การส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ท

VOICE OVER ETHERNET

โดย

นายเกียรติชัย พอแฮ้ว

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ภาควิชา
วิศวกรรมโทรคมนาคม

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

การส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ท

VOICE OVER ETHERNET

โดย

นายเกียรติชัย พอแฮ้ว 46010063

อาจารย์ที่ปรึกษา

รศ.ดร. ปราโมทย์ วาดเขียน

ผศ.ดร. จิรสุดา โกษียาภรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ท**

Voice over Ethernet

ผู้จัดทำ **1. นายเกียรติชัย พอแฮ้ว** **46010063**

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ปราโมทย์ วาดเขียน)

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. จีรสุดา โกษียาภรณ์)

โครงงาน 492409 **การส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ท**

Voice over Ethernet

โดย นายเกียรติชัย พอแฮ้ว 46010063

อาจารย์ที่ปรึกษา รศ.ดร. ปราโมทย์ วาดเขียน

ผศ.ดร. จีรสุดา โกษียาภรณ์

บทคัดย่อ

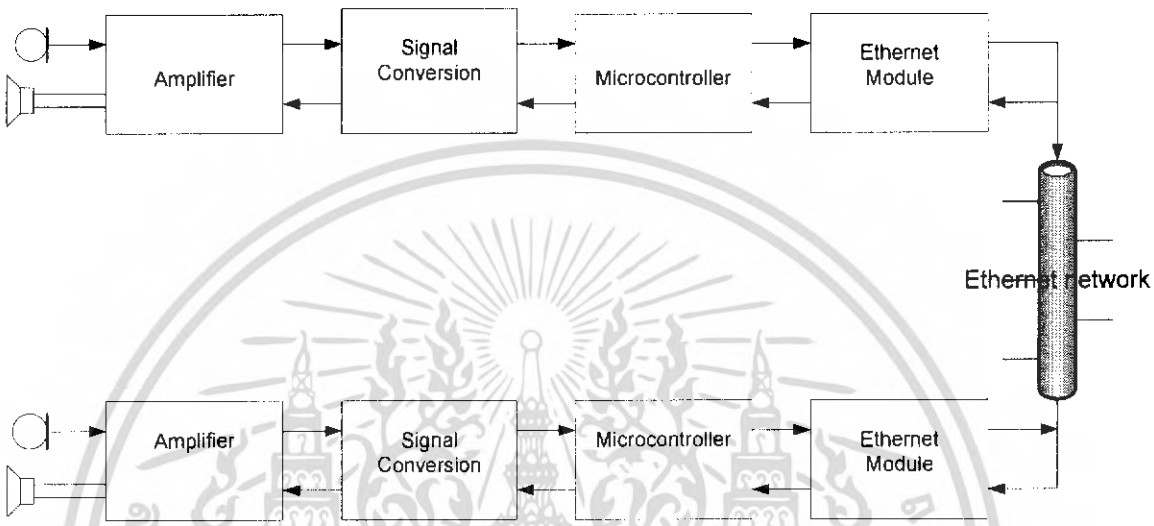
โครงงานนี้เป็นการสร้างระบบการส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ท ซึ่งโครงงานนี้ประกอบไปด้วยเอทูดีทำหน้าที่ในการแปลงสัญญาณจากอะนาล็อกเป็นดิจิตอล และอีเทอร์เน็ทโมดูลที่ทำหน้าที่ในการเชื่อมต่อสัญญาณเข้ากับโครงข่าย นอกจากนี้ยังมีคอนโทรลเลอร์ควบคุมการทำงานของระบบ โดยระบบสามารถส่งสัญญาณเสียงได้แบบฟูลดูเพล็กซ์

Abstract

This project constructs the system for transmit the voice signal through Ethernet. The project is composed of analog to digital converter which converts analog data to digital signal and Ethernet module which connect to the network . In addition the system is controlled by the controller where the system can send signal in full duplex mode.

บทที่ 2
ทฤษฎีและหลักการ

ในส่วนของบทที่ 2 นี้ จะกล่าวถึงทฤษฎีและหลักการที่เกี่ยวข้องกับองค์ประกอบหลักของโครงงาน ซึ่งจะเป็นการอธิบายเพื่อแสดงรายละเอียดต่างๆ เพื่อให้เข้าใจเกี่ยวกับโครงงานนี้มากขึ้น ซึ่งจะขออธิบายส่วนต่างๆ ตามภาพรวมของระบบดังแสดงดังรูปที่ 2.1

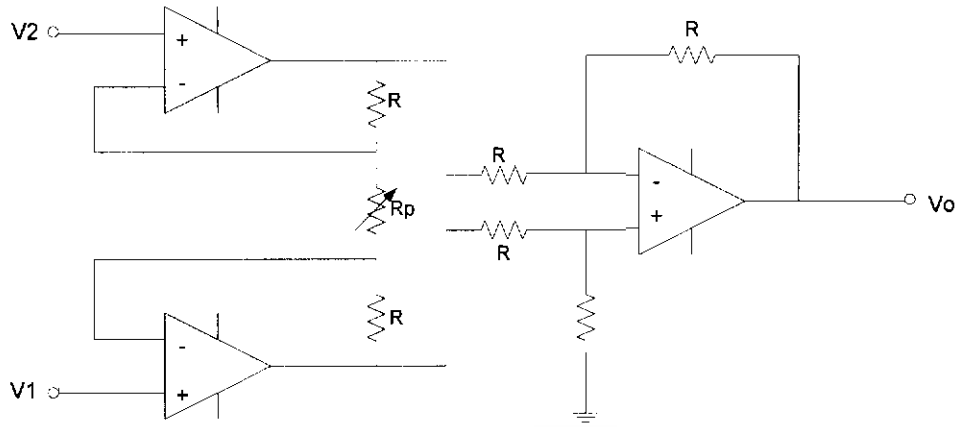


รูปที่ 2.1 ภาพรวมของระบบการส่งเสียงผ่านเครือข่ายอีเทอร์เน็ท

2.1 วงจรขยาย (Amplifier)

2.1.1 วงจรขยายในเครื่องมือวัด (Instrumentation Amplifier)

จากวงจรออปแอมป์พื้นฐานได้นำมาสู่วงจรวงจรขยายในเครื่องมือวัด วงจรขยายในเครื่องมือวัดเป็นวงจรขยายที่ได้รับความนิยมใช้กันมากในระบบการวัดและระบบควบคุม เพราะว่าสามารถปรับอัตราการขยายได้ ทั้งนี้เนื่องจากในระบบการวัดระบบหนึ่งอาจมีการเชื่อมต่อเพื่อรับสัญญาณจากตัวตรวจจับสัญญาณที่ต่างชนิดกัน ซึ่งสัญญาณที่ได้จากตัวตรวจจับสัญญาณหลายประเภทมักจะมีช่วงของระดับสัญญาณที่แตกต่างกันไป ดังนั้นการปรับอัตราการขยายได้จะทำให้ทุก ๆ ช่องสัญญาณถูกปรับให้เอาท์พุทมีระดับแรงดันที่เท่ากันทำให้สะดวกในการนำไปเชื่อมต่อกับวงจรอื่นต่อไป



รูปที่ 2.2 วงจรขยายอุปกรณ์วัดแบบปรับอัตราการขยายได้

ความต้านทานของโพเทนชิโอมีเตอร์หรือ Rp จะเป็นตัวปรับค่าตัวประกอบสเกล ดังนี้

$$\frac{V_O}{V_1 - V_2} = 1 + \frac{2R}{R_P}$$

$$V_O = (1 + \frac{2R}{R_P})(V_1 - V_2) = k(V_1 - V_2)$$

คุณสมบัติที่ดีของวงจรขยายในเครื่องมือวัดดังกล่าว คือ

1. ความต้านทานทางด้านอินพุทมีค่าสูงมาก และไม่เปลี่ยนแปลงตามอัตราการขยาย ในขณะที่ออปแอมป์ธรรมดาจะเปลี่ยนแปลงเสมอ

2. แรงดันเอาท์พุทที่ได้ไม่ขึ้นอยู่กับแรงดันคอมมอนที่มาจากทั้ง V1 และ V2 เลย แต่จะเป็นผลที่เกิดขึ้นกับค่าแรงดันดิฟเฟอร์เรนเชี่ยลระหว่าง V1 กับ V2 เท่านั้น

3. มีอัตราการขยายรวมสูง ทำให้สามารถนำไปใช้กับสัญญาณอินพุทที่มีระดับต่างกันได้อย่างกว้างขวาง

2.1.2 วงจรขยายกำลังงาน (Power Amplifier)

คุณสมบัติของวงจรขยายกำลังงาน

จุดประสงค์ในการออกแบบวงจรขยายกำลังงานทุกประเภท ต้องการให้เป็นดังต่อไปนี้

1.ประสิทธิภาพสูง เนื่องจากจะส่งผลกับวงจรภาคจ่ายไฟเลี้ยง เพราะถ้าวงจรขยายกำลังมีประสิทธิภาพสูงการใช้พลังงานจะเป็นไปอย่างคุ้มค่า ดังนั้นเมื่อต้องการกำลังงานที่เท่าถัน วงจรที่มีประสิทธิภาพสูงกว่าจะใช้กำลังงานจากแหล่งจ่ายไฟเลี้ยงน้อยกว่า การออกแบบแหล่งจ่ายไฟเลี้ยงจึงทำได้ง่ายกว่า ราคาก็จะถูก และอีกสาเหตุหนึ่งคือถ้าวงจรมีประสิทธิภาพสูงจะมีการสูญเสียกำลังงานเป็นความร้อนต่ำ จึงส่งผลต่อการระบายความร้อนโดยใช้แผ่นระบายความร้อน(Heatsink) ขนาดเล็ก ซึ่งแผ่นระบาย

ความร้อนนับเป็นอุปกรณ์อีกชิ้นที่มีราคาแพง เมื่อต้องระบายความร้อนต่ำก็จะใช้แผ่นระบายความร้อนขนาดเล็ก จึงประหยัดราคาในส่วนนี้ได้ และประสิทธิภาพในการขยายกำลังงานจะมีนิยมอยู่ 3 แบบ ดังนี้

1. การวัดประสิทธิภาพที่ขาเดรน (Drain) หรือขาคอลเลกเตอร์ (Collector) เป็นประสิทธิภาพของวงจรกำลังขยายกำลังในจุดที่ให้สัญญาณเอาท์พุท คำนวณได้ตามสมการ (2.1)

$$\eta = \frac{P_{RF(out)}}{P_{DC(in)}} \times 100 \qquad (2.1)$$

เมื่อ η คือประสิทธิภาพของวงจรขยายกำลัง(เปอร์เซ็นต์)

$P_{RF(out)}$ คือกำลังงานของสัญญาณเอาท์พุทที่ผ่านการขยายกำลัง(วัตต์)

$P_{DC(in)}$ คือกำลังงานที่แหล่งจ่ายไฟป้อนให้วงจรขยายกำลัง(วัตต์)

2. การวัดประสิทธิภาพของวงจรเมื่อรวมการคำนวณค่าพลังงานสัญญาณอินพุท ในกรณีที่สัญญาณอินพุทที่เข้าสู่ภาคการขยายกำลังมีกำลังงานสูง กำลังงานของสัญญาณอินพุทจะถือเป็นกำลังงานสูญเสียที่ต้องจ่ายให้กับวงจรขยายกำลังด้วย การวัดประสิทธิภาพดังนิยามแรกจะไม่สมเหตุสมผล ดังนั้นจึงควรคำนวณโดยลบค่ากำลังงานอินพุทที่ถือเป็นกำลังที่ต้องสูญเสียให้วงจรออกจากกำลังงานเอาท์พุทดังสมการ (2.2)

$$\eta = \frac{\left(P_{RF(out)} - P_{RF(in)}\right)}{P_{DC(in)}} \times 100 \qquad (2.2)$$

เมื่อ $P_{RF(in)}$ คือกำลังงานของสัญญาณอินพุท(วัตต์)

3.การวัดประสิทธิภาพโดยรวม (Overall efficiency) เป็นค่าประสิทธิภาพโดยรวมของวงจรขยายกำลังทั้งระบบ โดยรวมทุกภาคการขยายซึ่งเป็นค่าประสิทธิภาพที่ใช้ตัดสินวงจรขยายกำลังได้ดีที่สุด และคำนวณได้ตามสมการ (2.3)

$$\eta = \frac{P_{RF(out)}}{(P_{DC(in)} + P_{RF(in)})} \times 100 \qquad (2.3)$$

ซึ่งค่าประสิทธิภาพของวงจรการขยายกำลังแบบต่างๆมีค่าโดยประมาณดังตารางที่ 2.1

ตารางที่ 2.1 ประสิทธิภาพของวงจรขยายกำลังคลาสต่างๆ

ประเภทของคลาส	ประสิทธิภาพสูงสุด ตามทฤษฎี(η_{max})	ค่าประมาณประสิทธิภาพ เมื่อใช้งานจริง
เชิงเส้น(Linear)		
คลาส A	50%	40%
คลาส B	78.5%	65%
คลาส AB	50%<η<78.5%	60%
แบบสวิทชิ่ง (Switching) คลาส D , กลุ่มคลาส E	100%	95%

2.ค่าตัวเลขสัญญาณรบกวน (Noise figure : NF) ต่ำ เนื่องจากสัญญาณรบกวนรวมทั้งหมดที่ปรากฏออกมาที่เอาท์พุทของวงจรขยายกำลัง ประกอบด้วยสองส่วนคือ สัญญาณรบกวนเอาท์พุตที่เกิดจากการขยายกำลังสัญญาณรบกวนอินพุท และสัญญาณรบกวนเอาท์พุทที่เกิดจากวงจรขยายกำลังเอง ดังนั้นค่าตัวเลขสัญญาณรบกวนจึงเป็นค่าที่บอกว่าระบบหรือการขยายกำลังนั้นมีการก่อกำเนิดสัญญาณรบกวนจากภายในตัวเครื่องขยาย ออกมาปนกับสัญญาณที่ต้องการมากหรือน้อยอย่างไร โดยคำนวณได้ตามสมการ (2.4)

$$NF = \frac{\overline{n_{T_0}^2(t)}}{\overline{n_{S_0}^2(t)}} \qquad (2.4)$$

เมื่อ $\overline{n_{T_0}^2(t)}$ เป็นกำลังเฉลี่ยรวมทั้งหมดของสัญญาณรบกวนที่ปรากฏออกมาที่เอาท์พุทของเครื่องขยายสัญญาณ

$\overline{n_{S_0}^2(t)}$ เป็นกำลังเฉลี่ยของสัญญาณรบกวนที่เป็นส่วนหนึ่งของสัญญาณรบกวนเอาท์พุท ซึ่งเกิดจากการขยายสัญญาณรบกวนอินพุท

หรือสามารถพิสูจน์ให้ง่ายต่อการวัดค่าเป็นดังสมการ (2.5)

$$NF = \frac{\left[\frac{S}{N}\right]_i}{\left[\frac{S}{N}\right]_o} \qquad (2.5)$$

เมื่อ $\left[\frac{S}{N}\right]_i$ คือ ค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวนของสัญญาณอินพุท

$\left[\frac{S}{N}\right]_o$ คือ ค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวนของสัญญาณที่ผ่านการขยาย

โดยอัตราส่วนสัญญาณต่อสัญญาณรบกวน(Signal to Noise Ratio : SNR) เป็นค่าที่บอกความแตกต่างของกำลังงานข่าวสารที่ต้องการ กับกำลังงานของสัญญาณรบกวน โดยถ้ามีค่ามากจะดีเพราะแสดงว่าสัญญาณข่าวสารมีกำลังงานสูงเมื่อเปรียบเทียบกับ กำลังงานของสัญญาณรบกวนและจะคำนวณค่าได้ตามสมการ (2.6)

$$\frac{S}{N} = \frac{\overline{s^2(t)}}{\overline{n^2(t)}} \tag{2.6}$$

เมื่อ $\frac{S}{N}$ คือค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวน

$\overline{s^2(t)}$ คือค่ากำลังเฉลี่ยของสัญญาณข่าวสาร

$\overline{n^2(t)}$ คือค่ากำลังเฉลี่ยของสัญญาณรบกวน

ค่าเอสเอ็นอาร์นี้ปรกติมักแสดงค่าในหน่วยเดซิเบล (Decibel : dB) คำนวณได้ตามสมการ (2.7)

$$\left.\frac{S}{N}\right|_{dB} = 10\log\left(\frac{\overline{s^2(t)}}{\overline{n^2(t)}}\right) \tag{2.7}$$

3.ความเพี้ยนฮาร์มอนิกรวม (Total harmonic distortion : THD) ***ต่ำ*** เป็นค่าที่บอกว่าเมื่อนำสัญญาณไซน์รูปสมบูรณ์ (Pure sinusoidal wave) ป้อนเข้าสู่การขยายกำลังงานแล้วมีความถี่ของสัญญาณอื่นที่เป็นจำนวนเท่าของความถี่ข่าวสาร หรือเรียกว่าความถี่ฮาร์โมนิกเกิดขึ้นที่เอาท์พุทหรือไม่ เพราะสัญญาณฮาร์โมนิกเป็นสัญญาณที่ไม่ต้องการเนื่องจากเมื่อรวมกับข่าวสารเดิมจะทำให้รูปสัญญาณผิดไป ค่า THD คำนวณได้ตามสมการ (2.8) หรือคำนวณตามสมการ (2.8) แล้วคูณ 100 เพื่อทำให้เป็นหน่วยเปอร์เซ็นต์

$$THD = \frac{\sum_{n=2}^{\infty} c_n^2}{c_1^2} \tag{2.8}$$

เมื่อ c_1 คือค่าแอมปลิจูดของสัญญาณข่าวสาร

c_n คือค่าแอมปลิจูดของสัญญาณฮาร์โมนิกที่ n ของสัญญาณข่าวสาร

4. มีแบนด์วิดท์และอัตราการขยายกำลังเหมาะสมกับประเภทของงาน เนื่องจากวงจรขยายกำลังใช้กับงานหลายประเภท ซึ่งมีความถี่ที่ต้องการขยายแตกต่างกัน เช่น การขยายกำลังงานสำหรับเครื่องเสียง ก็ต้องขยายเสียงในช่วงความถี่ที่หูคนได้ยิน(ประมาณ 10 Hz ถึง 25 Hz) หรือถ้าเป็นเครื่องขยายกำลังเพื่อส่งข่าวสารไปในช่องสัญญาณ ก็ต้องการขยายกำลังของสัญญาณที่ผ่านการมอดูเลตมา ซึ่งส่วนมากจะเป็นความถี่สูงกว่าความถี่เสียง ดังนั้นการขยายกำลังต้องคำนึงถึงผลการตอบสนองทางความถี่ (Frequency response) ของวงจรด้วยว่าวงจรขยายกำลังจะต้องมีผลตอบสนองคงที่ตลอดย่านความถี่ที่ใช้งาน หรือต้องไม่ตัดองค์ประกอบความถี่ของข่าวสารเพื่อให้สัญญาณภายหลังการขยายเหมือนเดิม ส่วนความถี่นอกย่าน

ที่ต้องการขยายกำลังก็เปรียบเสือนสัญญาณรบกวน ดังนั้นจึงไม่ควรขยายความถี่ที่อยู่นอกแบนด์วิดท์ของสัญญาณข่าวสาร

5. ราคาถูกเมื่อเทียบกับคุณภาพที่ได้ เป็นอีกปัจจัยหนึ่งที่ต้องคำนึงในการออกแบบ เพื่อความคุ้มค่าของผู้ซื้อและความสามารถในการแข่งขันกับวงจรขยายเสียงที่มีอยู่แล้ว โดยราคาเป็นผลจากเทคนิคที่ใช้ในการขยายกำลัง รวมถึงการเลือกคุณภาพของอุปกรณ์ที่ใช้ในวงจร และขั้นตอนในการสร้างทั้งหมด เช่น ถ้าออกแบบให้วงจรมีประสิทธิภาพสูง ความร้อนที่เกิดก็จะน้อย แผ่นระบายความร้อนที่มีราคาแพงก็จะใช้ไม่มาก ราคาก็จะถูกลง เป็นต้น ซึ่งราคาและคุณภาพจะเป็นตัวชี้วัดคุณค่าและความนิยมของวงจรกำลังขยายนั้นๆ กล่าวคือถ้ามีสองวงจรที่มีคุณภาพเท่ากัน วงจรที่ราคาถูกจะได้รับความนิยมกว่า

จากคุณสมบัติของวงจรขยายกำลังงานเสียงต่างๆ เหล่านี้ ต้องคำนึงถึงลำดับความสำคัญในการออกแบบให้เหมาะสมกับแต่ละประเภทงาน เพราะบางงานอาจไม่จำเป็นต้องใช้คุณภาพดีมากเพราะจะทำให้ราคาแพงเกินความจำเป็น ดังนั้นวงจรขยายกำลังจึงมีจำหน่ายหลายคุณภาพและหลายราคา

2.2 การแปลงสัญญาณ (Signal Conversion)

2.2.1 การแปลงสัญญาณอะนาล็อก - ดิจิตอล

สัญญาณที่ใช้ในอุปกรณ์อิเล็กทรอนิกส์ มี 2 ชนิด คือ สัญญาณอะนาล็อก และสัญญาณดิจิตอล สัญญาณอะนาล็อก จะใช้ในอุปกรณ์ทั่วๆ ไป และใช้ในการควบคุมแบบเก่า

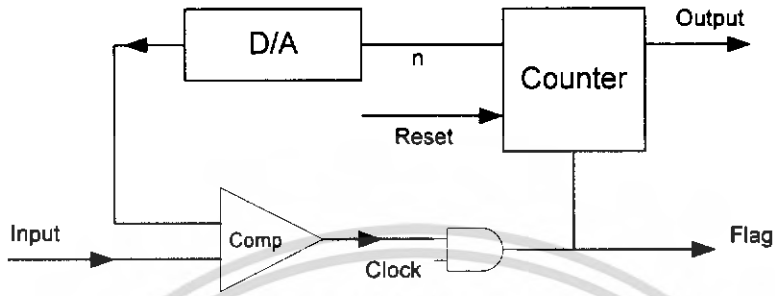
ในปัจจุบันมีไมโครโปรเซสเซอร์ และไมโครคอนโทรลเลอร์ เข้ามาช่วยในการควบคุมอุปกรณ์ต่างๆ มากมาย ซึ่งทำให้การควบคุมนั้นทำได้ง่าย และรวดเร็วยิ่งขึ้น แต่ในการควบคุมนั้น เราจำเป็นต้องใช้สัญญาณดิจิตอลในการติดต่อกับไมโครโปรเซสเซอร์ หรือไมโครคอนโทรลเลอร์ แต่ในความเป็นจริงนั้นเราใช้สัญญาณอะนาล็อกในการควบคุม ดังนั้นเราจึงจำเป็น ต้องมีการเปลี่ยนสัญญาณอะนาล็อก เป็นสัญญาณดิจิตอล แล้วจึงนำสัญญาณนั้นเข้ามาสู่ไมโครโปรเซสเซอร์ หรือไมโครคอนโทรลเลอร์ เพื่อใช้ควบคุมระบบต่อไป

แม้ว่าสัญญาณอะนาล็อกนั้นมีความแน่นอน และแม่นยำสูง แต่สัญญาณอะนาล็อกนั้นก็ควบคุมได้ยาก เนื่องจากในสภาพแวดล้อม มีสัญญาณรบกวนอยู่มาก และการที่จะทำให้ การควบคุมแบบอะนาล็อก มีความสามารถควบคุม เท่ากับการควบคุมแบบดิจิตอลนั้น ทำได้ยาก เนื่องจากวงจรควบคุมแบบ อะนาล็อกจะต้องมีความซับซ้อนสูง

อย่างไรก็ตาม สัญญาณดิจิตอลก็ไม่สามารถทดแทนความละเอียดของสัญญาณอะนาลือกได้อย่างสมบูรณ์ แต่ทำให้การควบคุมนั้นทำให้ง่าย และสะดวกยิ่งขึ้น

Counting Converter

Counting Converter เป็นวิธีที่ง่ายที่สุดของการแปลงสัญญาณอะนาล็อก เป็นสัญญาณดิจิตอล โดยใช้อัลกอริทึม การนับค่าเพิ่มขึ้นเรื่อยๆ แล้วนำผลที่ได้จากการนับไปเปรียบเทียบกับค่าที่ต้องการที่ตั้งไว้ ลักษณะการทำงานเป็นดังรูปที่ 2.3



รูปที่ 2.3 Analog to digital converter

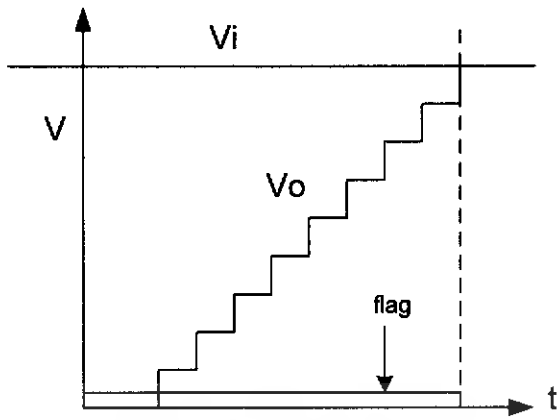
จากวงจร Counter เป็นอุปกรณ์นับค่าที่เพิ่มขึ้นทีละหนึ่ง แล้วส่งค่าที่ได้ให้ D/A มีขา Reset รับสัญญาณ Reset เมื่อต้องการให้เริ่มนับใหม่

D/A เมื่อรับค่าที่นับเพิ่มขึ้นทีละหนึ่งจากตัวนับ ก็แปลงค่าให้เป็นสัญญาณ อะนาล็อกที่มีค่าความต่างศักย์ค่าๆ หนึ่ง แล้วส่งต่อเข้าไปที่อุปกรณ์ตัวเปรียบเทียบ(Comparator)

Comparator จะเป็นอุปกรณ์ตัวเปรียบเทียบค่าความต่างศักย์ ของอินพุต และค่าจากที่ตัวนับ ถ้าหากทั้งสองสัญญาณมีค่าเท่ากันส่งค่าความต่างศักย์ 0 โวลต์ออกมา(ลอจิก 0) ถ้าไม่เท่ากันก็จะส่งความต่างศักย์ที่ไม่ใช่ 0 โวลต์ออกมา(ลอจิก 1)

ซึ่งค่าความต่างศักย์ที่ออกมา จะนำมาเข้าลอจิกเกต "และ" กับ สัญญาณนาฬิกา จะได้ค่าลอจิกออกมา ถ้าผลลัพธ์ออกมาเป็นสัญญาณนาฬิกาแสดงว่ายังไม่ได้ผลลัพธ์เท่าที่ต้องการ สัญญาณนาฬิกาก็จะไปทำให้ตัวนับนับเพิ่มขึ้นต่อไป และเมื่อได้ค่าผลลัพธ์ดิจิตอลที่ต้องการแล้ว ค่าที่ได้จาก ตัวเปรียบเทียบจะให้ค่าความต่างศักย์เป็น 0 (ลอจิก 0) ซึ่งเมื่อนำมาเข้าลอจิกเกต "และ" กับสัญญาณนาฬิกาแล้ว ก็จะให้ลอจิก 0 ซึ่งทำให้ตัวนับไม่นับเพิ่มอีก ก็จะได้ค่าดิจิตอลจากตัวนับที่ต้องการ

จากคำอธิบายข้างต้นจะได้กราฟของ V_o ดังรูปที่ 2.4

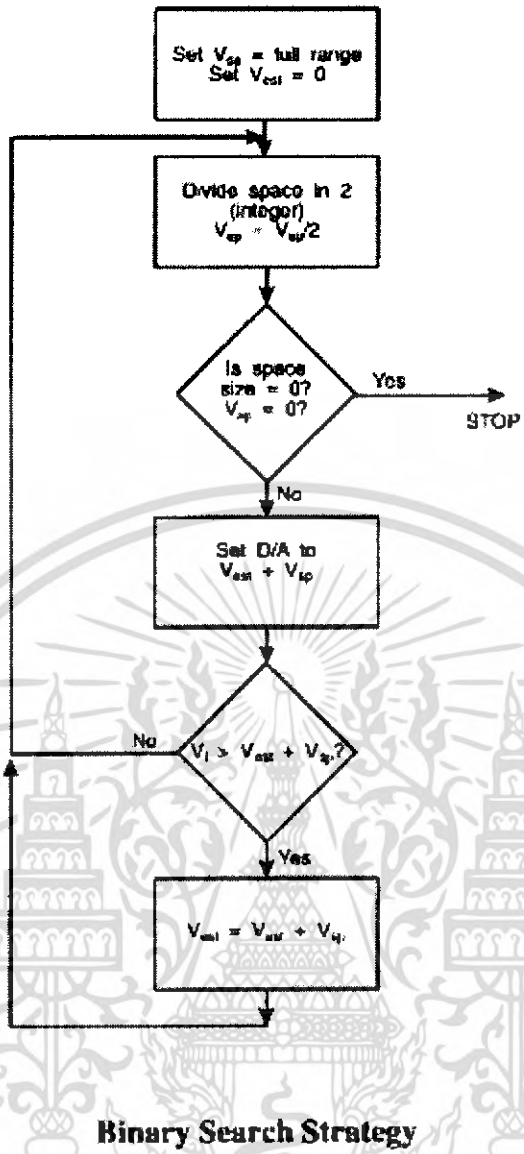


รูปที่ 2.4 Output voltage Graph from A/D converter

ข้อเสียของวิธีนี้ คือ การนับต้องเริ่มนับที่ 0 เสมอ และนับเพิ่มขึ้นเรื่อยๆ ทำให้ช้า เอาท์พุทที่ได้จะมี delay จึงไม่ค่อยนิยมใช้เท่าที่ควร จึงได้เปลี่ยนตัวนับเป็นแบบนับลงได้ด้วย ซึ่งจะอ้างอิงระดับจากระดับเก่า ทำให้ไม่จำเป็นต้องนับ 0 ใหม่ เมื่อมีการเปลี่ยนอินพุทใหม่ แต่ให้อ้างอิงกับผลลัพธ์เดิม ทำให้ได้ผลลัพธ์เร็วขึ้น

Successive Approximation

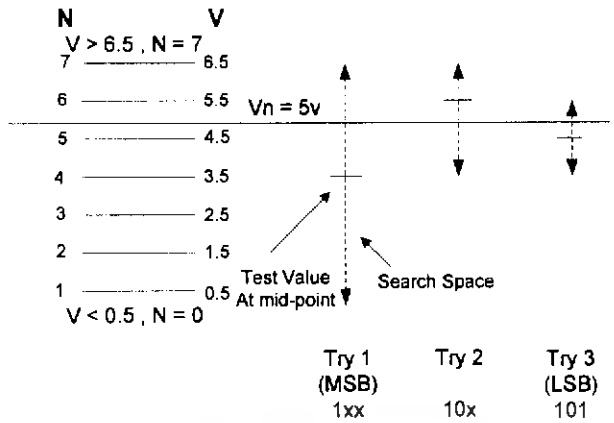
ใช้หลักการของ "binary search" ในการหาคำตอบ โดยนำค่าผลลัพธ์มาเปรียบเทียบกับค่ากึ่งกลางของช่วง เพื่อให้ทราบว่า ค่านั้นๆ มากกว่า หรือน้อยกว่า โดยจะปรับช่วงให้แคบลงมาเรื่อยๆ แล้วเปรียบเทียบผลลัพธ์กับค่ากึ่งกลางของช่วงไปเรื่อยๆ จนได้ผลลัพธ์ที่ต้องการ เช่น เลขที่เป็นคำตอบคือ 3 จากช่วงของคำตอบที่ 0-7 ครั้งแรกเอาค่า (0+7)/2 = 4 มาเปรียบเทียบ ได้ผลว่า คำตอบที่ต้องการอยู่ในช่วงที่น้อยกว่า 4 ครั้งที่ 2 ก็เลือกค่า (0+4)/2 = 2 มาเปรียบเทียบ ได้ผลว่าคำตอบที่ต้องการอยู่ในช่วงที่มากกว่า 2 แต่น้อยกว่า 4 ครั้งที่ 3 ก็เลือกค่า (2+4)/2 = 3 มาเปรียบเทียบ ได้ผลว่าคำตอบที่ต้องการ จากหลักการที่กล่าวมาอาจเขียน flow chart ได้ดังรูปที่ 2.5



รูปที่ 2.5 flowchart แสดงหลักการของ "binary search" ในการหาคำตอบ

ข้อดีของวิธีนี้ คือ เวลาที่ใช้ในการหาคำตอบ n รอบ แน่นอน (สำหรับ n bit converter ซึ่งอ้างอิงได้ 2^n ระดับ และระดับ V_{in} ที่คงที่) ซึ่งใช้เวลาน้อยกว่าแบบ "Counting Algorithm"

แต่มีข้อเสีย คือถ้า V_{in} เปลี่ยนทันทีทันใด ขณะที่กำลังทำ binary search อยู่นั้น คำตอบที่ได้จะผิดพลาด ตัวอย่างเช่น เปลี่ยน V_{in} จาก 5 Volt เป็น 2 Volt



รูปที่ 2.6 กราฟแสดงหลักการ Binary Search

ช่วงของ V_{in} คือ 1-7 ใช้ n=3 (เพราะว่า 2^3=8)

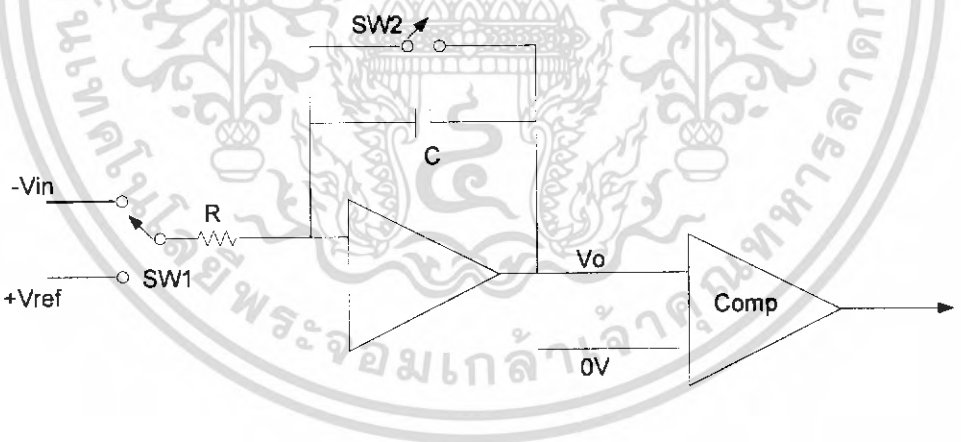
ครั้งแรก ใช้ 4 เปรียบเทียบกับ V_{in} (ซึ่งเท่ากับ 5 โวลต์) พบว่า อยู่ในช่วง lower ได้ 1xx

ครั้งที่ 2 ใช้ 2 เปรียบเทียบกับ V_{in} (ซึ่งเท่ากับ 5 โวลต์) พบว่า อยู่ในช่วง upper ได้ 10x

ครั้งที่ 3 ใช้ 3 เปรียบเทียบกับ V_{in} (ซึ่งเท่ากับ 5 โวลต์) พบว่า ผลลัพธ์ที่ได้จะผิดพลาด ได้ 100

Dual-Slope ADC

ใช้หลักการของวงจร Integrator ทำงานร่วมกับตัว Comparator ดังรูปที่ 2.7



รูปที่ 2.7 Dual Slope A/D converter

Input Voltage มี 2 ตัว คือ ค่าความต่างศักย์อะนาล็อกที่ต้องการแปลงเป็นดิจิตอล ($-V_{in}$) และความต่างศักย์ที่คงที่ค่าหนึ่ง (V_{ref}) และมีสวิตช์ SW1 ซึ่งทำหน้าที่เลือกค่าสัญญาณ

จากวงจรตอนเริ่มต้นสวิตช์ SW2 ทำหน้าที่คายประจุของตัวเก็บประจุ C แล้วจึงเปิด SW2 ออก

เมื่อสวิตช์ SW1 สับมาที่ $-V_{in}$ จากวงจร Integrator จะพิสูจน์สมการได้ดังนี้

$$I = C\frac{dV_o}{dt}$$

$$-V_{in} + iR - V_o + V_o = 0$$

$$-V_{in} + RC\frac{dV_o}{dt} = 0$$

$$V_{in} = RC\frac{dV_o}{dt}$$

$$\int dV_o = \int \frac{V_{in}}{RC} dx$$

$$V_o = \frac{V_{in}(t)}{RC}$$

slope มีค่าเท่ากับ $\frac{V_{in}}{RC}$

ค่า t ที่ใช้มีค่าคงที่ t_m

เมื่อ t เพิ่มจากศูนย์ถึง t_m ให้SW1 สับไปที่ V_{ref}

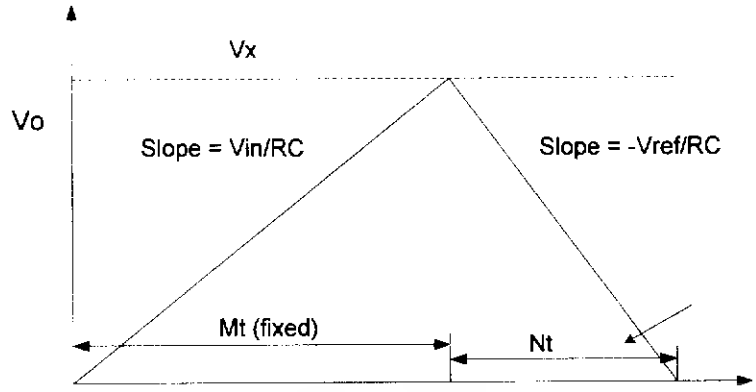
จะได้ดังสมการที่ (2.9)

$$V_o = \frac{V_{ref}(t)}{RC} \qquad (2.9)$$

slope มีค่า $\frac{V_{ref}}{RC}$

สมมติ ช่วงเวลาตั้งแต่ความต่างศักย์ที่ t_m จนความต่างศักย์เป็น 0 มีค่าเท่ากับ t_n

ได้ดังแสดงในกราฟรูปที่ 2.8



รูปที่ 2.8 กราฟ Dual Slope A/D Converter Output and Timing

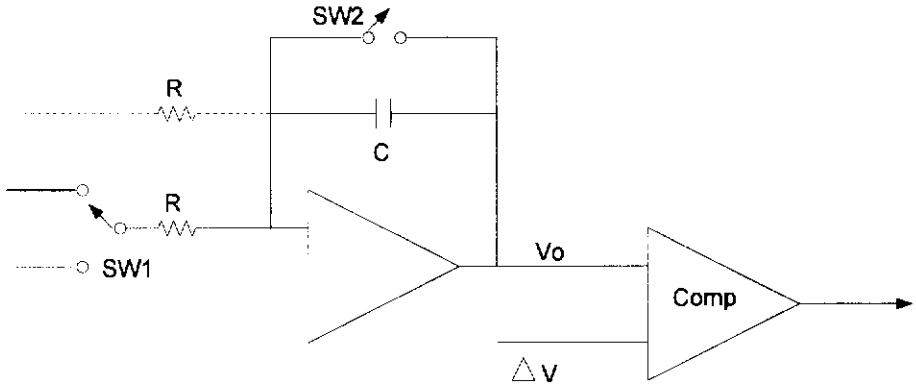
จากหลักของสามเหลี่ยมคล้าย จะได้ดังสมการที่ (2.10)

$$V_{in} = V_{ref} \frac{t_n}{t_m} \qquad (2.10)$$

เนื่องจาก V_{ref} และ t_n มีค่าคงที่ สัญญาณอะนาล็อกขึ้นกับค่า t_n เพราะการควบคุมการเปลี่ยนสัญญาณดิจิตอล ที่ขึ้นกับค่า t_n

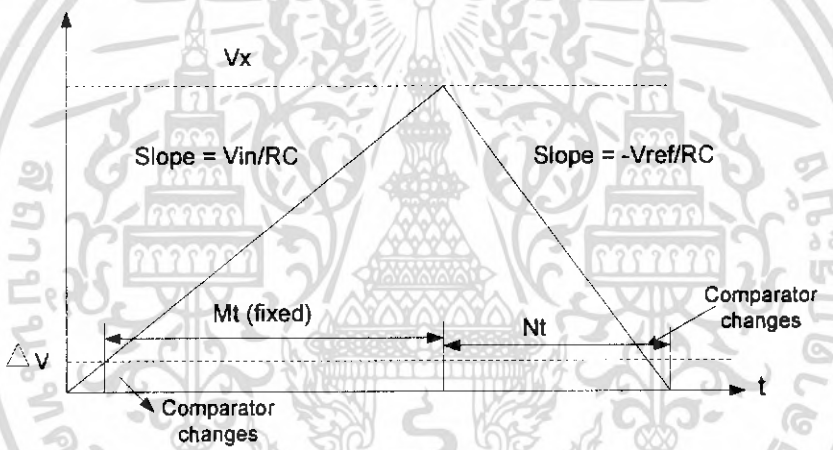
การแปลงเป็นสัญญาณดิจิตอลจะทำโดยจับคู่ค่า t_n กับเอาท์พุทค่าๆ หนึ่ง ตามความเหมาะสม สำหรับ V_{ref} นั้นๆ เหมือนการเทียบค่าในตาราง

ความเร็วของการแปลงสัญญาณแบบนี้ ขึ้นอยู่กับ V_{in} และ Slope ของวงจร integrator โดยธรรมชาติแล้ว ลักษณะของตัวเปรียบเทียบเองนั้น จะไม่เป็นอุดมคติ คือจะมีผลต่างของความต่างศักย์อยู่ V โวลต์ แม้ว่าจะต่ออินพุททั้งสองลงกราวด์แล้วก็ตาม ซึ่งถ้า V_{ref} ที่ใช้อยู่นั้นมีค่าน้อยกว่าค่าผลต่างของความต่างศักย์ที่เกิดจากตัวเปรียบเทียบ ความชันก็จะน้อย ทำให้เวลา t_m ใช้เวลานานมาก กว่าที่จะพ้นค่าความต่างศักย์ที่เกิดจากตัวเปรียบเทียบ เราจึงต้องนำค่าความต่างศักย์มาเพิ่มให้กับ V_{ref} เพื่อหาผลลัพธ์ ดังรูปที่ 2.9



รูปที่ 2.9 Dual Slope A/D Converter – Full Circuit

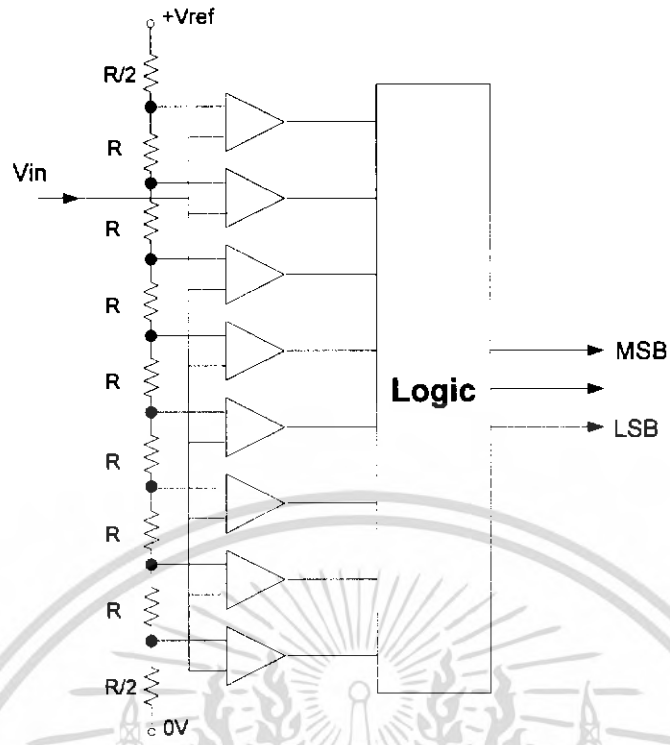
จากวงจรดังกล่าวทำให้ได้กราฟดังรูปที่ 2.10



รูปที่ 2.10 Dual Slope A/D Converter – Zero Offset

Flash Converter

หลักการของ Flash Converter คือการใช้การแบ่งแรงดันเป็น Voltage หลายๆ ค่า แล้วเปรียบเทียบกับ V_{in} เป็นคู่ๆ พร้อมกัน แล้วกระทำการทาง logic มี Voltage เปรียบเทียบ 8 bit ดังรูปที่ 2.11

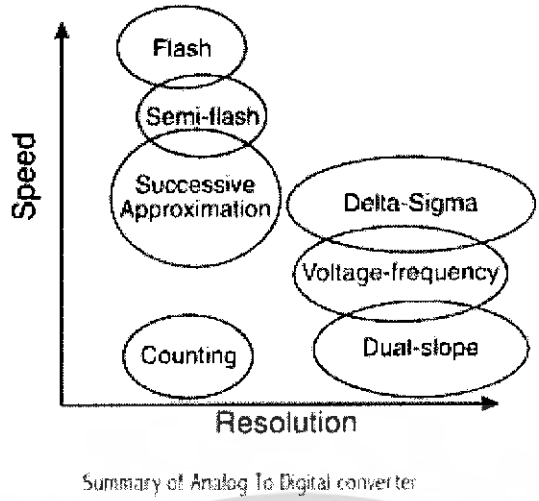


รูปที่ 2.11 Flash Converter

ค่าความต่างศักย์จะเพิ่มขึ้นเรื่อยๆ จากค่าความต้านทานที่ต่อเพิ่มขึ้น ความต่างศักย์ที่ได้นั้น เมื่อนำไปเปรียบเทียบกับ V_{in} แล้วมากกว่าก็จะปล่อยลอจิกออกมา ถ้ามากกว่าก็จะให้ลอจิก 1 ถ้าน้อยกว่าหรือเท่ากันก็จะให้ลอจิก 0 วิธี Flash Converter นี้จะเร็วที่สุด แต่ใช้อุปกรณ์ทาง Hardware มากกว่าแบบอื่นๆ

การแปลงสัญญาณอนาล็อก เป็นสัญญาณดิจิตอล มีประโยชน์มากในการควบคุมอุปกรณ์สวิตชิ่ง ซึ่งมีลักษณะการแปลงสัญญาณได้หลายวิธี แต่ละวิธีจะมีอัลกอริทึม ความรวดเร็วในการทำงาน และการใช้อุปกรณ์ฮาร์ดแวร์ต่างกันด้วย ทำให้ขนาด และราคาต่างกัน ขึ้นกับความต้องการของผู้ใช้ที่จะต้องเลือกให้เหมาะสมกับงานที่ใช้ และงบประมาณที่มีอยู่

ลำดับของความเร็ว และความละเอียดของอัลกอริทึมต่างๆ เป็นดังรูปที่ 2.12



รูปที่ 2.12 ลำดับของความเร็ว และความละเอียดของอัลกอริทึมต่างๆ

2.2.2 การแปลงดิจิตอลเป็นอะนาล็อก (Digital to Analog Conversion)

การแปลงดิจิตอลเป็นอะนาล็อก จะใช้วงจรหรืออุปกรณ์ที่ทำหน้าที่แปลงสัญญาณดิจิตอลซึ่งอาจจะเป็นแรงดันหรือกระแส ให้เป็นสัญญาณอะนาล็อกที่เป็นสัดส่วนกับสัญญาณดิจิตอลที่ป้อนเข้าไปเป็นอินพุทของวงจร เราสามารถเขียนสมการเอาท์พุทของการแปลงดิจิตอลเป็นอะนาล็อก ได้ดังนี้

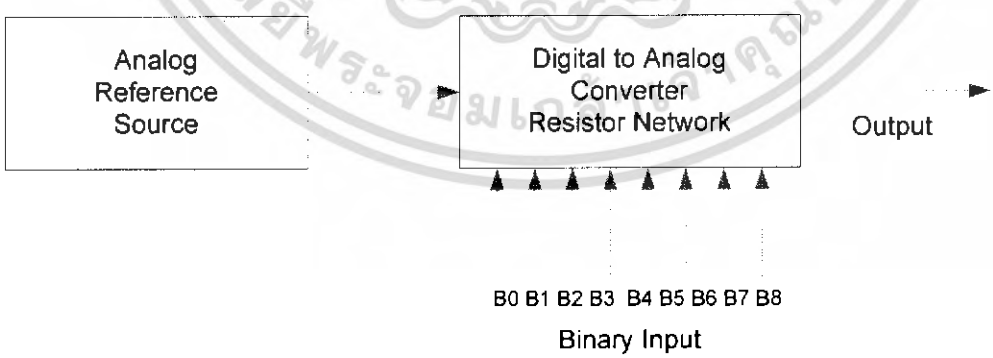
$$X = K \times A \times B$$

โดยที่ X = เป็นค่าแรงดันหรือกระแสทางด้านเอาท์พุท (อะนาล็อก)

A = ค่าอ้างอิงอะนาล็อก (เป็นแรงดันหรือกระแสก็ได้)

B = จำนวน (ค่า) ของตัวเลข Binary

K = ค่าคงที่จะมีค่าเป็น 1 เสมอ



รูปที่ 2.13 แสดงส่วนประกอบพื้นฐานของการแปลงดิจิตอลเป็นอะนาล็อก

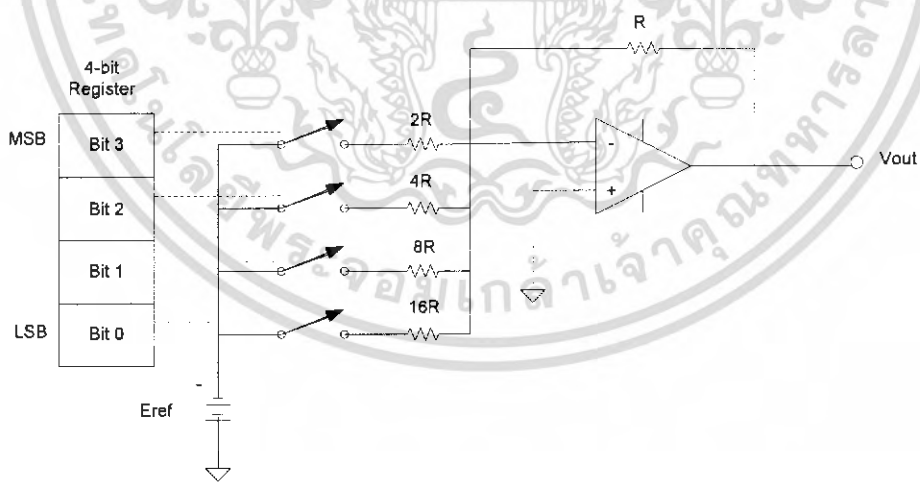
จากรูปที่ 2.13 แสดงส่วนประกอบต่างๆของระบบการแปลงดิจิตอลเป็นอะนาล็อก โดยที่แหล่งอ้างอิงอะนาล็อก จะมีค่าคงที่อยู่ค่าหนึ่ง อาจจะเป็นแหล่งสายแรงดันหรือกระแสที่ความเที่ยงตรงสูง จุดประสงค์เพื่อใช้เปรียบเทียบอินพุทที่จะสร้างแรงดันหรือกระแสที่เอาท์พุท

การแปลงดิจิตอลเป็นอะนาล็อกจะแบ่งตามการใช้ตัวต้านทานซึ่งจะมีการใช้ตัวต้านทานโดยทั่วๆไปอยู่ 2 ลักษณะ คือ แบบใช้ตัวต้านทานแบบถ่วงน้ำหนัก (Binary Weighted Register Ladder) และแบบใช้ตัวต้านทานขั้นบันได (R -2R Ladder)

การแปลงดิจิตอลเป็นอะนาล็อกแบบใช้ตัวต้านทานแบ่งน้ำหนัก (Binary Weighted Register Ladder)

การใช้ตัวต้านทานแบบนี้จะเป็นการแปลงข้อมูลดิจิตอลให้เป็นอะนาล็อกโดยตรง จะมีตัวต้านทานอนุกรมอยู่กับแรงดันอ้างอิง (V_{ref}) โดยจะมีอิเล็กทรอนิกส์สวิตช์ใช้ในการเปิดปิดสัญญาณตามสถานะของสัญญาณดิจิตอลซึ่งมีได้ 2 ระดับ คือ ลอจิก 0 กับ 1 การต่อลักษณะนี้ค่าของตัวต้านทานจะมีค่าในแต่ละน้ำหนักที่คูณด้วย 2 ตลอด คือ จะมีตั้งแต่ R,2R,4R,...จนถึง nR หรือเขียนเป็นสมการได้ว่า $2^{(n-1)}$ R โดยที่ n คือจำนวนบิตของไบนารี่ที่ทำการแปลง

เมื่อตัวต้านทานถูกต่อลงกราวนด์จะไม่มีกระแสไหลผ่านตัวต้านทานที่ต่ออยู่ แต่ถ้าตัวต้านทานตัวนั้นต่ออยู่กับแรงดันอ้างอิง(แทนด้วยระดับลอจิกที่ตรงข้ามกับการต่อตัวต้านทานกราวนด์) จะมีกระแสไหลผ่านตัวต้านทานตัวนั้น ดังนั้นถ้าตัวต้านทานค่า R และ 2R ถูกต่อกับแรงดันอ้างอิงพร้อมกัน ก็จะไม่มีกระแสไหลผ่าน ตัวต้านทาน R มีค่าเท่ากับ V_{ref} /R และกระแสไหลผ่านสองอัน มีค่าเท่ากับ V_{ref} /2R ถ้ากระแสทั้งสองไหลมารวมกันที่จุดผสม (Summing Point) ดังรูปที่ 2.14



รูปที่ 2.14 แสดงวงจรD/A Converter แบบ แบ่งน้ำหนัก

จากรูปที่ 2.14 สามารถเขียนสมการได้ดังสมการที่ (2.11)

$$I_{out} = \frac{V_{ref}}{R}[\sum_{i=1}^{n} \frac{b_i}{2^{(i-1)}}] \quad (2.11)$$

โดยที่ I_{out} = กระแสเอาท์พุท (สัญญานอะนาล็อก)มีหน่วยเป็นแอมแปร์ (A)

b_i = ค่าตัวเลขไบนารี่อินพุท (0 หรือ 1)

R = ค่าตัวต้านทานตัวแรกที่มีค่าต่ำที่สุด (เป็นค่า R ของ บิตในระยะสำคัญสูงสุด (MSB - Most Significant Bit)

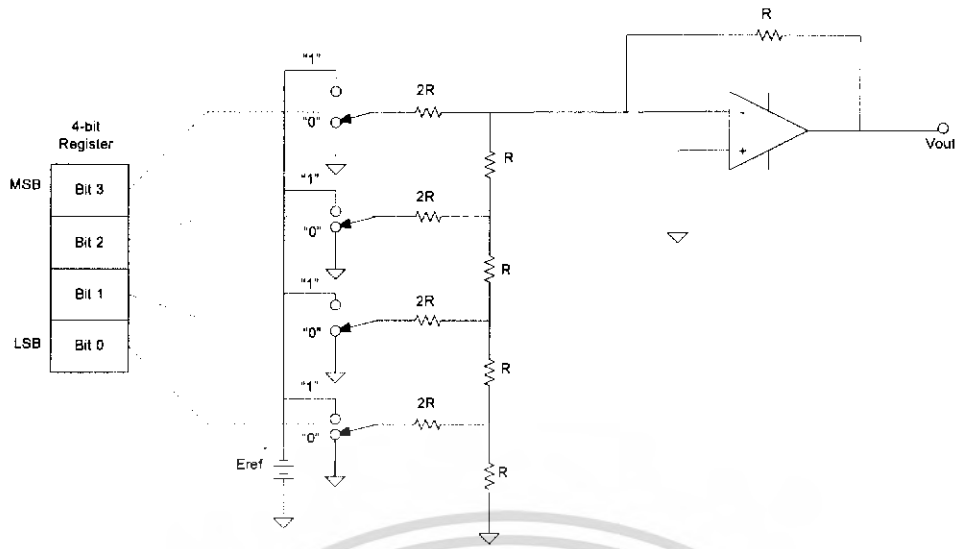
จากรูปที่ 2.15 จะเห็นได้ว่า ตัวต้านทานที่ทำให้กระแสไหลได้สูงสุดคือ R ดังนั้น ตัวต้านทาน R จะเป็นตัวต้านทานของบิตในนัยสำคัญสูงสุดและยังเห็นได้ว่า ค่าของตัวต้านทานเพิ่มขึ้นทีละมากๆ เช่น ถ้าหากมีการแปลงสัญญานดิจิตอลขนาด 8 บิต แล้วถ้าใช้ค่า R = 10 $k\Omega$ แล้วตัวต้านทานตัวที่ 8 จะมีค่าเท่ากับ $2^{(8-1)}R = 28R = 128 \times 10\ k\Omega = 1280\ k\Omega$ หรือ 1.28 $M\Omega$ จะทำให้มีกระแสไหลผ่านตัวต้านทานตัวนี้น้อยมากและจากธรรมชาติที่ว่า ตัวต้านทานค่ามากจะสร้างได้ยาก และยังมีผลกระทบจากสิ่งแวดล้อมภายนอกอีก คือ ความร้อนจะทำให้ค่าความต้านทานเปลี่ยนไป (โดยเฉพาะตัวต้านทานที่มีค่ามากๆ)ทำให้ความละเอียดและความแม่นยำของการแปลง D/A แบบนี้ลดลง ดังนั้น จากข้อจำกัดที่ว่าเมื่อมีจำนวนบิตสูงขึ้น จะทำให้ต้องใช้ตัวต้านทานที่มีค่ามากๆซึ่งหาได้ยากและสร้างได้ยากรวมถึงยังมีผลกระทบต่อความแม่นยำในการแปลง เพราะฉะนั้น เมื่อมีจำนวนบิตสูงขึ้น จึงได้มีการเปลี่ยนไปใช้แบบตัวต้านทานแบบขั้นบันได ซึ่งจะมีการใช้ตัวต้านทานเพียงสองค่าเท่านั้น ซึ่งเป็นการขจัดปัญหาที่กล่าวมาข้างต้นได้ ดังจะกล่าวในหัวข้อถัดไป

การแปลงดิจิตอลเป็นอะนาล็อกแบบตัวต้านทานขั้นบันได (R – 2R Ladder)

จะมีรูปวงจรดังแสดงใน 2.15โดยที่การใช้คล้ายการแปลง D/A แบบ R – 2R Ladder นี้ ใช้แก้ปัญหาของแบบแรกที่ต้องใช้ความต้านทานหลายค่า ซึ่งอาจจะหาค่าได้ไม่ตรงกับที่ต้องการมาใช้ตัวต้านทานเพียงสองค่าเท่านั้น ในการแปลงแบบนี้คือค่า R และ 2R ดังนั้นจากรูปที่ 2.15 วงจรสามารถเขียนสมการได้ดังสมการที่ (2.12)

$$I_{out} = [\sum_{i=1}^{n} \frac{b_i}{2^{(i-1)}}] \quad (2.12)$$

คือเป็นสมการแบบเดียวกับแบบใช้ตัวต้านทานแบ่งน้ำหนัก



รูปที่ 2.15 แสดงวงจรการแปลงดิจิตอลเป็นอะนาล็อกแบบตัวต้านทานขั้นบันได

คุณสมบัติและข้อกำหนดของตัวแปลง D/A (D/A characteristics and specification)

สำหรับตัวแปลงสัญญานดิจิตอลเป็นอะนาล็อกจะมีคุณสมบัติสำคัญดังต่อไปนี้

1.ความละเอียด (Resolution)

คุณสมบัติประการแรกของ D/A นี้คือความละเอียดของความสามารถในการเปลี่ยนแปลง พิจารณาบิตเลขฐาน 2 ที่ป้อนเข้าที่อินพุทของตัวแปร D/A ถ้าอินพุทเป็นเลขฐาน 2 แปดตัว จะแสดงว่า มีระดับของเอาท์พุทที่เป็นไปได้ เท่ากับ $2^8 = 256$ ดังนั้น ค่าความละเอียดของ D/A ตัวนี้ คือ 1 ใน 256 ในบางครั้งอาจจะกล่าวในรูปของเปอร์เซ็นต์ เช่น ความละเอียดของ D/A ขนาด 8 บิต คือประมาณ 0.39 %

2.ค่าเต็มพิกัดของแรงดันเอาท์พุท (Full Scale Output Voltage)

คุณสมบัติประการที่สองของ D/A คือค่าเต็มพิกัดของแรงดันเอาท์พุท คือค่าแรงดันสูงสุดที่ D/A สามารถให้ออกมาได้เมื่ออินพุทดิจิตอลมีค่าลอจิกเป็น 1 หมดทุกบิต

3.ความเที่ยงตรง (Accuracy)

ข้อกำหนดความเที่ยงตรงสำหรับ D/A คือการเปรียบเทียบระหว่าง เอ้าท์พุทที่เกิดขึ้นจริงกับเอาท์พุทที่คาดหวังไว้ (มาจากการคำนวณ) โดยที่จะมีการกำหนดค่าไว้เป็นเปอร์เซ็นต์ เช่น D/A มีแรงดันเต็มพิกัดของเอาท์พุท 10 โวลต์และความเที่ยงตรง 0.2 เปอร์เซ็นต์ ดังนั้นค่าผิดพลาดสูงสุดของเอาท์พุทใดๆเป็น $10V \times 0.002 = 20$ mV สำหรับ D/A ในอุดมคตินั้น จะมีค่าความผิดพลาดสูงสุดไม่มากกว่า 1/2 ของบิตนัยสำคัญต่ำสุด (LSB – Least Significant Bit)

4.ความเป็นเชิงเส้น (Linearity)

คือการวัดค่าความเบี่ยงเบนของเอาท์พุทเป็นพิสัยจากเส้นตรงที่ตัวแปลงสัญญาณไม่มีการทำงานเลย ไปจนถึงทำงานครบทุกบิตซึ่งค่าเบี่ยงเบนในอุดมคติของเอาท์พุทจากเส้นตรงควรมีไม่มากกว่า 1/2 ของค่า LSB

5.ค่าเวลาในการแปลง (Setting Time)

คือเมื่อเกิดการเปลี่ยนแปลงค่าไบนารี่ที่ทำการป้อนกับเป็นอินพุทเข้ามา ในแต่ละบิตของ D/A เอาท์พุทจะเปลี่ยนแปลงเป็นสัญญาณอะนาล็อกค่าใหม่ได้ถูกต้อง จะต้องใช้เวลาค่าหนึ่งซึ่งขึ้นกับข้อกำหนดของเวลาในการแปลงของตัว D/A ค่าเวลาที่เอาท์พุทใช้ไปภายใน 1/2 LSBของค่าสุดท้ายจะเรียกว่า ค่าเวลาในการแปลง สำหรับค่าการเปลี่ยนแปลงเอาท์พุทเต็มพิกัด คุณสมบัตินี้สำคัญเพราะถ้าตัวแปลงสัญญาณทำงานที่ความถี่สูงๆมันอาจจะไม่มีเวลาจัดตั้งค่าก่อนที่มันจะสวิตช์ไปสู่อีกสถานะหนึ่ง

2.3 ไมโครคอนโทรลเลอร์ (Microcontroller)

ไมโครคอนโทรลเลอร์ เป็นอุปกรณ์ไอซี (IC: Integrated Circuit) ที่สามารถโปรแกรมการทำงานได้หลายครั้งสามารถรับข้อมูลในรูปสัญญาณดิจิตอลเข้าไปทำการประมวลผลแล้วส่งผลลัพท์ข้อมูลดิจิตอลออกมาเพื่อนำไปใช้งานตามที่ต้องการได้

ไมโครคอนโทรลเลอร์หรืออาจจะเรียกได้ว่าไมโครโพรเซสเซอร์ชิปเดี่ยว(Single-Chip Microprocessor) เป็นไมโครโพรเซสเซอร์ชนิดหนึ่ง เช่นเดียวกับหน่วยประมวลผลกลาง (CPU: Central Processing Unit) ที่ใช้ในคอมพิวเตอร์ แต่ได้รับการพัฒนาแยกออกมาภายหลังเพื่อนำไปใช้ในวงจรทางด้านงานควบคุม คือ แทนที่ในการใช้งานจะต้องต่อวงจรภายนอกต่าง ๆ เพิ่มเติมเช่นเดียวกับไมโครโพรเซสเซอร์ ก็จะทำการรวมวงจรที่จำเป็น เช่น หน่วยความจำ, ส่วนอินพุต/เอาท์พุต บางส่วนเข้าไปในตัวไอซีเดียวกัน และเพิ่มวงจรบางอย่างเข้าไปด้วยเพื่อให้มีความสามารถเหมาะกับการใช้ในงานควบคุม เช่น วงจรตั้งเวลา วงจรการสื่อสารอนุกรม เป็นต้น ดังนั้นไมโครคอนโทรลเลอร์สามารถจะทำงานได้เสมือนกับเป็นคอมพิวเตอร์เล็กๆเครื่องหนึ่ง กล่าวโดยสรุปคือ

Microcontroller = Microprocessor + Memory + I/O

ปัจจุบันไมโครคอนโทรลเลอร์ถูกนำไปใช้อย่างกว้างขวาง โดยมักจะเป็นการนำไปใช้ฝังในระบบของอุปกรณ์อื่น ๆ (Embedded Systems) เพื่อใช้ควบคุมการทำงานบางอย่างเช่น ใช้ในรถยนต์, เครื่องปรับอากาศ, เครื่องซักผ้าอัตโนมัติ เป็นต้น เพราะว่าไมโครคอนโทรลเลอร์มีข้อดีเหมาะสมต่อการใช้ในงานควบคุมหลายประการ เช่น

- ไอซี และระบบที่ได้มีขนาดเล็ก
- ระบบที่ได้มีราคาถูกกว่าการใช้ชิปไมโครโพรเซสเซอร์
- วงจรที่ได้จะมีความซับซ้อนน้อย ช่วยลดข้อผิดพลาดที่อาจจะเกิดขึ้นได้ในการต่อวงจร
- มีคุณสมบัติเพิ่มเติมสำหรับงานควบคุมโดยเฉพาะซึ่งใช้งานได้ง่าย
- ช่วยลดระยะเวลาในการพัฒนาระบบได้

ไมโครคอนโทรลเลอร์มีหลายยี่ห้อ หลายตระกูล และหลายเบอร์ด้วยกัน ซึ่งแต่ละเบอร์ก็จะมีโครงสร้างภายในและความสามารถในการทำงานที่แตกต่างกัน ทำให้เลือกใช้งานได้อย่างเหมาะสม

2.3.1 ไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีด้วยกันหลายเบอร์ขึ้นกับโครงสร้างภายในของมัน บางเบอร์จะมีหน่วยความจำภายในเป็นแบบ ROM บางเบอร์เป็นแบบ EPROM บางเบอร์มี RAM ภายใน 128 ไบต์ บางเบอร์มี 256 ไบต์ เป็นต้น ซึ่งรายละเอียดสามารถศึกษาได้จากคู่มือได้โดยตรง และลักษณะของขาต่าง ๆ จะเหมือนกัน

คุณสมบัติที่สำคัญของ MCS-51 มีดังนี้

- มีหน่วยความจำ ROM 4 Kbytes, 8 Kbytes, 20 Kbytes
- มีหน่วยความจำ RAM 128 byte
- มีพอร์ท I/O ขนาด 8 บิต 4 พอร์ท
- มี Timer 16 บิต 2 ตัว
- สามารถอินเทอร์รัพท์ได้ 5 แหล่ง
- มีวงจรออสซิลเลเตอร์และวงจรนาฬิกาบนชิพ
- มีพอร์ทอนุกรมที่สามารถรับส่งข้อมูลแบบ Full Duplex ด้วยความเร็วสูง
- อ้างหน่วยความจำโปรแกรมภายนอกได้ 64K
- อ้างหน่วยความจำข้อมูลภายนอกได้ 64K
- สามารถประมวลผลทีละบิตได้
- สามารถอ้างหน่วยความจำแบบบิตได้ 210 ตำแหน่ง
- หนึ่งวัฏจักรคำสั่งกินเวลาประมาณ 1 ไมโครวินาที ขณะทำงานด้วย Clock 11.0592 MHz

ไมโครคอนโทรลเลอร์ MCS-51 จะมีชุดคำสั่ง (Instruction Set) อยู่จำนวนหนึ่ง สำหรับสั่งงานให้ทำงานต่าง ๆ และเนื่องจาก MCS-51 จะประมวลผลแบบ 8 บิต รหัสภาษาเครื่องจะมีขนาด 8 บิตด้วย ซึ่งชุดคำสั่งจะมีได้จำนวนสูงสุด 28 = 256 ชุดคำสั่ง คำสั่งแต่ละคำสั่งอาจมีขนาด 1, 2 หรือ 3 ไบต์

MCS-51 มีโหมดการอ้างแอดเดรส (Addressing Modes) เพื่อติดต่อกับหน่วยความจำซึ่งอาจจะเป็นการอ่านข้อมูลออกมาหรือเขียนข้อมูลเข้าไปได้ทั้งหมด 8 โหมดคือ Register, Direct, Indirect, Immediate, Relative, Absolute และ Index

ใน MCS-51 จะแบ่งชุดของคำสั่งออกได้ 5 ประเภท ได้แก่

1. Arithmetic Instructions เป็นกลุ่มคำสั่งที่ทำงานด้านคณิตศาสตร์ เช่น ADD, SUBB, INC, DIV เป็นต้น

2. Logical Instructions มีลักษณะการทำงานคล้ายกับ Boolean Operation ซึ่งสามารถกระทำแบบไบต์ต่อไบต์ หรือ บิตต่อบิตได้ เช่น ANL, ORL เป็นต้น

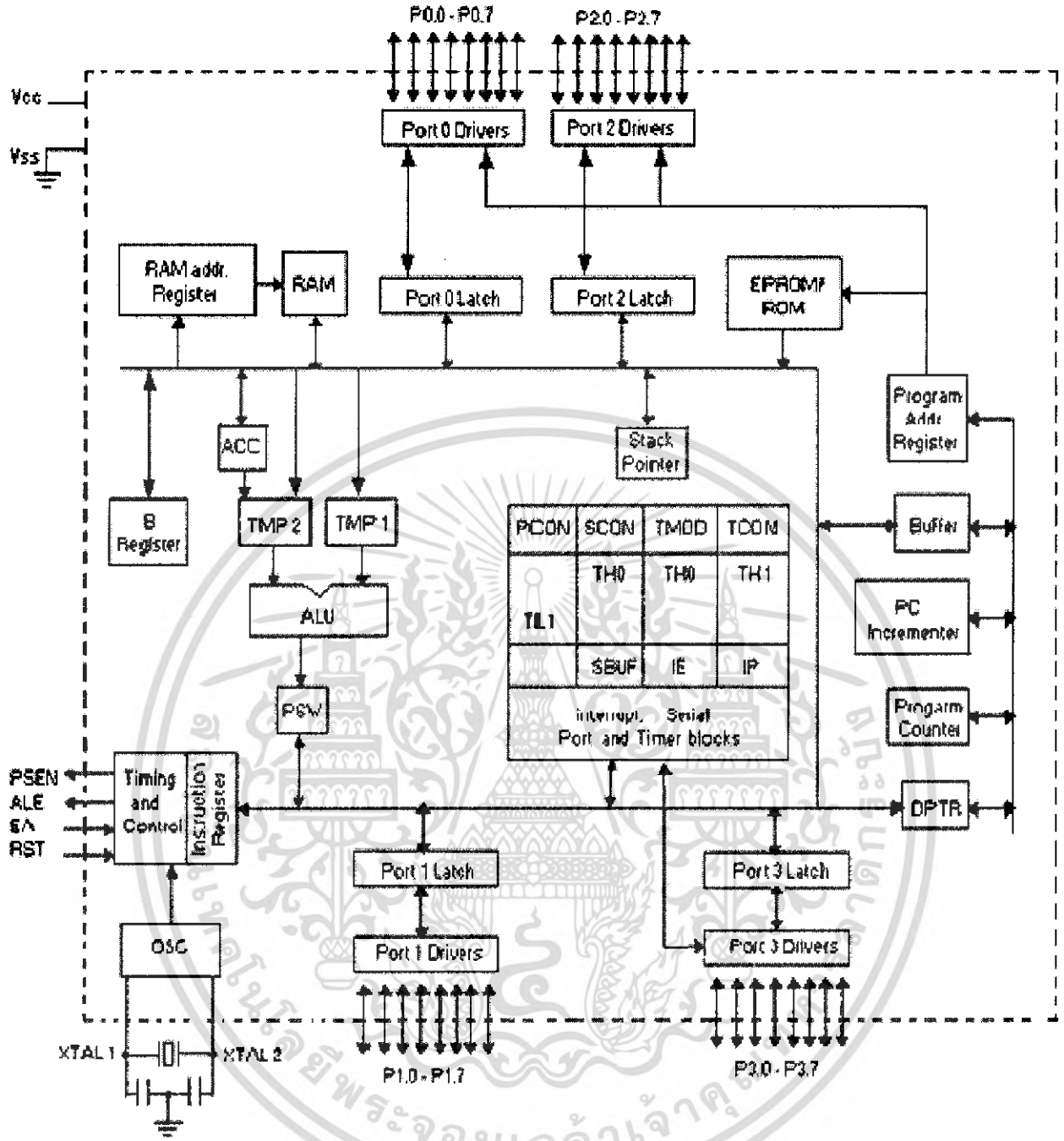
3. Data Transfer Instructions เป็นกลุ่มคำสั่งที่ใช้ในการเคลื่อนย้าย คัดลอกข้อมูลซึ่งสามารถติดต่อกับหน่วยความจำได้หลายแบบ เช่น MOV, XCH, XCHD เป็นต้น

4. Boolean Instructions เช่น ANL, ORL, CLR, SETB เป็นต้น

5. Program Branching เป็นกลุ่มคำสั่งสำหรับสั่งให้โปรแกรมกระโดดไปทำงานในตำแหน่งที่ต้องการ แบ่งเป็นกลุ่มย่อยได้ 2 กลุ่มคือ กระโดดแบบมีเงื่อนไข เช่น AJMP, LJMP, SJMP กับ กระโดดแบบมีเงื่อนไข เช่น JZ, JNZ, CJNE, DJNZ เป็นต้น

2.3.2 โครงสร้างภายในของไมโครคอนโทรลเลอร์ MCS – 51

วงจรภายในของไมโครคอนโทรลเลอร์ตระกูล MCS – 51 ประกอบไปด้วยวงจรพอร์ทอินพุทและเอาท์พุททั้งหมด 4 พอร์ท แต่ละพอร์ตจะเป็นแบบ 8 บิต หน่วยความจำภายในโปรแกรม (EPROM , EEPROM และ Flash) หน่วยความจำที่เป็นข้อมูล (RAM) ซึ่งรวมอยู่ในวงจรหลักของไมโครคอนโทรลเลอร์ตลอดจนวงจรการคำนวณทางคณิตศาสตร์และลอจิก (ALU) วงจรรีจิสเตอร์ทั่วไปและรีจิสเตอร์ฟังก์ชันการใช้งานเฉพาะ แสดงดังรูปที่ 2.16

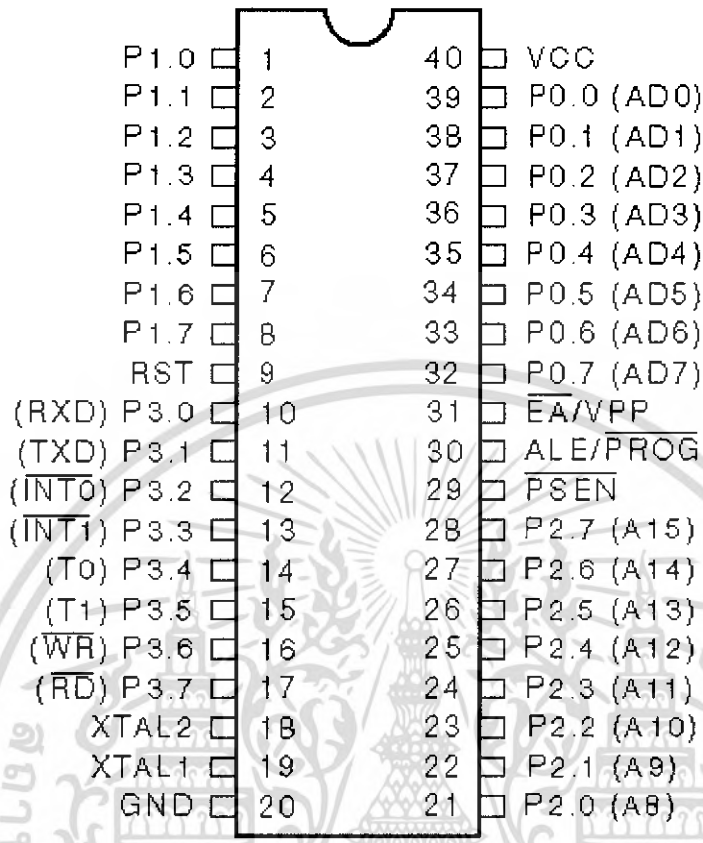


รูปที่ 2.16 แสดงโครงสร้างภายในของไมโครคอนโทรลเลอร์ MCS – 51

2.3. 3 ลักษณะการจัดขาของ MCS-51

การจัดตำแหน่งขาของไมโครคอนโทรลเลอร์ MCS – 51 ไมโครคอนโทรลเลอร์ MCS – 51 ทุกเบอร์จะมีโครงสร้างและการใช้งานพื้นฐานเหมือนกันตัวอย่างเช่น แบบดิป (DIP) ซึ่งมีทั้งหมด 40

ขา ได้แบ่งการใช้งานออกเป็นขาอินพุท / เอาท์พุท ขาสัญญาณควบคุม ขาสัญญาณกำหนดตำแหน่งหน่วยความจำ และขาสัญญาณข้อมูลดังรูปที่ 2.17



รูปที่ 2.17 แสดงตำแหน่งขาไมโครคอนโทรลเลอร์ MCS – 51

ตำแหน่งขาของไมโครคอนโทรลเลอร์ MCS – 51 และหน้าที่การทำงาน

1. P0.0 – P0.7 (ขาที่ 32 - 39) พอร์ท 0 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกได้ 2 ทิศทาง สามารถรับข้อมูลอินพุทและส่งข้อมูลเอาท์พุทได้ มีขนาด 8 บิต การตั้งค่าให้พอร์ท 0 รับข้อมูลอินพุท ทำได้โดยการส่งค่าสถานะ 1 ไปยังบิตที่ต้องการให้รับข้อมูลอินพุท วงจรภายในจะทำให้บิตนั้นมีค่าความต้านทานสูงและสามารถรับข้อมูลอินพุทได้ และยังใช้เป็นขาสัญญาณกำหนดตำแหน่งหน่วยความจำ (A_0 – A_7) และขาสัญญาณข้อมูล (D_0 – D_7) โดยการใช้ตัวแยกสัญญาณ (D – latch 74LS373) ทำหน้าที่เป็นมัลติเพล็กซ์ (multiplex) โดยเลือกช่วงเวลาของสัญญาณกำหนดตำแหน่งหน่วยความจำและสัญญาณข้อมูลออกจากกัน

ในขณะที่ใช้เป็นพอร์ตอินพุทและเอาท์พุท วงจรภายในจะไม่มีวงจรเพิ่มกระแสไฟฟ้า (Pull Up) จึงจำเป็นต้องต่อวงจรเพิ่มกระแสไฟฟ้าภายนอกเข้าไป

2. P1.0 – P1.7 (ขาที่ 1 - 8) พอร์ท 1 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกได้ 2 ทิศทาง สามารถเป็นได้ทั้งอินพุทและเอาท์พุท มีขนาด 8 บิต สามารถอ้างถึงการทำงานได้ทีละบิต และ

วงจรภายในมีตัวต้านทานเพิ่มกระแสไฟฟ้า (Pull Up) ในกรณีที่ต้องการให้รับข้อมูลอินพุตก็สามารถทำได้เหมือนพอร์ท 0

3. P2.0 – P2.7 (ขาที่ 21 -28) พอร์ท 2 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกได้ 2 ทิศทางเป็นได้ทั้งอินพุทและเอาท์พุท มีขนาด 8 บิต สามารถใช้เป็นขาสัญญาณกำหนดตำแหน่งหน่วยความจำ (A_8 – A_{15}) และมีวงจรเพิ่มกระแสภายใน การกำหนดให้เป็นขาอินพุตทำได้โดยการส่งค่าข้อมูลสถานะ 1 ไปยังบิตที่ต้องการให้เป็นอินพุท ก็จะสามารถรับค่าข้อมูลอินพุทได้

4. P3.0 – P3.7 (ขาที่ 10 -17) พอร์ท 3 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกอินพุทและเอาท์พุท 2 ทิศทาง มีขนาด 8 บิต คุณสมบัติทั่วไปจะเหมือนกับพอร์ทอื่นๆ แต่จะมีคุณสมบัติที่ต่างออกไป คือ ใช้ทำหน้าที่พิเศษเป็นสัญญาณควบคุมการทำงานต่างๆ ของไมโครคอนโทรลเลอร์ ดังตารางที่ 2.2

ตารางที่ 2.2 แสดงสัญญาณควบคุมการทำงานต่างๆ ของไมโครคอนโทรลเลอร์

บิตของพอร์ต	สัญญาณ	หน้าที่การทำงาน
P3.0	RXD	รับข้อมูลจากพอร์ตอนุกรม (serial input port)
P3.1	TXD	ส่งข้อมูลจากพอร์ตอนุกรม (serial output port)
P3.2	INT0	รับสัญญาณอินเตอร์รัปต์หมายเลข 0 (external interrupt 0)
P3.3	INT1	รับสัญญาณอินเตอร์รัปต์หมายเลข 1 (external interrupt 1)
P3.4	T0	ใช้ตั้งเวลา / นับเวลาตัวที่ 0 (Timer 0 external input)
P3.5	T1	ใช้ตั้งเวลา / นับเวลาตัวที่ 1 (Timer 1 external input)
P3.6	WR	เป็นสัญญาณเขียนข้อมูลหน่วยความจำหรืออุปกรณ์ภายนอก (external data memory write strobe)
P3.7	RD	เป็นสัญญาณเขียนข้อมูลหน่วยความจำหรืออุปกรณ์ภายนอก (external data memory read strobe)

2.3.4 พอร์ทอินพุทและพอร์ทเอาท์พุท

1. การใช้งานเป็นพอร์ทอินพุต

เนื่องจากพอร์ททั้งหมดของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลชสามารถเป็นได้ทั้งอินพุทและเอาท์พุท ดังนั้นจึงมีความจำเป็นอย่างยิ่งต้องทำความเข้าใจถึงการกำหนดลักษณะการทำงานให้แก่พอร์ทของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลช

ในการกำหนดให้เป็นพอร์ทอินพุท ต้องเริ่มต้นด้วยการเขียนข้อมูล "1" มาที่แต่ละบิตของพอร์ทที่ต้องการใช้งานเป็นอินพุท เพื่อหยุดการทำงานของเฟตที่ใช้ในการขับสัญญาณเอาท์พุทของบิตนั้นๆ ทำให้ขาสัญญาณของพอร์ทเชื่อมต่อกับเข้ากับวงจรพูลอัพภายในโดยตรง ส่งผลให้ขาพอร์ทนั้นมีลอจิกเป็น "1" สามารถรับสัญญาณลอจิก "0" จากอุปกรณ์ภายนอกได้ง่าย สัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ท แล้วรอให้ซีพียูมาอ่านค่าเข้าไป เมื่อเป็นเช่นนี้ อุปกรณ์

ภายนอกที่เชื่อมต่อกับพอร์ทอินพุทของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลชควรกำหนดให้ทำงานในสภาวะลอจิก “0” จะดีและสะดวกที่สุด (ซึ่งในปัจจุบันอุปกรณ์อินพุทที่เชื่อมต่อไมโครคอนโทรลเลอร์แบบทั้งหมดทำงานที่ลอจิก “0” แล้ว)

2. การใช้งานเป็นพอร์ทเอาท์พุท

โดยปกติแล้ว ขาพอร์ทจะกำหนดให้มีลักษณะเป็นเอาท์พุทอยู่แล้ว ดังนั้นจึงสามารถส่งข้อมูลออกไปได้อย่างง่ายดายและตรงไปตรงมา กล่าวคือ เมื่อต้องการส่งข้อมูล “0” ออกไปทางเอาท์พุทก็ให้เขียนข้อมูล “0” ไปยังวงจรแลตช์ ซึ่งก็จะส่งต่อไปขับเฟต ทำให้เฟตทำงาน ที่ขาพอร์ตที่กำหนดให้ทำงานก็จะเกิดลอจิก “0” ขึ้น ในทางตรงข้ามหากต้องการส่งข้อมูล “1” ออกไป ก็ให้เขียนข้อมูล “1” ไปยังวงจรแลตช์ วงจรขับก็จะหยุดทำงาน ทำให้ที่ขาพอร์ทเชื่อมต่อกับวงจรพูลอัพภายในเกิดเป็นลอจิก “1” ที่ขาพอร์ทนั้น ซึ่งจะคล้ายกับการกำหนดให้เป็นขาอินพุทมาก เพียงแต่แตกต่างกันที่กระบวนการในการเคลื่อนย้ายข้อมูล โดยถ้าเป็นอินพุทจะมีสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่ถ้าเป็นเอาท์พุทจะไม่มีการอ่านข้อมูลที่บัฟเฟอร์แต่อย่างใด เว้นแต่ในกรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาท์พุท

เมื่อใช้งานเป็นพอร์ทเอาท์พุท แต่ละขา (หรือแต่ละบิต) ของแต่ละพอร์ทมีความสามารถในการจ่ายกระแสหรือที่เรียกว่า กระแสซอร์ต (Source current) ได้สูงสุด 500 μA ในขณะที่สามารถรับกระแสซิงค์ (sink current) เมื่อทำงานด้วยลอจิก “0” ได้สูงถึง 10 mA ต่อขาและทุกขารวมกันในแต่ละพอร์ท (ทั้ง 8 บิต) สูงสุด 26 mA สำหรับพอร์ท 0 และ 15 mA สำหรับพอร์ท 1 ถึง 3 ในกรณีที่ใช้งานทุกพอร์ทเอาท์พุทจะสามารถรับกระแสซิงก์ได้รวมกันสูงสุด 71 mA ดังนั้นในการใช้งานเป็นพอร์ทเอาท์พุทเพื่อไม่ให้เกิดปัญหาเกี่ยวกับความสามารถในการจ่ายกระแสจึงควรต่อวงจรบัฟเฟอร์ทางเอาท์พุตเพื่อช่วยในการขับกระแสอีกทางหนึ่ง

2.3.5 การใช้งานพอร์ทสื่อสารอนุกรมแบบ Single Processor

พอร์ทสื่อสารอนุกรม

พอร์ทสื่อสารอนุกรมมีโครงสร้างการทำงานในแบบที่เรียกว่า **ฟูลดูเพล็กซ์** (Full Duplex) สามารถรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน

- ทางด้านส่งใช้ขา TxD (พอร์ท 3.1)
- ทางด้านรับใช้ขา RxD (พอร์ท 3.0)

Serial Port Buffer (SBUF) ใช้เป็นบัฟเฟอร์สำหรับรับและส่งข้อมูลอนุกรมโดยมีอยู่ 2 ตัว

การส่งข้อมูล ข้อมูลที่จะส่งให้ใส่ใน SBUF โดยใช้คำสั่ง MOV SBUF,A โดยเตรียมข้อมูลที่จะส่งเข้า A ก่อน

การรับข้อมูล ข้อมูลที่รับได้จะอยู่ใน SBUF การถ่ายข้อมูลออกมาใช้คำสั่ง MOV A,SBUF แล้วจึงนำข้อมูลใน A ไปใช้

พอร์ทสื่อสารอนุกรมสามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM1 และ SM0 ซึ่งอยู่ในรีจิสเตอร์ควบคุม SCON การทำงานทั้ง 4 โหมด ของพอร์ทสื่อสารอนุกรม มีดังนี้

SM0, SM1 บิตเลือกโหมดการทำงาน

ตารางที่ 2.3 แสดง การเลือกโหมดการทำงานในการรับส่งข้อมูล

SM0	SM1	โหมด	การทำงาน
0	0	**0**	**Shift register** ความเร็วในการรับหรือส่งข้อมูลเท่ากับ **(1/12) ของ CPU OSC**
0	1	**1**	**8 Bit UART** ความเร็วในการรับหรือส่งข้อมูลกำหนดได้จาก Timer 1,2
1	0	**2**	**9 Bit UART** ความเร็วในการรับหรือส่งข้อมูล = (1/32) หรือ (1/64)เท่าของ**CPU OSC** โดยขึ้นกับบิต SMOD ใน PCON
1	1	**3**	**9 Bit UART** ความเร็วในการรับหรือส่งข้อมูลกำหนดที่ Timer 1, 2

REN (Receive Enable) **บิตควบคุมให้รับหรือไม่รับข้อมูล**

1 : ให้รับข้อมูลได้

0 : ห้ามรับข้อมูล

หมายเหตุ (การรับข้อมูลสามารถห้ามได้ แต่การส่งข้อมูลห้ามไม่ได้)

TI แฟลกซ์ TI จะเป็น 1 **เมื่อสิ้นสุดการส่งข้อมูล 1 ไบต์**

RI แฟลกซ์ RI จะเป็น 1 **เมื่อรับข้อมูลเสร็จ 1ไบต์** (บิต RI,TI ผู้เขียนโปรแกรมจะต้องเคลียร์เอง)

การเขียนโปรแกรมควบคุมการรับและส่งข้อมูลทำได้ 2 วิธี

- **การตรวจสอบบิต TI หรือ RI** โดยใช้คำสั่งตรวจสอบบิต เช่นใช้คำสั่ง **WAIT:JNB TI,WAIT**

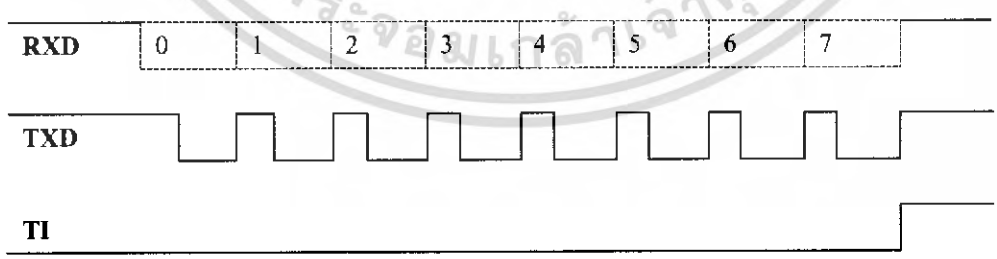
  คำสั่งนี้หมายความว่า ถ้า **TI = 0** ให้วนไปยังแอดเดรสชื่อ WAIT

  ถ้า **TI = 1** ถือว่าส่งข้อมูลเสร็จแล้วให้ทำคำสั่งถัดไป

- **การใช้อินเตอร์รัพต์ควบคุม**

โหมด 0 : พอร์ทสื่อสารอนุกรม 8 บิต โดยการส่งข้อมูลจะเลื่อนออกทีละบิตโดยส่งบิต D0 ออกไปก่อนทางขา RxD เนื่องจากไม่มีการส่ง Start bit แต่จะส่ง shift clock ทางขา TxD

[ความเร็ว (1/12) เท่าของ CPU Clock]

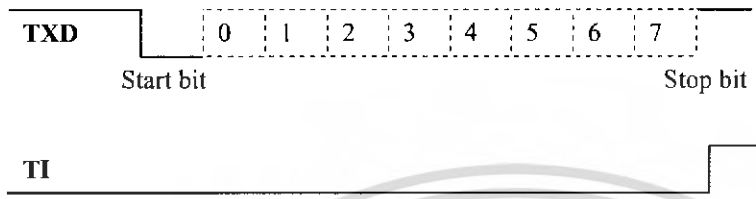


รูปที่ 2.18 แสดง Timing Diagram การส่งข้อมูลโหมด 0

โหมด 1 : พอร์ทสื่อสารอนุกรม 10 บิต ข้อมูล 8 บิต 1 start bit และ 1 stop bit และสามารถเปลี่ยนแปลงความเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิด SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1,2

$$\textbf{Baud Rate Mode 1,3} = \frac{2^{SMOD} \times \text{CPU OSC}}{32 \times 12 \times [256 - (TH1)]}$$ โดยใช้ Timer 1

$$\textbf{Baud Rate Mode 1,3} = \frac{\text{CPU OSC}}{32 \times [65536 - (RCAP2H,RCAP2L)]}$$ โดยใช้ Timer 2

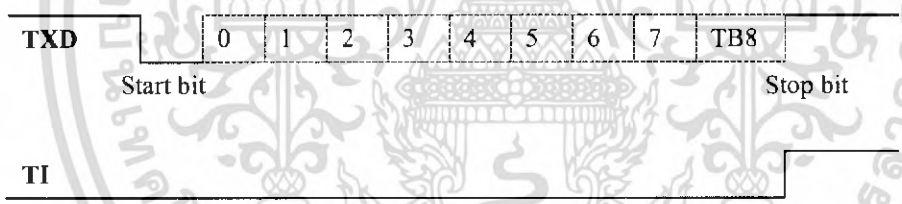


รูปที่ 2.19 แสดง Timing Diagram การส่งข้อมูลโหมด 1

โหมด 2 : พอร์ทสื่อสารอนุกรม 11 บิต ข้อมูล 9 บิต 1 start bit และ 1 stop bit (TB8 นิยมนำมาใช้ส่ง Parity bit) ความเร็วในการรับส่งข้อมูลเท่ากับ (1/32) หรือ (1/64) เท่าของ CPU OSC โดยขึ้นกับบิต SMOD ใน PCON

-Baud Rate (Mode 2) = (1/32) CPU OSC เมื่อ SMOD = 1

-Baud Rate (Mode 2) = (1/64) CPU OSC เมื่อ SMOD = 0

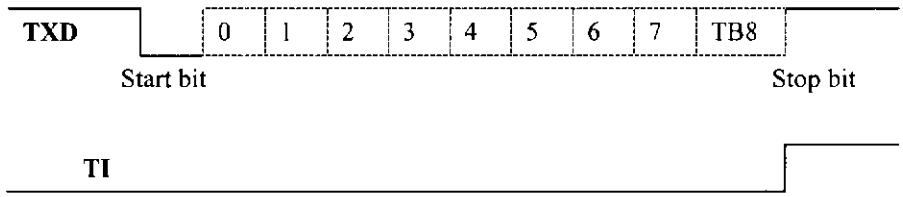


รูปที่ 2.20 แสดง Timing Diagram การส่งข้อมูลโหมด 2

โหมด 3 : พอร์ทสื่อสารอนุกรม 10 bit UART โดย DATA 8 bit, 1 start bit และ 1 stop bit เหมือนโหมด 2 ยกเว้นอัตราความเร็วจะขึ้นกับบิต SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1 สำหรับ 8051 หรือ อัตราโอเวอร์โฟลว์ของ Timer 2 (สำหรับ 80C154D)

$$\text{Baud Rate Mode 3} = \frac{2^{SMOD} \times \text{CPU OSC}}{32 \times 12 \times [256 - (TH1)]}$$ **โดยใช้ Timer 1**

$$\text{Baud Rate Mode 3} = \frac{\text{CPU OSC}}{32 \times [65536 - (RCAP2H,RCAP2L)]}$$ **โดยใช้ Timer 2**



รูปที่ 2.21 แสดง Timing Diagram การส่งข้อมูลโหมด 3

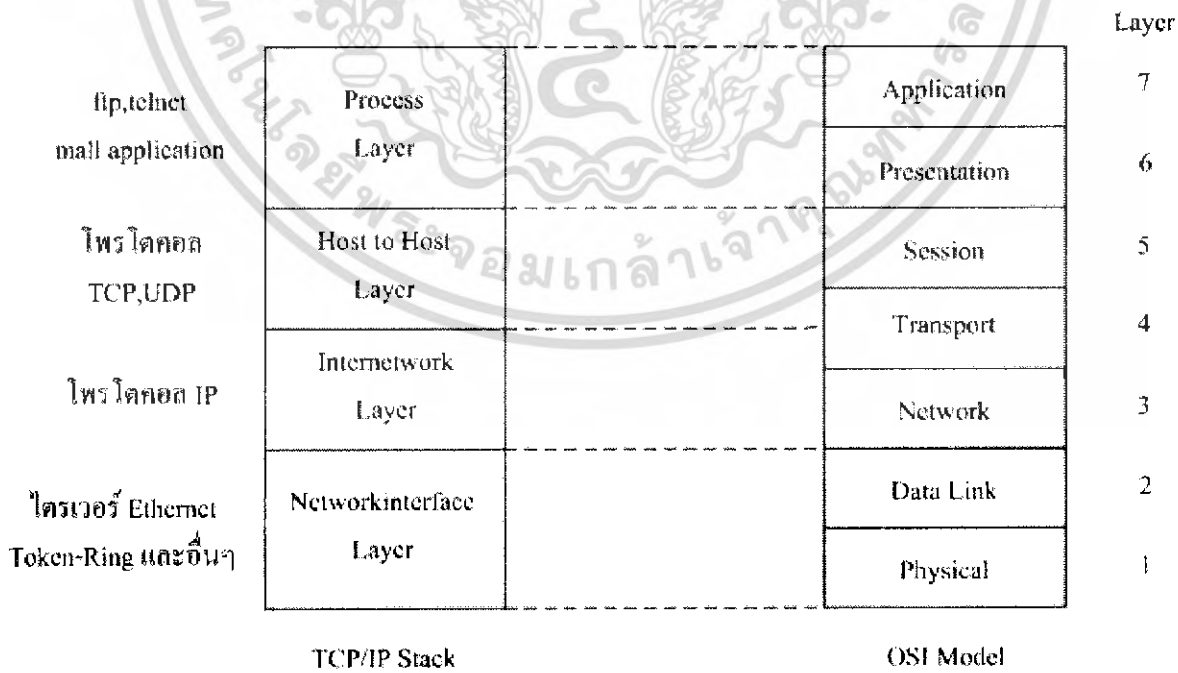
2.4 โพรโตคอล TCP/IP

2.4.1 โครงสร้างของโพรโตคอล TCP/IP

โพรโตคอล TCP/IP มีการจัดกลไกการทำงานเป็นชั้นหรือ Layer เรียงต่อกัน โดยในแต่ละ Layer จะมีการทำงานเทียบได้กับ OSI model มาตรฐาน แต่บาง Layer ของโพรโตคอล TCP/IP จะทำงานเทียบกับ OSI หลาย Layer ปนกัน ซึ่งในแต่ละ Layer ของโพรโตคอล TCP/IP จะประกอบด้วย

- Process Layer
- Host – to – Host Layer
- Internetwork Layer
- Network Interface Layer

โดยเมื่อเทียบกับมาตรฐาน OSI model ดังรูปที่ 2.22 ซึ่งเราจะเห็นว่าบางกลไกของโพรโตคอล TCP/IP เทียบได้กับมาตรฐาน OSI model สองชั้น หรือบางกลไกก็จะทำงานคาบเกี่ยวกันระหว่างบางชั้นของ OSI model ตัวอย่างเช่น กลไกการทำงานของโพรโตคอล TCP/IP ในส่วน Network Interface Layer เมื่อเทียบกับมาตรฐาน OSI model จะเทียบได้กับ Data Link Layer และ Physical Layer 2 ชั้นรวมกัน เป็นต้น ในแต่ละกลไกของโพรโตคอล TCP/IP และโพรโตคอลอื่น ๆ ในชุดของ TCP/IP ร่วมทำงานอยู่ด้วย



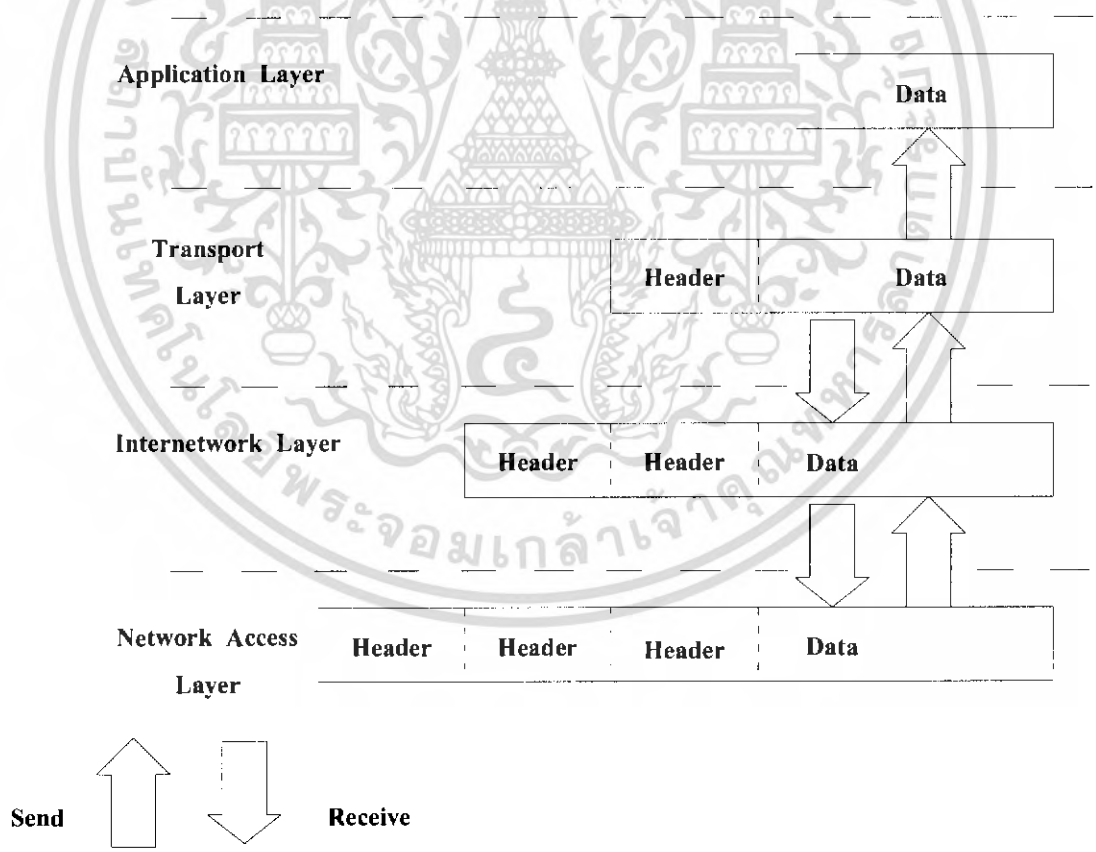
รูปที่ 2.22 แสดงกลไกของโพรโตคอลมาตรฐาน OSI model

สำหรับการศึกษาโพรโตคอล TCP/IP นั้นเราจะไม่อ้างอิง OSI Reference Model นี้เพราะจะเข้าใจได้ยาก ดังนั้นเราจึงจะสร้างโมเดลขึ้นมาใหม่โดยแบ่งออกเป็น 4 ชั้นดังนี้

4	Application Layer
3	Host – to – Host Transport Layer
2	Internet Layer
1	Network Access Layer

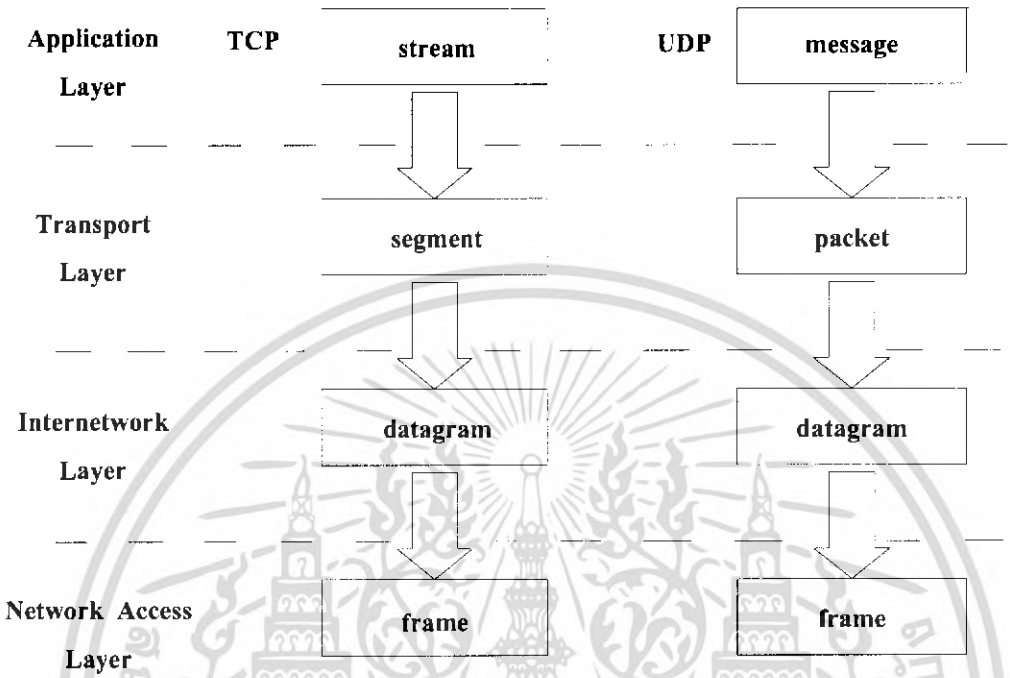
ลักษณะการทำงานของโมเดลในลักษณะนี้คือ ข้อมูลจะถูกส่งลงมาจากชั้นข้างบนลงมายังชั้นข้างล่างสุดซึ่งมีหน้าที่จัดการเกี่ยวกับการส่งข้อมูลผ่านสายสัญญาณไปยังจุดหมายปลายทาง เมื่อข้อมูลไปถึงจุดหมายแล้วก็จะกลับย้อนจากชั้นล่างขึ้นไปชั้นบนสุด ซึ่งเป็นชั้นที่โปรแกรมใช้งานต่างๆ ทำงานอยู่

ขณะที่ข้อมูลถูกส่งผ่านจากชั้นบนลงมายังชั้นล่าง แต่ละชั้นจะทำการเพิ่มข้อมูลควบคุมเข้าไปเพื่อให้การส่งข้อมูลถูกต้อง และเป็นการส่งพารามิเตอร์ที่จำเป็นไปให้กับชั้นของมันในเครื่องปลายทาง ข้อมูลควบคุมเหล่านี้เราเรียกว่า Header แต่ละชั้นจะมี Header ที่มีรูปแบบเป็นของตัวเอง การเพิ่มเฮดเดอร์เข้าไปกับข้อมูลนั้นเราเรียกว่า Data Encapsulation ซึ่งได้แสดงไว้ในรูปที่ 2.23



รูปที่ 2.23 Data Encapsulation

หน่วยของข้อมูลที่จะทำการส่งนั้นนั้นจะมีชื่อเรียกต่างกันเมื่อมันเดินทางผ่านแต่ละ Layer ดังแสดงไว้ในรูปที่ 2.24 ซึ่งจะเห็นว่าการส่งใน TCP/IP นั้นมีอยู่ 2 โพรโตคอล ย่อยคือ TCP กับ UDP ชื่อของข้อมูลที่ผ่านโพรโตคอลทั้งสองนั้นแตกต่างกันในชั้นบนๆ เท่านั้น ชื่อสำคัญๆ ที่เราจะต้องพบและใช้ต่อไปบ่อยๆ ได้แก่ ดาต้าแกรม และเฟรม



รูปที่ 2.24 โครงสร้างของข้อมูล

2.4.2 Network Interface Layer

เนื่องจากในด้านกายภาพของเครือข่ายนั้น มีหลายวิธีการและหลายรูปแบบในการเชื่อมต่อระบบให้เป็นเครือข่าย แต่อย่างไรก็ตามในเครือข่ายอินเตอร์เนตนี้ ข้อมูลหรือ IP datagram จะถูกถ่ายทอดและส่งผ่านไปยังปลายทางโดยไม่คำนึงถึงรูปแบบการเชื่อมต่อทางกายภาพ ไม่ว่าจะเป็นการใช้เครือข่ายใยแก้วนำแสงหรือเครือข่ายสาย Unshielded Twist Pair (UTP) เชื่อมต่อเป็นแบบเครือข่ายอีเทอร์เนต ธรรมดาหรือเครือข่าย Token Ring, ATM, ISDN ฯลฯ ก็ตาม

2.4.2.1 อีเทอร์เน็ท

อีเทอร์เน็ทเป็นการใช้สายโคแอคเชียลเบสบัสเชื่อมต่อระบบเข้าด้วยกันเพื่อทำการส่งถ่ายข้อมูลในระบบดิจิตอลระหว่างคอมพิวเตอร์ โดยบริษัทอุปกรณ์ดิจิตอล บริษัท Intel และบริษัท Xerox ได้ใช้ระบบนี้อ้างอิงเรื่อยมา ซึ่งอีเทอร์เน็ทจะใช้เทคนิคการส่งเบสแบนด์ในการเข้าถึงข้อมูล และยังสามารถใช้บนสายโคแอคเชียลที่มี 2 ขนาด คือ สายอีเทอร์เน็ทแบบหนา และสายอีเทอร์เน็ทแบบบาง โดยสายเคเบิลทั้งสองนี้เป็นที่ใช้กันอย่างกว้างขวางในปัจจุบัน

ส่วนประกอบหลักที่สำคัญของเครือข่ายอีเทอร์เน็ท

ระบบเครือข่ายอีเทอร์เน็ท มีส่วนประกอบหลักซึ่งเมื่อทำงานด้วยกันแล้วก็จะเป็นเครือข่ายที่มีประสิทธิภาพการทำงานสูงดังนี้

1. ตัวเฟรมเป็นชุดรูปแบบของบิตข้อมูลข่าวสารที่ใช้ส่งผ่านมาบนระบบ หากไม่มีเฟรมเราจะไม่สามารถสื่อสารข้อมูลบนเครือข่ายได้โดยเด็ดขาด การรับส่งข้อมูลข่าวสารบนเครือข่ายอีเทอร์เน็ท จะต้องเป็นไปในรูปแบบเฟรมมาตรฐาน 2 แบบ และเป็นแบบใดแบบหนึ่งเท่านั้น (การ์ด LAN เป็นผู้สร้างเฟรมนี้ขึ้นมา)
2. ชุดโปรโตคอลที่ใช้ในการควบคุมการแอกเซสเข้าไปที่เครือข่าย (Media Access Control Protocol) ซึ่งประกอบด้วยชุดของกฎกติกาที่อยู่ใน Ethernet Interface (เช่น การ์ด LAN เป็นต้น) ซึ่งเป็นกฎมาตรฐานที่จะยอมให้คอมพิวเตอร์ต่างๆสามารถเข้ามาที่เครือข่าย และแบ่งใช้ทรัพยากรต่างๆ บนเครือข่ายได้อย่างมีประสิทธิภาพ
3. อุปกรณ์ที่ใช้รับส่งสัญญาณบนเครือข่าย (Signaling Components) ประกอบด้วยชุดของอุปกรณ์ที่ใช้เชื่อมต่อและส่งสัญญาณเพื่อการรับส่งข้อมูลภายในเครือข่าย
4. สื่อที่ใช้ในการรับส่งสัญญาณข้อมูลบนเครือข่าย (Physical Medium) ประกอบด้วยสายสัญญาณรวมทั้งอุปกรณ์ทางฮาร์ดแวร์อื่นๆ ที่จะช่วยในการนำพาข้อมูลข่าวสารต่างๆ ในรูปแบบดิจิตอลวิ่งไปมาบนเครือข่าย

2.4.2.2 เฟรมอีเทอร์เน็ท

อีเทอร์เน็ทได้ถูกระบุไว้อย่างแน่นอนไว้ในชั้น Physical Layer โดยมันจะแสดงรายละเอียดของรูปแบบเป็นแพ็กเก็ต ซึ่งโดยส่วนมากจะเรียกว่า เฟรม ซึ่งเป็นหัวใจสำคัญของระบบอีเทอร์เน็ท จากรูปที่ 2.25 ได้แสดงรูปแบบของเฟรมอีเทอร์เน็ท โดยเฟรมนี้มันจะ Encapsulates TCP/IP โปรโตคอล และสามารถทำการตอบสนองสำหรับส่งข้อมูลข้ามเข้าระบบที่เชื่อมต่อไปยังอีก Layer ได้โดย Gateway หรือ End Node

Preamble	Destination MAC Address (6 Byte)	Source MAC Address (6 Byte)	Type (2 Byte)	Data Field (1500 Byte Max)	Cyclic Redundancy Check (4 Byte)

รูปที่ 2.25 ลักษณะของเฟรมอีเทอร์เน็ท

2.4.2.3 MAC Address

เนื่องจากคอมพิวเตอร์แต่ละเครื่องสามารถแชร์ข้อมูลกันได้ในระบบเครือข่ายเดียวกัน ดังนั้นแต่ละ เครื่องควรมีสิ่งที่ชี้ลักษณะเฉพาะตัวของมัน เช่น เราต้องมีบัตรประจำตัวประชาชน ซึ่งในทางคอมพิวเตอร์นี้เราจะใช้เลขฐาน 16 จำนวน 12 digits เป็นตัวบ่งชี้ลักษณะเฉพาะนั้น ๆ ซึ่งเราเรียกว่า MAC Address เนื่องจาก MAC Address เป็นตัวบ่งชี้ลักษณะเฉพาะของแต่ละเครื่อง ดังนั้นจึงต้องเป็นค่าที่ไม่ซ้ำกัน (Unique) MAC Addressเป็นเลข 48 บิตโดยแบ่งออกเป็น 2 ส่วนโดย 24 บิตแรกเป็นค่าที่แสดงถึง

บริษัทที่ผลิตการ์ดนั้น ๆ ส่วน 24 บิต หลังเป็น Serial Number ที่ทางบริษัทกำหนดให้ ซึ่งแต่ละตัวต้องไม่ซ้ำกัน เราเรียกเลข 24 บิต นี้ว่า OUI (Organizationally Unique Identifier) ซึ่ง OUI จะใช้เพียง 22 บิต เท่านั้น ส่วนอีก 2 บิตที่เหลือจะถูกใช้เพื่อวัตถุประสงค์อื่น โดยบิตหนึ่งจะใช้เพื่อแสดงว่าแอดเดรสนั้นเป็น Broadcast/Multicast Address ส่วนอีกบิตหนึ่งนั้นไว้แสดงว่า Adapter นั้นถูกกำหนด Locally Administered Address ซึ่ง Admin ของระบบจะทำการกำหนด MAC Address เพื่อความเหมาะสมของนโยบายระบบ เช่น MAC Address = 03 00 00 00 00 00 01 ซึ่งจะเห็นว่าไบต์แรก = 03 = 00000011 นั่นคือทั้ง 2 bits ถูก set (reset = 0) ซึ่งเอาไว้กรณี Multicast ให้ทุกเครื่องที่รันบนโปรโตคอล NetBEUI

2.4.2.4 Type field

ประเภทของฟิลด์จะใช้ระบุความแตกต่างของโปรโตคอล คอมพิวเตอร์จะดำเนินการให้โปรโตคอลหลายๆตัว สามารถเห็นความแตกต่างของพวกมันได้ง่ายและผ่านการยินยอมของเฟรมที่เกี่ยวเนื่องกับเครือข่าย

ระบบ TCP/IP โดยทั่วไปจะใช้อีเทอร์เน็ท 3 ฟิลด์ ซึ่งมีค่าดังตารางที่ 2.4 ประเภทของหมายเลขอีเทอร์เน็ทเป็นรีจีสเตอร์กับ IEEE โดยจะใช้หมายเลขที่เฉพาะเจาะจงไม่เหมือนใคร

ตารางที่ 2.4 Ethernet type fields

Type	Protocol
0x0800	IP
0x0806	ARP
0x0835	RARP

2.4.2.5 Cyclic Redundancy Check (CRC)

ที่ปลายสุดของเฟรม คือ Cyclic Redundancy Check (CRC) มีขนาด 32 บิต ที่คำนวณได้จากบิตทั้งหมดของอีเทอร์เน็ทเฟรม แต่จะไม่สนใจ Preamble ถ้าในเฟรมมีความผิดพลาดเกิดขึ้น ค่าที่คำนวณได้จะแตกต่างไปจากเฟรมเดิม ทำให้ข้อมูลไม่สามารถที่จะผ่านเฟรมไปยังชั้น Network layer ได้

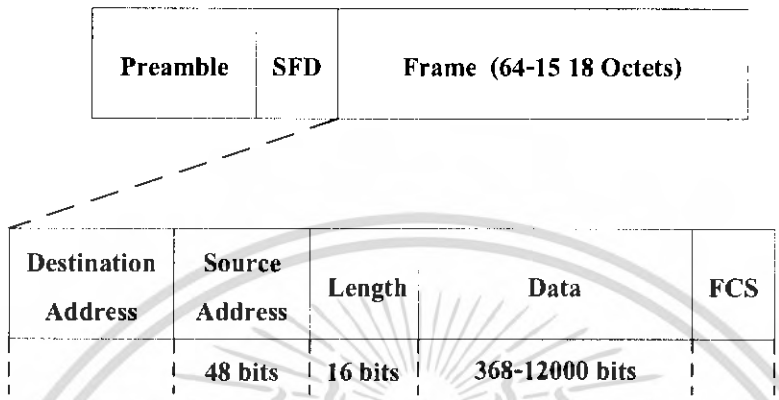
2.4.2.6 IEEE 802.3 เฟรม

IEEE 802.3 หรือ อีเทอร์เน็ท (Ethernet) เป็นเครือข่ายที่มีความเร็วสูงการส่งข้อมูล 10 เมกะบิตต่อวินาที สถานีในเครือข่ายอาจมีโทโปโลยีแบบบัสหรือแบบดาว IEEE ได้กำหนดมาตรฐานอีเทอร์เน็ทซึ่งทำงานที่ความเร็ว 10 เมกะบิตต่อวินาทีไว้หลายประเภทตามชนิดสายสัญญาณเช่น

- 10Base5 อีเทอร์เน็ทโทโปโลยีแบบบัสซึ่งใช้สายโคแอกเชียลแบบหนา (Thick Ethernet) ความยาวของสายในเซกเมนต์หนึ่ง ๆ ไม่เกิน 500 เมตร
- 10Base2 อีเทอร์เน็ทโทโปโลยีแบบบัสซึ่งใช้สายโคแอกเชียลแบบบาง (Thin Ethernet) ความยาวของสายในเซกเมนต์หนึ่ง ๆ ไม่เกิน 185 เมตร

- 10BaseT อีเทอร์เน็ทโทโปโลยีแบบดาวซึ่งใช้ฮับเป็นศูนย์กลาง สถานีและฮับเชื่อมด้วยสายยูทีพี (Unshield Twisted Pair) ด้วยความยาวไม่เกิน 100 เมตร

เฟรมข้อมูลสำหรับระบบอีเทอร์เน็ทประกอบขึ้นด้วยกลุ่มของบิตที่เป็นข้อมูลและข่าวสารสำคัญ แบ่งออกเป็นขนาดสัดส่วนที่แน่นอนที่เรียกว่าช่อง Field ดังรูปที่ 2.26



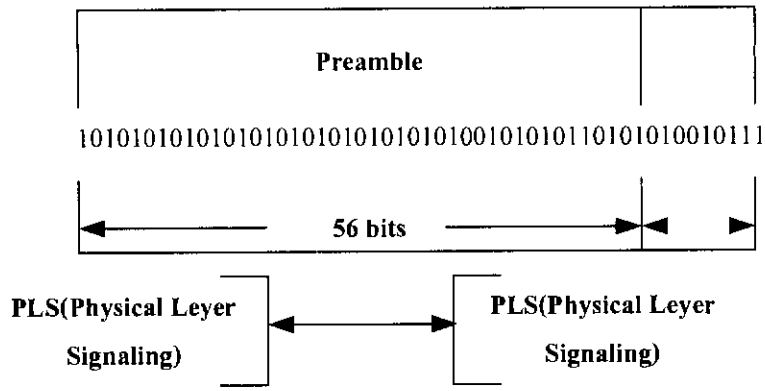
รูปที่ 2.26 ลักษณะโครงสร้างของเฟรมข้อมูลตามมาตรฐาน IEEE802.3

รูปที่ 2.26 แสดงให้เห็นรูปแบบของเฟรมข้อมูลที่ใช้บนอีเทอร์เน็ทตามมาตรฐาน IEEE802.3 ต่อไปเราจะมาทำความเข้าใจเฟรมมาตรฐานของ IEEE802.3

ช่อง Preamble

ช่อง Preamble ประกอบด้วยบิตข่าวสารที่เป็นเลข 1 และ 0 สลับกัน และสิ้นสุดที่ 11 ซึ่งเป็นบิตที่ 63 และ 64 เป็นบิตข่าวสารที่ยังไม่ใช่ข้อมูลจริงของผู้ส่ง Preamble ประกอบด้วยข่าวสารที่มีขนาด 7 หรือ 8 ไบต์ จุดประสงค์ของข่าวสารนี้ก็เพื่อใช้สร้างจังหวะการรับข้อมูลให้แก่ผู้รับโดยที่ส่วนนี้จะไปถึงตัวผู้รับก่อน ทำให้เครื่องคอมพิวเตอร์ของผู้รับสามารถปรับจังหวะความเร็วให้เข้ากับผู้ส่งได้ (Synchronize) สำหรับเฟรมแบบ Ethernet II จะมีขนาด 8 ไบต์ และถ้าเป็นมาตรฐาน IEEE802.3 แล้ว ช่องนี้จะถูกแบ่งออกเป็น 2 ส่วน (ดังรูปที่ 2.27) ได้แก่

1. Preamble
2. Start of Frame Delimiter



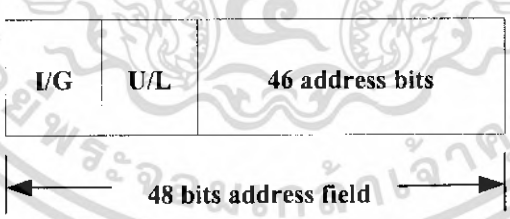
รูปที่ 2.27 ลักษณะส่วนการทำงานภายในของ Preamble

ช่อง Destination Address

ในช่อง Destination Address ประกอบด้วยข้อมูลข่าวสารเกี่ยวกับแอดเดรสหรือที่อยู่ของผู้รับปลายทาง ดูเผินๆแล้วเป็นช่องที่เรียบง่ายไม่มีอะไรมาก แต่โดยความจริงแล้วช่องนี้ถูกแบ่งออกเป็นช่องย่อยๆที่เรียกว่า Sub-Fields ซึ่งเก็บข้อมูลข่าวสารที่ใช้ดูแลการทำงานของเครือข่าย ทั้งนี้ขึ้นอยู่กับแอดเดรสที่ปรากฏอยู่ในช่องนี้ไม่ว่าช่องแอดเดรสนี้จะมีแอดเดรสที่ถูกเจาะจงเป็นรายบุคคลหรือเป็นกลุ่มของผู้รับก็ตาม

ก. ช่องข่าวสารขนาด 2 ไบต์

ข. ช่องข่าวสารขนาด 6 ไบต์



หมายเหตุ

หากค่าบิตในช่องย่อย I/G มีค่าเป็น "0" ก็หมายถึงแอดเดรสที่ระบุตัวคอมพิวเตอร์ผู้รับอย่างเฉพาะเจาะจง แต่ถ้ามีค่าเป็น "1" ก็หมายถึงคอมพิวเตอร์ที่ระบุแอดเดรสเป็นกลุ่ม ไม่เจาะจงเครื่องใดเครื่องหนึ่ง

หากค่าบิตในช่อง U/L มีค่าเป็น "0" ก็หมายความว่าแอดเดรสนี้ถูกกำหนดมาตรฐานโดย IEEE และถ้ามีค่าเป็น "1" ก็จะหมายถึงเป็นแอดเดรสมาตรฐานเฉพาะองค์กรที่ระบุมาตรฐานในซอฟต์แวร์ ซึ่งแอดเดรสมาตรฐานที่เราใช้กันอยู่ทุกวันนี้ถูกควบคุมโดย IEEE

ช่อง I/G

มีการจัดตั้งค่าบิตขึ้นในช่องนี้โดยการ์ต LAN ซึ่งหากค่านี้ถูกตั้งค่าไว้ที่ "0" ก็แสดงว่าตัวแอดเดรสที่ระบุอยู่ในช่อง Destination Address นั้นเป็นแอดเดรสที่ระบุตัวคอมพิวเตอร์ผู้รับแบบเฉพาะเจาะจง แต่ถ้าถูกตั้งค่าเป็น "1" ก็แสดงว่าแอดเดรสในช่อง Destination Address นี้เป็นแอดเดรสที่ใช้ติดต่อผู้รับที่เป็นกลุ่มคอมพิวเตอร์ทั้งหลาย เราเรียก Group Address ตัวอย่างของ Group Address ได้แก่ "FFFFFFFFFFFF" ซึ่งถือว่าเป็น Broadcasting Address หรือแอดเดรสที่ไม่เจาะจงผู้รับ โดยผู้รับเป็นกลุ่มหรือทั้งหมดก็สามารถรับข้อมูลข่าวสารนี้ได้

ช่องย่อย U/L

ช่องย่อย U/L มีไว้สำหรับช่องขนาด 6 ไบต์เท่านั้น ค่าที่ถูกตั้งไว้ในช่องย่อยนี้เป็นการบ่งบอกให้ทราบว่าแอดเดรสที่ปรากฏอยู่ในช่อง Destination Address นี้เป็นแอดเดรสที่ถูกกำหนดมาตรฐานโดย IEEE หรือองค์กรอย่างเฉพาะเจาะจง

ช่อง Source Address

สำหรับช่อง Souree Address นี้มีไว้เพื่อแสดงตัวสถานีเครือข่ายต้นทางที่เป็นต้นทางส่งข้อมูลข่าวสารเข้ามาและเช่นเดียวกับช่อง Destination Address กล่าวคือ ช่อง Source Address สามารถมีช่องย่อยได้ทั้งแบบ 2 ไบต์หรือ 6 ไบต์อย่างใดอย่างหนึ่ง

ตารางที่ 2.5 เป็นตัวอย่างของ Source Address ที่แสดงรหัสแอดเดรสของผู้ผลิตดังนี้คือ

ผู้ผลิตการ์ด LAN	รหัสผู้ผลิตขนาด 3 ไบต์
Cisco	00-00-0C
Cabletron	00-00-1D
Intel	00-AA-00
3 Com	02-60-8C
Hewlett Packard	08-00-09
Sun	08-00-20
DEC	08-00-2B
Shiva	00-80-D3
Xerox	00-00-AA
IBM	08-00-5A

ช่องแสดง Type

ช่องแสดง Type มีขนาด 2 ไบต์ ใช้กับอีเทอร์เน็ทเฟรม เท่านั้น โดยช่องนี้ใช้เพื่อแสดงว่าโปรโตคอลการทำงานของเฟรมนี้เป็นแบบใด จุดประสงค์คือเพื่อต้องการให้ทราบว่าข้อมูลที่อยู่ในเฟรมนี้

จะทำงานภายใต้โปรโตคอลใด ซึ่งผู้รับจะได้เตรียมการแปลความหมายที่อยู่ในช่องข้อมูล (Data Field) ได้ถูกต้อง

ภายใต้ระบบเครือข่ายอีเทอร์เนตเราสามารถใช้โปรโตคอลได้หลายตัวพร้อมกันบนเครือข่าย LAN และบริษัท XEROX ทำหน้าที่เป็นผู้ให้บริการ กำหนดพิกัดระยะของแอดเดรสที่เป็นลิขสิทธิ์ให้แก่ผู้ผลิตการ์ด LAN ต่างๆรวมทั้งการกำหนดค่าที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type แห่งนี้

ตารางที่ 2.6 เป็นตัวอย่างของรหัสที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type 18 รายการดังนี้

โปรโตคอลที่ใช้	ค่าที่เป็นรหัสแบบเลขฐาน 16
IP	0800
X.75 Internet	0801
X.25 Level 3	0805
Address Resolution Protocol (ARP)	0806
Banyan Systems	0BAD
BBN Simnet	5208
DEC MOP Dump/Load	6001
DEC MOP Remote Console	6002
DEC DECNET Phase IV Route	6003
DEC LAT	6004
DEC Diagnostic Protocol	6005
DEC LANBridge	8038
DEC Ethernet Encryption	803D
Apple Talk	809B
IBM SNA Service on Ethernet	80D5
Apple Talk ARP	80F3
NetWare IPX/SPX	8137
SNMP	814C

ช่อง Length

ช่องนี้มีขนาดความยาวเพียง 2 ไบต์ใช้ได้กับเฟรมมาตรฐาน IEEE802.3 เท่านั้นเป็นช่องที่ใช้แสดงขนาดจำนวนของไบต์ที่มีปรากฏอยู่ในช่อง Data

ช่อง Data (Data Field)

ดังที่ได้กล่าวมาแล้วว่าช่องของ Data อย่างน้อยต้องมีขนาดไม่เล็กกว่า 46 ไบต์ เพื่อให้แน่ใจว่าเฟรมมีขนาดไม่ต่ำกว่า 64 ไบต์ซึ่งหมายความว่าการแพร่ข้อมูลขนาดหนึ่งไม่ว่า 1 หรือ 10 ไบต์ก็ตามต้องมาจาก 46ไบต์นี้แต่ถ้าข้อมูลในช่องนี้เล็กกว่า 46ไบต์ แน่นอนว่าต้องมีการเพิ่มไบต์ลงไปอีกเพื่อให้ได้ขนาด 46 ไบต์อยู่ดี

ขนาดของข้อมูลที่อยู่ใน Data จะต้องมีขนาดสูงสุดไม่เกิน 1,500 ไบต์

ช่องตรวจสอบความผิดพลาดของข้อมูลในเฟรม (Frame Check Sequence)

ช่อง Frame Check Sequence นี้ใช้ได้ในเฟรมมาตรฐาน ทั้งเฟรมมาตรฐาน ทั้งอีเทอร์เน็ทและ IEEE802.3 เป็นช่องที่ประกอบด้วยข้อมูลที่ใช้เป็นกลไกในการตรวจสอบความผิดพลาดของข้อมูลภายในเฟรม

หลักการทำงานมีอยู่ว่าก่อนที่เครื่องผู้ส่งจะส่งข้อมูลออกไปที่เครือข่าย การ์ด LAN ของมันจะคำนวณค่าต่างๆในช่องต่างๆซึ่งครอบคลุมตั้งแต่ช่อง Address ต่างๆของ Type และช่อง Length รวมทั้งช่อง Data การคำนวณค่าแบบนี้เรียกว่า Cyclic Redundancy Check (CRC) ซึ่งหลังจากที่ได้คำนวณค่าเสร็จสิ้นแล้ว ผลลัพธ์ที่คำนวณได้มีขนาด 4 ไบต์จะถูกนำไปใส่ไว้ในช่อง Frame Check Sequence แห่งนี้

2.4.3 Internetwork Layer

ในระดับล่างต่อมาในชั้น Internetwork Layer มีหน้าที่ส่งผ่านข้อมูลในระหว่างเครือข่าย โดยมีโพรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใด ๆ บนอินเตอร์เน็ท คือ โพรโตคอล IP (Internet Protocol) นอกจากนี้ในชั้น Internetwork Layer ยังมีโพรโตคอลทำงานอยู่ด้วยอีก 2 ชนิดคือ โพรโตคอล Internet Control Message Protocol (ICMP) และโพรโตคอล Address Resolution Protocol (ARP) ซึ่งอยู่ภายในโพรโตคอล IP

2.4.3.1 โพรโตคอล IP (Internet Protocol)

โพรโตคอล IP ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาจาก Host – to – Host Layer เพื่อส่งข้ามไปยังเครือข่ายใด ๆ ได้อย่างถูกต้อง แม้ว่าจะมีเครือข่ายเชื่อมต่อกันอยู่ในอินเตอร์เน็ทเป็นล้าน ๆ เครือข่ายก็ตาม เนื่องจากโพรโตคอล IP มีข้อมูลตำแหน่ง IP ปลายทางที่จะส่งข้อมูลไปให้โดยทำงานร่วมกับอุปกรณ์ Router เพื่อส่งข้อมูลข้ามเครือข่ายออกไปได้ ตัวโพรโตคอล IP จะทำงานแบบ Packet Switching คือมีการส่งข้อมูลผ่านสวิทช์ (Switch) ไปยังปลายทาง โดยข้อมูลจะเดินทางไปยังเครือข่ายต่าง ๆ ผ่านสวิทช์นี้ไปเรื่อยๆ จนกว่าจะถึงปลายทาง ตัววงจรผ่านหรือสวิทช์นี้อาจเป็น Gateway หรือ Router ในระบบเครือข่ายก็ได้ ซึ่งในข้อมูลของโพรโตคอล IP จะมีข้อมูลของหมายเลข IP ที่จะส่งข้อมูลไปและเมื่อถึงเครือข่ายปลายทางแล้ว จะมีกลไกแปลงหมายเลข IPให้เป็นหมายเลขฮาร์ดแวร์ประจำเครื่องที่ถูกต้องอีกทีหนึ่งด้วยโพรโตคอล ARP

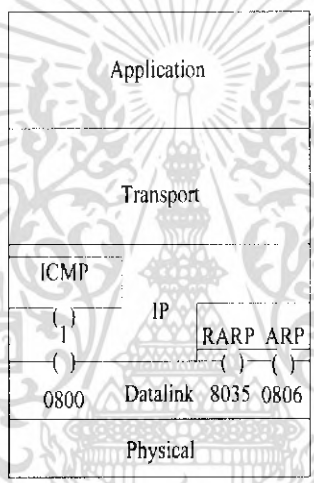
IP จะให้บริการชนิด Connectionless สำหรับ User ดังนั้น Data ที่ถูกส่งผ่าน IP โดยกระบวนการ Send ยังไม่แน่ว่าจะสามารถส่งถึง ข้อมูลของ IP หน่วยต่างๆที่ถูกส่ง เราเรียกว่า IP Datagram ซึ่งผ่านการ

Encapsulation ข้อมูลที่ถูกส่งมาจาก Layer ที่สูงกว่าด้วย IP Header ถ้า IP Datagram ไม่ได้ถูกนำส่งภายในเวลาที่กำหนด (Time – to –Live) Datagram นั้นก็จะไม่ถูกส่งอีกเลย สำหรับช่วงเวลา Time – to –Live นั้นสามารถกำหนดไว้ในกระบวนการส่ง

2.4.3.2 ส่วนประกอบของ IP

IP Address จะต้องมีค่าเฉพาะเจาะจงไม่เหมือนใคร เพื่อที่จะใช้เชื่อมต่อเข้ากับเครือข่ายที่หมายเลขของเครือข่ายที่เฉพาะเจาะจงเช่นกัน โดยในรูปที่ 2.26 แสดงให้เห็นว่าใน IP Layer จะประกอบไปด้วย 3โปรโตคอล ได้แก่

1. The Address Resolution Protocol (ARP)
2. The Reverse Address Resolution Protocol (RARP)
3. The Internet Control Message Protocol (ICMP)



รูปที่ 2.28 ส่วนประกอบของ IP

ARP และ RARP จะอยู่ที่ส่วนปลายสุดของ IP Layer เพราะทั้งสองโปรโตคอลไม่ใช้ IP และจะถูกแยกโปรโตคอลโดย Data link Layer ที่สนับสนุนตัวมันอยู่ ส่วน ICMP จะอยู่ในส่วนบนสุดของ IP Layer ซึ่งจะข้ามเข้าไปในเครือข่ายใน IP Datagram

2.4.3.3 IP Datagram

Datagram เป็น Basic Unit ของข้อมูลที่ IP จะทำการส่ง ซึ่งถูกออกแบบมาให้ทำงานกับเครือข่ายแบบ Packet Switch ซึ่งข้อมูลของผู้ใช้มักถูกแบ่งออกเป็นหลาย Datagram โดยแต่ละตัวจะมี Header ที่เก็บรายละเอียดเกี่ยวกับตัวมันเอง และปลายทางที่มันจะไป Datagram แต่ละตัวจะเดินทางโดยไม่เกี่ยวข้องกัน นั่นคือ Datagram แต่ละตัวอาจจะเดินทางไปยังปลายทางโดยใช้เส้นทางคนละเส้น และลำดับที่ของการไปถึงจุดหมายปลายทางก็ไม่แน่นอน เป็นหน้าที่ของ Internet Protocol ในเครื่องปลายทางที่จะต้องประกอบ Datagram เหล่านี้ให้กลายเป็นข้อมูลของผู้ใช้ที่สมบูรณ์อีกครั้ง

IP Datagram ประกอบด้วย IP Header และ IP Payload

IP Header	IP Payload

2.4.3.3a **IP Header** เป็นขนาดที่เปลี่ยนแปลงได้ระหว่าง 20 และ 60 ไบต์ ในการเพิ่มขึ้น 4 ไบต์ มันจะจัดเตรียมการสนับสนุน Routing , การแสดงตัว Payload , การชี้ให้เห็นถึงขนาด IP Header และ Datagram , การสนับสนุน Fragmentation โดยมีโครงสร้างดังรูปที่ 2.29

Version	IP Header Length	Type of Service	Total Length	Identifier	Flags	Fragment Offset
4 bit	4 bit	8 bit	16 bit	16 bit	3 bit	13 bit

Time – to – Live	Protocol	Header Checksum	Source IP Address	Destination IP Address	IP Option and Padding
8 bit	8 bit	16 bit	32 bit	32 bit	32 bit

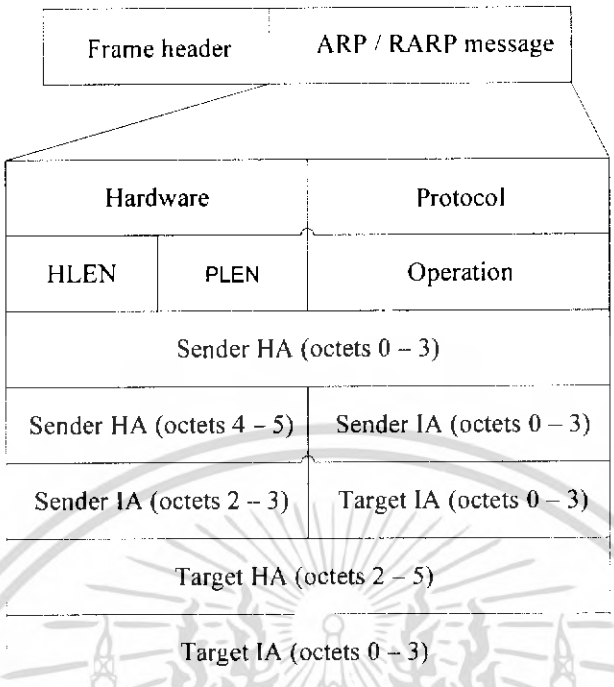
รูปที่ 2.29 แสดงโครงสร้าง IP Header

2.4.3.3b **IP Payload** เป็นขนาดที่เปลี่ยนแปลงโดยมีลำดับจาก 8 ไบต์ (68 ไบต์ IP Datagram กับ 60 ไบต์ IP Header) ถึง 65,515 ไบต์ (65,535 ไบต์ IP Datagram กับ 20 ไบต์ IP Header)

2.4.3.4 Address Resolution Protocol (ARP)

โพรโตคอล ARP (Address Resolution Protocol) ถูกเรียกใช้งานโดยโพรโตคอล IP เพื่อช่วยแปลงหมายเลข IP ไปเป็นหมายเลขฮาร์ดแวร์ปลายทาง ตัวอย่างเช่น เว็บเซิร์ฟเวอร์เครื่องหนึ่งเชื่อมต่ออยู่ในเครือข่ายอินเตอร์เน็ท และในการเชื่อมต่อนี้ต้องอาศัย Network Interface Card (NIC) หรือ LAN Card ติดตั้งอยู่ ที่LAN Card นี้เองจะมีหมายเลขเฉพาะประจำฮาร์ดแวร์ที่ไม่ซ้ำกับใคร เพื่อใช้อ้างอิงการส่งข้อมูลในเครือข่าย แต่เมื่อมาใช้งานในโพรโตคอล TCP/IP ก็จะต้องมีการกำหนดหมายเลข IP Address ประจำตัวเพื่อใช้อ้างอิงกัน และโพรโตคอล ARP จะทำหน้าที่แปลงค่าหมายเลข IP ให้เป็นหมายเลขฮาร์ดแวร์จริงให้ในระดับการทำงานที่ Internetwork Layer นี้ ซึ่งกลไกการแปลงนี้เรียกว่า Address Resolution

2.4.3.4a ARP Datagram



รูปที่ 2.30 แสดง ARP Datagram

ในรูปที่ 2.30 แสดง ARP Datagram โดยที่มันไม่สามารถนำมาใช้ได้ทั้งหมดสำหรับ TCP/IP หรือเครือข่ายใดเครือข่ายหนึ่งโดยเฉพาะ โดยมันแค่ถูกกำหนดให้เป็นตัวกลางที่มีความสามารถทำการส่งเฟรม Broadcast กระบวนการของ ARP จะดำเนินการโดยตรงเข้าไปอยู่เหนือ Datalink Layer และจะทำการ Encapsulate ในเฟรม Datalink โดย ARP Datagram จะประกอบไปด้วย

2.4.3.4b กระบวนการทำงานของ ARP

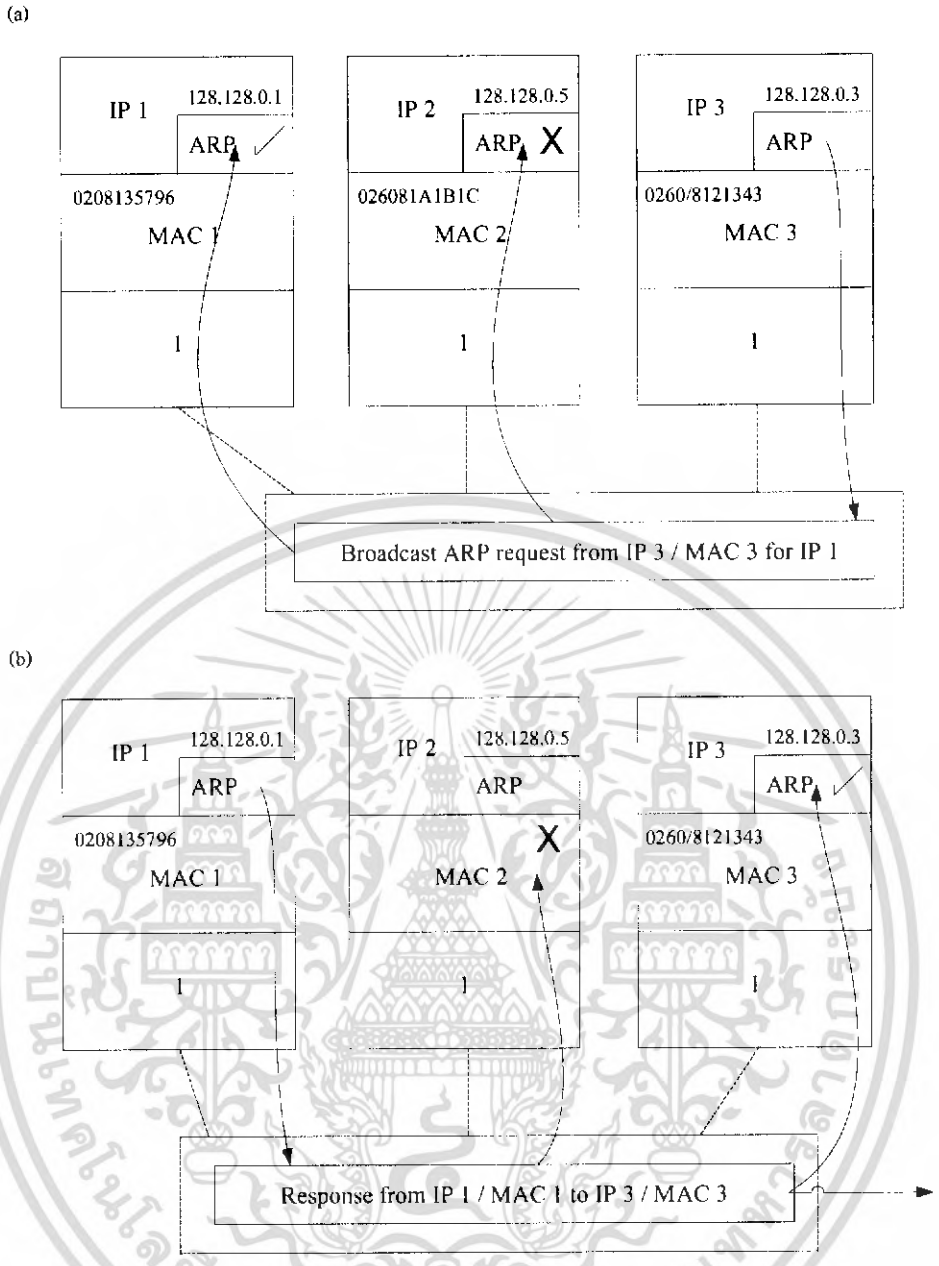
การติดต่อสื่อสารระหว่างคอมพิวเตอร์บนเครือข่ายนั้นจะต้องอาศัยค่า MAC Address เป็นสำคัญ ฉะนั้นหากว่าเครื่องส่งไม่ทราบค่า MAC Address ของเครื่องรับแล้ว การส่งข้อมูลก็ไม่สามารถเกิดขึ้นได้ ดังนั้นจำเป็นที่ต้องมีกลไกที่ใช้ค้นหาค่า MAC Address ของเครื่องรับ ในแบบจำลอง TCP/IP ได้มีการเลือกใช้ ARP ซึ่งเป็นกลไกที่อนุญาตให้ IP Protocol ค้นหาค่า MAC Address ที่สัมพันธ์กับ IP Address

การทำงานของ ARP ประกอบไปด้วยกระบวนการต่อไปนี้

1. เครื่อง Client จะใช้ค่าของ IP Address เพื่อค้นหา MAC Address ที่สอดคล้องกัน ซึ่งเก็บไว้ใน ARP Cache
2. ถ้าไม่พบค่า MAC Address สำหรับ IP Address ดังกล่าวใน ARP Cache แล้วเครื่อง Client ก็จะทำการส่ง ARP Request Packet ด้วยวิธีการ Broadcast ในระดับชั้นย่อย MACให้กับทุกๆ Host ใน LAN ทราบ

3. ถ้าหากมี Host ที่มีค่า IP Address สอดคล้องกับข้อมูล ARP Request Packet ดังกล่าวมันจะทำการส่ง ARP Reply Packet แบบ Unicast ในระดับชั้นย่อย MAC ซึ่งภายในมีข้อมูลของ MAC Address และ IP Address ของ Host ดังกล่าวกลับไปที่เครื่อง Client
4. เครื่อง Client จะบันทึกข้อมูลค่า MAC Address และ IP Address ดังกล่าวลงบน ARP Cache

ในรูปที่ 2.29 แสดงตัวอย่างการการดำเนินการของ ARP โดย IP Address 128.128.0.3 และค่า MAC Address 0x0268121343 จะทำการร้องขอโดยตัวมัน IP Layer ก็จะค้นหาค่า MAC Address กับ IP Address 128.128.0.1 โดยเริ่มแรก 128.128.0.3 จะส่ง ARP Request ที่จะถูกรับโดย ARP Software บนทุกๆจุดในเครือข่าย 128.128.0.1 จะยอมรับ Address ของมันในการร้องขอ และจะส่ง ARP Reply กลับไป แต่จะใช้ค่า MAC Address ที่ ARP Request ส่งมา ARP Reply จะถูกดำเนินการ และถูกละทิ้งไปอย่างรวดเร็วโดย Card LAN ที่ทุกๆจุดบนเครือข่าย ขณะที่มันเป็น Unicast Frame กับ Destination Address ที่ผิด



รูปที่ 2.31 ตัวอย่างกระบวนการของ ARP, (a) ARP Request, (b) ARP Response

2.4.3.5 Reverse Address Resolution Protocol (RARP)

วิธีการ ARP ช่วยแก้ปัญหาในการค้นหาที่อยู่ของข้อมูลที่ใช้การกำหนดที่อยู่แบบฮาร์ดแวร์ แต่ถ้าทราบที่อยู่แบบฮาร์ดแวร์แล้วต้องการแปลงที่อยู่เป็น IP จะทำอย่างไร ปัญหานี้มักเกิดขึ้นกับเครื่อง Computer ที่เริ่มทำงานด้วยการอ่านข้อมูลทั้งหมดจากเครื่อง Host เครื่องประเภทนี้จะทราบเพียงที่อยู่ ของตนเองจากอุปกรณ์สื่อสารเครือข่ายเท่านั้น

การค้นหาคำตอบสามารถทำได้โดยวิธีควบคุมการสื่อสารแบบ ARP ย้อนกลับ หรือ RARP (Reverse Address Resolution Protocol) วิธีการนี้ Computer ที่เพิ่งจะเริ่มทำงาน (หรือเครื่องใดก็แล้วแต่) จะส่งคำถามออกไปในทำนอง "ที่อยู่ขนาด 48 บิตแบบฮาร์ดแวร์ของฉันคือ 14.04.05.18.01.25 มีใครทราบ

ที่อยู่ IP ของฉันบ้าง" เครื่องที่ให้บริการ RARP จะตรวจดูข้อมูลในตารางข้อมูลของตนเองแล้วจึงส่งหมายเลข IP กลับไปให้ วิธีการนี้ช่วยให้เกิดความอ่อนตัวและเพิ่มประสิทธิภาพในการใช้หมายเลข IP เนื่องจากผู้ใช้ไม่มีหมายเลข IP เป็นของตนเอง ผู้ควบคุมระบบสามารถกำหนดหมายเลข IP ใดๆที่ไม่มีผู้ใช้งานในขณะนั้นให้ใช้ได้ หมายเลข IP ในที่นี้จึงเป็นเสมือนสมบัติส่วนกลางที่ทุกคนใช้ร่วมกัน

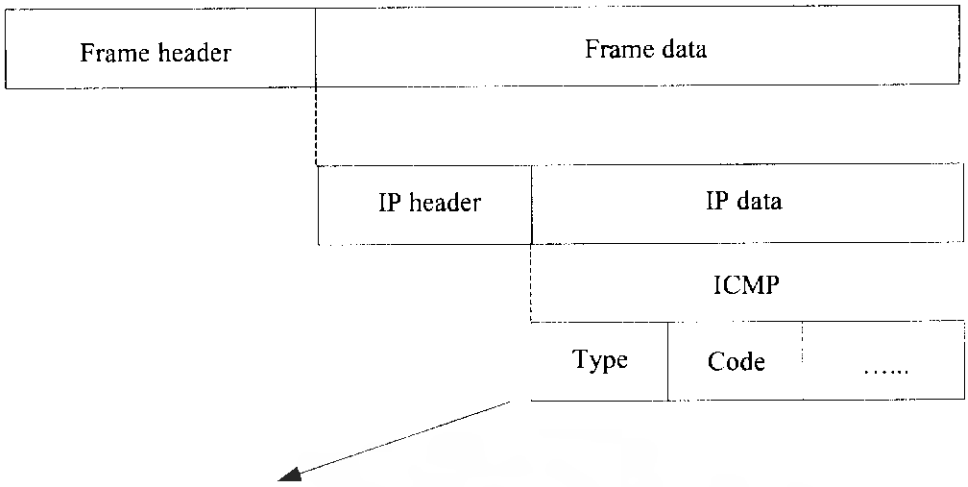
2.4.3.6 Internet Control Message Protocol (ICMP)

ถึงแม้ว่า IP จะเป็น Datagram Service และไม่มีการรับประกันรูปแบบการส่ง โดย Internet Control Message Protocol (ICMP) จะถูกจัดเตรียมไว้ภายใน IP ทำให้เกิด Error Messages ให้เข้าไปช่วย IP Layer ให้มีความสามารถในการส่งที่ดีที่สุด โดยหน้าที่หลักของโปรโตคอล ICMP (Internet Control Message Protocol) คือ การแจ้งหรือแสดงข้อความจากระบบ เพื่อบอกให้ผู้ใช้ ทราบว่า เกิดอะไรขึ้นในการส่งผ่านข้อมูลนั้น ซึ่งปัญหาส่วนมากที่พบคือ ส่งไปไม่ได้ หรือปลายทางรับข้อมูลไม่ได้ เป็นต้น นอกจากนี้ โปรโตคอล ICMP ยังถูกเรียกใช้งานจากเครื่อง Server และ Router อีกด้วย เพื่อแลกเปลี่ยนข้อมูลที่ใช้ควบคุม ส่วนรูปแบบการทำงานของโปรโตคอล ICMP นั้นจะทำควบคู่กับโปรโตคอล IP ในระบบเดียวกัน และข้อความต่าง ๆ ที่แจ้งให้ทราบจะถูกผนึกอยู่ภายในข้อมูล IP (IP datagram) อีกทีหนึ่ง

รูปที่ 2.26แสดงรูปแบบพื้นฐานของ ICMP message encapsulated ใน IP Datagram ใน ICMP มีหมายเลข IP โปรโตคอลของตัวมันเอง ดังนั้น IP Layer รู้ว่ามันรับ ICMP แม้ว่า ICMP ใช้ IP Layer ที่เป็นตัวพิจารณาว่า IP ภายในเป็นของใคร เพราะว่ามันไม่สามารถจัดเตรียมให้กับ Layer ที่สูงกว่าได้

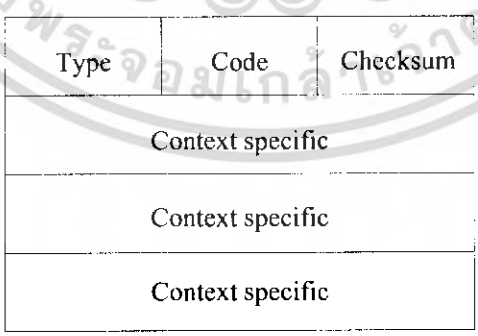
นับตั้งแต่ IP Message ที่ถูกขนย้ายใน IP มันจะถูกทิ้งไปเหมือนกับ IP Datagram โดยมันจะไม่สามารถรักษาสถานภาพไว้ได้ ICMP Message จะไม่ถูกทำให้เกิดขึ้นในกรณีที่ ICMP Message เกิดความผิดพลาดเกิดขึ้น

รูปแบบพื้นฐานของ ICMP Datagram แสดงไว้ในรูปที่ 2.30โดยจะประกอบไปด้วยการแบ่งประเภทของ ICMP Message และ Code ที่ต้องจัดเตรียมรายละเอียด Checksum ต้องถูกนำมาใช้ เพราะ IP ไม่สามารถที่จะป้องกันข้อมูลของมันได้ เมื่อทำการดำเนินการมาอยู่เหนือ Physical Network ที่มี Frame Check Sequence ICMP Checksum ต้องเป็น 0 นั่นหมายความว่าจะไม่มีการคำนวณเกิดขึ้น



Type field	Message type
0	Echo reply
3	Destination unreachable
4	Source quench
5	Redirect (change route)
8	Echo request
11	Time exceeded for datagram
12	Parameter problem on datagram
13	Time stamp request
14	Time stamp reply
15	Information request
16	Information reply
17	Address mask request
18	Address mask response

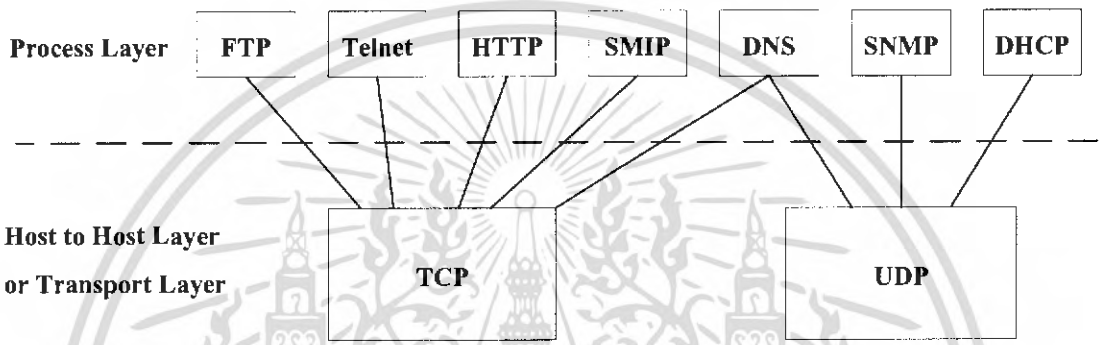
รูปที่ 2.32 ICMP encapsulated ใน IP และประเภทของ ICMP



รูปที่ 2.33 รูปแบบของ ICMP Datagram

2.4.4 Host-To-Host Layer

การทำงานที่ชั้นของ Host-to- Host Layer นี้จะมีบทบาทในการจัดการต่อจาก Process Layer บางครั้งเรามักเรียกชั้น Host-to-Host ว่าเป็น Transport Layer ซึ่งไม่ใช่ชั้นของ Transport Layer ในมาตรฐาน OSI Model การทำงานของ Host-to-Host Layer นี้จะมีการสร้าง Connection หรือการเชื่อมต่อกันระหว่างแอพพลิเคชันกับ Host – to – Host Layer โดยจุดที่เชื่อมกันเพื่อรับส่งข้อมูลที่เรียกว่า พอร์ท หรือ Socket (คำว่าพอร์ทในที่นี้ไม่ได้หมายถึง พอร์ททางฮาร์ดแวร์) และในแต่ละแอพพลิเคชันก็จะสร้างการเชื่อมต่อผ่านพอร์ทได้พร้อมกันหลายแอพพลิเคชัน ซึ่งการใช้งานพอร์ทของแต่ละแอพลิเคชันที่อยู่ในชั้น Process Layer จะแตกต่างกันตามหมายเลขที่กำหนดไว้ และแต่ละโพรโตคอลจะมีการใช้งาน port หมายเลขต่าง ๆ ไม่ซ้ำกัน ดังรูปที่ 2.34

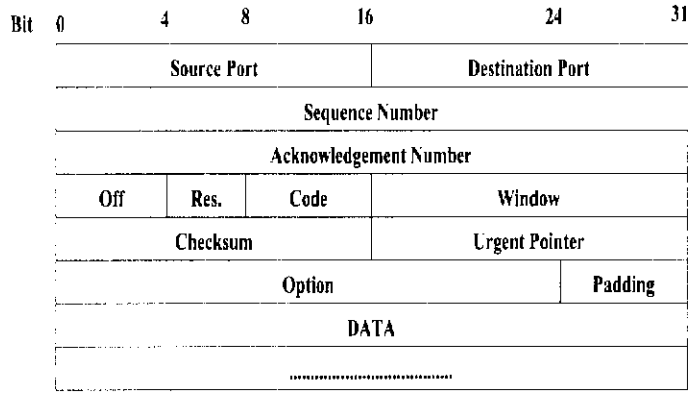


รูปที่ 2.34 แสดงการใช้งานพอร์ทของแต่ละโพรโตคอล

ในชั้น Host-to-Host หรือ Transport Layer ของ TCP/IP นี้จะมีโพรโตคอลทำงานอยู่ 2 โพรโตคอลที่แตกต่างกัน คือ โพรโตคอล TCP และโพรโตคอล UDP (User Datagram Protocol) ในการส่งผ่านข้อมูลลงไปที่ชั้นถัดๆ ไป เราจะเห็นว่าโพรโตคอล TCP และ UDP จะถูกผนึกเข้าไปในโพรโตคอล IP อีกทีหนึ่งและส่งต่อไปยังเครือข่ายอินเตอร์เน็ตต่อไป

2.4.4.1 โพรโตคอล TCP

โพรโตคอล TCP (Transmission Control Protocol) เป็นโพรโตคอลที่มีการรับส่งข้อมูลแบบ stream oriented protocol หมายความว่า การรับส่งข้อมูลจะไม่คำนึงถึงปริมาณข้อมูลที่จะส่งไป แต่จะแบ่งข้อมูลเป็นส่วนย่อย ๆ ก่อน แล้วจึงส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งสูญหายไป ก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้ง สำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูล datagram ดังนั้นแอพพลิเคชันหรือโปรเซ็สใดที่อาศัยการส่งผ่านข้อมูลด้วยโพรโตคอล TCP จะต้องใช้หน่วยความจำและขนาดของช่องสัญญาณมากกว่า UDP



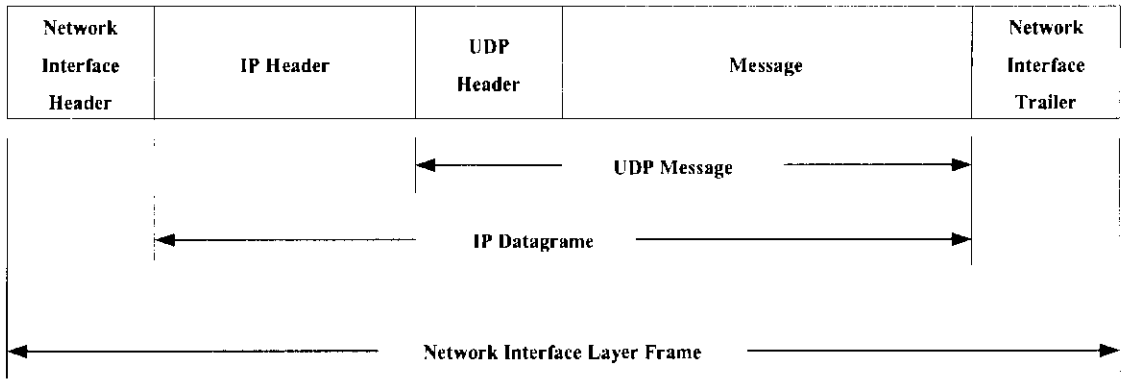
รูปที่ 2.35 แสดงโครงสร้างของโปรโตคอล TCP

2.4.4.2 โพรโตคอล UDP

ใน Host-to-Host Layer นอกจากจะมีโพรโตคอล TCP ทำงานแล้ว ก็ยังมีโพรโตคอล UDP (User Datagram Protocol) ในการส่งข้อมูลแต่ละครั้งและไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชันหรือโปรเซ็สใดที่ต้องอาศัยโพรโตคอล UDP จะเป็นแบบที่ทั้งสองด้านไม่จำเป็นต้องอาศัยการสร้างช่องทางเชื่อมต่อกัน (Connectionless) ระหว่างเครื่องเซิร์ฟเวอร์ให้บริการกับเครื่องที่ขอใช้บริการ โดยไม่ต้องแจ้งให้ฝ่ายรับข้อมูลเตรียมรับข้อมูลเหมือนโพรโตคอล TCP และไม่มีการตรวจสอบความถูกต้องครบถ้วนในการรับส่งข้อมูลนั้นๆ ด้วย เนื่องจากโพรโตคอล UDP ไม่มีสัญญาณสอบทานข้อมูล (acknowledgement) ในการส่งข้อมูลแต่ละครั้ง และไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชันหรือโปรเซ็สใดที่ต้องอาศัยโพรโตคอล UDP ในการส่งผ่านข้อมูลก็อาจจะต้องสร้างขบวนการตรวจสอบข้อมูลขึ้นมาเอง

UDP Message

UDP Message ประกอบด้วย UDP Header และ Message โดยจะนำไปรวมกับ IP Header เพื่อเข้าสู่ IP Datagram ซึ่งใช้ IP Protocol หมายเลข 17 (0x11) โดย message สามารถมีขนาดสูงสุดได้ถึง 65,535 ไบต์ และมีขนาดเล็กที่สุดเท่ากับ 65,507 ไบต์ โดยเล็กกว่า IP Header (20 ไบต์) และ UDP Header (8 ไบต์) IP Datagram ที่ได้มันจะเป็น encapsulated กับ Network Interface Layer ที่เหมาะสม



รูปที่ 2.36 UDP Message Encapsulation

UDP Header

มีขนาด 8 ไบต์ โดยประกอบด้วย 4 ส่วนดังรูปที่ 2.37

Source Port	Destination Port	Length	Checksum
2 ไบต์	2 ไบต์	2 ไบต์	2 ไบต์

รูปที่ 2.37 แสดงโครงสร้างของ UDP Header

- **Source Port** มี 2 ไบต์ใช้ระบุเป็น Source Application Layer Protocol ทำการส่ง UDP Message โดย Source Port เป็นพอร์ทที่ใช้ในการเลือก เมื่อใดที่ไม่ได้ใช้มัน มันจะตั้งค่าเป็น 0x00-00 IP Multicast Traffic เปรียบเสมือน Video casts ใช้ส่ง UDP สามารถใช้ค่า 0x00-00 เพราะจะไม่ตอบรับ Video Traffic เป็นเพียงการสมมุติ Application Layer ใช้ Source Port ในการนำ UDP Message เข้ามา Destination Port สำหรับการตอบรับ

- **Destination Port** มี 2 ไบต์ใช้ระบุเป็น Destination Application Layer Protocol การรวมของ Destination IP Address ของ IP Header และ Destination Port ของ UDP Header จะไม่เหมือนใครสำหรับกระบวนการที่จะส่งข้อมูล

- **Length** มี 2 ไบต์ที่ใช้ในการแสดงความยาวใน UDP Message มีความยาวน้อยที่สุด 8 ไบต์ (ขนาดของUDP Header) และมากที่สุด 65,515 ไบต์ (ค่าสูงสุด IP Datagram 65,535 ไบต์ น้อยกว่าค่าน้อยที่สุด IP Header 20 ไบต์) ความยาวมากที่สุดที่แท้จริงถูกจำกัดโดย MTU ซึ่งจะทำการเชื่อมโยงโดย UDP Message เป็นตัวส่ง ความยาว UDP สามารถคำนวณได้จากความยาวทั้งหมดและความยาวของ IP Header Field ใน IP Header

- **Checksum** มี 2 ไบต์ โดยจะทำการตรวจระดับของบิตอย่างสมบูรณ์สำหรับ UDP Message โดยที่ UDP Checksum คำนวณโดยใช้วิธีเดียวกันกับ IP Header Checksum

UDP Checksum

Checksum เป็น เลข 16 บิตถูกคำนวณด้วยวิธี 1's Complement โดยนำ Pseudo Header และข้อมูลทั้งหมดใน UDP Datagram มาคำนวณ

Pseudo Header เป็นข้อมูลที่อยู่ในส่วนของ IP Header ประกอบด้วยฟิลด์ Source IP Address, Destination IP Address, Zero, Protocol, UDP Length ดังแสดงในรูปที่ 2.38

16-bit Source IP address		
16-bit Destination IP address		
Zero	8-bit protocol (17 for UDP)	16-bit length

รูปที่ 2.38 Pseudo Header

หากค่า Checksum ที่คำนวณออกมาเป็น 0 ค่า checksum จะถูกเซตเป็น 1 ทั้งหมดแทน (มีค่าเท่ากันในระบบ 1's complement) ทั้งนี้เพราะในบางแอพพลิเคชั่นที่ไม่ต้องการตรวจสอบค่า Checksum ในระดับ UDP จะเซตค่านี้เป็น 0 (Disable Checksum)

2.4.5 Process Layer

การแสดงลำดับชั้นการทำงานของโพรโตคอล TCP/IP เทียบกับมาตรฐาน OSI model นั้น ในชั้นบนสุดเรียกว่า Process Layer ทำงาน 2 หน้าที่เทียบได้กับ Application Layer และ Presentation Layer ในชั้นนี้จะรองรับการทำงานของแอพพลิชันต่าง ๆ ที่ทำงานเป็นโปรเซ็ส อยู่ในเครื่องเซิร์ฟเวอร์ที่ให้บริการและเครื่องที่ขอใช้บริการ หรือไคลเอนต์ (Client) ซึ่งจะติดต่อกันผ่านโพรโตคอลเฉพาะแอพพลิเคชันอีกทีหนึ่ง ตัวอย่างเช่น เมื่อผู้ใช้งานอินเตอร์เน็ตต้องการโอนถ่ายไฟล์หรือ Download ข้อมูลจากเครื่องเซิร์ฟเวอร์ที่ให้บริการ โดยอาจจะเรียกใช้โปรแกรม FTP Client ทั่วไป เช่น โปรแกรม WS_FTP ติดต่อกับโปรเซ็ส FTP ที่กำลังให้บริการอยู่ที่เครื่องเซิร์ฟเวอร์ จากนั้นตัวโปรเซ็ส FTP ก็จะเรียกใช้โพรโตคอล FTP (File Transfer Protocol) เพื่อทำการโอนถ่ายไฟล์นี้ หรือถ้าผู้ใช้ต้องการเรียกใช้งานคอมพิวเตอร์ที่อยู่ห่างไกลออกไปด้วยการใช้โปรแกรม Telnet ที่เครื่องเซิร์ฟเวอร์ให้บริการ ตัวโปรเซ็ส Telnet ที่ทำงานอยู่ก็จะเรียกใช้โพรโตคอล Telnet เพื่อติดต่อกัน หรือในกรณีที่มีการเรียกใช้โปรแกรม Web Browser เช่น Netscape Navigator เพื่อเรียกดูเว็บไซท์ CNN ที่เครื่องซึ่งให้บริการเว็บของ CNN ก็จะมีโปรเซ็ส HTTP (Hypertext Transfer Protocol) ทำงานอยู่และจะติดต่อกับผู้ใช้ผ่านโพรโตคอล HTTP เป็นต้น

การทำงานของแอพพลิเคชันต่าง ๆ จะอยู่ที่ Process Layer นี้ และมีการติดต่อกันตามแต่ละโพรโตคอลเฉพาะแล้วแต่แอพพลิเคชันที่ใช้งาน จากการที่ Process Layer ของ TCP/IP รองรับให้โพรโตคอลอื่นทำงานได้หลายโปรเซ็สและหลายโพรโตคอลได้พร้อมกันนั้น ทำให้ผู้ใช้สามารถเปิดโปรแกรมใช้งานได้หลาย ๆ โปรแกรมพร้อมกัน เช่น เปิดโปรแกรม Internet Explorer เพื่อเรียกดูเว็บเพจพร้อมกับใช้งานโปรแกรม Outlook Express เพื่อรับส่งอีเมล์ไปพร้อมกันได้โดยไม่ต้องรอให้ทำงานอย่าง

หนึ่งอย่างใดเสร็จก่อน หรือในปัจจุบันมีการพัฒนาโปรแกรม Web Browser ให้สามารถเรียกใช้งานโพรโตคอลอื่น ๆ ได้มากขึ้น ทำให้เราสามารถใช้โปรแกรม Web Browser โอนถ่ายไฟล์ข้อมูลที่ใช้โพรโตคอล FTP ได้โดยไม่ต้องไปหาโปรแกรมอื่นมาใช้

โพรโตคอลหลัก ๆ ที่ทำงานใน Process Layer ซึ่งผู้ใช้มักจะคุ้นเคยกันดีได้แก่ FTP (File Transfer Protocol), Telnet, HTTP(Hyper Text Transfer Protocol), SMT(Simple Mail Transfer protocol) นอกจากนี้ยังมีโพรโตคอลอื่นที่อยู่เบื้องหลัง ซึ่งทำงานโดยที่ผู้ใช้ไม่ได้มีการใช้งานโดยตรง เช่น

- โพรโตคอล DNS (Domain Name System) ที่ทำหน้าที่แปลงข้อมูลชื่อ Domain Name หรือชื่อเว็บไซท์ทั้งหลายให้เป็นหมายเลข IP address
- โพรโตคอล DHCP (Dynamic Host Configuration Protocol) ทำหน้าที่แจกจ่ายข้อมูลพารามิเตอร์ของเครือข่ายให้กับเครื่องลูกข่ายที่เชื่อมต่ออยู่

สารบัญรูปภาพ

หน้า

สารบัญตาราง

หน้า

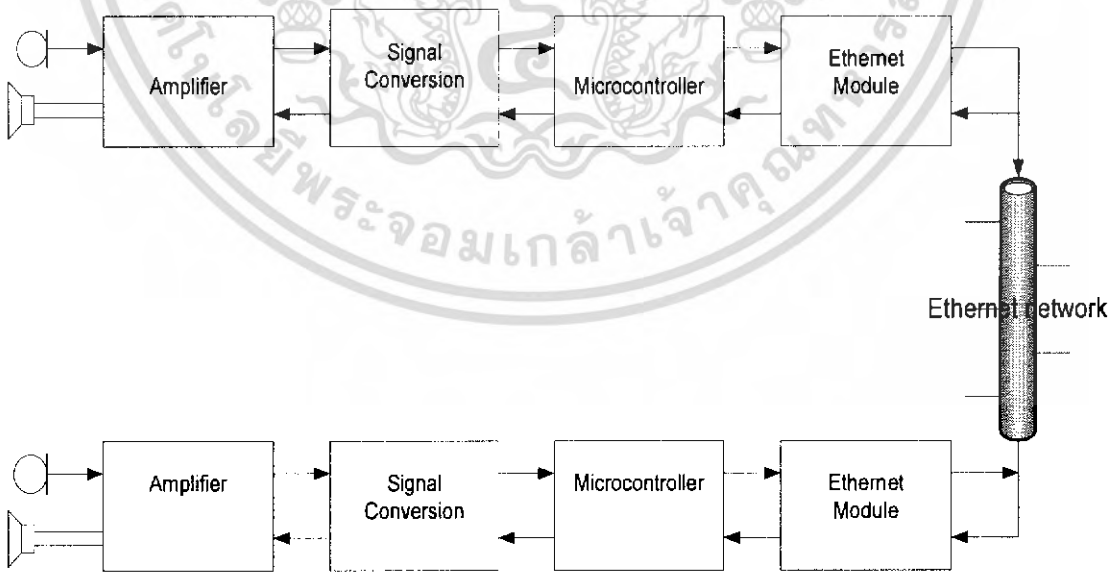
บทที่ 1
บทนำ

1.1 แนวคิดและที่มาของโครงงาน

การติดต่อสื่อสารเป็นสิ่งที่มีความสำคัญอย่างยิ่งในชีวิตคนเรา ตั้งแต่สมัยอดีตจนถึงปัจจุบัน โดยการสื่อสารที่ใช้กันมากที่สุดและสามารถเข้าใจกันได้ง่ายที่สุดนั้น คงหลีกเลี่ยงไม่พ้น การพูดคุยสนทนากันหรือการสื่อสารทางเสียงนั่นเอง

เมื่อในปัจจุบันระบบเครือข่ายอีเทอร์เน็ท (Ethernet) เป็นเทคโนโลยีสำหรับเครือข่ายแบบแลน (LAN) ที่ได้รับความนิยมสูงสุดและใช้งานกันอย่างกว้างขวาง โดยเฉพาะอย่างยิ่งความจำเป็นที่จะต้องใช้ข้อมูลร่วมกันและติดต่อสื่อสารกันระหว่างสำนักงาน ดังนั้นหากเราสามารถนำสัญญาณเสียงมาส่งผ่านระบบเครือข่ายอีเทอร์เน็ทเหมือนดังเช่นระบบเครือข่ายโทรศัพท์แล้ว ก็จะเป็นการดีในเรื่องของการประหยัดค่าใช้จ่ายและเป็นการช่วยเพิ่มประสิทธิภาพการใช้งานให้กับระบบเครือข่ายอีเทอร์เน็ทให้สามารถใช้ประโยชน์ได้มากขึ้นอีกด้วย

โครงงานการสื่อสารทางเสียงผ่านเครือข่ายอีเทอร์เน็ทนั้นจึงได้เกิดขึ้น โดยเป็นระบบที่นำเอาสัญญาณข้อมูลเสียงที่เป็นสัญญาณอะนาล็อก (Analog Signal) นำมาแปลงเป็นสัญญาณดิจิตอล (Digital Signal) แล้วส่งไปยังอีเทอร์เน็ตโมดูลที่ทำหน้าที่ในการเชื่อมต่อสัญญาณเข้ากับเครือข่ายอีเทอร์เน็ท นอกจากนี้แล้วยังใช้คอนโทรลเลอร์ (Controller)ในการควบคุมการทำงานของระบบ โดยระบบสามารถส่งสัญญาณเสียงได้แบบฟูลดูเพล็กซ์ (Full-Duplex Mode) เพื่อให้เกิดการสื่อสารที่มีประสิทธิภาพนั่นเอง ดังแสดงในรูปที่ 1.1



รูปที่ 1.1 บล็อกไดอะแกรมแสดงภาพรวมของโครงงาน

1.2 วัตถุประสงค์ของโครงงาน

สามารถสร้างระบบการส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ท ซึ่งโครงงานนี้ประกอบไปด้วยเอทูดีทำหน้าที่ในการแปลงสัญญาณจากอะนาล็อกเป็นดิจิตอล และอีเทอร์เน็ทโมดูลที่ทำหน้าที่ในการเชื่อมต่อสัญญาณเข้ากับโครงข่าย นอกจากนี้ยังมีคอนโทรลเลอร์ควบคุมการทำงานของระบบ โดยระบบสามารถส่งสัญญาณเสียงได้แบบฟูลดูเพล็กซ์

1.3 องค์ประกอบหลักของโครงงาน

1.3.1 การสื่อสารบนระบบเครือข่ายอีเทอร์เน็ท

ในส่วนที่เกี่ยวข้องกับการสื่อสารบนระบบเครือข่ายอีเทอร์เน็ท คือ ส่วนที่เป็นอีเทอร์เน็ทโมดูล ซึ่งมีหน้าที่ในการเชื่อมต่อสัญญาณเข้ากับโครงข่าย โดยมีไมโครคอนโทรเลอร์ทำหน้าที่ควบคุมการทำงาน

1.3.2 วงจรอิเล็กทรอนิกส์ (Electronic circuits)

เป็นที่แน่นอนว่าในการออกแบบระบบนั้นจะต้องมีการออกแบบวงจรเพื่อใช้ในระบบ ในโครงงานนี้ได้ทำการออกแบบและสร้างวงจรต่างๆ อันได้แก่ วงจรขยายสัญญาณ วงจรแปลงสัญญาณ และอีเทอร์เน็ทโมดูล

1.3.3 การใช้ไมโครคอนโทรลเลอร์

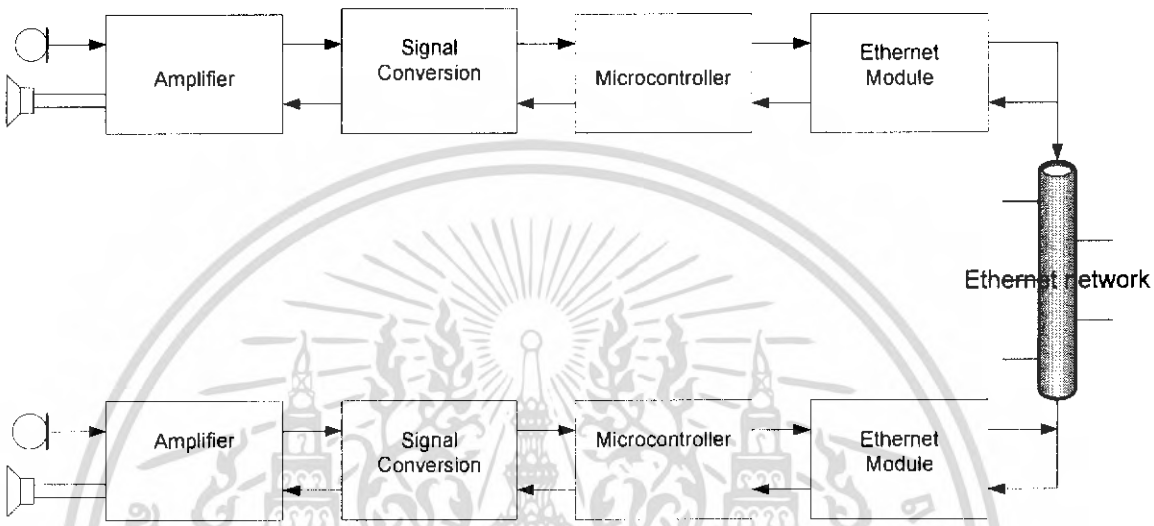
ในโครงงานงานนี้ได้มีการใช้ไมโครคอนโทรลเลอร์ตระกูล MCS-51 เบอร์ AT89C52 สำหรับควบคุมอุปกรณ์ต่างๆ ในระบบ

1.3.4 การเขียนโปรแกรมคอมพิวเตอร์

การเขียนโปรแกรมเพื่อใช้ในการควบคุมไมโครคอนโทรลเลอร์นั้น ใช้โปรแกรมภาษาซี ซึ่งจะแบ่งโปรแกรมออกเป็น 2 ส่วน คือ ส่วนแรกเป็นส่วนที่ใช้ในการควบคุมการแปลงสัญญาณของเอทูดีและดีทูเอ และส่วนที่สองเป็นส่วนที่ใช้ในการควบคุมการรับส่งข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์ผ่านเครือข่าย

บทที่ 2
ทฤษฎีและหลักการ

ในส่วนของบทที่ 2 นี้ จะกล่าวถึงทฤษฎีและหลักการที่เกี่ยวข้องกับองค์ประกอบหลักของโครงงาน ซึ่งจะเป็นการอธิบายเพื่อแสดงรายละเอียดต่างๆ เพื่อให้เข้าใจเกี่ยวกับโครงงานนี้มากขึ้น ซึ่งจะขออธิบายส่วนต่างๆ ตามภาพรวมของระบบดังแสดงดังรูปที่ 2.1

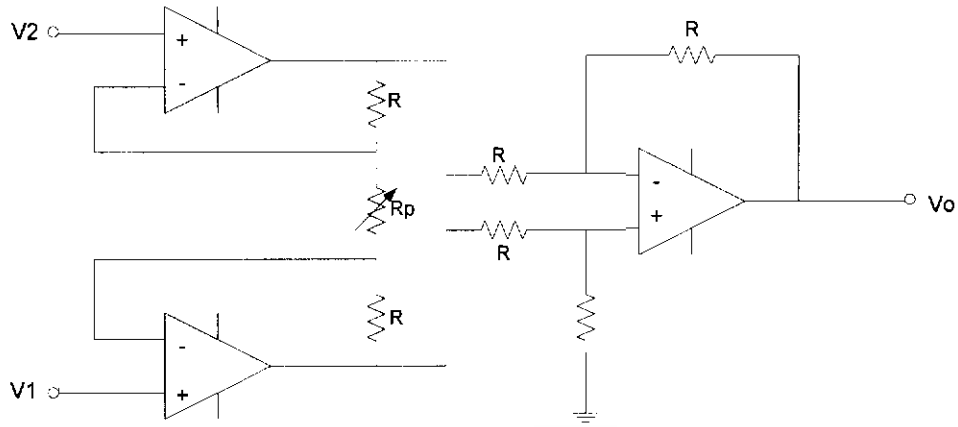


รูปที่ 2.1 ภาพรวมของระบบการส่งเสียงผ่านเครือข่ายอีเทอร์เน็ท

2.1 วงจรขยาย (Amplifier)

2.1.1 วงจรขยายในเครื่องมือวัด (Instrumentation Amplifier)

จากวงจรออปแอมป์พื้นฐานได้นำมาสู่วงจรวงจรขยายในเครื่องมือวัด วงจรขยายในเครื่องมือวัดเป็นวงจรขยายที่ได้รับความนิยมใช้กันมากในระบบการวัดและระบบควบคุม เพราะว่าสามารถปรับอัตราการขยายได้ ทั้งนี้เนื่องจากในระบบการวัดระบบหนึ่งอาจมีการเชื่อมต่อเพื่อรับสัญญาณจากตัวตรวจจับสัญญาณที่ต่างชนิดกัน ซึ่งสัญญาณที่ได้จากตัวตรวจจับสัญญาณหลายประเภทมักจะมีช่วงของระดับสัญญาณที่แตกต่างกันไป ดังนั้นการปรับอัตราการขยายได้จะทำให้ทุก ๆ ช่องสัญญาณถูกปรับให้เอาท์พุทมีระดับแรงดันที่เท่ากันทำให้สะดวกในการนำไปเชื่อมต่อกับวงจรอื่นต่อไป



รูปที่ 2.2 วงจรขยายอุปกรณ์วัดแบบปรับอัตราการขยายได้

ความต้านทานของโพเทนชิโอมีเตอร์หรือ Rp จะเป็นตัวปรับค่าตัวประกอบสเกล ดังนี้

$$\frac{V_O}{V_1 - V_2} = 1 + \frac{2R}{R_P}$$

$$V_O = (1 + \frac{2R}{R_P})(V_1 - V_2) = k(V_1 - V_2)$$

คุณสมบัติที่ดีของวงจรขยายในเครื่องมือวัดดังกล่าว คือ

1. ความต้านทานทางด้านอินพุทมีค่าสูงมาก และไม่เปลี่ยนแปลงตามอัตราการขยาย ในขณะที่ออปแอมป์ธรรมดาจะเปลี่ยนแปลงเสมอ

2. แรงดันเอาท์พุทที่ได้ไม่ขึ้นอยู่กับแรงดันคอมมอนที่มาจากทั้ง V1 และ V2 เลย แต่จะเป็นผลที่เกิดขึ้นกับค่าแรงดันดิฟเฟอร์เรนเชี่ยลระหว่าง V1 กับ V2 เท่านั้น

3. มีอัตราการขยายรวมสูง ทำให้สามารถนำไปใช้กับสัญญาณอินพุทที่มีระดับต่างกันได้อย่างกว้างขวาง

2.1.2 วงจรขยายกำลังงาน (Power Amplifier)

คุณสมบัติของวงจรขยายกำลังงาน

จุดประสงค์ในการออกแบบวงจรขยายกำลังงานทุกประเภท ต้องการให้เป็นดังต่อไปนี้

1.ประสิทธิภาพสูง เนื่องจากจะส่งผลกับวงจรภาคจ่ายไฟเลี้ยง เพราะถ้าวงจรขยายกำลังมีประสิทธิภาพสูงการใช้พลังงานจะเป็นไปอย่างคุ้มค่า ดังนั้นเมื่อต้องการกำลังงานที่เท่าถัน วงจรที่มีประสิทธิภาพสูงกว่าจะใช้กำลังงานจากแหล่งจ่ายไฟเลี้ยงน้อยกว่า การออกแบบแหล่งจ่ายไฟเลี้ยงจึงทำได้ง่ายกว่า ราคาก็จะถูก และอีกสาเหตุหนึ่งคือถ้าวงจรมีประสิทธิภาพสูงจะมีการสูญเสียกำลังงานเป็นความร้อนต่ำ จึงส่งผลต่อการระบายความร้อนโดยใช้แผ่นระบายความร้อน(Heatsink) ขนาดเล็ก ซึ่งแผ่นระบาย

ความร้อนนับเป็นอุปกรณ์อีกชิ้นที่มีราคาแพง เมื่อต้องระบายความร้อนต่ำก็จะใช้แผ่นระบายความร้อนขนาดเล็ก จึงประหยัดราคาในส่วนนี้ได้ และประสิทธิภาพในการขยายกำลังงานจะมีนิยมอยู่ 3 แบบ ดังนี้

1. การวัดประสิทธิภาพที่ขาเดรน (Drain) หรือขาคอลเลกเตอร์ (Collector) เป็นประสิทธิภาพของวงจรกำลังขยายกำลังในจุดที่ให้สัญญาณเอาท์พุท คำนวณได้ตามสมการ (2.1)

$$\eta = \frac{P_{RF(out)}}{P_{DC(in)}} \times 100 \qquad (2.1)$$

เมื่อ η คือประสิทธิภาพของวงจรขยายกำลัง(เปอร์เซ็นต์)

$P_{RF(out)}$ คือกำลังงานของสัญญาณเอาท์พุทที่ผ่านการขยายกำลัง(วัตต์)

$P_{DC(in)}$ คือกำลังงานที่แหล่งจ่ายไฟป้อนให้วงจรขยายกำลัง(วัตต์)

2. การวัดประสิทธิภาพของวงจรเมื่อรวมการคำนวณค่าพลังงานสัญญาณอินพุท ในกรณีที่สัญญาณอินพุทที่เข้าสู่ภาคการขยายกำลังมีกำลังงานสูง กำลังงานของสัญญาณอินพุทจะถือเป็นกำลังงานสูญเสียที่ต้องจ่ายให้กับวงจรขยายกำลังด้วย การวัดประสิทธิภาพดังนิยามแรกจะไม่สมเหตุสมผล ดังนั้นจึงควรคำนวณโดยลบค่ากำลังงานอินพุทที่ถือเป็นกำลังที่ต้องสูญเสียให้วงจรออกจากกำลังงานเอาท์พุทดังสมการ (2.2)

$$\eta = \frac{\left(P_{RF(out)} - P_{RF(in)}\right)}{P_{DC(in)}} \times 100 \qquad (2.2)$$

เมื่อ $P_{RF(in)}$ คือกำลังงานของสัญญาณอินพุท(วัตต์)

3.การวัดประสิทธิภาพโดยรวม (Overall efficiency) เป็นค่าประสิทธิภาพโดยรวมของวงจรขยายกำลังทั้งระบบ โดยรวมทุกภาคการขยายซึ่งเป็นค่าประสิทธิภาพที่ใช้ตัดสินวงจรขยายกำลังได้ดีที่สุด และคำนวณได้ตามสมการ (2.3)

$$\eta = \frac{P_{RF(out)}}{(P_{DC(in)} + P_{RF(in)})} \times 100 \qquad (2.3)$$

ซึ่งค่าประสิทธิภาพของวงจรการขยายกำลังแบบต่างๆมีค่าโดยประมาณดังตารางที่ 2.1

ตารางที่ 2.1 ประสิทธิภาพของวงจรขยายกำลังคลาสต่างๆ

ประเภทของคลาส	ประสิทธิภาพสูงสุด ตามทฤษฎี(η_{max})	ค่าประมาณประสิทธิภาพ เมื่อใช้งานจริง
เชิงเส้น(Linear)		
คลาส A	50%	40%
คลาส B	78.5%	65%
คลาส AB	50%<η<78.5%	60%
แบบสวิทชิ่ง (Switching) คลาส D , กลุ่มคลาส E	100%	95%

2.ค่าตัวเลขสัญญาณรบกวน (Noise figure : NF) ต่ำ เนื่องจากสัญญาณรบกวนรวมทั้งหมดที่ปรากฏออกมาที่เอาท์พุทของวงจรขยายกำลัง ประกอบด้วยสองส่วนคือ สัญญาณรบกวนเอาท์พุตที่เกิดจากการขยายกำลังสัญญาณรบกวนอินพุท และสัญญาณรบกวนเอาท์พุทที่เกิดจากวงจรขยายกำลังเอง ดังนั้นค่าตัวเลขสัญญาณรบกวนจึงเป็นค่าที่บอกว่าระบบหรือการขยายกำลังนั้นมีการก่อกำเนิดสัญญาณรบกวนจากภายในตัวเครื่องขยาย ออกมาปนกับสัญญาณที่ต้องการมากหรือน้อยอย่างไร โดยคำนวณได้ตามสมการ (2.4)

$$NF = \frac{\overline{n_{T_0}^2(t)}}{\overline{n_{S_0}^2(t)}} \qquad (2.4)$$

เมื่อ $\overline{n_{T_0}^2(t)}$ เป็นกำลังเฉลี่ยรวมทั้งหมดของสัญญาณรบกวนที่ปรากฏออกมาที่เอาท์พุทของเครื่องขยายสัญญาณ

$\overline{n_{S_0}^2(t)}$ เป็นกำลังเฉลี่ยของสัญญาณรบกวนที่เป็นส่วนหนึ่งของสัญญาณรบกวนเอาท์พุท ซึ่งเกิดจากการขยายสัญญาณรบกวนอินพุท

หรือสามารถพิสูจน์ให้ง่ายต่อการวัดค่าเป็นดังสมการ (2.5)

$$NF = \frac{\left[\frac{S}{N}\right]_i}{\left[\frac{S}{N}\right]_o} \qquad (2.5)$$

เมื่อ $\left[\frac{S}{N}\right]_i$ คือ ค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวนของสัญญาณอินพุท

$\left[\frac{S}{N}\right]_o$ คือ ค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวนของสัญญาณที่ผ่านการขยาย

โดยอัตราส่วนสัญญาณต่อสัญญาณรบกวน(Signal to Noise Ratio : SNR) เป็นค่าที่บอกความแตกต่างของกำลังงานข่าวสารที่ต้องการ กับกำลังงานของสัญญาณรบกวน โดยถ้ามีค่ามากจะดีเพราะแสดงว่าสัญญาณข่าวสารมีกำลังงานสูงเมื่อเปรียบเทียบกับ กำลังงานของสัญญาณรบกวนและจะคำนวณค่าได้ตามสมการ (2.6)

$$\frac{S}{N} = \frac{\overline{s^2(t)}}{\overline{n^2(t)}} \tag{2.6}$$

เมื่อ $\frac{S}{N}$ คือค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวน

$\overline{s^2(t)}$ คือค่ากำลังเฉลี่ยของสัญญาณข่าวสาร

$\overline{n^2(t)}$ คือค่ากำลังเฉลี่ยของสัญญาณรบกวน

ค่าเอสเอ็นอาร์นี้ปรกติมักแสดงค่าในหน่วยเดซิเบล (Decibel : dB) คำนวณได้ตามสมการ (2.7)

$$\left.\frac{S}{N}\right|_{dB} = 10\log\left(\frac{\overline{s^2(t)}}{\overline{n^2(t)}}\right) \tag{2.7}$$

3.ความเพี้ยนฮาร์มอนิกรวม (Total harmonic distortion : THD) ***ต่ำ*** เป็นค่าที่บอกว่าเมื่อนำสัญญาณไซน์รูปสมบูรณ์ (Pure sinusoidal wave) ป้อนเข้าสู่การขยายกำลังงานแล้วมีความถี่ของสัญญาณอื่นที่เป็นจำนวนเท่าของความถี่ข่าวสาร หรือเรียกว่าความถี่ฮาร์โมนิกเกิดขึ้นที่เอาท์พุทหรือไม่ เพราะสัญญาณฮาร์โมนิกเป็นสัญญาณที่ไม่ต้องการเนื่องจากเมื่อรวมกับข่าวสารเดิมจะทำให้รูปสัญญาณผิดไป ค่า THD คำนวณได้ตามสมการ (2.8) หรือคำนวณตามสมการ (2.8) แล้วคูณ 100 เพื่อทำให้เป็นหน่วยเปอร์เซ็นต์

$$THD = \frac{\sum_{n=2}^{\infty} c_n^2}{c_1^2} \tag{2.8}$$

เมื่อ c_1 คือค่าแอมปลิจูดของสัญญาณข่าวสาร

c_n คือค่าแอมปลิจูดของสัญญาณฮาร์โมนิกที่ n ของสัญญาณข่าวสาร

4. มีแบนด์วิดท์และอัตราการขยายกำลังเหมาะสมกับประเภทของงาน เนื่องจากวงจรขยายกำลังใช้กับงานหลายประเภท ซึ่งมีความถี่ที่ต้องการขยายแตกต่างกัน เช่น การขยายกำลังงานสำหรับเครื่องเสียง ก็ต้องขยายเสียงในช่วงความถี่ที่หูคนได้ยิน(ประมาณ 10 Hz ถึง 25 Hz) หรือถ้าเป็นเครื่องขยายกำลังเพื่อส่งข่าวสารไปในช่องสัญญาณ ก็ต้องการขยายกำลังของสัญญาณที่ผ่านการมอดูเลตมา ซึ่งส่วนมากจะเป็นความถี่สูงกว่าความถี่เสียง ดังนั้นการขยายกำลังต้องคำนึงถึงผลการตอบสนองทางความถี่ (Frequency response) ของวงจรด้วยว่าวงจรขยายกำลังจะต้องมีผลตอบสนองคงที่ตลอดย่านความถี่ที่ใช้งาน หรือต้องไม่ตัดองค์ประกอบความถี่ของข่าวสารเพื่อให้สัญญาณภายหลังการขยายเหมือนเดิม ส่วนความถี่นอกย่าน

ที่ต้องการขยายกำลังก็เปรียบเสือนสัญญาณรบกวน ดังนั้นจึงไม่ควรขยายความถี่ที่อยู่นอกแบนด์วิดท์ของสัญญาณข่าวสาร

5. ราคาถูกเมื่อเทียบกับคุณภาพที่ได้ เป็นอีกปัจจัยหนึ่งที่ต้องคำนึงในการออกแบบ เพื่อความคุ้มค่าของผู้ซื้อและความสามารถในการแข่งขันกับวงจรขยายเสียงที่มีอยู่แล้ว โดยราคาเป็นผลจากเทคนิคที่ใช้ในการขยายกำลัง รวมถึงการเลือกคุณภาพของอุปกรณ์ที่ใช้ในวงจร และขั้นตอนในการสร้างทั้งหมด เช่น ถ้าออกแบบให้วงจรมีประสิทธิภาพสูง ความร้อนที่เกิดก็จะน้อย แผ่นระบายความร้อนที่มีราคาแพงก็จะใช้ไม่มาก ราคาก็จะถูกลง เป็นต้น ซึ่งราคาและคุณภาพจะเป็นตัวชี้วัดคุณค่าและความนิยมของวงจรกำลังขยายนั้นๆ กล่าวคือถ้ามีสองวงจรที่มีคุณภาพเท่ากัน วงจรที่ราคาถูกจะได้รับความนิยมกว่า

จากคุณสมบัติของวงจรขยายกำลังงานเสียงต่างๆ เหล่านี้ ต้องคำนึงถึงลำดับความสำคัญในการออกแบบให้เหมาะสมกับแต่ละประเภทงาน เพราะบางงานอาจไม่จำเป็นต้องใช้คุณภาพดีมากเพราะจะทำให้ราคาแพงเกินความจำเป็น ดังนั้นวงจรขยายกำลังจึงมีจำหน่ายหลายคุณภาพและหลายราคา

2.2 การแปลงสัญญาณ (Signal Conversion)

2.2.1 การแปลงสัญญาณอะนาล็อก - ดิจิตอล

สัญญาณที่ใช้ในอุปกรณ์อิเล็กทรอนิกส์ มี 2 ชนิด คือ สัญญาณอะนาล็อก และสัญญาณดิจิตอล สัญญาณอะนาล็อก จะใช้ในอุปกรณ์ทั่วๆ ไป และใช้ในการควบคุมแบบเก่า

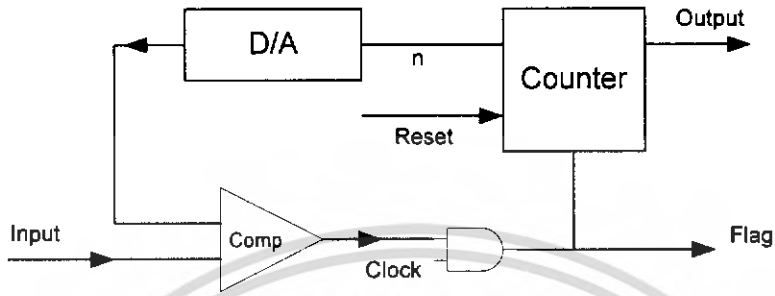
ในปัจจุบันมีไมโครโปรเซสเซอร์ และไมโครคอนโทรลเลอร์ เข้ามาช่วยในการควบคุมอุปกรณ์ต่างๆ มากมาย ซึ่งทำให้การควบคุมนั้นทำได้ง่าย และรวดเร็วยิ่งขึ้น แต่ในการควบคุมนั้น เราจำเป็นต้องใช้สัญญาณดิจิตอลในการติดต่อกับไมโครโปรเซสเซอร์ หรือไมโครคอนโทรลเลอร์ แต่ในความเป็นจริงนั้นเราใช้สัญญาณอะนาล็อกในการควบคุม ดังนั้นเราจึงจำเป็น ต้องมีการเปลี่ยนสัญญาณอะนาล็อก เป็นสัญญาณดิจิตอล แล้วจึงนำสัญญาณนั้นเข้ามาสู่ไมโครโปรเซสเซอร์ หรือไมโครคอนโทรลเลอร์ เพื่อใช้ควบคุมระบบต่อไป

แม้ว่าสัญญาณอะนาล็อกนั้นมีความแน่นอน และแม่นยำสูง แต่สัญญาณอะนาล็อกนั้นก็ควบคุมได้ยาก เนื่องจากในสภาพแวดล้อม มีสัญญาณรบกวนอยู่มาก และการที่จะทำให้ การควบคุมแบบอะนาล็อก มีความสามารถควบคุม เท่ากับการควบคุมแบบดิจิตอลนั้น ทำได้ยาก เนื่องจากวงจรควบคุมแบบ อะนาล็อกจะต้องมีความซับซ้อนสูง

อย่างไรก็ตาม สัญญาณดิจิตอลก็ไม่สามารถทดแทนความละเอียดของสัญญาณอะนาลือกได้อย่างสมบูรณ์ แต่ทำให้การควบคุมนั้นทำให้ง่าย และสะดวกยิ่งขึ้น

Counting Converter

Counting Converter เป็นวิธีที่ง่ายที่สุดของการแปลงสัญญาณอะนาล็อก เป็นสัญญาณดิจิตอล โดยใช้อัลกอริทึม การนับค่าเพิ่มขึ้นเรื่อยๆ แล้วนำผลที่ได้จากการนับไปเปรียบเทียบกับค่าที่ต้องการที่ตั้งไว้ ลักษณะการทำงานเป็นดังรูปที่ 2.3



รูปที่ 2.3 Analog to digital converter

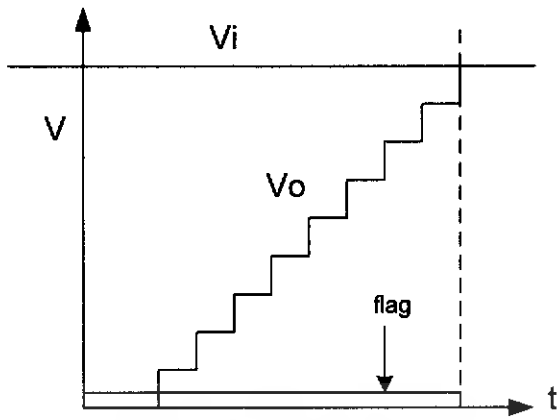
จากวงจร Counter เป็นอุปกรณ์นับค่าที่เพิ่มขึ้นทีละหนึ่ง แล้วส่งค่าที่ได้ให้ D/A มีขา Reset รับสัญญาณ Reset เมื่อต้องการให้เริ่มนับใหม่

D/A เมื่อรับค่าที่นับเพิ่มขึ้นทีละหนึ่งจากตัวนับ ก็แปลงค่าให้เป็นสัญญาณ อะนาล็อกที่มีค่าความต่างศักย์ค่าๆ หนึ่ง แล้วส่งต่อเข้าไปที่อุปกรณ์ตัวเปรียบเทียบ(Comparator)

Comparator จะเป็นอุปกรณ์ตัวเปรียบเทียบค่าความต่างศักย์ ของอินพุต และค่าจากที่ตัวนับ ถ้าหากทั้งสองสัญญาณมีค่าเท่ากันส่งค่าความต่างศักย์ 0 โวลต์ออกมา(ลอจิก 0) ถ้าไม่เท่ากันก็จะส่งความต่างศักย์ที่ไม่ใช่ 0 โวลต์ออกมา(ลอจิก 1)

ซึ่งค่าความต่างศักย์ที่ออกมา จะนำมาเข้าลอจิกเกต "และ" กับ สัญญาณนาฬิกา จะได้ค่าลอจิกออกมา ถ้าผลลัพธ์ออกมาเป็นสัญญาณนาฬิกาแสดงว่ายังไม่ได้ผลลัพธ์เท่าที่ต้องการ สัญญาณนาฬิกาก็จะไปทำให้ตัวนับนับเพิ่มขึ้นต่อไป และเมื่อได้ค่าผลลัพธ์ดิจิตอลที่ต้องการแล้ว ค่าที่ได้จาก ตัวเปรียบเทียบจะให้ค่าความต่างศักย์เป็น 0 (ลอจิก 0) ซึ่งเมื่อนำมาเข้าลอจิกเกต "และ" กับสัญญาณนาฬิกาแล้ว ก็จะให้ลอจิก 0 ซึ่งทำให้ตัวนับไม่นับเพิ่มอีก ก็จะได้ค่าดิจิตอลจากตัวนับที่ต้องการ

จากคำอธิบายข้างต้นจะได้กราฟของ V_o ดังรูปที่ 2.4

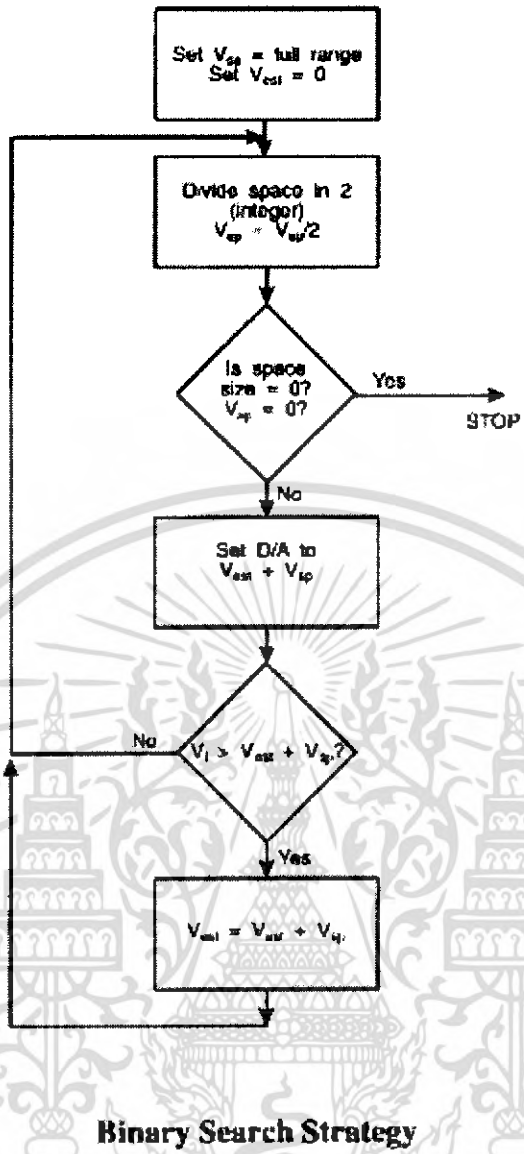


รูปที่ 2.4 Output voltage Graph from A/D converter

ข้อเสียของวิธีนี้ คือ การนับต้องเริ่มนับที่ 0 เสมอ และนับเพิ่มขึ้นเรื่อยๆ ทำให้ช้า เอาท์พุทที่ได้จะมี delay จึงไม่ค่อยนิยมใช้เท่าที่ควร จึงได้เปลี่ยนตัวนับเป็นแบบนับลงได้ด้วย ซึ่งจะอ้างอิงระดับจากระดับเก่า ทำให้ไม่จำเป็นต้องนับ 0 ใหม่ เมื่อมีการเปลี่ยนอินพุทใหม่ แต่ให้อ้างอิงกับผลลัพธ์เดิม ทำให้ได้ผลลัพธ์เร็วขึ้น

Successive Approximation

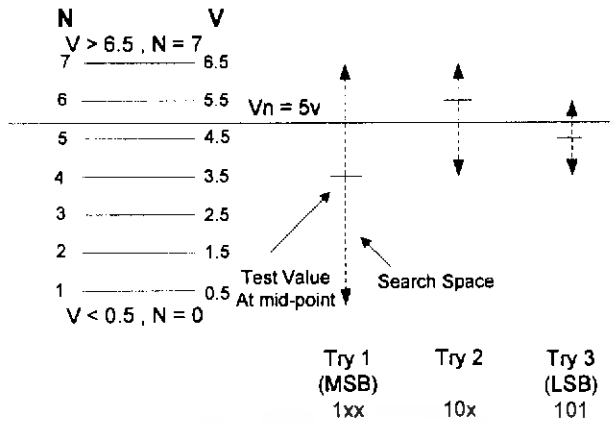
ใช้หลักการของ "binary search" ในการหาคำตอบ โดยนำค่าผลลัพธ์มาเปรียบเทียบกับค่ากึ่งกลางของช่วง เพื่อให้ทราบว่า ค่านั้นๆ มากกว่า หรือน้อยกว่า โดยจะปรับช่วงให้แคบลงมาเรื่อยๆ แล้วเปรียบเทียบผลลัพธ์กับค่ากึ่งกลางของช่วงไปเรื่อยๆ จนได้ผลลัพธ์ที่ต้องการ เช่น เลขที่เป็นคำตอบคือ 3 จากช่วงของคำตอบที่ 0-7 ครั้งแรกเอาค่า (0+7)/2 = 4 มาเปรียบเทียบ ได้ผลว่า คำตอบที่ต้องการอยู่ในช่วงที่น้อยกว่า 4 ครั้งที่ 2 ก็เลือกค่า (0+4)/2 = 2 มาเปรียบเทียบ ได้ผลว่าคำตอบที่ต้องการอยู่ในช่วงที่มากกว่า 2 แต่น้อยกว่า 4 ครั้งที่ 3 ก็เลือกค่า (2+4)/2 = 3 มาเปรียบเทียบ ได้ผลว่าคำตอบที่ต้องการ จากหลักการที่กล่าวมาอาจเขียน flow chart ได้ดังรูปที่ 2.5



รูปที่ 2.5 flowchart แสดงหลักการของ "binary search" ในการหาคำตอบ

ข้อดีของวิธีนี้ คือ เวลาที่ใช้ในการหาคำตอบ n รอบ แน่นอน (สำหรับ n bit converter ซึ่งอ้างอิงได้ 2^n ระดับ และระดับ V_{in} ที่คงที่) ซึ่งใช้เวลาน้อยกว่าแบบ "Counting Algorithm"

แต่มีข้อเสีย คือถ้า V_{in} เปลี่ยนทันทีทันใด ขณะที่กำลังทำ binary search อยู่นั้น คำตอบที่ได้จะผิดพลาด ตัวอย่างเช่น เปลี่ยน V_{in} จาก 5 Volt เป็น 2 Volt



รูปที่ 2.6 กราฟแสดงหลักการ Binary Search

ช่วงของ V_{in} คือ 1-7 ใช้ n=3 (เพราะว่า 2^3=8)

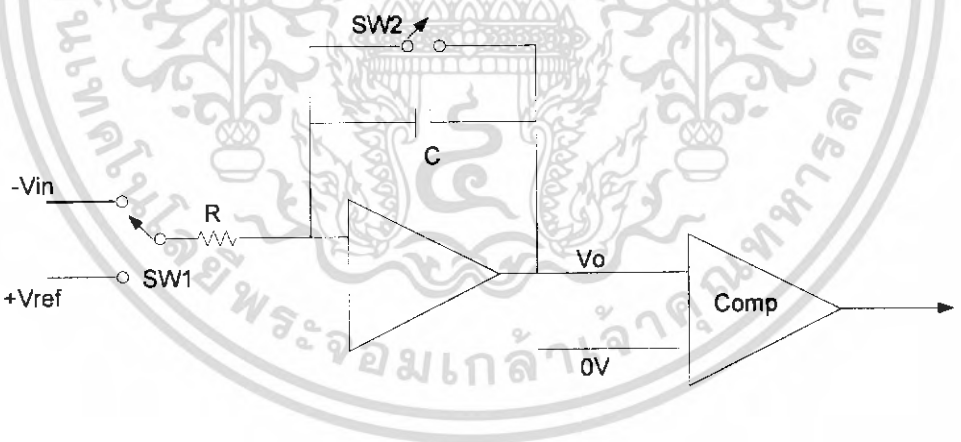
ครั้งแรก ใช้ 4 เปรียบเทียบกับ V_{in} (ซึ่งเท่ากับ 5 โวลต์) พบว่า อยู่ในช่วง lower ได้ 1xx

ครั้งที่ 2 ใช้ 2 เปรียบเทียบกับ V_{in} (ซึ่งเท่ากับ 5 โวลต์) พบว่า อยู่ในช่วง upper ได้ 10x

ครั้งที่ 3 ใช้ 3 เปรียบเทียบกับ V_{in} (ซึ่งเท่ากับ 5 โวลต์) พบว่า ผลลัพธ์ที่ได้จะผิดพลาด ได้ 100

Dual-Slope ADC

ใช้หลักการของวงจร Integrator ทำงานร่วมกับตัว Comparator ดังรูปที่ 2.7



รูปที่ 2.7 Dual Slope A/D converter

Input Voltage มี 2 ตัว คือ ค่าความต่างศักย์อะนาล็อกที่ต้องการแปลงเป็นดิจิตอล ($-V_{in}$) และความต่างศักย์ที่คงที่ค่าหนึ่ง (V_{ref}) และมีสวิตช์ SW1 ซึ่งทำหน้าที่เลือกค่าสัญญาณ

จากวงจรตอนเริ่มต้นสวิตช์ SW2 ทำหน้าที่คายประจุของตัวเก็บประจุ C แล้วจึงเปิด SW2 ออก

เมื่อสวิตช์ SW1 สับมาที่ $-V_{in}$ จากวงจร Integrator จะพิสูจน์สมการได้ดังนี้

$$I = C\frac{dV_o}{dt}$$

$$-V_{in} + iR - V_o + V_o = 0$$

$$-V_{in} + RC\frac{dV_o}{dt} = 0$$

$$V_{in} = RC\frac{dV_o}{dt}$$

$$\int dV_o = \int \frac{V_{in}}{RC} dx$$

$$V_o = \frac{V_{in}(t)}{RC}$$

slope มีค่าเท่ากับ $\frac{V_{in}}{RC}$

ค่า t ที่ใช้มีค่าคงที่ t_m

เมื่อ t เพิ่มจากศูนย์ถึง t_m ให้SW1 สับไปที่ V_{ref}

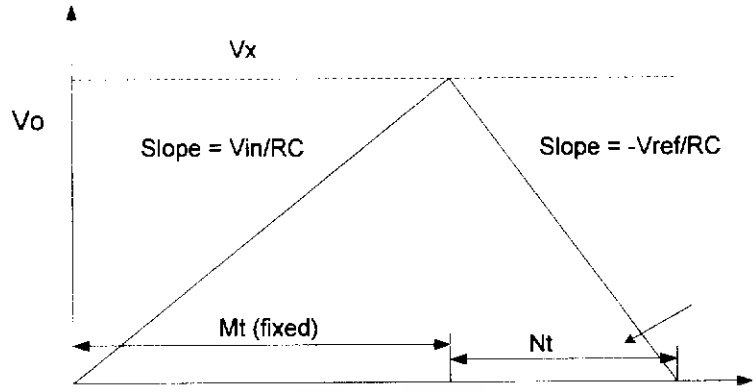
จะได้ดังสมการที่ (2.9)

$$V_o = \frac{V_{ref}(t)}{RC} \qquad (2.9)$$

slope มีค่า $\frac{V_{ref}}{RC}$

สมมติ ช่วงเวลาตั้งแต่ความต่างศักย์ที่ t_m จนความต่างศักย์เป็น 0 มีค่าเท่ากับ t_n

ได้ดังแสดงในกราฟรูปที่ 2.8



รูปที่ 2.8 กราฟ Dual Slope A/D Converter Output and Timing

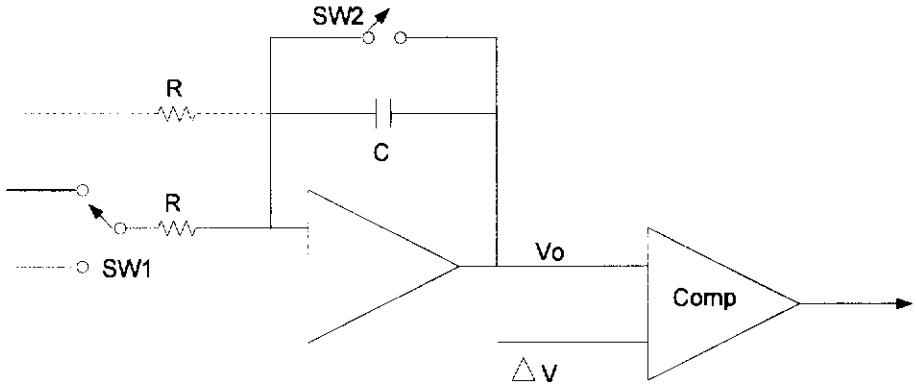
จากหลักของสามเหลี่ยมคล้าย จะได้ดังสมการที่ (2.10)

$$V_{in} = V_{ref} \frac{t_n}{t_m} \qquad (2.10)$$

เนื่องจาก V_{ref} และ t_n มีค่าคงที่ สัญญาณอะนาล็อกขึ้นกับค่า t_n เพราะการควบคุมการเปลี่ยนสัญญาณดิจิตอล ที่ขึ้นกับค่า t_n

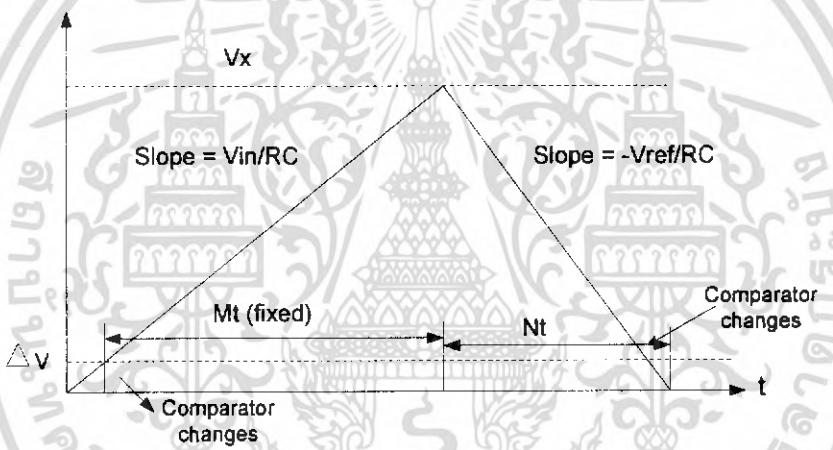
การแปลงเป็นสัญญาณดิจิตอลจะทำโดยจับคู่ค่า t_n กับเอาท์พุทค่าๆ หนึ่ง ตามความเหมาะสม สำหรับ V_{ref} นั้นๆ เหมือนการเทียบค่าในตาราง

ความเร็วของการแปลงสัญญาณแบบนี้ ขึ้นอยู่กับ V_{in} และ Slope ของวงจร integrator โดยธรรมชาติแล้ว ลักษณะของตัวเปรียบเทียบเองนั้น จะไม่เป็นอุดมคติ คือจะมีผลต่างของความต่างศักย์อยู่ V โวลต์ แม้ว่าจะต่ออินพุททั้งสองลงกราวด์แล้วก็ตาม ซึ่งถ้า V_{ref} ที่ใช้อยู่นั้นมีค่าน้อยกว่าค่าผลต่างของความต่างศักย์ที่เกิดจากตัวเปรียบเทียบ ความชันก็จะน้อย ทำให้เวลา t_m ใช้เวลานานมาก กว่าที่จะพ้นค่าความต่างศักย์ที่เกิดจากตัวเปรียบเทียบ เราจึงต้องนำค่าความต่างศักย์มาเพิ่มให้กับ V_{ref} เพื่อหาผลลัพธ์ ดังรูปที่ 2.9



รูปที่ 2.9 Dual Slope A/D Converter – Full Circuit

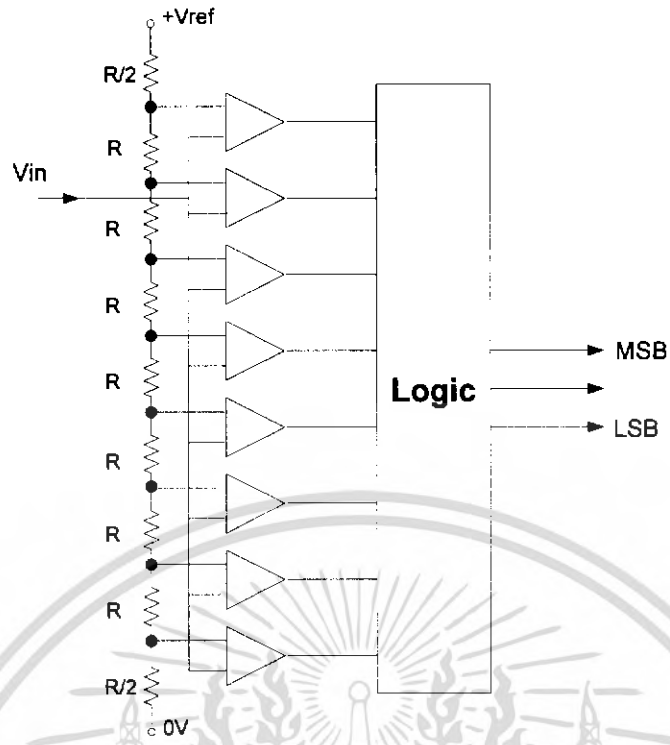
จากวงจรดังกล่าวทำให้ได้กราฟดังรูปที่ 2.10



รูปที่ 2.10 Dual Slope A/D Converter – Zero Offset

Flash Converter

หลักการของ Flash Converter คือการใช้การแบ่งแรงดันเป็น Voltage หลายๆ ค่า แล้วเปรียบเทียบกับ V_{in} เป็นคู่ๆ พร้อมกัน แล้วกระทำการทาง logic มี Voltage เปรียบเทียบ 8 bit ดังรูปที่ 2.11

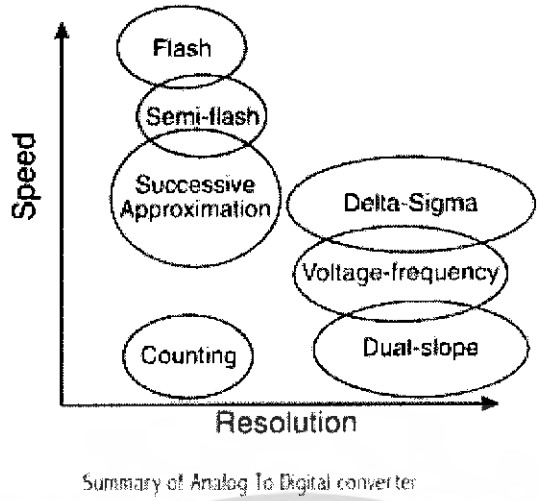


รูปที่ 2.11 Flash Converter

ค่าความต่างศักย์จะเพิ่มขึ้นเรื่อยๆ จากค่าความต้านทานที่ต่อเพิ่มขึ้น ความต่างศักย์ที่ได้นั้น เมื่อนำไปเปรียบเทียบกับ V_{in} แล้วมากกว่าก็จะปล่อยลอจิกออกมา ถ้ามากกว่าก็จะให้ลอจิก 1 ถ้าน้อยกว่าหรือเท่ากันก็จะให้ลอจิก 0 วิธี Flash Converter นี้จะเร็วที่สุด แต่ใช้อุปกรณ์ทาง Hardware มากกว่าแบบอื่นๆ

การแปลงสัญญาณอนาล็อก เป็นสัญญาณดิจิตอล มีประโยชน์มากในการควบคุมอุปกรณ์สวิตชิ่ง ซึ่งมีลักษณะการแปลงสัญญาณได้หลายวิธี แต่ละวิธีจะมีอัลกอริทึม ความรวดเร็วในการทำงาน และการใช้อุปกรณ์ฮาร์ดแวร์ต่างกันด้วย ทำให้ขนาด และราคาต่างกัน ขึ้นกับความต้องการของผู้ใช้ที่จะต้องเลือกให้เหมาะสมกับงานที่ใช้ และงบประมาณที่มีอยู่

ลำดับของความเร็ว และความละเอียดของอัลกอริทึมต่างๆ เป็นดังรูปที่ 2.12



รูปที่ 2.12 ลำดับของความเร็ว และความละเอียดของอัลกอริทึมต่างๆ

2.2.2 การแปลงดิจิตอลเป็นอะนาล็อก (Digital to Analog Conversion)

การแปลงดิจิตอลเป็นอะนาล็อก จะใช้วงจรหรืออุปกรณ์ที่ทำหน้าที่แปลงสัญญาณดิจิตอลซึ่งอาจจะเป็นแรงดันหรือกระแส ให้เป็นสัญญาณอะนาล็อกที่เป็นสัดส่วนกับสัญญาณดิจิตอลที่ป้อนเข้าไปเป็นอินพุทของวงจร เราสามารถเขียนสมการเอาท์พุทของการแปลงดิจิตอลเป็นอะนาล็อก ได้ดังนี้

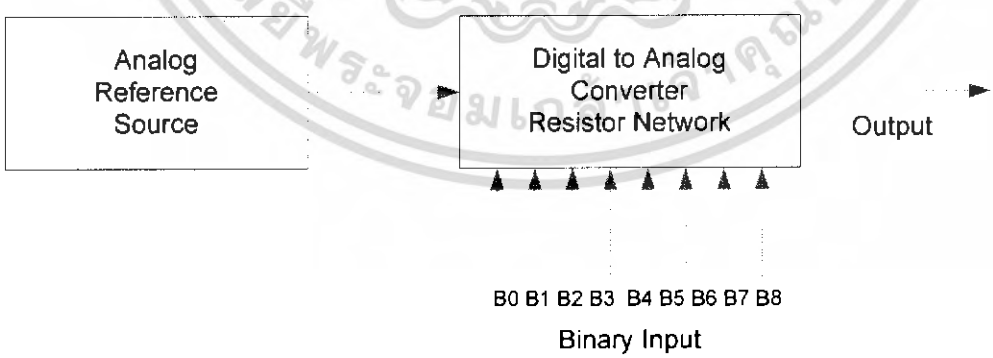
$$X = K \times A \times B$$

โดยที่ X = เป็นค่าแรงดันหรือกระแสทางด้านเอาท์พุท (อะนาล็อก)

A = ค่าอ้างอิงอะนาล็อก (เป็นแรงดันหรือกระแสก็ได้)

B = จำนวน (ค่า) ของตัวเลข Binary

K = ค่าคงที่จะมีค่าเป็น 1 เสมอ



รูปที่ 2.13 แสดงส่วนประกอบพื้นฐานของการแปลงดิจิตอลเป็นอะนาล็อก

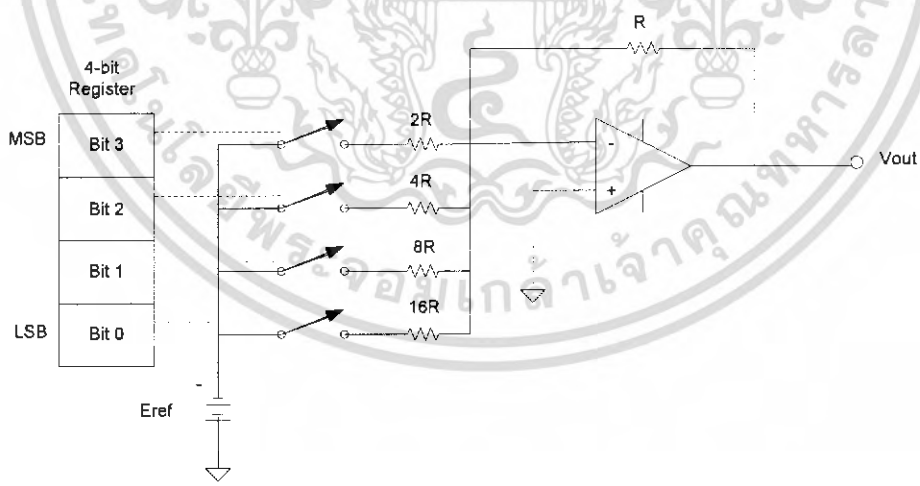
จากรูปที่ 2.13 แสดงส่วนประกอบต่างๆของระบบการแปลงดิจิตอลเป็นอะนาล็อก โดยที่แหล่งอ้างอิงอะนาล็อก จะมีค่าคงที่อยู่ค่าหนึ่ง อาจจะเป็นแหล่งสายแรงดันหรือกระแสที่ความเที่ยงตรงสูง จุดประสงค์เพื่อใช้เปรียบเทียบอินพุทที่จะสร้างแรงดันหรือกระแสที่เอาท์พุท

การแปลงดิจิตอลเป็นอะนาล็อกจะแบ่งตามการใช้ตัวต้านทานซึ่งจะมีการใช้ตัวต้านทานโดยทั่วๆไปอยู่ 2 ลักษณะ คือ แบบใช้ตัวต้านทานแบบถ่วงน้ำหนัก (Binary Weighted Register Ladder) และแบบใช้ตัวต้านทานขั้นบันได (R -2R Ladder)

การแปลงดิจิตอลเป็นอะนาล็อกแบบใช้ตัวต้านทานแบ่งน้ำหนัก (Binary Weighted Register Ladder)

การใช้ตัวต้านทานแบบนี้จะเป็นการแปลงข้อมูลดิจิตอลให้เป็นอะนาล็อกโดยตรง จะมีตัวต้านทานอนุกรมอยู่กับแรงดันอ้างอิง (V_{ref}) โดยจะมีอิเล็กทรอนิกส์สวิตช์ใช้ในการเปิดปิดสัญญาณตามสถานะของสัญญาณดิจิตอลซึ่งมีได้ 2 ระดับ คือ ลอจิก 0 กับ 1 การต่อลักษณะนี้ค่าของตัวต้านทานจะมีค่าในแต่ละน้ำหนักที่คูณด้วย 2 ตลอด คือ จะมีตั้งแต่ R,2R,4R,...จนถึง nR หรือเขียนเป็นสมการได้ว่า $2^{(n-1)}$ R โดยที่ n คือจำนวนบิตของไบนารี่ที่ทำการแปลง

เมื่อตัวต้านทานถูกต่อลงกราวนด์จะไม่มีกระแสไหลผ่านตัวต้านทานที่ต่ออยู่ แต่ถ้าตัวต้านทานตัวนั้นต่ออยู่กับแรงดันอ้างอิง(แทนด้วยระดับลอจิกที่ตรงข้ามกับการต่อตัวต้านทานกราวนด์) จะมีกระแสไหลผ่านตัวต้านทานตัวนั้น ดังนั้นถ้าตัวต้านทานค่า R และ 2R ถูกต่อกับแรงดันอ้างอิงพร้อมกัน ก็จะไม่มีกระแสไหลผ่าน ตัวต้านทาน R มีค่าเท่ากับ V_{ref} /R และกระแสไหลผ่านสองอัน มีค่าเท่ากับ V_{ref} /2R ถ้ากระแสทั้งสองไหลมารวมกันที่จุดผสม (Summing Point) ดังรูปที่ 2.14



รูปที่ 2.14 แสดงวงจรD/A Converter แบบ แบ่งน้ำหนัก

จากรูปที่ 2.14 สามารถเขียนสมการได้ดังสมการที่ (2.11)

$$I_{out} = \frac{V_{ref}}{R}[\sum_{i=1}^{n} \frac{b_i}{2^{(i-1)}}] \quad (2.11)$$

โดยที่ I_{out} = กระแสเอาท์พุท (สัญญานอะนาล็อก)มีหน่วยเป็นแอมแปร์ (A)

b_i = ค่าตัวเลขไบนารี่อินพุท (0 หรือ 1)

R = ค่าตัวต้านทานตัวแรกที่มีค่าต่ำที่สุด (เป็นค่า R ของ บิตในระยะสำคัญสูงสุด (MSB - Most Significant Bit)

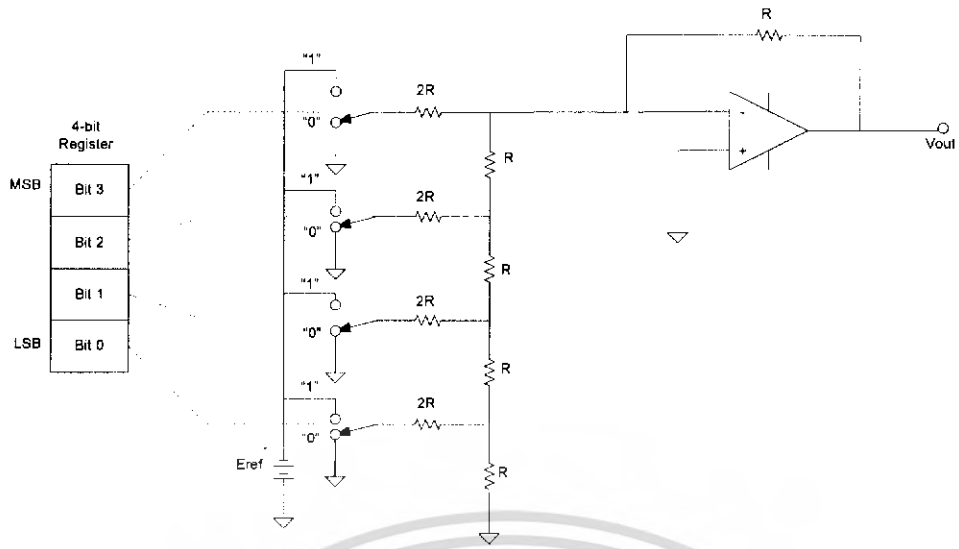
จากรูปที่ 2.15 จะเห็นได้ว่า ตัวต้านทานที่ทำให้กระแสไหลได้สูงสุดคือ R ดังนั้น ตัวต้านทาน R จะเป็นตัวต้านทานของบิตในนัยสำคัญสูงสุดและยังเห็นได้ว่า ค่าของตัวต้านทานเพิ่มขึ้นทีละมากๆ เช่น ถ้าหากมีการแปลงสัญญานดิจิตอลขนาด 8 บิต แล้วถ้าใช้ค่า R = 10 $k\Omega$ แล้วตัวต้านทานตัวที่ 8 จะมีค่าเท่ากับ $2^{(8-1)}R = 28R = 128 \times 10\ k\Omega = 1280\ k\Omega$ หรือ 1.28 $M\Omega$ จะทำให้มีกระแสไหลผ่านตัวต้านทานตัวนี้น้อยมากและจากธรรมชาติที่ว่า ตัวต้านทานค่ามากจะสร้างได้ยาก และยังมีผลกระทบจากสิ่งแวดล้อมภายนอกอีก คือ ความร้อนจะทำให้ค่าความต้านทานเปลี่ยนไป (โดยเฉพาะตัวต้านทานที่มีค่ามากๆ)ทำให้ความละเอียดและความแม่นยำของการแปลง D/A แบบนี้ลดลง ดังนั้น จากข้อจำกัดที่ว่าเมื่อมีจำนวนบิตสูงขึ้น จะทำให้ต้องใช้ตัวต้านทานที่มีค่ามากๆซึ่งหาได้ยากและสร้างได้ยากรวมถึงยังมีผลกระทบต่อความแม่นยำในการแปลง เพราะฉะนั้น เมื่อมีจำนวนบิตสูงขึ้น จึงได้มีการเปลี่ยนไปใช้แบบตัวต้านทานแบบขั้นบันได ซึ่งจะมีการใช้ตัวต้านทานเพียงสองค่าเท่านั้น ซึ่งเป็นการขจัดปัญหาที่กล่าวมาข้างต้นได้ ดังจะกล่าวในหัวข้อถัดไป

การแปลงดิจิตอลเป็นอะนาล็อกแบบตัวต้านทานขั้นบันได (R – 2R Ladder)

จะมีรูปวงจรดังแสดงใน 2.15โดยที่การใช้คล้ายการแปลง D/A แบบ R – 2R Ladder นี้ ใช้แก้ปัญหาของแบบแรกที่ต้องใช้ความต้านทานหลายค่า ซึ่งอาจจะหาค่าได้ไม่ตรงกับที่ต้องการมาใช้ตัวต้านทานเพียงสองค่าเท่านั้น ในการแปลงแบบนี้คือค่า R และ 2R ดังนั้นจากรูปที่ 2.15 วงจรสามารถเขียนสมการได้ดังสมการที่ (2.12)

$$I_{out} = [\sum_{i=1}^{n} \frac{b_i}{2^{(i-1)}}] \quad (2.12)$$

คือเป็นสมการแบบเดียวกับแบบใช้ตัวต้านทานแบ่งน้ำหนัก



รูปที่ 2.15 แสดงวงจรการแปลงดิจิตอลเป็นอะนาล็อกแบบตัวต้านทานขั้นบันได

คุณสมบัติและข้อกำหนดของตัวแปลง D/A (D/A characteristics and specification)

สำหรับตัวแปลงสัญญานดิจิตอลเป็นอะนาล็อกจะมีคุณสมบัติสำคัญดังต่อไปนี้

1.ความละเอียด (Resolution)

คุณสมบัติประการแรกของ D/A นี้คือความละเอียดของความสามารถในการเปลี่ยนแปลง พิจารณาบิตเลขฐาน 2 ที่ป้อนเข้าที่อินพุทของตัวแปร D/A ถ้าอินพุทเป็นเลขฐาน 2 แปดตัว จะแสดงว่า มีระดับของเอาท์พุทที่เป็นไปได้ เท่ากับ $2^8 = 256$ ดังนั้น ค่าความละเอียดของ D/A ตัวนี้ คือ 1 ใน 256 ในบางครั้งอาจจะกล่าวในรูปของเปอร์เซ็นต์ เช่น ความละเอียดของ D/A ขนาด 8 บิต คือประมาณ 0.39 %

2.ค่าเต็มพิกัดของแรงดันเอาท์พุท (Full Scale Output Voltage)

คุณสมบัติประการที่สองของ D/A คือค่าเต็มพิกัดของแรงดันเอาท์พุท คือค่าแรงดันสูงสุดที่ D/A สามารถให้ออกมาได้เมื่ออินพุทดิจิตอลมีค่าลอจิกเป็น 1 หมดทุกบิต

3.ความเที่ยงตรง (Accuracy)

ข้อกำหนดความเที่ยงตรงสำหรับ D/A คือการเปรียบเทียบระหว่าง เอ้าท์พุทที่เกิดขึ้นจริงกับเอาท์พุทที่คาดหวังไว้ (มาจากการคำนวณ) โดยที่จะมีการกำหนดค่าไว้เป็นเปอร์เซ็นต์ เช่น D/A มีแรงดันเต็มพิกัดของเอาท์พุท 10 โวลต์และความเที่ยงตรง 0.2 เปอร์เซ็นต์ ดังนั้นค่าผิดพลาดสูงสุดของเอาท์พุทใดๆเป็น $10V \times 0.002 = 20$ mV สำหรับ D/A ในอุดมคตินั้น จะมีค่าความผิดพลาดสูงสุดไม่มากกว่า 1/2 ของบิตนัยสำคัญต่ำสุด (LSB – Least Significant Bit)

4.ความเป็นเชิงเส้น (Linearity)

คือการวัดค่าความเบี่ยงเบนของเอาท์พุทเป็นพิสัยจากเส้นตรงที่ตัวแปลงสัญญาณไม่มีการทำงานเลย ไปจนถึงทำงานครบทุกบิตซึ่งค่าเบี่ยงเบนในอุดมคติของเอาท์พุทจากเส้นตรงควรมีไม่มากกว่า 1/2 ของค่า LSB

5.ค่าเวลาในการแปลง (Setting Time)

คือเมื่อเกิดการเปลี่ยนแปลงค่าไบนารี่ที่ทำการป้อนกับเป็นอินพุทเข้ามา ในแต่ละบิตของ D/A เอาท์พุทจะเปลี่ยนแปลงเป็นสัญญาณอะนาล็อกค่าใหม่ได้ถูกต้อง จะต้องใช้เวลาค่าหนึ่งซึ่งขึ้นกับข้อกำหนดของเวลาในการแปลงของตัว D/A ค่าเวลาที่เอาท์พุทใช้ไปภายใน 1/2 LSBของค่าสุดท้ายจะเรียกว่า ค่าเวลาในการแปลง สำหรับค่าการเปลี่ยนแปลงเอาท์พุทเต็มพิกัด คุณสมบัตินี้สำคัญเพราะถ้าตัวแปลงสัญญาณทำงานที่ความถี่สูงๆมันอาจจะไม่มีเวลาจัดตั้งค่าก่อนที่มันจะสวิตช์ไปสู่อีกสถานะหนึ่ง

2.3 ไมโครคอนโทรลเลอร์ (Microcontroller)

ไมโครคอนโทรลเลอร์ เป็นอุปกรณ์ไอซี (IC: Integrated Circuit) ที่สามารถโปรแกรมการทำงานได้หลายครั้งสามารถรับข้อมูลในรูปสัญญาณดิจิตอลเข้าไปทำการประมวลผลแล้วส่งผลลัพท์ข้อมูลดิจิตอลออกมาเพื่อนำไปใช้งานตามที่ต้องการได้

ไมโครคอนโทรลเลอร์หรืออาจจะเรียกได้ว่าไมโครโพรเซสเซอร์ชิปเดี่ยว(Single-Chip Microprocessor) เป็นไมโครโพรเซสเซอร์ชนิดหนึ่ง เช่นเดียวกับหน่วยประมวลผลกลาง (CPU: Central Processing Unit) ที่ใช้ในคอมพิวเตอร์ แต่ได้รับการพัฒนาแยกออกมาภายหลังเพื่อนำไปใช้ในวงจรทางด้านงานควบคุม คือ แทนที่ในการใช้งานจะต้องต่อวงจรภายนอกต่าง ๆ เพิ่มเติมเช่นเดียวกับไมโครโพรเซสเซอร์ ก็จะทำการรวมวงจรที่จำเป็น เช่น หน่วยความจำ, ส่วนอินพุต/เอาท์พุต บางส่วนเข้าไปในตัวไอซีเดียวกัน และเพิ่มวงจรบางอย่างเข้าไปด้วยเพื่อให้มีความสามารถเหมาะกับการใช้ในงานควบคุม เช่น วงจรตั้งเวลา วงจรการสื่อสารอนุกรม เป็นต้น ดังนั้นไมโครคอนโทรลเลอร์สามารถจะทำงานได้เสมือนกับเป็นคอมพิวเตอร์เล็กๆเครื่องหนึ่ง กล่าวโดยสรุปคือ

Microcontroller = Microprocessor + Memory + I/O

ปัจจุบันไมโครคอนโทรลเลอร์ถูกนำไปใช้อย่างกว้างขวาง โดยมักจะเป็นการนำไปใช้ฝังในระบบของอุปกรณ์อื่น ๆ (Embedded Systems) เพื่อใช้ควบคุมการทำงานบางอย่างเช่น ใช้ในรถยนต์, เครื่องปรับอากาศ, เครื่องซักผ้าอัตโนมัติ เป็นต้น เพราะว่าไมโครคอนโทรลเลอร์มีข้อดีเหมาะสมต่อการใช้ในงานควบคุมหลายประการ เช่น

- ไอซี และระบบที่ได้มีขนาดเล็ก
- ระบบที่ได้มีราคาถูกกว่าการใช้ชิปไมโครโพรเซสเซอร์
- วงจรที่ได้จะมีความซับซ้อนน้อย ช่วยลดข้อผิดพลาดที่อาจจะเกิดขึ้นได้ในการต่อวงจร
- มีคุณสมบัติเพิ่มเติมสำหรับงานควบคุมโดยเฉพาะซึ่งใช้งานได้ง่าย
- ช่วยลดระยะเวลาในการพัฒนาระบบได้

ไมโครคอนโทรลเลอร์มีหลายยี่ห้อ หลายตระกูล และหลายเบอร์ด้วยกัน ซึ่งแต่ละเบอร์ก็จะมีโครงสร้างภายในและความสามารถในการทำงานที่แตกต่างกัน ทำให้เลือกใช้งานได้อย่างเหมาะสม

2.3.1 ไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีด้วยกันหลายเบอร์ขึ้นกับโครงสร้างภายในของมัน บางเบอร์จะมีหน่วยความจำภายในเป็นแบบ ROM บางเบอร์เป็นแบบ EPROM บางเบอร์มี RAM ภายใน 128 ไบต์ บางเบอร์มี 256 ไบต์ เป็นต้น ซึ่งรายละเอียดสามารถศึกษาได้จากคู่มือได้โดยตรง และลักษณะของขาต่าง ๆ จะเหมือนกัน

คุณสมบัติที่สำคัญของ MCS-51 มีดังนี้

- มีหน่วยความจำ ROM 4 Kbytes, 8 Kbytes, 20 Kbytes
- มีหน่วยความจำ RAM 128 byte
- มีพอร์ท I/O ขนาด 8 บิต 4 พอร์ท
- มี Timer 16 บิต 2 ตัว
- สามารถอินเทอร์รัพท์ได้ 5 แหล่ง
- มีวงจรออสซิลเลเตอร์และวงจรนาฬิกาบนชิพ
- มีพอร์ทอนุกรมที่สามารถรับส่งข้อมูลแบบ Full Duplex ด้วยความเร็วสูง
- อ้างหน่วยความจำโปรแกรมภายนอกได้ 64K
- อ้างหน่วยความจำข้อมูลภายนอกได้ 64K
- สามารถประมวลผลทีละบิตได้
- สามารถอ้างหน่วยความจำแบบบิตได้ 210 ตำแหน่ง
- หนึ่งวัฏจักรคำสั่งกินเวลาประมาณ 1 ไมโครวินาที ขณะทำงานด้วย Clock 11.0592 MHz

ไมโครคอนโทรลเลอร์ MCS-51 จะมีชุดคำสั่ง (Instruction Set) อยู่จำนวนหนึ่ง สำหรับสั่งงานให้ทำงานต่าง ๆ และเนื่องจาก MCS-51 จะประมวลผลแบบ 8 บิต รหัสภาษาเครื่องจะมีขนาด 8 บิตด้วย ซึ่งชุดคำสั่งจะมีได้จำนวนสูงสุด 28 = 256 ชุดคำสั่ง คำสั่งแต่ละคำสั่งอาจมีขนาด 1, 2 หรือ 3 ไบต์

MCS-51 มีโหมดการอ้างแอดเดรส (Addressing Modes) เพื่อติดต่อกับหน่วยความจำซึ่งอาจจะเป็นการอ่านข้อมูลออกมาหรือเขียนข้อมูลเข้าไปได้ทั้งหมด 8 โหมดคือ Register, Direct, Indirect, Immediate, Relative, Absolute และ Index

ใน MCS-51 จะแบ่งชุดของคำสั่งออกได้ 5 ประเภท ได้แก่

1. Arithmetic Instructions เป็นกลุ่มคำสั่งที่ทำงานด้านคณิตศาสตร์ เช่น ADD, SUBB, INC, DIV เป็นต้น

2. Logical Instructions มีลักษณะการทำงานคล้ายกับ Boolean Operation ซึ่งสามารถกระทำแบบไบต์ต่อไบต์ หรือ บิตต่อบิตได้ เช่น ANL, ORL เป็นต้น

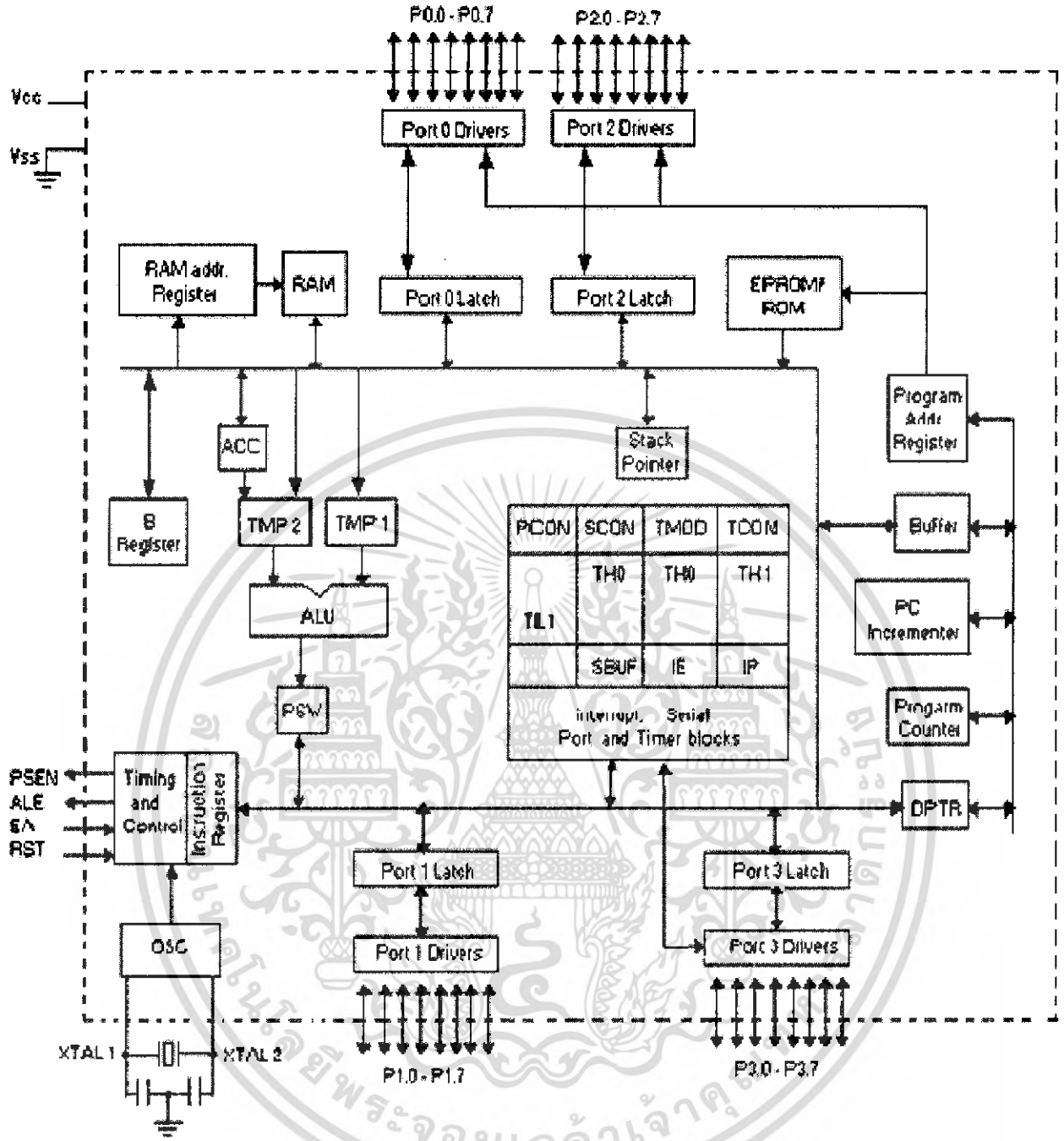
3. Data Transfer Instructions เป็นกลุ่มคำสั่งที่ใช้ในการเคลื่อนย้าย คัดลอกข้อมูลซึ่งสามารถติดต่อกับหน่วยความจำได้หลายแบบ เช่น MOV, XCH, XCHD เป็นต้น

4. Boolean Instructions เช่น ANL, ORL, CLR, SETB เป็นต้น

5. Program Branching เป็นกลุ่มคำสั่งสำหรับสั่งให้โปรแกรมกระโดดไปทำงานในตำแหน่งที่ต้องการ แบ่งเป็นกลุ่มย่อยได้ 2 กลุ่มคือ กระโดดแบบมีเงื่อนไข เช่น AJMP, LJMP, SJMP กับ กระโดดแบบมีเงื่อนไข เช่น JZ, JNZ, CJNE, DJNZ เป็นต้น

2.3.2 โครงสร้างภายในของไมโครคอนโทรลเลอร์ MCS – 51

วงจรภายในของไมโครคอนโทรลเลอร์ตระกูล MCS – 51 ประกอบไปด้วยวงจรพอร์ทอินพุทและเอาท์พุททั้งหมด 4 พอร์ท แต่ละพอร์ตจะเป็นแบบ 8 บิต หน่วยความจำภายในโปรแกรม (EPROM , EEPROM และ Flash) หน่วยความจำที่เป็นข้อมูล (RAM) ซึ่งรวมอยู่ในวงจรหลักของไมโครคอนโทรลเลอร์ตลอดจนวงจรการคำนวณทางคณิตศาสตร์และลอจิก (ALU) วงจรรีจิสเตอร์ทั่วไปและรีจิสเตอร์ฟังก์ชันการใช้งานเฉพาะ แสดงดังรูปที่ 2.16

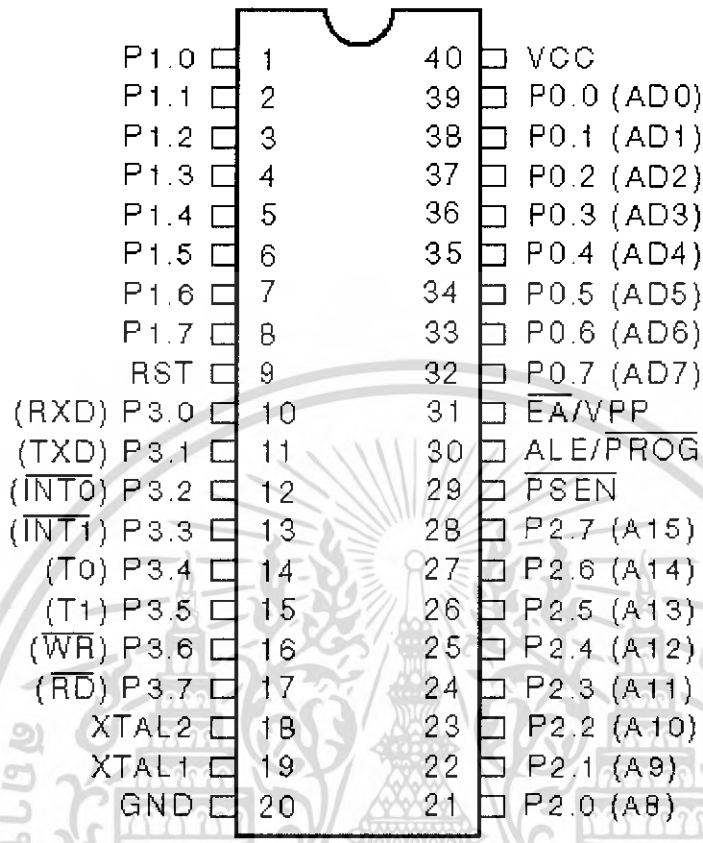


รูปที่ 2.16 แสดงโครงสร้างภายในของไมโครคอนโทรลเลอร์ MCS – 51

2.3. 3 ลักษณะการจัดขาของ MCS-51

การจัดตำแหน่งขาของไมโครคอนโทรลเลอร์ MCS – 51 ไมโครคอนโทรลเลอร์ MCS – 51 ทุกเบอร์จะมีโครงสร้างและการใช้งานพื้นฐานเหมือนกันตัวอย่างเช่น แบบดิป (DIP) ซึ่งมีทั้งหมด 40

ขา ได้แบ่งการใช้งานออกเป็นขาอินพุท / เอาท์พุท ขาสัญญาณควบคุม ขาสัญญาณกำหนดตำแหน่งหน่วยความจำ และขาสัญญาณข้อมูลดังรูปที่ 2.17



รูปที่ 2.17 แสดงตำแหน่งขาไมโครคอนโทรลเลอร์ MCS – 51

ตำแหน่งขาของไมโครคอนโทรลเลอร์ MCS – 51 และหน้าที่การทำงาน

1. P0.0 – P0.7 (ขาที่ 32 - 39) พอร์ท 0 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกได้ 2 ทิศทาง สามารถรับข้อมูลอินพุทและส่งข้อมูลเอาท์พุทได้ มีขนาด 8 บิต การตั้งค่าให้พอร์ท 0 รับข้อมูลอินพุท ทำได้โดยการส่งค่าสถานะ 1 ไปยังบิตที่ต้องการให้รับข้อมูลอินพุท วงจรภายในจะทำให้บิตนั้นมีค่าความต้านทานสูงและสามารถรับข้อมูลอินพุทได้ และยังใช้เป็นขาสัญญาณกำหนดตำแหน่งหน่วยความจำ (A_0 – A_7) และขาสัญญาณข้อมูล (D_0 – D_7) โดยการใช้ตัวแยกสัญญาณ (D – latch 74LS373) ทำหน้าที่เป็นมัลติเพล็กซ์ (multiplex) โดยเลือกช่วงเวลาของสัญญาณกำหนดตำแหน่งหน่วยความจำและสัญญาณข้อมูลออกจากกัน

ในขณะที่ใช้เป็นพอร์ตอินพุทและเอาท์พุท วงจรภายในจะไม่มีวงจรเพิ่มกระแสไฟฟ้า (Pull Up) จึงจำเป็นต้องต่อวงจรเพิ่มกระแสไฟฟ้าภายนอกเข้าไป

2. P1.0 – P1.7 (ขาที่ 1 - 8) พอร์ท 1 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกได้ 2 ทิศทาง สามารถเป็นได้ทั้งอินพุทและเอาท์พุท มีขนาด 8 บิต สามารถอ้างถึงการทำงานได้ทีละบิต และ

วงจรภายในมีตัวต้านทานเพิ่มกระแสไฟฟ้า (Pull Up) ในกรณีที่ต้องการให้รับข้อมูลอินพุตก็สามารถทำได้เหมือนพอร์ท 0

3. P2.0 – P2.7 (ขาที่ 21 -28) พอร์ท 2 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกได้ 2 ทิศทางเป็นได้ทั้งอินพุทและเอาท์พุท มีขนาด 8 บิต สามารถใช้เป็นขาสัญญาณกำหนดตำแหน่งหน่วยความจำ (A_8 – A_{15}) และมีวงจรเพิ่มกระแสภายใน การกำหนดให้เป็นขาอินพุตทำได้โดยการส่งค่าข้อมูลสถานะ 1 ไปยังบิตที่ต้องการให้เป็นอินพุท ก็จะสามารถรับค่าข้อมูลอินพุทได้

4. P3.0 – P3.7 (ขาที่ 10 -17) พอร์ท 3 ทำหน้าที่เป็นสัญญาณควบคุมอุปกรณ์ภายนอกอินพุทและเอาท์พุท 2 ทิศทาง มีขนาด 8 บิต คุณสมบัติทั่วไปจะเหมือนกับพอร์ทอื่นๆ แต่จะมีคุณสมบัติที่ต่างออกไป คือ ใช้ทำหน้าที่พิเศษเป็นสัญญาณควบคุมการทำงานต่างๆ ของไมโครคอนโทรลเลอร์ ดังตารางที่ 2.2

ตารางที่ 2.2 แสดงสัญญาณควบคุมการทำงานต่างๆ ของไมโครคอนโทรลเลอร์

บิตของพอร์ต	สัญญาณ	หน้าที่การทำงาน
P3.0	RXD	รับข้อมูลจากพอร์ตอนุกรม (serial input port)
P3.1	TXD	ส่งข้อมูลจากพอร์ตอนุกรม (serial output port)
P3.2	INT0	รับสัญญาณอินเตอร์รัปต์หมายเลข 0 (external interrupt 0)
P3.3	INT1	รับสัญญาณอินเตอร์รัปต์หมายเลข 1 (external interrupt 1)
P3.4	T0	ใช้ตั้งเวลา / นับเวลาตัวที่ 0 (Timer 0 external input)
P3.5	T1	ใช้ตั้งเวลา / นับเวลาตัวที่ 1 (Timer 1 external input)
P3.6	WR	เป็นสัญญาณเขียนข้อมูลหน่วยความจำหรืออุปกรณ์ภายนอก (external data memory write strobe)
P3.7	RD	เป็นสัญญาณเขียนข้อมูลหน่วยความจำหรืออุปกรณ์ภายนอก (external data memory read strobe)

2.3.4 พอร์ทอินพุทและพอร์ทเอาท์พุท

1. การใช้งานเป็นพอร์ทอินพุต

เนื่องจากพอร์ททั้งหมดของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลชสามารถเป็นได้ทั้งอินพุทและเอาท์พุท ดังนั้นจึงมีความจำเป็นอย่างยิ่งต้องทำความเข้าใจถึงการกำหนดลักษณะการทำงานให้แก่พอร์ทของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลช

ในการกำหนดให้เป็นพอร์ทอินพุท ต้องเริ่มต้นด้วยการเขียนข้อมูล "1" มาที่แต่ละบิตของพอร์ทที่ต้องการใช้งานเป็นอินพุท เพื่อหยุดการทำงานของเฟตที่ใช้ในการขับสัญญาณเอาท์พุทของบิตนั้นๆ ทำให้ขาสัญญาณของพอร์ทเชื่อมต่อกับเข้ากับวงจรพูลอัพภายในโดยตรง ส่งผลให้ขาพอร์ทนั้นมีลอจิกเป็น "1" สามารถรับสัญญาณลอจิก "0" จากอุปกรณ์ภายนอกได้ง่าย สัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ท แล้วรอให้ซีพียูมาอ่านค่าเข้าไป เมื่อเป็นเช่นนี้ อุปกรณ์

ภายนอกที่เชื่อมต่อกับพอร์ทอินพุทของไมโครคอนโทรลเลอร์ MCS – 51 แบบแฟลชควรกำหนดให้ทำงานในสภาวะลอจิก “0” จะดีและสะดวกที่สุด (ซึ่งในปัจจุบันอุปกรณ์อินพุทที่เชื่อมต่อไมโครคอนโทรลเลอร์แบบทั้งหมดทำงานที่ลอจิก “0” แล้ว)

2. การใช้งานเป็นพอร์ทเอาท์พุท

โดยปกติแล้ว ขาพอร์ทจะกำหนดให้มีลักษณะเป็นเอาท์พุทอยู่แล้ว ดังนั้นจึงสามารถส่งข้อมูลออกไปได้อย่างง่ายดายและตรงไปตรงมา กล่าวคือ เมื่อต้องการส่งข้อมูล “0” ออกไปทางเอาท์พุทก็ให้เขียนข้อมูล “0” ไปยังวงจรแลตช์ ซึ่งก็จะส่งต่อไปขับเฟต ทำให้เฟตทำงาน ที่ขาพอร์ตที่กำหนดให้ทำงานก็จะเกิดลอจิก “0” ขึ้น ในทางตรงข้ามหากต้องการส่งข้อมูล “1” ออกไป ก็ให้เขียนข้อมูล “1” ไปยังวงจรแลตช์ วงจรขับก็จะหยุดทำงาน ทำให้ที่ขาพอร์ทเชื่อมต่อกับวงจรพูลอัพภายในเกิดเป็นลอจิก “1” ที่ขาพอร์ทนั้น ซึ่งจะคล้ายกับการกำหนดให้เป็นขาอินพุทมาก เพียงแต่แตกต่างกันที่กระบวนการในการเคลื่อนย้ายข้อมูล โดยถ้าเป็นอินพุทจะมีสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่ถ้าเป็นเอาท์พุทจะไม่มีการอ่านข้อมูลที่บัฟเฟอร์แต่อย่างใด เว้นแต่ในกรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาท์พุท

เมื่อใช้งานเป็นพอร์ทเอาท์พุท แต่ละขา (หรือแต่ละบิต) ของแต่ละพอร์ทมีความสามารถในการจ่ายกระแสหรือที่เรียกว่า กระแสซอร์ต (Source current) ได้สูงสุด 500 μA ในขณะที่สามารถรับกระแสซิงค์ (sink current) เมื่อทำงานด้วยลอจิก “0” ได้สูงถึง 10 mA ต่อขาและทุกขารวมกันในแต่ละพอร์ท (ทั้ง 8 บิต) สูงสุด 26 mA สำหรับพอร์ท 0 และ 15 mA สำหรับพอร์ท 1 ถึง 3 ในกรณีที่ใช้งานทุกพอร์ทเอาท์พุทจะสามารถรับกระแสซิงก์ได้รวมกันสูงสุด 71 mA ดังนั้นในการใช้งานเป็นพอร์ทเอาท์พุทเพื่อไม่ให้เกิดปัญหาเกี่ยวกับความสามารถในการจ่ายกระแสจึงควรต่อวงจรบัฟเฟอร์ทางเอาท์พุตเพื่อช่วยในการขับกระแสอีกทางหนึ่ง

2.3.5 การใช้งานพอร์ทสื่อสารอนุกรมแบบ Single Processor

พอร์ทสื่อสารอนุกรม

พอร์ทสื่อสารอนุกรมมีโครงสร้างการทำงานในแบบที่เรียกว่า **ฟูลดูเพล็กซ์** (Full Duplex) สามารถรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน

- ทางด้านส่งใช้ขา TxD (พอร์ท 3.1)
- ทางด้านรับใช้ขา RxD (พอร์ท 3.0)

Serial Port Buffer (SBUF) ใช้เป็นบัฟเฟอร์สำหรับรับและส่งข้อมูลอนุกรมโดยมีอยู่ 2 ตัว

การส่งข้อมูล ข้อมูลที่จะส่งให้ใส่ใน SBUF โดยใช้คำสั่ง MOV SBUF,A โดยเตรียมข้อมูลที่จะส่งเข้า A ก่อน

การรับข้อมูล ข้อมูลที่รับได้จะอยู่ใน SBUF การถ่ายข้อมูลออกมาใช้คำสั่ง MOV A,SBUF แล้วจึงนำข้อมูลใน A ไปใช้

พอร์ทสื่อสารอนุกรมสามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM1 และ SM0 ซึ่งอยู่ในรีจิสเตอร์ควบคุม SCON การทำงานทั้ง 4 โหมด ของพอร์ทสื่อสารอนุกรม มีดังนี้

SM0, SM1 บิตเลือกโหมดการทำงาน

ตารางที่ 2.3 แสดง การเลือกโหมดการทำงานในการรับส่งข้อมูล

SM0	SM1	โหมด	การทำงาน
0	0	**0**	**Shift register** ความเร็วในการรับหรือส่งข้อมูลเท่ากับ **(1/12) ของ CPU OSC**
0	1	**1**	**8 Bit UART** ความเร็วในการรับหรือส่งข้อมูลกำหนดได้จาก Timer 1,2
1	0	**2**	**9 Bit UART** ความเร็วในการรับหรือส่งข้อมูล = (1/32) หรือ (1/64)เท่าของ**CPU OSC** โดยขึ้นกับบิต SMOD ใน PCON
1	1	**3**	**9 Bit UART** ความเร็วในการรับหรือส่งข้อมูลกำหนดที่ Timer 1, 2

REN (Receive Enable) **บิตควบคุมให้รับหรือไม่รับข้อมูล**

1 : ให้รับข้อมูลได้

0 : ห้ามรับข้อมูล

หมายเหตุ (การรับข้อมูลสามารถห้ามได้ แต่การส่งข้อมูลห้ามไม่ได้)

TI แฟลกซ์ TI จะเป็น 1 **เมื่อสิ้นสุดการส่งข้อมูล 1 ไบต์**

RI แฟลกซ์ RI จะเป็น 1 **เมื่อรับข้อมูลเสร็จ 1ไบต์** (บิต RI,TI ผู้เขียนโปรแกรมจะต้องเคลียร์เอง)

การเขียนโปรแกรมควบคุมการรับและส่งข้อมูลทำได้ 2 วิธี

- **การตรวจสอบบิต TI หรือ RI** โดยใช้คำสั่งตรวจสอบบิต เช่นใช้คำสั่ง **WAIT:JNB TI,WAIT**

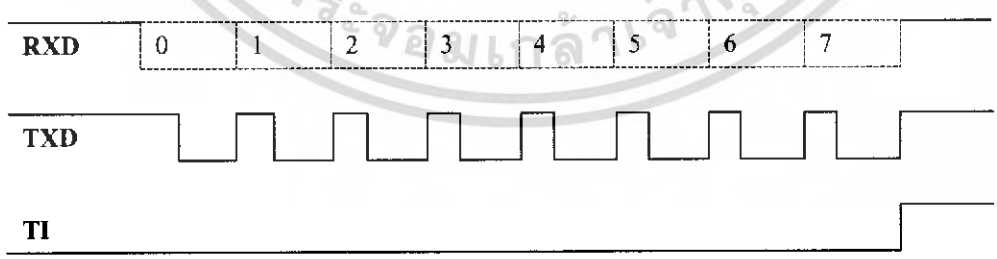
  คำสั่งนี้หมายความว่า ถ้า **TI = 0** ให้วนไปยังแอดเดรสชื่อ WAIT

  ถ้า **TI = 1** ถือว่าส่งข้อมูลเสร็จแล้วให้ทำคำสั่งถัดไป

- **การใช้อินเตอร์รัพต์ควบคุม**

โหมด 0 : พอร์ทสื่อสารอนุกรม 8 บิต โดยการส่งข้อมูลจะเลื่อนออกทีละบิตโดยส่งบิต D0 ออกไปก่อนทางขา RxD เนื่องจากไม่มีการส่ง Start bit แต่จะส่ง shift clock ทางขา TxD

[ความเร็ว (1/12) เท่าของ CPU Clock]

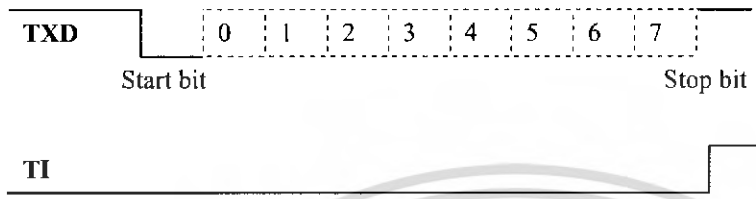


รูปที่ 2.18 แสดง Timing Diagram การส่งข้อมูลโหมด 0

โหมด 1 : พอร์ทสื่อสารอนุกรม 10 บิต ข้อมูล 8 บิต 1 start bit และ 1 stop bit และสามารถเปลี่ยนแปลงความเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิด SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1,2

$$\textbf{Baud Rate Mode 1,3} = \frac{2^{SMOD} \times \text{CPU OSC}}{32 \times 12 \times [256 - (TH1)]}$$ โดยใช้ Timer 1

$$\textbf{Baud Rate Mode 1,3} = \frac{\text{CPU OSC}}{32 \times [65536 - (RCAP2H,RCAP2L)]}$$ โดยใช้ Timer 2

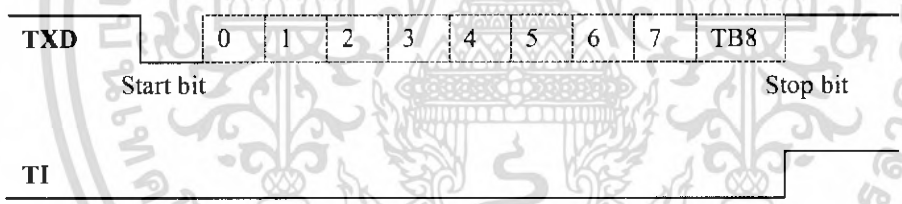


รูปที่ 2.19 แสดง Timing Diagram การส่งข้อมูลโหมด 1

โหมด 2 : พอร์ทสื่อสารอนุกรม 11 บิต ข้อมูล 9 บิต 1 start bit และ 1 stop bit (TB8 นิยมนำมาใช้ส่ง Parity bit) ความเร็วในการรับส่งข้อมูลเท่ากับ (1/32) หรือ (1/64) เท่าของ CPU OSC โดยขึ้นกับบิต SMOD ใน PCON

-Baud Rate (Mode 2) = (1/32) CPU OSC เมื่อ SMOD = 1

-Baud Rate (Mode 2) = (1/64) CPU OSC เมื่อ SMOD = 0

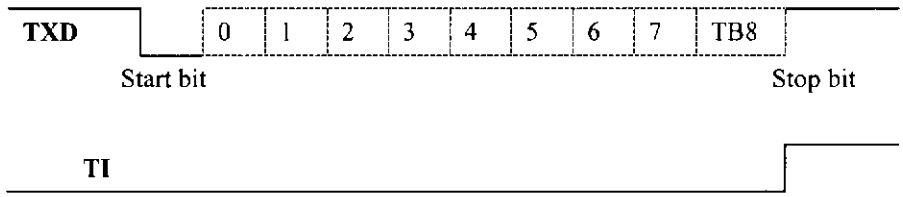


รูปที่ 2.20 แสดง Timing Diagram การส่งข้อมูลโหมด 2

โหมด 3 : พอร์ทสื่อสารอนุกรม 10 bit UART โดย DATA 8 bit, 1 start bit และ 1 stop bit เหมือนโหมด 2 ยกเว้นอัตราความเร็วจะขึ้นกับบิต SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1 สำหรับ 8051 หรือ อัตราโอเวอร์โฟลว์ของ Timer 2 (สำหรับ 80C154D)

$$\text{Baud Rate Mode 3} = \frac{2^{SMOD} \times \text{CPU OSC}}{32 \times 12 \times [256 - (TH1)]}$$ **โดยใช้ Timer 1**

$$\text{Baud Rate Mode 3} = \frac{\text{CPU OSC}}{32 \times [65536 - (RCAP2H,RCAP2L)]}$$ **โดยใช้ Timer 2**



รูปที่ 2.21 แสดง Timing Diagram การส่งข้อมูลโหมด 3

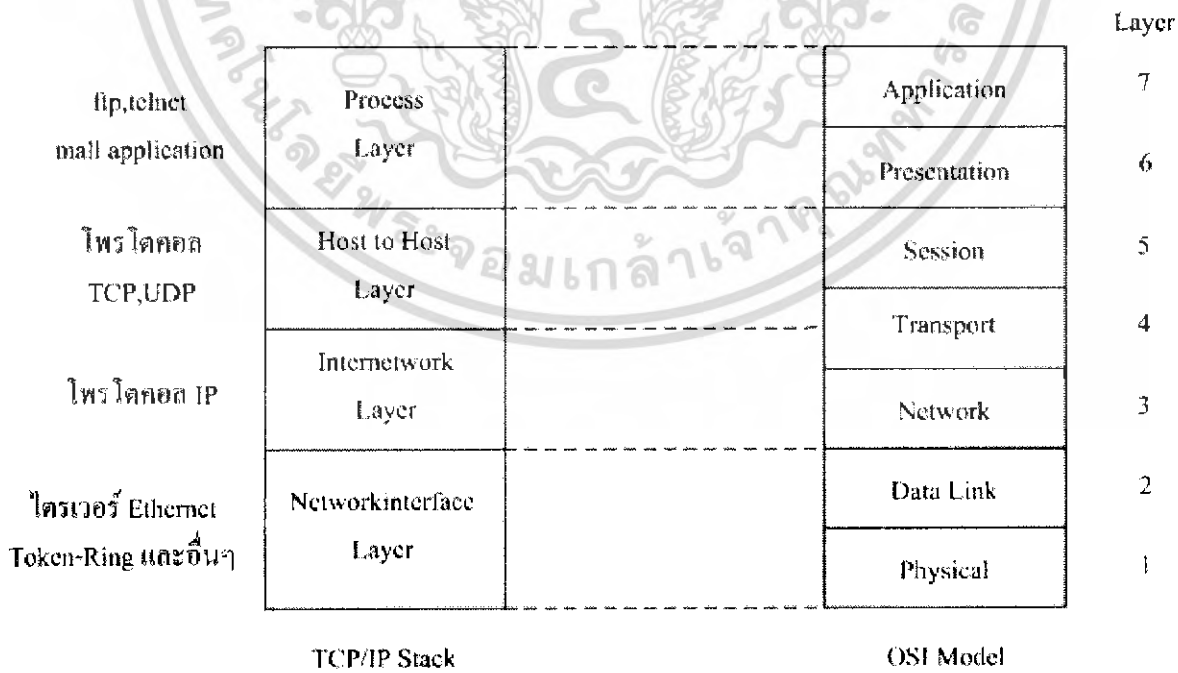
2.4 โพรโตคอล TCP/IP

2.4.1 โครงสร้างของโพรโตคอล TCP/IP

โพรโตคอล TCP/IP มีการจัดกลไกการทำงานเป็นชั้นหรือ Layer เรียงต่อกัน โดยในแต่ละ Layer จะมีการทำงานเทียบได้กับ OSI model มาตรฐาน แต่บาง Layer ของโพรโตคอล TCP/IP จะทำงานเทียบกับ OSI หลาย Layer ปนกัน ซึ่งในแต่ละ Layer ของโพรโตคอล TCP/IP จะประกอบด้วย

- Process Layer
- Host – to – Host Layer
- Internetwork Layer
- Network Interface Layer

โดยเมื่อเทียบกับมาตรฐาน OSI model ดังรูปที่ 2.22 ซึ่งเราจะเห็นว่าบางกลไกของโพรโตคอล TCP/IP เทียบได้กับมาตรฐาน OSI model สองชั้น หรือบางกลไกก็จะทำงานคาบเกี่ยวกันระหว่างบางชั้นของ OSI model ตัวอย่างเช่น กลไกการทำงานของโพรโตคอล TCP/IP ในส่วน Network Interface Layer เมื่อเทียบกับมาตรฐาน OSI model จะเทียบได้กับ Data Link Layer และ Physical Layer 2 ชั้นรวมกัน เป็นต้น ในแต่ละกลไกของโพรโตคอล TCP/IP และโพรโตคอลอื่น ๆ ในชุดของ TCP/IP ร่วมทำงานอยู่ด้วย



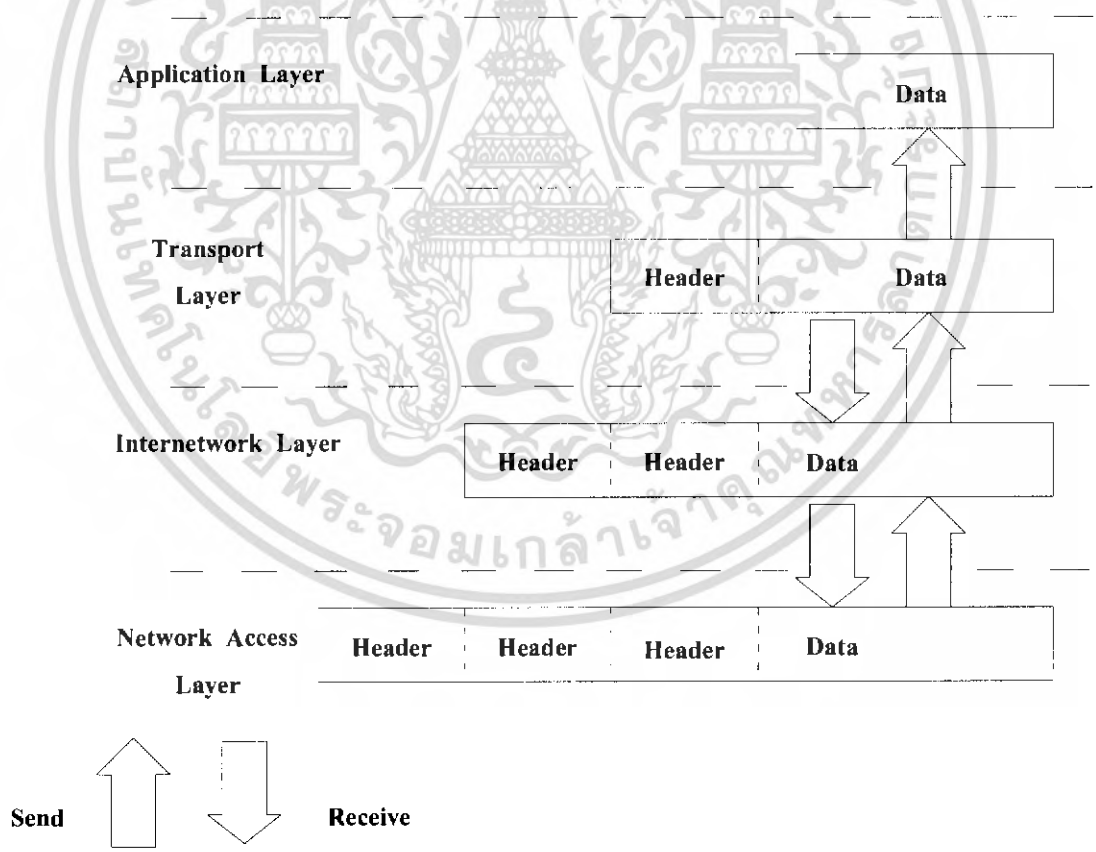
รูปที่ 2.22 แสดงกลไกของโพรโตคอลมาตรฐาน OSI model

สำหรับการศึกษาโพรโตคอล TCP/IP นั้นเราจะไม่อ้างอิง OSI Reference Model นี้เพราะจะเข้าใจได้ยาก ดังนั้นเราจึงจะสร้างโมเดลขึ้นมาใหม่โดยแบ่งออกเป็น 4 ชั้นดังนี้

4	Application Layer
3	Host – to – Host Transport Layer
2	Internet Layer
1	Network Access Layer

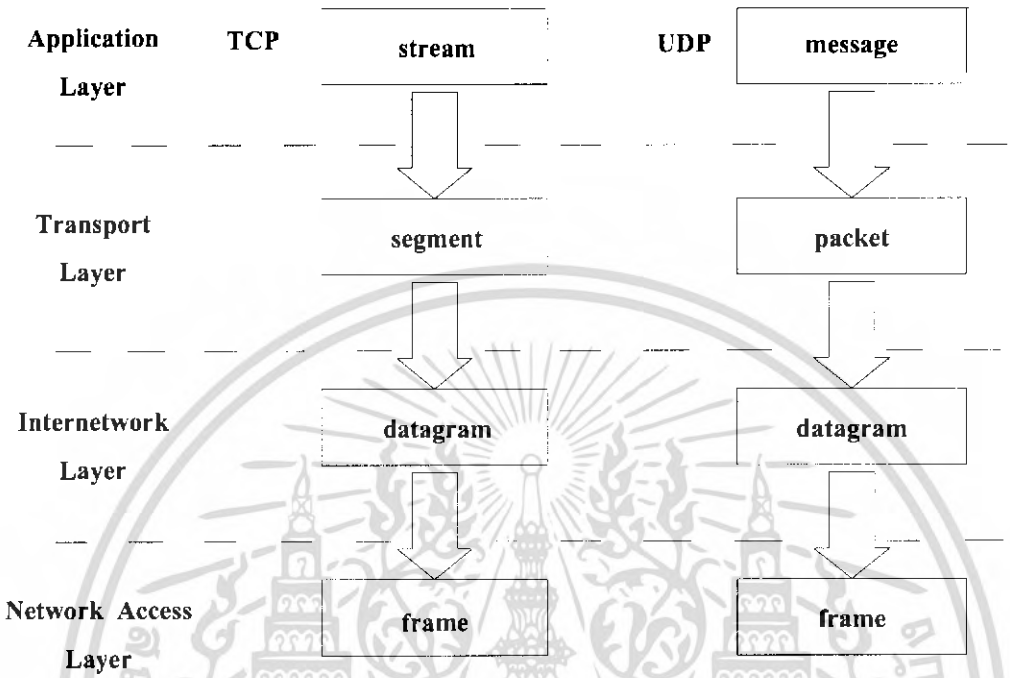
ลักษณะการทำงานของโมเดลในลักษณะนี้คือ ข้อมูลจะถูกส่งลงมาจากชั้นข้างบนลงมายังชั้นข้างล่างสุดซึ่งมีหน้าที่จัดการเกี่ยวกับการส่งข้อมูลผ่านสายสัญญาณไปยังจุดหมายปลายทาง เมื่อข้อมูลไปถึงจุดหมายแล้วก็จะกลับย้อนจากชั้นล่างขึ้นไปชั้นบนสุด ซึ่งเป็นชั้นที่โปรแกรมใช้งานต่างๆ ทำงานอยู่

ขณะที่ข้อมูลถูกส่งผ่านจากชั้นบนลงมายังชั้นล่าง แต่ละชั้นจะทำการเพิ่มข้อมูลควบคุมเข้าไปเพื่อให้การส่งข้อมูลถูกต้อง และเป็นการส่งพารามิเตอร์ที่จำเป็นไปให้กับชั้นของมันในเครื่องปลายทาง ข้อมูลควบคุมเหล่านี้เราเรียกว่า Header แต่ละชั้นจะมี Header ที่มีรูปแบบเป็นของตัวเอง การเพิ่มเฮดเดอร์เข้าไปกับข้อมูลนั้นเราเรียกว่า Data Encapsulation ซึ่งได้แสดงไว้ในรูปที่ 2.23



รูปที่ 2.23 Data Encapsulation

หน่วยของข้อมูลที่จะทำการส่งนั้นนั้นจะมีชื่อเรียกต่างกันเมื่อมันเดินทางผ่านแต่ละ Layer ดังแสดงไว้ในรูปที่ 2.24 ซึ่งจะเห็นว่าการส่งใน TCP/IP นั้นมีอยู่ 2 โพรโตคอล ย่อยคือ TCP กับ UDP ชื่อของข้อมูลที่ผ่านโพรโตคอลทั้งสองนั้นแตกต่างกันในชั้นบนๆ เท่านั้น ชื่อสำคัญๆ ที่เราจะต้องพบและใช้ต่อไปบ่อยๆ ได้แก่ ดาต้าแกรม และเฟรม



รูปที่ 2.24 โครงสร้างของข้อมูล

2.4.2 Network Interface Layer

เนื่องจากในด้านกายภาพของเครือข่ายนั้น มีหลายวิธีการและหลายรูปแบบในการเชื่อมต่อระบบให้เป็นเครือข่าย แต่อย่างไรก็ตามในเครือข่ายอินเตอร์เนตนี้ ข้อมูลหรือ IP datagram จะถูกถ่ายทอดและส่งผ่านไปยังปลายทางโดยไม่คำนึงถึงรูปแบบการเชื่อมต่อทางกายภาพ ไม่ว่าจะเป็นการใช้เครือข่ายใยแก้วนำแสงหรือเครือข่ายสาย Unshielded Twist Pair (UTP) เชื่อมต่อเป็นแบบเครือข่ายอีเทอร์เนต ธรรมดาหรือเครือข่าย Token Ring, ATM, ISDN ฯลฯ ก็ตาม

2.4.2.1 อีเทอร์เน็ท

อีเทอร์เน็ทเป็นการใช้สายโคแอคเชียลเบสบัสเชื่อมต่อระบบเข้าด้วยกันเพื่อทำการส่งถ่ายข้อมูลในระบบดิจิตอลระหว่างคอมพิวเตอร์ โดยบริษัทอุปกรณ์ดิจิตอล บริษัท Intel และบริษัท Xerox ได้ใช้ระบบนี้อ้างอิงเรื่อยมา ซึ่งอีเทอร์เน็ทจะใช้เทคนิคการส่งเบสแบนด์ในการเข้าถึงข้อมูล และยังสามารถใช้บนสายโคแอคเชียลที่มี 2 ขนาด คือ สายอีเทอร์เน็ทแบบหนา และสายอีเทอร์เน็ทแบบบาง โดยสายเคเบิลทั้งสองนี้เป็นที่ใช้กันอย่างกว้างขวางในปัจจุบัน

ส่วนประกอบหลักที่สำคัญของเครือข่ายอีเทอร์เน็ท

ระบบเครือข่ายอีเทอร์เน็ท มีส่วนประกอบหลักซึ่งเมื่อทำงานด้วยกันแล้วก็จะเป็นเครือข่ายที่มีประสิทธิภาพการทำงานสูงดังนี้

1. ตัวเฟรมเป็นชุดรูปแบบของบิตข้อมูลข่าวสารที่ใช้ส่งผ่านมาบนระบบ หากไม่มีเฟรมเราจะไม่สามารถสื่อสารข้อมูลบนเครือข่ายได้โดยเด็ดขาด การรับส่งข้อมูลข่าวสารบนเครือข่ายอีเทอร์เน็ท จะต้องเป็นไปในรูปแบบเฟรมมาตรฐาน 2 แบบ และเป็นแบบใดแบบหนึ่งเท่านั้น (การ์ด LAN เป็นผู้สร้างเฟรมนี้ขึ้นมา)
2. ชุดโปรโตคอลที่ใช้ในการควบคุมการแอกเซสเข้าไปที่เครือข่าย (Media Access Control Protocol) ซึ่งประกอบด้วยชุดของกฎกติกาที่อยู่ใน Ethernet Interface (เช่น การ์ด LAN เป็นต้น) ซึ่งเป็นกฎมาตรฐานที่จะยอมให้คอมพิวเตอร์ต่างๆสามารถเข้ามาที่เครือข่าย และแบ่งใช้ทรัพยากรต่างๆ บนเครือข่ายได้อย่างมีประสิทธิภาพ
3. อุปกรณ์ที่ใช้รับส่งสัญญาณบนเครือข่าย (Signaling Components) ประกอบด้วยชุดของอุปกรณ์ที่ใช้เชื่อมต่อและส่งสัญญาณเพื่อการรับส่งข้อมูลภายในเครือข่าย
4. สื่อที่ใช้ในการรับส่งสัญญาณข้อมูลบนเครือข่าย (Physical Medium) ประกอบด้วยสายสัญญาณรวมทั้งอุปกรณ์ทางฮาร์ดแวร์อื่นๆ ที่จะช่วยในการนำพาข้อมูลข่าวสารต่างๆ ในรูปแบบดิจิตอลวิ่งไปมาบนเครือข่าย

2.4.2.2 เฟรมอีเทอร์เน็ท

อีเทอร์เน็ทได้ถูกระบุไว้อย่างแน่นอนไว้ในชั้น Physical Layer โดยมันจะแสดงรายละเอียดของรูปแบบเป็นแพ็กเก็ต ซึ่งโดยส่วนมากจะเรียกว่า เฟรม ซึ่งเป็นหัวใจสำคัญของระบบอีเทอร์เน็ท จากรูปที่ 2.25 ได้แสดงรูปแบบของเฟรมอีเทอร์เน็ท โดยเฟรมนี้มันจะ Encapsulates TCP/IP โปรโตคอล และสามารถทำการตอบสนองสำหรับส่งข้อมูลข้ามเข้าระบบที่เชื่อมต่อไปยังอีก Layer ได้โดย Gateway หรือ End Node

Preamble	Destination MAC Address (6 Byte)	Source MAC Address (6 Byte)	Type (2 Byte)	Data Field (1500 Byte Max)	Cyclic Redundancy Check (4 Byte)

รูปที่ 2.25 ลักษณะของเฟรมอีเทอร์เน็ท

2.4.2.3 MAC Address

เนื่องจากคอมพิวเตอร์แต่ละเครื่องสามารถแชร์ข้อมูลกันได้ในระบบเครือข่ายเดียวกัน ดังนั้นแต่ละ เครื่องควรมีสิ่งที่ชี้ลักษณะเฉพาะตัวของมัน เช่น เราต้องมีบัตรประจำตัวประชาชน ซึ่งในทางคอมพิวเตอร์นี้เราจะใช้เลขฐาน 16 จำนวน 12 digits เป็นตัวบ่งชี้ลักษณะเฉพาะนั้น ๆ ซึ่งเราเรียกว่า MAC Address เนื่องจาก MAC Address เป็นตัวบ่งชี้ลักษณะเฉพาะของแต่ละเครื่อง ดังนั้นจึงต้องเป็นค่าที่ไม่ซ้ำกัน (Unique) MAC Addressเป็นเลข 48 บิตโดยแบ่งออกเป็น 2 ส่วนโดย 24 บิตแรกเป็นค่าที่แสดงถึง

บริษัทที่ผลิตการ์ดนั้น ๆ ส่วน 24 บิต หลังเป็น Serial Number ที่ทางบริษัทกำหนดให้ ซึ่งแต่ละตัวต้องไม่ซ้ำกัน เราเรียกเลข 24 บิต นี้ว่า OUI (Organizationally Unique Identifier) ซึ่ง OUI จะใช้เพียง 22 บิต เท่านั้น ส่วนอีก 2 บิตที่เหลือจะถูกใช้เพื่อวัตถุประสงค์อื่น โดยบิตหนึ่งจะใช้เพื่อแสดงว่าแอดเดรสนั้นเป็น Broadcast/Multicast Address ส่วนอีกบิตหนึ่งนั้นไว้แสดงว่า Adapter นั้นถูกกำหนด Locally Administered Address ซึ่ง Admin ของระบบจะทำการกำหนด MAC Address เพื่อความเหมาะสมของนโยบายระบบ เช่น MAC Address = 03 00 00 00 00 00 01 ซึ่งจะเห็นว่าไบต์แรก = 03 = 00000011 นั่นคือทั้ง 2 bits ถูก set (reset = 0) ซึ่งเอาไว้กรณี Multicast ให้ทุกเครื่องที่รันบนโปรโตคอล NetBEUI

2.4.2.4 Type field

ประเภทของฟิลด์จะใช้ระบุความแตกต่างของโปรโตคอล คอมพิวเตอร์จะดำเนินการให้โปรโตคอลหลายๆตัว สามารถเห็นความแตกต่างของพวกมันได้ง่ายและผ่านการยินยอมของเฟรมที่เกี่ยวเนื่องกับเครือข่าย

ระบบ TCP/IP โดยทั่วไปจะใช้อีเทอร์เน็ท 3 ฟิลด์ ซึ่งมีค่าดังตารางที่ 2.4 ประเภทของหมายเลขอีเทอร์เน็ทเป็นรีจีสเตอร์กับ IEEE โดยจะใช้หมายเลขที่เฉพาะเจาะจงไม่เหมือนใคร

ตารางที่ 2.4 Ethernet type fields

Type	Protocol
0x0800	IP
0x0806	ARP
0x0835	RARP

2.4.2.5 Cyclic Redundancy Check (CRC)

ที่ปลายสุดของเฟรม คือ Cyclic Redundancy Check (CRC) มีขนาด 32 บิต ที่คำนวณได้จากบิตทั้งหมดของอีเทอร์เน็ทเฟรม แต่จะไม่สนใจ Preamble ถ้าในเฟรมมีความผิดพลาดเกิดขึ้น ค่าที่คำนวณได้จะแตกต่างไปจากเฟรมเดิม ทำให้ข้อมูลไม่สามารถที่จะผ่านเฟรมไปยังชั้น Network layer ได้

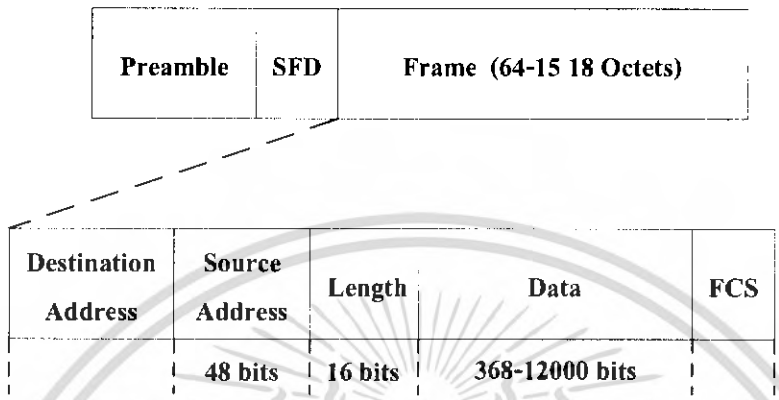
2.4.2.6 IEEE 802.3 เฟรม

IEEE 802.3 หรือ อีเทอร์เน็ท (Ethernet) เป็นเครือข่ายที่มีความเร็วสูงการส่งข้อมูล 10 เมกะบิตต่อวินาที สถานีในเครือข่ายอาจมีโทโปโลยีแบบบัสหรือแบบดาว IEEE ได้กำหนดมาตรฐานอีเทอร์เน็ทซึ่งทำงานที่ความเร็ว 10 เมกะบิตต่อวินาทีไว้หลายประเภทตามชนิดสายสัญญาณเช่น

- 10Base5 อีเทอร์เน็ทโทโปโลยีแบบบัสซึ่งใช้สายโคแอกเชียลแบบหนา (Thick Ethernet) ความยาวของสายในเซกเมนต์หนึ่ง ๆ ไม่เกิน 500 เมตร
- 10Base2 อีเทอร์เน็ทโทโปโลยีแบบบัสซึ่งใช้สายโคแอกเชียลแบบบาง (Thin Ethernet) ความยาวของสายในเซกเมนต์หนึ่ง ๆ ไม่เกิน 185 เมตร

- 10BaseT อีเทอร์เน็ทโทโปโลยีแบบดาวซึ่งใช้ฮับเป็นศูนย์กลาง สถานีและฮับเชื่อมด้วยสายยูทีพี (Unshield Twisted Pair) ด้วยความยาวไม่เกิน 100 เมตร

เฟรมข้อมูลสำหรับระบบอีเทอร์เน็ทประกอบขึ้นด้วยกลุ่มของบิตที่เป็นข้อมูลและข่าวสารสำคัญ แบ่งออกเป็นขนาดสัดส่วนที่แน่นอนที่เรียกว่าช่อง Field ดังรูปที่ 2.26



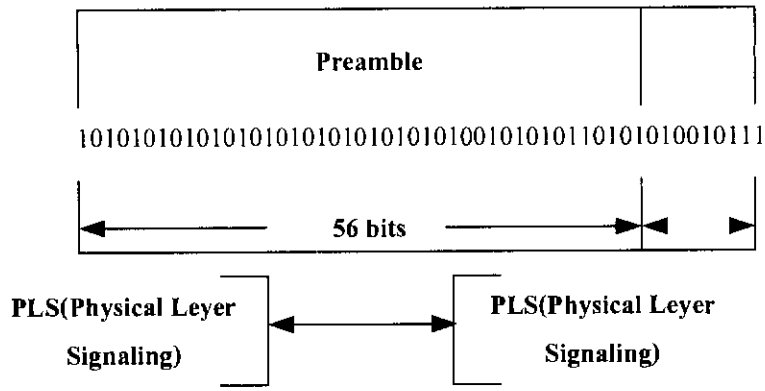
รูปที่ 2.26 ลักษณะโครงสร้างของเฟรมข้อมูลตามมาตรฐาน IEEE802.3

รูปที่ 2.26 แสดงให้เห็นรูปแบบของเฟรมข้อมูลที่ใช้บนอีเทอร์เน็ทตามมาตรฐาน IEEE802.3 ต่อไปเราจะมาทำความเข้าใจเฟรมมาตรฐานของ IEEE802.3

ช่อง Preamble

ช่อง Preamble ประกอบด้วยบิตข่าวสารที่เป็นเลข 1 และ 0 สลับกัน และสิ้นสุดที่ 11 ซึ่งเป็นบิตที่ 63 และ 64 เป็นบิตข่าวสารที่ยังไม่ใช่ข้อมูลจริงของผู้ส่ง Preamble ประกอบด้วยข่าวสารที่มีขนาด 7 หรือ 8 ไบต์ จุดประสงค์ของข่าวสารนี้ก็เพื่อใช้สร้างจังหวะการรับข้อมูลให้แก่ผู้รับโดยที่ส่วนนี้จะไปถึงตัวผู้รับก่อน ทำให้เครื่องคอมพิวเตอร์ของผู้รับสามารถปรับจังหวะความเร็วให้เข้ากับผู้ส่งได้ (Synchronize) สำหรับเฟรมแบบ Ethernet II จะมีขนาด 8 ไบต์ และถ้าเป็นมาตรฐาน IEEE802.3 แล้ว ช่องนี้จะถูกแบ่งออกเป็น 2 ส่วน (ดังรูปที่ 2.27) ได้แก่

1. Preamble
2. Start of Frame Delimiter



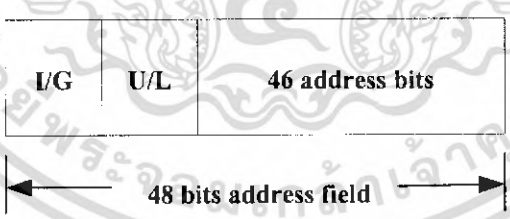
รูปที่ 2.27 ลักษณะส่วนการทำงานภายในของ Preamble

ช่อง Destination Address

ในช่อง Destination Address ประกอบด้วยข้อมูลข่าวสารเกี่ยวกับแอดเดรสหรือที่อยู่ของผู้รับปลายทาง ดูเผินๆแล้วเป็นช่องที่เรียบง่ายไม่มีอะไรมาก แต่โดยความจริงแล้วช่องนี้ถูกแบ่งออกเป็นช่องย่อยๆที่เรียกว่า Sub-Fields ซึ่งเก็บข้อมูลข่าวสารที่ใช้ดูแลการทำงานของเครือข่าย ทั้งนี้ขึ้นอยู่กับแอดเดรสที่ปรากฏอยู่ในช่องนี้ไม่ว่าช่องแอดเดรสนี้จะมีแอดเดรสที่ถูกเจาะจงเป็นรายบุคคลหรือเป็นกลุ่มของผู้รับก็ตาม

ก. ช่องข่าวสารขนาด 2 ไบต์

ข. ช่องข่าวสารขนาด 6 ไบต์



หมายเหตุ

หากค่าบิตในช่องย่อย I/G มีค่าเป็น "0" ก็หมายถึงแอดเดรสที่ระบุตัวคอมพิวเตอร์ผู้รับอย่างเฉพาะเจาะจง แต่ถ้ามีค่าเป็น "1" ก็หมายถึงคอมพิวเตอร์ที่ระบุแอดเดรสเป็นกลุ่ม ไม่เจาะจงเครื่องใดเครื่องหนึ่ง

หากค่าบิตในช่อง U/L มีค่าเป็น "0" ก็หมายความว่าแอดเดรสนี้ถูกกำหนดมาตรฐานโดย IEEE และถ้ามีค่าเป็น "1" ก็จะหมายถึงเป็นแอดเดรสมาตรฐานเฉพาะองค์กรที่ระบุมาตรฐานในซอฟต์แวร์ ซึ่งแอดเดรสมาตรฐานที่เราใช้กันอยู่ทุกวันนี้ถูกควบคุมโดย IEEE

ช่อง I/G

มีการจัดตั้งค่าบิตขึ้นในช่องนี้โดยการ์ต LAN ซึ่งหากค่านี้ถูกตั้งค่าไว้ที่ "0" ก็แสดงว่าตัวแอดเดรสที่ระบุอยู่ในช่อง Destination Address นั้นเป็นแอดเดรสที่ระบุตัวคอมพิวเตอร์ผู้รับแบบเฉพาะเจาะจง แต่ถ้าถูกตั้งค่าเป็น "1" ก็แสดงว่าแอดเดรสในช่อง Destination Address นี้เป็นแอดเดรสที่ใช้ติดต่อผู้รับที่เป็นกลุ่มคอมพิวเตอร์ทั้งหลาย เราเรียก Group Address ตัวอย่างของ Group Address ได้แก่ "FFFFFFFFFFFF" ซึ่งถือว่าเป็น Broadcasting Address หรือแอดเดรสที่ไม่เจาะจงผู้รับ โดยผู้รับเป็นกลุ่มหรือทั้งหมดก็สามารถรับข้อมูลข่าวสารนี้ได้

ช่องย่อย U/L

ช่องย่อย U/L มีไว้สำหรับช่องขนาด 6 ไบต์เท่านั้น ค่าที่ถูกตั้งไว้ในช่องย่อยนี้เป็นการบ่งบอกให้ทราบว่าแอดเดรสที่ปรากฏอยู่ในช่อง Destination Address นี้เป็นแอดเดรสที่ถูกกำหนดมาตรฐานโดย IEEE หรือองค์กรอย่างเฉพาะเจาะจง

ช่อง Source Address

สำหรับช่อง Souree Address นี้มีไว้เพื่อแสดงตัวสถานีเครือข่ายต้นทางที่เป็นต้นทางส่งข้อมูลข่าวสารเข้ามาและเช่นเดียวกับช่อง Destination Address กล่าวคือ ช่อง Source Address สามารถมีช่องย่อยได้ทั้งแบบ 2 ไบต์หรือ 6 ไบต์อย่างใดอย่างหนึ่ง

ตารางที่ 2.5 เป็นตัวอย่างของ Source Address ที่แสดงรหัสแอดเดรสของผู้ผลิตดังนี้คือ

ผู้ผลิตการ์ด LAN	รหัสผู้ผลิตขนาด 3 ไบต์
Cisco	00-00-0C
Cabletron	00-00-1D
Intel	00-AA-00
3 Com	02-60-8C
Hewlett Packard	08-00-09
Sun	08-00-20
DEC	08-00-2B
Shiva	00-80-D3
Xerox	00-00-AA
IBM	08-00-5A

ช่องแสดง Type

ช่องแสดง Type มีขนาด 2 ไบต์ ใช้กับอีเทอร์เน็ทเฟรม เท่านั้น โดยช่องนี้ใช้เพื่อแสดงว่าโปรโตคอลการทำงานของเฟรมนี้เป็นแบบใด จุดประสงค์คือเพื่อต้องการให้ทราบว่าข้อมูลที่อยู่ในเฟรมนี้

จะทำงานภายใต้โปรโตคอลใด ซึ่งผู้รับจะได้เตรียมการแปลความหมายที่อยู่ในช่องข้อมูล (Data Field) ได้ถูกต้อง

ภายใต้ระบบเครือข่ายอีเทอร์เนตเราสามารถใช้โปรโตคอลได้หลายตัวพร้อมกันบนเครือข่าย LAN และบริษัท XEROX ทำหน้าที่เป็นผู้ให้บริการ กำหนดพิกัดระยะของแอดเดรสที่เป็นลิขสิทธิ์ให้แก่ผู้ผลิตการ์ด LAN ต่างๆรวมทั้งการกำหนดค่าที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type แห่งนี้

ตารางที่ 2.6 เป็นตัวอย่างของรหัสที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type 18 รายการดังนี้

โปรโตคอลที่ใช้	ค่าที่เป็นรหัสแบบเลขฐาน 16
IP	0800
X.75 Internet	0801
X.25 Level 3	0805
Address Resolution Protocol (ARP)	0806
Banyan Systems	0BAD
BBN Simnet	5208
DEC MOP Dump/Load	6001
DEC MOP Remote Console	6002
DEC DECNET Phase IV Route	6003
DEC LAT	6004
DEC Diagnostic Protocol	6005
DEC LANBridge	8038
DEC Ethernet Encryption	803D
Apple Talk	809B
IBM SNA Service on Ethernet	80D5
Apple Talk ARP	80F3
NetWare IPX/SPX	8137
SNMP	814C

ช่อง Length

ช่องนี้มีขนาดความยาวเพียง 2 ไบต์ใช้ได้กับเฟรมมาตรฐาน IEEE802.3 เท่านั้นเป็นช่องที่ใช้แสดงขนาดจำนวนของไบต์ที่มีปรากฏอยู่ในช่อง Data

ช่อง Data (Data Field)

ดังที่ได้กล่าวมาแล้วว่าช่องของ Data อย่างน้อยต้องมีขนาดไม่เล็กกว่า 46 ไบต์ เพื่อให้แน่ใจว่าเฟรมมีขนาดไม่ต่ำกว่า 64 ไบต์ซึ่งหมายความว่าการแพร่ข้อมูลขนาดหนึ่งไม่ว่า 1 หรือ 10 ไบต์ก็ตามต้องมาจาก 46ไบต์นี้แต่ถ้าข้อมูลในช่องนี้เล็กกว่า 46ไบต์ แน่นอนว่าต้องมีการเพิ่มไบต์ลงไปอีกเพื่อให้ได้ขนาด 46 ไบต์อยู่ดี

ขนาดของข้อมูลที่อยู่ใน Data จะต้องมีขนาดสูงสุดไม่เกิน 1,500 ไบต์

ช่องตรวจสอบความผิดพลาดของข้อมูลในเฟรม (Frame Check Sequence)

ช่อง Frame Check Sequence นี้ใช้ได้ในเฟรมมาตรฐาน ทั้งเฟรมมาตรฐาน ทั้งอีเทอร์เน็ทและ IEEE802.3 เป็นช่องที่ประกอบด้วยข้อมูลที่ใช้เป็นกลไกในการตรวจสอบความผิดพลาดของข้อมูลภายในเฟรม

หลักการทำงานมีอยู่ว่าก่อนที่เครื่องผู้ส่งจะส่งข้อมูลออกไปที่เครือข่าย การ์ด LAN ของมันจะคำนวณค่าต่างๆในช่องต่างๆซึ่งครอบคลุมตั้งแต่ช่อง Address ต่างๆของ Type และช่อง Length รวมทั้งช่อง Data การคำนวณค่าแบบนี้เรียกว่า Cyclic Redundancy Check (CRC) ซึ่งหลังจากที่ได้คำนวณค่าเสร็จสิ้นแล้ว ผลลัพธ์ที่คำนวณได้มีขนาด 4 ไบต์จะถูกนำไปใส่ไว้ในช่อง Frame Check Sequence แห่งนี้

2.4.3 Internetwork Layer

ในระดับล่างต่อมาในชั้น Internetwork Layer มีหน้าที่ส่งผ่านข้อมูลในระหว่างเครือข่าย โดยมีโพรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใด ๆ บนอินเตอร์เน็ท คือ โพรโตคอล IP (Internet Protocol) นอกจากนี้ในชั้น Internetwork Layer ยังมีโพรโตคอลทำงานอยู่ด้วยอีก 2 ชนิดคือ โพรโตคอล Internet Control Message Protocol (ICMP) และโพรโตคอล Address Resolution Protocol (ARP) ซึ่งอยู่ภายในโพรโตคอล IP

2.4.3.1 โพรโตคอล IP (Internet Protocol)

โพรโตคอล IP ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาจาก Host – to – Host Layer เพื่อส่งข้ามไปยังเครือข่ายใด ๆ ได้อย่างถูกต้อง แม้ว่าจะมีเครือข่ายเชื่อมต่อกันอยู่ในอินเตอร์เน็ทเป็นล้าน ๆ เครือข่ายก็ตาม เนื่องจากโพรโตคอล IP มีข้อมูลตำแหน่ง IP ปลายทางที่จะส่งข้อมูลไปให้โดยทำงานร่วมกับอุปกรณ์ Router เพื่อส่งข้อมูลข้ามเครือข่ายออกไปได้ ตัวโพรโตคอล IP จะทำงานแบบ Packet Switching คือมีการส่งข้อมูลผ่านสวิทช์ (Switch) ไปยังปลายทาง โดยข้อมูลจะเดินทางไปยังเครือข่ายต่าง ๆ ผ่านสวิทช์นี้ไปเรื่อยๆ จนกว่าจะถึงปลายทาง ตัววงจรผ่านหรือสวิทช์นี้อาจเป็น Gateway หรือ Router ในระบบเครือข่ายก็ได้ ซึ่งในข้อมูลของโพรโตคอล IP จะมีข้อมูลของหมายเลข IP ที่จะส่งข้อมูลไปและเมื่อถึงเครือข่ายปลายทางแล้ว จะมีกลไกแปลงหมายเลข IPให้เป็นหมายเลขฮาร์ดแวร์ประจำเครื่องที่ถูกต้องอีกทีหนึ่งด้วยโพรโตคอล ARP

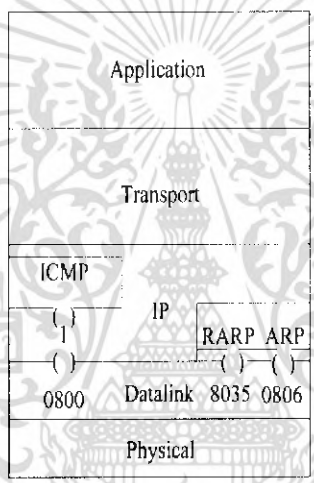
IP จะให้บริการชนิด Connectionless สำหรับ User ดังนั้น Data ที่ถูกส่งผ่าน IP โดยกระบวนการ Send ยังไม่แน่ว่าจะสามารถส่งถึง ข้อมูลของ IP หน่วยต่างๆที่ถูกส่ง เราเรียกว่า IP Datagram ซึ่งผ่านการ

Encapsulation ข้อมูลที่ถูกส่งมาจาก Layer ที่สูงกว่าด้วย IP Header ถ้า IP Datagram ไม่ได้ถูกนำส่งภายในเวลาที่กำหนด (Time – to –Live) Datagram นั้นก็จะไม่ถูกส่งอีกเลย สำหรับช่วงเวลา Time – to –Live นั้นสามารถกำหนดไว้ในกระบวนการส่ง

2.4.3.2 ส่วนประกอบของ IP

IP Address จะต้องมีค่าเฉพาะเจาะจงไม่เหมือนใคร เพื่อที่จะใช้เชื่อมต่อเข้ากับเครือข่ายที่หมายเลขของเครือข่ายที่เฉพาะเจาะจงเช่นกัน โดยในรูปที่ 2.26 แสดงให้เห็นว่าใน IP Layer จะประกอบไปด้วย 3โปรโตคอล ได้แก่

1. The Address Resolution Protocol (ARP)
2. The Reverse Address Resolution Protocol (RARP)
3. The Internet Control Message Protocol (ICMP)



รูปที่ 2.28 ส่วนประกอบของ IP

ARP และ RARP จะอยู่ที่ส่วนปลายสุดของ IP Layer เพราะทั้งสองโปรโตคอลไม่ใช้ IP และจะถูกแยกโปรโตคอลโดย Data link Layer ที่สนับสนุนตัวมันอยู่ ส่วน ICMP จะอยู่ในส่วนบนสุดของ IP Layer ซึ่งจะข้ามเข้าไปในเครือข่ายใน IP Datagram

2.4.3.3 IP Datagram

Datagram เป็น Basic Unit ของข้อมูลที่ IP จะทำการส่ง ซึ่งถูกออกแบบมาให้ทำงานกับเครือข่ายแบบ Packet Switch ซึ่งข้อมูลของผู้ใช้มักถูกแบ่งออกเป็นหลาย Datagram โดยแต่ละตัวจะมี Header ที่เก็บรายละเอียดเกี่ยวกับตัวมันเอง และปลายทางที่มันจะไป Datagram แต่ละตัวจะเดินทางโดยไม่เกี่ยวข้องกัน นั่นคือ Datagram แต่ละตัวอาจจะเดินทางไปยังปลายทางโดยใช้เส้นทางคนละเส้น และลำดับที่ของการไปถึงจุดหมายปลายทางก็ไม่แน่นอน เป็นหน้าที่ของ Internet Protocol ในเครื่องปลายทางที่จะต้องประกอบ Datagram เหล่านี้ให้กลายเป็นข้อมูลของผู้ใช้ที่สมบูรณ์อีกครั้ง

IP Datagram ประกอบด้วย IP Header และ IP Payload

IP Header	IP Payload

2.4.3.3a **IP Header** เป็นขนาดที่เปลี่ยนแปลงได้ระหว่าง 20 และ 60 ไบต์ ในการเพิ่มขึ้น 4 ไบต์ มันจะจัดเตรียมการสนับสนุน Routing , การแสดงตัว Payload , การชี้ให้เห็นถึงขนาด IP Header และ Datagram , การสนับสนุน Fragmentation โดยมีโครงสร้างดังรูปที่ 2.29

Version	IP Header Length	Type of Service	Total Length	Identifier	Flags	Fragment Offset
4 bit	4 bit	8 bit	16 bit	16 bit	3 bit	13 bit

Time – to – Live	Protocol	Header Checksum	Source IP Address	Destination IP Address	IP Option and Padding
8 bit	8 bit	16 bit	32 bit	32 bit	32 bit

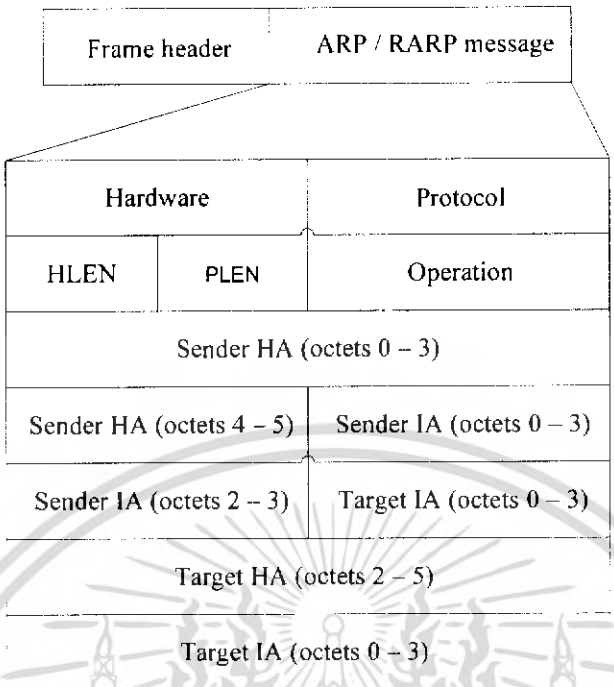
รูปที่ 2.29 แสดงโครงสร้าง IP Header

2.4.3.3b **IP Payload** เป็นขนาดที่เปลี่ยนแปลงโดยมีลำดับจาก 8 ไบต์ (68 ไบต์ IP Datagram กับ 60 ไบต์ IP Header) ถึง 65,515 ไบต์ (65,535 ไบต์ IP Datagram กับ 20 ไบต์ IP Header)

2.4.3.4 Address Resolution Protocol (ARP)

โพรโตคอล ARP (Address Resolution Protocol) ถูกเรียกใช้งานโดยโพรโตคอล IP เพื่อช่วยแปลงหมายเลข IP ไปเป็นหมายเลขฮาร์ดแวร์ปลายทาง ตัวอย่างเช่น เว็บเซิร์ฟเวอร์เครื่องหนึ่งเชื่อมต่ออยู่ในเครือข่ายอินเตอร์เน็ท และในการเชื่อมต่อนี้ต้องอาศัย Network Interface Card (NIC) หรือ LAN Card ติดตั้งอยู่ ที่LAN Card นี้เองจะมีหมายเลขเฉพาะประจำฮาร์ดแวร์ที่ไม่ซ้ำกับใคร เพื่อใช้อ้างอิงการส่งข้อมูลในเครือข่าย แต่เมื่อมาใช้งานในโพรโตคอล TCP/IP ก็จะต้องมีการกำหนดหมายเลข IP Address ประจำตัวเพื่อใช้อ้างอิงกัน และโพรโตคอล ARP จะทำหน้าที่แปลงค่าหมายเลข IP ให้เป็นหมายเลขฮาร์ดแวร์จริงให้ในระดับการทำงานที่ Internetwork Layer นี้ ซึ่งกลไกการแปลงนี้เรียกว่า Address Resolution

2.4.3.4a ARP Datagram



รูปที่ 2.30 แสดง ARP Datagram

ในรูปที่ 2.30 แสดง ARP Datagram โดยที่มันไม่สามารถนำมาใช้ได้ทั้งหมดสำหรับ TCP/IP หรือเครือข่ายใดเครือข่ายหนึ่งโดยเฉพาะ โดยมันแค่ถูกกำหนดให้เป็นตัวกลางที่มีความสามารถทำการส่งเฟรม Broadcast กระบวนการของ ARP จะดำเนินการโดยตรงเข้าไปอยู่เหนือ Datalink Layer และจะทำการ Encapsulate ในเฟรม Datalink โดย ARP Datagram จะประกอบไปด้วย

2.4.3.4b กระบวนการทำงานของ ARP

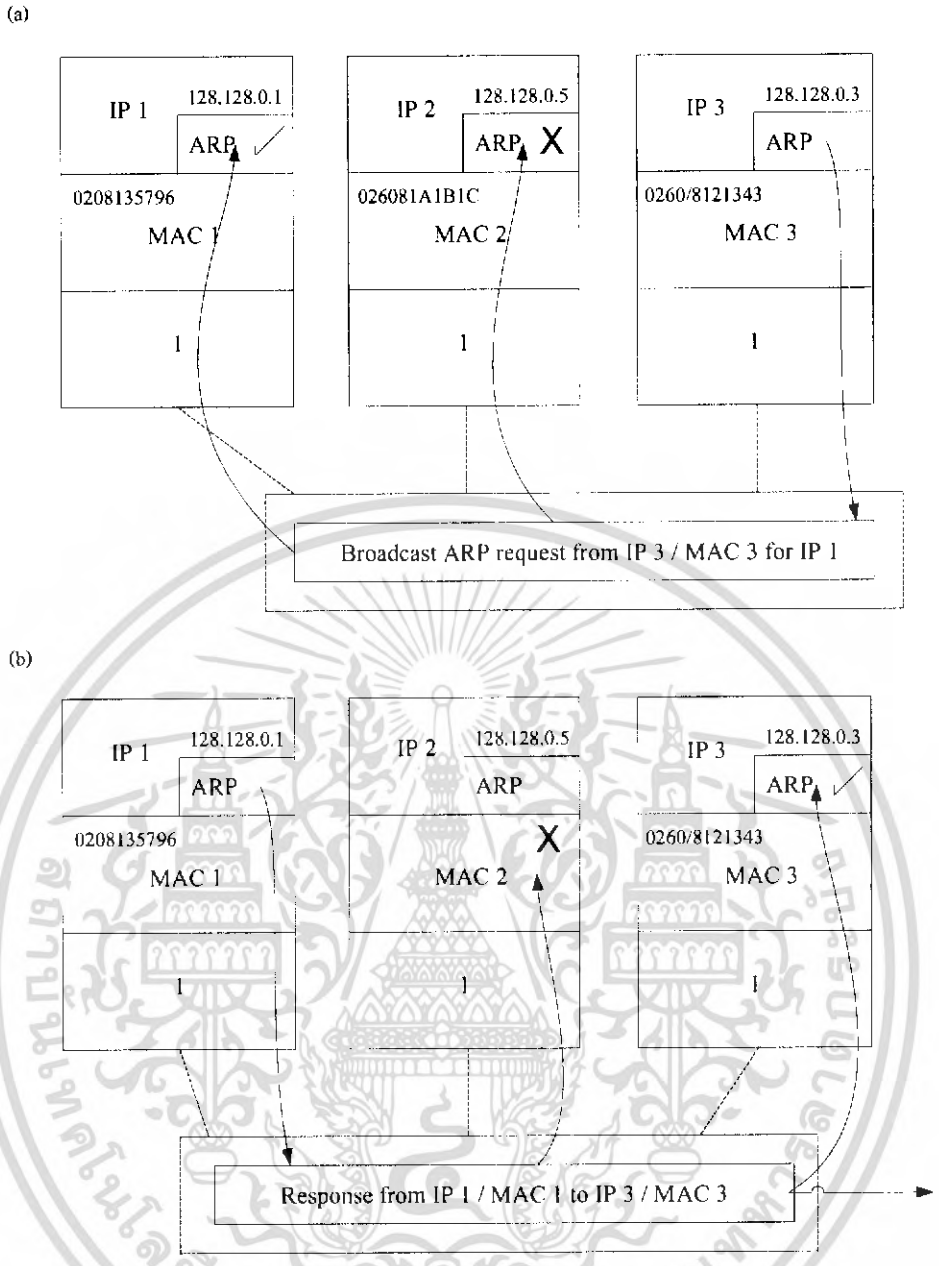
การติดต่อสื่อสารระหว่างคอมพิวเตอร์บนเครือข่ายนั้นจะต้องอาศัยค่า MAC Address เป็นสำคัญ ฉะนั้นหากว่าเครื่องส่งไม่ทราบค่า MAC Address ของเครื่องรับแล้ว การส่งข้อมูลก็ไม่สามารถเกิดขึ้นได้ ดังนั้นจำเป็นที่ต้องมีกลไกที่ใช้ค้นหาค่า MAC Address ของเครื่องรับ ในแบบจำลอง TCP/IP ได้มีการเลือกใช้ ARP ซึ่งเป็นกลไกที่อนุญาตให้ IP Protocol ค้นหาค่า MAC Address ที่สัมพันธ์กับ IP Address

การทำงานของ ARP ประกอบไปด้วยกระบวนการต่อไปนี้

1. เครื่อง Client จะใช้ค่าของ IP Address เพื่อค้นหา MAC Address ที่สอดคล้องกัน ซึ่งเก็บไว้ใน ARP Cache
2. ถ้าไม่พบค่า MAC Address สำหรับ IP Address ดังกล่าวใน ARP Cache แล้วเครื่อง Client ก็จะทำการส่ง ARP Request Packet ด้วยวิธีการ Broadcast ในระดับชั้นย่อย MACให้กับทุกๆ Host ใน LAN ทราบ

3. ถ้าหากมี Host ที่มีค่า IP Address สอดคล้องกับข้อมูล ARP Request Packet ดังกล่าวมันจะทำการส่ง ARP Reply Packet แบบ Unicast ในระดับชั้นย่อย MAC ซึ่งภายในมีข้อมูลของ MAC Address และ IP Address ของ Host ดังกล่าวกลับไปที่เครื่อง Client
4. เครื่อง Client จะบันทึกข้อมูลค่า MAC Address และ IP Address ดังกล่าวลงบน ARP Cache

ในรูปที่ 2.29 แสดงตัวอย่างการการดำเนินการของ ARP โดย IP Address 128.128.0.3 และค่า MAC Address 0x0268121343 จะทำการร้องขอโดยตัวมัน IP Layer ก็จะค้นหาค่า MAC Address กับ IP Address 128.128.0.1 โดยเริ่มแรก 128.128.0.3 จะส่ง ARP Request ที่จะถูกรับโดย ARP Software บนทุกๆจุดในเครือข่าย 128.128.0.1 จะยอมรับ Address ของมันในการร้องขอ และจะส่ง ARP Reply กลับไป แต่จะใช้ค่า MAC Address ที่ ARP Request ส่งมา ARP Reply จะถูกดำเนินการ และถูกละทิ้งไปอย่างรวดเร็วโดย Card LAN ที่ทุกๆจุดบนเครือข่าย ขณะที่มันเป็น Unicast Frame กับ Destination Address ที่ผิด



รูปที่ 2.31 ตัวอย่างกระบวนการของ ARP, (a) ARP Request, (b) ARP Response

2.4.3.5 Reverse Address Resolution Protocol (RARP)

วิธีการ ARP ช่วยแก้ปัญหาในการค้นหาที่อยู่ของข้อมูลที่ใช้การกำหนดที่อยู่แบบฮาร์ดแวร์ แต่ถ้าทราบที่อยู่แบบฮาร์ดแวร์แล้วต้องการแปลงที่อยู่เป็น IP จะทำอย่างไร ปัญหานี้มักเกิดขึ้นกับเครื่อง Computer ที่เริ่มทำงานด้วยการอ่านข้อมูลทั้งหมดจากเครื่อง Host เครื่องประเภทนี้จะทราบเพียงที่อยู่ ของตนเองจากอุปกรณ์สื่อสารเครือข่ายเท่านั้น

การค้นหาคำตอบสามารถทำได้โดยวิธีควบคุมการสื่อสารแบบ ARP ย้อนกลับ หรือ RARP (Reverse Address Resolution Protocol) วิธีการนี้ Computer ที่เพิ่งจะเริ่มทำงาน (หรือเครื่องใดก็แล้วแต่) จะส่งคำถามออกไปในทำนอง "ที่อยู่ขนาด 48 บิตแบบฮาร์ดแวร์ของฉันคือ 14.04.05.18.01.25 มีใครทราบ

ที่อยู่ IP ของฉันบ้าง" เครื่องที่ให้บริการ RARP จะตรวจดูข้อมูลในตารางข้อมูลของตนเองแล้วจึงส่งหมายเลข IP กลับไปให้ วิธีการนี้ช่วยให้เกิดความอ่อนตัวและเพิ่มประสิทธิภาพในการใช้หมายเลข IP เนื่องจากผู้ใช้ไม่มีหมายเลข IP เป็นของตนเอง ผู้ควบคุมระบบสามารถกำหนดหมายเลข IP ใดๆที่ไม่มีผู้ใช้งานในขณะนั้นให้ใช้ได้ หมายเลข IP ในที่นี้จึงเป็นเสมือนสมบัติส่วนกลางที่ทุกคนใช้ร่วมกัน

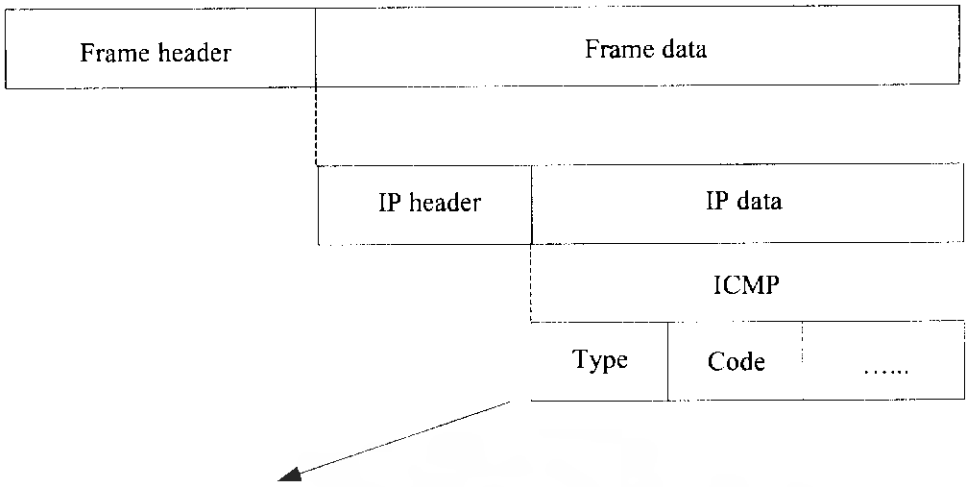
2.4.3.6 Internet Control Message Protocol (ICMP)

ถึงแม้ว่า IP จะเป็น Datagram Service และไม่มีการรับประกันรูปแบบการส่ง โดย Internet Control Message Protocol (ICMP) จะถูกจัดเตรียมไว้ภายใน IP ทำให้เกิด Error Messages ให้เข้าไปช่วย IP Layer ให้มีความสามารถในการส่งที่ดีที่สุด โดยหน้าที่หลักของโปรโตคอล ICMP (Internet Control Message Protocol) คือ การแจ้งหรือแสดงข้อความจากระบบ เพื่อบอกให้ผู้ใช้ ทราบว่า เกิดอะไรขึ้นในการส่งผ่านข้อมูลนั้น ซึ่งปัญหาส่วนมากที่พบคือ ส่งไปไม่ได้ หรือปลายทางรับข้อมูลไม่ได้ เป็นต้น นอกจากนี้ โปรโตคอล ICMP ยังถูกเรียกใช้งานจากเครื่อง Server และ Router อีกด้วย เพื่อแลกเปลี่ยนข้อมูลที่ใช้ควบคุม ส่วนรูปแบบการทำงานของโปรโตคอล ICMP นั้นจะทำควบคู่กับโปรโตคอล IP ในระบบเดียวกัน และข้อความต่าง ๆ ที่แจ้งให้ทราบจะถูกผนึกอยู่ภายในข้อมูล IP (IP datagram) อีกทีหนึ่ง

รูปที่ 2.26แสดงรูปแบบพื้นฐานของ ICMP message encapsulated ใน IP Datagram ใน ICMP มีหมายเลข IP โปรโตคอลของตัวมันเอง ดังนั้น IP Layer รู้ว่ามันรับ ICMP แม้ว่า ICMP ใช้ IP Layer ที่เป็นตัวพิจารณาว่า IP ภายในเป็นของใคร เพราะว่ามันไม่สามารถจัดเตรียมให้กับ Layer ที่สูงกว่าได้

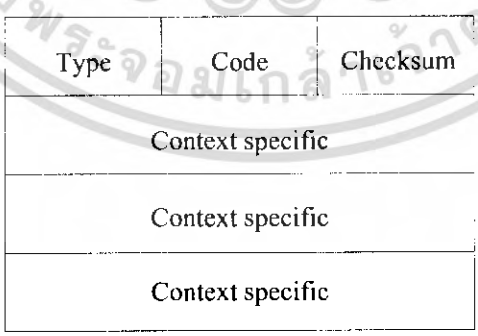
นับตั้งแต่ IP Message ที่ถูกขนย้ายใน IP มันจะถูกทิ้งไปเหมือนกับ IP Datagram โดยมันจะไม่สามารถรักษาสถานภาพไว้ได้ ICMP Message จะไม่ถูกทำให้เกิดขึ้นในกรณีที่ ICMP Message เกิดความผิดพลาดเกิดขึ้น

รูปแบบพื้นฐานของ ICMP Datagram แสดงไว้ในรูปที่ 2.30โดยจะประกอบไปด้วยการแบ่งประเภทของ ICMP Message และ Code ที่ต้องจัดเตรียมรายละเอียด Checksum ต้องถูกนำมาใช้ เพราะ IP ไม่สามารถที่จะป้องกันข้อมูลของมันได้ เมื่อทำการดำเนินการมาอยู่เหนือ Physical Network ที่มี Frame Check Sequence ICMP Checksum ต้องเป็น 0 นั่นหมายความว่าจะไม่มีการคำนวณเกิดขึ้น



Type field	Message type
0	Echo reply
3	Destination unreachable
4	Source quench
5	Redirect (change route)
8	Echo request
11	Time exceeded for datagram
12	Parameter problem on datagram
13	Time stamp request
14	Time stamp reply
15	Information request
16	Information reply
17	Address mask request
18	Address mask response

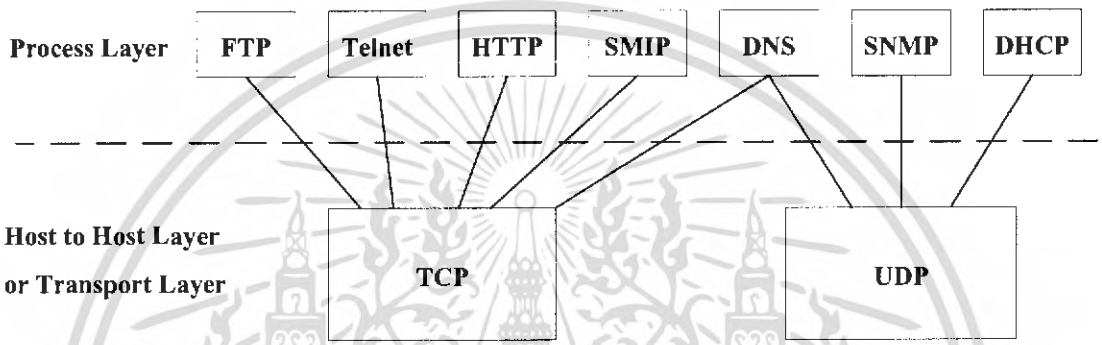
รูปที่ 2.32 ICMP encapsulated ใน IP และประเภทของ ICMP



รูปที่ 2.33 รูปแบบของ ICMP Datagram

2.4.4 Host-To-Host Layer

การทำงานที่ชั้นของ Host-to- Host Layer นี้จะมีบทบาทในการจัดการต่อจาก Process Layer บางครั้งเรามักเรียกชั้น Host-to-Host ว่าเป็น Transport Layer ซึ่งไม่ใช่ชั้นของ Transport Layer ในมาตรฐาน OSI Model การทำงานของ Host-to-Host Layer นี้จะมีการสร้าง Connection หรือการเชื่อมต่อกันระหว่างแอพพลิเคชันกับ Host – to – Host Layer โดยจุดที่เชื่อมกันเพื่อรับส่งข้อมูลที่เรียกว่า พอร์ท หรือ Socket (คำว่าพอร์ทในที่นี้ไม่ได้หมายถึง พอร์ททางฮาร์ดแวร์) และในแต่ละแอพพลิเคชันก็จะสร้างการเชื่อมต่อผ่านพอร์ทได้พร้อมกันหลายแอพพลิเคชัน ซึ่งการใช้งานพอร์ทของแต่ละแอพลิเคชันที่อยู่ในชั้น Process Layer จะแตกต่างกันตามหมายเลขที่กำหนดไว้ และแต่ละโพรโตคอลจะมีการใช้งาน port หมายเลขต่าง ๆ ไม่ซ้ำกัน ดังรูปที่ 2.34

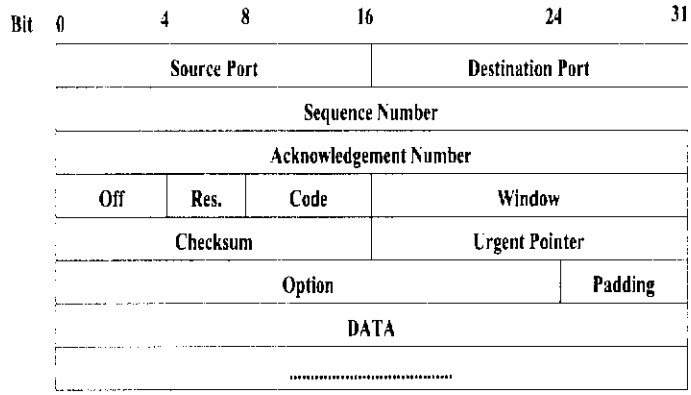


รูปที่ 2.34 แสดงการใช้งานพอร์ทของแต่ละโพรโตคอล

ในชั้น Host-to-Host หรือ Transport Layer ของ TCP/IP นี้จะมีโพรโตคอลทำงานอยู่ 2 โพรโตคอลที่แตกต่างกัน คือ โพรโตคอล TCP และโพรโตคอล UDP (User Datagram Protocol) ในการส่งผ่านข้อมูลลงไปที่ชั้นถัดๆ ไป เราจะเห็นว่าโพรโตคอล TCP และ UDP จะถูกผนึกเข้าไปในโพรโตคอล IP อีกทีหนึ่งและส่งต่อไปยังเครือข่ายอินเตอร์เน็ตต่อไป

2.4.4.1 โพรโตคอล TCP

โพรโตคอล TCP (Transmission Control Protocol) เป็นโพรโตคอลที่มีการรับส่งข้อมูลแบบ stream oriented protocol หมายความว่า การรับส่งข้อมูลจะไม่คำนึงถึงปริมาณข้อมูลที่จะส่งไป แต่จะแบ่งข้อมูลเป็นส่วนย่อย ๆ ก่อน แล้วจึงส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งสูญหายไป ก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้ง สำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูล datagram ดังนั้นแอพพลิเคชันหรือโปรเซ็สใดที่อาศัยการส่งผ่านข้อมูลด้วยโพรโตคอล TCP จะต้องใช้หน่วยความจำและขนาดของช่องสัญญาณมากกว่า UDP



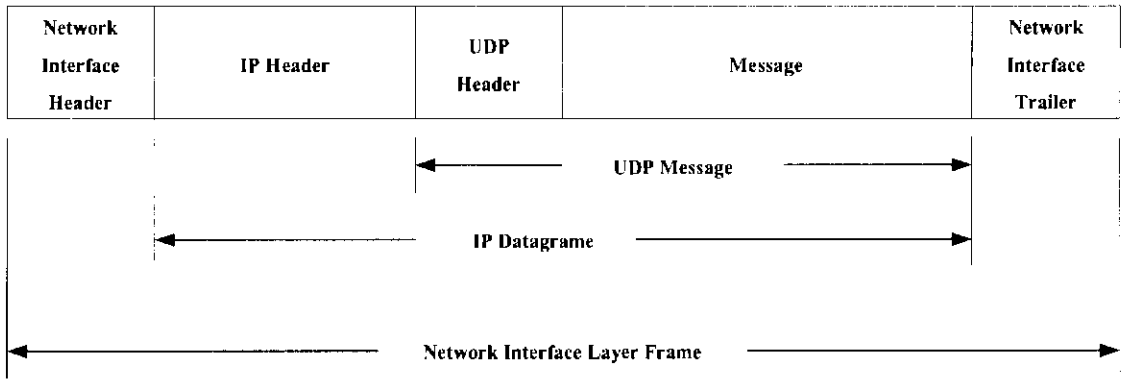
รูปที่ 2.35 แสดงโครงสร้างของโปรโตคอล TCP

2.4.4.2 โพรโตคอล UDP

ใน Host-to-Host Layer นอกจากจะมีโพรโตคอล TCP ทำงานแล้ว ก็ยังมีโพรโตคอล UDP (User Datagram Protocol) ในการส่งข้อมูลแต่ละครั้งและไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชันหรือโปรเซ็สใดที่ต้องอาศัยโพรโตคอล UDP จะเป็นแบบที่ทั้งสองด้านไม่จำเป็นต้องอาศัยการสร้างช่องทางเชื่อมต่อกัน (Connectionless) ระหว่างเครื่องเซิร์ฟเวอร์ให้บริการกับเครื่องที่ขอใช้บริการ โดยไม่ต้องแจ้งให้ฝ่ายรับข้อมูลเตรียมรับข้อมูลเหมือนโพรโตคอล TCP และไม่มีการตรวจสอบความถูกต้องครบถ้วนในการรับส่งข้อมูลนั้นๆ ด้วย เนื่องจากโพรโตคอล UDP ไม่มีสัญญาณสอบทานข้อมูล (acknowledgement) ในการส่งข้อมูลแต่ละครั้ง และไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชันหรือโปรเซ็สใดที่ต้องอาศัยโพรโตคอล UDP ในการส่งผ่านข้อมูลก็อาจจะต้องสร้างขบวนการตรวจสอบข้อมูลขึ้นมาเอง

UDP Message

UDP Message ประกอบด้วย UDP Header และ Message โดยจะนำไปรวมกับ IP Header เพื่อเข้าสู่ IP Datagram ซึ่งใช้ IP Protocol หมายเลข 17 (0x11) โดย message สามารถมีขนาดสูงสุดได้ถึง 65,535 ไบต์ และมีขนาดเล็กที่สุดเท่ากับ 65,507 ไบต์ โดยเล็กกว่า IP Header (20 ไบต์) และ UDP Header (8 ไบต์) IP Datagram ที่ได้มันจะเป็น encapsulated กับ Network Interface Layer ที่เหมาะสม



รูปที่ 2.36 UDP Message Encapsulation

UDP Header

มีขนาด 8 ไบต์ โดยประกอบด้วย 4 ส่วนดังรูปที่ 2.37

Source Port	Destination Port	Length	Checksum
2 ไบต์	2 ไบต์	2 ไบต์	2 ไบต์

รูปที่ 2.37 แสดงโครงสร้างของ UDP Header

- **Source Port** มี 2 ไบต์ใช้ระบุเป็น Source Application Layer Protocol ทำการส่ง UDP Message โดย Source Port เป็นพอร์ทที่ใช้ในการเลือก เมื่อใดที่ไม่ได้ใช้มัน มันจะตั้งค่าเป็น 0x00-00 IP Multicast Traffic เปรียบเสมือน Video casts ใช้ส่ง UDP สามารถใช้ค่า 0x00-00 เพราะจะไม่ตอบรับ Video Traffic เป็นเพียงการสมมุติ Application Layer ใช้ Source Port ในการนำ UDP Message เข้ามา Destination Port สำหรับการตอบรับ

- **Destination Port** มี 2 ไบต์ใช้ระบุเป็น Destination Application Layer Protocol การรวมของ Destination IP Address ของ IP Header และ Destination Port ของ UDP Header จะไม่เหมือนใครสำหรับกระบวนการที่จะส่งข้อมูล

- **Length** มี 2 ไบต์ที่ใช้ในการแสดงความยาวใน UDP Message มีความยาวน้อยที่สุด 8 ไบต์ (ขนาดของUDP Header) และมากที่สุด 65,515 ไบต์ (ค่าสูงสุด IP Datagram 65,535 ไบต์ น้อยกว่าค่าน้อยที่สุด IP Header 20 ไบต์) ความยาวมากที่สุดที่แท้จริงถูกจำกัดโดย MTU ซึ่งจะทำการเชื่อมโยงโดย UDP Message เป็นตัวส่ง ความยาว UDP สามารถคำนวณได้จากความยาวทั้งหมดและความยาวของ IP Header Field ใน IP Header

- **Checksum** มี 2 ไบต์ โดยจะทำการตรวจระดับของบิตอย่างสมบูรณ์สำหรับ UDP Message โดยที่ UDP Checksum คำนวณโดยใช้วิธีเดียวกันกับ IP Header Checksum

UDP Checksum

Checksum เป็น เลข 16 บิตถูกคำนวณด้วยวิธี 1's Complement โดยนำ Pseudo Header และข้อมูลทั้งหมดใน UDP Datagram มาคำนวณ

Pseudo Header เป็นข้อมูลที่อยู่ในส่วนของ IP Header ประกอบด้วยฟิลด์ Source IP Address, Destination IP Address, Zero, Protocol, UDP Length ดังแสดงในรูปที่ 2.38

16-bit Source IP address		
16-bit Destination IP address		
Zero	8-bit protocol (17 for UDP)	16-bit length

รูปที่ 2.38 Pseudo Header

หากค่า Checksum ที่คำนวณออกมาเป็น 0 ค่า checksum จะถูกเซตเป็น 1 ทั้งหมดแทน (มีค่าเท่ากันในระบบ 1's complement) ทั้งนี้เพราะในบางแอพพลิเคชั่นที่ไม่ต้องการตรวจสอบค่า Checksum ในระดับ UDP จะเซตค่านี้เป็น 0 (Disable Checksum)

2.4.5 Process Layer

การแสดงลำดับชั้นการทำงานของโพรโตคอล TCP/IP เทียบกับมาตรฐาน OSI model นั้น ในชั้นบนสุดเรียกว่า Process Layer ทำงาน 2 หน้าที่เทียบได้กับ Application Layer และ Presentation Layer ในชั้นนี้จะรองรับการทำงานของแอพพลิชันต่าง ๆ ที่ทำงานเป็นโปรเซ็ส อยู่ในเครื่องเซิร์ฟเวอร์ที่ให้บริการและเครื่องที่ขอใช้บริการ หรือไคลเอนต์ (Client) ซึ่งจะติดต่อกันผ่านโพรโตคอลเฉพาะแอพพลิเคชันอีกทีหนึ่ง ตัวอย่างเช่น เมื่อผู้ใช้งานอินเตอร์เน็ตต้องการโอนถ่ายไฟล์หรือ Download ข้อมูลจากเครื่องเซิร์ฟเวอร์ที่ให้บริการ โดยอาจจะเรียกใช้โปรแกรม FTP Client ทั่วไป เช่น โปรแกรม WS_FTP ติดต่อกับโปรเซ็ส FTP ที่กำลังให้บริการอยู่ที่เครื่องเซิร์ฟเวอร์ จากนั้นตัวโปรเซ็ส FTP ก็จะเรียกใช้โพรโตคอล FTP (File Transfer Protocol) เพื่อทำการโอนถ่ายไฟล์นี้ หรือถ้าผู้ใช้ต้องการเรียกใช้งานคอมพิวเตอร์ที่อยู่ห่างไกลออกไปด้วยการใช้โปรแกรม Telnet ที่เครื่องเซิร์ฟเวอร์ให้บริการ ตัวโปรเซ็ส Telnet ที่ทำงานอยู่ก็จะเรียกใช้โพรโตคอล Telnet เพื่อติดต่อกัน หรือในกรณีที่มีการเรียกใช้โปรแกรม Web Browser เช่น Netscape Navigator เพื่อเรียกดูเว็บไซท์ CNN ที่เครื่องซึ่งให้บริการเว็บของ CNN ก็จะมีโปรเซ็ส HTTP (Hypertext Transfer Protocol) ทำงานอยู่และจะติดต่อกับผู้ใช้ผ่านโพรโตคอล HTTP เป็นต้น

การทำงานของแอพพลิเคชันต่าง ๆ จะอยู่ที่ Process Layer นี้ และมีการติดต่อกันตามแต่ละโพรโตคอลเฉพาะแล้วแต่แอพพลิเคชันที่ใช้งาน จากการที่ Process Layer ของ TCP/IP รองรับให้โพรโตคอลอื่นทำงานได้หลายโปรเซ็สและหลายโพรโตคอลได้พร้อมกันนั้น ทำให้ผู้ใช้สามารถเปิดโปรแกรมใช้งานได้หลาย ๆ โปรแกรมพร้อมกัน เช่น เปิดโปรแกรม Internet Explorer เพื่อเรียกดูเว็บเพจพร้อมกับใช้งานโปรแกรม Outlook Express เพื่อรับส่งอีเมล์ไปพร้อมกันได้โดยไม่ต้องรอให้ทำงานอย่าง

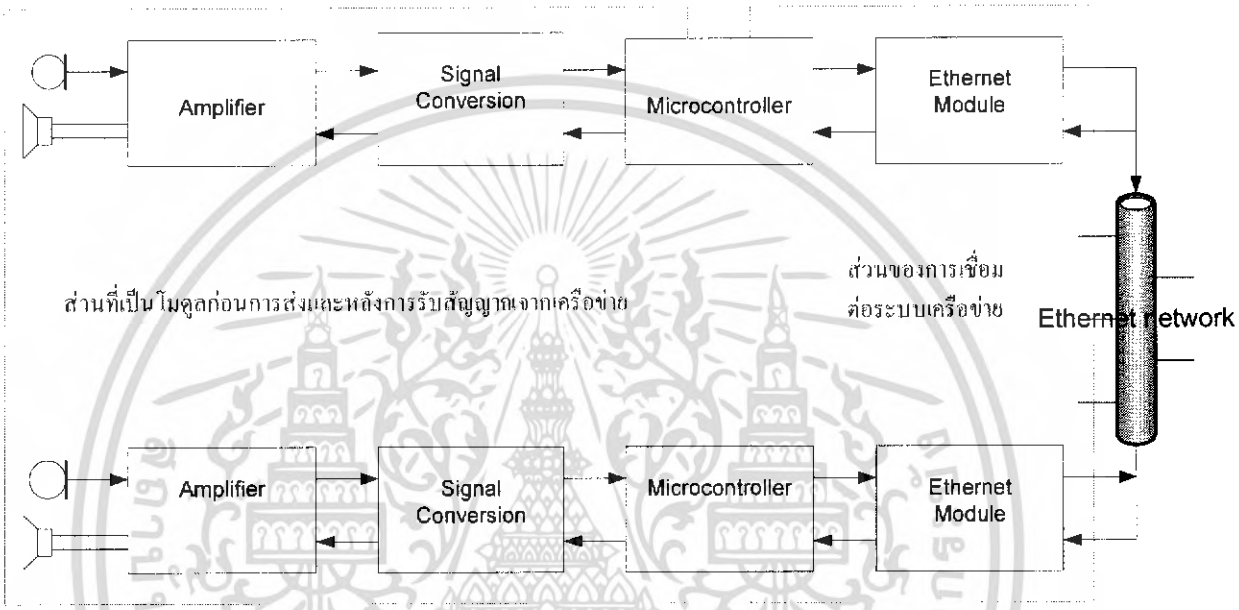
หนึ่งอย่างใดเสร็จก่อน หรือในปัจจุบันมีการพัฒนาโปรแกรม Web Browser ให้สามารถเรียกใช้งานโพรโตคอลอื่น ๆ ได้มากขึ้น ทำให้เราสามารถใช้โปรแกรม Web Browser โอนถ่ายไฟล์ข้อมูลที่ใช้โพรโตคอล FTP ได้โดยไม่ต้องไปหาโปรแกรมอื่นมาใช้

โพรโตคอลหลัก ๆ ที่ทำงานใน Process Layer ซึ่งผู้ใช้มักจะคุ้นเคยกันดีได้แก่ FTP (File Transfer Protocol), Telnet, HTTP(Hyper Text Transfer Protocol), SMT(Simple Mail Transfer protocol) นอกจากนี้ยังมีโพรโตคอลอื่นที่อยู่เบื้องหลัง ซึ่งทำงานโดยที่ผู้ใช้ไม่ได้มีการใช้งานโดยตรง เช่น

- โพรโตคอล DNS (Domain Name System) ที่ทำหน้าที่แปลงข้อมูลชื่อ Domain Name หรือชื่อเว็บไซท์ทั้งหลายให้เป็นหมายเลข IP address
- โพรโตคอล DHCP (Dynamic Host Configuration Protocol) ทำหน้าที่แจกจ่ายข้อมูลพารามิเตอร์ของเครือข่ายให้กับเครื่องลูกข่ายที่เชื่อมต่ออยู่

บทที่ 3
การคำนวณและการสร้าง

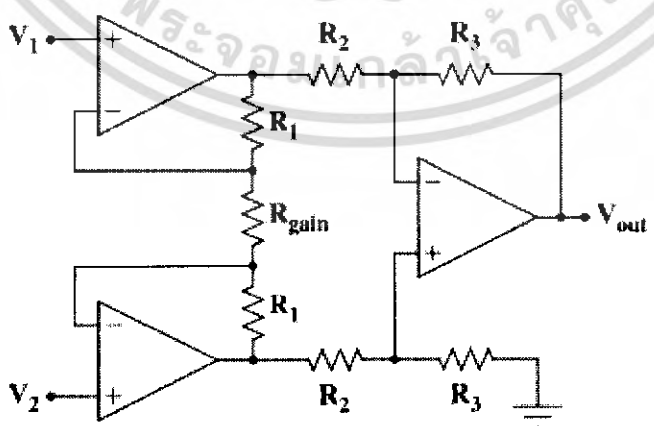
ในบทนี้จะกล่าวถึงการคำนวณต่างๆที่ใช้ในการออกแบบวงจร เพื่อนำไปสู่การสร้างฮาร์ดแวร์ของระบบการส่งเสียงผ่านเครือข่ายอีเทอร์เน็ท และเพื่อง่ายต่อการทำความเข้าใจจึงได้แบ่งการอธิบายออกเป็น 2 ส่วนหลักๆ คือ ส่วนที่เป็นโมดูลก่อนการส่งและหลังการรับสัญญาณจากเครือข่าย อีกส่วนนั้นจะกล่าวถึง ส่วนของการเชื่อมต่อระบบเครือข่ายตาม ดังสามารถดูได้จากรูปที่ 3.1



รูปที่ 3.1 ส่วนประกอบหลักของระบบการส่งเสียงผ่านเครือข่ายอีเทอร์เน็ท

3.1 ส่วนที่เป็นโมดูลก่อนการส่งและหลังการรับสัญญาณจากเครือข่าย

3.1.1 วงจรขยายสัญญาณในเครื่องมือวัด (Instrument Amplifier)



รูปที่ 3.2 วงจรขยายสัญญาณในเครื่องมือวัด

จากรูปที่ 3.2 จะสามารถหาค่าอัตราขยายได้จากสมการ

$$\frac{V_{out}}{V_2 - V_1} = \left(1 + \frac{2R_1}{R_{gain}}\right)\frac{R_3}{R_2}$$

โดยเมื่อกำหนดให้ค่าอัตราขยายมีค่าเท่ากับ 100 จะทำให้ได้ค่า R ดังนี้ คือ

หลังจากทำการต่อวงจรตามรูปที่ 3.2 โดยที่ใช้ค่าความต้านทานดังต่อไปนี้

R_1 = 39k Ω

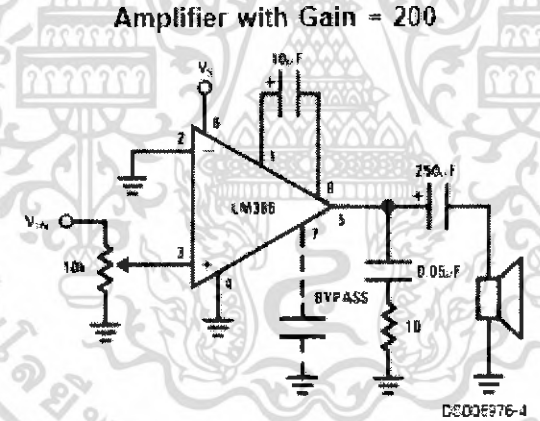
R_2 = 22k Ω

R_3 = 22k Ω

R_{gain} = 1k Ω

3.1.2 วงจรเพาเวอร์แอมปลิไฟเออร์ (Power Amplifier)

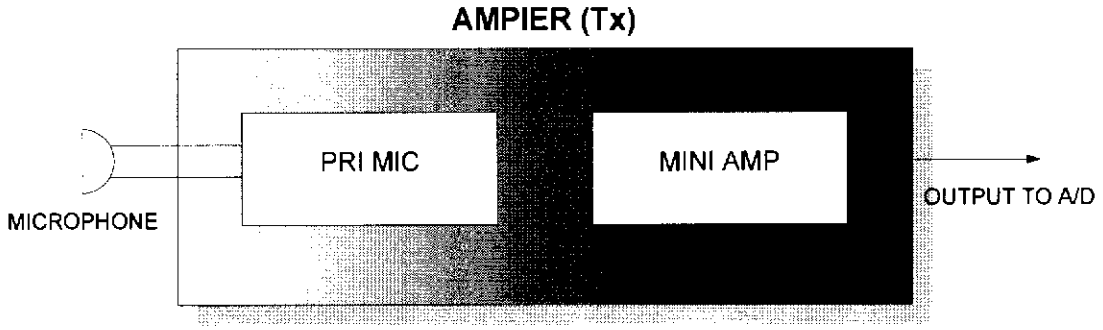
ในโครงงานนี้ได้เลือกใช้ไอซีเบอร์ LM386 เพื่อให้ได้กำลังขยายเท่ากับ 200 เท่า โดยทำการต่อวงจรดังรูปที่ 3.3



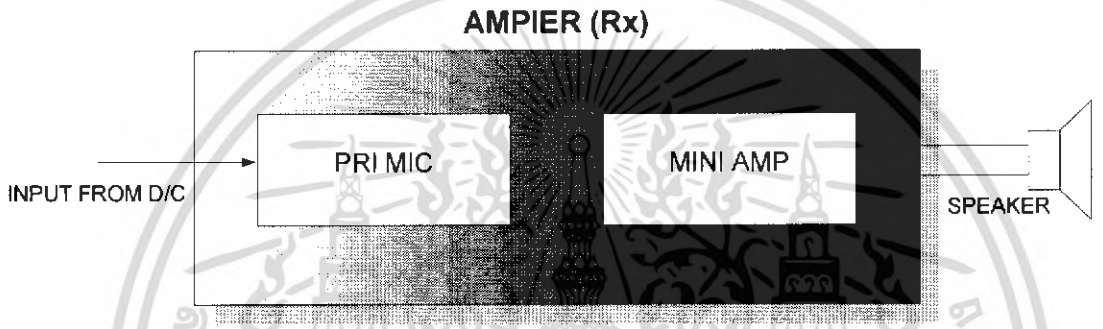
รูปที่ 3.3 วงจรเพาเวอร์แอมปลิไฟเออร์

3.1.3 วงจรขยายสัญญาณ (Amplifier circuits)

สำหรับในภาคเรียนที่ 2 นี้ในส่วนของวงจรแอมพลิไฟเออร์ได้ใช้ชุดของวงจรขยายสัญญาณประกอบไปด้วย วงจรปรีไมค์ และวงจรมินิแอมป์เชื่อมต่อกันซึ่งจะแบ่งเป็น 2 ชุด คือ ชุดที่อยู่กับด้านส่งซึ่งมีสัญญาณอินพุทเป็นไมค์คอนเดนเซอร์และสัญญาณเอาท์พุตต่อเข้ากับเอทูดี ดังแสดงในรูปที่ 3.4 และชุดที่อยู่กับด้านรับ สัญญาณอินพุทนั้นจะรับมาจากดีทูเอ ส่วนเอาท์พุทนั้นต่อเข้ากับลำโพง ดังแสดงในรูปที่ 3.5



รูปที่ 3.4 แสดงชุดวงจรขยายสัญญาณด้านส่งซึ่งมีสัญญาณอินพุทเป็นไมค์คอนเดนเซอร์และสัญญาณเอาท์พุทต่อเข้ากับเอทูดี

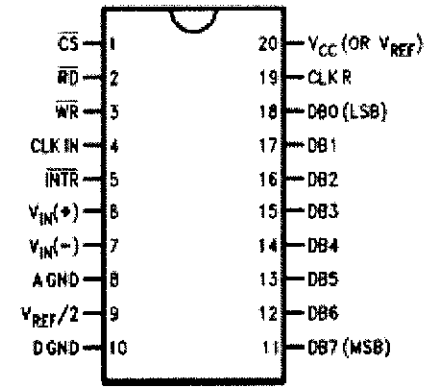


รูปที่ 3.5 แสดงชุดวงจรขยายสัญญาณด้านรับ สัญญาณอินพุทนั้นจะรับมาจากดีทูเอ ส่วนเอาท์พุทนั้นต่อเข้ากับลำโพง

3.1.3 วงจรแปลงสัญญาณจากอะนาล็อกเป็นสัญญาณดิจิตอล : เอทูดี (Analog to Digital Converter : A/D)

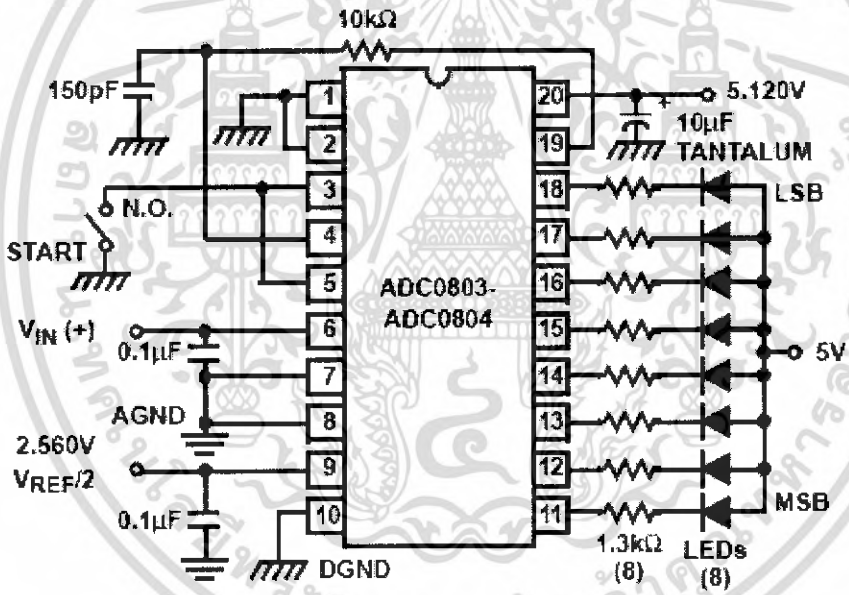
โดยทั่วไปแล้วเอทูดีจะแบ่งออกเป็นชนิดที่ให้เอาท์พุทออกมาเป็นเลขฐานสิบ และเป็นเลขฐานสอง เอทูดีที่ทำหน้าที่เปลี่ยนสัญญาณอะนาล็อกเป็นดิจิตอลที่มีเอาท์พุทเป็นเลขฐานสองมักจะใช้เป็นดิจิตอลโวลต์มิเตอร์และถูกใช้ใน Digital Panel Meter และ DMM คอนเวอร์เตอร์ที่เปลี่ยนสัญญาณอะนาล็อกเป็นดิจิตอลที่มีเอาท์พุทเป็นเลขฐานสองจะมีเอาท์พุทตั้งแต่ 4 ถึง 16 เอาท์พุท เอทูดีที่มีเอาท์พุทเป็นเลขฐานสอง จะเป็นอุปกรณ์อินพุทชนิดหนึ่งในระบบที่มีไมโครโปรเซสเซอร์เป็นเลขฐานในการควบคุม (Microprocessor - Base) เรียกว่าเอทูดีแบบ μP – Type

เอทูดีที่ใช้ในโครงงานนี้ คือเบอร์ ADC 0804 ซึ่งเป็นเอทูดีแบบ Successive –Approximation ขนาด 8 บิต มีความเร็วในการแปลงสัญญาณ 100 μS (Conversion Time = 100 μS) มีรายละเอียดการจัดวางขาของ ADC 0804 ดังรูปที่ 3.6



รูปที่ 3.6 การจัดวางขาของ ADC 0804

วิธีสตาร์ทเอทูดี ทำได้โดยส่งสัญญาณเข้าขา $\overline{WR}$ และ $\overline{CS}$ ด้วยศักดาต่ำ (Low) แล้วให้รออีกประมาณ 100 μS หลังจากนั้นเอทูดีจะส่งสัญญาณออกมาที่ขา $\overline{INTR}$ (คือสัญญาณที่บอกว่าแปลงสัญญาณเสร็จแล้ว) มีระดับสัญญาณเป็น Low ขานี้มักจะต่อกับขา $\overline{INT0}$, $\overline{INT1}$ ของซีพียู (CPU) เพื่อบอกให้ซีพียูทำขบวนการอินเตอร์รัพท์เพื่อนำข้อมูลที่แปลงแล้วไปเก็บ ในการอ่านข้อมูลไปเก็บ ซีพียูต้องส่งสัญญาณ $\overline{RD}$ มาเข้า $\overline{RD}$ ของเอทูดี (โดยขา $\overline{CS}$ ต้องเป็น Low อยู่ก่อนหน้านี้)



รูปที่ 3.7 วงจรทดสอบเอทูดี

การทดสอบเอทูดี ทำได้โดยป้อน (Vref/2) = 2.560 V คงที่

แล้วป้อนแรงดัน Vin ที่ขา Vin(+) มีค่า 0 ถึง 5 V

แล้วกดสวิตช์ Start แล้วปล่อยให้กลับมาอยู่ตำแหน่ง Open Circuit

ก็จะเห็น LED ติดหรือดับเปลี่ยนแปลงตามค่าอินพุทที่เข้ามา

โดยที่ผลที่จะได้เมื่อ ป้อน Vin(+) = 0 V จะได้ดิจิตอลเอาท์พุท = 0000 0000B

ป้อน Vin(-) = 5 V จะได้ดิจิตอลเอาท์พุท = 1111 1111B =(Full Scale)

การเชื่อมโยง ADC 0804 เข้ากับ MCS-51

การเชื่อมต่อ ADC 0804 เข้ากับ MCS-51 ทำได้ 2 รูปแบบ คือ

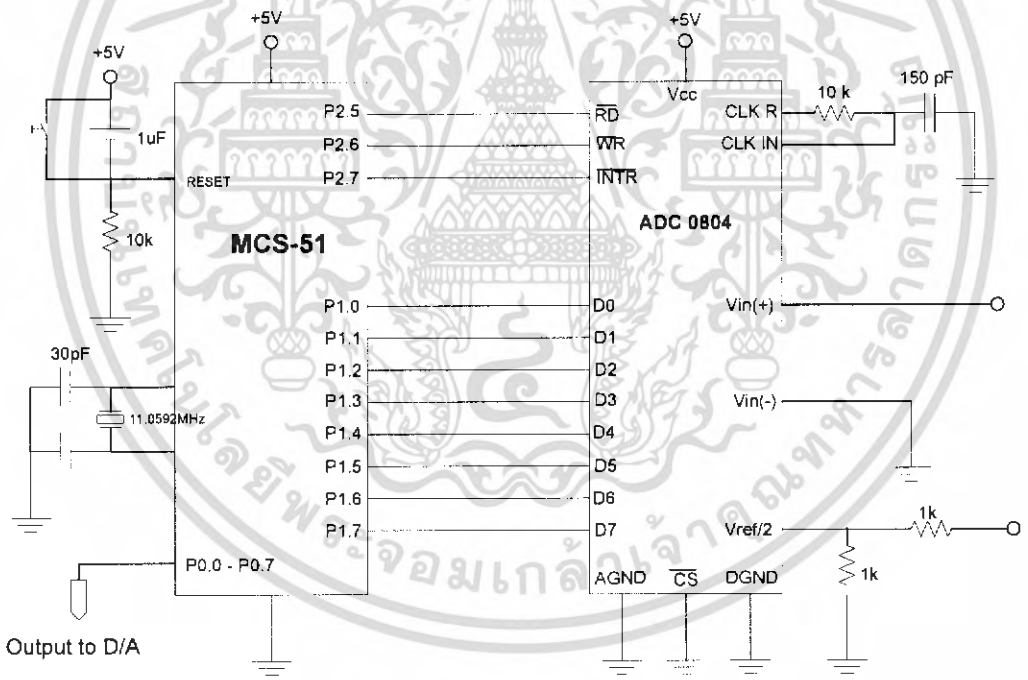
- ติดต่อแบบ I/O Port
- ติดต่อแบบ Memory Map

ในโครงงานนี้จะใช้การติดต่อแบบ I/O Port ซึ่งสามารถต่อ ADC 0804 เข้ากับ MCS-51 ผ่านทาง Port 0, Port 1, Port 2 , Port 3 เราสามารถใช้พอร์ทไหนเป็น Data Port ก็ได้ ส่วนที่เหลือเป็นสัญญาณควบคุม

การเชื่อมต่อ ADC 0804 เข้ากับ MCS-51 โดยกำหนดให้

- ขา DATA ต่อเข้ากับขา Port 1 (P1)
- ขา $\overline{RD}$ ต่อเข้ากับขา P 2.5
- ขา $\overline{WR}$ ต่อเข้ากับขา P 2.6
- ขา $\overline{INTR}$ ต่อเข้ากับขา P 2.7

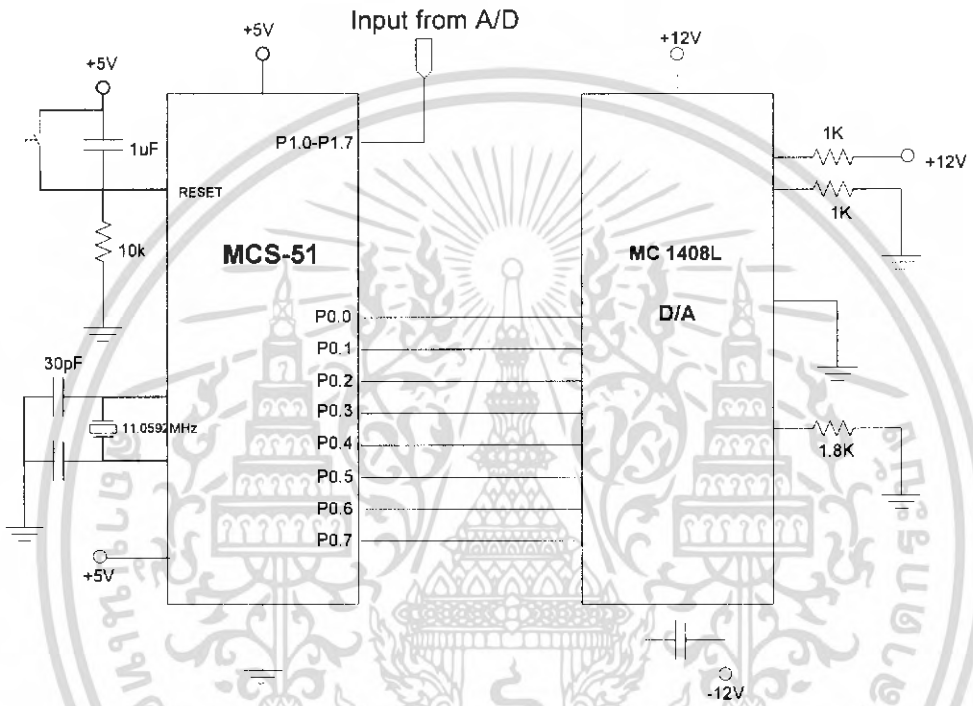
วิธีการต่อดังแสดงในรูป 3.8



รูปที่3.8 การต่อ ADC 0804 เข้ากับ MCS-51

3.1.4 วงจรแปลงสัญญาณจากดิจิตอลเป็นสัญญาณอะนาล็อก : ดีทูเอ (Digital to Analog Converter : D/A)

ในโครงงานได้เลือกใช้ดีทูเอเบอร์ MC1408 ซึ่งเป็นดีทูเอแบบ Monolithic and Hybrid D/A Converter ซึ่งเป็นแบบที่ใช้ค่าความต้านทานที่สร้างขึ้นมาพร้อมกันในขบวนการผลิตทางไอซี ทำให้ค่าของความต้านทานมีความแม่นยำมากทำให้ D/A Converter มีความแม่นยำ ซึ่งจำเป็นที่ต้องใช้แหล่งจ่ายไฟถึง 2 แหล่งโดยมีการต่อ MCS-51 กับ MC 1408 ดังแสดงในรูป 3.9 โดยที่อินพุทนั้นจะรับมาจากเอทูดีทางด้านพอร์ท 1 (ในที่นี้เป็นเพียงวงจรที่ใช้ทำการทดสอบไอซีเท่านั้น)



รูปที่ 3.9 การเชื่อมโยง MCS-51 กับ MC 1408

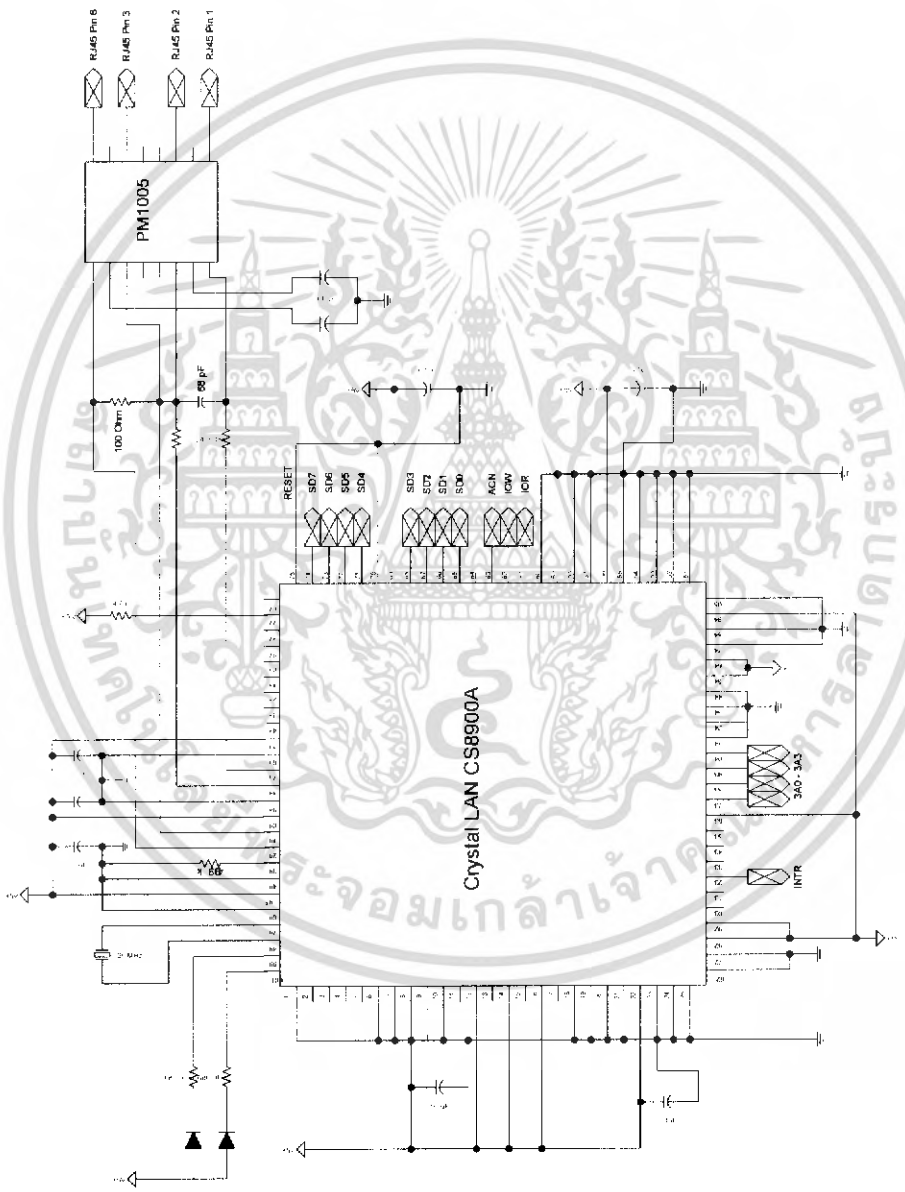
โดยส่วนของโปรแกรมที่ทำการเขียนเพื่อให้ไมโครคอนโทรลเลอร์ใช้ในการควบคุมการทำงานของเอทูดี รวมไปถึงการรับส่งข้อมูลแบบอนุกรมกับไมโครคอนโทรลเลอร์อีกตัวที่ทำหน้าที่ในการควบคุมการรับส่งข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์ นั้นใช้ภาษาแอสเซมบลี ซึ่งมีลำดับในการเขียนดังนี้

1. ทำการสตาร์ทเอทูดีโดยการป้อนพัลส์ให้กับขา $\overline{WR}$ สัญญาณที่แปลงได้นำไปเก็บไว้ในรีจีสเตอร์ R7
2. นำข้อมูลจากรีจีสเตอร์ R7 ทำการส่งข้อมูลแบบอนุกรมกับไมโครคอนโทรลเลอร์อีกตัวที่ทำหน้าที่ในการควบคุมการรับส่งข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์ โดยใช้การส่งข้อมูลอนุกรม ด้วยความเร็ว 9600 Buad
3. ในขณะเดียวกันก็สามารถทำการรับได้ด้วย (ดูจากโปรแกรมในภาคผนวก)

3.2 ส่วนของการเชื่อมต่อระบบเครือข่าย

3.2.1 ฮาร์ดแวร์ของระบบที่ใช้ในการรับ - ส่งข้อมูล ผ่านวงแลน

การเชื่อมต่อกับระบบเครือข่ายนั้น ในส่วนของฮาร์ดแวร์ของระบบควบคุมนั้นได้ใช้การควบคุมแบบฝังตัว (Embedded) โดยมีไมโครคอนโทรลเลอร์ทำหน้าที่ในการควบคุม I/O พอร์ต ของอีเทอร์เน็ทคอนโทรลเลอร์ในการรับส่งข้อมูลจากวงแลน ซึ่งอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย (Ethernet Controller) โดยภายในวงจรจะประกอบไปด้วยส่วนประกอบหลัก คือ ชิป (Chip)ควบคุมอีเทอร์เน็ท ในที่นี้จะทำการเลือกใช้ CS8900A-CQ , Transformer (PM1005) และ RJ-45 Connectorดังแสดงในรูปที่3.10



รูปที่ 3.10 แสดงวงจรส่วนเชื่อมต่อระบบเครือข่าย

วงจรควบคุมการเชื่อมต่อระบบเครือข่ายจะมี PIN OUT 18 ขา ดังรูปที่ 3.11

PIN 1	PIN 2	PIN 3	PIN 4	PIN 5	PIN 6	PIN 7	PIN 8	PIN 9
GND	VCC	INTR	SA0	SA1	SA2	SA3	$\overline{IOR}$	$\overline{IOW}$

PIN 10	PIN 11	PIN 12	PIN 13	PIN 14	PIN 15	PIN 16	PIN 17	PIN 18
$\overline{AEN}$	SD0	SD1	SD2	SD3	SD4	SD5	SD6	SD7

รูปที่ 3.11 แสดง PIN OUT ของวงจรควบคุมการเชื่อมต่อระบบเครือข่าย

VCC : แหล่งจ่ายไฟตรง +5 V

GND : กราวด์อ้างอิง 0 V

INTR : ใช้ในโหมดอินเตอร์รัปต์

SA0 – SA3 : Address Bus เชื่อมต่อกับ MCS-51

$\overline{IOR}$: I/O Port Read (Active low)

$\overline{IOW}$: I/O Port Write (Active low)

$\overline{AEN}$: Chip Select (Active low)

SD0 – SD7 : Data Bus เชื่อมต่อกับ MCS-51

จาก LED ในวงจรจะสว่างด้วยการจ่ายไฟแล้วเสียบสาย RJ-45 ที่ต่อในวง LAN เมื่อ

- Link LED (Green) จะกระพริบเมื่อมี Datagram ส่งออกจาก Ethernet Controller
- LAN LED (Yellow) จะกระพริบเมื่อมี Datagram เข้ามายัง Ethernet Controller

3.2.2 การติดต่อกับอุปกรณ์ต่างๆ

ในการใช้ไมโครคอนโทรลเลอร์ ในที่นี้จะใช้ MCS-51 เบอร์ AT89C52 เป็นไมโครคอนโทรลเลอร์ 8 บิตไปควบคุมอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายโดย SD0 – SD7 จะเป็นดาต้าบัส (Data Bus) เชื่อมต่อกับพอร์ท 0 (P0.0 - P0.7) , SA0 – SA3 จะเป็นแอดเดรส (Address Bus) เชื่อมต่อกับพอร์ท 2 (P2.0 – P2.3) , $\overline{IOW}$ ต่อเข้ากับ P1.0 , $\overline{IOR}$ ต่อเข้ากับ P1.1 , $\overline{AEN}$ จะต่อลงกราวด์ , ขา INTR จะปล่อยลอยเพราะไม่ได้ใช้อินเตอร์รัปต์ ส่วนของ P0.2 จะต่อกับ LED เพื่อแสดงสถานะของการส่งข้อมูล P0.3 จะต่อกับ LED เพื่อแสดงสถานะของการรับข้อมูล และต่อพอร์ทอนุกรมเข้ากับพอร์ทอนุกรมของไมโครคอนโทรลเลอร์อีกตัว

3.2.3 การเข้าถึงหน่วยความจำของระบบ

ในอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายจะมีชิปประมวลผลที่สำคัญคือ CS8900A-CQ โดยชิปนี้มีรูปแบบการทำงาน 3 โหมด คือ Memory Mode, I/O Mode และ DMA Mode แต่ในที่นี้จะใช้เพียง I/O Mode เพียงอย่างเดียว ใน I/O Mode จะประกอบด้วย I/O พอร์ท อยู่ 8 พอร์ท แต่ละพอร์ทจะมีความยาว 16 บิต แต่เนื่องจากไมโครคอนโทรลเลอร์ทำงานทีละ 8 บิต จึงจำเป็นต้องส่งข้อมูล 2 รอบ จึงจะเข้าถึง I/O พอร์ท ได้ คำสั่งต่างๆใน I/O Mode จะประกอบด้วย

- **Receive / Transmit Data (พอร์ต 0, พอร์ต 1)**
  ใช้ในการรับส่งข้อมูลจากอีเทอร์เน็ท ส่วนมากจะใช้พอร์ท 0
- **TxCMD (Transmit Command)**
  ใช้ในการออกคำสั่งให้อุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายเตรียมตัวส่งข้อมูล
- **TxLength (Transmit Length)**
  ใช้ในการระบุความยาวของข้อมูลที่จะส่งเป็นไบต์
- **Interrupt Status Queue**
  ใช้ในการอินเทอร์รัพต์
- **PacketPage Pointer**
  ใช้ในการระบุรีจีสเตอร์ภายในของ CS8900A-CQ สามารถหาดูได้จาก Datasheet
- **PacketPage Data (พอร์ท 0, พอร์ท 1)**
  ใช้ในการอ่านหรือเขียนรีจีสเตอร์ภายใน ก็ถูกระบุโดย PacketPage Pointer

ในการระบุคำสั่งต่างๆของ I/O พอร์ต ทำได้โดยการจ่ายไฟไปยัง Address Bus (SA0 – SA4) ดังตารางที่ 3.1

ตารางที่ 3.1 I/O พอร์ทของชิป CS8900A-CQ

Offset	Type	Description
0000h	Read/Write	Receive/Transmit Data (Port 0)
0002h	Read/Write	Receive/Transmit Data (Port 1)
0004h	Write-only	TxCMD (Transmit Command)
0006h	Write-only	TxLength(Transmit Length)
0008h	Read/Write	Interrupt Status Queue
000Ah	Read/Write	PacketPage Pointer
000Ch	Read/Write	PacketPage Data (Port 0)
000Eh	Read/Write	PacketPage Data (Port 1)

ตัวอย่างการเข้าถึงรีจีสเตอร์ภายในของ I/O Mode

ถ้าต้องการเข้าถึงรีจีสเตอร์ Receiver Event (Rx Event) ที่มี Address อยู่ที่ 0x0124 สามารถทำได้โดย

1. Write 0x24 (Least Significant 8 bit) to PacketPage Pointer นั่นคือ Address Bus = 0xa และ Data Bus = 0x24
2. Write 0x21 (Most Significant 8 bit) to PacketPage Pointer + 1 นั่นคือ Address Bus = 0xa + 1 = 0xb และ Data Bus = 0x01
3. เมื่อระบุรีจีสเตอร์แล้วก็จะสามารถ Read หรือ Write รีจีสเตอร์ได้โดยใช้ PacketPage Data พอร์ท 0 ทำได้โดย

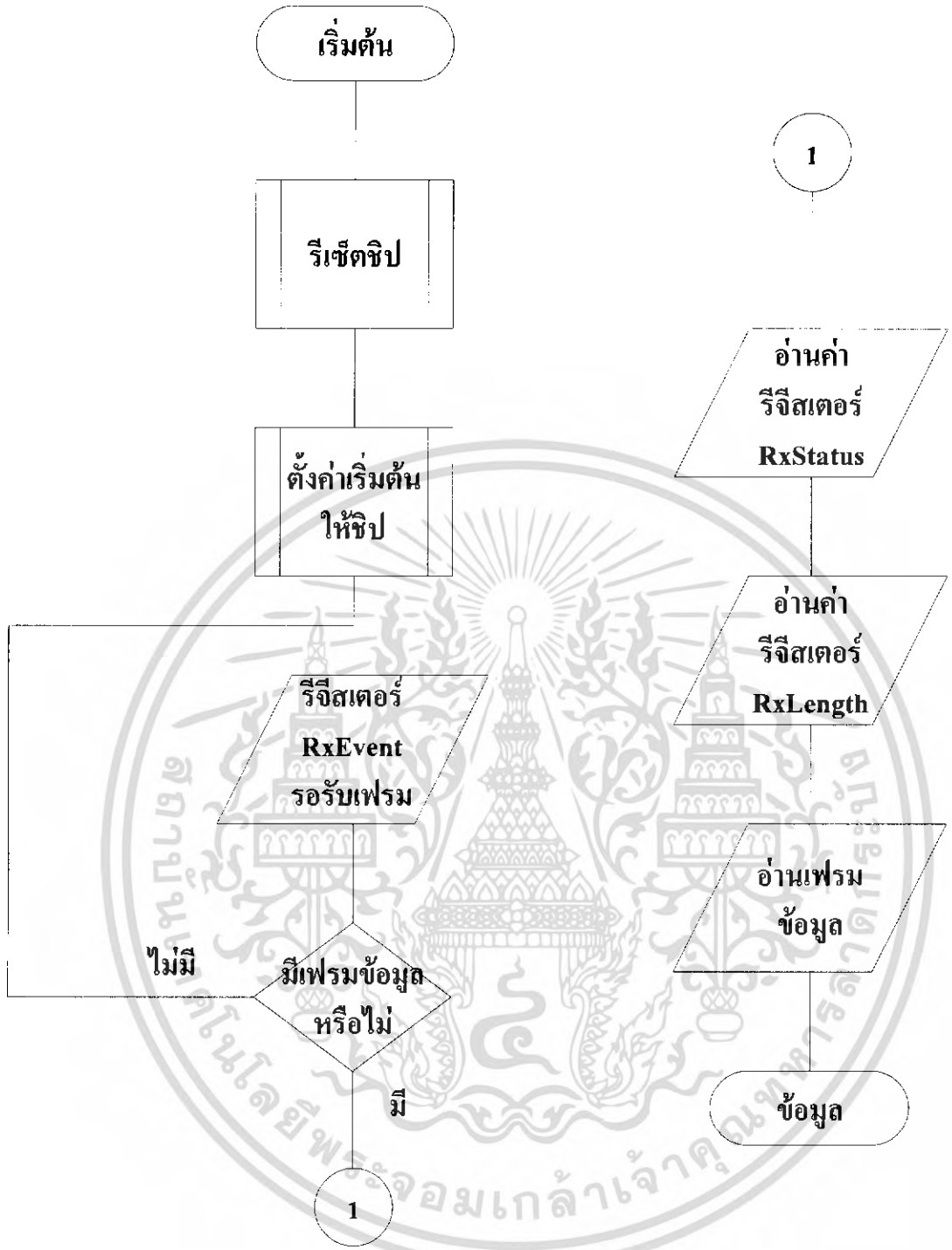
- Read / Write to PacketPage Data จะได้ Least Significant 8 บิต นั่นคือ Address Bus = 0xc และ Data Bus = Data Least Significant 8 บิต ที่ต้องการจะ Read หรือ Write
- Read / Write to PacketPage Data จะได้ Most Significant 8 บิต นั่นคือ Address Bus = 0xc + 1 = 0xd และ Data Bus = Data Most Significant 8 บิต ที่ต้องการจะ Read หรือ Write

3.2.4 กระบวนการส่งและรับข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์

ในกระบวนการส่งข้อมูลในอีเทอร์เน็ทคอนโทรลเลอร์ เริ่มต้นด้วยการรีเซ็ตเพื่อลบค่าเก่าออกจากอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายก่อน จากนั้นทำการตั้งค่าเริ่มต้นของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย โดยใช้รีจีสเตอร์ RxCTL, LineCTL และค่า MAC Address ต่อมาตามด้วยการเขียนคำสั่งต้องการที่จะส่ง (TxCMD) และ เขียนความยาวของข้อมูล (TxLength) จากนั้นตรวจดูว่าบัสที่ต้องการส่งว่าว่างหรือไม่ โดยตรวจดูจากรีจีสเตอร์ BusST เมื่อ Bus ว่างก็จะสามารถส่งข้อมูลโดยเข้า I/O พอร์ท คือ Transmit Data (พอร์ท 0) ที่ Address 0x00 ดังในโฟว์ชาร์ตในการส่งข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์ ดังรูปที่ 3.12

ส่วนในกระบวนการรับข้อมูลใน Ethernet Controller เริ่มต้นทำการรีเซ็ตและตั้งค่าเริ่มต้นของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย และใช้รีจีสเตอร์ RxEvent รอรับ เฟรมเมื่อมีเฟรมเข้ามาให้ทำการอ่านค่า RxStatus และค่า RxLength จาก Receive Data พอร์ท 0 แต่มีข้อต้องระวังคือ RxStatus และ RxLength จะต้องอ่านจาก Most Significant bit ก่อนแล้วจึงค่อยอ่านจาก Least Significant bit คือ Set Address = 0x01 แล้วจึง Set Address = 0x00 จากนั้นก็ทำการอ่านค่าตามปกติจาก I/O Receive Data พอร์ท 0 คือ อ่าน Address = 0x00 แล้วจึงอ่านค่า 0x01 วนไปเรื่อยจนข้อมูลที่ส่งมาครบหมด ดังในโฟว์ชาร์ตในการรับข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์ ดังรูปที่ 3.13

รูปที่ 3.12 แสดงโฟว์ชาร์ตในการส่งข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์



รูปที่ 3.13 แสดงโฟว์ชาร์ตในการรับข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์

บทที่ 4
ผลการทดลอง

ในบทที่ 4 นี้จะกล่าวถึงผลการทดลองที่ได้จากการทดสอบฮาร์ดแวร์ต่างๆในระบบ รวมไปถึงผลการทดลองในการส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ท เพื่อให้ง่ายต่อการพิจารณาและการทดสอบส่วนต่างๆของระบบ จึงได้ผลการทดลองออกเป็นส่วนๆ แล้วจึงจะกล่าวถึงผลการทดลองของระบบโดยรวม ซึ่งผลการทดลองที่ได้มีดังนี้

4.1 ผลการทดลองวงจรภาคขยายอินสตรูเมนท์แอมพลิไฟเออร์และเพาเวอร์แอมพลิไฟเออร์

4.1.1 ทดสอบวงจรวงจรอินสตรูเมนท์แอมปลิไฟเออร์

หลังจากทำการต่อวงจรตามรูปที่ 3.2 โดยที่ใช้ค่าความต้านทานดังต่อไปนี้

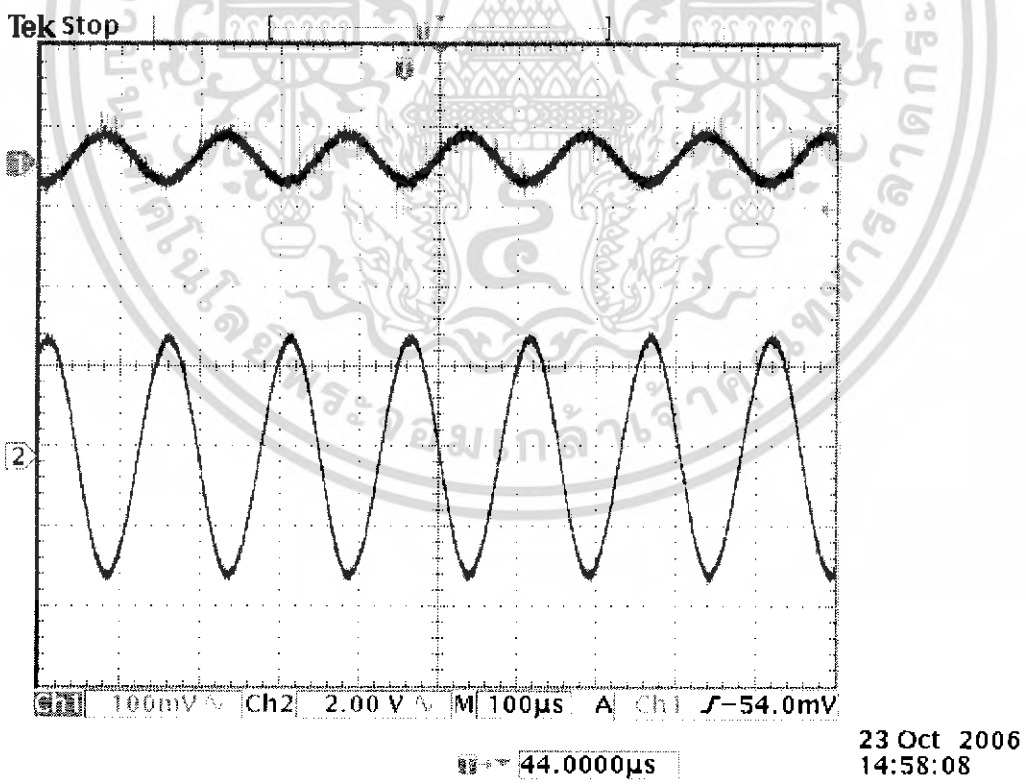
R_1 = 39kΩ

R_2 = 22kΩ

R_3 = 22kΩ

R_{gain} = 1kΩ

จะทำให้ได้สัญญาณเอาท์พุทดังรูปที่ 4.1



รูปที่ 4.1 แสดงการเปรียบเทียบสัญญาณอินพุทที่มีขนาด 75 มิลลิโวลท์ (แชนแนล 1) เมื่อผ่านวงจรอินสตูเมนท์แอมพลิไฟเออร์ ได้สัญญาณเอาท์พุทขนาด 6 โวลท์ (แชนแนล 2)

จากรูปที่ 4.1 เมื่อทำการหาค่าอัตราขยายระหว่างสัญญาณเอาท์พุทและสัญญาณอินพุทคือ

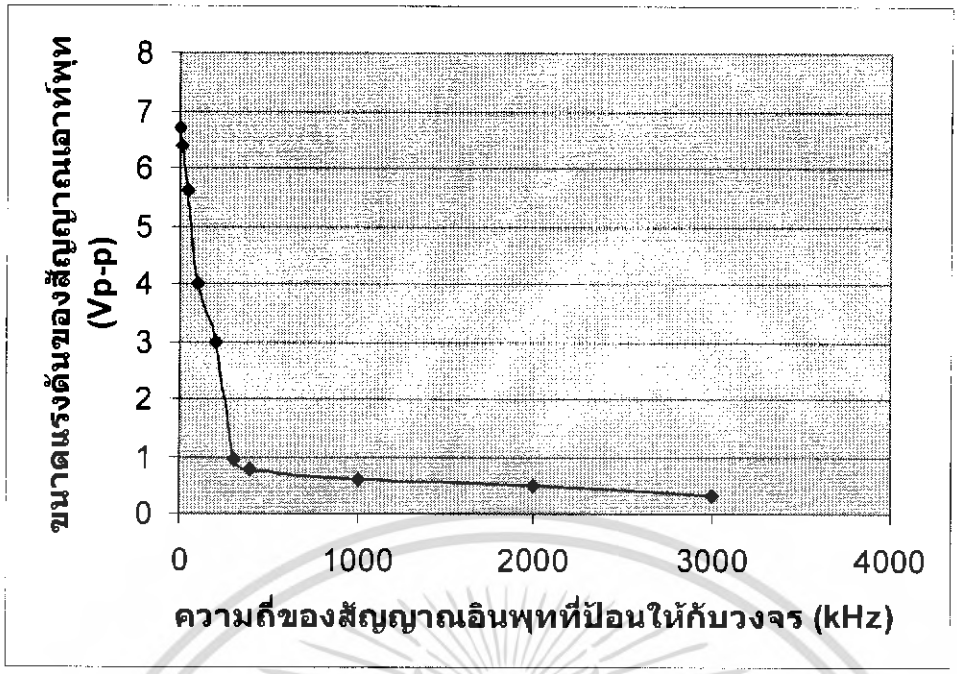
$$\text{อัตราการขยาย} = \frac{V_o}{V_{in}} = \frac{6}{75\times10^{-3}} = 80$$

และเมื่อทำการเปลี่ยนค่าความถี่ของสัญญาณอินพุทให้มีค่ามากขึ้น พบว่า ค่าความถี่ดังกล่าวมีผลต่อขนาดของสัญญาณเอาท์พุททำให้มีขนาดลดลง นั่นก็หมายความว่า ค่าความถี่ของสัญญาณอินพุทมีผลต่อค่าอัตราขยายด้วย ซึ่งสามารถดูได้จากผลการทดลองในตารางที่ 4.1 โดยมีค่าแรงดันของสัญญาณอินพุทเท่ากับ 360 มิลลิโวลท์พีคทูพีค (peak to peak)

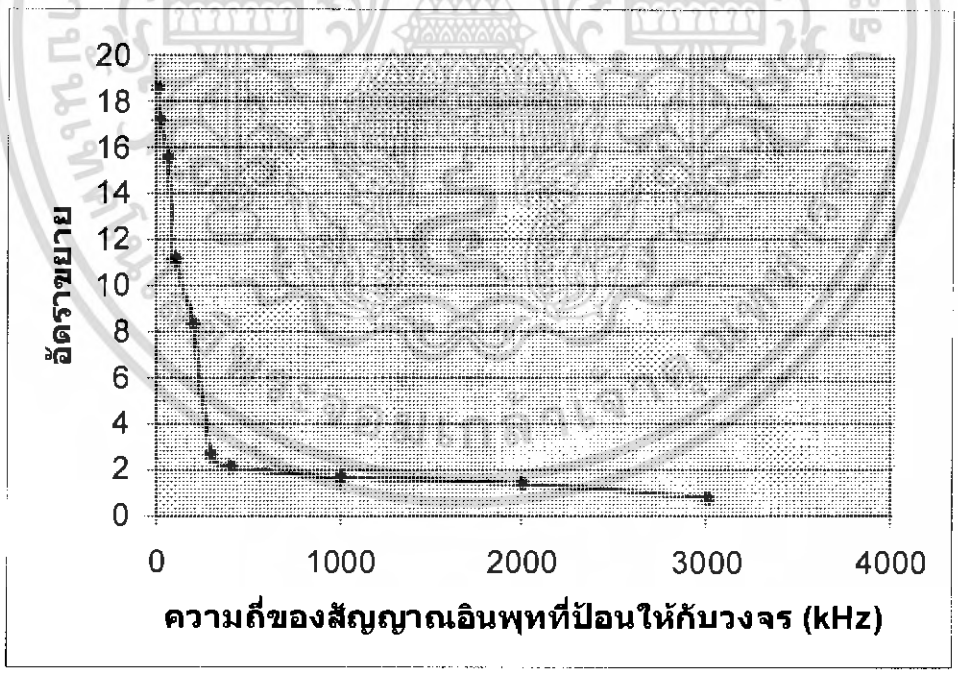
ตารางที่ 4.1 แสดงผลของความถี่ของสัญญาณอินพุทที่มีต่อขนาดของสัญญาณเอาท์พุทและอัตราขยาย

ความถี่ของสัญญาณอินพุทที่ป้อนให้กับวงจร (Hz)	ขนาดแรงดันของสัญญาณเอาท์พุท V_o (V_{p-p})	อัตราขยาย $= \frac{V_o}{V_{in}} = \frac{V_o}{360\times10^{-3}}$
1k	6.7	18.61
10 k	6.4	17.18
50 k	5.6	15.56
100 k	4.0	11.11
200 k	3.0	8.33
300 k	960 m	2.67
400 k	760 m	2.11
1 M	600 m	1.67
2 M	500 m	1.39
3 M	300 m	0.83

จากตารางที่ 4.1 เมื่อเรานำค่าที่ได้มาพล็อตเป็นกราฟโดยมีแกนตั้งเป็นค่าขนาดแรงดันของสัญญาณเอาท์พุท และแกนนอนเป็นค่าความถี่ของสัญญาณอินพุท ดังรูปที่ 4.2 และพล็อตเป็นกราฟโดยมีแกนตั้งเป็นค่าอัตราขยาย และแกนนอนเป็นค่าความถี่ของสัญญาณอินพุทดังรูปที่ 4.3 จะได้กราฟคาแรคเตอริสติค (characteristic) ของวงจรอินสตรูเมนท์แอมปลิไฟเออร์



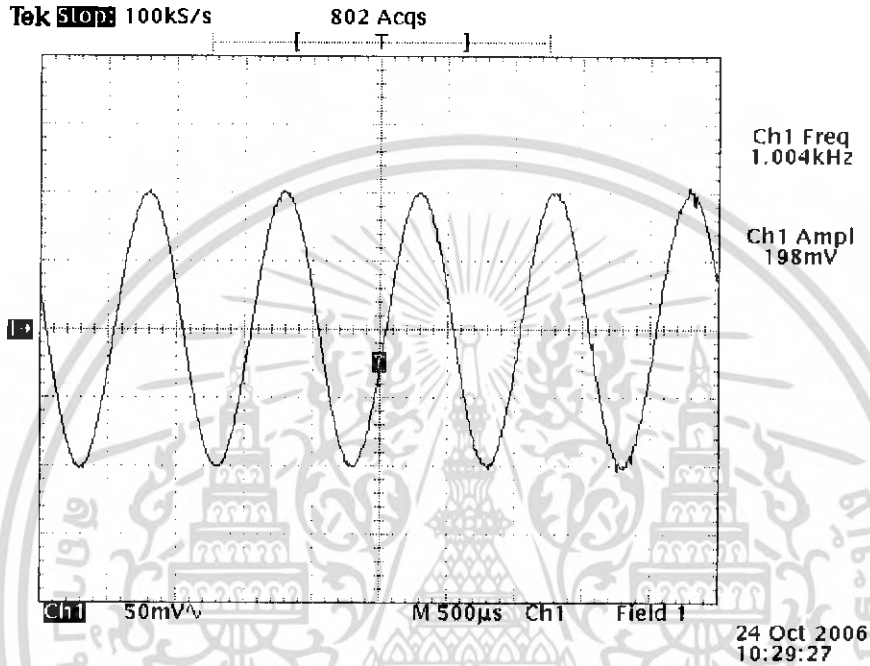
รูปที่ 4.2 กราฟคาแรคเตอริสติคของวงจรอินสตรูเมนท์แอมปลิไฟเออร์แกนตั้งเป็นค่าขนาดแรงดันของสัญญาณเอาท์พุท และแกนนอนเป็นค่าความถี่ของสัญญาณอินพุท



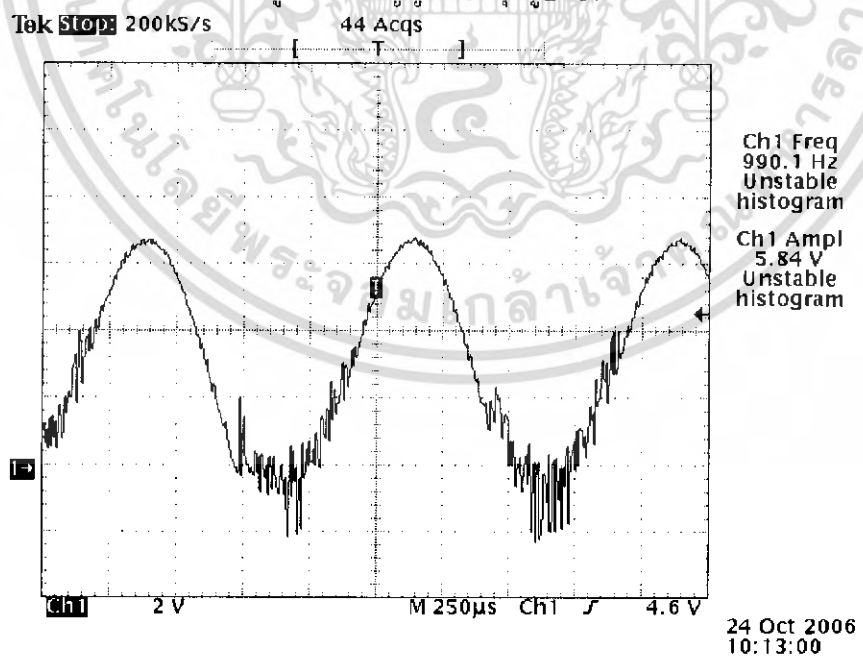
รูปที่ 4.3 กราฟคาแรคเตอริสติคของวงจรอินสตรูเมนท์แอมปลิไฟเออร์โดยมีแกนตั้งเป็นค่าอัตราขยาย และแกนนอนเป็นค่าความถี่ของสัญญาณอินพุท

4.1.2 ทดสอบวงจรเพาเวอร์แอมพลิไฟเออร์

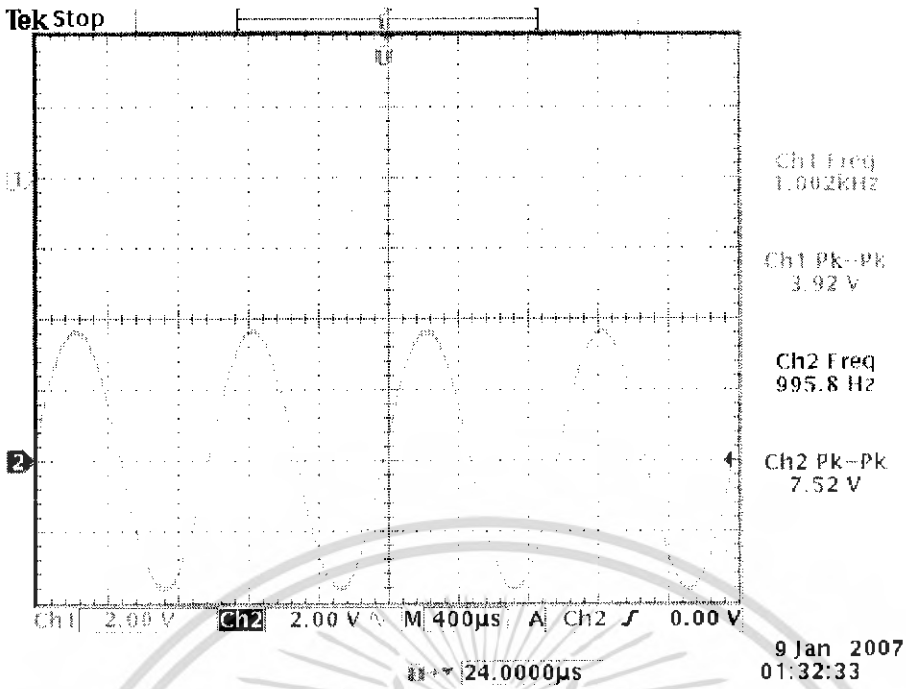
สำหรับวงจรเพาเวอร์แอมพลิไฟเออร์ เมื่อทดลองต่อวงจรและป้อนสัญญาณไฟเข้าไป โดยป้อนแรงดันทั้งขาบวกและลบที่แรงดัน 5 V และทำการป้อนสัญญาณรูปไซน์เพื่อวัดสัญญาณอินพุทและเอาท์พุท โดยปรับค่าความต้านทานปรับค่าได้ เพื่อทำการขยายขนาดสัญญาณ จะได้สัญญาณดังรูปที่4.4 และ 4.5 ซึ่งจะได้กำลังขยายมีค่าประมาณ 10 เท่า และได้ทำการทดสอบวงจรมินิแอมพลิไฟเออร์เปรียบเทียบสัญญาณอินพุทและเอาท์พุทได้ดังรูปที่ 4.6



รูปที่ 4.4 สัญญาณอินพุทรูปไซน์



รูปที่4.5 แสดงสัญญาณเอาท์พุทที่ได้เมื่อป้อนอินพุทเป็นสัญญาณรูปไซน์



รูปที่4.6 แสดงสัญญาณอินพุทเทียบกับสัญญาณเอาท์พุทที่ได้เมื่อป้อนอินพุทเป็นสัญญาณรูปไซน์ โดยรูปบนแสดงสัญญาณอินพุตขนาด 3.92 V_{p_p} ความถี่ประมาณ 1 kHz รูปล่างแสดงสัญญาณ เอาท์พุตขนาด 7.52 V_{p_p} ความถี่ประมาณ 1 kHz ซึ่งมีกำลังขยายประมาณ 2 เท่า

4.2 ผลการทดลองวงจรเอทูดีและดีทูเอ

4.2.1 การทดสอบเอทูดี

หลังจากที่ได้เลือกใช้เอทูดีบอร์ ADC 0804 ซึ่งเอทูดีเบอร์นี้มีวิธีการทดสอบว่าสามารถใช้งานได้เป็นปกติหรือไม่ทำได้ โดยป้อน (Vref/2) = 2.560 V คงที่

แล้วป้อนแรงดัน Vin ที่ขา Vin(+) มีค่า 0 ถึง 5 V

แล้วกดสวิตช์ Start แล้วปล่อยให้กลับมาอยู่ตำแหน่ง Open Circuit

ก็จะเห็น LED ติดหรือดับเปลี่ยนแปลงตามค่าอินพุทที่เข้ามา

ซึ่งเมื่อทำการปรับค่าระดับแรงดันที่ป้อนเข้าไป ทำให้ได้ผลการทดลองของเอทูดีดังตารางที่ 4.2

ตารางที่ 4.2 ตารางแสดงผลการทดสอบเอทูดีเบอร์ ADC0804 เมื่อป้อนค่าแรงดันที่ระดับต่างๆ

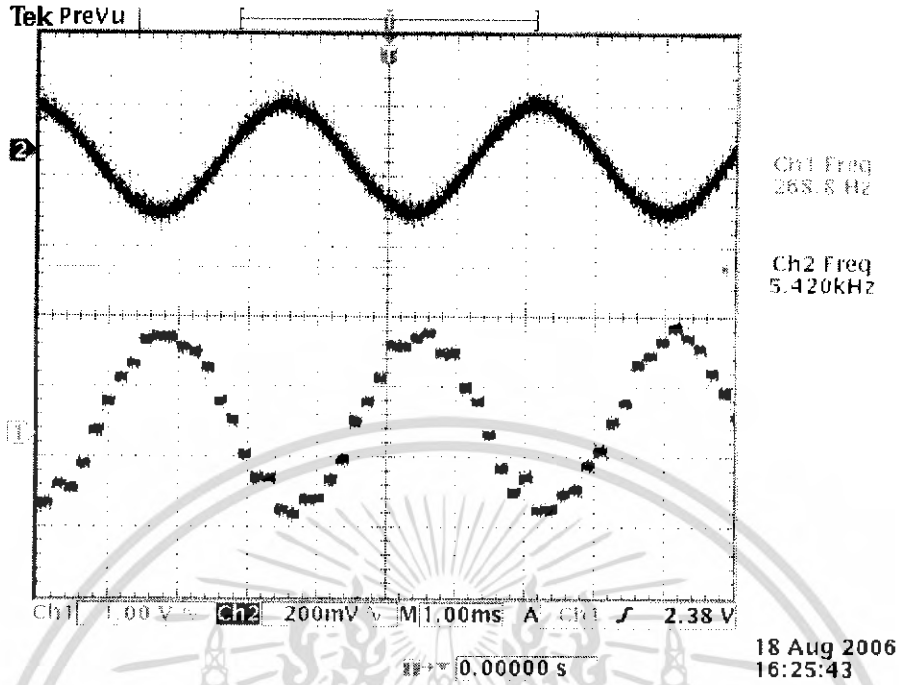
ค่าVin	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0
0.5	0	0	0	1	1	1	0	0
1.0	0	0	1	1	0	1	0	0
1.5	0	1	0	0	1	1	1	0
2.0	0	1	1	0	1	0	0	0
2.5	1	0	0	0	0	0	0	0
3.0	1	0	0	1	1	0	1	0
3.5	1	0	1	1	0	0	0	0
4.0	1	1	0	0	1	0	0	0
4.5	1	1	1	0	0	1	0	0
5.0	1	1	1	1	1	1	1	1

4.2.2 ผลการทดลองเมื่อทำการการเชื่อมโยง ADC 0804 เข้ากับ MCS-51และดีทูเอ MC1408

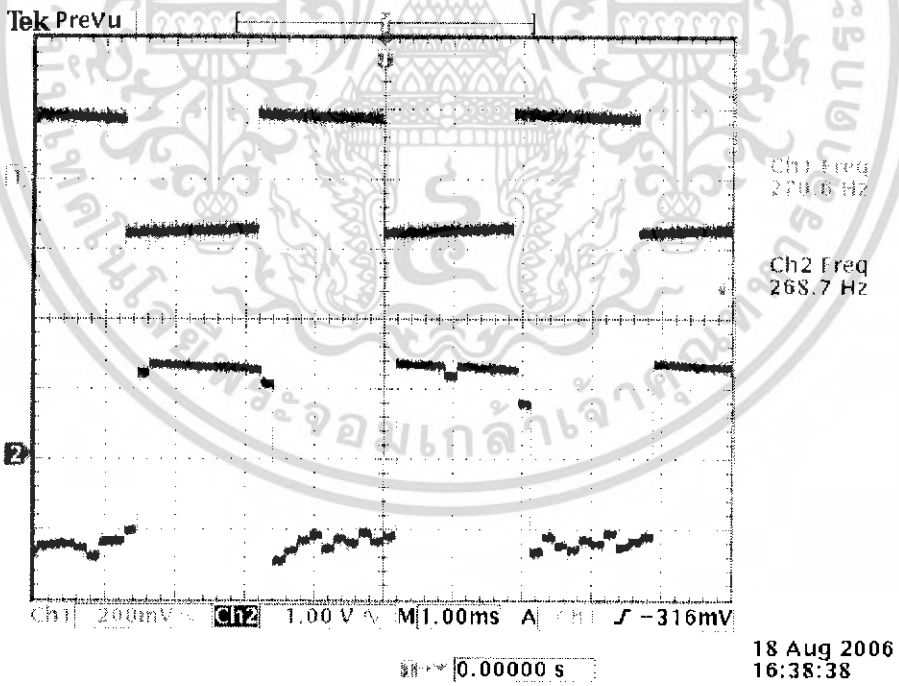
การเชื่อมต่อ ADC 0804 เข้ากับ MCS-51 โดยกำหนดให้

- ขา DATA in ต่อเข้ากับขา Port 1 (P1)
- ขา $\overline{RD}$ ต่อเข้ากับขา P 2.5
- ขา $\overline{WR}$ ต่อเข้ากับขา P 2.6
- ขา $\overline{INTR}$ ต่อเข้ากับขา P 2.7
- ขา DATAout ต่อเข้ากับขา Port 0 (P0)

โดยที่ อินพุทนั้นให้เป็นสัญญาณในรูปต่างๆ และทำการเขียนโปรแกรมด้วยภาษาแอสเซมบลีเพื่อใช้ในการควบคุมเอทูดีโดยไมโครคอนโทรลเลอร์ แล้วเอาท์พุทของไมโครคอลโทรลเลอร์มาต่อเป็นอินพุทของดีทูเอทางพอร์ท 0 ซึ่งได้ผลการทดลองออกมาเป็นสัญญาณอะนาล็อกเปรียบเทียบกับอินพุทที่ใส่เข้าไปด้านเอทูดี ดังรูปที่ 4.7 และ 4.8



รูปที่ 4.7 เปรียบเทียบระหว่างอินพุทและเอาท์พุทเมื่อป้อนอินพุทเป็นสัญญาณรูปไซน์

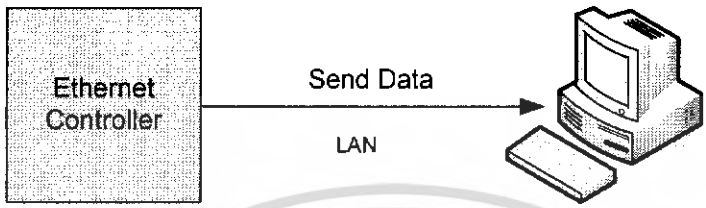


รูปที่ 4.8 เปรียบเทียบระหว่างอินพุทและเอาท์พุทเมื่อป้อนอินพุทเป็นสัญญาณสี่เหลี่ยม

4.3 การทดสอบการรับ - ส่งขอมูลของอีเทอร์เน็ทคอนโทรลเลอร์

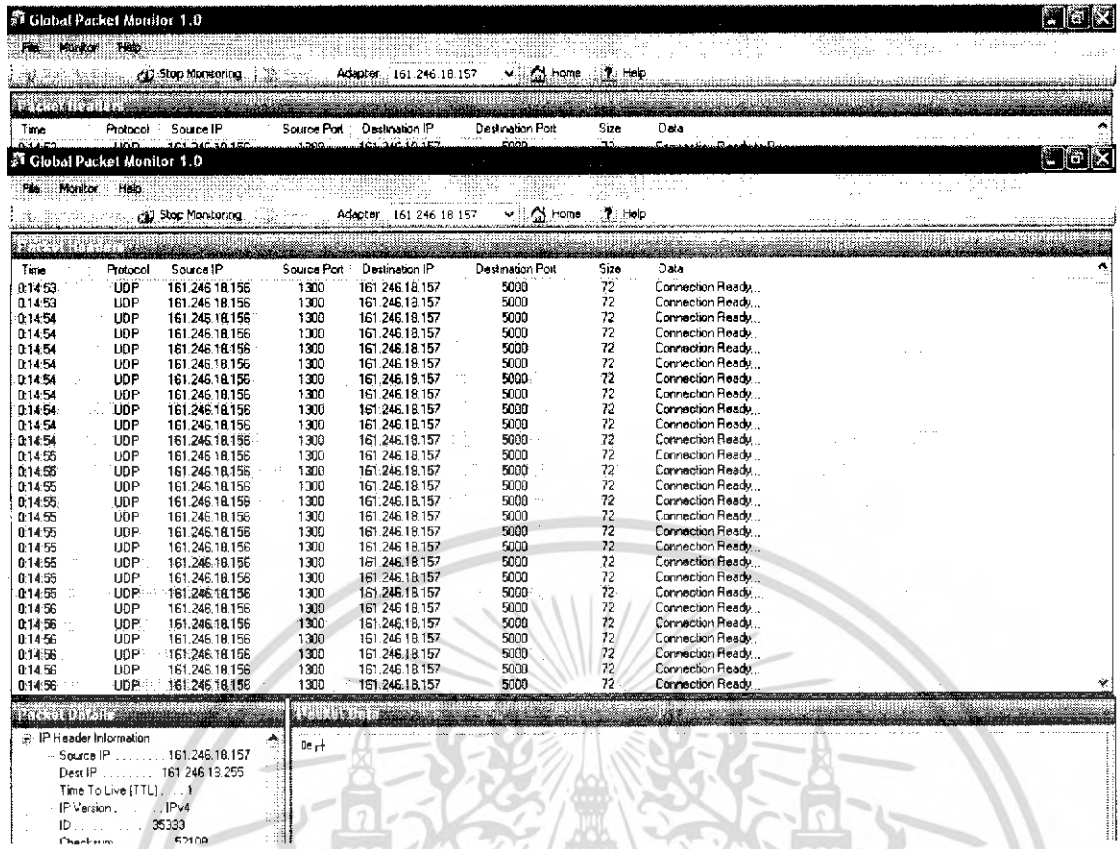
ในที่นี้ได้ใช้การเขียนโปรแกรมเพื่อใช้ในการทดสอบการทำงาน รับ-ส่ง ข้อมูลของอีเทอร์เน็ทคอนโทรลเลอร์ผ่านเครือข่ายอีเทอร์เน็ท ซึ่งในการทดสอบนั้นได้แบ่ง 4 ขั้นตอน ดังต่อไปนี้

4.3.1.การทดสอบการส่งข้อมูลจากอีเทอร์เน็ทคอนโทรเลอร์ไปยังเครื่องคอมพิวเตอร์



รูปที่ 4.9 แสดงการทดสอบการส่งข้อมูลจากอีเทอร์เน็ทคอนโทรเลอร์ไปยังเครื่องคอมพิวเตอร์

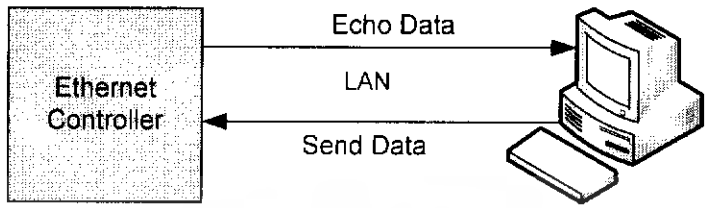
จากรูปที่ 4.9 ที่แสดงการส่งข้อมูลจากอีเทอร์เน็ทคอนโทรเลอร์ไปยังเครื่องคอมพิวเตอร์โดยผ่านสายแลน ซึ่งได้ทำการแสดงสถานะการส่งข้อมูลด้วยโปรแกรม Global Packet Monitor 1.0 เพื่อการแสดงสถานะพร้อมรับข้อมูลด้วยคำว่า Connection Ready แสดงในฟิลด์ Data ของ UDP Packet ดังแสดงในรูปที่ 4.10



รูปที่4.10 แสดงสถานะการส่งข้อมูลด้วยโปรแกรม Global Packet Monitor 1.0 เพื่อการแสดงสถานะพร้อมรับข้อมูลด้วยคำว่า Connection Ready แสดงในฟิลด์ Data ของ UDP Packet

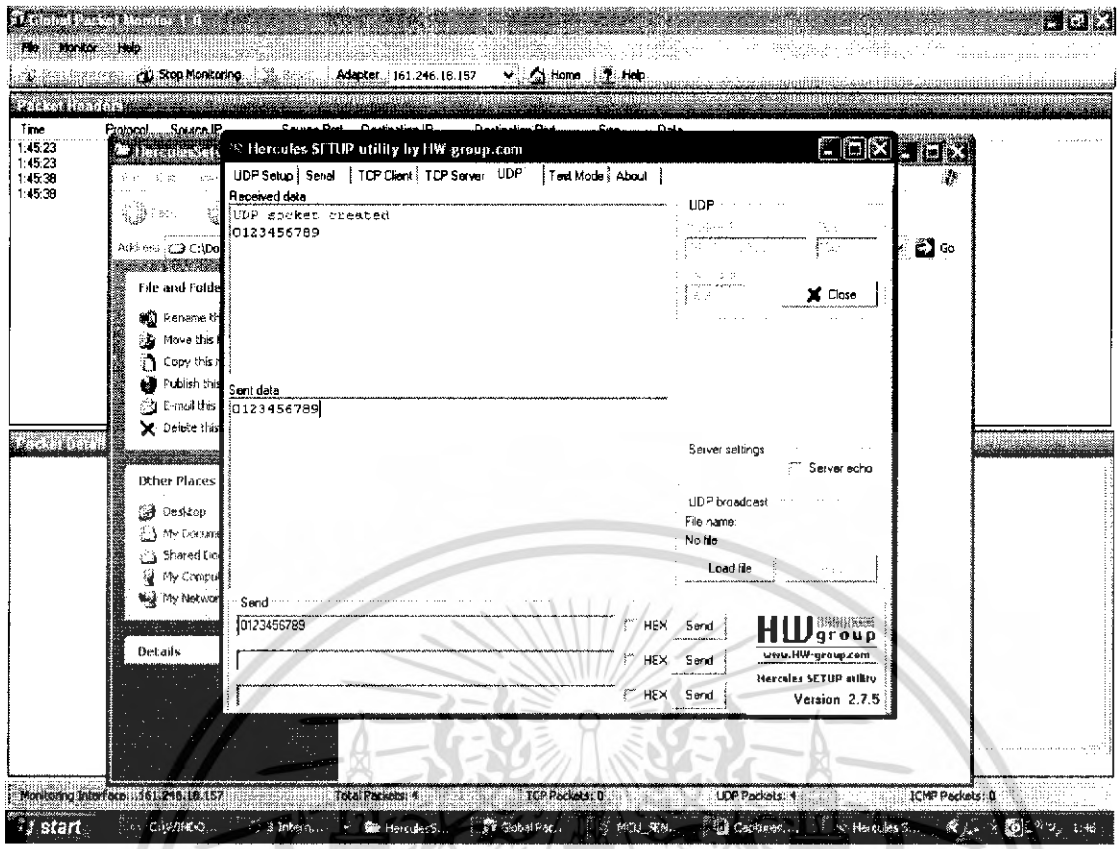
4.3.2. การส่งข้อมูลจากเครื่องคอมพิวเตอร์ไปยังอีเทอร์เน็ทคอนโทรลเลอร์

โดยทำการส่งข้อมูลจากเครื่องคอมพิวเตอร์ไปยังอีเทอร์เน็ทคอนโทรลเลอร์ แล้วให้อีเทอร์เน็ทคอนโทรลเลอร์ส่งข้อมูลที่รับได้กลับมายังเครื่องคอมพิวเตอร์ ดังแสดงในรูปที่4.11

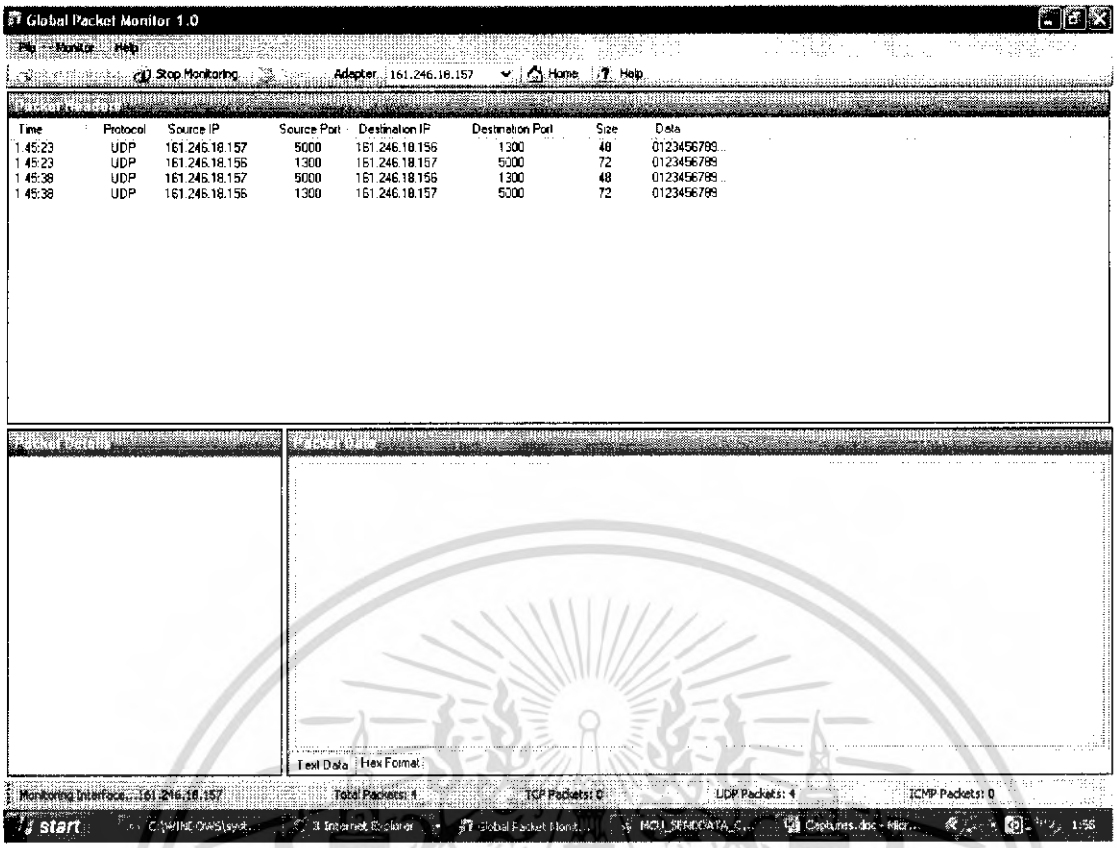


รูปที่4.11 แสดงการส่งข้อมูลจากเครื่องคอมพิวเตอร์ไปยังอีเทอร์เน็ทคอนโทรลเลอร์

ในการแสดงสถานการณ์ส่งข้อมูลที่รับได้ของอีเทอร์เน็ทคอนโทรลเลอร์นั้น ที่เครื่องคอมพิวเตอร์จะใช้โปรแกรมแสดงการทดสอบการทำงานในโหมด echo ของอีเทอร์เน็ทคอนโทรลเลอร์ด้วยโปรแกรม Hercules โดยการส่งข้อมูลขนาด 10 ไบต์ เพื่อให้มีการตอบกลับถึงข้อมูลที่ได้รับไป ผลปรากฏว่าอีเทอร์เน็ทคอนโทรลเลอร์สามารถทำงานได้เป็นอย่างดี ดังแสดงในรูปที่4.12 และแสดงการทดสอบการทำงานในโหมด echo ของอีเทอร์เน็ทคอนโทรลเลอร์ ด้วยโปรแกรม Global Packet Monitor 1.0 โดยการส่งข้อมูลขนาด 10 ไบต์ เพื่อให้มีการตอบกลับถึงข้อมูลที่ได้รับไป ผลปรากฏว่าอีเทอร์เน็ทคอนโทรลเลอร์สามารถทำงานได้เป็นอย่างดีเช่นกัน ดังแสดงในรูปที่

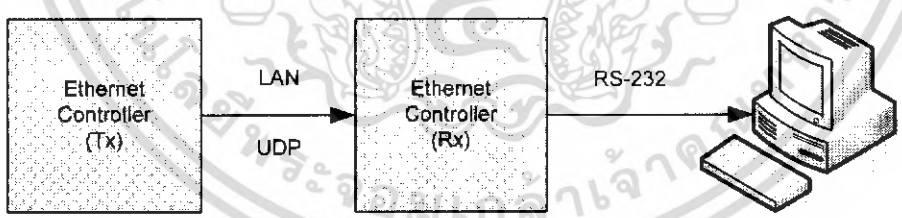


รูปที่4.12 แสดงการทดสอบการทำงานในโหมด echo ของอีเทอร์เน็ทโทรลเลอร์ ด้วยโปรแกรม Hercules



รูปที่ 4.13 แสดงสถานะการส่งข้อมูลจากเครื่องคอมพิวเตอร์ไปยังอีเทอร์เน็ทคอนโทรลเลอร์แล้วให้อีเทอร์เน็ทคอนโทรลเลอร์ส่งข้อมูลที่รับได้กลับมายังเครื่องคอมพิวเตอร์ใช้โปรแกรม Global Packet Monitor 1.0

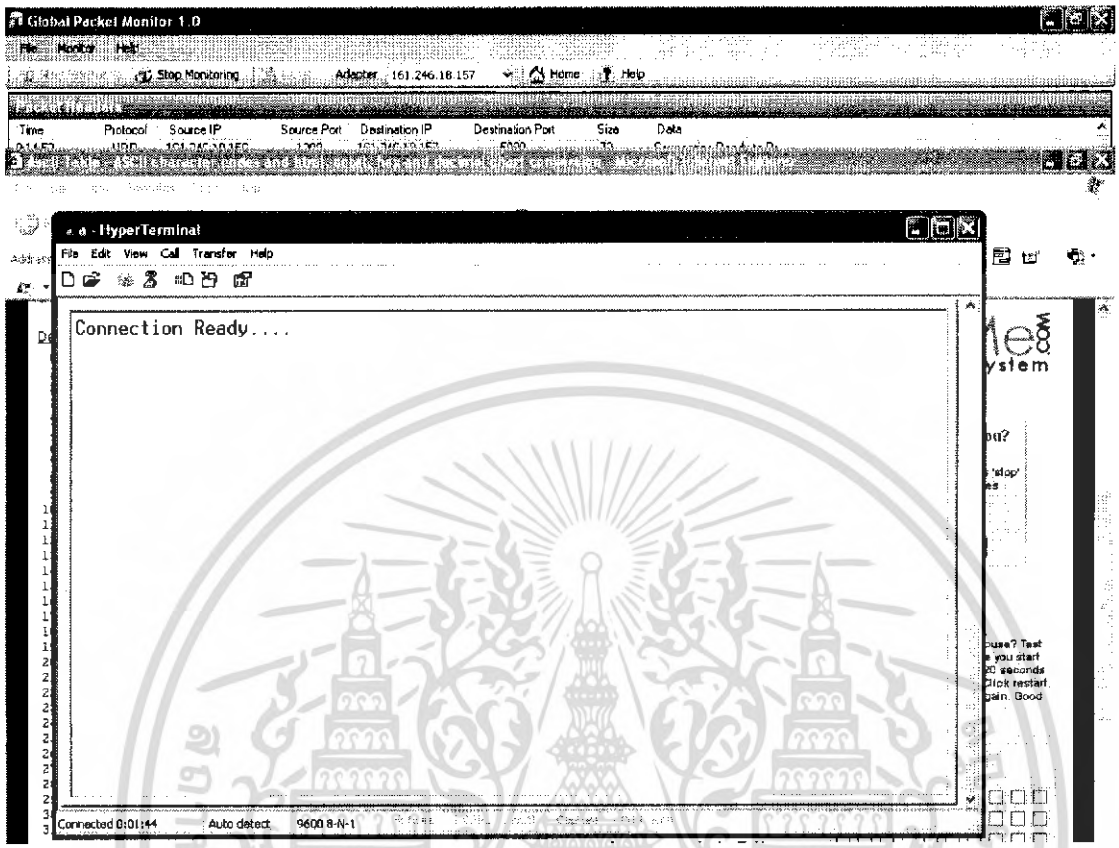
4.3.3.การทดสอบการส่งข้อมูลจากอีเทอร์เน็ทคอนโทรลเลอร์จากตัวส่งไปยังตัวรับแล้วแสดงผลที่เครื่องคอมพิวเตอร์



รูปที่4.14 แสดงการเชื่อมต่อการทดสอบการส่งข้อมูลจากอีเทอร์เน็ทคอนโทรลเลอร์จากตัวส่งไปยังตัวรับแล้วแสดงผลที่เครื่องคอมพิวเตอร์

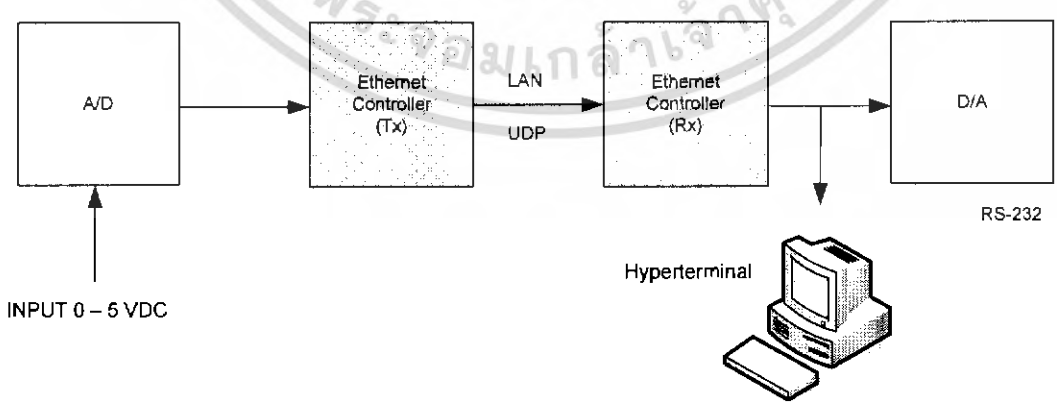
จากรูปที่4.14 เป็นการเชื่อมต่อการทดสอบการส่งข้อมูลจากอีเทอร์เน็ทคอนโทรลเลอร์จากตัวส่งไปยังตัวรับแล้วแสดงผลที่เครื่องคอมพิวเตอร์โดยระหว่างอีเทอร์เน็ทคอนโทรลเลอร์นั้นเชื่อมต่อกันโดยผ่านสายแลน และระหว่างอีเทอร์เน็ทคอนโทรลเลอร์มาที่คอมเชื่อมต่อโดยใช้พอร์ทอนุกรม RS-232

แล้วทำการแสดงผลผ่านหน้าจอไฮเปอร์เทอร์มินอล (HyperTerminal) ซึ่งในการทดสอบนี้ได้ใช้คำว่า Connection Ready.... ดังแสดงในรูปที่4.15



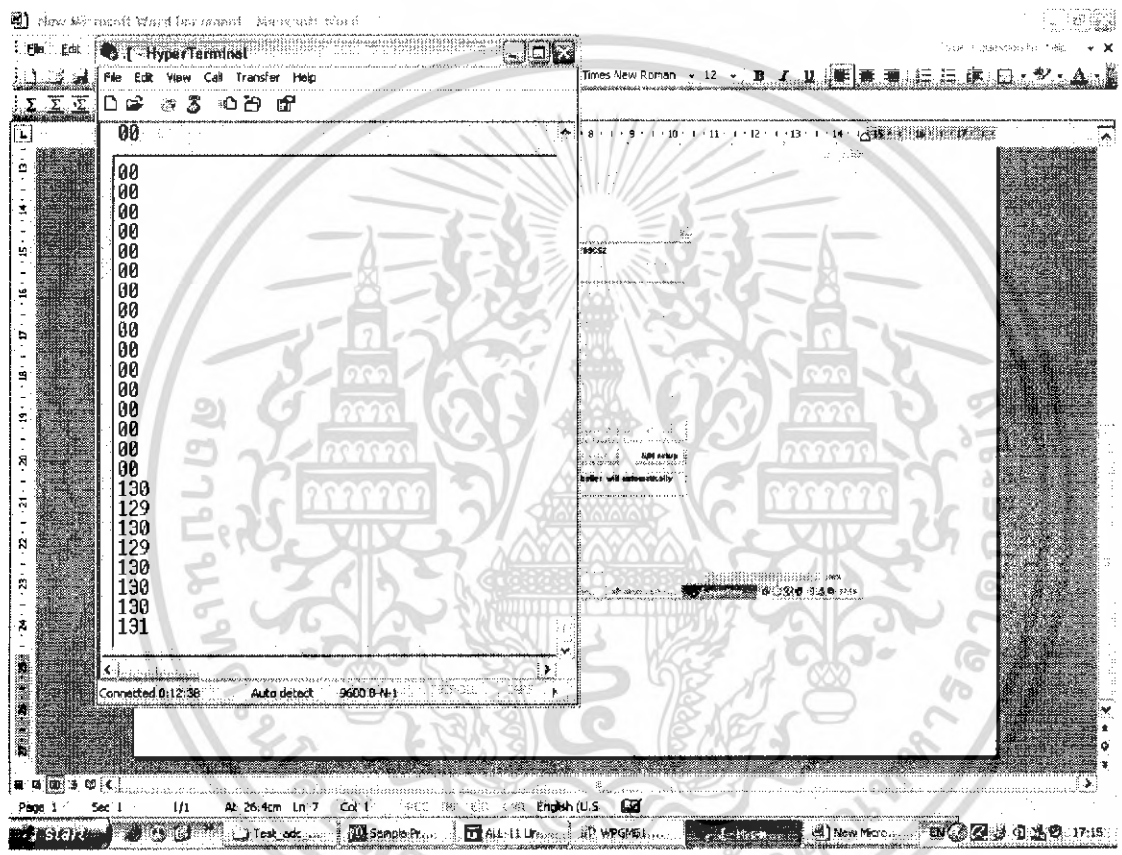
รูปที่4.15 แสดงการทดสอบด้วยโปรแกรม HyperTerminal ด้วยการส่งค่าด้วย UDP จากตัวส่งมายังตัวรับ และตัวรับทำการแปลงข้อมูลเพื่อส่งออกมาทางพอร์ตอนุกรม ด้วยการกำหนดค่าบอดเรทเป็น 9600

4.3.4.การทดสอบการส่งและรับข้อมูลระหว่างอีเทอร์เน็ทคอนโทรลเลอร์โดยใช้ข้อมูลจากเอทูดี

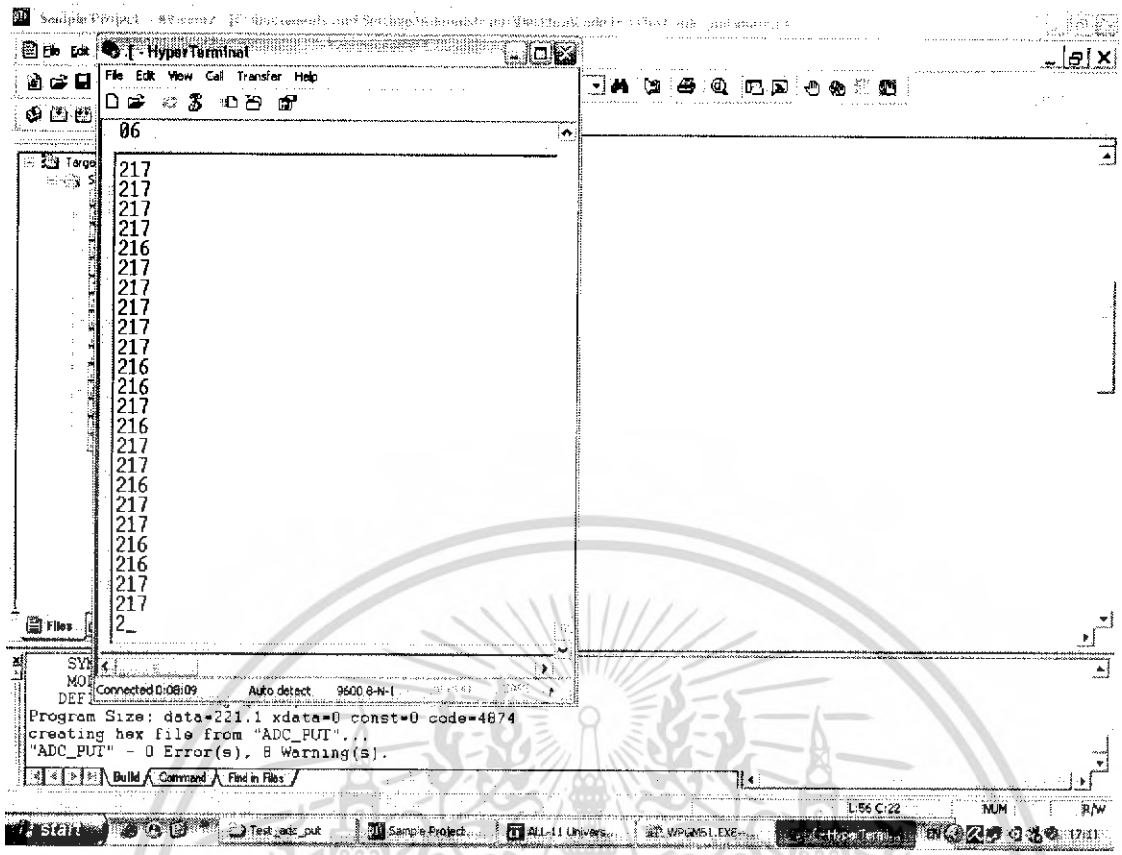


รูปที่4.16แสดงการทดสอบการส่งและรับข้อมูลระหว่างอีเทอร์เน็ทคอนโทรลเลอร์โดยใช้ข้อมูลจากเอทูดี

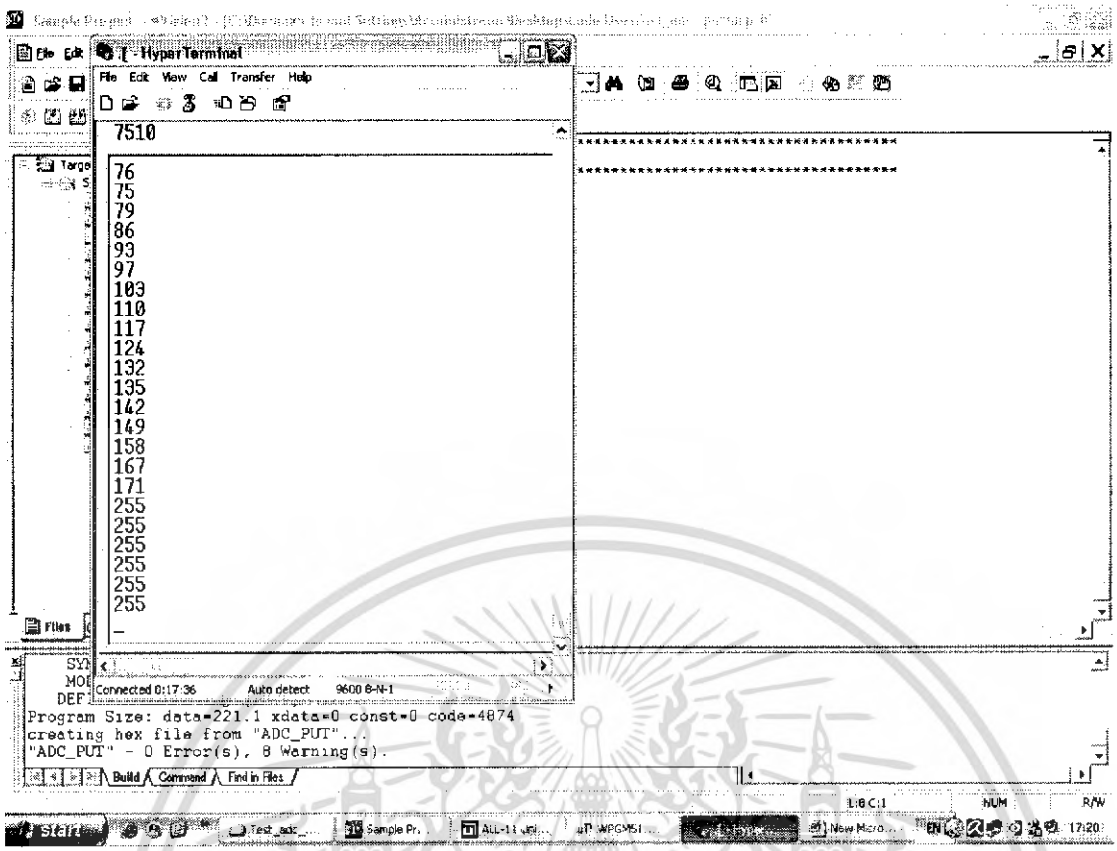
จากรูปที่ 4.16 ได้ทำการป้อนอินพุทเป็นแรงดันไฟตรง 0 – 5 โวลต์ให้กับเอทูดี เอทูดีก็แปลงสัญญาณให้อยู่ในรูปของสัญญาณดิจิตอลแล้ว จากนั้นอีเทอร์เน็ทคอนโทรลเลอร์จะทำหน้าที่ในการจัดการกับข้อมูลดิจิตอลดังกล่าวให้อยู่ในรูปของแพ็คเก็ทเพื่อที่จะส่งผ่านเครือข่ายแลนโดยโปรโตคอลยูดีพี ไปยังด้านรับที่มีอีเทอร์เน็ทคอนโทรลเลอร์ทำหน้าที่ในการตรวจสอบแพ็คเก็ทที่ส่งมารวมถึงทำการตรวจสอบค่าแมคแอดเดรสว่าตรงกับที่ร้องขอมาหรือไม่ จากนั้นอีเทอร์เน็ทคอนโทรลเลอร์ทางด้านรับก็จะส่งข้อมูลที่สามรถรับได้ไปยังดีทูเอให้ทำการแปลงสัญญาณให้กลับมาอยู่ในรูปของสัญญาณอะนาล็อก แล้วส่งต่อผ่านไปยังเครื่องคอมพิวเตอร์ผ่านสาย RS-232 แล้วแสดงผลผ่านหน้าจอไฮเปอร์เทอร์มินอลดังแสดงในรูปที่4.17,4.18 และ 4.19



รูปที่4.17 แสดงหน้าจอไฮเปอร์เทอร์มินอลที่ด้านรับเมื่อป้อนอินพุทเป็นแรงดันไฟตรง 0 - 5 โวลต์



รูปที่4.18 แสดงหน้าจอไฮเปอร์เทอร์มินอลที่ด้านรับเมื่อป้อนอินพุทเป็นแรงดันไฟตรง 0 - 5 โวลต์



รูปที่4.19 แสดงหน้าจอไฮเปอร์เทอร์มินอลที่ด้านรับเมื่อป้อนอินพุทเป็นแรงดันไฟตรง 0 - 5 โวลต์

บทที่ 5
บทวิจารณ์และบทสรุป

หลังจากที่ได้ทำการทดลองมาแล้ว สามารถสรุปสิ่งที่ได้จากโครงงานในส่วนของผลการทดลองและแนวทางในการดำเนินการต่อไปได้ดังนี้

5.1 การทำงานของระบบ

1. ขั้นตอนในการส่งสัญญาณเสียงผ่านเครือข่ายอีเทอร์เน็ทนั้นประกอบไปด้วย การนำสัญญาณเสียงพูดของคนเราซึ่งเป็นสัญญาณอะนาล็อกมาแปลงให้อยู่ในรูปของสัญญาณดิจิตอลเพื่อใช้กับไมโครคอนโทรลเลอร์ในการควบคุมการทำให้สัญญาณอยู่ในรูปแบบที่เหมาะสมที่จะส่งผ่านเครือข่ายอีเทอร์เน็ท ซึ่งทางด้านรับเมื่อได้รับสัญญาณที่อยู่ในรูปสัญญาณดิจิตอลก็จะนำมาแปลงกลับให้เป็นสัญญาณอะนาล็อกได้แก่สัญญาณเสียงกลับคืนมานั่นเอง โดยโครงงานนี้จะทำการรับส่งแบบฟูลดูเพล็กซ์ คือ ด้านหนึ่งสามารถเป็นได้ทั้งผู้รับและผู้ส่ง ตอบโต้สื่อสารกับฝ่ายตรงข้ามได้

2. เสียงพูดของคนเราโดยปกตินั้นจะมีช่วงความถี่อยู่ระหว่าง 30 - 3000 Hz เพราะฉะนั้นในการออกแบบระบบจึงควรที่จะสามารถรองรับความถี่ในช่วงนี้ได้ดี ซึ่งในโครงงานนี้ยังมีส่วนบกพร่องเล็กน้อยเนื่องจากส่วนของความถี่ที่รับได้แล้วทำให้แปลงสัญญาณกลับออกมาได้ดีนั้นอยู่ในช่วงประมาณ 200 – 300 kHz ซึ่งในความจริงเมื่อความถี่มากกว่านี้ก็ต้องสามารถแปลงสัญญาณได้ดีด้วย

5.2 ปัญหาและแนวทางการแก้ไข

1. ผลในการทดลองในส่วนของระบบรวมนั้นยังไม่สามารถที่จะส่งสัญญาณเสียงได้อาจจะเนื่องมาจากช่วงของการเปลี่ยนแปลงระดับแรงดันของวงจรไมค์อาจมีช่วงน้อยเกินไปเมื่อเทียบกับการเปลี่ยนแปลงระดับการเปลี่ยนระดับของเอทูดีที่มีค่าตั้งแต่ 0 – 255 ซึ่งอาจทำการแก้ไขได้โดย

- เพิ่มวงจรแอมพลิไฟเออร์เพื่อปรับช่วงของไมค์ให้อยู่ในย่านที่เหมาะสม
- เปลี่ยนเอทูดีที่มีจำนวนบิตที่แทนแต่ละระดับมากกว่านี้
- เพิ่ม Memory เพื่อเก็บ Transfer ของเสียงเพราะโดยปกติแล้วความเร็วในการส่งสัญญาณเสียงของมาตรฐาน VOIP (Voice Over IP) จะอยู่ที่ 64 kbps ซึ่งจะทำการบีบอัดสัญญาณเสียงให้เป็น 10 kbps ก่อนเพื่อลดแบนด์วิดท์ของสัญญาณ
- เพิ่มอัลกอริทึ่มในการจัดการกับเสียงที่มีคุณภาพกว่านี้ เช่น DSP

2. อาจพัฒนาให้เทียบเท่ามาตรฐานของเทคโนโลยี VOIP ซึ่งสามารถรับส่งภาพและเสียงผ่านเครือข่ายอินเตอร์เน็ท ให้มีความใกล้เคียงมากที่สุด

หนังสืออ้างอิง

[1] รศ.สมยศ จุณณะปิยะ, **การประยุกต์ใช้งานไมโครคอนโทรลเลอร์**, ภาควิชาวิศวกรรมโทรคมนาคม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

[2] บัณฑิต จามรภูติ, **คู่มือการใช้งาน Protel 99**, บัณฑิตเพรส:เชียงใหม่; 2544, 290 หน้า

[3] นิรุธ อำนวยสิลป์,**Network and Protocals Programming using C/C++**,จ.เจริญการพิมพ์: กรุงเทพ;210 หน้า

ภาคผนวก

Source Code ของการทดสอบการส่งและรับข้อมูลระหว่างอีเทอร์เน็ตคอนโทรลเลอร์โดยใช้ข้อมูลจากเอทูดีด้านตัวส่ง(TX)

Main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "cs8900.h"
#include "packets.h"
#include "icmp.h"
#include "arp.h"
#include "udp.h"
#include "adc.h"
#include "utils.h"
unsigned char code my_ipaddr[4] = {161,246,18,156};
unsigned char code my_macaddr[6] = {0x00,0x04,0xe2,0x7a,0x75,0x71};
unsigned char code my_portH = 0x05;
unsigned char code my_portL = 0x14;

unsigned char code des_ipaddr[4] = {161,246,18,157};
unsigned char code des_macaddr[6] = {0x00,0x02,0x3f,0xbc,0xcb,0x36};
unsigned char code des_portH = 0x13;
unsigned char code des_portL = 0x88;

unsigned char idata RxBuffer[160];
unsigned char idata adc_buf[22];

sbit p2_4 = P2^4;
sbit p2_5 = P2^5;
void uart_init(void)
{
      SCON  = 0x50;                 /* SCON: mode 1, 8-bit UART,
enable rcvr      */
      TMOD |= 0x20;                 /* TMOD: timer 1, mode 2, 8-bit
reload           */
      TH1   = 0xFA;                 /* TH1:  reload value for 9600
baud             */
      TR1   = 1;                    /* TR1:  timer 1 run
*/
      TI    = 1;                    /* TI:   set TI to send first
char of UART     */
      PCON |= 0x80;
}

void main()
{
            unsigned char checkbit;
            unsigned char i;
            static unsigned char tmp_buf[3];
            unsigned char adc;

            uart_init();
            init_cs8900();
            printf("start tx\n");

while(1)
{
            for(i=0;i<22;i++)
```

```
                {
                        adc_buf[i] = adc_read();
                        itoa(adc_buf[i],tmp_buf,10);
                        //printf("%d",adc_buf[i]);
                        //printf(",");
                        putchar(tmp_buf[0]);
                        putchar(tmp_buf[1]);
                        putchar(tmp_buf[2]);
                        printf("\n");
                }

                tx_adc();

}

while(1);

        //CheckPacket();
/*              for(i=0;i<22;i++)
                adc_buf[i] = adc_read();
                tx_adc();*/

        /*while(1){

                tx_icmp_req();

                checkbit = 0;
                while(checkbit == 0)
                {
                        rxpacket();
                        // For recieve ARP Request and send ARP reply
                        if((RxBuffer[13]==0x08)
                        &&(RxBuffer[14]==0x06)
                        &&(RxBuffer[39]==my_ipaodr[0])
                        &&(RxBuffer[40]==my_ipaddr[1])
                        &&(RxBuffer[41]==my_ipaddr[2])
                        &&(RxBuffer[42]==my_ipaddr[3]))
                        {
                                tx_arp_reply();
                        }
                        // For recieve ICMP Echo form and passing while
loop
                        if((RxBuffer[13]==0x08)
                        &&(RxBuffer[14]==0x00)
                        &&(RxBuffer[24]==0x01)
                        &&(RxBuffer[27]==des_ipaddr[0])
                        &&(RxBuffer[28]==des_ipaddr[1])
                        &&(RxBuffer[29]==des_ipaddr[2])
                        &&(RxBuffer[30]==des_ipaddr[3])
                        &&(RxBuffer[35]==ICMP_ECHO_REPLY)//type ICMP_REPLY
                        &&(RxBuffer[36]==0x00)

        &&(RxBuffer[41]==0x00)&&(RxBuffer[42]==0x01)&&(RxBuffer[43]==0x
01))
                        {
                                checkbit = 1;

                        }
                }

                p2_4 = 0;
```

```
            p2_5 = 0;
            //CheckPacket();
            for(i=0;i<10;i++)
            adc_buf[i] = adc_read();
            tx_adc();

      }*/


}
```

Initial.h

```
#include "packets.h"

#define addr P2
#define data P0
#define Dbit P3

#define RxTxData  0xF0          // Receive/Transmit data (port 0)
#define RxTxData1 0xF2          // Receive/Transmit data (port 1)
#define TxCmd           0xF4        // Transmit Command
#define TxLength  0xF6          // Transmit Length
#define ISQ             0xF8      // Interrupt status queue
#define PPPtr           0xFA      // PacketPage pointer
#define PPData          0xFC      // PacketPage data (port 0)
#define PPData1         0xFE      // PacketPage data (port 1)

sbit write    = P2^7;           // IOW active low
sbit read     = P2^6;           // IOR active low

//------DELAY1ms------//
void DELAY1ms(unsigned char round)
      {
            unsigned char X;
                  TMOD = (TMOD|0x01);
                  IE      = (IE|0x80);

            for (X=0; X<round; X++)
            {
                  TH0 = 0xFC;
                  TL0 = 0x66;
                  TF0 = 0;
                  TR0 = 1;
                  while(TF0 == 0);
                  TR0 = 0;
            }
      }
//--------------------//

//--Function ioRead Used to Read from the Data Bus--//
      unsigned char ioRead(unsigned char address)
      {
            unsigned char value;
            data  = 0xFF;                 // Set P0 = 0xFF for
recieve data
            addr  = (address&0xFF);
            addr  = (address&0xBF); // read = 0 Active ioread
            value = data;                 // Keep data to register
            addr  = (address&0xF0); // read = 1 Deactive ioread
            return value;                 // Return to ioread
```

```
      }
//--Function ioWrite Used to write to the Data Bus--//
      void ioWrite(unsigned char address, unsigned char value)
      {
            data  = value;
            addr  = (address&0xFF);
            addr  = (address&0x7F); // write = 0 Active iowrite
            addr  = (address&0xF0);// write = 1 Deactive iowrite

      }


  ////////////////////////
      // Initialize Registers //
      ////////////////////////
void init cs8900()
 {


      //------Reset Chip------//
      // (1) Write 0x0114 to PacketPage Pointer (Data Sheet P.64)
      // (2) Write 0x0040 to PacketPage Data Port (Set bit 6)
            ioWrite(PPPtr,      0x14);
            ioWrite(PPPtr + 1,  0x01);
            ioWrite(PPData,     0x40);
            ioWrite(PPData + 1, 0x00);
      // Delay time about 125 ms for chip resetting in progress
            DELAYlms(126);

      //---Configure Receiver Control fo Promiscious mode, RxOK---//
      // (1) Write 0x0104 to PacketPage Pointer (Data Sheet P.54)
      // (2) Write 0x0180 to PacketPage Data Port (Set bit 7,8)
            ioWrite(PPPtr,      0x04);
            ioWrite(PPPtr + 1,  0x01);
            ioWrite(PPData,     0x00);
            ioWrite(PPData + 1, 0x2d);

      //--Set 10BaseT, SerRxOn, SerTxOn in Line Control--//
      // (1) Write 0x0112 to PacketPage Pointer (Data Sheet P.62)
      // (2) Write 0x00c0 to PacketPage Data Port (Set bit 6,7)
            ioWrite(PPPtr,      0x12);
            ioWrite(PPPtr + 1,  0x01);
            ioWrite(PPData,     0xc0);
            ioWrite(PPData + 1, 0x00);

      //--Module Embedded Ethernet MAC Address (00-04-E2-7A-75-71)--
//
      //Write MAC(IEEE) Address (Data Sheet P.71)

            ioWrite(PPPtr,      0x58);
            ioWrite(PPPtr + 1,  0x01);
            ioWrite(PPData,     0x00);
            ioWrite(PPData+1 ,  0x04);

            ioWrite(PPPtr,          0x5A);
            ioWrite(PPPtr + 1,      0x01);
            ioWrite(PPData,         0xE2);
            ioWrite(PPData-1 ,  0x7a);
```

```
            ioWrite(PPPtr,          0x5C);
            ioWrite(PPPtr + 1,      0x01);
            ioWrite(PPData,         0x75);
            ioWrite(PPData+1 ,  0x71);
}

void rxpacket()
{
      unsigned int Length;
      unsigned char BusST0,BusST1;
      unsigned char DataL,DataH,i,j,k;
      unsigned int x,y;
      do {
      ioWrite(PPPtr,     0x24);
      ioWrite(PPPtr + 1, 0x01);
      BusST0 = ioRead(PPData);
      BusST1 = ioRead(PPData+1);
      }while((BusST0==0x04)&(BusST1==0x00));
      // frame is present //
      ioWrite(PPPtr,     0x02);
      ioWrite(PPPtr + 1, 0x04);
      DataL=ioRead(PPData);
      DataH=ioRead(PPData+1);
      x=DataL;
      y=DataH;
      y=y<<8;
      Length=x+y;
      //Protect Loop more than Buffer Array
      if (Length>159)
            Length=159;
      j=0;
      k=0;
      for (i=1;i<=Length;i++)
      {
            ioWrite(PPPtr,     0x04 + j);
            ioWrite(PPPtr + 1, 0x04);
            RxBuffer[i]=ioRead(PPData + k);
                  if(i%2==1) k=1;
                        else {
                        j=j+2;
                        k=0;
                        }
      }

}
```

cs8900.c

```
#include "packets.h"
#include "cs8900.h"

//------DELAY1ms------//
void DELAY1ms(unsigned char round)
      {
            unsigned char X;
                  TMOD = (TMOD|0x01);
                  IE     = (IE|0x80);

            for (X=0; X<round; X++)
            {
                  TH0 = 0xFC;
```

```
                TL0 = 0x66;
                TF0 = 0;
                TR0 = 1;
                while(TF0 == 0);
                TR0 = 0;
            }
        }
//--------------------//

//--Function ioRead Used to Read from the Data Bus--//
unsigned char ioRead(unsigned char address)
        {
            unsigned char value;
            Pdata  = 0xFF;                  // Set P0 = 0xFF for
recieve data
            addr  = (address&0xFF);
            addr  = (address&0xBF); // read = 0 Active ioread
            value = Pdata;                  // Keep data to register
            addr  = (address&0xF0); // read = 1 Deactive ioread
            return value;                   // Return to ioread


        }
//--Funtcion ioWrite Used to write to the Data Bus--//
void ioWrite(unsigned char address, unsigned char value)
        {
            Pdata  = value;
            addr  = (address&0xFF);
            addr  = (address&0x7F); // write = 0 Active iowrite
            addr  = (address&0xF0);// write = 1 Deactive iowrite

        }


  ////////////////////////
        // Initialize Registers //
        ////////////////////////
void init_cs8900()
 {


        //------Reset Chip------//
        // (1) Write 0x0114 to PacketPage Pointer (Data Sheet P.64)
        // (2) Write 0x0040 to PacketPage Data Port (Set bit 6)
            ioWrite(PPPtr,      0x14);
            ioWrite(PPPtr + 1,  0x01);
            ioWrite(PPData,     0x40);
            ioWrite(PPData + 1, 0x00);
        // Delay time about 125 ms for chip resetting in progress
            DELAY1ms(126);

        //---Configure Receiver Control fo Promiscious mode, RxOK---//
        // (1) Write 0x0104 to PacketPage Pointer (Data Sheet P.54)
        // (2) Write 0x0180 to PacketPage Data Port (Set bit 7,8)
            ioWrite(PPPtr,      0x04);
            ioWrite(PPPtr + 1,  0x01);
            ioWrite(PPData,     0x00);
            ioWrite(PPData + 1, 0x2d);

        //--Set 10BaseT, SerRxOn, SerTxOn in Line Control--//
        // (1) Write 0x0112 to PacketPage Pointer (Data Sheet P.62)
```

```
	// (2) Write 0x00c0 to PacketPage Data Port (Set bit 6,7)
		ioWrite(PPPtr,		0x12);
		ioWrite(PPPtr + 1,	0x01);
		ioWrite(PPData,		0xc0);
		ioWrite(PPData + 1,	0x00);

	//--Module Embedded Ethernet MAC Adrress (00-04-E2-7A-75-71)--
//
	//Write MAC(IEEE) Address (Data Sheet P.71)

		ioWrite(PPPtr,		0x58);
		ioWrite(PPPtr + 1,	0x01);
		ioWrite(PPData,		my_macaddr[0]);
		ioWrite(PPData+1 ,	my_macaddr[1]);

		ioWrite(PPPtr,			0x5A);
		ioWrite(PPPtr + 1,		0x01);
		ioWrite(PPData,			my_macaddr[2]);
		ioWrite(PPData+1 ,	my_macaddr[3]);

		ioWrite(PPPtr,			0x5C);
		ioWrite(PPPtr + 1,		0x01);
		ioWrite(PPData,			my_macaddr[4]);
		ioWrite(PPData+1 ,	my_macaddr[5]);
}

void rxpacket()
{
	unsigned int Length;
	unsigned char BusST0,BusST1;
	unsigned char DataL,DataH,i,j,k;
	unsigned int x,y;
	do {
	ioWrite(PPPtr,		0x24);
	ioWrite(PPPtr + 1,	0x01);
	BusST0 = ioRead(PPData);
	BusST1 = ioRead(PPData+1);
	}while((BusST0==0x04)&(BusST1==0x00));
	// frame is present //
	ioWrite(PPPtr,		0x02);
	ioWrite(PPPtr + 1,	0x04);
	DataL=ioRead(PPData);
	DataH=ioRead(PPData+1);
	x=DataL;
	y DataH;
	y=y<<8;
	Length=x+y;
	//Protect Loop more than Buffer Array
	if (Length>159)
		Length=159;
	j=0;
	k=0;
	for (i=1;i<=Length;i++)
	{
		ioWrite(PPPtr,		0x04 + j);
		ioWrite(PPPtr + 1,	0x04);
		RxBuffer[i]=ioRead(PPData + k);
			if(i%2==1) k=1;
				else {
				j=j+2;
				k=0;
```

```
                        }
        }

;
//
*************************************************************************
******
// cs8900.h: defines for cs8900.c
//
*************************************************************************
******

#include <reg52.h>

#ifndef H__CS8900
#define H__CS8900

#define addr P2
#define Pdata P0
#define Dbit P3

#define RxTxData  0xF0          // Receive/Transmit data (port 0)
#define RxTxData1 0xF2          // Receive/Transmit data (port 1)
#define TxCmd           0xF4        // Transmit Command
#define TxLength  0xF6          // Transmit Length
#define ISQ             0xF8        // Interrupt status queue
#define PPPtr           0xFA          // PacketPage pointer
#define PPData          0xFC          // PacketPage data (port 0)
#define PPData1         0xFE          // PacketPage data (port 1)

sbit write    = P2^7;               // IOW active low
sbit read     = P2^6;               // IOR active low
void DELAY1ms(unsigned char round);
unsigned char ioRead(unsigned char address);
void ioWrite(unsigned char address, unsigned char value);
void init_cs8900();
void rxpacket();

#endif
```

Packet.h

```
//
*************************************************************************
******
// An 8051 Based Web Server
// packets.h: definitions for packet structures
// By Mason Kidd 10/25/01
//
*************************************************************************
******

#ifndef H__PACKETS
#define H__PACKETS

#define MAC_ADDR_LEN 6

#define ETH_HEADER 14
#define ARP_HEADER 8
#define IP_HEADER 20
#define ICMP_HEADER 8
```

```
#define TCP_HEADER 20
#define UDP_HEADER 8

#define UDP_DATA_INDEX 43

#define BUF_LEN 100
#define ADC_BUF_LEN 10

extern  unsigned char code my_macaddr[6];
extern  unsigned char code my_ipaddr[4];
extern  unsigned char code des_ipaddr[4];
extern  unsigned char code des_macaddr[6];

extern unsigned char code my_portH;
extern unsigned char code my_portL;
extern unsigned char code des_portH;
extern unsigned char code des_portL;

extern unsigned char idata RxBuffer[160];
struct eth_hdr
{
  unsigned char dhost[MAC_ADDR_LEN];              // destination MAC
address
  unsigned char shost[MAC_ADDR_LEN];              // source MAC address
  unsigned int type;                            // packet type
};

#define ETHER_IP 0x0800
#define ETHER_ARP 0x0806

/*struct arp_hdr
{
      unsigned int ar_hrd;              // Hardware Address format
      unsigned int ar_pro;              // Protocol Address format
      unsigned char ar_hln;             // Byte length of each hardware
address
      unsigned char ar_pln;             // Byte length of each protocol
address
      unsigned int ar_op;               // Opcode
};*/

#define ARP_REQUEST 0x01
#define ARP_REPLY 0x02
#define ARP_HRD_ETHER 0x01

/*struct eth_arp
{
      struct arp_hdr eth_arp_hdr;
      unsigned char ar_sha[MAC_ADDR_LEN]; // Sender's MAC address
      unsigned char ar_spa[4];            // Sender's protocol
address
      unsigned char ar_tha[MAC_ADDR_LEN]; // Target's MAC address
      unsigned char ar_tpa[4];            // Target's protocol
address
};*/

struct ip_hdr
{
      unsigned char verIHL;         // version - 4 bits, Internet
Header Length - 4 bits
      unsigned char TOS;            // Type of Service - 8 bits
```

```
      unsigned int totlen;          // Total Length of the datagram -
16 bits
      unsigned int id;        // Identification of datagram - 16 bits
      unsigned int fragoff;         // Flags - 3 bits, Fragment
Offset - 13 bits
      unsigned char TTL;            // Time to Live - 8 bits
      unsigned char proto;          // Protocol - 8 bits
      unsigned int hdrchksum;       // Header Checksum - 16 bits
      unsigned char srcIP[4];       // Source IP Address - 32 bits
      unsigned char destIP[4];      // Destination IP Address - 32
bits
};

#define IP_ICMP 0x01
#define IP_TCP 0x06
#define IP_UDP 0x11

/*struct icmp_hdr
{
      unsigned char type;
      unsigned char icode;
      unsigned int checksum;
      unsigned int identifier;
      unsigned int sequence;
//    unsigned long icdata;
};*/

#define ICMP_ECHO_REPLY         0x00      /* Echo Reply
*/
#define ICMP_DEST_UNREACH       3         /* Destination Unreachable
*/
#define ICMP_SOURCE_QUENCH      4         /* Source Quench
*/
#define ICMP_REDIRECT           5         /* Redirect (change route)
*/
#define ICMP_ECHO_REQUEST       0x08      /* Echo Request
*/
#define ICMP_TIME_EXCEEDED      11        /* Time Exceeded
*/
#define ICMP_PARAMETERPROB      12        /* Parameter Problem
*/
#define ICMP_TIMESTAMP          13        /* Timestamp Request
*/
#define ICMP_TIMESTAMPREPLY     14        /* Timestamp Reply
*/
#define ICMP_INFO_REQUEST       15        /* Information Request
*/
#define ICMP_INFO_REPLY         16        /* Information Reply
*/
#define ICMP_ADDRESS            17        /* Address Mask Request
*/
#define ICMP_ADDRESSREPLY       18        /* Address Mask Reply
*/

/*struct udp_hdr
{
      unsigned int src_port;        // Source port, optional
      unsigned int dst_port;        // Destination port
      unsigned int length;          // Length of datagram in bytes,
incl. header
      unsigned int checksum;        // Checksum of psuedo-header
```

```
};
*/
/*
struct tcp_hdr
{
      unsigned int src_port;        // Source port
      unsigned int dst_port;        // Destination port
      unsigned long seq;                  // Sequence number
      unsigned long ack;                  // Acknowledgement number
      unsigned char data_off;       // Data offset - upper 4 bits
      unsigned char cntrl_bits;     // Control bits - lower 6 bits
      unsigned int window;          // Window
      unsigned int checksum;        // Checksum
      unsigned int urgent;          // Urgent pointer
      // Options may additionally follow
};*/


#endif
```

Arp.c

```
#include "packets.h"
#include "cs8900.h"

void tx_arp_reply()
{
      ////////////////////////
      // Transmit  Datagram //
      ////////////////////////

      unsigned char BusST0,BusST1;

      // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
            ioWrite(TxCmd,          0xc0);
            ioWrite(TxCmd + 1,      0x00);

      // 42 bytes to be sent
      // Ethernet Header(14) + IP Header(20) + UDP Header(8) +
Data(2)
            ioWrite(TxLength,   0x2a);
            ioWrite(TxLength-1, 0x00);

      // Check Bus status (Data Sheet P.67)
            ioWrite(PPPtr,      0x38);
            ioWrite(PPPtr + 1, 0x01);

      // Check Bus status,Is ready for transmit (Check bit 7)
            do {
                  BusST0 = ioRead(PPData);
                  BusST1 = ioRead(PPData+1);
            }while(!(BusST1==0x01));

      //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

      ////////////////////////
      //  Ethernet Header  // (14 bytes)
      ////////////////////////

// Send Destination (6 bytes)
```

```
      // MAC Address Microcomputer Top (00-08-02-f4-f8-5b) Loon(00-
0c-6e-a6-34-a6)
            ioWrite(RxTxData,       des_macaddr[0]);
            ioWrite(RxTxData+1, des_macaddr[1]);
            ioWrite(RxTxData, des_macaddr[2]);
            ioWrite(RxTxData+1,     des_macaddr[3]);
            ioWrite(RxTxData,       des_macaddr[4]);
            ioWrite(RxTxData+1,     des_macaddr[5]);

      // Send Source (6 bytes)
      // MAC Address Embedded Device (00-04-e2-7a-75-71)
            ioWrite(RxTxData,       my_macaddr[0]);
            ioWrite(RxTxData+1, my_macaddr[1]);
            ioWrite(RxTxData, my_macaddr[2]);
            ioWrite(RxTxData+1,     my_macaddr[3]);
            ioWrite(RxTxData,       my_macaddr[4]);
            ioWrite(RxTxData+1,     my_macaddr[5]);

      // Send Protocol Type (2 bytes)
      // ARP=0x0806
            ioWrite(RxTxData,       0x08);
            ioWrite(RxTxData+1, 0x06);


      ////////////////
      //  ARP Header  // (28 bytes)
      ////////////////

      // Hardware Type (2 byte)
      // Ethernet
            ioWrite(RxTxData,       0x00);
            ioWrite(RxTxData+1, 0x01);

      // Protocol type (2 bytes)
            ioWrite(RxTxData,       0x08);
            ioWrite(RxTxData+1, 0x00);

      // Hardware size (1 bytes)
            ioWrite(RxTxData,       0x06);
      //Protocol size (1 byte)
            ioWrite(RxTxData+1, 0x04);

      // Opcode (2 bytes)
      // reply 02
            ioWrite(RxTxData, 0x00);
            ioWrite(RxTxData+1, 0x02);
      //sender MAC hardware (6bytes) (Embedded Device 00-04-e2-7a-75-
71)
            ioWrite(RxTxData,       my_macaddr[0]);
            ioWrite(RxTxData+1, my_macaddr[1]);
            ioWrite(RxTxData, my_macaddr[2]);
            ioWrite(RxTxData+1,     my_macaddr[3]);
            ioWrite(RxTxData,       my_macaddr[4]);
            ioWrite(RxTxData+1,     my_macaddr[5]);
      // Source Address (4 bytes)
      // IP address 161.246.18.156 (Embedded Device)
            ioWrite(RxTxData,       my_ipaddr[0]);   // 161
            ioWrite(RxTxData+1, my_ipaddr[1]);   // 246
            ioWrite(RxTxData,       my_ipaddr[2]);   // 18
            ioWrite(RxTxData+1, my_ipaddr[3]);   // 156
```

```
      //Dest MAC computer (6bytes) Top (00-08-02-f4-f8-5b) Loon(00-
0c-6e-a6-34-a6)
            ioWrite(RxTxData,       des_macaddr[0]);
            ioWrite(RxTxData+1, des_macaddr[1]);
            ioWrite(RxTxData, des_macaddr[2]);
            ioWrite(RxTxData+1,     des_macaddr[3]);
            ioWrite(RxTxData,       des_macaddr[4]);
            ioWrite(RxTxData+1,     des_macaddr[5]);

      // Destination Address (4 bytes)
      // IP address 161.246.18.157 (Labtop)
            ioWrite(RxTxData,   des_ipaddr[0]);   // 161
            ioWrite(RxTxData+1, des_ipaddr[1]);   // 246
            ioWrite(RxTxData,       des_ipaddr[2]);   // 18
            ioWrite(RxTxData+1, des_ipaddr[3]);   // 157
}

/
*****************************************************************
******
// arp.h: defines for ARP
//
*****************************************************************
******

#ifndef H__ARP
#define H__ARP

void tx_arp_reply();

#endif
```

Icmp.c

```
include "cs8900.h"
#include "packets.h"

void tx_icmp_req()
{
    unsigned char BusST0,BusST1;

      // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
            ioWrite(TxCmd,          0xc0);
            ioWrite(TxCmd + 1,      0x00);

      // 64 bytes to be sent
      // Ethernet Header(14) + IP Header(20) + ICMP Header(8) +
Data(22)
            ioWrite(TxLength,   0x40);
            ioWrite(TxLength+1, 0x00);

      // Check Bus status (Data Sheet P.67)
            ioWrite(PPPtr,     0x38);
            ioWrite(PPPtr + 1, 0x01);

      // Check Bus status,Is ready for transmit (Check bit 7)
            do {
                  BusST0 = ioRead(PPData);
                  BusST1 = ioRead(PPData+1);
            }while(!(BusST1==0x01));
```

```
      //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

      ////////////////////////
      //  Ethernet Header  // (14 bytes)
      ////////////////////////

      // Send Destination (6 bytes)
      // MAC Address Microcomputer Top (00-08-02-f4-f8-5b) Loon(00-
0c-6e-a6-34-a6)
            ioWrite(RxTxData,       des_macaddr[0]);
            ioWrite(RxTxData+1, des_macaddr[1]);
            ioWrite(RxTxData, des_macaddr[2]);
            ioWrite(RxTxData+1,     des_macaddr[3]);
            ioWrite(RxTxData,       des_macaddr[4]);
            ioWrite(RxTxData+1,     des_macaddr[5]);
      // Send Source (6 bytes)
      // MAC Address Embedded Device (00-04-e2-7a-75-71)
            ioWrite(RxTxData,       my_macaddr[0]);
            ioWrite(RxTxData+1,     my_macaddr[1]);
            ioWrite(RxTxData,       my_macaddr[2]);
            ioWrite(RxTxData+1, my_macaddr[3]);
            ioWrite(RxTxData,       my_macaddr[4]);
            ioWrite(RxTxData+1,     my_macaddr[5]);

      // Send Protocol Type (2 bytes)
      // IP=0x0800
            ioWrite(RxTxData,       0x08);
            ioWrite(RxTxData+1, 0x00);

      ////////////////
      //  IP Header  // (20 bytes)
      ////////////////

      // Version Header Length (1 byte)
      // Using IPv4
            ioWrite(RxTxData,       0x45);

      // Service (1 byte)
      // Normally set to zero because not used
            ioWrite(RxTxData+1, 0x00);

      // Length (2 bytes)
      // 50 byte Length (IP Header(20) + ICMP Header(8) + Data(22))
            ioWrite(RxTxData,       0x00);
            ioWrite(RxTxData+1, 0x32);

      // Identifier (2 bytes)
      // 0x0000
            ioWrite(RxTxData,       0x00);
            ioWrite(RxTxData+1, 0x00);

      // Flags Fragment offset (2 bytes)(no flag)
      // 0x0000
            ioWrite(RxTxData, 0x00);
            ioWrite(RxTxData+1, 0x00);

      // Time to Live (1 byte)
      // 63
            ioWrite(RxTxData,       0x3f);

      // Protocol (1 byte)
```

```
// ICMP 1 = 0x01
      ioWrite(RxTxData+1, 0x01);

// Checksum IP Header (2 bytes)
// 0x12a6
      ioWrite(RxTxData,       0x12);
      ioWrite(RxTxData+1, 0xa6);

// Source Address (4 bytes)
// IP address 161.246.18.156 (Embedded Device)
      ioWrite(RxTxData,       my_ipaddr[0]);   // 161
      ioWrite(RxTxData+1, my_ipaddr[1]);   // 246
      ioWrite(RxTxData,       my_ipaddr[2]);   // 18
      ioWrite(RxTxData+1, my_ipaddr[3]);   // 156

// Destination Address (4 bytes)
// IP address 161.246.18.157 (Labtop)
      ioWrite(RxTxData,       des_ipaddr[0]);   // 161
      ioWrite(RxTxData+1, des_ipaddr[1]); // 246
      ioWrite(RxTxData, des_ipaddr[2]);   // 18
      ioWrite(RxTxData+1, des_ipaddr[3]); // 157

//////////////////
//  ICMP Header// (8 bytes)
//////////////////

// Type (1 bytes)
// 0x08
      ioWrite(RxTxData, ICMP_ECHO_REQUEST);

// Code (1 byte)
// 0x00
      ioWrite(RxTxData+1, 0x00);

// Checksum (2 bytes)
// 0xECF3
      ioWrite(RxTxData,       0xec);
      ioWrite(RxTxData+1, 0xf3);

// Identifier (2 bytes)
// 0xC00C
      ioWrite(RxTxData,       0x00);
      ioWrite(RxTxData+1, 0x00);

// Sequence (2 bytes)
// 0x0001
      ioWrite(RxTxData,       0x00);
      ioWrite(RxTxData+1, 0x01);

////////////
//  Data  // (22 bytes)
////////////

// Write Message (22 bytes)
      ioWrite(RxTxData, 0x01);
      ioWrite(RxTxData+1,     0x01);
      ioWrite(RxTxData, 0x01);
      ioWrite(RxTxData+1,     0x01);
      ioWrite(RxTxData, 0x01);
      ioWrite(RxTxData+1,     0x01);
      ioWrite(RxTxData, 0x01);
```

```
            ioWrite(RxTxData+1,     0x01);
            ioWrite(RxTxData, 0x01);
            ioWrite(RxTxData+1,     0x01);
            ioWrite(RxTxData, 0x01);
            ioWrite(RxTxData+1,     0x01);
            ioWrite(RxTxData, 0x01);
            ioWrite(RxTxData+1,     0x01);
            ioWrite(RxTxData, 0x01);
            ioWrite(RxTxData+1,     0x01);
            ioWrite(RxTxData, 0x01);
            ioWrite(RxTxData+1,     0x01);
            ioWrite(RxTxData, 0x01);
            ioWrite(RxTxData+1,     0x01);
            ioWrite(RxTxData, 0x01);
            ioWrite(RxTxData+1,     0x01);

}

//
*************************************************************************
******
// icmp.h: defines for icmp.c
//
*************************************************************************
******

#ifndef H__ICMP
#define H__ICMP

void tx_icmp_req();

#endif
```

Extram.h

```
sbit rd = P2^4;
sbit wr = P2^5;
void memwr(unsigned int add,unsigned char dat)
{
      unsigned char addrhigh,addrlow;
      addr = 0xF0;
      addrlow = add & 0x00FF;
      addrhigh = (add & 0xFF00)>>8;
      if (addrhigh==0x00) {A8=0;A9=0;}
      if (addrhigh==0x01) {A8=1;A9=0;}
      if (addrhigh==0x02) {A8=0;A9=1;}
      if (addrhigh==0x03) {A8=1;A9=1;}
      io = dat;
      data = addrlow;
      addr = 0xD0;
      addr = 0xF0;
}
unsigned char memrd(unsigned int add)
{
      unsigned char addrhigh,addrlow;
      addr = 0xF0;
      addrlow = add & 0x00FF;
      addrhigh = (add & 0xFF00)>>8;
      if (addrhigh==0x00) {A8=0;A9=0;}
      if (addrhigh==0x01) {A8=1;A9=0;}
```

```
        if (addrhigh==0x02) {A8=0;A9=1;}
        if (addrhigh==0x03) {A8=1;A9=1;}
        io = 0xFF;
        data = addrlow;
        addr   0xE0;
        return(io);
        addr = 0xF0;
}
```

Udp.c

```
#include "packets.h"
#include "cs8900.h"
#include "udp.h"

extern unsigned char idata adc_buf[10];
void CheckPacket()
{
            unsigned char passloop;

            passloop = 0;
            while(passloop == 0)
            {
                  rxpacket();

      if((RxBuffer[13]==0x08)&&(RxBuffer[14]==0x00)&&(RxBuffer[24]==0
x11)
                  //161.246.18.157

      &&(RxBuffer[27]==des_ipaddr[0])&&(RxBuffer[28]==des_ipaddr[1])&
&(RxBuffer[29]==des_ipaddr[2])
                  &&(RxBuffer[30]==des_ipaddr[3])
                  //161.246.18.156

      &&(RxBuffer[31]==my_ipaddr[0])&&(RxBuffer[32]==my_ipaddr[1])&&(
RxBuffer[33]==my_ipaddr[2])
                  &&(RxBuffer[34]==my_ipaddr[3])
                  //Src port 5000 Dest Port 1300

      &&(RxBuffer[35]==des_portH)&&(RxBuffer[36]==des_portL)&&(RxBuff
er[37]==my_portH)
                  &&(RxBuffer[38]==my_portL))
                        {

                              tx_echo();
                              passloop = 1;

                        }
            }

}
void tx_adc()
{
    ////////////////////////
      // Transmit  Datagram //
      ////////////////////////
            unsigned char BusST0,BusST1;

      // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
            ioWrite(TxCmd,          0xc0);
            ioWrite(TxCmd + 1,      0x00);
```

```
      // 64 bytes to be sent
      // Ethernet Header(14) + IP Header(20) + UDP Header(8) +
Data(22)
            ioWrite(TxLength,   0x40);
            ioWrite(TxLength+1, 0x00);

      // Check Bus status (Data Sheet P.67)
            ioWrite(PPPtr,     0x38);
            ioWrite(PPPtr + 1, 0x01);

      // Check Bus status,Is ready for transmit (Check bit 7)
            do {
                  BusST0 = ioRead(PPData);
                  BusST1 = ioRead(PPData+1);
            }while(!(BusST1==0xC1));

      //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

      ////////////////////////
      //  Ethernet Header  // (14 bytes)
      ////////////////////////

      // Send Destination (6 bytes)
      // MAC Address Microcomputer Top (00-08-C2-f4-f8-5b) Loon(00-
0c-6e-a6-34-a6)
            ioWrite(RxTxData,       des_macaddr[0]);
            ioWrite(RxTxData+1, des_macaddr[1]);
            ioWrite(RxTxData, des_macaddr[2]);
            ioWrite(RxTxData+1,     des_macaddr[3]);
            ioWrite(RxTxData,       des_macaddr[4]);
            ioWrite(RxTxData+1,     des_macaddr[5]);

      // Send Source (6 bytes)
      // MAC Address Embedded Device (00-04-e2-7a-75-71)
            ioWrite(RxTxData,       my_macaddr[0]);
            ioWrite(RxTxData+1, my_macaddr[1]);
            ioWrite(RxTxData, my_macaddr[2]);
            ioWrite(RxTxData+1,     my_macaddr[3]);
            ioWrite(RxTxData,       my_macaddr[4]);
            ioWrite(RxTxData+1,     my_macaddr[5]);

      // Send Protocol Type (2 bytes)
      // IP 0x0800
            ioWrite(RxTxData,       0x08);
            ioWrite(RxTxData+1, 0x00);


      ////////////////
      //  IP Header  // (20 bytes)
      ////////////////

      // Version Header Length (1 byte)
      // Using IPv4
            ioWrite(RxTxData,       0x45);

      // Service (1 byte)
      // Normally set to zero because not used
            ioWrite(RxTxData+1, 0x00);

      // Length (2 bytes)
```

```
// 50 byte Length (IP Header(20) + UDP Header(8) + Data(22))
      ioWrite(RxTxData,       0x00);
      ioWrite(RxTxData+1, 0x32);

// Identifier (2 bytes)
// 0x0000
      ioWrite(RxTxData,       0x00);
      ioWrite(RxTxData+1, 0x00);

// Flags Fragment offset (2 bytes)(no flag)
// 0x4000
      ioWrite(RxTxData, 0x40);
      ioWrite(RxTxData+1, 0x00);

// Time to Live (1 byte)
// 63
      ioWrite(RxTxData,       0x3f);

// Protocol (1 byte)
// UDP 17 = 0x11
      ioWrite(RxTxData+1, 0x11);

// Checksum IP Header (2 bytes)
// 0xd295
      ioWrite(RxTxData,       0xd2);
      ioWrite(RxTxData+1, 0x95);

// Source Address (4 bytes)
// IP address 161.246.18.156 (Embedded Device)
      ioWrite(RxTxData,       my_ipaddr[0]);   // 161
      ioWrite(RxTxData+1, my_ipaddr[1]);   // 246
      ioWrite(RxTxData,       my_ipaddr[2]);   // 18
      ioWrite(RxTxData+1, my_ipaddr[3]);   // 156

// Destination Address (4 bytes)
// IP address 161.246.18.157 (Labtop)
      ioWrite(RxTxData,       des_ipaddr[0]);   // 161
      ioWrite(RxTxData+1, des_ipaddr[1]);   // 246
      ioWrite(RxTxData,       des_ipaddr[2]);   // 18
      ioWrite(RxTxData+1, des_ipaddr[3]);   // 157

////////////////
//  UDP Header // (8 bytes)
////////////////

// Source Port (2 bytes)
// Port 1300 (0514)
      ioWrite(RxTxData,       0x05);
      ioWrite(RxTxData+1, 0x14);

// Destination Port (2 bytes)
// Port 5000 (0x1388)
      ioWrite(RxTxData,       0x13);
      ioWrite(RxTxData+1, 0x88);

// Message Length (2 bytes)
// 30 bytes (UDP Header(8) + Data(22))
      ioWrite(RxTxData,       0x00);
      ioWrite(RxTxData+1, 0x1e);

// Checksum (2 bytes)
```

```
        // 0x0000 (no checksum used)
              ioWrite(RxTxData,       0x00);
              ioWrite(RxTxData+1, 0x00);

        ////////////
        //  Data  // (22 bytes)
        ////////////

        // Write Message (22 bytes)
              ioWrite(RxTxData,adc_buf[0]);
              ioWrite(RxTxData+1,adc_buf[1]);
              ioWrite(RxTxData,adc_buf[2]);
              ioWrite(RxTxData+1,adc_buf[3]);
              ioWrite(RxTxData,adc_buf[4]);
              ioWrite(RxTxData+1,adc_buf[5]);
              ioWrite(RxTxData,adc_buf[6]);
              ioWrite(RxTxData+1,adc_buf[7]);
              ioWrite(RxTxData,adc_buf[8]);
              ioWrite(RxTxData+1,adc_buf[9]);
              ioWrite(RxTxData,adc_buf[10]);
              ioWrite(RxTxData+1,adc_buf[11]);
              ioWrite(RxTxData,adc_buf[12]);
              ioWrite(RxTxData+1,adc_buf[13]);
              ioWrite(RxTxData,adc_buf[14]);
              ioWrite(RxTxData+1,adc_buf[15]);
              ioWrite(RxTxData,adc_buf[16]);
              ioWrite(RxTxData+1,adc_buf[17]);
              ioWrite(RxTxData,adc_buf[18]);
              ioWrite(RxTxData+1,adc_buf[19]);
              ioWrite(RxTxData,adc_buf[20]);
              ioWrite(RxTxData+1,adc_buf[21]);

}

void txudp()
{
    ////////////////////////
      // Transmit  Datagram //
      ////////////////////////
            unsigned char BusST0,BusST1;

      // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
            ioWrite(TxCmd,          0xc0);
            ioWrite(TxCmd + 1,      0x00);

      // 64 bytes to be sent
      // Ethernet Header(14) + IP Header(20) + UDP Header(8) +
Data(22)
            ioWrite(TxLength,   0x40);
            ioWrite(TxLength+1, 0x00);

      // Check Bus status (Data Sheet P.67)
            ioWrite(PPPtr,     0x38);
            ioWrite(PPPtr + 1, 0x01);

      // Check Bus status,Is ready for transmit (Check bit 7)
            do {
                  BusST0 = ioRead(PPData);
                  BusST1 = ioRead(PPData+1);
            }while(!(BusST1== 0x01));
```

```
      //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

      ////////////////////////
      //  Ethernet Header  // (14 bytes)
      ////////////////////////

      // Send Destination (6 bytes)
      // MAC Address Microcomputer Top (00-08-02-f4-f8-5b) Loon(00-
0c-6e-a6-34-a6)
            ioWrite(RxTxData,       des_macaddr[0]);
            ioWrite(RxTxData+1, des_macaddr[1]);
            ioWrite(RxTxData, des_macaddr[2]);
            ioWrite(RxTxData+1,     des_macaddr[3]);
            ioWrite(RxTxData,       des_macaddr[4]);
            ioWrite(RxTxData+1,     des_macaddr[5]);

      // Send Source (6 bytes)
      // MAC Address Embedded Device (00-04-e2-7a-75-71)
            ioWrite(RxTxData,       my_macaddr[0]);
            ioWrite(RxTxData+1, my_macaddr[1]);
            ioWrite(RxTxData, my_macaddr[2]);
            ioWrite(RxTxData+1,     my_macaddr[3]);
            ioWrite(RxTxData,       my_macaddr[4]);
            ioWrite(RxTxData+1,     my_macaddr[5]);

      // Send Protocol Type (2 bytes)
      // IP 0x0800
            ioWrite(RxTxData,       0x08);
            ioWrite(RxTxData+1, 0x00);


      ////////////////
      //  IP Header  // (20 bytes)
      ////////////////

      // Version Header Length (1 byte)
      // Using IPv4
            ioWrite(RxTxData,       0x45);

      // Service (1 byte)
      // Normally set to zero because not used
            ioWrite(RxTxData+1, 0x00);

      // Length (2 bytes)
      // 50 byte Length (IP Header(20) + UDP Header(8) + Data(22))
            ioWrite(RxTxData,       0x00);
            ioWrite(RxTxData+1, 0x32);

      // Identifier (2 bytes)
      // 0x0000
            ioWrite(RxTxData,       0x00);
            ioWrite(RxTxData+1, 0x00);

      // Flags Fragment offset (2 bytes)(no flag)
      // 0x4000
            ioWrite(RxTxData, 0x40);
            ioWrite(RxTxData+1, 0x00);

      // Time to Live (1 byte)
      // 63
            ioWrite(RxTxData,       0x3f);
```

```
// Protocol (1 byte)
// UDP 17 = 0x11
      ioWrite(RxTxData+1, 0x11);

// Checksum IP Header (2 bytes)
// 0xd295
      ioWrite(RxTxData,        0xd2);
      ioWrite(RxTxData+1, 0x95);

// Source Address (4 bytes)
// IP address 161.246.18.156 (Embedded Device)
      ioWrite(RxTxData,        my_ipaddr[0]);   // 161
      ioWrite(RxTxData+1, my_ipaddr[1]);   // 246
      ioWrite(RxTxData,        my_ipaddr[2]);   // 18
      ioWrite(RxTxData+1, my_ipaddr[3]);   // 156

// Destination Address (4 bytes)
// IP address 161.246.18.157 (Labtop)
      ioWrite(RxTxData,        des_ipaddr[0]);   // 161
      ioWrite(RxTxData+1, des_ipaddr[1]); // 246
      ioWrite(RxTxData, des_ipaddr[2]);   // 18
      ioWrite(RxTxData+1, des_ipaddr[3]); // 157

//////////////////
//  UDP Header // (8 bytes)
//////////////////

// Source Port (2 bytes)
// Port 1300 (0514)
      ioWrite(RxTxData,        my_portH);
      ioWrite(RxTxData+1, my_portL);

// Destination Port (2 bytes)
// Port 5000 (0x1388)
      ioWrite(RxTxData,        des_portH);
      ioWrite(RxTxData+1, des_portL);

// Message Length (2 bytes)
// 30 bytes (UDP Header(8) + Data(22))
      ioWrite(RxTxData,        0x00);
      ioWrite(RxTxData+1, 0x1e);

// Checksum (2 bytes)
// 0x0000 (no checksum used)
      ioWrite(RxTxData,        0x00);
      ioWrite(RxTxData-1, 0x00);

////////////
//  Data  // (22 bytes)
////////////

// Write Message (22 bytes)
      ioWrite(RxTxData,'C');
      ioWrite(RxTxData+1,'o');
      ioWrite(RxTxData,'n');
      ioWrite(RxTxData+1,'n');
      ioWrite(RxTxData,'e');
      ioWrite(RxTxData+1,'c');
      ioWrite(RxTxData,'t');
      ioWrite(RxTxData+1,'i');
```

```
            ioWrite(RxTxData,'o');
            ioWrite(RxTxData+1,'n');
            ioWrite(RxTxData,' ');
            ioWrite(RxTxData+1,'R');
            ioWrite(RxTxData,'e');
            ioWrite(RxTxData+1,'a');
            ioWrite(RxTxData,'d');
            ioWrite(RxTxData+1,'y');
            ioWrite(RxTxData,'.');
            ioWrite(RxTxData+1,'.');
            ioWrite(RxTxData,'.');
            ioWrite(RxTxData+1,'.');
            ioWrite(RxTxData,0x0a);
            ioWrite(RxTxData+1,0x0d);

}

void tx_echo()
{
    ////////////////////////
      // Transmit  Datagram //
      ////////////////////////
            unsigned char BusST0,BusST1;

      // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
            ioWrite(TxCmd,          0xc0);
            ioWrite(TxCmd + 1,      0x00);

      // 64 bytes to be sent
      // Ethernet Header(14) + IP Header(20) + UDP Header(8) +
Data(22)
            ioWrite(TxLength,   0x40);
            ioWrite(TxLength+1, 0x00);

      // Check Bus status (Data Sheet P.67)
            ioWrite(PPPtr,      0x38);
            ioWrite(PPPtr + 1, 0x01);

      // Check Bus status,Is ready for transmit (Check bit 7)
            do {
                  BusST0 = ioRead(PPData);
                  BusST1 = ioRead(PPData+1);
            }while(!(BusST1==0x01));

      //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

      ////////////////////////
      //  Ethernet Header  // (14 bytes)
      ////////////////////////

      // Send Destination (6 bytes)
      // MAC Address Microcomputer Top (00-08-02-f4-f8-5b) Loon(00-
0c-6e-a6-34-a6)
            ioWrite(RxTxData,       des_macaddr[0]);
            ioWrite(RxTxData+1, des_macaddr[1]);
            ioWrite(RxTxData, des_macaddr[2]);
            ioWrite(RxTxData+1,     des_macaddr[3]);
            ioWrite(RxTxData,       des_macaddr[4]);
            ioWrite(RxTxData+1,     des_macaddr[5]);

      // Send Source (6 bytes)
```

```
// MAC Address Embedded Device (00-04-e2-7a-75-71)
      ioWrite(RxTxData,       my_macaddr[0]);
      ioWrite(RxTxData+1, my_macaddr[1]);
      ioWrite(RxTxData, my_macaddr[2]);
      ioWrite(RxTxData+1,     my_macaddr[3]);
      ioWrite(RxTxData,       my_macaddr[4]);
      ioWrite(RxTxData+1,     my_macaddr[5]);

// Send Protocol Type (2 bytes)
// IP=0x0800
      ioWrite(RxTxData,       0x08);
      ioWrite(RxTxData+1, 0x00);


//////////////////
//  IP Header  // (20 bytes)
//////////////////

// Version Header Length (1 byte)
// Using IPv4
      ioWrite(RxTxData,       0x45);

// Service (1 byte)
// Normally set to zero because not used
      ioWrite(RxTxData+1, 0x00);

// Length (2 bytes)
// 50 byte Length (IP Header(20) + UDP Header(8) + Data(22))
      ioWrite(RxTxData,       0x00);
      ioWrite(RxTxData+1, 0x32);

// Identifier (2 bytes)
// 0x0000
      ioWrite(RxTxData,       0x00);
      ioWrite(RxTxData+1, 0x00);

// Flags Fragment offset (2 bytes)(no flag)
// 0x4000
      ioWrite(RxTxData, 0x40);
      ioWrite(RxTxData+1, 0x00);

// Time to Live (1 byte)
// 63
      ioWrite(RxTxData,       0x3f);

// Protocol (1 byte)
// UDP 17 = 0x11
      ioWrite(RxTxData+1, 0x11);

// Checksum IP Header (2 bytes)
// 0xd295
      ioWrite(RxTxData,       0xd2);
      ioWrite(RxTxData+1, 0x95);

// Source Address (4 bytes)
// IP address 161.246.18.156 (Embedded Device)
      ioWrite(RxTxData,       my_ipaddr[0]);   // 161
      ioWrite(RxTxData+1, my_ipaddr[1]);   // 246
      ioWrite(RxTxData,       my_ipaddr[2]);   // 18
      ioWrite(RxTxData+1, my_ipaddr[3]);   // 156
```

```
        // Destination Address (4 bytes)
        // IP address 161.246.18.157 (Labtop)
              ioWrite(RxTxData,       des_ipaddr[0]);   // 161
              ioWrite(RxTxData+1, des_ipaddr[1]);   // 246
              ioWrite(RxTxData,       des_ipaddr[2]);   // 18
              ioWrite(RxTxData+1, des_ipaddr[3]);   // 157

        //////////////////
        //  UDP Header // (8 bytes)
        //////////////////

        // Source Port (2 bytes)
        // Port 1300 (0514)
              ioWrite(RxTxData,       0x05);
              ioWrite(RxTxData+1, 0x14);

        // Destination Port (2 bytes)
        // Port 5000 (0x1388)
              ioWrite(RxTxData,       0x13);
              ioWrite(RxTxData+1, 0x88);

        // Message Length (2 bytes)
        // 30 bytes (UDP Header(8) + Data(22))
              ioWrite(RxTxData,       0x00);
              ioWrite(RxTxData+1, 0x1e);

        // Checksum (2 bytes)
        // 0x0000 (no checksum used)
              ioWrite(RxTxData,       0x00);
              ioWrite(RxTxData+1, 0x00);

        ////////////
        //  Data  // (22 bytes)
        ////////////

        // Write Message (22 bytes)
              ioWrite(RxTxData,RxBuffer[43]);
              ioWrite(RxTxData+1,RxBuffer[44]);
              ioWrite(RxTxData,RxBuffer[45]);
              ioWrite(RxTxData+1,RxBuffer[46]);
              ioWrite(RxTxData,RxBuffer[47]);
              ioWrite(RxTxData+1,RxBuffer[48]);
              ioWrite(RxTxData,RxBuffer[49]);
              ioWrite(RxTxData+1,RxBuffer[50]);
              ioWrite(RxTxData,RxBuffer[51]);
              ioWrite(RxTxData+1,RxBuffer[52]);
              ioWrite(RxTxData,RxBuffer[53]);
              ioWrite(RxTxData+1,RxBuffer[54]);
              ioWrite(RxTxData,RxBuffer[55]);
              ioWrite(RxTxData+1,RxBuffer[56]);
              ioWrite(RxTxData,RxBuffer[57]);
              ioWrite(RxTxData+1,RxBuffer[58]);
              ioWrite(RxTxData,RxBuffer[59]);
              ioWrite(RxTxData+1,RxBuffer[60]);
              ioWrite(RxTxData,RxBuffer[61]);
              ioWrite(RxTxData+1,RxBuffer[62]);
              ioWrite(RxTxData,RxBuffer[63]);
              ioWrite(RxTxData+1,RxBuffer[64]);

}
```

```
//
*****************************************************************************
******
// udp.h: defines for udp.c
//
*****************************************************************************
******

#ifndef H__UDP
#define H__UDP

void CheckPacket();
void txudp();
void tx_echo();
void tx_adc();
#endif
```

Sio.c

```
/*---------------------------------------------------------------------------
-----------
SIO.C:  Serial Communication Routines.

Copyright 1995-2002 KEIL Software, Inc.
---------------------------------------------------------------------------
---------*/

#include <reg52.h>
#include <string.h>
#include "sio.h"

/*---------------------------------------------------------------------------
-----------
Notes:

The length of the receive and transmit buffers must be a power of 2.

Each buffer has a next_in and a next_out index.

If next_in = next_out, the buffer is empty.

(next_in - next_out) % buffer_size = the number of characters in the
buffer.
---------------------------------------------------------------------------
---------*/
#define TBUF_SIZE   32          /*** Must be one of these powers of
2 (2,4,8,16,32,64,128) ***/
#define RBUF_SIZE   8           /*** Must be one of these powers of 2
(2,4,8,16,32,64,128) ***/

#define TBUF_SPACE  idata       /*** Memory space where the transmit
buffer resides ***/
#define RBUF_SPACE  idata       /*** Memory space where the receive
buffer resides ***/

#define CTRL_SPACE  data        /*** Memory space for the buffer
indexes ***/

/*---------------------------------------------------------------------------
-----------
```

```
------------------------------------------------------------------------
---------*/
#if TBUF_SIZE < 2
#error TBUF_SIZE is too small.  It must be larger than 1.
#elif TBUF_SIZE > 128
#error TBUF_SIZE is too large.  It must be smaller than 129.
#elif ((TBUF_SIZE & (TBUF_SIZE-1)) != 0)
#error TBUF_SIZE must be a power of 2.
#endif

#if RBUF_SIZE < 2
#error RBUF_SIZE is too small.  It must be larger than 1.
#elif RBUF_SIZE > 128
#error RBUF_SIZE is too large.  It must be smaller than 129.
#elif ((RBUF_SIZE & (RBUF_SIZE-1)) != 0)
#error RBUF_SIZE must be a power of 2.
#endif

/*------------------------------------------------------------------------
-----------
------------------------------------------------------------------------
---------*/
static TBUF_SPACE unsigned char tbuf [TBUF_SIZE];
static RBUF_SPACE unsigned char rbuf [RBUF_SIZE];

static CTRL_SPACE unsigned char t_in = 0;
static CTRL_SPACE unsigned char t_out = 0;

static CTRL_SPACE unsigned char r_in = 0;
static CTRL_SPACE unsigned char r_out = 0;

static bit ti_restart = 0;  /* NZ if TI=1 is required */


/*------------------------------------------------------------------------
-----------
------------------------------------------------------------------------
---------*/
static void ser_isr (void) interrupt 4
{
/*-------------------------------------------------
Received data interrupt.
-------------------------------------------------*/
if (RI != 0)
  {
  RI = 0;

  if (((r_in - r_out) & ~(RBUF_SIZE-1)) == 0)
    {
    rbuf [r_in & (RBUF_SIZE-1)] = SBUF;
    r_in++;
    }
  }

/*-------------------------------------------------
Transmitted data interrupt.
-------------------------------------------------*/
if (TI != 0)
  {
  TI = 0;
```

```
  if (t_in != t_out)
    {
    SBUF = tbuf [t_out & (TBUF_SIZE-1)];
    t_out++;
    ti_restart = 0;
    }
  else
    {
    ti_restart = 1;
    }
  }

}

/*-------------------------------------------------------------------------
-----------
-------------------------------------------------------------------------
---------*/
#pragma disable

void ser_init(void)
{
/*-----------------------------------------------
Setup TIMER1 to generate the proper baud rate.
-----------------------------------------------*/
ser_baudrate ();// 9600 bps

/*-----------------------------------------------
Clear com buffer indexes.
-----------------------------------------------*/
t_in = 0;
t_out = 0;

r_in = 0;
r_out = 0;

/*-----------------------------------------------
Setup serial port registers.
-----------------------------------------------*/
SM0 = 0; SM1 = 1;   /* serial port MODE 1 */
SM2 = 0;
REN = 1;            /* enable serial receiver */

RI = 0;             /* clear receiver interrupt */
TI = 0;             /* clear transmit interrupt */
ti_restart = 1;

ES = 1;             /* enable serial interrupts */
PS = 0;             /* set serial interrupts to low priority */
}

/*-------------------------------------------------------------------------
-----------
-------------------------------------------------------------------------
---------*/
#pragma disable

void ser_baudrate ()
{
/*-----------------------------------------------
Clear transmit interrupt and buffer.
```

```
-----------------------------------------------*/
      TI = 0;               /* clear transmit interrupt */
      t_in = 0;             /* empty transmit buffer */
      t_out = 0;

/*-----------------------------------------------
Set timer 1 up as a baud rate generator.
-----------------------------------------------*/
      TR1 = 0;              /* stop timer 1 */
      ET1 = 0;              /* disable timer 1 interrupt */

      PCON |= 0x80;         /* 0x80=SMOD: set serial baudrate doubler
*/

      TMOD &= ~0xF0;        /* clear timer 1 mode bits */
      TMOD |= 0x20;         /* put timer 1 into MODE 2 */

      //TH1 = (unsigned char) (256 - (XTAL / (16L * 12L *
baudrate)));
      TH1 = 0xFA;

      TR1 = 1;              /* start timer 1 */
}

/*-----------------------------------------------------------------------
-----------
-----------------------------------------------------------------------
---------*/
#pragma disable

char ser_putchar (
  unsigned char c)
{
/*-----------------------------------------------
If the buffer is full, return an error value.
-----------------------------------------------*/
if (ser_tbuflen () >= TBUF_SIZE)
  return (-1);

/*-----------------------------------------------
Add the data to the transmit buffer.  If the
transmit interrupt is disabled, then enable it.
-----------------------------------------------*/
      tbuf [t_in & (TBUF_SIZE - 1)] = c;
      t_in++;

      if (ti_restart)
      {
            ti_restart = 0;
            TI = 1;               /* generate transmit interrupt */
      }

return (0);
}

/*-----------------------------------------------------------------------
-----------
-----------------------------------------------------------------------
---------*/
#pragma disable
```

```
int ser_getchar (void)
{
      if (ser_rbuflen () == 0)
      return (-1);

      return (rbuf [(r_out++) & (RBUF_SIZE - 1)]);
}

/*-------------------------------------------------------------------
----------
-------------------------------------------------------------------
---------*/
#pragma disable

unsigned char ser_rbuflen (void)
{
      return (r_in - r_out);
}

/*-------------------------------------------------------------------
----------
-------------------------------------------------------------------
---------*/
#pragma disable

unsigned char ser_tbuflen (void)
{
      return (t_in - t_out);
}

void ser_send(unsigned char *buf)
{
      unsigned char i;
      unsigned char len = strlen(buf);
      for(i=0;i<len;i++)
      ser_putchar(buf[i]);
}
/*-------------------------------------------------------------------
----------
-------------------------------------------------------------------
---------*/

//
*******************************************************************
******
// sio.h:
//
*******************************************************************
******

#ifndef H__SIO
#define H__SIO

void ser_init(void);

void ser_baudrate (void);

char ser_putchar (
  unsigned char c);

int ser_getchar (void);
```

```
unsigned char ser_rbuflen (void);

unsigned char ser_tbuflen (void);

void ser_send(unsigned char *buf);

#endif
```

Until.c

```
//--------------------//
//-------------------------------------------------------------------------
-----
// This function converts an integer to an ASCII string.  It is a
// normally provided as a standard library function but the Keil
// libraries do not include it.  Caution: The string passed to this
// must be at least 12 bytes long
//-------------------------------------------------------------------------
-----
char *itoa(unsigned int value, char * buf, unsigned char radix)
{
      unsigned int i;
      char *ptr;
      char *temphold;

      temphold = buf;
      ptr = buf + 12;
      *--ptr = 0;         // Insert NULL char
      do
      {
         // First create string in reverse order
         i = (value % radix) + 0x30;
            if(i > 0x39) i += 7;
            *--ptr = i;
      value = value / radix;
      } while(value != 0);

      // Next, move the string 6 places to the left
      // Include NULL character
      for( ; (*buf++ = *ptr++); );
      return(temphold);
}
```

Until.h

```
#ifndef H__UTILS
#define H__UTILS

char * itoa(unsigned int value, char buf[], unsigned char radix);

#endif
```

Adc.c

```
#include "adc.h"
#include "cs8900.h"


#define ADC_Data P1
sbit ADC_RD = P3^3;
```

```
sbit ADC_WR = P3^4;

unsigned char adc_read(void)
{
      ADC_WR = 0;
      ADC_RD = 1;
      ADC_WR = 1;
      DELAY1ms(1);
      ADC_RD = 0;
      return ADC_Data;
}
//
*************************************************************************
******
// adc.h
//
*************************************************************************
******

#ifndef H_ADC
#define H_ADC

unsigned char adc_read(void);

#endif
```