,peicesize);
          SendFrame(header,hsize,(void*)packet,
                    FileOverhead+peicesize);
        }
       }
   }
 fclose(fpt);
}  /* send file completed */
```

```
    printf(" Do you want to quit this program?(Y/N) :\n ");
    do{ select = getch();
       select = toupper(select);
    }while(select != 'Y' && select != 'N');
    switch(select) {
    case 'N' :
      break;
    case 'Y' :
      StopCard(true);
      exit(1);
    };
  } /* end of while(1) loop */
}
```

```
/*
 * ether.h -- General Ethernet definition.
 * This file defines data types, constant and function prototypes
 * that the protocol driver software needs when calling Ethernet
 * access function in enet.c
 */

/*
 * ether address and frame header format
 */

#define ALEN 6

typedef struct  {
  unsigned char bytes[ALEN];
}Address;

typedef struct {
  unsigned char name[6];
}Fname;

typedef struct {
  Address toaddr;
  Address fromaddr;
  unsigned short type;
} FrameHeader;

typedef enum { false = 0 , true } bool;

/* pointer to interrupt handler */
typedef void interrupt (*InterruptVector)();

#define ETHERMAX 1514 /* maximum ethernet frame length */

#ifdef _NEDRVR_
/*
 * Globals
 */

extern int etherdebug;

/*
 * prototypes
 */

int StartCard(Address *address);
void StopCard(bool reset);
int SendFrame(void *header, unsigned hlen, void *data, unsigned dlen);
void ReceiveFrame(void *buf1,unsigned len1,void *buf2, unsigned len2);

#endif
```

```
/*
 * DP901.h -- Ethernet controller constant.
 * This is defines data types and constant for sending and receiving
 * Ethernet frame through ENET16/U 16-bit Ethernet Adapter DP83901
 * SNIC
 * (Serial Network Interface Controller)  chip.
 */

/*
 * SNIC receive buffer format
 */

typedef struct{          /*  5 byte packet header */
  unsigned char status;  /* receive status */
  unsigned char next;    /* next buffer page */
  unsigned short length;     /* frame length */
  unsigned char frame[1];    /* start of frame (frame may
                                wrap around) */
}SNICFrame;

typedef struct{                 /* File information */
  Fname    fname;
  unsigned char number;
  unsigned char totalnumber;
  unsigned int  size;
  unsigned int  totalsize;
}Info;



/*
 * SNIC constant (default configuration)
 */


#define ENETMEM  0xD000      /* share memory segment (D000:0000) */
#define ENETINT  0x04        /* interupt level */

#define ENETBANK 0x358    /* SNIC bank register port */
#define SB_RESET 0x00      /* reset card */
#define SB_RUN   0x80      /* run card */
#define SB_ROM   0x14      /* set bank to address ROM */
#define SB_RAM0  0x10      /* set bank to first 32K RAM */
#define SB_RAM1  0x11      /* set bank to last 32K RAM (not used) */

#define SNICCOM   0x0340     /* SNIC command register port */
#define NC_STOP   0x21      /* stop chip */
#define NC_START  0x22      /* start chip */
#define NC_SEND   0x24      /* send frame */
#define NC_PAGE0  0x20      /* switch to register page 0 */
#define NC_PAGE1  0x60      /* switch to register page 1 */

#define SNICPAR0   0x0341     /* SNIC physical address register 0
                                 port */

#define SNICDCR    0x034E  /* SNIC data configuration register port */
#define ND_NORMAL  0x49       /* no loopback, word transfer */
#define SNICTCR    0x034D  /* SNIC transmit configuration register
                              port */
#define NT_NORMAL  0x00       /* no loopback */
```

```
#define NT_LOOPBACK  0x02     /* internal loopback */
#define SNICRCR  0x034C /* SNIC recieve configuration register port */
#define NR_NORMAL   0x04     /* accept addressed and broadcast frame */
#define SNICTPSR  0x0344  /* SNIC transmit page start register port */
#define NT_START   0x00  /* 0000 - (the very start of shared memory) */
#define SNICPSTART 0x0341     /* SNIC page start register port */
#define NP_START    0x06      /* 0600 - ( send frame occupied 1.5K ) */
#define SNICPSTOP   0x0342    /* SNIC page stop register port */
#define NP_STOP    0x80    /* -8000 (up ot 32K);lower half of 64K RAM */
#define SNICBNDRY   0x0343    /* SNIC boundary pointer register port */
#define SNICCURR    0x0347     /* SNIC current page register port */

#define SNICISR     0x0347    /* SNIC interrupt status register port */
#define SNICIMR     0x034F    /* SNIC interrupt mark register port */
#define NI_RESET    0x80      /* SNIC reset */
#define NI_RXOK     0x01      /* packet received successfully */
#define NI_RXBAD    0x04      /* packet received error */
#define NI_RX  (NI_RXOK | NI_RXBAD)  /* be used at overflow routine */
#define NI_TXOK     0x01      /* packet transmited successfully */
#define NI_TXBAD    0x08      /* packet transmited error */
#define NI_TX  (NI_TXOK | NI_TXBAD)  /* be used at overflow routine */
#define NI_FULL     0x10      /* no page left for packet reception */

#define SNICTBCR0 0x0345 /* SNIC transmit byte count register0 port */
#define SNICTBCR1 0x0346 /* SNIC transmit byte count register1 port */
#define SNICTSR  0x0344  /* SNIC transmit status register port */
```

บทสรุป

โครงการวิจัยนี้ได้ทำการพัฒนาซอฟท์แวร์สำหรับใช้ในระบบโลคัลแอเรียเนตเวอร์คแบบไร้สาย โดยซอฟท์แวร์ดังกล่าวถูกแบ่งออกเป็นสองส่วน คือส่วนที่ใช้ควบคุมกับระบบและส่วนที่ใช้ทางด้านการใช้งาน โดยที่ส่วนทั้งสองได้ถูกออกแบบและถูกเขียนออกมาเป็นโปรแกรมด้วยภาษาC ในลักษณะขนานกันไป สำหรับส่วนที่ใช้ในการควบคุมนั้นจะถูกแบ่งออกเป็นส่วนย่อย ๆ อีก ซึ่งเรียกว่าโมดูล ซึ่งโมดูลย่อย ๆนั้นได้แก่โมดูลเซทค่าพารามิเตอร์ โมดูลการส่ง โมดูลการรับ เป็นต้น ส่วนโปรแกรมทางด้านรับก็เช่นเดียวกันก็จะถูกแบ่งเป็นส่วนย่อย ๆที่เรียกว่าโมดูลเช่นกัน สำหรับโปรแกรมการใช้งานที่นิยมกันในปัจจุบันได้แก่โปรแกรมการส่งไฟล์ข้อมูล โปรแกรมการส่งจดหมายอิเลคทรอนิกส์ เป็นต้น ซึ่งทางงานวิจัยนี้ได้สร้างโปรแกรมง่าย ๆสำหรับการถ่ายเทข้อมูลซึ่งอาจจะเป็นข้อมูลที่เป็นตังอักษร หรือข้อมูลที่เป็นภาพเป็นต้น

อย่างไรก็ตามเนื่องจากอุปกรณ์ทางฮาร์ดแวร์แพงมากจนไม่สามารถซื้อมาใช้ทำการทดลองได้ดังนั้นงานวิจัยนี้นอกจากจะพัฒนาทางด้านซอฟท์แวร์แล้วยังจำเป็นที่จะต้องออกแบบฮาร์ดแวร์เพื่อใช้ในการประกอบเป็นอุปกรณ์ที่สามารถใช้สำหรับงานทดลอง ส่วนประกอบที่จำเป็นจะมีอยู่สองส่วนคือ ส่วนของระบบวิทยุและส่วนของอินเตอร์เฟส สำหรับส่วนของระบบวิทยุนั้นเป็นส่วนที่รับเอารหัสที่ส่งออกมาจากส่วนควบคุมระบบเพื่อส่งรหัสนั้นออกอากาศโดยการเข้ารหัสที่เหมาะสม ส่วนอินเตอร์เฟสนั้นจะใช้เป็นตัวปรับเงื่อนไขต่าง ๆของระบบควบคุมกับส่วนของวิทยุ ซึ่งโครงงานดังกล่าวกำลังดำเนินการอยู่

หนังสืออ้างอิง

(1) National Gurewich, Communication System : Practical quide to Intel's connectivity, Intel/McGraw-Hill, 1992

(2) Local Area Network Databook, National Semiconductor, 1992

(3) Andrew Schulman, Raymond J. Micheals, Jim Kyle, Tim Paterson, David Maxey, and Ralf Brow, " Undocumented DOS ", Addison Wesley, 1990

(4) Joe Campell, "C Programmer's Guide to Serial Communications",Macmillan 1990

(5) Nathan Gurewich," Communication Systems", McGraw-Hill 1992