ระบบวัดผ่านเครือข่ายคอมพิวเตอร์

นาย นฤพล พวงแก้ว

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

Measurement System On Computer Network

Mr. Narupon Poungkaew

A Special Project Submitted in Partial Fulfillment of the Requirement for the Degree of

Bachelor of Science

Department of Applied Physics

Faculty of Science

King Mongkut's Institute of Technology Ladkrabang

Academic Year 2002

โครงงานพิเศษเรื่อง ระบบวัดผ่านเครือข่ายคอมพิวเตอร์

นักศึกษา นายนฤพล พวงแก้ว
ภาควิชา ฟิสิกส์ประยุกต์
สาขาวิชา ฟิสิกส์ประยุกต์
อาจารย์ที่ปรึกษา รองศาสตราจารย์วิชิต ศิริโชติ

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

คณะกรรมการตรวจสอบ	ลายมือชื่อ
ประธานกรรมการ ผศ.วิชาญ เตชิตธีระ	.....
กรรมการ รศ.วิชิต ศิริโชติ	.....
กรรมการ อ.ธนภรณ์ ลีลาวัฒนานนท์	.....
กรรมการ อ.บดินทร์ ดำรงศักดิ์	.....

.....
(ผู้ช่วยศาสตราจารย์วิชาญ เตชิตธีระ)
หัวหน้าภาควิชา

โครงงานพิเศษ	ระบบวัดผ่านเครือข่ายคอมพิวเตอร์
นักศึกษา	นายนฤพล พวงแก้ว
ภาควิชา	ฟิสิกส์ประยุกต์ **คณะวิทยาศาสตร์**
สาขาวิชา	ฟิสิกส์ประยุกต์
ปีการศึกษา	2545
อาจารย์ที่ปรึกษา	รองศาสตราจารย์วิชิต ศิริโชติ

บทคัดย่อ

โครงงานพิเศษนี้เป็นการออกแบบและสร้างระบบฝังตัวชนิด เอสเอ็นเอ็มพีเอเจนต์ อย่างง่าย โดยฮาร์ดแวร์ที่ใช้คือ ไมโครคอนโทรลเลอร์ เอ็มซีเอส 51 เชื่อมต่อกับ การ์ดแลนชนิดเทนเบสที ประกอบกับเซนเซอร์อุณหภูมิและความชื้น ซึ่งเซนเซอร์แต่ละตัวจะมี โอ ไอดี ของตัวเอง สามารถเข้าได้กับ เอสเอ็นเอ็มพี เวอร์ชั่น 1 ตัวโปรโตคอลของเอเจนต์นี้เขียนโดยใช้ โคดภาษาซี โปรโตคอลที่ใช้ได้คือ เออาร์พี ไอซีเอ็มพี ยูดีพี และ เอสเอ็นเอ็มพี เราได้พัฒนาโปรแกรมซึ่งทำงานบนเครื่องคอมพิวเตอร์เพื่อใช้สำหรับ รับ-ส่ง เอสเอนเอ็มพีแพคเก็ต นอกจากที่เครื่องสามารถอ่านค่าอุณหภูมิและความชื้นผ่านทางระบบอินเตอร์เน็ตได้แล้วยังสามารถบันทึกค่าอุณหภูมิและความชื้นกับเวลาได้อีกด้วย

Special Project Title	Measurement System on Computer Network
Name	Narupon Poungkaew
Department	Applied Physics
Program	Applied Physics
Academic Year	2002
Special Project Advisor	Assoc.Prof.Wichit Sirichote

ABTRACT

A simple embedded SNMP agent has been designed and built. The agent's hardware is based on using the MCS51 controller interfaced with a 10baseT LAN card. The temperature and humidity sensors are provided. Each sensor has its own OID and compatible with SNMP version 1. The protocol stack on the agent was written using c coding. The available protocols are ARP, ICMP, UDP and SNMP. We have developed the GUI software running on PC for sending and receiving SNMP packet. In addition to reading the value over the net, the agent can also record temperature and humidify with time as well.

กิตติกรรมประกาศ

โครงงานพิเศษนี้เสร็จลุลวงไปได้ด้วยดี ต้องขอขอบคุณ รองศาสตราจารย์วิชิต ศิริโชติ อาจารย์ที่ปรึกษาโครงงานพิเศษนี้ โดยให้ทั้งความรู้ คำปรึกษาและข้อเสนอแนะในการแก้ปัญหาต่างๆ อย่างสม่ำเสมอ อีกทั้งยังมีบิดา มารดา ที่มีความห่วงใยและให้การดูแลตลอดมา นอกจากนี้ยังมีอาจารย์อีกหลายท่านซึ่งไม่อาจกล่าวในที่นี้ได้หมด ที่ให้ความรู้ตั้งแต่ขั้นพื้นฐานง่ายๆจนถึงขั้นที่สามารถนำเอาไปประยุกต์ใช้งานได้ ทำให้ผู้ทำโครงงานพิเศษนี้มีความรู้สะสมเรื่อยมาจนถึงปัจจุบัน ทำให้มีโอกาสสร้างโครงงานพิเศษนี้ขึ้นมาได้

นฤพล พวงแก้ว

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูป

หน้า

สารบัญรูป(ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงงานพิเศษ

เครื่องมือวัดส่วน ใหญ่นั้นมีความสามารถในการวัดค่าหรือตรวจสอบค่าได้ในเฉพาะพื้นที่ที่เครื่องมือนั้นติดตั้งอยู่ และ ต้องมีมนุษย์ผู้ทำการเก็บรวบรวมข้อมูลจากเครื่องมือวัดนั้นโดยตรง จึงทำให้ไม่ได้รับความสะดวกในการทำงานบางอย่างเช่น การบันทึกค่ามิเตอร์วัดไฟฟ้าตามบ้านเรือน การเฝ้าดูค่าความชื้นภายในโกดังเก็บอาหาร การเฝ้าดูค่าอุณหภูมิของห้องทำความเย็น เป็นต้น ทำให้เสียเวลา และ ค่าใช้จ่ายสูง ในการที่จะได้ข้อมูลเหล่านั้นมาวิเคราะห์ ใช้งาน จึงเกิดแนวคิดที่ จะนำระบบเน็ตเวิร์กคอมพิวเตอร์ หรือ เครือข่ายอินเตอร์เน็ต ที่ใช้กับอย่างแพร่หลายมากในปัจจุบันนี้ มาใช้กับเครื่องมือวัด เพื่อจะทำให้ เครื่องมือวัด มีสามารถในการรายงานผลของการวัดได้จากระยะไกล โดยที่มนุษย์ไม่จำเป็นต้องเป็นผู้ อ่านค่าจากเครื่องมือวัดนั้นโดยตรง ถึงแม้ว่าการนำเครื่องคอมพิวเตอร์มาเป็นผู้ทำหน้าที่ติดต่อกับเครื่องมือวัดโดยตรงนั้น จะทำให้ได้รับความสะดวกในการเก็บรวบรวมข้อมูลก็จริง แต่ก็ยังคงเกิดปัญหาเรื่องค่าใช้จ่ายสูง ทั้งด้าน hardware และ software ทำให้ไม่อาจนำเครื่องคอมพิวเตอร์มาใช้ในงานเหล่านี้ได้ หลังจากได้ทำการศึกษาเกี่ยวกับ ไมโครคอนโทรเลอร์ และ เน็ตเวิร์กคอมพิวเตอร์ จึงมีแนวคิดอีกว่า ควรใช้ไมโครคอนโทรเลอร์เป็นผู้ติดต่อโดยตรงกับเครื่องมือวัดแล้วส่งข้อมูลนั้น ผ่านเครือข่ายเน็ตเวิร์กคอมพิวเตอร์ ไปที่ เครื่องคอมพิวเตอร์อีกทีหนึ่ง เพื่อลดต้นทุน และ ค่าใช้จ่ายในการดูแลรักษา ง่ายต่อการติดต่อกับเครื่องมือวัดเพราะเครือข่ายเน็ตเวิร์กคอมพิวเตอร์เป็นระบบที่มีการใช้งานกันอย่างแพร่หลายอยู่แล้ว ถึงอีกทั้งยังสามารถต่อเข้ากับเครื่องมือวัดที่ติดตั้งอยู่ที่ใดก็ได้ที่มีเครือข่ายเน็ตเวิร์กอยู่ด้วย

1.2 วัตถุประสงค์

1. เพื่อศึกษาโครงสร้างและการทำงานขั้นพื้นฐานของการสื่อสารผ่านเครือข่ายอินเตอร์เน็ต
2. เพื่อศึกษาการประยุกต์ นำเอาไมโครคอนโทรเลอร์ ตระกูล MCS51 ไปใช้งานได้จริง
3. เพื่อฝึกทักษะในการออกแบบทาง hardware และ software
4. เพื่อสร้างอุปกรณ์ที่สามารถส่งข้อมูลผ่านเครือข่ายเน็ตเวิร์กคอมพิวเตอร์ ที่มีราคาถูก และมีขนาดเล็ก

1.3 ขอบเขตของการวิจัย

1. นำไมโครคอนโทรเลอร์ตระกูล MCS51 มาใช้ในการเชื่อมต่อระหว่าง เครื่องมือวัด กับ เครือข่ายเน็ตเวิร์กคอมพิวเตอร์

2. ผู้ใช้สามารถ อ่านข้อมูลจากเครื่องวัดโดยผ่าน Simple Network Management Protocol

1.4 ขั้นตอนการวิจัยและวิธีการดำเนินงาน

1. รวบรวมข้อมูลของการใช้งานไมโครคอนโทรเลอร์ตระกูล MCS51
2. ศึกษาการทำงานของ Ethernet controller ร่วมกับ ไมโครคอนโทรเลอร์
3. ศึกษาการทำงานขั้นพื้นฐานของเครือข่ายเน็ตเวิร์กคอมพิวเตอร์
4. ศึกษาการทำงานของ Simple Network Management Protocol
5. ออกแบบวงจรเชื่อมต่อระหว่าง Ethernet Card กับ ไมโครคอนโทรเลอร์ตระกูล MCS51
6. ออกแบบโปรแกรม ติดต่อระหว่าง Ethernet Card กับ ไมโครคอนโทรเลอร์ตระกูล MCS51
7. ออกแบบโปรแกรมติดต่อระหว่าง ไมโครคอนโทรเลอร์ตระกูล MCS51 กับ เครื่องคอมพิวเตอร์โดยผ่าน โปรโตคอล SNMP
8. ศึกษาการเชื่อมต่อระหว่าง เครื่องมือวัด กับ ไมโครคอนโทรเลอร์ตระกูล MCS51
9. ออกแบบวงจรเชื่อมต่อระหว่าง เครื่องมือวัด กับ ไมโครคอนโทรเลอร์ตระกูล MCS51
10. ออกแบบโปรแกรมติดต่อระหว่างเครื่องมือวัด กับ ไมโครคอนโทรเลอร์ตระกูล MCS51
11. สร้างวงจรรวม
12. เขียนโปรแกรมหลักควบคุมการทำงานทั้งระบบ
13. ทดสอบการทำงานของเครื่องมือวัด ผ่านอุปกรณ์ที่สร้างขึ้น

1.5 ผลที่คาดว่าจะได้รับ

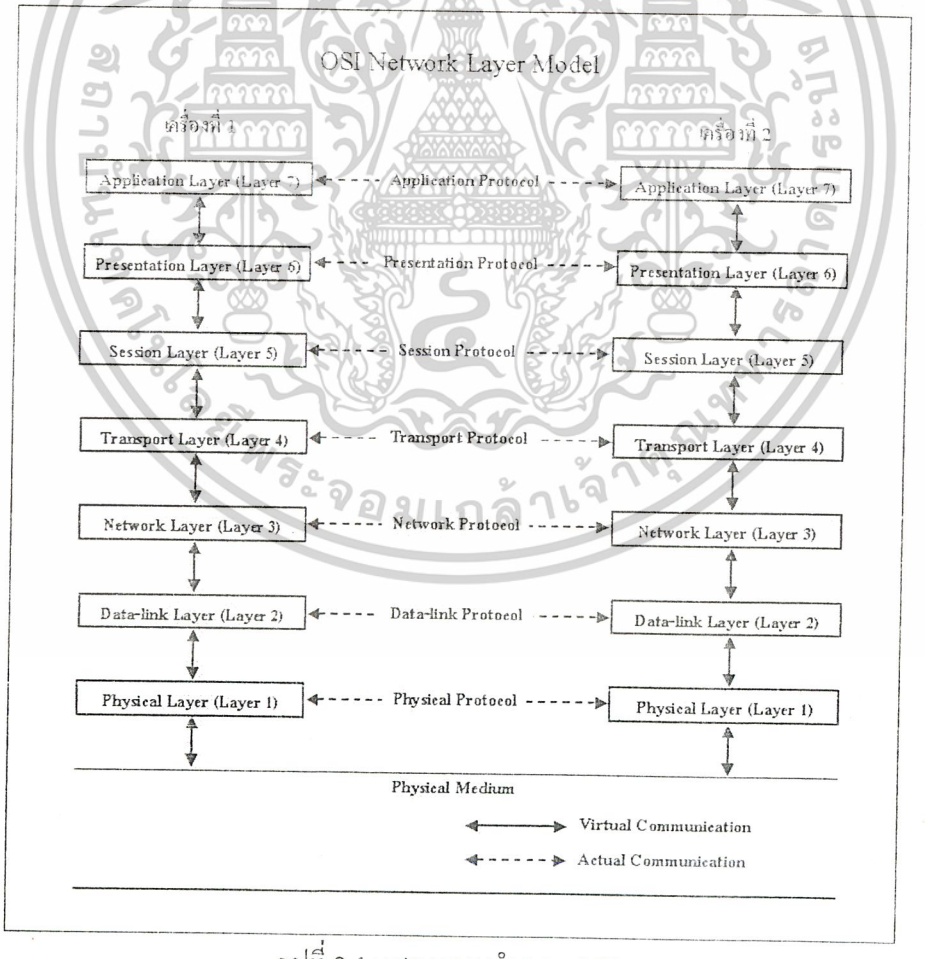
1. ได้รับความรู้เรื่องการทำงานขั้นพื้นฐานของการสื่อสารผ่านเครือข่ายอินเตอร์เนต
2. ได้ฝึกทักษะในการประยุกต์ใช้ ไมโครคอนโทรเลอร์ตระกูล MCS51
3. ได้ฝึกทักษะในการออกแบบทาง hardware และ software
4. ได้ต้นแบบของอุปกรณ์ที่สามารถส่งข้อมูลผ่านเครือข่ายคอมพิวเตอร์เน็ตเวิร์ค ที่มีราคาถูก มีขนาดเล็ก และ ใช้งานได้อย่างสะดวก รวดเร็ว

บทที่ 2
ทฤษฎีและหลักการ

2.1 แบบจำลองของ OSI (The OSI Reference model)

ระบบการสื่อสารข้อมูลในเครือข่ายคอมพิวเตอร์ประกอบด้วยทั้ง ฮาร์ดแวร์และซอฟต์แวร์ที่ซับซ้อน การมองภาพของระบบโดยรวมทั้งหมดเป็นหน่วยใหญ่ย่อมยากต่อการทำความเข้าใจ การใช้แบบอ้างอิงที่แบ่งระบบออกเป็นส่วนย่อยจะช่วยลดความซับซ้อนและสร้างความเข้าใจได้ง่ายกว่า

เพื่อที่จะอธิบายเกี่ยวกับสถาปัตยกรรมของเครือข่ายคอมพิวเตอร์ รวมทั้งแสดงถึงตัวอย่างของการออกแบบเลเยอร์และรูปแบบของโปรโตคอลให้เข้าใจได้อย่างละเอียดนั้น จะต้องมีการพัฒนาแบบจำลอง (model) ของเน็ตเวิร์กขึ้นมาช่วยในการอธิบายดังกล่าว แบบจำลองนี้ได้ถูกพัฒนาโดยองค์การมาตรฐานสากล หรือ International Standard Organization (ISO) และเรียกชื่อว่ามาตรฐาน OSI (Open System Interconnection) ซึ่งมีทั้งหมด 7 ชั้นหรือเลเยอร์



รูปที่ 2.1 แสดงแบบจำลอง OSI

หลักในการออกแบบเลเยอร์คือ

1. เลเยอร์จะถูกกำหนดขึ้นมาเมื่อมีข้อแตกต่างทางด้านแนวคิด (abstraction) ในการทำงาน

2. แต่ละเลเยอร์จะมีการกำหนดกลไกการทำงานภายในอย่างละเอียด

3. ฟังก์ชันภายในเลเยอร์จะพยายามมุ่งไปสู่ระดับมาตรฐานของโปรโตคอล

4. ขอบเขตของเลเยอร์จะถูกเลือกและจำกัด ทั้งนี้เพื่อให้มีปริมาณของข้อมูลหรือการเชื่อมต่อระหว่างเลเยอร์ให้น้อยที่สุด

5. จำนวนของเลเยอร์จะต้องมากพอที่จะทำให้ฟังก์ชันที่จำเป็นและแตกต่างกันไม่อยู่ในเลเยอร์เดียวกัน และในขณะเดียวกันจำนวนของเลเยอร์ก็จะต้องไม่มากจนเกินไป

รายละเอียดของแต่ละเลเยอร์เป็นดังนี้

เลเยอร์ที่ 1 Physical layer เลเยอร์นี้จะเกี่ยวข้องกับการส่งข้อมูลระดับบิตผ่านช่องทางการสื่อสารข้อมูลโดยการออกแบบจะต้องแน่ใจว่าจะสามารถส่งข้อมูลออกไปได้ และปลายทางก็จะต้องรับข้อมูลนั้นได้อย่างถูกต้องด้วย ส่วนใหญ่จะเป็นข้อกำหนดเกี่ยวกับเรื่องของแรงดันไฟฟ้าว่าจะต้องใช้แรงดันเท่าไรสำหรับแทนเลข "1" และเท่าใดสำหรับแทนเลข "0" ระยะเวลาในการส่งแต่ละบิตจะต้องห่างกันเท่าใด รวมถึงรูปแบบของหัวต่อหรือ connector ทั้งนี้วิศวกรไฟฟ้าคือ ผู้ที่จะเข้ามาเกี่ยวข้องโดยตรงกับเลเยอร์ลำดับนี้

เลเยอร์ที่ 2 Data-link layer จุดประสงค์หลักของเลเยอร์นี้ คือ จะควบคุมการส่งข้อมูลดับให้ไม่มีข้อผิดพลาดใดๆ เกิดขึ้น ซึ่งจะทำให้เลเยอร์ในลำดับถัดไปไม่ต้องมาคอยสนใจในเรื่องนี้อีก วิธีการก็คือจะต้องแตกข้อมูลออกเป็นก้อนๆ เรียกว่า เฟรมข้อมูล (data frame) แล้วส่งออกไปทีละชุด จากนั้นก็รอคอยให้มีการตอบรับ (acknowledge frame) กลับมา ในกรณีที่มีสัญญาณรบกวนหรือสัญญาณขาดหายไป จะต้องมีการบอกให้เครื่องต้นทางส่งข้อมูลที่หายไปนั้นกลับมาให้ใหม่ นอกจากนี้ยังจะต้องมีการพักข้อมูลไว้ในบัฟเฟอร์ หากความเร็วในการส่งข้อมูลทั้งสองฝ่ายไม่เท่ากัน

เลเยอร์ที่ 3 Network layer หน้าที่หลักของเลเยอร์นี้จะเกี่ยวข้องกับการหาเส้นทาง (route) ในการส่ง packet จากต้นทางไปยังปลายทาง การกำหนดเส้นทางนี้อาจจะกำหนดตั้งแต่เริ่มต้นติดต่อเลย หรือจะเปลี่ยนแปลงตลอดเวลา (dynamic) ก็ได้ อย่างในกรณีที่มีการส่งข้อมูลข้ามเน็ตเวิร์กที่มีความแตกต่างระหว่างกัน หรือใช้โปรโตคอลที่แตกต่างกัน Network layer จะต้องจัดการกับปัญหาดังกล่าวเพื่อให้แต่ละเน็ตเวิร์คสามารถเชื่อมต่อกันได้เสมือนเป็นเน็ตเวิร์คเดียวกัน

เลเยอร์ที่ 4 Transport layer เลเยอร์นี้จะคอยติดต่อกับเลเยอร์ที่อยู่ถัดขึ้นไป (session layer) เพื่อคอยแยกข้อมูลให้มีขนาดพอเหมาะ จากนั้นก็ส่งต่อให้กับ network layer พร้อมทั้งตรวจสอบว่า

ข้อมูลถูกส่งออกไปยังปลายทางเรียบร้อยหรือไม่ โดยเลเยอร์นี้จะต้องจัดการอย่างมีประสิทธิภาพ เพื่อแยก session layer ให้เป็นอิสระจากการเปลี่ยนแปลงทางด้านฮาร์ดแวร์

เลเยอร์ที่ 5 Session layer เลเยอร์นี้นอกจากจะมีหน้าที่ส่งข้อมูลแบบเดียวกับ transport layer แล้วยังมีให้บริการอื่นๆ อีกอย่างเช่น การยอมให้ผู้ใช้งานเข้าไปใช้งานยังเครื่องที่อยู่ห่างไกลออกไป (remote login) หรือถ่ายโอนไฟล์ระหว่างเครื่อง และยังทำหน้าที่เกี่ยวกับการซิงโครไนซ์เซชั่น (synchronization) หรือการทำให้สองระบบทำงานสัมพันธ์กันอีกด้วย

เลเยอร์ที่ 6 Presentation layer เลเยอร์นี้จะสนใจในเรื่องรูปแบบของข้อมูล เช่น การเปลี่ยนรหัสข้ามจากรหัส ASCII เป็น EBCDIC เพื่อให้คอมพิวเตอร์ที่มีการแทนรหัสต่างกันสามารถสื่อสารกันได้ นอกจากนี้ยังทำการลดขนาดของข้อมูล (data compression) หรือการเข้ารหัสข้อมูล (data encryption) เพื่อป้องกันการ โจรกรรมข้อมูลด้วย

เลเยอร์ที่ 7 Application layer เป็นเลเยอร์บนสุดที่ทำงานเกี่ยวข้องกับแอพพลิเคชั่นหลาย ๆ แบบ ซึ่งจะมีลักษณะการใช้งานที่แตกต่างกันออกไป ตัวอย่าง เช่น การควบคุมเทอร์มินอล ชนิดต่างๆ ซึ่งมีรูปแบบการแสดงผลบนจอภาพที่แตกต่างกัน ก็อาจจะมีการกำหนดลักษณะหรือรูปแบบของเทอร์มินอลเสมือน (Virtual terminal) ขึ้นมาเพื่อเป็นมาตรฐานกลางที่โปรแกรมจะติดต่อกับเทอร์มินอล (คล้ายกับ Java bytecode ที่เป็นรหัสกลางสำหรับ Java compiler จะทำการตีความให้สามารถทำงานที่เครื่องต่างชนิดกันได้)

2.2 Address Resolution Protocol

Address Resolution Protocol หรือ ARP เป็น โปรโตคอลที่ออกแบบมาเพื่อใช้ในเครือข่ายที่สนับสนุน Broadcast หน้าที่ของARPในเครือข่าย คือ ช่วยแปลง IP Address ไปสู่ Hardware Address เพื่อให้สถานีสามารถนำ Hardware Address ไปสร้างเฟรมระดับ Data link ได้

การสื่อสารระดับล่างระหว่างสถานีในเครือข่ายจะใช้ฮาร์ดแวร์ซึ่งกำหนดอยู่ในการ์ดอินเทอร์เฟส ดีไวซ์ไดรเวอร์ในสถานีต้นทางจะต้องทราบ Hardware Addressของสถานีปลายทางในเครือข่ายเพื่อนำมาใช้สร้างเฟรม Data link

การหา Hardware Address อาศัยการค้นจากตารางซึ่งเก็บคู่ของ IP Address และ Hardware Address การสร้างตารางและการค้นหามีรูปแบบที่กำหนดด้วยโปรโตคอล ARP ตารางที่เก็บ คู่แอดเดรสเรียกว่า ARP Table

2.2.1 หลักการทำงานของARP

สถานีต้นทางที่ต้องการส่งแพ็กเก็ตไปยังสถานีปลายทางจะเปิดตารางหาค่า Hardware Address ที่ตรงกับ IP Address จาก ตาราง ARP หากไม่พบค่า สถานีจะสร้างเฟรม ARP และ Broadcast เดทาแกรมร้องขอ (ARP request) โดยใส่ IP Address ของสถานีที่ต้องการถามหา Hardware Address การ Broadcast เดทาแกรมร้องขอทุก สถานีเครือข่ายเมื่อได้รับเฟรม Broadcast จะตรวจสอบ IP Address ประจำตัวกับ IP Address กับ IP Address ที่ร้องขอ หากพบว่าตรงกันสถานีนั้นก็จะส่ง เดทาแกรมตอบรับ (ARP reply) โดยส่ง Hardware Address กลับไปยังสถานีที่ร้องขอ สถานีต้นทางที่ได้รับคำตอบก็จะนำ Hardware Address ไปใช้งานและเก็บเข้าตาราง ในขณะเดียวกันสถานีที่ตอบรับก็จะนำค่าไอพีและ Hardware Address ของสถานีที่ร้องขอเก็บเข้าตารางเช่นกัน เพื่อใช้งานในโอกาสต่อไป

2.2.2 ARP Datagram

เนื่องจากARPทำงานในระดับ Data link เดทาแกรมของ ARP ก็จะบรรจุอยู่ในเฟรม Data link อย่าง เช่น อีเทอร์เน็ตโดยมีค่าประจำโปรโตคอลเท่ากับ 0x0806 แต่ละฟิลด์มีความหมายดังนี้

1. hardware 16 บิต : กำหนดชนิดของฮาร์ดแวร์เครือข่ายที่ ARP ทำงานอยู่ ค่าที่ใช้งานมีดังตัวอย่างต่อไปนี้

1 อีเทอร์เน็ต
4 โทเล็นริง
5 เคออส (Chaos)
6 เครือข่าย IEEE 802
7 อาร์คเน็ต
12 โลคัลทอล์ค

2. protocol 16 บิต : ชนิดของโปรโตคอลที่ร้องขอใช้ ARP

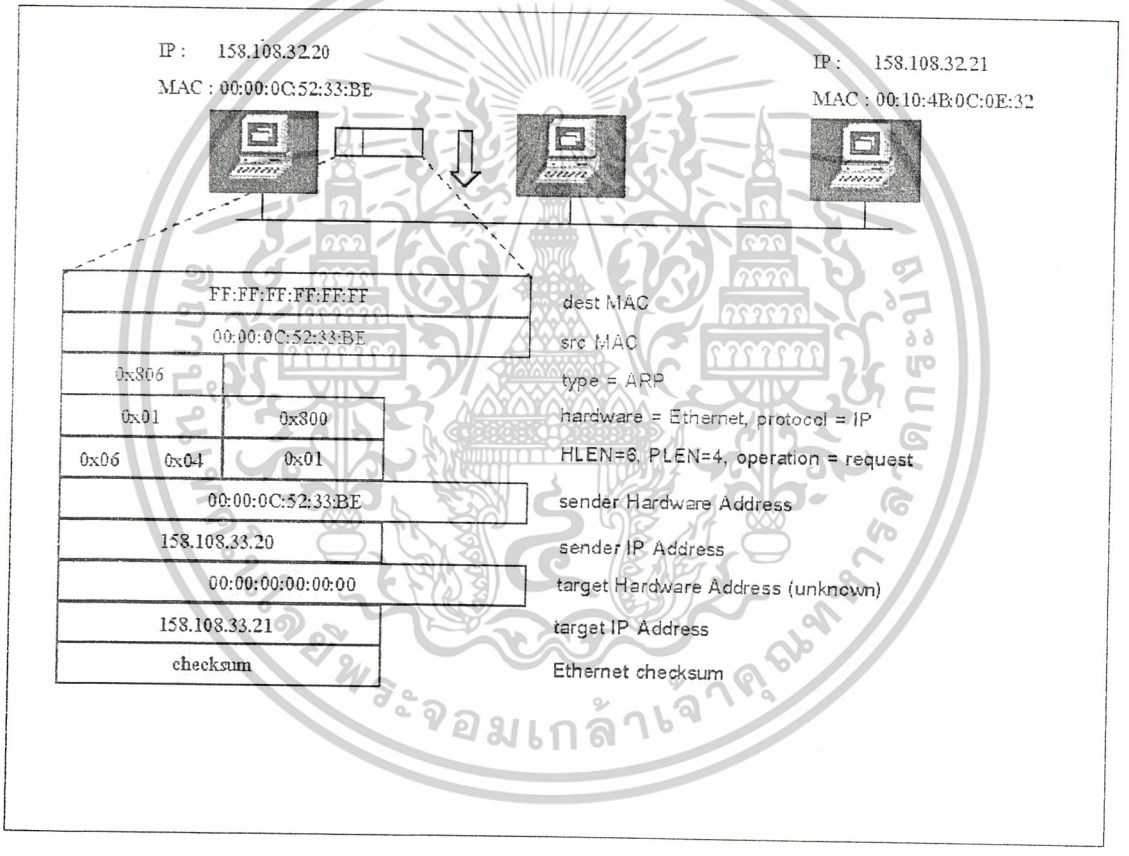
3. HLEN 8 บิต : ขนาดของ Hardware Address เป็นจำนวนไบต์ ค่าปกติที่ใช้งาน คือ 6 ซึ่งเท่ากับขนาด 6 ไบต์ของอีเทอร์เน็ต Hardware Address

4. PLEN 8 บิต : ขนาดของแอดเดรสระดับเน็ตเวิร์คเป็นจำนวนไบต์ ค่าปกติที่ใช้ คือ 4 ซึ่งเท่ากับขนาด 4 ไบต์ของ IP Address

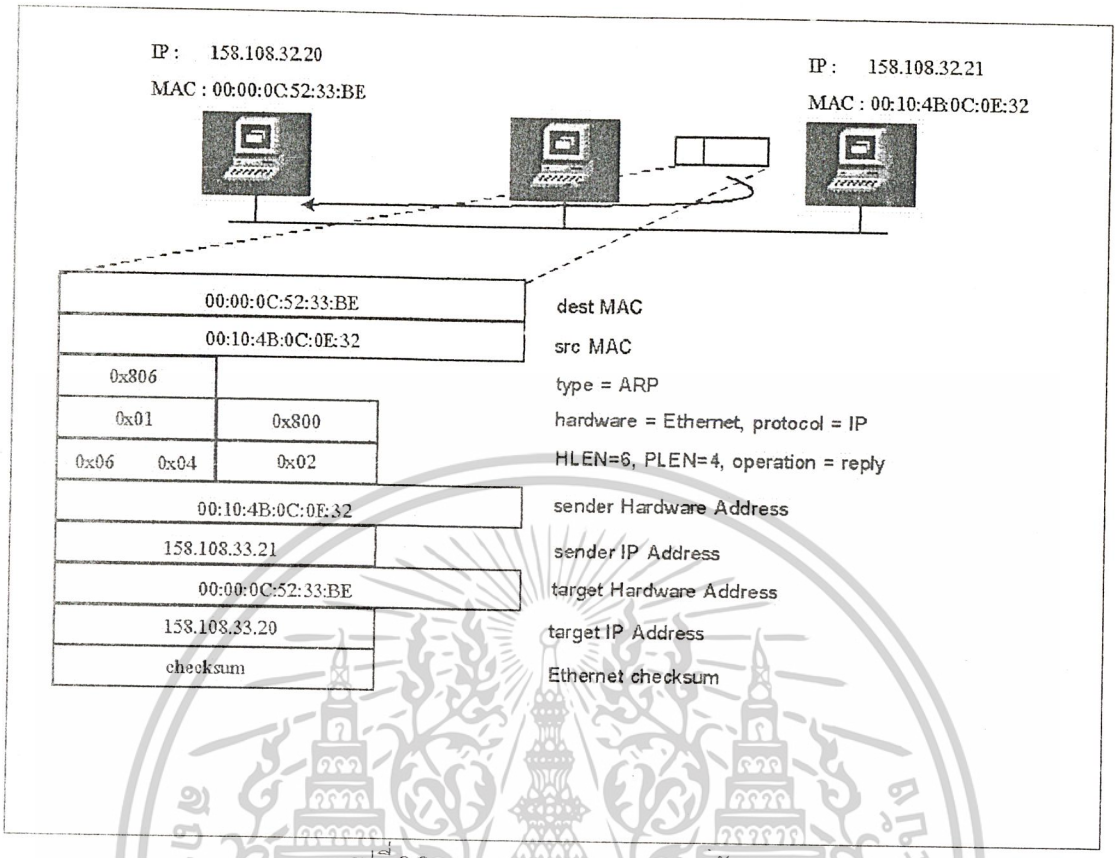
5. Operation 16 บิต : กำหนดรูปแบบการใช้เดทาแกรม ค่าในฟิลด์นี้ใช้กำหนดการทำงานของทั้ง ARP และ RARP ซึ่งมี 4 ค่าคือ

- ARP request (ค่าเท่ากับ 1)
- ARP reply (ค่าเท่ากับ 2)
- RARP request (ค่าเท่ากับ 3)
- RARP reply (ค่าเท่ากับ 4)

6. Address : ฟิลด์แอดเดรสเรียงลำดับจากฮาร์ดแวร์และเน็ตเวิร์คแอดเดรสของสถานที่ร้องขอ ตามด้วย Hardware Address ของสถานีที่ตอบรับ



รูปที่ 2.2 แสดงเดทาแกรมARPร้องขอ



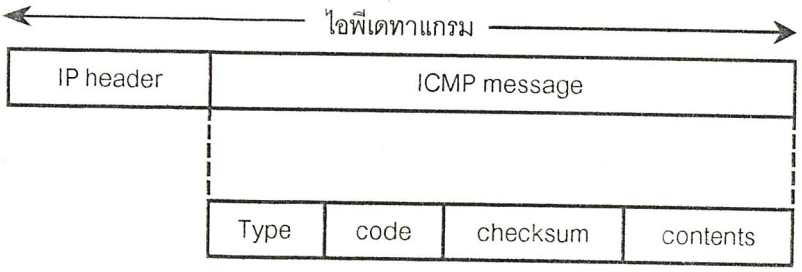
รูปที่ 2.3 แสดงเดทาแกรมARPตอบกลับ

2.3 Internet Control Message Protocol

Internet Control Message Protocol หรือ ICMP เป็นโปรโตคอลที่ใช้แจ้งข้อผิดพลาดและปัญหาที่เกิดจากการลำเลียงไอพีเดทาแกรม ภายในแพ็กเก็ต ICMP จะบรรจุข่าวสารบ่งบอกถึงข้อผิดพลาดและปัญหาที่เกิดขึ้น

2.3.1 แพ็กเก็ต ICMP

ICMP จะใช้ไอพีเดทาแกรมสำหรับแจ้งข่าวสาร ข้อมูลของไอพีเดทาแกรมก็คือ ข่าวสารที่บ่งบอกถึงปัญหาข้อผิดพลาด



รูปที่ 2.4 แสดงข่าวสาร ICMP บรรจุอยู่ในไอพี

ICMP จัดรูปแบบการรายงานข้อผิดพลาดโดยแบ่งฟิลด์ของข้อมูลในเดทาแกรมออกเป็น 4 ฟิลด์ แต่ละส่วนมีความหมายดังนี้

1. Type ขนาด 8 บิต : กำหนดทั้งค่าความผิดพลาดและการรายงานสถานะ การใช้งานในปัจจุบันมีทั้งหมด 15 ประเภทตามตารางที่ 2.1

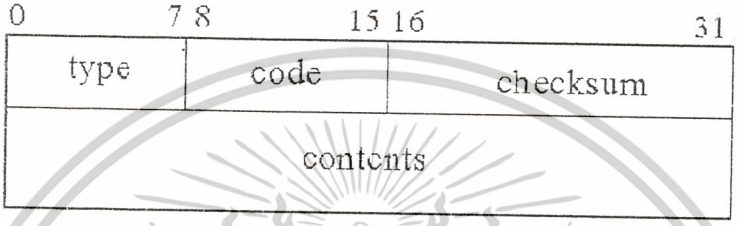
Type	ความหมาย	การใช้งาน
0	Echo reply	แจ้งตอบรับกลับ
3	Destination unreachable	ไม่สามารถติดต่อปลายทางได้
4	Source quench	ให้ต้นทางลดระดับภาระงาน
5	Redirect	แจ้งเส้นทางที่เหมาะสมกว่า
8	Echo request	ร้องขอการตอบกลับ
9	Router advertisement	เราเตอร์แจ้งประกาศตัวเอง
10	Router solicitation	โฮสต์ตรวจหาเราเตอร์
11	Time exceeded for datagram	เดทาแกรมใช้เวลาเกินกำหนด
12	Parameter problem on datagram	มีปัญหาพารามิเตอร์ในเดทาแกรม
13	Time stamp request	ร้องขอเวลาระบบ
14	Time stamp reply	แจ้งเวลาระบบกลับ
15	Information request	ร้องขอข่าวสาร
16	Information reply	แจ้งข่าวสารกลับ
17	Address mask request	ร้องขอแอดเดรสมาสค์
18	Address mask reply	แจ้งแอดเดรสมาสค์

ตารางที่ 2.1 แสดงประเภทความผิดพลาดของ ICMP

2. Code ขนาด 8 บิต : รหัสความผิดพลาดย่อย

3. Checksum ขนาด 16 บิต : ค่าผลรวมตรวจสอบแบบ 1's complement สำหรับใช้ตรวจสอบความผิดพลาด โดยคำนวณผลรวมของ type, code และ contents

4. Contents ขนาดไม่คงที่ : ฟิลด์นี้ใช้บรรจุข้อมูลข่าวสารเพิ่มเติมเพื่อแจ้งกลับซึ่งจะขึ้นอยู่กับค่า byte และ code



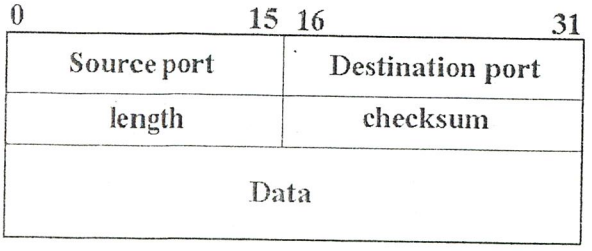
รูปที่ 2.5 ฟอร์แมตของ ICMP

2.4 User Datagram Protocol

User Datagram Protocol หรือ UDP เป็นโปรโตคอลระดับทรานสพอร์ตทำหน้าที่นำส่งข้อมูลจากโปรโตคอลประยุกต์ไปยังไอพี ข้อมูลรวมกับยูดีพีเฮดเดอร์เรียกว่า ยูดีพีเดทาแกรม หรือยูสเซอร์เดทาแกรม

UDP ให้บริการแบบ "connectionless" กล่าวคือไม่สถาปนาการเชื่อมต่อระหว่าง สถานีต้นทางและปลายทาง UDP ส่งเดทาแกรมโดยไม่ตรวจสอบว่าสถานีปลายทางพร้อมที่จะติดต่อหรือไม่ การสื่อสารลักษณะนี้อาจเทียบได้กับการส่งจดหมาย ผู้ส่งเพียงแต่มอบหมายให้ไปรษณีย์จัดส่งโดยไม่ต้องทราบว่าผู้รับปลายทางพร้อมรับหรือไม่

2.4.1 ฟอร์แมตของ UDP



รูปที่ 2.6 แสดง ฟอร์แมตของ UDP

1. source port ขนาด 16 บิต : พอร์ตสถานีต้นทาง
2. destination port ขนาด 16 บิต : พอร์ตสำหรับสถานีปลายทาง
3. length ขนาด 16 บิต : บอกความยาวของเฮดเดอร์และข้อมูล เป็นจำนวนไบต์
4. checksum ขนาด 16 บิต : ผลรวมตรวจสอบ คำนวณจากผลรวมของเฮดเดอร์และข้อมูล

2.4.2 การคำนวณค่าผลรวมตรวจสอบ

ยูดีพีคำนวณผลรวมตรวจสอบจากเฮดเดอร์และข้อมูล ต่างจากผลรวมตรวจสอบของไอพีที่คำนวณจากเฮดเดอร์โดยไม่รวมข้อมูล ค่าผลรวมตรวจสอบของยูดีพีเป็นค่าเผื่อเลือก (optional) ที่จะใช้หรือไม่ใช้ก็ได้ ในขณะที่ทีซีพีบังคับว่าต้องคำนวณผลรวมตรวจสอบเสมอ

ผลรวมตรวจสอบของยูดีพีคำนวณ โดยบวกค่า 16 บิต แบบเลขเติมเต็มหนึ่ง แต่ถ้าความยาวของยูดีพีเดทาแกรมเป็นเลขคี่ให้แทรกค่า 0 อีก 1 ไบต์ เพื่อใช้เฉพาะขั้นตอนการคำนวณเท่านั้น

ยูดีพีจะนำค่า เฮดเดอร์เทียม (psuedo-header) อีก 12 ไบต์เข้ามาร่วมคำนวณผลรวมตรวจสอบด้วย เฮดเดอร์เทียมนำมาจากบางฟิลด์ของไอพีเดทาแกรม

ยูดีพีใช้เฮดเดอร์เทียมเพื่อคำนวณผลรวมตรวจสอบเท่านั้นและไม่ส่งเฮดเดอร์เทียมออกไป หากสถานีปลายทางได้รับเดทาแกรมและพบว่าผลรวมตรวจสอบไม่ถูกต้อง สถานีจะทิ้งเดทาแกรมนั้นไป แต่ละฟิลด์ในเฮดเดอร์เทียมมีความหมายดังนี้

1. source address ขนาด 32 บิต : IP Addressต้นทาง
2. destination address ขนาด 32 บิต : IP Addressปลายทาง
3. zero ขนาด 8 บิต : เป็นศูนย์หมดทุกบิต
4. protocol ขนาด 8 บิต : ชนิดของโปรโตคอล กรณีของยูดีพีจะมีค่าเท่ากับ 17
5. length ขนาด 16 บิต : ขนาดของเฮดเดอร์รวมกับข้อมูล

บริการของยูดีพีเหมาะสมกับโปรโตคอลประยุกต์หลายชนิดโดยเฉพาะโปรโตคอลที่ทำงานแบบไคเอ็นต์-เซอร์ฟเวอร์ ชนิดที่ใช้การร้องขอและตอบกลับ ไคลเอ็นต์ทำหน้าที่ขอบริการและเซอร์ฟเวอร์ตอบกลับไปตามการร้องขอ การตอบกลับนี้เป็นรูปแบบหนึ่งของการตอบรับ (acknowledge) หากไคลเอ็นต์ไม่ได้รับคำตอบกลับภายในระยะเวลาที่กำหนดก็อาจขอบริการซ้ำใหม่ หากไคลเอ็นต์ร้องขอบริการซ้ำหลายครั้งเกินกำหนดโดยไม่ได้รับคำตอบ ไคลเอ็นต์จะถือว่าเซอร์ฟเวอร์ไม่อยู่ในสภาพที่ให้บริการได้

2.5 Simple Network Management Protocol

การบริหารเครือข่ายใน TCP/IP อาศัยรูปแบบการจัดการมาตรฐานตามข้อกำหนดของโปรโตคอล Simple Network Management Protocol หรือ SNMP ซึ่งเป็นโปรโตคอลระดับประยุกต์ที่กำหนดรูปแบบและกรรมวิธีจัดการเครือข่าย โดยมีสถานีจัดการเครือข่ายส่วนกลางทำหน้าที่ดูแล ตรวจสอบ และควบคุมการทำงานของอุปกรณ์เครือข่าย การศึกษา SNMP ต้องเข้าพื้นฐานการทำงานและองค์ประกอบภายในการทำงานของ SNMP และฐานข้อมูลของ SNMP

ประโยชน์จากการใช้คอมพิวเตอร์และเครือข่ายคือประหยัดเวลาและค่าใช้จ่าย แต่ในขณะเดียวกันการใช้ คอมพิวเตอร์ก็ต้องลงทุนทั้งเวลาและค่าใช้จ่ายเพื่อดูแลให้คอมพิวเตอร์ และเครือข่ายทำงานได้ด้วย เครือข่ายขนาดใหญ่ที่ประกอบด้วยคอมพิวเตอร์และอุปกรณ์จำนวนมากจะทำงานได้อย่างมีประสิทธิภาพจำเป็นต้องใช้คอมพิวเตอร์ช่วยบริหารและจัดการตัวระบบเองด้วย

การบริหารเครือข่ายคือ การตรวจ ควบคุม และวางแผนการใช้ทรัพยากรระบบเพื่อให้เครือข่ายทำงานได้อย่างมีประสิทธิภาพและสามารถตรวจหาจุดบกพร่องที่เกิดขึ้นเพื่อแก้ปัญหาได้อย่างรวดเร็ว คอมพิวเตอร์อย่างน้อยหนึ่งเครื่องในเครือข่ายทำหน้าที่เป็น แมเนเจอร์(manager) เพื่อใช้เป็นสถานีจัดการแมเนเจอร์อาจเรียกอีกชื่อว่า สถานีจัดการเครือข่าย (network management station) หรือ เอ็นเอ็มเอส (NMS)

2.5.1 SNMP Agent

การจัดการเครือข่ายใน TCP/IP อาศัยรูปแบบการจัดการมาตรฐานตามข้อกำหนดของโปรโตคอล เอสเอ็นเอ็มพี (SNMP : Simple Network Management Protocol) ซึ่งเป็นโปรโตคอลประยุกต์ที่กำหนดรูปแบบและกรรมวิธีจัดการเครือข่าย

อุปกรณ์เครือข่ายที่เป็นเอเจนต์อาจเป็นพีซี โมเด็ม ฮับ สวิตซ์ หรือ เราเตอร์ อุปกรณ์เหล่านี้อาจมีส่วนทำงานที่เป็นซอฟต์แวร์และฮาร์ดแวร์ และมีเอสเอ็นเอ็มพีเอเจนต์ เชื่อมต่ออยู่ เอเจนต์จะนำข้อมูลจากส่วนซอฟต์แวร์หรือ ฮาร์ดแวร์เมื่อเอ็นเอ็มเอสร้องขอข้อมูล และปรับเปลี่ยนการทำงานของซอฟแวร์หรือ ฮาร์ดแวร์เมื่อเอ็นเอ็มเอสสั่งงาน โดยมีการแจ้งยืนยันสิทธิในรูปรหัสผ่านว่าเอ็นเอ็มเอสมีอำนาจหน้าที่ในการร้องขอและปรับค่า

2.5.2 MIB

เอเจนต์ประกอบด้วยส่วนสำคัญสองส่วนคือ โปรโตคอลเอ็นจิน (protocol engine) และฐานข้อมูลสารสนเทศการจัดการ (Management Information Base : MIB) โปรโตคอลเอ็นจินทำหน้าที่ประมวลคำสั่งที่มาจากเอ็นเอ็มเอสซึ่งได้แก่ รับคำสั่ง ถอดรหัสคำสั่ง ทำงานตามคำสั่งและส่ง

ผลตอบกลับ ฐานสารสนเทศการจัดการหรือเรียกสั้นๆ ว่า มิบ เป็นส่วนที่เก็บตัวแปรและค่ากำหนดการทำงานประจำอุปกรณ์

โครงสร้างมิบ ข้อมูลประจำอุปกรณ์เครือข่ายชิ้นหนึ่งๆ มีได้หลากหลาย อีกทั้งอุปกรณ์ต่างประเภทกันย่อมมีข้อมูลประจำอุปกรณ์แตกต่างกัน ดังนั้นการสอบถาม (อ่าน) หรือเปลี่ยนค่า (เขียน) ฐานข้อมูลจึงต้องมีรูปแบบมาตรฐานให้กับอุปกรณ์ทุกประเภท โครงสร้างต้นไม้แบบลำดับชั้นเป็นโครงสร้างที่เหมาะสมสำหรับใช้เป็นฐานข้อมูลเพื่อจัดเก็บตัวแปรเหล่านี้

ลำดับขั้นแรกจะมีโหนดหลักสามโหนดซึ่งกำหนดกลุ่มองค์กรสามกลุ่มคือ ITU-T(0), ISO (1) และ Joint-ISO-ITU-T (2) ภายใต้โหนด ISO มีโหนดลำดับที่สามคือ org (3) กำหนดองค์กรนานาชาติ และ ส่วนหนึ่งขององค์กรนี้คือ dod(6) หรือ Department of Defense และมีโหนด internet (1) เพื่อกำหนดกลุ่มการจัดการเครือข่ายในอินเทอร์เน็ต

เมื่อต้องการอ้างอิงถึงโหนดใดในโครงสร้างให้เขียนหมายเลขจากรากไปตามเส้นทางถึงโหนดนั้นและคั่นด้วยจุด ลำดับตัวเลขนี้เรียกว่า อ็อบเจ็กไอเด็นติไฟเออร์ (object identifier) หรือ โอไอดี (OID)

ตัวอย่างเช่น 1.3.6.1.2.1.1 เป็นอ็อบเจ็กไอเด็นติไฟเออร์ โดยมีชื่อที่สมนัยกันคือ iso.org.dod.internet.mib-2.system โหนดที่อยู่ภายใต้ 1.3.6.1.2.1 หรือในกลุ่ม mib-2 เป็นโหนดสำหรับใช้งานเอสเอ็มเอ็มพี แต่ละโหนดจะมีโหนดย่อยเพื่ออ้างอิงถึงตัวแปรเช่น 1.3.6.1.2.1.1.1 คือ ตัวแปร sysDescr (System Description) ซึ่งเก็บคำอธิบายเกี่ยวกับอุปกรณ์นั้น

ชนิดของตัวแปรมิบ

แต่ละตัวแปรในเอสเอ็นเอ็มพีมีแบบข้อมูลประจำ แบบข้อมูลที่ใช้อยู่ในเอสเอ็นเอ็มพีมีดังนี้

1. Integer : จำนวนเต็ม เช่น หมายเลขพอร์ตของ โปรโตคอล ทีซีพีหรือ ยูดีพี มีค่าได้ตั้งแต่ 0 ถึง 65535

2. OctetString : สายอักขระขนาดตั้งแต่ 0 อ็อกเท็ต แต่ละอ็อกเท็ตต้องเป็นรหัสแอสกีเอ็นวีที ข้อมูลประเภทนี้มีความยาวตั้งแต่ 0 ถึง 255 อักษร

3. Null : ใช้บอกว่าตัวแปรนั้นไม่มีค่าข้อมูลใดอยู่ เช่น เมื่อสอบถามข้อมูลด้วยคำสั่ง get หรือ get-next-request จะกำหนดแบบข้อมูลตัวแปรเท่ากับ null

4. ObjectIdentifier : ชื่อตัวแปรในรูปของการอ้างอิงถึงแบบตัวเลขตามโครงสร้างมิบ

5. IpAddress : สายอักขระ 4 อ็อกเท็ต แต่ล่ะอ็อกเท็ตแทนไอพีแอสเดรสแต่ละตำแหน่ง

6. PhysicalAddress : สายอักขระกำหนด Hardware Addressเช่น อีเทอร์เน็ตแอดเดรสใช้สายอักขระ 6 อ็อกเท็ต

7. Counter : เลขจำนวนเต็มไม่คิดเครื่องหมาย มีค่าตั้งแต่ 0 ถึง 223-1 (4,294,967,295) การใช้ข้อมูล Counter เป็นแบบเพิ่มค่าขึ้นเพียงอย่างเดียวและเมื่อถึงค่ามากสุดจะกลับเป็น 0 ใหม่

8. Gauge : เลขจำนวนเต็มไม่คิดเครื่องหมาย มีค่าตั้งแต่ 0 ถึง 223-1 โดยสามารถเพิ่มหรือลดค่าได้ แต่เมื่อเพิ่มไปสูงสุดแล้วจะคงค่าไว้จนกว่าจะถูกปรับค่ากลับมาเป็นศูนย์อีกครั้ง ตัวอย่างตัวแปรที่ใช้ค่านี้เช่น จำนวนการเชื่อมโยงทีซีพีที่อนุญาตให้มีได้

9. TimeTicks : เลขจำนวนเต็มใช้นับเวลาในหน่วยเศษหนึ่งส่วนร้อยของวินาที เช่น เวลานับตั้งแต่ที่ระบบเริ่มทำงาน (system uptime)

10. Sequence : โครงสร้างแบบเรคอร์ด หรือคล้ายกับแบบข้อมูล struct ในภาษาซี

11. Sequence of : โครงสร้างแบบตารางหรือมองในรูปของอาเรย์ เช่นตารางเลือกเส้นทางของไอพี

2.5.3 อ็อบเจ็กไอเด็นติไฟเออร์

ภายในมิบประกอบด้วยตัวแปรจำนวนมากเรียกโดยทั่วไปว่า อ็อบเจ็กจัดการ (managed object) อ็อบเจ็กในความหมายนี้เป็นชื่อที่ใช้เรียกตัวแปรและลักษณะเฉพาะของตัวแปรในมิบโดยไม่เกี่ยวข้องกับเรื่อง เชิงวัตถุพิสัย (oject oriented) แต่อย่างใด ขอให้พิจารณาว่าอ็อบเจ็กในเอสเอ็นเอ็มพี มีลักษณะเช่นเดียวกับเรคอร์ดในฐานข้อมูล

แต่ละอ็อบเจ็กจะมีชื่อเฉพาะเรียกว่า อ็อบเจ็กไอเด็นติไฟเออร์ (object identifier) หรือเรียกโดยย่อว่า ไอเด็นติไฟเออร์ (identifier) เพื่อใช้อ้างอิงถึงอ็อบเจ็กนั้น

อ็อบเจ็กทุกตัวมีนิยามที่กำหนด ชื่อ แบบข้อมูล สิทธิ์การเข้าถึง คำอธิบายลักษณะและค่าข้อมูล การนิยามอ็อบเจ็กมีกฎเกณฑ์ตามข้อกำหนด โครงสร้างฐานข้อมูลสารสนเทศการจัดการ (Structure of Management Information : SMI)

2.5.4 โปรโตคอล

การติดต่อระหว่างสถานีจัดการกับเอเจนต์มีรูปแบบในการติดต่อหลายรูปแบบด้วยกันตามวัตถุประสงค์ในการติดต่อ แบบของการติดต่อในเอสเอ็นเอ็มพีรุ่น 1 มี 5 แบบคือ

1. get-request ใช้สอบถามข้อมูลจากตัวเอเจนต์ที่อยู่บนอุปกรณ์ที่ต้องการตรวจสอบในระบบเครือข่าย

2. get-next-request ใช้สอบถามข้อมูลที่เรียงเป็นลำดับ เช่นข้อมูลที่เก็บอยู่ ในรูปตาราง หรือในกรณีที่ไม่ทราบชื่อตัวแปรที่แน่ชัด

3. get-response เอเจนต์ส่งคำตอบกลับมายังผู้สอบถาม

4. set-request ใช้เปลี่ยนแปลงค่าตัวแปรที่เอเจนต์รับผิดชอบอยู่

5. trap ใช้แจ้งเหตุการณ์ที่เกิดขึ้นในระบบเครือข่าย เช่นการเริ่มต้นทำงานใหม่ของอุปกรณ์หรือเส้นทางขัดข้อง

เอสเอ็นเอ็มพีอาศัยโปรโตคอลยูดีพี โดยใช้พอร์ตหมายเลข 161 สำหรับการติดต่อในแบบที่ 1 ถึง 4 และใช้พอร์ตหมายเลข 162 สำหรับแบบที่ 5

2.5.4.1 การเอ็นแคปซูเลต

การเอ็นแคปซูเลตคำสั่งและข้อมูลในเอสเอ็นเอ็มพี ฟอร์แมตของเอสเอ็นเอ็มพีประกอบด้วย 2 ส่วนคือ เฮดเดอร์และพีดียู เฮดเดอร์ประกอบด้วยฟิลด์ย่อยสองฟิลด์คือ

version : รุ่นของโปร โตคอลที่ใช้ ถ้าเป็นโปรโตคอลรุ่น 1 จะมีค่า 0 หากเป็นรุ่น 2 จะมีค่า 1

community : รหัสผ่านในรูปสายอักขระเพื่อให้เอเจนต์ใช้ตรวจสอบว่าข้อความที่ส่งมามีสิทธิ์ในการสอบถามหรือเปลี่ยนแปลงข้อมูลหรือไม่

ในส่วนของพีดียูประกอบด้วยฟิลด์ย่อยตามชนิดของข้อความ หากเป็นข้อความ get, get-next และ get-response จะมีโครงสร้างเดียวกัน

PDU type : แบบการติดต่อ (1 ถึง 5)

Request ID : กำหนดบอกหมายเลขข้อความเพื่อใช้จับคู่เมื่อรับคำตอบกลับมา

Error Status : สถานะความผิดพลาดที่เกิดขึ้น

Error Index : ดรรชนีชี้ค่าผิดพลาดที่เกิดขึ้นเกิดจากตัวแปรลำดับที่เท่าไหร่ของตัวแปรทั้งหมดที่สอบถามไป

VarBindList : ค่าผูกพันตัวแปร (variable binding) แสดงอยู่ในรูปของตัวแปรและค่าของตัวแปรต่อเนื่องกันไปเป็นรายการ

2.6 ความรู้เบื้องต้นเกี่ยวกับ I^2C

I^2C ย่อมาจาก Intergrated Circuit Communication หมายถึง การติดต่อสื่อสารระหว่าง ไอซีต่างๆ พัฒนาขึ้นโดย ฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อ สั่งงาน และควบคุมภายใต้สายสัญญาณเพียง 2 เส้น เส้นหนึ่งคือ สายข้อมูล อีกเส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I^2C ทำได้ง่ายมาก เพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานกันไปหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I^2C มีชื่อเรียกอีกชื่อหนึ่งว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data line) ส่วน สายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock line)

2.6.1 คุณสมบัติโดยทั่วไปของบัส I^2C

สาย SDA และ SCL เป็นสายสัญญาณสองทิศทาง (bi – directional line) ต้องมีการต่อตัวต้านทานพูลอัปกับแรงดัน +5V ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสอง วงจรเอาท์พุทของอุปกรณ์ที่ต่ออยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (Open drain) หรือ Open collector

อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปกติ (standard mode) และสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง (fast mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pF การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับ การเข้าถึง 2 ค่าคือ 7 บิต (7 – bit addressing) หรือ 10 บิต (10-bit addressing)

2.6.2 หลักการของบัส I^2C

บัส I^2C ประกอบด้วยสายสัญญาณ 2 เส้น คือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัส หรือเรียกว่า โปรโตคอล (protocol) เพื่อให้ผู้ใช้งานทราบว่า ขณะนี้ติดต่อกับอุปกรณ์ตัวใดอยู่ และอุปกรณ์ตัวใดเป็นตัวรับหรือส่ง

อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูลเรียกว่า ตัวส่ง (transmitter)

อุปกรณ์ที่เป็นผู้รับข้อมูล เรียกว่า ตัวรับ (receiver) อุปกรณ์บนบัส I^2C สามารถเป็นได้ทั้งตัวรับและตัวส่ง บางอุปกรณ์ทำหน้าที่เป็นตัวรับอย่างเดียว จะไม่มีอุปกรณ์ใดบนบัส I^2C ที่ทำหน้าที่เป็นตัวส่งเพียงอย่างเดียว

อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I^2C เรียกว่า มาสเตอร์ (master)

อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (slave)

ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I^2C คือ

(1) การถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น

(2) ในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะเป็นลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดการเปลี่ยนแปลงขึ้นเด็ดขาด มิฉะนั้น สัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

2.6.3 สภาวะที่เกิดขึ้นบนบัส I^2C

มีด้วยกัน 5 สภาวะ ดังนี้

(1) บัสว่าง (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทั้งคู่ นั่นหมายความว่า การถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้

(2) เริ่มต้นการถ่ายทอดข้อมูล (Start Data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะเริ่มต้น (START)

(3) หยุดการถ่ายทอดข้อมูล (Stop data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากต่ำไปสูง ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะหยุด (STOP)

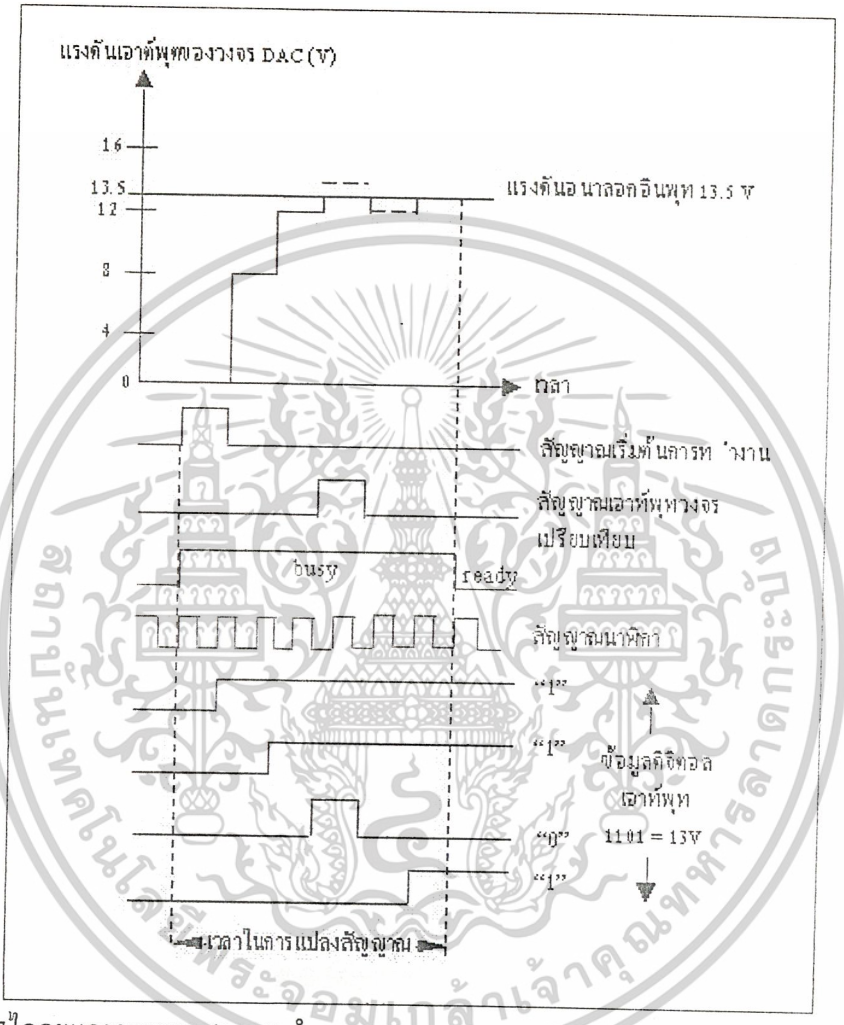
(4) ข้อมูลคำรงอยู่บนบัส (data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่า เป็น "0" หรือ "1" ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดขึ้น

(5) รับรู้ข้อมูล (acknowledge) เกิดขึ้นหลังจากที่การถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิต เรียกว่า บิตรับรู้ (acknowledge) มีสถานะเป็นลอจิกสูง หลังจากส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกาเพื่อตอบสนองบิตรับรู้ที่ส่งมาจากตัวส่ง ทางด้านตัวรับจะส่งบิตรับรู้ที่มีสถานะลอจิกต่ำลงบนบัส อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้เพื่อตอบสนองให้ทราบว่า ได้รับข้อมูลในแต่ละไบต์เรียบร้อยแล้ว

2.7 การแปลงสัญญาณอะนาลอกดิจิตอลแบบซักเซสซีฟแอปพร็อกซิเมชั่น (Successive Approximation ADC)

การแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC) ที่ได้รับความนิยมสูงสุดและมีประสิทธิภาพดีคือ การแปลงแบบซักเซสซีฟแอปพร็อกซิเมชั่น หรือเราอาจเรียกอีกอย่างหนึ่งว่า เป็นการแปลงแบบประมาณค่าใกล้เคียง

วงจร ADC แบบซักเซสซีฟแอปพร็อกซิเมชั่น นี้จะใช้รีจิสเตอร์ในการส่งข้อมูลดิจิตอลของวงจร DAC ภายในแต่ละบิตของรีจิสเตอร์จะเซตและรีเซตโดยการควบคุมจากวงจรควบคุม ต่อไปเป็นการอธิบายการทำงานของ ADC แบบนี้ไปทีละขั้นตอน ขอให้พิจารณาไดอะแกรมเวลารูปร่วมด้วย



รูปที่ 2.7 ไดอะแกรมเวลาแสดงการทำงานของวงจร ADC แบบ Successive Approximation

กำหนดให้แรงดันอะนาลอกอินพุต (Vin) มีค่า 13.5 V

1.ส่งสัญญาณเริ่มต้นการทำงาน (start converter) มายังซักเซสซีฟแอปพร็อกซิเมชั่นรีจิสเตอร์

2.ขณะนี้สถานะการทำงานของรีจีสเตอร์ไม่ว่าง (busy) สัญญาณนาฬิการลูกแรกถูกส่งเข้ามาเพื่อกำหนดให้ค่าของรีจีสเตอร์เท่ากับ 0000

3.เอาต์พุตของ DAC จะเป็น 0V ส่งไปในวงจรเปรียบเทียบ เพื่อเปรียบเทียบกับ Vin ในขณะนี้จะได้เอาต์พุตเท่ากับ –5V กำหนดเป็นลอจิก 0

4.เมื่อสัญญาณนาฬิกาลูกต่อไปเข้ามา จะทำการเซตบิต MSB ของรีจีสเตอร์เป็น “1”

5.ในกรณีนี้เป็น ADC ขนาด 4 บิต ดังนั้นการที่บิต MSB เซตจะทำให้วงจร DAC แปลงค่าเป็นแรงดัน 8V นำไปเปรียบเทียบแรงดัน แต่ก็ยังน้อยกว่า Vin ดังนั้นเอาท์พุตของวงจรเปรียบเทียบยังคงเป็น “0” ทำให้รีจีสเตอร์ยังคงค่าบิต MSB ให้เป็น “1” ต่อไป

6. ต่อมาบิต B2 (ถัดจาก MSB 1 บิต เนื่องจากมี 4 บิต กำหนดบิต MSB = B3) จะเซตเนื่องจากมีค่าเท่ากับ 4V นำไปรวมกับค่าของบิต MSB ที่มีอยู่ 8V เช่น 12V นำไปเปรียบเทียบกับ Vin ก็ยังน้อยกว่า รีจีสเตอร์จึงยังคงค่า B2 ไว้ที่ “1” เช่นกัน

7. ต่อมาบิต B1 จะเซตทำให้แรงดันเอาต์พุทมา DAC กลายเป็น 8+4+2 = 14V ซึ่งมากกว่า Vin ทำให้วงจรเปรียบเทียบเกิดการเปลี่ยนสถานะเป็น “1” ซึ่งจะส่งสัญญาณมาควบคุมให้ B1 กลายเป็น “0”

8. เมื่อบิต LSB ถูกเซต จะมีค่าแรงดัน 1V เข้ามารวมกับค่าของ B3 B2 และ B1 เป็น 8+4+0+1 = 13V นำไปเปรียบเทียบกับ Vin ปรากฏว่าน้อยกว่า Vin ทำให้บิต B0 หรือ LSB มีค่าเป็น “1”

9. ขณะนี้ทุกบิตในรีจีสเตอร์ถูกนำมาแปลงค่าเรียบร้อยแล้ว ทำให้สถานะของรีจีสเตอร์กลับมาเป็น พร้อมทำงาน (ready)

10. ข้อมูลดิจิตอลที่ได้จากการ ADC แบบนี้ จะมีค่า 1101_2 หรือ 13V ซึ่งใกล้เคียงกับ Vin 13.5 V มากที่สุด ถ้าหากรีจีสเตอร์มีจำนวนบิตมากกว่านี้ ความละเอียดของข้อมูลที่แปลงได้จะมีความใกล้เคียงกันมากขึ้น ช่วงเวลาของการแปลงสัญญาณจะเริ่มต้นขึ้นตั้งแต่สัญญาณนาฬิกาลูกแรกถูกส่งเข้าไปเตรียมระบบ ไปจนถึงเมื่อสภาวะของรีจีสเตอร์กลับมาเป็น “พร้อมทำงาน” อีกครั้งหนึ่ง ซึ่งจะต้องใช้จำนวนสัญญาณนาฬิกาเท่ากับ n+1 พัลส์ โดย n เท่ากับจำนวนบิตของรีจีสเตอร์

2.8 ระบบสื่อสารข้อมูอนุกรมแบบหนึ่งสาย (1 – Wire Serial Bus)

ระบบการสื่อสารข้อมูลแบบนี้ผู้คิดค้นคือ ดัลลัสเซมิคอมดัคเตอร์ ดังนั้นในบางครั้งจึงเรียกระบบการสื่อสารข้อมูลแบบนี้ว่า ระบบสื่อสารข้อมูลดัลลัสหนึ่งสาย (The Dallas 1-Wire Bus) ระบบสื่อสารข้อมูลแบบนี้เป็นระบบที่มีความชาญฉลาด และใช้จำนวนสายเพียง 1 เส้นเท่านั้นโดยไม่ต้องมีสายสัญญาณนาฬิกามาควบคุมจังหวะการถ่ายทอดข้อมูลเหมือนกับระบบสื่อสารข้อมูลแบบอนุกรมในแบบอื่นๆ เนื่องจากสายข้อมูลนั้นจะทำหน้าที่เสมือนหนึ่งเป็นสายสัญญาณนาฬิกาในตัว ส่วนค่าของข้อมูลจะพิจารณาจากลักษณะของรูปสัญญาณที่ปรากฏบนสายสัญญาณในแต่ละช่วงของเวลาหรือต่อไปนี้จะขอเรียกว่า ไทม์สล็อต (time – slot) โดยคาบเวลาต่ำสุดและสูงสุดของสถานะต่างๆ ในการสื่อสารข้อมูลในแต่ละไทม์สล็อต มีการกำหนดขอบเขตไว้อย่างชัดเจน การถ่าย

ทอดข้อมูลจะเกิดขึ้นในแต่ละไทม์สล็อตนั้น รูปแบบการถ่ายทอดข้อมูลจะเป็นแบบอะซิงโครนัสในระดับบิต ไม่มีการกำหนดความยาวของข้อมูลเป็นระดับไบต์ ระบบสื่อสารแบบนี้เหมาะที่จะใช้ในการสื่อสารข้อมูลระหว่างไอซีบนแผงวงจรเดียวกัน

2.8.1 คุณสมบัติทางเทคนิคของระบบบัสหนึ่งสาย

สายสัญญาณบนระบบบัสแบบหนึ่งสายนี้จะเป็นสายสัญญาณแบบสองทิศทาง แต่ข้อมูลจะสามารถเดินทางได้ในทิศทางเดียวภายในช่วงเวลาหนึ่งๆ นั่นคือ มีลักษณะคล้ายกับระบบสื่อสารแบบ half-duplex ตัวอย่างที่เห็นได้ชัดเจนคือ การใช้งานวิทยุสื่อสารหรือวิทยุสมัครเล่น อุปกรณ์บนระบบบัสต้องมีการระบุอย่างชัดเจนว่าตัวใดเป็นอุปกรณ์มาสเตอร์ ตัวใดเป็นอุปกรณ์สเลฟ โดยส่วนใหญ่อุปกรณ์มาสเตอร์คือไมโครคอนโทรเลอร์ ส่วนอุปกรณ์สเลฟได้แก่ ไอซีตรวจอุณหภูมิ, ไอซีหน่วยความจำแรม เป็นต้น อุปกรณ์มาสเตอร์จะเป็นตัวจัดเตรียมความพร้อมของสัญญาณและควบคุมการถ่ายทอดข้อมูลบนสายสัญญาณนั้น ข้อมูลทั้งหมดไม่ว่าจะเป็นข้อมูลควบคุมหรือข้อมูลใช้งานจะถูกส่งลงบนสายสัญญาณที่มีอยู่เพียงเส้นเดียวนี้ทั้งหมด ในระหว่างการทำงานอุปกรณ์มาสเตอร์และสเลฟสามารถเป็นได้ทั้งตัวส่งและตัวรับ ขึ้นอยู่กับเงื่อนไขการทำงานในขณะนั้น ยกตัวอย่าง ถ้าหากมีการเขียนข้อมูลจากอุปกรณ์มาสเตอร์ไปยังอุปกรณ์สเลฟ ตัวส่งคืออุปกรณ์มาสเตอร์ ตัวรับคือ อุปกรณ์สเลฟ ในทางตรงข้าม หากเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ ตัวส่งจะกลายเป็นอุปกรณ์สเลฟ และตัวรับคืออุปกรณ์มาสเตอร์ ในระบบบัส 1 ระบบต้องมีอุปกรณ์มาสเตอร์เพียงตัวเดียวเท่านั้น

สายสัญญาณของระบบบัสนี้ต้องกำหนดสภาวะปกติไว้ที่ลอจิกสูง สามารถทำได้โดยการต่อตัวต้านทานค่าประมาณ 4.7 kΩ พูลอัปกับไฟเลี้ยง+5V ดังนั้นอุปกรณ์ที่นำเข้ามาต่อบนระบบบัสนี้ต้องออกแบบให้ภาคเอาต์พุทที่ต่อกับสายสัญญาณมีลักษณะเป็นคอลเล็กเตอร์เปิดหรือเดรนเปิด

2.8.2 คุณสมบัติของไทม์สล็อต

อุปกรณ์มาสเตอร์จะเป็นอุปกรณ์เพียงตัวเดียวบนระบบบัสหนึ่งสายนี้ที่สามารถทำการอินนิเชียลสายสัญญาณได้ โดยอุปกรณ์มาสเตอร์จะกำเนิดจุดเริ่มต้นของไทม์สล็อตด้วยการทำให้สายสัญญาณเป็นลอจิกต่ำในช่วงเวลาหนึ่ง จากนั้นก็จะทำให้กลับมาเป็นลอจิกสูง ถ้าหากอุปกรณ์สเลฟต้องการส่งข้อมูลมายังอุปกรณ์มาสเตอร์ อุปกรณ์สเลฟจะเป็นตัวควบคุมสภาวะของสายสัญญาณต่อไป จนเสร็จสิ้นกระบวนการ แต่ถ้าหากอุปกรณ์มาสเตอร์ต้องการส่งข้อมูลก็จะสามารถดำเนินการต่อไปได้เลย

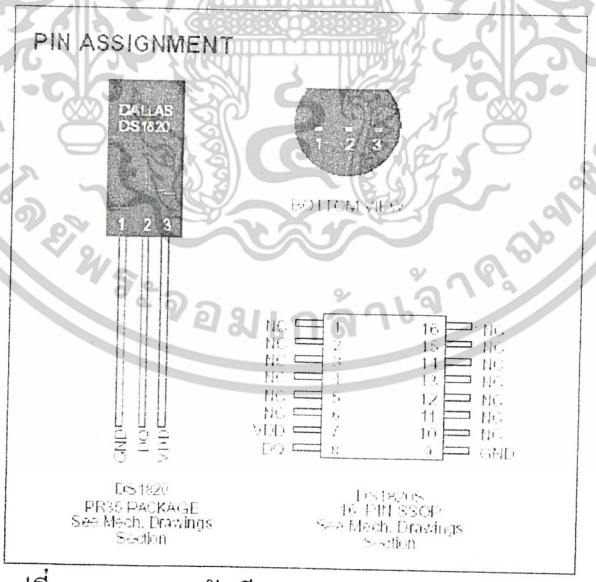
ฟังก์ชันของไทม์สล็อตที่กำหนดโดยอุปกรณ์มาสเตอร์มีด้วยกัน 4 ฟังก์ชันคือ RESET, การอ่านข้อมูล (READ DATA), การเขียนข้อมูล “1” (WRITE ONE) และการเขียนข้อมูล “0” (WRITE

ZERO) ไทม์สล็อตที่ใช้ในการรีเซตใช้ในการเริ่มต้นติดต่อกับอุปกรณ์สเลฟ ในขณะที่ไทม์สล็อตการอ่านสำหรับอ่านข้อมูลที่ส่งมาจากอุปกรณ์สเลฟ ส่วนไทม์สล็อตการเขียนข้อมูล “1” และ “0” ใช้สำหรับเขียนข้อมูลไปยังอุปกรณ์สเลฟผ่านสายสัญญาณของระบบ

ทางด้านอุปกรณ์สเลฟมีฟังก์ชั่นของไทม์สล็อตอยู่ทั้งสิ้น 3 ฟังก์ชันคือ ไทม์สล็อตการตอบสนอง (PRESENCE), การเขียนข้อมูล “1” (WRITE ONE) และการเขียนข้อมูล”0” ไทม์สล็อตของอุปกรณ์ตอบสนองใช้สำหรับตอบสนองการติดต่อจากอุปกรณ์มาสเตอร์ โดยอุปกรณ์สเลฟตัวที่ถูกเลือกจากอุปกรณ์มาสเตอร์จะต้องส่งสัญญาณตอบสนองลงบนสายสัญญาณเพื่อแจ้งให้อุปกรณ์มาสเตอร์ทราบว่า ขณะนี้สามารถติดต่อกันได้แล้ว ส่วนไทม์สล็อตการเขียนข้อมูล “1” และ “0” ใช้สำหรับส่งข้อมูลไปยังอุปกรณ์มาสเตอร์ผ่านสายสัญญาณของระบบ ซึ่งจะสัมพันธ์กับไทม์สล็อตการอ่านข้อมูลของอุปกรณ์มาสเตอร์

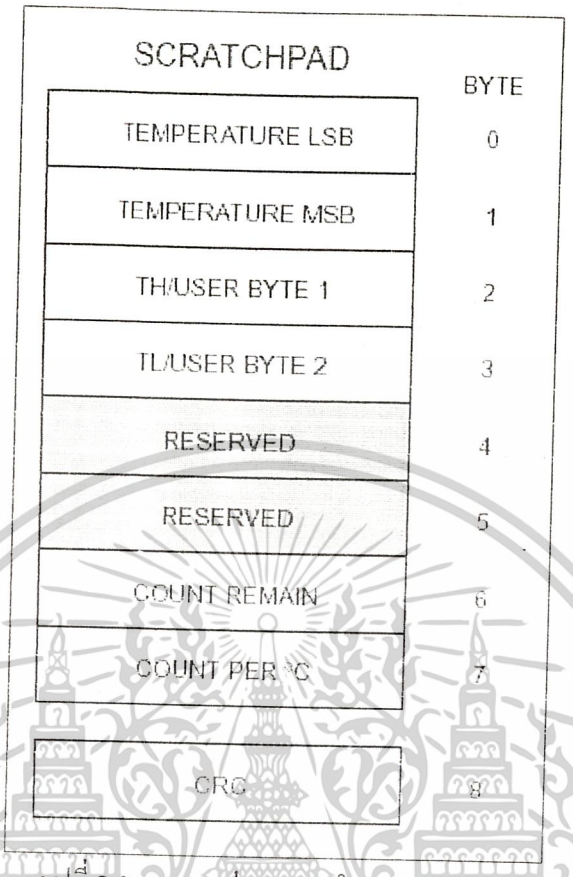
2.9 ไอซีตรวจจับอุณหภูมิ DS1820

เป็นไอซีตรวจจับอุณหภูมิที่ใช้การติดต่อแบบระบบบัสหนึ่งสาย มีขาต่อใช้งานเพียง 3 ขา คือ DQ ซึ่งเป็นการเชื่อมต่อกับระบบบัส, ขาต่อไฟเลี้ยงภายนอก และขากราวด์ ดังแสดงการจัดขาของไอซี DS1820 ในรูป



รูปที่ 2.8 แสดงการจัดเรียงขาของ DS1820 แบบต่างๆ

หัวใจสำคัญของ DS1820 อยู่ที่ตัวตรวจจับอุณหภูมิและหน่วยความจำความเร็วสูงที่เรียกว่า สแครตช์แพด (scratchpad) ซึ่งมีขนาด 9 ไบต์มีการจัดสรรหน่วยความจำส่วนนี้แสดงได้ดังรูปที่ 2.9



รูปที่ 2.9 แสดงหน่วยความจำของ DS1820

เมื่อวัดอุณหภูมิได้ก็จะนำค่าที่วัดได้นี้มาเก็บไว้ใน สแครตช์แพดที่ไบต์ 0 และ 1 ทั้งนี้เนื่องจากไอซี DS1820 สามารถให้ข้อมูลของอุณหภูมิได้ละเอียดถึง 16 บิต เมื่อนำมาแปลงเป็นข้อมูลเลขฐานสิบจึงสามารถแสดงความละเอียดของค่าอุณหภูมิได้ถึง 0.5 องศาเซลเซียลและ 0.9 องศาฟาเรนไฮต์ โดยมีย่านวัดอุณหภูมิ –55 ถึง +125 องศาเซลเซียล หรือ –67 ถึง +257 องศาฟาเรนไฮต์ โดยค่าขององศาฟาเรนไฮต์ต้องใช้การแปลงหน่วยเข้ามาช่วยใช้เวลาในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลประมาณ 200 มิลลิวินาที สามารถกำหนดขอบเขตของอุณหภูมิที่ทำการวัดได้ และให้แจ้งเตือนเมื่อค่าของอุณหภูมิสูงขึ้นหรือลดต่ำลงถึงค่าที่กำหนด โดยค่าอุณหภูมิที่กำหนดนี้จะเก็บไว้ที่สแครตช์แพดในไบต์ 2 และ 3

คำสั่งเพื่อควบคุมการทำงานของ DS1820

ในการติดต่อกับไอซี DS1820 จะมีคำสั่งที่ต้องส่งให้แก่ DS1820 เพื่อกำหนดรูปแบบการทำงาน คำสั่งที่ใช้มากที่สุดมีด้วยกัน 3 คำสั่งดังนี้

1. คำสั่งไม่ติดต่อกับหน่วยความจำรอม หรือ Skip ROM เนื่องจากในการใช้งาน DS1820 โดยปกติแล้วจะมี DS1820 อยู่บนสายสัญญาณเพียงตัวเดียว จึงไม่จำเป็นต้องใช้ข้อมูลกำหนดแอดเดรส ดังนั้นจึงไม่ต้องติดต่อกับหน่วยความจำรอมเพื่ออ่านข้อมูล ข้อมูลของคำสั่ง Skip ROM ที่ต้องส่งให้ DS1820 คือ 0CCH

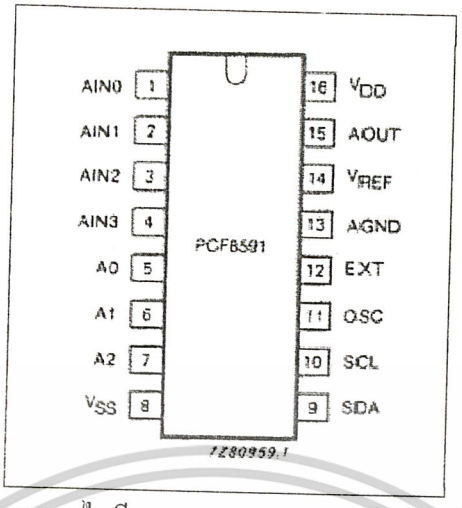
2. คำสั่งแปลงอุณหภูมิ (Convert T) มีค่าเท่ากับ 44H เมื่อส่งคำสั่งนี้ให้กับ DS1820 จะต้องทำการวนลูปรออย่างน้อย 200 มิลลิวินาที เพื่อให้ DS1820 ได้ใช้เวลานี้ในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลมาเก็บไว้ในสแครตช์แพด

3. คำสั่งอ่านข้อมูลจากสแครตช์แพด (Read Scratchpad) มีค่าเท่ากับ 0BEH เมื่อส่งคำสั่งนี้ DS1820 จะทยอยส่งข้อมูลค่าอุณหภูมิออกมาทั้งหมด 9 ไบต์

2.10 PCF8591 A/D & D/A

PCF8591 เป็นชิพเพอริเฟอรัลสำหรับแอพพลิเคชั่นที่ต้องการให้ไมโครคอนโทรลเลอร์สามารถอินเตอร์เฟสกับสัญญาณอนาลอก ภายในมีวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล (ADC) ขนาดความละเอียด 8 บิต จำนวน 4 ช่อง และมีวงจรแปลงข้อมูลดิจิตอลเป็นอนาลอก (DAC) ขนาดความละเอียด 8 บิต หนึ่งช่อง การอินเตอร์เฟสกับไมโครคอนโทรลเลอร์จะกระทำผ่านระบบบัส I^2C โดยคำสั่งและข้อมูลที่อ่านและเขียนจะรับส่งผ่านบัส I^2C ด้วยสายสัญญาณ 2 เส้นคือ SDA และ SCL ขาสัญญาณและตัวถังขนาด 16 ขา วงจร ADC และ DAC ทำงานเป็นแบบ Successive Approximation สามารถนำไปประยุกต์ใช้งานได้อย่างกว้างขวาง มีรายละเอียดคุณสมบัติทางเทคนิคดังนี้

- ทำงานโดยใช้แหล่งจ่ายไฟชุดเดียว
- ทำงานที่แรงดัน 2.5V ถึง 6V
- ติดต่อกับไมโครคอมพิวเตอร์หรือไมโครคอนโทรลเลอร์ผ่านระบบบัส I^2C
- เลือกตำแหน่งแอดเดรสทางฮาร์ดแวร์จากขา A0, A1, A2 ทำให้สามารถต่อพ่วงกันได้สูงสุดถึง 8 ตัว
- สัญญาณอนาลอกมีระดับแรงดันตั้งแต่ V_{SS} ถึง V_{DD}
- มีวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอกขนาด 8 บิต 1 ช่อง



รูปที่ 2.10 การจัดขาของไอซี ADC/DAC ขนาด 8 บิตผ่านบัส I^2C เบอร์ PCF8591

ส่วนรายละเอียดตำแหน่งขาต่างๆมีดังนี้

ขา AN0-AN3 (ขา 1-4) เป็นขาอินพุทสำหรับป้อนสัญญาณอนาลอกที่ต้องการแปลง

ขา A0-A2 (ขา 5-7) เป็นขาสำหรับกำหนดข้อมูลแอดเดรสทางฮาร์ดแวร์ ปกติต่อลงกราวด์ แต่ถ้ามีการใช้งาน PCF8591 มากกว่า 1 ตัวต้องกำหนดการต่อขา A0-A2 ของ PCF8591 ให้ไม่ตรงกัน จึงทำให้สามารถต่อใช้งานร่วมกันได้

ขา V_{ss} (ขา 8) เป็นขาต่อลงกราวด์

ขา SDA, SCL (ขา 9 และ 10) เป็นขาเชื่อมต่อระบบบัส I^2C

ขา OSC (ขา 11) เป็นขาสำหรับต่อกับสัญญาณนาฬิกาภายนอกเมื่อขา EXT ต่อกับไฟ +5V และจะทำงานเป็นขาเอาท์พุทสัญญาณนาฬิกา ถ้าขา EXT ต่อลงกราวด์

ขา EXT (ขา 12) เป็นขาสำหรับเลือกแหล่งกำเนิดสัญญาณนาฬิกา ถ้าต่อไฟ +5V จะเป็นการเลือกใช้สัญญาณนาฬิกาจากภายนอก โดยต่อสัญญาณนาฬิกาเข้าที่ขา OSC ถ้าต่อขานี้ลงกราวด์ จะเป็นการเลือกใช้สัญญาณนาฬิกาจากภายใน

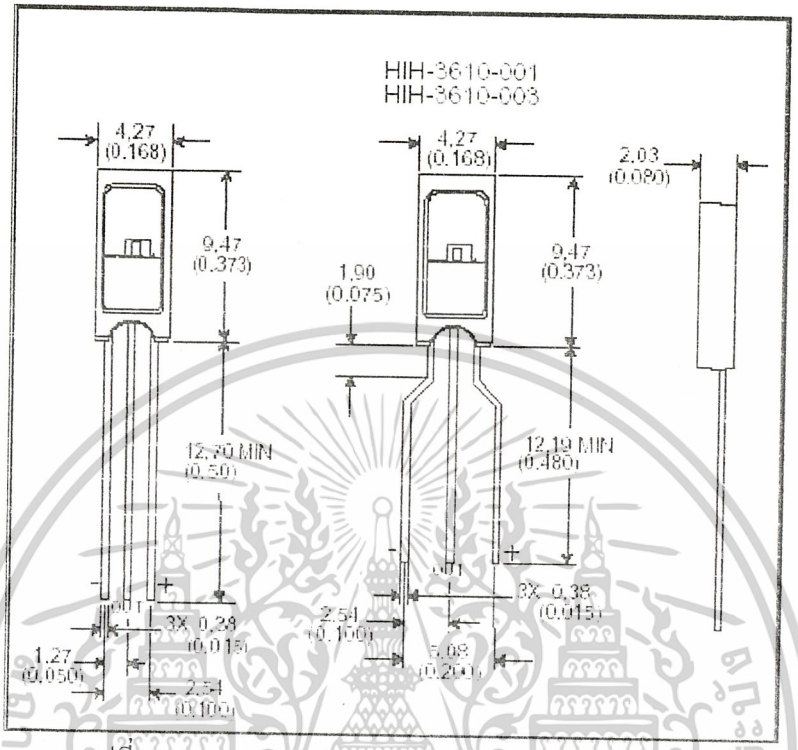
ขา AGND (ขา 13) เป็นขากราวด์ของแรงดันอ้างอิง ปกติต่อลงกราวด์

ขา V_{REF} (ขา 14) เป็นขาสำหรับป้อนแรงดันอ้างอิง

ขา AOUT (ขา 15) เป็นขาเอาท์พุทของวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอก

ขา V_{DD} (ขา 16) เป็นขาต่อไฟเลี้ยง จ่ายได้ตั้งแต่ +2V ถึง +6V

2.11 Humidity Sensor HIH – 3610



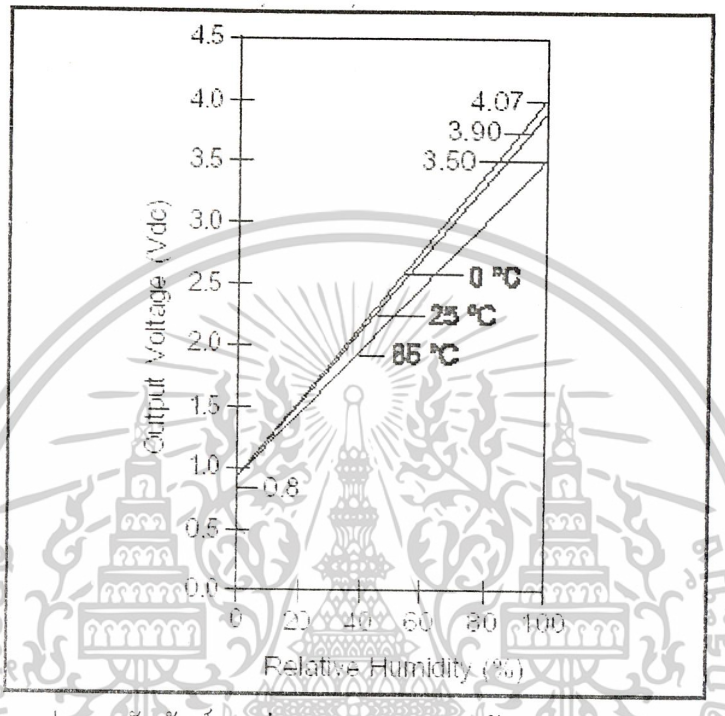
รูปที่ 2.11 แสดงขนาดและการจัดเรียงขาของ HIH-3610

เซนเซอร์ความชื้น HIH 3610 สามารถแสดงค่าเปอร์เซ็นต์ของความชื้นสัมพัทธ์ (Relative Humidity) ได้โดยมีความสัมพันธ์กับค่าแรงดันทางไฟฟ้า (voltage output) เป็นแบบเชิงเส้น แสดงดังตาราง

Model	HIH-3610-001
Channel	92
Wafer	030996M
MRP	337313
Calculated values at 5 V V_{out} @ 0% RH V_{out} @ 75.3% RH	 0.958 V 3.268 V
Linear output for 2% RH accuracy @ 25 °C Zero offset Slope RH	 0.958 V 30.680 mV/%RH (V_{out}-zero offset)/slope (V_{out}-0.958)/0.0307
Ratiometric response for 0 to 100% RH V_{out}	 V_{supply} (0.1915 to 0.8130)

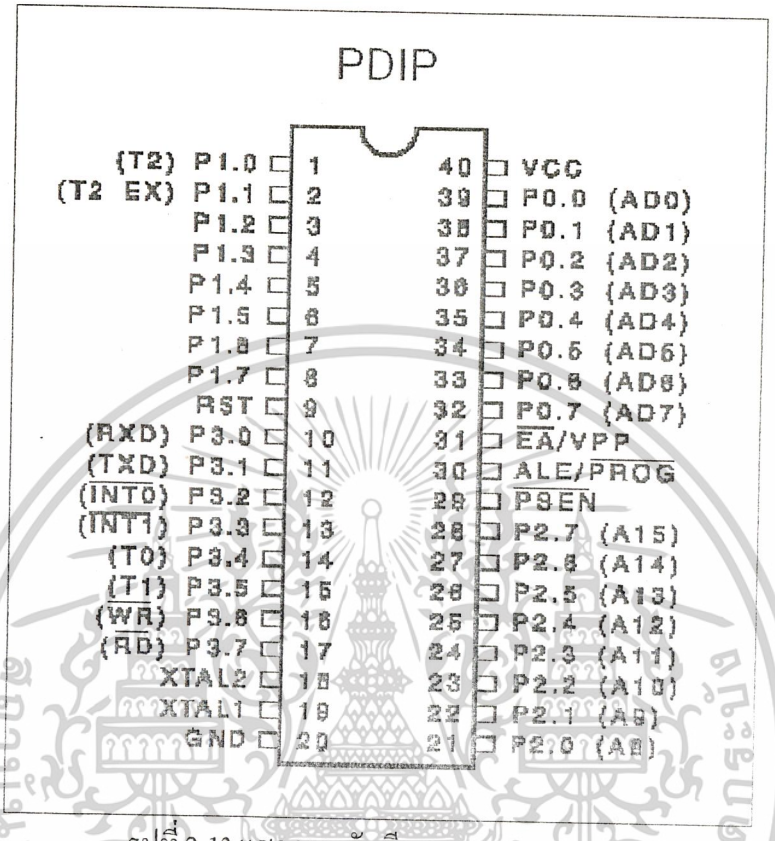
ตารางที่ 2.2 ตารางแสดงการปรับค่าตั้งจากผู้ผลิตของ HIH-3610

ส่วนการนำเอา HIH-3610 ไปใช้งานทำได้โดยการต่อร่วมกับไมโครคอนโทรลเลอร์หรืออุปกรณ์ที่มีความสามารถรับค่าแรงดันเอาท์พุทจาก HIH-3610 แล้วนำมาคำนวณหาค่าเปอร์เซ็นต์ความชื้นจากค่าแรงดัน Offset ที่ผลิตกำหนดให้มาดังรูป



รูปที่ 2.12 แสดงกราฟความสัมพันธ์ระหว่าง Output voltage กับ Relative Humidity ที่ 0 °C, 25 °C และ 85 °C

2.12 AT89C55 8 Bit Microcontroller with 20 Kbytes Flash



รูปที่ 2.13 แสดงการจัดเรียงขาของ AT89C55

AT89C55 เป็นไมโครคอนโทรเลอร์ 8 bit ขนาด 40 ขา สร้างด้วยเทคโนโลยี Atmel's High Density Nonvolatile Memory Technology โดยมีหน่วยความจำโปรแกรมภายในขนาด 20 Kbytes แบบ Flash Memory หรือที่เรียกว่า Programmable and Erasable Read Only Memory (PEROM) หน่วยความจำข้อมูลภายในขนาด 256 ไบต์ สร้างด้วยเทคโนโลยี CMOS AT89C55 เป็นไมโครคอนโทรเลอร์ผลิตโดย ATMEL ชุดคำสั่งและสถาปัตยกรรมภายในเหมือนไมโครคอนโทรเลอร์ตระกูล MCS-51 ซึ่งผลิตโดย INTEL ที่ใช้พลังงานต่ำและหน่วยความจำภายใน สามารถเขียนและลบได้ไม่น้อยกว่า 1,000 ครั้ง AT89C55 เป็นไมโครคอนโทรเลอร์ที่มีความสามารถในการเปลี่ยนแปลงสูง และผลที่ได้คุ้มค่าซึ่งสามารถนำมาประยุกต์ใช้ในการควบคุมและสามารถเลือกใช้ซอร์ฟแวร์ในการประหยัดพลังงานได้ 2 โหมด คือ โหมด Idle และ Powerdown

คุณสมบัติของ AT89C55

1. เข้ากันได้กับไมโครคอนโทรเลอร์ตระกูล MCS – 51

2. หน่วยความจำโปรแกรมเป็นแบบ Flash Memory ขนาด 20 Kbytes สามารถเขียนหรือลบโปรแกรมได้ถึง 1,000 ครั้งและเก็บข้อมูลได้นานถึง 10 ปี
3. ใช้ไฟเลี้ยงตั้งแต่ 2.7 – 6 volt
4. ทำงานได้ในช่วงความถี่ 0 Hz ถึง 33 MHz
5. ระบบหน่วยความจำโปรแกรมมี 3 ระดับ
6. มี I/O port 32 bit
7. มี Counter/Timer ขนาด 16 บิต 3 ตัว
8. พอร์ตอนุกรมแบบ Full Duplex
9. มี Interrupt 8 แหล่ง
10. มีทั้งโหมด Low power Idle และ Power Down

2.13 ความชื้นสัมพัทธ์ (Relative Humidity)

ความชื้นของอากาศ หมายถึง ปริมาณไอน้ำที่ปะปนอยู่ในอากาศ ได้มาจากแหล่งน้ำต่างๆ ความชื้นในอากาศ มี 2 แบบ คือ

1. ความชื้นสัมบูรณ์

ความชื้นสัมบูรณ์หมายถึง อัตราส่วนระหว่างมวลของไอน้ำที่มีอยู่จริงในอากาศ กับปริมาตรของอากาศนั้น

ความชื้นสัมบูรณ์ = มวลของไอน้ำที่มีอยู่จริงในอากาศหารด้วยปริมาตรของอากาศ โดยมีหน่วยวัดเป็น กรัมต่อลูกบาศก์เมตร

2. ความชื้นสัมพัทธ์ (Relative Humidity)

ความชื้นสัมพัทธ์ หมายถึง ปริมาณเปรียบเทียบระหว่างมวลของไอน้ำที่มีอยู่จริงในอากาศขณะนั้น กับมวลของไอน้ำในอากาศอิ่มตัวที่อุณหภูมิและปริมาตรเดียวกัน นิยมคิดค่าเป็นร้อยละหรือ%

ความชื้นสัมพัทธ์ = มวลของไอนำที่มีอยู่จริงหารด้วยมวลของไอนำอิ่มตัวแล้วคูณด้วย100

เมื่อความชื้นในอากาศมาก หมายถึง ในอากาศมีไอน้ำมาก สามารถรับไอนำได้อีกเล็กน้อยเท่านั้น อากาศก็จะอิ่มตัว ซึ่งจะทำให้นำจากที่ต่างๆระเหยได้น้อย รวมทั้งตัวเราด้วย(เหงื่อ) ทำให้รู้สึกอึดอัดและเหนียวตัว ผ้าแห้งช้า โดยจะตรงข้ามกับอากาศแห้ง

เครื่องมือวัดความชื้นในอากาศเรียกว่า ไฮกรอมิเตอร์ โดยมีที่นิยมใช้อยู่ 2 ชนิดคือ

1.ไฮกรอมิเตอร์แบบเส้นผม เป็นเครื่องมือวัดความชื้นในอากาศโดยใช้เส้นผมที่สะอาดของมนุษย์ โดยอาศัยหลักการที่ว่า เส้นผมจะหดตัวหรือยืดตัว เมื่อความชื้นเปลี่ยนไป ถ้าความชื้นสัมพัทธ์สูง เส้นผมจะยืด แต่ถ้าความชื้นสัมพัทธ์ต่ำ เส้นผมจะหด โดยเส้นผมจะถูกต่อไว้กับเข็มชี้และอ่านค่าจากหน้าปัด แต่ถ้าเครื่องมือชนิดนี้มีนการแสดงค่าความชื้นสัมพัทธ์บนกระดาษกราฟจะเรียกว่า ไฮกรอกราฟ โดยจะมีการบันทึกต่อเนื่องตลอดเวลา

2.ไฮกรอมิเตอร์แบบกระเปาะเปียกและกระเปาะแห้ง หรือ ไซโครมิเตอร์โดยประกอบด้วยเทอร์มอมิเตอร์2 อัน โดยอันหนึ่งหุ้มกระเปาะด้วยผ้าชื้น เรียกว่ากระเปาะเปียก และจะมีการทำให้น้ำที่อยู่ในผ้าชื้นระเหย โดยใช้พัดลมเป่าหรือการแกว่ง ซึ่งจะปล่อยน้ำให้ระเหยจากผ้า

บทที่ 3
วิธีดำเนินการวิจัย

3.1 การออกแบบในส่วนของฮาร์ดแวร์

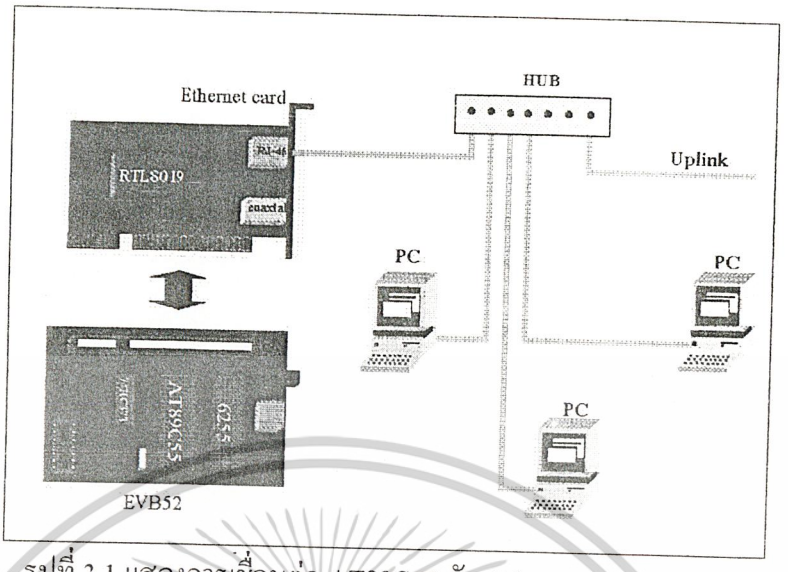
ใช้ แผงวงจร C52-EVB เป็น วงจรควบคุมการทำงานหลักของ โครงงานพิเศษนี้ โดยได้มีการเปลี่ยนตัวประมวลผล จาก AT89C52 เป็น AT89C55 เพื่อให้มีขนาดของ หน่วยความจำชนิดโปรแกรม(Program Memory)มากขึ้น เป็น 20 KBYTES และ มี หน่วยความจำชนิดข้อมูล(Data Memory)เท่าเดิมคือ 256 BYTES

3.1.1 การออกแบบวงจรเชื่อมต่อ Ethernet Card กับ AT89C55

Ethernet Card ที่ใช้เป็นแบบ ISA ซึ่งใช้ IC RTL8019 และปรับค่า การใช้งานอินเตอร์รัปเป็นหมายเลข 9 และตำแหน่งแอดเดรสหมายเลข 0x300 – 0x31F มี Register จำนวน 32 ตัว ในการนำ Ethernet Card นี้มาใช้กับ AT89C55 อ่านและเขียนค่า Register ใน Ethernet Card ทั้ง 32 ตัวได้ที่แอดเดรส 0x7FE0 – 0x7FFF ทำได้โดย

1. กำหนด SA5 - SA7, SA10 – SA19 ให้มีค่าเป็นลอจิก "0" โดยการต่อลงกราวด์และ SA8,SA9 มีค่าลอจิกเป็น "1" โดยการต่อไฟ +5 โวล์ต เพื่อกำหนดแอดเดรสให้เป็น 0x300 – 0x31F
2. ต่อ SA0 – SA4 ของ Ethernet Card เข้ากับ A0 – A4 ของ AT89C55 และ ต่อขา AEN ของ Ethernet Card เข้ากับ A15 ของ AT89C55
3. ต่อ SD0 – SD7 ของ Ethernet Card เข้ากับ D0 – D7 ของ AT89C55

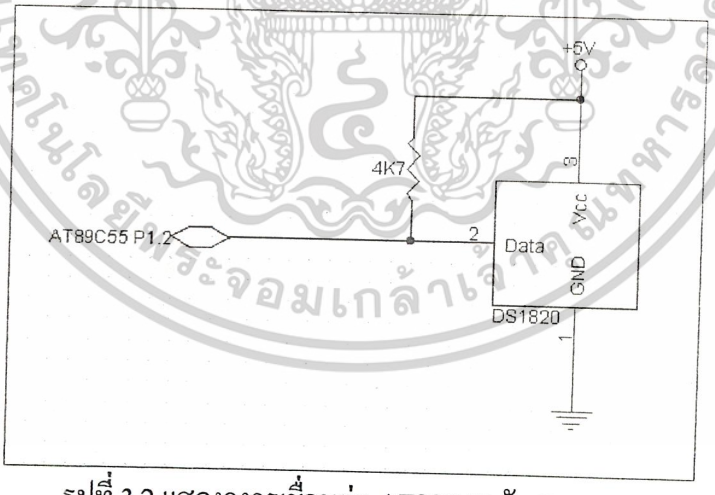
ต่อขาอินเตอร์รัปหมายเลข 9 เข้ากับขาอินเตอร์รัป 0 ของ AT89C55 โดยผ่าน Nand Gate ที่ต่อเป็นตัว Inverter เพื่อให้ใช้อินเทอร์รัปที่ลอจิก 0



รูปที่ 3.1 แสดงการเชื่อมต่อ AT89C55 กับ Ethernet Card

3.1.2 ออกแบบวงจรเชื่อมต่อ AT89C55 กับ ไอซีตรวจจับอุณหภูมิ DS1820

การเชื่อมต่อระหว่าง AT89C55 กับ DS1820 สามารถทำได้โดยใช้พอร์ต P1.2(PORT1 BIT 2)ของ AT89C55 ต่อกับขาที่ 2 (ONE WIRE BUS) ของ DS1820 โดยมีตัวต้านทานค่า 4.7kΩ ต่อ pull up กับไฟเลี้ยง +5 โวล์ต



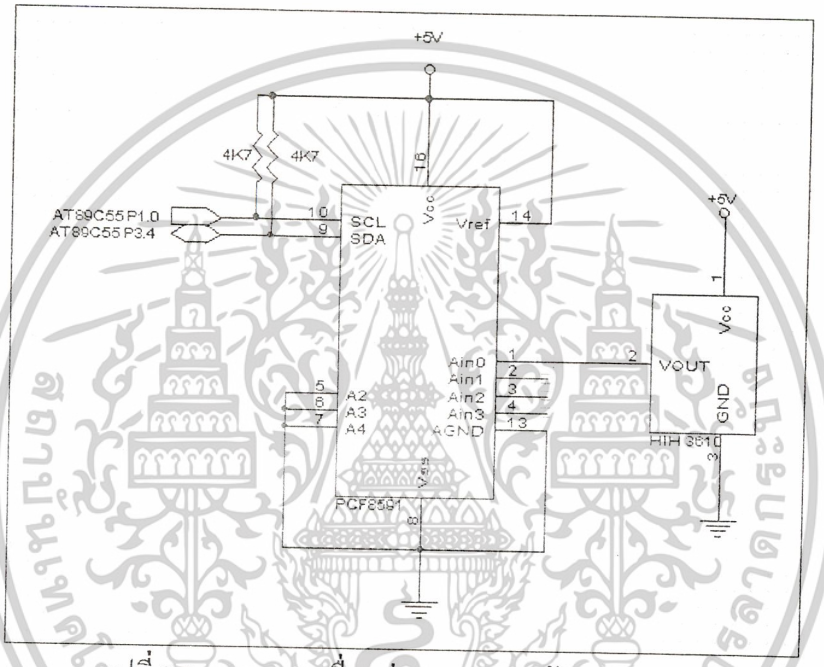
รูปที่ 3.2 แสดงวงจรเชื่อมต่อ AT89C55 กับ DS1820

3.1.3 ออกแบบวงจรเชื่อมต่อ AT89C55 กับ เซนเซอร์ตรวจวัดความชื้น HIH – 3610

การออกแบบวงจรอ่านค่าเปอร์เซนต์ความชื้นจากเซนเซอร์ความชื้น HIH – 3610 เนื่องจากเซนเซอร์ความชื้น HIH – 3610 จะให้เอาท์พุทออกมาเป็นแรงดัน ดังนั้นการที่จะนำค่าแรงดันซึ่ง

เป็นค่าอนาลอกมาให้ตัวไมโครคอนโทรลเลอร์ AT89C55 มาทำการคำนวณหาเปอร์เซ็นต์ความชื้น นี้ต้องมีการแปลงค่าอนาลอกให้เป็นค่าดิจิตอลก่อน

โดยได้เลือกใช้ IC PCF8591 ซึ่งเป็น ADC ขนาด 8 บิต 4 ช่องสัญญาณขาเข้า โดยใช้แรงดันอ้างอิง +5 โวล์ต แล้วใช้ช่องสัญญาณที่ 1 รับค่าแรงดันจากเซนเซอร์วัดความชื้น HIH-3610 และติดต่อกับไมโครคอนโทรลเลอร์ AT89C55 โดยใช้พอร์ต P1.0 และ P3.4 ของ AT89C55 ต่อกับ ขา SCK และ SDATA ของ PCF8591 ตามลำดับ เป็นการ จำลองการติดต่อ ในระบบบัส I^2C แสดงดังรูปที่



รูปที่ 3.3 แสดงวงจรเชื่อมต่อ AT89C55 กับ HIH-3610

3.2 การออกแบบในส่วนของซอฟต์แวร์

การออกแบบในส่วนของซอฟต์แวร์จะแบ่งออกเป็น 2 ส่วนนั่นคือ ซอฟต์แวร์ควบคุมการทำงานของไมโครคอนโทรลเลอร์ AT89C55 ซึ่งจะเกี่ยวข้องกับการสั่งตัวไมโครคอนโทรลเลอร์ AT89C55 นี้ไปอ่านค่าอุณหภูมิจากเซนเซอร์อุณหภูมิ หรืออ่านค่าความชื้นจากเซนเซอร์ความชื้น นอกจากนั้นยังรวมไปถึงการทำงานส่งข้อมูลแบบ SNMP โดยผ่านระบบเครือข่ายคอมพิวเตอร์ (Network System) โดยการทำงานรวมกับ Network Interface Card (NIC) หรือที่เรียกว่า Network Adapter โดยซอฟต์แวร์นี้จะต้องใช้ตัวคอมไพล์เลอร์ที่ทำการเปลี่ยนภาษาซีให้เป็นรหัสคำสั่งที่ตัวไมโครคอนโทรลเลอร์สามารถทำงานได้

ส่วนซอฟต์แวร์อีกส่วนหนึ่งเป็นส่วนการใช้งานติดต่อกันระหว่างอุปกรณ์หัววัด(ซึ่งตั้งอยู่กับที่ที่เรากำหนด) กับผู้ใช้งาน(อยู่กับเครื่องคอมพิวเตอร์ที่ใดก็ได้) เพื่อให้ผู้ใช้งานสามารถทราบค่า

ของอุณหภูมิ หรือ ความชื้นที่ระบบวัดนั้นตั้งอยู่ โดยทำการเรียกใช้โปรแกรมผ่านทางคอมพิวเตอร์ที่ต่ออยู่กับระบบเครือข่ายคอมพิวเตอร์ โปรแกรมตัวนี้จะทำงานบนคอมพิวเตอร์ ดังจะอธิบายต่อไป

3.2.1 ออกแบบโปรแกรมการอ่านค่าอุณหภูมิจากไอซีตรวจจับอุณหภูมิ DS1820

จากวงจรแสดงการเชื่อมต่อระหว่าง AT89C55 กับ DS1820 เราสามารถสั่งให้ตัวไมโครคอมโทรลเลอร์ไปทำการอ่านค่าอุณหภูมิจากตัว DS1820 ได้โดยทำการเขียนโปรแกรมสั่งงานซึ่งมีขั้นตอนดังนี้

เนื่องจาก DS1820 ใช้การติดต่อแบบระบบบัสหนึ่งสาย จึงต้องมีรูปแบบของการอ่านค่าอุณหภูมิตามระบบบัสหนึ่งสาย นั่นคือ

1. ต้องทำการส่งคำสั่งให้ DS1820 รู้ว่าจะทำการอ่านค่าอุณหภูมิโดยทำการส่งค่า 0xCC
2. จากนั้นก็ส่งคำสั่ง Convert T ซึ่งมีค่าเท่ากับ 0x44 เมื่อส่งคำสั่งนี้แล้วต้องให้เวลา DS1820 แปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลเก็บไว้ในสแครตซ์แพด ซึ่งใช้เวลาอย่างน้อยประมาณ 200 มิลลิวินาที
3. หลังจากส่งคำสั่ง Convert T ให้กับ DS1820 แล้ว ทำให้ตอนนี้มีค่าอุณหภูมิเก็บไว้ในสแครต์ซแพดแล้ว จึงต้องส่งคำสั่งอ่านข้อมูลจากสแกรต์ซแพด มีค่าเท่ากับ 0xBE เมื่อเมื่อส่งคำสั่งนี้ไปแล้ว DS1820 จะทยอยส่งข้อมูลอุณหภูมิออกมาทั้งหมด 9 ไบต์

ข้อมูลที่อ่านออกมาจาก DS1820 ในไบต์ที่ 0 และ 1 นั่นคือค่าอุณหภูมิมีขนาด 16 บิต ดังแสดงในตาราง เป็นตัวอย่างของค่าอุณหภูมิที่อ่านได้จาก DS1820 ซึ่งข้อมูลถูกส่งออกมาแบบอนุกรมบนสายเพียง 1 เส้น DS1820 สามารถวัดค่าอุณหภูมิได้ตั้งแต่ –55 °C ถึง +125 °C มีความละเอียด 0.5 °C

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	00000000 11111010	00FA
+25°C	00000000 00110010	0032h
$+^1/_2$°C	00000000 00000001	0001h
+0°C	00000000 00000000	0000h
$-^1/_2$°C	11111111 11111111	FFFFh
–25°C	11111111 11001110	FFCEh
–55°C	11111111 10010010	FF92h

ตารางที่ 3.1 แสดงตัวอย่างของค่าอุณหภูมิที่อ่านได้จาก DS1820

3.2.2 ออกแบบโปรแกรมการอ่านค่าเปอร์เซนต์ความชื้นจากเซนเซอร์ความชื้น

จากรูปที่ 3.3 แสดงวงจรการเชื่อมต่อระหว่าง AT89C55 กับ HIH-3610 การออกแบบโปรแกรมให้ไมโครคอนโทรลเลอร์อ่านค่าจาก HIH – 3610 นั่นต้องอ่านค่าดิจิตอลจาก ADC PCF8591 เป็น ADC ขนาดความละเอียด 8 บิตซึ่งใช้ Voltage Reference 5V ดังนั้น LSB (Least Significant Bit) จะมีค่าประมาณ 19.53 mV หรือ 5V/256

เมื่อได้ค่าดิจิตอลมาจาก ADC แล้ว ไมโครคอนโทรลเลอร์ก็จะนำค่ามาคำนวณหาค่าเปอร์เซ็นต์ความชื้น จากข้อมูลของ HIH-3610

Vout ที่ 0%RH คือ 0.958 V

Vout ที่ 75.3%RH คือ 3.268 V

ดังนั้น Slope = (3.268 – 0.958)/75.3

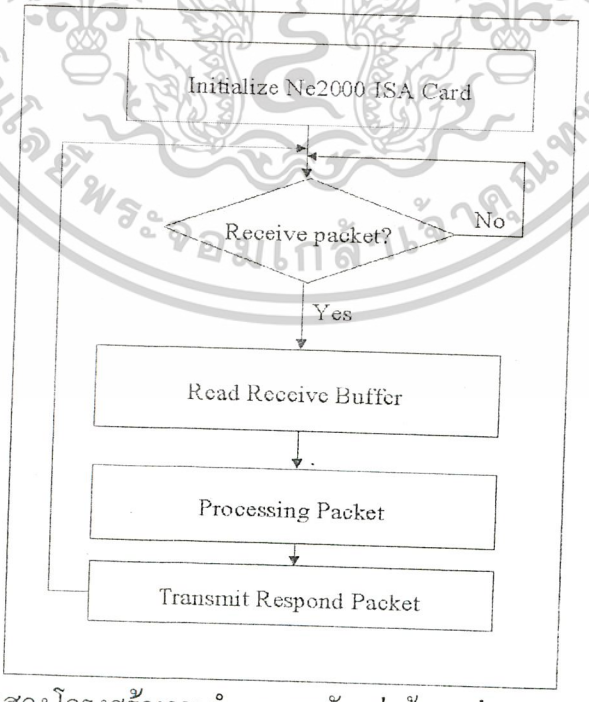
= 30.68mV/%RH

Zero offset = 0.958 V

ดังนั้น RH = (Vout – Zero offset)/Slope

= (Vout – 0.958)/0.03068

3.2.3 ออกแบบโปรแกรมติดต่อระหว่าง Ethernet card กับ AT89C55

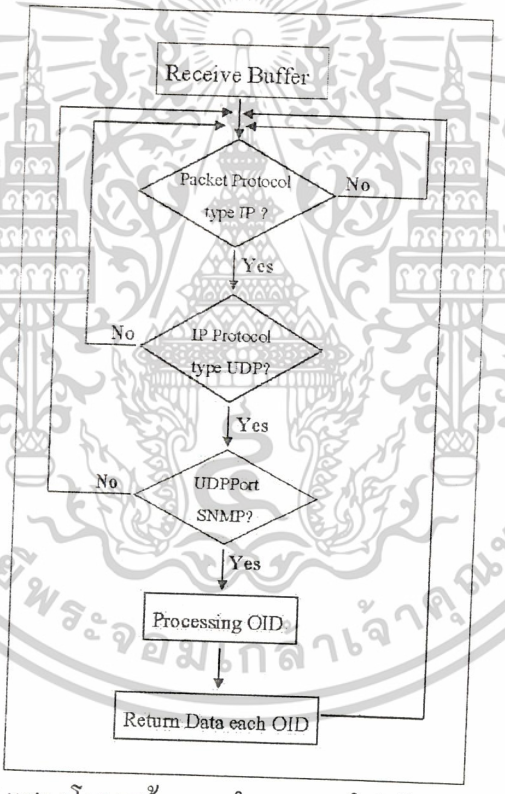


รูปที่ 3.4 แสดงโครงสร้างการทำงานการรับ-ส่งข้อมูลผ่าน Ethernet card

จากรูปที่ 3.4 แสดงโครงสร้างการทำงานการรับ-ส่งข้อมูลผ่าน Ethernet Card นั้น เราต้องเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์ AT89C55 ทำงานร่วมกับ Ethernet Card โดยเมื่อมี Packet เข้ามา AT89C55 ต้องทำการอ่าน Packet จาก Ethernet Card มาแล้วทำการตรวจสอบ Packet นั้น จากนั้นก็ส่ง Packet กลับไปหาต้นทางได้

ขั้นแรกต้องทำคือการเซ็ตค่ารีจีสเตอร์ต่างที่อยู่ใน Ethernet Card ซึ่งเป็นการเตรียมให้ Ethernet Card สามารถทำงานได้ กำหนดรูปแบบการรับ Packet จากแหล่งต่างๆ โดยกำหนดให้สามารถรับ Packet ทุกชนิด (Broadcast)

3.2.4 ออกแบบโปรแกรมในส่วนของการทำงาน SNMP



รูปที่ 3.5 แสดงโครงสร้างการทำงานของโปรโตคอล SNMP

จากรูปแสดงโครงสร้างการทำงานของโปรโตคอล SNMP สามารถอธิบายได้ดังนี้

1. เมื่อมี Packet ข้อมูลเข้ามาไมโครคอนโทรลเลอร์จะไปอ่านข้อมูลนั้นมา
2. จากนั้นจะทำการตรวจสอบ Packet นั้นเวลาเป็น Packet ชนิด IP (Internet Protocol) หรือไม่หากไม่ใช่ก็จะไม่สนใจ Packet นั้น แล้วกลับไปอ่าน Packet ต่อไปที่เข้ามา

3. เมื่อทำการตรวจสอบแล้ว เป็น Packet ชนิด IP จากนั้นทำการตรวจสอบต่อไปอีกว่าเป็น IP Protocol ชนิด UDP (User Datagram Protocol) หรือไม่ หากไม่ใช่ก็จะไปทำในขั้นตอนที่ 1 ใหม่

4. เมื่อ Packet นั้นเป็น IP Protocol ชนิด UDP แล้วก็ทำการตรวจสอบ Port ปลายทางว่าเป็น Port SNMP หรือไม่ หากไม่ใช่ ก็กลับไปทำในขั้นตอนที่ 1 ใหม่

5. หาก Port นั้นเป็น Port SNMP แล้วก็ทำการตรวจสอบภายใน Packet SNMP ในส่วนของ OID นั่นคือ OID เป็นส่วนที่ระบุตัวเซนเซอร์ที่ต้องการทราบค่า หากไมโครคอนโทรลเลอร์ทำการตรวจสอบ OID แล้วตรงกับ OID ของเซนเซอร์อุณหภูมิ ก็จะทำการส่งค่าอุณหภูมิกลับไปหายังเป้าหมายที่ทำการส่ง Packet นั้นมา หรือหาก OID นั้นตรงกับของเซนเซอร์ความชื้น ก็จะทำการส่งค่าความชื้นกลับมา

6. เมื่อทำการส่งข้อมูลกลับหาตัวต้นทางที่ทำการส่ง Packet มาแล้ว ไมโครคอนโทรลเลอร์ก็จะกลับไปอ่านค่า Packet อื่นๆ ต่อไป

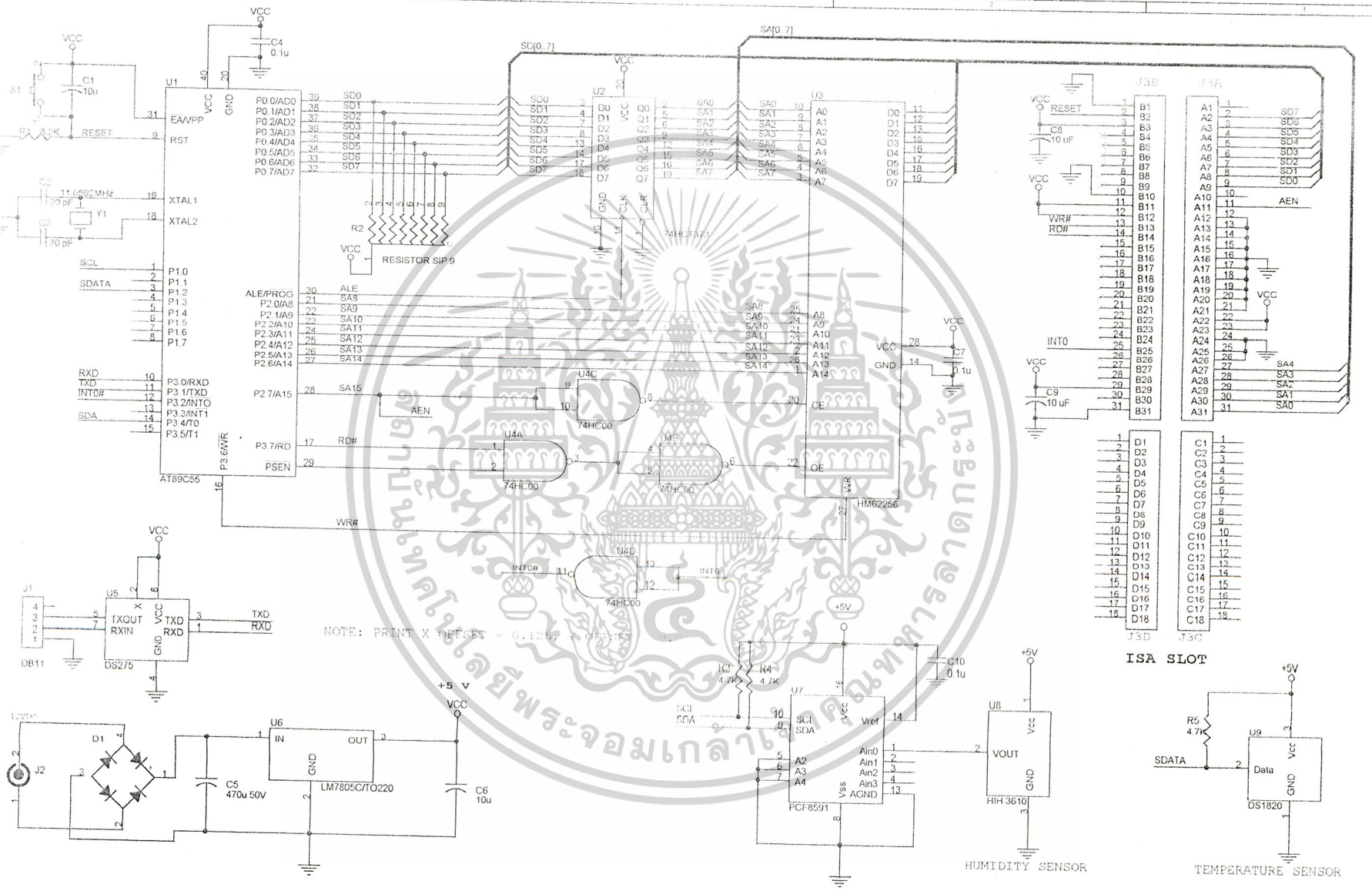
3.2.5 ออกแบบโปรแกรมติดต่อระหว่างผู้ใช้งานกับระบบวัด

เมื่อเราได้ทำการสร้างระบบวัดค่าอุณหภูมิและเปอร์เซ็นต์ความชื้น พร้อมทั้งสามารถทำงานส่งข้อมูลผ่านโปรโตคอล SNMP เรียบร้อยแล้ว ก็มาถึงในส่วนที่ใช้ติดต่อกับระบบวัดนั้นเพื่อเป็นส่วนรับข้อมูลมาแสดงผลให้ผู้ที่ต้องการทราบได้เห็น และเก็บข้อมูลไปวิเคราะห์

ซอฟต์แวร์ที่ทำหน้าที่ส่วนนี้พัฒนาขึ้นบนไมโครคอมพิวเตอร์ โดยทำการส่งค่ามาตราฐานโปรโตคอล SNMP เพื่อทำการเรียกดูค่าจากระบบวัด อันประกอบไปด้วย OID (Object identifier) สำหรับอ่านค่า อุณหภูมิและเปอร์เซ็นต์ความชื้น และนำค่าอุณหภูมิ และ เปอร์เซ็นต์ความชื้นที่ได้รับมาจากตัวเครื่องมือมาแสดงผลทางจอภาพ (Terminal) และสามารถทำการเก็บข้อมูลได้จากตัวโปรแกรมนี้ การออกแบบโปรแกรมตัวนี้เขียนโดยใช้โปรแกรม Visual Basic โดยสามารถทำการเก็บข้อมูลที่อ่านได้ไว้ในรูปของไฟล์ข้อมูลเพื่อนำมาวิเคราะห์ได้ในภายหลัง

โดยโปรแกรม SNMP Browser นี้ประกอบไปด้วย

1. IP Address ปลายทางที่ต้องการส่ง Packet SNMP ไป
2. Community Name
3. OID ที่ต้องการทำการส่ง



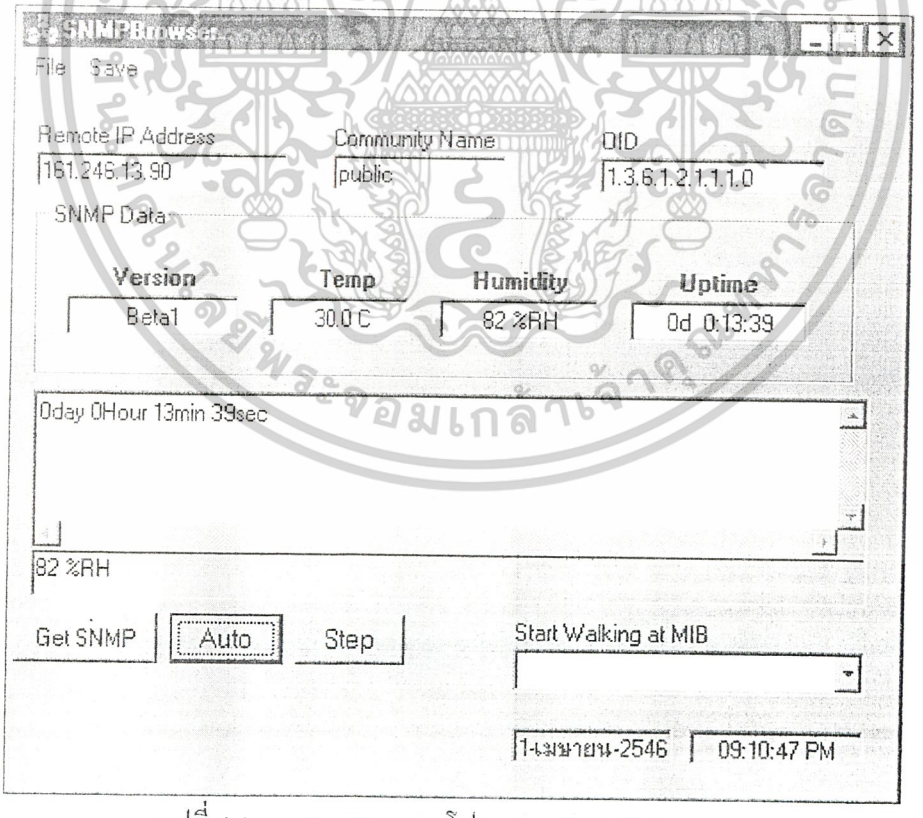
รูปที่ 3.6 แสดงวงจรรวมของแผงวงจร ไมโครคอนโทรเลอร์ AT89C55

บทที่ 4

ผลการทดลองและอภิปลาย

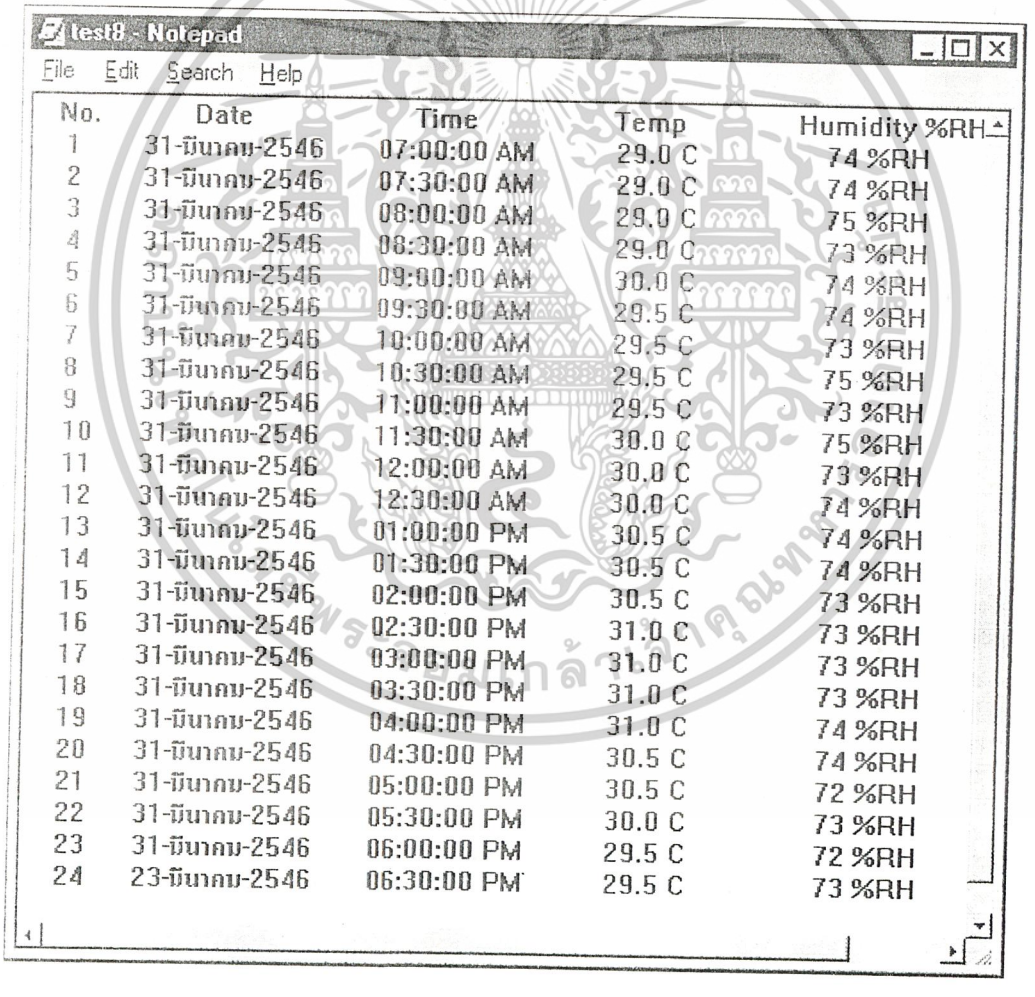
4.1 การทดลองโปรแกรม SNMP Browser

หลังจากได้ทำการเขียนโปรแกรมควบคุมการทำงานของ AT89C55 เกี่ยวกับการทำงานอ่านค่าอุณหภูมิจากเซนเซอร์อุณหภูมิ DS1820 และ อ่านเปอร์เซ็นต์ความชื้นจากเซนเซอร์ความชื้น humidity sensor HIH - 3610 โดยผ่านไอซีแปลงสัญญาณอนาลอกเป็นดิจิตอลแล้วสามารถส่งค่าในรูปแบบ protocol SNMP จากนั้น download program ลงไปในชิป แล้วนำมาใส่ใน บอร์ดเรียนรู้ EVB52 ที่ได้เชื่อมต่อ Ethernet card ที่ได้ต่อสาย Coaxial จาก hub แล้วใช้โปรแกรม SNMPBrowser โดยใส่ค่า Remote IP Address เป็น "161.246.13.90" และ Community Name เป็น "public" และ OID เป็น "1.3.6.1.2.1.1.1.0" เพื่อทำการอ่านค่า SysName จากระบบวัดหรือถ้าเปลี่ยนค่า OID เป็น "1.3.6.1.2.1.1.8.0" เพื่อทำการอ่านค่าอุณหภูมิ และ "1.3.6.1.2.1.1.9.0" เปอร์เซ็นต์ความชื้น ซึ่งได้ค่าดังรูป



รูปที่ 4.1 แสดงการทดลองโปรแกรม SNMP Browser

เมื่ออ่านค่าอุณหภูมิและเปอร์เซ็นต์ความชื้นจากโปรแกรม SNMPBrowser ได้ดังรูปที่ 4.1 แล้ว ได้ทดลองเก็บค่าอุณหภูมิและเปอร์เซ็นต์ความชื้น ให้เป็นไฟล์ข้อมูลโดยโปรแกรม SNMPBrowser สามารถทำการจัดเก็บข้อมูลที่อ่านมาได้ พร้อมกับวันและเวลา บันทึกเป็นไฟล์ข้อมูล (text file) โดยทำการกดปุ่ม Save จากนั้นโปรแกรมจะให้ตั้งชื่อไฟล์ที่ต้องการเก็บ หลังจากนั้นโปรแกรมจะทำการอ่านค่าอุณหภูมิและความชื้น และบันทึกลงไฟล์ที่เลือกไว้ทุกๆ 3 นาที และเมื่อต้องการหยุดเก็บข้อมูลให้ทำการกดปุ่ม Save แล้วตามด้วย Stop ซึ่งไฟล์ข้อมูลที่ได้จากโปรแกรม SNMPBrowser ดังรูปที่ 4.2 ซึ่งแสดงข้อมูลที่เก็บได้ทุกๆ ครึ่งชั่วโมง จากนั้นได้นำข้อมูลที่ได้มาสร้างเขียนกราฟระหว่างค่าอุณหภูมิ และ ค่าเปอร์เซ็นต์ความชื้น กับ เวลา ดังรูปที่ 4.3 และ 4.4 ตามลำดับ

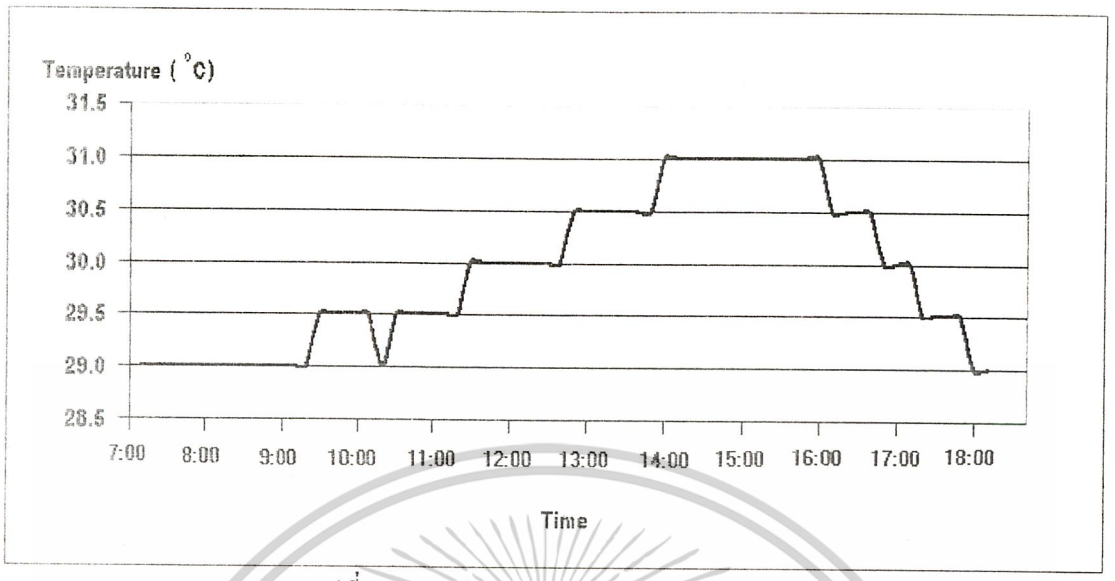


test8 - Notepad

File Edit Search Help

No.	Date	Time	Temp	Humidity %RH
1	31-มีนาคม-2546	07:00:00 AM	29.0 C	74 %RH
2	31-มีนาคม-2546	07:30:00 AM	29.0 C	74 %RH
3	31-มีนาคม-2546	08:00:00 AM	29.0 C	75 %RH
4	31-มีนาคม-2546	08:30:00 AM	29.0 C	73 %RH
5	31-มีนาคม-2546	09:00:00 AM	30.0 C	74 %RH
6	31-มีนาคม-2546	09:30:00 AM	29.5 C	74 %RH
7	31-มีนาคม-2546	10:00:00 AM	29.5 C	73 %RH
8	31-มีนาคม-2546	10:30:00 AM	29.5 C	75 %RH
9	31-มีนาคม-2546	11:00:00 AM	29.5 C	73 %RH
10	31-มีนาคม-2546	11:30:00 AM	30.0 C	75 %RH
11	31-มีนาคม-2546	12:00:00 AM	30.0 C	73 %RH
12	31-มีนาคม-2546	12:30:00 AM	30.0 C	74 %RH
13	31-มีนาคม-2546	01:00:00 PM	30.5 C	74 %RH
14	31-มีนาคม-2546	01:30:00 PM	30.5 C	74 %RH
15	31-มีนาคม-2546	02:00:00 PM	30.5 C	73 %RH
16	31-มีนาคม-2546	02:30:00 PM	31.0 C	73 %RH
17	31-มีนาคม-2546	03:00:00 PM	31.0 C	73 %RH
18	31-มีนาคม-2546	03:30:00 PM	31.0 C	73 %RH
19	31-มีนาคม-2546	04:00:00 PM	31.0 C	74 %RH
20	31-มีนาคม-2546	04:30:00 PM	30.5 C	74 %RH
21	31-มีนาคม-2546	05:00:00 PM	30.5 C	72 %RH
22	31-มีนาคม-2546	05:30:00 PM	30.0 C	73 %RH
23	31-มีนาคม-2546	06:00:00 PM	29.5 C	72 %RH
24	23-มีนาคม-2546	06:30:00 PM	29.5 C	73 %RH

รูปที่ 4.2 แสดงข้อมูลที่เก็บได้จากโปรแกรม Snmp Browser



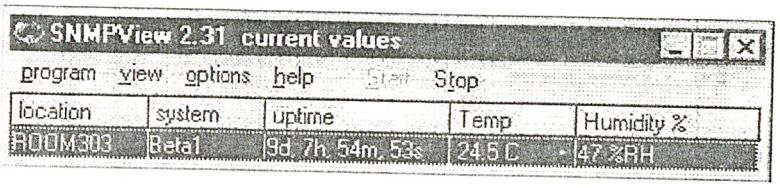
รูปที่ 4.3 กราฟระหว่างค่าอุณหภูมิ กับ เวลา

รูปที่ 4.4 กราฟระหว่างค่าเปอร์เซ็นต์ความชื้นกับเวลา

4.2 การทดลองโปรแกรม SNMP อื่นๆ

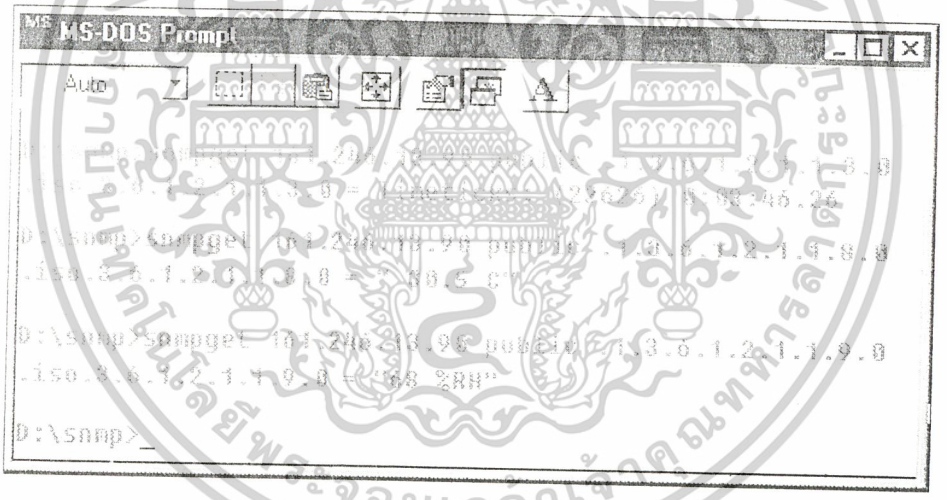
เมื่อได้ทดลองโดยใช้ โปรแกรม SNMP Browser ทำการรับค่าอุณหภูมิและค่าเปอร์เซ็นต์ความชื้นจากระบบวัดแล้ว เมื่อทดลองใช้โปรแกรมอื่นๆ ที่ใช้ Protocol SNMP ในการรับ-ส่งข้อมูลมาทำการติดต่อกับ ระบบวัด โดยตั้งค่าเหมือน SNMP Browser คือ Remote IP Address ให้เป็น IP ของระบบวัดคือ "161.246.13.90" และใส่ Community เป็น "Public" และกำหนดค่า OID ให้ตรงกับค่าที่เรากำหนดไว้คือ "1.3.6.1.2.1.8.0" สำหรับค่าอุณหภูมิ และ "1.3.6.1.2.1.9.0" สำหรับค่า

เปอร์เซ็นต์ความชื้น ผลจากการใช้โปรแกรม SNMPView ติดต่ออ่านค่าอุณหภูมิ กับ เปอร์เซ็นต์ความชื้นได้ผลดังรูป



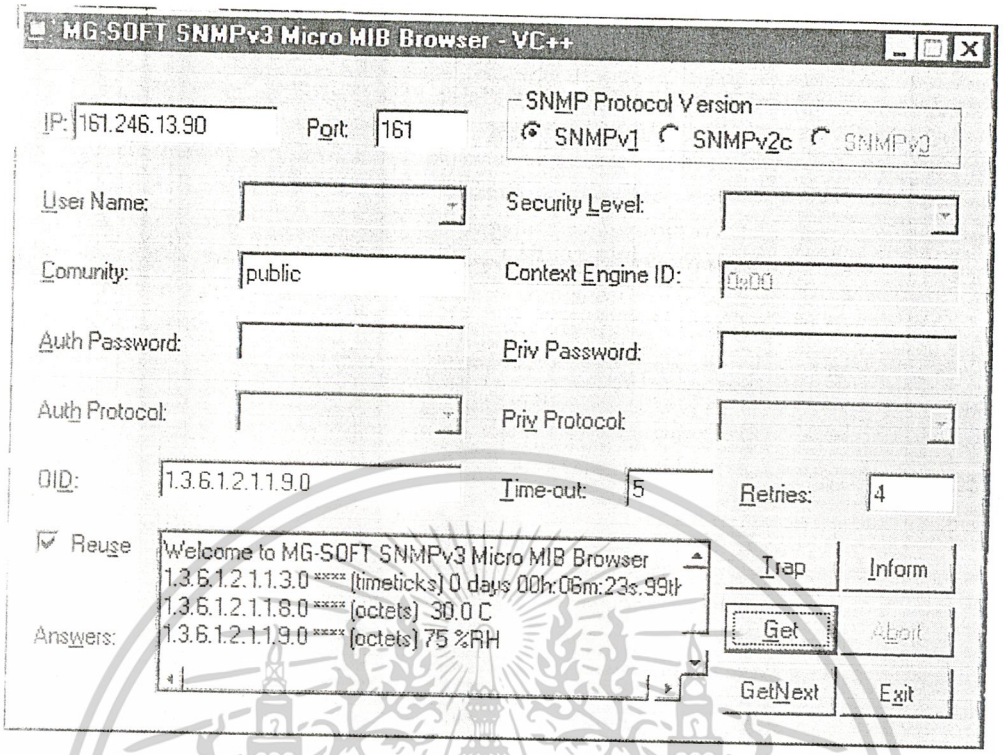
รูปที่ 4.5 แสดงผลจากการทดลองด้วยโปรแกรม SNMPView

จากรูปที่ 4.5 แสดงผลด้วยโปรแกรม SNMPView มีรูปการแสดงผลข้อมูลที่สวยงามและอ่านค่าได้ง่าย อีกทั้งยังสามารถทำการปรับค่าระยะเวลาในการเรียกดูข้อมูล และส่งข้อมูลที่อ่านได้ทางอีเมลล์ที่ป้อนไว้ได้อีกด้วย



รูปที่ 4.6 แสดงผลที่จากการรันโปรแกรม SNMP แบบ Command Line

จากรูปที่ 4.6 นี้เป็นการใช้คำสั่ง snmpget เพื่อทำการส่งคำสั่งไปร้องขอข้อมูลจาก SNMPAgent เมื่อได้ข้อมูลมาแล้วก็นำมาแสดงผลตามชนิดของข้อมูลที่ได้รับมา เช่น Timeticks หรือ String ซึ่งคำสั่งนี้รันบนระบบปฏิบัติการ DOS เป็นโปรแกรม SNMP ขั้นพื้นฐานที่ใช้กันทั่วไป



รูปที่ 4.7 แสดงผลที่ได้จากการใช้โปรแกรม MG-SOFT SNMPv3 Micro MIB Browser

จากการทดลองด้วยโปรแกรมต่างๆ ที่อาศัยการรับ-ส่งข้อมูลโดยมีรูปแบบเป็นชนิด SNMP พบว่าสามารถอ่านค่าจากระบบวัด(Measurement System On Computer Network) ได้ เนื่องจากตัวระบบวัดนี้ใช้โปรโตคอล SNMP ซึ่งเป็นมาตราฐานทั่วไปที่ทุกคนใช้กัน จึงทำให้สามารถติดต่อกับโปรแกรม SNMP ต่างๆ ได้เป็นอย่างดี

บทที่ 5
สรุปและข้อเสนอแนะ

โครงงานพิเศษนี้สามารถทำการส่งข้อมูลผ่านระบบเน็ตเวิร์กโดยใช้มาตราฐานโปรโตคอล SNMP โดยสามารถส่งค่าอุณหภูมิ และค่าเปอร์เซ็นต์ความชื้นจากที่ที่ตั้งอุปกรณ์ไว้ และสามารถเรียกดูได้จากเครื่องคอมพิวเตอร์ที่ต่อกับระบบเน็ตเวิร์กไว้ โดยแสดงผลข้อมูลผ่านโปรแกรมที่ใช้รูปแบบการสื่อสารตามมาตราฐานโปรโตคอล SNMP ทำให้มีความสะดวกในการเก็บข้อมูลให้แหล่งที่อยู่ห่างไกลที่มนุษย์ไม่สามารถไปเก็บข้อมูลได้ด้วยตัวเองได้อย่างบ่อยๆ และยังเป็นการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ ตระกูล MCS – 51 มาใช้งานแทนเครื่องคอมพิวเตอร์ที่มีราคาแพงและมีขนาดใหญ่

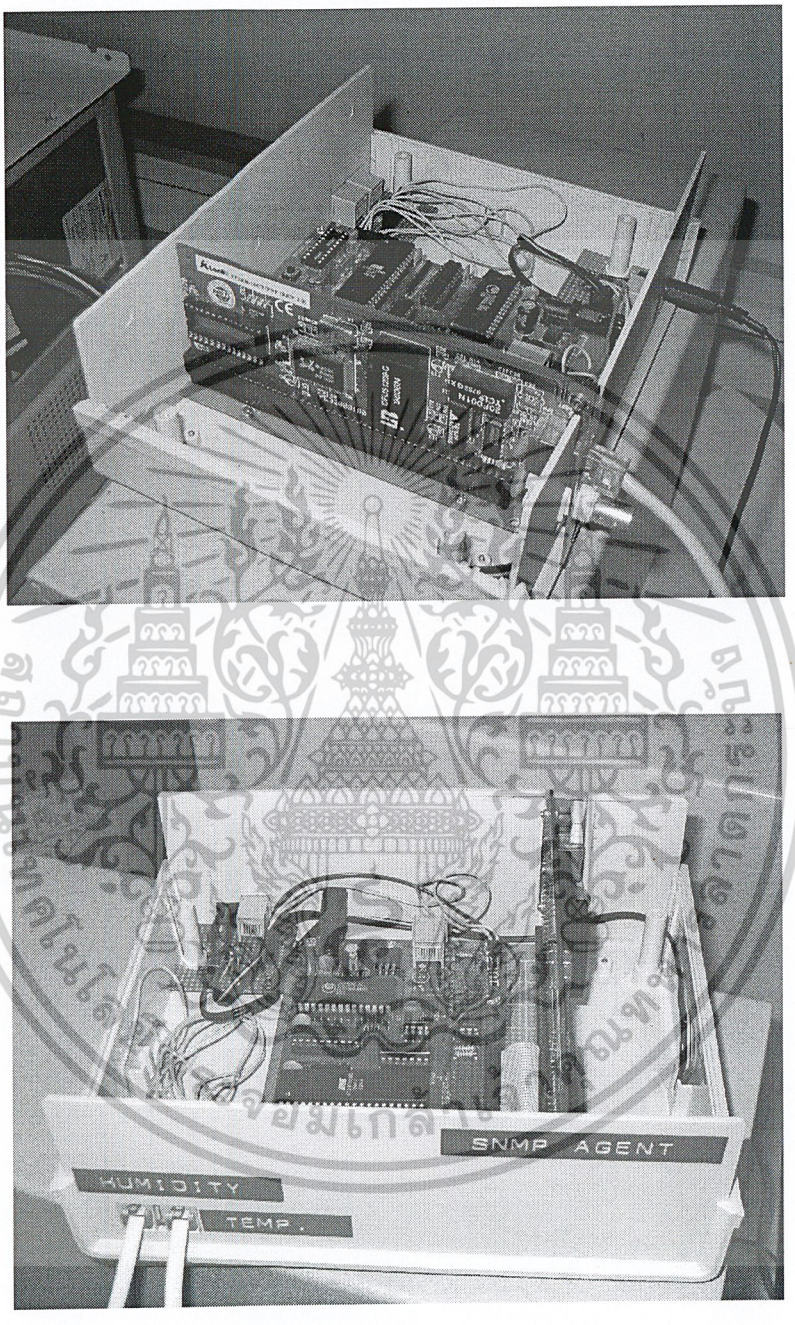
เนื่องจากสามารถทำการเก็บค่าอุณหภูมิและเปอร์เซนต์ความชื้น กับ เวลาขณะที่ทำการอ่านค่ามาเก็บไว้ในรูปของไฟล์ข้อมูล จึงสามารถนำเอาค่าอุณหภูมิและค่าเปอร์เซ็นต์ความชื้นนั้น มาเขียนกราฟกับเวลา จะทำให้เราสามารถเห็นสภาพอุณหภูมิและเปอร์เซ็นต์ความชื้นโดยรวมของห้องหรือ บริเวณที่เราทำการตั้งระบบวัดไว้ ทำให้สามารถวิเคราะห์ว่าช่วงเวลาใดมีการเปลี่ยนแปลงค่าอุณหภูมิ หรือ เปอร์เซ็นต์ความชื้นมากที่สุด หรือน้อยที่สุด โดยสามารถปรับเปลี่ยนระยะเวลาในการบันทึกค่าได้เพื่อเพิ่มความละเอียดในช่วงเวลาสั้นๆ

ข้อเสนอแนะและแนวทางการพัฒนา

1. ระบบวัดที่สร้างขึ้นมามีขนาดเล็ก สามารถนำไปใช้งานที่ที่มนุษย์ไม่สามารถอยู่ได้เป็นเวลานาน และสามารถสั่งเก็บข้อมูลเพื่อนำมาใช้ในการวิเคราะห์ ได้อย่างสะดวกสบาย
2. มีราคาถูก เมื่อเทียบการใช้คอมพิวเตอร์เป็นตัวทำงานแทนไมโครคอนโทรลเลอร์
3. ระบบวัดที่สร้างขึ้นมานี้เป็นต้นแบบของตัว Agent สามารถนำไปประยุกต์ใช้ในมีความสามารถมากขึ้น เช่น อาจเพิ่มเซนเซอร์ชนิดอื่นๆ เข้าไปอีก หรือ สามารถอ่านค่าได้โดยใช้ Web Browser ที่ง่ายและเป็นที่นิยมทั่วไปเช่น Microsoft Internet Explore เป็นต้น
4. เนื่องจากรูปแบบที่นำมาใช้ในการติดต่อสื่อสารกันระหว่างเครื่องมือ กับ ผู้ใช้งานใช้โปรโตคอล UDP ซึ่งเป็นแบบ Connectionless นั่นคือ ไม่มีการสถาปนาการเชื่อมต่อกันก่อนรับ-ส่งข้อมูลกัน หากข้อมูลสูญหาย ซ้ำซ้อน หรือ ลำดับไม่ถูกต้อง UDP จะปล่อยให้โปรโตคอลที่เรียกใช้งานดำเนินการกับปัญหาเหล่านี้เอง ทำให้ความน่าเชื่อถือในการที่จะได้รับข้อมูลนั้นมีน้อยกว่า ชนิดที่มีการเชื่อมต่อกันก่อน รับ-ส่งข้อมูล หากเปลี่ยนมาใช้โปรโตคอล TCP โดย TCP ทำหน้าที่ส่งข้อมูลโดย

รับประกันความเชื่อถือ TCP ด้านส่งต้องส่ง Packet ซ้ำใหม่หาก Packet สูญหาย TCP ด้านรับมีหน้าที่จัด Packet ให้ถูกต้องตามลำดับและกำจัด Packet ซ้ำซ้อน TCP เป็นโปรโตคอลแบบ "connection oriented" จะทำให้มีความน่าเชื่อถือว่าข้อมูลจากตัวส่งจะไปถึงผู้รับแน่นอน

ภาคผนวก



รูปแสดงส่วนประกอบของ SNMPAgent

(Measurement System on Computer Network)

code โปรแกรมภาษาซีควบคุมการทำงานของ AT89c55

```
//---------------------------------- ip.h ----------------------------------------
//Function Prototype
short IPcksum(unsigned short *buf, int nLength);
void testProcess(void);
void testSendUdp(char *pStr);
void doUDPecho(char *pStr);
void doSNMP(void);
//---------------------------- udp.h ----------------------------------------
unsigned short UDP_TCPchksum(unsigned char *pBuf, short len);
unsigned short udpchksum(unsigned char *pBuf);

// ---------------------- i2c.h ----------------------------------------
#define ACK   1
#define NO_ACK 0

void i2c_start(void);
void i2c_stop(void);
char i2c_write(unsigned char output_data);
char i2c_read(char ack);
// ---------------------------------- hih3610.h ----------------------------------
unsigned char humidity_read(void);

//---------------------------------- ds1820.h ----------------------------------
void ds1820Start(void);
int ds1820ReadTemp(void);
int ds1820ReadWaitTemp(void);

// ---------------------------------- pcf8591.h ----------------------------------
void dac(char value);
unsigned char adc(char channel);
```

```
#include <at89x52.h>
#include <stdio.h>
#include "rtl8019.h"
#include "ip.h"
#include "ds1820.h"
#include "hih3610.h"

extern idata unsigned char myMAC[6];
extern xdata unsigned char msgTxBuff[1024];   //Tx frame buffer

char destMac[6] = {0x00,0x40,0x05,0x48,0x50,0x86};

idata unsigned char myIP[4] = {161,246,13,90};//{161,246,45,30};//{10,1,1,9};
idata unsigned char tick10msec;
idata unsigned long tickSec;
idata unsigned long nextTick;

idata unsigned long tempTick;
int TempData;
char bTemp;                          // for DS1820start
unsigned char humidityValue;

// Prototype
void doTemp(void);
void doHumidity(void);

void print_mac(void)
{
  char i;

  printf("MAC :");
  for(i=0; i < 6; i++) {
    printf("%c%02bx", (i?'-':' '), myMAC[i]);
  }
```

```
    putchar('\n');
  }
  void printIP(char *msg, char *ip)
  {
    unsigned char i;
    printf(msg);
    for(i = 0; i < 4; i++) {
      printf("%c%bu",(i?'.':' '), ip[i]);
    }
    putchar('\n');
  }

  void serial_init(void)
  {
    TMOD = (TMOD & 0x0f) | 0x20; // TMOD: timer 1, mode 2, 8-bit reload
    TH1 = 0xfd;          // TH1:  reload value for 9600 baud @ 11.0592MHz
    TL1 = 0xfd;
    TR1 = 1;
    SCON = 0x52;
  }

  void timer_init(void)
  {
    TMOD = (TMOD & 0xf0) | 0x09; // Set Mode (16-bit timer )
    TH0 = 0xdc;//0xdc; // Reload TL1 to count 100 clocks
    TL0 = 0x00;//0x00;
    ET0 = 1; // Enable Timer 1 Interrupts
    TR0 = 1; // Start Timer 1 Running
  }

  void main(void)
  {
    EA = 0;
    EX0 = 0;
```

```
    timer_init();
    serial_init();
    nic_init();
    print_mac();
    printIP("my IP :",myIP);
    EA = 1;
    for (;;){
        testProcess();
        doTemp();
        doHumidity();
            if(nextTick <= tickSec) {
                nextTick+=1;
                printf("tickSec:%lu\n", tickSec);
                P1_7 ^= 1;
            }
    }
}
void doTemp(void)
{
        static char tempState = 0;
        if(tempState == 0) {
                ds1820Start();
                tempTick = 0;
                tempState = 1;
        }
        else if(tempState == 1) {
                if(tempTick >= 100) {
                        TempData = ds1820ReadTemp();
                        tempState = 0;

                }
        }
}
```

```
void doHumidity(void)
{
//  printf("Humidity = %bu %cRH\n",humidity_read(),'%');
  humidityValue = humidity_read();
}


void isr_timer0 (void) interrupt 1
{
        TH0 = 0xdc;//0xdc;
        TL0 = 0x00;//0x00;
        tick10msec++;
        if(tick10msec >= 100) {                    //every 1 sec
    tick10msec = 0;
    tickSec++;
  }
  tempTick++;
}
```

```
#include <at89x52.h>
#include <stdio.h>
#include <intrins.h>
#include "i2c.h"
#define TRUE 1
#define FALSE 0
#define   SDATA                 P3_4                    // Serial data
#define   SCLK                  P1_0                    // Serial clock
#define     HIGH      0x01                  // Value representing ON
#define               LOW       0x00                    // Value representing OFF

//-------------------------------------------------------------------
// I2C Functions - Bit Banged
//-------------------------------------------------------------------
//-------------------------------------------------------------------
//          Routine: i2c_start
//          Inputs:           none
//          Outputs: none
//          Purpose: Sends I2C Start Trasfer - State "B"
//-------------------------------------------------------------------

void i2c_delay(void)
{
          _nop_();
          _nop_();
          _nop_();
          _nop_();

}
void i2c_start (void)
{
          SDATA = HIGH;                                 // Set data line high
          SCLK = HIGH;                                  // Set clock line high
          i2c_delay();
```

```
        SDATA = LOW;                                    // Set data line low (START SIGNAL)
        SCLK = LOW;                                     // Set clock line low
}
//-------------------------------------------------------------------------
//      Routine: i2c_stop
//      Inputs:         none
//      Outputs: none
//      Purpose: Sends I2C Stop Trasfer - State "C"
//-------------------------------------------------------------------------
void i2c_stop (void)
{
        unsigned char input_var;
        SCLK = LOW;                                     // Set clock line low
        SDATA = LOW;                                    // Set data line low
        i2c_delay();
        SCLK = HIGH;                                    // Set clock line high
        SDATA = HIGH;                                   // Set data line high (STOP SIGNAL)
        input_var = SDATA;                              // Put port pin into HiZ
}
char i2c_write (unsigned char output_data)
{
        unsigned char index;
        for(index = 0; index < 8; index++)      // Send 8 bits to the I2C Bus
        {
// Output the data bit to the I2C Bus
                SDATA = ((output_data & 0x80) ? 1 : 0);
                output_data <<= 1;              // Shift the byte by one bit
                SCLK = HIGH;                                    // Clock the data into the I2C Bus
                i2c_delay();
                SCLK = LOW;
        }
        SDATA = HIGH;
        index = SDATA;                                  // Put data pin into read mode
        SCLK = HIGH;                                    // Clock the ACK from the I2C Bus
```

```
        i2c_delay();
        index = SDATA;                          // read ack bit
        SCLK = LOW;
        return (index ? 0 : 1); //return 1 if receive ack
}
char i2c_read (char ack)
{
        unsigned char index, input_data;
        index = SDATA;                          // Put data pin into read mode
        input_data = 0x00;
        for(index = 0; index < 8; index++)      // Send 8 bits to the I2C Bus
        {
                input_data <<= 1;               // Shift the byte by one bit
                SCLK = HIGH;                    // Clock the data into the I2C Bus
                i2c_delay();
                input_data |= SDATA;            // Input the data from the I2C Bus
                _nop_();
                SCLK = LOW;
        }
        if(ack)  //ACK
                SDATA = LOW;
        else                    // NO ACK
                SDATA = HIGH;
        SCLK = HIGH;
        i2c_delay();
        SCLK = LOW;
        SDATA = HIGH;

 return input_data;
}
```

```
#include "i2c.h"
#include "pcf8591.h"

#define I2C_PCF8591_BASE 0x90
#define I2C_PCF8591_ADDR (I2C_PCF8591_BASE + 0) // A0 = 1, A1 = A2 = 0


/* read ADC value of channel: 0, 1, 2 and 3 */
unsigned char adc(char channel)
{
  unsigned char value;

  i2c_start();// Send I2C Start Transfer
  (void)i2c_write(I2C_PCF8591_ADDR);     // Send identifier I2C address - Write
  (void)i2c_write(0x40 + channel);       // Send control byte to device (last 2 bits is the channel)
  i2c_stop();                            // Send I2C Stop Transfer

  i2c_start();                           // Send I2C Start Transfer
  (void)i2c_write(I2C_PCF8591_ADDR + 1); // Send identifier I2C address - Read
  (void)i2c_read(ACK);                   // Sampling conversion

  value = i2c_read(NO_ACK);              // Read adc value
  i2c_stop();                            // Send I2C Stop Transfer

  return value;
}
```

```
#include <at89x52.h>
#include <stdio.h>
#include "i2c.h"
#include "pcf8591.h"
unsigned char humidity_read(void)
{
        unsigned int value;
  char i;

//  percent = 100 * (Vo - 0.8) / (3.9-0.8);
//  percent = 100 * (adc - 41) / (199 - 41);

        i = adc(0);

  value = (unsigned char)adc(0);

  if(value < 49)
    value = 49; // 0%

  if(value > 199)
    value = 199; // 100%

  value = (100 * (value - 41) / (199 - 41));

  return (unsigned char)value;
}
```

ds1820.c

```
#include <at89x52.h>
#include <intrins.h>

#define SDATA P1_2
extern int tick;
extern int TempData;
extern idata unsigned char tick10msec;
extern idata unsigned long tickSec;
extern idata unsigned long nextTick;
char TempBuff[100];
extern char next;

void delay20us()
{
 unsigned char i;
 i = 9;
 while(--i)
   ;
}
void delay60us()
{
 unsigned char i;
 i = 14;
 while(--i)
   ;
}

void delay500us()
{
 unsigned char i;
 i = 250;
 while(--i)
   ;
}
```

```
char ds1820ReadByte()
{
  char i;
  char Byte;

  for(i = 0; i < 8; i++) {
    Byte >>= 1;
    SDATA = 0;
    _nop_();
    _nop_();
    SDATA = 1;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    if(SDATA)
      Byte |= 0x80;
    else
      Byte &= ~0x80;
    delay60us();
  }
  return Byte;
}
void ds1820WriteByte(char Byte)
{
 char i;
 EA = 0;
 for(i = 0; i < 8; i++) {
  if(Byte & 0x01) {
    SDATA = 0;
    _nop_();
    _nop_();
    _nop_();
    _nop_();
```

```
        SDATA = 1;
        delay60us();
      }
      else {
        SDATA = 0;
        delay60us();
        SDATA = 1;
        _nop_();
        _nop_();
        _nop_();
        _nop_();
      }
      Byte >>= 1;
    }
    EA = 1;
}

void reset_pulse()
{
    SDATA = 0;
    delay500us();
    SDATA = 1;
    delay20us();
}

char presence_pulse()
{
        char i,j;
        for(i=0;i<10;i++) {
                if(SDATA)
                        j = 0;
// SDATA is HIGH
                else {
                        j = 1;
// SDATA is LOW
                        break;
                }
        }
        if(j == 1) {
                for(i=0; i < 10; i++) {
                        if(!SDATA)
                                j = 0;
                        else {
                                j = 1;
                                return 1;
// Complete presence pulse
                        }
                }
        }
        else return 0;
}

void ds1820Start()
{
  ds1820WriteByte(0xcc);  // skip ROM command
  ds1820WriteByte(0x44);
  //Initiates temperature conversion
}

int ds1820ReadTemp()
{
  int temp;
  reset_pulse();  //Tx reset pulse
        if(presence_pulse()){  //Rx presence pulse
          ds1820WriteByte(0xcc);
// skip ROM command
          ds1820WriteByte(0xbe);
// Read scratchpad command
```

```
            temp = ds1820ReadByte(); //(LSB)
            temp = temp + (ds1820ReadByte() << 8);
//(MSB)
             reset_pulse();  //Tx reset pulse
             presence_pulse();  //Rx presence pulse

             return temp;
           }
          else return 0;
}
```

```
#include <at89x52.h>
#include <stdio.h>
#include <string.h>
#include <absacc.h>
#include "rtl8019.h"
#include "ip.h"

#define FRAME_TYPE_IP      0x0800
#define FRAME_TYPE_ARP     0x0806
#define FRAME_TYPE_RARP    0x8035
#define FRAME_TYPE_NETWARE 0x8137

extern idata unsigned char myMAC[6];
extern idata unsigned char myIP[4];
extern xdata unsigned char msgRxBuff[1024];
extern xdata unsigned char msgTxBuff[1024];
void doARP(void)
{
  unsigned int i, n;
  unsigned int type;
  type = msgRxBuff[20] << 8;
  type += msgRxBuff[21];
  if(type != 0x0001) {
    return;
  }
  if( memcmp(&msgRxBuff[38], myIP, 4) ) {
    return;
  }
  //do ARP respond
  //destMAC
  memmove(msgTxBuff, &msgRxBuff[6], 6);

  //srcMac
  memmove(&msgTxBuff[6], myMAC, 6);
  n = 12;
  //ethernet arp type
  msgTxBuff[n++] = 0x08;
  msgTxBuff[n++] = 0x06;
  //load hardware type (0x0001 = ethernet)
  msgTxBuff[n++] = 0x00;
  msgTxBuff[n++] = 0x01;

  //load protocol type (0x0800 = IP)
  msgTxBuff[n++] = 0x08;
  msgTxBuff[n++] = 0x00;

  //load hardware address length (6 bytes)
  msgTxBuff[n++] = 0x06;

  //load protocol address length (4 bytes)
  msgTxBuff[n++] = 0x04;

  //load opcode (ARP reply)
  msgTxBuff[n++] = 0x00;
  msgTxBuff[n++] = 0x02;

  //load my hardware address
  for(i = 0; i < 6; i++) {
    msgTxBuff[n++] = myMAC[i];
  }

  //load my ip address
  for(i = 0; i < 4; i++) {
    msgTxBuff[n++] = myIP[i];
  }

  //load dest hardware address
  for(i = 0; i < 6; i++) {
```

```
    msgTxBuff[n++] = msgRxBuff[22 + i];
  }

  //load dest ip address
  for(i = 0; i < 4; i++) {
    msgTxBuff[n++] = msgRxBuff[28 + i];
  }

  //dump data 18 bytes
  for(i = 0; i < 18; i++) {
    msgTxBuff[n++] = 0;
  }

  nic_transmit(n);
}

short IPcksum(unsigned short *buf, int nLength)
{
  unsigned long sum;
  sum = 0;
  while (nLength > 1) {
    sum += *buf++;
    nLength -= sizeof(short);
  }
  if(nLength) {
    sum += (*buf & 0xff00);
  }
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  return (~sum);
}

void doPing(void)                    // ICMP
{
  unsigned int i, n;
  char destIP[4];
  int nTotalLength;
  int dataLength;
  short checksum;
  static int id = 0x300;

  n = 26;
  for(i = 0; i < 4; i++)
    destIP[i] = msgRxBuff[n++];
  memmove(msgTxBuff, &msgRxBuff[6], 6);
  memmove(&msgTxBuff[6], myMAC, 6);

  n = 12;
  //type
  msgTxBuff[n++] = 0x08;
  msgTxBuff[n++] = 0x00;

  //--IP
  //version
  msgTxBuff[n] = msgRxBuff[n];
  n++;

  //header length(LSB 4 bits)
  msgTxBuff[n] = msgRxBuff[n];
  n++;

  //service type
  msgTxBuff[n] = msgRxBuff[n];
  n++;

  //Total length
  msgTxBuff[16] = msgRxBuff[16];
  msgTxBuff[17] = msgRxBuff[17];
```

```
nTotalLength = (msgRxBuff[16] << 8)  + msgRxBuff[17];
 dataLength = nTotalLength - 4;

 //Flagment ID
 msgTxBuff[18] = (id >> 8);
 msgTxBuff[19] = id;
 id++;
 dataLength -= 2;


 //Flags
 msgTxBuff[20] = msgRxBuff[20];
 msgTxBuff[21] = msgRxBuff[21];
 dataLength -= 2;

 //Time to live
 msgTxBuff[22] = 0x1f;
 dataLength--;

 //IP protocol type
 msgTxBuff[23] = 0x01;
 dataLength--;

//IP Header checksum
msgTxBuff[24] = 0;
msgTxBuff[25] = 0;
dataLength -= 2;

n = 26;
//My IP Address
for(i = 0; i < 4; i++)
  msgTxBuff[n++] = myIP[i];
dataLength -= 4;

//Dest IP Address
for(i = 0; i < 4; i++)
  msgTxBuff[n++] = destIP[i];
dataLength -= 4;

//-- ICMP
//Type
msgTxBuff[n++] = 0; //echo reply
dataLength --;

//Code
msgTxBuff[n++] = 0;
dataLength --;

//Data checksum
msgTxBuff[n++] = 0;
msgTxBuff[n++] = 0;
dataLength -= 2;

//Identifier
msgTxBuff[n] = msgRxBuff[n];
n++;
msgTxBuff[n] = msgRxBuff[n];
n++;
dataLength -= 2;

//Sequence Number
msgTxBuff[n] = msgRxBuff[n];
n++;
msgTxBuff[n] = msgRxBuff[n];
n++;
dataLength -= 2;
```

```
    for(i = 0; i < dataLength; i++) {
      msgTxBuff[n] = msgRxBuff[n];
      n++;
    }

    // IP Header length is 20, IPcksum uses shorts for count
  checksum = IPcksum((unsigned short *)&msgTxBuff[14], 20);
    msgTxBuff[24] = (char)(checksum >> 8);
    msgTxBuff[25] = (char)checksum;

    // Length is (nTotalLength - Header Length) divided by 2
    checksum = IPcksum((unsigned short *)&msgTxBuff[34], nTotalLength - 20);
    msgTxBuff[36] = (char)(checksum >> 8);
    msgTxBuff[37] = (char)checksum;

    nic_transmit(n);
}


void doIP(void)
{

  if(msgRxBuff[23] == 0x01) { // ICMP
    if(msgRxBuff[34] == 0x08) { // ICMP echo
      doPing();
    }
  }

          else if(msgRxBuff[23] == 0x11) {       // UDP
                    if((msgRxBuff[36] == 0x00)&&
(msgRxBuff[37] == 0x07)) { // DestPort = echo
                }
          else if((msgRxBuff[36] == 0x00)&&(msgRxBuff[37] == 0xa1))
                    doSNMP();
          }
          }
}
void testProcess(void)
{
  static int i = 0;
  int type;

#ifdef _DEBUG_PRINT_
  printf("testProcess()\n");
#endif // _DEBUG_PRINT_
  if(nic_receive()) {
    type = msgRxBuff[12] << 8;   //MSB
    type += msgRxBuff[13]; //LSB

#ifdef _DEBUG_PRINT_
  printf("type = %04x\n", type);
#endif // _DEBUG_PRINT_

    switch(type) {
      case FRAME_TYPE_ARP://ARP
        doARP();
        break;
      case FRAME_TYPE_IP://IP
        doIP();
        break;
    }
  }
  else {
    nic_sample_broadcast();
  }
}
```

```
#include <at89x52.h>
#include <stdio.h>
#include <string.h>
#include <absacc.h>
unsigned short UDP_TCPchksum(unsigned char *pBuf, short len)
{
  unsigned long sum;
  int i;
  sum = 0;
  for(i = 14; i < 22;) {     // add up the 8 IP address octects from the IP header, src and dest.
    sum += (unsigned int)(pBuf[i++] << 8);
    sum += pBuf[i++];
  }
  sum += pBuf[11];           // add in the protocol type, 0x11 for UDP, 0x06 for TCP
  sum += len;                // add the Length
  if(len & 0x01) {           // check for an odd length
    sum += (unsigned int)(pBuf[21 + len] << 8);
    len--;
  }
  len += 22;
  for(i = 22; i < len;) {
    sum += (unsigned int)(pBuf[i++] << 8);
    sum += pBuf[i++];
  }
  sum = (sum >> 16) + (sum & 0xffff);   // add in carry
  sum += (sum >> 16);  // maybe one more
  return(~sum);
}
unsigned short udpchksum(unsigned char *pBuf)
{
  short len;
```

```
    len = (short)(pBuf[26] << 8) + pBuf[27];
    return(UDP_TCPchksum(pBuf, len));
}
```

```
#include <at89x52.h>
#include <stdio.h>
#include <string.h>
#include "rtl8019.h"
#include "ip.h"
#include "udp.h"

extern idata unsigned char myMAC[6];
extern idata unsigned char myIP[4];

extern idata unsigned long tickSec;
extern idata unsigned char tick10msec;

extern xdata unsigned char msgRxBuff[1024];  //Rx frame buffer
extern xdata unsigned char msgTxBuff[1024];  //Tx frame buffer

unsigned char snmpObjValueType;
unsigned char snmpObjValueLength;
char snmpObjValue[256];

const char myCommunity[] = "public";
const char text1[] = "Beta1";
//MIB OID
const char mib_SysDescr[] = {1,3,6,1,2,1,1,1,0};
const char mib_UpTime[] = {1,3,6,1,2,1,1,3,0};
//const char mib_SysName[] = {1,3,6,1,2,1,1,5,0};
const char mib_Temp[] = {1,3,6,1,2,1,1,8,0};
const char mib_Humidity[] = {1,3,6,1,2,1,1,9,0};
char mib_exe[8];

unsigned int snmpMsgLength;
unsigned int snmpComLength;
char snmpCommunity[20];
char snmpEvent;
```

```
unsigned int reqPduLength;
unsigned int snmpReqID;
unsigned int snmpVarBindLength;
unsigned int snmpObjEntryLength;
unsigned char snmpOidLength;
char snmpOid[20];

unsigned int snmpVarBind;
unsigned int snmpVarBindList;
unsigned int Resp_PDU;
unsigned int snmpMsg;
unsigned int UDP_totalLength;
unsigned int IP_totalLength;

char dataSnmpBuff[20];
//////////// for ds1820_2.c ////////////
extern int TempData;
// ********* Humidity **********
extern unsigned char humidityValue;

// This function support Octet string
******************
void sendSnmp(char *buff, unsigned int size)
{
        char n,j,i;
        short checksum;
        int nTotalLength;
  //ethernet packget

// number of packet
snmpVarBind = (1+snmpOidLength)+1+(1+size);
snmpVarBindList = snmpVarBind + 4;
Resp_PDU = (4+6+4+snmpVarBindList);
snmpMsg = (3+2)+snmpComLength+(4+Resp_PDU);
UDP_totalLength = 8+4+snmpMsg;
IP_totalLength = 20+UDP_totalLength;
nTotalLength = 14+IP_totalLength;
memmove(msgTxBuff, &msgRxBuff[6], 6);
memmove(&msgTxBuff[6], myMAC, 6);
msgTxBuff[12] = 0x08; // ip
msgTxBuff[13] = 0x00;
msgTxBuff[14] = 0x45;
msgTxBuff[15] = 0x00;
msgTxBuff[16] = (char)(IP_totalLength>> 8);
msgTxBuff[17] = (char)IP_totalLength;
msgTxBuff[18] = snmpReqID>>8; // id = 28929
msgTxBuff[19] = snmpReqID;
msgTxBuff[20] = 0x00; // flags
msgTxBuff[21] = 0x00; // offset
msgTxBuff[22] = msgRxBuff[22];
msgTxBuff[23] = 0x11; // Protocol UDP
msgTxBuff[24] = 0; // Header checksum
msgTxBuff[25] = 0;
memmove(&msgTxBuff[26], myIP, 4
memmove(&msgTxBuff[30], &msgRxBuff[26], 4
msgTxBuff[36] = msgRxBuff[34]; // Dest Port
msgTxBuff[37] = msgRxBuff[35];
msgTxBuff[34] = 0x00; // Port 0x00a1 = snmp
msgTxBuff[35] = 0xa1;
msgTxBuff[38] = (char)(UDP_totalLength >> 8);
msgTxBuff[39] = ((char)UDP_totalLength);//nMsgLength + 8;
msgTxBuff[40] = 0; // UDP Checksum
msgTxBuff[41] = 0;

//************ snmp **********************
  n = 42;
  //msg length
  msgTxBuff[n++] = 0x30;
  msgTxBuff[n++] = 0x82; //default 16 bit
```

```
    msgTxBuff[n++] = snmpMsg>>8; //length
    msgTxBuff[n++] = snmpMsg;

    //version
  msgTxBuff[n++] = 0x02;
  msgTxBuff[n++] = 0x01;
  msgTxBuff[n++] = 0x00;
  msgTxBuff[n++] = 0x04;
  msgTxBuff[n++] = strlen(myCommunity);
  for(j=0; j<snmpComLength; j++, n++ ){
          msgTxBuff[n] = myCommunity[j];
  }
  msgTxBuff[n++] = snmpEvent;
  msgTxBuff[n++] = 0x82;
  msgTxBuff[n++] = Resp_PDU >> 8;
          // Response - PDU Length first byte
  msgTxBuff[n++] = Resp_PDU;
  //********** Read Request ID
  msgTxBuff[n++] = 0x02;
  msgTxBuff[n++] = 0x02;
  msgTxBuff[n++] = snmpReqID >> 8;
  msgTxBuff[n++] = snmpReqID;

  //skip status eeror  and index error, 6 bytes
  msgTxBuff[n++] = 0x02;
  msgTxBuff[n++] = 0x01;
  msgTxBuff[n++] = 0x00;
  msgTxBuff[n++] = 0x02;
  msgTxBuff[n++] = 0x01;
  msgTxBuff[n++] = 0x00;
  //Read Variable bindings list
  msgTxBuff[n++] = 0x30;
  msgTxBuff[n++] = 0x82;
  //varBindLength = 4
  msgTxBuff[n++] = snmpVarBindList>>8;
  msgTxBuff[n++] = snmpVarBindList ;
  msgTxBuff[n++] = 0x30;
  msgTxBuff[n++] = 0x82;
  msgTxBuff[n++] = snmpVarBind >> 8;
  msgTxBuff[n++] = snmpVarBind;
  // OID
  msgTxBuff[n++] = 0x06;
  msgTxBuff[n++] = 0x08;
  msgTxBuff[n++] = 0x2b;
  for(j=2; j<=8; j++,n++){
          msgTxBuff[n] = mib_exe[j];
  }
  msgTxBuff[n++] = snmpObjValueType;
  msgTxBuff[n++] = size;
  snmpObjEntryLength = size;
  for(i = 0; i < size; i++,n++)
          msgTxBuff[n] = buff[i];
  checksum = IPcksum((unsigned short *)&msgTxBuff[14], 20);
  msgTxBuff[24] = (char)(checksum >> 8);
  msgTxBuff[25] = (char)checksum;
  checksum = udpchksum(&msgTxBuff[12]);
  msgTxBuff[40] = (char)(checksum >> 8);
  msgTxBuff[41] = (char)checksum;
  nic_transmit(nTotalLength);
  }
  void snmpSysDescr(void)
  {
  switch(snmpEvent) {
          case 0xA0:
          snmpObjValueType = 0x04;
          memcpy(dataSnmpBuff,text1,sizeof(text1));
          break;
```

```
        }
snmpEvent = 0xA2;       // Get Resp
memmove(mib_exe, &mib_SysDescr[0], 9);
sendSnmp(dataSnmpBuff,strlen(dataSnmpBuff));
}

void snmpUpTime(void)
{
unsigned long *p;
switch(snmpEvent) {
        case 0xA0:
        snmpObjValueType = 0x43;
        memcpy(dataSnmpBuff,tickSec,4);
memset(dataSnmpBuff,0,sizeof(dataSnmpBuff));
memset(dataSnmpBuff,tickSec,);
p = (unsigned long *)dataSnmpBuff;
EA = 0;
*p = (tickSec*100)+tick10msec;
EA = 1;
break;
        }
snmpEvent = 0xA2;
memmove(mib_exe, &mib_UpTime[0], 9);
sendSnmp(dataSnmpBuff,4);
}
void snmpTemp(void)
{
        switch (snmpEvent) {
                case 0xA0 :
                snmpObjValueType = 0x04;
(oid)sprintf(dataSnmpBuff,"%c%d.%c C",((TempData & 0xff00)?'-':' ')
                        , (TempData >> 1)
                        , ((TempData & 1)?'5':'0'));
break;
case 0xA3 :
// return snmpObjValueType
snmpObjValueType = 0x04; // Octet String
memset(dataSnmpBuff,'0',sizeof(dataSnmpBuff));
(void)sprintf(dataSnmpBuff,"No Function");
break;
}
snmpEvent = 0xA2;
memmove(mib_exe, &mib_Temp[0], 9);
sendSnmp(dataSnmpBuff, strlen(dataSnmpBuff));
}

void snmpHumidity(void)
{
        switch (snmpEvent) {
                case 0xA0 :
snmpObjValueType = 0x04; // Octet String
(void)sprintf(dataSnmpBuff,"%u %cRH",humidityValue,'%');
break;
case 0xA3 :
snmpObjValueType = 0x04; // Octet String
(void)sprintf(dataSnmpBuff,"No function");
break;
}
memmove(mib_exe, &mib_Humidity[0], 9);
snmpEvent = 0xA2;       // Get Resp
sendSnmp(dataSnmpBuff, strlen(dataSnmpBuff));
}
void doMib(void)
{
  if(snmpOidLength == sizeof(mib_SysDescr))
{
  if(!memcmp(snmpOid, mib_SysDescr, sizeof(mib_SysDescr))) {
        snmpSysDescr();
```

```
            return;
          }
        }
      if(snmpOidLength == sizeof(mib_UpTime)) {
          if(!memcmp(snmpOid, mib_UpTime, sizeof(mib_UpTime))) {
            snmpUpTime();
            return;
          }
        }
      if(snmpOidLength == sizeof(mib_Temp)) {
          if(!memcmp(snmpOid, mib_Temp, sizeof(mib_Temp))) {
            snmpTemp();
            return;
          }
      }
      if(snmpOidLength == sizeof(mib_Humidity)) {
          if(!memcmp(snmpOid, mib_Humidity, sizeof(mib_Humidity))) {
            snmpHumidity();
            return;
          }
      }
    }

char formatSnmp(void)
{
  unsigned int i,j;
i = 42;   // Begin of SNMP Packet
if(msgRxBuff[i] != 0x30)
return 1;
i++;
  // check SnmpMsg is 16 bit or 8 bit  and Read
message Length
          if(msgRxBuff[i]== 0x82){
            i++;
            snmpMsgLength = msgRxBuff[i];
            i++;
            snmpMsgLength <<= 8;
            snmpMsgLength = msgRxBuff[i];
            i++;
            }
            else {
            snmpMsgLength = msgRxBuff[i];
            i++;
            }
        // Read Version
            if(msgRxBuff[i] != 0x02)
                  return 2;
            i++;
            if(msgRxBuff[i] != 0x01)
                  return 3;
            i++;
            if(msgRxBuff[i] != 0x00)
                  return 4;
            i++;
        // Read community
            if(msgRxBuff[i] != 0x04)
                  return 5;
          i++;
          snmpComLength = msgRxBuff[i];
          i++;

for(j = 0; j < snmpComLength; j++,i++){
    snmpCommunity[j] = msgRxBuff[i];
}
snmpCommunity[j] = 0;
```

```
//Read SNMP event
snmpEvent = msgRxBuff[i];
i++;
//Read Req-PDU length
if(msgRxBuff[i]== 0x82){
        i++;
        reqPduLength = msgRxBuff[i];
        i++;
        reqPduLength <<= 8;
        reqPduLength = msgRxBuff[i];
        i++;
}
else {
        reqPduLength = msgRxBuff[i];
        i++;
}

//Read Request ID
if(msgRxBuff[i] != 0x02)
        return 6;
i++;
if(msgRxBuff[i] == 0x02){ // 2 Bytes
//              return 7;
        i++;
        snmpReqID = msgRxBuff[i];
        // kept for Response
        i++;
        snmpReqID <<= 8;
        snmpReqID += msgRxBuff[i];
}
if(msgRxBuff[i] == 0x01) { // 1 Byte
        i++;
        snmpReqID = msgRxBuff[i];
}
        i++;
//skip status eeror  and index error, 6 bytes
        i += 6;

        //Read Variable bindings list
        //Read varBindLength
        if(msgRxBuff[i] != 0x30)
                return 8;
        i++;
        if(msgRxBuff[i]== 0x82){
        i++;
        snmpVarBindLength = msgRxBuff[i];
        i++;
        snmpVarBindLength <<= 8;
        snmpVarBindLength = msgRxBuff[i];
        i++;
        }
        else {
        snmpVarBindLength = msgRxBuff[i];
        i++;
        }

        // Read OID
        if(msgRxBuff[i] != 0x30)
                return 9;
        i++;
        if(msgRxBuff[i]== 0x82){
        i++;
        snmpObjEntryLength = msgRxBuff[i];
        i++;
        snmpObjEntryLength <<= 8;
        snmpObjEntryLength = msgRxBuff[i];
        i++;
        }
```

```
        else {
        snmpObjEntryLength = msgRxBuff[i];
        i++;
        }
        if(msgRxBuff[i] != 0x06)
        return 10;
        i++;
        snmpOidLength = msgRxBuff[i];

  //format first oid byte 0x2b --> 1,3
        i++;
  if(msgRxBuff[i] != 0x2b) {
    return 11; // oid must start with 0x2b (.1.3...)
  }
  snmpOid[0] = 1;
  snmpOid[1] = 3;
  snmpOidLength++; //we use 1,3 instant 0x2b
  i--;
        for(j = 2; j < snmpOidLength; j++,i++) {
                snmpOid[j] = msgRxBuff[i];

        }

        //value type
  snmpObjValueType = msgRxBuff[i++];

  //value length
  snmpObjValueLength = msgRxBuff[i++];

  //value
  for(j = 0; j < snmpObjValueLength; j++,i++) {
    snmpObjValue[j] = msgRxBuff[i];
  }
    return 0; //format finish
  }


  void doSnmp(void)
  {
    unsigned char i;

    //printf("doSnmp()\n");

    if(i = formatSnmp()) {
      printf("SNMP format Error <%bu>\n", i);
      return;
    }

    for(i = 0; i < snmpOidLength; i++)
  //    printf("%c%bd", '.', snmpOid[i]);
    //  putchar('\n');
    doMib();

  return;
  }
```

Code Visual Basic โปรแกรม SNMPBrowser

```
Begin VB.Form Form1
   Caption         =   "SNMPBrowser"
   ClientHeight    =   5280
   ClientLeft      =   165
   ClientTop       =   450
   ClientWidth     =   6705
   Icon            =   "SnmpAgent.frx":0000
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   ScaleHeight     =   5280
   ScaleWidth      =   6705
   Begin VB.Timer Timer2
      Enabled         =   0   'False
      Interval        =   60000
      Left            =   1680
      Top             =   0
   End
   Begin MSComDlg.CommonDialog cmnDlg
      Left            =   4560
      Top             =   -240
      _ExtentX        =   847
      _ExtentY        =   847
      _Version        =   393216
   End
   Begin VB.CommandButton cmdStep
      Caption         =   "Step"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   222
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   375
      Left            =   2160
      TabIndex        =   2
      Top             =   3960
      Width           =   855
   End
   Begin VB.Timer timeSave
      Enabled         =   0   'False
      Interval        =   3000
      Left            =   960
      Top             =   4680
   End
   Begin VB.CommandButton cmdRun
      Caption         =   "Auto"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   222
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   375
      Left            =   1200
      TabIndex        =   1
      Top             =   3960
      Width           =   855
   End
   Begin VB.Frame FrmData
      Caption         =   "SNMP Data"
      Height          =   1335
      Left            =   120
      TabIndex        =   19
      Top             =   840
      Width           =   6255
      Begin VB.TextBox txtUptime
         Alignment       =   2  'Center
         Height          =   315
         Left            =   4560
         TabIndex        =   10
```

```
      Top             =   720
      Width           =   1335
   End
   Begin VB.TextBox txtHumi
      Alignment       =   2  'Center
      Height          =   315
      Left            =   3120
      TabIndex        =   9
      Top             =   720
      Width           =   1215
   End
   Begin VB.TextBox txtTemp
      Alignment       =   2  'Center
      Height          =   315
      Left            =   1800
      TabIndex        =   8
      Top             =   720
      Width           =   1095
   End
   Begin VB.TextBox txtName
      Alignment       =   2  'Center
      Height          =   315
      Left            =   240
      TabIndex        =   7
      Top             =   720
      Width           =   1335
   End
   Begin VB.Label Label18
      AutoSize        =   -1  'True
      Caption         =   "Uptime"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   222
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   195
      Left            =   4920
      TabIndex        =   23
      Top             =   480
      Width           =   585
   End
   Begin VB.Label Label17
      AutoSize        =   -1  'True
      Caption         =   "Humidity"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   222
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   195
      Left            =   3360
      TabIndex        =   22
      Top             =   480
      Width           =   720
   End
   Begin VB.Label Label16
      AutoSize        =   -1  'True
      Caption         =   "Temp"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   222
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   195
      Left            =   2160
```

```
      TabIndex        =   21
      Top             =   480
      Width           =   465
   End
   Begin VB.Label Label5
      AutoSize        =   -1  'True
      Caption         =   "Version"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   222
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   195
      Left            =   600
      TabIndex        =   20
      Top             =   480
      Width           =   630
   End
End
Begin VB.Timer Timer1
   Interval        =   1000
   Left            =   5880
   Top             =   4560
End
Begin VB.ComboBox Combo1
   Height          =   330
   Left            =   3840
   TabIndex        =   6
   Top             =   4200
   Width           =   2655
End
Begin VB.TextBox Text4
   BeginProperty DataFormat
      Type            =   0
      Format          =   "0"
      HaveTrueFalseNull=   0
      FirstDayOfWeek  =   0
      FirstWeekOfYear =   0
      LCID            =   1054
      SubFormatType   =   0
   EndProperty
   Height          =   1215
   Left            =   120
   MultiLine       =   -1  'True
   ScrollBars      =   3   'Both
   TabIndex        =   11
   Top             =   2280
   Width           =   6375
End
Begin VB.TextBox Text1
   Height          =   285
   Left            =   120
   TabIndex        =   3
   Text            =   "161.246.13.90"
   Top             =   480
   Width           =   1935
End
Begin VB.CommandButton Command1
   Caption         =   "Get SNMP"
   Height          =   375
   Left            =   0
   TabIndex        =   0
   Top             =   3960
   Width           =   1095
End
Begin VB.TextBox Text3
   Height          =   285
   Left            =   1440
   TabIndex        =   5
   Text            =   "1.3.6.1.2.1.1.1.0"
   Top             =   480
   Width           =   1695
End
```

```
Begin VB.TextBox Text2
   Height          =   285
   Left            =   2400
   TabIndex        =   4
   Text            =   "public"
   Top             =   480
   Width           =   1335
End
Begin MSWinsockLib.Winsock pwSocket
   Left            =   6120
   Top             =   -120
   _ExtentX        =   741
   _ExtentY        =   741
   _Version        =   393216
   Protocol        =   1
   RemotePort      =   161
End
Begin VB.Label Label9
   BorderStyle     =   1  'Fixed Single
   Height          =   375
   Left            =   120
   TabIndex        =   12
   Top             =   3480
   Width           =   6375
End
Begin VB.Label LabelDate
   Alignment       =   2  'Center
   BorderStyle     =   1  'Fixed Single
   Height          =   270
   Left            =   3840
   TabIndex        =   18
   Top             =   4800
   Width           =   1215
End
Begin VB.Label LabelTime
   Alignment       =   2  'Center
   BorderStyle     =   1  'Fixed Single
   Height          =   270
   Left            =   5175
   TabIndex        =   17
   Top             =   4800
   Width           =   1320
End
Begin VB.Label Label4
   Caption         =   "Start Walking at MIB Value"
   Height          =   255
   Left            =   3840
   TabIndex        =   16
   Top             =   3960
   Width           =   1695
End
Begin VB.Label Label3
   Caption         =   "OID"
   Height          =   255
   Left            =   4440
   TabIndex        =   15
   Top             =   240
   Width           =   735
End
Begin VB.Label Label2
   Caption         =   "Community Name"
   Height          =   255
   Left            =   2400
   TabIndex        =   14
   Top             =   240
   Width           =   1575
End
Begin VB.Label Label1
   Caption         =   "Remote IP Address"
   Height          =   255
   Left            =   120
   TabIndex        =   13
   Top             =   240
   Width           =   1935
End
Begin VB.Menu mnuFile
   Caption         =   "File"
```

```
      Begin VB.Menu mnuExit
         Caption         =   "Exit"
      End
   End
   Begin VB.Menu mnuSave
      Caption         =   "Save"
      Begin VB.Menu mnuStart
         Caption         =   "Start"
      End
      Begin VB.Menu mnuStop
         Caption         =   "Stop"
      End
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Public DataArrivalCount As Integer
Public DataReturned As String
Public Count1 As Integer


Dim FileName As String
Dim Count2 As Integer
Dim count3 As Integer


Public Sub snmpSend(destination As String, Community As String, Oid As String, ge
   Dim iINT As Integer
   Dim convertType(255) As Byte
   Dim convertString As String
   Dim xsnmp As snmpPacket
   Dim OIDArray As Variant
   Dim OIDsize As Integer

   'Text4.Text = ""

   xsnmp.UniSeq = &H30
   xsnmp.verSNMP.byteType = 2
   xsnmp.verSNMP.packetLen = 1
   ReDim xsnmp.verSNMP.packetData(0)
   xsnmp.verSNMP.packetData(0) = 0

   xsnmp.commSNMP.byteType = 4
   ReDim xsnmp.commSNMP.packetData(CByte(Len(Trim$(Community)) - 1))
   For iINT = 0 To UBound(xsnmp.commSNMP.packetData)
      xsnmp.commSNMP.packetData(iINT) = Asc(Mid(Trim$(Community), iINT + 1, 1))
   Next iINT
   xsnmp.commSNMP.packetLen = CByte(UBound(xsnmp.commSNMP.packetData) + 1)

   xsnmp.contextSNMP.byteType = getType

   xsnmp.requestSNMP.byteType = 2
   ReDim xsnmp.requestSNMP.packetData(0)
   xsnmp.requestSNMP.packetData(0) = 1
   xsnmp.requestSNMP.packetLen = 1

   xsnmp.errorSNMP.byteType = 2
   ReDim xsnmp.errorSNMP.packetData(0)
   xsnmp.errorSNMP.packetData(0) = 0
   xsnmp.errorSNMP.packetLen = 1

   xsnmp.indexSNMP.byteType = 2
   ReDim xsnmp.indexSNMP.packetData(0)
   xsnmp.indexSNMP.packetData(0) = 0
   xsnmp.indexSNMP.packetLen = 1

   xsnmp.struct1SNMP.byteType = &H30

   xsnmp.struct2SNMP.byteType = &H30

   xsnmp.objectSNMP.byteType = 6

   OIDArray = Split(Trim(Oid), ".", , vbBinaryCompare)
   OIDsize = UBound(OIDArray)
```

```
    If OIDArray(UBound(OIDArray)) = "" Then OIDsize = OIDsize - 1

    ReDim xsnmp.objectSNMP.packetData(OIDsize - 1)
    xsnmp.objectSNMP.packetData(0) = &H2B
    For i = 2 To OIDsize
    xsnmp.objectSNMP.packetData(i - 1) = OIDArray(i)
    Next i

    xsnmp.endSNMP.byte01 = 5
    xsnmp.endSNMP.byte02 = 0
    xsnmp.objectSNMP.packetLen = OIDsize
    xsnmp.struct2SNMP.packetLen = xsnmp.objectSNMP.packetLen + 4
    xsnmp.struct1SNMP.packetLen = xsnmp.struct2SNMP.packetLen + 2
    xsnmp.contextSNMP.packetLen = xsnmp.struct1SNMP.packetLen + 11

    Call convertBinArray(xsnmp)
    Debug.Print snmpBinary
    pwSocket.RemotePort = 161
    pwSocket.RemoteHost = Trim(destination)
    pwSocket.SendData snmpBinary
End Sub

Private Sub cmdRun_Click()
        Count1 = 0
        timeSave.Enabled = True

End Sub

Private Sub cmdStep_Click()
        timeSave.Enabled = False
End Sub

Private Sub Combo1_Click()
   Dim OIDStart As String
   Dim OIDSplit As Variant
   OIDStart = Combo1.Text
   OIDSplit = Split(OIDStart, "-", , vbBinaryCompare)
   Text3.Text = Trim(OIDSplit(1))
End Sub

Private Sub Command1_Click()
'If pwSocket.State = sckOpen Then
   'vbWSAStartup                ' Initialize Winsock
   snmpSend Text1.Text, Text2.Text, Text3.Text, pduGet
'End If
End Sub



Private Sub Form_Load()
   DataArrivalCount = 0
   Count1 = 0
   count3 = 1
   'Open "packetData.bin" For Binary Access Write As #1

   'Combo1.AddItem "MIB2 - 1.3.6.1.2.1."
   Combo1.AddItem "Name - 1.3.6.1.2.1.1.1.0"
   Combo1.AddItem "UpTime - 1.3.6.1.2.1.1.3.0"
   Combo1.AddItem "Temperature - 1.3.6.1.2.1.1.8.0"
   Combo1.AddItem "Humidity - 1.3.6.1.2.1.1.9.0"
   'Combo1.AddItem "Enterprise - 1.3.6.1.4.1"
   'Combo1.AddItem "SNMP Modules - 1.3.6.1.6.3"
    pwSocket.Close
    pwSocket.RemoteHost = "161.246.13.90"
    pwSocket.RemotePort = "1001"
    pwSocket.Connect

        ChDir App.Path 'Always set the working directory to the directory to the
                                    ' the application

End Sub


Private Sub mnuExit_Click()
    End
End Sub
```

```
Private Sub mnuStart_Click()
        On Error GoTo ErrHandler
            cmnDlg.ShowSave
            FileName = cmnDlg.FileName
            Open FileName For Output As #1
            Print #1, "   No.                    Date                              Time
            Print #1, "   "; Str(count3); "        "; LabelDate.Caption; "
            Close #1
            Timer2.Enabled = True
            Exit Sub
ErrHandler:
            Exit Sub
End Sub

Private Sub mnuStop_Click()
    Timer2.Enabled = False
    End Sub
Private Sub pwSocket_Close()
   MsgBox "Closed"
End Sub

Private Sub pwSocket_Connect()
   MsgBox "Connected"
End Sub

Private Sub pwSocket_ConnectionRequest(ByVal requestID As Long)
MsgBox "Connection Request"
End Sub

Private Sub pwSocket_DataArrival(ByVal bytesTotal As Long)
  Dim OIDData As String
  pwSocket.GetData OIDData, vbByte
  'Put #1, , OIDData
'  Text4.Text = OIDData
  'TestCvt (OIDData)

  'convertSnmp (OIDData)
  toSNMP (OIDData)
  DataArrivalCount = DataArrivalCount + 1
End Sub

'Private Sub TestCvt(x)
 'For i = 1 To Len(x)
  ' Debug.Print Asc(Mid(x, i, 1)) & vbTab & Mid(x, i, 1)
 'Next
'End Sub

Private Sub pwSocket_Error(ByVal Number As Integer, description As String, ByVal
   MsgBox "Socket Error", vbInformation, description & " " & Source
End Sub

Private Sub Timer1_Timer()
    'labelTime.Caption = Format(Timer1, "hh : mm")
    LabelTime.Caption = Format(Now, "hh:mm:ss AMPM")
    LabelDate.Caption = Format(Date, "d-mmmm-yyyy")
End Sub

Private Sub Timer2_Timer()
        Count2 = Count2 + 1
    If Count2 > 5 Then
    'cmnDlg.ShowSave
            count3 = count3 + 1
            FileName = cmnDlg.FileName
            Open FileName For Append As #1
            Print #1, "    "; Str(count3); "        "; LabelDate.Caption; "
            Close #1
            Count2 = 0
    End If
End Sub

Private Sub timeSave_Timer()
        Count1 = Count1 + 1
        If Count1 > 4 Then
                Count1 = 1
        End If
        If Count1 = 1 Then
```

```
                snmpSend Text1.Text, Text2.Text, "1.3.6.1.2.1.1.1.0", pduGet
        ElseIf Count1 = 2 Then
                snmpSend Text1.Text, Text2.Text, "1.3.6.1.2.1.1.8.0", pduGet
        ElseIf Count1 = 3 Then
                snmpSend Text1.Text, Text2.Text, "1.3.6.1.2.1.1.9.0", pduGet
        ElseIf Count1 = 4 Then
                snmpSend Text1.Text, Text2.Text, "1.3.6.1.2.1.1.3.0", pduGet
        End If
End Sub
```

Datasheet

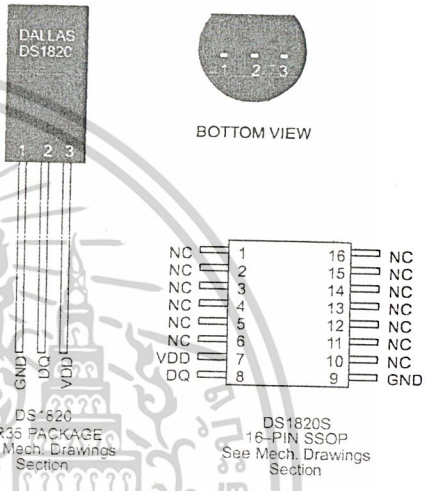
DALLAS SEMICONDUCTOR

DS1820
1–Wire™ Digital Thermometer

FEATURES

- Unique 1–Wire™ interface requires only one port pin for communication
- Multidrop capability simplifies distributed temperature sensing applications
- Requires no external components
- Can be powered from data line
- Zero standby power required
- Measures temperatures from –55°C to +125°C in 0.5°C increments. Fahrenheit equivalent is –67°F to +257°F in 0.9°F increments
- Temperature is read as a 9–bit digital value.
- Converts temperature to digital word in 200 ms (typ.)
- User–definable, nonvolatile temperature alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN ASSIGNMENT



DS1820
PR35 PACKAGE
See Mech. Drawings Section

DS1820S
16–PIN SSOP
See Mech. Drawings Section

PIN DESCRIPTION

GND	–	Ground
DQ	–	Data In/Out
V_{DD}	–	Optional V_{DD}
NC	–	No Connect

DESCRIPTION

The DS1820 Digital Thermometer provides 9–bit temperature readings which indicate the temperature of the device.

Information is sent to/from the DS1820 over a 1–Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS1820. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.

Because each DS1820 contains a unique silicon serial number, multiple DS1820s can exist on the same 1–Wire bus. This allows for placing temperature sensors in many different places. Applications where this feature is useful include HVAC environmental controls, sensing temperatures inside buildings, equipment or machinery, and in process monitoring and control.

DETAILED PIN DESCRIPTION

PIN 16–PIN SSOP	PIN PR35	SYMBOL	DESCRIPTION
9	1	GND	**Ground.**
8	2	DQ	**Data Input/Output pin.** For 1–Wire operation: Open drain. (See "Parasite Power" section.)
7	3	V_{DD}	**Optional V_{DD} pin.** See "Parasite Power" section for details of connection.

DS1820S (16–pin SSOP): All pins not specified in this table are not to be connected.

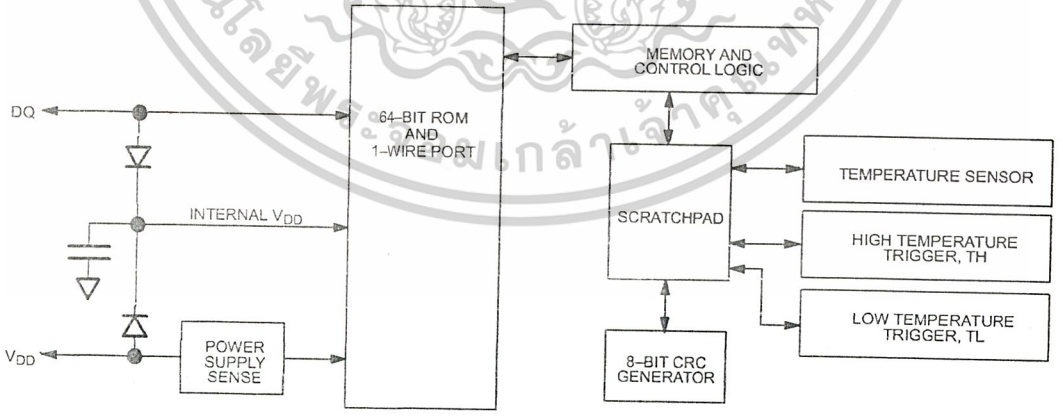
OVERVIEW

The block diagram of Figure 1 shows the major components of the DS1820. The DS1820 has three main data components: 1) 64–bit lasered ROM, 2) temperature sensor, and 3) nonvolatile temperature alarm triggers TH and TL. The device derives its power from the 1–Wire communication line by storing energy on an internal capacitor during periods of time when the signal line is high and continues to operate off this power source during the low times of the 1–Wire line until it returns high to replenish the parasite (capacitor) supply. As an alternative, the DS1820 may also be powered from an external 5 volts supply.

Communication to the DS1820 is via a 1–Wire port. With the 1–Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64–bit lasered ROM portion of each device and can single out a specific device if many are present on the 1–Wire line as well as indicate to the Bus Master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master may then provide any one of the six memory and control function commands.

One control function command instructs the DS1820 to perform a temperature measurement. The result of this measurement will be placed in the DS1820's scratchpad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of one byte EEPROM each. If the alarm search command is not applied to the DS1820, these registers may be used as general purpose user memory. Writing TH and TL is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first.

DS1820 BLOCK DIAGRAM Figure 1



PARASITE POWER

The block diagram (Figure 1) shows the parasite powered circuitry. This circuitry "steals" power whenever the I/O or V_{DD} pins are high. I/O will provide sufficient power as long as the specified timing and voltage requirements are met (see the section titled "1–Wire Bus System"). The advantages of parasite power are two–fold: 1) by parasiting off this pin, no local power source is needed for remote sensing of temperature, and 2) the ROM may be read in absence of normal power.

In order for the DS1820 to be able to perform accurate temperature conversions, sufficient power must be provided over the I/O line when a temperature conversion is taking place. Since the operating current of the DS1820 is up to 1 mA, the I/O line will not have sufficient drive due to the 5K pull–up resistor. This problem is particularly acute if several DS1820's are on the same I/O and attempting to convert simultaneously.

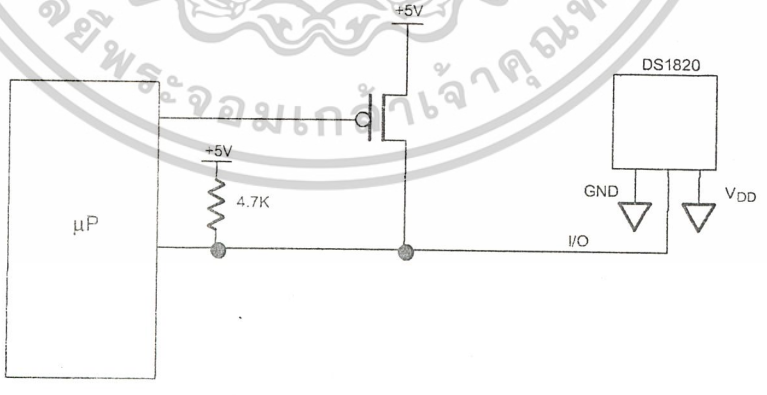
There are two ways to assure that the DS1820 has sufficient supply current during its active conversion cycle. The first is to provide a strong pull–up on the I/O line whenever temperature conversions or copies to the E^2 memory are taking place. This may be accomplished by using a MOSFET to pull the I/O line directly to the power supply as shown in Figure 2. The I/O line must be switched over to the strong pull–up within 10 µs maximum after issuing any protocol that involves copying to the E^2 memory or initiates temperature conversions. When using the parasite power mode, the V_{DD} pin must be tied to ground.

Another method of supplying current to the DS1820 is through the use of an external power supply tied to the V_{DD} pin, as shown in Figure 3. The advantage to this is that the strong pull–up is not required on the I/O line, and the bus master need not be tied up holding that line high during temperature conversions. This allows other data traffic on the 1–Wire bus during the conversion time. In addition, any number of DS1820's may be placed on the 1–Wire bus, and if they all use external power, they may all simultaneously perform temperature conversions by issuing the Skip ROM command and then issuing the Convert T command. Note that as long as the external power supply is active, the GND pin may not be floating.

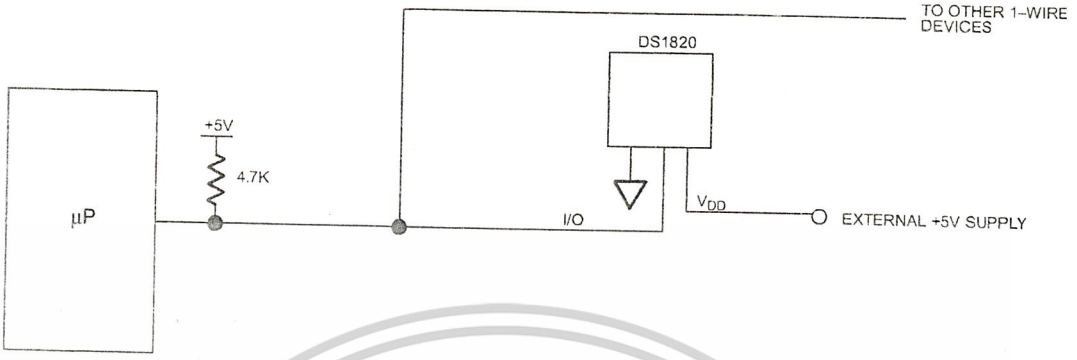
The use of parasite power is not recommended above 100°C, since it may not be able to sustain communications given the higher leakage currents the DS1820 exhibits at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that V_{DD} be applied to the DS1820.

For situations where the bus master does not know whether the DS1820's on the bus are parasite powered or supplied with external V_{DD}, a provision is made in the DS1820 to signal the power supply scheme used. The bus master can determine if any DS1820's are on the bus which require the strong pull–up by sending a Skip ROM protocol, then issuing the read power supply command. After this command is issued, the master then issues read time slots. The DS1820 will send back "0" on the 1–Wire bus if it is parasite powered; it will send back a "1" if it is powered from the V_{DD} pin. If the master receives a "0", it knows that it must supply the strong pull–up on the I/O line during temperature conversions. See "Memory Command Functions" section for more detail on this command protocol.

STRONG PULL–UP FOR SUPPLYING DS1820 DURING TEMPERATURE CONVERSION Figure 2



USING V_{DD} TO SUPPLY TEMPERATURE CONVERSION CURRENT Figure 3



OPERATION – MEASURING TEMPERATURE

The DS1820 measures temperature through the use of an on–board proprietary temperature measurement technique. A block diagram of the temperature measurement circuitry is shown in Figure 4.

The DS1820 measures temperature by counting the number of clock cycles that an oscillator with a low temperature coefficient goes through during a gate period determined by a high temperature coefficient oscillator. The counter is preset with a base count that corresponds to –55°C. If the counter reaches zero before the gate period is over, the temperature register, which is also preset to the –55°C value, is incremented, indicating that the temperature is higher than –55°C.

At the same time, the counter is then preset with a value determined by the slope accumulator circuitry. This circuitry is needed to compensate for the parabolic behavior of the oscillators over temperature. The counter is then clocked again until it reaches zero. If the gate period is still not finished, then this process repeats.

The slope accumulator is used to compensate for the non–linear behavior of the oscillators over temperature, yielding a high resolution temperature measurement. This is done by changing the number of counts necessary for the counter to go through for each incremental degree in temperature. To obtain the desired resolution, therefore, both the value of the counter and the number of counts per degree C (the value of the slope accumulator) at a given temperature must be known.

Internally, this calculation is done inside the DS1820 to provide 0.5°C resolution. The temperature reading is provided in a 16–bit, sign–extended two's complement reading. Table 1 describes the exact relationship of output data to measured temperature. The data is transmitted serially over the 1–Wire interface. The DS1820 can measure temperature over the range of –55°C to +125°C in 0.5°C increments. For Fahrenheit usage, a lookup table or conversion factor must be used.

Note that temperature is represented in the DS1820 in terms of a $^1/_2$°C LSB, yielding the following 9–bit format:

MSB								LSB
1	1	1	0	0	1	1	1	0

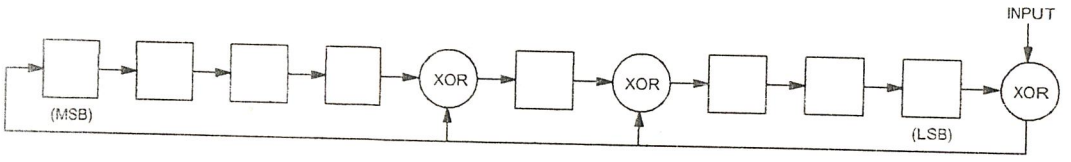
= –25°C

The most significant (sign) bit is duplicated into all of the bits in the upper MSB of the two–byte temperature register in memory. This "sign–extension" yields the 16–bit temperature readings as shown in Table 1.

Higher resolutions may be obtained by the following procedure. First, read the temperature, and truncate the 0.5°C bit (the LSB) from the read value. This value is TEMP_READ. The value left in the counter may then be read. This value is the count remaining (COUNT_REMAIN) after the gate period has ceased. The last value needed is the number of counts per degree C (COUNT_PER_C) at that temperature. The actual temperature may be then be calculated by the user using the following:

$$\text{TEMPERATURE} = \text{TEMP\_READ} - 0.25 + \frac{(\text{COUNT\_PER\_C} - \text{COUNT\_REMAIN})}{\text{COUNT\_PER\_C}}$$

1–WIRE CRC CODE Figure 7



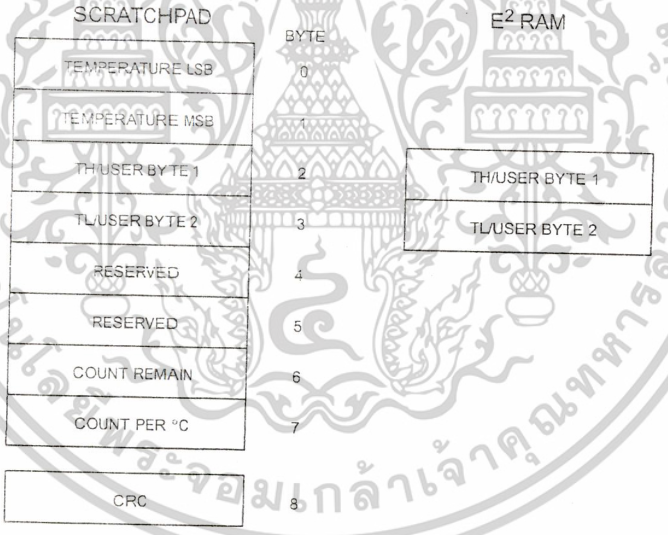
MEMORY

The DS1820's memory is organized as shown in Figure 8. The memory consists of a scratchpad RAM and a nonvolatile, electrically erasable (E^2) RAM, which stores the high and low temperature triggers TH and TL. The scratchpad helps insure data integrity when communicating over the 1–Wire bus. Data is first written to the scratchpad where it can be read back. After the data has been verified, a copy scratchpad command will transfer the data to the nonvolatile (E^2) RAM. This process insures data integrity when modifying the memory.

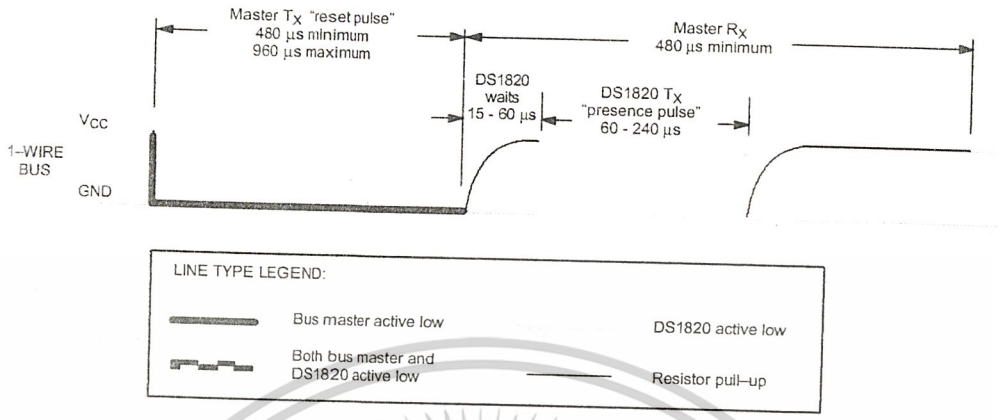
The scratchpad is organized as eight bytes of memory. The first two bytes contain the measured temperature information. The third and fourth bytes are volatile copies of TH and TL and are refreshed with every power–on reset. The next two bytes are not used; upon reading back, however, they will appear as all logic 1's. The seventh and eighth bytes are count registers, which may be used in obtaining higher temperature resolution (see "Operation–measuring Temperature" section).

There is a ninth byte which may be read with a Read Scratchpad command. This byte contains a cyclic redundancy check (CRC) byte which is the CRC over all of the eight previous bytes. This CRC is implemented in the fashion described in the section titled "CRC Generation".

DS1820 MEMORY MAP Figure 8



INITIALIZATION PROCEDURE "RESET AND PRESENCE PULSES" Figure 11



DS1820 COMMAND SET Table 2

INSTRUCTION	DESCRIPTION	PROTOCOL	1–WIRE BUS AFTER ISSUING PROTOCOL	NOTES
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	<read temperature busy status>	1
MEMORY COMMANDS				
Read Scratchpad	Reads bytes from scratchpad and reads CRC byte.	BEh	<read data up to 9 bytes>	
Write Scratchpad	Writes bytes into scratchpad at addresses 2 and 3 (TH and TL temperature triggers).	4Eh	<write data into 2 bytes at addr. 2 and addr. 3>	
Copy Scratchpad	Copies scratchpad into nonvolatile memory (addresses 2 and 3 only).	48h	<read copy status>	2
Recall E^2	Recalls values stored in nonvolatile memory into scratchpad (temperature triggers).	B8h	<read temperature busy status>	
Read Power Supply	Signals the mode of DS1820 power supply to the master.	B4h	<read supply status>	

NOTES:

1. Temperature conversion takes up to 500 ms. After receiving the Convert T protocol, if the part does not receive power from the V_{DD} pin, the I/O line for the DS1820 must be held high for at least 500 ms to provide power during the conversion process. As such, no other activity may take place on the 1–Wire bus for at least this period after a Convert T command has been issued.
2. After receiving the Copy Scratchpad protocol, if the part does not receive power from the V_{DD} pin, the I/O line for the DS1820 must be held high for at least 10 ms to provide power during the copy process. As such, no other activity may take place on the 1–Wire bus for at least this period after a Copy Scratchpad command has been issued.

Read Scratchpad [BEh]
This command reads the contents of the scratchpad. Reading will commence at byte 0, and will continue through the scratchpad until the 9th (byte–8, CRC) byte is read. If not all locations are to be read, the master may issue a reset to terminate reading at any time.

Copy Scratchpad [48h]
This command copies the scratchpad into the E^2 memory of the DS1820, storing the temperature trigger bytes in nonvolatile memory. If the bus master issues read time slots following this command, the DS1820 will output "0" on the bus as long as it is busy copying the scratchpad to E^2; it will return a "1" when the copy process is complete. If parasite powered, the bus master has to enable a strong pull–up for at least 10 ms immediately after issuing this command.

Convert T [44h]
This command begins a temperature conversion. No further data is required. The temperature conversion will be performed and then the DS1820 will remain idle. If the bus master issues read time slots following this command, the DS1820 will output "0" on the bus as long as it is busy making a temperature conversion; it will return a "1" when the temperature conversion is complete. If parasite powered, the bus master has to enable a strong pullup for 500 ms immediately after issuing this command.

Recall E2 [B8h]
This command recalls the temperature trigger values stored in E^2 to the scratchpad. This recall operation happens automatically upon power–up to the DS1820 as well, so valid data is available in the scratchpad as soon as the device has power applied. With every read data time slot issued after this command has been sent, the device will output its temperature converter busy flag "0"=busy, "1"=ready.

Read Power Supply [B4h]
With every read data time slot issued after this command has been sent to the DS1820, the device will signal its power mode: "0"=parasite power, "1"=external power supply provided.

READ/WRITE TIME SLOTS
DS1820 data is read and written through the use of time slots to manipulate bits and a command word to specify the transaction.

Write Time Slots
A write time slot is initiated when the host pulls the data line from a high logic level to a low logic level. There are two types of write time slots: Write One time slots and Write Zero time slots. All write time slots must be a minimum of 60 µs in duration with a minimum of a one µs recovery time between individual write cycles.

The DS1820 samples the I/O line in a window of 15 µs to 60 µs after the I/O line falls. If the line is high, a Write One occurs. If the line is low, a Write Zero occurs (see Figure 12).

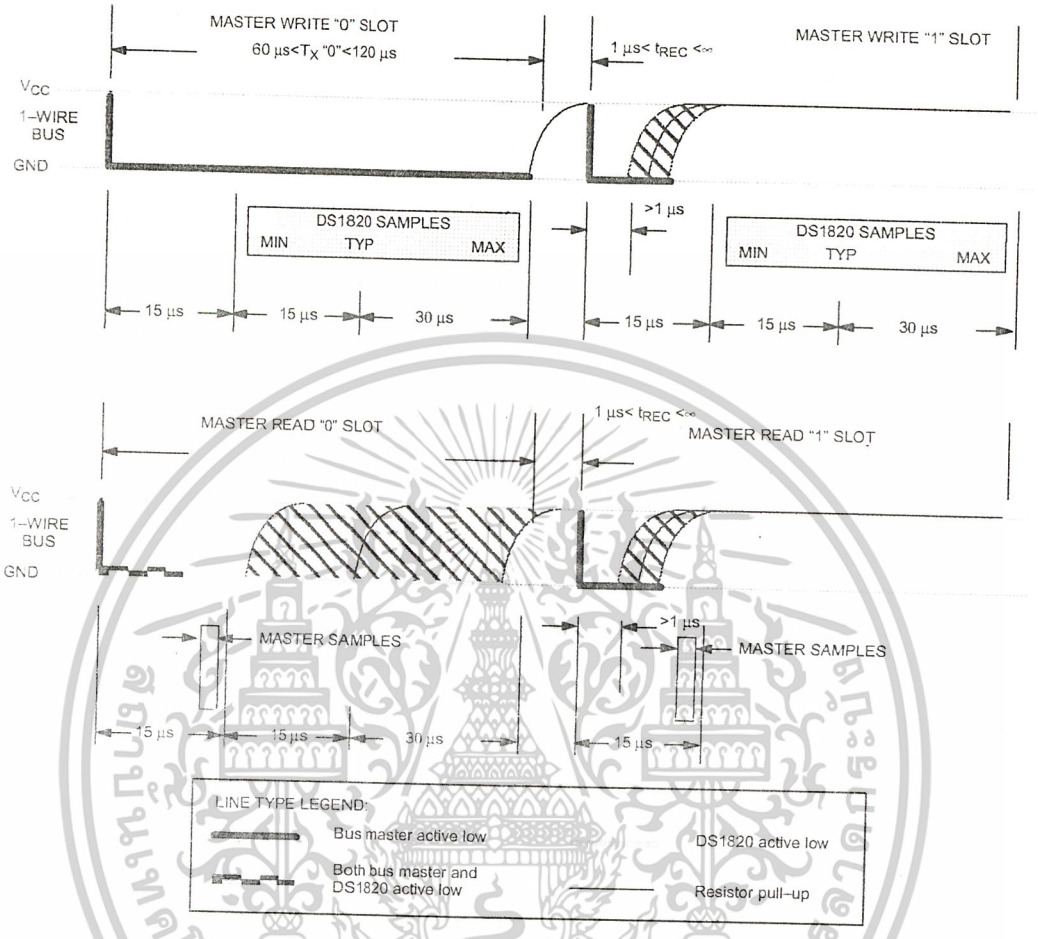
For the host to generate a Write One time slot, the data line must be pulled to a logic low level and then released, allowing the data line to pull up to a high level within 15 µs after the start of the write time slot.

For the host to generate a Write Zero time slot, the data line must be pulled to a logic low level and remain low for 60 µs.

Read Time Slots
The host generates read time slots when data is to be read from the DS1820. A read time slot is initiated when the host pulls the data line from a logic high level to logic low level. The data line must remain at a low logic level for a minimum of one µs; output data from the DS1820 is valid for 15 µs after the falling edge of the read time slot. The host therefore must stop driving the I/O pin low in order to read its state 15 µs from the start of the read slot (see Figure 12). By the end of the read time slot, the I/O pin will pull back high via the external pull–up resistor. All read time slots must be a minimum of 60 µs in duration with a minimum of a one µs recovery time between individual read slots.

Figure 13 shows that the sum of T_{INIT}, T_{RC}, and T_{SAMPLE} must be less than 15 µs. Figure 14 shows that system timing margin is maximized by keeping T_{INIT} and T_{RC} as small as possible and by locating the master sample time towards the end of the 15 µs period.

READ/WRITE TIMING DIAGRAM Figure 12



Humidity Sensors
Humidity Sensor

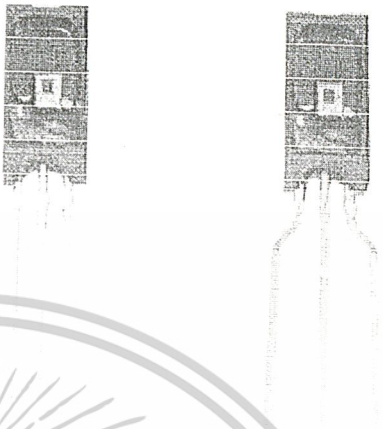
HIH-3610 Series

FEATURES

- Molded thermoset plastic housing with cover
- Linear voltage output vs %RH
- Laser trimmed interchangeability
- Low power design
- High accuracy
- Fast response time
- Stable, low drift performance
- Chemically resistant

TYPICAL APPLICATIONS

- Refrigeration
- Drying
- Metrology
- Battery-powered systems
- OEM assemblies



The HIH-3610 Series humidity sensor is designed specifically for high volume OEM (Original Equipment Manufacturer) users. Direct input to a controller or other device is made possible by this sensor's linear voltage output. With a typical current draw of only 200 µA, the HIH-3610 Series is ideally suited for low drain, battery operated systems. Tight sensor interchangeability reduces or eliminates OEM production calibration costs. Individual sensor calibration data is available.

The HIH-3610 Series delivers instrumentation-quality RH (Relative Humidity) sensing performance in a low cost, solderable SIP (Single In-line Package). Available in two lead spacing configurations, the RH sensor is a laser trimmed thermoset polymer capacitive sensing element with on-chip integrated signal conditioning. The sensing element's multilayer construction provides excellent resistance to application hazards such as wetting, dust, dirt, oils, and common environmental chemicals.

⚠WARNING
PERSONAL INJURY
- DO NOT USE these products as safety or emergency stop devices, or in any other application where failure of the product could result in personal injury.

Failure to comply with these instructions could result in death or serious injury.

⚠WARNING
MISUSE OF DOCUMENTATION
- The information presented in this product sheet is for reference only. Do not use this document as system installation information
- Complete installation, operation, and maintenance information is provided in the instructions supplied with each product.

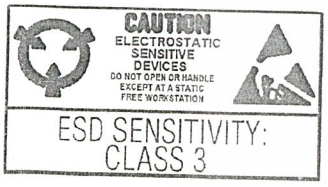
Failure to comply with these instructions could result in death or serious injury.

TABLE 1: PERFORMANCE SPECIFICATIONS

Parameter	Condition
RH Accuracy[1]	±2% RH, 0-100% RH non-condensing, 25 °C, V_{supply} = 5 Vdc
RH Interchangeability	±5% RH, 0-60% RH; ±8% @ 90% RH typical
RH Linearity	±0.5% RH typical
RH Hysteresis	±1.2% RH span maximum
RH Repeatability	±0.5% RH
RH Response Time, 1/e	15 sec in slowly moving air at 25 °C
RH Stability	±1% RH typical at 50% RH in 5 years
Power Requirements	
Voltage Supply	4 Vdc to 5.8 Vdc, sensor calibrated at 5 Vdc
Current Supply	200 µA at 5 Vdc
Voltage Output	$V_{out} = V_{supply}(0.0062(\text{Sensor RH}) + 0.16)$, typical @ 25 °C
	(Data printout option provides a similar, but sensor specific, equation at 25 °C.)
V_{supply} = 5 Vdc	0.8 Vdc to 3.9 Vdc output @ 25 °C typical
Drive Limits	Push/pull symmetric; 50 µA typical, 20 µA minimum, 100 µA maximum
	Turn-on ≤ 0.1 sec
Temperature Compensation	True RH = (Sensor RH)/(1.093-0.0021T), T in °F
	True RH = (Sensor RH)/(1.0546-0.00216T), T in °C
Effect @ 0% RH	±0.007 %RH/°C (negligible)
Effect @ 100% RH	-0.22% RH/°C (<1% RH effect typical in occupied space systems above 15 °C (59 °F))
Humidity Range	
Operating	0 to 100% RH, non-condensing[1]
Storage	0 to 90% RH, non-condensing
Temperature Range	
Operating	-40 °C to 85 °C (-40 °F to 185 °F)
Storage	-51 °C to 125 °C (-60 °F to 257 °F)
Package[2]	Three pin, solderable SIP in molded thermoset plastic housing with thermoplastic cover
Handling	Static sensitive diode protected to 15 kV maximum

Notes:

1. Extended exposure to ≥90% RH causes a reversible shift of 3% RH.
2. This sensor is light sensitive. For best results, shield the sensor from bright light.



FACTORY CALIBRATION

HIH-3610 sensors may be ordered with a calibration and data printout (Table 2). See order guide on back page.

TABLE 2: EXAMPLE DATA PRINTOUT

Model	HIH-3610-001
Channel	92
Wafer	030996M
MRP	337313
Calculated values at 5 V V_{out} @ 0% RH V_{out} @ 75.3% RH	 0.958 V 3.268 V
Linear output for 2% RH accuracy @ 25 °C Zero offset Slope RH	 0.958 V 30.680 mV/%RH (V_{out}-zero offset)/slope (V_{out}-0.958)/0.0307
Ratiometric response for 0 to 100% RH V_{out}	 V_{supply} (0.1915 to 0.8130)

FIGURE 1: RH SENSOR CONSTRUCTION

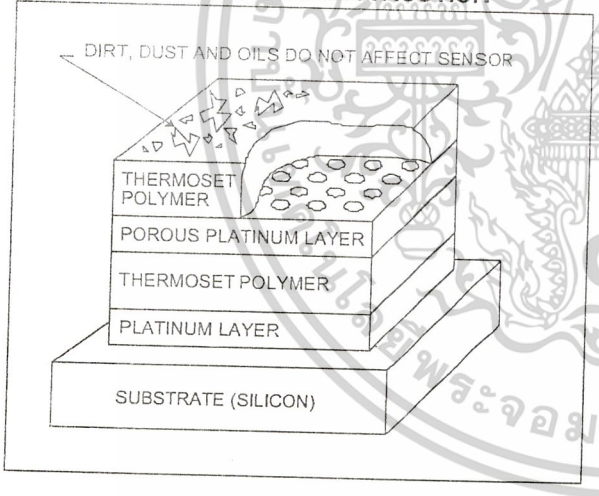


FIGURE 2: OUTPUT VOLTAGE VS RELATIVE HUMIDITY AT 0 °C

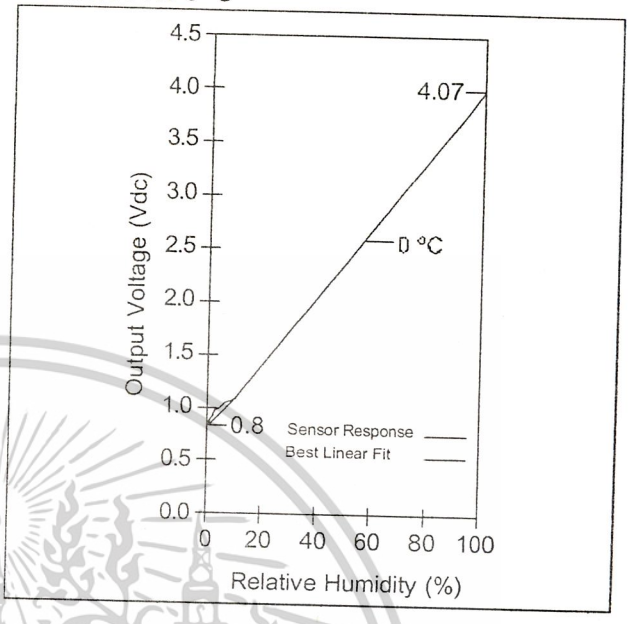
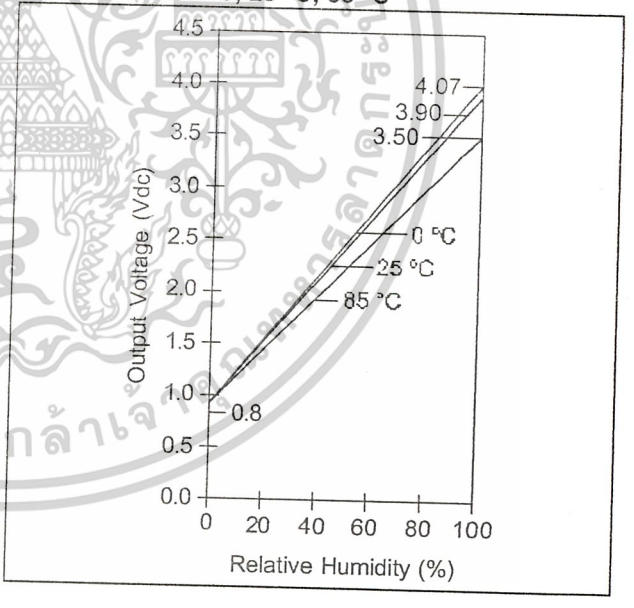


FIGURE 3: OUTPUT VOLTAGE VS RELATIVE HUMIDITY AT 0 °C, 25 °C, 85 °C



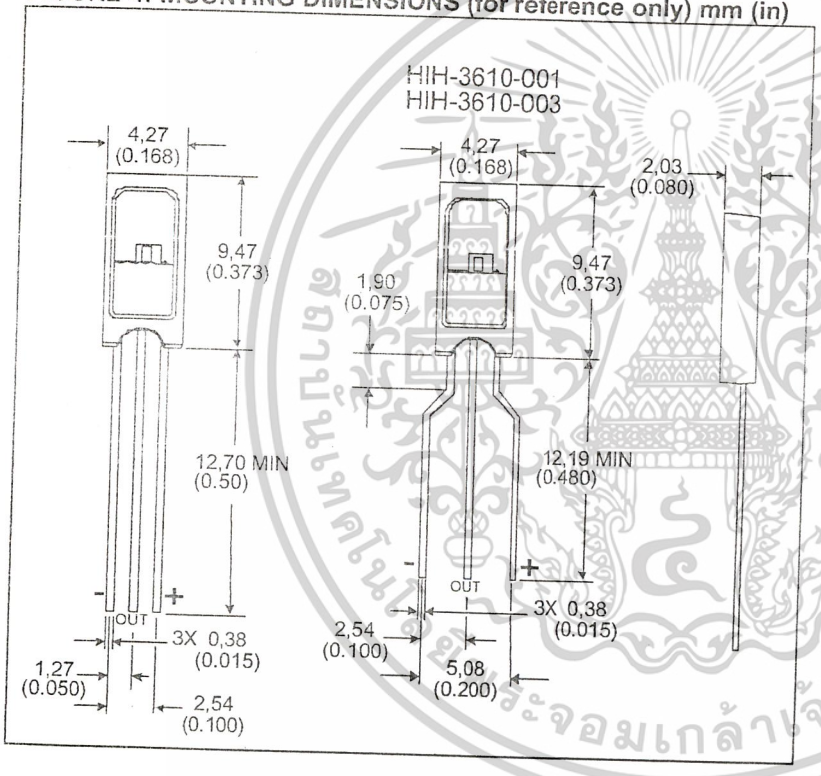
Humidity/Moisture Sensors
Humidity Sensor

HIH-3610 Series

ORDER GUIDE

Catalog Listing	Description
HIH-3610-001	Integrated circuit humidity sensor, 0.100 in lead pitch SIP
HIH-3610-002	Integrated circuit humidity sensor, 0.050 in lead pitch SIP
HIH-3610-003	Integrated circuit humidity sensor, 0.100 in lead pitch SIP with calibration and data printout
HIH-3610-004	Integrated circuit humidity sensor, 0.050 in lead pitch SIP with calibration and data printout

FIGURE 4: MOUNTING DIMENSIONS (for reference only) mm (in)



WARRANTY/REMEDY

Honeywell warrants goods of its manufacture as being free of defective materials and faulty workmanship. Contact your local sales office for warranty information. If warranted goods are returned to Honeywell during the period of coverage, Honeywell will repair or replace without charge those items it finds defective. The foregoing is Buyer's sole remedy and is **in lieu of all other warranties, expressed or implied, including those of merchantability and fitness for a particular purpose.**

Specifications may change without notice. The information we supply is believed to be accurate and reliable as of this printing. However, we assume no responsibility for its use.

While we provide application assistance personally, through our literature and the Honeywell web site, it is up to the customer to determine the suitability of the product in the application.

For application assistance, current specifications, or name of the nearest Authorized Distributor, check the Honeywell web site or call:

1-800-537-6945 USA
1-800-737-3360 Canada
1-815-235-6847 International
FAX
1-815-235-6545 USA
INTERNET
www.honeywell.com/sensing
info.sc@honeywell.com

Honeywell

Sensing and Control
Honeywell
11 West Spring Street
Freeport, Illinois 61032

Printed with Soy Ink on 50% Recycled Paper
00901-2-1-EN IL50 GLC 600 Printed in USA

www.honeywell.com/sensing

DATA SHEET

PCF8591
8-bit A/D and D/A converter

Preliminary specification
File under Integrated Circuits, IC01

September 1991

8-bit A/D and D/A converter

PCF8591

FEATURES

- Single power supply
- Operating supply voltage 2,5 V to 6 V
- Low standby current
- Serial input/output via I^2C bus
- Address by 3 hardware address pins
- Sampling rate given by I^2C bus speed
- 4 analogue inputs programmable as single-ended or differential inputs
- Auto-incremented channel selection
- Analogue voltage range from V_{SS} to V_{DD}
- On-chip track and hold circuit
- 8-bit successive approximation A/D conversion
- Multiplying DAC with one analogue output.

APPLICATIONS

Closed loop control systems; low power converter for remote data acquisition; battery operated equipment; acquisition of analogue values in automotive, audio and TV applications.

PACKAGE OUTLINES

PCF8591P:16-lead DIL; plastic (SOT38); SOT38-1; 1996 August 28.

PCF8591T:16-lead mini-pack; plastic (SO16L; SOT162A); SOT162-1; 1996 August 28.

GENERAL DESCRIPTION

The PCF8591 is a single chip, single supply low power 8-bit CMOS data acquisition device with four analogue inputs, one analogue output and a serial I^2C bus interface. Three address pins A0, A1 and A2 are used for programming the hardware address, allowing the use of up to eight devices connected to the I^2C bus without additional hardware. Address, control and data to and from the device are transferred serially via the two-line bidirectional bus (I^2C).

The functions of the device include analogue input multiplexing, on-chip track and hold function, 8-bit analogue-to-digital conversion and an 8-bit digital-to-analogue conversion. The maximum conversion rate is given by the maximum speed of the I^2C bus.

PINNING

1.	AIN0	analogue inputs (A/D converter)
2.	AIN1	
3.	AIN2	
4.	AIN3	
5.	A0	hardware address
6.	A1	
7.	A2	
8.	V_{SS}	negative supply voltage
9.	SDA	I^2C bus data input/output
10.	SCL	I^2C bus clock input/output
11.	OSC	oscillator input/output
12.	EXT	external/internal switch for oscillator input
13.	AGND	analogue ground
14.	V_{REF}	voltage reference input
15.	AOUT	analogue output (D/A converter)
16.	V_{DD}	positive supply voltage

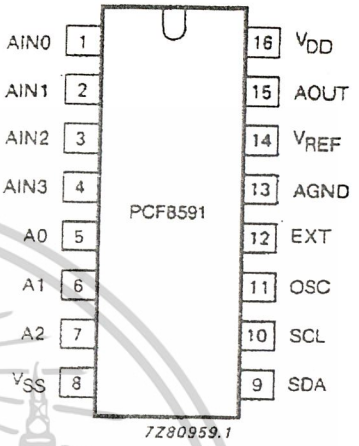


Fig.2 Pinning diagram.

FUNCTIONAL DESCRIPTION

Addressing

Each PCF8591 device in an I^2C bus system is activated by sending a valid address to the device. The address consists of a fixed part and a programmable part. The programmable part must be set according to the address pins A0, A1 and A2. The address always has to be sent as the first byte after the start condition in the I^2C bus protocol. The last bit of the address byte is the read/write-bit which sets the direction of the following data transfer (see Figs 3, 11 and 12).

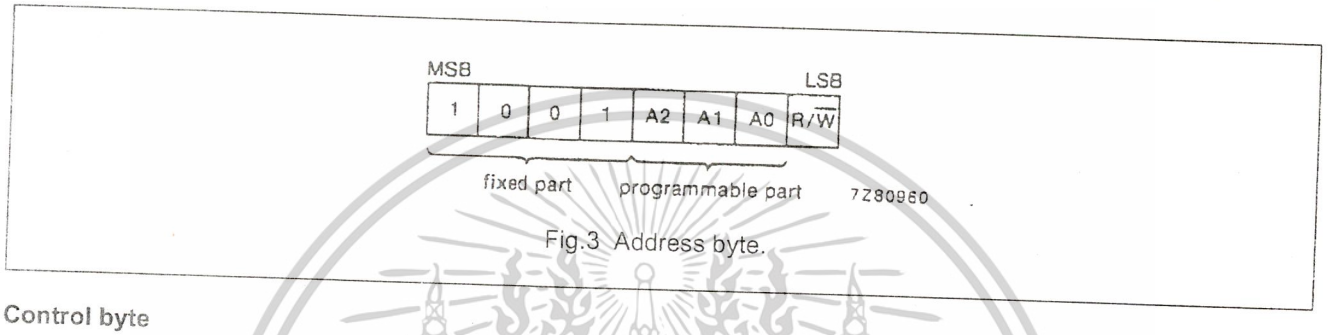


Fig.3 Address byte.

Control byte

The second byte sent to a PCF8591 device will be stored in its control register and is required to control the device function.

The upper nibble of the control register is used for enabling the analogue output, and for programming the analogue inputs as single-ended or differential inputs. The lower nibble selects one of the analogue input channels defined by the upper nibble (see Fig.4). If the auto-increment flag is set the channel number is incremented automatically after each A/D conversion.

If the auto-increment mode is desired in applications where the internal oscillator is used, the analogue output enable flag in the control byte (bit 6) should be set. This allows the internal oscillator to run continuously, thereby preventing conversion errors resulting from oscillator start-up delay. The analogue output enable flag may be reset at other times to reduce quiescent power consumption.

The selection of a non-existing input channel results in the highest available channel number being allocated. Therefore, if the auto-increment flag is set, the next selected channel will be always channel 0. The most significant bits of both nibbles are reserved for future functions and have to be set to 0. After a power-on reset condition all bits of the control register are reset to 0. The D/A converter and the oscillator are disabled for power saving. The analogue output is switched to a high impedance state.

เอกสารอ้างอิง

สุรศักดิ์ สงวนพงษ์. 2543. **สถาปัตยกรรมและโปรโตคอลทีซีพี/ไอพี.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น.

วิชิต ศิริโชติ. 2544. **การเขียนโปรแกรม MSC51 ด้วยภาษา C.** กรุงเทพฯ : แอนนาดิจิท กรุ๊ป จำกัด

สุทธิศักดิ์ พงศ์ธนาพานิช. 2542. **Visual BASIC 6.0 Professional,** กรุงเทพฯ : ซีเอ็ดยูเคชั่น

เน็ก, ดิลลิป ซี. 2524. **มาตรฐานอินเตอร์เน็ตและโปรโตคอล / โดย ดิลลิป ซี. เน็ก ; แปลและเรียบเรียงโดย เกรียงศักดิ์ พงษ์ชุมแพ ; บรรณาธิการ ชัยดำรงค์ อุทิรัมย์.** กรุงเทพ ฯ : สามย่าน.คอม

Stallings, W. 1993. **SNMP, SNMPv2, and RMON: Practical Network Management, 2nd Edition.** MA : Addison-Weslay Publishing Company

Rose, Marshall T. 1995. **How to management your network using SNMP : the networking management practicum.** Englewood Cliffs : PTR Prentice Hall

Miller, Mark A. 1993. **Managing internetworks with SNMP : the definitive guide to the Simple Network Management Protocol (SNMP) and SNMP version 2.** New York : M & T Books

Simoneau, Paul. 1999. **SNMP network management.** New York : McGraw-Hill

Knightson, Keith G. 1993. **OSI protocol conformance testing : IS 9646 explained.** New York : McGraw-Hill